



# OpenShift Container Platform 4.9

## セキュリティおよびコンプライアンス

OpenShift Container Platform のセキュリティについての理解および管理



# OpenShift Container Platform 4.9 セキュリティーおよびコンプライアンス

---

OpenShift Container Platform のセキュリティーについての理解および管理

## 法律上の通知

Copyright © 2023 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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

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

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

All other trademarks are the property of their respective owners.

## 概要

本書では、クラスタのセキュリティー保護に役立つコンテナのセキュリティー、証明書の設定、および暗号の有効化について説明します。

## 目次

<b>第1章 OPENSIFT CONTAINER PLATFORM のセキュリティーおよびコンプライアンス</b> .....	<b>5</b>
1.1. セキュリティーの概要	5
1.2. コンプライアンスの概要	6
1.3. 関連情報	6
<b>第2章 コンテナのセキュリティー</b> .....	<b>7</b>
2.1. コンテナのセキュリティーについて	7
2.2. ホストおよび仮想マシンのセキュリティーについて	9
2.3. RHCOS のハードニング	12
2.4. コンテナイメージの署名	14
2.5. コンプライアンスについて	21
2.6. コンテナのコンテンツのセキュリティー保護	22
2.7. コンテナレジストリーのセキュアな使用	27
2.8. ビルドプロセスのセキュリティー保護	30
2.9. コンテナのデプロイ	34
2.10. コンテナプラットフォームのセキュリティー保護	38
2.11. ネットワークのセキュリティー保護	42
2.12. 割り当てられたストレージのセキュリティー保護	43
2.13. クラスタイベントとログの監視	45
<b>第3章 証明書の設定</b> .....	<b>47</b>
3.1. デフォルトの INGRESS 証明書の置き換え	47
3.2. API サーバー証明書の追加	48
3.3. サービス提供証明書のシークレットによるサービストラフィックのセキュリティー保護	50
3.4. CA バンドルの更新	58
<b>第4章 証明書の種類および説明</b> .....	<b>60</b>
4.1. API サーバーのユーザーによって提供される証明書	60
4.2. プロキシ証明書	60
4.3. サービス CA 証明書	63
4.4. ノード証明書	65
4.5. ブートストラップ証明書	65
4.6. ETCD 証明書	66
4.7. OLM 証明書	66
4.8. 集合 API クライアント証明書	67
4.9. MACHINE CONFIG OPERATOR 証明書	67
4.10. デフォルト INGRESS のユーザーによって提供される証明書	68
4.11. INGRESS 証明書	69
4.12. モニタリングおよび OPENSIFT LOGGING OPERATOR コンポーネント証明書	72
4.13. コントロールプレーンの証明書	72
<b>第5章 コンプライアンス OPERATOR</b> .....	<b>74</b>
5.1. コンプライアンス OPERATOR リリースノート	74
5.2. サポートされているコンプライアンスプロファイル	84
5.3. コンプライアンス OPERATOR のインストール	86
5.4. コンプライアンス OPERATOR の更新	89
5.5. コンプライアンス OPERATOR のスキャン	91
5.6. コンプライアンス OPERATOR について	99
5.7. コンプライアンス OPERATOR の管理	102
5.8. コンプライアンス OPERATOR の調整	104
5.9. コンプライアンス OPERATOR の未加工の結果の取得	108
5.10. コンプライアンス OPERATOR の結果と修復の管理	109

5.11. 高度なコンプライアンス OPERATOR タスクの実行	121
5.12. コンプライアンス OPERATOR のトラブルシューティング	127
5.13. コンプライアンス OPERATOR のアンインストール	137
5.14. OC-COMPLIANCE プラグインの使用	139
5.15. カスタムリソース定義を理解する	145
<b>第6章 FILE INTEGRITY OPERATOR</b>	<b>158</b>
6.1. FILE INTEGRITY OPERATOR リリースノート	158
6.2. FILE INTEGRITY OPERATOR のインストール	160
6.3. FILE INTEGRITY OPERATOR の更新	162
6.4. FILE INTEGRITY OPERATOR について	164
6.5. カスタム FILE INTEGRITY OPERATOR の設定	171
6.6. 高度なカスタム FILE INTEGRITY OPERATOR タスクの実行	175
6.7. FILE INTEGRITY OPERATOR のトラブルシューティング	177
<b>第7章 監査ログの表示</b>	<b>178</b>
7.1. API の監査ログについて	178
7.2. 監査ログの表示	179
7.3. 監査ログのフィルター	182
7.4. 監査ログの収集	183
7.5. 関連情報	183
<b>第8章 監査ログポリシーの設定</b>	<b>185</b>
8.1. 監査ログポリシープロファイルについて	185
8.2. 監査ログポリシーの設定	186
8.3. カスタムルールによる監査ログポリシーの設定	187
8.4. 監査ロギングの無効化	188
<b>第9章 TLS セキュリティープロファイルの設定</b>	<b>191</b>
9.1. TLS セキュリティープロファイルについて	191
9.2. TLS セキュリティープロファイルの詳細表示	192
9.3. INGRESS コントローラーの TLS セキュリティープロファイルの設定	194
9.4. コントロールプレーンの TLS セキュリティープロファイルの設定	196
9.5. KUBELET の TLS セキュリティープロファイルの設定	199
<b>第10章 SECCOMP プロファイルの設定</b>	<b>202</b>
10.1. すべての POD のデフォルトの SECCOMP プロファイルを有効にする	202
10.2. カスタム SECCOMP プロファイルの設定	203
10.3. 関連情報	205
<b>第11章 追加ホストから API サーバーへの JAVASCRIPT ベースのアクセスの許可</b>	<b>206</b>
11.1. 追加ホストから API サーバーへの JAVASCRIPT ベースのアクセスの許可	206
<b>第12章 ETCD データの暗号化</b>	<b>208</b>
12.1. ETCD 暗号化について	208
12.2. ETCD 暗号化の有効化	208
12.3. ETCD 暗号化の無効化	210
<b>第13章 POD の脆弱性のスキャン</b>	<b>212</b>
13.1. RED HAT QUAY CONTAINER SECURITY OPERATOR の実行	212
13.2. CLI でのイメージ脆弱性のクエリー	214
<b>第14章 NETWORK-BOUND DISK ENCRYPTION (NBDE)</b>	<b>216</b>
14.1. ディスクの暗号化技術について	216
14.2. TANG サーバーのインストールに関する考慮事項	221
14.3. TANG サーバーの暗号化キー管理	223







# 第1章 OPENSIFT CONTAINER PLATFORM のセキュリティーおよびコンプライアンス

## 1.1. セキュリティーの概要

OpenShift Container Platform クラスターの各種の側面を適切に保護する方法を理解しておくことが重要です。

### コンテナのセキュリティー

OpenShift Container Platform セキュリティーを理解するためのスタート地点として、[コンテナのセキュリティーについて](#) の概念を確認すると良いでしょう。本セクションとこの後のセクションでは、OpenShift Container Platform で有効なコンテナのセキュリティー対策についての概要を説明します。これには、ホスト層、コンテナとオーケストレーション層、およびビルドとアプリケーション層の各種ソリューションが含まれます。これらのセクションでは、以下のトピックについても説明します。

- コンテナのセキュリティーが重要である理由、および既存のセキュリティー標準との違い。
- ホスト (RHCOS および RHEL) 層で提供されるコンテナのセキュリティー対策と OpenShift Container Platform で提供されるコンテナのセキュリティー対策。
- 脆弱性についてコンテナのコンテンツとソースを評価する方法。
- コンテナのコンテンツをプロアクティブに検査できるようにビルドおよびデプロイメントプロセスを設計する方法。
- 認証および認可によってコンテナへのアクセスを制御する方法。
- OpenShift Container Platform でネットワークと割り当て済みストレージのセキュリティーを保護する方法。
- API 管理および SSO のコンテナ化ソリューション。

### 監査

OpenShift Container Platform 監査は、システムに影響を与えた一連のアクティビティーを個別のユーザー、管理者その他システムのコンポーネント別に記述したセキュリティー関連の時系列のレコードを提供します。管理者は [監査ログポリシーの設定](#) と [監査ログの表示](#) が可能です。

### 証明書

証明書は、クラスターへのアクセスを検証するためにさまざまなコンポーネントによって使用されます。管理者は、[デフォルトの Ingress 証明書の置き換え](#)、[API サーバー証明書の追加](#)、または [サービス証明書の追加](#) が可能です。

クラスターで使用される証明書の種類の詳細を確認することもできます。

- [API サーバーのユーザーによって提供される証明書](#)
- [プロキシ証明書](#)
- [サービス CA 証明書](#)
- [ノード証明書](#)
- [ブートストラップ証明書](#)

- [etcd 証明書](#)
- [OLM 証明書](#)
- [集合 API クライアント証明書](#)
- [Machine Config Operator 証明書](#)
- [デフォルト ingress のユーザーによって提供される証明書](#)
- [Ingress 証明書](#)
- [モニターリングおよびクラスターロギング Operator コンポーネント証明書](#)
- [コントロールプレーンの証明書](#)

### データの暗号化

クラスターの [etcd 暗号化を有効](#) にして、データセキュリティの層を追加で提供することができます。たとえば、etcd バックアップが正しくない公開先に公開される場合に機密データが失われないように保護することができます。

### 脆弱性スキャン

管理者は Red Hat Quay Container Security Operator を使用して [vulnerability scans](#) を実行し、検出された脆弱性の情報を確認できます。

## 1.2. コンプライアンスの概要

多くの OpenShift Container Platform のお客様においては、システムが実稼働環境で使用される前に、一定レベルでの規制への対応またはコンプライアンスが必要になります。この規制対応は、国家標準、業界標準または組織の企業ガバナンスフレームワークによって課せられます。

### コンプライアンスの確認

管理者は [コンプライアンス Operator](#) を使用してコンプライアンススキャンを実行し、検出された問題の修復を提案できます。[oc-compliance プラグイン](#) は、コンプライアンス Operator を簡単に操作するための一連のユーティリティーを提供する OpenShift CLI (**oc**) プラグインです。

### ファイルの整合性チェック

管理者は [File Integrity Operator](#) を使用して、クラスターノードでファイルの整合性チェックを継続的に実行し、変更されたファイルのログを提供できます。

## 1.3. 関連情報

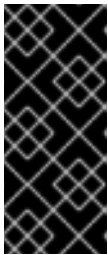
- [認証について](#)
- [内部 OAuth サーバーの設定](#)
- [アイデンティティプロバイダー設定について](#)
- [RBAC の使用によるパーミッションの定義および適用](#)
- [Security Context Constraints の管理](#)

## 第2章 コンテナのセキュリティ

### 2.1. コンテナのセキュリティについて

コンテナ化されたアプリケーションのセキュリティ保護においては、複数のセキュリティレベルが関係します。

- コンテナのセキュリティは、信頼できるベースコンテナイメージから始まり、CI/CD パイプラインを通過するためにコンテナのビルドプロセスまで適用されます。



#### 重要

デフォルトでは、イメージストリームは自動的に更新されません。このデフォルトの動作では、イメージストリームによって参照されるイメージに対するセキュリティ更新は自動的に行われなため、セキュリティの問題が発生する可能性があります。このデフォルト動作を上書きする方法の詳細は、[イメージストリームタグの定期的なインポートの設定](#) について参照してください。

- コンテナがデプロイされると、そのセキュリティはセキュアなオペレーティングシステムやネットワーク上で実行されているかどうかによって依存し、コンテナ自体とこれと対話するユーザーやホスト間に明確な境界を確立することが必要です。
- セキュリティを継続して保護できるかどうかは、コンテナイメージをスキャンして脆弱性の有無を確認でき、脆弱なイメージを効率的に修正し、置き換える効率的な方法があるかどうかによって依存します。

OpenShift Container Platform などのプラットフォームが追加設定なしで提供する内容のほかに、各組織には独自のセキュリティ需要がある可能性があります。OpenShift Container Platform をデータセンターにデプロイする前にも、一定レベルのコンプライアンス検証が必要になる場合があります。

同様に、独自のエージェント、特殊ハードウェアドライバーまたは暗号化機能を OpenShift Container Platform に追加して組織のセキュリティ基準を満たす必要がある場合があります。

本書では、OpenShift Container Platform で有効なコンテナのセキュリティ対策についての概要を説明します。これには、ホスト層、コンテナとオーケストレーション層、およびビルドとアプリケーション層の各種ソリューションが含まれます。次に、これらのセキュリティ対策を実行するのに役立つ特定の OpenShift Container Platform ドキュメントを参照します。

本書には、以下の情報が記載されています。

- コンテナのセキュリティが重要である理由、および既存のセキュリティ標準との違い。
- ホスト (RHCOS および RHEL) 層で提供されるコンテナのセキュリティ対策と OpenShift Container Platform で提供されるコンテナのセキュリティ対策。
- 脆弱性についてコンテナのコンテンツとソースを評価する方法。
- コンテナのコンテンツをプロアクティブに検査できるようにビルドおよびデプロイメントプロセスを設計する方法。
- 認証および認可によってコンテナへのアクセスを制御する方法。
- OpenShift Container Platform でネットワークと割り当て済みストレージのセキュリティを保護する方法。

- API 管理および SSO のコンテナ化ソリューション。

本書の目的は、コンテナ化されたワークロードに OpenShift Container Platform を使用するセキュリティー上の重要な利点と、Red Hat エコシステム全体がコンテナのセキュリティーを確保し、維持する際にどのようなロールを果たしているかについて理解を促すことにあります。また、OpenShift Container Platform の使用により組織のセキュリティー関連の目標を達成する方法について理解するのに役立ちます。

### 2.1.1. コンテナについて

コンテナは、アプリケーションとそのすべての依存関係を1つのイメージにパッケージ化します。このイメージは、変更なしに開発環境からテスト環境、実稼働環境へとプロモートすることができます。コンテナは、他のコンテナと密接に動作する大規模なアプリケーションの一部である可能性があります。

コンテナは、複数の環境、および物理サーバー、仮想マシン (VM)、およびプライベートまたはパブリッククラウドなどの複数のデプロイメントターゲット間に一貫性をもたらします。

コンテナを使用するメリットには以下が含まれます。

インフラストラクチャー	アプリケーション
共有される Linux オペレーティングシステムのカーネル上でのアプリケーションプロセスのサンドボックス化	アプリケーションとそのすべての依存関係のパッケージ化
仮想マシンを上回る単純化、軽量化、高密度化の実現	すべての環境に数秒でデプロイが可能。CI/CD の実現
複数の異なる環境間での移植性	コンテナ化されたコンポーネントへのアクセスと共有が容易になる

Linux コンテナについての詳細は、Red Hat カスタマーポータル上にある [Understanding Linux containers](#) を参照してください。RHEL コンテナについての詳細は、RHEL 製品ドキュメントの [Building, running, and managing containers](#) を参照してください。

### 2.1.2. OpenShift Container Platform について

コンテナ化されたアプリケーションがデプロイされ、実行され、管理される方法を自動化することは、OpenShift Container Platform をはじめとするプラットフォームのジョブです。OpenShift Container Platform は、コアとして Kubernetes プロジェクトに依存し、スケーラブルなデータセンターの多数のノード間でコンテナをオーケストレーションするエンジンを提供します。

Kubernetes は、複数の異なるオペレーティングシステムおよびアドオンのコンポーネントを使用して実行できるプロジェクトです。これらのオペレーティングシステムおよびアドオンコンポーネントは、プロジェクトでのサポート容易性を保証していません。そのため、Kubernetes プラットフォームによって、セキュリティーの内容が異なる可能性があります。

OpenShift Container Platform は、Kubernetes セキュリティーをロックダウンし、プラットフォームを各種の拡張コンポーネントと統合するように設計されています。このため、OpenShift Container Platform は、オペレーティングシステム、認証、ストレージ、ネットワーク、開発ツール、ベースコンテナイメージ、その他の多くのコンポーネントを含む、各種オープンソース技術の大規模な Red Hat エコシステムを利用します。

OpenShift Container Platform には、プラットフォーム自体およびプラットフォーム上で実行されるコンテナ化されたアプリケーションの脆弱性の発見、およびその脆弱性に対する修正の迅速なデプロイにおける Red Hat の豊富な経験が最大限に活用されます。また、Red Hat は、新規コンポーネントが利用可能になる時点でそれらのコンポーネントを OpenShift Container Platform に効率的に統合し、各種テクノロジーをお客様の個々のニーズに適応させる点においても多くの経験があります。

## 関連情報

- [OpenShift Container Platform アーキテクチャー](#)
- [OpenShift セキュリティーガイド](#)

## 2.2. ホストおよび仮想マシンのセキュリティーについて

コンテナと仮想マシンはいずれも、ホストで実行されているアプリケーションをオペレーティングシステム自体から分離する方法を提供します。RHCOS (OpenShift Container Platform で使用されるオペレーティングシステム) についての理解は、ホストシステムがコンテナおよびホストを相互から保護する方法を確認する際に役立ちます。

### 2.2.1. Red Hat Enterprise Linux CoreOS (RHCOS) でのコンテナのセキュリティー保護

コンテナは、それぞれのコンテナを起動するために同じカーネルおよびコンテナランタイムを使用して、同じホストで実行される多数のアプリケーションのデプロイメントを単純化します。アプリケーションは多くのユーザーが所有できます。これらのアプリケーションを分離した状態に維持し、これらのアプリケーションの別々のバージョン、また互換性のないバージョンも問題なく同時に実行できるためです。

Linux では、コンテナは特殊なタイプのプロセスに過ぎないため、コンテナのセキュリティーを保護することは、他の実行中のプロセスのセキュリティーを保護することと同じです。コンテナを実行する環境は、オペレーティングシステムで起動します。このオペレーティングシステムでは、ホストで実行しているコンテナや他のプロセスからホストカーネルのセキュリティーを保護するだけでなく、複数のコンテナのセキュリティーを相互から保護できる必要があります。

OpenShift Container Platform 4.9 は RHCOS ホストで実行され、Red Hat Enterprise Linux (RHEL) をワーカーノードとして使用するオプションが指定されるため、デフォルトで以下の概念がデプロイされた OpenShift Container Platform クラスターに適用されます。これらの RHEL セキュリティー機能は、OpenShift Container Platform で実行中のコンテナのセキュリティーを強化するためのコアとなる機能です。

- **Linux namespace** は特定のグローバルシステムリソースを抽象化し、これを namespace 内の複数のプロセスに対して分離したインスタンスとして表示できます。これにより、複数のコンテナが競合せずに同じコンピューティングリソースを同時に使用することができます。デフォルトでホストから分離されているコンテナの namespace には、マウントテーブル、プロセステーブル、ネットワークインターフェイス、ユーザー、コントロールグループ、UTS、および IPC namespace が含まれます。ホスト namespace に直接アクセスする必要があるコンテナには、そのアクセスを要求するために特権昇格が必要です。namespace のタイプについての詳細は、RHEL 8 コンテナのドキュメントの [Overview of Containers in Red Hat Systems](#) を参照してください。
- **SELinux** はセキュリティーの層を追加し、コンテナを相互に、またホストから分離させます。SELinux により、管理者は、それぞれのユーザー、アプリケーション、プロセスおよびファイルに対して強制アクセス制御 (MAC) を実施できます。



## 警告

RHCOS での SELinux の無効化はサポートされていません。

- **CGroup** (コントロールグループ) はプロセスのコレクションについてのリソースの使用 (CPU、メモリー、ディスク I/O、ネットワークなど) を制限し、設定し、分離します。CGroup は、同じホスト上のコンテナーが相互に影響を与えないようにするために使用されます。
- **Secure computing mode (seccomp)** プロファイルは、利用可能なシステム呼び出しを制限するためにコンテナーに関連付けることができます。seccomp についての詳細は、[OpenShift Security Guide](#) の 94 ページを参照してください。
- **RHCOS** を使用したコンテナーのデプロイは、ホスト環境を最小化してコンテナー向けに調整することで、攻撃される対象の規模を縮小します。[CRI-O コンテナーエンジン](#) は、デスクトップ指向のスタンドアロン機能を実装する他のコンテナーエンジンとは対照的に、Kubernetes および OpenShift Container Platform が必要とする機能のみを実装してコンテナーを実行し、管理することで、その攻撃対象領域をさらに削減します。

RHCOS は、OpenShift Container Platform クラスターでコントロールプレーン (マスター) およびワーカーノードとして機能するように特別に設定された Red Hat Enterprise Linux (RHEL) のバージョンです。そのため、RHCOS は、Kubernetes および OpenShift Container Platform サービスと共にコンテナーのワークロードを効率的に実行するように調整されます。

OpenShift Container Platform クラスターの RHCOS システムをさらに保護するには、ホストシステム自体の管理またはモニターリングを行うコンテナーを除き、ほとんどのコンテナーを root 以外のユーザーとして実行する必要があります。権限レベルを下げたり、付与する権限を可能な限り低くしてコンテナーを作成することが、独自の OpenShift Container Platform クラスターを保護する方法として推奨されます。

## 関連情報

- [ノードによるリソースの制約の適用方法](#)
- [Security Context Constraints の管理](#)
- [OpenShift クラスターでサポートされるプラットフォーム](#)
- [ユーザーによってプロビジョニングされるインフラストラクチャーを使用したクラスターの要件](#)
- [RHCOS の設定方法の選択](#)
- [Ignition](#)
- [カーネル引数](#)
- [カーネルモジュール](#)
- [FIPS 暗号](#)
- [ディスクの暗号化](#)



- [Chrony タイムサービス](#)
- [OpenShift Update Service について](#)

### 2.2.2. 仮想化とコンテナの比較

従来の仮想化は、アプリケーション環境を同じ物理ホスト上で分離させた状態にするためのもう1つの方法です。ただし、仮想マシンはコンテナとは異なる方法で動作します。仮想化は、ゲスト仮想マシン (VM) を起動するハイパーバイザーを使用します。仮想マシンにはそれぞれ、実行中のカーネルで代表される独自のオペレーティングシステム (OS) のほか、実行されるアプリケーションとその依存関係があります。

仮想マシンの場合、ハイパーバイザーはゲスト同士を分離させ、ゲストをホストカーネルから分離します。ハイパーバイザーにアクセスする個々のユーザーおよびプロセスの数は少ないため、物理サーバーで攻撃される対象の規模が縮小します。ただし、この場合もセキュリティの監視が依然として必要になります。あるゲスト仮想マシンがハイパーバイザーのバグを利用して、別の仮想マシンまたはホストカーネルにアクセスできる可能性があります。また、OS にパッチを当てる必要がある場合は、その OS を使用するすべてのゲスト仮想マシンにパッチを当てる必要があります。

コンテナはゲスト仮想マシン内で実行可能であり、これが必要になる場合のユースケースもあるでしょう。たとえば、リフトアンドシフト方式でアプリケーションをクラウドに移行するなど、コンテナに従来型のアプリケーションをデプロイする場合などです。

しかし、単一ホストでのコンテナの分離は、柔軟性があり、スケーリングしやすいデプロイメントソリューションを提供します。このデプロイメントモデルは、クラウドネイティブなアプリケーションにとくに適しています。コンテナは通常、仮想マシンよりもはるかに小さいため、メモリーと CPU の消費量が少なくなります。

コンテナと仮想マシンの違いについては、RHEL 7 コンテナドキュメントの [Linux Containers Compared to KVM Virtualization](#) を参照してください。

### 2.2.3. OpenShift Container Platform のセキュリティ保護

OpenShift Container Platform をデプロイする際に、インストーラーでプロビジョニングされるインフラストラクチャー (利用可能ないくつかのプラットフォーム) またはユーザーによってプロビジョニングされるインフラストラクチャーを選択できます。FIPS コンプライアンスの有効化や初回の起動時に必要なカーネルモジュールの追加など、低レベルのセキュリティ関連の設定は、ユーザーによってプロビジョニングされるインフラストラクチャーの場合に役立つ場合があります。同様に、ユーザーによってプロビジョニングされるインフラストラクチャーは、非接続の OpenShift Container Platform デプロイメントに適しています。

セキュリティが強化され、OpenShift Container Platform に他の設定変更が行われる場合、以下を含む目標を明確にするようにしてください。

- 基礎となるノードを可能な限り汎用的な状態で維持する。同様のノードをすぐ、かつ指定した方法で破棄したり起動したりできるようにする必要があります。
- ノードに対して直接的に1回限りの変更を行うのではなく、OpenShift Container Platform でのノードへの変更をできる限り管理する。

上記を目標とすると、ほとんどのノードの変更はインストール時に Ignition で行うか、または Machine Config Operator によってノードのセットに適用される MachineConfig を使用して後で行う必要があります。この方法で実行できるセキュリティ関連の設定変更の例を以下に示します。

- カーネル引数の追加

- カーネルモジュールの追加
- FIPS 暗号のサポートの有効化
- ディスク暗号化の設定
- chrony タイムサービスの設定

Machine Config Operator のほかにも、Cluster Version Operator (CVO) によって管理される OpenShift Container Platform インフラストラクチャーの設定に使用できる他の Operator が複数あります。CVO は、OpenShift Container Platform クラスタ更新の多くの部分を自動化できます。

## 2.3. RHCOS のハードニング

RHCOS は、OpenShift Container Platform にデプロイするように作成され、調整されました。RHCOS ノードへの変更はほとんど不要です。OpenShift Container Platform を採用するすべての組織には、システムハードニングに関する独自の要件があります。OpenShift 固有の変更および機能 (Ignition、ostree、読み取り専用 `usr` など) が追加された RHEL システムとして、RHCOS を RHEL システムと同様に強化できます。ハードニングの管理方法には違いがあります。

OpenShift Container Platform およびその Kubernetes エンジンの主要機能は、必要に応じてアプリケーションおよびインフラストラクチャーを迅速にスケールアップおよびダウンできることです。避けられない状況でない限り、ホストにログインしてソフトウェアを追加したり設定を変更したりして RHCOS に直接変更を加える必要はありません。OpenShift Container Platform インストーラーおよびコントロールプレーンで RHCOS への変更を管理し、手動による介入なしに新規ノードを起動できるようにする必要があります。

そのため、独自のセキュリティ上のニーズに対応するために OpenShift Container Platform で RHCOS ノードをハードニングする場合、ハードニングする内容とハードニング方法の両方を考慮する必要があります。

### 2.3.1. RHCOS でのハードニングの内容の選択

[RHEL 8 セキュリティ強化](#) ガイドでは、RHEL システムのセキュリティのアプローチについて説明しています。

本書では、暗号化のアプローチ、脆弱性の評価方法、および各種サービスへの脅威の評価方法について説明します。また、コンプライアンス基準についてのスキャン、ファイルの整合性の確認、監査の実行、およびストレージデバイスの暗号化の方法を確認することができます。

ハードニングする機能についての理解に基づいて、RHCOS でそれらをハードニングする方法を決定することができます。

### 2.3.2. RHCOS のハードニング方法の選択

OpenShift Container Platform での RHCOS システムの直接的な変更は推奨されません。代わりに、ワーカーノードやコントロールプレーンノードなどのノードのプールにあるシステムを変更することについて考慮する必要があります。新規ノードが必要な場合、ベアメタル以外のインストールでは、必要なタイプの新規ノードを要求でき、ノードは RHCOS イメージおよび先に行った変更に基づいて作成されます。

インストール前や、インストール時、およびクラスタの稼働後に RHCOS を変更することができます。

#### 2.3.2.1. インストール前のハードニング



ベアメタルのインストールでは、OpenShift Container Platform のインストールを開始する前にハードニング機能を RHCOS に追加できます。たとえば、RHCOS インストーラーの起動時に、SELinux ブール型や対称マルチスレッドなどの各種の低レベル設定などのセキュリティー機能をオンまたはオフにするためにカーネルオプションを追加できます。



### 警告

RHCOS ノードでの SELinux の無効化はサポートされていません。

ベアメタル RHCOS のインストールの場合は難易度が上がりますが、この場合、OpenShift Container Platform インストールを開始する前にオペレーティングシステムの変更を取得することができます。これは、ディスクの暗号化や特別なネットワーク設定など、特定の機能を可能な限り早期に設定する必要がある場合に重要になります。

#### 2.3.2.2. インストール時のハードニング

OpenShift Container Platform インストールプロセスを中断し、Ignition 設定を変更できます。Ignition 設定を使用して、独自のファイルおよび systemd サービスを RHCOS ノードに追加できます。また、インストールに使用する **install-config.yaml** ファイルに基本的なセキュリティー関連の変更を加えることもできます。この方法で追加した内容は、各ノードの初回起動時に利用できます。

#### 2.3.2.3. クラスターの実行後のハードニング

OpenShift Container Platform クラスターの起動後にハードニング機能を RHCOS に適用する方法は複数あります。

- デモンセット: すべてのノードでサービスを実行する必要がある場合は、そのサービスを [Kubernetes DaemonSet オブジェクト](#) で追加できます。
- マシン設定: **MachineConfig** オブジェクトには、同じ形式の Ignition 設定のサブセットが含まれます。マシン設定をすべてのワーカーノードまたはコントロールプレーンノードに適用することで、クラスターに追加される同じタイプの次のノードで同じ変更が適用されるようにできます。

ここで説明しているすべての機能は、OpenShift Container Platform の製品ドキュメントに記載されています。

#### 関連情報

- [OpenShift セキュリティーガイド](#)
- [RHCOS の設定方法の選択](#)
- [ノードの変更](#)
- [インストール設定ファイルの手動作成](#)
- [Kubernetes マニフェストおよび Ignition 設定ファイルの作成](#)
- [ISO イメージを使用した RHCOS のインストール](#)

- [ノードのカスタマイズ](#)
- [カーネル引数のノードへの追加](#)
- [インストール設定パラメーター: fips](#) を参照
- [FIPS 暗号のサポート](#)
- [RHEL コア crypto コンポーネント](#)

## 2.4. コンテナイメージの署名

Red Hat は、Red Hat Container Registry でイメージの署名を提供します。これらの署名は、Machine Config Operator (MCO) を使用して OpenShift Container Platform 4 クラスターにプルされる際に自動的に検証されます。

[Quay.io](#) は OpenShift Container Platform を設定するほとんどのイメージを提供し、リリースイメージのみが署名されます。リリースイメージは承認済みの OpenShift Container Platform イメージを参照するため、サプライチェーン攻撃からの一定レベルの保護が得られます。ただし、ロギング、モニタリング、サービスメッシュなどの OpenShift Container Platform への拡張機能の一部は、Operator Lifecycle Manager (OLM) から Operator として提供されます。それらのイメージは、[Red Hat Ecosystem Catalog Container イメージ](#) レジストリーから提供されます。

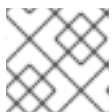
Red Hat レジストリーとインフラストラクチャー間のイメージの整合性を確認するには、署名の検証を有効にします。

### 2.4.1. Red Hat Container Registry の署名の検証の有効化

Red Hat Container レジストリーのコンテナ署名の検証を有効にするには、これらのレジストリーからのイメージを検証するキーを指定する署名検証ポリシーファイルを作成する必要があります。RHEL8 ノードの場合、レジストリーはデフォルトで `/etc/containers/registries.d` にすでに定義されています。

#### 手順

1. ワーカーノードに必要な設定を含む Butane 設定ファイル `51-worker-rh-registry-trust.bu` を作成します。



#### 注記

Butane の詳細は、Butane を使用したマシン設定の作成を参照してください。

```
variant: openshift
version: 4.9.0
metadata:
  name: 51-worker-rh-registry-trust
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
  - path: /etc/containers/policy.json
    mode: 0644
    overwrite: true
  contents:
    inline: |
```

```

{
  "default": [
    {
      "type": "insecureAcceptAnything"
    }
  ],
  "transports": {
    "docker": {
      "registry.access.redhat.com": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
        }
      ],
      "registry.redhat.io": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
        }
      ]
    },
    "docker-daemon": {
      "": [
        {
          "type": "insecureAcceptAnything"
        }
      ]
    }
  }
}

```

2. Butane を使用して、ワーカーノードのディスクに書き込まれるファイルが含まれるように、マシン設定 YAML ファイル **51-worker-rh-registry-trust.yaml** を生成します。

```
$ butane 51-worker-rh-registry-trust.bu -o 51-worker-rh-registry-trust.yaml
```

3. 作成されたマシン設定を適用します。

```
$ oc apply -f 51-worker-rh-registry-trust.yaml
```

4. ワーカーマシン設定プールが新しいマシン設定でロールアウトされていることを確認します。
  - a. 新しいマシン設定が作成されたことを確認します。

```
$ oc get mc
```

### 出力例

NAME	GENERATEDBYCONTROLLER
IGNITIONVERSION AGE	
00-master	a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0 25m	

```

00-worker                                a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0      25m
01-master-container-runtime
a2178ad522c49ee330b0033bb5cb5ea132060b0a 3.2.0      25m
01-master-kubelet                        a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0      25m
01-worker-container-runtime
a2178ad522c49ee330b0033bb5cb5ea132060b0a 3.2.0      25m
01-worker-kubelet                        a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0      25m
51-master-rh-registry-trust              3.2.0      13s
51-worker-rh-registry-trust              3.2.0      53s ①
99-master-generated-crio-seccomp-use-default 3.2.0
25m
99-master-generated-registries
a2178ad522c49ee330b0033bb5cb5ea132060b0a 3.2.0      25m
99-master-ssh                            3.2.0      28m
99-worker-generated-crio-seccomp-use-default 3.2.0
25m
99-worker-generated-registries
a2178ad522c49ee330b0033bb5cb5ea132060b0a 3.2.0      25m
99-worker-ssh                            3.2.0      28m
rendered-master-af1e7ff78da0a9c851bab4be2777773b
a2178ad522c49ee330b0033bb5cb5ea132060b0a 3.2.0      8s
rendered-master-cd51fd0c47e91812bfef2765c52ec7e6
a2178ad522c49ee330b0033bb5cb5ea132060b0a 3.2.0      24m
rendered-worker-2b52f75684fbc711bd1652dd86fd0b82
a2178ad522c49ee330b0033bb5cb5ea132060b0a 3.2.0      24m
rendered-worker-be3b3bce4f4aa52a62902304bac9da3c
a2178ad522c49ee330b0033bb5cb5ea132060b0a 3.2.0      48s ②

```

- ① 新しいマシン設定
- ② 新しいレンダリングされたマシン設定

b. ワーカーマシン設定プールが新しいマシン設定で更新されていることを確認します。

```
$ oc get mcp
```

### 出力例

```

NAME CONFIG                                UPDATED UPDATING DEGRADED
MACHINECOUNT READYMACHINECOUNT UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT AGE
master rendered-master-af1e7ff78da0a9c851bab4be2777773b True False
False 3 3 3 0 30m
worker rendered-worker-be3b3bce4f4aa52a62902304bac9da3c False True
False 3 0 0 0 30m ①

```

- ① **UPDATING** フィールドが **True** の場合、マシン設定プールは新しいマシン設定で更新されます。フィールドが **False** になると、ワーカーマシン設定プールが新しいマシン設定にロールアウトされます。

5. クラスタが RHEL7 ワーカーノードを使用している場合、ワーカーマシンの設定プールが更新されたら、それらのノードに YAML ファイルを `/etc/containers/registries.d` ディレクトリーに作成します。これにより、特定のレジストリーサーバーの切り離された署名の場所が指定されます。次の例は、**registry.access.redhat.com** および **registry.redhat.io** でホストされているイメージに対してのみ機能します。

- a. 各 RHEL7 ワーカーノードへのデバッグセッションを開始します。

```
$ oc debug node/<node_name>
```

- b. ルートディレクトリーを `/host` に変更します。

```
sh-4.2# chroot /host
```

- c. 以下を含む `/etc/containers/registries.d/registry.redhat.io.yaml` ファイルを作成します。

```
docker:
  registry.redhat.io:
    sigstore: https://registry.redhat.io/containers/sigstore
```

- d. 以下を含む `/etc/containers/registries.d/registry.access.redhat.com.yaml` ファイルを作成します。

```
docker:
  registry.access.redhat.com:
    sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

- e. デバッグセッションを終了します。

### 2.4.2. 署名の検証設定の確認

マシン設定をクラスタに適用すると、Machine Config Controller は新規の **MachineConfig** オブジェクトを検出し、新規の **rendered-worker-`<hash>`** バージョンを生成します。

#### 前提条件

- マシン設定ファイルを使用して署名の検証を有効にしている。

#### 手順

1. コマンドラインで以下のコマンドを実行し、必要なワーカーの情報を表示します。

```
$ oc describe machineconfigpool/worker
```

#### 初期ワーカーモニターリングの出力例

```
Name:      worker
Namespace:
Labels:    machineconfiguration.openshift.io/mco-built-in=
Annotations: <none>
API Version: machineconfiguration.openshift.io/v1
Kind:      MachineConfigPool
Metadata:
```

Creation Timestamp: 2019-12-19T02:02:12Z  
Generation: 3  
Resource Version: 16229  
Self Link: /apis/machineconfiguration.openshift.io/v1/machineconfigpools/worker  
UID: 92697796-2203-11ea-b48c-fa163e3940e5  
Spec:  
Configuration:  
Name: rendered-worker-f6819366eb455a401c42f8d96ab25c02  
Source:  
API Version: machineconfiguration.openshift.io/v1  
Kind: MachineConfig  
Name: 00-worker  
API Version: machineconfiguration.openshift.io/v1  
Kind: MachineConfig  
Name: 01-worker-container-runtime  
API Version: machineconfiguration.openshift.io/v1  
Kind: MachineConfig  
Name: 01-worker-kubelet  
API Version: machineconfiguration.openshift.io/v1  
Kind: MachineConfig  
Name: 51-worker-rh-registry-trust  
API Version: machineconfiguration.openshift.io/v1  
Kind: MachineConfig  
Name: 99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries  
API Version: machineconfiguration.openshift.io/v1  
Kind: MachineConfig  
Name: 99-worker-ssh  
Machine Config Selector:  
Match Labels:  
machineconfiguration.openshift.io/role: worker  
Node Selector:  
Match Labels:  
node-role.kubernetes.io/worker:  
Paused: false  
Status:  
Conditions:  
Last Transition Time: 2019-12-19T02:03:27Z  
Message:  
Reason:  
Status: False  
Type: RenderDegraded  
Last Transition Time: 2019-12-19T02:03:43Z  
Message:  
Reason:  
Status: False  
Type: NodeDegraded  
Last Transition Time: 2019-12-19T02:03:43Z  
Message:  
Reason:  
Status: False  
Type: Degraded  
Last Transition Time: 2019-12-19T02:28:23Z  
Message:  
Reason:  
Status: False  
Type: Updated

```

Last Transition Time: 2019-12-19T02:28:23Z
Message:           All nodes are updating to rendered-worker-
f6819366eb455a401c42f8d96ab25c02
Reason:
Status:           True
Type:             Updating
Configuration:
Name: rendered-worker-d9b3f4ffcf5c30dcf591a0e8cf9b2e
Source:
  API Version:    machineconfiguration.openshift.io/v1
  Kind:           MachineConfig
  Name:           00-worker
  API Version:    machineconfiguration.openshift.io/v1
  Kind:           MachineConfig
  Name:           01-worker-container-runtime
  API Version:    machineconfiguration.openshift.io/v1
  Kind:           MachineConfig
  Name:           01-worker-kubelet
  API Version:    machineconfiguration.openshift.io/v1
  Kind:           MachineConfig
  Name:           99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries
  API Version:    machineconfiguration.openshift.io/v1
  Kind:           MachineConfig
  Name:           99-worker-ssh
Degraded Machine Count: 0
Machine Count:         1
Observed Generation:  3
Ready Machine Count:  0
Unavailable Machine Count: 1
Updated Machine Count: 0
Events:               <none>

```

2. **oc describe** コマンドを再度実行します。

```
$ oc describe machineconfigpool/worker
```

### ワーカーの更新後の出力例

```

...
Last Transition Time: 2019-12-19T04:53:09Z
Message:           All nodes are updated with rendered-worker-
f6819366eb455a401c42f8d96ab25c02
Reason:
Status:           True
Type:             Updated
Last Transition Time: 2019-12-19T04:53:09Z
Message:
Reason:
Status:           False
Type:             Updating
Configuration:
Name: rendered-worker-f6819366eb455a401c42f8d96ab25c02
Source:
  API Version:    machineconfiguration.openshift.io/v1
  Kind:           MachineConfig

```

```

Name:          00-worker
API Version:   machineconfiguration.openshift.io/v1
Kind:          MachineConfig
Name:          01-worker-container-runtime
API Version:   machineconfiguration.openshift.io/v1
Kind:          MachineConfig
Name:          01-worker-kubelet
API Version:   machineconfiguration.openshift.io/v1
Kind:          MachineConfig
Name:          51-worker-rh-registry-trust
API Version:   machineconfiguration.openshift.io/v1
Kind:          MachineConfig
Name:          99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries
API Version:   machineconfiguration.openshift.io/v1
Kind:          MachineConfig
Name:          99-worker-ssh
Degraded Machine Count:  0
Machine Count:           3
Observed Generation:     4
Ready Machine Count:     3
Unavailable Machine Count: 0
Updated Machine Count:   3
...

```



### 注記

**Observed Generation** パラメーターは、コントローラーで作成される設定の生成に基づいて増加するカウントを表示します。このコントローラーは、仕様の処理とリビジョンの生成に失敗する場合でも、この値を更新します。**Configuration Source** 値は **51-worker-rh-registry-trust** 設定を参照します。

- 以下のコマンドを使用して、**policy.json** ファイルが存在することを確認します。

```
$ oc debug node/<node> -- chroot /host cat /etc/containers/policy.json
```

### 出力例

```

Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
{
  "default": [
    {
      "type": "insecureAcceptAnything"
    }
  ],
  "transports": {
    "docker": {
      "registry.access.redhat.com": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
        }
      ]
    }
  }
}

```



```

    ],
    "registry.redhat.io": [
      {
        "type": "signedBy",
        "keyType": "GPGKeys",
        "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
      }
    ]
  },
  "docker-daemon": {
    "insecureAcceptAnything": [
      {
        "type": "insecureAcceptAnything"
      }
    ]
  }
}
}
}

```

4. 以下のコマンドを使用して、**registry.redhat.io.yaml** ファイルが存在することを確認します。

```

$ oc debug node/<node> -- chroot /host cat
/etc/containers/registries.d/registry.redhat.io.yaml

```

#### 出力例

```

Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
docker:
  registry.redhat.io:
    sigstore: https://registry.redhat.io/containers/sigstore

```

5. 以下のコマンドを使用して、**registry.access.redhat.com.yaml** ファイルが存在することを確認します。

```

$ oc debug node/<node> -- chroot /host cat
/etc/containers/registries.d/registry.access.redhat.com.yaml

```

#### 出力例

```

Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
docker:
  registry.access.redhat.com:
    sigstore: https://access.redhat.com/webassets/docker/content/sigstore

```

### 2.4.3. 関連情報

- [マシン設定の概要](#)

## 2.5. コンプライアンスについて

多くの OpenShift Container Platform のお客様においては、システムが実稼働環境で使用される前に、一定レベルでの規制への対応またはコンプライアンスが必要になります。この規制対応は、国家標準、業界標準または組織の企業ガバナンスフレームワークによって課せられます。

### 2.5.1. コンプライアンスおよびリスク管理について

FIPS コンプライアンスは、安全な環境で必要とされる最も重要なコンポーネントの1つであり、サポートされている暗号化技術のみがノード上で許可されるようにします。



#### 重要

FIPS 検証済み/進行中のモジュール (Modules in Process) 暗号ライブラリーの使用は、**x86\_64** アーキテクチャーの OpenShift Container Platform デプロイメントでのみサポートされています。

OpenShift Container Platform コンプライアンスフレームワークについての Red Hat のアプローチについては、[OpenShift セキュリティーガイド](#) のリスク管理および規制対応の章を参照してください。

#### 関連情報

- [FIPS モードでのクラスターのインストール](#)

## 2.6. コンテナのコンテンツのセキュリティー保護

コンテナ内のコンテンツのセキュリティーを確保するには、まず信頼できるベースイメージ (Red Hat Universal Base Images など) を使用し、信頼できるソフトウェアを追加する必要があります。コンテナイメージのセキュリティーを継続的に確認するには、Red Hat およびサードパーティーのツールの両方を使用してイメージをスキャンできます。

### 2.6.1. コンテナ内のセキュリティー

アプリケーションとインフラストラクチャーは、すぐに利用できるコンポーネントで設定されています。その多くは、Linux オペレーティングシステム、JBoss Web Server、PostgreSQL、および Node.js などのオープンソースパッケージです。

これらのパッケージのコンテナ化されたバージョンも利用できます。ただし、パッケージの出所や、ビルドした人、パッケージの中に悪質なコードが含まれているかどうかを確認する必要があります。

確認するべき点には以下が含まれます。

- コンテナの内容がインフラストラクチャーを危険にさらす可能性はあるか？
- アプリケーション層に既知の脆弱性が存在するか？
- ランタイムおよびオペレーティングシステム層は最新の状態にあるか？

Red Hat [Universal Base Images](#) (UBI) でコンテナをビルドすることにより、コンテナイメージのベースが Red Hat Enterprise Linux に含まれる同じ RPM パッケージのソフトウェアで設定されるものであることを確認できます。UBI イメージの使用または再配布にサブスクリプションは必要ありません。

コンテナ自体のセキュリティーが継続的に保護されるようにするには、RHEL から直接使用されるか、または OpenShift Container Platform に追加されているセキュリティースキャン機能は、使用しているイメージに脆弱性がある場合に警告を出します。OpenSCAP イメージスキャンは RHEL で利用でき、[Red Hat Quay Container Security Operator](#) は、OpenShift Container Platform で使用されるコンテナイメージをチェックするために追加できます。

## 2.6.2. UBI を使用した再配布可能なイメージの作成

コンテナ化されたアプリケーションを作成するには、通常オペレーティングシステムによって提供されるコンポーネントを提供する信頼されたベースイメージの使用から開始します。これらには、ライブラリー、ユーティリティー、およびその他の機能が含まれます。これらは、アプリケーションがオペレーティングシステムのファイルシステムで認識することが予想されます。

Red Hat Universal Base Images (UBI) は、独自のコンテナのビルドを試行される方に、まず Red Hat Enterprise Linux rpm パッケージやその他のコンテンツで作成されたコンテナを使用するよう奨励するために作成されています。このような UBI イメージは、セキュリティーパッチを適用し、独自のソフトウェアを組み込むためにビルドされたコンテナイメージと共に自由に使用し、再配布するために定期的に更新されます。

[Red Hat Ecosystem Catalog](#) を検索して、異なる UBI イメージを見つけ、そのイメージの正常性を確認します。セキュアなコンテナイメージを作成する場合は、以下の 2 つの一般的な UBI イメージのタイプを使用することを検討できるかもしれません。

- **UBI:** RHEL 7 および 8 の標準 UBI イメージ (`ubi7/ubi` および `ubi8/ubi`)、およびそれらのシステムをベースとする最小イメージ (`ubi7/ubi-minimal` および `ubi8/ubi-minimal`) があります。これらのイメージはすべて、標準の `yum` コマンドおよび `dnf` コマンドを使用して、ビルドするコンテナイメージに追加できる RHEL ソフトウェアの空きのリポジトリを参照するように事前に設定されています。Red Hat は、Fedora や Ubuntu などの他のディストリビューションでこのイメージを使用することを推奨しています。
- **Red Hat Software Collections** Red Hat Ecosystem Catalog で `rhsc/` を検索し、特定タイプのアプリケーションのベースイメージとして使用するために作成されたイメージを見つけます。たとえば、Apache httpd (`rhsc/httpd-*`)、Python (`rhsc/python-*`)、Ruby (`rhsc/ruby-*`)、Node.js (`rhsc/nodejs-*`) および Perl (`rhsc/perl-*`) `rhsc` イメージがあります。

UBI イメージは自由に利用でき、再配布可能ですが、このイメージに対する Red Hat のサポートは、Red Hat 製品サブスクリプションでのみ利用できることに注意してください。

標準、最小および init UBI イメージを使用し、これを使用してビルドする方法については、Red Hat Enterprise Linux ドキュメントの [Red Hat Universal Base イメージの使用](#) を参照してください。

## 2.6.3. RHEL におけるセキュリティースキャン

Red Hat Enterprise Linux (RHEL) システムでは、`openscap-utils` パッケージで OpenSCAP スキャンを利用できます。RHEL では、`openscap-podman` コマンドを使用して、イメージで脆弱性の有無をスキャンできます。Red Hat Enterprise Linux ドキュメントの [Scanning containers and container images for vulnerabilities](#) を参照してください。

OpenShift Container Platform では、RHEL スキャナーを CI/CD プロセスで利用することができます。たとえば、ソースコードのセキュリティー上の欠陥をテストする静的コード解析ツールや、既知の脆弱性などのメタデータを提供するために使用するオープンソースライブラリーを特定するソフトウェアコンポジション解析ツールを統合することができます。

### 2.6.3.1. OpenShift イメージのスキャン

OpenShift Container Platform で実行され、Red Hat Quay レジストリーからプルされるコンテナイメージの場合、Operator を使用してそれらのイメージの脆弱性を一覧表示できます。[Red Hat Quay Container Security Operator](#) を OpenShift Container Platform に追加して、選択した namespace に追加されたイメージの脆弱性レポートを提供することができます。

Red Hat Quay のコンテナイメージスキャンは、[Clair セキュリティースキャナー](#) によって実行されます。Red Hat Quay では、Clair は RHEL、CentOS、Oracle、Alpine、Debian、および Ubuntu のオペ

レーティングシステムソフトウェアでビルドされたイメージの脆弱性を検索し、報告することができます。

## 2.6.4. 外部サービスの統合

OpenShift Container Platform は、[オブジェクトのアノテーション \(object annotations\)](#) を利用して機能を拡張します。脆弱性スキャナーなどの外部ツールはイメージオブジェクトにメタデータのアノテーションを付けることで、結果の要約を表示したり、Pod の実行を制御したりできます。本セクションでは、このアノテーションの認識される形式について説明します。この形式を使用することで、アノテーションをコンソールで安全に使用し、ユーザーに役立つデータを表示することができます。

### 2.6.4.1. イメージのメタデータ

イメージの品質データには、パッケージの脆弱性およびオープンソースソフトウェア (OSS) ライセンスのコンプライアンスなどの様々なタイプがあります。さらに、複数のプロバイダーがこのメタデータを提供する場合があります。このため、以下のアノテーションの形式が保持されます。

```
quality.images.openshift.io/<qualityType>.<providerId>: {}
```

表2.1 アノテーションキーの形式

コンポーネント	説明	許可される値
<b>qualityType</b>	メタデータのタイプ	<b>vulnerability</b> <b>license</b> <b>operations</b> <b>policy</b>
<b>providerId</b>	プロバイダー ID の文字列	<b>openscap</b> <b>redhatcatalog</b> <b>redhatinsights</b> <b>blackduck</b> <b>jfrog</b>

#### 2.6.4.1.1. アノテーションキーの例

```
quality.images.openshift.io/vulnerability.blackduck: {}
quality.images.openshift.io/vulnerability.jfrog: {}
quality.images.openshift.io/license.blackduck: {}
quality.images.openshift.io/vulnerability.openscap: {}
```

イメージの品質アノテーションの値は、以下の形式に従った構造化データになります。

表2.2 アノテーション値の形式

フィールド	必須?	説明	タイプ
<b>name</b>	はい	プロバイダーの表示名	文字列
<b>timestamp</b>	はい	スキャンのタイムスタンプ	文字列

フィールド	必須?	説明	タイプ
<b>description</b>	いいえ	簡単な説明	文字列
<b>reference</b>	はい	情報ソースの URL または詳細情報。ユーザーのデータ検証に必要。	文字列
<b>scannerVersion</b>	いいえ	スキャナーバージョン	文字列
<b>compliant</b>	いいえ	コンプライアンスの合否	ブール値
<b>summary</b>	いいえ	検出された問題の要約	一覧 (以下の表を参照)

**summary** フィールドは、以下の形式に従う必要があります。

表2.3 要約フィールド値の形式

フィールド	説明	タイプ
<b>label</b>	コンポーネントの表示ラベル (例: critical、important、moderate、low または health)	文字列
<b>data</b>	このコンポーネントのデータ (例: 検出された脆弱性の数またはスコア)	文字列
<b>severityIndex</b>	順序付けおよびグラフィック表示の割り当てを可能にするコンポーネントのインデックス。値は <b>0..3</b> の範囲内にあり、 <b>0</b> = low になります。	整数
<b>reference</b>	情報ソースの URL または詳細情報。オプション。	文字列

#### 2.6.4.1.2. アノテーション値の例

以下の例は、脆弱性の要約データおよびコンプライアンスのブール値を含むイメージの OpenSCAP アノテーションを示しています。

#### OpenSCAP アノテーション

```
{
  "name": "OpenSCAP",
  "description": "OpenSCAP vulnerability score",
  "timestamp": "2016-09-08T05:04:46Z",
  "reference": "https://www.open-scap.org/930492",
  "compliant": true,
```

```

"scannerVersion": "1.2",
"summary": [
  { "label": "critical", "data": "4", "severityIndex": 3, "reference": null },
  { "label": "important", "data": "12", "severityIndex": 2, "reference": null },
  { "label": "moderate", "data": "8", "severityIndex": 1, "reference": null },
  { "label": "low", "data": "26", "severityIndex": 0, "reference": null }
]
}

```

以下の例は、詳細情報として外部 URL と正常性のインデックスデータを含むイメージの [Red Hat Ecosystem Catalog](#) の [コンテナイメージのセクション](#) のアノテーションを示しています。

### Red Hat Container Catalog アノテーション

```

{
  "name": "Red Hat Ecosystem Catalog",
  "description": "Container health index",
  "timestamp": "2016-09-08T05:04:46Z",
  "reference": "https://access.redhat.com/errata/RHBA-2016:1566",
  "compliant": null,
  "scannerVersion": "1.2",
  "summary": [
    { "label": "Health index", "data": "B", "severityIndex": 1, "reference": null }
  ]
}

```

#### 2.6.4.2. イメージオブジェクトのアノテーション

OpenShift Container Platform のエンドユーザーはイメージストリームオブジェクトに対して操作を行います。セキュリティメタデータでアノテーションが付けられるのはイメージオブジェクトです。イメージオブジェクトはクラスター全体でそのスコープが設定され、多くのイメージストリームおよびタグで参照される可能性のある単一イメージをポイントします。

##### 2.6.4.2.1. アノテーションが使用されている CLI コマンドの例

**<image>** をイメージダイジェストに置き換えます (例: **sha256:401e359e0f45bfdcf004e258b72e253fd07fba8cc5c6f2ed4f4608fb119ecc2**)。

```

$ oc annotate image <image> \
  quality.images.openshift.io/vulnerability.redhatcatalog='{ \
  "name": "Red Hat Ecosystem Catalog", \
  "description": "Container health index", \
  "timestamp": "2020-06-01T05:04:46Z", \
  "compliant": null, \
  "scannerVersion": "1.2", \
  "reference": "https://access.redhat.com/errata/RHBA-2020:2347", \
  "summary": "[ \
  { "label": "Health index", "data": "B", "severityIndex": 1, "reference": null } ]'

```

#### 2.6.4.3. Pod 実行の制御

**images.openshift.io/deny-execution** イメージポリシーを使用して、イメージを実行するかどうかをプログラムで制御します。

### 2.6.4.3.1. アノテーションの例

```
annotations:
  images.openshift.io/deny-execution: true
```

### 2.6.4.4. 統合リファレンス

ほとんどの場合、脆弱性スキャナーなどの外部ツールはイメージの更新を監視し、スキャンを実施し、関連するイメージオブジェクトに結果のアノテーションを付けるスクリプトまたはプラグインを開発します。この自動化では通常、OpenShift Container Platform 4.9 REST API を呼び出してアノテーションを作成します。REST API の一般的な情報については、OpenShift Container Platform REST API を参照してください。

#### 2.6.4.4.1. REST API 呼び出しの例

`curl` を使用する以下の呼び出しの例では、アノテーションの値を上書きします。<token>、<openshift\_server>、<image\_id>、および <image\_annotation> の値を置き換えてください。

#### パッチ API 呼び出し

```
$ curl -X PATCH \
  -H "Authorization: Bearer <token>" \
  -H "Content-Type: application/merge-patch+json" \
  https://<openshift_server>:6443/apis/image.openshift.io/v1/images/<image_id> \
  --data '{ <image_annotation> }'
```

以下は、**PATCH** ペイロードデータの例です。

#### パッチ呼び出しデータ

