OpenShift Container Platform 3.11

コンテナーセキュリティーガイド

OpenShift Container Platform 3.11 コンテナーセキュリティーガイド

Last Updated: 2023-03-25

OpenShift Container Platform 3.11 コンテナーセキュリティーガイド

OpenShift Container Platform 3.11 コンテナーセキュリティーガイド

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

本書に記載の推奨事項に基づいてクラスターのセキュリティーを保護します。

目次

第1章 はじめに

1.1. 本ガイドについて

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。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 (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。
- 認証および承認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。

1.2. 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、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 OS 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

参考文献

- **OpenShift Container Platform Architecture**: Core Concepts → Containers and Images
- Red Hat Enterprise Linux Atomic Host コンテナーセキュリティーガイド

1.3. OPENSHIFT CONTAINER PLATFORM におけるコンテナーセキュリティー

本書では、コンテナーソリューションスタックの各層でのセキュリティーの重要な要素について説明します。 さらに、各段階および各層でのセキュリティーに留意しつつ OpenShift Container Platform を使用したコンテナーの作成、デプロイ、およびスケーリング時の管理の方法についても説明します。

参考文献

- Red Hat Enterprise Linux Atomic Host: **Overview of Containers in Red Hat Systems**
- **Red Hat Enterprise Linux Atomic Host コンテナーセキュリティーガイド**

第2章 コンテナーのホストおよびマルチテナンシー

2.1. RHEL 上でコンテナーのセキュリティーを保護する方法

コンテナーは、複数のアプリケーションを単一ホストにデプロイすることで、マルチテナンシーのデプロイメントを単純化します。 この際、カーネルおよび Docker ランタイムを使用して各コンテナーをスピンアップします。

ホストカーネルのセキュリティーを保護し、コンテナーを相互にセキュアにすることのできるオペレーティングシステム (OS) を使用する必要があります。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、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。コンテナーは非 root ユーザーとして実行される必要があります。権限レベルを下げることや、可能な限り権限レベルを低くした状態でコンテナーを作成することが推奨されます。

OpenShift Container Platform は Red Hat Enterprise Linux (RHEL) および RHEL Atomic Host で実行されるため、デフォルトで以下の概念がデプロイされる OpenShift Container Platform クラスターに適用されます。 また、これらはプラットフォーム上でコンテナーのセキュリティーを保護する上で中核を成す概念になります。

- **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リソースを同時に使用することができます。namespace のタイプ (例: マウント、PID、およびネットワーク) の詳細は、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。

- **RHEL Atomic Host** 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。

参考文献

- Linux の man ページ: **namespaces(7)**

- **Red Hat Enterprise Linux Atomic Host Overview of Containers in Red Hat Systems**Secure Containers with SELinux

- **Red Hat Enterprise Linux** リソース管理ガイド: コントロールグループについて (CGroup)

- **Red Hat Enterprise Linux Atomic Host** コンテナーセキュリティーガイド: Linux Capabilities and seccomp

- カーネルのドキュメント: **seccomp**

2.2. マルチテナンシー: 仮想化対コンテナー

従来の仮想化もマルチテナンシーを可能にしますが、コンテナーの場合とは全く異なる方法でこれを実行します。仮想化は、ゲスト仮想マシン (VM) をスピンアップするハイパーバイザーを使用します。 仮想マシンはそれぞれ、独自のオペレーティングシステム (OS) のほか、実行するアプリケーションとその依存関係を備えています。

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。とはいえ、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マルチテナンシーはより軽量で、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。

参考文献

- **Red Hat Enterprise Linux Atomic Host Overview of Containers in Red Hat Systems** Linux Containers Compared to KVM Virtualization

第3章 コンテナーのコンテンツ

3.1. 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 OS 層は最新の状態か？

参考文献

- OpenShift Container Platform Using Images
  - これは、OpenShift Container Platform で使用するために Red Hat が提供するフレームワーク、データベース、およびサービスコンテナーイメージに関する参照ドキュメントです。

3.2. コンテナーのコンテンツのスキャン

コンテナースキャンツールは、継続的に更新される脆弱性のデータベースを利用して、コンテナーのコンテンツの既知の脆弱性についての最新情報を常に利用可能にします。既知の脆弱性についての一覧は絶えず変更されるため、コンテナーイメージの初回ダウンロード時にそれらのコンテンツについて確認し、承認済みおよびデプロイ済みのすべてのイメージの脆弱性について継続的に追跡する必要があります。

