OpenShift Container Platform 4.5

アーキテクチャー

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

Last Updated: 2022-11-29

OpenShift Container Platform 4.5 アーキテクチャー

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

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

法律上の通知

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

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

本書では、OpenShift Container Platform のプラットフォームおよびアプリケーションのアーキテクチャーの概要を解説します。

目次

第1章 OPENSHIFT CONTAINER PLATFORM アーキテクチャー 4
1.1. OPENSHIFT CONTAINER PLATFORM の紹介 4
1.1.1. Kubernetes について 4
1.1.2. コンテナー化されたアプリケーションの利点 4
1.1.2.1. オペレーティングシステムの利点 4
1.1.2.2. デプロイメントとスケーリングの利点 5
1.1.3. OpenShift Container Platform の概要 5
1.1.3.1. カスタムオペレーティングシステム 6
1.1.3.2. 単純化されたインストールおよび更新プロセス 6
1.1.3.3. その他の主な機能 6
1.1.3.4. OpenShift Container Platform のライフサイクル 7
1.1.4. OpenShift Container Platform のインターネットアクセスおよび Telemetry アクセス 7

第2章 インストールおよび更新 9
2.1. OPENSHIFT CONTAINER PLATFORM インストールの概要 9
2.1.1. 利用可能なプラットフォーム 10
2.1.2. インストールプロセス 11
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でのインストールプロセス 12
ユーザーによってプロビジョニングされるインフラストラクチャーを使用したインストールプロセス 12
インストールプロセスの詳細 13
インストールのスコープ 14
2.2. OPENSHIFT CONTAINER PLATFORM の更新サービスについて 14
2.3. 管理外の OPERATOR のサポートポリシー 15

第3章 OPENSHIFT CONTAINER PLATFORM コントロールプレーン 17
3.1. OPENSHIFT CONTAINER PLATFORM コントロールプレーンについて 17
3.1.1. マシン設定プールを使用したノード設定管理 17
3.1.2. OpenShift Container Platform のマシンのロール 18
3.1.2.1. クラスターのワーカー 18
3.1.2.2. クラスターのマスター 18
3.1.3. OpenShift Container Platform の Operator 20
3.1.3.1. OpenShift Container Platform の Platform Operator 20
3.1.3.2. OLM によって管理される Operator 20
3.1.3.3. OpenShift Container Platform の更新サービスについて 21
3.1.3.4. Machine Config Operator について 21

第4章 OPENSHIFT CONTAINER PLATFORM の開発について 23
4.1. コンテナー化されたアプリケーションの開発について 23
4.2. 単純なコンテナーのビルド 23
4.2.1. コンテナービルドツールのオプション 25
4.2.2. ベースイメージのオプション 25
4.2.3. レジストリーオプション 26
4.3. OPENSHIFT CONTAINER PLATFORM 用の KUBERNETES マニフェストの作成 27
4.3.1. Kubernetes Pod およびサービスについて 27
4.3.2. アプリケーションのタイプ 28
4.3.3. 利用可能なサポートコンポーネント 28
4.3.4. マニフェストの適用 29
4.3.5. 次のステップ 29
4.4. OPERATOR 向けの開発 29

第5章 RED HAT ENTERPRISE LINUX COREOS (RHCOS) 31
5.1. RHCOS について 31

5.1.1. RHCOS の主な機能 31
5.1.2. RHCOS の設定方法の選択 32
5.1.3. RHCOS の設定方法の選択 33
5.1.4. Ignition について 34
5.1.4.1. Ignition の仕組み 34
5.1.4.2. Ignition の順序 35
5.2. IGNITION 設定ファイルの表示 36
5.3. インストール後の IGNITION 設定の変更 37

第6章 CI/CD の方法論および実践 39
6.1. クラスター管理およびアプリケーション設定管理向けの CI/CD 39
6.2. GITOPS の方法論および実戦 39
6.2.1. 単一クラスターおよびマルチクラスター管理向けの GitOps 39
6.2.2. アプリケーション設定管理向けの GitOps 40
6.2.3. GitOps テクノロジープロバイダーおよびインテグレーター 40

第7章 OPENSHIFT CONTAINER PLATFORM での ARGOCD の使用 41
7.1. ARGOCD の機能 41
7.2. サポートについての説明 41
7.3. ARGOCD ドキュメント 41

第8章 受付プラグイン 42
8.1. 受付プラグインについて 42
8.2. デフォルトの受付プラグイン 42
8.3. WEBHOOK 受付プラグイン 42
8.4. WEBHOOK 受付プラグインのタイプ 44
8.4.1. 受付プラグインの変更 44
8.4.2. 受付プラグインの検証 45
8.5. 動的受付の設定 46
8.6. 関連情報 54

第1章 OPENSHIFT CONTAINER PLATFORM アーキテクチャー

1.1. OPENSHIFT CONTAINER PLATFORM の紹介

OpenShift Container Platform は、コンテナー化されたアプリケーションを開発し、実行するためのプラットフォームです。アプリケーションおよびアプリケーションをサポートするデータセンターで、わずか数台のマシンとアプリケーションから、何百万ものクライアントに対応する何千ものマシンに拡張できるように設計されています。

Kubernetes をその基盤とする OpenShift Container Platform には、大規模な通信、ビデオストリーミング、ゲーミング、バンキングその他のアプリケーションのエンジンと同様に機能する技術が組み込まれています。Red Hat のオープンテクノロジーに実装することで、コンテナー化されたアプリケーションを、単一クラウドを超えてオンプレミスおよびマルチクラウド環境へと拡張することが可能です。

1.1.1. Kubernetes について

コンテナーイメージとそれらのイメージから実行されるコンテナーは、最先端のアプリケーション開発における主要な設定要素ですが、それらを大規模に実行するには、信頼性と柔軟性に優れた分配システムが必要となります。Kubernetes は、コンテナーをオーケストレーションするための事実上の業界標準です。

Kubernetes は、コンテナー化されたアプリケーションのデプロイ、スケーリング、管理を自動化するための、オープンソースのコンテナーオーケストレーションエンジンです。Kubernetes の一般的概念は非常にシンプルです。

- 1 つまたは複数のワーカーノードを使用することからスタートし、コンテナーのワークロードを実行します。
- 1 つまたは複数のマスターノードからワークロードのデプロイを管理します。
- Pod と呼ばれるデプロイメント単位にコンテナーをラップします。Pod を使うことでコンテナーに追加のメタデータが付与され、複数のコンテナーを単一のデプロイメントエンティティーにグループ化する機能が提供されます。
- 特殊な種類のアセットを作成します。たとえば、サービスは一連の Pod とそのアクセス方法を定義するポリシーによって表されます。このポリシーにより、コンテナーはサービス用の特定の IP アドレスを持っていない場合でも、必要とするサービスに接続することができます。レプリケーションコントローラーは、一度に実行するのに必要な Pod レプリカ数を示すもう一つの特殊なアセットです。この機能を使うと、現在の需要に対応できるようにアプリケーションを自動的にスケーリングすることができます。

Kubernetes は、わずか数年でクラウドとオンプレミスに非常に幅広く採用されるようになりました。このオープンソースの開発モデルにより、多くの人々がネットワーク、ストレージ、認証といったコンポーネント向けの各種の技術を実装し、Kubernetes を拡張することができます。

1.1.2. コンテナー化されたアプリケーションの利点

コンテナー化されたアプリケーションには、従来のデプロイメント方法を使用する場合と比べて多くの利点があります。アプリケーションはこれまで、すべての依存関係を含むオペレーティングシステムにインストールすることが必要でしたが、コンテナーの場合はアプリケーションがそれぞれの依存関係を持ち込むことができます。コンテナー化されたアプリケーションを作成すると多くの利点が得られます。

1.1.2.1. オペレーティングシステムの利点

コンテナーは、小型の、専用の Linux オペレーティングシステムをカーネルなしで使用します。ファイルシステム、ネットワーク、cgroups、プロセステーブル、namespace は、ホストの Linux システムから分離されていますが、コンテナーは、必要に応じてホストとシームレスに統合できます。Linux を基盤とすることで、コンテナーでは、迅速なイノベーションを可能にするオープンソース開発モデルに備わっているあらゆる利点を活用することができます。

各コンテナーは専用のオペレーティングシステムを使用するため、競合するソフトウェアの依存関係を必要とする複数のアプリケーションを、同じホストにデプロイできます。各コンテナーは、それぞれの依存するソフトウェアを持ち運び、ネットワークやファイルシステムなどの独自のインターフェイスを管理します。 したがってアプリケーションはそれらのアセットについて競い合う必要はありません。

1.1.2.2. デプロイメントとスケーリングの利点

アプリケーションのメジャーリリース間でローリングアップグレードを行うと、ダウンタイムなしにアプリケーションを継続的に改善し、かつ現行リリースとの互換性を維持することができます。

さらに、アプリケーションの新バージョンを、旧バージョンと並行してデプロイおよびテストすることもできます。コンテナーがテストにパスしたら、新規コンテナーを追加でデプロイし、古いコンテナーを削除できます。

アプリケーションのソフトウェアの依存関係すべてはコンテナー内で解決されるので、データセンターの各ホストには標準化されたオペレーティングシステムを使用できます。各アプリケーションホスト向けに特定のオペレーティングシステムを設定する必要はありません。データセンターでさらに多くの容量が必要な場合は、別の汎用ホストシステムをデプロイできます。

同様に、コンテナー化されたアプリケーションのスケーリングも簡単です。OpenShift Container Platform には、どのようなコンテナー化したサービスでもスケーリングできる、シンプルで標準的な方法が用意されています。アプリケーションを大きなモノリシックな (一枚岩的な) サービスではなく、マイクロサービスのセットとしてビルドする場合、個々のマイクロサービスを、需要に合わせて個別にスケーリングできます。この機能により、アプリケーション全体ではなく必要なサービスのみをスケーリングすることができ、使用するリソースを最小限に抑えつつ、アプリケーションの需要を満たすことができます。

1.1.3. OpenShift Container Platform の概要

OpenShift Container Platform は、以下を含むエンタープライズ対応の拡張機能を Kubernetes に提供します。

- ハイブリッドクラウドのデプロイメント。OpenShift Container Platform クラスターをさまざまなパブリッククラウドのプラットフォームまたはお使いのデータセンターにデプロイできます。
- Red Hat の統合されたテクノロジー。OpenShift Container Platform の主なコンポーネントは、Red Hat Enterprise Linux (RHEL) と関連する Red Hat の技術に由来します。OpenShift Container Platform は、Red Hat の高品質エンタープライズソフトウェアの集中的なテストや認定の取り組みによる数多くの利点を活用しています。
- オープンソースの開発モデル。開発はオープンソースで行われ、ソースコードはソフトウェアのパブリックリポジトリーから入手可能です。このオープンな共同作業が迅速な技術と開発を促進します。

Kubernetes はアプリケーションの管理で威力を発揮しますが、プラットフォームレベルの各種要件やデプロイメントプロセスを指定したり、管理したりすることはありません。そのため、OpenShift Container Platform 4.5 が提供する強力かつ柔軟なプラットフォーム管理ツールとプロセスは重要な利点の 1 つとなります。以下のセクションでは、OpenShift Container Platform のいくつかのユニークな機能と利点について説明します。

1.1.3.1. カスタムオペレーティングシステム

OpenShift Container Platform はコンテナー指向のオペレーティングシステムであり、CoreOS と Red Hat Atomic Host オペレーティングシステムの最良の機能の一部を組み合わせた Red Hat Enterprise Linux CoreOS (RHCOS) を採用しています。RHCOS は、OpenShift Container Platform のコンテナー化されたアプリケーションを実行する目的で設計されており、新規ツールと連携して迅速なインストール、Operator ベースの管理、および単純化されたアップグレードを実現します。

RHCOS には以下が含まれます。

- Ignition。OpenShift Container Platform が使用するマシンを最初に起動し、設定するための初回起動時のシステム設定です。
- CRI-O、Kubernetes ネイティブコンテナーランタイム実装。これはオペレーティングシステムに密接に統合し、Kubernetes の効率的で最適化されたエクスペリエンスを提供します。CRI-O は、コンテナーを実行、停止および再起動を実行するための機能を提供します。これは、OpenShift Container Platform 3 で使用されていた Docker Container Engine を完全に置き換えます。
- Kubelet、Kubernetes のプライマリーノードエージェント。 これは、コンテナーを起動し、これを監視します。

OpenShift Container Platform 4.5 ではすべてのコントロールプレーンマシンで RHCOS を使用する必要がありますが、Red Hat Enterprise Linux (RHEL) をコンピュートまたはワーカーマシンのオペレーティングシステムとして使用することができます。RHEL のワーカーを使用する選択をする場合、すべてのクラスターマシンに 対して RHCOS を使用する場合よりも多くのシステムメンテナーンスを実行する必要があります。

1.1.3.2. 単純化されたインストールおよび更新プロセス

OpenShift Container Platform 4.5 では、適切なパーミッションを持つアカウントを使用している場合、単一のコマンドを実行し、いくつかの値を指定することで、サポートされているクラウドに実稼働用のクラスターをデプロイすることができます。また、サポートされているプラットフォームを使用している場合、クラウドのインストールをカスタマイズしたり、クラスターをお使いのデータセンターにインストールすることも可能です。

クラスターのすべてのマシンが RHCOS を使用している場合、OpenShift Container Platform の更新またはアップグレードは、高度に自動化された単純なプロセスで実行できます。OpenShift Container Platform は、各マシンで実行される、オペレーティングシステム自体を含むシステムとサービスを中央のコントロールプレーンから完全に制御するので、アップグレードは自動イベントになるように設計されています。クラスターに RHEL のワーカーマシンが含まれる場合、コントロールプレーンの使用には単純化された更新プロセスの利点があるものの、RHEL マシンのアップグレードには、より多くのタスクの実行が必要になります。

1.1.3.3. その他の主な機能

Operator は、OpenShift Container Platform 4.5 コードベースの基本単位であるだけでなく、アプリケーションとアプリケーションで使用されるソフトウェアコンポーネントをデプロイするための便利な手段です。Operator をプラットフォームの基盤として使用することで、OpenShift Container Platform ではオペレーティングシステムおよびコントロールプレーンアプリケーションの手動によるアップグレードが不要になります。Cluster Version Operator や Machine Config Operator などの OpenShift Container Platform の Operator が、それらの重要なコンポーネントのクラスター全体での管理を単純化します。

Operator Lifecycle Manager (OLM) および OperatorHub は、Operator を保管し、アプリケーションの開発やデプロイを行う人々に Operator を提供する機能を提供します。

Red Hat Quay Container Registry は、ほとんどのコンテナーイメージと Operator を OpenShift Container Platform クラスターに提供する Quay.io コンテナーレジストリーです。Quay.io は、何百万ものイメージやタグを保存する Red Hat Quay の公開レジストリー版です。

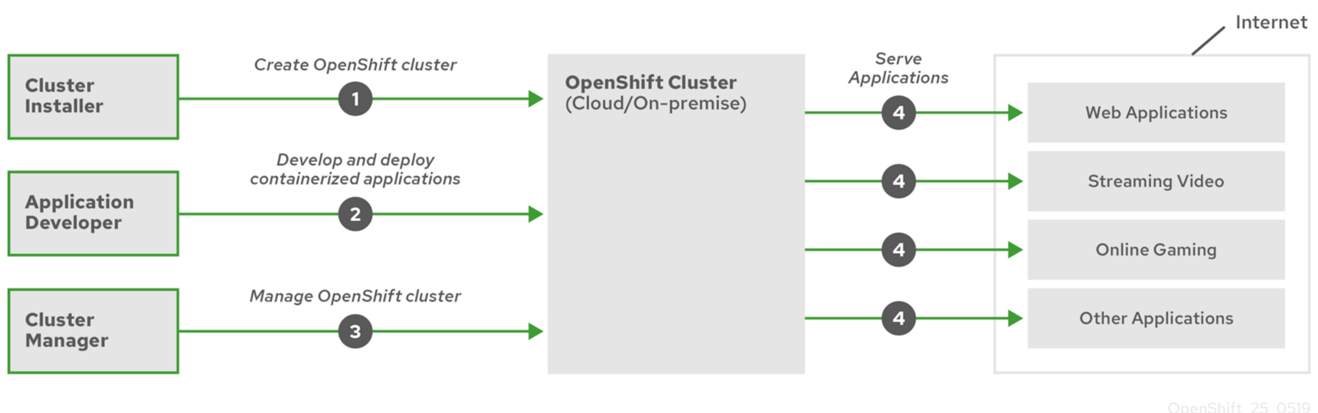
OpenShift Container Platform での Kubernetes のその他の拡張には、SDN (Software Defined Networking)、認証、ログ集計、監視、およびルーティングの強化された機能が含まれます。OpenShift Container Platform は、包括的な Web コンソールとカスタム OpenShift CLI (**oc**) インタフェースも提供します。

1.1.3.4. OpenShift Container Platform のライフサイクル

以下の図は、OpenShift Container Platform の基本的なライフサイクルを示しています。

- OpenShift Container Platform クラスターの作成
- クラスターの管理
- アプリケーションの開発とデプロイ
- アプリケーションのスケールアップ

図1.1 OpenShift Container Platform の概要



1.1.4. OpenShift Container Platform のインターネットアクセスおよび Telemetry アクセス

OpenShift Container Platform 4.5 では、クラスターをインストールするためにインターネットアクセスが必要になります。クラスターの健全性および正常に実行された更新についてのメトリクスを提供するためにデフォルトで実行される Telemetry サービスにもインターネットアクセスが必要です。クラスターがインターネットに接続されている場合、Telemetry は自動的に実行され、クラスターは Red Hat OpenShift Cluster Manager (OCM) に登録されます。