```
{
  "metadata": {
    "annotations": {
      "quality.images.openshift.io/vulnerability.redhatcatalog":
        "{ 'name': 'Red Hat Ecosystem Catalog', 'description': 'Container health index', 'timestamp': '2020-06-01T05:04:46Z', 'compliant': null, 'reference': 'https://access.redhat.com/errata/RHBA-2020:2347', 'summary': [{'label': 'Health index', 'data': '4', 'severityIndex': 1, 'reference': null}] }"
    }
  }
}
```

#### 関連情報

- [イメージストリームオブジェクト](#)

## 2.7. コンテナレジストリーのセキュアな使用

コンテナレジストリーは、以下を実行するためにコンテナイメージを保存します。

- イメージに他からアクセスできるようにする
- イメージをイメージの複数バージョンを含むことができるリポジトリに整理する



- オプションで、異なる認証方法に基づいてイメージへのアクセスを制限するか、またはイメージを一般に利用できるようにする。

Quay.io や Docker Hub などのパブリックコンテナレジストリーがあり、ここでは多くの人や組織がイメージを共有します。Red Hat レジストリーは、サポート対象の Red Hat およびパートナーのイメージを提供しますが、Red Hat Ecosystem Catalog ではこれらのイメージに関する詳細な説明およびヘルスチェックが提供されます。独自のレジストリーを管理するには、[Red Hat Quay](#) などのコンテナレジストリーを購入することができます。

セキュリティーの観点では、一部のレジストリーは、コンテナの正常性を確認し、強化するために特別な機能を提供します。たとえば、Red Hat Quay は、Clair セキュリティースキャナーを使用したコンテナ脆弱性のスキャン、GitHub およびその他の場所でソースコードが変更された場合にイメージを自動的に再ビルドするためのビルドのトリガー、およびイメージへのアクセスをセキュア化するためのロールベースのアクセス制御 (RBAC) を使用できる機能を提供します。

### 2.7.1. コンテナのソースの確認

ダウンロード済みかつデプロイ済みのコンテナイメージのコンテンツをスキャンし、追跡するには各種のツールを使用できます。しかし、コンテナイメージの公開ソースは数多くあります。公開されているコンテナレジストリーを使用する場合は、信頼されるソースを使用して保護用の層を追加することができます。

### 2.7.2. イミュータブルで認定済みのコンテナ

イミュータブルなコンテナを管理する際に、セキュリティー更新を使用することはとくに重要になります。イミュータブルなコンテナは、実行中には変更されることのないコンテナです。イミュータブルなコンテナをデプロイする場合には、実行中のコンテナにステップインして1つ以上のバイナリを置き換えることはできません。運用上の観点では、更新されたコンテナイメージを再ビルド、再デプロイし、コンテナを変更するのではなく、コンテナの置き換えを行います。

以下は、Red Hat 認定イメージの特徴になります。

- プラットフォームの各種コンポーネントまたは層に既知の脆弱性がない。
- ベアメタルからクラウドまで、RHEL プラットフォーム全体で互換性がある。
- Red Hat によってサポートされる。

既知の脆弱性の一覧は常に更新されるので、デプロイ済みのコンテナイメージのコンテンツのほか、新規にダウンロードしたイメージを継続的に追跡する必要があります。[Red Hat セキュリティーアドバイザリー \(RHSA\)](#) を利用して、Red Hat 認定コンテナイメージで新たに発見される問題についての警告を受け、更新されたイメージを確認することができます。または、Red Hat Ecosystem Catalog にアクセスして、その問題および各 Red Hat イメージの他のセキュリティー関連の問題について検索することもできます。

### 2.7.3. Red Hat レジストリーおよび Ecosystem Catalog からのコンテナの取得

Red Hat では、Red Hat Ecosystem Catalog の [Container Images](#) セクションから、Red Hat 製品およびパートナーオフリングの認定コンテナイメージを一覧表示しています。このカタログから、CVE、ソフトウェアパッケージの一覧、ヘルススコアなどの各イメージの詳細を確認できます。

Red Hat イメージは、パブリックコンテナレジストリー ([registry.access.redhat.com](#)) および認証されたレジストリー ([registry.redhat.io](#)) によって代表される、Red Hat レジストリーというレジストリーに実際に保存されます。どちらにも基本的に Red Hat サブスクリプション認証情報での認証を必要とするいくつかの追加イメージを含む [registry.redhat.io](#) と同様に、同じコンテナイメージのセットが含まれます。



Red Hat ではコンテナのコンテンツの脆弱性を監視し、コンテンツを定期的に更新しています。[glibc](#)、[DROWN](#)、または [Dirty Cow](#) の修正など、Red Hat がセキュリティ更新をリリースする際に、影響を受けるすべてのコンテナイメージも再ビルドされ、Red Hat Registry にプッシュされます。

Red Hat では **health index** を使用して、Red Hat Ecosystem Catalog 経由で提供される各コンテナのセキュリティ上のリスクを考慮します。コンテナは Red Hat およびエラータプロセスで提供されるソフトウェアを使用するため、セキュリティのレベルは、コンテナが古いと低くなり、新規のコンテナの場合はセキュリティのレベルが上がります。

コンテナの年数について、Red Hat Ecosystem Catalog では格付けシステムを使用します。最新度についての評価は、イメージに利用できる最も古く、最も重大度の高いセキュリティエラータに基づいて行われます。格付けは A から F まであり、A が最新となります。この格付けシステムの詳細については、[Container Health Index grades as used inside the Red Hat Ecosystem Catalog](#) を参照してください。

Red Hat ソフトウェアに関連するセキュリティ更新および脆弱性についての詳細は、[Red Hat Product Security Center](#) を参照してください。[Red Hat セキュリティアドバイザリー](#) を参照して、特定のアドバイザリーおよび CVE を検索できます。

#### 2.7.4. OpenShift Container レジストリー

OpenShift Container Platform には、コンテナイメージを管理するために使用できるプラットフォームの統合されたコンポーネントとして実行される、プライベートレジストリーの **OpenShift Container レジストリー** が含まれます。OpenShift Container レジストリーは、ロールベースのアクセス制御を提供します。これにより、どのコンテナイメージを誰がプル/プッシュするのかを管理できるようになります。

また、OpenShift Container Platform は Red Hat Quay などのすでに使用している可能性のある他のプライベートレジストリーとの統合もサポートしています。

#### 関連情報

- [統合 OpenShift Container Platform レジストリー](#)

#### 2.7.5. Red Hat Quay を使用したコンテナの保存

[Red Hat Quay](#) は、Red Hat のエンタープライズレベルの品質の高いコンテナレジストリー製品です。Red Hat Quay の開発は、アップストリームの [Project Quay](#) で行われます。Red Hat Quay は、オンプレミスまたは [Quay.io](#) のホスト型バージョンの Red Hat Quay でデプロイできます。

Red Hat Quay のセキュリティ関連機能には、以下が含まれます。

- **Time Machine** (マシンの時間設定): 設定した期間またはユーザーが選択した有効期限に基づいて、古いタグを持つイメージの有効期限が切れるようにします。
- **Repository mirroring** (リポジトリのミラーリング): セキュリティ上の理由から他のレジストリーをミラーリングします。たとえば、会社のファイアウォールの背後の Red Hat Quay でパブリックリポジトリをホストしたり、パフォーマンス上の理由からレジストリーを使用される場所の近くに配置したりします。
- **Action log storage** (アクションログの保存): Red Hat Quay のロギング出力を [Elasticsearch ストレージ](#) に保存し、後に検索および分析に使用できるようにします。
- **Clair security scanning (Clair セキュリティスキャン)**: 各コンテナイメージの起点に基づいて、さまざまな Linux 脆弱性データベースに対してイメージをスキャンします。各コンテナ

イメージの起点に基づいて、さまざまな Linux 脆弱性データベースに対してイメージをスキャンします。

- **Internal authentication (内部認証)**: Red Hat Quay への RBAC 認証を処理するデフォルトのローカルデータベースを使用するか、LDAP、Keystone (OpenStack)、JWT Custom Authentication、または External Application Token 認証から選択します。
- **External authorization (OAuth) (外部認証 (OAuth))**: GitHub、GitHub Enterprise、または Google 認証からの Red Hat Quay への認証を許可します。
- **Access settings (アクセス設定)**: docker、rkt、匿名アクセス、ユーザー作成のアカウント、暗号化されたクライアントパスワード、または接頭辞、ユーザー名の自動補完での Red Hat Quay へのアクセスを可能にするトークンを生成します。

Red Hat Quay と OpenShift Container Platform の統合が継続されており、とくに関連する OpenShift Container Platform Operator との統合が継続されています。 [Quay Bridge Operator](#) を使用すると、内部 OpenShift Container Platform レジストリーを Red Hat Quay に置き換えることができます。 [Quay Red Hat Quay Container Security Operator](#) を使用すると、Red Hat Quay レジストリーからプルされた OpenShift Container Platform で実行されているイメージの脆弱性を確認できます。

## 2.8. ビルドプロセスのセキュリティ保護

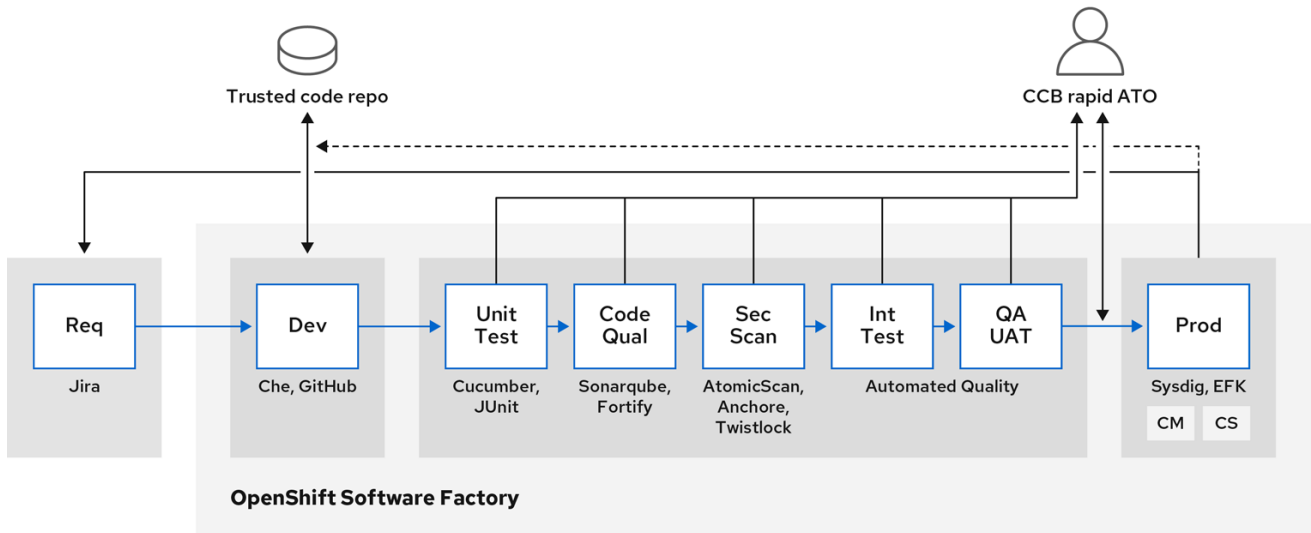
コンテナ環境では、ソフトウェアのビルドプロセスはライフサイクルのステージであり、ここでは、アプリケーションコードが必要なランタイムライブラリーと統合されます。このビルドプロセスの管理は、ソフトウェアのスタックのセキュリティを保護する上で鍵となります。

### 2.8.1.1 回のビルドでどこにでもデプロイが可能

OpenShift Container Platform をコンテナビルドの標準プラットフォームとして使用することで、ビルド環境のセキュリティを確保できます。1回のビルドでどこにでもデプロイが可能という理念を背景に、ビルドプロセスの製品がそのままの状態を実稼働にデプロイされるようにすることができます。

コンテナのイミュータブルな状態を維持することも重要です。実行中のコンテナにパッチを当てることはできません。その代わりに再ビルドおよび再デプロイを実行します。

ソフトウェアがビルド、テスト、および実稼働環境の複数ステージを通過する際に、ソフトウェアのサプライチェーンを設定するツールが信頼できるかどうかは重要です。以下の図は、コンテナ化されたソフトウェアの信頼できるソフトウェアサプライチェーンに組み込むことができるプロセスおよびツールを示しています。



107\_OpenShift\_0720

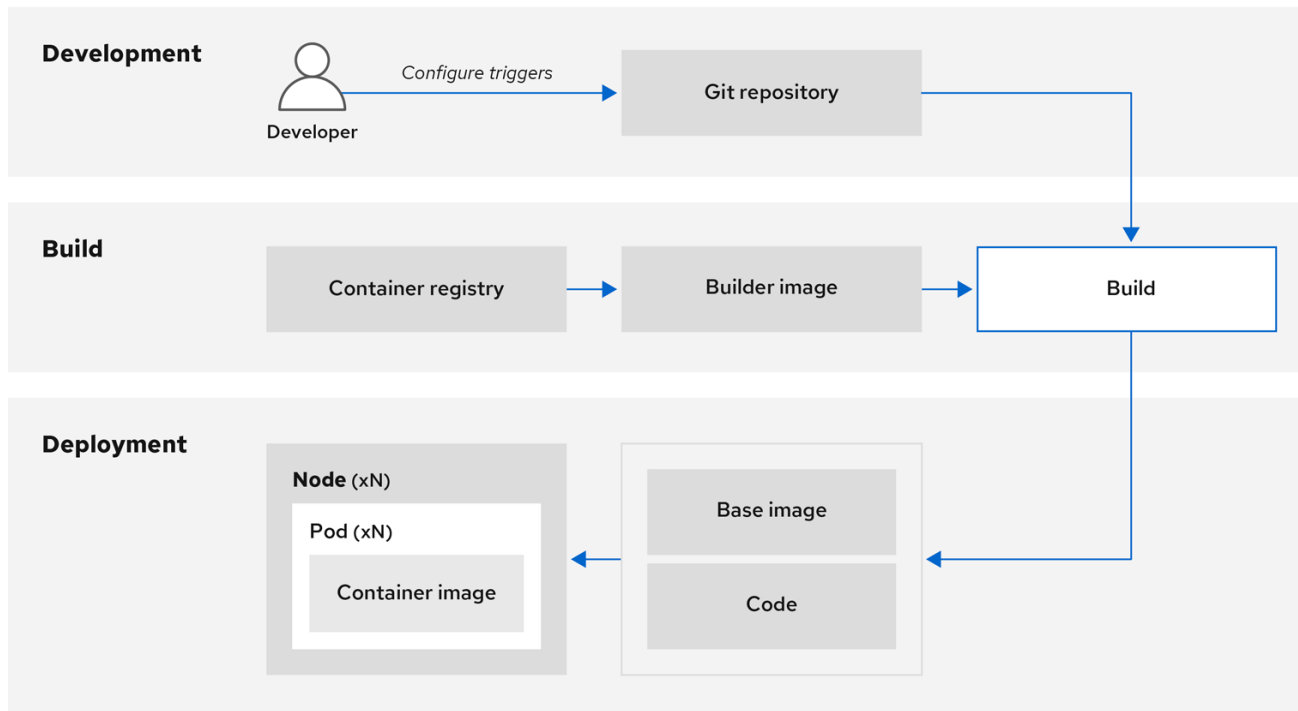
OpenShift Container Platform は、セキュアなコードを作成し、管理できるように、信頼できるコードリポジトリ (GitHub など) および開発プラットフォーム (Che など) と統合できます。単体テストは、[Cucumber](#) および [JUnit](#) に依存する必要がある場合があります。コンテナの脆弱性の有無や、[Anchore](#) または [Twistlock](#) などのコンプライアンス関連の問題の有無を検査し、[AtomicScan](#) または [Clair](#) などのイメージスキャンツールを使用できます。[Sysdig](#) などのツールは、コンテナ化されたアプリケーションの継続的なモニタリングを提供できます。

## 2.8.2. ビルドの管理

Source-to-Image (S2I) を使用して、ソースコードとベースイメージを組み合わせたことができます。**ビルダーイメージ** は S2I を利用し、開発および運用チームの再現可能なビルド環境での協業を可能にします。Red Hat S2I イメージが Universal Base Image (UBI) イメージとして利用可能な場合、実際の RHEL RPM パッケージからビルドされたベースイメージでソフトウェアを自由に再配布できます。Red Hat は、これを可能にするためにサブスクリプションの制限を削除しました。

開発者がビルドイメージを使用して、アプリケーション用に Git でコードをコミットする場合、OpenShift Container Platform は以下の機能を実行できます。

- コードリポジトリの Webhook または他の自動化された継続的インテグレーション (CI) プロセスのいずれかで、利用可能なアーティファクト、S2I ビルダーイメージ、および新たにコミットされたコードを使用して新規イメージの自動アセンブルをトリガーします。
- 新規にビルドしたイメージを自動的にデプロイし、テストします。
- テスト済みのイメージを実稼働にプロモートします。ここでは CI プロセスを使用して自動的にデプロイされます。



107\_OpenShift\_0720

統合された OpenShift Container レジストリーを使用して、最終イメージへのアクセスを管理できます。S2I イメージおよびネイティブビルドイメージの両方は OpenShift Container レジストリーに自動的にプッシュされます。

CI の組み込まれた Jenkins のほかに、独自のビルドおよび CI 環境を RESTful API および API 準拠のイメージレジストリーを使用して OpenShift Container Platform に統合することもできます。

### 2.8.3. ビルド時の入力のセキュリティ保護

シナリオによっては、ビルド操作において、依存するリソースにアクセスするために認証情報が必要になる場合がありますが、この認証情報をビルドで生成される最終的なアプリケーションイメージで利用可能にすることは適切ではありません。このため、入力シークレットを定義することができます。

たとえば、Node.js アプリケーションのビルド時に、Node.js モジュールのプライベートミラーを設定できます。プライベートミラーからモジュールをダウンロードするには、URL、ユーザー名、パスワードを含む、ビルド用のカスタム **.npmrc** ファイルを指定する必要があります。セキュリティ上の理由により、認証情報はアプリケーションイメージで公開しないでください。

この例で示したシナリオを使用して、入力シークレットを新規の **BuildConfig** オブジェクトに追加できます。

1. シークレットがない場合は作成します。

```
$ oc create secret generic secret-npmrc --from-file=.npmrc=~/.npmrc
```

これにより、**secret-npmrc** という名前の新規シークレットが作成されます。これには、**~/.npmrc** ファイルの base64 でエンコードされたコンテンツが含まれます。

2. シークレットを既存の **BuildConfig** オブジェクトの **source** セクションに追加します。

```
source:
  git:
```

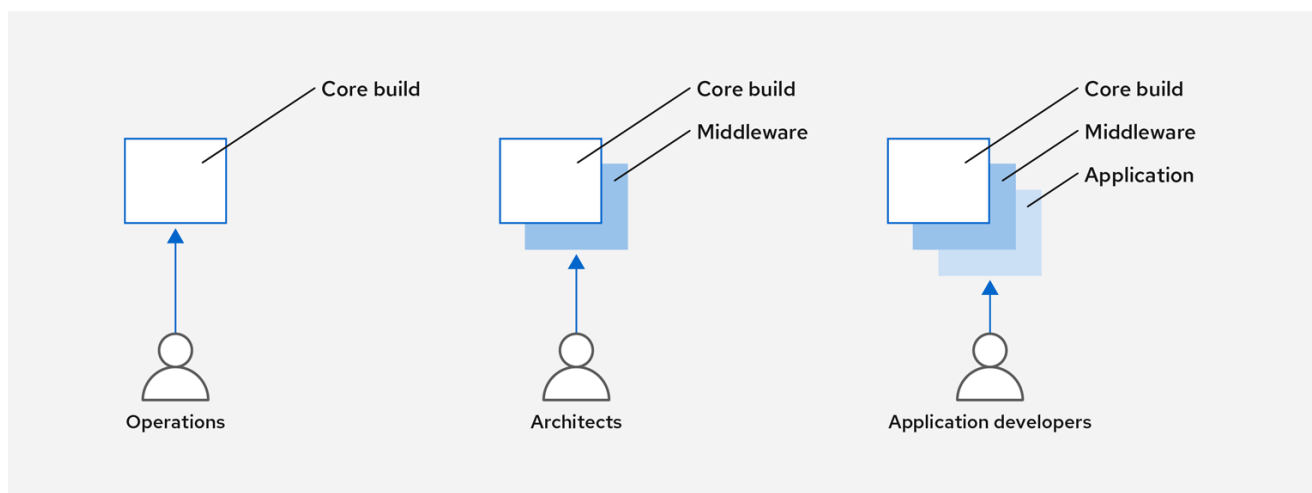
```
uri: https://github.com/sclorg/nodejs-ex.git
secrets:
- destinationDir: .
  secret:
    name: secret-npmrc
```

- シークレットを新規の **BuildConfig** オブジェクトに追加するには、以下のコマンドを実行します。

```
$ oc new-build \
  openshift/nodejs-010-centos7~https://github.com/sclorg/nodejs-ex.git \
  --build-secret secret-npmrc
```

## 2.8.4. ビルドプロセスの設計

コンテナの層を使用できるようにコンテナイメージ管理およびビルドプロセスを設計して、制御を分離可能にすることができます。



107\_OpenShift\_0720

たとえば、運用チームはベースイメージを管理します。一方で、アーキテクトはミドルウェア、ランタイム、データベース、その他のソリューションを管理します。これにより、開発者はアプリケーション層のみを使用し、コードの作成に集中することができます。

新しい脆弱性情報は常に更新されるので、コンテナのコンテンツを継続的かつプロアクティブに確認する必要があります。これを実行するには、自動化されたセキュリティーテストをビルドまたはCIプロセスに統合する必要があります。以下に例を示します。

- SAST / DAST - 静的および動的なセキュリティーテストツール
- 既知の脆弱性をリアルタイムにチェックするためのスキャナー。このようなツールは、コンテナ内のオープンソースパッケージをカタログ化し、既知の脆弱性について通知し、スキャン済みのパッケージに新たな脆弱性が検出されるとその更新情報を送信します。

CIプロセスには、セキュリティースキャンで発見される問題について担当チームが適切に対処できるように、これらの問題のフラグをビルドに付けるポリシーを含める必要があります。カスタマイズしたコンテナに署名することで、ビルドとデプロイメント間に改ざんが発生しないようにします。

GitOpsの方法を使用すると、同じCI/CDメカニズムを使用してアプリケーションの設定だけでなく、OpenShift Container Platform インフラストラクチャーも管理できます。

## 2.8.5. Knative サーバーレスアプリケーションのビルド

Kubernetes と Kourier を利用すると、OpenShift Container Platform で OpenShift Serverless を使用して、サーバーレスアプリケーションを構築、デプロイ、管理できます。

他のビルドと同様に、S2I イメージを使用してコンテナをビルドしてから、Knative サービスを使用してそれらを提供できます。OpenShift Container Platform Web コンソールの **Topology** ビューを使用して Knative アプリケーションのビルドを表示します。

## 2.8.6. 関連情報

- [イメージビルドについて](#)
- [ビルドのトリガーおよび変更](#)
- [ビルド入力の作成](#)
- [入力シークレットおよび設定マップ](#)
- [OpenShift Serverless について](#)
- [Topology ビューを使用したアプリケーション設定の表示](#)

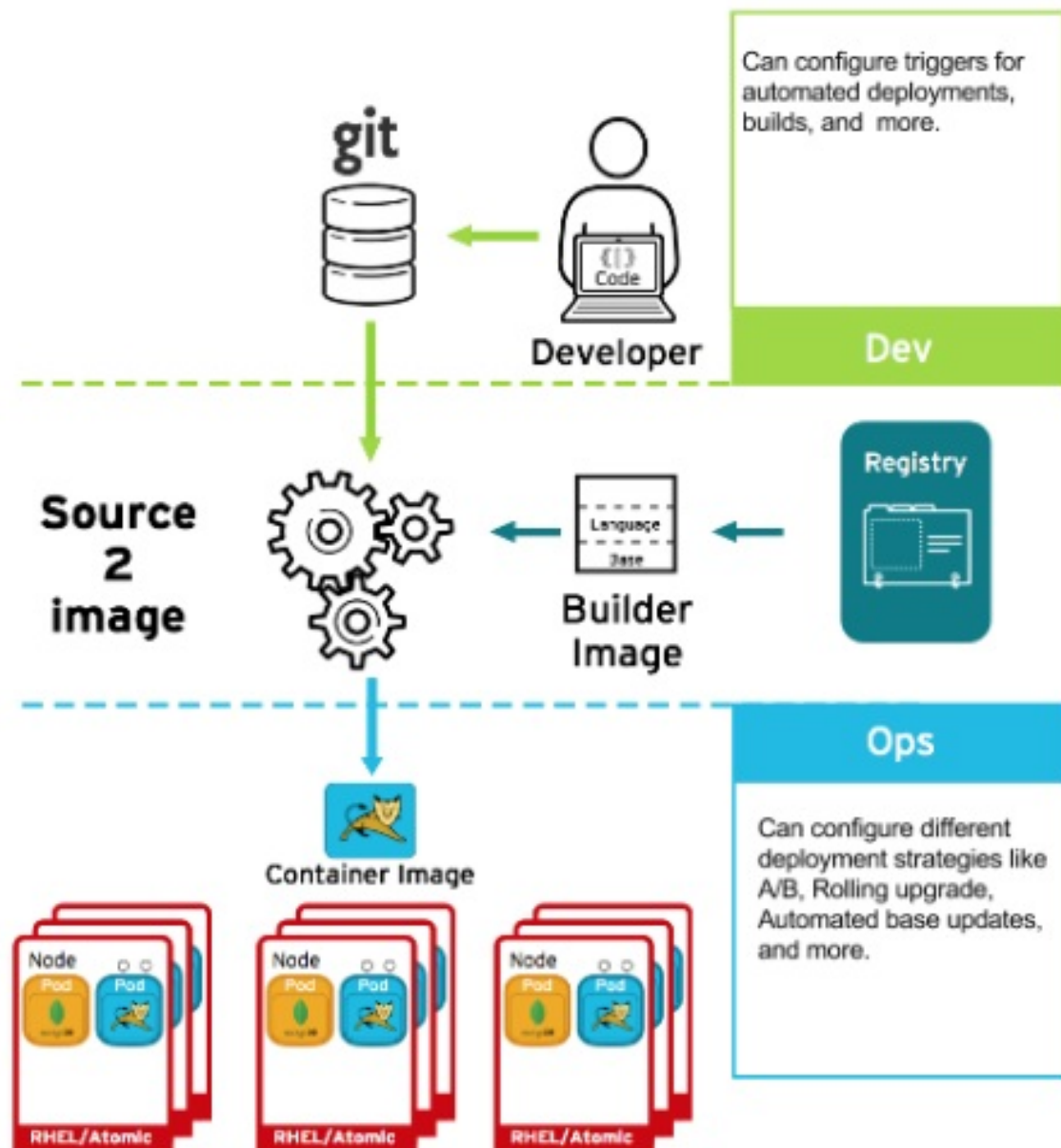
## 2.9. コンテナのデプロイ

各種の手法を使用して、デプロイするコンテナが最新の実稼働に適した品質のコンテンツを保持し、改ざんされていないことを確認することができます。これらの手法には、最新のコードを組み込むためのビルドトリガーのセットアップやコンテナが信頼できるソースから取得され、変更されないようにするための署名の使用が含まれます。

### 2.9.1. トリガーによるコンテナデプロイメントの制御

ビルドプロセスで何らかの問題が生じる場合や、イメージのデプロイ後に脆弱性が発見される場合に、自動化されるポリシーベースのデプロイのためのツールを使用して修復できます。イメージの再ビルドおよび置き換えはトリガーを使用して実行し、イミュータブルなコンテナのプロセスを確認できます。実行中のコンテナにパッチを当てる方法は推奨されていません。





たとえば、3つのコンテナイメージ層(コア、ミドルウェア、アプリケーション)を使用してアプリケーションをビルドするとします。コアイメージに問題が見つかり、そのイメージは再ビルドされました。ビルドが完了すると、イメージは OpenShift Container レジストリーにプッシュされます。OpenShift Container Platform はイメージが変更されたことを検知し、定義されたトリガーに基づいてアプリケーションイメージを自動的に再ビルドし、デプロイします。この変更には修正されたライブラリーが組み込まれ、実稼働コードが最新のイメージと同じ状態になります。

**oc set triggers** コマンドを使用してデプロイメントトリガーを設定できます。たとえば、`deployment-example` という名前のデプロイメントのトリガーを設定するには、以下を実行します。

```
$ oc set triggers deploy/deployment-example \
  --from-image=example:latest \
  --containers=web
```

### 2.9.2. イメージソースのデプロイの制御

重要な点として、対象とするイメージが実際にデプロイされていることや、組み込まれているコンテンツを持つイメージが信頼されるソースからのものであること、またそれらが変更されていないことを確認する必要があります。これは、暗号による署名を使用して実行できます。OpenShift Container

Platform では、クラスター管理者がデプロイメント環境とセキュリティ要件を反映した (広義または狭義のものを含む) セキュリティーポリシーを適用できます。このポリシーは、以下の 2 つのパラメーターで定義されます。

- 1 つ以上のレジストリー (オプションのプロジェクト namespace を使用)
- 信頼タイプ (accept、reject、または require public key(s))

これらのポリシーパラメーターを使用して、レジストリー全体、レジストリーの一部、または個別のイメージに対して信頼関係を許可、拒否、または要求することができます。信頼されたパブリックキーを使用して、ソースが暗号で検証されていることを確認できます。このポリシールールはノードに適用されます。ポリシーは、すべてのノード全体に均一に適用されるか、または異なるノードのワークロード (例: ビルド、ゾーン、または環境) ごとにターゲットが設定される場合があります。

### イメージ署名ポリシーファイルの例

```
{
  "default": [{"type": "reject"}],
  "transports": {
    "docker": {
      "access.redhat.com": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
        }
      ]
    },
    "atomic": {
      "172.30.1.1:5000/openshift": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
        }
      ],
      "172.30.1.1:5000/production": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/example.com/pubkey"
        }
      ],
      "172.30.1.1:5000": [{"type": "reject"}]
    }
  }
}
```

ポリシーは `/etc/containers/policy.json` としてノードに保存できます。このファイルのノードへの保存は、新規の **MachineConfig** オブジェクトを使用して実行するのが最適な方法です。この例では、以下のルールを実施しています。

- Red Hat レジストリー (**registry.access.redhat.com**) からのイメージは Red Hat パブリックキーで署名される必要がある。



- **openshift** namespace 内の OpenShift Container レジストリーからのイメージは Red Hat パブリックキーで署名される必要がある。
- **production** namespace 内の OpenShift Container レジストリーからのイメージは **example.com** のパブリックキーで署名される必要がある。
- グローバルの **default** 定義で指定されていないその他すべてのレジストリーは拒否される。

### 2.9.3. 署名トランスポートの使用

署名トランスポートは、バイナリーの署名 Blob を保存および取得する方法です。署名トランスポートには、2つのタイプがあります。

- **atomic**: OpenShift Container Platform API で管理される。
- **docker**: ローカルファイルとして提供されるか、または Web サーバーによって提供される。

OpenShift Container Platform API は、**atomic** トランスポートタイプを使用する署名を管理します。このタイプの署名を使用するイメージは OpenShift Container レジストリーに保存する必要があります。docker/distribution **extensions** API はイメージ署名のエンドポイントを自動検出するため、追加の設定は不要になります。

**docker** トランスポートタイプを使用する署名は、ローカルファイルまたは Web サーバーによって提供されます。これらの署名には柔軟性があります。任意のコンテナレジストリーからイメージを提供でき、バイナリー署名の送信に個別のサーバーを使用することができます。

ただし、**docker** トランスポートタイプの場合には追加の設定が必要です。任意に名前が付けられた YAML ファイルをホストシステムのディレクトリー (**/etc/containers/registries.d**) にデフォルトとして配置し、ノードを署名サーバーの URI で設定する必要があります。YAML 設定ファイルには、レジストリー URI および署名サーバー URI が含まれます。署名サーバー URI は、**sigstore** と呼ばれます。

#### registries.d ファイルの例

```
docker:
  access.redhat.com:
    sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

この例では、Red Hat レジストリー (**access.redhat.com**) は、**docker** タイプのトランスポートの署名を提供する署名サーバーです。Red Hat レジストリーの URI は、**sigstore** パラメーターで定義されます。このファイルに **/etc/containers/registries.d/redhat.com.yaml** という名前を付け、Machine Config Operator を使用してこのファイルをクラスター内の各ノード上に自動的に配置することができます。ポリシーと **registries.d** ファイルはコンテナのランタイムで動的に読み込まれるため、サービスを再起動する必要はありません。

### 2.9.4. シークレットおよび設定マップの作成

**Secret** オブジェクトタイプはパスワード、OpenShift Container Platform クライアント設定ファイル、**dockercfg** ファイル、プライベートソースリポジトリーの認証情報などの機密情報を保持するメカニズムを提供します。シークレットは機密内容を Pod から切り離します。シークレットはボリュームプラグインを使用してコンテナにマウントすることも、システムが Pod の代わりにシークレットを使用して各種アクションを実行することもできます。

たとえば、プライベートイメージリポジトリーにアクセスできるように、シークレットをデプロイメント設定に追加するには、以下を実行します。

## 手順

1. OpenShift Container Platform Web コンソールにログインします。
2. 新規プロジェクトを作成します。
3. **Resources** → **Secrets** に移動し、新規シークレットを作成します。**Secret Type** を **Image Secret** に、**Authentication Type** を **Image Registry Credentials** に設定し、プライベートイメージリポジトリにアクセスするために必要な認証情報を入力します。
4. デプロイメント設定を作成する場合 (例: **Add to Project** → **Deploy Image** ページに移動する)、**Pull Secret** を新規シークレットに設定します。

設定マップはシークレットに似ていますが、機密情報を含まない文字列の使用をサポートするように設計されています。**ConfigMap** オブジェクトは、Pod で使用したり、コントローラーなどのシステムコンポーネントの設定データを保存するために使用できる設定データのキーと値のペアを保持します。

### 2.9.5. 継続的デプロイメントの自動化

独自の継続的デプロイメント (CD) のツールを OpenShift Container Platform に統合することができます。

CI/CD および OpenShift Container Platform を利用することで、アプリケーションの再ビルドプロセスを自動化し、最新の修正の組み込み、テスト、および環境内の至るところでのデプロイを可能にします。

## 関連情報

- [入力シークレットおよび設定マップ](#)

## 2.10. コンテナプラットフォームのセキュリティー保護

OpenShift Container Platform および Kubernetes API は、スケーリング時にコンテナ管理を自動化する鍵となります。API は以下の目的で使用されます。

- Pod、サービス、およびレプリケーションコントローラーのデータの検証および設定。
- 受信要求におけるプロジェクト検証の実施と、他の主要なシステムコンポーネントでのトリガーの呼び出し。

Kubernetes をベースとする OpenShift Container Platform のセキュリティー関連機能には、以下が含まれます。

- マルチテナンシー: ロールベースのアクセス制御とネットワークポリシーを統合し、複数のレベルでコンテナを分離します。
- API と API の要求側との間の境界を形成する受付プラグイン。

OpenShift Container Platform は Operator を使用して Kubernetes レベルのセキュリティー機能の管理を自動化し、単純化します。

### 2.10.1. マルチテナンシーによるコンテナの分離

マルチテナンシーは、複数のユーザーによって所有され、複数のホストおよび namespace で実行される OpenShift Container Platform クラスターの複数アプリケーションが、相互に分離された状態のままにし、外部の攻撃から隔離された状態にすることができます。ロールベースアクセス制御 (RBAC) を

Kubernetes namespace に適用して、マルチテナンシーを取得します。

Kubernetes では、**namespace** はアプリケーションを他のアプリケーションと分離した状態で実行できるエリアです。OpenShift Container Platform は、SELinux の MCS ラベルを含む追加のアノテーションを追加して namespace を使用し、これを拡張し、これらの拡張された namespace を **プロジェクト** として特定します。プロジェクトの範囲内で、ユーザーは、サービスアカウント、ポリシー、制約、およびその他のオブジェクトなど、独自のクラスターリソースを維持できます。

RBAC オブジェクトはプロジェクトに割り当てられ、選択されたユーザーのそれらのプロジェクトへのアクセスを認可します。この認可には、ルール、ロール、およびバインディングの形式が使用されます。

- ルールは、ユーザーがプロジェクト内で作成またはアクセスできるものを定義します。
- ロールは、選択されたユーザーまたはグループにバインドできるルールのコレクションです。
- バインディングは、ユーザーまたはグループとロール間の関連付けを定義します。

ローカル RBAC ロールおよびバインディングは、ユーザーまたはグループを特定のプロジェクトに割り当てます。クラスター RBAC では、クラスター全体のロールおよびバインディングをクラスターのすべてのプロジェクトに割り当てることができます。**admin**、**basic-user**、**cluster-admin**、および **cluster-status** アクセスを提供するために割り当てることができるデフォルトのクラスターロールがあります。

## 2.10.2. 受付プラグインでのコントロールプレーンの保護

RBAC はユーザーおよびグループと利用可能なプロジェクト間のアクセスルールを制御しますが、**受付プラグイン** は OpenShift Container Platform マスター API へのアクセスを定義します。受付プラグインは、以下で設定されるルールのチェーンを形成します。

- デフォルトの受付プラグイン: これは、OpenShift Container Platform コントロールプレーンのコンポーネントに適用されるポリシーおよびリソース制限のデフォルトセットを実装します。
- ミューティングアドミッションプラグイン: これらのプラグインは、アドミッションチェーンを動的に拡張します。これらは Webhook サーバーに対する呼び出しを実行し、要求の認証および選択されたリソースの変更の両方を実行します。
- 検証用の受付プラグイン: 選択されたリソースの要求を検証し、要求を検証すると共にリソースが再度変更されないようにすることができます。

API 要求はチェーン内の受付プラグインを通過し、途中で失敗した場合には要求が拒否されます。それぞれの受付プラグインは特定のリソースに関連付けられ、それらのリソースの要求にのみ応答します。

### 2.10.2.1. SCC (Security Context Constraints)

**Security Context Constraints (SCC)** を使用して、Pod のシステムでの受け入れを可能にするために Pod の実行時に必要となる一連の条件を定義することができます。

以下は、SCC で管理できる分野の一部です。

- 特権付きコンテナの実行
- コンテナが要求できる機能の追加
- ホストディレクトリーのボリュームとしての使用
- コンテナの SELinux コンテキスト

- コンテナのユーザー ID

必要なパーミッションがある場合は、必要に応じてデフォルトの SCC ポリシーの許容度を上げるように調整することができます。

### 2.10.2.2. ロールのサービスアカウントへの付与

ロールは、ユーザーにロールベースのアクセスを割り当てるのと同じ方法で、サービスアカウントに割り当てることができます。各プロジェクトに3つのデフォルトサービスアカウントが作成されます。サービスアカウント:

- スcopeが特定プロジェクトに制限される
- その名前はそのプロジェクトから派生している。
- OpenShift Container レジストリーにアクセスするために API トークンおよび認証情報が自動的に割り当てられる。

プラットフォームのコンポーネントに関連付けられたサービスアカウントでは、キーが自動的にローテーションされます。

### 2.10.3. 認証および認可

#### 2.10.3.1. OAuth を使用したアクセスの制御

コンテナプラットフォームのセキュリティを保護するために、認証および承認で API アクセス制御を使用することができます。OpenShift Container Platform マスターには、ビルトインの 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 に対して認証することができます。

管理者として、LDAP、GitHub、または Google などの **アイデンティティプロバイダー** を使用して認証できるように OAuth を設定できます。新規の OpenShift Container Platform デプロイメントには、デフォルトでアイデンティティプロバイダーが使用されますが、これを初期インストール時またはインストール後に設定できます。

#### 2.10.3.2. API アクセス制御および管理

アプリケーションには、管理を必要とする各種のエンドポイントを持つ複数の独立した API サービスを設定できます。OpenShift Container Platform には 3scale API ゲートウェイのコンテナ化されたバージョンが含まれており、これにより API を管理し、アクセスを制御することができます。

3scale は、API の認証およびセキュリティについての様々な標準オプションを提供します。これらは、認証情報を発行し、アクセスを制御するために単独で使用することも、他と組み合わせて使用することもできます (例: 標準 API キー、アプリケーション ID とキーペア、OAuth 2.0 など)。

アクセスについては、特定のエンドポイント、メソッド、およびサービスに制限することができ、アクセスポリシーをユーザーグループに適用することができます。アプリケーションの計画に基づいて、API の使用にレート制限を設定したり、開発者グループのトラフィックフローを制御したりすることが可能です。

APIcast v2 (コンテナ化された 3scale API ゲートウェイ) の使用についてのチュートリアルは、3scale ドキュメントの [Running APIcast on Red Hat OpenShift](#) を参照してください。

#### 2.10.3.3. Red Hat Single Sign-On

Red Hat Single Sign-On サーバーを使用すると、SAML 2.0、OpenID Connect、および OAuth 2.0 などの標準に基づく Web サインオン機能を提供し、アプリケーションのセキュリティーを保護することができます。このサーバーは、SAML または OpenID Connect ベースのアイデンティティープロバイダー (IdP) として機能します。つまり、標準ベースのトークンを使用して、アイデンティティー情報およびアプリケーションについてエンタープライズユーザーディレクトリーまたはサードパーティーのアイデンティティープロバイダーとの仲介を行います。Red Hat Single Sign-On を Microsoft Active Directory および Red Hat Enterprise Linux Identity Management を含む LDAP ベースのディレクトリーサービスと統合することが可能です。

#### 2.10.3.4. セルフサービス Web コンソールのセキュリティー保護

OpenShift Container Platform はセルフサービスの Web コンソールを提供して、チームが認証なしに他の環境にアクセスできないようにします。OpenShift Container Platform は以下の条件に基づいてセキュアなマルチテナントマスターを提供します。

- マスターへのアクセスは Transport Layer Security (TLS) を使用する。
- API サーバーへのアクセスは X.509 証明書または OAuth アクセストークンを使用する。
- プロジェクトのクォータは不正トークンによるダメージを制限する。
- etcd サービスはクラスターに直接公開されない。

#### 2.10.4. プラットフォームの証明書の管理

OpenShift Container Platform には、そのフレームワーク内に、TLS 証明書による暗号化を利用した REST ベースの HTTPS 通信を使用する複数のコンポーネントがあります。OpenShift Container Platform のインストーラーは、これらの認証をインストール時に設定します。以下は、このトラフィックを生成するいくつかの主要コンポーネントです。

- マスター (API サーバーとコントローラー)
- etcd
- ノード
- レジストリー
- ルーター

##### 2.10.4.1. カスタム証明書の設定

API サーバーおよび Web コンソールのパブリックホスト名のカスタム提供証明書は、初回のインストール時または証明書の再デプロイ時に設定できます。カスタム CA を使用することも可能です。

#### 関連情報

- [OpenShift Container Platform の紹介](#)
- [RBAC の使用によるパーミッションの定義および適用](#)
- [受付プラグインについて](#)
- [Security Context Constraints の管理](#)
- [SCC リファレンスコマンド](#)

- [ロールをサービスアカウントに付与する例](#)
- [内部 OAuth サーバーの設定](#)
- [アイデンティティプロバイダー設定について](#)
- [証明書の種類および説明](#)
- [プロキシ証明書](#)

## 2.11. ネットワークのセキュリティー保護

ネットワークセキュリティーは、複数のレベルで管理できます。Pod レベルでは、ネットワーク namespace はネットワークアクセスを制限することで、コンテナが他の Pod やホストシステムを認識できないようにすることができます。ネットワークポリシーにより、拒否している接続の許可について制御することができます。コンテナ化されたアプリケーションに対する ingress および egress トラフィックを管理することができます。

### 2.11.1. ネットワーク namespace の使用

OpenShift Container Platform はソフトウェア定義ネットワーク (SDN) を使用して、クラスター全体でのコンテナ間の通信を可能にする統一クラスターネットワークを提供します。

ネットワークポリシーモードは、デフォルトで他の Pod およびネットワークエンドポイントからプロジェクトのすべての Pod にアクセスできるようにします。プロジェクトで1つ以上の Pod を分離するには、そのプロジェクトで **NetworkPolicy** オブジェクトを作成し、許可する着信接続を指定します。マルチテナントモードを使用すると、Pod およびサービスのプロジェクトレベルの分離を実行できます。

### 2.11.2. ネットワークポリシーを使用した Pod の分離

ネットワークポリシーを使用して、同じプロジェクトの Pod を相互に分離することができます。ネットワークポリシーでは、Pod へのネットワークアクセスをすべて拒否し、Ingress コントローラーの接続のみを許可したり、他のプロジェクトの Pod からの接続を拒否したり、ネットワークの動作についての同様のルールを設定したりできます。

#### 関連情報

- [ネットワークポリシーについて](#)

### 2.11.3. 複数の Pod ネットワークの使用

実行中の各コンテナには、デフォルトでネットワークインターフェイスが1つだけあります。Multus CNI プラグインを使用すると、複数の CNI ネットワークを作成し、それらのネットワークのいずれかを Pod に割り当てることができます。このようにして、プライベートデータをより制限されたネットワークに分離し、各ノードに複数のネットワークインターフェイスを持たせることができます。

#### 関連情報

- [複数ネットワークの使用](#)

### 2.11.4. アプリケーションの分離

OpenShift Container Platform では、ユーザー、チーム、アプリケーション、および環境を非グローバルリソースから分離するマルチテナントのクラスターを作成するために、単一のクラスター上でネットワークのトラフィックをセグメント化することができます。

#### 関連情報

- [OpenShiftSDN を使用したネットワーク分離の設定](#)

### 2.11.5. Ingress トラフィックのセキュリティー保護

OpenShift Container Platform クラスター外から Kubernetes サービスへのアクセスを設定する方法に関連し、セキュリティー上の影響についての多数の考慮点があります。Ingress ルーティングでは、HTTP および HTTPS ルートを公開するほか、NodePort または LoadBalancer Ingress タイプを設定できます。NodePort は、それぞれのクラスターワーカーからアプリケーションのサービス API オブジェクトを公開します。LoadBalancer を使用すると、外部ロードバランサーを OpenShift Container Platform クラスターの関連付けられたサービス API オブジェクトに割り当てることができます。

#### 関連情報

- [ingress クラスタートラフィックの設定](#)

### 2.11.6. Egress トラフィックのセキュリティー保護

OpenShift Container Platform は、ルーターまたはファイアウォールのいずれかを使用して Egress トラフィックを制御する機能を提供します。たとえば、IP のホワイトリストを使用して、データベースのアクセスを制御できます。クラスター管理者は、1つ以上の egress IP アドレスを OpenShift Container Platform SDN ネットワークプロバイダーのプロジェクトに割り当てることができます。同様に、クラスター管理者は egress ファイアウォールを使用して、egress トラフィックが OpenShift Container Platform クラスター外に送信されないようにできます。

固定 egress IP アドレスを割り当てることで、すべての送信トラフィックを特定プロジェクトのその IP アドレスに割り当てることができます。egress ファイアウォールを使用すると、Pod が外部ネットワークに接続されないようにしたり、Pod が内部ネットワークに接続されないようにするか、または Pod の特定の内部サブネットへのアクセスを制限したりできます。

#### 関連情報

- [外部 IP アドレスへのアクセスを制御するための egress ファイアウォールの設定](#)
- [プロジェクトの egress IP の設定](#)

## 2.12. 割り当てられたストレージのセキュリティー保護

OpenShift Container Platform は、オンプレミスおよびクラウドプロバイダーの両方で、複数のタイプのストレージをサポートします。とくに、OpenShift Container Platform は Container Storage Interface をサポートするストレージタイプを使用できます。

### 2.12.1. 永続ボリュームプラグイン

コンテナは、ステートレスとステートフルの両方のアプリケーションに役立ちます。割り当て済みのストレージを保護することは、ステートフルサービスのセキュリティーを保護する上で重要な要素になります。Container Storage Interface (CSI) を使用すると、OpenShift Container Platform は CSI インターフェイスをサポートするストレージバックエンドからのストレージを組み込むことができます。



OpenShift Container Platform は、以下を含む複数のタイプのストレージのプラグインを提供します。

- Red Hat OpenShift Container Storage \*
- AWS Elastic Block Stores (EBS) \*
- AWS Elastic File System (EFS) \*
- Azure Disk
- Azure File
- OpenStack Cinder \*
- GCE Persistent Disks \*
- VMware vSphere \*
- ネットワークファイルシステム (NFS)
- FlexVolume
- ファイバーチャネル
- iSCSI

動的プロビジョニングでのこれらのストレージタイプのプラグインには、アスタリスク (\*) が付いています。送信中のデータは、相互に通信している OpenShift Container Platform のすべてのコンポーネントについて HTTPS 経由で暗号化されます。

永続ボリューム (PV) はストレージタイプでサポートされる方法でホスト上にマウントできます。異なるタイプのストレージにはそれぞれ異なる機能があり、各 PV のアクセスモードは、特定のボリュームによって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。各 PV には、**ReadWriteOnce**、**ReadOnlyMany**、および **ReadWriteMany** など、特定の PV 機能を説明したアクセスモードの独自のセットがあります。

### 2.12.2. 共有ストレージ

NFS のような共有ストレージプロバイダーの場合、PV はグループ ID (GID) を PV リソースのアノテーションとして登録します。次に、Pod が PV を要求する際に、アノテーションが付けられた GID が Pod の補助グループに追加され、この Pod に共有ストレージのコンテンツへのアクセスを付与します。

### 2.12.3. ブロックストレージ

AWS Elastic Block Store (EBS)、GCE Persistent Disks、および iSCSI などのブロックストレージプロバイダーの場合、OpenShift Container Platform は SELinux 機能を使用し、権限のない Pod のマウントされたボリュームについて、そのマウントされたボリュームが関連付けられたコンテナにのみ所有され、このコンテナにのみ表示されるようにしてそのルートを保護します。

#### 関連情報

- [永続ストレージについて](#)



- CSI ボリュームの設定
- 動的プロビジョニング
- NFS を使用した永続ストレージ
- AWS Elastic Block Store を使用した永続ストレージ
- GCE Persistent Disk を使用した永続ストレージ

## 2.13. クラスタイベントとログの監視

OpenShift Container Platform クラスタを監視および監査する機能は、不適切な利用に対してクラスタおよびそのユーザーを保護する上で重要な要素となります。

これに関連し、イベントとログという2つの主な情報源をクラスタレベルの情報として使用できます。

### 2.13.1. クラスタイベントの監視

クラスタ管理者は、関連するイベントを判別できるようにイベントのリソースタイプについて理解し、システムイベントの一覧を確認することをお勧めします。イベントは、関連するリソースの namespace または **default** namespace (クラスタイベントの場合) のいずれかの namespace に関連付けられます。デフォルトの namespace は、クラスタを監視または監査するための関連するイベントを保持します。たとえば、これにはノードイベントおよびインフラストラクチャーコンポーネントに関連したリソースイベントが含まれます。

マスター API および **oc** コマンドは、イベントの一覧をノードに関連するものに制限するパラメータを提供しません。これを実行する簡単な方法として **grep** を使用することができます。

```
$ oc get event -n default | grep Node
```

#### 出力例

```
1h      20h      3      origin-node-1.example.local Node      Normal NodeHasDiskPressure ...
```

より柔軟な方法として、他のツールで処理できる形式でイベントを出力することができます。たとえば、以下の例では **NodeHasDiskPressure** イベントのみを展開するために JSON 出力に対して **jq** ツールを使用しています。

```
$ oc get events -n default -o json \
  | jq '.items[] | select(.involvedObject.kind == "Node" and .reason == "NodeHasDiskPressure")'
```

#### 出力例

```
{
  "apiVersion": "v1",
  "count": 3,
  "involvedObject": {
    "kind": "Node",
    "name": "origin-node-1.example.local",
    "uid": "origin-node-1.example.local"
  },
}
```

```
"kind": "Event",
"reason": "NodeHasDiskPressure",
...
}
```

リソースの作成や変更、または削除に関連するイベントも、クラスターの不正な使用を検出するために使用することができます。たとえば、以下のクエリーは、イメージの過剰なプルの有無を確認するために使用できます。

```
$ oc get events --all-namespaces -o json \
| jq '[.items[] | select(.involvedObject.kind == "Pod" and .reason == "Pulling")] | length'
```

## 出力例

```
4
```



### 注記

namespace を削除すると、そのイベントも削除されます。イベントも期限切れになる可能性があり、etcd ストレージが一杯にならないように削除されます。イベントは永続するレコードとして保存されず、一定期間の統計データを取得するためにポーリングを頻繁に実行する必要があります。

## 2.13.2. ロギング

**oc log** コマンドを使用して、コンテナログ、ビルド設定およびデプロイメントをリアルタイムで表示できます。ユーザーによって、ログへの異なるアクセスが必要になる場合があります。

- プロジェクトにアクセスできるユーザーは、デフォルトでそのプロジェクトのログを確認することができます。
- 管理ロールを持つユーザーは、すべてのコンテナログにアクセスできます。

詳細な監査および分析のためにログを保存するには、**cluster-logging** アドオン機能を有効にして、システム、コンテナ、監査ログを収集し、管理し、表示できます。OpenShift Elasticsearch Operator および Red Hat OpenShift Logging Operator を使用して OpenShift Logging をデプロイし、管理し、アップグレードできます。

## 2.13.3. 監査ログ

**監査ログ** を使用すると、ユーザー、管理者、またはその他の OpenShift Container Platform コンポーネントの動作に関連する一連のアクティビティーをフォローできます。API 監査ロギングは各サーバーで行われます。

### 関連情報

- [システムイベントの一覧](#)
- [OpenShift Logging について](#)
- [監査ログの表示](#)

## 第3章 証明書の設定

### 3.1. デフォルトの INGRESS 証明書の置き換え

#### 3.1.1. デフォルトの Ingress 証明書について

デフォルトで、OpenShift Container Platform は Ingress Operator を使用して内部 CA を作成し、**.apps** サブドメインの下にあるアプリケーションに有効なワイルドカード証明書を発行します。Web コンソールと CLI のどちらもこの証明書を使用します。

内部インフラストラクチャー CA 証明書は自己署名型です。一部のセキュリティーまたは PKI チームにとってこのプロセスは適切とみなされない可能性があります。ここで想定されるリスクは最小限度のもので、これらの証明書を暗黙的に信頼するクライアントがクラスター内の他のコンポーネントになります。デフォルトのワイルドカード証明書を、コンテナユーザー空間で提供される CA バンドルにすでに含まれているパブリック CA に置き換えることで、外部クライアントは **.apps** サブドメインで実行されるアプリケーションに安全に接続できます。

#### 3.1.2. デフォルトの Ingress 証明書の置き換え

**.apps** サブドメインにあるすべてのアプリケーションのデフォルトの Ingress 証明書を置き換えることができます。証明書を置き換えた後に、Web コンソールや CLI を含むすべてのアプリケーションには、指定された証明書で提供される暗号化が設定されます。

#### 前提条件

- 完全修飾 **.apps** サブドメインおよびその対応するプライベートキーのワイルドカード証明書が必要です。それぞれが個別の PEM 形式のファイルである必要があります。
- プライベートキーの暗号化は解除されている必要があります。キーが暗号化されている場合は、これを OpenShift Container Platform にインポートする前に復号化します。
- 証明書には、**\*.apps.<clustername>.<domain>** を示す **subjectAltName** 拡張が含まれている必要があります。
- 証明書ファイルでは、チェーンに1つ以上の証明書を含めることができます。ワイルドカード証明書は、ファイルの最初の証明書である必要があります。この後には中間証明書が続き、ファイルの最後はルート CA 証明書にすることができます。
- ルート CA 証明書を追加の PEM 形式のファイルにコピーします。

#### 手順

1. ワイルドカード証明書の署名に使用されるルート CA 証明書のみが含まれる設定マップを作成します。

```
$ oc create configmap custom-ca \  
  --from-file=ca-bundle.crt=</path/to/example-ca.crt> \ 1  
  -n openshift-config
```

- 1** **</path/to/example-ca.crt>** は、ローカルファイルシステム上のルート CA 証明書ファイルへのパスです。

2. 新たに作成された設定マップでクラスター全体のプロキシ設定を更新します。

```
$ oc patch proxy/cluster \
  --type=merge \
  --patch='{"spec":{"trustedCA":{"name":"custom-ca}}}'
```

3. ワイルドカード証明書チェーンおよびキーが含まれるシークレットを作成します。

```
$ oc create secret tls <secret> \ 1
  --cert=</path/to/cert.crt> \ 2
  --key=</path/to/cert.key> \ 3
  -n openshift-ingress
```

- 1 **<secret>** は、証明書チェーンおよびプライベートキーが含まれるシークレットの名前です。
- 2 **</path/to/cert.crt>** は、ローカルファイルシステム上の証明書チェーンへのパスです。
- 3 **</path/to/cert.key>** は、この証明書に関連付けられるプライベートキーへのパスです。

4. Ingress コントローラー設定を、新規に作成されたシークレットで更新します。

```
$ oc patch ingresscontroller.operator default \
  --type=merge -p \
  '{"spec":{"defaultCertificate":{"name":"<secret>}}}' 1
  -n openshift-ingress-operator
```

- 1 **<certificate>** を、直前の手順でシークレットに使用された名前に置き換えます。

## 関連情報

- [CA バンドル証明書の置き換え](#)
- [プロキシ証明書のカスタマイズ](#)

## 3.2. API サーバー証明書の追加

デフォルトの API サーバー証明書は、内部 OpenShift Container Platform クラスター CA によって発行されます。クラスター外のクライアントは、デフォルトで API サーバーの証明書を検証できません。この証明書は、クライアントが信頼する CA によって発行される証明書に置き換えることができます。

### 3.2.1. API サーバーの名前付き証明書の追加

デフォルトの API サーバー証明書は、内部 OpenShift Container Platform クラスター CA によって発行されます。リバースプロキシやロードバランサーが使用される場合など、クライアントが要求する完全修飾ドメイン名 (FQDN) に基づいて、API サーバーが返す代替証明書を1つ以上追加できます。

#### 前提条件

- FQDN とそれに対応するプライベートキーの証明書が必要です。それぞれが個別の PEM 形式のファイルである必要があります。
- プライベートキーの暗号化は解除されている必要があります。キーが暗号化されている場合は、これを OpenShift Container Platform にインポートする前に復号化します。

- 証明書には、FQDN を示す **subjectAltName** 拡張が含まれる必要があります。
- 証明書ファイルでは、チェーンに1つ以上の証明書を含めることができます。API サーバー FQDN の証明書は、ファイルの最初の証明書である必要があります。この後には中間証明書が続き、ファイルの最後はルート CA 証明書にすることができます。



### 警告

内部ロードバランサーに名前付きの証明書を指定しないようにしてください (ホスト名 **api-int.<cluster\_name>.<base\_domain>**)。これを指定すると、クラスターの状態は動作の低下した状態になります。

## 手順

1. **kubeadmin** ユーザーとして新しい API にログインします。

```
$ oc login -u kubeadmin -p <password> https://FQDN:6443
```

2. **kubeconfig** ファイルを取得します。

```
$ oc config view --flatten > kubeconfig-newapi
```

3. **openshift-config** namespace に証明書およびプライベートキーが含まれるシークレットを作成します。

```
$ oc create secret tls <secret> \ ①
  --cert=</path/to/cert.crt> \ ②
  --key=</path/to/cert.key> \ ③
  -n openshift-config
```

① **<secret>** は、証明書チェーンおよびプライベートキーが含まれるシークレットの名前です。

② **</path/to/cert.crt>** は、ローカルファイルシステム上の証明書チェーンへのパスです。

③ **</path/to/cert.key>** は、この証明書に関連付けられるプライベートキーへのパスです。

4. API サーバーを作成されたシークレットを参照するように更新します。

```
$ oc patch apiserver cluster \
  --type=merge -p \
  '{"spec":{"servingCerts": {"namedCertificates":
  [{"names": ["<FQDN>"], ①
  "servingCertificate": {"name": "<secret>"}}]}}' ②
```

① **<FQDN>** を、API サーバーが証明書を提供する FQDN に置き換えます。

② **<certificate>** を、直前の手順でシークレットに使用された名前に置き換えます。

5. **apiserver/cluster** オブジェクトを検査し、シークレットが参照されていることを確認します。

```
$ oc get apiserver cluster -o yaml
```

#### 出力例

```
...
spec:
  servingCerts:
    namedCertificates:
      - names:
        - <FQDN>
      servingCertificate:
        name: <secret>
...

```

6. **kube-apiserver** Operator を確認し、Kubernetes API サーバーの新しいリビジョンがロールアウトされることを確認します。Operator が設定の変更を検出して新しいデプロイメントをトリガーするのに1分かかる場合があります。新しいリビジョンが公開されている間、**PROGRESSING** は **True** を報告します。

```
$ oc get clusteroperators kube-apiserver
```

以下の出力にあるように、**PROGRESSING** が **False** と表示されるまで次の手順に移行しないでください。

#### 出力例

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
kube-apiserver	4.9.0	True	False	False	145m

**PROGRESSING** が **True** と表示されている場合は、数分待機してから再試行します。

## 3.3. サービス提供証明書のシークレットによるサービストラフィックのセキュリティ保護

### 3.3.1. サービス提供証明書について

サービス提供証明書は、暗号化を必要とする複雑なミドルウェアアプリケーションをサポートすることが意図されています。これらの証明書は、TLS Web サーバー証明書として発行されます。

**service-ca** コントローラーは、サービス証明書を生成するために **x509.SHA256WithRSA** 署名アルゴリズムを使用します。

生成される証明書およびキーは PEM 形式のもので、作成されたシークレット内の **tls.crt** および **tls.key** にそれぞれ保存されます。証明書およびキーは、有効期間に近づくと自動的に置き換えられます。

サービス証明書を発行するサービス CA 証明書は 26 ヶ月間有効であり、有効期間が 13 ヶ月未満になると自動的にローテーションされます。ローテーション後も、直前のサービス CA 設定は有効期限が切れるまで信頼されます。これにより、影響を受けるすべてのサービスについて、期限が切れる前にそれらのキーの情報を更新できるように猶予期間が許可されます。この猶予期間中にクラスターをアップグレード (サービスを再起動してそれらのキー情報を更新する) を実行しない場合、直前のサービス CA の期限が切れた後の失敗を防ぐためにサービスを手動で再起動する必要がある場合があります。



## 注記

以下のコマンドを使用して、クラスター内のすべての Pod を手動で再起動できます。このコマンドは、すべての namespace で実行されているすべての Pod を削除するため、このコマンドを実行するとサービスが中断します。これらの Pod は削除後に自動的に再起動します。

```
$ for I in $(oc get ns -o jsonpath='{range .items[*]} {.metadata.name}{"\n"} {end}'); \
do oc delete pods --all -n $I; \
sleep 1; \
done
```

### 3.3.2. サービス証明書の追加

サービスとの通信のセキュリティを保護するには、サービスと同じ namespace のシークレットに署名済みの提供証明書とキーのペアを生成します。

生成される証明書は、内部サービス DNS 名 `<service.name>.<service.namespace>.svc` にのみ有効であり、内部通信用にのみ有効です。サービスがヘッドレスサービス (`clusterIP` 値が設定されていない) である場合、生成された証明書には `*.<service.name>.<service.namespace>.svc` 形式のワイルドカードのサブジェクトも含まれます。



## 重要

生成された証明書にはヘッドレスサービスのワイルドカードサブジェクトが含まれるため、クライアントが個別の Pod を区別する必要がある場合はサービス CA を使用しないでください。この場合は、以下のようになります。

- 別の CA を使用して個別の TLS 証明書を生成します。
- サービス CA は、個々の Pod に送信される接続についての信頼される CA として許可することはできず、他の Pod がこの権限を借用することはできません。これらの接続は、個別の TLS 証明書の生成に使用されている CA を信頼するように設定される必要があります。

### 前提条件:

- サービスが定義されていること。

### 手順

1. サービスに `service.beta.openshift.io/serving-cert-secret-name` のアノテーションを付けます。

```
$ oc annotate service <service_name> \
service.beta.openshift.io/serving-cert-secret-name=<secret_name>
```

① `<service_name>` を、セキュリティ保護するサービスの名前に置き換えます。

② `<secret_name>` は、証明書とキーのペアを含む生成されたシークレットの名前です。便宜上、これを `<service_name>` と同じにすることが推奨されます。

たとえば、以下のコマンドを使用してサービス `test1` にアノテーションを付けます。

-



```
$ oc annotate service test1 service.beta.openshift.io/serving-cert-secret-name=test1
```

2. アノテーションが存在することを確認するためにサービスを検査します。

```
$ oc describe service <service_name>
```

### 出力例

```
...
Annotations:      service.beta.openshift.io/serving-cert-secret-name: <service_name>
                  service.beta.openshift.io/serving-cert-signed-by: openshift-service-serving-
                  signer@1556850837
...
```

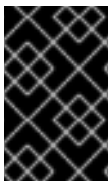
3. クラスターがサービスのシークレットを生成した後に、**Pod** 仕様がこれをマウントでき、Pod はシークレットが利用可能になった後にこれを実行できます。

### 関連情報

- サービス証明書を使用して、reencrypt TLS 終端を使用してセキュアなルートを設定できます。詳細は、[カスタム証明書を使用した re-encrypt ルートの作成](#) を参照してください。

### 3.3.3. サービス CA バンドルの設定マップへの追加

Pod は、**service.beta.openshift.io/inject-cabundle=true** のアノテーションの付いた **ConfigMap** オブジェクトをマウントしてサービス CA 証明書にアクセスできます。アノテーションが付けられると、クラスターはサービス CA 証明書を設定マップの **service-ca.crt** キーに自動的に挿入します。この CA 証明書にアクセスできると、TLS クライアントはサービス提供証明書を使用してサービスへの接続を検証できます。



#### 重要

このアノテーションが設定マップに追加されると、その中に含まれるすべての既存データが削除されます。**service-ca.crt** を組み込む設定マップとしては、Pod の設定の保存先と同じ設定マップではなく、別の設定マップを使用することが推奨されます。

### 手順

1. 設定マップに **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付けます。

```
$ oc annotate configmap <config_map_name> \1
    service.beta.openshift.io/inject-cabundle=true
```

- 1 **<config\_map\_name>** を、アノテーションを付ける設定マップの名前に置き換えます。



#### 注記

ボリュームマウントの **service-ca.crt** キーを明示的に参照することにより、設定マップが CA バンドルと共に挿入されるまで、Pod を起動できなくなります。この動作は、ボリュームの提供証明書の設定について **optional** フィールドを **true** に設定して上書きできます。



たとえば、以下のコマンドを使用して設定マップ **test1** にアノテーションを付けます。

```
$ oc annotate configmap test1 service.beta.openshift.io/inject-cabundle=true
```

- 設定マップを表示して、サービス CA バンドルが挿入されていることを確認します。

```
$ oc get configmap <config_map_name> -o yaml
```

CA バンドルは、YAML 出力の **service-ca.crt** キーの値として表示されます。

```
apiVersion: v1
data:
  service-ca.crt: |
    -----BEGIN CERTIFICATE-----
...