RHEL はプラグ可能な API を提供し、複数のスキャナーをサポートします。Red Hat CloudForms を OpenSCAP と共に使用して、コンテナーイメージをスキャンしてセキュリティー問題の有無を確認することもできます。RHEL での OpenSCAP についての一般的な情報は、コンプライアンスおよび OpenSCAP を使った脆弱性のスキャン を参照してください。 また、OpenSCAP 統合の詳細については、Red Hat CloudForms Policies and Profiles Guide を参照してください。

OpenShift Container Platform では、このようなスキャナーを 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必要のあ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。詳細は、ビルドプロセス で説明されています。

3.3. 外部のスキャンツールの OPENSHIFT への統合

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。

3.3.1. 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3.1 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**
providerId	プロバイダー ID の文字列	**openscap** **redhatcatalog** **redhatinsights** **blackduck** **jfrog**

3.3.1.1. 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3.2 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 および/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

フィールド	必須？	説明	タイプ
compliant	いいえ	コンプライアンスに合格/不合格	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3.3 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
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 および/または詳細情報。オプション。	String

3.3.1.2. 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 と健全性のインデックスデータを含むイメージの Red Hat Container Catalog 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 Container 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 }
  ]
}
```

3.3.2. 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。

3.3.2.1. 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:fec8a395afe3e804b3db5cb277869142d2b5c561ebb517585566e160ff321988**)。

```
$ oc annotate image <image> \
    quality.images.openshift.io/vulnerability.redhatcatalog='{ \
    "name": "Red Hat Container Catalog", \
    "description": "Container health index", \
    "timestamp": "2016-09-08T05:04:46Z", \
    "compliant": null, \
    "scannerVersion": "1.2", \
    "reference": "https://access.redhat.com/errata/RHBA-2016:1566", \
    "summary": "[ \
      { "label": "Health index", "data": "B", "severityIndex": 1, "reference": null } ]" }'
```

3.3.3. Pod 実行の制御

イメージが実行するかどうかをプログラムで制御するには、**images.openshift.io/deny-execution** イメージポリシーを使用できます。詳細は、イメージポリシー を参照してください。

3.3.3.1. アノテーションの例

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

3.3.4. 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 REST API を呼び出してアノテーションを作成します。REST API およびイメージの更新に使用する PATCH 呼び出しについての一般的な情報は、**REST API Reference** を参照してください。

3.3.4.1. 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 Container Catalog', 'description': 'Container health index', 'timestamp': '2016-
09-08T05:04:46Z', 'compliant': null, 'reference': 'https://access.redhat.com/errata/RHBA-2016:1566',
'summary': [{'label': 'Health index', 'data': '4', 'severityIndex': 1, 'reference': null}] }"
    }
  }
}
```

第4章 レジストリー

4.1. コンテナーの出所

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。

4.2. 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。 更新されたコンテナーイメージを再度ビルドし、デプロイする必要があります。

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ます。

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。

参考文献

- OpenShift Container Platform のイミュータブルなコンテナーに関する詳細情報:
  - **OpenShift Container Platform Architecture**:Image Streams
  - **OpenShift Container Platform** 開発者ガイド: イメージストリームでのイメージの参照

4.3. RED HAT レジストリーおよび RED HAT CONTAINER CATALOG

Red Hat は、**Red Hat** レジストリーを使用して Red Hat 製品およびパートナー企業のオファリングの認定コンテナーを提供します。これは、registry.redhat.io で Red Hat がホストする公開コンテナーレジストリーです。**Red Hat Container Catalog** を使用すると、Red Hat レジストリーで提供されるコンテナーイメージと関連したバグ修正またはセキュリティーアドバイザリーを確認できます。

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 は、Red Hat Container Catalog で提供されるコンテナーのセキュリティーリスクについて健全性のインデックスを使用します。Red Hat は、エラータプロセスでソフトウェアを提供します。 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 Container 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 Container Catalog を参照してください。

参考文献

- Red Hat Container Catalog FAQ
- Red Hat Product Security Center
- Red Hat セキュリティーアドバイザリー

4.4. OPENSHIFT CONTAINER レジストリー

OpenShift Container Platform には、**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 はすでに使用中の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。

