OpenShift Container Platform 4.5

セキュリティー

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

Last Updated: 2023-05-22

OpenShift Container Platform 4.5 セキュリティー

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

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

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

目次

第1章 コンテナーのセキュリティー

1.1. コンテナーのセキュリティーについて

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

- コンテナーのセキュリティーは、信頼できるベースコンテナー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ー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ー更新は自動的に行われないため、セキュリティー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。

- コンテナーがデプロイされると、そのセキュリティー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依存し、コンテナー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。

- セキュリティーを継続して保護できるかどうかは、コンテナー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ー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ー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 で有効なコンテナー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、コンテナー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ー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。

- コンテナー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ーのセキュリティー対策と OpenShift Container Platform で提供されるコンテナーのセキュリティー対策。

- 脆弱性についてコンテナーのコンテンツとソースを評価する方法。

- コンテナー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ー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 のコンテナー化ソリューション。

本書の目的は、コンテナー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ー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。

1.1.1. コンテナーについて

コンテナーは、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。コンテナーは、他のコンテナー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。

コンテナーは、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。

コンテナー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	コンテナー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 コンテナーについての詳細は、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 コンテナーについての詳細は、RHEL 製品ドキュメントの Building, running, and managing containers を参照してください。

1.1.2. OpenShift Container Platform について

コンテナー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ー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、ベースコンテナーイメージ、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ー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

1.2. ホストおよび仮想マシンのセキュリティーについて

コンテナー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ー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。

1.2.1. Red Hat Enterprise Linux CoreOS (RHCOS) でのコンテナーのセキュリティー保護

コンテナーは、それぞれのコンテナーを起動するために同じカーネルおよびコンテナー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、コンテナーは特殊なタイプのプロセスに過ぎないため、コンテナー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。コンテナーを実行する環境は、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ー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、複数のコンテナー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 4.5 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 で実行中のコンテナー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、複数のコンテナー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ーの 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のあるコンテナーには、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 7 コンテナーのドキュメントの 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、コンテナーを相互に、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。

- **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 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 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
- SSC (Security Context Constraints) の管理
- 利用可能なプラットフォーム
- 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する場合のクラスターのマシン要件
- 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 Container Platform クラスターの更新

1.2.2. 仮想化とコンテナーの比較

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ー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ー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。

コンテナー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、コンテナー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ーの分離は、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。コンテナーは通常、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。

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

1.2.3. OpenShift Container Platform のセキュリティー保護

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ー関連の設定は、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。

セキュリティーが強化され、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ー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 クラスター更新の多くの部分を自動化できます。

1.3. 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ー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。

1.3.1. RHCOS でのハードニングの内容の選択

RHEL 8 セキュリティー強化 ガイドでは、RHEL システムのセキュリティー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。

1.3.2. 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、ワーカーノードやマスター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、およびクラスターの稼働後に RHCOS を変更することができます。

1.3.2.1. 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 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。

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。

1.3.2.2. インストール時のハードニング

OpenShift 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。

1.3.2.3. 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。

- デーモンセット: 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 Red Hat Enterprise Linux CoreOS (RHCOS) マシンの作成
- 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**
- FIPS 暗号のサポート
- RHEL コア crypto コンポーネント

1.4. コンテナーイメージの署名

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、モニターリング、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。

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

コンテナーの署名の検証を有効にするには、レジストリー URL を sigstore にリンクし、次にイメージを検証するキーを指定するためのファイルが必要です。

手順

1. レジストリー URL を sigstore にリンクし、イメージの検証に使用するキーを指定するためのファイルを作成します。
   - **policy.json** ファイルを作成します。

```
$ cat > policy.json <<EOF
{
  "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"
        }
      ]
    }
  }
}
EOF
```

- **registry.access.redhat.com.yaml** ファイルを作成します。

```
$ cat <<EOF > registry.access.redhat.com.yaml
docker:
    registry.access.redhat.com:
        sigstore: https://access.redhat.com/webassets/docker/content/sigstore
EOF
```

- **registry.redhat.io.yaml** ファイルを作成します。

```
$ cat <<EOF > registry.redhat.io.yaml
docker:
    registry.redhat.io:
        sigstore: https://registry.redhat.io/containers/sigstore
EOF
```

2. マシン設定テンプレートに使用される **base64** でエンコードされた形式でファイルを設定します。

```
$ export ARC_REG=$( cat registry.access.redhat.com.yaml | base64 -w0 )
$ export RIO_REG=$( cat registry.redhat.io.yaml | base64 -w0 )
$ export POLICY_CONFIG=$( cat policy.json | base64 -w0 )
```

3. エクスポートされたファイルをワーカーノードのディスクに書き込むマシン設定を作成します。

```
$ cat > 51-worker-rh-registry-trust.yaml <<EOF
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 51-worker-rh-registry-trust
spec:
  config:
    ignition:
      config: {}
      security:
        tls: {}
      timeouts: {}
      version: 2.2.0
    networkd: {}
    passwd: {}
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-8;base64,${ARC_REG}
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/containers/registries.d/registry.access.redhat.com.yaml
      - contents:
          source: data:text/plain;charset=utf-8;base64,${RIO_REG}
          verification: {}
        filesystem: root
```

```
      mode: 420
      path: /etc/containers/registries.d/registry.redhat.io.yaml
    - contents:
        source: data:text/plain;charset=utf-8;base64,${POLICY_CONFIG}
        verification: {}
      filesystem: root
      mode: 420
      path: /etc/containers/policy.json
  osImageURL: ""
EOF
```

4. 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
```

5. エクスポートしたファイルをマスターノードのディスクに書き込むマシン設定を作成します。

```
$ cat > 51-master-rh-registry-trust.yaml <<EOF
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 51-master-rh-registry-trust
spec:
  config:
    ignition:
      config: {}
      security:
        tls: {}
      timeouts: {}
      version: 2.2.0
    networkd: {}
    passwd: {}
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-8;base64,${ARC_REG}
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/containers/registries.d/registry.access.redhat.com.yaml
      - contents:
          source: data:text/plain;charset=utf-8;base64,${RIO_REG}
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/containers/registries.d/registry.redhat.io.yaml
      - contents:
          source: data:text/plain;charset=utf-8;base64,${POLICY_CONFIG}
          verification: {}
        filesystem: root
        mode: 420
```