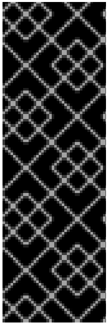
Red Hat OpenShift Cluster Manager インベントリーが Telemetry によって自動的に維持されるか、または OCM を手動で使用しているかのいずれによって正常であることを確認した後に、subscription watch を使用して、アカウントまたはマルチクラスターレベルで OpenShift Container Platform サブスクリプションを追跡します。

インターネットへのアクセスは以下を実行するために必要です。

- Red Hat OpenShift Cluster Manager ページにアクセスし、インストールプログラムをダウンロードし、サブスクリプション管理を実行します。クラスターにインターネットアクセスがあり、Telemetry を無効にしない場合、そのサービスは有効なサブスクリプションでクラスター

を自動的に使用します。

- クラスターのインストールに必要なパッケージを取得するために Quay.io にアクセスします。
- クラスターの更新を実行するために必要なパッケージを取得します。



重要

クラスターでインターネットに直接アクセスできない場合、プロビジョニングする一部のタイプのインフラストラクチャーでネットワークが制限されたインストールを実行できます。このプロセスで、必要なコンテンツをダウンロードし、これを使用してミラーレジストリーにクラスターのインストールおよびインストールプログラムの生成に必要なパッケージを設定します。インストールタイプによっては、クラスターのインストール環境でインターネットアクセスが不要となる場合があります。クラスターを更新する前に、ミラーレジストリーのコンテンツを更新します。

第2章 インストールおよび更新

2.1. OPENSHIFT CONTAINER PLATFORM インストールの概要

OpenShift Container Platform インストールプログラムは柔軟性を提供します。インストールプログラムを使用して、インストールプログラムがプロビジョニングし、クラスターで維持するインフラストラクチャーでクラスターをデプロイしたり、ユーザーが独自に準備し、維持するインフラストラクチャーでクラスターをデプロイしたりすることができます。

OpenShift Container Platform クラスターの基本的な 2 つのタイプとして、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クラスターと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クラスターがあります。

これらのクラスターのタイプにはどちらにも以下の特徴があります。

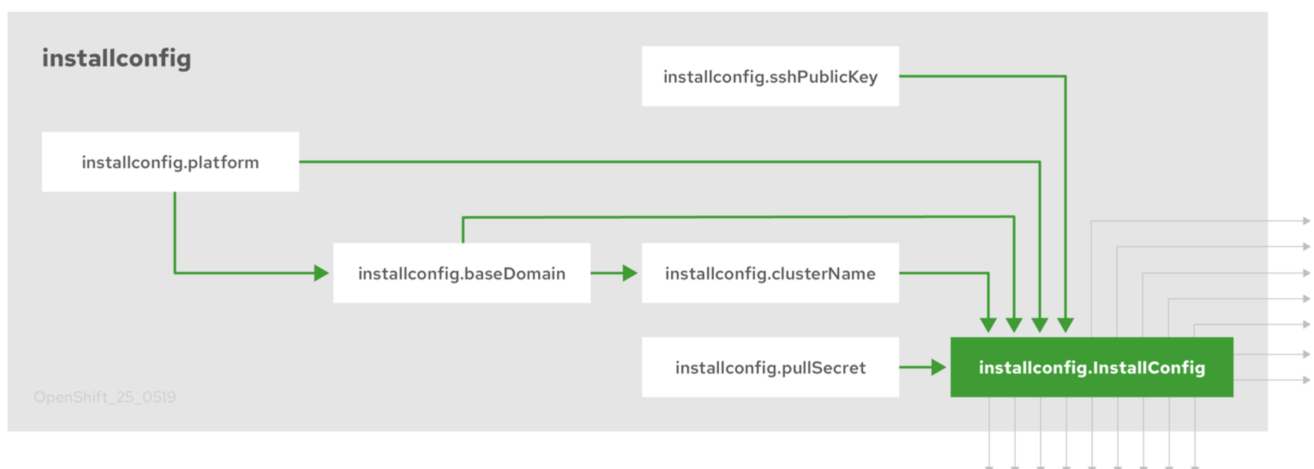
- 単一障害点のない可用性の高いインフラストラクチャーがデフォルトで利用可能である。
- 管理者は適用される更新内容および更新タイミングを制御できる。

同一のインストールプログラムを使用してこれらクラスターの両方のタイプをデプロイできます。インストールプログラムで生成される主なアセットは ブートストラップ、マスターおよびワーカーマシンの Ignition 設定です。これらの 3 つの設定および適切に設定されたインフラストラクチャーを使用して、OpenShift Container Platform クラスターを起動することができます。

OpenShift Container Platform インストールプログラムは、クラスターのインストールを管理するために一連のターゲットおよび依存関係を使用します。インストールプログラムには、達成する必要のある一連のターゲットが設定され、それぞれのターゲットには一連の依存関係が含まれます。各ターゲットはそれぞれの依存関係の条件が満たされ次第、別個に解決されるため、インストールプログラムは複数のターゲットを並行して達成できるように動作します。最終的なターゲットはクラスターを実行することです。コマンドを実行するのではなく依存関係の条件を満たすことにより、インストールプログラムは、コマンドを実行してコンポーネントを再度作成する代わりに、既存のコンポーネントを認識し、それらを使用することができます。

以下の図は、インストールのターゲットと依存関係のサブセットを示しています。

図2.1 OpenShift Container Platform インストールのターゲットおよび依存関係



インストール後に、各クラスターマシンは Red Hat Enterprise Linux CoreOS (RHCOS) をオペレーティングマシンとして使用します。RHCOS は Red Hat Enterprise Linux (RHEL) の不変のコンテナーホストのバージョンであり、デフォルトで SELinux が有効にされた RHEL カーネルを特長としています。これ

には、Kubernetes ノードエージェントである **kubelet** や、Kubernetes に対して最適化される CRI-O コンテナーランタイムが含まれます。

OpenShift Container Platform 4.5 クラスターのすべてのコントロールプレーンは、Ignition と呼ばれる最初の起動時に使用される重要なプロビジョニングツールが含まれる RHCOS を使用する必要があります。このツールは、クラスターのマシンの設定を可能にします。オペレーティングシステムの更新は、Operator によってクラスター全体に展開されるコンテナーイメージに組み込まれる Atomic OSTree リポジトリーとして提供されます。実際のオペレーティングシステムの変更については、rpm-ostree を使用する atomic 操作として各マシンでインプレースで実行されます。これらのテクノロジーを組み合わせて使用することにより、OpenShift Container Platform では、プラットフォーム全体を最新の状態に保つインプレースアップグレードで、その他のアプリケーションをクラスターで管理するかのようにオペレーティングシステムを管理することができます。これらのインプレースアップグレードにより、オペレーションチームの負担を軽減することができます。

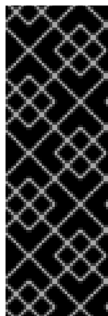
RHCOS をすべてのクラスターマシンのオペレーティングシステムとして使用する場合、クラスターはオペレーティングシステムを含む、そのコンポーネントとマシンのすべての側面を管理します。このため、インストールプログラムと Machine Config Operator のみがマシンを変更することができます。インストールプログラムは Ignition 設定ファイルを使用して各マシンの状態を設定し、Machine Config Operator はインストール後に、新規証明書またはキーの適用などのマシンへの変更を実行します。

2.1.1. 利用可能なプラットフォーム

OpenShift Container Platform バージョン 4.5 では、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を使用するクラスターの場合、以下のプラットフォームにインストールできます。

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure
- Red Hat OpenStack Platform (RHOSP) バージョン 13 および 16
  - OpenShift Container Platform の最新リリースは、最新の RHOSP のロンググライフリリースおよび中間リリースの両方をサポートします。RHOSP リリースの互換性についての詳細は、OpenShift Container Platform on RHOSP support matrix を参照してください。
- Red Hat Virtualization (RHV)
- VMware vSphere

これらのクラスターの場合、インストールプロセスを実行するコンピューターを含むすべてのマシンが、プラットフォームコンテナーのイメージをプルし、Telemetry データを Red Hat に提供できるようインターネットに直接アクセスできる必要があります。



重要

インストール後は、以下の変更はサポートされません。

- クラウドプロバイダープラットフォームの組み合わせ
- クラスターがインストールされているプラットフォームとは異なるプラットフォームの永続ストレージフレームワークを使用するなどの、クラウドプロバイダーのコンポーネントの組み合わせ

OpenShift Container Platform バージョン 4.5 では、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使用するクラスターの場合、以下のプラットフォームにインストールできます。

- AWS
- Azure
- GCP
- RHOSP
- VMware vSphere
- ベアメタル
- IBM Z または LinuxONE
- IBM Power Systems

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でのインストールでは、各マシンにインターネットのフルアクセスを持たせることができ、クラスターをプロキシーの背後に配置するか、または **ネットワークの制限されたインストール** を実行できます。ネットワークが制限された環境でのインストールでは、クラスターのインストールに必要なイメージをダウンロードして、ミラーレジストリーに配置し、そのデータを使用してクラスターをインストールできます。vSphere またはベアメタルインフラストラクチャーのネットワークが制限されたインストールでは、プラットフォームコンテナーのイメージをプルするためにインターネットにアクセスする必要がありますが、クラスターマシンはインターネットへの直接のアクセスを必要としません。

OpenShift Container Platform 4.x Tested Integrations のページには、各種プラットフォームの統合テストについての詳細が記載されています。

2.1.2. インストールプロセス

OpenShift Container Platform クラスターをインストールする場合は、Red Hat OpenShift Cluster Manager サイトの該当する インフラストラクチャープロバイダー ページからインストールプログラムをダウンロードします。このサイトでは以下を管理しています。

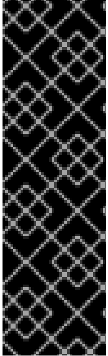
- アカウントの REST API
- 必要なコンポーネントを取得するために使用するプルシークレットであるレジストリートークン
- クラスターのアイデンティティーを Red Hat アカウントに関連付けて使用状況のメトリクスの収集を容易にするクラスター登録

OpenShift Container Platform 4.5 では、インストールプログラムは、一連のアセットに対して一連のファイル変換を実行する Go バイナリーファイルです。インストールプログラムと対話する方法は、インストールタイプによって異なります。

- 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のクラスターの場合、インフラストラクチャーのブートストラップおよびプロビジョニングは、ユーザーが独自に行うのではなくインストールプログラムが代行します。インストールプログラムは、クラスターをサポートするために必要なネットワーク、マシン、およびオペレーティングシステムのすべてを作成します。
- クラスターのインフラストラクチャーを独自にプロビジョニングし、管理する場合には、ブートストラップマシン、ネットワーク、負荷分散、ストレージ、および個々のクラスターマシン

を含む、すべてのクラスターインフラストラクチャーおよびリソースを指定する必要があります。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クラスターが提供する高度なマシン管理およびスケーリング機能を使用することはできません。

インストール時には、お使いのマシンタイプ用の **install-config.yaml** という名前のインストール設定ファイル、Kubernetes マニフェスト、および Ingition 設定ファイルの 3 つのファイルセットを使用します。

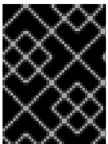


重要

インストール時に、Kubernetes および基礎となる RHCOS オペレーティングシステムを制御する Ignition 設定ファイルを変更することができます。ただし、これらのオブジェクトに対して加える変更の適合性を確認するための検証の方法はなく、これらのオブジェクトを変更するとクラスターが機能しなくなる可能性があります。これらのオブジェクトを変更する場合、クラスターが機能しなくなる可能性があります。このリスクがあるために、変更方法についての文書化された手順に従っているか、または Red Hat サポートが変更することを指示した場合を除き、Kubernetes および Ignition 設定ファイルの変更はサポートされていません。

インストール設定ファイルは Kubernetes マニフェストに変換され、その後マニフェストは Ignition 設定にラップされます。インストールプログラムはこれらの Ignition 設定ファイルを使用してクラスターを作成します。

インストール設定ファイルはインストールプログラムの実行時にすべてプルーニングされるため、再び使用する必要のあるすべての設定ファイルをバックアップしてください。



重要

インストール時に設定したパラメーターを変更することはできませんが、インストール後に数多くのクラスター属性を変更することができます。

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でのインストールプロセス

デフォルトのインストールタイプは、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です。デフォルトで、インストールプログラムはインストールウィザードとして機能し、独自に判別できない値の入力を求めるプロンプトを出し、残りのパラメーターに妥当なデフォルト値を提供します。インストールプロセスは、高度なインフラストラクチャーシナリオに対応するようカスタマイズすることもできます。インストールプログラムは、クラスターの基盤となるインフラストラクチャーをプロビジョニングします。

標準クラスターまたはカスタマイズされたクラスターのいずれかをインストールすることができます。標準クラスターの場合、クラスターをインストールするために必要な最小限の詳細情報を指定します。カスタマイズされたクラスターの場合、コントロールプレーンが使用するマシン数、クラスターがデプロイする仮想マシンのタイプ、または Kubernetes サービスネットワークの CIDR 範囲などのプラットフォームについての詳細を指定することができます。

可能な場合は、この機能を使用してクラスターインフラストラクチャーのプロビジョニングと保守の手間を省くようにしてください。他のすべての環境の場合には、インストールプログラムを使用してクラスターインフラストラクチャーをプロビジョニングするために必要なアセットを生成できます。

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クラスターの場合、OpenShift Container Platform は、オペレーティングシステム自体を含むクラスターのすべての側面を管理します。各マシンは、それが参加するクラスターでホストされるリソースを参照する設定に基づいて起動します。この設定により、クラスターは更新の適用時に自己管理できます。

ユーザーによってプロビジョニングされるインフラストラクチャーを使用したインストールプロセス

OpenShift Container Platform はユーザーが独自にプロビジョニングするインフラストラクチャーにインストールすることもできます。インストールプログラムを使用してクラスターインフラストラクチャーのプロビジョニングに必要なアセットを生成し、クラスターインフラストラクチャーを作成し、その後にクラスターをプロビジョニングしたインフラストラクチャーにデプロイします。

インストールプログラムがプロビジョニングするインフラストラクチャーを使用しない場合、以下を含むクラスターリソースを管理し、維持する必要があります。

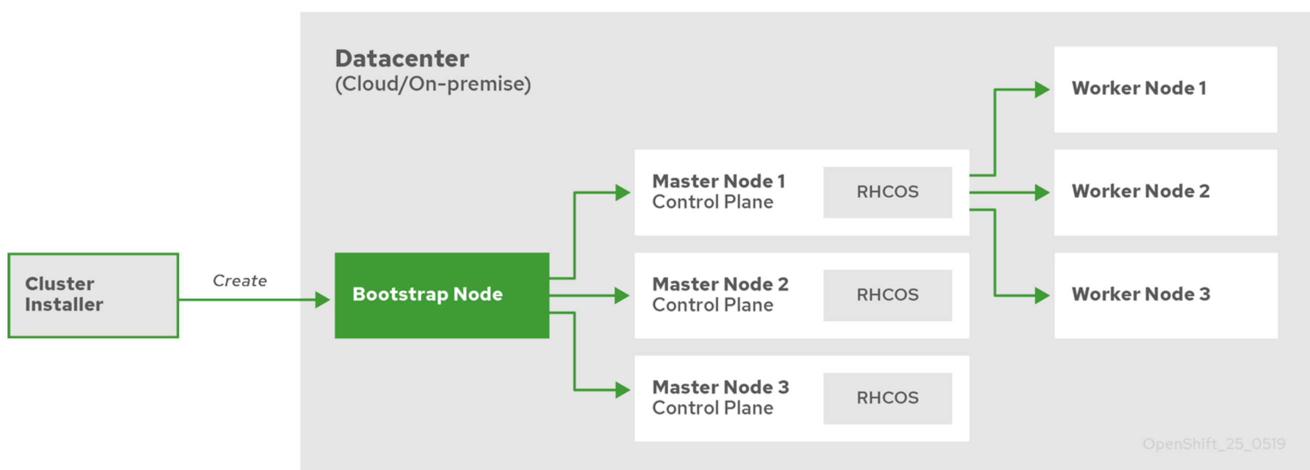
- クラスターを設定するコントロールプレーンおよびコンピュートマシンの基礎となるインフラストラクチャー
- ロードバランサー
- DNS レコードおよび必要なサブネットを含むクラスターネットワーク
- クラスターインフラストラクチャーおよびアプリケーションのストレージ

クラスターで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には、RHEL ワーカーマシンをクラスターに追加するオプションを使用できます。

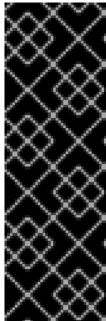
インストールプロセスの詳細

クラスターの各マシンにはプロビジョニング時にクラスターについての情報が必要になるため、OpenShift Container Platform は初期設定時に一時的な **bootstrap** マシンを使用し、必要な情報を永続的なコントロールマシンに提供します。これは、クラスターの作成方法を記述する Ignition 設定ファイルを使用して起動されます。ブートストラップマシンは、コントロールプレーンを設定するマスターマシンを作成します。その後、コントロールプレーンマシンはコンピュートマシン (ワーカーマシンとしても知られる) を作成します。以下の図はこのプロセスを示しています。

図2.2 ブートストラップ、マスター、およびワーカーマシンの作成



クラスターマシンを初期化した後、ブートストラップマシンは破棄されます。すべてのクラスターがこのブートストラッププロセスを使用してクラスターを初期化しますが、ユーザーがクラスターのインフラストラクチャーをプロビジョニングする場合には、多くの手順を手動で実行する必要があります。