参考文献

- **OpenShift Container Platform Architecture**: Infrastructure Components → Container Registry

第5章 ビルドプロセス

5.1. 1回のビルドでどこにでもデプロイが可能

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。

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。

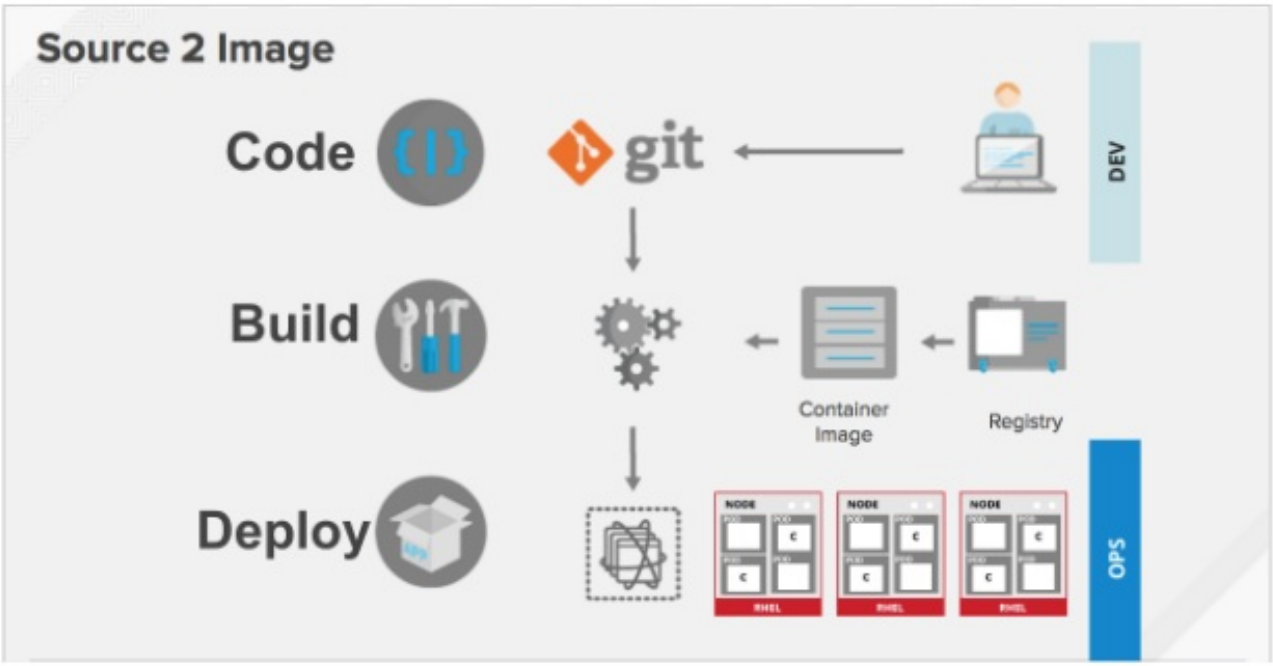
コンテナーのイミュータブルな状態を維持することも重要です。実行中のコンテナーにパッチを当てることはできません。 その代わりに再ビルドおよび再デプロイを実行します。

5.2. ビルドの管理およびセキュリティー

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。

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。



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/ 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。

参考文献

- **OpenShift Container Platform** 開発者ガイド
  - ビルドの仕組み
  - ビルドのトリガー
- **OpenShift Container Platform Architecture**: Source-to-Image (S2I) Build
- **OpenShift Container Platform Using Images**: Other Images → Jenkins

5.3. 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** 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** の **source** セクションに追加します。

