



# OpenShift Container Platform 3.11

## クラスタの設定

OpenShift Container Platform 3.11 のインストールおよび設定



## OpenShift Container Platform 3.11 クラスターの設定

---

OpenShift Container Platform 3.11 のインストールおよび設定

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

## 法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Configuring\_Clusters.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

OpenShift のインストールと設定のトピックでは、ご使用の環境で OpenShift をインストールし、設定するための基本事項を説明します。扱われるトピックを参照して、OpenShift の稼働に必要な一度だけ実行するタスク (one-time task) を実行してください。

## 目次

第1章 概要 .....	21
第2章 レジストリーのセットアップ .....	22
2.1. 内部レジストリーの概要	22
2.1.1. レジストリーについて	22
2.1.2. 統合レジストリーまたはスタンドアロンレジストリー	22
2.2. 既存クラスターへのレジストリーのデプロイ	22
2.2.1. 概要	22
2.2.2. レジストリーホスト名の設定	22
2.2.3. レジストリーのデプロイ	23
2.2.4. レジストリーの DaemonSet としてのデプロイ	23
2.2.5. レジストリーのコンピュートリソース	23
2.2.6. レジストリーのストレージ	24
2.2.6.1. 実稼働環境での使用	24
2.2.6.1.1. Amazon S3 のストレージのバックエンドとしての使用	25
2.2.6.2. 非実稼働環境での使用	26
2.2.7. レジストリーコンソールの有効化	27
2.2.7.1. レジストリーコンソールのデプロイ	27
2.2.7.2. レジストリーコンソールのセキュリティー保護	27
2.2.7.3. レジストリーコンソールのトラブルシューティング	29
2.2.7.3.1. デバッグモード	29
2.2.7.3.2. SSL 証明書パスの表示	29
2.3. レジストリーへのアクセス	30
2.3.1. ログの表示	30
2.3.2. ファイルストレージ	30
2.3.3. レジストリーへの直接アクセス	32
2.3.3.1. ユーザーの前提条件	32
2.3.3.2. レジストリーへのログイン	33
2.3.3.3. イメージのプッシュとプル	33
2.3.4. レジストリーメトリクスへのアクセス	34
2.4. レジストリーのセキュリティー保護および公開	35
2.4.1. 概要	35
2.4.2. レジストリーを手動でセキュリティー保護する	36
2.4.3. セキュアなレジストリーの手動による公開	39
2.4.4. 非セキュアなレジストリーを手動で公開する	41
2.5. レジストリー設定の拡張	42
2.5.1. レジストリー IP アドレスの維持	42
2.5.2. 外部レジストリーの検索一覧の設定	43
2.5.3. レジストリーホスト名の設定	44
2.5.4. レジストリー設定の上書き	45
2.5.5. レジストリー設定の参照	47
2.5.5.1. Log	47
2.5.5.2. フック	47
2.5.5.3. ストレージ	47
2.5.5.4. 認証	48
2.5.5.5. ミドルウェア	48
2.5.5.5.1. S3 ドライバー設定	49
2.5.5.5.2. CloudFront ミドルウェア	50
2.5.5.5.3. ミドルウェア設定オプションの上書き	51
2.5.5.5.4. イメージのプルスルー	52
2.5.5.5.5. Manifest Schema v2 サポート	53

2.5.5.6. OpenShift	54
2.5.5.7. レポート	55
2.5.5.8. HTTP	55
2.5.5.9. 通知	55
2.5.5.10. Redis	55
2.5.5.11. Health	56
2.5.5.12. Proxy	56
2.5.5.13. Cache	56
2.6. 既知の問題	56
2.6.1. 概要	56
2.6.2. レジストリーのプルスルーに伴う同時ビルド	56
2.6.3. 共有 NFS ボリュームとスケールされたレジストリーの使用時のイメージのプッシュエラー	57
2.6.4. 内部で管理されたイメージのプルに失敗し見つかりません (not found) のエラーが表示される	58
2.6.5. S3 ストレージでのイメージのプッシュが失敗し 500 内部サーバーエラー (500 Internal Server Error) と表示される	58
2.6.6. イメージのプルニングの失敗	58
<b>第3章 ルーターのセットアップ</b> .....	<b>60</b>
3.1. ルーターの概要	60
3.1.1. ルーターについて	60
3.1.2. ルーターのサービスアカウント	60
3.1.2.1. ラベルにアクセスするためのパーミッション	60
3.2. デフォルト HAPROXY ルーターの使用	60
3.2.1. 概要	61
3.2.2. ルーターの作成	62
3.2.3. その他の基本ルーターコマンド	62
3.2.4. ルートを特定のルーターに絞り込む	64
3.2.5. HAProxy Strict SNI	65
3.2.6. TLS 暗号化スイート	65
3.2.7. 相互 TLS 認証	65
3.2.8. 高可用性ルーター	66
3.2.9. ルーターサービスポートのカスタマイズ	66
3.2.10. 複数ルーターの使用	67
3.2.11. デプロイメント設定へのノードセレクターの追加	67
3.2.12. ルーターシャードの使用	68
3.2.12.1. ルーターシャードの作成	70
3.2.12.2. ルーターシャードの変更	72
3.2.13. ルーターのホスト名の検索	73
3.2.14. デフォルトのルーティングサブドメインのカスタマイズ	74
3.2.15. カスタムルーティングサブドメインへのルートホスト名の強制	74
3.2.16. ワイルドカード証明書の使用	75
3.2.17. 証明書を手動で再デプロイする	75
3.2.18. セキュリティー保護されたルートの使用	76
3.2.19. (サブドメインの) ワイルドカードルートの使用	78
3.2.20. コンテナネットワークスタックの使用	83
3.2.21. Dynamic Configuration Manager の使用	83
3.2.22. ルーターメトリクスの公開	85
3.2.23. 大規模クラスターの ARP キャッシュのチューニング	87
3.2.24. DDoS 攻撃からの保護	88
3.2.25. HAProxy スレッドの有効化	89
3.3. カスタマイズされた HAPROXY ルーターのデプロイ	89
3.3.1. 概要	89
3.3.2. ルーター設定テンプレートの取得	90

3.3.3. ルーター設定テンプレートの変更	91
3.3.3.1. 背景情報	91
3.3.3.2. Go テンプレートアクション	91
3.3.3.3. ルーターが提供する情報	92
3.3.3.4. アノテーション	96
3.3.3.5. 環境変数	97
3.3.3.6. 使用例	97
3.3.4. ConfigMap を使用してルーター設定テンプレートを置き換える	99
3.3.5. Stick Table の使用	100
3.3.6. ルーターの再ビルド	101
3.4. PROXY プロトコルを使用するように HAPROXY ルーターを設定する	102
3.4.1. 概要	102
3.4.2. PROXY プロトコルを使用する理由	102
3.4.3. PROXY プロトコルの使用	103
<b>第4章 RED HAT CLOUDFORMS のデプロイ</b>	<b>107</b>
4.1. RED HAT CLOUDFORMS の OPENSIFT CONTAINER PLATFORM へのデプロイ	107
4.1.1. はじめに	107
4.2. RED HAT CLOUDFORMS を OPENSIFT CONTAINER PLATFORM で使用するための要件	108
4.3. ロール変数の設定	109
4.3.1. 概要	109
4.3.2. 一般的な変数	109
4.3.3. テンプレートパラメーターのカスタマイズ	110
4.3.4. データベース変数	110
4.3.4.1. コンテナ化された (Pod 化された) データベース	110
4.3.4.2. 外部データベース	110
4.3.5. ストレージクラス変数	111
4.3.5.1. NFS (デフォルト)	112
4.3.5.2. NFS (外部)	112
4.3.5.3. クラウドプロバイダー	113
4.3.5.4. 事前設定 (詳細)	113
4.4. インストーラーの実行	113
4.4.1. OpenShift Container Platform のインストール時またはインストール後の Red Hat CloudForms のデプロイ	113
4.4.2. インベントリーファイルの例	114
4.4.2.1. すべてのデフォルト	114
4.4.2.2. 外部 NFS ストレージ	114
4.4.2.3. PV サイズの上書き	115
4.4.2.4. メモリー要件の上書き	115
4.4.2.5. 外部 PostgreSQL データベース	115
4.5. コンテナプロバイダー統合の有効化	115
4.5.1. 単一コンテナプロバイダーの追加	115
4.5.1.1. 手動の追加	116
4.5.1.2. 自動の追加	116
4.5.2. 複数のコンテナプロバイダー	116
4.5.2.1. スクリプトの作成	116
4.5.2.1.1. 例	117
4.5.2.2. Playbook の実行	118
4.5.3. プロバイダーの更新	118
4.6. RED HAT CLOUDFORMS のアンインストール	118
4.6.1. アンインストール Playbook の実行	118
4.6.2. トラブルシューティング	119

<b>第5章 PROMETHEUS クラスターモニタリング</b> .....	<b>120</b>
5.1. 概要	120
5.2. OPENSIFT CONTAINER PLATFORM クラスターモニターリングの設定	121
5.2.1. モニターリングの前提条件	122
5.2.2. モニターリングスタックのインストール	122
5.2.3. 永続ストレージ	123
5.2.3.1. 永続ストレージの有効化	123
5.2.3.2. 必要なストレージサイズの判別	123
5.2.3.3. 永続ストレージサイズの設定	123
5.2.3.4. 十分な永続ボリュームの割り当て	123
5.2.3.5. 動的にプロビジョニングされたストレージの有効化	123
5.2.4. サポートされる設定	124
5.3. ALERTMANAGER の設定	124
5.3.1. Dead man's switch	125
5.3.2. アラートのグループ化	126
5.3.3. Dead man's switch PagerDuty	126
5.3.4. アラートルール	126
5.4. ETCD モニターリングの設定	133
5.5. PROMETHEUS、ALERTMANAGER、および GRAFANA へのアクセス	137
<b>第6章 RED HAT レジストリーへのアクセスおよびその設定</b> .....	<b>138</b>
6.1. 認証が有効にされている RED HAT レジストリー	138
6.1.1. ユーザーアカウントの作成	139
6.1.2. Red Hat レジストリー	139
6.1.3. インストールおよびアップグレード時のレジストリー認証情報の管理	139
6.1.4. Red Hat レジストリーでのサービスアカウントの使用	140
<b>第7章 マスターとノードの設定</b> .....	<b>143</b>
7.1. インストール後のマスターおよびノード設定のカスタマイズ	143
7.2. インストールの依存関係	143
7.3. マスターとノードの設定	143
7.4. ANSIBLE を使用した設定の変更	143
7.4.1. htpasswd コマンドの使用	145
7.5. 手動による設定変更	146
7.6. マスター設定ファイル	147
7.6.1. 受付制御の設定	147
7.6.2. アセットの設定	148
7.6.3. 認証と認可の設定	150
7.6.4. コントローラーの設定	150
7.6.5. etcd の設定	151
7.6.6. 付与の設定	152
7.6.7. イメージ設定	153
7.6.8. イメージポリシーの設定	153
7.6.9. Kubernetes のマスター設定	154
7.6.10. Network Configuration	155
7.6.11. OAuth 認証設定	156
7.6.12. プロジェクトの設定	158
7.6.13. スケジューラーの設定	159
7.6.14. セキュリティーアロケーターの設定	160
7.6.15. サービスアカウントの設定	160
7.6.16. 提供情報の設定	161
7.6.17. ボリュームの設定	162
7.6.18. 基本的な監査	163

7.6.18.1. 基本監査を有効にする	164
7.6.19. 高度な監査	165
7.6.20. etcd の TLS 暗号の指定	168
7.7. ノード設定ファイル	169
7.7.1. Pod とノードの設定	172
7.7.2. Docker の設定	172
7.7.3. ローカルストレージの設定	172
7.7.4. 1秒あたりのノードクエリー数 (QPS) の制限およびバースト値の設定	173
7.7.5. Docker 1.9 以降を使用したイメージの並行プル	173
7.8. パスワードおよびその他の機密データ	174
7.9. 新規設定ファイルの作成	175
7.10. 設定ファイルの使用によるサーバーの起動	175
7.11. マスターおよびノードログの表示	176
7.11.1. ロギングレベルの設定	177
7.12. マスターおよびノードサービスの再起動	182
<b>第8章 OPENSIFT ANSIBLE BROKER の設定</b> .....	<b>183</b>
8.1. 概要	183
8.2. RED HAT PARTNER CONNECT レジストリーでの認証	184
8.3. OPENSIFT ANSIBLE BROKER 設定の変更	184
8.4. レジストリー設定	184
8.4.1. 実稼働または開発	186
8.4.2. レジストリー認証情報の保存	186
8.4.3. APB のフィルターリング	188
8.4.4. モックレジストリー	189
8.4.5. Dockerhub レジストリー	189
8.4.6. Ansible Galaxy レジストリー	190
8.4.7. ローカルの OpenShift Container レジストリー	190
8.4.8. Red Hat Container Catalog レジストリー	190
8.4.9. Red Hat Partner Connect レジストリー	191
8.4.10. Helm チャートレジストリー	191
8.4.11. API V2 Docker レジストリー	191
8.4.12. Quay Docker レジストリー	192
8.4.13. 複数のレジストリー	192
8.5. ブローカー認証	192
8.5.1. Basic 認証	193
8.5.1.1. デプロイメントテンプレートおよびシークレット	193
8.5.1.2. サービスカタログおよびブローカー通信の設定	194
8.5.2. Bearer 認証	195
8.5.2.1. デプロイメントテンプレートおよびシークレット	195
8.5.2.2. サービスカタログおよびブローカー通信の設定	196
8.6. DAO 設定	196
8.7. ログ設定	196
8.8. OPENSIFT 設定	197
8.9. ブローカー設定	197
8.10. シークレット設定	198
8.11. プロキシ環境での実行	199
8.11.1. レジストリーアダプターのホワイトリスト	199
8.11.2. Ansible を使用したプロキシ環境でのブローカーの設定	199
8.11.3. プロキシ環境でのブローカーの手動設定	200
8.11.4. Pod でのプロキシ環境変数の設定	200
<b>第9章 ホストの既存クラスターへの追加</b> .....	<b>202</b>

9.1. ホストの追加	202
手順	202
9.2. ETCD ホストの既存クラスターへの追加	204
9.3. 共存する ETCD での既存のマスターの置き換え	205
9.4. ノードの移行	207
<b>第10章 デフォルトのイメージストリームとテンプレートの追加</b>	<b>209</b>
10.1. 概要	209
10.2. サブスクリプションタイプ別のサービス	209
10.2.1. OpenShift Container Platform サブスクリプション	209
10.2.2. xPaaS ミドルウェアアドオンサブスクリプション	210
10.3. 操作を始める前に	210
10.4. 前提条件	210
10.5. OPENSIFT CONTAINER PLATFORM イメージのイメージストリームの作成	211
10.6. XPAAS ミドルウェアイメージのイメージストリームの作成	212
10.7. データベースサービステンプレートの作成	212
10.8. インスタントアプリケーションおよびクイックスタートテンプレートの作成	212
10.9. 次のステップ	213
<b>第11章 カスタム証明書の設定</b>	<b>215</b>
11.1. 概要	215
11.2. 証明書チェーンの設定	215
11.3. インストール時のカスタム証明書の設定	215
11.4. WEB コンソールまたは CLI 用のカスタム証明書の設定	216
11.5. カスタムマスターホスト証明書の設定	217
11.6. デフォルトルーター用のカスタムワイルドカード証明書の設定	218
11.7. イメージレジストリー用のカスタム証明書の設定	219
11.8. ロードバランサー用のカスタム証明書の設定	220
11.9. カスタム証明書の変更およびクラスターへの組み込み	221
11.9.1. カスタムマスター証明書の変更およびクラスターへの組み込み	222
11.9.2. カスタムルーター証明書の変更およびクラスターへの組み込み	222
11.10. 他のコンポーネントでのカスタム証明書の使用	223
<b>第12章 証明書の再デプロイ</b>	<b>224</b>
12.1. 概要	224
12.2. 証明書の有効期限のチェック	224
12.2.1. ロール変数	224
12.2.2. 証明書の有効期限切れ Playbook の実行	225
他のサンプル Playbook	226
12.2.3. 出力形式	226
HTML レポート	226
JSON レポート	226
12.3. 証明書の再デプロイ	227
12.3.1. 現行の OpenShift Container Platform および etcd CA を使用したすべての証明書の再デプロイ	228
12.3.2. 新規またはカスタムの OpenShift Container Platform CA の再デプロイ	229
12.3.3. 新規 etcd CA の再デプロイ	230
12.3.4. マスターおよび Web コンソール証明書の再デプロイ	230
12.3.5. 名前付き証明書のための再デプロイ	231
12.3.6. etcd 証明書のための再デプロイ	231
12.3.7. ノード証明書の再デプロイ	232
12.3.8. レジストリー証明書またはルーター証明書のための再デプロイ	232
12.3.8.1. レジストリー証明書のための再デプロイ	232
12.3.8.2. ルーター証明書のための再デプロイ	232
12.3.9. カスタムのレジストリー証明書またはルーター証明書の再デプロイ	233

12.3.9.1. 手動によるレジストリー証明書の再デプロイ	233
12.3.9.2. 手動によるルーター証明書の再デプロイ	234
12.4. 証明書署名要求の管理	236
12.4.1. 証明書署名要求の確認	236
12.4.2. 証明書署名要求の承認	237
12.4.3. 証明書署名要求の拒否	237
12.4.4. 証明書署名要求の自動承認の設定	237
<b>第13章 認証およびユーザーエージェントの設定</b> .....	<b>238</b>
13.1. 概要	238
13.2. アイデンティティプロバイダーパラメーター	238
13.3. アイデンティティプロバイダーの設定	240
13.3.1. Ansible を使用したアイデンティティプロバイダーの設定	240
13.3.2. マスター設定ファイルでのアイデンティティプロバイダーの設定	241
13.3.2.1. lookup マッピング方法を使用する場合のユーザーの手動プロビジョニング	242
13.3.3. Allow All	242
13.3.4. Deny All	243
13.3.5. HTTPasswd	244
13.3.6. Keystone	245
13.3.6.1. マスターでの認証の設定	246
13.3.6.2. Keystone 認証を使用するユーザーの作成	248
13.3.6.3. ユーザーの確認	248
13.3.7. LDAP 認証	248
13.3.8. Basic 認証 (リモート)	251
13.3.8.1. マスターでの認証の設定	252
13.3.8.2. トラブルシューティング	254
13.3.9. 要求ヘッダー	255
Microsoft Windows での SSPI 接続サポート	258
要求ヘッダーを使用した Apache 認証	258
前提条件のインストール	259
Apache の設定	260
マスターの設定	262
サービスの再起動	262
設定の確認	262
13.3.10. GitHub および GitHub Enterprise	263
13.3.10.1. GitHub でのアプリケーションの登録	264
13.3.10.2. マスターでの認証の設定	264
13.3.10.3. GitHub 認証を持つユーザーの作成	267
13.3.10.4. ユーザーの確認	267
13.3.11. GitLab	267
13.3.12. Google	268
13.3.13. OpenID Connect	269
13.4. トークンオプション	272
13.5. 付与オプション	273
13.6. セッションオプション	274
13.7. ユーザーエージェントによる CLI バージョンの不一致の防止	275
<b>第14章 グループと LDAP の同期</b> .....	<b>278</b>
14.1. 概要	278
14.2. LDAP 同期の設定	278
14.2.1. LDAP クライアント設定	278
14.2.2. LDAP クエリー定義	279
14.2.3. ユーザー定義の名前マッピング	280

14.3. LDAP 同期の実行	280
14.4. グループのプルーニングジョブの実行	281
14.5. 同期の例	281
14.5.1. RFC 2307 スキーマの使用によるグループの同期	282
14.5.1.1. ユーザー定義の名前マッピングに関する RFC2307	284
14.5.2. ユーザー定義のエラートランスに関する RFC 2307 の使用によるグループの同期	285
14.5.3. Active Directory の使用によるグループの同期	288
14.5.4. 拡張された Active Directory の使用によるグループの同期	290
14.6. ネスト化されたメンバーシップ同期の例	292
14.7. LDAP 同期設定の仕様	296
14.7.1. v1.LDAPSyncConfig	296
14.7.2. v1.StringSource	298
14.7.3. v1.LDAPQuery	299
14.7.4. v1.RFC2307Config	300
14.7.5. v1.ActiveDirectoryConfig	302
14.7.6. v1.AugmentedActiveDirectoryConfig	302
<b>第15章 LDAP フェイルオーバーの設定</b>	<b>304</b>
15.1. 基本リモート認証設定の前提条件	304
15.2. 証明書の生成およびリモート BASIC 認証サーバーとの共有	304
15.3. SSSD での LDAP フェイルオーバーの設定	305
15.4. APACHE での SSSD の使用の設定	307
15.5. SSSD を基本リモート認証サーバーとして使用するよう OPENSIFT CONTAINER PLATFORM を設定する	310
<b>第16章 SDN の設定</b>	<b>312</b>
16.1. 概要	312
16.2. 利用可能な SDN プロバイダー	312
VMware NSX-T (™) の OpenShift Container Platform へのインストール	312
16.3. ANSIBLE を使用した POD ネットワークの設定	312
16.4. マスターでの POD ネットワークの設定	313
16.5. クラスターネットワークの VXLAN ポートの変更	315
16.6. ノードでの POD ネットワークの設定	316
16.7. サービスネットワークの拡張	316
16.8. SDN プラグイン間の移行	317
16.8.1. ovs-multitenant から ovs-networkpolicy への移行	318
16.9. クラスターネットワークへの外部アクセス	319
16.10. FLANNEL の使用	319
<b>第17章 NUAGE SDN の設定</b>	<b>322</b>
17.1. NUAGE SDN と OPENSIFT CONTAINER PLATFORM	322
17.2. 開発者のワークフロー	322
17.3. オペレーションワークフロー	322
17.4. インストールシステム	322
<b>第18章 NSX-T SDN の設定</b>	<b>325</b>
18.1. NSX-T SDN および OPENSIFT CONTAINER PLATFORM	325
18.2. トポロジーの例	325
18.3. VMWARE NSX-T のインストール	325
18.4. OPENSIFT CONTAINER PLATFORM デプロイ後の NSX-T の確認	330
<b>第19章 KURYR SDN の設定</b>	<b>333</b>
19.1. KURYR SDN および OPENSIFT CONTAINER PLATFORM	333
19.2. KURYR SDN のインストール	333

19.3. 検証	333
<b>第20章 AMAZON WEB サービス (AWS) の設定</b>	<b>335</b>
20.1. 概要	335
20.1.1. Amazon Web サービス (AWS) の認証の設定	335
20.1.1.1. インストール時の OpenShift Container Platform クラウドプロバイダーの設定	336
20.1.1.2. インストール後の OpenShift Container Platform クラウドプロバイダーの設定	337
20.2. セキュリティーグループの設定	337
20.2.1. 検出された IP アドレスとホスト名の上書き	338
20.2.1.1. Amazon Web Services (AWS) の OpenShift Container Platform レジストリーの設定	339
20.2.1.1.1. OpenShift Container Platform インベントリーを S3 を使用するように設定する	340
20.2.1.1.2. S3 を使用するための OpenShift Container Platform レジストリーの手動設定	341
20.2.1.1.3. レジストリーが S3 ストレージを使用することを確認します。	342
20.3. AWS 変数の設定	345
20.4. OPENSIFT CONTAINER PLATFORM での AWS の設定	345
20.4.1. Ansible を使用した OpenShift Container Platform での AWS の設定	345
20.4.2. OpenShift Container Platform マスターでの AWS の手動設定	346
20.4.3. OpenShift Container Platform ノードでの AWS の手動設定	346
20.4.4. キーと値のアクセスペアの手動設定	347
20.5. 設定変更の適用	347
20.6. クラスターに対する AWS のラベリング	348
20.6.1. タグを必要とするリソース	348
20.6.2. 既存クラスターへのタグ付け	348
20.6.3. Red Hat OpenShift Container Storage について	349
<b>第21章 RED HAT VIRTUALIZATION の設定</b>	<b>350</b>
21.1. BASTION 仮想マシンの作成	350
21.2. BASTION 仮想マシンを使用した OPENSIFT CONTAINER PLATFORM のインストール	353
<b>第22章 OPENSTACK の設定</b>	<b>359</b>
22.1. 概要	359
22.2. 作業開始前の準備	359
22.2.1. OpenShift Container Platform SDN	359
22.2.2. Kuryr SDN	359
22.2.3. OpenShift Container Platform の前提条件	360
22.2.3.1. Octavia の有効化: OpenStack の LBaaS (Load Balancing as a Service)	360
22.2.3.2. OpenStack ユーザーアカウント、プロジェクトおよびロールの作成	362
22.2.3.3. Kuryr SDN の追加手順	363
22.2.3.4. RC ファイルの設定	364
22.2.3.5. OpenStack フレーバーの作成	365
22.2.3.6. OpenStack キーペアの作成	366
22.2.3.7. OpenShift Container Platform の DNS の設定	367
22.2.3.8. OpenStack 経由での OpenShift Container Platform ネットワークの作成	368
22.2.3.9. OpenStack デプロイメントホストセキュリティーグループの作成	369
22.2.3.10. OpenStack Cinder ボリューム	370
22.2.3.10.1. Docker ボリューム	370
22.2.3.10.2. レジストリーボリューム	370
22.2.3.11. デプロイメントインスタンスの作成および設定	371
22.2.3.12. OpenShift Container Platform のデプロイメントホスト設定	372
22.3. OPENSIFT ANSIBLE PLAYBOOK を使用した OPENSIFT CONTAINER PLATFORM インスタンスのプロ ビジョニング	375
22.3.1. プロビジョニング用のインベントリーの準備	375
22.3.1.1. OpenShiftSDN の All YAML ファイル	376
22.3.1.2. KuryrSDN All YAML ファイル	377

22.3.1.2.1. グローバル namespace アクセスの設定	380
22.3.1.3. OSEv3 YAML ファイル	383
22.3.2. OpenStack 前提条件 Playbook	384
22.3.3. スタック名の設定	385
22.4. OPENSIFT CONTAINER PLATFORM インスタンスについての SUBSCRIPTION MANAGER の登録	386
22.5. ANSIBLE PLAYBOOK を使用した OPENSIFT CONTAINER PLATFORM のインストール	387
22.6. 設定変更を既存の OPENSIFT CONTAINER PLATFORM 環境に適用する	387
22.6.1. 既存の OpenShift 環境での OpenStack 変数の設定	388
22.6.2. 動的に作成した OpenStack PV のゾーンラベルの設定	388
<b>第23章 GOOGLE COMPUTE ENGINE の設定</b> .....	<b>390</b>
23.1. 作業を開始する前に	390
23.1.1. Google Cloud Platform の認証の設定	390
23.1.2. Google Compute Engine オブジェクト	391
23.2. OPENSIFT CONTAINER PLATFORM での GCE の設定	394
23.2.1. オプション 1: Ansible を使用した OpenShift Container Platform での GCP の設定	395
23.2.2. オプション 2: OpenShift Container Platform での GCE の手動設定	396
23.2.2.1. GCE 向けのマスターホストの手動設定	396
23.2.2.2. GCE 向けのノードホストの手動設定	397
23.2.3. GCP の OpenShift Container Platform レジストリーの設定	398
23.2.3.1. GCP 向けの OpenShift Container Platform レジストリーの手動設定	399
23.2.3.1.1. レジストリーが GCP オブジェクトストレージを使用していることを確認します。	400
23.2.4. OpenShift Container Platform が GCP ストレージを使用するように設定する	402
23.2.5. Red Hat OpenShift Container Storage について	403
23.3. サービスとしての GCP 外部のロードバランサー使用	403
<b>第24章 AZURE の設定</b> .....	<b>406</b>
24.1. 作業を開始する前に	406
24.1.1. Microsoft Azure の認証の設定	406
24.1.2. Microsoft Azure オブジェクトの設定	407
24.2. AZURE 設定ファイル	409
24.3. MICROSOFT AZURE 上の OPENSIFT CONTAINER PLATFORM のインベントリーサンプル	410
24.4. OPENSIFT CONTAINER PLATFORM での MICROSOFT AZURE の設定	412
24.4.1. Ansible を使用した OpenShift Container Platform での Azure の設定	412
24.4.2. OpenShift Container Platform での Microsoft Azure の手動設定	413
24.4.2.1. Microsoft Azure 向けのマスターホストの手動設定	413
24.4.2.2. Microsoft Azure 向けのノードホストの手動設定	414
24.4.3. Microsoft Azure の OpenShift Container Platform レジストリーの設定	415
24.4.4. OpenShift Container Platform を Microsoft Azure ストレージを使用するように設定する	420
24.4.5. Red Hat OpenShift Container Storage について	420
24.5. MICROSOFT AZURE 外部ロードバランサーのサービスとしての使用	421
24.5.1. ロードバランサーを使用したアプリケーションサンプルのデプロイ	421
<b>第25章 VMWARE VSPHERE の設定</b> .....	<b>423</b>
25.1. 作業を開始する前に	423
25.1.1. 要件	423
25.1.1.1. パーミッション	424
25.1.1.2. OpenShift Container Platform と vMotion の使用	426
25.2. OPENSIFT CONTAINER PLATFORM での VSPHERE の設定	426
25.2.1. オプション 1: Ansible を使用した OpenShift Container Platform での vSphere の設定	426
25.2.2. オプション 2: OpenShift Container Platform での vSphere の手動設定	430
25.2.2.1. vSphere 向けのマスターホストの手動設定	430
25.2.2.2. vSphere 向けのノードホストの手動設定	433
25.2.2.3. 設定変更の適用	434

25.3. OPENSIFT CONTAINER PLATFORM が VSPHERE ストレージを使用するように設定する	435
前提条件	435
25.3.1. VMware vSphere ボリュームの動的プロビジョニング	435
25.3.2. VMware vSphere ボリュームの静的プロビジョニング	436
25.3.2.1. PersistentVolume の作成	436
25.3.2.2. VMware vSphere ボリュームのフォーマット	437
25.4. VSPHERE の OPENSIFT CONTAINER PLATFORM レジストリーの設定	438
25.4.1. Ansible を使用した vSphere の OpenShift Container Platform レジストリーの設定	438
25.4.2. OpenShift Container Platform レジストリーの動的にプロビジョニングされるストレージ	438
25.4.3. OpenShift Container Platform レジストリーの手動でプロビジョニングされるストレージ	439
25.4.4. Red Hat OpenShift Container Storage について	439
25.5. 永続ボリュームのバックアップ	439
<b>第26章 ローカルボリュームの設定</b> .....	<b>441</b>
26.1. 概要	441
26.2. ローカルボリュームのマウント	441
26.3. ローカルプロビジョナーの設定	442
26.4. ローカルプロビジョナーのデプロイ	443
26.5. 新規デバイスの追加	444
26.6. RAW ブロックデバイスの設定	444
26.6.1. raw ブロックデバイスの準備	445
26.6.2. raw ブロックデバイスプロビジョナーのデプロイ	446
26.6.3. raw ブロックデバイスの永続ボリュームの使用	447
<b>第27章 永続ストレージの設定</b> .....	<b>449</b>
27.1. 概要	449
27.2. NFS を使用した永続ストレージ	449
27.2.1. 概要	449
27.2.2. プロビジョニング	450
27.2.3. ディスククォータの実施	451
27.2.4. NFS ボリュームのセキュリティー	451
27.2.4.1. グループ ID	452
27.2.4.2. ユーザー ID	453
27.2.4.3. SELinux	454
27.2.4.4. エクスポート設定	454
27.2.5. リソースの回収	455
27.2.6. 自動化	456
27.2.7. その他の設定とトラブルシューティング	456
27.3. RED HAT GLUSTER STORAGE を使用する永続ストレージ	457
27.3.1. 概要	457
27.3.1.1. コンバインドモード	457
27.3.1.2. インデペンデントモード	458
27.3.1.3. スタンドアロンの Red Hat Gluster Storage	458
27.3.1.4. GlusterFS ボリューム	459
27.3.1.5. gluster-block ボリューム	459
27.3.1.6. Gluster S3 Storage	460
27.3.2. 留意事項	460
27.3.2.1. ソフトウェア要件	460
27.3.2.2. ハードウェア要件	461
27.3.2.3. ストレージのサイジング	461
27.3.2.4. ボリューム操作の動作	462
27.3.2.5. ボリュームのセキュリティー	462
27.3.2.5.1. POSIX パーミッション	463

27.3.2.5.2. SELinux	463
27.3.3. サポート要件	464
27.3.4. インストールシステム	464
27.3.4.1. 独立モード: Red Hat Gluster Storage ノードのインストール	464
27.3.4.2. インストーラーの使用	465
27.3.4.2.1. ホスト変数	467
27.3.4.2.2. ロール変数	468
27.3.4.2.3. イメージ名とバージョンタグ変数	468
27.3.4.2.4. 例: 基本的なコンバインドモードのインストール	469
27.3.4.2.5. 例: 基本的なインデペンデントモードのインストール	471
27.3.4.2.6. 例: 統合 OpenShift Container レジストリーを使用する接続モード	472
27.3.4.2.7. 例: OpenShift ロギングおよびメトリクス用のコンバインドモード	473
27.3.4.2.8. 例: アプリケーション、レジストリー、ロギングおよびメトリクス用のコンバインドモード	475
27.3.4.2.9. 例: アプリケーション、レジストリー、ロギングおよびメトリクス用のインデペンデントモード	478
27.3.5. 接続モードのアンインストール	480
27.3.6. プロビジョニング	481
27.3.6.1. 静的プロビジョニング	481
27.3.6.2. 動的プロビジョニング	484
27.4. OPENSTACK CINDER を使用した永続ストレージ	485
27.4.1. 概要	485
27.4.2. Cinder PV のプロビジョニング	485
27.4.2.1. 永続ボリュームの作成	486
27.4.2.2. Cinder の PV 形式	487
27.4.2.3. Cinder ボリュームのセキュリティー	487
27.4.2.4. cinder ボリュームの制限	488
27.5. CEPH RADOS ブロックデバイス (RBD) を使用した永続ストレージ	488
27.5.1. 概要	488
27.5.2. プロビジョニング	489
27.5.2.1. Ceph シークレットの作成	489
27.5.2.2. 永続ボリュームの作成	490
27.5.3. Ceph ボリュームのセキュリティー	491
27.6. AWS ELASTIC BLOCK STORE を使用した永続ストレージ	492
27.6.1. 概要	492
27.6.2. プロビジョニング	493
27.6.2.1. 永続ボリュームの作成	493
27.6.2.2. ボリュームのフォーマット	494
27.6.2.3. ノード上の EBS ボリュームの最大数	494
27.7. GCE PERSISTENT DISK を使用した永続ストレージ	494
27.7.1. 概要	494
27.7.2. プロビジョニング	495
27.7.2.1. 永続ボリュームの作成	495
27.7.2.2. ボリュームのフォーマット	496
27.8. ISCSI を使用した永続ストレージ	496
27.8.1. 概要	496
27.8.2. プロビジョニング	496
27.8.2.1. ディスククォータの実施	497
27.8.2.2. iSCSI ボリュームのセキュリティー	497
27.8.2.3. iSCSI のマルチパス化	497
27.8.2.4. iSCSI のカスタムイニシエーター IQN	498
27.9. ファイバーチャネルを使用した永続ストレージ	498
27.9.1. 概要	498

27.9.2. プロビジョニング	499
27.9.2.1. ディスククォータの実施	500
27.9.2.2. ファイバーチャネルボリュームのセキュリティー	500
27.10. AZURE DISK を使用した永続ストレージ	500
27.10.1. 概要	500
27.10.2. 前提条件	500
27.10.3. プロビジョニング	500
27.10.4. Azure Disk での地域クラウドの設定	501
27.10.4.1. 永続ボリュームの作成	501
27.10.4.2. ボリュームのフォーマット	502
27.11. AZURE FILE を使用した永続ストレージ	503
27.11.1. 概要	503
27.11.2. 作業を開始する前に	503
27.11.3. 設定ファイルのサンプル	504
27.11.4. Azure File でのリージョンクラウドの設定	505
27.11.5. Azure Storage Account シークレットの作成	505
27.12. FLEXVOLUME プラグインを使用した永続ストレージ	506
27.12.1. 概要	506
27.12.2. FlexVolume ドライバー	507
27.12.2.1. マスター実行の割り当て/割り当て解除がある FlexVolume ドライバー	508
27.12.2.2. マスター実行の割り当て/割り当て解除がない FlexVolume ドライバー	510
27.12.3. FlexVolume ドライバーのインストール	511
27.12.4. FlexVolume ドライバーを使用したストレージの使用	511
27.13. 永続ストレージ用の VMWARE VSPHERE ボリューム	512
27.13.1. 概要	512
前提条件	513
27.13.2. VMware vSphere ボリュームの動的プロビジョニング	513
27.13.3. VMware vSphere ボリュームの静的プロビジョニング	514
27.13.3.1. VMDK の作成	514
27.13.3.2. PersistentVolume の作成	514
27.13.3.3. VMware vSphere ボリュームのフォーマット	515
27.14. ローカルボリュームを使用した永続ストレージ	516
27.14.1. 概要	516
27.14.2. プロビジョニング	516
27.14.3. ローカル永続ボリュームの作成	516
27.14.4. ローカルの Persistent Volume Claim (永続ボリューム要求) の作成	517
27.14.5. 機能のステータス	517
27.15. CONTAINER STORAGE INTERFACE (CSI) を使用した永続ストレージ	518
27.15.1. 概要	518
27.15.2. アーキテクチャー	518
27.15.2.1. 外部の CSI コントローラー	519
27.15.2.2. CSI ドライバーの DaemonSet	520
27.15.3. デプロイメント例	520
27.15.4. 動的プロビジョニング	525
27.15.5. 使用方法	525
27.16. OPENSTACK MANILA を使用した永続ストレージ	525
27.16.1. 概要	525
27.16.2. インストールと設定	526
27.16.2.1. 外部プロビジョナーの起動	526
27.16.3. 使用方法	529
27.17. 動的プロビジョニングとストレージクラスの作成	529
27.17.1. 概要	529
27.17.2. 動的にプロビジョニングされる使用可能なプラグイン	530

27.17.3. StorageClass の定義	531
27.17.3.1. 基本 StorageClass オブジェクト定義	531
27.17.3.2. StorageClass のアノテーション	532
27.17.3.3. OpenStack Cinder オブジェクトの定義	532
27.17.3.4. AWS ElasticBlockStore (EBS) オブジェクトの定義	533
27.17.3.5. GCE PersistentDisk (gcePD) オブジェクトの定義	533
27.17.3.6. GlusterFS オブジェクトの定義	534
27.17.3.7. Ceph RBD オブジェクトの定義	536
27.17.3.8. Trident オブジェクトの定義	536
27.17.3.9. VMWare vSphere オブジェクトの定義	537
27.17.3.10. Azure File のオブジェクト定義	537
27.17.3.11. Azure Disk オブジェクト定義	538
27.17.4. デフォルト StorageClass の変更	539
27.17.5. 追加情報とサンプル	540
27.18. ボリュームのセキュリティー	540
27.18.1. 概要	540
27.18.2. SCC、デフォルト値および許可される範囲	541
27.18.3. 補助グループ	544
27.18.4. fsGroup	548
27.18.5. ユーザー ID	550
27.18.6. SELinux オプション	552
27.19. セレクターとラベルによるボリュームのバインディング	553
27.19.1. 概要	553
27.19.2. 必要になる状況	553
27.19.3. Deployment	554
27.19.3.1. 前提条件	554
27.19.3.2. 永続ボリュームと要求の定義	554
27.19.3.3. オプション: PVC の特定の PV へのバインド	555
27.19.3.4. オプション: 特定の PVC に対する PV の確保	555
27.19.3.5. 永続ボリュームと要求のデプロイ	556
27.20. コントローラー管理の割り当ておよび割り当て解除	557
27.20.1. 概要	557
27.20.2. 割り当て/割り当て解除の管理元の判別	557
27.20.3. コントローラー管理の割り当て/割り当て解除を有効にするノードの設定	557
27.21. 永続ボリュームスナップショット	558
27.21.1. 概要	558
27.21.2. 機能	558
27.21.3. インストールと設定	558
27.21.3.1. 外部のコントローラーおよびプロビジョナーの起動	559
27.21.3.2. スナップショットユーザーの管理	561
27.21.4. ボリュームスナップショットとボリュームスナップショットデータのライフサイクル	562
27.21.4.1. Persistent Volume Claim (永続ボリューム要求) と永続ボリューム	562
27.21.4.1.1. スナップショットプロモーター	562
27.21.4.2. スナップショットの作成	562
27.21.4.3. スナップショットの復元	564
27.21.4.4. スナップショットの削除	564
27.22. HOSTPATH の使用	564
27.22.1. 概要	565
27.22.2. Pod 仕様での hostPath ボリュームの設定	565
27.22.3. hostPath ボリュームの静的なプロビジョニング	566
27.22.4. hostPath 共有の特権付き Pod でのマウント	567
27.22.5. 関連情報	568

第28章 永続ストレージの例 .....	569
28.1. 概要	569
28.2. 2つの PERSISTENT VOLUME CLAIM (永続ボリューム要求) 間での NFS マウントの共有	569
28.2.1. 概要	569
28.2.2. 永続ボリュームの作成	569
28.2.3. Persistent Volume Claim (永続ボリューム要求、PVC) の作成	570
28.2.4. NFS ボリュームアクセスの確保	571
28.2.5. Pod の作成	572
28.2.6. 同じ PVC を参照する追加 Pod の作成	576
28.3. CEPH RBD を使用した詳細例	578
28.3.1. 概要	578
28.3.2. ceph-common パッケージのインストール	578
28.3.3. Ceph シークレットの作成	578
28.3.4. 永続ボリュームの作成	579
28.3.5. Persistent Volume Claim (永続ボリューム要求、PVC) の作成	580
28.3.6. Pod の作成	581
28.3.7. グループ ID と所有者 ID の定義 (オプション)	582
28.3.8. ceph-user-secret をプロジェクトのデフォルトとして設定する方法	582
28.4. 動的プロビジョニングでの CEPH RBD の使用	583
28.4.1. 概要	583
28.4.2. 動的ボリューム用プールの作成	583
28.4.3. 動的な永続ストレージでの既存の Ceph クラスターの使用	584
28.4.4. ceph-user-secret をプロジェクトのデフォルトとして設定する方法	587
28.5. GLUSTERFS を使用する詳細例	588
28.5.1. 概要	588
28.5.2. 前提条件	588
28.5.3. 静的プロビジョニング	589
28.5.4. ストレージの使用	592
28.6. GLUSTERFS を動的プロビジョニングに使用する詳細例	594
28.6.1. 概要	594
28.6.2. 前提条件	594
28.6.3. 動的プロビジョニング	595
28.6.4. ストレージの使用	596
28.7. 特権付き POD へのボリュームのマウント	598
28.7.1. 概要	598
28.7.2. 前提条件	598
28.7.3. 永続ボリュームの作成	598
28.7.4. 通常ユーザーの作成	599
28.7.5. Persistent Volume Claim (永続ボリューム要求、PVC) の作成	599
28.7.6. 設定の検証	600
28.7.6.1. Pod の SCC の確認	600
28.7.6.2. マウントの検証	600
28.8. マウントの伝播	601
28.8.1. 概要	601
28.8.2. 値	601
28.8.3. 設定	601
28.9. 統合 OPENSIFT CONTAINER レジストリーから GLUSTERFS への切り替え	602
28.9.1. 概要	602
28.9.2. 前提条件	602
28.9.3. GlusterFS PersistentVolumeClaim の手動プロビジョニング	602
28.9.4. PersistentVolumeClaim のレジストリーへの割り当て	605
28.10. ラベルによる永続ボリュームのバインド	606
28.10.1. 概要	606

28.10.1.1. 想定条件	606
28.10.2. 仕様の定義	606
28.10.2.1. ラベルのある永続ボリューム	607
28.10.2.2. セクターのある Persistent Volume Claim (永続ボリューム要求)	607
28.10.2.3. ボリュームエンドポイント	608
28.10.2.4. PV、PVC、およびエンドポイントのデプロイ	608
28.11. ストレージクラスを使用した動的プロビジョニング	609
28.11.1. 概要	609
28.11.2. シナリオ 1: 2 種類の StorageClass を持つ基本的な動的プロビジョニング	609
28.11.3. シナリオ 2: クラスターにおけるデフォルトの StorageClass の動作を有効にする方法	611
28.12. 既存のレガシーストレージに対するストレージクラスの使用	616
28.12.1. 概要	616
28.12.1.1. シナリオ 1: レガシーデータを含む既存の永続ボリュームに StorageClass をリンクさせる	616
28.13. AZURE BLOB ストレージでの統合コンテナイメージレジストリーの設定	619
28.13.1. 概要	619
28.13.2. 操作を始める前に	619
28.13.3. レジストリー設定の上書き	619
<b>第29章 一時ストレージの設定</b>	<b>621</b>
29.1. 概要	621
29.2. 一時ストレージの有効化	621
<b>第30章 HTTP プロキシの使用</b>	<b>623</b>
30.1. 概要	623
30.2. NO_PROXY の設定	623
30.3. ホストでのプロキシの設定	624
30.4. ANSIBLE を使用したホストでのプロキシ設定	625
30.5. DOCKER PULL のプロキシ設定	625
30.6. プロキシの背後での MAVEN の使用	626
30.7. S2I ビルドでのプロキシの設定	626
30.8. デフォルトテンプレートでのプロキシの設定	627
30.9. POD でのプロキシ環境変数の設定	627
30.10. GIT リポジトリーのアクセス	627
<b>第31章 グローバルビルドのデフォルトと上書きの設定</b>	<b>629</b>
31.1. 概要	629
31.2. グローバルビルドのデフォルトの設定	630
31.2.1. Ansible を使用したグローバルビルドのデフォルトの設定	630
31.2.2. グローバルビルドのデフォルトの手動設定	631
31.3. グローバルビルドの上書きの設定	632
31.3.1. Ansible を使用したグローバルビルドの上書きの設定	632
31.3.2. グローバルビルドの上書きの手動設定	633
<b>第32章 パイプライン実行の設定</b>	<b>635</b>
32.1. 概要	635
32.2. OPENSIFT JENKINS CLIENT プラグイン	636
32.3. OPENSIFT JENKINS の同期プラグイン	636
<b>第33章 ルートのタイムアウトの設定</b>	<b>638</b>
<b>第34章 ネイティブのコンテナルーティングの設定</b>	<b>639</b>
34.1. ネットワークの概要	639
34.2. ネイティブのコンテナルーティングの設定	639
34.3. コンテナネットワーク用のノードのセットアップ	640
34.4. コンテナネットワーク用のルーターのセットアップ	640

<b>第35章 エッジロードバランサーからのルーティング</b> .....	<b>641</b>
35.1. 概要	641
35.2. ロードバランサーを SDN に含める	641
35.3. RAMP ノードを使用したトンネルの確立	641
35.3.1. 高可用性を備えた Ramp ノードの設定	644
<b>第36章 コンテナログの集計</b> .....	<b>645</b>
36.1. 概要	645
36.2. デプロイ前の設定	645
36.3. ロギング ANSIBLE 変数の指定	645
36.4. EFK スタックのデプロイ	657
36.5. デプロイメントの概要と調整	658
36.5.1. Ops クラスタ	658
36.5.2. Elasticsearch	659
36.5.2.1. 永続的な Elasticsearch ストレージ	660
36.5.2.1.1. 永続ボリュームとしての NFS の使用	662
36.5.2.1.2. NFS のローカルストレージとしての使用	663
36.5.2.1.3. Elasticsearch の hostPath ストレージの設定	665
36.5.2.1.4. Elasticsearch のスケールの変更	667
36.5.2.1.5. Elasticsearch レプリカ数の変更	667
36.5.2.1.6. Elasticsearch のルートとしての公開	667
36.5.3. Fluentd	668
36.5.4. Kibana	682
36.5.5. Curator	688
36.5.5.1. Curator Actions File の使用	689
36.5.5.2. Curator 設定の作成	691
36.6. CLEANUP	692
36.7. 外部 ELASTICSEARCH インスタンスへのログの送信	692
36.8. 外部 SYSLOG サーバーへのログの送信	692
36.9. ELASTICSEARCH 管理操作の実行	695
36.10. EFK 証明書の再デプロイ	696
36.11. 集計されたロギングのドライバーの変更	696
36.12. ELASTICSEARCH の手動ロールアウト	698
36.12.1. Elasticsearch ローリングクラスタ再起動の実行	698
36.12.2. Elasticsearch フルクラスタ再起動の実行	699
36.13. EFK のトラブルシューティング	700
36.13.1. すべての EFK コンポーネントに関連するトラブルシューティング	700
36.13.2. Elasticsearch に関連するトラブルシューティング	701
36.13.3. Kibana	702
<b>第37章 集計ロギングのサイジングに関するガイドライン</b> .....	<b>705</b>
37.1. 概要	705
37.2. インストールシステム	705
37.2.1. 大規模クラスタ	707
37.3. SYSTEMD-JOURNALD と RSYSLOG	707
37.4. EFK ロギングのスケールアップ	708
37.4.1. マスターイベントは、ログとして EFK に集計されます。	708
37.5. ストレージに関する考慮事項	708
<b>第38章 クラスタメトリクスの有効化</b> .....	<b>709</b>
38.1. 概要	709
38.2. 操作を始める前に	709
38.3. メトリクスデータストレージ	709
38.3.1. 永続ストレージ	709

38.3.2. クラスターメトリクスのキャパシティーピニング	710
既知の問題と制限	712
38.3.3. 非永続ストレージ	712
38.4. メトリクス ANSIBLE ロール	713
38.4.1. メトリクス Ansible 変数の指定	713
38.4.2. シークレットの使用	716
38.4.2.1. ユーザー固有の証明書の提供	717
38.5. メトリックコンポーネントのデプロイ	717
38.5.1. メトリクスの診断	718
38.6. メトリクスのパブリック URL の設定	718
38.7. HAWKULAR METRICS への直接アクセス	719
38.7.1. OpenShift Container Platform プロジェクトと Hawkular テナント	719
38.7.2. 承認	719
38.8. OPENSIFT CONTAINER PLATFORM クラスターメトリクス POD のスケーリング	719
38.9. CLEANUP	719
<b>第39章 WEB コンソールのカスタマイズ</b> .....	<b>721</b>
39.1. 概要	721
39.2. 拡張スクリプトとスタイルシートの読み込み	721
39.2.1. 拡張プロパティの設定	722
39.3. 外部ロギングソリューションの拡張オプション	723
39.4. ガイド付きツアーのカスタマイズと無効化	723
39.5. ドキュメントリンクのカスタマイズ	723
39.6. ロゴのカスタマイズ	723
39.7. メンバーシップのホワイトリストのカスタマイズ	724
39.8. ドキュメントへのリンクの変更	724
39.9. CLI をダウンロードするリンクの追加または変更	725
39.9.1. About ページのカスタマイズ	725
39.10. ナビゲーションメニューの設定	726
39.10.1. トップナビゲーションドロップダウンメニュー	726
39.10.2. アプリケーションランチャー	727
39.10.3. システムステータスバッジ	727
39.10.4. プロジェクトの左ナビゲーション	728
39.11. 主要アプリケーションの設定	729
39.12. カタログカテゴリーの設定	730
39.13. クォータ通知メッセージの設定	731
39.14. CREATE FROM URL NAMESPACE ホワイトリストの設定	732
39.15. COPY LOGIN コマンドの無効化	732
39.15.1. ワイルドカードルートの有効化	732
39.16. ログインページのカスタマイズ	733
39.16.1. 使用例	733
39.17. OAUTH エラーページのカスタマイズ	733
39.18. ログアウト URL の変更	734
39.19. ANSIBLE を使用した WEB コンソールカスタマイズの設定	734
39.20. WEB コンソール URL ポートおよび証明書の変更	735
<b>第40章 外部永続ボリュームプロビジョナーのデプロイ</b> .....	<b>736</b>
40.1. 概要	736
40.2. 操作を始める前に	736
40.2.1. 外部プロビジョナーの Ansible ロール	736
40.2.2. 外部プロビジョナーの Ansible 変数	736
40.2.3. AWS EFS プロビジョナーの Ansible 変数	737
40.3. プロビジョナーのデプロイ	738

---

40.3.1. AWS EFS プロビジョナーのデプロイ	738
40.3.1.1. AWS EFS オブジェクト定義	738
40.4. CLEANUP	739
<b>第41章 OPERATOR FRAMEWORK (テクノロジープレビュー) のインストール</b> .....	<b>740</b>
41.1. このテクノロジープレビューについて	740
41.2. ANSIBLE を使用した OPERATOR LIFECYCLE MANAGER のインストール	742
41.3. 最初の OPERATOR の起動	743
41.4. ご協力をお願い	749
<b>第42章 OPERATOR LIFECYCLE MANAGER のアンインストール</b> .....	<b>750</b>
42.1. ANSIBLE を使用した OPERATOR LIFECYCLE MANAGER のアンインストール	750



## 第1章 概要

本書では、OpenShift Container Platform クラスターのインストール後に利用できる追加の設定オプションについて説明します。

## 第2章 レジストリーのセットアップ

### 2.1. 内部レジストリーの概要

#### 2.1.1. レジストリーについて

OpenShift Container Platform は、ソースコードから [コンテナイメージ](#) をビルドし、デプロイし、そのライフサイクルを管理することができます。これを可能にするため、OpenShift Container Platform は内部の [統合コンテナイメージレジストリー](#) を提供しています。このレジストリーは OpenShift Container Platform 環境にデプロイでき、ここからイメージをローカルで管理できます。

#### 2.1.2. 統合レジストリーまたはスタンドアロンレジストリー

完全な OpenShift Container Platform クラスターの初回インストール時に、レジストリーはインストールプロセスで自動的にデプロイされている可能性があります。自動的にデプロイされていなかった場合やレジストリー設定のカスタマイズが追加で必要になる場合には、[既存クラスターへのレジストリーのデプロイ](#) を参照してください。

OpenShift Container Platform のレジストリーは、完全な OpenShift Container Platform クラスターの統合部分として実行されるようにデプロイすることが可能である一方、スタンドアロンのコンテナイメージレジストリーとして個別にインストールすることも可能です。

スタンドアロンレジストリーをインストールするには、[スタンドアロンレジストリーのインストール](#) の手順に従ってください。このインストールパスは、レジストリーと専用の Web コンソールを実行するオールインワンのクラスターをデプロイします。

### 2.2. 既存クラスターへのレジストリーのデプロイ

#### 2.2.1. 概要

OpenShift Container Platform クラスターの初回インストール時に統合レジストリーが事前に自動的にデプロイされなかった場合や、正常に実行されず、既存のクラスターに再デプロイする必要がある場合は、以下のセクションで新規レジストリーをデプロイするためのオプションを参照してください。



#### 注記

[スタンドアロンレジストリー](#) をインストールしていない場合、このトピックは不要になります。

#### 2.2.2. レジストリーホスト名の設定

内部参照と外部参照の両方でレジストリーを認識するために使用するホスト名およびポートを設定できます。これを実行すると、イメージストリームはイメージのホスト名ベースのプッシュおよびプル仕様を提供することができ、これによってイメージのコンシューマーがレジストリーのサービス IP の変更の影響を受けないようにし、イメージストリームとその参照をクラスター間で移植できる可能性があります。

クラスター内からレジストリーを参照するために使用されるホスト名を設定するには、マスター設定ファイルの `imagePolicyConfig` セクションで `internalRegistryHostname` を設定します。外部のホスト名は、同じ場所で `externalRegistryHostname` の値を設定することで管理されます。

#### イメージポリシーの設定

```
imagePolicyConfig:
  internalRegistryHostname: docker-registry.default.svc.cluster.local:5000
  externalRegistryHostname: docker-registry.mycompany.com
```

レジストリー自体は、同じ内部ホスト名の値で設定する必要があります。これは、レジストリーのデプロイメント設定で **REGISTRY\_OPENSHIFT\_SERVER\_ADDR** 環境変数を設定するか、またはレジストリー設定の [OpenShift セクション](#) で値を設定することで実行できます。



### 注記

レジストリーで TLS を有効にしている場合、サーバー証明書にはレジストリーの参照に使用されるホスト名が含まれている必要があります。ホスト名をサーバー証明書に追加する方法については、[レジストリーのセキュリティー保護](#) を参照してください。

## 2.2.3. レジストリーのデプロイ

統合コンテナイメージレジストリーをデプロイするには、クラスター管理者権限を持つユーザーとして **oc adm registry** コマンドを使用します。以下に例を示します。

```
$ oc adm registry --config=/etc/origin/master/admin.kubeconfig \ ❶
  --service-account=registry \ ❷
  --images='registry.redhat.io/openshift3/ose-${component}:${version}' ❸
```

- ❶ **--config** は、[クラスター管理者](#) のための [CLI 設定ファイル](#) へのパスです。
- ❷ **--service-account** は、レジストリーの Pod の実行に使用されるサービスアカウントです。
- ❸ OpenShift Container Platform の適切なイメージをプルするために必要です。**\${component}** および **\${version}** はインストール時に動的に置き換えられます。

これにより、**docker-registry** と呼ばれるサービスとデプロイメント設定が作成されます。正常にデプロイされると、Pod が **docker-registry-1-cpty9** のような名前で作成されます。

レジストリーの作成時に指定できるオプションの詳細の一覧を表示するには、以下を実行します。

```
$ oc adm registry --help
```

**--fs-group** の値は、レジストリーが使用する [SCC](#) (通常は制限付き SCC) によって許可されている必要があります。

## 2.2.4. レジストリーの DaemonSet としてのデプロイ

レジストリーを **--daemonset** オプションを使用して **DaemonSet** としてデプロイするには、**oc adm registry** コマンドを使用します。

DaemonSet は、ノードの作成時に、それらに指定された Pod のコピーが含まれていることを確認します。ノードが削除されると、Pod はガベージコレクションされます。

**DaemonSet** の詳細は、[Daemonset の使用](#) を参照してください。

## 2.2.5. レジストリーのコンピュートリソース

デフォルトでは、レジストリーは [コンピュートリソースの要求や制限](#) に関する設定がない状態で作成されます。実稼働環境では、レジストリーのデプロイメント設定を更新してレジストリー Pod のリソース要求や制限を設定しておくことが強く推奨されます。設定しない場合、レジストリー Pod は [BestEffort Pod](#) と判断されます。

要求や制限の設定に関する詳細は、[コンピュートリソース](#) を参照してください。

## 2.2.6. レジストリーのストレージ

レジストリーには、コンテナのイメージとメタデータが格納されています。Pod をレジストリーと共に単純にデプロイする場合、Pod の終了時に破棄される一時的なボリュームを使用します。誰かがビルドまたはレジストリーにプッシュしたイメージは消えます。

このセクションでは、サポートされているレジストリーのストレージドライバーの一覧を示します。詳細は、[コンテナイメージレジストリーのドキュメント](#) を参照してください。

以下の一覧には、レジストリーの設定ファイルで設定する必要があるストレージドライバーが含まれます。

- [ファイルシステム](#)。ファイルシステムはデフォルトですので、設定する必要はありません。
- [S3](#)。詳細は、[CloudFront の設定](#) のドキュメントを参照してください。
- [OpenStack Swift](#)
- [Google Cloud Storage \(GCS\)](#)
- [Microsoft Azure](#)
- [Aliyun OSS](#)

レジストリーの一般的なストレージ設定オプションはサポートされています。詳細は、[コンテナイメージレジストリーのドキュメント](#) を参照してください。

以下のストレージオプションは、[ファイルシステムドライバー](#) で設定する必要があります。

- [GlusterFS ストレージ](#)
- [Ceph Rados ブロックデバイス](#)



### 注記

サポートされている永続ストレージドライバーの詳細は、[永続ストレージの設定](#) および [永続ストレージの例](#) を参照してください。

### 2.2.6.1. 実稼働環境での使用

実稼働環境での使用の場合には、リモートボリュームを割り当てるか、または [各自が選択した永続ストレージ方法を定義して使用します](#)。

たとえば、既存の Persistent Volume Claim (永続ボリューム要求) を使用するには、以下を実行します。

```
$ oc set volume deploymentconfigs/docker-registry --add --name=registry-storage -t pvc \
  --claim-name=<pvc_name> --overwrite
```

## 重要

テストにより、RHEL NFS サーバーをコンテナイメージレジストリーのストレージバックエンドとして使用することに関する問題が検出されています。これには、OpenShift Container レジストリーおよび Quay が含まれます。そのため、コアサービスで使用される PV をサポートするために RHEL NFS サーバーを使用することは推奨されていません。

他の NFS の実装ではこれらの問題が検出されない可能性があります。OpenShift コアコンポーネントに対して実施された可能性のあるテストに関する詳細情報は、個別の NFS 実装ベンダーにお問い合わせください。

### 2.2.6.1.1. Amazon S3 のストレージのバックエンドとしての使用

Amazon Simple Storage Service のストレージを内部コンテナイメージレジストリーと共に使用する方法もあります。Amazon Simple Storage Service は [AWS マネジメントコンソール](#) で管理できるセキュアなクラウドストレージです。これを使用するには、レジストリーの設定ファイルを手動で編集し、レジストリー Pod にマウントする必要があります。ただし、設定を開始する前に、アップストリームの [推奨される手順](#) を確認してください。

[デフォルトの YAML 設定ファイル](#) をベースとして取得し、`storage` セクションの `filesystem` エントリーを、以下のように `s3` エントリーに置き換えます。これにより、ストレージのセクションは以下のようになります。

```
storage:
  cache:
    layerinfo: inmemory
  delete:
    enabled: true
  s3:
    accesskey: awsaccesskey 1
    secretkey: awssecretkey 2
    region: us-west-1
    regionendpoint: http://myobjects.local
    bucket: bucketname
    encrypt: true
    keyid: mykeyid
    secure: true
    v4auth: false
    chunksize: 5242880
    rootdirectory: /s3/object/name/prefix
```

- 1** Amazon のアクセスキーに置き換えてください。
- 2** Amazon のシークレットキーに置き換えてください。

`s3` のすべての設定オプションは、アップストリームの [ドライバー参照ドキュメント](#) に記載されています。

[レジストリー設定を上書き](#) すると、設定ファイルを Pod にマウントするための追加の手順に進みます。

**警告**

レジストリーが S3 ストレージのバックエンドで実行されると、[問題が報告されま](#)す。

使用している統合レジストリーでサポートされていない S3 リージョンを使用する必要がある場合は、[S3 ドライバー設定](#) を参照してください。

**2.2.6.2. 非実稼働環境での使用**

非実稼働環境の場合、`--mount-host=<path>` オプションを使って、永続ストレージに使用するレジストリーのディレクトリーを指定します。次に、レジストリーのボリュームがホストのマウントとして、指定された `<path>` に作成されます。

**重要**

`--mount-host` オプションは、レジストリーのコンテナが実行されているノードからディレクトリーをマウントします。`docker-registry` デプロイメント設定をスケールアップすると、レジストリー Pod とコンテナが別々のノードで実行されるので、それぞれ独自のローカルストレージを使用したレジストリーコンテナが 2 つ以上作成される可能性があります。これは予期しない動作を生じさせます。その後には繰り返される同一イメージのプル要求が最終的に到達するコンテナによっては必ずしも成功しない場合があります。

`--mount-host` オプションは、レジストリーコンテナを特権モードで実行することを要求します。この要求は、ユーザーが `--mount-host` を指定すると自動的に有効にされます。ただしデフォルトでは、すべての Pod が [特権付きコンテナ](#) をデフォルトで実行できる訳ではありません。それでもこのオプションを使用する必要がある場合は、レジストリーを作成してから、レジストリーがインストール時に作成された `registry` サービスアカウントを使用するように指定してください。

```
$ oc adm registry --service-account=registry \
  --config=/etc/origin/master/admin.kubeconfig \
  --images='registry.redhat.io/openshift3/ose-${component}:${version}' \ 1
  --mount-host=<path>
```

**1** OpenShift Container Platform の適切なイメージをプルするために必要です。 `${component}` および `${version}` はインストール時に動的に置き換えられます。

**重要**

コンテナイメージレジストリー Pod は、ユーザー `1001` として実行されます。このユーザーは、ホストのディレクトリーへの書き込みができなければなりません。したがって、以下のコマンドでディレクトリーの所有権をユーザー ID `1001` に変更する必要があります。

```
$ sudo chown 1001:root <path>
```

## 2.2.7. レジストリーコンソールの有効化

OpenShift Container Platform は、統合レジストリーへの Web ベースのインターフェイスを提供します。このレジストリーコンソールは、イメージを参照および管理するためのコンポーネント (任意) です。Pod として実行されているステートレスサービスとしてデプロイされます。



### 注記

OpenShift Container Platform を [スタンドアロンレジストリー](#) としてインストールした場合、レジストリーコンソールはインストール時にすでにデプロイされ、そのセキュリティが自動的に保護されています。



### 重要

Cockpit がすでに実行されている場合、レジストリーコンソールとのポート競合 (デフォルトでは 9090) を避けるために、次に進む前にこれをシャットダウンする必要があります。

### 2.2.7.1. レジストリーコンソールのデプロイ



### 重要

最初に [レジストリーを公開](#) しておく必要があります。

1. デフォルトのプロジェクトにパススルールートを作成します。このルートは、以下の手順でレジストリーコンソールのアプリケーションを作成する際に必要になります。

```
$ oc create route passthrough --service registry-console \
  --port registry-console \
  -n default
```

2. レジストリーコンソールのアプリケーションをデプロイします。<openshift\_oauth\_url> を OpenShift Container Platform OAuth プロバイダーの URL に置き換えます。通常これはマスターになります。

```
$ oc new-app -n default --template=registry-console \
  -p OPENSIFT_OAUTH_PROVIDER_URL="https://<openshift_oauth_url>:8443" \
  -p REGISTRY_HOST=$(oc get route docker-registry -n default --template='{{ .spec.host }}') \
  -p COCKPIT_KUBE_URL=$(oc get route registry-console -n default --template='https://{{ .spec.host }}')
```



### 注記

レジストリーコンソールへのログインの試行時にリダイレクト URL が間違っていた場合は、`oc get oauthclients` で OAuth クライアントを確認してください。

3. 最後に、Web ブラウザーでこのルート URI を使用しているコンソールを表示します。

### 2.2.7.2. レジストリーコンソールのセキュリティ保護

デフォルトでは、レジストリーコンソールは、[レジストリーコンソールのデプロイ](#) の手順ごとに手動でデプロイされる場合に自己署名 TLS 証明書を生成します。詳細は、[レジストリーコンソールのトラブルシューティング](#) を参照してください。

組織の署名済み証明書をシークレットボリュームとして追加する際には、以下の手順に従ってください。ここでは、証明書が **oc** クライアントホストで利用可能であることを前提としています。

1. 証明書とキーを含む **.cert** ファイルを作成します。以下を使用してファイルをフォーマットします。

- サーバー証明書と中間証明機関のための1つ以上数の **BEGIN CERTIFICATE** ブロック。
- キーの **BEGIN PRIVATE KEY** または同種のものを含むブロック。キーは暗号化することができません。  
以下に例を示します。

```
-----BEGIN CERTIFICATE-----
MIIDUzCCAjugAwIBAgIJAPXW+CuNYS6QMA0GCSqGSIb3DQEBCwUAMD8xKTAkBgNV
BAoMIGI0OGE2NGNkNmMwNTQ1YThhZTgxOTEzZDE5YmJmRjMRLwEAYDVQQDD
Als
...
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIDUzCCAjugAwIBAgIJAPXW+CuNYS6QMA0GCSqGSIb3DQEBCwUAMD8xKTAkBgNV
BAoMIGI0OGE2NGNkNmMwNTQ1YThhZTgxOTEzZDE5YmJmRjMRLwEAYDVQQDD
Als
...
-----END CERTIFICATE-----
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBAKgwggSkAgEAAoIBAQCyOJ5garOYw0sm
8TBCDSqQ/H1awGMzDYdB11xuHHsxYS2VepPMzMzryHR13714dGFLhvdTvJUH8IUS
...
-----END PRIVATE KEY-----
```

- セキュリティーの保護されたレジストリーには、以下の SAN (サブジェクトの別名: Subject Alternative Names) 一覧が含まれているはずでず。
  - 2つのサービスのホスト名。  
以下に例を示します。

```
docker-registry.default.svc.cluster.local
docker-registry.default.svc
```

- サービス IP アドレス。  
以下に例を示します。

```
172.30.124.220
```

以下のコマンドを使ってコンテナイメージレジストリーのサービス IP アドレスを取得します。

```
oc get service docker-registry --template='{{.spec.clusterIP}}'
```

- パブリックホスト名。  
以下に例を示します。

```
docker-registry-default.apps.example.com
```

以下のコマンドを使ってコンテナイメージレジストリーのパブリックホスト名を取得します。

```
oc get route docker-registry --template '{{.spec.host}}'
```

たとえば、サーバー証明書には以下のような SAN の詳細が記載されるはずです。

```
X509v3 Subject Alternative Name:
      DNS:docker-registry-public.openshift.com, DNS:docker-registry.default.svc,
      DNS:docker-registry.default.svc.cluster.local, DNS:172.30.2.98, IP
      Address:172.30.2.98
```

レジストリーコンソールは、証明書を `/etc/cockpit/ws-certs.d` ディレクトリーから読み込みます。また、拡張子 `.cert` が付いたファイルをアルファベット順で (最後から) 使用します。したがって、`.cert` ファイルには OpenSSL スタイルでフォーマットされた PEM ブロックが少なくとも 2 つ含まれる必要があります。

証明書がない場合、自己署名証明書が `openssl` コマンドで作成され、`O-self-signed.cert` ファイルに保存されます。

2. シークレットを作成します。

```
$ oc create secret generic console-secret \
  --from-file=/path/to/console.cert
```

3. このシークレットを `registry-console` デプロイメント設定に追加します。

```
$ oc set volume dc/registry-console --add --type=secret \
  --secret-name=console-secret -m /etc/cockpit/ws-certs.d
```

これにより、レジストリーコンソールの新規デプロイメントがトリガーされ、署名済み証明書が組み込まれます。

### 2.2.7.3. レジストリーコンソールのトラブルシューティング

#### 2.2.7.3.1. デバッグモード

レジストリーコンソールのデバッグモードは環境変数を使用して有効にされます。以下のコマンドは、レジストリーコンソールをデバッグモードで再デプロイします。

```
$ oc set env dc registry-console G_MESSAGES_DEBUG=cockpit-ws,cockpit-wrapper
```

デバッグモードを有効にすると、より詳細なログがレジストリーコンソールの Pod ログに表示されます。

#### 2.2.7.3.2. SSL 証明書パスの表示

レジストリーコンソールが使用している証明書を確認するには、コマンドをコンソール Pod 内から実行できます。

1. **default** プロジェクトで Pod を一覧表示し、レジストリーコンソールの Pod 名を検索します。

```
$ oc get pods -n default
NAME                READY   STATUS    RESTARTS   AGE
registry-console-1-rssrw 1/1     Running  0          1d
```

2. 直前のコマンドで取得した Pod 名を使って、**cockpit-ws** プロセスが使用している証明書パスを取得します。以下は、自動生成された証明書を使用しているコンソールの例です。

```
$ oc exec registry-console-1-rssrw remotectl certificate
certificate: /etc/cockpit/ws-certs.d/0-self-signed.cert
```

## 2.3. レジストリーへのアクセス

### 2.3.1. ログの表示

コンテナイメージレジストリーのログを表示するには、デプロイメント設定で **oc logs** コマンドを使用します。

```
$ oc logs dc/docker-registry
2015-05-01T19:48:36.300593110Z time="2015-05-01T19:48:36Z" level=info
msg="version=v2.0.0+unknown"
2015-05-01T19:48:36.303294724Z time="2015-05-01T19:48:36Z" level=info msg="redis not
configured" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
2015-05-01T19:48:36.303422845Z time="2015-05-01T19:48:36Z" level=info msg="using inmemory
layerinfo cache" instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
2015-05-01T19:48:36.303433991Z time="2015-05-01T19:48:36Z" level=info msg="Using OpenShift
Auth handler"
2015-05-01T19:48:36.303439084Z time="2015-05-01T19:48:36Z" level=info msg="listening on :5000"
instance.id=9ed6c43d-23ee-453f-9a4b-031fea646002
```

### 2.3.2. ファイルストレージ

タグとイメージメタデータは OpenShift Container Platform に格納されますが、レジストリーは、レイヤーと署名データを **/registry** にあるレジストリーコンテナにマウントされているボリュームに格納します。**oc exec** は特権付きコンテナでは機能しないため、レジストリーの内容を確認するには、レジストリー Pod のコンテナを格納しているノードに対して SSH を手動で実行し、コンテナ自体で **docker exec** を実行します。

1. 現在の Pod を一覧表示し、コンテナイメージレジストリーの Pod 名を検索します。

```
# oc get pods
```

次に、**oc describe** を使用して、コンテナを実行しているノードのホスト名を検索します。

```
# oc describe pod <pod_name>
```

2. 目的のノードにログインします。

```
# ssh node.example.com
```

3. ノードホストのデフォルトプロジェクトから実行中のコンテナを一覧表示し、コンテナイメージレジストリーのコンテナ ID を特定します。

```
# docker ps --filter=name=registry_docker-registry.*_default_
```

4. **oc rsh** コマンドを使用してレジストリーの内容を一覧表示します。

```
# oc rsh dc/docker-registry find /registry
/registry/docker
/registry/docker/registry
/registry/docker/registry/v2
/registry/docker/registry/v2/blobs 1
/registry/docker/registry/v2/blobs/sha256
/registry/docker/registry/v2/blobs/sha256/ed
/registry/docker/registry/v2/blobs/sha256/ed/ede17b139a271d6b1331ca3d83c648c24f92cece5f89d95ac6c34ce751111810
/registry/docker/registry/v2/blobs/sha256/ed/ede17b139a271d6b1331ca3d83c648c24f92cece5f89d95ac6c34ce751111810/data 2
/registry/docker/registry/v2/blobs/sha256/a3
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
/registry/docker/registry/v2/blobs/sha256/a3/a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4/data
/registry/docker/registry/v2/blobs/sha256/f7
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb26f259582bb33502bdb0fcf5011e03c60577c4284845
/registry/docker/registry/v2/blobs/sha256/f7/f72a00a23f01987b42cb26f259582bb33502bdb0fcf5011e03c60577c4284845/data
/registry/docker/registry/v2/repositories 3
/registry/docker/registry/v2/repositories/p1
/registry/docker/registry/v2/repositories/p1/pause 4
/registry/docker/registry/v2/repositories/p1/pause/_manifests
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256/e9a2ac6418981e97b399d3709f1b4a6d2723cd38a4909215ce2752a5c068b1cf
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256/e9a2ac6418981e97b399d3709f1b4a6d2723cd38a4909215ce2752a5c068b1cf/signatures 5
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256/e9a2ac6418981e97b399d3709f1b4a6d2723cd38a4909215ce2752a5c068b1cf/signatures/sha256
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256/e9a2ac6418981e97b399d3709f1b4a6d2723cd38a4909215ce2752a5c068b1cf/signatures/sha256/ede17b139a271d6b1331ca3d83c648c24f92cece5f89d95ac6c34ce751111810
/registry/docker/registry/v2/repositories/p1/pause/_manifests/revisions/sha256/e9a2ac6418981e97b399d3709f1b4a6d2723cd38a4909215ce2752a5c068b1cf/signatures/sha256/ede17b139a271d6b1331ca3d83c648c24f92cece5f89d95ac6c34ce751111810/link 6
/registry/docker/registry/v2/repositories/p1/pause/_uploads 7
/registry/docker/registry/v2/repositories/p1/pause/_layers 8
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/a3ed95caeb02ffe68cdd9fd844
```

```
06680ae93d633cb16422d00e8a7c22955b46d4/link 9
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/f72a00a23f01987b42cb26f25
9582bb33502bdb0fcf5011e03c60577c4284845
/registry/docker/registry/v2/repositories/p1/pause/_layers/sha256/f72a00a23f01987b42cb26f25
9582bb33502bdb0fcf5011e03c60577c4284845/link
```

- 1 このディレクトリーには、すべてのレイヤーと署名が Blob として格納されます。
- 2 このファイルには、Blob の内容が含まれます。
- 3 このディレクトリーには、すべてのイメージリポジトリーが格納されます。
- 4 このディレクトリーは単一イメージリポジトリーの `p1/pause` 用です。
- 5 このディレクトリーには、特定のイメージマニフェストリビジョンの署名が含まれます。
- 6 このファイルには、Blob (署名データを含む) への参照が含まれます。
- 7 このディレクトリーには、指定されたりポジトリーに対して現在アップロードされステージングされているすべてのレイヤーが含まれます。
- 8 このディレクトリーには、このリポジトリーが参照するすべてのレイヤーへのリンクが含まれます。
- 9 このファイルには、イメージを介してこのリポジトリーにリンクされている特定のレイヤーへの参照が含まれます。

### 2.3.3. レジストリーへの直接アクセス

さらに高度な使用方法として、レジストリーに直接アクセスし、**docker** コマンドを起動することが可能です。これにより、**docker push** や **docker pull** などの操作で直接イメージをプッシュするか、または統合レジストリーからイメージをプルすることができます。これを実行するには、**docker login** コマンドを使ってレジストリーにログインする必要があります。実行できる操作は、以下のセクションで説明されているようにユーザーが持つパーミッションによって異なります。

#### 2.3.3.1. ユーザーの前提条件

レジストリーに直接アクセスするには、使用するユーザーが、使用目的に応じて以下の前提条件を満たしている必要があります。

- 直接アクセスするには、ユーザーは選択する [アイデンティティプロバイダー](#) の [通常ユーザー](#) である必要があります。通常ユーザーは、レジストリーへのログインに必要なアクセストークンを生成できます。`system:admin` などの [システムユーザー](#) はアクセストークンを取得できないため、レジストリーに直接アクセスすることはできません。たとえば `HTPASSWD` 認証を使用している場合は、以下のコマンドを使用してこれを作成することができます。

```
# htpasswd /etc/origin/master/htpasswd <user_name>
```

- **docker pull** コマンドを使用する場合などにイメージをプルするには、ユーザーに `registry-viewer` ロールがなければなりません。このロールを追加するには、以下を実行します。

```
$ oc policy add-role-to-user registry-viewer <user_name>
```

- イメージの書き出しやプッシュを実行するには (**docker push** コマンドを使用する場合など)、ユーザーに **registry-editor** ロールが必要です。このロールを追加するには、以下を実行します。

```
$ oc policy add-role-to-user registry-editor <user_name>
```

ユーザーパーミッションに関する詳細は、[ロールバインディングの管理](#) を参照してください。

### 2.3.3.2. レジストリーへのログイン



#### 注記

ユーザーが、レジストリーに直接アクセスできるための [前提条件](#) を満たしていることを確認してください。

レジストリーに直接ログインするには、以下を実行します。

1. OpenShift Container Platform に **通常ユーザー** としてログインしていることを確認します。

```
$ oc login
```

2. アクセストークンを使用してコンテナイメージレジストリーにログインします。

```
docker login -u openshift -p $(oc whoami -t) <registry_ip>:<port>
```



#### 注記

ユーザー名の任意の値を渡すことができ、トークンには必要な情報がすべて含まれます。コロンを含むユーザー名を渡すとログインに失敗します。

### 2.3.3.3. イメージのプッシュとプル

[レジストリーにログイン](#) すると、レジストリーに **docker pull** および **docker push** 操作を実行できます。



#### 重要

任意のイメージをプルできますが、**system:registry** ロールを追加している場合は、各自のプロジェクトにあるレジストリーにのみイメージをプッシュすることができます。

次の例では、以下を使用します。

コンポーネント	値
<registry_ip>	<b>172.30.124.220</b>
<port>	<b>5000</b>
<project>	<b>openshift</b>

<image>	<b>busybox</b>
<tag>	省略 (デフォルトは <b>latest</b> )

1. 任意のイメージをプルします。

```
$ docker pull docker.io/busybox
```

2. 新規イメージに **<registry\_ip>:<port>/<project>/<image>** 形式でタグ付けします。プロジェクト名は、イメージを正しくレジストリーに配置し、これに後でアクセスできるようにするために OpenShift Container Platform の [プル仕様](#) に表示される **必要があります**。

```
$ docker tag docker.io/busybox 172.30.124.220:5000/openshift/busybox
```



### 注記

通常ユーザーには、指定されたプロジェクトの **system:image-builder** ロールが必要です。このロールにより、ユーザーはイメージの書き出しやプッシュを実行できます。そうしないと、次の手順の **docker push** は失敗します。テストするには、[新しいプロジェクトを作成](#) して、**busybox** イメージをプッシュします。

3. 新しくタグ付けされたイメージをレジストリーにプッシュします。

```
$ docker push 172.30.124.220:5000/openshift/busybox
...
cf2616975b4a: Image successfully pushed
Digest: sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f77403cab55
```

### 2.3.4. レジストリーメトリクスへのアクセス

OpenShift Container レジストリーは、[Prometheus メトリクス](#) のエンドポイントを提供します。Prometheus はスタンドアロンのオープンソースのシステムモニタリングおよびアラートツールキットです。

メトリクスは、レジストリーエンドポイントの **/extensions/v2/metrics** パスに公開されます。ただし、このルートは最初に有効にされている必要があります。有効化の方法については [レジストリー設定の拡張](#) を参照してください。

以下は、メトリクスクエリーの簡単な例です。

```
$ curl -s -u <user>:<secret> \ 1
  http://172.30.30.30:5000/extensions/v2/metrics | grep openshift | head -n 10

# HELP openshift_build_info A metric with a constant '1' value labeled by major, minor, git commit &
git version from which OpenShift was built.
# TYPE openshift_build_info gauge
openshift_build_info{gitCommit="67275e1",gitVersion="v3.6.0-alpha.1+67275e1-
803",major="3",minor="6+"} 1
# HELP openshift_registry_request_duration_seconds Request latency summary in microseconds for
each operation
# TYPE openshift_registry_request_duration_seconds summary
```

```

openshift_registry_request_duration_seconds{name="test/origin-pod",operation="blobstore.create",quantile="0.5"} 0
openshift_registry_request_duration_seconds{name="test/origin-pod",operation="blobstore.create",quantile="0.9"} 0
openshift_registry_request_duration_seconds{name="test/origin-pod",operation="blobstore.create",quantile="0.99"} 0
openshift_registry_request_duration_seconds_sum{name="test/origin-pod",operation="blobstore.create"} 0
openshift_registry_request_duration_seconds_count{name="test/origin-pod",operation="blobstore.create"} 5

```

- 1 **<user>** は任意ですが、**<secret>** は [レジストリー設定](#) で指定された値と一致していなければなりません。

メトリクスにアクセスするための別の方法は、クラスターロールの使用です。エンドポイントはここでも有効にする必要がありますが、**<secret>** を指定する必要はありません。設定ファイルのメトリクスに対応する部分は以下のようになります。

```

openshift:
  version: 1.0
  metrics:
    enabled: true
  ...

```

メトリクスにアクセスするために必要なクラスターロールがない場合、これを作成する必要があります。

```

$ cat <<EOF |
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus-scraper
rules:
- apiGroups:
  - image.openshift.io
  resources:
  - registry/metrics
  verbs:
  - get
EOF
oc create -f -

```

このロールをユーザーに追加するには、以下のコマンドを実行します。

```
$ oc adm policy add-cluster-role-to-user prometheus-scraper <username>
```

高度なクエリーと推奨されるビジュアライザーについては、[アップストリームの Prometheus ドキュメント](#) を参照してください。

## 2.4. レジストリーのセキュリティー保護および公開

### 2.4.1. 概要

デフォルトでは、OpenShift Container Platform レジストリーは、TLS 経由でトラフィックを提供できるようにクラスターのインストール時にセキュリティー保護されます。サービスを外部に公開するために、パススルールートもデフォルトで作成されます。

何らかの理由でレジストリーが保護されていないか、または公開されていない場合は、これらを手動で実行する方法について以下のセクションを参照してください。

## 2.4.2. レジストリーを手動でセキュリティー保護する

レジストリーを手動でセキュリティー保護して TLS 経由でトラフィックを処理するには、以下を実行します。

1. レジストリーをデプロイします。
2. レジストリーのサービス IP とポートを取得します。

```
$ oc get svc/docker-registry
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP  PORT(S)    AGE
docker-registry ClusterIP  172.30.82.152 <none>      5000/TCP   1d
```

3. 既存のサーバー証明書を使用するか、またはキーと、指定された CA で署名された指定 IP およびホスト名に有効なサーバー証明書を作成します。レジストリーのサービス IP と **docker-registry.default.svc.cluster.local** ホスト名のサーバー証明書を作成するには、Ansible ホストインベントリーファイル (デフォルトでは `/etc/ansible/hosts`) に一覧表示されている最初のマスターから以下のコマンドを実行します。

```
$ oc adm ca create-server-cert \
  --signer-cert=/etc/origin/master/ca.crt \
  --signer-key=/etc/origin/master/ca.key \
  --signer-serial=/etc/origin/master/ca.serial.txt \
  --hostnames='docker-registry.default.svc.cluster.local,docker-
registry.default.svc,172.30.124.220' \
  --cert=/etc/secrets/registry.crt \
  --key=/etc/secrets/registry.key
```

ルーターを **外部に公開する** 場合、公開ルートのホスト名を **--hostnames** フラグに追加します。

```
--hostnames='mydocker-registry.example.com,docker-
registry.default.svc.cluster.local,172.30.124.220'
```

ルートを外部にアクセス可能にするためにデフォルトの証明書を更新する際のその他詳細については、[レジストリーとルーター証明書の再デプロイ](#) を参照してください。



### 注記

**oc adm ca create-server-cert** コマンドは、2 年間有効な証明書を生成します。この期間は **--expire-days** オプションを使って変更することができますが、セキュリティー上の理由から、値をこれより大きくすることは推奨されません。

4. レジストリー証明書のシークレットを作成します。

```
$ oc create secret generic registry-certificates \
  --from-file=/etc/secrets/registry.crt \
  --from-file=/etc/secrets/registry.key
```

- シークレットをレジストリー Pod のサービスアカウント (デフォルトのサービスアカウントなど) に追加します。

```
$ oc secrets link registry registry-certificates
$ oc secrets link default registry-certificates
```



### 注記

シークレットを参照しているサービスアカウントにのみにシークレットを制限することはデフォルトで無効にされています。つまり、マスターの設定ファイルで **serviceAccountConfig.limitSecretReferences** がマスター設定の **false** (デフォルトの設定) に設定されている場合は、サービスにシークレットをリンクする必要はありません。

- docker-registry** サービスを一時停止します。

```
$ oc rollout pause dc/docker-registry
```

- シークレットボリュームをレジストリーのデプロイメント設定に追加します。

```
$ oc set volume dc/docker-registry --add --type=secret \
  --secret-name=registry-certificates -m /etc/secrets
```

- 以下の環境変数をレジストリーのデプロイメント設定に追加して TLS を有効にします。

```
$ oc set env dc/docker-registry \
  REGISTRY_HTTP_TLS_CERTIFICATE=/etc/secrets/registry.crt \
  REGISTRY_HTTP_TLS_KEY=/etc/secrets/registry.key
```

詳細は、[Docker ドキュメントのレジストリーの設定](#) のセクションを参照してください。

- レジストリーの liveness probe に使用されているスキームを HTTP から HTTPS に更新します。

```
$ oc patch dc/docker-registry -p '{"spec": {"template": {"spec": {"containers": [{"name": "registry", "livenessProbe": {"httpGet": {"scheme": "HTTPS"}}]}}}}'
```

- レジストリーを OpenShift Container Platform 3.2 以降に初めてデプロイした場合は、レジストリーの readiness probe として使用されているスキームを HTTP から HTTPS に更新します。

```
$ oc patch dc/docker-registry -p '{"spec": {"template": {"spec": {"containers": [{"name": "registry", "readinessProbe": {"httpGet": {"scheme": "HTTPS"}}]}}}}'
```

- docker-registry** サービスを再開します。

-

```
$ oc rollout resume dc/docker-registry
```

12. レジストリーが TLS モードで実行されていることを検証します。最後の **docker-registry** のデプロイメントが完了するまで待機し、Docker ログでレジストリーコンテナを確認します。 **listening on :5000, tls** のエントリーが見つかるはずですが。

```
$ oc logs dc/docker-registry | grep tls
time="2015-05-27T05:05:53Z" level=info msg="listening on :5000, tls" instance.id=deeba528-c478-41f5-b751-dc48e4935fc2
```

13. CA 証明書を Docker 証明書ディレクトリーにコピーします。これはクラスター内のすべてのノードで実行する必要があります。

```
$ dcertsdir=/etc/docker/certs.d
$ destdir_addr=$dcertsdir/172.30.124.220:5000
$ destdir_name=$dcertsdir/docker-registry.default.svc.cluster.local:5000

$ sudo mkdir -p $destdir_addr $destdir_name
$ sudo cp ca.crt $destdir_addr ①
$ sudo cp ca.crt $destdir_name
```

① **ca.crt** ファイルは、マスター上の **/etc/origin/master/ca.crt** のコピーです。

14. 認証を使用している場合、**docker** のバージョンによっては、OS レベルで証明書を信頼するようにクラスターを設定することも必要になります。
  - a. 証明書をコピーします。

```
$ cp /etc/origin/master/ca.crt /etc/pki/ca-trust/source/anchors/myregistrydomain.com.crt
```

- b. 以下を実行します。

```
$ update-ca-trust enable
```

15. **/etc/sysconfig/docker** ファイルのこの特定のレジストリーに対してのみ、**--insecure-registry** オプションを削除します。次にデーモンを再度読み込み、**docker** サービスを再起動して設定の変更を反映させます。

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

16. **docker** クライアント接続を検証します。レジストリーに対する **docker push**、またはレジストリーからの **docker pull** が正常に実行されるはずですが。必ず **このレジストリーにログインしていること** を確認してください。

```
$ docker tag|push <registry/image> <internal_registry/project/image>
```

以下に例を示します。

```
$ docker pull busybox
$ docker tag docker.io/busybox 172.30.124.220:5000/openshift/busybox
$ docker push 172.30.124.220:5000/openshift/busybox
```

```
...
cf2616975b4a: Image successfully pushed
Digest: sha256:3662dd821983bc4326bee12caec61367e7fb6f6a3ee547cbaff98f77403cab55
```

### 2.4.3. セキュアなレジストリーの手動による公開

OpenShift Container Platform クラスター内から、OpenShift Container Platform レジストリーにログインするのではなく、外部からレジストリーにアクセスできるようにするには、先にレジストリーのセキュリティを確保してから、このレジストリーをルートに公開します。この方法を使うと、ルートアドレスを使ってクラスターの外部からレジストリーにログインし、ルートのホストを使ってイメージにタグ付けしたり、イメージをプッシュしたりできます。

1. 以下の前提条件に関するそれぞれの手順は、標準的なクラスターのインストール時にデフォルトで実行されます。これが実行されていない場合は、手動で実行してください。
  - a. レジストリーを手動でデプロイする。
  - b. レジストリーを手動で保護する。
  - c. ルーターを手動でデプロイする。
2. パススルールート は、クラスターの初回のインストール時にレジストリーについてデフォルトで作成されている必要があります。
  - a. ルートが存在するかどうかを確認します。

```
$ oc get route/docker-registry -o yaml
apiVersion: v1
kind: Route
metadata:
  name: docker-registry
spec:
  host: <host> ❶
  to:
    kind: Service
    name: docker-registry ❷
  tls:
    termination: passthrough ❸
```

- ❶ ルートのホスト。この名前は、外部から DNS 経由でルーターの IP アドレスに解決できる必要があります。
- ❷ レジストリーのサービス名。
- ❸ このルートを通すパススルールートとして指定します。



#### 注記

Re-encrypt ルートはセキュアなレジストリーを公開するためにもサポートされています。

- b. ルートが存在していない場合、**oc create route passthrough** コマンドで、レジストリーをルートのサービスとして指定してルートを作成します。デフォルトでは、作成したルートの名前はサービス名と同じです。

- i. `docker-registry` サービスの詳細を取得します。

```
$ oc get svc
NAME                CLUSTER_IP      EXTERNAL_IP  PORT(S)          SELECTOR
AGE
docker-registry    172.30.69.167  <none>       5000/TCP         docker-
registry=default  4h
kubernetes         172.30.0.1     <none>       443/TCP,53/UDP,53/TCP <none>
4h
router             172.30.172.132 <none>       80/TCP          router=router
4h
```

- ii. ルートを作成します。

```
$ oc create route passthrough \
  --service=docker-registry \ 1
  --hostname=<host>
route "docker-registry" created 2
```

- 1 レジストリーをルートのサービスとして指定します。
- 2 ルート名はサービス名と同じです。

3. 次に、ホストシステム上のレジストリーに使用されている証明書を信頼し、ホストによるイメージのプッシュおよびプルを許可する必要があります。参照される証明書は、レジストリーのセキュリティ保護を実行したときに作成されたものです。

```
$ sudo mkdir -p /etc/docker/certs.d/<host>
$ sudo cp <ca_certificate_file> /etc/docker/certs.d/<host>
$ sudo systemctl restart docker
```

4. レジストリーのセキュリティ保護の実行時に得られた情報を使って [レジストリーにログイン](#) します。ただし、ここではサービス IP ではなくルートで使用されているホスト名を指定します。セキュリティが保護され、公開されているレジストリーにログインする際は、必ず `docker login` コマンドのレジストリーを指定してください。

```
# docker login -e user@company.com \
  -u f83j5h6 \
  -p Ju1PeM47R0B92Lk3AZp-bWJSck2F7aGCiZ66aFGZrs2 \
  <host>
```

5. これでルートのホストを使ってイメージのタグ付けとプッシュを実行できます。たとえば、`test` という名前のプロジェクトにある `busybox` イメージをタグ付けし、プッシュするには、以下を実行します。

```
$ oc get imagestreams -n test
NAME    DOCKER REPO  TAGS    UPDATED

$ docker pull busybox
$ docker tag busybox <host>/test/busybox
$ docker push <host>/test/busybox
The push refers to a repository [<host>/test/busybox] (len: 1)
8c2e06607696: Image already exists
```

```

6ce2e90b0bc7: Image successfully pushed
cf2616975b4a: Image successfully pushed
Digest:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147d8ca31

$ docker pull <host>/test/busybox
latest: Pulling from <host>/test/busybox
cf2616975b4a: Already exists
6ce2e90b0bc7: Already exists
8c2e06607696: Already exists
Digest:
sha256:6c7e676d76921031532d7d9c0394d0da7c2906f4cb4c049904c4031147d8ca31
Status: Image is up to date for <host>/test/busybox:latest

$ oc get imagestreams -n test
NAME      DOCKER REPO          TAGS      UPDATED
busybox   172.30.11.215:5000/test/busybox  latest   2 seconds ago

```



### 注記

イメージストリームにはルート名とポートではなく、レジストリーサービスの IP アドレスとポートが設定されます。詳細は **oc get imagestreams** を参照してください。

#### 2.4.4. 非セキュアなレジストリーを手動で公開する

レジストリーを公開するためにレジストリーのセキュリティーを保護する代わりに、OpenShift Container Platform の非実稼働環境で、非セキュアなレジストリーを簡単に公開できます。これにより、SSL 証明書を使用せずにレジストリーへの外部ルートを作成することができます。



### 警告

非実稼働環境以外では、非セキュアなレジストリーを外部アクセスに公開すべきではありません。

非セキュアなレジストリーを公開するには、以下を実行します。

1. レジストリーを公開します。

```
# oc expose service docker-registry --hostname=<hostname> -n default
```

以下の JSON ファイルが作成されます。

```

apiVersion: v1
kind: Route
metadata:
  creationTimestamp: null
  labels:
    docker-registry: default

```

```

name: docker-registry
spec:
  host: registry.example.com
  port:
    targetPort: "5000"
  to:
    kind: Service
    name: docker-registry
  status: {}

```

2. ルートが正常に作成されたことを確認します。

```

# oc get route
NAME          HOST/PORT          PATH    SERVICE    LABELS
INSECURE POLICY  TLS TERMINATION
docker-registry registry.example.com    docker-registry  docker-registry=default

```

3. レジストリーの健全性を確認します。

```
$ curl -v http://registry.example.com/healthz
```

HTTP 200 / OK メッセージが表示されるはずですが、

レジストリーを公開したら、ポート番号を **OPTIONS** エントリーに追加して `/etc/sysconfig/docker` ファイルを更新します。以下に例を示します。

```

OPTIONS='--selinux-enabled --insecure-registry=172.30.0.0/16 --insecure-registry
registry.example.com:80'

```



### 重要

上記のオプションは、ログインしようとしているクライアントに追加してください。

また、Docker がクライアント上で実行されていることも確認してください。

公開されている非セキュアなレジストリーに [ログイン](#) する際に、**docker login** コマンドでレジストリーを指定していることを確認します。以下に例を示します。

```

# docker login -e user@company.com \
-u f83j5h6 \
-p Ju1PeM47R0B92Lk3AZp-bWJSck2F7aGCiZ66aFGZrs2 \
<host>

```

## 2.5. レジストリー設定の拡張

### 2.5.1. レジストリー IP アドレスの維持

OpenShift Container Platform はサービス IP アドレスによって統合レジストリーを参照します。したがって **docker-registry** サービスを削除し、再作成しようとする場合、古い IP アドレスを新しいサービスで再利用するように調整すると、完全に透過的な移行が可能になります。新規 IP アドレスを取得

することが避けられない場合は、マスターのみを再起動してクラスターの中断を最小限に抑えられます。

### アドレスの再利用

IP アドレスを再利用するには、古い **docker-registry** サービスの IP アドレスを削除する前に保存し、新たに割り当てられた IP アドレスを、新しい **docker-registry** サービスに保存されているアドレスに置き換えるように調整します。

1. サービスの **clusterIP** を書き留めておきます。

```
$ oc get svc/docker-registry -o yaml | grep clusterIP:
```

2. サービスを削除します。

```
$ oc delete svc/docker-registry dc/docker-registry
```

3. レジストリーの定義を **registry.yaml** に作成し、**<options>** を、たとえば **非実稼働環境での使用** のセクションの手順 3 で使用されているものなどに置き換えます。

```
$ oc adm registry <options> -o yaml > registry.yaml
```

4. **registry.yaml** を編集し、そこで **Service** を検索して、**clusterIP** を手順 1 で書き留めたアドレスに変更します。

5. 変更した **registry.yaml** を使ってレジストリーを作成します。

```
$ oc create -f registry.yaml
```

### マスターの再起動

IP アドレスを再利用できない場合、古い IP アドレスを含む **プル仕様** を使った操作は失敗します。クラスターの中断を最小限に抑えるには、マスターを再起動する必要があります。

```
# master-restart api
# master-restart controllers
```

これで、古い IP アドレスを含む古いレジストリー URL がキャッシュからクリアされます。



#### 注記

クラスター全体を再起動すると、Pod に不要なダウンタイムが発生し、実質、キャッシュのクリアが行われないので、これは推奨していません。

### 2.5.2. 外部レジストリーの検索一覧の設定

**/etc/containers/registries.conf** ファイルを使用して、コンテナイメージを検索するための Docker レジストリーの一覧を作成できます。

**/etc/containers/registries.conf** ファイルは、レジストリーサーバーの一覧で、ユーザーが **myimage:latest** などのイメージの短い名前を使用してイメージをプルするときに OpenShift Container Platform はこの一覧を検索する必要があります。検索の順序をカスタマイズしたり、安全なレジストリーと安全でないレジストリーを指定したり、ブロックするレジストリーリストを定義したりできます。OpenShift Container Platform は、ブロックするリストのレジストリーからプルを検索したり、許可したりしません。

たとえば、ユーザーが **myimage:latest** イメージをプルする必要がある場合、OpenShift Container Platform は **myimage:latest** が見つかるまで一覧に表示される順序でレジストリーを検索します。

レジストリー検索の一覧を指定すると、OpenShift Container Platform ユーザーがダウンロードできるイメージとテンプレートのセットをキュレートできます。これらのイメージを1つ以上の Docker レジストリーに配置し、レジストリーを一覧に追加して、それらのイメージをクラスター内にプルできます。



### 注記

レジストリーの検索一覧を使用する場合、OpenShift Container Platform は検索一覧にないレジストリーからイメージをプルしません。

レジストリーの検索一覧を設定するには、以下を実行します。

1. `/etc/containers/registries.conf` ファイルを編集し、必要に応じて以下のパラメーターを追加または編集します。

```
[registries.search] 1
registries = ["reg1.example.com", "reg2.example.com"]

[registries.insecure] 2
registries = ["reg3.example.com"]

[registries.block] 3
registries = ['docker.io']
```

- 1 ユーザーが SSL/TLS を使用してイメージをダウンロードできるセキュアなレジストリーを指定します。
- 2 ユーザーが TLS を使用せずにイメージをダウンロードできる、セキュアではないレジストリーを指定します。
- 3 ユーザーがイメージをダウンロードできないレジストリーを指定します。

### 2.5.3. レジストリーホスト名の設定

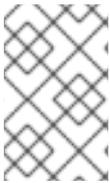
内部参照と外部参照の両方でレジストリーを認識するために使用するホスト名およびポートを設定できます。これを実行すると、イメージストリームはイメージのホスト名ベースのプッシュおよびプル仕様を提供することができ、これによってイメージのコンシューマーがレジストリーのサービス IP の変更の影響を受けないようにし、イメージストリームとその参照をクラスター間で移植できる可能性があります。

クラスター内からレジストリーを参照するために使用されるホスト名を設定するには、マスター設定ファイルの **imagePolicyConfig** セクションで **internalRegistryHostname** を設定します。外部のホスト名は、同じ場所で **externalRegistryHostname** の値を設定することで管理されます。

### イメージポリシーの設定

```
imagePolicyConfig:
  internalRegistryHostname: docker-registry.default.svc.cluster.local:5000
  externalRegistryHostname: docker-registry.mycompany.com
```

レジストリー自体は、同じ内部ホスト名の値で設定する必要があります。これは、レジストリーのデプロイメント設定で **REGISTRY\_OPENSIFT\_SERVER\_ADDR** 環境変数を設定するか、またはレジストリー設定の [OpenShift セクション](#) で値を設定することで実行できます。



### 注記

レジストリーで TLS を有効にしている場合、サーバー証明書にはレジストリーの参照に使用されるホスト名が含まれている必要があります。ホスト名をサーバー証明書に追加する方法については、[レジストリーのセキュリティー保護](#) を参照してください。

## 2.5.4. レジストリー設定の上書き

統合レジストリーのデフォルトの設定 (デフォルトでは実行中のレジストリーコンテナの `/config.yml` にあります) は、独自の [カスタム設定](#) で上書きできます。



### 注記

このファイルのアップストリームの設定オプションも環境変数を使って上書きできます。[ミドルウェアのセクション](#) は、環境変数を使って上書きできるオプションがごく少数であるため例外とします。[特定の設定オプションを上書きする方法についてこちら](#) を参照してください。

レジストリー設定ファイルの直接管理を有効にし、[ConfigMap](#) を使用して更新された設定をデプロイするには、以下を実行します。

1. [レジストリーをデプロイします](#)。
2. 必要に応じて、レジストリー設定ファイルをローカルで編集します。以下は、レジストリーにデプロイされている初期 YAML ファイルです。[サポートされているオプション](#) を確認してください。

### レジストリー設定ファイル

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /registry
delete:
  enabled: true
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
repository:
  - name: openshift
options:
```

```

acceptschema2: true
pullthrough: true
enforcequota: false
projectcachettl: 1m
blobrepositorycachettl: 10m
storage:
  - name: openshift
openshift:
  version: 1.0
metrics:
  enabled: false
secret: <secret>

```

3. 各ファイルの内容を保持する **ConfigMap** をこのディレクトリーに作成します。

```

$ oc create configmap registry-config \
  --from-file=</path/to/custom/registry/config.yml>/

```

4. **registry-config** ConfigMap をボリュームとしてレジストリーのデプロイメント設定に追加し、カスタム設定ファイルを **/etc/docker/registry/** にマウントします。

```

$ oc set volume dc/docker-registry --add --type=configmap \
  --configmap-name=registry-config -m /etc/docker/registry/

```

5. 以下の環境変数をレジストリーのデプロイメント設定に追加して、直前の手順の設定パスを参照するようにレジストリーを更新します。

```

$ oc set env dc/docker-registry \
  REGISTRY_CONFIGURATION_PATH=/etc/docker/registry/config.yml

```

上記の手順は、必要な設定を行えるように繰り返し実行される場合があります。たとえば、トラブルシューティング時に、**デバッグ** モードに切り換えるよう設定が一時的に更新される場合があります。

既存の設定を更新するには、以下を実行します。



### 警告

この手順を実行すると、現在デプロイされているレジストリー設定が上書きされます。

1. ローカルのレジストリー設定ファイル **config.yml** を編集します。
2. **registry-config** configmap を削除します。

```

$ oc delete configmap registry-config

```

3. 更新された設定ファイルを参照するよう configmap を再作成します。

```
$ oc create configmap registry-config\
  --from-file=</path/to/custom/registry/config.yml>/
```

4. 更新された設定を読み取るためにレジストリーを再デプロイします。

```
$ oc rollout latest docker-registry
```

## ヒント

設定ファイルをソース管理リポジトリに保持します。

### 2.5.5. レジストリー設定の参照

アップストリームの [docker ディストリビューション](#) ライブラリーでは、多数の設定オプションを利用できます。ただし、すべての [設定オプション](#) がサポートされているか、または有効にされているわけではありません。このセクションは、[レジストリー設定を上書きする](#) 際の参考としてご利用ください。



#### 注記

このファイルのアップストリームの設定オプションも環境変数を使って上書きできます。ただし、[ミドルウェアのセクション](#) は、環境変数を使って上書きすることはできません。[特定の設定オプションを上書きする方法についてこちら](#) を参照してください。

#### 2.5.5.1. Log

[アップストリームのオプション](#) はサポートされています。

以下に例を示します。

```
log:
  level: debug
  formatter: text
  fields:
    service: registry
    environment: staging
```

#### 2.5.5.2. フック

メールフックはサポートされていません。

#### 2.5.5.3. ストレージ

このセクションでは、サポートされているレジストリーのストレージドライバーの一覧を示します。詳細は、[コンテナイメージレジストリーのドキュメント](#) を参照してください。

以下の一覧には、レジストリーの設定ファイルで設定する必要のあるストレージドライバーが含まれません。

- [ファイルシステム](#)。ファイルシステムはデフォルトですので、設定する必要はありません。
- [S3](#)。詳細は、[CloudFront の設定](#) のドキュメントを参照してください。
- [OpenStack Swift](#)

- [Google Cloud Storage \(GCS\)](#)
- [Microsoft Azure](#)
- [Aliyun OSS](#)

レジストリーの一般的なストレージ設定オプションはサポートされています。詳細は、[コンテナイメージレジストリーのドキュメント](#) を参照してください。

以下のストレージオプションは、[ファイルシステムドライバー](#) で設定する必要があります。

- [GlusterFS ストレージ](#)
- [Ceph Rados ブロックデバイス](#)



### 注記

サポートされている永続ストレージドライバーの詳細は、[永続ストレージの設定](#) および [永続ストレージの例](#) を参照してください。

## 一般的なストレージ設定オプション

```
storage:
  delete:
    enabled: true 1
  redirect:
    disable: false
  cache:
    blobdescriptor: inmemory
  maintenance:
    uploadpurging:
      enabled: true
      age: 168h
      interval: 24h
      dryrun: false
  readonly:
    enabled: false
```

**1** このエントリーは、イメージのプルーニングが正常に機能するために**必須**です。

### 2.5.5.4. 認証

認証オプションは変更することができません。`openshift` の拡張がサポートされている唯一のオプションです。

```
auth:
  openshift:
    realm: openshift
```

### 2.5.5.5. ミドルウェア

リポジトリーのみドルウェアの拡張を使用して、OpenShift Container Platform やイメージプロキシとの対話を行う OpenShift Container Platform ミドルウェアの設定を行うことができます。

```

middleware:
  registry:
    - name: openshift ①
  repository:
    - name: openshift ②
    options:
      acceptschema2: true ③
      pullthrough: true ④
      mirrorpullthrough: true ⑤
      enforcequota: false ⑥
      projectcachettl: 1m ⑦
      blobrepositorycachettl: 10m ⑧
  storage:
    - name: openshift ⑨

```

① ② ⑨ これらの入力は必須です。これらの入力によって必要なコンポーネントが読み込まれていることを確認できます。これらの値は変更しないでください。

③ レジストリーへのプッシュ時に [manifest schema v2](#) を格納できます。詳細は、[こちら](#) を参照してください。

④ レジストリーがリモート Blob のプロキシとして機能できるようにします。詳細は、[こちら](#) を参照してください。

⑤ レジストリーキャッシュの Blob がリモートレジストリーから提供されるようにし、その後の迅速なアクセスを可能にします。Blob の初回アクセス時にミラーリングが開始されます。このオプションは、[プルスルー](#) が無効にされている場合は機能しません。

⑥ ターゲットに設定されているプロジェクトで定義されているサイズ制限を超える Blob のアップロードを防止します。

⑦ レジストリーにキャッシュされている制限の有効期限のタイムアウト。値が小さいほど、制限の変更がレジストリーに伝播するまでの時間が短くなります。ただし、レジストリーは制限をサーバーからより頻繁に照会するため、結果としてプッシュは遅くなります。

⑧ Blob とリポジトリ間の記憶されている関連付けの有効期限のタイムアウト。値が高いほどルックアップが高速になり、レジストリー操作がより効率的になる可能性が高くなります。一方、アクセスできなくなったユーザーにイメージレイヤーを提供するリスクと同様にメモリー使用量も上昇します。

#### 2.5.5.5.1. S3 ドライバー設定

使用している統合レジストリーでサポートされていない S3 リージョンを使用する必要がある場合は、**regionendpoint** を指定して **region** 検証エラーを防ぐことができます。

Amazon Simple Storage Service ストレージの使用についての詳細は、[ストレージバックエンドとしての Amazon S3](#) を参照してください。

以下に例を示します。

```

version: 0.1
log:
  level: debug
http:

```

```

addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  delete:
    enabled: true
s3:
  accesskey: BJKMSZBRESWJQXRWMAEQ
  secretkey: 5ah5I91SNXbeoUXDasFtadRqOdy62JzlnOW1goS
  bucket: docker.myregistry.com
  region: eu-west-3
  regionendpoint: https://s3.eu-west-3.amazonaws.com
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
storage:
  - name: openshift

```



### 注記

**region** および **regionendpoint** フィールドの間に整合性があることを確認します。そうでない場合、統合レジストリーは開始されますが、S3 ストレージに対する読み取りまたは書き込みを行うことはできません。

また、Amazon S3 とは異なる S3 ストレージを使用する場合に、**regionendpoint** が役に立ちます。

#### 2.5.5.5.2. CloudFront ミドルウェア

**CloudFront ミドルウェア拡張**は、CloudFront CDN ストレージプロバイダーである AWS をサポートするために追加することができます。CloudFront ミドルウェアは、イメージコンテンツの海外への配信を高速化します。Blob は世界中の複数のエッジロケーションに配信されます。クライアントは常に最小の待機時間でエッジにアクセスできます。



### 注記

**CloudFront ミドルウェア拡張**を使用できるストレージは **S3** ストレージのみです。また、使用できるのは Blob が提供されている間のみです。したがって、高速化できるのは Blob のダウンロードのみで、アップロードは高速化しません。

以下は、CloudFront ミドルウェアを含む S3 ストレージドライバーの最小限の設定例です。

```

version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory

```

```

delete:
  enabled: true
s3: ❶
  accesskey: BJKMSZBRESWJQXRWMAEQ
  secretkey: 5ah5I91SNXbeoUXXDasFtadRqOdy62JzlnOW1goS
  region: us-east-1
  bucket: docker.myregistry.com
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
storage:
  - name: cloudfront ❷
    options:
      baseurl: https://jrpbyn0k5k88bi.cloudfront.net/ ❸
      privatekey: /etc/docker/cloudfront-ABCDEFGHJKLMNOPQRST.pem ❹
      keypairid: ABCDEFGHIJKLMNOPQRST ❺
    - name: openshift

```

- ❶ S3 ストレージは、CloudFront ミドルウェアに関係なく同じ方法で設定する必要があります。
- ❷ CloudFront ストレージミドルウェアは、OpenShift ミドルウェアより前に一覧表示される必要があります。
- ❸ CloudFront ベースの URL。AWS マネジメントコンソールでは、これは CloudFront ディストリビューションの **Domain Name** として一覧表示されます。
- ❹ AWS のプライベートキーが格納されているファイルシステムの場所。Amazon EC2 のキーペアと混同しないよう注意してください。信頼される署名者の CloudFront キーペアの作成については、[AWS ドキュメント](#) を参照してください。このファイルは、レジストリー Pod に **シークレット** としてマウントする必要があります。
- ❺ Cloudfront キーペアの ID。

### 2.5.5.5.3. ミドルウェア設定オプションの上書き

`middleware` セクションは、環境変数を使って上書きできません。ただし例外がいくつかあります。以下に例を示します。

```

middleware:
  repository:
    - name: openshift
    options:
      acceptschema2: true ❶
      pullthrough: true ❷
      mirrorpullthrough: true ❸
      enforcequota: false ❹
      projectcachettl: 1m ❺
      blobrepositorycachettl: 10m ❻

```

- 1 ブール環境変数 **REGISTRY\_MIDDLEWARE\_REPOSITORY\_OPENSHIFT\_ACCEPTSCHEMA2** で上書きできる設定オプション。manifest schema v2 をマニフェストの Put 要求で受け入れる機能を有効にします。認識される値は **true** と **false** (以下の他のすべてのブール変数に適用されます) になります。
- 2 ブール環境変数 **REGISTRY\_MIDDLEWARE\_REPOSITORY\_OPENSHIFT\_PULLTHROUGH** で上書きできる設定オプション。リモートレジストリーのプロキシモードを有効にします。
- 3 ブール環境変数 **REGISTRY\_MIDDLEWARE\_REPOSITORY\_OPENSHIFT\_MIRRORPULLTHROUGH** で上書きできる設定オプション。リモート Blob が提供されている場合、レジストリーに対して Blob をローカルにミラーリングするように指示します。
- 4 ブール環境変数 **REGISTRY\_MIDDLEWARE\_REPOSITORY\_OPENSHIFT\_ENFORCEQUOTA** で上書きできる設定オプション。クォータ実施をオンまたはオフにする機能を有効にします。デフォルトでは、クォータの実施はオフになっています。
- 5 環境変数 **REGISTRY\_MIDDLEWARE\_REPOSITORY\_OPENSHIFT\_PROJECTCACHETTTL** で上書きできる設定オプション。有効な時間文字列 (例: **2m**) を取ります。空白の場合は、デフォルトのタイムアウトが取得されます。ゼロ (**0m**) の場合、キャッシングは無効にされます。
- 6 環境変数 **REGISTRY\_MIDDLEWARE\_REPOSITORY\_OPENSHIFT\_BLOBREPOSITORYCACHETTTL** で上書きできる設定オプション。Blob と含まれているレポジトリーの関連付けについてのエビクシオンタイムアウトを指定します。値のフォーマットは **projectcachettl** のケースと同じです。

#### 2.5.5.5.4. イメージのプルスルー

この機能を有効にすると、Blob がローカルに存在しない限り、レジストリーは要求された Blob をリモートレジストリーから取得しようと試みます。リモートの候補は、クライアントがプルするイメージストリームの状態で保存された [DockerImage](#) エントリーから算出されます。このエントリーのすべての固有のリモートレジストリーの参照は Blob が見つかるまで繰り返し試行されます。

プルスルーは、プルされているイメージのイメージストリームタグが存在する場合にのみ発生します。たとえば、プルされているイメージが **docker-registry.default.svc:5000/yourproject/yourimage:prod** である場合、レジストリーは、プロジェクト **yourproject** で **yourimage:prod** という名前のイメージストリームタグを検索します。タグが見つかったら、レジストリーはそのイメージストリームタグに関連付けられた **dockerImageReference** を使ってイメージのプルを試行します。

プルスルーを実行すると、レジストリーは、参照されているイメージストリームタグに関連付けられたプロジェクトにあるプル認証情報を使用します。また、この機能を使用すると、ユーザーは、イメージを参照しているイメージストリームタグにアクセスできる限り、アクセスに必要な認証情報を持たないレジストリーのイメージをプルすることができます。

使用しているレジストリーに、プルスルーの実行対象である外部レジストリーを信頼するのに必要な証明書があることを確認してください。証明書は Pod の **/etc/pki/tls/certs** ディレクトリーに配置する必要があります。証明書は [設定マップ](#) または [シークレット](#) を使ってマウントできます。ここで、**/etc/pki/tls/certs** ディレクトリー全体を置き換える必要があることに注意してください。新しい証明書を組み込み、マウントするシークレットまたは設定マップのシステム証明書を置き換えます。

デフォルトでは、イメージストリームタグは **Source** の参照ポリシータイプを使用することに注意してください。これは、イメージストリームの参照がイメージのプル仕様に対して解決される場合、使用される仕様はイメージのソースを参照することを意味します。外部レジストリーでホストされているイメージであれば、外部レジストリーが参照され、この結果としてリソースは外部レジストリーでイメージを参照し、プルします。たとえば、この場合、**registry.redhat.io/openshift3/jenkins-2-rhel7** とプル

スルーは適用されません。イメージストリームを参照しているリソースが内部レジストリーを参照しているプル仕様を使用するようにするには、イメージストリームタグは **Local** の参照ポリシータイプを使用する必要があります。詳細は、[参照ポリシー](#) を参照してください。

この機能はデフォルトでオンになっています。ただし、[設定オプション](#) を使って無効にすることができます。

デフォルトでは、この方法で提供されるすべてのリモート Blob は、**mirrorpullthrough** が無効にされていない限りローカルに格納され、その後のアクセスが高速になります。ミラーリング機能の欠点はストレージの使用量が増えることにあります。



### 注記

ミラーリングは、クライアントが Blob を 1 バイト以上取得しようとする際に開始されません。実際に必要になる前に、特定イメージを統合レジストリーに事前にフェッチするには、以下のコマンドを実行します。

```
$ oc get imagestreamtag/${IS}:${TAG} -o jsonpath='{
.image.dockerImageLayers[*].name }' | \
xargs -n1 -l {} curl -H "Range: bytes=0-1" -u user:${TOKEN} \
http://${REGISTRY_IP}:${PORT}/v2/default/mysql/blobs/{}'
```



### 注記

OpenShift Container Platform のミラーリング機能をアップストリームの [レジストリーのプルスルーキャッシュ機能](#) と混同しないようにしてください。これらは似ていますが、全く異なる機能です。

#### 2.5.5.5. Manifest Schema v2 サポート

各イメージには、Blob を記述するマニフェスト、それを実行するための命令、および追加のメタデータがあります。マニフェストはバージョン管理されており、バージョンごとに構造やフィールドが異なります。同一イメージが複数のマニフェストバージョンで表現されます。それぞれのバージョンはそれぞれ異なるダイジェストがあります。

現在サポートされているレジストリーは [manifest v2 schema 1 \(schema1\)](#) と [manifest v2 schema 2 \(schema2\)](#) です。前者は古くなっていますが、今後もしばらくはサポートされます。

デフォルト設定は **schema2** で保持されます。

各種の Docker クライアントとの互換性の問題に注意する必要があります。

- Docker クライアントのバージョン 1.9 以前は、**schema1** のみをサポートしています。このクライアントがプルまたはプッシュするマニフェストはレガシースキーマになります。
- Docker クライアントのバージョン 1.10 は、**schema1** と **schema2** の両方をサポートしています。デフォルトでは、新しい方のスキーマをサポートする場合はこちらをレジストリーにプッシュします。

イメージを **schema1** で格納するレジストリーは、常にイメージを変更せずにクライアントに返します。**Schema2** は新規の Docker クライアントにのみ変更しない状態で移動します。古いクライアントの場合は、オンザフライで **schema1** に変換されます。

これにより、大きな影響が想定されます。たとえば、新規 Docker クライアントでレジストリーにプッシュされたイメージは、古い Docker のダイジェストでプルすることはできません。格納されたイメー

ジのmanifestは `schema2` であり、そのダイジェストはこのバージョンのmanifestをプルする場合にのみ使用できるためです。

すべてのレジストリークライアントが `schema2` をサポートしていることを確認できたら、そのサポートをレジストリーで有効にすることができます。特定のオプションについては、上記の [ミドルウェア設定の参照](#) を参照してください。

### 2.5.5.6. OpenShift

このセクションでは、OpenShift Container Platform に特有の機能のグローバル設定について説明します。今後のリリースでは、**Middleware** セクションにある `openshift` 関連の設定は非推奨になる予定です。

現在、このセクションではレジストリーメトリクスの収集を設定できます。

```
openshift:
  version: 1.0 ①
  server:
    addr: docker-registry.default.svc ②
  metrics:
    enabled: false ③
    secret: <secret> ④
  requests:
    read:
      maxrunning: 10 ⑤
      maxinqueue: 10 ⑥
      maxwaitinqueue 2m ⑦
    write:
      maxrunning: 10 ⑧
      maxinqueue: 10 ⑨
      maxwaitinqueue 2m ⑩
```

- ① このセクションの設定バージョンを指定する必須エントリ。サポートされている値は **1.0** のみです。
- ② レジストリーのホスト名。マスターで設定されている値と同じ値に設定される必要があります。これは環境変数 `REGISTRY_OPENSIFT_SERVER_ADDR` で上書きされる可能性があります。
- ③ メトリクスの収集を有効にするには `true` に設定します。これはブール環境変数 `REGISTRY_OPENSIFT_METRICS_ENABLED` で上書きされる可能性があります。
- ④ クライアント要求の承認に使用されるシークレット。メトリクスのクライアントはこれを `Authorization` ヘッダーでベアラートークンとして使用します。環境変数 `REGISTRY_OPENSIFT_METRICS_SECRET` で上書きできます。
- ⑤ 同時に行えるプル要求の最大数。環境変数 `REGISTRY_OPENSIFT_REQUESTS_READ_MAXRUNNING` で上書きできます。ゼロは無制限を意味します。
- ⑥ キューに入れられるプル要求の最大数。環境変数 `REGISTRY_OPENSIFT_REQUESTS_READ_MAXINQUEUE` で上書きできます。ゼロは無制限を意味します。
- ⑦ 拒否されるまでのキューにあるプル要求の最大待機時間。環境変数 `REGISTRY_OPENSIFT_REQUESTS_READ_MAXWAITINQUEUE` で上書きできます。ゼロは無

制限を意味します。

- 8 同時に行えるプッシュ要求の最大数。環境変数 **REGISTRY\_OPENSIFT\_REQUESTS\_WRITE\_MAXRUNNING** で上書きできます。ゼロは無制限を意味します。
- 9 キューにあるプッシュ要求の最大数。環境変数 **REGISTRY\_OPENSIFT\_REQUESTS\_WRITE\_MAXINQUEUE** で上書きできます。ゼロは無制限を意味します。
- 10 拒否されるまでのキューにあるプッシュ要求の最大待機時間。環境変数 **REGISTRY\_OPENSIFT\_REQUESTS\_WRITE\_MAXWAITINQUEUE** で上書きできます。ゼロは無制限を意味します。

使用状況の情報については [レジストリーメトリクスへのアクセス](#) を参照してください。

### 2.5.5.7. レポート

レポート (Reporting) はサポートされていません。

### 2.5.5.8. HTTP

[アップストリームのオプション](#) はサポートされています。環境変数を使って設定を変更する方法については [こちら](#) を参照してください。変更の必要があるのは `tls` セクションのみです。以下に例を示します。

```
http:  
  addr: :5000  
tls:  
  certificate: /etc/secrets/registry.crt  
  key: /etc/secrets/registry.key
```

### 2.5.5.9. 通知

[アップストリームのオプション](#) はサポートされています。REST API リファレンス [はより包括的な統合オプションを提供します。](#)

以下に例を示します。

```
notifications:  
  endpoints:  
    - name: registry  
      disabled: false  
      url: https://url:port/path  
  headers:  
    Accept:  
      - text/plain  
  timeout: 500  
  threshold: 5  
  backoff: 1000
```

### 2.5.5.10. Redis

Redis はサポートされていません。

### 2.5.5.11. Health

[アップストリームのオプション](#) はサポートされています。レジストリーのデプロイメント設定は、`/healthz` で統合されたヘルスチェックを提供します。

### 2.5.5.12. Proxy

プロキシ設定は有効にできません。この機能は [OpenShift Container Platform リポジトリー](#) の [ミドルウェア拡張](#)、`pullthrough: true` で提供されます。

### 2.5.5.13. Cache

統合レジストリーは、データをアクティブにキャッシュして、速度の遅い外部リソースに対する呼び出しの回数を減らします。キャッシュには 2 種類あります。

1. Blob のメタデータのキャッシュに使用されるストレージキャッシュ。このキャッシュには有効期限がなく、データは明示的に削除されるまで残り続けます。
2. アプリケーションキャッシュには、Blob とリポジトリーの関連付けが含まれます。このキャッシュ内のデータには有効期限があります。

キャッシュを完全にオフにするには設定を変更する必要があります。

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: "" ❶
openshift:
  version: 1.0
  cache:
    disabled: true ❷
  blobrepositoryttl: 10m
```

- ❶ ストレージのバックエンドでアクセスしたメタデータのキャッシュを無効にします。このキャッシュがない場合、レジストリーサーバーはメタデータのバックエンドに絶えずアクセスします。
- ❷ Blob とリポジトリーの関連付けを含むキャッシュを無効にします。このキャッシュがない場合、レジストリーサーバーは継続的にマスター API のデータを照会し、関連付けを再計算します。

## 2.6. 既知の問題

### 2.6.1. 概要

以下は、統合レジストリーのデプロイまたは使用時の既知の問題です。

### 2.6.2. レジストリーのプルスルーに伴う同時ビルド

ローカルの `docker-registry` デプロイメントは追加の負荷を受けます。デフォルトで、ここでは `registry.redhat.io` からのコンテンツをキャッシュするようになりました。STI ビルドの `registry.redhat.io` のイメージはローカルレジストリーに保存されます。それらをプルしようとする、ローカル `docker-registry` からのプルが試行され、その結果として、過剰な数の同時ビルドがプルでタイムアウトになり、ビルドが失敗する可能性のある状況になります。この問題を軽減するには、`docker-registry` デプロイメントを複数のレプリカにスケーリングします。Pod のログでタイムアウトの有無をチェックします。

### 2.6.3. 共有 NFS ボリュームとスケーリングされたレジストリーの使用時のイメージのプッシュエラー

スケーリングされたレジストリーを共有 NFS ボリュームで使用すると、イメージのプッシュ時に以下のいずれかのエラーが発生することがあります。

- **digest invalid: provided digest did not match uploaded content**
- **blob upload unknown**
- **blob upload invalid**

これらのエラーは、Docker のイメージのプッシュの試行時に内部レジストリーサービスによって返されます。その原因は、ノード間のファイル属性の同期に起因します。NFS クライアント側のキャッシング、ネットワーク待機時間およびレイヤーサイズなどはすべて、デフォルトのラウンドロビンロードバランシング設定を使用してイメージをプッシュする際に発生するエラーの要因になる可能性があります。

このような障害の可能性を最小限に抑えるには、以下の手順を実行します。

1. `docker-registry` サービスの **sessionAffinity** が **ClientIP** に設定されていることを確認します。

```
$ oc get svc/docker-registry --template='{{.spec.sessionAffinity}}'
```

これにより **ClientIP** が返されるはずです。ClientIP は OpenShift Container Platform の最近のバージョンのデフォルトです。これが返されない場合は、以下のように変更してください。

```
$ oc patch svc/docker-registry -p '{"spec":{"sessionAffinity": "ClientIP"}}'
```

2. NFS サーバー上のレジストリーボリュームの NFS エクスポート行に **no\_wdelay** オプションが一覧表示されていることを確認します。**no\_wdelay** オプションは、サーバーによる書き込みの遅延を防ぎ、このレジストリーの要件である、書き込み直後の読み取りの整合性を大幅に改善します。

#### 重要

テストにより、RHEL NFS サーバーをコンテナイメージレジストリーのストレージバックエンドとして使用することに関する問題が検出されています。これには、OpenShift Container レジストリーおよび Quay が含まれます。そのため、コアサービスで使用される PV をサポートするために RHEL NFS サーバーを使用することは推奨されていません。

他の NFS の実装ではこれらの問題が検出されない可能性があります。OpenShift コアコンポーネントに対して実施された可能性のあるテストに関する詳細情報は、個別の NFS 実装ベンダーにお問い合わせください。

## 2.6.4. 内部で管理されたイメージのプルに失敗し見つかりません (not found) のエラーが表示される

このエラーは、プルされたイメージがプルしたイメージストリームとは異なるイメージストリームにプッシュされた場合に発生します。これは、ビルドされたイメージを任意のイメージストリームに再タグ付けすることによって発生します。

```
$ oc tag srcimagestream:latest anyproject/pullimagestream:latest
```

その後、以下のようなイメージ参照を使用してプルします。

```
internal.registry.url:5000/anyproject/pullimagestream:latest
```

Docker を手動でプルするときにも同様のエラーが発生します。

```
Error: image anyproject/pullimagestream:latest not found
```

このエラーを防ぐには、内部で管理されたイメージのタグ付けを完全に避けるか、またはビルドしたイメージを必要な namespace に [手動で](#) 再プッシュします。

## 2.6.5. S3 ストレージでのイメージのプッシュが失敗し 500 内部サーバーエラー (500 Internal Server Error) と表示される

レジストリーが S3 ストレージのバックエンドで実行されると、問題が報告されます。コンテナイメージレジストリーへのプッシュは、以下のエラーを出して失敗することがあります。

```
Received unexpected HTTP status: 500 Internal Server Error
```

これをデバッグするには、[レジストリーのログを表示](#) する必要があります。ログで、プッシュの失敗時に発生した同様のエラーメッセージを探します。

```
time="2016-03-30T15:01:21.22287816-04:00" level=error msg="unknown error completing upload: driver.Error{DriverName:\"s3\", Enclosed:(*url.Error)(0xc20901cea0)}" http.request.method=PUT
...
time="2016-03-30T15:01:21.493067808-04:00" level=error msg="response completed with error" err.code=UNKNOWN err.detail="s3: Put https://s3.amazonaws.com/oso-tsi-docker/registry/docker/registry/v2/blobs/sha256/ab/abe5af443833d60cf672e2ac57589410dddec060ed725d3e676f1865af63d2e2/data: EOF" err.message="unknown error" http.request.method=PUT
...
time="2016-04-02T07:01:46.056520049-04:00" level=error msg="error putting into main store: s3: The request signature we calculated does not match the signature you provided. Check your key and signing method." http.request.method=PUT
atest
```

このようなエラーを確認した場合には、Amazon S3 サポートにお問い合わせください。お住まいの地域や特定のバケットに関連した問題がある可能性があります。

## 2.6.6. イメージのプルーニングの失敗

イメージのプルーニング時に以下のエラーが発生した場合:

```
BLOB sha256:49638d540b2b62f3b01c388e9d8134c55493b1fa659ed84e97cb59b87a6b8e6c error  
deleting blob
```

さらに、[レジストリーのログ](#) に以下の情報が含まれている場合。

```
error deleting blob  
\"sha256:49638d540b2b62f3b01c388e9d8134c55493b1fa659ed84e97cb59b87a6b8e6c\": operation  
unsupported
```

上記に該当する場合、お使いの [カスタム設定ファイル](#) には [ストレージセクション](#) に必須のエントリー、つまり `true` に設定された `storage:delete:enabled` が含まれていないことを意味します。これらを追加し、レジストリーを再デプロイして、再度イメージプルーニングの操作を行ってください。

## 第3章 ルーターのセットアップ

### 3.1. ルーターの概要

#### 3.1.1. ルーターについて

[トラフィックをクラスターに送る](#) 方法は多数あります。最も一般的な方法として、OpenShift Container Platform インストールで OpenShift Container Platform [ルーター](#) を、[サービス](#) 向けの外部トラフィックの ingress ポイントとして使用できます。

OpenShift Container Platform は以下のルータープラグインを提供し、サポートしています。

- [HAProxy テンプレートルーター](#) はデフォルトのプラグインです。これは、`openshift3/ose-haproxy-router` イメージを使用して、OpenShift Container Platform のコンテナにあるテンプレートルータープラグインと共に HAProxy インスタンスを実行します。現在は、HTTP(S) トラフィックと SNI を使用した TLS 対応のトラフィックをサポートしています。ルーターのコンテナは、プライベート IP でのみリッスンする多数のコンテナとは異なり、ホストのネットワークインターフェイスでリッスンします。ルーターは、ルートに関連付けられたサービスで識別される実際の Pod の IP に対するルート名の外部要求をプロキシ処理します。
- [F5 ルーター](#) は、ルートを同期するためにお使いの環境で既存の **F5 BIG-IP®** システムに統合されます。F5 iControl REST API を使用するには、**F5 BIG-IP®** バージョン 11.4 以降が必要です。
- [デフォルトの HAProxy ルーターのデプロイ](#)
- [カスタム HAProxy ルーターのデプロイ](#)
- [PROXY プロトコルを使用するように HAProxy ルーターを設定する](#)
- [ルートのタイムアウトの設定](#)

#### 3.1.2. ルーターのサービスアカウント

OpenShift Container Platform クラスターをデプロイする前に、ルーターの [サービスアカウント](#) を用意しておく必要があります。これには、クラスターのインストール時に自動的に作成されます。このサービスアカウントには、ホストのポートの指定に使用できる [SCC \(security context constraints\)](#) へのパーミッションがあります。

##### 3.1.2.1. ラベルにアクセスするためのパーミッション

[namespace ラベル](#) が使用されている場合 ([ルーターシャード](#) を作成している場合など)、ルーターのサービスアカウントには `cluster-reader` のパーミッションが必要になります。

```
$ oc adm policy add-cluster-role-to-user \
  cluster-reader \
  system:serviceaccount:default:router
```

サービスアカウントを準備したら、[デフォルト HAProxy ルーター](#)、[カスタマイズ HAProxy ルーター](#) のインストールに進むことができます。

### 3.2. デフォルト HAPROXY ルーターの使用

### 3.2.1. 概要

**oc adm router** コマンドには、新規インストールでのルーターのセットアップタスクを単純化する管理者 CLI が提供されます。**oc adm router** コマンドは、サービスとデプロイメント設定オブジェクトを作成します。**--service-account** オプションを使用して、ルーターがマスターとの通信で使用するサービスアカウントを指定します。

**ルーターサービスアカウント** は事前に作成するか、**oc adm router --service-account** コマンドで作成することができます。

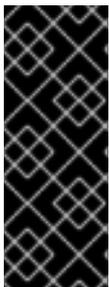
OpenShift Container Platform コンポーネント間のあらゆる形式の通信は TLS によって保護され、各種の証明書や認証方法を使用します。**--default-certificate** .pem フォーマットファイルは提供されるか、または **oc adm router** コマンドで作成されます。**ルート** が作成されると、ユーザーはルートの処理時にルーターが使用するルート証明書を提供できるようになります。



#### 重要

ルーターを削除する際に、デプロイ設定やサービス、およびシークレットも削除されていることを確認します。

ルーターは特定のノードにデプロイされます。これにより、クラスター管理者と外部ネットワークマネージャーはどの IP アドレスがルーターを実行し、ルーターがどのトラフィックを処理するかについて調整しやすくなります。**ノードセクター** を使用してルーターを特定のノードにデプロイします。



#### 重要

ルーターはデフォルトでホストネットワークを使用し、ホストのすべてのインターフェイスのポート 80 と 443 に直接割り当てられます。ポート 80/443 が利用できるホストにルーターを制限し、他のサービスに使用されないようにします。これは **ノードセクター** と **スケジューラー設定** を使用して設定します。たとえば、これはルートなどのサービスの実行用として専用のインフラストラクチャーノードを設定することで実行できます。



#### 重要

お使いのルーターで個別の **openshift-router** サービスアカウントを使用することをお勧めします。**--service-account** フラグを **oc adm router** コマンドで使用してこれを指定できます。

```
$ oc adm router --dry-run --service-account=router 1
```

**1** **--service-account** は、**openshift-router** の **サービスアカウント** の名前です。



#### 重要

**oc adm router** を使用して作成されるルーター Pod には、ルーター Pod がデプロイされるためにノードが満たさなければならないデフォルトのリソース要求が設定されます。デフォルトのリソース要求は、インフラストラクチャーコンポーネントの信頼性を向上させる取り組みとして、ルーター Pod の QoS 層をリソース要求を持たない Pod よりも高く設定するために使用されます。デフォルト値は、基本的なルーターがデプロイされるのに必要な最小限のリソースを表しており、ルーターデプロイメント設定で編集できます。また、ルーターの負荷に基づいてそれらの値を引き上げることもできます。

### 3.2.2. ルーターの作成

ルーターが存在しない場合は、以下を実行してルーターを作成します。

```
$ oc adm router <router_name> --replicas=<number> --service-account=router --extended-logging=true
```

高可用性の設定が作成されない限り、**high availability** は通常 **1** になります。

**--extended-logging=true** は、ルーターを、HAProxy で生成されるログを syslog コンテナに転送するように設定します。

ルーターのホスト IP アドレスを見つけるには、以下を実行します。

```
$ oc get po <router-pod> --template={{.status.hostIP}}
```

また、**ルーターシャードを使用**して、ルーターを特定の namespace またはルートに絞り込むか、ルーターの作成後に **環境変数を設定** できます。この場合には、シャードごとにルーターを作成します。

### 3.2.3. その他の基本ルーターコマンド

#### デフォルトルーターの確認

**router** という名前のデフォルトのルーターサービスアカウントは、クラスターのインストール時に自動的に作成されます。このアカウントがすでに存在することを確認するには、以下を実行します。

```
$ oc adm router --dry-run --service-account=router
```

#### デフォルトルーターの表示

デフォルトルーターが作成されている場合でそれを確認するには、以下を実行します。

```
$ oc adm router --dry-run -o yaml --service-account=router
```

#### HAProxy ログを転送するためのルーター設定

HAProxy で生成されるログを syslog サイドカーコンテナに転送するようにルーターを設定します。**--extended-logging=true** パラメーターは、syslog コンテナを追加して、HAProxy ログを標準出力に転送します。

```
$ oc adm router --extended-logging=true
```

以下の例は、**--extended-logging=true** を使用するルーターの設定です。

```
$ oc get pod router-1-xhdb9 -o yaml
apiVersion: v1
kind: Pod
spec:
  containers:
  - env:
    ....
  - name: ROUTER_SYSLOG_ADDRESS 1
```

```

    value: /var/lib/rsyslog/rsyslog.sock

    ....

- command: ❷
  - /sbin/rsyslogd
  - -n
  - -i
  - /tmp/rsyslog.pid
  - -f
  - /etc/rsyslog/rsyslog.conf
image: registry.redhat.io/openshift3/ose-haproxy-router:v3.11.188
imagePullPolicy: IfNotPresent
name: syslog

```

- ❶ **--extended-logging=true** パラメーターは、ログのソケットファイルを作成します。
- ❷ **--extended-logging=true** パラメーターはコンテナをルーターに追加します。コンテナでは、rsyslog プロセスが `/sbin/rsyslogd -n -i /tmp/rsyslog.pid -f /etc/rsyslog/rsyslog.conf` として実行します。

以下のコマンドを使用して HAProxy ログを表示します。

```

$ oc set env dc/test-router ROUTER_LOG_LEVEL=info ❶
$ oc logs -f <pod-name> -c syslog ❷

```

- ❶ ログレベルを **info** または **debug** に設定します。デフォルトは **warning** です。
- ❷ ログを表示するルーター Pod の名前を指定します。

HAProxy ログは以下の形式を使用します。

```

2020-04-14T03:05:36.629527+00:00 test-311-node-1 haproxy[43]: 10.0.151.166:59594
[14/Apr/2020:03:05:36.627] fe_no_sni~ be_secure:openshift-console:console/pod:console-
b475748cb-t6qkq:console:10.128.0.5:8443 0/0/1/1/2 200 393 - - --NI 2/1/0/1/0 0/0 "HEAD / HTTP/1.1"
2020-04-14T03:05:36.633024+00:00 test-311-node-1 haproxy[43]: 10.0.151.166:59594
[14/Apr/2020:03:05:36.528] public_ssl be_no_sni/fe_no_sni 95/1/104 2793 -- 1/1/0/0/0 0/0

```

### ラベル付けされたノードへのルーターのデプロイ

指定された [ノードラベル](#) に一致するノードにルーターをデプロイするには、以下を実行します。

```

$ oc adm router <router_name> --replicas=<number> --selector=<label> \
  --service-account=router

```

たとえば、**router** という名前のルーターを作成し、それを **node-role.kubernetes.io/infra=true** とラベルが付けられたノードに配置するには、以下を実行します。

```

$ oc adm router router --replicas=1 --selector='node-role.kubernetes.io/infra=true' \
  --service-account=router

```

クラスターのインストール時に、**openshift\_router\_selector** および **openshift\_registry\_selector** の Ansible 設定はデフォルトで **node-role.kubernetes.io/infra=true** に設定されます。デフォルトのルー

ターおよびレジストリーは、**node-role.kubernetes.io/infra=true** ラベルに一致するノードがある場合にのみ自動的にデプロイされます。

ラベルの更新に関する情報については、[ノードのラベルの更新](#) を参照してください。

複数のインスタンスが [スケジューラーポリシー](#) に従って複数の異なるホストに作成されます。

### 複数の異なるルーターイメージの使用

複数の異なるルーターイメージを使用し、使用されるルーター設定を表示するには、以下を実行します。

```
$ oc adm router <router_name> -o <format> --images=<image> \
--service-account=router
```

以下に例を示します。

```
$ oc adm router region-west -o yaml --images=myrepo/somrouter:mytag \
--service-account=router
```

### 3.2.4. ルートを特定のルーターに絞り込む

**ROUTE\_LABELS** 環境変数を使用してルートを絞り込むことで、特定のルーターによってのみ使用されるようになります。

たとえば、複数のルーターと 100 のルートがある場合、ラベルをルートに割り当てることで、それらの一部を 1 つのルーターで処理し、残りを別のルーターで処理することができます。

1. [ルーターの作成](#) 後に、**ROUTE\_LABELS** 環境変数を使用してルーターにタグ付けします。

```
$ oc set env dc/<router=name> ROUTE_LABELS="key=value"
```

2. ラベルを必要なルートに追加します。

```
oc label route <route=name> key=value
```

3. ラベルがルートに割り当てられていることを確認するには、ルートの設定をチェックします。

```
$ oc describe route/<route_name>
```

### 同時接続の最大数の設定

ルーターはデフォルトで最大 20000 の接続を処理できるようになっています。この上限は必要に応じて変更できます。接続が少なすぎると、ヘルスチェックが機能せず、不必要な再起動が実行されます。システムをこの接続の最大数に対応するように設定する必要があります。'**sysctl fs.nr\_open**' と '**sysctl fs.file-max**' に示される上限は十分に大きな値である必要があります。そうでない場合には HAproxy は起動しません。

ルーターを作成したら、**--max-connections=** オプションで必要な上限を設定します。

```
$ oc adm router --max-connections=10000 ....
```

ルーターのデプロイメント設定で **ROUTER\_MAX\_CONNECTIONS** 環境変数を編集し、値を変更します。ルーター Pod は新しい値で再起動されます。**ROUTER\_MAX\_CONNECTIONS** が存在しない場合は、デフォルト値の 20000 が使用されます。



## 注記

接続にはフロントエンドおよび内部バックエンドが含まれます。これは2つの接続としてカウントされます。必ず **ROUTER\_MAX\_CONNECTIONS** の値を作成しようとしている接続数の2倍以上になるように設定してください。

### 3.2.5. HAProxy Strict SNI

HAProxy **strict-sni** は、ルーターのデプロイメント設定の **ROUTER\_STRICT\_SNI** 環境変数によって管理できます。また、これは **--strict-sni** コマンドラインオプションを使用してルーターを作成する時にも設定できます。

```
$ oc adm router --strict-sni
```

### 3.2.6. TLS 暗号化スイート

ルーターの作成時に **--ciphers** オプションを使用して、ルーターの **暗号化スイート** を設定します。

```
$ oc adm router --ciphers=modern ....
```

値は **modern**、**intermediate**、または **old** で、デフォルトは **intermediate** です。または、":" 区切りで暗号化を指定することも可能です。暗号化は、以下のコマンドで表示されたセットの中から選択する必要があります。

```
$ openssl ciphers
```

また、既存のルーターには **ROUTER\_CIPHERS** 環境変数を使用します。

### 3.2.7. 相互 TLS 認証

ルーターおよびバックエンドサービスへのクライアントアクセスは、手動の TLS 認証を使用して制限できます。ルーターは、その **authenticated** セットではなくクライアントからの要求を拒否します。相互 TLS 認証はクライアント証明書で実装でき、証明書を発行する認証局 (CA)、証明書失効一覧および/または証明書件名フィルターに基づいて制御できます。ルーターの作成時に相互 TLS 設定オプションの **--mutual-tls-auth**、**--mutual-tls-auth-ca**、**--mutual-tls-auth-crl** および **--mutual-tls-auth-filter** を使用します。

```
$ oc adm router --mutual-tls-auth=required \  
  --mutual-tls-auth-ca=/local/path/to/cacerts.pem ....
```

**--mutual-tls-auth** の値は **required**、**optional**、または **none** であり、**none** はデフォルトになります。 **--mutual-tls-auth-ca** 値は1つ以上の CA 証明書を含むファイルを指定します。これらの CA 証明書はクライアントの証明書を検証するためにルーターによって使用されます。

**--mutual-tls-auth-crl** は、(有効な認証局が発行する) 証明書が取り消されているケースを処理するために証明書失効一覧を指定するために使用できます。

```
$ oc adm router --mutual-tls-auth=required \  
  --mutual-tls-auth-ca=/local/path/to/cacerts.pem \  
  --mutual-tls-auth-filter='^/CN=my.org/ST=CA/C=US/O=Security/OU=OSE$' \  
  ....
```

**--mutual-tls-auth-filter** 値は、証明書件名に基づくきめ細やかなアクセス制御に使用できます。この値は、証明書の件名に一致させるために使用される正規表現です。



## 注記

上記の相互 TLS 認証フィルターの例では、**^** および **\$** のアンカーが使用された制限的な正規表現 (regex) を示しており、これは証明書件名に **完全に** 一致します。制限的な正規表現を使用することにした場合、有効とみなされるすべての CA によって発行された証明書に一致する可能性があることに留意してください。発行された証明書に対してよりきめ細やかな制御を行えるように **--mutual-tls-auth-ca** オプションを使用することも推奨されています。

**--mutual-tls-auth=required** を使用することにより、バックエンドリソースへのアクセスを認証されたクライアント **のみ** に許可することができます。これは、クライアントが常に認証情報 (またはクライアント証明書) を提供するために **必要である** ことを意味します。相互 TLS 認証を Optional (オプション) にするには、**--mutual-tls-auth=optional** を使用 (または、デフォルトの **none** を使用) してこれを無効にします。ここで、**optional** とは、クライアントに認証情報の提示を要求する必要が **ない** ことを意味し、クライアントが認証情報を提供する場合は、その情報は **X-SSL\*** HTTP ヘッダーでバックエンドに渡されることを意味します。

```
$ oc adm router --mutual-tls-auth=optional \
  --mutual-tls-auth-ca=/local/path/to/cacerts.pem \
  ....
```

相互 TLS 認証サポートが有効にされる場合 (**--mutual-tls-auth** フラグに **required** または **optional** 値のいずれかを使用)、クライアント認証情報は **X-SSL\*** HTTP ヘッダーの形式でバックエンドに渡されます。

**X-SSL\*** HTTP ヘッダー **X-SSL-Client-DN** の例: 証明書の件名の完全な識別名 (DN)。 **X-SSL-Client-NotBefore**: YYMMDDhhmmss[Z] 形式のクライアント証明書の開始日。 **X-SSL-Client-NotAfter**: YYMMDDhhmmss[Z] 形式のクライアント証明書の開始日。 **X-SSL-Client-SHA1**: クライアント証明書の SHA-1 フィンガープリント。 **X-SSL-Client-DER**: クライアント証明書への完全なアクセスを提供します。 base-64 形式でエンコードされた DER フォーマットのクライアント証明書が含まれます。

### 3.2.8. 高可用性ルーター

IP フェイルオーバーを使用して OpenShift Container Platform クラスターで [高可用性ルーターをセットアップ](#) できます。このセットアップには複数の異なるノードの複数のレプリカが含まれるため、フェイルオーバーソフトウェアは現在のレプリカが失敗したら別のレプリカに切り替えることができます。

### 3.2.9. ルーターサービスポートのカスタマイズ

環境変数の [ROUTER\\_SERVICE\\_HTTP\\_PORT](#) と [ROUTER\\_SERVICE\\_HTTPS\\_PORT](#) を設定することで、テンプレートルーターがバインドするサービスポートをカスタマイズできます。これは、テンプレートルーターを作成し、そのデプロイメント設定を編集することで実行できます。

以下の例では、**0** レプリカのルーターデプロイメントを作成し、ルーターサービス HTTP と HTTPS ポートをカスタマイズし、これを適切に (**1** レプリカに) スケーリングしています。

```
$ oc adm router --replicas=0 --ports='10080:10080,10443:10443' 1
$ oc set env dc/router ROUTER_SERVICE_HTTP_PORT=10080 \
  ROUTER_SERVICE_HTTPS_PORT=10443
$ oc scale dc/router --replicas=1
```

- 1 公開されるポートがコンテナネットワークモード `--host-network=false` を使用するルーターに対して適切に設定されていることを確認します。



### 重要

テンプレートルーターサービスポートをカスタマイズする場合は、ルーター Pod が実行されるノードのファイアウォールでカスタムポートが開いているようにする必要があります (Ansible または `iptables` のいずれか、または `firewall-cmd` で使用するその他のカスタム方法を使用します)。

以下は、カスタムルーターサービスポートを開くために `iptables` を使用した例です。

```
$ iptables -A OS_FIREWALL_ALLOW -p tcp --dport 10080 -j ACCEPT
$ iptables -A OS_FIREWALL_ALLOW -p tcp --dport 10443 -j ACCEPT
```

### 3.2.10. 複数ルーターの使用

管理者は同じ定義を使用して複数のルーターを作成し、同じルートのセットを提供することができます。各ルーターは複数の異なる `ノード` に置かれ、異なる IP アドレスを持ちます。ネットワーク管理者は、各ノードに必要なトラフィックを送る必要があります。

複数のルーターをグループ化して、クラスター内や複数テナントでのルーティングの負荷を別のルーターまたは `シャード` に分散することができます。グループの各ルーターまたはシャードは、ルーターのセレクターに基づいてルートを許可します。管理者は `ROUTE_LABELS` を使用してクラスター全体でシャードを作成できます。ユーザーは `NAMESPACE_LABELS` を使用して namespace (プロジェクト) でシャードを作成できます。

### 3.2.11. デプロイメント設定へのノードセレクターの追加

特定のルーターを特定のノードにデプロイするには、2つの手順を実行する必要があります。

1. `ラベル` を必要なノードに追加します。

```
$ oc label node 10.254.254.28 "router=first"
```

2. ノードセレクターをルーターのデプロイメント設定に追加します。

```
$ oc edit dc <deploymentConfigName>
```

`template.spec.nodeSelector` フィールドに、ラベルに対応するキーと値を追加します。

```
...
template:
  metadata:
    creationTimestamp: null
    labels:
      router: router1
  spec:
    nodeSelector:
      router: "first"
...

```

1

- 1 **router=first** ラベルに対応するキーと値はそれぞれ **router** と **first** になります。

### 3.2.12. ルーターシャードの使用

**ルーターのシャード化**により、**NAMESPACE\_LABELS**と**ROUTE\_LABELS**を使用してルーターの namespace とルートの絞り込みが実行されます。これにより、複数のルーターデプロイメントでルートのサブセットを分散させることができます。重複しないサブセットを使用することにより、ルートセットのパーティション設定を効果的に行うことができます。または、重複するルートのサブセットを設定する複数のシャードを定義することもできます。

デフォルトで、ルーターはすべての **プロジェクト (namespace)** からすべてのルートを選択します。シャード化によってラベルがルートまたはルーターの namespace およびラベルセクターに追加されます。各ルーターシャードは特定のラベルのセットで選択されるルーターを設定するか、または特定のラベルセクターで選択される namespace に属します。



#### 注記

ルーターサービスアカウントには [**cluster reader**] パーミッションセットを設定し、他の namespace のラベルにアクセスできるようにする必要があります。

### ルーターのシャード化および DNS

外部 DNS サーバーは要求を必要なシャードにルートするために必要となるので、管理者はプロジェクトの各ルーターに個別の DNS エントリーを作成する必要があります。ルーターは不明なルートを別のルーターに転送することはありません。

以下の例を考慮してください。

- Router A はホスト 192.168.0.5 にあり、**\*.foo.com** のルートを持つ。
- Router B はホスト 192.168.1.9 にあり、**\*.example.com** のルートを持つ。

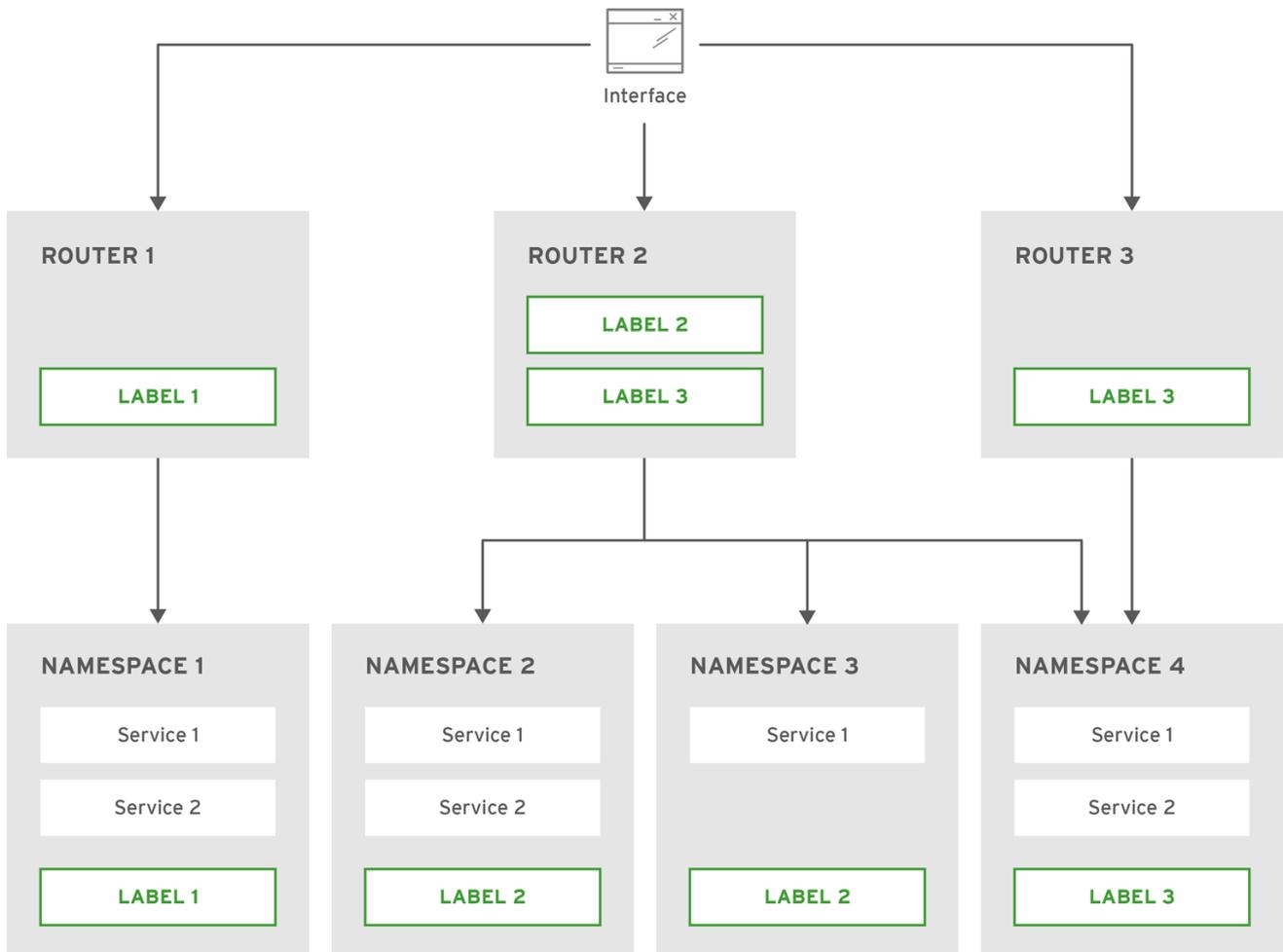
各 DNS エントリーは \*.foo.com を Router A をホストするノードに解決し、\*.example.com を Router B をホストするノードに解決する必要があります。

- **\*.foo.com A IN 192.168.0.5**
- **\*.example.com A IN 192.168.1.9**

### ルーターのシャード化の例

このセクションでは、namespace およびルートラベルを使用するルーターのシャード化について説明します。

図3.1 namespace ラベルに基づくルーターのシャード化



OPENSIFT\_415490\_0217

1. namespace ラベルセクターでルーターを設定します。

```
$ oc set env dc/router NAMESPACE_LABELS="router=r1"
```

2. ルーターには namespace にセクターがあるため、ルーターは一致する namespace のルートのみを処理します。このセクターが namespace に一致させるようにするには、namespace に適宜ラベルを付けます。

```
$ oc label namespace default "router=r1"
```

3. ルートをデフォルトの namespace に作成すると、ルートはデフォルトのルーターで利用できるようになります。

```
$ oc create -f route1.yaml
```

4. 新規プロジェクト (namespace) を作成し、**route2** というルートを作成します。

```
$ oc new-project p1
$ oc create -f route2.yaml
```

ルートがルーターで利用できないことを確認します。

5. namespace **p1** に **router=r1** のラベルを付けます。

-

```
$ oc label namespace p1 "router=r1"
```

このラベルを追加すると、ルートはルーターで利用できるようになります。

#### 例

ルーターのデプロイメント **finops-router** はルートセレクター **NAMESPACE\_LABELS="name in (finance, ops)"** を使用して実行され、ルーターのデプロイメント **dev-router** はラベルセレクター **NAMESPACE\_LABELS="name=dev"** を使用して設定されます。

すべてのルートが **name=finance**、**name=ops**、および **name=dev** というラベルの付けられた namespace がない場合、この設定により、2つのルーターのデプロイメント間でルートが効果的に分散されます。

上記のシナリオでは、シャード化は重複するセットを持たないパーティション設定の特別なケースとなります。ルートは複数のルーターシャード間で分割されます。

ルート選択の基準によって、ルートの分散方法が決まります。複数のルーターデプロイメントに重複するルートのサブセットを設定することも可能です。

#### 例

上記の例では **finops-router** と **dev-router** のほかに **devops-router** があり、これはラベルセレクター **NAMESPACE\_LABELS="name in (dev, ops)"** を使用して設定されます。

**name=dev** または **name=ops** というラベルが付けられた namespace のルートは2つの異なるルーターデプロイメントによって提供されるようになりました。これは、[namespace ラベルに基づくルーターのシャード化](#) の手順で説明されているように、ルートの重複するサブセットを定義するケースです。

また、これによりさらに複雑なルーティングルールを作成し、優先度の高いトラフィックを専用の **finops-router** に転送し、優先度の低いトラフィックは **devops-router** に送信できます。

### ルートラベルに基づくルーターのシャード化

**NAMESPACE\_LABELS** によって、提供するプロジェクトをフィルターでき、それらのプロジェクトからすべてのルートを選択できますが、ルートはルート自体に関連する他の基準に基づいてパーティション設定する必要がある場合があります。**ROUTE\_LABELS** セレクターを使用すると、ルート自体を細かくフィルターできます。

#### 例

ルーターデプロイメント **prod-router** はルートセレクター **ROUTE\_LABELS="mydeployment=prod"** を使用して設定され、ルーターデプロイメント **devtest-router** はラベルセレクター **ROUTE\_LABELS="mydeployment in (dev, test)"** を使用して設定されます。

この設定は、namespace の種類を問わず、ルートのラベルに基づいて2つのルーターデプロイメント間のルートのパーティション設定を行います。

この例では、提供されるルートすべてがラベル **"mydeployment=<tag>"** でタグ付けされていることを想定しています。

#### 3.2.12.1. ルーターシャードの作成

このセクションでは、ルーターシャードのさらに詳細な例を示します。さまざまなラベルを持つ **a-z** という26のルートがあることを想定してください。

#### ルートで使用可能なラベル

```
sla=high   geo=east   hw=modest  dept=finance
sla=medium geo=west   hw=strong  dept=dev
sla=low    dept=ops
```

これらのラベルは、サービスレベルアグリーメント、地理的な場所、ハードウェア要件、部門などの概念を表しています。ルートは各列のラベルを最大1つ持つことができます。ルートによっては他のラベルを持つこと、ラベルをまったく持たないこともあります。

名前	SLA	Geo (地理的な場所)	HW	Dept (部門)	その他のラベル
a	high	east	modest	finance	type=static
b		west	strong		type=dynamic
c, d, e	low		modest		type=static
g – k	medium		strong	dev	
l – s	high		modest	ops	
t – z		west			type=dynamic

これは **oc adm router**、**oc set env** および **oc scale** がどのように連携してルーターシャードを作成するかを表している便利なスクリプト **mkshard** です。

```
#!/bin/bash
# Usage: mkshard ID SELECTION-EXPRESSION
id=$1
sel="$2"
router=router-shard-$id
oc adm router $router --replicas=0
dc=dc/router-shard-$id
oc set env $dc ROUTE_LABELS="$sel"
oc scale $dc --replicas=3
```

- ❶ 作成されたルーターは **router-shard-<id>** という名前を持ちます。
- ❷ ここではスケーリングを指定しません。
- ❸ ルーターのデプロイメント設定。
- ❹ **oc set env** を使用して選択式を設定します。選択式は環境変数 **ROUTE\_LABELS** の値です。
- ❺ 拡張します。

**mkshard** を複数回実行して、複数のルーターを作成します。

ルーター	選択式	ルート
router-shard-1	sla=high	a, l – s
router-shard-2	geo=west	b, t – z
router-shard-3	dept=dev	g – k

### 3.2.12.2. ルーターシャードの変更

ルーターシャードは [ラベルに基づいた](#) 設定なので、(`oc label` を使用して) ラベルまたは (`oc set env` を使用して) 選択式のいずれかを変更できます。

このセクションでは [ルーターシャードの作成](#) セクションで扱った例をさらに詳細に取り上げ、選択式の変更方法を示します。

これは、新規の選択式を使用できるように既存のルーターを変更する便利なスクリプト `modshard` です。

```
#!/bin/bash
# Usage: modshard ID SELECTION-EXPRESSION...
id=$1
shift
router=router-shard-$id
dc=dc/$router
oc scale $dc --replicas=0
oc set env $dc "$@"
oc scale $dc --replicas=3
```

- 1 変更後のルーターの名前は `router-shard-<id>` になります。
- 2 変更が発生するデプロイメント設定です。
- 3 縮小します。
- 4 `oc set env` を使用して新しい選択式を設定します。[ルーターシャードの作成](#) セクションの `mkshard` とは異なり、`modshard` の ID 以外の引数として指定される選択式には環境変数名とその値が含まれている必要があります。
- 5 拡大して元に戻します。



#### 注記

`modshard` では、`router-shard-<id>` の [デプロイメントストラテジー](#) が **Rolling** の場合、`oc scale` コマンドは不要です。

たとえば `router-shard-3` の部門を拡張して `ops` と `dev` を含めるには、以下を実行します。

```
$ modshard 3 ROUTE_LABELS='dept in (dev, ops)'
```

結果として、`router-shard-3` はルート `g – s` (`g – k` と `l – s` の組み合わせ) を選択します。

この例ではシャードから除外する1つの部門を指定します。このシナリオ例では3つの部門しかないため、これによって前述の例と同じ結果が得られます。

```
$ modshard 3 ROUTE_LABELS='dept != finance'
```

この例は3つのコマンドで区切られた属性を指定しており、結果としてルート **b** のみが選択されます。

```
$ modshard 3 ROUTE_LABELS='hw=strong,type=dynamic,geo=west'
```

ルートのラベルを使用する **ROUTE\_LABELS** と同様に、**NAMESPACE\_LABELS** 環境変数を使用して、ルートはルートの namespace ラベルのラベルに基づいて選択できます。この例では、ラベル **frequency=weekly** を持つルートの namespace を提供するように **router-shard-3** を変更します。

```
$ modshard 3 NAMESPACE_LABELS='frequency=weekly'
```

最後の例は **ROUTE\_LABELS** と **NAMESPACE\_LABELS** を組み合わせて、ラベル **sla=low** を持ち、ラベル **frequency=weekly** を持つ namespace のルートを選択します。

```
$ modshard 3 \
  NAMESPACE_LABELS='frequency=weekly' \
  ROUTE_LABELS='sla=low'
```

### 3.2.13. ルーターのホスト名の検索

サービスを公開する際に、ユーザーは外部ユーザーがアプリケーションにアクセスするために使用する DNS 名からの同じルートを使用できます。外部ネットワークのネットワーク管理者は、ホスト名がルートを許可したルーター名に解決することを確認する必要があります。ユーザーはこのホスト名を指す CNAME を使用して DNS をセットアップできます。ただし、ルーターのホスト名が不明な場合があります。不明な場合は、クラスター管理者は指定できます。

クラスター管理者は、**--router-canonical-hostname** オプションをルーター作成時のルーターの正規ホスト名で使用できます。以下に例を示します。

```
# oc adm router myrouter --router-canonical-hostname="rtr.example.com"
```

これは、ルーターのホスト名を含む **ROUTER\_CANONICAL\_HOSTNAME** 環境変数をルーターのデプロイメント設定に作成します。

すでに存在しているルーターの場合、クラスター管理者はルーターのデプロイメント設定を編集し、**ROUTER\_CANONICAL\_HOSTNAME** 環境変数を追加します。

```
spec:
  template:
    spec:
      containers:
        - env:
            - name: ROUTER_CANONICAL_HOSTNAME
              value: rtr.example.com
```

**ROUTER\_CANONICAL\_HOSTNAME** 値は、ルートを許可したすべてのルーターのルートステータスに表示されます。ルートステータスはルーターがリロードされるたびに更新されます。

ユーザーがルートを作成すると、すべてのアクティブなルーターはそのルートを評価し、条件を満たし

ていればそのルートを許可します。**ROUTER\_CANONICAL\_HOSTNAME** 環境変数を定義するルーターがルートを許可すると、ルーターはルートステータスの **routerCanonicalHostname** フィールドに値を入力します。ユーザーはルートステータスを検証して、どのルーターがルートを許可したかを確認でき、ルーターを一覧から選択し、ネットワーク管理者に渡すルーターのホスト名を見つけることができます (該当する場合)。

```
status:
  ingress:
    conditions:
      lastTransitionTime: 2016-12-07T15:20:57Z
      status: "True"
      type: Admitted
    host: hello.in.mycloud.com
    routerCanonicalHostname: rtr.example.com
    routerName: myrouter
    wildcardPolicy: None
```

**oc describe** にはホスト名が含まれます (利用可能な場合)。

```
$ oc describe route/hello-route3
...
Requested Host: hello.in.mycloud.com exposed on router myroute (host rtr.example.com) 12 minutes ago
```

上記の情報を使用して、ユーザーは DNS 管理者に対し、ルートのホスト **hello.in.mycloud.com** から CNAME をルーターの正規ホスト名 **rtr.example.com** に応じてセットアップするよう依頼できます。この結果として、**hello.in.mycloud.com** へのトラフィックがユーザーのアプリケーションに到達するようになります。

### 3.2.14. デフォルトのルーティングサブドメインのカスタマイズ

**マスター設定ファイル** (デフォルトでは `/etc/origin/master/master-config.yaml` ファイル) を変更することで、お使いの環境のデフォルトルーティングサブドメインとして使用される接尾辞をカスタマイズできます。ホスト名を指定しないルートの場合、このデフォルトのルーティングサブドメインを使用してホスト名が生成されます。

以下の例は、設定された接尾辞を **v3.openshift.test** に設定する方法を示しています。

```
routingConfig:
  subdomain: v3.openshift.test
```



#### 注記

この変更には、マスターを実行している場合は再起動が必要となります。

OpenShift Container Platform マスターが上記の設定を実行している場合、`namespace` の **mynamespace** に追加されるホスト名を持たない **no-route-hostname** というルートの例では、**生成されるホスト名**は以下ようになります。

```
no-route-hostname-mynamespace.v3.openshift.test
```

### 3.2.15. カスタムルーティングサブドメインへのルートホスト名の強制

管理者がすべてのルートを実際のルーティングサブドメインに限定する場合、**--force-subdomain** オプションを **oc adm router** コマンドに渡すことができます。これはルートで指定されたホスト名を上書きし、**--force-subdomain** オプションに提供されるテンプレートに基づいてホスト名を生成するようルーターに強制します。

以下の例ではルーターを実行し、カスタムサブドメインテンプレート **`\${name}-\${namespace}.apps.example.com`** を使用してルートホスト名を上書きしています。

```
$ oc adm router --force-subdomain='${name}-${namespace}.apps.example.com'
```

### 3.2.16. ワイルドカード証明書の使用

証明書を含まない TLS 対応のルートはルーターのデフォルト証明書を代わりに使用します。ほとんどの場合、この証明書は信頼された認証局から提供されますが、利便性を考慮して OpenShift Container Platform CA を使用して証明書を作成することができます。以下に例を示します。

```
$ CA=/etc/origin/master
$ oc adm ca create-server-cert --signer-cert=$CA/ca.crt \
  --signer-key=$CA/ca.key --signer-serial=$CA/ca.serial.txt \
  --hostnames='*.cloudapps.example.com' \
  --cert=cloudapps.crt --key=cloudapps.key
```

#### 注記

**oc adm ca create-server-cert** コマンドは、2年間有効な証明書を生成します。この期間は **--expire-days** オプションを使って変更することができますが、セキュリティ上の理由から、値をこれより大きくすることは推奨されません。

Ansible ホストインベントリーファイル (デフォルトで **/etc/ansible/hosts**) に最初に一覧表示されているマスターから **oc adm** コマンドを実行します。

ルーターは、証明書とキーが単一ファイルに PEM 形式で入力されていると予想します。

```
$ cat cloudapps.crt cloudapps.key $CA/ca.crt > cloudapps.router.pem
```

ここで **--default-cert** フラグを使用できます。

```
$ oc adm router --default-cert=cloudapps.router.pem --service-account=router
```

#### 注記

ブラウザーは、ワイルドカードを1つ深いレベルのサブドメインに有効であると見なしません。この例では、証明書は **a.cloudapps.example.com** に対して有効ですが、**a.b.cloudapps.example.com** には有効ではありません。

### 3.2.17. 証明書を手動で再デプロイする

ルーター証明書を手動で再デプロイするには、以下を実行します。

1. デフォルトのルーター証明書を含むシークレットがルーターに追加されているかどうかを確認します。

```
$ oc set volume dc/router
deploymentconfigs/router
secret/router-certs as server-certificate
mounted at /etc/pki/tls/private
```

証明書が追加されている場合は、以下の手順を省略してシークレットを上書きします。

2. デフォルト証明書ディレクトリーが以下の変数 **DEFAULT\_CERTIFICATE\_DIR** に設定されていることを確認します。

```
$ oc set env dc/router --list
DEFAULT_CERTIFICATE_DIR=/etc/pki/tls/private
```

設定されていない場合は、以下のコマンドを使用してディレクトリーを作成します。

```
$ oc set env dc/router DEFAULT_CERTIFICATE_DIR=/etc/pki/tls/private
```

3. 証明書を PEM 形式にエクスポートします。

```
$ cat custom-router.key custom-router.crt custom-ca.crt > custom-router.crt
```

4. ルーター証明書シークレットを上書きするか、またはこれを作成します。  
証明書シークレットがルーターに追加されている場合は、シークレットを上書きします。追加されていない場合は、新規シークレットを作成します。

シークレットを上書きするには、以下のコマンドを実行します。

```
$ oc create secret generic router-certs --from-file=tls.crt=custom-router.crt --from-
file=tls.key=custom-router.key --type=kubernetes.io/tls -o json --dry-run | oc replace -f -
```

新規シークレットを作成するには、以下のコマンドを実行します。

```
$ oc create secret generic router-certs --from-file=tls.crt=custom-router.crt --from-
file=tls.key=custom-router.key --type=kubernetes.io/tls

$ oc set volume dc/router --add --mount-path=/etc/pki/tls/private --secret-name='router-certs'
--name router-certs
```

5. ルーターをデプロイします。

```
$ oc rollout latest dc/router
```

### 3.2.18. セキュリティー保護されたルートの使用

現時点で、パスワードで保護されたキーファイルはサポートされていません。HAProxy は開始時にパスワードを求めるプロンプトを出しますが、このプロセスを自動化する方法はありません。キーファイルからパスフレーズを削除するために、以下を実行できます。

```
# openssl rsa -in <passwordProtectedKey.key> -out <new.key>
```

以下の例は、トラフィックが宛先にプロキシ処理される前に TLS 終端がルーターで生じる場合にセキュアな edge termination ルートを使用する方法を示しています。セキュアな edge termination ルートは TLS 証明書とキー情報を指定します。TLS 証明書は、ルーターのフロントエンドで提供されません。

最初にルーターインスタンスを起動します。

```
# oc adm router --replicas=1 --service-account=router
```

次に、セキュアな edge ルートのプライベートキー、CSR、証明書を作成します。この手順はお使いの認証局やプロバイダーによって異なります。**www.example.test** というドメインの単純な自己署名証明書の場合は、以下の例を参照してください。

```
# sudo openssl genrsa -out example-test.key 2048
#
# sudo openssl req -new -key example-test.key -out example-test.csr \
  -subj "/C=US/ST=CA/L=Mountain View/O=OS3/OU=Eng/CN=www.example.test"
#
# sudo openssl x509 -req -days 366 -in example-test.csr \
  -signkey example-test.key -out example-test.crt
```

上記の証明書とキーを使用してルートを生成します。

```
$ oc create route edge --service=my-service \
  --hostname=www.example.test \
  --key=example-test.key --cert=example-test.crt
route "my-service" created
```

その定義を確認します。

```
$ oc get route/my-service -o yaml
apiVersion: v1
kind: Route
metadata:
  name: my-service
spec:
  host: www.example.test
  to:
    kind: Service
    name: my-service
  tls:
    termination: edge
    key: |
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

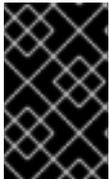
**www.example.test** の DNS エントリがルーターインスタンスを指し、ドメインへのルートが利用できることを確認します。以下の例では、Curl をローカルリゾルバーと共に使用して DNS ルックアップのシミュレーションを行っています。

-

```
# routerip="4.1.1.1" # replace with IP address of one of your router instances.
# curl -k --resolve www.example.test:443:$routerip https://www.example.test/
```

### 3.2.19. (サブドメインの) ワイルドカードルートの使用

HAProxy ルーターはワイルドカードルートをサポートしており、**ROUTER\_ALLOW\_WILDCARD\_ROUTES** 環境変数を **true** に設定することでこれを有効にできます。ルーター許可のチェックをパスする **Subdomain** のワイルドカードポリシーを持つすべてのルートは HAProxy ルーターによって提供されます。次に、HAProxy ルーターはルートのワイルドカードポリシーに基づいて (ルートの) 関連サービスを公開します。



#### 重要

ルートのワイルドカードポリシーを変更するには、ルートを削除してから更新されたワイルドカードポリシーでこれを再作成する必要があります。ルートの **.yaml** ファイルでルートのワイルドカードポリシーのみを編集しても機能しません。

```
$ oc adm router --replicas=0 ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
$ oc scale dc/router --replicas=1
```

Web コンソールでワイルドカードルートを設定する方法については [こちら](#) を参照してください。

### セキュアなワイルドカード edge termination ルートの使用

以下の例では、トラフィックが宛先にプロキシ処理される前にルーターで生じる TLS 終端を反映しています。サブドメイン **example.org** (**\*.example.org**) のホストに送られるトラフィックは公開されるサービスにプロキシされます。

セキュアな edge termination ルートは TLS 証明書とキー情報を指定します。TLS 証明書は、サブドメイン (**\*.example.org**) に一致するすべてのホストのルーターのフロントエンドによって提供されます。

1. ルーターインスタンスを起動します。

```
$ oc adm router --replicas=0 --service-account=router
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
$ oc scale dc/router --replicas=1
```

2. セキュリティ保護された edge ルートについてのプライベートキー、証明書署名要求 (CSR) および証明書を作成します。

この手順はお使いの認証局やプロバイダーによって異なります。**\*.example.test** というドメインの単純な自己署名証明書の場合は、以下の例を参照してください。

```
# sudo openssl genrsa -out example-test.key 2048
#
# sudo openssl req -new -key example-test.key -out example-test.csr \
  -subj "/C=US/ST=CA/L=Mountain View/O=OS3/OU=Eng/CN=*.example.test"
#
# sudo openssl x509 -req -days 366 -in example-test.csr \
  -signkey example-test.key -out example-test.crt
```

3. 上記の証明書とキーを使用してワイルドカードのルートを生成します。

```
$ cat > route.yaml <<EOF
apiVersion: v1
kind: Route
metadata:
  name: my-service
spec:
  host: www.example.test
  wildcardPolicy: Subdomain
  to:
    kind: Service
    name: my-service
  tls:
    termination: edge
    key: "$(perl -pe 's/\n//n/' example-test.key)"
    certificate: "$(perl -pe 's/\n//n/' example-test.cert)"
EOF
$ oc create -f route.yaml
```

\***example.test** の DNS エントリーがお使いのルーターインスタンスを指し、ドメインへのルートが利用できることを確認します。

この例では **curl** をローカルリゾルバーと共に使用し、DNS ルックアップのシミュレーションを行います。

```
# routerip="4.1.1.1" # replace with IP address of one of your router instances.
# curl -k --resolve www.example.test:443:$routerip https://www.example.test/
# curl -k --resolve abc.example.test:443:$routerip https://abc.example.test/
# curl -k --resolve anyname.example.test:443:$routerip https://anyname.example.test/
```

ワイルドカードルートを許可しているルーター (**ROUTER\_ALLOW\_WILDCARD\_ROUTES** を **true** に設定する) の場合、ワイルドカードルートに関連付けられたサブドメインの所有権についてのいくつかの注意点があります。

ワイルドカードルートの設定前に、所有権は、最も古いルートを持つ namespace のホスト名についての要求に基づいて設定されました (これはその他の要求を行うルートよりも優先されました)。たとえば、ルート **r1** がルート **r2** より古い場合、**one.example.test** の要求を持つ namespace **ns1** のルート **r1** は同じホスト名 **one.example.test** について namespace **ns2** のルート **ns2** よりも優先されます。

さらに、他の namespace のルートは重複しないホスト名を要求することを許可されていました。たとえば、namespace **ns1** のルート **rone** は **www.example.test** を要求でき、namespace **d2** の別のルート **rtwo** は **c3po.example.test** を要求できました。

これは、同じサブドメイン (上記の例では **example.test**) を要求するワイルドカードルートがない場合には同様になります。

ただし、ワイルドカードルートはサブドメイン内のホスト名 (**/\*.example.test** 形式のホスト名) をすべて要求する必要があります。ワイルドカードルートの要求は、そのサブドメイン (**example.test**) の最も古いルートがワイルドカードルートと同じ namespace 内にあるかどうかによって許可または拒否されます。最も古いルートは通常のルートまたはワイルドカードルートのいずれかになります。

たとえば、ホスト **owner.example.test** を要求する **最も古い** ルートが namespace **ns1** にすでに存在し、後からそのサブドメイン (**example.test**) のルートを要求する新規のワイルドカードルート **wildthing** が追加される場合、そのワイルドカードルートによる要求は、そのルートが所有ルートと同じ namespace (**ns1**) にある場合にのみ許可されます。

以下の例では、ワイルドカードルートの要求が成功する場合と失敗する場合のさまざまなシナリオを示しています。

以下の例では、ワイルドカードルートを許可するルーターは、ワイルドカードルートがサブドメインを要求していない限り、サブドメイン **example.test** のホストに対する重複しない要求を許可します。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test
$ oc expose service myservice --hostname=aname.example.test
$ oc expose service myservice --hostname=bname.example.test

$ oc project ns2
$ oc expose service anotherservice --hostname=second.example.test
$ oc expose service anotherservice --hostname=cname.example.test

$ oc project othersns
$ oc expose service thirdservice --hostname=emmy.example.test
$ oc expose service thirdservice --hostname=webby.example.test
```

以下の例では、ワイルドカードルートを許可するルーターは、所有している namespace が **ns1** のので、**owner.example.test** または **aname.example.test** の要求を許可しません。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test
$ oc expose service myservice --hostname=aname.example.test

$ oc project ns2
$ oc expose service secondservice --hostname=bname.example.test
$ oc expose service secondservice --hostname=cname.example.test

$ # Router will not allow this claim with a different path name `/p1` as
$ # namespace `ns1` has an older route claiming host `aname.example.test`.
$ oc expose service secondservice --hostname=aname.example.test --path="/p1"

$ # Router will not allow this claim as namespace `ns1` has an older route
$ # claiming host name `owner.example.test`.
$ oc expose service secondservice --hostname=owner.example.test

$ oc project othersns

$ # Router will not allow this claim as namespace `ns1` has an older route
$ # claiming host name `aname.example.test`.
$ oc expose service thirdservice --hostname=aname.example.test
```

以下の例では、ワイルドカードルートを許可するルーターは、所有している namespace が **ns1** で、そのワイルドカードルートが同じ namespace に属しているので、**/\*.example.test** の要求を許可します。

```
$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true
```

```

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ # host: www.example.test
$ # wildcardPolicy: Subdomain

$ oc create -f route.yaml # router will allow this claim.

```

以下の例では、ワイルドカードルートを許可するルーターは、所有している namespace が **ns1** で、ワイルドカードルートが別の namespace **cyclone** に属するため、`*.example.test` の要求を許可しません。

```

$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ # Switch to a different namespace/project.
$ oc project cyclone

$ # Reusing the route.yaml from a prior example.
$ # spec:
$ # host: www.example.test
$ # wildcardPolicy: Subdomain

$ oc create -f route.yaml # router will deny (_NOT_ allow) this claim.

```

同様に、ワイルドカードルートを持つ namespace がサブドメインを要求すると、その namespace 内のルートのみがその同じサブドメインでホストを要求できます。

以下の例では、ワイルドカードルートを持つ namespace **ns1** のルートがサブドメイン **example.test** を要求すると、namespace **ns1** 内のルートのみがその同じサブドメインのホストを要求することを許可されます。

```

$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=owner.example.test

$ oc project othersns

$ # namespace `othersns` is allowed to claim for other.example.test
$ oc expose service otherservice --hostname=other.example.test

$ oc project ns1

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ # host: www.example.test
$ # wildcardPolicy: Subdomain

```

```

$ oc create -f route.yaml # Router will allow this claim.

$ # In addition, route in namespace othersns will lose its claim to host
$ # `other.example.test` due to the wildcard route claiming the subdomain.

$ # namespace `ns1` is allowed to claim for deux.example.test
$ oc expose service mysecondservice --hostname=deux.example.test

$ # namespace `ns1` is allowed to claim for deux.example.test with path /p1
$ oc expose service mythirdservice --hostname=deux.example.test --path="/p1"

$ oc project othersns

$ # namespace `othersns` is not allowed to claim for deux.example.test
$ # with a different path '/otherpath'
$ oc expose service otherservice --hostname=deux.example.test --path="/otherpath"

$ # namespace `othersns` is not allowed to claim for owner.example.test
$ oc expose service yetanotherservice --hostname=owner.example.test

$ # namespace `othersns` is not allowed to claim for unclaimed.example.test
$ oc expose service yetanotherservice --hostname=unclaimed.example.test

```

以下の例では、所有権のあるルートが削除され、所有権が namespace 内または namespace 間で渡されるさまざまなシナリオが示されています。namespace **ns1** のホスト **eldest.example.test** を要求するルートが存在する場合、その namespace 内のワイルドカードルートはサブドメイン **example.test** を要求できます。ホスト **eldest.example.test** のルートが削除されると、次に古いルート **senior.example.test** が最も古いルートになりますが、これは他のルートに影響を与えません。ホスト **senior.example.test** のルートが削除されると、次に古いルート **junior.example.test** が最も古いルートになり、ワイルドカードルートの要求をブロックします。

```

$ oc adm router ...
$ oc set env dc/router ROUTER_ALLOW_WILDCARD_ROUTES=true

$ oc project ns1
$ oc expose service myservice --hostname=eldest.example.test
$ oc expose service seniorservice --hostname=senior.example.test

$ oc project othersns

$ # namespace `othersns` is allowed to claim for other.example.test
$ oc expose service juniorservice --hostname=junior.example.test

$ oc project ns1

$ # Reusing the route.yaml from the previous example.
$ # spec:
$ # host: www.example.test
$ # wildcardPolicy: Subdomain

$ oc create -f route.yaml # Router will allow this claim.

