OpenShift Container Platform 3.10

アーキテクチャー

OpenShift Container Platform 3.10 アーキテクチャー情報

Last Updated: 2019-07-05

OpenShift Container Platform 3.10 アーキテクチャー

OpenShift Container Platform 3.10 アーキテクチャー情報

法律上の通知

概要

インフラストラクチャーおよびコアコンポーネントを含む OpenShift Container Platform 3.10 のアーキテクチャーについて説明します。これらのトピックでは、認証、ネットワークおよびソースコード管理についても扱います。

目次

第1章 概要

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

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

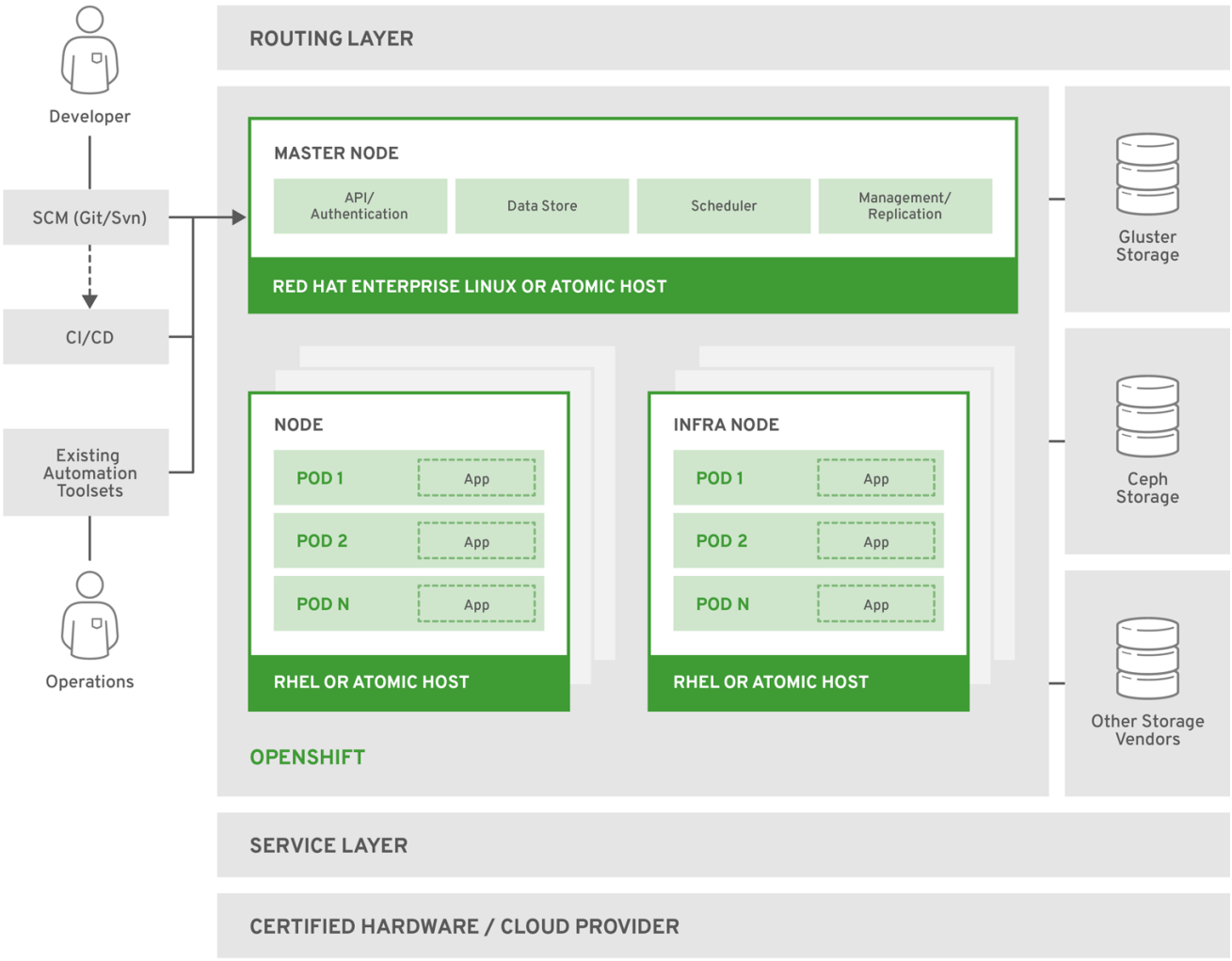
1.1. レイヤーとは

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

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

- 開発者向けのソースコードの管理、ビルド、およびデプロイメント。
- システム全体で移行するイメージの大規模な管理およびプロモート
- 大規模なアプリケーション管理
- 大規模な開発者組織を編成するためのチームおよびユーザー追跡
- クラスターをサポートするネットワークインフラストラクチャー

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



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

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

- 「REST API」: コアオブジェクトをそれぞれ公開します。
- これらの API を読み取るコントローラーは変更を別のオブジェクトに適用し、ステータスを報告し、オブジェクトに再び書き込みます。

ユーザーは REST API を呼び出して、システムの状態を変更します。コントローラーは REST API を使用してユーザーが希望する状態を読み取ってから、システムの別の部分を同期しようとします。たとえば、ユーザーが「ビルド」を要求する場合には、コントローラーは「ビルド」オブジェクトを作成します。次に、ビルドコントローラーは新規ビルドが作成されていることを確認し、そのビルドを実行するためにクラスターでプロセスを実行します。ビルドが完了すると、コントローラーは REST API でビルドオブジェクトを更新して、ユーザーはビルドが完了したことを確認できます。

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

システム管理の視点では、これは API を使用して繰り返されるスケジュールで共通の管理アクションについてのスクリプトを作成できることを意味しています。これらのスクリプトは変更を確認し、アクションを実行するコントローラーでもあります。OpenShift Container Platform でこの方法でクラスターをカスタマイズする機能をファーストクラスの動作として使用できます。

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

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

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

開発者 (システムのクライアント) は通常、ほとんどの通信に対して、「クライアントプログラム」(**oc** など) からか、またはブラウザーで「Web コンソール」に対して REST API を呼び出して、OAuth ベアラートークンを使用します。インフラストラクチャーコンポーネント (ノードなど) は、それらの ID が含まれる、システム生成のクライアント証明書を使用します。コンテナーで実行されるインフラストラクチャーコンポーネントは、「サービスアカウント」に関連付けられるトークンを使用して API に接続します。

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

クラスターで実行されるすべてのコンテナーはサービスアカウントに関連付けられるため、「シークレット」をサービスアカウントに関連付けて、コンテナーに自動的に配信することもできます。これにより、インフラストラクチャーはイメージ、ビルドおよびデプロイメントコンポーネントのプルおよびプッシュを行うためにシークレットを管理でき、アプリケーションコードでシークレットを簡単に利用することもできます。

1.3.1. TLS サポート

REST API とのすべての通信チャネル、および etcd および API サーバーなどの「マスターコンポーネント」間の通信のセキュリティーは TLS で保護されます。TLS は、X.509 サーバー証明書および公開鍵インフラストラクチャーを使用して、強力な暗号、データの整合性、およびサーバーの認証を提供します。デフォルトで、新規の内部 PKI は OpenShift Container Platform のデプロイメントごとに作成されます。内部 PKI は 2048 ビット RSA キーおよび SHA-256 署名を使用します。パブリックホストの「カスタム証明書」もサポートされます。

OpenShift Container Platform は Golang の標準ライブラリーの実装である **crypto/tls** を使用し、外部の crypto および TLS ライブラリーには依存しません。追加で、外部ライブラリーに依存して、クライアントは GSSAPI 認証および OpenPGP 署名を使用できます。GSSAPI は通常 OpenSSL の libcrypto

を使用する MIT Kerberos または Heimdal Kerberos のいずれかによって提供されます。OpenPGP 署名の検証は libgpgme および GnuPG によって処理されます。

セキュアでない SSL 2.0 および SSL 3.0 バージョンは、サポート対象外であり、利用できません。OpenShift Container Platform サーバーおよび **oc** クライアントはデフォルトで TLS 1.2 のみを提供します。TLS 1.0 および TLS 1.1 はサーバー設定で有効にできます。サーバーおよびクライアントは共に認証される暗号化アルゴリズムと完全な前方秘匿性を持つ最新の暗号スイートを優先的に使用します。暗号スイートと RC4、3DES、および MD5 などの非推奨で、セキュアでないアルゴリズムは無効化されています。また、内部クライアント (LDAP 認証など) によっては、TLS 1.0 から 1.2 設定の制限が少なく、より多くの暗号スイートが有効化されています。

表1.1 サポートされる TLS バージョン

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

[a] デフォルトで無効にされますが、サーバー設定で有効にできます。

[b] 一部の内部クライアント (LDAP クライアントなど)。

[c] TLS 1.3 は現在も開発中です。

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

- **TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305**
- **TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305**
- **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256**
- **TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256**
- **TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384**
- **TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384**
- **TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256**
- **TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256**

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA

第2章 インフラストラクチャーコンポーネント |

2.1. KUBERNETES インフラストラクチャー

2.1.1. 概要

OpenShift Container Platform 内で、Kubernetes はコンテナーのセット全体でコンテナー化されたアプリケーションを管理し、デプロイメント、メンテナンス、およびアプリケーションののメカニズムを提供します。コンテナーのランタイムは、コンテナー化されたアプリケーションのパッケージを作成してインスタンス化し、実行します。Kubernetes クラスターは 1 つ以上のマスターおよびノードセットで構成されます。

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

注記

OpenShift Container Platform は Kubernetes 1.10 および Docker 1.13 を使用します。

2.1.2. マスター

マスターは、API サーバー、コントローラーマネージャーサーバー、および etcd などのコントロールプレーンのコンポーネントが含まれるホストです。マスターはその Kubernetes クラスターで「ノード」を管理し、「Pod」がノードで実行されるようスケジュールします。

表2.1 マスターコンポーネント

コンポーネント	説明
API サーバー	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。さらに Pod をノードに割り当て、Pod の情報をサービス設定に同期します。
etcd	etcd は、コンポーネントが etcd で必要な状態に戻すための変更の有無を確認する間に永続マスター状態を保存します。etcd は、通常 2n+1 ピアサービスでデプロイされるなど、高可用性のためにオプションで設定できます。
コントローラーマネージャーサーバー	コントローラーマネージャーサーバーは、レプリケーションコントローラーのオブジェクトに変更がないか etcd を監視し、API を使用して希望とする状態を有効化します。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。
HAProxy	オプションで、「高可用性のマスター」を **native** の方法で設定して API マスターのエンドポイント間の負荷を分散する場合に使用します。「クラスターのインストールプロセス」では、**native** の方法で HAProxy 設定できます。または、**native** の方法を使用しつつ、任意のロードバランサーを事前設定できます。

2.1.2.1. コントロールプレーンの静的 Pod

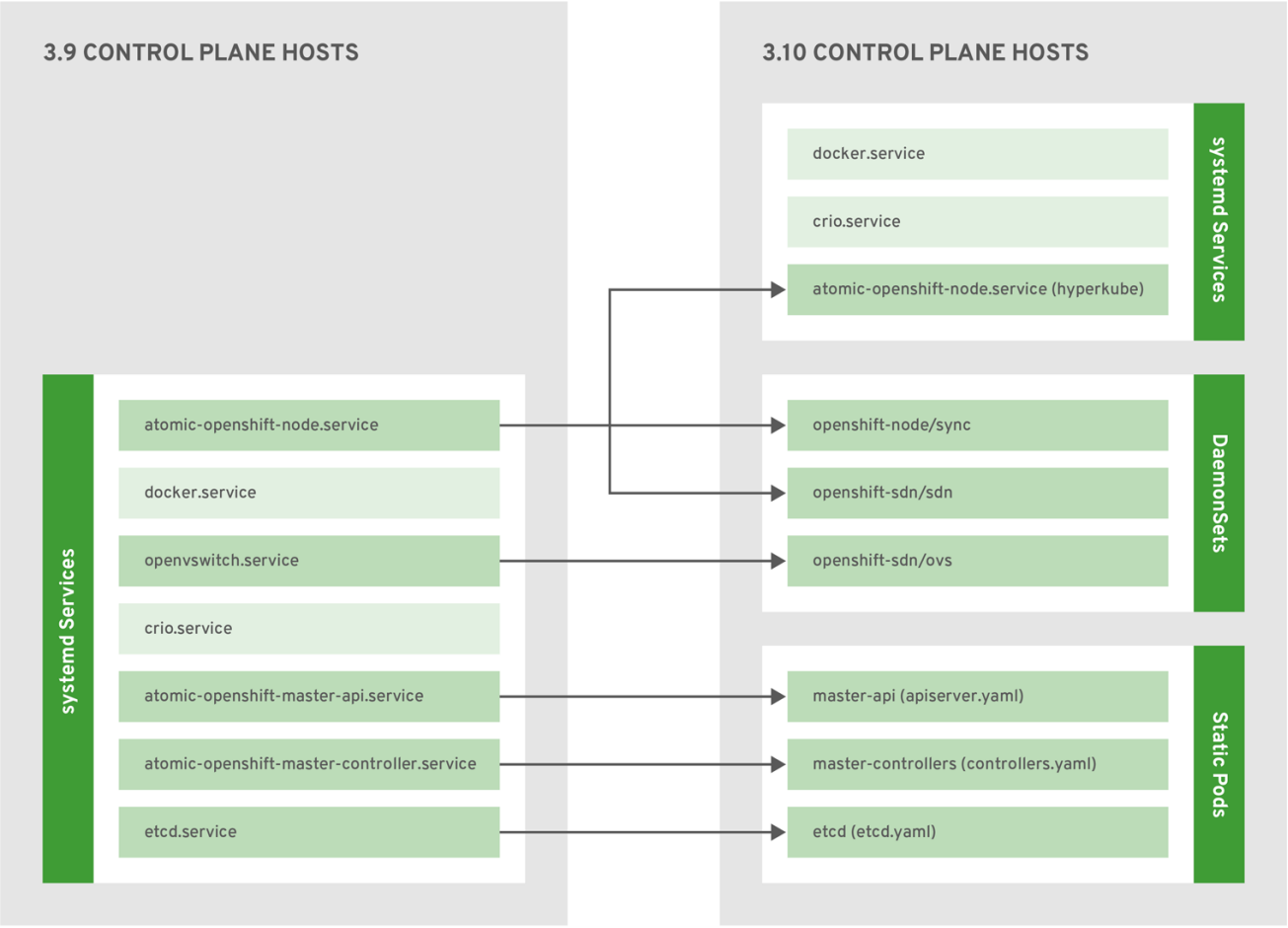
OpenShift Container Platform 3.10 以降で、コアとなるコントロールプレーンのコンポーネントをインストール、操作するデプロイメントモデルが変更されました。3.10 以前は、API サーバーとコントロー

ラーマネージャーのコンポーネントは、**systemd** が操作するスタンドアロンのホストプロセスとして実行されていました。3.10 では、これらのコンポーネントが kubelet が操作する **静的 Pod** に移行されています。

etcd が同じホストに共存しているマスターでは、etcd も静的 Pod に移動されます。RPM ベースの etcd は依然として、マスターではない etcd ホスト上でサポートされます。

さらに、ノードコンポーネントの **openshift-sdn** と **openvswitch** が **systemd** サービスではなく、DaemonSet を使用して実行されます。

図2.1 コントロールプレーンホストのアーキテクチャーの変更



「マスターおよびノードの設定」のトピックに記載されているように、静的 Pod として実行中のコントロールプレーンのコンポーネントの場合でさえも、マスターホストは **/etc/origin/master/master-config.yaml** ファイルからの設定をソースとして使用します。

Pod のミラーリング
マスターノード上の kubelet は自動的に、コントロールプレーンの静的 Pod 毎に、API サーバー上に **ミラーの Pod** を作成し、**kube-system** プロジェクトのクラスターで表示できるようにします。これらの静的 Pod のマニフェストは、デフォルトで、マスターホスト上の **/etc/origin/node/pods** ディレクトリーに配置されている **openshift-ansible** インストーラーによりインストールされます。

これらの Pod は、以下の**hostPath** ボリュームを定義します。

/etc/origin/master	全照明書、設定ファイル、**admin.kubeconfig** ファイルが含まれます。

/var/lib/origin	バイナリーの潜在的なコアダンプとボリュームが含まれます。
/etc/origin/cloudprovider	クラウドプロバイダー固有の設定 (AWS、Azure など) が含まれます。
/usr/libexec/kubernetes/kubelet-plugins	追加のサードパーティーのボリュームプラグインが含まれます。
/etc/origin/kubelet-plugins	システムコンテナーに向けた追加のサードパーティーのボリュームプラグインが含まれます。

以下など、静的 Pod で実行可能な操作には限りがあります。

```
$ oc logs master-api-<hostname> -n kube-system
```

API サーバーからの標準出力を返します。ただし、

```
$ oc delete pod master-api-<hostname> -n kube-system
```

上記のコマンドでは実際には Pod は削除されません。

別の例として、クラスター管理者は、API サーバーの **loglevel** を増やすなど、一般的な操作を実行して、問題が発生した場合により詳細なデータを参照できるようにする場合があります。OpenShift Container Platform 3.10 では、コンテナー内で実行中のプロセスに渡せるように、**/etc/origin/master/master.env** ファイルを編集して、**OPTIONS** 変数の **--loglevel** パラメーターを変更する必要があります。変更を適用するには、コンテナー内で実行中のプロセスを再起動する必要があります。

マスターサービスの再起動
コントロールプレーンの静的 Pod で実行中のコントロールプレーンサービスを再起動するには、マスターホストの **master-restart** コマンドを使用します。

マスター API を再起動します。

```
# master-restart api
```

コントローラーを再起動します。

```
# master-restart controllers
```

etcd を再起動します。

```
# master-restart etcd
```

マスターサービスログの表示

コントロールプレーンの静的 Pod で実行中のコントロールプレーンサービスのログを表示するには、適切なコンポーネントの **master-logs** コマンドを使用します。

```
# master-logs api api
# master-logs controllers controllers
# master-logs etcd etcd
```

2.1.2.2. 高可用性マスター

マスターまたはそのサービスのいずれかが失敗しても、実行中のアプリケーションの可用性はそのまま維持されます。ただし、マスターサービスが失敗すると、システムのアプリケーションの失敗に対応する機能、または新規アプリケーションの作成に対応する機能が低下します。オプションで、クラスターに単一障害点をなくせるように、高可用性 (HA) のマスターを設定できます。

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

1. runbook エントリーは、マスターの再作成のために作成される必要があります。runbook エントリーは、いずれの高可用性サービスに対しても必要なバックアップです。追加ソリューションは、runbook が参照される頻度を制御するのみです。たとえば、マスターホストのコールドスタンドバイは新規アプリケーションの作成または失敗したアプリケーションコンポーネントの復元に 1 分未満のダウンタウンのみを要求する SLA を十分に満たせる可能性があります。

2. 高可用性ソリューションを使用してマスターを設定し、クラスターに単一障害点がなくなるようにします。「クラスターのインストールに関するドキュメント」では、**native** HA 方法を使用し、HAProxy を設定した具体的なサンプルについて説明します。さらに、これらの概念を採用し、HAProxy ではなく **native** 方法を使用して既存の HA ソリューションに対して適用できます。

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

表2.2 HAProxy による可用性マトリクス

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

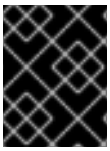
クラスター化された etcd では定足数を維持するためにホストの数は奇数である必要がありますが、マスターサービスには定足数やホストの数が奇数でなければならないという要件はありません。ただし、HA 用に 2 つ以上のマスターサービスが必要になるため、マスターサービスと etcd を共存させる場合には、一律奇数のホストを維持することが一般的です。

2.1.3. ノード

ノードでは、コンテナーのランタイム環境が提供されます。Kubernetes クラスターの各ノードには、マスターで管理される必須のサービスが含まれます。また、ノードには、コンテナーランタイム、kubelet、サービスプロキシーなど、Pod の実行に必要なサービスも含まれます。

OpenShift Container Platform は、ノードをクラウドプロバイダー、物理システムまたは仮想システムから作成します。Kubernetes は、それらのノードの表現であるノードオブジェクトと対話します。マスターはノードオブジェクトからの情報を使用してヘルスチェックでノードを検証します。ノードはこれがヘルスチェックをパスするまで無視され、マスターはノードが有効になるまでチェックを継続します。Kubernetes ドキュメントにはノードのステータスと管理についての詳細が記載されています。

管理者は CLI を使用して OpenShift Container Platform インスタンスの「ノードを管理」できます。ノードサーバー起動時の全設定およびセキュリティーオプションを定義するには、「専用ノードの設定ファイル」を使用します。



重要

推奨される最大ノード数については、「クラスターの制限」のセクションを参照してください。

2.1.3.1. Kubelet

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

コンテナーマニフェストは以下によって kubelet に提供できます。

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

2.1.3.2. サービスプロキシー

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

2.1.3.3. ノードオブジェクト定義

以下は、Kubernetes のノードオブジェクト定義の例になります。

```
apiVersion: v1 (1)
kind: Node (2)
metadata:
  creationTimestamp: null
  labels: (3)
    kubernetes.io/hostname: node1.example.com
  name: node1.example.com (4)
spec:
  externalID: node1.example.com (5)
```

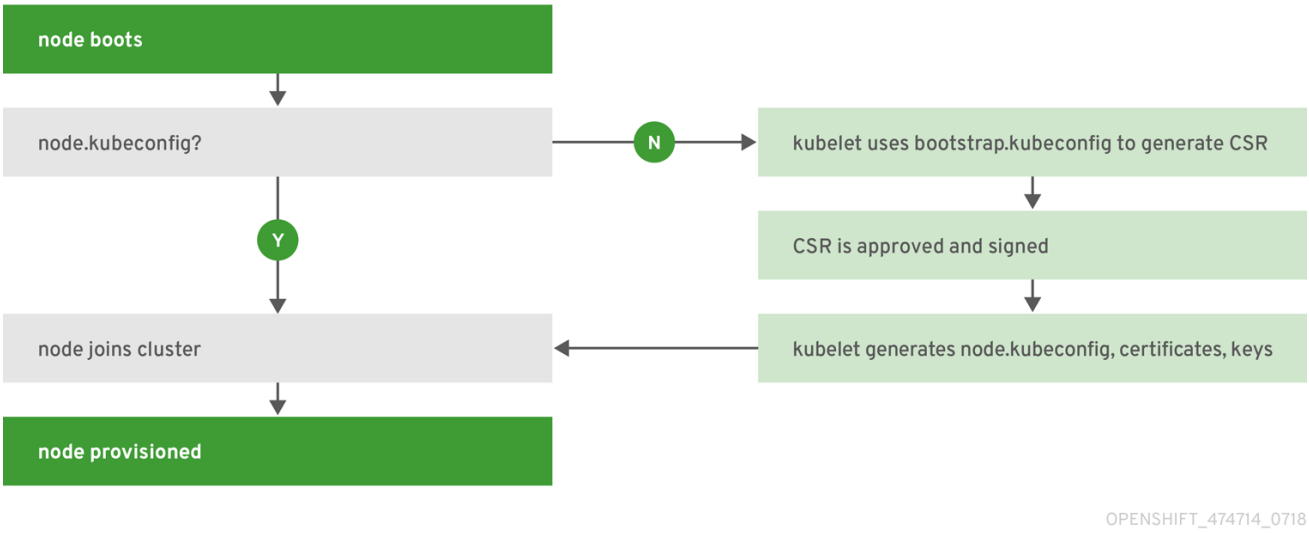
```
status:
  nodeInfo:
    bootID: ""
    containerRuntimeVersion: ""
    kernelVersion: ""
    kubeProxyVersion: ""
    kubeletVersion: ""
    machineID: ""
    osImage: ""
    systemUUID: ""
```

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

2.1.3.4. ノードのブートストラップ

OpenShift Container Platform 3.10 以降では、ノードの設定はマスターからブートストラップされます。つまり、ノードが事前定義済みの設定、クライアントとサーバーの証明書をマスターからプルします。これにより、ノード間の相違点を減らして、より多くの設定を集約し、希望の状態でのクラスターを収束することで、ノードの起動時間を短縮することができます。証明書の回転や集約された証明書の管理は、デフォルトで有効になっています。

図2.2 ノードブートストラップのワークフローに関する概要



ノードサービスの起動時に、ノードは **/etc/origin/node/node.kubeconfig** ファイルと他のノード設定ファイルが存在するかチェックしてから、クラスターに参加します。存在しない場合には、ノードはマスターから設定をプルしてから、クラスターに参加します。

ConfigMaps は、クラスターにノードの設定を保存するのに使用し、ノードホスト上の **/etc/origin/node/node-config.yaml** い設定ファイルを生成します。デフォルトのノードグループやそ

の ConfigMaps の定義については、『クラスターのインストール』ガイドの「ノードグループとホストマッピングの定義」を参照してください。

ノードブートストラップのワークフロー
自動ノードブートストラップのプロセスでは、以下のワークフローを使用します。

1. デフォルトでは、クラスターのインストール時に、**clusterrole**、**clusterrolebinding** および **serviceaccount** オブジェクトのセットが、ノードブートストラップで使用するために作成されます。

   - **system:node-bootstrapper** クラスターロールは、ノードのブートストラップ時に、証明書署名要求 (CSR) の作成に使用します。

     ```
     # oc describe clusterrole.authorization.openshift.io/system:node-bootstrapper

     Name:   system:node-bootstrapper
     Created:  17 hours ago
     Labels:   kubernetes.io/bootstrapping=rbac-defaults
     Annotations:  authorization.openshift.io/system-only=true
       openshift.io/reconcile-protect=false
     Verbs   Non-Resource URLs Resource Names API Groups  Resources
     [create get list watch] []   []  [certificates.k8s.io] [certificatesigningrequests]
     ```

   - 以下の **node-bootstrapper** サービスアカウントを、**openshift-infra** プロジェクトに作成します。

     ```
     # oc describe sa node-bootstrapper -n openshift-infra

     Name:               node-bootstrapper
     Namespace:          openshift-infra
     Labels:             <none>
     Annotations:        <none>
     Image pull secrets:  node-bootstrapper-dockercfg-f2n8r
     Mountable secrets:   node-bootstrapper-token-79htp
                         node-bootstrapper-dockercfg-f2n8r
     Tokens:             node-bootstrapper-token-79htp
                         node-bootstrapper-token-mqn2q
     Events:             <none>
     ```

   - 以下の **system:node-bootstrapper** のクラスターロールのバインディングは、ノードブートストラップのクラスターロールとサービスアカウント向けです。

     ```
     # oc describe clusterrolebindings system:node-bootstrapper

     Name:   system:node-bootstrapper
     Created:  17 hours ago
     Labels:   <none>
     Annotations:  openshift.io/reconcile-protect=false
     Role:   /system:node-bootstrapper
     Users:   <none>
     Groups:   <none>
     ServiceAccounts: openshift-infra/node-bootstrapper
     Subjects:  <none>
     Verbs   Non-Resource URLs Resource Names API Groups  Resources
     [create get list watch] []   []  [certificates.k8s.io] [certificatesigningrequests]
     ```

2. また、デフォルトでは、クラスターのインストール時に、**openshift-ansible** のインストーラーにより、OpenShift Container Platform の証明局とその他のさまざまな証明書、鍵、**kubeconfig** ファイルが **/etc/origin/master** ディレクトリーに作成されます。注目すべき 2 つのファイルは以下のとおりです。

/etc/origin/master/admin.kubeconfig	**system:admin** ユーザーを使用します。
/etc/origin/master/bootstrap.kubeconfig	マスター以外のノードブートストラップノードに使用します。

   a. **etc/origin/master/bootstrap.kubeconfig** は、インストーラーが以下のように **node-bootstrapper** のサービスアカウントを使用するときに作成されます。

```
$ oc --config=/etc/origin/master/admin.kubeconfig \
    serviceaccounts create-kubeconfig node-bootstrapper \
    -n openshift-infra
```

   b. マスターノードでは、ブートストラップファイルとして **/etc/origin/master/admin.kubeconfig** を使用し、**/etc/origin/node/boostrap.kubeconfig** にコピーされます。他のマスター以外のノードでは、**/etc/origin/master/bootstrap.kubeconfig** ファイルは、他のノードすべてに対して、各ノードホストごとに **/etc/origin/node/boostrap.kubeconfig** にコピーされます。

   c. 次に、**/etc/origin/master/bootstrap.kubeconfig** は、以下のように、フラグ **--bootstrap-kubeconfig** を使用して kubelet に渡されます。

```
--bootstrap-kubeconfig=/etc/origin/node/bootstrap.kubeconfig
```

3. kubelet は、指定の **/etc/origin/node/bootstrap.kubeconfig** ファイルで先に起動します。内部での初期接続後に、kubelet は証明書署名要求 (CSR) を作成して、マスターに送信します。

4. CSR はコントローラーマネージャーを使用して検証、承認されます (特に、証明署名コントローラー)。承認された場合には、kubelet クライアントとサーバー証明書は **/etc/origin/node/ceritificates** ディレクトリーに作成されます。以下に例を示します。

```
# ls -al /etc/origin/node/certificates/
total 12
drwxr-xr-x. 2 root root  212 Jun 18 21:56 .
drwx------. 4 root root  213 Jun 19 15:18 ..
-rw-------. 1 root root 2826 Jun 18 21:53 kubelet-client-2018-06-18-21-53-15.pem
-rw-------. 1 root root 1167 Jun 18 21:53 kubelet-client-2018-06-18-21-53-45.pem
lrwxrwxrwx. 1 root root   68 Jun 18 21:53 kubelet-client-current.pem ->
/etc/origin/node/certificates/kubelet-client-2018-06-18-21-53-45.pem
-rw-------. 1 root root 1447 Jun 18 21:56 kubelet-server-2018-06-18-21-56-52.pem
lrwxrwxrwx. 1 root root   68 Jun 18 21:56 kubelet-server-current.pem ->
/etc/origin/node/certificates/kubelet-server-2018-06-18-21-56-52.pem
```