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

3. シークレットを新規の **BuildConfig** 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
```

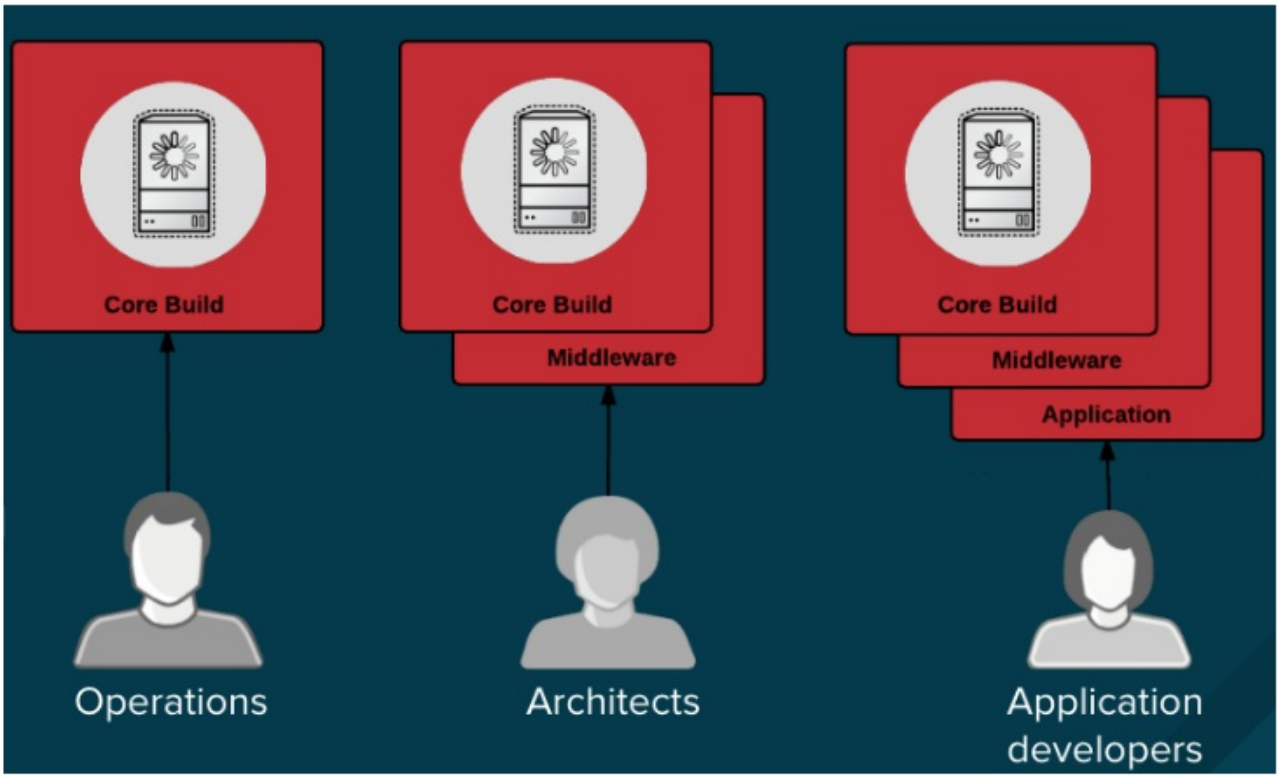
参考文献

- OpenShift Container Platform 開発者ガイド 入力シークレット

- OpenShift Container Platform 開発者ガイド:入力シークレット

5.4. 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。



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。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。

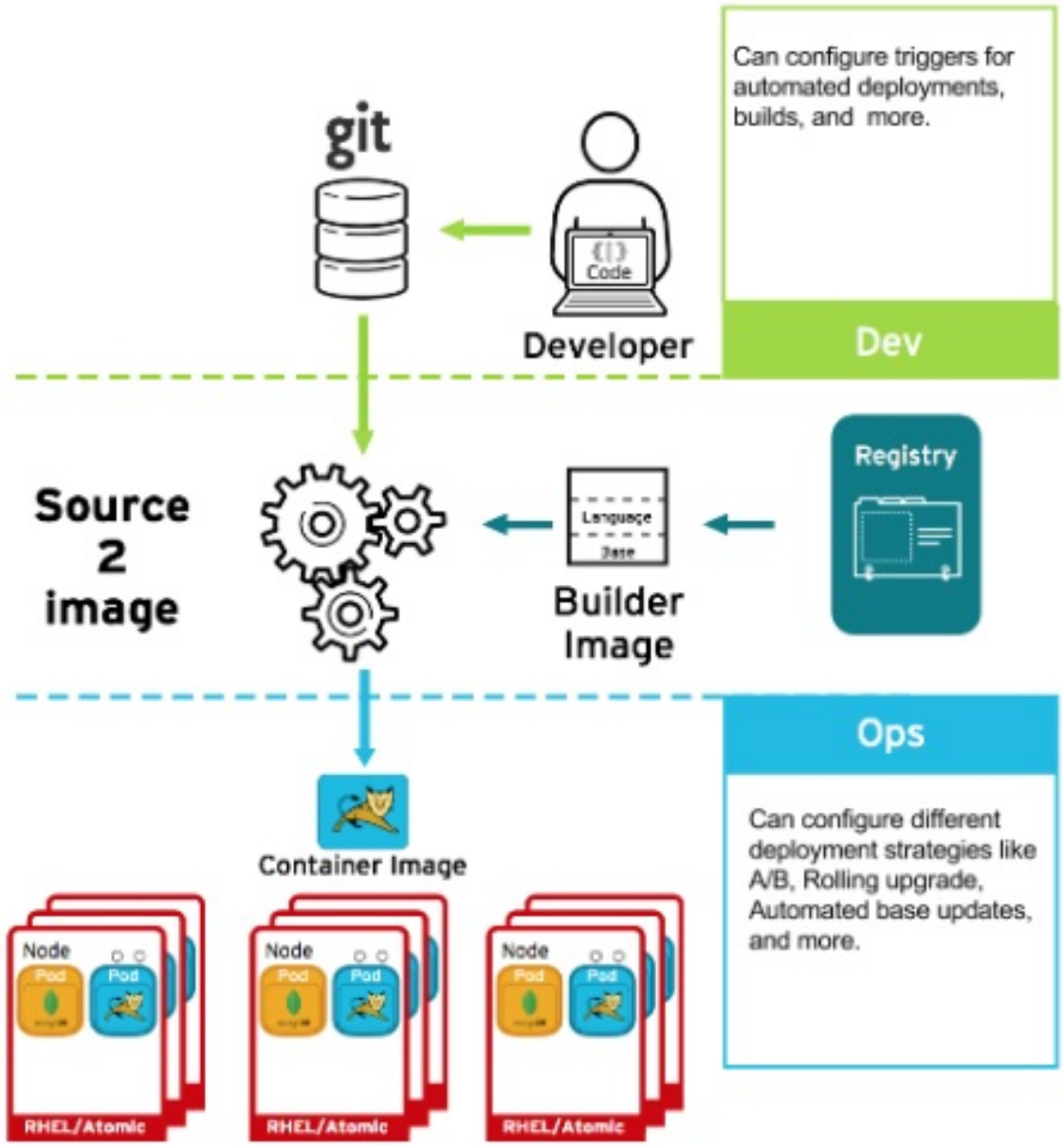
参考文献

- **Red Hat Enterprise Linux Atomic Host Managing Containers**Signing Container Images

第6章 DEPLOYMENT

6.1. コンテナーへのデプロイの制御

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を実行するツールを使用できます。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 コマンドは、デプロイメント設定のデプロイメントトリガーを設定するために使用できます。たとえば、**frontend** というデプロイメント設定に **ImageChangeTrigger** を設定する場合は、以下を実行します。

