



OpenShift Container Platform 3.9

アーキテクチャー

OpenShift Container Platform 3.9 Architecture Information

OpenShift Container Platform 3.9 アーキテクチャー

OpenShift Container Platform 3.9 Architecture Information

法律上の通知

Copyright © 2021 Red Hat, Inc.

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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] 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[®] 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

Learn the architecture of OpenShift Container Platform 3.9 including the infrastructure and core components. These topics also cover authentication, networking and source code management.

目次

第1章 概要	7
1.1. レイヤーについて	7
1.2. OPENSIFT CONTAINER PLATFORM アーキテクチャーについて	8
1.3. OPENSIFT CONTAINER PLATFORM をセキュリティ保護する方法	9
1.3.1. TLS サポート	9
第2章 インフラストラクチャーコンポーネント	12
2.1. KUBERNETES インフラストラクチャー	12
2.1.1. 概要	12
2.1.2. マスター	12
2.1.2.1. 高可用性マスター	12
2.1.3. ノード	13
2.1.3.1. Kubelet	14
2.1.3.2. サービスプロキシ	14
2.1.3.3. ノードオブジェクト定義	14
2.2. CONTAINER レジストリー	15
2.2.1. 概要	15
2.2.2. 統合 OpenShift Container レジストリー	15
2.2.3. サードパーティーレジストリー	15
2.2.3.1. 認証	15
2.3. WEB コンソール	15
2.3.1. 概要	16
2.3.2. CLI ダウンロード	17
2.3.3. ブラウザーの要件	17
2.3.4. プロジェクトの概要	17
2.3.5. JVM コンソール	19
2.3.6. StatefulSets	20
第3章 コアとなる概念	22
3.1. 概要	22
3.2. コンテナおよびイメージ	22
3.2.1. コンテナ	22
3.2.1.1. Init コンテナ	22
3.2.2. イメージ	23
イメージバージョンタグポリシー	23
3.2.3. Container Registries	24
3.3. POD およびサービス	24
3.3.1. Pod	24
3.3.1.1. Pod 再起動ポリシー	27
3.3.1.2. Injecting Information into Pods Using Pod Presets	28
3.3.2. Init コンテナ	28
3.3.3. Services (サービス)	29
3.3.3.1. サービス externalIP	30
3.3.3.2. サービス ingressIP	31
3.3.3.3. サービス NodePort	32
3.3.3.4. サービスプロキシモード	32
3.3.3.5. ヘッドレスサービス	32
3.3.3.5.1. ヘッドレスサービスの作成	33
3.3.3.5.2. ヘッドレスサービスを使用したエンドポイントの検出	34
3.3.4. ラベル	34
3.3.5. Endpoints (エンドポイント)	35

3.4. プロジェクトとユーザー	35
3.4.1. ユーザー	35
3.4.2. Namespace	35
3.4.3. プロジェクト	36
3.4.3.1. インストール時にプロビジョニングされるプロジェクト	37
3.5. ビルドおよびイメージストリーム	37
3.5.1. ビルド	37
3.5.1.1. Docker ビルド	37
3.5.1.2. Source-to-Image (S2I) ビルド	38
3.5.1.3. カスタムビルド	38
3.5.1.4. Pipeline ビルド	39
3.5.2. イメージストリーム	39
3.5.2.1. 重要な用語	41
3.5.2.2. イメージストリームの設定	42
3.5.2.3. イメージストリームイメージ	43
3.5.2.4. イメージストリームタグ	44
3.5.2.5. イメージストリーム変更トリガー	45
3.5.2.6. イメージストリームのマッピング	46
3.5.2.7. イメージストリームの使用	48
3.5.2.7.1. イメージストリームについての情報の取得	48
3.5.2.7.2. 追加タグのイメージストリームへの追加	49
3.5.2.7.3. 外部イメージのタグの追加	50
3.5.2.7.4. イメージストリームタグの更新	50
3.5.2.7.5. イメージストリームタグのイメージストリームからの削除	51
3.5.2.7.6. タグの定期的なインポートの設定	51
3.6. デプロイメント	51
3.6.1. レプリケーションコントローラー	51
3.6.2. ジョブ	52
3.6.3. デプロイメントおよびデプロイメント設定	53
3.7. TEMPLATES (テンプレート)	54
3.7.1. 概要	54
第4章 追加の概念	55
4.1. 認証	55
4.1.1. 概要	55
4.1.2. ユーザーとグループ	55
4.1.3. API 認証	55
4.1.3.1. 権限の借用	56
4.1.4. OAuth	56
4.1.4.1. OAuth クライアント	57
4.1.4.2. OAuth クライアントとしてのサービスアカウント	58
4.1.4.3. OAuth クライアントとしてのサービスアカウントの URI のリダイレクト	58
4.1.4.3.1. OAuth の API イベント	60
4.1.4.3.1.1. 誤設定の場合に引き起こされる API イベントのサンプル	62
4.1.4.4. 統合	64
4.1.4.5. OAuth サーバーメタデータ	64
4.1.4.6. OAuth トークンの取得	66
4.1.4.7. Prometheus の認証メトリクス	66
4.2. 承認	67
4.2.1. 概要	67
4.2.2. 承認の評価	72
4.2.3. クラスターおよびローカル RBAC	73
4.2.4. クラスターロールおよびローカルロール	73

4.2.4.1. クラスターロールの更新	74
4.2.4.2. カスタムロールおよびパーミッションの適用	74
4.2.4.3. クラスターロールの集計	75
4.2.5. SCC (Security Context Constraints)	75
4.2.5.1. SCC ストラテジー	78
4.2.5.1.1. RunAsUser	78
4.2.5.1.2. SELinuxContext	79
4.2.5.1.3. SupplementalGroups	79
4.2.5.1.4. FSGroup	79
4.2.5.2. ポリ्यूムの制御	79
4.2.5.3. FlexVolume へのアクセスの制限	80
4.2.5.4. Seccomp	81
4.2.5.5. 受付	81
4.2.5.5.1. SCC の優先度設定	82
4.2.5.5.2. 事前に割り当てられた値および SCC (Security Context Constraints) について	82
4.2.6. 認証済みのユーザーとして何が実行できるのかを判断する方法	83
4.3. 永続ストレージ	84
4.3.1. 概要	84
4.3.2. Lifecycle of a Volume and Claim	84
4.3.2.1. Provisioning	84
4.3.2.2. Binding	84
4.3.2.3. Using	85
4.3.2.4. Persistent Volume Claim Protection	85
4.3.2.5. Releasing	86
4.3.2.6. Reclaiming	86
4.3.2.6.1. Recycling	86
4.3.3. Persistent Volumes	87
4.3.3.1. Types of Persistent Volumes	87
4.3.3.2. 容量	88
4.3.3.3. Access Modes	88
4.3.3.4. Reclaim Policy	90
4.3.3.5. フェーズ	90
4.3.3.6. Mount Options	91
4.3.4. Persistent Volume Claims	92
4.3.4.1. Storage Class	92
4.3.4.2. Access Modes	92
4.3.4.3. リソース	92
4.3.4.4. Claims As Volumes	93
4.3.5. Block Volume Support	93
4.4. ソースコントロール管理	96
4.5. 受付コントローラー	96
4.5.1. 概要	96
4.5.2. 一般的な受付ルール	97
4.5.3. カスタマイズ可能な受付プラグイン	98
4.5.4. コンテナを使用した受付コントローラー	98
4.6. カスタム受付コントローラー	98
4.6.1. 概要	98
4.6.2. 受付 Webhook	98
4.6.2.1. 受付 Webhook のタイプ	100
4.6.2.2. 受付 Webhook を作成します。	103
4.6.2.3. 受付 Webhook オブジェクトのサンプル	104
4.7. 他の API オブジェクト	105
4.7.1. LimitRange	105

4.7.2. ResourceQuota	105
4.7.3. リソース	105
4.7.4. Secret	105
4.7.5. PersistentVolume	105
4.7.6. PersistentVolumeClaim	105
4.7.6.1. カスタムリソース	105
4.7.7. OAuth オブジェクト	106
4.7.7.1. OAuthClient	106
4.7.7.2. OAuthClientAuthorization	107
4.7.7.3. OAuthAuthorizeToken	107
4.7.7.4. OAuthAccessToken	108
4.7.8. ユーザーオブジェクト	109
4.7.8.1. Identity	109
4.7.8.2. ユーザー	110
4.7.8.3. UserIdentityMapping	110
4.7.8.4. グループ	111
第5章 ネットワーク	112
5.1. ネットワーク	112
5.1.1. 概要	112
5.1.2. OpenShift Container Platform DNS	112
5.2. OPENSIFT SDN	113
5.2.1. 概要	113
5.2.2. マスター上の設計	113
5.2.3. ノード上の設計	114
5.2.4. パケットフロー	115
5.2.5. ネットワーク分離	115
5.3. 利用可能な SDN プラグイン	116
5.3.1. OpenShift SDN	116
5.3.2. サードパーティーの SDN プラグイン	116
5.3.2.1. Flannel SDN	116
5.3.2.2. Nuage SDN	117
5.4. 利用可能なルータープラグイン	120
5.4.1. Default HAProxy Router	121
5.4.2. HAProxy Template Router	121
5.4.3. F5 BIG-IP® Router plug-in	125
5.4.3.1. Routing Traffic to Pods Through the SDN	125
5.4.3.2. F5 Integration Details	126
5.4.3.3. F5 Native Integration	126
Connection	126
Data Flow: Packets to Pods	127
Data Flow from the F5 Host	128
Data Flow: Return Traffic to the F5 Host	128
5.5. ポート転送	129
5.5.1. 概要	129
5.5.2. サーバー操作	129
5.6. リモートコマンド	129
5.6.1. 概要	129
5.6.2. サーバー操作	129
5.7. ROUTES (ルート)	130
5.7.1. 概要	130
5.7.2. ルーター	130
5.7.2.1. テンプレートルーター	131

5.7.3. 利用可能なルータープラグイン	131
5.7.4. スティックセッション	131
5.7.5. ルーターの環境変数	132
5.7.6. ロードバランシングストラテジー	138
5.7.7. HAProxy Strict SNI	139
5.7.8. ルーターの暗号スイート	139
5.7.9. ルートホスト名	140
5.7.10. ルートタイプ	141
5.7.10.1. パスベースのルート	141
5.7.10.2. セキュリティー保護されたルート	142
5.7.11. ルーターのシャード化	146
5.7.12. 他のバックエンドおよび重み	147
5.7.13. ルート固有のアノテーション	148
5.7.14. ルート固有の IP ホワイトリスト	150
5.7.15. ワイルドカードサブドメインポリシーを指定するルートの作成	151
5.7.16. ルートステータス	151
5.7.17. ルート内の特定ドメインの拒否または許可	151
5.7.18. Namespace 所有権チェックの無効化	153
第6章 サービスカタログコンポーネント	155
6.1. サービスカタログ	155
6.1.1. 概要	155
6.1.2. 設計	155
6.1.2.1. リソースの定義	156
6.1.3. 概念および用語	156
6.1.4. 提供されるクラスターサービスブローカー	159
6.2. テンプレートサービスブローカー	159
6.3. OPENSIFT ANSIBLE BROKER	159
6.3.1. 概要	159
6.3.2. Ansible Playbook Bundle	160

第1章 概要

OpenShift v3 は、基礎となる Docker 形式のコンテナイメージおよび Kubernetes 概念を可能な限り正確に公開することを目的にレイヤー化されたシステムであり、開発者がアプリケーションを簡単に作成できることに重点が置かれています。たとえば、Ruby のインストール、コードのプッシュ、および MySQL の追加などを簡単に実行できます。

OpenShift v2 とは異なり、作成後の設定ではモデルのすべての側面において柔軟性が向上されています。アプリケーションを別個のオブジェクトとみなす概念は削除され、より柔軟性の高い「サービス」の作成という概念が利用されるようになり、2つの Web コンテナでデータベースを再使用したり、データベースをネットワークに直接公開したりできるようになりました。

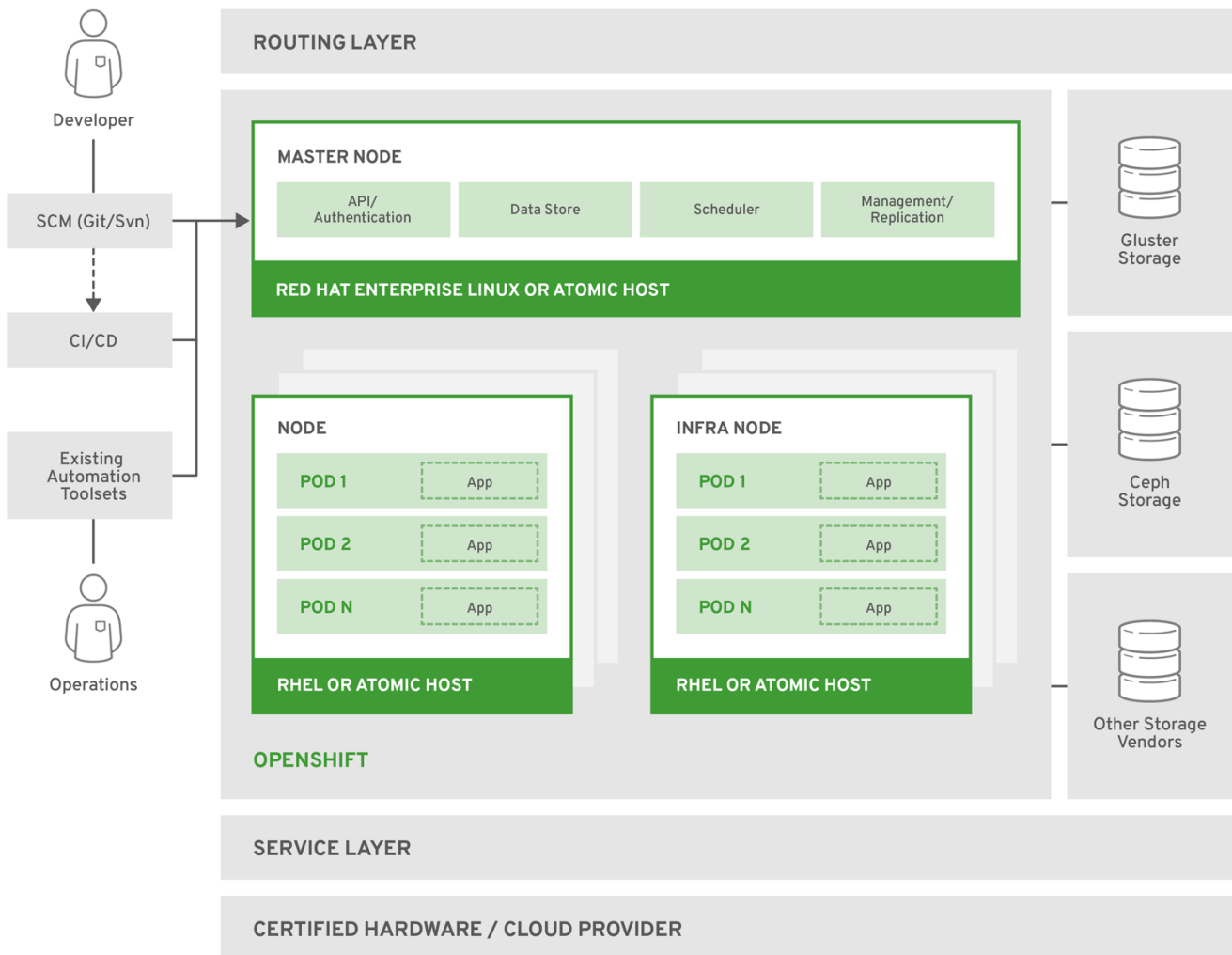
1.1. レイヤーについて

Docker サービスは、Linux ベースの軽量なコンテナイメージをパッケージ化して作成するために抽象化を可能にします。Kubernetes はクラスター管理を行い、複数のホストでコンテナのオーケストレーションを行います。

OpenShift Container Platform は以下を追加します。

- Source code management, [builds](#), and [deployments](#) for developers
- システム全体で移行する [イメージ](#) の大規模な管理およびプロモート
- 大規模なアプリケーション管理
- 大規模な開発者組織を編成するためのチームおよびユーザー追跡
- クラスターをサポートするネットワークインフラストラクチャー

図1.1 OpenShift Container Platform アーキテクチャーの概要



OPENSIFT_415489_0218

1.2. OPENSIFT CONTAINER PLATFORM アーキテクチャーについて

OpenShift Container Platform のアーキテクチャーは、連携する小規模な分割されたユニットからなるマイクロサービスベースとなっており、[Kubernetes クラスタ](#)で実行されます。この際、オブジェクト関連のデータは、信頼できるクラスター化されたキーと値のストアである [etcd](#) に保存されます。これらのサービスは機能別に分類されています。

- [REST APIs](#), which expose each of the [core objects](#).
- これらの API を読み取るコントローラーは変更を別のオブジェクトに適用し、ステータスを報告し、オブジェクトに再び書き込みます。

Users make calls to the REST API to change the state of the system. Controllers use the REST API to read the user's desired state, and then try to bring the other parts of the system into sync. For example, when a user requests a [build](#) they create a "build" object. The build controller sees that a new build has been created, and runs a process on the cluster to perform that build. When the build completes, the controller updates the build object via the REST API and the user sees that their build is complete.

コントローラーパターンとは、OpenShift Container Platform の機能の多くが拡張可能であることを意味しています。ビルドを実行し、起動する方法は、イメージ管理方法や[デプロイメント](#)が実行される方法とは独立してカスタマイズできます。コントローラーはシステムの「ビジネスロジック」を実行し、ユーザーのアクションを実行して、それを実際に実装します。これらのコントローラーをカスタマイズするか、またはこれらを独自のロジックに置き換えることにより、複数の異なる動作を実装できます。システム管理の視点では、これは API を使用して繰り返されるスケジュールで共通の管理アクションに

ついでにスクリプトを作成できることを意味しています。これらのスクリプトは変更を確認し、アクションを実行するコントローラーでもあります。OpenShift Container Platform でこの方法でクラスターをカスタマイズする機能をファーストクラスの動作として使用できます。

コントローラーは、これを可能にするために、システムへの変更が含まれる、信頼できるストリームを活用して、システムのビューとユーザーの実行内容とを同期します。このイベントストリームは、変更の発生後すぐに、etcd から REST API に変更をプッシュしてから、コントローラーにプッシュするので、システムへの変更は、非常に素早くかつ効率的に伝搬できます。ただし、障害はいつでも発生する可能性があるため、コントローラーは、起動時にシステムの最新状態を取得し、すべてが適切な状態であることを確認できる必要があります。このような再同期は、問題が発生した場合でも、オペレーターが影響を受けたコンポーネントを再起動して、システムによる全体の再チェックを実行してから続行できるので、重要です。コントローラーはシステムの同期をいつでも行えるので、システムは最終的に、ユーザーの意図に合わせて収束されるはずで

1.3. OPENSIFT CONTAINER PLATFORM をセキュリティー保護する方法

OpenShift Container Platform および Kubernetes API は、認証情報を提示するユーザーの認証を行ってから、それらのロールに基づいてユーザーの認可します。開発者および管理者はどちらも多くの方法で認証できますが、主に OAuth トークンおよび X.509 クライアント証明書が使用されます。OAuth トークンは JSON Web Algorithm RS256 を使用して署名されます。これは、SHA-256 を使用した RSA 署名アルゴリズムです。

Developers (clients of the system) typically make REST API calls from a [client program](#) like `oc` or to the [web console](#) via their browser, and use OAuth bearer tokens for most communications. Infrastructure components (like nodes) use client certificates generated by the system that contain their identities. Infrastructure components that run in containers use a token associated with their [service account](#) to connect to the API.

認可は、「Pod の作成」または「サービスの一覧表示」などのアクションを定義する OpenShift Container Platform ポリシーエンジンで処理され、それらをポリシードキュメントのロールにグループ化します。ロールは、ユーザーまたはグループ ID によってユーザーまたはグループにバインドされます。ユーザーまたはサービスアカウントがアクションを試行すると、ポリシーエンジンはユーザーに割り当てられた1つ以上のロール (例: クラスター管理者または現行プロジェクトの管理者) をチェックし、その継続を許可します。

Since every container that runs on the cluster is associated with a service account, it is also possible to associate [secrets](#) to those service accounts and have them automatically delivered into the container. This enables the infrastructure to manage secrets for pulling and pushing images, builds, and the deployment components, and also allows application code to easily leverage those secrets.

1.3.1. TLS サポート

All communication channels with the REST API, as well as between [master components](#) such as etcd and the API server, are secured with TLS. TLS provides strong encryption, data integrity, and authentication of servers with X.509 server certificates and public key infrastructure. By default, a new internal PKI is created for each deployment of OpenShift Container Platform. The internal PKI uses 2048 bit RSA keys and SHA-256 signatures. [Custom certificates](#) for public hosts are supported as well.

OpenShift Container Platform は Golang の標準ライブラリーの実装である `crypto/tls` を使用し、外部の `crypto` および TLS ライブラリーには依存しません。追加で、外部ライブラリーに依存して、クライアントは GSSAPI 認証および OpenPGP 署名を使用できます。GSSAPI は通常 OpenSSL の `libcrypto` を使用する MIT Kerberos または Heimdal Kerberos のいずれかによって提供されます。OpenPGP 署名の検証は `libpgpme` および `GnuPG` によって処理されます。

セキュアでない SSL 2.0 および SSL 3.0 バージョンは、サポート対象外であり、利用できません。OpenShift Container Platform サーバーおよび `oc` クライアントはデフォルトで TLS 1.2 のみを提供しま

す。TLS 1.0 および TLS 1.1 はサーバー設定で有効にできます。サーバーおよびクライアントは共に認証される暗号化アルゴリズムと完全な前方秘匿性を持つ最新の暗号スイートを優先的に使用します。暗号スイートと RC4、3DES、および MD5 などの非推奨で、セキュアでないアルゴリズムは無効化されています。また、内部クライアント (LDAP 認証など) によっては、TLS 1.0 から 1.2 設定の制限が少なく、より多くの暗号スイートが有効化されています。

表1.1 サポートされる TLS バージョン

TLS バージョン	OpenShift Container Platform Server	oc クライアント	他のクライアント
SSL 2.0	非対応	非対応	非対応
SSL 3.0	非対応	非対応	非対応
TLS 1.0	No [a]	No [a]	Maybe [b]
TLS 1.1	No [a]	No [a]	Maybe [b]
TLS 1.2	Yes	Yes	Yes
TLS 1.3	N/A [c]	N/A [c]	N/A [c]

[a] Disabled by default, but can be enabled in the server configuration.

[b] Some internal clients, such as the LDAP client.

[c] TLS 1.3 is still under development.

以下は OpenShift Container Platform のサーバーの有効にされた暗号スイートの一覧であり、oc クライアントは優先される順序で並べ替えられます。

- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA

第2章 インフラストラクチャーコンポーネント

2.1. KUBERNETES インフラストラクチャー

2.1.1. 概要

Within OpenShift Container Platform, Kubernetes manages containerized applications across a set of containers or hosts and provides mechanisms for deployment, maintenance, and application-scaling. The Docker service packages, instantiates, and runs containerized applications. A Kubernetes cluster consists of one or more masters and a set of nodes.

オプションとして、[高可用性 \(HA\)](#) のマスターを設定し、クラスターから単一障害点がなくなるようにします。



注記

OpenShift Container Platform uses Kubernetes 1.9 and Docker 1.13.

2.1.2. マスター

The master is the host or hosts that contain the master components, including the API server, controller manager server, and etcd. The master manages [nodes](#) in its Kubernetes cluster and schedules [pods](#) to run on nodes.

表2.1 マスターコンポーネント

コンポーネント	説明
API サーバー	The Kubernetes API server validates and configures the data for pods, services, and replication controllers. It also assigns pods to nodes and synchronizes pod information with service configuration. Can be run as a standalone process.
etcd	etcd は、コンポーネントが etcd で必要な状態に戻すための変更の有無を確認する間に永続マスター状態を保存します。etcd は、通常 2n+1 ピアサービスでデプロイされるなど、高可用性のためにオプションで設定できます。
コントローラーマネージャーサーバー	The controller manager server watches etcd for changes to replication controller objects and then uses the API to enforce the desired state. Can be run as a standalone process. Several such processes create a cluster with one active leader at a time.
HAProxy	Optional, used when configuring highly-available masters with the native method to balance load between API master endpoints. The advanced installation method can configure HAProxy for you with the native method. Alternatively, you can use the native method but pre-configure your own load balancer of choice.

2.1.2.1. 高可用性マスター

While in a single master configuration, the availability of running applications remains if the master or any of its services fail. However, failure of master services reduces the ability of the system to respond

to application failures or creation of new applications. You can optionally configure your masters for high availability (HA) to ensure that the cluster has no single point of failure.

マスターの可用性についての懸念を少なくするために、2つのアクティビティを実行することが推奨されます。

1. **runbook** エントリは、マスターの再作成のために作成される必要があります。runbook エントリは、いずれの高可用性サービスに対しても必要なバックアップです。追加ソリューションは、runbook が参照される頻度を制御するのみです。たとえば、マスターホストのコールドスタンバイは新規アプリケーションの作成または失敗したアプリケーションコンポーネントの復元に1分未満のダウンタイムのみを要求する SLA を十分に満たせる可能性があります。
2. Use a high availability solution to configure your masters and ensure that the cluster has no single point of failure. The [advanced installation method](#) provides specific examples using the **native** HA method and configuring HAProxy. You can also take the concepts and apply them towards your existing HA solutions using the **native** method instead of HAProxy.



注記

Moving from a single master cluster to multiple masters after installation is not supported.

native HA 方式を HAProxy で使用する際に、マスターコンポーネントには以下の可用性があります。

表2.2 HAProxy による可用性マトリクス

ロール	スタイル	備考
etcd	Active-active	ロードバランシング機能のある完全に冗長性のあるデプロイメントです。別個のホストにインストールすることも、マスターホストに共存させることもできます。
API サーバー	Active-active	HAProxy で管理されます。
コントローラーマネージャーサーバー	Active-passive	一度に1つのインスタンスがクラスターリーダーとして選択されます。
HAProxy	Active-passive	API マスターエンドポイント間に負荷を分散します。

クラスター化された etcd では定足数を維持するためにホストの数は奇数である必要がありますが、マスターサービスには定足数やホストの数が奇数でなければならないという要件はありません。ただし、HA 用に2つ以上のマスターサービスが必要になるため、マスターサービスと etcd を共存させる場合には、一律奇数のホストを維持することが一般的です。

2.1.3. ノード

A node provides the runtime environments for containers. Each node in a Kubernetes cluster has the required services to be managed by the [master](#). Nodes also have the required services to run pods, including the Docker service, a [kubelet](#), and a [service proxy](#).

OpenShift Container Platform creates nodes from a cloud provider, physical systems, or virtual systems. Kubernetes interacts with [node objects](#) that are a representation of those nodes. The master uses the

information from node objects to validate nodes with health checks. A node is ignored until it passes the health checks, and the master continues checking nodes until they are valid. The [Kubernetes documentation](#) has more information on node management.

Administrators can [manage nodes](#) in an OpenShift Container Platform instance using the CLI. To define full configuration and security options when launching node servers, use [dedicated node configuration files](#).



重要

See the [cluster limits](#) section for the recommended maximum number of nodes.

2.1.3.1. Kubelet

各ノードには、Pod を記述する YAML ファイルであるコンテナマニフェストで指定されるようにノードを更新する kubelet があります。kubelet は一連のマニフェストを使用して、そのコンテナが起動しており、継続して実行することを確認します。

コンテナマニフェストは以下によって kubelet に提供できます。

- 20 秒ごとにチェックされるコマンドのファイルパス。
- 20 秒ごとにチェックされるコマンドラインで渡される HTTP エンドポイント。
- `/registry/hosts/${hostname -f}` などの etcd サーバーを監視し、変更に作用する kubelet。
- HTTP をリッスンし、単純な API に対応して新規マニフェストを提出する kubelet。

2.1.3.2. サービスプロキシ

各ノードは、ノード上で API で定義されるサービスを反映した単純なネットワークプロキシも実行します。これにより、ノードは一連のバックエンドで単純な TCP および UDP ストリームの転送を実行できます。

2.1.3.3. ノードオブジェクト定義

以下は、Kubernetes のノードオブジェクト定義の例になります。

```
apiVersion: v1 1
kind: Node 2
metadata:
  creationTimestamp: null
  labels: 3
    kubernetes.io/hostname: node1.example.com
  name: node1.example.com 4
spec:
  externalID: node1.example.com 5
status:
  nodeInfo:
    bootID: ""
    containerRuntimeVersion: ""
    kernelVersion: ""
    kubeProxyVersion: ""
    kubeletVersion: ""
```

```
machineID: ""  
osImage: ""  
systemUUID: ""
```

- 1 **apiVersion** は使用する API バージョンを定義します。
- 2 **Node** に設定された **kind** はこれをノードオブジェクトの定義として特定します。
- 3 **metadata.labels** は、ノードに追加されている **ラベル**を一覧表示します。
- 4 **metadata.name** はノードオブジェクトの名前を定義する必須の値です。この値は、**oc get nodes** コマンドの実行時に **NAME** 列に表示されます。
- 5 **spec.externalID** は、ノードに到達できる完全修飾ドメイン名です。空の場合、デフォルトは **metadata.name** 値に設定されます。

2.2. CONTAINER レジストリー

2.2.1. 概要

OpenShift Container Platform can utilize any server implementing the Docker registry API as a source of images, including the Docker Hub, private registries run by third parties, and the integrated OpenShift Container Platform registry.

2.2.2. 統合 OpenShift Container レジストリー

OpenShift Container Platform provides an integrated container registry called **OpenShift Container Registry** (OCR) that adds the ability to automatically provision new image repositories on demand. This provides users with a built-in location for their application **builds** to push the resulting images.

Whenever a new image is pushed to OCR, the registry notifies OpenShift Container Platform about the new image, passing along all the information about it, such as the namespace, name, and image metadata. Different pieces of OpenShift Container Platform react to new images, creating new **builds** and **deployments**.

OCR can also be deployed as a stand-alone component that acts solely as a container registry, without the build and deployment integration. See [Installing a Stand-alone Deployment of OpenShift Container Registry](#) for details.

2.2.3. サードパーティーレジストリー

OpenShift Container Platform はサードパーティーのイメージを使用してコンテナを作成できますが、これらのレジストリーは統合 OpenShift Container Platform レジストリーと同じイメージ通知のサポートがあるわけではありません。取得したタグの更新は、**oc import-image <stream>** の実行と同程度に簡単です。新規イメージが検出されると、前述のビルドおよびデプロイメントの応答が生じます。

2.2.3.1. 認証

OpenShift Container Platform can communicate with registries to access private image repositories using credentials supplied by the user. This allows OpenShift to push and pull images to and from private repositories. The [Authentication](#) topic has more information.

2.3. WEB コンソール

2.3.1. 概要

OpenShift Container Platform Web コンソールは、Web ブラウザーからアクセスできるユーザーインターフェースです。開発者は Web コンソールを使用してプロジェクトのコンテンツの可視化、ブラウザ、および管理を実行できます。



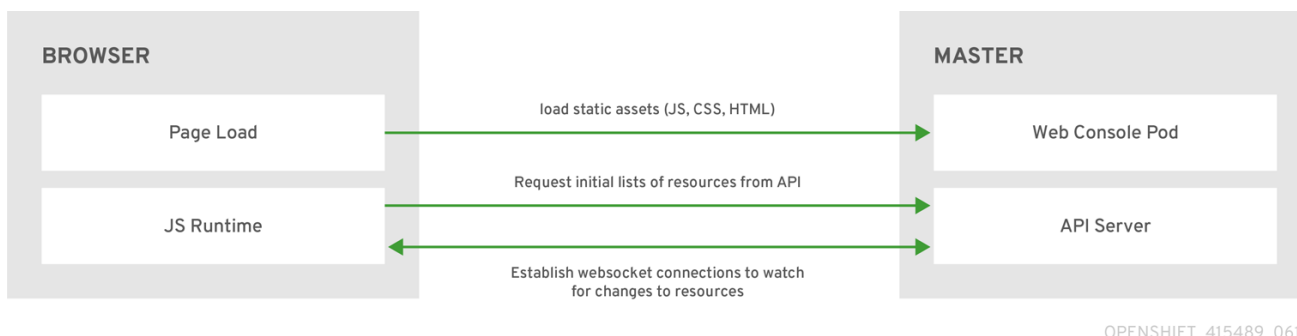
注記

Web コンソールを使用するには JavaScript が有効にされている必要があります。WebSocket をサポートする Web ブラウザーを使用することが最も推奨されます。

The web console runs as a pod on the [master](#). The static assets required to run the web console are served by the pod. Administrators can also [customize the web console](#) using extensions, which let you run scripts and load custom stylesheets when the web console loads.

ブラウザから Web コンソールにアクセスする際に、まず必要な静的アセットをすべて読み込みます。次に、**openshift start** オプションの **--public-master** で定義される値、**openshift-web-console namespace** で定義される **webconsole-config** 設定マップの関連パラメーター **masterPublicURL** から定義される値を使用して、OpenShift Container Platform API に要求を行います。Web コンソールは WebSocket を使用して API サーバーとの永続的な接続を維持し、更新情報を利用可能になる時点で受信します。

図2.1 s Web コンソール要求アーキテクチャー



OPENSIFT_415489_0618

The configured host names and IP addresses for the web console are whitelisted to access the API server safely even when the browser would consider the requests to be [cross-origin](#). To access the API server from a web application using a different host name, you must whitelist that host name by specifying the **--cors-allowed-origins** option on **openshift start** or from the related [master configuration file parameter corsAllowedOrigins](#).

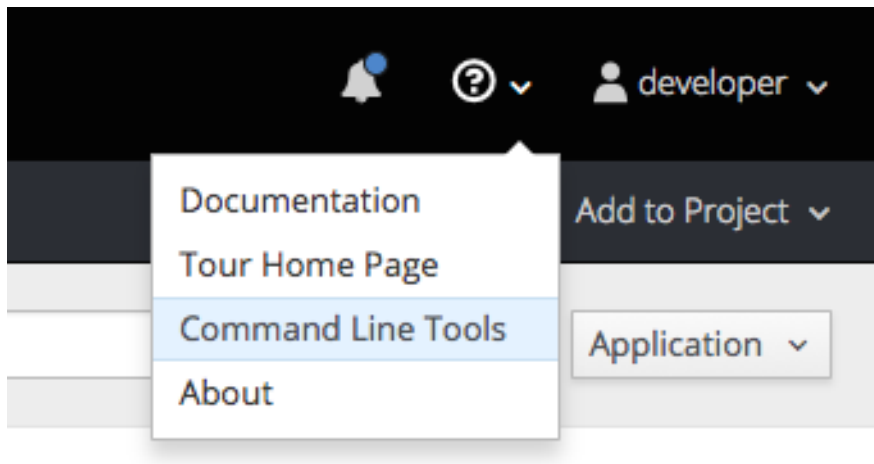
corsAllowedOrigins パラメーターは設定フィールドで制御されます。値に対してピニングやエスケープは実行されません。以下は、ホスト名をピニングし、ドットをエスケープする方法の例を示しています。

```
corsAllowedOrigins:
- (?i)//my\.subdomain\.domain\.com(?:|z)
```

- **(?i)** は大文字/小文字を区別します。
- **//** はドメインの開始にピニングします (または **http:** または **https:** の後のダブルスラッシュに一致します)。
- **\.** はドメイン名のドットをエスケープします。
- **(:|z)** はドメイン名の終了に一致します (**z**) またはポートセパレーター (**:**)。

2.3.2. CLI ダウンロード

Web コンソールのヘルプアイコンから CLI ダウンロードにアクセスできます。



Cluster administrators can [customize these links further](#).

Command Line Tools

With the OpenShift command line interface (CLI), you can create applications and manage OpenShift projects from a terminal. You can download the `oc` client tool using the links below. For more information about downloading and installing it, please refer to the [Get Started with the CLI](#) documentation.

Download `oc` :

[Latest Release](#)

After downloading and installing it, you can start by logging in. You are currently logged into this console as **developer**. If you want to log into the CLI using the same session token:

```
oc login https://127.0.0.1:8443 --token=<hidden>
```



A token is a form of a password. Do not share your API token. To reveal your token, press the copy to clipboard button and then paste the clipboard contents.

After you login to your account you will get a list of projects that you can switch between:

```
oc project <project-name>
```

If you do not have any existing projects, you can create one:

```
oc new-project <project-name>
```

To show a high level overview of the current project:

```
oc status
```

For other information about the command line tools, check the [CLI Reference](#) and [Basic CLI Operations](#).

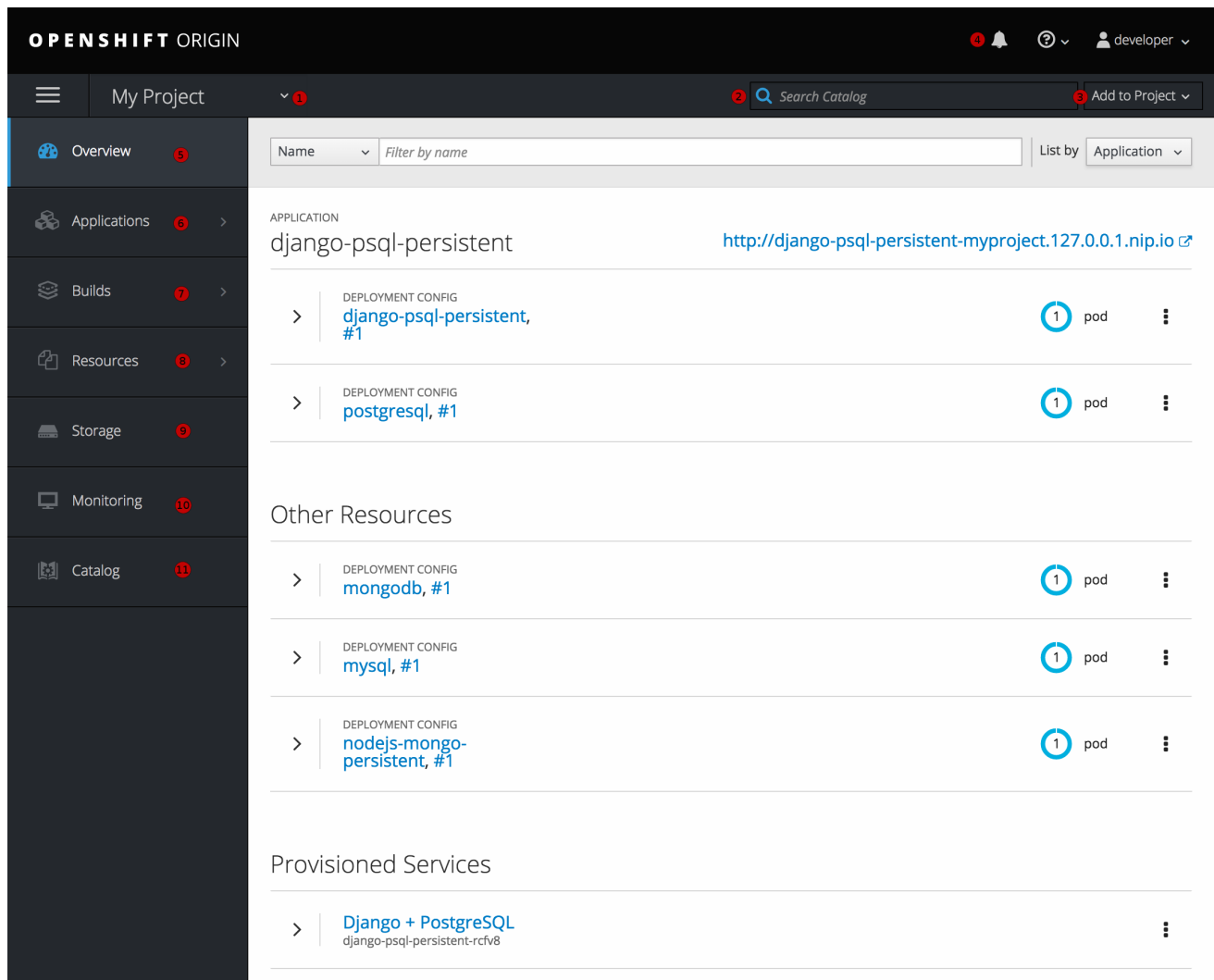
2.3.3. ブラウザーの要件

OpenShift Container Platform の[テスト済みの統合](#)を確認します。

2.3.4. プロジェクトの概要

After [logging in](#), the web console provides developers with an overview for the currently selected [project](#):

図2.2 Web コンソールのプロジェクト概要



The project selector allows you to [switch between projects](#) you have access to.

プロジェクトビューからサービスをすぐに見つけるには、検索条件に入力します。

Create new applications [using a source repository](#) or service from the service catalog.

プロジェクトに関連する通知。

Overview タブ (現在選択されている) は各コンポーネントのハイレベルビューと共にプロジェクトのコンテンツを可視化します。

Applications タブ: デプロイメント、Pod、サービスおよびルートでアクションを参照し、実行します。

Builds タブ: ビルドおよびイメージストリームでアクションを参照し、実行します。

Resources タブ: 現在のクォータの消費およびその他のリソースを表示します。

Storage タブ: Persistent Volume Claim (PVC、永続ボリューム要求) を表示し、アプリケーションのストレージを要求します。

Monitoring タブ: ビルド、Pod、デプロイメントのログ、およびプロジェクトのすべてのオブジェクト通知を表示します。

Catalog タブ: プロジェクト内からカタログにすぐに移動します。



注記

[Cockpit](#) is automatically installed and enabled in OpenShift Container Platform 3.1 and later to help you monitor your development environment. [Red Hat Enterprise Linux Atomic Host: Getting Started with Cockpit](#) provides more information on using Cockpit.