5. CSR の承認後に、**node.kubeconfig** ファイルが **/etc/origin/node/node.kubeconfig** に作成されます。

6. kubelet は、**/etc/origin/node/certificates/** ディレクトリーにある **/etc/origin/node/node.kubeconfig** ファイルと証明書で再起動されます。再起動後に、クラスターへの参加の準備ができます。

ノード設定のワークフロー
ノードの設定を取得するには、以下のワークフローを使用します。

1. 最初に、ノードの kubelet は、**/etc/origin/node/** ディレクトリーにあるブートストラップの設定ファイル **bootstrap-node-config.yaml** を使用して起動され、ノードのプロビジョニング時に作成されます。

2. 各ノードで、**atomic-openshift-node** サービスファイルは、**/usr/local/bin/** ディレクトリーにあるローカルスクリプト **openshift-node** を使用して、指定の **bootstrap-node-config.yaml** で kubelet を起動します。

3. マスター毎に、**/etc/origin/node/pods** ディレクトリーには、マスターに静的 Pod として作成された **apiserver**、**controller** および **etcd** の Pod のマニフェストが含まれます。

4. クラスターのインストール時に、sync DaemonSet が作成され、ノードごとに同期 Pod を作成します。同期 Pod は、**/etc/sysconfig/atomic-openshift-node** ファイル内の変更をモニタリングします。特に、設定される **BOOTSTRAP_CONFIG_NAME** を監視します。**BOOTSTRAP_CONFIG_NAME** は、**openshift-ansible** インストーラーにより設定され、対象のノードが所属するノード設定グループをもとにした ConfigMap の名前です。
デフォルトでは、インストーラーは以下のノード設定グループを作成します。

   - **node-config-master**
   - **node-config-infra**
   - **node-config-compute**
   - **node-config-all-in-one**
   - **node-config-master-infra**

   各グループの ConfigMap は **openshift-node** プロジェクトに作成されます。

5. 同期 Pod は、**BOOTSTRAP_CONFIG_NAME** の値セットをもとに適切な ConfigMap を抽出します。

6. 同期 Pod は kubelet 設定に ConfigMap データを変換し、対象のノードホスト用に **/etc/origin/node/node-config.yaml** を作成します。このファイルに変更が加えられると (またはファイルが初めて作成される場合には)、kubelet が再起動されます。

ノード設定の変更
ノードの設定は、**openshift-node** プロジェクトの適切な ConfigMap を編集して変更します。**/etc/origin/node/node-config.yaml** は直接変更しないでください。

たとえば、**node-config-compute** グループに含まれるノードの場合は、以下のコマンドを使用して ConfigMap を編集します。

```
$ oc edit cm node-config-compute -n openshift-node
```

2.2. CONTAINER レジストリー

2.2.1. 概要

OpenShift Container Platform は、Docker Hub、サードパーティーによって実行されるプライベートレジストリー、および統合 OpenShift Container Platform レジストリーを含む、イメージのソースとして Docker レジストリー API を実装するすべてのサーバーを利用できます。

2.2.2. 統合 OpenShift Container レジストリー

OpenShift Container Platform には **OpenShift Container レジストリー** (OCR) という統合コンテナーレジストリーがあり、新規イメージリポジトリーのプロビジョニングをオンデマンドで自動化できるようになります。この OCR を使用することで、組み込みのアプリケーション「ビルド」の保管場所が用意され、作成されたイメージをプッシュできます。

新規イメージが OCR にプッシュされるたびに、レジストリーは OpenShift Container Platform に新規イメージ、および namespace、名前、およびイメージメタデータなどの関連する情報について通知します。OpenShift Container Platform の異なる部分が新規イメージに対応し、新規ビルドおよびデプロイメントを作成します。

また、OCR はビルドおよびデプロイメントの統合なしに単独でコンテナーレジストリーとして機能するスタンドアロンコンポーネントとしてデプロイできます。詳細については、「OpenShift Container レジストリーのスタンドアロンデプロイメントのインストール」を参照してください。

2.2.3. サードパーティーレジストリー

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

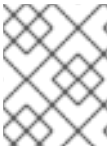
2.2.3.1. 認証

OpenShift Container Platform はユーザーが指定する認証情報を使ってプライベートリポジトリーにアクセスするためにレジストリーと通信します。これにより、OpenShift はベートレジストリーから/へのイメージのプッシュ/プルを行うことができます。「認証」のトピックには詳細が記載されています。

2.3. WEB コンソール

2.3.1. 概要

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



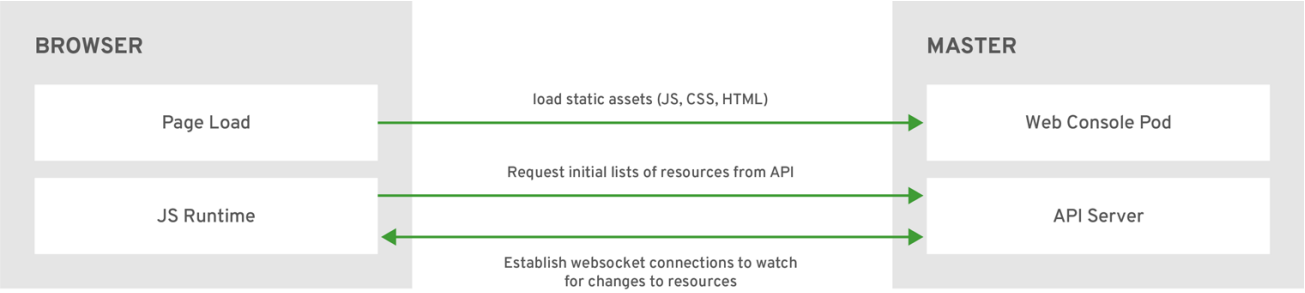
注記

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

Web コンソールはマスターで Pod として実行されます。Web コンソールを実行するために必要な静的なアセットは Pod によって提供されます。また、管理者は拡張を使用して「Web コンソールのカスタマイズ」を実行できます。これにより、Web コンソールの読み込み時にスクリプトを実行し、カスタムスタイルシートを読み込むことができます。

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

図2.3 s Web コンソール要求アーキテクチャー



Web コンソール用に設定されたホスト名と IP アドレスは、ブラウザーが要求を クロスオリジン とみなした場合でさえも、安全に API サーバーにアクセスできるように、ホワイトリスト化されます。別のホスト名を使用して、Web アプリケーションから API サーバーにアクセスするには、**openshift start** の **--cors-allowed-origins** オプションを指定するか、関連の master 設定ファイルパラメーター **corsAllowedOrigins** から、対象のホスト名をホワイトリスト化する必要があります。

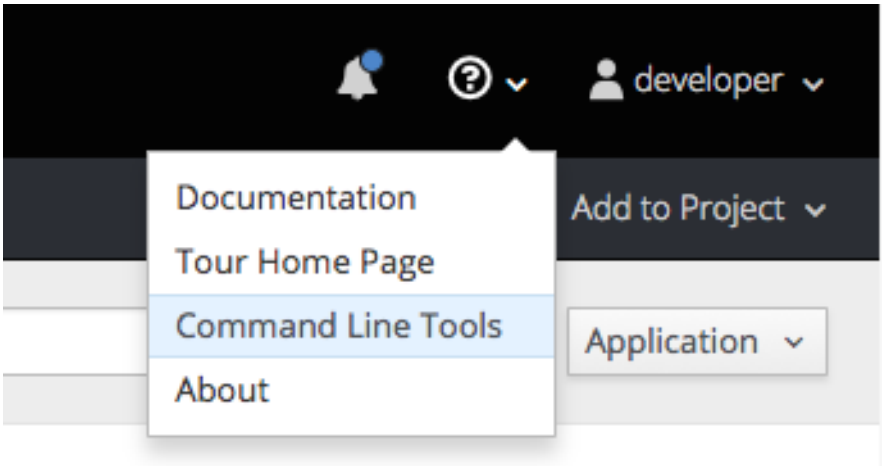
corsAllowedOrigins パラメーターは設定フィールドで制御されます。値に対してピニングやエスケープは実行されません。以下は、ホスト名をピニングし、ドットをエスケープする方法の例を示しています。

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

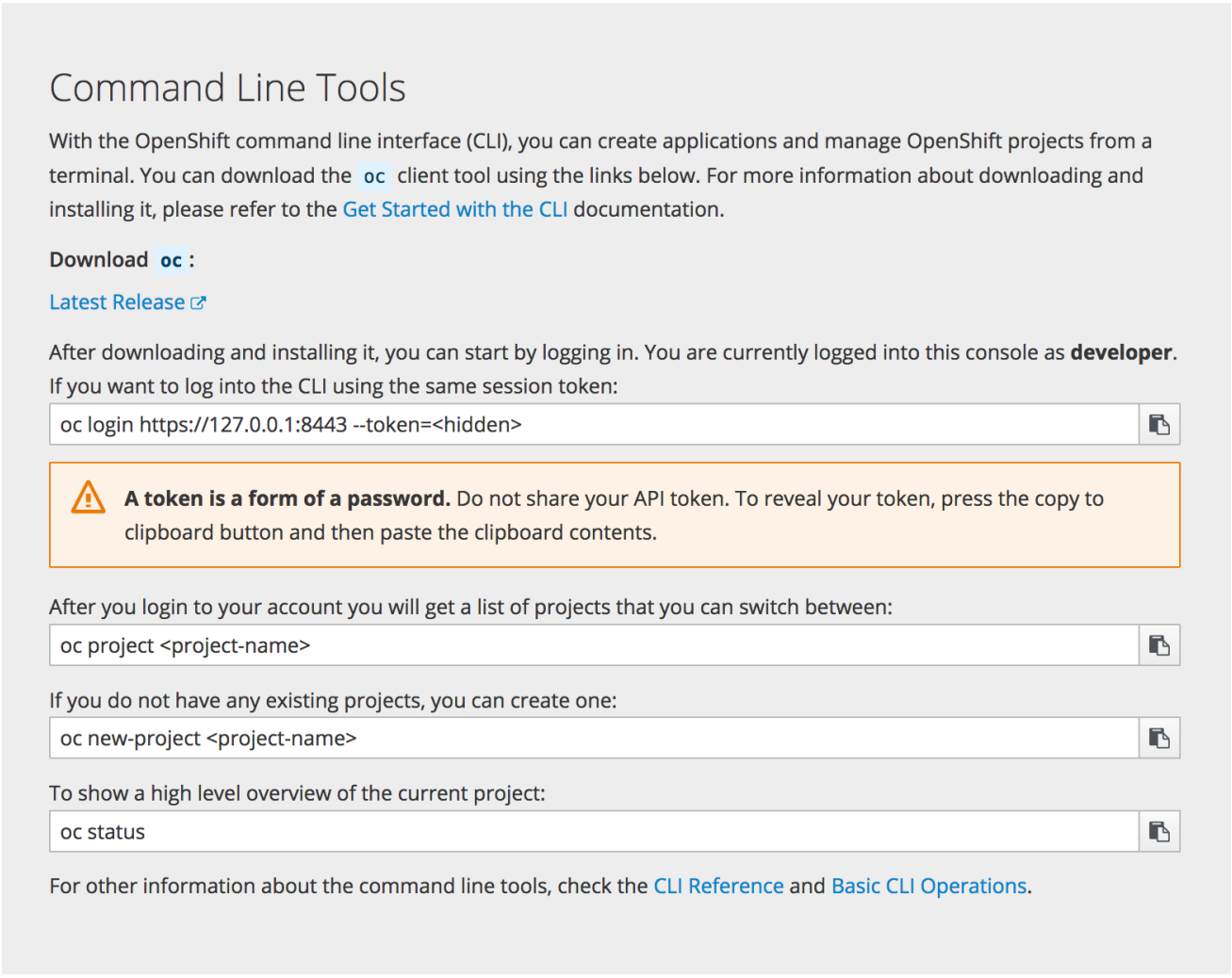
- **(?i)** は大文字/小文字を区別します。
- // はドメインの開始にピニングします (または **http:** または **https:** の後のダブルスラッシュに一致します)。
- **\.** はドメイン名のドットをエスケープします。
- **(:|\z)** はドメイン名の終了に一致します **(\z)** またはポートセパレーター **(:)**。

2.3.2. CLI ダウンロード

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



クラスター管理者は、これらのリンクの追加のカスタマイズを実行できます。



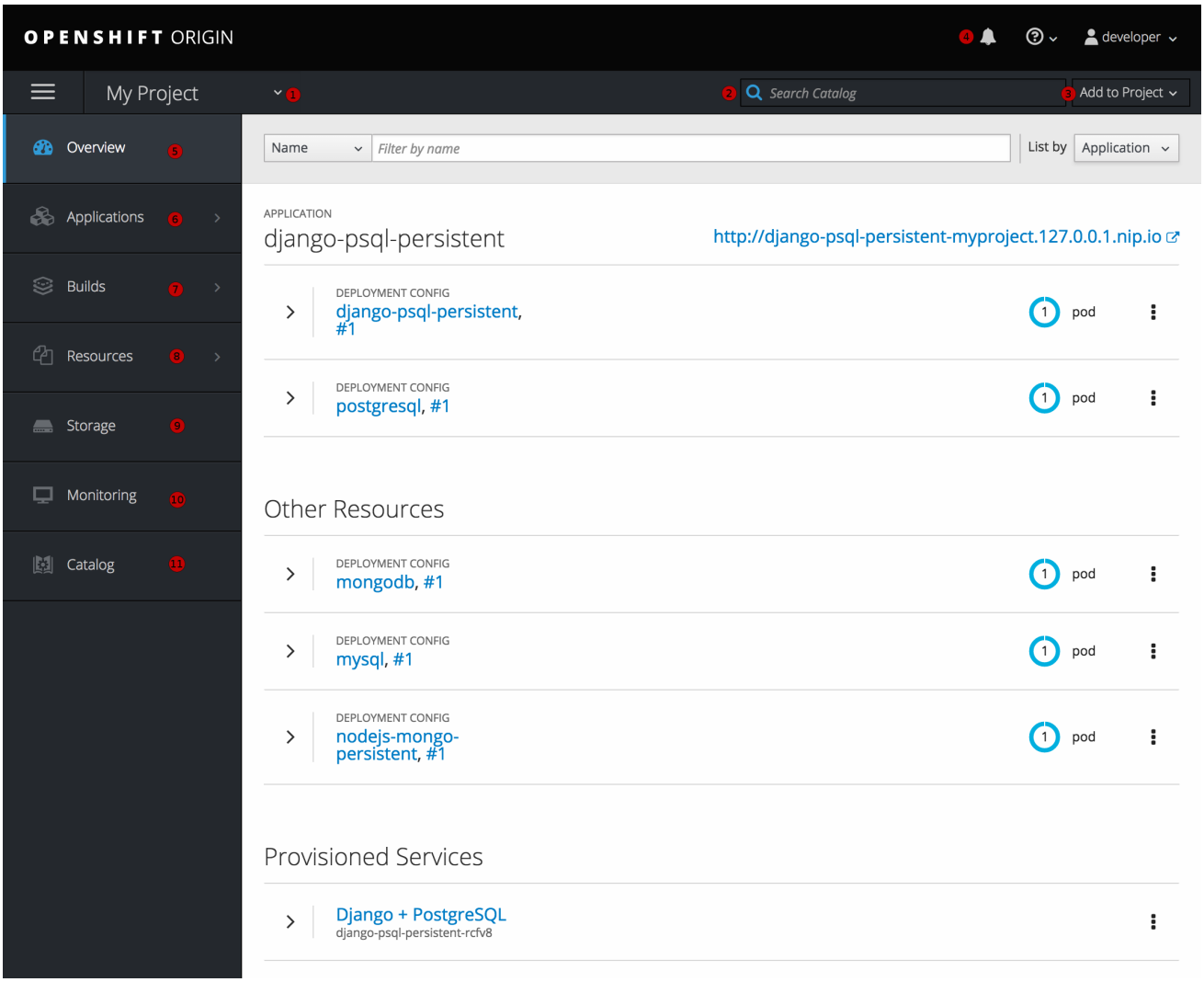
2.3.3. ブラウザーの要件

OpenShift Container Platform のテスト済みの統合を確認します。

2.3.4. プロジェクトの概要

「ログイン」後に、Web コンソールは開発者に現在選択されている「プロジェクト」の概要を提供します。

図2.4 Web コンソールのプロジェクト概要



プロジェクトセレクターを使うと、アクセスできる「プロジェクト間の切り替え」を実行できます。

プロジェクトビューからサービスをすぐに見つけるには、検索条件に入力します。

「ソースリポジトリー」またはサービスカタログのサービスを使用して新規アプリケーションを作成します。

プロジェクトに関連する通知。

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

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

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

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

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

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

Catalog タブ: プロジェクト内からカタログにすぐに移動します。

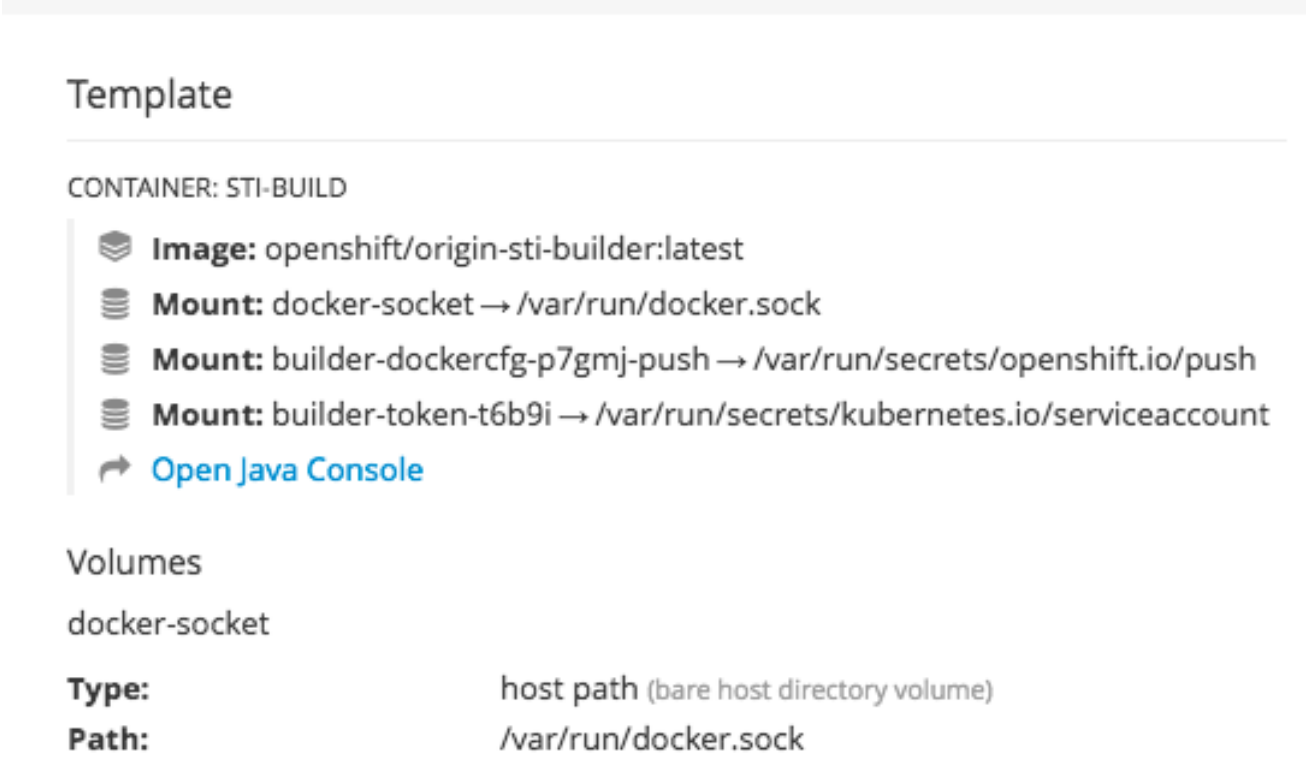
注記

Cockpit は OpenShift Container Platform 3.1 以降に自動的にインストールされて有効化されるので、後で開発環境をモニターするのに役立ちます。『Red Hat Enterprise Linux Atomic Host: Getting Started with Cockpit』は Cockpit についての詳細を記載しています。

2.3.5. JVM コンソール

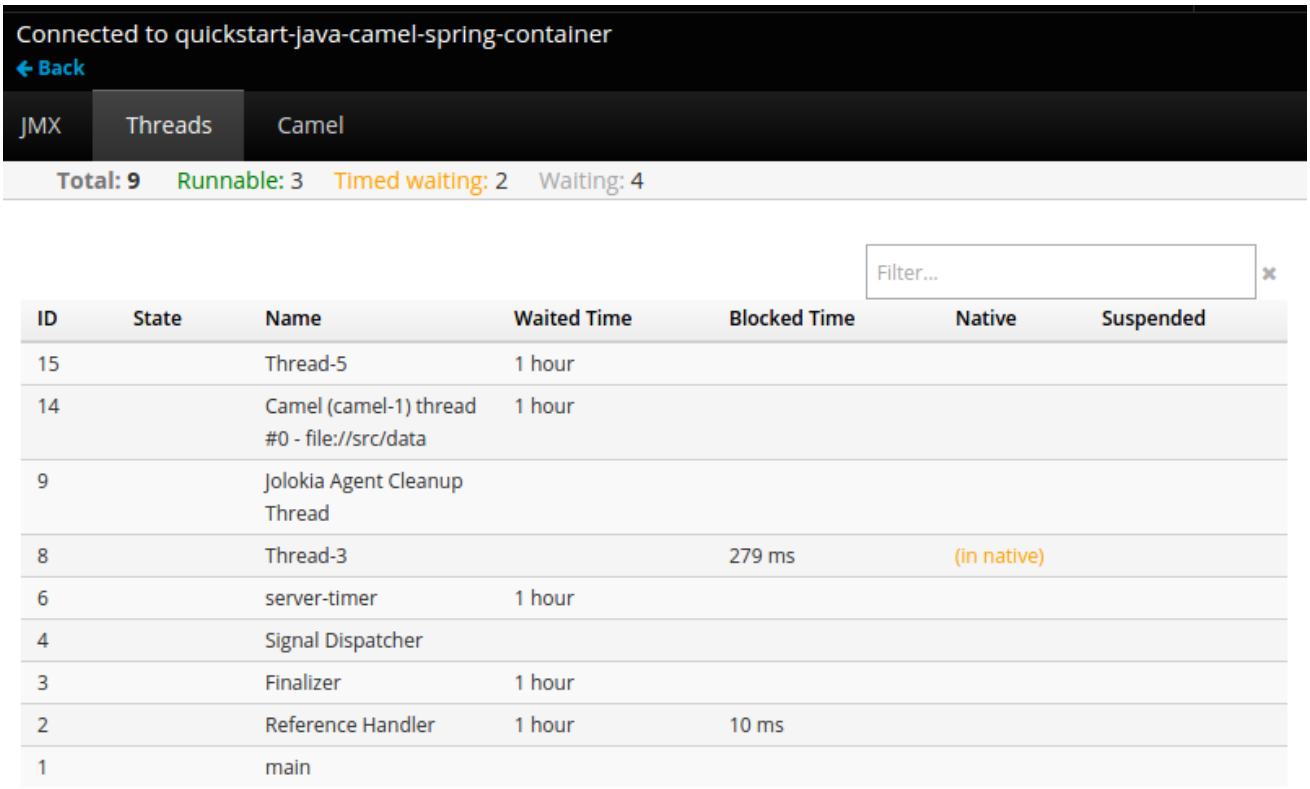
Java イメージをベースとする Pod の場合、Web コンソールは関連する統合コンポーネントを表示し、管理するための hawt.io ベースの JVM コンソールへのアクセスも公開します。**Connect** リンクは、コンテナーに **jolokia** という名前のポートがある場合は、**Browse → Pods** ページの Pod の詳細に表示されます。

図2.5 JVM コンソールへのリンクを持つ Pod



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

図2.6 JVM コンソール



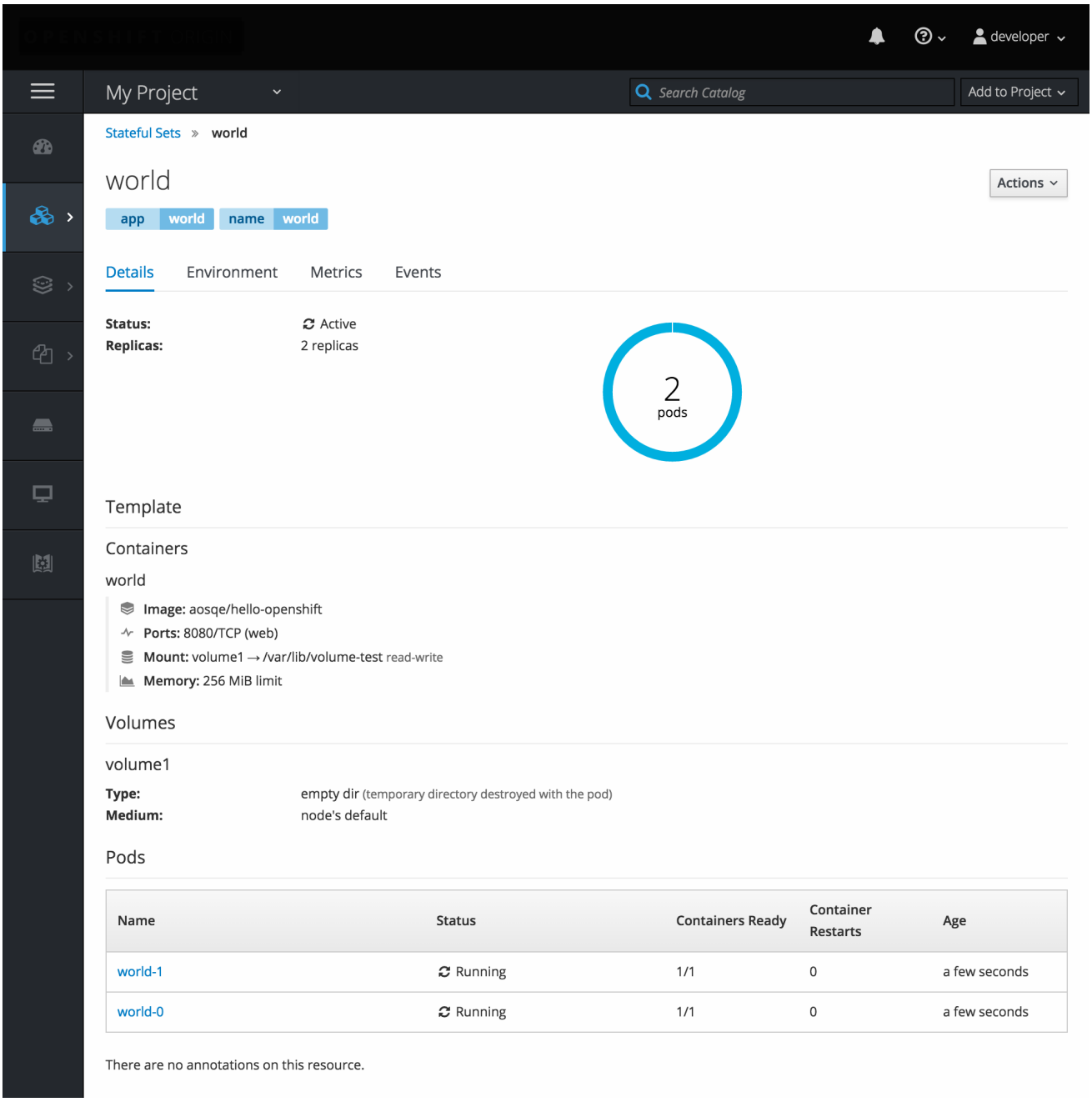
以下のページが利用可能になります。

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

2.3.6. StatefulSets

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

図2.7 OpenShift Container Platform の StatefulSet



第3章 コアとなる概念

3.1. 概要

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

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

3.2. コンテナーおよびイメージ

3.2.1. コンテナー

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

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

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

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

3.2.1.1. Init コンテナー

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

詳細については、「Pod およびサービス」を参照してください。

3.2.2. イメージ

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

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

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

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

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

警告

公式の OpenShift Container Platform イメージには **latest** タグを使用しないでください。これらは **openshift3/** で開始するイメージです。**latest** は **3.4**、または **3.5** などの数多くのバージョンを参照できます。

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

vX.Y

vX.Y タグは X.Y.Z-<number> をポイントします。たとえば、**registry-console** イメージが v3.4 に更新されると、これは最新の 3.4.Z-<number> タグをポイントします (例: 3.4.1-8)。

X.Y.Z

上記の vX.Y サンプルと同様です。X.Y.Z タグは最新の X.Y.Z-<number> をポイントします。たとえば、3.4.1 は 3.4.1-8 をポイントします。