```

### 3.3.4. サービス CA バンドルの API サービスへの追加

**APIService** オブジェクトに **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付け、その **spec.caBundle** フィールドにサービス CA バンドルを設定できます。これにより、Kubernetes API サーバーはターゲットに設定されたエンドポイントのセキュリティーを保護するために使用されるサービス CA 証明書を検証することができます。

#### 手順

- API サービスに **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付けます。

```
$ oc annotate apiservice <api_service_name> \1
  service.beta.openshift.io/inject-cabundle=true
```

- <api\_service\_name>** を、アノテーションを付ける API サービスの名前に置き換えます。

たとえば、以下のコマンドを使用して API サービス **test1** にアノテーションを付けます。

```
$ oc annotate apiservice test1 service.beta.openshift.io/inject-cabundle=true
```

- API サービスを表示し、サービス CA バンドルが挿入されていることを確認します。

```
$ oc get apiservice <api_service_name> -o yaml
```

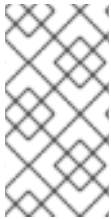
CA バンドルは YAML 出力の **spec.caBundle** フィールドに表示されます。

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
spec:
  caBundle: <CA_BUNDLE>
...

```

### 3.3.5. サービス CA バンドルのカスタムリソース定義への追加

**CustomResourceDefinition** (CRD) オブジェクトに **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付け、その **spec.conversion.webhook.clientConfig.caBundle** フィールドにサービス CA バンドルを設定できます。これにより、Kubernetes API サーバーはターゲットに設定されたエンドポイントのセキュリティを保護するために使用されるサービス CA 証明書を検証することができます。



#### 注記

サービス CA バンドルは、CRD が変換に Webhook を使用するように設定されている場合にのみ CRD にインジェクトされます。CRD の Webhook がサービス CA 証明書でセキュリティ保護されている場合にのみ、サービス CA バンドルを挿入することは役に立ちます。

#### 手順

1. CRD に **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付けます。

```
$ oc annotate crd <crd_name> \1
service.beta.openshift.io/inject-cabundle=true
```

- 1 **<crd\_name>** をアノテーションを付ける CRD の名前に置き換えます。

たとえば、以下のコマンドを使用して CRD **test1** にアノテーションを付けます。

```
$ oc annotate crd test1 service.beta.openshift.io/inject-cabundle=true
```

2. CRD を表示して、サービス CA バンドルが挿入されていることを確認します。

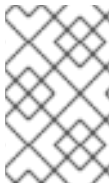
```
$ oc get crd <crd_name> -o yaml
```

CA バンドルは、YAML 出力の **spec.conversion.webhook.clientConfig.caBundle** フィールドに表示されます。

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
  ...
spec:
  conversion:
    strategy: Webhook
    webhook:
      clientConfig:
        caBundle: <CA_BUNDLE>
  ...
```

### 3.3.6. サービス CA バンドルの変更用 Webhook 設定への追加

**MutatingWebhookConfiguration** オブジェクトに **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付け、各 Webhook の **clientConfig.caBundle** フィールドにサービス CA バンドルを設定できます。これにより、Kubernetes API サーバーはターゲットに設定されたエンドポイントのセキュリティを保護するために使用されるサービス CA 証明書を検証することができます。



## 注記

異なる Webhook に異なる CA バンドルを指定する必要がある受付 Webhook 設定にはこのアノテーションを設定しないでください。これを実行する場合、サービス CA バンドルはすべての Webhook について挿入されます。

## 手順

1. 変更用 Webhook 設定に **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付けます。

```
$ oc annotate mutatingwebhookconfigurations <mutating_webhook_name> \1
service.beta.openshift.io/inject-cabundle=true
```

- 1 **<mutatingwebhook-name>** を、アノテーションを付ける変更用 webhook 設定の名前に置き換えます。

たとえば、以下のコマンドを使用して変更用 webhook 設定 **test1** にアノテーションを付けます。

```
$ oc annotate mutatingwebhookconfigurations test1 service.beta.openshift.io/inject-cabundle=true
```

2. 変更用 webhook 設定を表示して、サービス CA バンドルが挿入されていることを確認します。

```
$ oc get mutatingwebhookconfigurations <mutating_webhook_name> -o yaml
```

CA バンドルは、YAML 出力のすべての Webhook の **clientConfig.caBundle** フィールドに表示されます。

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
  ...
webhooks:
- myWebhook:
  - v1beta1
  clientConfig:
    caBundle: <CA_BUNDLE>
  ...
```

### 3.3.7. サービス CA バンドルの変更用 webhook 設定への追加

**ValidatingWebhookConfiguration** オブジェクトに **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付け、各 Webhook の **clientConfig.caBundle** フィールドにサービス CA バンドルを設定できます。これにより、Kubernetes API サーバーはターゲットに設定されたエンドポイントのセ

セキュリティーを保護するために使用されるサービス CA 証明書を検証することができます。



## 注記

異なる Webhook に異なる CA バンドルを指定する必要がある受付 Webhook 設定にはこのアノテーションを設定しないでください。これを実行する場合、サービス CA バンドルはすべての Webhook について挿入されます。

## 手順

1. 検証用 Webhook 設定に **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付けます。

```
$ oc annotate validatingwebhookconfigurations <validating_webhook_name> \1
    service.beta.openshift.io/inject-cabundle=true
```

- 1 **<validating\_webhook\_name>** をアノテーションを付ける検証用 webhook 設定の名前に置き換えます。

たとえば、以下のコマンドを使用して検証用 webhook 設定 **test1** にアノテーションを付けます。

```
$ oc annotate validatingwebhookconfigurations test1 service.beta.openshift.io/inject-
cabundle=true
```

2. 検証用 webhook 設定を表示して、サービス CA バンドルが挿入されていることを確認します。

```
$ oc get validatingwebhookconfigurations <validating_webhook_name> -o yaml
```

CA バンドルは、YAML 出力のすべての Webhook の **clientConfig.caBundle** フィールドに表示されます。

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
  ...
webhooks:
- myWebhook:
  - v1beta1
  clientConfig:
    caBundle: <CA_BUNDLE>
  ...
```

### 3.3.8. 生成されたサービス証明書の手動によるローテーション

関連付けられたシークレットを削除することにより、サービス証明書をローテーションできます。シークレットを削除すると、新規のシークレットが自動的に作成され、新規証明書が作成されます。

## 前提条件

- 証明書とキーのペアを含むシークレットがサービス用に生成されていること。

## 手順

1. 証明書を含むシークレットを確認するためにサービスを検査します。これは、以下に示すように **servicing-cert-secret-name** アノテーションにあります。

```
$ oc describe service <service_name>
```

## 出力例

```
...
service.beta.openshift.io/serving-cert-secret-name: <secret>
...
```

2. サービスの生成されたシークレットを削除します。このプロセスで、シークレットが自動的に再作成されます。

```
$ oc delete secret <secret> ❶
```

- ❶ **<secret>** を、直前の手順のシークレットの名前に置き換えます。

3. 新規シークレットを取得し、**AGE** を調べて、証明書が再作成されていることを確認します。

```
$ oc get secret <service_name>
```

## 出力例

```
NAME          TYPE          DATA AGE
<service.name>  kubernetes.io/tls  2    1s
```

### 3.3.9. サービス CA 証明書の手動によるローテーション

サービス CA は 26 ヶ月間有効で、有効期間が 13 ヶ月未満になると自動的に更新されます。

必要に応じて、以下の手順でサービス CA を手動で更新することができます。



#### 警告

手動でローテーションされるサービス CA は、直前のサービス CA で信頼を維持しません。クラスターの Pod が再起動するまでサービスが一時的に中断する可能性があります。これにより、Pod が新規サービス CA で発行されるサービス提供証明書を使用できるようになります。

## 前提条件

- クラスター管理者としてログインしている必要があります。

## 手順

1. 以下のコマンドを使用して、現在のサービス CA 証明書の有効期限を表示します。

```
$ oc get secrets/signing-key -n openshift-service-ca \
  -o template={{index .data "tls.crt"}} \
  | base64 --decode \
  | openssl x509 -noout -enddate
```

2. サービス CA を手動でローテーションします。このプロセスは、新規サービス証明書に署名するために使用される新規サービス CA を生成します。

```
$ oc delete secret/signing-key -n openshift-service-ca
```

3. 新規証明書をすべてのサービスに適用するには、クラスター内のすべての Pod を再起動します。このコマンドにより、すべてのサービスが更新された証明書を使用ようになります。

```
$ for I in $(oc get ns -o jsonpath='{range .items[*]} {.metadata.name}{"\n"} {end}'); \
  do oc delete pods --all -n $I; \
  sleep 1; \
done
```



### 警告

このコマンドは、すべての namespace で実行されているすべての Pod を調べ、これらを削除するため、サービスを中断させます。これらの Pod は削除後に自動的に再起動します。

## 3.4. CA バンドルの更新

### 3.4.1. CA バンドル証明書について

プロキシ証明書により、ユーザーは egress 接続の実行時にプラットフォームコンポーネントによって使用される1つ以上のカスタム認証局 (CA) を指定できます。

プロキシオブジェクトの **trustedCA** フィールドは、ユーザーによって提供される信頼される認証局 (CA) バンドルを含む設定マップの参照です。このバンドルは Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルにマージされ、egress HTTPS 呼び出しを行うプラットフォームコンポーネントの信頼ストアに挿入されます。たとえば、**image-registry-operator** は外部イメージレジストリーを呼び出してイメージをダウンロードします。**trustedCA** が指定されていない場合、RHCOS 信頼バンドルのみがプロキシされる HTTPS 接続に使用されます。独自の証明書インフラストラクチャーを使用する場合は、カスタム CA 証明書を RHCOS 信頼バンドルに指定します。

**trustedCA** フィールドは、プロキシバリデーターによってのみ使用される必要があります。バリデーターは、必要なキー **ca-bundle.crt** から証明書バンドルを読み取り、これを **openshift-config-managed** namespace の **trusted-ca-bundle** という名前の設定マップにコピーします。**trustedCA** によって参照される設定マップの namespace は **openshift-config** です。

apiVersion: v1

```
kind: ConfigMap
metadata:
  name: user-ca-bundle
  namespace: openshift-config
data:
  ca-bundle.crt: |
    -----BEGIN CERTIFICATE-----
    Custom CA certificate bundle.
    -----END CERTIFICATE-----
```

### 3.4.2. CA バンドル証明書の置き換え

#### 手順

1. ワイルドカード証明書の署名に使用されるルート CA 証明書が含まれる設定マップを作成します。

```
$ oc create configmap custom-ca \
  --from-file=ca-bundle.crt=</path/to/example-ca.crt> \ 1
  -n openshift-config
```

- 1** </path/to/example-ca.crt> は、ローカルファイルシステム上の CA 証明書バンドルへのパスです。

2. 新たに作成された設定マップでクラスター全体のプロキシ設定を更新します。

```
$ oc patch proxy/cluster \
  --type=merge \
  --patch='{"spec":{"trustedCA":{"name":"custom-ca}}}'
```

#### 関連情報

- [デフォルトの Ingress 証明書の置き換え](#)
- [クラスター全体のプロキシの有効化](#)
- [プロキシ証明書のカスタマイズ](#)

## 第4章 証明書の種類および説明

### 4.1. API サーバーのユーザーによって提供される証明書

#### 4.1.1. 目的

API サーバーは、**api.<cluster\_name>.<base\_domain>** のクラスター外にあるクライアントからアクセスできます。クライアントに別のホスト名で API サーバーにアクセスさせたり、クラスター管理の認証局 (CA) 証明書をクライアントに配布せずに API サーバーにアクセスさせたりする必要が生じる場合があります。管理者は、コンテンツを提供する際に API サーバーによって使用されるカスタムデフォルト証明書を設定する必要があります。

#### 4.1.2. 場所

ユーザーによって提供される証明書は、**openshift-config** namespace の **kubernetes.io/tls** タイプの **Secret** で指定される必要があります。ユーザーによって提供される証明書を使用できるように、API サーバークラスター設定の **apiserver/cluster** リソースを更新します。

#### 4.1.3. 管理

ユーザーによって提供される証明書はユーザーによって管理されます。

#### 4.1.4. 有効期限

API サーバークライアント証明書の有効期限は 5 分未満です。

ユーザーによって提供される証明書はユーザーによって管理されます。

#### 4.1.5. カスタマイズ

必要に応じて、ユーザーが管理する証明書を含むシークレットを更新します。

### 関連情報

- [API サーバー証明書の追加](#)

### 4.2. プロキシ証明書

#### 4.2.1. 目的

プロキシ証明書により、ユーザーは egress 接続の実行時にプラットフォームコンポーネントによって使用される 1 つ以上のカスタム認証局 (CA) 証明書を指定できます。

プロキシオブジェクトの **trustedCA** フィールドは、ユーザーによって提供される信頼される認証局 (CA) バンドルを含む設定マップの参照です。このバンドルは Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルにマージされ、egress HTTPS 呼び出しを行うプラットフォームコンポーネントの信頼ストアに挿入されます。たとえば、**image-registry-operator** は外部イメージレジストリーを呼び出してイメージをダウンロードします。**trustedCA** が指定されていない場合、RHCOS 信頼バンドルのみがプロキシされる HTTPS 接続に使用されます。独自の証明書インフラストラクチャーを使用する場合は、カスタム CA 証明書を RHCOS 信頼バンドルに指定します。

**trustedCA** フィールドは、プロキシバリデーターによってのみ使用される必要があります。バリデー



ターは、必要なキー **ca-bundle.crt** から証明書バンドルを読み取り、これを **openshift-config-managed** namespace の **trusted-ca-bundle** という名前の設定マップにコピーします。trustedCA によって参照される設定マップの namespace は **openshift-config** です。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-ca-bundle
  namespace: openshift-config
data:
  ca-bundle.crt: |
    -----BEGIN CERTIFICATE-----
    Custom CA certificate bundle.
    -----END CERTIFICATE-----
```

### 関連情報

- [クラスター全体のプロキシの設定](#)

### 4.2.2. インストール時のプロキシ証明書の管理

インストーラー設定の **additionalTrustBundle** 値は、インストール時にプロキシ信頼 CA 証明書を指定するために使用されます。以下に例を示します。

```
$ cat install-config.yaml
```

### 出力例

```
...
proxy:
  httpProxy: http://<https://username:password@proxy.example.com:123/>
  httpsProxy: https://<https://username:password@proxy.example.com:123/>
  noProxy: <123.example.com,10.88.0.0/16>
  additionalTrustBundle: |
    -----BEGIN CERTIFICATE-----
    <MY_HTTPS_PROXY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
...

```

### 4.2.3. 場所

ユーザーによって提供される信頼バンドルは、設定マップとして表現されます。設定マップは、egress HTTPS 呼び出しを行うプラットフォームコンポーネントのファイルシステムにマウントされます。通常、Operator は設定マップを **/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem** にマウントしますが、これはプロキシでは必要ありません。プロキシは HTTPS 接続を変更したり、検査したりできます。いずれの場合も、プロキシは接続用の新規証明書を生成して、これに署名する必要があります。

完全なプロキシサポートとは、指定されたプロキシに接続し、生成した署名を信頼することを指します。そのため、信頼されたルートに接続しているいずれの証明書チェーンも信頼されるように、ユーザーがその信頼されたルートを指定する必要があります。

RHCOS 信頼バンドルを使用している場合、CA 証明書を **/etc/pki/ca-trust/source/anchors** に配置します。

詳細は、Red Hat Enterprise Linux ドキュメントの [共有システム証明書の使用](#) を参照してください。

#### 4.2.4. 有効期限

ユーザーは、ユーザーによって提供される信頼バンドルの有効期限を設定します。

デフォルトの有効期限は CA 証明書自体で定義されます。この設定は、OpenShift Container Platform または RHCOS で使用する前に、CA 管理者が証明書に対して行います。



#### 注記

Red Hat では、CA の有効期限が切れるタイミングを監視しません。ただし、CA の有効期間は長く設定されるため、通常問題は生じません。ただし、信頼バンドルを定期的に更新する必要がある場合があります。

#### 4.2.5. サービス

デフォルトで、egress HTTPS 呼び出しを行うすべてのプラットフォームコンポーネントは RHCOS 信頼バンドルを使用します。**trustedCA** が定義される場合、これも使用されます。

RHCOS ノードで実行されているすべてのサービスは、ノードの信頼バンドルを使用できます。

#### 4.2.6. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

#### 4.2.7. カスタマイズ

ユーザーによって提供される信頼バンドルを更新するには、以下のいずれかを実行します。

- **trustedCA** で参照される設定マップの PEM でエンコードされた証明書の更新
- 新しい信頼バンドルが含まれる namespace **openshift-config** での設定マップの作成、および新規設定マップの名前を参照できるようにするための **trustedCA** の更新

CA 証明書を RHCOS 信頼バンドルに書き込むメカニズムは、マシン設定を使用して行われるその他のファイルの RHCOS への書き込みと全く同じです。Machine Config Operator (MCO) が新規 CA 証明書が含まれる新規マシン設定を適用すると、ノードは再起動されます。次回の起動時に、サービス **coreos-update-ca-trust.service** は RHCOS ノードで実行されます。これにより、新規 CA 証明書で信頼バンドルが自動的に更新されます。以下に例を示します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 50-examplecorp-ca-cert
spec:
  config:
    ignition:
      version: 3.1.0
    storage:
      files:
        - contents:
```

```

source: data:text/plain;charset=utf-
8;base64,LS0tLS1CRUdJTlBDRVJUSUZJQ0FURStLS0tCk1JSUVORENDQXh5Z0F3SUJBZ0IKQU5
1bkkwRDY2MmNuTUeWR0NTcUdTswIzRFFFQkN3VUFNSUdsTVFzd0NRWUQKV1FRR0V3SIZVek
VYTUJVR0ExVUVDQXdPVG05eWRHZ2dRMkZ5YjJ4cGJtRXhFREFPQmdOVkKJBY01CMUpoYkdWcA
pBMmd4RmpBVUJnTlZCQW9NRfZKbFpDQkZlZWFFZSUUVsdVI5NHhFekFSQmdOVkKJBC01DbEpsWk
NCSVIYUWdTVIF4Ckh6QVpCZ05WQkFNTUVsSmxaQ0JJWVhRZ1NWUWdVbTI2ZENCRRFFURWhN
QjhHQ1Nxr1NjYjNEUUVKQVJZU2FXNW0KWGpDQnBURUxNQWtHQTFVRUJoTUNWVnk14RnpBV
kJnTlZCQWdNRG1dmNuUm9JRU5oY205c2FXNWWhNUkF3RGdZRApXUVFIREFkU1IXeGxhV2RvTV
Jzd0ZBZURWUUVFLREExU1pXUWdTR0YwTENCsSmJtTXVNUk13RVFZRFZRUUxEQXBTCkFXUWd
TR0YwSUUVsVU1Sc3dHUUVIEVFRERERCSINaV1FnU0dGMEIFbFVJRkp2YjNRZ1EwRXhJVEFmQmdrc
WhraUcKMhCwQkNRRVdFbWx1Wm05elpXTkFjbVZrYUdGMEExtTnZiVENDQVNjd0RRWUpLb1pJaH
ZjTKFRRUJCUUFEZ2dFUApCRENDQVFvQ2dnRUJBTFF0OU9KUWg2R0M1TFQxZzgwU5oMHU1
MEJRNHNal3laOGFFVHh0KzVsbIBWWDZNSet6CmQvaTdsRHFUZIRjZkxMMm55VUJkMmZRRGsx
QjBmeHJza2hHSUlaM2ImUDFQczRsdFRrdjhoUINvYjNWdE5xU28KSHhrS2Z2RDJQS2pUUHhEUfDz
eXJ1eTlpcKxaaW9NZmZpM2kvZ0N1dDBaV3RBeU8zTVZINXFXRi9lbkt3Z1BFUwpZOXBvK1RkQ3ZS
Qi9SVU9iQmFNNzYxRWNYTFNNMUdxSE51ZVNmcW5obzNBakxRNmRCbIBXbG82MzhabTFWZWJ
LCKNFTHloa0xXTVNGa0t3RG1uZTBqUTAyWTRnMDc1dkNLdkNzQ0F3RUFBUU5qTUdFd0hRWUR
WUjBPQkZRUZIN1IKNXIDK1VlaElJUGV1TDhacXczUHpiZ2NaTUI4R0ExVWRJd1FZTUJhQUZIN1I0
eUMrVWVoSUIQZXVMOFpxdzNQegpjZ2NaTUE4R0ExVWRfD0VCL3dRRk1BTUJBJh3RGdZRFZS
MFBBUUGvQkFRREFnR0dNQTBHQ1Nxr1NjYjNEUUVCCkR3VUFBNEICQVFCRE52RDJWbTlZQT
VBOUFsT0pSOcTljbVYejloWGN4Sk1cGh4Y1pROGpGb0cwNFZzaHZkMGUKTUVuVXJNY2ZGZ0laN
G5qTUtUUUNNNFpGVVBBaWV5THg0ZjUySHVEb3BwM2U1SnlJTWZkX0tGY05JcEt3Q3NhawpwU2
9LdEIVT3NVSk3cUJWWhnjckl5ZVFWMnFjWU9lWmh0UzV3QnFJd09BaEZ3bENFVDdaZTU4UUhtUz
Q4c2xqCjVIVGtSaml2QWxFeHJGektjbGpDNGF4S1Fsbk92VkF6eitHbTMyVTB4UEJGNEJ5ZVBWeEN
KVUh3MVRzeVRtZWwKU3hORXA3eUhwWGN3bitmWG5hK3Q1SlidoMWd4VvP0eTMKLS0tLS1FTkQ
gQ0VSVEIGSUNBVEUtlS0tLQo=
mode: 0644
overwrite: true
path: /etc/pki/ca-trust/source/anchors/examplecorp-ca.crt

```

マシンの信頼ストアは、ノードの信頼ストアの更新もサポートする必要があります。

#### 4.2.8. 更新

RHCOS ノードで証明書を自動更新できる Operator はありません。



#### 注記

Red Hat では、CA の有効期限が切れるタイミングを監視しません。ただし、CA の有効期間は長く設定されるため、通常問題は生じません。ただし、信頼バンドルを定期的に更新する必要がある場合があります。

### 4.3. サービス CA 証明書

#### 4.3.1. 目的

**service-ca** は、OpenShift Container Platform クラスターのデプロイ時に自己署名の CA を作成する Operator です。

#### 4.3.2. 有効期限

カスタムの有効期限はサポートされません。自己署名 CA は、フィールド **tls.crt** (証明書)、**tls.key** (プライベートキー)、および **ca-bundle.crt** (CA バンドル) の修飾名 **service-ca/signing-key** を持つシークレットに保存されます。

他のサービスは、サービスリソースに **service.beta.openshift.io/serving-cert-secret-name: <secret name>** のアノテーションを付けてサービス提供証明書を要求できます。応答として、Operator は、名前付きシークレットに対し、新規証明書を **tls.crt** として、プライベートキーを **tls.key** として生成します。証明書は 2 年間有効です。

他のサービスは、サービス CA から生成される証明書の検証をサポートするために、**service.beta.openshift.io/inject-cabundle: true** のアノテーションを付けてサービス CA の CA バンドルを API サービスまたは設定マップリソースに挿入するように要求します。応答として、Operator はその現在の CA バンドルを API サービスの **CABundle** フィールドに書き込むか、または **service-ca.crt** として設定マップに書き込みます。

OpenShift Container Platform 4.3.5 の時点で、自動ローテーションはサポートされ、一部の 4.2.z および 4.3.z リリースにバックポートされます。自動ローテーションをサポートするすべてのリリースについて、サービス CA は 26 ヶ月間有効であり、有効期間までの残りの期間が 13 ヶ月未満になると自動的に更新されます。必要に応じて、サービス CA を手動で更新することができます。

サービス CA 有効期限の 26 ヶ月は、サポートされる OpenShift Container Platform クラスターの予想されるアップグレード間隔よりも長くなります。そのため、サービス CA 証明書のコントロールプレーン以外のコンシューマーは CA のローテーション後に更新され、またローテーション前の CA の有効期限が切れる前に更新されます。



#### 警告

手動でローテーションされるサービス CA は、直前のサービス CA で信頼を維持しません。クラスターの Pod が再起動するまでサービスが一時的に中断する可能性があります。これにより、Pod が新規サービス CA で発行されるサービス提供証明書を使用できるようになります。

### 4.3.3. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

### 4.3.4. サービス

サービス CA 証明書を使用するサービスには以下が含まれます。

- cluster-autoscaler-operator
- cluster-monitoring-operator
- cluster-authentication-operator
- cluster-image-registry-operator
- cluster-ingress-operator
- cluster-kube-apiserver-operator
- cluster-kube-controller-manager-operator
- cluster-kube-scheduler-operator

- [cluster-networking-operator](#)
- [cluster-openshift-apiserver-operator](#)
- [cluster-openshift-controller-manager-operator](#)
- [cluster-samples-operator](#)
- [machine-config-operator](#)
- [console-operator](#)
- [insights-operator](#)
- [machine-api-operator](#)
- [operator-lifecycle-manager](#)

これはすべてを網羅した一覧ではありません。

### 関連情報

- [サービス提供証明書の手動ローテーション](#)
- [サービス提供証明書のシークレットによるサービストラフィックのセキュリティー保護](#)

## 4.4. ノード証明書

### 4.4.1. 目的

ノード証明書はクラスターによって署名されます。それらは、ブートストラッププロセスで生成される認証局 (CA) からの証明書です。クラスターがインストールされると、ノード証明書は自動的にローテーションされます。

### 4.4.2. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

### 関連情報

- [ノードの使用](#)

## 4.5. ブートストラップ証明書

### 4.5.1. 目的

OpenShift Container Platform 4 以降では、kubelet は **`/etc/kubernetes/kubeconfig`** にあるブートストラップ証明書を使用して初回のブートストラップを実行します。その次に、[ブートストラップの初期化プロセス](#) および [CSR を作成するための kubelet の認証](#) に進みます。

このプロセスでは、kubelet はブートストラップチャンネル上での通信中に CSR を生成します。コントローラーマネージャーは CSR に署名すると、kubelet が管理する証明書が作成されます。

### 4.5.2. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

### 4.5.3. 有効期限

このブートストラップ CA は 10 年間有効です。

kubelet が管理する証明書は 1 年間有効であり、その 1 年の約 80 パーセントマークで自動的にローテーションします。

### 4.5.4. カスタマイズ

ブートストラップ証明書をカスタマイズすることはできません。

## 4.6. ETCD 証明書

### 4.6.1. 目的

etcd 証明書は etcd-signer によって署名されます。それらの証明書はブートストラッププロセスで生成される認証局 (CA) から提供されます。

### 4.6.2. 有効期限

CA 証明書は 10 年間有効です。ピア、クライアント、およびサーバーの証明書は 3 年間有効です。

### 4.6.3. 管理

これらの証明書はシステムによってのみ管理され、自動的にローテーションされます。

### 4.6.4. サービス

etcd 証明書は、etcd メンバーのピア間の暗号化された通信と暗号化されたクライアントトラフィックに使用されます。以下の証明書は etcd および etcd と通信する他のプロセスによって生成され、使用されます。

- **ピア証明書:** etcd メンバー間の通信に使用されます。
- **クライアント証明書:** 暗号化されたサーバーとクライアント間の通信に使用されます。現時点で、クライアント証明書は API サーバーによってのみ使用され、プロキシを除いてその他のサービスは etcd に直接接続されません。クライアントシークレット (**etcd-client**、**etcd-metric-client**、**etcd-metric-signer**、および **etcd-signer**) は **openshift-config**、**openshift-monitoring**、および **openshift-kube-apiserver** namespace に追加されます。
- **サーバー証明書:** クライアント要求を認証するために etcd サーバーによって使用されます。
- **メトリクス証明書:** メトリクスのすべてのコンシューマーは metric-client 証明書を使用してプロキシに接続します。

### 関連情報

- [クラスタの直前の状態への復元](#)

## 4.7. OLM 証明書

### 4.7.1. 管理

OpenShift Lifecycle Manager (OLM) コンポーネント (**olm-operator**、**catalog-operator**、**packageserver**、および **marketplace-operator**) のすべての証明書はシステムによって管理されます。

Webhook または API サービスを含む Operator を **ClusterServiceVersion** (CSV) オブジェクトにインストールする場合、OLM はこれらのリソースの証明書を作成し、ローテーションします。**openshift-operator-lifecycle-manager** namespace のリソースの証明書は OLM によって管理されます。

OLM はプロキシ環境で管理する Operator の証明書を更新しません。これらの証明書は、ユーザーがサブスクリプション設定で管理する必要があります。

## 4.8. 集合 API クライアント証明書

### 4.8.1. 目的

集約 API クライアント証明書は、集約 API サーバーに接続するときに KubeAPIServer を認証するために使用されます。

### 4.8.2. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

### 4.8.3. 有効期限

この CA は 30 日間有効です。

管理クライアント証明書は 30 日間有効です。

CA およびクライアント証明書は、コントローラーを使用して自動的にローテーションされます。

### 4.8.4. カスタマイズ

集約された API サーバー証明書をカスタマイズすることはできません。

## 4.9. MACHINE CONFIG OPERATOR 証明書

### 4.9.1. 目的

Machine Config Operator 証明書は、Red Hat Enterprise Linux CoreOS (RHCOS) ノードと Machine Config Server 間の接続を保護するために使用されます。





## 重要

現在、マシン設定サーバーエンドポイントをブロックまたは制限する方法はサポートされていません。マシン設定サーバーは、既存の設定または状態を持たない新しくプロビジョニングされたマシンが設定を取得できるように、ネットワークに公開する必要があります。このモデルでは、信頼のルートは証明書署名要求 (CSR) エンドポイントであり、kubelet がクラスターに参加するための承認のために証明書署名要求を送信する場所です。このため、シークレットや証明書などの機密情報を配布するためにマシン設定を使用しないでください。

マシン設定サーバーエンドポイント、ポート 22623 および 22624 がベアメタルシナリオで確実に保護されるようにするには、顧客は適切なネットワークポリシーを設定する必要があります。

## 関連情報

- [OpenShift SDN ネットワークプラグインについて](#)

### 4.9.2. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

### 4.9.3. 有効期限

この CA は 10 年間有効です。

発行されたサービング証明書は 10 年間有効です。

### 4.9.4. カスタマイズ

Machine Config Operator 証明書をカスタマイズすることはできません。

## 4.10. デフォルト INGRESS のユーザーによって提供される証明書

### 4.10.1. 目的

アプリケーションは通常 `<route_name>.apps.<cluster_name>.<base_domain>` で公開されます。`<cluster_name>` および `<base_domain>` はインストール設定ファイルから取得されます。`<route_name>` は、(指定されている場合) ルートのホストフィールド、またはルート名です。例: `hello-openshift-default.apps.username.devcluster.openshift.com` はルートの名前前で、ルートは default namespace に置かれます。クラスター管理の CA 証明書をクライアントに分散せずに、クライアントにアプリケーションにアクセスさせる必要がある場合があります。管理者は、アプリケーションコンテンツを提供する際にカスタムのデフォルト証明書を設定する必要があります。





### 警告

Ingress Operator は、カスタムのデフォルト証明書を設定するまで、プレースホルダーとして機能する Ingress コントローラーのデフォルト証明書を生成します。実稼働クラスターで Operator が生成するデフォルト証明書を使用しないでください。

## 4.10.2. 場所

ユーザーによって提供される証明書は、**openshift-ingress** namespace の **tls** タイプの **Secret** で指定される必要があります。ユーザーがユーザーによって提供される証明書を有効にできるようにするために、**openshift-ingress-operator** namespace で **IngressController** CR を更新します。このプロセスの詳細は、[カスタムデフォルト証明書の設定](#) を参照してください。

## 4.10.3. 管理

ユーザーによって提供される証明書はユーザーによって管理されます。

## 4.10.4. 有効期限

ユーザーによって提供される証明書はユーザーによって管理されます。

## 4.10.5. サービス

クラスターにデプロイされるアプリケーションは、デフォルト Ingress にユーザーによって提供される証明書を使用します。

## 4.10.6. カスタマイズ

必要に応じて、ユーザーが管理する証明書を含むシークレットを更新します。

## 関連情報

- [デフォルトの Ingress 証明書の置き換え](#)

## 4.11. INGRESS 証明書

### 4.11.1. 目的

Ingress Operator は以下の目的で証明書を使用します。

- Prometheus のメトリクスへのアクセスのセキュリティーを保護する。
- ルートへのアクセスのセキュリティーを保護する。

### 4.11.2. 場所

Ingress Operator および Ingress コントローラーメトリクスへのアクセスのセキュリティーを保護するために、Ingress Operator はサービス提供証明書を使用します。Operator は独自のメトリクスについて

**service-ca** コントローラーから証明書を要求し、**service-ca** コントローラーは証明書を **openshift-ingress-operator** namespace の **metrics-tls** という名前のシークレットに配置します。さらに、Ingress Operator は各 Ingress コントローラーの証明書を要求し、**service-ca** コントローラーは証明書を **router-metrics-certs-<name>** という名前のシークレットに配置します。ここで、**<name>** は **openshift-ingress** namespace の Ingress コントローラーの名前です。

各 Ingress コントローラーには、独自の証明書を指定しないセキュリティ保護されたルートに使用するデフォルト証明書があります。カスタム証明書を指定しない場合、Operator はデフォルトで自己署名証明書を使用します。Operator は独自の自己署名証明書を使用して、生成するデフォルト証明書に署名します。Operator はこの署名証明書を生成し、これを **openshift-ingress-operator** namespace の **router-ca** という名前のシークレットに配置します。Operator がデフォルトの証明書を生成する際に、デフォルト証明書を **openshift-ingress** namespace の **router-certs-<name>** という名前のシークレットに配置します (ここで、**<name>** は Ingress コントローラーの名前です)。



**警告**

Ingress Operator は、カスタムのデフォルト証明書を設定するまで、プレースホルダーとして機能する Ingress コントローラーのデフォルト証明書を生成します。実際稼働クラスターで Operator が生成するデフォルト証明書は使用しないでください。

4.11.3. ワークフロー

図4.1 カスタム証明書のワークフロー

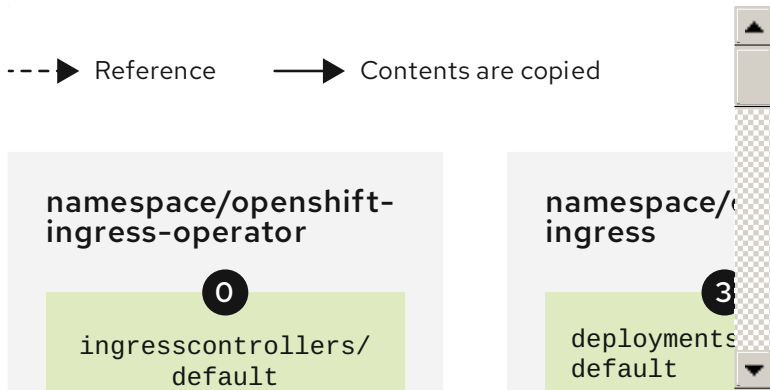
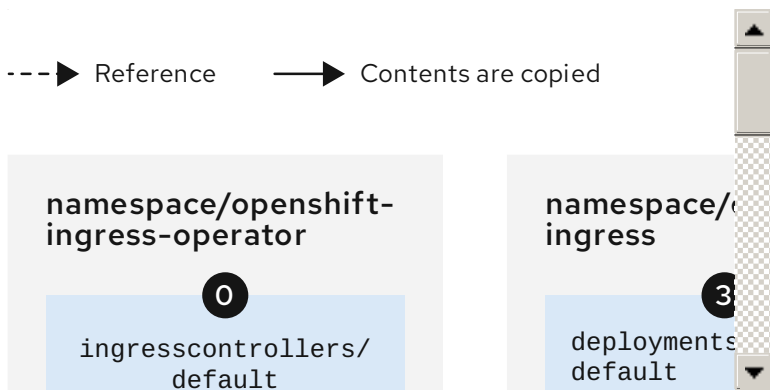


図4.2 デフォルトの証明書ワークフロー



- 0 空の **defaultCertificate** フィールドにより、Ingress Operator はその自己署名 CA を使用して指定されたドメインの提供証明書を生成します。
- 1 Ingress Operator によって生成されるデフォルトの CA 証明書およびキー。Operator が生成するデフォルトの提供証明書に署名するために使用されます。
- 2 デフォルトのワークフローでは、Ingress Operator によって作成され、生成されるデフォルト CA 証明書を使用して署名されるワイルドカードのデフォルト提供証明書です。カスタムワークフローでは、これはユーザーによって提供される証明書です。
- 3 ルーターのデプロイメント。 **secrets/router-certs-default** の証明書を、デフォルトのフロントエンドサーバー証明書として使用します。
- 4 デフォルトのワークフローでは、ワイルドカードのデフォルト提供証明書 (パブリックおよびプライベートの部分) の内容がここにコピーされ、OAuth 統合が有効になります。カスタムワークフローでは、これはユーザーによって提供される証明書です。
- 5 デフォルト提供証明書のパブリック (証明書) の部分です。 **configmaps/router-ca** リソースを置き換えます。
- 6 ユーザーは **ingresscontroller** 提供証明書に署名した CA 証明書でクラスタープロキシ設定を更新します。これにより、 **auth**、 **console** などのコンポーネントや、提供証明書を信頼するために使用するレジストリーが有効になります。
- 7 ユーザーバンドルが指定されていない場合に、組み合わせた Red Hat Enterprise Linux CoreOS (RHCOS) およびユーザーによって提供される CA バンドルまたは RHCOS のみのバンドルを含むクラスター全体の信頼される CA バンドルです。
- 8 他のコンポーネント (**auth** および **console** など) がカスタム証明書で設定された **ingresscontroller** を信頼するよう指示するカスタム CA 証明書バンドルです。
- 9 **trustedCA** フィールドは、ユーザーによって提供される CA バンドルを参照するように使用されます。
- 10 Cluster Network Operator は、信頼される CA バンドルを **proxy-ca** 設定マップに挿入します。
- 11 OpenShift Container Platform 4.9 以降では、 **default-ingress-cert** を使用します。

#### 4.11.4. 有効期限

Ingress Operator の証明書の有効期限は以下の通りです。

- **service-ca** コントローラーが作成するメトリクス証明書の有効期限は、作成日から 2 年間です。
- Operator の署名証明書の有効期限は、作成日から 2 年間です。
- Operator が生成するデフォルト証明書の有効期限は、作成日から 2 年間です。

Ingress Operator または **service-ca** コントローラーが作成する証明書のカスタム有効期限を指定することはできません。

Ingress Operator または **service-ca** コントローラーが作成する証明書について OpenShift Container Platform をインストールする場合に、有効期限を指定することはできません。

#### 4.11.5. サービス

Prometheus はメトリクスのセキュリティを保護する証明書を使用します。

Ingress Operator はその署名証明書を使用して、カスタムのデフォルト証明書を設定しない Ingress コントローラー用に生成するデフォルト証明書に署名します。

セキュリティ保護されたルートを使用するクラスターコンポーネントは、デフォルトの Ingress コントローラーのデフォルト証明書を使用できます。

セキュリティ保護されたルート経由でのクラスターへの Ingress は、ルートが独自の証明書を指定しない限り、ルートがアクセスされる Ingress コントローラーのデフォルト証明書を使用します。

#### 4.11.6. 管理

Ingress 証明書はユーザーによって管理されます。詳細は、[デフォルト ingress 証明書の置き換え](#) を参照してください。

#### 4.11.7. 更新

**service-ca** コントローラーは、これが発行する証明書を自動的にローテーションします。ただし、**oc delete secret <secret>** を使用してサービス提供証明書を手動でローテーションすることができます。

Ingress Operator は、独自の署名証明書または生成するデフォルト証明書をローテーションしません。Operator が生成するデフォルト証明書は、設定するカスタムデフォルト証明書のプレースホルダーとして使用されます。

## 4.12. モニターリングおよび OPENSIFT LOGGING OPERATOR コンポーネント証明書

### 4.12.1. 有効期限

モニターリングコンポーネントは、サービス CA 証明書でトラフィックのセキュリティを保護します。これらの証明書は 2 年間有効であり、13 ヶ月ごとに実行されるサービス CA のローテーションで自動的に置き換えられます。

証明書が **openshift-monitoring** または **openshift-logging** namespace にある場合、これはシステムで管理され、自動的にローテーションされます。

### 4.12.2. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

## 4.13. コントロールプレーンの証明書

### 4.13.1. 場所

コントロールプレーンの証明書はこれらの namespace に含まれます。

- openshift-config-managed
- openshift-kube-apiserver
- openshift-kube-apiserver-operator
- openshift-kube-controller-manager
- openshift-kube-controller-manager-operator
- openshift-kube-scheduler

#### 4.13.2. 管理

コントロールプレーンの証明書はシステムによって管理され、自動的にローテーションされます。

稀なケースとしてコントロールプレーンの証明書の有効期限が切れる場合は、[コントロールプレーン証明書の期限切れの状態からのリカバリー](#)を参照してください。

## 第5章 コンプライアンス OPERATOR

### 5.1. コンプライアンス OPERATOR リリースノート

コンプライアンス Operator を使用すると、OpenShift Container Platform 管理者はクラスターの必要なコンプライアンス状態を記述し、存在するギャップやそれらを修復する方法についての概要を提供します。

これらのリリースノートは、OpenShift Container Platform での コンプライアンス Operator の開発を追跡します。

Compliance Operator の概要については、[Understanding the Compliance Operator](#) を参照してください。

最新リリースにアクセスするには、[コンプライアンス Operator の更新](#) を参照してください。

#### 5.1.1. OpenShift Compliance Operator 1.0.0

OpenShift Compliance Operator 1.0.0 については、以下のアドバイザリーを利用できます。

- [RHBA-2023:1682 - OpenShift Compliance Operator バグ修正更新](#)

##### 5.1.1.1. 新機能および拡張機能

- Compliance Operator は **stable** になり、リリースチャンネルは **stable** にアップグレードされました。将来のリリースは [Semantic Versioning](#) に従います。最新リリースにアクセスするには、[コンプライアンス Operator の更新](#) を参照してください。

##### 5.1.1.2. バグ修正

- この更新前は、`compliance_operator_compliance_scan_error_total` メトリックには、エラーメッセージごとに異なる値を持つ ERROR ラベルがありました。この更新により、`compliance_operator_compliance_scan_error_total` メトリックの値は増加しません。[\(OCPBUGS-1803\)](#)
- この更新前は、`ocp4-api-server-audit-log-maxsize` ルールにより **FAIL** 状態が発生していました。この更新により、エラーメッセージがメトリックから削除され、ベストプラクティスに従ってメトリックのカーディナリティが減少しました。[\(OCPBUGS-7520\)](#)
- この更新前は、`rhcos4-enable-fips-mode` ルールの説明が分かりにくく、インストール後に FIPS を有効化できるという誤解を招くものでした。この更新により、`rhcos4-enable-fips-mode` ルールの説明で、インストール時に FIPS を有効化する必要があることを明確に示されました。[\(OCPBUGS-8358\)](#)

#### 5.1.2. OpenShift Compliance Operator 0.1.61

OpenShift Compliance Operator 0.1.61 については、以下のアドバイザリーを利用できます。

- [RHBA-2023:0557 - OpenShift Compliance Operator bug fix update](#)

##### 5.1.2.1. 新機能および拡張機能

- Compliance Operator は Scanner Pod のタイムアウト設定をサポートするようになりました。タイムアウトは `ScanSetting` オブジェクトで指定されます。スキャンがタイムアウト内に完了

しない場合、スキャンは最大再試行回数に達するまで再試行します。詳細は、[ScanSetting タイムアウトの設定](#) を参照してください。

### 5.1.2.2. バグ修正

- 今回の更新以前は、Compliance Operator は必要な変数を入力として修復していました。変数が設定されていない修復はクラスター全体に適用され、修復が正しく適用された場合でも、ノードがスタックしていました。今回の更新により、Compliance Operator は、修復のために **TailoredProfile** を使用して変数を提供する必要があるかどうかを検証します。(OCPBUGS-3864)
- 今回の更新以前は、**ocp4-kubelet-configure-tls-cipher-suites** の手順が不完全であるため、ユーザーはクエリーを手動で調整する必要がありました。今回の更新により、**ocp4-kubelet-configure-tls-cipher-suites** で提供されるクエリーが実際の結果を返し、監査ステップを実行します。(OCPBUGS-3017)
- この更新の前に、**settingRef** 変数なしで作成された **ScanSettingBinding** オブジェクトは、適切なデフォルト値を使用しませんでした。今回の更新により、**settingRef** 変数のない **ScanSettingBinding** オブジェクトは **default** 値を使用します。(OCPBUGS-3420)
- 今回の更新以前は、システム予約パラメーターが kubelet 設定ファイルで生成されず、Compliance Operator はマシン設定プールの一時停止に失敗していました。今回の更新により、Compliance Operator は、マシン設定プールの評価時にシステム予約パラメーターを省略します。(OCPBUGS-4445)
- 今回の更新以前は、**ComplianceCheckResult** オブジェクトに正しい説明がありませんでした。今回の更新により、Compliance Operator は、ルールの説明から **ComplianceCheckResult** 情報を取得します。(OCPBUGS-4615)
- この更新の前は、Compliance Operator はマシン設定の解析時に空の kubelet 設定ファイルをチェックしませんでした。その結果、Compliance Operator はパニックになり、クラッシュします。今回の更新により、Compliance Operator は kubelet 設定データ構造の改善されたチェックを実装し、完全にレンダリングされた場合にのみ続行します。(OCPBUGS-4621)
- この更新の前は、Compliance Operator は、マシン設定プール名と猶予期間に基づいて kubelet エビクションの修復を生成していたため、単一のエビクションルールの複数の修復が発生していました。今回の更新により、Compliance Operator は単一のルールに対してすべての修復を適用するようになりました。(OCPBUGS-4338)
- 今回の更新以前は、修復コンテンツに変更がなくても、再スキャンの実行後に、以前に **Applied** が **Outdated** とマークされていた可能性のある修復でスキャンを再実行していました。スキャンの比較では、修復メタデータが正しく考慮されませんでした。今回の更新により、修復は以前に生成された **Applied** ステータスを保持します。(OCPBUGS-6710)
- この更新の前に、デフォルト以外の **MachineConfigPool** で **TailoredProfile** を使用する **ScanSettingBinding** を作成しようとする、**ScanSettingBinding** が **Failed** としてマークされるというリグレッションが発生していました。今回の更新で、機能が復元され、**TailoredProfile** を使用してカスタム **ScanSettingBinding** が正しく実行されるようになりました。(OCPBUGS-6827)
- 今回の更新以前は、一部の kubelet 設定パラメーターにデフォルト値がありませんでした。今回の更新により、次のパラメーターにデフォルト値が含まれます (OCPBUGS-6708)。
  - **ocp4-cis-kubelet-enable-streaming-connections**
  - **ocp4-cis-kubelet-eviction-thresholds-set-hard-imagefs-available**



- **ocp4-cis-kubelet-eviction-thresholds-set-hard-imagefs-inodesfree**
- **ocp4-cis-kubelet-eviction-thresholds-set-hard-memory-available**
- **ocp4-cis-kubelet-eviction-thresholds-set-hard-ndefs-available**
- 今回の更新以前は、kubelet の実行に必要なパーミッションが原因で、**selinux\_confinement\_of\_daemons** ルールが kubelet で実行できませんでした。今回の更新で、**selinux\_confinement\_of\_daemons** ルールが無効になります。( [OCPBUGS-6968](#) )

### 5.1.3. OpenShift コンプライアンス Operator 0.1.59

OpenShift コンプライアンス Operator 0.1.59 については、以下のアドバイザリーが利用できます。

- [RHBA-2022:8538 - OpenShift コンプライアンス Operator バグ修正更新](#)

#### 5.1.3.1. 新機能および拡張機能

- Compliance Operator は、**ppc64le** アーキテクチャーで Payment Card Industry Data Security Standard (PCI-DSS) **ocp4-pci-dss** および **ocp4-pci-dss-node** プロファイルをサポートするようになりました。

#### 5.1.3.2. バグ修正

- 以前は、Compliance Operator は **ppc64le** などの異なるアーキテクチャーで Payment Card Industry Data Security Standard (PCI DSS) **ocp4-pci-dss** および **ocp4-pci-dss-node** プロファイルをサポートしていませんでした。Compliance Operator は、**ppc64le** アーキテクチャーで **ocp4-pci-dss** および **ocp4-pci-dss-node** プロファイルをサポートするようになりました。( [OCPBUGS-3252](#) )
- 以前は、バージョン 0.1.57 への最近の更新後、**rerunner** サービスアカウント (SA) はクラスターサービスバージョン (CSV) によって所有されなくなり、Operator のアップグレード中に SA が削除されました。現在、CSV は 0.1.59 で **rerunner** SA を所有しており、以前のバージョンからアップグレードしても SA が欠落することはありません。( [OCPBUGS-3452](#) )
- 0.1.57 では、Operator はポート **8080** でリッスンするコントローラーメトリックエンドポイントを開始しました。これにより、クラスター監視の予期されるポートが **8383** であるため、**TargetDown** アラートが発生しました。0.1.59 では、Operator は期待どおりにポート **8383** でリッスンするエンドポイントを開始します。( [OCPBUGS-3097](#) )

### 5.1.4. OpenShift コンプライアンス Operator 0.1.57

OpenShift コンプライアンス Operator 0.1.57 については、以下のアドバイザリーを利用できます。

- [RHBA-2022:6657 - OpenShift コンプライアンス Operator バグ修正更新](#)

#### 5.1.4.1. 新機能および拡張機能

- **KubeletConfig** チェックが **Node** から **Platform** タイプに変更されました。**KubeletConfig** は、**KubeletConfig** のデフォルト設定を確認します。設定ファイルは、すべてのノードからノードプールごとに1つの場所に集約されます。[デフォルトの設定値をもとにした KubeletConfig ルールの評価](#) を参照してください。



- **ScanSetting** カスタムリソースでは、**scanLimits** 属性を使用して、スキャナー Pod のデフォルトの CPU およびメモリー制限をオーバーライドできるようになりました。詳細は、[コンプライアンス Operator リソース制限の増加](#) を参照してください。
- **PriorityClass** オブジェクトは **ScanSetting** で設定できるようになりました。これにより、コンプライアンス Operator が優先され、クラスターがコンプライアンス違反になる可能性が最小限に抑えられます。詳細は、[ScanSetting スキャンの PriorityClass の設定](#) を参照してください。

#### 5.1.4.2. バグ修正

- 以前のバージョンでは、コンプライアンス Operator は通知をデフォルトの **openshift-compliance** namespace にハードコーディングしていました。Operator がデフォルト以外の namespace にインストールされている場合、通知は予想通りに機能しませんでした。今回のリリースより、通知はデフォルト以外の **openshift-compliance** namespace で機能するようになりました。(BZ#2060726)
- 以前のバージョンでは、コンプライアンス Operator は kubelet オブジェクトによって使用されるデフォルト設定を評価できず、不正確な結果と誤検出が発生していました。[この新機能](#) は kubelet 設定を評価し、正確に報告するようになりました。(BZ#2075041)
- 以前は、コンプライアンス Operator は、**eventRecordQPS** の値がデフォルト値よりも大きい場合、自動修復の適用後に **FAIL** 状態で **ocp4-kubelet-configure-event-creation** ルールを報告していました。**ocp4-kubelet-configure-event-creation** ルール修復はデフォルト値を設定し、ルールが正しく適用されるようになりました。(BZ#2082416)
- **ocp4-configure-network-policies** ルールでは、効果的に実行するために手動の介入が必要です。新しい説明的な指示とルールの更新により、Calico CNI を使用するクラスターに対する **ocp4-configure-network-policies** ルールの適用性が向上します。(BZ#2091794)
- 以前のリリースでは、コンプライアンスオペレーターは、スキャン設定で **debug=true** オプションを使用すると、インフラストラクチャーのスキャンに使用される Pod をクリーンアップしませんでした。これにより、**ScanSettingBinding** を削除した後でも Pod がクラスターに残されました。現在、**ScanSettingBinding** が削除されると、Pod は常に削除されます (BZ#2092913)。
- 以前のバージョンでは、コンプライアンス Operator は古いバージョンの **operator-sdk** コマンドを使用しており、非推奨の機能についてアラートが発生していました。現在、**operator-sdk** コマンドの更新バージョンが含まれており、廃止された機能に関するアラートが発生することはなくなりました。(BZ#2098581)
- 以前のバージョンでは、コンプライアンス Operator は kubelet とマシン設定の関係を判別できない場合に、修復の適用に失敗しました。コンプライアンス Operator はマシン設定の処理を改善し、kubelet 設定がマシン設定のサブセットであるかどうかを判別できるようになりました。(BZ#2102511)
- 以前のリリースでは、**ocp4-cis-node-master-kubelet-enable-cert-rotation** のルールで成功基準が適切に記述されていませんでした。その結果、**RotateKubeletClientCertificate** の要件が不明確でした。今回のリリースでは、**ocp4-cis-node-master-kubelet-enable-cert-rotation** のルールで、kubelet 設定ファイルにある設定に関係なく正確にレポートされるようになりました。(BZ#2105153)
- 以前のリリースでは、アイドルストリーミングタイムアウトをチェックするルールでデフォルト値が考慮されなかったため、ルールレポートが正確ではありませんでした。今回のリリースより、チェックがより厳密に行われるようになり、デフォルトの設定値に基づいて結果の精度が向上しました。(BZ#2105878)

- 以前のバージョンでは、コンプライアンス Operator は Ignition 仕様なしでマシン設定を解析する際に API リソースを取得できず、**api-check-pods** プロセスがクラッシュしていました。コンプライアンス Operator は Ignition 仕様が正しくない場合も Machine Config Pool を処理するようになりました。(BZ#2117268)
- 以前のリリースでは、**modprobe** 設定の値が一致しないことが原因で、修復の適用後に **modprobe** 設定を評価するルールが失敗することがありました。今回のリリースより、チェックおよび修復の **modprobe** 設定に同じ値が使用されるようになり、結果の一貫性が保たれるようになりました。(BZ#2117747)

#### 5.1.4.3. 非推奨

- **Install into all namespaces in the cluster** を指定するか、**WATCH\_NAMESPACES** 環境変数を "" に設定しても、すべての namespace に設定が適用されなくなりました。コンプライアンス Operator のインストール時に指定されていない namespace にインストールされた API リソースは動作しなくなります。API リソースでは、選択した namespace またはデフォルトで **openshift-compliance** namespace を作成する必要がある場合があります。今回の変更により、コンプライアンス Operator のメモリー使用量が改善されます。

### 5.1.5. OpenShift コンプライアンス Operator 0.1.53

OpenShift コンプライアンス Operator 0.1.53 については、以下のアドバイザリーが利用できます。

- [RHBA-2022:5537 - OpenShift コンプライアンス Operator bug fix update](#)

#### 5.1.5.1. バグ修正

- 以前は、**ocp4-kubelet-enable-streaming-connections** ルールに誤った変数比較が含まれていたため、スキャン結果が誤検出されていました。現在、コンプライアンス Operator は、**streamingConnectionIdleTimeout** を設定するときに正確なスキャン結果を提供します。(BZ#2069891)
- 以前は、**/etc/openswitch/conf.db** のグループ所有権が IBM Z アーキテクチャーで正しくなかったため、**ocp4-cis-node-worker-file-groupowner-ovs-conf-db** のチェックが失敗していました。現在、このチェックは IBM Z アーキテクチャーシステムでは **NOT-APPLICABLE** とマークされています。(BZ#2072597)
- 以前は、デプロイメント内のセキュリティーコンテキスト制約 (SCC) ルールに関するデータが不完全なため、**ocp4-cis-scc-limit-container-allowed-capabilities** ルールが **FAIL** 状態で報告されていました。現在は、結果は **MANUAL** ですが、これは、人間の介入を必要とする他のチェックと一致しています。(BZ#2077916)
- 以前は、以下のルールが API サーバーおよび TLS 証明書とキーの追加の設定パスを考慮していなかったため、証明書とキーが適切に設定されていても失敗が報告されていました。
  - **ocp4-cis-api-server-kubelet-client-cert**
  - **ocp4-cis-api-server-kubelet-client-key**
  - **ocp4-cis-kubelet-configure-tls-cert**
  - **ocp4-cis-kubelet-configure-tls-key**

これで、ルールは正確にレポートし、kubelet 設定ファイルで指定されたレガシーファイルパスを監視します。(BZ#2079813)

- 以前は、**content\_rule\_oauth\_or\_oauthclient\_inactivity\_timeout** ルールは、タイムアウトのコンプライアンスを評価するときに、デプロイメントによって設定された設定可能なタイムアウトを考慮していませんでした。その結果、タイムアウトが有効であってもルールが失敗していました。現在、コンプライアンス Operator は、**var\_oauth\_inactivity\_timeout** 変数を使用して、有効なタイムアウトの長さを設定しています。(BZ#2081952)
- 以前は、コンプライアンス Operator は、特権使用に適切にラベル付けされていない namespace に対して管理者権限を使用していたため、Pod のセキュリティーレベル違反に関する警告メッセージが表示されていました。現在、コンプライアンス Operator は、適切な namespace ラベルと権限調整を行い、権限に違反することなく結果にアクセスできるようになっています。(BZ#2088202)
- 以前は、**rhcos4-high-master-sysctl-kernel-yama-pttrace-scope** および **rhcos4-sysctl-kernel-core-pattern** に自動修復を適用すると、それらのルールが修復されても、その後のスキャン結果で失敗することがありました。現在、修復が適用された後でも、ルールは **PASS** を正確に報告します。(BZ#2094382)
- 以前は、コンプライアンス Operator は、メモリー不足の例外が原因で **CrashLoopBackoff** 状態で失敗していました。現在では、コンプライアンス Operator は、メモリー内の大規模なマシン設定データセットを処理し、正しく機能するように改良されました。(BZ#2094854)

#### 5.1.5.2. 既知の問題

- **ScanSettingBinding** オブジェクト内で **debug:true** が設定されている場合に、そのバインディングが削除されても、**ScanSettingBinding** オブジェクトによって生成された Pod は削除されません。回避策として、次のコマンドを実行して残りの Pod を削除します。

```
$ oc delete pods -l compliance.openshift.io/scan-name=ocp4-cis
```

(BZ#2092913)

#### 5.1.6. OpenShift コンプライアンス Operator 0.1.52

OpenShift コンプライアンス Operator 0.1.52 については、以下のアドバイザリーが利用できます。

- [RHBA-2022:4657 - OpenShift コンプライアンス Operator bug fix update](#)

#### 5.1.6.1. 新機能および拡張機能

- FedRAMP の SCAP プロファイル high が、OpenShift Container Platform 環境で使用できるようになりました。詳細は、[サポートされているコンプライアンスプロファイル](#) を参照してください。

#### 5.1.6.2. バグ修正

- 以前は、**DAC\_OVERRIDE** 機能が削除されているセキュリティー環境でのマウントパーミッションの問題が原因で、**OpenScap** コンテナがクラッシュしていました。今回は、実行可能なマウントパーミッションがすべてのユーザーに適用されるようになりました。(BZ#2082151)
- 以前は、コンプライアンスルール **ocp4-configure-network-policies** を **MANUAL** として設定できました。今回は、コンプライアンスルール **ocp4-configure-network-policies** が **AUTOMATIC** に設定されるようになりました。(BZ#2072431)
- 以前は、コンプライアンス Operator のスキャン Pod はスキャン後に削除されないため、Cluster Autoscaler はスケールダウンできませんでした。今回は、デバッグ目的で明示的に保存

されていない限り、Pod はデフォルトで各ノードから削除されるようになりました。

([BZ#2075029](#))

- 以前は、コンプライアンス Operator を **KubeletConfig** に適用すると、マシン設定プールの一時停止が早すぎてノードが **NotReady** 状態になることがありました。今回は、マシン設定プールは停止されず、ノードが正常に機能するようになりました。( [BZ#2071854](#) )
- 以前のバージョンでは、Machine Config Operator は最新のリリースで **url-encoded** コードではなく **base64** を使用していたため、コンプライアンス Operator の修復が失敗していました。コンプライアンス Operator は、**base64** および **url-encoded** マシン設定コードの両方を処理するようにエンコーディングをチェックし、修復が適切に実行されるようになりました。( [BZ#2082431](#) )

### 5.1.6.3. 既知の問題

- **ScanSettingBinding** オブジェクト内で **debug:true** が設定されている場合に、そのバインディングが削除されても、**ScanSettingBinding** オブジェクトによって生成された Pod は削除されません。回避策として、次のコマンドを実行して残りの Pod を削除します。

```
$ oc delete pods -l compliance.openshift.io/scan-name=ocp4-cis
```

([BZ#2092913](#))