2.3.5. JVM コンソール

Java イメージをベースとする Pod の場合、Web コンソールは関連する統合コンポーネントを表示し、管理するための [hawt.io](#) ベースの JVM コンソールへのアクセスも公開します。**Connect** リンクは、コンテナに `jolokia` という名前のポートがある場合は、**Browse** → **Pods** ページの Pod の詳細に表示されます。

図2.3 JVM コンソールへのリンクを持つ Pod

Template

CONTAINER: STI-BUILD

-  **Image:** openshift/origin-sti-builder:latest
-  **Mount:** docker-socket → /var/run/docker.sock
-  **Mount:** builder-dockercfg-p7gmj-push → /var/run/secrets/openshift.io/push
-  **Mount:** builder-token-t6b9i → /var/run/secrets/kubernetes.io/serviceaccount
-  [Open Java Console](#)

Volumes

docker-socket

Type: host path (bare host directory volume)
Path: /var/run/docker.sock

JVM コンソールへの接続後に、接続されている Pod に関連するコンポーネントに応じて異なるページが表示されます。

図2.4 JVM コンソール

Connected to quickstart-java-camel-spring-container
 ← Back

JMX Threads Camel

Total: 9 Runnable: 3 Timed waiting: 2 Waiting: 4

ID	State	Name	Waited Time	Blocked Time	Native	Suspended
15		Thread-5	1 hour			
14		Camel (camel-1) thread #0 - file://src/data	1 hour			
9		Jolokia Agent Cleanup Thread				
8		Thread-3		279 ms	(in native)	
6		server-timer	1 hour			
4		Signal Dispatcher				
3		Finalizer	1 hour			
2		Reference Handler	1 hour	10 ms		
1		main				

以下のページが利用可能になります。

ページ	説明
JMX	JMX ドメインおよび mbeans を表示し、管理します。
スレッド	スレッドの状態を表示し、モニターします。
ActiveMQ	Apache ActiveMQ ブローカーを表示し、管理します。
Camel	Apache Camel ルートおよび依存関係を表示し、管理します。
OSGi	JBoss Fuse OSGi 環境を表示し、管理します。

2.3.6. StatefulSets

StatefulSet コントローラーは Pod の一意のアイデンティティを提供し、デプロイメントおよびスケーリングの順序を定めます。**StatefulSet** は一意のネットワークアイデンティティ、永続ストレージ、正常なデプロイメントおよびスケーリング、および正常な削除および停止に役立ちます。

図2.5 OpenShift Container Platform の StatefulSet

The screenshot displays the OpenShift console interface for a StatefulSet named 'world'. The breadcrumb navigation shows 'Stateful Sets > world'. The main content area is divided into several sections:

- Status:** Active (indicated by a refresh icon).
- Replicas:** 2 replicas.
- Pod Count:** A large blue circle contains the number '2' with 'pods' written below it.
- Template:** A section for the pod template.
- Containers:** A list of containers for the 'world' StatefulSet:
 - Image:** aosqe/hello-openshift
 - Ports:** 8080/TCP (web)
 - Mount:** volume1 → /var/lib/volume-test read-write
 - Memory:** 256 MiB limit
- Volumes:** A section for the 'volume1' volume:
 - Type:** empty dir (temporary directory destroyed with the pod)
 - Medium:** node's default
- Pods:** A table listing the individual pods.

At the bottom, a message states: "There are no annotations on this resource."

Name	Status	Containers Ready	Container Restarts	Age
world-1	Running	1/1	0	a few seconds
world-0	Running	1/1	0	a few seconds

第3章 コアとなる概念

3.1. 概要

以下のトピックでは、OpenShift Container Platform を使用する際に生じるコアとなる概念およびオブジェクトについてのハイレベルのアーキテクチャー情報を提供します。これらのオブジェクトの多くは、さらに機能が充実した開発ライフサイクルプラットフォームを提供するために OpenShift Container Platform で拡張された Kubernetes のオブジェクトです。

- **コンテナおよびイメージ**は、アプリケーションのビルディングブロックです。
- **Pod およびサービス**は、コンテナの相互通信およびプロキシ接続を許可します。
- **プロジェクトおよびユーザー**は、コミュニティがコンテンツを編成し、管理するためのスペースと手段を提供します。
- **ビルドおよびイメージストリーム**は、作業中のイメージのビルドおよび新規イメージへの応答を許可します。
- **デプロイメント**は、ソフトウェア開発およびデプロイメントライフサイクルの拡張したサポートを追加します。
- **ルート**はサービスを一般に公開します。
- **テンプレート**は多くのオブジェクトがカスタマイズされたパラメーターに基づいて一度に作成されるようにします。

3.2. コンテナおよびイメージ

3.2.1. コンテナ

OpenShift Container Platform アプリケーションの基本的な単位は **コンテナ** と呼ばれています。**Linux コンテナテクノロジー**は、指定されたリソースのみと対話するために実行中のプロセスを分離する軽量なメカニズムです。

数多くのアプリケーションインスタンスは、相互のプロセス、ファイル、ネットワークなどを可視化せずに単一ホストのコンテナで実行される可能性があります。通常、コンテナは任意のワークロードに使用されますが、各コンテナは Web サーバーまたはデータベースなどの (通常は「マイクロサービス」と呼ばれることの多い) 単一サービスを提供します。

Linux カーネルは数年にわたりコンテナテクノロジーの各種機能を統合してきました。最近では、Docker プロジェクトはホストで Linux コンテナの便利な管理インターフェースを開発しました。OpenShift Container Platform および Kubernetes は複数ホストのインストール間で Docker 形式のコンテナのオーケストレーションを実行する機能を追加します。

OpenShift Container Platform の使用時に Docker CLI と直接対話することはないものの、それらの機能および用語を理解しておくことは、OpenShift Container Platform のロールやアプリケーションのコンテナ内での機能を理解する上で重要です。**docker** RPM は RHEL 7、CentOS および Fedora の一部として利用できるため、これを OpenShift Container Platform とは別に実験的に使用することができます。ガイド付きの情報については、『[Get Started with Docker Formatted Container Images on Red Hat Systems](#)』という記事を参照してください。

3.2.1.1. Init コンテナ

Pod にはアプリケーションコンテナのほかに init コンテナがあります。Init コンテナにより、設定スクリプトやバインディングコードを再編成できます。init コンテナは、常に完了するまで実行される点で通常のコンテナとは異なります。各 init コンテナは次のコンテナが起動する前に正常に完了する必要があります。

詳細については、「[Pod およびサービス](#)」を参照してください。

3.2.2. イメージ

OpenShift Container Platform のコンテナは Docker 形式のコンテナの **イメージ** をベースにしています。イメージは、単一コンテナを実行するためのすべての要件、およびそのニーズおよび機能を記述するメタデータを含むバイナリーです。

これはパッケージ化テクノロジーとして考えることができます。コンテナには、作成時にコンテナに追加のアクセスを付与しない限り、イメージで定義されるリソースにのみアクセスできます。同じイメージを複数ホスト間の複数コンテナにデプロイし、それらの間の負荷を分散することにより、OpenShift Container Platform はイメージにパッケージ化されたサービスの冗長および水平的なスケールリングを提供できます。

Docker CLI を直接使用してイメージをビルドすることができますが、OpenShift Container Platform はコードおよび設定を既存イメージに追加して新規イメージの作成を支援するビルダーイメージも提供します。

アプリケーションは時間の経過と共に開発されていくため、単一イメージ名は「同じ」イメージの数多くの異なるバージョンを実際に参照することができます。それぞれの異なるイメージは、通常は 12 文字 (例: **fd44297e2ddb**) に省略されるそのハッシュ (**fd44297e2ddb050ec4f...** などの長い 16 進数) で一意に参照されます。

イメージバージョンタグポリシー

バージョン番号ではなく、Docker サービスはタグ (**v1**、**v2.1**、**GA**、またはデフォルト **latest**) を必要なイメージを指定するためのイメージ名に追加して適用するため、同じイメージが **centos** (これは **latest** タグを意味します)、**centos:centos7**、または **fd44297e2ddb** として参照される場合があります。



警告

Do not use the **latest** tag for any official OpenShift Container Platform images. These are images that start with **openshift3/**. **latest** can refer to a number of versions, such as **3.4**, or **3.5**.

イメージへのタグの付け方は更新ポリシーを定めます。より具体的なタグを使用すると、イメージが更新される頻度は低くなります。以下を使用して選択した OpenShift Container Platform イメージポリシーを決定します。

vX.Y

The vX.Y tag points to X.Y.Z-<number>. For example, if the **registry-console** image is updated to v3.4, it points to the newest 3.4.Z-<number> tag, such as 3.4.1-8.

X.Y.Z

Similar to the vX.Y example above, the X.Y.Z tag points to the latest X.Y.Z-<number>. For example, 3.4.1 would point to 3.4.1-8

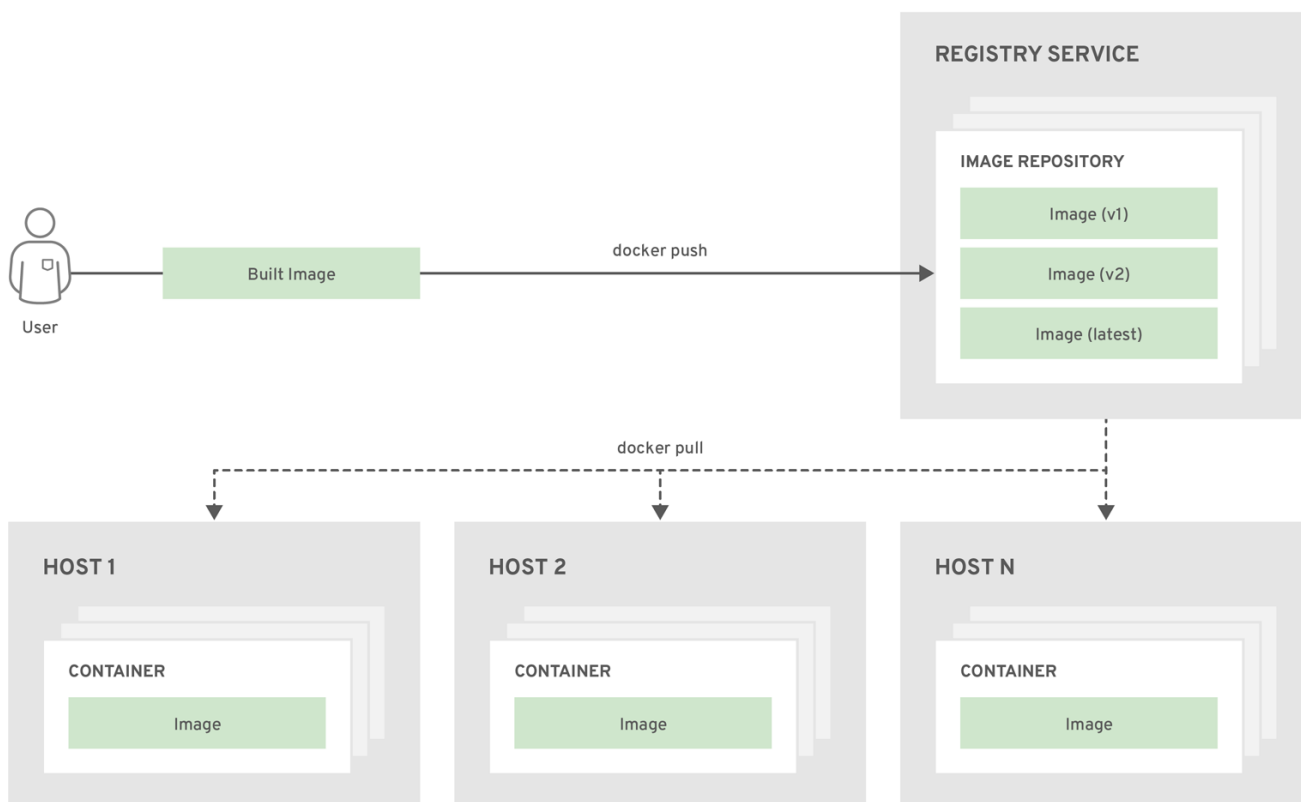
X.Y.Z-<number>

The tag is unique and does not change. When using this tag, the image does not update if an image is updated. For example, the 3.4.1-8 will always point to 3.4.1-8, even if an image is updated.

3.2.3. Container Registries

A container registry is a service for storing and retrieving Docker-formatted container images. A registry contains a collection of one or more image repositories. Each image repository contains one or more tagged images. Docker provides its own registry, the [Docker Hub](#), and you can also use private or third-party registries. Red Hat provides a registry at registry.access.redhat.com for subscribers. OpenShift Container Platform can also supply its own internal registry for managing custom container images.

以下の図では、コンテナ、イメージ、およびレジストリー間の関係が描写されています。



OPENSIFT_415489_0218

3.3. POD およびサービス

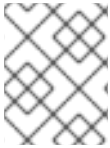
3.3.1. Pod

OpenShift Container Platform は、**Pod** の Kubernetes の概念を使用します。これはホスト上に共にデプロイされる1つ以上の**コンテナ**であり、定義され、デプロイされ、管理される最小のコンピュータ単位です。

Pod はコンテナに対するマシンインスタンス (物理または仮想) とほぼ同じです。各 Pod は独自の内部 IP アドレスで割り当てられるため、そのポートスペース全体を所有し、Pod 内のコンテナはそれらのローカルストレージおよびネットワークを共有できます。

Pod にはライフサイクルがあります。それらは定義されてから、ノードを実行するために割り当てられ、コンテナが終了するまで実行されるか、その他の理由でコンテナが削除されるまで実行されます。ポリシーおよび終了コードによっては、Pod は終了後に削除されるか、コンテナのログへのアクセスを有効にするために保持される可能性があります。

OpenShift Container Platform は Pod をほとんどがイミュータブルなものとして処理します。Pod が実行中の場合は Pod に変更を加えることができません。OpenShift Container Platform は既存 Pod を終了し、これを変更された設定、ベースイメージのいずれかまたはその両方で再作成して変更を実装します。Pod は拡張可能なものとして処理されますが、再作成時に状態を維持しません。そのため、通常 Pod はユーザーから直接管理されるのではなく、ハイレベルの [コントローラー](#) で管理される必要があります。



注記

For the maximum number of pods per OpenShift Container Platform node host, see the [Cluster Limits](#).



警告

[レプリケーションコントローラー](#)によって管理されないベア Pod はノードの中断時に再スケジュールされません。

Below is an example definition of a pod that provides a long-running service, which is actually a part of the OpenShift Container Platform infrastructure: the integrated container registry. It demonstrates many features of pods, most of which are discussed in other topics and thus only briefly mentioned here:

例3.1 Pod オブジェクト定義 (YAML)

```

apiVersion: v1
kind: Pod
metadata:
  annotations: { ... }
  labels: 1
    deployment: docker-registry-1
    deploymentconfig: docker-registry
    docker-registry: default
  generateName: docker-registry-1- 2
spec:
  containers: 3
  - env: 4
    - name: OPENSIFT_CA_DATA
      value: ...
    - name: OPENSIFT_CERT_DATA
      value: ...
    - name: OPENSIFT_INSECURE
      value: "false"
    - name: OPENSIFT_KEY_DATA
      value: ...
    - name: OPENSIFT_MASTER
      value: https://master.example.com:8443
  image: openshift/origin-docker-registry:v0.6.2 5
  imagePullPolicy: IfNotPresent
  name: registry
  ports: 6

```

```

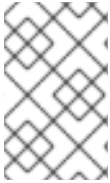
- containerPort: 5000
  protocol: TCP
  resources: {}
  securityContext: { ... }
  volumeMounts:
  - mountPath: /registry
    name: registry-storage
  - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
    name: default-token-br6yz
    readOnly: true
  dnsPolicy: ClusterFirst
  imagePullSecrets:
  - name: default-dockercfg-at06w
  restartPolicy: Always
  serviceAccount: default
  volumes:
  - emptyDir: {}
    name: registry-storage
  - name: default-token-br6yz
    secret:
      secretName: default-token-br6yz

```

- 1 Pod には1つまたは複数のラベルで「タグ付け」することができ、このラベルを使用すると、一度の操作で Pod グループの選択や管理が可能になります。これらのラベルは、キー/値形式でメタデータ ハッシュに保存されます。この例で使用されているラベルは `docker-registry=default` です。
- 2 Pod にはそれらの namespace 内に任意の名前がなければなりません。Pod 定義は `generateName` 属性で名前のベースを指定できますが、一意の名前を生成するためにランダムな文字が自動的に追加されます。
- 3 コンテナはコンテナ定義の配列を指定します。この場合(ほとんどの場合)、これは1つのみになります。
- 4 必要な値を各コンテナに渡すために、環境変数を指定することができます。
- 5 Pod の各コンテナは独自の Docker 形式のコンテナイメージからインスタンス化されます。
- 6 コンテナは、Pod の IP で利用可能にされるポートにバインドできます。
- 7 OpenShift Container Platform defines a `security context` for containers which specifies whether they are allowed to run as privileged containers, run as a user of their choice, and more. The default context is very restrictive but administrators can modify this as needed.
- 8 コンテナは外部ストレージボリュームがコンテナ内にマウントされるかどうかを指定します。この場合、レジストリーのデータを保存するためのボリュームと、OpenShift Container Platform API に対して要求を行うためにレジストリーが必要とする認証情報へのアクセス用のボリュームがあります。
- 9 Pod 再起動ポリシーと使用可能な値の **Always**、**OnFailure**、および **Never** です。デフォルト値は **Always** です。
- 10 Pods making requests against the OpenShift Container Platform API is a common enough pattern that there is a `serviceAccount` field for specifying which `service account` user the pod should authenticate as when making the requests. This enables fine-grained access control for custom

infrastructure components.

- 11 Pod は、コンテナで使用できるストレージボリュームを定義します。この場合、レジストリーストレージの一時的なボリュームおよびサービスアカウントの認証情報が含まれる **シークレット** ボリュームが提供されます。



注記

この Pod 定義には、Pod が作成され、ライフサイクルが開始された後に OpenShift Container Platform によって自動的に設定される属性が含まれません。[Kubernetes Pod ドキュメント](#)には、Pod の機能および目的についての詳細が記載されています。

3.3.1.1. Pod 再起動ポリシー

Pod 再起動ポリシーは、Pod のコンテナの終了時に OpenShift Container Platform が応答する方法を決定します。このポリシーは Pod のすべてのコンテナに適用されます。

以下の値を使用できます。

- **Always:** Pod が再起動するまで、Pod で正常に終了したコンテナの継続的な再起動を、指数関数のバックオフ遅延 (10 秒、20 秒、40 秒) で試行します。デフォルトは **Always** です。
- **OnFailure:** Pod で失敗したコンテナの継続的な再起動を、5 分を上限として指数関数のバックオフ遅延 (10 秒、20 秒、40 秒) で試行します。
- **Never:** Pod で終了したコンテナまたは失敗したコンテナの再起動を試行しません。Pod はただちに失敗し、終了します。

いったんノードにバインドされた Pod は別のノードにバインドされなくなります。これは、Pod がノードの失敗後も存続するにはコントローラーが必要であることを示しています。

条件	コントローラーのタイプ	再起動ポリシー
(バッチ計算など) 終了することが予想される Pod	ジョブ	OnFailure または Never
(Web サービスなど) 終了しないことが予想される Pod	レプリケーションコントローラー	Always.
マシンごとに1回実行される必要のある Pod	Daemonset	すべて

Pod のコンテナが失敗し、再起動ポリシーが **OnFailure** に設定される場合、Pod はノード上に留まり、コンテナが再起動します。コンテナを再起動させない場合には、再起動ポリシーの **Never** を使用します。

Pod 全体が失敗すると、OpenShift Container Platform は新規 Pod を起動します。開発者はアプリケーションが新規 Pod で再起動される可能性に対応する必要があります。とくに、アプリケーションは、一時的なファイル、ロック、以前の実行で生じた未完了の出力などを処理する必要があります。

OpenShift Container Platform が失敗したコンテナについて再起動ポリシーを使用する方法についての詳細は、Kubernetes ドキュメントの「[Example States](#)」を参照してください。

3.3.1.2. Injecting Information into Pods Using Pod Presets

A **pod preset** is an object that injects user-specified information into pods as they are created.



重要

Pod presets is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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。

Using pod preset objects you can inject:

- [secret objects](#)
- **ConfigMap** objects
- [storage volumes](#)
- container volume mounts
- environment variables

Developers need to ensure the pod labels match the label selector on the PodPreset in order to add all that information to the pod. The [label](#) on a pod associates the pod with one or more pod preset objects that have a matching [label selectors](#).

Using pod presets, a developer can provision pods without needing to know the details about the services the pod will consume. An administrator can keep configuration items of a service invisible from a developer without preventing the developer from deploying pods.



注記

The Pod Preset feature is available only if the [Service Catalog](#) has been installed.

You can exclude specific pods from being injected using the **podpreset.admission.kubernetes.io/exclude: "true"** parameter in the pod specification. See the [example pod specification](#).

For more information, see [Injecting Information into Pods Using Pod Presets](#).

3.3.2. Init コンテナ

init コンテナは、Pod アプリコンテナが起動する前に起動する Pod のコンテナです。Init コンテナはボリュームを共有し、ネットワーク操作を実行し、計算を実行してから残りのコンテナを起動します。Init コンテナは一部の条件が満たされるまでアプリケーションの起動をブロックしたり、遅延させたりすることもできます。

Pod の起動時でボリュームの初期化後に、init コンテナは順番に起動します。各 init コンテナは、次のコンテナが起動する前に正常に終了する必要があります。init コンテナが (ランタイムを原因に) 起動に失敗するか、または失敗して終了する場合、これは Pod の [再起動ポリシー](#)に基づいてリタイ

アします。

Pod は init コンテナがすべて成功するまで準備状態になりません。

一部の [init コンテナの使用例](#) については、Kubernetes ドキュメントを参照してください。

以下の例は、2つの init コンテナを持つ単純な Pod の概要を示しています。最初の init コンテナは **myservice** を待機し、2つ目は **mydb** を待機します。両方のコンテナに成功すると、Pod は起動します。

例3.2 Init コンテナ Pod オブジェクト定義のサンプル (YAML)

```

apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice ❶
    image: busybox
    command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice; sleep 2; done;']
  - name: init-mydb ❷
    image: busybox
    command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sleep 2; done;']

```

❶ **myservice** コンテナを指定します。

❷ **mydb** コンテナを指定します。

Each init container has all of the [fields of an app container](#) except for [readinessProbe](#). Init containers must exit for pod startup to continue and cannot define readiness other than completion.

Init containers can include [activeDeadlineSeconds](#) on the pod and [livenessProbe](#) on the container to prevent init containers from failing forever. The active deadline includes init containers.

3.3.3. Services (サービス)

Kubernetes [サービス](#) は内部ロードバランサーとして機能します。これは、受信する接続をプロキシして送信するために一連のレプリケートされた [Pod](#) を特定します。バックアップ Pod は、サービスが一貫して利用可能な状態の間に任意でサービスに追加されたり、削除されたりします。これにより、サービスに依存して同じアドレスの Pod を参照するすべてのものを有効にします。デフォルトのサービス clusterIP アドレスは OpenShift Container Platform 内部ネットワークからのもので、Pod が相互にアクセスできるように使用されます。

To permit external access to the service, additional [externalIP](#) and [ingressIP](#) addresses that are [external](#) to the cluster can be assigned to the service. These [externalIP](#) addresses can also be virtual IP addresses that provide [highly available](#) access to the service.

サービスには IP アドレスとポートのペアが割り当てられるため、アクセスされる際に、適切なバックギングポートにプロキシ送信されます。サービスは、ラベルセクターを使用して特定ポートで特定のネットワークサービスを提供する実行中のすべてのコンテナを見つけます。

Pod と同様に、サービスは REST オブジェクトです。以下の例は、上記の定義された Pod のサービス定義を示しています。

例3.3 サービスオブジェクト定義 (YAML)

```
apiVersion: v1
kind: Service
metadata:
  name: docker-registry ①
spec:
  selector:
    docker-registry: default ②
  clusterIP: 172.30.136.123 ③
  ports:
  - nodePort: 0
    port: 5000 ④
    protocol: TCP
    targetPort: 5000 ⑤
```

- ① サービス名 **docker-registry** は、同じ namespace の別の Pod に挿入されるサービス IP で環境変数を作成するためにも使用されます。名前の最大長さは 63 文字です。
- ② ラベルセクターは、すべての Pod をバックギング Pod として割り当てられる **docker-registry=default** ラベルで識別します。
- ③ 作成時に内部 IP のプールから自動的に割り当てられるサービスの仮想 IP です。
- ④ サービスがリスンするポートです。
- ⑤ サービスが接続を転送するバックギング Pod のポートです。

[Kubernetes ドキュメント](#) には、サービスについての詳細が記載されています。

3.3.3.1. サービス externalIP

クラスターの内部 IP アドレスに加えて、ユーザーはクラスターの外部にある IP アドレスを設定することができます。管理者は、トラフィックがこの IP を持つノードに到達することを確認する必要があります。

The externalIPs must be selected by the cluster administrators from the **ExternalIPNetworkCIDRs** range configured in **master-config.yaml** file. When **master-config.yaml** is changed, the master services must be restarted.

例3.4 Sample ExternalIPNetworkCIDR /etc/origin/master/master-config.yaml

```
networkConfig:
  ExternalIPNetworkCIDR: 192.0.1.0/24
```

例3.5 サービス externalIP 定義 (JSON)

```

{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "my-service"
  },
  "spec": {
    "selector": {
      "app": "MyApp"
    },
    "ports": [
      {
        "name": "http",
        "protocol": "TCP",
        "port": 80,
        "targetPort": 9376
      }
    ],
    "externalIPs": [
      "192.0.1.1"
    ]
  }
}

```

- ① ポート が公開される外部 IP アドレスの一覧です。これは内部 IP アドレス一覧に追加される一覧です。

3.3.3.2. サービス ingressIP

クラウド以外のクラスターで、externalIP アドレスは、アドレスのプールから自動的に割り当てることができます。これにより、管理者がそれらを手動で割り当てる必要がなくなります。

プールは `/etc/origin/master/master-config.yaml` ファイルで設定されます。このファイルを変更した後にマスターサービスを再起動します。

`ingressIPNetworkCIDR` はデフォルトで `172.29.0.0/16` に設定されます。クラスター環境でこのプライベート範囲を使用していない場合、デフォルトの範囲を使用するか、またはカスタム範囲を使用します。



注記

If you are using [high availability](#), then this range must be less than 256 addresses.

例3.6 サンプル ingressIPNetworkCIDR /etc/origin/master/master-config.yaml

```

networkConfig:
  ingressIPNetworkCIDR: 172.29.0.0/16

```

3.3.3.3. サービス NodePort

サービス **type=NodePort** を設定して、フラグで設定された範囲 (デフォルト: 30000-32767) からポートを割り当て、各ノードはポート (すべてのノードの同じポート番号) をサービスにプロキシー送信します。

選択されたポートは、サービス設定の **spec.ports[*].nodePort** の下に報告されます。

カスタムポートを指定するには、単純にポート番号を **nodePort** フィールドに配置します。カスタムポート番号は **nodePorts** の指定された範囲内になければなりません。'master-config.yaml' が変更される場合、マスターサービスは再起動する必要があります。

例3.7 サンプル servicesNodePortRange /etc/origin/master/master-config.yaml

```
kubernetesMasterConfig:
  servicesNodePortRange: ""
```

サービスは **<NodeIP>:spec.ports[].nodePort** および **spec.clusterIp:spec.ports[].port** として表示されます。



注記

nodePort の設定は特権付きの操作で実行されます。

3.3.3.4. サービスプロキシーモード

OpenShift Container Platform にはサービスルーティングインフラストラクチャーの2つの異なる実装があります。デフォルトの実装は完全に **iptables** をベースとしており、エンドポイント Pod 間の受信サービス接続を分散するための確率的な **iptables** 再作成ルールを使用します。古い方の実装はユーザースペースプロセスを使用して受信接続を受け入れた後に、クライアントとエンドポイント Pod のいずれかの間のトラフィックをプロキシー送信します。

The **iptables**-based implementation is much more efficient, but it requires that all endpoints are always able to accept connections; the user space implementation is slower, but can try multiple endpoints in turn until it finds one that works. If you have good [readiness checks](#) (or generally reliable nodes and pods), then the **iptables**-based service proxy is the best choice. Otherwise, you can enable the user space-based proxy when installing, or after deploying the cluster by editing the node configuration file.

3.3.3.5. ヘッドレスサービス

アプリケーションがロードバランシングや単一サービス IP アドレスを必要しない場合、ヘッドレスサービスを作成できます。ヘッドレスサービスを作成する場合、ロードバランシングやプロキシー送信は実行されず、クラスター IP はこのサービスに割り当てられません。これらのサービスの場合、サービスにセレクターが定義されているかどうかによって DNS が自動的に設定されます。

サービスとセレクター: セレクターを定義するヘッドレスサービスの場合、エンドポイントコントローラーは API の **Endpoints** レコードを作成し、DNS 設定を変更して、サービスをサポートする Pod を直接ポイントする **A** レコード (アドレス) を返します。

セレクターなしのサービス: セレクターを定義しないヘッドレスサービスの場合、エンドポイントコントローラーは **Endpoints** レコードを作成しません。ただし、DNS システムは以下のレコードを検索し、設定します。

- **ExternalName** タイプサービスの場合は、**CNAME** レコードになります。

- それ以外のすべてのサービスタイプの場合、サービスと名前を共有するエンドポイントの **A** レコードになります。

3.3.3.5.1. ヘッドレスサービスの作成

ヘッドレスサービスの作成は標準的なサービスの作成と同様ですが、**ClusterIP** アドレスを宣言しません。ヘッドレスサービスを作成するには、**clusterIP: None** パラメーター値をサービス YAML 定義に追加します。

たとえば、以下は Pod のグループを同じクラスターまたはサービスの一部として組み込む場合です。

List of pods

```
$ oc get pods -o wide
NAME             READY STATUS  RESTARTS  AGE  IP           NODE
frontend-1-287hw 1/1   Running  0         7m   172.17.0.3  node_1
frontend-1-68km5 1/1   Running  0         7m   172.17.0.6  node_1
```

ヘッドレスサービスを以下のように定義できます。

Headless service definition

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: ruby-helloworld-sample
    template: application-template-stibuild
  name: frontend-headless 1
spec:
  clusterIP: None 2
  ports:
  - name: web
    port: 5432
    protocol: TCP
    targetPort: 8080
  selector:
    name: frontend 3
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

- 1** ヘッドレスサービスの名前です。
- 2** **clusterIP** 変数を **None** に設定すると、ヘッドレスサービスが宣言されます。
- 3** **frontend** のラベルが付いたすべての Pod を選択します。

また、ヘッドレスサービスには独自の IP アドレスがありません。

```
$ oc get svc
NAME      TYPE        CLUSTER-IP  EXTERNAL-IP  PORT(S)  AGE
frontend  ClusterIP   172.30.232.77 <none>       5432/TCP 12m
```

```
frontend-headless ClusterIP None <none> 5432/TCP 10m
```

3.3.3.5.2. ヘッドレスサービスを使用したエンドポイントの検出

ヘッドレスサービスを使用する利点として、Pod の IP アドレスを直接検出できることが挙げられます。標準サービスはロードバランサーまたはプロキシとして機能するか、またはサービス名を使用してワークロードオブジェクトへのアクセスを付与します。ヘッドレスサービスの場合には、サービスごとに分類された Pod の IP アドレスセットに、サービス名を解決します。

標準サービスの DNS **A** レコードを検出する際に、サービスの loadbalanced IP を取得します。

```
$ dig frontend.test A +search +short
172.30.232.77
```

ヘッドレスサービスの場合、個別 Pod の IP の一覧を取得します。

```
$ dig frontend-headless.test A +search +short
172.17.0.3
172.17.0.6
```



注記

ヘッドレスサービスを、StatefulSet および初期化および停止時に DNS を解決する必要のあるユースケースで使用する場合は、**publishNotReadyAddresses** を **true** に設定します (デフォルト値は **false** です)。**publishNotReadyAddresses** が **true** に設定されている場合、これは DNS 実装がサービスに関連付けられたエンドポイントのサブセットの **notReadyAddresses** を公開する必要があることを示します。

3.3.4. ラベル

ラベルは、API オブジェクトを編成し、分類し、選択するために使用されます。たとえば、Pod にはラベルで「タグ付け」されてから、サービスはラベルセレクターを使用してそれらがプロキシ送信する Pod を識別します。これにより、サービスが Pod のグループを参照することを可能にし、Pod を関連エンティティとして異なるコンテナで処理することもできます。

ほとんどのオブジェクトには、そのメタデータにラベルを組み込むことができます。そのため、ラベルは任意で関連付けられたオブジェクトを分類するために使用できます。たとえば、特定アプリケーションのすべての Pod、サービス、レプリケーションコントローラー、およびデプロイメント設定を分類できます。

ラベルは、以下の例にあるように単純なキー/値のペアです。

```
labels:
  key1: value1
  key2: value2
```

以下を検討してください。

- nginx コンテナで構成される、ラベル `role=webserver` を持つ Pod。
- Apache httpd コンテナで構成される、同じラベル `role=webserver` を持つ Pod。

`role=webserver` ラベルを持つ Pod を使用するために定義されるサービスまたはレプリケーションコントローラーはこれらの Pod のいずれも同じグループの一部として処理します。

[Kubernetes ドキュメント](#)には、ラベルについての詳細が記載されています。

3.3.5. Endpoints (エンドポイント)

サービスをサポートするサーバーはそのエンドポイントと呼ばれ、サービスと同じ名前を持つタイプ **Endpoints** のオブジェクトで指定されます。サービスが Pod でサポートされる場合、それらの Pod は通常はサービス仕様のラベルセレクターで指定され、OpenShift Container Platform はそれらの Pod をポイントするエンドポイントオブジェクトを自動的に作成します。

In some cases, you may want to create a service but have it be backed by external hosts rather than by pods in the OpenShift Container Platform cluster. In this case, you can leave out the **selector** field in the service, and [create the Endpoints object manually](#).

OpenShift Container Platform は、大半のユーザーが Pod およびサービス用に予約されたネットワークブロックの IP アドレスを参照するエンドポイントオブジェクトの手動による作成を許可しないことに注意してください。**endpoints/restricted**のリソースの **create**パーミッションを持つクラスター管理者その他ユーザーのみがこれらのエンドポイントオブジェクトを作成できます。

3.4. プロジェクトとユーザー

3.4.1. ユーザー

Interaction with OpenShift Container Platform is associated with a user. An OpenShift Container Platform user object represents an actor which may be granted permissions in the system by [adding roles to them or to their groups](#).

ユーザーにはいくつかのタイプが存在します。

通常ユーザー	これは、大半の対話型の OpenShift Container Platform ユーザーが表示される方法です。通常ユーザーは、初回ログイン時にシステムに自動的に作成され、API で作成できます。通常ユーザーは、 User オブジェクトで表示されます。例: joe alice
システムユーザー	Many of these are created automatically when the infrastructure is defined, mainly for the purpose of enabling the infrastructure to interact with the API securely. They include a cluster administrator (with access to everything), a per-node user, users for use by routers and registries, and various others. Finally, there is an anonymous system user that is used by default for unauthenticated requests. Examples: system:admin system:openshift-registry system:node:node1.example.com
サービスアカウント	特殊なシステムユーザーがプロジェクトに関連付けられます。それらの中には、プロジェクト管理者が各プロジェクトのコンテンツへのアクセスを定義する目的で追加作成する状態で、プロジェクトの初回作成時に自動作成されるものがあります。サービスアカウントは ServiceAccount オブジェクトで表示されます。例: system:serviceaccount:default:deployer system:serviceaccount:foo:builder

すべてのユーザーには、OpenShift Container Platform にアクセスするために何らかの**認証**が必要になります。認証がないか、無効な認証を持つ API 要求は、**匿名システムユーザー**によって要求される際に認証されます。認証が実行されると、ユーザーが実行を**認証されている**内容がポリシーによって決定されます。

3.4.2. Namespace

Kubernetes namespace はクラスターのリソースの範囲を設定するメカニズムを提供します。OpenShift Container Platform では、[プロジェクト](#)は追加のアノテーションを含む Kubernetes namespace です。

Namespace は以下の一意的な範囲を提供します。

- 基本的な命名の衝突を避けるための名前付きリソース。
- 信頼されるユーザーに委任された管理権限。
- コミュニティリソースの消費を制限する機能。

システム内の大半のオブジェクトの範囲は namespace で設定されますが、一部はノードやユーザーを含め、除外され、namespace が設定されません。

[Kubernetes ドキュメント](#)には namespace についての詳細が記載されています。

3.4.3. プロジェクト

プロジェクトは追加のアノテーションを持つ Kubernetes namespace であり、通常ユーザーのリソースへのアクセスが管理される中心的な手段です。プロジェクトはユーザーのコミュニティが他のコミュニティとは切り離してコンテンツを編成し、管理することを許可します。ユーザーには、管理者によってプロジェクトへのアクセスが付与される必要があり、許可される場合はプロジェクトを作成でき、それらの独自のプロジェクトへのアクセスが自動的に付与されます。

プロジェクトには、別個の**名前**、**displayName**、および**説明**を含めることができます。

- 必須の **name** はプロジェクトの一意的な ID であり、CLI ツールまたは API を使用する場合に最も明確に表示されます。名前の最大長さは 63 文字です。
- オプションの **displayName** はプロジェクトが Web コンソールで表示される方法を示します (デフォルトは **name** に設定されます)。
- オプションの **description** には、プロジェクトのさらに詳細な記述を施すことができ、これも Web コンソールで表示できます。

各プロジェクトは、以下の独自のセットの範囲を設定します。

オブジェクト	Pod、サービス、レプリケーションコントローラーなど。
ポリシー	ユーザーがオブジェクトに対してアクションを実行できるか、できないかについてのルール。
制約	制限を設定できるそれぞれの種類のオブジェクトのクォータ。
サービスアカウント	サービスアカウントは、プロジェクトのオブジェクトへの指定されたアクセスで自動的に機能します。

Cluster administrators can [create projects](#) and [delegate administrative rights](#) for the project to any member of the user community. Cluster administrators can also allow developers to create [their own projects](#).

Developers and administrators can [interact with projects](#) using [the CLI](#) or the [web console](#).

3.4.3.1. インストール時にプロビジョニングされるプロジェクト

OpenShift Container Platform comes with a number of projects out of the box, and **openshift** is the most essential to users:

openshift A user-facing project, mainly for housing objects for day-to-day tasks. These include any application objects for access by multiple projects, such as templates and images. These objects should be those that do not require communication between the pods.

3.5. ビルドおよびイメージストリーム

3.5.1. ビルド

A **build** is the process of transforming input parameters into a resulting object. Most often, the process is used to transform input parameters or source code into a runnable image. A **BuildConfig** object is the definition of the entire build process.

OpenShift Container Platform leverages Kubernetes by creating Docker-formatted containers from build images and pushing them to a [container registry](#).

ビルドオブジェクトは共通の特性を共有します。これらには、ビルドの入力、ビルドプロセスを完了する必要性、リソースを正常なビルドからパブリッシュすること、およびビルドの最終ステータスをパブリッシュすることが含まれます。ビルドはリソースの制限を利用し、CPU 使用、メモリー使用およびビルドまたは Pod の実行時間などのリソースの制限を設定します。

OpenShift Container Platform ビルドシステムは、ビルド API で指定の、選択可能なタイプに基づくビルドストラテジーを幅広くサポートします。利用可能なビルドストラテジーは主に3つあります。

- [Docker ビルド](#)
- [Source-to-Image \(S2I\) ビルド](#)
- [カスタムビルド](#)

デフォルトで、Docker ビルドおよび S2I ビルドがサポートされます。

ビルドの結果作成されるオブジェクトはこれを作成するために使用されるビルダーによって異なります。Docker および S2I ビルドの場合、作成されるオブジェクトは実行可能なイメージです。カスタムビルドの場合、作成されるオブジェクトはビルダーイメージの作成者が指定したものになります。

さらに、[Pipeline ビルド](#)ストラテジーを使用して、高度なワークフローを実装することができます。

- [継続的インテグレーション](#)
- [継続的デプロイメント](#)

For a list of build commands, see the [Developer's Guide](#).

OpenShift Container Platform の Docker を使用したビルドについての詳細は、[アップストリームドキュメント](#)を参照してください。

3.5.1.1. Docker ビルド

Docker ビルドストラテジーは `docker build` コマンドを起動するため、**Dockerfile** とそれに含まれるすべての必要なアーティファクトのあるのリポジトリが実行可能なイメージを生成することを予想します。

3.5.1.2. Source-to-Image (S2I) ビルド

[Source-to-Image \(S2I\)](#) is a tool for building reproducible, Docker-formatted container images. It produces ready-to-run images by injecting application source into a container image and assembling a new image. The new image incorporates the base image (the builder) and built source and is ready to use with the **docker run** command. S2I supports incremental builds, which re-use previously downloaded dependencies, previously built artifacts, etc.

S2I の利点には以下が含まれます。

イメージの柔軟性	S2I スクリプトを作成して、アプリケーションコードをほとんどすべての既存の Docker 形式コンテナに挿入し、既存のエコシステムを活用することができます。現時点で S2I は tar を使用してアプリケーションソースを挿入するため、イメージは tar が実行されたコンテンツを処理できる必要があることに注意してください。
速度	S2I の場合、アSEMBルプロセスは、各手順で新規の層を作成せずに多数の複雑な操作を実行でき、これによりプロセスが高速になります。さらに、S2I スクリプトを作成すると、ビルドが実行されるたびにダウンロードまたはビルドを実行することなく、アプリケーションイメージの以前のバージョンに保存されたアーティファクトを再利用できます。
パッチ容易性 (Patchability)	S2I により、基礎となるイメージがセキュリティ上の問題でパッチを必要とする場合にアプリケーションを一貫して再ビルドできるようになります。
運用効率	Dockerfile が許可するように任意のアクションを実行する代わりにビルド操作を制限することで、PaaS オペレーターはビルドシステムの意図しない、または意図した誤用を避けることができます。
運用上のセキュリティ	任意の Dockerfile をビルドすると、root の権限昇格のためにホストシステムを公開します。これは、Docker ビルドプロセス全体が Docker 権限を持つユーザーとして実行されるため、悪意あるユーザーが悪用する可能性があります。S2I は root ユーザーとして実行される操作を制限し、スクリプトを root 以外のユーザーとして実行できます。
ユーザー効率	S2I は開発者が任意の yum install タイプの操作を実行することを防ぐため、アプリケーションのビルド時の開発の反復スピードを低下させる可能性があります。
エコシステム	S2I により、アプリケーションのベストプラクティスを利用できるイメージの共有されたエコシステムが促進されます。
再現性	生成されるイメージには、特定バージョンのビルドツールおよび依存関係などのすべての入力が含まれる可能性があります。これにより、イメージを正確に再現できます。

3.5.1.3. カスタムビルド

カスタムビルドストラテジーにより、開発者はビルドプロセス全体を対象とする特定のビルダーイメージを定義できます。独自のビルダーイメージを使用することにより、ビルドプロセスをカスタマイズできます。

A [Custom builder image](#) is a plain Docker-formatted container image embedded with build process logic, for example for building RPMs or base images. The **openshift/origin-custom-docker-builder** image is available on the [Docker Hub](#) registry as an example implementation of a Custom builder image.

3.5.1.4. Pipeline ビルド

開発者は、Pipeline ストラテジーを利用して Jenkins Pipeline プラグインで実行できるように、**Jenkins パイプライン** を定義することができます。ビルドは他のビルドタイプの場合と同様に OpenShift Container Platform での起動、モニタリング、管理が可能です。

Pipeline ワークフローは、ビルド設定に直接組み込むか、Git リポジトリに配置してビルド設定で参照して Jenkinsfile で定義します。

プロジェクトが Pipeline ストラテジーを使用してはじめてビルド設定を定義する場合に、OpenShift Container Platform は Jenkins サーバーをインスタンス化して Pipeline を実行します。プロジェクトの後の Pipeline ビルド設定はこの Jenkins サーバーを共有します。

For more details on how the Jenkins server is deployed and how to configure or disable the autoprovisioning behavior, see [Configuring Pipeline Execution](#).