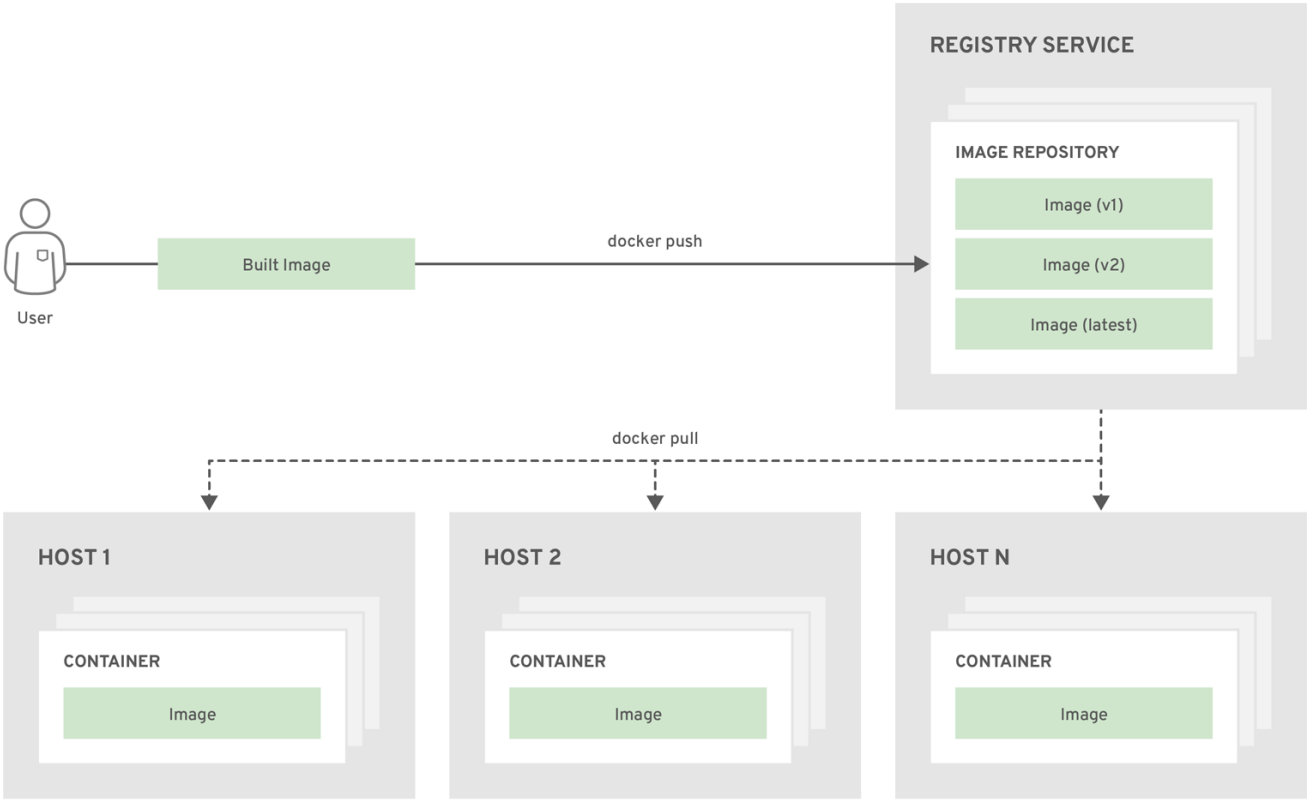
X.Y.Z-<number>

タグは一意であり、変更されません。このタグを使用する際、イメージが更新される際にイメージはタグを更新しません。たとえば、イメージが更新される場合でも、3.4.1-8 は 3.4.1-8 を常にポイントします。

3.2.3. コンテナーレジストリー

コンテナーレジストリーは Docker 形式のコンテナーイメージの保存および取得を行うサービスです。レジストリーには、1 つ以上のイメージリポジトリーのコレクションが含まれます。各イメージリポジトリーには 1 つ以上のタグ付けされたイメージが含まれます。Docker は独自のレジストリーである Docker Hub を提供しますが、プライベートまたはサードパーティーのレジストリーを使用することもできます。Red Hat はサブスクライバー用に **registry.access.redhat.com** でレジストリーを提供します。OpenShift Container Platform はカスタムコンテナーイメージを管理するための独自の内部レジストリーも提供します。

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



3.3. POD およびサービス

3.3.1. Pod

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

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

Pod にはライフサイクルがあります。それらは定義されてから、ノードを実行するために割り当てら

れ、コンテナーが終了するまで実行されるか、その他の理由でコンテナーが削除されるまで実行されます。ポリシーおよび終了コードによっては、Pod は終了後に削除されるか、コンテナーのログへのアクセスを有効にするために保持される可能性があります。

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

注記

OpenShift Container Platform ノードホストの最大数については、「クラスター制限」を参照してください。

警告

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

以下は、実際に OpenShift Container Platform インフラストラクチャーの一部である統合コンテナーレジストリーという、長期に実行されるサービスの Pod のサンプル定義です。これは数多くの Pod の機能を示し、それらのほとんどは他のトピックで説明されるため、ここではこれらについて簡単に説明します。

例3.1 Pod オブジェクト定義 (YAML)

```
apiVersion: v1
kind: Pod
metadata:
  annotations: { ... }
  labels:                                  (1)
    deployment: docker-registry-1
    deploymentconfig: docker-registry
    docker-registry: default
  generateName: docker-registry-1-         (2)
spec:
  containers:                              (3)
  - env:                                   (4)
    - name: OPENSHIFT_CA_DATA
      value: ...
    - name: OPENSHIFT_CERT_DATA
      value: ...
    - name: OPENSHIFT_INSECURE
      value: "false"
    - name: OPENSHIFT_KEY_DATA
      value: ...
    - name: OPENSHIFT_MASTER
      value: https://master.example.com:8443
```

```
    image: openshift/origin-docker-registry:v0.6.2 5
    imagePullPolicy: IfNotPresent
    name: registry
    ports: 6
    - containerPort: 5000
      protocol: TCP
    resources: {}
    securityContext: { ... } 7
    volumeMounts: 8
    - mountPath: /registry
      name: registry-storage
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-br6yz
      readOnly: true
  dnsPolicy: ClusterFirst
  imagePullSecrets:
  - name: default-dockercfg-at06w
  restartPolicy: Always 9
  serviceAccount: default 10
  volumes: 11
  - emptyDir: {}
    name: registry-storage
  - name: default-token-br6yz
    secret:
      secretName: default-token-br6yz
```

1. Pod には 1 つまたは複数の「ラベル」で「タグ付け」することができ、このラベルを使用すると、一度の操作で Pod グループの選択や管理が可能になります。これらのラベルは、キー/値形式で **メタデータ** ハッシュに保存されます。この例で使用されているラベルは **docker-registry=default** です。

2. Pod にはそれらの namespace 内に任意の名前がなければなりません。Pod 定義は **generateName** 属性で名前のベースを指定できますが、一意の名前を生成するためにランダムな文字が自動的に追加されます。

3. **コンテナー** はコンテナー定義の配列を指定します。この場合 (ほとんどの場合)、これは 1 つのみになります。

4. 必要な値を各コンテナーに渡すために、環境変数を指定することができます。

5. Pod の各コンテナーは独自の Docker 形式のコンテナーイメージ からインスタンス化されます。

6. コンテナーは、Pod の IP で利用可能にされるポートにバインドできます。

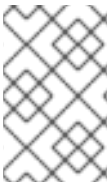
7. OpenShift Container Platform は、コンテナーが特権付きコンテナーとして実行されるか、選択したユーザーとして実行されるかどうかを指定するセキュリティーコンテキストを定義します。デフォルトのコンテキストには多くの制限がありますが、管理者は必要に応じてこれを変更できます。

8. コンテナーは外部ストレージボリュームがコンテナー内にマウントされるかどうかを指定します。この場合、レジストリーのデータを保存するためのボリュームと、OpenShift Container Platform API に対して要求を行うためにレジストリーが必要とする認証情報へのアクセス用のボリュームがあります。

9 Pod 再起動ポリシーと使用可能な値の **Always**、**OnFailure**、および **Never** です。デフォルト値は **Always** です。

10 OpenShift Container Platform API に対して要求する Pod は一般的なパターンです。この場合、**serviceAccount** フィールドがあり、これは要求を行う際に Pod が認証する必要のある「サービスアカウント」ユーザーを指定するために使用されます。これにより、カスタムインフラストラクチャーコンポーネントの詳細なアクセス制御が可能になります。

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



注記

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

3.3.1.1. Pod 再起動ポリシー

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

以下の値を使用できます。

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

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

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

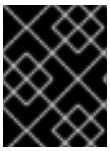
Pod のコンテナーが失敗し、再起動ポリシーが **OnFailure** に設定される場合、Pod はノード上に留まり、コンテナーが再起動します。コンテナーを再起動させない場合には、再起動ポリシーの **Never** を使用します。

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

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

3.3.1.2. Pod の Preset (プリセット) を使用した情報の Pod への挿入

Pod の Preset は、ユーザーが指定する情報を Pod の作成時に Pod に挿入するオブジェクトです。



重要

OpenShift Container Platform 3.7 の時点で、Pod の Preset はサポートされなくなりました。

挿入できる Pod の Preset オブジェクトの使用

- シークレットオブジェクト
- **ConfigMap** オブジェクト
- ストレージボリューム
- コンテナーボリュームのマウント
- 環境変数

開発者は、すべての情報を Pod に追加するために Pod ラベルが PodPreset のラベルセレクターに一致することを確認する必要があります。Pod のラベルは Pod を一致するラベルセレクターを持つ 1 つ以上の Pod Preset オブジェクトに関連付けます。

Pod Preset を使用し、開発者は Pod が消費するサービスの詳細を把握せずに Pod のプロビジョニングを実行できます。管理者はサービスの設定項目を開発者に非表示にできます。この際、開発者の Pod のデプロイを妨げることはありません。

注記

Pod Preset 機能は Service Catalog がインストールされている場合にのみ利用できます。

Pod 仕様で **podpreset.admission.kubernetes.io/exclude: "true"** パラメーターを使用し、特定の Pod が挿入されないようにすることができます。「Pod 仕様のサンプル」を参照してください。

詳細は、「Pod の Preset (プリセット) を使用した情報の Pod への挿入」を参照してください。

3.3.2. Init コンテナー

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

Pod の起動時でボリュームの初期化後に、init コンテナーは順番に起動します。各 init コンテナーは、次のコンテナーが起動する前に正常に終了する必要があります。init コンテナーが (ランタイムを原因

に) 起動に失敗するか、または失敗して終了する場合、これは Pod の 再起動ポリシーに基づいてリタイアします。

Pod は init コンテナーがすべて成功するまで準備状態になりません。

一部の init コンテナーの使用例については、Kubernetes ドキュメントを参照してください。

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

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

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice (1)
    image: busybox
    command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice; sleep 2; done;']
  - name: init-mydb (2)
    image: busybox
    command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sleep 2; done;']
```

(1) **myservice** コンテナーを指定します。

(2) **mydb** コンテナーを指定します。

各 init コンテナーには、**readinessProbe** を除くすべてのアプリコンテナーのフィールドが含まれます。Init コンテナーは Pod 起動が継続するために終了する必要があり、完了以外の readiness を定義することはできません。

Init コンテナーには Pod の **activeDeadlineSeconds** およびコンテナーの **livenessProbe** を追加し、init コンテナーが完全に機能しなくなるのを防ぐことができます。有効な期限には init コンテナーで使用される時間が含まれます。

3.3.3. Services

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

サービスへの外部アクセスを許可するには、クラスターの「外部」にある追加の **externalIP** および **ingressIP** アドレスをサービスに割り当てることができます。これらの **externalIP** アドレスには、サービスへの「高可用」アクセスを提供する仮想 IP アドレスを使用することもできます。

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

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

例3.3 サービスオブジェクト定義 (YAML)

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

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

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

3.3.3.1. サービス externalIP

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

externalIP は、クラスター管理者が master-config.yaml ファイルで設定される **ExternalIPNetworkCIDRs** 範囲から選択する必要があります。**master-config.yaml** に変更が加えられると、マスターサービスを再起動する必要があります。

例3.4 externalIPNetworkCIDR /etc/origin/master/master-config.yaml のサンプル

```
networkConfig:
  externalIPNetworkCIDR: 192.0.1.0.0/24
```

例3.5 サービス externalIP 定義 (JSON)

```
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
        "name": "my-service"
    },
    "spec": {
        "selector": {
            "app": "MyApp"
        },
        "ports": [
            {
                "name": "http",
                "protocol": "TCP",
                "port": 80,
                "targetPort": 9376
            }
        ],
        "externalIPs" : [
            "192.0.1.1"          (1)
        ]
    }
}
```

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

3.3.3.2. サービス ingressIP

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

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

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

注記

「高可用性」を設定している場合、この範囲は 256 アドレス未満にする必要があります。

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

```
networkConfig:
  ingressIPNetworkCIDR: 172.29.0.0/16
```

3.3.3.3. サービス NodePort

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

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

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

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

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

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

注記

nodePort の設定は特権付きの操作で実行されます。

3.3.3.4. サービスプロキシーモード

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

iptables ベースの実装はさらに効率的ですが、これには、すべてのエンドポイントが接続を常に受け入れられることが条件になります。ユーザースペースの実装は速度が遅くなりますが、機能するエンドポイントが見つかるまで複数のエンドポイントを試行できます。適切な readiness チェック (または通常信頼できるノードおよび Pod) が必要であり、次に **iptables** ベースのサービスプロキシーが最適なオプションになります。または、ノード設定ファイルを編集してクラスターのインストール時またはデプロイ後にユーザースペースベースのプロキシーを有効にできます。

3.3.3.5. ヘッドレスサービス

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

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

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

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

3.3.3.5.1. ヘッドレスサービスの作成

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

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

Pod の一覧

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

ヘッドレスサービスを以下のように定義できます。

ヘッドレスサービス定義

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

(1) ヘッドレスサービスの名前です。

2 **clusterIP** 変数を **None** に設定すると、ヘッドレスサービスが宣言されます。

3 **frontend** のラベルが付いたすべての Pod を選択します。

また、ヘッドレスサービスには独自の IP アドレスがありません。

```
$ oc get svc
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
frontend           ClusterIP   172.30.232.77   <none>        5432/TCP   12m
frontend-headless  ClusterIP   None            <none>        5432/TCP   10m
```

3.3.3.5.2. ヘッドレスサービスを使用したエンドポイントの検出

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

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

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

ヘッドレスサービスの場合、個別 Pod の IP の一覧を取得します。

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

注記

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

3.3.4. ラベル

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

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

ラベルは、以下の例にあるように単純なキー/値のペアです。

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

以下を検討してください。

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

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

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

3.3.5. エンドポイント

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

場合によっては、サービスを作成する場合でも、OpenShift Container Platform クラスターの Pod ではなく、外部ホストでサポートされるようにする必要があります。この場合、サービスの **selector** フィールドを省略し、「エンドポイントオブジェクトを手動で作成」できます。

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

3.4. プロジェクトとユーザー

3.4.1. ユーザー

OpenShift Container Platform との対話はユーザーに関連付けられます。OpenShift Container Platform ユーザーオブジェクトは、「ユーザーまたはユーザーグループにロールを追加」して、システム内のパーミッションを付与できるアクターを表示します。

ユーザーにはいくつかのタイプが存在します。

通常ユーザー	これは、大半の対話型の OpenShift Container Platform ユーザーが表示される方法です。通常ユーザーは、初回ログイン時にシステムに自動的に作成され、API で作成できます。通常ユーザーは、**User** オブジェクトで表示されます。例: **joe alice**
システムユーザー	これらの多くは、インフラストラクチャーが API と安全に対話できるようにすることを主な目的として定義される際に自動的に作成されます。これらには、クラスター管理者 (すべてのものへのアクセスを持つ)、ノードごとのユーザー、ルーターおよびレジストリーで使用できるユーザー、その他が含まれます。最後に、非認証要求に対してデフォルトで使用される **匿名** システムユーザーがあります。例: **system:admin system:openshift-registry system:node:node1.example.com**

サービスアカウント	特殊なシステムユーザーがプロジェクトに関連付けられます。それらの中には、プロジェクト管理者が各プロジェクトのコンテンツへのアクセスを定義する目的で追加作成する状態で、プロジェクトの初回作成時に自動作成されるものがあります。サービスアカウントは **ServiceAccount** オブジェクトで表示されます。例: **system:serviceaccount:default:deployer** **system:serviceaccount:foo:builder**

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

3.4.2. Namespace

Kubernetes namespace はクラスターのリソースのスコープを設定するメカニズムを提供します。OpenShift Container Platform では、プロジェクトは追加のアノテーションを含む Kubernetes namespace です。

Namespace は以下の一意のスコープを提供します。

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

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

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

3.4.3. プロジェクト

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

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

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

各プロジェクトは、以下の独自のセットのスコープを設定します。

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

クラスター管理者は「プロジェクトの作成」や、プロジェクトの「管理者権限の委任」をユーザーコミュニティーの任意のメンバーに対して実行できます。また、クラスター管理者は開発者が「独自のプロジェクト」を作成することも許可します。

開発者および管理者は、「CLI」または「Web コンソール」を使用してプロジェクトとの対話が可能です。

3.4.3.1. インストール時にプロビジョニングされるプロジェクト

OpenShift Container Platform は、追加設定なしで使用できるプロジェクトが多数含まれていますが、**openshift-** で始まるプロジェクトがユーザーにとって最も重要なプロジェクトです。

3.10 以降のバージョンでは、**openshift-** で始まるプロジェクトが多数あり、Pod や他のインフラストラクチャーコンポーネントとしてマスターコンポーネントをホストすることができます。「重要な Pod アノテーション」が指定されている場合に、これらの namespace で作成された Pod は、重要とみなされ、kubelet で必ず受け付けられるようになります。これらの namespace でマスターコンポーネント用に作成された Pod は、すでに重要とマークされています。

3.5. ビルドおよびイメージストリーム

3.5.1. ビルド

ビルド は、入力パラメーターを結果として作成されるオブジェクトに変換するプロセスです。一般的に、このプロセスは入力パラメーターまたはソースコードを実行可能なイメージに変換するために使用されます。**BuildConfig** オブジェクトはビルドプロセス全体の定義です。

OpenShift Container Platform は、Docker 形式のコンテナーをビルドイメージから作成し、それらをコンテナーレジストリーにプッシュして Kubernetes を利用します。

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

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

- Docker ビルド
- Source-to-Image (S2I) ビルド
- カスタムビルド

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

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

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

- 継続的インテグレーション
- 継続的デプロイメント

ビルドコマンドの一覧については、『開発者ガイド』を参照してください。

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

3.5.1.1. Docker ビルド

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

3.5.1.2. Source-to-Image (S2I) ビルド

「Source-to-Image (S2I)」は複製可能な Docker 形式のコンテナーイメージをビルドするためのツールです。これはアプリケーションソースをコンテナーイメージに挿入し、新規イメージをアセンブルして実行可能なイメージを生成します。新規イメージはベースイメージ (ビルダー) とビルドされたソースを組み込み、**docker run** コマンドで利用可能な状態になります。S2I は増分ビルドをサポートします。これは以前にダウンロードされた依存関係や、以前にビルドされたアーティファクトなどを再利用します。

S2I の利点には以下が含まれます。

イメージの柔軟性	S2I スクリプトを作成して、アプリケーションコードをほとんどすべての既存の Docker 形式コンテナーに挿入し、既存のエコシステムを活用することができます。現時点で S2I は **tar** を使用してアプリケーションソースを挿入するため、イメージは tar が実行されたコンテンツを処理できる必要があることに注意してください。
速度	S2I の場合、アセンブルプロセスは、各手順で新規の層を作成せずに多数の複雑な操作を実行でき、これによりプロセスが高速になります。さらに、S2I スクリプトを作成すると、ビルドが実行されるたびにダウンロードまたはビルドを実行することなく、アプリケーションイメージの以前のバージョンに保存されたアーティファクトを再利用できます。
パッチ容易性 (Patchability)	S2I により、基礎となるイメージがセキュリティー上の問題でパッチを必要とする場合にアプリケーションを一貫して再ビルドできるようになります。
運用効率	**Dockerfile** が許可するように任意のアクションを実行する代わりにビルド操作を制限することで、PaaS オペレーターはビルドシステムの意図しないまたは意図した誤用を避けることができます。

運用上のセキュリティー	任意の **Dockerfile** をビルドすると、root の権限昇格のためにホストシステムを公開します。これは、Docker ビルドプロセス全体が Docker 権限を持つユーザーとして実行されるため、悪意あるユーザーが悪用する可能性があります。S2I は root ユーザーとして実行される操作を制限し、スクリプトを root 以外のユーザーとして実行できます。
ユーザー効率	S2I は開発者が任意の **yum install** タイプの操作を実行することを防ぐため、アプリケーションのビルド時の開発の反復スピードを低下させる可能性があります。
エコシステム	S2I により、アプリケーションのベストプラクティスを利用できるイメージの共有されたエコシステムが促進されます。
再現性	生成されるイメージには、特定バージョンのビルドツールおよび依存関係などのすべての入力が含まれる可能性があります。これにより、イメージを正確に再現できます。

3.5.1.3. カスタムビルド

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

「カスタムビルダーイメージ」は、RPM またはベースイメージのビルドなど、ビルドプロセスのロジックを使って組み込まれたプレーンな Docker 形式のコンテナーイメージです。**openshift/origin-custom-docker-builder** イメージは、カスタムビルダーイメージの実装例として Docker Hub レジストリーで利用できます。

3.5.1.4. Pipeline ビルド

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

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

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

Jenkins サーバーのデプロイ方法や自動プロビジョニングの設定または無効にする方法についての詳細は、「Pipeline 実行の設定」を参照してください。

注記

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

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

3.5.2. イメージストリーム

イメージストリームおよびその関連付けられたタグは、OpenShift Container Platform 内で Docker イ

メージを参照するための抽象化を提供します。イメージストリームとそのタグを使用して、利用可能なイメージを確認し、リポジトリーのイメージが変更される場合でも必要な特定のイメージを使用していることを確認できます。

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

ビルドおよびデプロイメントをそれぞれ実行して、新規イメージ追加時の通知がないか、イメージストリームを監視して対応できるように、「ビルド」および「デプロイメント」を設定することができます。

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

デプロイメントまたはビルドで使用するイメージストリームタグが更新されない場合には、Docker レジストリーの Docker イメージが更新されても、ビルドまたはデプロイメントは以前の (既知でおそらく適切であると予想される) イメージをそのまま使用します。

ソースイメージは以下のいずれかに保存できます。

- OpenShift Container Platform の統合レジストリー
- たとえば、外部レジストリーの **registry.access.redhat.com** または **hub.docker.com**
- OpenShift Container Platform クラスターの他のイメージストリーム

(ビルドまたはデプロイメント設定などの) イメージストリームタグを参照するオブジェクトを定義する場合には、Docker リポジトリーではなく、イメージストリームタグを参照します。アプリケーションのビルドまたはデプロイ時に、OpenShift Container Platform がこのイメージストリームタグを使用して Docker リポジトリーにクエリーを送信して、対象のイメージに関連付けられた ID を特定して、正確なイメージを使用します。

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

以下のイメージストリームには、Python v3.4 イメージをポイントする **34** と、Python v3.5 イメージをポイントする **35** の 2 つのタグが含まれます。

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

34
  tagged from centos/python-34-centos7

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

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

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

イメージストリームの使用には、いくつかの大きな利点があります。

- コマンドラインを使用して再プッシュすることなく、タグ付けや、タグのロールバック、およびイメージの迅速な処理を実行できます。
- 新規イメージがレジストリーにプッシュされると、ビルドおびデプロイメントをトリガーできます。また、OpenShift Container Platform には他のリソースの汎用トリガーがあります (Kubernetes オブジェクトなど)。
- 「定期的な再インポート用のタグをマーク付する」こともできます。ソースイメージが変更されると、その変更は選択され、イメージストリームに反映されます。これにより、ビルドたはデプロイメント設定に応じてビルドおよび/またはデプロイメントフローがトリガーされます。
- 詳細なアクセス制御を使用してイメージを共有し、チーム間でイメージを迅速に分散できます。
- ソースイメージが変更されると、イメージストリームタグはイメージの既知の適切なバージョンをポイントしたままになり、アプリケーションが予期せずに損傷しないようにします。
- イメージストリームオブジェクトのパーミッションを使用して、イメージを閲覧し、使用できるユーザーについてセキュリティー設定を行うことができます。
- クラスターレベルでイメージを読み込んだり、一覧表示するパーミッションのないユーザーは、イメージストリームを使用してプロジェクトでタグ付けされたイメージを取得できます。

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

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

- イメージストリームタグが Docker イメージタグをポイントする場合、Docker イメージタグの更新方法を理解する必要があります。たとえば、Docker イメージタグ **docker.io/ruby:2.4** は v2.4 ruby イメージを常にポイントするとします。しかし、Docker イメージタグ **docker.io/ruby:latest** はメジャーバージョンで変更させる可能性があります。そのため、イメージストリームタグがポイントする Docker イメージタグは、これを参照することを選択している場合はイメージストリームタグの安定度を示します。
- イメージストリームタグが別のイメージストリームに従う場合 (これが Docker イメージタグを直接ポイントしない)、イメージストリームタグが今後別のイメージストリームタグに従うように更新される可能性があります。また、これにより、互換性のないバージョンの変更が選択される可能性があります。

3.5.2.1. 重要な用語

Docker リポジトリー

関連する Docker イメージおよびそれらを識別するタグのコレクションです。たとえば、OpenShift Jenkins イメージは Docker リポジトリーにあります。

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

Docker レジストリー

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

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

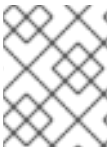
Docker イメージ

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

Docker イメージタグ

特定のイメージを差異化するリポジトリーの Docker イメージに適用されます。たとえば、ここでは **3.6.0** はタグです。

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



注記

新規の Docker イメージコンテンツにいつでもポイントするように更新できる Docker イメージタグです。

Docker イメージ ID

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

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

注記

SHA イメージ ID は変更できません。特定の SHA ID は同一の Docker イメージコンテンツを常に参照します。

イメージストリーム

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

イメージストリームタグ

イメージストリーム内のイメージへの名前付きポインターです。イメージストリームタグは Docker イメージタグに似ています。以下の「イメージストリームタグ」を参照してください。

イメージストリームイメージ

タグ付けされている特定のイメージストリームから特定の Docker イメージを取得できるようにするイメージです。イメージストリームのイメージは、特定のイメージの SHA ID についてのメタデータをプルする API リソースオブジェクトです。以下の「イメージストリームのイメージ」を参照してください。

イメージストリームトリガー

イメージストリームタグの変更時に特定のアクションを生じさせるトリガーです。たとえば、インポートにより、タグの値が変更され、これにより Deployments、Builds またはそれらをリッスンす

る他のリソースがある場合にトリガーが実行されます。以下の「イメージストリームトリガー」を参照してください。

3.5.2.2. イメージストリームの設定

イメージストリームオブジェクトには以下の要素が含まれます。

注記

イメージおよびイメージストリームの管理についての詳細は、『開発者ガイド』を参照してください。

イメージストリームオブジェクト定義

```
apiVersion: v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  creationTimestamp: 2017-09-29T13:33:49Z
  generation: 1
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample (1)
  namespace: test
  resourceVersion: "633"
  selflink: /oapi/v1/namespaces/test/imagestreams/origin-ruby-sample
  uid: ee2b9405-c68c-11e5-8a99-525400f25e34
spec: {}
status:
  dockerImageRepository: 172.30.56.218:5000/test/origin-ruby-sample (2)
  tags:
  - items:
    - created: 2017-09-02T10:15:09Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d (3)
      generation: 2
      image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5 (4)
    - created: 2017-09-29T13:40:11Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
      generation: 1
      image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
    tag: latest (5)
```

(1) イメージストリームの名前です。

(2) 新規イメージをこのイメージストリームで追加/更新するためにプッシュできる Docker リポジトリーパスです。

(3) イメージストリームが現在参照する SHA ID です。このイメージストリームタグを参照するリソースはこの ID を使用します。

4 このイメージストリームタグが以前に参照した SHA ID です。古いイメージにロールバックするために使用できます。

5 イメージストリームタグ名です。

イメージストリームを参照するビルド設定のサンプルについては、設定の **Strategy** スタンザで「BuildConfig の内容」を参照してください。

イメージストリームを参照するデプロイメント設定のサンプルについては、設定の **Strategy** スタンザで「デプロイメント設定の作成」の部分を参照してください。

3.5.2.3. イメージストリームイメージ

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

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

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

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

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

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

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

3.5.2.4. イメージストリームタグ

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

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

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

履歴の 2 つのイメージを持つイメージストリームタグ

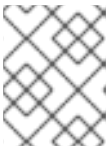
```
tags:
- items:
```

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

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

- **永続タグ** は、Python 3.5 などの特定バージョンのイメージをポイントするバージョン固有のタグです。

- **追跡タグ** は別のイメージストリームタグに従う参照タグで、シンボリックリンクなどのように、フォローするイメージを変更するために今後更新される可能性があります。このような新規レベルでは後方互換性が確保されない点に注意してください。
  たとえば、OpenShift Container Platform に同梱する **latest** イメージストリームタグは追跡タグです。これは、**latest** イメージストリームタグのコンシューマーが新規のレベうが利用可能になるとイメージで提要されるフレームワークの最新レベルに更新されることを意味します。**v3.6** への **latest** イメージストリームタグはいつでも **v3.7** に変更される可能性があります。これらの **latest** イメージストリームタグは Docker **latest** タグと異なる動作をすることに注意してください。この場合、**latest** イメージストリームタグは Docker リポジトリーの最新イメージをポイントしません。これは、イメージの最新バージョンでない可能性のある別のイメージストリームタグをポイントします。たとえば、**3.3** バージョンのリリース時に、**latest** イメージストリームタグがイメージの **v3.2** をポイントする場合、**latest** タグは **v3.3** に自動的に更新されず、これが **v3.3** イメージストリームタグをポイントするように手動で更新されるまで **v3.2** のままになります。



注記

追跡タグは単一のイメージストリームに制限され、他のイメージストリームを参照できません。

それぞれのニーズに合わせて独自のイメージストリームタグを作成できます。「推奨されるタグ付け規則」を参照してください。

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

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

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

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

3.5.2.5. イメージストリーム変更トリガー

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

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

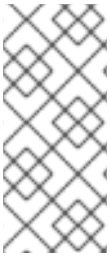
ImageChange トリガーにより、イメージストリームタグの変更時に常に新規レプリケーションコントローラーが生成されます (イメージの新規バージョンがプッシュされるタイミング)。

例3.8 ImageChange トリガー

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

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

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

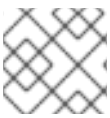


注記

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

3.5.2.6. イメージストリームのマッピング

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



注記

イメージストリームのマッピングの設定は高度な機能です。

この情報は、新規イメージを作成するする際 (すでに存在しない場合) やイメージをイメージストリームにタグ付けする際に使用されます。OpenShift Container Platform は、コマンド、エントリーポイント、および開発変数などの各イメージについての完全なメタデータを保存します。OpenShift Container

Platform のイメージはイミュータブルであり、名前の最大長さは 63 文字です。

注記

イメージの手動のタグ付けの詳細については、『開発者ガイド』を参照してください。

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

イメージストリームマッピングオブジェクト定義