### 5.1.7. OpenShift コンプライアンス Operator 0.1.49

OpenShift コンプライアンス Operator 0.1.49 については、以下のアドバイザリーが利用できます。

- [RHBA-2022:1148 - OpenShift コンプライアンス Operator bug fix and enhancement update](#)

#### 5.1.7.1. バグ修正

- 以前は、**openshift-compliance** コンテンツには、ネットワークタイプのプラットフォーム固有のチェックが含まれていませんでした。その結果、OVN および SDN 固有のチェックは、ネットワーク設定に基づいて **not-applicable** ではなく、**failed** したものと表示されます。現在、新しいルールにはネットワークルールのプラットフォームチェックが含まれているため、ネットワーク固有のチェックをより正確に評価できます。( [BZ#1994609](#) )
- 以前は、**ocp4-moderate-routes-protected-by-tls** ルールは TLS 設定を誤ってチェックしていたため、接続がセキュアな SSL/TLS プロトコルであっても、ルールがチェックに失敗していました。現在、このチェックでは、ネットワークガイダンスおよびプロファイルの推奨事項と一致する TLS 設定が適切に評価されます。( [BZ#2002695](#) )
- 以前は、**ocp-cis-configure-network-policies-namespace** は、namespaces を要求するときにページネーションを使用していました。これにより、デプロイメントで 500 を超える namespaces のリストが切り捨てられたため、ルールが失敗しました。今回のバージョンでは、namespace の全リストが必要になり、namespace が 500 以上ある場合でも、設定済みのネットワークポリシーをチェックするルールが機能するようになりました。( [BZ#2038909](#) )
- 以前は、**sshd jinja** マクロを使用した修復は、特定の sshd 設定にハードコーディングされていました。その結果、設定がルールがチェックしているコンテンツと一致せず、チェックが失敗していました。これで、sshd 設定がパラメーター化され、ルールが正常に適用されます。( [BZ#2049141](#) )
- 以前は、**ocp4-cluster-version-operator-verify-integrity** は、常に Cluter バージョンオペレーター (CVO) 履歴の最初のエントリーをチェックしていました。その結果、`{product-name}` の



後続のバージョンが検証される状況では、アップグレードは失敗します。これで、**ocp4-cluster-version-operator-verify-integrity** のコンプライアンスチェック結果は、検証済みバージョンを検出でき、CVO 履歴で正確になります。(BZ#2053602)

- 以前は、**ocp4-api-server-no-adm-ctrl-plugins-disabled** ルールは、空のアドミッションコントローラープラグインのリストをチェックしませんでした。その結果、すべてのアドミッションプラグインが有効になっている場合でも、ルールは常に失敗します。今回のリリースでは、**ocp4-api-server-no-adm-ctrl-plugins-disabled** ルールのチェックが強力になり、すべての受付コントローラープラグインを有効にすると、問題なく成功するようになりました。(BZ#2058631)
- 以前は、スキャンには Linux ワーカーノードに対して実行するためのプラットフォームチェックが含まれていませんでした。その結果、Linux ベースではないワーカーノードに対してスキャンを実行すると、スキャンループが終了することはありませんでした。今回のリリースでは、スキャンは、プラットフォームのタイプとラベルに基づいて適切にスケジュールされ、正確に実行されます。(BZ#2056911)

## 5.1.8. OpenShift コンプライアンス Operator 0.1.48

OpenShift コンプライアンス Operator 0.1.48 については、以下のアドバイザリーを利用できます。

- [RHBA-2022:0416 - OpenShift コンプライアンス Operator のバグ修正と機能拡張の更新](#)

### 5.1.8.1. バグ修正

- 以前は、拡張された Open Vulnerability and Assessment Language (OVAL) 定義に関連する一部のルールの **checkType** は **None** でした。これは、コンプライアンス Operator が、ルールを解析するときに拡張 OVAL 定義を処理していなかったためです。この更新により、拡張 OVAL 定義のコンテンツが解析され、これらのルールの **checkType** が **Node** または **Platform** になります。(BZ#2040282)
- 以前は、**KubeletConfig** 用に手動で作成された **MachineConfig** オブジェクトにより、**KubeletConfig** オブジェクトが修復用に生成されなくなり、修復が **Pending** 状態のままになりました。このリリースでは、**KubeletConfig** 用に手動で作成された **MachineConfig** オブジェクトがあるかどうかに関係なく、修復によって **KubeletConfig** オブジェクトが作成されます。その結果、**KubeletConfig** の修正が期待どおりに機能するようになりました。(BZ#2040401)

## 5.1.9. OpenShift コンプライアンス Operator 0.1.47

OpenShift コンプライアンス Operator 0.1.47 については、以下のアドバイザリーを利用できます。

- [RHBA-2022:0014 - OpenShift コンプライアンス Operator のバグ修正と機能拡張の更新](#)

### 5.1.9.1. 新機能および拡張機能

- コンプライアンス Operator は、Payment Card Industry Data Security Standard (PCI DSS) の次のコンプライアンスベンチマークをサポートするようになりました。
  - ocp4-pci-dss
  - ocp4-pci-dss-node
- FedRAMP の中程度の影響レベルの追加のルールと修正が、OCP4-moderate、OCP4-moderate-node、および rhcos4-moderate プロファイルに追加されます。

- KubeletConfig の修正がノードレベルのプロファイルで利用できるようになりました。

### 5.1.9.2. バグ修正

- 以前は、クラスターが OpenShift Container Platform 4.6 以前を実行していた場合、中程度のプロファイルでは USBGuard 関連のルールの修正が失敗していました。これは、コンプライアンス Operator によって作成された修正が、ドロップインディレクトリーをサポートしていない古いバージョンの USBGuard に基づいていたためです。現在、OpenShift Container Platform 4.6 を実行しているクラスターでは、USBGuard 関連のルールの無効な修復は作成されません。クラスターが OpenShift Container Platform 4.6 を使用している場合は、USBGuard 関連のルールの修正を手動で作成する必要があります。さらに、修正は、最小バージョン要件を満たすルールに対してのみ作成されます。[\(BZ#1965511\)](#)
- 以前のリリースでは、修正をレンダリングする場合には、コンプライアンス Operator は、その修正が適切に設定されているかを、厳密すぎる正規表現を使用して確認していました。その結果、**sshd\_config** をレンダリングするような一部の修正は、正規表現チェックに合格しないため、作成されませんでした。正規表現は不要であることが判明し、削除されました。修復が正しくレンダリングされるようになりました。[\(BZ#2033009\)](#)

### 5.1.10. OpenShift コンプライアンス Operator 0.1.44

OpenShift コンプライアンス Operator 0.1.44 については、以下のアドバイザリーを利用できます。

- [RHBA-2021:4530-OpenShift コンプライアンス Operator のバグ修正と機能拡張の更新](#)

#### 5.1.10.1. 新機能および拡張機能

- このリリースでは、**strictNodeScan** オプションが **ComplianceScan**、**ComplianceSuite**、および **ScanSetting** CR に追加されました。このオプションのデフォルトは **true** で、これは以前の動作と一致します。ノードでスキャンをスケジュールできなかった場合にエラーが発生しました。このオプションを **false** に設定すると、コンプライアンス Operator は、スキャンのスケジュールについてより寛容になります。エフェメラルノードのある環境では、**strictNodeScan** 値を **false** に設定できます。これにより、クラスター内の一部のノードがスケジューリングに使用できない場合でも、コンプライアンススキャンを続行できます。
- **ScanSetting** オブジェクトの **nodeSelector** 属性と **tolerations** 属性を設定することにより、結果サーバーのワークロードをスケジュールするために使用されるノードをカスタマイズできるようになりました。これらの属性は、PV ストレージボリュームをマウントし、生の Asset Reporting Format (ARF) 結果を格納するために使用される Pod である **ResultServer** Pod を配置するために使用されます。以前は、**nodeSelector** パラメーターおよび **tolerations** パラメーターは、デフォルトでコントロールプレーンノードの1つを選択し、**node-role.kubernetes.io/master taint** を許容していました。これは、コントロールプレーンノードが PV をマウントすることを許可されていない環境では機能しませんでした。この機能は、ノードを選択し、それらの環境で異なる汚染を許容する方法を提供します。
- コンプライアンス Operator は、**KubeletConfig** オブジェクトを修正できるようになりました。
- エラーメッセージを含むコメントが追加され、コンテンツ開発者がクラスター内に存在しないオブジェクトとフェッチできないオブジェクトを区別できるようになりました。
- ルールオブジェクトには、**checkType** および **description** の2つの新しい属性が含まれるようになりました。これらの属性を使用すると、ルールがノードチェックとプラットフォームチェックのどちらに関連しているかを判断したり、ルールの機能を確認したりできます。

- 今回の機能拡張により、既存のプロファイルを拡張して調整されたプロファイルを作成する必要があるという要件がなくなります。これは、**TailoredProfile** CRD の **extends** フィールドが必須ではなくなったことを意味します。これで、ルールオブジェクトのリストを選択して、調整されたプロファイルを作成できます。**compliance.openshift.io/product-type:** アノテーションを設定するか、**TailoredProfile** CR の **-node** 接尾辞を設定して、プロファイルをノードに適用するかプラットフォームに適用するかを選択する必要があることに注意してください。
- このリリースでは、コンプライアンス Operator は、汚染に関係なく、すべてのノードでスキャンをスケジュールできるようになりました。以前は、スキャン Pod は **node-role.kubernetes.io/master taint** のみを許容していました。つまり、汚染のないノードで実行するか、**node-role.kubernetes.io/master** 汚染のあるノードでのみ実行していました。ノードにカスタム汚染を使用するデプロイメントでは、これにより、それらのノードでスキャンがスケジュールされませんでした。現在、スキャン Pod はすべてのノードの汚染を許容します。
- このリリースでは、コンプライアンス Operator は、次の North American Electric Reliability Corporation (NERC) のセキュリティープロファイルをサポートしています。
  - ocp4-nerc-cip
  - ocp4-nerc-cip-node
  - rhcos4-nerc-cip
- このリリースでは、コンプライアンス Operator は、NIST 800-53 Moderate-Impact Baseline for the Red Hat OpenShift - Node level, ocp4-moderate-node セキュリティープロファイルをサポートします。

### 5.1.10.2. テンプレートと変数の使用

- このリリースでは、修復テンプレートで複数値の変数を使用できるようになりました。
- この更新により、コンプライアンス Operator は、コンプライアンスプロファイルで設定された変数に基づいて修正を変更できます。これは、タイムアウト、NTP サーバーのホスト名などのデプロイメント固有の値を含む修復に役立ちます。さらに、**ComplianceCheckResult** オブジェクトは、チェックが使用した変数をリストするラベル **compliance.openshift.io/check-has-value** を使用するようになりました。

### 5.1.10.3. バグ修正

- 以前は、スキャンの実行中に、Pod のスキャナーコンテナの1つで予期しない終了が発生しました。このリリースでは、コンプライアンス Operator は、クラッシュを回避するために最新の OpenSCAP バージョン 1.3.5 を使用します。
- 以前は、**autoReplyRemediations** を使用して修復を適用すると、クラスターノードの更新がトリガーされていました。一部の修復に必要な入力変数がすべて含まれていない場合、これは中断されました。これで、修復に1つ以上の必要な入力変数が欠落している場合、**NeedsReview** の状態が割り当てられます。1つ以上の修復が **NeedsReview** 状態にある場合、マシン設定プールは一時停止されたままになり、必要なすべての変数が設定されるまで修復は適用されません。これにより、ノードの中断を最小限に抑えることができます。
- Prometheus メトリックに使用される RBAC Role と Role Binding が ClusterRole と ClusterRoleBinding に変更なり、カスタマイズなしで監視が機能するようになりました。
- 以前は、プロファイルの解析中にエラーが発生した場合、ルールまたは変数オブジェクトが削除され、プロファイルから削除されていました。これで、解析中にエラーが発生した場合、**profileparser** はオブジェクトに一時的な注釈を付けて、解析が完了するまでオブジェクト

が削除されないようにします。(BZ#1988259)

- 以前のバージョンでは、調整されたプロファイルにタイトルまたは説明がない場合、エラーが発生しました。XCCDF 標準では、調整されたプロファイルのタイトルと説明が必要なため、タイトルと説明を **TailoredProfile** CR で設定する必要があります。
- 以前は、調整されたプロファイルを使用する場合、特定の選択セットのみを使用して **TailoredProfile** 変数値を設定できました。この制限がなくなり、**TailoredProfile** 変数を任意の値に設定できるようになりました。

### 5.1.11. コンプライアンス Operator 0.1.39 リリースノート

OpenShift コンプライアンス Operator 0.1.39 については、以下のアドバイザリーが利用できます。

- [RHBA-2021:3214 - OpenShift コンプライアンス Operator のバグ修正と機能拡張の更新](#)

#### 5.1.11.1. 新機能および拡張機能

- 以前は、コンプライアンス Operator は、Payment Card Industry Data Security Standard (PCI DSS) 参照を解析できませんでした。これで、オペレーターは PCI DSS プロファイルに付属するコンプライアンスコンテンツを解析できます。
- 以前は、コンプライアンス Operator は、中程度のプロファイルで AU-5 制御のルールを実行できませんでした。これで、**Prometheusrules.monitoring.coreos.com** オブジェクトを読み取り、中程度のプロファイルで AU-5 コントロールを扱うルールを実行できるように、Operator にアクセス許可が追加されました。

### 5.1.12. 関連情報

- [コンプライアンス Operator について](#)

## 5.2. サポートされているコンプライアンスプロファイル

コンプライアンス Operator (CO) のインストールの一部として利用可能なプロファイルは複数あります。



### 重要

Compliance Operator は、OpenShift Dedicated、Red Hat OpenShift Service on AWS、Azure Red Hat OpenShift などのマネージドプラットフォームで誤った結果を報告する場合があります。詳細は、Red Hat ナレッジベースソリューション [Red Hat Knowledgebase Solution #6983418](#) を参照してください。

### 5.2.1. コンプライアンスプロファイル

コンプライアンス Operator は、次のコンプライアンスプロファイルを提供します。

表5.1 サポートされているコンプライアンスプロファイル



プロファイル	プロファイルタイトル	コンプライアンス Operator バージョン	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー
ocp4-cis	CIS Red Hat OpenShift Container Platform 4 Benchmark	0.139 +	<a href="#">CIS Benchmarks™</a> footnote:cisbenchmark[CIS Red Hat OpenShift Container Platform v4 ベンチマークを見つけるには、 <a href="#">CIS ベンチマーク</a> に移動し、検索ボックスに <b>Kubernetes</b> と入力します。Kubernetes をクリックしてから <b>Download Latest CIS Benchmark</b> をクリックすると、登録してベンチマークをダウンロードできます。	x86_64 ppc64le s390x
ocp4-cis-node	CIS Red Hat OpenShift Container Platform 4 Benchmark	0.139 +	<a href="#">CIS Benchmarks™</a> footnote:cisbenchmark[]	x86_64 ppc64le s390x
ocp4-e8	Australian Cyber Security Centre (ACSC) Essential Eight	0.139 +	<a href="#">ACSC Hardening Linux ワークステーションおよびサーバー</a>	x86_64
ocp4-moderate	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - Platform level	0.139 +	<a href="#">NIST SP-800-53 Release Search</a>	x86_64
rhcos 4-e8	Australian Cyber Security Centre (ACSC) Essential Eight	0.139 +	<a href="#">ACSC Hardening Linux ワークステーションおよびサーバー</a>	x86_64
rhcos 4-moderate	NIST 800-53 Moderate-Impact Baseline for Red Hat Enterprise Linux CoreOS	0.139 +	<a href="#">NIST SP-800-53 Release Search</a>	x86_64
ocp4-moderate-node	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - Node level	0.144 +	<a href="#">NIST SP-800-53 Release Search</a>	x86_64
ocp4-nerc-cip	North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity standards profile for the Red Hat OpenShift Container Platform - Platform レベル	0.144 +	<a href="#">NERC CIP 標準</a>	x86_64

プロファイル	プロファイルタイトル	コンプライアンス Operator バージョン	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー
ocp4- nerc- cip- node	North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity standards profile for the Red Hat OpenShift Container Platform - Node レベル	0.144 +	<a href="#">NERC CIP 標準</a>	x86_64
rhcos 4- nerc- cip	North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity standards profile for Red Hat Enterprise Linux CoreOS	0.144 +	<a href="#">NERC CIP 標準</a>	x86_64
ocp4- pci- dss	PCI-DSS v3.2.1 Control Baseline for Red Hat OpenShift Container Platform 4	0.147 +	<a href="#">PCI Security Standards® Council Document Library</a>	x86_64 ppc64le
ocp4- pci- dss- node	PCI-DSS v3.2.1 Control Baseline for Red Hat OpenShift Container Platform 4	0.147 +	<a href="#">PCI Security Standards® Council Document Library</a>	x86_64 ppc64le
ocp4- high	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - プラットフォームレベル	0.152 +	<a href="#">NIST SP-800-53 Release Search</a>	x86_64
ocp4- high- node	NIST 800-53 High-Impact Baseline for Red Hat OpenShift - ノードレベル	0.152 +	<a href="#">NIST SP-800-53 Release Search</a>	x86_64
rhcos 4-high	NIST 800-53 High-Impact Baseline for Red Hat Enterprise Linux CoreOS	0.152 +	<a href="#">NIST SP-800-53 Release Search</a>	x86_64

### 5.2.2. 関連情報

- システムで使用可能なコンプライアンスプロファイルの表示の詳細については、Compliance Operator についての [Compliance Operator profiles](#) を参照してください。

### 5.3. コンプライアンス OPERATOR のインストール

コンプライアンス Operator を使用する前に、これがクラスターにデプロイされていることを確認する必要があります。

### 5.3.1. Web コンソールを使用したコンプライアンス Operator のインストール

#### 前提条件

- **admin** 権限がある。

#### 手順

1. OpenShift Container Platform Web コンソールで、**Operators** → **OperatorHub** ページに移動します。
2. コンプライアンス Operator を検索し、**Install** をクリックします。
3. **Installation mode** および **namespace** のデフォルトの選択を維持し、Operator が **openshift-compliance** namespace にインストールされていることを確認します。
4. **Install** をクリックします。

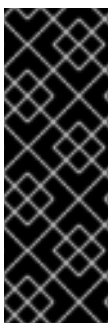
#### 検証

インストールが正常に行われたことを確認するには、以下を実行します。

1. **Operators** → **Installed Operators** ページに移動します。
2. コンプライアンス Operator が **openshift-compliance** namespace にインストールされ、そのステータスが **Succeeded** であることを確認します。

Operator が正常にインストールされていない場合、以下を実行します。

1. **Operators** → **Installed Operators** ページに移動し、**Status** 列でエラーまたは失敗の有無を確認します。
2. **Workloads** → **Pods** ページに移動し、**openshift-compliance** プロジェクトの Pod で問題を報告しているログの有無を確認します。



#### 重要

**restricted** なセキュリティーコンテキスト制約 (SCC) が **system:authenticated** グループを含むように変更されているか、**requiredDropCapabilities** を追加している場合、権限の問題により コンプライアンス Operator が正しく機能しない可能性があります。

ComplianceOperator スキャナー Pod サービスアカウント用のカスタム SCC を作成できます。詳細については [Creating a custom SCC for the Compliance Operator](#) を参照してください。

### 5.3.2. CLI を使用したコンプライアンス Operator のインストール

#### 前提条件

- **admin** 権限がある。

#### 手順

1. **Namespace** オブジェクトを定義します。

### namespace-object.yaml の例

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: openshift-compliance
```

2. **Namespace** オブジェクトを作成します。

```
$ oc create -f namespace-object.yaml
```

3. **OperatorGroup** オブジェクトを定義します。

### operator-group-object.yaml の例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: compliance-operator
  namespace: openshift-compliance
spec:
  targetNamespaces:
    - openshift-compliance
```

4. **OperatorGroup** オブジェクトを作成します。

```
$ oc create -f operator-group-object.yaml
```

5. **Subscription** オブジェクトを定義します。

### subscription-object.yaml の例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: compliance-operator-sub
  namespace: openshift-compliance
spec:
  channel: "release-0.1"
  installPlanApproval: Automatic
  name: compliance-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

6. **Subscription** オブジェクトを作成します。

```
$ oc create -f subscription-object.yaml
```



## 注記

グローバルスケジューラー機能を設定する際に、**defaultNodeSelector** を有効にする場合、namespace を手動で作成し、**openshift-compliance** のアノテーションを更新するか、または **openshift.io/node-selector: ""** を使用してコンプライアンス Operator がインストールされている namespace のアノテーションを更新する必要があります。これにより、デフォルトのノードセクターが削除され、デプロイメントの失敗を防ぐことができます。

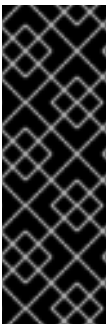
## 検証

1. CSV ファイルを確認して、インストールが正常に完了したことを確認します。

```
$ oc get csv -n openshift-compliance
```

2. コンプライアンス Operator が稼働していることを確認します。

```
$ oc get deploy -n openshift-compliance
```



## 重要

**restricted** なセキュリティーコンテキスト制約 (SCC) が **system:authenticated** グループを含むように変更されているか、**requiredDropCapabilities** を追加している場合、権限の問題により コンプライアンス Operator が正しく機能しない可能性があります。

ComplianceOperator スキャナー Pod サービスアカウント用のカスタム SCC を作成できません。詳細について [Creating a custom SCC for the Compliance Operator](#) を参照してください。

### 5.3.3. 関連情報

- コンプライアンス Operator はネットワークが制限された環境でサポートされています。詳細は、[ネットワークが制限された環境での Operator Lifecycle Manager の使用](#) を参照してください。

## 5.4. コンプライアンス OPERATOR の更新

クラスター管理者は、OpenShift Container Platform クラスターで コンプライアンス Operator を更新できます。

### 5.4.1. Operator 更新の準備

インストールされた Operator のサブスクリプションは、Operator の更新を追跡および受信する更新チャンネルを指定します。更新チャンネルを変更して、新しいチャンネルからの更新の追跡と受信を開始できます。

サブスクリプションの更新チャンネルの名前は Operator 間で異なる可能性があります。命名スキーム通常、特定の Operator 内の共通の規則に従います。たとえば、チャンネル名は Operator によって提供されるアプリケーションのマイナーリリース更新ストリーム (**1.2**、**1.3**) またはリリース頻度 (**stable**、**fast**) に基づく可能性があります。



## 注記

インストールされた Operator は、現在のチャンネルよりも古いチャンネルに切り換えることはできません。

Red Hat Customer Portal Labs には、管理者が Operator の更新を準備するのに役立つ以下のアプリケーションが含まれています。

- [Red Hat OpenShift Container プラットフォーム Operator Update Information Checker](#)

このアプリケーションを使用して、Operator Lifecycle Manager ベースの Operator を検索し、OpenShift Container Platform の異なるバージョン間で更新チャンネルごとに利用可能な Operator バージョンを確認できます。Cluster Version Operator ベースの Operator は含まれません。

### 5.4.2. Operator の更新チャンネルの変更

OpenShift Container Platform Web コンソールを使用して、Operator の更新チャンネルを変更できます。

#### ヒント

サブスクリプションの承認ストラテジーが **Automatic** に設定されている場合、アップグレードプロセスは、選択したチャンネルで新規 Operator バージョンが利用可能になるとすぐに開始します。承認ストラテジーが **Manual** に設定されている場合は、保留中のアップグレードを手動で承認する必要があります。

#### 前提条件

- Operator Lifecycle Manager (OLM) を使用して以前にインストールされている Operator。

#### 手順

1. Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** に移動します。
2. 更新チャンネルを変更する Operator の名前をクリックします。
3. **Subscription** タブをクリックします。
4. **Channel** の下にある更新チャンネルの名前をクリックします。
5. 変更する新しい更新チャンネルをクリックし、**Save** をクリックします。
6. **Automatic** 承認ストラテジーのあるサブスクリプションの場合、更新は自動的に開始します。**Operators → Installed Operators** ページに戻り、更新の進捗をモニターします。完了時に、ステータスは **Succeeded** および **Up to date** に変更されます。  
**Manual** 承認ストラテジーのあるサブスクリプションの場合、**Subscription** タブから更新を手動で承認できます。

### 5.4.3. 保留中の Operator 更新の手動による承認

インストールされた Operator のサブスクリプションの承認ストラテジーが **Manual** に設定されている場合、新規の更新が現在の更新チャンネルにリリースされると、インストールを開始する前に更新を手動で承認する必要があります。

## 前提条件

- Operator Lifecycle Manager (OLM) を使用して以前にインストールされている Operator。

## 手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Operators** → **Installed Operators** に移動します。
2. 更新が保留中の Operator は **Upgrade available** のステータスを表示します。更新する Operator の名前をクリックします。
3. **Subscription** タブをクリックします。承認が必要な更新は、**アップグレードステータス** の横に表示されます。たとえば、**1 requires approval** が表示される可能性があります。
4. **1 requires approval** をクリックしてから、**Preview Install Plan** をクリックします。
5. 更新に利用可能なリソースとして一覧表示されているリソースを確認します。問題がなければ、**Approve** をクリックします。
6. **Operators** → **Installed Operators** ページに戻り、更新の進捗をモニターします。完了時に、ステータスは **Succeeded** および **Up to date** に変更されます。

## 5.5. コンプライアンス OPERATOR のスキャン

**ScanSetting** および **ScanSettingBinding** API は、コンプライアンス Operator でコンプライアンススキャンを実行するために使用することが推奨されます。これらの API オブジェクトの詳細については、以下を実行します。

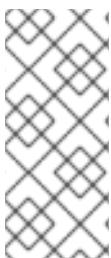
```
$ oc explain scansettings
```

または

```
$ oc explain scansettingbindings
```

### 5.5.1. コンプライアンススキャンの実行

Center for Internet Security (CIS) プロファイルを使用してスキャンを実行できます。便宜上、コンプライアンス Operator は起動時に適切なデフォルト値を持つ **ScanSetting** オブジェクトを作成します。この **ScanSetting** オブジェクトの名前は **default** です。



#### 注記

オールインワンのコントロールプレーンノードとワーカーノードの場合、コンプライアンススキャンはワーカーノードとコントロールプレーンノードで2回実行されます。コンプライアンススキャンは、一貫性のないスキャン結果を生成する可能性があります。**ScanSetting** オブジェクトで単一のロールのみを定義することにより、一貫性のない結果を回避できます。

## 手順

1. 以下を実行して **ScanSetting** オブジェクトを検査します。

```
$ oc describe scansettings default -n openshift-compliance
```

## 出力例

```

Name:      default
Namespace: openshift-compliance
Labels:    <none>
Annotations: <none>
API Version: compliance.openshift.io/v1alpha1
Kind:      ScanSetting
Metadata:
  Creation Timestamp: 2022-10-10T14:07:29Z
  Generation:        1
  Managed Fields:
    API Version: compliance.openshift.io/v1alpha1
    Fields Type: FieldsV1
    fieldsV1:
      f:rawResultStorage:
        ..
      f:nodeSelector:
        ..
      f:node-role.kubernetes.io/master:
      f:pvAccessModes:
      f:rotation:
      f:size:
      f:tolerations:
      f:roles:
      f:scanTolerations:
      f:schedule:
      f:showNotApplicable:
      f:strictNodeScan:
  Manager:      compliance-operator
  Operation:    Update
  Time:         2022-10-10T14:07:29Z
  Resource Version: 56111
  UID:          c21d1d14-3472-47d7-a450-b924287aec90
Raw Result Storage:
Node Selector:
  node-role.kubernetes.io/master:
Pv Access Modes:
  ReadWriteOnce 1
Rotation: 3 2
Size: 1Gi 3
Tolerations:
  Effect:      NoSchedule
  Key:         node-role.kubernetes.io/master
  Operator:    Exists
  Effect:      NoExecute
  Key:         node.kubernetes.io/not-ready
  Operator:    Exists
  Toleration Seconds: 300
  Effect:      NoExecute
  Key:         node.kubernetes.io/unreachable
  Operator:    Exists
  Toleration Seconds: 300
  Effect:      NoSchedule

```



```

Key:          node.kubernetes.io/memory-pressure
Operator:     Exists
Roles:
  master ④
  worker ⑤
Scan Tolerations: ⑥
  Operator:   Exists
Schedule:     0 1 * * * ⑦
Show Not Applicable: false
Strict Node Scan: true
Events:      <none>

```

- ① コンプライアンス Operator は、スキャンの結果が含まれる永続ボリューム (PV) を作成します。デフォルトで、コンプライアンス Operator はクラスターに設定されるストレージクラスについて何らかの仮定をすることができないため、PV はアクセスモード **ReadWriteOnce** を使用します。さらに、**ReadWriteOnce** アクセスモードはほとんどのクラスターで利用できます。スキャン結果を取得する必要がある場合は、ボリュームもバインドするヘルパー Pod を使用して実行できます。**ReadWriteOnce** アクセスモードを使用するボリュームは、一度に1つの Pod のみがマウントできるため、必ずヘルパー Pod を削除してください。そうでない場合は、コンプライアンス Operator は後続のスキャンのためにボリュームを再利用できません。
- ② コンプライアンス Operator は、後続の3つのスキャンの結果をボリュームに保持し、古いスキャンはローテーションされます。
- ③ コンプライアンス Operator はスキャンの結果用に1GBのストレージを割り当てます。
- ④ ⑤ スキャン設定がクラスターノードをスキャンするプロファイルを使用する場合は、これらのノードロールをスキャンします。
- ⑥ デフォルトのスキャン設定オブジェクトは、すべてのノードをスキャンします。
- ⑦ デフォルトのスキャン設定オブジェクトは、毎日 01:00 にスキャンを実行します。

デフォルトのスキャン設定の代わりに、以下の設定を含む **default-auto-apply** を使用できます。

```

Name:          default-auto-apply
Namespace:     openshift-compliance
Labels:        <none>
Annotations:   <none>
API Version:   compliance.openshift.io/v1alpha1
Auto Apply Remediations: true
Auto Update Remediations: true
Kind:          ScanSetting
Metadata:
  Creation Timestamp: 2022-10-18T20:21:00Z
  Generation:        1
  Managed Fields:
    API Version: compliance.openshift.io/v1alpha1
    Fields Type: FieldsV1
    fieldsV1:
      f:autoApplyRemediations: ①
      f:autoUpdateRemediations: ②

```

```

f:rawResultStorage:
  .:
  f:nodeSelector:
    .:
    f:node-role.kubernetes.io/master:
  f:pvAccessModes:
  f:rotation:
  f:size:
  f:tolerations:
  f:roles:
  f:scanTolerations:
  f:schedule:
  f:showNotApplicable:
  f:strictNodeScan:
  Manager:      compliance-operator
  Operation:    Update
  Time:         2022-10-18T20:21:00Z
  Resource Version: 38840
  UID:         8cb0967d-05e0-4d7a-ac1c-08a7f7e89e84
Raw Result Storage:
Node Selector:
  node-role.kubernetes.io/master:
Pv Access Modes:
  ReadWriteOnce
Rotation: 3
Size: 1Gi
Tolerations:
  Effect:      NoSchedule
  Key:         node-role.kubernetes.io/master
  Operator:    Exists
  Effect:      NoExecute
  Key:         node.kubernetes.io/not-ready
  Operator:    Exists
  Toleration Seconds: 300
  Effect:      NoExecute
  Key:         node.kubernetes.io/unreachable
  Operator:    Exists
  Toleration Seconds: 300
  Effect:      NoSchedule
  Key:         node.kubernetes.io/memory-pressure
  Operator:    Exists
Roles:
  master
  worker
Scan Tolerations:
  Operator:    Exists
Schedule:     0 1 * * *
Show Not Applicable: false
Strict Node Scan: true
Events:      <none>

```

- 1 2 **autoUpdateRemediations** および **autoApplyRemediations** フラグを **true** に設定すると、追加の手順なしに自動修復する **ScanSetting** オブジェクトを簡単に作成できます。

2. デフォルトの **ScanSetting** オブジェクトにバインドし、**cis** および **cis-node** プロファイルを使用してクラスターをスキャンする **ScanSettingBinding** オブジェクトを作成します。以下に例を示します。

```

apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: cis-compliance
  namespace: openshift-compliance
profiles:
  - name: ocp4-cis-node
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
  - name: ocp4-cis
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
settingsRef:
  name: default
  kind: ScanSetting
  apiGroup: compliance.openshift.io/v1alpha1

```

3. 以下を実行して **ScanSettingBinding** オブジェクトを作成します。

```
$ oc create -f <file-name>.yaml -n openshift-compliance
```

プロセスのこの時点で、**ScanSettingBinding** オブジェクトは調整され、**Binding** および **Bound** 設定に基づいて調整されます。コンプライアンス Operator は **ComplianceSuite** オブジェクトおよび関連付けられる **ComplianceScan** オブジェクトを作成します。

4. 以下を実行してコンプライアンススキャンの進捗を確認します。

```
$ oc get compliancescan -w -n openshift-compliance
```

スキャンはスキャンフェーズに進み、完了すると最終的に **DONE** フェーズに移行します。ほとんどの場合、スキャンの結果は **NON-COMPLIANT** になります。スキャン結果を確認し、クラスターがコンプライアンスに基づくように修復の適用を開始することができます。詳細は、[コンプライアンス Operator 修復の管理](#) を参照してください。

### 5.5.2. ワーカーノードでの結果サーバー Pod のスケジューリング

結果サーバー Pod は、生の Asset Reporting Format (ARF) スキャン結果を格納する永続ボリューム (PV) をマウントします。**nodeSelector** 属性および **tolerations** 属性を使用すると、結果サーバー Pod の場所を設定できます。

これは、コントロールプレーンノードが永続ボリュームをマウントすることを許可されていない環境で役立ちます。

#### 手順

- コンプライアンス Operator 用の **ScanSetting** カスタムリソース (CR) を作成します。
  - a. **ScanSetting** CR を定義し、YAML ファイルを保存します (例: **rs-workers.yaml**)。

```

apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting

```

```

metadata:
  name: rs-on-workers
  namespace: openshift-compliance
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/worker: "" ❶
  pvAccessModes:
  - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:
  - operator: Exists ❷
roles:
  - worker
  - master
scanTolerations:
  - operator: Exists
schedule: 0 1 * * *

```

- ❶ コンプライアンス Operator は、このノードを使用してスキャン結果を ARF 形式で保存します。
- ❷ 結果のサーバー Pod は、すべての汚染を許容します。

b. **ScanSetting** CR を作成するには、次のコマンドを実行します。

```
$ oc create -f rs-workers.yaml
```

## 検証

- **ScanSetting** オブジェクトが作成されたことを確認するには、次のコマンドを実行します。

```
$ oc get scansettings rs-on-workers -n openshift-compliance -o yaml
```

## 出力例

```

apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  creationTimestamp: "2021-11-19T19:36:36Z"
  generation: 1
  name: rs-on-workers
  namespace: openshift-compliance
  resourceVersion: "48305"
  uid: 43fdcf5f-15a7-445a-8bbc-0e4a160cd46e
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  pvAccessModes:
  - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:

```

```

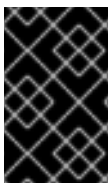
- operator: Exists
roles:
- worker
- master
scanTolerations:
- operator: Exists
schedule: 0 1 * * *
strictNodeScan: true

```

### 5.5.3. ScanSetting カスタムリソース

**ScanSetting** カスタムリソースでは、scan limits 属性を使用して、スキャナー Pod のデフォルトの CPU およびメモリー制限をオーバーライドできるようになりました。コンプライアンス Operator は、スキャナーコンテナに 500Mi メモリー、100m CPU のデフォルトを使用し、**api-resource-collector** コンテナに 100m CPU の 200Mi メモリーを使用します。Operator のメモリー制限を設定するには、OLM または Operator デプロイメント自体を介してインストールされている場合は **Subscription** オブジェクトを変更します。

コンプライアンス Operator のデフォルトの CPU およびメモリーの制限を増やすには、**コンプライアンス Operator リソース制限の増加** を参照してください。



#### 重要

デフォルトの制限が十分ではなく、Operator またはスキャナー Pod が Out Of Memory (OOM) プロセスによって終了した場合は、コンプライアンス Operator または Scanner Pod のメモリー制限を増やす必要があります。

### 5.5.4. リソース要求および制限の適用

kubelet が Pod の一部としてコンテナを起動すると、kubelet はそのコンテナの要求および制限をメモリーおよび CPU の要求および制限をコンテナランタイムに渡します。Linux では、コンテナランタイムは、定義した制限を適用して有効にするカーネル cgroup を設定します。

CPU 制限は、コンテナが使用できる CPU 時間を定義します。各スケジューリング期間中、Linux カーネルはこの制限を超えるかどうかを確認します。その場合、カーネルは、cgroup の実行を再開できるようにするまで待機します。

複数の異なるコンテナ (cgroup) を競合するシステムで実行する場合、CPU 要求が大きいワークロードには、要求が小さいワークロードよりも多くの CPU 時間が割り当てられます。メモリー要求は Pod のスケジューリング時に使用されます。cgroups v2 を使用するノードでは、コンテナランタイムがメモリーリクエストをヒントとして使用して、**memory.min** および **memory.low** の値を設定する場合があります。

コンテナがこの制限を超えるメモリーを割り当てようとする、Linux カーネルのメモリー不足サブシステムがアクティブになり、メモリーを割り当てようとしたコンテナ内のプロセスの1つを停止して介入します。Pod またはコンテナのメモリー制限は、emptyDir などのメモリーベースのボリュームのページにも適用できます。

kubelet は、ローカルの一時的ストレージとしてではなく、コンテナメモリーが使用されているときに **tmpfs emptyDir** ボリュームを追跡します。コンテナがメモリー要求を超え、それが実行されているノードが全体的にメモリー不足になった場合、Pod のコンテナが削除される可能性があります。



## 重要

コンテナは、長期間にわたって CPU 制限を超えることはできません。コンテナのランタイムは、CPU 使用率が過剰に使用されている場合も Pod またはコンテナを停止しません。リソース制限のためにコンテナをスケジュールできないか、強制終了されているかを判断するには、[コンプライアンス Operator のトラブルシューティング](#)を参照してください。

### 5.5.5. コンテナリソース要求を使用した Pod のスケジューリング

Pod が作成されると、スケジューラーは Pod を実行するノードを選択します。各ノードには、リソースタイプごとに、Pod に提供できる CPU とメモリの最大容量があります。スケジューラーは、スケジュールされたコンテナのリソース要求の合計が、各リソースタイプの容量ノードよりも少なくなるようにします。

ノードのメモリまたは CPU リソースの使用率が非常に低い場合でも、容量チェックでノードのリソース不足を防ぐことができない場合、スケジューラーはノードへの Pod の配置を拒否することがあります。

コンテナごとに、以下のリソース制限および要求を指定できます。

```
spec.containers[].resources.limits.cpu
spec.containers[].resources.limits.memory
spec.containers[].resources.limits.hugepages-<size>
spec.containers[].resources.requests.cpu
spec.containers[].resources.requests.memory
spec.containers[].resources.requests.hugepages-<size>
```

個々のコンテナに対してのみ要求と制限を指定できますが、Pod の全体的なリソース要求と制限を考慮することも役立ちます。特定のリソースの場合には、コンテナリソースの要求または制限は、Pod 内にあるコンテナごとに割り当てられた、対象タイプのリソース要求または制限を合計したものです。

### コンテナリソース要求および制限の例

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests: 1
        memory: "64Mi"
        cpu: "250m"
      limits: 2
        memory: "128Mi"
        cpu: "500m"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
```

```
cpu: "250m"
limits:
  memory: "128Mi"
  cpu: "500m"
```

- 1 コンテナは 64 Mi のメモリーと 250 m CPU を要求しています。
- 2 コンテナの制限は 128 Mi のメモリーと 500 m CPU です。

## 5.6. コンプライアンス OPERATOR について

コンプライアンス Operator を使用すると、OpenShift Container Platform 管理者はクラスターの必要なコンプライアンス状態を記述し、存在するギャップやそれらを修復する方法についての概要を提供します。コンプライアンス Operator は、OpenShift Container Platform の Kubernetes API リソースと、クラスターを実行するノードの両方のコンプライアンスを評価します。コンプライアンス Operator は、NIST 認定ツールである OpenSCAP を使用して、コンテンツが提供するセキュリティーポリシーをスキャンし、これを適用します。



### 重要

コンプライアンス Operator は Red Hat Enterprise Linux CoreOS (RHCOS) デプロイメントでのみ利用できます。

### 5.6.1. Compliance Operator のプロファイル

コンプライアンス Operator のインストールの一部として利用可能なプロファイルは複数あります。**oc get** コマンドを使用して、使用可能なプロファイル、プロファイルの詳細、および特定のルールを表示できます。

- 利用可能なプロファイルを表示します。

```
$ oc get -n openshift-compliance profiles.compliance
```

#### 出力例

```
NAME                AGE
ocp4-cis             94m
ocp4-cis-node        94m
ocp4-e8              94m
ocp4-high            94m
ocp4-high-node       94m
ocp4-moderate        94m
ocp4-moderate-node  94m
ocp4-nerc-cip        94m
ocp4-nerc-cip-node  94m
ocp4-pci-dss         94m
ocp4-pci-dss-node   94m
rhcos4-e8            94m
rhcos4-high          94m
rhcos4-moderate      94m
rhcos4-nerc-cip     94m
```

これらのプロファイルは、複数の異なるコンプライアンスベンチマークを表します。各プロ

ファイルには、適用先の製品名がプロファイル名の接頭辞として追加されます。**ocp4-e8** は Essential 8 ベンチマークを OpenShift Container Platform 製品に適用し、**rhcos4-e8** は Essential 8 ベンチマークを Red Hat Enterprise Linux CoreOS (RHCOS) 製品に適用します。

- 以下のコマンドを実行して、**rhcos4-e8** プロファイルの詳細を表示します。

```
$ oc get -n openshift-compliance -oyaml profiles.compliance rhcos4-e8
```

## 出力例

```
apiVersion: compliance.openshift.io/v1alpha1
description: 'This profile contains configuration checks for Red Hat Enterprise Linux
  CoreOS that align to the Australian Cyber Security Centre (ACSC) Essential Eight.
  A copy of the Essential Eight in Linux Environments guide can be found at the ACSC
  website: https://www.cyber.gov.au/acsc/view-all-content/publications/hardening-linux-
  workstations-and-servers'
id: xccdf_org.ssgproject.content_profile_e8
kind: Profile
metadata:
  annotations:
    compliance.openshift.io/image-digest: pb-rhcos4hrdkm
    compliance.openshift.io/product: redhat_enterprise_linux_coreos_4
    compliance.openshift.io/product-type: Node
  creationTimestamp: "2022-10-19T12:06:49Z"
  generation: 1
  labels:
    compliance.openshift.io/profile-bundle: rhcos4
  name: rhcos4-e8
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ProfileBundle
    name: rhcos4
    uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
  resourceVersion: "43699"
  uid: 86353f70-28f7-40b4-bf0e-6289ec33675b
rules:
- rhcos4-accounts-no-uid-except-zero
- rhcos4-audit-rules-dac-modification-chmod
- rhcos4-audit-rules-dac-modification-chown
- rhcos4-audit-rules-execution-chcon
- rhcos4-audit-rules-execution-restorecon
- rhcos4-audit-rules-execution-semanage
- rhcos4-audit-rules-execution-setfiles
- rhcos4-audit-rules-execution-setsebool
- rhcos4-audit-rules-execution-seunshare
- rhcos4-audit-rules-kernel-module-loading-delete
- rhcos4-audit-rules-kernel-module-loading-finit
- rhcos4-audit-rules-kernel-module-loading-init
- rhcos4-audit-rules-login-events
- rhcos4-audit-rules-login-events-faillock
- rhcos4-audit-rules-login-events-lastlog
- rhcos4-audit-rules-login-events-tallylog
```



```

- rhcos4-audit-rules-networkconfig-modification
- rhcos4-audit-rules-sysadmin-actions
- rhcos4-audit-rules-time-adjtimex
- rhcos4-audit-rules-time-clock-settime
- rhcos4-audit-rules-time-settimeofday
- rhcos4-audit-rules-time-stime
- rhcos4-audit-rules-time-watch-localtime
- rhcos4-audit-rules-usergroup-modification
- rhcos4-auditd-data-retention-flush
- rhcos4-auditd-freq
- rhcos4-auditd-local-events
- rhcos4-auditd-log-format
- rhcos4-auditd-name-format
- rhcos4-auditd-write-logs
- rhcos4-configure-crypto-policy
- rhcos4-configure-ssh-crypto-policy
- rhcos4-no-empty-passwords
- rhcos4-selinux-policytype
- rhcos4-selinux-state
- rhcos4-service-auditd-enabled
- rhcos4-sshd-disable-empty-passwords
- rhcos4-sshd-disable-gssapi-auth
- rhcos4-sshd-disable-rhosts
- rhcos4-sshd-disable-root-login
- rhcos4-sshd-disable-user-known-hosts
- rhcos4-sshd-do-not-permit-user-env
- rhcos4-sshd-enable-strictmodes
- rhcos4-sshd-print-last-log
- rhcos4-sshd-set-loglevel-info
- rhcos4-sysctl-kernel-dmesg-restrict
- rhcos4-sysctl-kernel-kptr-restrict
- rhcos4-sysctl-kernel-randomize-va-space
- rhcos4-sysctl-kernel-unprivileged-bpf-disabled
- rhcos4-sysctl-kernel-yama-pttrace-scope
- rhcos4-sysctl-net-core-bpf-jit-harden
title: Australian Cyber Security Centre (ACSC) Essential Eight

```

- 以下のコマンドを実行して、**rhcos4-audit-rules-login-events** ルールの詳細を表示します。

```
$ oc get -n openshift-compliance -oyaml rules rhcos4-audit-rules-login-events
```

## 出力例

```

apiVersion: compliance.openshift.io/v1alpha1
checkType: Node
description: |-
  The audit system already collects login information for all users and root. If the auditd
  daemon is configured to use the augenrules program to read audit rules during daemon
  startup (the default), add the following lines to a file with suffix.rules in the directory
  /etc/audit/rules.d in order to watch for attempted manual edits of files involved in storing
  logon events:

```

```

-w /var/log/tallylog -p wa -k logins
-w /var/run/faillock -p wa -k logins
-w /var/log/lastlog -p wa -k logins

```

If the auditd daemon is configured to use the auditctl utility to read audit rules during daemon startup, add the following lines to /etc/audit/audit.rules file in order to watch for unattempted manual edits of files involved in storing logon events:

```
-w /var/log/tallylog -p wa -k logins
-w /var/run/faillock -p wa -k logins
-w /var/log/lastlog -p wa -k logins
id: xccdf_org.ssgproject.content_rule_audit_rules_login_events
kind: Rule
metadata:
  annotations:
    compliance.openshift.io/image-digest: pb-rhcos4hrdkm
    compliance.openshift.io/rule: audit-rules-login-events
    control.compliance.openshift.io/NIST-800-53: AU-2(d);AU-12(c);AC-6(9);CM-6(a)
    control.compliance.openshift.io/PCI-DSS: Req-10.2.3
    policies.open-cluster-management.io/controls: AU-2(d),AU-12(c),AC-6(9),CM-6(a),Req-10.2.3
    policies.open-cluster-management.io/standards: NIST-800-53,PCI-DSS
  creationTimestamp: "2022-10-19T12:07:08Z"
  generation: 1
  labels:
    compliance.openshift.io/profile-bundle: rhcos4
  name: rhcos4-audit-rules-login-events
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ProfileBundle
    name: rhcos4
    uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
  resourceVersion: "44819"
  uid: 75872f1f-3c93-40ca-a69d-44e5438824a4
rationale: Manual editing of these files may indicate nefarious activity, such as
an attacker attempting to remove evidence of an intrusion.
severity: medium
title: Record Attempts to Alter Logon and Logout Events
warning: Manual editing of these files may indicate nefarious activity, such as an
attacker attempting to remove evidence of an intrusion.
```

## 5.7. コンプライアンス OPERATOR の管理

本セクションでは、コンプライアンスコンテンツの更新されたバージョンを使用する方法や、カスタム **ProfileBundle** オブジェクトを作成する方法など、セキュリティコンテンツのライフサイクルについて説明します。

### 5.7.1. ProfileBundle CR の例

**ProfileBundle** オブジェクトには、**contentImage** が含まれるコンテナイメージの URL と、コンプライアンスコンテンツが含まれるファイルの 2 つの情報が必要です。**contentFile** パラメーターはファイルシステムのルートに相対します。以下の例のように、ビルトインの **rhcos4 ProfileBundle** オブジェクトを定義できます。

```
apiVersion: compliance.openshift.io/v1alpha1
```

```

kind: ProfileBundle
metadata:
  creationTimestamp: "2022-10-19T12:06:30Z"
  finalizers:
  - profilebundle.finalizers.compliance.openshift.io
  generation: 1
  name: rhcos4
  namespace: openshift-compliance
  resourceVersion: "46741"
  uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
spec:
  contentFile: ssg-rhcos4-ds.xml 1
  contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:900e...
2
status:
  conditions:
  - lastTransitionTime: "2022-10-19T12:07:51Z"
    message: Profile bundle successfully parsed
    reason: Valid
    status: "True"
    type: Ready
  dataStreamStatus: VALID

```

**1** コンプライアンスコンテンツが含まれるファイルの場所。

**2** コンテンツイメージの場所。



### 重要

コンテンツイメージに使用されるベースイメージには、**coreutils** が含まれる必要があります。

## 5.7.2. セキュリティーコンテンツの更新

セキュリティーコンテンツは、**ProfileBundle** オブジェクトが参照するコンテナイメージとして含まれます。**ProfileBundles** や、ルールまたはプロファイルなどのバンドルから解析されたカスタムリソースへの更新を正確に追跡するには、タグの代わりにダイジェストを使用してコンプライアンスコンテンツを持つコンテナイメージを識別します。

```
$ oc -n openshift-compliance get profilebundles rhcos4 -oyaml
```

### 出力例

```

apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
metadata:
  creationTimestamp: "2022-10-19T12:06:30Z"
  finalizers:
  - profilebundle.finalizers.compliance.openshift.io
  generation: 1
  name: rhcos4
  namespace: openshift-compliance
  resourceVersion: "46741"

```

```
uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
spec:
  contentFile: ssg-rhcos4-ds.xml
  contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:900e...
1
status:
  conditions:
  - lastTransitionTime: "2022-10-19T12:07:51Z"
    message: Profile bundle successfully parsed
    reason: Valid
    status: "True"
    type: Ready
  dataStreamStatus: VALID
```

### 1 セキュリティーコンテナイメージ。

それぞれの **ProfileBundle** はデプロイメントでサポートされます。コンプライアンス Operator がコンテナイメージダイジェストが変更されたことを検知すると、デプロイメントは変更を反映し、コンテンツを再び解析するように更新されます。タグの代わりにダイジェストを使用すると、安定した予測可能なプロファイルセットを使用できます。

### 5.7.3. 関連情報

- コンプライアンス Operator はネットワークが制限された環境でサポートされています。詳細は、[ネットワークが制限された環境での Operator Lifecycle Manager の使用](#) を参照してください。

## 5.8. コンプライアンス OPERATOR の調整

コンプライアンス Operator には、追加の設定なしで使用できるプロファイルが同梱されますが、それらは組織のニーズおよび要件を満たすために変更される必要があります。プロファイルを変更するプロセスは、**テラリング** と呼ばれます。

コンプライアンス Operator は、**TailoredProfile** オブジェクトを提供し、プロファイルを調整します。

### 5.8.1. 調整されたプロファイルの新規作成

**TailoredProfile** オブジェクトを使用して、調整されたプロファイルをゼロから作成できます。適切な **タイトル** と **説明** を設定し、**extends** フィールドを空のままにします。このカスタムプロファイルが生成するスキャンのタイプをコンプライアンス Operator に指定します。

- ノードのスキャン: オペレーティングシステムをスキャンします。
- プラットフォームスキャン: OpenShift 設定をスキャンします。

### 手順

**TailoredProfile** オブジェクトに以下のアノテーションを設定します。

+ .例 **new-profile.yaml**

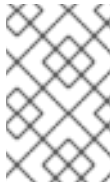
```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
```

```

name: new-profile
annotations:
  compliance.openshift.io/product-type: Node ❶
spec:
  extends:
  description: My custom profile ❷
  title: Custom profile ❸

```

- ❶ それに応じて **Node** または **Platform** を設定します。
- ❷ **description** フィールドを使用して、新しい **TailoredProfile** オブジェクトの機能を記述します。
- ❸ **title** フィールドで、**TailoredProfile** オブジェクトにタイトルを付けます。



### 注記

**TailoredProfile** オブジェクトの **名前** フィールドに **-node** 接尾辞を追加することは、**Node** 製品タイプのアノテーションを追加することと同様であり、オペレーティングシステムスキャンを生成します。

## 5.8.2. 調整されたプロファイルを使用した既存の **ProfileBundles** の拡張

**TailoredProfile** CR は最も一般的なテーラリング操作を有効にする一方で、XCCDF 標準は OpenSCAP プロファイルの調整におけるより多くの柔軟性を提供します。さらに、組織が以前に OpenScap を使用していた場合、既存の XCCDF テーラリングファイルが存在し、これを再利用できる可能性があります。

**ComplianceSuite** オブジェクトには、カスタムのテーラリングファイルにポイントできるオプションの **TailoringConfigMap** 属性が含まれます。**TailoringConfigMap** 属性の値は設定マップの名前です。これには、**tailoring.xml** というキーが含まれる必要があり、このキーの値はテーラリングのコンテンツです。

### 手順

1. Red Hat Enterprise Linux CoreOS (RHCOS) **ProfileBundle** の利用可能なルールを参照します。

```
$ oc get rules.compliance -n openshift-compliance -l compliance.openshift.io/profile-bundle=rhcos4
```

2. 同じ **ProfileBundle** で利用可能な変数を参照します。

```
$ oc get variables.compliance -n openshift-compliance -l compliance.openshift.io/profile-bundle=rhcos4
```

3. **nist-moderate-modified** という名前の調整されたプロファイルを作成します。
  - a. 調整されたプロファイル **nist-moderate-modified** に追加するルールを選択します。この例では、2つのルールを無効にし、1つの値を変更することにより、**rhcos4-moderate** プロファイルを拡張します。**rationale** 値を使用して、これらの変更が加えられた理由を記述します。

### new-profile-node.yaml の例

■

```

apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: nist-moderate-modified
spec:
  extends: rhcos4-moderate
  description: NIST moderate profile
  title: My modified NIST moderate profile
  disableRules:
  - name: rhcos4-file-permissions-var-log-messages
    rationale: The file contains logs of error messages in the system
  - name: rhcos4-account-disable-post-pw-expiration
    rationale: No need to check this as it comes from the IdP
  setValue:
  - name: rhcos4-var-selinux-state
    rationale: Organizational requirements
    value: permissive

```

表5.2 スペック変数の属性

属性	説明
<b>extends</b>	この <b>TailoredProfile</b> がビルドされる <b>Profile</b> オブジェクトの名前。
<b>title</b>	人間が判読できる <b>TailoredProfile</b> のタイトル。
<b>disableRules</b>	名前および理論的根拠のペアの一覧。名前ごとに、無効にするルールオブジェクトの名前を参照します。理論的根拠の値は、人間が判読できるテキストで、ルールが無効になっている理由を記述します。
<b>manualRules</b>	名前および理論的根拠のペアの一覧。手動ルールを追加すると、チェック結果のステータスは常に <b>manual</b> になり、修復は生成されません。この属性は自動であり、手動ルールとして設定されている場合、デフォルトでは値がありません。
<b>enableRules</b>	名前および理論的根拠のペアの一覧。名前ごとに、有効にするルールオブジェクトの名前を参照します。理論的根拠の値は、人間が判読できるテキストで、ルールが有効になっている理由を記述します。
<b>description</b>	<b>TailoredProfile</b> を記述する人間が判読できるテキスト。
<b>setValue</b>	名前、理論的根拠、および値のグループ化の一覧。各名前は値セットの名前を参照します。この理論的根拠は、値セットを記述する、人間が判読できるテキストです。この値は実際の設定です。

- b. **tailoredProfile.spec.manualRules** 属性を追加します。

**tailoredProfile.spec.manualRules.yaml** 例

```

apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: ocp4-manual-scc-check
spec:
  extends: ocp4-cis
  description: This profile extends ocp4-cis by forcing the SCC check to always return
MANUAL
  title: OCP4 CIS profile with manual SCC check
  manualRules:
    - name: ocp4-scc-limit-container-allowed-capabilities
      rationale: We use third party software that installs its own SCC with extra privileges

```

- c. **TailoredProfile** オブジェクトを作成します。

```
$ oc create -n openshift-compliance -f new-profile-node.yaml ❶
```

- ❶ **TailoredProfile** オブジェクトは、デフォルトの **openshift-compliance** namespace で作成されます。

#### 出力例

```
tailoredprofile.compliance.openshift.io/nist-moderate-modified created
```

4. **ScanSettingBinding** オブジェクトを定義して、新しい調整されたプロファイル **nist-moderate-modified** をデフォルトの **ScanSetting** オブジェクトにバインドします。

#### new-scansettingbinding.yaml の例

```

apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: nist-moderate-modified
profiles:
  - apiGroup: compliance.openshift.io/v1alpha1
    kind: Profile
    name: ocp4-moderate
  - apiGroup: compliance.openshift.io/v1alpha1
    kind: TailoredProfile
    name: nist-moderate-modified
settingsRef:
  apiGroup: compliance.openshift.io/v1alpha1
  kind: ScanSetting
  name: default

```

5. **ScanSettingBinding** オブジェクトを作成します。

```
$ oc create -n openshift-compliance -f new-scansettingbinding.yaml
```

#### 出力例

```
scansettingbinding.compliance.openshift.io/nist-moderate-modified created
```

## 5.9. コンプライアンス OPERATOR の未加工の結果の取得

OpenShift Container Platform クラスターのコンプライアンスを証明する際に、監査目的でスキャンの結果を提供する必要がある場合があります。

### 5.9.1. 永続ボリュームからのコンプライアンス Operator の未加工の結果の取得

#### 手順

コンプライアンス Operator は、未加工の結果を生成し、これを永続ボリュームに保存します。これらの結果は Asset Reporting Format (ARF) で生成されます。

1. **ComplianceSuite** オブジェクトを確認します。

```
$ oc get compliancesuites nist-moderate-modified \
-o json -n openshift-compliance | jq '.status.scanStatuses[].resultsStorage'
```

#### 出力例

```
{
  "name": "ocp4-moderate",
  "namespace": "openshift-compliance"
}
{
  "name": "nist-moderate-modified-master",
  "namespace": "openshift-compliance"
}
{
  "name": "nist-moderate-modified-worker",
  "namespace": "openshift-compliance"
}
```

これは、未加工の結果にアクセスできる永続ボリューム要求 (PVC) を表示します。

2. 結果のいずれかの名前と namespace を使用して、未加工データの場所を確認します。

```
$ oc get pvc -n openshift-compliance rhcos4-moderate-worker
```

#### 出力例

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES
rhcos4-moderate-worker	Bound	pvc-548f6cfe-164b-42fe-ba13-a07cfbc77f3a	1Gi	RWO
gp2	92m			

3. ボリュームをマウントする Pod を起動し、結果をコピーして、未加工の結果をフェッチします。

```
$ oc create -n openshift-compliance -f pod.yaml
```

#### pod.yaml の例

```
apiVersion: "v1"
```



```

kind: Pod
metadata:
  name: pv-extract
spec:
  containers:
  - name: pv-extract-pod
    image: registry.access.redhat.com/ubi8/ubi
    command: ["sleep", "3000"]
    volumeMounts:
    - mountPath: "/workers-scan-results"
      name: workers-scan-vol
  volumes:
  - name: workers-scan-vol
    persistentVolumeClaim:
      claimName: rhcos4-moderate-worker

```

4. Pod の実行後に、結果をダウンロードします。

```
$ oc cp pv-extract:/workers-scan-results -n openshift-compliance .
```



### 重要

永続ボリュームをマウントする Pod を起動すると、要求は **Bound** として保持されます。使用中のボリュームのストレージクラスのパーミッションが **ReadWriteOnce** に設定されている場合、ボリュームは一度に1つの Pod によってのみマウント可能です。完了したら Pod を削除する必要があります。そうしないと、Operator は Pod をスケジュールし、継続して結果をこの場所に保存し続けることができなくなります。

5. 展開が完了した後に、Pod を削除できます。

```
$ oc delete pod pv-extract -n openshift-compliance
```

## 5.10. コンプライアンス OPERATOR の結果と修復の管理

それぞれの **ComplianceCheckResult** は、1つのコンプライアンスルールチェックの結果を表します。ルールを自動的に修復できる場合、**ComplianceCheckResult** によって所有される、同じ名前を持つ **ComplianceRemediation** オブジェクトが作成されます。修復が要求されない限り、修復は自動的に適用されません。これにより、OpenShift Container Platform の管理者は修復内容を確認し、検証後のみ修復を適用することができます。

### 5.10.1. コンプライアンスチェック結果のフィルター

デフォルトで、**ComplianceCheckResult** オブジェクトには、チェックのクエリーおよび結果の生成後に次のステップを決定することを可能にする便利なラベルが複数付けられます。

特定のスイートに属するチェックを一覧表示します。

```
$ oc get -n openshift-compliance compliancecheckresults \
-l compliance.openshift.io/suite=workers-compliancesuite
```

特定のスキャンに属するチェックを一覧表示します。