注記

Jenkins サーバーは、すべての Pipeline ビルド設定が削除されても自動的に削除されないため、ユーザーが手動で削除する必要があります。

Jenkins Pipeline についての詳細は、[Jenkins ドキュメント](#) を参照してください。

3.5.2. イメージストリーム

An image stream and its associated tags provide an abstraction for referencing [Docker images](#) from within OpenShift Container Platform. The image stream and its tags allow you to see what images are available and ensure that you are using the specific image you need even if the image in the repository changes.

イメージストリームには実際のイメージデータは含まれませんが、イメージリポジトリと同様に、関連するイメージの単一の仮想ビューが提示されます。

You can configure [Builds](#) and [Deployments](#) to watch an image stream for notifications when new images are added and react by performing a Build or Deployment, respectively.

たとえば、デプロイメントで特定のイメージを使用しており、そのイメージの新規バージョンを作成する場合に、対象のイメージの新しいバージョンが選択されるように、デプロイメントを自動的に実行することができます。

However, if the image stream tag used by the Deployment or Build is not updated, then even if the Docker image in the Docker registry is updated, the Build or Deployment will continue using the previous (presumably known good) image.

ソースイメージは以下のいずれかに保存できます。

- OpenShift Container Platform の [統合レジストリー](#)
- An external registry, for example **registry.access.redhat.com** or **hub.docker.com**
- OpenShift Container Platform クラスターの他のイメージストリーム

When you define an object that references an image stream tag (such as a Build configuration or Deployment configuration), you point to an image stream tag, not the Docker repository. When you Build or Deploy your application, OpenShift Container Platform queries the Docker repository using the image stream tag to locate the associated ID of the image and uses that exact image.

イメージストリームメタデータは他のクラスター情報と共に etcd インスタンスに保存されます。

The following image stream contains two tags: **34** which points to a Python v3.4 image and **35** which points to a Python v3.5 image:

```
oc describe is python
Name: python
Namespace: imagestream
Created: 25 hours ago
Labels: app=python
Annotations: openshift.io/generated-by=OpenShiftWebConsole
  openshift.io/image.dockerRepositoryCheck=2017-10-03T19:48:00Z
Docker Pull Spec: docker-registry.default.svc:5000/imagestream/python
Image Lookup: local=false
Unique Images: 2
Tags: 2

34
  tagged from centos/python-34-centos7

  * centos/python-34-
centos7@sha256:28178e2352d31f240de1af1370be855db33ae9782de737bb005247d8791a54d0
  14 seconds ago

35
  tagged from centos/python-35-centos7

  * centos/python-35-
centos7@sha256:2efb79ca3ac9c9145a63675fb0c09220ab3b8d4005d35e0644417ee552548b10
  7 seconds ago
```

イメージストリームの使用には、いくつかの大きな利点があります。

- コマンドラインを使用して再プッシュすることなく、タグ付けや、タグのロールバック、およびイメージの迅速な処理を実行できます。
- 新規イメージがレジストリーにプッシュされると、ビルドおよびデプロイメントをトリガーできます。また、OpenShift Container Platform には他のリソースの汎用トリガーがあります (Kubernetes オブジェクトなど)。
- You can [mark a tag for periodic re-import](#) . If the source image has changed, that change is picked up and reflected in the image stream, which triggers the Build and/or Deployment flow, depending upon the Build or Deployment configuration.
- 詳細なアクセス制御を使用してイメージを共有し、チーム間でイメージを迅速に分散できます。
- ソースイメージが変更されると、イメージストリームタグはイメージの既知の適切なバージョンをポイントしたままになり、アプリケーションが予期せずに損傷しないようにします。
- You can configure security around who can view and use the images through permissions on the image stream objects.
- クラスターレベルでイメージを読み込んだり、一覧表示するパーミッションのないユーザーは、イメージストリームを使用してプロジェクトでタグ付けされたイメージを取得できます。

イメージストリームのキュレートされたセットについては、[OpenShift Image Streams and Templates library](#) を参照してください。

イメージストリームの使用時に、イメージストリームタグのポイント先およびタグおよびイメージへの変更の影響について把握しておくことは重要デス。以下は例になります。

- If your image stream tag points to a Docker image tag, you need to understand how that Docker image tag is updated. For example, a Docker image tag **docker.io/ruby:2.4** will probably always point to a v2.4 ruby image. But, a Docker image tag **docker.io/ruby:latest** will probably change with major versions. So, the Docker image tag that a image stream tag points to can tell you how stable the image stream tag will be, if you choose to reference it.
- If your image stream tag follows another image stream tag (it does not point directly to a docker image tag), it is possible that the image stream tag will be updated to follow a different image stream tag in the future. Again, this could result in picking up an incompatible version change.

3.5.2.1. 重要な用語

Docker リポジトリ

A collection of related docker images and tags identifying them. For example, the OpenShift Jenkins images are in a Docker repository:

```
docker.io/openshift/jenkins-2-centos7
```

Docker registry

Docker リポジトリからイメージを保存し、提供できるコンテンツサーバーです。以下は例になります。

```
registry.access.redhat.com
```

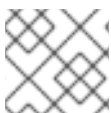
Docker image

コンテナとして実行できる特定のコンテナセットです。通常は Docker リポジトリ内の特定のタグに関連付けられます。

Docker image tag

A label applied to a Docker image in a repository that distinguishes a specific image. For example, here **3.6.0** is a tag:

```
docker.io/openshift/jenkins-2-centos7:3.6.0
```



注記

A Docker image tag can be updated to point to new Docker image content at any time.

Docker image ID

イメージをプルするために使用できる SHA (セキュアハッシュアルゴリズム) コードです。以下は例になります。

```
docker.io/openshift/jenkins-2-centos7@sha256:ab312bda324
```

**注記**

A SHA image ID cannot change. A specific SHA identifier always references the exact same docker image content.

イメージストリーム

タグで識別される任意の数の Docker 形式のコンテナイメージへのポインターが含まれる OpenShift Container Platform オブジェクトです。イメージストリームを Docker リポジトリと同等のものとしてみなすことができます。

イメージストリームタグ

A named pointer to an image in an image stream. An image stream tag is similar to a Docker image tag. See [Image Stream Tag](#) below.

イメージストリームイメージ

An image that allows you to retrieve a specific Docker image from a particular image stream where it is tagged. An image stream image is an API resource object that pulls together some metadata about a particular image SHA identifier. See [Image Stream Images](#) below.

イメージストリームトリガー

イメージストリームタグの変更時に特定のアクションを生じさせるトリガーです。たとえば、インポートにより、タグの値が変更され、これにより Deployments、Builds またはそれらをリッスンする他のリソースがある場合にトリガーが実行されます。以下の「[イメージストリームトリガー](#)」を参照してください。

3.5.2.2. イメージストリームの設定

イメージストリームオブジェクトには以下の要素が含まれます。

**注記**

See the [Developer Guide](#) for details on managing images and image streams.

イメージストリームオブジェクト定義

```
apiVersion: v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  creationTimestamp: 2017-09-29T13:33:49Z
  generation: 1
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample 1
  namespace: test
  resourceVersion: "633"
  selflink: /oapi/v1/namespaces/test/imagestreams/origin-ruby-sample
  uid: ee2b9405-c68c-11e5-8a99-525400f25e34
spec: {}
status:
  dockerImageRepository: 172.30.56.218:5000/test/origin-ruby-sample 2
  tags:
  - items:
```

```

- created: 2017-09-02T10:15:09Z
  dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d ❸
  generation: 2
  image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5 ❹
- created: 2017-09-29T13:40:11Z
  dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
  generation: 1
  image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
tag: latest ❺

```

- ❶ イメージストリームの名前です。
- ❷ 新規イメージをこのイメージストリームで追加/更新するためにプッシュできる Docker リポジトリパスです。
- ❸ イメージストリームが現在参照する SHA ID です。このイメージストリームタグを参照するリソースはこの ID を使用します。
- ❹ このイメージストリームタグが以前に参照した SHA ID です。古いイメージにロールバックするために使用できます。
- ❺ イメージストリームタグ名です。

For a sample build configuration that references an image stream, see [What Is a BuildConfig?](#) in the **Strategy** stanza of the configuration.

For a sample deployment configuration that references an image stream, see [Creating a Deployment Configuration](#) in the **Strategy** stanza of the configuration.

3.5.2.3. イメージストリームイメージ

イメージストリームイメージは、イメージストリームから特定のイメージ ID をポイントします。

イメージストリームイメージにより、タグ付けされている特定のイメージストリームからイメージについてのメタデータを取得できます。

イメージストリームイメージオブジェクトは、イメージをイメージストリームにインポートしたり、タグ付けしたりする場合には OpenShift Container Platform に常に自動的に作成されます。イメージストリームを作成するために使用するイメージストリームイメージオブジェクトをイメージストリーム定義に明示的に定義する必要はありません。

イメージストリームイメージはリポジトリからのイメージストリーム名およびイメージ ID で構成されており、@ 記号で区切られています。

```
<image-stream-name>@<image-id>
```

上記のイメージストリームオブジェクトサンプルのイメージを参照するには、イメージストリームイメージは以下ようになります。

```
origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

3.5.2.4. イメージストリームタグ

イメージストリームタグは、イメージストリームのイメージに対する名前付きポインターです。これは `istag` として省略されることが多くあります。イメージストリームタグは、指定のイメージストリームおよびタグのイメージを参照するか、または取得するために使用されます。

イメージストリームタグは、ローカル、または外部で管理されるイメージを参照できます。これには、タグがポイントしたすべてのイメージのスタックとして参照されるイメージの履歴が含まれます。新規または既存のイメージがた億艇のイメージストリームタグでタグ付けされる場合はいつでも、これは履歴スタックの最初の位置に置かれます。これまで先頭の位置を占めていたイメージは2番目の位置などに置かれます。これにより、タグを過去のイメージに再びポイントさせるよう簡単にロールバックできます。

以下のイメージストリームタグは、[上記のイメージストリームオブジェクトのサンプル](#)からのもので

履歴の2つのイメージを持つイメージストリームタグ

```
tags:
- items:
  - created: 2017-09-02T10:15:09Z
    dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
  generation: 2
  image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
  - created: 2017-09-29T13:40:11Z
    dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
  generation: 1
  image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
tag: latest
```

イメージストリームタグは `permanent` タグまたは `tracking` タグにすることができます。

- **永続タグ** は、Python 3.5 などの特定バージョンのイメージをポイントするバージョン固有のタグです。
- **トラッキングタグ** は別のイメージストリームタグに従う参照タグで、シンボリックリンクなどのように、フォローするイメージを変更するために今後更新される可能性があります。このような新規レベルでは後方互換性が確保されない点に注意してください。

For example, the **latest** image stream tags that ship with OpenShift Container Platform are tracking tags. This means consumers of the **latest** image stream tag will be updated to the newest level of the framework provided by the image when a new level becomes available. A **latest** image stream tag to **v3.6** could be changed to **v3.7** at any time. It is important to be aware that these **latest** image stream tags behave differently than the Docker **latest** tag. The **latest** image stream tag, in this case, does not point to the latest image in the Docker repository. It points to another image stream tag, which might not be the latest version of an image. For example, if the **latest** image stream tag points to **v3.2** of an image, when the **3.3** version is released, the **latest** tag is not automatically updated to **v3.3**, and remains at **v3.2** until it is manually updated to point to a **v3.3** image stream tag.



注記

トラッキングタグは単一のイメージストリームに制限され、他のイメージストリームを参照できません。

You can create your own image stream tags for your own needs. See the [Recommended Tagging Conventions](#).

イメージストリームタグは、コロンで区切られた、イメージストリームの名前とタグで構成されています。

```
<image stream name>:<tag>
```

たとえば、[上記のイメージストリームオブジェクトのサンプルで sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d](#) イメージを参照するには、イメージストリームタグは以下のようになります。

```
origin-ruby-sample:latest
```

3.5.2.5. イメージストリーム変更トリガー

イメージストリームトリガーにより、ビルドおよびデプロイメントは、アップストリームの新規バージョンが利用可能になると自動的に起動します。

たとえば、ビルドおよびデプロイメントは、イメージストリームタグの変更時に自動的に起動します。これは、特定のイメージストリームタグをモニターし、変更の検出時にビルドまたはデプロイメントに通知することで実行されます。

ImageChange トリガーにより、[イメージストリームタグ](#)の変更時に常に新規レプリケーションコントローラーが生成されます (イメージの新規バージョンがプッシュされるタイミング)。

例3.8 ImageChange トリガー

```
triggers:
- type: "ImageChange"
  imageChangeParams:
    automatic: true ①
    from:
      kind: "ImageStreamTag"
      name: "origin-ruby-sample:latest"
      namespace: "myproject"
    containerNames:
      - "helloworld"
```

① **imageChangeParams.automatic** フィールドが **false** に設定されると、トリガーが無効になります。

上記の例では、**origin-ruby-sample** イメージストリームの **latest** タグの値が変更され、新しいイメージの値がデプロイメント設定の **helloworld** コンテナに指定されている現在のイメージと異なる場合に、**helloworld** コンテナの新規イメージを使用して、新しいレプリケーションコントローラーが作成されます。



注記

ImageChange トリガーがデプロイメント設定 (**ConfigChange** トリガーと **automatic=false**、または **automatic=true**) で定義されていて、**ImageChange** トリガーで参照されている **ImageStreamTag** がまだ存在していない場合には、ビルドにより、イメージが、**ImageStreamTag** にインポートまたはプッシュされた直後に初回のデプロイメントプロセスが自動的に開始されます。

3.5.2.6. イメージストリームのマッピング

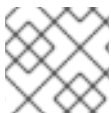
統合レジストリーが新規イメージを受信する際、これは OpenShift Container Platform にマップするイメージストリームを作成し、送信し、イメージのプロジェクト、名前、タグおよびイメージメタデータを提供します。



注記

イメージストリームのマッピングの設定は高度な機能です。

この情報は、新規イメージを作成する際 (すでに存在しない場合) やイメージをイメージストリームにタグ付けする際に使用されます。OpenShift Container Platform は、コマンド、エントリーポイント、および開発変数などの各イメージについての完全なメタデータを保存します。OpenShift Container Platform のイメージはイミュータブルであり、名前の最大長さは 63 文字です。



注記

See the [Developer Guide](#) for details on manually tagging images.

以下のイメージストリームマッピングのサンプルにより、イメージが **test/origin-ruby-sample:latest** としてタグ付けされます。

イメージストリームマッピングオブジェクト定義

```
apiVersion: v1
kind: ImageStreamMapping
metadata:
  creationTimestamp: null
  name: origin-ruby-sample
  namespace: test
tag: latest
image:
  dockerImageLayers:
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ee1dd2cb6df21971f4af6de0f1d7782b81fb63156801cfde2bb47b4247c23c29
    size: 196634330
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ca062656bff07f18bff46be00f40cfbb069687ec124ac0aa038fd676cfaea092
    size: 177723024
  - name: sha256:63d529c59c92843c395befd065de516ee9ed4995549f8218eac6ff088bfa6b6e
    size: 55679776
  - name: sha256:92114219a04977b5563d7dff71ec4caa3a37a15b266ce42ee8f43dba9798c966
```

```

size: 11939149
dockerImageMetadata:
  Architecture: amd64
  Config:
    Cmd:
      - /usr/libexec/s2i/run
    Entrypoint:
      - container-entrypoint
    Env:
      - RACK_ENV=production
      - OPENSIFT_BUILD_NAMESPACE=test
      - OPENSIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
      - OPENSIFT_BUILD_NAME=ruby-sample-build-1
      - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
      - RUBY_VERSION=2.2
    ExposedPorts:
      8080/tcp: {}
    Labels:
      build-date: 2015-12-23
      io.k8s.description: Platform for building and running Ruby 2.2 applications
      io.k8s.display-name: 172.30.56.218:5000/test/origin-ruby-sample:latest
      io.openshift.build.commit.author: Ben Parees <bparees@users.noreply.github.com>
      io.openshift.build.commit.date: Wed Jan 20 10:14:27 2016 -0500
      io.openshift.build.commit.id: 00cadc392d39d5ef9117cbc8a31db0889eedd442
      io.openshift.build.commit.message: 'Merge pull request #51 from php-coder/fix_url_and_sti'
      io.openshift.build.commit.ref: master
      io.openshift.build.image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
      io.openshift.build.source-location: https://github.com/openshift/ruby-hello-world.git
      io.openshift.builder-base-version: 8d95148
      io.openshift.builder-version: 8847438ba06307f86ac877465eadc835201241df
      io.openshift.s2i.scripts-url: image:///usr/libexec/s2i
      io.openshift.tags: builder,ruby,ruby22
      io.s2i.scripts-url: image:///usr/libexec/s2i
      license: GPLv2
      name: CentOS Base Image
      vendor: CentOS
      User: "1001"
      WorkingDir: /opt/app-root/src
    Container: 86e9a4a3c760271671ab913616c51c9f3cea846ca524bf07c04a6f6c9e103a76
    ContainerConfig:
      AttachStdout: true
      Cmd:
        - /bin/sh
        - -c
        - tar -C /tmp -xf - && /usr/libexec/s2i/assemble
      Entrypoint:
        - container-entrypoint

```

```

Env:
- RACK_ENV=production
- OPENSIFT_BUILD_NAME=ruby-sample-build-1
- OPENSIFT_BUILD_NAMESPAC=test
- OPENSIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
- EXAMPLE=sample-app
- PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
- STI_SCRIPTS_URL=image:///usr/libexec/s2i
- STI_SCRIPTS_PATH=/usr/libexec/s2i
- HOME=/opt/app-root/src
- BASH_ENV=/opt/app-root/etc/scl_enable
- ENV=/opt/app-root/etc/scl_enable
- PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
- RUBY_VERSION=2.2
ExposedPorts:
  8080/tcp: {}
Hostname: ruby-sample-build-1-build
Image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
  OpenStdin: true
  StdinOnce: true
  User: "1001"
  WorkingDir: /opt/app-root/src
Created: 2016-01-29T13:40:00Z
DockerVersion: 1.8.2.fc21
Id: 9d7fd5e2d15495802028c569d544329f4286dcd1c9c085ff5699218dbaa69b43
Parent: 57b08d979c86f4500dc8cad639c9518744c8dd39447c055a3517dc9c18d6fccd
Size: 441976279
apiVersion: "1.0"
kind: DockerImage
dockerImageMetadataVersion: "1.0"
dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d

```

3.5.2.7. イメージストリームの使用

The following sections describe how to use image streams and image stream tags. For more information on working with image streams, see [Managing Images](#).

3.5.2.7.1. イメージストリームについての情報の取得

To get general information about the image stream and detailed information about all the tags it is pointing to, use the following command:

```
oc describe is/<image-name>
```

例:

```

oc describe is/python

Name: python
Namespace: default
Created: About a minute ago
Labels: <none>

```

```
Annotations: openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup: local=false
Unique Images: 1
Tags: 1
```

3.5

tagged from centos/python-35-centos7

```
* centos/python-35-centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
  About a minute ago
```

特定のイメージストリームタグについて利用可能な情報をすべて取得するには、以下を実行します。

```
oc describe istag/<image-stream>:<tag-name>
```

例:

```
oc describe istag/python:latest
```

```
Image Name: sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Docker Image: centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Name: sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Created: 2 minutes ago
Image Size: 251.2 MB (first layer 2.898 MB, last binary layer 72.26 MB)
Image Created: 2 weeks ago
Author: <none>
Arch: amd64
Entrypoint: container-entrypoint
Command: /bin/sh -c $STI_SCRIPTS_PATH/usage
Working Dir: /opt/app-root/src
User: 1001
Exposes Ports: 8080/tcp
Docker Labels: build-date=20170801
```



注記

表示されている以上の情報が出力されます。

3.5.2.7.2. 追加タグのイメージストリームへの追加

既存タグのいずれかをポイントするタグを追加するには、**oc tag** コマンドを使用できます。

```
oc tag <image-name:tag> <image-name:tag>
```

例:

```
oc tag python:3.5 python:latest
```

```
Tag python:latest set to
python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25.
```

Use the **oc describe** command to confirm the image stream has two tags, one (**3.5**) pointing at the external Docker image and another tag (**latest**) pointing to the same image because it was created based on the first tag.

```
oc describe is/python
```

```
Name: python
```

```
Namespace: default
```

```
Created: 5 minutes ago
```

```
Labels: <none>
```

```
Annotations: openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
```

```
Docker Pull Spec: docker-registry.default.svc:5000/default/python
```

```
Image Lookup: local=false
```

```
Unique Images: 1
```

```
Tags: 2
```

```
latest
```

```
tagged from python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
```

```
* centos/python-35-centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
  About a minute ago
```

```
3.5
```

```
tagged from centos/python-35-centos7
```

```
* centos/python-35-centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
  5 minutes ago
```

3.5.2.7.3. 外部イメージのタグの追加

内部または外部イメージをポイントする追加タグなど、タグ関連のすべての操作に **oc tag** コマンドを使用します。

```
oc tag <repository/image> <image-name:tag>
```

たとえば、このコマンドは **docker.io/python:3.6.0** イメージを **python** イメージストリームの **3.6** タグにマップします。

```
oc tag docker.io/python:3.6.0 python:3.6
Tag python:3.6 set to docker.io/python:3.6.0.
```

If the external image is secured, you will need to create a secret with credentials for accessing that registry. See [Importing Images from Private Registries](#) for more details.

3.5.2.7.4. イメージストリームタグの更新

別のタグをイメージストリームに反映するようタグを更新するには、以下を実行します。

```
oc tag <image-name:tag> <image-name:latest>
```

たとえば、以下は **latest** タグを更新し、**3.6** タグをイメージタグに反映させます。

```
oc tag python:3.6 python:latest
Tag python:latest set to
python@sha256:438208801c4806548460b27bd1fbc7bb188273d13871ab43f.
```

3.5.2.7.5. イメージストリームタグのイメージストリームからの削除

古いタグをイメージストリームから削除するには、以下を実行します。

```
oc tag -d <image-name:tag>
```

例:

```
oc tag -d python:3.5

Deleted tag default/python:3.5.
```

3.5.2.7.6. タグの定期的なインポートの設定

When working with an external Docker registry, to periodically re-import an image (such as, to get latest security updates), use the **--scheduled** flag:

```
oc tag <repository/image> <image-name:tag> --scheduled
```

例:

```
oc tag docker.io/python:3.6.0 python:3.6 --scheduled

Tag python:3.6 set to import docker.io/python:3.6.0 periodically.
```

このコマンドにより、OpenShift Container Platform はこの特定のイメージストリームタグを定期的に更新します。この期間はデフォルトではクラスター全体で15分に設定されます。

定期的なチェックを削除するには、上記のコマンド再実行しますが、**--scheduled** フラグを省略します。これにより、その動作がデフォルトに再設定されます。

```
oc tag <repository/image> <image-name:tag>
```

3.6. デプロイメント

3.6.1. レプリケーションコントローラー

レプリケーションコントローラーは、指定した Pod のレプリカ数が常に実行されていることを確認します。Pod の終了または削除が行われた場合に、レプリケーションコントローラーが機能し、定義した数になるまでインスタンス化する数を増やします。希望の数よりも実行数が多い場合には、定義数に合わせて、必要な数だけ削除します。

レプリケーションコントローラー設定は以下で構成されています。

1. 必要なレプリカ数 (これはランタイム時に調整可能)。
2. レプリケートされた Pod の作成時に使用する Pod 定義。

3. 管理された Pod を識別するためのセレクター。

セレクターは、レプリケーションコントローラーが管理する Pod に割り当てられるラベルセットです。これらのラベルは、Pod 定義に組み込まれ、レプリケーションコントローラーがインスタンス化します。レプリケーションコントローラーは、必要に応じて調節するために、セレクターを使用して、すでに実行中の Pod 数を判断します。

レプリケーションコントローラーは、追跡もしませんが、負荷またはトラフィックに基づいて自動スケールリングを実行することはありません。この場合、そのレプリカ数が外部の自動スケーラーで調整される必要があります。

レプリケーションコントローラーは、**ReplicationController** というコアの Kubernetes オブジェクトです。

以下は、**ReplicationController** 定義のサンプルです。

```
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
```

- ① 実行する Pod のコピー数です。
- ② 実行する Pod のラベルセレクターです。
- ③ コントローラーが作成する Pod のテンプレートです。
- ④ Pod のラベルにはラベルセレクターからのものが含まれている必要があります。
- ⑤ パラメーターの拡張後の名前の最大長さは 63 文字です。

3.6.2. ジョブ

ジョブは、その目的が特定の理由のために Pod を作成することである点でレプリケーションコントローラーと似ています。違いは、レプリケーションコントローラーの場合は、継続的に実行されている Pod を対象としていますが、ジョブは 1 回限りの Pod を対象としています。ジョブは正常な完了を追跡し、指定された完了数に達すると、ジョブ自体が完了します。

以下の例は、 π (Pi) を 2000 桁計算し、これを出力してから完了します。

```
apiVersion: extensions/v1
kind: Job
metadata:
  name: pi
spec:
  selector:
    matchLabels:
      app: pi
  template:
    metadata:
      name: pi
    labels:
      app: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
        restartPolicy: Never
```

See the [Jobs](#) topic for more information on how to use jobs.

3.6.3. デプロイメントおよびデプロイメント設定

レプリケーションコントローラーでビルドする OpenShift Container Platform はデプロイメントの概念を使用したソフトウェアの開発およびデプロイメントライフサイクルの拡張サポートを追加します。最も単純な場合に、デプロイメントは新規アプリケーションコントローラーのみを作成し、それに Pod を起動させます。ただし、OpenShift Container Platform デプロイメントは、イメージの既存デプロイメントから新規デプロイメントに移行する機能を提供し、レプリケーションコントローラーの作成前後に実行するフックも定義します。

OpenShift Container Platform **DeploymentConfig** オブジェクトはデプロイメントの以下の詳細を定義します。

1. **ReplicationController** 定義の要素。
2. 新規デプロイメントの自動作成のトリガー。
3. デプロイメント間の移行ストラテジー。
4. ライフサイクルフック。

デプロイヤー Pod は、デプロイメントがトリガーされるたびに、手動または自動であるかを問わず、(古いレプリケーションコントローラーの縮小、新規レプリケーションコントローラーの拡大およびフックの実行などの) デプロイメントを管理します。デプロイメント Pod は、デプロイメントのログを維持するためにデプロイメントの完了後は無期限で保持されます。デプロイメントが別のものに置き換えられる場合、以前のレプリケーションコントローラーは必要に応じて簡単なロールバックを有効にできるように保持されます。

For detailed instructions on how to create and interact with deployments, refer to [Deployments](#).

以下は、いくつかの省略およびコールアウトを含む **DeploymentConfig** 定義のサンプルです。

```
apiVersion: v1
```

```
kind: DeploymentConfig
metadata:
  name: frontend
spec:
  replicas: 5
  selector:
    name: frontend
  template: { ... }
  triggers:
  - type: ConfigChange ①
  - imageChangeParams:
      automatic: true
      containerNames:
      - helloworld
      from:
        kind: ImageStreamTag
        name: hello-openshift:latest
      type: ImageChange ②
  strategy:
    type: Rolling ③
```

- ① **ConfigChange** トリガーにより、新規デプロイメントが、レプリケーションコントローラーテンプレートが変更すると常に作成されます。
- ② **ImageChange** トリガーにより、新規デプロイメントが、バックインイメージの最新バージョンが名前付きイメージストリームで利用可能になる際には常に作成されます。
- ③ デフォルトのローリングストラテジーにより、デプロイメント間のダウンタイムなしの移行が行われます。

3.7. TEMPLATES (テンプレート)

3.7.1. 概要

A template describes a set of [objects](#) that can be parameterized and processed to produce a list of objects for creation by OpenShift Container Platform. The objects to create can include anything that users have permission to create within a project, for example [services](#), [build configurations](#), and [deployment configurations](#). A template may also define a set of [labels](#) to apply to every object defined in the template.

See the [template guide](#) for details about creating and using templates.

第4章 追加の概念

4.1. 認証

4.1.1. 概要

認証層は、OpenShift Container Platform API への要求に関連付けられたユーザーを識別します。次に、認証層は要求が許可されるかどうかを判別するために要求側のユーザーについての情報を使用します。

As an administrator, you can [configure authentication](#) using a [master configuration file](#).

4.1.2. ユーザーとグループ

OpenShift Container Platform の **ユーザー** は、OpenShift Container Platform API に要求できるエンティティです。通常、これは OpenShift Container Platform と対話している開発者または管理者のアカウントを表します。

A user can be assigned to one or more **groups**, each of which represent a certain set of users. Groups are useful when [managing authorization policies](#) to grant permissions to multiple users at once, for example allowing access to [objects](#) within a [project](#), versus granting them to users individually.

In addition to explicitly defined groups, there are also system groups, or **virtual groups**, that are automatically provisioned by OpenShift. These can be seen when [viewing cluster bindings](#).

仮想グループのデフォルトセットでは、とくに以下の点に留意してください。

仮想グループ	説明
<code>system:authenticated</code>	認証されたユーザーに自動的に関連付けられます。
<code>system:authenticated:oauth</code>	OAuth アクセストークンで認証されたすべてのユーザーに自動的に関連付けられます。
<code>system:unauthenticated</code>	認証されていないすべてのユーザーに自動的に関連付けられます。

4.1.3. API 認証

OpenShift Container Platform API への要求は以下の方法で認証されます。

OAuth アクセストークン

- `<master>/oauth/authorize` および `<master>/oauth/token` エンドポイントを使用して OpenShift Container Platform OAuth サーバーから取得されます。
- Sent as an **Authorization: Bearer...** header
- Sent as an **access_token=...** query parameter for websocket requests prior to OpenShift Container Platform server version 3.6.

- Sent as a websocket subprotocol header in the form **base64url.bearer.authorization.k8s.io.<base64url-encoded-token>** for websocket requests in OpenShift Container Platform server version 3.6 and later.

X.509 クライアント証明書

- API サーバーへの HTTPS 接続を要求します。
- 信頼される認証局バンドルに対して API サーバーによって検証されます。
- API サーバーは証明書を作成し、これをコントローラーに配布してそれらを認証できるようにします。

無効なアクセストークンまたは無効な証明書を使用した要求は認証層によって拒否され、401 エラーが出されます。

アクセストークンまたは証明証が提供されない場合、認証層は **system:anonymous** 仮想ユーザーおよび **system:unauthenticated** 仮想グループを要求に割り当てます。これにより、認証層は匿名ユーザーが実行できる要求 (ある場合) を決定できます。

4.1.3.1. 権限の借用

OpenShift Container Platform API への要求いは、要求側が要求を指定されたユーザーからのものであるかのように処理されることを希望することを示す、**Impersonate-User** ヘッダーが含まれる場合があります。このユーザーのなりすましは、**--as=<user>** フラグを要求に追加して実行できます。

ユーザー A がユーザー B の権限を借用できるのは、ユーザー A が認証されてからです。ユーザー A がユーザー B という名前のユーザーの権限を借用できるように、認証チェックが行われます。ユーザー A が、サービスアカウント **system:serviceaccount:namespace:name** の権限借用を要求する場合には、OpenShift Container Platform は、ユーザー A が **namespace** の **name** という名前の **serviceaccount** の権限を借用できることを確認します。チェックに失敗すると、この要求は 403 (Forbidden) エラーコードで失敗します。

デフォルトで、プロジェクト管理者およびエディターは、その namespace に含まれるサービスアカウントの権限を借用できます。ユーザーは、**sudoer** ロールを使用して、**system:admin** の権限を借用できるので、クラスター管理者のパーミッションが使えるようになります。**system:admin** の権限を借用することで、誤植の発生を防ぐことはできませんが、クラスターの管理者に対してセキュリティーを確保するわけではありません。たとえば、**oc delete nodes --all** を実行すると失敗するにもかかわらず、**oc delete nodes --all --as=system:admin** を実行すると成功します。以下のコマンドを実行してユーザーにこのパーミッションを付与できます。

```
$ oc create clusterrolebinding <any_valid_name> --clusterrole=sudoer --user=<username>
```

If you need to create a project request on behalf of a user, include the **--as=<user> --as-group=<group1> --as-group=<group2>** flags in your command. Because **system:authenticated:oauth** is the only bootstrap group that can create project requests, you must impersonate that group, as shown in the following example:

```
$ oc new-project <project> --as=<user> \
--as-group=system:authenticated --as-group=system:authenticated:oauth
```

4.1.4. OAuth

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。ユーザーは OAuth アクセストークンを取得して、API に対して認証を実行します。

When a person requests a new OAuth token, the OAuth server uses the configured [identity provider](#) to determine the identity of the person making the request.

次に、そのアイデンティティーがマップするユーザーを判別し、そのユーザーのアクセスユーザーを作成し、使用できるようにトークンを返します。

4.1.4.1. OAuth クライアント

OAuth トークンのすべての要求は、トークンを受信し、使用する OAuth クライアントを指定する必要があります。以下の OAuth クライアントは、OpenShift Container Platform API の起動時に自動的に作成されます。

OAuth クライアント	使用法
openshift-web-console	Web コンソールのトークンを要求します。
openshift-browser-client	対話式ログインを処理できるユーザーエージェントで <code><master>/oauth/token/request</code> でトークンを要求します。
openshift-challenging-client	WWW-Authenticate チャレンジを処理できるユーザーエージェントでトークンを要求します。

追加のクライアントを登録するには、以下を実行します。

```
$ oc create -f <(echo '
kind: OAuthClient
apiVersion: oauth.openshift.io/v1
metadata:
  name: demo ①
  secret: "... " ②
  redirectURIs:
  - "http://www.example.com/" ③
grantMethod: prompt ④
')
```

- ① `<master>/oauth/authorize` および `<master>/oauth/token` への要求を実行する際には、OAuth クライアントの **name** が **client_id** パラメーターとして使用されます。
- ② `<master>/oauth/token` への要求の実行時に、**secret** は **client_secret** パラメーターとして使用されます。
- ③ `<master>/oauth/authorize` および `<master>/oauth/token` への要求で指定される **redirect_uri** パラメーターは、**redirectURIs** のいずれかに等しい (またはこれによってプレフィックスが付けられた) 状態でなければなりません。
- ④ **grantMethod** は、このクライアントがトークンを要求する際に実行するアクションを決定しますが、ユーザーによるアクセスは付与されていません。「Grant Options」に表示されるものと同じ値を使用します。

4.1.4.2. OAuth クライアントとしてのサービスアカウント

サービスアカウントは、OAuth クライアントの制限されたフォームで使用できます。サービスアカウントは一部の基本ユーザー情報へのアクセスを許可するスコープのサブセットと、サービスアカウント自体の namespace 内のロールベースの権限のみを要求できます。

- **user:info**
- **user:check-access**
- **role:<any_role>:<serviceaccount_namespace>**
- **role:<any_role>:<serviceaccount_namespace>:!**

サービスアカウントを OAuth クライアントとして使用する場合:

- **client_id** は **system:serviceaccount:<serviceaccount_namespace>:<serviceaccount_name>** になります。
- **client_secret** には、サービスアカウントの API トークンのいずれかを指定できます。以下は例になります。

```
$ oc sa get-token <serviceaccount_name>
```

- **WWW-Authenticate** チャレンジを取得するには、サービスアカウントの **serviceaccounts.openshift.io/oauth-want-challenges** アノテーションを **true** に設定します。
- **redirect_uri** はサービスアカウントのアノテーションに一致する必要があります。詳細は、「[OAuth クライアントとしてのサービスアカウントの URI のリダイレクト](#)」を参照してください。

4.1.4.3. OAuth クライアントとしてのサービスアカウントの URI のリダイレクト

アノテーションキーには、以下のようにプレフィックス **serviceaccounts.openshift.io/oauth-redirecturi**。または **serviceaccounts.openshift.io/oauth-redirectreference**。が含まれる必要があります。

```
serviceaccounts.openshift.io/oauth-redirecturi.<name>
```

最も単純なフォームでは、アノテーションは有効なリダイレクト URI を直接指定するために使用できません。以下は例になります。

```
"serviceaccounts.openshift.io/oauth-redirecturi.first": "https://example.com"
"serviceaccounts.openshift.io/oauth-redirecturi.second": "https://other.com"
```

上記の例の **first** および **second** ポストフィックスは 2 つの有効なリダイレクト URI を分離するために使用されます。

さらに複雑な設定では、静的なリダイレクト URI のみでは不十分な場合があります。たとえば、ルートすべての ingress が有効とみなされる必要があるかもしれません。この場合、**serviceaccounts.openshift.io/oauth-redirectreference**。プレフィックスを使用した動的なリダイレクト URI を使用できます。

例:

```
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{"kind":"OAuthRedirectReference","apiVersion":"v1","reference":
{"kind":"Route","name":"jenkins\"}}"
```

このアノテーションの値にはシリアライズされた JSON データが含まれるため、これを拡張フォーマットで表示するとより容易になります。

```
{
  "kind": "OAuthRedirectReference",
  "apiVersion": "v1",
  "reference": {
    "kind": "Route",
    "name": "jenkins"
  }
}
```

ここでは、**OAuthRedirectReference** により **jenkins** という名前のルートを参照できます。そのため、そのルートのすべての ingress は有効とみなされます。**OAuthRedirectReference** の詳細な仕様は以下のようにになります。

```
{
  "kind": "OAuthRedirectReference",
  "apiVersion": "v1",
  "reference": {
    "kind": ..., ①
    "name": ..., ②
    "group": ... ③
  }
}
```

- ① **kind** は参照されているオブジェクトのタイプを参照します。現時点では、**route** のみがサポートされています。
- ② **name** はオブジェクトの名前を参照します。このオブジェクトはサービスアカウントと同じ namespace にある必要があります。
- ③ **group** はオブジェクトのグループを参照します。ルートのグループは空の文字列であるため、これを空白のままにします。

アノテーションはどちらも、プレフィックスも組み合わせて、参照オブジェクトで提供されるデータを上書きできます。以下は例になります。

```
"serviceaccounts.openshift.io/oauth-redirecturi.first": "custompath"
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{"kind":"OAuthRedirectReference","apiVersion":"v1","reference":
{"kind":"Route","name":"jenkins\"}}"
```

first ポストフィックスはアノテーションを関連付けるために使用されます。**jenkins** ルートに <https://example.com> の ingress がある場合に、<https://example.com/custompath> が有効とみなされますが、<https://example.com> は有効とみなされません。上書きデータを部分的に指定するためのフォーマットは以下のようにになります。

タイプ	構文
スキーム	"https://"
ホスト名	"//website.com"
ポート	"//:8000"
パス	"examplepath"



注記

ホスト名の上書きを指定すると、参照されるオブジェクトのホスト名データが置き換わりますが、これは望ましい動作ではありません。

上記の構文のいずれの組み合わせも、以下のフォーマットを使って実行できます。

<scheme:>//<hostname><:port>/<path>

同じオブジェクトを複数回参照して、柔軟性を向上することができます。

```
"serviceaccounts.openshift.io/oauth-redirecturi.first": "custompath"
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{"kind":"OAuthRedirectReference","apiVersion":"v1","reference":
{"kind":"Route","name":"jenkins"}}
"serviceaccounts.openshift.io/oauth-redirecturi.second": "//:8000"
"serviceaccounts.openshift.io/oauth-redirectreference.second": "
{"kind":"OAuthRedirectReference","apiVersion":"v1","reference":
{"kind":"Route","name":"jenkins"}}"
```

jenkins という名前のルートに **https://example.com** の ingress がある場合には、**https://example.com:8000** と **https://example.com/custompath** の両方が有効とみなされます。

必要な動作を得るために、静的で動的なアノテーションを同時に使用できます。

```
"serviceaccounts.openshift.io/oauth-redirectreference.first": "
{"kind":"OAuthRedirectReference","apiVersion":"v1","reference":
{"kind":"Route","name":"jenkins"}}
"serviceaccounts.openshift.io/oauth-redirecturi.second": "https://other.com"
```

4.1.4.3.1. OAuth の API イベント

API サーバーは、API マスターログへの直接的なアクセスがないとデバッグが困難な **unexpected condition** のエラーメッセージを返すことがあります。このエラーの根本的な理由は意図的に非表示にされます。認証されていないユーザーにサーバーの状態についての情報を提供することを避けるためです。

これらのエラーのサブセットは、サービスアカウントの OAuth 設定の問題に関連するものです。これらの問題は、管理者以外のユーザーが確認できるイベントでキャプチャーされます。 **unexpected condition** というサーバーエラーが OAuth の実行時に発生する場合、 **oc get events** を実行し、これらのイベントについて **ServiceAccount** で確認します。

以下の例では、適切な OAuth リダイレクト URI がないサービスアカウントに対して警告を出しています。

```
$ oc get events | grep ServiceAccount
1m      1m      1      proxy      ServiceAccount      Warning
NoSAOAuthRedirectURIs  service-account-oauth-client-getter
system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-
redirecturi.<some-value>=<redirect> or create a dynamic URI using
serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>
```

oc describe sa/<service-account-name> を実行すると、指定のサービスアカウント名に関連付けられた OAuth イベントが報告されます。

```
$ oc describe sa/proxy | grep -A5 Events
Events:
  FirstSeen    LastSeen    Count  From                                     SubObjectPath  Type      Reason
Message
-----
3m          3m          1      service-account-oauth-client-getter      Warning
NoSAOAuthRedirectURIs  system:serviceaccount:myproject:proxy has no redirectURIs; set
serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or create a dynamic URI
using serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>
```

以下は生じる可能性のあるイベントエラーの一覧です。

リダイレクト URI アノテーションが指定されていないか、無効な URI が指定されている

```
Reason          Message
NoSAOAuthRedirectURIs  system:serviceaccount:myproject:proxy has no redirectURIs; set
serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or create a dynamic URI
using serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>
```

無効なルートが指定されている

```
Reason          Message
NoSAOAuthRedirectURIs  [routes.route.openshift.io "<name>" not found,
system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-
redirecturi.<some-value>=<redirect> or create a dynamic URI using
serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>]
```

無効な参照タイプが指定されている

```
Reason          Message
NoSAOAuthRedirectURIs  [no kind "<name>" is registered for version "v1",
system:serviceaccount:myproject:proxy has no redirectURIs; set serviceaccounts.openshift.io/oauth-
redirecturi.<some-value>=<redirect> or create a dynamic URI using
serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>]
```

SA トークンがない

```
Reason          Message
NoSAOAuthTokens    system:serviceaccount:myproject:proxy has no tokens
```

4.1.4.3.1.1. 誤設定の場合に引き起こされる API イベントのサンプル

以下の手順は、ユーザーが破損状態に入る 1 つの経緯とこの問題の解決方法を示しています。

1. サービスアカウントを OAuth クライアントとして利用するプロジェクトを作成します。
 - a. プロキシサービスアカウントオブジェクトの YAML を作成し、これがルートの **proxy** を使用することを確認します。

```
vi serviceaccount.yaml
```

以下のサンプルコードを追加します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: proxy
  annotations:
    serviceaccounts.openshift.io/oauth-redirectreference.primary:
      '{"kind": "OAuthRedirectReference", "apiVersion": "v1", "reference":
        {"kind": "Route", "name": "proxy"}}'
```

- b. プロキシへのセキュアな接続を作成するために、ルートオブジェクトの YAML を作成します。

```
vi route.yaml
```

以下のサンプルコードを追加します。

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: proxy
spec:
  to:
    name: proxy
  tls:
    termination: Reencrypt
apiVersion: v1
kind: Service
metadata:
  name: proxy
  annotations:
    service.alpha.openshift.io/serving-cert-secret-name: proxy-tls
spec:
  ports:
    - name: proxy
      port: 443
      targetPort: 8443
  selector:
    app: proxy
```

- c. プロキシをサイドカーとして起動するために、デプロイメント設定の YAML を作成します。

```
vi proxysidecar.yaml
```

以下のサンプルコードを追加します。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: proxy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: proxy
  template:
    metadata:
      labels:
        app: proxy
    spec:
      serviceAccountName: proxy
      containers:
        - name: oauth-proxy
          image: openshift/oauth-proxy:v1.0.0
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8443
              name: public
          args:
            - --https-address=:8443
            - --provider=openshift
            - --openshift-service-account=proxy
            - --upstream=http://localhost:8080
            - --tls-cert=/etc/tls/private/tls.crt
            - --tls-key=/etc/tls/private/tls.key
            - --cookie-secret=SECRET
          volumeMounts:
            - mountPath: /etc/tls/private
              name: proxy-tls

        - name: app
          image: openshift/hello-openshift:latest
      volumes:
        - name: proxy-tls
          secret:
            secretName: proxy-tls
```

d. Create the objects

```
oc create -f serviceaccount.yaml
oc create -f route.yaml
oc create -f proxysidecar.yaml
```

2. **oc edit sa/proxy** を実行してサービスアカウントを編集し、**serviceaccounts.openshift.io/oauth-redirectreference** アノテーションを、存在しないルートにポイントするように変更します。

```

apiVersion: v1
imagePullSecrets:
- name: proxy-dockercfg-08d5n
kind: ServiceAccount
metadata:
  annotations:
    serviceaccounts.openshift.io/oauth-redirectreference.primary:
'{"kind":"OAuthRedirectReference","apiVersion":"v1","reference":
{"kind":"Route","name":"notexist"}}'
...