```
        path: /etc/containers/policy.json
  osImageURL: ""
EOF
```

6. マスターマシン設定の変更をクラスターに適用します。

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

1.4.2. 署名の検証設定の確認

マシン設定をクラスターに適用すると、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-d9b3f4ffcfd65c30dcf591a0e8cf9b2e
    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。

3. 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": {
      "": [
        {
          "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
```

1.5. 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。

1.5.1. 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。

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

1.6. コンテナーのコンテンツのセキュリティー保護

コンテナー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。コンテナー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。

1.6.1. コンテナー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ー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。

- コンテナー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) でコンテナーをビルドすることにより、コンテナー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。

コンテナー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、Container Security Operator は、OpenShift Container Platform で使用されるコンテナーイメージをチェックするために追加できます。

1.6.2. UBI を使用した再配布可能なイメージの作成

コンテナー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、独自のコンテナー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ー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ー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ー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-mimimal**) 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ー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ー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 **rhscl/** 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 (**rhscl/httpd-***)、 Python (**rhscl/python-***)、Ruby (**rhscl/ruby-***)、Node.js (**rhscl/nodejs-***) および Perl (**rhscl/perl-***) rhscl 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。

1.6.3. 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。

1.6.3.1. 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ーイメージの場合、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。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 のコンテナー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。

1.6.4. 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。

1.6.4.1. 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1.1 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
qualityType	メタデータのタイプ	**vulnerability** **license** **operations** **policy**

コンポーネント	説明	許可される値
providerId	プロバイダー ID の文字列	**openscap** **redhatcatalog** **redhatinsights** **blackduck** **jfrog**

1.6.4.1.1. 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1.2 アノテーション値の形式

フィールド	必須？	説明	タイプ
name	はい	プロバイダーの表示名	文字列
timestamp	はい	スキャンのタイムスタンプ	文字列
description	いいえ	簡単な説明	文字列
reference	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
scannerVersion	いいえ	スキャナーバージョン	文字列
compliant	いいえ	コンプライアンスの合否	ブール値
summary	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1.3 要約フィールド値の形式

フィールド	説明	タイプ
label	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

フィールド	説明	タイプ
data	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
severityIndex	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 **0..3** の範囲内にあり、**0** = low になります。	整数
reference	情報ソースの URL または詳細情報。オプション。	文字列

1.6.4.1.2. 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 のコンテナー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 }
  ]
}
```

1.6.4.2. 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が、セキュリティー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。

1.6.4.2.1. 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 } ]" }'
```

1.6.4.3. 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。

1.6.4.3.1. アノテーションの例

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

1.6.4.4. 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 4.5 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。

1.6.4.4.1. 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>:8443/oapi/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

1.7. コンテナーレジストリーのセキュアな使用

コンテナーレジストリーは、以下を実行するためにコンテナー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ーに整理する
- 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ー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 などのコンテナー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、コンテナーの正常性を確認し、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ー脆弱性のスキャン、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。

1.7.1. コンテナーのソースの確認

ダウンロード済みかつデプロイ済みのコンテナー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、コンテナー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ー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。

1.7.2. イミュータブルで認定済みのコンテナー

イミュータブルなコンテナー 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ーは、実行中には変更されることのないコンテナーです。イミュータブルなコンテナーをデプロイする場合には、実行中のコンテナーにステップインして 1 つ以上のバイナリー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ーイメージを再ビルド、再デプロイし、コンテナーを変更するのではなく、コンテナー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ー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 認定コンテナー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。

1.7.3. Red Hat レジストリーおよび Ecosystem Catalog からのコンテナーの取得

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ァリングの認定コンテナー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ーレジストリー (**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、同じコンテナーイメージのセットが含まれます。

Red Hat ではコンテナー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 がセキュリティー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ー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ーのセキュリティー上のリスクを考慮します。コンテナーは 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、セキュリティーのレベルは、コンテナーが古いと低くなり、新規のコンテナーの場合はセキュリティーのレベルが上がります。

コンテナーの年数について、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ー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ー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 セキュリティーアドバイザリー 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。

1.7.4. OpenShift Container レジストリー

OpenShift Container Platform には、コンテナー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、どのコンテナー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

1.7.5. Red Hat Quay を使用したコンテナーの保存

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ー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 のセキュリティー関連機能には、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** (リポジトリーのミラーリング): セキュリティー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ーをホストしたり、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 セキュリティースキャン)**: 各コンテナー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。各コンテナー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 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。

1.8. ビルドプロセスのセキュリティー保護

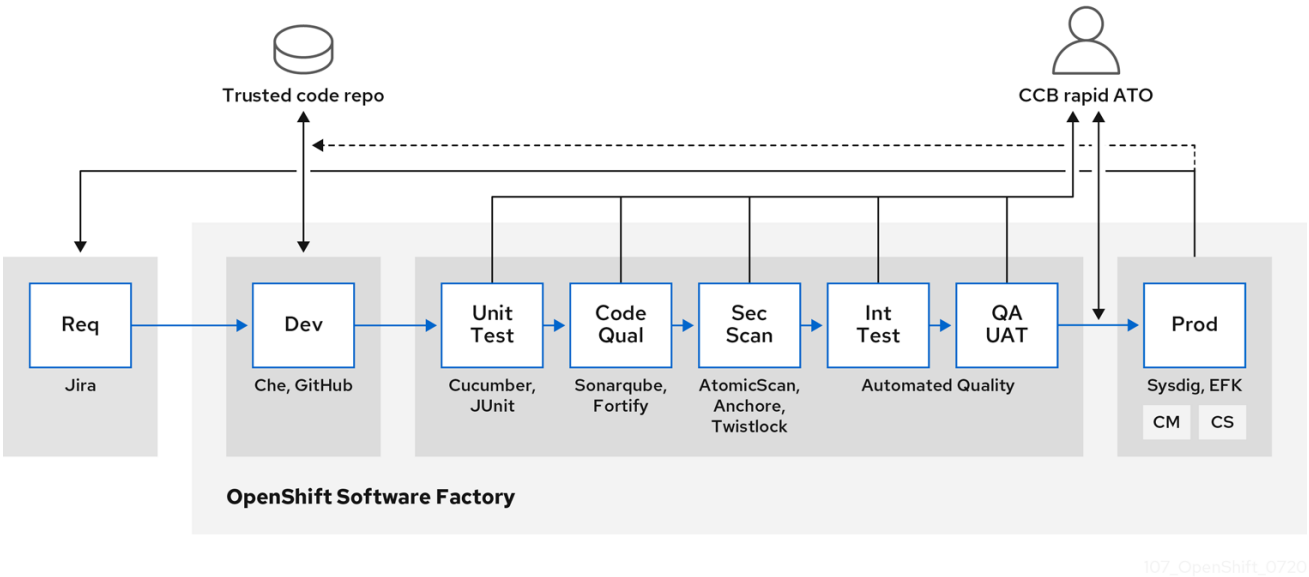
コンテナー環境では、ソフトウェアのビルドプロセスはライフサイクルのステージであり、ここでは、アプリケーションコードが必要なランタイムライブラリーと統合されます。このビルドプロセスの管理は、ソフトウェアのスタックのセキュリティーを保護する上で鍵となります。