```
apiVersion: v1
kind: ImageStreamMapping
metadata:
  creationTimestamp: null
  name: origin-ruby-sample
  namespace: test
tag: latest
image:
  dockerImageLayers:
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ee1dd2cb6df21971f4af6de0f1d7782b81fb63156801cfde2bb47b4247c23c29
    size: 196634330
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ca062656bff07f18bff46be00f40cfbb069687ec124ac0aa038fd676cfaea092
    size: 177723024
  - name: sha256:63d529c59c92843c395befd065de516ee9ed4995549f8218eac6ff088bfa6b6e
    size: 55679776
  - name: sha256:92114219a04977b5563d7dff71ec4caa3a37a15b266ce42ee8f43dba9798c966
    size: 11939149
  dockerImageMetadata:
    Architecture: amd64
    Config:
      Cmd:
      - /usr/libexec/s2i/run
      Entrypoint:
      - container-entrypoint
      Env:
      - RACK_ENV=production
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
      - OPENSHIFT_BUILD_NAME=ruby-sample-build-1
      - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
```

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

```
      User: "1001"
      WorkingDir: /opt/app-root/src
    Created: 2016-01-29T13:40:00Z
    DockerVersion: 1.8.2.fc21
    Id: 9d7fd5e2d15495802028c569d544329f4286dcd1c9c085ff5699218dbaa69b43
    Parent: 57b08d979c86f4500dc8cad639c9518744c8dd39447c055a3517dc9c18d6fccd
    Size: 441976279
    apiVersion: "1.0"
    kind: DockerImage
  dockerImageMetadataVersion: "1.0"
  dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

3.5.2.7. イメージストリームの使用

以下のセクションでは、イメージストリームおよびイメージストリームタグを使用する方法について説明します。イメージストリームの使用方法についての詳細は、「イメージのマッピング」を参照してください。

3.5.2.7.1. イメージストリームについての情報の取得

イメージストリームについての一般的な情報およびこれがポイントするすべてのタグについての詳細情報を取得するには、以下のコマンドを使用します。

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

以下に例を示します。

```
oc describe is/python

Name:   python
Namespace:  default
Created:  About a minute ago
Labels:   <none>
Annotations:  openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup:  local=false
Unique Images:  1
Tags:   1

3.5
  tagged from centos/python-35-centos7

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

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

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

以下に例を示します。

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

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

注記

表示されている以上の情報が出力されます。

3.5.2.7.2. 追加タグのイメージストリームへの追加

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

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

以下に例を示します。

```
oc tag python:3.5 python:latest

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

oc describe コマンドを使用して、イメージストリームに、外部 Docker イメージをポイントするタグ (**3.5**) と、最初のタグに基づいて作成されているために同じイメージをポイントする別のタグ (**latest**) の 2 つのタグが含まれることを確認します。

```
oc describe is/python

Name:   python
Namespace:  default
Created:  5 minutes ago
Labels:   <none>
Annotations:  openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup:  local=false
Unique Images:  1
Tags:   2

latest
  tagged from python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25

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

```
3.5
  tagged from centos/python-35-centos7

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

3.5.2.7.3. 外部イメージのタグの追加

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

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

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

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

外部イメージのセキュリティーが保護されている場合、そのレジストリーにアクセスするために認証情報を使ってシークレットを作成する必要があります。詳細については、「プライベートレジストリーからのイメージのインポート」を参照してください。

3.5.2.7.4. イメージストリームタグの更新

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

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

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

```
oc tag python:3.6 python:latest
Tag python:latest set to
python@sha256:438208801c4806548460b27bd1fbcb7bb188273d13871ab43f.
```

3.5.2.7.5. イメージストリームタグのイメージストリームからの削除

古いタグをイメージストリームから削除するには、以下を実行します。

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

以下に例を示します。

```
oc tag -d python:3.5

Deleted tag default/python:3.5.
```

3.5.2.7.6. タグの定期的なインポートの設定

外部 Docker レジストリーを使用している場合、イメージを定期的に再インポート (最新セキュリティー更新を取得する場合など) するには、**--scheduled** フラグを使用します。

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

以下に例を示します。

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

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

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

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

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

3.6. DEPLOYMENTS (デプロイメント)

3.6.1. レプリケーションコントローラー

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

レプリケーションコントローラー設定は以下で構成されています。

1. 必要なレプリカ数 (これはランタイム時に調整可能)。
2. レプリケートされた Pod の作成時に使用する Pod 定義。
3. 管理された Pod を識別するためのセレクター。

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

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

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

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

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1 (1)
  selector:   (2)
```

```
    name: frontend
  template:    3
    metadata:
      labels:    4
        name: frontend    5
    spec:
      containers:
      - image: openshift/hello-openshift
        name: helloworld
        ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
```

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

3.6.2. レプリカセット

「レプリケーションコントローラー」と同様に、レプリカセットで、指定数の Pod レプリカが特定の時間実行されるようにします。レプリカセットとレプリケーションコントローラーの相違点は、レプリカセットではセットベースのセレクター要件をサポートし、レプリケーションコントローラーは等価ベースのセレクター要件のみをサポートする点です。

注記

カスタム更新のオーケストレーションが必要な場合や、更新が全く必要のない場合にのみレプリカセットを使用し、それ以外は「デプロイメント」を使用してください。レプリカセットは個別に使用できますが、Pod 作成/削除/更新のオーケストレーションにはデプロイメントでレプリカセットを使用します。デプロイメントは、自動的にレプリカセットを管理し、Pod に宣言の更新を加えるので、作成するレプリカセットを手動で管理する必要はありません。

レプリカセットは、**ReplicaSet** と呼ばれるコアの Kubernetes オブジェクトです。

以下は、**ReplicaSet** 定義のサンプルです。

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend-1
  labels:
    tier: frontend
spec:
  replicas: 3
  selector:    1
```

```
    matchLabels: 2
      tier: frontend
    matchExpressions: 3
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - image: openshift/hello-openshift
        name: helloworld
        ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
```

1 一連のリソースに対するラベルのクエリー。**matchLabels** と **matchExpressions** の結果は論理的に結合されます。

2 セレクターに一致するラベルでリソースを指定する等価ベースのセレクター

3 鍵をフィルターするセットベースのセレクター。これは、**tier** と同等の鍵、**frontend** と同等の値のリソースをすべて選択します。

3.6.3. ジョブ

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

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

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

ジョブの使用方法についての詳細は、「ジョブ」のトピックを参照してください。

3.6.4. デプロイメントおよびデプロイメント設定

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

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

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

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

デプロイメントの作成およびその対話方法についての詳細は、「デプロイメント」を参照してください。

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

```
apiVersion: v1
kind: DeploymentConfig
metadata:
  name: frontend
spec:
  replicas: 5
  selector:
    name: frontend
  template: { ... }
  triggers:
  - type: ConfigChange 1
  - imageChangeParams:
      automatic: true
      containerNames:
      - helloworld
      from:
        kind: ImageStreamTag
        name: hello-openshift:latest
    type: ImageChange 2
  strategy:
    type: Rolling 3
```

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

3.7. テンプレート

3.7.1. 概要

テンプレートは、OpenShift Container Platform で作成されるオブジェクトの一覧を生成するためにパラメーター化され、処理されるオブジェクトのセットを記述します。作成するオブジェクトには、ユーザーがプロジェクト内で作成するパーミッションを持つすべてのものを作成します。たとえば、サービス、ビルド設定、およびデプロイメント設定などがあります。テンプレートは、テンプレートで定義されたすべてのオブジェクトに適用されるラベルのセットも定義します。

テンプレートの作成および使用についての詳細は、「テンプレートについてのガイド」を参照してください。

第4章 追加の概念

4.1. 認証

4.1.1. 概要

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

管理者は、「マスター設定ファイル」を使用して、「認証の設定」を実行できます。

4.1.2. ユーザーとグループ

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

ユーザーは 1 つ以上の **グループ** に割り当てることができます。それぞれのグループはユーザーの特定のセットを表します。グループは、「承認ポリシーを管理」し、パーミッションを一度に複数ユーザーに付与する場合などに役立ちます。この例としては、アクセス権限をユーザーに別々に付与するのではなく、プロジェクト内のオブジェクトへのアクセスを許可する場合などがあります。

明示的に定義されたグループのほかにも、OpenShift で自動的にプロビジョニングされるシステムグループまたは **仮想グループ** があります。このようなグループは、「クラスターのバインディングを表示」すると確認できます。

仮想グループのデフォルトセットでは、とくに以下の点に留意してください。

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

4.1.3. API 認証

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

OAuth アクセストークン

- **<master>/oauth/authorize** および **<master>/oauth/token** エンドポイントを使用して OpenShift Container Platform OAuth サーバーから取得されます。
- **Authorization: Bearer…** ヘッダーとして送信されます。
- OpenShift Container Platform server version 3.6 よりも前のバージョンでは、websocket 要求の **access_token=…** クエリーパラメーターとして送信されます。

- OpenShift Container Platform サーバーのバージョン 3.6 以降で、websocket 要求の **base64url.bearer.authorization.k8s.io.<base64url-encoded-token>** 形式で websocket サブプロトコルヘッダーとして送信されます。

X.509 クライアント証明書

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

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

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

4.1.3.1. 権限の借用

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

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

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

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

ユーザーの代わりにプロジェクトの要求を作成する必要がある場合、**--as=<user> --as-group=<group1> --as-group=<group2>** フラグをコマンドに組み込みます。**system:authenticated:oauth** はプロジェクト要求を作成できる唯一のブートストラップグループであるため、そのグループを以下の例に示されるようになりすます必要があります。

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

4.1.4. OAuth

OpenShift Container Platform マスターには、ビルトイン OAuth サーバーが含まれます。ユーザーは OAuth アクセストークンを取得して、API に対して認証を実行します。

新規の OAuth トークンが要求されると、OAuth サーバーは設定された「アイデンティティープロバイダー」を使用して要求したユーザーのアイデンティティーを判別します。

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

4.1.4.1. OAuth クライアント

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

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

追加のクライアントを登録するには、以下を実行します。

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

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

4.1.4.2. OAuth クライアントとしてのサービスアカウント

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

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

サービスアカウントを OAuth クライアントとして使用する場合:

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

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

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

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

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

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

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

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

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

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

以下に例を示します。

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

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

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

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

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

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

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

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

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

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

注記

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

上記の構文のいずれの組み合わせも、以下のフォーマットを使って実行できます。

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

同じオブジェクトを複数回参照して、柔軟性を向上することができます。

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

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

必要な動作を得るために、静的で動的なアノテーションを同時に使用できます。

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

4.1.4.3.1. OAuth の API イベント

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

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

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

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

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

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

以下は生じる可能性のあるイベントエラーの一覧です。

No redirect URI annotations or an invalid URI is specified

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

Invalid route specified

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

Invalid reference type specified

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

Missing SA tokens

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

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

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

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

      ```
      vi serviceaccount.yaml
      ```

      以下のサンプルコードを追加します。

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

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

      ```
      vi route.yaml
      ```

      以下のサンプルコードを追加します。

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

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

```
vi proxysidecar.yaml
```

以下のサンプルコードを追加します。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: proxy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: proxy
  template:
    metadata:
      labels:
        app: proxy
    spec:
      serviceAccountName: proxy
      containers:
      - name: oauth-proxy
        image: openshift3/oauth-proxy
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8443
          name: public
        args:
        - --https-address=:8443
        - --provider=openshift
        - --openshift-service-account=proxy
        - --upstream=http://localhost:8080
        - --tls-cert=/etc/tls/private/tls.crt
        - --tls-key=/etc/tls/private/tls.key
        - --cookie-secret=SECRET
        volumeMounts:
        - mountPath: /etc/tls/private
          name: proxy-tls

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

d. オブジェクトを作成します。

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

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

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

3. OAuth ログでサービスを確認し、サーバーエラーを見つけます。

```
承認サーバーに、要求を満たすことを阻む予期しないエラーが発生しました。
```

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

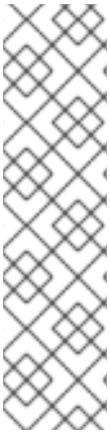
```
oc get events | grep ServiceAccount

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

4.1.4.4. 統合

OAuth トークンのすべての要求には、**<master>/oauth/authorize** への要求が関係します。ほとんどの認証統合は、認証プロキシーをこのエンドポイントの前に配置するか、または OpenShift Container Platform を、サポートする「アイデンティティープロバイダー」に対して認証情報を検証するように設定します。**<master>/oauth/authorize** の要求は、CLI などの対話式ログインページを表示できないユーザーエージェントから送られる場合があります。そのため、OpenShift Container Platform は、対話式ログインフローのほかにも **WWW-Authenticate** チャレンジを使用した認証をサポートします。

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



注記

ブラウザークライアントに対するクロスサイトリクエストフォージェリー (CSRF) 攻撃を防止するため、基本的な認証チャレンジは **X-CSRF-Token** ヘッダーが要求に存在する場合にのみ送信されます。基本的な **WWW-Authenticate** チャレンジを受信する必要があるクライアントでは、このヘッダーを空でない値に設定する必要があります。

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

4.1.4.5. OAuth サーバーメタデータ

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

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

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

```
{
  "issuer": "https://<master>", (1)
  "authorization_endpoint": "https://<master>/oauth/authorize", (2)
  "token_endpoint": "https://<master>/oauth/token", (3)
  "scopes_supported": [ (4)
    "user:full",
    "user:info",
    "user:check-access",
    "user:list-scoped-projects",
    "user:list-projects"
  ],
  "response_types_supported": [ (5)
    "code",
    "token"
  ],
  "grant_types_supported": [ (6)
    "authorization_code",
    "implicit"
  ],
  "code_challenge_methods_supported": [ (7)
    "plain",
    "S256"
  ]
}
```

(1) **https** スキームを使用し、クエリーまたはフラグメントコンポーネントがない承認サーバーの発行者 IDです。これは、承認サーバーについての情報が含まれる **.well-known** RFC 5785 リソースが公開される場所です。

(2) 承認サーバーの承認エンドポートの URL です。RFC 6749 を参照してください。

(3) 承認サーバーのトークンエンドポイントの URL です。RFC 6749 を参照してください。

(4) この承認サーバーがサポートする OAuth 2.0 RFC 6749 スコープの値の一覧を含む JSON 配列です。サポートされるスコープの値すべてが公開される訳ではないことに注意してください。

(5) この承認サーバーがサポートする OAuth 2.0 **response_type** 値の一覧を含む JSON 配列です。使用される配列の値は、RFC 7591 の OAuth 2.0 Dynamic Client Registration Protocol で定義される **response_types** パラメーターで使用されるものと同じです。

(6) この承認サーバーがサポートする OAuth 2.0 grant type の値の一覧が含まれる JSON 配列です。使用される配列の値は、RFC 7591 の **OAuth 2.0 Dynamic Client Registration Protocol** で定義される **grant_types** パラメーターで使用されるものと同じです。

(7) この承認サーバーでサポートされる PKCE RFC 7636 コードのチャレンジメソッドの一覧が含まれる JSON 配列です。コードのチャレンジメソッドの値は、RFC 7636 のセクション 4.3 で定義される **code_challenge_method** パラメーターで使用されます。有効なコードのチャレンジメソッ

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

4.1.4.6. OAuth トークンの取得

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

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

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

4.1.4.7. Prometheus の認証メトリクス

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

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

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

4.2. 承認

4.2.1. 概要

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

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

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

承認は以下を使用して管理されます。

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

クラスター管理者は、「CLI を使用」して、ルール、ロールおよびバインディングを可視化できます。

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

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

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

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

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

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


Name:  basic-user
Labels:  <none>
Annotations: openshift.io/description=A user that can get basic information about projects.
  rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources      Non-Resource URLs Resource Names Verbs
  ---------      ----------------- -------------- -----
  clusterroles      []   []  [get list]
  clusterroles.authorization.openshift.io  []   []  [get list]
  clusterroles.rbac.authorization.k8s.io  []   []  [get list watch]
  projectrequests     []   []  [list]
  projectrequests.project.openshift.io   []   []  [list]
  projects      []   []  [list watch]
  projects.project.openshift.io    []   []  [list watch]
  selfsubjectaccessreviews.authorization.k8s.io  []   []  [create]
  selfsubjectrulesreviews    []   []  [create]
  selfsubjectrulesreviews.authorization.openshift.io []   []  [create]
```

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

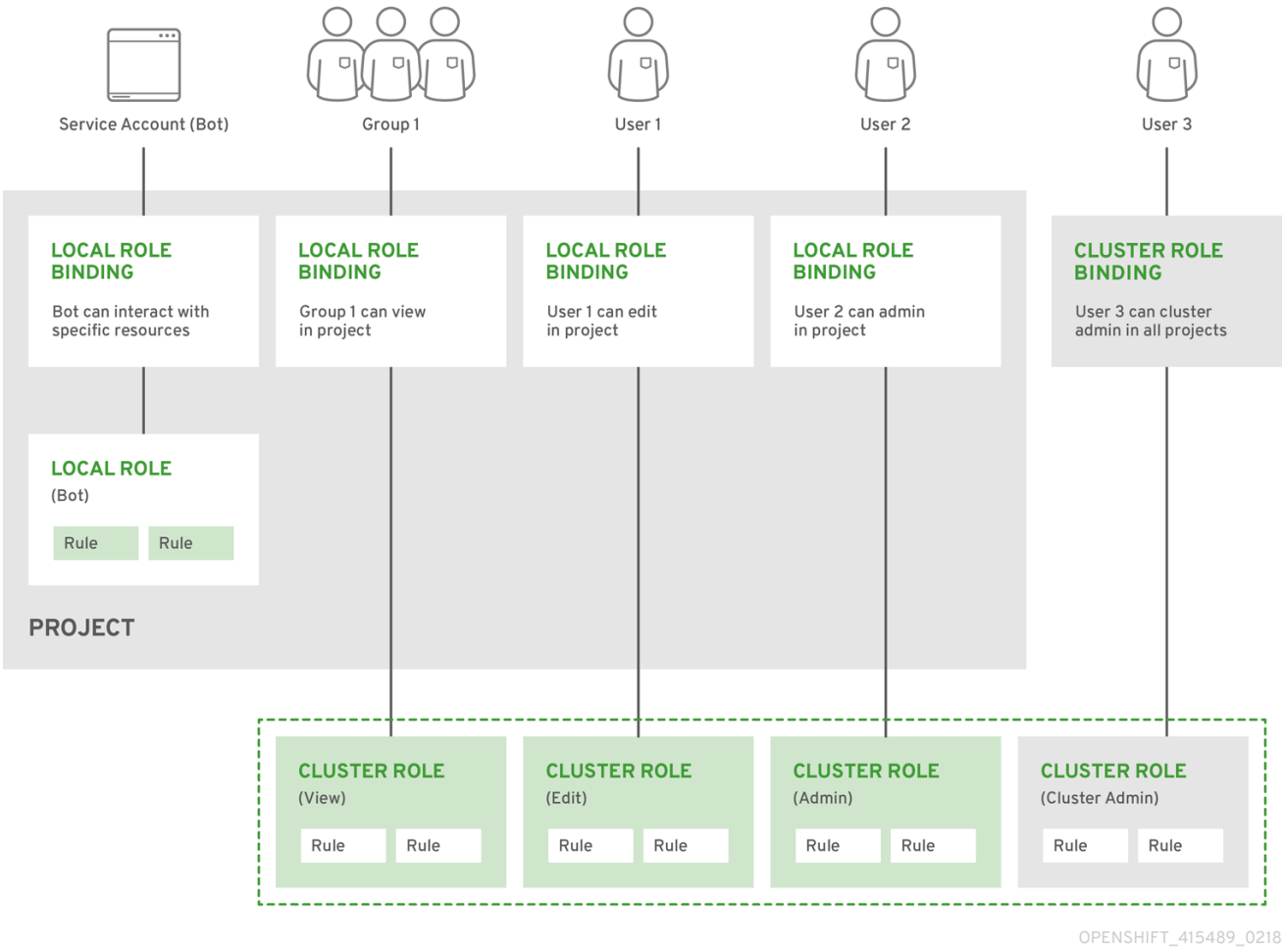
ローカルロールバインディングを表示して得られる以下の概要は、各種のユーザーおよびグループにバインドされた上記のロールを示しています。

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

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


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

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



4.2.2. 承認の評価

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

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

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

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

4.2.3. クラスターおよびローカル RBAC

承認を制御する 2 つのレベルの RBAC ロールおよびバインディングがあります。

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

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

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

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

4.2.4. クラスターロールおよびローカルロール

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

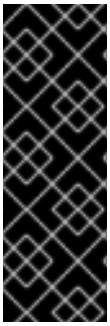
デフォルトのクラスターロール	説明
admin	プロジェクトマネージャーです。ローカルバインディングで使用されている場合、admin ユーザーにはプロジェクトのリソースを閲覧し、クォータを除くプロジェクトのすべてのリソースを変更する権限があります。
basic-user	プロジェクトおよびユーザーについての基本的な情報を取得できるユーザーです。

デフォルトのクラスターロール	説明
cluster-admin	どのプロジェクトでもすべてのアクションを実行可能なスーパーユーザー。ローカルバインディングでバインドされているユーザーの場合は、対象のプロジェクトに含まれる全リソースのクォータや全アクションに対して「フルコントロール」があります。
cluster-status	基本的なクラスターのステータス情報を取得できるユーザーです。
edit	プロジェクトのほとんどのプロジェクトを変更できるが、ロールまたはバインディングを表示したり、変更したりする機能を持たないユーザーです。
self-provisioner	独自のプロジェクトを作成できるユーザーです。
view	変更できないものの、プロジェクトでほとんどのオブジェクトを確認できるユーザーです。それらはロールまたはバインディングを表示したり、変更したりできません。

ヒント

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

プロジェクト管理者は、CLI を使用してそれぞれが関連付けられる動詞およびリソースのマトリックスなど、ロールを可視化し、 「ローカルロールやバインディングを表示」できます。



重要

プロジェクト管理者にバインドされるクラスターロールは、「ローカルバインディング」によってプロジェクトに制限されます。これは、**cluster-admin** または **system:admin** に付与されるクラスターロールの場合のように「クラスター全体」でバインドされる訳ではありません。

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

「プロジェクトのローカルロールの作成方法についてはこちら」を参照してください。

4.2.4.1. クラスターロールの更新

「OpenShift Container Platform のクラスターをアップグレードした後に」、デフォルトのロールが更新され、サーバーの起動時に自動調整されます。調整時に、デフォルトのロールで足りないパーミッションは追加されます。ロールに別途パーミッションを追加していた場合には、削除されます。

デフォルトのロールをカスタマイズし、自動的にロールを調節されないように設定していた場合には、OpenShift Container Platform のアップグレード時に、手動でポリシー定義を更新する必要があります。

4.2.4.2. カスタムロールおよびパーミッションの適用

カスタムロールおよびパーミッションを追加するか、または更新するには、以下のコマンドを使用することを強く推奨します。

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

このコマンドは、他のクライアントを切断しない方法で新規パーミッションが適切に適用されるようにします。

4.2.4.3. クラスターロールの集計

デフォルトのクラスターの管理、編集および表示ロールは、クラスターロールの集計をサポートします。ここでは、各ロールのクラスタールールは新規ルールの作成時に動的に更新されます。この機能は、「カスタムリソースを作成」して Kubernetes API を拡張する場合にのみ該当します。

「クラスターロールの集計の使用方法についてはこちら」を参照してください。

4.2.5. SCC (Security Context Constraints)

ユーザーの実行できる内容を制御する「RBAC リソース」のほかに、OpenShift Container Platform には Pod が実行できる内容および Pod がアクセスできる内容を制御する **SCC (security context constraints)** が含まれます。管理者は CLI を使用して SCC を管理することができます。

SCC は永続ストレージへのアクセスを管理する場合にも非常に便利です。

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

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

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

```
$ oc get scc
NAME            PRIV    CAPS    SELINUX    RUNASUSER        FSGROUP     SUPGROUP
PRIORITY   READONLYROOTFS   VOLUMES
anyuid          false   []      MustRunAs   RunAsAny        RunAsAny    RunAsAny    10      false
```

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

重要

デフォルトの SCC を変更しないでください。デフォルトの SCC をカスタマイズすると、OpenShift Container Platform のアップグレード時に問題が発生する可能性があります。その代わりに、「新規 SCC を作成」します。

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

```
# oc export scc/privileged
allowHostDirVolumePlugin: true
allowHostIPC: true
allowHostNetwork: true
allowHostPID: true
allowHostPorts: true
allowPrivilegedContainer: true
allowedCapabilities: 1
- '*'
apiVersion: v1
defaultAddCapabilities: [] 2
fsGroup: 3
  type: RunAsAny
groups: 4
- system:cluster-admins
- system:nodes
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: 'privileged allows access to all privileged and host
      features and the ability to run as any user, any group, any fsGroup, and with
      any SELinux context.  WARNING: this is the most relaxed SCC and should be used
      only for cluster administration. Grant with caution.'
  creationTimestamp: null
  name: privileged
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: [] 5
runAsUser: 6
  type: RunAsAny
seLinuxContext: 7
```

```
  type: RunAsAny
seccompProfiles:
- '*'
supplementalGroups: 8
  type: RunAsAny
users: 9
- system:serviceaccount:default:registry
- system:serviceaccount:default:router
- system:serviceaccount:openshift-infra:build-controller
volumes:
- '*'
```

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

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

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

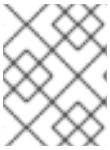
特権付き SCC:

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

- Pod が FSGroup を使用することを許可します。
- Pod が補助グループを使用することを許可します。
- Pod が seccomp プロファイルを使用することを許可します。
- Pod が任意の機能を要求することを許可します。

制限付き SCC:

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



注記

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

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

ブール値による制御	このタイプのフィールドはデフォルトで最も制限のある値に設定されます。たとえば、**AllowPrivilegedContainer** は指定されていない場合は、**false** に常に設定されます。
許可されるセットによる制御	このタイプのフィールドはセットに対してチェックされ、それらの値が許可されることを確認します。
ストラテジーによる制御	値を生成するストラテジーを持つ項目は以下を提供します。 ● 値を生成するメカニズム ● 指定された値が許可される値のセットに属するようにするメカニズム

4.2.5.1. SCC ストラテジー

4.2.5.1.1. RunAsUser

1. **MustRunAs** - **runAsUser** が設定されることを要求します。デフォルトで設定済みの **runAsUser** を使用します。設定済みの **runAsUser** に対して検証します。
2. **MustRunAsRange** - 事前に割り当てられた値を使用していない場合に、最小および最大値が定義されることを要求します。デフォルトでは最小値を使用します。許可される範囲全体に対して検証します。

3. **MustRunAsNonRoot** - Pod がゼロ以外の **runAsUser** で送信されること、または **USER** ディレクティブをイメージに定義することを要求します。デフォルトは指定されません。

4. **RunAsAny** - デフォルトは指定されません。**runAsUser** の指定を許可します。

4.2.5.1.2. SELinuxContext

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

2. **RunAsAny** - デフォルトは指定されません。**seLinuxOptions** の指定を許可します。

4.2.5.1.3. SupplementalGroups

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

2. **RunAsAny** - デフォルトは指定されません。**supplementalGroups** の指定を許可します。

4.2.5.1.4. FSGroup

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

2. **RunAsAny** - デフォルトは指定されません。**fsGroup** ID の指定を許可します。

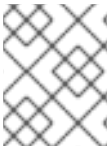
4.2.5.2. ボリュームの制御

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

- azureFile
- azureDisk
- flocker
- flexVolume
- hostPath
- emptyDir
- gcePersistentDisk
- awsElasticBlockStore
- gitRepo
- secret
- nfs

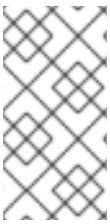
- iscsi
- glusterfs
- persistentVolumeClaim
- rbd
- cinder
- cephFS
- downwardAPI
- fc
- configMap
- vsphereVolume
- quobyte
- photonPersistentDisk
- projected
- portworxVolume
- scaleIO
- storageos
- * (すべてのボリュームタイプの使用を許可する特殊な値)
- none (すべてのボリュームタイプの使用を無効にする特殊な値。後方互換の場合にのみ存在します)

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



注記

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



注記

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

4.2.5.3. FlexVolume へのアクセスの制限

OpenShift Container Platform は、それらのドライバーに基づいて FlexVolume の追加の制御を提供します。SCC が FlexVolume の使用を許可する場合、Pod は任意の FlexVolume を要求できます。ただ

し、クラスター管理者が **AllowedFlexVolumes** フィールドでドライバー名を指定する場合、Pod はこれらのドライバーでのみ FlexVolumes を使用する必要があります。

アクセスを 2 つの FlexVolume のみに制限する例

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

4.2.5.4. Seccomp

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

カスタムプロファイルの設定および使用についての詳細は、「seccomp についての説明」を参照してください。

4.2.5.5. 受付

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

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

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

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

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

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

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

注記

これらの例は、事前に割り当てられる値を使用するストラテジーに関連するものです。

MustRunAs の FSGroup SCC ストラテジー

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

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

MustRunAs の SupplementalGroups SCC ストラテジー

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

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

4.2.5.5.1. SCC の優先度設定

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

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

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

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

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

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

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

4. MustRunAs の **SupplementalGroups** ストラテジーです。受付は **openshift.io/sa.scc.supplemental-groups** アノテーションを検索します。

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

1. **RunAsAny** および **MustRunAsNonRoot** ストラテジーはデフォルトの値を提供しません。そのため、Pod が定義されるフィールドを必要とする場合 (グループ ID など)、このフィールドは Pod 仕様内に定義する必要があります。