```
$ oc get -n openshift-compliance compliancecheckresults \
-l compliance.openshift.io/scan=workers-scan
```

すべての **ComplianceCheckResult** オブジェクトが **ComplianceRemediation** オブジェクトを作成する訳ではありません。自動的に修復できる **ComplianceCheckResult** オブジェクトのみになります。 **ComplianceCheckResult** オブジェクトには、 **compliance.openshift.io/automated-remediation** ラベルでラベル付けされる場合に関連する修復が含まれます。修復の名前はチェックの名前と同じです。

自動的に修復できる障害のあるチェックをすべて一覧表示します。

```
$ oc get -n openshift-compliance compliancecheckresults \
-l 'compliance.openshift.io/check-status=FAIL,compliance.openshift.io/automated-remediation'
```

失敗したすべてのチェックを重大度順に一覧表示します。

```
$ oc get compliancecheckresults -n openshift-compliance \
-l 'compliance.openshift.io/check-status=FAIL,compliance.openshift.io/check-severity=high'
```

## 出力例

NAME	STATUS	SEVERITY
nist-moderate-modified-master-configure-crypto-policy	FAIL	high
nist-moderate-modified-master-coreos-pti-kernel-argument	FAIL	high
nist-moderate-modified-master-disable-ctrlaltdel-burstaction	FAIL	high
nist-moderate-modified-master-disable-ctrlaltdel-reboot	FAIL	high
nist-moderate-modified-master-enable-fips-mode	FAIL	high
nist-moderate-modified-master-no-empty-passwords	FAIL	high
nist-moderate-modified-master-selinux-state	FAIL	high
nist-moderate-modified-worker-configure-crypto-policy	FAIL	high
nist-moderate-modified-worker-coreos-pti-kernel-argument	FAIL	high
nist-moderate-modified-worker-disable-ctrlaltdel-burstaction	FAIL	high
nist-moderate-modified-worker-disable-ctrlaltdel-reboot	FAIL	high
nist-moderate-modified-worker-enable-fips-mode	FAIL	high
nist-moderate-modified-worker-no-empty-passwords	FAIL	high
nist-moderate-modified-worker-selinux-state	FAIL	high
ocp4-moderate-configure-network-policies-namespaces	FAIL	high
ocp4-moderate-fips-mode-enabled-on-all-nodes	FAIL	high

手動で修復する必要のある障害のあるチェックをすべて一覧表示します。

```
$ oc get -n openshift-compliance compliancecheckresults \
-l 'compliance.openshift.io/check-status=FAIL,!compliance.openshift.io/automated-remediation'
```

手動による修復の手順は、通常 **ComplianceCheckResult** オブジェクトの **description** 属性に保存されます。

表5.3 ComplianceCheckResult Status

ComplianceCheckResult Status	説明
PASS	コンプライアンスチェックが完了し、パシしました。

ComplianceCheckResult Status	説明
FAIL	コンプライアンスチェックが完了するまで実行され、失敗しました。
INFO	コンプライアンスチェックが完了するまで実行され、エラーと見なされるほど深刻ではないものが見つかりました。
MANUAL	コンプライアンスチェックには、成功または失敗を自動的に評価する方法がないため、手動でチェックする必要があります。
INCONSISTENT	コンプライアンスチェックは、さまざまなソース (通常はクラスターノード) からのさまざまな結果を報告します。
ERROR	コンプライアンスチェックは実行されましたが、正しく完了できませんでした。
NOT-APPLICABLE	該当しない、または選択されていないため、コンプライアンスチェックは実行されませんでした。

### 5.10.2. 修復の確認

**ComplianceRemediation** オブジェクト、および修復を所有する **ComplianceCheckResult** オブジェクトの両方を確認します。**ComplianceCheckResult** オブジェクトには、チェック内容やサーバーの強化措置などの人間が判読できる記述、および重大度や関連するセキュリティーコントロールなどの他の **metadata** が含まれます。**ComplianceRemediation** オブジェクトは、**ComplianceCheckResult** に説明されている問題を修正する方法を表します。最初のスキャン後、**MissingDependencies** 状態の修復を確認します。

以下は、**sysctl-net-ipv4-conf-all-accept-redirects** というチェックおよび修復の例です。この例では、**spec** および **status** のみを表示し、**metadata** は省略するように編集されています。

```
spec:
  apply: false
  current:
  object:
    apiVersion: machineconfiguration.openshift.io/v1
    kind: MachineConfig
  spec:
    config:
      ignition:
        version: 3.2.0
    storage:
      files:
        - path: /etc/sysctl.d/75-sysctl_net_ipv4_conf_all_accept_redirects.conf
          mode: 0644
          contents:
            source: data:,.net.ipv4.conf.all.accept_redirects%3D0
```

```

outdated: {}
status:
  applicationState: NotApplied

```

修復ペイロードは **spec.current** 属性に保存されます。ペイロードには Kubernetes オブジェクトを使用することができますが、この修復はノードスキャンによって生成されたため、上記の例の修復ペイロードは **MachineConfig** オブジェクトになります。Platform スキャンでは、修復ペイロードは多くの場合 (**ConfigMap** や **Secret** オブジェクトなど) 異なる種類のオブジェクトになりますが、通常は修復を適用については管理者が判断します。そうでない場合には、コンプライアンス Operator には汎用の Kubernetes オブジェクトを操作するために非常に幅広いパーミッションのセットが必要になるためです。Platform チェックの修復例は、後ほど提供されます。

修復が適用される際に修復内容を正確に確認できるようにするために、**MachineConfig** オブジェクトの内容は設定の Ignition オブジェクトを使用します。形式についての詳細は、[Ignition 仕様](#) を参照してください。この例では、**spec.config.storage.files[0].path** 属性は、この修復によって作成されるファイル (`/etc/sysctl.d/75-sysctl_net_ipv4_conf_all_accept_redirects.conf`) を指定し、**spec.config.storage.files[0].contents.source** 属性はそのファイルの内容を指定します。



#### 注記

ファイルの内容は URL でエンコードされます。

以下の Python スクリプトを使用して、コンテンツを表示します。

```

$ echo "net.ipv4.conf.all.accept_redirects%3D0" | python3 -c "import sys, urllib.parse;
print(urllib.parse.unquote(''.join(sys.stdin.readlines())))"

```

#### 出力例

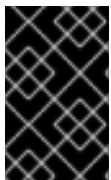
```

net.ipv4.conf.all.accept_redirects=0

```

### 5.10.3. カスタマイズされたマシン設定プールを使用するときに修復を適用する

カスタム **MachineConfigPool** を作成するときは、**MachineConfigPool** にラベルを追加して、**KubeletConfig** に存在する **machineConfigPoolSelector** がそのラベルを **MachineConfigPool** と一致させることができるようにします。



#### 重要

コンプライアンス Operator が修復の適用を終了した後に、**MachineConfigPool** オブジェクトが予期せず一時停止を解除できない可能性があるため、**KubeletConfig** ファイルでは、**protectKernelDefaults:false** を設定しないでください。

#### 手順

1. ノードを一覧表示します。

```

$ oc get nodes -n openshift-compliance

```

#### 出力例

NAME	STATUS	ROLES	AGE	VERSION
------	--------	-------	-----	---------

```
ip-10-0-128-92.us-east-2.compute.internal Ready master 5h21m v1.23.3+d99c04f
ip-10-0-158-32.us-east-2.compute.internal Ready worker 5h17m v1.23.3+d99c04f
ip-10-0-166-81.us-east-2.compute.internal Ready worker 5h17m v1.23.3+d99c04f
ip-10-0-171-170.us-east-2.compute.internal Ready master 5h21m v1.23.3+d99c04f
ip-10-0-197-35.us-east-2.compute.internal Ready master 5h22m v1.23.3+d99c04f
```

2. ノードにラベルを追加します。

```
$ oc -n openshift-compliance \
label node ip-10-0-166-81.us-east-2.compute.internal \
node-role.kubernetes.io/<machine_config_pool_name>=
```

### 出力例

```
node/ip-10-0-166-81.us-east-2.compute.internal labeled
```

3. カスタム **MachineConfigPool** CR を作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: <machine_config_pool_name>
  labels:
    pools.operator.machineconfiguration.openshift.io/<machine_config_pool_name>: " 1"
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,
<machine_config_pool_name>]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/<machine_config_pool_name>: ""
```

- 1 **labels** フィールドは、マシン設定プール (MCP) に追加するラベル名を定義します。

4. MCP が正常に作成されたことを確認します。

```
$ oc get mcp -w
```

#### 5.10.4. デフォルトの設定値をもとにした KubeletConfig ルールの評価

OpenShift Container Platform インフラストラクチャーには、実行時に不完全な設定ファイルが含まれる場合があります。ノードは、設定オプションが欠落している場合には、設定値がデフォルトであることを想定します。一部の設定オプションは、コマンドライン引数として渡すことができます。その結果、コンプライアンス Operator はノード上の設定ファイルが完全かどうかを確認できません。これは、ルールチェックで使用されるオプションが欠落している可能性があるためです。

デフォルトの設定値がチェックに合格するといったフォールスネガティブの結果に陥らないように、コンプライアンス Operator はノード/プロキシ API を使用してノードのプール内にあるノードごとに設定をフェッチし、次にノードプール内のノード全体で整合性が取れた設定オプションすべてをファイルに保存することで、このファイルが対象ノードプール内の全ノードの設定を表します。これにより、スキャン結果の精度が向上します。

デフォルトの **マスター** および **ワーカー** ノードプール設定でこの機能を使用するために、追加の設定変更は必要ありません。

### 5.10.5. カスタムノードプールのスキャン

コンプライアンス Operator は、各ノードプール設定のコピーを維持しません。コンプライアンス Operator は、単一ノードプール内のすべてのノードについて一貫性のある設定オプションを設定ファイルの1つのコピーに集約します。次に、コンプライアンス Operator は、特定のノードプールの設定ファイルを使用して、そのプール内のノードに対してルールを評価します。

クラスターがデフォルトの **ワーカー** ノードおよび **マスター** ノードプール外のカスタムノードプールを使用する場合、コンプライアンス Operator がそのノードプールの設定ファイルを集約できるように追加の変数を指定する必要があります。

#### 手順

1. **master**、**worker**、およびカスタムの **example** ノードプールを含むサンプルクラスター内のすべてのプールに対して設定をチェックするには、**TailoredProfile** オブジェクトの **ocp-var-role-master** および **ocp-var-role-worker** フィールドの値を **example** に設定します。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: cis-example-tp
spec:
  extends: ocp4-cis
  title: My modified NIST profile to scan example nodes
  setValues:
    - name: ocp4-var-role-master
      value: example
      rationale: test for example nodes
    - name: ocp4-var-role-worker
      value: example
      rationale: test for example nodes
  description: cis-example-scan
```

2. **ScanSettingBinding** CR に保存される **ScanSetting** オブジェクトに **example** ロールを追加します。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  rotation: 3
  size: 1Gi
roles:
  - worker
  - master
  - example
scanTolerations:
  - effect: NoSchedule
```

```
key: node-role.kubernetes.io/master
operator: Exists
schedule: '0 1 * * *'
```

3. **ScanSettingBinding** CR を使用するスキャンを作成します。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: cis
  namespace: openshift-compliance
profiles:
- apiGroup: compliance.openshift.io/v1alpha1
  kind: Profile
  name: ocp4-cis
- apiGroup: compliance.openshift.io/v1alpha1
  kind: Profile
  name: ocp4-cis-node
- apiGroup: compliance.openshift.io/v1alpha1
  kind: TailoredProfile
  name: cis-example-tp
settingsRef:
  apiGroup: compliance.openshift.io/v1alpha1
  kind: ScanSetting
  name: default
```

コンプライアンス Operator は **Node/Proxy** API オブジェクトを使用してランタイム **KubeletConfig** をチェックし、**ocp-var-role-master** および **ocp-var-role-worker** などの変数を使用してチェックを実行するノードを判別します。**ComplianceCheckResult** では、**KubeletConfig** ルールは **ocp4-cis-kubelet-\*** と表示されます。スキャンは、選択したすべてのノードがこのチェックに合格した場合のみ、合格します。

### 検証

- Platform KubeletConfig ルールは、**Node/Proxy** オブジェクトを介してチェックされます。これらのルールは、以下のコマンドを実行して検索できます。

```
$ oc get rules -o json | jq '.items[] | select(.checkType == "Platform") | select(.metadata.name | contains("ocp4-kubelet-")) | .metadata.name'
```

### 5.10.6. KubeletConfig サブプールの修復

**KubeletConfig** 修復ラベルは **MachineConfigPool** サブプールに適用できます。

#### 手順

- ラベルをサブプール **MachineConfigPool** CR に追加します。

```
$ oc label mcp <sub-pool-name> pools.operator.machineconfiguration.openshift.io/<sub-pool-name>=>
```

### 5.10.7. 修復の適用



プール値の属性 **spec.apply** は、コンプライアンス Operator で修復を適用するかどうかを制御します。属性を **true** に設定すると、修復を適用することができます。

```
$ oc -n openshift-compliance \
  patch complianceremediations/<scan-name>-sysctl-net-ipv4-conf-all-accept-redirects \
  --patch '{"spec":{"apply":true}}' --type=merge
```

コンプライアンス Operator が適用された修復を処理した後に、**status.ApplicationState** 属性は **Applied** に、または正しくない場合には **Error** に切り替わります。マシン設定の修復が適用されると、その修復と他のすべての適用済みの修復が **75-\$scan-name-\$suite-name** という名前の **MachineConfig** オブジェクトにレンダリングされます。**MachineConfig** オブジェクトはその後 Machine Config Operator によってレンダリングされ、最終的に各ノードで実行されるマシン制御デーモンのインスタンスによってマシン設定プールのすべてのノードに適用されます。

Machine Config Operator が新規 **MachineConfig** オブジェクトをプール内のノードに適用すると、そのプールに属するすべてのノードが再起動されることに注意してください。これは、それぞれが複合の **75-\$scan-name-\$suite-name MachineConfig** オブジェクトを再度レンダリングする、複数の修復を適用する際に不都合が生じる可能性があります。修復をすぐに適用しないようにするには、**MachineConfigPool** オブジェクトの **.spec.paused** 属性を **true** に設定してマシン設定プールを一時停止できます。

コンプライアンス Operator は修復を自動的に適用できます。**ScanSetting** の最上位のオブジェクトに **autoApplyRemediations: true** を設定します。



#### 警告

修復の自動敵に適用するかどうかについては、慎重に考慮する必要があります。

### 5.10.8. プラットフォームチェックの手動による修復

通常、プラットフォームスキャンをチェックする場合、以下の2つの理由のために管理者が手動で修復する必要があります。

- 設定する必要がある値は、常に自動的に判別できる訳ではありません。チェックのいずれかで許可されたレジストリーの一覧が指定されることが必要になりますが、スキャナー側が組織が許可する必要があるレジストリーを認識する方法はありません。
- 異なるチェックで異なる API オブジェクトが変更されるため、クラスター内のオブジェクトを変更するために自動化された修復で **root** またはスーパーユーザーアクセスを所有することが要求されますが、これは推奨されていません。

#### 手順

1. 以下の例では、**ocp4-ocp-allowed-registries-for-import** ルールを使用します。これはデフォルトの OpenShift Container Platform のインストール時に失敗します。ルール **oc get rule.compliance/ocp4-ocp-allowed-registries-for-import -oyaml** を検査します。このルールは、**allowedRegistriesForImport** 属性を設定して、ユーザーのイメージのインポートを許可するレジストリーを制限します。さらに、ルールの **warning** 属性はチェックされた API オブジェクトを表示するため、これを変更し、問題を修復できます。



```
$ oc edit image.config.openshift.io/cluster
```

## 出力例

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
    creationTimestamp: "2020-09-10T10:12:54Z"
  generation: 2
  name: cluster
  resourceVersion: "363096"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: 2dcb614e-2f8a-4a23-ba9a-8e33cd0ff77e
spec:
  allowedRegistriesForImport:
    - domainName: registry.redhat.io
status:
  externalRegistryHostnames:
    - default-route-openshift-image-registry.apps.user-cluster-09-10-12-07.devcluster.openshift.com
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

2. スキャンを再実行します。

```
$ oc -n openshift-compliance \
  annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

### 5.10.9. 修復の更新

新しいバージョンのコンプライアンスコンテンツが使用されると、以前のバージョンとは異なる新しいバージョンの修復が提供される可能性があります。コンプライアンス Operator は、適用される修復の以前のバージョンを保持します。OpenShift Container Platform の管理者は、確認し、適用する新規バージョンについても通知されます。以前に適用されたものの、更新された ComplianceRemediation オブジェクトはそのステータスを **Outdated** に変更します。古いオブジェクトには簡単に検索できるようにラベルが付けられます。

以前に適用された修復内容は **ComplianceRemediation** オブジェクトの **spec.outdated** 属性に保存され、新規に更新された内容は **spec.current** 属性に保存されます。コンテンツを新しいバージョンに更新した後に、管理者は修復を確認する必要があります。**spec.outdated** 属性が存在する限り、これは結果として作成される **MachineConfig** オブジェクトをレンダリングするために使用されます。**spec.outdated** 属性が削除された後に、コンプライアンス Operator は結果として生成される **MachineConfig** オブジェクトを再度レンダリングし、これにより Operator は設定をノードにプッシュします。

#### 手順

1. 古い修復について検索します。

```
$ oc -n openshift-compliance get complianceremediations \
  -l complianceoperator.openshift.io/outdated-remediation=
```

## 出力例

```
NAME                               STATE
workers-scan-no-empty-passwords  Outdated
```

現在適用されている修復は **Outdated** 属性に保存され、新規の、適用されていない修復は **Current** 属性に保存されます。新規バージョンに問題がなければ、**Outdated** フィールドを削除します。更新された内容を保持する必要がある場合には、**Current** および **Outdated** 属性を削除します。

- 修復の新しいバージョンを適用します。

```
$ oc -n openshift-compliance patch complianceremediations workers-scan-no-empty-
passwords \
--type json -p '{"op":"remove", "path":"/spec/outdated}'
```

- 修復の状態は、**Outdated** から **Applied** に切り替わります。

```
$ oc get -n openshift-compliance complianceremediations workers-scan-no-empty-
passwords
```

## 出力例

```
NAME                               STATE
workers-scan-no-empty-passwords  Applied
```

- ノードは新規の修復バージョンを適用し、再起動します。

### 5.10.10. 修復の適用解除

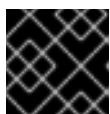
以前に適用された修復を適用解除することが必要になる場合があります。

#### 手順

- apply** フラグを **false** に設定します。

```
$ oc -n openshift-compliance \
patch complianceremediations/rhcos4-moderate-worker-sysctl-net-ipv4-conf-all-accept-
redirects \
--patch '{"spec":{"apply":false}}' --type=merge
```

- 修復ステータスは **NotApplied** に変更され、複合の **MachineConfig** オブジェクトは修復を含まないように再度レンダリングされます。



#### 重要

修復を含む影響を受けるすべてのノードが再起動されます。

### 5.10.11. KubeletConfig 修復の削除

**KubeletConfig** の修正は、ノードレベルのプロファイルに含まれています。KubeletConfig 修復を削除するには、**KubeletConfig** オブジェクトから手動で削除する必要があります。この例は、**one-rule-tp-**

**node-master-kubelet-eviction-thresholds-set-hard-imagefs-available** 修正のコンプライアンスチェックを削除する方法を示しています。

## 手順

1. **one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available** 修復の **scan-name** およびコンプライアンスチェックを見つけます。

```
$ oc -n openshift-compliance get remediation \ one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available -o yaml
```

## 出力例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceRemediation
metadata:
  annotations:
    compliance.openshift.io/xccdf-value-used: var-kubelet-evictionhard-imagefs-available
  creationTimestamp: "2022-01-05T19:52:27Z"
  generation: 1
  labels:
    compliance.openshift.io/scan-name: one-rule-tp-node-master 1
    compliance.openshift.io/suite: one-rule-ssb-node
  name: one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ComplianceCheckResult
    name: one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available
    uid: fe8e1577-9060-4c59-95b2-3e2c51709adc
  resourceVersion: "84820"
  uid: 5339d21a-24d7-40cb-84d2-7a2ebb015355
spec:
  apply: true
  current:
    object:
      apiVersion: machineconfiguration.openshift.io/v1
      kind: KubeletConfig
      spec:
        kubeletConfig:
          evictionHard:
            imagefs.available: 10% 2
  outdated: {}
  type: Configuration
status:
  applicationState: Applied
```

**1** 修復のスキャン名。

**2** **KubeletConfig** オブジェクトに追加された修復。



### 注記

修復によって **evictionHard** kubelet 設定が呼び出される場合は、すべての **evictionHard** パラメーター (**memory.available**、**nodefs.available**、**nodefs.inodesFree**、**imagefs.available**、および **imagefs.inodesFree**) を指定する必要があります。すべてのパラメーターを指定しないと、指定したパラメーターのみが適用され、修正が正しく機能しません。

#### 2. 修復を削除します。

- a. 修復オブジェクトの **apply** を **false** に設定します。

```
$ oc -n openshift-compliance patch \
  complianceremediations/one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-
  imagefs-available \
  -p '{"spec":{"apply":false}}' --type=merge
```

- b. **scan-name** を使用して、修復が適用された **KubeletConfig** オブジェクトを見つけます。

```
$ oc -n openshift-compliance get kubeletconfig \
  --selector compliance.openshift.io/scan-name=one-rule-tp-node-master
```

#### 出力例

```
NAME                                AGE
compliance-operator-kubelet-master  2m34s
```

- c. **KubeletConfig** オブジェクトから修復 **imagefs.available: 10%** を手動で削除します。

```
$ oc edit -n openshift-compliance KubeletConfig compliance-operator-kubelet-master
```



### 重要

修復を含む影響を受けるすべてのノードが再起動されます。



### 注記

また、修正を自動適用する調整済みプロファイルのスケジュールされたスキャンからルールを除外する必要があります。除外しない場合、修正は次のスケジュールされたスキャン中に再適用されます。

## 5.10.12. 一貫性のない ComplianceScan

**ScanSetting** オブジェクトは、**ScanSetting** または **ScanSettingBinding** オブジェクトから生成されるコンプライアンススキャンがスキャンするノードロールを一覧表示します。通常、各ノードのロールはマシン設定プールにマップされます。



### 重要

マシン設定プールのすべてのマシンが同一であり、プールのノードからのすべてのスキャン結果が同一であることが予想されます。

一部の結果が他とは異なる場合、コンプライアンス Operator は一部のノードが **INCONSISTENT** として報告される **ComplianceCheckResult** オブジェクトにフラグを付けます。すべての **ComplianceCheckResult** オブジェクトには、**compliance.openshift.io/inconsistent-check** のラベルも付けられます。

プール内のマシン数は非常に大きくなる可能性があるため、コンプライアンス Operator は最も一般的な状態を検索し、一般的な状態とは異なるノードを一覧表示しようとします。最も一般的な状態は **compliance.openshift.io/most-common-status** アノテーションに保存され、アノテーション **compliance.openshift.io/inconsistent-source** には、最も一般的なステータスとは異なるチェックステータスの **hostname:status** のペアが含まれます。共通状態が見つからない場合、すべての **hostname:status** ペアが **compliance.openshift.io/inconsistent-source** アノテーションに一覧表示されます。

可能な場合は、修復が依然として作成され、クラスターが準拠したステータスに収束できるようにします。ただし、これが常に実行可能な訳ではなく、ノード間の差異の修正は手作業で行う必要があります。スキャンに **compliance.openshift.io/rescan=** オプションのアノテーションを付けて一貫性のある結果を取得するために、コンプライアンススキャンを再実行する必要があります。

```
$ oc -n openshift-compliance \
  annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

### 5.10.13. 関連情報

- [Modifying nodes](#)

## 5.11. 高度なコンプライアンス OPERATOR タスクの実行

コンプライアンス Operator には、デバッグや既存のツールとの統合を目的とする上級ユーザー向けのオプションが含まれます。

### 5.11.1. ComplianceSuite オブジェクトおよび ComplianceScan オブジェクトの直接使用

ユーザーは **ScanSetting** および **ScanSettingBinding** オブジェクトを利用してスイートおよびスキャンを定義することが推奨されますが、**ComplianceSuite** オブジェクトを直接定義する有効なユースケースもあります。

- スキャンする単一ルールのみを指定する。これは、デバッグモードが非常に詳細な情報を取得する可能性があるため、**debug: true** 属性を使用してデバッグする際に役立ちます。これにより、OpenSCAP スキャナーの詳細度が上がります。テストを1つのルールに制限すると、デバッグ情報の量を減らすことができます。
- カスタム `nodeSelector` を指定する。修復を適用可能にするには、`nodeSelector` はプールと一致する必要があります。
- テーラリングファイルでスキャンをカスタム設定マップをポイントする。
- テストまたは開発目的の場合で、バンドルからのプロファイルの解析に伴うオーバーヘッドが不要な場合。

以下の例は、単一のルールのみでワーカーマシンをスキャンする **ComplianceSuite** を示しています。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
```

```

name: workers-compliancesuite
spec:
  scans:
    - name: workers-scan
      profile: xccdf_org.ssgproject.content_profile_moderate
      content: ssg-rhcos4-ds.xml
      contentImage: quay.io/complianceascode/ocp4:latest
      debug: true
      rule: xccdf_org.ssgproject.content_rule_no_direct_root_logins
      nodeSelector:
        node-role.kubernetes.io/worker: ""

```

上記の **ComplianceSuite** オブジェクトおよび **ComplianceScan** オブジェクトは、OpenSCAP が想定する形式で複数の属性を指定します。

プロファイル、コンテンツ、またはルールを見つけるには、最初に **ScanSetting** および **ScanSettingBinding** から同様の Suite を作成して開始するか、またはルールやプロファイルなどの **ProfileBundle** オブジェクトから解析されたオブジェクトを検査することができます。これらのオブジェクトには、**ComplianceSuite** から参照するために使用できる **xccdf\_org** 識別子が含まれます。

### 5.11.2. ScanSetting スキャンの PriorityClass の設定

大規模な環境では、デフォルトの **PriorityClass** オブジェクトの優先順位が低すぎて、Pod が時間どおりにスキャンを実行することを保証できない場合があります。コンプライアンスを維持するか、自動スキャンを保証する必要があるクラスターの場合、**PriorityClass** 変数を設定して、コンプライアンス Operator がリソースの制約の状況で常に優先順位が指定されるようにします。

#### 手順

- **PriorityClass** 変数を設定します。

```

apiVersion: compliance.openshift.io/v1alpha1
strictNodeScan: true
metadata:
  name: default
  namespace: openshift-compliance
priorityClass: compliance-high-priority 1
kind: ScanSetting
showNotApplicable: false
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/master: ""
pvAccessModes:
  - ReadWriteOnce
rotation: 3
size: 1Gi
tolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
    operator: Exists
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute

```

```

key: node.kubernetes.io/unreachable
operator: Exists
tolerationSeconds: 300
- effect: NoSchedule
  key: node.kubernetes.io/memory-pressure
  operator: Exists
schedule: 0 1 * * *
roles:
- master
- worker
scanTolerations:
- operator: Exists

```

- 1 **ScanSetting** で参照される **PriorityClass** が見つからない場合、Operator は **PriorityClass** を空のまま警告を発行し、**PriorityClass** なしでスキャンのスケジュールを続行します。

### 5.11.3. 未加工の調整済みプロファイルの使用

**TailoredProfile** CR は最も一般的なテーラリング操作を有効にする一方で、XCCDF 標準は OpenSCAP プロファイルの調整におけるより多くの柔軟性を提供します。さらに、組織が以前に OpenScap を使用していた場合、既存の XCCDF テーラリングファイルが存在し、これを再利用できる可能性があります。

**ComplianceSuite** オブジェクトには、カスタムのテーラリングファイルにポイントできるオプションの **TailoringConfigMap** 属性が含まれます。**TailoringConfigMap** 属性の値は設定マップの名前です。これには、**tailoring.xml** というキーが含まれる必要があり、このキーの値はテーラリングのコンテンツです。

#### 手順

1. ファイルから **ConfigMap** オブジェクトを作成します。

```

$ oc -n openshift-compliance \
create configmap nist-moderate-modified \
--from-file=tailoring.xml=/path/to/the/tailoringFile.xml

```

2. スイートに属するスキャンでテーラリングファイルを参照します。

```

apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
  name: workers-compliancesuite
spec:
  debug: true
  scans:
    - name: workers-scan
      profile: xccdf_org.ssgproject.content_profile_moderate
      content: ssg-rhcos4-ds.xml
      contentImage: quay.io/complianceascode/ocp4:latest
      debug: true
  tailoringConfigMap:

```



```
name: nist-moderate-modified
nodeSelector:
  node-role.kubernetes.io/worker: ""
```

#### 5.11.4. 再スキャンの実行

通常、毎週月曜日または日次など、定義されたスケジュールでスキャンを再実行する必要がある場合があります。また、ノードの問題を修正した後にスキャンを1回再実行することは役に立ちます。単一スキャンを実行するには、スキャンに **compliance.openshift.io/rescan=** オプションでアノテーションを付けます。

```
$ oc -n openshift-compliance \
  annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

再スキャンにより、**rhcos-moderate** プロファイルの4つの追加 **mc** が生成されます。

```
$ oc get mc
```

#### 出力例

```
75-worker-scan-chronyd-or-ntpd-specify-remote-server
75-worker-scan-configure-usbguard-auditbackend
75-worker-scan-service-usbguard-enabled
75-worker-scan-usbguard-allow-hid-and-hub
```



#### 重要

スキャン設定の **default-auto-apply** ラベルが適用されると、修復は自動的に適用され、古い修復が自動的に更新されます。依存関係により適用されなかった修復や、古くなった修復がある場合には、再スキャンにより修復が適用され、再起動がトリガーされる可能性があります。**MachineConfig** オブジェクトを使用する修復のみが再起動をトリガーします。適用する更新または依存関係がない場合は、再起動は発生しません。

#### 5.11.5. 結果についてのカスタムストレージサイズの設定

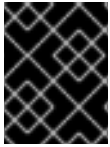
**ComplianceCheckResult** などのカスタムリソースは、スキャンされたすべてのノード間での単一チェックの集計された結果を表しますが、スキャナーによって生成される未加工の結果を確認することが役に立つ場合もあります。未加工の結果は ARF 形式で生成され、サイズが大きくなる可能性があります(ノードあたり数十メガバイト)、それらを **etcd** のキーと値のストアでサポートされる Kubernetes リソースに保存することは実際的ではありません。すべてのスキャンは、1GB のサイズにデフォルト設定される永続ボリューム (PV) を作成します。環境によっては、PV サイズを適宜増やす必要がある場合があります。これは、**ScanSetting** および **ComplianceScan** リソースの両方に公開される **rawResultStorage.size** 属性を使用して実行されます。

関連するパラメーターは **rawResultStorage.rotation** であり、これは古いスキャンのローテーションが行われる前に保持されるスキャンの数を制御します。デフォルト値は3です。ローテーションポリシーを0に設定するとローテーションは無効になります。デフォルトのローテーションポリシーと、未加工の ARF スキャンレポートにつき100 MBを見積もり、お使いの環境に適した PV サイズを計算できます。

##### 5.11.5.1. カスタム結果ストレージ値の使用

OpenShift Container Platform は各種のパブリッククラウドまたはベアメタルにデプロイできるので、

コンプライアンス Operator は利用可能なストレージ設定を判別できません。デフォルトで、コンプライアンス Operator はクラスターのデフォルトのストレージクラスを使用して結果を保存するために PV の作成を試行しますが、カスタムストレージクラスは **rawResultStorage.StorageClassName** 属性を使用して設定できます。



### 重要

クラスターがデフォルトのストレージクラスを指定しない場合、この属性を設定する必要があります。

標準ストレージクラスを使用するように **ScanSetting** カスタムリソースを設定し、サイズが 10GB の永続ボリュームを作成し、最後の 10 件の結果を保持します。

### ScanSetting CR の例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  storageClassName: standard
  rotation: 10
  size: 10Gi
roles:
- worker
- master
scanTolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
schedule: '0 1 * * *
```

#### 5.11.6. スイートスキャンによって生成される修復の適用

**ComplianceSuite** オブジェクトで **autoApplyRemediations** ブール値パラメーターを使用することもできますが、オブジェクトに **compliance.openshift.io/apply-remediations** のアノテーションを付けることもできます。これにより、Operator は作成されるすべての修復を適用できます。

#### 手順

- 以下を実行して、**compliance.openshift.io/apply-remediations** アノテーションを適用します。

```
$ oc -n openshift-compliance \
  annotate compliancesuites/workers-compliancesuite compliance.openshift.io/apply-remediations=
```

#### 5.11.7. 修復の自動更新

新しいコンテンツを含むスキャンは、修復に **OUTDATED** というマークを付ける可能性があります。管理者は、**compliance.openshift.io/remove-outdated** アノテーションを適用して新規の修復を適用し、古い修復を削除できます。

## 手順

- **compliance.openshift.io/remove-outdated** アノテーションを適用します。

```
$ oc -n openshift-compliance \
  annotate compliancesuites/workers-compliancesuite compliance.openshift.io/remove-outdated=
```

または、**ScanSetting** または **ComplianceSuite** オブジェクトに **autoUpdateRemediations** フラグを設定し、修復を自動的に更新します。

### 5.11.8. コンプライアンスオペレーター向けのカスタム SCC の作成

一部の環境では、カスタムのセキュリティーコンテキスト制約 (SCC) ファイルを作成して、コンプライアンスオペレーターの **api-resource-collector** が正しいアクセス許可を利用できるようにする必要があります。

#### 前提条件

- **admin** 権限がある。

#### 手順

1. **restricted-adjusted-compliance.yaml** という名前の YAML ファイルで SCC を定義します。

#### SecurityContextConstraints オブジェクト定義

```
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
kind: SecurityContextConstraints
metadata:
  name: restricted-adjusted-compliance
priority: 30 ①
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- SETUID
- SETGID
- MKNOD
runAsUser:
  type: MustRunAsRange
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users:
```

```
- system:serviceaccount:openshift-compliance:api-resource-collector 2
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

- 1** この SCC の優先度は、**system:authenticated** グループに適用される他のどの SCC よりも高くなければなりません。
- 2** コンプライアンスオペレータースキャナー Pod で使用されるサービスアカウント。

2. SCC を作成します。

```
$ oc create -n openshift-compliance -f restricted-adjusted-compliance.yaml
```

### 出力例

```
securitycontextconstraints.security.openshift.io/restricted-adjusted-compliance created
```

### 検証

1. SCC が作成されたことを確認します。

```
$ oc get -n openshift-compliance scc restricted-adjusted-compliance
```

### 出力例

```
NAME                PRIV CAPS      SELINUX     RUNASUSER      FSGROUP
SUPGROUP PRIORITY READONLYROOTFS VOLUMES
restricted-adjusted-compliance false <no value> MustRunAs MustRunAsRange
MustRunAs RunAsAny 30 false
["configMap","downwardAPI","emptyDir","persistentVolumeClaim","projected","secret"]
```

### 5.11.9. 関連情報

- [Security Context Constraints の管理](#)

## 5.12. コンプライアンス OPERATOR のトラブルシューティング

このセクションでは、コンプライアンス Operator のトラブルシューティングの方法について説明します。この情報は、問題を診断したり、バグレポートに情報を提供したりする際に役立ちます。一般的なヒントには、以下が含まれます。

- コンプライアンス Operator は、重要なことが発生すると Kubernetes イベントを生成します。コマンドを使用して、クラスター内のすべてのイベントを表示できます。

```
$ oc get events -n openshift-compliance
```

または、コマンドを使用してスキャンなどのオブジェクトのイベントを表示します。

```
$ oc describe -n openshift-compliance compliancescan/cis-compliance
```

- コンプライアンス Operator は複数のコントローラーで設定されており、API オブジェクトごとに約1つのコントローラーで設定されます。問題のある API オブジェクトに対応するコントローラーのみをフィルターすることが役に立つ場合があります。**ComplianceRemediation** を適用できない場合、**remediationctrl** コントローラーのメッセージを表示します。**jq** で解析することにより、単一のコントローラーからのメッセージをフィルターできます。

```
$ oc -n openshift-compliance logs compliance-operator-775d7bddbd-gj58f \
| jq -c 'select(.logger == "profilebundlectrl")'
```

- タイムスタンプについては、UTC の UNIX エポックからの経過時間で秒単位でログに記録されます。人間が判読可能な日付に変換するには、**date -d @timestamp --utc** を使用します。以下は例になります。

```
$ date -d @1596184628.955853 --utc
```

- 数多くのカスタムリソースで、最も重要な **ComplianceSuite** および **ScanSetting** で **debug** オプションを設定できます。このオプションを有効にすると、OpenSCAP スキャナー Pod や他のヘルパー Pod の詳細度が上がります。
- 単一ルールの場合が予期せずに出される場合、そのルールのみを使用して単一スキャンまたはスイートを実行し、対応する **ComplianceCheckResult** オブジェクトからルール ID を見つけ、それを **Scan** CR の **rule** 属性として使用することが役に立つ場合があります。次に、**debug** オプションが有効になると、スキャナー Pod の **scanner** コンテナログに未加工の OpenSCAP ログが表示されます。

### 5.12.1. スキャンの仕組み

以下のセクションでは、コンプライアンス Operator スキャンのコンポーネントおよびステージについて説明します。

#### 5.12.1.1. コンプライアンスのソース

コンプライアンスコンテンツは、**ProfileBundle** オブジェクトから生成される **Profile** オブジェクトに保存されます。コンプライアンス Operator は、**ProfileBundle** をクラスター用とクラスターノード用に作成します。

```
$ oc get -n openshift-compliance profilebundle.compliance
```

```
$ oc get -n openshift-compliance profile.compliance
```

**ProfileBundle** オブジェクトは、**Bundle** の名前ではラベルが付けられたデプロイメントで処理されます。**Bundle** で問題のトラブルシューティングを行うには、デプロイメントを見つけ、デプロイメントで Pod のログを表示できます。

```
$ oc logs -n openshift-compliance -lprofile-bundle=ocp4 -c profileparser
```

```
$ oc get -n openshift-compliance deployments,pods -lprofile-bundle=ocp4
```

```
$ oc logs -n openshift-compliance pods/<pod-name>
```

```
$ oc describe -n openshift-compliance pod/<pod-name> -c profileparser
```

### 5.12.1.2. ScanSetting および ScanSettingBinding のライフサイクルおよびデバッグ

有効なコンプライアンスコンテンツソースを使用して、高レベルの **ScanSetting** および **ScanSettingBinding** オブジェクトを、**ComplianceSuite** および **ComplianceScan** オブジェクトを生成するために使用できます。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: my-companys-constraints
debug: true
# For each role, a separate scan will be created pointing
# to a node-role specified in roles
roles:
  - worker
---
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: my-companys-compliance-requirements
profiles:
  # Node checks
  - name: rhcos4-e8
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
  # Cluster checks
  - name: ocp4-e8
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
settingsRef:
  name: my-companys-constraints
  kind: ScanSetting
  apiGroup: compliance.openshift.io/v1alpha1
```

**ScanSetting** および **ScanSettingBinding** オブジェクトはどちらも、**logger=scansettingbindingctrl** のタグの付けられた同じコントローラーで処理されます。これらのオブジェクトにはステータスがありません。問題はイベントの形式で通信されます。

```
Events:
  Type    Reason      Age   From          Message
  ----    -
  Normal  SuiteCreated 9m52s scansettingbindingctrl ComplianceSuite openshift-compliance/my-companys-compliance-requirements created
```

今回のリリースにより、**ComplianceSuite** オブジェクトが作成されました。フローは新規に作成された **ComplianceSuite** を継続して調整します。

### 5.12.1.3. ComplianceSuite カスタムリソースのライフサイクルおよびデバッグ

**ComplianceSuite** CR は **ComplianceScan** CR に関連したラッパーです。 **ComplianceSuite** CR は、

**logger=suitedctrl** のタグが付けられたコントローラーによって処理されます。このコントローラーは、スイートからのスキャンの作成を処理し、個別のスキャンのステータスを調整し、これを単一の Suite ステータスに集約します。スイートが定期的に行われるよう設定されている場合、**suitedctrl** は、初回の実行後にスキャンをスイートで再実行する **CronJob** CR の作成にも対応します。

```
$ oc get cronjobs
```

## 出力例

NAME	SCHEDULE	SUSPEND	ACTIVE	LAST SCHEDULE	AGE
<cron_name>	0 1 * * *	False	0	<none>	151m

最も重要な問題について、イベントが生成されます。**oc describe compliancesuites/<name>** を使用してそれらを表示します。**Suite** オブジェクトには、**Status** サブリソースも含まれており、これはこのスイートに属する **Scan** オブジェクトが **Status** サブリソースを更新すると更新されます。予想されるすべてのスキャンが作成されると、コントローラーがスキャンコントローラーに渡されます。

### 5.12.1.4. ComplianceScan カスタムリソースのライフサイクルおよびデバッグ

**ComplianceScan** CR は **scancntrl** コントローラーによって処理されます。これは、実際のスキャンとスキャン結果が作成される場所でもあります。それぞれのスキャンは複数のフェーズを経由します。

#### 5.12.1.4.1. Pending (保留) フェーズ

このフェーズでは、スキャンがその正確性について検証されます。ストレージサイズなどの一部のパラメーターが無効な場合、スキャンは ERROR 結果と共に DONE に移行します。それ以外の場合は、Launching (起動) フェーズに進みます。

#### 5.12.1.4.2. Launching (起動) フェーズ

このフェーズでは、いくつかの設定マップにスキャナー Pod の環境またはスキャナー Pod が評価するスクリプトが直接含まれます。設定マップを一覧表示します。

```
$ oc -n openshift-compliance get cm \
-l compliance.openshift.io/scan-name=rhcos4-e8-worker,complianceoperator.openshift.io/scan-script=
```

これらの設定マップはスキャナー Pod によって使用されます。スキャナーの動作を変更したり、スキャナーのデバッグレベルを変更したり、未加工の結果を出力したりする必要がある場合は、1つの方法として設定マップを変更することができます。その後、未加工の ARF の結果を保存するために永続ボリューム要求 (PVC) がスキャンごとに作成されます。

```
$ oc get pvc -n openshift-compliance -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

PVC はスキャンごとの **ResultServer** デプロイメントでマウントされます。**ResultServer** は、個別のスキャナー Pod が完全な ARF 結果をアップロードする単純な HTTP サーバーです。各サーバーは、異なるノードで実行できます。完全な ARF の結果のサイズは非常に大きくなる可能性があり、複数のノードから同時にマウントできるボリュームを作成できると想定することはできません。スキャンが終了した後に、**ResultServer** デプロイメントはスケールダウンされます。未加工の結果のある PVC は別のカスタム Pod からマウントでき、結果はフェッチしたり、検査したりできます。スキャナー Pod と **ResultServer** 間のトラフィックは相互 TLS プロトコルで保護されます。

最後に、スキャナー Pod はこのフェーズで起動します。**Platform** スキャンインスタンスの1つのス



キャナー Pod と、**node** スキャンインスタンスの一致するノードごとに1つのスキャナー Pod です。ノードごとの Pod にはノード名のラベルが付けられます。それぞれの Pod には、常に **ComplianceScan** という名前のラベルが付けられます。

```
$ oc get pods -lcompliance.openshift.io/scan-name=rhcos4-e8-worker,workload=scanner --show-labels
```

## 出力例

```
NAME                                READY STATUS   RESTARTS AGE LABELS
rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod 0/2   Completed 0    39m
compliance.openshift.io/scan-name=rhcos4-e8-worker,targetNode=ip-10-0-169-90.eu-north-1.compute.internal,workload=scanner
```

+ スキャンは Running (実行) フェーズに進みます。

### 5.12.1.4.3. Running (実行) フェーズ

実行フェーズは、スキャナー Pod の完了後に開始します。以下の用語およびプロセスは実行フェーズで使用されます。

- **init container: content-container** という1つの init コンテナがあります。これは、**contentImage** コンテナを実行し、**contentFile** を、この Pod の他のコンテナで共有される **/content** ディレクトリーにコピーします。
- **scanner**: このコンテナはスキャンを実行します。ノードのスキャンの場合には、コンテナはノードファイルシステムを **/host** としてマウントし、init コンテナによって配信されるコンテンツをマウントします。また、コンテナは Launching (起動) フェーズで作成される **entrypoint ConfigMap** をマウントし、これを実行します。エントリーポイント **ConfigMap** のデフォルトスクリプトは OpenSCAP を実行し、結果ファイルを Pod のコンテナ間で共有される **/results** ディレクトリーに保存します。この Pod のログを表示して、OpenSCAP スキャナーがチェックした内容を判別できます。より詳細な出力は、**debug** フラグを使用して表示できます。
- **logcollector**: logcollector コンテナは、スキャナーコンテナが終了するまで待機します。その後、これは **ConfigMap** として完全な ARF 結果を **ResultServer** にアップロードし、スキャン結果および OpenSCAP 結果コードと共に XCCDF 結果を個別にアップロードします。これらの結果の Config Map には、スキャン名 (**compliance.openshift.io/scan-name=rhcos4-e8-worker**) のラベルが付けられます。

```
$ oc describe cm/rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod
```

## 出力例

```
Name:      rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod
Namespace: openshift-compliance
Labels:    compliance.openshift.io/scan-name-scan=rhcos4-e8-worker
           complianceoperator.openshift.io/scan-result=
Annotations: compliance-remediations/processed:
             compliance.openshift.io/scan-error-msg:
             compliance.openshift.io/scan-result: NON-COMPLIANT
             OpenSCAP-scan-result/node: ip-10-0-169-90.eu-north-1.compute.internal
```

Data

```
====
exit-code:
----
2
results:
----
<?xml version="1.0" encoding="UTF-8"?>
...

```

**Platform** スキャンのスカナー Pod も同様ですが、以下の点で異なります。

- **api-resource-collector** という追加の init コンテナがあります。これは content-container init、コンテナで提供される OpenSCAP コンテンツを読み取り、コンテンツで確認する必要のある API リソースを判別し、それらの API リソースを **scanner** コンテナが読み取りを行う共有ディレクトリに保存します。
- **scanner** コンテナは、ホストファイルシステムをマウントする必要はありません。

スカナー Pod が実行されると、スキャンは Aggregating (集計) フェーズに移行します。

#### 5.12.1.4.4. Aggregating (集計) フェーズ

Aggregating (集計) フェーズでは、スキャンコントローラーがアグリゲーター Pod と呼ばれる別の Pod を起動します。その目的は、結果の **ConfigMap** オブジェクトを取り、結果を読み取り、それぞれのチェックの結果について対応する Kubernetes オブジェクトを作成することにあります。チェックの失敗を自動的に修復できる場合は、**ComplianceRemediation** オブジェクトが作成されます。チェックと修復についての人間が判読できるメタデータを提供するために、アグリゲーター Pod は init コンテナを使用して OpenSCAP コンテンツもマウントします。

設定マップがアグリゲーター Pod によって処理される場合、これには **compliance-remediations/processed** ラベルでラベルが付けられます。このフェーズの結果は **ComplianceCheckResult** オブジェクトになります。

```
$ oc get compliancecheckresults -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

#### 出力例

```
NAME                                STATUS SEVERITY
rhcos4-e8-worker-accounts-no-uid-except-zero    PASS  high
rhcos4-e8-worker-audit-rules-dac-modification-chmod    FAIL  medium
```

**ComplianceRemediation** オブジェクト:

```
$ oc get complianceremediations -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

#### 出力例

```
NAME                                STATE
rhcos4-e8-worker-audit-rules-dac-modification-chmod    NotApplied
rhcos4-e8-worker-audit-rules-dac-modification-chown    NotApplied
rhcos4-e8-worker-audit-rules-execution-chcon          NotApplied
rhcos4-e8-worker-audit-rules-execution-restorecon     NotApplied
rhcos4-e8-worker-audit-rules-execution-semanage       NotApplied
rhcos4-e8-worker-audit-rules-execution-setfiles       NotApplied
```

これらの CR が作成されると、アグリゲーター Pod は終了し、スキャンは Done (終了) フェーズに移行します。

#### 5.12.1.4.5. Done (終了) フェーズ

最終のスキャンフェーズでは、必要な場合にスキャンリソースがクリーンアップされ、**ResultServer** デプロイメントは (スキャンが1回限りの場合) スケールダウンされるか、または (スキャンが継続される場合) 削除されます。次のスキャンインスタンスではデプロイメントを再作成します。

また、Done (終了) フェーズでは、スキャンにアノテーションを付けてスキャンの再実行をトリガーすることもできます。

```
$ oc -n openshift-compliance \
  annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

スキャンが Done (終了) フェーズに到達した後に、修復が **autoApplyRemediations: true** を指定して自動的に適用されるように設定されていない限り、自動的に実行されることは何ともありません。OpenShift Container Platform 管理者は、修復を確認し、必要に応じてそれらを適用できるようになりました。修復が自動的に適用されるように設定されている場合、**ComplianceSuite** コントローラーが Done (終了) フェーズで引き継ぎ、マシン設定プールをスキャンがマップされるポイントで一時停止し、すべての修復を1回で適用します。修復が適用されると、**ComplianceRemediation** コントローラーが引き継ぎます。

#### 5.12.1.5. ComplianceRemediation コントローラーのライフサイクルおよびデバッグ

サンプルスキャンは特定の結果を報告します。修復の1つを有効にするには、**apply** 属性を **true** に切り替えます。

```
$ oc patch complianceremediations/rhcos4-e8-worker-audit-rules-dac-modification-chmod --patch
 '{"spec":{"apply":true}}' --type=merge
```

**ComplianceRemediation** コントローラー (**logger=remediationctrl**) は変更されたオブジェクトを調整します。調整の結果として、調整される修復オブジェクトのステータスが変更されますが、適用されたすべての修復が含まれる、レンダリングされるスイートごとの **MachineConfig** オブジェクトも変更されます。

**MachineConfig** オブジェクトは常に **75-** で開始し、スキャンとスイートに基づいて名前が付けられます。

```
$ oc get mc | grep 75-
```

#### 出力例

```
75-rhcos4-e8-worker-my-companys-compliance-requirements          3.2.0
2m46s
```

**mc** を現在設定している修復はマシン設定のアノテーションに一覧表示されます。

```
$ oc describe mc/75-rhcos4-e8-worker-my-companys-compliance-requirements
```

#### 出力例

```
Name:      75-rhcos4-e8-worker-my-companys-compliance-requirements
Labels:    machineconfiguration.openshift.io/role=worker
Annotations: remediation/rhcos4-e8-worker-audit-rules-dac-modification-chmod:
```

**ComplianceRemediation** コントローラーのアルゴリズムは以下のようになります。

- 現在適用されているすべての修復は初期の修復セットに読み込まれます。
- 調整された修復が適用されることが予想される場合、それはセットに追加されます。
- **MachineConfig** オブジェクトはセットからレンダリングされ、セット内の修復の名前でアノテーションが付けられます。セットが空の場合 (最後の修復は適用されない)、レンダリングされる **MachineConfig** オブジェクトは削除されます。
- レンダリングされたマシン設定がクラスターにすでに適用されているものとは異なる場合にのみ、適用される MC は更新されます (または作成されるか、削除されます)。
- **MachineConfig** オブジェクトの作成または変更により、**machineconfiguration.openshift.io/role** ラベルに一致するノードの再起動がトリガーされます。詳細は、Machine Config Operator のドキュメントを参照してください。

修復ループは、レンダリングされたマシン設定が更新され (必要な場合)、調整された修復オブジェクトのステータスが更新されると終了します。この場合、修復を適用すると再起動がトリガーされます。再起動後、スキャンにアノテーションを付け、再度実行します。

```
$ oc -n openshift-compliance \
  annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

スキャンが実行され、終了します。渡される修復の有無を確認します。

```
$ oc -n openshift-compliance \
  get compliancecheckresults/rhcos4-e8-worker-audit-rules-dac-modification-chmod
```

## 出力例

```
NAME                                STATUS SEVERITY
rhcos4-e8-worker-audit-rules-dac-modification-chmod  PASS  medium
```

### 5.12.1.6. 役に立つラベル

コンプライアンス Operator によって起動する各 Pod には、それが属するスキャンおよびその機能と多くに関連するラベルが付けられます。スキャン ID には **compliance.openshift.io/scan-name** ラベルが付けられます。ワークロード ID には、**workload** ラベルでラベルが付けられます。

コンプライアンス Operator は以下のワークロードをスケジュールします。

- **scanner**: コンプライアンススキャンを実行します。
- **resultserver**: コンプライアンススキャンの未加工の結果を保存します。
- **aggregator**: 結果を集計し、不整合を検出し、結果オブジェクト (チェックの結果と修復) を出力します。
- **suiterunner**: 再実行するスイートにタグを付けます (スケジュールが設定されている場合)。

- **profileparser**: データストリームを解析し、適切なプロファイル、ルールおよび変数を作成します。

デバッグおよびログが特定のワークロードに必要な場合は、以下を実行します。

```
$ oc logs -l workload=<workload_name> -c <container_name>
```

### 5.12.2. コンプライアンス Operator のリソース制限の増加

コンプライアンス Operator は、デフォルトの制限よりもメモリーを多く必要とする場合があります。この問題を軽減する最善の方法は、カスタムリソースの制限を設定することです。

スキャナー Pod のデフォルトのメモリーおよび CPU 制限を増やすには、**ScanSetting カスタムリソース** を参照してください。

#### 手順

1. Operator のメモリー制限を 500 Mi に増やすには、**co-memlimit-patch.yaml** という名前の以下のパッチファイルを作成します。

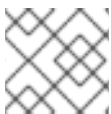
```
spec:
  config:
    resources:
      limits:
        memory: 500Mi
```

2. パッチファイルを適用します。

```
$ oc patch sub compliance-operator -nopenshift-compliance --patch-file co-memlimit-patch.yaml --type=merge
```

### 5.12.3. Operator リソース制約の設定

**resources** フィールドは、Operator Lifecycle Manager (OLM) によって作成される Pod 内のすべてのコンテナの Resource Constraints を定義します。



#### 注記

このプロセスで適用されるリソース制約は、既存のリソース制約を上書きします。

#### 手順

- **Subscription** オブジェクトを編集して、各コンテナに 0.25 cpu と 64 Mi のメモリーの要求と、0.5 cpu と 128 Mi のメモリーの制限を挿入します。

```
kind: Subscription
metadata:
  name: custom-operator
spec:
  package: etcd
  channel: alpha
  config:
    resources:
```

```
requests:
  memory: "64Mi"
  cpu: "250m"
limits:
  memory: "128Mi"
  cpu: "500m"
```

#### 5.12.4. ScanSetting タイムアウトの設定

**ScanSetting** オブジェクトには、**ComplianceScanSetting** オブジェクトで **1h30m** などの期間文字列として指定できるタイムアウトオプションがあります。指定されたタイムアウト内にスキャンが終了しない場合、スキャンは **maxRetryOnTimeout** 制限に達するまで再試行されます。

##### 手順

- ScanSetting で **timeout** と **maxRetryOnTimeout** を設定するには、既存の **ScanSetting** オブジェクトを変更します。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  rotation: 3
  size: 1Gi
roles:
- worker
- master
scanTolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
schedule: '0 1 * * *'
timeout: '10m0s' ①
maxRetryOnTimeout: 3 ②
```

- ① **timeout** 変数は、**1h30m** などの期間文字列として定義されます。デフォルト値は **30m** です。タイムアウトを無効にするには、値を **0s** に設定します。
- ② **maxRetryOnTimeout** 変数は、再試行を試行する回数を定義します。デフォルト値は **3** です。

#### 5.12.5. サポート

本書で説明されている手順、または OpenShift Container Platform で問題が発生した場合は、[Red Hat カスタマーポータル](#) にアクセスしてください。カスタマーポータルでは、次のことができます。

- Red Hat 製品に関するアーティクルおよびソリューションを対象とした Red Hat ナレッジベースの検索またはブラウズ。
- Red Hat サポートに対するサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

クラスターの問題を特定するには、[OpenShift Cluster Manager](#) で Insights を使用できます。Insights により、問題の詳細と、利用可能な場合は問題の解決方法に関する情報が提供されます。

本書の改善への提案がある場合、またはエラーを見つけた場合は、最も関連性の高いドキュメントコンポーネントの [Jira Issue](#) を送信してください。セクション名や OpenShift Container Platform バージョンなどの具体的な情報を提供してください。

## 5.13. コンプライアンス OPERATOR のアンインストール

OpenShift Container Platform Web コンソールまたは CLI を使用して、クラスターから OpenShift コンプライアンス Operator を削除できます。

### 5.13.1. Web コンソールを使用した OpenShift Container Platform からの OpenShift コンプライアンス Operator のアンインストール




コンプライアンス Operator を削除するには、まず namespace のオブジェクトを削除する必要があります。オブジェクトが削除されたら、**openshift-compliance** プロジェクトを削除することで、Operator とその namespace を削除できます。

#### 前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。
- OpenShift コンプライアンス Operator をインストールする必要があります。

#### 手順

OpenShift Container Platform Web コンソールを使用して コンプライアンス Operator を削除するには、以下を行います。

1. **Operators** → **Installed Operators** → **Compliance Operator** ページに移動します。
  - a. **All instances** をクリックします。
  - b. **All namespaces** で、 オプションメニューをクリックし、すべての ScanSettingBinding、ComplianceSuite、ComplianceScan、および ProfileBundle オブジェクトを削除します。
2. **Administration** → **Operators** → **Installed Operators** ページに切り替えます。
3. **Compliance Operator** エントリーのオプションメニュー  をクリックして **Uninstall Operator** を選択します。
4. **Home** → **Projects** ページに切り替えます。
5. コンプライアンスを検索します。
6. **openshift-compliance** プロジェクトの横にある Options メニュー  をクリックし、**Delete Project** を選択します。
  - a. ダイアログボックスに **openshift-compliance** と入力して削除を確認し、**Delete** をクリッ

削除します。

### 5.13.2. CLI を使用した OpenShift Container Platform からの OpenShift コンプライアンス Operator のアンインストール

コンプライアンス Operator を削除するには、まず namespace のオブジェクトを削除する必要があります。オブジェクトが削除されたら、**openshift-compliance** プロジェクトを削除することで、Operator とその namespace を削除できます。

#### 前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスタにアクセスできる。
- OpenShift コンプライアンス Operator をインストールする必要があります。

#### 手順

1. namespace のすべてのオブジェクトを削除します。

a. **ScanSettingBinding** オブジェクトを削除します。

```
$ oc delete ssb <ScanSettingBinding-name> -n openshift-compliance
```

b. **ScanSetting** オブジェクトを削除します。

```
$ oc delete ss <ScanSetting-name> -n openshift-compliance
```

c. **ComplianceSuite** オブジェクトを削除します。

```
$ oc delete suite <compliancesuite-name> -n openshift-compliance
```

d. **ComplianceScan** オブジェクトを削除します。

```
$ oc delete scan <compliancescan-name> -n openshift-compliance
```

e. **ProfileBundle** オブジェクトを取得します。

```
$ oc get profilebundle.compliance -n openshift-compliance
```

#### 出力例

```
NAME          CONTENTIMAGE                                     CONTENTFILE
STATUS
ocp4 registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:
<hash> ssg-ocp4-ds.xml VALID
rhcos4 registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:
<hash> ssg-rhcos4-ds.xml VALID
```

f. **ProfileBundle** オブジェクトを削除します。

```
$ oc delete profilebundle.compliance ocp4 rhcos4 -n openshift-compliance
```



## 出力例

```
profilebundle.compliance.openshift.io "ocp4" deleted
profilebundle.compliance.openshift.io "rhcos4" deleted
```

- Subscription オブジェクトを削除します。

```
$ oc delete sub <Subscription-Name> -n openshift-compliance
```

- CSV オブジェクトを削除します。

```
$ oc delete CSV -n openshift-compliance
```

- プロジェクトを削除します。

```
$ oc delete project -n openshift-compliance
```

## 出力例

```
project.project.openshift.io "openshift-compliance" deleted
```

## 検証

- namespace が削除されていることを確認します。

```
$ oc get project/openshift-compliance
```

## 出力例

```
Error from server (NotFound): namespaces "openshift-compliance" not found
```

## 5.14. OC-COMPLIANCE プラグインの使用

[コンプライアンス Operator](#) はクラスターのチェックおよび修復の多くを自動化しますが、クラスターを準拠させる完全なプロセスでは、管理者がコンプライアンス Operator API や他のコンポーネントと対話する必要があります。**oc-compliance** プラグインはプロセスを容易にします。

### 5.14.1. oc-compliance プラグインのインストール

#### 手順

- oc-compliance** イメージを展開して、**oc-compliance** バイナリーを取得します。

```
$ podman run --rm -v ~/.local/bin:/mnt/out:Z registry.redhat.io/compliance/oc-compliance-rhel8:stable /bin/cp /usr/bin/oc-compliance /mnt/out/
```

## 出力例

```
W0611 20:35:46.486903 11354 manifest.go:440] Chose linux/amd64 manifest from the manifest list.
```

**oc-compliance** を実行できるようになりました。

## 5.14.2. 未加工の結果の取得

コンプライアンススキャンが完了すると、個別のチェックの結果は生成される

**ComplianceCheckResult** カスタムリソース (CR) に一覧表示されます。ただし、管理者または監査担当者にはスキャンの詳細情報が必要になる場合があります。OpenSCAP ツールは、詳細な結果で Advanced Recording Format (ARF) 形式のファイルを作成します。この ARF ファイルは設定マップまたは他の標準の Kubernetes リソースに保存するには大きすぎるため、永続ボリューム (PV) がこれを組み込むように作成されます。

### 手順

- コンプライアンス Operator を使用した PV からの結果の取得は 4 つの手順で実行されます。ただし、**oc-compliance** プラグインでは、単一のコマンドを使用できます。