1.8.1. 1回のビルドでどこにでもデプロイが可能

OpenShift Container Platform をコンテナー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ーを確保できます。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で実稼働にデプロイされるようにすることができます。

コンテナーのイミュータブルな状態を維持することも重要です。実行中のコンテナー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、コンテナー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。



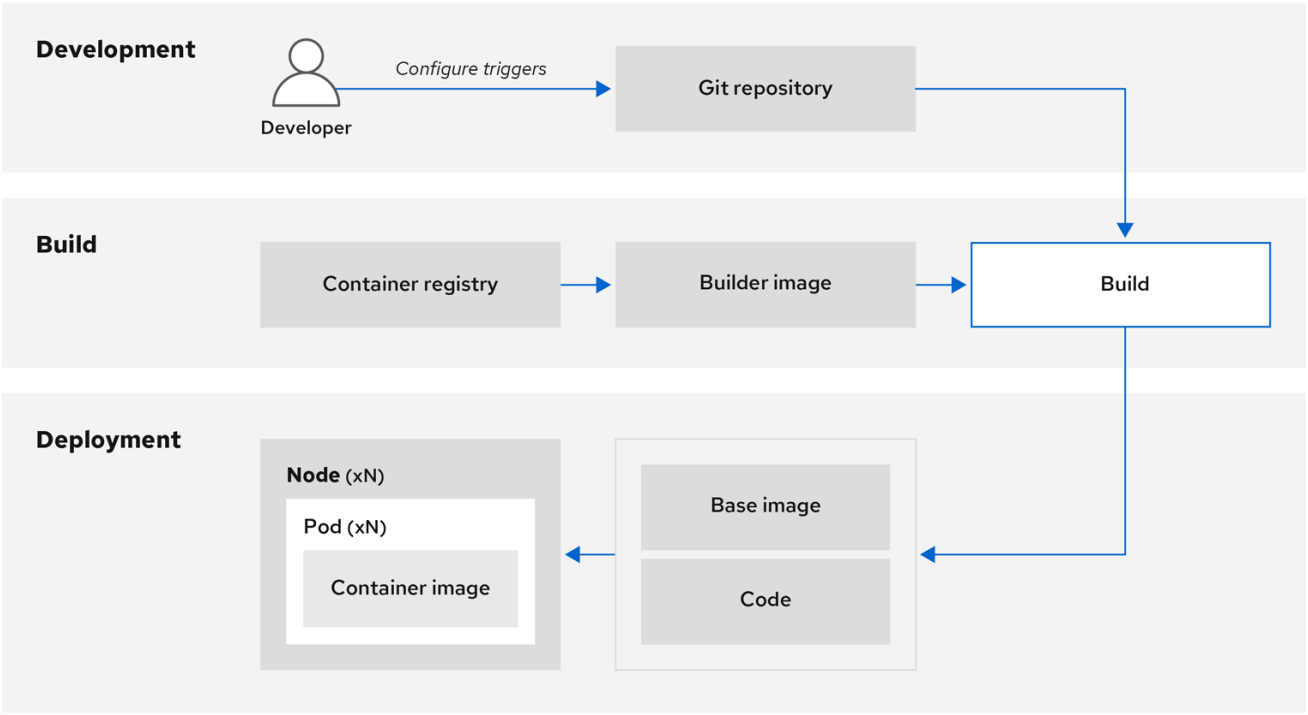
OpenShift Container Platform は、セキュアなコードを作成し、管理できるように、信頼できるコードリポジトリー (GitHub など) および開発プラットフォーム (Che など) と統合できます。単体テストは、Cucumber および JUnit に依存する必要がある場合があります。コンテナーの脆弱性の有無や、Anchore または Twistlock などのコンプライアンス関連の問題の有無を検査し、AtomicScan または Clair などのイメージスキャンツールを使用できます。Sysdig などのツールは、コンテナー化されたアプリケーションの継続的なモニターリ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

1.8.2. 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ることができます。**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。

- コードリポジトリーの 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。



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。

1.8.3. ビルド時の入力のセキュリティー保護

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。セキュリティー上の理由により、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。

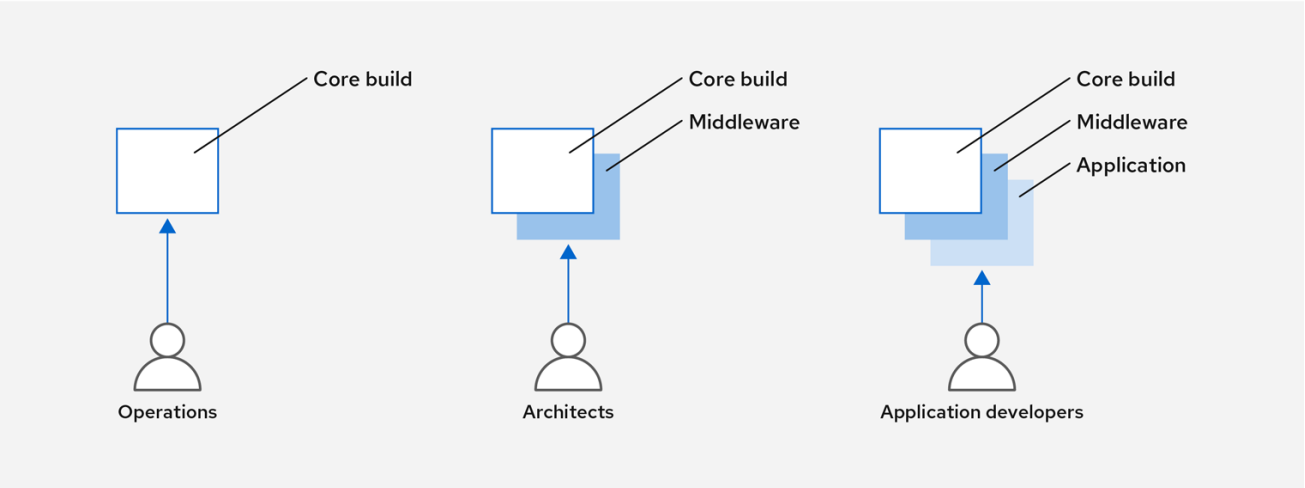
```
source:
  git:
    uri: https://github.com/sclorg/nodejs-ex.git
  secrets:
  - destinationDir: .
    secret:
      name: secret-npmrc
```