2. **MustRunAs** (単一の値) ストラテジーは、常に使用されるデフォルト値を提供します。たとえば、グループ ID の場合、Pod 仕様が独自の ID 値を定義する場合でも、namespace のデフォルトフィールドが Pod のグループに表示されます。

3. **MustRunAsRange** および **MustRunAs** (範囲ベース) ストラテジーは、範囲の最小値を提供します。単一値の **MustRunAs** ストラテジーの場合のように、namespace のデフォルト値は実行中の Pod に表示されます。範囲ベースのストラテジーが複数の範囲で設定可能な場合、これは最初に設定された範囲の最小値を指定します。

注記

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

注記

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

注記

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

4.2.6. 認証済みのユーザーとして何が実行できるのかを判断する方法

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

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

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

ユーザーが判読可能な形式で情報を取得し直すには、以下を実行します。

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

この出力により、詳細な一覧が表示されます。

特定の動詞が実行可能かどうかを判断するには、以下を実行します。

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

「ユーザースコープ」では、指定のスコープについて詳しく説明しています。以下に例を示します。

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

4.3. 永続ストレージ

4.3.1. 概要

ストレージの管理は、コンピュートリソースの管理とは異なります。OpenShift Container Platform は Kubernetes 永続ボリューム (PV) フレームワークを使用してクラスター管理者がクラスターの永続ストレージのプロビジョニングを実行できるようにします。開発者は、Persistent Volume Claims (PVC、永続ボリューム要求) を使用して、基礎となるストレージインフラストラクチャーについての特定の知識なしに PV リソースを要求できます。

PVC はプロジェクト固有のもので、開発者が PV を使用する手段として作成し、使用します。PV リソース自体はいずれの単一プロジェクトにもスコープ設定されず、それらは OpenShift Container Platform クラスターで共有でき、任意のプロジェクトから要求できますが、PV が PVC に バインド された後は、その PV は追加の PVC にバインドできなくなります。これにはバインドされた PV を単一の namespace (バインディングプロジェクトの namespace) にスコープ設定する作用があります。

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

重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

PVC は、開発者によるストレージの要求を表す **PersistentVolumeClaim** API オブジェクトによって定義されます。これは Pod がノードリソースを消費する点で Pod に似ており、PVC は PV リソースを消費します。たとえば、Pod は特定のレベルのリソース (CPU およびメモリーなど) を要求し、PVC は特定のストレージ容量およびアクセスモードを要求できます (たとえば、それらは読み取り/書き込みで 1 回、読み取り専用で複数回マウントできます)。

4.3.2. ボリュームおよび要求のライフサイクル

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

4.3.2.1. ストレージのプロビジョニング

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

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

4.3.2.2. 要求のバインド

PVC の作成時に、ストレージの特定容量の要求、必要なアクセスモードの指定のほか、ストレージクラスを作成してストレージの記述や分類を行います。マスターのコントロールループは新規 PVC の有無を監視し、新規 PVC を適切な PV にバインドします。適切な PV がない場合には、ストレージクラスのプロビジョナーが PV を作成します。

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

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

4.3.2.3. Pod および要求した PV の使用

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

要求が存在し、その要求がバインドされている場合、バインドされた PV は必要な限り所属させることができます。Pod のスケジュールおよび要求された PV のアクセスは、**persistentVolumeClaim** を Pod のボリュームブロックに組み込んで実行できます。構文の詳細については、以下を参照してください。

4.3.2.4. PVC 保護

デフォルトでは、PVC 保護は有効になっています。

4.3.2.5. ボリュームの開放

ボリュームの処理が終了したら、API から PVC オブジェクトを削除できます。これにより、リソースをまた要求できるようになります。ボリュームは要求の削除時に「開放 (リリース)」されたものとみなされますが、別の要求で利用できる状態にはありません。以前の要求者のデータはボリューム上に残るので、ポリシーに基づいて処理される必要があります。

4.3.2.6. ボリュームの回収

PersistentVolume の回収ポリシーは、クラスターに対してリリース後のボリュームの処理方法について指示します。ボリュームの回収ポリシーは、**Retained**、**Recycled**、または **Deleted** のいずれかにすることができます。

Retained 回収ポリシーにより、これをサポートするボリュームプラグインについてのリソースの手動による回収を可能にします。**Deleted** 回収ポリシーは、OpenShift Container Platform の **PersistentVolume** オブジェクトと、および AWS EBS、GCE PD、または Cinder ボリュームなどの外部インフラストラクチャーの関連ストレージアセットの両方を削除します。

注記

動的にプロビジョニングされたボリュームは常に削除されます。

4.3.2.6.1. ボリュームのリサイクル

適切なボリュームプラグインでサポートされる場合、リサイクルはボリュームの基本的なスクラブを実行 (**rm -rf /thevolume/***) し、これを新規の要求で再び利用できるようにします。

警告

OpenShift Container Platform 3.6 で導入された動的プロビジョニングが優先的に使用されるようになったため、**recycle** の回収ポリシーは非推奨になり、削除されました。

「ControllerArguments」のセクションで説明されているように、コントローラーマネージャーのコマンドライン引数を使用してカスタムのリサイクラー Pod テンプレートを設定できます。カスタムのリサイクラー Pod テンプレートには、以下の例のような **volumes** 仕様が含まれます。

カスタムのリサイクラー Pod テンプレート例

```
apiVersion: v1
kind: Pod
metadata:
  name: pv-recycler-
  namespace: openshift-infra (1)
spec:
  restartPolicy: Never
  serviceAccount: pv-recycler-controller (2)
  volumes:
  - name: nfsvol
    nfs:
      server: any-server-it-will-be-replaced (3)
      path: any-path-it-will-be-replaced (4)
  containers:
  - name: pv-recycler
    image: "gcr.io/google_containers/busybox"
    command: ["/bin/sh", "-c", "test -e /scrub && rm -rf /scrub/..?* /scrub/.[!.]* /scrub/*  && test -z \"$(ls -A /scrub)\" || exit 1"]
    volumeMounts:
    - name: nfsvol
      mountPath: /scrub
```

(1) リサイクラー Pod が実行される namespace です。**openshift-infra** には、ボリュームをリサイクルできる **pv-recycler-controller** サービスアカウントがすでに含まれているため、これは推奨される namespace になります。

(2) NFS ボリュームのマウントが許可されるサービスアカウントの名前です。これは指定された namespace に存在している必要があります。**pv-recycler-controller** アカウントは、**openshift-infra** namespace に自動作成され、必要なすべてのパーミッションを含むため、推奨されるアカウ

ントになります。

3 4 カスタムリサイクラー Pod テンプレートの **volumes** 部分に指定される特定の **server** および **path** 値は、リサイクルされる PV の特定の対応する値に置き換えられます。

4.3.3. 永続ボリューム

各 PV には、以下の例のように、ボリュームの仕様およびステータスである **spec** および **status** が含まれます。

PV オブジェクト定義例

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

4.3.3.1. PV の種類

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

- NFS
- HostPath
- GlusterFS
- Ceph RBD
- OpenStack Cinder
- AWS Elastic Block Store (EBS)
- GCE 永続ディスク
- iSCSI
- 「ファイバーチャネル」
- Azure ディスク
- Azure ファイル
- VMWare vSphere
- ローカル

4.3.3.2. 容量

通常、PV には特定のストレージ容量があります。これは PV の **capacity** 属性を使用して設定されます。

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

4.3.3.3. アクセスモード

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

要求は、同様のアクセスモードのボリュームに一致します。一致する条件はアクセスモードとサイズの 2 つの条件のみです。要求のアクセスモードは要求 (request) を表します。そのため、より多くのアクセスを付与することはできますが、アクセスを少なくすることはできません。たとえば、要求により RWO が要求されるものの、利用できる唯一のボリュームが NFS PV (RWO+ROX+RWX) の場合に、要求は RWO をサポートする NFS に一致します。

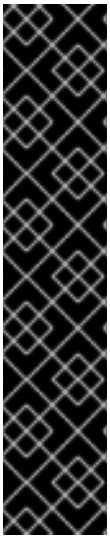
常に直接一致が最初に試行されます。ボリュームのモードは、要求モードと一致するか、要求した内容以上のものを含む必要があります。また、サイズは期待値以上である必要があります。2 つのタイプのボリューム (NFS および iSCSI など) に同じアクセスモードセットがある場合には、いずれかをこれらのモードを含む要求と一致させることができます。ボリュームのタイプの中で順番はなく、どちらかを優先して選択する手段はありません。

同じモードが含まれるボリュームはすべて分類され、サイズ別 (一番小さいものから一番大きいもの順) に分類されます。バインダーは一致するモードのグループを取得し、1 つのサイズが一致するまでそれぞれを (サイズの順序で) 繰り返し処理します。

以下の表では、アクセスモードをまとめています。

表4.1 アクセスモード

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



重要

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

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

iSCSI およびファイバーチャネルボリュームには現在、フェンシングメカニズムがありません。ボリュームが一度に 1 つのノードでのみ使用されるようにする必要があります。ノードのドレイン (解放) などの特定の状況では、ボリュームは 2 つのノードで同時に使用できます。ノードをドレイン (解放) する前に、まずこれらのボリュームを使用する Pod が削除されていることを確認してください。

以下の表では、異なる PV でサポートされるアクセスモードが表示されています。

表4.2 サポート対象の PV 向けアクセスモード

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

ボリュームプラグイン	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
VMWare vSphere	▮	-	-
ローカル	▮	-	-

注記

AWS EBS、GCE 永続ディスクまたは OpenStack Cinder PV に依存する Pod には、「再作成デプロイメントストラテジー」を使用します。

4.3.3.4. 回収ポリシー

以下の表には、現在の回収ポリシーをまとめています。

表4.3 現在の回収ポリシー

回収ポリシー	説明
Retain (保持)	手動による回収
Recycle (リサイクル)	基本的なスクラブ (例: **rm -rf /<volume>/***)

注記

現時点では、NFS および HostPath のみが「リサイクル」回収ポリシーをサポートしています。

警告

OpenShift Container Platform 3.6 で導入された動的プロビジョニングが優先的に使用されるようになったため、**recycle** の回収ポリシーは非推奨になり、削除されました。

4.3.3.5. フェーズ

ボリュームは以下のフェーズのいずれかにあります。

表4.4 ボリュームフェーズ

フェーズ	説明
Available	まだ要求にバインドされていない空きリソースです。

フェーズ	説明
Bound	ボリュームが要求にバインドされています。
Released	要求が検出されていますが、リソースがまだクラスターにより回収されていません。
Failed	ボリュームが自動回収に失敗しています。

CLI には PV にバインドされている PVC の名前が表示されます。

4.3.3.6. マウントオプション

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

以下に例を示します。

マウントオプションの例

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

(1) 指定のマウントオプションは、PV がディスクにマウントされている時に使用されます。

以下の永続ボリュームタイプがマウントオプションをサポートします。

- NFS
- GlusterFS
- Ceph RBD
- OpenStack Cinder
- AWS Elastic Block Store (EBS)

- GCE Persistent Disk
- iSCSI
- Azure Disk
- Azure File
- VMWare vSphere

注記

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

4.3.4. Persistent Volume Claim (永続ボリューム要求、PVC)

各 PVC には、要求の仕様およびステータスである **spec** および **status** が含まれます。以下に例を示します。

PVC オブジェクト定義例

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

4.3.4.1. ストレージクラス

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

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

4.3.4.2. アクセスモード

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

4.3.4.3. リソース

要求は、Pod の場合のようにリソースの特定の数量を要求できます。今回の例では、ストレージに対する要求です。同じリソースモデルがボリュームと要求の両方に適用されます。

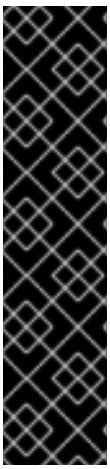
4.3.4.4. ボリュームとしての要求

Pod は要求をボリュームとして使用することでストレージにアクセスします。この要求を使用して、Pod と同じ namespace 内に、要求を共存させる必要があります。クラスターは Pod の namespace で要求を見つけ、これを使用して要求をサポートする **PersistentVolume** を取得します。以下のように、ボリュームはホストにマウントされ、Pod に組み込まれます。

ホストおよび Pod のサンプルへのボリュームのマウント

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

4.3.5. ブロックボリュームのサポート



重要

ブロックボリュームサポートは、テクノロジープレビュー機能で、手動でプロビジョニングされた PV でのみ利用できます。

テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、https://access.redhat.com/ja/support/offerings/techpreview を参照してください。

PV および PVC 仕様に　API フィールドを組み込み、raw ブロックボリュームを静的にプロビジョニングできます。

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

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

PV の例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: block-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: Block 1
  persistentVolumeReclaimPolicy: Retain
  fc:
    targetWWNs: ["50060e801049cfd1"]
    lun: 0
    readOnly: false
```

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

PVC の例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block 1
  resources:
    requests:
      storage: 10Gi
```

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

Pod の仕様例

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

```
    volumeDevices:  1
      - name: data
        devicePath: /dev/xvda  2
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: block-pvc  3
```

1 (**volumeMounts** と同様に) **volumeDevices** は、ブロックデバイスに使用され、**PersistentVolumeClaim** ソースでのみ使用できます。

2 (**mountPath** と同様に) **devicePath** は、物理デバイスへのパスを表します。

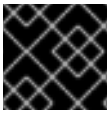
3 ボリュームソースのタイプは **persistentVolumeClaim** であり、想定通りに PVC の名前に一致する必要があります。

表4.5 VolumeMode の許容値

値	デフォルト
Filesystem	Yes
Block	No

表4.6 ブロックボリュームのバインディングシナリオ

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



重要

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

4.4. 一時ローカルストレージ

4.4.1. 概要

注記

このトピックは、一時ストレージのテクノロジープレビューを OpenShift Container Platform 3.10 で有効化した場合にのみ該当します。この機能は、デフォルトでは無効になっています。有効にすると、OpenShift Container Platform クラスターは一時ストレージを使用して、クラスターが破棄された後に、永続する必要のない情報が保存されます。この機能を有効にする方法は、『一時ストレージの設定』を参照してください。

永続ストレージに加え、Pod とコンテナーは、操作に一時または短期的なローカルストレージを必要とする場合があります。この一時ストレージは、個別の Pod の寿命より長くなることはなく、一時ストレージは Pod 間で共有することはできません。

OpenShift Container Platform 3.10 以前は、一時ローカルストレージは、コンテナーの書き込み可能な階層、logs ディレクトリー、EmptyDir ボリュームを使用して Pod に公開されていました。Pod は、スクラッチスペース、キャッシュ、ログに一時ローカルストレージを使用します。ローカルストレージのアカウントや分離がないことに関連する問題には、以下が含まれます。

- Pod では、利用可能なローカルストレージの数が分からない。
- Pod がローカルストレージを要求しても確実に割り当てられない可能性がある。
- ローカルストレージはベストエフォートのリソースである。
- Pod は、他の Pod でローカルストレージがいっぱいになるとエビクトされる可能性があり、十分なストレージが回収されるまで、新しい Pod は入れない。

一時ストレージは、永続ボリュームとは異なり、体系化されておらず、システム、コンテナーランタイム、Openshift Container Platform での他の用途に加え、ノードで実行中のすべての Pod 間で実際のデータではなく領域を共有します。一時ストレージのフレームワークは、Pod が一時的なストレージのニーズを指定するだけでなく、随時 OpenShift Container Platform が Pod をスケジューリングし、ローカルストレージが過剰に使用されないように保護します。

一時ストレージフレームワークでは、管理者および開発者がこのローカルストレージの管理を改善できますが、I/O スループットやレイテンシーに関する確約はありません。

4.4.2. 一時ストレージのタイプ

一時ローカルストレージは常に、プライマリーパーティションで利用できるようになっています。プライマリーパーティション、Root、ランタイムを作成する基本的な方法には 2 種類あります。

4.4.2.1. Root

このパーティションでは、デフォルトで kubelet の root ディレクトリー **/var/lib/origin/** と **/var/log/** ディレクトリーを保持します。このパーティションは、ユーザーの Pod、OS、Kubernetes システムのデーモン間で共有できます。Pod は、EmptyDir ボリューム、コンテナーログ、イメージ階層、コンテ

ナーの書き込み可能な階層を使用して、このパーティションを使用できます。Kubelet はこのパーティションの共有アクセスおよび分離を管理します。このパーティションは一時的なもので、アプリケーションは、このパーティションから、ディスク IOPS などのパフォーマンス SLA は期待できません。

4.4.2.2. ランタイム

これは、ランタイムがオーバーレイファイルシステムに使用可能なオプションのパーティションです。OpenShift Container Platform は、このパーティションの分離および共有アクセスを特定して提供します。このパーティションには、イメージ階層と書き込み可能な階層が含まれます。ランタイムパーティションが存在する場合は、**root** パーティションにはイメージ階層もその他の書き込み可能階層も含まれません。

注記

DeviceMapper を使用してランタイムストレージを提供する場合には、一時ストレージ管理には、コンテナーの Copy-on-Write 階層は含まれません。オーバーレイストレージを使用してこの一時ストレージを監視してください。

4.5. ソースコントロール管理

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

SCM 統合はビルドに密接に関連し、以下の 2 つの点を実行します。

- リポジトリーを使用して **BuildConfig** を作成します。これにより OpenShift Container Platform 内でのアプリケーションのビルドが可能になります。**BuildConfig** を「手動で作成」することも、リポジトリーを検査して OpenShift Container Platform で「自動的に作成」することもできます。
- リポジトリーの変更時のビルドをトリガーします。

4.6. 受付コントローラー

4.6.1. 概要

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

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

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

警告

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

4.6.2. 一般的な受付ルール

3.3 以降で、OpenShift Container Platform は Kubernetes および OpenShift Container Platform リソースの単一の受付チェーンを使用します。これは、別個の受付チェーンが使用されていた 3.2 以前とは異なります。これは、トップレベルの **admissionConfig.pluginConfig** 要素に **kubernetesMasterConfig.admissionConfig.pluginConfig** に含まれていた受付プラグイン設定が含まれることを意味しています。

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

また 3.3 より、サポートされるすべての受付プラグインは単一チェーン内で順序付けられます。**admissionConfig.pluginOrderOverride** または **kubernetesMasterConfig.admissionConfig.pluginOrderOverride** を設定する必要はなくなります。代わりに、プラグイン固有の設定を追加するか、または以下のような **DefaultAdmissionConfig** スタンザを追加してデフォルトでオフになっているプラグインを有効にする必要があります。

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

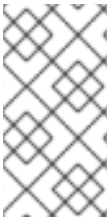
1 受付プラグイン名です。

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

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

警告

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



注記

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

4.6.3. カスタマイズ可能な受付プラグイン

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

- ユーザーあたりのセルフプロビジョニングされたプロジェクト数の制限
- グローバルビルドのデフォルトと上書きの設定
- Pod 配置の制御
- ロールバインディングの管理

4.6.4. コンテナーを使用した受付コントローラー

コンテナーを使用する受付コントローラーも init コンテナーをサポートします。

4.7. カスタム受付コントローラー

4.7.1. 概要

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

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

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

4.7.2. 受付 Webhook

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

設定可能な 2 種類の受付 Webhook オブジェクトがあります。

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

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

重要

受付 Webhook はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポートについての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

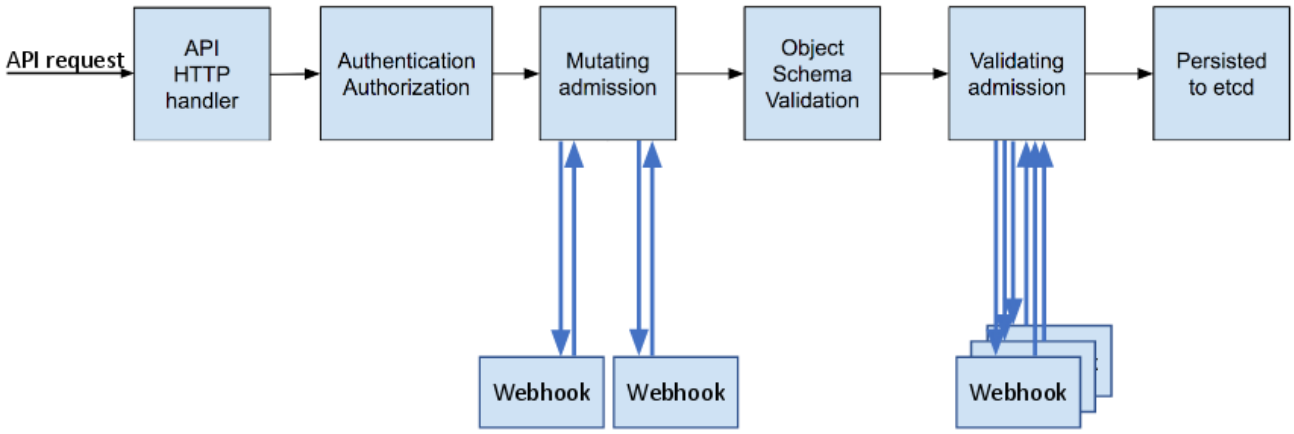
オブジェクトがインスタンス化されると、OpenShift Container Platform は API 呼び出しを実行してオブジェクトを許可します。受付プロセスでは、**変更用の受付コントローラー** は Webhook を呼び出して、アフィニティーラベルの挿入などのタスクを実行します。受付プロセスの終了時に、**検証用の受付コントローラー** は Webhook を呼び出し、アフィニティーラベルの検証などにより、オブジェクトが適切に設定されていることを確認します。検証にパスすると、OpenShift Container Platform はオブジェクトを設定済みとしてスケジュールします。

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

- Webhook の **すべて** が要求を承認する場合、受付チェーンは継続します。
- Webhook の **いずれか** が要求を拒否する場合、受付要求は拒否され、これは、**初回の** webhook の拒否理由に基づいて実行されます。
  複数の Webhook が受付要求を拒否する場合、最初のものだけがユーザーに返されます。
- Webhook の呼び出し時にエラーが生じる場合、その要求は無視され、受付要求を承認/拒否するために使用されます。

受付コントローラーと Webhook サーバー間の通信のセキュリティーは TLS を使用して保護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 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受付コントローラーに提供されます。

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



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

共通のユースケースには以下が含まれます。

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

4.7.2.1. 受付 Webhook のタイプ

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

変更用の受付 Webhook は、受付プロセスの変更フェーズで呼び出されるので、リソースコンテンツが永続化される前にリソースを変更できます。受付 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。

受付 Webhook 設定の変更例:

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

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

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

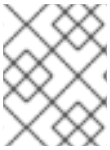
検証用の受付 Webhook は受付プロセスの検証フェーズで起動します。このフェーズでは、特定 API リソースの変更がない項目の実施を可能にし、リソースが再び変更されないようにすることができます。Pod ノードセレクターも、すべての **nodeSelector** フィールドがプロジェクトのノードセレクターの制限で制約されていることを確認する、検証用の受付の例となります。

検証用の受付 Webhook 設定の例:

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

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

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



注記

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

4.7.2.2. 受付 Webhook を作成します。

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

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

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

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

3. 受付 Webhook のフロントエンドサービスを作成します。

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

(1)(2) Webhook をトリガーするための自由形式のラベルです。

4. 以下のコマンドを実行してオブジェクトを作成します。

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

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

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

(1) Webhook をトリガーするためのラベルです。

注記

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

4.7.2.3. 受付 Webhook オブジェクトのサンプル

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

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

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

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

以下は Webhook のフロントエンドサービスです。

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

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

4.8. 他の API オブジェクト

4.8.1. LimitRange

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

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

4.8.2. ResourceQuota

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

ResourceQuota についての詳細は、『クラスター管理』を参照してください。

4.8.3. リソース

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

詳細は、『開発者ガイド』を参照してください。

4.8.4. Secret

「シークレット」は、キー、パスワード、および証明書などの機密情報のストレージです。これらは所定の Pod でアクセスできますが、定義とは別に保持されます。

4.8.5. PersistentVolume

「永続ボリューム」は、クラスター管理者によってプロビジョニングされるインフラストラクチャーのオブジェクト (**PersistentVolume**) です。永続ボリュームは、ステートフルなアプリケーションに対して、耐久性のあるストレージを提供します。

4.8.6. PersistentVolumeClaim

PersistentVolumeClaim オブジェクトは、「Pod 作成者によるストレージの要求」です。Kubernetes は、要求を利用可能なボリュームのプールと照合し、それらをバインドします。この要求は、Pod によりボリュームとして使用されます。Kubernetes は、ボリュームを要求する Pod と同じノードで、そのボリュームが利用可能であることを確認します。

4.8.6.1. カスタムリソース

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

> リンク 「カスタムリソースによる Kubernetes API の拡張」を参照してください。

4.8.7. OAuth オブジェクト

4.8.7.1. OAuthClient

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

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

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

OAuthClient オブジェクト定義

```
kind: "OAuthClient"
accessTokenMaxAgeSeconds: null (1)
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "openshift-web-console" (2)
  selflink: "/oapi/v1/oAuthClients/openshift-web-console"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01Z"
respondWithChallenges: false (3)
secret: "45e27750-a8aa-11e4-b2ea-3c970e4b7ffe" (4)
redirectURIs:
  - "https://localhost:8443" (5)
```

(1) アクセストークンの有効期間 (秒単位)(以下の説明を参照してください)。

(2) **name** は OAuth 要求の **client_id** パラメーターとして使用されます。

(3) **respondWithChallenges** が **true** に設定される場合、**/oauth/authorize** への認証されていない要求は、設定される認証方法でサポートされている場合には **WWW-Authenticate** チャレンジを生じさせます。

(4) **secret** パラメーターの値は、承認コードフローの **client_secret** パラメーターとして使用されます。

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

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

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

4.8.7.2. OAuthClientAuthorization

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

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

OAuthClientAuthorization オブジェクト定義

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

4.8.7.3. OAuthAuthorizeToken

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

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

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

OAuthAuthorizeToken オブジェクト定義

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

```
  name: "MDAwYjM5YjMtMzM1MC00NDY4LTkxODItOTA2OTE2YzE0M2Fj" (1)
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
clientName: "openshift-web-console" (2)
expiresIn: 300 (3)
scopes: []
redirectURI: "https://localhost:8443/console/oauth" (4)
userName: "bob" (5)
userUID: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" (6)
```

(1) **name** は、OAuthAccessToken を交換するために承認コードとして使用されるトークン名を表します。

(2) **clientName** 値は、このトークンを要求した OAuthClient です。

(3) **expiresIn** 値は creationTimestamp の有効期限 (秒単位) です。

(4) **redirectURI** 値は、このトークンが作成された承認フローでユーザーがリダイレクトされた場所です。

(5) **userName** は、このトークンが OAuthAccessToken の取得を許可するユーザーの名前を表します。

(6) **userUID** は、このトークンが OAuthAccessToken の取得を許可するユーザーの UID を表します。

4.8.7.4. OAuthAccessToken

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

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

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

OAuthAccessToken オブジェクト定義

```
kind: "OAuthAccessToken"
apiVersion: "oauth.openshift.io/v1"
metadata:
  name: "ODliOGE5ZmMtYzczYi00Nzk1LTg4MGEtNzQyZmUxZmUwY2Vh" (1)
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:02-00:00"
clientName: "openshift-web-console" (2)
expiresIn: 86400 (3)
scopes: []
redirectURI: "https://localhost:8443/console/oauth" (4)
userName: "bob" (5)
userUID: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" (6)
authorizeToken: "MDAwYjM5YjMtMzM1MC00NDY4LTkxODItOTA2OTE2YzE0M2Fj" (7)
```

(1) **name** は、API に対して認証を行うためにベアラートークンとして使用されるトークン名です。

2 **clientName** 値は、このトークンを要求した OAuthClient です。

3 **expiresIn** 値は creationTimestamp の有効期限 (秒単位) です。

4 **redirectURI** は、このトークンが作成された承認フローでユーザーがリダイレクトされた場所です。

5 **userName** は、このトークンが認証を許可するユーザーを表します。

6 **userUID** は、このトークンが認証を許可するユーザーの UID を表します。

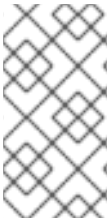
7 **authorizeToken** は、このトークンを取得するために使用される OAuthAuthorizationToken の名前です (ある場合)。

4.8.8. ユーザーオブジェクト

4.8.8.1. Identity

設定された「アイデンティティープロバイダー」を使用して、OpenShift Container Platform にユーザーはログインします。これにより、ユーザーのアイデンティティーが決定され、その情報が OpenShift Container Platform に提供されます。

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



注記

アイデンティティープロバイダーが **lookup** マッピング方法などで設定されている場合、外部の LDAP システムを使用している場合には、この自動マッピングは実行されません。この場合、マッピングは手動で作成する必要があります。詳細は、「Lookup マッピング方法」を参照してください。

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

Identity オブジェクト定義

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

```
user:
  name: "bob" 4
  uid: "9311ac33-0fde-11e5-97a1-3c970e4b7ffe" 5
```

1. アイデンティティー名は providerName:providerUserName の形式である必要があります。
2. **providerName** はアイデンティティープロバイダーの名前です。
3. **providerUserName** は、アイデンティティープロバイダーのスコープでこのアイデンティティーを一意に表す名前です。
4. **user** パラメーターの **name** は、このアイデンティティーがマップされるユーザーの名前です。
5. **uid** は、このアイデンティティーがマップされるユーザーの UID を表します。