```
$ oc compliance fetch-raw <object-type> <object-name> -o <output-path>
```

- **<object-type>** は、スキャンの機能に使用されたオブジェクトによって、**scansettingbinding**、**compliancecan** または **compliancesuite** のいずれかにすることができます。
- **<object-name>** は、ARF ファイルを収集に使用するバインディング、スイート、またはスキャンオブジェクトの名前です。**<output-path>** は、結果を配置するローカルディレクトリーです。  
以下に例を示します。

```
$ oc compliance fetch-raw scansettingbindings my-binding -o /tmp/
```

### 出力例

```
Fetching results for my-binding scans: ocp4-cis, ocp4-cis-node-worker, ocp4-cis-node-master
Fetching raw compliance results for scan 'ocp4-cis'.....
The raw compliance results are available in the following directory: /tmp/ocp4-cis
Fetching raw compliance results for scan 'ocp4-cis-node-worker'.....
The raw compliance results are available in the following directory: /tmp/ocp4-cis-node-worker
Fetching raw compliance results for scan 'ocp4-cis-node-master'.....
The raw compliance results are available in the following directory: /tmp/ocp4-cis-node-master
```

ディレクトリー内のファイルの一覧を表示します。

```
$ ls /tmp/ocp4-cis-node-master/
```

### 出力例

```
ocp4-cis-node-master-ip-10-0-128-89.ec2.internal-pod.xml.bzip2 ocp4-cis-node-master-ip-10-0-150-5.ec2.internal-pod.xml.bzip2 ocp4-cis-node-master-ip-10-0-163-32.ec2.internal-pod.xml.bzip2
```

結果を抽出します。

```
$ bunzip2 -c resultsdir/worker-scan/worker-scan-stage-459-tqkg7-compute-0-pod.xml.bzip2 >
resultsdir/worker-scan/worker-scan-ip-10-0-170-231.us-east-2.compute.internal-pod.xml
```

結果を表示します。

```
$ ls resultsdir/worker-scan/
```

### 出力例

```
worker-scan-ip-10-0-170-231.us-east-2.compute.internal-pod.xml
worker-scan-stage-459-tqkg7-compute-0-pod.xml.bzip2
worker-scan-stage-459-tqkg7-compute-1-pod.xml.bzip2
```

### 5.14.3. スキャンの再実行

スキャンをスケジュールされたジョブとして実行することは可能ですが、多くの場合、修復の適用後、またはクラスターへの他の変更が加えられるなどの場合にとくにスキャンをオンデマンドで再実行する必要があります。

#### 手順

- コンプライアンス Operator でスキャンを再実行するには、スキャンオブジェクトでアノテーションを使用する必要があります。ただし、**oc-compliance** プラグインを使用すると、1つのコマンドでスキャンを再実行できます。以下のコマンドを入力して、**my-binding** という名前の **ScanSettingBinding** オブジェクトのスキャンを再実行します。

```
$ oc compliance rerun-now scansettingbindings my-binding
```

#### 出力例

```
Rerunning scans from 'my-binding': ocp4-cis
Re-running scan 'openshift-compliance/ocp4-cis'
```

### 5.14.4. ScanSettingBinding カスタムリソースの使用

コンプライアンス Operator が提供する **ScanSetting** および **ScanSettingBinding** カスタムリソース (CR) を使用する場合は、**schedule**、**machine roles**、**tolerations** などのスキャンオプションのセットを使用している間に、複数のプロファイルに対してスキャンを実行できます。複数の **ComplianceSuite** オブジェクトまたは **ComplianceScan** オブジェクトを使用することがより容易になりますが、新規ユーザーが混乱を生じさせる可能性があります。

**oc compliance bind** サブコマンドは、**ScanSettingBinding** CR の作成に役立ちます。

#### 手順

1. 以下を実行します。

```
$ oc compliance bind [--dry-run] -N <binding name> [-S <scansetting name>]
<objtype/objname> [..<objtype/objname>]
```

- **-S** フラグを省略する場合、コンプライアンス Operator によって提供される **default** のスキャン設定が使用されます。

- オブジェクトタイプは、Kubernetes オブジェクトタイプです。 **profile** または **tailoredprofile** にすることができます。複数のオブジェクトを指定できます。
- オブジェクト名は、 **.metadata.name** などの Kubernetes リソースの名前です。
- **--dry-run** オプションを追加して、作成されるオブジェクトの YAML ファイルを表示します。  
たとえば、以下のプロファイルとスキャン設定を指定します。

```
$ oc get profile.compliance -n openshift-compliance
```

#### 出力例

```
NAME          AGE
ocp4-cis      9m54s
ocp4-cis-node 9m54s
ocp4-e8       9m54s
ocp4-moderate 9m54s
ocp4-ncp     9m54s
rhcos4-e8    9m54s
rhcos4-moderate 9m54s
rhcos4-ncp   9m54s
rhcos4-ospp  9m54s
rhcos4-stig  9m54s
```

```
$ oc get scansettings -n openshift-compliance
```

#### 出力例

```
NAME          AGE
default       10m
default-auto-apply 10m
```

2. **default** 設定を **ocp4-cis** および **ocp4-cis-node** プロファイルに適用するには、以下を実行します。

```
$ oc compliance bind -N my-binding profile/ocp4-cis profile/ocp4-cis-node
```

#### 出力例

```
Creating ScanSettingBinding my-binding
```

**ScanSettingBinding** CR が作成されると、バインドされたプロファイルは、関連する設定で両方のプロファイルのスキャンを開始します。全体的に、これはコンプライアンス Operator でスキャンを開始するための最も高速な方法です。

### 5.14.5. コントロールの表示

コンプライアンス標準は通常、以下のように階層に編成されます。

- ベンチマークは、特定の標準についての一連のコントロールの最上位の定義です。例: FedRAMP Moderate または Center for Internet Security (CIS) v.1.6.0。

- コントロールは、ベンチマークへの準拠を確保するために満たさなければならない要件のファミリーを説明します。例: FedRAMP AC-01(アクセス制御ポリシーおよび手順)。
- ルールは、コンプライアンスを取るシステムに固有の単一のチェックであり、これらの1つ以上のルールがコントロールにマップされます。
- コンプライアンス Operator は、単一のベンチマークについてのプロファイルへのルールのグループ化に対応します。プロファイルの一連のルールの条件を満たすコントロールを判別することが容易ではない場合があります。

## 手順

- **oc compliance controls** サブコマンドは、指定されたプロファイルが満たす標準およびコントロールのレポートを提供します。

```
$ oc compliance controls profile ocp4-cis-node
```

## 出力例

```
+-----+-----+
| FRAMEWORK | CONTROLS |
+-----+-----+
| CIS-OCF  | 1.1.1  |
+       +-----+
|       | 1.1.10 |
+       +-----+
|       | 1.1.11 |
+       +-----+
...
```

### 5.14.6. コンプライアンス修復の詳細情報の取得

コンプライアンス Operator は、クラスターが準拠できるようにする必要な変更を自動化するために使用される修復オブジェクトを提供します。**fetch-fixes** サブコマンドは、使用する設定の修復を正確に理解するのに役立ちます。**fetch-fixes** サブコマンドを使用して、検査するディレクトリーにプロファイル、ルール、または **ComplianceRemediation** オブジェクトから修復オブジェクトを展開します。

## 手順

1. プロファイルの修復を表示します。

```
$ oc compliance fetch-fixes profile ocp4-cis -o /tmp
```

## 出力例

```
No fixes to persist for rule 'ocp4-api-server-api-priority-flowschema-catch-all' 1
No fixes to persist for rule 'ocp4-api-server-api-priority-gate-enabled'
No fixes to persist for rule 'ocp4-api-server-audit-log-maxbackup'
Persisted rule fix to /tmp/ocp4-api-server-audit-log-maxsize.yaml
No fixes to persist for rule 'ocp4-api-server-audit-log-path'
No fixes to persist for rule 'ocp4-api-server-auth-mode-no-aa'
No fixes to persist for rule 'ocp4-api-server-auth-mode-node'
No fixes to persist for rule 'ocp4-api-server-auth-mode-rbac'
```

```
No fixes to persist for rule 'ocp4-api-server-basic-auth'
No fixes to persist for rule 'ocp4-api-server-bind-address'
No fixes to persist for rule 'ocp4-api-server-client-ca'
Persisted rule fix to /tmp/ocp4-api-server-encryption-provider-cipher.yaml
Persisted rule fix to /tmp/ocp4-api-server-encryption-provider-config.yaml
```

- 1 ルールが自動的に修復されないか、または修復が行われなかったために対応する修復のないルールがプロファイルにある場合は常に、**No fixes to persist** の警告が出されることが予想されます。

2. YAML ファイルのサンプルを表示できます。**head** コマンドは、最初の 10 行を表示します。

```
$ head /tmp/ocp4-api-server-audit-log-maxsize.yaml
```

### 出力例

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  maximumFileSizeMegabytes: 100
```

3. スキャン後に作成される **ComplianceRemediation** オブジェクトから修復を表示します。

```
$ oc get complianceremediations -n openshift-compliance
```

### 出力例

```
NAME                                STATE
ocp4-cis-api-server-encryption-provider-cipher  NotApplied
ocp4-cis-api-server-encryption-provider-config  NotApplied
```

```
$ oc compliance fetch-fixes complianceremediations ocp4-cis-api-server-encryption-provider-cipher -o /tmp
```

### 出力例

```
Persisted compliance remediation fix to /tmp/ocp4-cis-api-server-encryption-provider-cipher.yaml
```

4. YAML ファイルのサンプルを表示できます。**head** コマンドは、最初の 10 行を表示します。

```
$ head /tmp/ocp4-cis-api-server-encryption-provider-cipher.yaml
```

### 出力例

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
```

```
spec:
  encryption:
    type: aescbc
```



### 警告

修復を直接適用するには注意が必要です。一部の修復は、適度なプロファイルの usbguard ルールなどのように一括して適用できない場合があります。このような場合、コンプライアンス Operator は依存関係に対応し、クラスターは正常な状態のままになるため、ルールを適用できます。

## 5.14.7. ComplianceCheckResult オブジェクトの詳細の表示

スキャンの実行が終了すると、**ComplianceCheckResult** オブジェクトが個別のスキャンルールについて作成されます。**view-result** サブコマンドは、**ComplianceCheckResult** オブジェクトの詳細の人間が判読できる出力を提供します。

### 手順

- 以下を実行します。

```
$ oc compliance view-result ocp4-cis-scheduler-no-bind-address
```

## 5.15. カスタムリソース定義を理解する

OpenShift Container Platform のコンプライアンス Operator は、コンプライアンススキャンを実行するためのいくつかのカスタムリソース定義 (CRD) を提供します。コンプライアンススキャンを実行するには、[ComplianceAsCode](#) コミュニティープロジェクトから派生した事前定義されたセキュリティーポリシーを利用します。コンプライアンスオペレーターは、これらのセキュリティーポリシーを CRD に変換します。これを使用して、コンプライアンススキャンを実行し、見つかった問題の修正を取得できます。

### 5.15.1. CRD ワークフロー

CRD は、コンプライアンススキャンを完了するための次のワークフローを提供します。

1. コンプライアンススキャン要件を定義する
2. コンプライアンススキャン設定を設定する
3. コンプライアンススキャン設定を使用してコンプライアンス要件を処理する
4. コンプライアンススキャンをモニターする
5. コンプライアンススキャンの結果を確認する

### 5.15.2. コンプライアンススキャン要件の定義

デフォルトでは、コンプライアンス Operator CRD には **ProfileBundle** オブジェクトと **Profile** オブ

ジェクトが含まれており、これらのオブジェクトでコンプライアンススキャン要件のルールを定義および設定できます。**TailoredProfile** オブジェクトを使用して、デフォルトのプロファイルのカスタマイズすることもできます。

### 5.15.2.1. ProfileBundle オブジェクト

コンプライアンス Operator のインストール時に、すぐに実行できる **ProfileBundle** オブジェクトが含まれます。コンプライアンスオペレーターは、**ProfileBundle** オブジェクトを解析し、バンドル内のプロファイルごとに **Profile** オブジェクトを作成します。また、**Profile** オブジェクトによって使用される **Rule** オブジェクトと **Variable** オブジェクトも解析します。

#### ProfileBundle オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
  name: <profile bundle name>
  namespace: openshift-compliance
status:
  dataStreamStatus: VALID ❶
```

❶ コンプライアンスオペレーターがコンテンツファイルを解析できたかどうかを示します。



#### 注記

**contentFile** が失敗すると、発生したエラーの詳細を提供する **errorMessage** 属性が表示されます。

#### トラブルシューティング

無効なイメージから既知のコンテンツイメージにロールバックすると、**ProfileBundle** オブジェクトは応答を停止し、**PENDING** 状態を表示します。回避策として、前のイメージとは異なるイメージに移動できます。または、**ProfileBundle** オブジェクトを削除して再作成し、作業状態に戻すこともできます。

### 5.15.2.2. プロファイルオブジェクト

**Profile** オブジェクトは、特定のコンプライアンス標準について評価できるルールと変数を定義します。XCCDF 識別子や **Node** または **Platform** タイプのプロファイルチェックなど、OpenSCAP プロファイルに関する解析済みの詳細が含まれています。**Profile** オブジェクトを直接使用することも、**TailorProfile** オブジェクトを使用してさらにカスタマイズすることもできます。



#### 注記

**Profile** オブジェクトは単一の **ProfileBundle** オブジェクトから派生しているため、手動で作成または変更することはできません。通常、1つの **ProfileBundle** オブジェクトに複数の **Profile** オブジェクトを含めることができます。

#### Profile オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
description: <description of the profile>
id: xccdf_org.ssgproject.content_profile_moderate ❶
```



```

kind: Profile
metadata:
  annotations:
    compliance.openshift.io/product: <product name>
    compliance.openshift.io/product-type: Node ❷
  creationTimestamp: "YYYY-MM-DDTMM:HH:SSZ"
  generation: 1
  labels:
    compliance.openshift.io/profile-bundle: <profile bundle name>
  name: rhcos4-moderate
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ProfileBundle
    name: <profile bundle name>
    uid: <uid string>
  resourceVersion: "<version number>"
  selfLink: /apis/compliance.openshift.io/v1alpha1/namespaces/openshift-compliance/profiles/rhcos4-moderate
  uid: <uid string>
  rules: ❸
  - rhcos4-account-disable-post-pw-expiration
  - rhcos4-accounts-no-uid-except-zero
  - rhcos4-audit-rules-dac-modification-chmod
  - rhcos4-audit-rules-dac-modification-chown
  title: <title of the profile>

```

- ❶ プロファイルの XCCDF 名を指定します。スキャンのプロファイル属性の値として **ComplianceScan** オブジェクトを定義する場合は、この識別子を使用します。
- ❷ **Node** または **Platform** のいずれかを指定します。ノードプロファイルはクラスターノードをスキャンし、プラットフォームプロファイルは Kubernetes プラットフォームをスキャンします。
- ❸ プロファイルのルールをリストを指定します。各ルールは1つのチェックに対応します。

### 5.15.2.3. ルールオブジェクト

プロファイルを形成する **Rule** オブジェクトも、オブジェクトとして公開されます。**Rule** オブジェクトを使用して、コンプライアンスチェック要件を定義し、それを修正する方法を指定します。

#### Rule オブジェクトの例

```

apiVersion: compliance.openshift.io/v1alpha1
checkType: Platform ❶
description: <description of the rule>
id: xccdf_org.ssgproject.content_rule_configure_network_policies_namespaces ❷
instructions: <manual instructions for the scan>
kind: Rule
metadata:
  annotations:
    compliance.openshift.io/rule: configure-network-policies-namespaces
    control.compliance.openshift.io/CIS-OCP: 5.3.2

```

```

control.compliance.openshift.io/NERC-CIP: CIP-003-3 R4;CIP-003-3 R4.2;CIP-003-3
R5;CIP-003-3 R6;CIP-004-3 R2.2.4;CIP-004-3 R3;CIP-007-3 R2;CIP-007-3 R2.1;CIP-007-3
R2.2;CIP-007-3 R2.3;CIP-007-3 R5.1;CIP-007-3 R6.1
control.compliance.openshift.io/NIST-800-53: AC-4;AC-4(21);CA-3(5);CM-6;CM-6(1);CM-7;CM-
7(1);SC-7;SC-7(3);SC-7(5);SC-7(8);SC-7(12);SC-7(13);SC-7(18)
labels:
  compliance.openshift.io/profile-bundle: ocp4
  name: ocp4-configure-network-policies-namespaces
  namespace: openshift-compliance
  rationale: <description of why this rule is checked>
  severity: high ❸
  title: <summary of the rule>

```

- ❶ このルールが実行するチェックのタイプを指定します。**Node** プロファイルはクラスターノードをスキャンし、**Platform** プロファイルは Kubernetes プラットフォームをスキャンします。空の値は、自動チェックがないことを示します。
- ❷ データストリームから直接解析されるルールの XCCDF 名を指定します。
- ❸ 失敗したときのルールの重大度を指定します。



#### 注記

**Rule** オブジェクトは、関連付けられた **ProfileBundle** オブジェクトを簡単に識別できるように適切なラベルを取得します。**ProfileBundle** は、このオブジェクトの **OwnerReferences** でも指定されます。

### 5.15.2.4. TailoredProfile オブジェクト

**TailoredProfile** オブジェクトを使用して、組織の要件に基づいてデフォルトの **Profile** オブジェクトを変更します。ルールを有効または無効にしたり、変数値を設定したり、カスタマイズの正当性を示したりすることができます。検証後、**TailoredProfile** オブジェクトは **ConfigMap** を作成します。これは、**ComplianceScan** オブジェクトから参照できます。

#### ヒント

**ScanSettingBinding** オブジェクトで参照することにより、**TailoredProfile** オブジェクトを使用できます。**ScanSettingBinding** の詳細については、ScanSettingBinding オブジェクトを参照してください。

#### TailoredProfile オブジェクトの例

```

apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: rhcos4-with-usb
spec:
  extends: rhcos4-moderate ❶
  title: <title of the tailored profile>
  disableRules:
    - name: <name of a rule object to be disabled>
      rationale: <description of why this rule is checked>
status:
  id: xccdf_compliance.openshift.io_profile_rhcos4-with-usb ❷

```

```
outputRef:
  name: rhcos4-with-usb-tp ❸
  namespace: openshift-compliance
state: READY ❹
```

- ❶ これは任意です。 **TailoredProfile** がビルドされる **Profile** オブジェクトの名前。値が設定されていない場合は、 **enableRules** リストから新しいプロファイルが作成されます。
- ❷ 調整されたプロファイルの XCCDF 名を指定します。
- ❸ **ConfigMap** 名を指定します。これは、 **ComplianceScan** の **tailoringConfigMap.name** 属性の値として使用できます。
- ❹ **READY**、**PENDING**、**FAILURE** などのオブジェクトの状態を表示します。オブジェクトの状態が **ERROR** の場合、属性 **status.errorMessage** が失敗の理由を提供します。

**TailoredProfile** オブジェクトを使用すると、 **TailoredProfile** コンストラクトを使用して新しい **Profile** オブジェクトを作成できます。新しい **Profile** を作成するには、次の設定パラメーターを設定します。

- 適切なタイトル
- **extends** は空でなければなりません
- **TailoredProfile** オブジェクトのスキャンタイプアノテーション:

```
compliance.openshift.io/product-type: Platform/Node
```



### 注記

**product-type** のアノテーションを設定していない場合、コンプライアンスオペレーターはデフォルトで **Platform** スキャンタイプになります。 **TailoredProfile** オブジェクトの名前に **-node** 接尾辞を追加すると、 **node** スキャンタイプになります。

## 5.15.3. コンプライアンススキャン設定の設定

コンプライアンススキャンの要件を定義した後、スキャンのタイプ、スキャンの発生、およびスキャンの場所を指定することにより、コンプライアンススキャンを設定できます。そのために、コンプライアンス Operator は **ScanSetting** オブジェクトを提供します。

### 5.15.3.1. ScanSetting オブジェクト

**ScanSetting** オブジェクトを使用して、スキャンを実行するための運用ポリシーを定義および再利用します。デフォルトでは、コンプライアンスオペレーターは次の **ScanSetting** オブジェクトを作成します。

- **default** - 1Gi Persistent Volume (PV) を使用して、マスターノードとワーカーノードの両方で毎日午前1時にスキャンを実行し、最後の3つの結果を保持します。修復は自動的に適用も更新もされません。
- **default** - 1Gi Persistent Volume (PV) を使用して、コントロールプレーンとワーカーノードの両方で毎日午前1時にスキャンを実行し、最後の3つの結果を保持します。 **autoApplyRemediations** と **autoUpdateRemediations** の両方が **true** に設定されています。

## ScanSetting オブジェクトの例

```

Name:          default-auto-apply
Namespace:     openshift-compliance
Labels:       <none>
Annotations:  <none>
API Version:  compliance.openshift.io/v1alpha1
Auto Apply Remediations: true
Auto Update Remediations: true
Kind:         ScanSetting
Metadata:
  Creation Timestamp: 2022-10-18T20:21:00Z
  Generation:        1
  Managed Fields:
    API Version: compliance.openshift.io/v1alpha1
    Fields Type: FieldsV1
    fieldsV1:
      f:autoApplyRemediations: 1
      f:autoUpdateRemediations: 2
      f:rawResultStorage:
        ..
      f:nodeSelector:
        ..
      f:node-role.kubernetes.io/master:
      f:pvAccessModes:
      f:rotation:
      f:size:
      f:tolerations:
      f:roles:
      f:scanTolerations:
      f:schedule:
      f:showNotApplicable:
      f:strictNodeScan:
    Manager:      compliance-operator
    Operation:    Update
    Time:         2022-10-18T20:21:00Z
  Resource Version: 38840
  UID:            8cb0967d-05e0-4d7a-ac1c-08a7f7e89e84
Raw Result Storage:
  Node Selector:
    node-role.kubernetes.io/master:
  Pv Access Modes:
    ReadWriteOnce
  Rotation: 3 3
  Size:    1Gi 4
  Tolerations:
    Effect:      NoSchedule
    Key:         node-role.kubernetes.io/master
    Operator:    Exists
    Effect:      NoExecute
    Key:         node.kubernetes.io/not-ready
    Operator:    Exists
    Toleration Seconds: 300
    Effect:      NoExecute
    Key:         node.kubernetes.io/unreachable

```

```

Operator:      Exists
Toleration Seconds: 300
Effect:        NoSchedule
Key:           node.kubernetes.io/memory-pressure
Operator:      Exists
Roles: ⑤
  master
  worker
Scan Tolerations:
  Operator:     Exists
Schedule:       "0 1 * * *" ⑥
Show Not Applicable: false
Strict Node Scan: true
Events:         <none>

```

- ① 自動修復を有効にするには、**true** に設定します。自動修復を無効にするには、**false** に設定します。
- ② コンテンツ更新の自動修復を有効にするには、**true** に設定します。コンテンツ更新の自動修復を無効にするには、**false** に設定します。
- ③ 生の結果形式で保存されたスキャンの数を指定します。デフォルト値は **3** です。古い結果がローテーションされると、管理者はローテーションが発生する前に結果を別の場所に保存する必要があります。
- ④ 生の結果を保存するためにスキャン用に作成する必要があるストレージサイズを指定します。デフォルト値は **1Gi** です。
- ⑥ スキャンを実行する頻度を cron 形式で指定します。



#### 注記

ローテーションポリシーを無効にするには、値を **0** に設定します。

- ⑤ **node-role.kubernetes.io** ラベル値を指定して、**Node** タイプのスキャンをスケジュールします。この値は、**MachineConfigPool** の名前と一致する必要があります。

### 5.15.4. コンプライアンススキャン設定を使用したコンプライアンススキャン要件の処理

コンプライアンススキャン要件を定義し、スキャンを実行するように設定すると、コンプライアンスオペレーターは **ScanSettingBinding** オブジェクトを使用してそれを処理します。

#### 5.15.4.1. ScanSettingBinding オブジェクト

**ScanSettingBinding** オブジェクトを使用して、**Profile** または **TailoredProfile** オブジェクトを参照してコンプライアンス要件を指定します。次に、スキャンの操作上の制約を提供する **ScanSetting** オブジェクトにリンクされます。次に、コンプライアンス Operator は、**ScanSetting** オブジェクトと **ScanSettingBinding** オブジェクトに基づいて **ComplianceSuite** オブジェクトを生成します。

#### ScanSettingBinding オブジェクトの例

```

apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding

```

```

metadata:
  name: <name of the scan>
profiles: ❶
  # Node checks
  - name: rhcos4-with-usb
    kind: TailoredProfile
    apiGroup: compliance.openshift.io/v1alpha1
  # Cluster checks
  - name: ocp4-moderate
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
settingsRef: ❷
  name: my-companys-constraints
  kind: ScanSetting
  apiGroup: compliance.openshift.io/v1alpha1

```

- ❶ **Profile** または **TailoredProfile** オブジェクトの詳細を指定して、環境をスキャンします。
- ❷ スケジュールやストレージサイズなどの運用上の制約を指定します。

**ScanSetting** オブジェクトと **ScanSettingBinding** オブジェクトを作成すると、コンプライアンススイートが作成されます。コンプライアンススイートのリストを取得するには、次のコマンドを実行します。

```
$ oc get compliancesuites
```



### 重要

**ScanSettingBinding** を削除すると、コンプライアンススイートも削除されます。

## 5.15.5. コンプライアンススキャンの追跡

コンプライアンススイートの作成後、**ComplianceSuite** オブジェクトを使用して、デプロイされたスキャンのステータスをモニターできます。

### 5.15.5.1. ComplianceSuite オブジェクト

**ComplianceSuite** オブジェクトは、スキャンの状態を追跡するのに役立ちます。スキャンと全体的な結果を作成するための生の設定が含まれています。

**Node** タイプのスキャンの場合、問題の修正が含まれているため、スキャンを **MachineConfigPool** にマップする必要があります。ラベルを指定する場合は、それがプールに直接適用されることを確認してください。

### ComplianceSuite オブジェクトの例

```

apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
  name: <name of the scan>
spec:
  autoApplyRemediations: false ❶
  schedule: "0 1 * * *" ❷

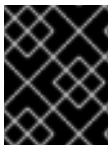
```

```
scans: ③
  - name: workers-scan
    scanType: Node
    profile: xccdf_org.ssgproject.content_profile_moderate
    content: ssg-rhcos4-ds.xml
    contentImage: quay.io/complianceascode/ocp4:latest
    rule: "xccdf_org.ssgproject.content_rule_no_netrc_files"
    nodeSelector:
      node-role.kubernetes.io/worker: ""
status:
  Phase: DONE ④
  Result: NON-COMPLIANT ⑤
  scanStatuses:
    - name: workers-scan
      phase: DONE
      result: NON-COMPLIANT
```

- ① 自動修復を有効にするには、**true** に設定します。自動修復を無効にするには、**false** に設定します。
- ② スキャンを実行する頻度を cron 形式で指定します。
- ③ クラスタで実行するスキャン仕様のリストを指定します。
- ④ スキャンの進行状況を示します。
- ⑤ スイートの全体的な判断を示します。

バックグラウンドのスイートは、**scans** パラメーターに基づいて **ComplianceScan** オブジェクトを作成します。プログラムで **ComplianceSuites** イベントを取得できます。スイートのイベントを取得するには、次のコマンドを実行します。

```
$ oc get events --field-selector involvedObject.kind=ComplianceSuite,involvedObject.name=<name of the suite>
```



### 重要

手動で **ComplianceSuite** を定義すると、XCCDF 属性が含まれているため、エラーが発生する可能性があります。

#### 5.15.5.2. 高度な ComplianceScan オブジェクト

コンプライアンスオペレーターには、デバッグまたは既存のツールとの統合のための上級ユーザー向けのオプションが含まれています。**ComplianceScan** オブジェクトを直接作成しないことをお勧めしますが、代わりに、**ComplianceSuite** オブジェクトを使用してオブジェクトを管理できます。

#### AdvancedComplianceScan オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceScan
metadata:
  name: <name of the scan>
spec:
```



```

scanType: Node ❶
profile: xccdf_org.ssgproject.content_profile_moderate ❷
content: ssg-ocp4-ds.xml
contentImage: quay.io/complianceascode/ocp4:latest ❸
rule: "xccdf_org.ssgproject.content_rule_no_netrc_files" ❹
nodeSelector: ❺
  node-role.kubernetes.io/worker: ""
status:
  phase: DONE ❻
  result: NON-COMPLIANT ❼

```

- ❶ **Node** または **Platform** のいずれかを指定します。ノードプロファイルはクラスターノードをスキャンし、プラットフォームプロファイルは Kubernetes プラットフォームをスキャンします。
- ❷ 実行するプロファイルの XCCDF 識別子を指定します。
- ❸ プロファイルファイルをカプセル化するコンテナイメージを指定します。
- ❹ これはオプションです。単一のルールを実行するスキャンを指定します。このルールは XCCDF ID で識別され、指定されたプロファイルに属している必要があります。



#### 注記

**rule** パラメーターをスキップすると、指定されたプロファイルで使用可能なすべてのルールに対してスキャンが実行されます。

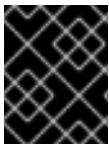
- ❺ OpenShift Container Platform を使用していて、修復を生成したい場合は、**nodeSelector** ラベルが **MachineConfigPool** ラベルと一致する必要があります。



#### 注記

**nodeSelector** パラメーターを指定しないか、**MachineConfig** ラベルと一致しない場合でも、スキャンは実行されますが、修復は作成されません。

- ❻ スキャンの現在のフェーズを示します。
- ❼ スキャンの判定を示します。



#### 重要

**ComplianceSuite** オブジェクトを削除すると、関連するすべてのスキャンが削除されません。

スキャンが完了すると、**ComplianceCheckResult** オブジェクトのカスタムリソースとして結果が生成されます。ただし、生の結果は ARF 形式で入手できます。これらの結果は、スキャンの名前に関連付けられた永続ボリュームクレーム (PVC) を持つ永続ボリューム (PV) に保存されます。プログラムで **ComplianceScans** イベントを取得できます。スイートのイベントを生成するには、次のコマンドを実行します。

```
oc get events --field-selector involvedObject.kind=ComplianceScan,involvedObject.name=<name of the suite>
```



### 5.15.6. コンプライアンス結果の表示

コンプライアンススイートが **DONE** フェーズに達すると、スキャン結果と可能な修正を表示できません。

#### 5.15.6.1. ComplianceCheckResult オブジェクト

特定のプロファイルでスキャンを実行すると、プロファイル内のいくつかのルールが検証されます。これらのルールごとに、**ComplianceCheckResult** オブジェクトが作成され、特定のルールのクラスターの状態が提供されます。

#### ComplianceCheckResult オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceCheckResult
metadata:
  labels:
    compliance.openshift.io/check-severity: medium
    compliance.openshift.io/check-status: FAIL
    compliance.openshift.io/suite: example-compliancesuite
    compliance.openshift.io/scan-name: workers-scan
  name: workers-scan-no-direct-root-logins
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ComplianceScan
    name: workers-scan
  description: <description of scan check>
  instructions: <manual instructions for the scan>
  id: xccdf_org.ssgproject.content_rule_no_direct_root_logins
  severity: medium ①
  status: FAIL ②
```

① スキャンチェックの重大度について説明します。

② チェックの結果を説明します。以下の値を使用できます。

- PASS: チェックは成功しました。
- FAIL: チェックに失敗しました。
- INFO: チェックは成功し、エラーと見なされるほど深刻ではないものが見つかりました。
- MANUAL: チェックではステータスを自動的に評価できないため、手動チェックが必要です。
- INCONSISTENT: ノードが異なれば、報告される結果も異なります。
- ERROR: チェックは正常に実行されましたが、完了できませんでした。
- NOTAPPLICABLE: 該当しないため、チェックは実行されませんでした。

スイートからすべてのチェック結果を取得するには、次のコマンドを実行します。

```
oc get compliancecheckresults \
-l compliance.openshift.io/suite=workers-compliancesuite
```

### 5.15.6.2. ComplianceRemediation オブジェクト

特定のチェックについては、データストリームで指定された修正を行うことができます。ただし、Kubernetes 修正が利用可能な場合、コンプライアンスオペレーターは **ComplianceRemediation** オブジェクトを作成します。

#### ComplianceRemediation オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceRemediation
metadata:
  labels:
    compliance.openshift.io/suite: example-compliancesuite
    compliance.openshift.io/scan-name: workers-scan
    machineconfiguration.openshift.io/role: worker
  name: workers-scan-disable-users-coredumps
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ComplianceCheckResult
    name: workers-scan-disable-users-coredumps
    uid: <UID>
spec:
  apply: false ❶
  object:
    current: ❷
    apiVersion: machineconfiguration.openshift.io/v1
    kind: MachineConfig
    spec:
      config:
        ignition:
          version: 2.2.0
        storage:
          files:
          - contents:
              source: data:,%2A%20%20%20%20%20hard%20%20%20core%20%20%20%200
            filesystem: root
            mode: 420
            path: /etc/security/limits.d/75-disable_users_coredumps.conf
    outdated: {} ❸
```

❶ **true** は、修復が適用されたことを示します。**false** は、修復が適用されなかったことを示します。

❷ 修復の定義が含まれています。

❸ 以前のバージョンのコンテンツから以前に解析された修復を示します。コンプライアンスオペレーターは、古いオブジェクトを引き続き保持して、管理者が新しい修復を適用する前に確認できるようにします。

スイートからすべての修復を取得するには、次のコマンドを実行します。

```
oc get complianceremediations \  
-l compliance.openshift.io/suite=workers-compliancesuite
```

自動的に修正できるすべての失敗したチェックを一覧表示するには、次のコマンドを実行します。

```
oc get compliancecheckresults \  
-l 'compliance.openshift.io/check-status in (FAIL),compliance.openshift.io/automated-remediation'
```

手動で修正できるすべての失敗したチェックを一覧表示するには、次のコマンドを実行します。

```
oc get compliancecheckresults \  
-l 'compliance.openshift.io/check-status in (FAIL),!compliance.openshift.io/automated-remediation'
```

## 第6章 FILE INTEGRITY OPERATOR

### 6.1. FILE INTEGRITY OPERATOR リリースノート

OpenShift Container Platform のファイル整合性オペレーターは、RHCOS ノードでファイル整合性チェックを継続的に実行します。

これらのリリースノートは、OpenShift Container Platform での File Integrity Operator の開発を追跡します。

File Integrity Operator の概要については、[Understanding the File Integrity Operator](#) を参照してください。

最新リリースにアクセスするには、[File Integrity Operator の更新](#) を参照してください。

#### 6.1.1. OpenShift File Integrity Operator 1.2.1

OpenShift File Integrity Operator 1.2.1 については、以下のアドバイザリーを利用できます。

- [RHBA-2023:1684 OpenShift File Integrity Operator Bug Fix Update](#)
- このリリースには、更新されたコンテナの依存関係が含まれています。

#### 6.1.2. OpenShift File Integrity Operator 1.2.0

OpenShift File Integrity Operator 1.2.0 については、以下のアドバイザリーを利用できます。

- [RHBA-2023:1273 OpenShift File Integrity Operator 機能拡張の更新](#)

##### 6.1.2.1. 新機能および拡張機能

- File Integrity Operator カスタムリソース (CR) には、最初の AIDE 整合性チェックを開始する前に待機する秒数を指定する **initialDelay** 機能が含まれるようになりました。詳細は、[FileIntegrity カスタムリソースの作成](#) を参照してください。
- File Integrity Operator は安定版になり、リリースチャンネルは **stable** にアップグレードされました。将来のリリースは [Semantic Versioning](#) に従います。最新リリースにアクセスするには、[File Integrity Operator の更新](#) を参照してください。

#### 6.1.3. OpenShift File Integrity Operator 1.0.0

OpenShift File Integrity Operator 1.0.0 については、以下のアドバイザリーを利用できます。

- [RHBA-2023:0037 OpenShift File Integrity Operator Bug Fix Update](#)

#### 6.1.4. OpenShift File Integrity Operator 0.1.32

OpenShift File Integrity Operator 0.1.32 については、以下のアドバイザリーを利用できます。

- [RHBA-2022:7095 OpenShift File Integrity Operator Bug Fix Update](#)

##### 6.1.4.1. バグ修正

- 以前は、File Integrity Operator によって発行されたアラートは namespace を設定していな

かったため、アラートが発生した namespace を理解することが困難でした。これで、Operator は適切な namespace を設定し、アラートに関する詳細情報を提供します。  
([BZ#2112394](#))

- 以前は、File Integrity Operator は Operator の起動時にメトリクスサービスを更新しなかったため、メトリクスターゲットに到達できませんでした。今回のリリースでは、File Integrity Operator により、Operator の起動時にメトリクスサービスが確実に更新されるようになりました。  
([BZ#2115821](#))

### 6.1.5. OpenShift File Integrity Operator 0.1.30

OpenShift File Integrity Operator 0.1.30 については、以下のアドバイザリーを利用できます。

- [RHBA-2022:5538 OpenShift File Integrity Operator のバグ修正と機能拡張の更新](#)

#### 6.1.5.1. バグ修正

- 以前は、File Integrity Operator が発行したアラートでは namespace が設定されていなかったため、アラートの発生場所を理解することが困難でした。現在、Operator は適切な namespace を設定し、アラートが理解できるように改善されています。  
([BZ#2101393](#))

### 6.1.6. OpenShift File Integrity Operator 0.1.24

OpenShift File Integrity Operator 0.1.24 については、以下のアドバイザリーを利用できます。

- [RHBA-2022:1331 OpenShift File Integrity Operator Bug Fix](#)

#### 6.1.6.1. 新機能および拡張機能

- **config.maxBackups** 属性を使用して、**FileIntegrity** カスタムリソース (CR) に保存されるバックアップの最大数を設定できるようになりました。この属性は、ノードに保持するために **re-init** プロセスから残された AIDE データベースおよびログのバックアップの数を指定します。設定された数を超える古いバックアップは自動的にブルーニングされます。デフォルトは5つのバックアップに設定されています。

#### 6.1.6.2. バグ修正

- 以前は、Operator を 0.1.21 より古いバージョンから 0.1.22 にアップグレードすると、**re-init** 機能が失敗する可能性がありました。これは、オペレーターが **configMap** リソースラベルの更新に失敗した結果です。現在、最新バージョンにアップグレードすると、リソースラベルが修正されます。  
([BZ#2049206](#))
- 以前は、デフォルトの **configMap** スクリプトの内容を適用するときに、間違っただータキーが比較されていました。これにより、Operator のアップグレード後に **aide-reinit** スクリプトが適切に更新されず、**re-init** プロセスが失敗することがありました。これで、**daemonSets** が完了するまで実行され、AIDE データベースの **re-init** プロセスが正常に実行されます。  
([BZ#2072058](#))

### 6.1.7. OpenShift File Integrity Operator 0.1.22

OpenShift File Integrity Operator 0.1.22 については、以下のアドバイザリーを利用できます。

- [RHBA-2022:0142 OpenShift File Integrity Operator のバグ修正](#)

### 6.1.7.1. バグ修正

- 以前は、File Integrity Operator がインストールされているシステムが、`/etc/kubernetes/aid.reinit` ファイルが原因で、OpenShift Container Platform の更新を中断する可能性があります。これは、`/etc/kubernetes/aide.reinit` ファイルが存在したが、後で **ostree** 検証の前に削除された場合に発生しました。今回の更新では、`/etc/kubernetes/aide.reinit` が `/run` ディレクトリーに移動し、OpenShift Container Platform の更新と競合しないようになっています。( [BZ#2033311](#) )

## 6.1.8. OpenShift File Integrity Operator 0.1.21

OpenShift File Integrity Operator 0.1.21 については、以下のアドバイザリーを利用できます。

- [RHBA-2021:4631 OpenShift File Integrity Operator のバグ修正と機能拡張の更新](#)

### 6.1.8.1. 新機能および拡張機能

- **FileIntegrity** スキャン結果および処理メトリックに関連するメトリックは、Web コンソールの監視ダッシュボードに表示されます。結果には、`file_integrity_operator_` の接頭辞が付けられます。
- ノードの整合性障害が1秒を超えると、Operator の namespace で提供されるデフォルトの **PrometheusRule** が警告を発します。
- 次の動的な Machine Config Operator および Cluster Version Operator 関連のファイルパスは、ノードの更新中の誤検知を防ぐために、デフォルトの AIDE ポリシーから除外されています。
  - `/etc/machine-config-daemon/currentconfig`
  - `/etc/pki/ca-trust/extracted/java/cacerts`
  - `/etc/cvo/updatepayloads`
  - `/root/.kube`
- AIDE デーモンプロセスは v0.1.16 よりも安定性が向上しており、AIDE データベースの初期化時に発生する可能性のあるエラーに対する耐性が高くなっています。

### 6.1.8.2. バグ修正

- 以前は、Operator が自動的にアップグレードしたときに、古いデーモンセットは削除されませんでした。このリリースでは、自動アップグレード中に古いデーモンセットが削除されます。

## 6.1.9. 関連情報

- [File Integrity Operator について](#)

## 6.2. FILE INTEGRITY OPERATOR のインストール

### 6.2.1. Web コンソールでの File Integrity Operator のインストール

#### 前提条件

- **admin** 権限がある。

## 手順

1. OpenShift Container Platform Web コンソールで、**Operators** → **OperatorHub** ページに移動します。
2. File Integrity Operator を検索し、**Install** をクリックします。
3. **Installation mode** および **namespace** のデフォルトの選択を維持し、Operator が **openshift-file-integrity** namespace にインストールされていることを確認します。
4. **Install** をクリックします。

## 検証

インストールが正常に行われたことを確認するには、以下を実行します。

1. **Operators** → **Installed Operators** ページに移動します。
2. Operator が **openshift-file-integrity** namespace にインストールされており、そのステータスが **Succeeded** であることを確認します。

Operator が正常にインストールされていない場合、以下を実行します。

1. **Operators** → **Installed Operators** ページに移動し、**Status** 列でエラーまたは失敗の有無を確認します。
2. **Workloads** → **Pods** ページに移動し、**openshift-file-integrity** プロジェクトの Pod で問題を報告しているログの有無を確認します。

## 6.2.2. CLI を使用した File Integrity Operator のインストール

### 前提条件

- **admin** 権限がある。

### 手順

1. 以下を実行して **Namespace** オブジェクト YAML ファイルを作成します。

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

### 出力例

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: openshift-file-integrity
```

2. **OperatorGroup** オブジェクト YAML ファイルを作成します。

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

### 出力例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: file-integrity-operator
  namespace: openshift-file-integrity
spec:
  targetNamespaces:
    - openshift-file-integrity
```

3. **Subscription** オブジェクト YAML ファイルを作成します。

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

### 出力例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: file-integrity-operator
  namespace: openshift-file-integrity
spec:
  channel: "stable"
  installPlanApproval: Automatic
  name: file-integrity-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

### 検証

1. CSV ファイルを確認して、インストールが正常に完了したことを確認します。

```
$ oc get csv -n openshift-file-integrity
```

2. File Integrity Operator が稼働していることを確認します。

```
$ oc get deploy -n openshift-file-integrity
```

### 6.2.3. 関連情報

- File Integrity Operator はネットワークが制限された環境でサポートされています。詳細は、[ネットワークが制限された環境での Operator Lifecycle Manager の使用](#) を参照してください。

## 6.3. FILE INTEGRITY OPERATOR の更新

クラスター管理者は、OpenShift Container Platform クラスターで File Integrity Operator を更新できません。

### 6.3.1. Operator 更新の準備



インストールされた Operator のサブスクリプションは、Operator の更新を追跡および受信する更新チャンネルを指定します。更新チャンネルを変更して、新しいチャンネルからの更新の追跡と受信を開始できます。

サブスクリプションの更新チャンネルの名前は Operator 間で異なる可能性があります。命名スキーム通常、特定の Operator 内の共通の規則に従います。たとえば、チャンネル名は Operator によって提供されるアプリケーションのマイナーリリース更新ストリーム (**1.2**、**1.3**) またはリリース頻度 (**stable**、**fast**) に基づく可能性があります。



### 注記

インストールされた Operator は、現在のチャンネルよりも古いチャンネルに切り換えることはできません。

Red Hat Customer Portal Labs には、管理者が Operator の更新を準備するのに役立つ以下のアプリケーションが含まれています。

- [Red Hat OpenShift Container プラットフォーム Operator Update Information Checker](#)

このアプリケーションを使用して、Operator Lifecycle Manager ベースの Operator を検索し、OpenShift Container Platform の異なるバージョン間で更新チャンネルごとに利用可能な Operator バージョンを確認できます。Cluster Version Operator ベースの Operator は含まれません。

## 6.3.2. Operator の更新チャンネルの変更

OpenShift Container Platform Web コンソールを使用して、Operator の更新チャンネルを変更できます。

### ヒント

サブスクリプションの承認ストラテジーが **Automatic** に設定されている場合、アップグレードプロセスは、選択したチャンネルで新規 Operator バージョンが利用可能になるとすぐに開始します。承認ストラテジーが **Manual** に設定されている場合は、保留中のアップグレードを手動で承認する必要があります。

### 前提条件

- Operator Lifecycle Manager (OLM) を使用して以前にインストールされている Operator。

### 手順

1. Web コンソールの **Administrator** パースペクティブで、**Operators** → **Installed Operators** に移動します。
2. 更新チャンネルを変更する Operator の名前をクリックします。
3. **Subscription** タブをクリックします。
4. **Channel** の下にある更新チャンネルの名前をクリックします。
5. 変更する新しい更新チャンネルをクリックし、**Save** をクリックします。
6. **Automatic** 承認ストラテジーのあるサブスクリプションの場合、更新は自動的に開始します。**Operators** → **Installed Operators** ページに戻り、更新の進捗をモニターします。完了時に、ステータスは **Succeeded** および **Up to date** に変更されます。

**Manual** 承認ストラテジーのあるサブスクリプションの場合、**Subscription** タブから更新を手動で承認できます。

### 6.3.3. 保留中の Operator 更新の手動による承認

インストールされた Operator のサブスクリプションの承認ストラテジーが **Manual** に設定されている場合、新規の更新が現在の更新チャンネルにリリースされると、インストールを開始する前に更新を手動で承認する必要があります。

#### 前提条件

- Operator Lifecycle Manager (OLM) を使用して以前にインストールされている Operator。

#### 手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Operators** → **Installed Operators** に移動します。
2. 更新が保留中の Operator は **Upgrade available** のステータスを表示します。更新する Operator の名前をクリックします。
3. **Subscription** タブをクリックします。承認が必要な更新は、**アップグレードステータス** の横に表示されます。たとえば、**1 requires approval** が表示される可能性があります。
4. **1 requires approval** をクリックしてから、**Preview Install Plan** をクリックします。
5. 更新に利用可能なリソースとして一覧表示されているリソースを確認します。問題がなければ、**Approve** をクリックします。
6. **Operators** → **Installed Operators** ページに戻り、更新の進捗をモニターします。完了時に、ステータスは **Succeeded** および **Up to date** に変更されます。

## 6.4. FILE INTEGRITY OPERATOR について

File Integrity Operator は OpenShift Container Platform Operator であり、クラスターノード上でファイルの整合性チェックを継続的に実行します。これは、各ノードで特権付きの AIDE (advanced intrusion detection environment; 高度な侵入検知環境) コンテナを各ノードで初期化し、実行するデーモンセットをデプロイし、ステータスオブジェクトをデーモンセット Pod の初回実行時に変更されるファイルのログと共に提供します。



#### 重要

現時点では、Red Hat Enterprise Linux CoreOS (RHCOS) ノードのみがサポートされません。

### 6.4.1. FileIntegrity カスタムリソースの作成

**FileIntegrity** カスタムリソース (CR) のインスタンスは、1つ以上のノードの継続的なファイル整合性スキャンのセットを表します。

それぞれの **FileIntegrity** CR は、**FileIntegrity** CR 仕様に一致するノード上で AIDE を実行するデーモンセットによってサポートされます。

#### 手順

1. **worker-fileintegrity.yaml** という名前の次の例の **FileIntegrity** CR を作成して、ワーカーノードでのスキャンを有効にします。

### サンプル FileIntegrity CR

```

apiVersion: fileintegrity.openshift.io/v1alpha1
kind: FileIntegrity
metadata:
  name: worker-fileintegrity
  namespace: openshift-file-integrity
spec:
  nodeSelector: ❶
    node-role.kubernetes.io/worker: ""
  tolerations: ❷
  - key: "myNode"
    operator: "Exists"
    effect: "NoSchedule"
  config: ❸
    name: "myconfig"
    namespace: "openshift-file-integrity"
    key: "config"
    gracePeriod: 20 ❹
    maxBackups: 5 ❺
    initialDelay: 60 ❻
  debug: false
status:
  phase: Active ❼

```

- ❶ ノードスキャンをスケジュールするためのセレクターを定義します。
- ❷ カスタムテイントを持つノードにスケジュールする **tolerations** を指定します。指定しない場合、メインノードとインフラノードでの実行を許可するデフォルトの容認が適用されます。
- ❸ 使用する AIDE 設定を含む **ConfigMap** を定義します。
- ❹ AIDE 整合性チェックの間に一時停止する秒数。ノード上で AIDE チェックを頻繁に実行すると、多くのリソースが消費する可能性があるため、間隔をより長く指定することができます。デフォルトは 900 秒 (15 分) です。
- ❺ ノードで保持する re-init プロセスから残った AIDE データベースとログのバックアップの最大数。この数を超える古いバックアップは、デーモンによって自動的に削除されます。デフォルトは 5 に設定されています。
- ❻ 最初の AIDE 整合性チェックを開始するまで待機する秒数。デフォルトは 0 に設定されています。
- ❼ **FileIntegrity** インスタンスの実行ステータス。ステータスは、**Initializing**、**Pending**、または **Active** です。

#### Initializing

**FileIntegrity** オブジェクトは現在、AIDE データベースを初期化または再初期化しています。

<b>Pending</b>	<b>FileIntegrity</b> デプロイメントはまだ作成中です。
<b>Active</b>	スキャンはアクティブで進行中です。

2. YAML ファイルを **openshift-file-integrity** namespace に適用します。

```
$ oc apply -f worker-fileintegrity.yaml -n openshift-file-integrity
```

#### 検証

- 次のコマンドを実行して、**FileIntegrity** オブジェクトが正常に作成されたことを確認します。

```
$ oc get fileintegrities -n openshift-file-integrity
```

#### 出力例

```
NAME          AGE
worker-fileintegrity 14s
```

### 6.4.2. FileIntegrity カスタムリソースのステータスの確認

**FileIntegrity** カスタムリソース (CR) は、**.status.phase** サブリソースでそのステータスを報告します。

#### 手順

- **FileIntegrity** CR ステータスをクエリーするには、以下を実行します。

```
$ oc get fileintegrities/worker-fileintegrity -o jsonpath="{.status.phase}"
```

#### 出力例

```
Active
```

### 6.4.3. FileIntegrity カスタムリソースの各種フェーズ

- **Pending**: カスタムリソース (CR) の作成後のフェーズ。
- **Active**: バックリングデーモンセットが実行するフェーズ。
- **Initializing**: AIDE データベースが再初期化されるフェーズ。

### 6.4.4. FileIntegrityNodeStatuses オブジェクトについて

**FileIntegrity** CR のスキャン結果は、**FileIntegrityNodeStatuses** という別のオブジェクトで報告されます。

```
$ oc get fileintegritynodestatuses
```

#### 出力例

NAME	AGE
worker-fileintegrity-ip-10-0-130-192.ec2.internal	101s
worker-fileintegrity-ip-10-0-147-133.ec2.internal	109s
worker-fileintegrity-ip-10-0-165-160.ec2.internal	102s



## 注記

**FileIntegrityNodeStatus** オブジェクトの結果が利用可能になるまで、しばらく時間がかかる場合があります。

ノードごとに1つの結果オブジェクトがあります。それぞれの **FileIntegrityNodeStatus** オブジェクトの **nodeName** 属性は、スキャンされるノードに対応します。ファイル整合性スキャンのステータスは、スキャン条件を保持する **results** 配列で表示されます。

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io -ojsonpath='{.items[*].results}' | jq
```

**fileintegritynodestatus** オブジェクトは AIDE 実行の最新のステータスを報告し、**status** フィールドに **Failed**、**Succeeded**、または **Errored** などのステータスを公開します。

```
$ oc get fileintegritynodestatuses -w
```

## 出力例

NAME	NODE	STATUS
example-fileintegrity-ip-10-0-134-186.us-east-2.compute.internal	ip-10-0-134-186.us-east-2.compute.internal	Succeeded
example-fileintegrity-ip-10-0-150-230.us-east-2.compute.internal	ip-10-0-150-230.us-east-2.compute.internal	Succeeded
example-fileintegrity-ip-10-0-169-137.us-east-2.compute.internal	ip-10-0-169-137.us-east-2.compute.internal	Succeeded
example-fileintegrity-ip-10-0-180-200.us-east-2.compute.internal	ip-10-0-180-200.us-east-2.compute.internal	Succeeded
example-fileintegrity-ip-10-0-194-66.us-east-2.compute.internal	ip-10-0-194-66.us-east-2.compute.internal	Failed
example-fileintegrity-ip-10-0-222-188.us-east-2.compute.internal	ip-10-0-222-188.us-east-2.compute.internal	Succeeded
example-fileintegrity-ip-10-0-134-186.us-east-2.compute.internal	ip-10-0-134-186.us-east-2.compute.internal	Succeeded
example-fileintegrity-ip-10-0-222-188.us-east-2.compute.internal	ip-10-0-222-188.us-east-2.compute.internal	Succeeded
example-fileintegrity-ip-10-0-194-66.us-east-2.compute.internal	ip-10-0-194-66.us-east-2.compute.internal	Failed
example-fileintegrity-ip-10-0-150-230.us-east-2.compute.internal	ip-10-0-150-230.us-east-2.compute.internal	Succeeded
example-fileintegrity-ip-10-0-180-200.us-east-2.compute.internal	ip-10-0-180-200.us-east-2.compute.internal	Succeeded

### 6.4.5. FileIntegrityNodeStatus CR ステータスタイプ

これらの条件は、対応する **FileIntegrityNodeStatus** CR ステータスの **results** 配列で報告されます。

- **Succeeded**: 整合性チェックにパスしました。データベースが最後に初期化されてから、AIDE チェックの対象となるファイルおよびディレクトリーは変更されていません。

- **Failed:** 整合性チェックにパスしません。データベースが最後に初期化されてから、AIDE チェックの対象となるファイルまたはディレクトリーが変更されています。
- **Errored:** AIDE スキャナーで内部エラーが発生しました。

#### 6.4.5.1. FileIntegrityNodeStatus CR の成功例

##### 成功ステータスのある状態の出力例

```
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:45:57Z"
  }
]
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:46:03Z"
  }
]
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:45:48Z"
  }
]
]
```

この場合、3つのスキャンがすべて正常に実行され、現時点ではその他の状態は存在しません。

#### 6.4.5.2. FileIntegrityNodeStatus CR の失敗ステータスの例

障害のある状態をシミュレートするには、AIDE が追跡するファイルの1つを変更します。たとえば、ワーカーノードのいずれかで `/etc/resolv.conf` を変更します。

```
$ oc debug node/ip-10-0-130-192.ec2.internal
```

##### 出力例

```
Creating debug namespace/openshift-debug-node-ldfbj ...
Starting pod/ip-10-0-130-192ec2internal-debug ...
To use host binaries, run `chroot /host`
Pod IP: 10.0.130.192
If you don't see a command prompt, try pressing enter.
sh-4.2# echo "# integrity test" >> /host/etc/resolv.conf
sh-4.2# exit

Removing debug pod ...
Removing debug namespace/openshift-debug-node-ldfbj ...
```

しばらくすると、**Failed** 状態が対応する **FileIntegrityNodeStatus** オブジェクトの `results` 配列で報告されます。前回の **Succeeded** 状態は保持されるため、チェックが失敗した時間を特定することができます。

■

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io/worker-fileintegrity-ip-10-0-130-192.ec2.internal -ojsonpath='{.results}' | jq -r
```

または、オブジェクト名に言及していない場合は、以下を実行します。

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io -ojsonpath='{.items[*].results}' | jq
```

## 出力例

```
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:54:14Z"
  },
  {
    "condition": "Failed",
    "filesChanged": 1,
    "lastProbeTime": "2020-09-15T12:57:20Z",
    "resultConfigMapName": "aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed",
    "resultConfigMapNamespace": "openshift-file-integrity"
  }
]
```

**Failed** 状態は、失敗した内容と理由に関する詳細を示す設定マップを参照します。

```
$ oc describe cm aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed
```

## 出力例

```
Name:      aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed
Namespace: openshift-file-integrity
Labels:    file-integrity.openshift.io/node=ip-10-0-130-192.ec2.internal
           file-integrity.openshift.io/owner=worker-fileintegrity
           file-integrity.openshift.io/result-log=
Annotations: file-integrity.openshift.io/files-added: 0
             file-integrity.openshift.io/files-changed: 1
             file-integrity.openshift.io/files-removed: 0
```

### Data

```
integritylog:
```

```
-----
```

```
AIDE 0.15.1 found differences between database and filesystem!!
Start timestamp: 2020-09-15 12:58:15
```

### Summary:

```
Total number of files: 31553
Added files:           0
Removed files:         0
Changed files:         1
```

```
-----
```

```
Changed files:
```

```
-----
changed: /hostroot/etc/resolv.conf
-----
```

```
Detailed information about changes:
-----
```

```
File: /hostroot/etc/resolv.conf
SHA512 : sTQYpB/AL7FeoGtu/1g7opv6C+KT1C BJ , qAeM+a8yTgHPnIHMaRIS+so61EN8VOpg
```

```
Events: <none>
```

設定マップのデータサイズの制限により、1MB を超える AIDE ログが base64 でエンコードされた gzip アーカイブとして障害用の設定マップに追加されます。この場合は、上記のコマンドの出力を **base64 -decode | gunzip** にパイプ処理する必要があります。圧縮されたログの有無は、設定マップにある **file-integrity.openshift.io/compressed** アノテーションキーで示唆されます。

#### 6.4.6. イベントについて

**FileIntegrity** および **FileIntegrityNodeStatus** オブジェクトのステータスの移行は **イベント** でログに記録されます。イベントの作成時間は、**Initializing** から **Active** など、最新の移行を反映し、必ずしも最新のスキャン結果ではありません。ただし、最新のイベントは常に最新のステータスを反映しています。

```
$ oc get events --field-selector reason=FileIntegrityStatus
```

#### 出力例

LAST SEEN	TYPE	REASON	OBJECT	MESSAGE
97s	Normal	FileIntegrityStatus	fileintegrity/example-fileintegrity	Pending
67s	Normal	FileIntegrityStatus	fileintegrity/example-fileintegrity	Initializing
37s	Normal	FileIntegrityStatus	fileintegrity/example-fileintegrity	Active

ノードのスキャンに失敗すると、イベントは **add/changed/removed** および設定マップ情報と共に作成されます。

```
$ oc get events --field-selector reason=NodeIntegrityStatus
```

#### 出力例

LAST SEEN	TYPE	REASON	OBJECT	MESSAGE
114m	Normal	NodeIntegrityStatus	fileintegrity/example-fileintegrity	no changes to node ip-10-0-134-173.ec2.internal
114m	Normal	NodeIntegrityStatus	fileintegrity/example-fileintegrity	no changes to node ip-10-0-168-238.ec2.internal
114m	Normal	NodeIntegrityStatus	fileintegrity/example-fileintegrity	no changes to node ip-10-0-169-175.ec2.internal
114m	Normal	NodeIntegrityStatus	fileintegrity/example-fileintegrity	no changes to node ip-10-0-152-92.ec2.internal
114m	Normal	NodeIntegrityStatus	fileintegrity/example-fileintegrity	no changes to node ip-10-0-158-144.ec2.internal



```
114m Normal NodeIntegrityStatus fileintegrity/example-fileintegrity no changes to node ip-10-0-131-30.ec2.internal
87m Warning NodeIntegrityStatus fileintegrity/example-fileintegrity node ip-10-0-152-92.ec2.internal has changed! a:1,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-fileintegrity-ip-10-0-152-92.ec2.internal-failed
```

ノードのステータスが移行されなかった場合でも、追加、変更、または削除されたファイルの数を変更すると、新しいイベントが生成されます。