```
$ oc set triggers dc/frontend \
    --from-image=myproject/origin-ruby-sample:latest \
    -c helloworld
```

参考文献

- **OpenShift Container Platform** 開発者ガイド
  - デプロイメントの仕組み
  - デプロイメントトリガーの設定
  - アプリケーションライフサイクル管理 → 環境全体におけるアプリケーションのプロモート

6.2. 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、その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、またはパブリックキーが必要)

これらのポリシーパラメーターにより、レジストリーまたはレジストリーの一部および個別のイメージもホワイトリスト化 (accept) またはブラックリスト化 (reject) するか、または信頼されたパブリックキーを使用して信頼関係を定義し、ソースが暗号を使って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": "insecureAcceptAnything"}]
      }
   }
}
```

ポリシーは **/etc/containers/policy.json** としてノードに保存できます。この例では、以下のルールを実施しています。

1. Red Hat レジストリー (**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。
2. **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。
3. **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。
4. 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。

ホストの設定に関する詳細な説明については、イメージ署名サポートの有効化 を参照してください。署名トランスポート に関する詳細は、以下のセクションを参照してください。イメージ署名ポリシーに関する詳細は、署名検証ポリシーファイル形式 のソースコードについてのドキュメントを参照してください。

6.2.1. 署名トランスポート

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。任意のコンテナーイメージ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。 署名アクセスプロトコル (Signature access protocol) に応じて各イメージの署名に直接アクセスでき、サーバーディレクトリーのルートにはそのファイル構造が表示されません。

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、Ansible 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。