```

3. OAuth ログでサービスを確認し、サーバーエラーを見つけます。

承認サーバーに、要求を満たすことを阻む予期しないエラーが発生しました。

4. `oc get events` を実行して **ServiceAccount** イベントを表示します。

```

oc get events | grep ServiceAccount

23m      23m      1      proxy      ServiceAccount      Warning
NoSAOAuthRedirectURIs service-account-oauth-client-getter [routes.route.openshift.io
"notexist" not found, system:serviceaccount:myproject:proxy has no redirectURIs; set
serviceaccounts.openshift.io/oauth-redirecturi.<some-value>=<redirect> or create a dynamic
URI using serviceaccounts.openshift.io/oauth-redirectreference.<some-value>=<reference>]

```

4.1.4.4. 統合

All requests for OAuth tokens involve a request to `<master>/oauth/authorize`. Most authentication integrations place an authenticating proxy in front of this endpoint, or configure OpenShift Container Platform to validate credentials against a backing [identity provider](#). Requests to `<master>/oauth/authorize` can come from user-agents that cannot display interactive login pages, such as the CLI. Therefore, OpenShift Container Platform supports authenticating using a **WWW-Authenticate** challenge in addition to interactive login flows.

認証プロキシが `<master>/oauth/authorize` エンドポイントの前に配置される場合、対話式ログインページを表示したり、対話式ログインフローにリダイレクトする代わりに、認証されていない、ブラウザ以外のユーザーエージェントの **WWW-Authenticate** チャレンジを送信します。

注記

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 **X-CSRF-Token** 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 **WWW-Authenticate** 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。

認証プロキシが **WWW-Authenticate** チャレンジをサポートしないか、または OpenShift Container Platform が **WWW-Authenticate** チャレンジをサポートしないアイデンティティプロバイダーを使用するように設定されている場合、ユーザーはブラウザで `<master>/oauth/token/request` にアクセスし、アクセストークンを手動で取得できます。

4.1.4.5. OAuth サーバーメタデータ

OpenShift Container Platform で実行されているアプリケーションは、ビルトイン OAuth サーバーにつ

いての情報を検出する必要がある場合があります。たとえば、それらは **<master>** サーバーのアドレスを手動の設定なしで検出する必要があります。これを支援するために、OpenShift Container Platform は IETF [OAuth 2.0 Authorization Server Metadata](#) のドラフト仕様を実装しています。

そのため、クラスター内で実行されているすべてのアプリケーションは、<https://openshift.default.svc/.well-known/oauth-authorization-server> に対して **GET** 要求を実行し、以下の情報を取得できます。

```
{
  "issuer": "https://<master>", ❶
  "authorization_endpoint": "https://<master>/oauth/authorize", ❷
  "token_endpoint": "https://<master>/oauth/token", ❸
  "scopes_supported": [ ❹
    "user:full",
    "user:info",
    "user:check-access",
    "user:list-scoped-projects",
    "user:list-projects"
  ],
  "response_types_supported": [ ❺
    "code",
    "token"
  ],
  "grant_types_supported": [ ❻
    "authorization_code",
    "implicit"
  ],
  "code_challenge_methods_supported": [ ❼
    "plain",
    "S256"
  ]
}
```

- ❶ **https** スキームを使用し、クエリーまたはフラグメントコンポーネントがない承認サーバーの発行者 ID です。これは、承認サーバーについての情報が含まれる **.well-known RFC 5785** リソースが公開される場所です。
- ❷ 承認サーバーの承認エンドポートの URL です。RFC 6749 を参照してください。
- ❸ 承認サーバーのトークンエンドポイントの URL です。RFC 6749 を参照してください。
- ❹ この承認サーバーがサポートする OAuth 2.0 RFC 6749 スコープの値の一覧を含む JSON 配列です。サポートされるスコープの値すべてが公開される訳ではないことに注意してください。
- ❺ この認可サーバーがサポートする OAuth 2.0 **response_type** 値の一覧を含む JSON 配列です。使用される配列の値は、RFC 7591 の OAuth 2.0 Dynamic Client Registration Protocol で定義される **response_types** パラメーターで使用されるものと同じです。
- ❻ この認可サーバーがサポートする OAuth 2.0 grant type の値の一覧が含まれる JSON 配列です。使用される配列の値は、RFC 7591 の OAuth 2.0 Dynamic Client Registration Protocol で定義される **grant_types** パラメーターで使用されるものと同じです。
- ❼ この認可サーバーでサポートされる PKCE RFC 7636 コードのチャレンジメソッドの一覧が含まれる JSON 配列です。コードのチャレンジメソッドの値は、RFC 7636 の **セクション 4.3** で定義される **code_challenge_method** パラメーターで使用されます。有効なコードのチャレンジメソッド

ドの値は、IANA PKCE Code Challenge Method レジストリーで登録される値です。「[IANA OAuth パラメーター](#)」を参照してください。

4.1.4.6. OAuth トークンの取得

OAuth サーバーは、標準的な [Authorization Code Grant \(認可コードによるグラント\)](#) および [Implicit Grant \(暗黙的グラント\)](#) の OAuth 認証フローをサポートします。

OAuth トークンを、(`openshift-challenging-client` などの) **WWW-Authenticate** チャレンジを要求するように設定された `client_id` で Implicit Grant (暗黙的グラント) フロー (`response_type=token`) を使用して要求する場合、以下が `/oauth/authorize` から送られる可能性のあるサーバー応答、およびそれらの処理方法になります。

ステータス	内容	クライアント応答
302	URL フラグメントに access_token パラメーターを含む Location ヘッダー (RFC 4.2.2)	access_token 値を OAuth トークンとして使用します。
302	error クエリーパラメーターを含む Location ヘッダー (RFC 4.1.2.1)	失敗します。オプションで error (およびオプションの error_description) クエリー値をユーザーに表示します。
302	他の Location ヘッダー	これらのルールを使用してリダイレクトに従い、結果を処理します。
401	WWW-Authenticate ヘッダーが存在する	タイプ (Basic 、 Negotiate など) が認識される場合にチャレンジに応答し、これらのルールを使用して要求を再送信し、結果を処理します。
401	WWW-Authenticate ヘッダーがない	チャレンジの承認ができません。失敗し、応答本体を表示します (これには、OAuth トークンを取得する別の方法についてのリンクまたは詳細が含まれる可能性があります)
その他	その他	失敗し、オプションでユーザーに応答本体を提示します。

4.1.4.7. Prometheus の認証メトリクス

OpenShift Container Platform は認証の試行中に以下の Prometheus システムメトリクスをキャプチャーします。

- `openshift_auth_basic_password_count` は `oc login` ユーザー名およびパスワードの試行回数をカウントします。
- `openshift_auth_basic_password_count_result` は `oc login` ユーザー名および結果 (成功またはエラー) で試行されるパスワードの回数をカウントします。

- `openshift_auth_form_password_count` は Web コンソールのログイン試行回数をカウントします。
- `openshift_auth_form_password_count_result` は結果 (成功またはエラー) による Web コンソールのログイン試行回数をカウントします。
- `openshift_auth_password_total` は `oc login` および Web コンソールのログイン試行回数をカウントします。

4.2. 承認

4.2.1. 概要

Role-based Access Control (RBAC) オブジェクトは、ユーザーがプロジェクト内の所定のアクションを実行することが許可されるかどうかを決定します。

これにより、プラットフォーム管理者は [クラスターロール](#) および [バインディング](#) を使用して、OpenShift Container Platform プラットフォーム自体およびすべてのプロジェクトへの各種のアクセスレベルを持つユーザーを制御できます。

これにより、開発者は [ローカルロール](#) および [バインディング](#) を使用し、それらのプロジェクトへのアクセスを持つユーザーを制御します。承認は [認証](#) とは異なる手順であることに注意してください。認証は、アクションを実行するユーザーのアイデンティティの判別により関連性があります。

承認は以下を使用して管理されます。

ルール	オブジェクト のセットで許可された動詞を設定します。たとえば、何かが Pod の <code>create</code> を実行できるかどうかが含まれます。
ロール	ルールのコレクションです。 ユーザー および グループ は、同時に複数の ロール に関連付けられるか、または バインド できます。
バインディング	ロール を使ったユーザーおよび/グループ間の関連付けです。

Cluster administrators can visualize rules, roles, and bindings [using the CLI](#).

たとえば、`admin` および `basic-user` の [デフォルトクラスターロール](#) のルールセットを示す以下の抜粋を考慮してみましょう。

```
$ oc describe clusterrole.rbac admin basic-user
```

```
Name: admin
Labels: <none>
Annotations: openshift.io/description=A user that has edit rights within the project and can change the project's membership.
             rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources      Non-Resource URLs Resource Names Verbs
-----
appliedclusterresourcequotas [] [] [get list watch]
appliedclusterresourcequotas.quota.openshift.io [] [] [get list watch]
```

```
bindings [] [] [get list watch]
buildconfigs [] [] [create delete deletecollection get list patch update watch]
buildconfigs.build.openshift.io [] [] [create delete deletecollection get list patch update watch]
buildconfigs/instantiate [] [] [create]
buildconfigs.build.openshift.io/instantiate [] [] [create]
buildconfigs/instantiatebinary [] [] [create]
buildconfigs.build.openshift.io/instantiatebinary [] [] [create]
buildconfigs/webhooks [] [] [create delete deletecollection get list patch update watch]
buildconfigs.build.openshift.io/webhooks [] [] [create delete deletecollection get list patch update
watch]
buildlogs [] [] [create delete deletecollection get list patch update watch]
buildlogs.build.openshift.io [] [] [create delete deletecollection get list patch update watch]
builds [] [] [create delete deletecollection get list patch update watch]
builds.build.openshift.io [] [] [create delete deletecollection get list patch update watch]
builds/clone [] [] [create]
builds.build.openshift.io/clone [] [] [create]
builds/details [] [] [update]
builds.build.openshift.io/details [] [] [update]
builds/log [] [] [get list watch]
builds.build.openshift.io/log [] [] [get list watch]
configmaps [] [] [create delete deletecollection get list patch update watch]
cronjobs.batch [] [] [create delete deletecollection get list patch update watch]
daemonsets.extensions [] [] [get list watch]
deploymentconfigrollbacks [] [] [create]
deploymentconfigrollbacks.apps.openshift.io [] [] [create]
deploymentconfigs [] [] [create delete deletecollection get list patch update watch]
deploymentconfigs.apps.openshift.io [] [] [create delete deletecollection get list patch update
watch]
deploymentconfigs/instantiate [] [] [create]
deploymentconfigs.apps.openshift.io/instantiate [] [] [create]
deploymentconfigs/log [] [] [get list watch]
deploymentconfigs.apps.openshift.io/log [] [] [get list watch]
deploymentconfigs/rollback [] [] [create]
deploymentconfigs.apps.openshift.io/rollback [] [] [create]
deploymentconfigs/scale [] [] [create delete deletecollection get list patch update watch]
deploymentconfigs.apps.openshift.io/scale [] [] [create delete deletecollection get list patch
update watch]
deploymentconfigs/status [] [] [get list watch]
deploymentconfigs.apps.openshift.io/status [] [] [get list watch]
deployments.apps [] [] [create delete deletecollection get list patch update watch]
deployments.extensions [] [] [create delete deletecollection get list patch update watch]
deployments.extensions/rollback [] [] [create delete deletecollection get list patch update watch]
deployments.apps/scale [] [] [create delete deletecollection get list patch update watch]
deployments.extensions/scale [] [] [create delete deletecollection get list patch update watch]
deployments.apps/status [] [] [create delete deletecollection get list patch update watch]
endpoints [] [] [create delete deletecollection get list patch update watch]
events [] [] [get list watch]
horizontalpodautoscalers.autoscaling [] [] [create delete deletecollection get list patch update
watch]
horizontalpodautoscalers.extensions [] [] [create delete deletecollection get list patch update
watch]
imagestreamimages [] [] [create delete deletecollection get list patch update watch]
imagestreamimages.image.openshift.io [] [] [create delete deletecollection get list patch update
watch]
imagestreamimports [] [] [create]
imagestreamimports.image.openshift.io [] [] [create]
```



```

imagestreammappings [] [] [create delete deletecollection get list patch update watch]
imagestreammappings.image.openshift.io [] [] [create delete deletecollection get list patch update
watch]
imagestreams [] [] [create delete deletecollection get list patch update watch]
imagestreams.image.openshift.io [] [] [create delete deletecollection get list patch update watch]
imagestreams/layers [] [] [get update]
imagestreams.image.openshift.io/layers [] [] [get update]
imagestreams/secrets [] [] [create delete deletecollection get list patch update watch]
imagestreams.image.openshift.io/secrets [] [] [create delete deletecollection get list patch update
watch]
imagestreams/status [] [] [get list watch]
imagestreams.image.openshift.io/status [] [] [get list watch]
imagestreamtags [] [] [create delete deletecollection get list patch update watch]
imagestreamtags.image.openshift.io [] [] [create delete deletecollection get list patch update
watch]
jenkins.build.openshift.io [] [] [admin edit view]
jobs.batch [] [] [create delete deletecollection get list patch update watch]
limitranges [] [] [get list watch]
localresourceaccessreviews [] [] [create]
localresourceaccessreviews.authorization.openshift.io [] [] [create]
localsubjectaccessreviews [] [] [create]
localsubjectaccessreviews.authorization.k8s.io [] [] [create]
localsubjectaccessreviews.authorization.openshift.io [] [] [create]
namespaces [] [] [get list watch]
namespaces/status [] [] [get list watch]
networkpolicies.extensions [] [] [create delete deletecollection get list patch update watch]
persistentvolumeclaims [] [] [create delete deletecollection get list patch update watch]
pods [] [] [create delete deletecollection get list patch update watch]
pods/attach [] [] [create delete deletecollection get list patch update watch]
pods/exec [] [] [create delete deletecollection get list patch update watch]
pods/log [] [] [get list watch]
pods/portforward [] [] [create delete deletecollection get list patch update watch]
pods/proxy [] [] [create delete deletecollection get list patch update watch]
pods/status [] [] [get list watch]
podsecuritypolicyreviews [] [] [create]
podsecuritypolicyreviews.security.openshift.io [] [] [create]
podsecuritypolicyselfsubjectreviews [] [] [create]
podsecuritypolicyselfsubjectreviews.security.openshift.io [] [] [create]
podsecuritypolicysubjectreviews [] [] [create]
podsecuritypolicysubjectreviews.security.openshift.io [] [] [create]
processedtemplates [] [] [create delete deletecollection get list patch update watch]
processedtemplates.template.openshift.io [] [] [create delete deletecollection get list patch update
watch]
projects [] [] [delete get patch update]
projects.project.openshift.io [] [] [delete get patch update]
replicasets.extensions [] [] [create delete deletecollection get list patch update watch]
replicasets.extensions/scale [] [] [create delete deletecollection get list patch update watch]
replicationcontrollers [] [] [create delete deletecollection get list patch update watch]
replicationcontrollers/scale [] [] [create delete deletecollection get list patch update watch]
replicationcontrollers.extensions/scale [] [] [create delete deletecollection get list patch update
watch]
replicationcontrollers/status [] [] [get list watch]
resourceaccessreviews [] [] [create]
resourceaccessreviews.authorization.openshift.io [] [] [create]
resourcequotas [] [] [get list watch]
resourcequotas/status [] [] [get list watch]

```

```

resourcequotasages [] [] [get list watch]
rolebindingrestrictions [] [] [get list watch]
rolebindingrestrictions.authorization.openshift.io [] [] [get list watch]
rolebindings [] [] [create delete deletecollection get list patch update watch]
rolebindings.authorization.openshift.io [] [] [create delete deletecollection get list patch update
watch]
rolebindings.rbac.authorization.k8s.io [] [] [create delete deletecollection get list patch update
watch]
roles [] [] [create delete deletecollection get list patch update watch]
roles.authorization.openshift.io [] [] [create delete deletecollection get list patch update watch]
roles.rbac.authorization.k8s.io [] [] [create delete deletecollection get list patch update watch]
routes [] [] [create delete deletecollection get list patch update watch]
routes.route.openshift.io [] [] [create delete deletecollection get list patch update watch]
routes/custom-host [] [] [create]
routes.route.openshift.io/custom-host [] [] [create]
routes/status [] [] [get list watch update]
routes.route.openshift.io/status [] [] [get list watch update]
scheduledjobs.batch [] [] [create delete deletecollection get list patch update watch]
secrets [] [] [create delete deletecollection get list patch update watch]
serviceaccounts [] [] [create delete deletecollection get list patch update watch impersonate]
services [] [] [create delete deletecollection get list patch update watch]
services/proxy [] [] [create delete deletecollection get list patch update watch]
statefulsets.apps [] [] [create delete deletecollection get list patch update watch]
subjectaccessreviews [] [] [create]
subjectaccessreviews.authorization.openshift.io [] [] [create]
subjectrulesreviews [] [] [create]
subjectrulesreviews.authorization.openshift.io [] [] [create]
templateconfigs [] [] [create delete deletecollection get list patch update watch]
templateconfigs.template.openshift.io [] [] [create delete deletecollection get list patch update
watch]
templateinstances [] [] [create delete deletecollection get list patch update watch]
templateinstances.template.openshift.io [] [] [create delete deletecollection get list patch update
watch]
templates [] [] [create delete deletecollection get list patch update watch]
templates.template.openshift.io [] [] [create delete deletecollection get list patch update watch]

```

Name: basic-user

Labels: <none>

Annotations: openshift.io/description=A user that can get basic information about projects.

rbac.authorization.kubernetes.io/autoupdate=true

PolicyRule:

Resources Non-Resource URLs Resource Names Verbs

```

-----
clusterroles [] [] [get list]
clusterroles.authorization.openshift.io [] [] [get list]
clusterroles.rbac.authorization.k8s.io [] [] [get list watch]
projectrequests [] [] [list]
projectrequests.project.openshift.io [] [] [list]
projects [] [] [list watch]
projects.project.openshift.io [] [] [list watch]
selfsubjectaccessreviews.authorization.k8s.io [] [] [create]
selfsubjectrulesreviews [] [] [create]
selfsubjectrulesreviews.authorization.openshift.io [] [] [create]

```

```
storageclasses.storage.k8s.io [] [] [get list]
users [] [~] [get]
users.user.openshift.io [] [~] [get]
```

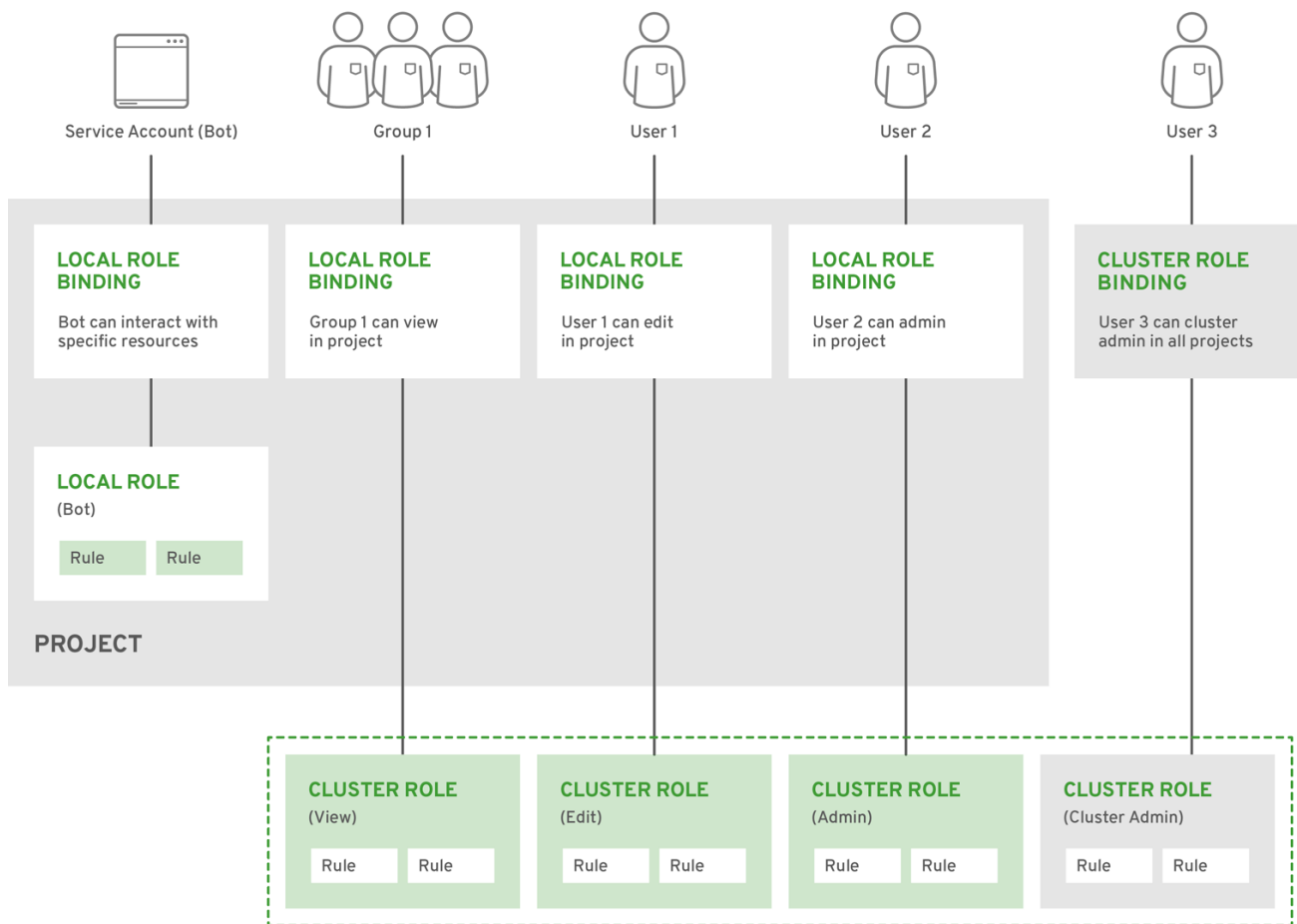
ローカルロールバインディングを表示して得られる以下の概要は、各種のユーザーおよびグループにバインドされた上記のロールを示しています。

```
oc describe rolebinding.rbac admin basic-user -n alice-project
```

```
Name: admin
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: admin
Subjects:
  Kind Name Namespace
  ----
  User system:admin
  User alice
```

```
Name: basic-user
Labels: <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: basic-user
Subjects:
  Kind Name Namespace
  ----
  User joe
  Group devel
```

クラスターロール、クラスターロールのバインディング、ローカルロールのバインディング、ユーザー、グループおよびサービスアカウントの関係は以下に説明されています。



OPENSIFT_415489_0218

4.2.2. 承認の評価

OpenShift Container Platform が承認を評価する際、いくつかの要因が組み合わさって決定が行われます。

アイデンティティ	承認のコンテキストでは、ユーザー名およびユーザーが属するグループの一覧になります。						
アクション	<p>実行されるアクションです。ほとんどの場合、これは以下で構成されます。</p> <table border="1"> <tr> <td>プロジェクト</td> <td>アクセスされるプロジェクト。</td> </tr> <tr> <td>動詞</td> <td>Can be get, list, create, update, delete, deletecollection or watch.</td> </tr> <tr> <td>リソース名</td> <td>アクセスされる API エンドポイント。</td> </tr> </table>	プロジェクト	アクセスされるプロジェクト。	動詞	Can be get, list, create, update, delete, deletecollection or watch .	リソース名	アクセスされる API エンドポイント。
プロジェクト	アクセスされるプロジェクト。						
動詞	Can be get, list, create, update, delete, deletecollection or watch .						
リソース名	アクセスされる API エンドポイント。						
バインディング	バインディングの詳細一覧です。						

OpenShift Container Platform は以下の手順を使って承認を評価します。

1. アイデンティティおよびプロジェクトでスコープ設定されたアクションは、ユーザーおよびそれらのグループに適用されるすべてのバインディングを検索します。
2. バインディングは、適用されるすべてのロールを見つけるために使用されます。
3. ロールは、適用されるすべてのルールを見つけるために使用されます。
4. 一致を見つけるために、アクションが各ルールに対してチェックされます。
5. 一致するルールが見つからない場合、アクションはデフォルトで拒否されます。

4.2.3. クラスターおよびローカル RBAC

承認を制御する2つのレベルのRBACロールおよびバインディングがあります。

クラスター RBAC	すべてのプロジェクトで適用可能な ロール およびバインディングです。クラスター全体で存在するロールは クラスターロール と見なされます。クラスターロールのバインディングはクラスターロールのみを参照できます。
ローカル RBAC	所定のプロジェクトにスコープ設定されている ロール およびバインディングです。プロジェクトにのみ存在するロールは ローカルロール とみなされます。ローカルロールのバインディングはクラスターロールおよびローカルロールの両方を参照できます。

この2つのレベルからなる階層により、ローカルロールでの個別プロジェクト内のカスタマイズが可能になる一方で、クラスターロールによる複数プロジェクト間での再利用が可能になります。

評価時に、クラスターロールのバインディングおよびローカルロールのバインディングが使用されません。以下は例になります。

1. クラスター全体の「allow」ルールがチェックされます。
2. ローカルにバインドされた「allow」ルールがチェックされます。
3. デフォルトで拒否します。

4.2.4. クラスターロールおよびローカルロール

ロールはポリシー**ルール**のコレクションであり、一連のリソースで実行可能な一連の許可された動詞です。OpenShift Container Platformには、**クラスター全体**または**ローカル**で、ユーザーおよびグループにバインドできるデフォルトのクラスターロールのセットが含まれます。

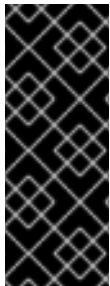
デフォルトのクラスターロール	説明
admin	プロジェクトマネージャーです。 ローカルバインディング で使用されている場合、 admin ユーザーにはプロジェクトのリソースを閲覧し、クォータを除くプロジェクトのすべてのリソースを変更する権限があります。
basic-user	プロジェクトおよびユーザーについての基本的な情報を取得できるユーザーです。

デフォルトのクラスターロール	説明
cluster-admin	A super-user that can perform any action in any project. When bound to a user with a local binding , they have full control over quota and every action on every resource in the project.
cluster-status	基本的なクラスターのステータス情報を取得できるユーザーです。
edit	プロジェクトのほとんどのプロジェクトを変更できるが、ロールまたはバインディングを表示したり、変更したりする機能を持たないユーザーです。
self-provisioner	独自のプロジェクトを作成できるユーザーです。
view	変更できないものの、プロジェクトでほとんどのオブジェクトを確認できるユーザーです。それらはロールまたはバインディングを表示したり、変更したりできません。

ヒント

[ユーザーおよびグループ](#)は複数のロールに同時に関連付けたり、[バインド](#)できることに注意してください。

Project administrators can visualize roles, including a matrix of the verbs and resources each are associated using the CLI to [view local bindings](#).



重要

The cluster role bound to the project administrator is limited in a project via a [local binding](#). It is not bound [cluster-wide](#) like the cluster roles granted to the **cluster-admin** or **system:admin**.

クラスターロールは、クラスターレベルで定義される [ロール](#) ですが、クラスターレベルまたはプロジェクトレベルのいずれかでバインドできます。

[Learn how to create a local role for a project](#) .

4.2.4.1. クラスターロールの更新

After any [OpenShift Container Platform cluster upgrade](#), the default roles are updated and automatically reconciled when the server is started. Additionally, see [Updating Policy Definitions](#) for instructions on getting other recommendations using:

```
$ oc adm policy reconcile-cluster-roles
```

4.2.4.2. カスタムロールおよびパーミッションの適用

To add or update custom roles and permissions, it is strongly recommended to use the following command:

```
# oc auth reconcile -f FILE
```

このコマンドは、他のクライアントを切断しない方法で新規パーミッションが適切に適用されるようにします。これは、ルールセット間の論理カバー操作を計算することにより内部で実行され、RBAC リリースの JSON マージでは実行できません。

4.2.4.3. クラスターロールの集計

The default admin, edit, and view cluster roles support [cluster role aggregation](#), where the cluster rules for each role are dynamically updated as new rules are created. This feature is relevant only if you extend the Kubernetes API by [creating custom resources](#).

[Learn how to use cluster role aggregation](#) .

4.2.5. SCC (Security Context Constraints)

In addition to the [RBAC resources](#) that control what a user can do, OpenShift Container Platform provides **security context constraints** (SCC) that control the actions that a [pod](#) can perform and what it has the ability to access. Administrators can [manage SCCs](#) using the CLI.

SCC は永続ストレージへのアクセスを管理する場合にも非常に便利です。

SCC は、システムで許可されるために Pod の実行時に必要となる一連の条件を定義するオブジェクトです。管理者は以下を制御できます。

1. Running of [privileged containers](#).
2. コンテナが要求できる機能の追加
3. ホストディレクトリーのボリュームとしての使用
4. コンテナの SELinux コンテキスト
5. ユーザー ID。
6. ホストの namespace およびネットワークの使用
7. Pod のボリュームを所有する **FSGroup** の割り当て
8. 許可される補助グループの設定
9. 読み取り専用のルートファイルシステムの要求
10. ボリュームタイプの使用の制御
11. 許可される seccomp プロファイルの設定

デフォルトでは、7つの SCC がクラスターに追加され、クラスター管理者は CLI を使用してそれらを表示できます。

```
$ oc get scc
NAME          PRIV  CAPS  SELINUX  RUNASUSER  FSGROUP  SUPGROUP
PRIORITY  READONLYROOTFS  VOLUMES
anyuid      false  []    MustRunAs  RunAsAny   RunAsAny  RunAsAny  10    false
[configMap downwardAPI emptyDir persistentVolumeClaim secret]
hostaccess  false  []    MustRunAs  MustRunAsRange  MustRunAs  RunAsAny  <none>
false      [configMap downwardAPI emptyDir hostPath persistentVolumeClaim secret]
hostmount-anyuid  false  []    MustRunAs  RunAsAny   RunAsAny  RunAsAny  <none>
```

```

false      [configMap downwardAPI emptyDir hostPath nfs persistentVolumeClaim secret]
hostnetwork false []      MustRunAs MustRunAsRange MustRunAs MustRunAs <none>
false      [configMap downwardAPI emptyDir persistentVolumeClaim secret]
nonroot    false []      MustRunAs MustRunAsNonRoot RunAsAny RunAsAny <none>
false      [configMap downwardAPI emptyDir persistentVolumeClaim secret]
privileged true  [*]     RunAsAny RunAsAny      RunAsAny RunAsAny <none>
false      [*]
restricted false []      MustRunAs MustRunAsRange MustRunAs RunAsAny <none>
false      [configMap downwardAPI emptyDir persistentVolumeClaim secret]

```



重要

Do not modify the default SCCs. Customizing the default SCCs can lead to issues when OpenShift Container Platform is upgraded. Instead, [create new SCCs](#).

各 SCC の定義についても、クラスター管理者は CLI を使用して表示できます。たとえば、特権付き SCC の場合は、以下のようになります。

```

# oc export scc/privileged
allowHostDirVolumePlugin: true
allowHostIPC: true
allowHostNetwork: true
allowHostPID: true
allowHostPorts: true
allowPrivilegedContainer: true
allowedCapabilities: ❶
- '*'
apiVersion: v1
defaultAddCapabilities: [] ❷
fsGroup: ❸
  type: RunAsAny
groups: ❹
- system:cluster-admins
- system:nodes
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: 'privileged allows access to all privileged and host
      features and the ability to run as any user, any group, any fsGroup, and with
      any SELinux context. WARNING: this is the most relaxed SCC and should be used
      only for cluster administration. Grant with caution.'
  creationTimestamp: null
  name: privileged
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: [] ❺
runAsUser: ❻
  type: RunAsAny
seLinuxContext: ❼
  type: RunAsAny
seccompProfiles:
- '*'
supplementalGroups: ❽
  type: RunAsAny

```



```

users: 9
- system:serviceaccount:default:registry
- system:serviceaccount:default:router
- system:serviceaccount:openshift-infra:build-controller
volumes:
- '*'

```

- 1 Pod で要求できる要求の一覧です。特殊な記号 * は任意の機能を許可しますが、一覧が空の場合は、いずれの機能も要求できないことを意味します。
- 2 Pod に含める追加機能の一覧です。
- 3 セキュリティーコンテキストの許可される値を定める **FSGroup** ストラテジータイプです。
- 4 この SCC へのアクセスを持つグループです。
- 5 Pod からドロップされる機能の一覧です。
- 6 セキュリティーコンテキストの許可される値を定める run as user ストラテジータイプです。
- 7 セキュリティーコンテキストの許可される値を定める SELinux コンテキストストラテジータイプです。
- 8 セキュリティーコンテキストの許可される補助グループを定める補助グループストラテジーです。
- 9 この SCC へのアクセスを持つユーザーです。

SCC の **users** および **groups** フィールドは使用できる SCC を制御します。デフォルトで、クラスター管理者、ノードおよびビルドコントローラーには特権付き SCC へのアクセスが付与されます。認証されるすべてのユーザーには制限付き SCC へのアクセスが付与されます。

Docker には、Pod の各コンテナについて許可される [デフォルトの機能一覧](#) があります。コンテナはこれらの機能をデフォルト一覧から使用しますが、Pod マニフェストの作成者は追加機能を要求したり、デフォルトから一部をドロップしてこの一覧を変更できます。 **allowedCapabilities**、 **defaultAddCapabilities**、 および **requiredDropCapabilities** フィールドは Pod からのこのような要求を制御し、要求できる機能を決定し、各コンテナに追加するものや禁止する必要のあるものを決定するために使用されます。

特権付き SCC:

- 特権付き Pod を許可します。
- ホストディレクトリーのボリュームとしてのマウントを許可します。
- Pod の任意ユーザーとしての実行を許可します。
- Pod の MCS ラベルを使った実行を許可します。
- Pod がホストの IPC namespace を使用することを許可します。
- Pod がホストの PID namespace を使用することを許可します。
- Pod が FSGroup を使用することを許可します。
- Pod が補助グループを使用することを許可します。
- Pod が seccomp プロファイルを使用することを許可します。

- Pod が任意の機能を要求することを許可します。

制限付き SCC:

- Pod が特権付きとして実行できないようにします。
- Pod がホストディレクトリーボリュームを使用できないようにします。
- Pod が事前に割り当てられた UID の範囲でユーザーとして実行されることを要求します。
- Pod が事前に割り当てられた MCS ラベルで実行されることを要求します。
- Pod が FSGroup を使用することを許可します。
- Pod が補助グループを使用することを許可します。



注記

各 SCC の詳細は、SCC で利用可能な kubernetes.io/description アノテーションを参照してください。

SCC は Pod がアクセスできるセキュリティ機能を制限する各種の設定およびストラテジーで構成されています。これらの設定は以下のカテゴリーに分類されます。

ブール値による制御	このタイプのフィールドはデフォルトで最も制限のある値に設定されます。たとえば、 AllowPrivilegedContainer は指定されていない場合は、 false に常に設定されます。
許可されるセットによる制御	このタイプのフィールドはセットに対してチェックされ、それらの値が許可されることを確認します。
ストラテジーによる制御	値を生成するストラテジーを持つ項目は以下を提供します。 <ul style="list-style-type: none"> ● 値を生成するメカニズム ● 指定された値が許可される値のセットに属するようにするメカニズム

4.2.5.1. SCC ストラテジー

4.2.5.1.1. RunAsUser

1. **MustRunAs - runAsUser** が設定されることを要求します。デフォルトで設定済みの **runAsUser** を使用します。設定済みの **runAsUser** に対して検証します。
2. **MustRunAsRange** - 事前に割り当てられた値を使用していない場合に、最小および最大値が定義されることを要求します。デフォルトでは最小値を使用します。許可される範囲全体に対して検証します。
3. **MustRunAsNonRoot** - Pod がゼロ以外の **runAsUser** で送信されること、または **USER** 命令をイメージに定義することを要求します。デフォルトは指定されません。
4. **RunAsAny** - デフォルトは指定されません。 **runAsUser** の指定を許可します。

4.2.5.1.2. SELinuxContext

1. **MustRunAs** - 事前に割り当てられた値を使用していない場合に **seLinuxOptions** が設定されることを要求します。デフォルトとして **seLinuxOptions** を使用します。 **seLinuxOptions** に対して検証します。
2. **RunAsAny** - デフォルトは指定されません。 **seLinuxOptions** の指定を許可します。

4.2.5.1.3. SupplementalGroups

1. **MustRunAs** - 事前に割り当てられた値を使用していない場合に、少なくとも1つの範囲が指定されることを要求します。デフォルトとして最初の範囲の最小値を使用します。すべての範囲に対して検証します。
2. **RunAsAny** - デフォルトは指定されません。 **supplementalGroups** の指定を許可します。

4.2.5.1.4. FSGroup

1. **MustRunAs** - 事前に割り当てられた値を使用していない場合に、少なくとも1つの範囲が指定されることを要求します。デフォルトとして最初の範囲の最小値を使用します。最初の範囲の最初の ID に対して検証します。
2. **RunAsAny** - デフォルトは指定されません。 **fsGroup** ID の指定を許可します。

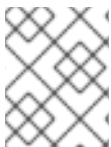
4.2.5.2. ボリュームの制御

特定のボリュームタイプの使用は、SCC の **volumes** フィールドを設定して制御できます。このフィールドの許容値は、ボリュームの作成時に定義されるボリュームソースに対応します。

- [azureFile](#)
- [azureDisk](#)
- [flocker](#)
- [flexVolume](#)
- [hostPath](#)
- [emptyDir](#)
- [gcePersistentDisk](#)
- [awsElasticBlockStore](#)
- [gitRepo](#)
- [secret](#)
- [nfs](#)
- [iscsi](#)
- [glusterfs](#)
- [persistentVolumeClaim](#)

- [rbd](#)
- [cinder](#)
- [cephFS](#)
- [downwardAPI](#)
- [fc](#)
- [configMap](#)
- [vsphereVolume](#)
- [quobyte](#)
- [photonPersistentDisk](#)
- [projected](#)
- [portworxVolume](#)
- [scaleIO](#)
- [storageos](#)
- * (すべてのボリュームタイプの使用を許可する特殊な値)
- [none](#) (すべてのボリュームタイプの使用を無効にする特殊な値。後方互換の場合にのみ存在します)

新規 SCC の許可されるボリュームの推奨される最小セットは `configMap`、`downwardAPI`、`emptyDir`、`persistentVolumeClaim`、`secret`、および `projected` です。



注記

許可されるボリュームタイプの一覧は、新規タイプが OpenShift Container Platform の各リリースと共に追加されるため、網羅的な一覧である必要はありません。



注記

後方互換性のために、`allowHostDirVolumePlugin` の使用は `volumes` フィールドの設定を上書きします。たとえば、`allowHostDirVolumePlugin` が `false` に設定されるが、`volumes` フィールドで許可される場合、`hostPath` 値は `volumes` から削除されません。

4.2.5.3. FlexVolume へのアクセスの制限

OpenShift Container Platform は、それらのドライバーに基づいて FlexVolume の追加の制御を提供します。SCC が FlexVolume の使用を許可する場合、Pod は任意の FlexVolume を要求できます。ただし、クラスター管理者が `AllowedFlexVolumes` フィールドでドライバー名を指定する場合、Pod はこれらのドライバーでのみ FlexVolumes を使用する必要があります。

アクセスを 2 つの FlexVolume のみに制限する例

```
volumes:
```

```
- flexVolume
allowedFlexVolumes:
- driver: example/lvm
- driver: example/cifs
```

4.2.5.4. Seccomp

SeccompProfiles は、Pod またはコンテナの **seccomp** アノテーションに設定できる許可されるプロファイルを一覧表示します。未使用 (nil) または空の値は、プロファイルが Pod またはコンテナで指定されないことを意味します。ワイルドカード * を使用してすべてのプロファイルを許可します。Pod の値を生成するために使用される場合、最初のワイルドカード以外のプロファイルがデフォルトとして使用されます。

Refer to the [seccomp documentation](#) for more information about configuring and using custom profiles.