$ # In addition, route in namespace othersns will lose its claim to host
$ # `junior.example.test` due to the wildcard route claiming the subdomain.

```

```

$ # namespace `ns1` is allowed to claim for dos.example.test
$ oc expose service mysecondservice --hostname=dos.example.test

$ # Delete route for host `eldest.example.test`, the next oldest route is
$ # the one claiming `senior.example.test`, so route claims are unaffected.
$ oc delete route myservice

$ # Delete route for host `senior.example.test`, the next oldest route is
$ # the one claiming `junior.example.test` in another namespace, so claims
$ # for a wildcard route would be affected. The route for the host
$ # `dos.example.test` would be unaffected as there are no other wildcard
$ # claimants blocking it.
$ oc delete route seniorservice

```

### 3.2.20. コンテナネットワークスタックの使用

OpenShift Container Platform ルーターはコンテナ内で実行され、デフォルトの動作として、ホスト (例: ルーターコンテナが実行されるノードなど) のネットワークスタックを使用します。このデフォルトの動作には、リモートクライアントからのネットワークトラフィックがターゲットサービスとコンテナに到達するためにユーザー空間で複数のホップを使用する必要がないので、パフォーマンス上のメリットがあります。

さらに、このデフォルト動作によってルーターはノードの IP アドレスではなくリモート接続の実際のソース IP アドレスを取得できます。これは、発信元の IP に基づいて ingress ルールを定義し、ステッキーセッションをサポートし、他に使用されているものの中でトラフィックを監視するのに役立ちます。

このホストネットワークの動作は **--host-network** ルーターコマンドラインオプションによって制御でき、デフォルトの動作は **--host-network=true** を使用した場合と等しくなります。コンテナネットワークスタックを使用してルーターを実行する場合は、ルーターを作成する際に **--host-network=false** オプションを使用します。以下に例を示します。

```
$ oc adm router --service-account=router --host-network=false
```

内部的には、これは外部ネットワークがルーターと通信するために、ルーターコンテナが 80 と 443 ポートを公開する必要があることを意味します。



#### 注記

コンテナネットワークスタックを使用して実行することで、ルーターは接続のソース IP アドレスを実際のリモート IP アドレスではなくノードの NAT された IP アドレスとして扱うことを意味します。



#### 注記

[マルチテナントネットワークの分離](#) を使用する OpenShift Container Platform クラスターでは、**--host-network=false** オプションを指定したデフォルト以外の namespace のルーターはクラスターのすべてのルートを読み込みますが、ネットワークの分離により複数の namespace にあるルートには到達できません。**--host-network=true** オプションを指定すると、ルートはコンテナネットワークをバイパスし、クラスターの任意の Pod にアクセスできます。この場合、分離が必要な場合は、複数の namespace のルートを追加しないでください。

### 3.2.21. Dynamic Configuration Manager の使用

HAProxy ルーターを Dynamic Configuration Manager を使用するように設定できます。

Dynamic Configuration Manager は、HAProxy リロードのダウンタイムなしに特定のタイプのルートオンラインにします。これは、ルートおよびエンドポイントの **addition|deletion|update** など、ルートおよびエンドポイントのすべてのライフサイクルイベントを処理します。

**ROUTER\_HAPROXY\_CONFIG\_MANAGER** 環境変数を **true** に設定して Dynamic Configuration Manager を有効にします。

```
$ oc set env dc/<router_name> ROUTER_HAPROXY_CONFIG_MANAGER='true'
```

Dynamic Configuration Manager が HAProxy を動的に設定できない場合、これは設定を再作成し、HAProxy プロセスをリロードします。これには、新規ルートにカスタムタイムアウトなどのカスタムアノテーションが含まれる場合や、ルートにカスタム TLS 設定が必要な場合などが含まれます。

動的な設定は、事前に割り当てられたルートおよびバックエンドサーバーのプールと共に HAProxy ソケットおよび設定 API を内部で使用します。ルートの事前に割り当てられたプールは、ルートのブループリントを使用して作成されます。ブループリントのデフォルトセットはセキュリティー保護のないルート、カスタム TLS 設定のないセキュリティー保護された edge ルート、および passthrough ルートをサポートします。

### 重要

**re-encrypt** ルートにはカスタム TLS 設定情報が必要であるため、Dynamic Configuration Manager でそれらを使用するには追加の設定が必要になります。

**ROUTER\_BLUEPRINT\_ROUTE\_NAMESPACE** を設定し、オプションで **ROUTER\_BLUEPRINT\_ROUTE\_LABELS** 環境変数を設定することで Dynamic Configuration Manager が使用できるブループリントを拡張します。

ブループリントルート namespace のすべてのルート、またはルートラベルに一致するルートは、ブループリントのデフォルトセットに似たカスタムブループリントとして処理されます。これには、**re-encrypt** ルートやカスタムアノテーションを使用するルート、またはカスタム TLS 設定のあるルートが含まれます。

以下の手順では、**reencrypt-blueprint**、**annotated-edge-blueprint**、および **annotated-unsecured-blueprint** の3つのルートオブジェクトがあることを前提としています。各種のルートタイプオブジェクトについては、[ルートタイプ](#)を参照してください。

### 手順

1. 新しいプロジェクトを作成します。

```
$ oc new-project namespace_name
```

2. 新規ルートを作成します。この方法では既存サービスを公開します。

```
$ oc create route edge edge_route_name --key=/path/to/key.pem \
  --cert=/path/to/cert.pem --service=<service> --port=8443
```

3. ルートにラベルを付けます。

```
$ oc label route edge_route_name type=route_label_1
```

4. ルートオブジェクト定義から3つの異なるルートを作成します。すべてにラベル `type=route_label_1` が付けられます。

```
$ oc create -f reencrypt-blueprint.yaml
$ oc create -f annotated-edge-blueprint.yaml
$ oc create -f annotated-unsecured-blueprint.json
```

また、ブループリントルートとしての使用を防ぐラベルをルートから削除することもできます。たとえば、**annotated-unsecured-blueprint** をブループリントルートとして使用されることを防ぐには、以下を実行します。

```
$ oc label route annotated-unsecured-blueprint type-
```

5. ブループリントプールに用される新規のルーターを作成します。

```
$ oc adm router
```

6. 新規ルーターの環境変数を設定します。

```
$ oc set env dc/router ROUTER_HAPROXY_CONFIG_MANAGER=true \
    ROUTER_BLUEPRINT_ROUTE_NAMESPACE=namespace_name \
    ROUTER_BLUEPRINT_ROUTE_LABELS="type=route_label_1"
```

ラベル `type=route_label_1` が設定された namespace またはプロジェクト `namespace_name` のすべてのルートはカスタムブループリントとして処理でき、使用できます。

ブループリントは、namespace `namespace_name` で通常実行するようにルートを管理することによって、追加し、更新し、削除できることに注意してください。Dynamic Configuration Manager は、ルーターが **routes** および **services** の有無を監視するのと同様の方法で namespace `namespace_name` のルートへの変更の有無を監視します。

7. 事前に割り当てられたルートおよびバックエンドサーバーのプールサイズは、**ROUTER\_BLUEPRINT\_ROUTE\_POOL\_SIZE** (デフォルトは **10**)、および **ROUTER\_MAX\_DYNAMIC\_SERVERS** (デフォルトは **5**) 環境変数で制御できます。また、Dynamic Configuration Manager が加える変更をディスクにコミットする頻度、つまり HAProxy 設定が再作成され、HAProxy プロセスがリロードされるタイミングを制御することもできます。デフォルトは1時間 (3600 秒) または Dynamic Configuration Manager のプールスペースが不足するタイミングになります。**COMMIT\_INTERVAL** 環境変数がこの設定を制御します。

```
$ oc set env dc/router -c router ROUTER_BLUEPRINT_ROUTE_POOL_SIZE=20 \
    ROUTER_MAX_DYNAMIC_SERVERS=3 COMMIT_INTERVAL=6h
```

この例では、各ブループリントルートのプールサイズを **20** に増やし、動的サーバーの数を **3** に減らし、またコミット期間を **6** 時間に増やしています。

### 3.2.22. ルーターメトリクスの公開

**HAProxy ルーターメトリクス** は、外部メトリクス収集および集約システム (例: Prometheus、statsd) で使用されるようにデフォルトで **Prometheus 形式** で公開されます。メトリクスは独自の HTML 形式でブラウザーで閲覧したり CSV ダウンロードを実行するために **HAProxy ルーター** から直接利用することもできます。これらのメトリクスには、HAProxy ネイティブメトリクスおよび一部のコントローラーメトリクスが含まれます。

以下のコマンドを使用してルーターを作成する場合、OpenShift Container Platform は Prometheus 形式のメトリクスをデフォルトが 1936 の統計ポートで利用可能にします。

```
$ oc adm router --service-account=router
```

- Prometheus 形式で未加工統計を抽出するには、以下を実行します。

```
curl <user>:<password>@<router_IP>:<STATS_PORT>
```

以下に例を示します。

```
$ curl admin:sLzdR6SgDJ@10.254.254.35:1936/metrics
```

メトリクスにアクセスするために必要な情報は、ルーターサービスのアノテーションで確認できます。

```
$ oc edit service <router-name>
```

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/port: "1936"
    prometheus.io/scrape: "true"
    prometheus.openshift.io/password: llmoDqON02
    prometheus.openshift.io/username: admin
```

**prometheus.io/port** はデフォルトが 1936 の統計ポートです。アクセスを許可するようファイウォールを設定する必要がある場合があります。直前のユーザー名およびパスワードを使用してメトリクスにアクセスします。パスは **/metrics** です。

```
$ curl <user>:<password>@<router_IP>:<STATS_PORT>
for example:
$ curl admin:sLzdR6SgDJ@10.254.254.35:1936/metrics
...
# HELP haproxy_backend_connections_total Total number of connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route-alt"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers tracked and the current threshold value.
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
```

```
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_no_sni",service=""}
0
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="default",pod="docker-registry-5-
nk5fz",route="docker-registry",server="10.130.0.89:5000",service="docker-registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-vkjqx",route="hello-
route",server="10.130.0.90:8080",service="hello-svc-1"} 0
...
```

- ブラウザーでメトリクスを取得するには、以下を実行します。
  1. 以下の [環境変数](#) をルーターデプロイメント設定ファイルから削除します。

```
$ oc edit dc router
- name: ROUTER_LISTEN_ADDR
  value: 0.0.0.0:1936
- name: ROUTER_METRICS_TYPE
  value: haproxy
```

2. HAProxy ルーターによって提供されるため、ルーターの readiness probe にパッチを適用し、これが liveness probe と同じパスを使用するようにします。

```
$ oc patch dc router -p "'spec': {'template': {'spec': {'containers': [{'name':
'router','readinessProbe': {'httpGet': {'path': '/healthz'}}]}}}'
```

3. ブラウザーで以下の URL を使用して統計ウィンドウを起動します。ここでは、**STATS\_PORT** 値はデフォルトで **1936** になります。

```
http://admin:<Password>@<router_IP>:<STATS_PORT>
```

;**csv** を URL に追加して CVS 形式の統計を取得できます。

以下に例を示します。

```
http://admin:<Password>@<router_IP>:1936;csv
```

ルーター IP、管理者名、およびパスワードを取得するには、以下を実行します。

```
oc describe pod <router_pod>
```

- メトリクスのコレクションを表示しないようにするには、以下を実行します。

```
$ oc adm router --service-account=router --stats-port=0
```

### 3.2.23. 大規模クラスターの ARP キャッシュのチューニング

(`net.ipv4.neigh.default.gc_thresh3` の値 (デフォルトで **65536**) を上回る) 多数のルートを持つ OpenShift Container Platform クラスターでは、ARP キャッシュでより多くのエントリを許可するた

めにルーター Pod を実行するクラスターの各ノードで `sysctl` 変数のデフォルト値を増やす必要があります。

問題が発生している場合、以下のようなカーネルメッセージが表示されます。

```
[ 1738.811139] net_ratelimit: 1045 callbacks suppressed
[ 1743.823136] net_ratelimit: 293 callbacks suppressed
```

この問題が発生すると、`oc` コマンドは以下のエラーを出して失敗することがあります。

```
Unable to connect to the server: dial tcp: lookup <hostname> on <ip>:<port>: write udp <ip>:<port>->
<ip>:<port>: write: invalid argument
```

IPv4 の ARP エントリーの実際の量を確認するには、以下を実行します。

```
# ip -4 neigh show nud all | wc -l
```

数字が `net.ipv4.neigh.default.gc_thresh3` しきい値に近づき始めたら、値を増やします。以下を実行して現在値を取得します。

```
# sysctl net.ipv4.neigh.default.gc_thresh1
net.ipv4.neigh.default.gc_thresh1 = 128
# sysctl net.ipv4.neigh.default.gc_thresh2
net.ipv4.neigh.default.gc_thresh2 = 512
# sysctl net.ipv4.neigh.default.gc_thresh3
net.ipv4.neigh.default.gc_thresh3 = 1024
```

以下の `sysctl` は変数を OpenShift Container Platform の現在のデフォルト値に設定します。

```
# sysctl net.ipv4.neigh.default.gc_thresh1=8192
# sysctl net.ipv4.neigh.default.gc_thresh2=32768
# sysctl net.ipv4.neigh.default.gc_thresh3=65536
```

これらの設定を永続化するには、[このドキュメント](#) を参照してください。

### 3.2.24. DDoS 攻撃からの保護

`timeout http-request` をデフォルトの HAProxy ルーターイメージに追加して、分散型の denial-of-service (DDoS) 攻撃 (slowloris など) からデプロイメントを保護します。

```
# and the haproxy stats socket is available at /var/run/haproxy.stats
global
  stats socket ./haproxy.stats level admin

defaults
  option http-server-close
  mode http
  timeout http-request 5s
  timeout connect 5s 1
  timeout server 10s
  timeout client 30s
```

- 1 `timeout http-request` は最大 5 秒に設定されます。HAProxy は HTTP 要求全体の送信のための 5 秒をクライアントに対して指定します。この指定がないと、HAProxy はエラーを出して接続を切断します。

また、環境変数 `ROUTER_SLOWLORIS_TIMEOUT` が設定されている場合、クライアントが HTTP 要求全体を送信するためにかかる合計時間が制限されます。これが設定されていない場合、HAProxy は接続を切断します。

環境変数を設定することで、情報をルーターのデプロイメント設定の一部として取得でき、テンプレートを手動で変更することが不要になります。一方、HAProxy 設定を手動で追加すると、ルーター Pod の再ビルドとルーターテンプレートファイルの保守が必要になります。

アノテーションを使用して、以下を制限する機能を含む基本的な DDoS 保護を HAProxy テンプレートルーターに実装します。

- 同時 TCP 接続の数
- クライアントが TCP 接続を要求できるレート
- HTTP 要求を実行できるレート

アプリケーションによってはトラフィックのパターンが完全に異なる場合があるため、これらはルートごとに有効にされます。

表3.1 HAProxy テンプレートルーター設定

設定	説明
<code>haproxy.router.openshift.io/rate-limit-connections</code>	設定した内容を有効にします (true に設定した場合など)。
<code>haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp</code>	このルートの同じ IP アドレスで接続できる同時 TCP 接続の数。
<code>haproxy.router.openshift.io/rate-limit-connections.rate-tcp</code>	クライアント IP で開くことができる TCP 接続の数。
<code>haproxy.router.openshift.io/rate-limit-connections.rate-http</code>	クライアント IP が 3 秒間で実行できる HTTP 要求の数。

### 3.2.25. HAProxy スレッドの有効化

`--threads` フラグを使用してスレッドを有効にします。このフラグは、HAProxy ルーターが使用するスレッド数を指定します。

## 3.3. カスタマイズされた HAPROXY ルーターのデプロイ

### 3.3.1. 概要

デフォルトの HAProxy ルーターは多数のユーザーのニーズに対応することを目的としています。ただし、このルーターは HAProxy のすべての機能を公開している訳ではありません。そのため、ユーザーはそれぞれのニーズに合わせてルーターを変更する必要があります。

新機能をアプリケーションバックエンド内で実装したり、現在の操作を変更する必要がある場合があります。ルータープラグインはこのカスタマイズを行うために必要なすべての機能を提供します。

ルーター Pod はテンプレートファイルを使用して必要な HAProxy 設定ファイルを作成します。テンプレートファイルは [golang テンプレート](#) です。テンプレートを処理する際に、ルーターはルーターのデプロイメント設定、許可されたルートのセット、一部のヘルパー機能などの OpenShift Container Platform 情報にアクセスします。

ルーター Pod が起動し、リロードされるたびに、HAProxy 設定ファイルが作成され、HAProxy が起動します。[HAProxy 設定マニュアル](#) には HAProxy のすべての機能と有効な設定ファイルの作成方法が記載されています。

[configMap](#) を使用して新規テンプレートをルーター Pod に追加することができます。この方法により、ルーターデプロイメント設定を変更して、[configMap](#) をルーター Pod のボリュームとしてマウントできます。`TEMPLATE_FILE` 環境変数はルーター Pod のテンプレートファイルのフルパス名に設定されます。



### 重要

OpenShift Container Platform のアップグレード後に、ルーターテンプレートのカスタマイズが引き続き機能する訳ではありません。

また、ルーターテンプレートのカスタマイズは、実行中のルーターのテンプレートバージョンに適用する必要があります。

または、カスタムルーターイメージをビルドし、ルーターの一部またはすべてをデプロイする際にこれを使用することができます。すべてのルーターが同じイメージを実行する必要はありません。これを実行するには、`haproxy-template.config` ファイルを変更し、ルーターイメージを [再ビルド](#) します。新しいイメージはクラスターの Docker リポジトリにプッシュされ、ルーターのデプロイメント設定の `image:` フィールドが新しい名前でも更新されます。クラスターが更新されたら、イメージを再ビルドし、プッシュする必要があります。

いずれの場合でも、ルーター Pod はテンプレートファイルを使用して起動します。

### 3.3.2. ルーター設定テンプレートの取得

HAProxy テンプレートファイルはかなり大きく複雑です。一部を変更するのであれば、すべてを書き換えるよりも既存のテンプレートを変更する方が簡単です。マスターでルーターを実行し、ルーター Pod を参照することで実行中のルーターから `haproxy-config.template` ファイルを取得できます。

```
# oc get po
NAME                READY  STATUS   RESTARTS  AGE
router-2-40fc3      1/1    Running  0          11d
# oc exec router-2-40fc3 cat haproxy-config.template > haproxy-config.template
# oc exec router-2-40fc3 cat haproxy.config > haproxy.config
```

または、ルーターを実行しているノードにログオンします。

```
# docker run --rm --interactive=true --tty --entrypoint=cat \
  registry.redhat.io/openshift3/ose-haproxy-router:v{product-version} haproxy-config.template
```

イメージ名は [コンテナイメージ](#) から取られます。

この内容をカスタマイズされたテンプレートのベースとして使用するためにファイルに保存します。保存された `haproxy.config` は実際に実行されているものを示します。

### 3.3.3. ルーター設定テンプレートの変更

#### 3.3.3.1. 背景情報

このテンプレートは [golang テンプレート](#) に基づいています。これは、ルーターのデプロイメント設定の環境変数や、以下に示す設定情報およびルーターが提供するヘルパー機能を参照することができます。

テンプレートファイルの構造は作成される HAProxy 設定ファイルを反映します。テンプレートの処理時に、`{{" something "}}` によって囲まれていないものはすべて設定ファイルに直接コピーされます。`{{" something "}}` で囲まれている部分は評価されます。生成されるテキスト (ある場合) は設定ファイルにコピーされます。

#### 3.3.3.2. Go テンプレートアクション

`define` アクションは、処理されるテンプレートを含むファイルに名前を付けます。

```
{{define "/var/lib/haproxy/conf/haproxy.config"}}pipeline{{end}}
```

表3.2 テンプレートルーター関数

関数	意味
<code>processEndpointsForAlias(alias ServiceAliasConfig, svc ServiceUnit, action string) []Endpoint</code>	有効なエンドポイントの一覧を返します。アクションが <code>Shuffle</code> の場合、エンドポイントの順序はランダム化されます。
<code>env(variable, default ...string)string</code>	Pod からの名前付き環境変数の取得を試行します。これが定義されていないか、または空の場合、オプションの2つ目の引数が返されます。それ以外の場合には空の文字列を返します。
<code>matchPattern(pattern, s string) bool</code>	1つ目の引数は正規表現を含む文字列で、2つ目の引数はテストに使用できる変数です。1つ目の引数として提供される正規表現が2つ目の引数として提供される文字列と一致するかどうかを示すブール値を返します。
<code>isInteger(s string) bool</code>	指定された変数が整数かどうかを判別します。
<code>firstMatch(s string, allowedValues ...string) bool</code>	所定の文字列を許可された文字列の一覧と比較します。左から右にスキャンし、最初の一致を返します。
<code>matchValues(s string, allowedValues ...string) bool</code>	所定の文字列を許可された文字列の一覧と比較します。文字列が許可される値の場合は <code>true</code> を返します。それ以外の場合は、 <code>false</code> を返します。
<code>generateRouteRegexp(hostname, path string, wildcard bool) string</code>	ルートホスト (とパス) に一致する正規表現を生成します。最初の引数はホスト名であり、2つ目はパス、3つ目はワイルドカードブール値です。

関数	意味
<b>genCertificateHostName(hostname string, wildcard bool) string</b>	証明書の提供/証明書のマッチングに使用するホスト名を生成します。1つ目の引数はホスト名で、2つ目はワイルドカードブール値です。
<b>isTrue(s string) bool</b>	所定の変数に true が含まれるかどうかを判別します。

これらの関数は、HAProxy テンプレートルータープラグインによって提供されます。

### 3.3.3.3. ルーターが提供する情報

このセクションでは、ルーターがテンプレートで利用可能にする OpenShift Container Platform の情報について説明します。ルーター設定パラメーターは HAProxy ルータープラグインに与えられるデータセットです。フィールドには **(dot).Fieldname** を使用してアクセスします。

以下のルーター設定パラメーター表は各種フィールドの定義を詳しく取り上げています。とくに **.State** には許可されたルートセットが設定されます。

表3.3 ルーター設定パラメーター

フィールド	タイプ	説明
<b>WorkingDir</b>	string	ファイルが書き込まれるディレクトリ。デフォルトは <code>/var/lib/containers/router</code> に設定されます。
<b>State</b>	<b>map[string]</b> <b>(ServiceAliasConfig)</b>	ルート。
<b>ServiceUnits</b>	<b>map[string]ServiceUnit</b>	サービスのルックアップ。
<b>DefaultCertificate</b>	string	pem 形式のデフォルト証明書へのフルパス名。
<b>PeerEndpoints</b>	<b>[]Endpoint</b>	ピア。
<b>StatsUser</b>	string	統計の公開に使用するユーザー名 (テンプレートがサポートしている場合)。
<b>StatsPassword</b>	string	統計の公開に使用するパスワード (テンプレートがサポートしている場合)。
<b>StatsPort</b>	int	統計の公開に使用するポート (テンプレートがサポートしている場合)。

フィールド	タイプ	説明
<b>BindPorts</b>	bool	ルーターがデフォルトポートをバインドすべきかどうか。

表3.4 ルーター ServiceAliasConfig (Route)

フィールド	タイプ	説明
<b>Name</b>	string	ルートのユーザー固有の名前。
<b>Namespace</b>	string	ルートの namespace。
<b>Host</b>	string	ホスト名です。例: <b>www.example.com</b> 。
<b>Path</b>	string	オプションのパス。例: <b>www.example.com/myservice</b> (ここでは、 <b>myservice</b> がパスになります)。
<b>TLSTermination</b>	<b>routeapi.TLSTerminationType</b>	このバックエンドの終了ポリシー。マッピングファイルとルーター設定を利用します。
<b>Certificates</b>	<b>map[string]Certificate</b>	このバックエンドをセキュリティー保護するために使用する証明書。証明書 ID で指定します。
<b>Status</b>	<b>ServiceAliasConfigStatus</b>	永続化する必要がある設定のステータスを示します。
<b>PreferPort</b>	string	ユーザーが公開したいポートを示します。空の場合、サービスのポートが選択されます。
<b>InsecureEdgeTerminationPolicy</b>	<b>routeapi.InsecureEdgeTerminationPolicyType</b>	edge termination ルートへの非セキュアな接続の必要な動作を示します。 <b>none</b> (または <b>disable</b> )、 <b>allow</b> 、または <b>redirect</b>
<b>RoutingKeyName</b>	string	Cookie ID を隠すために使用するルート + namespace 名のハッシュ。
<b>IsWildcard</b>	bool	このサービスユニットがワイルドカードサポートを必要とすることを示します。

フィールド	タイプ	説明
<b>Annotations</b>	<b>map[string]string</b>	このルートに割り当てられるアノテーション。
<b>ServiceUnitNames</b>	<b>map[string]int32</b>	このルートをサポートするサービスコレクション。マップの他のエントリーに関連してサービス名で指定され、これに割り当てられた重みで評価されます。
<b>ActiveServiceUnits</b>	int	ゼロ以外の重みを持つ <b>ServiceUnitNames</b> の数。

**ServiceAliasConfig** はサービスのルートです。ホスト + パスによって特定されます。デフォルトのテンプレートは `{{range $cfgIdx, $cfg := .State }}` を使用してルートを繰り返し処理します。その `{{range}}` ブロック内で、テンプレートは `$cfg.Field` を使用して現在の **ServiceAliasConfig** のフィールドを参照できます。

表3.5 ルーター ServiceUnit

フィールド	タイプ	説明
<b>Name</b>	string	名前はサービス名 + namespace に対応します。 <b>ServiceUnit</b> を特定します。
<b>EndpointTable</b>	<b>[]Endpoint</b>	サービスをサポートするエンドポイント。これはルーターの最終のバックエンド実装に変換されます。

**ServiceUnit** はサービスをカプセル化したものであり、そのサービスをサポートするエンドポイントであり、またサービスを参照するルートです。これは、ルーター設定ファイルの作成の基になるデータです。

表3.6 ルーターエンドポイント

フィールド	型
<b>ID</b>	string
<b>IP</b>	string
<b>Port</b>	string
<b>TargetName</b>	string
<b>PortName</b>	string

フィールド	型
IdHash	string
NoHealthCheck	bool

**Endpoint** は、Kubernetes エンドポイントの内部表現です。

表3.7 ルーター証明書、ServiceAliasConfigStatus

フィールド	タイプ	説明
<b>Certificate</b>	string	パブリック/プライベートキーのペアを表します。これはIDで識別され、ファイル名になります。CA 証明書には <b>PrivateKey</b> が設定されません。
<b>ServiceAliasConfigStatus</b>	string	この設定に必要なファイルがディスクに永続化されていることを示します。有効な値の例: "saved" または "" (ブランク)。

表3.8 ルーター証明書タイプ

フィールド	タイプ	説明
ID	string	
内容	string	証明書。
PrivateKey	string	プライベートキー。

表3.9 ルーター TLSTerminationType

フィールド	タイプ	説明
<b>TLSTerminationType</b>	string	セキュアな通信が停止する場合について指示します。
<b>InsecureEdgeTerminationPolicyType</b>	string	ルートへの非セキュアな接続に必要な動作を示します。各ルーターは公開するポートを独自に決定することがありますが、通常はポート 80 になります。

**TLSTerminationType** と **InsecureEdgeTerminationPolicyType** はセキュアな通信が停止する場合について指示します。

表3.10 ルーター TLSTerminationType 値

Constant	値	意味
<b>TLSTerminationEdge</b>	<b>edge</b>	edge ルーターでの暗号化を終了します。
<b>TLSTerminationPassthrough</b>	パススルー	宛先での暗号化を終了し、宛先でトラフィックを復号化します。
<b>TLSTerminationReencrypt</b>	<b>reencrypt</b>	edge ルーターで暗号化を終了し、宛先で提供される新規の証明書を使用して再暗号化します。

表3.11 ルーター InsecureEdgeTerminationPolicyType 値

型	意味
<b>Allow</b>	トラフィックは非セキュアなポートのサーバーに送信されます (デフォルト)。
<b>Disable</b>	トラフィックは非セキュアなポートでは許可されません。
<b>Redirect</b>	クライアントはセキュアなポートにリダイレクトされます。

なし ("" ) は **Disable** と同じです。

#### 3.3.3.4. アノテーション

各ルートにはアノテーションを割り当てることができます。各アノテーションは名前と値のみで設定されます。

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms
  [...]
```

名前は既存のアノテーションと競合しないものにできます。値は文字列です。文字列では複数のトークンをスペースで区切ることができます (例: **aa bb cc**)。このテンプレートは **{{index}}** を使用してアノテーションの値を抽出します。以下に例を示します。

```
{{ $balanceAlgo := index $cfg.Annotations "haproxy.router.openshift.io/balance" }}
```

この例は、これをどのようにクライアントの相互認証に使用できるかを示しています。

```
{{ with $cnList := index $cfg.Annotations "whiteListCertCommonName" }}
```

```

{{ if ne $cnList "" }}
  acl test ssl_c_s_dn(CN) -m str {{ $cnList }}
  http-request deny if !test
{{ end }}
{{ end }}

```

このコマンドを使用してホワイトリストの CN を処理できます。

```
$ oc annotate route <route-name> --overwrite whiteListCertCommonName="CN1 CN2 CN3"
```

詳細は、[ルート固有のアノテーション](#) を参照してください。

### 3.3.3.5. 環境変数

テンプレートはルーター Pod に存在する任意の環境変数を使用できます。環境変数はデプロイメント設定に設定できます。また、新規の環境変数を追加できます。

これらは **env** 関数で参照されます。

```
{{env "ROUTER_MAX_CONNECTIONS" "20000"}}
```

1つ目の文字列は変数であり、変数がないか、または **nil** の場合には2つ目の文字列がデフォルトになります。**ROUTER\_MAX\_CONNECTIONS** が設定されていないか、または **nil** の場合、20000 が使用されます。環境変数はマップと言えます。ここでキーは環境変数名で、内容は変数の値になります。

詳細は、[ルート固有の環境変数](#) を参照してください。

### 3.3.3.6. 使用例

以下で HAProxy テンプレートファイルに基づく単純なテンプレートを示します。

以下のコメントで開始します。

```

{*/
  Here is a small example of how to work with templates
  taken from the HAProxy template file.
*/}

```

テンプレートは任意の数の出力ファイルを作成できます。define コンストラクトを使用して出力ファイルを作成します。ファイル名は定義する引数として指定され、define ブロック内の一致する終了部分までのすべての情報がそのファイルの内容として書き込まれます。

```

{{ define "/var/lib/haproxy/conf/haproxy.config" }}
  global
{{ end }}

```

上記は **global** を `/var/lib/haproxy/conf/haproxy.config` ファイルにコピーし、ファイルを閉じます。

ロギングを環境変数に基づいてセットアップします。

```

{{ with (env "ROUTER_SYSLOG_ADDRESS" "") }}
  log {{.}} {{env "ROUTER_LOG_FACILITY" "local1"}} {{env "ROUTER_LOG_LEVEL" "warning"}}
{{ end }}

```

**env** 関数は、環境変数の値を抽出します。環境変数が定義されていないか、または **nil** の場合、2 つ目の引数が返されます。

with コンストラクトは with ブロック内の "." (ドット) の値を with に引数として提供される値に設定します。with アクションは Dot で **nil** をテストします。**nil** ではない場合、その句は **end** (終了部分) まで処理されます。上記では、**ROUTER\_SYSLOG\_ADDRESS** に `/var/log/msg` が含まれ、**ROUTER\_LOG\_FACILITY** が定義されておらず、**ROUTER\_LOG\_LEVEL** に **info** が含まれていると想定しています。以下が出力ファイルにコピーされます。

```
log /var/log/msg local1 info
```

許可された各ルートは設定ファイルの行を生成します。**range** を使用して、許可されたルートを確認します。

```
{{ range $cfgIdx, $cfg := .State }}
  backend be_http_{{ $cfgIdx }}
{{ end }}
```

**.State** は **ServiceAliasConfig** のマップです。ここでは、キーはルート名です。**range** はマップを通じて、パスするたびに **key** を使用して **\$cfgIdx** を設定し、**\$cfg** がルートを記述する **ServiceAliasConfig** をポイントするよう設定します。**myroute** と **hisroute** という 2 つのルートがある場合、上記により以下が出力ファイルにコピーされます。

```
backend be_http_myroute
backend be_http_hisroute
```

ルートアノテーション **\$cfg.Annotations** もマップであり、アノテーション名がキーとなり、コンテンツの文字列が値になっています。ルートは必要な数だけアノテーションを持つことができ、テンプレートの作成者が使用方法を定義します。ユーザーはアノテーションをルートにコード化し、テンプレート作成者は HAProxy テンプレートをカスタマイズしてそのアノテーションを処理できるようにします。

通常はアノテーションをインデックス化して値を取得します。

```
{{ $balanceAlgo := index $cfg.Annotations "haproxy.router.openshift.io/balance" }}
```

インデックスは指定されたアノテーションの値を抽出します。そのため、**\$balanceAlgo** はアノテーションまたは **nil** に関連付けられた文字列が含まれます。上記のように、**nil** 以外の文字列についてテストし、**with** コンストラクトを使用して影響を確認することができます。

```
{{ with $balanceAlgo }}
  balance $balanceAlgo
{{ end }}
```

**\$balanceAlgo** が **nil** でない場合、**balance \$balanceAlgo** は出力ファイルにコピーされます。

2 つ目の例では、アノテーションのタイムアウト値の設定に基づいてサーバータイムアウトを設定します。

```
$value := index $cfg.Annotations "haproxy.router.openshift.io/timeout"
```

**\$value** を評価して、適切に作成された文字列が含まれていることを確認できます。**matchPattern** 関数は正規表現を受け入れ、引数が表現を満たしていれば **true** を返します。

```
matchPattern "[1-9][0-9]*(us|ms|s|m|h|d)?" $value
```

これにより **5000ms** を受け入れますが、**7y** は受け取りません。この結果はテストで使用できます。

```
{{if (matchPattern "[1-9][0-9]*(us|ms|s|m|h|d)?" $value) }}
  timeout server {{$value}}
{{ end }}
```

これを使用してトークンに一致させることもできます。

```
matchPattern "roundrobin|leastconn|source" $balanceAlgo
```

または、**matchValues** を使用してトークンと一致させることもできます。

```
matchValues $balanceAlgo "roundrobin" "leastconn" "source"
```

### 3.3.4. ConfigMap を使用してルーター設定テンプレートを置き換える

**ConfigMap** を使用して、ルーターイメージを再ビルドせずにルーターインスタンスをカスタマイズできます。ルーター環境変数の作成し、変更することができるだけでなく、**haproxy-config.template**、**reload-haproxy** その他のスクリプトを変更することもできます。

1. [上記のように](#) 変更する **haproxy-config.template** をコピーします。必要に応じて変更します。
2. **ConfigMap** を作成します。

```
$ oc create configmap customrouter --from-file=haproxy-config.template
```

**customrouter** **ConfigMap** には変更された **haproxy-config.template** ファイルのコピーが含まれています。

3. ルーターデプロイメント設定を変更し、**ConfigMap** をファイルとしてマウントし、**TEMPLATE\_FILE** 環境変数がこれをポイントするようにします。これは、**oc set env** と **oc set volume** コマンドを使用するか、またはルーターデプロイメント設定を編集して実行できます。

#### oc コマンドの使用

```
$ oc set volume dc/router --add --overwrite \
  --name=config-volume \
  --mount-path=/var/lib/haproxy/conf/custom \
  --source='{"configMap": {"name": "customrouter"}}'
$ oc set env dc/router \
  TEMPLATE_FILE=/var/lib/haproxy/conf/custom/haproxy-config.template
```

#### ルーターデプロイメント設定の編集

**oc edit dc router** を使用して、テキストエディターでルーターデプロイメント設定を編集します。

```
...
- name: STATS_USERNAME
  value: admin
- name: TEMPLATE_FILE 1
```

```

    value: /var/lib/haproxy/conf/custom/haproxy-config.template
    image: openshift/origin-haproxy-routerp
  ...
  terminationMessagePath: /dev/termination-log
  volumeMounts: ❷
  - mountPath: /var/lib/haproxy/conf/custom
    name: config-volume
  dnsPolicy: ClusterFirst
  ...
  terminationGracePeriodSeconds: 30
  volumes: ❸
  - configMap:
    name: customrouter
    name: config-volume
  ...

```

- ❶ **spec.container.env** フィールドに **TEMPLATE\_FILE** 環境変数を追加して、マウントされた **haproxy-config.template** ファイルをポイントするようにします。
- ❷ **spec.container.volumeMounts** フィールドを追加して、マウントポイントを作成します。
- ❸ 新しい **spec.volumes** フィールドを追加し、ConfigMap を示唆します。

変更を保存し、エディターを終了します。ルーターが再起動します。

### 3.3.5. Stick Table の使用

以下のカスタマイズの例を [高可用性ルーティングセットアップ](#) で使用することで、ピア間の同期を行う `stick-table` を使用できます。

#### ピアセクションの追加

ピア間で `stick-table` を同期するには、HAProxy 設定でピアセクションを定義する必要があります。このセクションによって、HAProxy がどのようにピアを識別し、接続するかが決まります。プラグインはデータを **.PeerEndpoints** 変数にあるテンプレートに提供するので、ルーターサービスのメンバーを簡単に識別できます。以下を追加することで、ピアセクションをルーターイメージ内の `haproxy-config.template` ファイルに追加することができます。

```

{{ if (len .PeerEndpoints) gt 0 }}
peers openshift_peers
  {{ range $endpointID, $endpoint := .PeerEndpoints }}
    peer {{$endpoint.TargetName}} {{$endpoint.IP}}:1937
  {{ end }}
{{ end }}

```

#### リロードスクリプトの変更

`stick-table` を使用する場合、HAProxy にピアセクションでローカルホスト名として見なすものをオプションで指示することができます。エンドポイントの作成時に、プラグインは **TargetName** をエンドポイントの **TargetRef.Name** の値に設定するよう試みます。 **TargetRef** が設定されていない場

合、**TargetName** は IP アドレスに設定されます。**TargetRef.Name** は Kubernetes ホスト名に対応しているため、**-L** オプションを **reload-haproxy** スクリプトに追加してローカルホストをピアセクションで識別できます。

```
peer_name=$HOSTNAME ❶

if [ -n "$old_pid" ]; then
  /usr/sbin/haproxy -f $config_file -p $pid_file -L $peer_name -sf $old_pid
else
  /usr/sbin/haproxy -f $config_file -p $pid_file -L $peer_name
fi
```

❶ ピアセクションで使用されるエンドポイントターゲット名と一致する必要があります。

### バックエンドの変更

最後に、stick-table をバックエンド内で使用するために、HAProxy 設定を変更して stick-table およびピアセットを使用することができます。以下は、stick-table を使用するように TCP 接続の既存のバックエンドを変更している例です。

```
        {{ if eq $cfg.TLSTermination "passthrough" }}
backend be_tcp_{{ $cfgIdx }}
  balance leastconn
  timeout check 5000ms
  stick-table type ip size 1m expire 5m{{ if (len $.PeerEndpoints) gt 0 }} peers openshift_peers {{ end }}
stick on src
  {{ range $endpointID, $endpoint := $serviceUnit.EndpointTable }}
server {{ $endpointID }} {{ $endpoint.IP }}:{{ $endpoint.Port }} check inter 5000ms
  {{ end }}
{{ end }}
```

この変更を行った後に、[ルーターを再ビルド](#) できます。

### 3.3.6. ルーターの再ビルド

ルーターを再ビルドするには、実行中のルーターにある複数のファイルのコピーが必要になります。作業ディレクトリーを作成し、ルーターからファイルをコピーします。

```
# mkdir -p myrouter/conf
# cd myrouter
# oc get po
NAME          READY  STATUS   RESTARTS  AGE
router-2-40fc3 1/1    Running  0          11d
# oc exec router-2-40fc3 cat haproxy-config.template > conf/haproxy-config.template
# oc exec router-2-40fc3 cat error-page-503.http > conf/error-page-503.http
# oc exec router-2-40fc3 cat default_pub_keys.pem > conf/default_pub_keys.pem
# oc exec router-2-40fc3 cat ../Dockerfile > Dockerfile
# oc exec router-2-40fc3 cat ../reload-haproxy > reload-haproxy
```

これらのファイルのいずれも編集するか、または置き換えることができます。ただし、**conf/haproxy-config.template** と **reload-haproxy** が変更される可能性が高くなります。

ファイルの更新後:

```
# docker build -t openshift/origin-haproxy-router-myversion .  
# docker tag openshift/origin-haproxy-router-myversion 172.30.243.98:5000/openshift/haproxy-router-  
myversion ①  
# docker push 172.30.243.98:5000/openshift/origin-haproxy-router-pc:latest ②
```

- ① このバージョンをリポジトリでタグ付けします。この場合、リポジトリは **172.30.243.98:5000** になります。
- ② タグ付けバージョンをリポジトリにプッシュします。まずリポジトリに **docker ログイン** する必要がある場合があります。

新規ルーターを使用するには、**image:** 文字列を変更するか、または **--images=<repo>/<image>:<tag>** フラグを **oc adm router** コマンドに追加してルーターデプロイ設定を編集します。

変更のデバッグ時に、デプロイメント設定で **imagePullPolicy: Always** を設定して各 Pod の作成時にイメージプルを強制的に実行すると便利です。デバッグが完了したら、これを **imagePullPolicy: IfNotPresent** に戻して各 Pod の起動時のプルを回避します。

## 3.4. PROXY プロトコルを使用するように HAPROXY ルーターを設定する

### 3.4.1. 概要

デフォルトで HAProxy ルーターは、非セキュアな edge および re-encrypt ルートへの受信接続に HTTP を使用することを想定しています。ただし、**PROXY プロトコル** を使用することで、ルーターが受信要求を予想するよう設定することができます。このトピックでは、HAProxy ルーターと外部ロードバランサーを PROXY プロトコルを使用するように設定する方法を説明しています。

### 3.4.2. PROXY プロトコルを使用する理由

プロキシサーバーやロードバランサーなどの中間サービスが HTTP 要求を転送する際に、これは接続のソースアドレスを要求の Forwarded ヘッダーに追加して、この情報を後続の中間サービスと、要求が最終的に転送されるバックエンドサービスに提供できます。ただし、接続が暗号化されている場合、中間サービスは Forwarded ヘッダーを変更できません。この場合、要求が転送されても HTTP ヘッダーは元のソースアドレスを正確に通信することができません。

この問題を解決するために、一部のロードバランサーは、単純に HTTP を転送する代替手段として PROXY プロトコルを使用して HTTP 要求をカプセル化します。カプセル化によって、ロードバランサーは転送される要求自体を変更することなく、情報を要求に追加することができます。これにより、ロードバランサーは、暗号化された接続を転送する場合でもソースアドレスを通信できます。

HAProxy ルーターが PROXY プロトコルを受け入れ、HTTP 要求のカプセル化を解除するように設定できます。ルーターは edge および re-encrypt ルートの暗号化を終了するので、ルーターは要求の ForwardedHTTP ヘッダー (および関連する HTTP ヘッダー) を更新でき、PROXY プロトコルを使用して通信されるソースアドレスを追加できます。



### 警告

PROXY プロトコルと HTTP は互換性がなく、組み合わせることはできません。ルーターの前にロードバランサーを使用する場合、これらがどちらも PROXY プロトコルまたは HTTP のいずれかを使用する必要があります。一方を PROXY プロトコルを使用するように設定し、他方を HTTP を使用するように設定すると、ルーティングが失敗します。

### 3.4.3. PROXY プロトコルの使用

デフォルトで、HAProxy ルーターは PROXY プロトコルを使用しません。**ROUTER\_USE\_PROXY\_PROTOCOL** 環境変数を使用することで、ルーターが受信接続に PROXY プロトコルの使用を予想するように設定できます。

#### PROXY プロトコルの有効化

```
$ oc set env dc/router ROUTER_USE_PROXY_PROTOCOL=true
```

変数を **true** または **TRUE** 以外の値に設定し、PROXY プロトコルを無効にします。

#### PROXY プロトコルの無効化

```
$ oc set env dc/router ROUTER_USE_PROXY_PROTOCOL=false
```

ルーターで PROXY プロトコルを有効にする場合、ルーターの前のロードバランサーが PROXY プロトコルを使用するように設定する必要があります。以下は、Amazon の Elastic Load Balancer (ELB) サービスを PROXY プロトコルを使用するように設定した例です。この例では、ELB がポート 80 (HTTP)、443 (HTTPS)、5000 (イメージレジストリーの場合) を 1 つ以上の EC2 インスタンスで実行されるルーターに転送することを想定しています。

#### Amazon ELB を設定して PROXY プロトコルを使用する

1. 後続の手順を単純化するために、最初に一部の Shell 変数を設定します。

```
$ lb='infra-lb' ①  
$ instances=( 'i-079b4096c654f563c' ) ②  
$ secgroups=( 'sg-e1760186' ) ③  
$ subnets=( 'subnet-cf57c596' ) ④
```

- ① ELB の名前。
- ② ルーターが実行されているインスタンス。
- ③ この ELB のセキュリティーグループ。
- ④ この ELB のサブネット。

2. 次に、適切なリスナーやセキュリティーグループおよびサブネットを使用して ELB を作成します。



## 注記

すべてのリスナーが HTTP プロトコルではなく TCP プロトコルを使用するように設定する必要があります。

```
$ aws elb create-load-balancer --load-balancer-name "$lb" \
--listeners \
'Protocol=TCP,LoadBalancerPort=80,InstanceProtocol=TCP,InstancePort=80' \
'Protocol=TCP,LoadBalancerPort=443,InstanceProtocol=TCP,InstancePort=443' \
'Protocol=TCP,LoadBalancerPort=5000,InstanceProtocol=TCP,InstancePort=5000' \
--security-groups $secgroups \
--subnets $subnets
{
  "DNSName": "infra-lb-2006263232.us-east-1.elb.amazonaws.com"
}
```

3. ルーターインスタンスを ELB に登録します。

```
$ aws elb register-instances-with-load-balancer --load-balancer-name "$lb" \
--instances $instances
{
  "Instances": [
    {
      "InstanceId": "i-079b4096c654f563c"
    }
  ]
}
```

4. ELB のヘルスチェックを設定します。

```
$ aws elb configure-health-check --load-balancer-name "$lb" \
--health-check
'Target=HTTP:1936/healthz,Interval=30,UnhealthyThreshold=2,HealthyThreshold=2,Timeout=5
,
{
  "HealthCheck": {
    "HealthyThreshold": 2,
    "Interval": 30,
    "Target": "HTTP:1936/healthz",
    "Timeout": 5,
    "UnhealthyThreshold": 2
  }
}
```

5. 最後に、**ProxyProtocol** 属性を有効にしたロードバランサーのポリシーを作成し、ELB の TCP ポート 80 および 443 でポリシーを設定します。

```
$ aws elb create-load-balancer-policy --load-balancer-name "$lb" \
--policy-name "${lb}-ProxyProtocol-policy" \
--policy-type-name 'ProxyProtocolPolicyType' \
--policy-attributes 'AttributeName=ProxyProtocol,AttributeValue=true'
$ for port in 80 443
do
  aws elb set-load-balancer-policies-for-backend-server \
```

```

--load-balancer-name "$lb" \
--instance-port "$port" \
--policy-names "${lb}-ProxyProtocol-policy"
done

```

## 設定の確認

ロードバランサーを以下のように検証して、設定が正しいことを確認します。

```

$ aws elb describe-load-balancers --load-balancer-name "$lb" |
jq '.LoadBalancerDescriptions| [.]|.ListenerDescriptions]'
[
  [
    {
      "Listener": {
        "InstancePort": 80,
        "LoadBalancerPort": 80,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": ["infra-lb-ProxyProtocol-policy"] ❶
    },
    {
      "Listener": {
        "InstancePort": 443,
        "LoadBalancerPort": 443,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": ["infra-lb-ProxyProtocol-policy"] ❷
    },
    {
      "Listener": {
        "InstancePort": 5000,
        "LoadBalancerPort": 5000,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": [] ❸
    }
  ]
]

```

- ❶ TCP ポート 80 のリスナーには PROXY プロトコルを使用するポリシーが設定されている必要があります。
- ❷ TCP ポート 443 のリスナーには同じポリシーが設定されている必要があります。
- ❸ TCP ポート 5000 のリスナーにはこのポリシーを設定できません。

または、ELB をすでに設定していても、PROXY プロトコルを使用するよう設定されていない場合は、TCP ポート 80 の既存リスナーを、HTTP ではなく TCP プロトコルを使用するように変更する必要があります (TCP ポート 443 はすでに TCP プロトコルを使用しているはずですが)。

```

$ aws elb delete-load-balancer-listeners --load-balancer-name "$lb" \

```

```
--load-balancer-ports 80
$ aws elb create-load-balancer-listeners --load-balancer-name "$lb" \
--listeners 'Protocol=TCP,LoadBalancerPort=80,InstanceProtocol=TCP,InstancePort=80'
```

## プロトコル更新の確認

プロトコルが以下のように更新されていることを確認します。

```
$ aws elb describe-load-balancers --load-balancer-name "$lb" |
jq '[.LoadBalancerDescriptions[]].ListenerDescriptions'
[
  [
    {
      "Listener": {
        "InstancePort": 443,
        "LoadBalancerPort": 443,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
    },
    {
      "Listener": {
        "InstancePort": 5000,
        "LoadBalancerPort": 5000,
        "Protocol": "TCP",
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
    },
    {
      "Listener": {
        "InstancePort": 80,
        "LoadBalancerPort": 80,
        "Protocol": "TCP", ①
        "InstanceProtocol": "TCP"
      },
      "PolicyNames": []
    }
  ]
]
```

- ① TCP ポート 80 のリスナーなど、すべてのリスナーが TCP プロトコルを使用している必要があります。

次にロードバランサーポリシーを作成し、上記の手順 5 に説明されているようにこれを ELB に追加します。

## 第4章 RED HAT CLOUDFORMS のデプロイ

### 4.1. RED HAT CLOUDFORMS の OPENSIFT CONTAINER PLATFORM へのデプロイ

#### 4.1.1. はじめに

OpenShift Container Platform インストーラーには、Red Hat CloudForms 4.6 (CloudForms Management Engine 5.9 または CFME) を OpenShift Container Platform にデプロイするための Ansible ロールの `openshift-management` と Playbook が含まれています。



#### 警告

現在の実装には、[OpenShift Container Platform 3.6 ドキュメント](#) で説明されているように、Red Hat CloudForms 4.5 のテクノロジープレビューのデプロイメントプロセスとの互換性はありません。

Red Hat CloudForms を OpenShift Container Platform にデプロイする際には、以下の2点について決定する必要があります。

1. 外部またはコンテナ化された (Pod 化された) PostgreSQL データベースを使用するかどうか。
2. 永続ボリューム (PV) をどのストレージクラスでサポートするか。

最初の点については、Red Hat CloudForms で使用する PostgreSQL データベースが置かれている場所によって Red Hat CloudForms を以下のいずれかの方法でデプロイできます。

デプロイのバリエーション	説明
完全コンテナ化	すべてのアプリケーションサービスと PostgreSQL データベースは、OpenShift Container Platform で Pod として実行されます。
外部データベース	アプリケーションは外部でホストされた PostgreSQL データベースサーバーを使用し、その他すべてのサービスは OpenShift Container Platform で Pod として実行されます。

2つ目の点については、`openshift-management` ロールが、多くのデフォルトのデプロイメントパラメーターの上書き用にカスタマイズオプションを提供します。これには、PV をサポートするための以下のストレージクラスのオプションが含まれています。

Storage Class	説明
NFS (デフォルト)	ローカル、クラスター上
NFS (外部)	NFS の他の場所、ストレージアプライアンスなど
クラウドプロバイダー	クラウドプロバイダー (Google Cloud Engine、Amazon Web Services、または Microsoft Azure) の自動ストレージプロビジョニングを使用
事前設定 (詳細)	ユーザーが事前にすべてを作成済みであることを前提とする

本書では、Red Hat CloudForms を OpenShift Container Platform で実行するための要件、利用可能な設定変数の説明、および OpenShift Container Platform の初回インストール時かクラスターのプロビジョニング後のいずれかでインストーラーを実行する方法などについてのトピックを扱います。

## 4.2. RED HAT CLOUDFORMS を OPENSIFT CONTAINER PLATFORM で使用するための要件

デフォルトの要件は以下の表に記載されています。これらは [テンプレートパラメーターをカスタマイズ](#) して上書きできます。



### 重要

以下の要件を満たしていないと、アプリケーションのパフォーマンスが低下するか、またはデプロイに失敗する可能性があります。

表4.1 デフォルトの要件

項目	要件	説明	カスタマイズパラメーター
アプリケーションメモリー	≥ 4.0 Gi	アプリケーションのメモリー最小要件	<b>APPLICATION_MEM_REQ</b>
アプリケーションストレージ	≥ 5.0 Gi	アプリケーションの PV サイズ最小要件	<b>APPLICATION_VOLUME_CAPACITY</b>
PostgreSQL メモリー	≥ 6.0 Gi	データベースのメモリー最小要件	<b>POSTGRESQL_MEM_REQ</b>
PostgreSQL ストレージ	≥ 15.0 Gi	データベースの PV サイズ最小要件	<b>DATABASE_VOLUME_CAPACITY</b>
クラスターホスト	≥ 3	クラスター内のホスト数	該当なし

以上の要件をまとめると以下のようになります。

- ユーザーにはいくつかのクラスターノードが必要である。
- クラスターノードには多くの利用可能なメモリーが必要である。
- ユーザーは、ローカルまたはクラウドプロバイダーに数 GiB の利用可能なストレージを持っている必要がある。
- PV サイズはテンプレートパラメーターの値を上書きして変更できる。

## 4.3. ロール変数の設定

### 4.3.1. 概要

以下のセクションでは、[Ansible インベントリーファイル](#) で使用されるロール変数について説明します。ロール変数は、[インストーラーを実行](#) する際に Red Hat CloudForms インストールの動作を制御するために使用されます。

### 4.3.2. 一般的な変数

変数	必須	デフォルト	説明
<code>openshift_management_inst</code> <code>all_management</code>	いいえ	<code>false</code>	ブール値。アプリケーションをインストールするには <code>true</code> に設定します。
<code>openshift_management_app_template</code>	はい	<code>cfme-template</code>	インストールする Red Hat CloudForms のデプロイメントのバリエーションです。コンテナ化されたデータベースの <code>cfme-template</code> を設定するか、または外部データベースの <code>cfme-template-ext-db</code> を設定します。
<code>openshift_management_project</code>	いいえ	<code>openshift-management</code>	Red Hat CloudForms インストールの namespace (プロジェクト)。
<code>openshift_management_project_description</code>	いいえ	<code>CloudForms Management Engine</code>	namespace (プロジェクト) の説明。
<code>openshift_management_username</code>	いいえ	<code>admin</code>	デフォルトの管理ユーザー名。この値を変更してもユーザー名は変更されません。名前をすでに変更しており、統合スクリプト ( <a href="#">コンテナプロバイダーを追加する</a> ためのスクリプトなど) を実行している場合にのみこの値を変更してください。

変数	必須	デフォルト	説明
<code>openshift_management_password</code>	いいえ	<code>smartvm</code>	デフォルトの管理パスワード。この値を変更してもパスワードは変更されません。このパスワードをすでに変更しており、統合スクリプト ( <a href="#">コンテナプロバイダーを追加する</a> ためのスクリプトなど) を実行している場合にのみこの値を変更してください。

### 4.3.3. テンプレートパラメーターのカスタマイズ

`openshift_management_template_parameters` Ansible ロール変数は、アプリケーションまたは PV テンプレートで上書きする必要のあるテンプレートを指定するために使用します。

たとえば、PostgreSQL Pod のメモリー要件を減らしたい場合には、以下を設定します。

```
openshift_management_template_parameters={'POSTGRES_MEM_REQ': '1Gi'}
```

Red Hat CloudForms テンプレートが処理される際に、**1Gi** が `POSTGRES_MEM_REQ` テンプレートパラメーターの値に使用されます。

すべてのテンプレートパラメーターが (コンテナ化されたデータベースと外部データベースの) 両方のテンプレートバリエーションにある訳ではありません。たとえば、Pod 化されたデータベースのテンプレートには `POSTGRES_MEM_REQ` パラメーターがありますが、このパラメーターは外部のデータベーステンプレートにはありません。そこには Pod を必要とするデータベースが存在せず、この情報は必要とされないためです。

したがって、テンプレートパラメーターを上書きする場合には十分注意する必要があります。テンプレートで定義されていないパラメーターを追加するとエラーの原因になります。**管理アプリケーションの作成を確認する** タスクの実行時にエラーを受信した場合は、インストーラーを再度実行する前に [アンインストールのスクリプト](#) を実行してください。

### 4.3.4. データベース変数

#### 4.3.4.1. コンテナ化された (Pod 化された) データベース

`cfme-template.yaml` ファイルにある `POSTGRES_*` または `DATABASE_*` テンプレートパラメーターは、インベントリーファイルの `openshift_management_template_parameters` ハッシュを使用してカスタマイズすることができます。

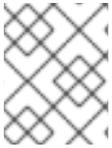
#### 4.3.4.2. 外部データベース

`cfme-template-ext-db.yaml` ファイルにある `POSTGRES_*` または `DATABASE_*` テンプレートパラメーターは、インベントリーファイルの `openshift_management_template_parameters` ハッシュを使用してカスタマイズすることができます。

外部 PostgreSQL データベースは、ユーザーにデータベース接続パラメーターを指定するように要求します。必要な接続キーはインベントリーの `openshift_management_template_parameters` パラメーターに設定する必要があります。必要なキー以下の通りです。

- `DATABASE_USER`

- DATABASE\_PASSWORD
- DATABASE\_IP
- DATABASE\_PORT (ほとんどの PostgreSQL サーバーはポート **5432** で実行されます)
- DATABASE\_NAME



#### 注記

外部データベースで PostgreSQL 9.5 が実行されていることを確認します。実行されていないと、CloudForms アプリケーションを正常にデプロイできない場合があります。

インベントリに、以下のような行が含まれているはずですが、

```
[OSEv3:vars]
```

```
openshift_management_app_template=cfme-template-ext-db 1
openshift_management_template_parameters={'DATABASE_USER': 'root',
'DATABASE_PASSWORD': 'mypassword', 'DATABASE_IP': '10.10.10.10', 'DATABASE_PORT':
'5432', 'DATABASE_NAME': 'cfme'}
```

- 1** `openshift_management_app_template` パラメーターを `cfme-template-ext-db` に設定します。

#### 4.3.5. ストレージクラス変数

変数	必須	デフォルト	説明
<code>openshift_management_storage_class</code>	いいえ	<b>nfs</b>	使用されるストレージのタイプ。オプションには <b>nfs</b> 、 <b>nfs_external</b> 、 <b>p_reconfigured</b> 、 <b>cloudprovider</b> があります。
<code>openshift_management_storage_nfs_external_hostname</code>	いいえ	<b>false</b>	NetApp アプライアンスなどの外部 NFS サーバーを使用している場合、ここにホスト名を設定する必要があります。外部 NFS を使用していない場合は値を <b>false</b> のままにしておきます。さらに外部 NFS では、ユーザーがアプリケーション PV とオプションでデータベース PV をサポートする NFS エクスポートを作成する必要があります。

変数	必須	デフォルト	説明
<code>openshift_management_storage_nfs_base_dir</code>	いいえ	<code>/exports/</code>	外部 NFS を使用している場合、ベースパスをこのエクスポートの位置に設定できます。ローカルの NFS の場合、ローカルの NFS エクスポートに使用されているデフォルトのパスを変更する必要がある場合にも、この値を変更します。
<code>openshift_management_storage_nfs_local_hostname</code>	いいえ	<code>false</code>	<b>[nfs]</b> グループがインベントリーにない場合や、ローカルの NFS ホストをクラスターに手動で定義する必要がある場合は、このパラメーターを必要な NFS サーバーのホスト名に設定します。サーバーは OpenShift Container Platform クラスターの一部である必要があります。

#### 4.3.5.1. NFS (デフォルト)

NFS ストレージクラスは、概念実証およびテストデプロイメントに最も適しています。このクラスは、デプロイメント用のデフォルトのステージクラスにもなります。これを選択した場合、追加の設定は不要です。

このストレージクラスは、必要な PV をサポートするように NFS をクラスターホスト (デフォルトではインベントリーファイルの最初のマスター) に設定します。アプリケーションは PV を必要とし、データベース (外部でホストされている可能性がある) は 2 番目の PV が必要となる場合があります。PV サイズの最小要件は、Red Hat CloudForms アプリケーションは 5GiB、PostgreSQL データベースは 15GiB です (NFS 専用で使用する場合は、ボリュームまたはパーティション上の使用可能な最小領域は 20GiB です)。

カスタマイズは、以下のロール変数を使って行われます。

- `openshift_management_storage_nfs_base_dir`
- `openshift_management_storage_nfs_local_hostname`

#### 4.3.5.2. NFS (外部)

外部 NFS は、必要な PV にエクスポートを提供するために事前設定された NFS サーバーを使用します。外部 NFS の場合、ユーザーは `cfme-app` とオプションで `cfme-db` (コンテナ化されたデータベース用) のエクスポートが必要です。

設定は、以下のロール変数を使って提供されます。

- `openshift_management_storage_nfs_external_hostname`

- `openshift_management_storage_nfs_base_dir`

`openshift_management_storage_nfs_external_hostname` パラメーターは、外部 NFS サーバーのホスト名または IP に設定する必要があります。

`/exports` がエクスポートの親ディレクトリーではない場合、ベースディレクトリーを `openshift_management_storage_nfs_base_dir` パラメーターで設定する必要があります。

たとえば、サーバーエクスポートが `/exports/hosted/prod/cfme-app` の場合は、`openshift_management_storage_nfs_base_dir=/exports/hosted/prod` を設定する必要があります。

#### 4.3.5.3. クラウドプロバイダー

ストレージクラスに OpenShift Container Platform のクラウドプロバイダー統合を使用している場合、Red Hat CloudForms は必要な PV をサポートするためにこのクラウドプロバイダーを使用することができます。これを正常に実行するには、`openshift_cloudprovider_kind` 変数 (AWS または GCE 用) と、ユーザーが選択したクラウドプロバイダーに固有のすべての関連パラメーターを設定する必要があります。

アプリケーションがこのストレージクラスを使って作成されている場合、必要な PV は、設定済みのクラウドプロバイダーのストレージ統合を使って自動的にプロビジョニングされます。

このストレージクラスの動作を設定するために使用される追加の変数はありません。

#### 4.3.5.4. 事前設定 (詳細)

`preconfigured` (事前設定されている) ストレージクラスの場合、ユーザーは各自の操作を正確に把握しており、すべてのストレージ要件が事前に配慮されていることを意味します。つまり、一般的には正しいサイズの PV を作成されています。インストーラーは、ストレージ設定を変更する必要がありません。

このストレージクラスの動作を設定するために使用される追加の変数はありません。

## 4.4. インストーラーの実行

### 4.4.1. OpenShift Container Platform のインストール時またはインストール後の Red Hat CloudForms のデプロイ

Red Hat CloudForms のデプロイを、OpenShift Container Platform の初回インストール時か、またはクラスターのプロビジョニング後のいずれかに実行することを選択できます。

1. `openshift_management_install_management` がインベントリーファイルの `[OSEv3:vars]` セクションの下で `true` に設定されていることを確認します。

```
[OSEv3:vars]
openshift_management_install_management=true
```

2. インベントリーファイルにある Red Hat CloudForms のその他のロール変数を、[ロール変数の設定](#) で説明されているように設定します。これを支援するリソースは、[インベントリーファイルの例](#) を参照してください。
3. OpenShift Container Platform がすでにプロビジョニングされているかどうかに応じて、実行する Playbook を選択します。

- a. Red Hat CloudForms を、OpenShift Container Platform クラスターのインストールと同時にインストールする必要がある場合には、[インストール Playbook の実行](#) で説明されているように標準の `config.yml` Playbook を呼び出して、OpenShift Container Platform クラスターと Red Hat CloudForms のインストールを開始します。
- b. Red Hat CloudForms を、すでにプロビジョニングされている OpenShift Container Platform クラスターにインストールする必要がある場合には、Playbook ディレクトリーに切り替えてから Red Hat CloudForms Playbook を直接呼び出してインストールを開始します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v [-i /path/to/inventory] \
  playbooks/openshift-management/config.yml
```

#### 4.4.2. インベントリーファイルの例

このセクションでは、インベントリーファイルのスニペットの例について説明し、使い始めに役立つ OpenShift Container Platform での Red Hat CloudForms の各種の設定について説明します。



##### 注記

変数の詳しい説明については、[ロール変数の設定](#) を参照してください。

##### 4.4.2.1. すべてのデフォルト

以下は、すべてのデフォルト値と選択肢を使用した最も単純な例です。これで、Red Hat CloudForms のインストールが完全にコンテナ化 (Pod 化) されます。すべてのアプリケーションコンポーネントと PostgreSQL データベースが OpenShift Container Platform に Pod として作成されます。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
```

##### 4.4.2.2. 外部 NFS ストレージ

以下は、前述の例と同様に (ただしローカルの NFS サービスをクラスターで使用する代わりに)、既存の外部 NFS サーバー (ストレージアプライアンスなど) を使用しています。新しいパラメーターが 2 つある点に注意してください。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
openshift_management_storage_class=nfs_external ①
openshift_management_storage_nfs_external_hostname=nfs.example.com ②
```

① `nfs_external` に設定します。

② NFS サーバーのホスト名に設定します。

外部 NFS ホストが `/exports/hosted/prod` などの異なる親ディレクトリーの下でエクスポートディレクトリーをホストしている場合は、以下の変数を追加します。

```
openshift_management_storage_nfs_base_dir=/exports/hosted/prod
```

#### 4.4.2.3. PV サイズの上書き

以下の例では、永続ボリューム (PV) のサイズを上書きします。PV サイズは **openshift\_management\_template\_parameters** で設定する必要があります。これにより、アプリケーションとデータベースは相互に干渉しあうことなく作成済みの PV で要求を実行できます。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
openshift_management_template_parameters={'APPLICATION_VOLUME_CAPACITY': '10Gi',
'DATABASE_VOLUME_CAPACITY': '25Gi'}
```

#### 4.4.2.4. メモリー要件の上書き

テストまたは概念実証のインストールでは、アプリケーションとデータベースのメモリー要件を設定している容量内に収めるようにする必要があります。メモリーの上限を下げると、パフォーマンスが低下するか、またはアプリケーションの初期化に完全に失敗したりする可能性があることに注意してください。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template
openshift_management_template_parameters={'APPLICATION_MEM_REQ': '3000Mi',
'POSTGRESQL_MEM_REQ': '1Gi', 'ANSIBLE_MEM_REQ': '512Mi'}
```

この例では、インストーラーに対し、パラメーター **APPLICATION\_MEM\_REQ** を **3000Mi** に、**POSTGRESQL\_MEM\_REQ** を **1Gi** に、さらに **ANSIBLE\_MEM\_REQ** を **512Mi** に設定してアプリケーションテンプレートを処理するように指示します。

これらのパラメーターは、前述の例 [PV サイズの上書き](#) に示されているパラメーターと組み合わせることができます。

#### 4.4.2.5. 外部 PostgreSQL データベース

外部データベースを使用するには、**openshift\_management\_app\_template** パラメーターの値を **cfme-template-ext-db** に変更する必要があります。

さらに、データベース接続情報を **openshift\_management\_template\_parameters** 変数を使って入力する必要があります。詳細は、[ロール変数の設定](#) を参照してください。

```
[OSEv3:vars]
openshift_management_app_template=cfme-template-ext-db
openshift_management_template_parameters={'DATABASE_USER': 'root',
'DATABASE_PASSWORD': 'mypassword', 'DATABASE_IP': '10.10.10.10', 'DATABASE_PORT':
'5432', 'DATABASE_NAME': 'cfme'}
```



#### 重要

PostgreSQL 9.5 が実行中であることを確認してください。実行されていないと、アプリケーションを正常にデプロイできない場合があります。

### 4.5. コンテナプロバイダー統合の有効化

#### 4.5.1. 単一コンテナプロバイダーの追加

[インストーラーの実行](#) で説明されているように Red Hat CloudForms を OpenShift Container Platform にデプロイした後に、コンテナプロバイダーの統合を有効にする方法は2つあります。OpenShift Container Platform をコンテナプロバイダーとして手動で追加するか、このロールに含まれる Playbook を試してみてください。

#### 4.5.1.1. 手動の追加

OpenShift Container Platform クラスターをコンテナプロバイダーとして手動で追加する手順については、以下の Red Hat CloudForms ドキュメントを参照してください。

- [Integration with OpenShift Container Platform](#)

#### 4.5.1.2. 自動の追加

コンテナプロバイダーの自動的な統合は、このロールに含まれている Playbook を使用して実行できます。

この Playbook は以下を実行します。

1. 必要な認証シークレットを収集します。
2. Red Hat CloudForms アプリケーションとクラスター API への公開ルートを検索します。
3. REST の呼び出しを実行し、OpenShift Container Platform クラスターをコンテナプロバイダーとして追加します。

Playbook ディレクトリーに切り替え、コンテナプロバイダーの Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v [-i /path/to/inventory] \
  openshift-management/add_container_provider.yml
```

### 4.5.2. 複数のコンテナプロバイダー

このロールには、最新の OpenShift Container Platform クラスターを Red Hat CloudForms デプロイメントに統合するための Playbook に加え、複数のコンテナプラットフォームを任意の Red Hat CloudForms サーバーにコンテナプロバイダーとして追加することを可能にするスクリプトが含まれています。コンテナプラットフォームは、OpenShift Container Platform または OpenShift Origin です。

複数のプロバイダースクリプトを使用する場合、Playbook の実行時に **EXTRA\_VARS** パラメーターを CLI に手動で設定することが必要になります。

#### 4.5.2.1. スクリプトの作成

複数のプロバイダースクリプトを作成するには、以下の手動の設定を実行します。

1. `/usr/share/ansible/openshift-ansible/roles/openshift_management/files/examples/container_providers.yml` のサンプルを、`/tmp/cp.yml` など別の場所にコピーします。このファイルは後で変更します。
2. Red Hat CloudForms の名前またはパスワードを変更している場合は、コピーした `container_providers.yml` ファイルの `management_server` キーにある `hostname`、`user` および `password` の各パラメーターを更新します。

3. コンテナプロバイダーとして追加する必要がある各 Container Platform クラスターについて、**container\_providers** キーの下にエントリーを入力します。

a. 以下のパラメーターを設定する必要があります。

- **auth\_key** - **cluster-admin** 権限を持つサービスアカウントのトークンです。
- **hostname** - クラスター API をポイントしているホスト名です。各コンテナプロバイダーは固有のホスト名を持っている必要があります。
- **name** - Red Hat CloudForms サーバーのコンテナプロバイダーの概要ページに表示されるクラスター名です。この名前は固有でなければなりません。

## ヒント

**auth\_key** ベアラートークンをクラスターから取得するには、以下を実行します。

```
$ oc serviceaccounts get-token -n management-infra management-admin
```

b. 以下のパラメーターは任意で設定することができます。

- **port** - Container Platform クラスターが API を **8443** 以外のポートで実行している場合は、このキーを更新します。
- **endpoint** - SSL 検証を有効にする (**verify\_ssl**) か、または検証設定を **ssl-with-validation** に変更することができます。現時点では、カスタムの信頼される CA 証明書のサポートは利用できません。

### 4.5.2.1.1. 例

例として以下のシナリオを見てみましょう。

- **container\_providers.yml** ファイルを **/tmp/cp.yml** にコピーしている。
- 2つの OpenShift Container Platform クラスターを追加する必要がある。
- Red Hat CloudForms サーバーが **mgmt.example.com** で実行されている。

このシナリオでは、**/tmp/cp.yml** を以下のようにカスタマイズします。

```
container_providers:
- connection_configurations:
  - authentication: {auth_key: "<token>", authtype: bearer, type: AuthToken} ❶
    endpoint: {role: default, security_protocol: ssl-without-validation, verify_ssl: 0}
    hostname: "<provider_hostname1>"
    name: <display_name1>
    port: 8443
    type: "ManagementIQ::Providers::Openshift::ContainerManager"
- connection_configurations:
  - authentication: {auth_key: "<token>", authtype: bearer, type: AuthToken} ❷
    endpoint: {role: default, security_protocol: ssl-without-validation, verify_ssl: 0}
    hostname: "<provider_hostname2>"
    name: <display_name2>
    port: 8443
    type: "ManagementIQ::Providers::Openshift::ContainerManager"
```

```
management_server:
  hostname: "<hostname>"
  user: <user_name>
  password: <password>
```

**1 2** **<token>** をこのクラスターの管理トークンに置き換えます。

#### 4.5.2.2. Playbook の実行

複数プロバイダー統合スクリプトを実行するには、コンテナプロバイダーの設定ファイルへのパスを **EXTRA\_VARS** パラメーターとして **ansible-playbook** コマンドに指定する必要があります。 **container\_providers\_config** を設定ファイルパスに設定するには、 **-e** (または **--extra-vars**) パラメーターを使用します。 Playbook ディレクトリーに切り替えて Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v [-i /path/to/inventory] \
  -e container_providers_config=/tmp/cp.yml \
  playbooks/openshift-management/add_many_container_providers.yml
```

Playbook が完成すると、Red Hat CloudForms サービスに 2 つの新しいコンテナプロバイダーが見つかるはずです。 **コンピューター → コンテナ → プロバイダー** の順にページを進んで概要を確認してください。

#### 4.5.3. プロバイダーの更新

単一または複数のコンテナプロバイダーを追加したら、新規プロバイダーを Red Hat CloudForms で更新し、コンテナプロバイダーと管理されているコンテナに関する最新データをすべて取得する必要があります。そのためには、Red Hat CloudForms Web コンソールの各プロバイダーに進み、それぞれについて更新ボタンをクリックします。

手順については、以下の Red Hat CloudForms ドキュメントを参照してください。

- [Managing Providers](#)

## 4.6. RED HAT CLOUDFORMS のアンインストール

### 4.6.1. アンインストール Playbook の実行

デプロイした Red Hat CloudForms インストールを OpenShift Container Platform からアンインストールして削除するには、Playbook ディレクトリーに切り替え、 **uninstall.yml** Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v [-i /path/to/inventory] \
  playbooks/openshift-management/uninstall.yml
```



#### 重要

NFS エクスポートの定義と NFS エクスポートに格納されているデータは自動的に削除されません。新規デプロイメントの初期化を試行する前に、古いアプリケーションまたはデータベースデプロイメントのデータを手動で消去することが推奨されます。

## 4.6.2. トラブルシューティング

古い PostgreSQL データの消去に失敗すると連鎖的なエラーが発生し、**postgresql** Pod が **crashloopbackoff** の状態になる可能性があります。この状態になると **cfme** Pod の起動が阻止されます。**crashloopbackoff** は、以前のデプロイ時に作成されたデータベース NFS エクスポートの不正なファイル権限に起因します。

次に進むには、PostgreSQL エクスポートからすべてのデータを削除し、Pod (デプロイヤー Pod ではない) を削除します。以下の Pod がある場合の例を示します。

```
$ oc get pods
NAME                READY   STATUS    RESTARTS   AGE
httpd-1-cx7fk       1/1    Running   1          21h
cfme-0              0/1    Running   1          21h
memcached-1-vkc7p  1/1    Running   1          21h
postgresql-1-deploy 1/1    Running   1          21h
postgresql-1-6w2t4  0/1    CrashLoopBackOff 1          21h
```

この場合、以下を実行します。

1. データベース NFS エクスポートのデータを消去します。
2. 以下を実行します。

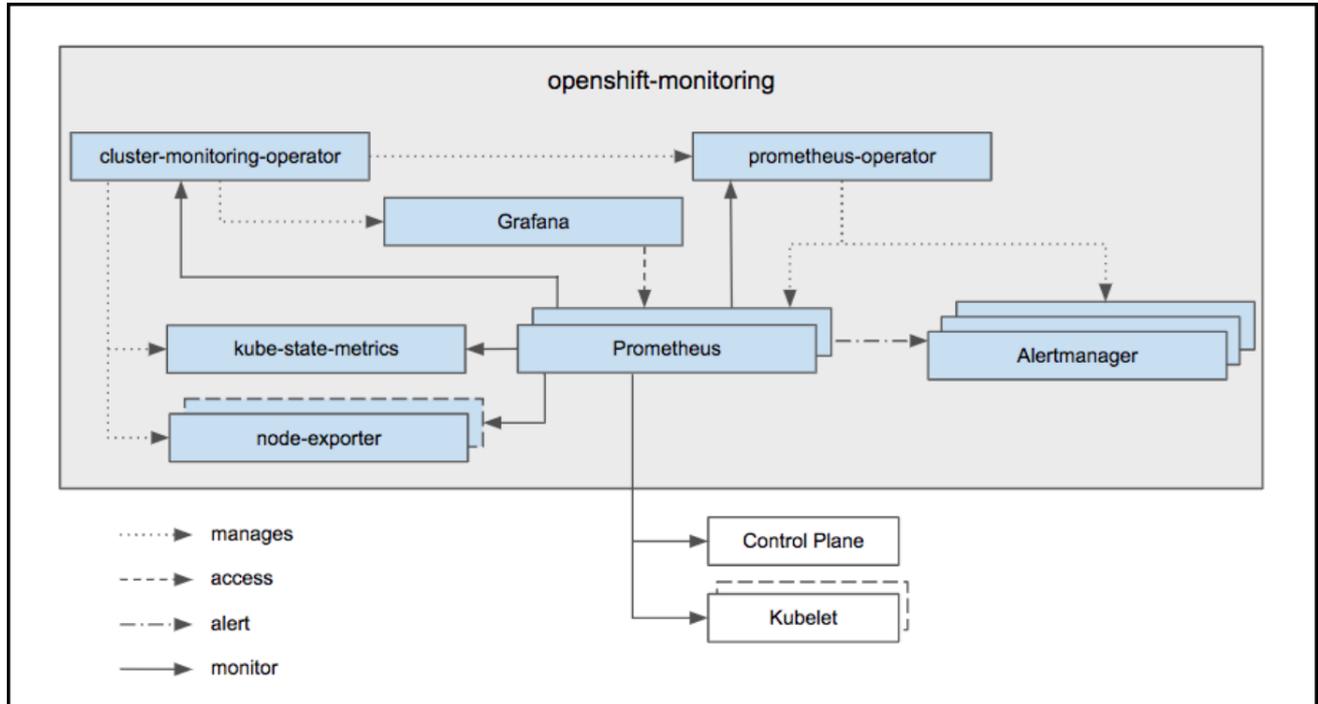
```
$ oc delete postgresql-1-6w2t4
```

PostgreSQL デプロイヤー Pod は、削除した Pod を置き換えるために新規の **postgresql** Pod の拡張を試みます。**postgresql** Pod が実行された後に、**cfme** Pod は阻止する操作を停止し、アプリケーションの初期化を開始します。

## 第5章 PROMETHEUS クラスターモニタリング

### 5.1. 概要

OpenShift Container Platform には、[Prometheus](#) オープンソースプロジェクトおよびその幅広いエコシステムをベースとする事前に設定された、自己更新型のモニタリングスタックが同梱されます。これは、クラスターコンポーネントのモニタリング機能を提供し、一連のアラートと共に提供されるためにクラスター管理者に対して問題の発生について即時に通知し、一連の [Grafana](#) ダッシュボードを提供します。



上図でハイライトされているように、OpenShift Container Platform Cluster Monitoring Operator (CMO) はモニタリングスタックの中心に位置します。これは、デプロイされたモニタリングコンポーネントおよびリソースを監視し、それらが最新の状態にあることを確認します。

Prometheus Operator (PO) は、Prometheus および Alertmanager インスタンスを作成し、設定し、管理します。また、Kubernetes ラベルのクエリーに基づいてモニタリングターゲットの設定を自動生成します。

Prometheus および Alertmanager に加えて、OpenShift Container Platform Monitoring には [node-exporter](#) および [kube-state-metrics](#) も含まれます。Node-exporter は、ノードについてのメトリクスを収集するために各ノードにデプロイされるエージェントです。kube-state-metrics exporter エージェントは、Kubernetes オブジェクトを Prometheus で消費できるメトリクスに変換します。

クラスターモニタリングの一部としてモニターされるターゲットには、以下が含まれます。

- Prometheus 自体
- Prometheus-Operator
- cluster-monitoring-operator
- Alertmanager クラスターインスタンス
- Kubernetes apiserver

- kubelet (コンテナメトリクス用に kubelet で組み込まれる cAdvisor)
- kube-controllers
- kube-state-metrics
- node-exporter
- etcd (etcd モニタリングが有効にされる場合)

これらのコンポーネントはすべて自動的に更新されます。

OpenShift Container Platform Cluster Monitoring Operator についての詳細は、[Cluster Monitoring Operator](#) GitHub プロジェクトを参照してください。



### 注記

互換性が保証された更新を提供できるようにするため、OpenShift Container Platform Monitoring スタックの設定は明示的に利用可能なオプションのみに制限されます。

## 5.2. OPENSIFT CONTAINER PLATFORM クラスターモニタリングの設定

OpenShift Container Platform の Ansible `openshift_cluster_monitoring_operator` ロールは、[インベントリーファイル](#) の変数を使用して Cluster Monitoring Operator を設定し、デプロイします。

表5.1 Ansible 変数

変数	説明
<code>openshift_cluster_monitoring_operator_install</code>	<code>true</code> の場合に Cluster Monitoring Operator をデプロイします。そうでない場合はデプロイ解除します。この変数は、デフォルトで <code>true</code> に設定されます。
<code>openshift_cluster_monitoring_operator_prometheus_storage_capacity</code>	各 Prometheus インスタンスの Persistent Volume Claim (永続ボリューム要求、PVC) のサイズです。変数は、 <code>openshift_cluster_monitoring_operator_prometheus_storage_enabled</code> が <code>true</code> に設定される場合にのみ適用されます。デフォルトは <b>50Gi</b> に設定されます。
<code>openshift_cluster_monitoring_operator_alertmanager_storage_capacity</code>	各 Alertmanager インスタンスの Persistent Volume Claim (永続ボリューム要求、PVC) のサイズです。変数は、 <code>openshift_cluster_monitoring_operator_alertmanager_storage_enabled</code> が <code>true</code> に設定されている場合にのみ適用されます。デフォルトは <b>2Gi</b> に設定されます。

変数	説明
<code>openshift_cluster_monitoring_operator_node_selector</code>	必要な既存の <a href="#">ノードセレクター</a> を設定して、Pod が特定のラベルを持つノードに配置されるようにします。デフォルトは <code>node-role.kubernetes.io/infra=true</code> になります。
<code>openshift_cluster_monitoring_operator_alertmanager_config</code>	Alertmanager を設定します。
<code>openshift_cluster_monitoring_operator_prometheus_storage_enabled</code>	Prometheus の時系列データの永続ストレージを有効にします。この変数は、デフォルトで <code>false</code> に設定されます。
<code>openshift_cluster_monitoring_operator_alertmanager_storage_enabled</code>	Alertmanager の通知および非通知 (silences) の永続ストレージを有効にします。この変数は、デフォルトで <code>false</code> に設定されます。
<code>openshift_cluster_monitoring_operator_prometheus_storage_class_name</code>	<code>openshift_cluster_monitoring_operator_prometheus_storage_enabled</code> オプションを有効にしている場合、特定の StorageClass を設定して、Pod がその <code>storageclass</code> で <code>PVC</code> を使用するよう設定されていることを確認します。デフォルトは <code>none</code> に設定され、これによりデフォルトのストレージクラス名が適用されます。
<code>openshift_cluster_monitoring_operator_alertmanager_storage_class_name</code>	<code>openshift_cluster_monitoring_operator_alertmanager_storage_enabled</code> オプションを有効にしている場合、特定の StorageClass を設定し、Pod がその <code>storageclass</code> で <code>PVC</code> を使用するよう設定されていることを確認します。デフォルトは <code>none</code> に設定され、これによりデフォルトのストレージクラス名が適用されます。

### 5.2.1. モニターリングの前提条件

モニタリングスタックには、追加のリソース要件があります。詳細は、[コンピューティングリソースについての推奨事項](#) を参照してください。

### 5.2.2. モニターリングスタックのインストール

モニタリングスタックは、デフォルトで OpenShift Container Platform と共にインストールされます。これがインストールされないようにすることもできます。これを実行するには、Ansible インベントリファイルでこの変数を `false` に設定します。

#### `openshift_cluster_monitoring_operator_install`

以下でこれを実行できます。

```
$ ansible-playbook [-i </path/to/inventory>] <OPENSHIFT_ANSIBLE_DIR>/playbooks/openshift-
monitoring/config.yml \
  -e openshift_cluster_monitoring_operator_install=False
```

Ansible ディレクトリーの共通パスは `/usr/share/ansible/openshift-ansible/` です。この場合、設定ファイルへのパスは `/usr/share/ansible/openshift-ansible/playbooks/openshift-monitoring/config.yml` になります。

### 5.2.3. 永続ストレージ

クラスターモニタリングを永続ストレージと共に実行すると、メトリクスは [永続ボリューム](#) に保存され、再起動または再作成される Pod を維持できます。これは、メトリクスデータまたはアラートデータをデータ損失から保護する必要がある場合に適しています。実稼働環境では、[ブロックストレージテクノロジー](#) を使用して永続ストレージを設定することを強く推奨します。

#### 5.2.3.1. 永続ストレージの有効化

デフォルトで、永続ストレージは Prometheus の時系列データおよび Alertmanager の通知および非通知 (silences) の両方に対して無効にされます。クラスターを、それらのいずれかまたは両方を永続的に保存するように設定することができます。

- Prometheus 時系列データの永続ストレージを有効にするには、Ansible インベントリーファイルでこの変数を **true** に設定します。  
**openshift\_cluster\_monitoring\_operator\_prometheus\_storage\_enabled**
- Alertmanager 通知および非通知 (silences) の永続ストレージを有効にするには、Ansible インベントリーファイルでこの変数を **true** に設定します。  
**openshift\_cluster\_monitoring\_operator\_alertmanager\_storage\_enabled**

#### 5.2.3.2. 必要なストレージサイズの判別

必要な永続ストレージは Pod 数によって異なります。ディスクが一杯にならないように十分なストレージを確保することは管理者の責任です。永続ストレージのシステム要件については、[Capacity Planning for Cluster Monitoring Operator](#) を参照してください。

#### 5.2.3.3. 永続ストレージサイズの設定

Prometheus および Alertmanager の Persistent Volume Claim (永続ボリューム要求、PVC) のサイズを指定するには、以下の Ansible 変数を変更します。

- **openshift\_cluster\_monitoring\_operator\_prometheus\_storage\_capacity** (デフォルト: 50Gi)
- **openshift\_cluster\_monitoring\_operator\_alertmanager\_storage\_capacity** (デフォルト: 2Gi)

上記の変数のそれぞれは、対応する **storage\_enabled** 変数が **true** に設定されている場合にのみ適用されます。

#### 5.2.3.4. 十分な永続ボリュームの割り当て

動的にプロビジョニングされるストレージを使用しない限り、PVC で要求される永続ボリューム (PV) が利用できる状態にあることを確認する必要があります。各レプリカに1つの PV が必要です。Prometheus には2つのレプリカがあり、Alertmanager には3つのレプリカがあるため、合計で5つの PV が必要になります。

#### 5.2.3.5. 動的にプロビジョニングされたストレージの有効化

静的にプロビジョニングされたストレージの代わりに、動的にプロビジョニングされたストレージを使用できます。詳細は、[Dynamic Volume Provisioning](#) を参照してください。

Prometheus および Alertmanager の動的ストレージを有効にするには、Ansible インベントリファイルで以下のパラメーターを **true** に設定します。

- **openshift\_cluster\_monitoring\_operator\_prometheus\_storage\_enabled** (Default: false)
- **openshift\_cluster\_monitoring\_operator\_alertmanager\_storage\_enabled** (Default: false)

動的ストレージを有効にした後に、Ansible インベントリファイルの以下のパラメーターで、各コンポーネントの Persistent Volume Claim (永続ボリューム要求、PVC) に **storageclass** を設定することもできます。

- **openshift\_cluster\_monitoring\_operator\_prometheus\_storage\_class\_name** (デフォルト: "")
- **openshift\_cluster\_monitoring\_operator\_alertmanager\_storage\_class\_name** (デフォルト: "")

上記の変数のそれぞれは、対応する **storage\_enabled** 変数が **true** に設定されている場合にのみ適用されます。

#### 5.2.4. サポートされる設定

OpenShift Container Platform モニターリングの設定は、本書で説明されているオプションを使用して行う方法がサポートされている方法です。このような明示的な設定オプション以外に、追加の設定をスタックに挿入できます。ただし、設定のパラダイムが Prometheus リリース間で変更される可能性があるため、これはサポートされていません。このような変更には、設定のすべての可能性が制御されている場合のみ適切に対応できます。

明示的にサポート対象外とされているケースには、以下が含まれます。

- 追加の **ServiceMonitor** オブジェクトを **openshift-monitoring** namespace に作成する。これにより、クラスターをモニターリングする Prometheus インスタンスの収集ターゲットが拡張されます。これは、認識不可能な競合および負荷の差異を生じさせるため、Prometheus のセットアップは不安定になる可能性があります。
- 追加の **ConfigMap** オブジェクトを作成する。これにより、クラスターをモニターリングする Prometheus インスタンスに追加のアラートおよび記録ルールが組み込まれます。Prometheus 2.0 は新規のルールファイル構文と共に提供されるため、この動作は互換性を失わせる可能性のある動作を引き起すものとして知られている点に注意してください。

### 5.3. ALERTMANAGER の設定

Alertmanager は受信アラートを管理し、これには、非通知 (silencing)、抑制 (inhibition)、集計 (aggregation)、およびメール、PagerDuty、および HipChat などの方法での通知の送信が含まれます。

OpenShift Container Platform Monitoring Alertmanager クラスターのデフォルト設定:

```
global:
  resolve_timeout: 5m
route:
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: default
routes:
- match:
  alertname: DeadMansSwitch
```

```

repeat_interval: 5m
receiver: deadmansswitch
receivers:
- name: default
- name: deadmansswitch

```

この設定は、**openshift\_cluster\_monitoring\_operator** ロールで Ansible 変数 **openshift\_cluster\_monitoring\_operator\_alertmanager\_config** を使用して上書きできます。

以下の例では、[PagerDuty](#) で通知を設定しています。**service\_key** を取得する方法については、[PagerDuty ドキュメントの Alertmanager](#) についての記述を参照してください。

```

openshift_cluster_monitoring_operator_alertmanager_config: |+
global:
  resolve_timeout: 5m
route:
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: default
routes:
- match:
  alertname: DeadMansSwitch
  repeat_interval: 5m
  receiver: deadmansswitch
- match:
  service: example-app
  routes:
- match:
  severity: critical
  receiver: team-frontend-page
receivers:
- name: default
- name: deadmansswitch
- name: team-frontend-page
pagerduty_configs:
- service_key: "<key>"

```

サブルートは重大度 (severity) が **critical** のアラートに対してのみ一致し、それらを **team-frontend-page** という receiver 経由で送信します。この名前が示すように、critical アラートについては、その送信先を設定する必要があります。各種のアラートレシーバー経由でアラートを設定する方法については、[Alertmanager configuration](#) を参照してください。

### 5.3.1. Dead man's switch

OpenShift Container Platform Monitoring には、モニターする側のインフラストラクチャーの可用性が確保するための **Dead man's switch** という機能が同梱されています。

Dead man's switch は、常にトリガーする単純な Prometheus アラートルールです。Alertmanager は、Dead man's switch の通知を、この機能をサポートする通知プロバイダーに絶えず送信します。また、これは Alertmanager と通知プロバイダー間の通信が機能していることを確認します。

この仕組みは、モニターリングシステム自体が停止した場合にアラートを発行するために PagerDuty によってサポートされています。詳細は、後続の [Dead man's switch PagerDuty](#) を参照してください。

### 5.3.2. アラートのグループ化

アラートが Alertmanager に対して発行される際に、Alertmanager はそれらを論理的にグループ化するように設定される必要があります。

この例では、新規ルートが **frontend** チームのアラートルートを反映するように追加されます。

#### 手順

1. 新規ルートを追加します。複数のルートが元のルートの下に追加される場合がありますが、通常ルートは通知の receiver を定義するために追加されます。以下の例では、Matcher を使用してサービス **example-app** から送られるアラートのみが使用されるようにします。

```
global:
  resolve_timeout: 5m
route:
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: default
  routes:
  - match:
    alertname: DeadMansSwitch
    repeat_interval: 5m
    receiver: deadmansswitch
  - match:
    service: example-app
    routes:
    - match:
      severity: critical
      receiver: team-frontend-page
receivers:
- name: default
- name: deadmansswitch
```

サブルートは重大度 (severity) が **critical** のアラートに対してのみ一致し、それらを **team-frontend-page** という receiver 経由で送信します。この名前が示すように、critical アラートについては、その送信先を設定する必要があります。

### 5.3.3. Dead man's switch PagerDuty

[PagerDuty](#) は、[Dead Man's Snitch](#) という統合によりこの仕組みをサポートしています。単純に **PagerDuty** 設定をデフォルトの **deadmansswitch** receiver に追加します。上記のプロセスを使用してこの設定を追加します。

Dead Man's Snitch を、Dead Man's Switch アラートが 15 分間サイレントな場合に Operator を呼び出すように設定します。デフォルトの Alertmanager 設定では、Dead Man's Switch アラートは 5 分ごとに繰り返されます。Dead Man's Snitch が 15 分後にトリガーされる場合、これは通知が少なくとも 2 回失敗したことを示します。

[configure Dead Man's Snitch を PagerDuty に設定する方法](#) について参照してください。

### 5.3.4. アラートルール

OpenShift Container Platform Cluster Monitoring には、デフォルトで設定される以下のアラートルールが同梱されます。現時点で、カスタムアラートルールを追加することはできません。

一部のアラートルールには同じ名前が付けられています。これは意図的な理由によるものです。これらのルールは、それぞれのしきい値、それぞれの重大度 (severity) またはそれらの両方を使って同じイベントについてのアラートを送ります。抑制ルールを使用すると、高い重大度のアラートが発生する場合に重大度の低いアラートが抑制されます。

アラートルールについての詳細は、[configuration file](#) を参照してください。

アラート	重要度	説明
<b>ClusterMonitoringOperatorErrors</b>	<b>critical</b>	Cluster Monitoring Operator で X% エラーが発生している。
<b>AlertmanagerDown</b>	<b>critical</b>	Alertmanager が Prometheus のターゲット検出に表示されない。
<b>ClusterMonitoringOperatorDown</b>	<b>critical</b>	ClusterMonitoringOperator が Prometheus のターゲット検出に表示されない。
<b>KubeAPIDown</b>	<b>critical</b>	KubeAPI が Prometheus のターゲット検出に表示されない。
<b>KubeControllerManagerDown</b>	<b>critical</b>	KubeControllerManager が Prometheus のターゲット検出に表示されない。
<b>KubeSchedulerDown</b>	<b>critical</b>	KubeScheduler が Prometheus のターゲット検出に表示されない。
<b>KubeStateMetricsDown</b>	<b>critical</b>	KubeStateMetrics が Prometheus のターゲット検出に表示されない。
<b>KubeletDown</b>	<b>critical</b>	Kubelet が Prometheus のターゲット検出に表示されない。
<b>NodeExporterDown</b>	<b>critical</b>	NodeExporter が Prometheus のターゲット検出に表示されない。
<b>PrometheusDown</b>	<b>critical</b>	Prometheus が Prometheus のターゲット検出に表示されない。
<b>PrometheusOperatorDown</b>	<b>critical</b>	PrometheusOperator が Prometheus のターゲット検出に表示されない。
<b>KubePodCrashLooping</b>	<b>critical</b>	Namespace/Pod (コンテナ) が再起動している (回数 / 秒)。

アラート	重要度	説明
<b>KubePodNotReady</b>	<b>critical</b>	Namespace/Pod の準備ができていない。
<b>KubeDeploymentGeneration Mismatch</b>	<b>critical</b>	デプロイメント Namespace/Deployment 生成の不一致。
<b>KubeDeploymentReplicas Mismatch</b>	<b>critical</b>	デプロイメント Namespace/Deployment レプリカの不一致。
<b>KubeStatefulSetReplicas Mismatch</b>	<b>critical</b>	StatefulSet Namespace/StatefulSet レプリカの不一致。
<b>KubeStatefulSetGeneration Mismatch</b>	<b>critical</b>	StatefulSet Namespace/StatefulSet 生成の不一致。
<b>KubeDaemonSetRolloutStuck</b>	<b>critical</b>	必要な Pod の X% のみがスケジュールされており、daemon set Namespace/DaemonSet に対して準備ができていない。
<b>KubeDaemonSetNotScheduled</b>	<b>warning</b>	daemonset Namespace/DaemonSet の多数の Pod がスケジュールされていない。
<b>KubeDaemonSetMisScheduled</b>	<b>warning</b>	daemonset Namespace/DaemonSet の多数の Pod が実行される場所ではない場所で実行されている。
<b>KubeCronJobRunning</b>	<b>warning</b>	CronJob Namespace/CronJob の完了に 1 時間を超える時間がかかる。
<b>KubeJobCompletion</b>	<b>warning</b>	ジョブ Namespaces/Job の完了に 1 時間を超える時間がかかる。
<b>KubeJobFailed</b>	<b>warning</b>	ジョブ Namespaces/Job を完了できない。
<b>KubeCPUOvercommit</b>	<b>warning</b>	Pod でのオーバーコミットされた CPU リソース要求がノードの失敗を許容できない。

アラート	重要度	説明
<b>KubeMemOvercommit</b>	<b>warning</b>	Pod でのオーバーコミットされたメモリーリソース要求がノードの失敗を許容できない。
<b>KubeCPUOvercommit</b>	<b>warning</b>	Namespace でのオーバーコミットされた CPU リソース要求のクォータ。
<b>KubeMemOvercommit</b>	<b>warning</b>	Namespace でのオーバーコミットされたメモリーリソース要求のクォータ。
<b>alerKubeQuotaExceeded</b>	<b>warning</b>	namespace <b>Namespace</b> での <b>Resource</b> が X% 使用されている。
<b>KubePersistentVolumeUsage Critical</b>	<b>critical</b>	namespace <b>Namespace</b> の <b>PersistentVolumeClaim</b> で要求される永続ボリュームに X% の空きがある。
<b>KubePersistentVolumeFullIn FourDays</b>	<b>critical</b>	直近のサンプリングにより、namespace <b>Namespace</b> の <b>PersistentVolumeClaim</b> で要求される永続ボリュームが 4 日以内に一杯になることが予想される。現時点で X バイトが利用可能。
<b>KubeNodeNotReady</b>	<b>warning</b>	<b>Node</b> が 1 時間を経過しても準備状態にならない。
<b>KubeVersionMismatch</b>	<b>warning</b>	Kubernetes コンポーネントの X 種類のバージョンが実行中である。
<b>KubeClientErrors</b>	<b>warning</b>	Kubernetes API サーバークライアントの ' <b>Job/Instance</b> ' で X% エラーが発生している。
<b>KubeClientErrors</b>	<b>warning</b>	Kubernetes API サーバークライアントの ' <b>Job/Instance</b> ' で毎秒 X エラーが発生している。
<b>KubeletTooManyPods</b>	<b>warning</b>	Kubelet <b>Instance</b> が上限の 110 に近い X Pod を実行している。

アラート	重要度	説明
KubeAPILatencyHigh	warning	API サーバーに Verb Resource について 99 番目のパーセンタイルのレイテンシー X 秒がある。
KubeAPILatencyHigh	critical	API サーバーに Verb Resource について 99 番目のパーセンタイルのレイテンシー X 秒がある。
KubeAPIErrorsHigh	critical	API サーバーで X% の要求についてエラーが生じている。
KubeAPIErrorsHigh	warning	API サーバーで X% の要求についてエラーが生じている。
KubeClientCertificateExpiration	warning	Kubernetes API 証明書の有効期限が 7 日以内に切れる。
KubeClientCertificateExpiration	critical	Kubernetes API 証明書の有効期限が 1 日以内に切れる。
AlertmanagerConfigInconsistent	critical	要約: 設定の同期が取れていない。説明: Alertmanager クラスター <b>Service</b> のインスタンスの設定の同期が取れていない。
AlertmanagerFailedReload	warning	要約: Alertmanager の設定のリロードが失敗。説明: Alertmanager の設定のリロードが Namespace/Pod に対して失敗する。
TargetDown	warning	要約: ターゲットがダウンしている。説明: X% の Job ターゲットがダウンしている。
DeadMansSwitch	none	要約: DeadMansSwitch のアラート。説明: アラートパイプライン全体が機能することを確認するための DeadMansSwitch。
NodeDiskRunningFull	warning	node-exporter Namespace/Pod のデバイス Device が 24 時間以内に一杯の状態で行われる。
NodeDiskRunningFull	critical	node-exporter Namespace/Pod のデバイス Device が 2 時間以内に一杯の状態で行われる。

アラート	重要度	説明
<b>PrometheusConfigReloadFailed</b>	<b>warning</b>	要約: Prometheus の設定のリロードに失敗。説明: Prometheus の設定が <b>Namespace/Pod</b> に対して失敗した。
<b>PrometheusNotificationQueueRunningFull</b>	<b>warning</b>	要約: Prometheus のアラート通知キューが一杯の状態で行われている。説明: Prometheus のアラート通知キューが <b>Namespace/Pod</b> に対して一杯の状態で行われている。
<b>PrometheusErrorSendingAlerts</b>	<b>warning</b>	要約: Prometheus からのアラートの送信時のエラー。説明: アラートの Prometheus <b>Namespace/Pod</b> から Alertmanager <b>Alertmanager</b> への送信時のエラー。
<b>PrometheusErrorSendingAlerts</b>	<b>critical</b>	要約: Prometheus からのアラートの送信時のエラー。説明: アラートの Prometheus <b>Namespace/Pod</b> から Alertmanager <b>Alertmanager</b> への送信時のエラー。
<b>PrometheusNotConnectedToAlertmanagers</b>	<b>warning</b>	要約: Prometheus が Alertmanager に接続されていない。説明: Prometheus <b>Namespace/Pod</b> が Alertmanager に接続されていない。
<b>PrometheusTSDBReloadsFailing</b>	<b>warning</b>	要約: Prometheus にディスクからのデータブロックのリロードの問題がある。説明: <b>Instance</b> の <b>Job</b> で、4 時間以内に X のリロードの問題が発生。
<b>PrometheusTSDBCompactionsFailing</b>	<b>warning</b>	要約: Prometheus でサンプルブロックのコンパクト化の問題がある。説明: <b>Instance</b> の <b>Job</b> で、4 時間以内に X のコンパクト化の問題が発生。

アラート	重要度	説明
<b>PrometheusTSDBWALCorruptions</b>	<b>warning</b>	要約: Prometheus ログ先行書き込みが破損している。説明: <b>Instance</b> の <b>Job</b> に破損したログ先行書き込み (WAL) がある。
<b>PrometheusNotIngestingSamples</b>	<b>warning</b>	要約: Prometheus がサンプルを取り入れていない。説明: Prometheus <b>Namespace/Pod</b> がサンプルを取り入れていない。
<b>PrometheusTargetScrapesDuplicate</b>	<b>warning</b>	要約: Prometheus の多くのサンプルが拒否されている。説明: <b>Namespace/Pod</b> には、重複したタイムスタンプ (ただし異なる値を含む) により多くのサンプルが拒否されている。
<b>EtcdInsufficientMembers</b>	<b>critical</b>	Etcd クラスター " <b>Job</b> ": メンバーが不十分 (X)。
<b>EtcdNoLeader</b>	<b>critical</b>	Etcd クラスター " <b>Job</b> ": メンバー <b>Instance</b> にリーダーがない。
<b>EtcdHighNumberOfLeaderChanges</b>	<b>warning</b>	Etcd クラスター " <b>Job</b> ": インスタンス <b>Instance</b> で1時間以内に X leader 変更が生じる。
<b>EtcdHighNumberOfFailedGRPCRequests</b>	<b>warning</b>	Etcd クラスター " <b>Job</b> ": <b>GRPC_Method</b> についての X% の要求が etcd インスタンス <b>Instance</b> で失敗。
<b>EtcdHighNumberOfFailedGRPCRequests</b>	<b>critical</b>	Etcd クラスター " <b>Job</b> ": <b>GRPC_Method</b> についての X% の要求が etcd インスタンス <b>Instance</b> で失敗。
<b>EtcdGRPCRequestsSlow</b>	<b>critical</b>	Etcd クラスター " <b>Job</b> ": <b>GRPC_Method</b> の gRPC 要求に X_s on etcd instance <b>_Instance</b> がかかっている。
<b>EtcdMemberCommunicationSlow</b>	<b>warning</b>	Etcd クラスター " <b>Job</b> ": To とのメンバー通信に X_s on etcd instance <b>_Instance</b> がかかっている。

アラート	重要度	説明
<b>EtcHighNumberOfFailedProposals</b>	<b>warning</b>	Etcd クラスター "Job": etcd インスタンス Instance での1時間以内の X proposal の失敗。
<b>EtcHighFsyncDurations</b>	<b>warning</b>	Etcd クラスター "Job": 99 番目のパーセントイルの fync 期間は X_s on etcd instance Instance。
<b>EtcHighCommitDurations</b>	<b>warning</b>	Etcd クラスター "Job": 99 番目のパーセントイルのコミット期間 X_s on etcd instance Instance
<b>FdExhaustionClose</b>	<b>warning</b>	Job インスタンス Instance がそのファイル記述子をすぐに使い切る。
<b>FdExhaustionClose</b>	<b>critical</b>	Job インスタンス Instance がそのファイル記述子をすぐに使い切る。

## 5.4. ETCD モニタリングの設定

**etcd** サービスが正常に実行されない場合、OpenShift Container Platform クラスター全体の運用に支障が及ぶ可能性があります。そのため、**etcd** のモニタリングを設定することができます。

以下の手順に従って **etcd** モニタリングを設定します。

### 手順

1. モニタリングスタックが実行中であることを確認します。

```
$ oc -n openshift-monitoring get pods
NAME                                READY   STATUS    RESTARTS   AGE
alertmanager-main-0                 3/3    Running   0          34m
alertmanager-main-1                 3/3    Running   0          33m
alertmanager-main-2                 3/3    Running   0          33m
cluster-monitoring-operator-67b8797d79-sphxj 1/1    Running   0          36m
grafana-c66997f-pxrf7               2/2    Running   0          37s
kube-state-metrics-7449d589bc-rt4mq 3/3    Running   0          33m
node-exporter-5tt4f                 2/2    Running   0          33m
node-exporter-b2mrp                 2/2    Running   0          33m
node-exporter-fd52p                 2/2    Running   0          33m
node-exporter-hfqgv                 2/2    Running   0          33m
prometheus-k8s-0                    4/4    Running   1          35m
prometheus-k8s-1                    0/4    ContainerCreating 0          21s
prometheus-operator-6c9fddd47f-9jfgk 1/1    Running   0          36m
```

2. クラスターモニタリングスタックの設定ファイルを開きます。

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

3. **config.yaml**: |+ の下に、**etcd** セクションを追加します。

- a. マスターノードの静的 Pod で **etcd** を実行する場合、セレクターを使用して **etcd** ノードを指定できます。

```
...
data:
  config.yaml: |+
  ...
  etcd:
    targets:
      selector:
        openshift.io/component: etcd
        openshift.io/control-plane: "true"
```

- b. 別のホストで **etcd** を実行する場合、IP アドレスを使用してノードを指定する必要があります。

```
...
data:
  config.yaml: |+
  ...
  etcd:
    targets:
      ips:
        - "127.0.0.1"
        - "127.0.0.2"
        - "127.0.0.3"
```

**etcd** ノードの IP アドレスが変更される場合は、この一覧を更新する必要があります。

4. **etcd** サービスモニターが実行されていることを確認します。

```
$ oc -n openshift-monitoring get servicemonitor
NAME          AGE
alertmanager  35m
etcd          1m ①
kube-apiserver 36m
kube-controllers 36m
kube-state-metrics 34m
kubelet       36m
node-exporter  34m
prometheus    36m
prometheus-operator 37m
```

- ① **etcd** サービスモニター。

**etcd** サービスモニターが起動するには1分程度の時間がかかる場合があります。

5. これで、Web インターフェイスに移動して **etcd** モニタリングのステータスについての詳細を確認できます。
- a. URL を取得するには、以下を実行します。

```
$ oc -n openshift-monitoring get routes
NAME          HOST/PORT          PATH
SERVICES      PORT  TERMINATION  WILDCARD
...
prometheus-k8s  prometheus-k8s-openshift-monitoring.apps.msvistun.origin-
gce.dev.openshift.com          prometheus-k8s  web  reencrypt  None
```

- b. **https** を使用して、**prometheus-k8s** について一覧表示されている URL に移動します。ログインします。
6. ユーザーが **cluster-monitoring-view** ロールに属することを確認します。このロールは、クラスターモニタリング UI を表示するためのアクセスを提供します。  
たとえば、ユーザー **developer** を **cluster-monitoring-view** ロールに追加するには、以下を実行します。

```
$ oc adm policy add-cluster-role-to-user cluster-monitoring-view developer
```

7. Web インターフェイスで、**cluster-monitoring-view** ロールに属するユーザーとしてログインします。
8. **Status** をクリックしてから **Targets** をクリックします。**etcd** エントリーが表示される場合、**etcd** はモニターされています。

Endpoint	State	Labels	Last Scrape	Error
<a href="https://10.142.0.2:2379/metrics">https://10.142.0.2:2379/metrics</a>	DOWN	endpoint="metrics" in stance="10.142.0.2:2379" namespace="kube-system" pod="master-etcd-m- svistun-ig-m-2j9g" servi ce="etcd"	1.787s ago	Get https://10.142.0.2:2379/metrics: x509: certificate signed by unknown authority

9. **etcd** がモニターされていても、Prometheus はまだ **etcd** に対して認証できないため、メトリックスを収集できません。  
Prometheus 認証を **etcd** に対して設定するには、以下を実行します。

- a. **/etc/etcd/ca/ca.crt** および **/etc/etcd/ca/ca.key** 認証情報ファイルをマスターノードからローカルマシンにコピーします。

```
$ ssh -i gcp-dev/ssh-privatekey cloud-user@35.237.54.213
```

- b. 以下の内容を含む **openssl.cnf** ファイルを作成します。

```
[ req ]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[ req_distinguished_name ]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, keyEncipherment, digitalSignature
extendedKeyUsage=serverAuth, clientAuth
```

- c. **etcd.key** プライベートキーファイルを生成します。

```
$ openssl genrsa -out etcd.key 2048
```

- d. **etcd.csr** 証明書署名要求ファイルを生成します。

```
$ openssl req -new -key etcd.key -out etcd.csr -subj "/CN=etcd" -config openssl.cnf
```

- e. **etcd.crt** 証明書ファイルを生成します。

```
$ openssl x509 -req -in etcd.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out etcd.crt -days 365 -extensions v3_req -extfile openssl.cnf
```

- f. 認証情報を OpenShift Container Platform で使用される形式で配置します。

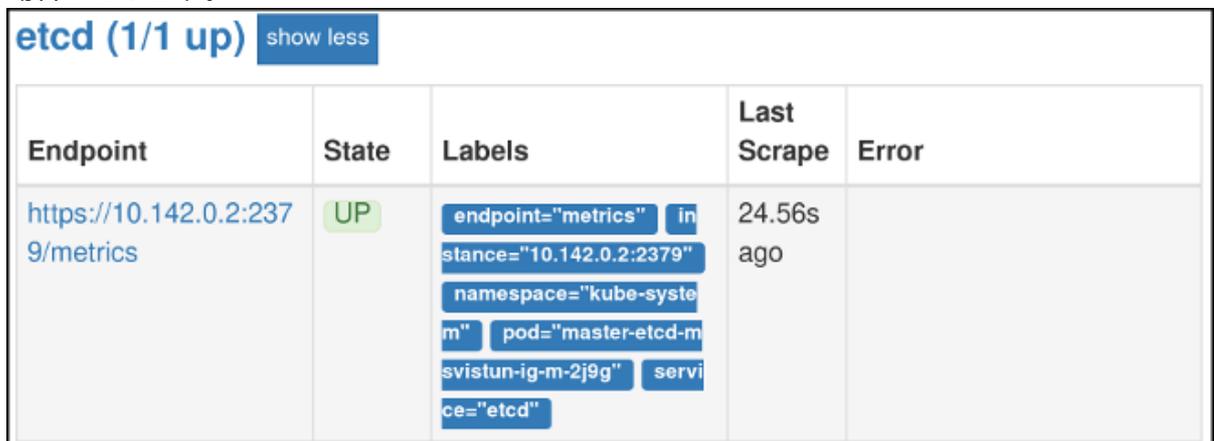
```
$ cat <<-EOF > etcd-cert-secret.yaml
apiVersion: v1
data:
  etcd-client-ca.crt: "$(cat ca.crt | base64 --wrap=0)"
  etcd-client.crt: "$(cat etcd.crt | base64 --wrap=0)"
  etcd-client.key: "$(cat etcd.key | base64 --wrap=0)"
kind: Secret
metadata:
  name: kube-etcd-client-certs
  namespace: openshift-monitoring
type: Opaque
EOF
```

これにより、**etcd-cert-secret.yaml** ファイルが作成されます。

- g. 認証情報ファイルをクラスターに適用します。

```
$ oc apply -f etcd-cert-secret.yaml
```

10. 認証を設定してから、Web インターフェイスの **Targets** ページに再度アクセスします。**etcd** が正常にモニターされていることを確認します。変更が有効になるまでに数分の時間がかかる場合があります。



Endpoint	State	Labels	Last Scrape	Error
<a href="https://10.142.0.2:2379/9/metrics">https://10.142.0.2:2379/9/metrics</a>	UP	endpoint="metrics" in stance="10.142.0.2:2379" namespace="kube-system" pod="master-etcd-m svistun-ig-m-2j9g" servi ce="etcd"	24.56s ago	

11. OpenShift Container Platform の更新時に **etcd** モニターリングを自動的に更新する必要がある場合、Ansible インベントリーファイルのこの変数を **true** に設定します。

```
openshift_cluster_monitoring_operator_etcd_enabled=true
```

別のホストで **etcd** を実行する場合、この Ansible 変数を使用して IP アドレスでノードを指定します。

```
openshift_cluster_monitoring_operator_etcd_hosts=[<address1>, <address2>, ...]
```

**etcd** ノードの IP アドレスが変更される場合は、この一覧を更新する必要があります。

## 5.5. PROMETHEUS、ALERTMANAGER、および GRAFANA へのアクセス

OpenShift Container Platform Monitoring には、クラスターモニタリング用の Prometheus インスタンスおよび中心的な Alertmanager クラスターが同梱されます。Prometheus および Alertmanager に加えて、OpenShift Container Platform Monitoring には [Grafana](#) インスタンスおよびクラスターモニタリングのトラブルシューティング向けの事前にビルドされたダッシュボードが含まれます。モニタリングスタックおよびダッシュボードと共に提供される Grafana インスタンスは読み取り専用です。

Prometheus、Alertmanager、および Grafana Web UI にアクセスするためのアドレスを取得するには、以下を実行します。

### 手順

1. 次のコマンドを実行します。

```
$ oc -n openshift-monitoring get routes
NAME          HOST/PORT
alertmanager-main alertmanager-main-openshift-monitoring.apps._url_.openshift.com
grafana       grafana-openshift-monitoring.apps._url_.openshift.com
prometheus-k8s prometheus-k8s-openshift-monitoring.apps._url_.openshift.com
```

**https://** をこれらのアドレスに追加します。暗号化されていない接続を使用して UI にアクセスすることはできません。

2. 認証は、OpenShift Container Platform アイデンティティに対して行われ、OpenShift Container Platform の他の場所で使用されるのと同じ認証情報および認証方法が使用されます。**cluster-monitoring-view** クラスターロールなどの、すべての namespace への読み取りアクセスを持つロールを使用する必要があります。

## 第6章 RED HAT レジストリーへのアクセスおよびその設定

### 6.1. 認証が有効にされている RED HAT レジストリー

Red Hat Container Catalog で利用可能なすべてのコンテナイメージはイメージレジストリーの **registry.access.redhat.com** でホストされます。OpenShift Container Platform 3.11 では、Red Hat Container Catalog は **registry.access.redhat.com** から **registry.redhat.io** に移行しました。

新規レジストリーの **registry.redhat.io** ではイメージおよび OpenShift Container Platform でホストされるコンテンツへのアクセスに認証が必要になります。新規レジストリーへの移行後も、既存レジストリーはしばらく利用可能になります。



#### 注記

OpenShift Container Platform はイメージを **registry.redhat.io** からプルするため、これを使用できるようにクラスターを設定する必要があります。

新規レジストリーは、以下の方法を使用して認証に標準の OAuth メカニズムを使用します。

- **認証トークン。** **管理者によって生成される** これらのトークンは、システムにコンテナイメージレジストリーに対する認証機能を付与するサービスアカウントです。サービスアカウントはユーザーアカウントの変更による影響を受けないので、トークンの認証方法は信頼性があり、回復性があります。これは、実稼働クラスター用にサポートされている唯一の認証オプションです。
- **Web ユーザー名およびパスワード。** これは、**access.redhat.com** などのリソースへのログインに使用する標準的な認証情報のセットです。OpenShift Container Platform でこの認証方法を使用することはできますが、これは実稼働デプロイメントではサポートされません。この認証方法の使用は、OpenShift Container Platform 外のスタンドアロンのプロジェクトに制限されます。

ユーザー名またはパスワード、または認証トークンのいずれかの認証情報を使用して **docker login** を使用し、新規レジストリーのコンテンツにアクセスします。

すべてのイメージストリームは新規レジストリーを参照します。新規レジストリーにはアクセスに認証が必要となるため、OpenShift namespace には **imagestreamsecret** という新規シークレットがあります。

認証情報は 2 つの場所に配置する必要があります。

- **OpenShift namespace** OpenShift namespace のイメージストリームがインポートできるように、認証情報は OpenShift namespace に存在する必要があります。
- **ホスト。** Kubernetes でイメージをプルする際にホストの認証情報を使用するため、認証情報はホスト上になければなりません。

新規レジストリーにアクセスするには、以下を実行します。

- イメージインポートシークレット **imagestreamsecret** が OpenShift namespace にあることを確認します。そのシークレットには、新規レジストリーへのアクセスを許可する認証情報があります。
- クラスターノードのすべてに、マスターからコピーされた **/var/lib/origin/.docker/config.json** が含まれていることを確認します。これは、Red Hat レジストリーへのアクセスを許可します。

### 6.1.1. ユーザーアカウントの作成

Red Hat 製品のエンタイトルメントを持たれる Red Hat のお客様は、ユーザー認証情報を持つアカウントをお持ちです。これは、お客様が Red Hat カスタマーポータルにログインされる際に使用されるユーザー名およびパスワードです。

アカウントをお持ちでない場合は、以下のオプションのいずれかに登録してアカウントを無料で取得することができます。

- **Red Hat Developer Program**。このアカウントを使用すると、開発者の各種ツールおよびプログラムにアクセスできます。
- **30 日間のトライアルサブスクリプション**。このアカウントを使用すると、一部の Red Hat ソフトウェア製品にアクセスできる 30 日間のトライアルサブスクリプションを利用できます。

### 6.1.2. Red Hat レジストリー

お客様の企業が共有アカウントを管理されている場合には、トークンを作成する必要があります。管理者は、組織に関連付けられたすべてのトークンを作成し、表示し、削除することができます。

#### 前提条件

- ユーザー認証情報

#### 手順

トークンを順番に作成するには、**docker login** を実行します。

1. **registry.redhat.io** に移動します。
2. Red Hat Network (RHN) のユーザー名とパスワードでログインします。
3. プロンプトが出されたら、同意書を読んでこれに同意します。
  - 同意書の同意を求めるプロンプトがすぐに出されない場合、以下の手順に進むとプロンプトが出されます。
4. **Registry Service Accounts** ページから、**Create Service Account** をクリックします。
  - a. サービスアカウントの名前を指定します。これには、ランダムな文字列が追加されます。
  - b. 説明を入力します。
  - c. **Create** をクリックします。
5. サービスアカウントに戻ります。
6. 作成したサービスアカウントをクリックします。
7. 追加された文字列を含むユーザー名をコピーします。
8. トークンをコピーします。

### 6.1.3. インストールおよびアップグレード時のレジストリー認証情報の管理

レジストリー認証情報は、Ansible インストーラーを使用してインストールまたはアップグレード時に管理することもできます。

これにより、以下がセットアップされます。

- OpenShift namespace の **imagestreamsecret**。
- すべてのノードの認証情報。

Ansible インストーラーでは、**openshift\_examples\_registryurl** または **oreg\_url** のいずれかのデフォルト値の **registry.redhat.io** を使用している場合に認証情報が必要になります。

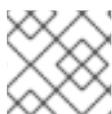
### 前提条件

- ユーザー認証情報
- サービスアカウント
- サービスアカウントトークン

### 手順

Ansible インストーラーを使用してインストールまたはアップグレード時にレジストリー認証情報を管理するには、以下を実行します。

- インストールまたはアップグレード時に、インストーラーインベントリーに **oreg\_auth\_user** および **oreg\_auth\_password** 変数を指定します。



### 注記

トークンを作成した場合、**oreg\_auth\_password** をトークンの値に設定します。

追加の認証されたレジストリーへのアクセスを必要とするクラスターは、**openshift\_additional\_registry\_credentials** を設定してレジストリーの一覧を設定できます。各レジストリーには、ホストおよびパスワードの値が必要であり、ユーザー名はユーザーを設定して指定できます。デフォルトで、指定された認証情報は、指定されたレジストリーでイメージ **openshift3/ose-pod** の検査を試行することによって検証されます。

代替イメージを指定するには、以下のいずれかを実行します。

- **test\_image** を設定します。
- **test\_login** を False に設定して認証情報の検証を無効にします。

レジストリーがセキュアでない場合、**tls\_verify** を False に設定します。

この一覧のすべての認証情報には、OpenShift namespace で作成された **imagestreamsecret** およびすべてのノードにデプロイされた認証情報が含まれます。

以下に例を示します。

```
openshift_additional_registry_credentials=
[{'host':'registry.example.com','user':'name','password':'pass1','test_login':'False'},
{'host':'registry2.example.com','password':'token12345','tls_verify':'False','test_image':'mongodb/mongod
b'}]
```

## 6.1.4. Red Hat レジストリーでのサービスアカウントの使用

Red Hat レジストリーのサービスアカウントを作成し、トークンを生成した後に、追加のタスクを実行できます。



## 注記

このセクションでは、インストールおよびアップグレード時のレジストリー認証情報の管理セクションで説明されているインベントリー変数を指定することによってインストール時に自動的に実行できる手動の手順について説明します。

## 前提条件

- ユーザー認証情報
- サービスアカウント
- サービスアカウントトークン

## 手順

[Registry Service Accounts](#) ページから、アカウント名をクリックします。このページから以下のタスクを実行できます。

- **Token Information** タブで、ユーザー名 (指定したランダムな文字列が追加された名前) およびパスワード (トークン) を表示できます。このタブで、トークンを再生成できます。
- **OpenShift Secret** タブで以下を実行できます。

- タブにあるリンクをクリックしてシークレットをダウンロードします。
- シークレットをクラスターに送信します。

```
# oc create -f <account-name>-secret.yml --namespace=openshift
```

- シークレットの参照を **imagePullSecrets** フィールドで Kubernetes Pod 設定に追加して Kubernetes 設定を更新します。

```
apiVersion: v1
kind: Pod
metadata:
  name: somepod
  namespace: all
spec:
  containers:
    - name: web
      image: registry.redhat.io/REPONAME

  imagePullSecrets:
    - name: <numerical-string-account-name>-pull-secret
```

- **Docker Login** タブで、**docker login** を実行できます。以下に例を示します。

```
# docker login -u='<numerical-string|account-name>'
-p=<token>
```

正常にログインした後に、`~/.docker/config.json` を `/var/lib/origin/.docker/config.json` にコピーし、ノードを再起動します。

```
# cp -r ~/.docker /var/lib/origin/  
systemctl restart atomic-openshift-node
```

- **Docker Configuration** タブで、以下を実行できます。
  - a. タブにあるリンクをクリックして認証情報の設定をダウンロードします。
  - b. ファイルを Docker 設定ディレクトリーに配置して設定をディスクに書き込みます。これにより、既存の認証情報が上書きされます。以下に例を示します。

```
# mv <account-name>-auth.json ~/.docker/config.json
```

## 第7章 マスターとノードの設定

### 7.1. インストール後のマスターおよびノード設定のカスタマイズ

**openshift start** コマンド (マスターサーバー向け) と **hyperkube** コマンド (ノードサーバー向け) に指定できる引数には限りがありますが、開発または実験的な環境でサーバーを起動するには十分です。ただしこれらの引数は、実稼働環境に必要な設定およびセキュリティオプションのセットを詳細に記述し、管理するには不十分です。

これらのオプションを指定するには、`/etc/origin/master/master-config.yaml` の **マスター設定ファイル**、および **ノード設定マップ** でこれらのオプションを指定する必要があります。上記のファイルは、デフォルトプラグインの上書き、etcd への接続、サービスアカウントの自動作成、イメージ名の作成、プロジェクト要求のカスタマイズ、ボリュームプラグインの設定などの各種オプションを定義します。

このトピックでは、OpenShift Container Platform のマスターとノードのホストのカスタマイズに使用できるオプションについて説明し、インストール後に設定を変更する方法を紹介します。

これらのファイルはデフォルト値なしで完全に指定されます。したがって、空の値は、ユーザーがそのパラメーターを空の値で起動しようとしていることを意味します。これによりユーザーの設定を正確に推測することを簡単になりますが、指定する必要があるすべてのオプションを思い出す必要があるという点では容易な作業にはなりません。これをより容易にするには、設定ファイルを **--write-config** オプションを使って作成し、次に **--config** オプションを指定して使用することができます。

### 7.2. インストールの依存関係

実稼働環境は、標準の **クラスターインストール** プロセスを使用してインストールする必要があります。実稼働環境では、**高可用性 (HA)** を維持するために **複数のマスター** を使用することが適しています。3つのマスターで設定されるクラスターアーキテクチャーが推奨され、**HAproxy** の使用が推奨されるソリューションになります。

#### 注意

etcd が **マスターホスト** にインストールされている場合は、etcd は権限を持つマスターを判別できないために、クラスターを3つ以上のマスターを使用するように設定する必要があります。2つのマスターのみを正常に実行できるのは、etcd をマスター以外のホストにインストールしている場合です。

### 7.3. マスターとノードの設定

マスターとノードの設定ファイルの設定に使用する方法は、OpenShift Container Platform クラスターのインストールに使用した方法と同じでなければなりません。標準の **クラスターインストール** プロセスに従う場合は、Ansible インベントリーファイルで設定の変更を加えます。

### 7.4. ANSIBLE を使用した設定の変更

このセクションは、Ansible に精通していることを前提に説明を行います。

**Ansible に公開** されているのは、利用可能なホスト設定オプションの一部のみです。OpenShift Container Platform のインストール後、Ansible はインベントリーファイルを置き換えられた値で作成します。このインベントリーファイルを変更し、Ansible インストーラー Playbook を再実行することで、OpenShift Container Platform クラスターをカスタマイズできます。

OpenShift Container Platform は Ansible をクラスターインストールで使用することに対応していますが、Ansible Playbook とインベントリーファイルを使うことで、[Puppet](#)、[Chef](#)、[Salt](#) などの他の管理ツールを使用することもできます。

## ユースケース: HTPasswd 認証を使用するようにクラスターを設定する



### 注記

- このユースケースは、Playbook で参照されているすべてのノードに [SSH キー](#) がセットアップされていることを前提としています。
- **htpasswd** ユーティリティーは **httpd-tools** パッケージにあります。

```
# yum install httpd-tools
```

Ansible インベントリーを変更し、設定の変更を行うには、以下を実行します。

1. `./hosts` インベントリーファイルを開きます。
2. 新規の変数をファイルの `[OSEv3:vars]` セクションに追加します。

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge':
'true', 'kind': 'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
#openshift_master_htpasswd_users={'<name>': '<hashed-password>', '<name>': '<hashed-
password>'}
# or
#openshift_master_htpasswd_file=/etc/origin/master/htpasswd
```

HTPasswd 認証の場合、**openshift\_master\_identity\_providers** 変数はその認証タイプを有効にします。HTPasswd を使用する 3 つの異なる認証オプションを設定できます。

- `/etc/origin/master/htpasswd` がすでに設定されており、ホスト上にある場合には、**openshift\_master\_identity\_providers** のみを指定します。
- ローカル htpasswd ファイルをホストにコピーするには、**openshift\_master\_identity\_providers** と **openshift\_master\_htpasswd\_file** の両方を指定します。
- ホストで新規 htpasswd ファイルを生成するには、**openshift\_master\_identity\_providers** と **openshift\_master\_htpasswd\_users** の両方を指定します。

OpenShift Container Platform は、HTPasswd 認証を設定するためにハッシュ化されたパスワードを必要とします。以下のセクションに示されるように `htpasswd` コマンドを使用してユーザー用のハッシュ化されたパスワードを生成するか、またはユーザーおよび関連付けられたハッシュ化されたパスワードを持つフラットファイルを作成することができます。

以下の例では、認証方法をデフォルトの **deny all** 設定から **htpasswd** に変更し、指定されたファイルを使って **jsmith** および **bloblaw** ユーザーのユーザー ID とパスワードを生成します。

```
# htpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge':
'true', 'kind': 'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
```

```
openshift_master_htpasswd_users={'jsmith': '$apr1$wIwXkFLI$bAygTKGmPOqaJftB',
'bloblaw': '7IRJ$2ODmeLoxf4I6sUEKfiA$2aDJqLJe'}
# or
#openshift_master_htpasswd_file=/etc/origin/master/htpasswd
```

3. 変更を有効にするために、Ansible Playbook を再実行します。

```
$ ansible-playbook -b -i ./hosts ~/src/openshift-ansible/playbooks/deploy_cluster.yml
```

Playbook が設定を更新し、OpenShift Container Platform マスターサービスを再起動して変更を適用します。

これで、Ansible を使用したマスターとノードの設定ファイルの変更が完了しました。ここまでは単純なユースケースですが、次は、どの [マスター](#) と [ノードの設定](#) オプションが [Ansible に公開](#) されているかを確認し、各自の Ansible インベントリーをカスタマイズします。

### 7.4.1. htpasswd コマンドの使用

OpenShift Container Platform クラスタを HTTPasswd 認証を使用するように設定するには、ハッシュ化されたパスワードを持つ1名以上のユーザーを [インベントリーファイル](#) に追加する必要があります。

以下を行うことができます。

- [ユーザー名とパスワードを生成](#) して `./hosts` インベントリーファイルに直接追加する。
- [フラットファイルを作成](#) して認証情報を `./hosts` インベントリーファイルに渡す。

ユーザーおよびハッシュ化されたパスワードを作成するには、以下を実行します。

1. 以下のコマンドを実行して指定されたユーザーを追加します。

```
$ htpasswd -n <user_name>
```



#### 注記

**-b** オプションを追加すると、パスワードをコマンドラインに指定できます。

```
$ htpasswd -nb <user_name> <password>
```

2. ユーザーのクリアテキストのパスワードを入力し、確定します。  
以下に例を示します。

```
$ htpasswd -n myuser
New password:
Re-type new password:
myuser:$apr1$vdW.cl3j$WSKIOzUPs6Q
```

コマンドにより、ハッシュされたバージョンのパスワードが生成されます。

これで、[HTTPasswd 認証](#) を設定する際にハッシュ化パスワードを使用できます。ハッシュ化パスワードは、:の後に続く文字列です。上記の例では、次を入力します。

```
openshift_master_htpasswd_users={'myuser': '$apr1$wIwXkFLI$bAygtlSk2eKGmqaJftB'}
```

ユーザー名とハッシュ化パスワードを持つフラットファイルを作成するには、以下を実行します。

1. 以下のコマンドを実行します。

```
$ htpasswd -c /etc/origin/master/htpasswd <user_name>
```



#### 注記

**-b** オプションを追加すると、パスワードをコマンドラインに指定できます。

```
$ htpasswd -c -b <user_name> <password>
```

2. ユーザーのクリアテキストのパスワードを入力し、確定します。  
以下に例を示します。

```
htpasswd -c /etc/origin/master/htpasswd user1
New password:
Re-type new password:
Adding password for user user1
```

このコマンドは、ユーザー名とユーザーパスワードのハッシュ化されたバージョンを含むファイルを生成します。

これで、[HTPasswd 認証](#) を設定する際にこのパスワードファイルを使用できます。



#### 注記

**htpasswd** コマンドについての詳細は、[HTPasswd Identity Provider](#) を参照してください。

## 7.5. 手動による設定変更

ユースケース: [HTPasswd 認証](#) を使用するようにクラスターを設定する

設定ファイルを手動で変更するには、以下を実行します。

1. 変更する必要がある設定ファイルを開きます。ここでは `/etc/origin/master/master-config.yaml` ファイルです。
2. 以下の新規変数をファイルの **identityProviders** スタンザに追加します。

```
oauthConfig:
  ...
  identityProviders:
    - name: my_htpasswd_provider
      challenge: true
      login: true
      mappingMethod: claim
      provider:
```

```
apiVersion: v1
kind: HTTPasswdPasswordIdentityProvider
file: /etc/origin/master/htpasswd
```

3. 変更を保存してファイルを閉じます。
4. 変更を有効にするために、マスターを再起動します。

```
# master-restart api
# master-restart controllers
```

これでマスターとノードの設定ファイルが手動で変更されました。ここまでは単純なユースケースです。次は、すべての [マスター](#) と [ノードの設定](#) オプションを確認し、変更を加えることでクラスターをさらにカスタマイズします。



### 注記

クラスターのノードを変更するには、[ノード設定マップ](#) を必要に応じて更新します。node-config.yaml ファイルは手動で変更しないようにしてください。

## 7.6. マスター設定ファイル

このセクションでは、master-config.yaml ファイルに記述されているパラメーターについて説明します。

[新規のマスター設定ファイルを作成](#) して、OpenShift Container Platform のインストール済みバージョンに有効なオプションを確認できます。



### 重要

master-config.yaml ファイルを変更する際には常にマスターを再起動して変更を有効にしてください。[OpenShift Container Platform サービスの再起動](#) を参照してください。

### 7.6.1. 受付制御の設定

表7.1 受付制御設定パラメーター

パラメーター名	説明
<b>AdmissionConfig</b>	<a href="#">受付制御プラグイン</a> 設定が含まれています。OpenShift Container Platform には、API オブジェクトが作成または変更されるたびにトリガーされる、受付制御プラグインの設定可能な一覧が含まれます。このオプションを使用して、一部のプラグインの無効化、他の設定の追加、順序の変更や設定の指定など、デフォルトのプラグイン一覧を上書きできます。プラグインの一覧とその設定はどちらも Ansible で制御することができます。

パラメーター名	説明
<b>APIServerArguments</b>	<p>API サーバーのコマンドライン引数に一致する Kube API サーバーに直接渡されるキーと値のペアです。これらは移行されませんが、存在しない値が参照されてもサーバーは起動しません。これらの値は、<b>KubernetesMasterConfig</b> の他の設定を上書きする場合があります、これにより無効な設定が生じる可能性があります。</p> <p>す。<b>APIServerArguments</b> を <b>event-ttl</b> という値で使用し、イベントを etcd に保存します。デフォルトは <b>2h</b> ですが、メモリーの増加を防ぐためにより小さい値に設定することができます。</p> <pre> apiServerArguments:   event-ttl:     - "15m" </pre>
<b>ControllerArguments</b>	<p>Kube コントローラーマネージャーのコマンドライン引数に一致する、コントローラーマネージャーに直接渡されるキーと値のペアです。これらは移行されませんが、存在しない値が参照されてもサーバーは起動しません。これらの値は、<b>KubernetesMasterConfig</b> の他の設定を上書きする場合があります、これにより無効な設定が生じる可能性があります。</p>
<b>DefaultAdmissionConfig</b>	<p>各種の受付プラグインの有効化または無効化に使用されます。このタイプが <b>pluginConfig</b> の下に <b>configuration</b> オブジェクトとして存在し、受付プラグインがこれをサポートしている場合、<b>デフォルトでオフ</b> にされている受付プラグインが有効になります。</p>
<b>PluginConfig</b>	<p>設定ファイルを受付制御プラグインごとに指定することができます。</p>
<b>PluginOrderOverride</b>	<p>マスターにインストールされる受付制御プラグイン名の一覧です。順序には意味があります。空の場合は、プラグインのデフォルトの一覧が使用されます。</p>
<b>SchedulerArguments</b>	<p>スケジューラーのコマンドライン引数に一致する、Kube スケジューラーに直接渡されるキーと値のペアです。これらは移行されませんが、存在しない値が参照されてもサーバーは起動しません。これらの値は、<b>KubernetesMasterConfig</b> の他の設定を上書きする場合があります、これにより無効な設定が生じる可能性があります。</p>

## 7.6.2. アセットの設定

表7.2 アセットの設定パラメーター

パラメーター名	説明
---------	----

パラメーター名	説明
<b>AssetConfig</b>	<p>これが存在する場合には、アセットサーバーは事前に定義されたパラメーターに基づいて起動します。以下に例を示します。</p> <pre>assetConfig:   logoutURL: ""   masterPublicURL: https://master.ose32.example.com:8443   publicURL: https://master.ose32.example.com:8443/console/   servingInfo:     bindAddress: 0.0.0.0:8443     bindNetwork: tcp4     certFile: master.server.crt     clientCA: ""     keyFile: master.server.key     maxRequestsInFlight: 0     requestTimeoutSeconds: 0</pre>
<b>corsAllowedOrigins</b>	<p>異なるホスト名を使用して Web アプリケーションから API サーバーにアクセスするには、設定フィールドに <b>corsAllowedOrigins</b> を指定するか、または <b>--cors-allowed-origins</b> オプションを <b>openshift start</b> に指定してそのホスト名をホワイトリスト化する必要があります。その値の固定 (pinning) やエスケープは実行されません。使用例については、<a href="#">Web Console</a> を参照してください。</p>
<b>DisabledFeatures</b>	<p>起動することのできない機能の一覧です。null に設定してください。この機能を手動で無効にする必要性はほとんどなく、この操作を実行することも推奨されません。</p>
<b>Extensions</b>	<p>サブコンテキストの下位のアセットサーバーファイルシステムから提供されるファイルです。</p>
<b>ExtensionDevelopment</b>	<p>true に設定されている場合、起動時だけでなく要求が出されるたびに拡張機能スクリプトとスタイルシートをリロードするようアセットサーバーに指示します。変更のたびにサーバーを再起動しなくても、拡張機能を展開することができます。</p>
<b>ExtensionProperties</b>	<p>コンソールのグローバル変数 <b>OPENSHIFT_EXTENSION_PROPERTIES</b> の下に挿入されるキー - (文字列) 値 - (文字列) のペアです。</p>
<b>ExtensionScripts</b>	<p>Web コンソールが読み込む際にスクリプトとして読み込まれるアセットサーバーファイル上のファイルパスです。</p>
<b>ExtensionStylesheets</b>	<p>Web コンソールが読み込む際にスタイルシートとして読み込まれるアセットサーバーファイル上のファイルパスです。</p>
<b>LoggingPublicURL</b>	<p>ロギング用のパブリックエンドポイント (オプション) です。</p>

パラメーター名	説明
<b>LogoutURL</b>	Web コンソールからログアウトした後に Web ブラウザーをリダイレクトするオプションの絶対 URL です。指定されていない場合は、ビルトインされたログアウトページが表示されます。
<b>MasterPublicURL</b>	Web コンソールが OpenShift Container Platform サーバーにアクセスする方法について示します。
<b>MetricsPublicURL</b>	メトリクス用のパブリックエンドポイント (オプション) です。
<b>PublicURL</b>	アセットサーバーの URL です。

### 7.6.3. 認証と認可の設定

表7.3 認証および認可パラメーター

パラメーター名	説明
<b>authConfig</b>	認証および認可設定オプションを保持します。
<b>AuthenticationCacheSize</b>	キャッシュされる認証結果の数を示します。0 の場合は、デフォルトのキャッシュサイズが使用されます。
<b>AuthorizationCacheTTL</b>	承認結果がキャッシュされる期間を示します。有効な時間文字列 (5m など) を取り、空の場合はデフォルトのタイムアウトが取得されます。空白の場合は、デフォルトのタイムアウトが取得されます。ゼロの場合 (0m など) キャッシュは無効です。

### 7.6.4. コントローラーの設定

表7.4 コントローラー設定パラメーター

パラメーター名	説明
<b>Controllers</b>	起動するコントローラーの一覧です。none に設定されている場合、コントローラーは自動的に起動されません。デフォルト値は * であり、これによりすべてのコントローラーが起動します。* を使用している場合は、コントローラーの名前の先頭に - を追加することでそのコントローラーを除外できます。この時点で他の値は認識されません。
<b>ControllerLeaseTTL</b>	コントローラーの選択を有効にし、マスターに対してコントローラーが起動する前にリースを取得するように指示して、この値で定義された秒数内にリースを更新します。負の値以外の値を設定することで <b>pauseControllers=true</b> が強制的に実行されます。デフォルトの値はオフ (0 または省略) であり、コントローラーの選択は -1 で無効にできません。

パラメーター名	説明
<b>PauseControllers</b>	マスターに対してコントローラーを自動的に開始せず、起動前にサーバーへの通知が受信するまで待機するように指示します。

### 7.6.5. etcd の設定

表7.5 etcd 設定パラメーター

パラメーター名	説明
<b>Address</b>	etcd へのクライアント接続用に公開される host:port です。
<b>etcdClientInfo</b>	etcd に接続する方法についての情報が含まれます。etcd を組み込みまたは組み込み以外の方法で実行するかどうかを指定し、ホストを指定します。残りの設定は Ansible インベントリで処理されます。以下に例を示します。 <pre> etcdClientInfo:   ca: ca.crt   certFile: master.etcd-client.crt   keyFile: master.etcd-client.key   urls:     - https://m1.aos.example.com:4001 </pre>
<b>etcdConfig</b>	これがある場合、etcd は定義されたパラメーターに基づいて起動します。以下に例を示します。 <pre> etcdConfig:   address: master.ose32.example.com:4001   peerAddress: master.ose32.example.com:7001   peerServingInfo:     bindAddress: 0.0.0.0:7001     certFile: etcd.server.crt     clientCA: ca.crt     keyFile: etcd.server.key   servingInfo:     bindAddress: 0.0.0.0:4001     certFile: etcd.server.crt     clientCA: ca.crt     keyFile: etcd.server.key   storageDirectory: /var/lib/origin/openshift.local.etcd </pre>
<b>etcdStorageConfig</b>	API リソースを etcd に格納する方法についての情報が含まれます。これらの値は、etcd がクラスターのバックストアである場合にのみ関連する値になります。
<b>KubernetesStoragePrefix</b>	Kubernetes のリソースがその下位に置かれる etcd 内のパスです。この値が変更されると etcd 内の既存のオブジェクトは検索できなくなります。デフォルトの値は <code>kubernetes.io</code> です。

パラメーター名	説明
<b>KubernetesStorageVersion</b>	etcd 内の Kubernetes リソースがシリアルライズされる API バージョン。etcd から読み取りを行うクラスター内のすべてのクライアントが新規バージョンの読み取りを可能にするコードを取得するまでこの値を変更することができません。
<b>OpenShiftStoragePrefix</b>	OpenShift Container Platform リソースがその下位に置かれる etcd 内のパスです。この値が変更されると、etcd 内の既存のオブジェクトは検索できなくなります。デフォルトの値は <code>openshift.io</code> です。
<b>OpenShiftStorageVersion</b>	etcd 内の OS リソースがシリアルライズされる API バージョンです。etcd から読み取りを行うクラスター内のすべてのクライアントが新規バージョンの読み取りを可能にするコードを取得するまで、この値を変更することができません。
<b>PeerAddress</b>	etcd へのピア接続用に公開される <code>host:port</code> です。
<b>PeerServingInfo</b>	etcd ピアの提供を開始する方法を記述します。
<b>ServingInfo</b>	提供を開始する方法を記述します。以下に例を示します。  <pre> servingInfo:   bindAddress: 0.0.0.0:8443   bindNetwork: tcp4   certFile: master.server.crt   clientCA: ca.crt   keyFile: master.server.key   maxRequestsInFlight: 500   requestTimeoutSeconds: 3600 </pre>
<b>StorageDir</b>	etcd ストレージディレクトリーへのパスです。

### 7.6.6. 付与の設定

表7.6 付与の設定パラメーター

パラメーター名	説明
<b>GrantConfig</b>	付与を処理する方法を記述します。
<b>GrantHandlerAuto</b>	クライアントの承認付与の要求を自動承認します。
<b>GrantHandlerDeny</b>	クライアントの認証付与要求を自動的に拒否します。
<b>GrantHandlerPrompt</b>	ユーザーに対し、クライアントの新規の認証付与要求を承認することを求めるプロンプトを出します。

パラメーター名	説明
<b>Method</b>	<p>OAuth クライアントが付与を要求したときに使用するデフォルトのストラテジーを決定します。この方法は特定の OAuth クライアントが独自のストラテジーを提供しない場合にのみ使用します。付与を処理するための有効な方法は以下の通りです。</p> <ul style="list-style-type: none"> <li>● auto: 付与要求を常に承認します。信頼されるクライアントの場合に役立ちます。</li> <li>● prompt: エンドユーザーに対し、付与要求の承認を求めるプロンプトを出します。サードパーティーのクライアントの場合に役立ちます。</li> <li>● deny: 付与要求を常に拒否します。ブラックリスト化されたクライアントの場合に役立ちます。</li> </ul>

### 7.6.7. イメージ設定

表7.7 イメージの設定パラメーター

パラメーター名	説明
<b>Format</b>	システムコンポーネント用に作成される名前のフォーマットです。
<b>Latest</b>	最新のタグをレジストリーからプルするかどうかを決定します。

### 7.6.8. イメージポリシーの設定

表7.8 イメージポリシー設定パラメーター

パラメーター名	説明
<b>DisableScheduledImport</b>	スケジュールされたイメージのバックグラウンドインポートの無効化を許可します。
<b>MaxImagesBulkImportedPer Repository</b>	ユーザーが Docker リポジトリーの一括インポートを行う際に、インポートされるイメージの数を制御します。デフォルトの値は5に設定され、ユーザーが誤って大量のイメージをインポートすることを防ぎます。-1に設定すると無制限になります
<b>MaxScheduledImageImports PerMinute</b>	スケジュールされたイメージストリームがバックグラウンドでインポートされる1分あたりの最大数です。デフォルト値は60です。
<b>ScheduledImageImportMinimumIntervalSeconds</b>	バックグラウンドのインポートがスケジュールされているイメージストリームが、アップストリームのリポジトリーと照合される際の最小間隔(秒単位)です。デフォルト値は15秒です。

パラメーター名	説明
<b>AllowedRegistriesForImport</b>	標準ユーザーがイメージのインポートに使用する Docker レジストリーを制限します。この一覧を、有効な Docker イメージを含むものとユーザーが信頼し、アプリケーションのインポート元となるレジストリーに設定します。Images または ImageStreamMappings を API 経由で作成するパーミッションを持つユーザーは、このポリシーによる影響を受けません。通常、これらのパーミッションを持っているのは管理者またはシステム統合管理者のみです。
<b>AdditionalTrustedCA</b>	イメージストリームのインポート時に信頼される必要のある追加の認証局を一覧表示した PEM エンコードされたファイルへのファイルパスを指定します。このファイルは API サーバープロセスによってアクセスできる必要があります。クラスターのインストール方法によっては、ファイルを API サーバー Pod にマウントする必要がある場合があります。
<b>InternalRegistryHostname</b>	デフォルトの内部イメージレジストリーのホスト名を設定します。値は <b>hostname[:port]</b> 形式である必要があります。後方互換性を考慮して、ユーザーは引き続き <b>OPENSIFT_DEFAULT_REGISTRY</b> 環境変数を使用できますが、この設定はこの環境変数を上書きします。このパラメーターが設定されると、内部レジストリーにはそのホスト名も設定される必要があります。詳細は、 <a href="#">レジストリーのホスト名の設定</a> を参照してください。
<b>ExternalRegistryHostname</b>	ExternalRegistryHostname は、デフォルトの外部イメージレジストリーのホスト名を設定します。この外部ホスト名は、イメージレジストリーが外部に公開される場合にのみ設定されます。値は ImageStreams の <b>publicDockerImageRepository</b> フィールドで使用されます。値は <b>hostname[:port]</b> 形式である必要があります。

### 7.6.9. Kubernetes のマスター設定

表7.9 Kubernetes のマスター設定パラメーター

パラメーター名	説明
<b>APILevels</b>	起動時に有効にする必要のある API レベルの一覧です (v1 など)。
<b>DisabledAPIGroupVersions</b>	無効にする必要のあるバージョン (または *) のグループのマップです。
<b>KubeletClientInfo</b>	Kubelets に接続する方法についての情報が含まれます。
<b>KubernetesMasterConfig</b>	kubelet の KubernetesMasterConfig への接続方法についての情報が含まれます。これがある場合、Kubernetes のマスターをこのプロセスで起動します。
<b>MasterCount</b>	実行されていることが予想されるマスターの数です。デフォルトの値は 1 であり、正の整数に設定できますが、-1 に設定されている場合はそれがクラスターの一部であることを示します。

パラメーター名	説明
<b>MasterIP</b>	Kubernetes リソースのパブリック IP アドレスです。空の場合、 <b>net.InterfaceAddrs</b> の最初の結果が使用されます。
<b>MasterKubeConfig</b>	このノードをマスターに接続する方法を記述した <b>.kubeconfig</b> ファイルのファイル名です。
<b>PodEvictionTimeout</b>	失敗したノード上の Pod を削除するための猶予期間を制御します。有効な時間文字列を取ります。空の場合、Pod のデフォルトのエビクションタイムアウトを取ります。デフォルトは <b>5m0s</b> です。
<b>ProxyClientInfo</b>	Pod へのプロキシ処理時に使用するクライアント証明書/キーを指定します。以下に例を示します。  <pre>proxyClientInfo:   certFile: master.proxy-client.crt   keyFile: master.proxy-client.key</pre>
<b>ServicesNodePortRange</b>	サービスのパブリックポートをホストに割り当てる際に使用される範囲です。デフォルトは 30000-32767 です。
<b>ServicesSubnet</b>	サービス IP の割り当てに使用されるサブネットです。
<b>StaticNodeNames</b>	静的に認識されるノードの一覧です。

### 7.6.10. Network Configuration

IPv4 アドレス領域はノード上のすべてのユーザーが共有するため、CIDR を以下のパラメーターで慎重に選択してください。OpenShift Container Platform はそれ自体に使用する CIDR を IPv4 アドレス領域から予約し、外部ユーザーとクラスターが共有するアドレス用の CIDR を IPv4 アドレス領域から予約します。

表7.10 ネットワーク設定パラメーター

パラメーター名	説明
<b>ClusterNetworkCIDR</b>	グローバルなオーバーレイネットワークの L3 領域を指定する CIDR 文字列です。クラスターネットワークの内部使用のために予約されています。
<b>externalIPNetworkCIDRs</b>	サービスの外部 IP フィールドで許可される値を制御します。空の場合、 <b>externalIP</b> は設定できません。これには CIDR の一覧を含めることができ、この一覧はアクセスについてチェックされます。CIDR に接頭辞 <b>!</b> が付いている場合、その CIDR の IP は拒否されます。最初に拒否が適用され、その後に IP が許可された CIDR のいずれかに対してチェックされます。セキュリティ上の理由から、この範囲はユーザーのノード、Pod、またはサービス CIDR と重複させることはできません。

パラメーター名	説明
<b>HostSubnetLength</b>	各ホストのサブネットに割り当てられるビット数です。たとえば 8 の場合は、ホスト上の /24 ネットワークを意味します。
<b>ingressIPNetworkCIDR</b>	ベアメタル上のタイプ <b>LoadBalancer</b> のサービス用に ingress IP を割り当てる範囲を制御します。そこから割り当てられる単一の CIDR を含めることができます。デフォルトは <b>172.46.0.0/16</b> に設定されています。セキュリティ上の理由から、この範囲は外部 IP、ノード、Pod、またはサービス用に予約されている CIDR と重複しないようにする必要があります。
<b>HostSubnetLength</b>	各ホストのサブネットに割り当てられるビット数です。たとえば 8 の場合は、ホスト上の /24 ネットワークを意味します。
<b>NetworkConfig</b>	<p>compiled-in-network プラグインに渡されます。ここでのオプションの多くは Ansible インベントリーで制御されます。</p> <ul style="list-style-type: none"> <li>● <b>NetworkPluginName</b> (文字列)</li> <li>● <b>ClusterNetworkCIDR</b> (文字列)</li> <li>● <b>HostSubnetLength</b> (署名なし整数)</li> <li>● <b>ServiceNetworkCIDR</b> (文字列)</li> <li>● <b>ExternalIPNetworkCIDRs</b> (文字列の配列): サービスの外部 IP フィールドで許可される値を制御します。空の場合、外部 IP を設定できません。CIDR の一覧を含むことができ、そのアクセスがチェックされます。CIDR に接頭辞 <b>!</b> が付いている場合、その CIDR の IP は拒否されます。最初に拒否が適用され、次に IP が許可された CIDR のいずれかに対してチェックされます。セキュリティ上の理由から、この範囲はユーザーのノード、Pod、またはサービス CIDR と重複しないようにする必要があります。</li> </ul> <p>以下に例を示します。</p> <pre>networkConfig:   clusterNetworks   - cidr: 10.3.0.0/16     hostSubnetLength: 8   networkPluginName: example/openshift-ovs-subnet # serviceNetworkCIDR must match kubernetesMasterConfig.servicesSubnet   serviceNetworkCIDR: 179.29.0.0/16</pre>
<b>NetworkPluginName</b>	使用されるネットワークプラグインの名前です。
<b>ServiceNetwork</b>	サービスネットワークを指定する CIDR 文字列です。

## 7.6.11. OAuth 認証設定

表7.11 OAuth 設定パラメーター

パラメーター名	説明
<b>AlwaysShowProviderSelection</b>	単一プロバイダーしかない場合でも、プロバイダーの選択ページを強制的にレンダリングします。
<b>AssetPublicURL</b>	外部アクセス用の有効なクライアントのリダイレクト URL の作成に使用されます。
<b>Error</b>	認証または付与フローでエラーページのレンダリングに使用される Go テンプレートを含むファイルへのパスです。指定しない場合、デフォルトのエラーページが使用されます。
<b>IdentityProviders</b>	ユーザーが自身を確認する方法の順序付きの一覧です。
<b>Login</b>	ログインページのレンダリングに使用される Go テンプレートを含むファイルへのパスです。指定しない場合、デフォルトのログインページが使用されます。
<b>MasterCA</b>	TLS 接続が <b>MasterURL</b> に戻っていることを確認するための CA です。
<b>MasterPublicURL</b>	外部アクセス用の有効なクライアントのリダイレクト URL の作成に使用されます。
<b>MasterURL</b>	アクセストークンの認可コードを交換するためのサーバー間の呼び出しに使用されます。

パラメーター名	説明
<b>OAuthConfig</b>	<p>これがある場合、/oauth エンドポイントは定義されたパラメーターに基づいて開始します。以下に例を示します。</p> <pre> oauthConfig:   assetPublicURL: https://master.ose32.example.com:8443/console/   grantConfig:     method: auto   identityProviders:   - challenge: true     login: true     mappingMethod: claim     name: htpasswd_all     provider:       apiVersion: v1       kind: HTPasswdPasswordIdentityProvider       file: /etc/origin/openshift-passwd   masterCA: ca.crt   masterPublicURL: https://master.ose32.example.com:8443   masterURL: https://master.ose32.example.com:8443   sessionConfig:     sessionMaxAgeSeconds: 3600     sessionName: ssn     sessionSecretsFile: /etc/origin/master/session-secrets.yaml   tokenConfig:     accessTokenMaxAgeSeconds: 86400     authorizeTokenMaxAgeSeconds: 500 </pre>
<b>OAuthTemplates</b>	ログインページなどページのカスタマイズを許可します。
<b>ProviderSelection</b>	プロバイダーの選択ページのレンダリングに使用される Go テンプレートを含むファイルへのパスです。指定されていない場合、デフォルトのプロバイダー選択ページが使用されます。
<b>SessionConfig</b>	セッションの設定に関する情報を保持します。
<b>Templates</b>	ログインページなどのページのカスタマイズを許可します。
<b>TokenConfig</b>	認証およびアクセストークンのオプションが含まれます。

## 7.6.12. プロジェクトの設定

表7.12 プロジェクト設定パラメーター

パラメーター名	説明
<b>DefaultNodeSelector</b>	デフォルトのプロジェクトノードのラベルセレクターを保持します。

パラメーター名	説明
<b>ProjectConfig</b>	<p>プロジェクト作成とデフォルトに関する情報を保持します。</p> <ul style="list-style-type: none"> <li>● <b>DefaultNodeSelector</b> (文字列): デフォルトのプロジェクトノードのラベルセレクターを保持します。</li> <li>● <b>ProjectRequestMessage</b> (文字列): この文字列は、ユーザーが projectrequest API エンドポイントからプロジェクトを要求できない場合に提示されます。</li> <li>● <b>ProjectRequestTemplate</b> (文字列): projectrequest への応答としてプロジェクトを作成する際に使用されるテンプレートです。フォーマット <code>&lt;namespace&gt;/&lt;template&gt;</code> が使用されません。これはオプションであり、指定されていない場合はデフォルトのテンプレートが使用されます。</li> <li>● <b>SecurityAllocator</b>: UID と MCS ラベルのプロジェクトに対する自動割り当てを制御します。空の場合、割り当ては無効にされます。 <ul style="list-style-type: none"> <li>○ <b>mcsAllocatorRange</b> (文字列): namespace に割り当てられる MCS カテゴリーの範囲を定義します。フォーマットは <code>&lt;prefix&gt;/&lt;numberOfLabels&gt;[,&lt;maxCategory&gt;]</code> です。デフォルトは <b>s0/2</b> であり、c0 から c1023 に割り当てられます。つまり、これは合計 535000 のラベルが利用可能であることを意味します。この値を起動後に変更すると、新規プロジェクトは、すでに他のプロジェクトに割り当てられているラベルを受信することがあります。接頭辞には SELinux の有効な用語のセット (ユーザー、ロール、タイプなど) を使用できます。ただし、接頭辞をデフォルトとして残しておく、サーバーはそれらを自動的に設定できます。たとえば、<b>s0:/2</b> はラベルを s0:c0,c0 から s0:c511,c511 に割り当て、<b>s0:/2,512</b> はラベルを s0:c0,c0 から s0:c511,c511,511 に割り当てます。</li> <li>○ <b>mcsLabelsPerProject</b> (整数): プロジェクトごとに予約するラベルの数を定義します。デフォルトは、デフォルトの UID と MCS の範囲に一致する <b>5</b> です。</li> <li>○ <b>uidAllocatorRange</b> (文字列): プロジェクトに自動的に割り当てられる Unix ユーザー ID (UID) の合計セット数と、各 namespace が取得するブロックのサイズを定義します。たとえば、<b>1000-1999/10</b> は namespace ごとに 10 の UID を割り当て、空間を使い果たす前に最大 100 のブロックを割り当てることが可能です。デフォルトでは、1万のブロックに 10 億から 20 億を割り当てます。これは、ユーザーの namespace の起動時にコンテナイメージについて予想される範囲のサイズです。</li> </ul> </li> </ul>
<b>ProjectRequestMessage</b>	この文字列は、プロジェクトの要求 API エンドポイントからプロジェクトを要求できない場合にユーザーに提示されます。
<b>ProjectRequestTemplate</b>	projectrequest への応答としてプロジェクトを作成する際に使用されるテンプレートです。フォーマットは namespace/template です。これはオプションであり、指定されていない場合はデフォルトのテンプレートが使用されます。

## 7.6.13. スケジューラーの設定

表7.13 スケジューラー設定パラメーター

パラメーター名	説明
<b>SchedulerConfigFile</b>	スケジューラーのセットアップ方法を記述しているファイルをポイントします。空の場合、デフォルトのスケジューリングルールが取得されず。

## 7.6.14. セキュリティーアロケーターの設定

表7.14 セキュリティーアロケーターパラメーター

パラメーター名	説明
<b>MCSAllocatorRange</b>	namespace に割り当てられる MCS カテゴリーの範囲を定義します。フォーマットは <code>&lt;prefix&gt;/&lt;numberOfLabels&gt;[,&lt;maxCategory&gt;]</code> です。デフォルトは <code>s0/2</code> であり、 <code>c0</code> から <code>c1023</code> に割り当てられます。つまり、合計 535000 のラベルが利用可能であることを意味します (1024 は 2~535000 を選択します)。この値を起動後に変更すると、新規プロジェクトは、すでに他のプロジェクトに割り当てられているラベルを受信することがあります。接頭辞には、SELinux の有効な用語のセット (ユーザー、ロール、タイプなど) にすることができます。ただしこれらをデフォルトとして残しておく、サーバーはこれらを自動的に設定できます。
<b>SecurityAllocator</b>	UID と MCS ラベルのプロジェクトへの自動割り当てを制御します。空の場合、割り当ては無効にされます。
<b>UIDAllocatorRange</b>	プロジェクトに自動的に割り当てられる Unix ユーザー ID (UID) の合計セット数と、各 namespace が取得するブロックのサイズを定義します。たとえば、 <code>1000-1999/10</code> は namespace ごとに 10 の UID を割り当て、空間を使い果たす前に最大 100 のブロックを割り当てることが可能です。デフォルトでは、1万のブロックに 10 億から 20 億 (ユーザー namespace の起動時にコンテナイメージが使用する範囲の予想されるサイズ) を割り当てます。

## 7.6.15. サービスアカウントの設定

表7.15 サービスアカウント設定パラメーター

パラメーター名	説明
<b>LimitSecretReferences</b>	サービスアカウントに、明示的な参照なしに namespace のシークレットの参照を許可するかどうかを制御します。
<b>ManagedNames</b>	すべての namespace に自動作成されるサービスアカウント名の一覧です。名前が指定されていない場合、 <b>ServiceAccountsController</b> は起動しません。

パラメーター名	説明
<b>MasterCA</b>	TLS 接続がマスターに戻っていることを確認する CA です。サービスアカウントのコントローラーは、マスターへの接続を検証できるようにこのファイルの内容を Pod に自動的に挿入します。
<b>PrivateKeyFile</b>	PEM でエンコードされた RSA プライベートキーを含むファイルです。これはサービスアカウントのトークンの署名に使用されます。プライベートキーが指定されていない場合、サービスアカウント <b>TokensController</b> は起動しません。
<b>PublicKeyFiles</b>	PEM でエンコードされた RSA パブリックキーを含むファイルの一覧です。いずれかのファイルにプライベートキーが含まれている場合、そのキーのパブリックの部分が使用されます。パブリックキーの一覧は、表示されているサービスアカウントのトークンの確認に使用されます。それぞれのキーは、一覧がすべて使用されるまで、または確認が正常に実行されるまで順番に試行されます。キーが指定されていない場合、サービスアカウントの認証は使用できません。
<b>ServiceAccountConfig</b>	<p>サービスアカウントに関連するオプションを保持します。</p> <ul style="list-style-type: none"> <li>● <b>LimitSecretReferences</b> (ブール値): サービスアカウントに、明示的な参照なしに namespace のシークレットの参照を許可するかどうかを制御します。</li> <li>● <b>ManagedNames</b> (文字列): それぞれの namespace に自動作成されるサービスアカウント名の一覧です。名前が指定されていない場合、<b>ServiceAccountsController</b> は起動しません。</li> <li>● <b>MasterCA</b> (文字列): TLS 接続がマスターに戻っていることを確認する認証局です。サービスアカウントコントローラーは、マスターへの接続を検証できるようにこのファイルの内容を Pod に自動的に挿入します。</li> <li>● <b>PrivateKeyFile</b> (文字列): PEM でエンコードされた RSA プライベートキーが含まれ、サービスアカウントのトークンの署名に使用されます。プライベートキーが指定されていない場合、サービスアカウント <b>TokensController</b> は起動しません。</li> <li>● <b>PublicKeyFiles</b> (文字列): PEM でエンコードされた RSA パブリックキーを含むファイルの一覧です。いずれかのファイルにプライベートキーが含まれている場合、OpenShift Container Platform はキーのパブリックの部分を使用します。パブリックキーの一覧は、サービスアカウントのトークンの確認に使用されます。それぞれのキーは、一覧がすべて使用されるまで、または確認が正常に実行されるまで順番に試行されます。キーが指定されていない場合、サービスアカウントの認証は使用できません。</li> </ul>

## 7.6.16. 提供情報の設定

表7.16 提供情報設定パラメーター

パラメーター名	説明
<b>AllowRecursiveQueries</b>	マスター上の DNS サーバーがクエリーに再帰的に応答することを許可します。オープンリゾルバーは DNS アンプ攻撃に使用される可能であり、マスター DNS は公開ネットワークでアクセスできないようにする必要のあることに注意してください。
<b>BindAddress</b>	提供に使用される ip:port です。
<b>BindNetwork</b>	イメージをインポートするための制限と動作を制御します。
<b>CertFile</b>	PEM でエンコードされた証明書を含むファイルです。
<b>CertInfo</b>	セキュアなトラフィックを提供するための TLS 証明書情報です。
<b>ClientCA</b>	クライアント証明書が受信される際にユーザーが認識するすべての署名者の証明書バンドルです。
<b>dnsConfig</b>	これがある場合、DNS サーバーが定義されたパラメーターに基づいて起動します。以下に例を示します。 <pre>dnsConfig:   bindAddress: 0.0.0.0:8053   bindNetwork: tcp4</pre>
<b>DNSDomain</b>	ドメイン接尾辞を保持します。
<b>DNSIP</b>	IP を保持します。
<b>KeyFile</b>	<b>CertFile</b> が指定した証明書の PEM でエンコードされたプライベートキーを含むファイルです。
<b>MasterClientConnectionOverrides</b>	マスターへの接続に使用されたクライアント接続を上書きします。このパラメーターはサポート対象外です。QPS およびバースト値を設定するには、 <a href="#">ノード QPS およびバースト値の設定</a> を参照してください。
<b>MaxRequestsInFlight</b>	サーバーに許可されている同時要求数です。ゼロの場合は無制限です。
<b>NamedCertificates</b>	特定のホスト名への要求を保護するのに使用される証明書の一覧です。
<b>RequestTimeoutSecond</b>	要求がタイムアウトするまでの秒数です。デフォルトは 60 分です。-1 の場合、要求について無制限となります。
<b>ServingInfo</b>	アセット用の HTTP 提供情報です。

## 7.6.17. ボリュームの設定

表7.17 ボリューム設定パラメーター

パラメーター名	説明
<b>DynamicProvisioningEnabled</b>	動的なプロビジョニングを有効化または無効化するブール値です。デフォルトは <b>true</b> です。
<b>FSGroup</b>	各ノードでそれぞれの FSGroup について <b>local storage quotas</b> を有効にします。現時点では、これは emptyDir ボリュームについて、基礎となる <b>volumeDirectory</b> が XFS ファイルシステムにある場合にのみ実装されます。
<b>MasterVolumeConfig</b>	マスターノードのボリュームプラグインを設定するオプションが含まれます。
<b>NodeVolumeConfig</b>	ノードのボリュームを設定するオプションが含まれます。
<b>VolumeConfig</b>	ノードのボリュームプラグインを設定するオプションが含まれます。 <ul style="list-style-type: none"> <li>● <b>DynamicProvisioningEnabled</b> (ブール値): デフォルト値は <b>true</b> です。 <b>false</b> の場合は動的プロビジョニングはオフに切り替わります。</li> </ul>
<b>VolumeDirectory</b>	ボリュームがその下に保存されるディレクトリーです。この値を変更するには、 <b>openshift_node_group_data_dir</b> パラメーターを使用します。

### 7.6.18. 基本的な監査

監査は、システムに影響を与えた一連のアクティビティーを個別のユーザー、管理者その他システムのコンポーネント別に記述したセキュリティー関連の時系列のレコードを提供します。

監査は API サーバーレベルで実行され、サーバーに送られるすべての要求をログに記録します。それぞれの監査ログには以下の2つのエントリーが含まれます。

1. 以下を含む要求行。
  - a. 応答行 (以下の 2 を参照してください) と一致する固有 ID
  - b. 要求のソース IP
  - c. 呼び出されている HTTP メソッド
  - d. 操作を呼び出している元のユーザー
  - e. 操作を実行するための偽装ユーザー (**self** はユーザー自身を指します)
  - f. 操作を実行するための偽装グループ (**lookup** はユーザーのグループを指します)
  - g. 要求または <none> の namespace
  - h. 要求される URI
2. 以下を含む応答行。
  - a. 上記 1 の固有の ID

a. 上記のインストール

b. 応答コード

Pod の一覧を要求するユーザー `admin` の出力例。

```
AUDIT: id="5c3b8227-4af9-4322-8a71-542231c3887b" ip="127.0.0.1" method="GET" user="admin"
as="<self>" asgroups="<lookup>" namespace="default" uri="/api/v1/namespaces/default/pods"
AUDIT: id="5c3b8227-4af9-4322-8a71-542231c3887b" response="200"
```

### 7.6.18.1. 基本監査を有効にする

次の手順では、インストール後の基本的な監査を有効にします。



#### 注記

インストール中に高度な監査を有効にする必要があります。

1. 次の例に示すように、すべてのマスターノードで `/etc/origin/master/master-config.yaml` ファイルを編集します。

```
auditConfig:
  auditFilePath: "/var/log/origin/audit-ocp.log"
  enabled: true
  maximumFileRetentionDays: 14
  maximumFileSizeMegabytes: 500
  maximumRetainedFiles: 15
```

2. クラスター内の API Pod を再起動します。

```
# /usr/local/bin/master-restart api
```

インストール中に基本的な監査を有効にするには、次の変数宣言をインベントリーファイルに追加します。必要に応じて値を調整します。

```
openshift_master_audit_config={"enabled": true, "auditFilePath": "/var/lib/origin/openpaas-ocsp-audit.log", "maximumFileRetentionDays": 14, "maximumFileSizeMegabytes": 500, "maximumRetainedFiles": 5}
```

監査設定では以下のパラメーターを使用できます。

表7.18 監査設定パラメーター

パラメーター名	説明
<code>enabled</code>	監査ログを有効または無効にするブール値です。デフォルトは <b>false</b> です。
<code>auditFilePath</code>	要求をログに記録するファイルパスです。設定されていない場合、ログはマスターログに出力されます。

パラメーター名	説明
<b>maximumFileRetentionDays</b>	ファイル名にエンコードされるタイムスタンプに基づいて古い監査ログファイルを保持する最大日数を指定します。
<b>maximumRetainedFiles</b>	古い監査ログファイルを保持する最大数を指定します。
<b>maximumFileSizeMegabytes</b>	ログファイルがローテーションされる前に、ファイルの最大サイズをメガバイトで指定します。デフォルトは100 MBです。

## 監査の設定例

```
auditConfig:
  auditFilePath: "/var/log/origin/audit-ocp.log"
  enabled: true
  maximumFileRetentionDays: 14
  maximumFileSizeMegabytes: 500
  maximumRetainedFiles: 15
```

**auditFilePath** パラメーターを定義すると、ディレクトリが存在しない場合に作成されます。

### 7.6.19. 高度な監査

高度な監査機能は、詳細なイベントのフィルターリングや複数の出力バックエンドなど、[基本的な監査機能](#) に対するいくつかの改良機能を提供します。

高度な監査機能を有効にするには、監査ポリシーファイルを作成し、以下の値を **openshift\_master\_audit\_config** および **openshift\_master\_audit\_policyfile** パラメーターに指定します。

```
openshift_master_audit_config={"enabled": true, "auditFilePath": "/var/log/origin/audit-ocp.log",
"maximumFileRetentionDays": 14, "maximumFileSizeMegabytes": 500, "maximumRetainedFiles": 5,
"policyFile": "/etc/origin/master/adv-audit.yaml", "logFormat": "json"}
openshift_master_audit_policyfile="/<path>/adv-audit.yaml"
```



#### 重要

**adv-audit.yaml** ファイルを作成してからクラスターをインストールし、その場所をクラスターのインベントリーファイルに指定する必要があります。

以下の表には、使用できる追加のオプションが含まれています。

表7.19 高度な監査設定パラメーター

パラメーター名	説明
<b>policyFile</b>	監査ポリシー設定を定義するファイルへのパスです。
<b>policyConfiguration</b>	組み込まれる監査ポリシー設定です。

パラメーター名	説明
<b>logFormat</b>	保存される監査ログのフォーマットを指定します。許可されている値は <b>legacy</b> (基本的な監査に使用されるフォーマット) と <b>json</b> です。
<b>webHookKubeConfig</b>	監査の Webhook 設定を定義する <b>.kubeconfig</b> でフォーマットされたファイルへのパスです。ここにイベントが送信されます。
<b>webHookMode</b>	監査イベントを送信するためのストラテジーを指定します。許可される値は <b>block</b> (直前のイベント処理が完了するまで別のイベントの処理をブロックする) と <b>batch</b> (イベントをバッファ処理し、バッチで提供する) です。



### 重要

高度な監査機能を有効にするには、監査ポリシールールを記述する **policyFile** または **policyConfiguration** を指定する必要があります。

### 監査ポリシーの設定例

```

apiVersion: audit.k8s.io/v1beta1
kind: Policy
rules:

  # Do not log watch requests by the "system:kube-proxy" on endpoints or services
  - level: None ①
    users: ["system:kube-proxy"] ②
    verbs: ["watch"] ③
    resources: ④
      - group: ""
        resources: ["endpoints", "services"]

  # Do not log authenticated requests to certain non-resource URL paths.
  - level: None
    userGroups: ["system:authenticated"] ⑤
    nonResourceURLs: ⑥
      - "/api*" # Wildcard matching.
      - "/version"

  # Log the request body of configmap changes in kube-system.
  - level: Request
    resources:
      - group: "" # core API group
        resources: ["configmaps"]
    # This rule only applies to resources in the "kube-system" namespace.
    # The empty string "" can be used to select non-namespaced resources.
    namespaces: ["kube-system"] ⑦

  # Log configmap and secret changes in all other namespaces at the metadata level.
  - level: Metadata
    resources:

```

```
- group: "" # core API group
  resources: ["secrets", "configmaps"]
```

*# Log all other resources in core and extensions at the request level.*

```
- level: Request
```

```
resources:
```

```
- group: "" # core API group
```

```
- group: "extensions" # Version of group should NOT be included.
```

*# A catch-all rule to log all other requests at the Metadata level.*

```
- level: Metadata 8
```

*# Log login failures from the web console or CLI. Review the logs and refine your policies.*

```
- level: Metadata
```

```
nonResourceURLs:
```

```
- /login* 9
```

```
- /oauth* 10
```

**1 8** すべてのイベントは以下の4つのレベルでログに記録できます。

- **None** - このルールに一致するイベントは記録されません。
- **Metadata** - 要求のメタデータ (要求しているユーザー、タイムスタンプ、リソース、verb など) をログに記録します。要求または応答本体はログに記録しません。基本的な監査で使用されるレベルと同じレベルになります。
- **Request** - イベントのメタデータと要求本体をログに記録します。応答本体はログに記録しません。
- **RequestResponse** - イベントのメタデータ、要求、および応答本体をログに記録します。

**2** このルールが適用されるユーザーの一覧です。一覧が空の場合はすべてのユーザーに適用されます。

**3** このルールが適用される verb の一覧です。一覧が空の場合はすべての verb に適用されます。これは API 要求に関連付けられる Kubernetes の verb です (**get**、**list**、**watch**、**create**、**update**、**patch**、**delete**、**deletecollection**、**proxy** など)。

**4** このルールが適用されるリソースの一覧です。一覧が空の場合はすべてのリソースに適用されます。各リソースは、それが割り当てられるグループ (例: 空の場合は Kubernetes core API、バッチ、build.openshift.ioなどを指します)、およびそのグループのリソース一覧として指定されます。

**5** このルールが適用されるグループの一覧です。一覧が空の場合はすべてのグループに適用されます。

**6** このルールが適用されるリソース以外の URL の一覧です。

**7** このルールが適用される namespace の一覧です。一覧が空の場合はすべての namespace に適用されます。

**9** Web コンソールが使用するエンドポイントです。

**10** CLI が使用するエンドポイントです。

高度な監査についての詳細は、[Kubernetes のドキュメント](#) を参照してください。

## 7.6.20. etcd の TLS 暗号の指定

マスターと etcd サーバー間の通信で使用する [サポートされている TLS 暗号](#) を指定できます。

1. 各 etcd ノードで、etcd をアップグレードします。

```
# yum update etcd iptables-services
```

2. お使いのバージョンが 3.2.22 以降であることを確認します。

```
# etcd --version
etcd Version: 3.2.22
```

3. 各マスターホストで、`/etc/origin/master/master-config.yaml` ファイルで有効にする暗号を指定します。

```

servingInfo:
  ...
  minTLSVersion: VersionTLS12
  cipherSuites:
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_RSA_WITH_AES_256_CBC_SHA
  - TLS_RSA_WITH_AES_128_CBC_SHA
  ...