重要

インストールプログラムが生成する Ignition 設定ファイルには、24 時間が経過すると期限切れになり、その後に更新される証明書が含まれます。証明書を更新する前にクラスターが停止し、24 時間経過した後にクラスターを再起動すると、クラスターは期限切れの証明書を自動的に復元します。例外として、kubelet 証明書を回復するために保留状態の **node-bootstrapper** 証明書署名要求 (CSR) を手動で承認する必要があります。詳細は、**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** についてのドキュメントを参照してください。

クラスターのブートストラップには、以下のステップが関係します。

1. ブートストラップマシンが起動し、コントロールプレーンマシンの起動に必要なリモートリソースのホスティングを開始します。(ユーザーがインフラストラクチャーをプロビジョニングする場合には手動の介入が必要になります)。
2. ブートストラップマシンは、単一ノードの etcd クラスターと一時的な Kubernetes コントロールプレーンを起動します。
3. コントロールプレーンマシンは、ブートストラップマシンからリモートリソースをフェッチし、起動を終了します。(ユーザーがインフラストラクチャーをプロビジョニングする場合には手動の介入が必要になります)。
4. 一時的なコントロールプレーンは、実稼働コントロールプレーンマシンに対して実稼働コントロールプレーンをスケジュールします。
5. Cluster Version Operator (CVO) はオンラインになり、etcd Operator をインストールします。etcd Operator はすべてのコントロールプレーンノードで etcd をスケールアップします。
6. 一時的なコントロールプレーンはシャットダウンし、コントロールを実稼働コントロールプレーンに渡します。
7. ブートストラップマシンは OpenShift Container Platform コンポーネントを実稼働コントロールプレーンに挿入します。
8. インストールプログラムはブートストラップマシンをシャットダウンします。(ユーザーがインフラストラクチャーをプロビジョニングする場合には手動の介入が必要になります)。
9. コントロールプレーンはワーカーノードをセットアップします。
10. コントロールプレーンは一連の Operator の形式で追加のサービスをインストールします。

このブートストラッププロセスの結果として、OpenShift Container Platform クラスターが完全に実行されます。次に、クラスターはサポートされる環境でのワーカーマシンの作成など、日常の操作に必要な残りのコンポーネントをダウンロードし、設定します。

インストールのスコープ

OpenShift Container Platform インストールプログラムのスコープは意図的に狭められています。単純さを確保し、確実にインストールを実行できるように設計されているためです。インストールが完了した後に数多くの設定タスクを実行することができます。

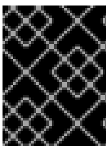
2.2. OPENSHIFT CONTAINER PLATFORM の更新サービスについて

OpenShift Container Platform の更新サービスとは、OpenShift Container Platform と Red Hat Enterprise Linux CoreOS (RHCOS) の両方に OTA(over-the-air) 更新を提供するホスト型サービスです。コンポーネント Operator のグラフ、または **頂点** とそれらを結ぶ **辺** を含む図表が提示されます。

グラフの辺は、どのバージョンであれば安全に更新できるかを示します。 頂点は、管理されたクラスターコンポーネントの想定された状態を特定する更新のペイロードです。

クラスター内の Cluster Version Operator (CVO) は、OpenShift Container Platform の更新サービスをチェックして、グラフの現在のコンポーネントバージョンとグラフの情報に基づき、有効な更新および更新パスを確認します。ユーザーが更新をリクエストすると、OpenShift Container Platform CVO はその更新のリリースイメージを使ってクラスターをアップグレードします。リリースアーティファクトは、コンテナーイメージとして Quay でホストされます。

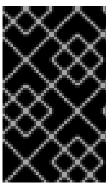
OpenShift Container Platform 更新サービスが互換性のある更新のみを提供できるようにするために、自動化を支援するリリース検証 Pipeline が使用されます。それぞれのリリースアーティファクトについて、他のコンポーネントパッケージだけでなくサポートされているクラウドプラットフォームおよびシステムアーキテクチャーとの互換性の有無が検証されます。Pipeline がリリースの適合性を確認した後に、OpenShift Container Platform 更新サービスは更新が利用可能であることを通知します。



重要

更新サービスが有効な更新をすべて表示するために、更新サービスが表示しないバージョンへの更新を強制することはできません。

連続更新モードでは、2 つのコントローラーが実行されます。1 つのコントローラーはペイロードマニフェストを絶えず更新し、それらをクラスターに適用し、Operator が利用可能か、アップグレード中か、または失敗しているかに応じて Operator の制御されたロールアウトのステータスを出力します。2 つ目のコントローラーは OpenShift Container Platform 更新サービスをポーリングして、更新が利用可能かどうかを判別します。



重要

クラスターを以前のバージョンに戻すこと、つまりロールバックはサポートされていません。サポートされているのは、新規バージョンへのアップグレードのみです。アップグレードできない場合は、Red Hat サポートにお問い合わせください。

アップグレードプロセスで、Machine Config Operator (MCO) は新規設定をクラスターマシンに適用します。これは、マシン設定プールの **maxUnavailable** フィールドによって指定されるノードの数を分離し、それらを利用不可としてマークします。デフォルトで、この値は **1** に設定されます。次に、新しい設定を適用して、マシンを再起動します。Red Hat Enterprise Linux (RHEL) マシンをワーカーとして使用する場合、まず OpenShift API をそれらのマシンで更新する必要があるため、MCO はそれらのマシンで kubelet を更新しません。新規バージョンの仕様は古い kubelet に適用されるため、RHEL マシンを **Ready** 状態に戻すことができません。マシンが利用可能になるまで更新を完了することはできません。ただし、利用不可のノードの最大数は、その数のマシンがサービス停止状態のマシンとして分離されても通常のクラスター操作が継続できるようにするために設定されます。

2.3. 管理外の OPERATOR のサポートポリシー

Operator の **管理状態** は、Operator が設計通りにクラスター内の関連するコンポーネントのリソースをアクティブに管理しているかどうかを定めます。Operator が **unmanaged** 状態に設定されている場合、これは設定の変更に応答せず、更新を受信しません。

これは非実稼働クラスターやデバッグ時に便利ですが、管理外の状態の Operator はサポートされず、クラスター管理者は個々のコンポーネント設定およびアップグレードを完全に制御していることを前提としています。

Operator は以下の方法を使用して管理外の状態に設定できます。

- **個別の Operator 設定**
  個別の Operator には、それらの設定に **managementState** パラメーターがあります。これは Operator に応じてさまざまな方法でアクセスできます。たとえば、Cluster Logging Operator は管理するカスタムリソース (CR) を変更することによってこれを実行しますが、Cluster Samples Operator はクラスター全体の設定リソースを使用します。

  managementState パラメーターを **Unmanaged** に変更する場合、Operator はそのリソースをアクティブに管理しておらず、コンポーネントに関連するアクションを取らないことを意味します。Operator によっては、クラスターが破損し、手動リカバリーが必要になる可能性があるため、この管理状態に対応しない可能性があります。

  警告

  個別の Operator を **Unmanaged** 状態に変更すると、特定のコンポーネントおよび機能がサポート対象外になります。サポートを継続するには、報告された問題を **Managed** 状態で再現する必要があります。

- **Cluster Version Operator (CVO) のオーバーライド**
  spec.overrides パラメーターを CVO の設定に追加すると、管理者はコンポーネントについての CVO の動作に対してオーバーライドの一覧を追加できます。コンポーネントについて **spec.overrides[].unmanaged** パラメーターを **true** に設定すると、クラスターのアップグレードがブロックされ、CVO のオーバーライドが設定された後に管理者にアラートが送信されます。

  > Disabling ownership via cluster version overrides prevents upgrades. Please remove overrides before continuing.

  警告

  CVO のオーバーライドを設定すると、クラスター全体がサポートされない状態になります。サポートを継続するには、オーバーライドを削除した後に、報告された問題を再現する必要があります。

第3章 OPENSHIFT CONTAINER PLATFORM コントロールプレーン

3.1. OPENSHIFT CONTAINER PLATFORM コントロールプレーンについて

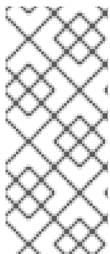
コンテナープレーンはマスターマシンから設定されており、OpenShift Container Platform クラスターを管理します。コントロールプレーンマシンは、コンピュートマシン (ワーカーマシンとしても知られる) のワークロードを管理します。クラスター自体は、Cluster Version Operator、Machine Config Operator、および個々の Operator のアクションによって、マシンへのすべてのアップグレードを管理します。

3.1.1. マシン設定プールを使用したノード設定管理

コントロールプレーンのコンポーネントまたはユーザーワークロードを実行するマシンは、それらが処理するリソースタイプに基づいてグループに分類されます。マシンのこれらのグループはマシン設定プール (MCP) と呼ばれます。それぞれの MCP はノードのセットおよびその対応するマシン設定を管理します。ノードのロールは、これが所属する MCP を判別します。MCP は割り当てられたノードロールラベルに基づいてノードを制御します。MCP のノードには同じ設定があります。つまり、ワークロードの増減に応じてノードのスケールアップおよび破棄が可能です。

デフォルトで、クラスターのインストール時にクラスターによって作成される 2 つの MCP (**master** および **worker**) があります。それぞれのデフォルト MCP には、Machine Config Operator (MCO) によって適用される定義された設定があり、これは MCP を管理し、MCP アップグレードを容易にするために使用されます。追加の MCP またはカスタムプールを作成して、デフォルトのノードタイプの範囲を超えるカスタムユースケースを持つノードを管理できます。

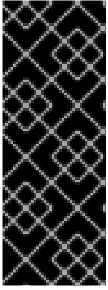
カスタムプールは、ワーカープールから設定を継承するプールです。これらはワーカープールのターゲット設定を使用しますが、カスタムプールのみをターゲットに設定する変更をデプロイする機能を追加します。カスタムプールはワーカープールから設定を継承するため、ワーカープールへの変更もカスタムプールに適用されます。ワーカープールから設定を継承しないカスタムプールは MCO ではサポートされません。



注記

ノードは 1 つの MCP にのみ含めることができます。ノードにいくつかの MCP に対応するラベルがある場合 (**worker,infra** など)、これはワーカープールではなく infra カスタムプールによって管理されます。カスタムプールは、ノードラベルに基づいて管理するノードの選択を優先します。カスタムプールに属さないノードはワーカープールによって管理されます。

クラスターで管理するすべてのノードロールについてカスタムプールを使用することが推奨されます。たとえば、infra ワークロードを処理するために infra ノードを作成する場合、それらのノードをまとめるためにカスタム infra MCP を作成することが推奨されます。**infra** ロールラベルをワーカーノードに適用し、これが **worker,infra** の二重ラベルを持つようにするものの、カスタム infra MCP がない場合、MCO はこれをワーカーノードと見なします。ノードから **worker** ラベルを削除して、これをカスタムプールで分類せずに **infra** ラベルを適用する場合、ノードは MCO によって認識されず、クラスターによって管理されません。



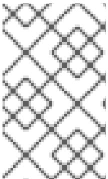
重要

infra ワークロードのみを実行する **infra** ロールのラベルが付いたノードは、サブスクリプションの合計数にカウントされません。infra ノードを管理する MCP は、クラスターでサブスクリプション料金を決定する方法と相互に排他的です。適切な **infra** ロールを持つノードにテイントを付け、テイントを使用してユーザーのワークロードがそのノードにスケジュールされないようにすることが、infra ワークロードのサブスクリプション料金を防ぐための唯一の要件になります。

MCO はプールの更新を個別に適用します。たとえば、すべてのプールに影響を与える更新がある場合、各プールのノードは相互に並行して更新されます。カスタムプールを追加する場合、そのプールのノードはマスターおよびワーカーノードとの同時更新を試みます。

3.1.2. OpenShift Container Platform のマシンのロール

OpenShift Container Platform はホストに複数の異なるロールを割り当てます。これらのロールは、クラスター内のマシンの機能を定義します。クラスターには、標準のマスターおよびワーカーのロールタイプの定義が含まれます。



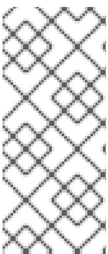
注記

また、クラスターにはブートストラップロールの定義も含まれます。ブートストラップマシンが使用されるのはクラスターのインストール時のみであり、この機能については、クラスターインストールのドキュメントで説明されています。

3.1.2.1. クラスターのワーカー

Kubernetes のクラスターでは、Kubernetes のユーザーがリクエストした実際のワークロードは、ワーカーノードで実行され、管理されます。ワーカーノードは、独自の容量と (マスターサービスの一部である) スケジューラーを公開し、どのノードでコンテナーと Pod を起動するかを決定します。重要なサービスは各ワーカーノードで実行されますが、これには、コンテナーエンジンである CRI-O、コンテナーのワークロードの実行と停止の要求を受け入れ、実行するサービスである Kubelet、ワーカー間での Pod の通信を管理するサービスプロキシーが含まれます。

OpenShift Container Platform では、マシンセットがワーカーマシンを制御します。ワーカーのロールを持つマシンは、自動スケーリングを行う特定のマシンプールによって制御されるコンピュートワークロードを実行します。OpenShift Container Platform は複数のマシンタイプをサポートすることができ、ワーカーマシンは **コンピュート**マシンとして分類されています。本リリースでは、コンピュートマシンの唯一のデフォルトタイプはワーカーマシンであるため、本リリースでは **ワーカーマシン** と **コンピュートマシン** は相互に置き換え可能な用語として使用されています。OpenShift Container Platform の今後のバージョンでは、インフラストラクチャーマシンなどの異なる種類のコンピュートマシンがデフォルトで使用される可能性があります。



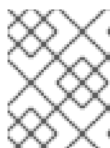
注記

マシンセットは **machine-api** namespace 下のマシンリソースのグループです。マシンセットは、特定のクラウドプロバイダーで新規マシンを起動するように設計されている設定です。マシン設定プール (MCP) は Machine Config Operator (MCO) namespace の一部です。MCP は、MCO がそれらの設定を管理し、それらのアップグレードを容易に実行できるようにマシンをまとめるために使用されます。

3.1.2.2. クラスターのマスター

Kubernetes のクラスターでは、マスターノードは Kubernetes クラスターの制御に必要なサービスを実

行します。OpenShift Container Platform では、マスターマシンはコントロールプレーンになります。これには、OpenShift Container Platform のクラスターを管理する Kubernetes サービス以外も含まれます。コントロールプレーンのロールを持つすべてのマシンがマスターマシンであるため、 **マスター** と **コントロールプレーン** はこれらを説明する際の相互に置き換え可能な用語として使用されています。マスターマシンは、マシンセットにグループ化されるのではなく、一連のスタンドアロンマシン API リソースによって定義されます。 すべてのマスターマシンが削除されてクラスターが切断されないようにするために、追加の制御がマスターマシンに適用されます。



注記

3 つのマスターノードのみが、すべての実稼働デプロイメントで使用される必要があります。

マスター上の Kubernetes カテゴリーに分類されるサービスには、Kubernetes API サーバー、etcd、Kubernetes コントローラーマネージャー、Kubernetes スケジューラーが含まれます。

表3.1 コントロールプレーンで実行される Kubernetes サービス

コンポーネント	説明
API サーバー	Kubernetes API サーバーは 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を検証し、設定します。また、クラスターの共有される状態を確認できる中心的な部分として機能します。
etcd	etcd はマスターの永続的な状態を保存し、他のコンポーネントは etcd で変更の有無を監視して、それぞれを指定された状態に切り替えます。
コントローラーマネージャーサーバー	コントローラーマネージャーサーバーは etcd でレプリケーション、namespace、サービスアカウントコントローラーのオブジェクトへの変更の有無を監視し、API を使用して指定された状態を実行します。このような複数のプロセスは、一度に 1 つのアクティブなリーダーを設定してクラスターを作成します。
Kubernetes スケジューラー	Kubernetes スケジューラーは、割り当て済みのノードなしで新規に作成された Pod の有無を監視し、Pod をホストする最適なノードを選択します。

マスターマシン上のこれらサービスの一部は systemd サービスとして実行し、それ以外は 静的な Pod として実行されます。

systemd サービスは、起動直後の特定のシステムで常に起動している必要のあるサービスに適しています。マスターマシンの場合は、リモートログインを可能にする sshd も含まれます。また、以下のようなサービスも含まれます。

- CRI-O コンテナーエンジン (crio): コンテナーを実行し、管理します。OpenShift Container Platform 4.5 は、Docker Container Engine ではなく CRI-O を使用します。
- Kubelet (kubelet): マシン上で、マスターサービスからのコンテナー管理要求を受け入れます。

CRI-O および Kubelet は、他のコンテナーを実行する前に実行されている必要があるため、systemd サービスとしてホスト上で直接実行される必要があります。

installer-* および **revision-pruner-*** コントロールプレーン Pod は、root ユーザーが所有する **/etc/kubernetes** ディレクトリーに書き込むため、root パーミッションで実行する必要があります。これらの Pod は以下の namespace に置かれます。

- **openshift-etcd**
- **openshift-kube-apiserver**
- **openshift-kube-controller-manager**
- **openshift-kube-scheduler**

3.1.3. OpenShift Container Platform の Operator

OpenShift Container Platform では、Operator はコントロールプレーンでサービスをパッケージ化し、デプロイし、管理するための推奨される方法です。Operator の使用は、ユーザーが実行するアプリケーションにも各種の利点があります。Operator は **kubectl** や **oc** コマンドなどの Kubernetes API および CLI ツールと統合します。Operator はアプリケーションの監視、ヘルスチェックの実行、OTA (over-the-air) 更新の管理を実行し、アプリケーションが指定した状態にあることを確認するための手段となります。