3. 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
```

1.8.4. ビルドプロセスの設計

コンテナーの層を使用できるようにコンテナー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、コンテナー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、コンテナー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ーに署名することで、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。

1.8.5. Knative サーバーレスアプリケーションのビルド

Kubernetes および Kourier を使用すると、OpenShift Container Platform で Knative を使用してサーバーレスアプリケーションをビルドし、デプロイし、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ーをビルドしてから、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。

関連情報

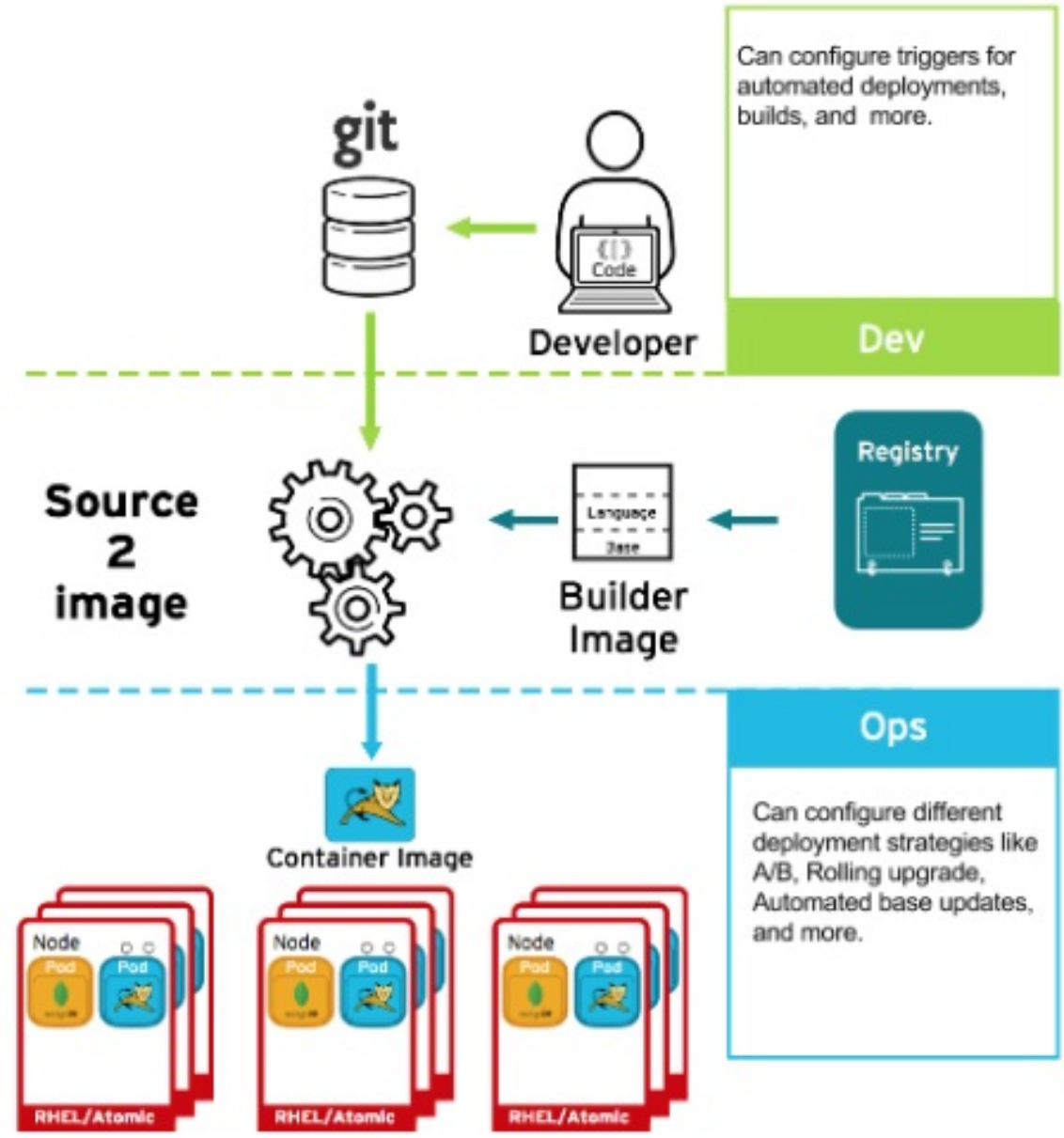
- イメージビルドについて
- ビルドのトリガーおよび変更
- ビルド入力の作成
- 入力シークレットおよび設定マップ
- CI/CD の方法論および実践
- Knative Serving アーキテクチャー
- Topology ビューを使用したアプリケーション設定の表示

1.9. コンテナーのデプロイ

各種の手法を使用して、デプロイするコンテナー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ー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。

1.9.1. トリガーによるコンテナー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ーのプロセスを確認できます。 実行中のコンテナー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 つのコンテナーイメージ層 (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
```

1.9.2. 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できる必要があります。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ー要件を反映した (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。

1.9.3. 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。任意のコンテナー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** とも呼ばれます。

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ー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。

1.9.4. 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ー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ー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。

1.9.5. 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

1.10. コンテナー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ー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ーを分離します。
- 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。

1.10.1. マルチテナンシーによるコンテナーの分離

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のできるデフォルトのクラスターロールがあります。

1.10.2. 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。
- 変更用の受付プラグイン: これらのプラグインは受付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。