```

4. 各マスターホストで、マスターサービスを再起動します。

```
# master-restart api
# master-restart controllers
```

5. 暗号が適用されていることを確認します。たとえば、TLSv1.2 暗号 **ECDHE-RSA-AES128-GCM-SHA256** については、以下のコマンドを実行します。

```

# openssl s_client -connect etcd1.example.com:2379 ①
CONNECTED(00000003)
depth=0 CN = etcd1.example.com
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 CN = etcd1.example.com
verify error:num=21:unable to verify the first certificate
verify return:1
139905367488400:error:14094412:SSL routines:ssl3_read_bytes:ssl3 alert bad
certificate:s3_pkt.c:1493:SSL alert number 42
139905367488400:error:140790E5:SSL routines:ssl23_write:ssl handshake
failure:s23_lib.c:177:
---
Certificate chain
0 s:/CN=etcd1.example.com
i:/CN=etcd-signer@1529635004
---
Server certificate
-----BEGIN CERTIFICATE-----

```

```

MIIEkjCCAnqgAwIBAgIBATANBgkqhkiG9w0BAQsFADAhMR8wHQYDVQQDBZldGNk
.....
....
eif87qttt0SI1vS8DG1KQO1oOBINkg==
-----END CERTIFICATE-----
subject=/CN=etcd1.example.com
issuer=/CN=etcd-signer@1529635004
---
Acceptable client certificate CA names
/CN=etcd-signer@1529635004
Client Certificate Types: RSA sign, ECDSA sign
Requested Signature Algorithms:
RSA+SHA256:ECDSA+SHA256:RSA+SHA384:ECDSA+SHA384:RSA+SHA1:ECDSA+SHA1

Shared Requested Signature Algorithms:
RSA+SHA256:ECDSA+SHA256:RSA+SHA384:ECDSA+SHA384:RSA+SHA1:ECDSA+SHA1

Peer signing digest: SHA384
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 1666 bytes and written 138 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
  Protocol : TLSv1.2
  Cipher   : ECDHE-RSA-AES128-GCM-SHA256
  Session-ID:
  Session-ID-ctx:
  Master-Key:
1EFA00A91EE5FC5EDDCFC67C8ECD060D44FD3EB23D834EDED929E4B74536F273C0F
9299935E5504B562CD56E76ED208D
  Key-Arg  : None
  Krb5 Principal: None
  PSK identity: None
  PSK identity hint: None
  Start Time: 1529651744
  Timeout  : 300 (sec)
  Verify return code: 21 (unable to verify the first certificate)

```

**1** **etcd1.example.com** は etcd ホストの名前です。

## 7.7. ノード設定ファイル

インストール時に、OpenShift Container Platform はそれぞれのノードグループに対して **openshift-node** プロジェクトに **configmap** を作成します。

- node-config-master
- node-config-infra

- node-config-compute
- node-config-all-in-one
- node-config-master-infra

既存のノードに設定の変更を加えるには、該当する設定マップを編集します。各ノードの `sync pod` は設定マップで変更の有無を監視します。インストール時に、同期 Pod は `sync Daemonsets` を使用して作成され、ノード設定パラメーターが存在する `/etc/origin/node/node-config.yaml` ファイルが各ノードに追加されます。同期 Pod が設定マップの変更を検出すると、そのノードグループ内のすべてのノードで `node-config.yaml` を更新し、適切なノードで `atomic-openshift-node.service` を再起動します。

```
$ oc get cm -n openshift-node
```

## 出力例

NAME	DATA	AGE
node-config-all-in-one	1	1d
node-config-compute	1	1d
node-config-infra	1	1d
node-config-master	1	1d
node-config-master-infra	1	1d

## node-config-compute グループの設定マップの例

```
apiVersion: v1
authConfig: ①
  authenticationCacheSize: 1000
  authenticationCacheTTL: 5m
  authorizationCacheSize: 1000
  authorizationCacheTTL: 5m
dnsBindAddress: 127.0.0.1:53
dnsDomain: cluster.local
dnsIP: 0.0.0.0 ②
dnsNameservers: null
dnsRecursiveResolvConf: /etc/origin/node/resolv.conf
dockerConfig:
  dockerShimRootDirectory: /var/lib/dockershim
  dockerShimSocket: /var/run/dockershim.sock
  execHandlerName: native
enableUnidling: true
imageConfig:
  format: registry.reg-aws.openshift.com/openshift3/ose-${component}:${version}
  latest: false
iptablesSyncPeriod: 30s
kind: NodeConfig
kubeletArguments: ③
  bootstrap-kubeconfig:
  - /etc/origin/node/bootstrap.kubeconfig
  cert-dir:
  - /etc/origin/node/certificates
  cloud-config:
  - /etc/origin/cloudprovider/aws.conf
```

```

cloud-provider:
- aws
enable-controller-attach-detach:
- 'true'
feature-gates:
- RotateKubeletClientCertificate=true,RotateKubeletServerCertificate=true
node-labels:
- node-role.kubernetes.io/compute=true
pod-manifest-path:
- /etc/origin/node/pods ④
rotate-certificates:
- 'true'
masterClientConnectionOverrides:
acceptContentTypes: application/vnd.kubernetes.protobuf,application/json
burst: 40
contentType: application/vnd.kubernetes.protobuf
qps: 20
masterKubeConfig: node.kubeconfig
networkConfig: ⑤
  mtu: 8951
  networkPluginName: redhat/openshift-ovs-subnet ⑥
servingInfo: ⑦
  bindAddress: 0.0.0.0:10250
  bindNetwork: tcp4
  clientCA: client-ca.crt ⑧
volumeConfig:
  localQuota:
    perFSGroup: null
volumeDirectory: /var/lib/origin/openshift.local.volumes

```

- ① 認証および承認設定のオプション
- ② Pod の `/etc/resolv.conf` に追加 IP アドレスです。
- ③ Kubelet のコマンドライン引数 に一致する Kubelet に直接渡されるキー/値のペアです。
- ④ Pod マニフェストまたはディレクトリーへのパスです。ディレクトリーには、1つ以上のマニフェストファイルが含まれている必要があります。OpenShift Container Platform はマニフェストファイルを使用してノードに Pod を作成します。
- ⑤ ノード上の Pod ネットワーク設定です。
- ⑥ SDN (Software defined network) プラグインです。ovs-subnet プラグインは `redhat/openshift-ovs-subnet`、ovs-multitenant プラグインは `redhat/openshift-ovs-multitenant`、または ovs-networkpolicy プラグインは `redhat/openshift-ovs-networkpolicy` にそれぞれ設定します。
- ⑦ ノードの証明書情報です。
- ⑧ オプション: PEM でエンコードされた証明書バンドルです。これが設定されている場合、要求ヘッダーのユーザー名をチェックする前に、有効なクライアント証明書が提示され、指定ファイルで認証局に対して検証される必要があります。

**注記**

/etc/origin/node/node-config.yaml ファイルは手動で変更できません。

ノード設定ファイルはノードのリソースを決定します。詳細は、[クラスター管理ガイドのノードリソースの割り当て](#) セクションを参照してください。

### 7.7.1. Pod とノードの設定

表7.20 Pod とノードの設定パラメーター

パラメーター名	説明
<b>NodeConfig</b>	OpenShift Container Platform ノードを起動する完全に指定された設定です。
<b>NodeName</b>	クラスターの特定ノードを識別するために使用される値です。可能な場合、この値はユーザーの完全修飾ホスト名にできます。ユーザーが静的ノードのセットをマスターに記述している場合、この値は一覧にある値のいずれかに一致している必要があります。

### 7.7.2. Docker の設定

表7.21 Docker 設定パラメーター

パラメーター名	説明
<b>AllowDisabledDocker</b>	true の場合、Kubelet は Docker のエラーを無視します。これは、Docker を起動させていないマシンでノードを起動できることを意味します。
<b>DockerConfig</b>	Docker 関連の設定オプションを保持します。
<b>ExecHandlerName</b>	コンテナでのコマンドの実行に使用するハンドラーです。

### 7.7.3. ローカルストレージの設定

[XFS クォータサブシステム](#) を使用して **emptyDir** ボリューム、および各ノードのシークレットおよび設定マップなどの **emptyDir** ボリュームに基づくボリュームのサイズを制限できます。

XFS ファイルシステムで **emptyDir** ボリュームのサイズを制限するには、**openshift-node** プロジェクトで **node-config-compute** 設定マップを使用し、それぞれの一意の **FSGroup** についてローカルボリュームクォータを設定します。

```
apiVersion: kubelet.config.openshift.io/v1
kind: VolumeConfig
  localQuota: ①
  perFSGroup: 1Gi ②
```

- ① ノードのローカルボリュームのクォータを制御するオプションが含まれます。

- 2 この値を、**1Gi**、**512Mi** など [FSGroup] 別、ノード別に必要なクォータを示すリソース量に設定します。volumeDirectory は **grpquota** オプションを指定してマウントされた XFS ファイルシステム

要求が **RunAsAny** の SCC に一致することを示す FSGroup が指定されていない場合、クォータの適用は省略されます。



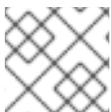
### 注記

/etc/origin/node/volume-config.yaml ファイルは直接編集しないでください。このファイルは node-config-compute 設定マップを基に作成されます。node-config-compute 設定マップを使用して volume-config.yaml ファイルでパラメーターの作成または編集を実行します。

## 7.7.4.1 秒あたりのノードクエリー数 (QPS) の制限およびバースト値の設定

kubelet が API サーバーと通信する速度は、qps およびバースト値によって異なります。各ノードで実行中の Pod に限りがある場合には、デフォルト値で十分です。ノードに CPU およびメモリーリソースが十分にある場合、qps および burst の値は /etc/origin/node/node-config.yaml ファイルで調整できます。

```
kubeletArguments:
  kube-api-qps:
    - "20"
  kube-api-burst:
    - "40"
```



### 注記

上記の qps およびバースト値は OpenShift Container Platform のデフォルトです。

表7.22 QPS およびバースト設定パラメーター

パラメーター名	説明
<b>kube-api-qps</b>	Kubelet が APIServer と通信する QPS レート。デフォルトは <b>20</b> です。
<b>kube-api-burst</b>	Kubelet が APIServer と通信するバーストレート。デフォルトは <b>40</b> です。
<b>ExecHandlerName</b>	コンテナでのコマンドの実行に使用するハンドラーです。

次に、[OpenShift Container Platform ノードサービスを再起動](#)します。

## 7.7.5. Docker 1.9 以降を使用したイメージの並行プル

Docker 1.9 以降を使用している場合は、イメージの並行プルを有効にしておくことができます。デフォルトでは、イメージは一度に1つずつプルされます。



## 注記

Docker 1.9 よりも前のバージョンでは、データの破損による問題が発生する可能性があります。1.9 以降では破損の問題は解消し、並行プルに安全に切り替えることができます。

```
kubeletArguments:
  serialize-image-pulls:
    - "false" ❶
```

❶ 並行プルを無効にするには **true** に変更します。これがデフォルト設定になります。

## 7.8. パスワードおよびその他の機密データ

[認証設定](#) によっては、LDAP **bindPassword** または OAuth **clientSecret** の値が必須になる場合があります。これらの値は、マスター設定ファイルに直接指定する代わりに、環境変数、外部ファイルまたは暗号化ファイルとして指定することができます。

### 環境変数の例

```
bindPassword:
  env: BIND_PASSWORD_ENV_VAR_NAME
```

### 外部ファイルの例

```
bindPassword:
  file: bindPassword.txt
```

### 暗号化された外部ファイルの例

```
bindPassword:
  file: bindPassword.encrypted
  keyFile: bindPassword.key
```

上記の例の暗号化ファイルとキーファイルを作成するには、以下を入力します。

```
$ oc adm ca encrypt --genkey=bindPassword.key --out=bindPassword.encrypted
> Data to encrypt: B1ndPass0rd!
```

Ansible ホストインベントリーファイル (デフォルトで `/etc/ansible/hosts`) に最初に一覧表示されているマスターから **oc adm** コマンドを実行します。



## 警告

暗号化データのセキュリティレベルは復号化キーと同程度です。ファイルシステムのパーミッションとキーファイルへのアクセスを制限する際には十分な注意が必要です。

## 7.9. 新規設定ファイルの作成

OpenShift Container Platform 設定をゼロから定義するときは、新規設定ファイルを作成することから開始します。

マスターホストの設定ファイルでは、**openshift start** コマンドを **--write-config** オプションと共に使用して設定ファイルを作成します。ノードホストの場合は、**oc adm create-node-config** コマンドを使用して設定ファイルを作成します。

以下のコマンドは、関連する起動設定ファイル、証明書ファイルその他ファイルを指定された **--write-config** または **--node-dir** ディレクトリーに書き込みます。

生成される証明書ファイルは2年間有効です。一方、認証局 (CA) の証明書は5年間有効です。この値は **--expire-days** と **--signer-expire-days** のオプションを使用して変更できますが、セキュリティ上の理由によりこの値をこれ以上大きい値に変更しないことが推奨されます。

オールインワンサーバー (マスターとノードが同一ホスト上にある) の設定ファイルを指定されたディレクトリーに作成するには、以下を入力します。

```
$ openshift start --write-config=/openshift.local.config
```

**マスター設定ファイル** とその他の必須ファイルを指定されたディレクトリーに作成するには、以下を実行します。

```
$ openshift start master --write-config=/openshift.local.config/master
```

**ノード設定ファイル** とその他の関連ファイルを指定されたディレクトリーに作成するには、以下を実行します。

```
$ oc adm create-node-config \
  --node-dir=/openshift.local.config/node-<node_hostname> \
  --node=<node_hostname> \
  --hostnames=<node_hostname>,<ip_address> \
  --certificate-authority="/path/to/ca.crt" \
  --signer-cert="/path/to/ca.crt" \
  --signer-key="/path/to/ca.key" \
  --signer-serial="/path/to/ca.serial.txt" \
  --node-client-certificate-authority="/path/to/ca.crt"
```

ノード設定ファイルを作成する際に、**--hostnames** オプションは、サーバー証明書を有効にする必要のあるすべてのホスト名または IP アドレスのコンマ区切りの一覧を受け入れます。

## 7.10. 設定ファイルの使用によるサーバーの起動

マスターまたはノード設定ファイルをユーザー仕様に変更すると、これを引数として指定してサーバーを起動すると、使用できるようになります。設定ファイルを指定する場合は、ユーザーが渡す他のコマンドラインオプションはいずれも認識されません。



### 注記

クラスターのノードを変更するには、**ノード設定マップ** を必要に応じて更新します。**node-config.yaml** ファイルは手動で変更しないようにしてください。

1. マスター設定ファイルを使用してマスターサーバーを起動します。

```
$ openshift start master \
  --config=/openshift.local.config/master/master-config.yaml
```

2. ノード設定ファイルおよび **node.kubeconfig** ファイルを使用して、ネットワークプロキシーおよび SDN プラグインを起動します。

```
$ openshift start network \
  --config=/openshift.local.config/node-<node_hostname>/node-config.yaml \
  --kubeconfig=/openshift.local.config/node-<node_hostname>/node.kubeconfig
```

3. ノード設定ファイルを使用してノードサーバーを起動します。

```
$ hyperkube kubelet \
  $(/usr/bin/openshift-node-config \
  --config=/openshift.local.config/node-<node_hostname>/node-config.yaml)
```

## 7.11. マスターおよびノードログの表示

OpenShift Container Platform は、ノードの **systemd-journald.service** およびスクリプトを使用して、デバッグ用にログメッセージを収集します (マスターの場合は **master-logs** を使用)。



### 注記

Web コンソールに表示される行数は 5000 にハードコーディングされ、これを変更することはできません。ログ全体を表示するには、CLI を使用します。

ロギングでは、以下のような Kubernetes のロギング規則に基づいて 5 段階のログメッセージの重要度を使用します。

表7.23 ログレベルのオプション

オプション	説明
0	エラーと警告のみ
2	通常の情報
4	デバッグレベルの情報
6	API レベルのデバッグ情報 (要求 / 応答)
8	本体レベルの API のデバッグ情報

ユーザーは、必要に応じて **マスターまたはノードのログレベル**をそれぞれ別個に変更できます。

### ノードログの表示

ノードシステムのログを表示するには、以下のコマンドを実行します。

```
# journalctl -r -u <journal_name>
```

最新のエントリーから表示するには、**-r** オプションを使用します。

### マスターログの表示

マスターコンポーネントのログを表示するには、以下のコマンドを実行します。

```
# /usr/local/bin/master-logs <component> <container>
```

以下に例を示します。

```
# /usr/local/bin/master-logs controllers controllers
# /usr/local/bin/master-logs api api
# /usr/local/bin/master-logs etcd etcd
```

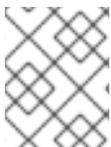
### マスターログのファイルへのリダイレクト

マスターログの出力をファイルにリダイレクトするには、以下のコマンドを実行します。

```
master-logs api api 2> file
```

## 7.11.1. ロギングレベルの設定

**DEBUG\_LOGLEVEL** オプションをマスターの `/etc/origin/master/master.env` ファイルに設定するか、ノードの `/etc/sysconfig/atomic-openshift-node` ファイルに設定して、ログに記録する INFO メッセージを制御できます。すべてのメッセージを収集するようにログを設定すると、解釈が困難な膨大なログが生成され、多くの領域が占領されます。すべてのメッセージの収集は、デバッグで使用する場合にとどめる必要があります。



### 注記

FATAL、ERROR、WARNING、および一部の INFO の重要度を伴うメッセージは、ログの設定に関係なくログに表示されます。

ロギングレベルを変更するには、以下を実行します。

1. マスターの `/etc/origin/master/master.env` ファイル、またはノードの `/etc/sysconfig/atomic-openshift-node` ファイルを編集します。
2. **Log Level Options** 表の値を **DEBUG\_LOGLEVEL** フィールドに入力します。以下に例を示します。

```
DEBUG_LOGLEVEL=4
```

3. マスターまたはノードを再起動します。[OpenShift Container Platform サービスの再起動](#) を参照してください。

再起動後は、すべての新しいログメッセージは新しい設定に従ったメッセージに従います。古いメッセージは変更されません。



### 注記

デフォルトのログレベルは標準のクラスターインストールプロセスを使用して設定できます。詳細は、[クラスター変数](#) を参照してください。

以下の例は、各種のログレベルのリダイレクトされたマスターログファイルの抜粋です。システム情報はこれらの例から削除されています。

### master-logs api api 2> file output at loglevel=2 の抜粋

```

W1022 15:08:09.787705      1 server.go:79] Unable to keep dnsmasq up to date, 0.0.0.0:8053 must
point to port 53
I1022 15:08:09.787894      1 logs.go:49] skydns: ready for queries on cluster.local. for
tcp4://0.0.0.0:8053 [rcode 0]
I1022 15:08:09.787913      1 logs.go:49] skydns: ready for queries on cluster.local. for
udp4://0.0.0.0:8053 [rcode 0]
I1022 15:08:09.889022      1 dns_server.go:63] DNS listening at 0.0.0.0:8053
I1022 15:08:09.893156      1 feature_gate.go:190] feature gates: map[AdvancedAuditing:true]
I1022 15:08:09.893500      1 master.go:431] Starting OAuth2 API at /oauth
I1022 15:08:09.914759      1 master.go:431] Starting OAuth2 API at /oauth
I1022 15:08:09.942349      1 master.go:431] Starting OAuth2 API at /oauth
W1022 15:08:09.977088      1 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
W1022 15:08:09.977176      1 swagger.go:38] No API exists for predefined swagger description
/api/v1
[restful] 2018/10/22 15:08:09 log.go:33: [restful/swagger] listing is available at
https://openshift.com:443/swaggerapi
[restful] 2018/10/22 15:08:09 log.go:33: [restful/swagger] https://openshift.com:443/swaggerui/ is
mapped to folder /swagger-ui/
I1022 15:08:10.231405      1 master.go:431] Starting OAuth2 API at /oauth
W1022 15:08:10.259523      1 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
W1022 15:08:10.259555      1 swagger.go:38] No API exists for predefined swagger description
/api/v1
I1022 15:08:23.895493      1 logs.go:49] http: TLS handshake error from 10.10.94.10:46322: EOF
I1022 15:08:24.449577      1 crdregistration_controller.go:110] Starting crd-autoregister controller
I1022 15:08:24.449916      1 controller_utils.go:1019] Waiting for caches to sync for crd-autoregister
controller
I1022 15:08:24.496147      1 logs.go:49] http: TLS handshake error from 127.0.0.1:39140: EOF
I1022 15:08:24.821198      1 cache.go:39] Caches are synced for APIServiceRegistrationController
controller
I1022 15:08:24.833022      1 cache.go:39] Caches are synced for AvailableConditionController
controller
I1022 15:08:24.865087      1 controller.go:537] quota admission added evaluator for: { events}
I1022 15:08:24.865393      1 logs.go:49] http: TLS handshake error from 127.0.0.1:39162: read tcp4
127.0.0.1:443->127.0.0.1:39162: read: connection reset by peer
I1022 15:08:24.966917      1 controller_utils.go:1026] Caches are synced for crd-autoregister
controller
I1022 15:08:24.967961      1 autoregister_controller.go:136] Starting autoregister controller
I1022 15:08:24.967977      1 cache.go:32] Waiting for caches to sync for autoregister controller
I1022 15:08:25.015924      1 controller.go:537] quota admission added evaluator for: {
serviceaccounts}
I1022 15:08:25.077984      1 cache.go:39] Caches are synced for autoregister controller
W1022 15:08:25.304265      1 lease_endpoint_reconciler.go:176] Resetting endpoints for master
service "kubernetes" to [10.10.94.10]
E1022 15:08:25.472536      1 memcache.go:153] couldn't get resource list for
servicecatalog.k8s.io/v1beta1: the server could not find the requested resource
E1022 15:08:25.550888      1 memcache.go:153] couldn't get resource list for
servicecatalog.k8s.io/v1beta1: the server could not find the requested resource
I1022 15:08:29.480691      1 healthz.go:72] /healthz/log check
I1022 15:08:30.981999      1 controller.go:105] OpenAPI AggregationController: Processing item

```

```

v1beta1.servicecatalog.k8s.io
E1022 15:08:30.990914    1 controller.go:111] loading OpenAPI spec for
"v1beta1.servicecatalog.k8s.io" failed with: OpenAPI spec does not exists
I1022 15:08:30.990965    1 controller.go:119] OpenAPI AggregationController: action for item
v1beta1.servicecatalog.k8s.io: Rate Limited Requeue.
I1022 15:08:31.530473    1 trace.go:76] Trace[1253590531]: "Get /api/v1/namespaces/openshift-
infra/serviceaccounts/serviceaccount-controller" (started: 2018-10-22 15:08:30.868387562 +0000
UTC m=+24.277041043) (total time: 661.981642ms):
Trace[1253590531]: [661.903178ms] [661.89217ms] About to write a response
I1022 15:08:31.531366    1 trace.go:76] Trace[83808472]: "Get /api/v1/namespaces/aws-
sb/secrets/aws-servicebroker" (started: 2018-10-22 15:08:30.831296749 +0000 UTC
m=+24.239950203) (total time: 700.049245ms):

```

### master-logs api api 2> file output at loglevel=4 の抜粋

```

I1022 15:08:09.746980    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: AlwaysDeny.
I1022 15:08:09.747597    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: ResourceQuota.
I1022 15:08:09.748038    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: openshift.io/ClusterResourceQuota.
I1022 15:08:09.786771    1 start_master.go:458] Starting master on 0.0.0.0:443 (v3.10.45)
I1022 15:08:09.786798    1 start_master.go:459] Public master address is https://openshift.com:443
I1022 15:08:09.786844    1 start_master.go:463] Using images from
"registry.access.redhat.com/openshift3/ose-<component>:v3.10.45"
W1022 15:08:09.787046    1 dns_server.go:37] Binding DNS on port 8053 instead of 53, which may
not be resolvable from all clients
W1022 15:08:09.787705    1 server.go:79] Unable to keep dnsmasq up to date, 0.0.0.0:8053 must
point to port 53
I1022 15:08:09.787894    1 logs.go:49] skydns: ready for queries on cluster.local. for
tcp4://0.0.0.0:8053 [rcache 0]
I1022 15:08:09.787913    1 logs.go:49] skydns: ready for queries on cluster.local. for
udp4://0.0.0.0:8053 [rcache 0]
I1022 15:08:09.889022    1 dns_server.go:63] DNS listening at 0.0.0.0:8053
I1022 15:08:09.893156    1 feature_gate.go:190] feature gates: map[AdvancedAuditing:true]
I1022 15:08:09.893500    1 master.go:431] Starting OAuth2 API at /oauth
I1022 15:08:09.914759    1 master.go:431] Starting OAuth2 API at /oauth
I1022 15:08:09.942349    1 master.go:431] Starting OAuth2 API at /oauth
W1022 15:08:09.977088    1 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
W1022 15:08:09.977176    1 swagger.go:38] No API exists for predefined swagger description
/api/v1
[restful] 2018/10/22 15:08:09 log.go:33: [restful/swagger] listing is available at
https://openshift.com:443/swaggerapi
[restful] 2018/10/22 15:08:09 log.go:33: [restful/swagger] https://openshift.com:443/swaggerui/ is
mapped to folder /swagger-ui/
I1022 15:08:10.231405    1 master.go:431] Starting OAuth2 API at /oauth
W1022 15:08:10.259523    1 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
W1022 15:08:10.259555    1 swagger.go:38] No API exists for predefined swagger description
/api/v1
[restful] 2018/10/22 15:08:10 log.go:33: [restful/swagger] listing is available at
https://openshift.com:443/swaggerapi
[restful] 2018/10/22 15:08:10 log.go:33: [restful/swagger] https://openshift.com:443/swaggerui/ is
mapped to folder /swagger-ui/

```

```

I1022 15:08:10.444303    1 master.go:431] Starting OAuth2 API at /oauth
W1022 15:08:10.492409    1 swagger.go:38] No API exists for predefined swagger description
/api/v1
W1022 15:08:10.492507    1 swagger.go:38] No API exists for predefined swagger description
/api/v1
[restful] 2018/10/22 15:08:10 log.go:33: [restful/swagger] listing is available at
https://openshift.com:443/swaggerapi
[restful] 2018/10/22 15:08:10 log.go:33: [restful/swagger] https://openshift.com:443/swaggerui/ is
mapped to folder /swagger-ui/
I1022 15:08:10.774824    1 master.go:431] Starting OAuth2 API at /oauth
I1022 15:08:23.808685    1 logs.go:49] http: TLS handshake error from 10.128.0.11:39206: EOF
I1022 15:08:23.815311    1 logs.go:49] http: TLS handshake error from 10.128.0.14:53054: EOF
I1022 15:08:23.822286    1 customresource_discovery_controller.go:174] Starting
DiscoveryController
I1022 15:08:23.822349    1 naming_controller.go:276] Starting NamingConditionController
I1022 15:08:23.822705    1 logs.go:49] http: TLS handshake error from 10.128.0.14:53056: EOF
+24.277041043) (total time: 661.981642ms):
Trace[1253590531]: [661.903178ms] [661.89217ms] About to write a response
I1022 15:08:31.531366    1 trace.go:76] Trace[83808472]: "Get /api/v1/namespaces/aws-
sb/secrets/aws-servicebroker" (started: 2018-10-22 15:08:30.831296749 +0000 UTC
m=+24.239950203) (total time: 700.049245ms):
Trace[83808472]: [700.049245ms] [700.04027ms] END
I1022 15:08:31.531695    1 trace.go:76] Trace[1916801734]: "Get /api/v1/namespaces/aws-
sb/secrets/aws-servicebroker" (started: 2018-10-22 15:08:31.031163449 +0000 UTC
m=+24.439816907) (total time: 500.514208ms):
Trace[1916801734]: [500.514208ms] [500.505008ms] END
I1022 15:08:44.675371    1 healthz.go:72] /healthz/log check
I1022 15:08:46.589759    1 controller.go:537] quota admission added evaluator for: { endpoints}
I1022 15:08:46.621270    1 controller.go:537] quota admission added evaluator for: { endpoints}
I1022 15:08:57.159494    1 healthz.go:72] /healthz/log check
I1022 15:09:07.161315    1 healthz.go:72] /healthz/log check
I1022 15:09:16.297982    1 trace.go:76] Trace[2001108522]: "GuaranteedUpdate etcd3:
*core.Node" (started: 2018-10-22 15:09:15.139820419 +0000 UTC m=+68.548473981) (total time:
1.158128974s):
Trace[2001108522]: [1.158012755s] [1.156496534s] Transaction committed
I1022 15:09:16.298165    1 trace.go:76] Trace[1124283912]: "Patch /api/v1/nodes/master-
0.com/status" (started: 2018-10-22 15:09:15.139695483 +0000 UTC m=+68.548348970) (total time:
1.158434318s):
Trace[1124283912]: [1.158328853s] [1.15713683s] Object stored in database
I1022 15:09:16.298761    1 trace.go:76] Trace[24963576]: "GuaranteedUpdate etcd3: *core.Node"
(started: 2018-10-22 15:09:15.13159057 +0000 UTC m=+68.540244112) (total time: 1.167151224s):
Trace[24963576]: [1.167106144s] [1.165570379s] Transaction committed
I1022 15:09:16.298882    1 trace.go:76] Trace[222129183]: "Patch /api/v1/nodes/node-
0.com/status" (started: 2018-10-22 15:09:15.131269234 +0000 UTC m=+68.539922722) (total time:
1.167595526s):
Trace[222129183]: [1.167517296s] [1.166135605s] Object stored in database

```

### master-logs api api 2> file output at loglevel=8 の抜粋

```

1022 15:11:58.829357    1 plugins.go:84] Registered admission plugin "NamespaceLifecycle"
I1022 15:11:58.839967    1 plugins.go:84] Registered admission plugin "Initializers"
I1022 15:11:58.839994    1 plugins.go:84] Registered admission plugin
"ValidatingAdmissionWebhook"
I1022 15:11:58.840012    1 plugins.go:84] Registered admission plugin
"MutatingAdmissionWebhook"

```

```
I1022 15:11:58.840025    1 plugins.go:84] Registered admission plugin "AlwaysAdmit"
I1022 15:11:58.840082    1 plugins.go:84] Registered admission plugin "AlwaysPullImages"
I1022 15:11:58.840105    1 plugins.go:84] Registered admission plugin
"LimitPodHardAntiAffinityTopology"
I1022 15:11:58.840126    1 plugins.go:84] Registered admission plugin "DefaultTolerationSeconds"
I1022 15:11:58.840146    1 plugins.go:84] Registered admission plugin "AlwaysDeny"
I1022 15:11:58.840176    1 plugins.go:84] Registered admission plugin "EventRateLimit"
I1022 15:11:59.850825    1 feature_gate.go:190] feature gates: map[AdvancedAuditing:true]
I1022 15:11:59.859108    1 register.go:154] Admission plugin AlwaysAdmit is not enabled. It will not
be started.
I1022 15:11:59.859284    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: AlwaysAdmit.
I1022 15:11:59.859809    1 register.go:154] Admission plugin NamespaceAutoProvision is not
enabled. It will not be started.
I1022 15:11:59.859939    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: NamespaceAutoProvision.
I1022 15:11:59.860594    1 register.go:154] Admission plugin NamespaceExists is not enabled. It
will not be started.
I1022 15:11:59.860778    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: NamespaceExists.
I1022 15:11:59.863999    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: NamespaceLifecycle.
I1022 15:11:59.864626    1 register.go:154] Admission plugin EventRateLimit is not enabled. It will
not be started.
I1022 15:11:59.864768    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: EventRateLimit.
I1022 15:11:59.865259    1 register.go:154] Admission plugin ProjectRequestLimit is not enabled. It
will not be started.
I1022 15:11:59.865376    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: ProjectRequestLimit.
I1022 15:11:59.866126    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: OriginNamespaceLifecycle.
I1022 15:11:59.866709    1 register.go:154] Admission plugin openshift.io/RestrictSubjectBindings
is not enabled. It will not be started.
I1022 15:11:59.866761    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: openshift.io/RestrictSubjectBindings.
I1022 15:11:59.867304    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: openshift.io/JenkinsBootstrapper.
I1022 15:11:59.867823    1 plugins.go:149] Loaded 1 admission controller(s) successfully in the
following order: openshift.io/BuildConfigSecretInjector.
I1022 15:12:00.015273    1 master_config.go:476] Initializing cache sizes based on 0MB limit
I1022 15:12:00.015896    1 master_config.go:539] Using the lease endpoint reconciler with
TTL=15s and interval=10s
I1022 15:12:00.018396    1 storage_factory.go:285] storing { apiServerIPInfo} in v1, reading as
__internal from storagebackend.Config{Type:"etcd3", Prefix:"kubernetes.io", ServerList:
[]string{"https://master-0.com:2379"}, KeyFile:"/etc/origin/master/master.etcd-client.key",
CertFile:"/etc/origin/master/master.etcd-client.crt", CAFile:"/etc/origin/master/master.etcd-ca.crt",
Quorum:true, Paging:true, DeserializationCacheSize:0, Codec:runtime.Codec(nil),
Transformer:value.Transformer(nil), CompactionInterval:300000000000,
CountMetricPollPeriod:60000000000}
I1022 15:12:00.037710    1 storage_factory.go:285] storing { endpoints} in v1, reading as __internal
from storagebackend.Config{Type:"etcd3", Prefix:"kubernetes.io", ServerList:[]string{"https://master-
0.com:2379"}, KeyFile:"/etc/origin/master/master.etcd-client.key",
CertFile:"/etc/origin/master/master.etcd-client.crt", CAFile:"/etc/origin/master/master.etcd-ca.crt",
Quorum:true, Paging:true, DeserializationCacheSize:0, Codec:runtime.Codec(nil),
Transformer:value.Transformer(nil), CompactionInterval:300000000000,
```

```

CountMetricPollPeriod:60000000000}
I1022 15:12:00.054112    1 compact.go:54] compactor already exists for endpoints [https://master-
0.com:2379]
I1022 15:12:00.054678    1 start_master.go:458] Starting master on 0.0.0.0:443 (v3.10.45)
I1022 15:12:00.054755    1 start_master.go:459] Public master address is https://openshift.com:443
I1022 15:12:00.054837    1 start_master.go:463] Using images from
"registry.access.redhat.com/openshift3/ose-<component>:v3.10.45"
W1022 15:12:00.056957    1 dns_server.go:37] Binding DNS on port 8053 instead of 53, which may
not be resolvable from all clients
W1022 15:12:00.065497    1 server.go:79] Unable to keep dnsmasq up to date, 0.0.0.0:8053 must
point to port 53
I1022 15:12:00.066061    1 logs.go:49] skydns: ready for queries on cluster.local. for
tcp4://0.0.0.0:8053 [rcache 0]
I1022 15:12:00.066265    1 logs.go:49] skydns: ready for queries on cluster.local. for
udp4://0.0.0.0:8053 [rcache 0]
I1022 15:12:00.158725    1 dns_server.go:63] DNS listening at 0.0.0.0:8053
I1022 15:12:00.167910    1 htpasswd.go:118] Loading htpasswd file /etc/origin/master/htpasswd...
I1022 15:12:00.168182    1 htpasswd.go:118] Loading htpasswd file /etc/origin/master/htpasswd...
I1022 15:12:00.231233    1 storage_factory.go:285] storing {apps.openshift.io deploymentconfigs}
in apps.openshift.io/v1, reading as apps.openshift.io/__internal from
storagebackend.Config{Type:"etcd3", Prefix:"openshift.io", ServerList:[]string{"https://master-
0.com:2379"}, KeyFile:"/etc/origin/master/master.etcd-client.key",
CertFile:"/etc/origin/master/master.etcd-client.crt", CAFile:"/etc/origin/master/master.etcd-ca.crt",
Quorum:true, Paging:true, DeserializationCacheSize:0, Codec:runtime.Codec(nil),
Transformer:value.Transformer(nil), CompactionInterval:300000000000,
CountMetricPollPeriod:60000000000}
I1022 15:12:00.248136    1 compact.go:54] compactor already exists for endpoints [https://master-
0.com:2379]
I1022 15:12:00.248697    1 store.go:1391] Monitoring deploymentconfigs.apps.openshift.io count at
<storage-prefix>//deploymentconfigs
W1022 15:12:00.256861    1 swagger.go:38] No API exists for predefined swagger description
/oapi/v1
W1022 15:12:00.258106    1 swagger.go:38] No API exists for predefined swagger description
/api/v1

```

## 7.12. マスターおよびノードサービスの再起動

マスターまたはノード設定の変更を適用するには、それぞれのサービスを再起動する必要があります。

マスター設定の変更を再度読み込むには、**master-restart** コマンドを使用してコントロールプレーンの静的 Pod で実行されているマスターサービスを再起動します。

```

# master-restart api
# master-restart controllers

```

ノード設定の変更を再度読み込むには、ノードホストでノードサービスを再起動します。

```

# systemctl restart atomic-openshift-node

```

## 第8章 OPENSIFT ANSIBLE BROKER の設定

### 8.1. 概要

OpenShift Ansible Broker (OAB) をクラスターにデプロイする際に、その動作の大半は、起動時に読み込まれるブローカーの設定ファイルによって決定されます。ブローカーの設定は、ブローカーの namespace (デフォルトでは (openshift-ansible-service-broker) に ConfigMap オブジェクトとして格納されます。

#### OpenShift Ansible Broker 設定ファイルの例

```
registry: ❶
- type: dockerhub
  name: docker
  url: https://registry.hub.docker.com
  org: <dockerhub_org>
  fail_on_error: false
- type: rhcc
  name: rhcc
  url: https://registry.redhat.io
  fail_on_error: true
  white_list:
    - "^foo.*-apb$"
    - ".*-apb$"
  black_list:
    - "bar.*-apb$"
    - "^my-apb$"
- type: local_openshift
  name: lo
  namespaces:
    - openshift
  white_list:
    - ".*-apb$"
dao: ❷
  etcd_host: localhost
  etcd_port: 2379
log: ❸
  logfile: /var/log/ansible-service-broker/asb.log
  stdout: true
  level: debug
  color: true
openshift: ❹
  host: ""
  ca_file: ""
  bearer_token_file: ""
  image_pull_policy: IfNotPresent
  sandbox_role: "edit"
  keep_namespace: false
  keep_namespace_on_error: true
broker: ❺
  bootstrap_on_startup: true
  dev_broker: true
  launch_apb_on_bind: false
  recovery: true
```

```

output_request: true
ssl_cert_key: /path/to/key
ssl_cert: /path/to/cert
refresh_interval: "600s"
auth:
  - type: basic
    enabled: true
secrets: 6
  - title: Database credentials
    secret: db_creds
    apb_name: dh-rhscl-postgresql-apb

```

- 1 詳細は [レジストリー設定](#) を参照してください。
- 2 詳細は [DAO 設定](#) を参照してください。
- 3 詳細は [ログ設定](#) を参照してください。
- 4 詳細は [OpenShift 設定](#) を参照してください。
- 5 詳細は [ブローカー設定](#) を参照してください。
- 6 詳細は [シークレット設定](#) を参照してください。

## 8.2. RED HAT PARTNER CONNECT レジストリーでの認証

Automation Broker を設定する前に、Red Hat Partner Connect を使用するように OpenShift Container Platform クラスターの全ノードで以下のコマンドを実行します。

```

$ docker --config=/var/lib/origin/.docker login -u <registry-user> -p <registry-password>
registry.connect.redhat.com

```

## 8.3. OPENSIFT ANSIBLE BROKER 設定の変更

OAB のデフォルト設定をデプロイした後に変更するには、以下を実行します。

1. OAB の namespace の **broker-config** ConfigMap オブジェクトを、**cluster-admin** 権限を持つユーザーとして編集します。

```

$ oc edit configmap broker-config -n openshift-ansible-service-broker

```

2. 更新内容を保存した後、変更を有効にするために OAB のデプロイメント設定を再デプロイします。

```

$ oc rollout latest dc/asb -n openshift-ansible-service-broker

```

## 8.4. レジストリー設定

**registry** セクションでは、ブローカーが APB 用に参照する必要があるレジストリーを定義できます。

表8.1 registry セクションの設定オプション

フィールド	説明	必須
<b>name</b>	レジストリーの名前です。このレジストリーから APB を識別するためにブローカーによって使用されます。	Y
<b>user</b>	レジストリーに対して認証するためのユーザー名です。 <b>auth_type</b> が <b>secret</b> または <b>file</b> に設定されている場合は使用されません。	N
<b>pass</b>	レジストリーに対して認証するためのパスワードです。 <b>auth_type</b> が <b>secret</b> または <b>file</b> に設定されている場合は使用されません。	N
<b>auth_type</b>	レジストリー認証情報が <b>user</b> と <b>pass</b> でブローカー設定に定義されていない場合にブローカーがレジストリー認証情報を読み取る方法です。 <b>secret</b> (シークレットをブローカー namespace で使用する) または <b>file</b> (マウントされたファイルを使用する) を指定できます。	N
<b>auth_name</b>	読み取る必要があるレジストリー認証情報を格納しているシークレットまたはファイルの名前です。 <b>auth_type</b> が <b>secret</b> に設定されている場合に使用されます。	N ( <b>auth_type</b> が <b>secret</b> または <b>file</b> に設定されている場合にのみ必要)
<b>org</b>	イメージが含まれている namespace または組織です。	N
<b>type</b>	レジストリーのタイプです。使用可能なアダプターは <b>mock</b> 、 <b>rhcc</b> 、 <b>openshift</b> 、 <b>dockerhub</b> 、および <b>local_openshift</b> です。	Y
<b>namespace</b>	<b>local_openshift</b> レジストリータイプの設定に使用する namespace の一覧です。デフォルトでは、ユーザーは <b>openshift</b> を使用する必要があります。	N
<b>url</b>	イメージ情報を取得するために使用される URL です。これは、RHCC の場合に広範囲に使用されます。 <b>dockerhub</b> タイプでは、ハードコードされた URL が使用されます。	N
<b>fail_on_error</b>	このレジストリーが失敗した場合にブートストラップ要求を失敗させるかどうかを指定します。失敗させる場合、その他のレジストリーの読み込みの実行を停止します。	N
<b>white_list</b>	許可されるイメージ名を定義するための正規表現の一覧です。カタログへの APB の追加を許可するホワイトリストを作成する必要があります。レジストリー内のすべての APB を取得する必要がある場合は、最も許容度の高い正規表現である <b>*-apb\$</b> を使用できます。詳細については、 <a href="#">APB のフィルターリング</a> を参照してください。	N

フィールド	説明	必須
<b>black_list</b>	許可できないイメージ名を定義するために使用される正規表現の一覧です。詳細については、 <a href="#">APB のフィルターリング</a> を参照してください。	N
<b>images</b>	OpenShift Container レジストリーで使用されるイメージの一覧です。	N

### 8.4.1. 実稼働または開発

実稼働ブローカー設定は、Red Hat Container Catalog (RHCC) などの信頼できるコンテナディストリビューションレジストリーを参照するように設計されています。

```
registry:
- name: rhcc
  type: rhcc
  url: https://registry.redhat.io
  tag: v3.11
  white_list:
  - ".*-apb$"
- type: local_openshift
  name: localregistry
  namespaces:
  - openshift
  white_list: []
```

開発ブローカー設定は、主にブローカーの開発作業に取り組む開発者によって使用されます。開発者設定を有効にするには、レジストリー名を **dev** に設定し、**broker** セクションの **dev\_broker** フィールドを **true** に設定します。

```
registry:
  name: dev

broker:
  dev_broker: true
```

### 8.4.2. レジストリー認証情報の保存

ブローカー設定は、ブローカーによるレジストリー認証情報の読み取り方法を決定します。レジストリー認証情報は、**registry** セクションの **user** 値と **pass** 値から読み取ることができます。以下は例になります。

```
registry:
- name: isv
  type: openshift
  url: https://registry.connect.redhat.com
  user: <user>
  pass: <password>
```

これらの認証情報にパブリックにアクセスできないようにするには、**registry** セクションの **auth\_type**

フィールドを **secret** または **file** タイプに設定します。**secret** タイプは、ブローカーの namespace からシークレットを使用するようにレジストリーを設定します。一方、**file** タイプは、ボリュームとしてマウントされているシークレットを使用するようにレジストリーを設定します。

**secret** または **file** タイプを使用するには、以下を実行します。

1. 関連するシークレットには、**username** と **password** の値が定義されている必要があります。シークレットを使用する場合は、**openshift-ansible-service-broker** namespace が存在していることを確認する必要があります。シークレットはこの namespace から読み取られるためです。  
たとえば、**reg-creds.yaml** ファイルを作成します。

```
$ cat reg-creds.yaml
---
username: <user_name>
password: <password>
```

2. このファイルから **openshift-ansible-service-broker** namespace にシークレットを作成します。

```
$ oc create secret generic \
  registry-credentials-secret \
  --from-file reg-creds.yaml \
  -n openshift-ansible-service-broker
```

3. **secret** または **file** のどちらのタイプを使用するか選択します。

- **secret** タイプを使用するには、以下を実行します。

- a. ブローカー設定で、**auth\_type** を **secret** に、**auth\_name** をシークレットの名前に設定します。

```
registry:
  - name: isv
    type: openshift
    url: https://registry.connect.redhat.com
    auth_type: secret
    auth_name: registry-credentials-secret
```

- b. シークレットが置かれている namespace を設定します。

```
openshift:
  namespace: openshift-ansible-service-broker
```

- **file** タイプを使用するには、以下を実行します。

- a. **asb** デプロイメント設定を編集し、ファイルを **/tmp/registry-credentials/reg-creds.yaml** にマウントします。

```
$ oc edit dc/asb -n openshift-ansible-service-broker
```

**containers.volumeMounts** セクションに、以下を追加します。

```

volumeMounts:
  - mountPath: /tmp/registry-credentials
    name: reg-auth

```

**volumes** セクションに、以下を追加します。

```

volumes:
  - name: reg-auth
    secret:
      defaultMode: 420
      secretName: registry-credentials-secret

```

b. ブローカー設定で、**auth\_type** を **file** に、**auth\_name** をファイルの場所に設定します。

```

registry:
  - name: isv
    type: openshift
    url: https://registry.connect.redhat.com
    auth_type: file
    auth_name: /tmp/registry-credentials/reg-creds.yaml

```

### 8.4.3. APB のフィルターリング

APB は、ブローカー設定内のレジストリベースに設定された **white\_list** または **black\_list** パラメーターの組み合わせを使用して、イメージ名でフィルターリングできます。

これらはどちらもオプションの正規表現の一覧であり、特定のレジストリで一致するものを判別できるように検出されたすべての APB に対して実行されます。

表8.2 APB フィルターの動作

存在するパラメーター	許可	ブロック
ホワイトリストのみ	一覧の正規表現に一致。	一致しないすべての APB。
ブラックリストのみ	一致しないすべての APB。	一覧の正規表現に一致する APB。
両方とも存在する	ホワイトリストの正規表現に一致し、ブラックリストの正規表現に一致しない。	ブラックリストの正規表現に一致する APB。
なし	レジストリーのどの APB も許可されない。	レジストリーのすべての APB。

以下に例を示します。

#### ホワイトリストのみ

```

white_list:
  - "foo.*-apb$"
  - "^my-apb$"

```

この場合は、**foo.\*-apb\$** と **my-apb** に一致する APB が許可されます。それ以外の APB はすべて拒否されます。

### ブラックリストのみ

```
black_list:
  - "bar.*-apb$"
  - "^foobar-apb$"
```

この場合は、**bar.\*-apb\$** と **foobar-apb** に一致する APB がブロックされます。それ以外の APB はすべて許可されます。

### ホワイトリストとブラックリスト

```
white_list:
  - "foo.*-apb$"
  - "^my-apb$"
black_list:
  - "^foo-rootkit-apb$"
```

ここでは、**foo-rootkit-apb** はホワイトリストに一致するにもかかわらず、ブラックリストによって明確にブロックされます。これは、ホワイトリストの一致が上書きされるためです。

そうでない場合は、**foo.\*-apb\$** と **my-apb** に一致する APB のみが許可されます。

### ブローカー設定の registry セクションのサンプル:

```
registry:
  - type: dockerhub
    name: dockerhub
    url: https://registry.hub.docker.com
    user: <user>
    pass: <password>
    org: <org>
    white_list:
      - "foo.*-apb$"
      - "^my-apb$"
    black_list:
      - "bar.*-apb$"
      - "^foobar-apb$"
```

#### 8.4.4. モックレジストリー

モックレジストリーは、ローカルの APB 仕様を読み取る場合に便利です。イメージ仕様を検索するために外部のレジストリーにアクセスする代わりに、ローカル仕様の一覧を使用します。モックレジストリーを使用するには、レジストリーの名前を **mock** に設定します。

```
registry:
  - name: mock
    type: mock
```

#### 8.4.5. Dockerhub レジストリー

**dockerhub** タイプを使用すると、DockerHub の特定の組織から APB を読み込むことができます。(例: [ansibleplaybookbundle](#))。

```
registry:
- name: dockerhub
  type: dockerhub
  org: ansibleplaybookbundle
  user: <user>
  pass: <password>
  white_list:
  - ".*-apb$"
```

#### 8.4.6. Ansible Galaxy レジストリー

**galaxy** タイプを使用すると、[Ansible Galaxy](#) から APB ロールを使用できます。また、オプションで組織を指定できます。

```
registry:
- name: galaxy
  type: galaxy
  # Optional:
  # org: ansibleplaybookbundle
  runner: docker.io/ansibleplaybookbundle/apb-base:latest
  white_list:
  - ".*$"
```

#### 8.4.7. ローカルの OpenShift Container レジストリー

**local\_openshift** タイプを使用すると、OpenShift Container Platform クラスター内の OpenShift Container レジストリーから APB を読み込むことができます。公開された APB を検索する namespace を設定できます。

```
registry:
- type: local_openshift
  name: lo
  namespaces:
  - openshift
  white_list:
  - ".*-apb$"
```

#### 8.4.8. Red Hat Container Catalog レジストリー

**rhcc** タイプを使用すると、[Red Hat Container Catalog](#) (RHCC) レジストリーに公開された APB を読み込むことができます。

```
registry:
- name: rhcc
  type: rhcc
  url: https://registry.redhat.io
  white_list:
  - ".*-apb$"
```

### 8.4.9. Red Hat Partner Connect レジストリー

Red Hat Container Catalog のサードパーティーのイメージは Red Hat Connect Partner Registry (<https://registry.connect.redhat.com>) から提供されます。`partner_rhcc` タイプは、APB の一覧を取得し、それらの仕様を読み込むためにブローカーを Partner Registry からブートストラップすることを要求します。Partner Registry には、有効な Red Hat カスタマーポータルของผู้ーザー名およびパスワードを使った、イメージをプルするための認証が必要です。

```
registry:
- name: partner_reg
  type: partner_rhcc
  url: https://registry.connect.redhat.com
  user: <registry_user>
  pass: <registry_password>
  white_list:
  - ".*-apb$"
```

Partner Registry には認証が必要なため、ブローカーを Partner Registry URL を使用するよう設定するには以下の手動の手順も必要になります。

1. OpenShift Container Platform クラスターのすべてのノードで以下のコマンドを実行します。

```
# docker --config=/var/lib/origin/.docker \
  login -u <registry_user> -p <registry_password> \
  registry.connect.redhat.com
```

### 8.4.10. Helm チャートレジストリー

`helm` タイプを使用して、Helm チャートリポジトリから Helm チャートを使用することができます。

```
registry:
- name: stable
  type: helm
  url: "https://kubernetes-charts.storage.googleapis.com"
  runner: "docker.io/automationbroker/helm-runner:latest"
  white_list:
  - ".*"
```



#### 注記

stable リポジトリの多くの Helm チャートは OpenShift Container Platform での使用には適さず、使用する場合はエラーを出して失敗します。

### 8.4.11. API V2 Docker レジストリー

`apiv2` タイプを使用して、Docker レジストリー HTTP API V2 プロトコルを実装する `docker` レジストリーからイメージを使用することができます。

```
registry:
- name: <registry_name>
  type: apiv2
  url: <registry_url>
  user: <registry-user>
```

```

pass: <registry-password>
white_list:
  - ".*-apb$"

```

イメージをプルするためにレジストリーで認証が必要な場合、以下のコマンドを既存クラスターのすべてのノードで実行して達成できます。

```
$ docker --config=/var/lib/origin/.docker login -u <registry-user> -p <registry-password> <registry_url>
```

#### 8.4.12. Quay Docker レジストリー

**quay** タイプを使用して、[CoreOS Quay レジストリー](#) に公開される APB をロードできます。認証トークンが提供される場合、トークンがアクセスするように設定されたプライベートリポジトリーがロードされます。指定された組織のパブリックリポジトリーではロードするのにトークンは必要ありません。

```

registry:
  - name: quay_reg
    type: quay
    url: https://quay.io
    token: <for_private_repos>
    org: <your_org>
    white_list:
      - ".*-apb$"

```

イメージをプルするために Quay レジストリーで認証が必要な場合、以下のコマンドを既存クラスターのすべてのノードで実行して達成できます。

```
$ docker --config=/var/lib/origin/.docker login -u <registry-user> -p <registry-password> quay.io
```

#### 8.4.13. 複数のレジストリー

複数のレジストリーを使用して APB を論理的な組織に分割し、それらを同じブローカーから管理できます。レジスターには一意の空でない名前が必要です。一意の名前がない場合、サービスブローカーは起動に失敗し、問題について警告するエラーメッセージを表示します。

```

registry:
  - name: dockerhub
    type: dockerhub
    org: ansibleplaybookbundle
    user: <user>
    pass: <password>
    white_list:
      - ".*-apb$"
  - name: rhcc
    type: rhcc
    url: <rhcc_url>
    white_list:
      - ".*-apb$"

```

### 8.5. ブローカー認証

ブローカーは認証をサポートします。つまり、ブローカーに接続する際に、呼び出し側は各要求に対して Basic Auth または Bearer Auth 認証情報を指定する必要があります。 **curl** を使用し、以下のように簡単に実行できます。

```
-u <user_name>:<password>
```

または

```
-h "Authorization: bearer <token>
```

上記をコマンドに指定します。サービスカタログをユーザー名とパスワードの組み合わせ、またはベアラートークンが含まれるシークレットで設定する必要があります。

### 8.5.1. Basic 認証

Basic 認証の使用を有効にするには、ブローカー設定で以下を設定します。

```
broker:
  ...
  auth:
    - type: basic ①
      enabled: true ②
```

① **type** フィールドは使用する認証タイプを指定します。

② **enabled** フィールドでは、特定の認証タイプを無効にすることができます。これにより、これを無効にするために **auth** のセクション全体を削除する必要がなくなります。

#### 8.5.1.1. デプロイメントテンプレートおよびシークレット

通常、ブローカーはデプロイメントテンプレートで **ConfigMap** を使用して設定されます。ファイル設定と同様の方法で認証設定を指定できます。

以下は、**デプロイメントテンプレート** のサンプルになります。

```
auth:
  - type: basic
    enabled: ${ENABLE_BASIC_AUTH}
```

Basic 認証の別の部分には、ブローカーに対して認証するために使用されるユーザー名とパスワードが含まれます。Basic 認証の実装は別のバックエンドサービスでサポートされる可能性があるものの、現時点でサポートされている実装は **シークレット** に対応します。シークレットは `/var/run/asb_auth` の場所にあるボリュームマウントで Pod に挿入される必要があります。これは、ブローカーがユーザー名とパスワードを読み取る場所です。

**デプロイメントテンプレート** では、シークレットが指定される必要があります。以下に例を示します。

```
- apiVersion: v1
  kind: Secret
  metadata:
    name: asb-auth-secret
    namespace: openshift-ansible-service-broker
```

```
data:
  username: ${BROKER_USER}
  password: ${BROKER_PASS}
```

シークレットにはユーザー名とパスワードが含まれる必要があります。値は base64 エンコードである必要があります。それらのエントリーの値を生成する最も簡単な方法として、**echo** および **base64** コマンドを使用できます。

```
$ echo -n admin | base64 ①
YWRtaW4=
```

① **-n** オプションは非常に重要になります。

このシークレットはボリュームマウントで Pod に挿入される必要があります。これはデプロイメントテンプレートでも設定されます。

```
spec:
  serviceAccount: asb
  containers:
  - image: ${BROKER_IMAGE}
    name: asb
    imagePullPolicy: IfNotPresent
  volumeMounts:
  ...
  - name: asb-auth-volume
    mountPath: /var/run/asb-auth
```

次に **volumes** セクションで、シークレットをマウントします。

```
volumes:
  ...
  - name: asb-auth-volume
    secret:
      secretName: asb-auth-secret
```

上記により、**/var/run/asb-auth** にボリュームマウントが作成されます。このボリュームには、**asb-auth-secret** シークレットで作成されるユーザー名およびパスワードの2つのファイルがあります。

### 8.5.1.2. サービスカタログおよびブローカー通信の設定

ブローカーが Basic 認証を使用するように設定されているため、サービスカタログに対してブローカーとの通信方法について指示する必要があります。これは、ブローカーリソースの **authInfo** セクションで実行できます。

以下は、サービスカタログで **broker** リソースを作成する例になります。**spec** はサービスカタログに対し、ブローカーがリスンしている URL を示唆します。**authInfo** は認証情報を取得するために読み取る必要のあるシークレットを示唆します。

```
apiVersion: servicecatalog.k8s.io/v1alpha1
kind: Broker
metadata:
  name: ansible-service-broker
spec:
```

```
url: https://asb-1338-openshift-ansible-service-broker.172.17.0.1.nip.io
authInfo:
  basicAuthSecret:
    namespace: openshift-ansible-service-broker
    name: asb-auth-secret
```

サービスカタログの v0.0.17 以降、ブローカーのリソース設定は変更されています。

```
apiVersion: servicecatalog.k8s.io/v1alpha1
kind: ServiceBroker
metadata:
  name: ansible-service-broker
spec:
  url: https://asb-1338-openshift-ansible-service-broker.172.17.0.1.nip.io
  authInfo:
    basic:
      secretRef:
        namespace: openshift-ansible-service-broker
        name: asb-auth-secret
```

## 8.5.2. Bearer 認証

デフォルトで、認証が指定されていない場合、ブローカーはベアータークン認証 (Bearer Auth) を使用します。Bearer 認証は [Kubernetes apiserver](#) ライブラリーから委任された認証を使用します。

この設定は、[Kubernetes RBAC](#) ロールおよびロールバインディングにより、URL 接頭辞へのアクセスを付与します。ブローカーは設定オプション `cluster_url` を追加して `url_prefix` を指定します。この値はデフォルトで `openshift-ansible-service-broker` になります。

### クラスターロールの例

```
- apiVersion: authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    name: access-asb-role
  rules:
  - nonResourceURLs: ["/ansible-service-broker", "/ansible-service-broker/*"]
    verbs: ["get", "post", "put", "patch", "delete"]
```

### 8.5.2.1. デプロイメントテンプレートおよびシークレット

以下は、サービスカタログが使用できるシークレットの作成例です。この例では、ロールの `access-asb-role` がすでに作成されていることを前提としています。また、[デプロイメントテンプレート](#) が使用されています。

```
- apiVersion: v1
  kind: ServiceAccount
  metadata:
    name: ansibleservicebroker-client
    namespace: openshift-ansible-service-broker

- apiVersion: authorization.openshift.io/v1
  kind: ClusterRoleBinding
  metadata:
```

```

name: ansible-service-broker-client
subjects:
- kind: ServiceAccount
  name: ansible-service-broker-client
  namespace: openshift-ansible-service-broker
roleRef:
  kind: ClusterRole
  name: access-asb-role

- apiVersion: v1
  kind: Secret
  metadata:
    name: ansible-service-broker-client
  annotations:
    kubernetes.io/service-account.name: ansible-service-broker-client
  type: kubernetes.io/service-account-token

```

上記の例ではサービスアカウントを作成し、アクセスを **access-asb-role** に付与し、サービスアカウントトークンの **シークレット** を作成します。

### 8.5.2.2. サービスカタログおよびブローカー通信の設定

ブローカーが Bearer 認証トークンを使用するように設定されているため、サービスカタログに対してブローカーとの通信方法について指示する必要があります。これは、**broker** リソースの **authInfo** セクションで実行できます。

以下は、サービスカタログで **broker** リソースを作成する例になります。**spec** はサービスカタログに対し、ブローカーがリッスンしている URL を示唆します。**authInfo** は認証情報を取得するために読み取る必要のあるシークレットを示唆します。

```

apiVersion: servicecatalog.k8s.io/v1alpha1
kind: ServiceBroker
metadata:
  name: ansible-service-broker
spec:
  url: https://asb.openshift-ansible-service-broker.svc:1338${BROKER_URL_PREFIX}/
  authInfo:
    bearer:
      secretRef:
        kind: Secret
        namespace: openshift-ansible-service-broker
        name: ansible-service-broker-client

```

## 8.6. DAO 設定

フィールド	説明	必須
<b>etcd_host</b>	etcd ホストの URL です。	Y
<b>etcd_port</b>	<b>etcd_host</b> との通信時に使用するポートです。	Y

## 8.7. ログ設定

フィールド	説明	必須
<b>logfile</b>	ブローカーのログを書き込む場所です。	Y
<b>stdout</b>	ログを標準出力に書き込みます。	Y
<b>level</b>	ログ出力のレベルです。	Y
<b>color</b>	ログに色付けします。	Y

## 8.8. OPENSIFT 設定

フィールド	説明	必須
<b>host</b>	OpenShift Container Platform ホストです。	N
<b>ca_file</b>	認証局ファイルの場所です。	N
<b>bearer_token_file</b>	使用するベアータークンの場所です。	N
<b>image_pull_policy</b>	イメージをプルするタイミングです。	Y
<b>namespace</b>	ブローカーがデプロイされている namespace です。シークレットを介して渡されるパラメーター値などに重要です。	Y
<b>sandbox_role</b>	APB サンドボックス環境に対して指定するロールです。	Y
<b>keep_namespace</b>	APB の実行後に namespace を常に保持します。	N
<b>keep_namespace_on_error</b>	APB の実行でエラーが発生した後に namespace を保持します。	N

## 8.9. ブローカー設定

**broker** セクションでは、有効/無効にする機能をブローカーに指示します。また、完全な機能を有効にするファイルがディスク上のどこにあるかをブローカーに指示します。

フィールド	説明	デフォルト値	必須
<b>dev_broker</b>	開発ルートにアクセスできるようにします。	<b>false</b>	N
<b>launch_apb_on_bind</b>	バインドが no-op (無処理) になることを許可します。	<b>false</b>	N

フィールド	説明	デフォルト値	必須
<b>bootstrap_on_startup</b>	ブローカーが起動時に自らをブートストラップできるようにします。APB を設定済みのレジストリーから取得します。	<b>false</b>	N
<b>recovery</b>	etcd にある保留中のジョブを処理することによって、ブローカーが自らをリカバリーできるようにします。	<b>false</b>	N
<b>output_request</b>	デバッグを容易に行えるように、ブローカーが要求の受信時にそれをログファイルに出力できるようにします。	<b>false</b>	N
<b>ssl_cert_key</b>	TLS キーファイルがどこにあるかをブローカーに指示します。これが設定されない場合、API サーバーはファイルの作成を試みます。	""	N
<b>ssl_cert</b>	TLS .crt ファイルがどこにあるかをブローカーに指示します。これが設定されない場合、API サーバーはファイルの作成を試みます。	""	N
<b>refresh_interval</b>	レジストリーで新規イメージ仕様をクエリーする間隔です。	<b>"600s"</b>	N
<b>auto_escalate</b>	ブローカーが APB の実行中にユーザーのパーミッションをエスカレーションできるようにします。	<b>false</b>	N
<b>cluster_url</b>	ブローカーが予期する URL の接頭辞を設定します。	<b>openshift-ansible-service-broker</b>	N



### 注記

非同期バインドおよびバインド解除は実験的な機能であり、デフォルトではサポートされておらず、有効にされていません。非同期バインドがない場合に **launch\_apb\_on\_bind** を **true** に設定すると、バインドアクションがタイムアウトになり、再試行が実行されます。これはパラメーターの異なる同じバインド要求であるため、ブローカーは 409 Conflicts で処理します。

## 8.10. シークレット設定

**secrets** セクションでは、ブローカーの namespace のシークレットとブローカーが実行する APB 間の関連付けを作成します。ブローカーは、これらのルールに従って実行中の APB にシークレットをマウントします。これにより、ユーザーはシークレットを使用して、パラメーターをカタログやユーザーに公開せずに渡すことができます。

このセクションは一覧の形式であり、各エントリーは以下の構造を持ちます。

フィールド	説明	必須
<b>title</b>	ルールのタイトルです。表示と出力の目的でのみ使用されま す。	Y
<b>apb_name</b>	指定されたシークレットに関連付けられる APB の名前です。こ れは完全修飾名 (<registry_name>-<image_name>) です。	Y
<b>secret</b>	パラメーターをプルするシークレットの名前です。	Y

[create\\_broker\\_secret.py](#) ファイルをダウンロードし、これを使用して、この設定セクションの作成とフォーマットを行うことができます。

```
secrets:
- title: Database credentials
  secret: db_creds
  apb_name: dh-rhscl-postgresql-apb
```

## 8.11. プロキシ環境での実行

プロキシ化された OpenShift Container Platform クラスター内で OAB を実行する場合は、その中心的概念を理解し、外部ネットワークアクセスに使用するプロキシのコンテキストで検討することが重要です。

概要として、ブローカー自体はクラスター内で Pod として実行されます。そのレジスターの設定方法に応じて外部ネットワークにアクセスする必要があります。

### 8.11.1. レジストリーアダプターのホワイトリスト

ブローカーの設定済みレジストリーアダプターは、正常にブートストラップしてリモートの APB マニフェストを読み込むために、外部レジスターと通信できなければなりません。これらの要求はプロキシ経由で実行できますが、プロキシでは必要なリモートホストにアクセスできるようにする必要があります。

必要なホワイトリスト化されたホストの例:

レジストリーアダプターのタイプ	ホワイトリスト化されたホスト
<b>rhcc</b>	<b>registry.redhat.io、access.redhat.com</b>
<b>dockerhub</b>	<b>docker.io</b>

### 8.11.2. Ansible を使用したプロキシ環境でのブローカーの設定

初期インストール時に OpenShift Container Platform クラスターがプロキシの環境下で実行されるように設定した場合 ([グローバルプロキシオプションの設定](#) を参照)、OAB はデプロイ時に以下を実行します。

- クラスター全体のプロキシ設定を自動的に継承する

- `cidr` フィールドおよび `serviceNetworkCIDR` を含む必要な `NO_PROXY` 一覧を生成する

それ以外の設定は必要ありません。

### 8.11.3. プロキシ環境でのブローカーの手動設定

クラスターのグローバルプロキシオプションが初期インストール時またはブローカーのデプロイ前に設定されていない場合や、グローバルプロキシ設定を変更した場合、ブローカーの外部アクセスについてプロキシ経由で手動で設定する必要があります。

1. OAB をプロキシ環境で実行する前に [HTTP プロキシの使用](#) を確認し、クラスターがプロキシ環境で実行されるように適切に設定されていることを確認してください。とくに、クラスターは内部クラスター要求をプロキシ処理しないように設定されている必要があります。通常、これは以下の `NO_PROXY` 設定を使用して設定されます。

```
.cluster.local,.svc,<serviceNetworkCIDR_value>,<master_IP>,<master_domain>,.default
```

その他の必要な `NO_PROXY` 設定も追加する必要があります。詳細については、[NO\\_PROXY の設定](#) を参照してください。



#### 注記

バージョンなしまたは v1 の APB をデプロイするブローカーは、**172.30.0.1** も `NO_PROXY` の一覧に追加する必要があります。v2 より前の APB は、シークレットの交換ではなく、`exec` HTTP 要求を使用して実行中の APB Pod から認証情報を抽出しました。実験的なプロキシサポートがあるブローカーを OpenShift Container Platform 3.9 より前のクラスターで実行していない限り、この点を心配する必要はおそらくありません。

2. ブローカーの `DeploymentConfig` を `cluster-admin` 権限を持つユーザーとして編集します。

```
$ oc edit dc/asb -n openshift-ansible-service-broker
```

3. 以下の環境変数を設定します。

- `HTTP_PROXY`
- `HTTPS_PROXY`
- `NO_PROXY`



#### 注記

詳細については、[Pod でのプロキシ環境変数の設定](#) を参照してください。

4. 更新内容を保存した後、変更を有効にするために OAB のデプロイメント設定を再デプロイします。

```
$ oc rollout latest dc/asb -n openshift-ansible-service-broker
```

### 8.11.4. Pod でのプロキシ環境変数の設定

一般に、APB Pod 自体もプロキシ経由の外部アクセスを必要とします。ブローカーは、自らにプロ

キシー設定があることを認識すると、生成する APB Pod にこれらの環境変数を透過的に適用します。APB 内で使用されるモジュールが環境変数経由でプロキシ設定に従う限り、APB もこれらの設定に基づいて動作します。

最後に、APB によって生成されたサービスもプロキシ経由の外部ネットワークアクセスを必要とする場合があります。APB は、それ自体の実行環境でこのようなサービスを検出した場合にこれらの環境変数を明示的に設定するように作成されている**必要があります**。そうでない場合には、クラスターオペレーターが必要なサービスを変更してそれらを環境に組み込む必要があります。

## 第9章 ホストの既存クラスターへの追加

### 9.1. ホストの追加

`scaleup.yml` Playbook を実行して新規ホストをクラスターに追加できます。この Playbook はマスターをクエリーし、新規ホストの新規証明書を生成し、配布してから、設定 Playbook を新規ホストにのみ実行します。`scaleup.yml` Playbook を実行する前に、前提条件となる [ホストの準備](#) 手順をすべて完了してください。



#### 重要

`scaleup.yml` の Playbook は新規ホストの設定のみを設定します。マスターサービスの `NO_PROXY` の更新やマスターサービスの再起動は行いません。

`scaleup.yml` Playbook を実行するには、現在のクラスター設定を表す既存のインベントリーファイル (`/etc/ansible/hosts` など) が必要です。以前に `atomic-openshift-installer` コマンドを使用してインストールを実行した場合は、`~/config/openshift/hosts` を調べて、インストーラーによって生成された最新のインベントリーファイルを見つけ、そのファイルをインベントリーファイルとして使用することができます。このファイルは必要に応じて変更することができます。後で `ansible-playbook` を実行する際に `-i` を使用して、そのファイルの場所を指定する必要があります。



#### 重要

[ノードの推奨の最大数](#)については、[クラスターの最大値](#)のセクションを参照してください。

#### 手順

1. `openshift-ansible` パッケージを更新して最新の Playbook を取得します。

```
# yum update openshift-ansible
```

2. `/etc/ansible/hosts` ファイルを編集し、`new_<host_type>` を `[OSEv3:children]` セクションに追加します。たとえば、新規ノードホストを追加するには、`new_nodes` を追加します。

```
[OSEv3:children]
masters
nodes
new_nodes
```

新規マスターホストを追加するには、`new_masters` を追加します。

3. `[new_<host_type>]` セクションを作成して、新規ホストのホスト情報を指定します。以下の新規ノードの追加例で示されているように、既存のセクションと同じ様にこのセクションをフォーマットします。

```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'
```

```
[new_nodes]
node3.example.com openshift_node_group_name='node-config-infra'
```

その他のオプションについては、[ホスト変数の設定](#) を参照してください。

新規マスターを追加する場合は、`[new_masters]` セクションと `[new_nodes]` セクションの両方にホストを追加して、新規マスターホストが OpenShift SDN の一部となるようにします。

```
[masters]
master[1:2].example.com

[new_masters]
master3.example.com

[nodes]
master[1:2].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'

[new_nodes]
master3.example.com
```



### 重要

マスターホストに `node-role.kubernetes.io/infra=true` ラベルを付け、それ以外に専用インフラストラクチャーノードがない場合は、エントリーに `openshift_schedulable=true` を追加してホストにスケジューラブルであることを示すマークを明示的に付ける必要があります。そうしないと、レジストリー Pod とルーター Pod をどこにも配置できなくなります。

- Playbook ディレクトリーに切り替え、`openshift_node_group.yml` Playbook を実行します。インベントリーファイルがデフォルトの `/etc/ansible/hosts` 以外の場所にある場合は、`-i` オプションで場所を指定します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i /path/to/file] \
  playbooks/openshift-master/openshift_node_group.yml
```

これにより、新規ノードグループの ConfigMap が作成され、最終的にホスト上のノードの設定ファイルが作成されます。



### 注記

`openshift_node_group.yml` Playbook を実行すると、新規ノードのみが更新されます。クラスター内の既存ノードを更新するために実行することはできません。

- `scaleup.yml` Playbook を実行します。インベントリーファイルがデフォルトの `/etc/ansible/hosts` 以外の場所にある場合は、`-i` オプションで場所を指定します。
  - ノードを追加する場合は、以下を指定します。

```
$ ansible-playbook [-i /path/to/file] \
  playbooks/openshift-node/scaleup.yml
```

- マスターを追加する場合は、以下を実行します。

```
$ ansible-playbook [-i /path/to/file] \
  playbooks/openshift-master/scaleup.yml
```

6. EFK スタックをクラスターにデプロイしている場合は、ノードラベルを **logging-infra-fluentd=true** に設定します。

```
# oc label node/new-node.example.com logging-infra-fluentd=true
```

7. Playbook の実行後に、[インストールの検証](#) を行います。
8. **[new\_<host\_type>]** セクションで定義したホストを適切なセクションに移動します。このようにホストを移動することで、このインベントリーファイルを使用するその後の Playbook の実行で、正しくノードが処理されるようになります。**[new\_<host\_type>]** セクションは空のままにできます。たとえば、新規ノードを追加する場合は、以下のように指定します。

```
[nodes]
master[1:3].example.com
node1.example.com openshift_node_group_name='node-config-compute'
node2.example.com openshift_node_group_name='node-config-compute'
node3.example.com openshift_node_group_name='node-config-compute'
infra-node1.example.com openshift_node_group_name='node-config-infra'
infra-node2.example.com openshift_node_group_name='node-config-infra'

[new_nodes]
```

## 9.2. ETCD ホストの既存クラスターへの追加

**etcd scaleup** Playbook を実行して新規 etcd ホストを追加することができます。この Playbook は、マスターをクエリーし、新規ホストの新規証明書を生成してこれを配布し、設定 Playbook を新規ホストにのみ実行します。etcd **scaleup.yml** Playbook を実行する前に、前提条件となる [ホストの準備](#) の手順をすべて完了してください。



### 警告

これらの手順により、Ansible インベントリーの設定がクラスターと同期されます。ローカルの変更が Ansible インベントリーに表示されていることを確認します。

etcd ホストを既存クラスターに追加するには、以下を実行します。

1. **openshift-ansible** パッケージを更新して最新の Playbook を取得します。

```
# yum update openshift-ansible
```

2. `/etc/ansible/hosts` ファイルを編集し、`new_<host_type>` を `[OSEv3:children]` グループに、ホストを `new_<host_type>` グループに追加します。たとえば、新規 `etcd` を追加するには、`new_etcd` を追加します。

```
[OSEv3:children]
masters
nodes
etcd
new_etcd

[etcd]
etcd1.example.com
etcd2.example.com

[new_etcd]
etcd3.example.com
```

3. Playbook ディレクトリーに切り替え、`openshift_node_group.yml` Playbook を実行します。インベントリーファイルがデフォルトの `/etc/ansible/hosts` 以外の場所にある場合は、`-i` オプションで場所を指定します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i /path/to/file] \
  playbooks/openshift-master/openshift_node_group.yml
```

これにより、新規ノードグループの ConfigMap が作成され、最終的にホスト上のノードの設定ファイルが作成されます。



#### 注記

`openshift_node_group.yml` Playbook を実行すると、新規ノードのみが更新されます。クラスター内の既存ノードを更新するために実行することはできません。

4. `etcd scaleup.yml` Playbook を実行します。インベントリーファイルがデフォルトの `/etc/ansible/hosts` 以外の場所にある場合は、`-i` オプションで場所を指定します。

```
$ ansible-playbook [-i /path/to/file] \
  playbooks/openshift-etcd/scaleup.yml
```

5. Playbook が正常に完了したら、[インストールの検証](#) を行います。

## 9.3. 共存する ETCD での既存のマスターの置き換え

マシンを別のデータセンターに移行し、割り当てられているネットワークと IP が変更される場合には、以下の手順を実行します。

1. プライマリー `etcd` および [マスター](#) ノードをバックアップします。



#### 重要

[etcd のバックアップ](#) の説明にあるように、`/etc/etcd/` ディレクトリーがバックアップされていることを確認します。

2. 置き換えるマスターの数だけ、新規マシンをプロビジョニングします。
3. クラスターを追加するか、または展開します。たとえば、etcd が共存するマスターを 3 つ追加する場合には、マスターノード 3 つに拡張します。



## 重要

OpenShift Container Platform バージョン 3.11 の初期リリースについては、**scaleup.yml** Playbook は etcd をスケールアップしません。これは OpenShift Container Platform 3.11.59 以降で修正されています。

- a. **マスター** を追加します。このプロセスのステップ 3 で、**[new\_masters]** と **[new\_nodes]** に新規データセンターのホストを追加して、**openshift\_node\_group.yml** Playbook を実行し、マスターの **scaleup.yml** Playbook を実行します。
- b. 同じホストを **etcd** セクションに配置して、etcd **scaleup.yml** Playbook を実行し、etcd **scaleup.yml** Playbook を実行します。
- c. ホストが追加されたことを確認します。

```
# oc get nodes
```

- d. マスターホストの IP が追加されたことを確認します。

```
# oc get ep kubernetes
```

- e. etcd が追加されたことを確認します。**ETCDCTL\_API** の値は、使用するバージョンにより異なります。

```
# source /etc/etcd/etcd.conf
# ETCDCTL_API=2 etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-
file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URLS member list
```

- f. **/etc/origin/master** ディレクトリーから、インベントリーファイルの最初に記載されている新規マスターホストに、**/etc/origin/master/ca.serial.txt** をコピーします。デフォルトでは、これは **/etc/ansible/hosts** です。

1. etcd ホストを削除します。

- g. **/etc/etcd/ca** ディレクトリーを、インベントリーファイルの最初に記載されている新規 etcd ホストにコピーします。デフォルトでは、これは **/etc/ansible/hosts** です。

- h. **master-config.yaml** ファイルから以前の etcd クライアントを削除します。

```
# grep etcdClientInfo -A 11 /etc/origin/master/master-config.yaml
```

- i. マスターを再起動します。

```
# master-restart api
# master-restart controllers
```

- j. クラスターから以前の etcd メンバーを削除します。**ETCDCTL\_API** の値は、使用するバージョンにより異なります。

```
# source /etc/etcd/etcd.conf
# ETCDCTL_API=2 etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-
file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URLS member list
```

- k. 上記のコマンドの出力から ID を取得して、この ID で以前のメンバーを削除します。

```
# etcdctl --cert-file=$ETCD_PEER_CERT_FILE --key-file=$ETCD_PEER_KEY_FILE \
  --ca-file=/etc/etcd/ca.crt --endpoints=$ETCD_LISTEN_CLIENT_URL member remove
1609b5a3a078c227
```

- l. etcd Pod 定義を削除し、古い etcd ホストで etcd サービスを停止します。

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
```

1. 定義ファイルを静的 Pod のディレクトリー /etc/origin/node/pods から移動して、古いマスター API およびコントローラーサービスをシャットダウンします。

```
# mkdir -p /etc/origin/node/pods/disabled
# mv /etc/origin/node/pods/controller.yaml /etc/origin/node/pods/disabled/:
```

2. ネイティブのインストールプロセス時にデフォルトでロードバランサーとしてインストールされていた、マスターノードを、HA プロキシ設定から削除します。

3. マシンの使用を停止します。

- m. etcd Pod 定義を削除し、ホストを再起動して、削除されるマスターでノードサービスを停止します。

```
# mkdir -p /etc/origin/node/pods-stopped
# mv /etc/origin/node/pods/* /etc/origin/node/pods-stopped/
# reboot
```

- n. ノードリソースを削除します。

```
# oc delete node
```

## 9.4. ノードの移行

優先される方法およびノードのサービスが実行され、スケーリングされる方法により、ノードを個別に、またはグループ (2、5、10 など) に移行することができます。

1. 移行するノードについては、新規データセンターでのノードの使用のために新規の仮想マシンをプロビジョニングします。
2. 新規ノードを追加するには、[インフラストラクチャーを拡大](#) します。新規ノードのラベルが適切に設定されており、新規 API サーバーがロードバランサーに追加され、トラフィックを適切に送信していることを確認します。
3. 評価し、縮小します。
  - a. 現在のノード (古いデータセンター内にある) に [スケジューリング対象外](#) のマークを付けます。

- b. [ノードの退避](#) を実行し、ノード上の Pod が他のノードにスケジュールされるようにします。
      - c. 退避したサービスが新規ノードで実行されていることを確認します。
4. ノードを削除します。
  - a. ノードが空であり、実行中のプロセスがないことを確認します。
  - b. サービスを停止するか、またはノードを削除します。

## 第10章 デフォルトのイメージストリームとテンプレートの追加

### 10.1. 概要

x86\_64 アーキテクチャーのサーバーに OpenShift Container Platform をインストールしている場合、クラスターには Red Hat が提供する [イメージストリーム](#) および [テンプレート](#) の便利なセットが含まれます。これにより、開発者は新規アプリケーションを簡単に作成することができます。デフォルトで、[クラスターインストール](#) プロセスは、すべてのユーザーが表示アクセスを持つデフォルトのグローバルプロジェクトである `openshift` プロジェクトにこれらのセットを自動的に作成します。

IBM POWER アーキテクチャーのサーバーに OpenShift Container Platform をインストールしている場合には、[イメージストリーム](#) および [テンプレート](#) をクラスターに追加することができます。

### 10.2. サブスクリプションタイプ別のサービス

お使いの Red Hat アカウントのアクティブなサブスクリプションに応じて、以下のイメージストリームとテンプレートのセットが Red Hat によって提供され、サポートされます。サブスクリプションの詳細については、Red Hat の営業担当者にお問い合わせください。

#### 10.2.1. OpenShift Container Platform サブスクリプション

アクティブな `OpenShift Container Platform` サブスクリプションにより、イメージストリームとテンプレートのコアのセットが提供され、サポートされます。これには以下のテクノロジーが含まれます。

型	Technology
言語とフレームワーク	<ul style="list-style-type: none"> <li>● <a href="#">.NET Core</a></li> <li>● <a href="#">Node.js</a></li> <li>● <a href="#">Perl</a></li> <li>● <a href="#">PHP</a></li> <li>● <a href="#">Python</a></li> <li>● <a href="#">Ruby</a></li> </ul>
データベース	<ul style="list-style-type: none"> <li>● <a href="#">MariaDB</a></li> <li>● <a href="#">MongoDB</a></li> <li>● <a href="#">MySQL</a></li> <li>● <a href="#">PostgreSQL</a></li> </ul>
ミドルウェアサービス	<ul style="list-style-type: none"> <li>● <a href="#">Red Hat JBoss Web Server</a>(Tomcat)</li> <li>● <a href="#">Red Hat Single Sign-on</a></li> </ul>

型	Technology
他のサービス	<ul style="list-style-type: none"> <li>● <a href="#">Jenkins</a></li> <li>● <a href="#">Jenkins スレーブ</a></li> </ul>

### 10.2.2. xPaaS ミドルウェアアドオンサブスクリプション

xPaaS ミドルウェアイメージのサポートは、xPaaS 製品ごとに提供されるサブスクリプションである xPaaS ミドルウェアアドオンサブスクリプションで提供されます。お使いのアカウントで該当するサブスクリプションがアクティブになっている場合は、以下のテクノロジーのイメージストリームとテンプレートが提供され、サポートされます。

型	Technology
ミドルウェアサービス	<ul style="list-style-type: none"> <li>● <a href="#">Red Hat JBoss A-MQ</a></li> <li>● <a href="#">Red Hat JBoss BPM Suite Intelligent Process Server</a></li> <li>● <a href="#">Red Hat JBoss BRMS Decision Server</a></li> <li>● <a href="#">Red Hat JBoss Data Grid</a></li> <li>● <a href="#">Red Hat JBoss EAP</a></li> <li>● <a href="#">Red Hat Fuse on OpenShift</a></li> <li>● <a href="#">Red Hat JBoss Data Virtualization</a></li> </ul>

## 10.3. 操作を始める前に

このトピックのタスクの実行を検討する前に、以下のいずれかを実行してこれらのイメージストリームとテンプレートが OpenShift Container Platform クラスターにすでに登録されているかどうかを確認してください。

- Web コンソールにログインして **Add to Project** をクリックします。
- CLI を使用して **openshift** プロジェクト用のイメージストリームとテンプレートの一覧を表示します。

```
$ oc get is -n openshift
$ oc get templates -n openshift
```

デフォルトのイメージストリームとテンプレートが削除または変更されている場合は、このトピックに従ってデフォルトのオブジェクトを各自で作成できます。そうしない場合は、以下の指示に従う必要はありません。

## 10.4. 前提条件

デフォルトのイメージストリームとテンプレートを作成する前に、以下を確認してください。

- **統合コンテナイメージレジストリー** サービスが OpenShift Container Platform インストールにデプロイされている必要があります。
- **oc create** コマンドを **cluster-admin** 権限で実行できる必要があります。このコマンドは、デフォルトの **openshift** プロジェクトで動作するためです。
- **openshift-ansible** RPM パッケージがインストールされている必要があります。手順については、**ソフトウェアの前提条件** を参照してください。
- IBM POWER8 または IBM POWER9 サーバーでのオンプレミスインストールの場合、**openshift** namespace に **registry.redhat.io** の **シークレット** を作成します。
- イメージストリームとテンプレートが含まれているディレクトリーのシェル変数を定義します。これにより、以降のセクションで使用するコマンドが大幅に短くなります。これを実行するには、以下を行います。
  - x86\_64 サーバーでのクラウドインストールおよびオンプレミスインストールの場合は、以下のようになります。

```
$ IMAGESTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/x86_64/image-streams"; \
  XPAASSTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/x86_64/xpaas-streams"; \
  XPAASTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/x86_64/xpaas-templates"; \
  DBTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/x86_64/db-templates"; \
  QSTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/x86_64/quickstart-templates"
```

- IBM POWER8 または IBM POWER9 サーバーでのオンプレミスインストールの場合は、以下のようになります。

```
IMAGESTREAMDIR="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/ppc64le/image-streams"; \
  DBTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/ppc64le/db-templates"; \
  QSTEMPLATES="/usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/ppc64le/quickstart-templates"
```

## 10.5. OPENSIFT CONTAINER PLATFORM イメージのイメージストリームの作成

ノードホストが Red Hat Subscription Manager を使用してサブスクライブされていて、Red Hat Enterprise Linux (RHEL) 7 ベースのイメージを使用したイメージストリームのコアセットを使用する必要がある場合には、以下を実行します。

```
$ oc create -f $IMAGESTREAMDIR/image-streams-rhel7.json -n openshift
```

または、CentOS 7 ベースのイメージを使用するイメージストリームのコアセットを作成するには、以下を実行します。

```
$ oc create -f $IMAGESTREAMDIR/image-streams-centos7.json -n openshift
```

CentOS と RHEL の両方のイメージストリームセットは同じ名前なので、両方を作成することはできません。両方のイメージストリームセットをユーザーが使用できるようにするには、一方のセットを別のプロジェクトに作成するか、いずれかのファイルを編集し、イメージストリームの名前を一意的な名前に変更します。

## 10.6. XPAAS ミドルウェアイメージのイメージストリームの作成

xPaaS ミドルウェアイメージストリームは、JBoss EAP、JBoss JWS、JBoss A-MQ、Red Hat Fuse on OpenShift、Decision Server、JBoss Data Virtualization、および JBoss Data Grid のイメージを提供します。これらのイメージは、提供されるテンプレートを使用してこれらのプラットフォームのアプリケーションを作成するために使用できます。

xPaaS ミドルウェアイメージストリームセットを作成するには、以下を実行します。

```
$ oc create -f $XPAASSTREAMDIR/jboss-image-streams.json -n openshift
```



### 注記

これらのイメージストリームによって参照されるイメージにアクセスするには、該当する xPaaS ミドルウェアサブスクリプションが必要です。

## 10.7. データベースサービステンプレートの作成

データベースサービステンプレートを使用すると、他のコンポーネントで利用できるデータベースイメージを簡単に実行できます。データベース ([MongoDB](#)、[MySQL](#)、および [PostgreSQL](#)) ごとに、2 つのテンプレートが定義されています。

1 つのテンプレートはコンテナ内の一時ストレージを使用します。つまり、保存データは Pod の移動などによってコンテナが再起動されると失われます。このテンプレートは、デモ目的にのみ使用してください。

もう 1 つのテンプレートは永続ボリュームをストレージに使用しますが、OpenShift Container Platform インストールに [永続ボリューム](#) が設定されている必要があります。

データベーステンプレートのコアセットを作成するには、以下を実行します。

```
$ oc create -f $DBTEMPLATES -n openshift
```

テンプレートを作成したら、ユーザーは各種のテンプレートを簡単にインスタンス化し、データベースデプロイメントにすばやくアクセスできるようになります。

## 10.8. インスタントアプリケーションおよびクイックスタートテンプレートの作成

インスタントアプリケーションおよびクイックスタートテンプレートでは、実行中のアプリケーションのオブジェクトの完全なセットを定義します。これらには以下が含まれます。

- [GitHub](#) パブリックリポジトリにあるソースからアプリケーションをビルドするための [ビルド設定](#)
- ビルド後にアプリケーションイメージをデプロイするための [デプロイメント設定](#)
- アプリケーション Pod に負荷分散を提供する [サービス](#)

- アプリケーションに外部アクセスを提供する [ルート](#)

いくつかのテンプレートでは、アプリケーションがデータベース操作を実行できるように、データベースデプロイメントとサービスも定義します。



### 注記

データベースを定義するテンプレートでは、一時ストレージを使用してデータベースコンテンツを格納します。データベース Pod が何らかの理由で再起動されると、データベースの全データが失われてしまうので、これらのテンプレートはデモ目的でのみ使用する必要があります。

これらのテンプレートを使用すると、ユーザーは、OpenShift Container Platform で提供される各種の言語イメージを使用する完全なアプリケーションを簡単にインスタンス化できます。また、インストール時にテンプレートのパラメーターをカスタマイズし、サンプルリポジトリではなく独自のリポジトリからソースがビルドされるようにできます。つまり、これは新規アプリケーションのビルドの単純な開始点となります。

コアのインスタントアプリケーションおよびクイックスタートテンプレートを作成するには、以下を実行します。

```
$ oc create -f $QSTEMPLATES -n openshift
```

各種の xPaaS ミドルウェア製品 (**JBoss EAP**、**JBoss JWS**、**JBoss A-MQ**、**Red Hat Fuse on OpenShift**、**Decision Server**、および **JBoss Data Grid**) を使用するアプリケーションを作成するためのテンプレートのセットも用意されています。このテンプレートセットを登録するには、以下を実行します。

```
$ oc create -f $XPAASTEMPLATES -n openshift
```



### 注記

xPaaS ミドルウェアテンプレートには、[xPaaS ミドルウェアイメージストリーム](#) が必要です。さらに、xPaaS ミドルウェアイメージストリームには、該当する xPaaS ミドルウェアサブスクリプションが必要です。



### 注記

データベースを定義するテンプレートでは、一時ストレージを使用してデータベースコンテンツを格納します。データベース Pod が何らかの理由で再起動されると、データベースの全データが失われてしまうので、これらのテンプレートはデモ目的でのみ使用する必要があります。

## 10.9. 次のステップ

これらのアーティファクトを作成したら、開発者は [Web コンソール](#) にログインし、[テンプレートからの作成](#) フローを実行できるようになります。任意のデータベースまたはアプリケーションテンプレートを選択し、現在のプロジェクトで実行するデータベースサービスまたはアプリケーションを作成できます。一部のアプリケーションテンプレートでは独自のデータベースサービスも定義することに注意してください。

サンプルアプリケーションはすべて、**SOURCE\_REPOSITORY\_URL** パラメーター値が示すように、テンプレートのデフォルトの参照先である GitHub リポジトリからビルドされます。これらのリポジト

リーはフォークすることができ、テンプレートから作成する際にフォークを **SOURCE\_REPOSITORY\_URL** パラメーター値として指定できます。これにより、開発者は独自のアプリケーションの作成を試行することができます。

開発者は、開発者ガイドの [インスタントアプリおよびクイックスタートテンプレートの使用](#) セクションでこれらの手順を確認できます。

## 第11章 カスタム証明書の設定

### 11.1. 概要

管理者は、OpenShift Container Platform API のパブリックホスト名および [Web コンソール](#) 用のカスタム提供証明書を設定できます。この設定は、[クラスターインストール](#) の実行時か、またはインストール後に行うことができます。

### 11.2. 証明書チェーンの設定

証明書チェーンが使用される場合、すべての証明書は単一の名前付き証明書ファイルに手動で連結される必要があります。これらの証明書は、以下の順序で配置する必要があります。

- OpenShift Container Platform マスターホスト証明書
- 中間 CA 証明書
- ルート CA 証明書
- サードパーティー証明書

この証明書チェーンを作成するには、証明書を共通ファイルに連結します。各証明書にこのコマンドを実行し、これらの証明書が以前に定義した順序で配置されていることを確認します。

```
$ cat <certificate>.pem >> ca-chain.cert.pem
```

### 11.3. インストール時のカスタム証明書の設定

クラスターインストールの実行時に、カスタム証明書はインベントリーファイルで設定可能な **openshift\_master\_named\_certificates** パラメーターと **openshift\_master\_overwrite\_named\_certificates** パラメーターを使用して設定できます。詳細は、[Ansible を使用したカスタム証明書の設定](#) を参照してください。

#### カスタム証明書の設定パラメーター

```
openshift_master_overwrite_named_certificates=true ❶
openshift_master_named_certificates=[{"certfile": "/path/on/host/to/crt-file", "keyfile":
"/path/on/host/to/key-file", "names": ["public-master-host.com"], "cafile": "/path/on/host/to/ca-file"}] ❷
openshift_hosted_router_certificate={"certfile": "/path/on/host/to/app-crt-file", "keyfile":
"/path/on/host/to/app-key-file", "cafile": "/path/on/host/to/app-ca-file"} ❸
```

❶ **openshift\_master\_named\_certificates** パラメーターの値を指定した場合は、このパラメーターを **true** に設定します。

❷ **マスター API 証明書** をプロビジョニングします。必要な場合は、**certFile** パラメーターに指定された証明書ファイル用に、**証明書チェーン** の形成に必要なファイルをすべて連結します。

❸ **ルーターのワイルドカード証明書** をプロビジョニングします。

以下は、マスター API 証明書のパラメーターの例です。

```
openshift_master_overwrite_named_certificates=true
```

```
openshift_master_named_certificates=[{"names": ["master.148.251.233.173.nip.io"], "certfile":
"/home/cloud-user/master.148.251.233.173.nip.io.cert.pem", "keyfile": "/home/cloud-
user/master.148.251.233.173.nip.io.key.pem", "cafile": "/home/cloud-user/master-bundle.cert.pem"}]
```

以下は、ルーターワイルドカード証明書のパラメーターの例です。

```
openshift_hosted_router_certificate={"certfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.cert.pem", "keyfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.key.pem", "cafile": "/home/cloud-user/ca-chain.cert.pem"}
```

## 11.4. WEB コンソールまたは CLI 用のカスタム証明書の設定

Web コンソールおよび CLI 用のカスタム証明書は、**マスター設定ファイル** の `servicingInfo` セクションで指定できます。

- **servicingInfo.namedCertificates** セクションでは、Web コンソール用のカスタム証明書を指定します。
- **servicingInfo** セクションでは、CLI およびその他の API 呼び出し用のカスタム証明書を指定します。

この方法で複数の証明書を設定し、それぞれの証明書を **複数のホスト名**、**複数のルーター**、または **OpenShift Container Platform イメージレジストリー** に関連付けることができます。

デフォルトの証明書は、**namedCertificates** のほかにも **servicingInfo.certFile** および **servicingInfo.keyFile** 設定セクションに設定する必要があります。



### 注記

**namedCertificates** セクションは、`/etc/origin/master/master-config.yaml` ファイルの **masterPublicURL** および **oauthConfig.assetPublicURL** 設定に関連付けられたホスト名についてのみ設定する必要があります。**masterURL** に関連付けられたホスト名にカスタム提供証明書を使用すると、インフラストラクチャーコンポーネントが内部の **masterURL** ホストを使用してマスター API と通信しようとするため、TLS エラーが発生します。

### カスタム証明書の設定

```
servicingInfo:
  logoutURL: ""
  masterPublicURL: https://openshift.example.com:8443
  publicURL: https://openshift.example.com:8443/console/
  bindAddress: 0.0.0.0:8443
  bindNetwork: tcp4
  certFile: master.server.crt ①
  clientCA: ""
  keyFile: master.server.key ②
  maxRequestsInFlight: 0
  requestTimeoutSeconds: 0
  namedCertificates:
    - certFile: wildcard.example.com.crt ③
      keyFile: wildcard.example.com.key ④
```

```
names:
  - "openshift.example.com"
metricsPublicURL: "https://metrics.os.example.com/hawkular/metrics"
```

- 1 CLI およびその他の API 呼び出し用の証明書ファイルへのパス。
- 2 CLI およびその他の API 呼び出しのキーファイルへのパス。
- 3 OpenShift Container Platform API および Web コンソールのパブリックホスト名の証明書ファイルへのパス。必要な場合は、**certFile** パラメーターに指定された証明書ファイル用に、[証明書チェーン](#) の形成に必要なファイルをすべて連結します。
- 4 OpenShift Container Platform API および Web コンソールのパブリックホスト名のキーファイルへのパスです。

**Ansible インベントリーファイル** (デフォルトでは `/etc/ansible/hosts`) の

`openshift_master_cluster_public_hostname` パラメーターと `openshift_master_cluster_hostname` パラメーターは異ならなければなりません。これらが同じであると、名前付き証明書が失敗し、証明書の再インストールが必要になります。

```
# Native HA with External LB VIPs
openshift_master_cluster_hostname=internal.paas.example.com
openshift_master_cluster_public_hostname=external.paas.example.com
```

OpenShift Container Platform で DNS を使用方法の詳細は、[DNS のインストールの前提条件](#) を参照してください。

この方法では、OpenShift Container Platform によって生成される自己署名証明書を利用して、必要に応じて信頼できるカスタム証明書を個々のコンポーネントに追加できます。

内部インフラストラクチャーの証明書は自己署名のままであることを注意してください。これは一部のセキュリティーチームや PKI チームから不適切な使用法と見なされる場合があります。ただし、これらの証明書を信頼するクライアントはクラスター内のその他のコンポーネントだけであるため、これに伴うリスクは最小限です。外部のユーザーとシステムはすべて、信頼できるカスタム証明書を使用します。

相対パスは、マスター設定ファイルの場所に基づいて解決されます。設定の変更が反映されるようにサーバーを再起動します。

## 11.5. カスタムマスターホスト証明書の設定

OpenShift Container Platform の外部ユーザーとの信頼できる接続を容易にするために、**Ansible インベントリーファイル** (デフォルトでは `/etc/ansible/hosts`) の `openshift_master_cluster_public_hostname` パラメーターで指定されたドメイン名に一致する名前付き証明書をプロビジョニングできます。

この証明書は Ansible がアクセスできるディレクトリーに置き、以下のように Ansible インベントリーファイルにパスを追加する必要があります。

```
openshift_master_named_certificates=[{"certfile": "/path/to/console.ocp-c1.myorg.com.crt", "keyfile":
"/path/to/console.ocp-c1.myorg.com.key", "names": ["console.ocp-c1.myorg.com"]}]
```

パラメーター値は以下ようになります。

- **certfile** は、OpenShift Container Platform カスタムマスター API 証明書を含むファイルへのパスです。
- **keyfile** は、OpenShift Container Platform カスタムマスター API 証明書キーを含むファイルへのパスです。
- **names** は、クラスターのパブリックホスト名です。

ファイルパスは、Ansible が実行されるシステムにとってローカルでなければなりません。証明書はマスターホストにコピーされ、`/etc/origin/master` ディレクトリー内にデプロイされます。

レジストリーのセキュリティを保護する場合、サービスのホスト名と IP アドレスをレジストリーのサーバー証明書に追加します。SAN (Subject Alternative Name) には以下の情報が含まれている必要があります。

- 2つのサービスホスト名。

```
docker-registry.default.svc.cluster.local
docker-registry.default.svc
```

- サービス IP アドレス。  
以下に例を示します。

```
172.30.252.46
```

以下のコマンドを使ってコンテナイメージレジストリーのサービス IP アドレスを取得します。

```
oc get service docker-registry --template='{{.spec.clusterIP}}'
```

- パブリックホスト名。

```
docker-registry-default.apps.example.com
```

以下のコマンドを使ってコンテナイメージレジストリーのパブリックホスト名を取得します。

```
oc get route docker-registry --template '{{.spec.host}}'
```

たとえば、サーバー証明書には以下のような SAN の詳細が記載されるはずです。

```
X509v3 Subject Alternative Name:
  DNS:docker-registry-public.openshift.com, DNS:docker-registry.default.svc, DNS:docker-registry.default.svc.cluster.local, DNS:172.30.2.98, IP Address:172.30.2.98
```

## 11.6. デフォルトルーター用のカスタムワイルドカード証明書の設定

OpenShift Container Platform のデフォルトルーターをデフォルトのワイルドカード証明書を使って設定できます。デフォルトのワイルドカード証明書を使用すると、OpenShift Container Platform にデプロイされているアプリケーションでカスタム証明書を使用せずにデフォルトの暗号化を簡単に使用することができます。



## 注記

デフォルトのワイルドカード証明書の使用は、非実稼働環境でのみ推奨されます。

デフォルトのワイルドカード証明書を設定するには、\*.<app\_domain>で有効な証明書をプロビジョニングします。ここで、<app\_domain> は、[Ansible インベントリーファイル](#) (デフォルトでは `/etc/ansible/hosts`) の `openshift_master_default_subdomain` の値です。プロビジョニングが完了したら、証明書、キー、CA 証明書ファイルを Ansible ホストに置き、以下の行を Ansible インベントリーファイルに追加します。

```
openshift_hosted_router_certificate={"certfile": "/path/to/apps.c1-ocp.myorg.com.crt", "keyfile":
"/path/to/apps.c1-ocp.myorg.com.key", "cafile": "/path/to/apps.c1-ocp.myorg.com.ca.crt"}
```

以下に例を示します。

```
openshift_hosted_router_certificate={"certfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.cert.pem", "keyfile": "/home/cloud-user/star-
apps.148.251.233.173.nip.io.key.pem", "cafile": "/home/cloud-user/ca-chain.cert.pem"}
```

パラメーター値は以下のようになります。

- `certfile` は、OpenShift Container Platform ルーターのワイルドカード証明書が含まれるファイルへのパスです。
- `keyfile` は、OpenShift Container Platform ルーターワイルドカード証明書キーが含まれるファイルへのパスです。
- `cafile` は、このキーと証明書のルート CA を含むファイルへのパスです。中間 CA を使用している場合は、中間 CA とルート CA の両方がこのファイルに含まれている必要があります。

これらの証明書ファイルを OpenShift Container Platform クラスターで初めて使用する場合は、Playbook ディレクトリーに切り替え、Ansible `deploy_router.yml` Playbook を実行し、これらのファイルを OpenShift Container Platform 設定ファイルに追加します。この Playbook は、証明書ファイルを `/etc/origin/master/` ディレクトリーに追加します。

```
# ansible-playbook [-i /path/to/inventory] \
/usr/share/ansible/openshift-ansible/playbooks/openshift-hosted/deploy_router.yml
```

これらの証明書を使用するのが初めてでない場合、既存の証明書を変更するか、期限切れの証明書を置き換える場合などは、Playbook ディレクトリーに切り替え、以下の Playbook を実行します。

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/redeploy-certificates.yml
```



## 注記

この Playbook を実行する場合は、証明書名を変更しないでください。証明書名を変更した場合は、新規証明書の場合と同様に Ansible `deploy_cluster.yml` Playbook を再実行してください。

## 11.7. イメージレジストリー用のカスタム証明書の設定

OpenShift Container Platform イメージレジストリーは、ビルドとデプロイメントを容易にする内部サービスです。レジストリーとの通信の大部分は、OpenShift Container Platform の内部コンポーネン

トによって処理されます。そのため、レジストリーサービス自体が使用する証明書を置き換える必要はありません。

ただし、デフォルトでは、レジストリーは、外部のシステムとユーザーがイメージのプルとプッシュを実行できるルートを使用します。内部の自己署名証明書を使用する代わりに、外部ユーザーに提供されるカスタム証明書を使用して **re-encrypt** ルートを使用することができます。

これを設定するには、コードの [以下の行](#) を、Ansible インベントリーファイル (デフォルトは `/etc/ansible/hosts` ファイルの **[OSEv3:vars]** セクションに追加します。レジストリールートで使用する証明書を指定します。

```
openshift_hosted_registry_routehost=registry.apps.c1-ocp.myorg.com ❶
openshift_hosted_registry_routecertificates={"certfile": "/path/to/registry.apps.c1-ocp.myorg.com.crt",
"keyfile": "/path/to/registry.apps.c1-ocp.myorg.com.key", "cafile": "/path/to/registry.apps.c1-ocp.myorg.com-ca.crt"} ❷
openshift_hosted_registry_routetermination=reencrypt ❸
```

❶ レジストリーのホスト名です。

❷ `cacert`、`cert`、および `key` ファイルの場所です。

- `certfile` は、OpenShift Container Platform レジストリー証明書を含むファイルへのパスです。
- `keyfile` は、OpenShift Container Platform レジストリーの証明書キーを含むファイルへのパスです。
- `cafile` は、このキーと証明書のルート CA を含むファイルへのパスです。中間 CA を使用している場合は、中間 CA とルート CA の両方がこのファイルに含まれている必要があります。

❸ 暗号化を実行する場所を指定します。

- edge ルーターで暗号化を終了し、送信先から提供される新規の証明書で再暗号化するには、**reencrypt** に設定し、**re-encrypt** ルートを指定します。
- 送信先で暗号化を終了するには、**passthrough** に設定します。トラフィックは送信先で復号化されます。

## 11.8. ロードバランサー用のカスタム証明書の設定

OpenShift Container Platform クラスターでデフォルトのロードバランサーか、またはエンタープライズレベルのロードバランサーを使用している場合、カスタム証明書の使用により、パブリックに署名されたカスタム証明書を使って Web コンソールと API を外部で利用できるようにし、既存の内部証明書を内部のエンドポイント用に残しておくことができます。

カスタム証明書をこの方法で使用するよう OpenShift Container Platform を設定するには、以下を実行します。

1. マスター設定ファイルの `servicingInfo` セクションを編集します。

```
servicingInfo:
  logoutURL: ""
  masterPublicURL: https://openshift.example.com:8443
```

```
publicURL: https://openshift.example.com:8443/console/
bindAddress: 0.0.0.0:8443
bindNetwork: tcp4
certFile: master.server.crt
clientCA: ""
keyFile: master.server.key
maxRequestsInFlight: 0
requestTimeoutSeconds: 0
namedCertificates:
  - certFile: wildcard.example.com.crt ❶
    keyFile: wildcard.example.com.key ❷
    names:
      - "openshift.example.com"
metricsPublicURL: "https://metrics.os.example.com/hawkular/metrics"
```

- ❶ OpenShift Container Platform API および Web コンソールのパブリックホスト名の証明書ファイルへのパス。必要な場合は、**certFile** パラメーターに指定された証明書ファイル用に、**証明書チェーン**の形成に必要なファイルをすべて連結します。
- ❷ OpenShift Container Platform API および Web コンソールのパブリックホスト名のキーファイルへのパスです。



### 注記

**masterPublicURL** および **oauthConfig.assetPublicURL** 設定に関連付けられたホスト名についてのみ **namedCertificates** セクションを設定します。**masterURL** に関連付けられたホスト名にカスタム提供証明書を使用すると、インフラストラクチャーコンポーネントが内部の masterURL ホストを使用してマスター API と通信しようとするため、TLS エラーが発生します。

2. Ansible インベントリーファイル (デフォルトでは `/etc/ansible/hosts`) で **openshift\_master\_cluster\_public\_hostname** と **openshift\_master\_cluster\_hostname** のパラメーターを指定します。これらの値は異ならなければなりません。同じ値を指定した場合、名前付き証明書が失敗します。

```
# Native HA with External LB VIPs
openshift_master_cluster_hostname=paas.example.com ❶
openshift_master_cluster_public_hostname=public.paas.example.com ❷
```

- ❶ SSL パススルー用に設定された内部ロードバランサーの FQDN です。
- ❷ カスタム (パブリック) 証明書を使用する外部ロードバランサーの FQDN です。

お使いのロードバランサー環境に固有の情報については、お使いのプロバイダーについての [OpenShift Container Platform リファレンスアーキテクチャー](#) と [Custom Certificate SSL Termination \(Production\)](#) を参照してください。

## 11.9. カスタム証明書の変更およびクラスターへの組み込み

カスタムマスターおよびカスタムルーター証明書を既存の OpenShift Container Platform クラスターで変更できます。

### 11.9.1. カスタムマスター証明書の変更およびクラスターへの組み込み

カスタム証明書を変更するには、以下を実行します。

1. Ansible インベントリーファイルを編集して **openshift\_master\_overwrite\_named\_certificates=true** を設定します。
2. **openshift\_master\_named\_certificates** パラメーターを使用して証明書へのパスを指定します。

```
openshift_master_overwrite_named_certificates=true
openshift_master_named_certificates=[{"certfile": "/path/on/host/to/crt-file", "keyfile":
"/path/on/host/to/key-file", "names": ["public-master-host.com"], "cafile": "/path/on/host/to/ca-
file"}] ❶
```

- ❶ マスター API 証明書 へのパスです。必要な場合は、**certFile** パラメーターに指定された証明書ファイル用に、**証明書チェーン** の形成に必要なファイルをすべて連結します。

3. Playbook ディレクトリーに切り替え、以下の Playbook を実行します。

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/redeploy-certificates.yml
```

4. 名前付き証明書を使用する場合:
  - a. 各マスターノードの **master-config.yaml** ファイルで **証明書パラメーター** を更新します。
  - b. 変更を有効にするために OpenShift Container Platform マスターサービスを再起動します。

```
# master-restart api
# master-restart controllers
```

### 11.9.2. カスタムルーター証明書の変更およびクラスターへの組み込み

カスタムルーター証明書を変更し、これを組み込むには、以下を実行します。

1. Ansible インベントリーファイルを編集して **openshift\_master\_overwrite\_named\_certificates=true** を設定します。
2. **openshift\_hosted\_router\_certificate** パラメーターを使用して証明書へのパスを指定します。

```
openshift_master_overwrite_named_certificates=true
openshift_hosted_router_certificate={"certfile": "/path/on/host/to/app-crt-file", "keyfile":
"/path/on/host/to/app-key-file", "cafile": "/path/on/host/to/app-ca-file"} ❶
```

- ❶ ルーターワイルドカード証明書 へのパスです。

3. Playbook ディレクトリーに切り替え、以下の Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook playbooks/openshift-hosted/redeploy-router-certificates.yml
```

## 11.10. 他のコンポーネントでのカスタム証明書の使用

ロギングやメトリクスなどのその他のコンポーネントでカスタム証明書を使用する方法については、[Certificate Management](#) を参照してください。

## 第12章 証明書の再デプロイ

### 12.1. 概要

OpenShift Container Platform は、以下のコンポーネントで証明書を使用してセキュアな接続を提供します。

- マスター (API サーバーとコントローラー)
- etcd
- ノード
- レジストリー
- ルーター

インストーラーによって提供される Ansible Playbook を使用すると、クラスター証明書の有効期限を自動的にチェックできます。これらの証明書を自動的にバックアップしたり、再デプロイしたりするための Playbook も提供されます。これらの Playbook を使用すると、一般的な証明書エラーを修正できます。

証明書の再デプロイのユースケースとして以下が考えられます。

- インストーラーによって誤ったホスト名が検出され、そのことがすぐに判明しなかった。
- 証明書の有効期限が切れており、証明書を更新する必要がある。
- 新しい CA があり、その CA を代わりに使用する証明書を作成する。

### 12.2. 証明書の有効期限のチェック

インストーラーを使用して、設定可能な日数内に有効期限が切れる証明書に関する警告やすでに有効期限が切れた証明書に関する通知を受け取ることができます。証明書の有効期限切れ Playbook では、Ansible の **openshift\_certificate\_expiry** ロールが使用されます。

ロールによって検査される証明書には、以下が含まれます。

- マスターおよびノードサービス証明書
- etcd シークレットのルーターおよびレジストリーサービス証明書
- マスター、ノード、ルーター、レジストリー、および **cluster-admin** ユーザーの **kubeconfig** ファイル
- etcd 証明書 (組み込み証明書を含む)

[すべての OpenShift TLS 証明書の有効期限を一覧表示する](#) 方法を説明します。

#### 12.2.1. ロール変数

**openshift\_certificate\_expiry** ロールは、以下の変数を使用します。

表12.1 コア変数

変数名	デフォルト値	説明
<code>openshift_certificate_expiry_config_base</code>	<code>/etc/origin</code>	OpenShift Container Platform 基本設定ディレクトリーです。
<code>openshift_certificate_expiry_warning_days</code>	365	現在を起点として指定された日数内に有効期限が切れる証明書にフラグを付けます。
<code>openshift_certificate_expiry_show_all</code>	いいえ	正常な (期限切れや警告のない) 証明書を結果に組み込みます。

表12.2 オプションの変数

変数名	デフォルト値	説明
<code>openshift_certificate_expiry_generate_html_report</code>	いいえ	有効期限切れのチェック結果に関するHTML レポートを生成します。
<code>openshift_certificate_expiry_html_report_path</code>	<code>\$HOME/cert-expiry-report.yyyymmddTHHMMSS.html</code>	HTML レポートを保存するための完全パス。デフォルトは、ホームディレクトリーとレポートファイルのタイムスタンプ接尾辞の設定になります。
<code>openshift_certificate_expiry_save_json_results</code>	いいえ	有効期限のチェック結果を JSON ファイルとして保存します。
<code>openshift_certificate_expiry_json_results_path</code>	<code>\$HOME/cert-expiry-report.yyyymmddTHHMMSS.json</code>	JSON レポートを保存するための完全パスです。デフォルトは、ホームディレクトリーとレポートファイルのタイムスタンプ接尾辞の設定になります。

### 12.2.2. 証明書の有効期限切れ Playbook の実行

OpenShift Container Platform インストーラーは、`openshift_certificate_expiry` ロールのさまざまな設定セットを使用して証明書の有効期限切れ Playbook のサンプル一式を提供します。

これらの Playbook は、クラスターを表す [インベントリーファイル](#) と一緒に使用する必要があります。最適な結果を得るには、`ansible-playbook` に `-v` オプションを指定して実行します。

`easy-mode.yaml` のサンプル Playbook を使用すると、ロールアウトを仕様に合わせて調整する前に試すことができます。この Playbook は以下を実行します。

- JSON レポートと定型化された HTML レポートを `$HOME` ディレクトリーに生成します。
- ほぼ常に結果が得られるように警告期間を長い期間に設定します。
- すべての証明書 (正常であるかどうかを問わず) を結果に組み込みます。

Playbook ディレクトリーに切り替え、**easy-mode.yaml** Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v -i <inventory_file> \
  playbooks/openshift-checks/certificate_expiry/easy-mode.yaml
```

### 他のサンプル Playbook

その他のサンプル Playbook も `/usr/share/ansible/openshift-ansible/playbooks/certificate_expiry/` ディレクトリーから直接実行できます。

表12.3 他のサンプル Playbook

ファイル名	使用方法
default.yaml	<b>openshift_certificate_expiry</b> ロールのデフォルトの動作を生成します。
html_and_json_default_paths.yaml	HTML および JSON アーティファクトをデフォルトのパスに生成します。
longer_warning_period.yaml	有効期限切れの警告期間を 1500 日に変更します。
longer-warning-period-json-results.yaml	有効期限切れの警告期間を 1500 日に変更し、結果を JSON ファイルとして保存します。

これらのサンプル Playbook を実行するには、以下を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v -i <inventory_file> \
  playbooks/openshift-checks/certificate_expiry/<playbook>
```

### 12.2.3. 出力形式

上述のように、チェックレポートは 2 つの形式で出力できます。機械の解析に適した JSON 形式と簡単にスキミングできる定型化された HTML ページを使用できます。

#### HTML レポート

インストーラーには、HTML レポートのサンプルが付属しています。以下のファイルをブラウザで開いて表示できます。

```
/usr/share/ansible/openshift-ansible/roles/openshift_certificate_expiry/examples/cert-expiry-report.html
```

#### JSON レポート

保存された JSON の結果には、**data** と **summary** の 2 つの最上位レベルのキーがあります。

**data** キーは、キーが検査された各ホストの名前で、値が該当する各ホストで識別された証明書のチェック結果であるハッシュです。

**summary** キーは、以下の条件に当てはまる証明書の合計数をまとめたハッシュです。

- クラスター全体で検査済み

- 問題なし
- 設定された警告期間内に有効期限が切れる
- すでに有効期限が切れている

完全な JSON レポートのサンプルについては、`/usr/share/ansible/openshift-ansible/roles/openshift_certificate_expiry/examples/cert-expiry-report.json` を参照してください。

JSON データのサマリーでは、さまざまなコマンドラインツールを使用して警告や有効期限を簡単にチェックできます。たとえば、`grep` を使用して `summary` という語を検索し、一致した箇所の後ろの 2 行を出力できます (`-A2`)。

```
$ grep -A2 summary $HOME/cert-expiry-report.yyyymmddTHHMMSS.json
"summary": {
  "warning": 16,
  "expired": 0
```

また、`jq` ツールが使用可能な場合は、このツールを使用して特定の値を選択できます。以下の最初の 2 つの例は、`warning` または `expired` のいずれかの値を選択する方法を示しています。3 つ目の例は、両方の値を一度に選択する方法を示しています。

```
$ jq '.summary.warning' $HOME/cert-expiry-report.yyyymmddTHHMMSS.json
16
```

```
$ jq '.summary.expired' $HOME/cert-expiry-report.yyyymmddTHHMMSS.json
0
```

```
$ jq '.summary.warning,.summary.expired' $HOME/cert-expiry-report.yyyymmddTHHMMSS.json
16
0
```

### 12.3. 証明書の再デプロイ



#### 警告

Playbook を再デプロイメントすると、コントロールプレーンサービスの再起動に、クラスタのダウンタイムが発生する可能性があります。あるサービスでエラーが発生すると、Playbook が失敗し、クラスタの正常性に影響する可能性があります。Playbook が失敗した場合は、問題を手動で解決し、Playbook を再起動する必要がある場合があります。Playbook は、すべてのタスクを順次終了して成功する必要があります。

該当するすべてのホストにマスター、etcd、ノード、レジストリーまたはルーター証明書を再デプロイするには、以下の Playbook を使用します。現在の CA を使用してこれらすべての証明書を一度に再デプロイすることも、特定のコンポーネント用の証明書のみを再デプロイすることも、新しく生成された CA またはカスタム CA 自体を再デプロイすることもできます。

証明書の有効期限切れ Playbook と同様に、これらの Playbook はクラスターを表す [インベントリーファイル](#) を使用して実行する必要があります。

とくにインベントリーでは、すべてのホスト名と IP アドレスが現在のクラスター設定に一致するように、以下の変数でそれらを指定するか、または上書きする必要があります。

- `openshift_public_hostname`
- `openshift_public_ip`
- `openshift_master_cluster_hostname`
- `openshift_master_cluster_public_hostname`

以下のコマンドを実行すると、必要な Playbook が利用できるようになります。

```
# yum install openshift-ansible
```



#### 注記

再デプロイ中に自動生成された証明書の有効性 (有効期限が切れるまでの日数) も Ansible で設定できます。 [証明書の有効性の設定](#) を参照してください。



#### 注記

OpenShift Container Platform CA および etcd 証明書の有効期限は 5 年です。署名付きの OpenShift Container Platform 証明書の有効期限は 2 年です。

### 12.3.1. 現行の OpenShift Container Platform および etcd CA を使用したすべての証明書の再デプロイ

`redploy-certificates.yml` Playbook は、OpenShift Container Platform CA 証明書を再生成 **しません**。新しいマスター、etcd、ノード、レジストリー、およびルーター証明書は、新規の証明書に署名するために現在の CA 証明書を使用して作成されます。

これには、以下の順次の再起動も伴います。

- etcd
- マスターサービス
- ノードサービス

現行の OpenShift Container Platform CA を使用してマスター、etcd、およびノード証明書を再デプロイするには、Playbook に切り替え、この Playbook を実行し、インベントリーファイルを指定します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/redploy-certificates.yml
```



## 重要

OpenShift Container Platform CA が [openshift-master/redeploy-openshift-ca.yml Playbook](#) で再デプロイされた場合、`-e openshift_redeploy_openshift_ca=true` をこのコマンドに追加する必要があります。

### 12.3.2. 新規またはカスタムの OpenShift Container Platform CA の再デプロイ

`openshift-master/redeploy-openshift-ca.yml` Playbook は、新規の CA 証明書を生成し、クライアントの `kubeconfig` ファイルとノードの信頼できる CA のデータベース (CA-trust) を含むすべてのコンポーネントに更新されたバンドルを配布することによって OpenShift Container Platform CA 証明書を再デプロイします。

これには、以下の順次の再起動も伴います。

- マスターサービス
- ノードサービス
- docker

さらに、証明書を再デプロイする際には、OpenShift Container Platform によって生成される CA を使用する代わりに、[カスタム CA 証明書](#) を指定することもできます。

マスターサービスが再起動すると、レジストリーとルーターは、再デプロイされなくてもマスターと引き続き通信できます。これは、マスターの提供証明書が同一であり、レジストリーとルーターの CA が依然として有効であるためです。

新たに生成される CA またはカスタム CA を再デプロイするには、以下を実行します。

1. カスタム CA を使用する場合は、インベントリーファイルに以下の変数を設定します。現在の CA を使用する場合は、この手順を省略します。

```
# Configure custom ca certificate
# NOTE: CA certificate will not be replaced with existing clusters.
# This option may only be specified when creating a new cluster or
# when redeploying cluster certificates with the redeploy-certificates
# playbook.
openshift_master_ca_certificate={'certfile': '</path/to/ca.crt>', 'keyfile': '</path/to/ca.key>'}
```

CA 証明書が中間 CA によって発行された場合、CA が子の証明書を検証できるように、バンドルされた証明書に完全なチェーン (中間証明書とルート証明書) が含まれている必要があります。

以下に例を示します。

```
$ cat intermediate/certs/intermediate.cert.pem \
  certs/ca.cert.pem >> intermediate/certs/ca-chain.cert.pem
```

2. Playbook ディレクトリーに切り替え、ご自身のイベントリーファイルを指定して、`openshift-master/redeploy-openshift-ca.yml` Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/openshift-master/redeploy-openshift-ca.yml
```

新規の OpenShift Container Platform CA が導入されている場合、新規の CA によって署名された証明書をすべてのコンポーネントに再デプロイする必要がある場合にはいつでも [redeploy-certificates.yml](#) Playbook を使用できます。



### 重要

新規 OpenShift Container Platform CA が有効にされた後に [redeploy-certificates.yml](#) Playbook を使用する場合は、**-e openshift\_redeploy\_openshift\_ca=true** を Playbook コマンドに追加する必要があります。

### 12.3.3. 新規 etcd CA の再デプロイ

[openshift-etcd/redeploy-ca.yml](#) Playbook は、新規 CA 証明書を生成し、すべての etcd ピアとマスタークライアントに更新したバンドルを配布することによって etcd CA 証明書を再デプロイします。

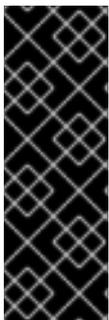
これには、以下の順次の再起動も伴います。

- etcd
- マスターサービス

新たに生成された etcd CA を再デプロイするには、以下を実行します。

1. [openshift-etcd/redeploy-ca.yml](#) Playbook を実行し、インベントリーファイルを指定します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/openshift-etcd/redeploy-ca.yml
```



### 重要

[playbooks/openshift-etcd/redeploy-ca.yml](#) Playbook の初回実行後に、CA 記号を含む圧縮バンドルは `/etc/etcd/etcd_ca.tgz` に永続化されます。CA 記号は新規 etcd 証明書の生成に必要です。そのため、それらがバックアップされていることが重要になります。

Playbook が再度実行されたときの予防措置として、ディスク上のこのバンドルが上書きされないようになっています。Playbook を再度実行するには、バンドルをこのパスからバックアップして移動してから Playbook を実行します。

新規 etcd CA が導入されている場合、新規 CA によって署名された証明書を etcd ピアとマスタークライアントに再デプロイする必要がある場合にはいつでも [openshift-etcd/redeploy-certificates.yml](#) Playbook を使用できます。または、[redeploy-certificates.yml](#) Playbook を使用して、etcd ピアとマスタークライアントに加えて、OpenShift Container Platform コンポーネントの証明書も再デプロイできます。



### 注記

etcd 証明書の再デプロイにより、**serial** がすべてのマスターホストにコピーされる可能性があります。

### 12.3.4. マスターおよび Web コンソール証明書の再デプロイ

`openshift-master/redeploy-certificates.yml` Playbook は、マスター証明書と Web コンソール証明書を再デプロイします。これには、マスターサービスの順次の再起動も伴います。

マスター証明書と Web コンソール証明書を再デプロイするには、Playbook ディレクトリーに移動し、インベントリーファイルを指定してこの Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/openshift-master/redeploy-certificates.yml
```

### 注記

名前付き証明書を使用する場合には、各マスターノードの `master-config.yml` ファイルで [証明書パラメーター](#) を更新します。必要な場合は、`certFile` パラメーターに指定された証明書ファイル用に、[証明書チェーン](#) の形成に必要なファイルをすべて連結します。

次に、OpenShift Container Platform マスターサービスを再起動して変更を適用します。

### 重要

この Playbook を実行した後に、サービス提供証明書を含む既存のシークレットを削除するか、またはアノテーションを削除し、適切なサービスに再度追加して、[サービス提供証明書またはキーペア](#) を再生成する必要があります。

必要に応じて、インベントリーファイルで `openshift_redeploy_service_signer=false` パラメーターを設定し、サービス署名証明書の再デプロイメントを省略できます。`openshift_redeploy_openshift_ca=true` および `openshift_redeploy_service_signer=true` をインベントリーファイルで設定すると、マスター証明書の再デプロイ時にサービス署名証明書が再デプロイされます。`openshift_redeploy_openshift_ca=false` を設定するか、パラメーターを省略すると、サービス署名証明書は再デプロイされません。

## 12.3.5. 名前付き証明書のための再デプロイ

`openshift-master/redeploy-named-certificates.yml` Playbook は、名前付き証明書のみを再デプロイします。この Playbook を実行すると、マスターサービスの順次の再起動も完了します。

名前付き証明書のみを再デプロイするには、Playbook が含まれるディレクトリーに切り替え、この Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/openshift-master/redeploy-named-certificates.yml
```

### 注記

Ansible インベントリーファイルの `_openshift_master_named_certificates_` パラメーターには、`master-config.yml` ファイルと同じ名前の証明書が含まれる必要があります。`certfile` および `keyfile` の名前が変更された場合には、各マスターノードの `master-config.yml` ファイルの [名前付き証明書パラメーター](#) を更新し、`api` と `controllers` サービスを再起動します。完全な ca チェーンが含まれる `cafile` が `/etc/origin/master/ca-bundle.crt` に追加されます。

## 12.3.6. etcd 証明書のみの再デプロイ

`openshift-etcd/redeploy-certificates.yml` Playbook は、マスタークライアント証明書を含む etcd 証明書のみを再デプロイします。

これには、以下の順次の再起動も伴います。

- etcd
- マスターサービス

etcd 証明書を再デプロイするには、Playbook ディレクトリーに切り替え、ご自身のイベントリーフファイルを指定してこの Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/openshift-etcd/redeploy-certificates.yml
```

### 12.3.7. ノード証明書の再デプロイ

デフォルトでは、ノード証明書の有効期限は1年間です。OpenShift Container Platform は、有効期限に近づくと、ノード証明書を自動的にローテーションします。[自動承認](#) が設定されていない場合には、[証明書署名要求 \(CSR\) を手動で承認する](#) 必要があります。

CA 証明書を変更したために証明書を再デプロイする必要がある場合には、`playbooks/redeploy-certificates.yml` Playbook に `-e openshift_redeploy_openshift_ca=true` フラグを立てて使用できます。詳細は、[現在の OpenShift Container Platform および etcd CA を使用したすべての証明書の再デプロイ](#) を参照してください。この Playbook の実行時に、CSR は自動的に承認されます。

### 12.3.8. レジストリー証明書またはルーター証明書のみ再デプロイ

`openshift-hosted/redeploy-registry-certificates.yml` Playbook と `openshift-hosted/redeploy-router-certificates.yml` Playbook は、インストーラーによって作成されたレジストリー証明書とルーター証明書を置き換えます。レジストリーとルーターでカスタム証明書が使用されている場合にそれらを手動で置き換えるには、[カスタムのレジストリー証明書またはルーター証明書の再デプロイ](#) を参照してください。

#### 12.3.8.1. レジストリー証明書のみ再デプロイ

レジストリー証明書を再デプロイするには、Playbook ディレクトリーに切り替え、ご自身のイベントリーフファイルを指定して以下の Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/openshift-hosted/redeploy-registry-certificates.yml
```

#### 12.3.8.2. ルーター証明書のみ再デプロイ

ルーター証明書を再デプロイするには、Playbook ディレクトリーに切り替え、ご自身のイベントリーフファイルを指定して以下の Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/openshift-hosted/redeploy-router-certificates.yml
```

### 12.3.9. カスタムのレジストリー証明書またはルーター証明書の再デプロイ

再デプロイされた CA が原因でノードが退避させられると、レジストリー Pod とルーター Pod が再起動されます。レジストリー証明書とルーター証明書を新規 CA と共に再デプロイしなかった場合は、それらが古い証明書を使用してマスターにアクセスできなくなるため、停止状態が生じることがあります。

#### 12.3.9.1. 手動によるレジストリー証明書の再デプロイ

レジストリー証明書を手動で再デプロイするには、新規レジストリー証明書を **registry-certificates** という名前のシークレットに追加してから、レジストリーを再デプロイする必要があります。

1. これ以降の手順では **default** プロジェクトに切り替えます。

```
$ oc project default
```

2. 最初にレジストリーを OpenShift Container Platform 3.1 以前で作成した場合は、環境変数が証明書を保存するために使用されている場合があります (この方法は現在は推奨されていません。代わりにシークレットをご使用ください)。

- a. 以下のコマンドを実行し、**OPENSIFT\_CA\_DATA**、**OPENSIFT\_CERT\_DATA**、および **OPENSIFT\_KEY\_DATA** 環境変数を探します。

```
$ oc set env dc/docker-registry --list
```

- b. これらの環境変数が存在しない場合は、この手順を省略します。存在する場合は、以下の **ClusterRoleBinding** を作成します。

```
$ cat <<EOF |
apiVersion: v1
groupNames: null
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: registry-registry-role
roleRef:
  kind: ClusterRole
  name: system:registry
subjects:
- kind: ServiceAccount
  name: registry
  namespace: default
userNames:
- system:serviceaccount:default:registry
EOF
oc create -f -
```

次に、以下のコマンドを実行して環境変数を削除します。

```
$ oc set env dc/docker-registry OPENSIFT_CA_DATA- OPENSIFT_CERT_DATA-
OPENSIFT_KEY_DATA- OPENSIFT_MASTER-
```

3. 以下の環境変数をローカルに設定し、後で使用するコマンドを単純化します。

```
$ REGISTRY_IP=`oc get service docker-registry -o jsonpath='{.spec.clusterIP}'`
$ REGISTRY_HOSTNAME=`oc get route/docker-registry -o jsonpath='{.spec.host}'`
```

4. 新規レジストリー証明書を作成します。

```
$ oc adm ca create-server-cert \
  --signer-cert=/etc/origin/master/ca.crt \
  --signer-key=/etc/origin/master/ca.key \
  --hostnames=$REGISTRY_IP,docker-registry.default.svc,docker-
registry.default.svc.cluster.local,$REGISTRY_HOSTNAME \
  --cert=/etc/origin/master/registry.crt \
  --key=/etc/origin/master/registry.key \
  --signer-serial=/etc/origin/master/ca.serial.txt
```

Ansible ホストインベントリーファイル (デフォルトで `/etc/ansible/hosts`) に最初に一覧表示されているマスターから **oc adm** コマンドを実行します。

5. **registry-certificates** シークレットを新規レジストリー証明書で更新します。

```
$ oc create secret generic registry-certificates \
  --from-file=/etc/origin/master/registry.crt,/etc/origin/master/registry.key \
  -o json --dry-run | oc replace -f -
```

6. レジストリーを再デプロイします。

```
$ oc rollout latest dc/docker-registry
```

### 12.3.9.2. 手動によるルーター証明書の再デプロイ

ルーター証明書を手動で再デプロイするには、新規ルーター証明書を **router-certs** という名前のシークレットに追加してから、ルーターを再デプロイする必要があります。

1. これ以降の手順では **default** プロジェクトに切り替えます。

```
$ oc project default
```

2. 最初にレジストリーを OpenShift Container Platform 3.1 以前で作成した場合は、環境変数が証明書を保存するために使用されている場合があります (この方法は現在は推奨されていません。代わりにサービス提供証明書シークレットをご使用ください)。

- a. 以下のコマンドを実行し、**OPENSIFT\_CA\_DATA**、**OPENSIFT\_CERT\_DATA**、および **OPENSIFT\_KEY\_DATA** 環境変数を探します。

```
$ oc set env dc/router --list
```

- b. それらの変数が存在する場合は、以下の **ClusterRoleBinding** を作成します。

```
$ cat <<EOF |
apiVersion: v1
groupNames: null
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
```

```

name: router-router-role
roleRef:
  kind: ClusterRole
  name: system:router
subjects:
- kind: ServiceAccount
  name: router
  namespace: default
userNames:
- system:serviceaccount:default:router
EOF
oc create -f -

```

- c. それらの変数が存在する場合は、以下のコマンドを実行してそれらを削除します。

```

$ oc set env dc/router OPENSIFT_CA_DATA- OPENSIFT_CERT_DATA-
OPENSIFT_KEY_DATA- OPENSIFT_MASTER-

```

### 3. 証明書を取得します。

- 外部の認証局 (CA) を使用して証明書に署名する場合、以下の内部プロセスに従って、新規の証明書を作成し、これを OpenShift Container Platform に指定します。
- 内部 OpenShift Container Platform CA を使用して証明書に署名する場合は、以下のコマンドを実行します。



#### 重要

以下のコマンドは、内部で署名される証明書を生成します。これは、OpenShift Container Platform CA を信頼するクライアントによってのみ信頼されます。

```

$ cd /root
$ mkdir cert ; cd cert
$ oc adm ca create-server-cert \
  --signer-cert=/etc/origin/master/ca.crt \
  --signer-key=/etc/origin/master/ca.key \
  --signer-serial=/etc/origin/master/ca.serial.txt \
  --hostnames='*.hostnames.for.the.certificate' \
  --cert=router.crt \
  --key=router.key \

```

これらのコマンドは以下のファイルを生成します。

- **router.crt** という名前の新規の証明書
  - 署名する CA 証明書チェーン **/etc/origin/master/ca.crt** のコピーです。このチェーンには中間の CA を使用する場合に複数の証明書が含まれる場合があります。
  - **router.key** という名前の対応するプライベートキー。
4. 生成された証明書を連結する新規ファイルを作成します。

```

$ cat router.crt /etc/origin/master/ca.crt router.key > router.pem

```



## 注記

この手順は、OpenShift CA が署名した証明書を使用している場合にのみ有効です。カスタム証明書を使用する場合は、`/etc/origin/master/ca.crt` の代わりに、正しい CA チェーンが含まれるファイルを使用する必要があります。

5. 新規シークレットを生成する前に、現在のシークレットをバックアップします。

```
$ oc get -o yaml --export secret router-certs > ~/old-router-certs-secret.yaml
```

6. 新規の証明書およびキーを保持する新規シークレットを作成し、既存のシークレットの内容を置き換えます。

```
$ oc create secret tls router-certs --cert=router.pem \ ❶
--key=router.key -o json --dry-run | \
oc replace -f -
```

- ❶ `router.pem` は、生成した証明書の連結を含むファイルです。

7. ルーターを再デプロイします。

```
$ oc rollout latest dc/router
```

ルーターの初回デプロイ時に、アノテーションが `router-metrics-tls` という名前の [サービス提供証明書シークレット](#) を自動的に作成するルーターのサービスに追加されます。

`router-metrics-tls` 証明書を手動で再デプロイするため、そのサービス提供証明書の再作成をトリガーできます。これは、シークレットを削除し、アノテーションを削除してからルーターサービスに再度追加し、`router-metrics-tls` シークレットを再デプロイして実行できます。

8. 以下のアノテーションを `router` サービスから削除します。

```
$ oc annotate service router \
service.alpha.openshift.io/serving-cert-secret-name- \
service.alpha.openshift.io/serving-cert-signed-by-
```

9. 既存の `router-metrics-tls` シークレットを削除します。

```
$ oc delete secret router-metrics-tls
```

10. アノテーションを再度追加します。

```
$ oc annotate service router \
service.alpha.openshift.io/serving-cert-secret-name=router-metrics-tls
```

## 12.4. 証明書署名要求の管理

クラスター管理者は、証明書署名要求 (CSR) を確認し、承認または拒否できます。

### 12.4.1. 証明書署名要求の確認

証明書署名要求 (CSR) の一覧を確認できます。

- 現在の CSR の一覧を取得します。

```
$ oc get csr
```

- CSR の詳細を表示して、これが有効であることを確認します。

```
$ oc describe csr <csr_name> ❶
```

❶ **<csr\_name>** は、現行の CSR の一覧からの CSR の名前です。

### 12.4.2. 証明書署名要求の承認

**oc certificate approve** コマンドを使用すると、証明書署名要求 (CSR) を手動で承認できます。

- CSR を承認します。

```
$ oc adm certificate approve <csr_name> ❶
```

❶ **<csr\_name>** は、現行の CSR の一覧からの CSR の名前です。

- 保留中の全 CSR を承認します。

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{"\n"}{{end}}\n{{end}}' | xargs oc adm certificate approve
```

### 12.4.3. 証明書署名要求の拒否

**oc certificate deny** コマンドを使用して、証明書署名要求 (CSR) を手動で拒否できます。

- CSR を拒否します。

```
$ oc adm certificate deny <csr_name> ❶
```

❶ **<csr\_name>** は、現行の CSR の一覧からの CSR の名前です。

### 12.4.4. 証明書署名要求の自動承認の設定

クラスターのインストール時に以下のパラメーターを Ansible インベントリーファイルに追加して、ノード証明書署名要求 (CSR) の自動承認を設定できます。

```
openshift_master_bootstrap_auto_approve=true
```

このパラメーターを追加すると、ブートストラップ認証情報を使用するか、またはホスト名が同じで、以前に認証されたノードから生成されたすべての CSR を、管理者の介入なしで承認できます。

詳細は、[クラスター変数の設定](#) を参照してください。

## 第13章 認証およびユーザーエージェントの設定

### 13.1. 概要

OpenShift Container Platform **マスター** には、ビルトイン **OAuth サーバー** が含まれます。開発者と管理者は、API に対する認証を実行するために **OAuth アクセストークン** を取得します。

管理者は **マスター設定ファイル** を使用して OAuth を **アイデンティティプロバイダー** を指定するように設定できます。アイデンティティプロバイダーは、**クラスターインストール** の実行中に設定するのが最適ですが、インストール後に設定することもできます。



#### 注記

、:、および % を含む OpenShift Container Platform ユーザー名はサポートされません。

**Deny All** アイデンティティプロバイダーがデフォルトで使用されます。このアイデンティティプロバイダーは、すべてのユーザー名とパスワードのアクセスを拒否します。アクセスを許可するには、別のアイデンティティプロバイダーを選択し、マスター設定ファイル (デフォルトでは、`/etc/origin/master/master-config.yaml` にあります) を適宜設定する必要があります。

設定ファイルなしでマスターを実行する場合は、**Allow All** アイデンティティプロバイダーがデフォルトで使用され、空でないユーザー名とパスワードによるログインをすべて許可します。これはテストを行う場合に便利です。その他のアイデンティティプロバイダーを使用するか、**token**、**grant** または **session** セッションを変更する場合は、設定ファイルからマスターを実行する必要があります。



#### 注記

外部ユーザーのセットアップを管理するための **ルール** が割り当てられている必要があります。

アイデンティティプロバイダーに変更を加えたら、変更を有効にするためにマスターサービスを再起動する必要があります。

```
# master-restart api
# master-restart controllers
```

### 13.2. アイデンティティプロバイダーパラメーター

すべてのアイデンティティプロバイダーには共通する 4 つのパラメーターがあります。

パラメーター	説明
<b>name</b>	プロバイダー名は、プロバイダーのユーザー名に接頭辞として付加され、アイデンティティ名が作成されます。

パラメーター	説明
<b>challenge</b>	<p><b>true</b> の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、<b>WWW-Authenticate</b> チャレンジ ヘッダー付きで送信されます。これはすべてのアイデンティティプロバイダーでサポートされる訳ではありません。</p> <p>ブラウザクライアントに対するクロスサイトリクエストフォージェリー (CSRF) 攻撃を防止するため、基本的な認証チャレンジは <b>X-CSRF-Token</b> ヘッダーが要求に存在する場合にのみ送信されます。基本的な <b>WWW-Authenticate</b> チャレンジを受信する必要があるクライアントでは、このヘッダーを空でない値に設定する必要があります。</p>
<b>login</b>	<p><b>true</b> の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。これはすべてのアイデンティティプロバイダーでサポートされる訳ではありません。</p> <p>ユーザーがアイデンティティプロバイダーのログインにリダイレクトされる前にブランドページに移動するようにする場合、マスター設定ファイルで <b>oauthConfig</b> → <b>alwaysShowProviderSelection: true</b> を設定します。このプロバイダー選択ページは <a href="#">カスタマイズ</a> できます。</p>
<b>mappingMethod</b>	<p>新規アイデンティティがログイン時にユーザーにマップされる方法を定義します。以下の値のいずれかを入力します。</p> <p><b>claim</b></p> <p>デフォルトの値です。アイデンティティの推奨ユーザー名を持つユーザーをプロビジョニングします。そのユーザー名を持つユーザーがすでに別のアイデンティティにマッピングされている場合は失敗します。</p> <p><b>lookup</b></p> <p>既存のアイデンティティ、ユーザーアイデンティティマッピング、およびユーザーを検索しますが、ユーザーまたはアイデンティティの自動プロビジョニングは行いません。これにより、クラスター管理者は手動で、または外部のプロセスを使用してアイデンティティとユーザーを設定できます。この方法を使用する場合は、ユーザーを手動でプロビジョニングする必要があります。 <a href="#">lookup マッピング方法を使用する場合のユーザーの手動プロビジョニング</a> を参照してください。</p> <p><b>generate</b></p> <p>アイデンティティの推奨ユーザー名を持つユーザーをプロビジョニングします。推奨ユーザー名を持つユーザーがすでに既存のアイデンティティにマッピングされている場合は、一意のユーザー名が生成されます。例: <b>myuser2</b>この方法は、OpenShift Container Platform のユーザー名とアイデンティティプロバイダーのユーザー名との正確な一致を必要とする外部プロセス (LDAP グループ同期など) と組み合わせて使用することはできません。</p> <p><b>add</b></p> <p>アイデンティティの推奨ユーザー名を持つユーザーをプロビジョニングします。推奨ユーザー名を持つユーザーがすでに存在する場合、アイデンティティは既存のユーザーにマッピングされ、そのユーザーの既存のアイデンティティマッピングに追加されます。これは、同じユーザーセットを識別して同じユーザー名にマッピングするアイデンティティプロバイダーが複数設定されている場合に必要です。</p>



## 注記

**mappingMethod** パラメーターを **add** に設定すると、アイデンティティプロバイダーの追加または変更時に新規プロバイダーのアイデンティティを既存ユーザーにマッピングできます。

### 13.3. アイデンティティプロバイダーの設定

OpenShift Container Platform は、同じアイデンティティプロバイダーを使用した複数の LDAP サーバーの設定をサポートしません。ただし、[LDAP フェイルオーバー](#) など、より複雑な設定の Basic 認証を拡張することが可能です。

これらのパラメーターを使用して、インストール時またはインストール後にアイデンティティプロバイダーを定義できます。

#### 13.3.1. Ansible を使用したアイデンティティプロバイダーの設定

初回クラスターインストールでは、[Deny All](#) アイデンティティプロバイダーが設定されますが、インベントリーファイルで **openshift\_master\_identity\_providers** パラメーターを設定して、これをインストール中に上書きできます。[OAuth 設定のセッションオプション](#) はインベントリーファイルで設定できます。

#### Ansible を使用したアイデンティティプロバイダー設定の例

```
# httpasswd auth
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge': 'true', 'kind':
'HTPasswdPasswordIdentityProvider'}]
# Defining htpasswd users
#openshift_master_htpasswd_users={'user1': '<pre-hashed password>', 'user2': '<pre-hashed
password>'}
# or
#openshift_master_htpasswd_file=/etc/origin/master/htpasswd

# Allow all auth
#openshift_master_identity_providers=[{'name': 'allow_all', 'login': 'true', 'challenge': 'true', 'kind':
'AllowAllPasswordIdentityProvider'}]

# LDAP auth
#openshift_master_identity_providers=[{'name': 'my_ldap_provider', 'challenge': 'true', 'login': 'true',
'kind': 'LDAPPasswordIdentityProvider', 'attributes': {'id': ['dn'], 'email': ['mail'], 'name': ['cn'],
'preferredUsername': ['uid']}, 'bindDN': '', 'bindPassword': '', 'insecure': 'false', 'url':
'ldap://ldap.example.com:389/ou=users,dc=example,dc=com?uid'}]
# Configuring the ldap ca certificate ❶
#openshift_master_ldap_ca=<ca text>
# or
#openshift_master_ldap_ca_file=<path to local ca file to use> ❷

# Available variables for configuring certificates for other identity providers:
#openshift_master_openshift_ca
#openshift_master_openshift_ca_file ❸
#openshift_master_request_header_ca
#openshift_master_request_header_ca_file ❹
```

- 1 LDAP アイデンティティプロバイダーについてのみ `openshift_master_identity_providers` パラメーターで `'insecure': 'true'` を指定している場合は、CA 証明書を省略できます。
- 2 3 4 Playbook を実行するホストにファイルを指定する場合、その内容は `/etc/origin/master/<identity_provider_name>_<identity_provider_type>_ca.crt` ファイルにコピーされます。アイデンティティプロバイダー名は `openshift_master_identity_providers` パラメーターの値であり、`ldap`、`openid`、または `request_header` になります。CA テキストまたはローカル CA ファイルのパスを指定しない場合は、CA 証明書をこの位置に配置する必要があります。複数のアイデンティティプロバイダーを指定する場合は、各プロバイダーの CA 証明書をこの位置に手動で配置する必要があります。この位置を変更することはできません。

複数のアイデンティティプロバイダーを指定することができます。これらを指定する場合は、各アイデンティティプロバイダーの CA 証明書を `/etc/origin/master/` ディレクトリーに配置する必要があります。たとえば、以下のプロバイダーを `openshift_master_identity_providers` 値に含めているとします。

```
openshift_master_identity_providers:
- name: foo
  provider:
    kind: OpenIDIdentityProvider
  ...
- name: bar
  provider:
    kind: OpenIDIdentityProvider
  ...
- name: baz
  provider:
    kind: RequestHeaderIdentityProvider
  ...
```

これらのアイデンティティプロバイダーの CA 証明書を以下のファイルに配置する必要があります。

- `/etc/origin/master/foo_openid_ca.crt`
- `/etc/origin/master/bar_openid_ca.crt`
- `/etc/origin/master/baz_requestheader_ca.crt`

### 13.3.2. マスター設定ファイルでのアイデンティティプロバイダーの設定

[マスター設定ファイル](#) を変更することで、必要なアイデンティティプロバイダーを使用してマスターホストで認証を設定できます。

#### 例13.1 マスター設定ファイルでのアイデンティティプロバイダーの設定例

```
...
oauthConfig:
  identityProviders:
  - name: htpasswd_auth
    challenge: true
    login: true
    mappingMethod: "claim"
  ...
```

デフォルトの **claim** 値に設定されている場合、アイデンティティを以前に存在していたユーザー名にマッピングすると OAuth が失敗します。

### 13.3.2.1. lookup マッピング方法を使用する場合のユーザーの手動プロビジョニング

**lookup** マッピング方法を使用する場合、ユーザープロビジョニングは外部システムによって API 経由で行われます。通常、アイデンティティは、ログイン時にユーザーに自動的にマッピングされます。lookup マッピング方法は、この自動マッピングを自動的に無効にします。そのため、ユーザーを手動でプロビジョニングする必要があります。

identity オブジェクトの詳細については、[Identity ユーザー API オブジェクト](#)を参照してください。

**lookup** マッピング方法を使用する場合は、アイデンティティプロバイダーを設定した後にユーザーごとに以下の手順を使用してください。

1. OpenShift Container Platform ユーザーを作成します (まだ作成していない場合)。

```
$ oc create user <username>
```

たとえば、以下のコマンドを実行して OpenShift Container Platform ユーザー **bob** を作成します。

```
$ oc create user bob
```

2. OpenShift Container Platform アイデンティティを作成します (まだ作成していない場合)。アイデンティティプロバイダーの名前と、アイデンティティプロバイダーの範囲でこのアイデンティティを一意に表す名前を使用します。

```
$ oc create identity <identity-provider>:<user-id-from-identity-provider>
```

**<identity-provider>** は、マスター設定のアイデンティティプロバイダーの名前であり、以下の該当するアイデンティティプロバイダーセクションに表示されています。

たとえば、以下のコマンドを実行すると、アイデンティティプロバイダーが **ldap\_provider**、アイデンティティプロバイダーのユーザー名が **bob\_s** のアイデンティティが作成されます。

```
$ oc create identity ldap_provider:bob_s
```

3. 作成したユーザーとアイデンティティのユーザー/アイデンティティマッピングを作成します。

```
$ oc create useridentitymapping <identity-provider>:<user-id-from-identity-provider>  
<username>
```

たとえば、以下のコマンドを実行すると、アイデンティティがユーザーにマッピングされます。

```
$ oc create useridentitymapping ldap_provider:bob_s bob
```

### 13.3.3. Allow All

空でないユーザー名とパスワードによるログインを許可するには、**identityProviders** スタンザに **AllowAllPasswordIdentityProvider** を設定します。

#### 例13.2 AllowAllPasswordIdentityProvider を使用したマスター設定

```
oauthConfig:
...
identityProviders:
- name: my_allow_provider ❶
  challenge: true ❷
  login: true ❸
  mappingMethod: claim ❹
  provider:
    apiVersion: v1
    kind: AllowAllPasswordIdentityProvider
```

- ❶ このプロバイダー名は、プロバイダーのユーザー名に接頭辞として付加され、アイデンティティ名が作成されます。
- ❷ **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。
- ❸ **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。
- ❹ このプロバイダーのアイデンティティとユーザーオブジェクト間のマッピングの確立方法を制御します ([上記](#) を参照してください)。

#### 13.3.4. Deny All

すべてのユーザー名とパスワードについてアクセスを拒否するには、**identityProviders** スタンザに **DenyAllPasswordIdentityProvider** を設定します。

#### 例13.3 DenyAllPasswordIdentityProvider を使用したマスター設定

```
oauthConfig:
...
identityProviders:
- name: my_deny_provider ❶
  challenge: true ❷
  login: true ❸
  mappingMethod: claim ❹
  provider:
    apiVersion: v1
    kind: DenyAllPasswordIdentityProvider
```

- ❶ このプロバイダー名は、プロバイダーのユーザー名に接頭辞として付加され、アイデンティティ名が作成されます。
- ❷ **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。

- 3 **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。
- 4 このプロバイダーのアイデンティティとユーザーオブジェクト間のマッピングの確立方法を制御します ([上記](#) を参照してください)。

### 13.3.5. HTTPasswd

ユーザー名およびパスワードを **htpasswd** を使用して生成されたフラットファイルに対して検証するには、**identityProviders** スタンザに **HTTPasswdPasswordIdentityProvider** を設定します。



#### 注記

**htpasswd** ユーティリティーは **httpd-tools** パッケージにあります。

```
# yum install httpd-tools
```

OpenShift Container Platform では、Bcrypt、SHA-1、および MD5 暗号化ハッシュ関数がサポートされ、MD5 は **htpasswd** のデフォルトです。プレーンテキスト、暗号化テキスト、およびその他のハッシュ関数は、現時点ではサポートされていません。

フラットファイルは、その変更時間が変わると再度読み取られます。サーバーの再起動は必要ありません。



#### 重要

OpenShift Container Platform マスター API は静的 Pod として実行されるため、**/etc/origin/master/** に **HTTPasswdPasswordIdentityProvider** **htpasswd** ファイルを作成し、これがコンテナによって読み取られるようにする必要があります。

**htpasswd** コマンドを使用するには、以下の手順を実行します。

- ユーザー名とハッシュされたパスワードを含むフラットファイルを作成するには、以下を実行します。

```
$ htpasswd -c /etc/origin/master/htpasswd <user_name>
```

次に、ユーザーのクリアテキストのパスワードを入力し、確認します。コマンドにより、ハッシュされたバージョンのパスワードが生成されます。

以下に例を示します。

```
htpasswd -c /etc/origin/master/htpasswd user1
New password:
Re-type new password:
Adding password for user user1
```



## 注記

**-b** オプションを追加すると、パスワードをコマンドラインに指定できます。

```
$ htpasswd -c -b <user_name> <password>
```

以下に例を示します。

```
$ htpasswd -c -b file user1 MyPassword!
Adding password for user user1
```

- ファイルにログインを追加するか、ログインを更新するには、以下のコマンドを実行します。

```
$ htpasswd /etc/origin/master/htpasswd <user_name>
```

- ファイルからログインを削除するには、以下のコマンドを実行します。

```
$ htpasswd -D /etc/origin/master/htpasswd <user_name>
```

### 例13.4 HTPasswdPasswordIdentityProvider を使用したマスター設定

```
oauthConfig:
...
identityProviders:
- name: my_htpasswd_provider ①
  challenge: true ②
  login: true ③
  mappingMethod: claim ④
  provider:
    apiVersion: v1
    kind: HTPasswdPasswordIdentityProvider
    file: /etc/origin/master/htpasswd ⑤
```

- ① このプロバイダー名は、プロバイダーのユーザー名に接頭辞として付加され、アイデンティティ名が作成されます。
- ② **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。
- ③ **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。
- ④ このプロバイダーのアイデンティティとユーザーオブジェクト間のマッピングの確立方法を制御します ([上記](#) を参照してください)。
- ⑤ **htpasswd** を使用して生成されたファイルです。

### 13.3.6. Keystone

Keystone は、アイデンティティ、トークン、カタログ、およびポリシーサービスを提供する

OpenStack プロジェクトです。OpenShift Container Platform クラスターと Keystone を統合すると、内部データベースにユーザーを格納するように設定された OpenStack Keystone v3 サーバーによる共有認証を有効にできます。この設定により、ユーザーは Keystone 認証情報を使って OpenShift Container Platform にログインできます。

新規 OpenShift Container Platform ユーザーが Keystone ユーザー名または一意の Keystone ID をベースに設定されるように Keystone との統合を設定できます。どちらの方法でも、ユーザーは Keystone ユーザー名およびパスワードを入力してログインします。OpenShift Container Platform ユーザーのベースを Keystone ID としない方法がより安全な方法になります。Keystone ユーザーを削除し、そのユーザー名で新規の Keystone ユーザーを作成する場合、新規ユーザーが古いユーザーのリソースにアクセスできる可能性があるためです。

### 13.3.6.1. マスターでの認証の設定

1. 状況に応じて以下のいずれかの手順を実行します。

- OpenShift のインストールがすでに完了している場合は、`/etc/origin/master/master-config.yaml` ファイルを新規ディレクトリーにコピーします。以下は例になります。

```
$ cd /etc/origin/master
$ mkdir keystoneconfig; cp master-config.yaml keystoneconfig
```

- OpenShift Container Platform をまだインストールしていない場合は、OpenShift Container Platform API サーバーを起動し、(将来の) OpenShift Container Platform マスターのホスト名と、起動コマンドによって作成された設定ファイルを格納するディレクトリーを指定します。

```
$ openshift start master --public-master=<apiserver> --write-config=<directory>
```

以下に例を示します。

```
$ openshift start master --public-master=https://myapiserver.com:8443 --write-config=keystoneconfig
```



#### 注記

Ansible を使用してインストールする場合は、**identityProvider** 設定を Ansible Playbook に追加する必要があります。Ansible を使用してインストールした後、以下の手順に従って設定を手動で変更した場合、インストールツールまたはアップグレードを再実行するたびに変更内容がすべて失われます。

2. 新規の `keystoneconfig/master-config.yaml` ファイルの **identityProviders** スタンザを編集し、**KeystonePasswordIdentityProvider** の設定例をコピーして貼り付け、既存のスタンザを置き換えます。

```
oauthConfig:
  ...
  identityProviders:
  - name: my_keystone_provider ①
    challenge: true ②
    login: true ③
    mappingMethod: claim ④
```

```

provider:
  apiVersion: v1
  kind: KeystonePasswordIdentityProvider
  domainName: default ⑤
  url: http://keystone.example.com:5000 ⑥
  ca: ca.pem ⑦
  certFile: keystone.pem ⑧
  keyFile: keystonekey.pem ⑨
  useKeystoneIdentity: false ⑩

```

- ① このプロバイダー名は、プロバイダーのユーザー名に接頭辞として付加され、アイデンティティ名が作成されます。
- ② **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。
- ③ **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。
- ④ このプロバイダーのアイデンティティとユーザーオブジェクト間のマッピングの確立方法を制御します ([上記](#) を参照してください)。
- ⑤ Keystone のドメイン名です。Keystone では、ユーザー名はドメインに固有の名前です。単一ドメインのみがサポートされます。
- ⑥ Keystone サーバーへの接続に使用する URL です (必須)。
- ⑦ オプション: 設定された URL のサーバー証明書を検証するために使用する証明書バンドルです。
- ⑧ オプション: 設定された URL に対して要求を実行する際に提示するクライアント証明書です。
- ⑨ クライアント証明書のキーです。certFile が指定されている場合は必須です。
- ⑩ **true** の場合、ユーザーが Keystone ユーザー名ではなく、Keystone ID で認証されていることを示します。ユーザー名で認証する場合は **false** に設定されます。

### 3. 以下の変更を **identityProviders** スタンザに加えます。

- a. プロバイダーの **name** (my\_keystone\_provider) を、使用する Keystone サーバーに合わせて変更します。この名前は、プロバイダーのユーザー名に接頭辞として付加され、アイデンティティ名が作成されます。
- b. 必要な場合、**mappingMethod** を変更して、プロバイダーのアイデンティティとユーザーオブジェクト間でマッピングを確立する方法を制御します。
- c. **domainName** を OpenStack Keystone サーバーのドメイン名に変更します。Keystone では、ユーザー名はドメイン固有です。単一ドメインのみがサポートされます。
- d. OpenStack Keystone への接続に使用する **url** を指定します。
- e. オプションとして、Keystone ユーザー名ではなく Keystone ID でユーザーを認証するには、**useKeystoneIdentity** を **true** に設定します。

- f. オプションで、設定された URL のサーバー証明書を検証できるように **ca** を使用する証明書バンドルに変更します。
  - g. オプションで、**certFile** を、設定された URL に対する要求の実行時に提示するクライアント証明書に変更します。
  - h. **certFile** が指定されている場合は、**keyFile** をクライアント証明書のキーに変更する必要があります。
4. 変更を保存してファイルを閉じます。
  5. OpenShift Container Platform API サーバーを起動し、変更したばかりの設定ファイルを指定します。

```
$ openshift start master --config=<path/to/modified/config>/master-config.yaml
```

設定が完了すると、OpenShift Container Platform Web コンソールにログインするすべてのユーザーに Keystone 認証情報を使用してログインすることを求めるプロンプトが出されます。

### 13.3.6.2. Keystone 認証を使用するユーザーの作成

外部認証プロバイダー (この場合は Keystone) と統合する場合、OpenShift Container Platform にはユーザーを作成しません。Keystone は system of record であり、ユーザーは Keystone データベースで定義され、設定された認証サーバーに対する有効な Keystone ユーザー名を持つ任意のユーザーがログインできます。

ユーザーを OpenShift Container Platform に追加するには、そのユーザーが Keystone データベースに存在する必要があります。また、必要な場合はそのユーザーの新しい Keystone アカウントを作成する必要があります。

### 13.3.6.3. ユーザーの確認

1名以上のユーザーがログインしたら、**oc get users** を実行してユーザーの一覧を表示し、ユーザーが正しく作成されていることを確認できます。

#### 例13.5 oc get users コマンドの出力

```
$ oc get users
NAME      UID                                FULL NAME  IDENTITIES
bobsmith  a0c1d95c-1cb5-11e6-a04a-002186a28631  Bob Smith  keystone:bobsmith ①
```

- ① OpenShift Container Platform のアイデンティティは、Keystone ユーザー名とそれに接頭辞として付加されるアイデンティティプロバイダー名で設定されます。

ここからは、[ユーザーロールの管理](#) 方法を学習することをお勧めします。

### 13.3.7. LDAP 認証

単純なバインド認証を使用してユーザー名とパスワードを LDAPv3 サーバーに対して検証するには、**identityProviders** スタンザに **LDAPPasswordIdentityProvider** を設定します。



## 注記

これらの手順に従うのではなく、LDAP サーバーのフェイルオーバーを要求する場合には、[SSSD で LDAP フェイルオーバーを設定](#)して Basic 認証の方法を拡張します。

認証時に、指定されたユーザー名に一致するエントリが LDAP ディレクトリーで検索されます。単一の一意の一致が見つかった場合、エントリーの識別名 (DN) と指定されたパスワードを使用した単純なバインドが試みられます。

以下の手順が実行されます。

1. 設定された **url** の属性およびフィルターとユーザーが指定したユーザー名を組み合わせで検索フィルターを生成します。
2. 生成されたフィルターを使用してディレクトリーを検索します。検索によって1つもエントリが返されない場合は、アクセスを拒否します。
3. 検索で取得したエントリーの DN とユーザー指定のパスワードを使用して LDAP サーバーへのバインドを試みます。
4. バインドが失敗した場合は、アクセスを拒否します。
5. バインドが成功した場合は、アイデンティティー、電子メールアドレス、表示名、および推奨ユーザー名として設定された属性を使用してアイデンティティーを作成します。

設定される **url** は、LDAP ホストと使用する検索パラメーターを指定する RFC 2255 URL です。URL の構文は以下のようになります。

```
ldap://host:port/basedn?attribute?scope?filter
```

上記の例は、以下のコンポーネントで設定されています。

URL コンポーネント	説明
<b>ldap</b>	通常の LDAP の場合は、文字列 <b>ldap</b> を使用します。セキュアな LDAP (LDAPS) の場合は、代わりに <b>ldaps</b> を使用します。
<b>host:port</b>	LDAP サーバーの名前とポートです。デフォルトは、ldap の場合は <b>localhost:389</b> 、LDAPS の場合は <b>localhost:636</b> です。
<b>basedn</b>	すべての検索が開始されるディレクトリーのブランチの DN です。これは少なくともディレクトリーツリーの最上位になければなりません、ディレクトリーのサブツリーを指定することもできます。
<b>attribute</b>	検索対象の属性です。RFC 2255 はコンマ区切りの属性の一覧を許可しますが、属性をどれだけ指定しても最初の属性のみが使用されます。属性を指定しない場合は、デフォルトで <b>uid</b> が使用されます。使用しているサブツリーのすべてのエントリー間で一意の属性を選択することを推奨します。
<b>scope</b>	検索の範囲です。 <b>one</b> または <b>sub</b> のいずれかを指定できます。範囲を指定しない場合は、デフォルトの範囲として <b>sub</b> が使用されます。

URL コンポーネント	説明
<b>filter</b>	有効な LDAP 検索フィルターです。指定しない場合、デフォルトは <b>(objectClass=*)</b> です。

検索の実行時に属性、フィルター、指定したユーザー名が組み合わされて以下のような検索フィルターが作成されます。

```
(<filter>(<attribute>=<username>))
```

たとえば、以下の URL について見てみましょう。

```
ldap://ldap.example.com/o=Acme?cn?sub?(enabled=true)
```

クライアントが **bob** というユーザー名を使用して接続を試みる場合、生成される検索フィルターは **(&(enabled=true)(cn=bob))** になります。

LDAP ディレクトリーの検索に認証が必要な場合は、エントリー検索の実行に使用する **bindDN** と **bindPassword** を指定します。

### LDAPPasswordIdentityProvider を使用したマスター設定

```
oauthConfig:
  ...
  identityProviders:
  - name: "my_ldap_provider" 1
    challenge: true 2
    login: true 3
    mappingMethod: claim 4
    provider:
      apiVersion: v1
      kind: LDAPPasswordIdentityProvider
      attributes:
        id: 5
        - dn
        email: 6
        - mail
        name: 7
        - cn
        preferredUsername: 8
        - uid
      bindDN: "" 9
      bindPassword: "" 10
      ca: my-ldap-ca-bundle.crt 11
      insecure: false 12
      url: "ldap://ldap.example.com/ou=users,dc=acme,dc=com?uid" 13
```

- 1 このプロバイダー名は返されるユーザー名に接頭辞として付加され、アイデンティティ名が作成されます。

- 2 **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。
- 3 **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。
- 4 このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します ([上記](#) を参照してください)。
- 5 アイデンティティーとして使用する属性の一覧です。最初の空でない属性が使用されます。少なくとも1つの属性が必要です。一覧表示される属性のいずれにも値がない場合、認証は失敗します。
- 6 メールアドレスとして使用する属性の一覧です。最初の空でない属性が使用されます。
- 7 表示名として使用する属性の一覧です。最初の空でない属性が使用されます。
- 8 このアイデンティティーのユーザーをプロビジョニングする際に推奨ユーザー名として使用する属性の一覧です。最初の空でない属性が使用されます。
- 9 検索フェーズでバインドするために使用するオプションの DN です。
- 10 検索フェーズでバインドするために使用するオプションのパスワードです。この値は [環境変数](#)、[外部ファイル](#)、または[暗号化されたファイル](#)でも指定できます。
- 11 設定された URL のサーバー証明書を検証するために使用する証明書バンドルです。このファイルのコンテンツは、`/etc/origin/master/<identity_provider_name>_ldap_ca.crt` ファイルにコピーされます。アイデンティティープロバイダー名は `openshift_master_identity_providers` パラメーターの値です。CA テキストまたはローカル CA ファイルのパスを指定しない場合は、CA 証明書を `/etc/origin/master/` ディレクトリーに配置する必要があります。複数のアイデンティティープロバイダーを指定する場合は、各プロバイダーの CA 証明書を `/etc/origin/master/` ディレクトリーに手動で配置する必要があります。この位置を変更することはできません。証明書バンドルの定義は、`insecure: false` がインベントリーファイルに設定されている場合にのみ適用されます。
- 12 **true** の場合、サーバーへの TLS 接続は行われません。**false** の場合、`ldaps://URL` は TLS を使用して接続し、`ldap://URL` は TLS にアップグレードされます。
- 13 LDAP ホストと使用する検索パラメーターを指定する RFC 2255 URL です ([上記](#) を参照してください)。



### 注記

LDAP 統合のためのユーザーのホワイトリストを作成するには、**lookup** マッピング方法を使用します。LDAP からのログインが許可される前に、クラスター管理者は各 LDAP ユーザーのアイデンティティーとユーザーオブジェクトを作成する必要があります。

### 13.3.8. Basic 認証 (リモート)

Basic 認証は、ユーザーがリモートのアイデンティティープロバイダーに対して検証した認証情報を使用して OpenShift Container Platform にログインすることを可能にする汎用バックエンド統合メカニズムです。

Basic 認証は汎用性があるため、このアイデンティティープロバイダーを使用して詳細な認証設定を実行できます。詳細なリモート Basic 認証設定の開始点として、[LDAP フェイルオーバー](#) を設定したり、[コンテナ化された Basic 認証](#) リポジトリーを使用できます。

## 注意

Basic 認証では、ユーザー ID とパスワードのスヌーピングを防ぎ、中間者攻撃を回避するためにリモートサーバーへの HTTPS 接続を使用する必要があります。

**BasicAuthPasswordIdentityProvider** を設定していると、ユーザーはユーザー名とパスワードを OpenShift Container Platform に送信し、サーバー間の要求を行い、認証情報を Basic 認証ヘッダーとして渡すことで、これらの認証情報をリモートサーバーに対して検証することができます。このため、ユーザーはログイン時に認証情報を OpenShift Container Platform に送信する必要があります。



### 注記

これはユーザー名/パスワードログインの仕組みにのみ有効で、OpenShift Container Platform はリモート認証サーバーに対するネットワーク要求を実行できる必要があります。

**identityProviders** スタンザで **BasicAuthPasswordIdentityProvider** を設定し、サーバー間の Basic 認証要求を使用してユーザー名とパスワードをリモートサーバーに対して検証できるようにします。ユーザー名とパスワードは Basic 認証で保護されるリモート URL に対して検証され、JSON を返します。

**401** 応答は認証の失敗を示しています。

**200** 以外のステータスまたは空でないエラーキーはエラーを示しています。

```
{"error": "Error message"}
```

**sub** (サブジェクト) キーを持つ **200** ステータスは、成功を示しています。

```
{"sub": "userid"} 1
```

**1** このサブジェクトは認証ユーザーに固有である必要があります、変更することができません。

正常な応答により、以下のような追加データがオプションで提供されることがあります。

- **name** キーを使用した表示名。以下に例を示します。

```
{"sub": "userid", "name": "User Name", ...}
```

- **email** キーを使用したメールアドレス。以下に例を示します。

```
{"sub": "userid", "email": "user@example.com", ...}
```

- **preferred\_username** キーを使用した推奨ユーザー名。これは、固有の変更できないサブジェクトがデータベースキーまたは UID であり、判読可能な名前が存在する場合に便利です。これは、認証されたアイデンティティの OpenShift Container Platform ユーザーをプロビジョニングする場合にヒントとして使われます。以下に例を示します。

```
{"sub": "014fbff9a07c", "preferred_username": "bob", ...}
```

### 13.3.8.1. マスターでの認証の設定

1. 状況に応じて以下のいずれかの手順を実行します。

- OpenShift のインストールがすでに完了している場合は、`/etc/origin/master/master-config.yaml` ファイルを新規ディレクトリーにコピーします。以下は例になります。

```
$ mkdir basicauthconfig; cp master-config.yaml basicauthconfig
```

- OpenShift Container Platform をまだインストールしていない場合は、OpenShift Container Platform API サーバーを起動し、(将来の) OpenShift Container Platform マスターのホスト名と、起動コマンドによって作成された設定ファイルを格納するディレクトリーを指定します。

```
$ openshift start master --public-master=<apiserver> --write-config=<directory>
```

以下に例を示します。

```
$ openshift start master --public-master=https://myapiserver.com:8443 --write-config=basicauthconfig
```



### 注記

Ansible を使用してインストールする場合は、**identityProvider** 設定を Ansible Playbook に追加する必要があります。Ansible を使用してインストールした後、以下の手順に従って設定を手動で変更した場合、インストールツールまたはアップグレードを再実行するたびに変更内容がすべて失われます。

2. 新規の `master-config.yaml` ファイルの **identityProviders** スタンザを編集し、[BasicAuthPasswordIdentityProvider 設定のサンプル](#) をコピーして貼り付け、既存のスタンザを置き換えます。

```
oauthConfig:
  ...
  identityProviders:
  - name: my_remote_basic_auth_provider ❶
    challenge: true ❷
    login: true ❸
    mappingMethod: claim ❹
    provider:
      apiVersion: v1
      kind: BasicAuthPasswordIdentityProvider
      url: https://www.example.com/remote-idp ❺
      ca: /path/to/ca.file ❻
      certFile: /path/to/client.crt ❼
      keyFile: /path/to/client.key ❽
```

- ❶ このプロバイダー名は返されるユーザー名に接頭辞として付加され、アイデンティティ名が作成されます。
- ❷ **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。
- ❸ **true** の場合、Web クライアント (Web コンソールなど) からの認証されていないトークン要求は、このプロバイダーがサポートするログインページにリダイレクトされます。

- 4 このプロバイダーのアイデンティティとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。
- 5 Basic 認証ヘッダーで認証情報を受け入れる URL。
- 6 オプション: 設定された URL のサーバー証明書を検証するために使用する証明書バンドルです。
- 7 オプション: 設定された URL に対して要求を実行する際に提示するクライアント証明書です。
- 8 クライアント証明書のキーです。 **certFile** が指定されている場合は必須です。

以下の変更を **identityProviders** スタンザに加えます。

- a. プロバイダーの **name** をデプロイメントに対して固有で関連するものに設定します。この名前は返されるユーザー ID に接頭辞として付加され、アイデンティティ名が作成されません。
  - b. 必要な場合、 **mappingMethod** を設定して、プロバイダーのアイデンティティとユーザーオブジェクト間でマッピングを確立する方法を制御します。
  - c. Basic 認証ヘッダーで認証情報を受け入れるサーバーへの接続に使用する HTTPS **url** を指定します。
  - d. オプションで、設定された URL のサーバー証明書を検証するために **ca** を使用する証明書バンドルに設定するか、またはこれを空にしてシステムで信頼されるルートを使用します。
  - e. オプションで、 **certFile** を削除するか、またはこれを設定された URL へ要求を行う時に提示するクライアント証明書に設定します。
  - f. **certFile** を指定する場合、 **keyFile** をクライアント証明書のキーに設定する必要があります。
3. 変更を保存してファイルを閉じます。
  4. OpenShift Container Platform API サーバーを起動し、変更したばかりの設定ファイルを指定します。

```
$ openshift start master --config=<path/to/modified/config>/master-config.yaml
```

これが設定されると、OpenShift Container Platform Web コンソールにログインするユーザーには、Basic 認証の認証情報を使用してログインすることを求めるプロンプトが表示されます。

### 13.3.8.2. トラブルシューティング

最もよく起こる問題は、バックエンドサーバーへのネットワーク接続に関連しています。簡単なデバッグの場合は、マスターで **curl** コマンドを実行します。正常なログインをテストするには、以下のコマンド例の **<user>** と **<password>** を有効な認証情報に置き換えます。無効なログインをテストするには、それらを正しくない認証情報に置き換えます。

```
curl --cacert /path/to/ca.crt --cert /path/to/client.crt --key /path/to/client.key -u <user>:<password> -v https://www.example.com/remote-idp
```

## 正常な応答

**sub** (サブジェクト) キーを持つ **200** ステータスは、成功を示しています。

```
{"sub": "userid"}
```

サブジェクトは認証ユーザーに固有である必要があります、変更することはできません。

正常な応答により、以下のような追加データがオプションで提供されることがあります。

- **name** キーを使用した表示名:

```
{"sub": "userid", "name": "User Name", ...}
```

- **email** キーを使用したメールアドレス:

```
{"sub": "userid", "email": "user@example.com", ...}
```

- **preferred\_username** キーを使用した推奨ユーザー名:

```
{"sub": "014fbff9a07c", "preferred_username": "bob", ...}
```

**preferred\_username** キーは、固有の変更できないサブジェクトがデータベースキーまたは UID であり、判読可能な名前が存在する場合に便利です。これは、認証されたアイデンティティーの OpenShift Container Platform ユーザーをプロビジョニングする場合にヒントとして使われます。

## 失敗の応答

- **401** 応答は認証の失敗を示しています。
- **200** 以外のステータスまたは空でないエラーキーはエラーを示しています: `{"error": "Error message"}`

### 13.3.9. 要求ヘッダー

**identityProviders** スタンザで **RequestHeaderIdentityProvider** を設定して、**X-Remote-User** などの要求ヘッダー値からユーザーを識別します。これは通常、プロキシ認証と組み合わせて使用され、要求ヘッダー値を設定します。これは [OpenShift Enterprise 2 のリモートユーザープラグイン](#) によって管理者が Kerberos、LDAP、その他の数多くの形式のエンタープライズ認証を指定する方法と似ています。



#### 注記

さらに、コミュニティでサポートされる [SAML 認証](#) などの詳細な設定に要求ヘッダーアイデンティティープロバイダーを使用できます。SAML 認証は Red Hat ではサポートされていないことに注意してください。

ユーザーがこのアイデンティティープロバイダーを使用して認証を行うには、認証プロキシ経由で **https://<master>/oauth/authorize** (およびサブパス) にアクセスする必要があります。これを実行するには、OAuth トークンに対する非認証の要求を **https://<master>/oauth/authorize** にプロキシ処理するプロキシエンドポイントにリダイレクトするよう OAuth サーバーを設定します。

ブラウザーベースのログインフローが想定されるクライアントからの非認証要求をリダイレクトするには、以下を実行します。

1. **login** パラメーターを **true** に設定します。
2. **provider.loginURL** パラメーターをインタラクティブなクライアントを認証する認証プロキシ URL に設定してから、要求を **https://<master>/oauth/authorize** にプロキシします。

**WWW-Authenticate** チャレンジが想定されるクライアントからの非認証要求をリダイレクトするには、以下を実行します。

1. **challenge** パラメーターを **true** に設定します。
2. **provider.challengeURL** パラメーターを **WWW-Authenticate** チャレンジが予想されるクライアントを認証する認証プロキシ URL に設定し、要求を **https://<master>/oauth/authorize** にプロキシします。

**provider.challengeURL** および **provider.loginURL** パラメーターには、URL のクエリー部分に以下のトークンを含めることができます。

- **\${url}** は現在の URL と置き換えられ、エスケープされてクエリーパラメーターで保護されません。  
例: **https://www.example.com/sso-login?then=\${url}**
- **\${query}** は最新のクエリー文字列と置き換えられ、エスケープされません。  
例: **https://www.example.com/auth-proxy/oauth/authorize?\${query}**



#### 警告

非認証要求が OAuth サーバーに到達することを想定する場合は、要求ヘッダーのユーザー名がチェックされる前に受信要求の有効なクライアント証明書をチェックするように、**clientCA** パラメーターをこのアイデンティティプロバイダーに対して設定する必要があります。これを設定しない場合、OAuth サーバーへの直接的な要求は、要求ヘッダーを設定するだけでこのプロバイダーのアイデンティティになりすます可能性があります。

## RequestHeaderIdentityProvider を使用したマスター設定

```

oauthConfig:
...
identityProviders:
- name: my_request_header_provider ❶
  challenge: true ❷
  login: true ❸
  mappingMethod: claim ❹
  provider:
    apiVersion: v1
    kind: RequestHeaderIdentityProvider
    challengeURL: "https://www.example.com/challenging-proxy/oauth/authorize?${query}" ❺
    loginURL: "https://www.example.com/login-proxy/oauth/authorize?${query}" ❻

```

```

clientCA: /path/to/client-ca.file 7
clientCommonNames: 8
- my-auth-proxy
headers: 9
- X-Remote-User
- SSO-User
emailHeaders: 10
- X-Remote-User-Email
nameHeaders: 11
- X-Remote-User-Display-Name
preferredUsernameHeaders: 12
- X-Remote-User-Login

```

- 1 このプロバイダー名は要求ヘッダーのユーザー名に接頭辞として付加され、アイデンティティー名が作成されます。
- 2 **RequestHeaderIdentityProvider** は、設定された **challengeURL** にリダイレクトすることで、**WWW-Authenticate** チャレンジを要求するクライアントに応答します。設定された URL は **WWW-Authenticate** チャレンジを使用して応答します。
- 3 **RequestHeaderIdentityProvider** は、設定された **loginURL** にリダイレクトすることで、ログインフローを要求するクライアントにのみ応答できます。設定される URL はログインフローを使用して応答します。
- 4 このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します ([上記](#) を参照してください)。
- 5 オプション: 非認証の **/oauth/authorize** 要求のリダイレクト先となる URL です。これにより、**WWW-Authenticate** チャレンジが予想されるクライアントの認証が行われ、それらの要求が **https://<master>/oauth/authorize** にプロキシされます。**#{url}** は現在の URL と置き換えられ、エスケープされてクエリーパラメーターで保護されます。**#{query}** は最新のクエリー文字列と置き換えられます。
- 6 オプション: 非認証の **/oauth/authorize** 要求のリダイレクト先となる URL です。これは、ブラウザーベースのクライアントを認証してから、その要求を **https://<master>/oauth/authorize** にプロキシ化します。**https://<master>/oauth/authorize** にプロキシする URL は **/authorize** (末尾にスラッシュはない) で終了し、OAuth 承認フローが適切に機能するようにサブパスもプロキシする必要があります。**#{url}** は現在の URL と置き換えられ、エスケープされてクエリーパラメーターで保護されます。**#{query}** は最新のクエリー文字列と置き換えられます。
- 7 オプション: PEM でエンコードされた証明書バンドルです。これが設定されている場合、要求ヘッダーのユーザー名をチェックする前に、有効なクライアント証明書が提示され、指定ファイルで認証局に対して検証される必要があります。
- 8 オプション: 共通名 (cn) の一覧。これが設定されている場合は、要求ヘッダーのユーザー名をチェックする前に指定される一覧の Common Name (cn) を持つ有効なクライアント証明書が提示される必要があります。空の場合、すべての Common Name が許可されます。これは **clientCA** との組み合わせる場合にのみ使用できます。
- 9 ユーザーアイデンティティーを順番にチェックする際に使用するヘッダー名。値を含む最初のヘッダーはアイデンティティーとして使用されます。これは必須であり、大文字小文字を区別します。
- 10 メールアドレスを順番にチェックする際に使用するヘッダー名。値を含む最初のヘッダーはメールアドレスとして使用されます。これは任意であり、大文字小文字を区別します。
- 11 表示名を順番にチェックする際に使用するヘッダー名。値を含む最初のヘッダーは表示名として使

用されます。これは任意であり、大文字小文字を区別します。

- 12** 推奨ユーザー名を順番にチェックする際に使用するヘッダー名 (**headers** に指定されるヘッダーで決定される変更不可のアイデンティティと異なる場合)。値を含む最初のヘッダーは、プロビジョニング時に推奨ユーザー名として使用されます。これは任意であり、大文字小文字を区別しません。

## Microsoft Windows での SSPI 接続サポート



### 重要

Microsoft Windows での SSPI 接続サポートの使用はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、<https://access.redhat.com/support/offerings/techpreview/> を参照してください。

### バージョン 3.11 以降

**oc** は、Microsoft Windows での SSO フローを許可するために Security Support Provider Interface (SSPI) をサポートします。要求ヘッダーのアイデンティティプロバイダーを GSSAPI 対応プロキシと共に使用して Active Directory サーバーを OpenShift Container Platform に接続する場合、ユーザーは、ドメイン参加済みの Microsoft Windows コンピューターから **oc** コマンドラインインターフェイスを使用して OpenShift Container Platform に対して自動的に認証されます。

### 要求ヘッダーを使用した Apache 認証

この例は、マスターと同じホストに認証プロキシを設定しています。同じホストにプロキシとマスターがあると便利ですが、ご使用中の環境に適さない場合があります。たとえば、すでにマスターで **ルーターを実行** している場合、ポート 443 が利用できなくなります。

この参照設定は Apache の **mod\_auth\_gssapi** を使用していますが、これは決して必須ではなく、以下の要件を満たしていれば他のプロキシを簡単に使用することができます。

1. クライアント要求の **X-Remote-User** ヘッダーをブロックして、スプーフィングを防ぎます。
2. **RequestHeaderIdentityProvider** 設定でクライアント証明書の認証を適用します。
3. チャレンジフローを使用してすべての認証要求についての **X-Csrf-Token** ヘッダーを設定する必要があります。
4. **/oauth/authorize** エンドポイントとそのサブパスのみがプロキシされる必要があります。バックエンドサーバーがクライアントを正しい場所へ送信できるようにリダイレクトは書き換えないでください。
5. **https://<master>/oauth/authorize** ヘプロキシする URL は **/authorize** で終了 (末尾のスラッシュなし) する必要があります。以下に例を示します。
  - **https://proxy.example.com/login-proxy/authorize?... → https://<master>/oauth/authorize?...**

6. `https://<master>/oauth/authorize` にプロキシされる URL のサブパスは、`https://<master>/oauth/authorize` のサブパスにプロキシする必要があります。以下に例を示します。
- `https://proxy.example.com/login-proxy/authorize/approve?... → https://<master>/oauth/authorize/approve?...`

### 前提条件のインストール

1. `mod_auth_gssapi` モジュールを [Optional チャンネル](#) から取得します。以下のパッケージをインストールします。

```
# yum install -y httpd mod_ssl mod_session apr-util-openssl mod_auth_gssapi
```

2. 信頼されたヘッダーを送信する要求を検証するために CA を生成します。この CA は **マスターのアイデンティティプロバイダーの設定** の `clientCA` のファイル名として使用されます。

```
# oc adm ca create-signer-cert \
  --cert='/etc/origin/master/proxyca.crt' \
  --key='/etc/origin/master/proxyca.key' \
  --name='openshift-proxy-signer@1432232228' \
  --serial='/etc/origin/master/proxyca.serial.txt'
```



### 注記

`oc adm ca create-signer-cert` コマンドは、5 年間有効な証明書を生成します。この期間は `--expire-days` オプションを使って変更することができますが、セキュリティ上の理由から、値をこれより大きくすることは推奨されません。

Ansible ホストインベントリーファイル (デフォルトで `/etc/ansible/hosts`) に最初に一覧表示されているマスターから `oc adm` コマンドを実行します。

3. このプロキシ用のクライアント証明書を生成します。x509 証明書ツールリングを使用して実行することができます。`oc adm CLI` を使用すると便利です。

```
# oc adm create-api-client-config \
  --certificate-authority='/etc/origin/master/proxyca.crt' \
  --client-dir='/etc/origin/master/proxy' \
  --signer-cert='/etc/origin/master/proxyca.crt' \
  --signer-key='/etc/origin/master/proxyca.key' \
  --signer-serial='/etc/origin/master/proxyca.serial.txt' \
  --user='system:proxy' 1
```

```
# pushd /etc/origin/master
# cp master.server.crt /etc/pki/tls/certs/localhost.crt 2
# cp master.server.key /etc/pki/tls/private/localhost.key
# cp ca.crt /etc/pki/CA/certs/ca.crt
# cat proxy/system\:proxy.crt \
  proxy/system\:proxy.key > \
  /etc/pki/tls/certs/authproxy.pem
# popd
```

- 1** ユーザー名は任意に指定できますが、ログにそのまま表示されるのでわかりやすい名前をつけると便利です。

- 2 マスターと異なるホスト名で認証プロキシーを実行する場合、上記のようにデフォルトのマスター証明書を使用するのではなく、ホスト名と一致する証明書を生成することが重要



### 注記

**oc adm create-api-client-config** コマンドは、2年間有効な証明書を生成します。この期間は **--expire-days** オプションを使って変更することができますが、セキュリティ上の理由から、値をこれより大きくすることは推奨されません。Ansible ホストインベントリーファイル (デフォルトで `/etc/ansible/hosts`) に最初に一覧表示されているマスターから **oc adm** コマンドを実行します。

### Apache の設定

このプロキシーはマスターと同じホストにある必要はありません。これはクライアント証明書を使用してマスターに接続し、**X-Remote-User** ヘッダーを信頼するように設定されます。

1. Apache 設定の証明書を作成します。**SSLProxyMachineCertificateFile** パラメーターの値として指定する証明書は、プロキシーをサーバーに対して認証するために使用されるプロキシーのクライアント証明書です。これは、拡張されたキーのタイプとして **TLS Web Client Authentication** を使用する必要があります。
2. Apache 設定を作成します。以下のテンプレートを使用して必要な設定および値を指定します。



### 重要

テンプレートを十分に確認し、その内容を環境に合うようにカスタマイズします。

```
LoadModule request_module modules/mod_request.so
LoadModule auth_gssapi_module modules/mod_auth_gssapi.so
# Some Apache configurations might require these modules.
# LoadModule auth_form_module modules/mod_auth_form.so
# LoadModule session_module modules/mod_session.so

# Nothing needs to be served over HTTP. This virtual host simply redirects to
# HTTPS.
<VirtualHost *:80>
    DocumentRoot /var/www/html
    RewriteEngine On
    RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
    # This needs to match the certificates you generated. See the CN and X509v3
    # Subject Alternative Name in the output of:
    # openssl x509 -text -in /etc/pki/tls/certs/localhost.crt
    ServerName www.example.com

    DocumentRoot /var/www/html
    SSLEngine on
    SSLCertificateFile /etc/pki/tls/certs/localhost.crt
    SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
    SSLCACertificateFile /etc/pki/CA/certs/ca.crt
```

```

SSLProxyEngine on
SSLProxyCACertificateFile /etc/pki/CA/certs/ca.crt
# It's critical to enforce client certificates on the Master. Otherwise
# requests could spoof the X-Remote-User header by accessing the Master's
# /oauth/authorize endpoint directly.
SSLProxyMachineCertificateFile /etc/pki/tls/certs/authproxy.pem

# Send all requests to the console
RewriteEngine On
RewriteRule ^/console(.*)$ https://%{HTTP_HOST}:8443/console$1 [R,L]

# In order to using the challenging-proxy an X-Csrftoken must be present.
RewriteCond %{REQUEST_URI} ^/challenging-proxy
RewriteCond %{HTTP:X-Csrftoken} ^$ [NC]
RewriteRule ^.* - [F,L]

<Location /challenging-proxy/oauth/authorize>
# Insert your backend server name/ip here.
ProxyPass https://[MASTER]:8443/oauth/authorize
AuthName "SSO Login"
# For Kerberos
AuthType GSSAPI
Require valid-user
RequestHeader set X-Remote-User %{REMOTE_USER}s

GssapiCredStore keytab:/etc/httpd/protected/auth-proxy.keytab
# Enable the following if you want to allow users to fallback
# to password based authentication when they do not have a client
# configured to perform kerberos authentication
GssapiBasicAuth On

# For ldap:
# AuthBasicProvider ldap
# AuthLDAPURL "ldap://ldap.example.com:389/ou=People,dc=my-domain,dc=com?uid?
sub?(objectClass=*)"

# It's possible to remove the mod_auth_gssapi usage and replace it with
# something like mod_auth_mellon, which only supports the login flow.
</Location>

<Location /login-proxy/oauth/authorize>
# Insert your backend server name/ip here.
ProxyPass https://[MASTER]:8443/oauth/authorize

AuthName "SSO Login"
AuthType GSSAPI
Require valid-user
RequestHeader set X-Remote-User %{REMOTE_USER}s env=REMOTE_USER

GssapiCredStore keytab:/etc/httpd/protected/auth-proxy.keytab
# Enable the following if you want to allow users to fallback
# to password based authentication when they do not have a client
# configured to perform kerberos authentication
GssapiBasicAuth On

ErrorDocument 401 /login.html

```

```

    </Location>

  </VirtualHost>

  RequestHeader unset X-Remote-User

```

### マスターの設定

`/etc/origin/master/master-config.yaml` ファイルの **identityProviders** スタンザも更新する必要があります。

```

identityProviders:
- name: requestheader
  challenge: true
  login: true
  provider:
    apiVersion: v1
    kind: RequestHeaderIdentityProvider
    challengeURL: "https://[MASTER]/challenging-proxy/oauth/authorize?${query}"
    loginURL: "https://[MASTER]/login-proxy/oauth/authorize?${query}"
    clientCA: /etc/origin/master/proxyca.crt
    headers:
    - X-Remote-User

```

### サービスの再起動

最後に、以下のサービスを再起動します。

```

# systemctl restart httpd
# master-restart api
# master-restart controllers

```

### 設定の確認

1. プロキシをバイパスしてテストします。正しいクライアント証明書とヘッダーを指定すれば、トークンを要求できます。

```

# curl -L -k -H "X-Remote-User: joe" \
  --cert /etc/pki/tls/certs/authproxy.pem \
  https://[MASTER]:8443/oauth/token/request

```

2. クライアント証明書を指定しない場合、要求は拒否されます。

```

# curl -L -k -H "X-Remote-User: joe" \
  https://[MASTER]:8443/oauth/token/request

```

3. これは、設定された **challengeURL** (追加のクエリーパラメーターを含む) へのリダイレクトを示します。

```

# curl -k -v -H 'X-Csrf-Token: 1' \
  '<masterPublicURL>/oauth/authorize?client_id=openshift-challenging-
  client&response_type=token'

```

4. これにより、**WWW-Authenticate** 基本チャレンジ、ネゴシエートチャレンジ、またはそれらの両方のチャレンジを含む 401 応答が表示されるはずですが。

```
# curl -k -v -H 'X-Csrf-Token: 1'\
  '<redirected challengeURL from step 3 +query>'
```

5. Kerberos チケットを使用または使用せずに、**oc** コマンドラインへのログインをテストします。

- a. **kinit** を使用して Kerberos チケットを生成した場合は、これを破棄します。

```
# kdestroy -c cache_name ❶
```

- ❶ Kerberos キャッシュの名前を指定します。

- b. Kerberos 認証情報を使用して **oc** コマンドラインにログインします。

```
# oc login
```

プロンプトで、Kerberos ユーザー名およびパスワードを入力します。

- c. **oc** コマンドラインからログアウトします。

```
# oc logout
```

- d. Kerberos 認証情報を使用してチケットを取得します。

```
# kinit
```

プロンプトで、Kerberos ユーザー名およびパスワードを入力します。

- e. **oc** コマンドラインにログインできることを確認します。

```
# oc login
```

設定が正しい場合は、別の認証情報を入力せずにログインできます。

### 13.3.10. GitHub および GitHub Enterprise

GitHub は OAuth を使用します。OAuth は OpenShift Container Platform と GitHub または GitHub Enterprise 間のトークン交換フローを容易にします。

GitHub 統合を使用して GitHub または GitHub Enterprise のいずれかに接続できます。GitHub Enterprise 統合の場合、インスタンスの **hostname** を指定する必要があり、サーバーへの要求で使用する **ca** 証明書バンドルをオプションで指定することができます。



#### 注記

とくに記述がない限り、以下の手順が GitHub および GitHub Enterprise の両方に適用されます。

GitHub 認証を設定することによって、ユーザーは GitHub 認証情報を使用して OpenShift Container Platform にログインできます。GitHub ユーザー ID を持つすべてのユーザーが OpenShift Container Platform クラスタにログインできないようにするために、アクセスを特定の GitHub 組織のユーザーに制限することができます。

### 13.3.10.1. GitHub でのアプリケーションの登録

1. アプリケーションを登録します。
  - GitHub の場合、 [Settings](#) → [Developer settings](#) → [Register a new OAuth application](#) をクリックします。
  - GitHub Enterprise の場合は、GitHub Enterprise ホームページに移動してから [Settings](#) → [Developer settings](#) → [Register a new application](#) をクリックします。
2. アプリケーション名を入力します (例: **My OpenShift Install**)。
3. ホームページの URL を入力します (例: <https://myapiserver.com:8443>)。
4. オプションで、アプリケーションの説明を入力します。
5. 承認コールバック URL を入力します。ここで、URL の末尾にはアイデンティティプロバイダーの **名前** (マスター設定ファイルの [identityProviders](#) スタンザで定義されます。このトピックの以下のセクションで設定を行います) が含まれます。

```
<apiserver>/oauth2callback/<identityProviderName>
```

以下に例を示します。

```
https://myapiserver.com:8443/oauth2callback/github/
```

6. **Register application** をクリックします。GitHub はクライアント ID とクライアントシークレットを提供します。このウィンドウを開いたままにして、それらの値をコピーし、マスター設定ファイルに貼り付けます。

### 13.3.10.2. マスターでの認証の設定

1. 状況に応じて以下のいずれかの手順を実行します。
  - OpenShift Container Platform のインストールがすでに完了している場合は、`/etc/origin/master/master-config.yaml` ファイルを新規ディレクトリーにコピーします。以下は例になります。

```
$ cd /etc/origin/master
$ mkdir githubconfig; cp master-config.yaml githubconfig
```

- OpenShift Container Platform をまだインストールしていない場合は、OpenShift Container Platform API サーバーを起動し、(将来の) OpenShift Container Platform マスターのホスト名と、起動コマンドによって作成された設定ファイルを格納するディレクトリーを指定します。

```
$ openshift start master --public-master=<apiserver> --write-config=<directory>
```

以下に例を示します。

```
$ openshift start master --public-master=https://myapiserver.com:8443 --write-config=githubconfig
```



### 注記

Ansible を使用してインストールする場合は、**identityProvider** 設定を Ansible Playbook に追加する必要があります。Ansible を使用してインストールした後、以下の手順に従って設定を手動で変更した場合、インストールツールまたはアップグレードを再実行するたびに変更内容がすべて失われます。



### 注記

**openshift start master** を使用するとホスト名が自動的に検出されますが、GitHub はアプリケーション登録時に指定した正確なホスト名にリダイレクトできる必要があります。このため、ID は間違っただレスにリダイレクトする可能性があるために自動検出することはできません。代わりに、Web ブラウザーが OpenShift Container Platform クラスターとの対話に使用するホスト名を指定する必要があります。

2. 新規 **master-config.yaml** ファイルの **identityProviders** スタンザを編集し、**GitHubIdentityProvider** 設定例をコピーして貼り付け、既存のスタンザを置き換えます。

```

oauthConfig:
  ...
  identityProviders:
  - name: github ①
    challenge: false ②
    login: true ③
    mappingMethod: claim ④
    provider:
      apiVersion: v1
      kind: GitHubIdentityProvider
      ca: ... ⑤
      clientId: ... ⑥
      clientSecret: ... ⑦
      hostname: ... ⑧
      organizations: ⑨
      - myorganization1
      - myorganization2
      teams: ⑩
      - myorganization1/team-a
      - myorganization2/team-b
  
```

- ① このプロバイダー名は GitHub の数字ユーザー ID に接頭辞として付加され、アイデンティティ名が作成されます。これはコールバック URL を作成するためにも使用されます。
- ② **GitHubIdentityProvider** を使用して **WWW-Authenticate** チャレンジを送信することはできません。
- ③ **true** の場合、(Web コンソールの場合のように) Web クライアントからの非認証トークン要求はログインする GitHub にリダイレクトされます。
- ④ このプロバイダーのアイデンティティとユーザーオブジェクト間のマッピングの確立方法を制御します ([上記](#) を参照してください)。
- ⑤ GitHub Enterprise の場合、CA は、サーバーに要求を行う際に使用するオプションの信頼される認証局バンドルです。デフォルトのシステムルート証明書を使用するにはそのパラ

これらの認証用ハンドルは、ノブオールドのシムノムルート証明書で使用するにはこのハンドルを省略します。GitHub の場合は、このパラメーターを省略します。

- 6 登録済みの [GitHub OAuth アプリケーション](#) のクライアント ID。アプリケーションは、`<master>/oauth2callback/<identityProviderName>` のコールバック URL を使用して設定する必要があります。
  - 7 GitHub で発行されるクライアントシークレットです。この値は [環境変数](#)、[外部ファイル](#)、または[暗号化されたファイル](#)でも指定できます。
  - 8 GitHub Enterprise の場合、[example.com](#) などのインスタンスのホスト名を指定する必要があります。この値は `/setup/settings` ファイルにある GitHub Enterprise `hostname` 値に一致する必要があります。ポート番号を含めることはできません。GitHub の場合は、このパラメーターを省略します。
  - 9 組織のオプションの一覧です。これが指定されている場合、少なくとも一覧のいずれかの組織のメンバーである GitHub ユーザーのみがログインできます。その組織が `clientID` で設定された GitHub OAuth アプリケーションを所有していない場合、組織の所有者はこのオプションを使用するためにサードパーティーのアクセスを付与する必要があります。これは組織の管理者が初回の GitHub ログイン時に、または GitHub の組織設定で実行できます。これは `teams` フィールドと組み合わせて使用することはできません。
  - 10 チームのオプションの一覧です。これが指定されている場合、少なくとも一覧のいずれかのチームのメンバーである GitHub ユーザーのみがログインできます。そのチームの組織が `clientID` で設定された GitHub OAuth アプリケーションを所有していない場合、組織の所有者はこのオプションを使用するためにサードパーティーのアクセスを付与する必要があります。これは組織の管理者が初回の GitHub ログイン時に、または GitHub の組織設定で実行できます。これは `organizations` フィールドと組み合わせて使用することはできません。
3. 以下の変更を `identityProviders` スタンザに加えます。
    - a. プロバイダーの `name` を変更して、GitHub で設定したコールバック URL に一致させます。  
たとえば、コールバック URL を <https://myapiserver.com:8443/oauth2callback/github/> として定義した場合、`name` は `github` にする必要があります。
    - b. `clientID` を [以前に登録した](#) GitHub のクライアント ID に変更します。
    - c. `clientSecret` を [以前に登録した](#) GitHub のクライアントシークレットに変更します。
    - d. `organizations` または `teams` を変更して、ユーザーが認証を行うためにメンバーシップを設定している必要がある 1 つ以上の GitHub 組織またはチームの一覧を組み込むようにします。これが指定されている場合、少なくとも一覧のいずれかの組織またはチームのメンバーである GitHub ユーザーのみがログインできます。これが指定されていない場合、有効な GitHub アカウントを持つすべてのユーザーがログインできます。
  4. 変更を保存してファイルを閉じます。
  5. OpenShift Container Platform API サーバーを起動し、変更したばかりの設定ファイルを指定します。

```
$ openshift start master --config=<path/to/modified/config>/master-config.yaml
```

これが設定されると、OpenShift Container Platform の Web コンソールにログインするユーザーには GitHub の認証情報を使用してログインすることを求めるプロンプトが出されます。初回ログイン時

に、ユーザーは **authorize application** をクリックして GitHub が OpenShift Container Platform でのユーザー名、パスワードおよび組織のメンバーシップを使用することを許可する必要があります。その後、ユーザーは Web コンソールにリダイレクトされます。

### 13.3.10.3. GitHub 認証を持つユーザーの作成

GitHub などの外部認証プロバイダーを統合する場合は、ユーザーを OpenShift Container Platform では作成しません。GitHub または GitHub Enterprise は system of record であり、ユーザーは GitHub で定義され、指定される組織に属するすべてのユーザーがログインできることになります。

ユーザーを OpenShift Container Platform に追加するには、そのユーザーを GitHub または GitHub Enterprise で承認された組織に追加する必要があり、必要な場合は、そのユーザーの新しい GitHub アカウントを作成します。

### 13.3.10.4. ユーザーの確認

1名以上のユーザーがログインしたら、**oc get users** を実行してユーザーの一覧を表示し、ユーザーが正しく作成されていることを確認できます。

#### 例13.6 oc get users コマンドの出力

```
$ oc get users
NAME      UID                                FULL NAME  IDENTITIES
bobsmith  433b5641-066f-11e6-a6d8-acfc32c1ca87  Bob Smith  github:873654 ①
```

- ① OpenShift Container Platform のアイデンティティは、アイデンティティプロバイダー名と GitHub の内部の数字のユーザー ID で設定されます。そのため、ユーザーが GitHub のユーザー名またはメールアドレスを変更した場合でも、GitHub アカウントに割り当てられる認証情報に依存せず、OpenShift Container Platform にログインできます。これにより安定したログインが作成されます。

ここからは、[ユーザーロールの管理](#) 方法を学習することをお勧めします。

## 13.3.11. GitLab

[GitLab.com](#) またはその他の GitLab インスタンスをアイデンティティプロバイダーとして使用するには、**identityProviders** スタンザに **GitLabIdentityProvider** を設定します。GitLab バージョン 7.7.0 から 11.0 を使用する場合は、[OAuth 統合](#) を使用して接続します。GitLab バージョン 11.1 以降の場合は、OAuth ではなく [OpenID Connect \(OIDC\)](#) を使用して接続します。

#### 例13.7 GitLabIdentityProvider を使用したマスター設定

```
oauthConfig:
  ...
  identityProviders:
  - name: gitlab ①
    challenge: true ②
    login: true ③
    mappingMethod: claim ④
  provider:
    apiVersion: v1
```