詳細については、レジストリー設定ディレクトリー (Registries Configuration Directory) または 署名アクセスプロトコル (Signature access protocol) のソースコードについてのドキュメントを参照してください。

参考文献

- **OpenShift Container Platform** クラスター管理ガイド
  - デフォルトスケジューリング
- **Red Hat** ナレッジベース
  - Container Image Signing Integration Guide
- ソースコードのリファレンス
  - Image signing policy
  - Signature transports
  - Signature format

6.3. シークレットと CONFIGMAP

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 Web コンソールを使用してデプロイメント設定に追加するには、以下を実行します。

1. 新しいプロジェクトを作成する。
2. **Resources → Secrets** に移動して、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。
3. デプロイメントの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。

ConfigMap 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。

参考文献

- **OpenShift Container Platform** 開発者ガイド
  - シークレット
  - ConfigMap

6.4. 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 (コンテナーのコレクションである) 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、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。

参考文献

- **OpenShift Container Platform Architecture**: Security Context Constraints
- **OpenShift Container Platform** クラスターのインストール: セキュリティー警告
  - 特権付きコンテナーについて説明されています。

6.5. 継続的デプロイメントのツール

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。

第7章 コンテナープラットフォームのセキュリティー保護

7.1. コンテナーのオーケストレーション

API は、スケーリング時にコンテナー管理を自動化する際に主要なロールを担います。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。

参考文献

- OpenShift Container Platform アーキテクチャー:How Is OpenShift Container Platform Secured?

7.2. 認証および承認

7.2.1. 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 Deny All アイデンティティープロバイダーが使用されます。ただし、これは初回のインストール時またはインストール後に設定することが可能です。アイデンティティープロバイダーの詳細な一覧については、認証とユーザーエージェントの設定 を参照してください。

たとえば、GitHub アイデンティティープロバイダーをインストール後に設定するには、以下を実行します。

1. **/etc/origin/master-config.yaml** で、マスター設定ファイルを編集します。
2. 以下のように **oauthConfig** スタンザを修正します。

```
oauthConfig:
  ...
  identityProviders:
  - name: github
    challenge: false
    login: true
    mappingMethod: claim
    provider:
      apiVersion: v1
      kind: GitHubIdentityProvider
      clientID: ...
      clientSecret: ...
      organizations:
      - myorganization1
      - myorganization2
```

```
teams:
- myorganization1/team-a
- myorganization2/team-b
```

注記

詳細情報と使用方法については、認証とユーザーエージェントの設定の GitHub セクションを参照してください。

3. 変更を保存したら、変更を有効にするためにマスターサービスを再起動します。

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

参考文献

- **OpenShift Container Platform** アーキテクチャー
  - Additional Concepts → Authentication
  - Additional Concepts → Authorization
- **OpenShift Container Platform CLI** リファレンス
- **OpenShift Container Platform** 開発者ガイド:CLI 認証

7.2.2. 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、Running APIcast on Red Hat OpenShift を参照してください。

7.2.3. Red Hat SSO

Red Hat Single Sign-On (RH-SSO) 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 SSO 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 SSO 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。

使用方法についてのチュートリアルは、Red Hat JBoss SSO for OpenShift のドキュメントを参照してください。

7.2.4. 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 はクラスターに直接公開されない。

参考文献

- **OpenShift Container Platform Architecture**:Infrastructure Components → Web Console
- **OpenShift Container Platform** 開発者ガイド:Web コンソール認証

7.3. 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 Ansible ベース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

7.3.1. 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。

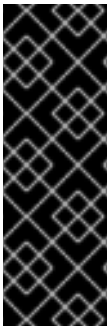
Ansible Playbook を使用したクラスターの初回のインストール時に、カスタム証明書は **openshift_master_overwrite_named_certificates** Ansible 変数を使用して設定できます。以下に例を示します。

```
openshift_master_named_certificates=[{"certfile": "/path/on/host/to/custom1.crt", "keyfile":
"/path/on/host/to/custom1.key", "cafile": "/path/on/host/to/custom-ca1.crt"}]
```

インストール Playbook の実行方法に関するオプションと説明については、カスタム証明書の設定 セクションを参照してください。

インストーラーは、すべてのクラスター証明書の有効期限を確認するための Ansible Playbook を提供し

ます。追加の Playbook は、最新の CA を使用してすべての証明書を一度に自動的に再デプロイしたり、特定の証明書のみを再デプロイしたり、または新たに生成した CA またはカスタム CA を独自に再デプロイしたりできます。これらの Playbook に関する詳細は、証明書の再デプロイ を参照してください。