1.10.2.1. 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。

- 特権付きコンテナーの実行
- コンテナー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
- コンテナーの SELinux コンテキスト

- コンテナーのユーザー 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。

1.10.2.2. 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:

- スコープ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。

1.10.3. 認証および認可

1.10.3.1. OAuth を使用したアクセスの制御

コンテナープラットフォームのセキュリティーを保護にするために、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 **アイデンティティープロバイダー** 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ー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。

1.10.3.2. 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ー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ー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 (コンテナー化された 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。

1.10.3.3. 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。

1.10.3.4. 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。

1.10.4. 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

1.10.4.1. 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
- SSC (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
- アイデンティティープロバイダー設定について
- 証明書の種類および説明
- プロキシー証明書

1.11. 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、コンテナーが他の 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。コンテナー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。

1.11.1. 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ー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。

1.11.2. 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

1.11.3. 複数の Pod ネットワークの使用

実行中の各コンテナーには、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

1.11.4. 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

1.11.5. 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

1.11.6. 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

1.12. 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。

1.12.1. 永続ボリュームプラグイン

コンテナーは、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。

1.12.2. 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。

1.12.3. 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ーにのみ所有され、このコンテナー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

1.13. クラスターイベントとログの監視

OpenShift Container Platform クラスター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ー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ーレベルの情報として使用できます。

1.13.1. クラスターイベントの監視

クラスター管理者は、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 (クラスターイベントの場合) のいずれかの namespace に関連付けられます。デフォルト の namespace は、クラスター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ーを提供しません。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。

1.13.2. ロギング

oc log コマンドを使用して、コンテナーログ、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、すべてのコンテナー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、コンテナー、監査ログを収集し、管理し、表示できます。Elasticsearch Operator および Cluster Logging Operator を使用してクラスターロギングをデプロイし、管理し、アップグレードできます。

1.13.3. 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
- クラスターロギングについて
- 監査ログの表示

第2章 証明書の設定

2.1. デフォルトの INGRESS 証明書の置き換え

2.1.1. 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が、ここで想定されるリスクは最小限度のものです。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、コンテナー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。

2.1.2. 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ー設定を更新します。

```
$ 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。

2.2. 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。

2.2.1. 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。リバースプロキシー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. **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。

2. 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。

3. **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>
...
```

2.3. 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

2.3.1. 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 ヵ月間有効であり、有効期間が 13 ヵ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
```

2.3.2. 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ーを保護するには、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。

重要

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。

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> \ 1
    service.beta.openshift.io/serving-cert-secret-name=<secret_name> 2
```

1 **<service_name>** を、セキュリティー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>
```

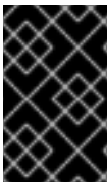
出力例

```
...
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。

2.3.3. 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
```

2. 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-----
...
```

2.3.4. 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

1. 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
```

(1) **<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
```

2. 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
```

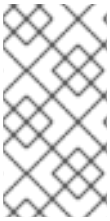
CA バンドルは YAML 出力の **spec.caBundle** フィールドに表示されます。

```
apiVersion: apiregistration.k8s.io/v1
```

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

2.3.5. 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

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ー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>
...
```

2.3.6. 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. 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
```

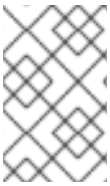
CA バンドルは、YAML 出力のすべての Webhook の **clientConfig.caBundle** フィールドに表示されます。

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

```
  clientConfig:
    caBundle: <CA_BUNDLE>
...
```

2.3.7. 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>
...
```

2.3.8. 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 **serving-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> 1
```

1 **<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
```

2.3.9. 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 ヵ月間有効で、有効期間が 13 ヵ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するようになります。

```
$ 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。

第3章 証明書の種類および説明

3.1. API サーバーのユーザーによって提供される証明書

3.1.1. 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。

3.1.2. 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。

3.1.3. 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。

3.1.4. 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。

3.1.5. 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

3.2. プロキシー証明書

3.2.1. 目的

プロキシー証明書により、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。

プロキシーオブジェクトの **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ーされる 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、プロキシー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ーの設定

3.2.2. インストール時のプロキシー証明書の管理

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ー信頼 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-----
...
```

3.2.3. 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、これはプロキシーでは必要ありません。プロキシーは 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、プロキシー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。

完全なプロキシーサポートとは、指定されたプロキシーに接続し、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。

3.2.4. 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。

3.2.5. 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。

3.2.6. 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。

3.2.7. 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: 2.2.0
    storage:
      files:
      - contents:
```