```
kind: GitLabIdentityProvider
legacy: 5
url: ... 6
clientID: ... 7
clientSecret: ... 8
ca: ... 9
```

- 1 このプロバイダー名は GitLab 数字ユーザー ID に接頭辞として付加され、アイデンティティ名が作成されます。これはコールバック URL を作成するためにも使用されます。
- 2 **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。これは [Resource Owner Password Credentials](#) 付与フローを使用して GitLab からアクセストークンを取得します。
- 3 **true** の場合、Web クライアント (Web コンソールなど) からの非認証トークン要求はログインする GitLab にリダイレクトされます。
- 4 このプロバイダーのアイデンティティとユーザーオブジェクト間のマッピングの確立方法を制御します ([上記](#) を参照してください)。
- 5 認証プロバイダーとして OAuth または OIDC を使用するかどうかを決定します。OAuth を使用する場合は、**true** に設定され、OIDC を使用する場合は **false** に設定されます。OIDC を使用するには、GitLab.com または GitLab バージョン 11.1 以降を使用する必要があります。値を指定しない場合は、OAuth が GitLab インスタンスに接続するために使用され、OIDC は GitLab.com に接続するために使用されます。
- 6 GitLab プロバイダーのホスト URL です。これは <https://gitlab.com/> か、または他の GitLab の自己ホストインスタンスのいずれかになります。
- 7 [登録済みの GitLab OAuth アプリケーション](#) のクライアント ID です。アプリケーションは、`<master>/oauth2callback/<identityProviderName>` のコールバック URL を使用して設定する必要があります。
- 8 GitLab で発行されるクライアントシークレットです。この値は [環境変数](#)、[外部ファイル](#)、または [暗号化されたファイル](#) でも指定できます。
- 9 CA は、GitLab インスタンスへの要求を行う際に使用する任意の信頼される認証局バンドルです。空の場合、デフォルトのシステムルートが使用されます。

### 13.3.12. Google

Google の **OpenID Connect 統合** を使用して Google をアイデンティティプロバイダーとして使用するには、**identityProviders** スタンザに [GoogleIdentityProvider](#) を設定します。



#### 注記

Google をアイデンティティプロバイダーとして使用するには、`<master>/oauth/token/request` を使用してトークンを取得し、コマンドラインツールで使用する必要があります。



## 警告

Google をアイデンティティプロバイダーとして使用することで、Google ユーザーはサーバーに対して認証されます。以下のように **hostedDomain** 設定属性を使用して、特定のホストドメインのメンバーに認証を限定することができます。

## 例13.8 GoogleIdentityProvider を使用したマスター設定

```
oauthConfig:
  ...
  identityProviders:
    - name: google ①
      challenge: false ②
      login: true ③
      mappingMethod: claim ④
      provider:
        apiVersion: v1
        kind: GoogleIdentityProvider
        clientID: ... ⑤
        clientSecret: ... ⑥
        hostedDomain: "" ⑦
```

- ① このプロバイダー名は Google の数字のユーザー ID に接頭辞として付加され、アイデンティティ名が作成されます。これはリダイレクト URL を作成するためにも使用されます。
- ② **GoogleIdentityProvider** を使用して **WWW-Authenticate** チャレンジを送信することはできません。
- ③ **true** の場合、Web クライアント (Web コンソールなど) からの非認証トークン要求はログインする Google へリダイレクトされます。
- ④ このプロバイダーのアイデンティティとユーザーオブジェクト間のマッピングの確立方法を制御します (上記を参照してください)。
- ⑤ 登録済みの Google プロジェクト のクライアント ID です。プロジェクトは、`<master>/oauth2callback/<identityProviderName>` のリダイレクト URI で設定する必要があります。
- ⑥ Google で発行されるクライアントシークレットです。この値は [環境変数](#)、[外部ファイル](#)、または [暗号化されたファイル](#) でも指定できます。
- ⑦ サインインアカウントを制限するオプションの [ホスト型ドメイン](#) です。空の場合は、すべての Google アカウントの認証が許可されます。

## 13.3.13. OpenID Connect

[Authorization Code Flow](#) を使用して OpenID Connect アイデンティティプロバイダーと統合するには、**identityProviders** スタンザに **OpenIDIdentityProvider** を設定します。

OpenShift Container Platform の OpenID Connect アイデンティティプロバイダーとして [Red Hat シングルサインオンを設定](#) できます。



### 注記

ID Token および UserInfo の復号化はサポートされていません。

デフォルトで、**openid** の範囲が要求されます。必要な場合は、**extraScopes** フィールドで追加の範囲を指定できます。

要求は、OpenID アイデンティティプロバイダーから返される JWT **id\_token** から読み取られ、指定される場合は **UserInfo** URL によって返される JSON から読み取られます。

1つ以上の要求をユーザーのアイデンティティを使用するように設定される必要があります。標準のアイデンティティ要求は **sub** になります。

また、どの要求をユーザーの推奨ユーザー名、表示名およびメールアドレスとして使用するか指定することができます。複数の要求が指定されている場合は、値が入力されている最初の要求が使用されます。標準のアイデンティティ要求は以下の通りです。

<b>sub</b>	subject identifier の省略形です。発行側のユーザーのリモートアイデンティティです。
<b>preferred_username</b>	ユーザーのプロビジョニング時に優先されるユーザー名です。 <b>janedoe</b> などのユーザーを参照する際に使用する省略形の名前です。通常は、ユーザー名またはメールなどの、認証システムのユーザーのログインまたはユーザー名に対応する値です。
<b>email</b>	メールアドレス。
<b>name</b>	表示名。

詳細は、[OpenID claim のドキュメント](#) を参照してください。



### 注記

OpenID Connect アイデンティティプロバイダーを使用するには、**<master>/oauth/token/request** を使用してトークンを取得し、コマンドラインツールで使用する必要があります。

## OpenIDIdentityProvider を使用する標準マスター設定

```

oauthConfig:
  ...
  identityProviders:
  - name: my_openid_connect ❶
    challenge: true ❷
    login: true ❸
    mappingMethod: claim ❹
    provider:
      apiVersion: v1
      kind: OpenIDIdentityProvider
      clientID: ... ❺

```

```

clientSecret: ... 6
claims:
  id: 7
  - sub
  preferredUsername:
  - preferred_username
  name:
  - name
  email:
  - email
urls:
  authorize: https://myidp.example.com/oauth2/authorize 8
  token: https://myidp.example.com/oauth2/token 9

```

- 1 このプロバイダー名はアイデンティティー要求の値に接頭辞として付加され、アイデンティティー名が作成されます。これはリダイレクト URL を作成するためにも使用されます。
- 2 **true** の場合、非 Web クライアント (CLI など) からの認証されていないトークン要求は、このプロバイダーの **WWW-Authenticate** challenge ヘッダーと共に送信されます。この場合、OpenID プロバイダーが [Resource Owner Password Credentials](#) 付与フローをサポートしている必要があります。
- 3 **true** の場合、Web クライアント (Web コンソールなど) からの非認証トークン要求は、ログインする認証 URL にリダイレクトされます。
- 4 このプロバイダーのアイデンティティーとユーザーオブジェクト間のマッピングの確立方法を制御します ([上記](#) を参照してください)。
- 5 OpenID プロバイダーに登録されているクライアントのクライアント ID です。このクライアントは `<master>/oauth2callback/<identityProviderName>` にリダイレクトすることを許可されている必要があります。
- 6 クライアントシークレット。この値は [環境変数](#)、[外部ファイル](#)、または[暗号化されたファイル](#)でも指定できます。
- 7 アイデンティティーとして使用する要求の一覧です。空でない最初の要求が使用されます。1つ以上の要求が必要になります。一覧表示される要求のいずれにも値がないと、認証は失敗します。たとえば、これは、ユーザーのアイデンティティーとして、返される `id_token` の `sub` 要求の値を使用します。
- 8 OpenID 仕様に記述される [承認エンドポイント](#) です。https を使用する必要があります。
- 9 OpenID 仕様に記述される [トークンエンドポイント](#) です。https を使用する必要があります。

カスタム証明書バンドル、追加の範囲、追加の承認要求パラメーター、および `userInfo` URL も指定できます。

### 例13.9 OpenIDIdentityProvider を使用する完全なマスター設定

```

oauthConfig:
  ...
  identityProviders:
  - name: my_openid_connect
    challenge: false
    login: true

```

```
mappingMethod: claim
provider:
  apiVersion: v1
  kind: OpenIDIdentityProvider
  clientId: ...
  clientSecret: ...
  ca: my-openid-ca-bundle.crt ①
  extraScopes: ②
  - email
  - profile
  extraAuthorizeParameters: ③
  include_granted_scopes: "true"
  claims:
    id: ④
    - custom_id_claim
    - sub
  preferredUsername: ⑤
  - preferred_username
  - email
  name: ⑥
  - nickname
  - given_name
  - name
  email: ⑦
  - custom_email_claim
  - email
  urls:
    authorize: https://myidp.example.com/oauth2/authorize
    token: https://myidp.example.com/oauth2/token
    userInfo: https://myidp.example.com/oauth2/userinfo ⑧
```

- ① 設定される URL のサーバー証明書を検証するために使用する証明書バンドルです。空の場合、システムで信頼されるルートを使用します。
- ② 承認トークン要求時に `openid` の範囲のほかに要求する範囲のオプションの一覧です。
- ③ 認証トークン要求に追加する追加パラメーターのオプションのマッピングです。
- ④ アイデンティティとして使用する要求の一覧です。空でない最初の要求が使用されます。1つ以上の要求が必要になります。一覧表示される要求のいずれにも値がないと、認証は失敗します。
- ⑤ このアイデンティティのユーザーをプロビジョニングする際に推奨ユーザー名として使用される要求の一覧です。空でない最初の要求が使用されます。
- ⑥ 表示名として使用する要求の一覧です。空でない最初の要求が使用されます。
- ⑦ メールアドレスとして使用する要求の一覧です。空でない最初の要求が使用されます。
- ⑧ OpenID 仕様に記述される [UserInfo エンドポイント](#) です。 **https** を使用する必要があります。

## 13.4. トークンオプション

OAuth サーバーは以下の 2 種類のトークンを生成します。

アクセストークン	API へのアクセスを付与する永続的なトークン。
認証コード	アクセストークンの交換にのみ使われる一時的なトークン。

`tokenConfig` スタンザを使用してトークンオプションを設定します。

### 例13.10 マスター設定のトークンオプション

```
oauthConfig:
...
tokenConfig:
  accessTokenMaxAgeSeconds: 86400 ①
  authorizeTokenMaxAgeSeconds: 300 ②
```

- ① **accessTokenMaxAgeSeconds** を設定して、アクセストークンの有効期間を制御します。デフォルトの期間は 24 時間です。
- ② **authorizeTokenMaxAgeSeconds** を設定して、認証コードの有効期間を制御します。デフォルトの期間は 5 分です。



#### 注記

**OAuthClient** オブジェクト定義により **accessTokenMaxAgeSeconds** 値を上書きできません。

## 13.5. 付与オプション

OAuth サーバーが、ユーザーが以前にパーミッションを付与していないクライアントに対するトークン要求を受信する場合、OAuth サーバーが実行するアクションは OAuth クライアントの付与ストラテジーによって変わります。

トークンを要求する OAuth クライアントが独自の付与ストラテジーを提供しない場合、サーバー全体でのデフォルトストラテジーが使用されます。デフォルトストラテジーを設定するには、`grantConfig` スタンザで **method** 値を設定します。**method** の有効な値は以下の通りです。

<b>auto</b>	付与を自動承認し、要求を再試行します。
<b>prompt</b>	ユーザーに対して付与の承認または拒否を求めるプロンプトを出します。
<b>deny</b>	付与を自動的に拒否し、失敗エラーをクライアントに返します。

### 例13.11 マスター設定の付与オプション

```
oauthConfig:
...
```

```
grantConfig:
  method: auto
```

## 13.6. セッションオプション

OAuth サーバーは、ログインおよびリダイレクトフローで署名および暗号化される Cookie ベースセッションを使用します。

**sessionConfig** スタンザを使用してセッションオプションを設定します。

### 例13.12 マスター設定のセッションオプション

```
oauthConfig:
  ...
  sessionConfig:
    sessionMaxAgeSeconds: 300 ①
    sessionName: ssn ②
    sessionSecretsFile: "... " ③
```

- ① セッションの最大期間を制御します。トークン要求が完了すると、セッションは自動的に期限切れとなります。**auto-grant** が有効にされていない場合、ユーザーがクライアント承認要求を承認または拒否するためにかかると想定される時間の間、セッションは継続する必要があります。
- ② セッションを保存するために使用される Cookie の名前です。
- ③ シリアライズされた **SessionSecrets** オブジェクトを含むファイル名です。空の場合、サーバーが起動されるたびにランダムな署名および暗号化シークレットが生成されます。

**sessionSecretsFile** が指定されていない場合、マスターサーバーが起動されるたびにランダムな署名および暗号化シークレットが生成されます。つまりマスターが再起動されると、進行中のログインではセッションが無効になります。また、これは他のマスターのいずれかによって生成されるセッションを復号化することはできないことも意味します。

使用する署名および暗号化シークレットを指定するには、**sessionSecretsFile** を指定します。これにより、シークレット値と設定ファイルを分離でき、たとえば、設定ファイルをデバッグなどの目的に合わせて配布可能な状態とすることができます。

複数のシークレットを **sessionSecretsFile** に指定してローテーションを有効にできます。一覧の最初のシークレットを使用して、新しいセッションに署名し、これを暗号化します。既存のセッションは、成功するまで各シークレットによって復号化され、認証されます。

### 例13.13 セッションシークレット設定:

```
apiVersion: v1
kind: SessionSecrets
secrets: ①
- authentication: "... " ②
  encryption: "... " ③
```

```
- authentication: "..."  
  encryption: "..."  
...
```

- 1 Cookie セッションの認証と暗号化に使用するシークレットの一覧です。シークレットを1つ以上指定する必要があります。各シークレットでは認証および暗号化シークレットを設定する必要があります。
- 2 署名シークレットです。HMAC を使用してセッションを認証するために使用されます。32 または 64 バイトでシークレットを使用することを推奨しています。
- 3 暗号化シークレットです。セッションを暗号化するために使用されます。16、24、または 32 文字で、AES-128、AES-192、または AES-256 を選択するために使用されます。

### 13.7. ユーザーエージェントによる CLI バージョンの不一致の防止

OpenShift Container Platform は、アプリケーション開発者の CLI が OpenShift Container Platform API にアクセスできないように、ユーザーエージェントを実装しています。

OpenShift Container Platform CLI のユーザーエージェントは、OpenShift Container Platform 内の値のセットで設定されています。

```
<command>/<version>+<git_commit> (<platform>/<architecture>) <client>/<git_commit>
```

たとえば、以下の場合を考慮しましょう。

- <command> = **oc**
- <version> = クライアントのバージョン。たとえば、**v3.3.0**。/api で Kubernetes API に対して行われる要求が Kubernetes のバージョンを受信し、/oapi で OpenShift Container Platform API に対して行われる要求が OpenShift Container Platform のバージョン (**oc version** によって指定される) を受信します。
- <platform> = **linux**
- <architecture> = **amd64**
- <client> = **openshift** または **kubernetes**。要求が /api で Kubernetes API に対して行われるか、/oapi で OpenShift Container Platform API に対して行われるかによって決まります。
- <git\_commit> = クライアントバージョンの Git コミット (例: **f034127**)

上記の場合、ユーザーエージェントは以下のようになります。

```
oc/v3.3.0+f034127 (linux/amd64) openshift/f034127
```

ユーザーエージェントはマスター設定ファイル `/etc/origin/master/master-config.yaml` で設定する必要があります。設定を適用するには、API サーバーを再起動します。

```
$ /usr/local/bin/master-restart api
```

OpenShift Container Platform 管理者として、マスター設定の **userAgentMatching** 設定を使用してクライアントが API にアクセスできないようにすることができます。そのため、クライアントが特定のライブラリーまたはバイナリーを使用している場合、クライアントは API にアクセスできなくなります。

以下のユーザーエージェントの例は、Kubernetes 1.2 クライアントバイナリー、OpenShift Origin 1.1.3 バイナリー、POST および PUT **httpVerbs** を拒否します。

```
policyConfig:
  userAgentMatchingConfig:
    defaultRejectionMessage: "Your client is too old. Go to https://example.org to update it."
    deniedClients:
      - regex: "\w+/v(?:1\.\1\1)|(?:1\0\1)) \\.+\.+\.+ openshift/\w{7}"
      - regex: "\w+/v(?:1\.\1\3) \\.+\.+\.+ openshift/\w{7}"
    httpVerbs:
      - POST
      - PUT
      - regex: "\w+/v1\2\0 \\.+\.+\.+ kubernetes/\w{7}"
    httpVerbs:
      - POST
      - PUT
    requiredClients: null
```

管理者は、予想されるクライアントに正確に一致しないクライアントを拒否することもできます。

```
policyConfig:
  userAgentMatchingConfig:
    defaultRejectionMessage: "Your client is too old. Go to https://example.org to update it."
    deniedClients: []
    requiredClients:
      - regex: "\w+/v1\1\3 \\.+\.+\.+ openshift/\w{7}"
      - regex: "\w+/v1\2\0 \\.+\.+\.+ kubernetes/\w{7}"
    httpVerbs:
      - POST
      - PUT
```

許可クライアントのセットに含まれるクライアントを拒否するには、**deniedClients** と **requiredClients** の値を一緒に使用します。以下の例では、1.13 以外のすべての 1.X クライアントバイナリーを許可します。

```
policyConfig:
  userAgentMatchingConfig:
    defaultRejectionMessage: "Your client is too old. Go to https://example.org to update it."
    deniedClients:
      - regex: "\w+/v1\13\0\+\w{7} \\.+\.+\.+ openshift/\w{7}"
      - regex: "\w+/v1\13\0\+\w{7} \\.+\.+\.+ kubernetes/\w{7}"
    requiredClients:
      - regex: "\w+/v1\.[1-9][1-9].[0-9]\+\w{7} \\.+\.+\.+ openshift/\w{7}"
      - regex: "\w+/v1\.[1-9][1-9].[0-9]\+\w{7} \\.+\.+\.+ kubernetes/\w{7}"
```



## 注記

クライアントのユーザーエージェントが設定と一致しない場合にエラーが発生します。変更する要求が一致するように、ホワイトリストを実施します。ルールは特定の verb にマップされるので、変化する要求を禁止し、変化しない要求を許可することができます。

## 第14章 グループと LDAP の同期

### 14.1. 概要

OpenShift Container Platform 管理者として、グループを使用してユーザーを管理し、権限を変更し、連携を強化できます。組織ではユーザーグループをすでに作成し、それらを LDAP サーバーに保存している場合があります。OpenShift Container Platform はそれらの LDAP レコードを内部 OpenShift Container Platform レコードと同期できるので、グループを1つの場所で管理できます。現時点で OpenShift Container Platform はグループメンバーシップを定義するための3つの共通スキーマ (RFC 2307、Active Directory、拡張された Active Directory) を使用してグループと LDAP サーバーの同期をサポートしています。



#### 注記

グループを同期するには **cluster-admin** 権限 が必要です。

### 14.2. LDAP 同期の設定

LDAP 同期を実行するには、同期設定ファイルが必要です。このファイルには LDAP クライアント設定の詳細が含まれます。

- LDAP サーバーへの接続の設定。
- LDAP サーバーで使用されるスキーマに依存する同期設定オプション。

同期設定ファイルには、OpenShift Container Platform Group 名を LDAP サーバーのグループにマップする管理者が定義した名前マッピングの一覧も含まれます。

#### 14.2.1. LDAP クライアント設定

##### LDAP クライアント設定

```
url: ldap://10.0.0.0:389 ①
bindDN: cn=admin,dc=example,dc=com ②
bindPassword: password ③
insecure: false ④
ca: my-ldap-ca-bundle.crt ⑤
```

- ① データベースをホストする LDAP サーバーの接続プロトコル、IP アドレス、および **scheme://host:port** としてフォーマットされる接続先のポートです。
- ② バインド DN として使用する任意の識別名 (DN) です。同期操作のエントリを取得するために昇格した権限が必要となる場合、OpenShift Container Platform はこれを使用します。
- ③ バインドに使用する任意のパスワードです。同期操作のエントリを取得するために昇格した権限が必要となる場合、OpenShift Container Platform はこれを使用します。この値は **環境変数**、**外部ファイル**、または**暗号化されたファイル**でも指定できます。
- ④ **false** の場合、セキュアな LDAP (**ldaps://**) URL は TLS を使用して接続し、非セキュアな LDAP (**ldap://**) URL は TLS にアップグレードされます。**true** の場合、**ldaps://** URL を指定しない場合はサーバーへの TLS 接続は行われません。指定している場合は、URL は TLS を使用して接続を試行します。

- 5 設定された URL のサーバー証明書を検証するために使用する証明書バンドルです。空の場合、OpenShift Container Platform はシステムで信頼されるルートを使用します。insecure が false に

### 14.2.2. LDAP クエリー定義

同期設定は、同期に必要なとなるエントリーの LDAP クエリー定義で設定されています。LDAP クエリーの特定の定義は、LDAP サーバーにメンバーシップ情報を保存するために使用されるスキーマに依存します。

#### LDAP クエリー定義

```
baseDN: ou=users,dc=example,dc=com 1
scope: sub 2
derefAliases: never 3
timeout: 0 4
filter: (objectClass=inetOrgPerson) 5
pageSize: 0 6
```

- 1 すべての検索が開始されるディレクトリーのブランチの識別名 (DN) です。ディレクトリーツリーの上部を指定する必要がありますが、ディレクトリーのサブツリーを指定することもできます。
- 2 検索の範囲です。有効な値は **base**、**one**、または **sub** です。これを定義しない場合、**sub** の範囲が使用されます。範囲オプションについては、[以下の表](#) で説明されています。
- 3 LDAP ツリーのエイリアスに関連する検索の動作です。有効な値は **never**、**search**、**base**、または **always** です。これを定義しない場合、デフォルトは **always** となり、エイリアスを逆参照します。逆参照の動作については [以下の表](#) で説明されています。
- 4 クライアントによって検索に許可される時間制限です。秒単位で表示されます。0 の値はクライアント側の制限がないことを意味します。
- 5 有効な LDAP 検索フィルターです。これを定義しない場合、デフォルトは **(objectClass=\*)** になります。
- 6 LDAP エントリーで測定される、サーバーからの応答ページの任意の最大サイズです。0 に設定すると、応答ページのサイズ制限はなくなります。クライアントまたはサーバーがデフォルトで許可しているエントリー数より多いエントリーをクエリーが返す場合、ページングサイズの設定が必要となります。

表14.1 LDAP 検索範囲オプション

LDAP 検索範囲	説明
<b>base</b>	クエリーに対して指定されるベース DN で指定するオブジェクトのみを考慮します。
<b>one</b>	クエリーについてベース DN とツリー内の同じレベルにあるすべてのオブジェクトを考慮します。
<b>sub</b>	クエリーに指定されるベース DN のサブツリー全体を考慮します。

表14.2 LDAP 逆参照動作

逆参照動作	説明
<b>never</b>	LDAP ツリーにあるエイリアスを逆参照しません。
<b>search</b>	検索中に見つかったエイリアスのみを逆参照します。
<b>base</b>	ベースオブジェクトを検索中にエイリアスのみを逆参照します。
<b>always</b>	LDAP ツリーにあるすべてのエイリアスを常に逆参照します。

### 14.2.3. ユーザー定義の名前マッピング

ユーザー定義の名前マッピングは、OpenShift Container Platform Groups の名前を LDAP サーバーでグループを検出する固有の識別子に明示的にマップします。マッピングは通常の YAML 構文を使用します。ユーザー定義のマッピングには LDAP サーバーのすべてのグループのエントリーを含めることも、それらのグループのサブセットのみを含めることもできます。ユーザー定義の名前マッピングを持たないグループが LDAP サーバーにある場合、同期時のデフォルト動作では OpenShift Container Platform Group の名前として指定される属性が使用されます。

#### ユーザー定義の名前マッピング

```
groupUIDNameMapping:
  "cn=group1,ou=groups,dc=example,dc=com": firstgroup
  "cn=group2,ou=groups,dc=example,dc=com": secondgroup
  "cn=group3,ou=groups,dc=example,dc=com": thirdgroup
```

## 14.3. LDAP 同期の実行

[同期設定ファイル](#) を作成すると、同期を開始できます。OpenShift Container Platform では、管理者は同じサーバーを使用して多数の異なる同期タイプを実行できます。



#### 注記

デフォルトでは、すべてのグループ同期またはプルーニング操作がドライランされるので、OpenShift Container Platform Group レコードを変更するために **sync-groups** コマンドで **--confirm** フラグを設定する必要があります。

LDAP サーバーからのすべてのグループを OpenShift Container Platform と同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=config.yaml --confirm
```

設定ファイルで指定された LDAP サーバーのグループに対応する OpenShift Container Platform の Group をすべて同期するには、以下を実行します。

```
$ oc adm groups sync --type=openshift --sync-config=config.yaml --confirm
```

LDAP グループのサブセットと OpenShift Container Platform を同期するには、ホワイトリストファイル、ブラックリストファイル、またはその両方を使用します。



## 注記

ブラックリストファイル、ホワイトリストファイル、またはホワイトリストのリテラルの組み合わせを使用できます。ホワイトリストおよびブラックリストのファイルには1行ごとに1つの固有のグループ識別子を含める必要があり、ホワイトリストのリテラルはコマンド自体に直接含めることができます。これらのガイドラインはLDAPサーバーにあるグループと OpenShift Container Platform にすでにあるグループに適用されません。

```
$ oc adm groups sync --whitelist=<whitelist_file> \
  --sync-config=config.yaml \
  --confirm
$ oc adm groups sync --blacklist=<blacklist_file> \
  --sync-config=config.yaml \
  --confirm
$ oc adm groups sync <group_unique_identifier> \
  --sync-config=config.yaml \
  --confirm
$ oc adm groups sync <group_unique_identifier> \
  --whitelist=<whitelist_file> \
  --blacklist=<blacklist_file> \
  --sync-config=config.yaml \
  --confirm
$ oc adm groups sync --type=openshift \
  --whitelist=<whitelist_file> \
  --sync-config=config.yaml \
  --confirm
```

## 14.4. グループのプルーニングジョブの実行

グループを作成したLDAPサーバーのレコードが存在しなくなった場合、管理者は OpenShift Container Platform レコードからグループを削除することを選択できます。プルーニングジョブは、同期ジョブに使用されるものと同じ同期設定ファイルとホワイトまたはブラックリストを受け入れます。詳細は、[グループのプルーニング](#) セクションを参照してください。

## 14.5. 同期の例

このセクションでは、[RFC 2307](#)、[Active Directory](#) と [拡張された Active Directory](#) スキーマの例を紹介しています。以下のすべての例では2名のメンバー (**Jane** と **Jim**) を持つ **admins** というグループを同期しています。それぞれの例では以下について説明しています。

- グループとユーザーがLDAPサーバーに追加される方法。
- LDAP同期設定ファイルの概観。
- 同期後に生成される OpenShift Container Platform の Group レコード。



## 注記

これらの例では、すべてのユーザーがそれぞれのグループの直接的なメンバーであることを想定しています。とくに、グループには他のグループがメンバーとして含まれません。ネスト化されたグループを同期する方法の詳細については、[ネスト化されたメンバーシップ同期の例](#) を参照してください。

### 14.5.1. RFC 2307 スキーマの使用によるグループの同期

RFC 2307 スキーマでは、ユーザー (Jane と Jim) とグループの両方がファーストクラスエントリーとして LDAP サーバーに存在し、グループメンバーシップはグループの属性に保存されます。以下の `ldif` のスニペットでは、このスキーマのユーザーとグループを定義しています。

#### RFC 2307 スキーマを使用する LDAP エントリー: `rfc2307.ldif`

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com ①
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com ②
member: cn=Jim,ou=users,dc=example,dc=com
```

- ① このグループは LDAP サーバーのファーストクラスエントリーです。
- ② グループのメンバーは、グループの属性としての識別参照と共に一覧表示されます。

このグループを同期するには、まず設定ファイルを作成する必要があります。RFC 2307 スキーマでは、ユーザーとグループエントリー両方の LDAP クエリー定義と内部 OpenShift Container Platform レコードでそれらを表すのに使用する属性を指定する必要があります。

明確にするために、OpenShift Container Platform で作成するグループは (可能な場合) ユーザーまたは管理者に表示されるフィールドに識別名以外の属性を使用する必要があります。たとえば、メールによって OpenShift Container Platform Group のユーザーを識別し、一般名としてグループの名前を使用します。以下の設定ファイルでは、このような関係を作成しています。



## 注記

ユーザー定義の名前マッピングを使用する場合は、[設定ファイル](#) が異なります。

### RFC 2307 スキーマを使用する LDAP 同期設定: rfc2307\_config.yaml

```
kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389 ❶
insecure: false ❷
rfc2307:
  groupsQuery:
    baseDN: "ou=groups,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
  groupUIDAttribute: dn ❸
  groupNameAttributes: [ cn ] ❹
  groupMembershipAttributes: [ member ] ❺
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
  userUIDAttribute: dn ❻
  userNameAttributes: [ uid ] ❼
  tolerateMemberNotFoundErrors: false
  tolerateMemberOutOfScopeErrors: false
```

- ❶ このグループのレコードが保存される LDAP サーバーの IP アドレスとホストです。
- ❷ **false** の場合、セキュアな LDAP (**ldaps://**) URL は TLS を使用して接続し、非セキュアな LDAP (**ldap://**) URL は TLS にアップグレードされます。**true** の場合、**ldaps://** URL を指定しない場合はサーバーへの TLS 接続は行われません。指定している場合は、URL は TLS を使用して接続を試行します。
- ❸ LDAP サーバーのグループを一意に識別する属性です。groupUIDAttribute に DN を使用している場合、**groupsQuery** フィルターを指定できません。詳細なフィルターを実行するには、[ホワイトリスト/ブラックリスト方法](#) を使用します。
- ❹ Group の名前として使用する属性です。
- ❺ メンバーシップ情報を保存するグループの属性です。
- ❻ LDAP サーバーでユーザーを一意に識別する属性です。userUIDAttribute に DN を使用している場合は、**usersQuery** フィルターを指定できません。詳細なフィルターを実行するには、[ホワイトリスト/ブラックリスト方法](#) を使用します。
- ❼ OpenShift Container Platform Group レコードでユーザー名として使用される属性です。

rfc2307\_config.yaml ファイルと同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=rfc2307_config.yaml --confirm
```

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

### rfc2307\_config.yaml ファイルを使用して作成される OpenShift Container Platform Group

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 ①
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com ②
    openshift.io/ldap.url: LDAP_SERVER_IP:389 ③
  creationTimestamp:
    name: admins ④
users: ⑤
- jane.smith@example.com
- jim.adams@example.com
```

- ① この OpenShift Container Platform Group と LDAP サーバーが最後に同期された時間です。ISO 6801 形式を使用します。
- ② LDAP サーバーのグループの固有識別子です。
- ③ このグループのレコードが保存される LDAP サーバーの IP アドレスとポートです。
- ④ 同期ファイルが指定するグループ名です。
- ⑤ グループのメンバーのユーザーです。同期ファイルで指定される名前が使用されます。

#### 14.5.1.1. ユーザー定義の名前マッピングに関する RFC2307

グループとユーザー定義の名前マッピングを同期する場合、設定ファイルは、以下に示すこれらのマッピングが含まれるように変更されます。

#### ユーザー定義の名前マッピングに関する RFC 2307 スキーマを使用する LDAP 同期設定: rfc2307\_config\_user\_defined.yaml

```
kind: LDAPSyncConfig
apiVersion: v1
groupUIDNameMapping:
  "cn=admins,ou=groups,dc=example,dc=com": Administrators ①
rfc2307:
  groupsQuery:
    baseDN: "ou=groups,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
  groupUIDAttribute: dn ②
  groupNameAttributes: [ cn ] ③
  groupMembershipAttributes: [ member ]
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
```

```
derefAliases: never
pageSize: 0
userUIDAttribute: dn ④
userNameAttributes: [ uid ]
tolerateMemberNotFoundErrors: false
tolerateMemberOutOfScopeErrors: false
```

- ① ユーザー定義の名前マッピングです。
- ② ユーザー定義の名前マッピングでキーに使用される固有の識別属性です。groupUIDAttribute に DN を使用している場合は **groupsQuery** フィルターを指定できません。詳細なフィルターを実行するには、[ホワイトリスト/ブラックリスト方法](#) を使用します。
- ③ 固有の識別子がユーザー定義の名前マッピングに存在しない場合に OpenShift Container Platform Group に名前を付けるための属性です。
- ④ LDAP サーバーでユーザーを一意に識別する属性です。userUIDAttribute に DN を使用している場合は、**usersQuery** フィルターを指定できません。詳細なフィルターを実行するには、[ホワイトリスト/ブラックリスト方法](#) を使用します。

rfc2307\_config\_user\_defined.yaml ファイルと同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=rfc2307_config_user_defined.yaml --confirm
```

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

rfc2307\_config\_user\_defined.yaml ファイルを使用して作成される OpenShift Container Platform Group

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp:
  name: Administrators ①
users:
- jane.smith@example.com
- jim.adams@example.com
```

- ① ユーザー定義の名前マッピングが指定するグループ名です。

#### 14.5.2. ユーザー定義のエラートレランスに関する RFC 2307 の使用によるグループの同期

デフォルトでは、同期されるグループにメンバークエリーで定義された範囲外にあるエントリーを持つメンバーが含まれる場合、グループ同期は以下のエラーを出して失敗します。

Error determining LDAP group membership for "<group>": membership lookup for user "<user>" in group "<group>" failed because of "search for entry with dn=<user-dn>" would search outside of the base dn specified (dn=<base-dn>").

これは **usersQuery** フィールドの **baseDN** が間違っていて設定されていることを示していることがよくあります。ただし、**baseDN** にグループの一部のメンバーが意図的に含まれていない場合、**tolerateMemberOutOfScopeErrors: true** を設定することでグループ同期が継続されます。範囲外のメンバーは無視されます。

同様に、グループ同期プロセスでグループのメンバーの検出に失敗した場合、同期はエラーを出して失敗します。

Error determining LDAP group membership for "<group>": membership lookup for user "<user>" in group "<group>" failed because of "search for entry with base dn=<user-dn>" refers to a non-existent entry".

Error determining LDAP group membership for "<group>": membership lookup for user "<user>" in group "<group>" failed because of "search for entry with base dn=<user-dn>" and filter "<filter>" did not return any results".

これは、**usersQuery** フィールドが間違っていて設定されていることを示していることがよくあります。ただし、グループに欠落していると認識されているメンバーエントリが含まれる場合、**tolerateMemberNotFoundErrors: true** を設定することでグループ同期が継続されます。問題のあるメンバーは無視されます。



#### 警告

LDAP グループ同期のエラートレランスを有効にすると、同期プロセスは問題のあるメンバーエントリを無視します。LDAP グループ同期が正しく設定されていない場合、同期された OpenShift Container Platform Group にメンバーが欠落する可能性があります。

### 問題のあるグループメンバーシップに関する RFC 2307 スキーマを使用する LDAP エントリ: rfc2307\_problematic\_users.ldif

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users
```

```
dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
```

```
dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
```

```

objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admins,ou=groups,dc=example,dc=com
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=Jim,ou=users,dc=example,dc=com
member: cn=INVALID,ou=users,dc=example,dc=com ❶
member: cn=Jim,ou=OUTOFSCOPE,dc=example,dc=com ❷

```

- ❶ LDAP サーバーに存在しないメンバーです。
- ❷ 存在する可能性はあるが、同期ジョブのユーザークエリーでは **baseDN** に存在しないメンバーです。

上記の例でエラーを許容するには、以下を同期設定ファイルに追加する必要があります。

#### エラーを許容する RFC 2307 スキーマを使用した LDAP 同期設定: rfc2307\_config\_tolerating.yaml

```

kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
rfc2307:
  groupsQuery:
    baseDN: "ou=groups,dc=example,dc=com"
    scope: sub
    derefAliases: never
    groupUIDAttribute: dn
    groupNameAttributes: [ cn ]
    groupMembershipAttributes: [ member ]
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
  userUIDAttribute: dn ❶
  userNameAttributes: [ uid ]
  tolerateMemberNotFoundErrors: true ❷
  tolerateMemberOutOfScopeErrors: true ❸

```

- ❷ **true** の場合、同期ジョブは一部のメンバーが見つからなかったグループを許容し、LDAP エントリーが見つからなかったメンバーは無視されます。グループのメンバーが見つからない場合、同期ジョブのデフォルト動作は失敗します。

- 3 true の場合、同期ジョブは、一部のメンバーが **usersQuery** ベース DN で指定されるユーザー範囲外にいるグループを許容し、メンバークエリー範囲外のメンバーは無視されます。グループのメ
- 1 LDAP サーバーでユーザーを一意に識別する属性です。userUIDAttribute に DN を使用している場合は、**usersQuery** フィルターを指定できません。詳細なフィルターを実行するには、[ホワイトリスト/ブラックリスト方法](#) を使用します。

rfc2307\_config\_tolerating.yaml ファイルを使用して同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=rfc2307_config_tolerating.yaml --confirm
```

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

rfc2307\_config.yaml ファイルを使用して作成される OpenShift Container Platform Group

```
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400
    openshift.io/ldap.uid: cn=admin,ou=groups,dc=example,dc=com
    openshift.io/ldap.url: LDAP_SERVER_IP:389
  creationTimestamp:
  name: admins
users: 1
- jane.smith@example.com
- jim.adams@example.com
```

- 1 同期ファイルで指定されるグループのメンバーのユーザーです。検索中に許容されるエラーがないメンバーです。

### 14.5.3. Active Directory の使用によるグループの同期

Active Directory スキーマでは、両方のユーザー (Jane と Jim) がファーストクラスエントリーとして LDAP サーバーに存在し、グループメンバーシップはユーザーの属性に保存されます。以下の **ldif** のスニペットでは、このスキーマのユーザーとグループを定義しています。

Active Directory スキーマを使用する LDAP エントリー: active\_directory.ldif

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
```

```
memberOf: admins ❶
```

```
dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: admins
```

- ❶ ユーザーのグループメンバーシップはユーザーの属性として一覧表示され、グループはサーバー上にエントリーとして存在しません。**memberOf** 属性はユーザーのリテラル属性である必要はありません。一部のLDAPサーバーでは、これは検索中に作成され、クライアントに返されますが、データベースにコミットされません。

このグループを同期するには、まず設定ファイルを作成する必要があります。Active Directory スキーマでは、ユーザーエントリーのLDAPクエリー定義と内部 OpenShift Container Platform Group レコードでそれらを表すのに使用する属性を指定する必要があります。

明確にするために、OpenShift Container Platform で作成するグループは(可能な場合)ユーザーまたは管理者に表示されるフィールドに識別名以外の属性を使用する必要があります。たとえば、メールによって OpenShift Container Platform Group のユーザーを識別しますが、LDAPサーバーのグループ名でグループの名前を定義します。以下の設定ファイルでは、このような関係を作成しています。

### Active Directory スキーマを使用する LDAP 同期設定: active\_directory\_config.yaml

```
kind: LDAPSvcConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
activeDirectory:
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    filter: (objectclass=inetOrgPerson)
    pageSize: 0
  userNameAttributes: [ uid ] ❶
  groupMembershipAttributes: [ memberOf ] ❷
```

- ❶ OpenShift Container Platform Group レコードでユーザー名として使用される属性です。
- ❷ メンバーシップ情報を保存するユーザーの属性です。

active\_directory\_config.yaml ファイルを使用して同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=active_directory_config.yaml --confirm
```

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

## active\_directory\_config.yaml ファイルを使用して作成される OpenShift Container Platform Group

```

apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 ❶
    openshift.io/ldap.uid: admins ❷
    openshift.io/ldap.url: LDAP_SERVER_IP:389 ❸
  creationTimestamp:
    name: admins ❹
users: ❺
- jane.smith@example.com
- jim.adams@example.com

```

- ❶ この OpenShift Container Platform Group と LDAP サーバーが最後に同期された時間です。ISO 6801 形式を使用します。
- ❷ LDAP サーバーのグループの固有識別子です。
- ❸ このグループのレコードが保存される LDAP サーバーの IP アドレスとホストです。
- ❹ LDAP サーバーに一覧表示されるグループ名です。
- ❺ グループのメンバーのユーザーです。同期ファイルで指定される名前が使用されます。

### 14.5.4. 拡張された Active Directory の使用によるグループの同期

拡張された Active Directory スキーマでは、両方のユーザー (Jane と Jim) とグループがファーストクラスエントリとして LDAP サーバーに存在し、グループメンバーシップはユーザーの属性に保存されます。以下の **Idif** のスニペットでは、このスキーマのユーザーとグループを定義しています。

#### 拡張された Active Directory スキーマを使用する LDAP エントリ: augmented\_active\_directory.ldif

```

dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users

dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com ❶

dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person

```

```

objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: cn=admin,ou=groups,dc=example,dc=com

dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups

dn: cn=admin,ou=groups,dc=example,dc=com ②
objectClass: groupOfNames
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
member: cn=Jane,ou=users,dc=example,dc=com
member: cn=Jim,ou=users,dc=example,dc=com

```

- ① ユーザーのグループメンバーシップはユーザーの属性として一覧表示されます。
- ② このグループは LDAP サーバーのファーストクラスエントリーです。

このグループを同期するには、まず設定ファイルを作成する必要があります。拡張された Active Directory スキーマでは、ユーザーエントリーとグループエントリーの両方の LDAP クエリー定義と内部 OpenShift Container Platform Group レコードでそれらを表すのに使用する属性を指定する必要があります。

明確にするために、OpenShift Container Platform で作成するグループは (可能な場合) ユーザーまたは管理者に表示されるフィールドに識別名以外の属性を使用する必要があります。たとえば、メールによって OpenShift Container Platform Group のユーザーを識別し、一般名としてグループの名前を使用します。以下の設定ファイルではこのような関係を作成しています。

### 拡張された Active Directory スキーマを使用する LDAP 同期設定: augmented\_active\_directory\_config.yaml

```

kind: LDAPSynConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
augmentedActiveDirectory:
  groupsQuery:
    baseDN: "ou=groups,dc=example,dc=com"
    scope: sub
    derefAliases: never
    pageSize: 0
  groupUIDAttribute: dn ①
  groupNameAttributes: [ cn ] ②
  usersQuery:
    baseDN: "ou=users,dc=example,dc=com"
    scope: sub
    derefAliases: never
    filter: (objectclass=inetOrgPerson)

```

```

  pageSize: 0
  userNameAttributes: [ uid ] ❸
  groupMembershipAttributes: [ memberOf ] ❹

```

- ❶ LDAP サーバーのグループを一意に識別する属性です。groupUIDAttribute に DN を使用している場合、**groupsQuery** フィルターを指定できません。詳細なフィルターを実行するには、[ホワイトリスト/ブラックリスト方法](#) を使用します。
- ❷ Group の名前として使用する属性です。
- ❸ OpenShift Container Platform Group レコードでユーザー名として使用される属性です。
- ❹ メンバーシップ情報を保存するユーザーの属性です。

`augmented_active_directory_config.yaml` ファイルを使用して同期するには、以下を実行します。

```
$ oc adm groups sync --sync-config=augmented_active_directory_config.yaml --confirm
```

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

`augmented_active_directory_config.yaml` ファイルを使用して作成される OpenShift Group

```

apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 ❶
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com ❷
    openshift.io/ldap.url: LDAP_SERVER_IP:389 ❸
  creationTimestamp:
    name: admins ❹
  users: ❺
  - jane.smith@example.com
  - jim.adams@example.com

```

- ❶ この OpenShift Container Platform Group と LDAP サーバーが最後に同期された時間です。ISO 6801 形式を使用します。
- ❷ LDAP サーバーのグループの固有識別子です。
- ❸ このグループのレコードが保存される LDAP サーバーの IP アドレスとポートです。
- ❹ 同期ファイルが指定するグループ名です。
- ❺ グループのメンバーのユーザーです。同期ファイルで指定される名前が使用されます。

## 14.6. ネスト化されたメンバーシップ同期の例

OpenShift Container Platform の Group はネスト化しません。LDAP サーバーはデータが使用される前にグループメンバーシップを平坦化する必要があります。Microsoft の Active Directory Server は、[LDAP\\_MATCHING\\_RULE\\_IN\\_CHAIN](#) ルールによりこの機能をサポートしており、これには OID

**1.2.840.113556.1.4.1941** が設定されています。さらに、このマッチングルールを使用すると、明示的に **ホワイトリスト化された** グループのみを同期できます。

このセクションでは、拡張された Active Directory スキーマの例を取り上げ、1名のユーザー **Jane** と1つのグループ **otheradmins** をメンバーとして持つ **admins** というグループを同期します。**otheradmins** グループには1名のユーザーメンバー **Jim** が含まれます。この例では以下のことを説明しています。

- グループとユーザーが LDAP サーバーに追加される方法。
- LDAP 同期設定ファイルの概観。
- 同期後に生成される OpenShift Container Platform の Group レコード。

拡張された Active Directory スキーマでは、ユーザー (**Jane** と **Jim**) とグループの両方がファーストクラスエントリーとして LDAP サーバーに存在し、グループメンバーシップはユーザーまたはグループの属性に保存されます。以下の **ldif** のスニペットはこのスキーマのユーザーとグループを定義します。

ネスト化されたメンバーを持つ拡張された Active Directory スキーマを使用する LDAP エントリー: **augmented\_active\_directory\_nested.ldif**

```
dn: ou=users,dc=example,dc=com
objectClass: organizationalUnit
ou: users
```

```
dn: cn=Jane,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jane
sn: Smith
displayName: Jane Smith
mail: jane.smith@example.com
memberOf: cn=admins,ou=groups,dc=example,dc=com ①
```

```
dn: cn=Jim,ou=users,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: testPerson
cn: Jim
sn: Adams
displayName: Jim Adams
mail: jim.adams@example.com
memberOf: cn=otheradmins,ou=groups,dc=example,dc=com ②
```

```
dn: ou=groups,dc=example,dc=com
objectClass: organizationalUnit
ou: groups
```

```
dn: cn=admins,ou=groups,dc=example,dc=com ③
objectClass: group
cn: admins
owner: cn=admin,dc=example,dc=com
description: System Administrators
```

```

member: cn=Jane,ou=users,dc=example,dc=com
member: cn=otheradmins,ou=groups,dc=example,dc=com

dn: cn=otheradmins,ou=groups,dc=example,dc=com 4
objectClass: group
cn: otheradmins
owner: cn=admin,dc=example,dc=com
description: Other System Administrators
memberOf: cn=admins,ou=groups,dc=example,dc=com 5 6
member: cn=Jim,ou=users,dc=example,dc=com

```

1 2 5 ユーザーとグループのメンバーシップはオブジェクトの属性として一覧表示されます。

3 4 このグループは LDAP サーバーのファーストクラスエントリーです。

6 **otheradmins** グループは **admins** グループのメンバーです。

Active Directory を使用してネスト化されたグループを同期するには、ユーザーエントリーとグループエントリーの両方の LDAP クエリ定義と内部 OpenShift Container Platform Group レコードでそれらを表すのに使用する属性を指定する必要があります。さらに、この設定では特定の変更が必要となります。

- **oc adm groups sync** コマンドはグループを明示的に **ホワイトリスト化** する必要があります。
- ユーザーの **groupMembershipAttributes** には "**memberOf:1.2.840.113556.1.4.1941:**" を含め、**LDAP\_MATCHING\_RULE\_IN\_CHAIN** ルールに従う必要があります。
- **groupUIDAttribute** は **dn** に設定される必要があります。
- **groupsQuery**:
  - **filter** を設定しないでください。
  - 有効な **derefAliases** を設定する必要があります。
  - **baseDN** を設定しないでください。この値は無視されます。
  - **scope** を設定しないでください。この値は無視されます。

明確にするために、OpenShift Container Platform で作成するグループは (可能な場合) ユーザーまたは管理者に表示されるフィールドに識別名以外の属性を使用する必要があります。たとえば、メールによって OpenShift Container Platform Group のユーザーを識別し、一般名としてグループの名前を使用します。以下の設定ファイルでは、このような関係を作成しています。

ネスト化されたメンバーを持つ拡張された Active Directory スキーマを使用する LDAP 同期設定です。 `augmented_active_directory_config_nested.yaml`

```

kind: LDAPSyncConfig
apiVersion: v1
url: ldap://LDAP_SERVICE_IP:389
augmentedActiveDirectory:
  groupsQuery: 1
  derefAliases: never
  pageSize: 0
  groupUIDAttribute: dn 2

```

```

groupNameAttributes: [ cn ] ③
usersQuery:
  baseDN: "ou=users,dc=example,dc=com"
  scope: sub
  derefAliases: never
  filter: (objectclass=inetOrgPerson)
  pageSize: 0
userNameAttributes: [ uid ] ④
groupMembershipAttributes: [ "memberOf:1.2.840.113556.1.4.1941:" ] ⑤

```

- ① **groupsQuery** フィルターは指定できません。**groupsQuery** ベース DN およびスコープの値は無視されます。**groupsQuery** では有効な **derefAliases** を設定する必要があります。
- ② LDAP サーバーのグループを一意に識別する属性です。**dn** に設定される必要があります。
- ③ Group の名前として使用する属性です。
- ④ OpenShift Container Platform Group レコードでユーザー名として使用される属性です。ほとんどのインストールでは、**uid** または **sAMAccountName** を使用することが推奨されます。
- ⑤ メンバーシップ情報を保存するユーザーの属性です。**LDAP\_MATCHING\_RULE\_IN\_CHAIN** を使用することに注意してください。

`augmented_active_directory_config_nested.yaml` ファイルを使用して同期するには、以下を実行します。

```

$ oc adm groups sync \
  'cn=admins,ou=groups,dc=example,dc=com' \
  --sync-config=augmented_active_directory_config_nested.yaml \
  --confirm

```



### 注記

`cn=admins,ou=groups,dc=example,dc=com` グループを明示的に ホワイトリスト化 する必要があります。

OpenShift Container Platform は、上記の同期操作の結果として以下のグループレコードを作成します。

`augmented_active_directory_config_nested.yaml` ファイルを使用して作成される OpenShift Group

```

apiVersion: user.openshift.io/v1
kind: Group
metadata:
  annotations:
    openshift.io/ldap.sync-time: 2015-10-13T10:08:38-0400 ①
    openshift.io/ldap.uid: cn=admins,ou=groups,dc=example,dc=com ②
    openshift.io/ldap.url: LDAP_SERVER_IP:389 ③
  creationTimestamp:
    name: admins ④

```

users: **5**

- jane.smith@example.com
- jim.adams@example.com

- 1 この OpenShift Container Platform Group と LDAP サーバーが最後に同期された時間です。ISO 6801 形式を使用します。
- 2 LDAP サーバーのグループの固有識別子です。
- 3 このグループのレコードが保存される LDAP サーバーの IP アドレスとホストです。
- 4 同期ファイルが指定するグループ名です。
- 5 グループのメンバーのユーザーです。同期ファイルで指定される名前が使用されます。グループメンバーシップは Microsoft Active Directory Server によって平坦化されているため、ネスト化されたグループのメンバーが含まれることに注意してください。

## 14.7. LDAP 同期設定の仕様

設定ファイルのオブジェクト仕様は以下で説明されています。スキーマオブジェクトにはそれぞれのフィールドがあることに注意してください。たとえば、`v1.ActiveDirectoryConfig` は `groupsQuery` フィールドを持ちませんが、`v1.RFC2307Config` と `v1.AugmentedActiveDirectoryConfig` の両方にこのフィールドがあります。



### 重要

バイナリー属性はサポートされていません。LDAP サーバーの全属性データは、UTF-8 エンコード文字列の形式である必要があります。たとえば、ID 属性として、バイナリー属性を使用することはできません (例: `objectGUID`)。代わりに `sAMAccountName` または `userPrincipalName` などの文字列属性を使用する必要があります。

### 14.7.1. v1.LDAPSyncConfig

`LDAPSyncConfig` は、LDAP グループ同期を定義するために必要な設定オプションを保持します。

名前	説明	スキーマ
<code>kind</code>	このオブジェクトが表す REST リソースを表す文字列の値です。サーバーはクライアントが要求を送信するエンドポイントからこれを推測できることがあります。これを更新することはできません。CamelCase。詳細については、 <a href="https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md#types-kinds">https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md#types-kinds</a> を参照してください。	文字列

名前	説明	スキーマ
<b>apiVersion</b>	オブジェクトのこの表現のバージョンスキーマを定義します。サーバーは認識されたスキーマを最新の内部値に変換し、認識されない値は拒否することがあります。詳細については、 <a href="https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md#resources">https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md#resources</a> を参照してください。	文字列
<b>url</b>	ホストは接続先のLDAPサーバーのスキーム、ホストおよびポートになります。 <b>scheme://host:port</b>	文字列
<b>bindDN</b>	LDAPサーバーをバインドする任意のDNです。	文字列
<b>bindPassword</b>	検索フェーズでバインドする任意のパスワードです。	v1.StringSource
<b>insecure</b>	<b>true</b> の場合、接続に TLS を使用できないことを示唆します。 <b>false</b> の場合、 <b>ldaps://URL</b> は TLS を使用して接続し、 <b>ldap://URL</b> は、 <a href="https://tools.ietf.org/html/rfc2830">https://tools.ietf.org/html/rfc2830</a> で指定されるように StartTLS を使用して TLS 接続にアップグレードされます。 <b>insecure</b> を <b>true</b> に設定し、 <b>ldaps://URL</b> スキームを使用する場合、URL は指定された <b>ca</b> を使用して TLS 接続を試行します。	ブール値
<b>ca</b>	サーバーへ要求を行う際に使用する任意の信頼された認証局バンドルです。空の場合、デフォルトのシステムルートが使用されます。	文字列
<b>groupUIDNameMapping</b>	LDAP グループ UID の OpenShift Container Platform Group 名への任意の直接マッピングです。	オブジェクト

名前	説明	スキーマ
<b>rfc2307</b>	RFC2307と同じ方法でセットアップされたLDAPサーバーからデータを抽出するための設定を保持します。ファーストクラスグループとユーザーエントリを抽出し、グループメンバーシップはメンバーを一覧表示するグループエントリの複数値の属性によって決定されます。	<a href="#">v1.RFC2307Config</a>
<b>activeDirectory</b>	Active Directoryに使用されるのと同じ方法でセットアップされたLDAPサーバーからデータを抽出するための設定を保持します。ファーストクラスユーザーエントリを抽出し、グループメンバーシップはメンバーが属するグループを一覧表示するメンバーの複数値の属性によって決定されます。	<a href="#">v1.ActiveDirectoryConfig</a>
<b>augmentedActiveDirectory</b>	上記のActive Directoryで使用されるのと同じ方法でセットアップされたLDAPサーバーからデータを抽出するための設定を保持します。1つの追加として、ファーストクラスグループエントリが存在し、それらはメタデータを保持するために使用されますが、グループメンバーシップは設定されません。	<a href="#">v1.AugmentedActiveDirectoryConfig</a>

### 14.7.2. v1.StringSource

**StringSource** によって文字列インラインを指定できます。または環境変数またはファイルを使用して外部から指定することもできます。文字列の値のみを含む場合、単純なJSON文字列にマーシャルします。

名前	説明	スキーマ
<b>value</b>	クリアテキスト値、または <b>keyFile</b> が指定されている場合は暗号化された値を指定します。	文字列
<b>env</b>	クリアテキスト値、または <b>keyFile</b> が指定されている場合は暗号化された値を含む環境変数を指定します。	文字列

名前	説明	スキーマ
<b>file</b>	クリアテキスト値、または <b>keyFile</b> が指定されている場合は暗号化された値を含むファイルを参照します。	文字列
<b>keyFile</b>	値を復号化するために使用するキーを含むファイルを参照します。	文字列

### 14.7.3. v1.LDAPQuery

**LDAPQuery** は LDAP クエリーの作成に必要なオプションを保持します。

名前	説明	スキーマ
<b>baseDN</b>	すべての検索が開始されるディレクトリーのブランチの DN です。	文字列
<b>scope</b>	検索の (任意の) 範囲です。 <b>base</b> (ベースオブジェクトのみ)、 <b>one</b> (ベースレベルのすべてのオブジェクト)、 <b>sub</b> (サブツリー全体) のいずれかになります。設定されていない場合は、デフォルトで <b>sub</b> になります。	文字列
<b>derefAliases</b>	エイリアスに関する検索の (任意の) 動作です。 <b>never</b> (エイリアスを逆参照しない)、 <b>search</b> (検索中の逆参照のみ)、 <b>base</b> (ベースオブジェクト検索時の逆参照のみ)、 <b>always</b> (常に逆参照を行う) のいずれかになります。設定されていない場合、デフォルトで <b>always</b> になります。	文字列
<b>timeout</b>	応答の待機を中止するまでにサーバーへの要求を未処理のままにする時間制限 (秒単位) を保持します。これが <b>0</b> の場合、クライアント側の制限が設定されないことになります。	整数
<b>filter</b>	ベース DN を持つ LDAP サーバーから関連するすべてのエントリーを取得する有効な LDAP 検索フィルターです。	文字列

名前	説明	スキーマ
<b>pageSize</b>	LDAP エントリーで測定される、推奨される最大ページサイズです。ページサイズ <b>0</b> はページングが実行されないことを意味します。	整数

#### 14.7.4. v1.RFC2307Config

**RFC2307Config** は、RFC2307 スキーマを使用してどのように LDAP グループ同期が LDAP サーバーに相互作用するかを定義するために必要な設定オプションを保持します。

名前	説明	スキーマ
<b>groupsQuery</b>	グループエントリーを返す LDAP クエリーのテンプレートを保持します。	<a href="#">v1.LDAPQuery</a>
<b>groupUIDAttribute</b>	LDAP グループエントリーのどの属性が固有の識別子として解釈されるかを定義します。 ( <b>IdapGroupUID</b> )	文字列
<b>groupNameAttributes</b>	LDAP グループエントリーのどの属性が OpenShift Container Platform Group に使用する名前として解釈されるかを定義します。	文字列の配列
<b>groupMembershipAttributes</b>	LDAP グループエントリーのどの属性がメンバーとして解釈されるかを定義します。それらの属性に含まれる値は <b>UserUIDAttribute</b> でクエリーできる必要があります。	文字列の配列
<b>usersQuery</b>	ユーザーエントリーを返す LDAP クエリーのテンプレートを保持します。	<a href="#">v1.LDAPQuery</a>
<b>userUIDAttribute</b>	LDAP ユーザーエントリーのどの属性が固有の識別子として解釈されるかを定義します。 <b>GroupMembershipAttributes</b> で検出される値に対応している必要があります。	文字列

名前	説明	スキーマ
<b>userNameAttributes</b>	LDAP ユーザーエントリーのどの属性が順番に OpenShift Container Platform ユーザー名として使われるかを定義します。空でない値を持つ最初の属性が使用されます。これは <b>LDAPPasswordIdentityProvider</b> の <b>PreferredUsername</b> 設定と一致している必要があります。OpenShift Container Platform Group レコードでユーザー名として使用される属性です。ほとんどのインストールでは、 <b>mail</b> または <b>sAMAccountName</b> を使用することが推奨されます。	文字列の配列
<b>tolerateMemberNotFoundErrors</b>	ユーザーエントリーがない場合の LDAP 同期ジョブの動作を決定します。 <b>true</b> の場合、何も検出しないユーザーの LDAP クエリーは許容され、エラーのみがログに記録されます。 <b>false</b> の場合、ユーザーのクエリーが何も検出しないと、LDAP 同期ジョブは失敗します。デフォルトの値は <b>false</b> です。 <b>true</b> に設定されたこのフラグを持つ LDAP 同期ジョブの設定が間違っていると、グループメンバーシップが削除されることがあるため、注意してこのフラグを使用してください。	boolean
<b>tolerateMemberOutOfScopeErrors</b>	範囲外のユーザーエントリーが検出される場合の LDAP 同期ジョブの動作を決定します。 <b>true</b> の場合、すべてのユーザークエリーに指定されるベース DN 外のユーザーの LDAP クエリーは許容され、エラーのみがログに記録されます。 <b>false</b> の場合、ユーザークエリーですべてのユーザークエリーで指定されるベース DN 外を検索すると LDAP 同期ジョブは失敗します。このフラグを <b>true</b> に設定した LDAP 同期ジョブの設定が間違っていると、ユーザーのいないグループが発生することがあるため、注意してこのフラグを使用してください。	ブール値

### 14.7.5. v1.ActiveDirectoryConfig

**ActiveDirectoryConfig** は必要な設定オプションを保持し、どのように LDAP グループ同期が Active Directory スキーマを使用して LDAP サーバーと相互作用するかを定義します。

名前	説明	スキーマ
<b>usersQuery</b>	ユーザーエントリを返す LDAP クエリーのテンプレートを保持します。	<a href="#">v1.LDAPQuery</a>
<b>userNameAttributes</b>	LDAP ユーザーエントリのどの属性が OpenShift Container Platform ユーザー名として解釈されるかを定義します。OpenShift Container Platform Group レコードでユーザー名として使用される属性です。ほとんどのインストールでは、 <b>mail</b> または <b>sAMAccountName</b> を使用することが推奨されます。	文字列の配列
<b>groupMembershipAttributes</b>	LDAP ユーザーのどの属性がメンバーの属するグループとして解釈されるかを定義します。	文字列の配列

### 14.7.6. v1.AugmentedActiveDirectoryConfig

**AugmentedActiveDirectoryConfig** は必要な設定オプションを保持し、どのように LDAP グループ同期が拡張された Active Directory スキーマを使用して LDAP サーバーに相互作用するかを定義します。

名前	説明	スキーマ
<b>usersQuery</b>	ユーザーエントリを返す LDAP クエリーのテンプレートを保持します。	<a href="#">v1.LDAPQuery</a>
<b>userNameAttributes</b>	LDAP ユーザーエントリのどの属性が OpenShift Container Platform ユーザー名として解釈されるかを定義します。OpenShift Container Platform Group レコードでユーザー名として使用される属性です。ほとんどのインストールでは、 <b>mail</b> または <b>sAMAccountName</b> を使用することが推奨されます。	文字列の配列
<b>groupMembershipAttributes</b>	LDAP ユーザーのどの属性がメンバーの属するグループとして解釈されるかを定義します。	文字列の配列

名前	説明	スキーマ
<b>groupsQuery</b>	グループエントリを返す LDAP クエリーのテンプレートを保持します。	<a href="#">v1.LDAPQuery</a>
<b>groupUIDAttribute</b>	LDAP グループエントリのどの属性が固有の識別子として解釈されるかを定義します。 ( <b>IdapGroupUID</b> )	文字列
<b>groupNameAttributes</b>	LDAP グループエントリのどの属性が OpenShift Container Platform Group に使用する名前として解釈されるかを定義します。	string array

## 第15章 LDAP フェイルオーバーの設定

OpenShift Container Platform は Lightweight Directory Access Protocol (LDAP) セットアップで使用するための **認証プロバイダー** を提供しますが、接続できるのは単一の LDAP サーバーのみです。OpenShift Container Platform インストール時に、LDAP フェイルオーバーについて System Security Services Daemon (SSSD) を設定し、ある LDAP サーバーが失敗した場合にクラスターにアクセスできるようにします。

この設定には、詳細な設定および通信先となる OpenShift Container Platform の認証サーバー (**リモート Basic 認証サーバー** と呼ばれます) が別途必要となります。メールアドレスなどの追加の属性を OpenShift Container Platform に渡すようにこのサーバーを設定し、それらの属性を Web コンソールで表示できるようにします。

このトピックでは、専用の物理または仮想マシン (VM) のセットアップを実行する方法や、コンテナでの SSSD の設定方法についても説明します。



### 重要

このトピックのすべてのセクションを完了する必要があります。

### 15.1. 基本リモート認証設定の前提条件

- セットアップを始める前に、LDAP サーバーの以下の情報について知っておく必要があります。
  - ディレクトリーサーバーが [FreeIPA](#)、Active Directory、または別の LDAP ソリューションでサポートされているかどうか。
  - LDAP サーバーの Uniform Resource Identifier (URI) (例: `ldap.example.com`)。
  - LDAP サーバーの CA 証明書の場所。
  - LDAP サーバーがユーザーグループの RFC 2307 または RFC2307bis に対応しているかどうか。
- サーバーを準備します。
  - `remote-basic.example.com`: リモート Basic 認証サーバーとして使用する VM。
    - Red Hat Enterprise Linux 7.0 以降などの、このサーバーの SSSD バージョン 1.12.0 を含むオペレーティングシステムを選択します。
  - `openshift.example.com`: OpenShift Container Platform の新規インストール。
    - このクラスターに認証方法を設定することはできません。
    - このクラスターで OpenShift Container Platform を起動することはできません。

### 15.2. 証明書の生成およびリモート BASIC 認証サーバーとの共有

Ansible ホストインベントリーファイル (デフォルトは `/etc/ansible/hosts`) に一覧表示された 1 つ目のマスターホストで以下の手順を実行します。

1. リモート Basic 認証サーバーと OpenShift Container Platform 間の通信を信頼できるものにするために、このセットアップの他のフェーズで使用する Transport Layer Security (TLS) 証明書のセットを作成します。次のコマンドを実行します。

```
# openshift start \
  --public-master=https://openshift.example.com:8443 \
  --write-config=/etc/origin/
```

出力には、`/etc/origin/master/ca.crt` および `/etc/origin/master/ca.key` の署名用証明書が含まれます。

- 署名用証明書を使用してリモート Basic 認証サーバーで使用するキーを生成します。

```
# mkdir -p /etc/origin/remote-basic/
# oc adm ca create-server-cert \
  --cert='/etc/origin/remote-basic/remote-basic.example.com.crt' \
  --key='/etc/origin/remote-basic/remote-basic.example.com.key' \
  --hostnames=remote-basic.example.com \ ❶
  --signer-cert='/etc/origin/master/ca.crt' \
  --signer-key='/etc/origin/master/ca.key' \
  --signer-serial='/etc/origin/master/ca.serial.txt'
```

- ❶ リモート Basic 認証サーバーにアクセスする必要がある、すべてのホスト名およびインターネット IP アドレスのコンマ区切りの一覧です。



#### 注記

生成する証明書ファイルは2年間有効です。この期間は、`--expire-days` および `--signer-expire-days` の値を変更して変更することができますが、セキュリティ上の理由により、730 より大きな値を設定しないでください。



#### 重要

リモート Basic 認証サーバーにアクセスする必要があるすべてのホスト名およびインターネット IP アドレスを一覧表示しない場合、HTTPS 接続は失敗します。

- 必要な証明書およびキーをリモート Basic 認証サーバーにコピーします。

```
# scp /etc/origin/master/ca.crt \
  root@remote-basic.example.com:/etc/pki/CA/certs/

# scp /etc/origin/remote-basic/remote-basic.example.com.crt \
  root@remote-basic.example.com:/etc/pki/tls/certs/

# scp /etc/origin/remote-basic/remote-basic.example.com.key \
  root@remote-basic.example.com:/etc/pki/tls/private/
```

### 15.3. SSSD での LDAP フェイルオーバーの設定

リモート Basic 認証サーバーで以下の手順を実行します。

メールアドレスおよび表示名などの属性を取得し、それらを OpenShift Container Platform に渡して Web インターフェイスに表示できるように SSSD を設定します。以下の手順では、メールアドレスを OpenShift Container Platform に指定するように SSSD を設定します。

1. 必要な SSSD および Web サーバーコンポーネントをインストールします。

```
# yum install -y sssd \
    sssd-dbus \
    realmd \
    httpd \
    mod_session \
    mod_ssl \
    mod_lookup_identity \
    mod_authnz_pam \
    php \
    mod_php
```

2. LDAP サーバーに対してこの VM を認証するように SSSD を設定します。LDAP サーバーが FreeIPA または Active Directory 環境の場合、**realmd** を使用してこのマシンをドメインに参加させることができます。

```
# realm join ldap.example.com
```

より高度なケースの場合は、[システムレベル認証ガイド](#) を参照してください。

3. SSSD を使用して LDAP のフェイルオーバーの状態を使用するには、**ldap\_uri** 行の `/etc/sss/sss.conf` ファイルにその他のエントリーを追加します。FreeIPA に登録されたシステムは DNS SRV レコードを使用してフェイルオーバーを自動的に処理します。
4. `/etc/sss/sss.conf` ファイルの `[domain/DOMAINNAME]` セクションを変更し、この属性を追加します。

```
[domain/example.com]
...
ldap_user_extra_attrs = mail 1
```

- 1** LDAP ソリューションについてのメールアドレスの取得に必要な適切な属性を指定します。IPA の場合は、**mail** を指定します。他の LDAP ソリューションは **email** などの別の属性を使用する可能性があります。

5. `/etc/sss/sss.conf` ファイルの `domain` パラメーターには `[domain/DOMAINNAME]` セクションに一覧表示されているドメイン名のみが含まれていることを確認します。

```
domains = example.com
```

6. メール属性を取得するために Apache パーミッションを付与します。以下の行を `/etc/sss/sss.conf` ファイルの `[ifp]` セクションに追加します。

```
[ifp]
user_attributes = +mail
allowed_uids = apache, root
```

7. すべての変更が適切に適用されていることを確認するには、SSSD を再起動します。

```
$ systemctl restart sssd.service
```

8. ユーザー情報が適切に取得できるかテストします。

```
$ getent passwd <username>
username.*:12345:12345:Example User:/home/username:/usr/bin/bash
```

9. 指定したメール属性がドメインからメールアドレスを返すことを確認します。

```
# dbus-send --print-reply --system --dest=org.freedesktop.sssd.infopipe \
  /org/freedesktop/sss/infopipe org.freedesktop.sssd.infopipe.GetUserAttr \
  string:username \ ①
  array:string:mail ②

method return time=1528091855.672691 sender=:1.2787 -> destination=:1.2795 serial=13
reply_serial=2
array [
  dict entry(
    string "mail"
    variant      array [
      string "username@example.com"
    ]
  )
]
```

- ① LDAP ソリューションでユーザー名を指定します。
- ② 設定した属性を指定します。

10. LDAP ユーザーとして VM へのログインを試行し、LDAP 認証情報を使用してログインできることを確認します。ログインにはローカルコンソールまたは SSH などのリモートサービスを使用できます。

### 重要

デフォルトで、すべてのユーザーは LDAP 認証情報を使用してリモート Basic 認証サーバーにログインできます。この動作は変更することができます。

- IPA に参加したシステムを使用する場合、[ホストベースのアクセス制御を設定](#)します。
- Active Directory に参加したシステムを使用する場合、[グループポリシーオブジェクト](#)を使用します。
- その他のケースについては、[SSSD 設定](#) についてのドキュメントを参照してください。

## 15.4. APACHE での SSSD の使用の設定

1. 以下の内容を含む `/etc/pam.d/openshift` ファイルを作成します。

```
auth required pam_sss.so
account required pam_sss.so
```

この設定により、認証要求が `openshift` スタックに対して発行された時に PAM (プラグ可能な認証モジュール) は `pam_sss.so` を使用して認証とアクセス制御を決定できるようになります。

2. `/etc/httpd/conf.modules.d/55-authnz_pam.conf` ファイルを編集して、以下の行のコメントを解除します。

```
LoadModule authnz_pam_module modules/mod_authnz_pam.so
```

3. Apache `httpd.conf` ファイルをリモート Basic 認証用に設定するには、`openshift-remote-basic-auth.conf` ファイルを `/etc/httpd/conf.d` ディレクトリーに作成します。以下のテンプレートを使用して必要な設定および値を指定します。



### 重要

テンプレートを十分に確認し、その内容を環境に合うようにカスタマイズします。

```
LoadModule request_module modules/mod_request.so
LoadModule php7_module modules/libphp7.so

# Nothing needs to be served over HTTP. This virtual host simply redirects to
# HTTPS.
<VirtualHost *:80>
    DocumentRoot /var/www/html
    RewriteEngine On
    RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
    # This needs to match the certificates you generated. See the CN and X509v3
    # Subject Alternative Name in the output of:
    # openssl x509 -text -in /etc/pki/tls/certs/remote-basic.example.com.crt
    ServerName remote-basic.example.com

    DocumentRoot /var/www/html

    # Secure all connections with TLS
    SSLEngine on
    SSLCertificateFile /etc/pki/tls/certs/remote-basic.example.com.crt
    SSLCertificateKeyFile /etc/pki/tls/private/remote-basic.example.com.key
    SSLCACertificateFile /etc/pki/CA/certs/ca.crt

    # Require that TLS clients provide a valid certificate
    SSLVerifyClient require
    SSLVerifyDepth 10

    # Other SSL options that may be useful
    # SSLCertificateChainFile ...
    # SSLCARevocationFile ...

    # Send logs to a specific location to make them easier to find
    ErrorLog logs/remote_basic_error_log
    TransferLog logs/remote_basic_access_log
    LogLevel warn

    # PHP script that turns the Apache REMOTE_USER env var
    # into a JSON formatted response that OpenShift understands
    <Location /check_user.php>
```

```

# all requests not using SSL are denied
SSLRequireSSL
# denies access when SSLRequireSSL is applied
SSLOptions +StrictRequire
# Require both a valid basic auth user (so REMOTE_USER is always set)
# and that the CN of the TLS client matches that of the OpenShift master
<RequireAll>
  Require valid-user
  Require expr %{SSL_CLIENT_S_DN_CN} == 'system:openshift-master'
</RequireAll>
# Use basic auth since OpenShift will call this endpoint with a basic challenge
AuthType Basic
AuthName openshift
AuthBasicProvider PAM
AuthPAMService openshift

# Store attributes in environment variables. Specify the email attribute that
# you confirmed.
LookupOutput Env
LookupUserAttr mail REMOTE_USER_MAIL
LookupUserGECOS REMOTE_USER_DISPLAY_NAME

# Other options that might be useful

# While REMOTE_USER is used as the sub field and serves as the immutable ID,
# REMOTE_USER_PREFERRED_USERNAME could be used to have a different
username
# LookupUserAttr <attr_name> REMOTE_USER_PREFERRED_USERNAME

# Group support may be added in a future release
# LookupUserGroupsIter REMOTE_USER_GROUP
</Location>

# Deny everything else
<Location ~ "^(?!/check_user\.php)\.*$">
  Deny from all
</Location>
</VirtualHost>

```

4. `check_user.php` スクリプトを `/var/www/html` ディレクトリーに作成します。以下のコードを組み込みます。

```

<?php
// Get the user based on the Apache var, this should always be
// set because we 'Require valid-user' in the configuration
$user = apache_getenv('REMOTE_USER');

// However, we assume it may not be set and
// build an error response by default
$data = array(
  'error' => 'remote PAM authentication failed'
);

// Build a success response if we have a user
if (!empty($user)) {
  $data = array(

```

```

    'sub' => $user
  );
  // Map of optional environment variables to optional JSON fields
  $env_map = array(
    'REMOTE_USER_MAIL' => 'email',
    'REMOTE_USER_DISPLAY_NAME' => 'name',
    'REMOTE_USER_PREFERRED_USERNAME' => 'preferred_username'
  );

  // Add all non-empty environment variables to JSON data
  foreach ($env_map as $env_name => $json_name) {
    $env_data = apache_getenv($env_name);
    if (!empty($env_data)) {
      $data[$json_name] = $env_data;
    }
  }
}

// We always output JSON from this script
header('Content-Type: application/json', true);

// Write the response as JSON
echo json_encode($data);
?>

```

5. Apache がモジュールを読み込めるようにします。/etc/httpd/conf.modules.d/55-lookup\_identity.conf ファイルを変更し、以下の行のコメントを解除します。

```
LoadModule lookup_identity_module modules/mod_lookup_identity.so
```

6. SELinux ブール値を設定し、SELinux が Apache が D-BUS を介して SSSD に接続することを許可するようにします。

```
# setsebool -P httpd_dbus_sssd on
```

7. SELinux に Apache による PAM サブシステムへの問い合わせを受け入れることを指示するブール値を設定します。

```
# setsebool -P allow_httpd_mod_auth_pam on
```

8. Apache を起動します。

```
# systemctl start httpd.service
```

## 15.5. SSSD を基本リモート認証サーバーとして使用するよう OPENSIFT CONTAINER PLATFORM を設定する

作成した新規のアイデンティティプロバイダーを使用するようクラスターのデフォルト設定を変更します。Ansible ホストインベントリーファイルに最初の一覧表示されるマスターホストで以下の手順を実行します。

1. /etc/origin/master/master-config.yaml ファイルを開きます。

2. **identityProviders** セクションの場所を見つけ、これを以下のコードに置き換えます。

```
identityProviders:  
- name: sssd  
  challenge: true  
  login: true  
  mappingMethod: claim  
  provider:  
    apiVersion: v1  
    kind: BasicAuthPasswordIdentityProvider  
    url: https://remote-basic.example.com/check_user.php  
    ca: /etc/origin/master/ca.crt  
    certFile: /etc/origin/master/openshift-master.crt  
    keyFile: /etc/origin/master/openshift-master.key
```

3. 更新された設定を使って OpenShift Container Platform を再起動します。

```
# /usr/local/bin/master-restart api api  
  
# /usr/local/bin/master-restart controllers controllers
```

4. **oc** CLI を使用してログインをテストします。

```
$ oc login https://openshift.example.com:8443
```

有効な LDAP 認証情報のみを使用してログインすることができます。

5. アイデンティティを一覧表示し、各ユーザー名のメールアドレスが表示されていることを確認します。次のコマンドを実行します。

```
$ oc get identity -o yaml
```

## 第16章 SDN の設定

### 16.1. 概要

OpenShift SDN は、OpenShift Container Platform クラスターでの Pod 間の通信を有効にして Pod ネットワークを構築します。現在利用可能な SDN プラグイン は 3 種類 (`ovs-subnet`、`ovs-multitenant` および `ovs-networkpolicy`) あり、これらは Pod ネットワークを設定するためのそれぞれ異なる方法を提供します。

### 16.2. 利用可能な SDN プロバイダー

アップストリームの Kubernetes プロジェクトはデフォルトのネットワークソリューションを備えていません。その代わりに、Kubernetes では Container Network Interface (CNI) を開発し、ネットワークプロバイダーが独自の SDN ソリューションを統合することを可能にしています。

Red Hat は、サードパーティーのプラグインのほかにも追加設定なしで使用できるいくつかの OpenShift SDN プラグインを提供しています。

Red Hat は数多くの SDN プロバイダーと協力し、Kubernetes CNI インターフェイスを使用した OpenShift Container Platform 上での SDN ネットワークソリューション (これには、製品のエンタイトルメントプロセスでの SDN プラグインのサポートプロセスが含まれます) の認定を行っています。Red Hat は、ユーザーが OpenShift でサポートケースを作成される場合、両社が共にユーザーのニーズに対応できるように交換プロセスを促進し、これを容易にできます。

以下は、サードパーティーのベンダーが OpenShift Container Platform で直接検証を行い、サポートしている SDN ソリューションです。

- Cisco ACI (™)
- Juniper Contrail (™)
- Nokia Nuage (™)
- Tigera Calico (™)
- VMware NSX-T (™)

#### VMware NSX-T (™) の OpenShift Container Platform へのインストール

VMware NSX-T (™) は、クラウドネイティブなアプリケーション環境を構築するための SDN およびセキュリティ基盤を提供しています。これらの環境には、vSphere Hypervisors (ESX) のほかに KVM とネイティブなパブリッククラウドが含まれます。

現在の統合には、NSX-T と OpenShift Container Platform の両方の新規インストールが必要です。現時点で、NSX-T バージョン 2.4 がサポートされ、これは ESX と KVM のハイパーバイザーの使用のみに対応しています。

詳細は [NSX-T Container Plug-in for OpenShift - Installation and Administration Guide](#) を参照してください。

### 16.3. ANSIBLE を使用した POD ネットワークの設定

初回のクラスターインストールでは、`ovs-subnet` プラグインはデフォルトでインストールされ、設定されますが、このプラグインは、`os_sdn_network_plugin_name` パラメーターを使ってインストール時に上書きされる可能性があります。これは Ansible イベントリーフファイルで設定できます。

たとえば、標準の `ovs-subnet` プラグインを上書きし、代わりに `ovs-multitenant` プラグインを使用するには、以下を実行します。

```
# Configure the multi-tenant SDN plugin (default is 'redhat/openshift-ovs-subnet')
os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
```

インベントリーファイルに設定できるネットワーク関連の Ansible 変数の詳細については、[クラスター変数の設定](#) を参照してください。

## 16.4. マスターでの POD ネットワークの設定

クラスター管理者は、[マスター設定ファイル](#) (デフォルトの場所は `/etc/origin/master/master-config.yaml`) の `networkConfig` セクションにあるパラメーターを変更することで、マスターホスト上で Pod ネットワーク設定を管理できます。

### 単一 CIDR の Pod ネットワーク設定

```
networkConfig:
  clusterNetworks:
    - cidr: 10.128.0.0/14 ①
      hostSubnetLength: 9 ②
  networkPluginName: "redhat/openshift-ovs-subnet" ③
  serviceNetworkCIDR: 172.30.0.0/16 ④
```

- ① ノード IP の割り当て用のクラスターネットワーク
- ② ノード内での Pod IP の割り当て用のビットの数
- ③ `ovs-subnet` プラグインに `redhat/openshift-ovs-subnet`、`ovs-multitenant` プラグインに `redhat/openshift-ovs-multitenant`、`ovs-networkpolicy` プラグインに `redhat/openshift-ovs-networkpolicy` をそれぞれ設定します。
- ④ クラスターに割り当てられるサービス IP

また、複数の CIDR 範囲を持つ Pod ネットワークを作成することもできます。これは、個別の範囲をその範囲と `hostSubnetLength` を指定した `clusterNetworks` フィールドに追加して実行できます。

複数の範囲は同時に使用することができ、この範囲は拡張または縮小することが可能です。ノードは、ノードの退避、削除および再作成によってある範囲から別の範囲に移動できます。詳細は、[ノードの管理](#) セクションを参照してください。ノードの割り当ては一覧の順序で行われ、範囲が一杯になると一覧の次の項目に移行します。

### 複数 CIDR の Pod ネットワークの設定

```
networkConfig:
  clusterNetworks:
    - cidr: 10.128.0.0/14 ①
      hostSubnetLength: 9 ②
    - cidr: 10.132.0.0/14
      hostSubnetLength: 9
  externalIPNetworkCIDRs: null
  hostSubnetLength: 9
```

```
ingressIPNetworkCIDR: 172.29.0.0/16
networkPluginName: redhat/openshift-ovs-multitenant ③
serviceNetworkCIDR: 172.30.0.0/16
```

- ① ノード IP の割り当て用クラスターネットワーク。
- ② ノード内での Pod IP の割り当て用のビットの数
- ③ `ovs-subnet` プラグインの場合は `redhat/openshift-ovs-subnet` に、`ovs-multitenant` プラグインの場合は `redhat/openshift-ovs-multitenant` に、`ovs-networkpolicy` プラグインの場合は `redhat/openshift-ovs-networkpolicy` に設定します。

`clusterNetworks` の値に要素を追加するか、またはノードがその CIDR 範囲を使用していない場合はそれを削除することができます。

### 重要

`hostSubnetLength` の値は、クラスターの初回作成後に変更することができません。`cidr` は、ノードが範囲内に割り当てられている場合に最初のネットワークが含まれるより大きいネットワークにのみ変更することができ、`serviceNetworkCIDR` のみを拡張できます。たとえば、値が一般的な `10.128.0.0/14` の場合、`cidr` を `10.128.0.0/9` (上半分の net 10 を参照) に変更することは可能ですが、`10.64.0.0/16` には元の値と重複しないために変更することができません。

`serviceNetworkCIDR` は `172.30.0.0/16` から `172.30.0.0/15` に変更できますが、`172.28.0.0/14` に変更できません。詳細は、[サービスネットワークの拡張](#) を参照してください。

API およびマスターサービスを再起動して変更を有効にしてください。

```
$ master-restart api
$ master-restart controllers
```

### 重要

ノードの Pod ネットワーク設定は、マスターの `networkConfig.clusterNetworks` パラメーターで設定される Pod ネットワーク設定と同じである必要があります。これには、[ノード設定マップ](#) の `networkConfig` セクションのパラメーターを変更します。

```
proxyArguments:
  cluster-cidr:
    - 10.128.0.0/12 ①
```

- ① CIDR の値は、マスターレベルで定義されるすべてのクラスターネットワーク CIDR 範囲に対応する必要がありますが、ノードやサービスなどの他の IP 範囲と競合しないようにしてください。

マスターサービスを再起動したら、設定をノードに伝播する必要があります。各ノードで `atomic-openshift-node` サービスおよび `ovs Pod` を再起動する必要があります。ダウンタイムを回避するには、[ノードの管理](#) に定義されている手順と、ノードまたはノードのグループについてそれぞれ以下の方法に記載されている手順を実行します。

1. ノードにスケジューラ対象外 (`unschedulable`) のマークを付けます。

```
# oc adm manage-node <node1> <node2> --schedulable=false
```

2. ノードをドレイン (解放) します。

```
# oc adm drain <node1> <node2>
```

3. ノードを再起動します。

```
# reboot
```

4. ノードを再度スケジュール対象としてマークします。

```
# oc adm manage-node <node1> <node2> --schedulable
```

## 16.5. クラスターネットワークの VXLAN ポートの変更

クラスター管理者として、システムが使用する VXLAN ポートを変更できます。

実行中の **clusternetwork** オブジェクトの VXLAN ポートを変更することはできないため、既存のネットワーク設定を削除し、マスター設定ファイルの **vxlanPort** 変数を編集して新規設定を作成する必要があります。

1. 既存の **clusternetwork** を削除します。

```
# oc delete clusternetwork default
```

2. デフォルトで `/etc/origin/master/master-config.yaml` に置かれているマスター設定ファイルを編集し、新規の **clusternetwork** を作成します。

```
networkConfig:
  clusterNetworks:
    - cidr: 10.128.0.0/14
      hostSubnetLength: 9
    - cidr: 10.132.0.0/14
      hostSubnetLength: 9
  externalIPNetworkCIDRs: null
  hostSubnetLength: 9
  ingressIPNetworkCIDR: 172.29.0.0/16
  networkPluginName: redhat/openshift-ovs-multitenant
  serviceNetworkCIDR: 172.30.0.0/16
  vxlanPort: 4889 ❶
```

- ❶ VXLAN ポートのノードで使用される値に設定されます。1-65535 の範囲の整数になります。デフォルト値は **4789** です。

3. 新規ポートを各クラスターノードの iptables ルールに追加します。

```
# iptables -A OS_FIREWALL_ALLOW -p udp -m state --state NEW -m udp --dport 4889 -j ACCEPT ❶
```

- ❶ **4889** はマスター設定ファイルに指定する **vxlanPort** 値です。

4. マスターサービスを再起動します。

```
# master-restart api
# master-restart controllers
```

5. 古い SDN Pod を削除し、新規の変更と共に新規 Pod を伝播します。

```
# oc delete pod -l app=sdn -n openshift-sdn
```

## 16.6. ノードでの POD ネットワークの設定

クラスター管理者は、[ノード設定ファイル](#) の **networkConfig** セクションにあるパラメーターを変更することで、ノード上の Pod ネットワーク設定を制御できます。

```
networkConfig:
  mtu: 1450 ①
  networkPluginName: "redhat/openshift-ovs-subnet" ②
```

- ① Pod オーバーレイネットワーク用の最大転送単位 (MTU)

- ② **ovs-subnet** プラグインに **redhat/openshift-ovs-subnet**、**ovs-multitenant** プラグインに **redhat/openshift-ovs-multitenant**、**ovs-networkpolicy** プラグインに **redhat/openshift-ovs-networkpolicy** をそれぞれ設定します。



### 注記

OpenShift Container Platform SDN を設定するすべてのマスターおよびノードで MTU サイズを変更する必要があります。また、`tun0` インターフェイスの MTU サイズはクラスターを設定するすべてのノードで同一である必要があります。

## 16.7. サービスネットワークの拡張

サービスネットワークのアドレスの低い位置で実行している場合、現在の範囲が新規の範囲の開始地点にある限り、範囲を拡張することができます。



### 注記

サービスネットワークは、拡張のみが可能で、変更や縮小はできません。

1. すべてのマスターの設定ファイル (デフォルトでは `/etc/origin/master/master-config.yaml`) の **serviceNetworkCIDR** および **servicesSubnet** パラメーターを変更します。/`/`の後の番号のみをより小さな番号に変更します。
2. **clusterNetwork** デフォルトのオブジェクトを削除します。

```
$ oc delete clusternetwork default
```

3. すべてのマスターでコントローラーコンポーネントを再起動します。

```
# master-restart controllers
```

4. Ansible インベントリーファイルの **openshift\_portal\_net** 変数の値を新規の CIDR に更新します。

```
# Configure SDN cluster network and kubernetes service CIDR blocks. These
# network blocks should be private and should not conflict with network blocks
# in your infrastructure that pods may require access to. Can not be changed
# after deployment.
openshift_portal_net=172.30.0.0/<new_CIDR_range>
```

クラスター内のノードごとに、以下の手順を実行します。

1. ノードにスケジューリング対象外 (unschedulable) のマークを付けます。
2. ノードから Pod を退避します。
3. ノードを再起動します。
4. ノードが再び利用可能になった後に、ノードを再度スケジューリング対象としてマークします。

## 16.8. SDN プラグイン間の移行

SDN プラグインをすでに1つ使用していて、別のプラグインへ切り替える場合には、以下を実行します。

1. 設定ファイル内のすべての **マスター** と **ノード** で **networkPluginName** パラメーターを変更します。
2. すべてのマスターで API およびマスターサービスを再起動します。

```
# master-restart api
# master-restart controllers
```

3. すべてのマスターおよびノードでノードサービスを停止します。

```
# systemctl stop atomic-openshift-node.service
```

4. OpenShift SDN プラグイン間の切り換えを行う場合、すべてのマスターおよびノードで OpenShift SDN を再起動します。

```
oc delete pod --all -n openshift-sdn
```

5. すべてのマスターおよびノードでノードサービスを再起動します。

```
# systemctl restart atomic-openshift-node.service
```

6. OpenShift SDN プラグインからサードパーティーのプラグインへ切り替える場合は、OpenShift SDN 固有のアーティファクトを消去します。

```
$ oc delete clusternetwork --all
$ oc delete hostsubnets --all
$ oc delete netnamespaces --all
```



## 重要

さらに、**ovs-multitenant** に切り替えると、ユーザーはサービスカタログを使用してサービスをプロビジョニングできなくなります。**openshift-monitoring** も同様です。これを修正するには、以下のプロジェクトをグローバルにします。

```
$ oc adm pod-network make-projects-global kube-service-catalog
$ oc adm pod-network make-projects-global openshift-monitoring
```

この問題は、クラスターが最初に **ovs-multitenant** でインストールされている場合には表示されません。これらのコマンドが Ansible Playbook の一部として実行されたためです。



## 注記

**ovs-subnet** から **ovs-multitenant** OpenShift SDN プラグインに切り替えると、クラスター内の既存のすべてのプロジェクトが完全に分離します (つまり、これらに固有の VNID が割り当てられます)。クラスター管理者は、管理者 CLI を使用して [プロジェクトネットワークの変更](#) を選択できます。

以下を実行して VNID をチェックします。

```
$ oc get netnamespace
```

### 16.8.1. ovs-multitenant から ovs-networkpolicy への移行



## 注記

**v1 NetworkPolicy** 機能のみが OpenShift Container Platform で利用可能です。つまり、egress ポリシータイプ、IPBlock、および **podSelector** と **namespaceSelector** の組み合わせは OpenShift Container Platform では使用できません。



## 注記

**NetworkPolicy** 機能はクラスターとの通信に障害を発生させる可能性があるため、これらの機能をデフォルトの OpenShift Container Platform プロジェクトに適用しないでください。

[SDN プラグインセクション間の移行](#) での上記の一般的なプラグインの移行に加え、**ovs-multitenant** プラグインから **ovs-networkpolicy** プラグインへの移行にはもう1つの追加の手順があります。すべての namespace には **NetID** がなければなりません。これは、以前に [プロジェクトを結合](#) している場合や、[プロジェクトをグローバル化](#) している場合に、**ovs-networkpolicy** プラグインに切り換える前にやり直しを実行する必要があります。そうしない場合には、NetworkPolicy オブジェクトが適切に機能しなくなる可能性があります。

ヘルパースクリプトを使うと、**NetID's** の修正、以前に分離した namespace を分離するための NetworkPolicy オブジェクトの作成、および以前に結合した namespace 間の接続の有効化を実行できます。

**ovs-multitenant** プラグインを実行した状態でヘルパースクリプトを使用して **ovs-networkpolicy** プラグインを移行するには、以下の手順に従ってください。

1. スクリプトをダウンロードし、実行ファイルパーミッションを追加します。

```
$ curl -O https://raw.githubusercontent.com/openshift/origin/release-3.11/contrib/migration/migrate-network-policy.sh
$ chmod a+x migrate-network-policy.sh
```

2. スクリプトを実行します (クラスター管理者ロールが必要です)。

```
$ ./migrate-network-policy.sh
```

このスクリプトを実行すると、すべての namespace が他のすべての namespace から完全に分離されるので、**ovs-networkpolicy** プラグインへの移行が完了するまでは、異なる namespace にある Pod 間で接続を試行してもこれに失敗します。

新たに作成した namespace に同じポリシーをデフォルトで適用したい場合には、移行スクリプトで作成した **default-deny** および **allow-from-global-namespaces** ポリシーと一致する **デフォルトの NetworkPolicy オブジェクト** が作成されるように設定します。



### 注記

スクリプトが失敗するか他のエラーが発生した場合、または **ovs-multitenant** プラグインに戻りたい場合は、**移行取り消し (un-migration) スクリプト** を使用します。このスクリプトは移行スクリプトが実行した変更を取り消し、以前に結合した namespace を再度結合します。

## 16.9. クラスターネットワークへの外部アクセス

OpenShift Container Platform の外部にあるホストがクラスターネットワークへのアクセスを要求した場合、ユーザーには 2 つの選択肢があります。

1. ホストを OpenShift Container Platform ノードとして設定する。ただし、これには **スケジューラ対象外 (unschedulable)** のマークを付け、マスターがこれにコンテナをスケジューラできないようにします。
2. ユーザーのホストとクラスターネットワーク上のホストの間にトンネルを作成する。

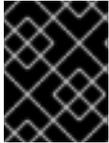
上記のオプションはどちらも **OpenShift SDN 内での edge ロードバランサーからコンテナへのルーティング** を設定するための実践的ユースケースの一部として本ドキュメントで紹介されています。

## 16.10. FLANNEL の使用

デフォルトの SDN の代替として、OpenShift Container Platform は、**flannel** ベースのネットワークをインストールするための Ansible Playbook を提供しています。これは、SDN に依存しているクラウドプロバイダーのプラットフォーム (Red Hat OpenStack Platform など) で OpenShift Container Platform を実行していて、両方のプラットフォームでパケットのカプセル化を 2 回実行することを避けたい場合に役立ちます。

Flannel は、すべてのコンテナに単一の IP ネットワーク空間を使用し、隣接する空間のサブセットを各インスタンスに割り当てることを可能にします。これにより、コンテナは何にも妨げられずに同じネットワーク空間内の任意の IP へ接続を試みるすることができます。これはネットワークを使ってアプリケーション内にあるコンテナを別のアプリケーションから分離することができないために、マルチテナンシーを妨げます。

マルチテナンシーの分離とパフォーマンスのどちらを優先するかに応じて、内部ネットワークに OpenShift SDN (マルチテナンシー) と Flannel (パフォーマンス) のいずれか適切な方を選択する必要があります。



## 重要

Flannel は、Red Hat OpenStack Platform の OpenShift Container Platform のみでサポートされています。



## 重要

Neutron の現行バージョンは、各ポートに対するポートセキュリティーをデフォルトで実施します。これにより、ポートでのポート上のアドレスと異なる MAC アドレスを使ったパケットの送信または受信を実行できません。Flannel は、仮想の MAC アドレスと IP アドレスを作成し、パケットをポート上で送受信できる必要があります。したがって、Flannel のトラフィックを実行するポートでは、ポートセキュリティーが無効にされている必要があります。

OpenShift Container Platform クラスターで Flannel を無効にするには、以下を実行します。

1. Neutron のポートセキュリティーコントロールは、Flannel と互換性を持つように設定される必要があります。Red Hat OpenStack Platform のデフォルト設定では、**port\_security** のユーザーコントロールは無効にされています。Neutron を、ユーザーが個々のポートで **port\_security** 設定を制御できるように設定します。
  - a. Neutron サーバーで、以下を `/etc/neutron/plugins/ml2/ml2_conf.ini` ファイルに追加します。

```
[ml2]
...
extension_drivers = port_security
```

- b. 次に、Neutron のサービスを再起動します。

```
service neutron-dhcp-agent restart
service neutron-ovs-cleanup restart
service neutron-metadata-agent restart
service neutron-l3-agent restart
service neutron-plugin-openvswitch-agent restart
service neutron-vpn-agent restart
service neutron-server restart
```

2. Red Hat OpenStack Platform で OpenShift Container Platform インスタンスが作成されている間に、ポートのポートセキュリティーとセキュリティーグループの両方を無効にします。ここで、コンテナネットワークの Flannel インターフェイスは以下のようになります。

```
neutron port-update $port --no-security-groups --port-security-enabled=False
```



## 注記

Flannel は、ノードのサブネットを設定し、割り当てるために etcd からの情報を収集します。したがって、etcd ホストに割り当てられるセキュリティーグループは、ノードからポート 2379/tcp へのアクセスを許可し、ノードのセキュリティーグループは etcd ホストでのそのポートへの egress 通信を許可する必要があります。

- a. インストールを実行する前に以下の変数を Ansible インベントリーファイルに設定します。

■

```
openshift_use_openshift_sdn=false ❶
openshift_use_flannel=true ❷
flannel_interface=eth0
```

- ❶ **openshift\_use\_openshift\_sdn** を **false** に設定してデフォルトのSDNを無効にします。
- ❷ **openshift\_use\_flannel** を **true** に設定して設定されている **flannel** を有効にします。

- b. オプションで、**flannel\_interface** 変数を使用してホスト間の通信に使用するインターフェイスを指定することができます。この変数がない場合は、OpenShift Container Platform インストールではデフォルトのインターフェイスが使用されます。



### 注記

Flannel を使用した Pod とサービスのカスタムのネットワーク CIDR は、今後のリリースでサポートされる予定です。 [BZ#1473858](#)

3. OpenShift Container Platform をインストールした後、iptables の一連のルールを各 OpenShift Container Platform ノードに追加します。

```
iptables -A DOCKER -p all -j ACCEPT
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

これらの変更を `/etc/sysconfig/iptables` で永続化するには、すべてのノードで以下のコマンドを使用します。

```
cp /etc/sysconfig/iptables{.,orig}
sh -c "tac /etc/sysconfig/iptables.orig | sed -e '0,/:DOCKER -/ s/:DOCKER -/:DOCKER ACCEPT/' | awk '!\"p && /POSTROUTING/{print \"-A POSTROUTING -o eth1 -j MASQUERADE\"; p=1} 1' | tac > /etc/sysconfig/iptables"
```



### 注記

**iptables-save** コマンドは、現在の **インメモリー** の iptables ルールをすべて保存します。ただし、Docker、Kubernetes、OpenShift Container Platform では永続化することが意図されていない iptables ルール (サービスなど) が多数作成されるために、これらのルールを保存すると問題が発生する可能性があります。

コンテナのトラフィックを OpenShift Container Platform の残りのトラフィックから分離するには、分離されたテナントネットワークを作成し、すべてのノードをこれに割り当てることを推奨します。異なるネットワークインターフェイス (eth1) を使用している場合は、インターフェイスをブート時に `/etc/sysconfig/network-scripts/ifcfg-eth1` ファイルを使用して起動するように設定してください。

```
DEVICE=eth1
TYPE=Ethernet
BOOTPROTO=dhcp
ONBOOT=yes
DEFROUTE=no
PEERDNS=no
```

## 第17章 NUAGE SDN の設定

### 17.1. NUAGE SDN と OPENSIFT CONTAINER PLATFORM

Nuage Networks Virtualized Services Platform (VSP) は、仮想ネットワークとソフトウェアによるネットワーク制御 (SDN) インフラストラクチャーをコンテナ環境に提供し、IT 運用を単純化して OpenShift Container Platform のネイティブなネットワーク機能を拡張します。

Nuage Networks VSP は、OpenShift Container Platform で実行される Docker ベースのアプリケーションに対応し、Pod と従来のワークロード間の仮想ネットワークのプロビジョニングを加速化し、セキュリティポリシーをクラウドインフラストラクチャー全体で有効にします。また、セキュリティアプライアンスによる、コンテナアプリケーション用の詳細なセキュリティとマイクロセグメンテーションポリシーの組み込みを自動化します。

VSP を OpenShift Container Platform のアプリケーションワークフローに統合することにより、DevOps チームが直面するネットワークのラグを取り除き、ビジネスアプリケーションのより迅速な調整と更新を可能にします。また、VSP は OpenShift Container Platform のさまざまなワークフローに対応し、ユーザーがポリシーベースの自動化によって使いやすさと完全な制御のいずれかを選択できる各種シナリオに対応します。

VSP を OpenShift Container Platform に統合する方法についての詳細は、[Networking](#) を参照してください。

### 17.2. 開発者のワークフロー

ワークフローは開発者環境で使用され、ネットワークのセットアップ時に開発者のインプットはほとんど必要ありません。ここでは `nuage-openshift-monitor` は、OpenShift Container Platform プロジェクトで作成される Pod の適切なポリシーとネットワークを提供するために必要な、VSP 設定 (ゾーン、サブネットなど) を作成します。プロジェクトが作成されると、`nuage-openshift-monitor` がそのプロジェクトのデフォルトのゾーンとデフォルトのサブネットを作成します。指定のプロジェクトに作成されたデフォルトのサブネットが使い果たされると、`nuage-openshift-monitor` が追加のサブネットを動的に作成します。



#### 注記

プロジェクト間の分離を確保するため、各 OpenShift Container Platform プロジェクトに個別の VSP ゾーンが作成されます。

### 17.3. オペレーションワークフロー

このワークフローは、アプリケーションをロールアウトするオペレーションチームが使用します。このワークフローでは、まずネットワークとセキュリティポリシーが、アプリケーションをデプロイするために組織が規定するルールに従って VSD に設定されます。管理ユーザーは、複数のゾーンとサブネットを作成し、ラベルを使ってそれらを同じプロジェクトにマップすることがあります。Pod をスピンアップする一方で、ユーザーは、Pod が接続すべきネットワークと、ポリシーが適用されるべきネットワークを Nuage Label を使って指定します。これにより、デプロイメントでプロジェクト間およびプロジェクト内のトラフィックを詳細に制御できます。たとえば、プロジェクト間の通信はプロジェクトベースで有効にされます。これは、共有プロジェクトにデプロイされた共通サービスに、プロジェクトを接続するために使用できます。

### 17.4. インストールシステム

VSP と OpenShift Container Platform の統合は、仮想マシン (VM) とベアメタルの OpenShift Container Platform インストールの両方で機能します。

高可用性 (HA) を備えた環境は、複数マスターと複数ノードを使って設定することができます。

マルチマスターモードによる Nuage VSP 統合は、このセクションで説明するネイティブの HA 設定方法のみに対応しています。この統合は、負荷分散ソリューション (デフォルトは HAProxy) と組み合わせることもできます。インベントリーファイルには、3つのマスターホスト、ノード、etcd サーバー、およびすべてのマスターホスト上でマスター API の負荷を分散するための HAProxy として機能するホストが含まれます。HAProxy ホストはインベントリーファイルの [lb] セクションに定義され、Ansible が HAProxy を負荷分散ソリューションとして自動的にインストールし、設定することを可能にします。

Ansible ノードファイルでは、Nuage VSP をネットワークプラグインとしてセットアップするために、以下のパラメーターを指定する必要があります。

```
# Create and OSEv3 group that contains masters, nodes, load-balancers, and etcd hosts
masters
nodes
etcd
lb

# Nuage specific parameters
openshift_use_openshift_sdn=False
openshift_use_nuage=True
os_sdn_network_plugin_name='nuage/vsp-openshift'
openshift_node_proxy_mode='userspace'

# VSP related parameters
vsd_api_url=https://192.168.103.200:8443
vsp_version=v4_0
enterprise=nuage
domain=openshift
vsc_active_ip=192.168.103.201
vsc_standby_ip=192.168.103.202
uplink_interface=eth0

# rpm locations
nuage_openshift_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/nuage-openshift-
monitor-4.0.X.1830.el7.centos.x86_64.rpm
vrs_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/nuage-openvswitch-
4.0.X.225.el7.x86_64.rpm
plugin_rpm=http://location_of_rpm_server/openshift/RPMS/x86_64/vsp-openshift-
4.0.X1830.el7.centos.x86_64.rpm

# Required for Nuage Monitor REST server and HA
openshift_master_cluster_method=native
openshift_master_cluster_hostname=lb.nuageopenshift.com
openshift_master_cluster_public_hostname=lb.nuageopenshift.com
nuage_openshift_monitor_rest_server_port=9443

# Optional parameters
nuage_interface_mtu=1460
nuage_master_adminusername='admin's user-name'
nuage_master_adminuserpasswd='admin's password'
nuage_master_cspadminpasswd='csp admin password'
```

```
nuage_openshift_monitor_log_dir=/var/log/nuage-openshift-monitor
```

```
# Required for brownfield install (where a {product-title} cluster exists without Nuage as the  
networking plugin)
```

```
nuage_dockker_bridge=lbr0
```

```
# Specify master hosts
```

```
[masters]
```

```
fqdn_of_master_1
```

```
fqdn_of_master_2
```

```
fqdn_of_master_3
```

```
# Specify load balancer host
```

```
[lb]
```

```
fqdn_of_load_balancer
```

## 第18章 NSX-T SDN の設定

### 18.1. NSX-T SDN および OPENSIFT CONTAINER PLATFORM

VMware NSX-T Data Center™ は、高度な SDN (software-defined networking)、セキュリティー、および IT 操作を単純化し、ネイティブの OpenShift Container Platform ネットワーク機能を拡張するコンテナ環境への可視化を提供します。

NSX-T Data Center は、複数のクラスターにまたがる仮想マシン、ベアメタル、およびコンテナのワークロードをサポートします。これにより、組織は、単一 SDN を使用して環境全体を完全に可視化できます。

NSX-T を OpenShift Container Platform に統合する方法についての詳細は、[NSX-T SDN in Available SDN plug-ins](#) を参照してください。

### 18.2. トポロジーの例

1つの典型的なユースケースとして、物理システムを仮想環境に接続する Tier-0 (T0) ルーターと、OpenShift Container Platform 仮想マシンのデフォルトゲートウェイとして機能する Tier-1 (T1) ルーターを設定します。

各仮想マシンには、vNIC が 2 つあります。1 つは、仮想マシンにアクセスするために Management Logical Switch に接続する vNIC です。もう 1 つは、Dump Logical Switch に接続する vNIC で、Pod ネットワークにアップリンクするために、**nsx-node-agent** が使用します。詳細は、[NSX Container Plug-in for OpenShift](#) を参照してください。

OpenShift Container Platform ルートの設定に使用される LoadBalancer およびすべてのプロジェクトの T1 ルーターおよび論理スイッチは、OpenShift Container Platform のインストール時に自動的に作成されます。

このトポロジーでは、デフォルトの OpenShift Container Platform HAProxy ルーターは、Grafana、Prometheus、Console、Service Catalog その他のインフラストラクチャーコンポーネントすべてに使用されます。HAProxy はホストネットワーク namespace を使用するため、インフラストラクチャーコンポーネントの DNS レコードがインフラストラクチャーノードの IP アドレスをポイントすることを確認します。これはインフラストラクチャールートでは機能しますが、インフラストラクチャーノード管理 IP を外部に公開することを防ぐため、アプリケーション固有のルートを NSX-T LoadBalancer にデプロイします。

このトポロジーの例では、3 つの OpenShift Container Platform マスター仮想マシンと 4 つの OpenShift Container Platform ワーカー仮想マシン (インフラストラクチャー用に 2 つ、コンピューター用に 2 つのワーカー仮想マシン) を使用していることを想定しています。

### 18.3. VMWARE NSX-T のインストール

#### 前提条件

- ESXi ホストの要件:
  - OpenShift Container Platform ノード仮想マシンをホストする ESXi サーバーは NSX-T トランスポートノードである必要があります。

図18.1 NSX UI での通常の高可用性環境のトランスポートノード表示

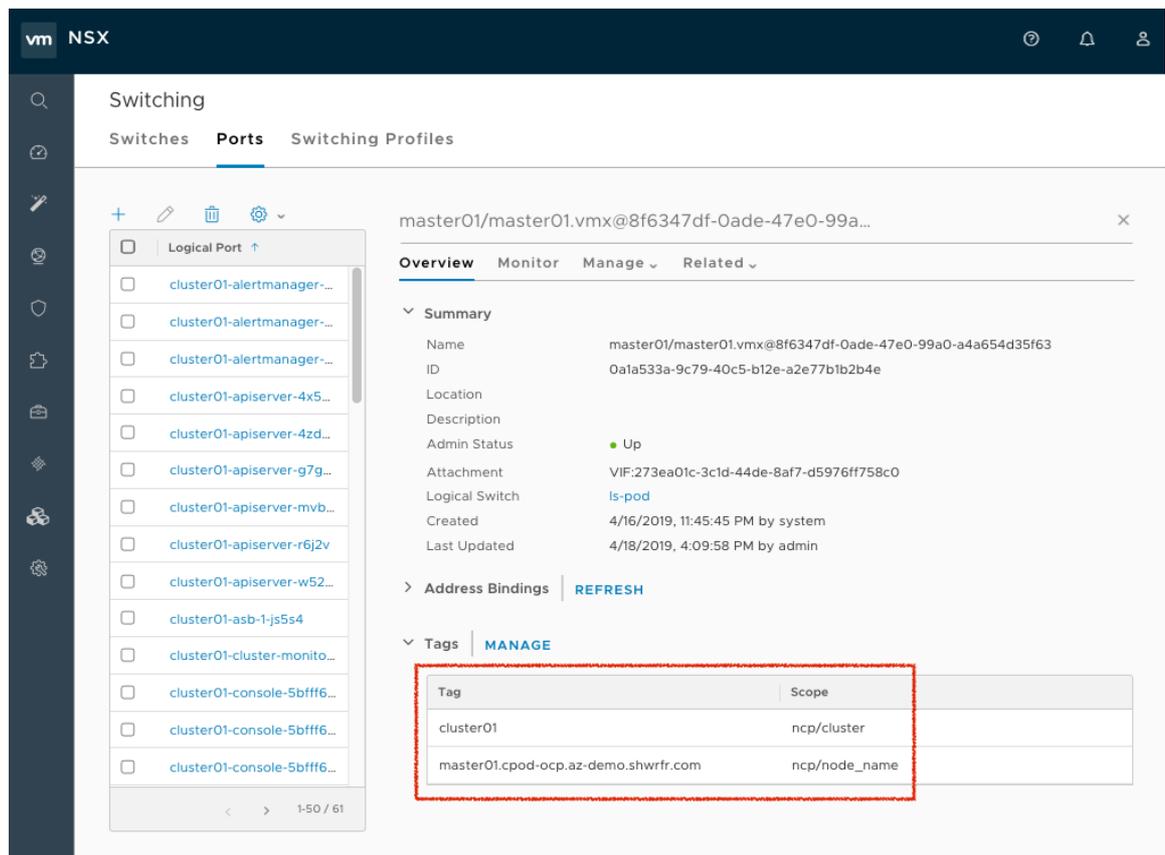
Transport Node	ID	N-VDS	Configuration Stat	Status	IP Addresses	Fabric Node Type	Transport Zones	NSX Version
esx-01.cpod-ocp-az-demo.shwrfr.com	a4c6...47b5	1	Success	Up	172.19.4.21	Host - ESXi 6.7.0	TZ-Overlay	2.3.1.0.0.1129...
esx-02.cpod-ocp-az-demo.shwrfr.com	a5f9...5653	1	Success	Up	172.19.4.22	Host - ESXi 6.7.0	TZ-Overlay	2.3.1.0.0.1129...
esx-03.cpod-ocp-az-demo.shwrfr.com	8f63...5f63	1	Success	Up	172.19.4.23	Host - ESXi 6.7.0	TZ-Overlay	2.3.1.0.0.1129...
esx-04.cpod-ocp-az-demo.shwrfr.com	b8aa...a5f7	1	Success	Up	172.19.4.24	Host - ESXi 6.7.0	TZ-Overlay	2.3.1.0.0.1129...
nsxedg-01	3868...cb2d	2	Success	Up	172.19.4.54	Edge - Virtual Machine	TZ-Overlay TZ-VLAN	2.3.1.0.0.1129...
nsxedg-02	3fbb...2962	2	Success	Up	172.19.4.55	Edge - Virtual Machine	TZ-Overlay TZ-VLAN	2.3.1.0.0.1129...

- DNS の要件:
  - ワイルドカードのある DNS サーバーの新規エントリーをインフラストラクチャーノードに追加する必要があります。これにより、NSX-T または他のサードパーティー LoadBalancer による負荷分散が許可されます。以下の **hosts** ファイルで、エントリーは **openshift\_master\_default\_subdomain** 変数によって定義されます。
  - DNS サーバーは、**openshift\_master\_cluster\_hostname** および **openshift\_master\_cluster\_public\_hostname** 変数で更新する必要があります。
- 仮想マシンの要件:
  - OpenShift Container Platform ノード仮想マシンには 2 つの vNIC が必要です。
  - Management vNIC が管理 T1 ルーターのアップリンクに使用される論理スイッチに接続されている必要があります。
  - すべての仮想マシンの 2 つ目の vNIC は NSX-T でタグ付けし、NSX Container Plug-in (NCP) が特定の OpenShift Container Platform ノードで実行されているすべてのポートの親 VIF として使用する必要のあるポートを識別できるようにする必要があります。タグは以下のようになります。

```
{'ncp/node_name': 'node_name'}
{'ncp/cluster': 'cluster_name'}
```

以下の図では、NSX UI でのすべてのノードについてのタグを示しています。大規模なクラスターの場合、API 呼び出しを使用するか、または Ansible を使用してタグ付けを自動化することができます。

図18.2 NSX UI でのノードタグの表示



NSX UI でのタグの順序は API とは反対になります。ノード名は kubelet が予想するものと完全に一致する必要があり、クラスター名は以下に示すように Ansible ホストファイルの **nsx\_openshift\_cluster\_name** と同じである必要があります。適切なタグがすべてのノードの 2 つ目の vNIC に適用されていることを確認してください。

- NSX-T の要件:  
以下の前提条件を NSX で満たしている必要があります。
  - Tier-0 ルーター。
  - オーバーレイトランポートゾーン。
  - POD ネットワークの IP ブロック。
  - (オプション) ルート指定された (NoNAT) POD ネットワークの IP ブロック
  - SNAT の IP プール。デフォルトで Pod ネットワークの IP ブロックからプロジェクトごとに指定されるサブネットは NSX-T 内でのみルーティング可能です。NCP はこの IP プールを使用して外部への接続を提供します。
  - (オプション) dFW (分散ファイアウォール) の Top および Bottom ファイアウォールセクション。NCP は 2 つのセクション間に Kubernetes ネットワークポリシールールを配置します。
  - Open vSwitch および CNI プラグイン RPM は OpenShift Container Platform ノード仮想マシンから到達可能な HTTP サーバーでホストされる必要があります (この例では <http://websrv.example.com>)。それらのファイルは、[Download NSX Container Plug-in 2.4.0](#) からダウンロードできる NCP Tar ファイルに含まれています。
- OpenShift Container Platform の要件:

- 以下のコマンドを実行して、OpenShift Container Platform の必要なソフトウェアパッケージをインストールします (ある場合)。

```
$ ansible-playbook -i hosts openshift-ansible/playbooks/prerequisites.yml
```

- NCP コンテナイメージがすべてのノードにローカルにダウンロードされていることを確認します。
- **prerequisites.yml** Playbook が正常に実行された後に、**xxx** を NCP ビルドバージョンに置き換えて、以下のコマンドをすべてのノードで実行します。

```
$ docker load -i nsx-ncp-rhel-xxx.tar
```

以下に例を示します。

```
$ docker load -i nsx-ncp-rhel-2.4.0.12511604.tar
```

- イメージ名を取得して、これに再度タグを付けます。

```
$ docker images
$ docker image tag registry.local/xxxxx/nsx-ncp-rhel nsx-ncp 1
```

- 1** **xxx** を NCP ビルドバージョンに置き換えます。以下に例を示します。

```
docker image tag registry.local/2.4.0.12511604/nsx-ncp-rhel nsx-ncp
```

- OpenShift Container Platform Ansible ホストファイルで、以下のパラメーターを指定し、ネットワークプラグインとして NSX-T をセットアップします。

```
[OSEv3:children]
masters
nodes
etcd

[OSEv3:vars]
ansible_ssh_user=root
openshift_deployment_type=origin
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider'}]
openshift_master_htpasswd_users={"admin" : "$apr1$H0QeP6oX$HHdscz5gqMdtTcT5eoCJ20"}
openshift_master_default_subdomain=demo.example.com
openshift_use_nsx=true
os_sdn_network_plugin_name=cni
openshift_use_openshift_sdn=false
openshift_node_sdn_mtu=1500
openshift_master_cluster_method=native
openshift_master_cluster_hostname=master01.example.com
openshift_master_cluster_public_hostname=master01.example.com
openshift_hosted_manage_registry=true
openshift_hosted_manage_router=true
openshift_enable_service_catalog=true
openshift_cluster_monitoring_operator_install=true
```

```
openshift_web_console_install=true
openshift_console_install=true

# NSX-T specific configuration
#nsx_use_loadbalancer=false
nsx_openshift_cluster_name='cluster01'
nsx_api_managers='nsxmgr.example.com'
nsx_api_user='nsx_admin'
nsx_api_password='nsx_api_password_example'
nsx_tier0_router='LR-Tier-0'
nsx_overlay_transport_zone='TZ-Overlay'
nsx_container_ip_block='pod-networking'
nsx_no_snat_ip_block='pod-nonat'
nsx_external_ip_pool='pod-external'
nsx_top_fw_section='containers-top'
nsx_bottom_fw_section='containers-bottom'
nsx_ovs_uplink_port='ens224'
nsx_cni_url='http://websrv.example.com/nsx-cni-buildversion.x86_64.rpm'
nsx_ovs_url='http://websrv.example.com/openvswitch-buildversion.rhel75-1.x86_64.rpm'
nsx_kmod_ovs_url='http://websrv.example.com/kmod-openvswitch-buildversion.rhel75-1.el7.x86_64.rpm'
nsx_insecure_ssl=true
# vSphere Cloud Provider
#openshift_cloudprovider_kind=vsphere
#openshift_cloudprovider_vsphere_username='administrator@example.com'
#openshift_cloudprovider_vsphere_password='viadmin_password'
#openshift_cloudprovider_vsphere_host='vcsa.example.com'
#openshift_cloudprovider_vsphere_datacenter='Example-Datacenter'
#openshift_cloudprovider_vsphere_cluster='example-Cluster'
#openshift_cloudprovider_vsphere_resource_pool='ocp'
#openshift_cloudprovider_vsphere_datastore='example-Datastore-name'
#openshift_cloudprovider_vsphere_folder='ocp'

[masters]
master01.example.com
master02.example.com
master03.example.com

[etcd]
master01.example.com
master02.example.com
master03.example.com

[nodes]
master01.example.com ansible_ssh_host=192.168.220.2
openshift_node_group_name='node-config-master'
master02.example.com ansible_ssh_host=192.168.220.3
openshift_node_group_name='node-config-master'
master03.example.com ansible_ssh_host=192.168.220.4
openshift_node_group_name='node-config-master'
node01.example.com ansible_ssh_host=192.168.220.5
openshift_node_group_name='node-config-infra'
node02.example.com ansible_ssh_host=192.168.220.6
openshift_node_group_name='node-config-infra'
node03.example.com ansible_ssh_host=192.168.220.7
```

```
openshift_node_group_name='node-config-compute'
node04.example.com ansible_ssh_host=192.168.220.8
openshift_node_group_name='node-config-compute'
```

OpenShift Container Platform インストールパラメーターについての詳細は、[インベントリーファイルの設定](#) を参照してください。

## 手順

すべての要件を満たしている場合、NSX Data Center および OpenShift Container Platform をデプロイできます。

1. OpenShift Container Platform クラスターをデプロイします。

```
$ ansible-playbook -i hosts openshift-ansible/playbooks/deploy_cluster.yml
```

OpenShift Container Platform インストールについての詳細は、[OpenShift Container Platform のインストール](#) を参照してください。

2. インストールが完了したら、NCP および nsx-node-agent Pod が実行されていることを確認します。

```
$ oc get pods -o wide -n nsx-system
NAME          READY  STATUS   RESTARTS  AGE  IP             NODE
NOMINATED NODE
nsx-ncp-5sggt  1/1    Running  0          1h   192.168.220.8 node04.example.com
<none>
nsx-node-agent-b8nkm 2/2    Running  0          1h   192.168.220.5 node01.example.com
<none>
nsx-node-agent-cldks 2/2    Running  0          2h   192.168.220.8 node04.example.com
<none>
nsx-node-agent-m2p5l 2/2    Running  28         3h   192.168.220.4 master03.example.com
<none>
nsx-node-agent-pcfd5 2/2    Running  0          1h   192.168.220.7 node03.example.com
<none>
nsx-node-agent-ptwnq 2/2    Running  26         3h   192.168.220.2 master01.example.com
<none>
nsx-node-agent-xgh5q 2/2    Running  26         3h   192.168.220.3 master02.example.com
<none>
```

## 18.4. OPENSIFT CONTAINER PLATFORM デプロイ後の NSX-T の確認

OpenShift Container Platform のインストール、および NCP および **nsx-node-agent-\*** Pod の確認後に、以下を実行します。

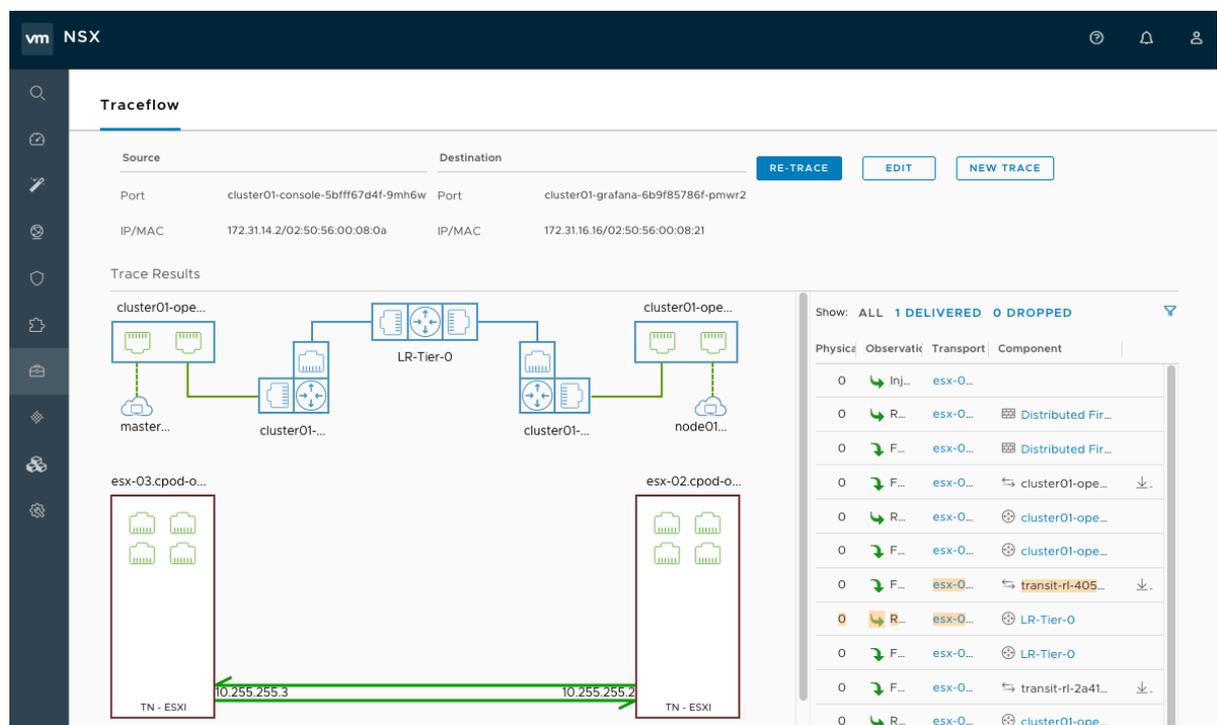
- ルーティングを確認します。Tier-1 ルーターがインストール時に作成されており、Tier-0 ルーターにリンクされていることを確認します。

図18.3 T1 ルーターを示す NSX UI の表示

Logical Router	ID	Type	Connected Tier-O Router	High Availability Mode	Transport Zone	Edge Cluster
cluster01-kube-proxy-and-dns	0c46...ae7e	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-kube-public	044e...29b0	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-kube-service-catalog	4d27...b9af	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-kube-system	88a6...2cf7	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-management-infra	4d50...8443	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-nsx-system	bed7...9a51	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-openshift	8818...ae95	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-openshift-ansible-service-broker	7923...f399	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-openshift-console	4058...4704	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-openshift-infra	d6ee...36d0	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-openshift-logging	963b...9ea4	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-openshift-monitoring	2a41...cdb2	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-openshift-node	95c8...2894	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-openshift-template-service-broker	b777...af45	Tier-1	LR-Tier-0		TZ-Overlay	
cluster01-openshift-web-console	edb1...b374	Tier-1	LR-Tier-0		TZ-Overlay	
lb-cluster01-iftpc	d9e3...c9f4	Tier-1	LR-Tier-0	Active-Standby	TZ-Overlay	Edge-Cluster
LR-Tier-0	d9e3...a860	Tier-0		Active-Standby	TZ-VLAN	Edge-Cluster
LR-Tier-1	d984...b34a	Tier-1	LR-Tier-0	Active-Standby	TZ-Overlay	Edge-Cluster

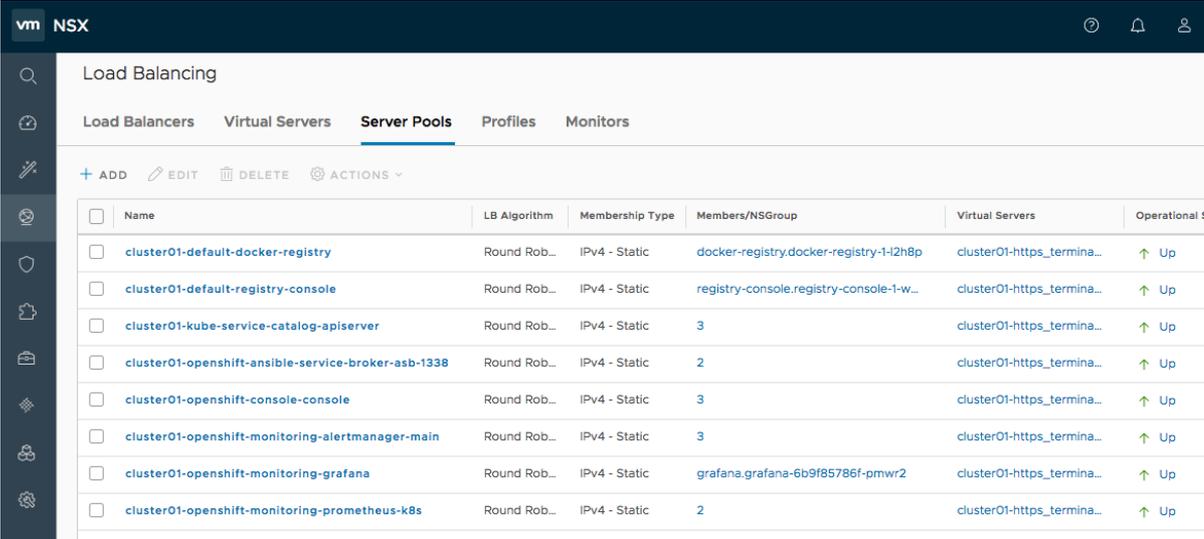
- ネットワークのトレースフローおよび可視性を確認します。たとえば、'console' と 'grafana' 間の接続を確認します。  
Pod、プロジェクト、仮想マシン、および外部サービス間の通信のセキュリティを保護し、最適化する方法についての詳細は、以下の例を参照してください。

図18.4 ネットワークのトレースフローを示す NSX UI の表示



- 負荷分散を確認します。以下の例が示すように NSX-T Data Center はロードバランサーおよび Ingress コントローラー機能を提供します。

図18.5 ロードバランサーを示す NSX UI の表示



<input type="checkbox"/>	Name	LB Algorithm	Membership Type	Members/NSGroup	Virtual Servers	Operational S
<input type="checkbox"/>	cluster01-default-docker-registry	Round Rob...	IPv4 - Static	docker-registry.docker-registry-1h2h8p	cluster01-https_termina...	↑ Up
<input type="checkbox"/>	cluster01-default-registry-console	Round Rob...	IPv4 - Static	registry-console.registry-console-1-w...	cluster01-https_termina...	↑ Up
<input type="checkbox"/>	cluster01-kube-service-catalog-apiserver	Round Rob...	IPv4 - Static	3	cluster01-https_termina...	↑ Up
<input type="checkbox"/>	cluster01-openshift-ansible-service-broker-asb-1338	Round Rob...	IPv4 - Static	2	cluster01-https_termina...	↑ Up
<input type="checkbox"/>	cluster01-openshift-console-console	Round Rob...	IPv4 - Static	3	cluster01-https_termina...	↑ Up
<input type="checkbox"/>	cluster01-openshift-monitoring-alertmanager-main	Round Rob...	IPv4 - Static	3	cluster01-https_termina...	↑ Up
<input type="checkbox"/>	cluster01-openshift-monitoring-grafana	Round Rob...	IPv4 - Static	grafana.grafana-6b9f85786f-pmwr2	cluster01-https_termina...	↑ Up
<input type="checkbox"/>	cluster01-openshift-monitoring-prometheus-k8s	Round Rob...	IPv4 - Static	2	cluster01-https_termina...	↑ Up

追加の設定およびオプションについては、[VMware NSX-T v2.4 OpenShift Plug-In ドキュメント](#)を参照してください。

## 第19章 KURYR SDN の設定

### 19.1. KURYR SDN および OPENSIFT CONTAINER PLATFORM

**Kuryr** (具体的には Kuryr-Kubernetes) は、[CNI](#) と [OpenStack Neutron](#) を使用して構築した SDN ソリューションです。Kuryr の利点として、幅広い Neutron SDN バックエンドを使用でき、Kubernetes Pod と OpenStack 仮想マシン (VM) の相互接続性が確保できる点が挙げられます。

Kuryr-Kubernetes と OpenShift Container Platform の統合は主に、OpenStack の仮想マシンで実行する OpenShift Container Platform クラスター用に設計されました。Kuryr-Kubernetes コンポーネントは OpenShift Container Platform の **kuryr** namespace で Pod としてインストールされます。

- **kuryr-controller: infra** ノードにインストールされる単一のサービスインスタンスです。**Deployment** として OpenShift Container Platform でモデリングされます。
- **kuryr-cni**: 各 OpenShift Container Platform ノードで Kuryr を CNI ドライバーとしてインストールし、設定するコンテナです。**DaemonSet** として OpenShift Container Platform でモデリングされます。

Kuryr コントローラーは OpenShift API サーバーで Pod、サービスおよび namespace の作成、更新、および削除イベントについて監視します。これは、OpenShift Container Platform API 呼び出しを Neutron および Octavia の対応するオブジェクトにマップします。そのため、Neutron トランクポート機能を実装するすべてのネットワークソリューションを使用して、Kuryr 経由で OpenShift Container Platform をサポートすることができます。これには、OVS および OVN などのオープンソースソリューションや Neutron と互換性のある市販の SDN が含まれます。

### 19.2. KURYR SDN のインストール

OpenStack クラウドへの Kuryr SDN インストールについては、[OpenStack 設定ドキュメント](#) で説明されている手順を実行する必要があります。

### 19.3. 検証

OpenShift Container Platform のインストールが完了したら、Kuryr Pod が正常にデプロイされているかどうかを確認できます。

```
$ oc -n kuryr get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP             NODE
kuryr-cni-ds-66kt2                  2/2    Running  0          3d    192.168.99.14  infra-node-
0.openshift.example.com
kuryr-cni-ds-ggcpz                  2/2    Running  0          3d    192.168.99.16  master-
0.openshift.example.com
kuryr-cni-ds-mhzjt                  2/2    Running  0          3d    192.168.99.6   app-node-
1.openshift.example.com
kuryr-cni-ds-njctb                  2/2    Running  0          3d    192.168.99.12  app-node-
0.openshift.example.com
kuryr-cni-ds-v8hp8                  2/2    Running  0          3d    192.168.99.5   infra-node-
1.openshift.example.com
kuryr-controller-59fc7f478b-qwk4k  1/1    Running  0          3d    192.168.99.5   infra-node-
1.openshift.example.com
```

kuryr-cni Pod は、全 OpenShift Container Platform ノードで実行します。単一の kuryr-controller インスタンスは **infra** ノードのいずれかで実行する必要があります。



## 注記

Kuryr SDN が有効にされている場合に、ネットワークポリシーおよびノードポートサービスはサポートされません。

## 第20章 AMAZON WEB サービス (AWS) の設定

### 20.1. 概要

OpenShift Container Platform は、[AWS ボリュームをアプリケーションデータの永続ストレージとして使用する](#) など、[AWS EC2 インフラストラクチャー](#) にアクセスできるように設定できます。これを実行するには、AWS を設定した後に OpenShift Container Platform ホストで追加の設定を行う必要があります。

#### 20.1.1. Amazon Web サービス (AWS) の認証の設定

**パーミッション:** AWS インスタンスでは、OpenShift Container Platform でロードバランサーおよびストレージを要求し、管理できるようにするために、アクセスおよびシークレットキーを使用する Programmatic Access (プログラムによるアクセス) を持つ IAM アカウントか、または作成時にインスタンスに割り当てられる IAM ロールのいずれかが必要になります。

IAM アカウントまたは IAM ロールには、完全なクラウドプロバイダー機能を利用できるようにするためのポリシーパーミッションが必要です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ec2:DescribeVolume*",
        "ec2:CreateVolume",
        "ec2:CreateTags",
        "ec2:DescribeInstances",
        "ec2:AttachVolume",
        "ec2:DetachVolume",
        "ec2>DeleteVolume",
        "ec2:DescribeSubnets",
        "ec2:CreateSecurityGroup",
        "ec2:DescribeSecurityGroups",
        "ec2>DeleteSecurityGroup",
        "ec2:DescribeRouteTables",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:RevokeSecurityGroupIngress",
        "elasticloadbalancing:DescribeTags",
        "elasticloadbalancing:CreateLoadBalancerListeners",
        "elasticloadbalancing:ConfigureHealthCheck",
        "elasticloadbalancing>DeleteLoadBalancerListeners",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:CreateLoadBalancer",
        "elasticloadbalancing>DeleteLoadBalancer",
        "elasticloadbalancing:ModifyLoadBalancerAttributes",
        "elasticloadbalancing:DescribeLoadBalancerAttributes"
      ],
      "Resource": "*",
      "Effect": "Allow",
      "Sid": "1"
    }
  ]
}
```

```
}
]
}
```

```
aws iam put-role-policy \
  --role-name openshift-role \
  --policy-name openshift-admin \
  --policy-document file://openshift_iam_policy
```

```
aws iam put-user-policy \
  --user-name openshift-admin \
  --policy-name openshift-admin \
  --policy-document file://openshift_iam_policy
```



### 注記

OpenShift ノードインスタンスは **ec2:DescribeInstance** パーミッションのみが必要ですが、インストーラーは単一の AWS アクセスキーおよびシークレットのみの定義を許可します。これは、IAM ロールを使用し、上記のパーミッションをマスターインスタンスに割り当てた後に **ec2:DescribeInstance** をノードに割り当てることでバイパスできます。

## 20.1.1.1. インストール時の OpenShift Container Platform クラウドプロバイダーの設定

### 手順

アクセスおよびシークレットキーのある IAM アカウントを使用して Amazon Web Services クラウドプロバイダーを設定するには、以下の値をインベントリーに追加します。

```
[OSEv3:vars]
openshift_cloudprovider_kind=aws
openshift_clusterid=openshift ①
openshift_cloudprovider_aws_access_key=AKIAJ6VLBLISADPBUA ②
openshift_cloudprovider_aws_secret_key=g/8PmDNYHVSQn0BQE+xtsHzbaZaGYjGNzhbdgwjH ③
```

- ① OpenShift に使用されるすべてのリソース (インスタンス、ロードバランサー、vpc など) に割り当てられるタグ。
- ② IAM アカウントによって使用される AWS アクセスキー。
- ③ IAM アカウントによって使用される AWS シークレットキー。

IAM ロールを使用して Amazon Web Services クラウドプロバイダーを設定するには、以下の値をインベントリーに追加します。

```
[source,yaml]
----
[OSEv3:vars]
openshift_cloudprovider_kind=aws
openshift_clusterid=openshift ①
----
<1> A tag assigned to all resources (instances, load balancers, vpc, etc) used for OpenShift.
```

NOTE: The IAM role takes the place of needing an access and secret key.

### 20.1.1.2. インストール後の OpenShift Container Platform クラウドプロバイダーの設定

Amazon Web Services クラウドプロバイダーの値がインストール時に提供されない場合でも、インストール後に設定を定義し、作成することができます。設定ファイルの設定手順に従い、マスターおよびノードを手動で設定します ([OpenShift Container Platform マスターでの AWS の手動設定](#))。

#### 重要

- マスターホスト、ノードホスト、サブネットにはすべて、`kubernetes.io/cluster/<clusterid>,Value=(owned|shared)` のタグが必要です。
- 1つのセキュリティグループ (ノードにリンクされていることが望ましい) に `kubernetes.io/cluster/<clusterid>,Value=(owned|shared)` タグが必要です。
  - すべてのセキュリティグループに `kubernetes.io/cluster/<clusterid>,Value=(owned|shared)` のタグを付けないでください。その場合、Elastic Load Balancing (ELB) がロードバランサーを作成できなくなります。

## 20.2. セキュリティーグループの設定

AWS に OpenShift Container Platform をインストールする際は、適切なセキュリティグループがセットアップされていることを確認してください。

セキュリティグループにはいくつかのポートを設定しておく必要があります、それがないとインストールは失敗します。また、インストールしようとしているクラスターの設定によっては、追加のポートが必要になる場合があります。セキュリティグループの詳細、およびその適切な調整方法については、[必要なポート](#) を参照してください。

すべての OpenShift Container Platform ホスト	<ul style="list-style-type: none"> <li>● インストーラー/Ansible を実行しているホストの tcp/22</li> </ul>
etcd セキュリティーグループ	<ul style="list-style-type: none"> <li>● マスターの tcp/2379</li> <li>● etcd ホストの tcp/2380</li> </ul>

マスターのセキュリティーグループ	<ul style="list-style-type: none"> <li>● tcp/8443 (0.0.0.0/0)</li> <li>● 3.2 よりも前にインストールされた環境、または 3.2 にアップグレードされた環境向けのすべての OpenShift Container Platform ホストの tcp/53</li> <li>● 3.2 よりも前にインストールされた環境、または 3.2 にアップグレードされた環境向けのすべての OpenShift Container Platform ホストの udp/53</li> <li>● 3.2 を使ってインストールされた新しい環境向けのすべての OpenShift Container Platform ホストの tcp/8053</li> <li>● 3.2 を使ってインストールされた新しい環境向けのすべての OpenShift Container Platform ホストの udp/8053</li> </ul>
ノードのセキュリティーグループ	<ul style="list-style-type: none"> <li>● マスターの tcp/10250</li> <li>● ノードの udp/4789</li> </ul>
インフラストラクチャーノード (OpenShift Container Platform ルーターをホストできるノード)	<ul style="list-style-type: none"> <li>● tcp/443 (0.0.0.0/0)</li> <li>● tcp/80 (0.0.0.0/0)</li> </ul>
CRI-O	CRI-O を使用している場合は、tcp/10010 を開き、 <b>oc exec</b> および <b>oc rsh</b> 操作を実行できるようにします。

マスターまたはルートの負荷分散のために外部のロードバランサー (ELB) を設定する場合は、ELB の Ingress および Egress のセキュリティーグループを設定することも必要になります。

### 20.2.1. 検出された IP アドレスとホスト名の上書き

AWS では、以下のような場合に変数の上書きが必要になります。

変数	使用法
hostname	ユーザーが <b>DNS hostnames</b> と <b>DNS resolution</b> の両方に対して設定されていない VPC にインストールしている。
ip	複数のネットワークインターフェイスを設定しており、デフォルト以外のインターフェイスを使用することを検討している。

変数	使用法
<b>public_hostname</b>	<ul style="list-style-type: none"> <li>● VPC サブネットが <b>Auto-assign Public IP</b> について設定されていないマスターインスタンス。このマスターへの外部アクセスについては、ユーザーは ELB か、または必要な外部アクセスを提供する他のロードバランサーを使用する必要があります。または、VPN 接続を介してホストの内部名に接続する必要があります。</li> <li>● メタデータが無効にされているマスターインスタンス。</li> <li>● この値は、実際にはノードによって使用されません。</li> </ul>
<b>public_ip</b>	<ul style="list-style-type: none"> <li>● VPC サブネットが <b>Auto-assign Public IP</b> について設定されていないマスターインスタンス。</li> <li>● メタデータが無効にされているマスターインスタンス。</li> <li>● この値は、実際にはノードによって使用されません。</li> </ul>

EC2 ホストの場合はとくに、**DNS ホスト名** と **DNS 解決** の両方が有効にされている VPC にデプロイされる必要があります。

### 20.2.1.1. Amazon Web Services (AWS) の OpenShift Container Platform レジストリーの設定

Amazon Web Services (AWS) は、OpenShift Container Platform が OpenShift Container Platform コンテナイメージレジストリーを使用してコンテナイメージを保存するために使用可能なオブジェクトクラウドストレージを提供します。

詳細は、[Amazon S3](#) を参照してください。

#### 前提条件

OpenShift Container Platform はイメージストレージに S3 を使用します。S3 バケット、IAM ポリシー、および **Programmatic Access** を持つ IAM ユーザーを作成し、インストーラーによるレジストリー設定を許可する必要があります。

以下の例では、awscli を使用して **us-east-1** のリージョンに **openshift-registry-storage** という名前のバケットを作成します。

```
# aws s3api create-bucket \
  --bucket openshift-registry-storage \
  --region us-east-1
```

デフォルトポリシー

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```

    "s3:ListBucket",
    "s3:GetBucketLocation",
    "s3:ListBucketMultipartUploads"
  ],
  "Resource": "arn:aws:s3:::S3_BUCKET_NAME"
},
{
  "Effect": "Allow",
  "Action": [
    "s3:PutObject",
    "s3:GetObject",
    "s3:DeleteObject",
    "s3:ListMultipartUploadParts",
    "s3:AbortMultipartUpload"
  ],
  "Resource": "arn:aws:s3:::S3_BUCKET_NAME/*"
}
]
}

```

### 20.2.1.1.1. OpenShift Container Platform インベントリを S3 を使用するように設定する

#### 手順

S3 バケットおよび IAM ユーザーを使用できるようにレジストリーの Ansible インベントリを設定するには、以下を実行します。

```

[OSEv3:vars]
# AWS Registry Configuration
openshift_hosted_manage_registry=true
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_provider=s3
openshift_hosted_registry_storage_s3_accesskey=AKIAJ6VLREDHATSPBUA ①
openshift_hosted_registry_storage_s3_secretkey=g/8PmTYDQVGssFWWFvfawHpDbZyGkjGNZhbw
QpjH ②
openshift_hosted_registry_storage_s3_bucket=openshift-registry-storage ③
openshift_hosted_registry_storage_s3_region=us-east-1 ④
openshift_hosted_registry_storage_s3_chunksize=26214400
openshift_hosted_registry_storage_s3_rootdirectory=/registry
openshift_hosted_registry_storage_s3_encrypt=false
openshift_hosted_registry_storage_s3_kmskeyid=aws_kms_key_id ⑤
openshift_hosted_registry_pullthrough=true
openshift_hosted_registry_acceptschema2=true
openshift_hosted_registry_enforcequota=true
openshift_hosted_registry_replicas=3

```

- ① IAM ユーザーのアクセスキー。(IAM ロールの場合は不要)
- ② IAM ユーザーのシークレットキー。(IAM ロールの場合は不要)
- ③ S3 ストレージバケット名。
- ④ バケットがあるリージョン。
- ⑤ クラスターのデータの暗号化に使用される暗号化キーの AWS Key Management Service (AWS KMS) キー ID。

KIMISJ ユー ID。

### 20.2.1.1.2. S3 を使用するための OpenShift Container Platform レジストリーの手動設定

Amazon Web Services (AWS) S3 オブジェクトストレージを使用するには、レジストリーの設定ファイルを編集し、レジストリー Pod にマウントします。

#### 手順

1. 現在の `config.yml` をエクスポートします。

```
$ oc get secret registry-config \
  -o jsonpath='{.data.config\.yml}' -n default | base64 -d \
  >> config.yml.old
```

2. 古い `config.yml` から新規の設定ファイルを作成します。

```
$ cp config.yml.old config.yml
```

3. ファイルを編集して S3 パラメーターを追加します。レジストリーの設定ファイルの `storage` セクションにアカウント名、アカウントキー、コンテナ、およびレルムを指定します。

```
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  s3:
    accesskey: AKIAJ6VLREDHATSPBUA ❶
    secretkey: g/8PmTYDQVGssFWWFvfawHpDbZyGkjGNZhbwQpjH ❷
    region: us-east-1 ❸
    bucket: openshift-registry-storage ❹
    encrypt: False
    secure: true
    v4auth: true
    rootdirectory: /registry ❺
    chunksize: "26214400"
```

- ❶ S3 バケットにアクセスすることが承認されている AWS アクセスキーに置き換えます。
- ❷ 定義済みの AWS アクセスキーに対応するシークレットキー。
- ❸ レジストリーとして使用される S3 バケットの名前。
- ❹ レジストリーがイメージおよびメタデータを保管できる場所。(デフォルトは `/registry` です)

4. `registry-config` シークレットを削除します。

```
$ oc delete secret registry-config -n default
```

5. シークレットを再作成して、更新された設定ファイルを参照します。

-

```
$ oc create secret generic registry-config \
  --from-file=config.yml -n default
```

6. 更新された設定を読み取るためにレジストリーを再デプロイします。

```
$ oc rollout latest docker-registry -n default
```

### 20.2.1.1.3. レジストリーが S3 ストレージを使用することを確認します。

レジストリーが Amazon S3 ストレージを使用していることを確認するには、以下を実行します。

#### 手順

1. レジストリーの正常なデプロイ後に、レジストリー **deploymentconfig** は registry-storage を AWS S3 ではなく **emptydir** として記述しますが、AWS S3 バケットの設定はシークレット **docker-config** に置かれます。**docker-config** シークレットは **REGISTRY\_CONFIGURATION\_PATH** にマウントされ、これにより、レジストリーオブジェクトストレージに AWS S3 を使用する場合にすべてのパラメーターが指定されます。

```
$ oc describe dc docker-registry -n default
...
Environment:
  REGISTRY_HTTP_ADDR:      :5000
  REGISTRY_HTTP_NET:      tcp
  REGISTRY_HTTP_SECRET:
  SPLR83SDsPaGbGuwSMDfnDwrDRvGf6YXI4h9JQrToQU=
  REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA: false
  REGISTRY_HTTP_TLS_KEY:   /etc/secrets/registry.key
  OPENSHIFT_DEFAULT_REGISTRY: docker-registry.default.svc:5000
  REGISTRY_CONFIGURATION_PATH: /etc/registry/config.yml
  REGISTRY_OPENSHIFT_SERVER_ADDR: docker-registry.default.svc:5000
  REGISTRY_HTTP_TLS_CERTIFICATE: /etc/secrets/registry.crt
Mounts:
  /etc/registry from docker-config (rw)
  /etc/secrets from registry-certificates (rw)
  /registry from registry-storage (rw)
Volumes:
  registry-storage:
    Type: EmptyDir (a temporary directory that shares a pod's lifetime) 1
    Medium:
  registry-certificates:
    Type: Secret (a volume populated by a Secret)
    SecretName: registry-certificates
    Optional: false
  docker-config:
    Type: Secret (a volume populated by a Secret)
    SecretName: registry-config
    Optional: false
....
```

1 Pod の寿命を共有する一時ディレクトリー

2. **/registry** マウントポイントが空であることを確認します。

```
$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') -i -t -- ls -l /registry
total 0
```

空になるのは、S3 設定が **registry-config** シークレットに定義されているためです。

```
$ oc describe secret registry-config
Name:      registry-config
Namespace: default
Labels:    <none>
Annotations: <none>

Type: Opaque

Data
====
config.yml: 398 bytes
```

3. インストーラーは、[インストールドキュメントのストレージセクション](#) で記載されているように、拡張されたレジストリー機能を使用して、希望の設定で **config.yml** ファイルを作成します。以下のコマンドで、ストレージバケット設定が保存されている **storage** セクションを含めて設定ファイルを表示します。

```
$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') \
  cat /etc/registry/config.yml

version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  s3:
    accesskey: AKIAJ6VLREDHATSPBUA
    secretkey: g/8PmTYDQVGssFWWFvfawHpDbZyGkjGNZhbWQpjH
    region: us-east-1
    bucket: openshift-registry-storage
    encrypt: False
    secure: true
    v4auth: true
    rootdirectory: /registry
    chunksize: "26214400"
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
```

```

repository:
- name: openshift
  options:
    pullthrough: true
    acceptschema2: true
    enforcequota: true
storage:
- name: openshift

```

または、シークレットを表示することができます。

```

$ oc get secret registry-config -o jsonpath='{.data.config\.yaml}' | base64 -d
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
s3:
  accesskey: AKIAJ6VLREDHATSPBUA
  secretkey: g/8PmTYDQVGssFWWFvfawHpDbZyGkjGNZhbWQpjH
  region: us-east-1
  bucket: openshift-registry-storage
  encrypt: False
  secure: true
  v4auth: true
  rootdirectory: /registry
  chunksize: "26214400"
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
  options:
    pullthrough: true
    acceptschema2: true
    enforcequota: true
storage:
- name: openshift

```

**emptyDir** ボリュームを使用する場合には、**/registry** マウントポイントは以下ようになります。

```

$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') -i -t -- df -h /registry
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc        100G  226M   30G   1% /registry

```

```
$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
  -o=jsonpath='{.items[0].metadata.name}') -i -t -- ls -l /registry
total 0
drwxr-sr-x. 3 1000000000 1000000000 22 Jun 19 12:24 docker
```

## 20.3. AWS 変数の設定

必要な AWS 変数を設定するには、OpenShift Container Platform のマスターとノード両方のすべてのホストに、`/etc/origin/cloudprovider/aws.conf` ファイルを以下の内容で作成します。

```
[Global]
Zone = us-east-1c 1
```

- 1** これは AWS インスタンスのアベイラビリティゾーンであり、EBS ボリュームがある場所です。この情報は AWS マネジメントコンソールから取得されます。

## 20.4. OPENSIFT CONTAINER PLATFORM での AWS の設定

AWS は OpenShift Container Platform に 2 通りの方法で設定できます。

- [Ansible の使用](#) または
- [手動](#)

### 20.4.1. Ansible を使用した OpenShift Container Platform での AWS の設定

クラスターのインストール時

に、`openshift_cloudprovider_aws_access_key`、`openshift_cloudprovider_aws_secret_key`、`openshift_cloudprovider_kind`、`openshift_clusterid` のパラメーターを使用して、AWS を設定できます。これは、[インベントリーファイル](#) で設定可能です。

#### Ansible を使用した AWS の設定例

```
# Cloud Provider Configuration
#
# Note: You may make use of environment variables rather than store
# sensitive configuration within the ansible inventory.
# For example:
#openshift_cloudprovider_aws_access_key="{{ lookup('env','AWS_ACCESS_KEY_ID') }}"
#openshift_cloudprovider_aws_secret_key="{{ lookup('env','AWS_SECRET_ACCESS_KEY') }}"
#
#openshift_clusterid=unique_identifier_per_availability_zone
#
# AWS (Using API Credentials)
#openshift_cloudprovider_kind=aws
#openshift_cloudprovider_aws_access_key=aws_access_key_id
#openshift_cloudprovider_aws_secret_key=aws_secret_access_key
#
# AWS (Using IAM Profiles)
#openshift_cloudprovider_kind=aws
# Note: IAM roles must exist before launching the instances.
```



### 注記

Ansible が AWS を設定する際に、必要な変更が以下のファイルに自動的に実行されます。

- `/etc/origin/cloudprovider/aws.conf`
- `/etc/origin/master/master-config.yaml`
- `/etc/origin/node/node-config.yaml`

## 20.4.2. OpenShift Container Platform マスターでの AWS の手動設定

すべてのマスターでマスター設定ファイル (デフォルトは `/etc/origin/master/master-config.yaml`) を編集するか、または [作成](#) し、`apiServerArguments` と `controllerArguments` の各セクションの内容を更新します。

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "aws"
    cloud-config:
      - "/etc/origin/cloudprovider/aws.conf"
  controllerArguments:
    cloud-provider:
      - "aws"
    cloud-config:
      - "/etc/origin/cloudprovider/aws.conf"
```

現時点では、クラウドプロバイダーの統合を正常に機能させるため、`nodeName` は AWS のインスタンス名と一致している必要があります。また、この名前は RFC1123 に準拠している必要もあります。



### 重要

コンテナ化インストールをトリガーする場合、`/etc/origin` と `/var/lib/origin` のディレクトリのみがマスターとノードのコンテナにマウントされます。したがって、`aws.conf` は `/etc/` ではなく `/etc/origin/` になければなりません。

## 20.4.3. OpenShift Container Platform ノードでの AWS の手動設定

適切な [ノード設定マップ](#) を編集して、`kubeletArguments` セクションの内容を更新します。

```
kubeletArguments:
  cloud-provider:
    - "aws"
  cloud-config:
    - "/etc/origin/cloudprovider/aws.conf"
```



## 重要

コンテナ化インストールをトリガーする場合、`/etc/origin` と `/var/lib/origin` のディレクトリーのみがマスターとノードのコンテナにマウントされます。したがって、`aws.conf` は `/etc/` ではなく `/etc/origin/` になければなりません。

### 20.4.4. キーと値のアクセスペアの手動設定

以下の環境変数が、マスターの `/etc/origin/master/master.env` ファイルおよびノードの `/etc/sysconfig/atomic-openshift-node` ファイルに設定されていることを確認します。

```
AWS_ACCESS_KEY_ID=<key_ID>
AWS_SECRET_ACCESS_KEY=<secret_key>
```



## 注記

アクセスキーは、AWS IAM ユーザーを設定する際に取得されます。

### 20.5. 設定変更の適用

マスターおよびノードのすべてのホストで OpenShift Container Platform サービスを起動または再起動し、設定の変更を適用します。 [OpenShift Container Platform サービスの再起動](#) を参照してください。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```



## 注記

Kubernetes アーキテクチャーでは、クラウドプロバイダーからの信頼性のあるエンドポイントが必要です。クラウドプロバイダーが停止している場合、`kubelet` は OpenShift Container Platform が再起動されないようにします。基礎となるクラウドプロバイダーのエンドポイントに信頼性がない場合は、クラウドプロバイダー統合を使用してクラスターをインストールしないでください。クラスターをベアメタル環境の場合のようにインストールします。インストール済みのクラスターで、クラウドプロバイダー統合をオンまたはオフに切り替えることは推奨されていません。ただし、そのシナリオが避けられない場合は、以下のプロセスを実行してください。

クラウドプロバイダーを不使用から使用に切り替えるとエラーメッセージが表示されます。クラウドプロバイダーを追加すると、ノードが `hostname` を `externalID` として使用する (クラウドプロバイダーが使用されていなかった場合のケース) 代わりに、クラウドプロバイダーの `instance-id` (クラウドプロバイダーによって指定される) の使用に切り替えるため、ノードの削除が試みられます。この問題を解決するには、以下を実行します。

1. CLI にクラスター管理者としてログインします。
2. 既存のノードラベルをチェックし、これらをバックアップします。

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?=Taints)'
```

3. ノードを削除します。

```
$ oc delete node <node_name>
```

4. 各ノードホストで OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

5. 以前に使用していた [各ノードのラベル](#) を再度追加します。

## 20.6. クラスターに対する AWS のラベリング

AWS プロバイダー認証情報を設定する場合、すべてのホストにラベルが付けられていることを確認する必要があります。

クラスターに関連付けられているリソースを正確に特定するには、キー `kubernetes.io/cluster/<clusterid>` でリソースにタグを付けます。

- `<clusterid>` はクラスターに固有の名前です。

該当の値を、ノードがこのクラスターだけに所属する場合には `owned` に、また、他のシステムとリソースが共有されている場合には `shared` に設定します。

すべてのリソースに `kubernetes.io/cluster/<clusterid>,Value=(owned|shared)` タグを付けることで、複数ゾーンまたは複数クラスターに関連する潜在的な問題を回避できます。

OpenShift Container Platform へのラベル付けとタグ付けに関する詳細は、[Pods and Services](#) を参照してください。

### 20.6.1. タグを必要とするリソース

タグ付けが必要なリソースは以下の 4 種類です。

- インスタンス
- セキュリティーグループ
- ロードバランサー
- EBS ボリューム

### 20.6.2. 既存クラスターへのタグ付け

クラスターは `kubernetes.io/cluster/<clusterid>,Value=(owned|shared)` タグの値を使用して、AWS クラスターに所属するリソースを判断します。したがって、関連のリソースはすべて、そのキーの同じ値を使用して `kubernetes.io/cluster/<clusterid>,Value=(owned|shared)` タグでラベルを付ける必要があります。これらのリソースには以下が含まれます。

- 全ホスト
- AWS インスタンスで使用する関連のロードバランサーすべて
- EBS ボリュームすべて。タグ付けが必要な EBS ボリュームは以下を使用して見つけることができます。

```
$ oc get pv -o json|jq '.items[].spec.awsElasticBlockStore.volumeID'
```

- AWS インスタンスで使用する関連のセキュリティーグループすべて



### 注記

すべての既存セキュリティーグループに `kubernetes.io/cluster/<name>,Value=<clusterid>` のタグを付けないでください。その場合、Elastic Load Balancing (ELB) がロードバランサーを作成できなくなります。

リソースにタグを付けた後に、マスターサービスをマスター上で、ノードサービスをすべてのノード上で再起動します。[設定の適用](#) セクションを参照してください。

### 20.6.3. Red Hat OpenShift Container Storage について

Red Hat OpenShift Container Storage (RHOCS) は、インハウスまたはハイブリッドクラウドのいずれの場合でも、OpenShift Container Platform のすべてに対応する永続ストレージのプロバイダーです。Red Hat ストレージソリューションとして、RHOCS は OpenShift Container Platform にインストール (コンバージド) されているか、または OpenShift Container Platform と共にインストール (インデペンデント) されているかを問わず、デプロイメント、管理およびモニタリングを目的として OpenShift Container Platform に完全に統合されます。OpenShift Container Storage は単一のアベイラビリティゾーンまたはノードに制限されないため、停止した場合にも存続できる可能性があります。RHOCS の詳細の使用方法については、[RHOCS3.11 Deployment Guide](#) を参照してください。

## 第21章 RED HAT VIRTUALIZATION の設定

OpenShift Container Platform を Red Hat Virtualization に設定するには、bastion 仮想マシンを作成して、その仮想マシンを使用して OpenShift Container Platform をインストールできます。

### 21.1. BASTION 仮想マシンの作成

Red Hat Virtualization で bastion 仮想マシンを作成し、OpenShift Container Platform をインストールします。

#### 手順

1. SSH を使用して Manager マシンにログインします。
2. インストールファイル用に、`/bastion_installation` などの一時的な bastion インストールディレクトリーを作成します。
3. **ansible-vault** で暗号化された `/bastion_installation/secure_vars.yaml` ファイルを作成し、パスワードを記録します。

```
# ansible-vault create secure_vars.yaml
```

4. 以下のパラメーター値を `secure_vars.yaml` ファイルに追加します。

```
engine_password: <Manager_password> ①
bastion_root_password: <bastion_root_password> ②
rbsub_user: <Red_Hat_Subscription_Manager_username> ③
rbsub_pass: <Red_Hat_Subscription_Manager_password>
rbsub_pool: <Red_Hat_Subscription_Manager_pool_id> ④
root_password: <OpenShift_node_root_password> ⑤
engine_cafile: <RHVM_CA_certificate> ⑥
oreg_auth_user: <image_registry_authentication_username> ⑦
oreg_auth_password: <image_registry_authentication_password>
```

- ① 管理ポータルにログインするためのパスワード。
- ② bastion 仮想マシンの root パスワード
- ③ Red Hat Subscription Manager の認証情報。
- ④ Red Hat Virtualization Manager サブスクリプションプールのプール ID。
- ⑤ OpenShift Container Platform の root パスワード
- ⑥ Red Hat Virtualization Manager CA 証明書 Manager マシンから Playbook を実行していない場合は、**engine\_cafile** 値が必要です。Manager CA 証明書のデフォルト場所は `/etc/pki/ovirt-engine/ca.pem` です。
- ⑦ 認証が必要なイメージレジストリーを使用している場合は、認証情報を追加します。

5. ファイルを保存します。
6. Red Hat Enterprise Linux KVM Guest Image のダウンロードリンクを取得します。

- a. [Red Hat カスタマーポータル \(Red Hat Enterprise Linux のダウンロード\)](#) に移動します。
  - b. **製品ソフトウェア** タブで、**Red Hat Enterprise Linux KVM Guest Image**を見つけます。
  - c. **Download Now** を右クリックし、リンクをコピーして保存します。  
リンクは時間的に制約があるので、bastion 仮想マシンを作成する前にコピーする必要があります。
7. 以下の内容で `/bastion_installation/create-bastion-machine-playbook.yaml` ファイルを作成し、パラメーターの値を更新します。

```

---
- name: Create a bastion machine
  hosts: localhost
  connection: local
  gather_facts: false
  no_log: true

  roles:
    - oVirt.image-template
    - oVirt.vm-infra
  no_log: true

  vars:
    engine_url: https://_Manager_FQDN_/ovirt-engine/api ❶
    engine_user: <admin@internal>
    engine_password: "{{ engine_password }}"
    engine_cafile: /etc/pki/ovirt-engine/ca.pem

    qcow_url: <RHEL_KVM_guest_image_download_link> ❷
    template_cluster: Default
    template_name: rhelguest7
    template_memory: 4GiB
    template_cpu: 2
    wait_for_ip: true
    debug_vm_create: false

  vms:
    - name: rhel-bastion
      cluster: "{{ template_cluster }}"
      profile:
        cores: 2
        template: "{{ template_name }}"
        root_password: "{{ root_password }}"
        ssh_key: "{{ lookup('file', '/root/.ssh/id_rsa_ssh_ocp_admin.pub') }}"
        state: running
      cloud_init:
        custom_script: |
          rh_subscription:
            username: "{{ rsub_user }}"
            password: "{{ rsub_pass }}"
            auto-attach: true
            disable-repo: ["*"]
            # 'rhel-7-server-rhv-4.2-manager-rpms' supports RHV 4.2 and 4.3
            enable-repo: ['rhel-7-server-rpms', 'rhel-7-server-extras-rpms', 'rhel-7-server-ansible-
2.7-rpms', 'rhel-7-server-ose-3.11-rpms', 'rhel-7-server-supplementary-rpms', 'rhel-7-server-

```

```

rhv-4.2-manager-rpms]
  packages:
    - ansible
    - ovirt-ansible-roles
    - openshift-ansible
    - python-ovirt-engine-sdk4
  pre_tasks:
    - name: Create an ssh key-pair for OpenShift admin
      user:
        name: root
        generate_ssh_key: yes
        ssh_key_file: .ssh/id_rsa_ssh_ocp_admin

  roles:
    - oVirt.image-template
    - oVirt.vm-infra

- name: post installation tasks on the bastion machine
  hosts: rhel-bastion
  tasks:
    - name: create ovirt-engine PKI dir
      file:
        state: directory
        dest: /etc/pki/ovirt-engine/
    - name: Copy the engine ca cert to the bastion machine
      copy:
        src: "{{ engine_cafile }}"
        dest: "{{ engine_cafile }}"
    - name: Copy the secured vars to the bastion machine
      copy:
        src: secure_vars.yaml
        dest: secure_vars.yaml
        decrypt: false
    - file:
        state: directory
        path: /root/.ssh
    - name: copy the OpenShift_admin keypair to the bastion machine
      copy:
        src: "{{ item }}"
        dest: "{{ item }}"
        mode: 0600
      with_items:
        - /root/.ssh/id_rsa_ssh_ocp_admin
        - /root/.ssh/id_rsa_ssh_ocp_admin.pub

```

1 Manager マシンの FQDN

2 `<qcow_url>` は、Red Hat Enterprise Linux KVM Guest Imageのダウンロードリンクに置き換えます。Red Hat Enterprise Linux KVM Guest Imageには、このPlaybookに必要な **cloud-init** パッケージが含まれます。Red Hat Enterprise Linux を使用しない場合は、**cloud-init** パッケージをダウンロードし、このPlaybook を実行する前に手動でインストールします。

8. bastion 仮想マシンを作成します。

```
# ansible-playbook -i localhost create-bastion-machine-playbook.yaml -e @secure_vars.yaml
--ask-vault-pass
```

9. 管理ポータルにログインします。
10. **Compute** → **仮想マシン** をクリックし、**rhel-bastion** 仮想マシンが正常に作成されたことを確認します。

## 21.2. BASTION 仮想マシンを使用した OPENSIFT CONTAINER PLATFORM のインストール

Red Hat Virtualization で bastion 仮想マシンを使用して OpenShift Container Platform をインストールします。

### 手順

1. **rhel-bastion** にログインします。
2. 以下の内容を含む **install\_ocp.yaml** ファイルを作成します。

```
---
- name: Openshift on RHV
  hosts: localhost
  connection: local
  gather_facts: false

  vars_files:
    - vars.yaml
    - secure_vars.yaml

  pre_tasks:
    - ovirt_auth:
      url: "{{ engine_url }}"
      username: "{{ engine_user }}"
      password: "{{ engine_password }}"
      insecure: "{{ engine_insecure }}"
      ca_file: "{{ engine_cafile | default(omit) }}"

  roles:
    - role: openshift_ovirt

- import_playbook: setup_dns.yaml
- import_playbook: /usr/share/ansible/openshift-ansible/playbooks/prerequisites.yaml
- import_playbook: /usr/share/ansible/openshift-ansible/playbooks/openshift-
node/network_manager.yml
- import_playbook: /usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yaml
```

3. 以下の内容を含む **setup\_dns.yaml** ファイルを作成します。

```
- hosts: masters
  strategy: free
  tasks:
    - shell: "echo {{ ansible_default_ipv4.address }} {{ inventory_hostname }} etcd.{{
inventory_hostname.split('.', 1)[1] }} openshift-master.{{ inventory_hostname.split('.', 1)[1] }}"
```

```

openshift-public-master.{{ inventory_hostname.split('.', 1)[1] }} docker-registry-default.apps.{{
inventory_hostname.split('.', 1)[1] }} webconsole.openshift-web-console.svc registry-console-
default.apps.{{ inventory_hostname.split('.', 1)[1] }} >> /etc/hosts"
  when: openshift_ovirt_all_in_one is defined | ternary((openshift_ovirt_all_in_one | bool),
false)

```

4. 以下の内容を含む Ansible インベントリファイル `/etc/ansible/openshift_3_11.hosts` を作成します。

```

[workstation]
localhost ansible_connection=local

[all:vars]
openshift_ovirt_dns_zone="{{ public_hosted_zone }}"
openshift_web_console_install=true
openshift_master_overwrite_named_certificates=true
openshift_master_cluster_hostname="openshift-master.{{ public_hosted_zone }}"
openshift_master_cluster_public_hostname="openshift-public-master.{{ public_hosted_zone
}}"
openshift_master_default_subdomain="{{ public_hosted_zone }}"
openshift_public_hostname="{{openshift_master_cluster_public_hostname}}"
openshift_deployment_type=openshift-enterprise
openshift_service_catalog_image_version="{{ openshift_image_tag }}"

[OSEv3:vars]
# General variables
debug_level=1
containerized=false
ansible_ssh_user=root
os_firewall_use_firewalld=true
openshift_enable_excluders=false
openshift_install_examples=false
openshift_clock_enabled=true
openshift_debug_level="{{ debug_level }}"
openshift_node_debug_level="{{ node_debug_level | default(debug_level,true) }}"
osn_storage_plugin_deps=[]
openshift_master_bootstrap_auto_approve=true
openshift_master_bootstrap_auto_approver_node_selector={"node-
role.kubernetes.io/master":"true"}
osm_controller_args={"experimental-cluster-signing-duration": ["20m"]}
osm_default_node_selector="node-role.kubernetes.io/compute=true"
openshift_enable_service_catalog=false

# Docker
container_runtime_docker_storage_type=overlay2
openshift_docker_use_system_container=false

[OSEv3:children]
nodes
masters
etcd
lb

[masters]

```

```
[nodes]
[etcd]
[lb]
```

5. Red Hat Enterprise Linux KVM Guest Imageのダウンロードリンクを取得します。
  - a. [Red Hat カスタマーポータル \(Red Hat Enterprise Linux のダウンロード\)](#) に移動します。
  - b. **製品ソフトウェア** タブで、**Red Hat Enterprise Linux KVM Guest Image**を見つけます。
  - c. **Download Now** を右クリックし、リンクをコピーして保存します。  
bastion 仮想マシンの作成時にコピーしたリンクを使用しないでください。ダウンロードリンクでは、短期間しか有効ではないので、インストール Playbook を実行する前にコピーする必要があります。
6. 以下の内容で `vars.yaml` ファイルを作成し、そのパラメーター値を更新します。

```
---
# For detailed documentation of variables, see
# openshift_ovirt: https://github.com/openshift/openshift-
# ansible/tree/master/roles/openshift_ovirt#role-variables
# openshift installation: https://github.com/openshift/openshift-ansible/tree/master/inventory
engine_url: https://<Manager_FQDN>/ovirt-engine/api 1
engine_user: admin@internal
engine_password: "{{ engine_password }}"
engine_insecure: false
engine_cafile: /etc/pki/ovirt-engine/ca.pem

openshift_ovirt_vm_manifest:
  - name: 'master'
    count: 1
    profile: 'master_vm'
  - name: 'compute'
    count: 0
    profile: 'node_vm'
  - name: 'lb'
    count: 0
    profile: 'node_vm'
  - name: 'etcd'
    count: 0
    profile: 'node_vm'
  - name: infra
    count: 0
    profile: node_vm

# Currently, only all-in-one installation (`openshift_ovirt_all_in_one: true`) is supported.
# Multi-node installation (master and node VMs installed separately) will be supported in a
# future release.
openshift_ovirt_all_in_one: true
openshift_ovirt_cluster: Default
openshift_ovirt_data_store: data
openshift_ovirt_ssh_key: "{{ lookup('file', '/root/.ssh/id_rsa_ssh_ocp_admin.pub') }}"

public_hosted_zone:
# Uncomment to disable install-time checks, for smaller scale installations
#openshift_disable_check: memory_availability,disk_availability,docker_image_availability
```

```
qcow_url: <RHEL_KVM_guest_image_download_link> 2
image_path: /var/tmp
template_name: rhelguest7
template_cluster: "{{ openshift_ovirt_cluster }}"
template_memory: 4GiB
template_cpu: 1
template_disk_storage: "{{ openshift_ovirt_data_store }}"
template_disk_size: 100GiB
template_nics:
  - name: nic1
    profile_name: ovirtmgmt
    interface: virtio

debug_vm_create: false
wait_for_ip: true
vm_infra_wait_for_ip_retries: 30
vm_infra_wait_for_ip_delay: 20

node_item: &node_item
  cluster: "{{ openshift_ovirt_cluster }}"
  template: "{{ template_name }}"
  memory: "8GiB"
  cores: "2"
  high_availability: true
  disks:
    - name: docker
      size: 15GiB
      interface: virtio
      storage_domain: "{{ openshift_ovirt_data_store }}"
    - name: openshift
      size: 30GiB
      interface: virtio
      storage_domain: "{{ openshift_ovirt_data_store }}"
  state: running
  cloud_init:
    root_password: "{{ root_password }}"
    authorized_ssh_keys: "{{ openshift_ovirt_ssh_key }}"
    custom_script: "{{ cloud_init_script_node | to_nice_yaml }}"

openshift_ovirt_vm_profile:
  master_vm:
    <<: *node_item
    memory: 16GiB
    cores: "{{ vm_cores | default(4) }}"
    disks:
      - name: docker
        size: 15GiB
        interface: virtio
        storage_domain: "{{ openshift_ovirt_data_store }}"
      - name: openshift_local
        size: 30GiB
        interface: virtio
        storage_domain: "{{ openshift_ovirt_data_store }}"
      - name: etcd
        size: 25GiB
```

```

    interface: virtio
    storage_domain: "{{ openshift_ovirt_data_store }}"
  cloud_init:
    root_password: "{{ root_password }}"
    authorized_ssh_keys: "{{ openshift_ovirt_ssh_key }}"
    custom_script: "{{ cloud_init_script_master | to_nice_yaml }}"
  node_vm:
    <<: *node_item
  etcd_vm:
    <<: *node_item
  lb_vm:
    <<: *node_item

cloud_init_script_node: &cloud_init_script_node
packages:
  - ovirt-guest-agent
runcmd:
  - sed -i 's/# ignored_nics = ./ignored_nics = docker0 tun0 /' /etc/ovirt-guest-agent.conf
  - systemctl enable ovirt-guest-agent
  - systemctl start ovirt-guest-agent
  - mkdir -p /var/lib/docker
  - mkdir -p /var/lib/origin/openshift.local.volumes
  - /usr/sbin/mkfs.xfs -L dockerlv /dev/vdb
  - /usr/sbin/mkfs.xfs -L ocplv /dev/vdc
mounts:
  - [ '/dev/vdb', '/var/lib/docker', 'xfs', 'defaults,gquota' ]
  - [ '/dev/vdc', '/var/lib/origin/openshift.local.volumes', 'xfs', 'defaults,gquota' ]
power_state:
  mode: reboot
  message: cloud init finished - boot and install openshift
  condition: True
cloud_init_script_master:
  <<: *cloud_init_script_node
runcmd:
  - sed -i 's/# ignored_nics = ./ignored_nics = docker0 tun0 /' /etc/ovirt-guest-agent.conf
  - systemctl enable ovirt-guest-agent
  - systemctl start ovirt-guest-agent
  - mkdir -p /var/lib/docker
  - mkdir -p /var/lib/origin/openshift.local.volumes
  - mkdir -p /var/lib/etcd
  - /usr/sbin/mkfs.xfs -L dockerlv /dev/vdb
  - /usr/sbin/mkfs.xfs -L ocplv /dev/vdc
  - /usr/sbin/mkfs.xfs -L etcdlv /dev/vdd
mounts:
  - [ '/dev/vdb', '/var/lib/docker', 'xfs', 'defaults,gquota' ]
  - [ '/dev/vdc', '/var/lib/origin/openshift.local.volumes', 'xfs', 'defaults,gquota' ]
  - [ '/dev/vdd', '/var/lib/etcd', 'xfs', 'defaults,gquota' ]

```

1 Manager マシンの FQDN

2 `<qcow_url>` は、Red Hat Enterprise Linux KVM Guest Imageのダウンロードリンクに置き換えます。Red Hat Enterprise Linux KVM Guest Imageには、このPlaybookに必要な **cloud-init** パッケージが含まれます。Red Hat Enterprise Linux を使用しない場合は、**cloud-init** パッケージをダウンロードし、このPlaybook を実行する前に手動でインストールします。

## 7. OpenShift Container Platform のインストール

```
# export ANSIBLE_ROLES_PATH="/usr/share/ansible/roles:/usr/share/ansible/openshift-ansible/roles"
# export ANSIBLE_JINJA2_EXTENSIONS="jinja2.ext.do"
# ansible-playbook -i /etc/ansible/openshift_3_11.hosts install_ocp.yaml -e @vars.yaml -e @secure_vars.yaml --ask-vault-pass
```

8. 各インフラストラクチャーインスタンスに、ルーターの DNS エントリーを作成します。
9. ルーターがトラフィックをアプリケーションに渡すことができるように、ラウンドロビンのルーティングを設定します。
10. OpenShift Container Platform Web コンソールの DNS エントリーを作成します。
11. ロードバランサーノードの IP アドレスを指定します。

## 第22章 OPENSTACK の設定

### 22.1. 概要

OpenShift Container Platform は、OpenStack にデプロイする際に、OpenStack Cinder ボリュームをアプリケーションデータの永続ストレージとして使用など、OpenStack インフラストラクチャーにアクセスするように設定できます。



#### 重要

OpenShift Container Platform 3.11 は Red Hat OpenStack Platform 13 での使用をサポートしています。

OpenShift Container Platform の最新リリースは、最新の Red Hat OpenStack Platform のロングライフリリースおよび中間リリースの両方をサポートします。OpenShift Container Platform と Red Hat OpenStack Platform のリリースサイクルは異なり、テストされるバージョンは両方の製品のリリース日によって変わる可能性があります。

### 22.2. 作業開始前の準備

#### 22.2.1. OpenShift Container Platform SDN

デフォルトの OpenShift Container Platform SDN は [OpenShiftSDN](#) です。もう1つのオプションとして [Kuryr SDN](#) を使用できます。

#### 22.2.2. Kuryr SDN

Kuryr は、Neutron および Octavia を使用して Pod およびサービスのネットワークを提供する CNI プラグインです。これは主に、OpenStack 仮想マシンで実行される OpenShift Container Platform クラスター用に設計されています。Kuryr は、OpenShift Container Platform Pod を OpenStack SDN にプラグインしてネットワークのパフォーマンスを強化します。さらに、OpenShift Container Platform Pod と OpenStack 仮想インスタンス間の接続を可能にします。

Kuryr は、カプセル化された OpenStack テナントネットワーク上の OpenShift Container Platform デプロイメントに使用することが推奨されます。これは、OpenStack ネットワークでカプセル化された OpenShift SDN を実行するなど、二重のカプセル化を防ぐために必要です。Kuryr は、VXLAN、GRE、または GENEVE が必要な場合に常に推奨されます。

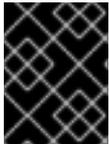
逆に、Kuryr を実装すると、以下の場合に意味はありません。

- Cisco ACI や Juniper Contrail などのプロバイダーネットワーク、テナント VLAN、またはサードパーティーの商用 SDN を使用する。
- デプロイメントに、少数のハイパーバイザーまたは OpenShift Container Platform 仮想マシンノードで、数多くのサービスを使用する。各 OpenShift Container Platform サービスが、必要なロードバランサーをホストする OpenStack で Octavia Amphora 仮想マシンを作成する。

Kuryr SDN を有効にするには、使用する環境が以下の要件を満たしている必要があります。

- OpenStack 13 以降を実行する
- Octavia を使用したオーバークラウド
- Neutron トランクポートの拡張を有効にする

- ML2/OVS Neutron ドライバーが OpenvSwitch ファイアウォールドライバーを使用する場合、ovs-hybrid の代わりに使用しておく。



### 重要

OpenStack 13.0.13 で Kuryr を使用するには、Kuryr コンテナイメージはバージョン 3.11.306 以上である必要があります。

## 22.2.3. OpenShift Container Platform の前提条件

OpenShift Container Platform を正常にデプロイするには数多くの要件を満たす必要があります。これには、Ansible を使用して OpenShift Container Platform を実際にインストールする前に一連のインフラストラクチャーおよびホスト設定の手順を実行する必要があります。以下のサブセクションでは、OpenStack 環境の OpenShift Container Platform に必要な前提条件および設定変更について詳しく説明します。



### 注記

このリファレンス環境での OpenStack CLI コマンドはすべて、director ノードとは異なるノード内の CLI **openstack** コマンドを使用して実行されます。このコマンドは、Ansible バージョン 2.6 以降とパッケージの競合を防ぐために、他のノードで実行されません。指定されたリポジトリに以下のパッケージをインストールしてください。

以下に例を示します。

[リポジトリの設定](#) に従い、`rhel-7-server-openstack-13-tools-rpms` と必要な OpenShift Container Platform リポジトリを有効にします。

```
$ sudo subscription-manager repos \
--enable rhel-7-server-openstack-{rhosp_version}-tools-rpms \
--enable rhel-7-server-openstack-14-tools-rpms
$ sudo subscription-manager repo-override --repo=rhel-7-server-openstack-14-tools-rpms --
add=includepkgs:"python2-openstacksdk.* python2-keystoneauth1.* python2-os-service-types.*"
$ sudo yum install -y python2-openstackclient python2-heatclient python2-octaviaclient ansible
```

パッケージは以下のバージョン以上であることを確認します (`rpm -q <package_name>` を使用します)。

- **python2-openstackclient - 3.14.1-1**
- **python2-heatclient 1.14.0-1**
- **python2-octaviaclient 1.4.0-1**
- **python2-openstacksdk 0.17.2**

### 22.2.3.1. Octavia の有効化: OpenStack の LBaaS (Load Balancing as a Service)

Octavia は、外部の受信トラフィックの負荷を分散し、各種アプリケーションについての OpenShift Container Platform マスターサービスの単一ビューを提供するために OpenShift Container Platform と併用することが推奨されているサポート対象のロードバランサーソリューションです。

Octavia を有効にするには、Octavia サービスを OpenStack オーバークラウドのインストール時に組み込むか、またはオーバークラウドがすでに存在する場合はアップグレードする必要があります。以下の

手順は、Octavia を有効にするためのカスタム手順を含まない基本的な手順であり、これらはオーバークラウドのクリーンインストールまたはオーバークラウドの更新の両方に適用されます。



## 注記

以下の手順では、Octavia を使用する場合に OpenStack のデプロイメント時に必要となる主な手順のみを説明します。詳細は、[Installation of OpenStack](#) のドキュメントを参照してください。また、レジストリーの方法が変更されることにも留意してください。詳細は、[Registry Methods](#) のドキュメントを参照してください。以下の例では、ローカルレジストリーの方法を使用しています。

ローカルレジストリーを使用している場合、イメージをレジストリーにアップロードするためのテンプレートを作成します。以下は例になります。

```
(undercloud) $ openstack overcloud container image prepare \
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/octavia.yaml \
--namespace=registry.access.redhat.com/rhosp13 \
--push-destination=<local-ip-from-undercloud.conf>:8787 \
--prefix=openstack- \
--tag-from-label {version}-{release} \
--output-env-file=/home/stack/templates/overcloud_images.yaml \
--output-images-file /home/stack/local_registry_images.yaml
```

作成された `local_registry_images.yaml` に Octavia イメージが含まれることを確認します。

## ローカルレジストリーファイルの Octavia イメージ

```
...
- imagename: registry.access.redhat.com/rhosp13/openstack-octavia-api:13.0-43
  push_destination: <local-ip-from-undercloud.conf>:8787
- imagename: registry.access.redhat.com/rhosp13/openstack-octavia-health-manager:13.0-45
  push_destination: <local-ip-from-undercloud.conf>:8787
- imagename: registry.access.redhat.com/rhosp13/openstack-octavia-housekeeping:13.0-45
  push_destination: <local-ip-from-undercloud.conf>:8787
- imagename: registry.access.redhat.com/rhosp13/openstack-octavia-worker:13.0-44
  push_destination: <local-ip-from-undercloud.conf>:8787
```



## 注記

Octavia コンテナのバージョンは、インストールされている特定の Red Hat OpenStack Platform リリースによって異なります。

以下の手順では、[registry.redhat.io](#) からアンダークラウドノードにコンテナイメージをプルします。ネットワークおよびアンダークラウドディスクの速度によっては、このプロセスに時間がかかる場合があります。

```
(undercloud) $ sudo openstack overcloud container image upload \
--config-file /home/stack/local_registry_images.yaml \
--verbose
```

Octavia ロードバランサーは OpenShift API にアクセスするために使用されるため、それらのリスナーの接続のデフォルトタイムアウトを増やす必要があります。デフォルトのタイムアウトは 50 秒です。以下のファイルをオーバークラウドデプロイコマンドに渡し、タイムアウトを 20 分に増やします。

```
(undercloud) $ cat octavia_timeouts.yaml
parameter_defaults:
  OctaviaTimeoutClientData: 1200000
  OctaviaTimeoutMemberData: 1200000
```



### 注記

これは Red Hat OpenStack Platform 14 以降では必要ありません。

Octavia を使用してオーバークラウドをインストールまたは更新します。

```
openstack overcloud deploy --templates \
.
.
.
-e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/octavia.yaml \
-e octavia_timeouts.yaml
.
.
.
```



### 注記

上記のコマンドには、Octavia に関連付けられたファイルのみが含まれます。このコマンドは、OpenStack の特定のインストールに応じて変わります。詳細は、公式の OpenStack ドキュメントを参照してください。Octavia インストールのカスタマイズについての詳細は、[Octavia デプロイメントのプランニング](#) を参照してください。



### 注記

Kuryr SDN が使用される場合、オーバークラウドのインストールにおいてトランクの拡張が Neutron で有効にされる必要があります。これは、Director デプロイメントでデフォルトで有効にされます。Neutron バックエンドが ML2/OVS の場合、デフォルトの ovs-hybrid の代わりに openvswitch ファイアウォールを使用します。バックエンドが ML2/OVN の場合には変更の必要がありません。

## 22.2.3.2. OpenStack ユーザーアカウント、プロジェクトおよびロールの作成

OpenShift Container Platform のインストール前に、Red Hat OpenStack Platform (RHOSP) 環境には **テナント** と呼ばれるプロジェクトが必要になります。これは OpenShift Container Platform をインストールするために使用される OpenStack インスタンスを保管します。このプロジェクトには、ユーザーおよび **\_member\_** に設定されるユーザーのロールによる所有権が必要になります。

以下の手順は上記を実行する方法を説明しています。

OpenStack オーバークラウド管理者として以下を実行します。

1. RHOSP インスタンスを保管するために使用されるプロジェクト (テナント) を作成します。

```
$ openstack project create <project>
```

2. 以前に作成したプロジェクトの所有権を持つ RHOSP ユーザーを作成します。

```
$ openstack user create --password <password> <username>
```

3. ユーザーのロールを設定します。

```
$ openstack role add --user <username> --project <project> _member_
```

新規の RH OSP プロジェクトに割り当てられるデフォルトのクォータは、OpenShift Container Platform インストールについては十分な値ではありません。クォータを少なくとも 30 セキュリティーグループ、200 セキュリティーグループルール、および 200 ポートに引き上げます。

```
$ openstack quota set --secgroups 30 --secgroup-rules 200 --ports 200 <project>
```

1

1 **<project>** に、変更するプロジェクトの名前を指定します。

### 22.2.3.3. Kuryr SDN の追加手順

Kuryr SDN が有効にされている場合 (特に namespace の分離を使用している場合)、プロジェクトのクォータはこれらの最小要件を満たすよう引き上げます。

- 300 セキュリティーグループ: namespace ごとに1つ、ロードバランサーごとに1つ
- 150 ネットワーク: namespace ごとに1つ
- 150 サブネット: namespace ごとに1つ
- 500 セキュリティーグループルール
- 500 ポート: Pod ごとにポート1つ、プールが Pod 作成を迅速化するための追加ポート



#### 注記

これはグローバルな推奨事項ではありません。要件に合わせてクォータを調整します。

namespace の分離を使用している場合に、それぞれの namespace には新規ネットワークおよびサブネットが割り当てられます。さらに、セキュリティーグループを作成して namespace の Pod 間のトラフィックを有効化します。

```
$ openstack quota set --networks 150 --subnets 150 --secgroups 300 --secgroup-rules 500 --ports 500 <project>
```

1

1 **<project>** に、変更するプロジェクトの名前を指定します。

namespace の分離を有効にしている場合には、プロジェクトの作成後にプロジェクト ID を **octavia.conf** 設定ファイルに追加する必要があります。この手順により、必要な LoadBalancer セキュリティーグループがそのプロジェクトに属し、それらを namespace 全体でサービスの分離を実行するように更新できます。

1. プロジェクト ID を取得します。

```
$ openstack project show *<project>*
```

```

+-----+-----+
| Field   | Value           |
+-----+-----+
| description |                |
| domain_id | default         |
| enabled    | True            |
| id        | PROJECT_ID     |
| is_domain  | False           |
| name      | *<project>*    |
| parent_id | default         |
| tags      | []              |
+-----+-----+

```

2. コントローラーでプロジェクト ID を [filename]octavia.conf に追加し、octavia ワーカーを再起動します。

```

$ source stackrc # undercloud credentials
$ openstack server list
+-----+-----+-----+-----+-----+-----+
-----+
| ID                | Name          | Status | Networks |
| Image            | Flavor       |        |          |
+-----+-----+-----+-----+-----+
-----+
| 6bef8e73-2ba5-4860-a0b1-3937f8ca7e01 | controller-0 | ACTIVE |          |
|ctlplane=192.168.24.8 | overcloud-full | controller |
|
| dda3173a-ab26-47f8-a2dc-8473b4a67ab9 | compute-0   | ACTIVE |          |
|ctlplane=192.168.24.6 | overcloud-full | compute   |
|
+-----+-----+-----+-----+-----+
-----+

$ ssh heat-admin@192.168.24.8 # ssh into the controller(s)

controller-0$ vi /var/lib/config-data/puppet-generated/octavia/etc/octavia/octavia.conf
[controller_worker]
# List of project ids that are allowed to have Load balancer security groups
# belonging to them.
amp_secgroup_allowed_projects = PROJECT_ID

controller-0$ sudo docker restart octavia_worker

```

#### 22.2.3.4. RC ファイルの設定

プロジェクトを設定したら、OpenStack 管理者は、OpenShift Container Platform 環境を実装するユーザーに対し、必要なすべての情報を含めて RC ファイルを作成できます。

RC ファイルのサンプル:

```

$ cat path/to/examplerc
# Clear any old environment that may conflict.