CRI-O と Kubelet はすべてのノード上で実行されるため、Operator を使用することにより、ほぼすべての他のクラスター機能をコントロールプレーンで管理できます。Operator は OpenShift Container Platform 4.5 の最も重要なコンポーネントです。Operator を使用してコントロールプレーンに追加されるコンポーネントには、重要なネットワークおよび認証情報サービスが含まれます。

OpenShift Container Platform クラスターの他の Operator を管理する Operator は Cluster Version Operator です。

OpenShift Container Platform 4.5 は複数の異なるクラスの Operator を使用してクラスター操作を実行し、アプリケーションが使用するクラスターでサービスを実行します。

3.1.3.1. OpenShift Container Platform の Platform Operator

OpenShift Container Platform 4.5 では、すべてのクラスター機能は一連の Platform Operator に分類されます。Platform Operator は、クラスター全体でのアプリケーションロギング、Kubernetes コントロールプレーンの管理、またはマシンプロビジョニングシステムなどの、クラスター機能の特定の分野を管理します。

各 Operator は、クラスター機能を判別するための単純な API を提供します。Operator は、コンポーネントのライフサイクルの管理についての詳細を非表示にします。Operator は単一コンポーネントも、数十のコンポーネントも管理できますが、最終目標は常に、共通アクションの自動化によって操作上の負担の軽減することにあります。また、Operator はより粒度の高いエクスペリエンスも提供します。各コンポーネントは、グローバル設定ファイルではなく、Operator が公開する API を変更して設定できます。

3.1.3.2. OLM によって管理される Operator

クラスターの Operator Lifecycle Management (OLM) コンポーネントはアプリケーションで使用できる Operator を管理します。これは OpenShift Container Platform を設定する Operator を管理しません。OLM は、Kubernetes ネイティブアプリケーションを Operator として管理するフレームワークです。これは、Kubernetes マニフェストを管理する代わりに、Kubernetes Operator を管理します。OLM は Red Hat Operator と認定 Operator の 2 つのクラスの Operator を管理します。

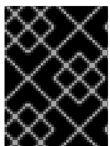
一部の Red Hat Operator は、スケジューラーおよび問題検出機能などのクラスター機能を実行します。他の Operator は etcd などのように各自で管理し、アプリケーションで使用できるように提供されます。また OpenShift Container Platform は、コミュニティーが構築し、保守する認定 Operator を提供します。これらの認定 Operator は API 層を従来のアプリケーションに提供し、アプリケーションを Kubernetes コンストラクトで管理できるようにします。

3.1.3.3. OpenShift Container Platform の更新サービスについて

OpenShift Container Platform の更新サービスとは、OpenShift Container Platform と Red Hat Enterprise Linux CoreOS (RHCOS) の両方に OTA(over-the-air) 更新を提供するホスト型サービスです。コンポーネント Operator のグラフ、または **頂点** とそれらを結ぶ **辺** を含む図表が提示されます。グラフの辺は、どのバージョンであれば安全に更新できるかを示します。 頂点は、管理されたクラスターコンポーネントの想定された状態を特定する更新のペイロードです。

クラスター内の Cluster Version Operator (CVO) は、OpenShift Container Platform の更新サービスをチェックして、グラフの現在のコンポーネントバージョンとグラフの情報に基づき、有効な更新および更新パスを確認します。ユーザーが更新をリクエストすると、OpenShift Container Platform CVO はその更新のリリースイメージを使ってクラスターをアップグレードします。リリースアーティファクトは、コンテナーイメージとして Quay でホストされます。

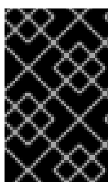
OpenShift Container Platform 更新サービスが互換性のある更新のみを提供できるようにするために、自動化を支援するリリース検証 Pipeline が使用されます。それぞれのリリースアーティファクトについて、他のコンポーネントパッケージだけでなくサポートされているクラウドプラットフォームおよびシステムアーキテクチャーとの互換性の有無が検証されます。Pipeline がリリースの適合性を確認した後に、OpenShift Container Platform 更新サービスは更新が利用可能であることを通知します。



重要

更新サービスが有効な更新をすべて表示するために、更新サービスが表示しないバージョンへの更新を強制することはできません。

連続更新モードでは、2 つのコントローラーが実行されます。1 つのコントローラーはペイロードマニフェストを絶えず更新し、それらをクラスターに適用し、Operator が利用可能か、アップグレード中か、または失敗しているかに応じて Operator の制御されたロールアウトのステータスを出力します。2 つ目のコントローラーは OpenShift Container Platform 更新サービスをポーリングして、更新が利用可能かどうかを判別します。



重要

クラスターを以前のバージョンに戻すこと、つまりロールバックはサポートされていません。サポートされているのは、新規バージョンへのアップグレードのみです。アップグレードできない場合は、Red Hat サポートにお問い合わせください。

アップグレードプロセスで、Machine Config Operator (MCO) は新規設定をクラスターマシンに適用します。これは、マシン設定プールの **maxUnavailable** フィールドによって指定されるノードの数を分離し、それらを利用不可としてマークします。デフォルトで、この値は **1** に設定されます。次に、新しい設定を適用して、マシンを再起動します。Red Hat Enterprise Linux (RHEL) マシンをワーカーとして使用する場合、まず OpenShift API をそれらのマシンで更新する必要があるため、MCO はそれらのマシンで kubelet を更新しません。新規バージョンの仕様は古い kubelet に適用されるため、RHEL マシンを **Ready** 状態に戻すことができません。マシンが利用可能になるまで更新を完了することはできません。ただし、利用不可のノードの最大数は、その数のマシンがサービス停止状態のマシンとして分離されても通常のクラスター操作が継続できるようにするために設定されます。

3.1.3.4. Machine Config Operator について

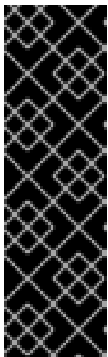
OpenShift Container Platform 4.5 は、オペレーティングシステムとクラスター管理を統合します。クラスターは、クラスターノードでの Red Hat Enterprise Linux CoreOS (RHCOS) への更新を含め、独自の更新を管理するので、OpenShift Container Platform では事前に設定されたライフサイクル管理が実行され、ノードのアップグレードのオーケストレーションが単純化されます。

OpenShift Container Platform は、ノードの管理を単純化するために 3 つのデーモンセットとコントローラーを採用しています。これらのデーモンセットは、Kubernetes 形式のコンストラクトを使用してオペレーティングシステムの更新とホストの設定変更をオーケストレーションします。これには、以下が含まれます。

- **machine-config-controller** : コントロールプレーンからマシンのアップグレードを調整します。すべてのクラスターノードを監視し、その設定の更新をオーケストレーションします。
- **machine-config-daemon** デーモンセット: クラスターの各ノードで実行され、マシン設定で定義された設定で、MachineConfigController の指示通りにマシンを更新します。 ノードは、変更を検知すると Pod からドレイン (解放) され、更新を適用して再起動します。これらの変更は、指定されたマシン設定を適用し、 kubelet 設定を制御する Ignition 設定ファイルの形式で実行されます。更新自体はコンテナーで行われます。このプロセスは、OpenShift Container Platform と RHCOS の更新を同時に管理する際に不可欠です。
- **machine-config-server** デーモンセット: コントロールプレーンノードがクラスターに参加する際に Ignition 設定ファイルをコントロールプレーンノードに提供します。

このマシン設定は Ignition 設定のサブセットです。**machine-config-daemon** はマシン設定を読み取り、OSTree の更新を行う必要があるか、または一連の systemd kubelet ファイルの変更、設定の変更、オペレーティングシステムまたは OpenShift Container Platform 設定などへのその他の変更を適用する必要があるかを確認します。

ノード管理操作の実行時に、**KubeletConfig** カスタムリソース (CR) を作成または変更します。



重要

マシン設定への変更が行われると、Machine Config Operator は変更を有効にするために、対応するすべてのノードを自動的に再起動します。

マシン設定の変更後、変更が適用される前にノードが自動的に起動されないようにするには、対応するマシンプール設定で **spec.paused** フィールドを **true** に設定して自動再起動プロセスを一時停止する必要があります。一時停止すると、**spec.paused** フィールドを **false** に設定し、ノードが新しい設定で再起動されるまで、マシン設定の変更は適用されません。

追加情報

Machine Config Operator によるマシン設定の変更後にコントロールプレーンマシンが自動的に起動されないようにする方法については、Disabling Machine Config Operator from automatically rebooting を参照してください。

第4章 OPENSHIFT CONTAINER PLATFORM の開発について

高品質のエンタープライズアプリケーションの開発および実行時にコンテナーの各種機能をフルに活用できるようにするには、使用する環境が、コンテナーの以下の機能を可能にするツールでサポートされている必要があります。

- 他のコンテナー化された/されていないサービスに接続できる分離したマイクロサービスとして作成される。たとえば、アプリケーションをデータベースに結合するが、またはアプリケーションに監視アプリケーションを割り当てることが必要になることがあります。
- 回復性がある。 サーバーがクラッシュしたときやメンテナーンスのために停止する必要があるとき、またはまもなく使用停止になる場合などに、コンテナーを別のマシンで起動することができます。
- 自動化されている。コードの変更を自動的に選択し、新規バージョンの起動およびデプロイを自動化します。
- スケールアップまたは複製が可能である。 需要の上下に合わせてクライアントに対応するインスタンスの数を増やしたり、インスタンスの数を減らしたりできます。
- アプリケーションの種類に応じて複数の異なる方法で実行できる。たとえば、あるアプリケーションは月一回実行してレポートを作成した後に終了させる場合があります。別のアプリケーションは継続的に実行して、クライアントに対する高可用性が必要になる場合があります。
- 管理された状態を保つ。 アプリケーションの状態を監視し、異常が発生したら対応できるようにします。

コンテナーが広く浸透し、エンタープライズレベルでの対応を可能にするためのツールや方法への要求が高まっていることにより、多くのオプションがコンテナーで利用できるようになりました。

このセクションの残りの部分では、 OpenShift Container Platform で Kubernetes のコンテナー化されたアプリケーションをビルドし、デプロイする際に作成できるアセットの各種のオプションについて説明します。また、各種の異なるアプリケーションや開発要件に適した方法についても説明します。

4.1. コンテナー化されたアプリケーションの開発について

コンテナーを使用したアプリケーションの開発にはさまざまな方法を状況に合わせて使用できます。単一コンテナーの開発から、最終的にそのコンテナーの大企業のミッションクリティカルなアプリケーションとしてのデプロイに対応する一連の方法の概要を示します。それぞれのアプローチと共に、コンテナー化されたアプリケーションの開発に使用できる各種のツール、フォーマットおよび方法を説明します。扱う内容は以下の通りです。

- 単純なコンテナーをビルドし、レジストリーに格納する
- Kubernetes マニフェストを作成し、それを Git リポジトリーに保存する
- Operator を作成し、アプリケーションを他のユーザーと共有する

4.2. 単純なコンテナーのビルド

たとえば、アプリケーションをコンテナー化しようと考えているとします。

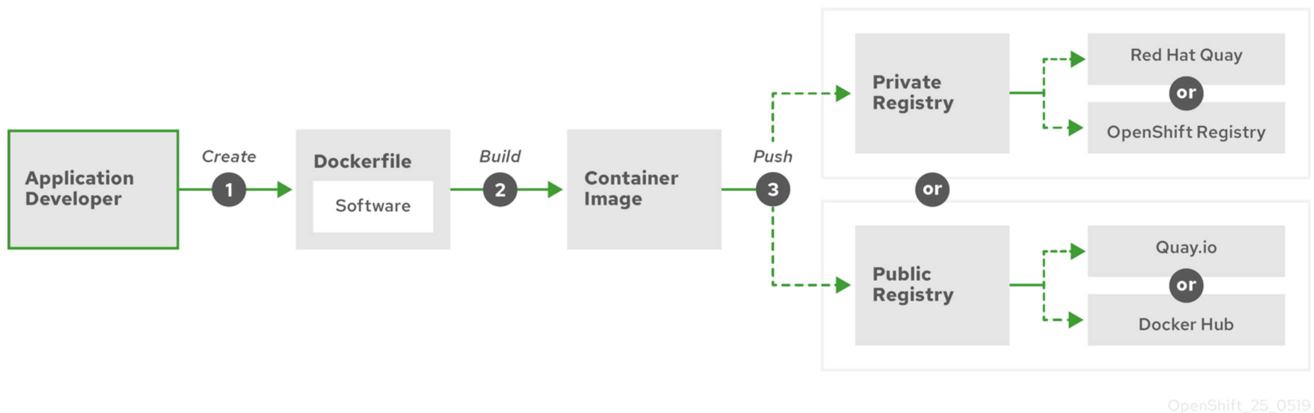
その場合、まず必要になるのは buildah や docker などのコンテナーをビルドするためのツール、およびコンテナーの内部で実行されることを記述したファイルです。 これは通常、Dockerfile になります。

次に、作成したコンテナーイメージをプッシュする場所が必要になります。ここからコンテナーイメージをプルすると、任意の場所で実行することができます。この場所はコンテナーレジストリーになります。

各コンポーネントのサンプルは、ほとんどの Linux オペレーティングシステムにデフォルトでインストールされています。 ただし Dockerfile はユーザーが各自で用意する必要があります。

以下の図は、イメージをビルドし、プッシュするプロセスを示しています。

図4.1 単純なコンテナー化アプリケーションを作成し、レジストリーにプッシュする



Red Hat Enterprise Linux (RHEL) をオペレーティングシステムとして実行しているコンピューターを使用している場合、コンテナー化されているアプリケーションを作成するプロセスには以下の手順が必要になります。

1. コンテナービルドツールのインストール。 RHEL には、コンテナーのビルドと管理に使用される podman、buildah、skopeo など一連のツールが含まれています。

2. Dockerfile を作成してベースイメージとソフトウェアを組み合わせる。コンテナーのビルドに関する情報は、**Dockerfile** というファイルに保管されます。このファイルでビルドの起点となるベースイメージ、インストールするソフトウェアパッケージ、コンテナーにコピーするソフトウェアを指定します。さらに、コンテナーの外部に公開するネットワークポートやコンテナーの内部にマウントするボリュームのなどのパラメーター値も指定します。Dockerfile とコンテナー化するソフトウェアは、 RHEL システムのディレクトリーに配置します。

3. buildah または docker build を実行する。**buildah build-using-dockerfile** または **docker build** コマンドを実行し、選択したベースイメージをローカルシステムにプルして、ローカルに保存されるコンテナーイメージを作成します。 buildah を使用して、Dockerfile なしにコンテナーイメージをビルドすることもできます。

4. タグ付けおよびレジストリーへのプッシュを実行します。コンテナーの格納および共有に使用するレジストリーの場所を特定する新しいコンテナーイメージにタグを追加します。次に、**podman push** または **docker push** コマンドを実行してそのイメージをレジストリーにプッシュします。

5. イメージをプルして実行する。Podman や Docker などのコンテナークライアントツールがある任意のシステムから、新しいイメージを特定するコマンドを実行します。たとえば、**podman run <image_name>** や **docker run <image_name>** のコマンドを実行します。ここで、**<image_name>** は新しいイメージの名前であり、**quay.io/myrepo/myapp:latest** のようになります。レジストリーでは、イメージをプッシュおよびプルするために認証情報が必要になる場合があります。

コンテナーイメージをビルドし、レジストリーにプッシュし、それらを実行するプロセスについての詳細は、Custom image builds with Buildah を参照してください。

4.2.1. コンテナービルドツールのオプション

Docker Container Engine と **docker** コマンドは、 RHEL や他の多くの Linux システムでコンテナーを使用する際に使う一般的なツールですが、代わりに、Podman、Skopeo、Buildah を含む、異なるコンテナーツールのセットを選択することもできます。さらに、Docker Container Engine ツールを使用して、OpenShift Container Platform やその他のコンテナープラットフォームで動作するコンテナーを作成することもできます。

Buildah、Podman、Skopeo を使用してコンテナービルドし、管理すると、それらのコンテナーを最終的に OpenShift Container Platform またはその他の Kubernetes 環境にデプロイする目的で調整された各種機能が含まれる業界標準のコンテナーイメージが生成されます。これらのツールにはデーモンが不要であり、root 権限なしで実行できるので、これらのツールを実行するときはオーバーヘッドが少なくてすみます。

コンテナーを最終的に OpenShift Container Platform で実行するときは、CRI-O コンテナーエンジンを使用します。CRI-O は、OpenShift Container Platform クラスターのすべてのワーカーマシンおよびマスターマシン上で実行されますが、CRI-O は、OpenShift Container Platform の外部のスタンドアロンランタイムとしてはまだサポートされていません。