```
        source: data:text/plain;charset=utf-
8;base64,LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVORENDQXh5Z0F3SUJBZ0lKQU5
1bkkwRDY2MmNuTUEwR0NTcUdTSWIzRFFFQkN3VUFNSUdsTVFzd0NRWUQKV1FRR0V3SlZVek
VYTUJVR0ExVUVDQXdPVG05eWRHZ2dRMkZ5YjJ4cGJtRXhFREFPQmdOVkJBY01CMUpoYkdWcA
pBMmd4RmpBVUJnTlZCQW9NRFZKbFpDQklZWFFzSUVsdVl5NHhFekFSQmdOVkJBc01DbEpsWk
NCSVlYUWdTVlF4Ckh6QVpCZ05WQkFNTUVsSmxaQ0JJWVhRZ1NWUWdVbTl2ZENCRFFURWhN
QjhHQ1NxR1NJYjNEUUVKQVJZU2FXNW0KWGpDQnBURUxNQWtHQTFVRUJoTUNWVk14RnpBV
kJnTlZCQWdNRGs1dmNuUm9JRU5oY205c2FXNWhNUkF3RGdZRApXUVFIREFkU1lXeGxhV2RvTV
JZd0ZBWURWUVFLREExU1pXUWdTR0YwTENCSmJtTXVNUk13RVFZRFZRUUxEQXBTCkFXUWd
TR0YwSUVsVU1Sc3dHUVlEVlFRRERCSlNaV1FnU0dGMElFbFVJRkp2YjNRZ1EwRXhJVEFmQmdrc
WhraUcKMHcwQkNRRVdFbWx1Wm05elpXTkFjbVZrYUdGMExtTnZiVENDQVNJd0RRWUpLb1pJaH
ZjTkFRRUJCUUFEZ2dFUApCRENDQVFvQ2dnRUJBTFF0OU9KUWg2R0M1TFQxZzgwcU5oMHU1
MEJRNHNaL3laOGFFVHh0KzVsblBWWDZNSEt6CmQvaTdsRHFUZlRjZkxMMm55VUJkMmZRRGsx
QjBmeHJza2hHSUlaM2lmUDFQczRsdFRrdjhoUlNvYjNWdE5xU28KSHhrS2Z2RDJQS2pUUHhEUFdZ
eXJ1eTlpckxaaW9NZmZpM2kvZ0N1dDBaV3RBeU8zTVZINXFXRi9lbkt3Z1BFUwpZOXBvK1RkQ3ZS
Qi9SVU9iQmFNNzYxRWNyTFNNMUdxSE51ZVNmcW5obzNBakxRNmRCblBXbG82MzhabTFWZWJ
LCkNFTHloa0xXTVNGa0t3RG1uZTBqUTAyWTRnMDc1dkNLdkNzQ0F3RUFBYU5qTUdFd0hRWUR
WUjBPQkJZRUZIN1IKNXlDK1VlaElJUGV1TDhacXczUHpiZ2NaTUI4R0ExVWRJd1FZTUJhQUZIN1I0
eUMrVWVoSUlQZXVMOFpxdzNQegpjZ2NaTUE4R0ExVWRFd0VCL3dRRk1BTUJBZjh3RGdZRFZS
MFBBUUgvQkFRREFnR0dNQTBHQ1NxR1NJYjNEUUVCCkR3VUFBNElCQVFCRE52RDJWbTlzQT
VBOUFsT0pSOCtlbjVYejloWGN4SkI1cGh4Y1pROGpGb0cwNFZzaHZkMGUKTUVuVXJNY2ZGZ0laN
G5qTUtUUUNNNFpGVVBBaWV5THg0ZjUySHVEb3BwM2U1SnlJTWZXK0tGY05JcEt3Q3NhawpwU2
9LdElVT3NVSks3cUJWWnhjckl5ZVFWMnFjWU9lWmh0UzV3QnFJd09BaEZ3bENFVDdaZTU4UUhtUz
Q4c2xqCjVlVGtSaml2QWxFeHJGektjbGpDNGF4S1Fsbk92VkF6eitHbTMyVTB4UEJGNEJ5ZVBWeEN
KVUh3MVRzeVRtZWwKU3hORXA3eUhvWGN3bitmWG5hK3Q1SldoMWd4VVp0eTMKLS0tLS1FTkQ
gQ0VSVElGSUNBVEUtLS0tLQo=
      filesystem: root
      mode: 0644
      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。

3.2.8. 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。

3.3. サービス CA 証明書

3.3.1. 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。

3.3.2. 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 ヵ月間有効であり、有効期間までの残りの期間が 13 ヵ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 ヵ月は、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。

3.3.3. 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。

3.3.4. 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

3.4. ノード証明書

3.4.1. 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。

3.4.2. 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

3.5. ブートストラップ証明書

3.5.1. 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。

3.5.2. 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。

3.5.3. 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。

3.5.4. 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。

3.6. ETCD 証明書

3.6.1. 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。

3.6.2. 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。

3.6.3. 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。

3.6.4. 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、プロキシー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ーに接続します。

追加リソース

- マスターホストの失われた状態からのリカバリー

3.7. OLM 証明書

3.7.1. 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 はプロキシー環境で管理する 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。

3.8. デフォルト INGRESS のユーザーによって提供される証明書

3.8.1. 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.hello-openshift** はルートの名前で、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。

3.8.2. 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。このプロセスについての詳細は、カスタムデフォルト証明書の設定 を参照してください。

3.8.3. 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。

3.8.4. 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。

3.8.5. 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。

3.8.6. 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

3.9. INGRESS 証明書

3.9.1. 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。

3.9.2. 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ー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。

3.9.3. ワークフロー

図3.1 カスタム証明書のワークフロー

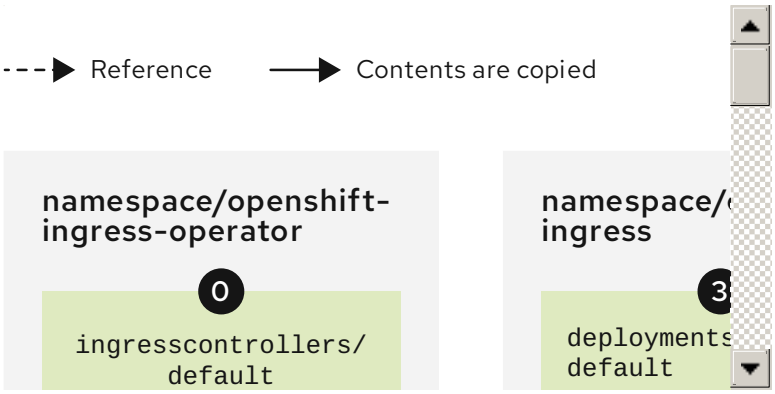
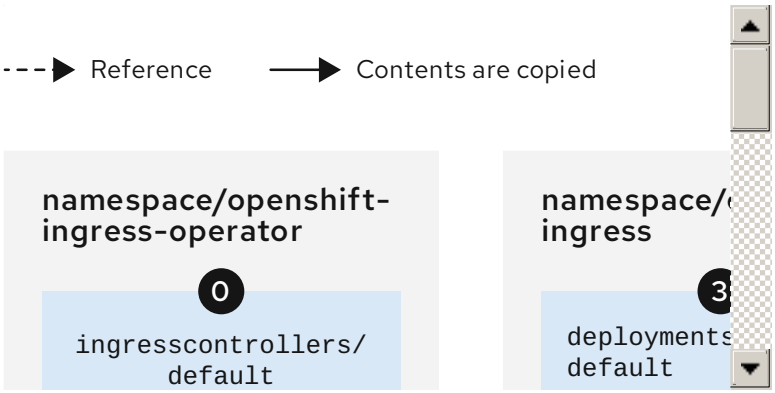


図3.2 デフォルトの証明書ワークフロー



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ー設定を更新します。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 4.5 以降では、**default-ingress-cert** を使用します。