```

```

for key in $( set | awk '{FS="="} /^OS_/ {print $1}' ); do unset $key ; done
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=<project-name>
export OS_USERNAME=<username>
export OS_PASSWORD=<password>
export OS_AUTH_URL=http://<ip>:5000/v3
export OS_CLOUDNAME=<cloud-name>
export OS_IDENTITY_API_VERSION=3

# Add OS_CLOUDNAME to PS1
if [ -z "${CLOUDPROMPT_ENABLED:-}" ]; then
  export PS1=${PS1:-""}
  export PS1=\${OS_CLOUDNAME:+"(\${OS_CLOUDNAME})"}\ $PS1
  export CLOUDPROMPT_ENABLED=1
fi

```



### 注記

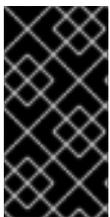
デフォルト値の `Changing_OS_PROJECT_DOMAIN_NAME` および `_OS_USER_DOMAIN_NAME` は、どちらも同じドメインを参照している限りサポートされます。

OpenStack director ノードまたはワークステーション内で OpenShift Container Platform 環境を実装するユーザーとして、以下のように認証情報の **source** を実行します。

```
$ source path/to/examplerc
```

### 22.2.3.5. OpenStack フレーバーの作成

OpenStack 内で、フレーバーは **nova** コンピューティングインスタンスのコンピュート、メモリー、およびストレージ容量を定義することで仮想サーバーのサイズを定義します。このリファレンスアーキテクチャー内のベースイメージは Red Hat Enterprise Linux 7.5 であるため、**m1.node** および **m1.master** のサイズが設定されたフレーバーが表22.1「OpenShift の最小システム要件」に示されるような仕様で作成されます。



### 重要

最低限のシステム要件を満たした場合にクラスターを実行できますが、パフォーマンスを改善するには、マスターノードで vCPU を増やすことが推奨されます。さらに、etcd がマスター上の同一の場所に配置されている場合は、メモリーを追加することも推奨されます。

表22.1 OpenShift の最小システム要件

ノードタイプ	CPU	RAM	ルートディスク	フレーバー
マスター	4	16 GB	45 GB	<b>m1.master</b>
ノード	1	8 GB	20 GB	<b>m1.node</b>

OpenStack 管理者として以下を実行します。

```
$ openstack flavor create <flavor_name> \  
  --id auto \  
  --ram <ram_in_MB> \  
  --disk <disk_in_GB> \  
  --vcpus <num_vcpus>
```

以下の例は、このリファレンス環境内でのフレーバーの作成について示しています。

```
$ openstack flavor create m1.master \  
  --id auto \  
  --ram 16384 \  
  --disk 45 \  
  --vcpus 4  
$ openstack flavor create m1.node \  
  --id auto \  
  --ram 8192 \  
  --disk 20 \  
  --vcpus 1
```



#### 注記

新規フレーバーを作成するために OpenStack 管理者権限にアクセスできない場合は、[表 22.1 「OpenShift の最小システム要件」](#) の要件を満たす OpenStack 環境内で既存フレーバーを使用します。

以下を実行して OpenStack フレーバーを検証します。

```
$ openstack flavor list
```

### 22.2.3.6. OpenStack キーペアの作成

Red Hat OpenStack Platform は、インスタンスへの **ssh** アクセスを許可するように作成されているため、**cloud-init** を使用して **ssh** パブリックキーを各インスタンスに配置します。Red Hat OpenStack Platform ではユーザーがプライベートキーを保持することを予想されています。



#### 警告

プライベートキーを失うと、インスタンスにアクセスできなくなります。

キーペアを生成するには、以下のコマンドを使用します。

```
$ openstack keypair create <keypair-name> > /path/to/<keypair-name>.pem
```

キーペアの作成は以下で検証できます。

```
$ openstack keypair list
```

キーペアが作成されたら、パーミッションを **600** に設定します。これにより、ファイルの所有者のみが該当ファイルの読み取り、および書き込みを行えるようになります。

```
$ chmod 600 /path/to/<keypair-name>.pem
```

### 22.2.3.7. OpenShift Container Platform の DNS の設定

DNS サービスは OpenShift Container Platform 環境における重要なコンポーネントです。DNS のプロバイダーの種類を問わず、組織は各種の OpenShift Container Platform コンポーネントを提供できるように特定のレコードを利用可能にしておく必要があります。



#### 警告

`/etc/hosts` の使用は有効ではありません。適切な DNS サービスがなければなりません。

DNS のキーシークレットを使用して、情報を OpenShift Ansible インストールに提供することができ、これにより、ターゲットインスタンスの A レコードおよび各種の OpenShift Container Platform コンポーネントが自動的に追加されます。このプロセス設定については、後程 OpenShift Ansible インストーラーの設定との関連で説明します。

DNS サーバーへのアクセスが必要になることが予想されます。アクセスについてのヘルプが必要な場合は、[Red Hat Labs DNS Helper](#) を使用できます。

#### アプリケーション DNS

OpenShift で提供されるアプリケーションはポート 80/TCP および 443/TCP のルーターによってアクセス可能です。ルーターは **ワイルドカード** レコードを使用して特定のサブドメインにあるすべてのホスト名を同じ IP アドレスにマップできます。この際、それぞれの名前に別個のレコードを用意する必要はありません。

これにより、OpenShift Container Platform は任意の名前が該当するサブドメインにある限り、それらの名前前のアプリケーションを追加することができます。

たとえば、**\*.apps.example.com** のワイルドカードレコードを使用すると、**tax.apps.example.com** および **home-goods.apps.example.com** の DNS 名検索により、同一の IP アドレス **10.19.x.y** が返されます。すべてのトラフィックは OpenShift ルーターに転送されます。ルーターはクエリーの HTTP ヘッダーを検査し、それらを正しい宛先に転送します。

Octavia などのロードバランサーを使用して、ホストアドレスの 10.19.x.y、ワイルドカード DNS レコードは以下のように追加できます。

表22.2 ロードバランサー DNS レコード

IP アドレス	ホスト名	目的
---------	------	----

IP アドレス	ホスト名	目的
10.19.x.y	*.apps.example.com	アプリケーション Web サービスへのユーザーアクセス

### 22.2.3.8. OpenStack 経由での OpenShift Container Platform ネットワークの作成

このセグメントで説明されているように OpenShift Container Platform を Red Hat OpenStack Platform でデプロイすると、**パブリック** および **内部** の2つのネットワークが必要になります。

#### パブリックネットワーク

**パブリック** ネットワークは外部アクセスを含むネットワークであり、外部からアクセスできるものです。**パブリック** ネットワークは OpenStack 管理者によってのみ作成されます。

以下のコマンドは、**パブリック** ネットワークアクセス用の OpenStack プロバイダーネットワークを作成する例を示しています。

OpenStack 管理者として (overcloudrc アクセス)、以下を実行します。

```
$ source /path/to/examplerc

$ openstack network create <public-net-name> \
  --external \
  --provider-network-type flat \
  --provider-physical-network datacentre

$ openstack subnet create <public-subnet-name> \
  --network <public-net-name> \
  --dhcp \
  --allocation-pool start=<float_start_ip>,end=<float_end_ip> \
  --gateway <ip> \
  --subnet-range <CIDR>
```

ネットワークおよびサブネットが作成されたら、以下のように検証します。

```
$ openstack network list
$ openstack subnet list
```



#### 注記

<float\_start\_ip> および <float\_end\_ip> は、**パブリック** ネットワークのラベルが付けられたネットワークに提供される関連付けられた Floating IP プールです。Classless Inter-Domain Routing (CIDR) は <ip>/<routing\_prefix> の形式を使用します (例: 10.0.0.1/24)。

#### 内部ネットワーク

**内部** ネットワークがネットワークの設定時にルーター経由で **パブリック** ネットワークに接続されます。これにより、**内部** ネットワークに割り当てられている各 Red Hat OpenStack Platform インスタンスには、パブリックアクセス用の **パブリック** ネットワークから Floating IP を要求する機能が付与され

ます。内部 ネットワークは、`openshift_openstack_private_network_name` を設定することにより OpenShift Ansible インストーラーによって自動的に作成されます。OpenShift Ansible インストーラーに必要な変更についての詳細は後で説明します。

### 22.2.3.9. OpenStack デプロイメントホストセキュリティーグループの作成

OpenStack ネットワークはユーザーがネットワーク上の各インスタンスに適用できる受信および送信トラフィックフィルターを定義することを許可します。これにより、ユーザーはインスタンスサービスの機能に基づいて各インスタンスへのネットワークトラフィックを制限でき、ホストベースのフィルターに依存する必要はありません。OpenShift Ansible インストーラーは、デプロイメントホスト以外の OpenShift Container Platform クラスターを設定するホストのそれぞれのタイプに必要なすべてのポートおよびサービスの作成を適切に処理します。

以下のコマンドは、デプロイメントホストにルールが設定されていない状態で空のセキュリティーグループを作成します。

```
$ source path/to/examplerc
$ openstack security group create <deployment-sg-name>
```

セキュリティーグループの作成を確認します。

```
$ openstack security group list
```

#### デプロイメントホストセキュリティーグループ

デプロイメントインスタンスの場合は、受信 `ssh` のみを許可する必要があります。このインスタンスは、オペレーターに対して OpenShift Container Platform 環境をデプロイし、モニターし、管理するための安定したベースを提供することを目的として存在します。

表22.3 デプロイメントホストのセキュリティーグループの TCP ポート

ポート/プロトコル	Service	リモートソース	目的
ICMP	ICMP	すべて	ping、traceroute などを許可。
22/TCP	SSH	すべて	セキュアなシェルログイン

上記のセキュリティーグループルールの作成は以下のように行われます。

```
$ source /path/to/examplerc
$ openstack security group rule create \
  --ingress \
  --protocol icmp \
  <deployment-sg-name>
$ openstack security group rule create \
  --ingress \
  --protocol tcp \
  --dst-port 22 \
  <deployment-sg-name>
```

セキュリティーグループルールの検証は以下のように行われます。

```
$ openstack security group rule list <deployment-sg-name>
+-----+-----+-----+-----+-----+
| ID | IP Protocol | IP Range | Port Range | Remote Security Group |
+-----+-----+-----+-----+-----+
| 7971fc03-4bfe-4153-8bde-5ae0f93e94a8 | icmp | 0.0.0.0/0 | | None |
| b8508884-e82b-4ee3-9f36-f57e1803e4a4 | None | None | | None |
| cb914caf-3e84-48e2-8a01-c23e61855bf6 | tcp | 0.0.0.0/0 | 22:22 | None |
| e8764c02-526e-453f-b978-c5ea757c3ac5 | None | None | | None |
+-----+-----+-----+-----+-----+
```

### 22.2.3.10. OpenStack Cinder ボリューム

OpenStack Block Storage は、**cinder** サービスを使用して永続ブロックストレージを管理します。ブロックストレージは OpenStack ユーザーによる各種の OpenStack インスタンスに割り当て可能なボリュームの作成を可能にします。

#### 22.2.3.10.1. Docker ボリューム

マスターおよびノードインスタンスには、**docker** イメージを保管するためのボリュームが含まれます。このボリュームの目的は、大規模なイメージまたはコンテナによりノードのパフォーマンスが下がったり、既存ノードの機能に影響が及ばないようにすることにあります。



#### 注記

コンテナを実行するには、最小 15GB の docker ボリュームが必要です。これについては、各ノードが実行するコンテナのサイズおよび数に応じて調整が必要になる可能性があります。

docker ボリュームは、変数 **openshift\_openstack\_docker\_volume\_size** を使用して OpenShift Ansible インストーラーによって作成されます。OpenShift Ansible インストーラーに必要な変更についての詳細は後で説明します。

#### 22.2.3.10.2. レジストリーボリューム

OpenShift イメージレジストリーには、レジストリーが別のノードに移行する必要がある場合でもイメージを保存できるようにするために **cinder** ボリュームが必要です。以下の手順では、OpenStack を使用してイメージレジストリーを作成する方法を説明します。ボリュームが作成されると、後述されるようにボリューム ID がパラメーター **openshift\_hosted\_registry\_storage\_openstack\_volumelD** により OpenShift Ansible Installer **OSEv3.yml** ファイルに組み込まれます。

```
$ source /path/to/examplerc
$ openstack volume create --size <volume-size-in-GB> <registry-name>
```



#### 注記

レジストリーのボリュームサイズは 30GB 以上である必要があります。

ボリュームの作成を確認します。

```
$ openstack volume list
+-----+-----+-----+-----+-----+
| ID | Name | Status | Size | Attached to |
+-----+-----+-----+-----+-----+
```

```
+-----+
| d65209f0-9061-4cd8-8827-ae6e2253a18d | <registry-name> | available | 30 | |
+-----+
```

### 22.2.3.11. デプロイメントインスタンスの作成および設定

デプロイメントインスタンスのロールは、OpenShift Container Platform のデプロイメントおよび管理のユーティリティーホストとして機能することにあります。

#### デプロイメントホストのネットワークおよびルーターの作成

インスタンスの作成前に、内部ネットワークおよびルーターはデプロイメントホストとの通信用に作成される必要があります。以下のコマンドは、そのネットワークおよびルーターを作成します。

```
$ source path/to/examplerc

$ openstack network create <deployment-net-name>

$ openstack subnet create --network <deployment-net-name> \
  --subnet-range <subnet_range> \
  --dns-nameserver <dns-ip> \
  <deployment-subnet-name>

$ openstack router create <deployment-router-name>

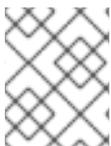
$ openstack router set --external-gateway <public-net-name> <deployment-router-name>

$ openstack router add subnet <deployment-router-name> <deployment-subnet-name>
```

#### デプロイメントインスタンスのデプロイ

作成されるネットワークおよびセキュリティグループで、インスタンスをデプロイします。

```
$ domain=<domain>
$ netid1=$(openstack network show <deployment-net-name> -f value -c id)
$ openstack server create \
  --nic net-id=$netid1 \
  --flavor <flavor> \
  --image <image> \
  --key-name <keypair> \
  --security-group <deployment-sg-name> \
  deployment.$domain
```



#### 注記

**m1.small** フレーバーがデフォルトで存在しない場合、1vCPU および 2GB RAM の要件を満たす既存フレーバーを使用します。

#### Floating IP の作成およびデプロイメントインスタンスへの追加

デプロイメントインスタンスの作成後に、Floating IP を作成し、これをインスタンスに割り当てる必要があります。以下は例になります。

```
$ source /path/to/examplerc
```

```
$ openstack floating ip create <public-network-name>
+-----+
| Field          | Value                               |
+-----+
| created_at     | 2017-08-24T22:44:03Z               |
| description    |                                     |
| fixed_ip_address | None                                |
| floating_ip_address | 10.20.120.150                       |
| floating_network_id | 084884f9-d9d2-477a-bae7-26dbb4ff1873 |
| headers       |                                     |
| id            | 2bc06e39-1efb-453e-8642-39f910ac8fd1 |
| port_id       | None                                |
| project_id    | ca304df9e9a04597b16d253efd0e2332 |
| project_id    | ca304df9e9a04597b16d253efd0e2332 |
| revision_number | 1                                    |
| router_id     | None                                |
| status        | DOWN                                |
| updated_at    | 2017-08-24T22:44:03Z               |
+-----+
```

上記の出力の **floating\_ip\_address** フィールドは Floating IP **10.20.120.150** が作成されていることを示しています。この IP をデプロイメントインスタンスに割り当てるには、以下のコマンドを実行します。

```
$ source /path/to/examplerc
$ openstack server add floating ip <deployment-instance-name> <ip>
```

たとえば、インスタンス **deployment.example.com** に IP **10.20.120.150** が割り当てられる場合、コマンドは以下ようになります。

```
$ source /path/to/examplerc
$ openstack server add floating ip deployment.example.com 10.20.120.150
```

### RC ファイルのデプロイメントホストへの追加

デプロイメントホストの存在を確認したら、以下のように先に作成した RC ファイルを **scp** でデプロイメントホストにコピーします。

```
scp <rc-file-deployment-host> cloud-user@<ip>:/home/cloud-user/
```

### 22.2.3.12. OpenShift Container Platform のデプロイメントホスト設定

以下のサブセクションでは、デプロイメントインスタンスを適切に設定するために必要なすべての手順について説明しています。

#### デプロイメントホストを Jump host として使用できるように ~/.ssh/config を設定する

OpenShift Container Platform 環境に簡単に接続するには、以下の手順に従ってください。

OpenStack director ノードまたはローカルワークステーションで プライベートキー <keypair-name>.pem を使用して以下を実行します。

```
$ exec ssh-agent bash
```

```
$ ssh-add /path/to/<keypair-name>.pem
Identity added: /path/to/<keypair-name>.pem (/path/to/<keypair-name>.pem)
```

~/**.ssh/config** ファイルに追加します。

```
Host deployment
  HostName    <deployment_fqdn_hostname OR IP address>
  User        cloud-user
  IdentityFile /path/to/<keypair-name>.pem
  ForwardAgent yes
```

認証エージェント接続の転送を有効にする **-A** オプションを指定し、デプロイメントホストに対して **ssh** を実行します。

パーミッションが ~/**.ssh/config** ファイルの所有者に対して読み取り/書き込み専用を設定されていることを確認します。

```
$ chmod 600 ~/.ssh/config
```

```
$ ssh -A cloud-user@deployment
```

デプロイメントホストにログインしたら、**SSH\_AUTH\_SOCK** をチェックして ssh エージェント転送が機能することを確認します。

```
$ echo "$SSH_AUTH_SOCK"
/tmp/ssh-NDFDQD02qB/agent.1387
```

### Subscription Manager および OpenShift Container Platform リポジトリーの有効化

デプロイメントインスタンス内で、Red Hat Subscription Manager への登録を行います。これは認証情報を使用して実行できます。

```
$ sudo subscription-manager register --username <user> --password '<password>'
```

または、アクティベーションキーを使用できます。

```
$ sudo subscription-manager register --org="<org_id>" --activationkey=<keyname>
```

登録が完了したら、以下のようにレジストリーを有効にします。

```
$ sudo subscription-manager repos \
  --enable="rhel-7-server-rpms" \
  --enable="rhel-7-server-extras-rpms" \
  --enable="rhel-7-server-ose-3.11-rpms" \
  --enable="rhel-7-server-ansible-2.6-rpms" \
  --enable="rhel-7-server-openstack-13-rpms" \
  --enable="rhel-7-server-openstack-13-tools-rpms"
```



#### 注記

[リポジトリーの設定](#) を参照し、有効にする OpenShift Container Platform リポジトリーおよび Ansible バージョンを確認します。上記のファイルはサンプルであることに注意してください。

## デプロイメントホストで必要なパッケージ

以下のパッケージがデプロイメントホストでインストールされる必要があります。

以下のパッケージをインストールします。

- **openshift-ansible**
- **python-openstackclient**
- **python2-heatclient**
- **python2-octaviaclient**
- **python2-shade**
- **python-dns**
- **git**
- **ansible**

```
$ sudo yum -y install openshift-ansible python-openstackclient python2-heatclient python2-octaviaclient python2-shade python-dns git ansible
```

## Ansible の設定

**ansible** は、マスターおよびノードインスタンスで登録やパッケージのインストール、および OpenShift Container Platform 環境のデプロイメントを実行するためにデプロイメントインスタンスにインストールされます。

Playbook を実行する前に、デプロイする環境を反映させるために **ansible.cfg** ファイルを作成する必要があります。

```
$ cat ~/ansible.cfg

[defaults]
forks = 20
host_key_checking = False
remote_user = openshift
gathering = smart
fact_caching = jsonfile
fact_caching_connection = $HOME/ansible/facts
fact_caching_timeout = 600
log_path = $HOME/ansible.log
nocows = 1
callback_whitelist = profile_tasks
inventory = /usr/share/ansible/openshift-ansible/playbooks/openstack/inventory.py,/home/cloud-user/inventory

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=600s -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=false
control_path = %(directory)s/%%h-%%r
pipelining = True
timeout = 10
```

```
[persistent_connection]
connect_timeout = 30
connect_retries = 30
connect_interval = 1
```



### 警告

以下のパラメーターの値は `ansible.cfg` ファイルで重要な値になります。

- `remote_user` はユーザー `openshift` のままにする必要があります。
- インベントリパラメーターでは、2つのインベントリ間にスペースが入っていないことを確認します。

例: `inventory = path/to/inventory1,path/to/inventory2`

上記のコードブロックはファイルのデフォルト値を上書きする可能性があります。<keypair-name>に、デプロイメントインスタンスにコピーしたキーペアを設定するようにしてください。



### 注記

`inventory` フォルダは「[プロビジョニング用のインベントリの準備](#)」に作成されません。

## OpenShift 認証

OpenShift Container Platform は多数の異なる認証プラットフォームを使用する機能を提供します。認証オプションの一覧については、[認証およびユーザーエージェントの設定](#)を参照してください。

デフォルトのアイデンティティプロバイダーを設定することは、デフォルト値が Deny All に設定されているので重要になります。

## 22.3. OPENSIFT ANSIBLE PLAYBOOK を使用した OPENSIFT CONTAINER PLATFORM インスタンスのプロビジョニング

デプロイメントホストの作成および設定が完了したら、Ansible を使用して OpenShift Container Platform のデプロイメント用に環境を準備します。以下のサブセクションでは、OpenShift Container Platform を OpenStack に適切にデプロイできるように Ansible が設定され、特定の YAML ファイルが変更されます。

### 22.3.1. プロビジョニング用のインベントリの準備

これまでの手順で `openshift-ansible` パッケージのインストールを完了したら、`sample-inventory` ディレクトリが用意されます。これをデプロイメントホストの `cloud-user` ホームディレクトリにコピーします。

デプロイメントホストで、以下を実行します。

```
$ cp -r /usr/share/ansible/openshift-ansible/playbooks/openstack/sample-inventory/ ~/inventory
```

インベントリーディレクトリー内の **all.yml** ファイルには、RHOCIP インスタンスを正常にプロビジョニングするために設定する必要がある複数の異なるパラメーターすべてが含まれます。**OSEv3.yml** ファイルには、**all.yml** ファイルで必要な一部の参照と、カスタマイズできる利用可能なすべての OpenShift Container Platform クラスターパラメーターが含まれます。

### 22.3.1.1. OpenShiftSDN の All YAML ファイル

**all.yml** ファイルには、特定のニーズに合わせて変更できるオプションが多数あります。このファイルに収集される情報は、OpenShift Container Platform の正常なデプロイメントに必要なインスタンスのプロビジョニングの部分に対応します。これらを注意深く確認するようにしてください。本書では All YAML ファイルの縮小バージョンを扱っており、適切なデプロイメントを実行するために設定する必要のある最重要のパラメーターに重点を置いています。

```
$ cat ~/inventory/group_vars/all.yml
---
openshift_openstack_clusterid: "openshift"
openshift_openstack_public_dns_domain: "example.com"
openshift_openstack_dns_nameservers: ["10.19.115.228"]
openshift_openstack_public_hostname_suffix: "-public"
openshift_openstack_nsupdate_zone: "{{ openshift_openstack_public_dns_domain }}"

openshift_openstack_keypair_name: "openshift"
openshift_openstack_external_network_name: "public"

openshift_openstack_default_image_name: "rhel75"

## Optional (Recommended) - This removes the need for floating IPs
## on the OpenShift Cluster nodes
openshift_openstack_node_subnet_name: *<deployment-subnet-name>*
openshift_openstack_router_name: *<deployment-router-name>*
openshift_openstack_master_floating_ip: *false*
openshift_openstack_infra_floating_ip: *false*
openshift_openstack_compute_floating_ip: *false*
## End of Optional Floating IP section

openshift_openstack_num_masters: *3*
openshift_openstack_num_infra: *3*
openshift_openstack_num_cns: *0*
openshift_openstack_num_nodes: *2*

openshift_openstack_master_flavor: "m1.master"
openshift_openstack_default_flavor: "m1.node"

openshift_openstack_use_lbaas_load_balancer: *true*

openshift_openstack_docker_volume_size: "15"

# # Roll-your-own DNS
*openshift_openstack_external_nsupdate_keys:*
public:
  *key_secret: '/alb8h0EAFWvb4i+CMA12w=='*
  *key_name: "update-key"*
  *key_algorithm: 'hmac-md5'
```

```

*server: '<ip-of-DNS>'*
private:
  *key_secret: '/alb8h0EAFWvb4i+CMA12w=='*
  *key_name: "update-key"*
  *key_algorithm: 'hmac-md5'*
  *server: '<ip-of-DNS>'*

ansible_user: openshift

## cloud config
openshift_openstack_disable_root: true
openshift_openstack_user: openshift

```



### 注記

外部 DNS サーバーを使用することにより、プライベートおよびパブリックのセクションは DNS サーバーのパブリック IP アドレスを使用します。DNS サーバーが OpenStack 環境内に置かれていないためです。

上記のアスタリスク (\*) で囲まれる値は、OpenStack 環境および DNS サーバーに基づいて変更する必要があります。

All YAML ファイルの DNS の部分を適切に変更するには、DNS サーバーにログインし、以下のコマンドを実行してキー名、キーアルゴリズムおよびキーシークレットを取得します。

```

$ ssh <ip-of-DNS>
$ sudo -i
# cat /etc/named/<key-name.key>
key "update-key" {
  algorithm hmac-md5;
  secret "/alb8h0EAFWvb4i+CMA02w==" ;
};

```



### 注記

キー名は変わる可能性があります、上記はサンプルであることに注意してください。

#### 22.3.1.2. KuryrSDN All YAML ファイル

以下の `all.yml` ファイルは、デフォルトの OpenShiftSDN ではなく Kuryr SDN を有効にします。以下の例は凝縮されたバージョンであり、デフォルトテンプレートを注意して確認する必要があることに注意してください。

```

$ cat ~/inventory/group_vars/all.yml
---
openshift_openstack_clusterid: "openshift"
openshift_openstack_public_dns_domain: "*"example.com"*
openshift_openstack_dns_nameservers: *["10.19.115.228"]*
openshift_openstack_public_hostname_suffix: "-public"
openshift_openstack_nsupdate_zone: "{{ openshift_openstack_public_dns_domain }}"

openshift_openstack_keypair_name: "*"openshift"*
openshift_openstack_external_network_name: "*"public"*

```

```
openshift_openstack_default_image_name: "rhel75"

## Optional (Recommended) - This removes the need for floating IPs
## on the OpenShift Cluster nodes
openshift_openstack_node_subnet_name: *<deployment-subnet-name>*
openshift_openstack_router_name: *<deployment-router-name>*
openshift_openstack_master_floating_ip: *false*
openshift_openstack_infra_floating_ip: *false*
openshift_openstack_compute_floating_ip: *false*
## End of Optional Floating IP section

openshift_openstack_num_masters: *3*
openshift_openstack_num_infra: *3*
openshift_openstack_num_cns: *0*
openshift_openstack_num_nodes: *2*

openshift_openstack_master_flavor: "m1.master"*
openshift_openstack_default_flavor: "m1.node"*

## Kuryr configuration
openshift_use_kuryr: True
openshift_use_openshift_sdn: False
use_trunk_ports: True
os_sdn_network_plugin_name: cni
openshift_node_proxy_mode: userspace
kuryr_openstack_pool_driver: nested
openshift_kuryr_precreate_subports: 5

kuryr_openstack_public_net_id: *<public_ID>*

# To disable namespace isolation, comment out the next 2 lines
openshift_kuryr_subnet_driver: namespace
openshift_kuryr_sg_driver: namespace
# If you enable namespace isolation, `default` and `openshift-monitoring` become the
# global namespaces. Global namespaces can access all namespaces. All
# namespaces can access global namespaces.
# To make other namespaces global, include them here:
kuryr_openstack_global_namespaces: default,openshift-monitoring

# If OpenStack cloud endpoints are accessible over HTTPS, provide the CA certificate
kuryr_openstack_ca: *<path-to-ca-certificate>*

openshift_master_open_ports:
- service: dns tcp
  port: 53/tcp
- service: dns udp
  port: 53/udp
openshift_node_open_ports:
- service: dns tcp
  port: 53/tcp
- service: dns udp
  port: 53/udp

# To set the pod network CIDR range, uncomment the following property and set its value:
#
# openshift_openstack_kuryr_pod_subnet_prefixlen: 24
```

```

#
# The subnet prefix length value must be smaller than the CIDR value that is
# set in the inventory file as openshift_openstack_kuryr_pod_subnet_cidr.
# By default, this value is /24.

# openshift_portal_net is the range that OpenShift services and their associated Octavia
# load balancer VIPs use. Amphora VMs use Neutron ports in the range that is defined by
# openshift_openstack_kuryr_service_pool_start and openshift_openstack_kuryr_service_pool_end.
#
# The value of openshift_portal_net in the OSEv3.yml file must be within the range that is
# defined by openshift_openstack_kuryr_service_subnet_cidr. This range must be half
# of openshift_openstack_kuryr_service_subnet_cidr's range. This practice ensures that
# openshift_portal_net does not overlap with the range that load balancers' VMs use, which is
# defined by openshift_openstack_kuryr_service_pool_start and
# openshift_openstack_kuryr_service_pool_end.
#
# For reference only, copy the value in the next line from OSEv3.yml:
# openshift_portal_net: "*"172.30.0.0/16"*

openshift_openstack_kuryr_service_subnet_cidr: "*"172.30.0.0/15"*
openshift_openstack_kuryr_service_pool_start: "*"172.31.0.1"*
openshift_openstack_kuryr_service_pool_end: "*"172.31.255.253"*

# End of Kuryr configuration

openshift_openstack_use_lbaas_load_balancer: *true*

openshift_openstack_docker_volume_size: "15"

## Roll-your-own DNS
*openshift_openstack_external_nsupdate_keys:*
  public:
    *key_secret: '/alb8h0EAFWvb4i+CMA12w=='*
    *key_name: "update-key"*
    *key_algorithm: 'hmac-md5'*
    *server: '<ip-of-DNS>'*
  private:
    *key_secret: '/alb8h0EAFWvb4i+CMA12w=='*
    *key_name: "update-key"*
    *key_algorithm: 'hmac-md5'*
    *server: '<ip-of-DNS>'*

ansible_user: openshift

## cloud config
openshift_openstack_disable_root: true
openshift_openstack_user: openshift

```



## 注記

namespace の分離を使用している場合に、Kuryr-controller は各 namespace 用に新規の Neutron ネットワークおよびサブネットを作成します。

**注記**

Kuryr SDN が有効にされている場合に、ネットワークポリシーおよびノードポートサービスはサポートされません。

**注記**

Kuryr が有効にされている場合、OpenShift Container Platform サービスは OpenStack Octavia Amphora 仮想マシンで実装されます。

Octavia は UDP 負荷分散をサポートしません。UDP ポートを公開するサービスはサポートされません。

**22.3.1.2.1. グローバル namespace アクセスの設定**

**kuryr\_openstack\_global\_namespace** パラメーターには、グローバル namespace を定義する一覧が含まれます。デフォルトで、**default** および **openshift-monitoring** namespace のみがこの一覧に含まれます。

OpenShift Container Platform 3.11 以前の z-release からアップグレードする場合は、グローバル namespace から他の namespace へのアクセスは、セキュリティグループ **\*-allow\_from\_default** で制御されることに注意してください。

**remote\_group\_id** ルールは、グローバル namespace から他の namespace へのアクセスを制御できませんが、これを使用すると、スケーリングおよび接続の問題が生じる可能性があります。これらの問題を回避するには、**\*\_allow\_from\_default** で **remote\_group\_id** を使用して **remote\_ip\_prefix** に切り替えます。

1. コマンドラインから、ネットワークの **subnetCIDR** 値を取得します。

```
$ oc get kuryrnets ns-default -o yaml | grep subnetCIDR
subnetCIDR: 10.11.13.0/24
```

2. この範囲の TCP および UDP ルールを作成します。

```
$ openstack security group rule create --remote-ip 10.11.13.0/24 --protocol tcp openshift-ansible-openshift.example.com-allow_from_default
$ openstack security group rule create --remote-ip 10.11.13.0/24 --protocol udp openshift-ansible-openshift.example.com-allow_from_default
```

3. **remote\_group\_id** を使用するセキュリティグループルールを削除します。

```
$ openstack security group show *-allow_from_default | grep remote_group_id
$ openstack security group rule delete REMOTE_GROUP_ID
```

表22.4 All YAML ファイルの変数の説明

変数	説明
openshift_openstack_clusterid	クラスターの固有名
openshift_openstack_public_dns_domain	パブリック DNS ドメイン名

変数	説明
openshift_openstack_dns_nameservers	DNS ネームサーバーの IP
openshift_openstack_public_hostname_suffix	パブリックおよびプライベートの両方について DNS レコードのノードホスト名に接尾辞を追加します。
openshift_openstack_nsupdate_zone	OCP インスタンス IP で更新されるゾーン
openshift_openstack_keypair_name	OCP インスタンスにログインするために使用される キーペア名
openshift_openstack_external_network_name	OpenStack パブリックネットワーク名
openshift_openstack_default_image_name	OCP インスタンスに使用される OpenStack イメージ
openshift_openstack_num_masters	デプロイするマスターノードの数
openshift_openstack_num_infra	デプロイするインフラストラクチャーノードの数
openshift_openstack_num_cns	デプロイするコンテナーネイティブストレージノードの数
openshift_openstack_num_nodes	デプロイするアプリケーションノードの数
openshift_openstack_master_flavor	マスターインスタンスに使用される OpenStack フレーバーの名前
openshift_openstack_default_flavor	特定のフレーバーが指定されていない場合に、すべてのインスタンスに使用される Openstack フレーバーの名前
openshift_openstack_use_lbaas_load_balancer	Octavia ロードバランサーを有効にするブール値 (Octavia はインストールされる必要があります)
openshift_openstack_docker_volume_size	Docker ボリュームの最小サイズ (必要な変数)
openshift_openstack_external_nsupdate_keys	DNS のインスタンス IP アドレスでの更新
ansible_user	OpenShift Container Platform をデプロイするために使用される Ansible ユーザー。"openshift" は必須の名前であり、変更することはできません。
openshift_openstack_disable_root	ルートアクセスを無効にするブール値
openshift_openstack_user	このユーザーで使用される OCP インスタンス

変数	説明
openshift_openstack_node_subnet_name	デプロイメントに使用する既存 OpenShift サブネットの名前。これはデプロイメントホストに使用されるものと同じサブネット名である必要があります。
openshift_openstack_router_name	デプロイメントに使用する既存 OpenShift ルーターの名前。これはデプロイメントホストに使用されるものと同じルーター名である必要があります。
openshift_openstack_master_floating_ip	デフォルトは <b>true</b> です。マスターノードに割り当てられた Floating IP が不要な場合は <b>false</b> に設定される必要があります。
openshift_openstack_infra_floating_ip	デフォルトは <b>true</b> です。インフラストラクチャーノードに割り当てられた Floating IP が不要な場合は <b>false</b> に設定される必要があります。
openshift_openstack_compute_floating_ip	デフォルトは <b>true</b> です。コンピュートノードに割り当てられた Floating IP が不要な場合は <b>false</b> に設定される必要があります。
openshift_use_openshift_sdn	openshift-sdn を無効にする必要がある場合には、 <b>false</b> に設定される必要があります。
openshift_use_kuryr	kuryr sdn を有効にする必要がある場合には、 <b>true</b> に設定される必要があります。
use_trunk_ports	(kuryr で必要な) トランクポートで OpenStack 仮想マシンを作成するには、 <b>true</b> に設定される必要があります。
os_sdn_network_plugin_name	SDN 動作の選択。kuryr について <b>cni</b> に設定される必要があります。
openshift_node_proxy_mode	kuryr について <b>userspace</b> に設定される必要があります。
openshift_master_open_ports	kuryr を使用する場合に仮想マシンで開かれるポート
kuryr_openstack_public_net_id	kuryr で必要です。FIP が取得されるパブリック OpenStack ネットワークの ID です。
openshift_kuryr_subnet_driver	kuryr サブネットドライバー。namespace ごとにサブネットを作成するには <b>namespace</b> にする必要があります。

変数	説明
openshift_kuryr_sg_driver	kuryr セキュリティーグループドライバー。 namespace を分離するには <b>namespace</b> する必要があります。
kuryr_openstack_global_namespaces	namespace の分離に使用するグローバル namespace。デフォルト値は <b>default</b> 、 <b>openshift-monitoring</b> です。
kuryr_openstack_ca	クラウドの CA 証明書へのパスです。OpenStack クラウドエンドポイントが HTTPS 経由でアクセス可能な場合に必須です。

### 22.3.1.3. OSEv3 YAML ファイル

OSEv3 YAML ファイルは、OpenShift のインストールに関連するすべての異なるパラメーターおよびカスタマイズを指定します。

以下は、正常なデプロイメントに必要なすべての変数を含むファイルの縮小バージョンです。特定の OpenShift Container Platform デプロイメントに必要なカスタマイズの内容によって、追加の変数が必要になる場合があります。

```
$ cat ~/inventory/group_vars/OSEv3.yml
---

openshift_deployment_type: openshift-enterprise
openshift_release: v3.11
oreg_url: registry.access.redhat.com/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams: true
oreg_auth_user: <oreg_auth_user>
oreg_auth_password: <oreg_auth_pw>
# The following is required if you want to deploy the Operator Lifecycle Manager (OLM)
openshift_additional_registry_credentials:
[{'host':'registry.connect.redhat.com','user':'REGISTRYCONNECTUSER','password':'REGISTRYCONNECTPASSWORD','test_image':'mongodb/enterprise-operator:0.3.2'}]

openshift_master_default_subdomain: "apps.{{ (openshift_openstack_clusterid|trim == "") |
ternary(openshift_openstack_public_dns_domain, openshift_openstack_clusterid + '.' +
openshift_openstack_public_dns_domain) }}"

openshift_master_cluster_public_hostname: "console.{{ (openshift_openstack_clusterid|trim == "") |
ternary(openshift_openstack_public_dns_domain, openshift_openstack_clusterid + '.' +
openshift_openstack_public_dns_domain) }}"

#OpenStack Credentials:
openshift_cloudprovider_kind: openstack
openshift_cloudprovider_openstack_auth_url: "{{ lookup('env','OS_AUTH_URL') }}"
openshift_cloudprovider_openstack_username: "{{ lookup('env','OS_USERNAME') }}"
openshift_cloudprovider_openstack_password: "{{ lookup('env','OS_PASSWORD') }}"
openshift_cloudprovider_openstack_tenant_name: "{{ lookup('env','OS_PROJECT_NAME') }}"
openshift_cloudprovider_openstack_blockstorage_version: v2
openshift_cloudprovider_openstack_domain_name: "{{ lookup('env','OS_USER_DOMAIN_NAME') }}"
```

```
openshift_cloudprovider_openstack_conf_file: <path_to_local_openstack_configuration_file>

#Use Cinder volume for Openshift registry:
openshift_hosted_registry_storage_kind: openstack
openshift_hosted_registry_storage_access_modes: ["ReadWriteOnce"]
openshift_hosted_registry_storage_openstack_filesystem: xfs
openshift_hosted_registry_storage_volume_size: 30Gi

openshift_hosted_registry_storage_openstack_volumeID: d65209f0-9061-4cd8-8827-ae6e2253a18d
openshift_hostname_check: false
ansible_become: true

#Setting SDN (defaults to ovs-networkpolicy) not part of OSEv3.yml
#For more info, on which to choose, visit:
#https://docs.openshift.com/container-platform/3.11/architecture/networking/sdn.html#overview
networkPluginName: redhat/ovs-networkpolicy
#networkPluginName: redhat/ovs-multitenant

#Configuring identity providers with Ansible
#For initial cluster installations, the Deny All identity provider is configured
#by default. It is recommended to be configured with either htpasswd
#authentication, LDAP authentication, or Allowing all authentication (not recommended)
#For more info, visit:
#https://docs.openshift.com/container-
platform/3.10/install_config/configuring_authentication.html#identity-providers-ansible
#Example of Allowing All
#openshift_master_identity_providers: [{ 'name': 'allow_all', 'login': 'true', 'challenge': 'true', 'kind':
'AllowAllPasswordIdentityProvider' }]

#Optional Metrics (uncomment below lines for installation)

#openshift_metrics_install_metrics: true
#openshift_metrics_cassandra_storage_type: dynamic
#openshift_metrics_storage_volume_size: 25Gi
#openshift_metrics_cassandra_nodeselector: {"node-role.kubernetes.io/infra":"true"}
#openshift_metrics_hawkular_nodeselector: {"node-role.kubernetes.io/infra":"true"}
#openshift_metrics_heapster_nodeselector: {"node-role.kubernetes.io/infra":"true"}

#Optional Aggregated Logging (uncomment below lines for installation)

#openshift_logging_install_logging: true
#openshift_logging_es_pvc_dynamic: true
#openshift_logging_es_pvc_size: 30Gi
#openshift_logging_es_cluster_size: 3
#openshift_logging_es_number_of_replicas: 1
#openshift_logging_es_nodeselector: {"node-role.kubernetes.io/infra":"true"}
#openshift_logging_kibana_nodeselector: {"node-role.kubernetes.io/infra":"true"}
#openshift_logging_curator_nodeselector: {"node-role.kubernetes.io/infra":"true"}
```

一覧表示されている変数のいずれかについての詳細は、[OpenShift-Ansible ホストインベントリーのサンプル](#) を参照してください。

### 22.3.2. OpenStack 前提条件 Playbook

OpenShift Container Platform Ansible インストーラーは Playbook を提供し、OpenStack インスタンスのすべてのプロビジョニング手順が確実に実行されることを確認します。

Playbook の実行前に、RC ファイルを取得します。

```
$ source path/to/examplerc
```

デプロイメントホストで **ansible-playbook** コマンドを使用し、**prerequisites.yml** Playbook を使用してすべての前提条件を満たしていることを確認します。

```
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openstack/openshift-cluster/prerequisites.yml
```

前提条件 Playbook が正常に完了した後に、プロビジョニング Playbook を以下のように実行します。

```
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openstack/openshift-cluster/provision.yml
```

### 重要

**provision.yml** が早期にエラーを出す場合、OpenStack スタックのステータスを確認し、これが終了するのを待機します。

```
$ watch openstack stack list
+-----+-----+-----+-----+
+-----+
| ID                | Stack Name    | Stack Status  | Creation Time  |
Updated Time |
+-----+-----+-----+-----+
+-----+
| 87cb6d1c-8516-40fc-892b-49ad5cb87fac | openshift-cluster |
CREATE_IN_PROGRESS | 2018-08-20T23:44:46Z | None          |
+-----+-----+-----+-----+
+-----+
```

スタックが **CREATE\_IN\_PROGRESS** を表示する場合は、スタックが **CREATE\_COMPLETE** などの最終結果を出して完了するのを待機します。スタックが正常に完了しない場合は、それが追加で必要な手順を終了するように **provision.yml** Playbook を再実行します。

スタックが **CREATE\_FAILED** を表示する場合は、以下のコマンドを実行してエラーの原因を確認します。

```
$ openstack stack failures list openshift-cluster
```

### 22.3.3. スタック名の設定

デフォルトでは、OpenShift Container Platform クラスタ用に OpenStack が作成する Heat スタックは **openshift-cluster** という名前です。別の名前を使用する必要がある場合は、Playbook を実行する前に **OPENSHIFT\_CLUSTER** 環境変数を設定する必要があります。

```
$ export OPENSHIFT_CLUSTER=openshift.example.com
```

デフォルト以外のスタック名を使用し、`openshift-ansible` Playbook を実行してデプロイメントを更新する場合は、エラーを回避するために **OPENSIFT\_CLUSTER** をスタック名に設定する必要があります。

## 22.4. OPENSIFT CONTAINER PLATFORM インスタンスについての SUBSCRIPTION MANAGER の登録

ノードが正常にプロビジョニングされると、次の手順としてすべてのノードを **subscription-manager** で正常に登録し、正常な OpenShift Container Platform インストールに必要なすべてのパッケージをインストールする必要があります。これを簡単に実行できるように `repos.yml` ファイルが作成され、提供されています。

```
$ cat ~/repos.yml
---
- name: Enable the proper repositories for OpenShift installation
  hosts: OSEv3
  become: yes
  tasks:
    - name: Register with activationkey and consume subscriptions matching Red Hat Cloud Suite or
      Red Hat OpenShift Container Platform
      redhat_subscription:
        state: present
        activationkey: <key-name>
        org_id: <orig_id>
        pool: '^(Red Hat Cloud Suite|Red Hat OpenShift Container Platform)$'

- name: Disable all current repositories
  rhsm_repository:
    name: '*'
    state: disabled

- name: Enable Repositories
  rhsm_repository:
    name: "{{ item }}"
    state: enabled
  with_items:
    - rhel-7-server-rpms
    - rhel-7-server-extras-rpms
    - rhel-7-server-ansible-2.6-rpms
    - rhel-7-server-ose-3.11-rpms
```



### 注記

[リポジトリの設定](#) を参照し、有効にする適切なリポジトリおよびバージョンを確認します。上記のファイルはサンプルであることに注意してください。

`repos.yml` を使用して **ansible-playbook** コマンドを実行します。

```
$ ansible-playbook repos.yml
```

上記の例では、すべての登録に Ansible の **redhat\_subscription** および **rhsm\_repository** モジュールを使用し、リポジトリの有効化および無効化を行います。この特定の例では、Red Hat アクティベーションキーを利用しています。アクティベーションキーがない場合は、Ansible の

**redhat\_subscription** モジュールにアクセスして、例に示されるようにユーザー名およびパスワードを使用して変更を実行してください

([https://docs.ansible.com/ansible/2.6/modules/redhat\\_subscription\\_module.html](https://docs.ansible.com/ansible/2.6/modules/redhat_subscription_module.html))。



### 注記

**redhat\_subscription** モジュールは特定ノードで失敗することが時折あります。この問題が生じる場合は、**subscription-manager** を使用して OpenShift Container Platform インスタンスを手動で登録してください。

## 22.5. ANSIBLE PLAYBOOK を使用した OPENSIFT CONTAINER PLATFORM のインストール

OpenStack インスタンスがプロビジョニングされると、OpenShift Container Platform のインストールに焦点が切り替わります。インストールおよび設定は、OpenShift RPM パッケージで提供される一連の Ansible Playbook およびロールで実行されます。事前に設定された OSEv3.yml ファイルで、すべてのオプションが適切に設定されていることを確認してください。

インストーラー Playbook を実行する前に、以下を実行してすべての {rhocp} 前提条件を満たしていることを確認します。

```
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml
```

インストーラー Playbook を実行して Red Hat OpenShift Container Platform をインストールします。

```
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openshift-cluster/install.yml
```



### 注記

OpenShift Container Platform バージョン 3.11 は RH OSP 14 および RH OSP 13 でサポートされます。OpenShift Container Platform バージョン 3.10 は RH OSP 13 でサポートされます。

## 22.6. 設定変更を既存の OPENSIFT CONTAINER PLATFORM 環境に適用する

マスターおよびノードのすべてのホストで OpenShift Container Platform サービスを起動または再起動し、設定の変更を適用します。 [OpenShift Container Platform サービスの再起動](#) を参照してください。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```



## 注記

Kubernetes アーキテクチャーでは、クラウドプロバイダーからの信頼性のあるエンドポイントが必要です。クラウドプロバイダーが停止している場合、kubelet は OpenShift Container Platform が再起動されないようにします。基礎となるクラウドプロバイダーのエンドポイントに信頼性がない場合は、クラウドプロバイダー統合を使用してクラスターをインストールしないでください。クラスターをベアメタル環境の場合のようにインストールします。インストール済みのクラスターで、クラウドプロバイダー統合をオンまたはオフに切り替えることは推奨されていません。ただし、そのシナリオが避けられない場合は、以下のプロセスを実行してください。

クラウドプロバイダーを不使用から使用に切り替えるとエラーメッセージが表示されます。クラウドプロバイダーを追加すると、ノードが `hostname` を `externalID` として使用する (クラウドプロバイダーが使用されていなかった場合のケース) 代わりに、クラウドプロバイダーの `instance-id` (クラウドプロバイダーによって指定される) の使用に切り替えるため、ノードの削除が試みられます。この問題を解決するには、以下を実行します。

1. CLI にクラスター管理者としてログインします。
2. 既存のノードラベルをチェックし、これらをバックアップします。

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?:=Taints)'
```

3. ノードを削除します。

```
$ oc delete node <node_name>
```

4. 各ノードホストで OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

5. 以前に使用していた [各ノードのラベル](#) を再度追加します。

### 22.6.1. 既存の OpenShift 環境での OpenStack 変数の設定

必要な OpenStack 変数を設定するには、OpenShift Container Platform のマスターとノード両方のすべてのホストにて、以下の内容で `/etc/origin/cloudprovider/openstack.conf` を変更します。

```
[Global]
auth-url = <OS_AUTH_URL>
username = <OS_USERNAME>
password = <password>
domain-id = <OS_USER_DOMAIN_ID>
tenant-id = <OS_TENANT_ID>
region = <OS_REGION_NAME>

[LoadBalancer]
subnet-id = <UUID of the load balancer subnet>
```

`OS_` 変数の値については OpenStack の管理者にお問い合わせください。この値は通常 OpenStack の設定で使用されます。

### 22.6.2. 動的に作成した OpenStack PV のゾーンラベルの設定

管理者は、動的に作成された OpenStack PV のゾーンラベルを設定できます。このオプションは、OpenStack Cinder ゾーン名がコンピュートゾーン名などに一致しない場合、Cinder ゾーンが1つだけで、コンピュートゾーンが多数ある場合に有用です。管理者は、動的に Cinder ボリュームを作成してから、ラベルをチェックできます。

PV のゾーンラベルを表示します。

```
# oc get pv --show-labels
NAME                                CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS
CLAIM                               STORAGECLASS  REASON  AGE  LABELS
pvc-1faa6f93-64ac-11e8-930c-fa163e3c373c  1Gi      RWO          Delete          Bound  openshift-
node/pvc1  standard          12s  failure-domain.beta.kubernetes.io/zone=nova
```

デフォルトの設定が有効になっています。**oc get pv --show-labels** コマンドは、**failure-domain.beta.kubernetes.io/zone=nova** ラベルを返します。

ゾーンラベルを無効にするには、以下を追加して **openstack.conf** を更新します。

```
[BlockStorage]
ignore-volume-az = yes
```

マスターサービスの再起動後に作成された PV にはゾーンラベルがありません。

## 第23章 GOOGLE COMPUTE ENGINE の設定

アプリケーションデータ用に [永続ストレージ](#)として GCE ボリュームを使用する など OpenShift Container Platform が既存の [Google Compute Engine \(GCE\) インフラストラクチャー](#) にアクセスするように設定します。

### 23.1. 作業を開始する前に

#### 23.1.1. Google Cloud Platform の認証の設定

##### ロール

OpenShift Container Platform に GCP を設定するには、以下の GCP ロールが必要です。

<b>roles/owner</b>	サービスアカウント、クラウドストレージ、インスタンス、イメージ、テンプレート、Cloud DNS エントリーの作成や、ロードバランサーとヘルスチェックのデプロイに必要です。
--------------------	--

ユーザーがテストフェーズ中に環境の再デプロイを想定している場合には、**delete** パーミッションが必要な場合もあります。

サービスアカウントを使用することで、GCP オブジェクトのデプロイ時に個人ユーザーの使用を回避することもできます。

ロールの設定方法に関する手順など、詳細は、[GCP ドキュメントのロールの理解のセクション](#) を参照してください。

##### スコープおよびサービスアカウント

GCP はスコープを使用して、承認済みのアイデンティティが認証され、リソース内で操作が実行できるかどうかを判断します。たとえば、読み取り専用のスコープのアクセストークンを持つアプリケーション A は、読み取りだけできますが、読み取り/書き込みスコープのアクセストークンを持つアプリケーション B はデータの読み取りと変更が可能です。

スコープは、GCP API レベルで <https://www.googleapis.com/auth/compute.readonly> として定義されます。

インスタンスの作成時に `--scopes=[SCOPE,...]` オプションを使用してスコープを指定するか、インスタンスが GCP API にアクセスしないようにする場合には、`--no-scopes` オプションを使用してスコープなしでインスタンスを作成できます。

詳細情報は、[GCP ドキュメントのスコープのセクション](#) を参照してください。

GCP の全プロジェクトには、プロジェクトエディターパーミッションの付いた `[PROJECT_NUMBER]-compute@developer.gserviceaccount.com` サービスアカウントが含まれています。

デフォルトでは、新規作成されたインスタンスは自動的に有効化されて以下のアクセススコープが割り当てられたデフォルトのサービスアカウントとして実行されます。

- [https://www.googleapis.com/auth/devstorage.read\\_only](https://www.googleapis.com/auth/devstorage.read_only)
- <https://www.googleapis.com/auth/logging.write>

- <https://www.googleapis.com/auth/monitoring.write>
- <https://www.googleapis.com/auth/pubsub>
- <https://www.googleapis.com/auth/service.management.readonly>
- <https://www.googleapis.com/auth/servicecontrol>
- <https://www.googleapis.com/auth/trace.append>
- <https://www.googleapis.com/auth/bigquery>
- <https://www.googleapis.com/auth/cloud-platform>
- <https://www.googleapis.com/auth/compute.readonly>
- <https://www.googleapis.com/auth/compute>
- <https://www.googleapis.com/auth/datastore>
- <https://www.googleapis.com/auth/logging.write>
- <https://www.googleapis.com/auth/monitoring>
- <https://www.googleapis.com/auth/monitoring.write>
- <https://www.googleapis.com/auth/servicecontrol>
- <https://www.googleapis.com/auth/service.management.readonly>
- <https://www.googleapis.com/auth/sqlservice.admin>
- [https://www.googleapis.com/auth/devstorage.full\\_control](https://www.googleapis.com/auth/devstorage.full_control)
- [https://www.googleapis.com/auth/devstorage.read\\_only](https://www.googleapis.com/auth/devstorage.read_only)
- [https://www.googleapis.com/auth/devstorage.read\\_write](https://www.googleapis.com/auth/devstorage.read_write)
- <https://www.googleapis.com/auth/taskqueue>
- <https://www.googleapis.com/auth/userinfo.email>

インスタンスの作成時に、**--service-account=SERVICE\_ACCOUNT** オプションで別のサービスアカウントを指定するか、**gcloud** CLI で **--no-service-account** オプションを使用してインスタンスのサービスアカウントを明示的に無効化します。

詳細情報は、[GCP ドキュメントの新規サービスアカウントの作成セクション](#) を参照してください。

### 23.1.2. Google Compute Engine オブジェクト

OpenShift Container Platform と Google Compute Engine (GCE) を統合するには、以下のコンポーネントまたはサービスが必要です。

#### GCP プロジェクト

GCP プロジェクトは、全 GCP サービスの作成、有効化、使用の基盤を形成するベースレベルの組織エンティティです。これには、API の管理、課金の有効化、コラボレーターの追加/削除、パーミッションの管理が含まれます。

詳細情報は、[GCP ドキュメントのプロジェクトリソースセクション](#) を参照してください。



### 重要

プロジェクト ID は一意識別子で、Google Cloud Engine すべてで一意的でなければなりません。つまり、**myproject** という名前のプロジェクトがすでに作成されている場合には、このプロジェクト ID を使用できません。

### 請求書

アカウントに課金がアタッチされていない限り、新規リソースを作成できません。新規プロジェクトは、既存のプロジェクトにリンクすることも、新規情報を入力することもできます。

詳細情報は、[GCP ドキュメントの請求アカウントの作成、変更、終了](#) を参照してください。

### クラウドのアイデンティティおよびアクセス管理

OpenShift Container Platform のデプロイには、適切なパーミッションが必要です。ユーザーは、サービスアカウント、クラウドストレージ、インスタンス、イメージ、テンプレート、Cloud DNS エントリーの作成、ロードバランサーやヘルスチェックのデプロイができる必要があります。テスト中に環境を再デプロイできるようにするには、削除のパーミッションが役立ちます。

特定のパーミッションを割り当ててサービスアカウントを作成し、このアカウントを使用して、通常のユーザーではなくインフラストラクチャーコンポーネントをデプロイします。また、異なるユーザーまたはサービスアカウントへのアクセスを制限するためのロールを作成することも可能です。

GCP インスタンスは、サービスアカウントを使用して、アプリケーションが GCP API を呼び出せるようにします。たとえば、OpenShift Container Platform ノードホストは、GCP ディスク API を呼び出して、アプリケーションに永続ボリュームを提供することができます。

IAM サービスを使用することで、さまざまなインフラストラクチャーやサービスリソースへのアクセス制御、粒度の細かいロールを利用できます。詳細情報は、[GCP ドキュメントのクラウドの概要へのアクセスのセクション](#) を参照してください。

### SSH キー

GCP は、作成したインスタンスで SSH を使用してログインできるように、SSH 公開キーを認証キーとして注入します。インスタンス別またはプロジェクト別に SSH キーを設定できます。

既存の SSH キーを使用できます。GCP メタデータは、ブート時にインスタンスに注入して SSH アクセスを可能にする、SSH キーの保存に役立ちます。

詳細情報は、[GCP ドキュメントのメタデータセクション](#) を参照してください。

### GCP リージョンおよびゾーン

GCP には、リージョンとアベイラビリティゾーンに対応するグローバルインフラストラクチャーがあります。GCP にある OpenShift Container Platform を異なるゾーンにデプロイすると、単一障害点を回避できますが、ストレージに関して注意点があります。

GCP ディスクがゾーン内に作成されます。そのため、OpenShift Container Platform ノードのホストがゾーン A でダウンし、Pod がゾーン B に移動した場合に、ディスクが異なるゾーンに配置されているので、永続ストレージはこれらの Pod にアタッチできません。

OpenShift Container Platform をインストールする前に、複数のゾーンからなる OpenShift Container Platform 環境のゾーンの 1 つをデプロイするかどうか判断するのは重要です。複数ゾーン環境をデプロイする場合には、単一のリージョンに 3 つの異なるゾーンを使用する設定が推奨されます。

詳細情報は、[GCP ドキュメントのリージョンとゾーン](#) および [Kubernetes ドキュメントの複数ゾーン](#) を参照してください。

## 外部 IP アドレス

GCP インスタンスがインターネットと通信できるように、インスタンスに外部 IP アドレスをアタッチする必要があります。また、外部 IP アドレスは、Virtual Private Cloud (VPC) ネットワーク外から、GCP にデプロイされたインスタンスと通信するのに必要です。



### 警告

インターネットアクセスに **外部 IP アドレス** が必要になるので、プロバイダーには制限となります。受信トラフィックが必要ない場合には、ファイアウォールルールを設定して、インスタンスで受信する外部トラフィックをブロックすることができます。

詳細情報は、[GCP ドキュメントの外部 IP アドレス](#) を参照してください。

## クラウド DNS

GCP クラウド DNS は、GCP DNS サーバーを使用してドメイン名をグローバル DNS に公開するために使用する DNS サービスです。

パブリッククラウドの DNS ゾーンには、Google のドメインサービスまたはサードパーティーのプロバイダーを使用して購入したドメイン名を使用する必要があります。ゾーンを作成する時に、[Google が提供するネームサーバーをレジストラに追加](#) する必要があります。

詳細情報は、[GCP ドキュメントの Cloud DNS](#) セクションを参照してください。



### 注記

GCP VPC ネットワークには、内部ホスト名を自動的に解決する内部の DNS サービスがあります。

インスタンスに対する内部の完全修飾ドメイン名 (FQDN) は、**[HOST\_NAME].c.[PROJECT\_ID].internal** 形式に従います。

詳細情報は、[GCP ドキュメントの内部 DNS](#) を参照してください。

## 負荷分散

GCP 負荷分散サービスにより、GCP クラウド内の複数のインスタンスに、トラフィックを分散することができます。

負荷分散には 5 つのタイプがあります。

- [内部](#)
- [ネットワーク負荷分散](#)
- [HTTP\(S\) 負荷分散](#)
- [SSL Proxy 負荷分散](#)

- [TCP Proxy 負荷分散](#)



### 注記

HTTPS および TCP Proxy 負荷分散は、マスターノードに HTTPS ヘルスチェックを使用する唯一の方法で、`/healthz` のステータスを確認します。

HTTPS 負荷分散には、カスタムの証明書が必要なため、この実装は、TCP Proxy 負荷分散を使用して、このプロセスを簡素化します。

詳細情報は、[GCP ドキュメントの負荷分散](#) を参照してください。

### インスタンスサイズ

正常な OpenShift Container Platform の環境には、最低でも以下のハードウェア要件を満たす必要があります。

表23.1 インスタンスサイズ

ロール	Size
マスター	<b>n1-standard-8</b>
ノード	<b>n1-standard-4</b>

GCP では、カスタムのインスタンスサイズを作成して、異なる要件に適合します。詳細は、[カスタムのマシンタイプでのインスタンス作成](#) を参照するか、インスタンスサイズに関する詳細は、[マシンタイプ](#) および [OpenShift Container Platform のハードウェア最小要件](#) を参照してください。

### ストレージオプション

デフォルトでは、GCP インスタンスごとに、オペレーティングシステムを含む小規模な Root 永続ディスクが含まれます。インスタンスで実行するアプリケーションで、より多くのストレージ容量が必要な場合に、このインスタンスにさらにストレージオプションを追加できます

- 標準の永続ディスク
- SSD 永続ディスク
- ローカル SSD
- クラウドストレージバケット

詳細情報は、[GCP ドキュメントのストレージオプション](#) を参照してください。

## 23.2. OPENSIFT CONTAINER PLATFORM での GCE の設定

OpenShift Container Platform は、GCE 用に 2 種類の方法で設定できます。

- [Ansible の使用](#)
- [master-config.yaml](#) ファイルを変更して手動で設定する

### 23.2.1. オプション 1: Ansible を使用した OpenShift Container Platform での GCP の設定

OpenShift Container Platform での Google Compute Platform (GCP) の設定は、インストール時またはインストール後に、[Ansible インベントリーファイル](#) を変更することで実行できます。

#### 手順

1. 最低でも、`openshift_cloudprovider_kind`、`openshift_gcp_project` と `openshift_gcp_prefix` のパラメーター、マルチゾーンのデプロイメントにはオプションで `openshift_gcp_multizone`、デフォルトのネットワーク名を使用しない場合は `openshift_gcp_network_name` を定義する必要があります。  
インストール時に Ansible インベントリーファイルに以下のセクションを追加して、OpenShift Container Platform 環境で GCP を設定します。

```
[OSEv3:vars]
openshift_cloudprovider_kind=gce
openshift_gcp_project=<projectid> 1
openshift_gcp_prefix=<uid> 2
openshift_gcp_multizone=False 3
openshift_gcp_network_name=<network name> 4
```

- 1 既存のインスタンスを実行している GCP プロジェクト ID を指定します。この ID は、Google Cloud Platform Console でプロジェクトを作成すると生成されます。
- 2 一意の文字列を指定して、各 OpenShift Container Platform クラスタを特定します。これは、GCP 全体で一貫でなければなりません。
- 3 オプションで **True** の設定して、GCP でのマルチゾーンのデプロイメントをトリガーします。デフォルトでは **False** に設定されます。
- 4 オプションで、**default** ネットワークを使用しない場合に、ネットワーク名を指定します。

Ansible でインストールすると、GCP 環境に適合されるように、以下のファイルが作成されて設定されます。

- `/etc/origin/cloudprovider/gce.conf`
  - `/etc/origin/master/master-config.yaml`
  - `/etc/origin/node/node-config.yaml`
2. [GCP を使用してロードバランサーサービスを実行](#) する場合は、Compute Engine VM ノードインスタンスには `ocp` の接尾辞が必要です。たとえば、`openshift_gcp_prefix` パラメーターが `mycluster` に設定されている場合には、ノードに `myclusterocp` のタグを付ける必要があります。Compute Engine VM インスタンスにネットワークタグを追加する方法については、[ネットワークタグの追加と削除](#) を参照してください。
  3. オプションで、マルチゾーンサポートを設定できます。  
クラスタのインストールプロセスでは、単一ゾーンのサポートがデフォルトで設定されますが、単一障害点を避けるためにマルチゾーンを設定することができます。

GCP ディスクがゾーン内に作成されるので、異なるゾーンで GCP に OpenShift Container Platform をデプロイすると、ストレージで問題が発生する可能性があります。OpenShift

Container Platform ノードのホストがゾーン A でダウンし、Pod がゾーン B に移動した場合に、ディスクが異なるゾーンに配置されているので、永続ストレージはこれらの Pod にアタッチできません。詳細情報は、Kubernetes ドキュメントの [マルチゾーンの制限](#) を参照してください。

Ansible インベントリを使用してマルチゾーンサポートを有効にするには、以下のパラメーターを追加します。

```
[OSEv3:vars]
openshift_gcp_multizone=true
```

シングルゾーンサポートに戻すには、**openshift\_gcp\_multizone** の値を **false** に設定して、Ansible インベントリファイルに戻ります。

## 23.2.2. オプション 2: OpenShift Container Platform での GCE の手動設定

### 23.2.2.1. GCE 向けのマスターホストの手動設定

全マスターホストで以下の手順を実行します。

#### 手順

1. デフォルトでは、**/etc/origin/master/master-config.yaml** のマスター設定ファイルの **apiServerArguments** と **controllerArguments** に GCE パラメーターを追加します。

```
apiServerArguments:
  cloud-provider:
    - "gce"
  cloud-config:
    - "/etc/origin/cloudprovider/gce.conf"
controllerArguments:
  cloud-provider:
    - "gce"
  cloud-config:
    - "/etc/origin/cloudprovider/gce.conf"
```

2. Ansible を使用して GCP 用に OpenShift Container Platform を設定する場合には、**/etc/origin/cloudprovider/gce.conf** ファイルは自動的に作成されます。GCP 用の OpenShift Container Platform を手動で設定するので、このファイルを作成して、以下を入力する必要があります。

```
[Global]
project-id = <project-id> ①
network-name = <network-name> ②
node-tags = <node-tags> ③
node-instance-prefix = <instance-prefix> ④
multizone = true ⑤
```

- ① 既存のインスタンスを実行している GCP プロジェクト ID を指定します。
- ② デフォルトを使用しない場合には、ネットワーク名を指定します。
- ③ GCP ノードにタグを付けます。接尾辞に **ocp** を含める必要があります。たとえば、**node-instance-prefix** パラメーターの値を **mycluster** に設定する場合は、ノードは

**myclusterocp** のタグを付ける必要があります。

- 4 一意の文字列を指定して、OpenShift Container Platform クラスターを特定します。
- 5 **True** の設定して、GCP でのマルチゾーンのデプロイメントをトリガーします。デフォルトでは **False** に設定されます。

クラスターインストールでは、シングルゾーンのサポートがデフォルトで設定されます。

異なるゾーンで GCP に OpenShift Container Platform をデプロイすると、単一障害点を回避しやすくなりますが、ストレージで問題が発生する可能性があります。これは、GCP ディスクがゾーン内に作成されるためです。OpenShift Container Platform ノードのホストがゾーン A でダウンし、Pod がゾーン B に移動した場合に、ディスクが異なるゾーンに配置されているので、永続ストレージはこれらの Pod にアタッチできません。詳細情報は、Kubernetes ドキュメントの [マルチゾーンの制限](#) を参照してください。



### 重要

GCP を使用してロードバランサーサービスを実行する場合は、Compute Engine VM ノードインスタンスには **ocp** の接尾辞: **<openshift\_gcp\_prefix>ocp** が必要です。たとえば、**openshift\_gcp\_prefix** パラメーターの値を **mycluster** に設定する場合は、このノードに **myclusterocp** のタグを付ける必要があります。Compute Engine VM インスタンスにネットワークタグを追加する方法については、[ネットワークタグの追加と削除](#) を参照してください。

3. OpenShift Container Platform サービスを再起動します。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

シングルゾーンサポートに戻すには、**multizone** の値を **false** に設定して、マスターとノードホストサービスを再起動します。

#### 23.2.2.2. GCE 向けのノードホストの手動設定

全ノードホスト上で以下を実行します。

#### 手順

1. 適切な [ノード設定マップ](#) を編集して、**kubeletArguments** セクションの内容を更新します。

```
kubeletArguments:
  cloud-provider:
    - "gce"
  cloud-config:
    - "/etc/origin/cloudprovider/gce.conf"
```



### 重要

クラウドプロバイダーの統合を正常に機能させるため、**nodeName** は GCP のインスタンス名と一致していなければなりません。また、この名前は RFC1123 に準拠している必要もあります。

- すべてのノードで OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

### 23.2.3. GCP の OpenShift Container Platform レジストリーの設定

Google Cloud Platform (GCP) は、OpenShift Container Platform が OpenShift Container Platform コンテナイメージレジストリーを使用してコンテナイメージを保存するために、使用可能なオブジェクトクラウドストレージを提供します。

詳細情報は、[GCP ドキュメントのクラウドストレージ](#) を参照してください。

#### 前提条件

インストールする前に、バケットを作成して、レジストリーイメージをホストする必要があります。以下のコマンドでは、設定したサービスアカウントを使用してリージョンバケットを作成します。

```
gsutil mb -c regional -l <region> gs://ocp-registry-bucket
cat <<EOF > labels.json
{
  "ocp-cluster": "mycluster"
}
EOF
gsutil label set labels.json gs://ocp-registry-bucket
rm -f labels.json
```



#### 注記

デフォルトでは、バケットのデータは、Google が管理するキーを使用して自動的に暗号化されます。データの暗号化に別のキーを指定する場合には、GCP で利用可能な [データ暗号化オプション](#) を参照してください。

詳細情報は、[ストレージバケットの作成ドキュメント](#) を参照してください。

#### 手順

レジストリーが Google Cloud Storage (GCS) バケットを使用できるように、[Ansible インベントリファイル](#) を設定します。

```
[OSEv3:vars]
# GCP Provider Configuration
openshift_hosted_registry_storage_provider=gcs
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_replicas=1 ①
openshift_hosted_registry_storage_gcs_bucket=<bucket_name> ②
openshift_hosted_registry_storage_gcs_keyfile=<bucket_keyfile> ③
openshift_hosted_registry_storage_gcs_rootdirectory=<registry_directory> ④
```

- ① 設定するレプリカ数
- ② レジストリーストレージのバケット名
- ③ データの暗号化にカスタムのキーファイルを使用する場合にバケットのキーファイルが配置されているインストーラーホストのパス

- データの保存に使用するディレクトリー。デフォルトで **/registry** です。

詳細情報は、[GCP ドキュメントのクラウドストレージ](#) を参照してください。

### 23.2.3.1. GCP 向けの OpenShift Container Platform レジストリーの手動設定

GCP オブジェクトストレージを使用するには、レジストリーの設定ファイルを編集してレジストリー Pod にマウントします。

ストレージドライバーの設定ファイルに関する詳細情報は、[Google Cloud ストレージドライバーのドキュメント](#) を参照してください。

#### 手順

- 現在の `/etc/registry/config.yml` ファイルをエクスポートします。

```
$ oc get secret registry-config \
  -o jsonpath='{.data.config\.yml}' -n default | base64 -d \
  >> config.yml.old
```

- 以前の `/etc/registry/config.yml` ファイルから新規の設定ファイルを作成します。

```
$ cp config.yml.old config.yml
```

- このファイルを編集して GCP パラメーターを追加します。レジストリーの設定ファイルの **storage** セクションに、バケットと **keyfile** を指定します。

```
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  gcs:
    bucket: ocp-registry ①
    keyfile: mykeyfile ②
```

- GCP バケット名に置き換えます。

- JSON 形式のサービスアカウントの秘密鍵ファイル。Google アプリケーションのデフォルトの認証情報を使用する場合は、**keyfile** パラメーターは指定しないでください。

- registry-config** シークレットを削除します。

```
$ oc delete secret registry-config -n default
```

- シークレットを再作成して、更新された設定ファイルを参照します。

```
$ oc create secret generic registry-config \
  --from-file=config.yml -n default
```

- 更新された設定を読み取るためにレジストリーを再デプロイします。

```
$ oc rollout latest docker-registry -n default
```

### 23.2.3.1.1. レジストリーが GCP オブジェクトストレージを使用していることを確認します。

レジストリーが GCP バケットストレージを使用しているかどうかを確認します。

#### 手順

1. GCP ストレージを使用して正常にレジストリーをデプロイした後に、GCP バケットストレージではなく **emptydir** が使用される場合には、**deploymentconfig** のレジストリーは、何も情報を表示しません。

```
$ oc describe dc docker-registry -n default
...
Mounts:
...
  /registry from registry-storage (rw)
Volumes:
registry-storage:
Type:     EmptyDir 1
...
```

- 1** Pod の寿命を共有する一時ディレクトリー

2. **/registry** のマウントポイントが空かどうかを確認します。これは、GCP ストレージが使用するボリュームです。

```
$ oc exec \
$(oc get pod -l deploymentconfig=docker-registry \
-o=jsonpath='{.items[0].metadata.name}') -i -t -- ls -l /registry
total 0
```

3. 空の場合は、GCP バケット設定が **registry-config** シークレットで実行されているためです。

```
$ oc describe secret registry-config
Name:     registry-config
Namespace: default
Labels:   <none>
Annotations: <none>

Type: Opaque

Data
====
config.yml: 398 bytes
```

4. インストーラーは、[インストールドキュメントのストレージセクション](#) で記載されているように、拡張されたレジストリー機能を使用して、希望の設定で **config.yml** ファイルを作成します。以下のコマンドで、ストレージバケット設定が保存されている **storage** セクションを含めて設定ファイルを表示します。

```
$ oc exec \
$(oc get pod -l deploymentconfig=docker-registry \
```

```
-o=jsonpath='{.items[0].metadata.name}') \  
cat /etc/registry/config.yml
```

```
version: 0.1  
log:  
  level: debug  
http:  
  addr: :5000  
storage:  
  delete:  
    enabled: true  
  cache:  
    blobdescriptor: inmemory  
  gcs:  
    bucket: ocp-registry  
auth:  
  openshift:  
    realm: openshift  
middleware:  
  registry:  
  - name: openshift  
  repository:  
  - name: openshift  
    options:  
      pullthrough: True  
      acceptschema2: True  
      enforcequota: False  
storage:  
  - name: openshift
```

または、以下でシークレットを表示できます。

```
$ oc get secret registry-config -o jsonpath='{.data.config\.yaml}' | base64 -d  
version: 0.1  
log:  
  level: debug  
http:  
  addr: :5000  
storage:  
  delete:  
    enabled: true  
  cache:  
    blobdescriptor: inmemory  
  gcs:  
    bucket: ocp-registry  
auth:  
  openshift:  
    realm: openshift  
middleware:  
  registry:  
  - name: openshift  
  repository:  
  - name: openshift  
    options:  
      pullthrough: True  
      acceptschema2: True
```

```
enforcequota: False
storage:
- name: openshift
```

GCP コンソールで **Storage** を表示して **Browser** をクリックし、バケットを選択するか、**gsutil** コマンドを実行して、イメージのプッシュが正常に行われたことを確認します。

```
$ gsutil ls gs://ocp-registry/
gs://ocp-registry/docker/

$ gsutil du gs://ocp-registry/
7660385 gs://ocp-registry/docker/registry/v2/blobs/sha256/03/033565e6892e5cc6dd03187d00a4575720a928db111274e0fbf31b410a093c10/data
7660385 gs://ocp-registry/docker/registry/v2/blobs/sha256/03/033565e6892e5cc6dd03187d00a4575720a928db111274e0fbf31b410a093c10/
7660385 gs://ocp-registry/docker/registry/v2/blobs/sha256/03/
...
```

**emptyDir** ボリュームを使用する場合には、**/registry** マウントポイントは以下のようになります。

```
$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') -i -t -- df -h /registry
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc        30G  226M   30G   1% /registry

$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') -i -t -- ls -l /registry
total 0
drwxr-sr-x. 3 1000000000 1000000000 22 Jun 19 12:24 docker
```

### 23.2.4. OpenShift Container Platform が GCP ストレージを使用するように設定する

OpenShift Container Platform は、永続ボリュームメカニズムを活用して GCP ストレージを使用できます。OpenShift Container Platform は、GCP にディスクを作成して、正しいインスタンスにこのディスクをアタッチします。

GCP ディスクは **ReadWriteOnce** アクセスモードで、1つのノードで読み取り/書き込み可能な状態でボリュームをマウントできます。詳細情報は、[アーキテクチャーガイドのアクセスモードのセクション](#)を参照してください。

#### 手順

1. OpenShift Container Platform は、**gce-pd** プロビジョナーを使用しており、Ansible インベントリで **openshift\_cloudprovider\_kind=gce** および **openshift\_gcp\_\*** 変数を使用する場合には、以下の **storageclass** を作成します。それ以外の場合、Ansible を使用せずに OpenShift Container Platform を設定しており、**storageclass** はインストール時に作成されていない場合には、以下のように、手動で作成できます。

```
$ oc get --export storageclass standard -o yaml
```

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  creationTimestamp: null
  name: standard
  selfLink: /apis/storage.k8s.io/v1/storageclasses/standard
parameters:
  type: pd-standard
  provisioner: kubernetes.io/gce-pd
  reclaimPolicy: Delete

```

PV を要求して、以前の手順で示した storageclass を使用すると、OpenShift Container Platform は GCP インフラストラクチャーにディスクを作成します。ディスクが作成されたことを確認するには以下を実行します。

```

$ gcloud compute disks list | grep kubernetes
kubernetes-dynamic-pvc-10ded514-7625-11e8-8c52-42010af00003 us-west1-b 10 pd-
standard READY

```

### 23.2.5. Red Hat OpenShift Container Storage について

Red Hat OpenShift Container Storage (RHOCS) は、インハウスまたはハイブリッドクラウドのいずれの場合でも、OpenShift Container Platform のすべてに対応する永続ストレージのプロバイダーです。Red Hat ストレージソリューションとして、RHOCS は OpenShift Container Platform にインストール (コンバージド) されているか、または OpenShift Container Platform と共にインストール (インデペンデント) されているかを問わず、デプロイメント、管理およびモニタリングを目的として OpenShift Container Platform に完全に統合されます。OpenShift Container Storage は単一のアベイラビリティゾーンまたはノードに制限されないため、停止した場合にも存続できる可能性があります。RHOCS の詳細の使用方法については、[RHOCS3.11 Deployment Guide](#) を参照してください。

## 23.3. サービスとしての GCP 外部のロードバランサー使用

**LoadBalancer** サービスを使用してサービスを外部に公開することで、OpenShift Container Platform が GCP ロードバランサーを使用するように設定できます。OpenShift Container Platform は GCP にロードバランサーを作成し、必要なファイアウォールルールを作成します。

### 手順

1. 新規アプリケーションを作成します。

```
$ oc new-app openshift/hello-openshift
```

2. ロードバランサーサービスを公開します。

```
$ oc expose dc hello-openshift --name='hello-openshift-external' --type='LoadBalancer'
```

このコマンドは、以下の例とよく似た **LoadBalancer** サービスを作成します。

```

apiVersion: v1
kind: Service
metadata:

```

```

labels:
  app: hello-openshift
  name: hello-openshift-external
spec:
  externalTrafficPolicy: Cluster
  ports:
  - name: port-1
    nodePort: 30714
    port: 8080
    protocol: TCP
    targetPort: 8080
  - name: port-2
    nodePort: 30122
    port: 8888
    protocol: TCP
    targetPort: 8888
  selector:
    app: hello-openshift
    deploymentconfig: hello-openshift
  sessionAffinity: None
  type: LoadBalancer

```

3. サービスが作成されたことを確認します。

```

$ oc get svc
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
hello-openshift                     ClusterIP           172.30.62.10    <none>           8080/TCP,8888/TCP
20m
hello-openshift-external            LoadBalancer       172.30.147.214  35.230.97.224
8080:31521/TCP,8888:30843/TCP      19m

```

**LoadBalancer** タイプと外部 IP の値は、サービスが GCP ロードバランサーを使用してアプリケーションを公開することを示します。

OpenShift Container Platform は、GCP インフラストラクチャーに、必要なオブジェクトを作成します。以下に例を挙げます。

- ファイアウォール:

```

$ gcloud compute firewall-rules list | grep k8s
k8s-4612931a3a47c204-node-http-hc    my-net INGRESS 1000    tcp:10256
k8s-fw-a1a8afaa7762811e88c5242010af0000 my-net INGRESS 1000
tcp:8080,tcp:8888

```



### 注記

これらのファイアウォールは、**<openshift\_gcp\_prefix>ocp** のタグが付いたインスタンスに適用されます。たとえば、**openshift\_gcp\_prefix** パラメーターが **mycluster** に設定されている場合には、ノードに **myclusterocp** のタグを付ける必要があります。Compute Engine VM インスタンスにネットワークタグを追加する方法については、[ネットワークタグの追加と削除](#) を参照してください。

- ヘルスチェック:

■

```
$ gcloud compute http-health-checks list | grep k8s
k8s-4612931a3a47c204-node    10256 /healthz
```

- ロードバランサー:

```
$ gcloud compute target-pools list | grep k8s
a1a8afaa7762811e88c5242010af0000 us-west1 NONE          k8s-
4612931a3a47c204-node
$ gcloud compute forwarding-rules list | grep a1a8afaa7762811e88c5242010af0000
a1a8afaa7762811e88c5242010af0000 us-west1 35.230.97.224 TCP      us-
west1/targetPools/a1a8afaa7762811e88c5242010af0000
```

ロードバランサーが正しく設定されたことを確認するには、外部ホストから以下のコマンドを実行します。

```
$ curl 35.230.97.224:8080
Hello OpenShift!
```

## 第24章 AZURE の設定

OpenShift Container Platform を、[Microsoft Azure](#) ロードバランサーおよび [永続的なアプリケーションデータのディスク](#) を使用するように設定できます。

### 24.1. 作業を開始する前に

#### 24.1.1. Microsoft Azure の認証の設定

##### Azure ロール

OpenShift Container Platform 向けに Microsoft Azure を設定するには、以下の Microsoft Azure ロールが必要です。

Contributor	すべての種類の Microsoft Azure リソースを作成し、管理します。
-------------	---

詳細については、[従来のサブスクリプション管理者のロールと、Azure RBAC ロールと、Azure AD 管理者ロールのドキュメント](#) を参照してください。

##### パーミッション

Microsoft Azure を OpenShift Container Platform 用に設定するには、Kubernetes サービ出力ドバランサーおよび永続ストレージのディスクの作成および管理を可能にするサービスプリンシパルが必要になります。サービスプリンシパルの値はインストール時に定義され、OpenShift Container Platform マスターおよびノードホストの `/etc/origin/cloudprovider/azure.conf` にある Azure 設定ファイルにデプロイされます。

##### 手順

1. Azure CLI を使用して、アカウントのサブスクリプション ID を取得します。

```
# az account list
[
  {
    "cloudName": "AzureCloud",
    "id": "<subscription>", ❶
    "isDefault": false,
    "name": "Pay-As-You-Go",
    "state": "Enabled",
    "tenantId": "<tenant-id>",
    "user": {
      "name": "admin@example.com",
      "type": "user"
    }
  }
]
```

❶ 新規パーミッションを作成するために使用するサブスクリプション ID。

2. コントリビューターの Microsoft Azure ロールおよび Microsoft Azure サブスクリプションおよびリソースグループの範囲でサービスプリンシパルを作成します。インベントリを定義する際に使用されるこれらの値の出力を記録します。以下の値の代わりに、直前の手順の `<subscription>` 値を使用します。

```
# az ad sp create-for-rbac --name openshiftcloudprovider \
  --password <secret> --role contributor \
  --scopes /subscriptions/<subscription>/resourceGroups/<resource-group>

Retrying role assignment creation: 1/36
Retrying role assignment creation: 2/36
{
  "appId": "<app-id>",
  "displayName": "ocpcloudprovider",
  "name": "http://ocpcloudprovider",
  "password": "<secret>",
  "tenant": "<tenant-id>"
}
```

## 24.1.2. Microsoft Azure オブジェクトの設定

OpenShift Container Platform と Microsoft Azure を統合する際に、可用性が高く、全機能装備の環境を作成するには以下のコンポーネントまたはサービスが必要です。



### 重要

十分な量のインスタンスを起動できるようにするために、インスタンスの作成前に Microsoft から CPU クォータの引き上げを要求します。

### リソースグループ

リソースグループには、ネットワーク、ロードバランサー、仮想マシンおよび DNS を含む、デプロイメント用のすべての Microsoft Azure コンポーネントが含まれます。クォータおよびパーミッションをリソースグループに適用し、Microsoft Azure でデプロイされるリソースを制御し、管理できます。リソースグループは地理的リージョンごとに作成され、定義されます。OpenShift Container Platform 環境用に作成されるすべてのリソースは同じ地理的リージョン内および同じリソースグループ内に置かれる必要があります。

詳細は、[Azure Resource Manager overview](#) を参照してください。

### Azure 仮想ネットワーク

Azure 仮想ネットワークは Azure クラウドネットワークを相互に分離するために使用されます。インスタンスおよびロードバランサーは仮想ネットワークを使用して、相互の通信およびインターネットでの通信を許可します。仮想ネットワークは1つまたは多くのサブネットがリソースグループ内のコンポーネントによって使用されることを許可します。仮想ネットワークは各種の VPN サービスに接続することもでき、オンプレミスサービスとの通信が可能にします。

詳細は、[What is Azure Virtual Network?](#) を参照してください。

### Azure DNS

Azure は、内部のインターネットでアクセス可能なホスト名およびロードバランサーを解決するマネージド DNS サービスを提供します。リファレンス環境は DNS ゾーンを使用して3つの DNS A レコードをホストし、パブリック IP の OpenShift Container Platform リソースおよび bastion へのマッピングを許可します。

詳細は、[What is Azure DNS?](#) を参照してください。

### 負荷分散

Azure ロードバランサーは、Azure 環境内の仮想マシンで実行されているサービスのスケーリングおよび高可用性に必要なネットワークの接続性を確保します。

詳細は、[What is Azure Load Balancer?](#) を参照してください。

## ストレージアカウント

ストレージアカウントは、仮想マシンなどのリソースが Microsoft Azure で提供される各種のストレージコンポーネントにアクセスできるようにします。インストール時に、ストレージアカウントは OpenShift Container Platform レジストリーに使用されるオブジェクトベースの **blob** ストレージの場所を定義します。

詳細は、[Introduction to Azure Storage](#) を参照してください。レジストリーのストレージアカウントを作成する手順については、[OpenShift Container Platform レジストリーでの Microsoft Azure の設定](#) セクションを参照してください。

## サービスプリンシパル

Azure は、Azure 内でコンポーネントにアクセスし、管理し、または作成するサービスアカウントを作成する機能を提供します。サービスアカウントは、特定のサービスへの API アクセスを付与します。たとえば、サービスプリンシパルは Kubernetes または OpenShift Container Platform インスタンスが永続ストレージおよびロードバランサーを要求できるようにします。サービスプリンシパルは、インスタンスまたはユーザーに特定の機能についての綿密なアクセスを付与することを可能にします。

詳細は、[Application and service principal objects in Azure Active Directory](#) を参照してください。

## 可用性セット

可用性セットにより、デプロイされた VM をクラスター内の複数の分離したハードウェアノードに分散できます。この分散により、クラウドプロバイダーハードウェアのメンテナンスが行われる場合など、複数のインスタンスすべてが特定のノードで実行されないようにするのに役立ちます。

インスタンスをそれぞれのロールに基づいて異なる可用性セットにセグメント化する必要があります。たとえば、3つのマスターノードを含む1つの可用性セット、インフラストラクチャーホストを含む1つの可用性セット、アプリケーションホストを含む1つの可用性セットにセグメント化できます。可用性セットによってセグメント化が可能になり、OpenShift Container Platform 内で外部ロードバランサーを使用することができます。

詳細は、[Manage the availability of Linux virtual machines](#) を参照してください。

## ネットワークセキュリティグループ

ネットワークセキュリティグループ (NSG) は、Azure 仮想ネットワーク内にデプロイされたりソースへのトラフィックを許可/拒否するルールの一覧を提供します。NSG は数値の優先度の値およびルールを使用して相互の通信を許可される項目を定義します。通信を許可する場所を制限することができ、仮想ネットワーク内のみ、ロードバランサーから、または別の場所からなどと通信を制限することができます。

優先度の値により、管理者はポート通信が許可/拒否される順序について細かい値を付与することができます。

詳細は、[Plan virtual networks](#) を参照してください。

## インスタンスサイズ

正常な OpenShift Container Platform の環境には、最低でも以下のハードウェア要件を満たす必要があります。

詳細は、[OpenShift Container Platform ドキュメントの Minimum Hardware Requirements](#) セクション、または [Sizes for Cloud Services](#) を参照してください。

## 24.2. AZURE 設定ファイル

Azure について OpenShift Container Platform を設定するには、各ノードホストに `/etc/azure/azure.conf` ファイルが必要です。

ファイルが存在しない場合は、これを作成できます。

```
tenantId: <> 1
subscriptionId: <> 2
aadClientId: <> 3
aadClientSecret: <> 4
aadTenantId: <> 5
resourceGroup: <> 6
cloud: <> 7
location: <> 8
vnetName: <> 9
securityGroupName: <> 10
primaryAvailabilitySetName: <> 11
```

- 1 クラスタがデプロイされているサブスクリプションの AAD テナント ID。
- 2 クラスタがデプロイされている Azure サブスクリプション ID。
- 3 Azure RM API と対話するための RBAC アクセス権を持つ AAD アプリケーションのクライアント ID。
- 4 Azure RM API と対話するための RBAC アクセス権を持つ AAD アプリケーションのクライアントシークレット。
- 5 これがテナント ID と同一であることを確認します (オプション)。
- 6 Azure VM が属する Azure のリソースグループ名。
- 7 特定のクラウドリージョン。 **AzurePublicCloud** など。
- 8 コンパクトな形式の Azure リージョン。 **southeastasia** など (オプション)。
- 9 インスタンスを含む仮想ネットワーク。ロードバランサー作成時に使用します。
- 10 インスタンスとロードバランサーに関連付けられているセキュリティグループ名。
- 11 ロードバランサーなどのリソースの作成時に使用するよう設定されている可用性 (オプション)。



### 重要

インスタンスへのアクセスに使用される NIC には **internal-dns-name** が設定されている必要があります。これがないと、ノードはクラスタに再結合できず、コンソールにビルドログを表示できず、**oc rsh** が正常に機能しなくなる可能性があります。

## 24.3. MICROSOFT AZURE 上の OPENSIFT CONTAINER PLATFORM のインベントリーサンプル

以下のインベントリーサンプルでは、以下の項目が作成されていることを前提としています。

- リソースグループ
- Azure 仮想ネットワーク
- 必要な OpenShift Container Platform ポートが含まれる1つ以上のネットワークセキュリティグループ
- ストレージアカウント
- サービスプリンシパル
- 2つのロードバランサー
- ルーターおよび OpenShift Container Platform Web コンソールの2つ以上の DNS エントリー
- 3つの可用性セット
- 3つのマスターインスタンス
- 3つのインフラストラクチャーインスタンス
- 1つ以上のアプリケーションインスタンス

以下のインベントリーはデフォルトの **storageclass** を使用して、サービスプリンシパルで管理されるメトリクス、ロギング、およびサービスカタログコンポーネントで使用される永続ボリュームを作成します。レジストリーは Microsoft Azure Blob ストレージを使用します。

### 重要

Microsoft Azure インスタンスがマネージドディスクを使用する場合、インベントリーに以下の変数を指定します。

```
openshift_storageclass_parameters={'kind': 'managed', 'storageaccounttype': 'Premium_LRS'}
```

または

```
openshift_storageclass_parameters={'kind': 'managed', 'storageaccounttype': 'Standard_LRS'}
```

これにより、**storageclass** はデプロイされたインスタンスに関連する **PVs** の適切なディスクタイプを作成します。アンマネージドディスクが使用される場合、**storageclass** は **shared** パラメーターを使用して、アンマネージドディスクが **PVs** に作成されることを許可します。

```
[OSEv3:children]
masters
etcd
nodes
```

```
[OSEv3:vars]
```

```
ansible_ssh_user=cloud-user
ansible_become=true
openshift_cloudprovider_kind=azure

#cloudprovider
openshift_cloudprovider_kind=azure
openshift_cloudprovider_azure_client_id=v9c97ead-1v7E-4175-93e3-623211bed834
openshift_cloudprovider_azure_client_secret=s3r3tR3gistryN0special
openshift_cloudprovider_azure_tenant_id=422r3f91-21fe-4esb-vad5-d96dfeooee5d
openshift_cloudprovider_azure_subscription_id=6003c1c9-d10d-4366-86cc-e3dddcooe2d
openshift_cloudprovider_azure_resource_group=openshift
openshift_cloudprovider_azure_location=eastus
#endcloudprovider

oreg_auth_user=service_account ❶
oreg_auth_password=service_account_token ❷
openshift_master_api_port=443
openshift_master_console_port=443
openshift_hosted_router_replicas=3
openshift_hosted_registry_replicas=1
openshift_master_cluster_method=native
openshift_master_cluster_hostname=openshift-master.example.com
openshift_master_cluster_public_hostname=openshift-master.example.com
openshift_master_default_subdomain=apps.openshift.example.com
openshift_deployment_type=openshift-enterprise
openshift_master_identity_providers=[{'name': 'idm', 'challenge': 'true', 'login': 'true', 'kind':
'LDAPPasswordIdentityProvider', 'attributes': {'id': ['dn'], 'email': ['mail'], 'name': ['cn'],
'preferredUsername': ['uid']}, 'bindDN': 'uid=admin,cn=users,cn=accounts,dc=example,dc=com',
'bindPassword': 'ldapadmin', 'ca': '/etc/origin/master/ca.crt', 'insecure': 'false', 'url':
'ldap://ldap.example.com/cn=users,cn=accounts,dc=example,dc=com?uid?sub?(memberOf=cn=ose-
user,cn=groups,cn=accounts,dc=example,dc=com)'}]
networkPluginName=redhat/ovs-networkpolicy
openshift_examples_modify_imagestreams=true

# Storage Class change to use managed storage
openshift_storageclass_parameters={'kind': 'managed', 'storageaccounttype': 'Standard_LRS'}

# service catalog
openshift_enable_service_catalog=true
openshift_hosted_etcd_storage_kind=dynamic
openshift_hosted_etcd_storage_volume_name=etcd-vol
openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
openshift_hosted_etcd_storage_volume_size=SC_STORAGE
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}

# metrics
openshift_metrics_install_metrics=true
openshift_metrics_cassandra_storage_type=dynamic
openshift_metrics_storage_volume_size=20Gi
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra": "true"}

# logging
openshift_logging_install_logging=true
```

```

openshift_logging_es_pvc_dynamic=true
openshift_logging_storage_volume_size=50Gi
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"}

# Setup azure blob registry storage
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_azure_blob_accountkey=uZdkVlbca6xzwBqK8VDz15/loLUoc8I6cPfl
31ZS+QOSxL6yIWt6CLrcadSqvtNTMgztxH4CGjYfVnRNUhvMiA==
openshift_hosted_registry_storage_provider=azure_blob
openshift_hosted_registry_storage_azure_blob_accountname=registry
openshift_hosted_registry_storage_azure_blob_container=registry
openshift_hosted_registry_storage_azure_blob_realm=core.windows.net

[masters]
ocp-master-1
ocp-master-2
ocp-master-3

[etcd]
ocp-master-1
ocp-master-2
ocp-master-3

[nodes]
ocp-master-1 openshift_node_group_name="node-config-master"
ocp-master-2 openshift_node_group_name="node-config-master"
ocp-master-3 openshift_node_group_name="node-config-master"
ocp-infra-1 openshift_node_group_name="node-config-infra"
ocp-infra-2 openshift_node_group_name="node-config-infra"
ocp-infra-3 openshift_node_group_name="node-config-infra"
ocp-app-1 openshift_node_group_name="node-config-compute"

```

- 1** **2** デフォルトのコンテナイメージレジストリーなどの、認証を必要とするコンテナレジストリーを使用する場合、該当アカウントの認証情報を指定します。[Red Hat レジストリーのアクセスおよび設定](#) を参照してください。

## 24.4. OPENSIFT CONTAINER PLATFORM での MICROSOFT AZURE の設定

以下の 2 つの方法で OpenShift Container Platform で Microsoft Azure を設定できます。

- [Ansible の使用](#)
- [master-config.yaml ファイルを変更して手動で設定する](#)

### 24.4.1. Ansible を使用した OpenShift Container Platform での Azure の設定

インストール時に、Azure 向けに OpenShift Container Platform を設定できます。

以下をデフォルトで `/etc/ansible/hosts` にある Ansible インベントリーファイルに追加し、Microsoft Azure 用に OpenShift Container Platform 環境を設定します。

```
[OSEv3:vars]
openshift_cloudprovider_kind=azure
openshift_cloudprovider_azure_client_id=<app_ID> ①
openshift_cloudprovider_azure_client_secret=<secret> ②
openshift_cloudprovider_azure_tenant_id=<tenant_ID> ③
openshift_cloudprovider_azure_subscription_id=<subscription> ④
openshift_cloudprovider_azure_resource_group=<resource_group> ⑤
openshift_cloudprovider_azure_location=<location> ⑥
```

- ① サービスプリンシパルのアプリ ID。
- ② サービスプリンシパルのパスワードを含むシークレット。
- ③ サービスプリンシパルがあるテナント。
- ④ サービスプリンシパルによって使用されるサブスクリプション
- ⑤ サービスアカウントが存在するリソースグループ。
- ⑥ リソースグループが存在する Microsoft Azure の場所。

Ansible でインストールすると、Microsoft Azure 環境に適合するように、以下のファイルが作成されて設定されます。

- /etc/origin/cloudprovider/azure.conf
- /etc/origin/master/master-config.yaml
- /etc/origin/node/node-config.yaml

## 24.4.2. OpenShift Container Platform での Microsoft Azure の手動設定

### 24.4.2.1. Microsoft Azure 向けのマスターホストの手動設定

全マスターホストで以下の手順を実行します。

#### 手順

1. すべてのマスター上の `/etc/origin/master/master-config.yaml` にデフォルトで置かれているマスター設定ファイルを編集し、`apiServerArguments` および `controllerArguments` セクションの内容を更新します。

```
kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "azure"
    cloud-config:
      - "/etc/origin/cloudprovider/azure.conf"
  controllerArguments:
    cloud-provider:
```

```
- "azure"
cloud-config:
- "/etc/origin/cloudprovider/azure.conf"
```



### 重要

コンテナ化インストールをトリガーすると、`/etc/origin` と `/var/lib/origin` のディレクトリのみがマスターとノードのコンテナにマウントされます。したがって、`master-config.yaml` は `/etc/` ではなく `/etc/origin/master` ディレクトリになければなりません。

2. Ansible を使用して Microsoft Azure について OpenShift Container Platform を設定する場合に、`/etc/origin/cloudprovider/azure.conf` ファイルが自動的に作成されます。OpenShift Container Platform で Microsoft Azure を手動で設定するため、すべてのノードインスタンスでファイルを作成し、以下を含める必要があります。

```
tenantId: <tenant_ID> ①
subscriptionId: <subscription> ②
aadClientId: <app_ID> ③
aadClientSecret: <secret> ④
aadTenantId: <tenant_ID> ⑤
resourceGroup: <resource_group> ⑥
location: <location> ⑦
```

- ① サービスプリンシパルがあるテナント。
- ② サービスプリンシパルによって使用されるサブスクリプション
- ③ サービスプリンシパルの appID の値。
- ④ サービスプリンシパルのパスワードを含むシークレット。
- ⑤ サービスプリンシパルがあるテナント。
- ⑥ サービスアカウントが存在するリソースグループ。
- ⑦ リソースグループが存在する Microsoft Azure の場所。

3. 次に OpenShift Container Platform マスターサービスを再起動します。

```
# master-restart api
# master-restart controllers
```

#### 24.4.2.2. Microsoft Azure 向けのノードホストの手動設定

全ノードホスト上で以下を実行します。

#### 手順

1. 適切な [ノード設定マップ](#) を編集して、`kubeletArguments` セクションの内容を更新します。

```
kubeletArguments:
```

```
cloud-provider:
- "azure"
cloud-config:
- "/etc/origin/cloudprovider/azure.conf"
```



### 重要

インスタンスへのアクセスに使用される NIC には内部 DNS 名が設定されている必要があります。これがないと、ノードはクラスターに再結合できず、コンソールにビルドログを表示できず、**oc rsh** が正常に機能しなくなる可能性があります。

2. すべてのノードで OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

### 24.4.3. Microsoft Azure の OpenShift Container Platform レジストリーの設定

Microsoft Azure は、OpenShift Container Platform が OpenShift Container Platform コンテナイメージレジストリーを使用してコンテナイメージを保存するために使用できるオブジェクトクラウドストレージを提供します。

詳細は、[Azure ドキュメントでクラウドストレージについて](#) を参照してください。

レジストリーは、Ansible を使用するか、またはレジストリー設定ファイルを手動で設定して設定することができます。

#### 前提条件

インストール前にレジストリーイメージをホストするためのストレージアカウントを作成する必要があります。以下のコマンドは、インストール時にイメージストレージに使用されるサービスアカウントを作成します。

コンテナイメージを保存するために Microsoft Azure Blob ストレージを使用できます。OpenShift Container Platform レジストリーは Blob ストレージを使用して、管理者の介入なしにレジストリーのサイズを動的に拡張できるようにします。

1. Azure ストレージアカウントを作成します。

```
az storage account create
--name <account_name> \
--resource-group <resource_group> \
--location <location> \
--sku Standard_LRS
```

これによってアカウントキーが作成されます。アカウントキーを表示するには、以下を実行します。

```
az storage account keys list \
--account-name <account-name> \
--resource-group <resource-group> \
--output table
```

KeyName	Permissions	Value
key1	Full	<account-key>
key2	Full	<extra-account-key>

OpenShift Container Platform レジストリーの設定には、1つのアカウントキー値のみが必要になります。

## オプション 1: Ansible を使用した Azure の OpenShift Container Platform レジストリーの設定

### 手順

1. レジストリーでストレージアカウントを使用できるように Ansible インベントリを設定します。

```
[OSEv3:vars]
# Azure Registry Configuration
openshift_hosted_registry_replicas=1 ①
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_azure_blob_accountkey=<account_key> ②
openshift_hosted_registry_storage_provider=azure_blob
openshift_hosted_registry_storage_azure_blob_accountname=<account_name> ③
openshift_hosted_registry_storage_azure_blob_container=<registry> ④
openshift_hosted_registry_storage_azure_blob_realm=core.windows.net
```

- ① 設定するレプリカ数
- ② <account-name> に関連付けられたアカウントキー。
- ③ ストレージアカウント名。
- ④ データの保存に使用するディレクトリー。デフォルトで **registry** です。

## オプション 2: Microsoft Azure についての OpenShift Container Platform レジストリーの手動設定

Microsoft Azure オブジェクトストレージを使用するには、レジストリーの設定ファイルを編集してレジストリー Pod にマウントします。

### 手順

1. 現在の **config.yml** をエクスポートします。

```
$ oc get secret registry-config \
  -o jsonpath='{.data.config\.yml}' -n default | base64 -d \
  >> config.yml.old
```

2. 古い **config.yml** から新規の設定ファイルを作成します。

```
$ cp config.yml.old config.yml
```

3. ファイルを編集して Azure パラメーターを含めます。

```
storage:
  delete:
```

```

enabled: true
cache:
  blobdescriptor: inmemory
azure:
  accountname: <account-name> ❶
  accountkey: <account-key> ❷
  container: registry ❸
  realm: core.windows.net ❹

```

- ❶ ストレージアカウント名に置き換えます。
- ❷ <account-name> に関連付けられたアカウントキー。
- ❸ データの保存に使用するディレクトリー。デフォルトで **registry** です。
- ❹ デフォルトのストレージレルム **core.windows.net**

4. **registry-config** シークレットを削除します。

```
$ oc delete secret registry-config -n default
```

5. シークレットを再作成して、更新された設定ファイルを参照します。

```
$ oc create secret generic registry-config \
  --from-file=config.yml -n default
```

6. 更新された設定を読み取るためにレジストリーを再デプロイします。

```
$ oc rollout latest docker-registry -n default
```

### レジストリーが Blob オブジェクトストレージを使用していることの確認

レジストリーが Microsoft Azure Blob ストレージを使用しているかどうかを確認するには、以下を実行します。

#### 手順

1. レジストリーの正常なデプロイ後に、レジストリー **deploymentconfig** は常にレジストリーが Microsoft Azure Blob ストレージではなく **emptydir** を使用していることを示します。

```

$ oc describe dc docker-registry -n default
...
Mounts:
...
  /registry from registry-storage (rw)
Volumes:
registry-storage:
Type:    EmptyDir ❶
...

```

- ❶ Pod の寿命を共有する一時ディレクトリー

2. `/registry` のマウントポイントが空かどうかを確認します。これは、Microsoft Azure ストレージが使用するボリュームです。

```
$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') -i -t -- ls -l /registry
total 0
```

3. 空の場合は、Microsoft Azure Blob 設定が **registry-config** シークレットで実行されていることを示します。

```
$ oc describe secret registry-config
Name:      registry-config
Namespace: default
Labels:    <none>
Annotations: <none>

Type: Opaque

Data
====
config.yml: 398 bytes
```

4. インストーラーは、[インストールドキュメントのストレージセクション](#) で記載されているように、拡張されたレジストリー機能を使用して、希望の設定で `config.yml` ファイルを作成します。以下のコマンドで、ストレージバケット設定が保存されている **storage** セクションを含めて設定ファイルを表示します。

```
$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') \
  cat /etc/registry/config.yml

version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
  azure:
    accountname: registry
    accountkey:
uZekVBJBa6xzwAqK8EDz15/hoHUoc8I6cPfP31ZS+QOSxLfo7WT7CLrVPKaqvtNTMgztxH7C
GjYfpFRNUhvMiA==
    container: registry
    realm: core.windows.net
auth:
  openshift:
    realm: openshift
middleware:
  registry:
```

```

- name: openshift
repository:
- name: openshift
options:
  pullthrough: True
  acceptschema2: True
  enforcequota: False
storage:
- name: openshift

```

または、以下でシークレットを表示できます。

```

$ oc get secret registry-config -o jsonpath='{.data.config\.yaml}' | base64 -d
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  delete:
    enabled: true
  cache:
    blobdescriptor: inmemory
azure:
  accountname: registry
  accountkey:
uZekVBJBa6xzwAqK8EDz15/hoHUoc8I6cPfP31ZS+QOSxLfo7WT7CLrVPKaqvtNTMgztxH7C
GjYfpFRNUhvMiA==
  container: registry
  realm: core.windows.net
auth:
  openshift:
    realm: openshift
middleware:
  registry:
- name: openshift
repository:
- name: openshift
options:
  pullthrough: True
  acceptschema2: True
  enforcequota: False
storage:
- name: openshift

```

**emptyDir** ボリュームを使用する場合には、**/registry** マウントポイントは以下ようになります。

```

$ oc exec \
  $(oc get pod -l deploymentconfig=docker-registry \
    -o=jsonpath='{.items[0].metadata.name}') -i -t -- df -h /registry
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc        30G  226M  30G   1% /registry

```

```
$ oc exec \
```

```
$(oc get pod -l deploymentconfig=docker-registry \
  -o=jsonpath='{.items[0].metadata.name}') -i -t -- ls -l /registry
total 0
drwxr-sr-x. 3 1000000000 1000000000 22 Jun 19 12:24 docker
```

#### 24.4.4. OpenShift Container Platform を Microsoft Azure ストレージを使用するように設定する

OpenShift Container Platform は、永続ボリュームメカニズムを活用して Microsoft Azure ストレージを使用できます。OpenShift Container Platform は、リソースグループにディスクを作成して、正しいインスタンスにこのディスクを割り当てます。

##### 手順

1. 以下の **storageclass** は、インストール時に、Ansible インベントリで **openshift\_cloudprovider\_kind=azure** および **openshift\_cloud\_provider\_azure** 変数を使用して Azure クラウドプロバイダーを設定する場合に作成されます。

```
$ oc get --export storageclass azure-standard -o yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  creationTimestamp: null
  name: azure-standard
parameters:
  kind: Shared
  storageaccounttype: Standard_LRS
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

Ansible を使用して OpenShift Container Platform と Microsoft Azure の統合を有効にしていない場合には、**storageclass** を手動で作成できます。詳細は、[動的プロビジョニングとストレージクラスの作成](#) セクションを参照してください。

2. 現時点で、デフォルトの **storageclass** の種類は **shared** であり、これは Microsoft Azure インスタンスがアンマネージドディスクを使用する必要があることを意味しています。インストール時に Ansible インベントリファイルに **openshift\_storageclass\_parameters={'kind': 'Managed', 'storageaccounttype': 'Premium\_LRS'}** または **openshift\_storageclass\_parameters={'kind': 'Managed', 'storageaccounttype': 'Standard\_LRS'}** 変数を指定し、インスタンスがマネージドディスクを使用できるようにすることで、この設定をオプションで変更することができます。



##### 注記

Microsoft Azure ディスクは **ReadWriteOnce** アクセスモードで、1つのノードで読み取り/書き込み可能な状態でボリュームをマウントできます。詳細情報は、[アーキテクチャガイドのアクセスモードのセクション](#) を参照してください。

#### 24.4.5. Red Hat OpenShift Container Storage について

Red Hat OpenShift Container Storage (RHOCS) は、インハウスまたはハイブリッドクラウドのいずれ

の場合でも、OpenShift Container Platform のすべてに対応する永続ストレージのプロバイダーです。Red Hat ストレージソリューションとして、RHOCS は OpenShift Container Platform にインストール (コンバージド) されているか、または OpenShift Container Platform と共にインストール (インデペンデント) されているかを問わず、デプロイメント、管理およびモニタリングを目的として OpenShift Container Platform に完全に統合されます。OpenShift Container Storage は単一のアベイラビリティゾーンまたはノードに制限されないため、停止した場合にも存続できる可能性があります。RHOCS の詳細の使用方法については、[RHOCS3.11 Deployment Guide](#) を参照してください。

## 24.5. MICROSOFT AZURE 外部ロードバランサーのサービスとしての使用

OpenShift Container Platform は、**LoadBalancer** サービスを使用してサービスを外部に公開することで Microsoft Azure ロードバランサーを利用できます。OpenShift Container Platform は、ロードバランサーを Microsoft Azure で作成し、適切なファイアウォールルールを作成します。



### 重要

現時点で、Microsoft Azure インフラストラクチャーをクラウドプロバイダーとして使用し、またこれを外部ロードバランサーとして使用する際に追加の変数がこれに組み込まれるというバグがありました。詳細は、以下を参照してください。

- [https://bugzilla.redhat.com/show\\_bug.cgi?id=1613546](https://bugzilla.redhat.com/show_bug.cgi?id=1613546)

### 前提条件

`/etc/origin/cloudprovider/azure.conf` にある [Azure 設定ファイル](#) が適切なオブジェクトで適切に設定されていることを確認します。`/etc/origin/cloudprovider/azure.conf` ファイルのサンプルについては、[OpenShift Container Platform での Microsoft Azure の手動設定](#) のセクションを参照してください。

値が追加されたら、すべてのホストで OpenShift Container Platform を再起動します。

```
# systemctl restart atomic-openshift-node
# master-restart api
# master-restart controllers
```

### 24.5.1. ロードバランサーを使用したアプリケーションサンプルのデプロイ

#### 手順

1. 新規アプリケーションを作成します。

```
$ oc new-app openshift/hello-openshift
```

2. ロードバランサーサービスを公開します。

```
$ oc expose dc hello-openshift --name='hello-openshift-external' --type='LoadBalancer'
```

これにより、以下と同様の **Loadbalancer** サービスが作成されます。

```
apiVersion: v1
kind: Service
metadata:
  labels:
```

```

  app: hello-openshift
  name: hello-openshift-external
spec:
  externalTrafficPolicy: Cluster
  ports:
  - name: port-1
    nodePort: 30714
    port: 8080
    protocol: TCP
    targetPort: 8080
  - name: port-2
    nodePort: 30122
    port: 8888
    protocol: TCP
    targetPort: 8888
  selector:
    app: hello-openshift
    deploymentconfig: hello-openshift
  sessionAffinity: None
  type: LoadBalancer

```

3. サービスが作成されたことを確認します。

```

$ oc get svc
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
hello-openshift                     ClusterIP           172.30.223.255  <none>           8080/TCP,8888/TCP
1m
hello-openshift-external            LoadBalancer       172.30.99.54    40.121.42.180
8080:30714/TCP,8888:30122/TCP      4m

```

**LoadBalancer** タイプおよび **External-IP** フィールドは、サービスが Microsoft Azure ロードバランサーを使用してアプリケーションを公開することを示しています。

これにより、Azure インフラストラクチャーで以下の必要なオブジェクトが作成されます。

- ロードバランサー:

```

az network lb list -o table
Location  Name          ProvisioningState  ResourceGroup  ResourceGuid
-----  -
eastus   kubernetes   Succeeded          refarch-azr    30ec1980-b7f5-407e-aa4f-
e570f06f168d
eastus   OcpMasterLB  Succeeded          refarch-azr    acb537b2-8a1a-45d2-aae1-
ea9eabfaea4a
eastus   OcpRouterLB  Succeeded          refarch-azr    39087c4c-a5dc-457e-a5e6-
b25359244422

```

ロードバランサーが正しく設定されたことを確認するには、外部ホストから以下を実行します。

```

$ curl 40.121.42.180:8080 1
Hello OpenShift!

```

- 1** 上記の **EXTERNAL-IP** 検証手順の値およびポート番号に置き換えます。

## 第25章 VMWARE VSPHERE の設定

OpenShift Container Platform は、[VMware vSphere VMDK](#) を使用して PersistentVolume をサポートするように設定できます。この設定には、アプリケーションデータについて [VMware vSphere VMDK を永続ストレージとして使用する](#) ことが含まれます。

vSphere Cloud Provider は OpenShift Container Platform で vSphere で管理されたストレージを使用することを許可し、Kubernetes が使用するすべてのストレージプリミティブをサポートします。

- PersistentVolume (PV)
- PersistentVolumesClaim (PVC)
- StorageClass

ステートフルなコンテナ化されたアプリケーションによって要求される PersistentVolume は、VMware vSAN、VVOL、VMFS、または NFS データストアでプロビジョニングできます。

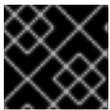
Kubernetes PV は Pod 仕様で定義されます。これらは動的プロビジョニングの使用時に静的プロビジョニングまたは PVC を使用する場合 (これは推奨される方法です)、VMDK ファイルを直接参照できます。

vSphere Cloud Provider への最新更新は [Kubernetes 用の vSphere ストレージ](#) にあります。

### 25.1. 作業を開始する前に

#### 25.1.1. 要件

VMware vSphere



#### 重要

スタンドアロンの ESXi はサポートされません。

- 完全な [VMware 検証設計](#) をサポートする場合、vSphere バージョン 6.0.x の推奨される最小バージョンの 6.7 U1b が必要になります。
- vSAN、VMFS および NFS がサポートされます。
  - vSAN サポートは、1つの vCenter の1つのクラスターに制限されます。



#### 注記

OpenShift Container Platform 3.11 がサポートされており、vSphere 7 クラスターにデプロイされます。vSphere インツリーストレージドライバーを使用する場合は、vSAN、VMFS、および NFS ストレージオプションもサポートされます。

#### 前提条件

各ノードの仮想マシンに VMware ツールをインストールする必要があります。詳しい情報は、[VMware ツールのインストール](#) を参照してください。

追加の設定およびトラブルシューティングにオープンソース VMware **govmomi** CLI ツールを使用できます。以下は、**govc** CLI 設定例になります。

```
export GOVC_URL='vCenter IP OR FQDN'
export GOVC_USERNAME='vCenter User'
export GOVC_PASSWORD='vCenter Password'
export GOVC_INSECURE=1
```

### 25.1.1.1. パーミッション

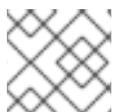
ロールを作成し、これを vSphere Cloud Provider に割り当てます。必要な [特権](#) セットを持つ vCenter ユーザーが必要です。

通常、vSphere Cloud Provider に指定される vSphere ユーザーには以下のパーミッションがなければなりません。

- **folder**、**host**、datacenter、datastore folder、datastore cluster などのノード仮想マシンの親エンティティの **Read** パーミッション。
- **vsphere.conf** で定義されるリソースプールの **VirtualMachine.Inventory.Create/Delete** パーミッション: これは、テスト仮想マシンを作成し、削除するために使用されます。

カスタムロール、ユーザー、ロールの割り当ての作成手順については、[vSphere Documentation Center](#) を参照してください。

vSphere Cloud Provider は、複数の vCenter にまたがる OpenShift Container Platform クラスターをサポートします。上記のすべての特権がすべての vCenter に対して適切に設定されていることを確認します。



#### 動的プロビジョニングのパーミッション

動的永続ボリュームの作成が推奨される方法です。

ロール	権限	エンティティ	子への伝播
manage-k8s-node-vm	Resource.AssignVMToPool, VirtualMachine.Config.AddExistingDisk, VirtualMachine.Config.AddNewDisk, VirtualMachine.Config.AddRemoveDevice, VirtualMachine.Config.RemoveDisk, VirtualMachine.Inventory.Create, VirtualMachine.Inventory.Delete, VirtualMachine.Config.Settings	クラスター、ホスト、 VM フォルダー	はい

ロール	権限	エンティティ	子への伝播
manage-k8s-volumes	Datastore.AllocateSpace, Datastore.FileManagement (低レベルのファイル操作)	データストア	いいえ
k8s-system-read-and-spbm-profile-view	StorageProfile.View (プロファイルで起動されるストレージビュー)	vCenter	いいえ
読み取り専用 (既存のデフォルトロール)	System.Anonymous System.Read System.View	データセンター、データストアクラスター、データストアストレージフォルダー	いいえ



### 静的プロビジョニングのパーミッション

静的にプロビジョニングされる PV にバインドする PVC を使用し、削除する回収ポリシーを設定する場合に、Datastore.FileManagement は manage-k8s-volumes ロールにのみ必要になります。PVC が削除される場合、関連付けられる静的にプロビジョニングされた PV も削除されます。

ロール	権限	Entities	子への伝播
manage-k8s-node-vm	VirtualMachine.Config.AddExistingDisk, VirtualMachine.Config.AddNewDisk, VirtualMachine.Config.AddRemoveDevice, VirtualMachine.Config.RemoveDisk	VM フォルダー	はい
manage-k8s-volumes	Datastore.FileManagement (低レベルのファイル操作)	データストア	いいえ
読み取り専用 (既存のデフォルトロール)	System.Anonymous System.Read System.View	vCenter、データセンター、データストアクラスター、データストアストレージフォルダー、クラスター、ホスト	No ...

### 手順

1. **VM フォルダー** を作成し、OpenShift Container Platform ノード VM をこのフォルダーに移動します。

2. 各ノード仮想マシンについて **disk.EnableUUID** パラメーターを **true** に設定します。この設定により、VMware vSphere の Virtual Machine Disk (VMDK) が一貫性のある UUID を常に仮想マシンに提示し、ディスクが正常にマウントされるようになります。  
クラスターに参加するすべての仮想マシンノードでは、**disk.EnableUUID** パラメーターが **true** に設定される必要があります。この値を設定するには、vSphere コンソールまたは **govc** CLI ツールのいずれかの手順を実行します。
  - a. vSphere HTML クライアントから、**VM properties** → **VM Options** → **Advanced** → **Configuration Parameters** → **disk.enableUUID=TRUE** に移動します。
  - b. または、govc CLI を使用してノード仮想マシンのパスを見つけます。

```
$govc ls /datacenter/vm/<vm-folder-name>
```

- i. すべての VM について、**disk.EnableUUID** を **true** に設定します。

```
$govc vm.change -e="disk.enableUUID=1" -vm='VM Path'
```



### 注記

OpenShift Container Platform ノード仮想マシンが仮想マシンテンプレートで作成されている場合、**disk.EnableUUID=1** をテンプレート仮想マシンに設定することができます。このテンプレートからクローン作成される仮想マシンはこのプロパティを継承しません。

## 25.1.1.2. OpenShift Container Platform と vMotion の使用



### 重要

OpenShift Container Platform は通常、コンピュータのみの vMotion をサポートしません。Storage vMotion を使用すると問題が発生する可能性があるため、これはサポートされていません。

Pod で vSphere ボリュームを使用している場合、手動でまたは Storage vMotion を使用して仮想マシンをデータストア間で移行すると、OpenShift Container Platform 永続ボリューム (PV) オブジェクト内で無効な参照が発生します。これらの参照により、影響を受ける Pod が起動しなくなり、データが失われる可能性があります。

同様に、OpenShift Container Platform は、仮想マシンのプロビジョニング用にデータストアクラスターを、または PV の動的または静的プロビジョニング用にデータストアクラスターを使用するか、PV の動的または静的プロビジョニング用にデータストアクラスターの一部であるデータストアを使用した VMDK のデータストア間での選択的な移行をサポートしません。

## 25.2. OPENSIFT CONTAINER PLATFORM での VSPHERE の設定

OpenShift Container Platform は、vSphere 用に 2 種類の方法で設定できます。

- [Ansible の使用](#)
- [master-config.yaml ファイルを変更して手動で設定する](#)

### 25.2.1. オプション 1: Ansible を使用した OpenShift Container Platform での vSphere の設定

OpenShift Container Platform での VMware vSphere (VCP) の設定は、[Ansible インベントリーファイル](#) を変更して行うことができます。これらの変更はインストール前に、または既存のクラスターに対して実行できます。

## 手順

1. Ansible インベントリーファイルに以下を追加します。

```
[OSEv3:vars]
openshift_cloudprovider_kind=vsphere
openshift_cloudprovider_vsphere_username=administrator@vsphere.local ❶
openshift_cloudprovider_vsphere_password=<password>
openshift_cloudprovider_vsphere_host=10.x.y.32 ❷
openshift_cloudprovider_vsphere_datacenter=<Datacenter> ❸
openshift_cloudprovider_vsphere_datastore=<Datastore> ❹
```

- ❶ vSphere でディスクを作成し、割り当てるために必要なパーミッションを持つユーザー名。
- ❷ vCenter サーバーアドレス。
- ❸ OpenShift Container Platform VM が置かれる vCenter Datacenter 名。
- ❹ VMDK の作成に使用されるデータストア。

2. **deploy\_cluster.yml** Playbook を実行します。

```
$ ansible-playbook -i <inventory_file> \
  playbooks/deploy_cluster.yml
```

Ansible でインストールすると、vSphere 環境に適合するように、以下のファイルが作成されて設定されます。

- **/etc/origin/cloudprovider/vsphere.conf**
- **/etc/origin/master/master-config.yaml**
- **/etc/origin/node/node-config.yaml**

参照として、詳細のインベントリーは以下のように表示されます。

**openshift\_cloudprovider\_vsphere** 値は、OpenShift Container Platform が永続ボリュームのデータストアで VMDK などの **vSphere** リソースを作成できるようにするために必要です。

```
$ cat /etc/ansible/hosts

[OSEv3:children]
ansible
masters
infras
apps
etcd
nodes
lb
```

```
[OSEv3:vars]
become=yes
ansible_become=yes
ansible_user=root
oreg_auth_user=service_account ❶
oreg_auth_password=service_account_token ❷
openshift_deployment_type=openshift-enterprise
# Required per https://access.redhat.com/solutions/3480921
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

# vSphere Cloud provider
openshift_cloudprovider_kind=vsphere
openshift_cloudprovider_vsphere_username="administrator@vsphere.local"
openshift_cloudprovider_vsphere_password="password"
openshift_cloudprovider_vsphere_host="vcsa65-dc1.example.com"
openshift_cloudprovider_vsphere_datacenter=Datacenter
openshift_cloudprovider_vsphere_cluster=Cluster
openshift_cloudprovider_vsphere_resource_pool=ResourcePool
openshift_cloudprovider_vsphere_datastore="datastore"
openshift_cloudprovider_vsphere_folder="folder"

# Service catalog
openshift_hosted_etcd_storage_kind=dynamic
openshift_hosted_etcd_storage_volume_name=etcd-vol
openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
openshift_hosted_etcd_storage_volume_size=1G
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}

openshift_master_ldap_ca_file=/home/cloud-user/mycert.crt
openshift_master_identity_providers=[{'name': 'idm', 'challenge': 'true', 'login': 'true', 'kind':
'LDAPPasswordIdentityProvider', 'attributes': {'id': ['dn'], 'email': ['mail'], 'name': ['cn'],
'preferredUsername': ['uid']}, 'bindDN': 'uid=admin,cn=users,cn=accounts,dc=example,dc=com',
'bindPassword': 'ldapadmin', 'ca': '/etc/origin/master/ca.crt', 'insecure': 'false', 'url':
'ldap://ldap.example.com/cn=users,cn=accounts,dc=example,dc=com?uid?sub?(memberOf=cn=ose-
user,cn=groups,cn=accounts,dc=openshift,dc=com)'}]

# Setup vsphere registry storage
openshift_hosted_registry_storage_kind=vsphere
openshift_hosted_registry_storage_access_modes=["ReadWriteOnce"]
openshift_hosted_registry_storage_annotations=['volume.beta.kubernetes.io/storage-provisioner:
kubernetes.io/vsphere-volume']
openshift_hosted_registry_replicas=1

openshift_hosted_router_replicas=3
openshift_master_cluster_method=native
openshift_node_local_quota_per_fsgroup=512Mi

default_subdomain=example.com
openshift_master_cluster_hostname=openshift.example.com
openshift_master_cluster_public_hostname=openshift.example.com
openshift_master_default_subdomain=apps.example.com

os_sdn_network_plugin_name='redhat/openshift-ovs-networkpolicy'
osm_use_cockpit=true
```

```
# Red Hat subscription name and password
rsub_user=username
rsub_pass=password
rsub_pool=8a85f9815e9b371b015e9b501d081d4b

# metrics
openshift_metrics_install_metrics=true
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=25Gi

# logging
openshift_logging_install_logging=true
openshift_logging_es_pvc_dynamic=true
openshift_logging_es_pvc_size=30Gi
openshift_logging_elasticsearch_storage_type=pvc
openshift_logging_es_cluster_size=1
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_fluentd_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_storage_kind=dynamic

#registry
openshift_public_hostname=openshift.example.com

[ansible]
localhost

[masters]
master-0.example.com vm_name=master-0 ipv4addr=10.x.y.103
master-1.example.com vm_name=master-1 ipv4addr=10.x.y.104
master-2.example.com vm_name=master-2 ipv4addr=10.x.y.105

[infras]
infra-0.example.com vm_name=infra-0 ipv4addr=10.x.y.100
infra-1.example.com vm_name=infra-1 ipv4addr=10.x.y.101
infra-2.example.com vm_name=infra-2 ipv4addr=10.x.y.102

[apps]
app-0.example.com vm_name=app-0 ipv4addr=10.x.y.106
app-1.example.com vm_name=app-1 ipv4addr=10.x.y.107
app-2.example.com vm_name=app-2 ipv4addr=10.x.y.108

[etcd]
master-0.example.com
master-1.example.com
master-2.example.com

[[lb]
haproxy-0.example.com vm_name=haproxy-0 ipv4addr=10.x.y.200

[nodes]
master-0.example.com openshift_node_group_name="node-config-master"
openshift_schedulable=true
master-1.example.com openshift_node_group_name="node-config-master"
```

```

openshift_schedulable=true
master-2.example.com openshift_node_group_name="node-config-master"
openshift_schedulable=true
infra-0.example.com openshift_node_group_name="node-config-infra"
infra-1.example.com openshift_node_group_name="node-config-infra"
infra-2.example.com openshift_node_group_name="node-config-infra"
app-0.example.com openshift_node_group_name="node-config-compute"
app-1.example.com openshift_node_group_name="node-config-compute"
app-2.example.com openshift_node_group_name="node-config-compute"

```

- 1** **2** デフォルトのコンテナイメージレジストリーなどの、認証を必要とするコンテナレジストリーを使用する場合、該当アカウントの認証情報を指定します。[Red Hat レジストリーのアクセスおよび設定](#) を参照してください。



### 注記

[vSphere 仮想マシン環境のデプロイ](#) は、Red Hat では正式にサポートされていませんが、設定が可能です。

## 25.2.2. オプション 2: OpenShift Container Platform での vSphere の手動設定

### 25.2.2.1. vSphere 向けのマスターホストの手動設定

全マスターホストで以下の手順を実行します。

#### 手順

1. すべてのマスターのデフォルトで `/etc/origin/master/master-config.yaml` に置かれているマスター設定ファイルを編集し、**apiServerArguments** および **controllerArguments** セクションの内容を更新します。

```

kubernetesMasterConfig:
  ...
  apiServerArguments:
    cloud-provider:
      - "vsphere"
    cloud-config:
      - "/etc/origin/cloudprovider/vsphere.conf"
  controllerArguments:
    cloud-provider:
      - "vsphere"
    cloud-config:
      - "/etc/origin/cloudprovider/vsphere.conf"

```



### 重要

コンテナ化インストールをトリガーすると、`/etc/origin` と `/var/lib/origin` のディレクトリーのみがマスターとノードのコンテナにマウントされます。したがって、`master-config.yaml` は `/etc/` ではなく `/etc/origin/master` になければなりません。

2. Ansible を使用して vSphere 用に OpenShift Container Platform を設定する場合に

は、`/etc/origin/cloudprovider/vsphere.conf` ファイルは自動的に作成されます。vSphere 用の OpenShift Container Platform を手動で設定する場合、このファイルを作成する必要があります。このファイルを作成する前に、複数の vCenter ゾーンが必要かどうかを判別します。クラスタのインストールプロセスは、デフォルトでシングルゾーンまたは単一 vCenter を設定します。異なるゾーンの vSphere に OpenShift Container Platform をデプロイすると、単一障害点を回避しやすくなりますが、ゾーン間での共有ストレージの必要が生じます。OpenShift Container Platform ノードホストがゾーン A でダウンした場合、Pod はゾーン B に移動する必要があります。詳細については、Kubernetes ドキュメントの [複数のゾーンでの実行](#) を参照してください。

- 単一の vCenter サーバーを設定するには、`/etc/origin/cloudprovider/vsphere.conf` ファイルに以下の形式を使用します。

```
[Global] 1
    user = "myusername" 2
    password = "mypassword" 3
    port = "443" 4
    insecure-flag = "1" 5
    datacenters = "mydatacenter" 6

[VirtualCenter "10.10.0.2"] 7
    user = "myvCenterusername"
    password = "password"

[Workspace] 8
    server = "10.10.0.2" 9
    datacenter = "mydatacenter"
    folder = "path/to/vms" 10
    default-datastore = "shared-datastore" 11
    resourcepool-path = "myresourcepoolpath" 12

[Disk]
    scsicontrollertype = pvscsi 13

[Network]
    public-network = "VM Network" 14
```

- 1 **[Global]** セクションに設定されるプロパティは、個別の **[VirtualCenter]** セクションの設定で上書きされない限り、すべての指定される vcenter で使用されます。
- 2 vSphere クラウドプロバイダーの vCenter ユーザー名。
- 3 指定されたユーザーの vCenter パスワード。
- 4 オプション。vCenter サーバーのポート番号。デフォルトはポート **443** になります。
- 5 vCenter が自己署名証明書を使用している場合は **1** に設定します。
- 6 ノード VM がデプロイされているデータセンターの名前。
- 7 この Virtual Center の特定の **[Global]** プロパティを上書きします。使用できる設定は **[Port]**、**[user]**、**[insecure-flag]**、**[datacenters]** です。指定されない設定は **[Global]** セクションからプルされます。

- 8 各種の vSphere Cloud Provider 機能で使用されるプロパティを設定します。たとえば、動的プロビジョニング、ストレージプロファイルベースのボリュームプロビジョニングなどがこれに含まれます。
  - 9 vCenter サーバーの IP アドレスまたは FQDN。
  - 10 ノード VM の VM ディレクトリーへのパス。
  - 11 ストレージクラスまたは動的プロビジョニングを使ったボリュームのプロビジョニングに使用されるデータストアの名前に設定されます。OpenShift Container Platform 3.9 よりも前のバージョンでは、データストアがストレージディレクトリーにあるか、またはデータストアクラスターのメンバーである場合には、完全パスが必要でした。
  - 12 オプション。ストレージプロファイルベースのボリュームプロビジョニングのダミー VM が作成される必要のあるリソースプールのパスに設定されます。
  - 13 VMDK が VM に割り当てられる際に設定される SCSI コントローラーのタイプ。
  - 14 vSphere がノードにアクセスするために使用されるネットワークポートグループに設定されます。これはデフォルトで VM ネットワークと呼ばれます。これは Kubernetes に登録されているノードホストの ExternalIP です。
- 複数の vCenter サーバーを設定するには、`/etc/origin/cloudprovider/vsphere.conf` ファイルに以下の形式を使用します。

```
[Global] 1
  user = "myusername" 2
  password = "mypassword" 3
  port = "443" 4
  insecure-flag = "1" 5
  datacenters = "us-east, us-west" 6

[VirtualCenter "10.10.0.2"] 7
  user = "myvCenterusername"
  password = "password"

[VirtualCenter "10.10.0.3"]
  port = "448"
  insecure-flag = "0"

[Workspace] 8
  server = "10.10.0.2" 9
  datacenter = "mydatacenter"
  folder = "path/to/vms" 10
  default-datastore = "shared-datastore" 11
  resourcepool-path = "myresourcepoolpath" 12

[Disk]
  scsicontrollertype = pvscsi 13

[Network]
  public-network = "VM Network" 14
```

- 1 **[Global]** セクションに設定されるプロパティは、個別の **[VirtualCenter]** セクションの設定で上書きされない限り、すべての指定される vcenter で使用されます。
- 2 vSphere クラウドプロバイダーの vCenter ユーザー名。
- 3 指定されたユーザーの vCenter パスワード。
- 4 オプション。vCenter サーバーのポート番号。デフォルトはポート **443** になります。
- 5 vCenter が自己署名証明書を使用している場合は **1** に設定します。
- 6 ノード仮想マシンがデプロイされているデータセンターの名前。
- 7 この Virtual Center の特定の **[Global]** プロパティを上書きします。使用できる設定は **[Port]**、**[user]**、**[insecure-flag]**、**[datacenters]** です。指定されない設定は **[Global]** セクションからプルされます。
- 8 各種の vSphere Cloud Provider 機能で使用されるプロパティを設定します。たとえば、動的プロビジョニング、ストレージプロファイルベースのボリュームプロビジョニングなどがこれに含まれます。
- 9 Cloud Provider が通信する vCenter サーバーの IP アドレスまたは FQDN。
- 10 ノード VM の VM ディレクトリーへのパス。
- 11 ストレージクラスまたは動的プロビジョニングを使ったボリュームのプロビジョニングに使用されるデータストアの名前に設定されます。OpenShift Container Platform 3.9 よりも前のバージョンでは、データストアがストレージディレクトリーにあるか、またはデータストアクラスターのメンバーである場合には、完全パスが必要でした。
- 12 オプション。ストレージプロファイルベースのボリュームプロビジョニングのダミー VM が作成される必要のあるリソースプールのパスに設定されます。
- 13 VMDK が VM に割り当てられる際に設定される SCSI コントローラーのタイプ。
- 14 vSphere がノードにアクセスするために使用されるネットワークポートグループに設定されます。これはデフォルトで VM ネットワークと呼ばれます。これは Kubernetes に登録されているノードホストの ExternalIP です。

3. OpenShift Container Platform サービスを再起動します。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

### 25.2.2.2. vSphere 向けのノードホストの手動設定

全ノードホスト上で以下を実行します。

#### 手順

vSphere について OpenShift Container Platform ノードを設定するには、以下を実行します。

1. 適切な [ノード設定マップ](#) を編集して、**kubeletArguments** セクションの内容を更新します。

```
kubeletArguments:
  cloud-provider:
    - "vsphere"
  cloud-config:
    - "/etc/origin/cloudprovider/vsphere.conf"
```



### 重要

クラウドプロバイダーの統合を正常に機能させるため、**nodeName** は vSphere の VM 名と一致していなければなりません。また、この名前は RFC1123 に準拠している必要もあります。

2. すべてのノードで OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

### 25.2.2.3. 設定変更の適用

マスターおよびノードのすべてのホストで OpenShift Container Platform サービスを起動または再起動し、設定の変更を適用します。 [OpenShift Container Platform サービスの再起動](#) を参照してください。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```



### 注記

Kubernetes アーキテクチャーでは、クラウドプロバイダーからの信頼性のあるエンドポイントが必要です。クラウドプロバイダーが停止している場合、kubelet は OpenShift Container Platform が再起動されないようにします。基礎となるクラウドプロバイダーのエンドポイントに信頼性がない場合は、クラウドプロバイダー統合を使用してクラスターをインストールしないでください。クラスターをベアメタル環境の場合のようにインストールします。インストール済みのクラスターで、クラウドプロバイダー統合をオンまたはオフに切り替えることは推奨されていません。ただし、そのシナリオが避けられない場合は、以下のプロセスを実行してください。

クラウドプロバイダーを不使用から使用に切り替えるとエラーメッセージが表示されます。クラウドプロバイダーを追加すると、ノードが **hostname** を **externalID** として使用する (クラウドプロバイダーが使用されていなかった場合のケース) 代わりに、クラウドプロバイダーの **instance-id** (クラウドプロバイダーによって指定される) の使用に切り替えるため、ノードの削除が試みられます。この問題を解決するには、以下を実行します。

1. CLI にクラスター管理者としてログインします。
2. 既存のノードラベルをチェックし、これらをバックアップします。

```
$ oc describe node <node_name> | grep -Poz '(?s)Labels.*\n.*(?:=Taints)'
```

3. ノードを削除します。

```
$ oc delete node <node_name>
```

4. 各ノードホストで OpenShift Container Platform サービスを再起動します。

```
# systemctl restart atomic-openshift-node
```

5. 以前に使用していた [各ノードのラベル](#) を再度追加します。

## 25.3. OPENSIFT CONTAINER PLATFORM が VSPHERE ストレージを使用するように設定する

OpenShift Container Platform では、VMWare vSphere の仮想マシンディスク (VMDK: Virtual Machine Disk) ボリュームがサポートされます。[VMware vSphere](#) を使用して、OpenShift Container Platform クラスタに [永続ストレージ](#) をプロビジョニングできます。これには、Kubernetes と VMWare vSphere についてのある程度の理解があることが前提となります。

OpenShift Container Platform は vSphere にディスクを作成し、ディスクを適切なインスタンスに割り当てます。

OpenShift Container Platform の [永続ボリューム \(PV\)](#) フレームワークを使用すると、管理者がクラスタに永続ストレージをプロビジョニングできるようになるだけでなく、ユーザーが、基盤となるインフラストラクチャーに精通していなくてもこれらのリソースを要求できるようになります。VMWare vSphere VMDK ボリュームは、[動的にプロビジョニング](#) できます。

永続ボリュームは、単一のプロジェクトまたは namespace にバインドされず、OpenShift Container Platform クラスタ全体で共有できます。ただし、[永続ボリューム要求](#) は、プロジェクトまたは namespace に固有で、ユーザーによる要求が可能です。



### 重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

### 前提条件

vSphere を使用して永続ボリュームを作成する前に、OpenShift Container Platform クラスタが以下の要件を満たしていることを確認してください。

- 最初に OpenShift Container Platform を [vSphere 用に設定](#) する必要があります。
- インフラストラクチャー内の各ノードホストは、vSphere 仮想マシン名に一致する必要があります。
- それぞれのノードホストは、同じリソースグループに属している必要があります。

### 25.3.1. VMware vSphere ボリュームの動的プロビジョニング

VMware vSphere ボリュームの動的プロビジョニングが推奨されるプロビジョニング方法です。

1. クラスタのプロビジョニング時に **openshift\_cloudprovider\_kind=vsphere** および **openshift\_vsphere\_\*** 変数を指定しない場合、以下の **StorageClass** を手動で作成し、**vsphere-volume** プロビジョナーを使用できるようにする必要があります。

```
$ oc get --export storageclass vsphere-standard -o yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
```

```

name: "vsphere-standard" ❶
provisioner: kubernetes.io/vsphere-volume ❷
parameters:
  diskformat: thin ❸
  datastore: "YourvSphereDatastoreName" ❹
reclaimPolicy: Delete

```

- ❶ StorageClass の名前。
- ❷ ストレージプロビジョナーのタイプ。 **vsphere-volume** を指定します。
- ❸ ディスクのタイプ。 **zeroedthick** または **thin** のいずれかを指定します。
- ❹ ディスクが作成されるソースデータストア。

2. 以前の手順で示した StorageClass を使用して PV を要求した後に、OpenShift Container Platform は vSphere インフラストラクチャーに VMDK ディスクを自動的に作成します。ディスクが作成されていることを確認するには、vSphere でデータストアブラウザーを使用します。



### 注記

vSphere ボリュームディスクは **ReadWriteOnce** アクセスモードで、1つのノードで読み取り/書き込み可能な状態でボリュームをマウントできます。詳細情報は、[アーキテクチャーガイドのアクセスモードのセクション](#) を参照してください。

## 25.3.2. VMware vSphere ボリュームの静的プロビジョニング

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。OpenShift Container Platform が [vSphere 用に設定されている](#) ことを確認した後、OpenShift Container Platform と vSphere に必要になるのは、VM フォルダーパス、ファイルシステムタイプ、および **PersistentVolume** API のみです。

### 25.3.2.1. PersistentVolume の作成

1. `vsphere-pv.yaml` などの PV オブジェクト定義を定義します。

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 ❶
spec:
  capacity:
    storage: 2Gi ❷
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  vsphereVolume: ❸
    volumePath: "[datastore1] volumes/myDisk" ❹
    fsType: ext4 ❺

```

- ❶ ボリュームの名前。これを使用して、[PV Claim \(永続ボリューム要求\)](#) で、または Pod からボリュームを識別する必要がなくなります。

ら、ボリュームを識別する必要があります。

- 2 このボリュームに割り当てられるストレージの量。
- 3 使用されるボリュームタイプです。この例では、**vsphereVolume** を使用し、ラベルは、vSphere VMDK ボリュームを Pod にマウントするために使用されます。ラベルは vSphere VMDK ボリュームを Pod にマウントするために使用されます。ボリュームの内容はアンマウントされても保持されます。このボリュームタイプは、VMFS データストアと VSAN データストアの両方がサポートされます。
- 4 使用する既存の VMDK ボリューム。ボリューム定義で示されるように、データストア名は角かっこ ([ ]) で囲む必要があります。
- 5 マウントするファイルシステムタイプです。**ext4**、**xfs**、または他のファイルシステムなどが例となります。



### 重要

ボリュームをフォーマットしてプロビジョニングした後に **fsType** パラメーターの値を変更すると、データ損失や Pod にエラーが発生する可能性があります。

2. PV を作成します。

```
$ oc create -f vsphere-pv.yaml
persistentvolume "pv0001" created
```

3. PV が作成されたことを確認します。

```
$ oc get pv
NAME LABELS CAPACITY ACCESSMODES STATUS CLAIM REASON AGE
pv0001 <none> 2Gi RWO Available 2s
```

これで、[PVC \(永続ボリューム要求\)](#) を使用して[ストレージを要求](#)し、永続ボリュームを活用できるようになります。



### 重要

PV Claim (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとする、Pod にエラーが発生します。

#### 25.3.2.2. VMware vSphere ボリュームのフォーマット

OpenShift Container Platform は、ボリュームをマウントしてコンテナに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

OpenShift Container Platform は初回の使用前にフォーマットするため、フォーマットされていない vSphere ボリュームを PV として使用できます。

## 25.4. VSPHERE の OPENSIFT CONTAINER PLATFORM レジストリーの設定

### 25.4.1. Ansible を使用した vSphere の OpenShift Container Platform レジストリーの設定

#### 手順

レジストリーで vSphere ボリュームを使用できるように Ansible インベントリーを設定するには、以下を実行します。

```
[OSEv3:vars]
# vSphere Provider Configuration
openshift_hosted_registry_storage_kind=vsphere ❶
openshift_hosted_registry_storage_access_modes=["ReadWriteOnce"] ❷
openshift_hosted_registry_storage_annotations=["volume.beta.kubernetes.io/storage-provisioner:
kubernetes.io/vsphere-volume"] ❸
openshift_hosted_registry_replicas=1 ❹
```

- ❶ ストレージタイプ。
- ❷ vSphere ボリュームは **RWO** のみをサポートします。
- ❸ ボリュームのアノテーション。
- ❹ 設定するレプリカ数



#### 注記

上記の設定ファイルでは括弧が必要です。

### 25.4.2. OpenShift Container Platform レジストリーの動的にプロビジョニングされるストレージ

vSphere ボリュームストレージを使用するには、レジストリーの設定ファイルを編集してレジストリー Pod にマウントします。

#### 手順

1. vSphere ボリュームから新規設定ファイルを作成します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: vsphere-registry-storage
  annotations:
    volume.beta.kubernetes.io/storage-class: vsphere-standard
spec:
  accessModes:
    - ReadWriteOnce
```

```
resources:
  requests:
    storage: 30Gi
```

- OpenShift Container Platform でファイルを作成します。

```
$ oc create -f pvc-registry.yaml
```

- 新規 PVC を使用するようにボリューム設定を更新します。

```
$ oc set volume dc docker-registry --add --name=registry-storage -t \
pvc --claim-name=vsphere-registry-storage --overwrite
```

- 更新された設定を読み取るためにレジストリーを再デプロイします。

```
$ oc rollout latest docker-registry -n default
```

- ボリュームが割り当てられていることを確認します。

```
$ oc set volume dc docker-registry -n default
```

### 25.4.3. OpenShift Container Platform レジストリーの手動でプロビジョニングされるストレージ

以下のコマンドを手動で実行することにより、**StorageClass** が利用できないか、または使用されていない場合にレジストリーのストレージを作成するために使用されるストレージが手動で作成されます。

```
# VMFS
cd /vmfs/volumes/datastore1/
mkdir kubevols # Not needed but good hygiene

# VSAN
cd /vmfs/volumes/vsanDatastore/
/usr/lib/vmware/osfs/bin/osfs-mkdir kubevols # Needed

cd kubevols

vmkfstools -c 25G registry.vmdk
```

### 25.4.4. Red Hat OpenShift Container Storage について

Red Hat OpenShift Container Storage (RHOCS) は、インハウスまたはハイブリッドクラウドのいずれの場合でも、OpenShift Container Platform のすべてに対応する永続ストレージのプロバイダーです。Red Hat ストレージソリューションとして、RHOCS は OpenShift Container Platform にインストール (コンバージド) されているか、または OpenShift Container Platform と共にインストール (インデペンデント) されているかを問わず、デプロイメント、管理およびモニタリングを目的として OpenShift Container Platform に完全に統合されます。OpenShift Container Storage は単一のアベイラビリティゾーンまたはノードに制限されないため、停止した場合にも存続できる可能性があります。RHOCS の詳細の使用方法については、[RHOCS3.11 Deployment Guide](#) を参照してください。

## 25.5. 永続ボリュームのバックアップ

OpenShift Container Platform は、自由にクラスターないのノードにあるボリュームをアタッチしたり、アタッチ解除できるように、**個別の永続ディスク**として新規ボリュームをプロビジョニングします。そのため、**スナップショットを使用するボリュームはバックアップできません**。

以下の方法で、PV のバックアップを作成します。

1. PV を使用してアプリケーションを停止します。
2. 永続ディスクのクローンを作成します。
3. アプリケーションを再起動します。
4. クローンしたディスクのバックアップを作成します。
5. クローンしたディスクを削除します。

## 第26章 ローカルボリュームの設定

### 26.1. 概要

OpenShift Container Platform は、アプリケーションデータの **ローカルボリューム** にアクセスするように設定できます。

ローカルのボリュームとは、ローカルにマウントされたファイルシステムを表す永続ボリューム (PV) です。これには raw ブロックデバイスも含まれます。raw デバイスは、物理デバイスに、さらにダイレクトなルートを提供し、アプリケーションがその物理デバイスに対する I/O 操作のタイミングをより詳細に制御できるようにします。このように、raw デバイスは、通常独自のキャッシングを行うデータベース管理システムなどの複雑なアプリケーションに適しています。ローカルボリュームにはユニークな機能が少し含まれています。ローカルボリューム PV を使用する Pod は、ローカルボリュームがマウントされるノードでスケジューリングされます。

また、ローカルボリュームには、ローカルにマウントされたデバイスに PV を自動作成するプロビジョナーが含まれます。現時点で、このプロビジョナーは事前設定されたディレクトリーのみをスキャンします。このプロビジョナーは、ボリュームを動的にプロビジョニングする機能はありませんが、今後のリリースで実装される可能性があります。

ローカルボリュームのプロビジョナーを使用すると、ローカルストレージを OpenShift Container Platform 内で使用することができます。ローカルボリュームのプロビジョナーは以下に対応しています。

- ボリューム
- PV



#### 重要

ローカルボリュームは、テクノロジープレビュー機能のみとなっています。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポートについての詳細は、<https://access.redhat.com/support/offerings/techpreview/> を参照してください。

### 26.2. ローカルボリュームのマウント



#### 注記

すべてのローカルボリュームは、OpenShift Container Platform によって PV として使用される前に手動でマウントする必要があります。

ローカルボリュームをマウントします。

1. すべてのボリュームは、`/mnt/local-storage/<storage-class-name>/<volume>`パスにマウントしてください。管理者は、必要に応じてディスクパーティションまたは LVM などの方法でローカルデバイスを作成し、これらのデバイスに適切なファイルシステムを作成して、スクリプトまたは `/etc/fstab` エントリーを使用してそれらをマウントします。

```
# device name # mount point # FS # options # extra
```

```

/dev/sdb1 /mnt/local-storage/ssd/disk1 ext4 defaults 1 2
/dev/sdb2 /mnt/local-storage/ssd/disk2 ext4 defaults 1 2
/dev/sdb3 /mnt/local-storage/ssd/disk3 ext4 defaults 1 2
/dev/sdc1 /mnt/local-storage/hdd/disk1 ext4 defaults 1 2
/dev/sdc2 /mnt/local-storage/hdd/disk2 ext4 defaults 1 2

```

2. コンテナ内で実行中のプロセスからすべてのボリュームにアクセスできるようにします。これを可能にするには、以下のようにマウントされたファイルシステムのラベルを変更してください。

```

---
$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-storage/
---

```

## 26.3. ローカルプロビジョナーの設定

OpenShift Container Platform は、ローカルデバイス用に PV を作成する場合や、再利用の有効化に使用されなくなり PV を消去する場合に、外部のプロビジョナーを使用します。



### 注記

- ローカルボリュームのプロビジョナーは大半のプロビジョナーとは異なり、動的なプロビジョニングに対応していません。
- ローカルボリュームのプロビジョナーを使用する場合には、管理者は、各ノードでローカルボリュームを事前設定して、`discovery` ディレクトリーの下にマウントする必要があります。その後プロビジョナーは各ボリュームについて PV の作成とクリーンアップを実行してボリュームを管理します。

ローカルプロビジョナーを設定します。

1. ConfigMap を使用して外部プロビジョナーが、ストレージクラスとディレクトリーを関連付けられるように設定します。この設定は、プロビジョナーがデプロイされる前に作成する必要があります。以下に例を示します。

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: local-volume-config
data:
  storageClassMap: |
    local-ssd: ①
      hostDir: /mnt/local-storage/ssd ②
      mountDir: /mnt/local-storage/ssd ③
    local-hdd:
      hostDir: /mnt/local-storage/hdd
      mountDir: /mnt/local-storage/hdd

```

- ① ストレージクラス名
- ② ホスト上のディレクトリーへのパス。/mnt/local-storage のサブディレクトリーでなければなりません。

- 3 プロビジョナー Pod のディレクトリーへのパス。ホストで使用されているディレクトリー構造と同じ構造を使用することを推奨します。今回の例では、**mountDir** は省略可能です。
2. (オプション) ローカルボリュームのプロビジョナーおよびその設定用にスタンドアロンの namespace を作成します。例: **oc new-project local-storage**

上記の設定により、プロビジョナーは以下を作成します。

- **/mnt/local-storage/ssd** ディレクトリーにマウントされる全サブディレクトリーに対して、**local-ssd** のストレージクラスを指定した PV1つ
- **/mnt/local-storage/hdd** ディレクトリーにマウントされる全サブディレクトリーに対して、**local-hdd** のストレージクラスを指定した PV1つ

## 26.4. ローカルプロビジョナーのデプロイ



### 注記

プロビジョナーを起動する前に、すべてのローカルデバイスをマウントし、ストレージクラスとそれらのディレクトリーと共に ConfigMap を作成します。

ローカルプロビジョナーをデプロイします。

1. **local-storage-provisioner-template.yaml** ファイルからローカルプロビジョナーをインストールします。
2. サービスアカウントを作成して、root ユーザーでの Pod 実行、hostPath ボリュームの使用をはじめ、SELinux コンテキストでローカルボリュームの監視、管理、消去ができるようにします。

```
$ oc create serviceaccount local-storage-admin
$ oc adm policy add-scc-to-user privileged -z local-storage-admin
```

プロビジョナー Pod で任意の Pod が作成したローカルボリュームのコンテンツを削除できるようにするには、root 権限と任意の SELinux コンテキストが必要です。ホスト上の **/mnt/local-storage** パスにアクセスするには **hostPath** が必要です。

3. テンプレートをインストールします。

```
$ oc create -f https://raw.githubusercontent.com/openshift/origin/release-3.11/examples/storage-examples/local-examples/local-storage-provisioner-template.yaml
```

4. **CONFIGMAP**、**SERVICE\_ACCOUNT**、**NAMESPACE** および **PROVISIONER\_IMAGE** パラメーターに値を指定して、テンプレートをインスタンス化します。

```
$ oc new-app -p CONFIGMAP=local-volume-config \
-p SERVICE_ACCOUNT=local-storage-admin \
-p NAMESPACE=local-storage \
-p PROVISIONER_IMAGE=registry.redhat.io/openshift3/local-storage-provisioner:v3.11 \
```

```
1 local-storage-provisioner
```

- 1 **v3.11** など、お使いの OpenShift Container Platform バージョン番号を指定します。

5. 必要なストレージクラスを追加します。

```
$ oc create -f ./storage-class-ssd.yaml
$ oc create -f ./storage-class-hdd.yaml
```

以下に例を示します。

#### storage-class-ssd.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-ssd
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

#### storage-class-hdd.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-hdd
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

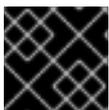
その他の設定オプションについては、[ローカルストレージプロビジョナーのテンプレート](#)を参照してください。このテンプレートは、すべてのノードで Pod を実行する DaemonSet を作成します。Pod は ConfigMap に指定されるディレクトリーを監視し、それらの PV を自動的に作成します。

このプロビジョナーは、PV が開放されると、変更されたディレクトリーからすべてのデータが削除されるので、Root パーミッションを使用して実行します。

## 26.5. 新規デバイスの追加

新規デバイスの追加は半自動で実行されます。プロビジョナーは設定されたディレクトリーで新規マウントについて定期的にチェックします。管理者は、以下のように、SELinux ラベルを適用して、新しいサブディレクトリーの作成、デバイスのマウント、Pod によるデバイスの使用許可を行う必要があります。

```
$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-storage/
```



### 重要

上記のいずれかの操作を省くと、適切な PV が作成されなくなることがあります。

## 26.6. RAW ブロックデバイスの設定

ローカルのボリュームプロビジョナーを使用すると、raw ブロックデバイスを静的にプロビジョニングできます。この機能はデフォルトでは無効になっており、追加の設定が必要です。

Raw ブロックデバイスを設定するには以下を行います。

1. 全マスターで、**BlockVolume** 機能ゲートを有効化します。全マスターでマスター設定ファイル

を編集または作成 (デフォルトは `/etc/origin/master/master-config.yaml`) して、**apiServerArguments** および **controllerArguments** セクションに、**BlockVolume=true** を追加します。

```
apiServerArguments:
  feature-gates:
  - BlockVolume=true
  ...

controllerArguments:
  feature-gates:
  - BlockVolume=true
  ...
```

2. ノード設定 ConfigMap を編集して、全ノードで機能ゲートを有効化します。

```
$ oc edit configmap node-config-compute --namespace openshift-node
$ oc edit configmap node-config-master --namespace openshift-node
$ oc edit configmap node-config-infra --namespace openshift-node
```

3. 以下のように、すべての ConfigMaps に、**kubeletArguments** の機能ゲートアレイに **BlockVolume=true** が含まれていることを確認します。

#### node configmap feature-gates setting

```
kubeletArguments:
  feature-gates:
  -
  RotateKubeletClientCertificate=true,RotateKubeletServerCertificate=true,BlockVolume=true
```

4. マスターを再起動します。ノードは、設定の変更後に自動的に再起動されます。これは数分かかる可能性があります。

### 26.6.1. raw ブロックデバイスの準備

プロビジョナーを起動する前に、Pod が使用できるすべての raw ブロックデバイスを `/mnt/local-storage/<storage class>` ディレクトリー構造にリンクします。たとえば、`/dev/dm-36` のディレクトリーを利用できるようにします。

1. `/mnt/local-storage` に、デバイスのストレージクラスのディレクトリーを作成します。

```
$ mkdir -p /mnt/local-storage/block-devices
```

2. このデバイスを参照するシンボリックリンクを作成します。

```
$ ln -s /dev/dm-36 dm-uuid-LVM-1234
```



#### 注記

名前の競合が発生するのを回避するには、`/dev/disk/by-uuid` または `/dev/disk/by-id` ディレクトリーからのリンクと、シンボリックリンクに同じ名前を使用します。

3. プロビジョナー設定用の ConfigMap を作成するか、更新します。

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: local-volume-config
data:
  storageClassMap: |
    block-devices: ❶
    hostDir: /mnt/local-storage/block-devices ❷
    mountDir: /mnt/local-storage/block-devices ❸

```

- ❶ ストレージクラス名
- ❷ ホスト上のディレクトリーへのパス。/mnt/local-storage のサブディレクトリーでなければなりません。
- ❸ プロビジョナー Pod のディレクトリーへのパス。ホストが使用するディレクトリー構造を使用する場合 (推奨) には、**mountDir** パラメーターを省略します。

4. デバイスと、/mnt/local-storage/ の SELinux ラベルを変更します。

```

$ chcon -R unconfined_u:object_r:svirt_sandbox_file_t:s0 /mnt/local-storage/
$ chcon unconfined_u:object_r:svirt_sandbox_file_t:s0 /dev/dm-36

```

5. Raw ブロックデバイスのストレージクラスを作成します。

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: block-devices
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer

```

プロビジョナーは、このブロックデバイス /dev/dm-36 を使用する準備ができ、PV としてプロビジョニングします。

## 26.6.2. raw ブロックデバイスプロビジョナーのデプロイ

Raw ブロックデバイスへのプロビジョナーのデプロイは、ローカルボリュームへのプロビジョナーのデプロイに似ていますが、2点相違点があります。

1. プロビジョナーは特権付きのコンテナで実行する必要があります。
2. プロビジョナーは、ホストから /dev のファイルシステムにアクセスできる必要があります。

Raw ブロックデバイスにプロビジョナーをデプロイします。

1. [local-storage-provisioner-template.yaml](#) ファイルからテンプレートをダウンロードします。
2. テンプレートを編集します。
  - a. コンテナ仕様の **securityContext** の **privileged** 属性を **true** に設定します。

```

...
  containers:
...
    name: provisioner
...
    securityContext:
      privileged: true
...

```

- b. **hostPath** を使用して、コンテナにホストの **/dev/** ファイルシステムをマウントします。

```

...
  containers:
...
    name: provisioner
...
    volumeMounts:
      - mountPath: /dev
        name: dev
...
  volumes:
    - hostPath:
        path: /dev
        name: dev
...

```

3. 変更済みの YAML ファイルからテンプレートを作成します。

```
$ oc create -f local-storage-provisioner-template.yaml
```

4. プロビジョナーを起動します。

```
$ oc new-app -p CONFIGMAP=local-volume-config \
-p SERVICE_ACCOUNT=local-storage-admin \
-p NAMESPACE=local-storage \
-p
PROVISIONER_IMAGE=registry.redhat.io/openshift3/local-storage-provisioner:v3.11 \
local-storage-provisioner
```

### 26.6.3. raw ブロックデバイスの永続ボリュームの使用

以下のように、Pod で Raw ブロックデバイスを使用するには、**volumeMode**: を **Block** に、**storageClassName** を **block-devices** に設定して、永続ボリューム要求 (PVC) を作成します。

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  storageClassName: block-devices
  accessModes:
    - ReadWriteOnce
  volumeMode: Block

```

```
resources:  
  requests:  
    storage: 1Gi
```

## raw ブロックデバイス PVC を使用する Pod

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: busybox-test  
  labels:  
    name: busybox-test  
spec:  
  restartPolicy: Never  
  containers:  
  - resources:  
    limits :  
      cpu: 0.5  
    image: gcr.io/google_containers/busybox  
    command:  
    - "/bin/sh"  
    - "-c"  
    - "while true; do date; sleep 1; done"  
    name: busybox  
    volumeDevices:  
    - name: vol  
      devicePath: /dev/xvda  
  volumes:  
  - name: vol  
    persistentVolumeClaim:  
      claimName: block-pvc
```



### 注記

ボリュームは、Pod にマウントされていませんが、`/dev/xvda` Raw ブロックデバイスとして公開されます。

## 第27章 永続ストレージの設定

### 27.1. 概要

Kubernetes の [永続ボリューム](#) フレームワークにより、お使いの環境で利用可能なネットワークストレージを使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これは、アプリケーションのニーズに応じて初回 OpenShift Container Platform インストールの完了後に行うことができ、ユーザーは基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようになります。

このトピックでは、以下のサポートされるボリュームプラグインを使って永続ボリュームを OpenShift Container Platform で設定する方法を説明します。

- [NFS](#)
- [GlusterFS](#)
- [OpenStack Cinder](#)
- [Ceph RBD](#)
- [AWS Elastic Block Store \(EBS\)](#)
- [GCE Persistent Disk](#)
- [iSCSI](#)
- [ファイバーチャネル](#)
- [Azure Disk](#)
- [Azure File](#)
- [FlexVolume](#)
- [VMware vSphere](#)
- [Container Storage Interface \(CSI\)](#)
- [動的プロビジョニングとストレージクラスの作成](#)
- [ボリュームのセキュリティー](#)
- [セレクターとラベルによるボリュームのバインディング](#)

### 27.2. NFS を使用した永続ストレージ

#### 27.2.1. 概要

OpenShift Container Platform クラスターは、NFS を使用している [永続ストレージ](#) を使ってプロビジョニングすることが可能です。永続ボリューム (PV) および Persistent Volume Claim (永続ボリューム要求、PVC) は、プロジェクト全体でボリュームを共有するための便利な方法を提供します。PV 定義に含まれる NFS に固有の情報は、Pod 定義で直接定義することも可能ですが、この方法の場合にはボリュームが一意的なクラスターリソースとして作成されられないため、ボリュームが競合の影響を受けやすくなります。

このトピックでは、NFS 永続ストレージタイプの具体的な使用方法について説明します。OpenShift Container Platform と NFS についてある程度理解していることを前提とします。OpenShift Container Platform 永続ボリューム (PV) の一般的なフレームワークについての詳細は、[永続ストレージ](#) の概念に関するトピックを参照してください。

## 27.2.2. プロビジョニング

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。NFS ボリュームをプロビジョニングするには、NFS サーバーの一覧とエクスポートパスのみが必要です。

最初に、PV のオブジェクト定義を作成します。

### 例27.1 NFS を使用した PV オブジェクト定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 ①
spec:
  capacity:
    storage: 5Gi ②
  accessModes:
  - ReadWriteOnce ③
  nfs: ④
    path: /tmp ⑤
    server: 172.17.0.2 ⑥
  persistentVolumeReclaimPolicy: Retain ⑦
```

- ① ボリュームの名前。これは、各種の `oc <command> pod` コマンドの PV アイデンティティです。
- ② このボリュームに割り当てられるストレージの量。
- ③ これはボリュームへのアクセスの制御に関連するようには見えますが、実際はラベルの場合と同様に、PVC を PV に一致させるために使用されます。現時点では、**accessModes** に基づくアクセスルールは適用されていません。
- ④ 使用されているボリュームタイプ。この場合は `nfs` プラグインです。
- ⑤ NFS サーバーがエクスポートしているパス。
- ⑥ NFS サーバーのホスト名または IP アドレス
- ⑦ PV の回収ポリシー。ボリュームが要求から解放されるタイミングでボリュームで実行されることを定義します。[リソースの回収](#) を参照してください。



### 注記

各 NFS ボリュームは、クラスター内のスケジュール可能なすべてのノードによってマウント可能でなければなりません。

定義をファイル (`nfs-pv.yaml` など) に保存し、PV を作成します。

```
$ oc create -f nfs-pv.yaml
persistentvolume "pv0001" created
```

PV が作成されたことを確認します。

```
# oc get pv
NAME                LABELS  CAPACITY  ACCESSMODES  STATUS   CLAIM   REASON
AGE
pv0001              <none>  5368709120  RWO          Available             31s
```

以下の手順で PVC が作成されます。これは新規 PV にバインドされます。

### 例27.2 PVC オブジェクト定義

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
    - ReadWriteOnce ①
  resources:
    requests:
      storage: 1Gi ②
```

- ① PV について前述されているように、**accessModes** はセキュリティーを実施するのではなく、PV を PVC と一致させるラベルとして機能します。
- ② この要求は 1Gi 以上の容量を提供する PV を検索します。

定義をファイル (`nfs-claim.yaml` など) に保存し、PVC を作成します。

```
# oc create -f nfs-claim.yaml
```

### 27.2.3. ディスククォータの実施

ディスクパーティションを使用して、ディスククォータとサイズ制限を実施することができます。それぞれのパーティションを独自のエクスポートとすることができ、それぞれのエクスポートは1つのPVになります。それぞれのエクスポートは1つのPVになります。OpenShift Container Platform は PV に固有の名前を適用しますが、NFS ボリュームのサーバーとパスの一意性については管理者に委ねられています。

この方法でクォータを実施すると、開発者は永続ストレージを具体的な量 (10Gi など) で要求することができます、同等かそれ以上の容量の対応するボリュームに一致させることができます。

### 27.2.4. NFS ボリュームのセキュリティー

このセクションでは、一致するパーミッションや SELinux の考慮点を含む、NFS ボリュームのセキュリティについて説明します。ユーザーは、POSIX パーミッションやプロセス UID、補助グループおよび SELinux の基礎的な点を理解している必要があります。



### 注記

NFS ボリュームを実装する前に [ボリュームのセキュリティ](#) のトピックをすべてお読みください。

開発者は、Pod 定義の **volumes** セクションで、PVC を名前参照するか、または NFS ボリュームのプラグインを直接参照して NFS ストレージを要求します。

NFS サーバーの `/etc/exports` ファイルにはアクセス可能な NFS ディレクトリーが含まれています。ターゲットの NFS ディレクトリーには、POSIX の所有者とグループ ID があります。OpenShift Container Platform NFS プラグインは、同じ POSIX の所有者とエクスポートされる NFS ディレクトリーにあるパーミッションを使って、コンテナの NFS ディレクトリーをマウントします。ただし、コンテナは NFS マウントの所有者と同等の有効な UID では実行されません。これは期待される動作です。

ターゲットの NFS ディレクトリーが NFS サーバーに表示される場合を例に取って見てみましょう。

```
# ls -lZ /opt/nfs -d
drwxrws---. nfsnobody 5555 unconfined_u:object_r:usr_t:s0 /opt/nfs

# id nfsnobody
uid=65534(nfsnobody) gid=65534(nfsnobody) groups=65534(nfsnobody)
```

コンテナは SELinux ラベルと一致している必要があります、ディレクトリーにアクセスするために UID 65534 (nfsnobody 所有者) か、または補助グループの 5555 のいずれかを使って実行する必要があります。



### 注記

ここで、所有者 ID 65534 は一例として使用されています。NFS の `root_squash` が `root` (0) を `nfsnobody` (65534) にマップしても、NFS エクスポートは任意の所有者 ID を持つことができます。所有者 65534 は NFS エクスポートには必要ありません。

#### 27.2.4.1. グループ ID

NFS アクセスに対応する際の推奨される方法として、補助グループを使用することができます (NFS エクスポートのパーミッションを変更するオプションがないことを前提としています)。OpenShift Container Platform の補助グループは共有ストレージに使用されます (例: NFS)。これとは対照的に、Ceph RBD や iSCSI などのブロックストレージは、Pod の **securityContext** で **fsGroup** SCC ストラテジーと **fsGroup** の値を使用します。



### 注記

一般的に、永続ストレージへのアクセスを取得する場合、**ユーザー ID** ではなく補助グループ ID を使用することが推奨されます。補助グループについては [ボリュームのセキュリティ](#) トピックで詳しく説明されています。

上記の [ターゲット NFS ディレクトリーの例](#) で使用したグループ ID は 5555 なので、Pod は、**supplementalGroups** を使用してグループ ID を Pod レベルの **securityContext** 定義の下で定義することができます。以下に例を示します。

```
spec:
  containers:
    - name:
      ...
  securityContext: ❶
  supplementalGroups: [5555] ❷
```

- ❶ **securityContext** は特定のコンテナの下位ではなく、この Pod レベルで定義します。
- ❷ Pod 向けに定義される GID の配列。ここでは、配列に1つの要素があり、追加の GID はコンマで区切られます。

Pod の要件を満たすカスタム SCC が存在しない場合、Pod は **制限付きの** SCC に一致する可能性があります。この SCC では、**supplementalGroups** ストラテジーが **RunAsAny** に設定されています。これは、指定されるグループ ID は範囲のチェックなしに受け入れられることを意味します。

その結果、上記の Pod は受付をパスして起動します。しかし、グループ ID の範囲をチェックすることが望ましい場合は、[Pod のセキュリティとカスタム SCC](#) で説明されているようにカスタム SCC の使用が推奨されます。カスタム SCC は、最小および最大のグループ ID が定義され、グループ ID の範囲チェックが実施され、グループ ID 5555 が許可されるように作成できます。



### 注記

カスタム SCC を使用するには、まずこれを適切なサービスアカウントに追加する必要があります。たとえば、所定のプロジェクトでは、Pod 仕様において別のサービスアカウントが指定されていない限り、**default** のサービスアカウントを使用してください。詳細は、[SCC のユーザー、グループまたはプロジェクトへの追加](#) を参照してください。

#### 27.2.4.2. ユーザー ID

ユーザー ID は、コンテナイメージまたは Pod 定義で定義できます。ユーザー ID に基づいてストレージアクセスを制御する方法については、[ボリュームのセキュリティ](#) のトピックで説明されています。NFS 永続ストレージをセットアップする前に、必ず読んでおいてください。



### 注記

一般的に、永続ストレージへのアクセスを取得する場合、[ユーザー ID](#) ではなく補助グループ ID を使用することが推奨されます。

上記の [ターゲット NFS ディレクトリーの例](#) では、コンテナは UID を 65534 (ここではグループ ID を省略します) に設定する必要があります。したがって以下を Pod 定義に追加することができます。

```
spec:
  containers: ❶
    - name:
      ...
  securityContext:
    runAsUser: 65534 ❷
```

- ❶ Pod には、各コンテナに固有の **securityContext** (ここに表示されている) と、その Pod で定義されたすべてのコンテナに適用される Pod レベルの **securityContext** が含まれます。

## 2 65534 は `nfsnobody` ユーザーです。

デフォルトのプロジェクトと制限付き SCC を前提とする場合は、Pod が要求するユーザー ID 65534 は許可されず、Pod は失敗します。Pod が失敗する理由は以下の通りです。

- 65534 をユーザー ID として要求している。
- ユーザー ID 65534 を許可する SCC を確認するために Pod で利用できるすべての SCC が検査される (実際は SCC のすべてのポリシーがチェックされますが、ここでのフォーカスはユーザー ID になります)。
- 利用可能なすべての SCC は `runAsUser` ストラテジーに `MustRunAsRange` を使用するため、UID の範囲チェックが必要である。
- 65534 は SCC またはプロジェクトのユーザー ID 範囲に含まれていない。

一般に、事前定義された SCC は変更しないことが勧められています。ただし、この状況を改善するには、[ボリュームのセキュリティ](#) のトピックで説明されているようにカスタム SCC を作成することが推奨されます。カスタム SCC は、最小および最大のユーザー ID が定義され、UID 範囲のチェックの実施が設定されており、UID 65534 が許可されるように作成できます。



### 注記

カスタム SCC を使用するには、まずこれを適切なサービスアカウントに追加する必要があります。たとえば、所定のプロジェクトでは、Pod 仕様において別のサービスアカウントが指定されていない限り、`default` のサービスアカウントを使用してください。詳細は、[SCC のユーザー、グループまたはプロジェクトへの追加](#) を参照してください。

#### 27.2.4.3. SELinux



### 注記

SELinux を使用してストレージアクセスを制御する方法についての詳細は、[ボリュームのセキュリティ](#) を参照してください。

デフォルトでは、SELinux では Pod からリモートの NFS サーバーへの書き込みは許可されません。NFS ボリュームは正常にマウントされますが、読み取り専用です。

SELinux を各ノードで有効にした状態で NFS ボリュームへの書き込みを有効にするには、以下を実行します。

```
# setsebool -P virt_use_nfs 1
```

上記の `-P` オプションは、ブール値をリブート間で継続させます。

`virt_use_nfs` ブール値は `docker-selinux` パッケージで定義されます。このブール値が定義されていないことを示すエラーが表示された場合は、このパッケージがインストールされていることを確認してください。

#### 27.2.4.4. エクスポート設定

任意のコンテナユーザーにボリュームの読み取りと書き出しを許可するには、NFS サーバーにエクスポートされる各ボリュームは以下の条件を満たしている必要があります。

- 各エクスポートを以下のように指定します。

```
/<example_fs> *(rw,root_squash)
```

- ファイアウォールは、マウントポイントへのトラフィックを許可するように設定する必要があります。
  - NFSv4 の場合、デフォルトのポート **2049** (nfs) を設定します。

#### NFSv4

```
# iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
```

- NFSv3 の場合、以下の 3 つのポートを設定します。 **2049** (nfs)、 **20048** (mountd)、 **111** (portmapper)。

#### NFSv3

```
# iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
# iptables -I INPUT 1 -p tcp --dport 20048 -j ACCEPT
# iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
```

- NFS エクスポートとディレクトリーは、ターゲット Pod からアクセスできるようにセットアップされる必要があります。この場合、エクスポートをコンテナのプライマリー UID で所有されるように設定するか、または上記の [グループ ID](#) で説明したように、 **supplementalGroups** を使用して Pod にグループアクセスを付与します。Pod のセキュリティーに関する追加情報は、 [ボリュームのセキュリティー](#) のトピックを参照してください。

### 27.2.5. リソースの回収

NFS は OpenShift Container Platform の **再利用可能な** プラグインインターフェイスを実装します。回収タスクは、それぞれの永続ボリュームに設定されるポリシーに基づいて自動プロセスによって処理されます。

デフォルトで、PV は **Retain** に設定されます。

PV への要求が解除される (PVC が削除される) と、PV オブジェクトは再利用できません。代わりに、新規の PV が元のボリュームと同じ基本ボリュームの情報を使って作成されます。

たとえば、管理者は **nfs1** という名前の PV を作成するとします。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs1
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

ユーザーは、**nfs1** にバインドされる **PVC1** を作成します。次にユーザーは **PVC1** を削除し、**nfs1** への要求を解除します。これにより、**nfs1** は **Released** になります。管理者が同じ NFS 共有を利用可能にする必要がある場合には、同じ NFS サーバー情報を使って新規 PV を作成する必要があります。この場合、PV の名前は元の名前とは異なる名前にします。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs2
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

元の PV を削除して、PV を同じ名前で再作成することは推奨されません。PV のステータスを **Released** から **Available** に手動で変更しようとする、エラーが発生し、データが失われる可能性があります。

### 27.2.6. 自動化

クラスターは、NFS を使用している永続ストレージを使って以下の方法でプロビジョニングすることができます。

- ディスクパーティションを使って [ストレージクォータを実施する](#)。
- 要求のあるプロジェクトに [ボリュームを制限](#) してセキュリティーを実施する。
- [破棄されたリソースの回収](#) を各 PV に設定する。

スクリプトを使用して前述のタスクを自動化する方法は多数あります。OpenShift Container Platform 3.11 リリースに関連付けられた [Ansible Playbook のサンプル](#) を使用して、開始することができます。

### 27.2.7. その他の設定とトラブルシューティング

適切なエクスポートとセキュリティーマッピングを行うため、使用している NFS のバージョンおよびその設定方法に応じて追加の設定が必要になることがあります。以下は例になります。

NFSv4 のマウントにすべてのファイルの所有者が **nobody:nobody** と誤って表示される。

- NFS の ID マッピング設定 (/etc/idmapd.conf) に原因がある可能性が高い。
- [この Red Hat ソリューション](#) を参照してください。

NFSv4 の ID マッピングが無効になっている	<ul style="list-style-type: none"><li>● NFS サーバーで以下を実行します。<pre># echo 'Y' &gt; /sys/module/nfsd/parameters/nfs4_disable_idmapping</pre></li><li>● NFS クライアントで、以下を実行します。<pre># echo 'Y' &gt; /sys/module/nfs/parameters/nfs4_disable_idmapping</pre></li></ul>
---------------------------	---

## 27.3. RED HAT GLUSTER STORAGE を使用する永続ストレージ

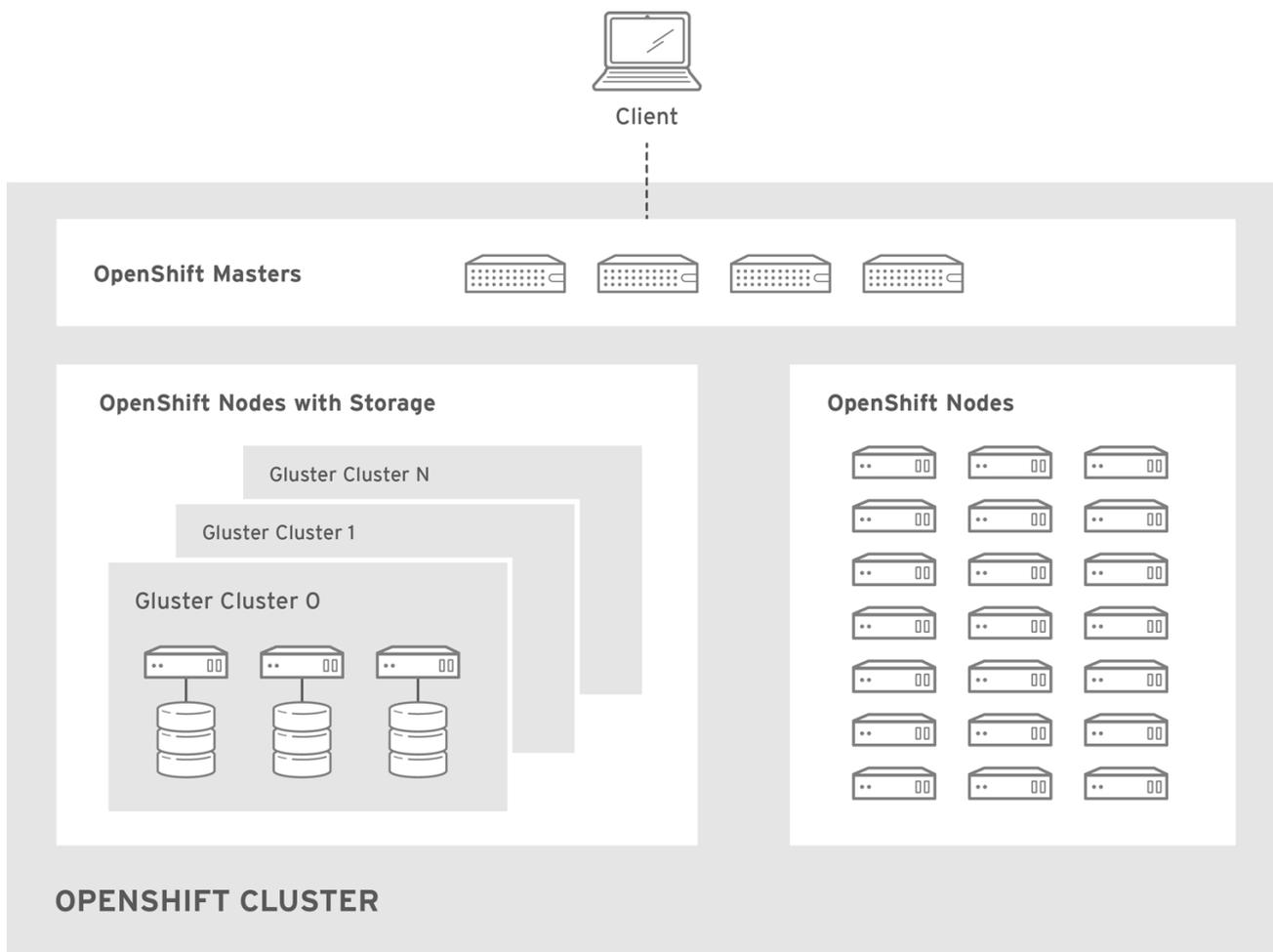
### 27.3.1. 概要

Red Hat Gluster Storage は、OpenShift Container Platform の永続ストレージおよび動的プロビジョニングを提供するように設定できます。OpenShift Container Platform 内のコンテナ化ストレージ (コンバージドモード) と、独自のノードでコンテナ化されていないノード (インデペンデントモード) の両方を使用することができます。

#### 27.3.1.1. コンバージドモード

接続モードの場合、Red Hat Gluster Storage は、Container-Native Storage を使って、コンテナ化されたディレクトリーを OpenShift Container Platform ノードで実行します。それにより、コンピュートおよびストレージインスタンスをスケジュールでき、同じハードウェアのセットから実行することができます。

図27.1 アーキテクチャー: コンバージドモード



OPENSIFT\_412816\_0716

コンバージドモードは Red Hat Gluster Storage 3.4 で利用できます。詳細は、[Container-Native Storage for OpenShift Container Platform](#) を参照してください。

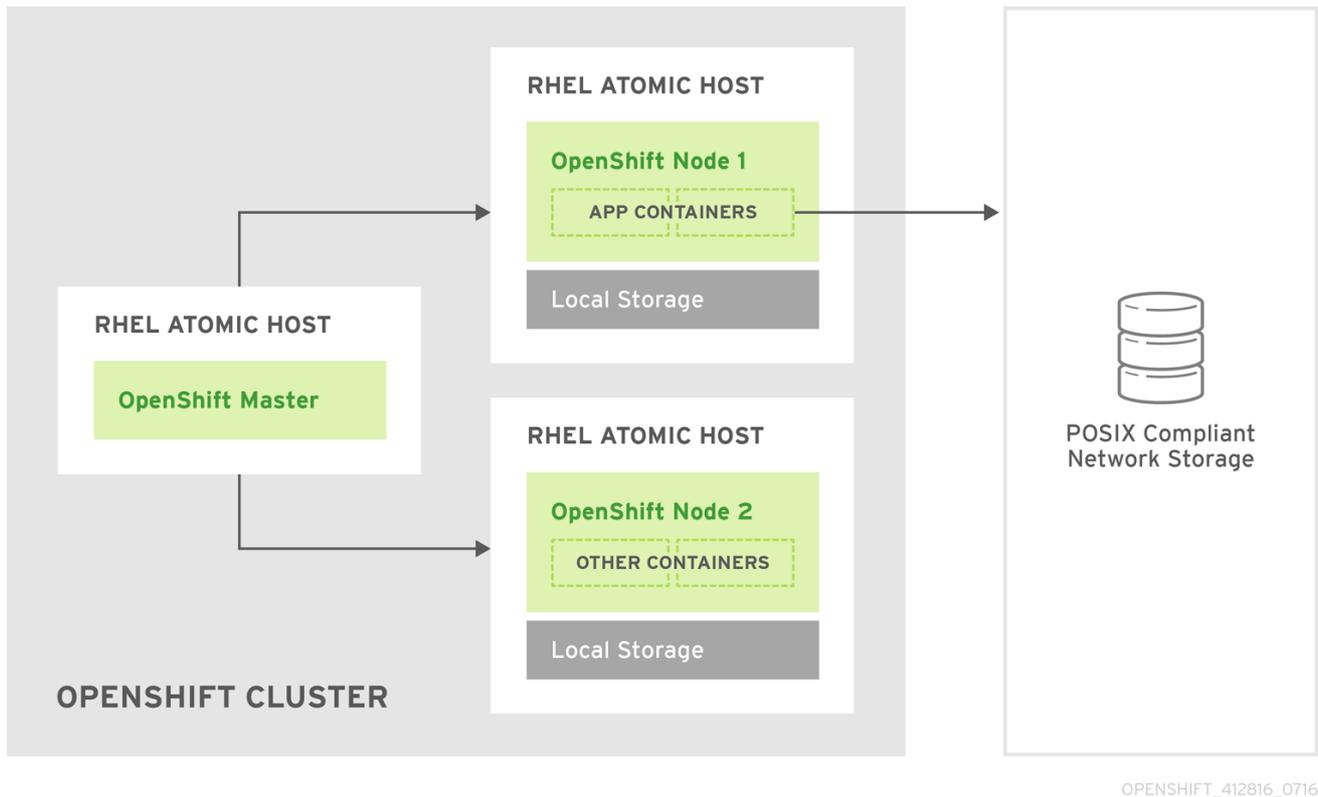
### 27.3.1.2. インデペンデントモード

独立モードの場合、Container-Ready Storage を使用することで、Red Hat Gluster Storage は独自の専用ノードで実行され、GlusterFS のボリューム管理 REST サービスの [heketi](#) のインスタンスによって管理されます。この heketi サービスは、スタンドアロンではなく、コンテナ化された状態で実行する必要があります。コンテナ化の場合、簡単なメカニズムで高可用性をサービスに提供できます。本書では、コンテナ化された heketi 設定にフォーカスします。

### 27.3.1.3. スタンドアロンの Red Hat Gluster Storage

スタンドアロンの Red Hat Gluster Storage クラスターが環境で使用できる場合、OpenShift Container Platform の GlusterFS ボリュームプラグインを使用してそのクラスター上でボリュームを使用することができます。この方法は、アプリケーションが専用のコンピュータノード、OpenShift Container Platform クラスターで実行され、ストレージはその専用ノードから提供される従来のデプロイメントです。

図27.2 アーキテクチャー - OpenShift Container Platform の GlusterFS ボリュームプラグインを使用したスタンドアロンの Red Hat Gluster Storage クラスタ



Red Hat Gluster Storage の詳細は、[Red Hat Gluster Storage Installation Guide](#) および [Red Hat Gluster Storage Administration Guide](#) を参照してください。



### 重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

#### 27.3.1.4. GlusterFS ボリューム

GlusterFS ボリュームは、POSIX に準拠したファイルシステムを提供し、クラスター上の1つ以上のノードにまたがる1つ以上のブリックから設定されます。このブリックは所定のストレージノード上のディレクトリであり、一般的にブロックストレージデバイスのマウントポイントになります。GlusterFS はボリュームの設定に応じて、所定のボリュームのブリック間でファイルの分散および複製を処理します。

heketi は、ボリューム管理において、作成、削除、サイズ変更といった一般的な操作に使用することが推奨されます。OpenShift Container Platform は、GlusterFS プロビジョナーを使用する際に heketi が存在していることを前提としています。heketi はデフォルトで、レプリカ数が3のボリュームを作成します。このボリュームの各ファイルには3つの異なるノードをまたがる3つのコピーがあります。したがって、heketi が使用する Red Hat Gluster Storage クラスタでは3つ以上のノードを利用可能にすることが推奨されます。

GlusterFS ボリュームに使用可能な機能は多数ありますが、これらについては本書では扱いません。

#### 27.3.1.5. gluster-block ボリューム

gluster-block ボリュームは、iSCSI 上にマウントすることが可能なボリュームです。既存の GlusterFS ボリュームにファイルを作成し、そのファイルをブロックデバイスとして iSCSI ターゲットを介して提

供することでマウントできます。このような GlusterFS ボリュームは、ブロックホスティングボリュームと呼ばれます。

gluster-block ボリュームにはトレードオフもあります。iSCSI ターゲットとして使用される場合、複数のノード/クライアントでマウントできる GlusterFS ボリュームとは対照的に、gluster-block ボリュームをマウントできるのは1回に1つのノード/クライアントのみです。ただし、バックエンドのファイルであるため、GlusterFS ボリュームでは一般にコストのかかる操作 (メタデータの参照など) を、GlusterFS ボリュームでの一般的により高速な操作 (読み取り、書き込みなど) に変換することが可能です。それにより、特定の負荷に対するパフォーマンスを大幅に改善できる可能性があります。



### 重要

OpenShift Container Storage と OpenShift Container Platform の相互運用性の詳細は、[OpenShift Container Storage and OpenShift Container Platform interoperability matrix](#) を参照してください。