4.2.2. ベースイメージのオプション

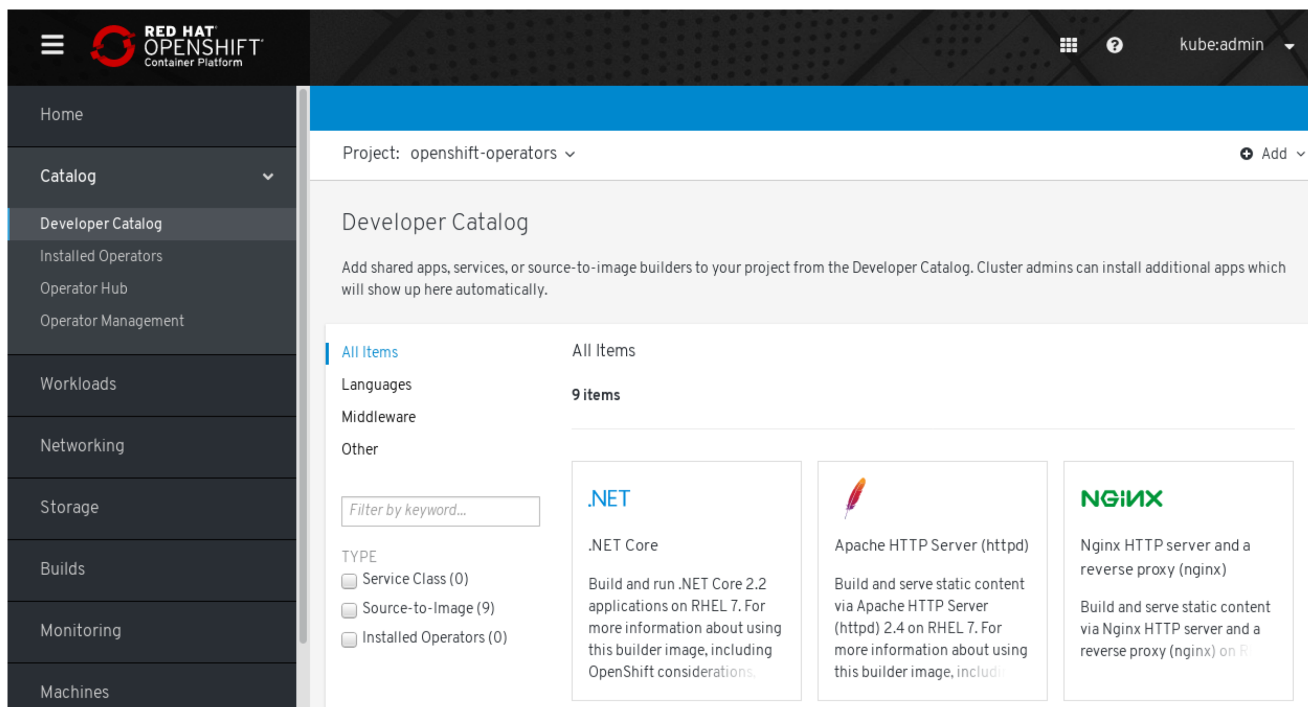
アプリケーションをビルドするために選択するベースイメージには、Linux システムがアプリケーションのように表示されるソフトウェアのセットが含まれます。ユーザーが独自のイメージをビルドする場合、ソフトウェアはそのファイルシステム内に配置され、ファイルシステムはオペレーティングシステムのように表示されます。このベースイメージの選択により、コンテナーの将来の安全性、効率性およびアップグレードの可能性に大きな影響を与えます。

Red Hat は、Red Hat Universal Base Images (UBI) と呼ばれるベースイメージの新たなセットを提供します。これらのベースイメージは、Red Hat Enterprise Linux をベースとし、Red Hat が過去に提供してきたベースイメージに類似したものですが、次の大きな違いがあります。UBI は、Red Hat のサブスクリプションなしの再ディストリビューションが可能です。 そのため、UBI イメージの共有方法を検討したり、環境ごとに異なるイメージを作成する必要なしに、UBI イメージでアプリケーションをビルドすることができます。

これらの UBI イメージは、標準で最小限の init バージョンです。また、Red Hat Software Collections イメージを、Node.js、Perl、Python などの特定のランタイム環境に依存するアプリケーションの基盤として使用することができます。これらのランタイムベースイメージの特殊なバージョンは Source-to-image (S2I) イメージと呼ばれています。S2I イメージを使用して、コードを、そのコードを実行できるベースイメージ環境に挿入することができます。

S2I イメージは、以下の図に示すように、OpenShift Container Platform Web UI の**Catalog** → **Developer Catalog** を選択することにより直接使用することができます。

図4.2 特定のランタイムを必要とするアプリケーションの S2I ベースイメージを選択する



4.2.3. レジストリーオプション

コンテナーレジストリーはコンテナーイメージを保管する場所です。ここから、コンテナーイメージを他の人と共有したり、最終的に実行するプラットフォームで使用できるようにしたりできます。無料アカウントを提供する大規模なパブリックコンテナーレジストリーや、容量の大きいストレージや特殊な機能を備えたプレミアムバージョンを選択することができます。また、ご自身の組織にのみ限定される独自のレジストリーをインストールしたり、共有する相手を選択して独自のレジストリーをインストールすることもできます。

Red Hat のイメージおよび認定パートナーのイメージを取得するには、Red Hat レジストリーから取り出すことができます。Red Hat レジストリーは、認証されていない非推奨の **registry.access.redhat.com**、および認証が必要な **registry.redhat.io** の 2 つの場所によって表わされます。Container images section of the Red Hat Ecosystem Catalog で、Red Hat レジストリーの Red Hat イメージおよびパートナーのイメージを確認することができます。これは、Red Hat コンテナーイメージのを一覧表示するだけでなく、適用されたセキュリティー更新に基づくヘルススコアなどの、これらのイメージのコンテンツと品質に関する広範な情報も表示します。

大規模なパブリックレジストリーには、Docker Hub および Quay.io が含まれます。Quay.io レジストリーは Red Hat が所有し、管理しています。OpenShift Container Platform で使用されるコンポーネントの多くは Quay.io に保管されます。これには OpenShift Container Platform 自体をデプロイするために使用されるコンテナーイメージおよび Operator が含まれます。Quay.io は、Helm チャートなどの他のタイプのコンテンツを保管する手段ともなります。

専用のプライベートコンテナーレジストリーが必要な場合、OpenShift Container Platform 自体にプライベートコンテナーレジストリーが含まれています。これは、OpenShift Container Platform と共にインストールされ、そのクラスター上で実行されます。また、Red Hat は <blink>Red Hat Quay</blink> と呼ばれるプライベートバージョンの Quay.io レジストリーも提供しています。Red Hat Quay には、geo レプリケーション、Git ビルドトリガー、Clair イメージスキャンなどの多くの機能が含まれています。

ここで言及したすべてのレジストリーでは、これらのレジストリーからイメージをダウンロードする際に認証情報が必要です。これらの認証情報の一部は OpenShift Container Platform のクラスター全体に提供されますが、他の認証情報は個別に割り当てられます。

4.3. OPENSHIFT CONTAINER PLATFORM 用の KUBERNETES マニフェストの作成

コンテナーイメージは、コンテナー化されたアプリケーション用の基本的なビルディングブロックですが、OpenShift Container Platform などの Kubernetes 環境でそのアプリケーションを管理し、デプロイするには、より多くの情報が必要になります。イメージ作成後に実行される通常の手順は以下のとおりです。

- Kubernetes Manifest マニフェストで使用する各種リソースを理解する
- 実行するアプリケーションの種類についての決定
- サポートコンポーネントの収集
- マニフェストの作成、およびそのマニフェストの Git リポジトリーへの保管。これにより、マニフェストをソースバージョン管理システムに保管し、監査と追跡、次の環境へのプロモートとデプロイ、必要な場合は以前のバージョンへのロールバックなどを実行でき、これを他者と共有することができます。

4.3.1. Kubernetes Pod およびサービスについて

コンテナーイメージは Docker を使用する基本単位であり、Kubernetes が使用する基本単位は Pod と呼ばれます。Pod はアプリケーションのビルドの次の手順で使用されます。Pod には、1 つ以上のコンテナーを含めることができます。Pod はデプロイやスケーリングおよび管理を実行する単一の単位であることに留意してください。

Pod での実行内容を決定する際に考慮する必要のある主要な点として、スケーラビリティーと namespace を考慮することができます。デプロイメントを容易にするには、コンテナーを Pod にデプロイして、Pod 内に独自のロギングとモニターリングコンテナーを含めることができるかもしれません。後に、Pod を実行し、追加のインスタンスをスケールアップすることが必要になると、それらの他のコンテナーもスケールアップできます。namespace の場合、Pod 内のコンテナーは同じネットワークインターフェイス、共有ストレージボリューム、メモリーや CPU などのリソース制限を共有します。これにより、Pod のコンテンツを単一の単位として管理することが容易になります。また Pod 内のコンテナーは、System V セマフォや POSIX 共有メモリーなどの標準的なプロセス間通信を使用することにより、相互に通信することができます。

個々の Pod が Kubernetes 内のスケーラブルな単位を表すのに対し、サービス は、負荷分散などの完全なタスクを実行する完全で安定したアプリケーションを作成するために複数の Pod をグループ化する手段を提供します。 また、サービスは削除されるまで同じ IP アドレスで利用可能な状態になるため、Pod より永続性があります。サービスが使用できる状態の場合、サービスは名前で要求され、OpenShift Container Platform クラスターはその名前を IP アドレスとポートに解決し、そこからサービスを設定する Pod に到達することができます。

性質上、コンテナー化されたアプリケーションは、それらが実行されるオペレーティングシステムから分離され、したがってユーザーからも分離されます。Kubernetes マニフェストの一部には、コンテナー化されたアプリケーションとの通信の詳細な制御を可能にする ネットワークポリシー を定義して、アプリケーションを内外のネットワークに公開する方法が記述されています。HTTP、HTTPS の受信要求やクラスター外からの他のサービスをクラスター内のサービスに接続するには、**Ingress** リソースを使用することができます。

コンテナーが、サービスを通じて提供されるデータベースストレージではなくディスク上のストレージを必要とする場合、ボリューム をマニフェストに追加して、そのディスクを Pod で使用可能にすることができます。永続ボリューム (PV) を作成するか、**Pod** 定義に追加されるボリュームを動的に作成するようにマニフェストを設定することができます。

アプリケーションを設定する Pod のグループを定義した後に、それらの Pod を **Deployment** および **DeploymentConfig** オブジェクトで定義することができます。

4.3.2. アプリケーションのタイプ

次に、アプリケーションのタイプが、その実行方法にどのように影響するかを検討します。

Kubernetes は、各種のアプリケーションに適した異なるタイプのワークロードのタイプを定義します。アプリケーションに適したワークロードを決定するために、アプリケーションが以下のどのタイプに該当するかを確認してください。

- 完了まで実行されることが意図されている。 例として、アプリケーションはレポートを作成するために起動される場合、レポートの完了時に終了することが想定されます。このアプリケーションはその後一ヵ月間再度実行されない場合もあります。これらのタイプのアプリケーションに適した OpenShift Container Platform オブジェクトには、**Job** および **CronJob** オブジェクトがあります。

- 継続的に実行することが予想されている。 長時間実行されるアプリケーションの場合、 デプロイメント を作成することができます。

- 高い可用性が必要。 使用しているアプリケーションに高可用性が必要な場合は、2 つ以上のインスタンスを持てるようにデプロイメントのサイズを設定する必要があります。**Deployment** または **DeploymentConfig** オブジェクトの場合、このタイプのアプリケーション用に レプリカセット を組み込むことができます。レプリカセットを使用すると、Pod は複数のノード間で実行され、ワーカーが停止してもアプリケーションを常に利用可能な状態にすることができます。

- すべてのノード上で実行される必要がある。 Kubernetes アプリケーションのタイプによっては、すべてのマスターまたはワーカーノード上のクラスターで実行することが意図されています。DNS およびモニターリングアプリケーションは、すべてのノード上で継続的に実行する必要があるアプリケーションの例です。このタイプのアプリケーションは、デーモンセット として実行することができます。また、デーモンセットはノードラベルに基づいて、ノードのサブセット上でも実行できます。

- ライフサイクル管理を必要とする。 アプリケーションが他者も使用できるようにする場合には、Operator を作成することを検討してください。Operator を使用すると、インテリジェンスを組み込むことが可能となり、自動的なバックアップやアップグレードなどを自動かできます。Operator Lifecycle Manager (OLM) と組み合わせることで、クラスターマネージャーは、Operator を選択された namespace に公開し、クラスター内のユーザーが Operator を実行できるようになります。

- アイデンティティーまたは番号付けの要件がある。アプリケーションには、アイデンティティーや番号付けの要件がある場合があります。 たとえば、アプリケーションの 3 つのインスタンスのみを実行し、インスタンスに **0**、**1**、**2** という名前を付けることが求められる場合があります。このアプリケーションには、ステートフルセット が適しています。 ステートフルセットは、データベースや zookeeper クラスターなどの独立したストレージが必要なアプリケーションに最も適しています。

4.3.3. 利用可能なサポートコンポーネント

作成するアプリケーションには、データベースやロギングコンポーネントなどのサポートコンポーネントが必要な場合があります。このニーズに対応するために、OpenShift Container Platform の Web コンソールで利用可能な以下のカタログから必要なコンポーネントを取得できる場合があります。

- OperatorHub: 各 OpenShift Container Platform 4.5 クラスターで利用できます。OperatorHub により、Red Hat、認定 Red Hat パートナー、コミュニティーメンバーおよびクラスターのオ

ペレーターなどから Operator が利用可能になります。クラスター Operator は、それらの Operator をクラスター内のすべての場所 または選択された namespace で利用可能にします。そのため、開発者は Operator を起動し、それらをアプリケーションと共に設定することができます。

- テンプレート: ワンオフタイプのアプリケーションの場合に役立ちます。この場合、インストール後のコンポーネントのライフサイクルは重要ではありません。テンプレートは、最小限のオーバーヘッドで Kubernetes アプリケーションの開発を始める簡単な方法を提供します。テンプレートは、**Deployment**、**Service**、**Route**、またはその他のオブジェクトなどのリソース定義の一覧である場合があります。名前またはリソースを変更する必要がある場合に、それらの値をパラメーターとしてテンプレートに設定できます。

サポートする Operator およびテンプレートは、開発チームの特定のニーズに合わせて設定し、開発者が作業に使用する namespace で利用可能にすることができます。 多くの場合、共有テンプレートは他のすべての namespace からアクセス可能になるために **openshift** namespace に追加されます。

4.3.4. マニフェストの適用

Kubernetes マニフェストを使用して、Kubernetes アプリケーションを設定するコンポーネントのより詳細な情報を得ることができます。これらのマニフェストは YAML ファイルとして作成し、**oc apply** などのコマンドを実行して、それらをクラスターに適用してデプロイできます。

4.3.5. 次のステップ

この時点で、コンテナー開発のプロセスを自動化する方法を検討します。この際、イメージをビルドしてレジストリーにプッシュするいくつかの CI パイプラインがあることが望ましいと言えます。とくに GitOps パイプラインは、アプリケーションのビルドに必要なソフトウェアを保管する Git リポジトリーにコンテナー開発を統合します。

ここまでのワークフローは以下のようになります。

- Day 1: YAML を作成します。次に **oc apply** コマンドを実行して、YAML をクラスターに適用し、機能することを確かめます。
- Day 2: YAML コンテナー設定ファイルを独自の Git リポジトリーに配置します。ここから、アプリのインストールやその改善の支援に携わるメンバーが YAML をプルダウンし、アプリを実行するクラスターにこれを適用できます。
- Day 3: アプリケーション用の Operator の作成を検討します。

4.4. OPERATOR 向けの開発

アプリケーションを他者が実行できるようにする場合、アプリケーションを Operator としてパッケージ化し、デプロイすることが適切である場合があります。前述のように、Operator は、ライフサイクルコンポーネントをアプリケーションに追加し、インストール後すぐにアプリケーションを実行するジョブが完了していないことを認識します。

アプリケーションを Operator として作成する場合、アプリケーションを実行し、維持する方法についての独自のノウハウを盛り込むことができます。アプリケーションのアップグレード、バックアップ、スケーリング、状態のトラッキングなどを行う機能を組み込むことができます。アプリケーションを正しく設定すれば、Operator の更新などのメンテナーンスタスクは、Operator のユーザーに非表示の状態で自動的に実行されます。

役に立つ Operator の一例として、データを特定のタイミングで自動的にバックアップするように設定された Operator を挙げることができます。Operator は設定されたタイミングでのアプリケーションの

バックアップを管理するため、システム管理者はバックアップのタイミングを覚えておく必要がありません。

データのバックアップや証明書のローテーションなど、これまで手作業で行われていたアプリケーションのメンテナーンスは、Operator によって自動化されます。

第5章 RED HAT ENTERPRISE LINUX COREOS (RHCOS)

5.1. RHCOS について

Red Hat Enterprise Linux に CoreOS (RHCOS) は、次世代の専用コンテナーオペレーティングシステムテクノロジーの代表格です。Red Hat Enterprise Linux Atomic Host と CoreOS Container Linux を開発した同じチームによって開発された RHCOS は、Red Hat Enterprise Linux (RHEL) の品質基準と Container Linux の自動化されたリモートアップグレード機能を兼ね備えています。

RHCOS は、すべての OpenShift Container Platform マシンの 1 つの OpenShift Container Platform 4.5 コンポーネントとしてのみサポートされます。RHCOS は、OpenShift Container Platform のコントロールプレーンまたはマスターマシン向けに唯一サポートされるオペレーティングシステムです。RHCOS はすべてのクラスターマシンのデフォルトオペレーティングシステムですが、 RHEL をオペレーティングシステムとして使用するコンピュートマシン (ワーカーマシンとしても知られる) を作成することもできます。RHCOS を OpenShift Container Platform 4.5 にデプロイするには、以下の 2 つの一般的な方法があります。

- クラスターがプロビジョニングするインフラストラクチャーにクラスターをインストールする場合は、インストール時に RHCOS イメージがターゲットのプラットフォームにダウンロードされ、RHCOS 設定を制御する適切な Ignition 設定ファイルがマシンをデプロイするために使用されます。

- クラスターを独自に管理するインフラストラクチャーにインストールする場合、インストールについてのドキュメントを参照して RHCOS イメージの取得や、Ignition 設定ファイルの生成、および設定ファイルの使用によるマシンのプロビジョニングを行ってください。

5.1.1. RHCOS の主な機能

以下は、RHCOS オペレーティングシステムの主要な機能について説明しています。

- **Based on RHEL** (RHEL ベース): この基礎となるオペレーティングシステムは、主に RHEL のコンポーネントで設定されます。RHEL をサポートする品質、セキュリティーおよび管理基準が RHCOS にも同様に適用されます。たとえば、RHCOS ソフトウェアは RPM パッケージにあり、各 RHCOS システムは、RHEL カーネル、および systemd init システムで管理される一連のサービスと共に起動します。