3.9.4. 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。

3.9.5. サービス

Prometheus はメトリクスのセキュリティー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。

セキュリティー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。

セキュリティー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。

3.9.6. 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。

3.9.7. 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。

3.10. 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

3.10.1. 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ーを保護します。これらの証明書は 2 年間有効であり、13 ヵ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。

3.10.2. 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。

3.11. コントロールプレーンの証明書

3.11.1. 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

3.11.2. 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4章 監査ログの表示

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。

4.1. 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4.1監査ログフィールド

フィールド	説明
level	イベントが生成された監査レベル。
auditID	要求ごとに生成される一意の監査 ID。
stage	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。
requestURI	クライアントによってサーバーに送信される要求 URI。
verb	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。
user	認証されたユーザーの情報。
impersonatedUser	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)。
sourceIPs	オプション。要求の送信元および中間プロキシーからのソース IP。
userAgent	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。
objectRef	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、**List** 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。
responseStatus	オプション。**ResponseObject** が **Status** 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
requestObject	オプション。JSON 形式の要求からの API オブジェクト。**RequestObject** 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。
responseObject	オプション。JSON 形式の応答で返される API オブジェクト。**ResponseObject** 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。
requestReceivedTimestamp	要求が API サーバーに到達した時間。
stageTimestamp	要求が現在の監査ステージに達した時間。
annotations	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 **metadata.annotations** 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: **podsecuritypolicy.admission.k8s.io/policy**)。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\""}}
```


4.2. 監査ログの表示

それぞれのマスターノードについて OpenShift Container Platform API サーバーまたは Kubernetes 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。

1. OpenShift Container Platform API サーバーログを表示します。

   a. 必要に応じて、表示する必要のあるログのノード名を取得します。

   ```
   $ oc adm node-logs --role=master --path=openshift-apiserver/
   ```

   出力例

   ```
   ip-10-0-140-97.ec2.internal audit-2019-04-09T00-12-19.834.log
   ip-10-0-140-97.ec2.internal audit-2019-04-09T11-13-00.469.log
   ip-10-0-140-97.ec2.internal audit.log
   ip-10-0-153-35.ec2.internal audit-2019-04-09T00-11-49.835.log
   ip-10-0-153-35.ec2.internal audit-2019-04-09T11-08-30.469.log
   ip-10-0-153-35.ec2.internal audit.log
   ip-10-0-170-165.ec2.internal audit-2019-04-09T00-13-00.128.log
   ip-10-0-170-165.ec2.internal audit-2019-04-09T11-10-04.082.log
   ip-10-0-170-165.ec2.internal audit.log
   ```

   b. 特定のマスターノードとタイムスタンプの OpenShift Container Platform API サーバーのログを表示したり、そのマスターのすべてのログを表示します。

   ```
   $ oc adm node-logs <node-name> --path=openshift-apiserver/<log-name>
   ```

   以下は例になります。

   ```
   $ oc adm node-logs ip-10-0-140-97.ec2.internal --path=openshift-apiserver/audit-2019-04-08T13-09-01.227.log
   ```

   ```
   $ oc adm node-logs ip-10-0-140-97.ec2.internal --path=openshift-apiserver/audit.log
   ```

   以下のような出力が表示されます。

   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\""}}
```

2. Kubernetes API サーバーログを表示します。

   a. 必要に応じて、表示する必要のあるログのノード名を取得します。

   ```
   $ oc adm node-logs --role=master --path=kube-apiserver/
   ```

   出力例

   ```
   ip-10-0-140-97.ec2.internal audit-2019-04-09T14-07-27.129.log
   ip-10-0-140-97.ec2.internal audit-2019-04-09T19-18-32.542.log
   ip-10-0-140-97.ec2.internal audit.log
   ip-10-0-153-35.ec2.internal audit-2019-04-09T19-24-22.620.log
   ip-10-0-153-35.ec2.internal audit-2019-04-09T19-51-30.905.log
   ip-10-0-153-35.ec2.internal audit.log
   ip-10-0-170-165.ec2.internal audit-2019-04-09T18-37-07.511.log
   ip-10-0-170-165.ec2.internal audit-2019-04-09T19-21-14.371.log
   ip-10-0-170-165.ec2.internal audit.log
   ```

   b. 特定のマスターノードとタイムスタンプの Kubernetes API サーバーログを表示したり、そのマスターのすべてのログを表示します。

   ```
   $ oc adm node-logs <node-name> --path=kube-apiserver/<log-name>
   ```

   以下は例になります。

   ```
   $ oc adm node-logs ip-10-0-140-97.ec2.internal --path=kube-apiserver/audit-2019-04-
   09T14-07-27.129.log
   ```

   ```
   $ oc adm node-logs ip-10-0-170-165.ec2.internal --path=kube-apiserver/audit.log
   ```

   以下のような出力が表示されます。

   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\""}}
```

第5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

5.1. 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、OpenShift 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. **additionalCORSAllowedOrigins** 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:
  additionalCORSAllowedOrigins:
  - (?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6章 ETCD データの暗号化

6.1. 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ー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。

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。

6.2. 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

初期の暗号化プロセスが完了するまでは、etcd のバックアップを取ることは推奨されません。暗号化プロセスが完了しない場合、バックアップは部分的にのみ暗号化され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, oauthaccesstokens.oauth.openshift.io, oauthauthorizetokens.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。

   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。

6.3. 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 (1)
```

(1) **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。

第7章 POD の脆弱性のスキャン

Container Security Operator (CSO) 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ー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。CSV

- 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ーを監視します。
- 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ーの出所となったコンテナー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。

この手順を使用すると、CSO は **openshift-operators** namespace にインストールされ、OpenShift クラスターのすべての namespace で利用可能になります。

7.1. 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 Container Security Operator を起動できます。