```
$ oc get events --field-selector reason=NodeIntegrityStatus
```

## 出力例

LAST SEEN	TYPE	REASON	OBJECT	MESSAGE
114m	Normal	NodeIntegrityStatus	fileintegrity/example-fileintegrity	no changes to node ip-10-0-134-173.ec2.internal
114m	Normal	NodeIntegrityStatus	fileintegrity/example-fileintegrity	no changes to node ip-10-0-168-238.ec2.internal
114m	Normal	NodeIntegrityStatus	fileintegrity/example-fileintegrity	no changes to node ip-10-0-169-175.ec2.internal
114m	Normal	NodeIntegrityStatus	fileintegrity/example-fileintegrity	no changes to node ip-10-0-152-92.ec2.internal
114m	Normal	NodeIntegrityStatus	fileintegrity/example-fileintegrity	no changes to node ip-10-0-158-144.ec2.internal
114m	Normal	NodeIntegrityStatus	fileintegrity/example-fileintegrity	no changes to node ip-10-0-131-30.ec2.internal
87m	Warning	NodeIntegrityStatus	fileintegrity/example-fileintegrity	node ip-10-0-152-92.ec2.internal has changed! a:1,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-fileintegrity-ip-10-0-152-92.ec2.internal-failed
40m	Warning	NodeIntegrityStatus	fileintegrity/example-fileintegrity	node ip-10-0-152-92.ec2.internal has changed! a:3,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-fileintegrity-ip-10-0-152-92.ec2.internal-failed

## 6.5. カスタム FILE INTEGRITY OPERATOR の設定

### 6.5.1. FileIntegrity オブジェクト属性の表示

Kubernetes カスタムリソース (CR) の場合と同様に、**oc explain fileintegrity** を実行してから、以下を使用して個別の属性を参照できます。

```
$ oc explain fileintegrity.spec
```

```
$ oc explain fileintegrity.spec.config
```

### 6.5.2. 重要な属性

表6.1 重要な spec および spec.config 属性

属性	説明
----	----

属性	説明
<b>spec.nodeSelector</b>	AIDE Pod が該当ノードでスケジュール対象にするために使用する、ノードのラベルと一致する必要があるキーと値のペアのマッピング。通常は、 <b>node-role.kubernetes.io/worker: ""</b> がすべてのワーカーノードで AIDE をスケジュールし、 <b>node.openshift.io/os_id: "rhcos"</b> がすべての Red Hat Enterprise Linux CoreOS (RHCOS) ノードでスケジュールする単一のキーと値のペアのみを設定します。
<b>spec.debug</b>	ブール値の属性。 <b>true</b> に設定すると、AIDE デモンセットの Pod で実行されるデーモンは追加の情報を出力します。
<b>spec.tolerations</b>	カスタムテイントを持つノードにスケジュールする容認を指定します。指定されない場合は、デフォルトの容認 (Toleration) が適用され、これにより容認はコントロールプレーンノードで実行できます。
<b>spec.config.gracePeriod</b>	AIDE 整合性チェックの間に一時停止する秒数。ノード上で AIDE チェックを頻繁に実行すると、多くのリソースが消費する可能性があるため、間隔をより長く指定することができます。デフォルトは <b>900</b> (15 分) です。
<b>maxBackups</b>	ノードで保持する <b>re-init</b> 化プロセスから残った AIDE データベースとログのバックアップの最大数。この数を超える古いバックアップは、デーモンによって自動的に削除されます。
<b>spec.config.name</b>	カスタム AIDE 設定を含む configMap の名前。省略した場合、デフォルトの設定が作成されます。
<b>spec.config.namespace</b>	カスタム AIDE 設定を含む configMap の namespace。設定されていない場合、FIO は RHCOS システムに適したデフォルト設定を生成します。
<b>spec.config.key</b>	<b>name</b> と <b>namespace</b> で指定された設定マッピングに実際の AIDE 設定を含むキー。デフォルト値は <b>aide.conf</b> です。
<b>spec.config.initialDelay</b>	最初の AIDE 整合性チェックを開始するまで待機する秒数。デフォルトは 0 に設定されています。この属性は任意です。

### 6.5.3. デフォルト設定の確認

デフォルトの File Integrity Operator 設定は、**FileIntegrity** CR と同じ名前で設定マップに保存されま

## 手順

- デフォルトの設定を確認するには、以下を実行します。

```
$ oc describe cm/worker-fileintegrity
```

### 6.5.4. デフォルトの File Integrity Operator 設定について

以下は、設定マップの **aide.conf** キーの抜粋です。

```
@@define DBDIR /hostroot/etc/kubernetes
@@define LOGDIR /hostroot/etc/kubernetes
database=file:@@{DBDIR}/aide.db.gz
database_out=file:@@{DBDIR}/aide.db.gz
gzip_dbout=yes
verbose=5
report_url=file:@@{LOGDIR}/aide.log
report_url=stdout
PERMS = p+u+g+acl+selinux+xattrs
CONTENT_EX = sha512+ftype+p+u+g+n+acl+selinux+xattrs

/hostroot/boot/ CONTENT_EX
/hostroot/root/\.* PERMS
/hostroot/root/ CONTENT_EX
```

**FileIntegrity** インスタンスのデフォルト設定は、以下のディレクトリ下にあるファイルの範囲を指定します。

- **/root**
- **/boot**
- **/usr**
- **/etc**

以下のディレクトリは対象外です。

- **/var**
- **/opt**
- **/etc/** 内の OpenShift Container Platform 固有の除外対象

### 6.5.5. カスタム AIDE 設定の指定

**DBDIR**、**LOGDIR**、**database**、および **database\_out** などの AIDE 内部動作を設定するエントリーは Operator によって上書きされます。Operator は、整合性に関する変更の有無についてすべてのパスを監視できるよう接頭辞を **/hostroot/** に追加します。これにより、コンテナ化された環境用にカスタマイズされない既存の AIDE 設定を再使用や、ルートディレクトリからの開始がより容易になります。



## 注記

`/hostroot` は、AIDE を実行する Pod がホストのファイルシステムをマウントするディレクトリです。設定を変更すると、データベースの再初期化がトリガーされます。

### 6.5.6. カスタム File Integrity Operator 設定の定義

この例では、**worker-fileintegrity** CR に提供されるデフォルト設定に基づいてコントロールプレーンノードで実行されるスキャナーのカスタム設定を定義することに重点を置いています。このワークフローは、デーモンセットとして実行されているカスタムソフトウェアをデプロイし、そのデータをコントロールプレーンノードの `/opt/mydaemon` の下に保存する場合に役立ちます。

#### 手順

1. デフォルト設定のコピーを作成します。
2. デフォルト設定を、監視するか、除外する必要があるファイルで編集します。
3. 編集したコンテンツを新たな設定マップに保存します。
4. **spec.config** の属性を使用して、**FileIntegrity** オブジェクトを新規の設定マップにポイントします。
5. デフォルト設定を抽出します。

```
$ oc extract cm/worker-fileintegrity --keys=aide.conf
```

これにより、編集可能な **aide.conf** という名前のファイルが作成されます。この例では、Operator のパスの後処理方法を説明するために、接頭辞なしで除外ディレクトリを追加します。

```
$ vim aide.conf
```

#### 出力例

```
/hostroot/etc/kubernetes/static-pod-resources
!/hostroot/etc/kubernetes/aide.*
!/hostroot/etc/kubernetes/manifests
!/hostroot/etc/docker/certs.d
!/hostroot/etc/selinux/targeted
!/hostroot/etc/openvswitch/conf.db
```

コントロールプレーンノードに固有のパスを除外します。

```
!/opt/mydaemon/
```

その他のコンテンツを **/etc** に保存します。

```
/hostroot/etc/ CONTENT_EX
```

6. このファイルに基づいて設定マップを作成します。

```
$ oc create cm master-aide-conf --from-file=aide.conf
```

## 7. 設定マップを参照する **FileIntegrity** CR マニフェストを定義します。

```

apiVersion: fileintegrity.openshift.io/v1alpha1
kind: FileIntegrity
metadata:
  name: master-fileintegrity
  namespace: openshift-file-integrity
spec:
  nodeSelector:
    node-role.kubernetes.io/master: ""
  config:
    name: master-aide-conf
    namespace: openshift-file-integrity

```

Operator は指定された設定マップファイルを処理し、結果を **FileIntegrity** オブジェクトと同じ名前の設定マップに保存します。

```
$ oc describe cm/master-fileintegrity | grep /opt/mydaemon
```

### 出力例

```
!/hostroot/opt/mydaemon
```

## 6.5.7. カスタムのファイル整合性設定の変更

ファイル整合性の設定を変更するには、生成される設定マップを変更しないでください。その代わりに、**spec.name**、**namespace**、および **key** 属性を使用して **FileIntegrity** オブジェクトにリンクされる設定マップを変更します。

## 6.6. 高度なカスタム FILE INTEGRITY OPERATOR タスクの実行

### 6.6.1. データベースの再初期化

File Integrity Operator が予定される変更を検知する場合、データベースの再初期化が必要になることがあります。

#### 手順

- **FileIntegrity** カスタムリソース (CR) に **file-integrity.openshift.io/re-init** のアノテーションを付けます。

```
$ oc annotate fileintegrities/worker-fileintegrity file-integrity.openshift.io/re-init=
```

古いデータベースとログファイルがバックアップされ、新しいデータベースが初期化されます。**oc debug** を使用して起動する Pod の以下の出力に示されるように、古いデータベースおよびログは **/etc/kubernetes** の下にあるノードに保持されます。

### 出力例

```

ls -lR /host/etc/kubernetes/aide.*
-rw-----. 1 root root 1839782 Sep 17 15:08 /host/etc/kubernetes/aide.db.gz
-rw-----. 1 root root 1839783 Sep 17 14:30 /host/etc/kubernetes/aide.db.gz.backup-

```

```

20200917T15_07_38
-rw-----. 1 root root 73728 Sep 17 15:07 /host/etc/kubernetes/aide.db.gz.backup-
20200917T15_07_55
-rw-r--r--. 1 root root 0 Sep 17 15:08 /host/etc/kubernetes/aide.log
-rw-----. 1 root root 613 Sep 17 15:07 /host/etc/kubernetes/aide.log.backup-
20200917T15_07_38
-rw-r--r--. 1 root root 0 Sep 17 15:07 /host/etc/kubernetes/aide.log.backup-
20200917T15_07_55

```

レコードの永続性を確保するために、生成される設定マップは **FileIntegrity** オブジェクトによって所有されません。そのため、手動のクリーンアップが必要になります。結果として、以前の整合性の失敗が依然として **FileIntegrityNodeStatus** オブジェクトに表示されます。

## 6.6.2. マシン設定の統合

OpenShift Container Platform 4 では、クラスターノード設定は **MachineConfig** オブジェクトで提供されます。**MachineConfig** オブジェクトによって生じるファイルへの変更が予想されますが、ファイルの整合性スキャンは失敗しないことを前提にすることができます。**MachineConfig** オブジェクトの更新によって生じるファイルの変更を抑制するために、File Integrity Operator はノードオブジェクトを監視します。ノードが更新されると、AIDE スキャンは更新の期間一時停止します。更新が完了すると、データベースが再初期化され、スキャンが再開されます。

この一時停止および再開ロジックは、ノードオブジェクトのアノテーションに反映されるため、**MachineConfig** API での更新にのみ適用されます。

## 6.6.3. デーモンセットの参照

それぞれの **FileIntegrity** オブジェクトはノード数についてのスキャンを表します。スキャン自体は、デーモンセットによって管理される Pod で実行されます。

**FileIntegrity** オブジェクトを表すデーモンセットを見つけるには、以下を実行します。

```
$ oc -n openshift-file-integrity get ds/aide-worker-fileintegrity
```

そのデーモンセットの Pod を一覧表示するには、以下を実行します。

```
$ oc -n openshift-file-integrity get pods -lapp=aide-worker-fileintegrity
```

単一の AIDE Pod のログを表示するには、Pod のいずれかで **oc logs** を呼び出します。

```
$ oc -n openshift-file-integrity logs pod/aide-worker-fileintegrity-mr8x6
```

## 出力例

```

Starting the AIDE runner daemon
initializing AIDE db
initialization finished
running aide check
...

```

AIDE デーモンによって作成された設定マップは保持されず、File Integrity Operator がこれらを処理した後には削除されます。ただし、障害およびエラーの発生時に、これらの設定マップの内容は **FileIntegrityNodeStatus** オブジェクトが参照する設定マップにコピーされます。

## 6.7. FILE INTEGRITY OPERATOR のトラブルシューティング

### 6.7.1. 一般的なトラブルシューティング

#### 問題

File Integrity Operator の問題をトラブルシューティングする必要がある場合があります。

#### 解決策

**FileIntegrity** オブジェクトでデフォルトフラグを有効にします。 **debug** フラグは、 **DaemonSet** Pod で実行されるデーモンの詳細度を上げ、AIDE チェックを実行します。

### 6.7.2. AIDE 設定の確認

#### 問題

AIDE 設定を確認する必要がある場合があります。

#### 解決策

AIDE 設定は、 **FileIntegrity** オブジェクトと同じ名前を設定マップに保存されます。すべての AIDE 設定の設定マップには、 **file-integrity.openshift.io/aide-conf** のラベルが付けられます。

### 6.7.3. FileIntegrity オブジェクトのフェーズの判別

#### 問題

**FileIntegrity** オブジェクトが存在するかどうかを判別し、その現在のステータスを確認する必要があります。

#### 解決策

**FileIntegrity** オブジェクトの現在のステータスを確認するには、以下を実行します。

```
$ oc get fileintegrities/worker-fileintegrity -o jsonpath="{.status}"
```

**FileIntegrity** オブジェクトおよびサポートするデーモンセットが作成されると、ステータスは **Active** に切り替わります。切り替わらない場合は、Operator Pod ログを確認してください。

### 6.7.4. デーモンセットの Pod が予想されるノードで実行されていることの判別

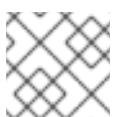
#### 問題

デーモンセットが存在し、その Pod が実行されることが予想されるノードで実行されていることを確認する必要がある場合があります。

#### 解決策

以下を実行します。

```
$ oc -n openshift-file-integrity get pods -lapp=aide-worker-fileintegrity
```



#### 注記

**-owide** を追加すると、Pod が実行されているノードの IP アドレスが含まれます。

デーモン Pod のログを確認するには、 **oc logs** を実行します。

AIDE コマンドの戻り値をチェックして、チェックが合否を確認します。

## 第7章 監査ログの表示

OpenShift Container Platform 監査は、システムに影響を与えた一連のアクティビティーを個別のユーザー、管理者その他システムのコンポーネント別に記述したセキュリティー関連の時系列のレコードを提供します。

### 7.1. API の監査ログについて

監査は API サーバーレベルで実行され、サーバーに送られるすべての要求をログに記録します。それぞれの監査ログには、以下の情報が含まれます。

表7.1 監査ログフィールド

フィールド	説明
<b>level</b>	イベントが生成された監査レベル。
<b>auditID</b>	要求ごとに生成される一意の監査 ID。
<b>stage</b>	このイベントインスタンスの生成時の要求処理のステージ。
<b>requestURI</b>	クライアントによってサーバーに送信される要求 URI。
<b>verb</b>	要求に関連付けられる Kubernetes の動詞。リソース以外の要求の場合、これは小文字の HTTP メソッドになります。
<b>user</b>	認証されたユーザーの情報。
<b>impersonatedUser</b>	オプション。偽装ユーザーの情報 (要求で別のユーザーを偽装する場合)。
<b>sourceIPs</b>	オプション。要求の送信元および中間プロキシからのソース IP。
<b>userAgent</b>	オプション。クライアントが報告するユーザーエージェントの文字列。ユーザーエージェントはクライアントによって提供されており、信頼できないことに注意してください。
<b>objectRef</b>	オプション。この要求のターゲットとなっているオブジェクト参照。これは、 <b>List</b> タイプの要求やリソース以外の要求には適用されません。
<b>responseStatus</b>	オプション。 <b>ResponseObject</b> が <b>Status</b> タイプでなくても設定される応答ステータス。正常な応答の場合、これにはコードのみが含まれます。ステータス以外のタイプのエラー応答の場合、これにはエラーメッセージが自動的に設定されます。



フィールド	説明
<b>requestObject</b>	オプション。JSON形式の要求からのAPIオブジェクト。 <b>RequestObject</b> は、バージョンの変換、デフォルト設定、受付またはマージの前に要求の場合のように記録されます(JSONとして再エンコードされる可能性がある)。これは外部のバージョン付けされたオブジェクトタイプであり、それ自体では有効なオブジェクトではない可能性があります。これはリソース以外の要求の場合には省略され、要求レベル以上でのみログに記録されます。
<b>responseObject</b>	オプション。JSON形式の応答で返されるAPIオブジェクト。 <b>ResponseObject</b> は外部タイプへの変換後に記録され、JSONとしてシリアライズされます。これはリソース以外の要求の場合には省略され、応答レベルでのみログに記録されます。
<b>requestReceivedTimestamp</b>	要求がAPIサーバーに到達した時間。
<b>stageTimestamp</b>	要求が現在の監査ステージに達した時間。
<b>annotations</b>	オプション。監査イベントと共に保存される構造化されていないキーと値のマップ。これは、認証、認可、受付プラグインなど、要求提供チェーンで呼び出されるプラグインによって設定される可能性があります。これらのアノテーションは監査イベント用のもので、送信されたオブジェクトの <b>metadata.annotations</b> に対応しないことに注意してください。キーは、名前の競合が発生しないように通知コンポーネントを一意に識別する必要があります(例: <b>podsecuritypolicy.admission.k8s.io/policy</b> )。値は短くする必要があります。アノテーションはメタデータレベルに含まれます。

Kubernetes API サーバーの出力例:

```
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "Metadata",
  "auditID": "ad209ce1-fec7-4130-8192-c4cc63f1d8cd",
  "stage": "ResponseComplete",
  "requestURI": "/api/v1/namespaces/openshift-kube-controller-manager/configmaps/cert-recovery-controller-lock?timeout=35s",
  "verb": "update",
  "user": {
    "username": "system:serviceaccount:openshift-kube-controller-manager:localhost-recovery-client",
    "uid": "dd4997e3-d565-4e37-80f8-7fc122ccd785",
    "groups": [
      "system:serviceaccounts",
      "system:serviceaccounts:openshift-kube-controller-manager",
      "system:authenticated"
    ],
    "sourceIPs": [
      "::1"
    ],
    "userAgent": "cluster-kube-controller-manager-operator/v0.0.0 (linux/amd64) kubernetes/$Format",
    "objectRef": {
      "resource": "configmaps",
      "namespace": "openshift-kube-controller-manager",
      "name": "cert-recovery-controller-lock",
      "uid": "5c57190b-6993-425d-8101-8337e48c7548",
      "apiVersion": "v1",
      "resourceVersion": "574307"
    },
    "responseStatus": {
      "metadata": {},
      "code": 200,
      "requestReceivedTimestamp": "2020-04-02T08:27:20.200962Z",
      "stageTimestamp": "2020-04-02T08:27:20.206710Z",
      "annotations": {
        "authorization.k8s.io/decision": "allow",
        "authorization.k8s.io/reason": "RBAC: allowed by ClusterRoleBinding \"system:openshift:operator:kube-controller-manager-recovery\" of ClusterRole \"cluster-admin\" to ServiceAccount \"localhost-recovery-client/openshift-kube-controller-manager\""
      }
    }
  }
}
```

## 7.2. 監査ログの表示

それぞれのコントロールプレーンノードについて OpenShift API サーバー、Kubernetes API サーバー、および OpenShift OAuth API サーバーのログを表示することができます。

## 手順

監査ログを表示するには、以下を実行します。

- OpenShift API サーバーログを表示します。
  - a. 各コントロールプレーンノードで利用可能な OpenShift API サーバーログを一覧表示します。

```
$ oc adm node-logs --role=master --path=openshift-apiserver/
```

### 出力例

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T00-12-19.834.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T00-11-49.835.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T00-13-00.128.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

- b. ノード名とログ名を指定して、特定の OpenShift API サーバーログを表示します。

```
$ oc adm node-logs <node_name> --path=openshift-apiserver/<log_name>
```

以下に例を示します。

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=openshift-apiserver/audit-2021-03-09T00-12-19.834.log
```

### 出力例

```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"381acf6d-5f30-4c7d-8175-c9c317ae5893","stage":"ResponseComplete","requestURI":"/metrics","verb":"get","user":{"username":"system:serviceaccount:openshift-monitoring:prometheus-k8s","uid":"825b60a0-3976-4861-a342-3b2b561e8f82","groups":["system:serviceaccounts","system:serviceaccounts:openshift-monitoring","system:authenticated"],"sourceIPs":["10.129.2.6"],"userAgent":"Prometheus/2.23.0","responseStatus":{"metadata":{"code":200},"requestReceivedTimestamp":"2021-03-08T18:02:04.086545Z","stageTimestamp":"2021-03-08T18:02:04.107102Z"},"annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"prometheus-k8s\" of ClusterRole \"prometheus-k8s\" to ServiceAccount \"prometheus-k8s/openshift-monitoring\"\"}}
```

- Kubernetes API サーバーログを表示します。
  - a. 各コントロールプレーンノードで利用可能な Kubernetes API サーバーログを一覧表示します。

```
$ oc adm node-logs --role=master --path=kube-apiserver/
```

### 出力例

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T14-07-27.129.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T19-24-22.620.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T18-37-07.511.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

- b. ノード名とログ名を指定して、特定の Kubernetes API サーバーログを表示します。

```
$ oc adm node-logs <node_name> --path=kube-apiserver/<log_name>
```

以下に例を示します。

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=kube-apiserver/audit-
2021-03-09T14-07-27.129.log
```

### 出力例

```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"cfce8a0b-b5f5-
4365-8c9f-
79c1227d10f9","stage":"ResponseComplete","requestURI":"/api/v1/namespaces/openshift-
kube-scheduler/serviceaccounts/openshift-kube-scheduler-sa","verb":"get","user":
{"username":"system:serviceaccount:openshift-kube-scheduler-operator:openshift-kube-
scheduler-operator","uid":"2574b041-f3c8-44e6-a057-baef7aa81516","groups":
["system:serviceaccounts","system:serviceaccounts:openshift-kube-scheduler-
operator","system:authenticated"],"sourceIPs":["10.128.0.8"],"userAgent":"cluster-kube-
scheduler-operator/v0.0.0 (linux/amd64) kubernetes/$Format","objectRef":
{"resource":"serviceaccounts","namespace":"openshift-kube-
scheduler","name":"openshift-kube-scheduler-sa","apiVersion":"v1"},"responseStatus":
{"metadata":{},"code":200,"requestReceivedTimestamp":"2021-03-
08T18:06:42.512619Z","stageTimestamp":"2021-03-
08T18:06:42.516145Z","annotations":{"authentication.k8s.io/legacy-
token":"system:serviceaccount:openshift-kube-scheduler-operator:openshift-kube-
scheduler-
operator","authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC:
allowed by ClusterRoleBinding \"system:openshift:operator:cluster-kube-scheduler-
operator\" of ClusterRole \"cluster-admin\" to ServiceAccount \"openshift-kube-scheduler-
operator/openshift-kube-scheduler-operator\""}}}
```

- OpenShift OAuth API サーバーログを表示します。
  - a. 各コントロールプレーンノードで利用可能な OpenShift OAuth API サーバーログを一覧表示します。

```
$ oc adm node-logs --role=master --path=oauth-apiserver/
```

### 出力例

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T13-06-26.128.log
```

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T18-23-21.619.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T17-36-06.510.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

- b. ノード名とログ名を指定して、特定の OpenShift OAuth API サーバーログを表示します。

```
$ oc adm node-logs <node_name> --path=oauth-apiserver/<log_name>
```

以下に例を示します。

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=oauth-apiserver/audit-2021-03-09T13-06-26.128.log
```

### 出力例

```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"dd4c44e2-3ea1-4830-9ab7-c91a5f1388d6","stage":"ResponseComplete","requestURI":"/apis/user.openshift.io/v1/users/~","verb":"get","user":{"username":"system:serviceaccount:openshift-monitoring:prometheus-k8s","groups":["system:serviceaccounts","system:serviceaccounts:openshift-monitoring","system:authenticated"],"sourceIPs":["10.0.32.4","10.128.0.1"],"userAgent":"dockerregistry/v0.0.0 (linux/amd64) kubernetes/$Format","objectRef":{"resource":"users","name":"~","apiGroup":"user.openshift.io","apiVersion":"v1"},"responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2021-03-08T17:47:43.653187Z","stageTimestamp":"2021-03-08T17:47:43.660187Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"basic-users\" of ClusterRole \"basic-user\" to Group \"system:authenticated\"}}}
```

## 7.3. 監査ログのフィルター

**jq** または別の JSON 解析ツールを使用して、API サーバー監査ログをフィルターできます。



### 注記

API サーバー監査ログに記録する情報量は、設定される監査ログポリシーで制御できません。

以下の手順では、**jq** を使用してコントロールプレーンノード **node-1.example.com** で監査ログをフィルターする例を示します。**jq** の使用に関する詳細は、[jq Manual](#) を参照してください。

### 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- **jq** がインストールされている。

## 手順

- OpenShift API サーバー監査ログをユーザーでフィルターします。

```
$ oc adm node-logs node-1.example.com \
  --path=openshift-apiserver/audit.log \
  | jq 'select(.user.username == "myusername")'
```

- OpenShift API サーバー監査ログをユーザーエージェントでフィルターします。

```
$ oc adm node-logs node-1.example.com \
  --path=openshift-apiserver/audit.log \
  | jq 'select(.userAgent == "cluster-version-operator/v0.0.0 (linux/amd64)
  kubernetes/$Format")'
```

- Kubernetes API サーバー監査ログを特定の API バージョンでフィルターし、ユーザーエージェントのみを出力します。

```
$ oc adm node-logs node-1.example.com \
  --path=kube-apiserver/audit.log \
  | jq 'select(.requestURI | startswith("/apis/apiextensions.k8s.io/v1beta1")) | .userAgent'
```

- 動詞を除外して OpenShift OAuth API サーバー監査ログをフィルターします。

```
$ oc adm node-logs node-1.example.com \
  --path=oauth-apiserver/audit.log \
  | jq 'select(.verb != "get")'
```

## 7.4. 監査ログの収集

must-gather ツールを使用して、クラスターをデバッグするための監査ログを収集できます。このログは、確認したり、Red Hat サポートに送信したりできます。

### 手順

1. `-- /usr/bin/gather_audit_logs` フラグを使用して `oc adm must-gather` コマンドを実行します。

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```

2. 作業ディレクトリーに作成された `must-gather` ディレクトリーから圧縮ファイルを作成します。たとえば、Linux オペレーティングシステムを使用するコンピューターで以下のコマンドを実行します。

```
$ tar cvaf must-gather.tar.gz must-gather.local.472290403699006248 1
```

- 1** `must-gather-local.472290403699006248` は、実際のディレクトリー名に置き換えます。

3. 圧縮ファイルを [Red Hat カスタマーポータル](#) で作成したサポートケースに添付します。

## 7.5. 関連情報

- [must-gather ツール](#)
- [API 監査ログイベントの構造](#)
- [監査ログポリシーの設定](#)
- [ログのサードパーティーシステムへの転送](#)

## 第8章 監査ログポリシーの設定

使用する監査ログポリシープロファイルを選択して、API サーバー監査ログに記録する情報量を制御できます。

### 8.1. 監査ログポリシープロファイルについて

監査ログプロファイルは、OpenShift API サーバー、Kubernetes API サーバー、および OAuth API サーバーに送信される要求をログに記録する方法を定義します。

OpenShift Container Platform は、以下の事前定義された監査ポリシープロファイルを提供します。

プロファイル	説明
デフォルト	読み取りおよび書き込み要求のメタデータのみをログに記録します。OAuth アクセストークン要求を除く要求の本文はログに記録されません。これはデフォルトポリシーになります。
WriteRequestBodies	すべての要求のメタデータをロギングする以外にも、API サーバーへの書き込み要求ごとに要求の本文をログに記録します ( <b>create</b> 、 <b>update</b> 、 <b>patch</b> )。このプロファイルには、 <b>Default</b> プロファイルよりも多くのリソースのオーバーヘッドがあります。 <sup>[1]</sup>
AllRequestBodies	すべての要求のメタデータをロギングする以外にも、API サーバーへの読み取りおよび書き込み要求ごとに要求の本文をログに記録します ( <b>get</b> 、 <b>list</b> 、 <b>create</b> 、 <b>update</b> 、 <b>patch</b> )。このプロファイルには最も多くのリソースのオーバーヘッドがあります。 <sup>[1]</sup>
なし	<p>要求はログに記録されません (OAuth アクセストークン要求および OAuth 認証トークン要求でさえもログへの記録なし)。</p> <div style="border: 1px solid #ccc; padding: 10px; background-color: #fff9c4;"> <p><b>警告</b></p> <p> 問題のトラブルシューティング時に有用なデータが記録されないリスクを完全に理解していない限り、<b>None</b> プロファイルを使用して監査ロギングを無効にすることは推奨していません。監査ロギングを無効にしてサポートが必要な状況が生じた場合は、適切にトラブルシューティングを行うために監査ロギングを有効にし、問題を再現する必要がある場合があります。</p> </div>

1. **Secret**、**Route**、**OAuthClient** オブジェクトなどの機密リソースは、メタデータレベルを超えるとログに記録されません。

デフォルトで、OpenShift Container Platform は **Default** 監査ログプロファイルを使用します。要求の本文もログに記録する別の監査ポリシープロファイルを使用できますが、リソース使用の増加について把握するようにしてください (CPU、メモリー、および I/O)。

## 8.2. 監査ログポリシーの設定

API サーバーに送信される要求をログに記録する際に使用する監査ログポリシーを設定できます。

### 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

### 手順

1. **APIServer** リソースを編集します。

```
$ oc edit apiserver cluster
```

2. **spec.audit.profile** フィールドを更新します。

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  ...
spec:
  audit:
    profile: WriteRequestBodies ①
```

- ① **Default**、**WriteRequestBodies**、**AllRequestBodies** または **None** に設定されます。デフォルトのプロファイルは **Default** です。



### 警告

問題のトラブルシューティング時に有用なデータが記録されないリスクを完全に理解していない限り、**None** プロファイルを使用して監査ロギングを無効にすることは推奨していません。監査ロギングを無効にしてサポートが必要な状況が生じた場合は、適切にトラブルシューティングを行うために監査ロギングを有効にし、問題を再現する必要がある場合があります。

3. 変更を適用するためにファイルを保存します。

### 検証

- Kubernetes API サーバー Pod の新規リビジョンがロールアウトされていることを確認します。すべてのノードが新規リビジョンに更新されるまで数分かかる場合があります。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```



Kubernetes API サーバーの **NodeInstallerProgressing** 状況条件を確認し、すべてのノードが最新のレビジョンであることを確認します。更新が正常に実行されると、この出力には **AllNodesAtLatestRevision** が表示されます。

```
AllNodesAtLatestRevision
3 nodes are at revision 12 ❶
```

- ❶ この例では、最新のレビジョン番号は **12** です。

出力に以下のようなメッセージが表示される場合は、更新が進行中です。数分待機した後に再試行します。

- **3 nodes are at revision 11; 0 nodes have achieved new revision 12**
- **2 nodes are at revision 11; 1 nodes are at revision 12**

### 8.3. カスタムルールによる監査ログポリシーの設定

カスタムルールを定義する監査ログポリシーを設定できます。複数のグループを指定し、対象のグループに使用するプロファイルを定義できます。

これらのカスタムルールは最上位のプロファイルフィールドよりも優先されます。カスタムルールはトップダウンで評価され、最初に一致するものが適用されます。

#### 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

#### 手順

1. **APIServer** リソースを編集します。

```
$ oc edit apiserver cluster
```

2. **spec.audit.customRules** フィールドを追加します。

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  ...
spec:
  audit:
    customRules: ❶
    - group: system:authenticated:oauth
      profile: WriteRequestBodies
    - group: system:authenticated
      profile: AllRequestBodies
      profile: Default ❷
```

- ❶ 1つ以上のグループを追加し、そのグループに使用するプロファイルを指定します。これらのカスタムルールは最上位のプロファイルフィールドよりも優先されます。カスタムルールはトップダウンで評価され、最初に一致するものが適用されます。

- 2 **Default**、**WriteRequestBodies**、**AllRequestBodies** または **None** に設定されます。この最上位の **audit.profile** フィールドを設定しない場合には、デフォルトは **Default** プロファイルに設定されます。



### 警告

問題のトラブルシューティング時に有用なデータが記録されないリスクを完全に理解していない限り、**None** プロファイルを使用して監査ロギングを無効にすることは推奨していません。監査ロギングを無効にしてサポートが必要な状況が生じた場合は、適切にトラブルシューティングを行うために監査ロギングを有効にし、問題を再現する必要がある場合があります。

3. 変更を適用するためにファイルを保存します。

### 検証

- Kubernetes API サーバー Pod の新規リビジョンがロールアウトされていることを確認します。すべてのノードが新規リビジョンに更新されるまで数分かかる場合があります。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

Kubernetes API サーバーの **NodeInstallerProgressing** 状況条件を確認し、すべてのノードが最新のリビジョンであることを確認します。更新が正常に実行されると、この出力には **AllNodesAtLatestRevision** が表示されます。

```
AllNodesAtLatestRevision
3 nodes are at revision 12 1
```

- 1 この例では、最新のリビジョン番号は **12** です。

出力に以下のようなメッセージが表示される場合は、更新が進行中です。数分待機した後に再試行します。

- **3 nodes are at revision 11; 0 nodes have achieved new revision 12**
- **2 nodes are at revision 11; 1 nodes are at revision 12**

## 8.4. 監査ロギングの無効化

OpenShift Container Platform の監査ロギングを無効にできます。監査ロギングを無効にする場合には、OAuth アクセストークン要求および OAuth 認証トークン要求もログに記録されません。



### 警告

問題のトラブルシューティング時に有用なデータが記録されないリスクを完全に理解していない限り、**None** プロファイルを使用して監査ロギングを無効にすることは推奨していません。監査ロギングを無効にしてサポートが必要な状況が生じた場合は、適切にトラブルシューティングを行うために監査ロギングを有効にし、問題を再現する必要がある場合があります。

### 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

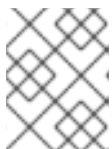
### 手順

1. **APIServer** リソースを編集します。

```
$ oc edit apiserver cluster
```

2. **spec.audit.profile** フィールドを **None** に設定します。

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  ...
spec:
  audit:
    profile: None
```



### 注記

また、**spec.audit.customRules** フィールドにカスタムルールを指定して、特定のグループについての監査ロギングのみを無効にすることもできます。

3. 変更を適用するためにファイルを保存します。

### 検証

- Kubernetes API サーバー Pod の新規リビジョンがロールアウトされていることを確認します。すべてのノードが新規リビジョンに更新されるまで数分かかる場合があります。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

Kubernetes API サーバーの **NodeInstallerProgressing** 状況条件を確認し、すべてのノードが最新のリビジョンであることを確認します。更新が正常に実行されると、この出力には **AllNodesAtLatestRevision** が表示されます。

```
AllNodesAtLatestRevision
3 nodes are at revision 12 1
```

-

- 1 この例では、最新のリビジョン番号は **12** です。

出力に以下のようなメッセージが表示される場合は、更新が進行中です。数分待機した後に再試行します。

- **3 nodes are at revision 11; 0 nodes have achieved new revision 12**
- **2 nodes are at revision 11; 1 nodes are at revision 12**

## 第9章 TLS セキュリティープロファイルの設定

TLS セキュリティープロファイルは、サーバーへの接続時に、クライアントが使用できる暗号を規制する方法をサーバーに提供します。これにより、OpenShift Container Platform コンポーネントは暗号化ライブラリーを使用するようになり、既知の安全ではないプロトコル、暗号、またはアルゴリズムを拒否します。

クラスター管理者は、以下のコンポーネントごとに使用する TLS セキュリティープロファイルを選択できます。

- Ingress コントローラー
- コントロールプレーン  
これには、Kubernetes API サーバー、Kubernetes コントローラーマネージャー、Kubernetes スケジューラー、OpenShift API サーバー、OpenShift OAuth サーバー および etcd が含まれます。
- kubelet (Kubernetes API サーバーの HTTP サーバーとして機能する場合)

### 9.1. TLS セキュリティープロファイルについて

TLS (Transport Layer Security) セキュリティープロファイルを使用して、さまざまな OpenShift Container Platform コンポーネントに必要な TLS 暗号を定義できます。OpenShift Container Platform の TLS セキュリティープロファイルは、[Mozilla が推奨する設定](#) に基づいています。

コンポーネントごとに、以下の TLS セキュリティープロファイルのいずれかを指定できます。

表9.1 TLS セキュリティープロファイル

プロファイル	説明
<b>Old</b>	<p>このプロファイルは、レガシークライアントまたはライブラリーでの使用を目的としています。このプロファイルは、<a href="#">Old 後方互換性</a> の推奨設定に基づいています。</p> <p><b>Old</b> プロファイルには、最小 TLS バージョン 1.0 が必要です。</p> <div style="display: flex; align-items: center;">  <p><b>注記</b></p> </div> <p>Ingress コントローラーの場合、TLS の最小バージョンは 1.0 から 1.1 に変換されます。</p>
<b>Intermediate</b>	<p>このプロファイルは、大多数のクライアントに推奨される設定です。これは、Ingress コントローラー、kubelet、およびコントロールプレーンのデフォルトの TLS セキュリティープロファイルです。このプロファイルは、<a href="#">Intermediate 互換性</a> の推奨設定に基づいています。</p> <p><b>Intermediate</b> プロファイルには、最小 TLS バージョン 1.2 が必要です。</p>

プロファイル	説明
<b>Modern</b>	<p>このプロファイルは、後方互換性を必要としない Modern のクライアントでの使用を目的としています。このプロファイルは、<a href="#">Modern 互換性</a>の推奨設定に基づいています。</p> <p><b>Modern</b> プロファイルには、最小 TLS バージョン 1.3 が必要です。</p>
<b>カスタム</b>	<p>このプロファイルを使用すると、使用する TLS バージョンと暗号を定義できます。</p> <div style="border: 1px solid #ccc; background-color: #fff9c4; padding: 10px; margin-top: 10px;"> <p style="text-align: center;"> <b>警告</b></p> <p>無効な設定により問題が発生する可能性があるため、<b>Custom</b> プロファイルを使用する際には注意してください。</p> </div>



### 注記

事前定義されたプロファイルタイプのいずれかを使用する場合、有効なプロファイル設定はリリース間で変更される可能性があります。たとえば、リリース X.Y.Z にデプロイされた Intermediate プロファイルを使用する仕様がある場合、リリース X.Y.Z+1 へのアップグレードにより、新規のプロファイル設定が適用され、ロールアウトが生じる可能性があります。

## 9.2. TLS セキュリティープロファイルの詳細表示

Ingress コントローラー、コントロールプレーン、および kubelet のコンポーネントごとに事前定義された TLS セキュリティープロファイルの最小 TLS バージョンおよび暗号を表示できます。



### 重要

プロファイルの最小 TLS バージョンと暗号の一覧は、コンポーネントによって異なる場合があります。

### 手順

- 特定の TLS セキュリティープロファイルの詳細を表示します。

```
$ oc explain <component>.spec.tlsSecurityProfile.<profile> 1
```

- 1** **<component>** には、**ingresscontroller**、**apiserver** または **kubeletconfig** を指定します。**<profile>** には、**old**、**intermediate** または **custom** を指定します。

たとえば、コントロールプレーンの **intermediate** プロファイルに含まれる暗号を確認するには、以下を実行します。

```
$ oc explain apiserver.spec.tlsSecurityProfile.intermediate
```

## 出力例

```
KIND: APIServer
VERSION: config.openshift.io/v1

DESCRIPTION:
  intermediate is a TLS security profile based on:

  https://wiki.mozilla.org/Security/Server_Side_TLS#Intermediate_compatibility_.28recommended_.29
  and looks like this (yaml):
  ciphers: - TLS_AES_128_GCM_SHA256 - TLS_AES_256_GCM_SHA384 -
  TLS_CHACHA20_POLY1305_SHA256 - ECDHE-ECDSA-AES128-GCM-SHA256 -
  ECDHE-RSA-AES128-GCM-SHA256 - ECDHE-ECDSA-AES256-GCM-SHA384 -
  ECDHE-RSA-AES256-GCM-SHA384 - ECDHE-ECDSA-CHACHA20-POLY1305 -
  ECDHE-RSA-CHACHA20-POLY1305 - DHE-RSA-AES128-GCM-SHA256 -
  DHE-RSA-AES256-GCM-SHA384 minTLSVersion: TLSv1.2
```

- コンポーネントの **tlsSecurityProfile** フィールドの詳細をすべて表示します。

```
$ oc explain <component>.spec.tlsSecurityProfile 1
```

- 1** **<component>** には、**ingresscontroller**、**apiserver** または **kubeletconfig** を指定します。

たとえば、Ingress コントローラーの **tlsSecurityProfile** フィールドの詳細をすべて確認するには、以下を実行します。

```
$ oc explain ingresscontroller.spec.tlsSecurityProfile
```

## 出力例

```
KIND: IngressController
VERSION: operator.openshift.io/v1

RESOURCE: tlsSecurityProfile <Object>

DESCRIPTION:
  ...

FIELDS:
  custom <>
    custom is a user-defined TLS security profile. Be extremely careful using a
    custom profile as invalid configurations can be catastrophic. An example
    custom profile looks like this:
    ciphers: - ECDHE-ECDSA-CHACHA20-POLY1305 - ECDHE-RSA-CHACHA20-
    POLY1305 -
    ECDHE-RSA-AES128-GCM-SHA256 - ECDHE-ECDSA-AES128-GCM-SHA256
  minTLSVersion:
    TLSv1.1
```

```

intermediate <>
intermediate is a TLS security profile based on:

https://wiki.mozilla.org/Security/Server_Side_TLS#Intermediate_compatibility_.28recommended.29
and looks like this (yaml):
... 1

modern <>
modern is a TLS security profile based on:
https://wiki.mozilla.org/Security/Server_Side_TLS#Modern_compatibility_and_looks_like_this_(yaml):
... 2
NOTE: Currently unsupported.

old <>
old is a TLS security profile based on:
https://wiki.mozilla.org/Security/Server_Side_TLS#Old_backward_compatibility_and_looks_like_this_(yaml):
... 3

type <string>
...

```

- 1 ここでは、**intermediate** プロファイルの暗号および最小バージョンを一覧表示します。
- 2 **modern** プロファイルの暗号と最小バージョンを一覧表示します。
- 3 ここでは、**old** プロファイルの暗号および最小バージョンを一覧表示します。

### 9.3. INGRESS コントローラーの TLS セキュリティープロファイルの設定

Ingress コントローラーの TLS セキュリティープロファイルを設定するには、**IngressController** カスタムリソース (CR) を編集して、事前定義済みまたはカスタムの TLS セキュリティープロファイルを指定します。TLS セキュリティープロファイルが設定されていない場合、デフォルト値は API サーバーに設定された TLS セキュリティープロファイルに基づいています。

#### Old TLS のセキュリティプロファイルを設定するサンプル IngressController CR

```

apiVersion: operator.openshift.io/v1
kind: IngressController
...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
...

```

TLS セキュリティープロファイルは、Ingress コントローラーの TLS 接続の最小 TLS バージョンと TLS 暗号を定義します。

設定された TLS セキュリティープロファイルの暗号と最小 TLS バージョンは、**Status.Tls Profile** 配下の **IngressController** カスタムリソース (CR) と **Spec.Tls Security Profile** 配下の設定された TLS セキュリティープロファイルで確認できます。**Custom** TLS セキュリティープロファイルの場合、特定の



暗号と最小 TLS バージョンは両方のパラメーターの下に一覧表示されます。



## 注記

HAProxy Ingress Controller イメージは、TLS1.3 と **Modern** プロファイルをサポートしています。

また、Ingress Operator は TLS 1.0 の **Old** または **Custom** プロファイルを 1.1 に変換します。

## 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

## 手順

1. **openshift-ingress-operator** プロジェクトの **IngressController** CR を編集して、TLS セキュリティープロファイルを設定します。

```
$ oc edit IngressController default -n openshift-ingress-operator
```

2. **spec.tlsSecurityProfile** フィールドを追加します。

### Custom プロファイルのサンプル IngressController CR

```
apiVersion: operator.openshift.io/v1
kind: IngressController
...
spec:
  tlsSecurityProfile:
    type: Custom ①
    custom: ②
    ciphers: ③
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
    minTLSVersion: VersionTLS11
...

```

① TLS セキュリティープロファイルタイプ (**Old**、**Intermediate**、または **Custom**) を指定します。デフォルトは **Intermediate** です。

② 選択したタイプに適切なフィールドを指定します。

- **old:** {}
- **intermediate:** {}
- **custom:**

③ **custom** タイプには、TLS 暗号の一覧と最小許容 TLS バージョンを指定します。

3. 変更を適用するためにファイルを保存します。

## 検証

- **IngressController** CR にプロファイルが設定されていることを確認します。

```
$ oc describe IngressController default -n openshift-ingress-operator
```

## 出力例

```
Name:      default
Namespace: openshift-ingress-operator
Labels:    <none>
Annotations: <none>
API Version: operator.openshift.io/v1
Kind:      IngressController
...
Spec:
...
Tls Security Profile:
  Custom:
    Ciphers:
      ECDHE-ECDSA-CHACHA20-POLY1305
      ECDHE-RSA-CHACHA20-POLY1305
      ECDHE-RSA-AES128-GCM-SHA256
      ECDHE-ECDSA-AES128-GCM-SHA256
    Min TLS Version: VersionTLS11
  Type:      Custom
...
```

## 9.4. コントロールプレーンの TLS セキュリティープロファイルの設定

コントロールプレーンの TLS セキュリティープロファイルを設定するには、**APIServer** カスタムリソース (CR) を編集して、事前定義済みまたはカスタムの TLS セキュリティープロファイルを指定します。**APIServer** CR に TLS セキュリティープロファイルを設定すると、設定は以下のコントロールプレーンのコンポーネントに伝播されます。

- Kubernetes API サーバー
- Kubernetes コントローラーマネージャー
- Kubernetes スケジューラー
- OpenShift API サーバー
- OpenShift OAuth API サーバー
- OpenShift OAuth サーバー
- etcd

TLS セキュリティープロファイルが設定されていない場合には、TLS セキュリティープロファイルは **Intermediate** になります。



## 注記

Ingress コントローラーのデフォルトの TLS セキュリティープロファイルは API サーバーの TLS セキュリティープロファイルに基づいています。

## Old TLS のセキュリティープロファイルを設定するサンプル APIServer CR

```

apiVersion: config.openshift.io/v1
kind: APIServer
...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
  ...

```

TLS セキュリティープロファイルは、コントロールプレーンのコンポーネントとの通信に必要な TLS の最小バージョンと TLS 暗号を定義します。

設定した TLS セキュリティープロファイルは、**Spec.Tls Security Profile** の **APIServer** カスタムリソース (CR) で確認できます。**Custom** TLS セキュリティープロファイルの場合には、特定の暗号と最小 TLS バージョンが一覧表示されます。



## 注記

コントロールプレーンは、最小 TLS バージョンとして TLS **1.3** をサポートしません。**Modern** プロファイルは TLS **1.3** を必要とするため、サポート対象外です。

## 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

## 手順

1. デフォルトの **APIServer** CR を編集して TLS セキュリティープロファイルを設定します。

```
$ oc edit APIServer cluster
```

2. **spec.tlsSecurityProfile** フィールドを追加します。

## Custom プロファイルの APIServer CR のサンプル

```

apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  tlsSecurityProfile:
    type: Custom 1
    custom: 2
      ciphers: 3
        - ECDHE-ECDSA-CHACHA20-POLY1305
        - ECDHE-RSA-CHACHA20-POLY1305

```

```
- ECDHE-RSA-AES128-GCM-SHA256
- ECDHE-ECDSA-AES128-GCM-SHA256
minTLSVersion: VersionTLS11
```

- 1 TLS セキュリティープロファイルタイプ (**Old**、**Intermediate**、または **Custom**) を指定します。デフォルトは **Intermediate** です。
- 2 選択したタイプに適切なフィールドを指定します。
  - **old:** {}
  - **intermediate:** {}
  - **custom:**
- 3 **custom** タイプには、TLS 暗号の一覧と最小許容 TLS バージョンを指定します。

3. 変更を適用するためにファイルを保存します。

## 検証

- TLS セキュリティープロファイルが **APIServer** CR に設定されていることを確認します。

```
$ oc describe apiserver cluster
```

## 出力例

```
Name:      cluster
Namespace:
...
API Version: config.openshift.io/v1
Kind:      APIServer
...
Spec:
  Audit:
    Profile: Default
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
        ECDHE-RSA-CHACHA20-POLY1305
        ECDHE-RSA-AES128-GCM-SHA256
        ECDHE-ECDSA-AES128-GCM-SHA256
      Min TLS Version: VersionTLS11
    Type:      Custom
...

```

- TLS セキュリティープロファイルが **etcd** CR に設定されていることを確認します。

```
$ oc describe etcd cluster
```

## 出力例

```

Name:      cluster
Namespace:
...
API Version: operator.openshift.io/v1
Kind:      Etcd
...
Spec:
  Log Level:      Normal
  Management State: Managed
  Observed Config:
    Serving Info:
      Cipher Suites:
        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_CHACHA20_POLY1305_SHA256
        TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
      Min TLS Version:      VersionTLS12
    ...

```

## 9.5. KUBELET の TLS セキュリティープロファイルの設定

HTTP サーバーとしての動作時に kubelet の TLS セキュリティープロファイルを設定するには、**KubeletConfig** カスタムリソース (CR) を作成して特定のノード用に事前定義済みの TLS セキュリティープロファイルまたはカスタム TLS セキュリティープロファイルを指定します。TLS セキュリティープロファイルが設定されていない場合には、TLS セキュリティープロファイルは **Intermediate** になります。

kubelet はその HTTP/GRPC サーバーを使用して Kubernetes API サーバーと通信し、コマンドを Pod に送信して kubelet 経由で Pod で exec コマンドを実行します。

### ワーカーノードで Old TLS セキュリティープロファイルを設定する KubeletConfig CR のサンプル

```

apiVersion: config.openshift.io/v1
kind: KubeletConfig
...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""

```

設定済みのノードの **kubelet.conf** ファイルで、設定済みの TLS セキュリティープロファイルの暗号化および最小 TLS セキュリティープロファイルを確認できます。

#### 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

#### 手順

1. **KubeletConfig** CR を作成し、TLS セキュリティープロファイルを設定します。

### カスタム プロファイルの KubeletConfig CR のサンプル

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-kubelet-tls-security-profile
spec:
  tlsSecurityProfile:
    type: Custom ①
    custom: ②
      ciphers: ③
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" ④

```

- ① TLS セキュリティープロファイルタイプ (**Old**、**Intermediate**、または **Custom**) を指定します。デフォルトは **Intermediate** です。
- ② 選択したタイプに適切なフィールドを指定します。
  - **old:** {}
  - **intermediate:** {}
  - **custom:**
- ③ **custom** タイプには、TLS 暗号の一覧と最小許容 TLS バージョンを指定します。
- ④ オプション: TLS セキュリティープロファイルを適用するノードのマシン設定プールラベルを指定します。

2. **KubeletConfig** オブジェクトを作成します。

```
$ oc create -f <filename>
```

クラスター内のワーカーノードの数によっては、設定済みのノードが1つずつ再起動されるのを待機します。

### 検証

プロファイルが設定されていることを確認するには、ノードが **Ready** になってから以下の手順を実行します。

1. 設定済みノードのデバッグセッションを開始します。

```
$ oc debug node/<node_name>
```

2. **/host** をデバッグシェル内のルートディレクトリーとして設定します。

```
sh-4.4# chroot /host
```

3. **kubelet.conf** ファイルを表示します。

```
sh-4.4# cat /etc/kubernetes/kubelet.conf
```

### 出力例

```
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
...
"tlsCipherSuites": [
  "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_CHACHA20_POLY1305_SHA256",
  "TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256"
],
"tlsMinVersion": "VersionTLS12",
```

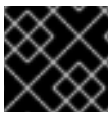
## 第10章 SECCOMP プロファイルの設定

OpenShift Container Platform コンテナまたは Pod は、1つ以上の明確に定義されたタスクを実行するアプリケーションを1つ実行します。アプリケーションには通常、基礎となるオペレーティングシステムカーネル API の小規模なサブセットのみが必要です。seccomp のセキュアコンピューティングモードは Linux カーネル機能で、これを使用して、コンテナで実行されているプロセスを制限して、利用可能なシステム呼び出しのサブセットだけが呼び出されるようにできます。これらのシステム呼び出しは、コンテナまたは Pod に適用されるプロファイルを作成して設定できます。seccomp プロファイルは、ディスクに JSON ファイルとして保存されます。



### 重要

seccomp プロファイルが適用されていない場合は、OpenShift ワークロードはデフォルトでは制限なしに実行されます。



### 重要

seccomp プロファイルは特権付きコンテナに適用できません。

### 10.1. すべての POD のデフォルトの SECCOMP プロファイルを有効にする

OpenShift Container Platform には、デフォルトの seccomp プロファイルが同梱されており、**runtime/default** として参照されます。カスタムセキュリティーコンテキスト制約 (SCC) を作成することで、Pod またはコンテナワークロードのデフォルトの seccomp プロファイルを有効にすることができます。



### 注記

カスタム SCC を作成する必要があります。デフォルトの SCC は編集しないでください。デフォルトの SCC を編集すると、プラットフォームの Pod をデプロイ時または OpenShift Container Platform のアップグレード時に問題が発生する可能性があります。詳細は、「デフォルトのセキュリティーコンテキストの制約」セクションを参照してください。

以下の手順に従って、すべての Pod に対してデフォルトの seccomp プロファイルを有効にします。

1. 使用可能な **restricted** SCC を yaml ファイルにエクスポートします。

```
$ oc get scc restricted -o yaml > restricted-seccomp.yaml
```

2. 作成された **restricted** SCC yaml ファイルを編集します。

```
$ vi restricted-seccomp.yaml
```

3. 次の例に示すように更新します。

```
kind: SecurityContextConstraints
metadata:
  name: restricted ①
<..snip..>
seccompProfiles: ②
- runtime/default ③
```



- 1 **restricted-seccomp** に変更します。
- 2 **seccompProfiles** を追加します。
- 3 **- runtime/default** を追加します。

4. カスタム SCC を作成します。

```
$ oc create -f restricted-seccomp.yaml
```

#### 予想される出力

```
securitycontextconstraints.security.openshift.io/restricted-seccomp created
```

5. カスタム SCC を ServiceAccount に追加します。

```
$ oc adm policy add-scc-to-user restricted-seccomp -z default
```



#### 注記

デフォルトのサービスアカウントは、ユーザーが別のアカウントを設定しない限り適用される ServiceAccount です。OpenShift Container Platform は、SCC の情報に基づいて Pod の seccomp プロファイルを設定します。

#### 予想される出力

```
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:restricted-seccomp added:
"default"
```

OpenShift Container Platform 4.9 では、pod アノテーション **seccomp.security.alpha.kubernetes.io/pod: runtime/default** および **container.seccomp.security.alpha.kubernetes.io/<container\_name>: runtime/default** を追加する機能は非推奨です。

## 10.2. カスタム SECCOMP プロファイルの設定

カスタム seccomp プロファイルを設定すると、アプリケーション要件に基づいてフィルターを更新できます。これにより、クラスター管理者は OpenShift Container Platform で実行されるワークロードのセキュリティをより詳細に制御できます。

### 10.2.1. カスタム seccomp プロファイルのセットアップ

#### 前提条件

- クラスター管理者パーミッションがある。
- カスタム SCC(Security Context Constraints) を作成している。詳細は、関連情報を参照してください。
- カスタム seccomp プロファイルを作成している。

## 手順

1. Machine Config を使用してカスタム seccomp プロファイルを `/var/lib/kubelet/seccomp/<custom-name>.json` にアップロードします。詳細な手順については、関連情報を参照してください。
2. 作成されたカスタム seccomp プロファイルへの参照を指定してカスタム SCC を更新します。

```
seccompProfiles:
- localhost/<custom-name>.json ❶
```

- ❶ カスタム seccomp プロファイルの名前を入力します。

### 10.2.2. カスタム seccomp プロファイルのワークロードへの適用

#### 前提条件

- クラスタ管理者はカスタム seccomp プロファイルを設定している。詳細は、カスタム seccomp プロファイルの設定を参照してください。

#### 手順

- **securityContext.seccompProfile.type** フィールドを次のように設定して、seccomp プロファイルをワークロードに適用します。

#### 例

```
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: <custom-name>.json ❶
```

- ❶ カスタム seccomp プロファイルの名前を入力します。

または、Pod アノテーション **seccomp.security.alpha.kubernetes.io/pod: localhost/<custom-name>.json** を使用できます。ただし、この手法は OpenShift Container Platform 4.9 では非推奨です。

デプロイメント時に、受付コントローラーは以下を検証します。

- 現在の SCC に対するアノテーションがユーザーロールで許可されている。
- seccomp プロファイルを含む SCC が Pod で許可されている。

SCC が Pod で許可される場合には、kubelet は指定された seccomp プロファイルで Pod を実行します。



#### 重要

seccomp プロファイルがすべてのワーカーノードにデプロイされていることを確認します。



## 注記

カスタム SCC は、Pod に適切な優先順位で自動的に割り当てられるか、または CAP\_NET\_ADMIN を許可するなど、Pod で必要な他の条件を満たす必要があります。

## 10.3. 関連情報

- [Security Context Constraints の管理](#)
- [インストール後のマシン設定タスク](#)

## 第11章 追加ホストから API サーバーへの JAVASCRIPT ベースのアクセスの許可

### 11.1. 追加ホストから API サーバーへの JAVASCRIPT ベースのアクセスの許可

デフォルトの OpenShift Container Platform 設定は、Web コンソールが要求を API サーバーに送信することのみを許可します。

別の名前を使用して JavaScript アプリケーションから API サーバーまたは OAuth サーバーにアクセスする必要がある場合、許可する追加のホスト名を設定できます。

#### 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

#### 手順

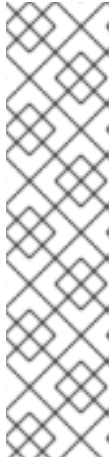
1. **APIServer** リソースを編集します。

```
$ oc edit apiserver.config.openshift.io cluster
```

2. **additionalCORSAAllowedOrigins** フィールドを **spec** セクションの下に追加し、1つ以上の追加のホスト名を指定します。

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-07-11T17:35:37Z"
  generation: 1
  name: cluster
  resourceVersion: "907"
  selfLink: /apis/config.openshift.io/v1/apiservers/cluster
  uid: 4b45a8dd-a402-11e9-91ec-0219944e0696
spec:
  additionalCORSAAllowedOrigins:
    - (?i)//my\.subdomain\.domain\.com(:|z) ①
```

- ① ホスト名は [Golang 正規表現](#) として指定されます。これは、API サーバーおよび OAuth サーバーに対する HTTP 要求の CORS ヘッダーに対するマッチングを行うために使用されます。



## 注記

この例では、以下の構文を使用します。

- **(?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、**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 (**z**) またはポートセパレーター (**:**) の終了部に一致します。

3. 変更を適用するためにファイルを保存します。

## 第12章 ETCD データの暗号化

### 12.1. ETCD 暗号化について

デフォルトで、etcd データは OpenShift Container Platform で暗号化されません。クラスターの etcd 暗号化を有効にして、データセキュリティのレイヤーを追加で提供することができます。たとえば、etcd バックアップが正しくない公開先に公開される場合に機密データが失われないように保護することができます。

etcd の暗号化を有効にすると、以下の OpenShift API サーバーおよび Kubernetes API サーバーリソースが暗号化されます。

- シークレット
- 設定マップ
- ルート
- OAuth アクセストークン
- OAuth 認証トークン

etcd 暗号を有効にすると、暗号化キーが作成されます。これらのキーは週ごとにローテーションされます。etcd バックアップから復元するには、これらのキーが必要です。



#### 注記

etcd 暗号化は、キーではなく、値のみを暗号化します。リソースの種類、namespace、およびオブジェクト名は暗号化されません。

バックアップ中に etcd 暗号化が有効になっている場合

は、**static\_kubereresources\_<datetimestamp>.tar.gz** ファイルに etcd スナップショットの暗号化キーが含まれています。セキュリティ上の理由から、このファイルは etcd スナップショットとは別に保存してください。ただし、このファイルは、それぞれの etcd スナップショットから etcd の以前の状態を復元するために必要です。

### 12.2. ETCD 暗号化の有効化

etcd 暗号化を有効にして、クラスターで機密性の高いリソースを暗号化できます。



## 警告

初期暗号化プロセスが完了するまで、etcd リソースをバックアップしないでください。暗号化プロセスが完了しない場合、バックアップは一部のみ暗号化される可能性があります。

etcd 暗号化を有効にすると、いくつかの変更が発生する可能性があります。

- etcd 暗号化は、いくつかのリソースのメモリー消費に影響を与える可能性があります。
- リーダーがバックアップを提供する必要があるため、バックアップのパフォーマンスに一時的な影響が生じる場合があります。
- ディスク I/O は、バックアップ状態を受け取るノードに影響を与える可能性があります。

## 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

## 手順

1. **APIServer** オブジェクトを変更します。

```
$ oc edit apiserver
```

2. **encryption** フィールドタイプを **aescbc** に設定します。

```
spec:
  encryption:
    type: aescbc ①
```

- ① **aescbc** タイプは、暗号化を実行するために PKCS#7 パディングを実装している AES-CBC と 32 バイトのキーが使用されることを意味します。

3. 変更を適用するためにファイルを保存します。  
暗号化プロセスが開始されます。クラスターのサイズによっては、このプロセスが完了するまで 20 分以上かかる場合があります。
4. etcd 暗号化が正常に行われたことを確認します。
  - a. OpenShift API サーバーの **Encrypted** ステータスを確認し、そのリソースが正常に暗号化されたことを確認します。