4.2.5.5. 受付

SCC が設定された **受付制御** により、ユーザーに付与された機能に基づいてリソースの作成に対する制御が可能になります。

SCC の観点では、これは受付コントローラーが、SCC の適切なセットを取得するためにコンテキストで利用可能なユーザー情報を検査できることを意味します。これにより、Pod はその運用環境についての要求を行ったり、Pod に適用する一連の制約を生成したりする権限が与えられます

受付が Pod を許可するために使用する SCC のセットはユーザーアイデンティティおよびユーザーが属するグループによって決定されます。さらに、Pod がサービスアカウントを指定する場合、許可される SCC のセットには、サービスアカウントでアクセスできる制約が含まれます。

受付は以下の方法を使用して、Pod の最終的なセキュリティーコンテキストを作成します。

1. 使用できるすべての SCC を取得します。
2. 要求に指定されていないセキュリティーコンテキストの設定のフィールド値を生成します。
3. 利用可能な制約に対する最終的な設定を検証します。

制約の一致するセットが検出される場合、Pod が受け入れられます。要求が SCC に一致しない場合、Pod は拒否されます。

Pod はすべてのフィールドを SCC に対して検証する必要があります。以下は、検証する必要のある 2 つのフィールドのみについての例になります。



注記

これらの例は、事前に割り当てられる値を使用するストレージに関連するものです。

MustRunAs の FSGroup SCC ストラテジー

Pod が **fsGroup** ID を定義する場合、その ID はデフォルトの **fsGroup** ID に等しくなければなりません。そうでない場合、Pod はその SCC によって検証されず、次の SCC が評価されます。

SecurityContextConstraints.fsGroup フィールドに値 **RunAsAny** があり、Pod 仕様が **Pod.spec.securityContext.fsGroup** を省略する場合、このフィールドは有効とみなされます。検証時に、他の SCC 設定が他の Pod フィールドを拒否し、そのため Pod を失敗させる可能性があることに注意してください。

MustRunAs の SupplementalGroups SCC ストラテジー

Pod 仕様が1つ以上の **supplementalGroups** ID を定義する場合、Pod の ID は namespace の **openshift.io/sa.scc.supplemental-groups** アノテーションの ID のいずれかに等しくなければなりません。そうでない場合は、Pod は SCC で検証されず、次の SCC が評価されます。

SecurityContextConstraints.supplementalGroups フィールドに値 **RunAsAny** があり、Pod 仕様が **Pod.spec.securityContext.supplementalGroups** を省略する場合、このフィールドは有効とみなされます。検証時に、他の SCC 設定が他の Pod フィールドを拒否し、そのため Pod を失敗させる可能性があることに注意してください。

4.2.5.5.1. SCC の優先度設定

SCC には、受付コントローラーによる要求の検証を試行する際の順序に影響を与える優先度フィールドがあります。優先度の高い SCC は並び替える際にセットの先頭に移動します。利用可能な SCC の完全なセットが決定されると、それらは以下に戻づいて順序付けられます。

1. 優先度が高い順。nil は優先度 0 とみなされます。
2. 優先度が等しい場合、SCC は最も制限の多いものから少ないものの順に並べ替えられます。
3. 優先度と制限のどちらも等しい場合、SCC は名前順に並べ替えられます。

デフォルトで、クラスター管理者に付与される anyuid SCC には SCC セットの優先度が指定されます。これにより、クラスター管理者は Pod の **SecurityContext** で **RunAsUser** を指定しなくても Pod を任意のユーザーとして実行できます。管理者は、希望する場合は依然として **RunAsUser** を指定できます。

4.2.5.5.2. 事前に割り当てられた値および SCC (Security Context Constraints) について

受付コントローラーは、これが namespace の事前に設定された値を検索し、Pod の処理前に SCC (Security Context Constraints) を設定するようにトリガーする SCC (Security Context Constraint) の特定の条件を認識します。各 SCC ストラテジーは他のストラテジーとは別個に評価されます。この際、(許可される場合に) Pod 仕様の値と共に集計された各ポリシーの事前に割り当てられた値が使用され、実行中の Pod で定義される各種 ID の最終の値が設定されます。

以下の SCC により、受付コントローラーは、範囲が Pod 仕様で定義されていない場合に事前に定義された値を検索できます。

1. 最小または最大値が設定されていない **MustRunAsRange** の **RunAsUser** ストラテジーです。受付は **openshift.io/sa.scc.uid-range** アノテーションを検索して範囲フィールドを設定します。
2. レベルが設定されていない **MustRunAs** の **SELinuxContext** ストラテジーです。受付は **openshift.io/sa.scc.mcs** アノテーションを検索してレベルを設定します。
3. **MustRunAs** の **FSGroup** ストラテジーです。受付は **openshift.io/sa.scc.supplemental-groups** アノテーションを検索します。
4. **MustRunAs** の **SupplementalGroups** ストラテジーです。受付は **openshift.io/sa.scc.supplemental-groups** アノテーションを検索します。

生成フェーズでは、セキュリティーコンテキストのプロバイダーが Pod にとくに設定されていない値をデフォルト設定します。デフォルト設定は使用されるストラテジーに基づいて行われます。

1. **RunAsAny** および **MustRunAsNonRoot** ストラテジーはデフォルトの値を提供しません。そのため、Pod が定義されるフィールドを必要とする場合 (グループ ID など)、このフィールドは Pod 仕様内に定義する必要があります。
2. **MustRunAs** (単一の値) ストラテジーは、常に使用されるデフォルト値を提供します。たとえば、グループ ID の場合、Pod 仕様が独自の ID 値を定義する場合でも、namespace のデフォルトフィールドが Pod のグループに表示されます。
3. **MustRunAsRange** および **MustRunAs** (範囲ベース) ストラテジーは、範囲の最小値を提供します。単一の値の **MustRunAs** ストラテジーの場合のように、namespace のデフォルト値は実行中の Pod に表示されます。範囲ベースのストラテジーが複数の範囲で設定可能な場合、これは最初に設定された範囲の最小値を指定します。



注記

FSGroup および **SupplementalGroups** ストラテジー

は、openshift.io/sa.scc.supplemental-groups アノテーションが namespace に存在しない場合に openshift.io/sa.scc.uid-range アノテーションにフォールバックします。いずれも存在しない場合、SCC は作成に失敗します。



注記

デフォルトで、アノテーションベースの **FSGroup** ストラテジーは、自らをアノテーションの最小値に基づく単一の範囲で設定します。たとえば、アノテーションが `1/3` を読み取る場合、**FSGroup** ストラテジーは `1` の最小値および最大値で自らを設定します。追加のグループを **FSGroup** フィールドで許可する必要がある場合、アノテーションを使用しないカスタム SCC を設定することができます。



注記

openshift.io/sa.scc.supplemental-groups アノテーションは、`<start>/<length>` または `<start>-<end>` 形式のカンマ区切りのブロックの一覧を受け入れます。openshift.io/sa.scc.uid-range アノテーションは単一ブロックのみを受け入れます。

4.2.6. 認証済みのユーザーとして何が実行できるのかを判断する方法

OpenShift Container Platform プロジェクト内で、namespace でスコープ設定されたすべてのリソース (サードパーティーのリソースを含む) に対して実行できる動詞を判別します。以下を実行します。

```
$ oc policy can-i --list --loglevel=8
```

この出力で、情報収集のために実行する必要のある API 要求を判断しやすくなります。

ユーザーが判読可能な形式で情報を取得し直すには、以下を実行します。

```
$ oc policy can-i --list
```

この出力により、詳細な一覧が表示されます。

特定の動詞が実行可能かどうかを判断するには、以下を実行します。

```
$ oc policy can-i <verb> <resource>
```

[User scopes](#) can provide more information about a given scope. For example:

```
$ oc policy can-i <verb> <resource> --scopes=user:info
```

4.3. 永続ストレージ

4.3.1. 概要

Managing storage is a distinct problem from managing compute resources. OpenShift Container Platform leverages the Kubernetes persistent volume (PV) framework to allow cluster administrators to provision persistent storage for a cluster. Using persistent volume claims (PVCs), developers can request PV resources without having specific knowledge of the underlying storage infrastructure.

PVCs are specific to a [project](#) and are created and used by developers as a means to use a PV. PV resources on their own are not scoped to any single project; they can be shared across the entire OpenShift Container Platform cluster and claimed from any project. After a PV is **bound** to a PVC, however, that PV cannot then be bound to additional PVCs. This has the effect of scoping a bound PV to a single [namespace](#) (that of the binding project).

PV は、クラスター管理者によってプロビジョニングされるクラスターの既存のネットワーク設定されたストレージの一部を表す **PersistentVolume** API オブジェクトで定義されます。これは、ノードがクラスターリソースであるのと同様にクラスター内のリソースです。PV は **Volumes** のようなボリュームプラグインですが、PV を使用する個々の **Pod** から独立したライフサイクルを持ちます。PV オブジェクトは、NFS、iSCSI、またはクラウドプロバイダー固有のストレージシステムのいずれの場合でも、ストレージの実装の詳細をキャプチャーします。



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。

PVCs are defined by a **PersistentVolumeClaim** API object, which represents a request for storage by a developer. It is similar to a pod in that pods consume node resources and PVCs consume PV resources. For example, pods can request specific levels of resources (e.g., CPU and memory), while PVCs can request specific [storage capacity](#) and [access modes](#) (e.g, they can be mounted once read/write or many times read-only).

4.3.2. Lifecycle of a Volume and Claim

PV はクラスターのリソースです。PVC はそれらのリソースの要求であり、リソースに対する要求チェックとして機能します。PV と PVC 間の相互作用には以下のライフサイクルが設定されます。

4.3.2.1. Provisioning

PVC で定義される開発者からの要求に対応し、クラスター管理者はストレージおよび一致する PV をプロビジョニングする 1 つ以上の動的プロビジョナーを設定します。

または、クラスター管理者は、使用可能な実際のストレージの詳細を保持する多数の PV を前もって作成できます。PV は API に存在し、利用可能な状態になります。

4.3.2.2. Binding

When you create a PVC, you request a specific amount of storage, specify the required access mode,

and can create a storage class to describe and classify the storage. The control loop in the master watches for new PVCs and binds the new PVC to an appropriate PV. If an appropriate PV does not exist, a provisioner for the storage class creates one.

PV ボリュームは、要求したボリュームを上回る可能性があります。これは、手動でプロビジョニングされた PV の場合に特にそう言えます。超過を最小限にするために、OpenShift Container Platform は他のすべての条件に一致する最小の PV にバインドします。

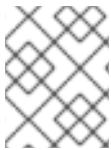
要求は、一致するボリュームが存在しないか、ストレージクラスを提供するいずれの利用可能なプロビジョナーで作成されない場合には無期限にバインドされないままになります。要求は、一致するボリュームが利用可能になるとバインドされます。たとえば、多数の手動でプロビジョニングされた 50Gi ボリュームを持つクラスターは 100Gi を要求する PVC に一致しません。PVC は 100Gi PV がクラスターに追加されるとバインドされます。

4.3.2.3. Using

Pod は要求をボリュームとして使用します。クラスターは要求を検査して、バインドされたボリュームを検索し、Pod にそのボリュームをマウントします。複数のアクセスモードをサポートするボリュームの場合、要求を Pod のボリュームとして使用する際に適用するモードを指定する必要があります。

Once you have a claim and that claim is bound, the bound PV belongs to you for as long as you need it. You can schedule pods and access claimed PVs by including **persistentVolumeClaim** in the pod's volumes block. See [below](#) for syntax details.

4.3.2.4. Persistent Volume Claim Protection



注記

PVC protection is an alpha feature and may change in a future release of OpenShift Container Platform.

The purpose of PVC protection is to ensure that PVCs in active use by a pod are not removed from the system, as this may result in data loss.



注記

A PVC is in active use by a pod when the pod status is **Pending**, and the pod is assigned to a node or the pod status is **Running**.

When the PVC protection feature is enabled, if you delete a PVC that is in active use by a pod, the PVC is not immediately removed. PVC removal is postponed until the PVC is no longer actively used by any pods.

You can see that a PVC is protected when the PVC's status is **Terminating** and the **Finalizers** list includes **kubernetes.io/pvc-protection**:

```
oc describe pvc hostpath
Name:          hostpath
Namespace:    default
StorageClass: example-hostpath
Status:       Terminating
Volume:
Labels:       <none>
Annotations:  volume.beta.kubernetes.io/storage-class=example-hostpath
```

```

    volume.beta.kubernetes.io/storage-provisioner=example.com/hostpath
Finalizers: [kubernetes.io/pvc-protection]
...

```

To enable PVC protection, see [Configuring Persistent Volume Claim Protection](#).

4.3.2.5. Releasing

When you are done with a volume, you can delete the PVC object from the API, which allows reclamation of the resource. The volume is considered "released" when the claim is deleted, but it is not yet available for another claim. The previous claimant's data remains on the volume and must be handled according to policy.

4.3.2.6. Reclaiming

The reclaim policy of a **PersistentVolume** tells the cluster what to do with the volume after it is released. Volumes reclaim policy can either be **Retained**, **Recycled**, or **Deleted**.

Retained reclaim policy allows manual reclamation of the resource for those volume plug-ins that support it. **Deleted** reclaim policy deletes both the **PersistentVolume** object from OpenShift Container Platform and the associated storage asset in external infrastructure, such as AWS EBS, GCE PD, or Cinder volume.



注記

Volumes that were dynamically provisioned are always deleted.

4.3.2.6.1. Recycling

If supported by appropriate volume plug-in, recycling performs a basic scrub (**rm -rf /thevolume/***) on the volume and makes it available again for a new claim.



警告

The **recycle** reclaim policy is deprecated in favor of dynamic provisioning and it will be removed in future releases.

You can configure a custom recycler pod template using the controller manager command line arguments as described in the [ControllerArguments](#) section. The custom recycler pod template must contain a **volumes** specification, as shown in the example below:

```

apiVersion: v1
kind: Pod
metadata:
  name: pv-recycler-
  namespace: openshift-infra 1
spec:
  restartPolicy: Never
  serviceAccount: pv-recycler-controller 2

```

```

volumes:
- name: nfsvol
  nfs:
    server: any-server-it-will-be-replaced 3
    path: any-path-it-will-be-replaced 4
containers:
- name: pv-recycler
  image: "gcr.io/google_containers/busybox"
  command: ["/bin/sh", "-c", "test -e /scrub && rm -rf /scrub/..?* /scrub/.!.* /scrub/* && test -z \"$(ls -
A /scrub)\" || exit 1"]
  volumeMounts:
  - name: nfsvol
    mountPath: /scrub

```

- 1** Namespace where the recycler pod runs. **openshift-infra** is the recommended namespace, as it already has a **pv-recycler-controller** service account that can recycle volumes.
- 2** Name of service account that is allowed to mount NFS volumes. It must exist in the specified namespace. A **pv-recycler-controller** account is recommended, as it is automatically created in **openshift-infra** namespace and has all the required permissions.
- 3** **4** The particular **server** and **path** values specified in the custom recycler pod template in the **volumes** part is replaced with the particular corresponding values from the PV that is being recycled.

4.3.3. Persistent Volumes

Each PV contains a **spec** and **status**, which is the specification and status of the volume.

Persistent Volume Object Definition

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: /tmp
    server: 172.17.0.2

```

4.3.3.1. Types of Persistent Volumes

OpenShift Container Platform は以下の **PersistentVolume** プラグインをサポートします。

- [NFS](#)
- [HostPath](#)
- [GlusterFS](#)

- [Ceph RBD](#)
- [OpenStack Cinder](#)
- [AWS Elastic Block Store \(EBS\)](#)
- [GCE Persistent Disk](#)
- [iSCSI](#)
- [Fibre Channel](#)
- [Azure Disk](#)
- [Azure File](#)
- [VMWare vSphere](#)
- [Local](#)

4.3.3.2. 容量

Generally, a PV has a specific storage capacity. This is set using the PV's **capacity** attribute.

現時点で、ストレージ容量は設定または要求できる唯一のリソースです。今後は属性として IOPS、スループットなどが含まれる可能性があります。

4.3.3.3. Access Modes

PersistentVolume はリソースプロバイダーでサポートされるすべての方法でホストにマウントできます。プロバイダーには各種の機能があり、それぞれの PV のアクセスモードは特定のボリュームでサポートされる特定のモードに設定されます。たとえば、NFS は複数の読み取り/書き込みクライアントをサポートしますが、特定の NFS PV は読み取り専用としてサーバー上でエクスポートされる可能性があります。それぞれの PV は、その特定の PV の機能について記述するアクセスモードの独自のセットを取得します。

Claims are matched to volumes with similar access modes. The only two matching criteria are access modes and size. A claim's access modes represent a request. Therefore, you might be granted more, but never less. For example, if a claim requests RWO, but the only volume available is an NFS PV (RWO+ROX+RWX), then the claim would match NFS because it supports RWO.

Direct matches are always attempted first. The volume's modes must match or contain more modes than you requested. The size must be greater than or equal to what is expected. If two types of volumes (NFS and iSCSI, for example) both have the same set of access modes, then either of them can match a claim with those modes. There is no ordering between types of volumes and no way to choose one type over another.

All volumes with the same modes are grouped, then sorted by size (smallest to largest). The binder gets the group with matching modes and iterates over each (in size order) until one size matches.

The access modes are:

アクセスモード	CLI Abbreviation	説明
---------	------------------	----

アクセスモード	CLI Abbreviation	説明
ReadWriteOnce	RWO	ボリュームは単一ノードで読み取り/書き込みとしてマウントできます。
ReadOnlyMany	ROX	ボリュームは数多くのノードで読み取り専用としてマウントできます。
ReadWriteMany	RWX	ボリュームは数多くのノードで読み取り/書き込みとしてマウントできます。

重要

ボリュームの **AccessModes** は、ボリュームの機能の記述子です。それらは施行されている制約ではありません。ストレージプロバイダーはリソースの無効な使用から生じるランタイムエラーに対応します。

たとえば、Ceph は **ReadWriteOnce** アクセスモードを提供します。ボリュームの **ROX** 機能を使用する必要がある場合は、要求に **read-only** のマークを付ける必要があります。プロバイダーのエラーは、マウントエラーとしてランタイム時に表示されます。

iSCSI and Fibre Channel volumes do not have any fencing mechanisms yet. You must ensure the volumes are only used by one node at a time. In certain situations, such as draining a node, the volumes can be used simultaneously by two nodes. Before draining the node, first ensure the pods that use these volumes are deleted.

The table below lists the access modes supported by different persistent volumes:

表4.1 Supported Access Modes for Persistent Volumes

ボリュームプラグイン	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWS EBS	■	-	-
Azure File	■	■	■
Azure Disk	■	-	-
Ceph RBD	■	■	-
ファイバーチャネル	■	■	-
GCE Persistent Disk	■	-	-
GlusterFS	■	■	■

ボリュームプラグイン	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
HostPath	■	-	-
iSCSI	■	■	-
NFS	■	■	■
Openstack Cinder	■	-	-
VMWare vSphere	■	-	-
ローカル	■	-	-



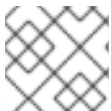
注記

- If pods rely on AWS EBS, GCE Persistent Disks, or Openstack Cinder PVs, use a [recreate deployment strategy](#)

4.3.3.4. Reclaim Policy

The current reclaim policies are:

Reclaim Policy	説明
Retain (保持)	Manual reclamation
Recycle	Basic scrub (e.g, <code>rm -rf /<volume>/*</code>)



注記

Currently, only NFS and HostPath support the 'Recycle' reclaim policy.



警告

The **recycle** reclaim policy is deprecated in favor of dynamic provisioning and it will be removed in future releases.

4.3.3.5. フェーズ

ボリュームは以下のフェーズのいずれかにあります。

フェーズ	説明
Available	A free resource that is not yet bound to a claim.
Bound	ボリュームが要求にバインドされています。
Released	要求が検出されていますが、リソースがまだクラスターにより回収されていません。
Failed	ボリュームが自動回収に失敗しています。

CLI には PV にバインドされている PVC の名前が表示されます。

4.3.3.6. Mount Options

アノテーション `volume.beta.kubernetes.io/mount-options` を使用して永続ボリュームのマウント中にマウントオプションを指定できます。

例:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
  annotations:
    volume.beta.kubernetes.io/mount-options: rw,nfsvers=4,noexec ❶
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    path: /tmp
    server: 172.17.0.2
  persistentVolumeReclaimPolicy: Recycle
  claimRef:
    name: claim1
    namespace: default
```

❶ Specified mount options are then used while mounting the persistent volume to the disk.

以下の永続ボリュームタイプがマウントオプションをサポートします。

- NFS
- GlusterFS
- Ceph RBD
- OpenStack Cinder

- AWS Elastic Block Store (EBS)
- GCE Persistent Disk
- iSCSI
- Azure Disk
- Azure File
- VMWare vSphere



注記

ファイバーチャネルおよび HostPath 永続ボリュームはマウントオプションをサポートしません。

4.3.4. Persistent Volume Claims

Each PVC contains a **spec** and **status**, which is the specification and status of the claim.

Persistent Volume Claim Object Definition

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
  storageClassName: gold
```

4.3.4.1. Storage Class

要求は、ストレージクラスの名前を **storageClassName** 属性に指定して特定のストレージクラスをオプションでリクエストできます。リクエストされたクラスの PV、つまり PVC と同じ **storageClassName** を持つ PV のみが PVC にバインドされます。クラスター管理者は1つ以上のストレージクラスを提供するように動的プロビジョナーを設定できます。クラスター管理者は、PVC の仕様に一致する PV をオンデマンドで作成できます。

クラスター管理者は、すべての PVC にデフォルトストレージクラスを設定することもできます。デフォルトのストレージクラスが設定されると、PVC は "" に設定された **StorageClass** または **storageClassName** アノテーションがストレージクラスなしの PV にバインドされるように明示的に要求する必要があります。

4.3.4.2. Access Modes

要求は、特定のアクセスモードのストレージを要求する際にボリュームと同じ規則を使用します。

4.3.4.3. リソース

Claims, like pods, can request specific quantities of a resource. In this case, the request is for storage. The same resource model applies to both volumes and claims.

4.3.4.4. Claims As Volumes

Pods access storage by using the claim as a volume. Claims must exist in the same namespace as the pod using the claim. The cluster finds the claim in the pod's namespace and uses it to get the **PersistentVolume** backing the claim. The volume is then mounted to the host and into the pod:

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
  - name: myfrontend
    image: dockerfile/nginx
    volumeMounts:
    - mountPath: "/var/www/html"
      name: mypd
  volumes:
  - name: mypd
    persistentVolumeClaim:
      claimName: myclaim
```

4.3.5. Block Volume Support



重要

Block Volume Support is a Technology Preview feature and it is only available for manually provisioned persistent volumes. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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。

You can statically provision raw block volumes by including some new API fields in your PV and PVC specifications.

ブロックボリュームを使用するには、まず **BlockVolume** 機能ゲートを有効にする必要があります。マスターの機能ゲートを有効にするには、**feature-gates** を **apiServerArguments** および **controllerArguments** に追加します。ノードの機能ゲートを有効にするには、**feature-gates** を **kubeletArguments** に追加します。以下は例になります。

```
kubeletArguments:
  feature-gates:
  - BlockVolume=true
```

For more information, see [Configuring for Local Volume](#).

Example Persistent Volume

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: block-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: Block ❶
  persistentVolumeReclaimPolicy: Retain
  fc:
    targetWWNs: ["50060e801049cfd1"]
    lun: 0
    readOnly: false

```

- ❶ **volumeMode** フィールドは、この PV が raw ブロックボリュームであることを示します。

Example Persistent Volume Claim

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block ❶
  resources:
    requests:
      storage: 10Gi

```

- ❶ **volumeMode** フィールドは、raw ブロック永続ボリュームが要求されていることを示します。

Example Pod Specification

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-with-block-volume
spec:
  containers:
    - name: fc-container
      image: fedora:26
      command: ["/bin/sh", "-c"]
      args: [ "tail -f /dev/null" ]
      volumeDevices: ❶
        - name: data
          devicePath: /dev/xvda ❷
  volumes:

```

```
- name: data
  persistentVolumeClaim:
    claimName: block-pvc 3
```

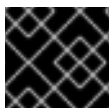
- 1** (**volumeMounts** と同様に) **volumeDevices** は、ブロックデバイスに使用され、**PersistentVolumeClaim** ソースでのみ使用できます。
- 2** (**mountPath** と同様に) **devicePath** は、物理デバイスへのパスを表します。
- 3** The volume source must be of type **persistentVolumeClaim** and should match the name of the PVC as expected.

表4.2 Accepted Values for VolumeMode

値	デフォルト
Filesystem	Yes
Block	No

表4.3 Binding Scenarios for Block Volumes

PV VolumeMode	PVC VolumeMode	バインディングの結果
Filesystem	Filesystem	バインド
Unspecified	Unspecified	バインド
Filesystem	Unspecified	バインド
Unspecified	Filesystem	バインド
Block	Block	バインド
Unspecified	Block	バインドなし
Block	Unspecified	バインドなし
Filesystem	Block	バインドなし
Block	Filesystem	バインドなし

**重要**

値を指定しないと、**Filesystem** のデフォルト値が指定されます。

表4.4 Status of Plug-ins That Support or Will Support Block Volumes

Plug-in	Support Block Volume
ファイバーチャネル	Merged OpenShift Container Platform 3.9
Ceph RBD	Merged OpenShift Container Platform 3.10
iSCSI	InProgress OpenShift Container Platform 3.10
AWS EBS	InProgress OpenShift Container Platform 3.10
GCE PD	InProgress OpenShift Container Platform 3.10
GlusterFS	InProgress OpenShift Container Platform 3.10

4.4. ソースコントロール管理

OpenShift Container Platform は、内部 (インハウス Git サーバーなど) または外部 ([GitHub](#)、[Bitbucket](#) など) でホストされている既存のソースコントロール管理 (SCM) システムを利用します。現時点で、OpenShift Container Platform は [Git](#) ソリューションのみをサポートします。

SCM 統合は [ビルド](#) に密接に関連し、以下の 2 つの点を実行します。

- Creating a **BuildConfig** using a repository, which allows building your application inside of OpenShift Container Platform. You can create a **BuildConfig** [manually](#) or let OpenShift Container Platform create it [automatically](#) by inspecting your repository.
- [Triggering a build](#) upon repository changes.

4.5. 受付コントローラー

4.5.1. 概要

受付制御プラグインはリソースの永続化の前にマスター API への要求をインターセプトしますが、要求の認証および承認後にこれを実行します。

クラスターに要求が受け入れられる前に、受付制御プラグインがそれぞれ、順番に実行されます。この順番に実行されているプラグインのいずれかが要求を拒否すると、要求全体がただちに拒否され、エンドユーザーにエラーが返されます。

受付制御プラグインは、システムで設定されたデフォルトを適用するために受信オブジェクトを変更する場合があります。さらに、受付制御プラグインはクォータ使用の増分などを実行する要求処理の一環として関連するリソースを変更する場合があります。



警告

OpenShift Container Platform マスターには、それぞれのタイプのリソース (Kubernetes および OpenShift Container Platform) についてデフォルトで有効にされているプラグインのデフォルトの一覧が含まれます。それらはマスターが適切に機能するために必要です。これらの一覧を変更することは、実際の変更内容を把握している場合でない限りは推奨されません。本製品の今後のバージョンでは異なるセットのプラグインを使用し、それらの順序を変更する可能性があります。マスター設定ファイルでプラグインのデフォルトの一覧を上書きする場合、新規バージョンの OpenShift Container Platform マスターの要件を反映できるように一覧を更新する必要があります。

4.5.2. 一般的な受付ルール

Starting in 3.3, OpenShift Container Platform uses a single admission chain for Kubernetes and OpenShift Container Platform resources. This changed from 3.2, and before where we had separate admission chains. This means that the top-level **admissionConfig.pluginConfig** element can now contain the admission plug-in configuration, which used to be contained in **kubernetesMasterConfig.admissionConfig.pluginConfig**.

kubernetesMasterConfig.admissionConfig.pluginConfig は **admissionConfig.pluginConfig** に移動し、マージされる必要があります。

Also, starting in 3.3, all the supported admission plug-ins are ordered in the single chain for you. You should no longer set **admissionConfig.pluginOrderOverride** or the **kubernetesMasterConfig.admissionConfig.pluginOrderOverride**. Instead, you should enable plug-ins that are off by default by either adding their plug-in-specific configuration, or adding a **DefaultAdmissionConfig** stanza like this:

```
admissionConfig:
  pluginConfig:
    AlwaysPullImages: ①
    configuration:
      kind: DefaultAdmissionConfig
      apiVersion: v1
      disable: false ②
```

- ① 受付プラグイン名です。
- ② プラグインを有効化する必要があることを示します。これはオプションで、ここでは参照としてのみ表示されます。

disable を **true** にすると、on にデフォルト設定される受付プラグインが無効になります。



警告

受付プラグインは、API サーバーのセキュリティーを実施するために一般的に使用されています。これらを無効にする場合には注意して行ってください。



注記

単一の受付チェーンに安全に組み込むことのできない **admissionConfig** 要素を使用していた場合は、API サーバーログで警告を受信し、API サーバーはレガシーの互換性のために 2 つの異なる受付チェーンで開始されることとなります。警告を解決するには、**admissionConfig** を更新します。

4.5.3. カスタマイズ可能な受付プラグイン

クラスター管理者は、一部の受付コントロールプラグインを、以下のような特定の動作を制御するように設定できます。

- [Limiting Number of Self-Provisioned Projects Per User](#)
- [Configuring Global Build Defaults and Overrides](#)
- [Controlling Pod Placement](#)
- [Managing Role Bindings](#)

4.5.4. コンテナを使用した受付コントローラー

コンテナを使用する受付コントローラーも [init コンテナ](#) をサポートします。

4.6. カスタム受付コントローラー

4.6.1. 概要

デフォルトの[受付コントローラー](#)のほかにも、[受付 Webhook](#) を受付チェーンの一部として使用できます。

受付 Webhook は Webhook サーバーを呼び出して、ラベルの挿入など、作成時に Pod を変更するか、または受付プロセス時に Pod 設定の特定の部分を検証します。

受付 Webhook はリソースの永続化の前にマスター API への要求をインターセプトしますが、要求の認証および承認後にこれを実行します。

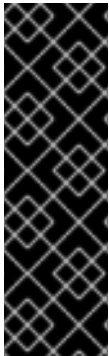
4.6.2. 受付 Webhook

OpenShift Container Platform では、API 受付チェーンで Webhook サーバーを呼び出す受付 Webhook オブジェクトを使用できます。

設定可能な 2 種類の受付 Webhook オブジェクトがあります。

- **変更用の受付 Webhook**は、変更用の Webhook を使用した、永続化する前のリソースコンテンツの変更を可能にします。
- **検証用の受付 Webhook** は、検証用の Webhook を使用したカスタム受付ポリシーの実施を可能にします。

Webhook および外部 Webhook サーバーの設定については本書では扱いません。ただし、Webhook サーバーは、OpenShift Container Platform で適切に機能するために、[インターフェース](#)に準拠する必要があります。



重要

受付 Webhook 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。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。

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。

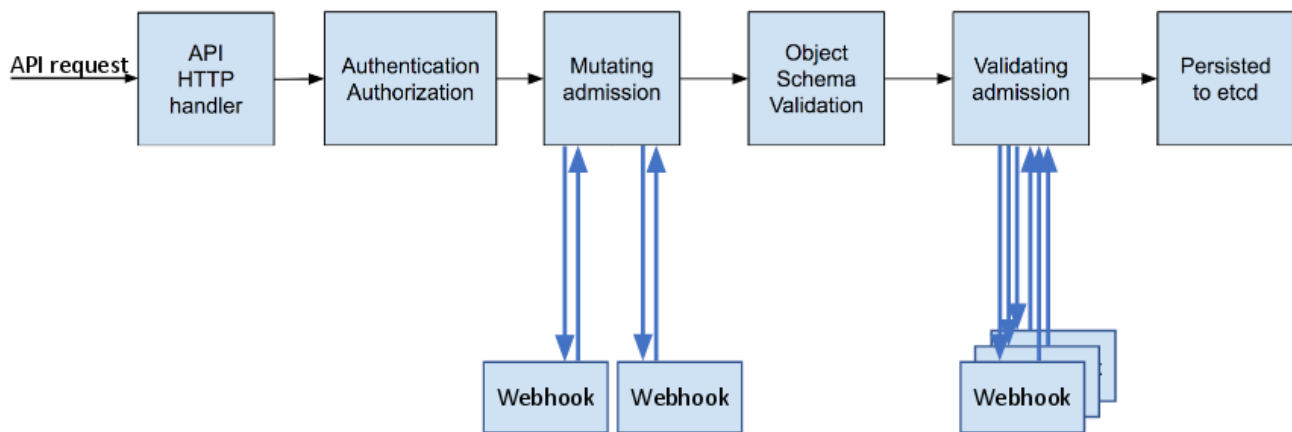
オブジェクトがインスタンス化されると、OpenShift Container Platform は API 呼び出しを実行してオブジェクトを許可します。受付プロセスでは、**変更用の受付コントローラー** は Webhook を呼び出して、アフィニティーラベルの挿入などのタスクを実行します。受付プロセスの終了時に、**検証用の受付コントローラー** は Webhook を呼び出し、アフィニティーラベルの検証などにより、オブジェクトが適切に設定されていることを確認します。検証にパスすると、OpenShift Container Platform はオブジェクトを設定済みとしてスケジュールします。

API 要求が送信されると、変更用または検証用の受付コントローラーは設定内の外部 Webhook の一覧を使用し、それらを並行して呼び出します。

- Webhook の **すべて** が要求を承認する場合、受付チェーンは継続します。
- Webhook の **いずれか** が要求を拒否する場合、受付要求は拒否され、これは、**初回**の webhook の拒否理由に基づいて実行されます。
複数の Webhook が受付要求を拒否する場合、最初のものだけがユーザーに返されます。
- If there is an error encountered when calling a webhook, that request is ignored and is be used to approve/deny the admission request.

The communication between the admission controller and the webhook server needs to be secured using TLS. Generate a CA certificate and use the certificate to sign the server certificate used by your webhook server. The PEM-formatted CA certificate is supplied to the admission controller using a mechanism, such as [Service Serving Certificate Secrets](#).

以下の図は、複数の Webhook を呼び出す 2 つの受付 Webhook を含むプロセスを示しています。



受付 Webhook の単純な使用事例として、リソースの構文検証が挙げられます。たとえば、すべての Pod に共通のラベルセットを指定する必要のあるインフラストラクチャーがあり、そのラベルが指定されていない Pod は永続化させないようにする場合に、Webhook を作成してこれらのラベルを挿入したり、別の Webhook でラベルの有無を検証したりすることができます。その後 OpenShift Container Platform は、ラベルがあり、検証をパスした Pod をスケジュールし、ラベルがないためにパスしない Pod を拒否します。

共通のユースケースには以下が含まれます。

- サイドカーコンテナを Pod に挿入するためのリソースの変更
- 一部のリソースをプロジェクトからブロックするためのプロジェクトの制限
- 依存するフィールドで複雑な検証を実行するためのカスタムリソース検証

4.6.2.1. 受付 Webhook のタイプ

クラスター管理者は、API サーバーの受付チェーンに **変更用の受付 Webhook** または **検証用の受付 Webhook** を含めることができます。

Mutating admission webhooks are invoked during the mutation phase of the admission process, which allows modification of the resource content before it is persisted. One example of a mutating admission webhook is the [Pod Node Selector](#) feature, which uses an annotation on a namespace to find a label selector and add it to the pod specification.

Sample mutating admission webhook configuration:

```

apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration ①
metadata:
  name: <controller_name> ②
webhooks:
- name: <webhook_name> ③
  clientConfig: ④
    service:
      namespace: ⑤
      name: ⑥
      path: <webhook_url> ⑦
    caBundle: <cert> ⑧
  
```



```

rules: 9
- operations: 10
  - <operation>
  apiGroups:
  - ""
  apiVersions:
  - ""
  resources:
  - <resource>
failurePolicy: <policy> 11

```

- 1 変更用の受付 Webhook 設定を指定します。
- 2 受付 Webhook オブジェクトの名前です。
- 3 呼び出す Webhook の名前です。
- 4 Webhook サーバーに接続し、これを信頼し、データをこれに送信する方法についての情報です。
- 5 フロントエンドサービスが作成されるプロジェクトです。
- 6 フロントエンドサービスの名前です。
- 7 受付要求に使用される Webhook URL です。
- 8 Webhook サーバーで使用されるサーバー証明書に署名する PEM でエンコーディングされた CA 証明書です。
- 9 API サーバーがこのコントローラーを使用するタイミングを定義するルールです。
- 10 このコントローラーを呼び出すために API サーバーをトリガーする操作です。
 - create
 - update
 - delete
 - connect
- 11 Webhook 受付サーバーが利用できない場合にポリシーを実行する方法を指定します。 **Ignore** (allow/fail open) または **Fail** (block/fail closed) になります。

検証用の受付 Webhook は受付プロセスの検証フェーズで起動します。このフェーズでは、特定 API リソースの変更がない項目の実施を可能にし、リソースが再び変更されないようにすることができます。Pod ノードセクターも、すべての **nodeSelector** フィールドがプロジェクトのノードセクターの制限で制約されていることを確認する、検証用の受付の例となります。

Sample validating admission webhook configuration:

```

apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration 1
metadata:
  name: <controller_name> 2
webhooks:

```

```

- name: <webhook_name> ③
  clientConfig: ④
    service:
      namespace: default ⑤
      name: kubernetes ⑥
      path: <webhook_url> ⑦
      caBundle: <cert> ⑧
  rules: ⑨
  - operations: ⑩
    - <operation>
    apiGroups:
      - ""
    apiVersions:
      - ""
    resources:
      - <resource>
  failurePolicy: <policy> ⑪

```

- ① 検証用の受付 Webhook 設定を指定します。
- ② Webhook 受付オブジェクトの名前です。
- ③ 呼び出す Webhook の名前です。
- ④ Webhook サーバーに接続し、これを信頼し、データをこれに送信する方法についての情報です。
- ⑤ フロントエンドサービスが作成されるプロジェクトです。
- ⑥ フロントエンドサービスの名前です。
- ⑦ 受付要求に使用される Webhook URL です。
- ⑧ Webhook サーバーで使用されるサーバー証明書に署名する PEM でエンコーディングされた CA 証明書です。
- ⑨ API サーバーがこのコントローラーを使用するタイミングを定義するルールです。
- ⑩ このコントローラーを呼び出すために API サーバーをトリガーする操作です。
 - create
 - update
 - delete
 - connect
- ⑪ Webhook 受付サーバーが利用できない場合にポリシーを実行する方法を指定します。 **Ignore** (allow/fail open) または **Fail** (block/fail closed) になります。



注記

Fail open の場合に、すべてのクライアントの予測できない動作が生じる可能性があります。

4.6.2.2. 受付 Webhook を作成します。

最初に外部 Webhook サーバーをデプロイし、これが適切に機能することを確認します。これを実行しない場合、Webhook が **fail open** または **fail closed** として設定されているかに応じて、操作は無条件に許可または拒否されます。

1. YAML ファイルを使用して**変更用**、または**検証用**受付 Webhook オブジェクトを設定します。
2. 以下のコマンドを実行してオブジェクトを作成します。

```
oc create -f <file-name>.yaml
```

受付 Webhook オブジェクトの作成後、OpenShift Container Platform が新規設定を反映するまでに数秒の時間がかかります。

3. 受付 Webhook のフロントエンドサービスを作成します。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    role: webhook 1
  name: <name>
spec:
  selector:
    role: webhook 2
```

1 **2** Webhook をトリガーするための自由形式のラベルです。

4. 以下のコマンドを実行してオブジェクトを作成します。

```
oc create -f <file-name>.yaml
```

5. Webhook で制御する必要のある Pod に受付 Webhook 名を追加します。

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    role: webhook 1
  name: <name>
spec:
  containers:
    - name: <name>
      image: myrepo/myimage:latest
      imagePullPolicy: <policy>
      ports:
        - containerPort: 8000
```

1 Webhook をトリガーするためのラベルです。



注記

独自のセキュアでポータブルな Webhook 受付サーバーをビルドする方法についてのエンドツーエンドの例については、[kubernetes-namespace-reservation プロジェクト](#)を参照し、ライブラリーについては [generic-admission-apiserver](#) を参照してください。

4.6.2.3. 受付 Webhook オブジェクトのサンプル

以下は、[namespace](#) が予約される場合に [namespace](#) の作成 を許可しない受付 Webhook のサンプルです。

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
metadata:
  name: namespace-reservations.admission.online.openshift.io
webhooks:
- name: namespace-reservations.admission.online.openshift.io
  clientConfig:
    service:
      namespace: default
      name: webhooks
      path: /apis/admission.online.openshift.io/v1beta1/namespace-reservations
      caBundle: KUBE_CA_HERE
    rules:
    - operations:
      - CREATE
      apiGroups:
      - ""
      apiVersions:
      - "b1"
      resources:
      - namespaces
  failurePolicy: Ignore
```

以下は、[webhook](#) という名前の受付 Webhook によって評価される Pod のサンプルです。

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    role: webhook
  name: webhook
spec:
  containers:
  - name: webhook
    image: myrepo/myimage:latest
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 8000
```

以下は Webhook のフロントエンドサービスです。

```
apiVersion: v1
kind: Service
metadata:
```

```
labels:  
  role: webhook  
  name: webhook  
spec:  
  ports:  
    - port: 443  
      targetPort: 8000  
  selector:  
    role: webhook
```

4.7. 他の API オブジェクト

4.7.1. LimitRange

制限範囲は、Kubernetes [namespace](#) のリソースに設定される最小/最大の制限を実施するメカニズムを提供します。

制限範囲を namespace に追加することで、個別の Pod またはコンテナによる CPU およびメモリーの最小および最大使用量を適用できます。

4.7.2. ResourceQuota

Kubernetes は、[namespace](#) で作成されるオブジェクト数と、namespace 内のオブジェクト間で要求されるリソース合計量の両方を制限できます。これにより、namespace 内の複数のチームで単一の Kubernetes クラスターを共有でき、あるチームによって別のチームがクラスターリソース不足になることを防ぐことができます。

See [Cluster Administration](#) for more information on **ResourceQuota**.

4.7.3. リソース

Kubernetes の **Resource** は、Pod またはコンテナによって要求され、割り当てられ、消費されるものです。例として、メモリー (RAM)、CPU、ディスク時間、およびネットワーク帯域幅があります。

See the [Developer Guide](#) for more information.

4.7.4. Secret

[Secrets](#) are storage for sensitive information, such as keys, passwords, and certificates. They are accessible by the intended pod(s), but held separately from their definitions.

4.7.5. PersistentVolume

A [persistent volume](#) is an object (**PersistentVolume**) in the infrastructure provisioned by the cluster administrator. Persistent volumes provide durable storage for stateful applications.

4.7.6. PersistentVolumeClaim

A **PersistentVolumeClaim** object is a [request for storage by a pod author](#). Kubernetes matches the claim against the pool of available volumes and binds them together. The claim is then used as a volume by a pod. Kubernetes makes sure the volume is available on the same node as the pod that requires it.

4.7.6.1. カスタムリソース

カスタムリソースは、API を拡張するか、独自の API をプロジェクトまたはクラスターに導入できるようにする Kubernetes API の拡張です。

See link:https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/cluster_administration/#admin-guide-custom-resources[Extend the Kubernetes API with Custom Resources].

4.7.7. OAuth オブジェクト

4.7.7.1. OAuthClient

OAuthClient は、[RFC 6749, section 2](#) に説明されているように OAuth クライアントを表します。

以下の **OAuthClient** オブジェクトは自動的に作成されます。

openshift-web-console	Web コンソールのトークンを要求するために使用されるクライアント
openshift-browser-client	対話式ログインを処理できるユーザーエージェントで /oauth/token/request でトークンを要求するために使用されるクライアント
openshift-challenging-client	WWW-Authenticate チャレンジを処理できるユーザーエージェントでトークンを要求するために使用されるクライアント

OAuthClient オブジェクト定義