- **Controlled immutability** (不変性の制御): RHCOS には、RHEL コンポーネントが含まれているものの、RHCOS はデフォルトの RHEL インストールの場合よりも厳密に管理されるように設計されています。これは OpenShift Container Platform クラスターからリモートで管理されます。RHCOS マシンをセットアップする際には、いくつかのシステム設定のみを変更するだけで対応することが可能です。RHCOS のこの管理機能および不変性により、OpenShift Container Platform では RHCOS システムの最新の状態をクラスターに保存し、最新の RHCOS 設定に基づいて追加のマシンの作成や更新を実行することができます。

- **CRI-O container runtime**(CRI-O コンテナーランタイム): RHCOS には Docker で必要な OCI および libcontainer 形式のコンテナーを実行する機能が含まれていますが、Docker コンテナーエンジンではなく CRI-O コンテナーエンジンが組み込まれています。OpenShift Container Platform などの、Kubernetes プラットフォームが必要とする機能に重点が置かれている CRI-O では、複数の異なる Kubernetes バージョンとの特定の互換性が提供されます。また CRI-O は、大規模な機能セットを提供するコンテナーエンジンの場合よりもフットプリントや攻撃領域を縮小できます。現時点で、 CRI-O は OpenShift Container Platform クラスター内で利用可能な唯一のエンジンです。

- **Set of container tools**(コンテナーツールのセット): ビルド、コピーまたはコンテナーの管理などのタスクに備え、RHCOS では Docker CLI ツールが互換性のあるコンテナーツールセット

に置き換えられます。Podman CLI ツールは、コンテナーおよびコンテナーイメージの実行、起動、停止、一覧表示および削除などの数多くのコンテナーランタイム機能をサポートします。skopeo CLI ツールは、イメージのコピー、認証およびイメージへの署名を実行できます。**crictl** CLI ツールを使用すると、CRI-O コンテナーエンジンからコンテナーおよび Pod を使用できます。これらのツールを RHCOS 内で直接使用することは推奨されていませんが、デバッグの目的で使用することは可能です。

- **rpm-ostree upgrades**: RHCOS は、**rpm-ostree** システムを使用したトランザクショナルアップグレードを特長としています。更新はコンテナーイメージ経由で提供され、OpenShift 更新プロセスの一部となっています。コンテナーイメージはデプロイされると、プルされ、展開されてディスクに書き込まれます。その後にブートローダーが新規バージョンで起動するように変更されます。 マシンはローリング方式で更新に対して再起動し、クラスターの容量への影響が最小限に抑えられます。

- **bootupd ファームウェアおよびブートローダー更新ツール**: パッケージマネージャーおよび **rpm-ostree** などのハイブリッドシステムはファームウェアやブートローダーを更新しません。**bootupd** を使用する RHCOS ユーザーは、x86_64、ppc64le、および aarch64 などの最新のアーキテクチャーで実行される UEFI およびレガシー BIOS ブートモードのファームウェアおよびブートの更新を管理するシステムに依存しない更新ツールにアクセスできます。
  bootupd のインストール方法の詳細は、**bootupd を使用したブートローダーの更新**についてのドキュメントを参照してください。

- **Updated through the Machine Config Operator**(MachineConfigOperator による更新): OpenShift Container Platform では、Machine Config Operator がオペレーティングシステムのアップグレードを処理します。**yum** で実行される場合のように個々のパッケージをアップグレードする代わりに、**rpm-ostree** は OS のアップグレードを atomic 単位として提供します。新規の OS デプロイメントはアップグレード時に段階が設定され、次回の再起動時に実行されます。アップグレードに不具合が生じた場合は、単一ロールバックおよび再起動によってシステムが以前の状態に戻ります。OpenShift Container Platform での RHCOS アップグレードは、クラスターの更新時に実行されます。

RHCOS システムの場合、**rpm-ostree** ファイルシステムのレイアウトには、以下の特徴があります。

- **/usr**: オペレーティングシステムのバイナリーとライブラリーが保管される場所で、読み取り専用です。これを変更することはサポートされていません。

- **/etc**、**/boot**、**/var**: システム上で書き込み可能ですが、Machine Config Operator によってのみ変更されることが意図されています。

- **/var/lib/containers**: コンテナーイメージを保管するためのグラフストレージの場所です。

5.1.2. RHCOS の設定方法の選択

RHCOS は、最低限のユーザー設定で OpenShift Container Platform クラスターにデプロイするように設計されています。これは最も基本的な形式であり、以下で設定されます。

- AWS など、プロビジョニングされたインフラストラクチャーの使用を開始するか、またはインフラストラクチャーを独自にプロビジョニングします。

- **openshift-install** の実行時に、**install-config.yaml** ファイルに認証情報およびクラスター名などの一部の情報を指定します。

OpenShift Container Platform の RHCOS システム は OpenShift Container Platform クラスターから完全に管理されるように設計されているため、RHCOS マシンに直接ログインすることは推奨されていません。RHCOS マシンクラスターへの直接のアクセスはデバッグ目的で制限的に実行できますが、

RHCOS システムを直接設定することはできません。その代わりに、OpenShift Container Platform ノードに機能を追加または変更する必要がある場合は、以下の方法で変更を行うことを検討してください。

- **Kubernetes ワークロードオブジェクト (DaemonSet、Deployment など)**: サービスや他のユーザーレベルの機能をクラスターに追加する必要がある場合、Kubernetes ワークロードオブジェクトとしてそれらを追加することを検討します。特定のノード設定以外にこれらの機能を保持することは、後続のアップグレードでクラスターを破損させるリスクを軽減する上での最も効果的な方法です。

- **Day-2 カスタマイズ**: 可能な場合は、クラスターノードをカスタマイズせずにクラスターを起動し、クラスターを起動してから必要なノードの変更を加えます。これらの変更については、後で追跡することが容易であり、更新に支障が及ぶ可能性がより低くなります。マシン設定の作成または Operator カスタムリソースの変更により、これらのカスタマイズを行うことができます。

- **Day-1 カスタマイズ**: クラスターの初回起動時に実装する必要のあるカスタマイズの場合、初回起動時に変更が実装されるようにクラスターを変更する方法があります。Day-1 のカスタマイズは、**openshift-install** の実行時に Ignition 設定およびマニフェストファイルを使用して実行することも、ユーザーがプロビジョニングする ISO インストール時に起動オプションを追加して実行することもできます。

以下は、Day-1 で実行できるカスタマイズの例です。

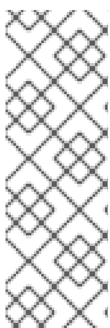
- **カーネル引数**: 特定のカーネル機能やチューニングがクラスターの初回起動時にノードで必要になる場合。

- **ディスクの暗号化**: セキュリティー上、FIPS サポートなど、ノード上のルートファイルシステムが暗号化されている必要がある場合。

- **カーネルモジュール**: ネットワークカードやビデオカードなど、特定のハードウェアデバイスに Linux カーネル内でデフォルトで使用可能なモジュールがない場合。

- **chronyd**: タイムサーバーの場所など、特定のクロック設定をノードに指定する必要がある場合

これらのタスクを実行する上で、**openshift-install** プロセスを拡張して **MachineConfig** などの追加のオブジェクトを含めることができます。マシン設定を作成するこれらの手順は、クラスターの起動後に Machine Config Operator に渡すことができます。



注記

インストールプログラムが生成する Ignition 設定ファイルには、24 時間が経過すると期限切れになり、その後に更新される証明書が含まれます。証明書を更新する前にクラスターが停止し、24 時間経過した後にクラスターを再起動すると、クラスターは期限切れの証明書を自動的に復元します。例外として、kubelet 証明書を回復するために保留状態の **node-bootstrapper** 証明書署名要求 (CSR) を手動で承認する必要があります。詳細は、**コントロールプレーン証明書からの回復** についてのドキュメントを参照してください。

5.1.3. RHCOS の設定方法の選択

OpenShift Container Platform の RHCOS インストールの相違点は、インストーラー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にデプロイするかどうかによって異なります。

- **インストーラーによるプロビジョニング**: 一部のクラウド環境は、最低限の設定で OpenShift Container Platform クラスターを起動することを可能にする事前に設定されたインフラストラ

クチャーを提供しています。このようなタイプのインストールでは、各ノードにコンテンツを配置する Ignition 設定を指定することができ、この場所でクラスターの初回時の起動が行われます。

- **ユーザーによるプロビジョニング**: 独自のインフラストラクチャーをプロビジョニングする場合、データを RHCOS ノードに追加する方法により柔軟性を持たせることができます。たとえば、RHCOS ISO インストーラーを起動して各システムをインストールする場合は、カーネル引数を追加できます。ただし、オペレーティングシステム自体で設定が必要となるほとんどの場合において、Ignition 設定で設定を指定する方法が最も適しています。

Ignition 機能は、RHCOS システムの初回セットアップ時にのみ実行されます。その後は、Ignition 設定はマシン設定を使用して指定できます。

5.1.4. Ignition について

Ignition は、初回設定時にディスクを操作するために RHCOS によって使用されるユーティリティーです。これにより、ディスクのパーティション設定やパーティションのフォーマット、ファイル作成、ユーザー設定などの一般的なディスク関連のタスクが実行されます。初回起動時に、Ignition はインストールメディアや指定した場所からその設定を読み込み、その設定をマシンに適用します。

クラスターをインストールする場合かマシンをクラスターに追加する場合かを問わず、Ignition は常に OpenShift Container Platform クラスターマシンの初期設定を実行します。実際のシステム設定のほとんどは、各マシン自体で行われます。各マシンで、Ignition は、RHCOS イメージを取得し、RHCOS カーネルを起動します。カーネルコマンドラインのオプションで、デプロイメントのタイプや Ignition で有効にされた初期 RAM ディスク (initramfs) の場所を特定します。

5.1.4.1. Ignition の仕組み

Ignition を使用してマシンを作成するには、Ignition 設定ファイルが必要です。OpenShift Container Platform のインストレーションプログラムは、クラスターを作成するのに必要な Ignition 設定ファイルを作成します。これらのファイルは、インストレーションプログラムに直接指定するか、または **install-config.yaml** ファイルを通じて提供される情報に基づくものです。

Ignition がマシンを設定する方法は、cloud-init や Linux Anaconda kickstart などのツールがシステムを設定する方法に似ていますが、以下のような重要な違いがあります。

- Ignition はインストール先のシステムから分離され初期 RAM ディスクから実行されます。そのため、Ignition はディスクのパーティション設定を再度実行し、ファイルシステムをセットアップし、さらにマシンの永続ファイルシステムに他の変更を加える可能性があります。これとは対照的に、cloud-init はシステムの起動時にマシンの init システムの一部として実行されるため、ディスクパーティションなどへの基礎的な変更を簡単に行うことはできません。cloud-ini では、ノードの起動プロセスの実行中の起動プロセスの再設定は簡単に実行できません。

- Ignition は既存システムを変更することなく、システムを初期化することが意図されています。マシンが初期化され、インストールされたシステムからカーネルが実行された後に、OpenShift Container Platform クラスターの Machine Config Operator がその後のすべてのマシン設定を行います。

- 定義されたアクションセットを実行する代わりに、Ignition は宣言型の設定を実装します。これは新規マシンの起動前にすべてのパーティション、ファイル、サービスその他のアイテムがあることを検査します。また、新規マシンを指定された設定に一致させるために必要なファイルのディスクへのコピーなどの変更を行います。

- Ignition がマシンの設定を終了した後もカーネルは実行し続けますが、初期 RAM ディスクを破棄し、ディスクにインストールされたシステムにピボットします。システムを再起動しなくても、すべての新しいシステムサービスとその他の機能は起動します。

- Ignition は新しいマシンすべてが宣言型の設定に一致することを確認するため、部分的に設定されたマシンを含めることはできません。マシンのセットアップが失敗し、初期化プロセスが終了しない場合、Ignition は新しいマシンを起動しません。クラスターには部分的に設定されたマシンが含まれることはありません。Ignition が完了しない場合、マシンがクラスターに追加されることはありません。その場合、新しいマシンを各自で追加する必要があります。この動作は、失敗した設定タスクに依存するタスクが後で失敗するまで設定タスクに問題があることが認識されない場合などの、マシンのデバックが困難になる状況を防ぐことができます。

- マシンのセットアップの失敗を引き起こす問題が Ignition 設定にある場合、Ignition は他のマシンの設定に同じ設定を使用することを避けようとします。たとえば、失敗の原因は、同じファイルを作成しようとする親と子で設定される Ignition 設定にある場合があります。この場合、問題が解決されない限り、その Ignition 設定を使用して他のマシンをセットアップすることはできません。

- 複数の Ignition 設定ファイルがある場合、それらの設定の集合体を取得します。Ignition は宣言型であるため、これらの設定間で競合が生じると、Ignition はマシンのセットアップに失敗します。ここで、それらのファイルの情報の順序は問題にはなりません。Ignition はそれぞれの設定を最も妥当な仕方で並べ替え、実行します。たとえば、あるファイルが数レベル分深いレベルのディレクトリーを必要としており、他のファイルがそのパスにあるディレクトリーを必要とする場合、後者のファイルが先に作成されます。Ignition はすべてのファイル、ディレクトリーおよびリンクを深さに応じて作成します。

- Ignition は完全に空のハードディスクで起動できるので、cloud-init では実行できないことを行うことができます。これには、 (PXE ブートなどの機能を使用して)、システムをベアメタルでゼロからセットアップすることが含まれます。ベアメタルの場合、Ignition 設定は起動パーティションに挿入されるので、 Ignition はこれを見つけ、システムを正しく設定することができます。

5.1.4.2. Ignition の順序

OpenShift Container Platform クラスター内の RHCOS マシンの Ignition プロセスには、以下の手順が含まれます。

- マシンがその Ignition 設定ファイルを取得します。マスターマシンは Ignition 設定ファイルをブートストラップマシンから取得し、ワーカーマシンは Ignition 設定ファイルをマスターから取得します。

- Ignition はマシン上でディスクパーティション、ファイルシステム、ディレクトリーおよびリンクを作成します。Ignition は RAID アレイをサポートしますが、LVM ボリュームはサポートしません。

- Ignition は永続ファイルシステムのルートを initramfs 内の **/sysroot** ディレクトリーにマウントし、その **/sysroot** ディレクトリーで機能し始めます。

- Ignition はすべての定義されたファイルシステムを設定し、それらをランタイム時に適切にマウントされるようセットアップします。

- Ignition は **systemd** 一時ファイルを実行して、必要なファイルを **/var** ディレクトリーに設定します。

- Ignition は Ignition 設定ファイルを実行し、ユーザー、systemd ユニットファイルその他の設定ファイルをセットアップします。

- Ignition は initramfs にマウントされた永続システムのコンポーネントをすべてアンマウントします。

- Ignition は新しいマシンの init プロセスを開始し、システムの起動時に実行されるマシン上の他のすべてのサービスを開始します。

マシンはクラスターに参加できる状態になります。再起動は不要です。

5.2. IGNITION 設定ファイルの表示

ブートストラップマシンをデプロイするのに使用される Ignition 設定ファイルを表示するには、以下のコマンドを実行します。

```
$ openshift-install create ignition-configs --dir $HOME/testconfig
```

いくつかの質問に回答すると、**bootstrap.ign**、**master.ign**、**worker.ign** ファイルが入力したディレクトリーに表示されます。

bootstrap.ign ファイルの内容を確認するには、そのファイルを **jq** フィルターでそのファイルをパイプします。以下は、そのファイルの抜粋です。

```
$ cat $HOME/testconfig/bootstrap.ign | jq
\\{
  "ignition": \\{
      "config": \\{},
  "storage": \\{
      "files": [
        \\{
          "filesystem": "root",
          "path": "/etc/motd",
          "user": \\{
            "name": "root"
          },
          "append": true,
          "contents": \\{
            "source": "data:text/plain;charset=utf-
8;base64,VGhpcyBpcyB0aGUgYm9vdHN0cmFwIG5vZGU7IGl0IHdpbGwgYmUgZGVzdHJveWVkIHdo
ZW4gdGhlIG1hc3RlciBpcyBmdWxseSB1cC4KClRoZSBwcmltYXJ5IHNlcnZpY2UgaXMgImJvb3RrdWJl
LnNlcnZpY2UiLiBUbyB3YXRjaCBpdHMgc3RhdHVzLCBydW4gZS5nLgoKICBqb3VybmFsY3RsIC1iIC1m
IC11IGJvb3RrdWJlLnNlcnZpY2UK",
```

bootstrap.ign ファイルに一覧表示されたファイルの内容をデコードするには、そのファイルの内容を表す base64 でエンコードされたデータ文字列を **base64 -d** コマンドに渡します。以下に示すのは、上記の出力からブートストラップマシンに追加された **/etc/motd** ファイルの内容の使用例です。

```
$ echo
VGhpcyBpcyB0aGUgYm9vdHN0cmFwIG5vZGU7IGl0IHdpbGwgYmUgZGVzdHJveWVkIHdoZW4gdG
hlIG1hc3RlciBpcyBmdWxseSB1cC4KClRoZSBwcmltYXJ5IHNlcnZpY2UgaXMgImJvb3RrdWJlLnNlcnZp
Y2UiLiBUbyB3YXRjaCBpdHMgc3RhdHVzLCBydW4gZS5nLgoKICBqb3VybmFsY3RsIC1iIC1mIC11IGJ
vb3RrdWJlLnNlcnZpY2UK | base64 --decode
```