4.8.8.2. ユーザー

User はシステムのアクターを表します。ユーザーには、「ロールをユーザーまたはグループに追加」してパーミッションが付与されます。

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

注記

/、:、および **%** を含む OpenShift Container Platform ユーザー名はサポートされません。

User オブジェクト定義

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

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

4.8.8.3. UserIdentityMapping

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

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

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

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

UserIdentityMapping オブジェクト定義

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

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

4.8.8.4. グループ

Groupは、システム内のユーザーの一覧を表します。グループには、「ロールをユーザーまたはグループに追加」してパーミッションが付与されます。

Group オブジェクト定義

```
kind: "Group"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "developers" (1)
  creationTimestamp: "2015-01-01T01:01:01-00:00"
users:
  - "bob" (2)
```

(1) **name** は、ロールをグループに追加する際のグループ名です。

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

第5章 NETWORKING

5.1. NETWORKING

5.1.1. 概要

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

Pod 間のリンクを作成する必要はないので、この IP アドレスを使用して Pod 間で直接相互に通信することは推奨されません。代わりに、「サービス」を作成して、そのサービスと対話することを推奨します。

5.1.2. OpenShift Container Platform DNS

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

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

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

```
0308 19:51:03.118430    4484 node.go:197] Started Kubelet for node
openshiftdev.local, server at 0.0.0.0:10250
I0308 19:51:03.118459    4484 node.go:199]   Kubelet is setting 10.0.2.15 as a
DNS nameserver for domain "local"
```

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

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

表5.1 DNS 名の例

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

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

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

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

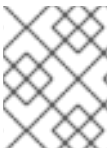
5.2. OPENSHIFT SDN

5.2.1. 概要

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

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

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



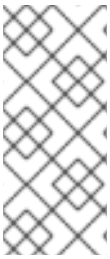
注記

マスターおよびノードでの SDN の設定に関する情報は、「SDN の設定」を参照してください。

5.2.2. マスター上の設計

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

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



注記

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

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

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

5.2.3. ノード上の設計

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

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

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

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

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

4. **ovs-multitenant** プラグインの場合は、Pod からのトラフィックには、その Pod の VNID をタグ付けし、トラフィックの VNID が Pod の VNID (または特権のある VNID 0) と一致する場合にはその Pod にトラフィックを許可するという OpenFlow ルールを追加します。一致しないトラフィックは、一般的なルールで除外されます。

OpenShift SDN ノードは、SDN マスターからのサブネットの更新も監視します。新しいサブネットが追加された場合には、リモートサブネットの宛先 IP アドレスノードを持つパケットが **vxlan0** (**br0** のポート 1) に移動してネットワーク送信されるように、**br0** に OpenFlow ルールを追加します。 **ovs-subnet** プラグインは VNID 0 が指定された VXLAN に全パケットを送信しますが、 **ovs-multitenant** プラグインは、ソースコンテナーに対して適切な VNID を使用します。

5.2.4. パケットフロー

A と B の 2 つのコンテナーがあり、コンテナー A の **eth0** をベースにするピア仮想 Ethernet デバイスの名前が **vethA**、コンテナー B の **eth0** のピア名が **vethB** とします。

注記

Docker サービスが使用するピアの仮想 Ethernet デバイスに慣れていない場合は、「Docker の高度なネットワークに関するドキュメント」を参照してください。

まず。コンテナー A がローカルホストにあり、コンテナー B もローカルホストにあると仮定します。コンテナー A からコンテナー B のパケットフローは以下のようになります。

eth0 (A の netns) →vethA →br0 →vethB →eth0 (B の netns)

次に、コンテナー A がローカルホストに、コンテナー B がクラスターネットワーク上のリモートホストにあると想定します。その場合には、コンテナー A からコンテナー B のパケットフローは以下のようになります。

eth0 (A の netns) →vethA →br0 →vxlan0 → ネットワーク[1] →vxlan0 →br0 →vethB →eth0 (B の netns)

最後に、コンテナー A が外部ホストに接続すると、トラフィックは以下のようになります。

eth0 (A の netns) →vethA →br0 →tun0 → (NAT) →eth0 (物理デバイス) → インターネット

パケット配信の意思決定はほぼ、OVS ブリッジ **br0** の OpenFlow ルールをもとに行われ、プラグインのネットワークアーキテクチャーを簡素化し、ルーティングを柔軟化します。**ovs-multitenant** プラグインの場合は、OpenFlow ルールを元にした意思決定により、強制的な「ネットワーク分離」が可能になります。

5.2.5. ネットワーク分離

ovs-multitenant プラグインを使用して、ネットワーク分離を実現できます。デフォルト以外のプロジェクトに割り当てられた Pod からパケットが送信される場合は、OVS ブリッジ **br0** により、このパケットに、プロジェクトが割り当てた VNID のタグを付けます。パケットが、ノードのクラスターサブネットに含まれる別の IP アドレスに転送される場合には、OVS ブリッジは、VNID が一致する場合にのみ、宛先の Pod に対するこのパケットの配信を許可します。

パケットが別のノードから VXLAN トンネル経由で受信された場合には、トンネル ID を VNID として使用し、OVS ブリッジは、トンネル ID が宛先の Pod の VNID に一致する場合にのみ、ローカル Pod へのパケットの配信を許可します。

他のクラスターサブネットが宛先のパケットは、その VNID でタグ付けされ、クラスターサブネットを所有するノードのトンネルの宛先アドレスが指定された VXLAN トンネルに配信されます。

前述の通り、VNID 0 は、VNID 0 が割り当てられた pod には、VNID が指定されたトラフィックを送信できる点で特権があります。**デフォルト** の OpenShift Container Platform プロジェクトには VNID 0 が割り当てられています。他のプロジェクトにはすべて、一意の分離可能な VNID が割り当てられています。クラスター管理者はオプションで、管理者 CLI を使用してプロジェクトの「Pod ネットワークを管理」できます。

5.3. 利用可能な SDN プラグイン

OpenShift Container Platform は、OpenShift Container Platform と Kubernetes の間のインターフェースとして、Kubernetes Container Network Interface (CNI) をサポートします。Software Defined Network (SDN) プラグインを使用することで、ネットワーク機能がユーザーのネットワークのニーズに対応します。必要に応じて、CNI インターフェースをサポートするプラグインをさらに追加できます。

5.3.1. OpenShift SDN

OpenShift SDN は、デフォルトでAnsible ベースのインストール手順の一部としてインストール設定されます。詳細情報は、「OpenShift SDN」のセクションを参照してください。

5.3.2. サードパーティーの SDN プラグイン

5.3.2.1. Flannel SDN

flannel は、コンテナー専用に設計された仮想ネットワーク層です。OpenShift Container Platform は、デフォルトの Software-Defined Networking (SDN) コンポーネントの代わりに、ネットワークコンテナーとして flannel を使用できます。これは、OpenStack など、SDN にも依存するクラウドプロバイダープロット内で OpenShift Container Platform を実行している場合や、両プラットフォームを通してパケットを 2 回カプセル化しなくても良いようにする場合に便利です。

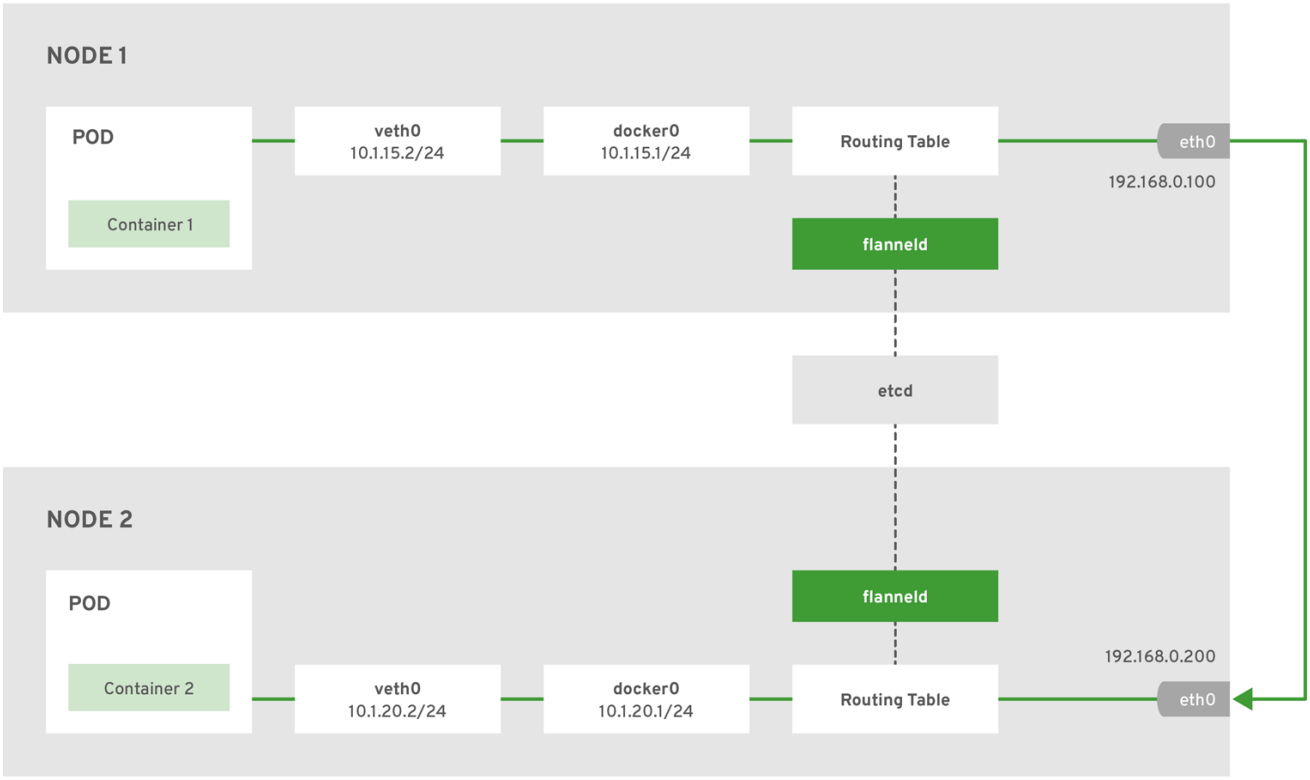
アーキテクチャー

OpenShift Container Platform は、**flannel** を **host-gw** モードで実行し、コンテナー間のルートをマッピングします。ネットワーク内の各ホストは、**flanneld** と呼ばれるエージェントを実行します。このエージェントは以下を行います。

- ホスト毎に一意のサブネットを管理する
- ホスト上の各コンテナーに IP アドレスを割り当てる
- 別のホスト上であっても、コンテナー間のルートをマッピングする

各 **flanneld** エージェントは、この情報を中央の **etcd** ストアに提供し、ホスト上の他のエージェントがパケットを、**flannel** ネットワーク内の他のコンテナーにルーティングできるようにします。

以下の図は、**flannel** ネットワークを使用したコンテナー間のアーキテクチャーおよびデータフローを示します。



ノード 1 には以下のルートが含まれます。

```
default via 192.168.0.100 dev eth0 proto static metric 100
10.1.15.0/24 dev docker0 proto kernel scope link src 10.1.15.1
10.1.20.0/24 via 192.168.0.200 dev eth0
```

ノード 2 には以下のルートが含まれます。

```
default via 192.168.0.200 dev eth0 proto static metric 100
10.1.20.0/24 dev docker0 proto kernel scope link src 10.1.20.1
10.1.15.0/24 via 192.168.0.100 dev eth0
```

5.3.2.2. Nuage SDN

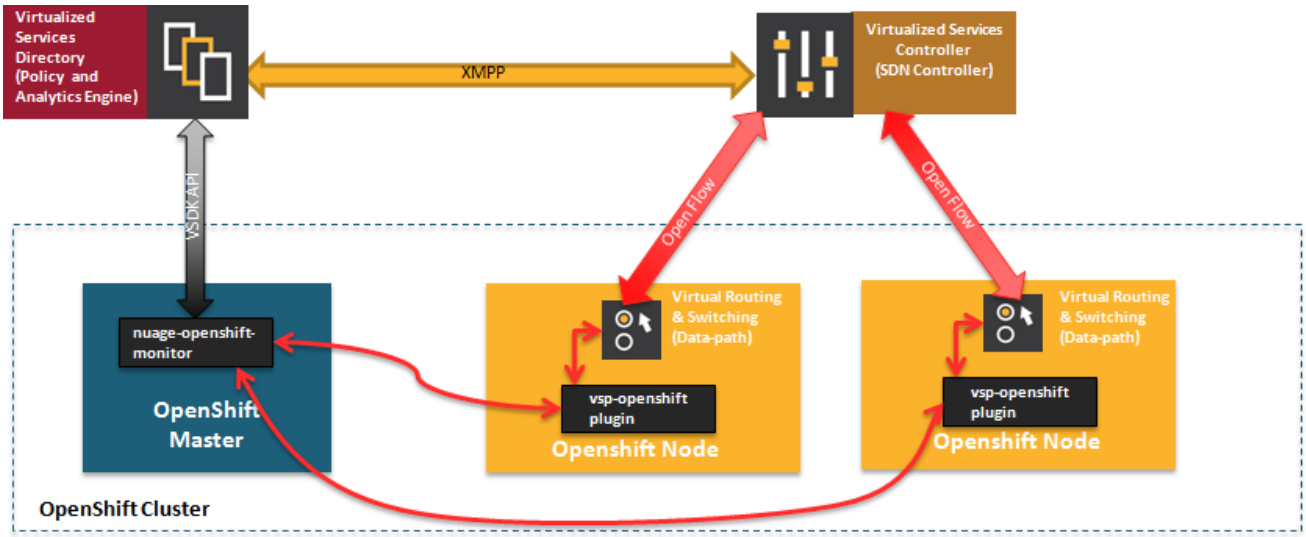
「Nuage Networks」の SDN ソリューションは、OpenShift Container Platform クラスターの Pod に対して、スケーラビリティーが高い、ポリシーベースのオーバーレイネットワークを提供します。Nuage SDN は、Ansible ベースのインストール手順の一部としてインストール、設定可能です。Nuage SDN での OpenShift Container Platform のインストールおよびデプロイ方法に関する情報は、「標準インストール」のセクションを参照してください。

Nuage Networks は、Virtualized Services Platform (VSP) と呼ばれる、スケーラビリティーの高い、ポリシーベースの SDN プラットフォームを提供します。Nuage VSP は、データプレーン用にオープンソースの Open vSwitch とともに、SDN Controller を使用します。

Nuage は、オーバーレイを使用して、OpenShift Container Platform と VM およびベアメタルサーバーからなる他の環境の間をポリシーベースで接続できるようにします。プラットフォームのリアルタイムアナリティクスエンジンでは、OpenShift Container Platform アプリケーションの可視化およびセキュリティー監視を実現します。

Nuage VSP は OpenShift Container Platform と統合し、DevOps チームが直面するネットワークのラグを取り除くことで、ビジネスアプリケーションがすばやく起動と更新ができるようにします。

図5.1 Nuage VSP と OpenShift Container Platform との統合



統合を行う固有のコンポーネントが 2 つあります。

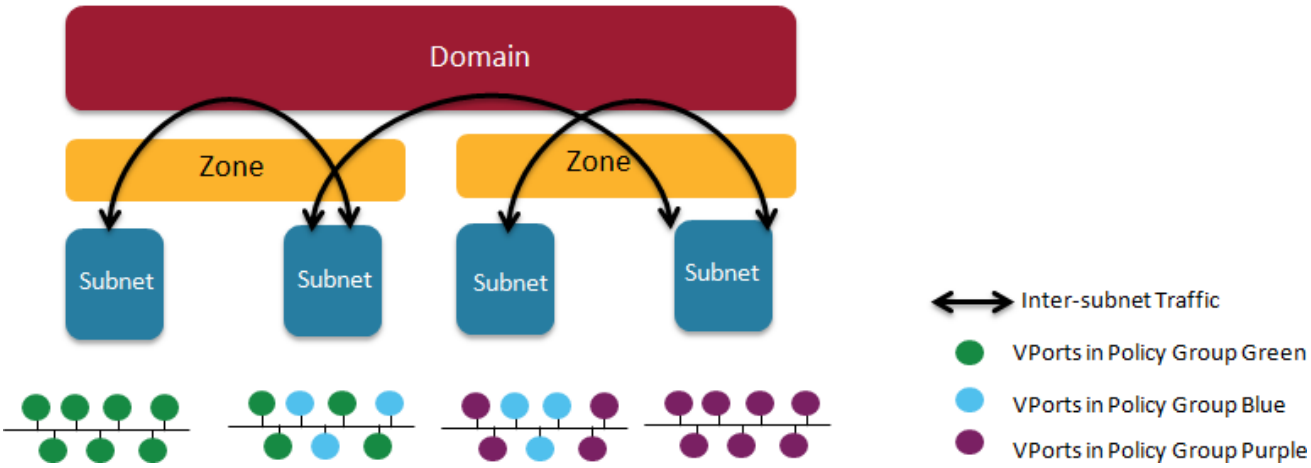
1. **nuage-openshift-monitor** サービス。OpenShift Container Platform マスターノードで個別のサービスとして実行されます。

2. **vsp-openshift** プラグイン。クラスターの各ノードで OpenShift Container Platform ランタイムにより呼び出されます。

Nuage Virtual Routing and Switching ソフトウェア (VRS) は、オープンソースの Open vSwitch をベースにしており、データパス転送を行います。VRS は各ノードで実行され、コントローラーからポリシー設定を取得します。

Nuage VSP の用語

図5.2 Nuage VSP のビルディングブロック



1. ドメイン: 組織には 1 つまたは複数のドメインが含まれます。ドメインは単一の「レイヤー 3」の領域を指します。標準のネットワーク用語では、ドメインは、VRF インスタンスと同じ位置づけです。

2. ゾーン: ゾーンは、ドメインの配下に定義されます。ゾーンは、直接ネットワーク上のなにかにマッピングされるわけではなく、ゾーンの全エンドポイントが同じポリシーセットに準拠するなど、ポリシーが関連付けられているオブジェクトとして機能します。

3. サブネット: サブネットはゾーンの配下に定義されます。サブネットは、ドメインインスタンス内の固有のレイヤー 2 サブネットを指します。サブネットは、ドメイン内で一意で他とは異な

ります。つまり、ドメイン内のサブネットは、重複したり、標準の IP サブネット定義に従って他のサブネットを含めたりすることもできません。

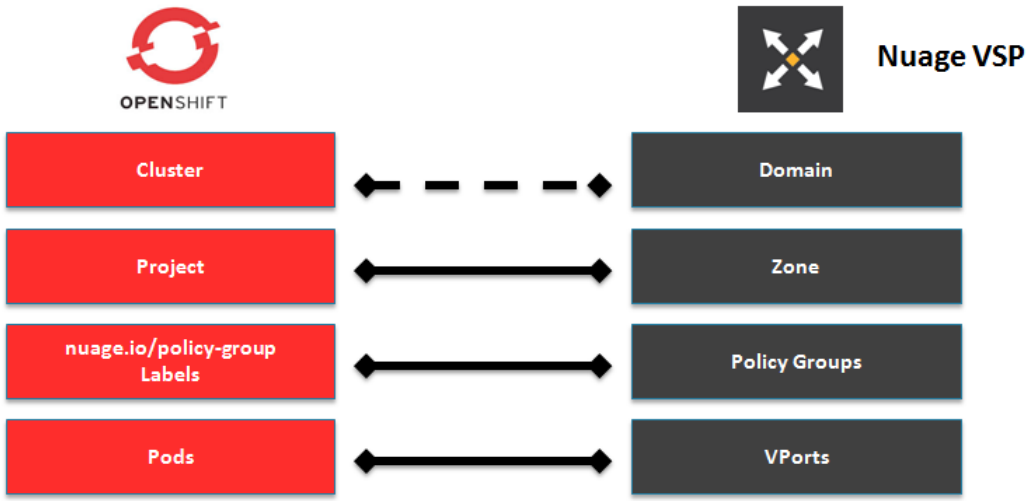
4. VPorts: VPort は、ドメイン階層の新しいレベルで、より粒度の高い設定を可能にするために設計されました。コンテナーや VM に加え、ホストやブリッジインターフェースにアタッチには VPorts も使用し、ベアメタルサーバー、アプリケーション、レガシー VLAN に接続できるようにします。

5. ポリシーグループ: ポリシーグループは VPorts のコレクションです。

コンストラクトのマッピング

OpenShift Container Platform のコンセプト の多くは、Nuage VSP のコンストラクトに直接マッピングできます。

図5.3 Nuage VSP および OpenShift Container Platform のマッピング



Nuage サブネットは、OpenShift Container Platform ノードにマッピングされませんが、特定のプロジェクトのサブネットは、OpenShift Container Platform 内の複数のノードに対応できます。

OpenShift Container Platform で起動する Pod は VSP で作成された仮想ポートに変換されます。**vsp-openshift** プラグインは、VRS と対話し、VSC 経由で VSD からその仮想ポートのポリシーを取得します。ポリシーグループは、複数の Pod をグループ化がサポートされていますが、同じポリシーが適用されている必要があります。現在、Pod は、「オペレーションワークフロー」を使用してポリシーグループに割り当てることができます。このワークフローでは、ポリシーグループは VSD の管理者ユーザーが作成します。ポリシーグループに含まれる Pod は、Pod の仕様で **nuage.io/policy-group** ラベルとして指定されます。

統合コンポーネント

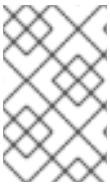
Nuage VSP は、2 つの主要コンポーネントを使用して OpenShift Container Platform と統合します。

1. **nuage-openshift-monitor**

2. **vsp-openshift plugin**

nuage-openshift-monitor

nuage-openshift-monitor は、プロジェクト、サービス、ユーザー、ユーザーグループなどが作成されていないか、OpenShift Container Platform API サーバーを監視するサービスです。



注記

複数のマスターがある高可用性の (HA) OpenShift Container Platform クラスターの場合には、**nuage-openshift-monitor** プロセスは、機能性に変更を加えずに、全マスター上で個別に実行されます。

開発者のワークフローでは、**nuage-openshift-monitor** も、VSD REST API を実行して OpenShift Container Platform コンストラクトを VSP コンストラクトにマッピングすることで、VSD オブジェクトを自動作成します。各クラスターインスタンスは、Nuage VSP の単一ドメインにマッピングします。これにより、Nuage でエンタープライズのドメインインスタンス毎に 1 つ設定するなど、特定のエンタープライズで複数のクラスターをインストールできます。各 OpenShift Container Platform プロジェクトは、Nuage VSP のクラスターのドメインに含まれるゾーンにマッピングされます。**nuage-openshift-monitor** で、プロジェクトの追加、削除が検出された場合に、対象のプロジェクトに対応する VSDK API を使用してゾーンをインスタンス化し、そのゾーンにサブネットのブロックを割り当てます。さらに、**nuage-openshift-monitor** は、このプロジェクトのネットワークマクログループも作成します。同様に、**nuage-openshift-monitor** でサービスの追加や削除が検出された場合には、サービス IP に対応するネットワークマクロを作成して、そのネットワークマクロを該当のプロジェクトのネットワークマクログループに割り当てて (ラベルを使用したユーザー提供のネットワークマクログループもサポートされています)、対象のサービスへの通信を有効化します。

開発者のワークフローでは、ゾーン内で作成された pod はすべて、そのサブネットプールからの IP を取得します。**nuage-openshift-monitor** が、master-config ファイルのプラグイン固有のパラメーター 2 つをもとにして、サブネットプールを割り当て、管理します。ただし、実際の IP アドレスの解決および vport ポリシーの解決は、プロジェクトの作成時にインスタンス化されたドメイン/ゾーンをもとに、VSD が行います。最初のサブネットプールが足りなくなった場合には、**nuage-openshift-monitor** はクラスターの CIDR から追加のサブネットを検出し、特定のプロジェクトに割り当てます。

オペレーションのワークフローでは、アプリケーションまたは pod 仕様に Nuage が認識するラベルを指定して、Pod と固有のユーザー定義ゾーンやサブネットを解決します。ただし、これは、**nuage-openshift-monitor** を使用して開発者ワークフローで作成したゾーンやサブネットの Pod を解決するために使用できません。

注記

オペレーションワークフローでは、管理者は VSD コンストラクトを事前作成し、Pod を特定のゾーン/サブエンっとにマッピングして、OpenShift エンティティー (ACL ルール、ポリシーグループ、ネットワークマクロ、ネットワークマクログループ) 間の通信を可能にします。Nuage ラベルの使用方法に関する説明は『Nuage VSP Openshift Integration Guide』に記載されています。

vsp-openshift Plug-in

vsp-openshift ネットワークプラグインは、OpenShift Container Platform ランタイムが各 OpenShift Container Platform ノードで呼び出します。このプラグインは、ネットワークプラグイン init および Pod の設定、破棄、ステータスフックを実装します。vsp-openshift プラグインは、Pod に IP アドレスも割り当てます。特に、VRS (転送エンジン) と通信して、IP 情報を Pod に設定します。

5.3.3. Kuryr SDN と OpenShift Container Platform

Kuryr (具体的には Kuryr-Kubernetes) は、CNI と OpenStack Neutron を使用して構築した SDN ソリューションです。Kuryr の利点として、幅広い Neutron SDN バックエンドを使用でき、Kubernetes Pod と OpenStack 仮想マシン (VM) の相互接続性が確保できる点が挙げられます。

Kuryr-Kubernetes と OpenShift Container Platform の統合は主に、OpenStack の仮想マシンで実行する OpenShift Container Platform クラスター用に設計されました。

5.3.3.1. OpenStack デプロイメント要件

Kuryr SDN では、使用する OpenStack 設定に関して要件があります。特に以下の要件が含まれます。

- サービスには最低でも、Keystone と Neutron が含まれていること
- Kuryr SDN は Octavia で機能すること
- トランクポート拡張が有効化されていること
- Neutron は Open vSwitch ファイアウォールドライバーを使用すること

5.3.3.2. kuryr-controller

kuryr-controller は、新しい Pod が起動され、その Pod 用に Neutron リソースが作成されていないかどうか、OpenShift Container Platform API を監視するサービスです。たとえば、Pod が作成されると、kuryr-controller はそれを認識し、新規ポートの作成のため、OpenStack Neutron を呼び出します。次に、そのポート (または VIF) に関する情報が Pod のアノテーションに保存されます。また、kuryr-controller は、作成済みのポートプールを使用して、Pod の作成時間を短縮することができます。

現在、kuryr-controller は単一のサービスインスタンスとして実行する必要があるおので、OpenShift Container Platform では、**replicas=1** の **Deployment** としてモデル化されています。kuryr-controller では、OpenStack サービス API にアクセスできる必要があります。

5.3.3.3. kuryr-cni

kuryr-cni コンテナーは、Kuryr-Kubernetes デプロイメントで、OpenShift Container Platform ノードへの Kuryr CNI スクリプトのインストールおよび設定と、ホスト上の **Pod** をネットワーク接続する kuryr-daemon サービスの実行の 2 つの役割を果たします。つまり、kuryr-cni コンテナーはすべての OpenShift Container Platform ノードで実行する必要があるので、**DaemonSet** としてモデル化されます。

OpenShift Container Platform CNI は、新しい Pod が起動するか、Pod が OpenShift Container Platform ホストから削除されるたびに、Kuryr CNI スクリプトを呼び出します。このスクリプトは、ローカルの kuryr-cni のコンテナー ID を Docker API から取得して、CNI 呼び出し引数すべてを渡す Docker 実行ファイルを使用して、Kuryr CNI プラグインバイナリーを実行します。次に、このプラグインは、ローカルの HTTP ソケット経由で kuryr-daemon を呼び出し、再度すべてのパラメーターを渡します。

kuryr-daemon サービスは、このサービス用に作成された Neutron VIF に関する **Pod** のアノテーションを監視します。特定の **Pod** に対する CNI 要求が受信されると、デーモンのメモリーで VIF 情報がすでに認識されているか、**Pod** 定義にアノテーションが表示されるのを待ちます。VIF 情報を認識したら、すべてのネットワーク操作が行われます。

5.4. 利用可能なルータープラグイン

ルーターは、ノードに割り当てて OpenShift Container Platform クラスターのトラフィックを制御することができます。OpenShift Container Platform はデフォルトのルーターとして HAProxy を使用しますが、オプションも提供されています。

5.4.1. HAProxy テンプレートルーター

HAProxy テンプレートのルーター実装は、テンプレートルータープラグインの参照実装です。これは、**openshift3/ose-haproxy-router** リポジトリーを使用して、テンプレートルータープラグインとともに、HAProxy インスタンスを実行します。

テンプレートルーターには 2 つのコンポーネントがあります。

- エンドポイントとルートを監視して変更をもとに HAProxy の再読み込みをトリガーするラッパー
- ルートとエンドポイントをベースに HAProxy 設定ファイルをビルドするコントローラー