#### 27.3.1.6. Gluster S3 Storage

Gluster S3 サービスは、ユーザーアプリケーションが S3 インターフェイスを介して GlusterFS ストレージへアクセスすることを可能にします。このサービスは GlusterFS の、オブジェクトデータ用とオブジェクトメタデータ用の2つのボリュームにバインドされ、受信する S3 REST 要求をボリューム上のファイルシステム操作に変換します。このサービスは OpenShift Container Platform 内の Pod として実行することが推奨されます。



### 重要

現時点では、Gluster S3 サービスの使用およびインストールはテクノロジープレビューの段階にあります。

## 27.3.2. 留意事項

このセクションでは、Red Hat Gluster Storage を OpenShift Container Platform で使用する際に考慮すべきトピックについて取り上げます。

### 27.3.2.1. ソフトウェア要件

GlusterFS ボリュームにアクセスするには、すべてのスケジュール可能なノードで `mount.glusterfs` コマンドを利用できる必要があります。RPM ベースのシステムの場合は、`glusterfs-fuse` パッケージがインストールされている必要があります。

```
# yum install glusterfs-fuse
```

このパッケージはすべての RHEL システムにインストールされています。ただし、サーバーが x86\_64 アーキテクチャーを使用する場合は Red Hat Gluster Storage の最新バージョンに更新することを推奨します。そのためには、以下の RPM リポジトリを有効にする必要があります。

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-server-rpms
```

`glusterfs-fuse` がノードにすでにインストールされている場合、最新バージョンがインストールされていることを確認します。

```
# yum update glusterfs-fuse
```

### 27.3.2.2. ハードウェア要件

コンバインドモードまたはインデペンデントモードのクラスターで使用されるノードはストレージノードとみなされます。単一ノードは複数のグループに分割できませんが、ストレージノードはそれぞれ別のクラスターグループに分類できます。ストレージノードの各グループについては、以下が当てはまります。

- Gluster ストレージのボリュームタイプオプションに基づき、1つのグループあたり最低でも1つまたは複数のストレージが必要です。
- 各ストレージノードには 8 GB 以上の RAM が必要です。これにより、Red Hat Gluster Storage Pod、その他のアプリケーションおよび基礎となる OS を実行できます。
  - 各 GlusterFS ボリュームはストレージクラスターにあるすべてのストレージノードのメモリ (約 30 MB) も消費します。RAM の合計量は、コンカレントボリュームがいくつ求められているか、またはいくつ予想されるかによって決める必要があります。
- 各ストレージノードには、現在のデータまたはメタデータを含まない1つ以上の raw ブロックデバイスが必要です。それらのブロックデバイス全体は GlusterFS ストレージで使用されません。以下が存在しないことを確認してください。
  - パーティションテーブル (GPT または MSDOS)
  - ファイルシステムまたは未処理のファイルシステムの署名
  - 以前のボリュームグループの LVM2 署名および論理ボリューム
  - LVM2 物理ボリュームの LVM2 メタデータ

不確かな場合には、**wipefs -a <device>** で上記のすべてを消去する必要があります。



#### 重要

2つのクラスター、つまりインフラストラクチャーアプリケーション (OpenShift Container レジストリーなど) のストレージ専用のクラスターと一般的なアプリケーションのストレージ専用のクラスターについて計画することをお勧めします。これには、合計で6つのストレージノードが必要になります。この設定は I/O およびボリューム作成のパフォーマンスへの潜在的な影響を回避するために推奨されます。

### 27.3.2.3. ストレージのサイジング

GlusterFS クラスターは、そのストレージを利用するアプリケーションの予想されるニーズに基づいてサイズを決定する必要があります。たとえば、[OpenShift ロギング](#) と [OpenShift メトリクス](#) の両方に適用できるサイジングについてのガイドを参照できます。

その他考慮すべき事項:

- 接続モードまたは独立モードのクラスターでは、デフォルトの動作により3通りのレプリケーションを持つ GlusterFS ボリュームが作成されます。そのため、合計のストレージの計画を立てる際には、必要な容量の3倍にする必要があります。
  - 例として、各 heketi インスタンスは 2 GB の **heketidbstorage** ボリュームを作成する場合、ストレージクラスター内の3つのノードに合計で 6 GB の raw ストレージが必要になります。この容量は常に必要となり、これをサイジングの際に考慮する必要があります。
  - 統合 OpenShift Container レジストリーなどのアプリケーションでは、GlusterFS の単一ボリュームがアプリケーションの複数インスタンスで共有されます。

- gluster-block ボリュームを使用する場合は、所定のブロックのボリューム容量をフルサイズで保持するための十分な容量を備えた GlusterFS ブロックホスティングボリュームがなければなりません。
  - デフォルトでは、このようなブロックホスティングボリュームが存在しない場合、これが設定されたサイズで自動的に1つ作成されます。デフォルトのサイズは100 GBです。クラスター内に新しいブロックホスティングボリュームを作成するための十分なスペースがない場合、ブロックボリュームの作成は失敗します。自動作成の動作、および自動作成されるボリュームのサイズはどちらも設定することが可能です。
  - OpenShift ロギングや OpenShift メトリクスなどの gluster-block ボリュームを使用する複数のインスタンスを持つアプリケーションは、各インスタンスにつき1つのボリュームを使用します。
- Gluster S3 サービスは、2つの GlusterFS ボリュームにバインドされます。[デフォルトのクラスターインストールでは](#)、ボリュームはそれぞれ1GBとなり、合計6GBのrawストレージを使用します。

#### 27.3.2.4. ボリューム操作の動作

作成や削除などのボリューム操作は、さまざまな環境条件の影響を受けることもあれば、アプリケーションに影響を与えることもあります。

- アプリケーション Pod が動的にプロビジョニングされた GlusterFS の Persistent Volume Claim (永続ボリューム要求、PVC) を要求する場合は、ボリュームの作成と対応する PVC へのバインドにかかる追加の時間を考慮する必要があります。これはアプリケーション Pod の起動時間に影響します。



#### 注記

GlusterFS ボリュームの作成時間は、ボリュームの数に応じて直線的に増加します。たとえば、クラスター内に推奨されるハードウェア仕様を使用する100のボリュームがある場合、各ボリュームの作成、割り当て、Pod へのバインドに約6秒の時間がかかりました。

- PVC が削除されると、基礎となる GlusterFS ボリュームの削除がトリガーされます。PVC は `oc get pvc` の出力からすぐに消えますが、ボリュームが完全に削除される訳ではありません。GlusterFS ボリュームは、`heketi-cli volume list` および `gluster volume list` のコマンドライン出力に表示されなくなったときにのみ削除されていると見なされます。



#### 注記

GlusterF ボリュームの削除とそのストレージのリサイクルの時間は、アクティブな GlusterFS ボリュームの数に応じて直線的に増加します。保留中のボリューム削除は実行中のアプリケーションに影響しませんが、ストレージ管理者は、とくにリソース消費を大きな規模で調整している場合には、ボリュームの削除にかかる時間を認識し、それを見積もることができるようにしておく必要があります。

#### 27.3.2.5. ボリュームのセキュリティ

このセクションでは、Portable Operating System Interface [Unix 向け] (POSIX) パーミッションや SELinux に関する考慮事項を含む、Red Hat Gluster Storage ボリュームのセキュリティについて説明します。[ボリュームのセキュリティ](#)、POSIX パーミッションおよび SELinux に関する基本を理解していることを前提とします。



## 重要

OpenShift Container Storage 3.11 では、永続ボリュームに対するアクセス制御のセキュリティを確保するには、SSL 暗号化を有効にする必要があります。

詳細は、[Red Hat OpenShift Container Storage 3.11 Operations Guide](#) を参照してください。

### 27.3.2.5.1. POSIX パーミッション

Red Hat Gluster Storage ボリュームは POSIX 準拠のファイルシステムを表します。そのため、`chmod` や `chown` などの標準コマンドラインツールを使用してアクセスパーミッションを管理できます。

接続モードと独立モードでは、ボリュームの作成時に、ボリュームの Root を所有するグループ ID を指定することも可能です。静的なプロビジョニングでは、これは、`heketi-cli` ボリュームの作成コマンドの一部として指定します。

```
$ heketi-cli volume create --size=100 --gid=10001000
```



## 警告

このボリュームに関連付けられる PersistentVolume には、PersistentVolume を使用する Pod がファイルシステムにアクセスできるように、グループ ID をアノテーションとして付加する必要があります。このアノテーションは以下の形式で指定します。

```
pv.beta.kubernetes.io/gid: "<GID>" ---
```

動的プロビジョニングの場合は、プロビジョナーがグループ ID を自動的に生成し、これを適用します。`gidMin` および `gidMax` StorageClass パラメーターを使用してこのグループ ID の選択範囲を制御できます ([動的プロビジョニング](#) を参照してください)。プロビジョナーは、生成される PersistentVolume にグループ ID をアノテーションとして付ける処理も行います。

### 27.3.2.5.2. SELinux

デフォルトでは、SELinux は Pod からリモート Red Hat Gluster Storage サーバーへの書き込みを許可しません。SELinux が有効な状態で Red Hat Gluster Storage ボリュームへの書き込みを有効にするには、GlusterFS を実行する各ノードで以下のコマンドを実行します。

```
$ sudo setsebool -P virt_sandbox_use_fusefs on 1
$ sudo setsebool -P virt_use_fusefs on
```

**1** `-P` オプションを使用すると、再起動した後もブール値が永続化されます。



## 注記

**virt\_sandbox\_use\_fusefs** ブール値は、**docker-selinux** パッケージによって定義されます。このブール値が定義されていないというエラーが表示される場合は、このパッケージがインストールされていることを確認してください。



## 注記

Atomic Host を使用しており、Atomic Host をアップグレードすると、SELinux のブール値が消去されます。Atomic Host をアップグレードする場合には、これらのブール値を設定し直す必要があります。

### 27.3.3. サポート要件

Red Hat Gluster Storage と OpenShift Container Platform のサポートされる統合を作成するには、以下の要件が満たされている必要があります。

インデペンデントモードまたは スタンドアロンの Red Hat Gluster Storage の場合:

- 最小バージョン: Red Hat Gluster Storage 3.4
- すべての Red Hat Gluster Storage ノードに Red Hat Network チャンネルとサブスクリプションマネージャーリポジトリへの有効なサブスクリプションが必要です。
- Red Hat Gluster Storage ノードは、[Planning Red Hat Gluster Storage Installation](#) に記載されている要件に準拠している必要があります。
- Red Hat Gluster Storage ノードは、最新のパッチとアップグレードが適用された完全に最新の状態であればなりません。最新バージョンへのアップグレードについては、[Red Hat Gluster Storage Installation Guide](#) を参照してください。
- 各 Red Hat Gluster Storage ノードには、完全修飾ドメイン名 (FQDN) が設定されている必要があります。適切な DNS レコードが存在すること、および FQDN が正引きと逆引きの両方の DNS ルックアップで解決できることを確認してください。

### 27.3.4. インストールシステム

スタンドアロンの Red Hat Gluster Storage の場合、OpenShift Container Platform で使用するためにインストールする必要があるコンポーネントはありません。OpenShift Container Platform には組み込み GlusterFS ポリウムドライバーが付属しており、これを使用して既存のボリュームを既存のクラスターで活用できます。既存のボリュームの使用方法については、[プロビジョニング](#) を参照してください。

コンバインドモードおよびインデペンデントモードでは、[クラスターインストール](#) プロセスを使用して、必要なコンポーネントをインストールすることを推奨します。

#### 27.3.4.1. 独立モード: Red Hat Gluster Storage ノードのインストール

独立モードの場合は、Red Hat Gluster Storage ノードに適切なシステム設定 (ファイアウォールポートやカーネルモジュールなど) が設定されており、Red Hat Gluster Storage サービスが実行されている必要があります。このサービスは追加で設定できず、Trusted Storage Pool を作成することはできません。

Red Hat Gluster Storage ノードのインストールは本書の対象外です。詳細については、[独立モードの設定](#) を参照してください。

### 27.3.4.2. インストーラーの使用



#### 重要

**glusterfs** と **glusterfs\_registry** のノードグループに別のノードを使用します。それぞれのインスタンスは、個別の **gluster** インスタンスである必要があります。**glusterfs** と **glusterfs\_registry** ノードグループに同じノードを使用すると、デプロイメントに失敗します。

**クラスターインストール** プロセスを使用すると、2つの GlusterFS ノードグループの1つまたは両方をインストールできます。

- **glusterfs**: ユーザーアプリケーションで使用するための一般的なストレージクラスター。
- **glusterfs\_registry**: 統合 OpenShift Container レジストリーなどのインフラストラクチャアプリケーション用の専用ストレージクラスター。

I/O およびボリューム作成のパフォーマンスへの潜在的な影響を回避するために、両方のグループをデプロイすることをお勧めします。これらは両方とも、インベントリーホストファイルで定義されています。

ストレージクラスターを定義するには、**[OSEv3:children]** グループに適切な名前を追加し、同じ名前のグループを作成します。次に、グループにノード情報を設定します。

**[OSEv3:children]** グループに **masters**、**nodes**、**etcd** および **glusterfs** と **glusterfs\_registry** のストレージクラスターを追加します。

グループの生成、設定後に、**[OSEv3:vars]** グループでより多くのパラメーター値を定義してクラスターを設定します。変数は GlusterFS クラスターと対話します。以下の例のようにインベントリーファイルに保存されます。

- **GlusterFS** 変数は **openshift\_storage\_glusterfs\_** で始まります。
- **glusterfs\_registry** 変数は **openshift\_storage\_glusterfs\_registry\_** で始まります。

以下のインベントリーファイルの例は、2つの GlusterFS ノードグループのデプロイ時に変数の使用を示しています。

```
[OSEv3:children]
masters
nodes
etcd
glusterfs
glusterfs_registry`

[OSEv3:vars]
install_method=rpm
os_update=false
install_update_docker=true
docker_storage_driver=devicemapper
ansible_ssh_user=root
openshift_release=v3.11
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:v3.11
#openshift_cockpit_deployer_image='registry.redhat.io/openshift3/registry-console:v3.11'
openshift_docker_insecure_registries=registry.access.redhat.com
openshift_deployment_type=openshift-enterprise
```

```
openshift_web_console_install=true
openshift_enable_service_catalog=false
osm_use_cockpit=false
osm_cockpit_plugins=['cockpit-kubernetes']
debug_level=5
openshift_set_hostname=true
openshift_override_hostname_check=true
openshift_disable_check=docker_image_availability
openshift_check_min_host_disk_gb=2
openshift_check_min_host_memory_gb=1
openshift_portal_net=172.31.0.0/16
openshift_master_cluster_method=native
openshift_clock_enabled=true
openshift_use_openshift_sdn=true

openshift_master_dynamic_provisioning_enabled=true

# logging
openshift_logging_install_logging=true
openshift_logging_es_pvc_dynamic=true
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_logging_es_pvc_size=20Gi
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-block"

# metrics
openshift_metrics_install_metrics=true
openshift_metrics_storage_kind=dynamic
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra": "true"}
openshift_metrics_storage_volume_size=20Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-registry-block"

# glusterfs
openshift_storage_glusterfs_timeout=900
openshift_storage_glusterfs_namespace=glusterfs
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_create=true
openshift_storage_glusterfs_block_host_vol_size=100

# glusterfs_registry
openshift_storage_glusterfs_registry_namespace=glusterfs-registry
openshift_storage_glusterfs_registry_storageclass=true
openshift_storage_glusterfs_registry_storageclass_default=false
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=false
```

```
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_create=true
openshift_storage_glusterfs_registry_block_host_vol_size=100

# glusterfs_registry_storage
openshift_hosted_registry_storage_kind=glusterfs
openshift_hosted_registry_storage_volume_size=20Gi
openshift_hosted_registry_selector="node-role.kubernetes.io/infra=true"

openshift_storage_glusterfs_heketi_admin_key='adminkey'
openshift_storage_glusterfs_heketi_user_key='heketiuserkey'

openshift_storage_glusterfs_image='registry.access.redhat.com/rhgs3/rhgs-server-rhel7:v3.11'

openshift_storage_glusterfs_heketi_image='registry.access.redhat.com/rhgs3/rhgs-volmanager-rhel7:v3.11'

openshift_storage_glusterfs_block_image='registry.access.redhat.com/rhgs3/rhgs-gluster-block-prov-rhel7:v3.11'

openshift_master_cluster_hostname=node101.redhat.com
openshift_master_cluster_public_hostname=node101.redhat.com

[masters]
node101.redhat.com

[etcd]
node101.redhat.com

[nodes]
node101.redhat.com openshift_node_group_name="node-config-master"
node102.redhat.com openshift_node_group_name="node-config-infra"
node103.redhat.com openshift_node_group_name="node-config-compute"
node104.redhat.com openshift_node_group_name="node-config-compute"
node105.redhat.com openshift_node_group_name="node-config-compute"
node106.redhat.com openshift_node_group_name="node-config-compute"
node107.redhat.com openshift_node_group_name="node-config-compute"
node108.redhat.com openshift_node_group_name="node-config-compute"

[glusterfs]
node103.redhat.com glusterfs_zone=1 glusterfs_devices='["/dev/sdd"]'
node104.redhat.com glusterfs_zone=2 glusterfs_devices='["/dev/sdd"]'
node105.redhat.com glusterfs_zone=3 glusterfs_devices='["/dev/sdd"]'

[glusterfs_registry]
node106.redhat.com glusterfs_zone=1 glusterfs_devices='["/dev/sdd"]'
node107.redhat.com glusterfs_zone=2 glusterfs_devices='["/dev/sdd"]'
node108.redhat.com glusterfs_zone=3 glusterfs_devices='["/dev/sdd"]'
```

#### 27.3.4.2.1. ホスト変数

**glusterfs** と **glusterfs\_registry** グループの各ホストには **glusterfs\_devices** 変数が定義されている必要があります。この変数は GlusterFS クラスターの一部として管理されるブロックデバイスの一覧を定

義します。パーティションなしか、LVM PV が設定されているベアメタルのデバイス1つ以上必要です。

ホストごとに以下の変数を定義することもできます。それらが定義されている場合、それらの変数はホスト設定を GlusterFS ノードとしてさらに制御します。

- **glusterfs\_cluster**: このノードが属するクラスターの ID。
- **glusterfs\_hostname**: 内部 GlusterFS 通信に使用されるホスト名または IP アドレス。
- **glusterfs\_ip**: Pod が GlusterFS ノードと通信するために使用する IP アドレス
- **glusterfs\_zone**: ノードのゾーン番号。クラスター内で、ゾーンは GlusterFS ボリュームのブリックを分散する方法を決定します。

#### 27.3.4.2.2. ロール変数

GlusterFS クラスターの新規または既存の OpenShift Container Platform クラスターへの統合を制御するために、インベントリーファイルに保存される多数のロール変数を定義することもできます。また、各ロール変数には、個別の GlusterFS クラスターを統合 Docker レジストリーのストレージとして使用するように任意で設定するための、対応する変数があります。

#### 27.3.4.2.3. イメージ名とバージョンタグ変数

OpenShift Container Platform Pod が停止後に、バージョンが異なる OpenShift Container Platform のクラスターにアップグレードされないようにするには、すべてのコンテナ化コンポーネントにイメージ名とバージョンタグを指定することが推奨されます。これらの変数は以下のとおりです。

- **openshift\_storage\_glusterfs\_image**
- **openshift\_storage\_glusterfs\_block\_image**
- **openshift\_storage\_glusterfs\_s3\_image**
- **openshift\_storage\_glusterfs\_heketi\_image**



#### 注記

**gluster-block** および **gluster-s3** のイメージ変数は、適切なデプロイメント変数 (末尾が **\_block\_deploy** および **\_s3\_deploy** の変数) が True の場合のみ必要です。

デプロイを正常に実行するには、有効なイメージタグが必要です。インベントリーファイルの以下の変数について [interoperability matrix](#) で説明されているように、**<tag>** を OpenShift Container Platform 3.11 と互換性のある Red Hat Gluster Storage のバージョンに置き換えます。

- **openshift\_storage\_glusterfs\_image=registry.redhat.io/rhgs3/rhgs-server-rhel7:<tag>**
- **openshift\_storage\_glusterfs\_block\_image=registry.redhat.io/rhgs3/rhgs-gluster-block-prov-rhel7:<tag>**
- **openshift\_storage\_glusterfs\_s3\_image=registry.redhat.io/rhgs3/rhgs-s3-server-rhel7:<tag>**
- **openshift\_storage\_glusterfs\_heketi\_image=registry.redhat.io/rhgs3/rhgs-volmanager-rhel7:<tag>**

- `openshift_storage_glusterfs_registry_image=registry.redhat.io/rhgs3/rhgs-server-rhel7:<tag>`
- `openshift_storage_glusterfs_block_registry_image=registry.redhat.io/rhgs3/rhgs-gluster-block-prov-rhel7:<tag>`
- `openshift_storage_glusterfs_s3_registry_image=registry.redhat.io/rhgs3/rhgs-s3-server-rhel7:<tag>`
- `openshift_storage_glusterfs_heketi_registry_image=registry.redhat.io/rhgs3/rhgs-volmanager-rhel7:<tag>`

変数の詳細な一覧については、GitHub の [GlusterFS ロールに関する README](#) を参照してください。

変数を設定したら、インストールの環境に応じて、いくつかの Playbook が利用可能になります。

- クラスターインストールのメイン Playbook を使用すると、OpenShift Container Platform の初期インストールと並行して GlusterFS クラスターをデプロイできます。
  - これには、GlusterFS ストレージを使用する統合された OpenShift Container Registry のデプロイが含まれます。
  - OpenShift ログインや OpenShift メトリクスは含まれません。詳細は、[OpenShift ログインおよびメトリクス用のコンバージドモード](#) を参照してください。
- `playbooks/openshift-glusterfs/config.yml` を使用して、クラスターを既存の OpenShift Container Platform インストールにデプロイできます。
- `playbooks/openshift-glusterfs/registry.yml` を使用して、クラスターを既存の OpenShift Container Platform インストールにデプロイできます。さらに、GlusterFS ストレージを使用する統合 OpenShift Container レジストリーもデプロイされます。



### 重要

OpenShift Container Platform クラスターに既存のレジストリーがあってはなりません。

- `playbooks/openshift-glusterfs/uninstall.yml` を使用して、インベントリーホストファイルの設定に一致する既存のクラスターを削除できます。これは、設定エラーによってデプロイメントが失敗した場合に OpenShift Container Platform 環境をクリーンアップするのに便利です。



### 注記

GlusterFS Playbook は、べき等である保証はありません。



### 注記

GlusterFS インストール全体 (ディスクデータを含む) を削除してインストールし直すことなく、特定のインストールに対して Playbook を複数回実行することは、現在はサポートされていません。

#### 27.3.4.2.4. 例: 基本的なコンバージドモードのインストール

1. インベントリーファイルの `[OSEv3:vars]` セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...
openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
```

2. **[OSEv3:children]** セクションに **glusterfs** を追加して、**[glusterfs]** グループを有効にします。

```
[OSEv3:children]
masters
nodes
glusterfs
```

3. GlusterFS ストレージをホストする各ストレージノードのエントリーを含む **[glusterfs]** セクションを追加します。ノードごとに、**glusterfs\_devices** を GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定します。少なくとも1つのデバイスを一覧に含める必要があります。各デバイスはパーティションや LVM PV がないベアでなければなりません。変数は次の形式で指定します。

```
<hostname_or_ip> glusterfs_devices='["</path/to/device1/>", "</path/to/device2/>", ... ]'
```

以下に例を示します。

```
[glusterfs]
node11.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'
```

4. **[glusterfs]** の下に一覧表示されているホストを **[nodes]** グループに追加します。

```
[nodes]
...
node11.example.com openshift_node_group_name="node-config-compute"
node12.example.com openshift_node_group_name="node-config-compute"
node13.example.com openshift_node_group_name="node-config-compute"
```



### 注記

前述の手順では、インベントリーファイルに追加する必要がある一部のオプションのみを指定しています。Red Hat Gluster Storage をデプロイするには、完全なインベントリーファイルを使用します。

5. Playbook ディレクトリーに切り替え、インストール Playbook を実行します。インベントリーファイルの相対パスをオプションとして指定します。

- OpenShift Container Platform の新規インストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
$ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
```

#### 27.3.4.2.5. 例: 基本的なインデペンデントモードのインストール

1. インベントリーファイルの **[OSEv3:vars]** セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...
openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
openshift_storage_glusterfs_is_native=false
openshift_storage_glusterfs_heketi_is_native=true
openshift_storage_glusterfs_heketi_executor=ssh
openshift_storage_glusterfs_heketi_ssh_port=22
openshift_storage_glusterfs_heketi_ssh_user=root
openshift_storage_glusterfs_heketi_ssh_sudo=false
openshift_storage_glusterfs_heketi_ssh_keyfile="/root/.ssh/id_rsa"
```

2. **[OSEv3:children]** セクションに **glusterfs** を追加して、**[glusterfs]** グループを有効にします。

```
[OSEv3:children]
masters
nodes
glusterfs
```

3. GlusterFS ストレージをホストする各ストレージノードのエントリーを含む **[glusterfs]** セクションを追加します。ノードごとに、**glusterfs\_devices** を GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定します。少なくとも1つのデバイスを一覧に含める必要があります。各デバイスはパーティションや LVM PV がないペアでなければなりません。また、**glusterfs\_ip** をノードの IP アドレスに設定します。変数は次の形式で指定します。

```
<hostname_or_ip> glusterfs_ip=<ip_address> glusterfs_devices=[ "</path/to/device1/>", "  
</path/to/device2/>", ... ]'
```

以下に例を示します。

```
[glusterfs]
gluster1.example.com glusterfs_ip=192.168.10.11 glusterfs_devices=[ "/dev/xvdc",  
"/dev/xvdd" ]'
```

```
gluster2.example.com glusterfs_ip=192.168.10.12 glusterfs_devices=[ "/dev/xvdc",
"/dev/xvdd" ]'
gluster3.example.com glusterfs_ip=192.168.10.13 glusterfs_devices=[ "/dev/xvdc",
"/dev/xvdd" ]'
```



### 注記

前述の手順では、インベントリーファイルに追加する必要のある一部のオプションのみを指定しています。Red Hat Gluster Storage をデプロイするには、完全なインベントリーファイルを使用します。

- Playbook ディレクトリーに切り替え、インストール Playbook を実行します。インベントリーファイルの相対パスをオプションとして指定します。

- OpenShift Container Platform の新規インストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
$ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
```

#### 27.3.4.2.6. 例: 統合 OpenShift Container レジストリーを使用する接続モード

- インベントリーファイルの **[OSEv3:vars]** セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs ①
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-role.kubernetes.io/infra=true'
```

- ① 統合 OpenShift Container Registry をインフラストラクチャーノードで実行することが推奨されます。インフラストラクチャーノードは、OpenShift Container Platform クラスターのサービスを提供するために管理者がデプロイするアプリケーションを実行する専用ノードです。

- [OSEv3:children]** セクションに **glusterfs\_registry** を追加して、**[glusterfs\_registry]** グループを有効にします。

```
[OSEv3:children]
masters
nodes
glusterfs_registry
```

- GlusterFS ストレージをホストする各ストレージノードのエントリーを含む **[glusterfs\_registry]** セクションを追加します。ノードごとに、**glusterfs\_devices** を GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定しま

す。少なくとも1つのデバイスを一覧に含める必要があります。各デバイスはパーティションやLVM PVがないペアでなければなりません。変数は次の形式で指定します。

```
<hostname_or_ip> glusterfs_devices='[ "</path/to/device1/>", "</path/to/device2/>", ... ]'
```

以下に例を示します。

```
[glusterfs_registry]
node11.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='[ "/dev/xvdc", "/dev/xvdd" ]'
```

4. **[glusterfs\_registry]** の下に一覧表示されているホストを **[nodes]** グループに追加します。

```
[nodes]
...
node11.example.com openshift_node_group_name="node-config-infra"
node12.example.com openshift_node_group_name="node-config-infra"
node13.example.com openshift_node_group_name="node-config-infra"
```



### 注記

前述の手順では、インベントリーファイルに追加する必要がある一部のオプションのみを指定しています。Red Hat Gluster Storage をデプロイするには、完全なインベントリーファイルを使用します。

5. Playbook ディレクトリーに切り替え、インストール Playbook を実行します。インベントリーファイルの相対パスをオプションとして指定します。

- OpenShift Container Platform の新規インストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
$ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
```

#### 27.3.4.2.7. 例: OpenShift ロギングおよびメトリクス用のコンバートモード

1. インベントリーファイルの **[OSEv3:vars]** セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...

openshift_metrics_install_metrics=true
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra": "true"} ①
openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra": "true"} ②
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra": "true"} ③
```

```

openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-registry-block" 4

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"} 5
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"} 6
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"} 7
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi 8
openshift_logging_elasticsearch_storage_type=pvc 9
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-block" 10

openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=false

```

- 1 2 3 5 6 7 統合 OpenShift Container レジストリー、ロギングおよびメトリクスは、管理者が OpenShift Container Platform クラスターにサービスを提供するためにデプロイしたアプリケーションであるインフラストラクチャーアプリケーション専用のノードで実行することを推奨します。
- 4 10 ロギングとメトリクスに使用する StorageClass を指定します。この名前は、ターゲットの GlusterFS クラスター (例: **glusterfs-<name>-block**) 名から生成されます。この例では **registry** にデフォルト設定されています。
- 8 OpenShift ロギングでは、PVC のサイズを指定する必要があります。ここで指定される値は単なる例であり、推奨される値ではありません。
- 9 永続 Elasticsearch ストレージを使用している場合は、ストレージタイプを **pvc** に設定します。



### 注記

これらの変数とその他の変数の詳細は [GlusterFS ロールに関する README](#) を参照してください。

2. **[OSEv3:children]** セクションに **glusterfs\_registry** を追加して、**[glusterfs\_registry]** グループを有効にします。

```

[OSEv3:children]
masters
nodes
glusterfs_registry

```

3. GlusterFS ストレージをホストする各ストレージノードのエントリーを含む **[glusterfs\_registry]** セクションを追加します。ノードごとに、**glusterfs\_devices** を GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定します。少なくとも1つのデバイスを一覧に含める必要があります。各デバイスはパーティションや LVM PV がないペアでなければなりません。変数は次の形式で指定します。

```
<hostname_or_ip> glusterfs_devices='["</path/to/device1/>", "</path/to/device2>", ... ]'
```

以下に例を示します。

```
[glusterfs_registry]
node11.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'
```

4. **[glusterfs\_registry]** の下に一覧表示されているホストを **[nodes]** グループに追加します。

```
[nodes]
...
node11.example.com openshift_node_group_name="node-config-infra"
node12.example.com openshift_node_group_name="node-config-infra"
node13.example.com openshift_node_group_name="node-config-infra"
```



### 注記

前述の手順では、インベントリーファイルに追加する必要のある一部のオプションのみを指定しています。Red Hat Gluster Storage をデプロイするには、完全なインベントリーファイルを使用します。

5. Playbook ディレクトリーに切り替え、インストール Playbook を実行します。インベントリーファイルの相対パスをオプションとして指定します。

- OpenShift Container Platform の新規インストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
$ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
```

#### 27.3.4.2.8. 例: アプリケーション、レジストリー、ロギングおよびメトリクス用のコンバージドモード

1. インベントリーファイルの **[OSEv3:vars]** セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs ❶
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-role.kubernetes.io/infra=true'

openshift_metrics_install_metrics=true
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra": "true"} ❷
openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra": "true"} ❸
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra": "true"} ❹
```

```

openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-registry-block" 5

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"} 6
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"} 7
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"} 8
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi 9
openshift_logging_elasticsearch_storage_type=pvc 10
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-block" 11

openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100 12
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false

openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=false

```

1 2 3 4 6 7 8 統合 OpenShift Container レジストリー、ロギングおよびメトリクスをインフラストラクチャーノードで実行することが推奨されます。インフラストラクチャーノードは、OpenShift Container Platform クラスターのサービスを提供するために管理者がデプロイするアプリケーションを実行する専用ノードです。

5 11 ロギングとメトリクスに使用する StorageClass を指定します。この名前は、ターゲットの GlusterFS クラスター (例: **glusterfs-`<name>-block`**) 名から生成されます。この例では **<name>** は **registry** にデフォルト設定されています。

9 OpenShift ロギングでは、PVC サイズを指定する必要があります。ここで指定される値は単なる例であり、推奨される値ではありません。

10 永続 Elasticsearch ストレージを使用している場合は、ストレージタイプを **pvc** に設定します。

12 Glusterblock ボリュームをホストするために自動作成される GlusterFS ボリュームのサイズ (GB)。この変数は、glusterblock volume create 要求で十分な領域がない場合にのみ使用されます。この値は、glusterblock ボリュームのサイズの上限を表します。ただし、GlusterFS ブロックホスティングボリュームを手動で作成した場合は除きます。

2. **[OSEv3:children]** セクションに **glusterfs** と **glusterfs\_registry** を追加し、**[glusterfs]** と **[glusterfs\_registry]** グループを有効にします。

```

[OSEv3:children]
...
glusterfs
glusterfs_registry

```

3. **[glusterfs]** セクションと **[glusterfs\_registry]** セクションを追加し、両セクションに GlusterFS ストレージをホストするストレージノードを入力します。ノードごとに **glusterfs\_devices** を、GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定します。少なくとも1つのデバイスを一覧に含める必要があります。各デバイスはパーティションや LVM PV がないベアでなければなりません。変数は次の形式で指定します。

```
<hostname_or_ip> glusterfs_devices='["<path/to/device1/>", "<path/to/device2/>", ... ]'
```

以下に例を示します。

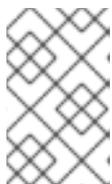
```
[glusterfs]
node11.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'
node12.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'
node13.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'

[glusterfs_registry]
node14.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'
node15.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'
node16.example.com glusterfs_devices='["/dev/xvdc", "/dev/xvdd" ]'
```

4. **[glusterfs]** と **[glusterfs\_registry]** に一覧表示されているホストを **[nodes]** グループに追加します。

```
[nodes]
...
node11.example.com openshift_node_group_name='node-config-compute' ①
node12.example.com openshift_node_group_name='node-config-compute' ②
node13.example.com openshift_node_group_name='node-config-compute' ③
node14.example.com openshift_node_group_name='node-config-infra' ④
node15.example.com openshift_node_group_name='node-config-infra' ⑤
node16.example.com openshift_node_group_name='node-config-infra' ⑥
```

- ① ② ③ ④ ⑤ ⑥ 各ノードには、一般的なアプリケーションまたはインフラストラクチャーアプリケーションのスケジューリングをそれらのノードで許可するかどうかを示すマークが付けられます。アプリケーションの制限方法は管理者が設定します。



### 注記

前述の手順では、インベントリーファイルに追加する必要のある一部のオプションのみを指定しています。Red Hat Gluster Storage をデプロイするには、完全なインベントリーファイルを使用します。

5. Playbook ディレクトリーに切り替え、インストール Playbook を実行します。インベントリーファイルの相対パスをオプションとして指定します。

- OpenShift Container Platform の新規インストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
$ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
```

### 27.3.4.2.9. 例: アプリケーション、レジストリー、ロギングおよびメトリクス用のインデペンデントモード

1. インベントリーファイルの **[OSEv3:vars]** セクションに次の変数を追加し、設定に合わせてそれらを調整します。

```
[OSEv3:vars]
...
openshift_hosted_registry_storage_kind=glusterfs 1
openshift_hosted_registry_storage_volume_size=5Gi
openshift_hosted_registry_selector='node-role.kubernetes.io/infra=true'

openshift_metrics_install_metrics=true
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra": "true"} 2
openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra": "true"} 3
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra": "true"} 4
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=10Gi
openshift_metrics_cassandra_pvc_storage_class_name="glusterfs-registry-block" 5

openshift_logging_install_logging=true
openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra": "true"} 6
openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra": "true"} 7
openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra": "true"} 8
openshift_logging_storage_kind=dynamic
openshift_logging_es_pvc_size=10Gi 9
openshift_logging_elasticsearch_storage_type 10
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-block" 11

openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=true
openshift_storage_glusterfs_block_host_vol_size=100 12
openshift_storage_glusterfs_block_storageclass=true
openshift_storage_glusterfs_block_storageclass_default=false
openshift_storage_glusterfs_is_native=false
openshift_storage_glusterfs_heketi_is_native=true
openshift_storage_glusterfs_heketi_executor=ssh
openshift_storage_glusterfs_heketi_ssh_port=22
openshift_storage_glusterfs_heketi_ssh_user=root
openshift_storage_glusterfs_heketi_ssh_sudo=false
openshift_storage_glusterfs_heketi_ssh_keyfile="/root/.ssh/id_rsa"

openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_host_vol_size=100
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=false
openshift_storage_glusterfs_registry_is_native=false
```

```

openshift_storage_glusterfs_registry_heketi_is_native=true
openshift_storage_glusterfs_registry_heketi_executor=ssh
openshift_storage_glusterfs_registry_heketi_ssh_port=22
openshift_storage_glusterfs_registry_heketi_ssh_user=root
openshift_storage_glusterfs_registry_heketi_ssh_sudo=false
openshift_storage_glusterfs_registry_heketi_ssh_keyfile="/root/.ssh/id_rsa"

```

- 1 2 3 4 6 7 8 統合 OpenShift Container レジストリーは、管理者が OpenShift Container Platform クラスターにサービスを提供するためにデプロイしたアプリケーションであるインフラストラクチャーアプリケーション専用のノードで実行することが推奨されます。インフラストラクチャーアプリケーション用のノードは、管理者が選択し、それにラベルを付けます。
- 5 11 ログイングとメトリクスに使用する StorageClass を指定します。この名前は、ターゲットの GlusterFS クラスター (例: **glusterfs-<name>-block**) 名から生成されます。この例では **registry** にデフォルト設定されています。
- 9 OpenShift ログイングでは、PVC のサイズを指定する必要があります。ここで指定される値は単なる例であり、推奨される値ではありません。
- 10 永続 Elasticsearch ストレージを使用している場合は、ストレージタイプを **pvc** に設定します。
- 12 Glusterblock ボリュームをホストするために自動作成される GlusterFS ボリュームのサイズ (GB)。この変数は、glusterblock volume create 要求で十分な領域がない場合にのみ使用されます。この値は、glusterblock ボリュームのサイズの上限を表します。ただし、GlusterFS ブロックホスティングボリュームを手動で作成した場合は除きます。

2. **[OSEv3:children]** セクションに **glusterfs** と **glusterfs\_registry** を追加し、**[glusterfs]** と **[glusterfs\_registry]** グループを有効にします。

```

[OSEv3:children]
...
glusterfs
glusterfs_registry

```

3. **[glusterfs]** セクションと **[glusterfs\_registry]** セクションを追加し、両セクションに GlusterFS ストレージをホストするストレージノードを入力します。ノードごとに **glusterfs\_devices** を、GlusterFS クラスターの一部として完全に管理される raw ブロックデバイスの一覧に設定します。少なくとも1つのデバイスを一覧に含める必要があります。各デバイスはパーティションや LVM PV がないベアでなければなりません。また、**glusterfs\_ip** をノードの IP アドレスに設定します。変数は次の形式で指定します。

```

<hostname_or_ip> glusterfs_ip=<ip_address> glusterfs_devices=[ "</path/to/device1/>",
</path/to/device2>", ... ]'

```

以下に例を示します。

```

[glusterfs]
gluster1.example.com glusterfs_ip=192.168.10.11 glusterfs_devices=[ "/dev/xvdc",
"/dev/xvdd" ]'
gluster2.example.com glusterfs_ip=192.168.10.12 glusterfs_devices=[ "/dev/xvdc",
"/dev/xvdd" ]'
gluster3.example.com glusterfs_ip=192.168.10.13 glusterfs_devices=[ "/dev/xvdc",

```

```
"/dev/xvdd" ]'
```

```
[glusterfs_registry]
```

```
gluster4.example.com glusterfs_ip=192.168.10.14 glusterfs_devices=[ "/dev/xvdc",
"/dev/xvdd" ]'
```

```
gluster5.example.com glusterfs_ip=192.168.10.15 glusterfs_devices=[ "/dev/xvdc",
"/dev/xvdd" ]'
```

```
gluster6.example.com glusterfs_ip=192.168.10.16 glusterfs_devices=[ "/dev/xvdc",
"/dev/xvdd" ]'
```



## 注記

前述の手順では、インベントリーファイルに追加する必要がある一部のオプションのみを指定しています。Red Hat Gluster Storage をデプロイするには、完全なインベントリーファイルを使用します。

4. Playbook ディレクトリーに切り替え、インストール Playbook を実行します。インベントリーファイルの相対パスをオプションとして指定します。

- OpenShift Container Platform の新規インストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/prerequisites.yml
$ ansible-playbook -i <path_to_inventory_file> playbooks/deploy_cluster.yml
```

- 既存の OpenShift Container Platform クラスターへのインストール:

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/config.yml
```

### 27.3.5. 接続モードのアンインストール

コンバインドモードの場合、OpenShift Container Platform のインストールには、クラスターから全リソースおよびアーティファクトをアンインストールするための Playbook が同梱されています。この Playbook を使用するには、コンバインドモードのターゲットインスタンスをインストールするのに使用した元のインベントリーファイルを指定して、Playbook ディレクトリーに切り替え、以下の Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> playbooks/openshift-glusterfs/uninstall.yml
```

さらに、Playbook は、**openshift\_storage\_glusterfs\_wipe** という変数の使用をサポートします。これは、有効にされている場合には、Red Hat Gluster Storage バックエンドストレージに使用されていたブロックデバイス上のデータを破棄します。**openshift\_storage\_glusterfs\_wipe** 変数を使用するには、Playbook ディレクトリーに切り替え、以下の Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <path_to_inventory_file> -e \
"openshift_storage_glusterfs_wipe=true" \
playbooks/openshift-glusterfs/uninstall.yml
```

**警告**

この手順では、データが破棄されます。注意して進めてください。

### 27.3.6. プロビジョニング

GlusterFS ボリュームは、静的または動的にプロビジョニングできます。静的プロビジョニングは、すべての設定で使用できます。動的プロビジョニングは、接続モードおよび独立モードでサポートされません。

#### 27.3.6.1. 静的プロビジョニング

1. 静的プロビジョニングを有効にするには、最初に GlusterFS ボリュームを作成します。**gluster** コマンドラインインターフェースの使用方法については、[Red Hat Gluster Storage Administration Guide](#)、**heketi-cli** を使用した方法については [heketi project site](#) のプロジェクトサイトを参照してください。この例では、ボリュームに **myVol1** という名前を付けます。
2. **gluster-endpoints.yaml** で以下のサービスとエンドポイントを定義します。

```

---
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster 1
spec:
  ports:
  - port: 1
---
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster 2
subsets:
- addresses:
  - ip: 192.168.122.221 3
  ports:
  - port: 1 4
- addresses:
  - ip: 192.168.122.222 5
  ports:
  - port: 1 6
- addresses:
  - ip: 192.168.122.223 7
  ports:
  - port: 1 8

```

1 2 これらの名前は一致している必要があります。

3 5 7 ip の値には、Red Hat Gluster Storage サーバーのホスト名ではなく、実際の IP アドレスを指定する必要があります。

4 6 8 ポート番号は無視されます。

- OpenShift Container Platform マスターホストからサービスとエンドポイントを作成します。

```
$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created
```

- サービスとエンドポイントが作成されたことを確認します。

```
$ oc get services
NAME                CLUSTER_IP      EXTERNAL_IP  PORT(S)  SELECTOR  AGE
glusterfs-cluster   172.30.205.34   <none>       1/TCP    <none>    44s

$ oc get endpoints
NAME                ENDPOINTS                                     AGE
docker-registry    10.1.0.3:5000                                  4h
glusterfs-cluster   192.168.122.221:1,192.168.122.222:1,192.168.122.223:1  11s
kubernetes          172.16.35.3:8443                                4d
```



#### 注記

エンドポイントはプロジェクトごとに一意です。GlusterFS にアクセスする各プロジェクトには独自のエンドポイントが必要です。

- ボリュームにアクセスするには、ボリューム上のファイルシステムにアクセスできるユーザー ID (UID) またはグループ ID (GID) でコンテナを実行する必要があります。この情報は以下の方法で取得できます。

```
$ mkdir -p /mnt/glusterfs/myVol1

$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1

$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0 myVol1 1 2
```

1 UID は 592 です。

2 GID は 590 です。

- gluster-pv.yaml** で以下の PersistentVolume (PV) を定義します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume 1
  annotations:
    pv.beta.kubernetes.io/gid: "590" 2
spec:
  capacity:
    storage: 2Gi 3
  accessModes: 4
```

```

- ReadWriteMany
glusterfs:
  endpoints: glusterfs-cluster ⑤
  path: myVol1 ⑥
  readOnly: false
persistentVolumeReclaimPolicy: Retain

```

- ① ボリュームの名前。
- ② GlusterFS ボリュームのルートの GID です。
- ③ このボリュームに割り当てられるストレージの量。
- ④ **accessModes** は、PV と PVC を一致させるためのラベルとして使用されます。現時点で、これらはいずれの形態のアクセス制御も定義しません。
- ⑤ 以前に作成されたエンドポイントリソースです。
- ⑥ アクセス対象の GlusterFS ボリュームです。

7. OpenShift Container Platform マスターホストから PV を作成します。

```
$ oc create -f gluster-pv.yaml
```

8. PV が作成されたことを確認します。

```

$ oc get pv
NAME                LABELS  CAPACITY  ACCESSMODES  STATUS  CLAIM
REASON  AGE
gluster-default-volume <none>  2147483648  RWX         Available    2s

```

9. **gluster-claim.yaml** で、新規 PV にバインドする PersistentVolumeClaim (PVC) を作成します。

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim ①
spec:
  accessModes:
    - ReadWriteMany ②
  resources:
    requests:
      storage: 1Gi ③

```

- ① この要求名は、**volumes** セクションで Pod によって参照されます。
- ② PV の **accessModes** に一致する必要があります。
- ③ この要求は、**1Gi** 以上の容量がある PV を検索します。

10. OpenShift Container Platform マスターホストから PVC を作成します。

```
$ oc create -f gluster-claim.yaml
```

- PV と PVC がバインドされていることを確認します。

```
$ oc get pv
NAME          LABELS    CAPACITY  ACCESSMODES  STATUS   CLAIM          REASON  AGE
gluster-pv   <none>   1Gi       RWX           Available gluster-claim  37s

$ oc get pvc
NAME          LABELS    STATUS    VOLUME     CAPACITY  ACCESSMODES  AGE
gluster-claim <none>   Bound    gluster-pv 1Gi       RWX          24s
```



### 注記

PVC はプロジェクトごとに一意です。GlusterFS ボリュームにアクセスする各プロジェクトには独自の PVC が必要です。PV は単一のプロジェクトにバインドされないため、複数のプロジェクトにまたがる PVC が同じ PV を参照する場合があります。

### 27.3.6.2. 動的プロビジョニング

- 動的プロビジョニングを有効にするには、最初に **StorageClass** オブジェクト定義を作成します。以下の定義は、OpenShift Container Platform でこの例を使用するために必要な最小要件に基づいています。その他のパラメーターと仕様定義については、[動的プロビジョニングとストレージクラスの作成](#) を参照してください。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: glusterfs
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://10.42.0.0:8080" ①
  restauthenabled: "false" ②
```

- heketi サーバーの URL です。
- この例では認証が有効ではないため、**false** に設定します。

- OpenShift Container Platform マスターホストから StorageClass を作成します。

```
# oc create -f gluster-storage-class.yaml
storageclass "glusterfs" created
```

- 新たに作成される StorageClass を使用して PVC を作成します。以下に例を示します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster1
spec:
  accessModes:
```

```
- ReadWriteMany
resources:
  requests:
    storage: 30Gi
  storageClassName: glusterfs
```

4. OpenShift Container Platform マスターホストから PVC を作成します。

```
# oc create -f glusterfs-dyn-pvc.yaml
persistentvolumeclaim "gluster1" created
```

5. PVC を表示し、ボリュームが動的に作成され、PVC にバインドされていることを確認します。

```
# oc get pvc
NAME      STATUS VOLUME                                     CAPACITY ACCESSMODES
STORAGECLASS AGE
gluster1  Bound  pvc-78852230-d8e2-11e6-a3fa-0800279cf26f  30Gi    RWX
glusterfs 42s
```

## 27.4. OPENSTACK CINDER を使用した永続ストレージ

### 27.4.1. 概要

OpenStack Cinder を使用して、OpenShift Container Platform クラスタに [永続ストレージ](#) をプロビジョニングできます。これには、Kubernetes と OpenStack についてある程度の理解があることが前提となります。



#### 重要

Cinder を使用して永続ボリューム (PV) を作成する前に、[OpenStack 向けに OpenShift Container Platform を設定](#) してください。

Kubernetes [永続ボリューム](#) フレームワークは、管理者がクラスタのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。OpenStack Cinder ボリュームを動的に [プロビジョニング](#) できます。

永続ボリュームは単一のプロジェクトまたは namespace にバインドされず、それらは OpenShift Container Platform クラスタ間で共有できます。ただし、[Persistent Volume Claim \(永続ボリューム要求\)](#) は、プロジェクトまたは namespace に固有で、ユーザーが要求できます。



#### 重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

### 27.4.2. Cinder PV のプロビジョニング

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。OpenShift Container Platform が [OpenStack 用に設定されている](#) ことを確認した後、Cinder に必要なのは、Cinder ボリューム ID と **PersistentVolume** API のみとなります。

### 27.4.2.1. 永続ボリュームの作成

OpenShift Container Platform に PV を作成する前に、PV をオブジェクト定義に定義する必要があります。

1. オブジェクト定義を `cinder-pv.yaml` などのファイルに保存します。

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" ❶
spec:
  capacity:
    storage: "5Gi" ❷
  accessModes:
    - "ReadWriteOnce"
  cinder: ❸
    fsType: "ext3" ❹
    volumeID: "f37a03aa-6212-4c62-a805-9ce139fab180" ❺
```

- ❶ 永続ボリューム要求 (PVC) または Pod によって使用されるボリュームの名前。
- ❷ このボリュームに割り当てられるストレージの量。
- ❸ ボリュームタイプ。今回の場合は `cinder` です。
- ❹ マウントするファイルシステムタイプです。
- ❺ 使用する Cinder ボリューム



#### 重要

ボリュームをフォーマットしてプロビジョニングした後は、**fstype** パラメータの値は変更しないでください。この値を変更すると、データの損失や、Pod の障害につながる可能性があります。

2. 永続ボリュームを作成します。

```
# oc create -f cinder-pv.yaml

persistentvolume "pv0001" created
```

3. 永続ボリュームの存在を確認します。

```
# oc get pv

NAME          LABELS          CAPACITY  ACCESSMODES  STATUS   CLAIM   REASON
AGE
pv0001       <none>         5Gi      RWO          Available             2s
```

次に、ユーザーは [Persistent Volume Claim \(永続ボリューム要求\)](#) を使用してストレージを要求し、この新規の永続ボリュームを活用できます。



## 重要

Persistent Volume Claim (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリューム要求にアクセスしようとすると、Pod にエラーが発生します。

### 27.4.2.2. Cinder の PV 形式

OpenShift Container Platform は、ボリュームをマウントしてコンテナに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

これにより、OpenShift Container Platform がフォーマットされていない Cinder ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用することが可能になります。

### 27.4.2.3. Cinder ボリュームのセキュリティー

お使いのアプリケーションで Cinder PV を使用する場合に、そのデプロイメント設定にセキュリティーを追加します。



## 注記

Cinder ボリュームを実装する前に、[ボリュームのセキュリティー](#)を確認します。

1. 適切な **fsGroup** ストラテジーを使用する **SCC** を作成します。
2. サービスアカウントを作成して、そのアカウントを SCC に追加します。

```
[source,bash]
$ oc create serviceaccount <service_account>
$ oc adm policy add-scc-to-user <new_scc> -z <service_account> -n <project>
```

3. アプリケーションのデプロイ設定で、サービスアカウント名と **securityContext** を指定します。

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1 1
  selector: 2
    name: frontend
  template: 3
    metadata:
      labels: 4
        name: frontend 5
    spec:
      containers:
        - image: openshift/hello-openshift
          name: helloworld
          ports:
            - containerPort: 8080
```

```

protocol: TCP
restartPolicy: Always
serviceAccountName: <service_account> ❹
securityContext:
  fsGroup: 7777 ❺

```

- ❶ 実行する Pod のコピー数です。
- ❷ 実行する Pod のラベルセレクターです。
- ❸ コントローラーが作成する Pod のテンプレートです。
- ❹ Pod のラベルには、ラベルセレクターからのラベルが含まれている必要があります。
- ❺ パラメーターの拡張後の名前の最大長さは 63 文字です。
- ❻ 作成したサービスアカウントを指定します。
- ❼ Pod の `fsGroup` を指定します。

#### 27.4.2.4. cinder ボリュームの制限

デフォルトでは、クラスターの各ノードに最大 256 個の Cinder ボリュームを割り当てることができます。この制限を変更するには、以下を実行します。

1. `KUBE_MAX_PD_VOLLS` 環境変数を整数に設定します。たとえば、`/etc/origin/master/master.env` で以下を実行します。

```
KUBE_MAX_PD_VOLLS=26
```

2. コマンドラインから API サービスを再起動します。

```
# master-restart api
```

3. コマンドラインからコントローラーサービスを再起動します。

```
# master-restart controllers
```

## 27.5. CEPH RADOS ブロックデバイス (RBD) を使用した永続ストレージ

### 27.5.1. 概要

OpenShift Container Platform クラスターでは、Ceph RBD を使用して [永続ストレージ](#) をプロビジョニングできます。

永続ボリューム (PV) と [Persistent Volume Claim \(永続ボリューム要求、PVC\)](#) は、単一プロジェクトでボリュームを共有できます。PV 定義に含まれている Ceph RBD 固有の情報は Pod 定義で直接定義することも可能ですが、この方法だとボリュームが一意的なクラスターリソースとして作成されず、競合の影響を受けやすくなります。

このトピックでは、OpenShift Container Platform と [Ceph RBD](#) についてある程度理解していることを前提とします。OpenShift Container Platform 永続ボリューム (PV) の一般的なフレームワークについての詳細は、[永続ストレージ](#) の概念に関するトピックを参照してください。



## 注記

本書では、プロジェクトと namespace は区別せずに使用されています。両者の関係については、[Projects and Users](#) を参照してください。



## 重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

### 27.5.2. プロビジョニング

Ceph ボリュームをプロビジョニングするには、以下が必要になります。

- 基礎となるインフラストラクチャーの既存ストレージデバイス。
- OpenShift Container Platform シークレットオブジェクトで使用される Ceph キー。
- Ceph イメージ名。
- ブロックストレージ上部のファイルシステムタイプ (ext4 など)。
- クラスタ内のスケジュール可能な各 OpenShift Container Platform ノードにインストールされた **ceph-common**。

```
# yum install ceph-common
```

#### 27.5.2.1. Ceph シークレットの作成

承認キーをシークレット設定に定義します。これは後に OpenShift Container Platform で使用できるように base64 に変換されます。



## 注記

Ceph ストレージを使用して永続ボリュームをサポートするには、シークレットを PVC や Pod と同じプロジェクトに作成する必要があります。シークレットは、単にデフォルトプロジェクトに置くことはできません。

1. Ceph MON ノードで **ceph auth get-key** を実行し、**client.admin** ユーザーのキー値を表示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
data:
  key: QVFBOFF2SIZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ==
type: kubernetes.io/rbd
```

2. シークレット定義を **ceph-secret.yaml** などのファイルに保存し、シークレットを作成します。

```
$ oc create -f ceph-secret.yaml
```

- シークレットが作成されたことを確認します。

```
# oc get secret ceph-secret
NAME      TYPE          DATA   AGE
ceph-secret  kubernetes.io/rbd  1      23d
```

### 27.5.2.2. 永続ボリュームの作成

開発者は、PVC を参照するか、Pod 仕様の **volumes** セクションにある Gluster ボリュームプラグインを直接参照することによって Ceph RBD ストレージを要求します。PVC は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

- OpenShift Container Platform に PV を作成する前に、PV をオブジェクト定義に定義します。

#### 例27.3 Ceph RBD を使用した永続ボリュームオブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-pv ①
spec:
  capacity:
    storage: 2Gi ②
  accessModes:
    - ReadWriteOnce ③
  rbd: ④
    monitors: ⑤
      - 192.168.122.133:6789
    pool: rbd
    image: ceph-image
    user: admin
    secretRef:
      name: ceph-secret ⑥
    fsType: ext4 ⑦
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

- Pod 定義で参照されるか、または各種の **oc** ボリュームコマンドで表示される PV の名前。
- このボリュームに割り当てられるストレージの量。
- accessModes** は、PV と PVC を一致させるためのラベルとして使用されます。現時点で、これらはいずれの形態のアクセス制御も定義しません。ブロックストレージはすべて、単一ユーザーに対して定義されます (非共有ストレージ)。
- 使用されるボリュームタイプ。この場合は、**rbd** プラグインです。
- Ceph モニターの IP アドレスとポートの配列。
- OpenShift Container Platform から Ceph サーバーへのセキュアな接続を作成するために使用される Ceph シークレット。
- Ceph RBD ブロックデバイスにマウントされるファイルシステムタイプ。



## 重要

ボリュームをフォーマットしてプロビジョニングした後に **fstype** パラメーターの値を変更すると、データ損失や Pod エラーが発生する可能性があります。

2. 定義を `ceph-pv.yaml` などのファイルに保存し、PV を作成します。

```
# oc create -f ceph-pv.yaml
```

3. 永続ボリュームが作成されたことを確認します。

```
# oc get pv
NAME          LABELS  CAPACITY  ACCESSMODES  STATUS  CLAIM
REASON  AGE
ceph-pv      <none>  2147483648  RWO          Available  ceph-pv
```

4. 新規 PV にバインドされる PVC を作成します。

### 例27.4 PVC オブジェクト定義

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim
spec:
  accessModes: 1
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi 2
```

**1** **accessModes** はアクセス権を実施しません。代わりに PV を PVC に一致させるラベルとして機能します。

**2** この要求は、**2Gi** 以上の容量がある PV を検索します。

5. 定義をファイル (`ceph-claim.yaml` など) に保存し、PVC を作成します。

```
# oc create -f ceph-claim.yaml
```

## 27.5.3. Ceph ボリュームのセキュリティー



### 注記

Ceph RBD ボリュームを実装する前に、[ボリュームのセキュリティー](#) トピックの詳細を参照してください。

共有ボリューム (NFS および GlusterFS) とブロックボリューム (Ceph RBD、iSCSI、およびほとんどの

クラウドストレージ)の大きな違いは、Pod 定義またはコンテナイメージで定義されたユーザー ID とグループ ID がターゲットの物理ストレージに適用されることです。これはブロックデバイスの所有権の管理と呼ばれます。たとえば、Ceph RBD マウントで所有者が 123 に、グループ ID が 567 に設定されていて、Pod で `runAsUser` が 222 に、`fsGroup` が 7777 に定義されている場合、Ceph RBD 物理マウントの所有権は 222:7777 に変更されます。



### 注記

ユーザー ID とグループ ID が Pod 仕様で定義されていない場合でも、生成される Pod では、これらの ID のデフォルト値が一致する SCC またはプロジェクトに基づいて定義されることがあります。[ボリュームのセキュリティー](#) トピックでは、SCC のストレージの側面とデフォルト値について説明しています。

Pod の `securityContext` 定義で `fsGroup` スタンザを使用して、Pod で Ceph RBD ボリュームのグループ所有権を定義します。

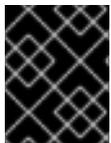
```
spec:
  containers:
    - name:
      ...
      securityContext: ①
        fsGroup: 7777 ②
```

- ① `securityContext` は特定のコンテナの配下ではなく、この Pod レベルで定義します。
- ② Pod 内のすべてのコンテナは同じ `fsGroup` ID を持ちます。

## 27.6. AWS ELASTIC BLOCK STORE を使用した永続ストレージ

### 27.6.1. 概要

OpenShift Container Platform は AWS Elastic Block Store volumes (EBS) をサポートします。[AWS EC2](#) を使用して、OpenShift Container Platform クラスターに [永続ストレージ](#) をプロビジョニングできます。これには、Kubernetes および AWS についてのある程度の理解があることが前提となります。



### 重要

AWS を使用して永続ボリュームを作成する前に、まず OpenShift Container Platform を [AWS ElasticBlockStore 用に適切に設定](#) する必要があります。

Kubernetes [永続ボリューム](#) フレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。AWS Elastic Block Store ボリュームは [動的にプロビジョニング](#) できます。永続ボリュームは、単一のプロジェクトまたは namespace にバインドされず、OpenShift Container Platform クラスター全体で共有できます。ただし、[Persistent Volume Claim \(永続ボリューム要求\)](#) は、プロジェクトまたは namespace に固有で、ユーザーが要求できます。



### 重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

## 27.6.2. プロビジョニング

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。OpenShift Container Platform が [AWS Elastic Block Store 用に設定されている](#) ことを確認した後、OpenShift と AWS に必要になるのは、AWS EBS ボリューム ID と **PersistentVolume** API のみです。

### 27.6.2.1. 永続ボリュームの作成

OpenShift Container Platform に永続ボリュームを作成する前に、永続ボリュームをオブジェクト定義で定義する必要があります。

#### 例27.5 AWS を使用した永続ボリュームオブジェクトの定義

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" ①
spec:
  capacity:
    storage: "5Gi" ②
  accessModes:
    - "ReadWriteOnce"
  awsElasticBlockStore: ③
    fsType: "ext4" ④
    volumeID: "vol-f37a03aa" ⑤
```

- ① ボリュームの名前。これは [Persistent Volume Claim \(永続ボリューム要求\)](#) を使用して、または Pod からボリュームを識別するために使用されます。
- ② このボリュームに割り当てられるストレージの量。
- ③ これは使用されるボリュームタイプを定義します。この場合は、`awsElasticBlockStore` プラグインです。
- ④ マウントするファイルシステムタイプです。
- ⑤ これは使用される AWS ボリュームです。



#### 重要

ボリュームをフォーマットしてプロビジョニングした後に `fstype` パラメーターの値を変更すると、データ損失や Pod エラーが発生する可能性があります。

定義を `aws-pv.yaml` などのファイルに保存し、永続ボリュームを作成します。

```
# oc create -f aws-pv.yaml
persistentvolume "pv0001" created
```

永続ボリュームが作成されたことを確認します。

```
# oc get pv
NAME      LABELS   CAPACITY  ACCESSMODES  STATUS   CLAIM    REASON  AGE
pv0001   <none>   5Gi       RWO           Available  2s
```

次に、ユーザーは [Persistent Volume Claim \(永続ボリューム要求\)](#) を使用してストレージを要求し、この新規の永続ボリュームを活用できます。



### 重要

Persistent Volume Claim (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

#### 27.6.2.2. ボリュームのフォーマット

OpenShift Container Platform は、ボリュームをマウントしてコンテナに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

これにより、OpenShift Container Platform がフォーマットされていない AWS ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用することが可能になります。

#### 27.6.2.3. ノード上の EBS ボリュームの最大数

OpenShift Container Platform では、デフォルトで1つのノードに最大 39 の EBS ボリュームを割り当てることができます。この制限は、[AWS ボリュームの制限](#) に合致します。

OpenShift Container Platform では、環境変数 **KUBE\_MAX\_PD\_VOLS** を設定することで、より大きな制限値を設定できます。ただし、AWS では、割り当てられるデバイスに特定の命名スキーム ([AWS デバイスの命名](#)) を使用する必要があります。この命名スキームでは、最大で 52 のボリュームしかサポートされません。これにより、OpenShift Container Platform 経由でノードに割り当てることができるボリュームの数が 52 に制限されます。

## 27.7. GCE PERSISTENT DISK を使用した永続ストレージ

### 27.7.1. 概要

OpenShift Container Platform では、GCE Persistent Disk ボリューム (gcePD) がサポートされます。[GCE](#) を使用して、OpenShift Container Platform クラスターに [永続ストレージ](#) をプロビジョニングできます。これには、Kubernetes と GCE についてある程度の理解があることが前提となります。



### 重要

GCE を使用して永続ボリュームを作成する前に、まず OpenShift Container Platform を [GCE Persistent Disk 用に適切に設定](#) する必要があります。

Kubernetes [永続ボリューム](#) フレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。GCE Persistent Disk ボリュームは [動的にプロビジョニング](#) できます。永続ボリュームは、単一のプロジェクトまたは namespace にバインドされず、OpenShift Container Platform クラスター全体で共有できます。ただし、[Persistent Volume Claim \(永続ボリューム要求\)](#) は、プロジェクトまたは namespace に固有で、ユーザーが要求できます。



## 重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

### 27.7.2. プロビジョニング

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。OpenShift Container Platform が [GCE Persistent Disk 用に設定されている](#) ことを確認した後、OpenShift Container Platform と GCE に必要となるのは、GCE Persistent Disk ボリューム ID と **PersistentVolume** API のみです。

#### 27.7.2.1. 永続ボリュームの作成

OpenShift Container Platform に永続ボリュームを作成する前に、永続ボリュームをオブジェクト定義で定義する必要があります。

##### 例27.6 GCE を使用した永続ボリュームオブジェクトの定義

```

apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" ①
spec:
  capacity:
    storage: "5Gi" ②
  accessModes:
    - "ReadWriteOnce"
  gcePersistentDisk: ③
  fsType: "ext4" ④
  pdName: "pd-disk-1" ⑤

```

- ① ボリュームの名前。これは [Persistent Volume Claim \(永続ボリューム要求\)](#) を使用して、または Pod からボリュームを識別するために使用されます。
- ② このボリュームに割り当てられるストレージの量。
- ③ これは使用されるボリュームタイプを定義します。この場合は、`gcePersistentDisk` プラグインです。
- ④ マウントするファイルシステムタイプです。
- ⑤ これは使用される GCE Persistent Disk ボリュームです。



## 重要

ボリュームをフォーマットしてプロビジョニングした後に `fstype` パラメーターの値を変更すると、データ損失や Pod にエラーが発生する可能性があります。

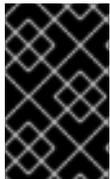
定義を `gce-pv.yaml` などのファイルに保存し、永続ボリュームを作成します。

```
# oc create -f gce-pv.yaml
persistentvolume "pv0001" created
```

永続ボリュームが作成されたことを確認します。

```
# oc get pv
NAME      LABELS  CAPACITY  ACCESSMODES  STATUS   CLAIM    REASON  AGE
pv0001    <none>  5Gi      RWO          Available             2s
```

次に、ユーザーは [Persistent Volume Claim \(永続ボリューム要求\)](#) を使用してストレージを要求し、この新規の永続ボリュームを活用できます。



### 重要

Persistent Volume Claim (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

#### 27.7.2.2. ボリュームのフォーマット

OpenShift Container Platform は、ボリュームをマウントしてコンテナに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

これにより、OpenShift Container Platform がフォーマットされていない GCE ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用することが可能になります。

## 27.8. iSCSI を使用した永続ストレージ

### 27.8.1. 概要

iSCSI を使用して、OpenShift Container Platform クラスターに [永続ストレージ](#) をプロビジョニングできます。これには、Kubernetes と iSCSI についてある程度の理解があることが前提となります。

Kubernetes [永続ボリューム](#) フレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。



### 重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

### 27.8.2. プロビジョニング

OpenShift Container Platform でストレージをボリュームとしてマウントする前に、基礎となるインフラストラクチャーにストレージが存在することを確認します。iSCSI に必要なのは、iSCSI ターゲットポータル、有効な iSCSI 修飾名 (IQN)、有効な LUN 番号、ファイルシステムタイプ、および **PersistentVolume** API のみです。

オプションで、マルチパスポータルと CHAP (チャレンジハンドシェイク認証プロトコル) 設定を指定できます。

## 例27.7 永続ボリュームオブジェクトの定義

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.16.154.81:3260
    portals: ['10.16.154.82:3260', '10.16.154.83:3260']
    iqn: iqn.2014-12.example.server:storage.target00
    lun: 0
    fsType: 'ext4'
    readOnly: false
    chapAuthDiscovery: true
    chapAuthSession: true
    secretRef:
      name: chap-secret

```

## 27.8.2.1. ディスククォータの実施

LUN パーティションを使用してディスククォータとサイズ制限を実施します。それぞれの LUN には 1 つの永続ボリュームです。Kubernetes では、永続ボリュームに一意の名前を使用する必要があります。

上記の方法でクォータを実施すると、エンドユーザーは永続ストレージを具体的な量 (10Gi など) で要求することができ、これを同等またはそれ以上の容量の対応するボリュームに一致させることができます。

## 27.8.2.2. iSCSI ボリュームのセキュリティー

ユーザーは **PersistentVolumeClaim** でストレージを要求します。この要求はユーザーの namespace にのみ存在し、同じ namespace 内の Pod からのみ参照できます。namespace をまたいで永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

それぞれの iSCSI LUN は、クラスター内のすべてのノードからアクセスできる必要があります。

## 27.8.2.3. iSCSI のマルチパス化

iSCSI ベースのストレージの場合は、複数のターゲットポータルの IP アドレスに同じ IQN を使用することでマルチパスを設定できます。マルチパス化により、パス内の 1 つ以上のコンポーネントで障害が発生した場合でも、永続ボリュームにアクセスすることができます。

Pod 仕様でマルチパスを指定するには、**portals** フィールドを使用します。以下に例を示します。

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv

```

```
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260'] ❶
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    fsType: ext4
    readOnly: false
```

- ❶ **portals** フィールドを使用してターゲットポータルを追加します。

#### 27.8.2.4. iSCSI のカスタムイニシエーター IQN

iSCSI ターゲットが特定に IQN に制限されている場合に、カスタムイニシエーターの iSCSI Qualified Name (IQN) を設定します。ただし、iSCSI PV が割り当てられているノードが必ずこれらの IQN を使用する保証はありません。

カスタムのイニシエーター IQN を指定するには、**initiatorName** フィールドを使用します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260']
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    initiatorName: iqn.2016-04.test.com:custom.iqn ❶
    fsType: ext4
    readOnly: false
```

- ❶ カスタムのイニシエーター IQN を追加するには、**initiatorName** フィールドを使用します。

## 27.9. ファイバーチャネルを使用した永続ストレージ

### 27.9.1. 概要

**ファイバーチャネル** (FC) を使用して、OpenShift Container Platform クラスターに **永続ストレージ** をプロビジョニングできます。これには、Kubernetes と FC についてある程度の理解があることが前提となります。

Kubernetes 永続ボリューム フレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。



## 重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

### 27.9.2. プロビジョニング

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。FC 永続ストレージに必要なものは、**PersistentVolume** API、**wwids** または **lun** 数が有効な **targetWWNs**、**fsType** です。永続ボリュームと LUN は 1対1でマッピングされます。

#### 永続ボリュームオブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  fc:
    wwids: [scsi-3600508b400105e210000900000490000] ①
    targetWWNs: ['500a0981891b8dc5', '500a0981991b8dc5'] ②
    lun: 2 ③
    fsType: ext4
```

① オプション: World wide identifier (WWID) **FCwwids** または FC **targetWWNs** および **lun** の組み合わせは設定する必要がありますが、両方を同時に設定することはできません。WWN ターゲットよりも FC WWID 識別子が推奨されます。FC WWID 識別子は、各ストレージデバイスに固有のものであり、デバイスのアクセスに使用されるパスに依存しないためです。この識別子は、SCSI Inquiry を発行して Device Identification Vital Product Data (**page 0x83**) または Unit Serial Number (**page 0x80**) を取得することにより獲得できます。FC WWID は、デバイスへのパスが変更したり、別のシステムからデバイスにアクセスする場合でも、ディスク上のデータ参照に **/dev/disk/by-id/** と識別されます。

②③ オプション: World wide names (WWN) **FCwwids** または FC **targetWWNs** および **lun** の組み合わせは設定する必要がありますが、両方を同時に設定することはできません。WWN ターゲットよりも FC WWID 識別子が推奨されます。FC WWID 識別子は、各ストレージデバイスに固有のものであり、デバイスのアクセスに使用されるパスに依存しないためです。FC WWN は、**/dev/disk/by-path/pci-<identifier>-fc-0x<wwn>-lun-<lun\_#>** として識別されます。ただし、**<wwn>** までのパス (**0x** を含む) と WWN の後の文字 (**-** (ハイフン) を含む) を入力する必要はありません。



## 重要

ボリュームをフォーマットしてプロビジョニングした後に **fstype** パラメーターの値を変更すると、データ損失や Pod エラーが発生する可能性があります。

### 27.9.2.1. ディスククォータの実施

LUN パーティションを使用してディスククォータとサイズ制限を実施します。それぞれの LUN には 1 つの永続ボリュームです。Kubernetes では、永続ボリュームに一意の名前を使用する必要があります。

この方法でクォータを実施すると、エンドユーザーは永続ストレージを具体的な量 (10Gi など) で要求することができ、これを同等またはそれ以上の容量の対応するボリュームに一致させることができます。

### 27.9.2.2. ファイバーチャネルボリュームのセキュリティー

ユーザーは **PersistentVolumeClaim** でストレージを要求します。この要求はユーザーの namespace にのみ存在し、同じ namespace 内の Pod からのみ参照できます。namespace をまたいで永続ボリューム要求 (PVC) にアクセスしようとすると、Pod にエラーが発生します。

それぞれの FC LUN は、クラスター内のすべてのノードからアクセスできる必要があります。

## 27.10. AZURE DISK を使用した永続ストレージ

### 27.10.1. 概要

OpenShift Container Platform では、[Microsoft Azure Disk](#) ボリュームがサポートされます。Azure を使用して、OpenShift Container Platform クラスターに [永続ストレージ](#) をプロビジョニングできます。これには、Kubernetes と Azure についてのある程度の理解があることが前提となります。

Kubernetes [永続ボリューム](#) フレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。

Azure Disk ボリュームは [動的にプロビジョニング](#) できます。永続ボリュームは、単一のプロジェクトまたは namespace にバインドされず、OpenShift Container Platform クラスター全体で共有できます。ただし、[Persistent Volume Claim \(永続ボリューム要求\)](#) は、プロジェクトまたは namespace に固有で、ユーザーが要求できます。



#### 重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

### 27.10.2. 前提条件

Azure を使用して永続ボリュームを作成する前に、OpenShift Container Platform クラスターが以下の要件を満たしていることを確認してください。

- まず、OpenShift Container Platform を [Azure Disk 用に設定](#) する必要があります。
- インフラストラクチャー内の各ノードは、Azure 仮想マシン名に一致している必要があります。
- それぞれのノードホストは、同じリソースグループに属している必要があります。

### 27.10.3. プロビジョニング

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。OpenShift Container Platform が [Azure Disk 用に設定されている](#) ことを確認した後、OpenShift Container Platform と Azure に必要になるのは、Azure Disk Name および Disk URI と **PersistentVolume** API のみです。

#### 27.10.4. Azure Disk での地域クラウドの設定

Azure には、インスタンスをデプロイする複数のリージョンがあります。必要なリージョンを指定するには、以下を `azure.conf` ファイルに追加します。

```
cloud: <region>
```

リージョンは以下のいずれかになります。

- ドイツクラウド: **AZUREGERMANCLOUD**
- 中国クラウド: **AZURECHINACLOUD**
- パブリッククラウド: **AZUREPUBLICCLOUD**
- 米国クラウド: **AZUREUSGOVERNMENTCLOUD**

##### 27.10.4.1. 永続ボリュームの作成

OpenShift Container Platform に永続ボリュームを作成する前に、永続ボリュームをオブジェクト定義で定義する必要があります。

###### 例27.8 Azure を使用した永続ボリュームオブジェクトの定義

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" ①
spec:
  capacity:
    storage: "5Gi" ②
  accessModes:
    - "ReadWriteOnce"
  azureDisk: ③
    diskName: test2.vhd ④
    diskURI: https://someaccount.blob.core.windows.net/vhds/test2.vhd ⑤
    cachingMode: ReadWrite ⑥
    fsType: ext4 ⑦
    readOnly: false ⑧
```

- ① ボリュームの名前。これは [Persistent Volume Claim \(永続ボリューム要求\)](#) を使用して、または Pod からボリュームを識別するために使用されます。
- ② このボリュームに割り当てられるストレージの量。
- ③ これは使用されるボリュームタイプを定義します (この例では、`azureDisk` プラグイン)。
- ④ Blob ストレージのデータディスクの名前。

- 5 Blob ストレージのデータディスクの URI
- 6 ホストのキャッシングモード: None、ReadOnly、または ReadWrite。
- 7 マウントするファイルシステムタイプ (**ext4**、**xfs** など)。
- 8 デフォルトは **false** (読み取り/書き込み) です。ここで **ReadOnly** を指定すると、**VolumeMounts** で **ReadOnly** 設定が強制的に実行されます。



### 重要

ボリュームをフォーマットしてプロビジョニングした後に **fsType** パラメーターの値を変更すると、データ損失や Pod にエラーが発生する可能性があります。

1. 定義を **azure-pv.yaml** などのファイルに保存し、永続ボリュームを作成します。

```
# oc create -f azure-pv.yaml
persistentvolume "pv0001" created
```

2. 永続ボリュームが作成されたことを確認します。

```
# oc get pv
NAME          LABELS          CAPACITY  ACCESSMODES  STATUS   CLAIM          REASON  AGE
pv0001        <none>         5Gi       RWO           Available  2s
```

これで、[Persistent Volume Claim \(永続ボリューム要求\)](#) を使用してストレージを要求し、新規の永続ボリュームを活用できるようになります。



### 重要

Azure Disk PVC を介してボリュームがマウントされている Pod の場合、新規ノードへの Pod のスケジューリングに数分の時間がかかります。**ディスクの割り当て解除操作**が完了するまで 2～3 分待ってから、新規デプロイメントを開始してください。**ディスクの割り当て解除操作**が完了する前に新規の Pod の作成要求が開始されると、Pod の作成によって開始された**ディスクの割り当て**操作が失敗し、結果として Pod の作成が失敗します。



### 重要

Persistent Volume Claim (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

## 27.10.4.2. ボリュームのフォーマット

OpenShift Container Platform は、ボリュームをマウントしてコンテナに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

これにより、OpenShift Container Platform がフォーマットされていない Azure ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用することが可能になります。

## 27.11. AZURE FILE を使用した永続ストレージ

### 27.11.1. 概要

OpenShift Container Platform では、[Microsoft Azure File](#) ボリュームがサポートされます。Azure を使用して、OpenShift Container Platform クラスタに [永続ストレージ](#) をプロビジョニングできます。これには、Kubernetes と Azure についてのある程度の理解があることが前提となります。



#### 重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

### 27.11.2. 作業を開始する前に

1. すべてのノードに **samba-client**、**samba-common**、および **cifs-utils** をインストールします。

```
$ sudo yum install samba-client samba-common cifs-utils
```

2. すべてのノードで SELinux ブール値を有効にします。

```
$ /usr/sbin/setsebool -P virt_use_samba on
$ /usr/sbin/setsebool -P virt_sandbox_use_samba on
```

3. **mount** コマンドを実行して **dir\_mode** および **file\_mode** パーミッションなどを確認します。

```
$ mount
```

**dir\_mode** および **file\_mode** のパーミッションが **0755** に設定されている場合には、デフォルト値 **0755** を **0777** または **0775** に変更します。OpenShift Container Platform 3.9 では、デフォルトの **dir\_mode** および **file\_mode** パーミッションが **0777** から **0755** に変更されるので、この手動の手順が必要です。以下の例では、変更された値が含まれる設定ファイルを紹介しています。

#### Azure File を使用する場合の考慮事項

以下のファイルシステム機能は Azure File ではサポートされません。

- シンボリックリンク
- ハードリンク
- 拡張属性
- スパースファイル
- 名前付きパイプ

また、Azure File がマウントされるディレクトリーの所有者 ID (UID) は、コンテナのプロセス UID とは異なります。

## 注意

サポートされていないファイルシステム機能を使用するコンテナイメージを使用した場合、環境が不安定になる可能性があります。PostgreSQL および MySQL のコンテナには Azure File で使用すると問題が生じることが確認されています。

## MySQL と Azure File を使用するための回避策

MySQL コンテナを使用する場合、回避策として、マウントされたディレクトリー UID とコンテナプロセス UID 間のファイル所有者の不一致に対して PV 設定を変更する必要があります。PV 設定ファイルに対して以下の変更を実行します。

1. PV 設定ファイルの **runAsUser** 変数に、Azure File のマウントされたディレクトリー UID を指定します。

```
spec:
  containers:
    ...
  securityContext:
    runAsUser: <mounted_dir_uid>
```

2. PV 設定ファイルの **mountOptions** でコンテナのプロセス UID を指定します。

```
mountOptions:
  - dir_mode=0700
  - file_mode=0600
  - uid=<container_process_uid>
  - gid=0
```

### 27.11.3. 設定ファイルのサンプル

以下の設定ファイルのサンプルは、Azure File を使用した PV 設定を表示しています。

#### PV 設定ファイル例

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "azpv"
spec:
  capacity:
    storage: "1Gi"
  accessModes:
    - "ReadWriteMany"
  azureFile:
    secretName: azure-secret
    shareName: azftest
    readOnly: false
  mountOptions:
    - dir_mode=0777
    - file_mode=0777
```

以下の設定ファイルのサンプルは、Azure File を使用したストレージクラスを表示しています。

## ストレージクラスの設定ファイル例

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azurefile
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0777
parameters:
  storageAccount: ocp39str
  location: centralus
```

### 27.11.4. Azure File でのリージョンクラウドの設定

Azure Disk は複数の地域クラウドに対応していますが、Azure File は、エンドポイントがハードコードされているために Azure パブリッククラウドのみをサポートしています。

### 27.11.5. Azure Storage Account シークレットの作成

Azure Storage Account の名前とキーをシークレット設定に定義します。これは後に OpenShift Container Platform で使用できるように base64 に変換されます。

1. Azure Storage Account の名前とキーを取得し、base64 にエンコードします。

```
apiVersion: v1
kind: Secret
metadata:
  name: azure-secret
type: Opaque
data:
  azurestorageaccountname: azhzdGVzdA==
  azurestorageaccountkey:
  eEIGMXpKYm5ub2pGTE1Ta0JwNTBteDAyckhzTUusyc2pVN21GdDRMMTNob0I3ZHJBYUo4a
  kQ2K0E0NDNqSm9nVjd5MkZVT2hRQ1dQbU02WWFOSHk3cWc9PQ==
```

2. シークレット定義を `azure-secret.yaml` などのファイルに保存し、シークレットを作成します。

```
$ oc create -f azure-secret.yaml
```

3. シークレットが作成されたことを確認します。

```
$ oc get secret azure-secret
NAME          TYPE      DATA   AGE
azure-secret  Opaque   1       23d
```

4. OpenShift Container Platform に PV を作成する前に、PV をオブジェクト定義に定義します。

#### Azure File を使用した PV オブジェクト定義の例

```
apiVersion: "v1"
```

```

kind: "PersistentVolume"
metadata:
  name: "pv0001" ❶
spec:
  capacity:
    storage: "5Gi" ❷
  accessModes:
    - "ReadWriteMany"
  azureFile: ❸
    secretName: azure-secret ❹
    shareName: example ❺
    readOnly: false ❻

```

- ❶ ボリュームの名前。これを使用して、[PV Claim \(永続ボリューム要求\)](#) で、または Pod から、ボリュームを識別する必要があります。
- ❷ このボリュームに割り当てられるストレージの量。
- ❸ これは使用されるボリュームタイプを定義します ([azureFile プラグイン](#))。
- ❹ 使用されるシークレットの名前。
- ❺ ファイル共有の名前。
- ❻ デフォルトは **false** (読み取り/書き込み) です。ここで **ReadOnly** を指定すると、**VolumeMounts** で **ReadOnly** 設定が強制的に実行されます。

5. 定義を `azure-file-pv.yaml` などのファイルに保存し、PV を作成します。

```

$ oc create -f azure-file-pv.yaml
persistentvolume "pv0001" created

```

6. PV が作成されたことを確認します。

```

$ oc get pv
NAME      LABELS    CAPACITY  ACCESSMODES  STATUS   CLAIM    REASON  AGE
pv0001    <none>    5Gi       RWM          Available             2s

```

これで、[PVC \(永続ボリューム要求\)](#) を使用してストレージを要求し、新規の永続ボリュームを活用できるようになります。



### 重要

PV Claim (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

## 27.12. FLEXVOLUME プラグインを使用した永続ストレージ

### 27.12.1. 概要

OpenShift Container Platform には、各種のストレージテクノロジーを使用するために [ボリュームプラグイン](#) が組み込まれています。組み込みプラグインがないバックエンドのストレージを使用する場合は、FlexVolume ドライバーを使用して OpenShift Container Platform を拡張し、アプリケーションに [永続ストレージ](#) を提供できます。

## 27.12.2. FlexVolume ドライバー

FlexVolume ドライバーは、クラスター内のすべてのマシン (マスターとノードの両方) の明確に定義されたディレクトリーに格納されている実行可能ファイルです。OpenShift Container Platform は、**flexVolume** をソースとする **PersistentVolume** によって表されるボリュームの割り当て、割り当て解除、マウント、またはアンマウントが必要になるたびに FlexVolume ドライバーを呼び出します。

ドライバーの最初のコマンドライン引数は常に操作名です。その他のパラメーターは操作ごとに異なります。ほとんどの操作は、JSON (JavaScript Object Notation) 文字列をパラメーターとして取ります。このパラメーターは完全な JSON 文字列であり、JSON データを含むファイルの名前ではありません。

FlexVolume ドライバーには以下が含まれます。

- すべての **flexVolume.options**。
- **kubernetes.io/** という接頭辞が付いた **flexVolume** のいくつかのオプション。たとえば、**fsType** や **readOnly** などです。
- **kubernetes.io/secret/** という接頭辞が付いた参照先シークレット (指定されている場合) の内容。

### FlexVolume ドライバーの JSON 入力例

```
{
  "fooServer": "192.168.0.1:1234", ①
  "fooVolumeName": "bar",
  "kubernetes.io/fsType": "ext4", ②
  "kubernetes.io/readonly": "ro", ③
  "kubernetes.io/secret/<key name>": "<key value>", ④
  "kubernetes.io/secret/<another key name>": "<another key value>",
}
```

- ① **flexVolume.options** のすべてのオプション。
- ② **flexVolume.fsType** の値。
- ③ **flexVolume.readOnly** に基づく **ro/rw**。
- ④ **flexVolume.secretRef** によって参照されるシークレットのすべてのキーと値。

OpenShift Container Platform は、ドライバーの標準出力に JSON データが含まれていると想定します。指定されていない場合、出力には操作の結果が示されます。

### FlexVolume ドライバーのデフォルトの出力

```
{
  "status": "<Success/Failure/Not supported>",
  "message": "<Reason for success/failure>"
}
```

ドライバーの終了コードは、成功の場合は **0**、エラーの場合は **1** です。

操作はべき等です。つまり、すでに割り当てられているボリュームの割り当て操作や、すでにマウントされているボリュームのマウント操作は成功します。

FlexVolume ドライバーは以下の 2 つのモードで動作します。

- [マスター実行の割り当て/割り当て解除操作あり](#)
- [マスター実行の割り当て/割り当て解除操作なし](#)

**attach/detach** 操作は、OpenShift Container Platform マスターにより、ノードにボリュームを割り当てるため、およびノードからボリュームの割り当てを解除するために使用されます。これは何らかの理由でノードが応答不能になった場合に役立ちます。その後、マスターはノード上のすべての Pod を強制終了し、ノードからすべてのボリュームの割り当てを解除して、ボリュームを他のノードに割り当てることで、元のノードがまだ到達不能な状態であってもアプリケーションを再開できます。



### 重要

マスター実行の、別のマシンからのボリュームの割り当て解除は、すべてのストレージバックエンドでサポートされる訳ではありません。

#### 27.12.2.1. マスター実行の割り当て/割り当て解除がある FlexVolume ドライバー

マスター制御の割り当て/割り当て解除をサポートする FlexVolume ドライバーは、以下の操作を実装する必要があります。

##### init

ドライバーを初期化します。マスターとノードの初期化中に呼び出されます。

- 引数: なし
- 実行場所: マスター、ノード
- 予期される出力: デフォルトの JSON

##### getvolumename

ボリュームの一意の名前を返します。この名前は、後続の **detach** 呼び出しで **<volume-name>** として使用されるため、すべてのマスターとノード間で一致する必要があります。**<volume-name>** の / 文字は自動的に ~ に置き換えられます。

- 引数: **<json>**
- 実行場所: マスター、ノード
- 予期される出力: デフォルトの JSON + **volumeName**:

```
{
  "status": "Success",
  "message": "",
  "volumeName": "foo-volume-bar" ❶
}
```

- ❶ ストレージバックエンド **foo** のボリュームの一意の名前。

## attach

指定されたノードに、JSON で表現したボリュームを割り当てます。この操作は、ノード上のデバイスが既知の場合 (つまり、そのデバイスが実行前にストレージバックエンドによって割り当て済みの場合)、そのデバイスの名前を返します。デバイスが既知でない場合は、後続の **waitforattach** 操作によってノード上のデバイスが検出される必要があります。

- 引数: **<json> <node-name>**
- 実行場所: マスター
- 予期される出力: デフォルトの JSON + **device** (既知の場合)。

```
{
  "status": "Success",
  "message": "",
  "device": "/dev/xvda" ❶
}
```

- ❶ ノード上のデバイスの名前 (既知の場合)。

## waitforattach

ボリュームがノードに完全に割り当てられ、デバイスが出現するまで待機します。前の **attach** 操作から **<device-name>** が返された場合は、それが入力パラメーターとして渡されます。そうでない場合、**<device-name>** は空であり、この操作によってノード上のデバイスを検出する必要があります。

- 引数: **<device-name> <json>**
- 実行場所: ノード
- 予期される出力: デフォルトの JSON + **device**

```
{
  "status": "Success",
  "message": "",
  "device": "/dev/xvda" ❶
}
```

- ❶ ノード上のデバイスの名前。

## detach

ノードから、指定されたボリュームの割り当てを解除します。**<volume-name>** は、**getvolumename** 操作によって返されるデバイスの名前です。**<volume-name>** の / 文字は自動的に ~ に置き換えられます。

- 引数: **<volume-name> <node-name>**
- 実行場所: マスター
- 予期される出力: デフォルトの JSON

## isattached

ボリュームがノードに割り当てられていることを確認します。

- 引数: `<json> <node-name>`
- 実行場所: マスター
- 予期される出力: デフォルトの JSON + **attached**

```
{
  "status": "Success",
  "message": "",
  "attached": true ①
}
```

- ① ノードへのボリュームの割り当てのステータス。

### mountdevice

ボリュームのデバイスをディレクトリーにマウントします。`<device-name>` は、前の `waitforattach` 操作で返されるデバイスの名前です。

- 引数: `<mount-dir> <device-name> <json>`
- 実行場所: ノード
- 予期される出力: デフォルトの JSON

### unmountdevice

ボリュームのデバイスをディレクトリーからアンマウントします。

- 引数: `<mount-dir>`
- 実行場所: ノード

その他のすべての操作は、`{"status": "Not supported"}` と終了コード **1** を出して JSON を返します。



### 注記

マスターで実行される割り当て/割り当て解除操作はデフォルトで有効にされています。有効にされていない場合、割り当て/割り当て解除操作は、ボリュームの割り当て/割り当て解除が必要なノードで実行されます。FlexVolume ドライバー呼び出しの構文およびすべてのパラメーターはどちらの場合も同じ内容です。

### 27.12.2.2. マスター実行の割り当て/割り当て解除がない FlexVolume ドライバー

マスター制御の割り当て/割り当て解除をサポートしない FlexVolume ドライバーは、ノードでのみ実行され、以下の操作を実装する必要があります。

#### init

ドライバーを初期化します。すべてのノードの初期化中に呼び出されます。

- 引数: なし
- 実行場所: ノード

- 予期される出力: デフォルトの JSON

### mount

ボリュームをディレクトリーにマウントします。これには、ノードへのボリュームの割り当て、ノードのデバイスの検出、その後のデバイスのマウントを含む、ボリュームのマウントに必要なあらゆる操作が含まれます。

- 引数: `<mount-dir> <json>`
- 実行場所: ノード
- 予期される出力: デフォルトの JSON

### unmount

ボリュームをディレクトリーからアンマウントします。これには、ノードからのボリュームの割り当て解除など、アンマウント後のボリュームのクリーンアップに必要なあらゆる操作が含まれます。

- 引数: `<mount-dir>`
- 実行場所: ノード
- 予期される出力: デフォルトの JSON

その他のすべての操作は、`{"status": "Not supported"}` と終了コード **1** を出して JSON を返します。

## 27.12.3. FlexVolume ドライバーのインストール

FlexVolume ドライバーをインストールします。

1. この実行可能ファイルがクラスター内のすべてのマスターとノードに存在することを確認します。
2. この実行可能ファイルをボリュームプラグインのパス (`/usr/libexec/kubernetes/kubelet-plugins/volume/exec/<vendor>~<driver>/<driver>`) に配置します。

たとえば、ストレージ **foo** の FlexVolume ドライバーをインストールするには、実行可能ファイルを `/usr/libexec/kubernetes/kubelet-plugins/volume/exec/openshift.com~foo/foo` に配置します。

OpenShift Container Platform 3.11 では、**controller-manager** が静的 Pod として実行されるので、割り当ておよび割り当て解除の操作を行う FlexVolume バイナリーファイルは、外部の依存関係がなく、独立する実行可能ファイルでなければなりません。

Atomic ホストでは、FlexVolume プラグインディレクトリーのデフォルトの場所は `/etc/origin/kubelet-plugins/` です。FlexVolume の実行可能ファイルを、クラスター内の全マスターとノードの `/etc/origin/kubelet-plugins/volume/exec/<vendor>~<driver>/<driver>` ディレクトリーに配置する必要があります。

## 27.12.4. FlexVolume ドライバーを使用したストレージの使用

インストールされているストレージを参照するには、**PersistentVolume** オブジェクトを使用します。OpenShift Container Platform の各 **PersistentVolume** オブジェクトは、ストレージバックエンドの 1 つのストレージアセット (通常はボリューム) を表します。

## FlexVolume ドライバーを使用した永続ボリュームのオブジェクト定義例

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 ❶
spec:
  capacity:
    storage: 1Gi ❷
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: openshift.com/foo ❸
    fsType: "ext4" ❹
    secretRef: foo-secret ❺
    readOnly: true ❻
    options: ❼
      fooServer: 192.168.0.1:1234
      fooVolumeName: bar

```

- ❶ ボリュームの名前。これは [永続ボリューム要求 \(PVC\)](#) を使用するか、または Pod からボリュームを識別するために使用されます。この名前は、バックエンドストレージのボリューム名とは異なるものにすることができます。
- ❷ このボリュームに割り当てられるストレージの量。
- ❸ ドライバーの名前。このフィールドは必須です。
- ❹ ボリュームに存在するオプションのファイルシステム。このフィールドはオプションです。
- ❺ シークレットへの参照。このシークレットのキーと値は、起動時に FlexVolume ドライバーに渡されます。このフィールドはオプションです。
- ❻ 読み取り専用のフラグ。このフィールドはオプションです。
- ❼ FlexVolume ドライバーの追加オプション。 **options** フィールドでユーザーが指定するフラグに加え、以下のフラグも実行可能ファイルに渡されます。

```

"fsType": "<FS type>",
"readwrite": "<rw>",
"secret/key1": "<secret1>"
...
"secret/keyN": "<secretN>"

```



## 注記

シークレットは、呼び出しのマウント/マウント解除のためにだけ渡されます。

## 27.13. 永続ストレージ用の VMWARE VSPHERE ボリューム

## 27.13.1. 概要

OpenShift Container Platform では、VMWare vSphere の仮想マシンディスク (VMDK: Virtual Machine Disk) ボリュームがサポートされます。VMware vSphere を使用して、OpenShift Container Platform クラスタに [永続ストレージ](#) をプロビジョニングできます。これには、Kubernetes と VMWare vSphere についてのある程度の理解があることが前提となります。

OpenShift Container Platform は vSphere にディスクを作成し、ディスクを適切なインスタンスに割り当てます。

OpenShift Container Platform の [永続ボリューム \(PV\)](#) フレームワークを使用すると、管理者がクラスタに永続ストレージをプロビジョニングできるようになるだけでなく、ユーザーが、基盤となるインフラストラクチャーに精通していなくてもこれらのリソースを要求できるようになります。VMWare vSphere VMDK ボリュームは、[動的にプロビジョニング](#) できます。

永続ボリュームは、単一のプロジェクトまたは namespace にバインドされず、OpenShift Container Platform クラスタ全体で共有できます。ただし、[永続ボリューム要求](#) は、プロジェクトまたは namespace に固有で、ユーザーによる要求が可能です。



### 重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

### 前提条件

vSphere を使用して永続ボリュームを作成する前に、OpenShift Container Platform クラスタが以下の要件を満たしていることを確認してください。

- 最初に OpenShift Container Platform を [vSphere 用に設定](#) する必要があります。
- インフラストラクチャー内の各ノードホストは、vSphere 仮想マシン名に一致する必要があります。
- それぞれのノードホストは、同じリソースグループに属している必要があります。

## 27.13.2. VMWare vSphere ボリュームの動的プロビジョニング

VMware vSphere ボリュームの動的プロビジョニングが推奨されるプロビジョニング方法です。

1. クラスタのプロビジョニング時に **openshift\_cloudprovider\_kind=vsphere** および **openshift\_vsphere\_\*** 変数を指定しない場合、以下の **StorageClass** を手動で作成し、**vsphere-volume** プロビジョナーを使用できるようにする必要があります。

```
$ oc get --export storageclass vsphere-standard -o yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: "vsphere-standard" ①
provisioner: kubernetes.io/vsphere-volume ②
parameters:
  diskformat: thin ③
  datastore: "YourvSphereDatastoreName" ④
reclaimPolicy: Delete
```

- ① StorageClass の名前。

- 2 ストレージプロビジョナーのタイプ。 **vsphere-volume** を指定します。
- 3 ディスクのタイプ。 **zeroedthick** または **thin** のいずれかを指定します。
- 4 ディスクが作成されるソースデータストア。

2. 以前の手順で示した StorageClass を使用して PV を要求した後に、OpenShift Container Platform は vSphere インフラストラクチャーに VMDK ディスクを自動的に作成します。ディスクが作成されていることを確認するには、vSphere でデータストアブラウザーを使用します。



### 注記

vSphere ボリュームディスクは **ReadWriteOnce** アクセスモードで、1つのノードで読み取り/書き込み可能な状態でボリュームをマウントできます。詳細情報は、[アーキテクチャーガイドのアクセスモードのセクション](#) を参照してください。

## 27.13.3. VMware vSphere ボリュームの静的プロビジョニング

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。OpenShift Container Platform が [vSphere 用に設定されている](#) ことを確認した後、OpenShift Container Platform と vSphere に必要になるのは、VM フォルダーパス、ファイルシステムタイプ、および **PersistentVolume** API のみです。

### 27.13.3.1. VMDK の作成



#### 重要

VMDK を使用する前に、以下のいずれかの方法を使用して VMDK を作成します。

- **vmkfstools** を使用して作成する。  
セキュアシェル (SSH) を使用して ESX にアクセスし、以下のコマンドを使用して vmdk ボリュームを作成します。

```
vmkfstools -c 40G /vmfs/volumes/DatastoreName/volumes/myDisk.vmdk
```

- **vmware-vdiskmanager** を使用して作成する。

```
shell vmware-vdiskmanager -c -t 0 -s 40GB -a lsilogic myDisk.vmdk
```

### 27.13.3.2. PersistentVolume の作成

1. **vsphere-pv.yaml** などの PV オブジェクト定義を定義します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
    storage: 2Gi 2
```

```
accessModes:
  - ReadWriteOnce
persistentVolumeReclaimPolicy: Retain
vsphereVolume: ❸
  volumePath: "[datastore1] volumes/myDisk" ❹
  fsType: ext4 ❺
```

- ❶ ボリュームの名前。これを使用して、[PV Claim \(永続ボリューム要求\)](#) で、または Pod から、ボリュームを識別する必要があります。
- ❷ このボリュームに割り当てられるストレージの量。
- ❸ 使用されるボリュームタイプです。この例では、**vsphereVolume** を使用し、ラベルは、vSphere VMDK ボリュームを Pod にマウントするために使用されます。ラベルは vSphere VMDK ボリュームを Pod にマウントするために使用されます。ボリュームの内容はアンマウントされても保持されます。このボリュームタイプは、VMFS データストアと VSAN データストアの両方がサポートされます。
- ❹ 使用する既存の VMDK ボリューム。ボリューム定義で示されるように、データストア名は角かっこ ([ ]) で囲む必要があります。
- ❺ マウントするファイルシステムタイプです。**ext4**、**xf**s、または他のファイルシステムなどが例となります。



### 重要

ボリュームをフォーマットしてプロビジョニングした後に **fsType** パラメーターの値を変更すると、データ損失や Pod にエラーが発生する可能性があります。

2. PV を作成します。

```
$ oc create -f vsphere-pv.yaml
persistentvolume "pv0001" created
```

3. PV が作成されたことを確認します。

```
$ oc get pv
NAME LABELS CAPACITY ACCESSMODES STATUS CLAIM REASON AGE
pv0001 <none> 2Gi RWO Available 2s
```

これで、[PVC \(永続ボリューム要求\)](#) を使用してストレージを要求し、永続ボリュームを活用できるようになります。



### 重要

PV Claim (永続ボリューム要求) は、ユーザーの namespace にのみ存在し、同じ namespace 内の Pod からしか参照できません。別の namespace から永続ボリュームにアクセスしようとすると、Pod にエラーが発生します。

#### 27.13.3.3. VMware vSphere ボリュームのフォーマット

OpenShift Container Platform は、ボリュームをマウントしてコンテナに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。

デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

OpenShift Container Platform は初回の使用前にフォーマットするため、フォーマットされていない vSphere ボリュームを PV として使用できます。

## 27.14. ローカルボリュームを使用した永続ストレージ

### 27.14.1. 概要

OpenShift Container Platform クラスターでは、ローカルボリュームを使用して [永続ストレージ](#) をプロビジョニングできます。ローカル永続ボリュームを使用すると、標準の PVC インターフェイスからディスク、パーティション、ディレクトリーなどのローカルストレージデバイスにアクセスできます。

ローカルボリュームは、Pod をノードに手動でスケジュールせずに使用できます。ただし、ローカルボリュームは、依然として基礎となるノードの可用性に依存しており、すべてのアプリケーションに適している訳ではありません。



#### 注記

ローカルボリュームはアルファ機能であり、OpenShift Container Platform の今後のリリースで変更される場合があります。既知の問題と回避策については、[Feature Status\(Local Volume\)](#) セクションを参照してください。



#### 警告

ローカルボリュームは、静的に作成された永続ボリュームとしてのみ使用できません。

### 27.14.2. プロビジョニング

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。[PersistentVolume](#) API を使用する前に、OpenShift Container Platform が **ローカルボリューム** 用に設定されていることを確認してください。

### 27.14.3. ローカル永続ボリュームの作成

永続ボリュームをオブジェクト定義に定義します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-local-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
```

```

storageClassName: local-storage
local:
  path: /mnt/disks/ssd1
nodeAffinity:
  required:
    nodeSelectorTerms:
      - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
              - my-node

```

#### 27.14.4. ローカルの Persistent Volume Claim (永続ボリューム要求) の作成

Persistent Volume Claim (永続ボリューム要求) をオブジェクト定義に定義します。

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-local-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi 1
  storageClassName: local-storage 2

```

- 1** ストレージボリュームの必要なサイズ。
- 2** ローカル PV で使用されるストレージクラスの名前。

#### 27.14.5. 機能のステータス

機能すること:

- ノードアフィニティがあるディレクトリーを指定して PV を作成する。
- 前述の PV にバインドされている PVC を使用する Pod は常に該当ノードにスケジューリングされる。
- ローカルディレクトリーを検出し、PV を作成、クリーンアップ、および削除する外部の静的プロビジョナーデーモンセット。

機能しないこと:

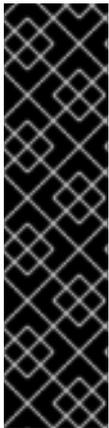
- 単一 Pod に複数のローカル PVC を持たせる。
- PVC バインディングでは Pod のスケジューリング要件を考慮しないため、最適でないか、または適切でない決定がなされる可能性がある。
  - 回避策
    - ローカルボリュームを必要とする Pod を最初に実行します。

- それらの Pod に高い優先順位を設定します。
  - 動作が中断している Pod への PVC のバインドを解除する回避策となるコントローラーを実行します。
- 外部プロビジョナーの起動後にマウントを追加した場合、外部プロビジョナーはマウントの正確な容量を検出できなくなる。
    - 回避策
      - 新規のマウントポイントを追加する前に、デーモンセットを停止し、新規マウントポイントを追加してから daemonset を起動します。
  - 同じ PV を使用する複数の Pod で別々の **fsgroup** を指定すると、**fsgroup** の競合が発生する。

## 27.15. CONTAINER STORAGE INTERFACE (CSI) を使用した永続ストレージ

### 27.15.1. 概要

Container Storage Interface (CSI) により、OpenShift Container Platform は [CSI インターフェイス](#) を [永続ストレージ](#) として実装するストレージバックエンドからストレージを使用できます。



#### 重要

現時点で CLI ボリュームはテクノロジープレビュー機能で、実稼働のワークロードを対象としていません。CSI ボリュームは、今後の OpenShift Container Platform のリリースで変更される可能性があります。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

次のリンクを参照してください。

[https://access.redhat.com/support/offerings/techpreview/\[Red Hat](https://access.redhat.com/support/offerings/techpreview/[Red Hat)



#### 注記

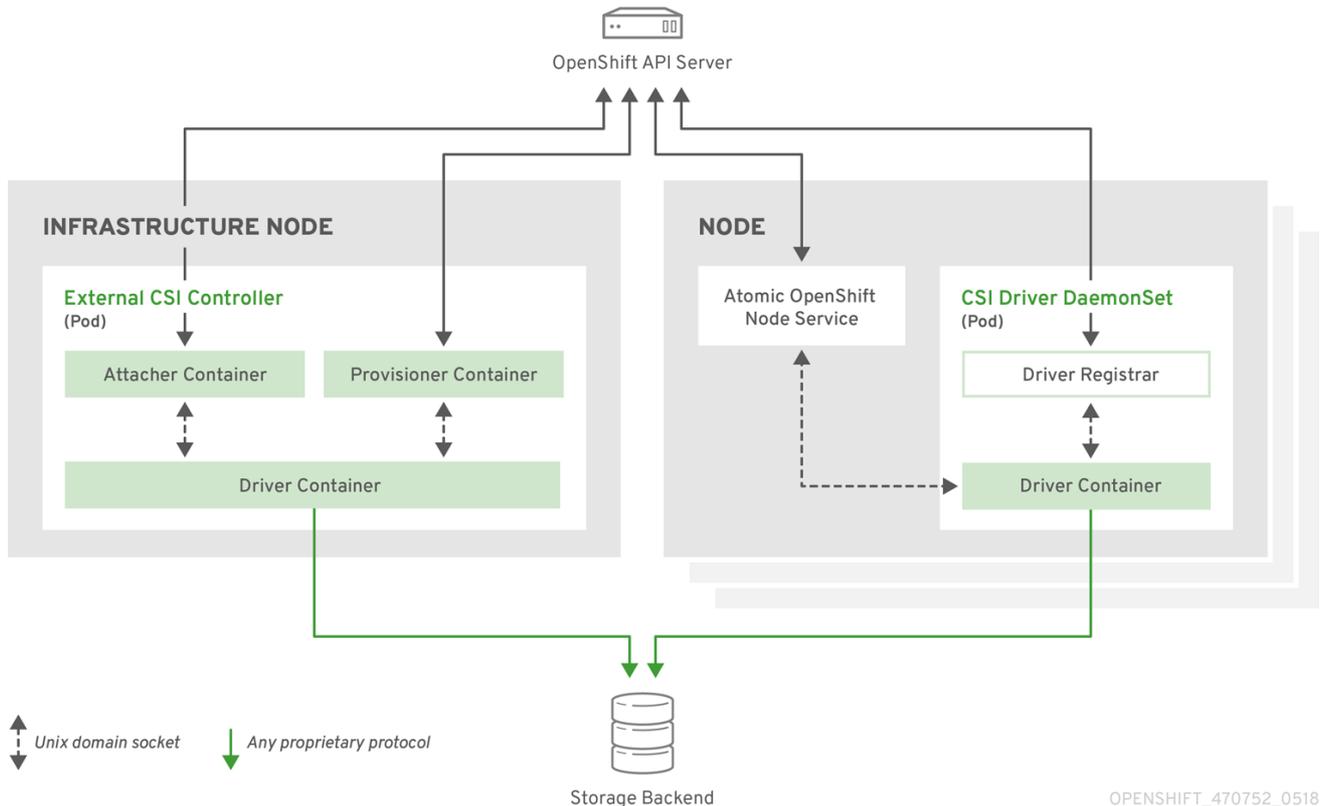
OpenShift Container Platform には CSI ドライバーが含まれていません。[コミュニティ](#) または [ストレージベンダー](#) が提供する CSI ドライバーを使用することが推奨されます。

OpenShift Container Platform 3.11 は、[CSI 仕様](#) のバージョン 0.2.0 をサポートします。

### 27.15.2. アーキテクチャー

CSI ドライバーは通常、コンテナイメージとして提供されます。これらのコンテナは、実行先の OpenShift Container Platform を認識しません。OpenShift Container Platform でサポートされる CSI 互換のストレージバックエンドを使用するには、クラスター管理者は、OpenShift Container Platform とストレージドライバーの橋渡しとして機能するコンポーネントを複数デプロイする必要があります。

以下の図では、OpenShift Container Platform クラスターの Pod で実行されるコンポーネントの俯瞰図を示しています。



OPENSIFT\_470752\_0518

異なるストレージバックエンドに対して複数の CSI ドライバーを実行できます。各ドライバーには、独自の外部コントローラーのデプロイメントおよびドライバーと CSI レジストラを含んだ DaemonSet が必要です。

### 27.15.2.1. 外部の CSI コントローラー

外部の CSI コントローラーは、3つのコンテナを含む1つまたは複数の Pod を配置するデプロイメントです。

- OpenShift Container Platform からの **attach** および **detach** の呼び出しを適切な CSI ドライバーへの **ControllerPublish** および **ControllerUnpublish** 呼び出しに変換する外部の CSI アタッチャーコンテナ。
- OpenShift Container Platform からの **provision** および **delete** 呼び出しを適切な CSI ドライバーへの **CreateVolume** および **DeleteVolume** 呼び出しに変換する外部の CSI プロビジョナーコンテナ。
- CSI ドライバーコンテナ

CSI アタッチャーおよび CSI プロビジョナーコンテナは、Unix Domain Socket を使用して、CSI ドライバーコンテナと対話し、CSI のコミュニケーションが Pod 外に出ないようにします。CSI ドライバーは Pod 外からはアクセスできません。



#### 注記

通常、**attach**、**detach**、**provision** および **delete** 操作では、CSI ドライバーがストレージバックエンドに対する認証情報を使用する必要があります。CSI コントローラー Pod を **インフラストラクチャーノード** で実行し、コンピューターノードで致命的なセキュリティ違反が発生した場合でさえも、認証情報がユーザープロセスに漏洩されないようにします。



## 注記

外部のアタッチャーは、サードパーティーの割り当て/割り当て解除操作をサポートしない CSI ドライバーに対しても実行する必要があります。外部のアタッチャーは、CSI ドライバーに対して **ControllerPublish** または **ControllerUnpublish** 操作を実行しません。ただし、必要な OpenShift Container Platform 割り当て API を実装できるように依然として実行する必要があります。

### 27.15.2.2. CSI ドライバーの DaemonSet

最後に CSI ドライバーの DaemonSet は、OpenShift Container Platform が CSI ドライバーで提供されるストレージをノードにマウントして、永続ボリューム (PV) としてユーザーワークロード (Pod) で使用できるように、全ノードで Pod を実行します。CSI ドライバーがインストールされた Pod には、以下のコンテナが含まれます。

- ノード上で実行中の **openshift-node** サービスに、CSI ドライバーを登録する CSI ドライバーレジストラ。このノードで実行中の **openshift-node** プロセスは、ノードで利用可能な Unix Domain Socket を使用して CSI ドライバーに直接接続します。
- CSI ドライバー

ノードにデプロイされた CSI ドライバーには、ストレージバックエンドへの認証情報をできる限り少なく指定する必要があります。OpenShift Container Platform は、**NodePublish/NodeUnpublish** および **NodeStage/NodeUnstage** (実装されている場合) などの CSI 呼び出しのノードプラグインセットのみを使用します。

### 27.15.3. デプロイメント例

OpenShift Container Platform は CSI ドライバーがインストールされた状態で提供されないので、この例では、OpenShift Container Platform に OpenStack Cinder 向けのコミュニティドライバーをデプロイする方法が示されています。

1. CSI コンポーネントの実行先のプロジェクトと、このコンポーネントを実行するサービスアカウントを新たに作成します。明示的なノードセクターを使用して、マスターノード上でも CSI ドライバーが設定された Daemonset を実行します。

```
# oc adm new-project csi --node-selector=""
Now using project "csi" on server "https://example.com:8443".
```

```
# oc create serviceaccount cinder-csi
serviceaccount "cinder-csi" created
```

```
# oc adm policy add-scc-to-user privileged system:serviceaccount:csi:cinder-csi
scc "privileged" added to: ["system:serviceaccount:csi:cinder-csi"]
```

2. この YAML ファイルを適用して、外部の CSI アタッチャーとプロビジョナーを含むデプロイメントおよび CSI ドライバーを含む DaemonSet を作成します。

```
# This YAML file contains all API objects that are necessary to run Cinder CSI
# driver.
#
# In production, this needs to be in separate files, e.g. service account and
# role and role binding needs to be created once.
#
# It serves as an example of how to use external attacher and external provisioner
```

```
# images that are shipped with OpenShift Container Platform with a community CSI driver.
```

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cinder-csi-role
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["create", "delete", "get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["volumeattachments"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
```

```
---
```

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cinder-csi-role
subjects:
  - kind: ServiceAccount
    name: cinder-csi
    namespace: csi
roleRef:
  kind: ClusterRole
  name: cinder-csi-role
  apiGroup: rbac.authorization.k8s.io
```

```
---
```

```
apiVersion: v1
data:
  cloud.conf:
    W0dsb2JhbF0KYXV0aC11cmwgPSBodHRwczovL2V4YW1wbGUuY29tOjEzMDAwL3YyLjAvC
    nVzZXJuYW1lID0gYWxhZGRpbGpwYXNzd29yZCA9IG9wZW5zZXNhbnVUKdGVuYW50LWki
    D0gZTBmYTg1YjZhMDY0NDM5NTlkMmQzYjQ5NzE3NGJlZDYKcmVnaW9uID0gcmVnaW9u
    T25lCg== 1
kind: Secret
metadata:
  creationTimestamp: null
  name: cloudconfig
```

```
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: cinder-csi-controller
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cinder-csi-controllers
  template:
    metadata:
      labels:
        app: cinder-csi-controllers
    spec:
      serviceAccount: cinder-csi
      containers:
        - name: csi-attacher
          image: registry.redhat.io/openshift3/csi-attacher:v3.11
          args:
            - "--v=5"
            - "--csi-address=$(ADDRESS)"
            - "--leader-election"
            - "--leader-election-namespace=$(MY_NAMESPACE)"
            - "--leader-election-identity=$(MY_NAME)"
          env:
            - name: MY_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: MY_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: ADDRESS
              value: /csi/csi.sock
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: csi-provisioner
          image: registry.redhat.io/openshift3/csi-provisioner:v3.11
          args:
            - "--v=5"
            - "--provisioner=csi-cinderplugin"
            - "--csi-address=$(ADDRESS)"
          env:
            - name: ADDRESS
              value: /csi/csi.sock
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: cinder-driver
          image: quay.io/jsafrane/cinder-csi-plugin
          command: [ "/bin/cinder-csi-plugin" ]
          args:
            - "--nodeid=$(NODEID)"
```

```

- "--endpoint=unix://$(ADDRESS)"
- "--cloud-config=/etc/cloudconfig/cloud.conf"
env:
- name: NODEID
  valueFrom:
    fieldRef:
      fieldPath: spec.nodeName
- name: ADDRESS
  value: /csi/csi.sock
volumeMounts:
- name: socket-dir
  mountPath: /csi
- name: cloudconfig
  mountPath: /etc/cloudconfig
volumes:
- name: socket-dir
  emptyDir:
- name: cloudconfig
  secret:
    secretName: cloudconfig

---

kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: cinder-csi-ds
spec:
  selector:
    matchLabels:
      app: cinder-csi-driver
  template:
    metadata:
      labels:
        app: cinder-csi-driver
    spec:
      2
      serviceAccount: cinder-csi
      containers:
        - name: csi-driver-registrar
          image: registry.redhat.io/openshift3/csi-driver-registrar:v3.11
          securityContext:
            privileged: true
          args:
            - "--v=5"
            - "--csi-address=$(ADDRESS)"
          env:
            - name: ADDRESS
              value: /csi/csi.sock
            - name: KUBE_NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
          volumeMounts:
            - name: socket-dir
              mountPath: /csi

```

```

- name: cinder-driver
  securityContext:
    privileged: true
    capabilities:
      add: ["SYS_ADMIN"]
    allowPrivilegeEscalation: true
  image: quay.io/jsafrane/cinder-csi-plugin
  command: [ "/bin/cinder-csi-plugin" ]
  args:
    - "--nodeid=$(NODEID)"
    - "--endpoint=unix://$(ADDRESS)"
    - "--cloud-config=/etc/cloudconfig/cloud.conf"
  env:
    - name: NODEID
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    - name: ADDRESS
      value: /csi/csi.sock
  volumeMounts:
    - name: socket-dir
      mountPath: /csi
    - name: cloudconfig
      mountPath: /etc/cloudconfig
    - name: mountpoint-dir
      mountPath: /var/lib/origin/openshift.local.volumes/pods/
      mountPropagation: "Bidirectional"
    - name: cloud-metadata
      mountPath: /var/lib/cloud/data/
    - name: dev
      mountPath: /dev
  volumes:
    - name: cloud-metadata
      hostPath:
        path: /var/lib/cloud/data/
    - name: socket-dir
      hostPath:
        path: /var/lib/kubelet/plugins/csi-cinderplugin
        type: DirectoryOrCreate
    - name: mountpoint-dir
      hostPath:
        path: /var/lib/origin/openshift.local.volumes/pods/
        type: Directory
    - name: cloudconfig
      secret:
        secretName: cloudconfig
    - name: dev
      hostPath:
        path: /dev

```

1 **OpenStack 設定** に記載されているように、OpenStack デプロイメントの `cloud.conf` に置き換えます。たとえば、シークレットは、`oc create secret generic cloudconfig --from-file cloud.conf --dry-run -o yaml` を使用して生成できます。

2

オプションで、**nodeSelector** を CSI ドライバー Pod テンプレートに追加し、CSI ドライバーが起動するノードを設定します。セクターに一致するノードのみが CSI ドライバー

#### 27.15.4. 動的プロビジョニング

永続ストレージの動的プロビジョニングは、CSI ドライバーおよび基礎となるストレージバックエンドの機能により異なります。CSI ドライバーのプロバイダーは、OpenShift Container Platform での StorageClass の作成方法および設定に利用できるパラメーターについての文書を作成する必要があります。

OpenStack Cinder の例に示されるように、動的プロビジョニングを有効にするためにこの StorageClass をデプロイできます。以下の例では、新規のデフォルトストレージクラスを作成して、特別なストレージクラスを必要としない PVC がすべて、インストールした CSI ドライバーによりプロビジョニングされるようにします。

```
# oc create -f - << EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: cinder
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi-cinderplugin
parameters:
EOF
```

#### 27.15.5. 使用方法

CSI ドライバーがデプロイされ、動的プロビジョニング用に StorageClass が作成されたら、OpenShift Container Platform は CSI を使用する準備が整います。以下の例では、変更を加えずに、デフォルトの MySQL テンプレートをインストールします。

```
# oc new-app mysql-persistent
--> Deploying template "openshift/mysql-persistent" to project default
...

# oc get pvc
NAME          STATUS  VOLUME                                     CAPACITY  ACCESS MODES  STORAGECLASS  AGE
mysql         Bound   kubernetes-dynamic-pv-3271ffcb4e1811e8  1Gi       RWO           cinder        3s
```

### 27.16. OPENSTACK MANILA を使用した永続ストレージ

#### 27.16.1. 概要



## 重要

OpenStack Manila を使用した永続ボリューム (PV) のプロビジョニングはテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、<https://access.redhat.com/support/offerings/techpreview/> を参照してください。

OpenShift Container Platform は、[OpenStack Manila](#) の共有ファイルシステムサービスを使用して PV をプロビジョニングできます。

OpenStack Manila サービスが正しく設定され、OpenShift Container Platform クラスターからアクセスできることが前提です。プロビジョニングが可能なのは、NFS 共有タイプのみです。

[PV、永続ボリューム要求 \(PVC\)、動的プロビジョニング](#) および [RBAC 認証](#) について理解しておくことを推奨します。

## 27.16.2. インストールと設定

この機能は外部プロビジョナーによって提供されます。OpenShift Container Platform クラスターにインストールし、設定する必要があります。

### 27.16.2.1. 外部プロビジョナーの起動

外部プロビジョナーサービスは、コンテナイメージとして配布され、OpenShift Container Platform クラスターで通常通り実行できます。

API オブジェクトを管理しているコンテナを許可するには、以下のようにして、必要なロールベースアクセス制御 (RBAC) ルールを管理者が設定する必要があります。

1. **ServiceAccount** を作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: manila-provisioner-runner
```

2. **ClusterRole** を作成します。

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: manila-provisioner-role
rules:
- apiGroups: [""]
  resources: ["persistentvolumes"]
  verbs: ["get", "list", "watch", "create", "delete"]
- apiGroups: [""]
  resources: ["persistentvolumeclaims"]
  verbs: ["get", "list", "watch", "update"]
- apiGroups: ["storage.k8s.io"]
```

```
resources: ["storageclasses"]
verbs: ["get", "list", "watch"]
- apiGroups: [""]
resources: ["events"]
verbs: ["list", "watch", "create", "update", "patch"]
```

3. **ClusterRoleBinding** を使用して、以下のようにルールをバインドします。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: manila-provisioner
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: manila-provisioner-role
subjects:
- kind: ServiceAccount
  name: manila-provisioner-runner
  namespace: default
```

4. 新しい **StorageClass** を作成します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: "manila-share"
provisioner: "externalstorage.k8s.io/manila"
parameters:
  type: "default" ①
  zones: "nova" ②
```

- ① プロビジョナーがボリュームのように作成する [Manila 共有タイプ](#)
- ② ボリュームが作成される可能性のある Manila アベイラビリティゾーンセット

環境変数を使用してプロビジョナーによる Manila サービスへの接続、認証、承認ができるように設定します。以下のリストから、お使いのインストールに合った適切な環境変数の組み合わせを選択します。

```
OS_USERNAME
OS_PASSWORD
OS_AUTH_URL
OS_DOMAIN_NAME
OS_TENANT_NAME
```

```
OS_USERID
OS_PASSWORD
OS_AUTH_URL
OS_TENANT_ID
```

```
OS_USERNAME
OS_PASSWORD
```

```
OS_AUTH_URL
OS_DOMAIN_ID
OS_TENANT_NAME
```

```
OS_USERNAME
OS_PASSWORD
OS_AUTH_URL
OS_DOMAIN_ID
OS_TENANT_ID
```

プロビジョナーに変数を渡すには、**Secret** を使用します。以下の例は、1つ目の変数の組み合わせ向けに設定した **Secret** です。

```
apiVersion: v1
kind: Secret
metadata:
  name: manila-provisioner-env
type: Opaque
data:
  os_username: <base64 encoded Manila username>
  os_password: <base64 encoded password>
  os_auth_url: <base64 encoded OpenStack Keystone URL>
  os_domain_name: <base64 encoded Manila service Domain>
  os_tenant_name: <base64 encoded Manila service Tenant/Project name>
```



### 注記

より新しいバージョンの OpenStack では、テナントではなくプロジェクトを使用します。ただし、プロビジョナーが使用する環境変数は、名前に **TENANT** を使用する必要があります。

最後の手順では、デプロイメントなどを使用して、プロビジョナー自体を起動します。

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: manila-provisioner
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: manila-provisioner
    spec:
      serviceAccountName: manila-provisioner-runner
      containers:
        - image: "registry.redhat.io/openshift3/manila-provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          name: manila-provisioner
          env:
            - name: "OS_USERNAME"
              valueFrom:
```

```

secretKeyRef:
  name: manila-provisioner-env
  key: os_username
- name: "OS_PASSWORD"
valueFrom:
  secretKeyRef:
    name: manila-provisioner-env
    key: os_password
- name: "OS_AUTH_URL"
valueFrom:
  secretKeyRef:
    name: manila-provisioner-env
    key: os_auth_url
- name: "OS_DOMAIN_NAME"
valueFrom:
  secretKeyRef:
    name: manila-provisioner-env
    key: os_domain_name
- name: "OS_TENANT_NAME"
valueFrom:
  secretKeyRef:
    name: manila-provisioner-env
    key: os_tenant_name

```

### 27.16.3. 使用方法

プロビジョナーの実行後に、PVC と適切な StorageClass を使用して PV をプロビジョニングすることができます。

```

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: manila-nfs-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2G
  storageClassName: manila-share

```

次に、**PersistentVolumeClaim** は、新たにプロビジョニングされた Manila 共有でサポートされる **PersistentVolume** にバインドされます。**PersistentVolumeClaim** の後に **PersistentVolume** を削除したら、プロビジョナーは Manila 共有を削除して、エクスポートを中止します。

## 27.17. 動的プロビジョニングとストレージクラスの作成

### 27.17.1. 概要

StorageClass リソースオブジェクトは、要求可能なストレージを記述し、分類するほか、**動的にプロビジョニングされるストレージ**のパラメーターを要求に応じて渡すための手段を提供します。StorageClass オブジェクトは、さまざまなレベルのストレージとストレージへのアクセスを制御

するための管理メカニズムとしても機能します。クラスター管理者 (**cluster-admin**) またはストレージ管理者 (**storage-admin**) は、ユーザーが基礎となるストレージボリュームソースに関する詳しい知識がなくても要求できる **StorageClass** オブジェクトを定義し、作成します。

OpenShift Container Platform の **永続ボリューム** フレームワークはこの機能を有効にし、管理者がクラスターに永続ストレージをプロビジョニングできるようにします。フレームワークにより、ユーザーは基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようになります。

OpenShift Container Platform では、数多くのストレージタイプを永続ボリュームとして使用することができます。これらはすべて管理者によって静的にプロビジョニングされますが、一部のストレージタイプは組み込みプロバイダーとプラグイン API を使用して動的に作成できます。



**注記**

動的プロビジョニングを有効にするには、**openshift\_master\_dynamic\_provisioning\_enabled** 変数を Ansible インベントリーファイルの **[OSEv3:vars]** セクションに追加し、その値を **True** に設定します。

[OSEv3:vars]

openshift\_master\_dynamic\_provisioning\_enabled=True

**27.17.2. 動的にプロビジョニングされる使用可能なプラグイン**

OpenShift Container Platform は、以下の**プロビジョナープラグイン**を提供します。これらには、クラスターの設定済みプロバイダーの API を使用して新規ストレージリソースを作成する動的プロビジョニング用の一般的な実装が含まれます。

Storage Type	プロビジョナープラグインの名前	必要な設定	備考
OpenStack Cinder	<b>kubernetes.io/cinder</b>	<a href="#">OpenStack の設定</a>	
AWS Elastic Block Store (EBS)	<b>kubernetes.io/aws-ebs</b>	<a href="#">AWS の設定</a>	複数クラスターを複数の異なるゾーンで使用する際の動的プロビジョニングの場合、各ノードに <b>Key=kubernetes.io/cluster/xxxx,Value=clusterid</b> のタグを付けます。ここで、 <b>xxxx</b> と <b>clusterid</b> はクラスターごとに固有の値になります。
GCE Persistent Disk (gcePD)	<b>kubernetes.io/gce-pd</b>	<a href="#">GCE の設定</a>	マルチゾーン設定では、PV が現行クラスターのノードが存在しないゾーンで作成されないようにするため、Openshift クラスターを GCE プロジェクトごとに実行することが推奨されます。

Storage Type	プロビジョナープラグインの名前	必要な設定	備考
GlusterFS	<a href="https://kubernetes.io/glusterfs">kubernetes.io/glusterfs</a>	<a href="#">GlusterFS の設定</a>	
Ceph RBD	<a href="https://kubernetes.io/rbd">kubernetes.io/rbd</a>	<a href="#">Ceph RBD の設定</a>	
NetApp の Trident	<a href="https://netapp.io/trident">netapp.io/trident</a>	<a href="#">Trident の設定</a>	NetApp ONTAP、SolidFire、および E-Series ストレージ向けのストレージオーケストレーター。
VMWare vSphere	<a href="https://kubernetes.io/vsphere-re-volume">kubernetes.io/vsphere-re-volume</a>	<a href="#">vSphere と Kubernetes の使用開始</a>	
Azure Disk	<a href="https://kubernetes.io/azure-disk">kubernetes.io/azure-disk</a>	<a href="#">Azure の設定</a>	



### 重要

選択したプロビジョナープラグインでは、関連するクラウド、ホスト、またはサードパーティープロバイダーを、関連するドキュメントに従って設定する必要があります。

## 27.17.3. StorageClass の定義

現時点で、StorageClass オブジェクトはグローバルスコープオブジェクトであり、**cluster-admin** または **storage-admin** ユーザーによって作成される必要があります。



### 注記

GCE と AWS の場合、デフォルトの StorageClass が OpenShift Container Platform のインストール時に作成されます。[デフォルトの StorageClass を変更](#) または削除することができます。

現時点で6つのプラグインがサポートされています。以下のセクションでは、StorageClass の基本オブジェクトの定義とサポートされている各プラグインタイプの具体的な例について説明します。

### 27.17.3.1. 基本 StorageClass オブジェクト定義

#### StorageClass 基本オブジェクトの定義

```
kind: StorageClass 1
apiVersion: storage.k8s.io/v1 2
metadata:
  name: foo 3
  annotations: 4
  ...
provisioner: kubernetes.io/plug-in-type 5
```

```
parameters: ⑥
  param1: value
  ...
  paramN: value
```

- ① (必須) API オブジェクトタイプ。
- ② (必須) 現在の apiVersion。
- ③ (必須) StorageClass の名前。
- ④ (オプション) StorageClass のアノテーション
- ⑤ (必須) このストレージクラスに関連付けられているプロビジョナーのタイプ。
- ⑥ (オプション) 特定のプロビジョナーに必要なパラメーター。これはプラグインによって異なります。

### 27.17.3.2. StorageClass のアノテーション

StorageClass をクラスター全体のデフォルトとして設定するには、以下を実行します。

```
storageclass.kubernetes.io/is-default-class: "true"
```

これにより、特定のボリュームを指定しない Persistent Volume Claim (永続ボリューム要求、PVC) がデフォルト StorageClass によって自動的にプロビジョニングされるようになります。



#### 注記

ベータアノテーションの **storageclass.beta.kubernetes.io/is-default-class** は機能しています。ただし、今後のリリースで削除される予定です。

StorageClass の記述を設定するには、以下のように指定します。

```
kubernetes.io/description: My StorageClass Description
```

### 27.17.3.3. OpenStack Cinder オブジェクトの定義

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gold
provisioner: kubernetes.io/cinder
parameters:
  type: fast ①
  availability: nova ②
  fsType: ext4 ③
```

- ① Cinder で作成されるボリュームタイプ。デフォルトは空です。

- 2 アベイラビリティゾーン。指定しない場合、ボリュームは通常 OpenShift Container Platform クラスターのノードがあるすべてのアクティブゾーンでラウンドロビンされます。
- 3 動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの **fsType** フィールドにコピーされ、ボリュームの初回マウント時にファイルシステムが作成されます。デフォルト値は **ext4** です。

#### 27.17.3.4. AWS ElasticBlockStore (EBS) オブジェクトの定義

##### aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 1
  zone: us-east-1d 2
  iopsPerGB: "10" 3
  encrypted: "true" 4
  kmsKeyId: keyvalue 5
  fsType: ext4 6
```

- 1 **io1**、**gp2**、**sc1**、**st1** から選択します。デフォルトは **gp2** です。有効な Amazon Resource Name (ARN) 値については、[AWS のドキュメント](#) を参照してください。
- 2 AWS ゾーン。ゾーンを指定しない場合、ボリュームは通常、OpenShift Container Platform クラスターのノードがあるすべてのアクティブゾーンでラウンドロビンされます。zone パラメーターと zones パラメーターを同時に使用することはできません。
- 3 **io1** ボリュームのみ。1 GiB あたり 1 秒あたりの I/O 処理数。AWS ボリュームプラグインは、この値と要求されたボリュームのサイズを乗算してボリュームの IOPS を算出します。値の上限は、AWS でサポートされる最大値である 20,000 IOPS です。詳細については、[AWS のドキュメント](#) を参照してください。
- 4 EBS ボリュームを暗号化するかどうかを示します。有効な値は **true** または **false** です。
- 5 オプション。ボリュームを暗号化する際に使用するキーの完全な ARN。値を指定しない場合でも **encrypted** が **true** に設定されている場合は、AWS によってキーが生成されます。有効な ARN 値については、[AWS のドキュメント](#) を参照してください。
- 6 動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの **fsType** フィールドにコピーされ、ボリュームの初回マウント時にファイルシステムが作成されます。デフォルト値は **ext4** です。

#### 27.17.3.5. GCE PersistentDisk (gcePD) オブジェクトの定義

##### gce-pd-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
```

```

metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard ①
  zone: us-central1-a ②
  zones: us-central1-a, us-central1-b, us-east1-b ③
  fsType: ext4 ④

```

- ① **pd-standard** または **pd-ssd** のいずれかを選択します。デフォルトは **pd-ssd** です。
- ② GCE ゾーン。ゾーンを指定しない場合、ボリュームは通常、OpenShift Container Platform クラスターのノードがあるすべてのアクティブゾーンでラウンドロビンされます。zone パラメーターと zones パラメーターを同時に使用することはできません。
- ③ GCE ゾーンのコマ区切りの一覧。ゾーンを指定しない場合、ボリュームは通常、OpenShift Container Platform クラスターのノードがあるすべてのアクティブゾーンでラウンドロビンされます。zone パラメーターと zones パラメーターを同時に使用することはできません。
- ④ 動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの **fsType** フィールドにコピーされ、ボリュームの初回マウント時にファイルシステムが作成されます。デフォルト値は **ext4** です。

### 27.17.3.6. GlusterFS オブジェクトの定義

#### glusterfs-storageclass.yaml

```

kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/glusterfs
parameters: ①
  resturl: http://127.0.0.1:8081 ②
  restuser: admin ③
  secretName: heketi-secret ④
  secretNamespace: default ⑤
  gidMin: "40000" ⑥
  gidMax: "50000" ⑦
  volumeoptions: group metadata-cache, nl-cache on ⑧
  volumetype: replicate:3 ⑨
  volumenameprefix: custom ⑩

```

- ① 一覧表示されているのは、必須パラメーターおよびいくつかのオプションのパラメーターです。追加のパラメーターについては、[Registering a Storage Class](#) を参照してください。
- ② GlusterFS ボリュームをオンデマンドでプロビジョニングする **heketi** (Gluster 用のボリューム管理 REST サービス) URL。一般的な形式は **{http/https}://{IPaddress}:{Port}** です。GlusterFS 動的プロビジョナーの場合、これは必須パラメーターです。heketi サービスが OpenShift Container Platform でルーティング可能なサービスとして公開されている場合には、解決可能な完全修飾ドメイン名 (FQDN) と heketi サービス URL が割り当てられます。

- 3 ボリュームを作成するためのアクセスを持つ heketi ユーザー。通常は admin です。
- 4 heketi との通信に使用するユーザーパスワードを含むシークレットの ID。オプション。 **secretNamespace** と **secretName** の両方を省略した場合、空のパスワードが使用されます。指定するシークレットは **"kubernetes.io/glusterfs"** タイプである必要があります。
- 5 前述の **secretName** の namespace。オプション。 **secretNamespace** と **secretName** の両方を省略した場合、空のパスワードが使用されます。指定するシークレットは **"kubernetes.io/glusterfs"** タイプである必要があります。
- 6 オプション。この StorageClass のボリュームの GID 範囲の最小値です。
- 7 オプション。この StorageClass のボリュームの GID 範囲の最大値です。
- 8 オプション。新規に作成されたボリュームのオプションです。これにより、パフォーマンスチューニングが可能になります。GlusterFS ボリュームの他のオプションについては、[Tuning Volume Options](#) を参照してください。
- 9 オプション。使用する [ボリュームのタイプ](#) です。
- 10 オプション。 `<volumenameprefix>_<namespace>_<claimname>_UUID` の形式を使用してカスタムボリューム名のサポートを有効にします。この storageClass を使用してプロジェクト **project1** に **myclaim** という新規の PVC を作成する場合、ボリューム名は **custom-project1-myclaim-UUID** になります。



### 注記

**gidMin** 値と **gidMax** 値を指定しない場合、デフォルトはそれぞれ 2000 と 2147483647 になります。動的にプロビジョニングされる各ボリュームには、この範囲 (**gidMin-gidMax**) の GID が割り当てられます。GID は、対応するボリュームが削除されるとプールから解放されます。GID プールは StorageClass ごとに設定されます。複数のストレージクラス間で GID 範囲が重複している場合、プロビジョナーによって、重複する GID が割り当てられる可能性があります。

heketi 認証を使用する場合は、管理キーを含むシークレットも存在している必要があります。

### heketi-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: heketi-secret
  namespace: default
data:
  key: bXlwYXNzd29yZA== 1
type: kubernetes.io/glusterfs
```

- 1 base64 でエンコードされたパスワード。例: `echo -n "mypassword" | base64`



## 注記

PV が動的にプロビジョニングされると、GlusterFS プラグインによってエンドポイントと **gluster-dynamic-`<claimname>`** という名前のヘッドレスサービスが自動的に作成されます。PVC が削除されると、これらの動的リソースは自動的に削除されます。

### 27.17.3.7. Ceph RBD オブジェクトの定義

#### ceph-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/rbd
parameters:
  monitors: 10.16.153.105:6789 ①
  adminId: admin ②
  adminSecretName: ceph-secret ③
  adminSecretNamespace: kube-system ④
  pool: kube ⑤
  userId: kube ⑥
  userSecretName: ceph-secret-user ⑦
  fsType: ext4 ⑧
```

- ① Ceph モニター (コンマ区切り)。必須です。
- ② Ceph クライアント ID。これにより、プール内にイメージを作成できます。デフォルトは `admin` です。
- ③ `adminId` のシークレット名。必須です。設定するシークレットのタイプは `kubernetes.io/rbd` である必要があります。
- ④ `adminSecret` の namespace。デフォルトは `default` です。
- ⑤ Ceph RBD プール。デフォルトは `rbd` です。
- ⑥ Ceph RBD イメージのマッピングに使用される Ceph クライアント ID。デフォルトは `adminId` と同じです。
- ⑦ Ceph RBD イメージをマッピングするために使用する `userId` 用の Ceph シークレットの名前。PVC と同じ namespace に存在する必要があります。必須です。
- ⑧ 動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの `fsType` フィールドにコピーされ、ボリュームの初回マウント時にファイルシステムが作成されます。デフォルト値は `ext4` です。

### 27.17.3.8. Trident オブジェクトの定義

#### trident.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
```

```

metadata:
  name: gold
provisioner: netapp.io/trident ❶
parameters: ❷
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"

```

Trident は、これらのパラメーターを Trident に登録されているさまざまなストレージプールの選択条件として使用します。Trident 自体は個別に設定されます。

- ❶ Trident を OpenShift Container Platform にインストールする方法の詳細については、[Trident のドキュメント](#) を参照してください。
- ❷ サポートされているパラメーターの詳細については、Trident のドキュメントの [ストレージ属性](#) に関するセクションを参照してください。

### 27.17.3.9. VMWare vSphere オブジェクトの定義

#### vsphere-storageclass.yaml

```

kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: slow
provisioner: kubernetes.io/vsphere-volume ❶
parameters:
  diskformat: thin ❷

```

- ❶ OpenShift Container Platform で VMware vSphere を使用方法の詳細については、[VMware vSphere のドキュメント](#) を参照してください。
- ❷ **diskformat: thin**、**zeroedthick**、および **eagerzeroedthick**。詳細については、vSphere のドキュメントを参照してください。デフォルト: **thin**

### 27.17.3.10. Azure File のオブジェクト定義

Azure file の動的プロビジョニングを設定するには、以下を実行します。

1. ユーザーのプロジェクトにロールを作成します。

```

$ cat azf-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: system:controller:persistent-volume-binder
  namespace: <user's project name>
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create", "get", "delete"]

```

2. **persistent-volume-binder** サービスにバインドされるロールを **kube-system** プロジェクトに作成します。

```
$ cat azf-rolebind.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: system:controller:persistent-volume-binder
  namespace: <user's project>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: system:controller:persistent-volume-binder
subjects:
- kind: ServiceAccount
  name: persistent-volume-binder
  namespace: kube-system
```

3. **admin** として、ユーザーのプロジェクトにサービスアカウントを追加します。

```
$ oc policy add-role-to-user admin system:serviceaccount:kube-system:persistent-volume-binder -n <user's project>
```

4. Azure File 向けにストレージクラスを作成します。

```
$ cat azfsc.yaml | oc create -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azfsc
provisioner: kubernetes.io/azure-file
mountOptions:
- dir_mode=0777
- file_mode=0777
```

ユーザーは、このストレージクラスを使用する PVC を作成できるようになりました。

### 27.17.3.11. Azure Disk オブジェクト定義

#### azure-advanced-disk-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/azure-disk
parameters:
  storageAccount: azure_storage_account_name ①
  storageaccounttype: Standard_LRS ②
kind: Dedicated ③
```

- 1 Azure ストレージアカウントの名前。これはクラスターと同じリソースグループに存在している必要があります。ストレージアカウントを指定した場合、**location** は無視されます。ストレージ
- 2 Azure ストレージアカウントの SKU の層。デフォルトは空です。注: プレミアム VM は **Standard\_LRS** ディスクと **Premium\_LRS** ディスクの両方を割り当て、標準 VM は **Standard\_LRS** ディスクのみを、マネージド VM はマネージドディスクのみを、アンマネージド VM はアンマネージドディスクのみを割り当てることができます。
- 3 許容値は、**Shared** (デフォルト)、**Dedicated** および **Managed** です。
  - a. **kind** が **Shared** に設定されている場合は、Azure は、クラスターと同じリソースグループにあるいくつかの共有ストレージアカウントで、アンマネージドディスクをすべて作成します。
  - b. **kind** が **Managed** に設定されている場合は、Azure は新しいマネージドディスクを作成します。
  - c. **kind** が **Dedicated** に設定されており、**storageAccount** が指定されている場合には、Azure は、クラスターと同じリソースグループ内にある新規のアンマネージドディスク用に、指定のストレージアカウントを使用します。これを機能させるには、以下が前提となります。
    - 指定のストレージアカウントが、同じリージョン内にあること。
    - Azure Cloud Provider にストレージアカウントへの書き込み権限があること。
  - d. **kind** が **Dedicated** に設定されており、**storageAccount** が指定されていない場合には、Azure はクラスターと同じリソースグループ内の新規のアンマネージドディスク用に、新しい専用のストレージアカウントを作成します。

### 重要

Azure StorageClass は OpenShift Container Platform バージョン 3.7 で改訂されています。以前のバージョンからアップグレードした場合には、以下のいずれかを実行します。

- **kind: dedicated** のプロパティを指定して、アップグレード前に作成した Azure StorageClass を使用し続ける。または
- **azure.conf** ファイルにロケーションのパラメーター (例: "**location**": "**southcentralus**") を追加して、デフォルトのプロパティ **kind: shared** を使用します。こうすることで、新しいストレージアカウントを作成し、今後使用できるようになります。

#### 27.17.4. デフォルト StorageClass の変更

GCE と AWS を使用している場合にデフォルトの StorageClass を変更するには、以下のプロセスを使用します。

1. StorageClass の一覧を表示します。

```
$ oc get storageclass
```

```
NAME          TYPE
```

gp2 (default)	kubernetes.io/aws-ebs <b>1</b>
standard	kubernetes.io/gce-pd

**1** (default) はデフォルトの StorageClass を示します。

2. デフォルトの StorageClass のアノテーション **storageclass.kubernetes.io/is-default-class** の値を **false** に変更します。

```
$ oc patch storageclass gp2 -p '{"metadata": {"annotations": \
{"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. アノテーション **storageclass.kubernetes.io/is-default-class=true** を追加するか、このアノテーションを変更して別の StorageClass をデフォルトにします。

```
$ oc patch storageclass standard -p '{"metadata": {"annotations": \
{"storageclass.kubernetes.io/is-default-class": "true"}}}'
```



### 注記

複数の StorageClass がデフォルトとしてマークされている場合、PVC は **storageClassName** が明示的に指定されている場合にのみ作成できます。そのため、1 つの StorageClass のみをデフォルトとして設定する必要があります。

1. 変更内容を確認します。

```
$ oc get storageclass
```

NAME	TYPE
gp2	kubernetes.io/aws-ebs
standard (default)	kubernetes.io/gce-pd

## 27.17.5. 追加情報とサンプル

- [動的プロビジョニング用の StorageClass の例と使用法](#)
- [動的プロビジョニングのない StorageClass の例と使用法](#)

## 27.18. ボリュームのセキュリティー

### 27.18.1. 概要

このトピックでは、ボリュームのセキュリティーに関連する Pod のセキュリティーについての一般的なガイドを提供します。Pod レベルのセキュリティーに関する全般的な情報については、[SCC \(Security Context Constraints\) の管理](#) と [SCC \(Security Context Constraints\) の概念に関するトピック](#) を参照してください。OpenShift Container Platform の永続ボリューム (PV) フレームワークに関する全般的な情報については、[永続ストレージ](#) の概念に関するトピックを参照してください。

永続ストレージにアクセスするには、クラスター管理者/ストレージ管理者とエンド開発者間の調整が必要です。クラスター管理者は、基礎となる物理ストレージを抽象化する PV を作成します。開発者は、容量などの一致条件に基づいて Pod と (必要に応じて) PV にバインドされる PVC を作成します。

同じプロジェクト内の複数の Persistent Volume Claim (永続ボリューム要求、PVC) を同じ PV にバイ

ンドできます。ただし、PVC を PV にバインドすると、最初の要求のプロジェクトの外部にある要求をその PV にバインドできなくなります。基礎となるストレージに複数のプロジェクトからアクセスする必要がある場合は、プロジェクトごとに独自の PV が必要です。これらの PV は同じ物理ストレージを参照できます。PV および PVC の詳細例は、[Deploying WordPress and MySQL with Persistent Volumes](#) の例を参照してください。

クラスター管理者が PV への Pod アクセスを許可するには、以下を行う必要があります。

- 実際のストレージに割り当てられるグループ ID またはユーザー ID を把握しておく
- SELinux に関する考慮事項を理解しておく
- これらの ID が Pod の要件に一致するプロジェクトまたは SCC に対して定義される正式な ID の範囲で許可されることを確認する

グループ ID、ユーザー ID および SELinux の値は、Pod 定義の **SecurityContext** セクションで定義します。グループ ID は、Pod に対してグローバルであり、Pod で定義されるすべてのコンテナに適用されます。ユーザー ID は同様にグローバルにすることも、コンテナごとに固有にすることもできます。ボリュームへのアクセスは以下の 4 つのセクションで制御します。

- [supplementalGroups](#)
- [fsGroup](#)
- [runAsUser](#)
- [seLinuxOptions](#)

### 27.18.2. SCC、デフォルト値および許可される範囲

SCC は、デフォルトのユーザー ID、**fsGroup** ID、補助グループ ID、および SELinux ラベルが Pod に割り当てられるかどうかに影響します。また、Pod 定義 (またはイメージ) で指定される ID が許容可能な ID の範囲に対して検証されるかどうかにも影響します。検証が必要な場合で検証が失敗すると、Pod も失敗します。

SCC は、**runAsUser**、**supplementalGroups**、**fsGroup** などのストラテジーを定義します。これらのストラテジーは Pod が承認されるかどうかを判断するのに役立ちます。**RunAsAny** に設定されるストラテジーの値は、Pod がそのストラテジーに関して必要なことを何でも実行できるということを実質的に宣言するものです。そのストラテジーに対する承認は省略され、そのストラテジーに基づいて生成される OpenShift Container Platform のデフォルト値はありません。したがって、生成されるコンテナの ID と SELinux ラベルは、OpenShift Container Platform ポリシーではなく、コンテナのデフォルトに基づいて割り当てられます。

以下に **RunAsAny** の簡単な概要を示します。

- Pod 定義 (またはイメージ) に定義されるすべての ID が許可されます。
- Pod 定義 (およびイメージ) に ID がない場合は、コンテナによって ID が割り当てられます。Docker の場合、この ID は **root** (0) です。
- SELinux ラベルは定義されないため、Docker によって一意のラベルが割り当てられます。

このような理由により、ID 関連のストラテジーについて **RunAsAny** が設定された SCC は、通常の開発者がアクセスできないように保護する必要があります。一方、**MustRunAs** または **MustRunAsRange** に設定された SCC ストラテジーは、(ID 関連のストラテジーについての) ID 検証をトリガーします。その結果、Pod 定義またはイメージに値が直接指定されていない場合は、OpenShift Container Platform によってデフォルト値がコンテナに割り当てられます。

## 注意

**RunAsAny FSGroup** ストラテジーが設定された SCC へのアクセスを許可すると、ユーザーがブロックデバイスにアクセスするのを防止することもできます。Pod では、ユーザーのブロックデバイスを引き継ぐために **fsGroup** を指定する必要があります。通常、これを行うには、SCC **FSGroup** ストラテジーを **MustRunAs** に設定します。ユーザーの Pod に **RunAsAny FSGroup** ストラテジーが設定された SCC が割り当てられている場合、ユーザーが **fsGroup** ストラテジーを各自で指定する必要があることに気づくまで、**permission denied** エラーが出される可能性があります。

SCC では、許可される ID (ユーザーまたはグループ) の範囲を定義できます。範囲チェックが必要な場合 (**MustRunAs** を使用する場合など) で、許容可能な範囲が SCC で定義されていない場合は、プロジェクトによって ID 範囲が決定されます。したがって、プロジェクトでは、許容可能な ID の範囲がサポートされます。ただし、SCC とは異なり、プロジェクトは **runAsUser** などのストラテジーを定義しません。

許容可能な範囲を設定すると、コンテナ ID の境界を定義するだけでなく、範囲の最小値を対象の ID のデフォルト値にできるので役に立ちます。たとえば、SCC ID ストラテジーの値が **MustRunAs** で、ID 範囲の最小値が 100 で、ID が Pod 定義に存在しない場合、100 がこの ID のデフォルト値になります。

Pod の受付プロセスの一環として、Pod で使用可能な SCC が (ほとんどの場合は優先順位の高い SCC から最も制限の厳しい SCC の順序で) 検査され、Pod の要求に最も一致する SCC が選択されます。SCC のストラテジータイプを **RunAsAny** に設定すると、制限が緩和されます。一方、**MustRunAs** タイプに設定すると、制限がより厳しくなります。これらすべてのストラテジーが評価されます。Pod に割り当てられた SCC を確認するには、**oc get pod** コマンドを使用します。

```
# oc get pod <pod_name> -o yaml
...
metadata:
  annotations:
    openshift.io/scc: nfs-scc ❶
  name: nfs-pod1 ❷
  namespace: default ❸
...
```

- ❶ Pod が使用した SCC (この場合は、カスタム SCC) の名前。
- ❷ Pod の名前。
- ❸ プロジェクトの名前。OpenShift Container Platform では、namespace とプロジェクトは置き換え可能な用語として使用されています。詳細は、[Projects and Users](#) を参照してください。

Pod で一致した SCC をすぐに確認できない場合があります。そのため、上記のコマンドは、ライブコンテナの UID、補助グループ、および SELinux のラベル変更を理解するのに非常に役立ちます。

ストラテジーが **RunAsAny** に設定された SCC では、そのストラテジーの特定の値を Pod 定義 (またはイメージ) に定義できます。これがユーザー ID (**runAsUser**) に適用される場合、コンテナが root として実行されないように SCC へのアクセスを制限することが推奨されます。

Pod が **制限付き** SCC に一致することが多くあるため、これに伴うセキュリティーについて理解しておくことが重要です。**制限付き** SCC には以下の特性があります。

- **runAsUser** ストラテジーが **MustRunAsRange** に設定されているため、ユーザー ID が制限されます。これにより、ユーザー ID の検証が強制的に実行されます。

- 許可できるユーザー ID の範囲が SCC で定義されないため (詳細については、`oc get -o yaml --export scc restricted` を参照してください)、プロジェクトの **openshift.io/sa.scc.uid-range** 範囲が範囲のチェックとデフォルト ID に使用されます (必要な場合)。
- デフォルトのユーザー ID は、ユーザー ID が Pod 定義で指定されておらず、一致する SCC の **runAsUser** が **MustRunAsRange** に設定されている場合に生成されます。
- SELinux ラベルが必要です (**seLinuxContext** が **MustRunAs** に設定されているため)。プロジェクトのデフォルトの MCS ラベルが使用されます。
- **FSGroup** ストラテジーが **MustRunAs** に設定され、指定される最初の範囲の最小値を値に使用するように指示されているため、**fsGroup** ID が単一の値に制限されます。
- 許容可能な **fsGroup** ID の範囲が SCC で定義されないため、プロジェクトの **openshift.io/sa.scc.supplemental-groups** の範囲 (またはユーザー ID に使用されるものと同じ範囲) の最小値が検証とデフォルト ID に使用されます (必要な場合)。
- デフォルトの **fsGroup** ID は、**fsGroup** ID が Pod で指定されておらず、一致する SCC の **FSGroup** が **MustRunAs** に設定されている場合に生成されます。
- 範囲チェックが必要でないため、任意の補助グループ ID が許可されます。これは **supplementalGroups** ストラテジーが **RunAsAny** に設定されているためです。
- 実行中の Pod に対してデフォルトの補助グループは生成されません。上記の 2 つのストラテジーについて **RunAsAny** が設定されているためです。したがって、グループが Pod 定義 (またはイメージ) に定義されていない場合は、コンテナには補助グループが事前に定義されません。

以下に、SCC とプロジェクトの相互関係をまとめた **default** プロジェクトとカスタム SCC (**my-custom-scc**) の例を示します。

```
$ oc get project default -o yaml ❶
...
metadata:
  annotations: ❷
    openshift.io/sa.scc.mcs: s0:c1,c0 ❸
    openshift.io/sa.scc.supplemental-groups: 1000000000/10000 ❹
    openshift.io/sa.scc.uid-range: 1000000000/10000 ❺

$ oc get scc my-custom-scc -o yaml
...
fsGroup:
  type: MustRunAs ❻
  ranges:
    - min: 5000
      max: 6000
runAsUser:
  type: MustRunAsRange ❼
  uidRangeMin: 1000100000
  uidRangeMax: 1000100999
seLinuxContext: ❽
  type: MustRunAs
SELinuxOptions: ❾
  user: <selinux-user-name>
  role: ...
```

```

type: ...
level: ...
supplementalGroups:
  type: MustRunAs 10
  ranges:
    - min: 5000
      max: 6000

```

- 1 default はプロジェクトの名前です。
- 2 デフォルト値は、対応する SCC 戦略が **RunAsAny** でない場合にのみ生成されます。
- 3 Pod 定義または SCC で定義されていない場合の SELinux のデフォルト値です。
- 4 許容可能なグループ ID の範囲です。ID の検証は、SCC ストラテジーが **RunAsAny** の場合にのみ実行されます。複数の範囲をコンマで区切って指定できます。[サポートされている形式についてはこちら](#)を参照してください。
- 5 <4>と同じですが、ユーザー ID に使用されます。また、ユーザー ID の単一の範囲のみがサポートされます。
- 6 **10** **MustRunAs** は、グループ ID の範囲チェックを実施して、コンテナのグループのデフォルトを指定します。この SCC の定義に基づく場合、デフォルトは 5000 になります (最小の ID 値)。範囲が SCC から省略される場合、デフォルトは 1000000000 になります (プロジェクトから派生します)。サポートされている別のタイプ **RunAsAny** では、範囲チェックを実行しないため、いずれのグループ ID も許可され、デフォルトグループは生成されません。
- 7 **MustRunAsRange** は、ユーザー ID の範囲チェックを実施して、UID のデフォルトを指定します。この SCC の定義に基づく場合、デフォルトの UID は 1000100000 になります (最小値)。最小範囲と最大範囲が SCC から省略される場合、デフォルトのユーザー ID は 1000000000 になります (プロジェクトから派生します)。これ以外には **MustRunAsNonRoot** と **RunAsAny** のタイプがサポートされます。許可される ID の範囲は、ターゲットのストレージに必要ないずれのユーザー ID も含めるように定義することができます。
- 8 **MustRunAs** に設定した場合は、コンテナは SCC の SELinux オプション、またはプロジェクトに定義される MCS のデフォルトを使用して作成されます。**RunAsAny** というタイプは、SELinux コンテキストが不要であることや、Pod に定義されていない場合はコンテナに設定されないことを示します。
- 9 SELinux のユーザー名、ロール名、タイプ、およびラベルは、ここで定義できます。

2つの形式が許可される範囲にサポートされています。

1. **M/N**。M は開始 ID で N はカウントです。したがって、範囲は M から **M+N-1** (これ自体を含む) までになります。
2. **M-N**。M は同じく開始 ID で N は終了 ID です。デフォルトのグループ ID が最初の範囲の開始 ID になります (このプロジェクトでは **1000000000** desu)。SCC で最小のグループ ID が定義されていない場合は、プロジェクトのデフォルトの ID が適用されます。

### 27.18.3. 補助グループ



## 注記

補助グループの操作にあたっては、事前に [SCC](#)、[デフォルト](#)、[許可される範囲](#) の説明をお読みください。

## ヒント

永続ストレージへのアクセスを取得する場合、通常はグループ ID (補助グループまたは `fsGroup`) を使用する方が [ユーザー ID](#) を使用するよりも適切です。

補助グループは Linux の正規グループです。Linux で実行されるプロセスには、UID、GID、および1つ以上の補助グループがあります。これらの属性はコンテナのメインプロセスで設定されます。`supplementalGroups` ID は、通常は NFS や GlusterFS などの共有ストレージへのアクセスを制御する場合に使用されます。一方、`fsGroup` は Ceph RBD や iSCSI などのブロックストレージへのアクセスを制御する場合に使用されます。

OpenShift Container Platform の共有ストレージプラグインは、マウントの POSIX パーミッションとターゲットストレージのパーミッションが一致するようにボリュームをマウントします。たとえば、ターゲットストレージの所有者 ID が 1234 で、そのグループ ID が 5678 の場合、ホストノードのマウントとコンテナのマウントはそれらの同じ ID を持ちます。したがって、ボリュームにアクセスするためにはコンテナのメインプロセスがこれらの ID の一方または両方と一致する必要があります。

たとえば、以下の NFS エクスポートについて見てみましょう。

OpenShift Container Platform ノード側:



## 注記

`showmount` では、NFS サーバーの `rpcbind` および `rpc.mount` が使用するポートへのアクセスが必要です。

```
# showmount -e <nfs-server-ip-or-hostname>
Export list for f21-nfs.vm:
/opt/nfs *
```

NFS サーバー側:

```
# cat /etc/exports
/opt/nfs *(rw,sync,root_squash)
...

# ls -lZ /opt/nfs -d
drwx-----. 1000100001 5555 unconfined_u:object_r:usr_t:s0 /opt/nfs
```

`/opt/nfs/` エクスポートには UID 1000100001 とグループ 5555 でアクセスすることができます。通常、コンテナは `root` として実行しないようにする必要があります。そのため、この NFS の例では、UID 1000100001 として実行されないコンテナやグループ 5555 のメンバーでないコンテナは、NFS エクスポートにアクセスできません。

多くの場合、Pod と一致する SCC では特定のユーザー ID の指定は許可されません。そのため、Pod へのストレージアクセスを許可する場合には、補助グループを使用する方法はより柔軟な方法として使用できます。たとえば、前述のエクスポートへの NFS アクセスを許可する場合は、グループ 5555 を Pod 定義に定義できます。

■

```

apiVersion: v1
kind: Pod
...
spec:
  containers:
  - name: ...
    volumeMounts:
    - name: nfs ❶
      mountPath: /usr/share/... ❷
  securityContext: ❸
    supplementalGroups: [5555] ❹
  volumes:
  - name: nfs ❺
    nfs:
      server: <nfs_server_ip_or_host>
      path: /opt/nfs ❻

```

- ❶ ボリュームマウントの名前。**volumes** セクションの名前と一致する必要があります。
- ❷ コンテナで表示される NFS エクスポートのパス。
- ❸ Pod のグローバルセキュリティコンテキスト。Pod 内部のすべてのコンテナに適用されます。コンテナではそれぞれ独自の **securityContext** を定義することもできますが、グループ ID は Pod に対してグローバルであり、個々のコンテナに対して定義することはできません。
- ❹ 補助グループ (ID の配列) は 5555 に設定しています。これで、エクスポートへのグループアクセスが許可されます。
- ❺ ボリュームの名前。**volumeMounts** セクションの名前と一致する必要があります。
- ❻ NFS サーバー上の実際の NFS エクスポートのパス。

前述の Pod にあるすべてのコンテナ (一致する SCC またはプロジェクトでグループ 5555 を許可することを前提とします) は、グループ 5555 のメンバーとなり、コンテナのユーザー ID に関係なくボリュームにアクセスできます。ただし、ここでの前提条件に留意してください。場合によっては、SCC が許可されるグループ ID の範囲を定義されておらず、代わりにグループ ID の検証が必要になることがあります (**supplementalGroups** を **MustRunAs** に設定した結果など)。ただし、**制限付き SCC** の場合はこれと異なります。プロジェクトがこの NFS エクスポートにアクセスするようにカスタマイズされていない限り、通常プロジェクトが 5555 というグループ ID を許可することはありません。したがって、このシナリオでは、前述の Pod の 5555 というグループ ID は SCC またはプロジェクトの許可されたグループ ID の範囲内にないために Pod は失敗します。

### 補助グループとカスタム SCC

前述の例にあるような状況に対応するため、以下のようなカスタム SCC を作成することができます。

- 最小と最大のグループ ID を定義する。
- ID の範囲チェックを実施する。
- グループ ID 5555 を許可する。

多くの場合、定義済みの SCC を修正したり、定義済みプロジェクトで許可される ID の範囲を変更したりするよりも、新規の SCC を作成する方が適切です。

新規 SCC を作成するには、既存の SCC をエクスポートし、新規の SCC の要件を満たすように YAML ファイルをカスタマイズするのが最も簡単な方法です。以下に例を示します。

1. 制限付き SCC を新規 SCC のテンプレートとして使用します。

```
$ oc get -o yaml --export scc restricted > new-scc.yaml
```

2. 必要な仕様に合うように `new-scc.yaml` ファイルを編集します。
3. 新規 SCC を作成します。

```
$ oc create -f new-scc.yaml
```



### 注記

`oc edit scc` コマンドを使用して、インスタンス化された SCC を修正することができます。

以下の例は `nfs-scc` という名前の新規 SCC の一部です。

```
$ oc get -o yaml --export scc nfs-scc

allowHostDirVolumePlugin: false ❶
...
kind: SecurityContextConstraints
metadata:
  ...
  name: nfs-scc ❷
priority: 9 ❸
...
supplementalGroups:
  type: MustRunAs ❹
  ranges:
  - min: 5000 ❺
    max: 6000
...
```

- ❶ **allow** ブール値は制限付き SCC の場合と同じです。
- ❷ 新規 SCC の名前。
- ❸ 数値が大きいほど優先順位が高くなります。Nil の場合や省略した場合は優先順位が最も低くなります。優先順位が高い SCC は優先順位が低い SCC より前に並べ替えられるため、新規 Pod と一致する確率が高くなります。
- ❹ **supplementalGroups** はストラテジーであり、**MustRunAs** に設定されています。つまり、グループ ID のチェックが必要になります。
- ❺ 複数の範囲を使用することができます。ここで許可されるグループ ID の範囲は 5000 ~ 5999 で、デフォルトの補助グループは 5000 です。

前述の Pod をこの新規 SCC に対して実行すると (当然ですが Pod が新規 SCC に一致することを前提とします)、Pod が開始されます。これは、Pod 定義で指定したグループ 5555 がカスタム SCC によって許可されるようになったためです。

#### 27.18.4. fsGroup



##### 注記

補助グループの操作にあたっては、事前に [SCC](#)、[デフォルト](#)、[許可される範囲](#) の説明をお読みください。

##### ヒント

永続ストレージへのアクセスを取得する場合、通常はグループ ID ([補助](#) グループまたは **fsGroup**) を使用する方が [ユーザー ID](#) を使用するよりも適切です。

**fsGroup** は Pod のファイルシステムグループの ID を定義するもので、コンテナの補助グループに追加されます。**supplementalGroups** の ID は共有ストレージに適用されますが、**fsGroup** の ID はブロックストレージに使用されます。

ブロックストレージ (Ceph RBD、iSCSI、各種クラウドストレージなど) は通常、ブロックストレージボリュームを直接に、または PVC を使用して要求した単一 Pod に専用のものとして設定されます。共有ストレージとは異なり、ブロックストレージは Pod によって引き継がれます。つまり、Pod 定義 (またはイメージ) で指定されたユーザー ID とグループ ID が実際の物理ブロックデバイスに適用されます。通常、ブロックストレージは共有されません。

以下の **fsGroup** の定義は Pod 定義の一部です。

```
kind: Pod
...
spec:
  containers:
  - name: ...
    securityContext: ①
      fsGroup: 5555 ②
  ...
```

- ① **supplementalGroups** と同じように、**fsGroup** はコンテナごとに定義するのではなく Pod に対してグローバルに定義する必要があります。
- ② 5555 はボリュームのグループパーミッションのグループ ID になり、ボリュームで作成されるすべての新規ファイルのグループ ID になります。

**supplementalGroups** と同様に、前述の Pod にあるすべてのコンテナ (一致する SCC またはプロジェクトでグループ 5555 を許可することを前提とします) は、グループ 5555 のメンバーとなり、コンテナのユーザー ID に関係なくブロックボリュームにアクセスできます。Pod が **制限付き** SCC に一致し、その **fsGroup** ストラテジーが **MustRunAs** である場合、Pod の実行は失敗します。しかし、SCC の **fsGroup** ストラテジーを **RunAsAny** に設定した場合は、任意の **fsGroup** ID (5555 を含む) が許可されます。SCC の **fsGroup** ストラテジーを **RunAsAny** に設定して、**fsGroup** ID を指定しない場合は、ブロックストレージの引き継ぎは行われず、Pod へのパーミッションが拒否される可能性があります。

#### fsGroups とカスタム SCC

前述の例にあるような状況に対応するため、以下のようにカスタム SCC を作成することができます。

- 最小と最大のグループ ID を定義する。
- ID の範囲チェックを実施する。
- グループ ID 5555 を許可する。

定義済みの SCC を修正したり、定義済みプロジェクトで許可される ID の範囲を変更したりするよりも、新規の SCC を作成する方が適切です。

以下の新規 SCC の定義の一部について見てみましょう。

```
# oc get -o yaml --export scc new-scc
...
kind: SecurityContextConstraints
...
fsGroup:
  type: MustRunAs ①
  ranges: ②
  - max: 6000
    min: 5000 ③
...
```

- ① **MustRunAs** ではグループ ID の範囲チェックをトリガーします。一方、**RunAsAny** では範囲チェックは必要ありません。
- ② 許可されるグループ ID の範囲は 5000 ~ 5999 (これら自体を含む) です。複数の範囲がサポートされていますが、1つしか使用していません。ここで許可されるグループ ID の範囲は 5000 ~ 5999 で、デフォルトの **fsGroup** は 5000 です。
- ③ 最小値 (または範囲全体) を SCC から省略することができます。その場合、範囲のチェックとデフォルト値の生成はプロジェクトの **openshift.io/sa.scc.supplemental-groups** の範囲に従います。**fsGroup** と **supplementalGroups** ではプロジェクト内の同じグループフィールドが使用されます。**fsGroup** に別の範囲が存在する訳ではありません。

前述の Pod をこの新規 SCC に対して実行すると (当然ですが Pod が新しい SCC に一致することを前提とします)、Pod が開始されます。これは、Pod 定義で指定したグループ 5555 がカスタム SCC によって許可されているためです。また、Pod でブロックデバイスの引き継ぎが行われます。そのため、ブロックストレージを Pod の外部のプロセスから表示する場合、そのグループ ID は実際には 5555 になります。

以下は、ブロックの所有権をサポートしているボリュームの例です。

- AWS Elastic Block Store
- OpenStack Cinder
- Ceph RBD
- GCE Persistent Disk
- iSCSI
- emptyDir



## 注記

この一覧にはすべてが網羅されている訳ではありません。

### 27.18.5. ユーザー ID



## 注記

補助グループの操作にあたっては、事前に [SCC](#)、[デフォルト](#)、[許可される範囲](#) の説明をお読みください。

## ヒント

永続ストレージへのアクセスを取得する場合、通常はグループ ID ([補助](#) グループまたは `fsGroup`) を使用する方がユーザー ID を使用するよりも適切です。

ユーザー ID は、コンテナイメージまたは Pod 定義で定義できます。Pod 定義では、1つのユーザー ID をすべてのコンテナに対してグローバルに定義するか、個々のコンテナに固有のものとして定義するか、またはその両方として定義できます。以下の Pod 定義の一部ではユーザー ID を指定しています。

```
spec:
  containers:
  - name: ...
    securityContext:
      runAsUser: 1000100001
```

1000100001 はコンテナ固有の ID であり、エクスポートの所有者 ID と一致します。NFS エクスポートの所有者 ID が 54321 である場合は、その番号が Pod 定義で使用されます。コンテナ定義の外部で **securityContext** を指定すると、ID は Pod 内のすべてのコンテナに対してグローバルになります。

グループ ID と同じように、SCC やプロジェクトで設定されているポリシーに従ってユーザー ID を検証することもできます。SCC の **runAsUser** ストラテジーを **RunAsAny** に設定した場合は、Pod 定義またはイメージで定義されているすべてのユーザー ID が許可されます。



## 警告

つまり、0 (root) の UID さえも許可されることとなります。

代わりに **runAsUser** ストラテジーを **MustRunAsRange** に設定した場合は、指定したユーザー ID について、許可される ID の範囲にあるかどうかを検証されます。Pod でユーザー ID を指定しない場合は、デフォルト ID が許可されるユーザー ID の範囲の最小値に設定されます。

先の [NFS の例](#) に戻って、コンテナでその UID を 1000100001 に設定する必要があります (上記の Pod の例を参照してください)。デフォルトプロジェクトと制限付き SCC を想定した場合、Pod で要求した 1000100001 というユーザー ID は許可されず、したがって Pod は失敗します。Pod が失敗するのは以下の理由によります。

- Pod が独自のユーザー ID として 1000100001 を要求している。

- 使用可能なすべての SCC が独自の **runAsUser** ストラテジーとして **MustRunAsRange** を使用しており、そのため UID の範囲チェックが要求される。
- 1000100001 が SCC にもプロジェクトのユーザー ID 範囲にも含まれていない。

この状況に対応するには、適切なユーザー ID 範囲を指定して新規 SCC を作成します。また、新規プロジェクトを適切なユーザー ID 範囲を定義して作成することもできます。さらに、以下のような推奨されない他のオプションがあります。

- 最小および最大のユーザー ID 範囲に 1000100001 を組み込むように **制限付き SCC** を変更できます。ただし、これは定義済みの SCC の変更をできる限り避ける必要があるため推奨されません。
- **RunAsAny** を **runAsUser** の値に使用できるように **制限付き SCC** を変更できます。これにより、ID 範囲のチェックを省略できます。この方法ではコンテナが root として実行される可能性があるため、この方法を使用しないことを強く推奨します。
- ユーザー ID 1000100001 を許可するように **デフォルトプロジェクト**の UID 範囲を変更できません。通常、この方法も推奨できません。ユーザー ID に単一範囲しか指定できず、範囲が変更された場合に他の Pod が実行されなくなる可能性があるためです。

### ユーザー ID とカスタム SCC

定義済みの SCC を変更することは可能な限り避ける必要があります。組織のセキュリティー上のニーズに合ったカスタム SCC を作成するか、または必要なユーザー ID をサポートする **新規プロジェクトを作成する** を推奨します。

前述の例にあるような状況に対応するため、以下のようにカスタム SCC を作成することができます。

- 最小と最大のユーザー ID を定義する。
- UID の範囲チェックを引き続き実施する。
- 1000100001 という UID を許可する。

以下に例を示します。

```
$ oc get -o yaml --export scc nfs-scc

allowHostDirVolumePlugin: false ❶
...
kind: SecurityContextConstraints
metadata:
  ...
  name: nfs-scc ❷
priority: 9 ❸
requiredDropCapabilities: null
runAsUser:
  type: MustRunAsRange ❹
  uidRangeMax: 1000100001 ❺
  uidRangeMin: 1000100001
...
```

❶ **allowXX** のブール値は**制限付き SCC** の場合と同じです。

❷ この新規 SCC の名前は **nfs-scc** です。

- 3 数値が大きいほど優先順位が高くなります。Nil の場合や省略した場合は優先順位が最も低くなります。優先順位が高い SCC は優先順位が低い SCC より前に並べ替えられるため、新規 Pod と一
- 4 **runAsUser** ストラテジーは **MustRunAsRange** に設定されています。つまり、UID の範囲チェックが実施されます。
- 5 UID の範囲は 1000100001 ~ 1000100001 です (1つの値の範囲)。

これで、先の例の Pod 定義に **runAsUser: 1000100001** が表示され、Pod が新規の **nfs-scc** と一致し、UID 1000100001 で実行できるようになります。

## 27.18.6. SELinux オプション

特権付き SCC を除くすべての定義済み SCC では、**seLinuxContext** を **MustRunAs** に設定します。そのため、Pod の要件と一致する可能性が高い SCC の場合、Pod での SELinux ポリシーの使用を強制的に実行します。Pod で使用される SELinux ポリシーは、その Pod 自体やイメージ、SCC、またはプロジェクト (デフォルトを指定する) で定義できます。

SELinux のラベルは Pod の **securityContext.seLinuxOptions** セクションで定義でき、**user**、**role**、**type**、および **level** を使用できます。



### 注記

このトピックでは、レベルと MCS ラベルを置き換え可能な用語として使用します。

```
...
securityContext: 1
  seLinuxOptions:
    level: "s0:c123,c456" 2
...
```

- 1 **level** は、Pod 全体に対してグローバルに定義することも、コンテナごとに個別に定義することもできます。
- 2 SELinux の level ラベル。

以下の例は SCC とデフォルトプロジェクトからの抜粋です。

```
$ oc get -o yaml --export scc scc-name
...
seLinuxContext:
  type: MustRunAs 1

# oc get -o yaml --export namespace default
...
metadata:
  annotations:
    openshift.io/sa.scc.mcs: s0:c1,c0 2
...
```

- 1 **MustRunAs** によりボリュームのラベルが再設定されます。

- 2 ラベルを Pod や SCC で指定しない場合は、プロジェクトのデフォルトが適用されます。

特権付き SCC を除くすべての定義済み SCC では、**seLinuxContext** を **MustRunAs** に設定します。これにより、Pod での MCS ラベルの使用が強制的に実行されます。MCS ラベルは、Pod 定義やイメージで定義するか、またはデフォルトとして指定されます。

SCC によって、SELinux ラベルを必要とするかどうかが決まります。また、SCC でデフォルトラベルを指定できます。**seLinuxContext** ストラテジーを **MustRunAs** に設定していて、Pod (またはイメージ) がラベルを定義していない場合は、OpenShift Container Platform は SCC 自体またはプロジェクトから選択されるラベルにデフォルト設定されます。

**seLinuxContext** を **RunAsAny** に設定した場合は、デフォルトラベルは提供されず、コンテナによって最終的なラベルが決められます。Docker の場合、コンテナでは一意の MCS ラベルが使用されますが、これが既存のストレージマウントのラベル付けに一致する可能性はほとんどありません。SELinux 管理をサポートしているボリュームについては、指定されるラベルからアクセスできるようにラベルの再設定がなされ、ラベルの排他性によってはそのラベルのみがアクセスできるようになります。

この場合、特権なしコンテナについては以下の 2 つが関係します。

- ボリュームには、特権なしのコンテナからのアクセス可能なタイプが指定されます。このタイプは、通常は Red Hat Enterprise Linux (RHEL) バージョン 7.5 以降の **container\_file\_t** になります。このタイプはボリュームをコンテナコンテキストとして処理します。以前の RHEL バージョンの RHEL 7.4、7.3 などでは、ボリュームに **svirt\_sandbox\_file\_t** タイプが指定されます。
- **level** を指定した場合は、指定される MCS ラベルを使用してボリュームのラベルが設定されます。

Pod からボリュームにアクセスできるようにするためには、Pod で両方のボリュームカテゴリを持つ必要があります。そのため、**s0:c1,c2** の Pod は、**s0:c1,c2** のボリュームにアクセスできます。**s0** のボリュームは、すべての Pod からアクセスできます。

Pod で承認が失敗する場合、またはパーミッションエラーが原因でストレージのマウントが失敗している場合は、SELinux の実施による干渉が生じている可能性があります。これについては、たとえば以下を実行してチェックできます。

```
# ausearch -m avc --start recent
```

これは、ログファイルに AVC (Access Vector Cache) のエラーがないかどうかを検査します。

## 27.19. セレクターとラベルによるボリュームのバインディング

### 27.19.1. 概要

ここでは、**selector** 属性と **label** 属性を使用して Persistent Volume Claim (永続ボリューム要求、PVC) を永続ボリューム (PV) にバインドするための必要な手順について説明します。セレクターとラベルを実装することで、通常のユーザーは、クラスター管理者が定義する識別子を使用して、**プロビジョニングされたストレージ** をターゲットに設定することができます。

### 27.19.2. 必要になる状況

静的にプロビジョニングされるストレージの場合、永続ストレージを必要とする開発者は、PVC をデプロイしてバインドするためにいくつかの PV の識別属性を把握しておく必要があります。その際、問題となる状況がいくつか生じます。通常のユーザーは、PVC のデプロイでも PV の値の指定においても、

クラスター管理者に問い合わせをする必要が生じる場合があります。PV 属性のみでは、ストレージボリュームの使用目的を達成できず、ボリュームのグループ化をする手段も提供できません。

selector 属性と label 属性を使用すると、ユーザーが意識せずに済むように PV の詳細を抽象化できると同時に、分かりやすくカスタマイズ可能なタグを使用してボリュームを識別する手段をクラスター管理者に提供できます。セレクターとラベルによるバインディングの方法を使用することで、ユーザーは管理者によって定義されているラベルのみを確認することが必要になります。



### 注記

セレクターとラベルの機能は、現時点では**静的に**プロビジョニングされるストレージの場合にのみ使用できます。現時点では、動的にプロビジョニングされるストレージ用には実装されていません。

## 27.19.3. Deployment

このセクションでは、PVC の定義方法とデプロイ方法を説明します。

### 27.19.3.1. 前提条件

1. 実行中の OpenShift Container Platform 3.3 以降のクラスター
2. サポート対象の [ストレージプロバイダー](#) によって提供されているボリューム
3. cluster-admin ロールのバインディングを持つユーザー

### 27.19.3.2. 永続ボリュームと要求の定義

1. cluster-admin ユーザーとして、PV を定義します。この例では [GlusterFS](#) ボリュームを使用します。プロバイダーの設定については、該当する [ストレージプロバイダー](#) を参照してください。

#### 例27.9 ラベルのある永続ボリューム

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-volume
  labels: ❶
    volume-type: ssd
    aws-availability-zone: us-east-1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster
    path: myVol1
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

- ❶ セレクターが**すべての** PV のラベルと一致する PVC がバインドされます (PV が使用可能であることを前提とします)。

## 2. PVC を定義します。

## 例27.10 セレクターのある Persistent Volume Claim (永続ボリューム要求)

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector: ❶
    matchLabels: ❷
      volume-type: ssd
      aws-availability-zone: us-east-1

```

❶ selectors セクションの始まりです。

❷ ユーザーがストレージを要求する際に使用するラベルすべてを一覧表示します。ターゲットとなる PV のすべてのラベルと一致する必要があります。

## 27.19.3.3. オプション: PVC の特定の PV へのバインド

PV 名またはセレクターを指定しない PVC はすべての PV と同じになります。

PVC をクラスター管理者として特定の PV にバインドするには、以下を実行します。

- PV 名を把握している場合は **pvc.spec.volumeName** を使用します。
- PV ラベルを把握している場合は **pvc.spec.selector** を使用します。  
セレクターを指定すると、PVC には PV に特定のラベルを指定する必要があります。

## 27.19.3.4. オプション: 特定の PVC に対する PV の確保

特定のタスク用に PV を確保するには、特定のストレージクラスを作成するか、または PV を PVC に事前にバインドするかの、2つのオプションがあります。

1. ストレージクラスの名前を指定して PV の特定のストレージクラスを要求します。  
以下のリソースは、StorageClass の設定に使用する必要な値を示しています。この例では、AWS ElasticBlockStore (EBS) オブジェクト定義を使用します。

## 例27.11 EBS の StorageClass 定義

```

kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: kafka
provisioner: kubernetes.io/aws-efs
...

```

**重要**

マルチテナント環境で必要な場合に、クォータ定義を使用してストレージクラスと PV を特定の namespace 専用として確保します。

2. PVC namespace および名前を使用して PV を PVC に事前バインドします。以下の例のように、このように定義された PV は指定された PVC のみにバインドされ、それ以外にはバインドされません。

**例27.12 PV 定義の claimRef**

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: mktg-ops--kafka--kafka-broker01
spec:
  capacity:
    storage: 15Gi
  accessModes:
    - ReadWriteOnce
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: kafka-broker01
    namespace: default
...