出力例

```
This is the bootstrap machine; it will be destroyed when the master is fully up.

The primary service is "bootkube.service". To watch its status, run, e.g.:
```

```
journalctl -b -f -u bootkube.service
```

これらのコマンドを **master.ign** と **worker.ign** ファイル上で繰り返し実行し、マシンタイプごとの Ignition 設定ファイルのソースを参照します。 ブートストラップマシンから Ignition 設定を取得する方法を特定する、**worker.ign** についての以下のような行が表示されるはずです。

```
"source": "https://api.myign.develcluster.example.com:22623/config/worker",
```

bootstrap.ign ファイルについて、以下のいくつかの点に留意してください

- フォーマット: ファイルのフォーマットは Ignition config spec に定義されています。同じフォーマットのファイルが後に MCO によって使用され、マシンの設定に変更がマージされます。
- コンテンツ: ブートストラップマシンは他のマシンの Ignition 設定を提供するため、マスターマシンとワーカーマシンの両方の Ignition 設定情報は、ブートストラップの設定情報と共に **bootstrap.ign** に保管されます。
- サイズ: 各種タイプのリソースへのパスを含むファイルのサイズは、1,300 行を超える長さです。
- マシンにコピーされる各ファイルの内容は実際にデータ URL にエンコードされます。この場合、内容は少し読み取りにくくなる傾向があります (前述の jq や base64 コマンドを使用すると内容がより読みやすくなります)。
- 設定: Ignition 設定ファイルのそれぞれのセクションは、一般的には既存ファイルを修正するコマンドではなく、マシンのファイルシステムに単にドロップされるファイルを含むことが想定されています。たとえば、そのサービスを設定する NFS 上のセクションを設定するのではなく、単に NFS 設定ファイルを追加します。 これはその後のシステムの起動時に init プロセスによって開始されます。
- ユーザー: core という名前のユーザーが作成され、SSH キーがそのユーザーに割り当てられます。これにより、そのユーザー名と認証情報を使用してクラスターにログインすることができます。
- ストレージ: ストレージセクションは、各マシンに追加されるファイルを特定します。これらのファイルには、(実際のクラスターがコンテナーイメージレジストリーからプルする必要のある認証情報を提供する)**/root/.docker/config.json** と、クラスターを設定するのに使用される **/opt/openshift/manifests** 内のマニフェストファイルのセットがあります。
- systemd: systemd セクションは、systemd ユニットファイルを作成するコンテンツを保持します。これらのファイルは、起動時にサービスを開始するために、また実行システムでサービスを管理するために使用されます。
- プリミティブ: Ignition は他のツールがビルドに使用できる低レベルのプリミティブも公開します。

5.3. インストール後の IGNITION 設定の変更

Machine 設定プールはノードのクラスターおよびそれらの対応するマシン設定を管理します。マシン設定にはクラスターの設定情報が含まれます。既知のすべてのマシン設定プールを一覧表示するには、以下を実行します。

```
$ oc get machineconfigpools
```

出力例

```
NAME   CONFIG                             UPDATED UPDATING DEGRADED
master master-1638c1aea398413bb918e76632f20799 False   False    False
worker worker-2feef4f8288936489a5a832ca8efe953 False   False    False
```

すべてのマシン設定を一覧表示するには、以下を実行します。

```
$ oc get machineconfig
```

出力例

```
NAME                                   GENERATEDBYCONTROLLER   IGNITIONVERSION   CREATED
OSIMAGEURL

00-master                              4.0.0-0.150.0.0-dirty   2.2.0             16m
00-master-ssh                            4.0.0-0.150.0.0-dirty                   16m
00-worker                              4.0.0-0.150.0.0-dirty   2.2.0             16m
00-worker-ssh                            4.0.0-0.150.0.0-dirty                   16m
01-master-kubelet                          4.0.0-0.150.0.0-dirty   2.2.0             16m
01-worker-kubelet                          4.0.0-0.150.0.0-dirty   2.2.0             16m
master-1638c1aea398413bb918e76632f20799   4.0.0-0.150.0.0-dirty   2.2.0             16m
worker-2feef4f8288936489a5a832ca8efe953   4.0.0-0.150.0.0-dirty   2.2.0             16m
```

Machine Config Operator が Machineconfig を適用するときの動作は Ignition とは若干異なります。マシン設定は (00* から 99* までの) 順序で読み取られます。マシン設定内のラベルは、それぞれのノードのタイプ (マスターまたはワーカー) を特定します。同じファイルが複数のマシン設定ファイルに表示される場合、最後のファイルが有効になります。たとえば、99* ファイルに出現するファイルは、00* ファイルに出現する同一のファイルを置き換えます。入力された **MachineConfig** オブジェクトはレンダリングされた **MachineConfig** オブジェクトに結合されます。 これは Operator のターゲットとして使用され、マシン設定プールで確認できる値です。

マシン設定から管理されているファイルを表示するには、特定の **MachineConfig** オブジェクト内で "Path:" を検索します。以下に例を示します。

```
$ oc describe machineconfigs 01-worker-container-runtime | grep Path:
```

出力例

```
        Path:           /etc/containers/registries.conf
        Path:           /etc/containers/storage.conf
        Path:           /etc/crio/crio.conf
```

マシン設定ファイルには (10-worker-container-runtime などの) より新しい名前を付けてください。各ファイルの内容については、URL 形式のデータであることに留意してください。次に、新規マシン設定をクラスターに適用します。

第6章 CI/CD の方法論および実践

継続的インテグレーション/継続的デリバリー (CI/CD) の方法論を採用すると、統合およびテストのフェーズからデリバリーおよびデプロイメントに至るまでのアプリケーション開発の複数の段階に自動化を導入し、アプリケーションをお客様に定期的に配信できます。CI/CD プロセスは、CI/CD パイプラインと呼ばれることがよくあります。 CI/CD に関連する主な概念には、継続的インテグレーション、継続的デリバリー、および継続的デプロイメントが含まれます。

6.1. クラスター管理およびアプリケーション設定管理向けの CI/CD

継続的インテグレーション は、開発者向けの自動化プロセスです。アプリケーションのコード変更は定期的にビルドされ、テストされ、共有リポジトリーにマージされます。

継続的デリバリー および **継続的デリバリー** は、相互に密接に関連する交換可能な概念であり、パイプラインの自動化に言及する概念です。継続的デリバリーでは、自動化により、開発者によるアプリケーションの変更をテストし、リポジトリーに送信し、オペレーションチームがそれらを実稼働環境にデプロイできるようにします。継続的デプロイメントは、リポジトリーから実稼働環境までに発生する変更のリリースを有効にします。継続的デプロイメントはアプリケーションの配信を迅速化し、オペレーションチームのオーバーロードを防ぎます。

6.2. GITOPS の方法論および実戦

GitOps は、Git プル要求を使用してインフラストラクチャーおよびアプリケーションの設定を管理する一連の手法です。GitOps の Git リポジトリーは、システムおよびアプリケーション設定の信頼できる唯一のソースです。リポジトリーにはシステムの全体の状態が含まれ、システム状態の変更の追跡情報は表示され、監査可能になります。GitOps を使用して DevOps の方法論を実装できます。

GitOps ツールを使用し、OpenShift Container Platform クラスターおよびアプリケーションの管理および再作成のための反復可能かつ予測可能なプロセスを作成できます。GitOps を使用することで、インフラストラクチャーおよびアプリケーション設定のスプロールの問題に対処できます。これは、インフラストラクチャーとアプリケーションの定義をコードとして定義することで、インフラストラクチャーおよびアプリケーション設定の変更の伝播を単純化します。 クラスター設定ファイルに GitOps を実装すると、自動インストールが容易になり、クラスターの自動化されたカスタマイズの設定が可能になります。Git リポジトリーでのソフトウェアの開発およびメンテナーンスのコアとなる原則を、クラスターおよびアプリケーション設定ファイルの作成および管理に適用できます。

OpenShift Container Platform を使用してクラスター設定およびコンテナー開発プロセスの両方を自動化することにより、GitOps 手法を導入する場面およびそのタイミングを選択することができます。GitOps ストラテジーと実行計画と共に CI パイプラインを使用することが理想的です。OpenShift Container Platform は、この方法論をビジネスプラクティスや各種パイプラインに統合するタイミングとその方法を選択する面での柔軟性を提供します。

GitOps 統合により、OpenShift Container Platform クラスター設定を宣言的に設定し、保存することができます。

GitOps はクラスターを宣言的に設定し、クラスター設定の状態を Git に保存することができるため、OpenShift Container Platform と効果的に機能します。詳細は、利用可能なクラスターのカスタマイズ を参照してください。

6.2.1. 単一クラスターおよびマルチクラスター管理向けの GitOps

独立した OpenShift Container Platform クラスターまたは連携型の OpenShift Container Platform クラスターのいずれが必要となる場合でも、GitOps ストラテジーを使用して以下のタスクを管理できます。

- クラスターに設定、モニターリングまたはストレージについての同様の状態の設定があることを確認します。
- クラスターを既知の状態から回復するか、または再作成します。
- 既知の状態でクラスターを作成します。
- 複数の OpenShift Container Platform クラスターに対する設定変更を適用するか、または元に戻します。
- テンプレート化された設定を複数の異なる環境に関連付けます。

6.2.2. アプリケーション設定管理向けの GitOps

GitOps 手法を使用してアプリケーション設定を管理することもできます。これにより、異なる環境 (開発、ステージ、実稼働環境など) の異なるクラスターにアプリケーションをデプロイする場合に、アプリケーションの一貫性が確保されます。また、GitOps でアプリケーション設定を管理することは、クラウドまたはオンプレミスのいずれであれ、複数のクラスターにアプリケーションをデプロイする必要がある場合に、可用性やスケーラビリティーを確保する上で有益です。

GitOps ストラテジーを使用して以下を行うことができます。

- ステージから実稼働環境へと、クラスター全体でアプリケーションをプロモートします。
- 複数の OpenShift Container Platform クラスターへのアプリケーションの変更を適用するか、または元に戻します。

6.2.3. GitOps テクノロジープロバイダーおよびインテグレーター

複数のコミュニティーオファリングおよびサードパーティーベンダーが OpenShift Container Platform との高度な統合機能を提供しています。

以下のコミュニティーパートナーおよびサードパーティーインテグレーターを使用して GitOps を OpenShift Container Platform に統合できます。

- ArgoCD

第7章 OPENSHIFT CONTAINER PLATFORM での ARGOCD の使用

7.1. ARGOCD の機能

ArgoCD は、GitOps を使用してクラスターリソースを維持する宣言型の継続的デリバリーツールです。ArgoCD は、Git リポジトリーで定義されるアプリケーション定義および設定を継続的に監視し、それらの設定の指定された状態をクラスターのライブ状態と比較するコントローラーとして実装されます。Git リポジトリーで指定された状態から逸脱した設定は OutOfSync として分類されます。ArgoCD はこれらの違いを報告し、管理者が定義した状態に合わせて設定を自動または手動で再同期できるようにします。

ArgoCD を使用すると、OpenShift Container Platform クラスターを設定するために使用されるリソースなどのグローバルカスタムリソースを配信できます。

7.2. サポートについての説明

Red Hat では、このツールのサポートを提供していません。ArgoCD のサポートを取得するには、ArgoCD ドキュメントの Support を参照してください。

7.3. ARGOCD ドキュメント

ArgoCD の使用についての詳細は、ArgoCD ドキュメント を参照してください。

第8章 受付プラグイン

8.1. 受付プラグインについて

受付プラグインは、OpenShift Container Platform 4.5 の機能の調整に役立ちます。受付プラグインは、マスター API への要求をインターセプトしてリソース要求の検証を行い、要求の認証および認可後にポリシーが順守されるようにします。たとえば、これらは一般的にセキュリティーポリシー、リソースの制限、設定要件の適用に使用されます。

受付プラグインは受付チェーン (admission chain) として順番に実行されます。シーケンス内の受付プラグインが要求を拒否すると、チェーン全体が中止され、エラーが返されます。

OpenShift Container Platform には、各リソースタイプについて有効にされている受付プラグインのデフォルトセットがあります。それらはマスターが適切に機能するために必要です。受付プラグインは、それらが対応していないリソースを無視します。

デフォルト以外にも、受付チェーンは、カスタム Webhook サーバーを呼び出す Webhook 受付プラグインを介して動的に拡張できます。Webhook 受付プラグインには、変更用の受付プラグインと検証用の受付プラグインの 2 種類の Webhook 受付プラグインがあります。変更用の受付プラグインが最初に実行され、リソースの変更および要求の検証の両方が可能です。検証用の受付プラグインは要求を検証し、変更用の受付プラグインによってトリガーされた変更も検証できるように変更用の受付プラグインの後に実行されます。

変更用の受付プラグインを使用して Webhook サーバーを呼び出すと、ターゲットオブジェクトに関連するリソースに影響を与える可能性があります。このような場合に、最終結果が想定通りであることを検証するためにいくつかの手順を実行する必要があります。

警告

動的な受付クラスターはコントロールプレーンの操作に影響するため、これは注意して使用する必要があります。OpenShift Container Platform 4.5 の Webhook 受付プラグインを使用して Webhook サーバーを呼び出す場合は、変更による影響についての情報を十分に確認し、それらの影響の有無についてテストするようにしてください。要求が受付チェーン全体を通過しない場合は、リソースを変更前の元の状態に復元する手順を追加します。

8.2. デフォルトの受付プラグイン

デフォルトの受付プラグインのセットは OpenShift Container Platform 4.5 で有効にされています。これらのデフォルトプラグインは、Ingress ポリシー、クラスターリソース制限の上書き、クォータポリシーなどの基本的なコントロールプレーンの機能に貢献するものです。

8.3. WEBHOOK 受付プラグイン

OpenShift Container Platform のデフォルト受付プラグインのほかに、受付チェーンの機能を拡張するために Webhook サーバーを呼び出す Webhook 受付プラグインを使用して動的な受付を実装できます。Webhook サーバーは、定義されたエンドポイントにて HTTP で呼び出されます。

OpenShift Container Platform には、2 種類の Webhook 受付プラグインがあります。

- 受付プロセスで、**変更用の受付プラグイン** は、アフィニティーラベルの挿入などのタスクを実行できます。
- 受付プロセスの最後に、**検証用の受付プラグイン** を使用して、アフィニティーラベルが予想通りにされているかどうかの確認など、オブジェクトが適切に設定されていることを確認できます。検証にパスすると、OpenShift Container Platform はオブジェクトを設定済みとしてスケジュールします。

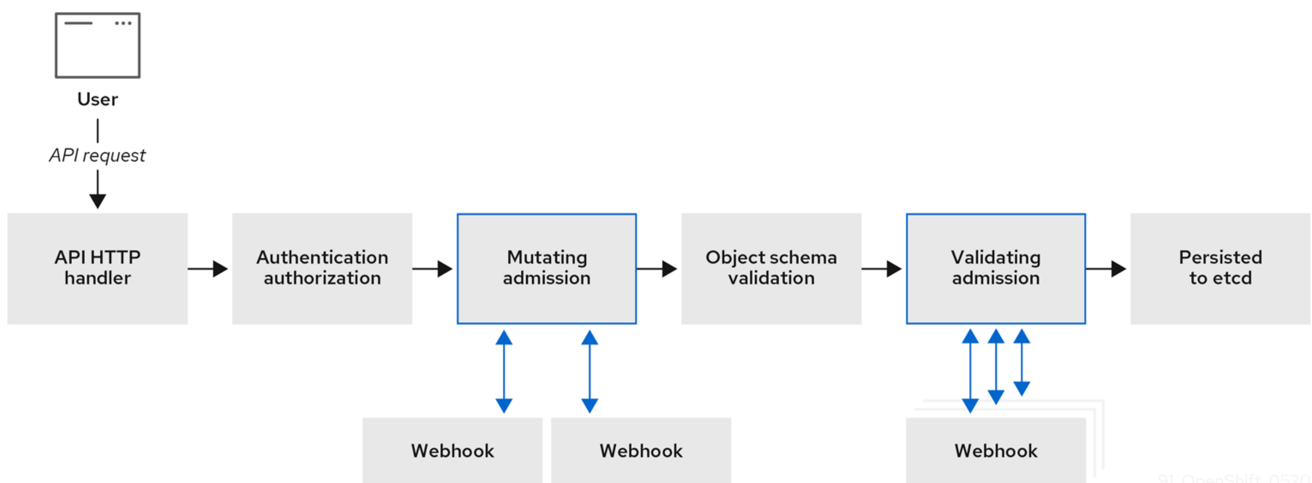
API 要求が送信されると、変更用または検証用の受付コントローラーは設定内の外部 Webhook の一覧を使用し、それらを並行して呼び出します。

- Webhook のすべてが要求を承認する場合、受付チェーンは継続します。
- Webhook のいずれかが要求を拒否する場合、受付要求は拒否され、これは、初回の拒否理由に基づいて実行されます。
- 複数の Webhook が受付要求を拒否する場合、初回の拒否理由のみがユーザーに返されます。
- Webhook の呼び出し時にエラーが発生した場合、要求は拒否されるか、または Webhook はエラーポリシーセットに応じて無視されます。エラーポリシーが **Ignore** に設定されている場合、要求は失敗すると無条件で受け入れられます。ポリシーが **Fail** に設定される場合、失敗した要求は拒否されます。**Ignore** を使用すると、すべてのクライアントの予測できない動作が生じる可能性があります。

Webhook の受付プラグインと Webhook サーバー間の通信は TLS を使用する必要があります。CA 証明書を生成し、その証明書を使用して Webhook 受付サーバーで使用されるサーバー証明書に署名します。PEM 形式の CA 証明書は、サービス提供証明書のシークレットなどのメカニズムを使用して Webhook 受付プラグインに提供されます。