```
kind: "OAuthClient"
accessTokenMaxAgeSeconds: null ❶
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "openshift-web-console" ❷
  selflink: "/oapi/v1/oauthClients/openshift-web-console"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01Z"
respondWithChallenges: false ❸
secret: "45e27750-a8aa-11e4-b2ea-3c970e4b7ffe" ❹
redirectURIs:
  - "https://localhost:8443" ❺
```

- ❶ アクセストークンの有効期間 (秒単位)(以下の説明を参照してください)。
- ❷ **name** は OAuth 要求の **client_id** パラメーターとして使用されます。
- ❸ **respondWithChallenges** が **true** に設定される場合、/oauth/authorize への認証されていない要求は、設定される認証方法でサポートされている場合には **WWW-Authenticate** チャレンジを生じさせます。
- ❹ **secret** パラメーターの値は、承認コードフローの **client_secret** パラメーターとして使用されます。

- 5 絶対 URI は、**redirectURIs** セクションに1つ以上配置できます。承認要求と共に送信される **redirect_uri** パラメーターには、指定の **redirectURIs** のいずれかをプレフィックスとして付加する必要があります。

accessTokenMaxAgeSeconds 値は、個別の OAuth クライアントのマスター設定に指定されている **accessTokenMaxAgeSeconds** 値を上書きします。この値をクライアントに設定すると、他のクライアントの有効期間に影響を与えることなく、クライアントのアクセストークンの有効期間を長く設定できます。

- **null** の場合、マスター設定ファイルのデフォルト値が使用されます。
- **0** に設定される場合、トークンは有効期限切れになりません。
- **0** よりも大きな値に設定される場合、クライアント用に発行されるトークンには指定された有効期限が設定されます。たとえば、**accessTokenMaxAgeSeconds: 172800** により、トークンは発行後 48 時間後に有効期限切れになります。

4.7.7.2. OAuthClientAuthorization

OAuthClientAuthorization は、特定の **OAuthClient** に特定のスコープが設定された **OAuthAccessToken** が付与されることについての **User** による承認を表します。

OAuthClientAuthorization オブジェクトの作成は、**OAuth** サーバーへの承認要求時に実行されます。

OAuthClientAuthorization オブジェクト定義

```
kind: "OAuthClientAuthorization"
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "bob:openshift-web-console"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
  clientName: "openshift-web-console"
  userName: "bob"
  userID: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe"
  scopes: []
```

4.7.7.3. OAuthAuthorizeToken

OAuthAuthorizeToken は、[RFC 6749, section 1.3.1](#) に説明されているように **OAuth** 承認コードを表します。

OAuthAuthorizeToken は、[RFC 6749, section 4.1.1](#) で説明されているように `/oauth/authorize` エンドポイントへの要求によって作成されます。

OAuthAuthorizeToken は次に、[RFC 6749, section 4.1.3](#) に説明されているように、`/oauth/token` エンドポイントへの要求で **OAuthAccessToken** を取得するために使用できます。

OAuthAuthorizeToken オブジェクト定義

```
kind: "OAuthAuthorizeToken"
apiVersion: "oauth.openshift.io/v1"
metadata:
```

```

name: "MDAwYjM5YjMtMzM1MC00NDY4LTkxODItOTA2OTE2YzE0M2Fj" ❶
resourceVersion: "1"
creationTimestamp: "2015-01-01T01:01:00:00"
clientName: "openshift-web-console" ❷
expiresIn: 300 ❸
scopes: []
redirectURI: "https://localhost:8443/console/oauth" ❹
userName: "bob" ❺
userID: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" ❻

```

- ❶ **name** は、OAuthAccessToken を交換するために承認コードとして使用されるトークン名を表します。
- ❷ **clientName** 値は、このトークンを要求した OAuthClient です。
- ❸ **expiresIn** 値は creationTimestamp の有効期限 (秒単位) です。
- ❹ **redirectURI** 値は、このトークンが作成された承認フローでユーザーがリダイレクトされた場所です。
- ❺ **userName** represents the name of the User this token allows obtaining an OAuthAccessToken for.
- ❻ **userID** represents the UID of the User this token allows obtaining an OAuthAccessToken for.

4.7.7.4. OAuthAccessToken

OAuthAccessToken は、RFC 6749, section 1.4 に説明されているように、**OAuth** アクセストークンを表します。

OAuthAccessToken は、RFC 6749, section 4.1.3 に説明されているように、/oauth/token エンドポイントへの要求によって作成されます。

アクセストークンは、API に対して認証を行うためにベアラートークンとして使用されます。

OAuthAccessToken オブジェクト定義

```

kind: "OAuthAccessToken"
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "ODliOGE5ZmMtYzczYi00Nzk1LTg4MGEtNzQyZmUxZmUwY2Vh" ❶
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:02:00:00"
  clientName: "openshift-web-console" ❷
  expiresIn: 86400 ❸
  scopes: []
  redirectURI: "https://localhost:8443/console/oauth" ❹
  userName: "bob" ❺
  userID: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" ❻
  authorizeToken: "MDAwYjM5YjMtMzM1MC00NDY4LTkxODItOTA2OTE2YzE0M2Fj" ❼

```

- ❶ **name** は、API に対して認証を行うためにベアラートークンとして使用されるトークン名です。
- ❷ **clientName** 値は、このトークンを要求した OAuthClient です。

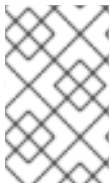
- 3 **expiresIn** 値は `creationTimestamp` の有効期限 (秒単位) です。
- 4 **redirectURI** は、このトークンが作成された承認フローでユーザーがリダイレクトされた場所です。
- 5 **userName** represents the User this token allows authentication as.
- 6 **userUID** represents the User this token allows authentication as.
- 7 **authorizeToken** は、このトークンを取得するために使用される `OAuthAuthorizationToken` の名前です (ある場合)。

4.7.8. ユーザーオブジェクト

4.7.8.1. Identity

When a user logs into OpenShift Container Platform, they do so using a configured [identity provider](#). This determines the user's identity, and provides that information to OpenShift Container Platform.

次に OpenShift Container Platform は **UserIdentityMapping** でその **Identity** を検索します。



注記

If the identity provider is configured with the **lookup** mapping method, for example, if you are using an external LDAP system, this automatic mapping is not performed. You must create the mapping manually. For more information, see [Lookup Mapping Method](#).

- **Identity** がすでに存在する場合でも、これが **User** にマップされていないと、ログインは失敗します。
- **Identity** がすでに存在し、これが **User** にマップされている場合、ユーザーには、マップされた **User** の **OAuthAccessToken** が付与されます。
- **Identity** が存在しない場合、**Identity**、**User**、および **UserIdentityMapping** が作成され、ユーザーには、マップされた **User** の **OAuthAccessToken** が付与されます。

Identity オブジェクト定義

```
kind: "Identity"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "anypassword:bob" ①
  uid: "9316ebad-0fde-11e5-97a1-3c970e4b7ffe"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
  providerName: "anypassword" ②
  providerUserName: "bob" ③
user:
  name: "bob" ④
  uid: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" ⑤
```

- ① アイデンティティ名は `providerName:providerUserName` の形式である必要があります。

- 2 **providerName** はアイデンティティプロバイダーの名前です。
- 3 **providerUserName** は、アイデンティティプロバイダーのスコープでこのアイデンティティを一意に表す名前です。
- 4 **user** パラメーターの **name** は、このアイデンティティがマップされるユーザーの名前です。
- 5 **uid** は、このアイデンティティがマップされるユーザーの UID を表します。

4.7.8.2. ユーザー

A **User** represents an actor in the system. Users are granted permissions by [adding roles to users or to their groups](#).

ユーザーオブジェクトは初回ログイン時に自動的に作成されるか、API で作成できます。



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。

User オブジェクト定義

```
kind: "User"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "bob" 1
  uid: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
identities:
  - "anypassword:bob" 2
fullName: "Bob User" 3
```

- 1 **name** は、ロールをユーザーに追加する際に使用されるユーザー名です。
- 2 **identities** の値は、このユーザーにマップされるアイデンティティオブジェクトです。これはログインできないユーザーについて **null** または空にすることができます。
- 3 **fullName** 値は、ユーザーのオプションの表示名です。

4.7.8.3. UserIdentityMapping

UserIdentityMapping は **Identity** を **User** にマップします。

UserIdentityMapping を作成し、更新し、または削除することにより、**Identity** および **User** オブジェクトの対応するフィールドが変更されます。

Identity は単一の **User** にのみマップされるため、特定のアイデンティティとしてログインすると、**User** が明確に判別されます。

User には複数のアイデンティティをマップできます。これにより、複数のログイン方法で同じ **User** を識別できます。

UserIdentityMapping オブジェクト定義

```
kind: "UserIdentityMapping"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "anypassword:bob" ❶
  uid: "9316ebad-0fde-11e5-97a1-3c970e4b7ffe"
  resourceVersion: "1"
identity:
  name: "anypassword:bob"
  uid: "9316ebad-0fde-11e5-97a1-3c970e4b7ffe"
user:
  name: "bob"
  uid: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe"
```

❶ **UserIdentityMapping** 名は、マップされた **Identity** 名に一致します。

4.7.8.4. グループ

A **Group** represents a list of users in the system. Groups are granted permissions by [adding roles to users or to their groups](#).

Group オブジェクト定義

```
kind: "Group"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "developers" ❶
  creationTimestamp: "2015-01-01T01:01:01-00:00"
users:
  - "bob" ❷
```

❶ **name** は、ロールをグループに追加する際のグループ名です。

❷ **users** の値は、このグループのメンバーであるユーザーオブジェクトの名前です。

第5章 ネットワーク

5.1. ネットワーク

5.1.1. 概要

Kubernetes は、確実に Pod 間がネットワークで接続されるようにし、内部ネットワークから IP アドレスを各 Pod に割り当てます。こうすることで、Pod 内の全コンテナが同じホスト上にあるかのように動作します。各 Pod に IP アドレスを割り当てると、ポートの割り当て、ネットワーク、名前の指定、サービス検出、負荷分散、アプリケーション設定、移行などの点で、Pod を物理ホストや仮想マシンのように扱うことができます。

Creating links between pods is unnecessary, and it is not recommended that your pods talk to one another directly using the IP address. Instead, it is recommend that you create a [service](#), then interact with the service.

5.1.2. OpenShift Container Platform DNS

フロントエンドサービスやバックエンドサービスなど、複数のサービスを実行して複数の Pod で使用している場合には、フロントエンド Pod がバックエンドサービスと通信できるように、ユーザー名、サービス IP などの環境変数を作成します。サービスが削除され、再作成された場合には、新規の IP アドレスがサービスに割り当てられるので、サービス IP の環境変数の更新値を取得するには、フロントエンド Pod を再作成する必要があります。さらに、バックエンドサービスは、フロントエンド Pod を作成する前に作成し、サービス IP が正しく生成され、フロントエンド Pod に環境変数として提供できるようにする必要があります。

このような理由から、サービスの DNS やサービスの IP/ポートがサービスに到達できるように、OpenShift Container Platform には DNS が含まれています。OpenShift Container Platform は、サービスの DNS クエリーに応答するマスターで [SkyDNS](#) を実行することで、スプリット DNS をサポートします。マスターは、デフォルトで、ポート 53 をリッスンします。

ノードが起動すると、以下のメッセージで、Kubelet が正しくマスターに解決されていることが分かります。

```
0308 19:51:03.118430 4484 node.go:197] Started Kubelet for node
openshiftdev.local, server at 0.0.0.0:10250
10308 19:51:03.118459 4484 node.go:199] Kubelet is setting 10.0.2.15 as a
DNS nameserver for domain "local"
```

2 番目のメッセージが表示されない場合は、Kubernetes サービスが利用できない可能性があります。

ノードホストで、各コンテナのネームサーバーのフロントにマスター名が追加され、コンテナの検索ドメインはデフォルトでは、`<pod_namespace>.cluster.local` になります。コンテナは、ノード上の他のネームサーバーよりも先にネームサーバーのクエリーをマスターに転送します。これは、Docker 形式のコンテナではデフォルトの動作です。マスターは、以下の形式の `.cluster.local` ドメインでクエリーに対応します

表5.1 DNS 名の例

オブジェクトタイプ	例
デフォルト	<code><pod_namespace>.cluster.local</code>

オブジェクトタイプ	例
Services (サービス)	<service>.<pod_namespace>.svc.cluster.local
Endpoints (エンドポイント)	<name>.<namespace>.endpoints.cluster.local

これにより、新しいサービスを取得するためにフロントエンドの pod を再起動し、サービスに対して新しい IP を作成せずに済みます。また、pod がサービスの DNS を使用するので、環境変数を使用する必要がなくなります。さらに、DNS は変更しないので、設定ファイルで **db.local** としてデータベースサービスを参照できます。また、検索はサービス IP に対して解決するため、ワイルドカードの検索もサポートされます。さらにサービス名 (つまり DNS) が事前に確立しているので、フロントエンド Pod の前にバックエンドサービスを作成する必要がなくなります。

この DNS 構造では、ポータル IP はサービスに割り当てられず、kube-proxy は負荷分散しないまたはエンドポイントのルーティングを提供するヘッドレスサービスに対応しています。サービス DNS は依然として使用でき、サービスの Pod ごとに1つずつある複数のレコードに対応し、クライアントによる Pod 間のラウンドロビンを可能にします。

5.2. OPENSIFT SDN

5.2.1. 概要

OpenShift Container Platform は、Software Defined Networking (SDN) アプローチを使用して、クラスターのネットワークを統合し、OpenShift Container Platform クラスターの Pod 間の通信を可能にします。OpenShift SDN により、このような Pod ネットワークが確立され、メンテナンスされます。OpenShift SDN は Open vSwitch (OVS) を使用してオーバーレイネットワークを設定します。

OpenShift SDN では以下のように、Pod ネットワークを構成するための SDN プラグインを 3 つ提供します。

- **ovs-subnet** プラグインはオリジナルのプラグインで、Pod が他の Pod やサービスすべてと通信できる「フラットな」Pod ネットワークを提供します。
- **ovs-multitenant** プラグインは、pod とサービスをプロジェクトレベルと分離します。プロジェクトごとに、一意の Virtual Network ID (VNID) を受け取り、プロジェクトに割り当てられた Pod からのトラフィックを特定します。別のプロジェクトからの Pod は、別のプロジェクトの Pod やサービスからパケットの送信や受信ができません。ただし、VNID 0 を受け取るプロジェクトは、他の Pod すべてとの間で通信できるという面で、追加の特権があります。OpenShift Container Platform クラスターでは、**default** プロジェクトに VNID 0 が割り当てられています。これにより、ロードバランサーなど、特定のサービスがクラスター内の他の全 Pod との間でスムーズに通信できるようにします。
- **ovs-networkpolicy** プラグインでは、プロジェクト管理者が NetworkPolicy オブジェクトを使用して分離ポリシーを設定できます。



注記

Information on configuring the SDN on masters and nodes is available in [Configuring the SDN](#).

5.2.2. マスター上の設計

OpenShift Container Platform master では、OpenShift SDN が、`etcd` に保存されている、ノードのレジストリーを管理します。システム管理者がノードを登録すると、OpenShift SDN がクラスターネットワークから未使用のサブネットを割り当てて、レジストリーのこのサブネットを保存します。ノードが削除されると、OpenShift SDN はレジストリーからサブネットを削除し、このサブネットを割り当て可能とみなします。

デフォルトの設定では、クラスターネットワークは `10.128.0.0/14` ネットワーク (つまり `10.128.0.0 - 10.131.255.255`) で、ノードには `/23` サブネット (つまり `10.128.0.0/23`、`10.128.2.0/23`、`10.128.4.0/23` など) が割り当てられます。つまり、このクラスターネットワークには、512 個のサブネットをノードに割り当てることができ、特定のノードには 510 個のアドレスが割り当てられ、このノードで実行中のコンテナに割り当てることができます。クラスターネットワークのサイズやアドレス範囲、さらにホストのサブネットサイズは、設定可能です。



注記

サブネットが次に大きい octet に拡張される場合には、共有の octet でサブネットのビットが 0 のものが先に割り当てられます。たとえば、ネットワークが `10.1.0.0/16` で `hostsubnetlength=6` が指定されている場合には、`10.1.0.0/26` および `10.1.1.0/26` から `10.1.255.0/26` が `10.1.0.64/26`、`10.1.1.64/26` が埋まる前に、割り当てられます。こうすることで、サブネットを把握しやすくなります。

マスター上の OpenShift SDN では、ローカル (マスター) ホストが、クラスターネットワークにアクセスできるように設定されないため、マスターホストは、ノードとして実行されない限り、クラスターネットワーク経由で Pod にアクセスできません。

`ovs-multitenant` プラグインを使用する場合には、OpenShift SDN マスターはプロジェクトの作成や削除を監視し、VXLAN VNID をプロジェクトに割り当てます。この VNID は後で、ノードが正しくトラフィックを分離するために使用します。

5.2.3. ノード上の設計

ノードでは OpenShift SDN は先に、前述のレジストリーに、SDN マスターを持つローカルホストを登録し、マスターがノードにサブネットを割り当てられるようにします。

次に OpenShift SDN は、3 つのネットワークデバイスを作成、設定します。

- **br0**: Pod コンテナが割り当てられる OVS ブリッジデバイスです。OpenShift SDN は、このブリッジにサブネット以外のフロールールも設定します。
- **tun0**: OVS の内部ポート (**br0** のポート 2) です。これには、クラスターサブネットゲートウェイアドレスが割り当てられ、外部のネットワークアクセスに使用されます。OpenShift SDN はクラスターサブネットから外部ネットワークに NAT 経由でアクセスできるように、`netfilter` およびルーティングルールを設定します。
- **vxlan_sys_4789**: OVS VXLAN デバイス (**br0** のポート 1) です。これはリモートノードのコンテナへのアクセスを提供します。OVS ルールでは **vxlan0** として参照されます。

Pod がホストで起動されるたびに、OpenShift SDN は以下を行います。

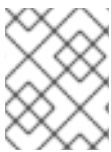
1. 対象の Pod に、ノードのクラスターサブネットから、空いている IP アドレスを割り当てます。
2. ホスト側の Pod の veth インターフェースペアを OVS ブリッジ **br0** に割り当てます。
3. OpenFlow ルールを OVS データベースに追加して、新規の Pod にアドレス指定されたトラフィックを正しい OVS ポートにルーティングします。

4. **ovs-multitenant** プラグインの場合は、Pod からのトラフィックには、その Pod の VNID をタグ付けし、トラフィックの VNID が Pod の VNID (または特権のある VNID 0) と一致する場合にはその Pod にトラフィックを許可するという OpenFlow ルールを追加します。一致しないトラフィックは、一般的なルールで除外されます。

OpenShift SDN nodes also watch for subnet updates from the SDN master. When a new subnet is added, the node adds OpenFlow rules on **br0** so that packets with a destination IP address the remote subnet go to **vxlان0** (port 1 on **br0**) and thus out onto the network. The **ovs-subnet** plug-in sends all packets across the VXLAN with VNID 0, but the **ovs-multitenant** plug-in uses the appropriate VNID for the source container.

5.2.4. パケットフロー

A と B の 2 つのコンテナがあり、コンテナ A の **eth0** をベースにするピア仮想 Ethernet デバイスの名前が **vethA**、コンテナ B の **eth0** のピア名が **vethB** とします。



注記

Docker サービスが使用するピアの仮想 Ethernet デバイスに慣れていない場合は、[Docker の高度なネットワークに関するドキュメント](#)を参照してください。

まず、コンテナ A がローカルホストにあり、コンテナ B もローカルホストにあると仮定します。コンテナ A からコンテナ B のパケットフローは以下のようになります。

eth0 (A の netns) → vethA → br0 → vethB → eth0 (B の netns)

次に、コンテナ A がローカルホストに、コンテナ B がクラスターネットワーク上のリモートホストにあると想定します。その場合には、コンテナ A からコンテナ B のパケットフローは以下のようになります。

eth0 (A の netns) → vethA → br0 → vxlan0 → ネットワーク^[1] → vxlan0 → br0 → vethB → eth0 (B の netns)

最後に、コンテナ A が外部ホストに接続すると、トラフィックは以下のようになります。

eth0 (A の netns) → vethA → br0 → tun0 → (NAT) → eth0 (物理デバイス) → インターネット

パケット配信の意思決定はほぼ OVS ブリッジ **br0** の OpenFlow ルールを基に行われ、プラグインのネットワークアーキテクチャを簡素化し、ルーティングを柔軟化します。**ovs-multitenant** プラグインの場合は、OpenFlow ルールを元にした意思決定により、強制的なネットワーク分離が可能になります。

5.2.5. ネットワーク分離

ovs-multitenant プラグインを使用して、ネットワーク分離を実現できます。デフォルト以外のプロジェクトに割り当てられた Pod からパケットが送信される場合は、OVS ブリッジ **br0** により、このパケットに、プロジェクトが割り当てた VNID のタグを付けます。パケットが、ノードのクラスターサブネットに含まれる別の IP アドレスに転送される場合には、OVS ブリッジは、VNID が一致する場合にのみ、宛先の Pod に対するこのパケットの配信を許可します。

パケットが別のノードから VXLAN トンネル経由で受信された場合には、トンネル ID を VNID として使用し、OVS ブリッジは、トンネル ID が宛先の Pod の VNID に一致する場合にのみ、ローカル Pod へのパケットの配信を許可します。

他のクラスターサブネットが宛先のパケットは、その VNID でタグ付けされ、クラスターサブネットを所有するノードのトンネルの宛先アドレスが指定された VXLAN トンネルに配信されます。

As described before, VNID 0 is privileged in that traffic with any VNID is allowed to enter any pod assigned VNID 0, and traffic with VNID 0 is allowed to enter any pod. Only the **default** OpenShift Container Platform project is assigned VNID 0; all other projects are assigned unique, isolation-enabled VNIDs. Cluster administrators can optionally [control the pod network](#) for the project using the administrator CLI.

5.3. 利用可能な SDN プラグイン

OpenShift Container Platform は、OpenShift Container Platform と Kubernetes の間のインターフェースとして、Kubernetes [Container Network Interface \(CNI\)](#) をサポートします。Software Defined Network (SDN) プラグインを使用することで、ネットワーク機能がユーザーのネットワークのニーズに対応します。必要に応じて、CNI インターフェースをサポートするプラグインをさらに追加できます。

5.3.1. OpenShift SDN

OpenShift SDN は、デフォルトで Ansible ベースのインストール手順の一部としてインストール設定されます。詳細情報は、「[OpenShift SDN](#)」のセクションを参照してください。

5.3.2. サードパーティーの SDN プラグイン

5.3.2.1. Flannel SDN

flannel は、コンテナ専用に設計された仮想ネットワーク層です。OpenShift Container Platform は、デフォルトの Software-Defined Networking (SDN) コンポーネントの代わりに、ネットワークコンテナとして **flannel** を使用できます。これは、OpenStack など、SDN にも依存するクラウドプロバイダープロット内で OpenShift Container Platform を実行している場合や、両プラットフォームを通してパケットを 2 回カプセル化しなくても良いようにする場合に便利です。

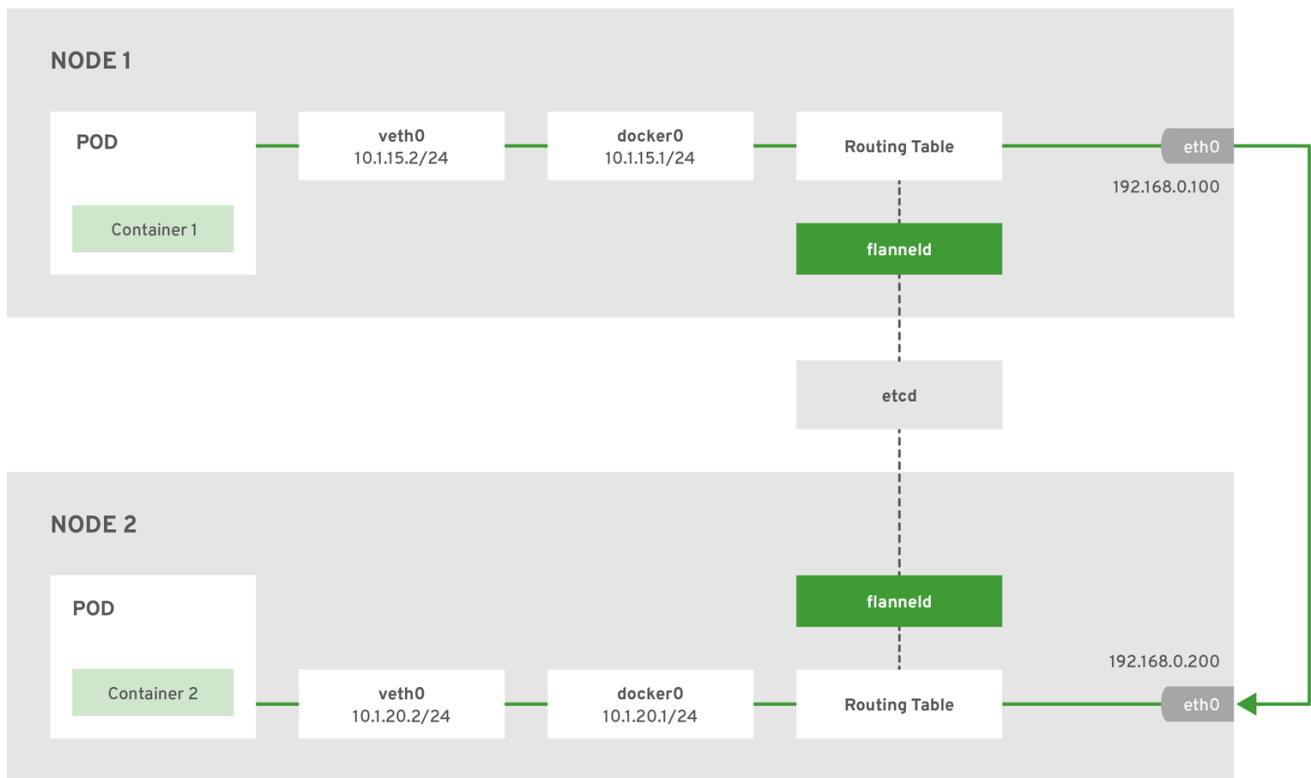
アーキテクチャー

OpenShift Container Platform は、**flannel** を **host-gw** モードで実行し、コンテナ間のルートをマッピングします。ネットワーク内の各ホストは、**flanneld** と呼ばれるエージェントを実行します。このエージェントは以下を行います。

- ホストごとに一意のサブネットを管理する
- ホスト上の各コンテナに IP アドレスを割り当てる
- 別のホスト上であっても、コンテナ間のルートをマッピングする

各 **flanneld** エージェントは、この情報を中央の **etcd** ストアに提供し、ホスト上の他のエージェントがパケットを、**flannel** ネットワーク内の他のコンテナにルーティングできるようにします。

以下の図は、**flannel** ネットワークを使用したコンテナ間のアーキテクチャーおよびデータフローを示します。



OPENSIFT_415489_0218

ノード 1 には以下のルートが含まれます。

```
default via 192.168.0.100 dev eth0 proto static metric 100
10.1.15.0/24 dev docker0 proto kernel scope link src 10.1.15.1
10.1.20.0/24 via 192.168.0.200 dev eth0
```

ノード 2 には以下のルートが含まれます。