```

**27.19.3.5. 永続ボリュームと要求のデプロイ**

cluster-admin ユーザーとして、永続ボリュームを作成します。

**例27.13 永続ボリュームの作成**

```

# oc create -f gluster-pv.yaml
persistentVolume "gluster-volume" created

# oc get pv
NAME          LABELS    CAPACITY  ACCESSMODES  STATUS    CLAIM    REASON
AGE
gluster-volume  map[]    2147483648  RWX          Available             2s

```

PV が作成されると、セレクターがそのすべてのラベルと一致するユーザーであれば PVC を作成できます。

**例27.14 Persistent Volume Claim (永続ボリューム要求) の作成**

```

# oc create -f gluster-pvc.yaml
persistentVolumeClaim "gluster-claim" created
# oc get pvc

```

NAME	LABELS	STATUS	VOLUME
gluster-claim		Bound	gluster-volume

## 27.20. コントローラー管理の割り当ておよび割り当て解除

### 27.20.1. 概要

デフォルトでは、ノードセット自体による各自のボリュームの割り当て/割り当て解除操作の管理のままにするのではなく、クラスターのマスターで実行されるコントローラーがノードセットに代わってボリュームの割り当て/割り当て解除操作を管理することができます。

コントローラー管理の割り当ておよび割り当て解除には、以下のメリットがあります。

- ノードが失われた場合でも、そのノードに割り当てられていたボリュームの割り当て解除をコントローラーによって実行でき、これを別の場所で再び割り当てることができます。
- 割り当て/割り当て解除に必要な認証情報をすべてのノードで用意する必要がないため、セキュリティが強化されます。

### 27.20.2. 割り当て/割り当て解除の管理元の判別

ノード自体にアノテーション `volumes.kubernetes.io/controller-managed-attach-detach` が設定されている場合、そのノードの割り当て/割り当て解除操作はコントローラーによって管理されます。コントローラーは、すべてのノードでこのアノテーションの有無を自動的に検査し、その有無に応じて動作します。したがって、ユーザーがノードでこのアノテーションの有無を調べることで、コントローラーが管理する割り当て/割り当て解除がそのノードで有効にされているかどうかを判別することができます。

さらに、ノードでコントローラー管理の割り当て/割り当て解除が選択されていることを確認するには、ノードのログで以下の行を検索します。

```
Setting node annotation to enable volume controller attach/detach
```

この行が見つからない場合は、以下の行が代わりにログに含まれているはずです。

```
Controller attach/detach is disabled for this node; Kubelet will attach and detach volumes
```

コントローラーの端末から、コントローラーが特定ノードの割り当て/割り当て解除操作を管理しているかどうかを確認するには、まずロギングレベルを少なくとも **4** に設定する必要があります。次に、以下の行を見つけます。

```
processVolumesInUse for node <node_hostname>
```

ログの表示方法とロギングレベルの設定方法については、[ロギングレベルの設定](#) を参照してください。

### 27.20.3. コントローラー管理の割り当て/割り当て解除を有効にするノードの設定

コントローラー管理の割り当て/割り当て解除を有効にするには、個々のノードで独自の割り当て/割り当て解除をオプトインし、無効にするように設定します。編集対象のノード設定ファイルについての詳細は、[ノード設定ファイル](#) を参照してください。このファイルには、以下の行を追加します。

```
kubeletArguments:
  enable-controller-attach-detach:
    - "true"
```

ノードを設定したら、設定を有効にするためにノードを再起動する必要があります。

## 27.21. 永続ボリュームスナップショット

### 27.21.1. 概要



#### 重要

永続ボリュームスナップショットはテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、<https://access.redhat.com/support/offerings/techpreview/> を参照してください。

ストレージシステムの多くは、データを損失から保護するために永続ボリューム (PV) のスナップショットを作成する機能を備えています。外部のスナップショットコントローラーおよびプロビジョナーは、この機能を OpenShift Container Platform クラスターで使用して OpenShift Container Platform API を使用してボリュームスナップショットを扱う方法を提供しています。

本書では、OpenShift Container Platform におけるボリュームスナップショットのサポートの現状について説明しています。PV、[Persistent Volume Claim \(永続ボリューム要求、PVC\)](#)、および [動的プロビジョニング](#) について理解しておくことが推奨されます。

### 27.21.2. 機能

- **PersistentVolumeClaim** にバインドされる **PersistentVolume** のスナップショットの作成
- 既存の **VolumeSnapshots** の一覧表示
- 既存の **VolumeSnapshot** の削除
- 既存の **VolumeSnapshot** からの **PersistentVolume** の新規作成
- サポートされている **PersistentVolume** [タイプ](#):
  - AWS Elastic Block Store (EBS)
  - Google Compute Engine (GCE) Persistent Disk (PD)

### 27.21.3. インストールと設定

外部のスナップショットコントローラーおよびプロビジョナーは、ボリュームスナップショットの機能を提供する外部コンポーネントです。これらの外部コンポーネントはクラスターで実行されます。コントローラーは、ボリュームスナップショットの作成、削除、および関連イベントのレポートを行います。プロビジョナーは、ボリュームスナップショットから新規の **PersistentVolumes** を作成します。詳細は、[スナップショットの作成](#) および [スナップショットの復元](#) を参照してください。

### 27.21.3.1. 外部のコントローラーおよびプロビジョナーの起動

外部のコントローラーおよびプロビジョナーサービスはコンテナイメージとして配布され、OpenShift Container Platform クラスタで通常どおり実行できます。また、コントローラーおよびプロビジョナーの RPM バージョンもあります。

API オブジェクトを管理しているコンテナを許可するには、以下のようにして、必要なロールベースアクセス制御 (RBAC) ルールを管理者が設定する必要があります。

1. **ServiceAccount** と **ClusterRole** を以下のように作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: snapshot-controller-runner
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: snapshot-controller-role
rules:
- apiGroups: [""]
  resources: ["persistentvolumes"]
  verbs: ["get", "list", "watch", "create", "delete"]
- apiGroups: [""]
  resources: ["persistentvolumeclaims"]
  verbs: ["get", "list", "watch", "update"]
- apiGroups: ["storage.k8s.io"]
  resources: ["storageclasses"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["events"]
  verbs: ["list", "watch", "create", "update", "patch"]
- apiGroups: ["apiextensions.k8s.io"]
  resources: ["customresourcedefinitions"]
  verbs: ["create", "list", "watch", "delete"]
- apiGroups: ["volumesnapshot.external-storage.k8s.io"]
  resources: ["volumesnapshots"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["volumesnapshot.external-storage.k8s.io"]
  resources: ["volumesnapshotdatas"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

2. **ClusterRoleBinding** を使用して、以下のようにルールをバインドします。

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: snapshot-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: snapshot-controller-role
subjects:
- kind: ServiceAccount
  name: snapshot-controller-runner
  namespace: default
```

外部のコントローラーおよびプロビジョナーを Amazon Web Services (AWS) にデプロイしている場合、それらはアクセスキーを使用して認証する必要があります。認証情報を Pod に提供するために、管理者が以下のように新規のシークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: awskeys
type: Opaque
data:
  access-key-id: <base64 encoded AWS_ACCESS_KEY_ID>
  secret-access-key: <base64 encoded AWS_SECRET_ACCESS_KEY>
```

AWS における外部のコントローラーおよびプロビジョナーコンテナのデプロイメント (どちらの Pod コンテナもシークレットを使用して AWS のクラウドプロバイダー API にアクセスします):

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: snapshot-controller
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: snapshot-controller
    spec:
      serviceAccountName: snapshot-controller-runner
      containers:
        - name: snapshot-controller
          image: "registry.redhat.io/openshift3/snapshot-controller:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "aws"]
          env:
            - name: AWS_ACCESS_KEY_ID
              valueFrom:
                secretKeyRef:
                  name: awskeys
                  key: access-key-id
            - name: AWS_SECRET_ACCESS_KEY
              valueFrom:
                secretKeyRef:
                  name: awskeys
                  key: secret-access-key
        - name: snapshot-provisioner
          image: "registry.redhat.io/openshift3/snapshot-provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "aws"]
          env:
            - name: AWS_ACCESS_KEY_ID
              valueFrom:
                secretKeyRef:
                  name: awskeys
```

```

    key: access-key-id
  - name: AWS_SECRET_ACCESS_KEY
    valueFrom:
      secretKeyRef:
        name: awskeys
        key: secret-access-key

```

GCE の場合、GCE のクラウドプロバイダー API にアクセスするためにシークレットを使用する必要はありません。管理者は以下のようにデプロイメントに進むことができます。

```

kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: snapshot-controller
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: snapshot-controller
    spec:
      serviceAccountName: snapshot-controller-runner
      containers:
        - name: snapshot-controller
          image: "registry.redhat.io/openshift3/snapshot-controller:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "gce"]
        - name: snapshot-provisioner
          image: "registry.redhat.io/openshift3/snapshot-provisioner:latest"
          imagePullPolicy: "IfNotPresent"
          args: ["-cloudprovider", "gce"]

```

### 27.21.3.2. スナップショットユーザーの管理

クラスターの設定によっては、管理者以外のユーザーが API サーバーで **VolumeSnapshot** オブジェクトを操作できるようにする必要があります。これは、特定のユーザーまたはグループにバインドされる **ClusterRole** を作成することで実行できます。

たとえば、ユーザー `alice` がクラスター内のスナップショットを操作する必要があるとします。クラスター管理者は以下の手順を実行します。

1. 新規の **ClusterRole** を定義します。

```

apiVersion: v1
kind: ClusterRole
metadata:
  name: volumesnapshot-admin
rules:
  - apiGroups:
    - "volumesnapshot.external-storage.k8s.io"
    attributeRestrictions: null
  resources:
    - volumesnapshots

```

```
verbs:
- create
- delete
- deletecollection
- get
- list
- patch
- update
- watch
```

2. **ClusterRole** バインドオブジェクトを作成してクラスターロールをユーザー alice にバインドします。

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: volumesnapshot-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: volumesnapshot-admin
subjects:
- kind: User
  name: alice
```



### 注記

これは API アクセス設定の一例にすぎません。**VolumeSnapshot** オブジェクトの動作は他の OpenShift Container Platform API オブジェクトと同様です。API RBAC の管理についての詳細は、[API アクセス制御についてのドキュメント](#) を参照してください。

## 27.21.4. ボリュームスナップショットとボリュームスナップショットデータのライフサイクル

### 27.21.4.1. Persistent Volume Claim (永続ボリューム要求) と永続ボリューム

**PersistentVolumeClaim** は **PersistentVolume** にバインドされます。**PersistentVolume** のタイプは、スナップショットがサポートするいずれかの永続ボリュームタイプである必要があります。

#### 27.21.4.1.1. スナップショットプロモーター

**StorageClass** を作成するには、以下を実行します。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: snapshot-promoter
provisioner: volumesnapshot.external-storage.k8s.io/snapshot-promoter
```

この **StorageClass** は、先に作成した **VolumeSnapshot** から **PersistentVolume** を復元する場合に必要です。

### 27.21.4.2. スナップショットの作成

PVのスナップショットを作成するには、新しい **VolumeSnapshot** オブジェクトを以下のように作成します。

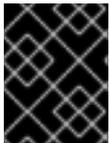
```
apiVersion: volumesnapshot.external-storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: snapshot-demo
spec:
  persistentVolumeClaimName: ebs-pvc
```

**persistentVolumeClaimName** は、**PersistentVolume** にバインドされる **PersistentVolumeClaim** の名前です。この特定 PV のスナップショットが作成されます。

次に、**VolumeSnapshot** に基づく **VolumeSnapshotData** オブジェクトが自動的に作成されます。**VolumeSnapshot** と **VolumeSnapshotData** の関係は **PersistentVolumeClaim** と **PersistentVolume** の関係に似ています。

PV のタイプによっては、反映される **VolumeSnapshot** の状態に応じ、操作が複数の段階にわたる場合があります。

1. 新規 **VolumeSnapshot** オブジェクトが作成されます。
2. コントローラーによってスナップショット操作が開始されます。スナップショット対象の **PersistentVolume** をフリーズし、アプリケーションを一時停止する必要がある場合があります。
3. ストレージシステムによるスナップショットの作成が完了し (スナップショットが切り取られ)、スナップショット対象の **PersistentVolume** が通常の操作に戻ります。スナップショット自体の準備はまだできていません。ここでの状態は **Pending** タイプで状態の値は **True** です。実際のスナップショットを表す **VolumeSnapshotData** オブジェクトが新たに作成されます。
4. 新規スナップショットの作成が完成し、使用できる状態になります。ここでの状態は **Ready** タイプで、状態の値は **True** です。



### 重要

データの整合性の確保 (Pod/アプリケーションの停止、キャッシュのフラッシュ、ファイルシステムのフリーズなど) はユーザーの責任で行う必要があります。



### 注記

エラーの場合は、**VolumeSnapshot** の状態が **Error** 状態に追加されます。

**VolumeSnapshot** の状態を表示するには、以下を実行します。

```
$ oc get volumesnapshot -o yaml
```

状態が以下のように表示されます。

```
apiVersion: volumesnapshot.external-storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  clusterName: ""
  creationTimestamp: 2017-09-19T13:58:28Z
```

```

generation: 0
labels:
  Timestamp: "1505829508178510973"
name: snapshot-demo
namespace: default
resourceVersion: "780"
selfLink: /apis/volumesnapshot.external-
storage.k8s.io/v1/namespaces/default/volumesnapshots/snapshot-demo
uid: 9cc5da57-9d42-11e7-9b25-90b11c132b3f
spec:
  persistentVolumeClaimName: ebs-pvc
  snapshotDataName: k8s-volume-snapshot-9cc8813e-9d42-11e7-8bed-90b11c132b3f
status:
  conditions:
  - lastTransitionTime: null
    message: Snapshot created successfully
    reason: ""
    status: "True"
    type: Ready
  creationTimestamp: null

```

### 27.21.4.3. スナップショットの復元

**VolumeSnapshot** から PV を復元するには、以下のように PVC を作成します。

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: snapshot-pv-provisioning-demo
  annotations:
    snapshot.alpha.kubernetes.io/snapshot: snapshot-demo
spec:
  storageClassName: snapshot-promoter

```

アノテーション:**snapshot.alpha.kubernetes.io/snapshot** は、復元する **VolumeSnapshot** の名前です。**storageClassName:StorageClass** は **VolumeSnapshot** を復元するために管理者が作成します。

新規の **PersistentVolume** が作成されて **PersistentVolumeClaim** にバインドされます。PV のタイプによっては処理に数分の時間がかかることがあります。

### 27.21.4.4. スナップショットの削除

**snapshot-demo** を削除するには、以下を実行します。

```
$ oc delete volumesnapshot/snapshot-demo
```

**VolumeSnapshot** にバインドされている **VolumeSnapshotData** が自動的に削除されます。

## 27.22. HOSTPATH の使用

OpenShift Container Platform クラスター内の **hostPath** ボリュームは、ファイルまたはディレクトリをホストノードのファイルシステムから Pod にマウントします。ほとんどの Pod には **hostPath** ボリュームは必要ありませんが、アプリケーションが必要とする場合は、テスト用のクイックオプションが提供されます。



## 重要

クラスター管理者は、特権付き Pod として実行するように Pod を設定する必要があります。これにより、同じノードの Pod へのアクセスが付与されます。

### 27.22.1. 概要

OpenShift Container Platform は単一ノードクラスターでの開発およびテスト用の **hostPath** マウントをサポートします。

実稼働クラスターでは、**hostPath** を使用しません。代わりにクラスター管理者は、GCE Persistent Disk ボリューム、Amazon EBS ボリュームなどのネットワークリソースをプロビジョニングします。ネットワークリソースは、ストレージクラスを使用した動的プロビジョニングの設定をサポートします。

**hostPath** ボリュームは静的にプロビジョニングする必要があります。

### 27.22.2. Pod 仕様での hostPath ボリュームの設定

**hostPath** ボリュームを使用して、ノード上の **読み取り/書き込み** ファイルにアクセスできます。これは、内部からホストを設定してモニターできる Pod に役立ちます。**hostPath** ボリュームを使用して、**mountPropagation** でホストにボリュームをマウントすることもできます。



## 警告

**hostPath** ボリュームを使用すると、Pod がホスト上の任意のファイルを読み書きできるため、リスクが伴います。注意して進めてください。

**PersistentVolume** オブジェクトではなく、**Pod** 仕様に **hostPath** ボリュームを直接指定することが推奨されます。この操作は、ノードの設定時にアクセスする必要があるパスがすでに Pod が認識しているので便利です。

## 手順

1. 特権付き Pod を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-name
spec:
  containers:
  ...
  securityContext:
    privileged: true
  volumeMounts:
  - mountPath: /host/etc/motd.confg 1
    name: hostpath-privileged
  ...
volumes:
```

```
- name: hostpath-privileged
  hostPath:
    path: /etc/motd.confg ②
```

- ① 特権付き Pod 内への **hostPath** 共有のマウントに使用されるパス。
- ② 特権付き Pod との共有に使用されるホストのパス。

この例では、Pod は **/etc/motd.confg** 内のホストのパスを **/host/etc/motd.confg** として確認できます。したがって、ホストに直接アクセスせずに **motd** を設定できます。

### 27.22.3. hostPath ボリュームの静的なプロビジョニング

**hostPath** ボリュームを使用する Pod は、手動の (または静的) プロビジョニングで参照される必要があります。



#### 注記

永続ストレージがない場合にのみ、**hostPath** での永続ボリュームを使用してください。

#### 手順

1. 永続ボリューム (PV) を定義します。 **PersistentVolume** オブジェクト定義を使用して **pv.yaml** ファイルを作成します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume ①
  labels:
    type: local
spec:
  storageClassName: manual ②
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce ③
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/mnt/data" ④
```

- ① ボリュームの名前。この名前は永続ボリューム要求 (PVC) または Pod で識別されるものです。
- ② 永続ボリューム要求 (PVC) をこの永続ボリュームにバインドするために使用されます。
- ③ ボリュームは単一ノードで **read-write** としてマウントできます。
- ④ 設定ファイルでは、ボリュームがクラスターのノードの **/mnt/data** にあるように指定します。

2. ファイルから PV を作成します。

```
$ oc create -f pv.yaml
```

- 永続ボリューム要求 (PVC) を定義します。**PersistentVolumeClaim** オブジェクト定義を使用して **pvc.yaml** ファイルを作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pvc-volume
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: manual
```

- ファイルから PVC を作成します。

```
$ oc create -f pvc.yaml
```

#### 27.22.4. hostPath 共有の特権付き Pod でのマウント

永続ボリューム要求 (PVC) の作成後に、アプリケーションによって Pod 内で使用できます。以下の例は、この共有を Pod 内にマウントする方法を示しています。

##### 前提条件

- 基礎となる **hostPath** 共有にマップされる永続ボリューム要求 (PVC) があること。

##### 手順

- 既存の永続ボリューム要求 (PVC) をマウントする特権付き Pod を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-name ①
spec:
  containers:
    ...
  securityContext:
    privileged: true ②
  volumeMounts:
    - mountPath: /data ③
      name: hostpath-privileged
    ...
  securityContext: {}
  volumes:
    - name: hostpath-privileged
      persistentVolumeClaim:
        claimName: task-pvc-volume ④
```

- ① Pod の名前。
- ② Pod は、ノードのストレージにアクセスするために特権付き Pod として実行される必要があります。
- ③ 特権付き Pod 内に hostPath 共有をマウントするパス。
- ④ 以前に作成された **PersistentVolumeClaim** オブジェクトの名前。

#### 27.22.5. 関連情報

- [マウントの伝播](#)

## 第28章 永続ストレージの例

### 28.1. 概要

以下のセクションでは、一般的なストレージのユースケースを定義するための総合的な手順について詳しく説明します。以下の例では、永続ボリュームとそのセキュリティの管理だけでなく、システムの利用者がボリュームに対する要求を行う方法についても取り上げます。

- [2つのPod間でのNFS PVの共有](#)
- [Ceph-RBDブロックストレージボリューム](#)
- [GlusterFSボリュームを使用した共有ストレージ](#)
- [GlusterFSを使用した動的プロビジョニングストレージ](#)
- [特権付きPodへのPVのマウント](#)
- [GlusterFSストレージによるDockerレジストリーのサポート](#)
- [ラベルによる永続ボリュームのバインド](#)
- [動的プロビジョニングでのStorageClassの使用](#)
- [既存レガシーストレージでのStorageClassの使用](#)
- [Azure Blobストレージでの統合コンテナイメージレジストリーの設定](#)

### 28.2. 2つのPERSISTENT VOLUME CLAIM (永続ボリューム要求) 間でのNFSマウントの共有

#### 28.2.1. 概要

以下のユースケースでは、クラスター管理者が2つの別々のコンテナで共有ストレージを利用しようとしている場合に、その解決策を設定する方法について説明します。ここではNFSの使用例を取り上げていますが、GlusterFSなど他の共有ストレージタイプにも簡単に応用できます。また、この例では共有ストレージに関連したPodのセキュリティの設定についても説明します。

[NFSを使用した永続ストレージ](#) では、永続ボリューム (PV)、Persistent Volume Claim (永続ボリューム要求、PVC)、永続ストレージとしてのNFSの使用について説明しています。このトピックでは既存のNFSクラスターとOpenShift Container Platform 永続ストアの使用について詳しく説明しますが、既存のNFSサーバーおよびエクスポートがOpenShift Container Platform インフラストラクチャーに存在することを前提とします。



#### 注記

**oc** コマンドはすべて OpenShift Container Platform のマスターホストで実行されます。

#### 28.2.2. 永続ボリュームの作成

PVオブジェクトをOpenShift Container Platformで作成する前に、永続ボリューム (PV) ファイルを以下のように定義します。

## 例28.1 NFS を使用した永続ボリュームオブジェクトの定義

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv ①
spec:
  capacity:
    storage: 1Gi ②
  accessModes:
    - ReadWriteMany ③
  persistentVolumeReclaimPolicy: Retain ④
  nfs: ⑤
    path: /opt/nfs ⑥
    server: nfs.f22 ⑦
    readOnly: false

```

- ① PV の名前。Pod 定義で参照されたり、各種の **oc** ボリュームコマンドで表示されたりします。
- ② このボリュームに割り当てられるストレージの量。
- ③ **accessModes** は、PV と PVC を一致させるためのラベルとして使用します。現時点で、これらはいずれの形態のアクセス制御も定義しません。
- ④ ボリューム回収ポリシー **Retain** は、ボリュームにアクセスする Pod が終了した後もボリュームが維持されることを示します。
- ⑤ 使用するボリュームタイプを定義します。この例では **NFS** プラグインを定義しています。
- ⑥ NFS マウントパスです。
- ⑦ NFS サーバーです。IP アドレスで指定することもできます。

PV の定義を `nfs-pv.yaml` などのファイルに保存し、以下のように永続ボリュームを作成します。

```

# oc create -f nfs-pv.yaml
persistentvolume "nfs-pv" created

```

永続ボリュームが作成されたことを確認します。

```

# oc get pv
NAME      LABELS  CAPACITY  ACCESSMODES  STATUS  CLAIM  REASON  AGE
nfs-pv   <none>  1Gi      RWX          Available  37s

```

### 28.2.3. Persistent Volume Claim (永続ボリューム要求、PVC) の作成

Persistent Volume Claim (永続ボリューム要求、PVC) では、必要なアクセスモードとストレージ容量を指定します。現時点では、これら 2 つの属性のみに基づいて PVC が 1 つの PV にバインドされます。PV が PVC にバインドされると、その PV は基本的に当該 PVC のプロジェクトに結び付けられ、別の

PVC にバインドすることはできません。PV と PVC には1対1のマッピングが存在します。ただし、同じプロジェクト内の複数の Pod が同じ PVC を使用することは可能です。以下の例ではそのユースケースを取り上げています。

### 例28.2 PVC オブジェクト定義

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc ①
spec:
  accessModes:
    - ReadWriteMany ②
  resources:
    requests:
      storage: 1Gi ③
```

- ① この要求名は、**volumes** セクションで Pod によって参照されます。
- ② PV についての先の説明にあるように、**accessModes** はアクセスを実施するものではなく、PV と PVC を一致させるためのラベルとして機能します。
- ③ この要求は、**1Gi** 以上の容量がある PV を検索します。

PVC の定義を `nfs-pvc.yaml` などのファイルに保存し、以下のように PVC を作成します。

```
# oc create -f nfs-pvc.yaml
persistentvolumeclaim "nfs-pvc" created
```

PVC が作成されていて、予想される PV にバインドされていることを確認します。

```
# oc get pvc
NAME          LABELS   STATUS   VOLUME   CAPACITY   ACCESSMODES   AGE
nfs-pvc      <none>  Bound   nfs-pv   1Gi        RWX           24s
                ①
```

- ① 要求の `nfs-pvc` が `nfs-pv` PV にバインドされています。

#### 28.2.4. NFS ボリュームアクセスの確保

NFS サーバー内のノードへのアクセスが必要です。このノードで、以下のように NFS エクスポートのマウントを確認します。

```
[root@nfs nfs]# ls -lZ /opt/nfs/
total 8
-rw-r--r--. 1 root 100003 system_u:object_r:usr_t:s0 10 Oct 12 23:27 test2b
                ①
```

- ① 所有者の ID は 0 です。

- 2 グループの ID は 100003 です。

NFS マウントにアクセスするためには、コンテナが SELinux ラベルを一致する必要があり、UID を 0 にして実行するか、または補助グループ範囲内の 100003 に指定して実行します。NFS マウントのグループ (後の Pod 定義で定義される) に一致させることでボリュームにアクセスできるようにします。

デフォルトでは、SELinux では Pod からリモートの NFS サーバーへの書き込みは許可されません。SELinux を各ノードで有効にした状態で NFS ボリュームへの書き込みを有効にするには、以下を実行します。

```
# setsebool -P virt_use_nfs on
```

### 28.2.5. Pod の作成

Pod 定義ファイルまたはテンプレートファイルを使用して Pod を定義できます。以下は、1つのコンテナを作成して NFS ボリュームを読み書きアクセス用にマウントする Pod 仕様です。

#### 例28.3 Pod オブジェクトの定義

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-nfs-pod 1
  labels:
    name: hello-openshift-nfs-pod
spec:
  containers:
    - name: hello-openshift-nfs-pod
      image: openshift/hello-openshift 2
      ports:
        - name: web
          containerPort: 80
      volumeMounts:
        - name: nfsvol 3
          mountPath: /usr/share/nginx/html 4
  securityContext:
    supplementalGroups: [100003] 5
    privileged: false
  volumes:
    - name: nfsvol
      persistentVolumeClaim:
        claimName: nfs-pvc 6
```

- 1 **oc get pod** によって表示されるこの Pod の名前。
- 2 この Pod が実行するイメージ。
- 3 ボリュームの名前。この名前は **containers** セクションと **volumes** セクションの両方で同じにする必要があります。
- 4 コンテナで表示されるマウントパス。
- 5 コンテナに割り当てるグループ ID。

## 6 先の手順で作成した PVC。

Pod 定義を `nfs.yaml` などのファイルに保存し、以下のように Pod を作成します。

```
# oc create -f nfs.yaml
pod "hello-openshift-nfs-pod" created
```

Pod が作成されていることを確認します。

```
# oc get pods
NAME                READY   STATUS    RESTARTS   AGE
hello-openshift-nfs-pod  1/1    Running   0          4s
```

詳細は `oc describe pod` コマンドで以下のように表示されます。

```
[root@ose70 nfs]# oc describe pod hello-openshift-nfs-pod
Name: hello-openshift-nfs-pod
Namespace: default 1
Image(s): fedora/S3
Node: ose70.rh7/192.168.234.148 2
Start Time: Mon, 21 Mar 2016 09:59:47 -0400
Labels: name=hello-openshift-nfs-pod
Status: Running
Reason:
Message:
IP: 10.1.0.4
Replication Controllers: <none>
Containers:
  hello-openshift-nfs-pod:
    Container ID:
docker://a3292104d6c28d9cf49f440b2967a0fc5583540fc3b062db598557b93893bc6f
    Image: fedora/S3
    Image ID:
docker://403d268c640894cbd76d84a1de3995d2549a93af51c8e16e89842e4c3ed6a00a
    QoS Tier:
      cpu: BestEffort
      memory: BestEffort
    State: Running
      Started: Mon, 21 Mar 2016 09:59:49 -0400
    Ready: True
    Restart Count: 0
    Environment Variables:
Conditions:
  Type Status
  Ready True
Volumes:
  nfsvol:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName: nfs-pvc 3
    ReadOnly: false
  default-token-a06zb:
    Type: Secret (a secret that should populate this volume)
```

SecretName: default-token-a06zb

Events: **4**

FirstSeen	LastSeen	Count	From	SubobjectPath	Reason	Message
4m	4m	1	{scheduler }		Scheduled	Successfully assigned hello-openshift-nfs-pod to ose70.rh7
4m	4m	1	{kubelet ose70.rh7}		Pulled	Container image "openshift3/ose-pod:v3.1.0.4" already present on machine
4m	4m	1	{kubelet ose70.rh7}		Created	Created with docker id 866a37108041
4m	4m	1	{kubelet ose70.rh7}		Started	Started with docker id 866a37108041
4m	4m	1	{kubelet ose70.rh7}	spec.containers{hello-openshift-nfs-pod}	Pulled	Container image "fedora/S3" already present on machine
4m	4m	1	{kubelet ose70.rh7}	spec.containers{hello-openshift-nfs-pod}	Created	Created with docker id a3292104d6c2
4m	4m	1	{kubelet ose70.rh7}	spec.containers{hello-openshift-nfs-pod}	Started	Started with docker id a3292104d6c2

- 1** プロジェクト (namespace) 名。
- 2** Pod を実行している OpenShift Container Platform ノードの IP アドレス。
- 3** Pod で使用される PVC 名。
- 4** Pod の起動と NFS ボリュームのマウントをもたらすイベントの一覧。ボリュームをマウントできない場合、コンテナは正常に起動しません。

**oc get pod <name> -o yaml** コマンドでは、Pod の承認に使用される SCC や Pod のユーザー ID とグループ ID、SELinux ラベルなどの内部情報がさらに表示されます。

```
[root@ose70 nfs]# oc get pod hello-openshift-nfs-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    openshift.io/scc: restricted 1
  creationTimestamp: 2016-03-21T13:59:47Z
  labels:
    name: hello-openshift-nfs-pod
    name: hello-openshift-nfs-pod
  namespace: default 2
  resourceVersion: "2814411"
  selflink: /api/v1/namespaces/default/pods/hello-openshift-nfs-pod
  uid: 2c22d2ea-ef6d-11e5-adc7-000c2900f1e3
spec:
  containers:
  - image: fedora/S3
    imagePullPolicy: IfNotPresent
    name: hello-openshift-nfs-pod
  ports:
  - containerPort: 80
    name: web
    protocol: TCP
```

```
resources: {}
securityContext:
  privileged: false
terminationMessagePath: /dev/termination-log
volumeMounts:
- mountPath: /usr/share/S3/html
  name: nfsvol
- mountPath: /var/run/secrets/kubernetes.io/serviceaccount
  name: default-token-a06zb
  readOnly: true
dnsPolicy: ClusterFirst
host: ose70.rh7
imagePullSecrets:
- name: default-dockercfg-xvdew
nodeName: ose70.rh7
restartPolicy: Always
securityContext:
  supplementalGroups:
  - 100003 ❸
serviceAccount: default
serviceName: default
terminationGracePeriodSeconds: 30
volumes:
- name: nfsvol
  persistentVolumeClaim:
    claimName: nfs-pvc ❹
- name: default-token-a06zb
  secret:
    secretName: default-token-a06zb
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: 2016-03-21T13:59:49Z
    status: "True"
    type: Ready
  containerStatuses:
  - containerID: docker://a3292104d6c28d9cf49f440b2967a0fc5583540fc3b062db598557b93893bc6f
    image: fedora/S3
    imageID: docker://403d268c640894cbd76d84a1de3995d2549a93af51c8e16e89842e4c3ed6a00a
    lastState: {}
    name: hello-openshift-nfs-pod
    ready: true
    restartCount: 0
    state:
      running:
        startedAt: 2016-03-21T13:59:49Z
  hostIP: 192.168.234.148
  phase: Running
  podIP: 10.1.0.4
  startTime: 2016-03-21T13:59:47Z
```

- ❶ Pod が使用する SCC。
- ❷ プロジェクト (namespace) 名。
- ❸ Pod の補助グループ ID (すべてのコンテナ)。

- 4 Pod で使用される PVC 名。

### 28.2.6. 同じ PVC を参照する追加 Pod の作成

以下の Pod 定義 (同じ namespace で作成されている) では別のコンテナを使用しています。ただし、以下の volumes セクションで要求名を指定することで、同じバックストレージを使用できます。

#### 例28.4 Pod オブジェクトの定義

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox-nfs-pod 1
  labels:
    name: busybox-nfs-pod
spec:
  containers:
  - name: busybox-nfs-pod
    image: busybox 2
    command: ["sleep", "60000"]
    volumeMounts:
    - name: nfsvol-2 3
      mountPath: /usr/share/busybox 4
      readOnly: false
  securityContext:
    supplementalGroups: [100003] 5
    privileged: false
  volumes:
  - name: nfsvol-2
    persistentVolumeClaim:
      claimName: nfs-pvc 6
```

- 1 `oc get pod` によって表示されるこの Pod の名前。
- 2 この Pod が実行するイメージ。
- 3 ボリュームの名前。この名前は **containers** セクションと **volumes** セクションの両方で同じにする必要があります。
- 4 コンテナで表示されるマウントパス。
- 5 コンテナに割り当てるグループ ID。
- 6 先に作成されており、別のコンテナでも使用されている PVC。

Pod 定義を `nfs-2.yaml` などのファイルに保存し、以下のように Pod を作成します。

```
# oc create -f nfs-2.yaml
pod "busybox-nfs-pod" created
```

Pod が作成されていることを確認します。

```
# oc get pods
NAME          READY  STATUS   RESTARTS  AGE
busybox-nfs-pod  1/1    Running  0         3s
```

詳細は **oc describe pod** コマンドで以下のように表示されます。

```
[root@ose70 nfs]# oc describe pod busybox-nfs-pod
Name: busybox-nfs-pod
Namespace: default
Image(s): busybox
Node: ose70.rh7/192.168.234.148
Start Time: Mon, 21 Mar 2016 10:19:46 -0400
Labels: name=busybox-nfs-pod
Status: Running
Reason:
Message:
IP: 10.1.0.5
Replication Controllers: <none>
Containers:
  busybox-nfs-pod:
    Container ID:
docker://346d432e5a4824ebf5a47fceb4247e0568ecc64eadcc160e9bab481aecfb0594
    Image: busybox
    Image ID: docker://17583c7dd0dae6244203b8029733bdb7d17fccbb2b5d93e2b24cf48b8bfd06e2
    QoS Tier:
      cpu: BestEffort
      memory: BestEffort
    State: Running
      Started: Mon, 21 Mar 2016 10:19:48 -0400
      Ready: True
      Restart Count: 0
    Environment Variables:
Conditions:
  Type Status
  Ready True
Volumes:
  nfsvol-2:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName: nfs-pvc
    ReadOnly: false
  default-token-32d2z:
    Type: Secret (a secret that should populate this volume)
    SecretName: default-token-32d2z
Events:
  FirstSeen LastSeen Count From SubobjectPath Reason Message
  -----
  4m 4m 1 {scheduler } Scheduled Successfully assigned busybox-nfs-pod to ose70.rh7
  4m 4m 1 {kubelet ose70.rh7} implicitly required container POD Pulled Container image
"openshift3/ose-pod:v3.1.0.4" already present on machine
  4m 4m 1 {kubelet ose70.rh7} implicitly required container POD Created Created with docker id
249b7d7519b1
  4m 4m 1 {kubelet ose70.rh7} implicitly required container POD Started Started with docker id
249b7d7519b1
  4m 4m 1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod} Pulled Container image "busybox"
already present on machine
```

```
4m 4m 1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod} Created Created with docker id
346d432e5a48
4m 4m 1 {kubelet ose70.rh7} spec.containers{busybox-nfs-pod} Started Started with docker id
346d432e5a48
```

ここから分かるように、いずれのコンテナでも、バックエンドの同じ NFS マウントに割り当てられた同じストレージ要求を使用しています。

## 28.3. CEPH RBD を使用した詳細例

### 28.3.1. 概要

このトピックでは、既存の Ceph クラスターを OpenShift Container Platform の永続ストアとして使用する詳細な例を紹介します。ここでは、作業用の Ceph クラスターがすでに設定されていることを前提とします。まだ設定されていない場合は、[Red Hat Ceph Storage の概要](#) について参照してください。

[Ceph RADOS ブロックデバイスを使用した永続ストレージ](#) では、永続ボリューム (PV)、Persistent Volume Claim (永続ボリューム要求、PVC)、永続ストレージとしての Ceph RBD の使用について説明しています。



#### 注記

**oc ...** コマンドはすべて OpenShift Container Platform のマスターホストで実行されません。

### 28.3.2. ceph-common パッケージのインストール

**ceph-common** ライブラリーは、**すべてのスケジュール可能な** OpenShift Container Platform ノードにインストールする必要があります。



#### 注記

OpenShift Container Platform のオールインワンホストは、Pod のワークロードを実行するために使用されることは多くありません。したがって、これはスケジュール可能なノードに含まれません。

```
# yum install -y ceph-common
```

### 28.3.3. Ceph シークレットの作成

**ceph auth get-key** コマンドを Ceph の **MON** ノードで実行すると、**client.admin** ユーザーのキー値が以下のように表示されます。

#### 例28.5 Ceph のシークレットの定義

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
data:
  key: QVFBOFF2SIZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ== 1
type: kubernetes.io/rbd 2
```

- 1 この base64 キーは、Ceph の MON ノードの1つで `ceph auth get-key client.admin | base64` コマンドを使用して生成されたものであり、出力をコピーし、これをシークレットキーの値として貼り付けています。
- 2 この値は、Ceph RBD を動的プロビジョニングで機能させるために必要です。

シークレット定義を `ceph-secret.yaml` などのファイルに保存し、シークレットを作成します。

```
$ oc create -f ceph-secret.yaml
secret "ceph-secret" created
```

シークレットが作成されたことを確認します。

```
# oc get secret ceph-secret
NAME          TYPE          DATA   AGE
ceph-secret   kubernetes.io/rbd 1       23d
```

### 28.3.4. 永続ボリュームの作成

次に、OpenShift Container Platform で PV オブジェクトを作成する前に、永続ボリュームファイルを定義します。

#### 例28.6 Ceph RBD を使用した永続ボリュームオブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-pv 1
spec:
  capacity:
    storage: 2Gi 2
  accessModes:
    - ReadWriteOnce 3
  rbd: 4
    monitors: 5
      - 192.168.122.133:6789
    pool: rbd
    image: ceph-image
    user: admin
    secretRef:
      name: ceph-secret 6
    fsType: ext4 7
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

- 1 PV の名前。Pod 定義で参照されたり、各種の `oc` ボリュームコマンドで表示されたりします。
- 2 このボリュームに割り当てられるストレージの量。

- 3 **accessModes** は、PV と PVC を一致させるためのラベルとして使用します。現時点で、これらはいずれの形態のアクセス制御も定義しません。ブロックストレージはすべて、単一ユーザーに対して定義されます (非共有ストレージ)。
- 4 使用するボリュームタイプを定義します。この例では **rbd** プラグインを定義しています。
- 5 Ceph モニターの IP アドレスとポートの配列です。
- 6 先に定義した Ceph のシークレットです。OpenShift Container Platform から Ceph サーバーへのセキュアな接続を作成するのに使用します。
- 7 Ceph RBD ブロックデバイスにマウントされるファイルシステムタイプです。

PV の定義を **ceph-pv.yaml** などのファイルに保存し、永続ボリュームを作成します。

```
# oc create -f ceph-pv.yaml
persistentvolume "ceph-pv" created
```

永続ボリュームが作成されたことを確認します。

```
# oc get pv
NAME          LABELS  CAPACITY  ACCESSMODES  STATUS  CLAIM  REASON  AGE
ceph-pv       <none>  2147483648  RWO          Available  2s
```

### 28.3.5. Persistent Volume Claim (永続ボリューム要求、PVC) の作成

Persistent Volume Claim (永続ボリューム要求、PVC) では、必要なアクセスモードとストレージ容量を指定します。現時点では、これら 2 つの属性のみに基づいて PVC が 1 つの PV にバインドされます。PV が PVC にバインドされると、その PV は基本的に当該 PVC のプロジェクトに結び付けられ、別の PVC にバインドすることはできません。PV と PVC には 1 対 1 のマッピングが存在します。ただし、同じプロジェクト内の複数の Pod が同じ PVC を使用することは可能です。

#### 例 28.7 PVC オブジェクト定義

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ceph-claim
spec:
  accessModes: ①
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi ②
```

- ① PV についての先の説明にあるように、**accessModes** はアクセスを実施するものではなく、PV と PVC を一致させるためのラベルとして機能します。
- ② この要求は 2Gi 以上の容量を提供する PV を探します。

PVC の定義を `ceph-claim.yaml` などのファイルに保存し、以下のように PVC を作成します。

```
# oc create -f ceph-claim.yaml
persistentvolumeclaim "ceph-claim" created

#and verify the PVC was created and bound to the expected PV:
# oc get pvc
NAME          LABELS   STATUS   VOLUME   CAPACITY   ACCESSMODES   AGE
ceph-claim    <none>  Bound    ceph-pv  1Gi        RWX           21s
```

1

1 要求が `ceph-pv` PV にバインドされています。

### 28.3.6. Pod の作成

Pod 定義ファイルまたはテンプレートファイルを使用して Pod を定義できます。以下は、1つのコンテナを作成し、Ceph RBD ボリュームを読み書きアクセス用にマウントする Pod 仕様です。

#### 例28.8 Pod オブジェクトの定義

```
apiVersion: v1
kind: Pod
metadata:
  name: ceph-pod1 1
spec:
  containers:
  - name: ceph-busybox
    image: busybox 2
    command: ["sleep", "60000"]
    volumeMounts:
    - name: ceph-vol1 3
      mountPath: /usr/share/busybox 4
      readOnly: false
  volumes:
  - name: ceph-vol1 5
    persistentVolumeClaim:
      claimName: ceph-claim 6
```

1 `oc get pod` によって表示されるこの Pod の名前。

2 この Pod が実行するイメージ。この例では、`busybox` に `sleep` の実行を指示しています。

3 5 ボリュームの名前。この名前は `containers` セクションと `volumes` セクションの両方で同じにする必要があります。

4 コンテナで表示されるマウントパス。

6 Ceph RBD クラスタにバインドされる PVC。

Pod 定義を `ceph-pod1.yaml` などのファイルに保存し、以下のように Pod を作成します。

```
# oc create -f ceph-pod1.yaml
pod "ceph-pod1" created

#verify pod was created
# oc get pod
NAME      READY   STATUS    RESTARTS   AGE
ceph-pod1 1/1     Running   0           2m
```

①

- ① しばらくすると、Pod が **Running** 状態になります。

### 28.3.7. グループ ID と所有者 ID の定義 (オプション)

Ceph RBD などのブロックストレージを使用する場合、物理ブロックストレージは Pod の**管理対象**になります。Pod で定義されたグループ ID は、コンテナ内の Ceph RBD マウントと実際のストレージ自体の**両方**のグループ ID になります。そのため、通常はグループ ID を Pod 仕様に定義する必要はありません。ただし、グループ ID が必要な場合は、以下の Pod 定義の例に示すように **fsGroup** を使用して定義することができます。

#### 例28.9 グループ ID の Pod 定義

```
...
spec:
  containers:
    - name:
      ...
      securityContext: ①
        fsGroup: 7777 ②
      ...
```

- ① **securityContext** は特定のコンテナの下位ではなく、この Pod レベルで定義します。
- ② Pod 内のコンテナはすべて同じ **fsGroup** ID になります。

### 28.3.8. ceph-user-secret をプロジェクトのデフォルトとして設定する方法

すべてのプロジェクトで永続ストレージを使用できるようにする場合は、デフォルトのプロジェクトテンプレートを修正する必要があります。デフォルトプロジェクトテンプレートの修正についての詳細を確認できます。詳細は、[デフォルトのプロジェクトテンプレートの修正](#) を参照してください。これをデフォルトのプロジェクトテンプレートに追加すると、プロジェクト作成の権限があるユーザーはすべて Ceph クラスターにアクセスできるようになります。

#### デフォルトプロジェクトの例

```
...
apiVersion: v1
kind: Template
metadata:
  creationTimestamp: null
  name: project-request
objects:
```

```

- apiVersion: v1
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT_DESCRIPTION}
      openshift.io/display-name: ${PROJECT_DISPLAYNAME}
      openshift.io/requester: ${PROJECT_REQUESTING_USER}
    creationTimestamp: null
    name: ${PROJECT_NAME}
  spec: {}
  status: {}
- apiVersion: v1
  kind: Secret
  metadata:
    name: ceph-user-secret
  data:
    key: yoursupersecretbase64keygoeshere ❶
  type:
    kubernetes.io/rbd
...

```

❶ base64 形式で Ceph のユーザーキーをここに配置します。

## 28.4. 動的プロビジョニングでの CEPH RBD の使用

### 28.4.1. 概要

このトピックでは、既存の Ceph クラスタを OpenShift Container Platform の永続ストレージとして使用する詳細な例を紹介します。ここでは、作業用の Ceph クラスタがすでに設定されていることを前提とします。まだ設定されていない場合は、[Red Hat Ceph Storage の概要](#) について参照してください。

[Ceph RADOS ブロックデバイスを使用した永続ストレージ](#) では、永続ボリューム (PV)、Persistent Volume Claim (永続ボリューム要求、PVC)、永続ストレージとしての Ceph RADOS ブロックデバイス (RBD) の使用方法について説明しています。



#### 注記

- OpenShift Container Platform のマスターホストで、全 **oc** コマンドを実行します。
- OpenShift Container Platform のオールインワンホストは、Pod のワークロードを実行するために使用されることは多くありません。したがって、これはスケジューラ可能なノードに含まれません。

### 28.4.2. 動的ボリューム用プールの作成

1. Install the latest ceph-common package:

```
yum install -y ceph-common
```



## 注記

**ceph-common** ライブラリーは、**all schedulable** OpenShift Container Platform ノードにインストールする必要があります。

2. 管理者または **MON** ノードによって、以下のように動的ボリューム用の新規プールが作成されます。

```
$ ceph osd pool create kube 1024
$ ceph auth get-or-create client.kube mon 'allow r, allow command "osd blacklist"' osd 'allow class-read object_prefix rbd_children, allow rwx pool=kube' -o ceph.client.kube.keyring
```



## 注記

RBD のデフォルトプールを使用することも可能ですが、このオプションは推奨されません。

### 28.4.3. 動的な永続ストレージでの既存の Ceph クラスターの使用

動的な永続ストレージに既存の Ceph クラスターを使用するには、以下を実行します。

1. client.admin 向けに base64 でエンコードされたキーを作成します。

```
$ ceph auth get client.admin
```

#### Ceph シークレット定義例

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
  namespace: kube-system
data:
  key: QVFBOFF2SIZheUJQRVJBQWgvS2cwT1laQUhPQno3akZwekxxdGc9PQ== 1
type: kubernetes.io/rbd 2
```

- 1** この base64 キーは、Ceph の MON ノードの1つで **ceph auth get-key client.admin | base64** コマンドを使用して生成されたものであり、出力をコピーし、これをシークレットキーの値として貼り付けています。
- 2** この値は、Ceph RBD を動的プロビジョニングで機能させるために必要です。

2. client.admin 用に Ceph シークレットを作成します。

```
$ oc create -f ceph-secret.yaml
secret "ceph-secret" created
```

3. シークレットが作成されたことを確認します。

```
$ oc get secret ceph-secret
NAME      TYPE          DATA   AGE
ceph-secret  kubernetes.io/rbd  1       5d
```

## 4. ストレージクラスを作成します。

```
$ oc create -f ceph-storageclass.yaml
storageclass "dynamic" created
```

## Ceph ストレージクラスの例

```
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: dynamic
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/rbd
parameters:
  monitors: 192.168.1.11:6789,192.168.1.12:6789,192.168.1.13:6789 ①
  adminId: admin ②
  adminSecretName: ceph-secret ③
  adminSecretNamespace: kube-system ④
  pool: kube ⑤
  userId: kube ⑥
  userSecretName: ceph-user-secret ⑦
```

- ① Ceph が監視する IP アドレスのコンマ区切りの一覧。この値は必須です。
- ② Ceph クライアント ID。デフォルトは **admin** です。
- ③ **adminId** のシークレット名。この値は必須です。設定するシークレットには **kubernetes.io/rbd** が含まれる必要があります。
- ④ **adminSecret** の namespace。デフォルトは **default** です。
- ⑤ Ceph RBD プール。デフォルトは **rbd** ですが、この値は推奨されません。
- ⑥ Ceph RBD イメージのマッピングに使用される Ceph クライアント ID。デフォルトは **adminId** のシークレット名と同じです。
- ⑦ Ceph RBD イメージをマッピングするための **userId** の Ceph シークレット名。PVC と同じ namespace に存在する必要があります。Ceph シークレットが新規プロジェクトのデフォルトとして設定されていない限り、このパラメーターの値を指定する必要があります。

## 5. ストレージクラスが作成されたことを確認します。

```
$ oc get storageclasses
NAME          TYPE
dynamic (default) kubernetes.io/rbd
```

## 6. PVC オブジェクト定義を作成します。

## PVC オブジェクト定義例

```
kind: PersistentVolumeClaim
```

```

apiVersion: v1
metadata:
  name: ceph-claim-dynamic
spec:
  accessModes: ❶
  - ReadWriteOnce
resources:
  requests:
    storage: 2Gi ❷

```

❶ **accessModes** はアクセス権としての効果はなく、代わりに PV と PVC を照合するラベルとして機能します。

❷ この要求は **2Gi** 以上の容量を提供する PV を探します。

7. PVC を作成します。

```

$ oc create -f ceph-pvc.yaml
persistentvolumeclaim "ceph-claim-dynamic" created

```

8. PVC が作成されていて、予想される PV にバインドされていることを確認します。

```

$ oc get pvc
NAME          STATUS VOLUME                                     CAPACITY ACCESSMODES AGE
ceph-claim   Bound  pvc-f548d663-3cac-11e7-9937-0024e8650c7a 2Gi      RWO          1m

```

9. Pod オブジェクト定義を以下のように作成します。

### Pod オブジェクトの定義例

```

apiVersion: v1
kind: Pod
metadata:
  name: ceph-pod1 ❶
spec:
  containers:
  - name: ceph-busybox
    image: busybox ❷
    command: ["sleep", "60000"]
    volumeMounts:
    - name: ceph-vol1 ❸
      mountPath: /usr/share/busybox ❹
      readOnly: false
  volumes:
  - name: ceph-vol1
    persistentVolumeClaim:
      claimName: ceph-claim-dynamic ❺

```

❶ **oc get pod** によって表示されるこの Pod の名前。

❷ この Pod が実行するイメージ。この例では、**busybox** は **sleep** に設定されています。

❸ ボリュームの名前。この名前は **containers** セクションと **volumes** セクションの両方で同じ名前を指定する必要があります。

しにする必要がありません。

- 4 コンテナでのマウントパス。
- 5 Ceph RBD クラスターにバインドされる PVC。

10. Pod を作成します。

```
$ oc create -f ceph-pod1.yaml
pod "ceph-pod1" created
```

11. Pod が作成されていることを確認します。

```
$ oc get pod
NAME      READY   STATUS    RESTARTS   AGE
ceph-pod1 1/1     Running   0           2m
```

しばらくすると、Pod のステータスが **Running** に変わります。

#### 28.4.4. ceph-user-secret をプロジェクトのデフォルトとして設定する方法

全プロジェクトで永続ストレージを使用できるようにするには、デフォルトのプロジェクトテンプレートを変更する必要があります。これをデフォルトのプロジェクトテンプレートに追加すると、プロジェクト作成の権限があるユーザーはすべて Ceph クラスターにアクセスできるようになります。詳細情報は、[デフォルトのプロジェクトテンプレート変更](#) を参照してください。

##### デフォルトプロジェクトの例

```
...
apiVersion: v1
kind: Template
metadata:
  creationTimestamp: null
  name: project-request
objects:
- apiVersion: v1
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT_DESCRIPTION}
      openshift.io/display-name: ${PROJECT_DISPLAYNAME}
      openshift.io/requester: ${PROJECT_REQUESTING_USER}
    creationTimestamp: null
    name: ${PROJECT_NAME}
  spec: {}
  status: {}
- apiVersion: v1
  kind: Secret
  metadata:
    name: ceph-user-secret
  data:
    key: QVFCbEV4OVpmaGJtQ0JBQW55d2Z0NHZtcS96cE42SW1JVUQvekE9PQ== 1
```

```
type:
  kubernetes.io/rbd
...
```

- 1 base64 形式で Ceph のユーザーキーをここに配置します。

## 28.5. GLUSTERFS を使用する詳細例

### 28.5.1. 概要

このトピックでは、既存の接続モード、独立モードまたはスタンドアロンの Red Hat Gluster Storage クラスターを OpenShift Container Platform の永続ストレージとして使用する詳細例を紹介します。ここでは作業用の Red Hat Gluster Storage クラスターがすでに設定されていることを前提とします。接続モードまたは独立モードのインストールについてのヘルプは、[Red Hat Gluster Storage を使用する永続ストレージ](#) を参照してください。スタンドアロンの Red Hat Gluster Storage の場合については、[Red Hat Gluster Storage Administration Guide](#) を参照してください。

GlusterFS ボリュームを動的にプロビジョニングする方法の詳細例については、[GlusterFS を動的プロビジョニングに使用する詳細例](#) を参照してください。



#### 注記

**oc** コマンドはすべて OpenShift Container Platform のマスターホストで実行されます。

### 28.5.2. 前提条件

GlusterFS ボリュームにアクセスするには、すべてのスケジュール可能なノードで **mount.glusterfs** コマンドを利用できる必要があります。RPM ベースのシステムの場合は、**glusterfs-fuse** パッケージがインストールされている必要があります。

```
# yum install glusterfs-fuse
```

このパッケージはすべての RHEL システムにインストールされています。ただし、サーバーが x86\_64 アーキテクチャーを使用する場合は Red Hat Gluster Storage の最新バージョンに更新することを推奨します。そのためには、以下の RPM リポジトリを有効にする必要があります。

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-server-rpms
```

**glusterfs-fuse** がノードにすでにインストールされている場合、最新バージョンがインストールされていることを確認します。

```
# yum update glusterfs-fuse
```

デフォルトでは、SELinux は Pod からリモート Red Hat Gluster Storage サーバーへの書き込みを許可しません。SELinux が有効な状態で Red Hat Gluster Storage ボリュームへの書き込みを有効にするには、GlusterFS を実行する各ノードで以下のコマンドを実行します。

```
$ sudo setsebool -P virt_sandbox_use_fusefs on 1
$ sudo setsebool -P virt_use_fusefs on
```

- 1 **-P** オプションを使用すると、再起動した後もブール値が永続化されます。



## 注記

**virt\_sandbox\_use\_fusefs** ブール値は、**docker-selinux** パッケージによって定義されます。このブール値が定義されていないというエラーが表示される場合は、このパッケージがインストールされていることを確認してください。



## 注記

Atomic Host を使用しており、Atomic Host をアップグレードすると、SELinux のブール値が消去されます。Atomic Host をアップグレードする場合には、これらのブール値を設定し直す必要があります。

### 28.5.3. 静的プロビジョニング

1. 静的プロビジョニングを有効にするには、最初に GlusterFS ボリュームを作成します。**gluster** コマンドラインインターフェイスの使用方法については、[Red Hat Gluster Storage Administration Guide](#)、**heketi-cli** を使用した方法については [heketi project site](#) のプロジェクトサイトを参照してください。この例では、ボリュームに **myVol1** という名前を付けます。
2. **gluster-endpoints.yaml** で以下のサービスとエンドポイントを定義します。

```
---
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster ①
spec:
  ports:
  - port: 1
---
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster ②
subsets:
  - addresses:
    - ip: 192.168.122.221 ③
    ports:
    - port: 1 ④
  - addresses:
    - ip: 192.168.122.222 ⑤
    ports:
    - port: 1 ⑥
  - addresses:
    - ip: 192.168.122.223 ⑦
    ports:
    - port: 1 ⑧
```

① ② これらの名前は一致している必要があります。

③ ⑤ ⑦ **ip** の値には、Red Hat Gluster Storage サーバーのホスト名ではなく、実際の IP アドレスを指定する必要があります。

④ ⑥ ⑧ ポート番号は無視されます。

3. OpenShift Container Platform マスターホストからサービスとエンドポイントを作成します。

```
$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created
```

4. サービスとエンドポイントが作成されたことを確認します。

```
$ oc get services
NAME                CLUSTER_IP      EXTERNAL_IP  PORT(S)  SELECTOR  AGE
glusterfs-cluster   172.30.205.34   <none>       1/TCP    <none>    44s

$ oc get endpoints
NAME                ENDPOINTS                                     AGE
docker-registry     10.1.0.3:5000                                  4h
glusterfs-cluster   192.168.122.221:1,192.168.122.222:1,192.168.122.223:1  11s
kubernetes          172.16.35.3:8443                               4d
```



### 注記

エンドポイントはプロジェクトごとに一意です。GlusterFS にアクセスする各プロジェクトには独自のエンドポイントが必要です。

5. ボリュームにアクセスするには、ボリューム上のファイルシステムにアクセスできるユーザー ID (UID) またはグループ ID (GID) でコンテナを実行する必要があります。この情報は以下の方法で取得できます。

```
$ mkdir -p /mnt/glusterfs/myVol1

$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1

$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0  myVol1 ① ②
```

① UID は 592 です。

② GID は 590 です。

6. **gluster-pv.yaml** で以下の PersistentVolume (PV) を定義します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume ①
  annotations:
    pv.beta.kubernetes.io/gid: "590" ②
spec:
  capacity:
    storage: 2Gi ③
  accessModes: ④
    - ReadWriteMany
  glusterfs:
```

```
endpoints: glusterfs-cluster ⑤
path: myVol1 ⑥
readOnly: false
persistentVolumeReclaimPolicy: Retain
```

- ① ボリュームの名前。
- ② GlusterFS ボリュームのルートの GID です。
- ③ このボリュームに割り当てられるストレージの量。
- ④ **accessModes** は、PV と PVC を一致させるためのラベルとして使用されます。現時点で、これらはいずれの形態のアクセス制御も定義しません。
- ⑤ 以前に作成されたエンドポイントリソースです。
- ⑥ アクセス対象の GlusterFS ボリュームです。

7. OpenShift Container Platform マスターホストから PV を作成します。

```
$ oc create -f gluster-pv.yaml
```

8. PV が作成されたことを確認します。

```
$ oc get pv
NAME          LABELS  CAPACITY  ACCESSMODES  STATUS  CLAIM
REASON  AGE
gluster-default-volume <none>  2147483648  RWX          Available           2s
```

9. **gluster-claim.yaml** で、新規 PV にバインドする PersistentVolumeClaim (PVC) を作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim ①
spec:
  accessModes:
  - ReadWriteMany ②
  resources:
    requests:
      storage: 1Gi ③
```

- ① この要求名は、**volumes** セクションで Pod によって参照されます。
- ② PV の **accessModes** に一致する必要があります。
- ③ この要求は、**1Gi** 以上の容量がある PV を検索します。

10. OpenShift Container Platform マスターホストから PVC を作成します。

```
$ oc create -f gluster-claim.yaml
```

11. PV と PVC がバインドされていることを確認します。

```
$ oc get pv
NAME          LABELS    CAPACITY  ACCESSMODES  STATUS   CLAIM          REASON  AGE
gluster-pv   <none>   1Gi      RWX          Available gluster-claim  37s

$ oc get pvc
NAME          LABELS    STATUS   VOLUME    CAPACITY  ACCESSMODES  AGE
gluster-claim <none>   Bound   gluster-pv 1Gi      RWX          24s
```



### 注記

PVC はプロジェクトごとに一意です。GlusterFS ボリュームにアクセスする各プロジェクトには独自の PVC が必要です。PV は単一のプロジェクトにバインドされないため、複数のプロジェクトにまたがる PVC が同じ PV を参照する場合があります。

## 28.5.4. ストレージの使用

この時点で、PVC にバインドされる GlusterFS ボリュームが動的に作成されています。そのため、この PVC を Pod で使用できるようになりました。

1. Pod オブジェクト定義を以下のように作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-pod
  labels:
    name: hello-openshift-pod
spec:
  containers:
  - name: hello-openshift-pod
    image: openshift/hello-openshift
    ports:
    - name: web
      containerPort: 80
    volumeMounts:
    - name: gluster-vol1
      mountPath: /usr/share/nginx/html
      readOnly: false
  volumes:
  - name: gluster-vol1
    persistentVolumeClaim:
      claimName: gluster1 ❶
```

- ❶ 先の手順で作成した PVC の名前。

2. OpenShift Container Platform マスターホストから、以下のように Pod を作成します。

```
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
```

3. Pod を表示します。イメージがまだ存在していない場合はダウンロードする必要があるために数分の時間がかかります。

```
# oc get pods -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP           NODE
hello-openshift-pod 1/1     Running  0          9m   10.38.0.0   node1
```

4. コンテナへの **oc exec** を実行し、**index.html** ファイルを Pod の **mountPath** 定義内に作成します。

```
$ oc exec -ti hello-openshift-pod /bin/sh
$ cd /usr/share/nginx/html
$ echo 'Hello OpenShift!!!' > index.html
$ ls
index.html
$ exit
```

5. ここで、Pod の URL に対して **curl** を実行します。

```
# curl http://10.38.0.0
Hello OpenShift!!!
```

6. Pod を削除してから再作成し、これが出現するまで待機します。

```
# oc delete pod hello-openshift-pod
pod "hello-openshift-pod" deleted
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
# oc get pods -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP           NODE
hello-openshift-pod 1/1     Running  0          9m   10.37.0.0   node1
```

7. もう一度 Pod に対して **curl** を実行します。データは前と同じになりますが、Pod の IP アドレスは変更されている可能性があることに注意してください。

```
# curl http://10.37.0.0
Hello OpenShift!!!
```

8. 以下の操作をいずれかのノードで実行して、**index.html** ファイルが GlusterFS ストレージに書き込まれていることを確認します。

```
$ mount | grep heketi
/dev/mapper/VolGroup00-LogVol00 on /var/lib/heketi type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_1e730a5462c352835055018e1874e578 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_1e730a5462c35283505
5018e1874e578 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,swidth=512,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_d8c06e606ff4cc29ccb9d018c73ee292 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c06e606ff4cc29ccb9d
018c73ee292 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,swidth=512,noquota)
```

```
$ cd
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c06e606ff4cc29ccb9d
018c73ee292/brick
$ ls
index.html
$ cat index.html
Hello OpenShift!!!
```

## 28.6. GLUSTERFS を動的プロビジョニングに使用する詳細例

### 28.6.1. 概要

このトピックでは、既存の接続モード、独立モードまたはスタンドアロンの Red Hat Gluster Storage クラスターを OpenShift Container Platform の動的永続ストレージとして使用する詳細例を紹介します。ここでは作業用の Red Hat Gluster Storage クラスターがすでに設定されていることを前提とします。接続モードまたは独立モードのインストールについてのヘルプは、[Red Hat Gluster Storage を使用する永続ストレージ](#) を参照してください。スタンドアロンの Red Hat Gluster Storage の場合については、[Red Hat Gluster Storage Administration Guide](#) を参照してください。



#### 注記

**oc** コマンドはすべて OpenShift Container Platform のマスターホストで実行されます。

### 28.6.2. 前提条件

GlusterFS ボリュームにアクセスするには、すべてのスケジュール可能なノードで **mount.glusterfs** コマンドを利用できる必要があります。RPM ベースのシステムの場合は、**glusterfs-fuse** パッケージがインストールされている必要があります。

```
# yum install glusterfs-fuse
```

このパッケージはすべての RHEL システムにインストールされています。ただし、サーバーが x86\_64 アーキテクチャーを使用する場合は Red Hat Gluster Storage の最新バージョンに更新することを推奨します。そのためには、以下の RPM リポジトリを有効にする必要があります。

```
# subscription-manager repos --enable=rh-gluster-3-client-for-rhel-7-server-rpms
```

**glusterfs-fuse** がノードにすでにインストールされている場合、最新バージョンがインストールされていることを確認します。

```
# yum update glusterfs-fuse
```

デフォルトでは、SELinux は Pod からリモート Red Hat Gluster Storage サーバーへの書き込みを許可しません。SELinux が有効な状態で Red Hat Gluster Storage ボリュームへの書き込みを有効にするには、GlusterFS を実行する各ノードで以下のコマンドを実行します。

```
$ sudo setsebool -P virt_sandbox_use_fusefs on 1
$ sudo setsebool -P virt_use_fusefs on
```

**1** **-P** オプションを使用すると、再起動した後もブール値が永続化されます。



## 注記

**virt\_sandbox\_use\_fusefs** ブール値は、**docker-selinux** パッケージによって定義されます。このブール値が定義されていないというエラーが表示される場合は、このパッケージがインストールされていることを確認してください。



## 注記

Atomic Host を使用しており、Atomic Host をアップグレードすると、SELinux のブール値が消去されます。Atomic Host をアップグレードする場合には、これらのブール値を設定し直す必要があります。

### 28.6.3. 動的プロビジョニング

- 動的プロビジョニングを有効にするには、最初に **StorageClass** オブジェクト定義を作成します。以下の定義は、OpenShift Container Platform でこの例を使用するために必要な最小要件に基づいています。その他のパラメーターと仕様定義については、[動的プロビジョニングとストレージクラスの作成](#) を参照してください。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: glusterfs
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://10.42.0.0:8080" ①
  restauthenabled: "false" ②
```

- ① heketi サーバーの URL です。
- ② この例では認証が有効ではないため、**false** に設定します。

- OpenShift Container Platform マスターホストから StorageClass を作成します。

```
# oc create -f gluster-storage-class.yaml
storageclass "glusterfs" created
```

- 新たに作成される StorageClass を使用して PVC を作成します。以下に例を示します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster1
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 30Gi
  storageClassName: glusterfs
```

- OpenShift Container Platform マスターホストから PVC を作成します。

```
# oc create -f glusterfs-dyn-pvc.yaml
persistentvolumeclaim "gluster1" created
```

- PVC を表示し、ボリュームが動的に作成され、PVC にバインドされていることを確認します。

```
# oc get pvc
NAME          STATUS  VOLUME                                     CAPACITY  ACCESSMODES
STORAGECLASS AGE
gluster1     Bound  pvc-78852230-d8e2-11e6-a3fa-0800279cf26f  30Gi      RWX
glusterfs    42s
```

#### 28.6.4. ストレージの使用

この時点で、PVC にバインドされる GlusterFS ボリュームが動的に作成されています。そのため、この PVC を Pod で使用できるようになりました。

- Pod オブジェクト定義を以下のように作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-openshift-pod
  labels:
    name: hello-openshift-pod
spec:
  containers:
  - name: hello-openshift-pod
    image: openshift/hello-openshift
    ports:
    - name: web
      containerPort: 80
    volumeMounts:
    - name: gluster-vol1
      mountPath: /usr/share/nginx/html
      readOnly: false
  volumes:
  - name: gluster-vol1
    persistentVolumeClaim:
      claimName: gluster1 ①
```

- ① 先の手順で作成した PVC の名前。

- OpenShift Container Platform マスターホストから、以下のように Pod を作成します。

```
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
```

- Pod を表示します。イメージがまだ存在していない場合はダウンロードする必要があるために数分の時間がかかります。

```
# oc get pods -o wide
NAME          READY  STATUS   RESTARTS  AGE  IP          NODE
hello-openshift-pod  1/1    Running  0          9m   10.38.0.0   node1
```

4. コンテナへの **oc exec** を実行し、**index.html** ファイルを Pod の **mountPath** 定義内に作成します。

```
$ oc exec -ti hello-openshift-pod /bin/sh
$ cd /usr/share/nginx/html
$ echo 'Hello OpenShift!!!' > index.html
$ ls
index.html
$ exit
```

5. ここで、Pod の URL に対して **curl** を実行します。

```
# curl http://10.38.0.0
Hello OpenShift!!!
```

6. Pod を削除してから再作成し、これが出現するまで待機します。

```
# oc delete pod hello-openshift-pod
pod "hello-openshift-pod" deleted
# oc create -f hello-openshift-pod.yaml
pod "hello-openshift-pod" created
# oc get pods -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP           NODE
hello-openshift-pod 1/1     Running   0          9m    10.37.0.0    node1
```

7. もう一度 Pod に対して **curl** を実行します。データは前と同じになりますが、Pod の IP アドレスは変更されている可能性があることに注意してください。

```
# curl http://10.37.0.0
Hello OpenShift!!!
```

8. 以下の操作をいずれかのノードで実行して、**index.html** ファイルが GlusterFS ストレージに書き込まれていることを確認します。

```
$ mount | grep heketi
/dev/mapper/VolGroup00-LogVol00 on /var/lib/heketi type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_1e730a5462c352835055018e1874e578 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_1e730a5462c35283505
5018e1874e578 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,swidth=512,noquota)
/dev/mapper/vg_f92e09091f6b20ab12b02a2513e4ed90-
brick_d8c06e606ff4cc29ccb9d018c73ee292 on
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c06e606ff4cc29ccb9d
018c73ee292 type xfs
(rw,noatime,seclabel,nouuid,attr2,inode64,logbsize=256k,sunit=512,swidth=512,noquota)

$ cd
/var/lib/heketi/mounts/vg_f92e09091f6b20ab12b02a2513e4ed90/brick_d8c06e606ff4cc29ccb9d
018c73ee292/brick
$ ls
```

```
index.html
$ cat index.html
Hello OpenShift!!!
```

## 28.7. 特権付き POD へのボリュームのマウント

### 28.7.1. 概要

永続ボリュームは、**特権付き** SCC (Security Context Constraint) が割り当てられた Pod にマウントすることができます。



#### 注記

このトピックでは、特権付き Pod へのボリュームのマウントのユースケースの例として GlusterFS を使用していますが、これはいずれの [サポート対象ストレージプラグイン](#) を使用するケースにも適用できます。

### 28.7.2. 前提条件

- 既存の Gluster ボリューム。
- すべてのホストにインストールされている `glusterfs-fuse`。
- GlusterFS の定義:
  - [エンドポイントとサービス](#): `gluster-endpoints-service.yaml` および `gluster-endpoints.yaml`
  - [永続ボリューム](#): `gluster-pv.yaml`
  - [Persistent Volume Claim \(永続ボリューム要求\)](#): `gluster-pvc.yaml`
  - [特権付き Pod](#): `gluster-S3-pod.yaml`
- `cluster-admin` ロールのバインディングが割り当てられたユーザー。このガイドでは、このユーザーを **admin** と呼んでいます。

### 28.7.3. 永続ボリュームの作成

`PersistentVolume` を作成すると、プロジェクトに関係なく、ユーザーがストレージにアクセスできるようになります。

1. admin として、サービス、エンドポイントオブジェクトおよび永続ボリュームを作成します。

```
$ oc create -f gluster-endpoints-service.yaml
$ oc create -f gluster-endpoints.yaml
$ oc create -f gluster-pv.yaml
```

2. オブジェクトが作成されたことを以下のように確認します。

```
$ oc get svc
NAME          CLUSTER_IP    EXTERNAL_IP  PORT(S)  SELECTOR  AGE
gluster-cluster  172.30.151.58 <none>      1/TCP    <none>    24s
```

```
$ oc get ep
NAME                ENDPOINTS                AGE
gluster-cluster    192.168.59.102:1,192.168.59.103:1    2m
```

```
$ oc get pv
NAME                LABELS    CAPACITY    ACCESSMODES    STATUS    CLAIM
REASON    AGE
gluster-default-volume    <none>    2Gi        RWX            Available    2d
```

#### 28.7.4. 通常ユーザーの作成

**通常ユーザー** を以下のように **特権付き SCC** (または SCC へのアクセスのあるグループ) に追加すると、当該ユーザーは**特権付き Pod** を実行できるようになります。

1. admin として、ユーザーを SCC に追加します。

```
$ oc adm policy add-scc-to-user privileged <username>
```

2. 通常ユーザーとしてログインします。

```
$ oc login -u <username> -p <password>
```

3. 次に、新規プロジェクトを作成します。

```
$ oc new-project <project_name>
```

#### 28.7.5. Persistent Volume Claim (永続ボリューム要求、PVC) の作成

1. 通常ユーザーとして、ボリュームにアクセスするための **PersistentVolumeClaim** を作成します。

```
$ oc create -f gluster-pvc.yaml -n <project_name>
```

2. 要求にアクセスするための Pod を定義します。

##### 例28.10 Pod 定義

```
apiVersion: v1
id: gluster-S3-pvc
kind: Pod
metadata:
  name: gluster-nginx-priv
spec:
  containers:
    - name: gluster-nginx-priv
      image: fedora/nginx
      volumeMounts:
        - mountPath: /mnt/gluster 1
          name: gluster-volume-claim
      securityContext:
        privileged: true
  volumes:
```

```
- name: gluster-volume-claim
  persistentVolumeClaim:
    claimName: gluster-claim ❷
```

- ❶ Pod 内のボリュームマウント。
- ❷ `gluster-claim` には `PersistentVolume` の名前を反映させる必要があります。

3. Pod を作成するとマウントディレクトリーが作成され、ボリュームがそのマウントポイントに割り当てられます。

通常ユーザーとして、以下のように定義から Pod を作成します。

```
$ oc create -f gluster-S3-pod.yaml
```

4. Pod が正常に作成されたことを確認します。

```
$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
gluster-S3-pod 1/1     Running   0           36m
```

Pod が作成されるまでに数分の時間がかかることがあります。

## 28.7.6. 設定の検証

### 28.7.6.1. Pod の SCC の確認

1. Pod 設定をエクスポートします。

```
$ oc get -o yaml --export pod <pod_name>
```

2. 出力を確認します。`openshift.io/scc` の値が `privileged` であることを確認します。

#### 例28.11 スニペットのエクスポート

```
metadata:
  annotations:
    openshift.io/scc: privileged
```

### 28.7.6.2. マウントの検証

1. Pod にアクセスし、ボリュームがマウントされていることを確認します。

```
$ oc rsh <pod_name>
[root@gluster-S3-pvc /]# mount
```

2. Gluster ボリュームの出力結果を確認します。

#### 例28.12 ボリュームマウント

```
192.168.59.102:gv0 on /mnt/gluster type fuse.gluster
(rw,relatime,user_id=0,group_id=0,default_permissions,allow_other,max_read=131072)
```

## 28.8. マウントの伝播

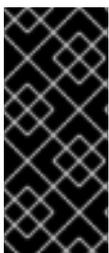
### 28.8.1. 概要

マウントの伝播により、コンテナでマウントされたボリュームを、同一 Pod の他のコンテナや同一ノード上の他の Pod にも共有できます。

### 28.8.2. 値

ボリュームのマウント伝播は、**Container.volumeMounts** の **mountPropagation** フィールドで制御します。その値は次のとおりです。

- **None:** このボリュームマウントでは、ホストによりボリュームまたはそのサブディレクトリーにマウントされている後続のマウントは受信されません。同様に、コンテナで作成されたマウントは、ホスト上では表示されません。これはデフォルトのモードであり、Linux カーネルの **プライベート** マウント伝播と同じです。
- **HostToContainer:** このボリュームマウントでは、ボリュームまたはそのサブディレクトリーのいずれかにマウントされている後続のマウントがすべて受信されます。つまり、ホストがボリュームマウント内に何かをマウントしていると、コンテナはそこにマウントが存在することを認識します。このモードは、Linux カーネルの **rslave** マウント伝播と同等です。
- **Bidirectional:** このボリュームマウントは **HostToContainer** マウントと同じように動作します。さらに、コンテナで作成されるすべてのボリュームマウントは、ホストおよび同じボリュームを使用する全 Pod のコンテナすべてに伝播されます。このモードの一般的なユースケースは、FlexVolume や CSI ドライバーまたは **hostPath** ボリュームを使用してホストにマウントする必要のある Pod です。このモードは、Linux カーネルの **rshared** マウント伝播と同等です。



#### 重要

**Bidirectional** マウント伝播はリスクを孕む可能性があります。ホストのオペレーティングシステムを破損する可能性があるため、特権コンテナでのみ使用できます。Linux カーネルの動作について理解しておくことが強く推奨されます。さらに、Pod のコンテナで作成されるボリュームマウントは、終了時にコンテナによって破棄されるか、またはアンマウントされる必要があります。

### 28.8.3. 設定

マウント共有を Docker で正しく設定しておかないと、マウント伝播が CoreOS、Red Hat Enterprise Linux/Centos、Ubuntu などの一部のデプロイメントで適切に機能しない可能性があります。

#### 手順

1. Docker の `systemd` サービスファイルを編集します。以下のように **MountFlags** を設定します。

```
MountFlags=shared
```

または、**MountFlags=slave** (存在する場合) を削除します。

2. Docker デーモンを再起動します。

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

## 28.9. 統合 OPENSIFT CONTAINER レジストリーから GLUSTERFS への切り替え

### 28.9.1. 概要

このトピックでは、GlusterFS ボリュームを統合 OpenShift Container レジストリーに割り当てる方法を説明します。この操作は、接続モード、独立モード、またはスタンドアロンの Red Hat Gluster Storage のいずれかで実行できます。ここでは、レジストリーがすでに起動されていて、ボリュームが作成済みであることを前提とします。

### 28.9.2. 前提条件

- ストレージを設定せずにデプロイされている既存の [レジストリー](#)。
- 既存の GlusterFS ボリューム。
- すべてのスケジュール可能なノードにインストールされている **glusterfs-fuse**。
- **cluster-admin** ロールのバインディングが割り当てられたユーザー。
  - このガイドでは、このユーザーを **admin** と呼んでいます。



#### 注記

**oc** コマンドはすべてマスターノードで **admin** ユーザーとして実行されます。

### 28.9.3. GlusterFS PersistentVolumeClaim の手動プロビジョニング

1. 静的プロビジョニングを有効にするには、最初に GlusterFS ボリュームを作成します。**gluster** コマンドラインインターフェイスの使用方法については、[Red Hat Gluster Storage Administration Guide](#)、**heketi-cli** を使用した方法については [heketi project site](#) のプロジェクトサイトを参照してください。この例では、ボリュームに **myVol1** という名前を付けます。
2. **gluster-endpoints.yaml** で以下のサービスとエンドポイントを定義します。

```
---
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster 1
spec:
  ports:
  - port: 1
---
apiVersion: v1
kind: Endpoints
```

```

metadata:
  name: glusterfs-cluster ❷
subsets:
  - addresses:
    - ip: 192.168.122.221 ❸
    ports:
    - port: 1 ❹
  - addresses:
    - ip: 192.168.122.222 ❺
    ports:
    - port: 1 ❻
  - addresses:
    - ip: 192.168.122.223 ❼
    ports:
    - port: 1 ❽

```

❶ ❷ これらの名前は一致している必要があります。

❸ ❺ ❼ ip の値には、Red Hat Gluster Storage サーバーのホスト名ではなく、実際の IP アドレスを指定する必要があります。

❹ ❻ ❽ ポート番号は無視されます。

- OpenShift Container Platform マスターホストからサービスとエンドポイントを作成します。

```

$ oc create -f gluster-endpoints.yaml
service "glusterfs-cluster" created
endpoints "glusterfs-cluster" created

```

- サービスとエンドポイントが作成されたことを確認します。

```

$ oc get services
NAME                CLUSTER_IP      EXTERNAL_IP  PORT(S)  SELECTOR  AGE
glusterfs-cluster  172.30.205.34   <none>       1/TCP    <none>    44s

$ oc get endpoints
NAME                ENDPOINTS                                     AGE
docker-registry    10.1.0.3:5000                                  4h
glusterfs-cluster  192.168.122.221:1,192.168.122.222:1,192.168.122.223:1  11s
kubernetes          172.16.35.3:8443                                4d

```



### 注記

エンドポイントはプロジェクトごとに一意です。GlusterFS にアクセスする各プロジェクトには独自のエンドポイントが必要です。

- ボリュームにアクセスするには、ボリューム上のファイルシステムにアクセスできるユーザー ID (UID) またはグループ ID (GID) でコンテナを実行する必要があります。この情報は以下の方法で取得できます。

```

$ mkdir -p /mnt/glusterfs/myVol1

```

```
$ mount -t glusterfs 192.168.122.221:/myVol1 /mnt/glusterfs/myVol1
```

```
$ ls -lnZ /mnt/glusterfs/
drwxrwx---. 592 590 system_u:object_r:fusefs_t:s0 myVol1 ① ②
```

① UID は 592 です。

② GID は 590 です。

6. **gluster-pv.yaml** で以下の PersistentVolume (PV) を定義します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume ①
  annotations:
    pv.beta.kubernetes.io/gid: "590" ②
spec:
  capacity:
    storage: 2Gi ③
  accessModes: ④
  - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster ⑤
    path: myVol1 ⑥
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

① ボリュームの名前。

② GlusterFS ボリュームのルート of GID です。

③ このボリュームに割り当てられるストレージの量。

④ **accessModes** は、PV と PVC を一致させるためのラベルとして使用されます。現時点で、これらはいずれの形態のアクセス制御も定義しません。

⑤ 以前に作成されたエンドポイントリソースです。

⑥ アクセス対象の GlusterFS ボリュームです。

7. OpenShift Container Platform マスターホストから PV を作成します。

```
$ oc create -f gluster-pv.yaml
```

8. PV が作成されたことを確認します。

```
$ oc get pv
NAME LABELS CAPACITY ACCESSMODES STATUS CLAIM
REASON AGE
gluster-default-volume <none> 2147483648 RWX Available 2s
```

9. **gluster-claim.yaml** で、新規 PV にバインドする PersistentVolumeClaim (PVC) を作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim ❶
spec:
  accessModes:
    - ReadWriteMany ❷
  resources:
    requests:
      storage: 1Gi ❸
```

- ❶ この要求名は、**volumes** セクションで Pod によって参照されます。
- ❷ PV の **accessModes** に一致する必要があります。
- ❸ この要求は、**1Gi** 以上の容量がある PV を検索します。

10. OpenShift Container Platform マスターホストから PVC を作成します。

```
$ oc create -f gluster-claim.yaml
```

11. PV と PVC がバインドされていることを確認します。

```
$ oc get pv
NAME          LABELS    CAPACITY  ACCESSMODES  STATUS   CLAIM          REASON  AGE
gluster-pv   <none>   1Gi      RWX          Available gluster-claim 37s

$ oc get pvc
NAME          LABELS    STATUS   VOLUME     CAPACITY  ACCESSMODES  AGE
gluster-claim <none>   Bound   gluster-pv 1Gi      RWX          24s
```



### 注記

PVC はプロジェクトごとに一意です。GlusterFS ボリュームにアクセスする各プロジェクトには独自の PVC が必要です。PV は単一のプロジェクトにバインドされないため、複数のプロジェクトにまたがる PVC が同じ PV を参照する場合があります。

#### 28.9.4. PersistentVolumeClaim のレジストリーへの割り当て

次に進む前に、**docker-registry** サービスが実行中であることを確認します。

```
$ oc get svc
NAME          CLUSTER_IP      EXTERNAL_IP  PORT(S)          SELECTOR                AGE
docker-registry 172.30.167.194 <none>       5000/TCP         docker-registry=default 18m
```



## 注記

`docker-registry` サービス、またはそれに関連付けられている Pod のいずれかが実行されていない場合は、[レジストリー](#) の設定手順を再度参照してトラブルシューティングを行ってから次に進んでください。

次に、PVC を割り当てます。

```
$ oc set volume deploymentconfigs/docker-registry --add --name=registry-storage -t pvc \
  --claim-name=gluster-claim --overwrite
```

OpenShift Container レジストリーの使用についての詳細は、[レジストリーのセットアップ](#) を参照してください。

## 28.10. ラベルによる永続ボリュームのバインド

### 28.10.1. 概要

このトピックでは、PV でラベルを定義して PVC 内でセレクターを一致させることで [Persistent Volume Claim \(永続ボリューム要求、PVC\)](#) を [永続ボリューム \(PV\)](#) にバインドする詳細例を紹介합니다。この機能はすべての [ストレージオプション](#) で使用できます。ここでは、OpenShift Container Platform クラスターに永続ストレージリソースが含まれていて、それらのリソースを PVC によるバインディングに使用できることを前提としています。

#### ラベルとセレクターに関する注記

ラベルは OpenShift Container Platform の機能であり、ユーザー定義のタグ (キーと値のペア) をオブジェクトの仕様の一部としてサポートします。その主な目的は、オブジェクト間で同一ラベルを定義してオブジェクトを任意にグループ化できるようにすることです。定義したラベルをセレクターでターゲットとして指定すると、指定のラベル値を持つすべてのオブジェクトが一致します。この機能により、PVC を PV にバインドすることができます。ラベルについての詳細は、[Pods and Services](#) を参照してください。



## 注記

この例では、変更された [GlusterFS](#) の PV および PVC 仕様を使用しています。ただし、実装したセレクターとラベルはすべての [ストレージオプション](#) で汎用的に使用できません。使用しているボリュームプロバイダーの独自の設定については、[関連するストレージオプション](#) を参照してください。

#### 28.10.1.1. 想定条件

以下があることを前提とします。

- 少なくとも1つのマスターと1つのノードがある既存の OpenShift Container Platform クラスター
- 少なくとも1つのサポート対象 [ストレージボリューム](#)
- `cluster-admin` 権限を持つユーザー

### 28.10.2. 仕様の定義



## 注記

ここでの仕様は **GlusterFS** に合わせてカスタマイズされています。使用しているボリュームプロバイダーの独自の設定については、[関連するストレージオプション](#) を参照してください。

### 28.10.2.1. ラベルのある永続ボリューム

#### 例28.13 glusterfs-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-volume
  labels: ❶
    storage-tier: gold
    aws-availability-zone: us-east-1
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
    endpoints: glusterfs-cluster ❷
    path: myVol1
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

- ❶ ラベルを使用して、ボリューム間で共有している共通の属性や特性を識別します。この例では、Gluster ボリュームを定義し、**storage-tier** という名前のカスタム属性 (キー) を持たせて、**gold** という値を割り当てています。要求で **storage-tier=gold** を使用して PV を選択すると、その PV に一致します。
- ❷ エンドポイントは、Gluster で信頼されるプールを定義します。これについては以下で説明します。

### 28.10.2.2. セレクターのある Persistent Volume Claim (永続ボリューム要求)

**selector** スタンザのある要求 (以下の例を参照してください) は、事前にバインドされていない既存の非要求の PV との一致を試みます。PVC セレクターがあるため、PV の容量は無視されます。ただし、**accessModes** は一致条件において考慮されます。

要求はその **selector** スタンザに含まれるすべてのキーと値のペアに一致する必要がある、ということに注意してください。要求に一致する PV がない場合、PVC はバインドされない (保留中の) ままになります。PV はその後作成され、要求によってラベルの一致の有無が自動的にチェックされます。

#### 例28.14 glusterfs-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
```

```
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector: ❶
  matchLabels:
    storage-tier: gold
    aws-availability-zone: us-east-1
```

- ❶ **selector** スタンザでは、この要求と一致させるために PV で必要なすべてのラベルを定義します。

### 28.10.2.3. ボリュームエンドポイント

PV を Gluster ボリュームに割り当てるには、オブジェクトを作成する前にエンドポイントを設定する必要があります。

#### 例28.15 glusterfs-ep.yaml

```
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster
subsets:
  - addresses:
    - ip: 192.168.122.221
    ports:
    - port: 1
  - addresses:
    - ip: 192.168.122.222
    ports:
    - port: 1
```

### 28.10.2.4. PV、PVC、およびエンドポイントのデプロイ

この例では、**oc** コマンドを **cluster-admin** 権限のあるユーザーとして実行します。実稼働環境では、クラスタークライアントが PVC の定義と作成を行うことなどが予想されます。

```
# oc create -f glusterfs-ep.yaml
endpoints "glusterfs-cluster" created
# oc create -f glusterfs-pv.yaml
persistentvolume "gluster-volume" created
# oc create -f glusterfs-pvc.yaml
persistentvolumeclaim "gluster-claim" created
```

最後に、PV と PVC が正常にバインドされていることを確認します。

```
# oc get pv,pvc
```

NAME	CAPACITY	ACCESSMODES	STATUS	CLAIM	REASON	AGE
gluster-volume	2Gi	RWX	Bound	gfs-trial/gluster-claim	7s	
NAME	STATUS	VOLUME	CAPACITY	ACCESSMODES	AGE	
gluster-claim	Bound	gluster-volume	2Gi	RWX	7s	



### 注記

PVC はプロジェクトに対してローカルですが、PV はクラスター全体にわたるグローバルリソースです。開発者および管理者以外のユーザーは、使用可能な PV のすべて（またはいずれか）にアクセスできない場合があります。

## 28.11. ストレージクラスを使用した動的プロビジョニング

### 28.11.1. 概要

この例では、[StorageClass](#) のさまざまな設定と Google Cloud Platform Compute Engine (GCE) を使用した動的プロビジョニングについて、いくつかのシナリオを紹介します。これらの例では、Kubernetes、GCE、および永続ディスクについて理解していること、また OpenShift Container Platform がインストールされていて [GCE を使用できるように適切に設定されている](#) ことを前提とします。

- [基本的な動的プロビジョニング](#)
- [クラスターの動的プロビジョニング動作のデフォルト設定](#)

### 28.11.2. シナリオ 1: 2 種類の StorageClass を持つ基本的な動的プロビジョニング

[StorageClass](#) を使用すると、ストレージのレベルや使用状況を区別し、記述することができます。この例では、**cluster-admin** または **storage-admin** が GCE で 2 つの異なるストレージのクラスを設定します。

- **slow**: 低コストで効率的なシーケンシャルデータの操作に最適化されている (低速読み取り/書き込み)
- **fast**: 高速なランダム IOPS と持続的スループットに最適化されている (高速読み取り/書き込み)

これらの [StorageClass](#) を作成することで、**cluster-admin** または **storage-admin** はユーザーに対して、[StorageClass](#) の特定のレベルまたはサービスについての要求の作成を許可することができます。

#### 例28.16 StorageClass 低速オブジェクトの定義

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow ①
provisioner: kubernetes.io/gce-pd ②
parameters:
  type: pd-standard ③
  zone: us-east1-d ④
```

- ① [StorageClass](#) の名前。
- ② 使用するプロビジョナープラグイン。[StorageClass](#) の必須フィールドです。

- 3 PD のタイプ。この例では **pd-standard** を使用しています。このタイプは、持続的 IOPS とスループットが高い **pd-ssd** と比べていくらかコストを下げられる一方、持続的 IOPS の速度や
- 4 ゾーンは必須です。

### 例28.17 StorageClass 高速オブジェクトの定義

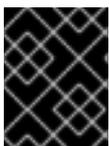
```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  zone: us-east1-d
```

**cluster-admin** または **storage-admin** として、両方の定義を YAML ファイルに保存します。例: **slow-gce.yaml** および **fast-gce.yaml**。次に **StorageClass** を作成します。

```
# oc create -f slow-gce.yaml
storageclass "slow" created

# oc create -f fast-gce.yaml
storageclass "fast" created

# oc get storageclass
NAME      TYPE
fast      kubernetes.io/gce-pd
slow      kubernetes.io/gce-pd
```



#### 重要

**cluster-admin** ユーザーまたは **storage-admin** ユーザーは、適切な **StorageClass** 名を適切なユーザー、グループ、およびプロジェクトに送る必要があります。

通常ユーザーとして、以下のように新規プロジェクトを作成します。

```
# oc new-project rh-eng
```

要求の YAML 定義を作成し、これをファイル (**pvc-fast.yaml**) に保存します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-engineering
spec:
  accessModes:
    - ReadWriteMany
resources:
```

```
requests:
  storage: 10Gi
storageClassName: fast
```

**oc create** コマンドを使用して要求を追加します。

```
# oc create -f pvc-fast.yaml
persistentvolumeclaim "pvc-engineering" created
```

要求がバインドされているかどうかをチェックします。

```
# oc get pvc
NAME                STATUS  VOLUME                                     CAPACITY  ACCESSMODES  AGE
pvc-engineering    Bound  pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  10Gi      RWX           2m
```



### 重要

この要求は **rh-eng** プロジェクトで作成され、バインドされているため、同じプロジェクトのいずれのユーザーにも共有できます。

**cluster-admin** ユーザーまたは **storage-admin** ユーザーとして、最近動的にプロビジョニングした永続ボリューム (PV) を表示します。

```
# oc get pv
NAME                CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS
CLAIM              REASON    AGE
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  10Gi      RWX          Delete         Bound  rh-
eng/pvc-engineering                    5m
```



### 重要

動的にプロビジョニングされたすべてのボリュームについて、**RECLAIMPOLICY** がデフォルトで **Delete** になっていることに注意してください。これは、ボリュームが要求がシステムに存在している間存続することを意味します。要求を削除するとボリュームも削除され、ボリュームのすべてのデータが失われます。

最後に GCE コンソールをチェックします。新規のディスクが作成され、使用できる状態になります。

```
kubernetes-dynamic-pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  SSD persistent disk  10 GB  us-
east1-d
```

これで、Pod で Persistent Volume Claim (永続ボリューム要求) を参照し、ボリュームの使用を開始することができます。

### 28.11.3. シナリオ 2: クラスタにおけるデフォルトの StorageClass の動作を有効にする方法

この例では、**cluster-admin** または **storage-admin** により、**StorageClass** を要求に暗黙的に指定していない他のすべてのユーザーおよびプロジェクトについてデフォルトのストレージクラスが有効になります。この方法は、特化した **StorageClasses** をクラスタ全体に設定し、伝達することなしに

**cluster-admin** または **storage-admin** がストレージボリュームを容易に管理できるようにする場合に役に立ちます。

以下の例は「シナリオ 1: 2 種類の **StorageClass** を持つ基本的な動的プロビジョニング」に基づいて作成されています。**cluster-admin** または **storage-admin** は、デフォルトの **StorageClass** として指定するための別の **StorageClass** を作成します。

#### 例28.18 デフォルトの StorageClass オブジェクトの定義

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: generic ❶
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" ❷
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  zone: us-east1-d
```

- ❶ **StorageClass** の名前。クラスター内で一意にする必要があります。
- ❷ この **StorageClass** にデフォルトクラスのマークを付けるアノテーション。このバージョンの API では引用符付きの **"true"** を使用する必要があります。このアノテーションがない場合、OpenShift Container Platform はこれをデフォルトの **StorageClass** ではないと見なします。

**cluster-admin** または **storage-admin** として、この定義を YAML ファイル (**generic-gce.yaml**) に保存し、**StorageClass** を作成します。

```
# oc create -f generic-gce.yaml
storageclass "generic" created

# oc get storageclass
NAME      TYPE
generic   kubernetes.io/gce-pd
fast      kubernetes.io/gce-pd
slow      kubernetes.io/gce-pd
```

通常ユーザーとして、**StorageClass** の要件なしに新規の要求定義を作成し、これをファイル (**generic-pvc.yaml**) に保存します。

#### 例28.19 デフォルトのストレージ要求オブジェクトの定義

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-engineering2
spec:
  accessModes:
    - ReadWriteMany
```

```
resources:
  requests:
    storage: 5Gi
```

以下のように実行し、要求がバインドされていることをチェックします。

```
# oc create -f generic-pvc.yaml
persistentvolumeclaim "pvc-engineering2" created
3s

# oc get pvc
NAME                STATUS  VOLUME                                     CAPACITY  ACCESSMODES  AGE
pvc-engineering     Bound  pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  10Gi      RWX          41m
pvc-engineering2    Bound  pvc-a9f70544-8bfd-11e6-9962-42010af00004  5Gi       RWX          7s
```

1

- 1 **pvc-engineering2** は、デフォルトで、動的にプロビジョニングされたボリュームにバインドされます。

**cluster-admin** または **storage-admin** として、ここまでに定義した永続ボリュームを表示します。

```
# oc get pv
NAME                CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS
CLAIM              REASON    AGE
pvc-a9f70544-8bfd-11e6-9962-42010af00004  5Gi      RWX          Delete         Bound  rh-
eng/pvc-engineering2  5m      1
pvc-ba4612ce-8b4d-11e6-9962-42010af00004  5Gi      RWO          Delete         Bound
mytest/gce-dyn-claim1  21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  10Gi     RWX          Delete         Bound  rh-
eng/pvc-engineering  46m     2
```

- 1 この PV は、デフォルトの **StorageClass** からデフォルトの動的ボリュームにバインドされています。
- 2 この PV は **高速 StorageClass** を使用して「シナリオ 1: 2 種類の **StorageClass** を持つ基本的な動的プロビジョニング」の1番目の PVC にバインドされています。

**GCE** を使用して (動的にプロビジョニングされるのではなく) 手動でプロビジョニングされるディスクを作成します。次に、新規の **GCE** ディスク (**pv-manual-gce.yaml**) に接続する **永続ボリューム** を作成します。

#### 例28.20 手動の PV オブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-manual-gce
spec:
  capacity:
    storage: 35Gi
  accessModes:
    - ReadWriteMany
```

```
gcePersistentDisk:
  readOnly: false
  pdName: the-newly-created-gce-PD
  fsType: ext4
```

オブジェクト定義ファイルを実行します。

```
# oc create -f pv-manual-gce.yaml
```

ここでもう一度 PV を表示します。**pv-manual-gce** ボリュームが **Available** になっていることに留意してください。

```
# oc get pv
NAME                                     CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS
CLAIM          REASON  AGE
pv-manual-gce  35Gi    RWX          Retain         Available
4s
pvc-a9f70544-8bfd-11e6-9962-42010af00004  5Gi    RWX          Delete         Bound    rh-
eng/pvc-engineering2          12m
pvc-ba4612ce-8b4d-11e6-9962-42010af00004  5Gi    RWO          Delete         Bound
mytest/gce-dyn-claim1         21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  10Gi   RWX          Delete         Bound    rh-
eng/pvc-engineering          53m
```

今度は **generic-pvc.yaml** PVC 定義と同一の別の要求を作成しますが、名前を変更し、ストレージクラス名は設定しません。

#### 例28.21 要求オブジェクトの定義

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-engineering3
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 15Gi
```

このインスタンスではデフォルトの **StorageClass** が有効になっているため、手動で作成された PV は要求のリクエストを満たしません。ユーザーは動的にプロビジョニングされた新規の永続ボリュームを受け取ります。

```
# oc get pvc
NAME          STATUS  VOLUME                                     CAPACITY  ACCESSMODES  AGE
pvc-engineering  Bound  pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  10Gi    RWX          1h
pvc-engineering2  Bound  pvc-a9f70544-8bfd-11e6-9962-42010af00004  5Gi    RWX          19m
pvc-engineering3  Bound  pvc-6fa8e73b-8c00-11e6-9962-42010af00004  15Gi   RWX          6s
```



## 重要

デフォルトの **StorageClass** がこのシステムで有効になっているため、手動で作成された永続ボリュームを前述の要求によってバインドし、動的にプロビジョニングされた新規ボリュームがバインドされないようにするには、PV をデフォルトの **StorageClass** で作成しておく必要があります。

デフォルトの **StorageClass** がこのシステムで有効になっているため、手動で作成された永続ボリュームを前述の要求によってバインドし、動的にプロビジョニングされた新規ボリュームをバインドしないようにするためには、PV をデフォルトの **StorageClass** で作成しておく必要があります。

これを解決するには、**cluster-admin** ユーザーまたは **storage-admin** ユーザーは、別の GCE ディスクを作成するか、または必要に応じて最初の手動の PV を削除し、**StorageClass** 名を割り当てる PV オブジェクト定義 (**pv-manual-gce2.yaml**) を使用することのみが必要になります。

### 例28.22 デフォルトの StorageClass 名を持つ手動 PV の仕様

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-manual-gce2
spec:
  capacity:
    storage: 35Gi
  accessModes:
    - ReadWriteMany
  gcePersistentDisk:
    readOnly: false
    pdName: the-newly-created-gce-PD
    fsType: ext4
    storageClassName: generic ❶
```

❶ 先に作成した汎用の **StorageClass** の名前。

オブジェクト定義ファイルを実行します。

```
# oc create -f pv-manual-gce2.yaml
```

PV を一覧表示します。

```
# oc get pv
NAME                                CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS
CLAIM      REASON  AGE
pv-manual-gce2  35Gi  RWX          Retain          Available
4s ❶
pv-manual-gce2  35Gi  RWX          Retain          Bound      rh-eng/pvc-
engineering3    4s ❷
pvc-a9f70544-8bfd-11e6-9962-42010af00004  5Gi  RWX          Delete          Bound      rh-
eng/pvc-engineering2    12m
pvc-ba4612ce-8b4d-11e6-9962-42010af00004  5Gi  RWO          Delete          Bound
```

```
mytest/gce-dyn-claim1      21h
pvc-e9b4fef7-8bf7-11e6-9962-42010af00004  10Gi  RWX  Delete  Bound  rh-
eng/pvc-engineering      53m
```

- 1 元の手動 PV はまだバインドされておらず、Available です。これは、**default StorageClass** で作成されていないためです。
- 2 2 番目の PVC (名前以外) は手動で作成された Available の PV である **pv-manual-gce2** にバインドされています。



### 重要

動的にプロビジョニングされたすべてのボリュームの **RECLAIMPOLICY** がデフォルトで **Delete** である点に注目してください。PV に動的にバインドされた PVC が削除されると、GCE ボリュームが削除され、すべてのデータが消失します。ただし、手動で作成された PV の **RECLAIMPOLICY** はデフォルトで **Retain** です。

## 28.12. 既存のレガシーストレージに対するストレージクラスの使用

### 28.12.1. 概要

この例では、レガシーデータボリュームが存在し、**cluster-admin** または **storage-admin** がそのボリュームを特定のプロジェクトで使用できるようにする必要があります。**StorageClass** を使用すると、他のユーザーおよびプロジェクトがこのボリュームへのアクセスを要求から取得する可能性が低くなります。これは、要求には完全一致する **StorageClass** 名の値が必要になるためです。また、この例では動的なプロビジョニングも無効にしています。この例では以下の要件を満たしていることを前提としています。

- OpenShift Container Platform、GCE、および永続ディスクについてある程度理解している。
- OpenShift Container Platform が [GCE](#) を使用するよう適切に設定されている。

#### 28.12.1.1. シナリオ 1: レガシーデータを含む既存の永続ボリュームに StorageClass をリンクさせる

**cluster-admin** または **storage-admin** として、過去の財務データ用の **StorageClass** を定義し、作成します。

#### 例28.23 StorageClass の finance-history オブジェクトの定義

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: finance-history 1
provisioner: no-provisioning 2
parameters: 3
```

- 1 StorageClass の名前。
- 2 これは必須フィールドです。しかし、動的なプロビジョニングは行われなため、実際のプロビジョナーのプラグインタイプでない限り、このフィールドに値を指定する必要があります。

- 3 パラメーターは動的プロビジョナーで使用されるだけなので、空白のままにしておくことができます。

この定義を YAML ファイル (**finance-history-storageclass.yaml**) に保存して、**StorageClass** を作成します。

```
# oc create -f finance-history-storageclass.yaml
storageclass "finance-history" created

# oc get storageclass
NAME          TYPE
finance-history no-provisioning
```



### 重要

**cluster-admin** ユーザーまたは **storage-admin** ユーザーは、適切な **StorageClass** 名を適切なユーザー、グループ、およびプロジェクトに送る必要があります。

**StorageClass** が存在します。**cluster-admin** または **storage-admin** は **StorageClass** で使用するための永続ボリューム (PV) を作成することができます。(動的にプロビジョニングされない) **GCE** および新しい **GCE** ディスク (**gce-pv.yaml**) に接続される **永続ボリューム** を使用して、手動でプロビジョニングされたディスクを作成します。

#### 例28.24 財務履歴の PV オブジェクト

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-finance-history
spec:
  capacity:
    storage: 35Gi
  accessModes:
    - ReadWriteMany
  gcePersistentDisk:
    readOnly: false
    pdName: the-existing-PD-volume-name-that-contains-the-valuable-data 1
    fsType: ext4
    storageClassName: finance-history 2
```

- 2 **StorageClass** 名。完全一致している必要があります。
- 1 すでに存在し、レガシーデータが含まれている **GCE** ディスクの名前。

**cluster-admin** または **storage-admin** として、PV を作成し、これを表示します。

```
# oc create -f gce-pv.yaml
persistentvolume "pv-finance-history" created
```

```
# oc get pv
NAME          CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS  CLAIM
REASON  AGE
pv-finance-history  35Gi    RWX          Retain         Available          2d
```

**pv-finance-history** が Available で、いつでも利用可能であることを留意してください。

ユーザーとして、Persistent Volume Claim (永続ボリューム要求、PVC) を YAML ファイルとして作成し、以下のように適切な **StorageClass** 名を指定します。

#### 例28.25 finance-history オブジェクト定義の要求

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-finance-history
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 20Gi
  storageClassName: finance-history ❶
```

- ❶ **StorageClass** 名。完全一致している必要があります。そうでない場合には、削除されるか、または名前が一致する別の **StorageClass** が作成されるまで要求が非バインドの状態になります。

PVC と PV を作成および表示して、バインドされているか確認します。

```
# oc create -f pvc-finance-history.yaml
persistentvolumeclaim "pvc-finance-history" created

# oc get pvc
NAME          STATUS  VOLUME          CAPACITY  ACCESSMODES  AGE
pvc-finance-history  Bound  pv-finance-history  35Gi    RWX          9m

# oc get pv (cluster/storage-admin)
NAME          CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS  CLAIM
REASON  AGE
pv-finance-history  35Gi    RWX          Retain         Bound   default/pvc-finance-history
5m
```



#### 重要

同じクラスター内の **StorageClass** を、レガシーデータ (動的プロビジョニングなし) および **動的プロビジョニング** の両方に対して使用することができます。

## 28.13. AZURE BLOB ストレージでの統合コンテナイメージレジストリーの設定

### 28.13.1. 概要

このトピックでは、[Microsoft Azure Blob Storage](#) で [OpenShift 統合コンテナイメージレジストリー](#) を設定する方法を説明します。

### 28.13.2. 操作を始める前に

- Microsoft Azure Portal、Microsoft Azure CLI、または Microsoft Azure Storage Explorer を使用してストレージコンテナを作成します。ストレージアカウント名、ストレージアカウントキー、およびコンテナ名をメモしてください。
- デプロイされていない場合は、[統合コンテナイメージレジストリーをデプロイ](#) します。

### 28.13.3. レジストリー設定の上書き

新規レジストリー Pod を作成し、古い Pod と自動的に置き換えるには、以下の手順を実行します。

1. `registryconfig.yaml` という名前の新規レジストリー設定ファイルを作成して、以下の情報を追加します。

```
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    blobdescriptor: inmemory
  delete:
    enabled: true
azure: ❶
  accountname: azureblobacc
  accountkey: azureblobacckey
  container: azureblobname
  realm: core.windows.net ❷
auth:
  openshift:
    realm: openshift
middleware:
  registry:
    - name: openshift
  repository:
    - name: openshift
    options:
      acceptschema2: false
      pullthrough: true
      enforcequota: false
      projectcachettl: 1m
      blobrepositorycachettl: 10m
storage:
  - name: openshift
```

- 1 **accountname**、**acountkey**、および **container** の値をそれぞれストレージアカウント名、ストレージアカウントキー、およびストレージコンテナ名で置き換えます。
- 2 Azure の地域クラウドを使用している場合は、必要なレルムに設定します。たとえば、ドイツの地域クラウドの場合は **core.cloudapi.de** に設定します。

2. 新規レジストリー設定を作成します。

```
$ oc create secret generic registry-config --from-file=config.yaml=registryconfig.yaml
```

3. シークレットを追加します。

```
$ oc set volume dc/docker-registry --add --type=secret \
--secret-name=registry-config -m /etc/docker/registry/
```

4. **REGISTRY\_CONFIGURATION\_PATH** 環境変数を設定します。

```
$ oc set env dc/docker-registry \
REGISTRY_CONFIGURATION_PATH=/etc/docker/registry/config.yaml
```

5. レジストリー設定をすでに作成している場合は、以下の手順を実行します。

a. シークレットを削除します。

```
$ oc delete secret registry-config
```

b. 新規レジストリー設定を作成します。

```
$ oc create secret generic registry-config --from-file=config.yaml=registryconfig.yaml
```

c. 新規ロールアウトを開始して設定を更新します。

```
$ oc rollout latest docker-registry
```

## 第29章 一時ストレージの設定

### 29.1. 概要

OpenShift Container Platform は、Pod およびコンテナの作業データの一時ストレージを管理できるように、設定可能です。コンテナは、記述可能な階層、ログディレクトリ、EmptyDir ボリュームを活用できる反面、このストレージには複数の制限があります。この点については、[ここで説明されています](#)。

一時ストレージ管理により、管理者は個別の Pod やコンテナが消費するリソースを制限できるようになるだけでなく、Pod およびコンテナが対象の一時ストレージの使用を制限したり、要求したりできます。これは、テクノロジープレビュー機能で、デフォルトでは無効になっています。



#### 注記

このテクノロジープレビュー機能は、OpenShift Container Platform でローカルストレージを公開するメカニズムを変更するわけではなく、既存のメカニズム、root ディレクトリや runtime ディレクトリはそのまま使用されます。このテクノロジープレビュー機能は、当リソースの使用の管理メカニズムを提供するだけです。

### 29.2. 一時ストレージの有効化

一時ストレージを有効化するには、以下を実行します。

1. すべてのマスターで、マスター設定ファイル (デフォルトは `/etc/origin/master/master-config.yaml`) を編集するか、または作成して **LocalStorageCapacityIsolation=true** を **apiServerArguments** と **controllerArguments** の各セクションに追加します。

```
apiServerArguments:
  feature-gates:
  - LocalStorageCapacityIsolation=true
...

controllerArguments:
  feature-gates:
  - LocalStorageCapacityIsolation=true
...
```

2. コマンドラインで、全ノードの ConfigMap を編集して、LocalStorageCapacityIsolation を有効化します。編集の必要のある ConfigMaps は以下のように特定します。

```
$ oc get cm -n openshift-node
NAME          DATA  AGE
node-config-compute  1     52m
node-config-infra  1     52m
node-config-master  1     52m
```

**node-config-compute**、**node-config-infra** および **node-config-master** のマップごとに、機能ゲートを追加する必要があります。

```
oc edit cm node-config-master -n openshift-node
```

すでに **feature-gates:** の宣言がある場合には、以下のテキストを機能ゲートの一覧に追加します。

```
,LocalStorageCapacityIsolation=true
```

**feature-gates:** の宣言がない場合には、以下のセクションを追加します。

```
feature-gates:  
- LocalStorageCapacityIsolation=true
```

3. **node-config-compute**、**node-config-infra** をはじめ、他の ConfigMaps についても同様に繰り返します。
4. OpenShift Container Platform を再起動して、apiserver を実行するコンテナを削除します。



#### 注記

これらの手順を省略すると、一時ストレージの管理が有効化されなくなる可能性があります。

## 第30章 HTTP プロキシの使用

### 30.1. 概要

実稼働環境では、インターネットへの直接アクセスを拒否し、代わりに HTTP または HTTPS プロキシを使用することができます。これらのプロキシを使用するように OpenShift Container Platform を設定することは、設定ファイルまたは JSON ファイルで標準的な環境変数を設定するのと同じくらい簡単に実行できます。この設定は、[クラスターインストール](#)時に実行するか、またはインストール後に行うことができます。

プロキシ設定はクラスター内の各ホストで同じである必要があります。したがって、プロキシの設定やその変更を行う場合は、各 OpenShift Container Platform ホスト上のファイルを更新して同じ値にする必要があります。その後はクラスター内の各ホストで OpenShift Container Platform サービスを再起動する必要があります。

**NO\_PROXY**、**HTTP\_PROXY**、および **HTTPS\_PROXY** 環境変数は、各ホストの `/etc/origin/master/master.env` および `/etc/sysconfig/atomic-openshift-node` ファイルにあります。

### 30.2. NO\_PROXY の設定

**NO\_PROXY** 環境変数は、OpenShift Container Platform のすべてのコンポーネントと、OpenShift Container Platform によって管理されるすべての IP アドレスを一覧表示します。

CIDR を許可する OpenShift サービスでは、以下のように **NO\_PROXY** にホスト、IP アドレス、または IP 範囲 (CIDR 形式) のコンマ区切りの一覧を指定できます。

#### マスターホストの場合

- ノードホスト名
- マスター IP またはホスト名
- etcd ホストの IP アドレス

#### ノードホストの場合

- マスター IP またはホスト名

#### Docker サービスの場合

- レジストリーのサービス IP とホスト名
- レジストリーサービス URL **docker-registry.default.svc.cluster.local**
- レジストリールートホスト名 (作成されている場合)



## 注記

Docker を使用している場合は、Docker はホスト、ドメイン拡張子、または IP アドレスのコンマ区切りの一覧を受け入れますが、CIDR 形式の IP 範囲は受け入れません。CIDR 形式の IP 範囲を受け入れるのは OpenShift サービスのみです。`no\_proxy` 変数には、プロキシを使用すべきでないドメイン拡張子のコンマ区切りの一覧を含める必要があります。

たとえば、**no\_proxy** が **.school.edu** に設定されている場合は、特定の学校からドキュメントを取得するためにプロキシが使用されることはありません。

**NO\_PROXY** には、**master-config.yaml** ファイルにあるように、SDN ネットワークとサービス IP アドレスも含まれています。

**/etc/origin/master/master-config.yaml**

```
networkConfig:
  clusterNetworks:
  - cidr: 10.1.0.0/16
    hostSubnetLength: 9
  serviceNetworkCIDR: 172.30.0.0/16
```

OpenShift Container Platform では、ドメイン接尾辞に付加するワイルドカードとして \* を使用できません。たとえば、以下は受け入れられません。

```
NO_PROXY=.example.com
```

しかし、以下の場合は受け入れられません。

```
NO_PROXY=*.example.com
```

**NO\_PROXY** で許可されるワイルドカードは \*1文字だけです。このワイルドカードはすべてのホストに一致し、プロキシを効果的に無効にします。

この一覧にある名前はそれぞれ、ホスト名を接尾辞として含むドメインまたはホスト名自体のいずれかと一致します。



## 注記

ノードを拡張するときは、ホスト名の一覧ではなくドメイン名を使用してください。

たとえば、**example.com** は **example.com**、**example.com:80**、および **www.example.com** と一致しません。

## 30.3. ホストでのプロキシの設定

1. OpenShift Container Platform 制御ファイルのプロキシ環境変数を編集します。クラスター内のすべてのファイルが正しいことを確認してください。

```
HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>/
HTTPS_PROXY=https://<user>:<password>@<ip_addr>:<port>/
NO_PROXY=master.hostname.example.com,10.1.0.0/16,172.30.0.0/16 1
```

- 1 ホスト名と CIDR をサポートします。SDN ネットワークとサービスの IP 範囲 **10.1.0.0/16,172.30.0.0/16** がデフォルトで含まれている必要があります。

2. マスターまたはノードホストを再起動します。

```
# master-restart api
# master-restart controllers
# systemctl restart atomic-openshift-node
```

## 30.4. ANSIBLE を使用したホストでのプロキシ設定

クラスターのインストール中に、**openshift\_no\_proxy**、**openshift\_http\_proxy** および **openshift\_https\_proxy** パラメーターを使用して、**NO\_PROXY**、**HTTP\_PROXY** および **HTTPS\_PROXY** の環境変数を設定することができます。これらは、[インベントリーファイル](#) で設定可能です。

### Ansible を使用したプロキシ設定例

```
# Global Proxy Configuration
# These options configure HTTP_PROXY, HTTPS_PROXY, and NO_PROXY environment
# variables for docker and master services.
openshift_http_proxy=http://<user>:<password>@<ip_addr>:<port>
openshift_https_proxy=https://<user>:<password>@<ip_addr>:<port>
openshift_no_proxy='.hosts.example.com,some-host.com'
#
# Most environments do not require a proxy between OpenShift masters, nodes, and
# etcd hosts. So automatically add those host names to the openshift_no_proxy list.
# If all of your hosts share a common domain you may wish to disable this and
# specify that domain above.
# openshift_generate_no_proxy_hosts=True
```

### 注記

Ansible パラメーターを使用して [ビルド用に設定](#) できる [その他のプロキシ設定](#) があります。以下に例を示します。

**openshift\_builddefaults\_git\_http\_proxy** パラメーターと **openshift\_builddefaults\_git\_https\_proxy** パラメーターを使用すると、[Git クローン作成のためにプロキシを使用](#) することができます。

**openshift\_builddefaults\_http\_proxy** パラメーターと **openshift\_builddefaults\_https\_proxy** パラメーターは、[Docker ビルドストラテジー](#) と [カスタムビルドストラテジー](#) のプロセスで環境変数を利用可能にできます。

## 30.5. DOCKER PULL のプロキシ設定

OpenShift Container Platform のノードホストは Docker レジストリーに対してプッシュ操作とプル操作を実行する必要があります。ノードがアクセスするためにプロキシを必要としないレジストリーの場合は、以下の項目を指定した **NO\_PROXY** パラメーターを含めてください。

- レジストリーのホスト名
- レジストリーサービスの IP アドレス

- サービス名

これにより、外部の HTTP プロキシをオプションのままにして、そのレジストリーをブラックリストに入れることができます。

1. 以下を実行して、レジストリーサービスの IP アドレス **docker\_registry\_ip** を取得します。

```
$ oc describe svc/docker-registry -n default
```

```
Name: docker-registry
Namespace: default
Labels: docker-registry=default
Selector: docker-registry=default
Type: ClusterIP
IP: 172.30.163.183 1
Port: 5000-tcp 5000/TCP
Endpoints: 10.1.0.40:5000
Session Affinity: ClientIP
No events.
```

- 1** レジストリーサービスの IP です。

2. `/etc/sysconfig/docker` ファイルを編集し、シェル形式の **NO\_PROXY** 変数を追加して、**<docker\_registry\_ip>** を直前の手順で取得した IP アドレスに置き換えます。

```
HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>/
HTTPS_PROXY=https://<user>:<password>@<ip_addr>:<port>/
NO_PROXY=master.hostname.example.com,<docker_registry_ip>,docker-registry.default.svc.cluster.local
```

3. Docker サービスを再起動します。

```
# systemctl restart docker
```

## 30.6. プロキシの背後での MAVEN の使用

プロキシで Maven を使用するには、**HTTP\_PROXY\_NONPROXYHOSTS** 変数を使用する必要があります。

Maven をプロキシの背後に設定する手順などを含めた Red Hat JBoss Enterprise Application Platform 向けの OpenShift Container Platform 環境の設定に関する詳細については、[Red Hat JBoss Enterprise Application Platform for OpenShift](#) を参照してください。

## 30.7. S2I ビルドでのプロキシの設定

S2I ビルドはさまざまな場所から依存関係を取得します。[.s2i/environment](#) ファイルを使用して単純なシェル変数を指定することができます。それに応じて OpenShift Container Platform はビルドイメージの参照時に応答します。

以下は、サンプル値を指定したサポート対象のプロキシ環境変数です。

```

HTTP_PROXY=http://USERNAME:PASSWORD@10.0.1.1:8080/
HTTPS_PROXY=https://USERNAME:PASSWORD@10.0.0.1:8080/
NO_PROXY=master.hostname.example.com

```

## 30.8. デフォルトテンプレートでのプロキシの設定

OpenShift Container Platform でデフォルトで利用可能な [テンプレートサンプル](#) には、HTTP プロキシの設定が含まれていません。これらのテンプレートに基づく既存のアプリケーションについては、アプリケーションのビルド設定の **source** セクションを変更して、以下のプロキシ設定を追加します。

```

...
source:
  type: Git
  git:
    uri: https://github.com/openshift/ruby-hello-world
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
    noProxy: somedomain.com, otherdomain.com
...

```

これは、[Git クローン作成用にプロキシを使用する](#) プロセスと似ています。

## 30.9. POD でのプロキシ環境変数の設定

デプロイメント設定の **templates.spec.containers** スタンザで **NO\_PROXY**、**HTTP\_PROXY**、および **HTTPS\_PROXY** 環境変数を設定して、プロキシ接続情報を渡すことができます。実行時の Pod のプロキシ設定についても同じことを実行できます。

```

...
containers:
- env:
  - name: "HTTP_PROXY"
    value: "http://<user>:<password>@<ip_addr>:<port>"
...

```

**oc set env** コマンドを使用して、既存のデプロイメント設定を新規の環境変数で更新することもできます。

```
$ oc set env dc/frontend HTTP_PROXY=http://<user>:<password>@<ip_addr>:<port>
```

OpenShift Container Platform インスタンスで [ConfigChange トリガー](#) を設定している場合は、変更が自動的に行われます。そうでない場合は、変更が反映されるようにアプリケーションを手動で再デプロイしてください。

## 30.10. GIT リポジトリのアクセス

プロキシの使用によってのみ Git リポジトリにアクセスできる場合は、使用するプロキシを **BuildConfig** の **source** セクションで定義できます。HTTP および HTTPS プロキシの両方を設定できますが、いずれのフィールドもオプションです。いずれのフィールドもオプションです。NoProxy フィールドで、プロキシを実行しないドメインを指定することもできます。



## 注記

実際に機能させるには、ソース URI で HTTP または HTTPS プロトコルを使用する必要があります。

source:

git:

uri: "https://github.com/openshift/ruby-hello-world"

httpProxy: http://proxy.example.com

httpsProxy: https://proxy.example.com

noProxy: somedomain.com, otherdomain.com

## 第31章 グローバルビルドのデフォルトと上書きの設定

### 31.1. 概要

開発者はプロジェクト内の特定のビルド設定において、[Git クローン作成のためのプロキシ設定](#)などの設定を定義することができます。開発者に特定の設定をそれぞれのビルド設定で定義するように要求するのではなく、管理者が受付制御プラグインを使用して、すべてのビルドでこの設定を自動的に使用するグローバルビルドのデフォルトと上書きを設定することができます。

これらのプラグインから取得した設定は、ビルドプロセス時に使用されるだけで、ビルド設定やビルド自体で設定されません。プラグインでの設定を使用することで、管理者はグローバル設定をいつでも変更することができ、既存のビルド設定またはビルドから実行されたビルドに、新規設定が割り当てられます。

- **BuildDefaults** 受付制御プラグインを使用すると、管理者は Git HTTP や HTTPS プロキシなどの設定についてのグローバルなデフォルトと、デフォルトの環境変数を設定することができます。これらのデフォルトによって、特定のビルド用に設定された値が上書きされることはありません。ただし、それらの値がビルド定義に存在しない場合は、デフォルト値に設定されません。
- **BuildOverrides** 受付制御プラグインを使用すると、管理者はビルドに保存されている値に関係なく、ビルドの設定を上書きできます。  
現時点では、このプラグインには、ビルド時にレジストリーからのローカルイメージが強制的に更新されるように、[ビルドストラテジーの forcePull フラグを上書き](#)するサポートがあります。つまり、ビルドの開始時にアクセスチェックがイメージで実行され、ユーザーがプルできるイメージでしかビルドできないようにします。強制的に更新され、ビルドにマルチテナンシーが提供されます。ただし、ビルドノードに保存されているイメージのローカルキャッシュは依存せず、常にレジストリーにアクセスできるようにする必要があります。

このプラグインは、イメージラベルセットが全ビルドイメージに適用されるように設定することも可能です。

**BuildOverrides** の受付制御プラグインと上書き可能な値の設定に関する情報は、[グローバルビルドの上書きの手動設定](#)を参照してください。

デフォルトのノードセクターおよび **BuildDefaults** または **BuildOverrides** 受付プラグインは以下のように連携します。

- マスター設定ファイルの **projectConfig.defaultNodeSelector** フィールドに定義されているデフォルトのプロジェクトノードセクターは、指定の **nodeSelector** 値なしに、全プロジェクトに作成済みの Pod に適用されます。これらの設定は、**BuildDefaults** または **BuildOverrides** ノードセクターが設定されていないクラスターで、**nodeSelector="null"** が指定されているビルドに適用されます。
- **nodeSelector="null"** パラメーターがビルド設定に設定されている場合のみ、クラスター全体のデフォルトのビルドノードセクター **admissionConfig.pluginConfig.BuildDefaults.configuration.nodeSelector** が適用されません。**nodeSelector=null** はデフォルト設定です。
- デフォルトのプロジェクトまたはクラスター全体のノードセクターの場合には、デフォルト設定がビルドのノードセクターに AND として追加されます。このビルドのノードセクターは、**BuildDefaults** または **BuildOverride** 受付プラグインで設定されます。これらの設定は、**BuildOverrides** ノードセクターとプロジェクトのデフォルトノードセクターの条件を満たすノードに対してのみスケジューリングされるという意味です。



## 注記

**RunOnceDuration** プラグインを使用することで、ビルド Pod の実行時間のハード制限を定義できます。

## 31.2. グローバルビルドのデフォルトの設定

グローバルビルドのデフォルトは以下の 2 通りの方法で設定できます。

- [Ansible の使用](#)
- [master-config.yaml ファイルを変更して手動で設定する](#)

### 31.2.1. Ansible を使用したグローバルビルドのデフォルトの設定

クラスターのインストール時に、[以下のパラメーター](#)を使用して、**BuildDefaults** プラグインを設定できます。これらのパラメーターは、インベントリーファイルで設定可能です。

- **openshift\_builddefaults\_http\_proxy**
- **openshift\_builddefaults\_https\_proxy**
- **openshift\_builddefaults\_no\_proxy**
- **openshift\_builddefaults\_git\_http\_proxy**
- **openshift\_builddefaults\_git\_https\_proxy**
- **openshift\_builddefaults\_git\_no\_proxy**
- **openshift\_builddefaults\_image\_labels**
- **openshift\_builddefaults\_nodeselectors**
- **openshift\_builddefaults\_annotations**
- **openshift\_builddefaults\_resources\_requests\_cpu**
- **openshift\_builddefaults\_resources\_requests\_memory**
- **openshift\_builddefaults\_resources\_limits\_cpu**
- **openshift\_builddefaults\_resources\_limits\_memory**

#### 例31.1 Ansible を使用したビルドのデフォルトの設定例

```
# These options configure the BuildDefaults admission controller which injects
# configuration into Builds. Proxy related values will default to the global proxy
# config values. You only need to set these if they differ from the global proxy settings.
openshift_builddefaults_http_proxy=http://USER:PASSWORD@HOST:PORT
openshift_builddefaults_https_proxy=https://USER:PASSWORD@HOST:PORT
openshift_builddefaults_no_proxy=mycorp.com
openshift_builddefaults_git_http_proxy=http://USER:PASSWORD@HOST:PORT
openshift_builddefaults_git_https_proxy=https://USER:PASSWORD@HOST:PORT
openshift_builddefaults_git_no_proxy=mycorp.com
openshift_builddefaults_image_labels=[{'name':'imagelabelname1','value':'imagelabelvalue1'}]
```

```

openshift_builddefaults_nodeselectors={'nodelabel1':'nodelabelvalue1'}
openshift_builddefaults_annotations={'annotationkey1':'annotationvalue1'}
openshift_builddefaults_resources_requests_cpu=100m
openshift_builddefaults_resources_requests_memory=256Mi
openshift_builddefaults_resources_limits_cpu=1000m
openshift_builddefaults_resources_limits_memory=512Mi

# Or you may optionally define your own build defaults configuration serialized as json
#openshift_builddefaults_json='{"BuildDefaults":{"configuration":{"apiVersion":"v1","env":
[{"name":"HTTP_PROXY","value":"http://proxy.example.com.redhat.com:3128"},
{"name":"NO_PROXY","value":"ose3-
master.example.com"}],"gitHTTPProxy":"http://proxy.example.com:3128","gitNoProxy":"ose3-
master.example.com"},"kind":"BuildDefaultsConfig"}}'

```

### 31.2.2. グローバルビルドのデフォルトの手動設定

**BuildDefaults** プラグインを設定するには、以下の手順を実行します。

1. マスターノードの `/etc/origin/master/master-config.yaml` ファイルにプラグインの設定を追加します。

```

admissionConfig:
  pluginConfig:
    BuildDefaults:
      configuration:
        apiVersion: v1
        kind: BuildDefaultsConfig
        gitHTTPProxy: http://my.proxy:8080 ①
        gitHTTPSPProxy: https://my.proxy:8443 ②
        gitNoProxy: somedomain.com, otherdomain.com ③
        env:
          - name: HTTP_PROXY ④
            value: http://my.proxy:8080
          - name: HTTPS_PROXY ⑤
            value: https://my.proxy:8443
          - name: BUILD_LOGLEVEL ⑥
            value: 4
          - name: CUSTOM_VAR ⑦
            value: custom_value
        imageLabels:
          - name: url ⑧
            value: https://containers.example.org
          - name: vendor
            value: ExampleCorp Ltd.
        nodeSelector: ⑨
          key1: value1
          key2: value2
        annotations: ⑩
          key1: value1
          key2: value2
        resources: ⑪
          requests:
            cpu: "100m"

```

```
memory: "256Mi"
limits:
  cpu: "100m"
  memory: "256Mi"
```

- 1 Git リポジトリからソースコードのクローンを作成する場合に使用する HTTP プロキシを設定します。
- 2 Git リポジトリからソースコードのクローンを作成する場合に使用する HTTPS プロキシを設定します。
- 3 プロキシを実行しないドメインの一覧を設定します。
- 4 ビルド時に使用する HTTP プロキシを設定するデフォルトの環境変数。この環境変数は、アセンブルおよびビルドの段階で依存関係をダウンロードするために使用できます。
- 5 ビルド時に使用する HTTPS プロキシを設定するデフォルトの環境変数。この環境変数は、アセンブルおよびビルドの段階で依存関係をダウンロードするために使用できます。
- 6 ビルド時にビルドのログレベルを設定するデフォルトの環境変数。
- 7 すべてのビルドに追加されるその他のデフォルトの環境変数。
- 8 すべてのイメージビルドに適用されるラベル。このラベルは **BuildConfig** で上書きできません。
- 9 ビルド Pod は **key1=value2** および **key2=value2** のラベルが付いたノード上でのみ実行されます。ユーザーは、これらの値が無視される場合に、ビルドに対して異なる **nodeSelectors** セットを定義することができます。
- 10 ビルド Pod にはこれらのアノテーションが追加されます。
- 11 **BuildConfig** に関連リソースが定義されていない場合は、デフォルトのリソースをビルド Pod に設定します。

2. 変更を有効にするために、マスターサービスを再起動します。

```
# master-restart api
# master-restart controllers
```

### 31.3. グローバルビルドの上書きの設定

グローバルビルドの上書きは以下の 2 通りの方法で設定できます。

- [Ansible の使用](#)
- [master-config.yaml ファイルを変更して手動で設定する](#)

#### 31.3.1. Ansible を使用したグローバルビルドの上書きの設定

クラスターのインストール時に、以下のパラメーターを使用して、**BuildOverrides** プラグインを設定できます。これらのパラメーターは、インベントリーファイルで設定可能です。

- **openshift\_buildoverrides\_force\_pull**

- `openshift_buildoverrides_image_labels`
- `openshift_buildoverrides_nodeselectors`
- `openshift_buildoverrides_annotations`
- `openshift_buildoverrides_tolerations`

### 例31.2 Ansible を使用したビルドの上書きの設定例

```
# These options configure the BuildOverrides admission controller which injects
# configuration into Builds.
openshift_buildoverrides_force_pull=true
openshift_buildoverrides_image_labels=[{'name':'imagelabelname1','value':'imagelabelvalue1'}]
openshift_buildoverrides_nodeselectors={'nodelabel1':'nodelabelvalue1'}
openshift_buildoverrides_annotations={'annotationkey1':'annotationvalue1'}
openshift_buildoverrides_tolerations=
[{'key':'mykey1','value':'myvalue1','effect':'NoSchedule','operator':'Equal'}]

# Or you may optionally define your own build overrides configuration serialized as json
#openshift_buildoverrides_json='{"BuildOverrides":{"configuration":
{"apiVersion":"v1","kind":"BuildOverridesConfig","forcePull":"true","tolerations":
[{"key":"mykey1","value":"myvalue1","effect":"NoSchedule","operator":"Equal"}]}}'
```



#### 注記

**BuildOverrides** プラグインを使用して容認を上書きするには、**BuildOverrides** ノードセレクターを使用する必要があります。

### 31.3.2. グローバルビルドの上書きの手動設定

**BuildOverrides** プラグインを設定するには、以下の手順を実行します。

1. マスターの `/etc/origin/master/master-config.yaml` ファイルにプラグインの設定を追加します。

```
admissionConfig:
  pluginConfig:
    BuildOverrides:
      configuration:
        apiVersion: v1
        kind: BuildOverridesConfig
        forcePull: true ①
        imageLabels:
          - name: distribution-scope ②
            value: private
        nodeSelector: ③
          key1: value1
          key2: value2
        annotations: ④
          key1: value1
          key2: value2
        tolerations: ⑤
```

```
- key: mykey1
  value: myvalue1
  effect: NoSchedule
  operator: Equal
- key: mykey2
  value: myvalue2
  effect: NoExecute
  operator: Equal
```

- 1 ビルドの開始前に、すべてのビルドがビルダーイメージとソースイメージをプルするよう強制的に実行します。
- 2 すべてのイメージビルドに適用される追加のラベル。ここで定義されたラベルは **BuildConfig** で定義されたラベルよりも優先されます。
- 3 ビルド Pod は **key1=value2** および **key2=value2** のラベルが付いたノード上でのみ実行されます。ユーザーはキー/値のラベルを追加定義して、ビルドが実行されるノードのセットをさらに制限することができます。ただし、ノードには少なくともこれらのラベルを付ける必要があります。
- 4 ビルド Pod にはこれらのアノテーションが追加されます。
- 5 ビルド Pod にある既存の容認はここに一覧されている値で上書きされます。



### 注記

**BuildOverrides** プラグインを使用して容認を上書きするには、**BuildOverrides** ノードセレクターを使用する必要があります。

2. 変更を有効にするために、マスターサービスを再起動します。

```
# master-restart api
# master-restart controllers
```

## 第32章 パイプライン実行の設定

### 32.1. 概要

ユーザーが Pipeline ビルドストラテジーを使用して初めてビルド設定を作成する際に、OpenShift Container Platform は **jenkins-ephemeral** という名前のテンプレートを **openshift** namespace で検索し、ユーザーのプロジェクト内でそれをインスタンス化します。OpenShift Container Platform に同梱される **jenkins-ephemeral** テンプレートは、インスタンス化の実行時に以下を作成します。

- OpenShift Container Platform の公式の Jenkins イメージを使用した Jenkins のデプロイメント設定
- Jenkins デプロイメントにアクセスするためのサービスとルート
- 新規の Jenkins サービスアカウント
- サービスアカウントにプロジェクトへの編集アクセスを付与する RoleBinding

クラスター管理者は、組み込みテンプレートのコンテンツの変更、またはクラスターを異なるテンプレート場所にポイントするためのクラスター設定の編集によって作成されるものを制御できます。

デフォルトテンプレートのコンテンツを変更するには、以下を実行します。

```
$ oc edit template jenkins-ephemeral -n openshift
```

Jenkins 用に永続ストレージを使用する **jenkins-persistent** テンプレートなどの異なるテンプレートを使用するには、以下をマスター設定ファイルに追加します。

```
jenkinsPipelineConfig:  
  autoProvisionEnabled: true ①  
  templateNamespace: openshift ②  
  templateName: jenkins-persistent ③  
  serviceName: jenkins-persistent-svc ④  
  parameters: ⑤  
    key1: value1  
    key2: value2
```

- ① これが指定されていない場合はデフォルトで **true** に設定されます。 **false** が指定されている場合は、いずれのテンプレートもインスタンス化されません。
- ② インスタンス化されるテンプレートが含まれる namespace。
- ③ インスタンス化されるテンプレートの名前。
- ④ インスタンス化の実行時にテンプレートによって作成されるサービスの名前。
- ⑤ インスタンス化の実行中にテンプレートに渡すオプションの値。

Pipeline ビルド設定の作成時に、OpenShift Container Platform は **serviceName** に一致するサービスを検索します。つまり、**serviceName** はプロジェクト内で一意であるように選択される必要があります。サービスが見つからない場合、OpenShift Container Platform は **jenkinsPipelineConfig** テンプレ

レートを実行してインスタンス化します。この方法が適さない場合 (たとえば、OpenShift Container Platform の外部にある Jenkins サーバーを使用する場合)、ユーザーのロールに応じていくつかのことに実行できません。

- クラスター管理者の場合は、単に **autoProvisionEnabled** を **false** に設定します。これにより、クラスター全体で自動プロビジョニングが無効にされます。
- 非特権ユーザーである場合は、OpenShift Container Platform で使用するサービスを作成する必要があります。サービス名は **jenkinsPipelineConfig** の **serviceName** のクラスター設定値と一致している必要があります。デフォルト値は **jenkins** です。プロジェクトの外部で Jenkins サーバーを実行しているために自動プロビジョニングが無効になっている場合は、この新規サービスを既存の Jenkins サーバーにポイントすることが推奨されます。[外部サービスの統合](#) を参照してください。

後者のオプションは、選択したプロジェクト内でのみ自動プロビジョニングを無効にするためにも使用できます。

## 32.2. OPENSIFT JENKINS CLIENT プラグイン

OpenShift Jenkins Client プラグインは、OpenShift API Server との高度な対話を実現するために、読み取り可能かつ簡潔で、包括的で Fluent (流れるような) な Jenkins パイプライン構文を提供することを目的とした Jenkins プラグインです。このプラグインは、スクリプトを実行するノードで使用できる必要がある OpenShift コマンドラインツール (**oc**) を活用します。

プラグインのインストールと設定の詳細については、以下のリンクから公式ドキュメントを参照し、確認してください。

- [インストール](#)
- [OpenShift クラスターの設定](#)
- [認証情報の設定](#)
- [Jenkins ノードの設定](#)



### 注記

このプラグインの使用に関する情報を必要としている開発者の場合は、その場合、[OpenShift Pipeline の概要](#) を参照してください。

## 32.3. OPENSIFT JENKINS の同期プラグイン

この Jenkins プラグインは、OpenShift BuildConfig および Build オブジェクトと Jenkins ジョブおよびビルドとの同期を維持します。

OpenShift Jenkins 同期プラグインは以下を実行します。

- Jenkins での動的なジョブ/実行の作成。
- ImageStreams、ImageStreamTag、または ConfigMap からのスレーブ Pod テンプレートの動的作成。
- 環境変数の挿入。
- OpenShift Web コンソールでの Pipeline の可視化。

- Jenkins Git プラグインとの統合。これにより、OpenShift ビルドから Jenkins Git プラグインにコミット情報が渡されます。

このプラグインの詳細については、以下を参照してください。

- [OpenShift Jenkins 同期プラグイン](#)
- [OpenShift Container Platform 同期プラグイン](#)

## 第33章 ルートのタイムアウトの設定

OpenShift Container Platform のインストールと [ルーターのデプロイ](#) 後、Service Level Availability (SLA) で必要とされるように、低タイムアウトが必要なサービスやバックエンドでの処理速度が遅いケースで高タイムアウトが必要なサービスがある場合は、既存のルートに対してデフォルトのタイムアウトを設定することができます。

**oc annotate** コマンドを使用して、ルートにタイムアウトを追加します。

```
# oc annotate route <route_name> \  
--overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit>
```

たとえば、**myroute** という名前のルートに 2 秒のタイムアウトを設定するには以下のようにします。

```
# oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

サポートされる時間単位は、マイクロ秒 (us)、ミリ秒 (ms)、秒 (s)、分 (m)、時間 (h)、または日 (d) です。

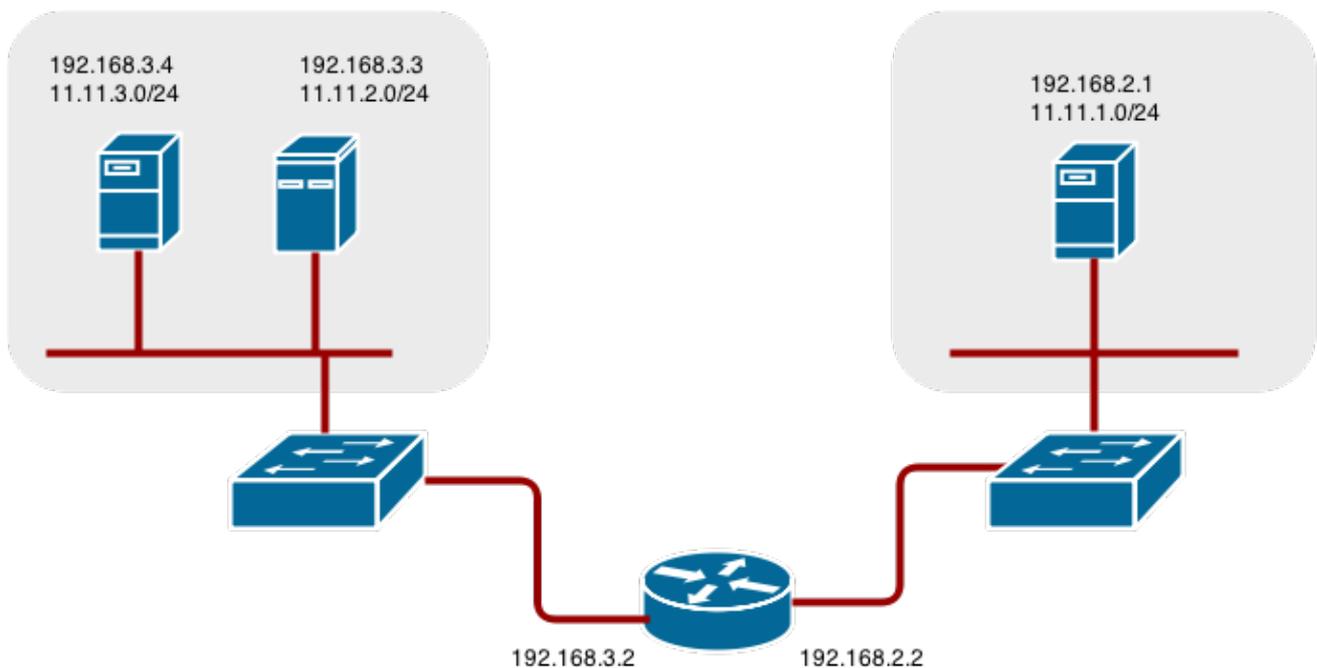
## 第34章 ネイティブのコンテナルーティングの設定

### 34.1. ネットワークの概要

一般的なネットワーク設定を以下に示します。

- 11.11.0.0/16 はコンテナネットワークです。
- 11.11.x.0/24 サブネットは各ノード用に予約済みで、Docker Linux ブリッジに割り当てられています。
- 各ノードには 11.11.0.0/16 の範囲内にある (ローカルサブネットを除く) あらゆるものに到達するために使用するルーターへのルートがあります。
- ルーターには各ノードのルートがあるため、適切なノードにポイントできます。
- ネットワークトポロジィが変更されない限り、新規ノードが追加されても既存のノードを変更する必要はありません。
- IP 転送が各ノードで有効になっています。

以下の図はこのトピックで説明されているコンテナのネットワーク設定を示しています。2つのネットワークインターフェイスカードを搭載したルーターとして機能する1つのLinux ノード、2つのスイッチ、およびこれらのスイッチに接続された3つのノードを使用しています。



### 34.2. ネイティブのコンテナルーティングの設定

既存のスイッチとルーター、Linux のカーネルネットワークスタックを使用してコンテナネットワークを設定できます。

ネットワーク管理者は、新規ノードがクラスターに追加される際に、ルーターを変更するか、ルーターを変更するスクリプトを作成する必要があります。

このプロセスを変化させて、あらゆるタイプのルーターで使用することができます。

### 34.3. コンテナネットワーク用のノードのセットアップ

1. 未使用の 11.11.x.0/24 サブネット IP アドレスをノードの Linux ブリッジに割り当てます。

```
# brctl addbr lbr0
# ip addr add 11.11.1.1/24 dev lbr0
# ip link set dev lbr0 up
```

2. 新規ブリッジを使用するように Docker 起動スクリプトを変更します。デフォルトでは、起動スクリプトは `/etc/sysconfig/docker` ファイルです。

```
# docker -d -b lbr0 --other-options
```

3. 11.11.0.0/16 ネットワークのルーターへのルートを追加します。

```
# ip route add 11.11.0.0/16 via 192.168.2.2 dev p3p1
```

4. ノードで IP 転送を有効にします。

```
# sysctl -w net.ipv4.ip_forward=1
```

### 34.4. コンテナネットワーク用のルーターのセットアップ

以下の手順は、複数の NIC を搭載した Linux ボックスがルーターとして使用されていることを前提としています。必要に応じて手順を変更して、特定のルーターに対応する構文を使用してください。

1. ルーターで IP 転送を有効にします。

```
# sysctl -w net.ipv4.ip_forward=1
```

2. クラスターに追加された各ノードのルートを追加します。

```
# ip route add <node_subnet> via <node_ip_address> dev <interface through which node is
L2 accessible>
# ip route add 11.11.1.0/24 via 192.168.2.1 dev p3p1
# ip route add 11.11.2.0/24 via 192.168.3.3 dev p3p2
# ip route add 11.11.3.0/24 via 192.168.3.4 dev p3p2
```

## 第35章 エッジロードバランサーからのルーティング

### 35.1. 概要

OpenShift Container Platform クラスター内の Pod は、クラスターネットワーク上の IP アドレスを介してのみ到達可能です。エッジロードバランサーを使用することで、ネットワーク外部からのトラフィックを受け取り、OpenShift Container Platform クラスター内の Pod にトラフィックをプロキシすることができます。ロードバランサーがクラスターネットワークに含まれていない場合は、内部クラスターネットワークがエッジロードバランサーからアクセスできないため、ルーティングが困難になります。

OpenShift Container Platform クラスターがクラスターネットワークソリューションとして [OpenShift Container Platform SDN](#) を使用している場合にこの問題を解決するには、Pod へのネットワークアクセスを以下の2つの方法で確保することができます。

### 35.2. ロードバランサーを SDN に含める

可能な場合は、ネットワークプラグインとして OpenShift SDN を使用するロードバランサー自体で OpenShift Container Platform ノードインスタンスを実行します。これにより、エッジマシンは独自の Open vSwitch ブリッジを取得し、このブリッジは、クラスターにある Pod とノードへのアクセスを提供するために SDN によって自動的に設定されます。**ルーティングテーブル** は、Pod が作成および削除される際に SDN によって動的に設定されるため、ルーティングソフトウェアは Pod に到達することができます。

Pod がロードバランサー自体に設定されないように、ロードバランサーマシンに [スケジューラ対象外ノード](#) というマークを付けます。

```
$ oc adm manage-node <load_balancer_hostname> --schedulable=false
```

ロードバランサーがコンテナとしてパッケージ化されている場合は、OpenShift Container Platform との統合が簡単になります。ロードバランサーをホストポートが公開されている状態の Pod として実行します。OpenShift Container Platform で事前にパッケージ化された [HAProxy ルーター](#) はこの方法で実行されます。

### 35.3. RAMP ノードを使用したトンネルの確立

前述のソリューションを使用できない場合もあります。たとえば、F5® は互換性のないカスタム Linux カーネルとディストリビューションを使用するため、F5 BIG-IP® ホストは OpenShift Container Platform ノードインスタンスまたは OpenShift Container Platform SDN を実行できません。

代わりに、F5 BIG-IP® を有効にして Pod に到達できるようにするために、クラスターネットワーク内の既存のノードを **Ramp ノード** として選択し、F5 BIG-IP® ホストと指定された Ramp ノード間でトンネルを確立することができます。Ramp ノードは通常の OpenShift Container Platform ノードであるため、Ramp ノードには、トラフィックをクラスターネットワーク内のノードにある Pod にルーティングするための設定が必要になります。そのため、Ramp ノードは F5 BIG-IP® ホストがクラスターネットワーク全体にアクセスする際に使用するゲートウェイのロールを引き継ぎます。

以下は、F5 BIG-IP® ホストと指定された Ramp ノード間で **ipip** トンネルを確立する例です。

**F5 BIG-IP® ホスト側:**

1. 以下の変数を設定してください。

```
# F5_IP=10.3.89.66 1
```

```
# RAMP_IP=10.3.89.89 ②
# TUNNEL_IP1=10.3.91.216 ③
# CLUSTER_NETWORK=10.128.0.0/14 ④
```

- ① ② **F5\_IP** 変数と **RAMP\_IP** 変数はそれぞれ、共有内部ネットワーク上にある **F5 BIG-IP®** ホストと Ramp ノードの IP アドレスを指しています。
- ③ **ipip** トンネルの **F5®** ホストの終端となる競合しない任意の IP アドレス。
- ④ OpenShift SDN が Pod にアドレスを割り当てるために使用するオーバーレイネットワークの CIDR 範囲。

2. 古い route、self、tunnel および SNAT pool を削除します。

```
# tmsch delete net route $CLUSTER_NETWORK || true
# tmsch delete net self SDN || true
# tmsch delete net tunnels tunnel SDN || true
# tmsch delete ltm snatpool SDN_snatpool || true
```

3. 新規の tunnel、self、route および SNAT pool を作成し、この SNAT pool を仮想サーバーで使用します。

```
# tmsch create net tunnels tunnel SDN \
  \{ description "OpenShift SDN" local-address \
    $F5_IP profile ipip remote-address $RAMP_IP \}
# tmsch create net self SDN \{ address \
  \{$TUNNEL_IP1\}/24 allow-service all vlan SDN \}
# tmsch create net route $CLUSTER_NETWORK interface SDN
# tmsch create ltm snatpool SDN_snatpool members add { $TUNNEL_IP1 }
# tmsch modify ltm virtual ose-vserver source-address-translation { type snat pool
  SDN_snatpool }
# tmsch modify ltm virtual https-ose-vserver source-address-translation { type snat pool
  SDN_snatpool }
```

#### Ramp ノード:



#### 注記

以下で、永続性のない設定が作成されます。つまり、ramp ノードまたは **openvswitch** サービスを再起動すると、この設定はなくなります。

1. 以下の変数を設定してください。

```
# F5_IP=10.3.89.66
# TUNNEL_IP1=10.3.91.216
# TUNNEL_IP2=10.3.91.217 ①
# CLUSTER_NETWORK=10.128.0.0/14 ②
```

- ① **ipip** トンネルの Ramp ノードの終端となる 2 番目の任意の IP アドレス。
- ② OpenShift SDN が Pod にアドレスを割り当てるために使用するオーバーレイネットワークの CIDR 範囲。

- 古いトンネルを削除します。

```
# ip tunnel del tun1 || true
```

- 適切な L2 接続インターフェイス (たとえば、eth0) を使用して、Ramp ノードで **ipip** トンネルを作成します。

```
# ip tunnel add tun1 mode ipip \
  remote $F5_IP dev eth0
# ip addr add $TUNNEL_IP2 dev tun1
# ip link set tun1 up
# ip route add $TUNNEL_IP1 dev tun1
# ping -c 5 $TUNNEL_IP1
```

- SDN サブネットの未使用の IP を使用してトンネル IP を SNAT します。

```
# source /run/openshift-sdn/config.env
# tap1=$(ip -o -4 addr list tun0 | awk '{print $4}' | cut -d/ -f1 | head -n 1)
# subaddr=$(echo ${OPENSIFT_SDN_TAP1_ADDR:-"$tap1"} | cut -d "." -f 1,2,3)
# export RAMP_SDN_IP=${subaddr}.254
```

- この **RAMP\_SDN\_IP** を追加のアドレスとして **tun0** (ローカル SDN のゲートウェイ) に割り当てます。

```
# ip addr add ${RAMP_SDN_IP} dev tun0
```

- SNAT の OVS ルールを変更します。

```
# ipflowopts="cookie=0x999,ip"
# arpflowopts="cookie=0x999, table=0, arp"
#
# ovs-ofctl -O OpenFlow13 add-flow br0 \
"$ipflowopts,nw_src=${TUNNEL_IP1},actions=mod_nw_src:${RAMP_SDN_IP},resubmit(,0)"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
"$ipflowopts,nw_dst=${RAMP_SDN_IP},actions=mod_nw_dst:${TUNNEL_IP1},resubmit(,0)"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
"$arpflowopts, arp_tpa=${RAMP_SDN_IP}, actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
"$arpflowopts, priority=200, in_port=2, arp_spa=${RAMP_SDN_IP},
arp_tpa=${CLUSTER_NETWORK}, actions=goto_table:30"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
"arp, table=5, priority=300, arp_tpa=${RAMP_SDN_IP}, actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0 \
"ip,table=5,priority=300,nw_dst=${RAMP_SDN_IP},actions=output:2"
# ovs-ofctl -O OpenFlow13 add-flow br0
"$ipflowopts,nw_dst=${TUNNEL_IP1},actions=output:2"
```

- 高可用性を実現するように Ramp ノードを設定する予定がない場合は、必要に応じて、Ramp ノードをスケジュール対象外としてマークします。以下のセクションに従う予定や、高可用性を備えた Ramp ノードを作成する予定がある場合は、この手順を省略してください。

```
$ oc adm manage-node <ramp_node_hostname> --schedulable=false
```

### 35.3.1. 高可用性を備えた Ramp ノードの設定

**keepalived** を内部で使用する OpenShift Container Platform の **ipfailover** 機能を使用することで、**F5 BIG-IP®** の観点から Ramp ノードの高可用性を確保することができます。これを実行するには、まず同じ L2 サブネット上の 2 つのノード (たとえば、**ramp-node-1** および **ramp-node-2**) を起動します。

次に、仮想 IP (VIP) を使用するために未割り当ての IP アドレスを同じサブネット内から選択します。この IP アドレスは、**F5 BIG-IP®** でトンネルを設定するとき使用する **RAMP\_IP** 変数として設定されます。

たとえば、Ramp ノードに対して使用するサブネットは **10.20.30.0/24** とし、**10.20.30.2** を **ramp-node-1** に、**10.20.30.3** を **ramp-node-2** に割り当てています。VIP については、同じ **10.20.30.0/24** サブネットから未割り当てのアドレスを選択します (例: **10.20.30.4**)。次に、**ipfailover** を設定するために、両方のノードに **f5ramptime** などのラベルでマークを付けします。

```
$ oc label node ramp-node-1 f5ramptime=true
$ oc label node ramp-node-2 f5ramptime=true
```

**ipfailover** [ドキュメント](#) で説明されているのと同様に、ここでサービスアカウントを作成し、そのアカウントを特権付き SCC に追加する必要があります。最初に、**f5ipfailover** サービスアカウントを作成します。

```
$ oc create serviceaccount f5ipfailover -n default
```

次に、**f5ipfailover** サービスを特権付き SCC に追加できます。**default namespace** の **f5ipfailover** を特権付き SCC に追加するには、以下を実行します。

```
$ oc adm policy add-scc-to-user privileged system:serviceaccount:default:f5ipfailover
```

最後に、選択した VIP (**RAMP\_IP** 変数) と **f5ipfailover** サービスアカウントを使用して、先に設定した **f5ramptime** ラベルを使用して VIP を 2 つのノードに割り当て、**ipfailover** を設定します。

```
# RAMP_IP=10.20.30.4
# IFNAME=eth0 ❶
# oc adm ipfailover <name-tag> \
  --virtual-ips=$RAMP_IP \
  --interface=$IFNAME \
  --watch-port=0 \
  --replicas=2 \
  --service-account=f5ipfailover \
  --selector='f5ramptime=true'
```

❶ **RAMP\_IP** を設定する必要があるインターフェイス。

上記の設定を行うと、現在 VIP が割り当てられている Ramp ノードホストで障害が発生した場合に VIP (**RAMP\_IP** 変数) が自動的に再割り当てされます。

## 第36章 コンテナログの集計

### 36.1. 概要

OpenShift Container Platform のクラスター管理者として、EFK スタックをデプロイして各種の OpenShift Container Platform サービスのログを集計できます。アプリケーション開発者は、表示アクセス権を持つプロジェクトのログを表示することができます。EFK スタックは、複数のコンテナまたは削除された Pod からののものであっても、ホストとアプリケーションからログを集計します。

EFK スタックは [ELK スタック](#) の変更バージョンであり、以下で設定されています。

- [Elasticsearch \(ES\)](#): すべてのログが格納されるオブジェクトストア。
- [Fluentd](#): ノードからログを収集し、そのログを Elasticsearch に送ります。
- [Kibana](#): Elasticsearch の Web UI。

クラスターにデプロイされると、スタックはすべてのノードおよびプロジェクトのログを Elasticsearch に集計し、ログを表示するために Kibana UI を提供します。クラスター管理者はすべてのログを表示できますが、アプリケーション開発者は表示権限を持つプロジェクトのログのみを表示できます。スタックのコンポーネントは安全な通信を実行します。



#### 注記

[Docker コンテナログの管理](#) では、コンテナログを管理し、ノードディスクが満杯になることを阻止するための [json-file](#) ロギングドライバーオプションの使用について説明しています。

### 36.2. デプロイ前の設定

1. Ansible Playbook は集計されたロギングをデプロイおよびアップグレードするために利用できます。[クラスターのインストール](#) ガイドに精通しておくようにしてください。Ansible を使用するための予備知識や、設定に関する情報が記載されています。EFK スタックの各種の領域を設定するために、パラメーターが Ansible インベントリーファイルに追加されています。
2. [サイジングのガイドライン](#) を確認して、デプロイメントの最適な設定方法を判断してください。
3. クラスターのルーターをデプロイしていることを確認します。
4. Elasticsearch に [必要なストレージ](#) があることを確認します。各 Elasticsearch ノードに独自のストレージボリュームが必要であることに注意してください。詳細は、[Elasticsearch](#) を参照してください。
5. [高可用性を備えた Elasticsearch](#) が必要かどうかを判別します。可用性の高い環境には3つ以上の Elasticsearch ノードが必要で、それぞれが別のホストに置かれる必要があります。デフォルトで、OpenShift Container Platform は各インデックスにつき1つのシャードを作成し、それらのシャードのレプリカは作成しません。高可用性を確保するには、各シャードの複数のレプリカも必要です。高可用性を実現するには、[openshift\\_logging\\_es\\_number\\_of\\_replicas](#) Ansible 変数を 1 よりも大きな値に設定します。詳細は、[Elasticsearch](#) を参照してください。

### 36.3. ロギング ANSIBLE 変数の指定

EFK デプロイメントのパラメーターを [インベントリーホストファイル](#) に指定して、[デフォルトパラメーター値](#) を上書きすることができます。

[Elasticsearch](#) および [Fluentd](#) のセクションを参照してパラメーターを選択します。



### 注記

デフォルトでは、Elasticsearch サービスはクラスターのノード間での TCP 通信にポート 9300 を使用します。

パラメーター	説明
<code>openshift_logging_install_logging</code>	ロギングをインストールする場合は <b>true</b> に設定します。ロギングをアンインストールする場合は <b>false</b> に設定します。 <b>true</b> に設定される場合、 <code>openshift_logging_es_nodeselector</code> を使用してノードセレクターを指定する必要があります。
<code>openshift_logging_use_ops</code>	<b>true</b> に設定されている場合、操作ログに対して 2 番目の Elasticsearch クラスターと Kibana を設定します。Fluentd はメインクラスターと操作ログ用に予約されているクラスター ( <code>default</code> 、 <code>openshift</code> 、 <code>openshift-infra</code> の各プロジェクトのログ、Docker、OpenShift、およびジャーナルのシステムログで設定される) の間でログを分割します。これにより、2 番目の Elasticsearch クラスターと Kibana がデプロイされます。デプロイメントは名前に含まれる <code>-ops</code> 接尾辞で見分けることができ、以下に一覧表示されている並列デプロイメントオプションがあります。並列デプロイメントオプションについては、 <a href="#">Curator 設定の作成</a> で説明されています。 <b>true</b> に設定すると、 <code>openshift_logging_es_ops_nodeselector</code> は必須です。
<code>openshift_logging_master_url</code>	Kubernetes マスターの URL。これは公開されている必要はありませんが、クラスター内からアクセスできる必要があります(例: <a href="https://&lt;PRIVATE-MASTER-URL&gt;:8443">https://&lt;PRIVATE-MASTER-URL&gt;:8443</a> )。
<code>openshift_logging_purge_logging</code>	通常のアインストールでは、再インストール中の予期せぬデータ損失を防ぐために PVC が維持されます。Ansible Playbook が完全かつ不可逆的にすべてのロギング永続データ (PVC を含む) を確実に削除するようにするには、 <code>openshift_logging_install_logging</code> を <b>false</b> に設定してアインストールをトリガーし、 <code>openshift_logging_purge_logging</code> を <b>true</b> に設定します。デフォルトは <b>false</b> に設定されます。
<code>openshift_logging_install_eventrouter</code>	<code>openshift_logging_install_logging</code> と併用します。いずれも <b>true</b> に設定されると、eventrouter がインストールされます。いずれも <b>false</b> の場合、eventrouter はアンインストールされます。
<code>openshift_logging_eventrouter_image</code>	Eventrouter のイメージバージョン。例: <code>registry.redhat.io/openshift3/ose-logging-eventrouter:v3.11</code>
<code>openshift_logging_eventrouter_image_version</code>	ロギング eventrouter のイメージバージョン
<code>openshift_logging_eventrouter_sink</code>	eventrouter のシンク (受信側) を選択します。 <b>stdout</b> と <b>glog</b> がサポートされています。デフォルトでは <b>stdout</b> に設定されています。

パラメーター	説明
<code>openshift_logging_eventrouter_nodeselector</code>	Pod が到達するノードを選択するためのラベルのマッピング (" <code>node</code> ":" <code>infra</code> "、" <code>region</code> ":" <code>west</code> " など)。
<code>openshift_logging_eventrouter_replicas</code>	デフォルトは <b>1</b> に設定されます。
<code>openshift_logging_eventrouter_cpu_limit</code>	eventrouter に割り当てる最小 CPU 量。デフォルトでは <b>100m</b> に設定されます。
<code>openshift_logging_eventrouter_memory_limit</code>	eventrouter Pod のメモリー制限。デフォルトでは <b>128Mi</b> に設定されます。
<code>openshift_logging_eventrouter_namespace</code>	eventrouter がデプロイされる project。デフォルトでは <b>default</b> に設定されます。  <div data-bbox="555 817 662 1102" style="display: inline-block; vertical-align: top; width: 40px; height: 127px; background: repeating-linear-gradient(45deg, transparent, transparent 2px, black 2px, black 4px);"></div> <div data-bbox="742 817 805 855" style="display: inline-block; vertical-align: top; margin-left: 10px;"><b>重要</b></div> <div data-bbox="742 884 1428 1102" style="display: inline-block; vertical-align: top; margin-left: 10px;">プロジェクトを <b>default</b> または <b>openshift-*</b> 以外に設定しないでください。異なるプロジェクトを指定する場合、他のプロジェクトからのイベント情報が運用ユーザーに制限されていないインデックスにリークされる可能性があります。デフォルト以外のプロジェクトを使用するには、<b>oc new-project</b> を使用して通常の方法でプロジェクトを作成します。</div>
<code>openshift_logging_image_pull_secret</code>	認証済みレジストリーからコンポーネントイメージをプルするために使用される既存のプルシークレットの名前を指定します。
<code>openshift_logging_curator_image</code>	Curator のイメージバージョン。例: <b>registry.redhat.io/openshift3/ose-logging-curator5:v3.11</b>
<code>openshift_logging_curator_default_days</code>	ログレコードを削除するために Curator が使用するデフォルトの最小期間 (日単位)。
<code>openshift_logging_curator_run_hour</code>	Curator が実行される時刻 (時間)。
<code>openshift_logging_curator_run_minute</code>	Curator が実行される時刻 (分)。
<code>openshift_logging_curator_run_timezone</code>	実行時間を割り出すために Curator が使用するタイムゾーン。タイムゾーンは、 <b>America/New_York</b> または <b>UTC</b> など、 <code>tzselect(8)</code> または <code>timedatectl(1)</code> の "リージョン/ローカリティー" 形式の文字列で指定します。
<code>openshift_logging_curator_script_log_level</code>	Curator のスクリプトログレベル。

パラメーター	説明
<code>openshift_logging_curator_log_level</code>	Curator プロセスのログレベル。
<code>openshift_logging_curator_cpu_limit</code>	Curator に割り当てる CPU 量。
<code>openshift_logging_curator_memory_limit</code>	Curator に割り当てるメモリー量。
<code>openshift_logging_curator_nodeselector</code>	Curator インスタンスをデプロイできるターゲットのノードを指定するノードセレクター。
<code>openshift_logging_curator_ops_cpu_limit</code>	<code>openshift_logging_use_ops</code> が <code>true</code> に設定されている場合は、Ops クラスターの <code>openshift_logging_curator_cpu_limit</code> と同等です。
<code>openshift_logging_curator_ops_memory_limit</code>	<code>openshift_logging_use_ops</code> が <code>true</code> に設定されている場合は、Ops クラスターの <code>openshift_logging_curator_memory_limit</code> と同等です。
<code>openshift_logging_curator_replace_configmap</code>	アップグレードで <code>logging-curator</code> ConfigMap を置き換えられないように <code>no</code> に設定します。ConfigMap を上書きできるようにするには、 <code>yes</code> に設定します。
<code>openshift_logging_kibana_image</code>	Kibana のイメージバージョン。例: <code>registry.redhat.io/openshift3/ose-logging-kibana5:v3.11</code>
<code>openshift_logging_kibana_hostname</code>	Kibana に到達する Web クライアントの外部ホスト名。
<code>openshift_logging_kibana_cpu_limit</code>	Kibana に割り当てる CPU 量。
<code>openshift_logging_kibana_memory_limit</code>	Kibana に割り当てるメモリー量。
<code>openshift_logging_kibana_proxy_image</code>	Kibana プロキシのイメージバージョン。例: <code>registry.redhat.io/openshift3/oauth-proxy:v3.11</code>
<code>openshift_logging_kibana_proxy_debug</code>	<code>true</code> の場合、Kibana プロキシのログレベルを <code>DEBUG</code> に設定します。
<code>openshift_logging_kibana_proxy_cpu_limit</code>	Kibana プロキシに割り当てる CPU 量。
<code>openshift_logging_kibana_proxy_memory_limit</code>	Kibana プロキシに割り当てるメモリー量。

パラメーター	説明
<b>openshift_logging_kibana_replica_count</b>	Kibana を拡張するノード数。
<b>openshift_logging_kibana_nodeselector</b>	Kibana インスタンスをデプロイできるターゲットのノードを指定するノードセレクター。
<b>openshift_logging_kibana_env_vars</b>	Kibana デプロイメント設定に追加する環境変数のマップ。例: {"ELASTICSEARCH_REQUESTTIMEOUT":"30000"}
<b>openshift_logging_kibana_key</b>	Kibana ルートの作成時に使用する公開されているキー。
<b>openshift_logging_kibana_cert</b>	Kibana ルートの作成時にキーに一致する証明書。
<b>openshift_logging_kibana_ca</b>	オプション。キーに添付する CA と、Kibana ルートの作成時に使用される証明書。
<b>openshift_logging_kibana_ops_hostname</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_kibana_hostname</b> と同等です。
<b>openshift_logging_kibana_ops_cpu_limit</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_kibana_cpu_limit</b> と同等です。
<b>openshift_logging_kibana_ops_memory_limit</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_kibana_memory_limit</b> と同等です。
<b>openshift_logging_kibana_ops_proxy_debug</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_kibana_proxy_debug</b> と同等です。
<b>openshift_logging_kibana_ops_proxy_cpu_limit</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_kibana_proxy_cpu_limit</b> と同等です。
<b>openshift_logging_kibana_ops_proxy_memory_limit</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_kibana_proxy_memory_limit</b> と同等です。
<b>openshift_logging_kibana_ops_replica_count</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_kibana_replica_count</b> と同等です。
<b>openshift_logging_es_allow_external</b>	<b>true</b> に設定すると、Elasticsearch が re-encrypt ルートとして公開されます。デフォルトでは <b>false</b> に設定されます。

パラメーター	説明
<b>openshift_logging_es_hostname</b>	<p>ルートと TLS サーバーの証明書に使用する外部向けホスト名。デフォルトでは <b>es</b> に設定されます。</p> <p>たとえば、<b>openshift_master_default_subdomain</b> が <b>=example.test</b> に設定されている場合、<b>openshift_logging_es_hostname</b> のデフォルト値は <b>es.example.test</b> になります。</p>
<b>openshift_logging_es_cert</b>	Elasticsearch が外部 TLS サーバー証明書に使用する証明書の場所。デフォルトは、生成される証明書になります。
<b>openshift_logging_es_key</b>	Elasticsearch が外部 TLS サーバー証明書に使用するキーの場所。デフォルトは、生成されるキーになります。
<b>openshift_logging_es_ca_ext</b>	Elasticsearch が外部 TLS サーバーに使用する CA 証明書の場所。デフォルトは内部 CA です。
<b>openshift_logging_es_ops_allow_external</b>	<b>true</b> に設定すると、Elasticsearch が re-encrypt ルートとして公開されます。デフォルトでは <b>false</b> に設定されます。
<b>openshift_logging_es_ops_hostname</b>	<p>ルートと TLS サーバーの証明書に使用する外部向けホスト名。デフォルトでは <b>es-ops</b> に設定されます。</p> <p>たとえば、<b>openshift_master_default_subdomain</b> が <b>=example.test</b> に設定されている場合、<b>openshift_logging_es_ops_hostname</b> のデフォルト値は <b>es-ops.example.test</b> になります。</p>
<b>openshift_logging_es_ops_cert</b>	Elasticsearch が外部 TLS サーバー証明書に使用する証明書の場所。デフォルトは、生成される証明書になります。
<b>openshift_logging_es_ops_key</b>	Elasticsearch が外部 TLS サーバー証明書に使用するキーの場所。デフォルトは、生成されるキーになります。
<b>openshift_logging_es_ops_ca_ext</b>	Elasticsearch が外部 TLS サーバーに使用する CA 証明書の場所。デフォルトは内部 CA です。
<b>openshift_logging_fluentd_image</b>	Fluentd のイメージバージョン。例: <b>registry.redhat.io/openshift3/ose-logging-fluentd:v3.11</b>
<b>openshift_logging_fluentd_nodeselector</b>	<p>Fluentd インスタンスをデプロイできるターゲットのノードを指定するノードセクター。Fluentd を実行する必要があるノード (通常はすべて) には、Fluentd が実行およびログ収集できるようになる前に、このラベルを付ける必要があります。</p> <p>インストール後に Aggregated Logging クラスターを拡張する際に、<b>openshift_logging</b> ロールがこのノードセクターを使用して <b>openshift_logging_fluentd_hosts</b> によって提供されるノードにラベルを付けます。</p> <p>インストールの一環として、Fluentd ノードセクターのラベルを永続化された <b>ノードラベル</b> の一覧に追加することをお勧めします。</p>

パラメーター	説明
<b>openshift_logging_fluentd_cpu_limit</b>	Fluentd Pod の CPU 上限。
<b>openshift_logging_fluentd_memory_limit</b>	Fluentd Pod のメモリー上限。
<b>openshift_logging_fluentd_journal_read_from_head</b>	最初の起動時に Fluentd がジャーナルの先頭から読み取る必要がある場合は、 <b>true</b> に設定します。これにより、現行のログレコードを受信する Elasticsearch で遅延が発生する可能性があります。
<b>openshift_logging_fluentd_hosts</b>	Fluentd をデプロイするためにラベルを付ける必要があるノードの一覧。デフォルトでは、全ノードに ['--all'] のラベルをつけます。null 値は <b>openshift_logging_fluentd_hosts={}</b> です。Fluentd Pod を起動するには、DaemonSet の <b>nodeSelector</b> を有効なラベルに更新します。例: ['host1.example.com', 'host2.example.com']
<b>openshift_logging_fluentd_audit_container_engine</b>	<b>openshift_logging_fluentd_audit_container_engine</b> が <b>true</b> に設定されている場合には、コンテナエンジンの監査ログが収集され、ES に保存されます。この変数を有効にすると、EFK が指定の監査ログファイルか、デフォルトの <b>/var/log/audit.log</b> ファイルを監視することができ、プラットフォームのコンテナエンジンに関する監査情報を収集して、Kibana に配置します。
<b>openshift_logging_fluentd_audit_file</b>	監査ログファイルの場所。デフォルトは <b>/var/log/audit/audit.log</b> です。この変数を有効にすると、EFK が指定の監査ログファイルか、デフォルトの <b>/var/log/audit.log</b> ファイルを監視することができ、プラットフォームのコンテナエンジンに関する監査情報を収集して、Kibana に配置します。
<b>openshift_logging_fluentd_audit_pos_file</b>	監査ログファイルの Fluentd <b>in_tail</b> の位置ファイルが配置される場所。デフォルトは <b>/var/log/audit/audit.log</b> です。この変数を有効にすると、EFK が指定の監査ログファイルか、デフォルトの <b>/var/log/audit.log</b> ファイルを監視することができ、プラットフォームのコンテナエンジンに関する監査情報を収集して、Kibana に配置します。

パラメーター	説明
<code>openshift_logging_fluentd_merge_json_log</code>	レコードの <b>ログ</b> または <b>MESSAGE</b> フィールドに埋め込まれた JSON ログの処理を有効にするには <b>true</b> に設定します。デフォルトは <b>true</b> です。
<code>openshift_logging_fluentd_extra_keep_fields</code>	<code>openshift_logging_fluentd_merge_json_log</code> を使用する場合に生成される追加フィールドの処理時に変更しないフィールドのコンマ区切りリストを指定します。それ以外の場合に、Fluentd は以下の他の未定義のフィールドの設定に合わせてフィールドを処理します。デフォルトは空です。
<code>openshift_logging_fluentd_keep_empty_fields</code>	<code>openshift_logging_fluentd_merge_json_log</code> を使用する場合に空のフィールドとして保持するコンマ区切りのフィールドの一覧を指定します。デフォルトでは、Fluentd は <b>message</b> フィールド以外、レコードから空のフィールドを削除します。
<code>openshift_logging_fluentd_replace_configmap</code>	アップグレードで <b>logging-fluentd</b> ConfigMap を置き換えないようにするには、 <b>no</b> に設定します。ConfigMap を上書きできるようにするには、 <b>yes</b> に設定します。
<code>openshift_logging_fluentd_use_undefined</code>	<code>openshift_logging_fluentd_merge_json_log</code> で生成されたフィールドを <code>openshift_logging_fluentd_undefined_name</code> パラメーターで指定されたサブフィールドに移動するには、 <b>true</b> に設定します。デフォルトで、Fluentd はこれらのフィールドをレコードの最上位に保持するので、Elasticsearch の競合およびスキーマエラーが生じる可能性があります。
<code>openshift_logging_fluentd_undefined_name</code>	<code>openshift_logging_fluentd_use_undefined</code> の使用時に、未定義のフィールドを移動するフィールドの名前を指定します。デフォルトは <b>undefined</b> です。
<code>openshift_logging_fluentd_undefined_to_string</code>	<code>openshift_logging_fluentd_merge_json_log</code> の使用時に、未定義のフィールド値を JSON 文字列表現に変換するには、 <b>true</b> に設定します。デフォルトは <b>false</b> です。
<code>openshift_logging_fluentd_undefined_dot_replace_char</code>	<code>openshift_logging_fluentd_merge_json_log</code> を使用する場合に、 <code>_</code> などの文字を指定して、フィールド内の <code>.</code> 文字を置き換えます。未定義フィールドで名前に <code>.</code> の文字が含まれると、Elasticsearch で問題が発生します。デフォルトは <b>UNUSED</b> で、フィールド名の <code>.</code> は保持されます。
<code>openshift_logging_fluentd_undefined_max_num_fields</code>	<code>openshift_logging_fluentd_merge_json_log</code> を使用する場合に、未定義のフィールドの数に制限を指定します。ログには数百の未定義フィールドが含まれる可能性があるため Elasticsearch で問題が発生します。指定された数以上のフィールドがある場合に、フィールドは JSON ハッシュ文字列に変換され、 <code>openshift_logging_fluentd_undefined_name</code> フィールドに保存されます。デフォルト値は <b>-1</b> で、これは無制限のフィールドを意味します。

パラメーター	説明
<code>openshift_logging_fluentd_use_multiline_json</code>	<code>openshift_logging_fluentd_merge_json_log</code> を使用する場合に、Fluentd のログで分割されている行を 1 行に強制的に再構築するように <b>true</b> に設定します。 <b>json-file</b> ドライバーにより、Docker はログ行を 16k バイトのサイズで分割します。デフォルトは <b>false</b> です。
<code>openshift_logging_fluentd_use_multiline_journal</code>	<code>openshift_logging_fluentd_merge_json_log</code> を使用する場合に、Fluentd のログで分割されている行を 1 行に強制的に再構築するように <b>true</b> に設定します。 <b>journald</b> ドライバーを使用して、Docker はログ行を 16k バイトのサイズに分割します。デフォルトは <b>false</b> です。
<code>openshift_logging_es_host</code>	Fluentd がログを送信する Elasticsearch サービスの名前。
<code>openshift_logging_es_port</code>	Fluentd がログを送信する Elasticsearch サービスのポート。
<code>openshift_logging_es_ca</code>	Fluentd が <code>openshift_logging_es_host</code> との通信に使用する CA の場所。
<code>openshift_logging_es_client_cert</code>	Fluentd が <code>openshift_logging_es_host</code> に使用するクライアント証明書 の場所。
<code>openshift_logging_es_client_key</code>	Fluentd が <code>openshift_logging_es_host</code> に使用するクライアントキーの場所。
<code>openshift_logging_es_cluster_size</code>	デプロイする Elasticsearch ノード。高可用性には 3 つ以上が必要です。
<code>openshift_logging_es_cpu_limit</code>	Elasticsearch クラスターの CPU 量の上限。
<code>openshift_logging_es_memory_limit</code>	Elasticsearch インスタンスごとに予約する RAM 容量。512 M 以上である必要があります。使用可能な接尾辞は G、g、M、m です。
<code>openshift_logging_es_number_of_replicas</code>	それぞれの新規インデックスのプライマリーシャードごとのレプリカ数。デフォルトは '0' に設定されます。実稼働環境のクラスターに対しては、1 以上を設定することが推奨されます。可用性の高い環境の場合は、この値を 1 以上に設定し、3 つ以上の Elasticsearch ノードを各ホストに 1 つずつ設定します。レプリカの数を変更すると、新しい値が新しいインデックスにのみ適用されます。新しい値は既存のインデックスには適用されません。既存のインデックスのレプリカ数を変更する方法については、 <a href="#">Elasticsearch Replicas の数の変更</a> を参照してください。
<code>openshift_logging_es_number_of_shards</code>	ES で作成される新規インデックスごとのプライマリーシャード数。デフォルトは 1 です。
<code>openshift_logging_es_pv_selector</code>	特定の PV を選択するために PVC に追加されるキー/値のマッピング。

パラメーター	説明
<b>openshift_logging_es_pvc_dynamic</b>	<p>バックストレージを動的にプロビジョニングするには、パラメーターの値を <b>true</b> に設定します。 <b>true</b> に設定される場合、storageClass 仕様が PVC 定義から省略されます。 <b>false</b> に設定される場合、<b>openshift_logging_es_pvc_size</b> パラメーターの値を指定する必要があります。</p> <p><b>openshift_logging_es_pvc_storage_class_name</b> パラメーターの値を設定する場合、その値は <b>openshift_logging_es_pvc_dynamic</b> パラメーターの値を上書きします。</p>
<b>openshift_logging_es_pvc_storage_class_name</b>	<p>デフォルト以外のストレージクラスを使用するには、<b>glusterprovisioner</b> または <b>cephrbdprovisioner</b> などのストレージクラス名を指定します。ストレージクラス名を指定した後は、<b>openshift_logging_es_pvc_dynamic</b> の値が何であっても、動的ボリュームプロビジョニングがアクティブになります。</p>
<b>openshift_logging_es_pvc_size</b>	<p>Elasticsearch インスタンスごとに作成する Persistent Volume Claim (永続ボリューム要求、PVC) のサイズ。たとえば、100G です。省略される場合、PVC が作成されず、代わりに一時ボリュームが使用されます。このパラメーターを設定すると、ロギングインストーラーは <b>openshift_logging_elasticsearch_storage_type</b> を <b>pvc</b> に設定します。</p> <p><b>openshift_logging_es_pvc_dynamic</b> パラメーターが <b>false</b> に設定されている場合、このパラメーターの値を設定する必要があります。詳細は、<b>openshift_logging_es_pvc_prefix</b> の説明を参照してください。</p>
<b>openshift_logging_elasticsearch_image</b>	<p>Elasticsearch のイメージバージョン。例: <b>registry.redhat.io/openshift3/ose-logging-elasticsearch5:v3.11</b></p>
<b>openshift_logging_elasticsearch_storage_type</b>	<p>Elasticsearch ストレージタイプを設定します。永続 Elasticsearch ストレージを使用している場合、ロギングインストーラーはこれを <b>pvc</b> に設定します。</p>
<b>openshift_logging_es_pvc_prefix</b>	<p>Elasticsearch ノードのストレージとして使用される Persistent Volume Claim (永続ボリューム要求、PVC) の名前の接頭辞。logging-es-1 のように、ノードごとに数字が付加されます。まだ存在していない場合は、サイズ <b>es-pvc-size</b> を使用して作成されます。</p> <p><b>openshift_logging_es_pvc_prefix</b> が設定されている場合:</p> <ul style="list-style-type: none"> <li>● <b>openshift_logging_es_pvc_dynamic=true</b> である と、<b>openshift_logging_es_pvc_size</b> の値はオプションになります。</li> <li>● <b>openshift_logging_es_pvc_dynamic=false</b> である と、<b>openshift_logging_es_pvc_size</b> の値を設定する必要があります。</li> </ul>
<b>openshift_logging_es_rec_over_after_time</b>	<p>リカバリーを試みる前に Elasticsearch が待機する時間。サポート対象の時間単位は、秒 (s) または分 (m) です。</p>

パラメーター	説明
<b>openshift_logging_es_storage_group</b>	Elasticsearch ストレージボリュームにアクセスするための補助グループ ID の数。バックアップボリュームはこのグループ ID によるアクセスを許可する必要があります。
<b>openshift_logging_es_no_deselector</b>	Elasticsearch ノードをデプロイできるターゲットのノードを判断するマップとして指定されているノードセクター。このマップは、Elasticsearch インスタンスの実行用に予約または最適化されているノードにこれらのインスタンスを配置するために使用することができます。たとえば、セクターは <code>{"node-role.kubernetes.io/infra":"true"}</code> と指定できます。Elasticsearch をデプロイする前に、少なくとも1つのアクティブノードにこのラベルを付ける必要があります。このパラメーターは、ロギングをインストールする場合に必須です。
<b>openshift_logging_es_ops_host</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_host</b> と同等です。
<b>openshift_logging_es_ops_port</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_port</b> と同等です。
<b>openshift_logging_es_ops_ca</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_ca</b> と同等です。
<b>openshift_logging_es_ops_client_cert</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_client_cert</b> と同等です。
<b>openshift_logging_es_ops_client_key</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_client_key</b> と同等です。
<b>openshift_logging_es_ops_cluster_size</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_cluster_size</b> と同等です。
<b>openshift_logging_es_ops_cpu_limit</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_cpu_limit</b> と同等です。
<b>openshift_logging_es_ops_memory_limit</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_memory_limit</b> と同等です。
<b>openshift_logging_es_ops_pv_selector</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_pv_selector</b> と同等です。
<b>openshift_logging_es_ops_pvc_dynamic</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_pvc_dynamic</b> と同等です。
<b>openshift_logging_es_ops_pvc_size</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_pvc_size</b> と同等です。
<b>openshift_logging_es_ops_pvc_prefix</b>	<b>openshift_logging_use_ops</b> が <b>true</b> に設定されている場合は、Ops クラスターの <b>openshift_logging_es_pvc_prefix</b> と同等です。

パラメーター	説明
<code>openshift_logging_es_ops_storage_group</code>	<code>openshift_logging_use_ops</code> が <code>true</code> に設定されている場合は、Ops クラスターの <code>openshift_logging_es_storage_group</code> と同等です。
<code>openshift_logging_es_ops_nodeselector</code>	Elasticsearch ノードをデプロイできるターゲットのノードを指定するノードセレクター。このノードセレクターは、Elasticsearch インスタンスの実行用に予約または最適化されているノード上にこれらのインスタンスを配置するために使用できます。たとえば、セレクターは <code>node-type=infrastructure</code> と指定できます。Elasticsearch をデプロイする前に、少なくとも1つのアクティブノードにこのラベルを付ける必要があります。このパラメーターは、 <code>openshift_logging_use_ops</code> が <code>true</code> に設定されている場合に必要です。
<code>openshift_logging_elasticsearch_kibana_index_mode</code>	<p>デフォルト値 <code>unique</code> では、各ユーザーが独自の Kibana インデックスを持つことができます。このモードでは、保存されたクエリー、ビジュアライゼーション、およびダッシュボードは共有されません。</p> <p>値 <code>shared_ops</code> を設定することもできます。このモードでは、すべての操作ユーザーが Kibana インデックスを共有します。これにより、各操作ユーザーが同じクエリー、ビジュアライゼーション、およびダッシュボードを参照することができます。操作ユーザーであるかどうかを判別するには、以下を実行します。</p> <pre>#oc auth can-i view pod/logs -n default yes</pre> <p>適切なアクセスがない場合は、クラスター管理者にお問い合わせください。</p>
<code>openshift_logging_elasticsearch_poll_timeout_minutes</code>	Ansible Playbook が Elasticsearch クラスターが指定された Elasticsearch ノードのアップグレード後に Green の状態になるまで待機する時間を調整します。50 GB 以上の大規模なシャードは初期化までに 60 分を超える時間がかかる場合があり、これにより、Ansible Playbook がアップグレード手順を中止する可能性があります。デフォルトは <b>60</b> です。
<code>openshift_logging_kibana_ops_nodeselector</code>	Kibana インスタンスをデプロイできるターゲットのノードを指定するノードセレクター。
<code>openshift_logging_curator_ops_nodeselector</code>	Curator インスタンスをデプロイできるターゲットのノードを指定するノードセレクター。
<code>openshift_logging_elasticsearch_replace_configmap</code>	<code>logging-elasticsearch</code> ConfigMap を現在のデフォルト値に置き換えるには、 <code>true</code> に設定します。現在の ConfigMap は <code>logging-elasticsearch.old</code> に保存されます。これを使用して、カスタマイズ内容を新規 ConfigMap にコピーできます。場合によっては、古い ConfigMap を使用するとアップグレードが失敗する場合があります。デフォルトは <code>false</code> に設定されます。

## Custom Certificates

デプロイメントプロセスで生成されるものを利用する代わりに、以下のインベントリー変数を使用して

カスタム証明書を指定できます。カスタム証明書は、ユーザーのブラウザと Kibana 間の通信を暗号化し、保護するために使用されます。これらが指定されない場合は、セキュリティー関連のファイルが生成されます。

ファイル名	説明
<code>openshift_logging_kibana_cert</code>	Kibana サーバーのブラウザ向けの証明書。
<code>openshift_logging_kibana_key</code>	ブラウザ向けの Kibana 証明書で使用されるキー。
<code>openshift_logging_kibana_ca</code>	ブラウザ向けの Kibana 証明書用に使用される CA ファイルへの制御ノード上の絶対パス。
<code>openshift_logging_kibana_ops_cert</code>	Ops Kibana サーバーのブラウザ向け証明書。
<code>openshift_logging_kibana_ops_key</code>	ブラウザ向けの Ops Kibana 証明書で使用されるキー。
<code>openshift_logging_kibana_ops_ca</code>	ブラウザ向けの Ops Kibana 証明書用に使用される CA ファイルへの制御ノード上の絶対パス。

これらの証明書を再デプロイする必要がある場合は、[EFK 証明書の再デプロイ](#) を参照してください。

## 36.4. EFK スタックのデプロイ

EFK スタックは Ansible Playbook を使用して EFK コンポーネントにデプロイされます。デフォルトの [インベントリ](#) ファイルを使用して、デフォルトの OpenShift Ansible の場所から Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i <path/to/inventory>] \
  playbooks/openshift-logging/config.yml
```

Playbook を実行すると、スタックをサポートするために必要なすべてのリソース (シークレット、ServiceAccount、DeploymentConfig など) がプロジェクト **openshift-logging** にデプロイされます。Playbook はスタックが実行されるまでコンポーネント Pod のデプロイを待機します。待機手順が失敗しても、デプロイメントは成功する可能性があり、レジストリーからコンポーネントイメージを取得できる場合があります。これには数分の時間がかかる可能性があります。以下を使用してプロセスを確認できます。

```
$ oc get pods -w
```

logging-curator-1541129400-l5h77	0/1	Running	0	11h	1
logging-es-data-master-ecu30lr4-1-deploy	0/1	Running	0	11h	2
logging-fluentd-2lgwn	1/1	Running	0	11h	3
logging-fluentd-lmvms	1/1	Running	0	11h	
logging-fluentd-p9nd7	1/1	Running	0	11h	
logging-kibana-1-zk94k	2/2	Running	0	11h	4

- 
- 1 Curator Pod。Curator には1つの Pod のみが必要です。
- 2 このホストの Elasticsearch Pod。
- 3 Fluentd Pod。クラスター内の各ノードに1つの Pod があります。
- 4 Kibana Pod。

`oc get pods -o wide` コマンドを使用して Fluentd Pod がデプロイされているノードを確認できます。

```
$ oc get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE
NOMINATED NODE
logging-es-data-master-5av030lk-1-2x494  2/2     Running   0          38m   154.128.0.80   ip-153-12-8-6.wef.internal <none>
logging-fluentd-lqdxg                    1/1     Running   0          2m    154.128.0.85   ip-153-12-8-6.wef.internal <none>
logging-kibana-1-gj5kc                    2/2     Running   0          39m   154.128.0.77   ip-153-12-8-6.wef.internal <none>
```

最終的に Pod のステータスは **Running** になります。関連イベントの取得によるデプロイメント時の Pod のステータスを確認するには、以下を実行します。

```
$ oc describe pods/<pod_name>
```

Pod の実行に失敗したら、ログを確認してください。

```
$ oc logs -f <pod_name>
```

## 36.5. デプロイメントの概要と調整

このセクションでは、デプロイされたコンポーネントに対して行うことができる調整について説明します。

### 36.5.1. Ops クラスター



#### 注記

**default**、**openshift**、および **openshift-infra** プロジェクトのログが自動的に集計され、Kibana インターフェイスで **.operations** 項目にグループ化されます。

EFK スタックをデプロイしたプロジェクト (ここでは **logging**) は **.operations** に集計されず、その ID の下に表示されます。

インベントリーファイルで **openshift\_logging\_use\_ops** を **true** に設定した場合、Fluentd はメインの Elasticsearch クラスターと操作ログ用に予約された別のクラスター間でログを分割するように設定されます。この操作ログはノードシステムログおよびプロジェクト **default**、**openshift**、**openshift-infra** として定義されています。したがって、操作ログのインデックス化、アクセス、管理を行うために個別の Elasticsearch クラスター、個別の Kibana、および個別の Curator がデプロイされます。これらのデ

プロイメントは、**-ops** が含まれる名前で区別します。このオプションを有効にする場合は、これらの個別のデプロイメントに留意してください。**-ops** が含まれるという名前の変更はありますが、以下の説明の大部分が操作クラスターにも適用されます (存在する場合)。

## 36.5.2. Elasticsearch

**Elasticsearch (ES)**: すべてのログが格納されるオブジェクトストア。

Elasticsearch はログデータを、**インデックス** と呼ばれる各データストアに編成します。Elasticsearch は各インデックスを **シャード** と呼ばれる複数の部分に分割し、クラスター内の Elasticsearch ノードセット全体に分散します。Elasticsearch を **レプリカ** というシャードのコピーを作成するように設定できます。Elasticsearch はレプリカを Elasticsearch ノード全体に分散することもできます。シャードとレプリカの組み合わせは、冗長性および耐障害性を提供することを意図しています。たとえば、1つのレプリカのあるインデックスの3つのシャードを設定する場合、Elasticsearch はそのインデックスの合計6つのシャード (3つのプライマリーシャードとバックアップとしての3つのレプリカ) を生成します。

OpenShift Container Platform ロギングインストーラーは、独自のストレージボリュームを含む固有のデプロイメント設定を使用して各 Elasticsearch ノードがデプロイされるようにします。ロギングシステムに追加する各 Elasticsearch ノードについて [追加のデプロイメント設定を作成](#) できます。インストール時に、**openshift\_logging\_es\_cluster\_size** Ansible 変数を使用して Elasticsearch ノードの数を指定できます。

または、インベントリーファイルで **openshift\_logging\_es\_cluster\_size** を変更し、ロギング Playbook を再実行することで、既存クラスターを拡張することができます。追加のクラスターリングパラメーターは変更可能で、[ロギング Ansible 変数の指定](#) で説明されています。

以下に示すように、ストレージおよびネットワークの場所を選択する際の留意事項については、[Elastic のドキュメント](#) を参照してください。



### 注記

可用性の高い Elasticsearch 環境には3つ以上の Elasticsearch ノードが必要であり、それぞれを異なるホストに配置する必要があります。レプリカを作成するために、**openshift\_logging\_es\_number\_of\_replicas** を1以上の値に設定します。

### すべての Elasticsearch デプロイメントの表示

現在の Elasticsearch デプロイメントをすべて表示するには、以下を実行します。

```
$ oc get dc --selector logging-infra=elasticsearch
```

### Elasticsearch での高可用性の設定

可用性の高い Elasticsearch 環境には3つ以上の Elasticsearch ノードが必要であり、それぞれを異なるホストに配置する必要があります。レプリカを作成するために、**openshift\_logging\_es\_number\_of\_replicas** Ansible 変数 を1以上の値に設定します。

3つの Elasticsearch ノードが含まれる OpenShift Container Platform クラスターについては、以下のシナリオを参照してください。

- **openshift\_logging\_es\_number\_of\_replicas** を1に設定すると、2つのノードに、クラスター内の全 Elasticsearch データのコピーが含まれるようになります。これにより、Elasticsearch データを持つノードがダウンした場合に、別のノードにクラスター内のすべての Elasticsearch データのコピーがあります。

- **openshift\_logging\_es\_number\_of\_replicas** を **3** に設定すると、4つのノードに、クラスター内の全 Elasticsearch データのコピーが含まれるようになります。これにより、Elasticsearch データを持つノードが3つダウンした場合に、1つのノードにクラスター内のすべての Elasticsearch データのコピーが含まれるようになっています。  
このシナリオでは、複数の Elasticsearch ノードがダウンしているため、Elasticsearch のステータスは RED になり、新しい Elasticsearch シャードは割り当てられません。ただし、高可用性のために Elasticsearch データが失われることはありません。

高可用性とパフォーマンスの間にはトレードオフの関係があることに注意してください。たとえば、**openshift\_logging\_es\_number\_of\_replicas=2** および

**openshift\_logging\_es\_number\_of\_shards=3** を設定すると、Elasticsearch が多くのリソースを使用してクラスター内のノード間でシャードデータを複製する必要があります。また、レプリカ数を増やすことにより、各ノードのデータストレージ要件を2倍または3倍に引き上げる必要があるため、Elasticsearch の [永続ストレージを計画する際に](#) 考慮に入れる必要があります。

### シャード数を設定する際の考慮事項

**openshift\_logging\_es\_number\_of\_shards** パラメーターについては、以下を考慮してください。

- パフォーマンスを上げるには、シャードの数を増やします。たとえば、3つのノードで設定されるクラスターでは、**openshift\_logging\_es\_number\_of\_shards=3** を設定します。これにより、各インデックスは3つの部分(シャード)に分割され、インデックスの処理による負荷は3つのノードすべてに分散されます。
- 多数のプロジェクトがあり、クラスターに数千を超えるシャードがある場合にはパフォーマンスが低下する可能性があります。この場合は、シャードの数を減らすか、またはキューションの時間を減らします。
- 少数の非常に大きなインデックスがある場合、**openshift\_logging\_es\_number\_of\_shards=3** 以上を設定する必要がある場合があります。Elasticsearch では最大のシャードサイズを 50 GB 未満にすることを推奨しています。