以下の図は、複数の Webhook サーバーが呼び出される連続した受付チェーンのプロセスを示しています。

図8.1 変更用および検証用の受付プラグインを含む API 受付チェーン



Webhook 受付プラグインのユースケースの例として使用できるケースでは、すべての Pod に共通のラベルのセットがなければなりません。この例では、変更用の受付プラグインはラベルを挿入でき、検証用の受付プラグインではラベルが予想通りであることを確認できます。OpenShift Container Platform は引き続いて必要なラベルが含まれる Pod をスケジュールし、それらのラベルが含まれない Pod を拒否します。

一般的な Webhook 受付プラグインのユースケースとして、以下が含まれます。

- namespace の予約。
- SR-IOV ネットワークデバイスプラグインによって管理されるカスタムネットワークリソースの制限。
- テイントでノードにスケジュールする必要のある Pod を特定できるようにする容認の定義。
- Pod 優先順位クラスの検証。

8.4. WEBHOOK 受付プラグインのタイプ

クラスター管理者は、API サーバーの受付チェーンで変更用の受付プラグインまたは検証用の受付プラグインを使用して Webhook サーバーを呼び出すことができます。

8.4.1. 受付プラグインの変更

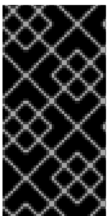
変更用の受付プラグインは、受付プロセスの変更フェーズで起動します。これにより、リソースコンテンツが永続化する前にそれらを変更することができます。変更用の受付プラグインで呼び出し可能な Webhook の一例として、Pod ノードセレクター機能があります。 この機能は namespace でアノテーションを使用してラベルセレクターを検索し、これを Pod 仕様に追加します。

変更用の受付プラグインの設定例:

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration (1)
metadata:
  name: <webhook_name> (2)
webhooks:
- name: <webhook_name> (3)
  clientConfig: (4)
    service:
      namespace: default (5)
      name: kubernetes (6)
     path: <webhook_url> (7)
    caBundle: <ca_signing_certificate> (8)
  rules: (9)
  - operations: (10)
    - <operation>
    apiGroups:
    - ""
    apiVersions:
    - "*"
    resources:
    - <resource>
  failurePolicy: <policy> (11)
  sideEffects: None
```

1. 変更用の受付プラグイン設定を指定します。
2. **MutatingWebhookConfiguration** オブジェクトの名前。**<webhook_name>** を適切な値に置き換えます。
3. 呼び出す Webhook の名前です。**<webhook_name>** を適切な値に置き換えます。

4. Webhook サーバーに接続し、これを信頼し、データをこれに送信する方法についての情報です。
5. フロントエンドサービスが作成される namespace です。
6. フロントエンドサービスの名前です。
7. 受付要求に使用される Webhook URL です。**<webhook_url>** を適切な値に置き換えます。
8. Webhook サーバーで使用されるサーバー証明書に署名する PEM でエンコーディングされた CA 証明書です。**<ca_signing_certificate>** を base64 形式の適切な証明書に置き換えます。
9. API サーバーがこの Webhook 受付プラグインを使用する必要があるタイミングを定義するルールです。
10. API サーバーをトリガーしてこの Webhook 受付プラグインを呼び出す 1 つ以上の操作です。使用できる値は、**create**、**update**、**delete**、または **connect** です。**<operation>** および **<resource>** を適切な値に置き換えます。
11. Webhook サーバーが利用できない場合にポリシーを実行する方法を指定します。**<policy>** を **Ignore** (失敗した場合に要求を無条件で受け入れる) または **Fail** (失敗した要求を拒否する) のいずれかに置き換えます。**Ignore** を使用すると、すべてのクライアントの予測できない動作が生じる可能性があります。



重要

OpenShift Container Platform 4.5 では、ユーザーによって作成されるオブジェクト、または変更用の受付プラグインを使用するコントロールループは、初回の要求で設定される値が上書きされる値が上書きされる場合などに予期しない結果を返す場合があるため、推奨されていません。

8.4.2. 受付プラグインの検証

検証用の受付 Webhook は受付プロセスの検証フェーズで起動します。このフェーズでは、特定 API リソースの変更がない項目の実施を可能にし、リソースが再び変更されないようにすることができます。Pod ノードセレクターは、すべての **nodeSelector** フィールドが namespace のノードセレクターの制限の制約を受けるようにするために、検証用の受付プラグインによって呼び出される Webhook の一例です。

検証用の受付 Webhook 設定の例:

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration (1)
metadata:
  name: <webhook_name> (2)
webhooks:
- name: <webhook_name> (3)
  clientConfig: (4)
    service:
      namespace: default (5)
      name: kubernetes (6)
     path: <webhook_url> (7)
    caBundle: <ca_signing_certificate> (8)
  rules: (9)
```

```
- operations: (10)
  - <operation>
  apiGroups:
  - ""
  apiVersions:
  - "*"
  resources:
  - <resource>
failurePolicy: <policy> (11)
sideEffects: Unknown
```

1. 検証用の受付 Webhook 設定を指定します。
2. **ValidatingWebhookConfiguration** オブジェクトの名前。**<webhook_name>** を適切な値に置き換えます。
3. 呼び出す Webhook の名前です。**<webhook_name>** を適切な値に置き換えます。
4. Webhook サーバーに接続し、これを信頼し、データをこれに送信する方法についての情報です。
5. フロントエンドサービスが作成される namespace です。
6. フロントエンドサービスの名前です。
7. 受付要求に使用される Webhook URL です。**<webhook_url>** を適切な値に置き換えます。
8. Webhook サーバーで使用されるサーバー証明書に署名する PEM でエンコーディングされた CA 証明書です。**<ca_signing_certificate>** を base64 形式の適切な証明書に置き換えます。
9. API サーバーがこの Webhook 受付プラグインを使用する必要があるタイミングを定義するルールです。
10. API サーバーをトリガーしてこの Webhook 受付プラグインを呼び出す 1 つ以上の操作です。使用できる値は、**create**、**update**、**delete**、または **connect** です。**<operation>** および **<resource>** を適切な値に置き換えます。
11. Webhook サーバーが利用できない場合にポリシーを実行する方法を指定します。**<policy>** を **Ignore** (失敗した場合に要求を無条件で受け入れる) または **Fail** (失敗した要求を拒否する) のいずれかに置き換えます。**Ignore** を使用すると、すべてのクライアントの予測できない動作が生じる可能性があります。

8.5. 動的受付の設定

この手順では、動的受付を設定するための手順の概要を説明します。受付チェーンの機能は、Webhook サーバーを呼び出すように Webhook 受付プラグインを設定することで拡張されます。

Webhook サーバーは集約された API サーバーとしても設定されます。これにより、他の OpenShift Container Platform コンポーネントは内部認証情報を使用して Webhook と通信でき、**oc** コマンドを使用したテストを容易にします。さらに、これによりロールベースのアクセス制御 (RBAC) が Webhook に対して可能となり、他の API サーバーからのトークン情報が Webhook に開示されないようになります。

前提条件

- クラスター管理者のアクセスを持つ OpenShift Container Platform アカウント。

- OpenShift Container Platform CLI(**oc**) がインストールされていること。
- 公開されている Webhook サーバーコンテナーイメージ。

手順

1. Webhook サーバーコンテナーイメージをビルドし、イメージレジストリーを使用してこれをクラスターで使用できるようにします。
2. ローカル CA キーおよび証明書を作成し、それらを使用して Webhook サーバーの証明書署名要求 (CSR) に署名します。
3. Webhook リソースの新規プロジェクトを作成します。

```
$ oc new-project my-webhook-namespace 1
```

1 Webhook サーバーで特定の名前が使用される可能性があることに注意してください。

4. **rbac.yaml** というファイルで集約された API サービスの RBAC ルールを定義します。

```
apiVersion: v1
kind: List
items:

- apiVersion: rbac.authorization.k8s.io/v1 1
  kind: ClusterRoleBinding
  metadata:
    name: auth-delegator-my-webhook-namespace
  roleRef:
    kind: ClusterRole
    apiGroup: rbac.authorization.k8s.io
    name: system:auth-delegator
  subjects:
  - kind: ServiceAccount
    namespace: my-webhook-namespace
    name: server

- apiVersion: rbac.authorization.k8s.io/v1 2
  kind: ClusterRole
  metadata:
    annotations:
    name: system:openshift:online:my-webhook-server
  rules:
  - apiGroups:
    - online.openshift.io
    resources:
    - namespacereservations 3
    verbs:
    - get
    - list
    - watch

- apiVersion: rbac.authorization.k8s.io/v1 4
  kind: ClusterRole
```

```
  metadata:
    name: system:openshift:online:my-webhook-requester
  rules:
  - apiGroups:
    - admission.online.openshift.io
    resources:
    - namespacereservations (5)
    verbs:
    - create

- apiVersion: rbac.authorization.k8s.io/v1 (6)
  kind: ClusterRoleBinding
  metadata:
    name: my-webhook-server-my-webhook-namespace
  roleRef:
    kind: ClusterRole
    apiGroup: rbac.authorization.k8s.io
    name: system:openshift:online:my-webhook-server
  subjects:
  - kind: ServiceAccount
    namespace: my-webhook-namespace
    name: server

- apiVersion: rbac.authorization.k8s.io/v1 (7)
  kind: RoleBinding
  metadata:
    namespace: kube-system
    name: extension-server-authentication-reader-my-webhook-namespace
  roleRef:
    kind: Role
    apiGroup: rbac.authorization.k8s.io
    name: extension-apiserver-authentication-reader
  subjects:
  - kind: ServiceAccount
    namespace: my-webhook-namespace
    name: server

- apiVersion: rbac.authorization.k8s.io/v1 (8)
  kind: ClusterRole
  metadata:
    name: my-cluster-role
  rules:
  - apiGroups:
    - admissionregistration.k8s.io
    resources:
    - validatingwebhookconfigurations
    - mutatingwebhookconfigurations
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - ""
    resources:
    - namespaces
    verbs:
```

```
    - get
    - list
    - watch

- apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: my-cluster-role
  roleRef:
    kind: ClusterRole
    apiGroup: rbac.authorization.k8s.io
    name: my-cluster-role
  subjects:
  - kind: ServiceAccount
    namespace: my-webhook-namespace
    name: server
```

1. 認証および認可を Webhook サーバー API に委任します。
2. Webhook サーバーがクラスターリソースにアクセスできるようにします。
3. リソースを参照します。この例では、**namespacereservations** リソースを参照します。
4. 集約された API サーバーが受付レビューを作成できるようにします。
5. リソースを参照します。この例では、**namespacereservations** リソースを参照します。
6. Webhook サーバーがクラスターリソースにアクセスできるようにします。
7. 認証を終了するために設定を読み取るためのロールバインディングです。
8. 集約された API サーバーのデフォルトのクラスターロールおよびクラスターロールバインディングです。

5. これらの RBAC ルールをクラスターに適用します。

```
$ oc auth reconcile -f rbac.yaml
```

6. namespace に Webhook をデーモンセットサーバーとして デプロイするために使用される **webhook-daemonset.yaml** という YAML ファイルを作成します。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  namespace: my-webhook-namespace
  name: server
  labels:
    server: "true"
spec:
  selector:
    matchLabels:
      server: "true"
  template:
    metadata:
      name: server
```

```
    labels:
      server: "true"
  spec:
    serviceAccountName: server
    containers:
    - name: my-webhook-container (1)
      image: <image_registry_username>/<image_path>:<tag> (2)
      imagePullPolicy: IfNotPresent
      command:
      - <container_commands> (3)
      ports:
      - containerPort: 8443 (4)
      volumeMounts:
      - mountPath: /var/serving-cert
        name: serving-cert
      readinessProbe:
        httpGet:
          path: /healthz
          port: 8443 (5)
          scheme: HTTPS
    volumes:
    - name: serving-cert
      secret:
        defaultMode: 420
        secretName: server-serving-cert
```

1. Webhook サーバーで特定のコンテナー名が使用される可能性があることに注意してください。
2. Webhook サーバーコンテナーイメージを参照します。**<image_registry_username>/<image_path>:<tag>** を適切な値に置き換えます。
3. Webhook コンテナー run コマンドを指定します。**<container_commands>** を適切な値に置き換えます。
4. Pod 内のターゲットポートを定義します。この例では、ポート 8443 を使用します。
5. Readiness プローブによって使用されるポートを指定します。この例では、ポート 8443 を使用します。

7. デーモンセットをデプロイします。

```
$ oc apply -f webhook-daemonset.yaml
```

8. サービス提供証明書の署名側のシークレットを **webhook-secret.yaml** という YAML ファイル内に定義します。

```
apiVersion: v1
kind: Secret
metadata:
  namespace: my-webhook-namespace
  name: server-serving-cert
type: kubernetes.io/tls
```

```
data:
  tls.crt: <server_certificate> (1)
  tls.key: <server_key> (2)
```

1. 署名された Webhook サーバー証明書を参照します。**<server_certificate>** を base64 形式の適切な証明書に置き換えます。

2. 署名された Webhook サーバーキーを参照します。**<server_key>** を base64 形式の適切なキーに置き換えます。

9. シークレットを作成します。

```
$ oc apply -f webhook-secret.yaml
```

10. サービスアカウントおよびサービスを、**webhook-service.yaml** という YAML ファイル内に定義します。

```
apiVersion: v1
kind: List
items:

- apiVersion: v1
  kind: ServiceAccount
  metadata:
    namespace: my-webhook-namespace
    name: server

- apiVersion: v1
  kind: Service
  metadata:
    namespace: my-webhook-namespace
    name: server
    annotations:
      service.alpha.openshift.io/serving-cert-secret-name: server-serving-cert
  spec:
    selector:
      server: "true"
    ports:
    - port: 443 (1)
      targetPort: 8443 (2)
```

1. サービスがリッスンするポートを定義します。この例では、ポート 443 を使用します。

2. サービスが接続を転送する Pod 内のターゲットポートを定義します。この例では、ポート 8443 を使用します。

11. クラスターに Webhook サーバーを公開します。

```
$ oc apply -f webhook-service.yaml
```

12. Webhook サーバーのカスタムリソース定義を **webhook-crd.yaml** という名前のファイルに定義します。

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: namespacereservations.online.openshift.io (1)
spec:
  group: online.openshift.io (2)
  version: v1alpha1 (3)
  scope: Cluster (4)
  names:
    plural: namespacereservations (5)
    singular: namespacereservation (6)
    kind: NamespaceReservation (7)
```

1. **CustomResourceDefinition spec** 値を反映させ、 **<plural>.<group>** 形式を使用します。この例では、**namespacereservations** リソースを使用します。
2. REST API グループ名です。
3. REST API バージョン名です。
4. 許可される値は **Namespaced** または **Cluster** です。
5. URL に含まれる複数形の名前です。
6. **oc** 出力に表示されるエイリアスです。
7. リソースマニフェストの参照です。

13. カスタムリソース定義を適用します。

```
$ oc apply -f webhook-crd.yaml
```

14. Webhook サーバーも、**webhook-api-service.yaml** というファイル内に集約された API サーバーとして設定します。

```
apiVersion: apiregistration.k8s.io/v1beta1
kind: APIService
metadata:
  name: v1beta1.admission.online.openshift.io
spec:
  caBundle: <ca_signing_certificate> (1)
  group: admission.online.openshift.io
  groupPriorityMinimum: 1000
  versionPriority: 15
  service:
    name: server
    namespace: my-webhook-namespace
  version: v1beta1
```

1. Webhook サーバーで使用されるサーバー証明書に署名する PEM でエンコーディングされた CA 証明書です。**<ca_signing_certificate>** を base64 形式の適切な証明書に置き換えます。

15. 集約された API サービスをデプロイします。

```
$ oc apply -f webhook-api-service.yaml
```

16. Webhook 受付プラグイン設定を **webhook-config.yaml** というファイル内に定義します。以下の例では、検証用の受付プラグインを使用します。

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
metadata:
  name: namespacereservations.admission.online.openshift.io (1)
webhooks:
- name: namespacereservations.admission.online.openshift.io (2)
  clientConfig:
    service: (3)
      namespace: default
      name: kubernetes
      path: /apis/admission.online.openshift.io/v1beta1/namespacereservations (4)
    caBundle: <ca_signing_certificate> (5)
  rules:
  - operations:
    - CREATE
    apiGroups:
    - project.openshift.io
    apiVersions:
    - "*"
    resources:
    - projectrequests
  - operations:
    - CREATE
    apiGroups:
    - ""
    apiVersions:
    - "*"
    resources:
    - namespaces
  failurePolicy: Fail
```

1. **ValidatingWebhookConfiguration** オブジェクトの名前。この例では、**namespacereservations** リソースを使用します。
2. 呼び出す Webhook の名前です。この例では、**namespacereservations** リソースを使用します。
3. 集約された API を使用して Webhook サーバーへのアクセスを有効にします。
4. 受付要求に使用される Webhook URL です。この例では、**namespacereservation** リソースを使用します。
5. Webhook サーバーで使用されるサーバー証明書に署名する PEM でエンコーディングされた CA 証明書です。**<ca_signing_certificate>** を base64 形式の適切な証明書に置き換えます。

17. Webhook をデプロイします。

```
$ oc apply -f webhook-config.yaml
```

18. Webhook が想定通りに機能していることを確認します。たとえば、特定の namespace を予約するように動的受付を設定している場合は、これらの namespace の作成要求が拒否され、予約されていない namespace の作成要求が正常に実行されることを確認します。

8.6. 関連情報

- Limiting custom network resources managed by the SR-IOV network device plug-in
- Defining tolerations that enable taints to qualify which pods should be scheduled on a node
- Pod priority class validation