```
default via 192.168.0.200 dev eth0 proto static metric 100
10.1.20.0/24 dev docker0 proto kernel scope link src 10.1.20.1
10.1.15.0/24 via 192.168.0.100 dev eth0
```

5.3.2.2. Nuage SDN

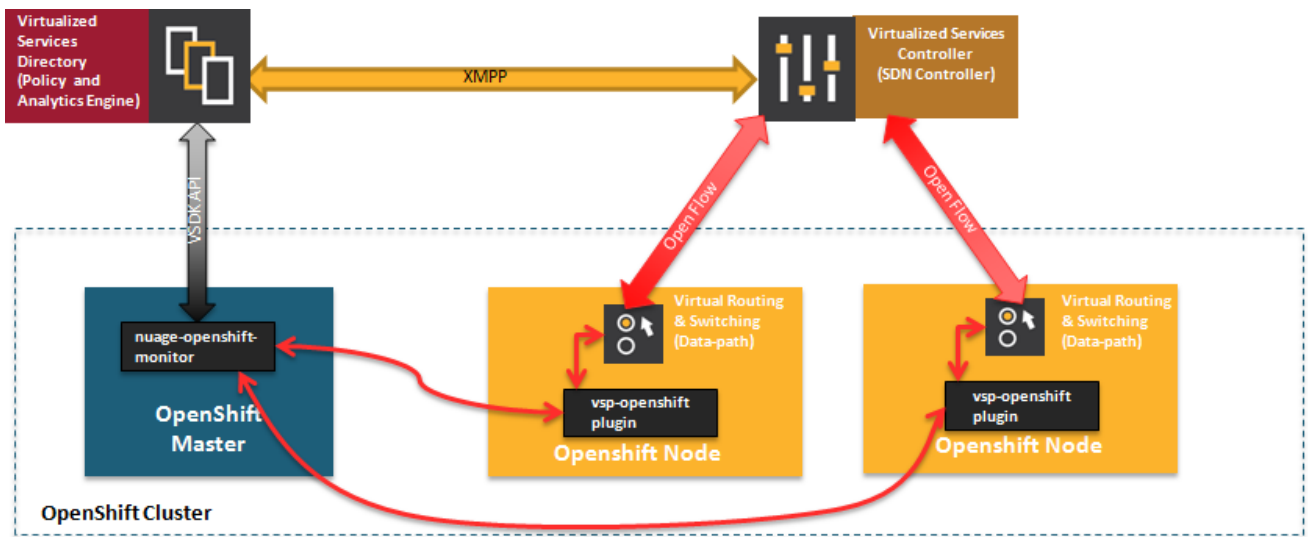
[Nuage Networks'](#) SDN solution delivers highly scalable, policy-based overlay networking for pods in an OpenShift Container Platform cluster. Nuage SDN can be installed and configured as a part of the Ansible-based installation procedure. See the [Advanced Installation](#) section for information on how to install and deploy OpenShift Container Platform with Nuage SDN.

[Nuage Networks](#) は、Virtualized Services Platform (VSP) と呼ばれる、スケーラビリティの高い、ポリシーベースの SDN プラットフォームを提供します。Nuage VSP は、データプレーン用にオープンソースの Open vSwitch とともに、SDN Controller を使用します。

Nuage は、オーバーレイを使用して、OpenShift Container Platform と VM およびベアメタルサーバーからなる他の環境の間をポリシーベースで接続できるようにします。プラットフォームのリアルタイムアナリティクスエンジンでは、OpenShift Container Platform アプリケーションの可視化およびセキュリティ監視を実現します。

Nuage VSP は OpenShift Container Platform と統合し、DevOps チームが直面するネットワークのラグを取り除くことで、ビジネスアプリケーションがすばやく起動と更新ができるようにします。

図5.1 Nuage VSP と OpenShift Container Platform との統合



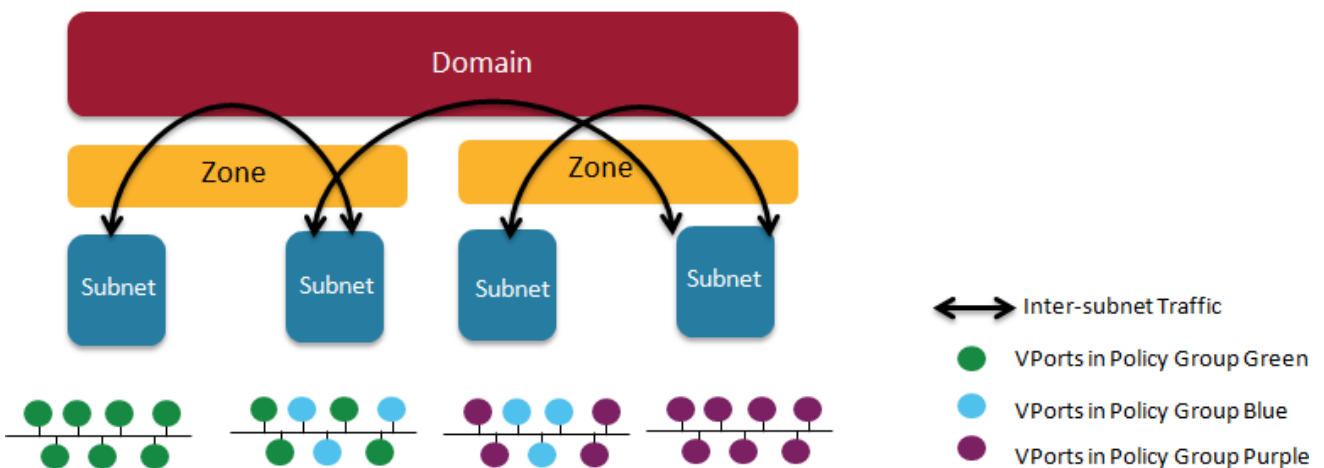
統合を行う固有のコンポーネントが2つあります。

1. **nuage-openshift-monitor** サービス。OpenShift Container Platform マスターノードで個別のサービスとして実行されます。
2. **vsp-openshift** プラグイン。クラスターの各ノードで OpenShift Container Platform ランタイムにより呼び出されます。

Nuage Virtual Routing and Switching ソフトウェア (VRS) は、オープンソースの Open vSwitch をベースにしており、データパス転送を行います。VRS は各ノードで実行され、コントローラーからポリシー設定を取得します。

Nuage VSP の用語

図5.2 Nuage VSP のビルディングブロック



1. **ドメイン**: 組織には1つまたは複数のドメインが含まれます。ドメインは単一の「レイヤー3」の領域を指します。標準のネットワーク用語では、ドメインは、VRF インスタンスと同じ位置づけです。
2. **ゾーン**: ゾーンは、ドメインの配下に定義されます。ゾーンは、直接ネットワーク上のなにかにマッピングされるわけではなく、ゾーンの全エンドポイントが同じポリシーセットに準拠するなど、ポリシーが関連付けられているオブジェクトとして機能します。
3. **サブネット**: サブネットはゾーンの配下に定義されます。サブネットは、ドメインインスタンス内の固有のレイヤー2サブネットを指します。サブネットは、ドメイン内で一意で他とは異なる

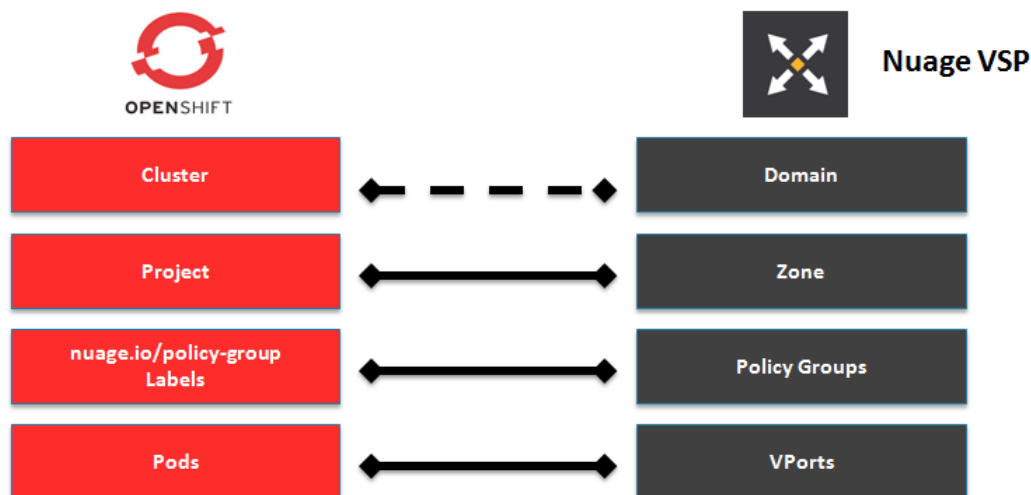
ります。つまり、ドメイン内のサブネットは、重複したり、標準の IP サブネット定義に従って他のサブネットを含めたりすることもできません。

4. VPorts: VPort は、ドメイン階層の新しいレベルで、より粒度の高い設定を可能にするために設計されました。コンテナや VM に加え、ホストやブリッジインターフェースにアタッチには VPorts も使用し、ベアメタルサーバー、アプリケーション、レガシー VLAN に接続できるようにします。
5. ポリシーグループ: ポリシーグループは VPorts のコレクションです。

コンストラクトのマッピング

OpenShift Container Platform のコンセプトの多くは、Nuage VSP のコンストラクトに直接マッピングできます。

図5.3 Nuage VSP および OpenShift Container Platform のマッピング



Nuage サブネットは、OpenShift Container Platform ノードにマッピングされませんが、特定のプロジェクトのサブネットは、OpenShift Container Platform 内の複数のノードに対応できます。

A pod spawning in OpenShift Container Platform translates to a virtual port being created in VSP. The **vsp-openshift** plug-in interacts with the VRS and gets a policy for that virtual port from the VSD via the VSC. Policy Groups are supported to group multiple pods together that must have the same set of policies applied to them. Currently, pods can only be assigned to policy groups using the [operations workflow](#) where a policy group is created by the administrative user in VSD. The pod being a part of the policy group is specified by means of **nuage.io/policy-group** label in the specification of the pod.

統合コンポーネント

Nuage VSP は、2つの主要コンポーネントを使用して OpenShift Container Platform と統合します。

1. `nuage-openshift-monitor`
2. `vsp-openshift plugin`

`nuage-openshift-monitor`

nuage-openshift-monitor は、プロジェクト、サービス、ユーザー、ユーザーグループなどが作成されていないか、OpenShift Container Platform API サーバーを監視するサービスです。



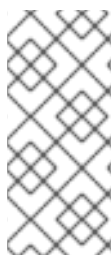
注記

複数のマスターがある高可用性の (HA) OpenShift Container Platform クラスターの場合には、**nuage-openshift-monitor** プロセスは、機能性に変更を加えずに、全マスター上で個別に実行されます。

開発者のワークフローでは、**nuage-openshift-monitor** も、VSD REST API を実行して OpenShift Container Platform コンストラクトを VSP コンストラクトにマッピングすることで、VSD オブジェクトを自動作成します。各クラスターインスタンスは、Nuage VSP の単一ドメインにマッピングします。これにより、Nuage でエンタープライズのドメインインスタンスごとに1つ設定するなど、特定のエンタープライズで複数のクラスターをインストールできます。各 OpenShift Container Platform プロジェクトは、Nuage VSP のクラスターのドメインに含まれるゾーンにマッピングされます。**nuage-openshift-monitor** で、プロジェクトの追加、削除が検出された場合に、対象のプロジェクトに対応する VSDK API を使用してゾーンをインスタンス化し、そのゾーンにサブネットのブロックを割り当てます。さらに、**nuage-openshift-monitor** は、このプロジェクトのネットワークマクログループも作成します。同様に、**nuage-openshift-monitor** でサービスの追加や削除が検出された場合には、サービス IP に対応するネットワークマクロを作成して、そのネットワークマクロを該当のプロジェクトのネットワークマクログループに割り当てて (ラベルを使用したユーザー提供のネットワークマクログループもサポートされています)、対象のサービスへの通信を有効化します。

開発者のワークフローでは、ゾーン内で作成された pod はすべて、そのサブネットプールからの IP を取得します。**nuage-openshift-monitor** が、master-config ファイルのプラグイン固有のパラメーター 2 つを基にして、サブネットプールを割り当て、管理します。ただし、実際の IP アドレスの解決および vport ポリシーの解決は、プロジェクトの作成時にインスタンス化されたドメイン/ゾーンを基に、VSD が行います。最初のサブネットプールが足りなくなった場合には、**nuage-openshift-monitor** はクラスターの CIDR から追加のサブネットを検出し、特定のプロジェクトに割り当てます。

オペレーションのワークフローでは、アプリケーションまたは pod 仕様に Nuage が認識するラベルを指定して、Pod と固有のユーザー定義ゾーンやサブネットを解決します。ただし、これは、**nuage-openshift-monitor** を使用して開発者ワークフローで作成したゾーンやサブネットの Pod を解決するために使用できません。



注記

オペレーションワークフローでは、管理者は VSD コンストラクトを事前作成し、Pod を特定のゾーン/サブネットにマッピングして、OpenShift エンティティー (ACL ルール、ポリシーグループ、ネットワークマクロ、ネットワークマクログループ) 間の通信を可能にします。Nuage ラベルの使用方法に関する説明は『[Nuage VSP OpenShift Integration Guide](#)』に記載されています。

vsp-openshift Plug-in

vsp-openshift ネットワークプラグインは、OpenShift Container Platform ランタイムが各 OpenShift Container Platform ノードで呼び出します。このプラグインは、ネットワークプラグイン init および Pod の設定、破棄、ステータスフックを実装します。vsp-openshift プラグインは、Pod に IP アドレスも割り当てます。特に、VRS (転送エンジン) と通信して、IP 情報を Pod に設定します。

5.4. 利用可能なルータープラグイン

A router can be assigned to a node to control traffic in an OpenShift cluster. OpenShift uses HAProxy as the default router, but options are available.

5.4.1. Default HAProxy Router

5.4.2. HAProxy Template Router

HAProxy テンプレートのルーター実装は、テンプレートルータープラグインの参照実装です。これは、[openshift3/ose-haproxy-router](#) リポジトリを使用して、テンプレートルータープラグインとともに、HAProxy インスタンスを実行します。

テンプレートルーターには2つのコンポーネントがあります。

- a wrapper that watches endpoints and routes and causes a HAProxy reload based on changes.
- a controller that builds the HAProxy configuration file based on routes and endpoints.



注記

[HAProxy ルーター](#) はバージョン 1.8.1 を使用します。

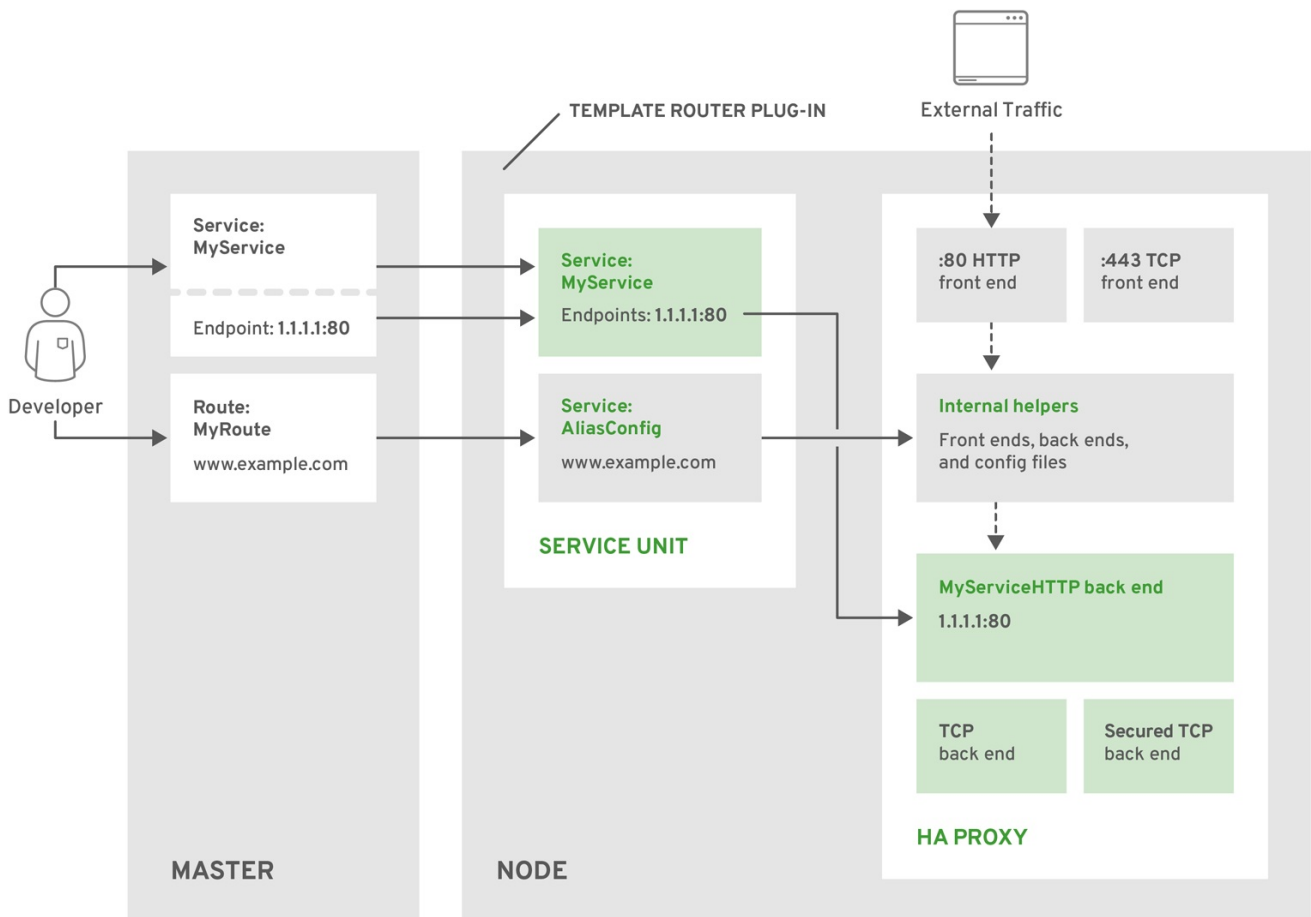
コントローラーおよび HAProxy は、Pod 内に常駐しており、デプロイメント設定で管理されます。ルーターの設定プロセスは、**oc adm router** コマンドで自動化されています。

コントローラーは、ルートとエンドポイントに変更がないか、また、HAProxy プロキシを監視します。変更が検出されたら、新しい haproxy-config ファイルを作成して、HAProxy を再起動します。haproxy-config ファイルは、ルーターのテンプレートファイルと OpenShift Container Platform からの情報をベースに構築されます。

HAProxy テンプレートファイルは、必要に応じてカスタマイズして、OpenShift Container Platform で現在サポートされていない機能をサポートすることができます。[HAProxy マニュアル](#) では、HAProxy がサポートする全機能を説明しています。

以下の図では、データがプラグインを使用してマスターから最終的に HAProxy 設定にどのように移動するかが記載されています。

図5.4 HAProxy ルーターデータフロー



OPENSIFT_415489_0218

HAProxy テンプレートルーターメトリクス

HAProxy ルーターは、外部のメトリクスコレクションや集計システム (例 Prometheus、statsd) で消費されるように、[Prometheus 形式](#) のメトリクスを提供して公開します。ルーターは、[HAProxy CSV 形式](#) のメトリクスを提供するように設定したり、まったくルーターメトリクスを提供しないように設定したりできます。

The metrics are collected from both the router controller and from HAProxy every 5 seconds. The router metrics counters start at zero when the router is deployed and increase over time. The HAProxy metrics counters are reset to zero every time haproxy is reloaded. The router collects HAProxy statistics for each frontend, backend and server. To reduce resource usage when there are more than 500 servers, the backends are reported instead of the servers since a backend can have multiple servers.

The statistics are a subset of the available HAProxy [Statistics](#).

The following HAProxy metrics are collected on a periodic basis and converted to Prometheus format. For every frontend the "F" counters are collected. When the counters are collected for each backend and the "S" server counters are collected for each server. Otherwise, the "B" counters are collected for each backend and no server counters are collected.

詳細は、[ルーター環境変数](#)を参照してください。

以下の表を参照してください。

列 1 - HAProxy CSV 統計のインデックス

列 2

F	Frontend metrics
b	Backend metrics when not showing Server metrics due to the Server Threshold,
B	Backend metrics when showing Server metrics
S	サーバーメトリクス

列3 - カウンター

列4 - カウンターの説明

インデックス	使用法	カウンター	説明
2	bBS	current_queue	サーバーに割り当てられていないキューに入れられた要求の現在の数。
4	FbS	current_sessions	アクティブなセッションの現在の数。
5	FbS	max_sessions	アクティブなセッションの観察される最大数。
7	FbBS	connections_total	接続の合計数。
8	FbS	bytes_in_total	受信バイトの現在の合計。
9	FbS	bytes_out_total	送信バイトの現在の合計。
13	bS	connection_errors_total	接続エラーの合計。
14	bS	response_errors_total	応答エラーの合計。
17	bBS	アップ	Current health status of the backend (1 = UP, 0 = DOWN).
21	S	check_failures_total	失敗したヘルスチェックの合計数。
24	S	downtime_seconds_total	合計ダウンタイム (秒)。

33	FbS	current_session_rate	直近の1秒間で、1秒あたりの現在のセッション数。
35	FbS	max_session_rate	1秒あたりの最大セッション実数。
40	FbS	http_responses_total	HTTP 応答合計数、コード 2xx。
43	FbS	http_responses_total	HTTP 合計応答数、コード 5xx。
60	bS	http_average_response_latency_milliseconds	直近の要求 1024 件のうちの HTTP 応答 (ミリ秒単位)。

ルーターコントローラーは、以下のアイテムを収集します。これらは、Prometheus 形式のメトリクスでのみ提供されます。

名前	説明
template_router_reload_seconds	ルーターの再読み込みにかかる時間を秒単位で測定します。
template_router_write_config_seconds	ルーター設定のディスクへの書き込みにかかる時間を秒単位で測定します。
haproxy_exporter_up	最後に成功した haproxy の収集です。
haproxy_exporter_csv_parse_failures	CSV の解析時のエラー数です。
haproxy_exporter_scrape_interval	次の収集が許可されるまでの秒単位の時間です (データのサイズに比例します)。
haproxy_exporter_server_threshold	追跡したサーバー数と現在のしきい値です。
haproxy_exporter_total_scrapes	現在の合計 HAProxy 収集数です。
http_request_duration_microseconds	マイクロ秒単位の HTTP 要求のレイテンシーです。
http_request_size_bytes	バイト単位の HTTP 要求サイズです。
http_response_size_bytes	バイト単位の HTTP 応答サイズです。

openshift_build_info	OpenShift がビルドされた major、minor、git commit、git version でラベル付けされた定数値 '1' のメトリクスです。
ssh_tunnel_open_count	SSH トンネルを開放しようと試行した合計数です。
ssh_tunnel_open_fail_count	SSH トンネルを開放しようとして失敗した合計数です。

5.4.3. F5 BIG-IP® Router plug-in

A router is one way to get traffic into the cluster. The F5 BIG-IP® Router plug-in is one of the available [router plugins](#).



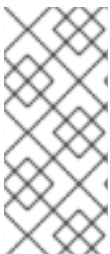
注記

The F5 router plug-in is available starting in OpenShift Enterprise 3.0.2.

The F5 router plug-in integrates with an existing **F5 BIG-IP®** system in your environment. **F5 BIG-IP®** version 11.4 or newer is required in order to have the F5 iControl REST API. The F5 router supports [unsecured](#), [edge terminated](#), [re-encryption terminated](#), and [passthrough terminated](#) routes matching on HTTP vhost and request path.

The F5 router has feature parity with the [HAProxy template router](#), and has additional features over the **F5 BIG-IP®** support in OpenShift Enterprise 2. Compared with the **routing-daemon** used in earlier versions, the F5 router additionally supports:

- path-based routing (using policy rules),
- re-encryption (implemented using client and server SSL profiles)
- passthrough of encrypted connections (implemented using an iRule that parses the SNI protocol and uses a data group that is maintained by the F5 router for the servername lookup).



注記

Passthrough routes are a special case: path-based routing is technically impossible with passthrough routes because **F5 BIG-IP®** itself does not see the HTTP request, so it cannot examine the path. The same restriction applies to the template router; it is a technical limitation of passthrough encryption, not a technical limitation of OpenShift Container Platform.

5.4.3.1. Routing Traffic to Pods Through the SDN

Because **F5 BIG-IP®** is external to the [OpenShift SDN](#), a cluster administrator must create a peer-to-peer tunnel between **F5 BIG-IP®** and a host that is on the SDN, typically an OpenShift Container Platform node host. This [ramp node](#) can be configured as [unschedulable](#) for pods so that it will not be doing anything except act as a gateway for the **F5 BIG-IP®** host. It is also possible to configure multiple such hosts and use the OpenShift Container Platform **ipfailover** feature for redundancy; the **F5 BIG-IP®** host would then need to be configured to use the **ipfailover** VIP for its tunnel's remote endpoint.

5.4.3.2. F5 Integration Details

The operation of the F5 router is similar to that of the OpenShift Container Platform **routing-daemon** used in earlier versions. Both use REST API calls to:

- create and delete pools,
- add endpoints to and delete them from those pools, and
- configure policy rules to route to pools based on vhost.

Both also use **scp** and **ssh** commands to upload custom TLS/SSL certificates to **F5 BIG-IP®**.

The F5 router configures pools and policy rules on virtual servers as follows:

- When a user creates or deletes a route on OpenShift Container Platform, the router creates a pool to **F5 BIG-IP®** for the route (if no pool already exists) and adds a rule to, or deletes a rule from, the policy of the appropriate vserver: the HTTP vserver for non-TLS routes, or the HTTPS vserver for edge or re-encrypt routes. In the case of edge and re-encrypt routes, the router also uploads and configures the TLS certificate and key. The router supports host- and path-based routes.



注記

Passthrough routes are a special case: to support those, it is necessary to write an iRule that parses the SNI ClientHello handshake record and looks up the servername in an F5 data-group. The router creates this iRule, associates the iRule with the vserver, and updates the F5 data-group as passthrough routes are created and deleted. Other than this implementation detail, passthrough routes work the same way as other routes.

- When a user creates a service on OpenShift Container Platform, the router adds a pool to **F5 BIG-IP®** (if no pool already exists). As endpoints on that service are created and deleted, the router adds and removes corresponding pool members.
- When a user deletes the route and all endpoints associated with a particular pool, the router deletes that pool.

5.4.3.3. F5 Native Integration

With [native integration of F5 with OpenShift Container Platform](#), you do not need to configure a ramp node for F5 to be able to reach the pods on the overlay network as created by OpenShift SDN.

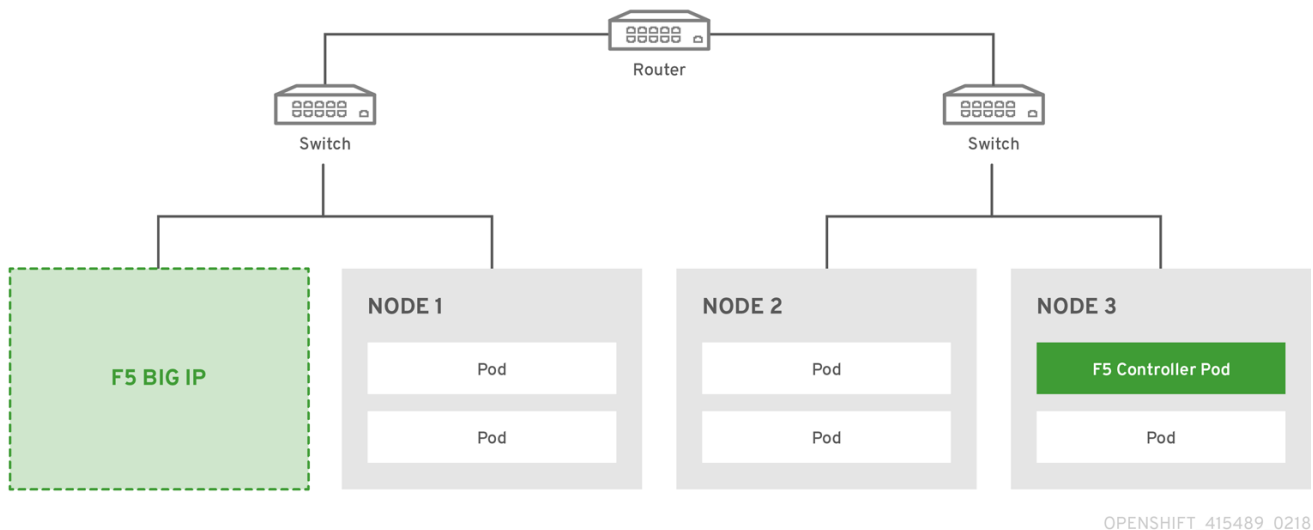
Also, only **F5 BIG-IP®** appliance version 12.x and above works with the native integration presented in this section. You also need **sdn-services** add-on license for the integration to work properly. For version 11.x, [set up a ramp node](#).

Connection

The F5 appliance can connect to the OpenShift Container Platform cluster via an L3 connection. An L2 switch connectivity is not required between OpenShift Container Platform nodes. On the appliance, you can use multiple interfaces to manage the integration:

- Management interface - Reaches the web console of the F5 appliance.
- External interface - Configures the virtual servers for inbound web traffic.

- Internal interface - Programs the appliance and reaches out to the pods.



An F5 controller pod has **admin** access to the appliance. The F5 image is launched within the OpenShift Container Platform cluster (scheduled on any node) that uses iControl REST APIs to program the virtual servers with policies, and configure the VxLAN device.

Data Flow: Packets to Pods



注記

This section explains how the packets reach the pods, and vice versa. These actions are performed by the F5 controller pod and the F5 appliance, not the user.

When natively integrated, The F5 appliance reaches out to the pods directly using VxLAN encapsulation. This integration works only when OpenShift Container Platform is using **openshift-sdn** as the network plug-in. The **openshift-sdn** plug-in employs VxLAN encapsulation for the overlay network that it creates.

To make a successful data path between a pod and the F5 appliance:

1. F5 needs to encapsulate the VxLAN packet meant for the pods. This requires the **sdn-services** license add-on. A VxLAN device needs to be created and the pod overlay network needs to be routed through this device.
2. F5 needs to know the VTEP IP address of the pod, which is the IP address of the node where the pod is located.
3. F5 needs to know which **source-ip** to use for the overlay network when encapsulating the packets meant for the pods. This is known as the **gateway address**.
4. OpenShift Container Platform nodes need to know where the F5 gateway address is (the VTEP address for the return traffic). This needs to be the internal interface's address. All nodes of the cluster must learn this automatically.
5. Since the overlay network is multi-tenant aware, F5 must use a VxLAN ID that is representative of an **admin** domain, ensuring that all tenants are reachable by the F5. Ensure that F5 encapsulates all packets with a **vni** of **0** (the default **vni** for the **admin** namespace in OpenShift Container Platform) by putting an annotation on the manually created **hostsubnet - pod.network.openshift.io/fixed-vni-host: 0**.

A ghost **hostsubnet** is manually created as part of the setup, which fulfills the third and fourth listed requirements. When the F5 controller pod is launched, this new ghost **hostsubnet** is provided so that the F5 appliance can be programmed suitably.



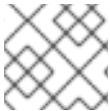
注記

The term **ghost hostsubnet** is used because it suggests that a subnet has been given to a node of the cluster. However, in reality, it is not a real node of the cluster. It is hijacked by an external appliance.

The first requirement is fulfilled by the F5 controller pod once it is launched. The second requirement is also fulfilled by the F5 controller pod, but it is an ongoing process. For each new node that is added to the cluster, the controller pod creates an entry in the VxLAN device's VTEP FDB. The controller pod needs access to the **nodes** resource in the cluster, which you can accomplish by giving the service account appropriate privileges. Use the following command:

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

Data Flow from the F5 Host



注記

These actions are performed by the F5 controller pod and the F5 appliance, not the user.

1. The destination pod is identified by the F5 virtual server for a packet.
2. VxLAN dynamic FDB is looked up with pod's IP address. If a MAC address is found, go to step 5.
3. Flood all entries in the VTEP FDB with ARP requests seeking the pod's MAC address. An entry is made into the VxLAN dynamic FDB with the pod's MAC address and the VTEP to be used as the value.
4. Encap an IP packet with VxLAN headers, where the MAC of the pod and the VTEP of the node is given as values from the VxLAN dynamic FDB.
5. Calculate the VTEP's MAC address by sending out an ARP or checking the host's neighbor cache.
6. Deliver the packet through the F5 host's internal address.

Data Flow: Return Traffic to the F5 Host



注記

These actions are performed by the F5 controller pod and the F5 appliance, not the user.

1. The pod sends back a packet with the destination as the F5 host's VxLAN gateway address.
2. The **openvswitch** at the node determines that the VTEP for this packet is the F5 host's internal interface address. This is learned from the ghost **hostsubnet** creation.
3. A VxLAN packet is sent out to the internal interface of the F5 host.



注記

During the entire data flow, the VNID is pre-fixed to be **0** to bypass multi-tenancy.

5.5. ポート転送

5.5.1. 概要

OpenShift Container Platform は、Kubernetes に組み込まれている機能を活用して、Pod へのポート転送をサポートします。これは、HTTP と SPDY または HTTP/2 などの多重化ストリーミングプロトコルを使用して実装されます。

Developers can use the CLI to port forward to a pod. The CLI listens on each local port specified by the user, forwarding via the described protocol.

5.5.2. サーバー操作

Kubelet は、クライアントからのポート転送要求を処理します。要求を受信すると、応答をアップグレードし、クライアントがポート転送ストリームを作成するまで待機します。新規ストリームを受信したら、ストリームと Pod ポート間のデータをコピーします。

アーキテクチャ的には、Pod のポートに転送するオプションがあります。OpenShift Container Platform でサポートされる現在の実装はノードホストで直接 **nsenter** を呼び出し、Pod ネットワークの namespace に入り、**socat** を呼び出してストリームと Pod のポート間のデータをコピーします。ただし、カスタムの実装には、**nsenter** と **socat** のバイナリーをホストにインストールしなくていいように、これらのバイナリーを実行する「ヘルパー」Pod の実行が含まれています。

5.6. リモートコマンド

5.6.1. 概要

OpenShift Container Platform は Kubernetes に内蔵されている機能を活用し、コンテナでのコマンド実行をサポートします。これは、HTTP と SPDY または HTTP/2 などの多重化ストリーミングプロトコルを使用して実装されます。

Developers can use the CLI to execute remote commands in containers.

5.6.2. サーバー操作

Kubelet は、クライアントからのリモート実行要求を処理します。要求を受信すると応答をアップグレードして、要求ヘッダーを評価してどのストリーム (**stdin**、**stdout** および/または **stderr**) を受信するか判断し、クライアントがストリームを作成するまで待機します。

Kubelet が全ストリームを受信したら、コンテナでコマンドを実行して、ストリームとコマンドの **stdin**、**stdout** および **stderr** を適切にコピーします。コマンドが中断されたら、Kubelet はアップグレードされた接続と基盤の接続を終了します。

アーキテクチャ的に、コンテナでコマンドを実行するオプションがあります。OpenShift Container Platform で現在サポートされている実装では、ノードホストで **nsenter** を直接呼び出して、コマンド実行前に、ノードホストがコンテナの namespace に入ることができるようにします。ただし、カスタム実装には **docker exec** の使用や、ホストでインストールする必要がある **nsenter** バイナリーが必須とならないように **nsenter** を実行する「ヘルパー」コンテナの実行が含まれる場合があります。

5.7. ROUTES (ルート)

5.7.1. 概要

OpenShift Container Platform ルートは、外部クライアントが名前ですべてサービスに到達できるように、[www.example.com](#) などのホスト名で [サービス](#) を公開します。

DNS resolution for a host name is handled separately from routing. Your administrator may have configured a [DNS wildcard entry](#) that will resolve to the OpenShift Container Platform node that is running the OpenShift Container Platform router. If you are using a different host name you may need to modify its DNS records independently to resolve to the node that is running the router.

各ルーターは名前 (63 文字に制限)、サービスセクター、およびオプションのセキュリティー設定で構成されています。

5.7.2. ルーター

An OpenShift Container Platform administrator can deploy **routers** to nodes in an OpenShift Container Platform cluster, which enable routes [created by developers](#) to be used by external clients. The routing layer in OpenShift Container Platform is pluggable, and several [router plug-ins](#) are provided and supported by default.



注記

See the [Installation and Configuration](#) guide for information on deploying a router.

ルーターは、サービスセクターを使用して、[サービス](#)と、サービスをバックアップするエンドポイントを検索します。ルーターとサービス両方が負荷分散を提供する場合には、OpenShift Container Platform はルーターの負荷分散を使用します。ルーターはサービスの IP アドレスに関連の変更がないかを検出して、その設定に合わせて変化します。これは、カスタムルーターが API オブジェクトの変更を外部のルーティングソリューションに通信できるので、便利です。

要求のパスは、1つまたは複数のルーターに対して、ホスト名の DNS を解決することから始まります。推奨の方法は、複数のルーターインスタンスでバックされる1つまたは複数の仮想 IP (VIP) アドレスを参照するワイルドカード DNS エントリーでクラウドドメインを定義する方法です。クラウドドメイン外の名前とアドレスを使用するルートには、個別の DNS エントリー設定が必要です。

When there are fewer VIP addresses than routers, the routers corresponding to the number of addresses are **active** and the rest are **passive**. A passive router is also known as a **hot-standby** router. For example, with two VIP addresses and three routers, you have an "active-active-passive" configuration. See [High Availability](#) for more information on router VIP configuration.

ルートは、ルーターセット間で [シャード化](#) されます。管理者は、クラスター全体でシャード化を設定し、ユーザーはプロジェクトに含まれる namespace にシャード化を設定できます。オペレーターは、シャード化すると、複数のルーターグループを定義できるようになります。このグループ内の各ルーターはトラフィックのサブネット1つにのみ対応できます。

OpenShift Container Platform ルーターは、外部のホスト名マッピングと、ルーターに区別情報を直接渡すプロトコルを使用して [サービス](#) エンドポイントの負荷分散を行います。ルーターは、送信先を判断できるように、プロトコルにホスト名が存在する必要があります。

ルータープラグインはデフォルトで、ホストポート 80 (HTTP) および 443 (HTTPS) にバインドできます。つまり、配置されていないと、これらのポートは使用されないため、ルーターはノードに配置されている必要があります。または、ルーターは、**ROUTER_SERVICE_HTTP_PORT** および

`ROUTER_SERVICE_HTTPS_PORT` 環境変数を設定して、他のポートをリッスンするように設定してください。

ルーターは、ホストノードのポートにバインドされるので、ルーターがホストネットワークを使用している場合には(デフォルト)、各ノードに1つしかこれらのポートをリッスンするルーターを配置できません。クラスターネットワークは、全ルーターがクラスター内のすべての Pod にアクセスできるように設定します。

ルーターは以下のプロトコルをサポートします。

- HTTP
- HTTPS (SNI を使用)
- WebSocket
- SNI 付きの TLS



注記

WebSocket トラフィックは、同じルート規則を使用し、他のトラフィックと同じ TLS 終端タイプをサポートします。

セキュアな接続を確立するには、クライアントとサーバーに共通する **暗号化** を取り決める必要があります。時間が経つにつれ、よりセキュリティの高く、新しい暗号化が利用でき、クライアントソフトウェアに統合されます。以前のクライアントは陳腐化し、セキュリティの低い、以前の暗号化は使用が停止されます。デフォルトでは、ルーターは、一般的に入手できる、幅広いクライアントに対応します。ルーターは、任意のクライアントをサポートする暗号化の中で選択したものを使用し、セキュリティが低い暗号化を使用しないように設定できます。

5.7.2.1. テンプレートルーター

テンプレートルーター は、ルーターの一種で、特定のインフラストラクチャー情報を基盤のルーター実装に提供します。以下に例を示します。

- エンドポイントおよびルートを監視するラッパーです。
- 消費可能なフォームに保存されるエンドポイントとルートデータ。
- 内部の状態を設定可能なテンプレートに渡し、テンプレートを実行します。
- 再ロードスクリプトを呼び出します。

5.7.3. 利用可能なルータープラグイン

検証された利用可能なルータープラグインについては、「[利用可能なプラグイン](#)」のセクションを参照してください。

Instructions on deploying these routers are available in [Deploying a Router](#).

5.7.4. スティックセッション

スティッキーセッションの実装は、基盤のルーター設定により異なります。デフォルトの HAProxy テンプレートは、**balance source** 命令を使用してスティッキーセッションを実装し、ソース IP を基に分散されます。さらに、このテンプレートルータープラグインは、サービス名と namespace を基盤の実

装に渡します。これは、ピア間で同期させるスティックテーブルの実装など、より高度な設定に使用できます。

スティッキーセッションは、ユーザー体験の向上のため、ユーザーのセッションからの全トラフィックが確実に同じ Pod に移動されるようにします。ユーザーの要求を満たしながら、Pod は後続の要求で使用できるように、データをキャッシュします。たとえば、バックエンド Pod が 5 つと負荷分散ルーターが 2 つあるクラスターでは、要求を処理するルーターがどれであっても、同じ Pod で、同じ Web ブラウザーからの web トラフィックを受信できるように確保できます。

ルーティングトラフィックを同じ Pod に返すことが望まれる場合でも、保証はできませんが、HTTP ヘッダーを使用して、cookie を設定し、最後の接続で使用した Pod を判断できます。ユーザーがアプリケーションに別の要求を送信した場合には、ブラウザーが cookie を再送することで、ルーターでトラフィックの送信先が分かります。

クラスター管理者は、スティッキネスをオフにして、他の接続とパススルールートを分割することも、完全にスティッキネスをオフにすることもできます。

デフォルトでは、パススルールートのスティッキーセッションは、**source 負荷分散ストラテジー**を使用して実装します。すべてのパスルート用には、**ROUTER_TCP_BALANCE_SCHEME 環境変数**で、個別ルート用には、**haproxy.router.openshift.io/balance ルート固有のアノテーション**で、デフォルトを変更することができます。

他の種類のルートはデフォルトで **leastconn 負荷分散ストラテジー**を使用しますが、これは **ROUTER_LOAD_BALANCE_ALGORITHM 環境変数**を使用して変更できます。また、個別ルートには **haproxy.router.openshift.io/balance ルート固有のアノテーション**を使用して変更することができます。



注記

cookie は、HTTP トラフィックを表示できないので、パススルールートで設定できません。代わりに、ソース IP アドレスをベースに数が計算され、バックエンドを判断します。

バックエンドが変わった場合には、トラフィックは誤ったサーバーに送られ、スティッキネスが低くなります。ロードバランサーを使用する場合は (ソース IP が表示されない)、同じ番号が全接続に設定され、トラフィックが同じ Pod に送信されます。

さらに、このテンプレートルータープラグインは、サービス名と namespace を基盤の実装に渡します。これは、ピア間で同期させるスティックテーブルの実装など、より高度な設定に使用できます。

Specific configuration for this router implementation is stored in the **haproxy-config.template** file located in the **/var/lib/haproxy/conf** directory of the router container. The file may be [customized](#).



注記

source の **負荷分散ストラテジー** は、NAT の設定が原因で、外部のクライアント IP アドレスを区別しないので、送信元の IP アドレス (HAProxy リモート) は同じです。HAProxy ルーターが **hostNetwork: true** で実行されない限り、すべての外部クライアントは単一の Pod にルーティングされます。

5.7.5. ルーターの環境変数

このセクションで説明されているアイテムはすべて、ルーターの **デプロイメント設定** に環境変数を指定して設定を変更するか、**oc set env** コマンドを使用します。


```
$ oc set env <object_type>/<object_name> KEY1=VALUE1 KEY2=VALUE2
```

例:

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=127.0.0.1 ROUTER_LOG_LEVEL=debug
```

表5.2 ルーターの環境変数

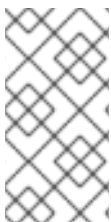
変数	デフォルト	説明
DEFAULT_CERTIFICATE		TLS サーバー証明書を公開しないルートに使用するデフォルトの証明書の内容。PEM 形式。
DEFAULT_CERTIFICATE_DIR		tls.crt というファイルを含むディレクトリへのパス。 tls.crt が PEM ファイルでなく、秘密鍵も含む場合には、同じディレクトリ内の tls.key というファイルと先に統合されます。PEM 形式のコンテンツは、デフォルトの証明書として使用されます。これは、 DEFAULT_CERTIFICATE または DEFAULT_CERTIFICATE_PATH が指定されていない場合のみ使用されます。
DEFAULT_CERTIFICATE_PATH		TLS サーバー証明書を公開しないルートに使用するデフォルト証明書へのパス。PEM 形式。 DEFAULT_CERTIFICATE が指定されていない場合のみ使用されます。
DROP_SYN_DURING_RESTART	false	When set to true , enables an iptables rule to drop certain packets during a restart to provide a seamless restart. See the install guide for details.
EXTENDED_VALIDATION	true	true の場合は、ルーターにより、証明書が構造的に正しいことを確認します。CA に対して証明書は検証されません。テストをオフにするには、 false に設定します。
NAMESPACE_LABELS		監視する namespace に適用するラベルセクターです。空はすべてを意味します。
PROJECT_LABELS		監視するプロジェクトに適用するラベルセクターです。空はすべてを意味します。
RELOAD_SCRIPT		ルーターを再ロードするために使用する再ロードスクリプトのパスです。
ROUTER_ALLOWEDDomains		ルートのホスト名のみを含めることができるドメインのコンマ区切りの一覧。ドメインに含まれるサブドメインを使用できます。 ROUTER_DENIEDDomains オプションは、このオプションに指定されている値を上書きします。これが設定されている場合は、許可されているドメイン以外はすべて拒否されず。

変数	デフォルト	説明
ROUTER_BACK_END_PROCESS_ENDPOINTS		テンプレート関数 <code>processEndpointsForAlias</code> の使用時にエンドポイントをどのように処理すべきかを指定する文字列。有効な値は ["shuffle", ""] です。"shuffle" は、全呼び出しごとに要素を無作為に決定します。デフォルトの動作は、事前に決定した順番に、値が返されます。
ROUTER_BIND_PORTS_AFTER_SYNC	false	true または TRUE に設定されている場合には、ルーターは、完全な同期状態になるまでポートにバインドされません。'true' または 'TRUE' に設定されていない場合は、ルーターはポートにバインドされ、即座に要求処理を開始しますが、ルートで読み込まれないものもあります。
ROUTER_COOKIE_NAME		cookie 名を指定して、内部で生成したデフォルト名を上書きします。名前は、大文字、小文字、数字、 "_" または "-" を任意に組み合わせて指定する必要があります。デフォルトは、ルートのハッシュ化された内部キー名です。
ROUTER_COMPRESSION_MIME	"text/html text/plain text/css"	圧縮するスペースで区切られた mime タイプの一覧です。
ROUTER_DENIED_DOMAINS		ルートのホスト名に含めることができないドメインのコンマ区切りの一覧。ドメインに含まれるサブドメインを使用できません。 ROUTER_ALLOWED_DOMAINS オプションを上書きします。
ROUTER_ENABLE_COMPRESSION		true または TRUE の場合、可能な場合には応答を圧縮します。
ROUTER_LISTEN_ADDR	0.0.0.0:1936	ルーターメトリクスのリッスンアドレスを設定します。
ROUTER_LOG_LEVEL	警告	syslog サーバーに送信するログレベルです。
ROUTER_MAX_CONNECTIONS	20000	同時接続の最大数です。
ROUTER_METRICS_HAPROXY_SERVER_THRESHOLD	500	
ROUTER_METRICS_HAPROXY_EXPORTED		CSV 形式で収集されるメトリクスです。例: defaultSelectedMetrics = []int{2, 4, 5, 7, 8, 9, 13, 14, 17, 21, 24, 33, 35, 40, 43, 60}

変数	デフォルト	説明
ROUTER_METRICS_HAPROXY_BASE_SCRAPING_INTERVAL	5s	
ROUTER_METRICS_HAPROXY_TIMEOUT	5s	
ROUTER_METRICS_TYPE	haproxy	HAProxy ルーターのメトリクスを生成します (haproxy のみがサポートされている値です)。
ROUTER_OVERRIDE_HOSTNAME		true に設定されている場合、 ROUTER_SUBDOMAIN のテンプレートのあるルートの spec.host 値を上書きします。
ROUTER_SERVICE_HTTPS_PORT	443	HTTPS 要求をリッスンするポートです。
ROUTER_SERVICE_HTTP_PORT	80	HTTP 要求をリッスンするポートです。
ROUTER_SERVICE_NAME	パブリック	ルーターがルートステータスで自らを識別する名前です。
ROUTER_CANONICAL_HOSTNAME		ルーターステータスに表示されるルーターの (オプションの) ホスト名です。
ROUTER_SERVICE_NAMESPACE		ルーターがルーターステータスで自らを識別する namespace です。 ROUTER_SERVICE_NAME が使用されている場合は必須です。
ROUTER_SERVICE_NO_SNI_PORT	10443	一部のフロントエンドからバックエンドへの通信に使用される内部ポートです (以下を参照してください)。
ROUTER_SERVICE_SNI_PORT	10444	一部のフロントエンドからバックエンドへの通信に使用される内部ポートです (以下を参照してください)。
ROUTER_SUBDOMAIN		spec.host なしでルートのホスト名を生成するために使用されるテンプレートです (例: <code>_\${name}-\${namespace}.myapps.mycompany.com</code>)。

変数	デフォルト	説明
ROUTER_SYSL LOG_ADDRESS		ログメッセージを送信するアドレスです。空の場合は無効になります。
ROUTER_SYSL LOG_FORMAT		設定されている場合は、基盤のルーター実装で使用されるデフォルトのログ形式が上書きされます。この値は、基盤のルーター実装の仕様に従う必要があります。
ROUTER_TCP_ BALANCE_ SCHEME	ソース	Load-balancing strategy for multiple endpoints for pass-through routes. Available options are source , roundrobin , or leastconn .
ROUTER_LOAD_ BALANCE_ ALGORITHM	leastconn	Load-balancing strategy for routes with multiple endpoints. Available options are source , roundrobin , and leastconn .
ROUTE_LABELS		監視するルートに適用するラベルセクターです。何も指定しない場合はすべてを意味します。
STATS_PASSW ORD		ルーターの統計にアクセスするのに必要なパスワード (ルーターの実装がサポートする場合)
STATS_PORT		統計を公開するポート (ルーターの実装がサポートする場合)。設定されていない場合は統計は公開されません。
STATS_USERN AME		ルーターの統計にアクセスするために必要なユーザー名 (ルーターの実装がこれをサポートしている場合)。
TEMPLATE_FIL E	<code>/var/lib/haproxy/ conf/custom/ haproxy-config- custom.template</code>	HAProxy テンプレートへのパス (コンテナイメージ内)。
ROUTER_USE_ PROXY_PROTO COL		true または TRUE に設定されている場合には、HAProxy は受信接続がポート 80 と 443 上で PROXY プロトコルを使用していることを想定します。ソース IP アドレスは、ロードバランサーが Amazon ELB などのプロトコルをサポートする場合には、ロードバランサーをパススルーすることができます。
ROUTER_ALLO W_WILDCARD_ ROUTES		true または TRUE が設定されている場合には、ルーターの受付チェックを合格するという Subdomain のワイルドカードポリシーが指定されたルートは、HAProxy ルーターがサービスを提供します。

変数	デフォルト	説明
ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK		namespace 所有権ポリシーを緩和するために true に設定されます。
ROUTER_STRICT_SNI		strict-sni
ROUTER_CIPHERS	intermediate	バインドでサポートされる 暗号 のセットを指定します。



注記

同じマシンで複数のルーターを実行する場合には、ルーターがリスンするポート (**ROUTER_SERVICE_SNI_PORT** および **ROUTER_SERVICE_NO_SNI_PORT**) を変更する必要があります。これらのポートは、マシン上で一意であれば、何でも指定できます。また、これらのポートは外部には公開されません。

ルータータイムアウト変数

TimeUnits は数字、その後に単位を指定して表現します。**us** *(マイクロ秒)、**ms** (ミリ秒、デフォルト)、**s** (秒)、**m** (分)、**h** *(時間)、**d** (日)

正規表現: `[1-9][0-9]*(us|ms|s|m|h|d)`

ROUTER_BACKEND_CHECK_INTERVAL	5000ms	バックエンドでの後続の liveness チェックの時間の長さ。
ROUTER_CLIENT_FIN_TIMEOUT	1s	クライアントがルートに接続する場合の TCP FIN タイムアウトの期間を制御します。接続切断のために送信された FIN が指定の時間内に応答されない場合には、HAProxy が接続を切断します。小さい値を設定し、ルーターでリソースをあまり使用していない場合には、リスクはありません。
ROUTER_DEFAULT_CLIENT_TIMEOUT	30s	クライアントがデータを確認するか、送信するための時間の長さ。
ROUTER_DEFAULT_CONNECT_TIMEOUT	5s	最大接続時間。
ROUTER_DEFAULT_SERVER_FIN_TIMEOUT	1s	ルーターからルートをバックグする Pod の TCP FIN タイムアウトを制御します。

ROUTER_DEFAULT_SERVER_TIMEOUT	30s	サーバーがデータを確認するか、送信するための時間の長さ。
ROUTER_DEFAULT_TUNNEL_TIMEOUT	1h	TCP または WebSocket 接続が開放された状態で保つ時間数。websockets/tcp 接続がある場合 (および HAProxy が再読み込みされる度) に、以前の HAProxy プロセスが指定された時間分、開放された状態に保たれます。
ROUTER_SLOWLORIS_HTTP_KEEPALIVE	300s	新しい HTTP 要求が表示されるまで待機する最大時間を設定します。この値が低すぎる場合には、ブラウザおよびアプリケーションの keepalive 値が低くなりすぎて、問題が発生する可能性があります。詳しい情報は以下の注記セクションを参照してください。
ROUTER_SLOWLORIS_TIMEOUT	10s	HTTP 要求の伝送にかかる時間。
RELOAD_INTERVAL	5s	新規の変更を許可するためにルーターの再ロードが許可される最小の頻度。
ROUTER_METRICS_HAPROXY_TIMEOUT	5s	HAProxy メトリクスの収集タイムアウト。



注記

有効なタイムアウト値には、想定した個別のタイムアウトではなく、特定の変数を合計した値に指定することができます。

例: **ROUTER_SLOWLORIS_HTTP_KEEPALIVE** は **timeout http-keep-alive** を調節し、デフォルトで **300s** に設定されていますが、haproxy は **tcp-request inspect-delay** も待機し、値は **5s** に設定されているので、今回の場合には、合計のタイムアウトは **300s** に **5s** を加えた値になります。

5.7.6. ロードバランシングストラテジー

ルートに複数のエンドポイントがある場合には、HAProxy は選択した負荷分散ストラテジーを基に、エンドポイント間に要求を分散します。これは、セッションの最初の要求など、永続情報が利用できない場合に適用されます。

ストラテジーには以下のいずれか使用できます。

- **roundrobin**: 各エンドポイントは、重みに従い、順番に使用されます。これは、サーバーの処理が均等に分散される場合に最もスムーズで公平なアルゴリズムです。
- **leastconn**: 接続数が最も少ないエンドポイントが要求を受信します。接続数が最も少ないエンドポイントが複数ある場合には、ラウンドロビンが実行されます。LDAP、SQL、TSE など、セッションが非常に長い場合にこのアルゴリズムを使用してください。HTTP など、短いセッ

ションを通常使用するプロトコルでの使用を目的とはしていません。

- **source**: ソース IP アドレスは、ハッシュ化され、実行中サーバーの合計の重みで分割されて、どのサーバーが要求を受信するか指定します。これにより、サーバーが終了/起動しない限り、同じクライアント IP アドレスは常に同じサーバーに到達します。実行中のサーバー数が変化したことで、ハッシュの結果が変化した場合には、多くのクライアントが異なるサーバーに転送されます。このアルゴリズムは一般的にパススルールートで使用されます。

The **ROUTER_TCP_BALANCE_SCHEME** environment variable sets the default strategy for passthrough routes. The **ROUTER_LOAD_BALANCE_ALGORITHM** environment variable sets the default strategy for the router for the remaining routes. A [route specific annotation](#), **haproxy.router.openshift.io/balance**, can be used to control specific routes.