前提条件

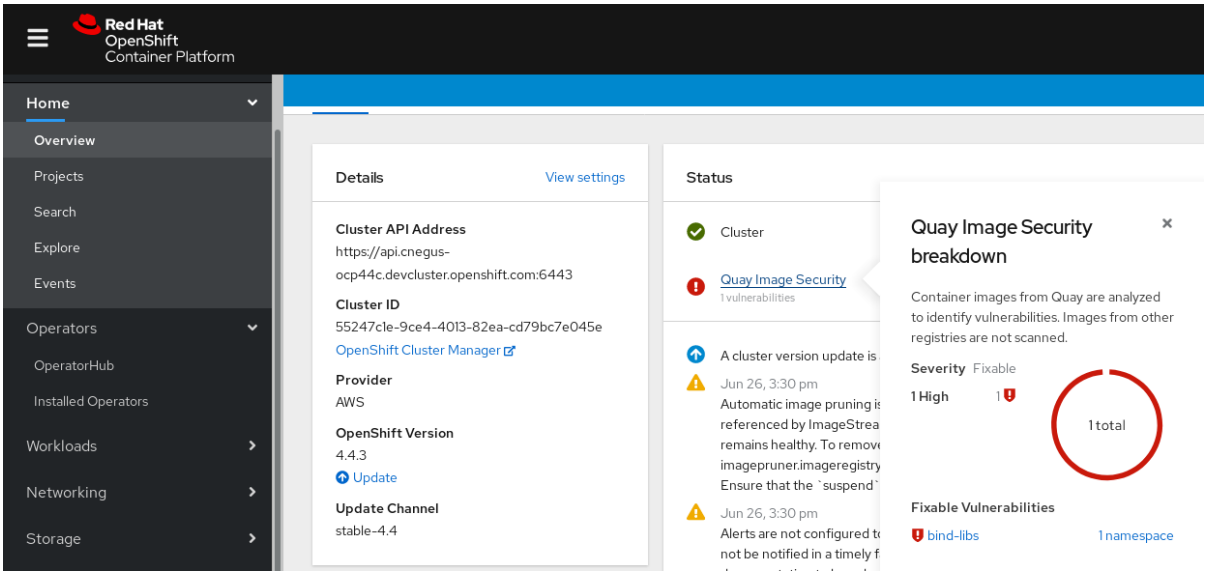
- OpenShift Container Platform クラスターへの管理者権限がある
- クラスターで実行される Red Hat Quay または Quay.io レジストリーのコンテナーがある

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. オプションで、カスタム証明書を CSO 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、以下のコマンドを実行して証明書を CSO に追加します。

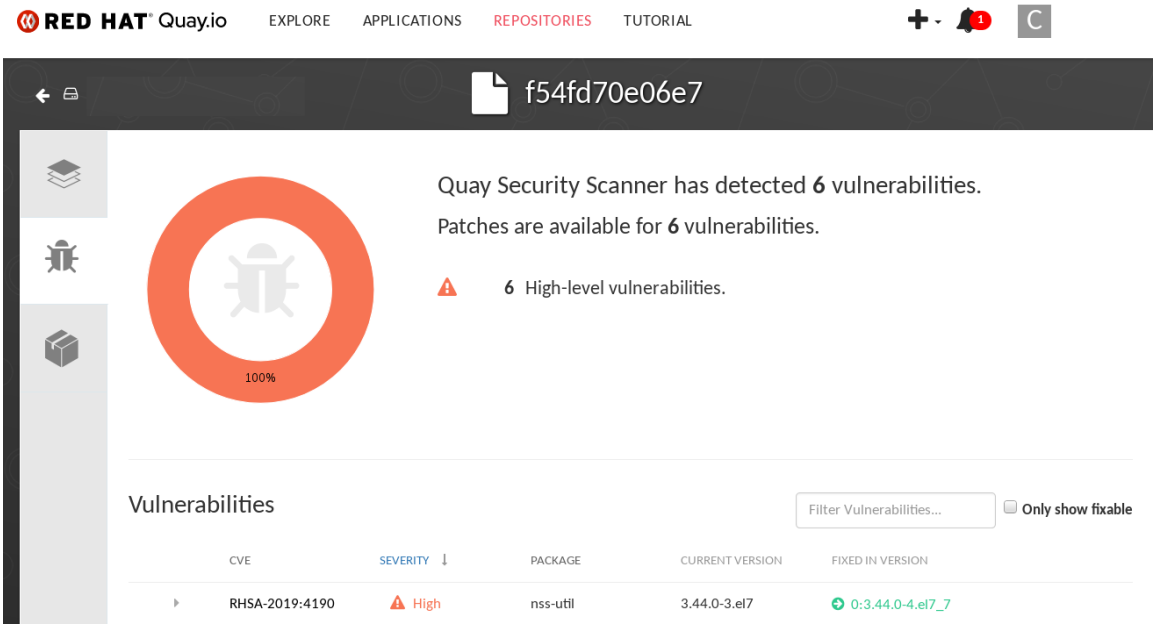
```
$ oc create secret generic container-security-operator-extra-certs --from-file=quay.crt -n
openshift-operators
```

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。

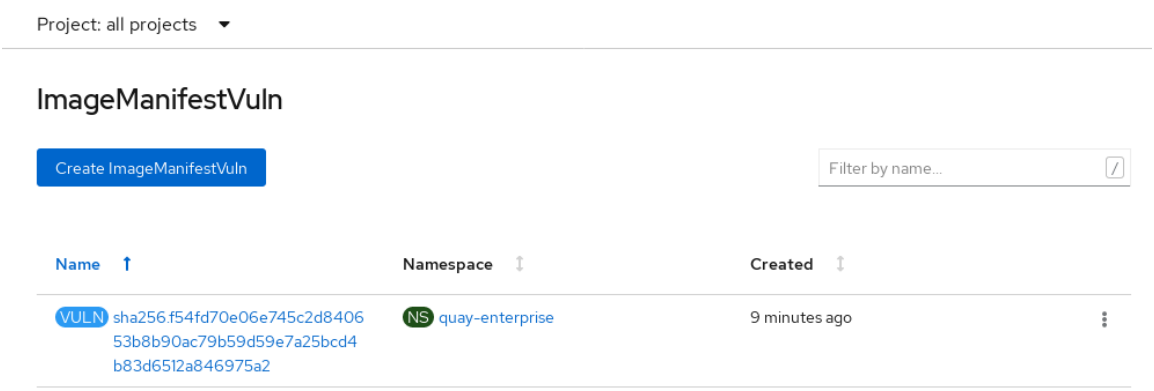


8. この時点で、検出された脆弱性をフォローするために以下の 2 つのいずれかの操作を実行できます。

   - 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。



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。

7.2. 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、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 Container Security Operator が実行されていること

手順

- 検出されたコンテナー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ーを示しています。

```
$ 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...
```