注記

HAProxy ルーター はバージョン 1.8.1 を使用します。

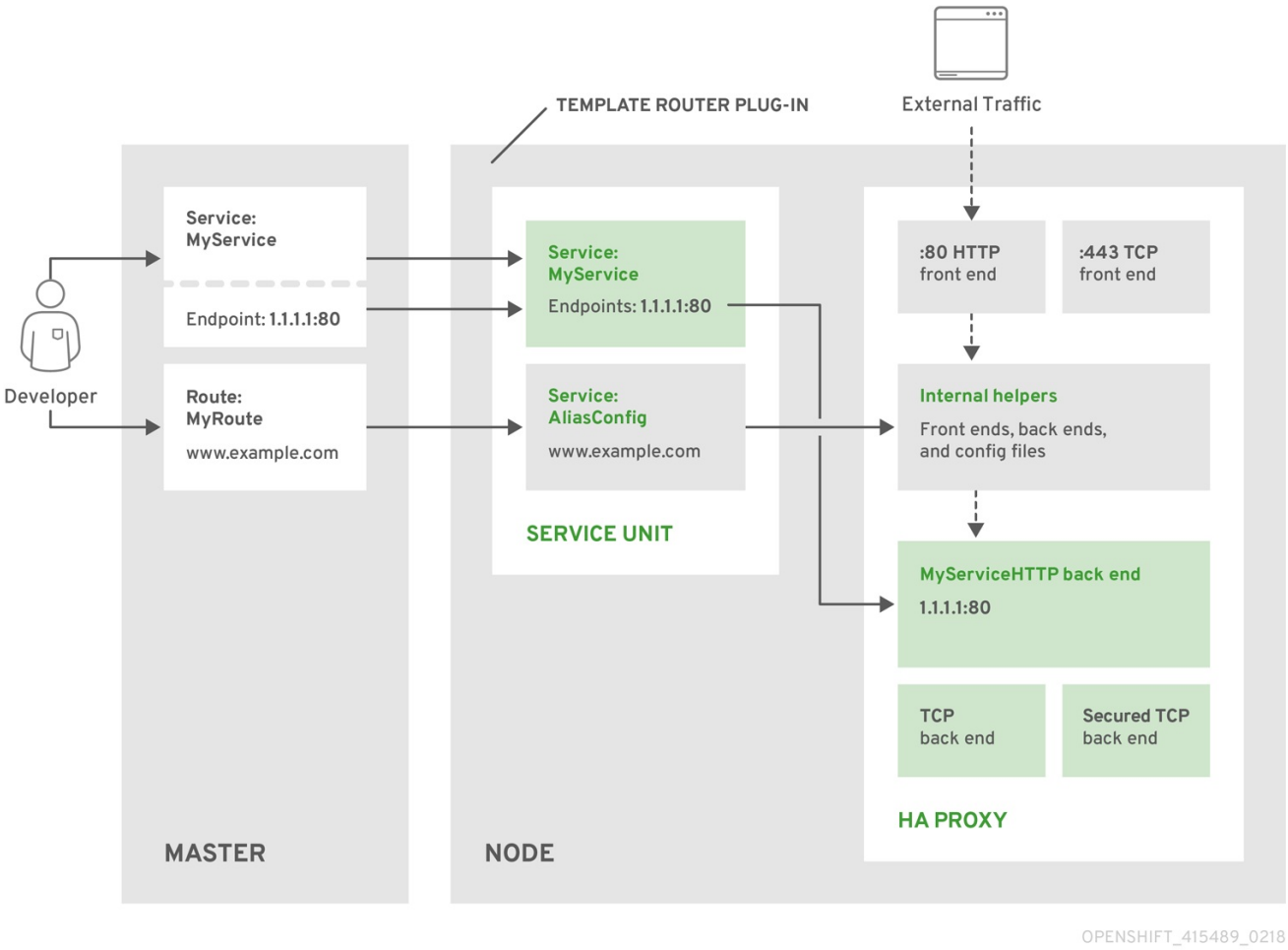
コントローラーおよび HAProxy は、Pod 内に常駐しており、デプロイメント設定で管理されます。ルーターの設定プロセスは、**oc adm router** コマンドで自動化されています。

コントローラーは、ルートとエンドポイントに変更がないか、また、HAProxy プロキシーを監視します。変更が検出されたら、新しい haproxy-config ファイルを作成して、HAProxy を再起動します。haproxy-config ファイルは、ルーターのテンプレートファイルと OpenShift Container Platform からの情報をベースに構築されます。

HAProxy テンプレートファイルは、必要に応じてカスタマイズして、OpenShift Container Platform で現在サポートされていない機能をサポートすることができます。HAProxy マニュアル では、HAProxy がサポートする全機能を説明しています。

以下の図では、データがプラグインを使用してマスターから最終的に HAProxy 設定にどのように移動するかが記載されています。

図5.4 HAProxy ルーターデータフロー



HAProxy テンプレートルーターメトリクス

HAProxy ルーターは、外部のメトリクスコレクションや集計システム (例 Prometheus、statsd) で消費されるように、Prometheus 形式 のメトリクスを提供して公開します。ルーターは、HAProxy CSV 形式 のメトリクスを提供するように設定したり、まったくルーターメトリクスを提供しないように設定したりできます。

メトリクスは、ルーターコントローラーおよび HAProxy の両方から 5 秒毎に取得されます。ルーターメトリクスカウンターは、ルーターのデプロイ時に 0 から開始され、経時的に増加します。HAProxy メトリクスカウンターは、HAProxy が再読み込みされるたびに 0 にリセットされます。ルーターはフロントエンド、バックエンド、サーバー毎に HAProxy 統計を収集します。サーバーが 500 台以上ある場合に、リソースの使用量を減らすには、バックエンドでは複数のサーバーを利用できるので、サービスではなく、バックエンドをレポートします。

この統計は 利用可能な HAProxy 統計 のサブセットです。

以下の HAProxy メトリクスは定期的に収集され、Prometheus 形式に変換されます。フロントエンドはすべて、"F" カウンターが収集されます。バックエンド毎のカウンターを収集する場合には、サーバー毎の "S" サーバーカウンターが収集されます。それ以外の場合は、バックエンド毎に "B" カウンターが収集され、サーバーカウンターは収集されません。

詳細は、ルーター環境変数を参照してください。

以下の表を参照してください。

列 1 - HAProxy CSV 統計のインデックス

列 2

F	フロントエンドのメトリクス
b	サーバーのしきい値が原因でサーバーメトリクスを表示しない場合のバックエンドメトリクス
B	サーバーメトリクスを表示する場合のバックエンドメトリクス
S	サーバーメトリクス

列 3 - カウンター

列 4 - カウンターの説明

インデックス	使用法	カウンター	説明
2	bBS	current_queue	サーバーに割り当てられていないキューに入れられた要求の現在の数。
4	FbS	current_sessions	アクティブなセッションの現在の数。
5	FbS	max_sessions	アクティブなセッションの観察される最大数。
7	FbBS	connections_total	接続の合計数。
8	FbS	bytes_in_total	受信バイトの現在の合計。
9	FbS	bytes_out_total	送信バイトの現在の合計。
13	bS	connection_errors_total	接続エラーの合計。
14	bS	response_errors_total	応答エラーの合計。
17	bBS	アップ	現在のバックエンドのヘルスステータス (1 = UP、0 = DOWN)。
21	S	check_failures_total	失敗したヘルスチェックの合計数。

24	S	downtime_seconds_total	合計ダウンタイム (秒)。
33	FbS	current_session_rate	直近の 1 秒間で、1 秒あたりの現在のセッション数。
35	FbS	max_session_rate	1 秒あたりの最大セッション実数。
40	FbS	http_responses_total	HTTP 応答合計数、コード 2xx。
43	FbS	http_responses_total	HTTP 合計応答数、コード 5xx。
60	bS	http_average_response_latency_milliseconds	直近の要求 1024 件のうちの HTTP 応答 (ミリ秒単位)。

ルーターコントローラーは、以下のアイテムを収集します。これらは、Prometheus 形式のメトリクスでのみ提供されます。

名前	説明
template_router_reload_seconds	ルーターの再読み込みにかかる時間を秒単位で測定します。
template_router_write_config_seconds	ルーター設定のディスクへの書き込みにかかる時間を秒単位で測定します。
haproxy_exporter_up	最後に成功した haproxy の収集です。
haproxy_exporter_csv_parse_failures	CSV の解析時のエラー数です。
haproxy_exporter_scrape_interval	次の収集が許可されるまでの秒単位の時間です (データのサイズに比例します)。
haproxy_exporter_server_threshold	追跡したサーバー数と現在のしきい値です。
haproxy_exporter_total_scrapes	現在の合計 HAProxy 収集数です。
http_request_duration_microseconds	ミクロ秒単位の HTTP 要求のレイテンシーです。
http_request_size_bytes	バイト単位の HTTP 要求サイズです。
http_response_size_bytes	バイト単位の HTTP 応答サイズです。

openshift_build_info	OpenShift がビルドされた major、minor、git commit、git version でラベル付けされた定数値 '1' のメトリクスです。
ssh_tunnel_open_count	SSH トンネルを開放しようと試行した合計数です。
ssh_tunnel_open_fail_count	SSH トンネルを開放しようとして失敗した合計数です。

5.4.2. F5 BIG-IP® ルータープラグイン

F5 BIG-IP® ルータープラグインは、利用可能な「ルータープラグイン」の 1 つです。

注記

F5 ルータープラグインは OpenShift Enterprise 3.0.2 以降で利用できます。

F5 ルータープラグインは、お使いの環境で既存の **F5 BIG-IP®** システムと統合します。F5 iControl REST API を使用するには、**F5 BIG-IP** バージョン 11.4 以降が必要です。F5 ルーターは、HTTP vhost および要求パスで一致する unsecured、edge termination、re-encryption termination および passthrough termination ルートをサポートします。

F5 ルータープラグインは、HAProxy テンプレートルーター と同等機能を備えており、以下の機能を追加でサポートします。

- パスベースのルーティング (ポリシールールを使用)
- Re-encryption (クライアントおよびサーバーの SSL プロファイルを使用した実装)
- 暗号化接続のパススルー (SNI プロトコルを解析し、サーバー名ルックアップ用に F5 ルーターが管理するデータグループを使用する iRule で実装)

注記

パススルールートは特別なケースです。**F5 BIG-IP** 自体が HTTP 要求を認識できず、パスを検証できないので、パスベースのルーティングは技術的に、パススルールートでは不可能です。同じ制限が、テンプレートルーターにも適用されます。これは、パススルー暗号化の技術的制限で、OpenShift Container Platform の技術的制限ではありません。

5.4.2.1. SDN 経由での Pod に対するトラフィックのルーティング

F5 BIG-IP は OpenShift SDN 外にあるので、クラスター管理者は **F5 BIG-IP** と SDN 上にあるホスト (通常は OpenShift Container Platform ノードホスト) 間でピアツーピアトンネルを作成する必要があります。この **ramp ノード** は、pod に対して スケジュール不可 と設定して、**F5 BIG-IP** ホストのゲートウェイ以外として機能しないようにすることができます。また、このようなホストを複数設定して、冗長化のために OpenShift Container Platform **ipfailover** 機能を使用できます。**F5 BIG-IP** ホストは、トンネルのリモートエンドポイントに、**ipfailover** VIP を使用するように設定する必要があります。

5.4.2.2. F5 統合の詳細

F5 ルータープラグインの操作は、以前のバージョンで使用していた OpenShift Container Platform **routing-daemon** とよく似ています。いずれも REST API 呼び出しを使用して以下を行います。

- プールを作成、削除する
- これらのプールにエンドポイントを追加して、このプールからエンドポイントを削除する
- ポリシールールが vhost をベースにしてププールにルーティングするように設定する

いずれも **scp** と **ssh** コマンドを使用してカスタムの TLS/SSL 証明書を **F5 BIG-IP** にアップロードします。

F5 ルータープラグインは、仮想サーバー上のプールとポリシールールを以下のように設定します。

- ユーザーが OpenShift Container Platform でルートを作成、または削除すると、ルーターは **F5 BIG-IP** にルート用のプールを作成して (すでにプールが存在しない場合)、TLS 以外のルートの HTTP vserver またはエッジまたは再暗号化ルートの HTTPS vserver など、適切な vserver のポリシーに対してルールを追加/削除します。edge および re-encrypt ルートの場合には、ルーターも TLS 証明書と鍵をアップロードして設定します。ルーターは、ホストおよびパスベースのルートをサポートします。

注記

パススルーは特別なケースです。これをサポートするには、SNI ClientHello ハンドシェークレコードを解析して、F5 データグループでサーバー名をルックアップするための iRule を記述する必要があります。ルーターはこの iRule を作成して、この iRule と vserver を関連付け、パススルールートの作成/削除に伴い、F5 データグループを更新します。この実装の詳細以外は、パススルールートは、他のルートと同じように機能します。

- OpenShift Container Platform でサービスを作成すると、ルートは **F5 BIG-IP** にプールを追加します (プールが存在しない場合)、このサービスにエンドポイントが作成/削除されると、ルーターは適切なプールメンバーを追加/削除します。
- ユーザーがルートを削除と、特定のプールに関連付けられたエンドポイントをすべて削除すると、ルーターはそのプールを削除します。

5.4.2.3. F5 ルータープラグイン

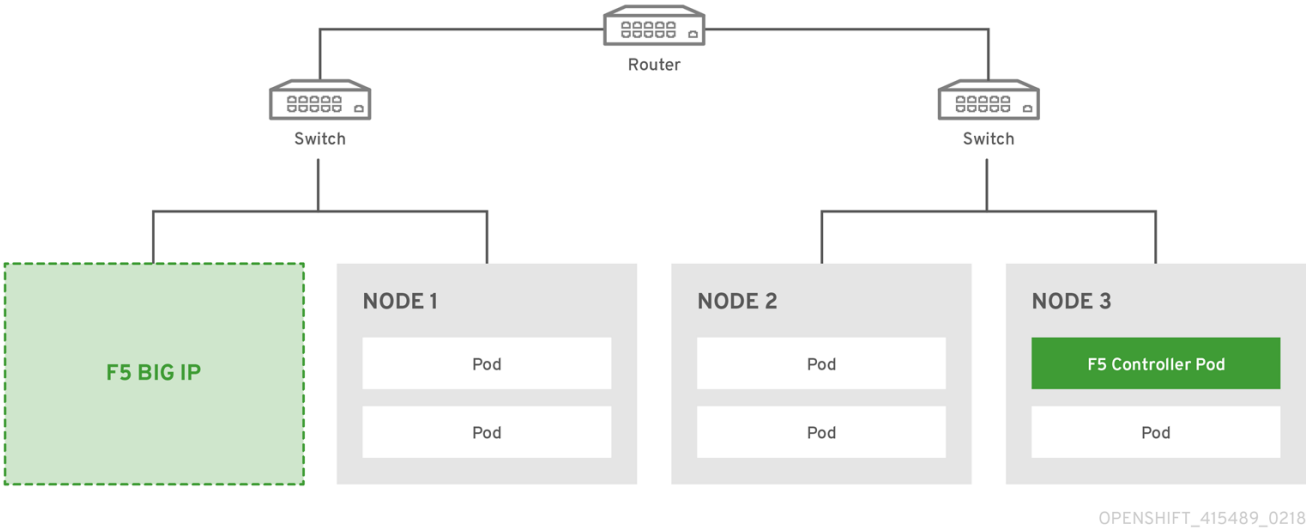
F5 Big-IP と OpenShift Container Platform をネイティブで統合する場合に 、ramp ノードの設定は OpenShift SDN で作成されるので、F5 Big-IP がオーバーレイネットワーク上の Pod に到達できるように ramp ノードを設定する必要はありません。

また、**F5 BIG-IP** アプライアンスのバージョン 12.x 以降のみが、このセクションに記載されている F5 ルータープラグインに対応しています。また、適切に統合を行うために **sdn-services** アドオンライセンスが必要になります。バージョン 11.x の場合は、ramp ノードを設定してください。

接続
F5 アプライアンスは、L3 接続で OpenShift Container Platform クラスターに接続できます。OpenShift Container Platform のノード間では、L2 スイッチの接続性は必要ありません。F5 アプライアンスでは、複数のインターフェースを使用して統合を管理できます。

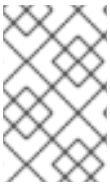
- 管理インターフェース: F5 アプライアンスの Web コンソールに到達する
- 外部インターフェース: 受信 Web トラフィック用に仮想サーバーを設定する

- 内部インターフェース: アプライアンスをプログラムして、Pod に到達する



F5 コントローラー Pod は、アプライアンスへの **admin** 権限があります。F5 イメージは、OpenShift Container Platform クラスター (ノード上にスケジュール) で起動して、iControl REST API を使用してポリシーで仮想サーバーをプログラムし、VxLAN デバイスを設定します。

データフロー: パケットから Pod へ



注記

このセクションでは、パケットが Pod に到達する方法および Pod がパケットに到達する方法について説明します。これらのアクションは、ユーザーではなく、F5 ルータープラグイン Pod と F5 アプライアンスにより実行されます。

ネイティブで統合されると、F5 アプライアンスは VxLAN のカプセル化を使用して直接 pod に到達します。この統合は、OpenShift Container Platform が **openshift-sdn** をネットワークプラグインとして使用している場合のみ機能します。**openshift-sdn** プラグインは、このプラグインが作成するオーバーレイネットワークに対して、VxLAN のカプセル化を採用します。

Pod と F5 アプライアンスの間でデータパスを正常に確立するには以下を行います。

1. F5 は、Pod 向けの VxLAN パケットをカプセル化する必要があります。これには、**sdn-services** ライセンスアドオンが必要です。VxLAN デバイスを作成する必要があり、Pod オーバーレイネットワークはこのデバイス経由でルーティングする必要があります。

2. F5 は Pod の VTEP IP アドレスを知る必要があります。これは、Pod が配置されているノードの IP アドレスです。

3. F5 は、Pod 向けのパケットをカプセル化する時に、オーバーレイネットワークに使用する **source-ip** を知っておく必要があります。これは **ゲートウェイアドレス** として知られています。

4. OpenShift Container Platform ノードは、F5 ゲートウェイアドレスがなにか (戻りトラフィックの VTEP アドレス) を知っておく必要があります。このアドレスは、内部インターフェースのアドレスでなければなりません。クラスターの全ノードは、自動的にこれを学習する必要があります。

5. オーバーレイネットワークはマルチテナントに対応しているので、F5 は **admin** を代表する VxLAN ID を使用して F5 が全テナントに到達できるようにする必要があります。手動作成した **hostsubnet** (**pod.network.openshift.io/fixed-vnid-host: 0**) でアノテーションを指定すること

で、F5 により **vnid** が **0** (OpenShift Container Platform の **admin** namespace でデフォルトの **vnid**) の全パケットがカプセル化されるようにします。

ghost **hostsubnet** は、設定の一部として手動で作成します。これは、3 番目と 4 番目の要件を満たします。F5 ルータープラグインの起動時に、F5 アプライアンスが適切にプログラムされるように、この新しい ghost **hostsubnet** が提供されます。

注記

サブネットがクラスターのノードに割り当てられるので、**ghost hostsubnet** という用語が使用されます。ただし、実際には、クラスターの実際のノードではなく、外部のアプライアンスによりハイジャックされます。

1 番目の要件は、F5 コントローラー Pod の起動時に、F5 ルータープラグインが満たします。2 番目の要件は、F5 プラグイン Pod が対応しますが、継続プロセスです。クラスターに追加される新規ノードごとに、コントローラー Pod は VxLAN デバイスの VTEP FDB のエントリーを作成します。コントローラー Pod には、クラスターの **nodes** リソースへのアクセス権が必要ですが、これは、サービスアカウントに適切な権限を割り当てることで対応できます。以下のコマンドを使用してください。

```
$ oc adm policy add-cluster-role-to-user system:sdn-reader system:serviceaccount:default:router
```

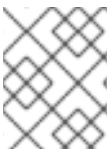
F5 ホストからのデータフロー

注記

以下のアクションは、ユーザーではなく F5 ルータープラグイン Pod および F5 アプライアンスによって実行されます。

1. 宛先 Pod はパケットの F5 仮想サーバーで識別されます。
2. VxLAN 動的 FDB は Pod の IP アドレスで検索されます。MAC アドレスが見つかる場合はステップ 5 に進みます。
3. Pod の MAC アドレスを求める ARP 要求で、VTEP FDB の全エントリーをいっぱいにします。Pod の MAC アドレスと VTEP を値として使用して、VxLAN ダイナミック FDB にエントリーが追加されます。
4. VxLAN ヘッダーで IP パケットをカプセル化します。Pod の MAC およびノードの VTEP が、VxLAN 動的 FDB からの値として指定されます。
5. ARP を送信し、ホストの周辺にあるキャッシュを確認して、VTEP の MAC アドレスを計算します。
6. F5 ホストの内部アドレスでパケットを送信します。

データフロー: トラフィックを F5 ホストに返す



注記

以下のアクションは、ユーザーではなく F5 ルータープラグイン Pod および F5 アプライアンスによって実行されます。

1. Pod は、宛先を F5 ホストの VxLAN ゲートウェイアドレスとしてパケットを戻します。

2. ノードの **openvswitch** は、このパケットの VTEP が F5 ホストの内部インターフェースアドレスであるかどうか判断します。これは、ghost **hostsubnet** の作成時に分かります。

3. VxLAN パケットは、F5 ホストの内部インターフェースに送信されます。

注記

マルチテナントを回避するように、VNID は予め、データフロー全体で **0** に固定されます。

5.5. ポート転送

5.5.1. 概要

OpenShift Container Platform は、Kubernetes に組み込まれている機能を活用して、Pod へのポート転送をサポートします。これは、HTTP と **SPDY** または **HTTP/2** などの多重化ストリーミングプロトコルを使用して実装されます。

開発者は「CLI」を使用して pod にポート転送できます。CLI は、ユーザーが指定した各ローカルポートをリッスンし、「記載のプロトコル」を使用して転送します。

5.5.2. サーバー操作

Kubelet は、クライアントからのポート転送要求を処理します。要求を受信すると、応答をアップグレードし、クライアントがポート転送ストリームを作成するまで待機します。新規ストリームを受信したら、ストリームと Pod ポート間のデータをコピーします。

アーキテクチャー的には、Pod のポートに転送するオプションがあります。OpenShift Container Platform でサポートされる現在の実装はノードホストで直接 **nsenter** を呼び出し、Pod ネットワークの namespace に入り、**socat** を呼び出してストリームと Pod のポート間のデータをコピーします。ただし、カスタムの実装には、**nsenter** と **socat** のバイナリーをホストにインストールしなくていいように、これらのバイナリーを実行する「ヘルパー」 Pod の実行が含まれています。

5.6. リモートコマンド

5.6.1. 概要

OpenShift Container Platform は Kubernetes に内蔵されている機能を活用し、コンテナーでのコマンド実行をサポートします。これは、HTTP と **SPDY** または **HTTP/2** などの多重化ストリーミングプロトコルを使用して実装されます。

開発者は「CLI を使用」して、コンテナーでリモートコマンドを実行します。

5.6.2. サーバー操作

Kubelet は、クライアントからのリモート実行要求を処理します。要求を受信すると応答をアップグレードして、要求ヘッダーを評価してどのストリーム (**stdin**、**stdout** および/または **stderr**) を受信するか判断し、クライアントがストリームを作成するまで待機します。

Kubelet が全ストリームを受信したら、コンテナーでコマンドを実行して、ストリームとコマンドの stdin、stdout および stderr を適切にコピーします。コマンドが中断されたら、Kubelet はアップグレードされた接続と基盤の接続を終了します。

アーキテクチャー的に、コンテナーでコマンドを実行するオプションがあります。OpenShift Container Platform で現在サポートされている実装では、ノードホストで **nsenter** を直接呼び出して、コマンド実行前に、ノードホストがコンテナーの namespace に入ることができるようにします。ただし、カスタム実装には **docker exec** の使用や、ホストでインストールする必要のある **nsenter** バイナリーが必須とならないように **nsenter** を実行する「ヘルパー」コンテナーの実行が含まれる場合があります。

5.7. ルート

5.7.1. 概要

OpenShift Container Platform ルートは、外部クライアントが名前でサービスに到達できるように、**www.example.com** などのホスト名で サービス を公開します。

ホスト名の DNS 解決はルーティングとは別に処理されます。管理者が、OpenShift Container Platform ルーターを実行する OpenShift Container Platform ノードを解決するように「DNS ワイルドカードエントリー」を設定している場合があります。異なるホスト名を使用している場合には、DNS レコードを個別に変更して、ルーターを実行するノードを解決する必要があります。

各ルーターは名前 (63 文字に制限)、サービスセレクター、およびオプションのセキュリティー設定で構成されています。

5.7.2. ルーター

OpenShift Container Platform 管理者は、OpenShift Container Platform のノードに **ルーター** をデプロイできるので、「開発者が作成した」ルートを外部クライアントが利用できるようになります。OpenShift Container Platform のルーティング層はプラグ可能で、複数の「ルータープラグイン」がデフォルトで提供、サポートされています。

注記

ルーターのデプロイに関する情報は、『クラスター設定ガイド』を参照してください。

ルーターは、サービスセレクターを使用して、「サービス」と、サービスをバッキングするエンドポイントを検索します。ルーターとサービス両方が負荷分散を提供する場合には、OpenShift Container Platform はルーターの負荷分散を使用します。ルーターはサービスの IP アドレスに関連の変更がないかを検出して、その設定に合わせて変化します。これは、カスタムルーターが API オブジェクトの変更を外部のルーティングソリューションに通信できるので、便利です。

要求のパスは、1 つまたは複数のルーターに対して、ホスト名の DNS を解決することから始まります。推奨の方法は、複数のルーターインスタンスでバックされる 1 つまたは複数の仮想 IP (VIP) アドレスを参照するワイルドカード DNS エントリーでクラウドドメインを定義する方法です。クラウドドメイン外の名前とアドレスを使用するルートには、個別の DNS エントリー設定が必要です。

VIP アドレスがルーターよりも少ない場合には、アドレス分のルーターが **active** で、残りは **passive** になります。passive ルーターは **ホットスタンバイ** ルーターとして知られています。たとえば、VIP アドレスが 2 つ、ルーターが 3 つの場合には、「active-active-passive」構成になります。ルーターの VIP 設定に関する詳細情報は、「高可用性」を参照してください。

ルートは、ルーターセット間で シャード化 されます。管理者は、クラスター全体でシャード化を設定し、ユーザーはプロジェクトに含まれる namespace にシャード化を設定できます。オペレーターは、シャード化すると、複数のルーターグループを定義できるようになります。このグループ内の各ルーターはトラフィックのサブネット 1 つにのみ対応できます。

OpenShift Container Platform ルーターは、外部のホスト名マッピングと、ルーターに区別情報を直接渡すプロトコルを使用して サービス エンドポイントの負荷分散を行います。ルーターは、送信先を判断できるように、プロトコルにホスト名が存在している必要があります。

ルータープラグインはデフォルトで、ホストポート 80 (HTTP) および 443 (HTTPS) にバインドできます。つまり、配置されていないと、これらのポートは使用されないので、ルーターはノードに配置されている必要があります。または、ルーターは、**ROUTER_SERVICE_HTTP_PORT** および **ROUTER_SERVICE_HTTPS_PORT** 環境変数を設定して、他のポートをリッスンするように設定してください。

ルーターは、ホストノードのポートにバインドされるので、ルーターがホストネットワークを使用している場合には (デフォルト)、各ノードに 1 つしかこれらのポートをリッスンするルーターを配置できません。クラスターネットワークは、全ルーターがクラスター内のすべての Pod にアクセスできるように設定します。

ルーターは以下のプロトコルをサポートします。

- HTTP
- HTTPS (SNI あり)
- WebSockets
- SNI 付きの TLS

注記

WebSocket トラフィックは、同じルート規則を使用し、他のトラフィックと同じ TLS 終端タイプをサポートします。

セキュアな接続を確立するには、クライアントとサーバーに共通する 暗号化 を取り決める必要があります。時間が経つにつれ、よりセキュリティーの高く、新しい暗号化が利用でき、クライアントソフトウェアに統合されます。以前のクライアントは陳腐化し、セキュリティーの低い、以前の暗号化は使用が停止されます。デフォルトでは、ルーターは、一般的に入手できる、幅広いクライアントに対応します。ルーターは、任意のクライアントをサポートする暗号化の中で選択したものを使用し、セキュリティーが低い暗号化を使用しないように設定できます。

5.7.2.1. テンプレートルーター

テンプレートルーター は、ルーターの一種で、特定のインフラストラクチャー情報を基盤のルーター実装に提供します。以下に例を示します。

- エンドポイントおよびルートを監視するラッパーです。
- 消費可能なフォームに保存されるエンドポイントとルートデータ。
- 内部の状態を設定可能なテンプレートに渡し、テンプレートを実行します。
- 再ロードスクリプトを呼び出します。

5.7.3. 利用可能なルータープラグイン

検証された利用可能なルータープラグインについては、「利用可能なプラグイン」のセクションを参照してください。

これらのルーターのデプロイ方法については、「ルーターのデプロイ」を参照してください。

5.7.4. スティッキーセッション

スティッキーセッションの実装は、基盤のルーター設定により異なります。デフォルトの HAProxy テンプレートは、**balance source** ディレクティブを使用してスティッキーセッションを実装し、ソース IP をもとに分散されます。さらに、このテンプレートルータープラグインは、サービス名と namespace を基盤の実装に渡します。これは、ピア間で同期させるスティックテーブルの実装など、より高度な設定に使用できます。

スティッキーセッションは、ユーザー体験の向上のため、ユーザーのセッションからの全トラフィックが確実に同じ Pod に移動されるようにします。ユーザーの要求を満たしながら、Pod は後続の要求で使用できるように、データをキャッシュします。たとえば、バックエンド Pod が 5 つと負荷分散ルーターが 2 つあるクラスターでは、要求を処理するルーターがどれであっても、同じ Pod で、同じ Web ブラウザーからの web トラフィックを受信できるように確保できます。

ルーティングトラフィックを同じ Pod に返すことが望まれる場合でも、保証はできませんが、HTTP ヘッダーを使用して、cookie を設定し、最後の接続で使用した Pod を判断できます。ユーザーがアプリケーに別の要求を送信した場合には、ブラウザーが cookie を再送することで、ルーターでトラフィックの送信先が分かります。

クラスター管理者は、スティッキネスをオフにして、他の接続とパススルールートを分割することも、完全にスティッキネスをオフにすることもできます。

デフォルトでは、パススルールートのスティッキーセッションは、**source** 「負荷分散ストラテジー」を使用して実装します。すべてのパスルート用には、**ROUTER_TCP_BALANCE_SCHEME** 「環境変数」で、個別ルート用には、**haproxy.router.openshift.io/balance** 「ルート固有のアノテーション」で、デフォルトを変更することができます。