重要

The **cafile** 証明書は、証明書のインストール時または再デプロイメント時にマスターの **ca-bundle.crt** ファイルにインポートされます。**ca-bundle.crt** ファイルは、OpenShift Container Platform で実行されるすべての Pod にマウントされます。複数の OpenShift Container Platform コンポーネントはデフォルトでは、**masterPublicURL** エンドポイントにアクセスする時に名前付き証明書を自動的に信頼します。certificates パラメーターから **cafile** オプションを省略すると、Web コンソールと他のコンポーネントの複数の機能は削減されます。

参考文献

- OpenShift Container Platform クラスターの設定
  - カスタム証明書の設定
  - 証明書の有効期限のチェック
  - 証明書の再デプロイ

第8章 ネットワークセキュリティー

8.1. ネットワークの NAMESPACE

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。

ネットワークの namespace を使用すると、Pod のネットワークを分離することができます。各 Pod には独自の IP およびバインドするポートの範囲が設定されるため、Pod のネットワークは相互に分離されます。異なるプロジェクトの Pod は、別のプロジェクトの Pod およびサービスとパケットの送受信をすることができなくなります。これを利用して、クラスター内の開発者環境、テスト環境、および実稼働環境を分離することもできます。

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も提供します。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。

参考文献

- **OpenShift Container Platform Architecture**:Networking
- **OpenShift Container Platform** クラスター管理:ネットワークの管理
- **Red Hat Enterprise Linux Atomic Host Managing Containers** Running Super-Privileged Containers

8.2. 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。

たとえば、クラスター内でプロジェクトネットワークを分離するか、またはその逆も行う場合は、以下を実行します。

```
$ oc adm pod-network isolate-projects <project1> <project2>
```

上記の例では、**<project1>** および **<project2>** 内のすべての Pod とサービスは、クラスター内の他の非グローバルプロジェクトの Pod とサービスにはアクセスできず、その逆からもアクセスすることはできません。

第9章 割り当て済みストレージ

9.1. 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。

OpenShift Container Platform は、NFS、AWS Elastic Block Stores (EBS)、GCE Persistent Disks、GlusterFS、iSCSI、RADOS (Ceph)、および Cinder を含む複数のタイプのストレージ向けのプラグインを提供します。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。

PersistentVolume (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。

参考文献

- **OpenShift Container Platform Architecture**:Additional Concepts → Storage
- **OpenShift Container Platform** クラスターの設定:永続ストレージの設定 → ボリュームのセキュリティー

9.2. 共有ストレージ

NFS、Ceph、および Gluster 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。

参考文献

- **OpenShift Container Platform** クラスターの設定
  - NFS を使用した永続ストレージ
  - Ceph RBD を使用した永続ストレージ
  - GlusterFS を使用する永続ストレージ

9.3. BLOCK STORAGE

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。

参考文献

- **OpenShift Container Platform クラスターの設定**
  - 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
  - iSCSI を使用した永続ストレージ

第10章 クラスターイベントとログの監視

10.1. はじめに

本書の他のセクションで説明されているセキュリティー対策のほかにも、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。

10.2. クラスターのイベント

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、イベントの一覧 を確認することをお勧めします。マスターコントローラーおよびプラグインの設定によっては、ここに一覧表示されているものよりも、多くのイベントタイプが使用される可能性があります。

イベントは、関連するリソースの namespace または デフォルト の 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
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")'

{
  "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'

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** ストレージがいっぱい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。

10.3. クラスターのログ

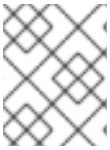
このセクションでは、クラスターで生成される運用ログの種類について説明します。

10.3.1. サービスログ

OpenShift Container Platform は、クラスターの静的 Pod で実行されるサービスのログを生成します。

- API (**master-logs api api** を使用)
- コントローラー (**master-logs controllers controllers** を使用)
- etcd (**master-logs etcd etcd** を使用)
- atomic-openshift-node (**journalctl -u atomic-openshift-node.service** を使用)