5.7.7. HAProxy Strict SNI

デフォルトでは、ホストは HTTPS または TLS SNI 要求のルートを解決しないので、デフォルト証明書が 503 応答の一部として、呼び出し元に返されます。これにより、デフォルト証明書が公開され、不正な証明書がサイトに提供されるので、セキュリティの問題を引き起こす可能性があります。バインド用の HAProxy **strict-sni** オプションを使用するとデフォルト証明書の使用が抑制されます。

ROUTER_STRICT_SNI 環境変数はバインド処理を制御します。 **true** または **TRUE** に設定されている場合には、 **strict-sni** が HAProxy バインドに追加されます。デフォルト設定は **false** です。

このオプションは、ルーターの作成時または後の追加時に設定できます。

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

これは、 **ROUTER_STRICT_SNI=true** を設定できます。

5.7.8. ルーターの暗号スイート

各クライアント (例: Chrome 30 または Java8) には、ルーターにセキュアに接続するために使用する暗号スイートが含まれます。ルーターには、接続を完了させるには、暗号化が最低でも1つ必要です。

表5.3 ルーター暗号プロファイル

プロファイル	互換性のある最も古いクライアント
modern	Firefox 27, Chrome 30, IE 11 on Windows 7, Edge, Opera 17, Safari 9, Android 5.0, Java 8
intermediate	Firefox 1, Chrome 1, IE 7, Opera 5, Safari 1, Windows XP IE8, Android 2.3, Java 7
old	Windows XP IE6, Java 6

詳細は、 [Security/Server Side TLS](#) リファレンスガイドを参照してください。

The router defaults to the **intermediate** profile. You can select a different profile using the **--ciphers** option when creating a route, or by changing the **ROUTER_CIPHERS** environment variable with the values **modern**, **intermediate**, or **old** for an existing router. Alternatively, a set of ":" separated ciphers can be provided. The ciphers must be from the set displayed by:

```
openssl ciphers
```

5.7.9. ルートホスト名

OpenShift Container Platform ルートを使用してサービスと外部に到達可能なホスト名を関連付けることで、サービスを外部に公開することができます。このエッジホスト名は次に、サービスにトラフィックをルーティングするのに使用します。

異なる namespace から複数のルートが同じホストを要求する場合に、一番古いルートが優先され、その namespace にホストを獲得します。同じ namespace 内に、追加のルートが異なるパスフィールドで定義されている場合には、これらのパスが追加されます。複数のルートに同じパスが使用されている場合には、一番古いものが優先されます。

あるユーザーがホスト名にルート 2 つ (1 つが新しく、1 が古い) を指定しようとしていると仮定します。このユーザーがホスト名に他の 2 つのルート指定する前に、別のユーザーが同じホスト名にルートを指定したうに、元のユーザーにより作成済みのルートが削除された場合に、このホスト名への要求は効果がなくなります。他の namespace がこのホスト名を要求し、最初のユーザーの要求はなくなります。

例5.1 指定されたホストを持つルート:

```
apiVersion: v1
kind: Route
metadata:
  name: host-route
spec:
  host: www.example.com ①
  to:
    kind: Service
    name: service-name
```

① サービスを公開するために使用される外部から到達可能なホスト名を指定します。

例5.2 ホスト内のルート:

```
apiVersion: v1
kind: Route
metadata:
  name: no-route-hostname
spec:
  to:
    kind: Service
    name: service-name
```

ホスト名がルート定義の一部として指定されていない場合には、OpenShift Container Platform が自動的に生成します。生成されたホスト名は以下のような形式をとります。

```
<route-name>[-<namespace>].<suffix>
```

以下の例では、namespace **mynamespace** にホストを追加せずに、上記のルート設定に対して OpenShift Container Platform が生成したホスト名を示します。

例5.3 生成されるホスト名

no-route-hostname-mynamespace.router.default.svc.cluster.local **1**

- 1** 生成されたホスト名のサフィックスは、デフォルトのルーティングサブドメイン `router.default.svc.cluster.local` です。

A cluster administrator can also [customize the suffix used as the default routing subdomain](#) for their environment.

5.7.10. ルートタイプ

ルートにセキュリティーを設定してもしなくても構いません。セキュアなルートは、複数の TLS 終端タイプを使用してクライアントに証明書を提供できます。ルーターは、[edge](#)、[passthrough](#) および [re-encryption](#) 終端をサポートします。

例5.4 セキュリティー保護されていないルートオブジェクト YAML 定義

```
apiVersion: v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name
```

セキュリティー保護されていないルートは、鍵や証明書が必要でないので、設定が最も単純ですが、セキュリティー保護されているルートは、接続を非公開のままにできるのでセキュリティーを確保できます。

Secured ルートは、ルートの TLS 終端が指定されたルートです。利用可能な終端タイプは、[以下で説明されています](#)。

5.7.10.1. パスベースのルート

パスベースのルートは、同じホスト名で、それぞれ異なるパスを使用して複数のルートにサービスを提供できるように、URL と比較可能なパスコンポーネントを指定します (ルートのトラフィックが HTTP ベースでなければならない)。ルーターは、最も限定的なものから順に、ルートを照合する必要がありますが、これはルーターの実装により左右されます。ホスト名とパスは、正常に要求に応答できるように、バックエンドサーバーにパススルーされます。たとえば、<http://example.com/foo/> への要求がルーターに移動すると、Pod に <http://example.com/foo/> への要求が表示されます。

以下の表は、ルートのサンプルおよびそれらのアクセシビリティを示しています。

表5.4 ルートの可用性

Route	比較対象	アクセス可能
<code>www.example.com/test</code>	<code>www.example.com/test</code>	Yes

Route	比較対象	アクセス可能
	www.example.com	No
www.example.com/test and www.example.com	www.example.com/test	Yes
	www.example.com	Yes
www.example.com	www.example.com/test	はい (ルートではなく、ホストに マッチ)
	www.example.com	Yes

例5.5 パスが1つでセキュリティー保護されていないルート

```

apiVersion: v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test" ①
  to:
    kind: Service
    name: service-name

```

① パスは、パスベースのルートに唯一追加される属性です。



注記

ルーターは TLS を終了させず、要求のコンテンツを読み込みことができないので、パスベースのルーティングは、パススルー TLS を使用する場合には利用できません。

5.7.10.2. セキュリティー保護されたルート

セキュリティー保護されたルートは、ルートの TLS 終端を指定し、オプションで鍵と証明書を提供します。



注記

OpenShift Container Platform の TLS 終端は、カスタム証明書を提供する SNI に依存します。ポート 443 で受信する SNI 以外のトラフィックは、TLS 終端およびデフォルトの証明書で処理されます (要求のホスト名と一致せず、バリデーションエラーが発生する可能性があります)。

セキュリティー保護されたルートは、以下の 3 種類のセキュアな TLS 終端を使用できます。

Edge Termination

With edge termination, TLS termination occurs at the router, prior to proxying traffic to its destination. TLS certificates are served by the front end of the router, so they must be configured into the route, otherwise the [router's default certificate](#) will be used for TLS termination.

例5.6 Edge Termination を使用したセキュリティー保護されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured 1
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name 2
  tls:
    termination: edge 3
    key: |- 4
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |- 5
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |- 6
      -----BEGIN CERTIFICATE-----
      [...]
```

1 2 オブジェクトの名前で、63 文字に制限されます。

3 **termination** フィールドは edge termination の **edge** です。

4 **key** フィールドは PEM 形式のキーファイルのコンテンツです。

5 **certificate** フィールドは PEM 形式の証明書ファイルのコンテンツです。

6 オプションの CA 証明書は、検証用に証明書チェーンを確立するために必要になる場合があります。

TLS がルーターで終端されるので、内部ネットワークを使用したルーターからエンドポイントへの接続は暗号化されません。

Edge termination ルートは **insecureEdgeTerminationPolicy** を指定して、セキュアでないスキーム (HTTP) 上にあるトラフィックを無効化、許可、リダイレクトすることができます。 **insecureEdgeTerminationPolicy** で使用できる値は **None** または空 (無効化する場合)、 **Allow** または **Redirect** です。デフォルトの **insecureEdgeTerminationPolicy** は、セキュアでないスキーム上のトラフィックを無効にします。一般的なユースケースは、セキュアなスキームを使用してコンテンツを、セキュアでないスキームを使用してアセット (例のイメージ、スタイルシート、javascript) を提供できるようにします。

例5.7 Edge Termination を使用したセキュリティー保護されたルートでの HTTP トラフィックの許可

```

apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured-allow-insecure ❶
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name ❷
  tls:
    termination: edge ❸
    insecureEdgeTerminationPolicy: Allow ❹
  [...]

```

❶ ❷ オブジェクトの名前で、63 文字に制限されます。

❸ **termination** フィールドは edge termination の **edge** です。

❹ セキュアでないスキーム **HTTP** で送信される要求を許可するセキュアでないポリシー

例5.8 Edge Termination を使用したセキュリティー保護されたルートでの HTTP トラフィックのリダイレクト

```

apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured-redirect-insecure ❶
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name ❷
  tls:
    termination: edge ❸
    insecureEdgeTerminationPolicy: Redirect ❹
  [...]

```

❶ ❷ オブジェクトの名前で、63 文字に制限されます。

❸ **termination** フィールドは edge termination の **edge** です。

❹ セキュアでないスキーム **HTTP** で送信される要求をセキュアなスキーム **HTTPS** にリダイレクトするセキュアでないポリシー

パススルーの停止

passthrough termination では、暗号化されたトラフィックが TLS 終端を提供するルーターなしに宛先に直接送信されます。そのため、鍵や証明書は必要ありません。

例5.9 パススルーの停止を使用したセキュリティー保護されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-passthrough-secured ❶
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name ❷
  tls:
    termination: passthrough ❸
```

❶ ❷ オブジェクトの名前で、63 文字に制限されます。

❸ **termination** フィールド **passthrough** に設定します。他の暗号化フィールドは必要ありません。

宛先 Pod は、エンドポイントでトラフィックに証明書を提供します。これは、必須となるクライアント証明書をサポートするための唯一の方法です (相互認証とも呼ばれる)。



注記

Passthrough ルートには **insecureEdgeTerminationPolicy** を指定できます。唯一有効な値は **None** (無効化する場合は空) または **Redirect** です。

Re-encryption の停止

Re-encryption は、edge termination の一種で、ルーターが証明書を使用して TLS を終端し、異なる証明書が設定されている可能性のあるエンドポイントへの接続を再暗号化します。そのため、内部ネットワークなどを含め、接続の全パスが暗号化されています。ルーターは、ヘルスチェックを使用して、ホストの信頼性を判断します。

例5.10 Re-Encrypt の停止を使用したセキュリティー保護されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-pt-secured ❶
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name ❷
  tls:
    termination: reencrypt ❸
    key: [as in edge termination]
    certificate: [as in edge termination]
```

```
caCertificate: [as in edge termination]
destinationCACertificate: |- 4
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----
```

- 1 2 オブジェクトの名前で、63 文字に制限されます。
- 3 **termination** フィールドは **reencrypt** に設定されます。他のフィールドは edge termination の場合と同じです。
- 4 re-encryption には必須です。**destinationCACertificate** は、エンドポイント証明書を検証する CA 証明書を指定して、ルーターから宛先 Pod への接続のセキュリティを確保します。サービスがサービス署名証明書を使用する場合または、管理者がデフォルトの CA 証明書をルーターに指定し、サービスにその CA により署名された証明書がある場合には、このフィールドは省略可能です。

destinationCACertificate フィールドが空の場合は、ルーターは自動的に証明書を提供するサービス用に生成される認証局を自動的に活用し、すべての Pod に `/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt` として注入します。これにより、ルートの証明書を生成する必要なしに、新しいルートがエンドツーエンドの暗号化を活用できるようになります。これは、管理者が許可しない限り、**destinationCACertificate** が使用できない、カスタムのルーターまたは F5 ルーターの場合に有用です。



注記

Re-encrypt ルートでは **insecureEdgeTerminationPolicy** に、edge termination ルートと同じ値にすべて指定することができます。

5.7.11. ルーターのシャード化

In OpenShift Container Platform, each route can have any number of **labels** in its **metadata** field. A router uses **selectors** (also known as a **selection expression**) to select a subset of routes from the entire pool of routes to serve. A selection expression can also involve labels on the route’s namespace. The selected routes form a **router shard**. You can **create** and **modify** router shards independently from the routes, themselves.

この設計では、**従来の** シャード化も、**重複** シャード化をサポートします。従来のシャード化では、選択した内容が重複セットにならず、ルートはシャード1つのみに所属します。重複シャード化では、選択した内容は重複セットに含まれ、ルートは多数の異なるシャードに所属できます。たとえば、あるルートは **SLA=high** シャード (**SLA=medium** または **SLA=low** シャードではない) や **geo=west** シャード (**geo=east** シャードではない) に所属することができます。

重複シャード化の他の例には、ルートの namespace をベースに選択するルーターセットなどがあります。

ルーター	選択	Namespace
router-1	A* – J*	A*, B*, C*, D*, E*, F*, G*, H*, I*, J*
router-2	K* – T*	K*, L*, M*, N*, O*, P*, Q*, R*, S*, T*

ルーター	選択	Namespace
router-3	Q* – Z*	Q*, R*, S*, T*, U*, V*, W*, X*, Y*, Z*

router-2 および **router-3** は、namespaces **Q***、**R***、**S***、**T*** のルートにサービスを提供します。この例を重複から従来のシャード化に変更するには、**router-2** の選択肢を **K* – P*** に変更して、重複をなくすことができます。

ルートがシャード化されている場合には、指定のルートはこのグループのルーター 0 個以上にバインドされます。ルートをバインドすることで、シャード全体でルートを一意に保つことができます。一意に保つことで、単一のシャード内に、同じルートでもセキュアなバージョンと、セキュアでないバージョンを存在させることができます。つまり、ルートは、作成、バインド、アクティブ化のライフサイクルが可視化されたこととなります。

シャード化の環境では、シャードに到達する最初のルートが再起動の有無にかかわらず、期限なしに存在できる権利を持ちます。

During a green/blue deployment a route may be selected in multiple routers. An OpenShift Container Platform application administrator may wish to bleed traffic from one version of the application to another and then turn off the old version.

シャードリングは、管理者によりクラスターレベルで、ユーザーにより、プロジェクト/namespace レベルで実行できます。namespace ラベルを使用する場合には、ルーターのサービスアカウントには、**cluster-reader** パーミッションを設定して、ルーターが namespace 内のラベルにアクセスできるようにします。



注記

同じホスト名を要求するルートが 2 つ以上ある場合には、解決する順番は、ルートの存在期間を基にし、一番古いルートがホストの要求を優先的に取得します。シャード化されたルーターの場合には、ルートは、ルーターの選択基準にあったラベルをベースに選択されます。ラベルがルートに追加されるタイミングを判断する方法に一貫性はありません。そのため、既存のホスト名を要求する以前のルートが「再度ラベル化されて」、ルーターの選択基準と照合させる場合には、上述の解決順に基づき既存のルートを置き換えます (最も古いルートが優先される)。

5.7.12. 他のバックエンドおよび重み

ルートは通常、**kind: Service** の **to:** トークンを使用したサービスと関連付けられます。ルートへの全要求は、**負荷分散ストラテジー** をベースに、サービス内のエンドポイントにより処理されます。

サービスは最大 4 つまでルートをサポートすることができます。各サービスが処理する要求の大きさは、サービスの **weight** により統制されます。

最初のサービスは、以前と同様に **to:** トークンを使用して入り、サービスは 3 つまで **alternateBackend:** トークンを使用して入ることができます。各サービスは、デフォルトの **kind: Service** が指定されている必要があります。

各サービスには、**weight** が関連付けられています。サービスが処理する要求の大きさは、**weight / sum_of_all_weights** で算出されます。サービスにエンドポイントが複数ある場合には、サービスの重みが 1 以上、各エンドポイントに割り当てられるように、エンドポイント全体に分散されます。サービスの **weight** が 0 の場合は、サービスの各エンドポイントには 0 が割り当てられます。

The **weight** must be in the range 0-256. The default is 1. When the **weight** is 0 no requests are passed to the service. If all services have **weight** 0, requests are returned with a 503 error. When a service has no endpoints, the weight is effectively 0.

alternateBackends を使用すると、**roundrobin** 負荷分散ストラテジーを使用して、要求が想定どおりに **weight** を基にサービスに分散されるようになります。ルートに **roundrobin** を設定する場合は、[ルートアノテーション](#)を使用するか、一般的なルーターには環境変数を使用します。

The following is an example route configuration using alternate backends for [A/B deployments](#).

alternateBackends および重みが指定されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-alternate-service
  annotations:
    haproxy.router.openshift.io/balance: roundrobin ①
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name ②
    weight: 20 ③
  alternateBackends:
  - kind: Service
    name: service-name2 ④
    weight: 10 ⑤
  kind: Service
    name: service-name3 ⑥
    weight: 10 ⑦
```

① This route uses **roundrobin** [load balancing strategy](#)

② 最初のサービス名は **service-name** であり、0 以上の Pod が含まれる可能性があります。

④⑥ alternateBackend サービスにも 0 以上の Pod が含まれる場合があります。

③⑤⑦ **weight** の合計は 40 で、**service-name** は要求の 20/40 または 1/2を、**service-name2** と **service-name3** はそれぞれ、要求の 1/4 を取得し、サービスごとに 1 または複数のエンドポイントが含まれると想定します。

5.7.13. ルート固有のアノテーション

環境変数を使用して、ルーターは、公開する全ルートにデフォルトオプションを設定できます。個別のルートは、アノテーションに個別の設定を指定して、デフォルトの一部を上書きできます。

ルートアノテーション

このセクションで説明されているすべての項目について、ルートがその設定を変更できるように **ルート定義** にアノテーションを設定できます。

表5.5 ルートアノテーション

変数	説明	デフォルトで使用される環境変数
<code>haproxy.router.openshift.io/balance</code>	ロードバランシングアルゴリズムを設定します。使用できるオプションは source 、 roundrobin 、および leastconn です。	passthrough ルートの ROUTER_TCP_BALANCE_SCHEME です。それ以外の場合は ROUTER_LOAD_BALANCE_ALGORITHM を使用します。
<code>haproxy.router.openshift.io/disable_cookies</code>	関連の接続を追跡する cookie の使用を無効にします。 true または TRUE に設定する場合は、分散アルゴリズムを使用して、受信する HTTP 要求ごとに、どのバックエンドが接続を提供するかを選択します。	
<code>haproxy.router.openshift.io/cookie_name</code>	Specifies an optional cookie to be used for this route. The name must consist of any combination of upper and lower case letters, digits, "_", and "-". The default is the hashed internal key name for the route.	
<code>haproxy.router.openshift.io/rate-limit-connections</code>	レート制限機能を有効にするために true または TRUE を設定します。	
<code>haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp</code>	IP アドレスで共有される同時 TCP 接続の数を制限します。	
<code>haproxy.router.openshift.io/rate-limit-connections.rate-http</code>	IP アドレスが HTTP 要求を実行できるレートを制限します。	
<code>haproxy.router.openshift.io/rate-limit-connections.rate-tcp</code>	IP アドレスが TCP 接続を行うレートを制限します。	
<code>haproxy.router.openshift.io/timeout</code>	ルートのサーバー側のタイムアウトを設定します。(TimeUnits)	ROUTER_DEFAULT_SERVER_TIMEOUT
<code>router.openshift.io/haproxy.health.check.interval</code>	バックエンドのヘルスチェックの間隔を設定します。(TimeUnits)	ROUTER_BACKEND_CHECK_INTERVAL
<code>haproxy.router.openshift.io/ip_whitelist</code>	ルートの ホワイトリスト を設定します。	

変数	説明	デフォルトで使用する環境変数
haproxy.router.openshift.io/https_header	edge terminated または re-encrypt ルートの Strick-Transport-Security ヘッダーを設定します。	

例5.11 ルート設定のカスタムタイムアウト

```

apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms 1
[...]
```

- 1** HAProxy 対応の単位 (us、ms、s、m、h、d) で新規のタイムアウトを指定します。単位が指定されていない場合は、ms がデフォルトになります。



注記

passthrough ルートのサーバー側のタイムアウトを低く設定しすぎると、WebSocket 接続がそのルートで頻繁にタイムアウトする可能性があります。

5.7.14. ルート固有の IP ホワイトリスト

選択した IP アドレスだけにルートへのアクセスを制限するには、ルートに **haproxy.router.openshift.io/ip_whitelist** アノテーションを追加します。ホワイトリストは、承認したソースアドレスの IP アドレスまたは/および CIDR をスペース区切りにします。ホワイトリストに含まれていない IP アドレスからの要求は破棄されます。

例:

ルートの編集時に、以下のアノテーションを追加して必要なソース IP を定義します。または、**oc annotate route <name>** を使用します。

唯一の特定の IP アドレスのみを許可します。

```

metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

複数の IP アドレスを許可します。

```

metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12
```

IP CIDR ネットワークを許可します。

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

混在した IP アドレスおよび IP CIDR ネットワークを許可します。

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

5.7.15. ワイルドカードサブドメインポリシーを指定するルートの作成

ワイルドカードポリシーでは、(許可できるようにルーターを設定する場合) ドメイン内の全ホストに対応するルートを定義できます。ルートは、**wildcardPolicy** フィールドを使用して、設定の一部としてワイルドカードポリシーを指定できます。ワイルドカードルートを許可するポリシーが指定されたルーターは、ワイルドカードポリシーを基に適切にルートを公開します。

[Learn how to configure HAProxy routers to allow wildcard routes](#) .

例5.12 サブドメインワイルドカードポリシーを指定するルート

```
apiVersion: v1
kind: Route
spec:
  host: wildcard.example.com ①
  wildcardPolicy: Subdomain ②
  to:
    kind: Service
    name: service-name
```

- ① サービスを公開するために使用される外部から到達可能なホスト名を指定します。
- ② 外部から到達可能なホスト名は、サブドメイン **example.com** 内の全ホストを許可するように指定します。***.example.com** は、公開されたサービスに到達するためのホスト名 **wildcard.example.com** のサブドメインです。

5.7.16. ルートステータス

The **route status** field is only set by routers. If changes are made to a route so that a router no longer serves a specific route, the status becomes stale. The routers do not clear the **route status** field. To remove the stale entries in the route status, use the [clear-route-status script](#).

5.7.17. ルート内の特定ドメインの拒否または許可

ルーターは、**ROUTER_DENIED_DOMAINS** および **ROUTER_ALLOWED_DOMAINS** 環境変数を使用して、ルート内のホスト名からのドメインサブセットを限定して拒否または許可するように設定できます。

ROUTER_DENIED_DOMAINS

一覧表示されるドメインは指定のルートで許可されません。

ROUTER_ALLOWED_DOMAINS

一覧表示されるドメインのみが指定のルートで許可されます。

拒否ドメインの一覧に含まれるドメインは、許可ドメイン一覧よりも優先されます。つまり、OpenShift Container Platform は先に、拒否リスト (該当する場合) をチェックして、ホスト名が拒否ドメイン一覧に含まれていない場合に、許可ドメインをチェックします。ただし、許可ドメインの一覧はより制限されており、ルーターは、その一覧に所属するホストが含まれるルートのみを許可します。

For example, to deny the `[*.]open.header.test`, `[*.]openshift.org` and `[*.]block.it` routes for the **myrouter** route:

```
$ oc adm router myrouter ...
$ oc set env dc/myrouter ROUTER_DENIED_DOMAINS="open.header.test, openshift.org, block.it"
```

これは、**myrouter** がルートの名前に基づいて以下を許可することを意味します。

```
$ oc expose service/<name> --hostname="foo.header.test"
$ oc expose service/<name> --hostname="www.allow.it"
$ oc expose service/<name> --hostname="www.openshift.test"
```

ただし、**myrouter** は以下を拒否します。

```
$ oc expose service/<name> --hostname="open.header.test"
$ oc expose service/<name> --hostname="www.open.header.test"
$ oc expose service/<name> --hostname="block.it"
$ oc expose service/<name> --hostname="franco.baresi.block.it"
$ oc expose service/<name> --hostname="openshift.org"
$ oc expose service/<name> --hostname="api.openshift.org"
```

Alternatively, to block any routes where the host name is **not** set to `[*.]stickshift.org` or `[*.]kates.net`:

```
$ oc adm router myrouter ...
$ oc set env dc/myrouter ROUTER_ALLOWED_DOMAINS="stickshift.org, kates.net"
```

これは、**myrouter** ルートが以下を許可することを意味します。

```
$ oc expose service/<name> --hostname="stickshift.org"
$ oc expose service/<name> --hostname="www.stickshift.org"
$ oc expose service/<name> --hostname="kates.net"
$ oc expose service/<name> --hostname="api.kates.net"
$ oc expose service/<name> --hostname="erno.r.kube.kates.net"
```

ただし、**myrouter** は以下を拒否します。

```
$ oc expose service/<name> --hostname="www.open.header.test"
$ oc expose service/<name> --hostname="drive.ottomatic.org"
$ oc expose service/<name> --hostname="www.wayless.com"
$ oc expose service/<name> --hostname="www.deny.it"
```

To implement both scenarios, run:

```
$ oc adm router adrouter ...
$ oc env dc/adrouter ROUTER_ALLOWED_DOMAINS="openshift.org, kates.net" \
  ROUTER_DENIED_DOMAINS="ops.openshift.org, metrics.kates.net"
```

これにより、ホスト名が **[*.]openshift.org** または **[*.]kates.net** に設定されているルートを許可し、ホスト名が **[*.]ops.openshift.org** または **[*.]metrics.kates.net** に設定されているルートは拒否します。

そのため、以下は拒否されます。

```
$ oc expose service/<name> --hostname="www.open.header.test"
$ oc expose service/<name> --hostname="ops.openshift.org"
$ oc expose service/<name> --hostname="log.ops.openshift.org"
$ oc expose service/<name> --hostname="www.block.it"
$ oc expose service/<name> --hostname="metrics.kates.net"
$ oc expose service/<name> --hostname="int.metrics.kates.net"
```

しかし、以下は許可されます。

```
$ oc expose service/<name> --hostname="openshift.org"
$ oc expose service/<name> --hostname="api.openshift.org"
$ oc expose service/<name> --hostname="m.api.openshift.org"
$ oc expose service/<name> --hostname="kates.net"
$ oc expose service/<name> --hostname="api.kates.net"
```

5.7.18. Namespace 所有権チェックの無効化

ホストとサブドメインは、最初に要求を作成したルートの namespace が所有します。その namespace で作成された他のルートは、サブドメイン上で要求を作成できます。他の namespace はすべて、請求済みのホストおよびサブドメインに要求を作成することはできません。ホストを所有する namespace は、**www.abc.xyz/path1** など、そのホストに関連付けられている全パスも所有します。

たとえば、ホスト **www.abc.xyz** がどのルートからも要求されていない場合に、**ns1** namespace にホストが **www.abc.xyz** のルート **r1** を作成すると、**ns1** の namespace が、ワイルドカードルートのホスト **www.abc.xyz** とサブドメイン **abc.xyz** を所有します。別の namespace **ns2** が異なるパス **www.abc.xyz/path1/path2** で、ルートを作成しようとする、別の namespace (今回は **ns1**) のルートがこのホストを所有するので失敗してしまいます。

With [wildcard routes](#) the namespace that owns the subdomain owns all hosts in the subdomain. If a namespace owns subdomain **abc.xyz** as in the above example, another namespace cannot claim **z.abc.xyz**.

namespace の所有権ルールを無効にするには、これらの制限を無効にして、namespace すべてでホスト (およびサブドメイン) を要求できるようにします。



警告

ルーターで namespace の所有権チェックを無効にする場合には、エンドユーザーが namespace に含まれるホストの所有権を要求できるようになる点に注意してください。この設定の変更は、特定の開発環境で価値がありますが、実稼働環境では慎重にこの機能を使用し、クラスターポリシーで、信頼されないエンドユーザーがルートを作成できないようにロックしてください。

たとえば、`ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true` では、namespace `ns1` が最も古い `r1 www.abc.xyz` を作成する場合に、このホスト名 (+ パス) のみを所有します。別の namespace は、このサブドメイン (`abc.xyz`) に最も古いルートがなくても、ワイルドカードルートを作成することができ、別の namespace が (`foo.abc.xyz`、`bar.abc.xyz`、`baz.abc.xyz` など) ワイルドカード以外の重複ホストを要求することも可能で、この要求は認められます。

別の namespace (例: `ns2`) はルート `r2 www.abc.xyz/p1/p2` を作成でき、許可されます。同様に、別の namespace (`ns3`) は、サブドメインのワイルドカードポリシーでルート `wildthing.abc.xyz` も作成でき、このワイルドカードを所有できます。

この例にあるように、ポリシー `ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true` は制約がゆるく、namespace 全体の要求を許可します。ルーターが namespace の所有を無効にしているルートを許可できるのは、ホストとパスがすでに請求済みである場合だけです。

たとえば、新規ルート `rx` が `www.abc.xyz/p1/p2` を要求しようとする場合に、ルート `r2` はホストとパスの組み合わせを所有しているので拒否されます。まったく同じホストとパスがすでに請求済みなので、これは、ルート `rx` が同じ namespace にある場合も、別の namespace にある場合でも同じです。

この機能は、ルーターの作成時か、またはルーターのデプロイメント設定に環境変数を設定して設定できます。

```
$ oc adm router ... --disable-namespace-ownership-check=true
```

```
$ oc env dc/router ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true
```

[1] これ以降は、デバイス名は、コンテナ B のホストのデバイスを参照します。

第6章 サービスカタログコンポーネント

6.1. サービスカタログ

6.1.1. 概要

マイクロサービスベースのアプリケーションを開発して、クラウドネイティブのプラットフォームで実行する場合には、サービスプロバイダーやプラットフォームに合わせて、異なるリソースをプロビジョニングして、その従属、認証情報、設定を共有する方法は多数あります。

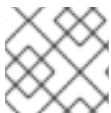
開発者がよりシームレスに作業できるように、OpenShift Container Platform には Kubernetes 向けの [Open Service Broker API \(OSB API\)](#) の実装である **サービスカタログ** が含まれています。これにより、OpenShift Container Platform にデプロイされているアプリケーションをさまざまな種類のサービスブローカーに接続できます。

サービスカタログでは、クラスター管理者が1つの API 仕様を使用して、複数のプラットフォームを統合できます。OpenShift Container Platform Web コンソールは、サービスカタログにサービスブローカーによって提供されるクラスターサービスカタログを表示するので、ユーザーはこれらのサービスをそれぞれのアプリケーションで使用できるようにサービスの検出やインスタンス化を実行できます。

このように、サービスユーザーは異なるプロバイダーが提供する異なるタイプのサービスを簡単かつ一貫性を保ちながら使用できるという利点が得られます。また、サービスプロバイダーは、複数のプラットフォームにアクセスできる統合ポイントから利点を得られます。

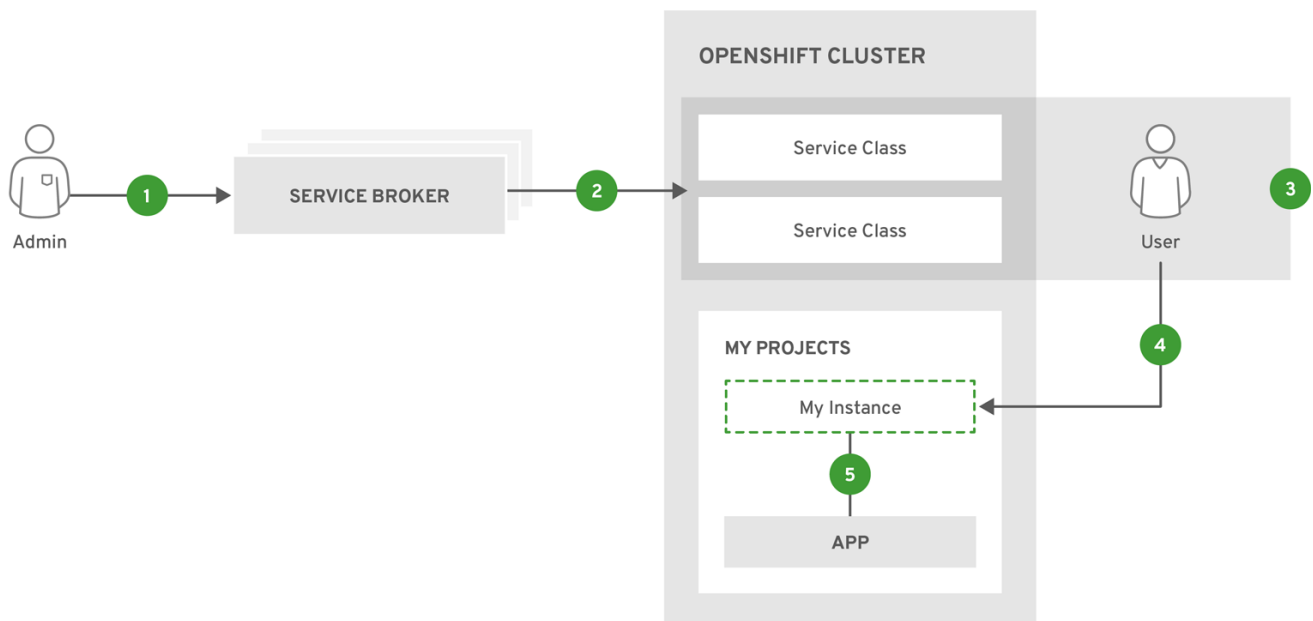
6.1.2. 設計

サービスカタログの設計は基本的なワークフローに基づいています。



注記

以下の新規の用語は「[概念および用語](#)」でさらに定義されています。



OPENSIFT_415489_0218

クラスター管理者は、1つまたは複数の **クラスターサービスブローカー** を OpenShift Container Platform クラスターに登録します。これは、デフォルトで提供されているサービスブローカーでインストール時に自動的に行うことも、手動で行うことも可能です。

各サービスブローカーは、ユーザーに提供すべき **クラスターサービスクラス セット** と、これらのサービスのバリエーション (**サービスプラン**) を、OpenShift Container Platform に指定します。

OpenShift Container Platform web コンソールまたは CLI を使用して、ユーザーは利用可能なサービスを検出します。たとえば、クラスターサービスクラスは、BestDataBase と呼ばれる `database-as-a-service` などの、サービスクラスが利用できる可能性があります。

ユーザーは、クラスターサービスクラスを選択して、自身の新しい **インスタンス** を要求します。たとえば、サービスは **my_db** という名前の BestDataBase インスタンスなどです。

ユーザーは、サービスを pod セット (アプリケーション) にリンクまたは **バインド** します。たとえば、**my_db** サービスインスタンスは、**my_app** と呼ばれるユーザーアプリケーションにバインドできます。

ユーザーがリソースのプロビジョニングおよびプロビジョニング解除を要求すると、この要求はサービスカタログに送信され、次に適切なクラスターサービスブローカーに要求が送信されます。サービスによっては、**provision**、**deprovision** および **update** などの操作に時間がかかる可能性があります。クラスターサービスブローカーが利用できない場合には、サービスカタログはこの操作の再試行を続けます。

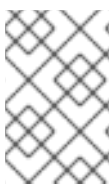
このインフラストラクチャーでは、OpenShift Container Platform で実行中のアプリケーションと、それが使用するサービスの間で疎結合することができます。こうすることで、これらのサービスを使用するアプリケーションが独自のビジネスロジックにフォーカスし、サービスの管理をプロバイダーに任せることができます。

6.1.2.1. リソースの定義

ユーザーがサービスを使用し終えた場合 (または、請求しなくてもいい場合) には、このサービスインスタンスは削除できます。サービスインスタンスを削除するには、サービスのバインドを先に削除する必要があります。サービスのバインドの削除は、**バインド解除** と呼ばれます。削除のプロセスで、削除予定のサービスバインディングを参照するシークレットも削除します。

サービスバインディングが削除されると、サービスインスタンスを削除できなくなります。サービスインスタンスの削除は **プロビジョニング解除** として知られています。

サービスバインディングやサービスインスタンスが含まれるプロジェクトや namespace を削除する場合は、サービスカタログは先にクラスターサービスブローカーに、関連のインスタンスとバインディングを削除するように要求する必要があります。これにより、サービスカタログは、クラスターのサービスブローカーと通信して、プロビジョニング解除を行うまで待つ必要があるため、実際のプロジェクトや namespace の削除が遅延してしまうことが予想されます。通常の場合では、サービスにより異なりますが、数分以上かかる場合があります。



注記

デプロイメントで使用されるサービスバインディングを削除する場合、デプロイメントからバインディングの参照を削除する必要があります。そうしないと、次のロールアウトは失敗します。

6.1.3. 概念および用語

クラスターサービスブローカー

A **cluster service broker** is a server that conforms to the OSB API specification and manages a set of one or more services. The software could be hosted within your own OpenShift Container Platform cluster or elsewhere.

クラスター管理者は、クラスターサービスブローカーを表す **ClusterServiceBroker** API リソースを作成して、OpenShift Container Platform クラスターに登録できます。これにより、クラスター管理者はクラスター内で利用可能なクラスターサービスブローカーを使用して新しい種類の管理済みサービスを作成できるようになります。

ClusterServiceBroker リソースは、クラスターサービスブローカーの接続詳細と、サービスセット（およびこれらのサービスのバリエーション）を OpenShift Container Platform に指定すると、ユーザーに提供できるはずですが、**authInfo** セクションには、クラスターサービスブローカーの認証に使用するデータが含まれている点に特に注意してください。

ClusterServiceBroker リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ClusterServiceBroker
metadata:
  name: BestCompanySaaS
spec:
  url: http://bestdatabase.example.com
  authInfo:
    basic:
      secretRef:
        namespace: test-ns
        name: secret-name
```

クラスターサービスクラス

また、サービスカタログのコンテキストで「サービス」と類義の **クラスターサービスクラス** は、特定のクラスターサービスブローカーが提供する管理サービスの1種です。新しいクラスターサービスブローカーのリソースがクラスターに追加されるたびに、サービスカタログコントローラーは、適切なクラスターサービスブローカーに接続し、サービスオフリングの一覧を取得します。新しい **ClusterServiceClass** リソースは自動的に、クラスターサービスブローカーごとに作成されます。



注記

さらに、OpenShift Container Platform には、**サービス**と呼ばれるコアとなるコンセプトがあります。サービスとは、内部の負荷分散に関連する別個の Kubernetes リソースです。これらのリソースは、サービスカタログや OSB API のコンテキストで使用する用語と混同しないようにしてください。

ClusterServiceClass リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ClusterServiceClass
metadata:
  name: smallDB
  brokerName: BestDataBase
plans: [...]
```

クラスターサービス計画

A **cluster service plan** represents tiers of a cluster service class. For example, a cluster service class may expose a set of plans that offer varying degrees of quality-of-service (QoS), each with a different cost associated with it.

サービスインスタンス

サービスインスタンス は、プロビジョニングされたクラスターサービスクラスのインスタンスです。サービスクラスが提供する機能を使用する場合には、新しいサービスインスタンスを作成してください。

新規の **ServiceInstance** リソースを作成した場合には、サービスカタログコントローラーは、適切なクラスターサービスブローカーと接続して、サービスインスタンスをプロビジョニングするように指示を出します。

ServiceInstance リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceInstance
metadata:
  name: my_db
  namespace: test-ns
spec:
  externalClusterServiceClassName: smallDB
  externalClusterServicePlanName: default
```

アプリケーション

アプリケーション という用語は、**サービスインスタンス** を使用するユーザーのプロジェクトで実行中の pod など、OpenShift Container Platform デプロイメントのアーティファクトを指します。

認証情報

認証情報 とは、サービスインスタンスと通信するアプリケーションに必要な情報です。

サービスバインディング

サービスバインディング は、サービスインスタンスとアプリケーションの間のリンクを指します。サービスバインディングは、アプリケーションからサービスインスタンスを参照して使用できるように希望するクラスターユーザーが作成します。

Upon creation, the service catalog controller creates a Kubernetes secret containing connection details and credentials for the service instance. Such secrets can be mounted into pods as usual. There is also integration with **PodPresets**, which allow you to express how the secret should be consumed, and in which pods.

ServiceBinding リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceBinding
metadata:
  name: myBinding
  namespace: test-ns
spec:
  instanceRef:
    name: my_db
  parameters:
    securityLevel: confidential
  secretName: mySecret
```

パラメーター

パラメーターは、サービスバインディングまたはサービスインスタンスの使用時に、追加のデータをクラスターサービスブローカーに渡すために提供されている特別なフィールドです。唯一のフォーマット要件は、パラメーターを有効なYAML (またはJSON) で指定することです。上記の例では、セキュリティーレベルのパラメーターをサービスバインディング要求でクラスターサービスブローカーに渡します。より高度なセキュリティーが必要なパラメーターの場合は、パラメーターをシークレットに配置し、`parametersFrom` を使用して参照します。

シークレットを参照するサービスバインディングリソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceBinding
metadata:
  name: myBinding
  namespace: test-ns
spec:
  instanceRef:
    name: my_db
  parametersFrom:
    - secretKeyRef:
        name: securityLevel
        key: myKey
    secretName: mySecret
```

6.1.4. 提供されるクラスターサービスブローカー

OpenShift Container Platform は、サービスカタログで使用する以下のクラスターサービスブローカーを提供します。

- [Template Service Broker](#)
- [OpenShift Ansible Broker](#)

6.2. テンプレートサービスブローカー

The **template service broker** (TSB) gives the service catalog visibility into the [default Instant App and Quickstart templates](#) that have shipped with OpenShift Container Platform since its initial release. The TSB can also make available as a service anything for which an OpenShift Container Platform [template](#) has been written, whether provided by Red Hat, a cluster administrator or user, or a third party vendor.

デフォルトでは、TSB は `openshift` プロジェクトからグローバルで入手可能なオブジェクトを表示します。また、クラスター管理者が選択する他のプロジェクトを監視するように設定することも可能です。

6.3. OPENSHIFT ANSIBLE BROKER

6.3.1. 概要

OpenShift Ansible Broker (OAB) は、[Ansible playbook bundle \(APB\)](#) で定義されるアプリケーションを管理する Open Service Broker (OSB) API の実装です。APB は、Ansible ランタイムと、コンテナイメージに同梱されている Ansible playbook のバンドルで構成されており、OpenShift Container Platform のコンテナアプリケーションを定義、配信する新しい方法を提供します。APB は Ansible を活用して、複雑なデプロイするを自動化する標準メカニズムを構築します。

OAB の設計はこの基本的なワークフローに従います。

1. ユーザーは、OpenShift Container Platform Web コンソールを使用してサービスカタログから利用可能なアプリケーションの一覧を要求します。
2. サービスカタログは利用可能なアプリケーションについて OAB に要求します。
3. The OAB communicates with a defined container registry to learn which APBs are available.
4. ユーザーは特定の APB をプロビジョニングする要求を実行します。
5. プロビジョニング要求は OAB に移動し、APB でプロビジョニングメソッドを呼び出して、ユーザー要求に対応します。

6.3.2. Ansible Playbook Bundle

Ansible Playbook Bundle (APB) は、Ansible ロールおよび Playbook の既存の投資を利用できる軽量のアプリケーション定義です。

APB は、名前の付いた playbook が含まれる単純なディレクトリーを使用し、プロビジョニングやバンドなどの OSB API アクションを実行します。apb.yml の仕様ファイルで定義するメタデータには、デプロイメント中に使用する必須/任意のパラメーターの一覧が含まれています。

See the [APB Development Guide](#) for details on the overall design and how APBs are written.