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}\n"{.message}\n"'
```

この出力には、暗号化が正常に実行されると **EncryptionCompleted** が表示されます。

```
EncryptionCompleted
All resources encrypted: routes.route.openshift.io
```

出力に **EncryptionInProgress** が表示される場合、これは暗号化が進行中であることを意味します。数分待機した後に再試行します。

- b. Kubernetes API サーバーの **Encrypted** ステータス状態を確認し、そのリソースが正常に暗号化されたことを確認します。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、暗号化が正常に実行されると **EncryptionCompleted** が表示されます。

```
EncryptionCompleted
All resources encrypted: secrets, configmaps
```

出力に **EncryptionInProgress** が表示される場合、これは暗号化が進行中であることを意味します。数分待機した後に再試行します。

- c. OpenShift OAuth API サーバーの **Encrypted** ステータスを確認し、そのリソースが正常に暗号化されたことを確認します。

```
$ oc get authentication.operator.openshift.io -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、暗号化が正常に実行されると **EncryptionCompleted** が表示されます。

```
EncryptionCompleted
All resources encrypted: oauthtokens.oauth.openshift.io,
oauthtokenreviews.oauth.openshift.io
```

出力に **EncryptionInProgress** が表示される場合、これは暗号化が進行中であることを意味します。数分待機した後に再試行します。

## 12.3. ETCD 暗号化の無効化

クラスターで etcd データの暗号化を無効にできます。

### 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

### 手順

1. **APIServer** オブジェクトを変更します。

```
$ oc edit apiserver
```

2. **encryption** フィールドタイプを **identity** に設定します。



```
spec:
  encryption:
    type: identity ❶
```

❶ **identity** タイプはデフォルト値であり、暗号化は実行されないことを意味します。

3. 変更を適用するためにファイルを保存します。  
復号化プロセスが開始されます。クラスターのサイズによっては、このプロセスが完了するまで 20 分以上かかる場合があります。
4. etcd の復号化が正常に行われたことを確認します。
  - a. OpenShift API サーバーの **Encrypted** ステータス条件を確認し、そのリソースが正常に暗号化されたことを確認します。

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、復号化が正常に実行されると **DecryptionCompleted** が表示されます。

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

出力に **DecryptionInProgress** が表示される場合、これは復号化が進行中であることを意味します。数分待機した後に再試行します。

- b. Kubernetes API サーバーの **Encrypted** ステータス状態を確認し、そのリソースが正常に復号化されたことを確認します。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、復号化が正常に実行されると **DecryptionCompleted** が表示されます。

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

出力に **DecryptionInProgress** が表示される場合、これは復号化が進行中であることを意味します。数分待機した後に再試行します。

- c. OpenShift API サーバーの **Encrypted** ステータス条件を確認し、そのリソースが正常に復号化されたことを確認します。

```
$ oc get authentication.operator.openshift.io -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、復号化が正常に実行されると **DecryptionCompleted** が表示されます。

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

出力に **DecryptionInProgress** が表示される場合、これは復号化が進行中であることを意味します。数分待機した後に再試行します。

## 第13章 POD の脆弱性のスキャン

Red Hat Quay Container Security Operator を使用すると、OpenShift Container Platform Web コンソールから、クラスターのアクティブな Pod で使用されるコンテナイメージについての脆弱性スキャンの結果にアクセスできます。The Red Hat Quay Container Security Operator:

- すべての namespace または指定された namespace の Pod に関連付けられたコンテナを監視します。
- イメージのレジストリーがイメージスキャンを実行している場合 (例: [Quay.io](#)、Clair スキャンを含む [Red Hat Quay](#) レジストリーなど)、脆弱性の情報についてコンテナの出所となったコンテナレジストリーをクエリーします。
- Kubernetes API の **ImageManifestVuln** オブジェクトを使用して脆弱性を公開します。

ここでの手順を使用すると、Red Hat Quay Container Security Operator は **openshift-operators** namespace にインストールされるため、OpenShift Container Platform クラスター上のすべての namespace で使用できます。

### 13.1. RED HAT QUAY CONTAINER SECURITY OPERATOR の実行

以下で説明されているように、Operator Hub から Operator を選択し、インストールして、OpenShift Container Platform Web コンソールから Red Hat Quay Container Security Operator を起動できます。

#### 前提条件

- OpenShift Container Platform クラスターへの管理者権限がある
- クラスターで実行される Red Hat Quay または Quay.io レジストリーのコンテナがある

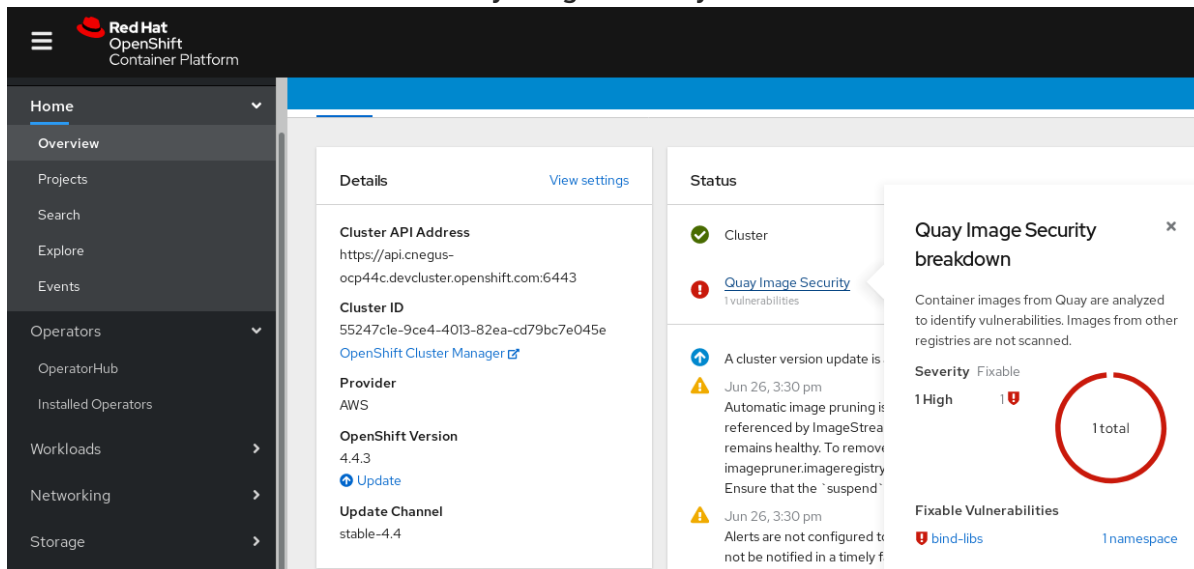
#### 手順

1. **Operators** → **OperatorHub** に移動し、**Security** を選択します。
2. **Container Security** Operator を選択し、**Install** を選択して Create Operator Subscription ページに移動します。
3. 設定を確認します。すべての namespace および自動承認ストラテジーがデフォルトで選択されます。
4. **Install** を選択します。**Container Security** Operator は、**Installed Operators** 画面に数分後に表示されます。
5. オプション: カスタム証明書を Red Hat Quay Container Security Operator に追加できます。以下の例では、現在のディレクトリーに **quay.crt** という名前の証明書を作成します。次に、次のコマンドを実行して、証明書を Red Hat Quay Container Security Operator に追加します。

```
$ oc create secret generic container-security-operator-extra-certs --from-file=quay.crt -n openshift-operators
```

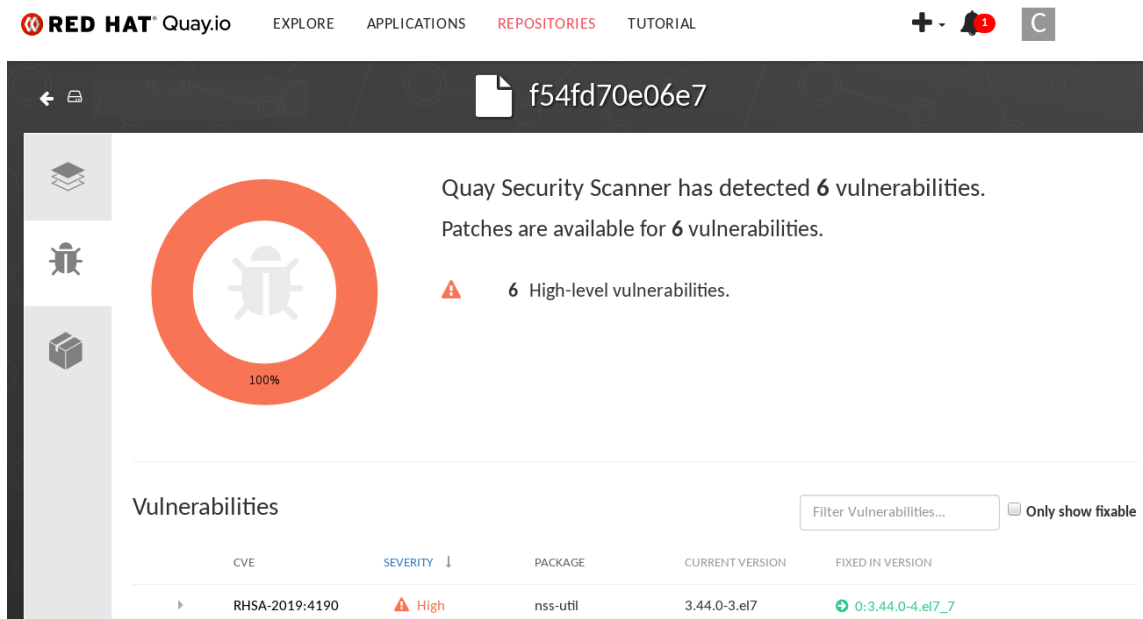
6. カスタム証明書を追加した場合、新規証明書を有効にするために Operator Pod を再起動します。

7. OpenShift Dashboard を開きます (**Home** → **Overview**)。Quay Image Security へのリンクが status セクションに表示され、これまでに見つかった脆弱性の数の一覧が表示されます。以下の図のように、リンクを選択して **Quay Image Security breakdown** を表示します。



8. この時点で、検出された脆弱性をフォローするために以下の2つのいずれかの操作を実行できます。

- 脆弱性へのリンクを選択します。コンテナーを取得したコンテナーレジストリーにアクセスし、脆弱性についての情報を確認できます。以下の図は、Quay.io レジストリーから検出された脆弱性の例を示しています。



- namespaces リンクを選択し、**ImageManifestVuln** 画面に移動します。ここでは、選択されたイメージの名前、およびイメージが実行されているすべての namespace を確認できます。以下の図は、特定の脆弱なイメージが **quay-enterprise** namespace で実行されていることを示しています。

Project: all projects ▾

## ImageManifestVuln

Create ImageManifestVuln

Filter by name... 

Name ↑	Namespace ↓	Created ↓
<b>VULN</b> sha256.f54fd70e06e745c2d840653b8b90ac79b59d59e7a25bcd4b83d6512a846975a2	<b>NS</b> quay-enterprise	9 minutes ago

この時点では、脆弱性のあるイメージや、イメージの脆弱性を解決するために必要なこと、およびイメージが実行されたすべての namespace を確認できます。以下を実行することができます。

- 脆弱性を修正する必要のあるイメージを実行しているユーザーに警告します。
- イメージが置かれている Pod を起動したデプロイメントまたは他のオブジェクトを削除して、イメージの実行を停止します。

Pod を削除すると、Dashboard で脆弱性のある状態がリセットされるまで数分かかる場合があります。

## 13.2. CLI でのイメージ脆弱性のクエリー

**oc** コマンドを使用して、Red Hat Quay Container Security Operator によって検出される脆弱性についての情報を表示できます。

### 前提条件

- OpenShift Container Platform インスタンスで Red Hat Quay Container Security Operator を実行している

### 手順

- 検出されたコンテナイメージの脆弱性についてクエリーするには、以下を入力します。

```
$ oc get vuln --all-namespaces
```

### 出力例

```
NAMESPACE  NAME                AGE
default    sha256.ca90...     6m56s
skynet     sha256.ca90...     9m37s
```

- 特定の脆弱性の詳細を表示するには、脆弱性の名前およびその namespace を **oc describe** コマンドに指定します。以下の例は、イメージに脆弱性のある RPM パッケージが含まれるアクティブなコンテナを示しています。

```
$ oc describe vuln --namespace mynamespace sha256.ac50e3752...
```

### 出力例

```
Name:      sha256.ac50e3752...
```

---

Namespace: quay-enterprise

...

Spec:

Features:

Name: nss-util

Namespace Name: centos:7

Version: 3.44.0-3.el7

Versionformat: rpm

Vulnerabilities:

Description: Network Security Services (NSS) is a set of libraries...

## 第14章 NETWORK-BOUND DISK ENCRYPTION (NBDE)

### 14.1. ディスクの暗号化技術について

Network-Bound Disk Encryption(NBDE) を使用すると、マシンの再起動時にパスワードを手動で入力しなくても、物理マシンおよび仮想マシン上のハードドライブのルートボリュームを暗号化できます。

#### 14.1.1. ディスク暗号化技術の比較

エッジサーバーにあるデータのセキュリティを保護するための Network-Bound Disk Encryption(NBDE) のメリットを理解するには、キー Escrow と Clevis を使用しない TPM ディスク暗号化と、Red Hat Enterprise Linux (RHEL) を実行するシステム上の NBDE を比較します。

以下の表は、脅威モデルや各暗号化ソリューションの複雑さについて考慮すべきトレードオフを示しています。

シナリオ	キー Escrow	TPM ディスク暗号化 (Clevis なし)	NBDE
単一ディスクの盗難からの保護	X	X	X
サーバー全体に対する保護	X		X
システムはネットワークから独立して再起動できます。		X	
定期的なキーの再生成なし		X	
キーはネットワーク経由で送信されることはありません。		X	X
OpenShift にサポート		X	X

##### 14.1.1.1. キー Escrow

キー Escrow は、暗号鍵を格納するための従来のシステムです。ネットワーク上の鍵サーバーは、暗号化されたブートディスクがあるノードの暗号化キーを保存して、クエリーされると返します。キー管理、トランスポートの暗号化、および認証が複雑であるため、ブートディスク暗号化は妥当な選択肢ではありません。

Red Hat Enterprise Linux(RHEL) で利用可能ですが、主要な escrow ベースのディスク暗号化の設定および管理は手動のプロセスであり、ノードの自動追加など OpenShift Container Platform 自動化操作には適しておらず、現在 OpenShift Container Platform でサポートされていません。

##### 14.1.1.2. TPM 暗号化

trusted Platform Module(TPM) ディスク暗号化は、リモートから保護された場所のデータセンターまたはインストールに適しています。dm-crypt and BitLocker などの完全なディスク暗号化ユーティリティーは、TPM バインドキーでディスクを暗号化し、TPM に TPM バンドキーを保存し、ノードのマザーボードにアタッチします。この方法の主な利点は、外部の依存関係がなく、ノードは外部の対話なしにブート時に独自のディスクを復号化できることです。

TPM ディスク暗号化は、ディスクがノードから盗まれて外部で分析される場合に、データが復号化されないように保護します。ただし、セキュアではない場所では、これは不十分です。たとえば、攻撃者がノード全体を盗んだ場合に、ノードは自身のディスクを復号化するので、ノードの電源を投入した時点でデータを傍受できてしまいます。これは、物理 TPM2 チップを備えたノードと、Virtual Trusted Platform Module(VTPM) アクセスのある仮想マシンに適用されます。

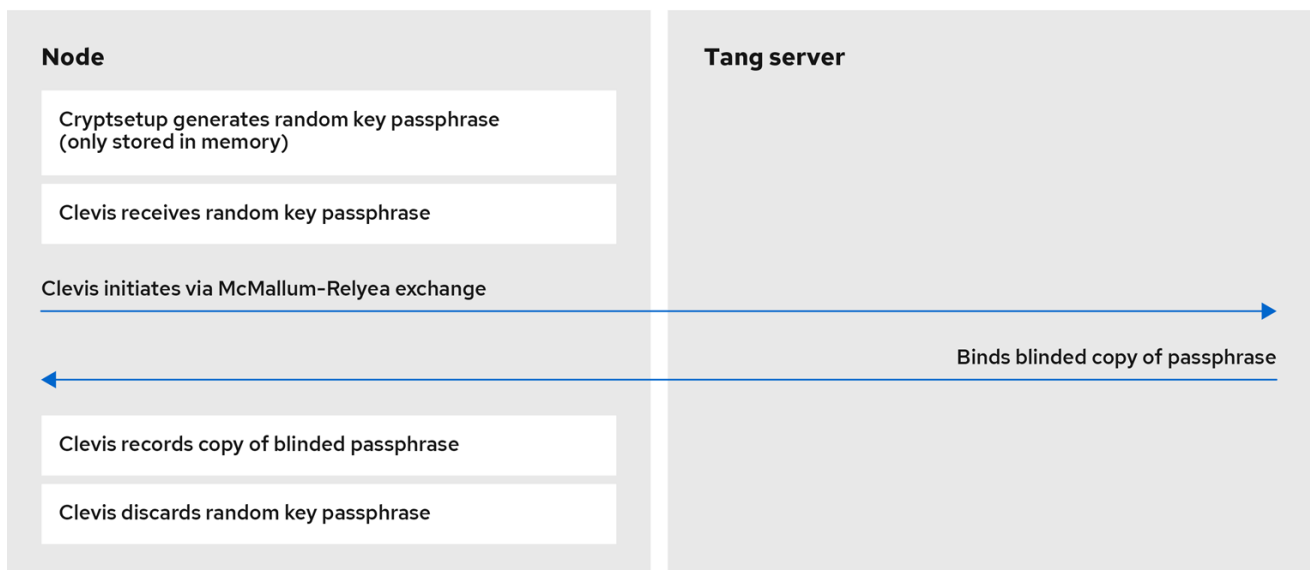
### 14.1.1.3. Network-Bound Disk Encryption (NBDE)

Network-Bound Disk Encryption(NBDE) は、ネットワーク全体で安全かつ匿名の方法で、暗号鍵を外部サーバーや、サーバーのセットに効果的に関連付けます。これは、ノードが暗号化キーの保存や、ネットワークでの転送を行わない点でキー escrow とは異なりますが、同じように動作します。

Clevis および Tang は、一般的なクライアントおよびサーバーのコンポーネントで、ネットワークがバインドされた暗号化を提供します。Red Hat Enterprise Linux CoreOS(RHCOS) は、Linux Unified Key Setup-on-disk-format(LUKS) と合わせて、これらのコンポーネントを使用して、Network-Bound Disk 暗号化を実現するために root および root 以外のストレージボリュームを暗号化および復号化します。

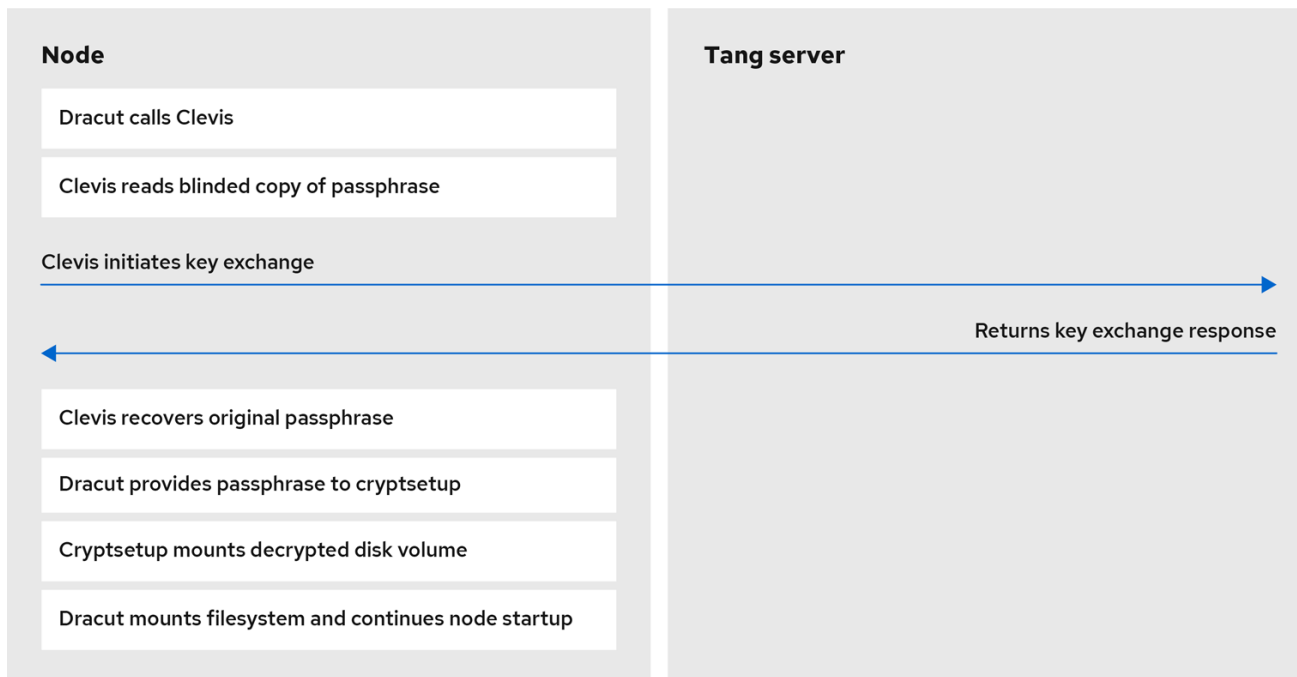
ノードが起動すると、暗号化ハンドシェイクを実行して事前に定義された Tang サーバーのセットへのアクセスを試みます。必要な数の Tang サーバーに到達できる場合は、ノードはディスクの復号化キーを作成し、ディスクのロックを解除して起動を続行できます。ネットワークが停止したり、サーバーが利用できなくなったりしたことが原因で、ノードが Tang サーバーにアクセスできない場合に、ノードは起動できず、Tang サーバーが再び利用可能になるまで無限に再試行し続けます。この鍵は実質的にはネットワーク内のノードに関連付けられるため、攻撃者が使用されていないデータへのアクセス権を取得しようとする、ノードのディスクと、Tang サーバーへのネットワークアクセスを取得する必要があります。

以下の図は、NBDE のデプロイメントモデルを示しています。



179\_OpenShift\_0821

以下の図は、リブート時の NBDE の動作を示しています。



179\_OpenShift\_0821

#### 14.1.1.4. シークレット共有の暗号化

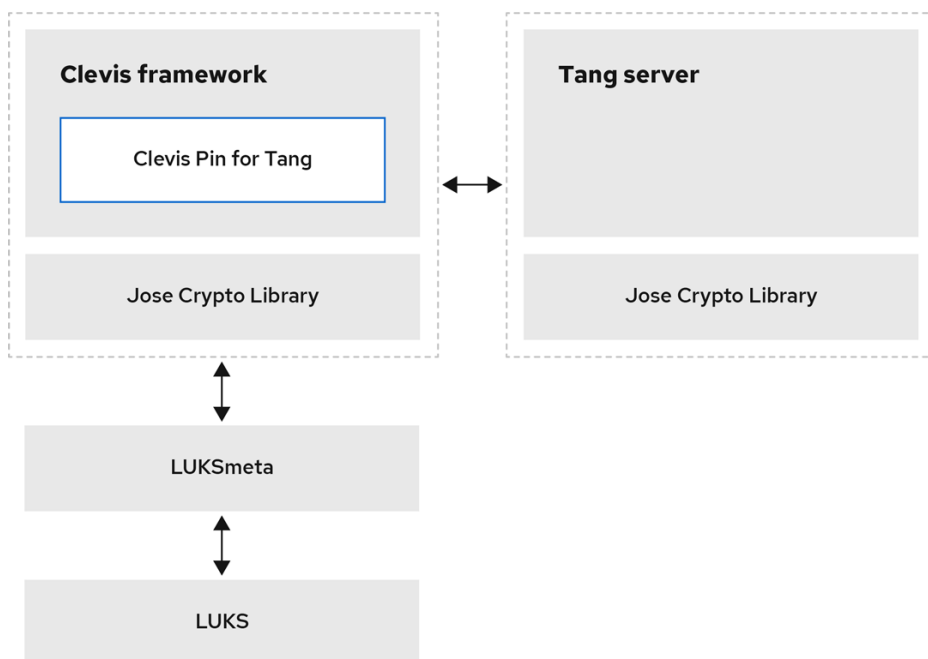
Shamir のシークレット共有 (sss) は、キーを安全に分割、配布、および再構築するための暗号化アルゴリズムです。このアルゴリズムを使用すると、OpenShift Container Platform は、より複雑なキー保護の組み合わせをサポートすることができます。

複数の Tang サーバーを使用するようにクラスターノードを設定する場合には、OpenShift Container Platform は、sss を使用して指定されたサーバーが1つ以上利用可能になると正常に実行される復号化ポリシーを設定します。追加でセキュリティー階層を作成できます。たとえば、OpenShift Container Platform でディスクの復号化に TPM と Tang サーバーの指定一覧の1つを必要とするポリシーを定義できます。

#### 14.1.2. Tang サーバーディスク暗号化

以下のコンポーネントおよびテクノロジーは、Network-Bound Disk Encryption(NBDE) を実装します。





179\_OpenShift\_0821

**Tang** は、ネットワークのプレゼンスにデータをバインドするためのサーバーです。これにより、ノードが特定のセキュアなネットワークにバインドされると、データが含まれるノードが利用可能になります。Tang はステートレスであり、Transport Layer Security(TLS) または認証は必要ありません。エスクローベースのソリューション (鍵サーバーが暗号鍵をすべて保存し、すべての鍵に関する情報を有する) とは異なり、Tang はノードの鍵と対話することはないため、ノードから識別情報を得ることがありません。

**Clevis** は、自動復号化用のプラグ可能なフレームワークで、Linux Unified Key Setup-on-disk-format(LUKS) ボリュームの自動ロック解除機能が含まれます。Clevis パッケージはノードで実行され、クライアント側の機能を提供します。

**Clevis ピン** は、Clevis フレームワークへのプラグインです。ピンニングには、以下の3つのタイプがあります。

#### TPM2

ディスク暗号化を TPM2 にバインドします。

#### Tang

ディスク暗号化を Tang サーバーにバインドし、NBDE を有効にします。

#### Shamir のシークレット共有 (ss)

より複雑な他のピンニングの組み合わせを可能にします。これにより、以下のような冗長なポリシーが可能になります。

- これらの3つの Tang サーバーのいずれかに到達する必要があります。
- これら5つの Tang サーバーのうち3つに到達する必要があります
- TPM2 と、これらの3つの Tang サーバーのいずれかに到達する必要があります。

### 14.1.3. Tang サーバーロケーションのプランニング

Tang サーバー環境を計画する時には、Tang サーバーの物理ネットワークおよびネットワークの場所を検討してください。

## 物理的な場所

Tang サーバーの地理的な場所は、不正アクセスや盗難から適切に保護されており、重要なサービスを実行するために必要な可用性があり、アクセスできる場合には、比較的重要ではありません。Clevis クライアントを使用するノードには、Tang サーバーが常に利用可能である限り、ローカルの Tang サーバーは必要ありません。障害復旧には、その場所に関係なく、Tang サーバーに対して電源とネットワーク接続がいずれも冗長設定されている必要があります。

## ネットワークの場所

Tang サーバーにネットワークアクセスできるノードは、独自のディスクパーティション、または同じ Tang サーバーで暗号化されたその他のディスクを復号化できます。

Tang サーバーのネットワークの場所を選択し、指定のホストからネットワーク接続の有無によって復号化のパーミッションが許可されるようにします。たとえば、ファイアウォール保護を適用して、あらゆるタイプのゲストネットワークやパブリックネットワーク、またはセキュリティーで保護されていない建物のエリアにあるネットワークジャックからのアクセスを禁止できます。

また、実稼働ネットワークと開発ネットワークの間でネットワークを分離します。これにより、適切なネットワークの場所を定義し、セキュリティーの層を追加するのに役立ちます。

ロック解除を行う Tang サーバーを同じリソース (例: 同じ `rolebindings.rbac.authorization.k8s.io` クラスター) にデプロイしないでください。ただし、Tang サーバーおよび他のセキュリティーリソースのクラスターは、複数の追加クラスターおよびクラスターリソースのサポートを有効にする、便利な設定である場合があります。

### 14.1.4. Tang サーバーのサイジング要件

可用性、ネットワーク、および物理的な場所に関する要件をもとに、サーバー容量の懸念点が決まるのではなく、使用する Tang サーバー数を決定できます。

Tang サーバーは、Tang リソースを使用して暗号化されたデータの状態を維持しません。Tang サーバーは完全に独立しているか、またはキー情報のみを共有するので、スケーリングが可能です。

以下の 2 つの方法を使用して、Tang サーバーで重要な情報を処理できます。

- 複数の Tang サーバーが重要な情報を共有します。
  - 同じ URL の背後にあるキーを共有する Tang サーバーを負荷分散する必要があります。この設定はラウンドロビン DNS と同様に簡単に実行することも、物理ロードバランサーを使用することもできます。
  - 単一の Tang サーバーから複数の Tang サーバーにスケーリングできます。Tang サーバーのスケーリングでは、Tang サーバーがキー情報と同じ URL を共有する場合にノードのキー変更やクライアントの再設定は必要ありません。
  - クライアントノードの設定とキーローテーションには Tang サーバー 1 台のみが必要です。
- 複数の Tang サーバーは独自のキー情報を生成します。
  - インストール時に複数の Tang サーバーを設定できます。
  - ロードバランサーの背後では個別の Tang サーバーをスケーリングできます。
  - クライアントノードの設定またはキーのローテーション時に全 Tang サーバーが利用可能である必要があります。

- クライアントノードがデフォルト設定を使用して起動すると、Clevis クライアントはすべての Tang サーバーに接続します。復号化を続行するには、オンラインにしておく必要のある Tang サーバーは  $n$  台のみです。 $n$  のデフォルト値は 1 です。
- Red Hat では、Tang サーバーの動作を変更するインストール後の設定をサポートしません。

### 14.1.5. ロギングについての考慮事項

Tang トラフィックの集中ロギングは、予期しない復号化要求などを検出できる可能性があるため、便利です。以下に例を示します。

- ブートシーケンスに対応しないパスフレーズの復号化を要求するノード
- 既知のメンテナンスアクティビティ外の復号化を要求するノード (例: 繰り返しキー)

## 14.2. TANG サーバーのインストールに関する考慮事項

### 14.2.1. インストールシナリオ

Tang サーバーインストールを計画する場合は、以下の推奨事項を検討してください。

- 小規模な環境では、複数の Tang サーバーを使用する場合でも、単一のキー情報のセットを使用できます。
  - キーのローテーションが容易になりました。
  - Tang サーバーは、高可用性を確保できるように簡単にスケーリングできます。
- 大規模な環境には、複数のキー情報のセットからメリットを得ることができます。
  - 物理的なインストールでは、地理的リージョン間のキー情報のコピーおよび同期は必要ありません。
  - キーローテーションは大規模な環境ではより複雑です。
  - ノードのインストールおよびキー変更には、すべての Tang サーバーへのネットワーク接続が必要です。
  - ブートノードが復号化中にすべての Tang サーバーに対してクエリーを実行するため、ネットワークトラフィックがわずかに増加する可能性があります。成功する必要があるのは Clevis クライアントクエリー 1 つのみですが、Clevis はすべての Tang サーバーにクエリーを実行することに注意してください。
- 複雑性:
  - 追加の手動再設定では、ディスクパーティションを復号化するために、**オンラインのサーバー M 台中 N 台** の Shamir シークレット共有 (sss) を許可できます。このシナリオでディスクを復号化するには、複数のキー情報のセットと、初回インストール後に Clevis クライアントが含まれる Tang サーバーとノードを手動で管理する必要があります。
- ハイレベルの推奨事項:
  - 単一の RAN デプロイメントの場合には、対応するドメインコントローラー (DC) で、一部の Tang サーバーセットを実行できます。

- 複数の RAN デプロイメントの場合には、対応する各 DC で Tang サーバーを実行するか、またはグローバル Tang 環境が他のシステムのニーズと要件に適しているかどうかを決定する必要があります。

## 14.2.2. Tang サーバーのインストール

### 手順

- 以下のコマンドのいずれかを使用して、Red Hat Enterprise Linux(RHEL) マシンに Tang サーバーをインストールできます。

- **yum** コマンドを使用して Tang サーバーをインストールします。

```
$ sudo yum install tang
```

- **dnf** コマンドを使用して Tang サーバーをインストールします。

```
$ sudo dnf install tang
```



### 注記

インストールはコンテナ化でき、非常に軽量です。

### 14.2.2.1. コンピュートの要件

Tang サーバーの計算要件は非常に低くなります。サーバーを実稼働環境にデプロイするのに使用する通常のサーバーグレードの設定であれば、十分なコンピュート容量をプロビジョニングできます。

高可用性に関する考慮事項は、可用性のみを対象としており、クライアントの要件に対応するための追加のコンピュート能力ではありません。

### 14.2.2.2. 起動時の自動開始

Tang サーバーが使用するキー情報は機密性が高いため、Tang サーバーの起動シーケンス中の手動介入のオーバーヘッドが有益である可能性があるため、忘れないでください。

デフォルトでは、Tang サーバーが起動し、想定されるローカルボリュームにキー情報がない場合には、新しい材料を作成し、それを提供します。既存のキー情報を使って開始するか、または起動を中止して手動の介入を待つことで、このデフォルトの動作を避けることができます。

### 14.2.2.3. HTTP 対 HTTPS

Tang サーバーへのトラフィックは、暗号化 (HTTPS) またはプレーンテキスト (HTTP) にすることができます。このトラフィックを暗号化してもセキュリティの面で大きな利点はなく、トラフィックを復号化したままにすると、Clevis クライアントを実行するノードでのトランスポート層セキュリティ (TLS) 証明書チェックに関連する複雑性や障害状態がなくなります。

ノードの Clevis クライアントと Tang サーバー間の暗号化されていないトラフィックのパッシブモニターリングを実行することは可能ですが、このトラフィックを使用してキーの情報を判断する機能はせいぜい、理論的な懸念事項を判断できる程度です。このようなトラフィック分析には、大量のキャプチャーデータが必要になります。キーローテーションはすぐに無効になります。最後に、パッシブモニターリングを実行できる脅威アクターは、Tang サーバーへの手動接続に必要なネットワークアクセスをすでに取得しており、キャプチャーされた Clevis ヘッダーの単純な復号化を実行できます。

ただし、インストールサイトで実施されている他のネットワークポリシーでは、アプリケーションに関係なくトラフィックの暗号化が必要になる場合があるため、この決定はクラスター管理者に任せることを検討してください。

### 14.2.3. Network-Bound Disk Encryption に関するインストール時の考慮事項

Network-Bound Disk Encryption(NBDE) はクラスターノードのインストール時に有効にする必要があります。ただし、ディスク暗号化ポリシーは、インストール時の初期化後に、いつでも変更できます。

#### 関連情報

- [ポリシーベースの複号を使用して暗号化ボリュームの自動アンロックの設定](#)
- [Tang の正式なサーバーコンテナ](#)
- [インストール時のディスクの暗号化およびミラーリング](#)

## 14.3. TANG サーバーの暗号化キー管理

暗号キーを再作成するための暗号化メカニズムは、ノードに保管されている **ブラインドキー** と関係する Tang サーバーの秘密鍵に基づいています。Tang サーバーの秘密鍵とノードの暗号化ディスクの両方を取得した者による攻撃から保護するには、定期的にキーを再生成することを推奨します。

Tang サーバーから古いキーを削除する前に、すべてのノードでキー変更操作を実行する必要があります。以下のセクションでは、古いキーを再生成し、削除する手順を説明します。

### 14.3.1. Tang サーバーのキーのバックアップ

Tang サーバーは **/usr/libexec/tangd-keygen** を使用して新しいキーを生成し、デフォルトで **/var/db/tang** ディレクトリーに保存します。障害が発生した場合に Tang サーバーを回復するには、このディレクトリーをバックアップします。キーは機密性が高く、使用した全ホストのブートディスクを復号化できるため、キーは適切に保護される必要があります。

#### 手順

- バックアップキーを **/var/db/tang** ディレクトリーから、キーを復元できる temp ディレクトリーにコピーします。

### 14.3.2. Tang サーバーのキーのリカバリー

バックアップからキーにアクセスして、Tang サーバーのキーを回復できます。

#### 手順

- キーをバックアップフォルダーから **/var/db/tang/** ディレクトリーに復元します。  
Tang サーバーの起動時に、これらの復元されたキーをアドバタイズして使用します。

### 14.3.3. Tang サーバーのキー変更

以下の手順では、例として一意のキーが割り当てられた 3 つの Tang サーバーセットを使用します。

冗長な Tang サーバーを使用すると、ノードの自動起動に失敗する可能性が低減します。

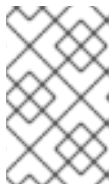
Tang サーバーおよび関連付けられたすべての NBDE 暗号化ノードのキーは 3 つの手順を使用して、再生成します。

**前提条件**

- 1 つ以上のノードで機能する Network-Bound Disk Encryption(NBDE) をインストールしておく。

**手順**

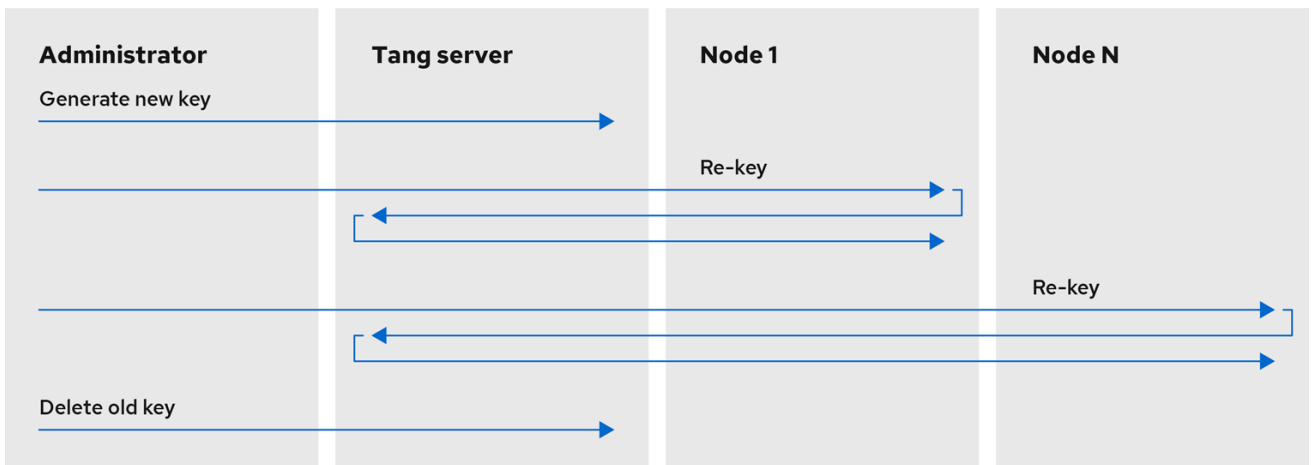
1. 新しい Tang サーバーキーを生成します。
2. NBDE で暗号化された全ノードに新規キーを使用するようにキーを再生成します。
3. 古い Tang サーバーキーを削除します。



**注記**

NBDE で暗号化されたすべてのノードがキーの再生成を完了する前に古いキーを削除すると、それらのノードは他の設定済みの Tang サーバーに過度に依存するようになります。

図14.1 Tang サーバーのキー再生成のワークフロー例



179\_OpenShift\_0821

**14.3.3.1. 新しい Tang サーバーキーの生成**

**前提条件**

- Tang サーバーを実行する Linux マシンのルートシェル。
- Tang サーバーキーのローテーションを容易に検証するには、古いキーで小規模なテストファイルを暗号化します。

```
# echo plaintext | clevis encrypt tang '{"url":"http://localhost:7500"}' -y >/tmp/encrypted.oldkey
```

- 暗号化が正常に完了し、ファイルを復号化して同じ文字列 **plaintext** を生成できることを確認します。

```
# clevis decrypt </tmp/encrypted.oldkey
```

## 手順

1. Tang サーバーキーの保存先のディレクトリーを見つけてアクセスします。通常、これは `/var/db/tang` ディレクトリーです。現在公開されているキーのサムプリントを確認します。

```
# tang-show-keys 7500
```

## 出力例

```
36AHjNH3NZDSnlONLz1-V4ie6t8
```

2. Tang サーバーのキーディレクトリーを入力します。

```
# cd /var/db/tang/
```

3. 現在の Tang サーバーキーを一覧表示します。

```
# ls -A1
```

## 出力例

```
36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
gJZiNPMLRBnyo_ZKfK4_5SrnhYo.jwk
```

通常の Tang サーバーの操作時に、このディレクトリーには、署名および検証用とキー派生用の2つの `.jwk` ファイルがあります。

4. 古いキーのアドバタイズを無効にします。

```
# for key in *.jwk; do \
  mv -- "$key" ".${key}"; \
done
```

Network-Bound Disk Encryption(NBDE) を使用する新規クライアント、またはキーを要求する新規クライアントには古いキーは表示されなくなりました。既存のクライアントは、削除されるまで以前のキーにアクセスして使用できます。Tang サーバーは、`.`文字で始まる、UNIX の非表示ファイルに保存されているキーを読み取りますがアドバタイズはしません。

5. 新しいキーを生成します。

```
# /usr/libexec/tangd-keygen /var/db/tang
```

6. これらのファイルは非表示になり、新しいキーが存在するので、現在の Tang サーバーキーを一覧表示して古いキーがアドバタイズされなくなったことを確認します。

```
# ls -A1
```

## 出力例

```
.36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
.gJZiNPMLRBnyo_ZKfK4_5SrnHYo.jwk
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

Tang は、新しいキーを自動的にアドバタイズします。



### 注記

より新しい Tang サーバーのインストールには、アドバタイズを無効にして新規キーを同時に生成するヘルパー `/usr/libexec/tangd-rotate-keys` ディレクトリーが含まれます。

7. 同じキー情報を共有するロードバランサーの背後で複数の Tang サーバーを実行している場合は、続行する前に、ここで加えた変更が一連のサーバー全体に適切に同期されていることを確認します。

## 検証

1. Tang サーバーが、古いキーではなく、新しいキーをアドバタイズすることを確認します。

```
# tang-show-keys 7500
```

### 出力例

```
WOjQYkyK7DxY_T5pMncMO5w0f6E
```

2. アドバタイズされていない古いキーがまだ復号化要求に使用できることを確認します。

```
# clevis decrypt </tmp/encrypted.oldkey
```

### 14.3.3.2. 全 NBDE ノードのキー変更

リモートクラスターにダウンタイムを発生させずに **DaemonSet** オブジェクトを使用することで、リモートクラスターのすべてのノードにキーを再生成できます。



### 注記

キー設定時にノードの電源が切れると、起動できなくなる可能性があり、Red Hat Advanced Cluster Management(RHACM) または GitOps パイプラインを使用して再デプロイする必要があります。

## 前提条件

- Network-Bound Disk Encryption(NBDE) ノードが割り当てられたすべてのクラスターへの **cluster-admin** アクセス。
- Tang サーバーのキーが変更されていない場合でも、キーの再生成が行われるすべての NBDE ノードからすべての Tang サーバーにアクセスする必要があります。
- すべての Tang サーバーの Tang サーバーの URL およびキーのサムプリントを取得します。



## 手順

1. 以下のテンプレートに基づいて **DaemonSet** オブジェクトを作成します。このテンプレートは3つの冗長 Tang サーバーを設定しますが、他の状況にも簡単に対応できます。**NEW\_TANG\_PIN** 環境の Tang サーバーの URL およびサムプリントを、実際の環境に合わせて変更します。

```

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: tang-rekey
  namespace: openshift-machine-config-operator
spec:
  selector:
    matchLabels:
      name: tang-rekey
  template:
    metadata:
      labels:
        name: tang-rekey
    spec:
      containers:
        - name: tang-rekey
          image: registry.access.redhat.com/ubi8/ubi-minimal:8.4
          imagePullPolicy: IfNotPresent
          command:
            - "/sbin/chroot"
            - "/host"
            - "/bin/bash"
            - "-ec"
          args:
            - |
              rm -f /tmp/rekey-complete || true
              echo "Current tang pin:"
              clevis-luks-list -d $ROOT_DEV -s 1
              echo "Applying new tang pin: $NEW_TANG_PIN"
              clevis-luks-edit -f -d $ROOT_DEV -s 1 -c "$NEW_TANG_PIN"
              echo "Pin applied successfully"
              touch /tmp/rekey-complete
              sleep infinity
      readinessProbe:
        exec:
          command:
            - cat
            - /host/tmp/rekey-complete
          initialDelaySeconds: 30
          periodSeconds: 10
      env:
        - name: ROOT_DEV
          value: /dev/disk/by-partlabel/root
        - name: NEW_TANG_PIN
          value: >-
            {"t":1,"pins":{"tang":[
              {"url":"http://tangserver01:7500","thp":"WOjQYkyK7DxY_T5pMncMO5w0f6E"},
              {"url":"http://tangserver02:7500","thp":"I5Ynh2JefoAO3tNH9TgI4oblaXI"},
              {"url":"http://tangserver03:7500","thp":"38qWZVeDKzCPG9pHLqKzs6k1ons"}
            ]}

```

```

    }}
  volumeMounts:
  - name: hostroot
    mountPath: /host
  securityContext:
    privileged: true
  volumes:
  - name: hostroot
    hostPath:
      path: /
  nodeSelector:
    kubernetes.io/os: linux
  priorityClassName: system-node-critical
  restartPolicy: Always
  serviceAccount: machine-config-daemon
  serviceAccountName: machine-config-daemon

```

この場合は、**tangserver01** のキーを再生成していても、**tangserver01** の新規サムプリントだけでなく、その他すべての Tang サーバーの現在のサムプリントも指定する必要があります。キーの再生成操作にすべてのサムプリントの指定に失敗すると、中間者攻撃の可能性が高まります。

2. キーの再生成が必要なすべてのクラスターにデーモンセットを配布するには、次のコマンドを実行します。

```
$ oc apply -f tang-rekey.yaml
```

ただし、スケーリングで実行するには、デーモンセットを ACM ポリシーでラップします。この ACM 設定には、デーモンセットをデプロイするポリシーが1つ、すべてのデーモンセット Pod が READY であることを確認する 2 番目のポリシー、およびクラスターの適切なセットに適用する配置ルールを含める必要があります。



## 注記

デーモンセットのすべてのサーバーが正常に再割り当てされたことを確認したら、デーモンセットを削除します。デーモンセットを削除しない場合は、次のキー再生成操作の前にこれを削除する必要があります。

## 検証

デーモンセットを配布したら、デーモンセットを監視し、キー作成が正常に完了したことを確認します。サンプルのデーモンセットのスクリプトは、キー変更に失敗するとエラーで終了し、成功した場合には **CURRENT** 状態のままになります。また、キーの再生成が正常に実行されると、Pod に **READY** のマークを付ける readiness プロブもあります。

- 以下は、キー変更が完了する前に設定されたデーモンセットの出力一覧の例です。

```
$ oc get -n openshift-machine-config-operator ds tang-rekey
```

## 出力例

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE
SELECTOR	AGE					
tang-rekey	1	1	0	1	0	kubernetes.io/os=linux 11s

- 以下は、キーの変更が正常に実行された後のデーモンセットの出力一覧の例です。

```
$ oc get -n openshift-machine-config-operator ds tang-rekey
```

### 出力例

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE
SELECTOR	AGE					
tang-rekey	1	1	1	1	kubernetes.io/os=linux	13h

キーの再生成には、通常完了までに数分かかります。



### 注記

ACM ポリシーを使用してデーモンセットを複数のクラスターに分散する場合には、すべてのデーモンセットの **READY** の数が **DESIRED** の数と等しいことを確認するコンプライアンスポリシーを含める必要があります。こうすることで、このようなポリシーに準拠すると、すべてのデーモンセット Pod が **READY** で、キーの再生成が正常に実行されていることが分かります。ACM 検索を使用して、デーモンセットのすべての状態をクエリーすることもできます。

#### 14.3.3.3. Tang サーバーの一時的な再生成エラーのトラブルシューティング

Tang サーバーのキーの再生成のエラー状態が一時的かどうかを判別するには、以下の手順を実行します。一時的なエラー状態には以下が含まれます。

- 一時的なネットワークの停止
- Tang サーバーのメンテナンス

通常、これらのタイプの一時的なエラー状態が発生した場合には、デーモンセットがエラーを解決して成功するまで待機するか、またはデーモンセットを削除し、一時的なエラー状態が解決されるまで再試行しないようにします。

### 手順

1. 通常の Kubernetes Pod 再起動ポリシーを使用してキーの再生成操作を実行する Pod を再起動します。
2. 関連付けられた Tang サーバーのいずれかが利用できない場合は、すべてのサーバーがオンラインに戻るまでキーの再生成を試みます。

#### 14.3.3.4. Tang サーバーの永続的な再生成エラーのトラブルシューティング

Tang サーバーのキーを再生成した後に、長期間経過しても **READY** の数が **DESIRED** の数と等しくない場合は、永続的な障害状態を示している可能性があります。この場合、以下のような状況である場合があります。

- **NEW\_TANG\_PIN** 定義で Tang サーバーの URL またはサムプリントの誤字がある。
- Tang サーバーが無効になっているか、またはキーが完全に失われている。

### 前提条件

- この手順で説明されているコマンドが、Tang サーバーまたは Tang サーバーへのネットワークアクセスのある Linux システムで実行できる。

## 手順

1. デモンセットの定義に従って各 Tang サーバーの設定に対して単純な暗号化および復号化操作を実行して、Tang サーバー設定を検証します。  
これは、不適切なサムプリントで暗号化および復号化を試行する例です。

```
$ echo "okay" | clevis encrypt tang \
  '{"url":"http://tangserver02:7500","thp":"badthumbprint"}' | \
  clevis decrypt
```

### 出力例

```
Unable to fetch advertisement: 'http://tangserver02:7500/adv/badthumbprint'!
```

これは、正常なサムプリントで暗号化および復号化を試行する例です。

```
$ echo "okay" | clevis encrypt tang \
  '{"url":"http://tangserver03:7500","thp":"goodthumbprint"}' | \
  clevis decrypt
```

### 出力例

```
okay
```

2. 根本的な原因を特定した後に、その原因となる状況に対応します。
  - a. 機能しないデモンセットを削除します。
  - b. デモンセットの定義を編集して基礎となる問題を修正します。これには、以下のアクションのいずれかが含まれる場合があります。
    - Tang サーバーのエントリーを編集して URL とサムプリントを修正します。
    - 使用されなくなった Tang サーバーを削除します。
    - 使用停止したサーバーの代わりとなる、新規の Tang サーバーを追加します。
3. 更新されたデモンセットを再度配布します。



## 注記

Tang サーバーを設定から置き換え、削除、または追加する場合に、現在キーの再生成が行われているサーバーを含め、少なくとも1つの元のサーバーが機能している限り、キーの再生成操作は成功します。元の Tang サーバーのいずれも機能しなくなるか、または復元できない場合には、システムの回復は不可能で、影響を受けるノードを再デプロイする必要があります。

## 検証

デーモンセットの各 Pod からのログをチェックして、キーの再生成が正常に完了したかどうかを判断します。キーの再生成に成功しなかった場合には、その失敗している状態がログに表示される可能性があります。

1. デーモンセットによって作成されたコンテナの名前を見つけます。

```
$ oc get pods -A | grep tang-rekey
```

#### 出力例

```
openshift-machine-config-operator tang-rekey-7ks6h 1/1 Running 20 (8m39s ago) 89m
```

2. コンテナからログを印刷します。キーの再生成操作に成功すると、以下のログのようになります。

```
$ oc logs tang-rekey-7ks6h
```

#### 出力例

```
Current tang pin:
1: sss '{"t":1,"pins":{"tang":[{"url":"http://10.46.55.192:7500"},{"url":"http://10.46.55.192:7501"},
{"url":"http://10.46.55.192:7502"}]}}'
Applying new tang pin: {"t":1,"pins":{"tang":[
{"url":"http://tangserver01:7500","thp":"WOjQYkyK7DxY_T5pMncMO5w0f6E"},
{"url":"http://tangserver02:7500","thp":"l5Ynh2JefoAO3tNH9Tgl4oblaXl"},
{"url":"http://tangserver03:7500","thp":"38qWZVeDKzCPG9pHLqKzs6k1ons"}
]}}
Updating binding...
Binding edited successfully
Pin applied successfully
```

### 14.3.4. 古い Tang サーバーキーの削除

#### 前提条件

- Tang サーバーを実行する Linux マシンのルートシェル。

#### 手順

1. Tang サーバーキーが保存されるディレクトリを見つけ、これにアクセスします。通常、これは `/var/db/tang` ディレクトリです。

```
# cd /var/db/tang/
```

2. 現在の Tang サーバーキーを一覧表示し、アドバタイズされたキーと、されていないキーを表示します。

```
# ls -A1
```

#### 出力例

```
.36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
```

```
.gJZiNPMLRBnyo_ZKfK4_5SrnHYo.jwk  
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk  
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

3. 古いキーを削除します。

```
# rm *.jwk
```

4. 現在の Tang サーバーのキーを一覧表示し、アドバタイズされていないキーが存在しなくなったことを確認します。

```
# ls -A1
```

### 出力例

```
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk  
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

## 検証

この時点では、サーバーは新しいキーを引き続きアドバタイズしますが、古いキーをもとに復号化の試行は失敗します。

1. Tang サーバーに、現在公開されているキーのサムプリントについてクエリーします。

```
# tang-show-keys 7500
```

### 出力例

```
WOjQYkyK7DxY_T5pMncMO5w0f6E
```

2. 先に作成したテストファイルを復号化して、以前のキーに対して復号化を検証します。

```
# clevis decrypt </tmp/encryptValidation
```

### 出力例

```
Error communicating with the server!
```

同じキー情報を共有するロードバランサーの背後に複数の Tang サーバーを実行している場合は、続行する前に、一連のサーバーで変更が適切に同期されていることを確認します。

## 14.4. 障害復旧に関する考慮事項

本項では、発生する可能性のある障害状況と、それぞれの状況に対応する手順について説明します。その他の状況は、検出または想定される可能性の高い状況として追加されます。

### 14.4.1. クライアントマシンの損失

Tang サーバーを使用してディスクパーティションを復号化するクラスターノードが失われた場合は、**障害ではありません**。マシンの盗難、ハードウェアの障害、別の損失のシナリオが発生したのかは重要ではありません。ディスクは暗号化されて、回復不能とみなされます。

ただし、盗難が発生した場合は、Tang サーバーのキーを予防的にローテーションし、残りのすべてのノードのキーを再設定して、後で Tang サーバーに不正アクセスできた場合に備え、ディスクが回復不能なままにしておくことが懸命です。

この状況から回復するには、ノードを再インストールするか、または置き換えます。

#### 14.4.2. クライアントネットワーク接続が失われた場合のプランニング

個々のノードに対してネットワーク接続が失われると、無人で起動できなくなります。

ネットワーク接続が失われる可能性のある作業を計画している場合には、オンサイトの技術者に、手動で使用するパスフレーズを公開し、その後にキーを無効にしてローテーションすることができます。

##### 手順

1. ネットワークが利用できなくなる前に、以下のコマンドでデバイス `/dev/vda2` の最初のスロット `-s 1` で使用されているパスワードを表示します。

```
$ sudo clevis luks pass -d /dev/vda2 -s 1
```

2. 以下のコマンドでその値を無効にし、新しい起動時のパスフレーズを無作為に再生成します。

```
$ sudo clevis luks regen -d /dev/vda2 -s 1
```

#### 14.4.3. ネットワーク接続の予期しない損失

ネットワークが予期せず中断され、ノードが再起動する場合には、以下のシナリオを検討してください。

- いずれかのノードがまだオンラインのままである場合は、ネットワーク接続が復元されるまで再起動しないようにしてください。これは単一ノードクラスターには適用されません。
- ノードは、ネットワーク接続が復元されるまで、またはコンソールで設定したパスフレーズを手動で入力するまでノードはオフラインのままになります。このような状況では、ネットワーク管理者はアクセスを再確立するためにネットワークセグメントを再設定できる可能性がありますが、これは NBDE の意図に反します。つまり、ネットワークアクセスがないと起動できなくなります。
- ノードへのネットワークアクセスがないと、ノードの機能およびブート機能に影響を与えることが予想されます。ノードが手動の介入で起動されたとしても、ネットワークアクセスがないため、ノードは事実上役に立たなくなります。

#### 14.4.4. ネットワーク接続の手動による回復

オンサイト技術者は、ネットワーク回復のために、やや複雑で手動での作業を多用するプロセスも利用できます。

##### 手順

1. オンサイト技術者は、ハードディスクから Clevis ヘッダーを抽出します。BIOS のロックダウンによっては、ディスクを削除してラボマシンにインストールする必要がある場合があります。

2. オンサイトの技術者は、Tang ネットワークへの正当なアクセス権があるスタッフに Clevis ヘッダーを送信し、そのスタッフが復号化を実行します。
3. Tang ネットワークへのアクセスを制限する必要があるため、技術者は VPN または他のリモート接続経由でそのネットワークにアクセス不可にする必要があります。同様に、技術者はこのネットワーク経由でリモートサーバーにパッチを適用して、ディスクを自動的に復号化できません。
4. ファイルシステムはディスクを再インストールし、それらが提供するプレーンテキストのパスフレーズを手動で入力します。
5. マシンは、Tang サーバーに直接アクセスしなくても、正常に起動します。インストールサイトからネットワークアクセスのある別のサイトにキー情報を転送する場合は注意して行ってください。
6. ネットワーク接続が回復すると、暗号鍵がローテーションされます。

#### 14.4.5. ネットワーク接続の緊急復旧

ネットワーク接続を手動で回復できない場合には、以下の手順を検討してください。ネットワーク接続を回復する他の方法が利用できる場合には、これらの手順は推奨されないことに注意してください。

- この方法は、信頼性の高い技術者のみが実行する必要があります。
- Tang サーバーキー鍵情報をリモートサイトに送付することは、キー情報の違反とみなされ、すべてのサーバーでキーをもう一度生成して再暗号化する必要があります。
- この方法は、それ以外方法がない究極の場合に利用するようにしてください。あるいは、その実行可能性を実証するための概念実証の回復方法として使用する必要があります。
- 同様に極端で、理論的には可能ですが、問題のサーバーに無停電電源装置 (UPS) で電力を供給し、サーバーをネットワーク接続のある場所に転送してディスクを起動および復号化し、サーバーをバッテリー電源のある元の場所に復元して操作を続行します。
- バックアップの手動パスフレーズを使用する場合は、障害が発生する前にこれを作成する必要があります。
- 攻撃シナリオが TPM と Tang とスタンドアロンの Tang インストールと比較すると、より複雑になるので、同じ方法を使用する場合には、障害復旧プロセスも複雑になります。

#### 14.4.6. ネットワークセグメントの喪失

ネットワークセグメントが失われ、Tang サーバーが一時的に使用できなくなると、次のような結果になります。

- OpenShift Container Platform ノードは、他のサーバーが利用可能な場合に通常通りに起動を続けます。
- 新規ノードは、ネットワークセグメントが復元されるまでその暗号化キーを確立できません。この場合は、高可用性および冗長性を確保するために、地理的に離れた場所への接続を確保します。これは、新規ノードのインストールや既存ノードの再割り当て時に、その操作で参照している Tang サーバーがすべて利用できる必要があるためです。

5つの地理的リージョンに各クライアントが最寄りの3つのクライアントに接続されている場合など、幅広いネットワークのハイブリッドモデルを検討する価値があります。

このシナリオでは、新規クライアントが到達可能なサーバーのサブセットを使用して暗号鍵を設定でき



ます。tang1、tang2 および tang3 サーバーのセットで、tang2 が到達できなくなった場合にクライアントは tang1 および tang3 で暗号鍵を設定して、後ほど完全なセットを使用して再確立できます。これには、手動による介入、またはより複雑な自動化のいずれかが必要になる場合があります。

#### 14.4.7. Tang サーバーの喪失

同じキー情報が使用されている負荷分散サーバー内の個別の Tang サーバーがなくなると、クライアントからも透過的に確認できます。

同じ URL に関連付けられているすべての Tang サーバー (負荷分散されたセット全体) で一時的に障害が発生すると、ネットワークセグメントの損失と同じと考えることができます。既存クライアントには、事前に設定された Tang サーバーが利用可能な限り、ディスクパーティションを復号化できます。これらのサーバーのいずれかが再びオンラインになるまで、新規クライアントは登録できません。

サーバーを再インストールするか、バックアップからサーバーを復元して、Tang サーバーの物理的な損失を軽減できます。キー情報のバックアップおよび復元プロセスが、権限のないアクセスから適切に保護されていることを確認します。

#### 14.4.8. 危険キー情報の再設定

Tang サーバーの物理的な移動や関連データなど、承認されていないサードパーティーにキーが公開される可能性のある場合は、キーをすぐにローテーションします。

##### 手順

1. 影響を受けた情報を保持する Tang サーバーのキーを再生成します。
2. Tang サーバーを使用してすべてのクライアントのキーを再生成します。
3. 元のキー情報を破棄します。
4. マスター暗号化キーが公開されてしまうという意図しない状況を精査します。可能な場合は、ノードをオフラインにし、ディスクを再暗号化します。

##### ヒント

同じ物理ハードウェアへの再フォーマットおよび再インストールは時間がかかりますが、自動化およびテストが簡単です。