### ノードセレクター

Elasticsearch は大量のリソースを使用する可能性があるため、クラスターのすべてのメンバーでは、メンバー同士およびリモートストレージとのネットワーク接続の待機時間が低く設定する必要があります。待機時間を低く設定するには、[ノードセレクター](#) を使用してインスタンスを専用ノード、つまりクラスター内の専用領域に指定します。

ノードセレクターを設定するには、インベントリーファイルで **openshift\_logging\_es\_nodeselector** 設定オプションを指定します。これはすべての Elasticsearch デプロイメントに適用されます。そのため、ノードセレクターを個別化する必要がある場合は、デプロイメント後に各デプロイメント設定を手動で編集する必要があります。ノードセレクターは Python と互換性のある dict (辞書) として指定されます。たとえば、`{"node-type":"infra", "region":"east"}` のようになります。

#### 36.5.2.1. 永続的な Elasticsearch ストレージ

デフォルトで、**openshift\_logging** Ansible ロールは、Pod のすべてのデータが再起動時に失われる一時デプロイメントを作成します。

実稼働環境の場合、それぞれの Elasticsearch デプロイメント設定には永続ストレージボリュームが必要です。既存の [Persistent Volume Claim \(永続ボリューム要求、PVC\)](#) を指定するか、または OpenShift Container Platform がこれを作成することを許可することができます。

- **既存 PVC を使用する。** デプロイメントに独自の PVC を作成する場合、OpenShift Container Platform はそれらの PVC を使用します。

`openshift_logging_es_pvc_prefix` 設定に一致するように PVC の名前を指定します。この設定のデフォルトは `logging-es` です。各 PVC にシーケンス番号が追加された名前 (`logging-es-0`、`logging-es-1`、`logging-es-2` など) を割り当てます。

- **OpenShift Container Platform による PVC の作成を許可する。** Elasticsearch の PVC が存在しない場合、OpenShift Container Platform は [Ansible インベントリーファイル](#) のパラメーターに基づいて PVC を作成します。

パラメーター	説明
<code>openshift_logging_es_pvc_size</code>	PVC 要求のサイズを指定します。
<code>openshift_logging_elasticsearch_storage_type</code>	ストレージタイプを <code>pvc</code> と指定します。  <div style="display: flex; align-items: center;">  <div style="margin-left: 10px;"> <p><b>注記</b></p> <p>これはオプションのパラメーターです。<code>openshift_logging_es_pvc_size</code> パラメーターを 0 よりも大きな値に設定する場合、ロギングインストーラーはこのパラメーターをデフォルトで <code>pvc</code> に自動的に設定します。</p> </div> </div>
<code>openshift_logging_es_pvc_prefix</code>	オプションで、PVC のカスタム接頭辞を指定します。

以下に例を示します。

```
openshift_logging_elasticsearch_storage_type=pvc
openshift_logging_es_pvc_size=104802308Ki
openshift_logging_es_pvc_prefix=es-logging
```

[動的にプロビジョニングされた PV](#) を使用している場合、OpenShift Container Platform ロギングインストーラーはデフォルトのストレージクラスを使用する PVC、または `openshift_logging_elasticsearch_pvc_storage_class_name` パラメーターで指定された PVC を作成します。

NFS ストレージを使用する場合、OpenShift Container Platform インストーラーは `openshift_logging_storage_*` パラメーターに基づいて永続ボリュームを作成し、[OpenShift Container Platform ロギングインストーラー](#) は `openshift_logging_es_pvc_*` パラメーターを使用して PVC を作成します。EFK で永続ボリュームを使用できるように正しいパラメーターを指定してください。また、ロギングインストーラーはコアインフラストラクチャーで NFS のインストールをデフォルトでブロックするため、Ansible インベントリーファイルで `openshift_enable_unsupported_configurations=true` パラメーターを設定してください。



### 警告

NFS ストレージをボリュームまたは永続ボリュームとして使用したり、Gluster などの NAS を使用したりすることは Elasticsearch ストレージではサポートされません。Lucene が NFS が指定しないファイルシステムの動作に依存するためです。データの破損およびその他の問題が発生する可能性があります。

お使いの環境で NFS ストレージが必要な場合は、以下の方法のいずれかを使用します。

- [永続ボリュームとしての NFS](#)
- [ローカルストレージとしての NFS ストレージ](#)

#### 36.5.2.1.1. 永続ボリュームとしての NFS の使用

NFS を [自動的にプロビジョニングされた永続ボリューム](#) として、または [事前に定義された NFS ボリューム](#) を使用してデプロイできます。

詳細は、2 つの [Persistent Volume Claim \(永続ボリューム要求\)](#) 間での [NFS マウントの共有](#) を参照して、2 つの別個のコンテナで使用されるシャードストレージを利用します。

#### 自動的にプロビジョニングされた NFS の使用

NFS が自動的にプロビジョニングされる永続ボリュームとして使用するには、以下を実行します。

1. 以下の行を Ansible インベントリーファイルに追加して、NFS の自動プロビジョニングされるストレージクラスを作成し、バックアップストレージを動的にプロビジョニングします。

```
openshift_logging_es_pvc_storage_class_name=$nfsclass
openshift_logging_es_pvc_dynamic=true
```

2. 以下のコマンドを使用して、ロギング Playbook を使用した NFS ボリュームのデプロイを実行します。

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openshift-logging/config.yml
```

3. 以下の手順を実行して PVC を作成します。
  - a. Ansible インベントリーファイルを編集して PVC サイズを設定します。

```
openshift_logging_es_pvc_size=50Gi
```



### 注記

ロギング Playbook はサイズに基づいてボリュームを選択し、その他の永続ボリュームのサイズが同じ場合に予想しないボリュームを使用する可能性があります。

- b. 以下のコマンドを使用して Ansible `deploy_cluster.yml` Playbook を返します。

■

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

インストーラー Playbook は、**openshift\_logging\_storage** 変数に基づいて NFS ボリュームを作成します。

### 事前に定義された NFS ボリュームの使用

既存の NFS ボリュームを使用して、OpenShift Container Platform クラスタと共にロギングをデプロイするには、以下を実行します。

1. Ansible インベントリーファイルを編集して、NFS ボリュームを設定し、PVC サイズを設定します。

```
openshift_logging_storage_kind=nfs
openshift_logging_storage_access_modes=['ReadWriteOnce']
openshift_logging_storage_nfs_directory=/share 1
openshift_logging_storage_nfs_options='*(rw,root_squash)' 2
openshift_logging_storage_labels={'storage': 'logging'}
openshift_logging_elasticsearch_storage_type=pvc
openshift_logging_es_pvc_size=10Gi
openshift_logging_es_pvc_storage_class_name=""
openshift_logging_es_pvc_dynamic=true
openshift_logging_es_pvc_prefix=logging
```

- 1 2** これらのパラメーターは、**/usr/share/ansible/openshift-ansible/playbooks/deploy\_cluster.yml** インストール Playbook でのみ動作します。このパラメーターは **/usr/share/ansible/openshift-ansible/playbooks/openshift-logging/config.yml** Playbook では機能しません。

2. 以下のコマンドを使用して、EFK スタックを再デプロイします。

```
ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

#### 36.5.2.1.2. NFS のローカルストレージとしての使用

NFS サーバーに大規模なファイルを割り当て、ファイルをノードにマウントすることができます。次に、ファイルをホストパスデバイスとして使用できます。

```
$ mount -F nfs nfserver:/nfs/storage/elasticsearch-1 /usr/local/es-storage
$ chown 1000:1000 /usr/local/es-storage
```

次に、以下で説明するように **/usr/local/es-storage** をホストマウントとして使用します。異なるバックアップファイルをそれぞれの Elasticsearch ノードのストレージとして使用します。

このルーパックはノード上で、OpenShift Container Platform の外部から手動で保守する必要があります。コンテナ内から保守することはできません。

各ノードホスト上のローカルディスクボリューム (利用可能な場合) を Elasticsearch レプリカのストレージとして使用することができます。これを実行するには、以下のような準備が必要です。

1. 関連するサービスアカウントに、ローカルボリュームをマウントし、編集する権限を指定する必要があります。

```
$ oc adm policy add-scc-to-user privileged \
system:serviceaccount:openshift-logging:aggregated-logging-elasticsearch
```



### 注記

以前のバージョンの OpenShift Container Platform からアップグレードする場合、クラスターロギングは **logging** プロジェクトにインストールされている可能性があります。サービスアカウントを随時調整する必要があります。

- それぞれの Elasticsearch ノード定義には、権限を要求できるようにパッチが適用されている必要があります。以下は例になります。

```
$ for dc in $(oc get deploymentconfig --selector component=es -o name); do
  oc scale $dc --replicas=0
  oc patch $dc \
    -p '{"spec":{"template":{"spec":{"containers":[{"name":"elasticsearch","securityContext":{"privileged": true}}]}}}}'
done
```

- Elasticsearch レプリカは、ローカルストレージを使用するために適切なノード上に配置する必要があります。また、適切なノードが一定期間ダウンしている場合であっても、移動させてはなりません。この場合、各 Elasticsearch レプリカに対し、管理者がストレージを割り当てたノードに固有のノードセクターを指定する必要があります。ノードセクターを設定するには、各 Elasticsearch デプロイメント設定を編集し、**nodeSelector** セクションを追加または編集して、必要な各ノードに対して適用した一意のラベルを指定します。

```
apiVersion: v1
kind: DeploymentConfig
spec:
  template:
    spec:
      nodeSelector:
        logging-es-node: "1" 1
```

- 1** このラベルは、該当するラベルを持つ単一ノードのレプリカを一意に特定できるものである必要があります。この場合は、**logging-es-node=1** になります。

- それぞれの必要なノードのノードセクターを作成します。
- oc label** コマンドを使用して、必要な分のノードにラベルを適用します。

たとえば、デプロイメントに3つのインフラストラクチャーノードが含まれる場合、以下のようにそれらのノードのラベルを追加できます。

```
$ oc label node <nodename1> logging-es-node=0
$ oc label node <nodename2> logging-es-node=1
$ oc label node <nodename3> logging-es-node=2
```

ラベルのノードへの追加に関する情報については、[ノードのラベルの更新](#) を参照してください。

代わりに **oc patch** コマンドを使用して、ノードセクターの適用を自動化することもできます。

```
$ oc patch dc/logging-es-<suffix> \
  -p '{"spec":{"template":{"spec":{"nodeSelector":{"logging-es-node":"0"}}}}}'
```

これらの手順を実行したら、ローカルホストマウントを各レプリカに適用できます。以下の例では、ストレージが各ノードの同じパスにマウントされていることを前提とします。

```
$ for dc in $(oc get deploymentconfig --selector component=es -o name); do
  oc set volume $dc \
    --add --overwrite --name=elasticsearch-storage \
    --type=hostPath --path=/usr/local/es-storage
  oc rollout latest $dc
  oc scale $dc --replicas=1
done
```

### 36.5.2.1.3. Elasticsearch の hostPath ストレージの設定

Elasticsearch の hostPath ストレージを使用して OpenShift Container Platform クラスターをプロビジョニングできます。

各ノードホストでローカルディスクボリュームを Elasticsearch レプリカのストレージとして使用するには、以下を実行します。

1. ローカル Elasticsearch ストレージのローカルマウントポイントを各インフラストラクチャーノードに作成します。

```
$ mkdir /usr/local/es-storage
```

2. ファイルシステムを Elasticsearch ボリュームに作成します。

```
$ mkfs.ext4 /dev/xxx
```

3. elasticsearch ボリュームをマウントします。

```
$ mount /dev/xxx /usr/local/es-storage
```

4. 以下の行を **/etc/fstab** に追加します。

```
$ /dev/xxx /usr/local/es-storage ext4
```

5. マウントポイントの所有権を変更します。

```
$ chown 1000:1000 /usr/local/es-storage
```

6. ローカルボリュームをマウントし、編集する権限を関連するサービスアカウントに付与します。

```
$ oc adm policy add-scc-to-user privileged \
  system:serviceaccount:openshift-logging:aggregated-logging-elasticsearch
```



## 注記

以前のバージョンの OpenShift Container Platform からアップグレードする場合、クラスターロギングは **logging** プロジェクトにインストールされている可能性があります。サービスアカウントを随時調整する必要があります。

7. 該当する権限を要求するには、例に示すように Ops クラスターの **--selector component=es-ops** を指定する各 Elasticsearch レプリカ定義にパッチを適用します。

```
$ for dc in $(oc get deploymentconfig --selector component=es -o name);
do
  oc scale $dc --replicas=0
  oc patch $dc \
    -p '{"spec":{"template":{"spec":{"containers":[{"name":"elasticsearch","securityContext":
{"privileged":
true}}}}}}}'
done
```

8. 適切なノードの Elasticsearch レプリカを見つけ、ローカルストレージを使用します。それらのノードが一定期間停止する場合でも、レプリカを移動することはできません。ノードの場所を指定するには、それぞれの Elasticsearch レプリカに対し、管理者がストレージを割り当てたノードに固有のノードセクターを指定します。ノードセクターを設定するには、各 Elasticsearch デプロイメント設定を編集し、**nodeSelector** セクションを追加または編集して、必要なそれぞれのノードに適用した一意のラベルを指定します。

```
apiVersion: v1
kind: DeploymentConfig
spec:
  template:
    spec:
      nodeSelector:
        logging-es-node: "1"
```

このラベルは、該当するラベルを持つ単一ノードのレプリカを一意に特定できるものである必要があります。この場合は、**logging-es-node=1** になります。

9. それぞれの必要なノードのノードセクターを作成します。**oc label** コマンドを使用して、必要な分のノードにラベルを適用します。たとえば、デプロイメントに3つのインフラストラクチャーノードが含まれる場合、以下のようにそれらのノードのラベルを追加できます。

```
$ oc label node <nodename1> logging-es-node=0
$ oc label node <nodename2> logging-es-node=1
$ oc label node <nodename3> logging-es-node=2
```

ノードセクターの適用を自動化するには、以下のように **oc label** コマンドではなく、**oc patch** コマンドを使用します。

```
$ oc patch dc/logging-es-<suffix> \
-p '{"spec":{"template":{"spec":{"nodeSelector":{"logging-es-node":"1"}}}}}'
```

10. これらの手順を実行したら、ローカルホストマウントを各レプリカに適用できます。以下の例では、ストレージが各ノードの同じパスにマウントされていることを前提とし、Ops クラスターに **--selector component=es-ops** を指定します。

```
$ for dc in $(oc get deploymentconfig --selector component=es -o name);
do
  oc set volume $dc \
    --add --overwrite --name=elasticsearch-storage \
    --type=hostPath --path=/usr/local/es-storage
  oc rollout latest $dc
  oc scale $dc --replicas=1
done
```

#### 36.5.2.1.4. Elasticsearch のスケールの変更

クラスター内の Elasticsearch ノード数を増やす必要がある場合は、追加する必要がある各 Elasticsearch ノードについてデプロイメント設定を作成することができます。

永続ボリュームの性質上、また Elasticsearch がデータを保存し、クラスターを復元する方法により、Elasticsearch デプロイメント設定でノードを単純に増やすことはできません。

Elasticsearch のスケールを変更する最も簡単な方法は、前述のようにインベントリーホストファイルを変更し、ロギング Playbook を再実行することです。デプロイメント用の永続ストレージを指定していると仮定すると、この方法によって混乱が生じることはないはずです。



#### 注記

**openshift\_logging\_es\_cluster\_size** の値が、クラスターにある現在の Elasticsearch のノード数 (スケールアップした後) より多い場合のみ、ロギング Playbook を使用して Elasticsearch クラスターのサイズを調節できます。

#### 36.5.2.1.5. Elasticsearch レプリカ数の変更

Elasticsearch レプリカ数は、インベントリーホストファイルの **openshift\_logging\_es\_number\_of\_replicas** 値を編集し、前述のようにロギング Playbook を再実行して変更できます。

変更は新規インデックスにのみ適用されます。既存のインデックスは、継続して以前のレプリカ数を使用します。たとえば、インデックスの数を 3 から 2 に変更すると、クラスターは、新規インデックスにはレプリカ数 2 を、既存のインデックスには 3 を使用します。

以下のコマンドを実行して、既存のインデックスのレプリカ数を変更できます。

```
$ oc exec -c elasticsearch $pod -- es_util --query=project.* -d '{"index":{"number_of_replicas":"2"}}'
```

1

- 1 既存のインデックスに使用するレプリカ数を指定します。

#### 36.5.2.1.6. Elasticsearch のルートとしての公開

デフォルトでは、OpenShift の集計ロギングでデプロイされた Elasticsearch はロギングクラスターの外部からアクセスできません。データにアクセスする必要があるツールに対しては、Elasticsearch への外部アクセスのルートを有効にすることができます。

OpenShift トークンを使用して Elasticsearch にアクセスできます。また、サーバー証明書を作成する際に (Kibana と同様に)、外部の Elasticsearch および Elasticsearch Ops ホスト名を指定することができます。

1. re-encrypt ルートとして Elasticsearch にアクセスするには、以下の変数を定義します。

```
openshift_logging_es_allow_external=True
openshift_logging_es_hostname=elasticsearch.example.com
```

2. Playbook ディレクトリーに切り替え、以下の Ansible Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i </path/to/inventory>] \
  playbooks/openshift-logging/config.yml
```

3. Elasticsearch にリモートでログインするには、要求に 3 つの HTTP ヘッダーが含まれている必要があります。

```
Authorization: Bearer $token
X-Proxy-Remote-User: $username
X-Forwarded-For: $ip_address
```

4. ログにアクセスできるようになるには、プロジェクトへのアクセスが必要です。以下に例を示します。

```
$ oc login <user1>
$ oc new-project <user1project>
$ oc new-app <httpd-example>
```

5. 要求内で使用するこの ServiceAccount のトークンを取得する必要があります。

```
$ token=$(oc whoami -t)
```

6. 以前に設定したトークンを使用して、公開ルート経由で Elasticsearch にアクセスできる必要があります。

```
$ curl -k -H "Authorization: Bearer $token" -H "X-Proxy-Remote-User: $(oc whoami)" -H "X-Forwarded-For: 127.0.0.1" https://es.example.test/project.my-project.*/_search?q=level:err |
python -mjson.tool
```

### 36.5.3. Fluentd

Fluentd はノードラベルセクターに従ってノードをデプロイする DaemonSet としてデプロイされます。ノードラベルセクターはインベントリーパラメーター

**openshift\_logging\_fluentd\_nodeselector** を使用して指定することができ、デフォルトは **logging-infra-fluentd** です。OpenShift クラスターのインストールの一環として、Fluentd ノードセクターを永続化された **ノードラベル** の一覧に追加することが推奨されます。

Fluentd は **journald** をシステムログソースとして使用します。これらは、オペレーティングシステム、コンテナランタイム、および OpenShift からのログメッセージです。

利用可能なコンテナのランタイムでは、ログメッセージのソースを特定するための最小限の情報が提供されます。ログ収集およびログの正規化は、Pod が削除されてから行われるので、ラベルまたはアノテーションなどの追加のメタデータが API サーバーから取得できません。

ログコレクターがログの処理を完了する前に、特定の名前および namespace が含まれる Pod が削除された場合には、よく似た名前の Pod および namespace とログメッセージを区別する方法がなくなる可能性があります。これにより、ログがインデックス化され、Pod にデプロイしていないユーザーが所有するインデックスにアノテーションが付けられる可能性があります。



## 重要

利用可能なコンテナランタイムは、ログメッセージのソースを特定するための最小限の情報を提供し、個別のログメッセージが一意となる確証はなく、これらのメッセージにより、そのソースを追跡できる訳ではありません。

OpenShift Container Platform 3.9 以降のクリーンインストールではデフォルトのログドライバーとして **json-file** を使用しますが、OpenShift Container Platform 3.7 からアップグレードされた環境は既存の **journald** ログのドライバー設定を維持します。**json-file** ログドライバーの使用を推奨します。既存のログドライバー設定を **json-file** に変更する手順については、[集計ロギングドライバーの変更](#) を参照してください。

## Fluentd ログの表示

ログの表示方法は、[LOGGING\\_FILE\\_PATH の設定](#) によって異なります。

- **LOGGING\_FILE\_PATH** がファイルを参照する場合、**logs** ユーティリティーを使用して Fluentd ログファイルの内容を出力します。

```
oc exec <pod> -- logs 1
```

- 1** Fluentd Pod の名前を指定します。**logs** の前にスペースがあることに注意してください。

以下に例を示します。

```
oc exec logging-fluentd-lmvms -- logs
```

ログファイルの内容は、最も古いログから順番に印刷されます。**-f** オプションを使用して、ログに書き込まれている内容をフォローします。

- **LOGGING\_FILE\_PATH=console** を使用している場合、Fluentd はログをデフォルトの場所 **/var/log/fluentd/fluentd.log** に書き込みます。**oc logs -f <pod\_name>** コマンドでログを取得できます。

以下に例を示します。

```
oc logs -f fluentd.log
```

## Fluentd ログの位置の設定

Fluentd は、**LOGGING\_FILE\_PATH** 環境変数に応じて、デフォルトの **/var/log/fluentd/fluentd.log** の指定ファイルか、またはコンソールにログを書き出します。

Fluentd ログ出力のデフォルトの位置を変更するには、[デフォルトインベントリーファイル](#) で **LOGGING\_FILE\_PATH** パラメーターを使用します。特定のファイルを指定するか、Fluentd のデフォルトの場所を使用できます。

```
LOGGING_FILE_PATH=console ①
LOGGING_FILE_PATH=<path-to-log/ fluentd.log> ②
```

- ① ログの出力を Fluentd のデフォルトの場所へ送信します。 **oc logs -f <pod\_name>** コマンドでログを取得します。
- ② ログ出力を指定されたファイルへ送信します。 **oc exec <pod\_name> -- logs** コマンドでログを取得します。

これらのパラメーターを変更した後に、[ロギングインストーラー Playbook](#) を再実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i <path/to/inventory>] \
  playbooks/openshift-logging/config.yml
```

### Fluentd ログローテーションの設定

現在の Fluentd ログファイルが指定されたサイズに達すると、OpenShift Container Platform は **fluentd.log** ログファイルの名前を自動的に変更し、新規のロギングデータを収集できるようにします。ログローテーションはデフォルトで有効にされます。

以下の例では、最大ログサイズが 1Mb で 4 つのログを保持する必要があるクラスターのログを示しています。 **fluentd.log** が 1Mb に達すると、OpenShift Container Platform は現在の **fluentd.log.4** を削除し、Fluentd ログのそれぞれの名前を順番に変更し、新規の **fluentd.log** を作成します。

```
fluentd.log 0b
fluentd.log.1 1Mb
fluentd.log.2 1Mb
fluentd.log.3 1Mb
fluentd.log.4 1Mb
```

環境変数を使用して、Fluentd ログファイルのサイズおよび OpenShift Container Platform が保持する名前変更されたファイルの数を制御することができます。

表36.1 Fluentd ログローテーションを設定するためのパラメーター

パラメーター	説明
<b>LOGGING_FILE_SIZE</b>	単一の Fluentd ログファイルの最大サイズ (バイト単位)。 <b>fluentd.log</b> ファイルのサイズがこの値を超える場合、OpenShift Container Platform は <b>fluentd.log.*</b> ファイルの名前を変更し、新規の <b>fluentd.log</b> を作成します。デフォルトは 1024000 (1MB) です。
<b>LOGGING_FILE_AGE</b>	Fluentd が削除する前に保持するログ数。デフォルト値は <b>10</b> です。

以下に例を示します。

```
$ oc set env ds/logging-fluentd LOGGING_FILE_AGE=30 LOGGING_FILE_SIZE=1024000"
```

**LOGGING\_FILE\_PATH=console** を設定してログローテーションをオフにします。これにより、Fluentd はログを Fluentd のデフォルトの場所である **/var/log/fluentd/fluentd.log** に書き込みます。ここでは、 **oc logs -f <pod\_name>** コマンドを使用してそれらのログを取得できます。

```
$ oc set env ds/fluentd LOGGING_FILE_PATH=console
```

## MERGE\_JSON\_LOG でのログの JSON 解析の無効化

デフォルトで、Fluentd はログメッセージが JSON 形式であり、そのメッセージを Elasticsearch に送信された JSON ペイロードドキュメントにマージするかどうかを判別します。

JSON 解析を使用する場合は、以下を行います。

- Elasticsearch が整合性のないタイプのマッピングによりドキュメントを拒否することでログが失われる。
- 拒否されるメッセージの繰り返し処理によってバッファーストレージ不足が生じる。
- 名前が同じフィールドのデータが上書きされる。

これらの問題の一部を軽減する方法は、[ログコレクターによってログを正規化する方法の設定](#) を参照してください。

これらの問題を回避する場合やログから JSON を解析する必要がない場合には、JSON 解析を無効にできます。

JSON 解析を無効にするには、以下を実行します。

1. 次のコマンドを実行します。

```
oc set env ds/logging-fluentd MERGE_JSON_LOG=false ❶
```

- ❶ これを **false** に設定してこの機能を無効にするか、または **true** に設定してこの機能を有効にします。

Ansible の実行時にこの設定が適用されるようにするには、Ansible インベントリに **openshift\_logging\_fluentd\_merge\_json\_log="false"** を追加します。

## ログコレクターによってログを正規化する方法の設定

クラスターロギングは、データベーススキーマなどの特定のデータモデルを使用して、ログレコードとそのメタデータをロギングストアに格納します。このデータについてはいくつかの制限があります。

- 実際のログメッセージが含まれる **"message"** フィールドがなければなりません。
- RFC 3339 形式のログレコードのタイムスタンプを含む **"@timestamp"** フィールドがなければなりません (ミリ秒以上の単位の使用が望ましい)。
- **err**、**info**、**unknown** などのログレベルが含まれる **"level"** フィールドがなければなりません。



### 注記

データモデルの詳細については、[Exported Fields](#) を参照してください。

これらの要件により、異なるサブシステムから収集されるログデータで競合や不整合が生じる場合があります。

たとえば、**MERGE\_JSON\_LOG** 機能 (**MERGE\_JSON\_LOG=true**) を使用する場合、アプリケーションの出力のログを JSON で記録し、ログコレクターが Elasticsearch 内でデータを自動的に解析し、インデックス化すると非常に便利です。ただし、これにより、以下のような問題が発生します。

- フィールド名が空になることや、Elasticsearch で使用できない文字が含まれる可能性があります。
- 同じ namespace 内にある異なるアプリケーションで、値のデータタイプが異なり、フィールド名が同じものを出力する可能性があります。
- アプリケーションが過剰にフィールドを出力する可能性があります。
- フィールドは、クラスターロギングの組み込みフィールドと競合する可能性があります。

以下の表で、Fluentd ログコレクターの daemonset を編集し、環境変数を設定することで、クラスターロギングが異種ソースからフィールドを処理する方法を設定できます。

- **未定義のフィールド**。ViaQ データモデルに対して不明なフィールドは **undefined** と呼ばれます。異種システムのログデータには、未定義のフィールドを含めることができます。このデータモデルでは、すべての最上位フィールドが定義され、記述される必要があります。パラメーターを使用して OpenShift Container Platform が、**undefined** という最上位フィールド以下のすべての未定義のフィールドを移動し、**既知**の最上位フィールドとの競合を避けます。未定義フィールドを最上位フィールドに追加し、他のフィールドを **undefined** コンテナに移動できます。

また、未定義のフィールドで特殊文字を置き換えることができ、未定義フィールドを JSON 文字列表現に変換することもできます。JSON 文字列に変換しても値の構造を維持されるため、値を後で取得してマップまたは配列に戻すことができます。

- 番号やブール値などの簡単なスカラー値は、引用符で囲まれた文字列に変更されます。たとえば、**10** は **"10"**、**3.1415** は **"3.1415"**、**false** は **"false"** となります。
  - map/dict の値と配列の値は JSON 文字列表現に変換されます：**"mapfield":{"key":"value"}** は **"mapfield":{"key":"value"}** に、**"arrayfield":[1,2,"three"]** は **"arrayfield":["1,2,\"three\"]** に変換されます。
- **定義されたフィールド**。定義されたフィールドはログの最上位に表示されます。定義されたフィールドとして考慮されるフィールドを設定できます。

**CDM\_DEFAULT\_KEEP\_FIELDS** パラメーターで定義されるデフォルトの最上位のフィールドは次のとおりです：

**CEE、time、@timestamp、aushape、ci\_job、collectd、docker、fedora-ci、file、foreman、geopip、hostname、ipaddr4、ipaddr6、kubernetes、level、message、namespace\_name、namespace\_uuid、offset、openstack、ovirt、pid、pipeline\_metadata、service、systemd、tags、testcase、tlog、viaq\_msg\_id**

**#{CDM\_DEFAULT\_KEEP\_FIELDS}** または **#{CDM\_EXTRA\_KEEP\_FIELDS}** に含まれないフィールドはすべて、**CDM\_USE\_UNDEFINED** が **true** の場合、**#{CDM\_UNDEFINED\_NAME}** に移動されます。これらのパラメーターの詳細は、以下の表を参照してください。



## 注記

**CDM\_DEFAULT\_KEEP\_FIELDS** パラメーターは、上級ユーザーのみが使用するか、Red Hat サポートによって使用することが指示された場合に使用されることが想定されます。

- **空のフィールド。**空のフィールドにはデータがありません。ログから、空のままにするフィールドを判別できます。

表36.2 ログの正規化に使用する環境パラメーター

パラメーター	定義	例
<b>CDM_EXTRA_KEEP_FIELDS</b>	<b>CDM_DEFAULT_KEEP_FIELDS</b> に加えて、ログの最上位に保持する定義されたフィールドの追加セットを指定します。デフォルトは "" です。	<b>CDM_EXTRA_KEEP_FIELDS="broker"</b>
<b>CDM_KEEP_EMPTY_FIELDS</b>	空の場合でも CSV 形式で保持するフィールドを指定します。空の定義されたフィールドで指定されないものはドロップされます。デフォルトは message で、空のメッセージを維持します。	<b>CDM_KEEP_EMPTY_FIELDS="message"</b>
<b>CDM_USE_UNDEFINED</b>	未定義のフィールドを <b>undefined</b> の最上位フィールドに移動するには、 <b>true</b> に設定します。デフォルトは <b>false</b> です。 <b>true</b> の場合、 <b>CDM_DEFAULT_KEEP_FIELDS</b> および <b>CDM_EXTRA_KEEP_FIELDS</b> の値は <b>undefined</b> に移動されません。	<b>CDM_USE_UNDEFINED=true</b>
<b>CDM_UNDEFINED_NAME</b>	<b>CDM_USE_UNDEFINED</b> を使用する場合は、未定義の最上位フィールドの名前を指定します。デフォルトは `undefined` です。 <b>CDM_USE_UNDEFINED</b> が <b>true</b> の場合にのみ有効にされます。	<b>CDM_UNDEFINED_NAME="undef"</b>
<b>CDM_UNDEFINED_MAX_NUM_FIELDS</b>	未定義フィールドの数がこの数を超える場合、未定義のフィールドはすべて JSON 文字列表現に変換され、 <b>CDM_UNDEFINED_NAME</b> フィールドに保存されます。レコードに未定義フィールド値を超える数が含まれる場合、これらのフィールドでの処理はこれ以上実行されません。代わりに、フィールドは最上位の <b>CDM_UNDEFINED_NAME</b> フィールドに保存された単一文字列の JSON 値に変換されます。デフォルトの <b>-1</b> を維持することで、未定義フィールドの無制限の値が許可されますが、これは推奨されません。  注: このパラメーターは、 <b>CDM_USE_UNDEFINED</b> が <b>false</b> の場合でも有効です。	<b>CDM_UNDEFINED_MAX_NUM_FIELDS=4</b>
<b>CDM_UNDEFINED_TO_STRING</b>	未定義フィールドをすべて JSON 文字列表現に変換するには、 <b>true</b> に設定します。デフォルトは <b>false</b> です。	<b>CDM_UNDEFINED_TO_STRING=true</b>

パラメーター	定義	例
<b>CDM_UNDEFINED_DOT_REPLACE_CHAR</b>	未定義フィールドでドット文字 '.' の代わりに使用する文字を指定します。 <b>MERGE_JSON_LOG</b> は <b>true</b> である必要があります。デフォルトは <b>UNUSED</b> です。 <b>MERGE_JSON_LOG</b> パラメーターを <b>true</b> に設定した場合は、以下の注を参照してください。	<b>CDM_UNDEFINED_DOT_REPLACE_CHAR="_"</b>

## 注記

Fluentd ログコレクター daemonset の **MERGE\_JSON\_LOG** パラメーターおよび **CDM\_UNDEFINED\_TO\_STRING** 環境変数を **true** に設定した場合、Elasticsearch 400 エラーを受信する可能性があります。**MERGE\_JSON\_LOG=true** の場合、ログコレクターが string 以外のデータタイプのフィールドを追加します。**CDM\_UNDEFINED\_TO\_STRING=true** を設定する場合、ログコレクターはそれらのフィールドへの文字列の値の追加を試行し、これにより Elasticsearch 400 エラーが生じます。ログコレクターが翌日のログにインデックスをロールオーバーすると、エラーはクリアされます。

ログコレクターがインデックスをロールオーバーする場合、完全に新規のインデックスが作成されます。フィールドの定義は更新され、400 エラーは発生しません。詳細は、[Setting MERGE\\_JSON\\_LOG](#) および [CDM\\_UNDEFINED\\_TO\\_STRING](#) を参照してください。

未定義および空のフィールドの処理を設定するには、**logging-fluentd** daemonset を編集します。

- 必要に応じてフィールドの処理方法を設定します。
  - CDM\_EXTRA\_KEEP\_FIELDS** を使用して移動するフィールドを指定します。
  - CSV 形式で **CDM\_KEEP\_EMPTY\_FIELDS** パラメーターで保持する空のフィールドをすべて指定します。
- 必要に応じて未定義フィールドの処理方法を設定します。
  - CDM\_USE\_UNDEFINED** を **true** に設定し、未定義のフィールドを最上位の **undefined** フィールドに移動します。
  - CDM\_UNDEFINED\_NAME** パラメーターを使用して、未定義フィールドの名前を指定します。
  - CDM\_UNDEFINED\_MAX\_NUM\_FIELDS** をデフォルトの **-1** 以外の値に設定し、単一コードにおける未定義フィールド数の上限を設定します。
- CDM\_UNDEFINED\_DOT\_REPLACE\_CHAR** を指定して、未定義フィールド名のすべてのドット (.) 文字を別の文字に変更します。たとえば、**CDM\_UNDEFINED\_DOT\_REPLACE\_CHAR=@@@** の場合で、**foo.bar.baz** という名前のフィールドがある場合、そのフィールドは **foo@@@bar@@@baz** に変換されます。
- UNDEFINED\_TO\_STRING** を **true** に設定し、未定義フィールドを JSON 文字列表現に変換します。

## 注記

**CDM\_UNDEFINED\_TO\_STRING** または **CDM\_UNDEFINED\_MAX\_NUM\_FIELDS** パラメーターを設定する場合、**CDM\_UNDEFINED\_NAME** を使用して未定義フィールドの名前を変更します。**CDM\_UNDEFINED\_TO\_STRING** または **CDM\_UNDEFINED\_MAX\_NUM\_FIELDS** が未定義フィールドの値タイプを変更する可能性があるため、このフィールドは必要になります。**CDM\_UNDEFINED\_TO\_STRING** または **CDM\_UNDEFINED\_MAX\_NUM\_FIELDS** が true に設定されていて、ログにさらに多くの未定義フィールドがある場合、値タイプは **string** になります。値タイプが変更される場合、Elasticsearch はレコードの受け入れを停止します (JSON から JSON 文字列への変更など)。

たとえば、**CDM\_UNDEFINED\_TO\_STRING** が **false** であるか、または **CDM\_UNDEFINED\_MAX\_NUM\_FIELDS** がデフォルトの **-1** の場合、未定義フィールドの値タイプは **json** になります。**CDM\_UNDEFINED\_MAX\_NUM\_FIELDS** をデフォルト以外の値に変更し、ログにさらに多くの未定義フィールドがある場合、値タイプは **string** (json string) になります。値タイプが変更された場合、Elasticsearch はレコードの受け入れを停止します。

## MERGE\_JSON\_LOG および CDM\_UNDEFINED\_TO\_STRING の設定

**MERGE\_JSON\_LOG** および **CDM\_UNDEFINED\_TO\_STRING** 環境変数を **true** に設定する場合、Elasticsearch 400 エラーを受信する可能性があります。**MERGE\_JSON\_LOG=true** の場合、ログコレクターが string 以外のデータタイプのフィールドを追加します。**CDM\_UNDEFINED\_TO\_STRING=true** を設定する場合、Fluentd はそれらのフィールドを文字列の値として追加することを試行し、これにより Elasticsearch 400 エラーが生じます。このエラーはインデックスが翌日にロールオーバーされるとクリアされます。

Fluentd が次の日のログのインデックスをロールオーバーする場合、全く新しいインデックスが作成されます。フィールドの定義は更新され、400 エラーは発生しません。

スキーマ違反やデータ破損などの **ハード (hard)** エラーのあるレコードについては、再試行できません。ログコレクターはエラー処理のためにレコードを送信します。最後に表示される **<label>** のように Fluentd 設定に **<label @ERROR>** セクションを追加する場合、それらのレコードを随時処理することができます。

以下に例を示します。

```
data:
  fluent.conf:
  ....

  <label @ERROR>
    <match **>
      @type file
      path /var/log/fluent/dlq
      time_slice_format %Y%m%d
      time_slice_wait 10m
      time_format %Y%m%dT%H%M%S%z
      compress gzip
    </match>
  </label>
```

このセクションではエラーレコードを **Elasticsearch dead letter queue (DLQ) ファイル** に書き込みます。ファイル出力の詳細は **fluentd のドキュメント** を参照してください。

次に、レコードを手動でクリーンアップし、ファイルを Elasticsearch / **\_bulk index** API で使用し、cURL を使用してそれらのレコードを追加できるようにファイルを編集することができます。Elasticsearch Bulk API についての詳細は [Elasticsearch のドキュメント](#) を参照してください。

### 複数行の Docker ログへの参加

Fluentd を Docker ログの部分的なフラグメントからログレコード全体を再構築するように Fluentd を設定できます。この機能は有効にすると、Fluentd は複数行の Docker ログを読み取り、それらを再作成し、データが欠落することなしにログを1つのレコードとして Elasticsearch に保存します。

ただし、この機能が原因でパフォーマンスの低下が発生する可能性があるため、機能はデフォルトでオフになっており、手動で有効にする必要があります。

以下の Fluentd 環境変数では、複数行の Docker ログを処理するようにクラスターロギングを設定します。

パラメーター	説明
USE_MULTILINE_JSON	<b>json-file</b> ログドライバーを使用する時にマルチライン Docker ログを処理するには <b>true</b> に設定します。このパラメーターは、デフォルトで <b>false</b> に設定されます。
USE_MULTILINE_JOURNAL	<b>journald</b> ログドライバーを使用する時にマルチライン Docker ログを処理するには <b>true</b> に設定します。Fluentd は docker ログの一部フラグメントからログレコード全体を再構築します。このパラメーターは、デフォルトで <b>false</b> に設定されます。

以下のコマンドで、使用されているログドライバーを確認できます。

```
$ docker info | grep -i log
```

以下のいずれかの出力が表示されます。

```
Logging Driver: json-file
```

```
Logging Driver: journald
```

複数行の Docker ログ処理をオンにするには、以下を実行します。

1. 以下のコマンドを使用して、複数行の Docker ログを有効にします。

- **json-file** ログドライバーの場合:

```
oc set env daemonset/logging-fluentd USE_MULTILINE_JSON=true
```

- **journald** ログドライバーの場合:

```
oc set env daemonset/logging-fluentd USE_MULTILINE_JOURNAL=true
```

クラスターの Fluentd Pod が再起動します。

### 外部ログアグリゲーターにログを送信するための Fluentd の設定

**secure-forward** プラグインを使用して、デフォルトの Elasticsearch に加えてログのコピーを外部ログアグリゲーターに送信するように Fluentd を設定できます。ローカルにホストされている Fluentd による処理の後に、ログレコードをさらに処理することができます。



## 重要

クライアント証明書を使用して **secure\_foward** プラグインを設定することはできません。認証は SSL/TLS プロトコルを介して実行できますが、**shared\_key** と宛先 **Fluentd** を **secure\_forward** 入力プラグインで設定する必要があります。

ロギングデプロイメントでは、外部ログアグリゲーターを設定するために、Fluentd configmap で **secure-forward.conf** セクションを指定します。

```
<store>
@type secure_forward
self_hostname pod-${HOSTNAME}
shared_key thisisasharedkey
secure yes
enable_strict_verification yes
ca_cert_path /etc/fluent/keys/your_ca_cert
ca_private_key_path /etc/fluent/keys/your_private_key
ca_private_key_passphrase passphrase
<server>
  host ose1.example.com
  port 24284
</server>
<server>
  host ose2.example.com
  port 24284
  standby
</server>
<server>
  host ose3.example.com
  port 24284
  standby
</server>
</store>
```

**oc edit** コマンドを使用して更新できます。

```
$ oc edit configmap/logging-fluentd
```

**secure-forward.conf** で使用される証明書を Fluentd Pod にマウントされる既存のシークレットに追加することができます。**your\_ca\_cert** と **your\_private\_key** の値は **configmap/logging-fluentd** の **secure-forward.conf** で指定されている値と一致している必要があります。

```
$ oc patch secrets/logging-fluentd --type=json \
  --patch "[{'op':'add','path':'/data/your_ca_cert','value':'$(base64 -w 0 /path/to/your_ca_cert.pem)}]"
$ oc patch secrets/logging-fluentd --type=json \
  --patch "[{'op':'add','path':'/data/your_private_key','value':'$(base64 -w 0
/path/to/your_private_key.pem)}]"
```



## 注記

`your_private_key` は、汎用的な名前に置き換えます。これは、JSON パスへのリンクであり、使用しているホストシステムのパスではありません。

外部アグリゲーターを設定する際は、Fluentd からのメッセージを安全に受信する必要があります。

外部アグリゲーターが別の Fluentd サーバーである場合、**fluent-plugin-secure-forward** プラグインがインストールされていて、それによって提供される入力プラグインを使用する必要があります。

```

<source>
  @type secure_forward

  self_hostname ${HOSTNAME}
  bind 0.0.0.0
  port 24284

  shared_key thisisasharedkey

  secure yes
  cert_path    /path/for/certificate/cert.pem
  private_key_path /path/for/certificate/key.pem
  private_key_passphrase secret_foo_bar_baz
</source>

```

**fluent-plugin-secure-forward** リポジトリ に **fluent-plugin-secure-forward** プラグインの設定方法に関する詳細の説明があります。

## Fluentd から API サーバーへの接続数の削減



### 重要

**mux** はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、<https://access.redhat.com/support/offerings/techpreview/> を参照してください。

**mux** は Secure Forward のリスナーサービスです。

パラメーター

説明

パラメーター	説明
<b>openshift_logging_use_mux</b>	デフォルトは <b>False</b> に設定されています。 <b>True</b> に設定されている場合は、 <b>mux</b> というサービスがデプロイされます。このサービスは、クラスターで実行されるノードエージェント Fluentd daemonset の Fluentd <b>secure_forward</b> アグリゲーターとして機能します。 <b>openshift_logging_use_mux</b> を使用して OpenShift API サーバーへの接続数を減らし、生ログを <b>mux</b> に送信して Kubernetes メタデータプラグインをオフにするように Fluentd の各ノードを設定します。これには、 <b>openshift_logging_mux_client_mode</b> を使用する必要があります。
<b>openshift_logging_mux_client_mode</b>	<b>openshift_logging_mux_client_mode</b> の値は <b>minimal</b> と <b>maximal</b> で、デフォルトはありません。 <b>openshift_logging_mux_client_mode</b> により、Fluentd ノードエージェントが Elasticsearch に直接ではなく mux にログを送信します。 <b>maximal</b> の値は、Fluentd がレコードを <b>mux</b> に送信する前にできるだけ多くの処理をノードで実行することを意味しています。 <b>maximal</b> 値は <b>mux</b> を使用する場合に推奨されます。 <b>minimal</b> の値は、Fluentd がまったく処理を行わないことを意味しており、処理用に生ログを <b>mux</b> に送信します。 <b>minimal</b> 値の使用は推奨されません。
<b>openshift_logging_mux_allow_external</b>	デフォルトは <b>False</b> に設定されています。 <b>True</b> に設定されている場合は、 <b>mux</b> サービスがデプロイされ、クラスターの外部で実行されている Fluentd クライアントが <b>secure_forward</b> を使用してログを送信できるように設定されます。これにより、OpenShift ログを OpenShift 以外のクライアント、または他の OpenShift クラスターの中央ロギングサービスとして使用することができます。
<b>openshift_logging_mux_hostname</b>	デフォルトは <b>mux</b> に <b>openshift_master_default_subdomain</b> を追加した値です。これは、 <b>external_clients</b> が <b>mux</b> に接続するために使用するホスト名であり、TLS サーバーの証明書サブジェクトで使用されます。
<b>openshift_logging_mux_port</b>	24284
<b>openshift_logging_mux_cpu_limit</b>	500M
<b>openshift_logging_mux_memory_limit</b>	2 Gi
<b>openshift_logging_mux_default_namespaces</b>	デフォルトは <b>mux-undefined</b> です。一覧の最初の値は未定義のプロジェクトに使用する namespace であり、これにデフォルトで作成する追加の namespace が続きます。通常、この値を設定する必要はありません。
<b>openshift_logging_mux_namespaces</b>	デフォルト値は空であり、外部の <b>mux</b> クライアント向けに追加の namespace を作成し、ログと関連付けることができます。この値を設定する必要があります。

## Fluentd でのログのスロットリング

とくに詳細なプロジェクトについては、管理者は処理される前の Fluentd によるログの読み取り速度を減速することができます。



### 警告

スロットリングは設定されたプロジェクトのログ集計が遅れる一因になる可能性があります。Fluentd が追いつく前に Pod が削除された場合、ログエントリが消失する可能性があります。



### 注記

Systemd ジャーナルをログソースとして使用している場合、スロットリングは機能しません。スロットリングの実装は、各プロジェクトの個々のログファイルの読み取りを調整できる機能によって決まります。ジャーナルからの読み取り時に、単一のログソースしか存在せず、ログファイルが存在しないと、ファイルベースのスロットリングは利用できません。Fluentd プロセスに読み込まれるログエントリを制限する方法はありません。

Fluentd に対して制限する必要があるプロジェクトを指示するには、デプロイメント後に ConfigMap のスロットル設定を編集します。

```
$ oc edit configmap/logging-fluentd
```

**throttle-config.yaml** キーの形式は、プロジェクト名と、各ノードでのログの読み取りに必要な速度が含まれる YAML ファイルです。デフォルトはノードごとに一度に 1000 行です。以下に例を示します。

- プロジェクト

```
project-name:
  read_lines_limit: 50

second-project-name:
  read_lines_limit: 100
```

- ロギング

```
logging:
  read_lines_limit: 500

test-project:
  read_lines_limit: 10

.operations:
  read_lines_limit: 100
```

Fluentd に変更を加えるには、設定を変更し、Fluentd Pod を再起動して変更を適用します。Elasticsearch に変更を加えるには、まず Fluentd をスケールダウンしてから、Elasticsearch をゼロにスケールダウンする必要があります。変更を行った後、最初に Elasticsearch をスケーリングしてから、Fluentd を元の設定にスケーリングします。

Elasticsearch をゼロに調整するには、以下を実行します。

```
$ oc scale --replicas=0 dc/<ELASTICSEARCH_DC>
```

デーモンセット設定の `nodeSelector` がゼロに一致するように変更します。

**Fluentd ノードセレクターを取得します。**

```
$ oc get ds logging-fluentd -o yaml |grep -A 1 Selector
nodeSelector:
  logging-infra-fluentd: "true"
```

**oc patch コマンドを使用して、daemonset の nodeSelector を変更します。**

```
$ oc patch ds logging-fluentd -p '{"spec":{"template":{"spec":{"nodeSelector":{"nonexistlabel":"true"}}}}}'
```

**Fluentd ノードセレクターを取得します。**

```
$ oc get ds logging-fluentd -o yaml |grep -A 1 Selector
nodeSelector:
  "nonexistlabel: "true"
```

Elasticsearch のサイズをゼロから元に戻します。

```
$ oc scale --replicas=# dc/<ELASTICSEARCH_DC>
```

daemonset 設定の `nodeSelector` を変更して `logging-infra-fluentd: "true"` に戻します。

**oc patch コマンドを使用して、daemonset の nodeSelector を変更します。**

```
oc patch ds logging-fluentd -p '{"spec":{"template":{"spec":{"nodeSelector":{"logging-infra-fluentd":"true"}}}}}'
```

### Buffer Chunk Limit の調整

Fluentd ロガーが多数のログを処理できない場合、メモリーの使用量を減らし、データ損失を防ぐためにファイルバッファリングに切り換える必要があります。

Fluentd `buffer_chunk_limit` は、デフォルト値が **8m** の環境変数 `BUFFER_SIZE_LIMIT` によって決定されます。出力ごとのファイルのバッファサイズは、デフォルト値が **256Mi** の環境変数 `FILE_BUFFER_LIMIT` によって決定されます。永続的なボリュームサイズは、`FILE_BUFFER_LIMIT` に出力を乗算した結果よりも大きくなければなりません。

Fluentd および Mux Pod では、永続ボリューム `/var/lib/fluentd` は PVC または `hostmount` などによって作成される必要があります。その領域はファイルバッファに使用されます。

`buffer_type` および `buffer_path` は、以下のように Fluentd 設定ファイルで設定されます。

```
$ egrep "buffer_type|buffer_path" *.conf
output-es-config.conf:
  buffer_type file
  buffer_path `var/lib/fluentd/buffer-output-es-config`
```

```
output-es-ops-config.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-output-es-ops-config`
filter-pre-mux-client.conf:
  buffer_type file
  buffer_path `/var/lib/fluentd/buffer-mux-client`
```

Fluentd `buffer_queue_limit` は変数 `BUFFER_QUEUE_LIMIT` の値です。この値はデフォルトで **32** になります。

環境変数 `BUFFER_QUEUE_LIMIT` は  $(\text{FILE\_BUFFER\_LIMIT} / (\text{number\_of\_outputs} * \text{BUFFER\_SIZE\_LIMIT}))$  として計算されます。

`BUFFER_QUEUE_LIMIT` 変数にデフォルトの値のセットが含まれる場合、以下のようになります。

- `FILE_BUFFER_LIMIT = 256Mi`
- `number_of_outputs = 1`
- `BUFFER_SIZE_LIMIT = 8Mi`

`buffer_queue_limit` の値は **32** になります。`buffer_queue_limit` を変更するには、`FILE_BUFFER_LIMIT` の値を変更する必要があります。

この数式では、`number_of_outputs` は、すべてのログが単一リソースに送信され、追加のリソースごとに 1 つずつ増分する場合に 1 になります。たとえば、`number_of_outputs` の値は以下のようになります。

- **1** - すべてのログが単一の Elasticsearch Pod に送信される場合
- **2** - アプリケーションログが Elasticsearch Pod に送信され、運用ログが別の Elasticsearch Pod に送信される場合
- **4** - アプリケーションログが Elasticsearch Pod に送信され、運用ログが別の Elasticsearch Pod に送信される場合で、それらがどちらも他の Fluentd インスタンスに転送される場合

### 36.5.4. Kibana

OpenShift Container Platform の Web コンソールから Kibana コンソールにアクセスするには、マスター `webconsole-config configmap` ファイルに `loggingPublicURL` パラメーターを追加し、Kibana コンソールの URL (`kibana-hostname` パラメーター) を指定します。値は HTTPS URL である必要があります。

```
...
clusterInfo:
  ...
  loggingPublicURL: "https://kibana.example.com"
  ...
```

`loggingPublicURL` パラメーターを設定すると、OpenShift Container Platform Web コンソールの **Browse** → **Pods** → `<pod_name>` → **Logs** タブに **View Archive** ボタンが作成されます。このボタンは Kibana コンソールにリンクします。



## 注記

有効なログイン cookie の期限が切れたら Kibana コンソールにログインする必要があります。以下のタイミングでログインする必要があります。

- 初回使用時
- ログアウト後

通常通り Kibana デプロイメントを拡張して冗長性を実現できます。

```
$ oc scale dc/logging-kibana --replicas=2
```



## 注記

ロギング Playbook の複数の実行にわたってスケールを維持するには、インベントリーファイルの **openshift\_logging\_kibana\_replica\_count** を必ず更新してください。

**openshift\_logging\_kibana\_hostname** 変数によって指定されたサイトにアクセスすることで、ユーザーインターフェイスを確認できます。

Kibana に関する詳細については、[Kibana のドキュメント](#) を参照してください。

## Kibana Visualize

Kibana Visualize を使用すると、視覚化機能やダッシュボードを作成してコンテナを監視できます。また Pod ログにより、管理者ユーザー (**cluster-admin** または **cluster-reader**) はデプロイメント、namespace、Pod、およびコンテナごとにログを表示することができます。

Kibana Visualize は Elasticsearch および ES-OPS Pod 内に存在し、それらの Pod 内で実行する必要があります。ダッシュボードとその他の Kibana UI オブジェクトを読み込むには、まずはダッシュボードに追加するユーザーとして Kibana にログインしてから、ログアウトする必要があります。これにより、以下の手順で使用するユーザーごとの必要な設定が作成されます。次に、以下を実行します。

```
$ oc exec <$espod> -- es_load_kibana_ui_objects <user-name>
```

ここで、**\$espod** はいずれかの Elasticsearch Pod の名前です。

カスタムフィールドを Kibana Visualize に追加するには、以下を実行します。

OpenShift Container Platform クラスターが、Elasticsearch **.operations.\*** または **project.\*** インデックスに定義されていないカスタムフィールドを含むログを JSON 形式で生成する場合、カスタムフィールドが Kibana で利用できないため、これらのフィールドでビジュアライゼーションを作成することはできません。

ただし、カスタムフィールドを Elasticsearch インデックスに追加できます。これにより、フィールドを Kibana インデックスパターンに追加して、Kibana Visualize で使用できます。



## 注記

カスタムフィールドは、テンプレートの更新後に作成されたインデックスにのみ適用されます。

カスタムフィールドを Kibana Visualize に追加するには、以下を実行します。

1. カスタムフィールドを Elasticsearch インデックステンプレートに追加します。
  - a. フィールドを追加する Elasticsearch インデックス (**.operations.\*** または **project.\*** インデックス) を判別します。カスタムフィールドが含まれる特定のプロジェクトがある場合は、そのプロジェクトの特定のインデックスにフィールドを追加します (例: **project.this-project-has-time-fields.\***)。
  - b. 以下のようなカスタムフィールドの JSON ファイルを作成します。以下に例を示します。

```
{
  "order": 20,
  "mappings": {
    "_default_": {
      "properties": {
        "mytimefield1": { ❶
          "doc_values": true,
          "format": "yyyy-MM-dd HH:mm:ss,SSSZ||yyyy-MM-dd'T'HH:mm:ss.SSSSSSZ||yyyy-MM-dd'T'HH:mm:ssZ||dateOptionalTime",
          "index": "not_analyzed",
          "type": "date"
        },
        "mytimefield2": {
          "doc_values": true,
          "format": "yyyy-MM-dd HH:mm:ss,SSSZ||yyyy-MM-dd'T'HH:mm:ss.SSSSSSZ||yyyy-MM-dd'T'HH:mm:ssZ||dateOptionalTime",
          "index": "not_analyzed",
          "type": "date"
        }
      }
    }
  },
  "template": "project.<project-name>.*" ❷
}
```

- ❶ カスタムフィールドおよびパラメーターを追加します。
- ❷ **.operations.\*** または **project.\*** インデックスを指定します。

- c. **openshift-logging** プロジェクトに切り替えます。

```
$ oc project openshift-logging
```

- d. Elasticsearch Pod のいずれかの名前を取得します。

```
$ oc get -n logging pods -l component=es
```

NAME	READY	STATUS	RESTARTS	AGE	IP
logging-es-data-master-5av030lk-1-2x494	2/2	Running	0	38m	
154.128.0.80	ip-153-12-8-6.wef.internal	<none>			

- e. JSON ファイルを Elasticsearch Pod に読み込みます。

```
$ cat <json-file-name> | \ ❶
oc exec -n logging -i -c elasticsearch <es-pod-name> -- \ ❷
  curl -s -k --cert /etc/elasticsearch/secret/admin-cert \
  --key /etc/elasticsearch/secret/admin-key \
  https://localhost:9200/_template/<json-file-name> -XPUT -d@- | \ ❸
python -mjson.tool
```

❶❸ 作成した JSON ファイルの名前。

❷ Elasticsearch Pod の名前。

```
{
  "acknowledged": true
}
```

- f. 個別の OPS クラスタがある場合は、es-ops Elasticsearch Pod のいずれかの名前を取得します。

```
$ oc get -n logging pods -l component=es-ops

NAME                                READY   STATUS    RESTARTS   AGE   IP
NODE                                NOMINATED NODE
logging-es-ops-data-master-o7nhcbo4-5-b7stm  2/2     Running   0          38m   154.128.0.80 ip-153-12-8-6.wef.internal <none>
```

- g. JSON ファイルを es-ops Elasticsearch Pod に読み込みます。

```
$ cat <json-file-name> | \ ❶
oc exec -n logging -i -c elasticsearch <esops-pod-name> -- \ ❷
  curl -s -k --cert /etc/elasticsearch/secret/admin-cert \
  --key /etc/elasticsearch/secret/admin-key \
  https://localhost:9200/_template/<json-file-name> -XPUT -d@- | \ ❸
python -mjson.tool
```

❶❸ 作成した JSON ファイルの名前。

❷ OPS クラスタ Elasticsearch Pod の名前。

以下のような出力が表示されます。

```
{
  "acknowledged": true
}
```

- h. インデックスが更新されていることを確認します。

```
oc exec -n logging -i -c elasticsearch <es-pod-name> -- \ ❶
  curl -s -k --cert /etc/elasticsearch/secret/admin-cert \
  --key /etc/elasticsearch/secret/admin-key \
  https://localhost:9200/project.*/_search?sort=<custom-field>:desc | \ ❷
python -mjson.tool
```

- 1 Elasticsearch または OPS クラスター Elasticsearch Pod の名前。
- 2 追加したカスタムフィールドの名前。

このコマンドは、カスタムフィールドのインデックスレコードを降順でソートします。



### 注記

この設定は、既存のインデックスには適用されません。既存のインデックスに設定を適用する場合は、インデックスを再度実行します。

## 2. カスタムフィールドを Kibana に追加します。

- a. Elasticsearch コンテナから既存のインデックスパターンファイルを取得します。

```
$ mkdir index_patterns
$ cd index_patterns
$ oc project openshift-logging
$ for espod in $( oc get pods -l component=es -o jsonpath='{.items[*].metadata.name}' );
do
> for ff in $( oc exec -c elasticsearch <es-pod-name> -- ls
/usr/share/elasticsearch/index_patterns ) ; do
> oc exec -c elasticsearch <es-pod-name> -- cat
/usr/share/elasticsearch/index_patterns/$ff > $ff
> done
> break
> done
```

インデックスパターンファイルは、`/usr/share/elasticsearch/index_patterns` ディレクトリーにダウンロードされます。

以下に例を示します。

```
index_patterns $ ls
com.redhat.viaq-openshift.index-pattern.json
```

- b. 対応するインデックスパターンファイルを編集して、各カスタムフィールドの定義を **fields** 値に追加します。  
以下に例を示します。

```
{"count": 0, "name": "mytimefield2", "searchable": true, "aggregatable": true,
"readFromDocValues": true, "type": "date", "scripted": false},
```

ビジュアライゼーションで使用するには、定義に **"searchable": true**, と **"aggregatable": true**, のパラメーターを追加する必要があります。データ型は、上記に追加した Elasticsearch フィールド定義に対応する必要があります。たとえば、**number** タイプの Elasticsearch に **myfield** フィールドを追加した場合に、**myfield** は **string** タイプとして追加できません。

- c. インデックスパターンファイルで、Kibana インデックスパターンの名前をインデックスパターンファイルに追加します。  
たとえば、**operation.\*** インデックスパターンを使用するには、以下を実行します。

■

```
"title": "*operations.*"
```

project.MYNAMESPACE.\* インデックスパターンを使用するには、以下を実行します。

```
"title": "project.MYNAMESPACE.*"
```

- d. ユーザー名を特定し、ユーザー名のハッシュ値を取得します。インデックスパターンは、ユーザー名のハッシュを使用して保存されます。以下の2つのコマンドを実行します。

```
$ get_hash() {
> printf "%s" "$1" | sha1sum | awk '{print $1}'
> }
```

```
$ get_hash admin
```

```
d0aeb5660fc2140aec35850c4da997
```

- e. インデックスパターンファイルを Elasticsearch に適用します。

```
cat com.redhat.viaq-openshift.index-pattern.json | \ 1
oc exec -i -c elasticsearch <espod-name> -- es_util \
--query=".kibana.<user-hash>/index-pattern/<index>" -XPUT --data-binary @- | \ 2
python -mjson.tool
```

**1** インデックスパターンファイルの名前。

**2** ユーザーハッシュおよびインデックス (**.operations.\*** or **project.\***)。

以下に例を示します。

```
cat index-pattern.json | \
oc exec -i -c elasticsearch mypod-23-gb9pl -- es_util \
--query=".kibana.d0aeb5660fc2140aec35850c4da997/index-
pattern/project.MYNAMESPACE.*" -XPUT --data-binary @- | \
python -mjson.tool
```

以下のような出力が表示されます。

```
{
  "_id": ".operations.*",
  "_index": ".kibana.d0aeb5660fc2140aec35850c4da997",
  "_shards": {
    "failed": 0,
    "successful": 2,
    "total": 2
  },
  "_type": "index-pattern",
  "_version": 1,
  "created": true,
  "result": "created"
}
```

- f. カスタムフィールドの Kibana コンソールを終了して再起動し、**Available Fields** 一覧に表示されるように、**Management** → **Index Patterns** ページのフィールド一覧に表示されま

す。

### 36.5.5. Curator

Curator を利用することで、管理者はスケジュールされた Elasticsearch のメンテナンス操作を設定し、プロジェクトごとに自動的に実行することができます。これは、設定に基づいてアクションを毎日実行するようにスケジュール設定されています。Elasticsearch クラスターごとに1つの Curator Pod のみを使用することが推奨されます。Curator Pod は coronjob で指定された時間にのみ実行され、Pod は完了時に終了します。Curator は以下の構造を持つ YAML 設定ファイルで設定されます。



#### 注記

タイムゾーンは Curator Pod が実行されるホストノードに基づいて設定されます。

```
$PROJECT_NAME:
  $ACTION:
    $UNIT: $VALUE

$PROJECT_NAME:
  $ACTION:
    $UNIT: $VALUE
...
```

利用可能なパラメーターを以下に示します。

変数名	説明
<b>PROJECT_NAME</b>	プロジェクトの実際の名前 ( <b>myapp-devel</b> など)。OpenShift Container Platform の操作ログについては、名前 <b>.operations</b> をプロジェクト名として使用します。
<b>ACTION</b>	実行するアクション。現在許可されているのは <b>delete</b> のみです。
<b>UNIT</b>	<b>days</b> 、 <b>weeks</b> 、または <b>months</b> のいずれか。
<b>VALUE</b>	単位数を示す整数。
<b>.defaults</b>	<b>.defaults</b> を <b>\$PROJECT_NAME</b> として使用して、指定されていないプロジェクトのデフォルトを設定します。
<b>.regex</b>	プロジェクト名に一致する正規表現の一覧。
<b>pattern</b>	適切にエスケープされた有効な正規表現パターン。一重引用符で囲まれています。

たとえば、以下のように Curator を設定します。

- **1 day** を経過した **myapp-dev** プロジェクトのインデックスを削除する

- **1 week** を経過した **myapp-qe** プロジェクトのインデックスを削除する
- **8 weeks** を経過した **operations** ログを削除する
- **31 days** を経過したその他すべてのプロジェクトのインデックスを削除する
- '^project\..+\-dev.\*\$' 正規表現で一致した 2 日以上経過したインデックスを削除します。
- '^project\..+\-test.\*\$' 正規表現で一致した 3 日以上経過したインデックスを削除します。

以下を使用します。

```
config.yaml: |
  myapp-dev:
    delete:
      days: 1

  myapp-qe:
    delete:
      weeks: 1

  .operations:
    delete:
      weeks: 8

  .defaults:
    delete:
      days: 31

  .regex:
    - pattern: '^project\..+\-dev\..*$'
      delete:
        days: 1
    - pattern: '^project\..+\-test\..*$'
      delete:
        days: 2
```

### 重要

**months** を操作の **\$UNIT** として使用する場合、Curator は今月の当日ではなく、今月の最初の日からカウントを開始します。たとえば、今日が 4 月 15 日であり、現時点で 2 カ月を経過したインデックスを削除する場合 (`delete: months: 2`)、Curator は 2 月 15 日より古い日付のインデックスを削除するのではなく、2 月 1 日より古いインデックスを削除します。つまり、今月の最初の日付まで遡り、そこから 2 カ月遡ります。Curator で厳密な設定をする必要がある場合、最も適切な方法として日数 (例: **delete: days: 30**) を使用することができます。

#### 36.5.5.1. Curator Actions File の使用

OpenShift Container Platform カスタム設定ファイルフォーマットを設定すると、内部のインデックスが間違っ削除されることはなくなります。

**actions file** を使用するには、除外ルールを Curator 設定に追加してこれらのインデックスを維持します。必要なパターンすべてを手動で追加する必要があります。

```
actions.yaml: |
actions:

  action: delete_indices
  description: be careful!
  filters:
  - exclude: false
    kind: regex
    filtertype: pattern
    value: '^project\.myapp\..*$'
  - direction: older
    filtertype: age
    source: name
    timestring: '%Y.%m.%d'
    unit_count: 7
    unit: days
  options:
    continue_if_exception: false
    timeout_override: '300'
    ignore_empty_list: true

  action: delete_indices
  description: be careful!
  filters:
  - exclude: false
    kind: regex
    filtertype: pattern
    value: '^\.operations\..*$'
  - direction: older
    filtertype: age
    source: name
    timestring: '%Y.%m.%d'
    unit_count: 56
    unit: days
  options:
    continue_if_exception: false
    timeout_override: '300'
    ignore_empty_list: true

  action: delete_indices
  description: be careful!
  filters:
  - exclude: true
    kind: regex
    filtertype: pattern
    value: '^project\.myapp\..*$|^\.operations\..*$|^\.searchguard\..*$|^\.kibana$'
  - direction: older
    filtertype: age
    source: name
    timestring: '%Y.%m.%d'
    unit_count: 30
    unit: days
  options:
    continue_if_exception: false
    timeout_override: '300'
    ignore_empty_list: true
```

### 36.5.5.2. Curator 設定の作成

**openshift\_logging** Ansible ロールは、Curator が設定の読み取りに使用する ConfigMap を提供します。この ConfigMap を編集するか、または置き換えることで、Curator を再設定することができます。現時点で、Ops および非 Ops 両方の Curator インスタンスを設定するために **logging-curator** ConfigMap が使用されています。**.operations** 設定はすべてアプリケーションログ設定と同じ場所にあります。

1. Curator 設定ファイルを作成するには、デプロイされた ConfigMap で設定を編集します。

```
$ oc edit configmap/logging-curator
```

または cronjob からジョブを手動で作成します。

```
oc create job --from=cronjob/logging-curator <job_name>
```

- スクリプト化されたデプロイメントの場合、インストーラーによって作成された設定ファイルをコピーし、新規の OpenShift Container Platform カスタム設定を作成します。

```
$ oc extract configmap/logging-curator --keys=curator5.yaml,config.yaml --to=/my/config
edit /my/config/curator5.yaml
edit /my/config/config.yaml
$ oc delete configmap logging-curator ; sleep 1
$ oc create configmap logging-curator \
  --from-file=curator5.yaml=/my/config/curator5.yaml \
  --from-file=config.yaml=/my/config/config.yaml \
  ; sleep 1
```

- または、**Actions File** を使用している場合は、以下を実行します。

```
$ oc extract configmap/logging-curator --keys=curator5.yaml,actions.yaml --
to=/my/config
edit /my/config/curator5.yaml
edit /my/config/actions.yaml
$ oc delete configmap logging-curator ; sleep 1
$ oc create configmap logging-curator \
  --from-file=curator5.yaml=/my/config/curator5.yaml \
  --from-file=actions.yaml=/my/config/actions.yaml \
  ; sleep 1
```

次にスケジュールされたジョブがこの設定を使用します。

以下のコマンドを用いて cronjob を制御できます。

```
# suspend cronjob
oc patch cronjob logging-curator -p '{"spec":{"suspend":true}}'

# resume cronjob
oc patch cronjob logging-curator -p '{"spec":{"suspend":false}}'

# change cronjob schedule
oc patch cronjob logging-curator -p '{"spec":{"schedule":"0 0 * * *"}}' ❶
```

- ❶ **schedule** オプションは、**cron 形式** のスケジュールを受け入れます。

## 36.6. CLEANUP

デプロイメント中に生成されたものをすべて削除します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook [-i </path/to/inventory>] \
  playbooks/openshift-logging/config.yml \
  -e openshift_logging_install_logging=False
```

## 36.7. 外部 ELASTICSEARCH インスタンスへのログの送信

Fluentd は、Elasticsearch デプロイメント設定の **ES\_HOST**、**ES\_PORT**、**OPS\_HOST**、および **OPS\_PORT** 環境変数の値にログを送信します。アプリケーションログは **ES\_HOST** の宛先に、操作ログは **OPS\_HOST** の宛先に送信されます。



### 注記

AWS Elasticsearch インスタンスへのログの直接送信はサポートされていません。Fluentd [Secure Forward](#) を使用して、**fluent-plugin-aws-elasticsearch-service** プラグインで設定した制御対象の Fluentd のインスタンスにログを送信してください。

ログを特定の Elasticsearch インスタンスに送信するには、デプロイメント設定を編集して、上記の変数の値を必要なインスタンスに置き換えます。

```
$ oc edit ds/<daemon_set>
```

外部 Elasticsearch インスタンスにアプリケーションログと操作ログの両方を含めるには、**ES\_HOST** と **OPS\_HOST** を同じ宛先に設定して、**ES\_PORT** と **OPS\_PORT** にも同一の値があるようにします。

相互 TLS 設定のみがサポートされます。これは、提供される Elasticsearch インスタンスがサポートするためです。クライアントキー、クライアント証明書、および CA で、**logging-fluentd** シークレットに対して、パッチを適用するか、またはこのシークレット再作成します。



### 注記

指定された Kibana と Elasticsearch イメージを使用していない場合、同じマルチテナント機能は利用できず、データは特定のプロジェクトへのユーザーアクセスによる制限を受けません。

## 36.8. 外部 SYSLOG サーバーへのログの送信

**fluent-plugin-remote-syslog** プラグインをホストで使用して、ログを外部 syslog サーバーに送信します。

環境変数を **logging-fluentd** または **logging-mux** daemonset に設定します。

```
- name: REMOTE_SYSLOG_HOST 1
  value: host1
- name: REMOTE_SYSLOG_HOST_BACKUP
  value: host2
- name: REMOTE_SYSLOG_PORT_BACKUP
  value: 5555
```

- 1 必要なりモート syslog ホスト。各ホストで必須です。

これによって2つの宛先が作成されます。**host1** の syslog サーバーはデフォルトポート **514** でメッセージを受信し、**host2** は同じメッセージをポート **5555** で受信します。

または、独自のカスタム `fluent.conf` を **logging-fluentd** または **logging-mux** ConfigMap に設定できます。

#### Fluentd 環境変数

パラメーター	説明
<b>USE_REMOTE_SYSLOG</b>	デフォルトは <b>false</b> です。 <b>fluent-plugin-remote-syslog</b> gem を使用できるようにするには、 <b>true</b> に設定します。
<b>REMOTE_SYSLOG_HOST</b>	(必須) リモート syslog サーバーのホスト名または IP アドレス。
<b>REMOTE_SYSLOG_PORT</b>	接続先のポート番号。デフォルトは <b>514</b> です。
<b>REMOTE_SYSLOG_SEVERITY</b>	syslog の重大度を設定します。デフォルトは <b>debug</b> です。
<b>REMOTE_SYSLOG_FACILITY</b>	syslog ファシリティを設定します。デフォルトは <b>local0</b> です。
<b>REMOTE_SYSLOG_USE_RECORD</b>	デフォルトは <b>false</b> です。レコードの重大度フィールドおよびファシリティフィールドを使用して syslog メッセージに設定するには、 <b>true</b> に設定します。
<b>REMOTE_SYSLOG_REMOVE_TAG_PREFIX</b>	タグから接頭辞を削除します。デフォルトは "" (空) です。
<b>REMOTE_SYSLOG_TAG_KEY</b>	これが指定されている場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにタグを設定します。
<b>REMOTE_SYSLOG_PAYLOAD_KEY</b>	これが指定されている場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにペイロードを設定します。
<b>REMOTE_SYSLOG_TYPE</b>	トランスポート層プロトコルタイプを設定します。デフォルトは <b>syslog_buffered</b> になり、これにより、TCP プロトコルが設定されます。UDP に切り替えるには、これを <b>syslog</b> に設定します。



#### 警告

この実装は安全ではないため、接続にスヌーピングがないことを保証できる環境でのみ使用してください。

## Fluentd ロギング Ansible 変数

パラメーター	説明
<code>openshift_logging_fluentd_remote_syslog</code>	デフォルトは <b>false</b> に設定されます。fluent-plugin-remote-syslog gem を使用できるようにするには、 <b>true</b> に設定します。
<code>openshift_logging_fluentd_remote_syslog_host</code>	リモート syslog サーバーのホスト名または IP アドレス。必須です。
<code>openshift_logging_fluentd_remote_syslog_port</code>	接続先のポート番号。デフォルトは <b>514</b> です。
<code>openshift_logging_fluentd_remote_syslog_severity</code>	syslog の重大度を設定します。デフォルトは <b>debug</b> です。
<code>openshift_logging_fluentd_remote_syslog_facility</code>	syslog ファシリティを設定します。デフォルトは <b>local0</b> です。
<code>openshift_logging_fluentd_remote_syslog_use_record</code>	デフォルトは <b>false</b> に設定されます。レコードの重大度フィールドおよびファシリティフィールドを使用して syslog メッセージに設定するには、 <b>true</b> に設定します。
<code>openshift_logging_fluentd_remote_syslog_remove_tag_prefix</code>	タグから接頭辞を削除します。デフォルトは "" (空) です。
<code>openshift_logging_fluentd_remote_syslog_tag_key</code>	文字列が指定された場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにタグを設定します。
<code>openshift_logging_fluentd_remote_syslog_payload_key</code>	文字列が指定された場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにペイロードを設定します。

## Mux ロギング Ansible 変数

パラメーター	説明
<code>openshift_logging_mux_remote_syslog</code>	デフォルトは <b>false</b> に設定されます。fluent-plugin-remote-syslog gem を使用できるようにするには、 <b>true</b> に設定します。
<code>openshift_logging_mux_remote_syslog_host</code>	リモート syslog サーバーのホスト名または IP アドレス。必須です。
<code>openshift_logging_mux_remote_syslog_port</code>	接続先のポート番号。デフォルトは <b>514</b> です。

パラメーター	説明
<code>openshift_logging_mux_remote_syslog_severity</code>	syslog の重大度を設定します。デフォルトは <b>debug</b> です。
<code>openshift_logging_mux_remote_syslog_facility</code>	syslog ファシリティーを設定します。デフォルトは <b>local0</b> です。
<code>openshift_logging_mux_remote_syslog_use_record</code>	デフォルトは <b>false</b> に設定されます。レコードの重大度フィールドおよびファシリティーフィールドを使用して syslog メッセージに設定するには、 <b>true</b> に設定します。
<code>openshift_logging_mux_remote_syslog_remove_tag_prefix</code>	タグから接頭辞を削除します。デフォルトは "" (空) です。
<code>openshift_logging_mux_remote_syslog_tag_key</code>	文字列が指定された場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにタグを設定します。
<code>openshift_logging_mux_remote_syslog_payload_key</code>	文字列が指定された場合、このフィールドをキーとして使用してレコードを検索し、syslog メッセージにペイロードを設定します。

## 36.9. ELASTICSEARCH 管理操作の実行

ロギングバージョン 3.2.0 以降では、Elasticsearch と通信して管理操作を実行するのに使用する管理者の証明書、キー、CA は、`logging-elasticsearch` シークレット内にあります。



### 注記

これらが EFK インストールにあるかどうかを確認するには、以下のコマンドを実行します。

```
$ oc describe secret logging-elasticsearch
```

1. メンテナンスを実行しようとしているクラスター内にある Elasticsearch Pod に接続します。
2. Pod をクラスター内で検索するには、以下のいずれかのコマンドを使用します。

```
$ oc get pods -l component=es -o name | head -1
$ oc get pods -l component=es-ops -o name | head -1
```

3. Pod に接続します。

```
$ oc rsh <your_Elasticsearch_pod>
```

4. Elasticsearch コンテナに接続すると、[インデックス API のマニュアル](#) に従い、シークレットからマウントされた証明書を使用して Elasticsearch と通信することができます。

Fluentd では、インデックス形式 `project.{project_name}.{project_uuid}.YYYY.MM.DD` を使用してログを Elasticsearch に送信します。ここで、YYYY.MM.DD はログレコードの日付です。

たとえば uuid が `3b3594fa-2ccd-11e6-acb7-0eb6b35eae3` の `openshift-logging` プロジェクトから 2016 年 6 月 15 日以降のすべてのログを削除するには、以下のコマンドを実行します。

```
$ curl --key /etc/elasticsearch/secret/admin-key \
  --cert /etc/elasticsearch/secret/admin-cert \
  --cacert /etc/elasticsearch/secret/admin-ca -XDELETE \
  "https://localhost:9200/project.logging.3b3594fa-2ccd-11e6-acb7-0eb6b35eae3.2016.06.15"
```

## 36.10. EFK 証明書の再デプロイ

Ansible Playbook を使用して、インストール/アップグレード Playbook を実行することなく、EFK スタックの証明書のローテーションを実行できます。

この Playbook は現在の証明書ファイルを削除し、新規 EFK 証明書を生成し、証明書シークレットを更新し、Kibana および Elasticsearch を再起動してそれらのコンポーネントが更新された証明書で読み込まれるようにします。

EFK 証明書を再デプロイするには、以下を実行します。

1. Ansible Playbook を使用して EFK 証明書を再デプロイします。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook playbooks/openshift-logging/redeploy-certificates.yml
```

## 36.11. 集計されたロギングのドライバーの変更

集計されたロギングについては、**json-file** ログドライバーの使用を推奨します。



### 重要

**json-file** ドライバーを使用する場合は、Docker バージョン `docker-1.12.6-55.gitc4618fb.el7_4 now` 以降を使用していることを確認してください。

Fluentd では、`/etc/docker/daemon.json` ファイルおよび `/etc/sysconfig/docker` ファイルをチェックして、Docker が使用しているドライバーを判別します。

**docker info** コマンドを使用すると、Docker が使用しているドライバーを確認できます。

```
# docker info | grep Logging
Logging Driver: journald
```

**json-file** に変更するには、以下の手順に従います。

1. `/etc/sysconfig/docker` ファイルまたは `/etc/docker/daemon.json` ファイルのいずれかを変更します。  
以下に例を示します。

```
# cat /etc/sysconfig/docker
OPTIONS=' --selinux-enabled --log-driver=json-file --log-opt max-size=1M --log-opt max-
file=3 --signature-verification=False'

cat /etc/docker/daemon.json
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "1M",
    "max-file": "1"
  }
}
```

2. Docker サービスを再起動します。

```
systemctl restart docker
```

3. Fluentd を再起動します。



#### 警告

13 以上のノードで一度に Fluentd を再起動すると、Kubernetes スケジューラーに大きな負荷が生成されます。以下の手順に従って Fluentd を再起動する場合は、細心の注意を払ってください。

Fluentd を再起動する方法は 2 つあります。1 つのノードまたはノードセット上で Fluentd を再起動するか、またはすべてのノードで Fluentd を再起動できます。

- a. 以下の手順は、1 つのノードまたはノードセット上で Fluentd を再起動する方法を示しています。

- i. Fluentd が実行しているノードの一覧を表示します。

```
$ oc get nodes -l logging-infra-fluentd=true
```

- ii. 各ノードについて、ラベルを削除して Fluentd をオフにします。

```
$ oc label node $node logging-infra-fluentd-
```

- iii. Fluentd がオフになっていることを確認します。

```
$ oc get pods -l component=fluentd
```

- iv. 各ノードについて Fluentd を再起動します。

```
$ oc label node $node logging-infra-fluentd=true
```

- b. 以下の手順は、すべてのノード上で Fluentd を再起動する方法を示しています。

- i. すべてのノードで Fluentd をオフにします。

```
$ oc label node -l logging-infra-fluentd=true --overwrite logging-infra-fluentd=false
```

- ii. Fluentd がオフになっていることを確認します。

```
$ oc get pods -l component=fluentd
```

- iii. すべてのノードで Fluentd を再起動します。

```
$ oc label node -l logging-infra-fluentd=false --overwrite logging-infra-fluentd=true
```

- iv. Fluentd がオンになっていることを確認します。

```
$ oc get pods -l component=fluentd
```

## 36.12. ELASTICSEARCH の手動ロールアウト

OpenShift Container Platform 3.7 では、集計ロギングスタックで Elasticsearch Deployment Config オブジェクトが更新され、Config Change Trigger が除外されました。このため、**dc** を変更しても自動ロールアウトは実行されなくなります。これは、Elasticsearch クラスターで意図しない再起動を回避するものでしたが、クラスターメンバーの再起動時にシャードのリバランスが過剰に発生しまう可能性があります。

このセクションでは、[ローリング再起動](#) と [フル再起動](#) の 2 つの再起動手順を説明します。ローリング再起動はダウンタイムを発生させずに Elasticsearch クラスターに適切な変更を適用し (3 つのマスターが設定されている場合)、フル再起動は既存データを損なわずに大規模な変更を安全に適用します。

### 36.12.1. Elasticsearch ローリングクラスター再起動の実行

以下のいずれかの変更を行う場合は、ローリング再起動を推奨します。

- Elasticsearch Pod の実行で再起動が必要なノード
- logging-elasticsearch configmap
- logging-es-\* デプロイメント設定
- 新規イメージのデプロイメントまたはアップグレード

これは今後推奨される再起動ポリシーになります。



#### 注記

**openshift\_logging\_use\_ops** が **True** に設定される場合、Elasticsearch クラスターへのアクションを ops クラスターに対して繰り返す必要があります。

1. ノードを意図的に停止する際のシャードのバランシングを防止します。

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> -- \
  curl -s \
  --cacert /etc/elasticsearch/secret/admin-ca \
  --cert /etc/elasticsearch/secret/admin-cert \
```

```
--key /etc/elasticsearch/secret/admin-key \
-XPUT 'https://localhost:9200/_cluster/settings' \
-d '{ "transient": { "cluster.routing.allocation.enable" : "none" } }'
```

- 完了したら、Elasticsearch クラスターのそれぞれの **dc** について **oc rollout latest** を実行して、最新バージョンの **dc** オブジェクトをデプロイします。

```
$ oc rollout latest <dc_name>
```

新しい Pod がデプロイされます。Pod に 2 つのコンテナが準備できたら、以下の **dc** に進むことができます。

- クラスターのすべての dc がロールアウトされたら、シャードバランシングを再度有効にします。

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> -- \
  curl -s \
  --cacert /etc/elasticsearch/secret/admin-ca \
  --cert /etc/elasticsearch/secret/admin-cert \
  --key /etc/elasticsearch/secret/admin-key \
  -XPUT 'https://localhost:9200/_cluster/settings' \
  -d '{ "transient": { "cluster.routing.allocation.enable" : "all" } }'
```

### 36.12.2. Elasticsearch フルクラスター再起動の実行

Elasticsearch のメジャーバージョンの変更など、変更プロセス中にデータ整合性が損なわれる危険性がある変更の場合は、フル再起動を推奨します。



#### 注記

**openshift\_logging\_use\_ops** が **True** に設定される場合、Elasticsearch クラスターへのアクションを ops クラスターに対して繰り返す必要があります。



#### 注記

**logging-es-ops** サービスに変更を行う場合は、パッチとして代わりにコンポーネント "es-ops-blocked" および "es-ops" を使用します。

- Elasticsearch クラスターが停止しているときに、Elasticsearch クラスターへのすべての外部通信を無効にします。以下を実行して、非クラスターロギングサービス (**logging-es** や **logging-es-ops** など) を編集して、Elasticsearch Pod に一致しないようにします。

```
$ oc patch svc/logging-es -p '{"spec":{"selector":{"component":"es-blocked"},"provider":"openshift"}}'
```

- シャードの同期フラッシュを実行して、シャットダウン前にディスクへの書き込みを待機している保留中の操作がないようにします。

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> -- \
  curl -s \
  --cacert /etc/elasticsearch/secret/admin-ca \
```

```
--cert /etc/elasticsearch/secret/admin-cert \
--key /etc/elasticsearch/secret/admin-key \
-XPOST 'https://localhost:9200/_flush/synced'
```

3. ノードを意図的に停止する際のシャードのバランシングを防止します。

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> -- \
  curl -s \
  --cacert /etc/elasticsearch/secret/admin-ca \
  --cert /etc/elasticsearch/secret/admin-cert \
  --key /etc/elasticsearch/secret/admin-key \
  -XPUT 'https://localhost:9200/_cluster/settings' \
  -d '{"transient": {"cluster.routing.allocation.enable": "none"} }'
```

4. 完了したら、Elasticsearch クラスターのそれぞれの **dc** に対して、すべてのノードのスケールダウンを実行します。

```
$ oc scale dc <dc_name> --replicas=0
```

5. 縮小したら、Elasticsearch クラスターのそれぞれの **dc** について **oc rollout latest** を実行して、最新バージョンの **dc** オブジェクトをデプロイします。

```
$ oc rollout latest <dc_name>
```

新しい Pod がデプロイされます。Pod に 2 つのコンテナが準備できたら、以下の **dc** に進むことができます。

6. デプロイメントが完了したら、Elasticsearch クラスターのそれぞれの **dc** に対して、ノードのスケールアップを実行します。

```
$ oc scale dc <dc_name> --replicas=1
```

7. 拡大したら、ES クラスターへのすべての外部通信を有効にします。以下のコマンドを再度実行して、非クラスターロギングサービス (**logging-es** や **logging-es-ops** など) を編集して、Elasticsearch Pod に一致させるようにします。

```
$ oc patch svc/logging-es -p '{"spec":{"selector":{"component":"es","provider":"openshift"}}}'
```

## 36.13. EFK のトラブルシューティング

以下は、クラスターロギングデプロイメントに関連したよくある問題についてのトラブルシューティング情報です。

### 36.13.1. すべての EFK コンポーネントに関連するトラブルシューティング

以下のトラブルシューティングの問題は、通常は EFK スタックに適用されます。

#### デプロイメントが失敗し、ReplicationControllers が 0 にスケールされる

10 分のタイムアウトの前にインスタンスを正常に起動しないデプロイメントを実行する場合、OpenShift Container Platform はデプロイメントを失敗とみなし、ゼロインスタンスに縮小します。**oc get pods** コマンドはゼロ以外の終了コードを出して、デプロイされた Pod がない状態でデプロイヤー Pod を表示します。

以下の例では、Elasticsearch デプロイメントのデプロイヤー Pod が表示されています。これは、DeploymentConfig **logging-es-2e7ut0iq** のデプロイメントである ReplicationController **logging-es-2e7ut0iq-1** から取られています。

```
NAME                READY  STATUS             RESTARTS  AGE
logging-es-2e7ut0iq-1-deploy  1/1    ExitCode:255      0          1m
```

デプロイメントの失敗は、イメージのプルに時間がかかりすぎたり、ノードが反応しないなどの数多くの一時的な理由によって生じる可能性があります。

デプロイヤー Pod で考えられる理由について調べるか、または再デプロイを試行します。

```
$ oc deploy --latest logging-es-2e7ut0iq
```

または、既存デプロイメントの拡大を試行します。

```
$ oc scale --replicas=1 logging-es-2e7ut0iq-1
```

問題が残る場合は、Pod、イベント、および systemd ユニットログを調べて問題の原因を判別します。

#### kubernetes.default.svc.cluster.local を解決できない

マスターのこの内部エイリアスは、マスター上の組み込まれている DNS サーバーによって解決可能でなければなりません。プラットフォームによっては、マスターに対して (たとえばコンテナ内で) **dig** コマンドを実行し、これを確認することができます。

```
$ dig kubernetes.default.svc.cluster.local @localhost
[...]
;; QUESTION SECTION:
;kubernetes.default.svc.cluster.local. IN A

;; ANSWER SECTION:
kubernetes.default.svc.cluster.local. 30 IN A 172.30.0.1
```

古いバージョンのクラスターロギングは、マスターのこの内部エイリアスを自動的に定義しませんでした。集計ロギングを使用するには、クラスターをアップグレードする必要がある場合があります。クラスターが最新の状態にある場合、Pod がマスターの SkyDNS リゾルバーに達する際に問題が生じているか、または Pod の実行がブロックされている可能性があります。この問題を解決してから再度デプロイする必要があります。

#### マスターまたはサービスに接続できない

DNS 解決が何も返さないか、またはアドレスが (fluentd Pod などの) Pod から接続できない場合、これはシステムのファイアウォール/ネットワークの問題があることを示している可能性があります。この問題はデバッグする必要があります。

### 36.13.2. ElasticSearch に関連するトラブルシューティング

以下のトラブルシューティングの問題は、EFK スタックの ElasticSearch コンポーネントに適用されます。

#### Elasticsearch デプロイメントは成功せず、以前のバージョンにロールバックする

この状況は、クラスターロギングが AWS にデプロイされた状態で OpenShift Container Platform で生じます。通常 Elasticsearch Pod の記述は Pod ストレージの再割り当てに関する問題を示唆します。

```
$ oc describe pod <elasticsearch-pod>
```

AWS がストレージを利用可能にするためのより多くの時間を確保できるように、各 Elasticsearch デプロイメント設定にパッチを適用することについて検討してください。

```
$ oc patch dc <elasticsearch-deployment-config> -p '{"spec":{"strategy":{"recreateParams":{"timeoutSeconds":1800}}}'
```

### Searchguard インデックスが赤の状態のままになる

これは、デプロイメント設定ごとに1つのインデックスではなく、クラスターごとに単一の SearchGuard インデックスをアップグレードまたはこれに移行することに関する既知の問題です。Elasticsearch Explain API はこの理由を検知するために使用され、ノード割り当てに対してインデックスを削除することが必要です。

```
$ oc -c elasticsearch exec ${pod} -- es_util --query=".searchguard/_settings" -XPUT -d '{"index.routing.allocation.include._name": ""}'
```

### Elasticsearch Pod が準備状態にならない

これは初期化およびシードプロセスが失敗する場合の既知の問題であり、**.searchguard** インデックスが赤の状態にあることに関係する可能性があります。

```
for p in $(oc get pods -l component=es -o jsonpath={.items[*].metadata.name}); do \
  oc exec -c elasticsearch $p -- touch /opt/app-root/src/init_failures; \
done
```

## 36.13.3. Kibana

以下のトラブルシューティングの問題は EFK スタックの Kibana コンポーネントに適用されます。

### Kibana でのログインのループ

Kibana コンソールを起動し、ログインが成功した場合に、誤って Kibana にリダイレクトされ、すぐにログイン画面にリダイレクトされます。

この問題の原因として考えられる点として、Kibana の前にある OAuth2 プロキシが、これを有効なクライアントとして特定するためにシークレットをマスターの OAuth2 サーバーと共有する点があります。この問題は、シークレットが一致しないことを示唆している可能性があります。シークレットが一致するかどうかをプログラムを使用して確認できる方法はありません。

これは、ロギングを複数回デプロイする場合に生じる可能性があります。たとえば、初期のデプロイメントを修正しても、Kibana によって使用される **secret** が置き換えられる一方で、一致するマスター **oauthclient** エントリは置き換えられません。

以下を実行できます。

```
$ oc delete oauthclient/kibana-proxy
```

**openshift-ansible** の説明に従って **openshift\_logging** ロールを再実行します。これにより **oauthclient** は置き換えられ、次のログインではループは生じません。

```
**"error": "invalid request" on login*
```

## Kibana でのログインエラー

Kibana コンソールにアクセスしようとする際に、以下のブラウザーエラーが表示される場合があります。

```
{"error": "invalid_request", "error_description": "The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed."}
```

この問題は、OAuth2 クライアントとサーバー間の不一致が原因で発生します。ログイン後にサーバーが安全にリダイレクトできるように、クライアントのリターンアドレスがホワイトリストで指定されている必要があります。不一致がある場合、エラーメッセージが表示されます。

これは、直前のデプロイメントからの残りである **oauthclient** エントリーによって引き起こされる可能性があります。この場合、エントリーを置き換えることができます。

```
$ oc delete oauthclient/kibana-proxy
```

**openshift-ansible** の説明に従って **openshift\_logging** ロールを再実行します。これにより、**oauthclient** エントリーが置き換えられます。Kibana コンソールに戻り、再びログインします。

問題が解決しない場合は、OAuth クライアントに一覧表示されている URL の Kibana にアクセスしていることを確認してください。この問題は、転送先ポート (標準の 443 HTTPS ポートではなく 1443 など) の URL にアクセスすることで発生する可能性があります。

**oauthclient** を編集してサーバーのホワイトリストを調整できます。

```
$ oc edit oauthclient/kibana-proxy
```

実際に使用しているアドレスを組み込むことが許可されているリダイレクト URI の一覧を編集します。保存し、終了した後に、エラーは解決するはずですが。

## Kibana のアクセスにより 503 エラーが表示される

Kibana コンソールを表示する時にプロキシーエラーが発生する場合は、2つの問題のうちのいずれかが原因である可能性があります。

- Kibana が Pod を認識していない可能性があります。ElasticSearch の起動が遅い場合、Kibana は ElasticSearch に到達しようとする際にエラーを出す可能性があり、この場合 Kibana はこれを有効であるとは見なしません。関連するサービスにエンドポイントがあるかどうかをチェックすることができます。

```
$ oc describe service logging-kibana
Name:          logging-kibana
[...]
Endpoints:    <none>
```

Kibana Pod が有効である場合に、エンドポイントは一覧表示されます。有効でない場合は、Kibana Pod およびデプロイメントの状態を確認してください。

- Kibana サービスにアクセスするための名前付きルートはマスクされている可能性があります。これは、あるプロジェクトでテストデプロイメントを実行し、次に最初のデプロイメントを完全に削除することなく別のプロジェクトでデプロイした場合に発生する可能性があります。複数のルートが同じ宛先に送信される場合、デフォルトルーターのみが最初に作成された宛先に

ルーティングされます。問題が発生するルートをチェックして、そのルートが複数の場所で定義されているかどうかを確認してください。

```
$ oc get route --all-namespaces --selector logging-infra=support
NAMESPACE NAME      HOST/PORT          PATH    SERVICE
logging  kibana    kibana.example.com  logging-kibana
logging  kibana-ops kibana-ops.example.com logging-kibana-ops
```

この例では、重複するルートがありません。

## 第37章 集計ロギングのサイジングに関するガイドライン

### 37.1. 概要

Elasticsearch、Fluentd、Kibana (EFK) スタックは、OpenShift Container Platform インストール内部で実行されるノードとアプリケーションからログを集計します。デプロイされると、Fluentd を使用して、すべてのノード、Pod からのログを Elasticsearch (ES) に集計します。また、Kibana Web UI が一元化され、ユーザーと管理者は集計されたデータを使用して、多彩な視覚化機能およびダッシュボードを作成できます。

### 37.2. インストールシステム

集計ロギングスタックを OpenShift Container Platform にインストールする一般的な手順は、[コンテナログの集計](#) に記載されています。インストールガイドを参照する際にいくつかの重要な事項に留意してください。

ロギング Pod をクラスター上に均等に分散させるため、プロジェクトの作成時に空の [ノードセレクター](#) を使用する必要があります。

```
$ oc adm new-project logging --node-selector=""
```

これは、後で実行されるノードのラベリングと共に、ロギングプロジェクトへの Pod 配置を制御します。

Elasticsearch (ES) は、ノード障害に対する回復性を確保するために、少なくとも 3 つのクラスターサイズでデプロイする必要があります。これは `openshift_logging_es_cluster_size` パラメーターをイベントリーホストファイルに設定することで指定されます。

パラメーターの詳細の一覧については、[Ansible 変数](#) を参照してください。

Kibana には、アクセスに使用される任意のブラウザーから解決できるホスト名が必要です。たとえば、Kibana にラップトップで実行される Web ブラウザーからアクセスできるように Kibana の DNS エイリアスを企業名サービスに追加する必要がある場合があります。ロギングのデプロイでは、インフラストラクチャーノードの 1 つの Kibana にルートを作成するか、または OpenShift ルーターの実行中にルートを作成します。Kibana ホスト名のエイリアスはこのマシンを参照する必要があります。このホスト名は Ansible `openshift_logging_kibana_hostname` 変数として指定されます。

イメージがレジストリーからすでに取得されているかどうかや、クラスターのサイズによっては、インストールに時間がかかる場合があります。

`openshift-logging` プロジェクトの内部で、`oc get all` を実行してデプロイメントを確認できます。

```
$ oc get all
```

NAME	REVISION	REPLICAS	TRIGGERED BY
logging-curator	1	1	
logging-es-6cvk237t	1	1	
logging-es-e5x4t4ai	1	1	
logging-es-xmwvnrsv	1	1	
logging-kibana	1	1	
NAME	DESIRED	CURRENT	AGE
logging-curator-1	1	1	3d
logging-es-6cvk237t-1	1	1	3d
logging-es-e5x4t4ai-1	1	1	3d

NAME	HOST/PORT	PATH	SERVICE	TERMINATION
logging-es-xmwnorv-1	1	1	3d	
logging-kibana-1	1	1	3d	
logging-kibana	kibana.example.com		logging-kibana	reencrypt
component=support,logging-infra=support,provider=openshift				
logging-kibana-ops	kibana-ops.example.com		logging-kibana-ops	reencrypt
component=support,logging-infra=support,provider=openshift				
NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
logging-es	172.24.155.177	<none>	9200/TCP	3d
logging-es-cluster	None	<none>	9300/TCP	3d
logging-es-ops	172.27.197.57	<none>	9200/TCP	3d
logging-es-ops-cluster	None	<none>	9300/TCP	3d
logging-kibana	172.27.224.55	<none>	443/TCP	3d
logging-kibana-ops	172.25.117.77	<none>	443/TCP	3d
NAME	READY	STATUS	RESTARTS	AGE
logging-curator-1-6s7wy	1/1	Running	0	3d
logging-deployer-un6ut	0/1	Completed	0	3d
logging-es-6cvk237t-1-cnvw3	1/1	Running	0	3d
logging-es-e5x4t4ai-1-v933h	1/1	Running	0	3d
logging-es-xmwnorv-1-adr5x	1/1	Running	0	3d
logging-fluentd-156xn	1/1	Running	0	3d
logging-fluentd-40biz	1/1	Running	0	3d
logging-fluentd-8k847	1/1	Running	0	3d

最終的には以下のようなセットアップになります。

```
$ oc get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	NODE
logging-curator-1-6s7wy	1/1	Running	0	3d	ip-172-31-24-239.us-west-2.compute.internal
logging-deployer-un6ut	0/1	Completed	0	3d	ip-172-31-6-152.us-west-2.compute.internal
logging-es-6cvk237t-1-cnvw3	1/1	Running	0	3d	ip-172-31-24-238.us-west-2.compute.internal
logging-es-e5x4t4ai-1-v933h	1/1	Running	0	3d	ip-172-31-24-235.us-west-2.compute.internal
logging-es-xmwnorv-1-adr5x	1/1	Running	0	3d	ip-172-31-24-233.us-west-2.compute.internal
logging-fluentd-156xn	1/1	Running	0	3d	ip-172-31-24-241.us-west-2.compute.internal
logging-fluentd-40biz	1/1	Running	0	3d	ip-172-31-24-236.us-west-2.compute.internal
logging-fluentd-8k847	1/1	Running	0	3d	ip-172-31-24-237.us-west-2.compute.internal
logging-fluentd-9a3qx	1/1	Running	0	3d	ip-172-31-24-231.us-west-2.compute.internal
logging-fluentd-abvgj	1/1	Running	0	3d	ip-172-31-24-228.us-west-2.compute.internal
logging-fluentd-bh74n	1/1	Running	0	3d	ip-172-31-24-238.us-west-2.compute.internal
...					
...					

デフォルトでは、各 ES インスタンスに割り当てられる RAM の容量は 16 GB で

す。`openshift_logging_es_memory_limit` は `openshift-ansible` ホストインベントリーファイルで使われるパラメーターです。この値の半分が個々の `elasticsearch` Pod `java` プロセスの **ヒープサイズ** に渡されることに注意してください。

[EFK のインストールの詳細はこちら](#) を参照してください。

### 37.2.1. 大規模クラスター

ノードが 100 以上ある場合、最初に `docker pull registry.redhat.io/openshift3/logging-fluentd:v3.11` からロギングイメージを事前にプルすることを推奨します。ロギングインフラストラクチャー Pod (`Elasticsearch`、`Kibana`、`Curator`) をデプロイしたら、ノードのラベリングを一度に 20 ノード単位で実行します。以下に例を示します。

単純なループを使用します。

```
$ while read node; do oc label nodes $node logging-infra-fluentd=true; done < 20_fluentd.lst
```

以下の方法も有効です。

```
$ oc label nodes 10.10.0.{100..119} logging-infra-fluentd=true
```

ノードをグループ化してラベリングすると、OpenShift ロギングで `DaemonSet` が一定のペースで使用されるので、イメージレジストリーなどの共有リソース上の競合を回避できます。



#### 注記

`CrashLoopBackOff` | `ImagePullFailed` | `Error` の問題が発生したかどうかを確認します。`oc logs <pod>`、`oc describe pod <pod>` および `oc get event` は、役に立つ診断コマンドです。

## 37.3. SYSTEMD-JOURNALD と RSYSLOG

Red Hat Enterprise Linux (RHEL) 7 では、`systemd-journald.socket` ユニットはブートプロセスで `/dev/log` を作成し、入力を `systemd-journald.service` に渡します。すべての `syslog()` 呼び出しはジャーナルに移ります。

`systemd-journald` のデフォルト速度制限により、`Fluentd` がシステムログを読み取る前にシステムログがドロップします。これを防ぐには、以下を `/etc/systemd/journald.conf` ファイルに追加します。

```
# Disable rate limiting
RateLimitInterval=1s
RateLimitBurst=10000
Storage=volatile
Compress=no
MaxRetentionSec=30s
```

次に、サービスを再起動します。

```
$ systemctl restart systemd-journald.service
$ systemctl restart rsyslog.service
```

これらの設定は、一括アップロードのバースト性を担う設定です。

レート制限を削除した後、システムロギングデーモンの CPU 使用率が高くなる場合があります。以前はスロットルされていた可能性のあるメッセージが処理されるためです。

## 37.4. EFK ロギングのスケールアップ

必要なスケールを最初のデプロイメントで指定しなかった場合、影響を最小限に抑えてクラスターを調整するには、更新したパラメーター値 `openshift_logging_es_cluster_size` でインベントリーファイルを更新してから、Ansible ロギング Playbook を再度実行します。詳細については、[Elasticsearch 管理操作の実行](#) のセクションを参照してください。



### 注記

可用性の高い Elasticsearch 環境には 3 つ以上の Elasticsearch ノードが必要であり、それぞれを異なるホストに配置する必要があります。レプリカを作成するために、`openshift_logging_es_number_of_replicas` を 1 以上の値に設定します。

### 37.4.1. マスターイベントは、ログとして EFK に集計されます。

`eventrouter` Pod は Kubernetes API からイベントをスクレープし、`STDOUT` に出力します。`fluentd` プラグインはログメッセージを変換し、Elasticsearch (ES) に送信します。

`true` に設定して `openshift_logging_install_eventrouter` を有効にします。デフォルトは `off` です。`Eventrouter` がデフォルトの namespace にデプロイされます。収集された情報は ES の操作インデックスにあり、クラスター管理者のみに表示権限が割り当てられます。

## 37.5. ストレージに関する考慮事項

Elasticsearch インデックスはシャードとそれに対応するレプリカのコレクションです。これにより ES は高可用性を内部に実装しているので、ハードウェアベースのミラーリング RAID のバリエーションを使用する必要はほとんどありません。RAID 0 を使用して全体的なディスクパフォーマンスを向上させることは可能です。

[永続ボリューム](#) がそれぞれの Elasticsearch デプロイメント設定に追加されます。OpenShift Container Platform では、これは通常 [Persistent Volume Claim \(永続ボリューム要求\)](#) を使用して実行されます。

PVC の名前は `openshift_logging_es_pvc_prefix` 設定に基づいて指定されます。詳細は、[永続 Elasticsearch ストレージ](#) を参照してください。

Fluentd では `systemd journal` および `/var/lib/docker/containers/*.log` からのログを Elasticsearch に送信します。[詳細はこちらを参照してください](#)。

最適なパフォーマンスを得るには、ローカルの SSD ドライブの使用を推奨します。Red Hat Enterprise Linux (RHEL) 7 では、SATA ディスク以外のすべてのブロックデバイスについては `deadline` IO スケジューラーがデフォルトです。SATA ディスクについては、デフォルトの IO スケジューラーは `cfq` になります。

ES 用のストレージのサイジングは、インデックスの最適化方法によって大きく変わります。このため、必要なデータ量とアプリケーションログデータの集計方法を事前に検討しておく必要があります。一部の Elasticsearch ユーザーは、[絶対的なストレージ使用率をおよそ 50% に維持し、常に 70% 未満にする必要があることを確認](#) しています。これは、大規模なマージ操作を実行する際に Elasticsearch が応答しなくなる状態を避けるのに役立ちます。

## 第38章 クラスターメトリクスの有効化

### 38.1. 概要

kubelet はメトリクスを公開しますが、これは [Heapster](#) によって収集され、バックエンドに保存されます。

OpenShift Container Platform 管理者として、すべてのコンテナおよびコンポーネントのクラスターのメトリクスを1つのユーザーインターフェイスで表示できます。



#### 注記

以前のバージョンの OpenShift Container Platform は Heapster のメトリクスを使用して Horizontal Pod Autoscaler を設定しました。Horizontal Pod Autoscaler は現時点では OpenShift Container Platform メトリクスサーバーのメトリクスを使用するようになりました。詳細は、[Horizontal Pod Autoscaler の要件](#) を参照してください。

このトピックでは、[Hawkular Metric](#) をメトリクスエンジンとして使用した例について説明します。このエンジンはデータを [Cassandra](#) データベースに永続的に保存します。これが設定されると、CPU、メモリー、ネットワークベースのメトリクスを OpenShift Container Platform Web コンソールから表示できるようになります。

Heapster はマスターサーバーからすべてのノードの一覧を取得して、`/stats` エンドポイントから各ノードへ個別に通信します。ここから、Heapster は CPU、メモリー、ネットワーク使用状況のメトリクスを収集して、Hawkular Metrics にエクスポートします。

kubelet で利用できるストレージボリュームメトリクスは、`/stats` エンドポイントからは利用できませんが、`/metrics` エンドポイントから利用できます。詳細は、[Prometheus モニターリング](#) を参照してください。

Web コンソールで個々の Pod を参照すると、メモリーと CPU に個別のスパークラインチャートが表示されます。表示される時間範囲は選択可能で、これらのチャートは 30 秒ごとに自動更新されます。Pod に複数のコンテナがある場合は、メトリクスを表示する特定のコンテナを選択します。

[リソース制限](#) がプロジェクトに定義されている場合、各 Pod のドーナツチャートも表示できます。ドーナツチャートはリソース制限に対する使用量を示します。たとえば、**145 Available of 200 MiB** は、ドーナツチャートでは **55 MiB Used** と表示されます。

### 38.2. 操作を始める前に

Ansible Playbook はクラスターメトリクスのデプロイとアップグレードに使用できます。[クラスターのインストール](#) ガイドに精通しておくようにしてください。Ansible を使用するための予備知識や、設定に関する情報が記載されています。クラスターメトリクスのさまざまな領域を設定するためのパラメーターが Ansible インベントリーファイルに追加されています。

以下に示すセクションでは、デフォルト値を変更するために Ansible インベントリーファイルに追加できる各種の領域とパラメーターについて説明します。

### 38.3. メトリクスデータストレージ

メトリクスデータは、[永続ストレージ](#) または一時的な [Pod ボリューム](#) のいずれかに保存できます。

#### 38.3.1. 永続ストレージ

OpenShift Container Platform クラスターメトリクスを永続ストレージと共に実行すると、メトリクスは [永続ボリューム](#) に保存され、再起動または再作成される Pod を維持できます。これは、メトリクスデータをデータ損失から保護する必要がある場合に適しています。実稼働環境では、メトリクス Pod に永続ストレージを設定することを強く推奨します。

Cassandra ストレージのサイズ要件は Pod 数に依存します。セットアップで十分なサイズ要件を確保し、ディスクが一杯にならないように使用状況を監視するのは、管理者の責任です。永続ボリューム要求のサイズは `openshift_metrics_cassandra_pvc_size` [ansible variable](#) で指定され、デフォルトは 10 GB に設定されています。

[動的にプロビジョニングされた永続ボリューム](#)を使用する場合は、インベントリーファイルで `openshift_metrics_cassandra_storage_type` 変数を `dynamic` に設定します。

### 38.3.2. クラスターメトリクスのキャパシティーピニング

`openshift_metrics` Ansible ロールを実行した後、`oc get pods` の出力は以下のようになります。

```
# oc get pods -n openshift-infra
NAME                READY    STATUS    RESTARTS    AGE
hawkular-cassandra-1-l5y4g    1/1     Running   0           17h
hawkular-metrics-1t9so       1/1     Running   0           17h
heapster-febru             1/1     Running   0           17h
```

OpenShift Container Platform メトリクスは Cassandra データベースを使用して保存されます。このデータベースは `openshift_metrics_cassandra_limits_memory: 2G` の設定でデプロイされます。この値は Cassandra の開始スクリプトで決定される空きメモリー容量に応じてさらに調整できます。この値はほとんどの OpenShift Container Platform メトリクスインストールに対応しますが、クラスターメトリクスをデプロイする前に、環境変数を使用して `MAX_HEAP_SIZE` とヒープの新しい生成サイズ `HEAP_NEWSIZE` を Cassandra Dockerfile で変更できます。

デフォルトで、メトリクスデータは 7 日間保存されます。7 日が経過すると、Cassandra は最も古いメトリクスデータのページを開始します。削除された Pod とプロジェクトのメトリクスデータは自動的にページされません。7 日を超えたデータのみが削除されます。

#### 例38.110 のノードと 1000 の Pod による累積データ

10 のノードと 1000 の Pod を含むテストシナリオでは、24 時間で 2.5 GB のメトリクスデータが累積されました。このため、このシナリオでのメトリクスデータの容量計画の計算式は以下のようになります。

$$(((2.5 \times 10^9) \div 1000) \div 24) \div 10^6 = \sim 0.125 \text{ MB/hour per pod.}$$

#### 例38.2120 のノードと 10000 の Pod による累積データ

120 のノードと 10000 の Pod を含むテストシナリオでは、24 時間で 25 GB のメトリクスデータが累積されました。このため、このシナリオでのメトリクスデータの容量計画の計算式は以下のようになります。

$$(((11.410 \times 10^9) \div 1000) \div 24) \div 10^6 = 0.475 \text{ MB/hour}$$

	1000 の Pod	10000 の Pod
24 時間で累積される Cassandra ストレージデータ (デフォルトのメトリクスパラメーター)	2.5 GB	11.4 GB

**openshift\_metrics\_duration** の 7 日間および **openshift\_metrics\_resolution** の 30 秒間というデフォルト値を維持する場合、Cassandra Pod の週次のストレージ要件は以下のようになります。

	1000 の Pod	10000 の Pod
7 日間で累積される Cassandra ストレージデータ (デフォルトのメトリクスパラメーター)	20 GB	90 GB

先の表では、予期せずモニターリングされた Pod 使用量のバッファとして、予期されたストレージ容量に対して予備の 10% が追加されています。



#### 警告

Cassandra の永続化ボリュームに十分な空き容量がなくなると、データ損失が発生します。

クラスターメトリクスを永続ストレージと共に使用するには、永続ボリュームに **ReadWriteOnce** アクセスモードが設定されていることを確認します。このモードがアクティブではない場合、Persistent Volume Claim (永続ボリューム要求) は永続ボリュームを特定できず、Cassandra の開始に失敗します。

永続ストレージをメトリックコンポーネントと併用するには、十分なサイズの **永続ボリューム** があることを確認します。[Persistent Volume Claim \(永続ボリューム要求\)](#) の作成は OpenShift Ansible **openshift\_metrics** ロールによって処理されます。

OpenShift Container Platform メトリクスは、動的にプロビジョニングされた永続ボリュームもサポートします。この機能を OpenShift Container Platform メトリクスで使用するには、**openshift\_metrics\_cassandra\_storage\_type** の値を **dynamic** に設定する必要があります。EBS、GCE、Cinder ストレージバックエンドを使用すると **永続ボリュームを動的にプロビジョニング** できます。

パフォーマンスの設定およびクラスターメトリクス Pod のスケーリングについては、[Scaling Cluster Metrics](#) のトピックを参照してください。

表38.1 クラスター内のノード/Pod の数に基づく Cassandra データベースのストレージ要件

ノード数	Pod 数	Cassandra ストレージの増加速度	1日あたりの Cassandra ストレージの増加量	1週間あたりの Cassandra ストレージの増加量
210	10500	1時間あたり 500 MB	15 GB	75 GB
990	11000	1時間に 1GB	30 GB	210 GB

上記の計算では、ストレージ要件が計算値を超過しないようにするためのオーバーヘッドとして、予想されたサイズのおよそ 20% が追加されています。

**METRICS\_DURATION** と **METRICS\_RESOLUTION** の値がデフォルト (それぞれ 7 日間と 15 秒間) に維持されている場合、1 週間の Cassandra ストレージサイズ要件を上記の値に設定することを問題なく計画できるでしょう。



### 警告

OpenShift Container Platform メトリクスは、メトリクスデータのデータストアとして Cassandra データベースを使用するので、メトリクス設定のプロセスで **USE\_PERSISTANT\_STORAGE=true** に設定した場合には、NFS でネットワークストレージの上層に **PV** がデフォルトで配置されます。ただし、[Cassandra ドキュメント](#) にあるように、ネットワークストレージと、Cassandra を組み合わせて使用することは推奨していません。

### 既知の問題と制限

テストの結果として、**heapster** メトリクスコンポーネントは最大 25,000 の Pod を処理できることが確認されています。Pod 数がこの数を超過すると、Heapster のメトリクス処理が遅くなり始め、メトリクスグラフが表示されなくなる可能性があります。Heapster がメトリクスを収集できる Pod 数を増加する作業と、メトリクスを収集する代替ソリューションのアップストリーム開発が進められています。

### 38.3.3. 非永続ストレージ

OpenShift Container Platform クラスターメトリクスを非永続ストレージと共に実行すると、Pod の削除時に、保存されていたすべてのメトリクスが削除されます。クラスターメトリクスは非永続データで実行する方がはるかに容易ですが、非永続データで実行すると永続的なデータが損失するリスクが伴います。ただし、メトリクスはコンテナが再起動されても存続します。

非永続ストレージを使用するためには、インベントリーファイルで **openshift\_metrics\_cassandra\_storage\_type** 変数を **emptyDir** に設定する必要があります。



### 注記

非永続ストレージを使用している場合、メトリクスデータは、Cassandra Pod が実行されるノード上の **/var/lib/origin/openshift.local.volumes/pods** に書き込まれます。**/var** にメトリクスを保存するために十分な空き容量があることを確認してください。

## 38.4. メトリクス ANSIBLE ロール

OpenShift Container Platform の Ansible **openshift\_metrics** ロールは、[Ansible の設定](#) のインベントリーファイルにある変数を使用して、すべてのメトリクスコンポーネントを設定し、デプロイします。

### 38.4.1. メトリクス Ansible 変数の指定

OpenShift Ansible に含まれる **openshift\_metrics** ロールはクラスターメトリクスをデプロイするタスクを定義します。以下は、上書きが必要な場合にインベントリーファイルに追加できるロール変数の一覧です。

表38.2 Ansible 変数

変数	説明
<b>openshift_metrics_install_metrics</b>	<b>true</b> の場合にメトリクスをデプロイします。そうでない場合はアンデプロイします。
<b>openshift_metrics_start_cluster</b>	コンポーネントをデプロイした後にメトリクスクラスターを起動します。
<b>openshift_metrics_startup_timeout</b>	再起動を試行する前に Hawkular Metrics および Heapster が起動するまでの時間 (秒単位)。
<b>openshift_metrics_duration</b>	メトリクスがパージされるまで保管される日数。
<b>openshift_metrics_resolution</b>	メトリクスが収集される頻度。数値と時間を示す識別子である秒 (s)、分 (m)、時間 (h) で定義されます。
<b>openshift_metrics_cassandra_pvc_name</b>	この変数を使用して、使用する Cassandra ボリュームを正確な名前で指定します。指定の名前のボリュームが存在しない場合は作成されます。この変数は、Cassandra レプリカ1つとだけ併用できます。複数の Cassandra レプリカの場合は、代わりに変数 <b>openshift_metrics_cassandra_pvc_prefix</b> を使用します。
<b>openshift_metrics_cassandra_pvc_prefix</b>	Cassandra に作成される Persistent Volume Claim (永続ボリューム要求) の接頭辞。接頭辞には 1 から始まる連番が付加されます。
<b>openshift_metrics_cassandra_pvc_size</b>	各 Cassandra ノードの Persistent Volume Claim (永続ボリューム要求) のサイズ。
<b>openshift_metrics_cassandra_pvc_storage_class_name</b>	使用するストレージクラスを指定します。明示的にストレージクラスを設定する場合には、 <b>openshift_metrics_cassandra_storage_type</b> も設定します。

変数	説明
<b>openshift_metrics_cassandra_storage_type</b>	<b>emptyDir</b> を一時ストレージとして使用します (テスト用)。 <b>pv</b> を永続ボリュームとして使用します。これはインストール前に作成する必要があります。または <b>dynamic</b> を動的永続ボリュームとして使用します。ストレージクラスを明示的に設定する必要がある場合は、 <b>pv</b> を指定し、 <b>openshift_metrics_cassandra_pvc_storage_class_name</b> を設定します。
<b>openshift_metrics_cassandra_replicas</b>	メトリクススタックの Cassandra ノードの数。この値は Cassandra レプリケーションコントローラーの数を指定します。
<b>openshift_metrics_cassandra_limits_memory</b>	Cassandra Pod のメモリ制限。たとえば、値 <b>2Gi</b> にすると Cassandra のメモリは 2 GB に制限されます。この値は開始スクリプトで、スケジュールされているノードの空きメモリ容量に基づいてさらに調整できます。
<b>openshift_metrics_cassandra_limits_cpu</b>	Cassandra Pod の CPU 制限。値 <b>4000m</b> (4000 ミリコア) の場合、Cassandra は 4 基の CPU に制限されます。
<b>openshift_metrics_cassandra_requests_memory</b>	Cassandra Pod について要求するメモリ量。値 <b>2Gi</b> の場合、2 GB のメモリが要求されます。
<b>openshift_metrics_cassandra_requests_cpu</b>	Cassandra Pod の CPU 要求。値 <b>4000m</b> (4000 ミリコア) の場合、4 基の CPU が要求されます。
<b>openshift_metrics_cassandra_storage_group</b>	Cassandra に使用される補助ストレージグループ。
<b>openshift_metrics_cassandra_nodeselector</b>	必要な既存の <a href="#">ノードセレクター</a> を設定して、Pod が特定のラベルを持つノードに配置されるようにします。たとえば、 <b>{"node-role.kubernetes.io/infra":"true"}</b> とします。指定されていない場合には、Cassandra Pod は任意のスケジュール可能なノードにデプロイされます。
<b>openshift_metrics_hawkular_ca</b>	Hawkular 証明書に署名するための認証局 (CA) ファイル (オプション)。

変数	説明
<b>openshift_metrics_hawkular_cert</b>	Hawkular メトリクスへのルートを再暗号化するために使用される証明書ファイル。証明書にはルートに使用されるホスト名を含める必要があります。これが指定されない場合、デフォルトのルーター証明書が使用されます。
<b>openshift_metrics_hawkular_key</b>	Hawkular 証明書で使用されるキーファイル。
<b>openshift_metrics_hawkular_limits_memory</b>	Hawkular Pod を制限するためのメモリー量。値 <b>2Gi</b> の場合、Hawkular Pod は 2 GB のメモリーに制限されます。この値は開始スクリプトで、スケジュールされているノードの空きメモリー容量に基づいてさらに調整できます。
<b>openshift_metrics_hawkular_limits_cpu</b>	Hawkular Pod の CPU 制限。値 <b>4000m</b> (4000 ミリコア) の場合、Hawkular Pod は 4 基の CPU に制限されます。
<b>openshift_metrics_hawkular_replicas</b>	Hawkular メトリクスのレプリカの数。
<b>openshift_metrics_hawkular_requests_memory</b>	Hawkular Pod について要求するメモリー量。値 <b>2Gi</b> の場合、2 GB のメモリーが要求されます。
<b>openshift_metrics_hawkular_requests_cpu</b>	Hawkular Pod の CPU 要求。値 <b>4000m</b> (4000 ミリコア) の場合、4 基の CPU が要求されます。
<b>openshift_metrics_hawkular_nodeselector</b>	必要な既存の <a href="#">ノードセレクター</a> を設定して、Pod が特定のラベルを持つノードに配置されるようにします。たとえば、 <code>{"node-role.kubernetes.io/infra":"true"}</code> とします。指定されていない場合には、Hawkular Pod は任意のスケジュール可能なノードにデプロイされます。
<b>openshift_metrics_heapster_allowed_users</b>	許可する CN のコンマ区切りの一覧。デフォルトで、OpenShift サービスプロキシが接続できるように設定されます。 <b>Horizontal Pod Autoscaling</b> を適切に機能させるには、上書きする際に <a href="#">system:master-proxy</a> を一覧に追加します。
<b>openshift_metrics_heapster_limits_memory</b>	Heapster Pod を制限するメモリー量。値 <b>2Gi</b> の場合、Heapster Pod は 2 GB のメモリーに制限されます。
<b>openshift_metrics_heapster_limits_cpu</b>	Heapster Pod の CPU 制限。値 <b>4000m</b> (4000 ミリコア) の場合、Heapster Pod は 4 基の CPU に制限されます。
<b>openshift_metrics_heapster_requests_memory</b>	Heapster Pod について要求するメモリー量。値 <b>2Gi</b> の場合、2 GB のメモリーが要求されます。

変数	説明
<code>openshift_metrics_heapster_requests_cpu</code>	Heapster Pod の CPU 要求。値 <b>4000m</b> (4000 ミリコア) の場合、4 基の CPU が要求されます。
<code>openshift_metrics_heapster_standalone</code>	Heapster のみをデプロイします。Hawkular Metrics や Cassandra コンポーネントはデプロイされません。
<code>openshift_metrics_heapster_nodeselector</code>	必要な既存の <a href="#">ノードセレクター</a> を設定して、Pod が特定のラベルを持つノードに配置されるようにします。たとえば、 <code>{"node-role.kubernetes.io/infra":"true"}</code> とします。これが指定されていない場合には、Heapster Pod は任意のスケジュール可能なノードにデプロイされます。
<code>openshift_metrics_hawkular_hostname</code>	Hawkular Metrics <a href="#">ルート</a> のホスト名を使用するので、 <code>openshift_metrics</code> Ansible ロールを実行する場合に設定します。この値は、完全修飾ドメイン名と対応している必要があります。

要求と制限を指定する方法の詳細については、[コンピュータリソース](#) を参照してください。

Cassandra で [永続ストレージ](#) を使用している場合、`openshift_metrics_cassandra_pvc_size` 変数を使用してクラスターに十分なディスクサイズを設定することは管理者の責任です。また、管理者はディスクが一杯にならないようにディスク使用量を監視する必要もあります。



#### 警告

Cassandra の永続化ボリュームの領域が不足するとデータが損失します。

その他のすべての変数はオプションで、詳細なカスタマイズが可能です。たとえば、Kubernetes マスターを `https://kubernetes.default.svc:443` で使用できないカスタムインストールでは、代わりに `openshift_metrics_master_url` パラメーターを使用して値を指定できます。

### 38.4.2. シークレットの使用

OpenShift Container Platform Ansible `openshift_metrics` ロールは、コンポーネント間で使用する自己署名証明書を自動生成し、[re-encrypt ルート](#) を生成して Hawkular Metrics サービスを公開します。このルートによって Web コンソールで Hawkular Metrics サービスにアクセスできます。

Web コンソールを実行するブラウザーがこのルート経由の接続を信頼するには、ルートの証明書を信頼する必要があります。これは、信頼された認証局によって署名された [ユーザー固有の証明書を提供](#) することで実行できます。`openshift_metrics` ロールによって、ユーザー固有の証明書を指定でき、この証明書がルートの作成時に使用されます。

ユーザー固有の証明書を提供しない場合、ルーターのデフォルト証明書が使用されます。

### 38.4.2.1. ユーザー固有の証明書の提供

ユーザー固有の証明書を提供し、`re-encrypt` ルートで使用するには、`openshift_metrics_hawkular_cert`、`openshift_metrics_hawkular_key`、`openshift_metrics_hawkular_ca` 変数をインベントリーファイルで設定します。

`hawkular-metrics.pem` 値には `.pem` 形式の証明書を含める必要があります。この `pem` ファイルに `hawkular-metrics-ca.cert` シークレット経由で署名した認証局の証明書を提供することも必要です。

詳細については、[re-encrypt ルートに関するマニュアル](#) を参照してください。

## 38.5. メトリックコンポーネントのデプロイ

すべてのメトリックコンポーネントのデプロイと設定は OpenShift Container Platform Ansible で処理されるので、すべてを1つのステップでデプロイできます。

以下の例では、デフォルトパラメーターを使用して、メトリクスのデプロイ時に永続ストレージを使用する場合の方法と使用しない場合の方法を示しています。



### 重要

Ansible Playbook を実行するホストには、ホストあたり 75MiB 以上の空きメモリーがインベントリーで必要になります。



### 重要

アップストリームの Kubernetes ルールに応じて、`eth0` のデフォルトインターフェイスでのみメトリクスを収集できます。

### 例38.3 永続ストレージを使用するデプロイ

以下のコマンドでは、Hawkular Metrics ルートを `hawkular-metrics.example.com` を使用し、永続ストレージを使用してデプロイされるように設定します。

十分な容量を備えた永続ボリュームを用意してください。

```
$ ansible-playbook [-i </path/to/inventory>] <OPENSIFT_ANSIBLE_DIR>/playbooks/openshift-
metrics/config.yml \
-e openshift_metrics_install_metrics=True \
-e openshift_metrics_hawkular_hostname=hawkular-metrics.example.com \
-e openshift_metrics_cassandra_storage_type=pv
```

### 例38.4 永続ストレージを使用しないデプロイ

以下のコマンドでは、Hawkular Metrics ルートを `hawkular-metrics.example.com` を使用し、永続ストレージを使用しないでデプロイするように設定します。

```
$ ansible-playbook [-i </path/to/inventory>] <OPENSIFT_ANSIBLE_DIR>/playbooks/openshift-
metrics/config.yml \
-e openshift_metrics_install_metrics=True \
```

```
-e openshift_metrics_hawkular_hostname=hawkular-metrics.example.com
```



### 警告

これは永続ストレージを使用しないでデプロイされるため、メトリックデータが損失する可能性があります。

## 38.5.1. メトリクスの診断

メトリクススタックの状態の評価に使用できるメトリクス用の診断があります。メトリクスの診断を実行するには、以下のコマンドを実行します。

```
$ oc adm diagnostics MetricsApiProxy
```

## 38.6. メトリクスのパブリック URL の設定

OpenShift Container Platform Web コンソールは、Hawkular Metrics サービスから受信したデータを使用してグラフを表示します。Hawkular Metrics サービスにアクセスする URL は、[マスター webconsole-config configmap ファイル](#) で **metricsPublicURL** オプションを使用して設定する必要があります。この URL はメトリクスコンポーネントの [デプロイメント](#) 時に使用される **openshift\_metrics\_hawkular\_hostname** インベントリ変数で作成されるルートに対応します。



### 注記

**openshift\_metrics\_hawkular\_hostname** はコンソールにアクセスするブラウザで解決できる必要があります。

たとえば、**openshift\_metrics\_hawkular\_hostname** が **hawkular-metrics.example.com** に対応する場合、webconsole-config configmap ファイルに以下の変更を行う必要があります。

```
clusterInfo:
  ...
  metricsPublicURL: "https://hawkular-metrics.example.com/hawkular/metrics"
```

**webconsole-config** configmap ファイルを更新して保存した後に、Web コンソール Pod は liveness プロブ設定 (デフォルトで 30 秒) で設定された遅延後に再起動します。

OpenShift Container Platform サーバーが再び稼働すると、メトリクスが Pod 概要のページに表示されます。

### 注意

自己署名証明書を使用している場合、Hawkular Metrics サービスはコンソールとは別のホスト名でホストされ、別の証明書を使用することに注意してください。場合によっては、ブラウザタブを明示的に開いて **metricsPublicURL** に指定される値を直接参照して、この証明書を受け入れる必要があります。

これを回避するには、お使いのブラウザで許可されるように設定された証明書を使用します。

## 38.7. HAWKULAR METRICS への直接アクセス

メトリクスに直接アクセスし、これを管理するには、[Hawkular Metrics API](#) を使用します。



### 注記

API から Hawkular Metrics にアクセスした場合は、読み取り操作しか実行できません。メトリクスの書き込みはデフォルトで無効にされています。個々のユーザーがメトリクスの書き込みも実行できるようにするには、**openshift\_metrics\_hawkular\_user\_write\_access** 変数を **true** に設定する必要があります。

ただし、デフォルトの設定を使用して Heapster からのみメトリクスを入力ことを推奨します。書き込みアクセスが有効になると、どのユーザーもメトリクスをシステムに書き込めるようになり、これがパフォーマンスに影響を及ぼし、Cassandra のディスク使用量が予期せず増加する可能性があります。

[Hawkular Metrics のマニュアル](#) では、API の使用方法を説明していますが、OpenShift Container Platform で使用するよう設定されたバージョンの Hawkular Metrics とは処理方法に多少の違いがあります。

### 38.7.1. OpenShift Container Platform プロジェクトと Hawkular テナント

Hawkular Metrics はマルチテナントアプリケーションです。これは、OpenShift Container Platform のプロジェクトが Hawkular Metrics のテナントに対応するように設定されます。

このため、**MyProject** という名前のプロジェクトのメトリクスにアクセスする場合は、**Hawkular-Tenant** ヘッダーを **MyProject** に設定する必要があります。

また、**\_system** という名前の特殊なテナントもあり、これにはシステムレベルのメトリクスが含まれています。これにアクセスするには、**cluster-reader** または **cluster-admin** レベルの権限が必要です。

### 38.7.2. 承認

Hawkular Metrics サービスは、ユーザーを OpenShift Container Platform に対して認証し、ユーザーがアクセスしようとしているプロジェクトに対するアクセスを持つかどうかを判別します。

Hawkular Metrics はクライアントからベアラートークンを受け取り、**SubjectAccessReview** を使用して、OpenShift Container Platform サーバーでトークンを検証します。ユーザーがプロジェクトに対する適切な読み取り権限を持つ場合、ユーザーはこのプロジェクトのメトリクスを読み取ることができます。**\_system** テナントについては、このテナントからの読み取りを要求するユーザーには、**cluster-reader** パーミッションがなければなりません。

Hawkular Metrics API にアクセスする場合、ベアラートークンは **Authorization** ヘッダーに渡す必要があります。

## 38.8. OPENSIFT CONTAINER PLATFORM クラスターメトリクス POD のスケーリング

クラスターメトリクス機能のスケーリングに関する情報は、[Scaling and Performance Guide](#) に記載されています。

## 38.9. CLEANUP

OpenShift Container Platform Ansible **openshift\_metrics** ロールによってデプロイされたものはすべて、以下の手順を実行して削除できます。

```
$ ansible-playbook [-i </path/to/inventory>] <OPENSIFT_ANSIBLE_DIR>/playbooks/openshift-  
metrics/config.yml \  
-e openshift_metrics_install_metrics=False
```

## 第39章 WEB コンソールのカスタマイズ

### 39.1. 概要

管理者は拡張機能を使用して **Web コンソール** をカスタマイズできます。拡張機能でスクリプトを実行すると、Web コンソールを読み込む際にカスタムスタイルシートを読み込むことができます。拡張スクリプトによって、Web コンソールのデフォルト動作を上書きし、必要に応じてカスタマイズできます。

たとえば、拡張スクリプトを使用して、自社独自のブランディングを追加したり、自社固有の機能を追加したりすることができます。この使用例として一般的なのは、さまざまな環境のリブランディングやホワイトラベリングです。同じ拡張コードを使用しながら、Web コンソールを変更する設定を指定できます。

### 注意

Web コンソールのスタイルや動作に対して、以下に記載されていないような広範な変更を実行するには注意が必要です。スクリプトやスタイルシートを追加することはできますが、大幅にカスタマイズすると、今後のバージョンで Web コンソールのマークアップや動作が変更された場合に、アップグレード時に再作業が必要になる可能性があります。

### 39.2. 拡張スクリプトとスタイルシートの読み込み

拡張スクリプトとスタイルシートは、URL がブラウザからアクセス可能であれば、任意の **https://** URL でホストできます。ファイルは、パブリックにアクセスできるルートを使用してプラットフォーム上の Pod からや OpenShift Container Platform 外部の別のサーバー上の Pod からホストできます。

スクリプトとスタイルシートを追加するには、**openshift-web-console** namespace で **webconsole-config** ConfigMap を編集します。Web コンソール設定は ConfigMap の **webconsole-config.yaml** キーにあります。

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

スクリプトを追加するには、**extensions.scriptURLs** プロパティを更新します。値は URL の配列です。

スタイルシートを追加するには、**extensions.stylesheetURLs** プロパティを更新します。値は URL の配列です。

#### extensions.stylesheetURLs 設定の例

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    extensions:
      scriptURLs:
        - https://example.com/scripts/menu-customization.js
        - https://example.com/scripts/nav-customization.js
      stylesheetURLs:
        - https://example.com/styles/logo.css
        - https://example.com/styles/custom-styles.css
    [...]
```

ConfigMap を保存した後、Web コンソールコンテナは数分以内に新規の拡張ファイルについて自動的に更新されます。



### 注記

スクリプトとスタイルシートは正しいコンテンツタイプで提供する必要があります。そうしないと、それらはブラウザで実行されません。スクリプトは **Content-Type: application/javascript**、スタイルシートは **Content-Type: text/css** で提供する必要があります。

最良の実践例として、拡張スクリプトを Immediately Invoked Function Expression (IIFE) でラップすることができます。これにより、Web コンソールや他の拡張で使用される名前と競合するグローバル変数を作成することを防ぐことができます。以下に例を示します。

```
(function() {
  // Put your extension code here...
})();
```

以降のセクションの例では、Web コンソールをカスタマイズする一般的な方法を示します。



### 注記

拡張の他の例については、GitHub の [OpenShift Origin](#) リポジトリを参照してください。

## 39.2.1. 拡張プロパティの設定

特定の拡張について環境ごとに異なるテキストを使用する場合、Web コンソール設定で環境を定義し、同じ拡張スクリプトを環境全体で使用できます。

拡張プロパティを追加するには、**openshift-web-console** namespace で **webconsole-config** ConfigMap を編集します。Web コンソール設定は ConfigMap の **webconsole-config.yaml** キーにあります。

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

**extensions.properties** の値を更新します。これはキーと値のペアのマップです。

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    extensions:
      [...]
    properties:
      doc_url: https://docs.openshift.com
      key1: value1
      key2: value2
    [...]
```

これによって、以下のコードが実行された場合と同様に、拡張からアクセスできるグローバル変数が生成されます。

```

window.OPENSIFT_EXTENSION_PROPERTIES = {
  doc_url: "https://docs.openshift.com",
  key1: "value1",
  key2: "value2"
}

```

### 39.3. 外部ロギングソリューションの拡張オプション

OpenShift Container Platform の EFK ロギングスタックを使用しなくても外部ロギングソリューションにリンクする拡張オプションを使用することができます。

```

'use strict';
angular.module("mylinkextensions", ['openshiftConsole'])
  .run(function(extensionRegistry) {
    extensionRegistry.add('log-links', $.spread(function(resource, options) {
      return {
        type: 'dom',
        node: '<span><a href="https://extension-point.example.com">' + resource.metadata.name +
          '</a><span class="action-divider">|</span></span>'
      };
    }));
  });
hawtioPluginLoader.addModule("mylinkextensions");

```

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているようにスクリプトを追加します。

### 39.4. ガイド付きツアーのカスタマイズと無効化

ユーザーの特定ブラウザへの初回ログイン時に、ガイド付きツアーが表示されます。新規ユーザーに対して **auto\_launch** を有効にできます。

```

window.OPENSIFT_CONSTANTS.GUIDED_TOURS.landing_page_tour.auto_launch = true;

```

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているようにスクリプトを追加します。

### 39.5. ドキュメントリンクのカスタマイズ

ランディングページのドキュメントリンクはカスタマイズ可能です。 **window.OPENSIFT\_CONSTANTS.CATALOG\_HELP\_RESOURCES** は、タイトルと **href** を含むオブジェクトの配列です。これらはリンクに変換されます。この配列を完全に上書きしたり、追加のリンクをプッシュまたはポップしたり、既存のリンクの属性を変更したりすることができます。以下に例を示します。

```

window.OPENSIFT_CONSTANTS.CATALOG_HELP_RESOURCES.links.push({
  title: 'Blog',
  href: 'https://blog.openshift.com'
});

```

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているようにスクリプトを追加します。

### 39.6. ロゴのカスタマイズ

以下のスタイルでは、Web コンソールヘッダーのロゴを変更します。

```
#header-logo {  
  background-image: url("https://www.example.com/images/logo.png");  
  width: 190px;  
  height: 20px;  
}
```

`example.com` URL を実際のイメージへの URL に置換して、幅と高さを調整します。理想的な高さは `20px` です。

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているスタイルシートを追加します。

## 39.7. メンバーシップのホワイトリストのカスタマイズ

メンバーシップページのデフォルトのホワイトリストには、**admin**、**basic-user**、**edit** などのクラスターロールのサブセットだけでなく、プロジェクト内に定義されているカスタムロールが表示されます。

たとえば、ホワイトリストに、独自のカスタムのクラスターロールセットを追加します。

```
window.OPENSIFT_CONSTANTS.MEMBERSHIP_WHITELIST = [  
  "admin",  
  "basic-user",  
  "edit",  
  "system:deployer",  
  "system:image-builder",  
  "system:image-puller",  
  "system:image-pusher",  
  "view",  
  "custom-role-1",  
  "custom-role-2"  
];
```

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているようにスクリプトを追加します。

## 39.8. ドキュメントへのリンクの変更

Web コンソールのさまざまなセクションに、外部ドキュメントへのリンクが表示されます。以下の例では、ドキュメントへの指定された 2 つのリンクの URL を変更します。

```
window.OPENSIFT_CONSTANTS.HELP['get_started_cli'] = "https://example.com/doc1.html";  
window.OPENSIFT_CONSTANTS.HELP['basic_cli_operations'] = "https://example.com/doc2.html";
```

または、すべてのドキュメントリンクのベース URL を変更できます。

この例では、デフォルトのヘルプ URL `https://example.com/docs/welcome/index.html` が作成されます。

```
window.OPENSIFT_CONSTANTS.HELP_BASE_URL = "https://example.com/docs/"; ❶
```

❶ パスの末尾は `/` にする必要があります。

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているようにスクリプトを追加します。

## 39.9. CLI をダウンロードするリンクの追加または変更

Web コンソールの **About** ページには、[コマンドラインインターフェイス \(CLI\)](#) ツールのダウンロードリンクがあります。これらのリンクはリンクテキストと URL の両方を提供して設定でき、それらをファイルパッケージに直接ポイントさせるか、または実際のパッケージを指す外部ページにポイントさせることを選択できます。

ダウンロードできるパッケージを直接ポイントするには、以下を実行します。ここでリンクテキストはパッケージプラットフォームになります。

```

window.OPENSIFT_CONSTANTS.CLI = {
  "Linux (32 bits)": "https://<cdn>/openshift-client-tools-linux-32bit.tar.gz",
  "Linux (64 bits)": "https://<cdn>/openshift-client-tools-linux-64bit.tar.gz",
  "Windows":       "https://<cdn>/openshift-client-tools-windows.zip",
  "Mac OS X":      "https://<cdn>/openshift-client-tools-mac.zip"
};

```

または、実際のダウンロードパッケージにリンクするページをポイントするには、以下を実行します。ここでは、**Latest Release** をリンクテキストに指定しています。

```

window.OPENSIFT_CONSTANTS.CLI = {
  "Latest Release": "https://<cdn>/openshift-client-tools/latest.html"
};

```

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているようにスクリプトを追加します。

### 39.9.1. About ページのカスタマイズ

Web コンソールの **About** ページをカスタマイズするには、以下の手順に従います。

1. 以下のような拡張を作成します。

```

angular
  .module('aboutPageExtension', ['openshiftConsole'])
  .config(function($routeProvider) {
    $routeProvider
      .when('/about', {
        templateUrl: 'https://example.com/extensions/about/about.html',
        controller: 'AboutController'
      });
  })
);

hawtioPluginLoader.addModule('aboutPageExtension');

```

2. カスタムテンプレートを作成します。  
使用している OpenShift Container Platform [リリース](#) の [about.html](#) のバージョンから開始します。テンプレート内には 2 つの angular スコープ変数、**version.master.openshift** および **version.master.kubernetes** があります。
3. Web コンソールの適切な Cross-Origin Resource Sharing (CORS) 応答ヘッダーを持つ URL でテンプレートをホストします。

- a. Web コンソールドメインからの要求を許可するように **Access-Control-Allow-Origin** 応答を設定します。
- b. **GET** を含めるように **Access-Control-Allow-Methods** を設定します。
- c. **Content-Type** を含めるように **Access-Control-Allow-Headers** を設定します。

または、AngularJS `$templateCache` を使用すると、テンプレートを JavaScript に直接含めることができます。

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているようにスクリプトを追加します。

## 39.10. ナビゲーションメニューの設定

### 39.10.1. トップナビゲーションドロップダウンメニュー

Web コンソールのトップナビゲーションバーには、ヘルプアイコンとユーザードロップダウンメニューがあります。これらには `angular-extension-registry` を使用してメニュー項目を追加できます。

以下の拡張ポイントを利用できます。

- **nav-help-dropdown** - ヘルプアイコンのドロップダウンメニュー、デスクトップ画面の幅で表示される
- **nav-user-dropdown** - ユーザードロップダウンメニュー、デスクトップ画面の幅で表示される
- **nav-dropdown-mobile** - トップナビゲーション項目の単一メニュー、モバイル画面の幅で表示される

以下の例では、**nav-help-dropdown** メニューを `<myExtensionModule>` の名前で拡張しています。



#### 注記

`<myExtensionModule>` はプレースホルダー名です。各ドロップダウンメニュー拡張は、今後作成される angular モジュールと競合しないように一意にする必要があります。

```
angular
.module('<myExtensionModule>', ['openshiftConsole'])
.run([
  'extensionRegistry',
  function(extensionRegistry) {
    extensionRegistry
      .add('nav-help-dropdown', function() {
        return [
          {
            type: 'dom',
            node: '<li><a href="http://www.example.com/report" target="_blank">Report a Bug</a></li>'
          }, {
            type: 'dom',
            node: '<li class="divider"></li>' // If you want a horizontal divider to appear in the menu
          }, {
            type: 'dom',
            node: '<li><a href="http://www.example.com/status" target="_blank">System Status</a>'
          }
        ];
      })
  }
]);
```

```

</li>'
    }
  ];
});
}
]);

hawtioPluginLoader.addModule('<myExtensionModule>');

```

拡張スクリプトとスタイルシートの読み込みで説明されているようにスクリプトを追加します。

### 39.10.2. アプリケーションランチャー

トップナビゲーションバーには、他の Web アプリケーションにリンクするオプションのアプリケーションランチャーもあります。このドロップダウンメニューはデフォルトでは空ですが、リンクを追加すると、マストヘッドのヘルプメニューの左側に表示されます。

```

// Add items to the application launcher dropdown menu.
window.OPENS SHIFT_CONSTANTS.APP_LAUNCHER_NAVIGATION = [{
  title: "Dashboard",           // The text label
  iconClass: "fa fa-dashboard", // The icon you want to appear
  href: "http://example.com/dashboard", // Where to go when this item is clicked
  tooltip: 'View dashboard'    // Optional tooltip to display on hover
}, {
  title: "Manage Account",
  iconClass: "pficon pficon-user",
  href: "http://example.com/account",
  tooltip: "Update email address or password."
}
];

```

拡張スクリプトとスタイルシートの読み込みで説明されているようにスクリプトを追加します。

### 39.10.3. システムステータスバッジ

トップナビゲーションバーにはオプションのシステムステータスバッジも追加できます。これによって、メンテナランス時期など、システム全体のイベントをユーザーに通知できます。黄色の警告アイコンを使う既存のスタイルをバッジに使用するには、以下の例に従います。

```

'use strict';

angular
  .module('mysystemstatusbadgeextension', ['openshiftConsole'])
  .run([
    'extensionRegistry',
    function(extensionRegistry) {
      // Replace http://status.example.com/ with your domain
      var system_status_elem = $('<a href="http://status.example.com/" +
        "target="_blank" class="nav-item-iconic system-status"><span title="" +
        'System Status' class="fa status-icon pficon-warning-triangle-o">' +
        '</span></a>');

      // Add the extension point to the registry so the badge appears
      // To disable the badge, comment this block out
      extensionRegistry
        .add('nav-system-status', function() {

```

```

    return [{
      type: 'dom',
      node: system_status_elem
    }];
  });
}
]);

hawtioPluginLoader.addModule('mysystemstatusbadgeextension');

```

拡張スクリプトとスタイルシートの読み込みで説明されているようにスクリプトを追加します。

### 39.10.4. プロジェクトの左ナビゲーション

プロジェクト内でナビゲートするとき、プライマリーおよびセカンダリーナビゲーションのメニューが左側に表示されます。このメニューの構造は定数として定義され、上書きや修正が可能です。



#### 注記

プロジェクトナビゲーションに大幅なカスタマイズを実行すると、ユーザーエクスペリエンスに影響することがあるので、十分に注意して行ってください。既存のナビゲーション項目を変更した場合、今後のアップグレードでこのカスタマイズを更新する必要があります。

```

// Append a new primary nav item. This is a simple direct navigation item
// with no secondary menu.
window.OPENSIFT_CONSTANTS.PROJECT_NAVIGATION.push({
  label: "Dashboard",           // The text label
  iconClass: "fa fa-dashboard", // The icon you want to appear
  href: "/dashboard"           // Where to go when this nav item is clicked.
                                // Relative URLs are pre-pended with the path
                                // '/project/<project-name>'
});

// Splice a primary nav item to a specific spot in the list. This primary item has
// a secondary menu.
window.OPENSIFT_CONSTANTS.PROJECT_NAVIGATION.splice(2, 0, { // Insert at the third spot
  label: "Git",
  iconClass: "fa fa-code",
  secondaryNavSections: [ // Instead of an href, a sub-menu can be defined
    {
      items: [
        {
          label: "Branches",
          href: "/git/branches",
          prefixes: [
            "/git/branches/" // Defines prefix URL patterns that will cause
                              // this nav item to show the active state, so
                              // tertiary or lower pages show the right context
          ]
        }
      ]
    }
  ]
},
{
  header: "Collaboration", // Sections within a sub-menu can have an optional header

```

```

    items: [
      {
        label: "Pull Requests",
        href: "/git/pull-requests",
        prefixes: [
          "/git/pull-requests/"
        ]
      }
    ]
  }
}
});

// Add a primary item to the top of the list. This primary item is shown conditionally.
window.OPENSIFT_CONSTANTS.PROJECT_NAVIGATION.unshift({
  label: "Getting Started",
  iconClass: "pficon pficon-screen",
  href: "/getting-started",
  prefixes: [ // Primary nav items can also specify prefixes to trigger
    "/getting-started/" // active state
  ],
  isValid: function() { // Primary or secondary items can define an isValid
    return isNewUser; // function. If present it will be called to test whether
    // the item should be shown, it should return a boolean
  }
});

// Modify an existing menu item
var applicationsMenu = _.find(window.OPENSIFT_CONSTANTS.PROJECT_NAVIGATION, { label:
'Applications' });
applicationsMenu.secondaryNavSections.push({ // Add a new secondary nav section to the
Applications menu
  // my secondary nav section
});

```

拡張スクリプトとスタイルシートの読み込みで説明されているようにスクリプトを追加します。

## 39.11. 主要アプリケーションの設定

Web コンソールのランディングページカタログには、主要アプリケーションへのリンクの (オプションの) 一覧があります。これらはページ上部近くに表示され、アイコン、タイトル、簡単な説明およびリンクを指定できます。



```
// Add featured applications to the top of the catalog.
window.OPENSIFT_CONSTANTS.SAAS_OFFERINGS = [{
  title: "Dashboard",           // The text label
  icon: "fa fa-dashboard",     // The icon you want to appear
  url: "http://example.com/dashboard", // Where to go when this item is clicked
  description: "Open application dashboard." // Short description
}, {
  title: "System Status",
  icon: "fa fa-heartbeat",
  url: "http://example.com/status",
  description: "View system alerts and outages."
}, {
  title: "Manage Account",
  icon: "pficon pficon-user",
  url: "http://example.com/account",
  description: "Update email address or password."
}];
```

拡張スクリプトとスタイルシートの読み込みで説明されているようにスクリプトを追加します。

## 39.12. カタログカテゴリーの設定

カタログカテゴリーは、Web コンソールカタログのランディングページ内の項目の表示を整理します。各カテゴリーには1つ以上のサブカテゴリーがあります。一致するサブカテゴリータグに含まれるタグがある場合、ビルダーイメージ、テンプレート、またはサービスがサブカテゴリーにグループ化されます。また、1つの項目を複数のサブカテゴリーに表示することができます。カテゴリーとサブカテゴリーは、1つ以上の項目がある場合にのみ表示されます。



### 注記

カタログカテゴリーに大幅なカスタマイズを実行すると、ユーザーエクスペリエンスに影響することがあるので、十分に注意して行ってください。既存のカテゴリー項目を変更した場合、今後のアップグレードでこのカスタマイズを更新することが必要になる可能性があります。

```
// Find the Languages category.
var category = _.find(window.OPENSIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES,
  { id: 'languages' });
// Add Go as a new subcategory under Languages.
category.subCategories.splice(2,0,{ // Insert at the third spot.
  // Required. Must be unique.
  id: "go",
  // Required.
  label: "Go",
  // Optional. If specified, defines a unique icon for this item.
  icon: "icon-go-gopher",
  // Required. Items matching any tag will appear in this subcategory.
  tags: [
    "go",
    "golang"
  ]
});
// Add a Featured category as the first category tab.
```

```

window.OPENSIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES.unshift({
  // Required. Must be unique.
  id: "featured",
  // Required
  label: "Featured",
  subCategories: [
    {
      // Required. Must be unique.
      id: "go",
      // Required.
      label: "Go",
      // Optional. If specified, defines a unique icon for this item.
      icon: "icon-go-gopher",
      // Required. Items matching any tag will appear in this subcategory.
      tags: [
        "go",
        "golang"
      ]
    },
    {
      // Required. Must be unique.
      id: "jenkins",
      // Required.
      label: "Jenkins",
      // Optional. If specified, defines a unique icon for this item.
      icon: "icon-jenkins",
      // Required. Items matching any tag will appear in this subcategory.
      tags: [
        "jenkins"
      ]
    }
  ]
});

```

拡張スクリプトとスタイルシートの読み込みで説明されているようにスクリプトを追加します。

### 39.13. クォータ通知メッセージの設定

ユーザーがクォータに達すると、クォータ通知が通知ドロワーに追加されます。カスタムクォータ通知メッセージ ([クォータリソースタイプ](#) 別) を通知に追加できます。以下に例を示します。

```

Your project is over quota. It is using 200% of 2 cores CPU (Limit). Upgrade
to OpenShift Online Pro if you need
additional resources.

```

通知の Upgrade to...の部分はカスタムメッセージで、追加リソースへのリンクなどの HTML を指定できます。



#### 注記

クォータメッセージは HTML マークアップなので、特殊文字はすべて HTML 向けに正しくエスケープする必要があります。

**window.OPENSIFT\_CONSTANTS.QUOTA\_NOTIFICATION\_MESSAGE** プロパティを拡張スクリプトに設定して、各リソースについてメッセージをカスタマイズします。

```
// Set custom notification messages per quota type/key
window.OPENSIFT_CONSTANTS.QUOTA_NOTIFICATION_MESSAGE = {
  'pods': 'Upgrade to <a href="https://www.openshift.com">OpenShift Online Pro</a> if you need
additional resources.',
  'limits.memory': 'Upgrade to <a href="https://www.openshift.com">OpenShift Online Pro</a> if you
need additional resources.'
};
```

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているようにスクリプトを追加します。

## 39.14. CREATE FROM URL NAMESPACE ホワイトリストの設定

[Create from URL](#) は、明示的に **OPENSIFT\_CONSTANTS.CREATE\_FROM\_URL\_WHITELIST** に指定される namespace からのイメージストリームまたはテンプレートとのみ機能します。namespace をホワイトリストに追加するには、以下の手順に従います。



### 注記

**openshift** はデフォルトでホワイトリストに含まれています。これは削除しないでください。

```
// Add a namespace containing the image streams and/or templates
window.OPENSIFT_CONSTANTS.CREATE_FROM_URL_WHITELIST.push(
  'shared-stuff'
);
```

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているようにスクリプトを追加します。

## 39.15. COPY LOGIN コマンドの無効化

Web コンソールではユーザーは、現在のアクセストークンなどのログインコマンドをユーザーメニューおよび Command Line Tools ページからクリップボードにコピーできます。この機能は、ユーザーのアクセストークンがコピーされたコマンドに含まれないように変更することができます。

```
// Do not copy the user's access token in the copy login command.
window.OPENSIFT_CONSTANTS.DISABLE_COPY_LOGIN_COMMAND = true;
```

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているようにスクリプトを追加します。

### 39.15.1. ワイルドカードルートの有効化

ワイルドカードルートをルーターで有効にした場合、Web コンソールでもワイルドカードルートを有効にできます。これによりユーザーはルートを作成するときに、**\*.example.com** などのアスタリスクで始まるホスト名を入力できます。ワイルドカードルートを有効にするには、以下を実行します。

```
window.OPENSIFT_CONSTANTS.DISABLE_WILDCARD_ROUTES = false;
```

[拡張スクリプトとスタイルシートの読み込み](#) で説明されているようにスクリプトを追加します。

ワイルドカードルートを許可するように HAProxy ルートを設定する方法については [こちら](#) を参照してください。

## 39.16. ログインページのカスタマイズ

Web コンソールのログインページおよびログインプロバイダー選択ページも変更できます。以下のコマンドを実行して、変更可能なテンプレートを作成します。

```
$ oc adm create-login-template > login-template.html
$ oc adm create-provider-selection-template > provider-selection-template.html
```

ファイルを編集して、スタイルを変更するか、または内容を追加します。ただし、波括弧内の必須パラメーターを削除しないよう注意してください。

カスタムログインページまたはプロバイダー選択ページを使用するには、以下のオプションをマスター設定ファイルに設定します。

```
oauthConfig:
  ...
  templates:
    login: /path/to/login-template.html
    providerSelection: /path/to/provider-selection-template.html
```

相対パスは、マスター設定ファイルを基準にして解決されます。この設定を変更したらサーバーを再起動する必要があります。

複数のログインプロバイダーが設定されている場合、または [master-config.yaml](#) ファイルの `alwaysShowProviderSelection` オプションが `true` に設定されている場合、OpenShift Container Platform へのユーザーのトークンが期限切れになるたびに、このカスタムページが他のタスクに進む前にユーザーに表示されます。

### 39.16.1. 使用例

サービス利用規約情報はカスタムログインページを使用して作成できます。このようなページは、GitHub や Google などのサードパーティーログインプロバイダーを使用している場合にも、ユーザーが信頼し、予想できるブランドのページを提示して、その後にユーザーを認証プロバイダーにリダイレクトする際に役立ちます。

## 39.17. OAUTH エラーページのカスタマイズ

認証中にエラーが発生した場合に表示されるページを変更できます。

1. 以下のコマンドを実行して、変更可能なテンプレートを作成します。

```
$ oc adm create-error-template > error-template.html
```

2. ファイルを編集して、スタイルを変更するか、または内容を追加します。テンプレートで **Error** 変数および **ErrorCode** 変数を使用できます。カスタムエラーページを使用するには、以下のオプションをマスター設定ファイルに設定します。

```
oauthConfig:
  ...
  templates:
```

```
error: /path/to/error-template.html
```

相対パスは、マスター設定ファイルを基準にして解決されます。

3. この設定を変更したらサーバーを再起動する必要があります。

## 39.18. ログアウト URL の変更

**webconsole-config** ConfigMap の **clusterInfo.logoutPublicURL** パラメーターを変更すると、コンソールをログアウトしたときにコンソールユーザーに表示される場所を変更できます。

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

以下の例では、ログアウト URL を <https://www.example.com/logout> に変更しています。

```
apiVersion: v1
kind: ConfigMap
data:
  webconsole-config.yaml: |
    apiVersion: webconsole.config.openshift.io/v1
    clusterInfo:
      [...]
      logoutPublicURL: "https://www.example.com/logout"
      [...]
```

これは、[要求ヘッダー](#) および OAuth または [OpenID](#) アイデンティティプロバイダーで認証する場合に便利です。この場合、外部 URL にアクセスしてシングルサインオンセッションを破棄する必要があります。

## 39.19. ANSIBLE を使用した WEB コンソールカスタマイズの設定

クラスターのインストール時に、以下のパラメーターを使用して、Web コンソールへの変更を多数設定できます。これらのパラメーターは、[インベントリーファイル](#) で設定可能です。

- [openshift\\_master\\_logout\\_url](#)
- [openshift\\_web\\_console\\_extension\\_script\\_urls](#)
- [openshift\\_web\\_console\\_extension\\_stylesheet\\_urls](#)
- [openshift\\_master\\_oauth\\_templates](#)
- [openshift\\_master\\_metrics\\_public\\_url](#)
- [openshift\\_master\\_logging\\_public\\_url](#)

### Ansible による Web コンソールカスタマイズの例

```
# Configure `clusterInfo.logoutPublicURL` in the web console configuration
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#changing-the-logout-url
#openshift_master_logout_url=https://example.com/logout
```

```

# Configure extension scripts for web console customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#loading-
custom-scripts-and-stylesheets
#openshift_web_console_extension_script_urls=['https://example.com/scripts/menu-
customization.js','https://example.com/scripts/nav-customization.js']

# Configure extension stylesheets for web console customization
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#loading-
custom-scripts-and-stylesheets
#openshift_web_console_extension_stylesheet_urls=
['https://example.com/styles/logo.css','https://example.com/styles/custom-styles.css']

# Configure a custom login template in the master config
# See:
https://docs.openshift.com/enterprise/latest/install_config/web_console_customization.html#customizing
the-login-page
#openshift_master_oauth_templates={'login': '/path/to/login-template.html'}

# Configure `clusterInfo.metricsPublicURL` in the web console configuration for cluster metrics.
Ansible is also able to configure metrics for you.
# See: https://docs.openshift.com/enterprise/latest/install_config/cluster_metrics.html
#openshift_master_metrics_public_url=https://hawkular-metrics.example.com/hawkular/metrics

# Configure `clusterInfo.loggingPublicURL` in the web console configuration for aggregate logging.
Ansible is also able to install logging for you.
# See: https://docs.openshift.com/enterprise/latest/install_config/aggregate_logging.html
#openshift_master_logging_public_url=https://kibana.example.com

```

## 39.20. WEB コンソール URL ポートおよび証明書の変更

ユーザーが Web コンソール URL にアクセスする際にカスタム証明書が提供されるようにするには、証明書および URL を `master-config.yaml` ファイルの `namedCertificates` セクションに追加します。詳細は、[Web コンソールまたは CLI のカスタム証明書の設定](#) を参照してください。

Web コンソールのリダイレクト URL を設定または変更するには、`openshift-web-console oauthclient` を変更します。

```
$ oc edit oauthclient openshift-web-console
```

ユーザーが適切にリダイレクトされるようにするには、`openshift-web-console configmap` の `PublicUrls` を更新します。

```
$ oc edit configmap/webconsole-config -n openshift-web-console
```

次に、`consolePublicURL` の値を更新します。

## 第40章 外部永続ボリュームプロビジョナーのデプロイ

### 40.1. 概要



#### 重要

OpenShift Container Platform の AWS EFS の外部プロビジョナーはテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) ではサポートされていません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

外部プロビジョナーは、特定のストレージプロバイダーの動的プロビジョニングを可能にするアプリケーションです。外部プロビジョナーは、OpenShift Container Platform が提供するプロビジョナープラグインと共に実行でき、**StorageClass** オブジェクトの設定と同じ方法で設定されます。この方法は、[動的プロビジョニングとストレージクラスの作成](#) のセクションで説明しています。これらのプロビジョナーは外部プロビジョナーであるため、OpenShift Container Platform とは独立してデプロイし、更新できます。

### 40.2. 操作を始める前に

外部プロビジョナーのデプロイとアップグレードには、[Ansible Playbook](#) も使用できます。



#### 注記

[クラスターメトリクス](#)の設定と [クラスターロギング](#)の設定 のセクションに十分に理解してから、先に進むようにしてください。

#### 40.2.1. 外部プロビジョナーの Ansible ロール

OpenShift Ansible **openshift\_provisioners** ロールは [Ansible](#) インベントリーファイルからの変数を使用して、外部プロビジョナーを設定し、デプロイします。それぞれの **install** 変数を **true** に書きし、インストールするプロビジョナーを指定する必要があります。

#### 40.2.2. 外部プロビジョナーの Ansible 変数

以下は、**install** 変数が **true** で、すべてのプロビジョナーに適用されるロール変数の一覧です。

表40.1 Ansible 変数

変数	説明
<b>openshift_provisioners_install_provisioners</b>	<b>true</b> の場合、それぞれの <b>install</b> 変数が <b>true</b> に設定されているすべてのプロビジョナーをデプロイします。それ以外の場合は削除します。
<b>openshift_provisioners_image_prefix</b>	コンポーネントイメージの接頭辞。たとえば、デフォルトで <b>registry.redhat.io/openshift3/</b> に設定されている場合、別のレジストリーを使用する場合はこれを異なる値に設定します。

変数	説明
<code>openshift_provisioners_image_version</code>	コンポーネントイメージのバージョン。たとえば <code>openshift3/ose-efs-provisioner:v3.11</code> の場合、バージョンを <code>v3.11</code> に設定します。
<code>openshift_provisioners_project</code>	プロビジョナーのデプロイ先のプロジェクト。デフォルトは <code>openshift-infra</code> です。

### 40.2.3. AWS EFS プロビジョナーの Ansible 変数

AWS EFS プロビジョナーは、所定の EFS ファイルシステムのディレクトリーに動的に作成されたディレクトリーを基盤とする `NFS PV` を動的にプロビジョニングします。AWS EFS プロビジョナー Ansible 変数を設定するには、以下の要件を満たす必要があります。

- `AmazonElasticFileSystemReadOnlyAccess` ポリシー (またはこれ以上) を割り当てられた IAM ユーザー。
- クラスターリージョン内の EFS ファイルシステム。
- 任意のノード (クラスターのリージョン内の任意のゾーン) が `ファイルシステムの DNS 名` で EFS ファイルシステムをマウントできる、`マウントターゲット` と `セキュリティグループ`。

表40.2 必須の EFS Ansible 変数

変数	説明
<code>openshift_provisioners_efs_fs_id</code>	EFS ファイルシステムの <code>ファイルシステム ID</code> 。例: <code>fs-47a2c22e</code>
<code>openshift_provisioners_efs_region</code>	EFS ファイルシステムの Amazon EC2 リージョン
<code>openshift_provisioners_efs_aws_access_key_id</code>	IAM ユーザーの AWS アクセスキー (指定された EFS ファイルシステムが存在するかチェックする)
<code>openshift_provisioners_efs_aws_secret_access_key</code>	IAM ユーザーの AWS シークレットアクセスキー (指定された EFS ファイルシステムが存在するかチェックする)

表40.3 オプションの EFS Ansible 変数

変数	説明
<code>openshift_provisioners_efs</code>	<code>true</code> の場合、AWS EFS プロビジョナーは <code>openshift_provisioners_install_provisioners</code> が <code>true</code> か <code>false</code> によって、インストールまたはアンインストールされます。デフォルトは <code>false</code> です。

変数	説明
<code>openshift_provisioners_efs_path</code>	EFS ファイルシステム内のディレクトリーのパスで、ここで EFS プロビジョナーがディレクトリーを作成して、作成する各 PV をサポートします。これは存在している必要があり、EFS プロビジョナーでマウントできる必要があります。デフォルトは <code>/persistentvolumes</code> です。
<code>openshift_provisioners_efs_name</code>	<code>StorageClasses</code> で指定する <b>provisioner</b> 名。デフォルトは <code>openshift.org/aws-efs</code> です。
<code>openshift_provisioners_efs_nodeselector</code>	Pod が配置されるノードを選択するラベルのマッピング。例: <code>{"node":"infra","region":"west"}</code>
<code>openshift_provisioners_efs_supplementalgroup</code>	EFS ファイルシステムへの書き込みパーミッションのために必要な場合の、Pod に指定する補助グループ。デフォルトは <b>65534</b> です。

### 40.3. プロビジョナーのデプロイ

OpenShift Ansible 変数で指定される設定に従って、すべてのプロビジョナーを同時にデプロイすることも、プロビジョナーを1つずつデプロイすることもできます。以下の例では、指定されたプロビジョナーをデプロイして、対応する **StorageClass** を作成し、設定する方法を示します。

#### 40.3.1. AWS EFS プロビジョナーのデプロイ

以下のコマンドは、EFS ボリューム内のディレクトリーを `/data/persistentvolumes` に設定します。このディレクトリーはファイルシステム内に存在している必要があり、プロビジョナー Pod によってマウントや書き込みができる必要があります。Playbook ディレクトリーに切り替え、以下の Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v -i <inventory_file> \
  playbooks/openshift-provisioners/config.yml \
  -e openshift_provisioners_install_provisioners=True \
  -e openshift_provisioners_efs=True \
  -e openshift_provisioners_efs_fsid=fs-47a2c22e \
  -e openshift_provisioners_efs_region=us-west-2 \
  -e openshift_provisioners_efs_aws_access_key_id=AKIAIOSFODNN7EXAMPLE \
  -e
  openshift_provisioners_efs_aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEK
  EY \
  -e openshift_provisioners_efs_path=/data/persistentvolumes
```

##### 40.3.1.1. AWS EFS オブジェクト定義

`aws-efs-storageclass.yaml`

■

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: slow
provisioner: openshift.org/aws-efs ❶
parameters:
  gidMin: "40000" ❷
  gidMax: "50000" ❸
```

- ❶ この値を `openshift_provisioners_efs_name` 変数と同じ値に設定します。デフォルトは `openshift.org/aws-efs` です。
- ❷ StorageClass の GID 範囲の最小値です。(オプション)
- ❸ StorageClass の GID 範囲の最大値です。(オプション)

動的にプロビジョニングされた各ボリュームに対応する NFS ディレクトリーには、`gidMin` から `gidMax` の範囲に一意の GID 所有者が割り当てられます。指定されない場合、デフォルトで `gidMin` は **2000**、`gidMax` は **2147483647** になります。要求によってプロビジョニングされるボリュームを使用するすべての Pod は、必要な GID を補助グループとして自動的に実行され、ボリュームの読み取り/書き込みができます。補助グループを持たない(かつルートとして実行されていない)他のマウンターは、ボリュームの読み取り/書き込みはできません。補助グループを使用して NFS アクセスを管理する方法については、[グループ ID](#) セクションの NFS ボリュームセキュリティのトピックを参照してください。

## 40.4. CLEANUP

以下のコマンドを実行すると、OpenShift Ansible `openshift_provisioners` ロールによってデプロイされたものをすべて削除できます。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -v -i <inventory_file> \
  playbooks/openshift-provisioners/config.yml \
  -e openshift_provisioners_install_provisioners=False
```

## 第41章 OPERATOR FRAMEWORK (テクノロジープレビュー) のインストール

Red Hat が発表したオープンソースツールキットの [Operator Framework](#) は、**Operator** と呼ばれる Kubernetes のネイティブアプリケーションをより効率的で自動化されたスケーラブルな方法で管理できるように設計されています。

以下のセクションでは、クラスター管理者が OpenShift Container Platform 3.11 のテクノロジープレビューである Operator Framework を試用する手順を説明しています。



### 重要

Operator Framework はテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、<https://access.redhat.com/support/offerings/techpreview/> を参照してください。

### 41.1. このテクノロジープレビューについて

テクノロジープレビューの Operator Framework は、クラスター管理者の OpenShift Container Platform クラスターでの Operator のインストール、アップグレードおよび Operator に対するアクセスの付与を支援する [Operator Lifecycle Manager \(OLM\)](#) をインストールします。

また、OpenShift Container Platform Web コンソールは、クラスター管理者が Operator をインストールし、クラスターで利用可能な Operator のカタログを使用するための特定のプロジェクトアクセスを付与する際に使用する新規の管理画面と共に更新されています。

開発者に対しては、セルフサービスを使用することで、専門的な知識がなくてもデータベースのプロビジョニングや設定、またモニターリング、ビッグデータサービスなどを実行できます。Operator にこれらのノウハウが織り込まれているためです。

## 図41.1 Operator カタログソース

## Operator Catalog Sources

Catalogs are groups of Operators you can make available on the cluster. Subscribe and grant a namespace access to use the installed Operators.

## Certified Operators

Packaged by Red Hat

[View catalog details](#)

NAME	LATEST VERSION	SUBSCRIPTIONS
 Couchbase Operator provided by Couchbase	1.0.0 (preview)	Not subscribed <a href="#">Create Subscription</a>
 Dynatrace OneAgent provided by Dynatrace, Inc	0.2.0 (preview)	Not subscribed <a href="#">Create Subscription</a>
 MongoDB provided by MongoDB, Inc	0.3.2 (preview)	Not subscribed <a href="#">Create Subscription</a>

## Red Hat Operators

Packaged by Red Hat

[View catalog details](#)

NAME	LATEST VERSION	SUBSCRIPTIONS
 AMQ Streams provided by Red Hat, Inc.	1.0.0-Beta (preview)	Not subscribed <a href="#">Create Subscription</a>
 etcd provided by CoreOS, Inc	0.9.2 (alpha)	Not subscribed <a href="#">Create Subscription</a>
 Prometheus Operator provided by Red Hat	0.22.2 (preview)	Not subscribed <a href="#">Create Subscription</a>

以下のスクリーンショットには、主要ソフトウェアベンダーのパートナー Operator の事前に読み込まれたカタログソースが表示されています。

## Couchbase Operator

Couchbase は、リレーショナルデータベースで使用される表形式のリレーション以外の方法でモデル化されたデータの保管および取得メカニズムを提供します。開発者プレビューとして OpenShift Container Platform 3.11 で利用可能な、Couchbase でサポートされている Operator は、Couchbase デプロイメントを OpenShift Container Platform でネイティブに実行することを可能にします。これは NoSQL クラスターをより効果的にインストールし、このフェイルオーバーを実行します。

## Dynatrace Operator

Dynatrace アプリケーションモニタリングはリアルタイムでパフォーマンスメトリクスを提供し、問題を自動的に検出し、診断するのに役立ちます。この Operator により、コンテナフォークスのモニタリングスタックのインストールやこの Dynatrace モニタリングクラウドへの接続、カスタムリソースの監視、および必要な状態の継続的なモニターなどをより簡単に実行することができます。

## MongoDB Operator

MongoDB は、データを柔軟な JSON のようなドキュメントに保存する分散型のトランザクショナルデータベースです。Operator は実稼働対応のレプリカセットおよび共有クラスター、およびスタンドアロンの開発/テストインスタンスのどちらのデプロイもサポートします。これは MongoDB Ops Manager と連携し、すべてのクラスターが運用上のベストプラクティスに基づいてデプロイされるようにします。

以下の Red Hat 提供の Operator も含まれます。

### Red Hat AMQ Streams Operator

Red Hat AMQ Streams は、Apache Kafka プロジェクトに基づく拡張性の高い、分散型の高パフォーマンスのデータストリーミングプラットフォームです。これは、マイクロサービスおよび他のアプリケーションが高スループットの、非常に低い待機時間でデータを共有できるようにする分散型バックボーンを提供します。

### etcd Operator

etcd は、マシンのクラスター全体でデータを保存するための信頼できる方法を提供する分散型のキーと値のストアです。この Operator により、ユーザーは、etcd クラスターを作成し、設定し、管理する単純な宣言型の設定を使用して複雑な etcd を設定し、管理できます。

### Prometheus Operator

Prometheus は CNCF 内で Kubernetes と共にホストされるクラウドネイティブのモニターリングシステムです。この Operator には、作成/破棄などの共通タスク、単純な設定、ラベルを使用したモニターリングターゲット設定の自動生成などを行うためのアプリケーションのドメイン知識が組み込まれています。

## 41.2. ANSIBLE を使用した OPERATOR LIFECYCLE MANAGER のインストール

テクノロジープレビューの Operator Framework をインストールするには、クラスターのインストール後に OpenShift Container Platform **openshift-ansible** インストーラーに組み込まれている Playbook を使用できます。



### 注記

または、テクノロジープレビューの Operator Framework をクラスターの初回インストール時にインストールできます。それぞれの説明については、[Configuring Your Inventory File](#) を参照してください。

### 前提条件

- 既存の OpenShift Container Platform 3.11 クラスター
- **cluster-admin** パーミッションのあるアカウントを使用したクラスターへのアクセス
- 最新の **openshift-ansible** インストーラーで提供される Ansible Playbook

### 手順

1. OpenShift Container Platform クラスターのインストールおよび管理に使用されるインベントリーファイルで、**openshift\_additional\_registry\_credentials** 変数を **[OSEv3:vars]** セクションに追加し、Operator コンテナのプルに必要な認証情報を設定します。

```
openshift_additional_registry_credentials=
[{'host':'registry.connect.redhat.com','user':<your_user_name>,'password':<your_password>,'test_image':'mongodb/enterprise-operator:0.3.2'}]
```

-

**user** および **password** を、Red Hat カスタマーポータル (<https://access.redhat.com>) へのログインに使用する認証情報に追加します。

**test\_image** は、指定した認証情報をテストするために使用されるイメージを表します。

2. Playbook ディレクトリーに切り替え、ご自身のイベントリーファイルを使用してレジストリー認証 Playbook を実行し、直前の手順の認証情報を使用してノードを認証します。

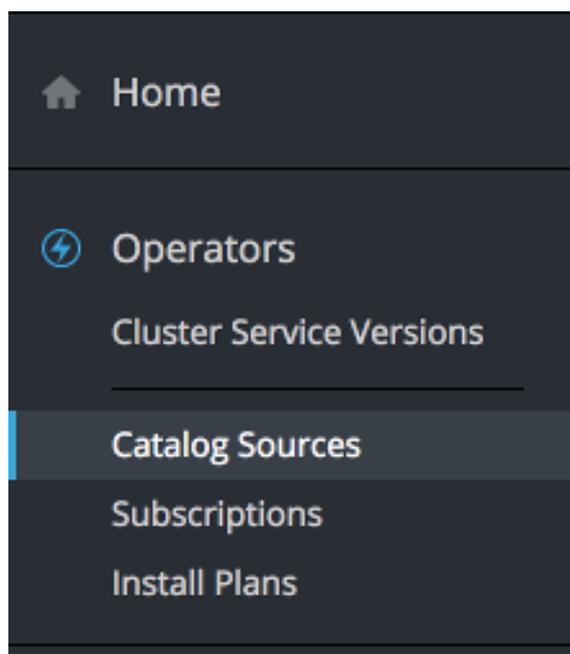
```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/updates/registry_auth.yml
```

3. Playbook ディレクトリーに切り替え、ご自身のイベントリーファイルを使用して OLM インストール Playbook を実行します。

```
$ cd /usr/share/ansible/openshift-ansible
$ ansible-playbook -i <inventory_file> \
  playbooks/olm/config.yml
```

4. ブラウザーを使用して、クラスターの Web コンソールに移動します。ページの左端にあるナビゲーションに新規のセクションが表示されます。

図41.2 新規 Operator ナビゲーションセクション



これは、Operator をインストールし、それらにプロジェクトアクセスを付与し、すべての環境用のインスタンスを起動する場所です。

### 41.3. 最初の OPERATOR の起動

このセクションでは、Couchbase Operator を使用して新規の Couchbase クラスターを作成する方法を説明します。

#### 前提条件

- テクノロジープレビューの OLM が有効にされた OpenShift Container Platform 3.11

- **cluster-admin** パーミッションのあるアカウントを使用したクラスターへのアクセス
- Operator カタログに読み込まれる Couchbase Operator (テクノロジープレビュー OLM と共にデフォルトで読み込まれる)

## 手順

1. クラスター管理者 (**cluster-admin** ロールを持つユーザー) として、この手順で OpenShift Container Platform Web コンソールを使用して新規プロジェクトを作成します。この例では、**couchbase-test** というプロジェクトを使用します。
2. プロジェクト内の Operator のインストールは、クラスター管理者がクラスター全体で作成し、管理できるサブスクリプションオブジェクトを使用して実行されます。利用可能なサブスクリプションを表示するには、ドロップダウンメニューから **Cluster Console** に移動し、左側のナビゲーションにある **Operators** → **Catalog Sources** 画面に移動します。



### 注記

プロジェクトでサブスクリプションを閲覧し、作成し、管理する追加のユーザーを有効にする必要がある場合、それらには、プロジェクトの **admin** および **view** ロールを持たせると共に、**operator-lifecycle-manager** プロジェクトの **view** ロールを持たせる必要があります。クラスター管理者は以下のコマンドを使用してそれらのロールを追加できます。

```
$ oc policy add-role-to-user admin <user> -n <target_project>
$ oc policy add-role-to-user view <user> -n <target_project>
$ oc policy add-role-to-user view <user> -n operator-lifecycle-manager
```

これについては、OLM の今後のリリースで単純化されます。

3. Web コンソールまたは CLI のいずれかより、必要なプロジェクトを Couchbase カタログソースにサブスクライブします。  
以下の方法のいずれかを選択します。
  - Web コンソールを使用した場合、必要なプロジェクトが表示されていることを確認したら、この画面の Operator で **Create Subscription** をクリックし、これをプロジェクトにインストールします。
  - CLI を使用した場合、以下の定義を使用して YAML ファイルを作成します。

### couchbase-subscription.yaml ファイル

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  generateName: couchbase-enterprise-
  namespace: couchbase-test ❶
spec:
  source: certified-operators
  name: couchbase-enterprise
  startingCSV: couchbase-operator.v1.0.0
  channel: preview
```

- 1 **metadata** セクションの **namespace** フィールドが必要なプロジェクトに設定されていることを確認します。

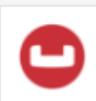
次に、CLI を使用してサブスクリプションを作成します。

```
$ oc create -f couchbase-subscription.yaml
```

4. サブスクリプションが作成されると、Operator は、カタログユーザーが Operator によって提供されるソフトウェアを起動するために使用できる **Cluster Service Versions** 画面に表示されません。Couchbase Operator をクリックすると、Operator の機能についての詳細を表示できます。

#### 図41.3 Couchbase Operator の概要

Project: couchbase-test ▾

 Couchbase Operator  
1.0.0 provided by Couchbase Actions ▾

[Overview](#) [YAML](#) [Instances](#)

[Create Couchbase Operator](#)

**PROVIDER**  
Couchbase

**CREATED AT**  
🕒 5 minutes ago

**LINKS**  
Couchbase  
<https://www.couchbase.com>

Documentation  
<https://docs.couchbase.com/ope>

Downloads  
<https://www.couchbase.com/dow>

**MAINTAINERS**  
Couchbase  
[support@couchbase.com](mailto:support@couchbase.com)

## Description

The Couchbase Autonomous Operator allows users to easily deploy, manage, and maintain Couchbase deployments on OpenShift. By installing this integration you will be able to deploy Couchbase Server clusters with a single command.

## Supported Features

- **Automated cluster provisioning** - Deploying a Couchbase Cluster has never been easier. Fill out a Couchbase specific configuration and let the Couchbase Operator take care of provisioning nodes and setting up cluster to your exact specification.
- **On-demand scalability** - Automatically scale your cluster up or down by changing a simple configuration parameter and let the Couchbase Operator handle provisioning of new nodes and joining them into the cluster.

5. Couchbase クラスターを作成する前に、Web コンソールまたは CLI を使用して以下の定義でスーパーユーザーのアカウントの認証情報を保持するシークレットを作成します。Operator は起動時にこれを読み取り、これらの詳細情報でデータベースを設定します。

#### Couchbase シークレット

```
apiVersion: v1
kind: Secret
```

```

metadata:
  name: couchbase-admin-creds
  namespace: couchbase-test ❶
type: Opaque
stringData:
  username: admin
  password: password

```

- ❶ **metadata** セクションの **namespace** フィールドが必要なプロジェクトに設定されていることを確認します。

以下の方法のいずれかを選択します。

- Web コンソールを使用した場合、左側のナビゲーションから **Workloads** → **Secrets** をクリックし、**Create** をクリックし、**Secret from YAML** を選択してシークレット定義を入力します。
- CLI を使用した場合、シークレット定義を YAML ファイル (**couchbase-secret.yaml** など) に保存し、CLI を使用してこれを必要なプロジェクトに作成します。

```
$ oc create -f couchbase-secret.yaml
```

## 6. 新規 Couchbase クラスターを作成します。



### 注記

指定プロジェクトで **edit** ロールを持つすべてのユーザーは、クラウドサービスのようにセルフサービスでプロジェクトにすでにインストールされている Operator が管理するアプリケーションインスタンス (この場合は Couchbase クラスター) を作成し、管理し、これを削除できます。この機能を持つ追加のユーザーを有効にする必要がある場合、クラスター管理者は以下のコマンドを使用してロールを追加できます。

```
$ oc policy add-role-to-user edit <user> -n <target_project>
```

- Web コンソールの **Cluster Service Versions** セクションから、Operator の **Overview** 画面の **Create Couchbase Operator** をクリックして新規 **CouchbaseCluster** オブジェクトの作成を開始できます。このオブジェクトは Operator がクラスターで利用可能にする新規のタイプです。オブジェクトはビルトインの **Deployment** または **ReplicaSet** オブジェクトのように機能しますが、これには Couchbase の管理に固有のロジックが含まれます。

### ヒント

**Create Couchbase Operator** ボタンを最初にクリックする際に、404 エラーが出される可能性があります。これは既知の問題であり、回避策としてこのページを更新してから続行するようにしてください。 ([BZ#1609731](#))

Web コンソールには最小限の開始用のテンプレートが含まれますが、Operator がサポートするすべての機能については、[Couchbase ドキュメント](#) を参照してください。

図41.4 Couchbase クラスターの作成

Project: couchbase-test ▾

### Create Couchbase Cluster

```

1  apiVersion: couchbase.com/v1
2  kind: CouchbaseCluster
3  metadata:
4    name: cb-example
5    namespace: couchbase-test
6  spec:
7    authSecret: couchbase-admin-creds
8    baseImage: registry.connect.redhat.com/couchbase/server
9    buckets:
10   - conflictResolution: seqno
11     enableFlush: true
12     evictionPolicy: fullEviction
13     ioPriority: high
14     memoryQuota: 128
15     name: default
16     replicas: 1
17     type: couchbase
18   cluster:
19     analyticsServiceMemoryQuota: 1024
20     autoFailoverMaxCount: 3
21     autoFailoverOnDataDiskIssues: true
22     autoFailoverOnDataDiskIssuesTimePeriod: 120
23     autoFailoverServerGroup: false
24     autoFailoverTimeout: 120
25     clusterName: cb-example
26     dataServiceMemoryQuota: 256
27     eventingServiceMemoryQuota: 256
28     indexServiceMemoryQuota: 256
29     indexStorageSetting: memory_optimized
30     searchServiceMemoryQuota: 256
31   servers:
32   - name: all_services
33     services:
34     - data
35     - index
36     - query
37     - search
38     - eventing
39     - analytics
40     size: 3
41   version: 5.5.1-1
42

```

Create Cancel

- b. **admin** 認証情報が含まれるシークレットの名前を設定していることを確認します。

```

apiVersion: couchbase.com/v1
kind: CouchbaseCluster
metadata:
  name: cb-example
  namespace: couchbase-test
spec:
  authSecret: couchbase-admin-creds
  baseImage: registry.connect.redhat.com/couchbase/server
  [...]

```

- c. オブジェクト定義が完了したら、Web コンソールの **Create** をクリック (または CLI を使用) してオブジェクトを作成します。これにより、Operator による Couchbase クラスターの Pod、サービスその他のコンポーネントの起動がトリガーされます。
7. プロジェクトには、Operator によって自動的に作成され、設定された多数のリソースが含まれています。

#### 図41.5 Couchbase クラスターの詳細

The screenshot shows the OpenShift console interface for a CouchbaseCluster resource named 'cb-example' in the 'couchbase-test' project. The 'Resources' tab is selected, showing a table of resources. The table has columns for NAME, TYPE, STATUS, and CREATED. There are 2 Services and 3 Pods listed.

NAME ↑	TYPE	STATUS	CREATED
cb-example	Service	Created	Sep 27, 3:12 pm
cb-example-0000	Pod	Running	Sep 27, 3:12 pm
cb-example-0001	Pod	Running	Sep 27, 3:13 pm
cb-example-0002	Pod	Running	Sep 27, 3:13 pm
cb-example-srv	Service	Created	Sep 27, 3:12 pm

**Resources** タブをクリックして、プロジェクトの他の Pod からのデータベースへのアクセスを可能にする Kubernetes サービスが作成されていることを確認します。

**cb-example** サービスを使用すると、シークレットに保存された認証情報を使用してデータベースに接続できます。他のアプリケーション Pod はこのシークレットをマウントし、使用でき、サービスと通信できます。

これで、Pod が正常でなくなる場合やクラスター内のノード間で移行される場合の障害に対応し、データをリバランスするフォールトトレラントな Couchbase のインストールが完了します。この際に、Couchbase クラスターリングまたはフェイルオーバーについての詳細な知識は不要です。

Couchbase Autonomous Operator の機能についての詳細は、[公式の Couchbase ドキュメント](#) を参照してください

## 41.4. ご協力をお願い

Operator Framework のユーザーエクスペリエンス、および Operator として提供されるサービスについてのご要望について、OpenShift チームにお聞かせください。

[openshift-operators@redhat.com](mailto:openshift-operators@redhat.com) 宛てにメールを送信していただき、OpenShift チームにお客様のご意見をお聞かせください。

## 第42章 OPERATOR LIFECYCLE MANAGER のアンインストール

クラスターをインストールした後、OpenShift Container Platform **openshift-ansible** インストーラーを使用して Operator Lifecycle Manager をアンインストールできます。

### 42.1. ANSIBLE を使用した OPERATOR LIFECYCLE MANAGER のアンインストール

クラスターをインストールした後、OpenShift Container Platform **openshift-ansible** インストーラーでこの手順を使用して、Technology Preview Operator Framework をアンインストールできます。

Technology Preview Operator Framework をアンインストールする前に、次の前提条件を確認する必要があります。

- 既存の OpenShift Container Platform 3.11 クラスター
- **cluster-admin** パーミッションのあるアカウントを使用したクラスターへのアクセス
- 最新の **openshift-ansible** インストーラーで提供される Ansible Playbook
  1. 次の変数を **config.yml** Playbook に追加します。

```
operator_lifecycle_manager_install=false
operator_lifecycle_manager_remove=true
```

2. Playbook ディレクトリーに移動します。

```
$ cd /usr/share/ansible/openshift-ansible
```

3. OLM インストール Playbook を実行して、インベントリーファイルを使用して OLM をアンインストールします。

```
$ ansible-playbook -i <inventory_file> playbooks/olm/config.yml
```