これらのログは、セキュリティー監査よりもでデバッグを目的として用意されています。ログは、**master-logs api api**、**master-logs controllers controllers**、または **master-logs etcd etcd** コマンドで、各サービスについてのログを取得できます。クラスターが集計ロギングスタックを実行する場合 (Ops クラスター など)、クラスター管理者はロギング **.operations** インデックスからログを取得できます。



注記

API サーバー、コントローラー、etcd 静的 pod は、kube-system の namespace で実行されます。

10.3.2. マスター API の監査ログ

ユーザー、管理者またはシステムコンポーネントによるマスター API 要求をログに記録するには、マスター API の 監査ロギングを有効にします。これにより、各マスターホストにファイルが作成されるか、またはファイルが設定されない場合は、ファイルはサービスジャーナルに含まれます。ジャーナルのエントリーは "**AUDIT**" を検索して見つけることができます。

監査ログのエントリー は、受信時にそれぞれの REST 要求を記録する 1 行と、完了時の HTTP レスポンスコードの 1 行で設定されます。以下の例では、システム管理者によるノード一覧を要求時の記録を示しています。

```
2017-10-17T13:12:17.635085787Z AUDIT: id="410eda6b-88d4-4491-87ff-394804ca69a1"
ip="192.168.122.156" method="GET" user="system:admin" groups="\"system:cluster-
admins\",\"system:authenticated\"" as="<self>" asgroups="<lookup>" namespace="<none>"
```

```
uri="/api/v1/nodes"
2017-10-17T13:12:17.636081056Z AUDIT: id="410eda6b-88d4-4491-87ff-394804ca69a1"
response="200"
```

以下の例で示すように、レスポンスコードごとに最新要求の数についての定期的なログのポーリングが役に立つ場合があります。

```
$ tail -5000 /var/log/origin/audit-ocp.log \
  | grep -Po 'response="..."' \
  | sort | uniq -c | sort -rn

  3288 response="200"
     8 response="404"
     6 response="201"
```

以下の一覧は、レスポンスコードのいくつかについて詳しく説明しています。

- **200** または **201** のレスポンスコードは、要求が正常であることを示します。
- **400** のレスポンスコードは、大半のクライアントでは発生するはずのない不正な要求を示すので、留意する必要のあるコードです。
- **404** のレスポンスコードは通常、存在しないリソースに対する無害な要求です。
- **500** - **599** のレスポンスコードは、サーバーエラーを示します。 これは、バグやシステム障害、または悪意のあるアクティビティーの結果として生じる場合があります。

エラーのレスポンス数が異常に多い場合には、対応する要求の監査ログエントリーを取得してさらに調査することができます。

注記

要求の IP アドレスは通常、クラスターホストまたは API ロードバランサーです。ロードバランサープロキシー要求の背後には IP アドレスの記録がありません (ただし、ロードバランサーのログは、要求元を判別するのに役立つことがあります)。

異常な数の要求については、特定のユーザーまたはグループごとに調べるのが適している場合があります。

以下の例では、監査ログの最後の 5000 行での要求数において上位 10 位のユーザーを表示しています。

```
$ tail -5000 /var/log/origin/audit-ocp.log \
  | grep -Po ' user="(.*?)(?<!\\)"' \
  | sort | uniq -c | sort -rn | head -10

   976  user="system:openshift-master"
   270  user="system:node:origin-node-1.example.local"
   270  user="system:node:origin-master.example.local"
    66  user="system:anonymous"
    32  user="system:serviceaccount:kube-system:cronjob-controller"
    24  user="system:serviceaccount:kube-system:pod-garbage-collector"
    18  user="system:serviceaccount:kube-system:endpoint-controller"
```

```
14  user="system:serviceaccount:openshift-infra:serviceaccount-pull-secrets-controller"
11  user="test user"
 4  user="test \" user"
```

通常は、さらに詳細なクエリーを実行するには追加のログ解析ツールを使用する必要があります。監査担当者は OpenShift v1 API および Kubernetes v1 API に精通し、関連するリソースの種類 (**uri** フィールド) に応じて要求の要約を監査ログから集計できるようにしておく必要があります。詳細は、REST API Reference を参照してください。

高度な監査ロギング機能 を利用することができます。この機能により、ログに記録する要求やログの詳細レベルを制御するための監査ポリシーファイルの提供が可能になります。高度な監査ログのエントリーは JSON 形式で詳細情報を提供し、ファイルまたはシステムジャーナルではなく Webhook を使用してログに記録することができます。