他の種類のルートはデフォルトで **leastconn** 「負荷分散ストラテジー」を使用しますが、これは **ROUTER_LOAD_BALANCE_ALGORITHM** 「環境変数」を使用して変更できます。また、個別ルートには**haproxy.router.openshift.io/balance** 「ルート固有のアノテーション」を使用して変更することができます。

注記

cookie は、HTTP トラフィックを表示できないので、パススルールートで設定できません。代わりに、ソース IP アドレスをベースに数が計算され、バックエンドを判断します。

バックエンドが変わった場合には、トラフィックは誤ったサーバーに送られ、スティッキネスが低くなります。ロードバランサーを使用する場合は (ソース IP が表示されない)、同じ番号が全接続に設定され、トラフィックが同じ Pod に送信されます。

さらに、このテンプレートルータープラグインは、サービス名と namespace を基盤の実装に渡します。これは、ピア間で同期させるスティックテーブルの実装など、より高度な設定に使用できます。

このルーター実装固有の設定は、ルーターコンテナーの **/var/lib/haproxy/conf** ディレクトリーにある **haproxy-config.template** ファイルに保存されます。ファイルは「カスタマイズ可能です」。

注記

source の 負荷分散ストラテジー は、NAT の設定が原因で、外部のクライアント IP アドレスを区別しないので、送信元の IP アドレス (HAProxy リモート) は同じです。HAProxy ルーターが **hostNetwork: true** で実行されない限り、すべての外部クライアントは単一の Pod にルーティングされます。

5.7.5. ルーターの環境変数

このセクションで説明されているアイテムはすべて、ルーターの **デプロイメント設定** に環境変数を指定して設定を変更するか、**oc set env** コマンドを使用します。

```
$ oc set env <object_type>/<object_name> KEY1=VALUE1 KEY2=VALUE2
```

以下に例を示します。

```
$ oc set env dc/router ROUTER_SYSLOG_ADDRESS=127.0.0.1 ROUTER_LOG_LEVEL=debug
```

表5.2 ルーターの環境変数

変数	デフォルト	説明
DEFAULT_CERTIFICATE		TLS サーバー証明書を公開しないルートに使用するデフォルトの証明書の内容。PEM 形式。
DEFAULT_CERTIFICATE_DIR		**tls.crt** というファイルを含むディレクトリーへのパス。**tls.crt** が PEM ファイルでなく、秘密鍵も含む場合には、同じディレクトリー内の tls.key というファイルと先に統合されます。PEM 形式のコンテンツは、デフォルトの証明書として使用されます。これは、**DEFAULT_CERTIFICATE** または **DEFAULT_CERTIFICATE_PATH** が指定されていない場合のみ使用されます。
DEFAULT_CERTIFICATE_PATH		TLS サーバー証明書を公開しないルートに使用するデフォルト証明書へのパス。PEM 形式。**DEFAULT_CERTIFICATE** が指定されていない場合のみ使用されます。
DROP_SYN_DURING_RESTART	**false**	**true** に設定されている場合は、iptables ルールを有効にして、再起動中に特定のパケットをドロップして、シームレスに再起動できるようにします。詳細は、『インストールガイド』を参照してください。
EXTENDED_VALIDATION	**true**	**true** の場合は、ルーターにより、証明書が構造的に正しいことを確認します。CA に対して証明書は検証されません。テストをオフにするには、**false** に設定します。
NAMESPACE_LABELS		監視する namespace に適用するラベルセレクターです。空はすべてを意味します。
PROJECT_LABELS		監視するプロジェクトに適用するラベルセレクターです。空はすべてを意味します。
RELOAD_SCRIPT		ルーターを再ロードするために使用する再ロードスクリプトのパスです。

変数	デフォルト	説明
ROUTER_ALLOWED_DOMAINS		ルートのホスト名のみを含めることができるドメインのコンマ区切りの一覧。ドメインに含まれるサブドメインを使用できます。**ROUTER_DENIED_DOMAINS** オプションは、このオプションに指定されている値を上書きします。これが設定されている場合は、許可されているドメイン以外はすべて拒否されます。
ROUTER_BACKEND_PROCESS_ENDPOINTS		テンプレート関数 processEndpointsForAlias の使用時にエンドポイントをどのように処理すべきかを指定する文字列。有効な値は ["shuffle", ""] です。"shuffle" は、全呼び出し毎に要素を無作為に決定します。デフォルトの動作は、事前に決定した順番に、値が返されます。
ROUTER_BIND_PORTS_AFTER_SYNC	**false**	**true** または **TRUE** に設定されている場合には、ルーターは、完全な同期状態になるまでポートにバインドされません。'true' または 'TRUE' に設定されていない場合は、ルーターはポートにバインドされ、即座に要求処理を開始しますが、ルートで読み込まれないものもあります。
ROUTER_COOKIE_NAME		cookie 名を指定して、内部で生成したデフォルト名を上書きします。名前は、大文字、小文字、数字、"_" または "-" を任意に組み合わせて指定する必要があります。デフォルトは、ルートのハッシュ化された内部キー名です。
ROUTER_COMPRESSION_MIME	"text/html text/plain text/css"	圧縮するスペースで区切られた mime タイプの一覧です。
ROUTER_DENIED_DOMAINS		ルートのホスト名に含めることができないドメインのコンマ区切りの一覧。ドメインに含まれるサブドメインを使用できません。**ROUTER_ALLOWED_DOMAINS** オプションを上書きします。
ROUTER_ENABLE_COMPRESSION		**true** または **TRUE** の場合、可能な場合には応答を圧縮します。
ROUTER_LISTEN_ADDR	0.0.0.0:1936	ルーターメトリクスのリッスンアドレスを設定します。
ROUTER_LOG_LEVEL	警告	syslog サーバーに送信するログレベルです。
ROUTER_MAX_CONNECTIONS	20000	同時接続の最大数です。

変数	デフォルト	説明
ROUTER_METRICS_HAPROXY_SERVER_THRESHOLD	500	
ROUTER_METRICS_HAPROXY_EXPORTED		CSV 形式で収集されるメトリクスです。例: **defaultSelectedMetrics** = []int{2, 4, 5, 7, 8, 9, 13, 14, 17, 21, 24, 33, 35, 40, 43, 60}
ROUTER_METRICS_HAPROXY_BASE_SCRAPE_INTERVAL	5s	
ROUTER_METRICS_HAPROXY_TIMEOUT	5s	
ROUTER_METRICS_TYPE	haproxy	HAProxy ルーターのメトリクスを生成します (haproxy のみがサポートされている値です)。
ROUTER_OVERRIDE_DOMAINS		コンマ区切りのドメイン名リスト。ルーターのドメイン名がルートのホストと一致する場合には、ホスト名は無視され、**ROUTER_SUBDOMAIN** に定義されているパターンが使用されます。
ROUTER_OVERRIDE_HOSTNAME		**true** に設定されている場合、**ROUTER_SUBDOMAIN** のテンプレートのあるルートの spec.host 値を上書きします。
ROUTER_SERVICE_HTTPS_PORT	443	HTTPS 要求をリッスンするポートです。
ROUTER_SERVICE_HTTP_PORT	80	HTTP 要求をリッスンするポートです。
ROUTER_SERVICE_NAME	パブリック	ルーターがルートステータスで自らを識別する名前です。
ROUTER_CANONICAL_HOSTNAME		ルーターステータスに表示されるルーターの (オプションの) ホスト名です。

変数	デフォルト	説明
ROUTER_SERVICE_NAMESPACE		ルーターがルーターステータスで自らを識別する namespace です。**ROUTER_SERVICE_NAME** が使用されている場合は必須です。
ROUTER_SERVICE_NO_SNI_PORT	10443	一部のフロントエンドからバックエンドへの通信に使用される内部ポートです (以下を参照してください)。
ROUTER_SERVICE_SNI_PORT	10444	一部のフロントエンドからバックエンドへの通信に使用される内部ポートです (以下を参照してください)。
ROUTER_SUBDOMAIN		spec.host なしでルートのホスト名を生成するために使用されるテンプレートです (例: ${name}-${namespace}.myapps.mycompany.com)。
ROUTER_SYSLOG_ADDRESS		ログメッセージを送信するアドレスです。空の場合は無効になります。
ROUTER_SYSLOG_FORMAT		設定されている場合は、基盤のルーター実装で使用されるデフォルトのログ形式が上書きされます。この値は、基盤のルーター実装の仕様に従う必要があります。
ROUTER_TCP_BALANCE_SCHEME	ソース	パススルールートを行うための複数のエンドポイント用「負荷分散ストラテジー」。利用可能なオプションは **source**、**roundrobin** または **leastconn** です。
ROUTER_LOAD_BALANCE_ALGORITHM	leastconn	複数のエンドポイントを持つルート用の「負荷分散エンドポイント」。利用可能なオプションは、**source**、**roundrobin** および **leastconn** です。
ROUTE_LABELS		監視するルートに適用するラベルセレクターです。何も指定しない場合はすべてを意味します。
STATS_PASSWORD		ルーターの統計にアクセスするのに必要なパスワード (ルーターの実装がサポートする場合)
STATS_PORT		統計を公開するポート (ルーターの実装がサポートする場合)。設定されていない場合は統計は公開されません。
STATS_USERNAME		ルーターの統計にアクセスするために必要なユーザー名 (ルーターの実装がこれをサポートしている場合)。
TEMPLATE_FILE	**/var/lib/haproxy/conf/custom/haproxy-config-custom.template**	HAProxy テンプレートへのパス (コンテナーイメージ内)。

変数	デフォルト	説明
ROUTER_USE_PROXY_PROTOCOL		**true** または **TRUE** に設定されている場合には、HAProxy は受信接続がポート 80 と 443 上で **PROXY** プロトコルを使用していることを想定します。ソース IP アドレスは、ロードバランサーが Amazon ELB などのプロトコルをサポートする場合には、ロードバランサーをパススルーすることができます。
ROUTER_ALLOW_WILDCARD_ROUTES		**true** または **TRUE** が設定されている場合には、ルーターの受付チェックを合格するという **Subdomain** のワイルドカードポリシーが指定されたルートは、HAProxy ルーターがサービスを提供します。
ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK		namespace 所有権ポリシーを緩和するために **true** に設定されます。
ROUTER_STRICT_SNI		strict-sni
ROUTER_CIPHERS	intermediate	バインドでサポートされる暗号のセットを指定します。

注記

同じマシンで複数のルーターを実行する場合には、ルーターがリッスンするポート (**ROUTER_SERVICE_SNI_PORT** および **ROUTER_SERVICE_NO_SNI_PORT**) を変更する必要があります。これらのポートは、マシン上で一意であれば、何でも指定できます。また、これらのポートは外部には公開されません。

ルータータイムアウト変数

TimeUnits は数字、その後に単位を指定して表現します。**us** *(マイクロ秒)、**ms** (ミリ秒、デフォルト)、**s** (秒)、**m** (分)、**h** *(時間)、**d** (日)

正規表現: [1-9][0-9]*(us\|ms\|s\|m\|h\|d)

ROUTER_BACKEND_CHECK_INTERVAL	5000ms	バックエンドでの後続の liveness チェックの時間の長さ。
ROUTER_CLIENT_FIN_TIMEOUT	1s	クライアントがルートに接続する場合の TCP FIN タイムアウトの期間を制御します。接続切断のために送信された FIN が指定の時間内に応答されない場合には、HAProxy が接続を切断します。小さい値を設定し、ルーターでリソースをあまり使用していない場合には、リスクはありません。

ROUTER_DEFAULT_CLIENT_TIMEOUT	30s	クライアントがデータを確認するか、送信するための時間の長さ。
ROUTER_DEFAULT_CONNECT_TIMEOUT	5s	最大接続時間。
ROUTER_DEFAULT_SERVER_FIN_TIMEOUT	1s	ルーターからルートをバッキングする Pod の TCP FIN タイムアウトを制御します。
ROUTER_DEFAULT_SERVER_TIMEOUT	30s	サーバーがデータを確認するか、送信するための時間の長さ。
ROUTER_DEFAULT_TUNNEL_TIMEOUT	1h	TCP または WebSocket 接続が開放された状態で保つ時間数。websockets/tcp 接続がある場合 (および HAProxy が再読み込みされる度) に、以前の HAProxy プロセスが指定された時間分、開放された状態に保たれます。
ROUTER_SLOWLORIS_HTTP_KEEPALIVE	300s	新しい HTTP 要求が表示されるまで待機する最大時間を設定します。この値が低すぎる場合には、ブラウザーおよびアプリケーションの **keepalive** 値が低くなりすぎて、問題が発生する可能性があります。詳しい情報は以下の注記セクションを参照してください。
ROUTER_SLOWLORIS_TIMEOUT	10s	HTTP 要求の伝送にかかる時間。
RELOAD_INTERVAL	5s	新規の変更を許可するためにルーターの再ロードが許可される最小の頻度。
ROUTER_METRICS_HAPROXY_TIMEOUT	5s	HAProxy メトリクスの収集タイムアウト。

注記

有効なタイムアウト値には、想定した個別のタイムアウトではなく、特定の変数を合計した値に指定することができます。

例: **ROUTER_SLOWLORIS_HTTP_KEEPALIVE** は **timeout http-keep-alive** を調節し、デフォルトで **300s** に設定されていますが、haproxy は **tcp-request inspect-delay** も待機し、値は **5s** に設定されているので、今回の場合には、合計のタイムアウトは **300s** に **5s** を加えた値になります。

5.7.6. ロードバランシングストラテジー

ルートに複数のエンドポイントがある場合には、HAProxy は選択した負荷分散ストラテジーをもとに、エンドポイント間に要求を分散します。これは、セッションの最初の要求など、永続情報が利用できない場合に適用されます。

ストラテジーには以下のいずれか使用できます。

- **roundrobin**: 各エンドポイントは、加重に従い、順番に使用されます。これは、サーバーの処理が均等に分散される場合に最もスムーズで公平なアルゴリズムです。
- **leastconn**: 接続数が最も少ないエンドポイントが要求を受信します。接続数が最も少ないエンドポイントが複数ある場合には、ラウンドロビンが実行されます。LDAP、SQL、TSE など、セッションが非常に長い場合にこのアルゴリズムを使用してください。HTTP など、短いセッションを通常使用するプロトコルでの使用を目的とはしていません。
- **source**: ソース IP アドレスは、ハッシュ化され、実行中サーバーの合計加重で分割されて、どのサーバーが要求を受信するか指定します。これにより、サーバーが終了/起動しない限り、同じクライアント IP アドレスは常に同じサーバーに到達します。実行中のサーバー数が変化したことで、ハッシュの結果が変化した場合には、多くのクライアントが異なるサーバーに転送されます。このアルゴリズムは一般的にパススルールートで使用されます。

ROUTER_TCP_BALANCE_SCHEME 環境変数 はパススルールートのデフォルトストラテジーを設定します。**ROUTER_LOAD_BALANCE_ALGORITHM** 環境変数 は、残りのルートに対してルーターのデフォルトストラテジーを設定します。ルート固有のアノテーション、**haproxy.router.openshift.io/balance** を使用して、個別のルートを制御できます。

5.7.7. HAProxy Strict SNI

デフォルトでは、ホストは HTTPS または TLS SNI 要求のルートを解決しないので、デフォルト証明書が 503 応答の一部として、呼び出し元に返されます。これにより、デフォルト証明書が公開され、不正な証明書がサイトに提供されるので、セキュリティーの問題を引き起こす可能性があります。バインド用の HAProxy **strict-sni** オプションを使用するとデフォルト証明書の使用が抑制されます。

ROUTER_STRICT_SNI 環境変数はバインド処理を制御します。**true** または **TRUE** に設定されている場合には、**strict-sni** が HAProxy バインドに追加されます。デフォルト設定は **false** です。

このオプションは、ルーターの作成時または後の追加時に設定できます。

```
$ oc adm router --strict-sni
```

これは、**ROUTER_STRICT_SNI=true** を設定できます。

5.7.8. ルーターの暗号スイート

各クライアント (例: Chrome 30 または Java8) には、ルーターにセキュアに接続するために使用する暗号スイートが含まれます。ルーターには、接続を完了させるには、暗号化が最低でも 1 つ必要です。

表5.3 ルーター暗号プロファイル

プロファイル	互換性のある最も古いクライアント
modern	Firefox 27, Chrome 30, IE 11 on Windows 7, Edge, Opera 17, Safari 9, Android 5.0, Java 8
intermediate	Firefox 1, Chrome 1, IE 7, Opera 5, Safari 1, Windows XP IE8, Android 2.3, Java 7

プロファイル	互換性のある最も古いクライアント
old	Windows XP IE6, Java 6

詳細は、Security/Server Side TLS リファレンスガイドを参照してください。

ルーターは **intermediate** プロファイルにデフォルト設定されています。ルートを作成する時または、既存のルーターの **ROUTER_CIPHERS** を **modern**、**intermediate** または **old** に変更する時に、**--ciphers** オプションを使用して別のプロジェクトを選択します。または、":" 区切りで暗号化を指定することも可能です。暗号化は、以下のコマンドで表示されたセットの中から選択する必要があります。

```
openssl ciphers
```

5.7.9. ルートホスト名

OpenShift Container Platform ルートを使用してサービスと外部に到達可能なホスト名を関連付けることで、サービスを外部に公開することができます。このエッジホスト名は次に、サービスにトラフィックをルーティングするのに使用します。

異なる namespace から複数のルートが同じホストを要求する場合に、一番古いルートが優先され、その namespace にホストを獲得します。同じ namespace 内に、追加のルートが異なるパスフィールドで定義されている場合には、これらのパスが追加されます。複数のルートに同じパスが使用されている場合には、一番古いものが優先されます。

あるユーザーがホスト名にルート 2 つ (1 つが新しく、1 が古い) を指定しようとしていると仮定します。このユーザーがホスト名に他の 2 つのルートを指定する前に、別のユーザーが同じホスト名にルートを指定したうえに、元のユーザーにより作成済みのルートが削除された場合に、このホスト名への要求は効果がなくなります。他の namespace がこのホスト名を要求し、最初のユーザーの要求はなくなります。

例5.1 指定されたホストを持つルート:

```
apiVersion: v1
kind: Route
metadata:
  name: host-route
spec:
  host: www.example.com 1
  to:
    kind: Service
    name: service-name
```

1 サービスを公開するために使用される外部から到達可能なホスト名を指定します。

例5.2 ホスト内のルート:

```
apiVersion: v1
kind: Route
metadata:
  name: no-route-hostname
```

```
spec:
  to:
    kind: Service
    name: service-name
```

ホスト名がルート定義の一部として指定されていない場合には、OpenShift Container Platform が自動的に生成します。生成されたホスト名は以下のような形式をとります。

```
<route-name>[-<namespace>].<suffix>
```

以下の例では、namespace **mynamespace** にホストを追加せずに、上記のルート設定に対して OpenShift Container Platform が生成したホスト名を示します。

例5.3 生成されるホスト名

```
no-route-hostname-mynamespace.router.default.svc.cluster.local 1
```

1 生成されたホスト名のサフィックスは、デフォルトのルーティングサブドメイン **router.default.svc.cluster.local** です。

クラスター管理者は、環境に合わせて「デフォルトのルーティングサブドメインとして使用するサフィックスをカスタマイズ」することもできます。

5.7.10. ルートタイプ

ルートにセキュリティーを設定してもしなくても構いません。セキュアなルートは、複数の TLS 終端タイプを使用してクライアントに証明書を提供できます。ルーターは、edge、passthrough および re-encryption 終端をサポートします。

例5.4 セキュリティー保護されていないルートオブジェクト YAML 定義

```
apiVersion: v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name
```

セキュリティー保護されていないルートは、鍵や証明書が必要でないので、設定が最も単純ですが、セキュリティー保護されているルートは、接続を非公開のままにできるのでセキュリティーを確保できます。

Secured ルートは、ルートの TLS 終端が指定されたルートです。利用可能な終端タイプは、以下で説明されています。

5.7.10.1. パスベースのルート

パスベースのルートは、同じホスト名で、それぞれ異なるパスを使用して複数のルートにサービスを提供できるように、URL と比較可能なパスコンポーネントを指定します (ルートのトラフィックが HTTP ベースでなければならない)。ルーターは、最も限定的なものから順に、ルートを照合する必要がありますが、これはルーターの実装により左右されます。ホスト名とパスは、正常に要求に応答できるように、バックエンドサーバーにパススルーされます。たとえば、http://example.com/foo/ への要求がルーターに移動すると、Pod に http://example.com/foo/ への要求が表示されます。

以下の表は、ルートのサンプルおよびそれらのアクセシビリティーを示しています。

表5.4 ルートの可用性

ルート	比較対象	アクセス可能
www.example.com/test	**www.example.com/test**	Yes
	www.example.com	No
www.example.com/test and **www.example.com**	**www.example.com/test**	Yes
	www.example.com	Yes
www.example.com	**www.example.com/test**	はい (ルートではなく、ホストにマッチ)
	www.example.com	Yes

例5.5 パスが 1 つでセキュリティー保護されていないルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test" 1
  to:
    kind: Service
    name: service-name
```

1 パスは、パスベースのルートに唯一追加される属性です。

注記

ルーターは TLS を終了させず、要求のコンテンツを読み込みことができないので、パスベースのルーティングは、パススルー TLS を使用する場合には利用できません。

5.7.10.2. セキュリティー保護されたルート

セキュリティー保護されたルートは、ルートの TLS 終端を指定し、オプションで鍵と証明書を提供します。

注記

OpenShift Container Platform の TLS 終端は、カスタム証明書を提供する SNI に依存します。ポート 443 で受信する SNI 以外のトラフィックは、TLS 終端およびデフォルトの証明書で処理されます (要求のホスト名と一致せず、バリデーションエラーが発生する可能性があります)。

セキュリティー保護されたルートは、以下の 3 種類のセキュアな TLS 終端を使用できます。

Edge Termination

edge termination では、TLS 終端は、宛先にトラフィックをプロキシ化する前に、ルーターで発生します。TLS 証明書は、ルーターのフロントエンドで提供されるので、ルートに設定する必要があります。設定されていない場合には、「ルーターのデフォルトの証明書」が TLS 終端に使用されます。

例5.6 Edge Termination を使用したセキュリティー保護されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured (1)
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name (2)
  tls:
    termination: edge (3)
    key: |- (4)
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |- (5)
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |- (6)
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

(1)(2) オブジェクトの名前で、63 文字に制限されます。

(3) **termination** フィールドは edge termination の **edge** です。

(4) **key** フィールドは PEM 形式のキーファイルのコンテンツです。

(5) **certificate** フィールドは PEM 形式の証明書ファイルのコンテンツです。

(6) オプションの CA 証明書は、検証用に証明書チェーンを確立するために必要になる場合があります。

TLS がルーターで終端されるので、内部ネットワークを使用したルーターからエンドポイントへの接続は暗号化されません。

Edge termination ルートは **insecureEdgeTerminationPolicy** を指定して、セキュアでないスキーム (**HTTP**) 上にあるトラフィックを無効化、許可、リダイレクトすることができます。**insecureEdgeTerminationPolicy** で使用できる値は **None** または空 (無効化する場合)、**Allow** または **Redirect** です。デフォルトの **insecureEdgeTerminationPolicy** は、セキュアでないスキーム上のトラフィックを無効にします。一般的なユースケースは、セキュアなスキームを使用してコンテンツを、セキュアでないスキームを使用してアセット (例のイメージ、スタイルシート、javascript) を提供できるようにします。

例5.7 Edge Termination を使用したセキュリティー保護されたルートでの HTTP トラフィックの許可

```
apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured-allow-insecure (1)
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name (2)
  tls:
    termination:                edge (3)
    insecureEdgeTerminationPolicy: Allow (4)
    [ ... ]
```

(1) (2) オブジェクトの名前で、63 文字に制限されます。

(3) **termination** フィールドは edge termination の **edge** です。

(4) セキュアでないスキーム **HTTP** で送信される要求を許可するセキュアでないポリシー

例5.8 Edge Termination を使用したセキュリティー保護されたルートでの HTTP トラフィックのリダイレクト

```
apiVersion: v1
kind: Route
metadata:
  name: route-edge-secured-redirect-insecure (1)
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name (2)
  tls:
    termination:                edge (3)
    insecureEdgeTerminationPolicy: Redirect (4)
    [ ... ]
```

1 2 オブジェクトの名前で、63 文字に制限されます。

3 **termination** フィールドは edge termination の **edge** です。

4 セキュアでないスキーム **HTTP** で送信される要求をセキュアなスキーム **HTTPS** にリダイレクトするセキュアでないポリシー

パススルーの停止

passthrough termination では、暗号化されたトラフィックが TLS 終端を提供するルーターなしに宛先に直接送信されます。そのため、鍵や証明書は必要ありません。

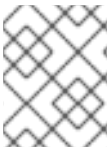
例5.9 パススルーの停止を使用したセキュリティー保護されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-passthrough-secured 1
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name 2
  tls:
    termination: passthrough 3
```

1 2 オブジェクトの名前で、63 文字に制限されます。

3 **termination** フィールド **passthrough** に設定します。他の暗号化フィールドは必要ありません。

宛先 Pod は、エンドポイントでトラフィックに証明書を提供します。これは、必須となるクライアント証明書をサポートするための唯一の方法です (相互認証とも呼ばれる)。



注記

Passthrough ルートには **insecureEdgeTerminationPolicy** を指定できます。唯一有効な値は**None** (無効化する場合は空) または **Redirect** です。

Re-encryption の停止

Re-encryption は、edge termination の一種で、ルーターが証明書を使用して TLS を終端し、異なる証明書が設定されている可能性のあるエンドポイントへの接続を再暗号化します。そのため、内部ネットワーなどを含め、接続の全パスが暗号化されています。 ルーターは、ヘルスチェックを使用して、ホストの信頼性を判断します。

例5.10 Re-Encrypt の停止を使用したセキュリティー保護されたルート

```
apiVersion: v1
kind: Route
```

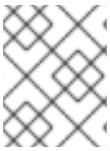
```
metadata:
  name: route-pt-secured (1)
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name (2)
  tls:
    termination: reencrypt (3)
    key: [as in edge termination]
    certificate: [as in edge termination]
    caCertificate: [as in edge termination]
    destinationCACertificate: |- (4)
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

(1)(2) オブジェクトの名前で、63 文字に制限されます。

(3) **termination** フィールドは **reencrypt** に設定あれます。他のフィールドは edge termination の場合と同じです。

(4) re-encryption には必須です。**destinationCACertificate** は、エンドポイント証明書を検証する CA 証明書を指定して、ルーターから宛先 Pod への接続のセキュリティーを確保します。サービスがサービス署名証明書を使用する場合または、管理者がデフォルトの CA 証明書をルーターに指定し、サービスにその CA により署名された証明書がある場合には、このフィールドは省略可能です。

destinationCACertificate フィールドが空の場合は、ルーターは自動的に証明書を提供するサービス用に生成される証明局を自動的に活用し、すべての Pod に **/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt** として注入します。これにより、ルートの証明書を生成する必要なしに、新しいルートがエンドツーエンドの暗号化を活用できるようになります。これは、管理者が許可しない限り、**destinationCACertificate** が使用できない、カスタムのルーターまたは F5 ルーターの場合に有用です。



注記

Re-encrypt ルートでは **insecureEdgeTerminationPolicy** に、edge termination ルートと同じ値にすべて指定することができます。

5.7.11. ルーターのシャード化

OpenShift Container Platform では、各ルートは **metadata** フィールドにいくつでも「labels」を指定できます。ルーターは **selectors** (**選択式** とも呼ばれる) を使用して、サービスを提供するルートの全プールからルートのサブセットを選択します。選択式でも、ルートの namespace でラベルを使用できます。選択したルートは、**ルーターのシャード** を形成します。ルートと分けて、ルーターシャードだけを「作成」して、「変更」できます。

この設計では、**従来の** シャード化も、**重複** シャード化をサポートします。従来のシャード化では、選択した内容が重複セットにならず、ルートはシャード 1 つのみに所属します。重複シャード化では、選択した内容は重複セットに含まれ、ルートは多数の異なるシャードに所属できます。たとえば、あるルートは **SLA=high** シャード (**SLA=medium** または **SLA=low** シャードではない) や **geo=west** シャード (**geo=east** シャードではない)に所属することができます。

重複シャード化の他の例には、ルートの namespace をベースに選択するルーターセットなどがあります。

ルーター	選択	Namespace
router-1	**A*** – **J***	**A***, **B***, **C***, **D***, **E***, **F***, **G***, **H***, **I***, **J***
router-2	**K*** – **T***	**K***, **L***, **M***, **N***, **O***, **P***, **Q***, **R***, **S***, **T***
router-3	**Q*** – **Z***	**Q***, **R***, **S***, **T***, **U***, **V***, **W***, **X***, **Y***, **Z***

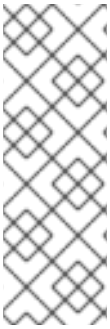
router-2 および **router-3** は、namespaces **Q***、**R***、**S***、**T*** のルートにサービスを提供します。この例を重複から従来のシャード化に変更するには、**router-2** の選択肢を **K*** – **P*** に変更して、重複をなくすことができます。

ルートがシャード化されている場合には、指定のルートはこのグループのルーター 0 個以上にバインドされます。ルートをバインドすることで、シャード全体でルートを一意に保つことができます。一意に保つことで、単一のシャード内に、同じルートでもセキュアなバージョンと、セキュアでないバージョンを存在させることができます。つまり、ルートは、作成、バインド、アクティブ化のライフサイクルが可視化されたことになります。

シャード化の環境では、シャードに到達する最初のルートが再起動の有無に拘わらず、期限なしに存在できる権利を持ちます。

green/blue デプロイメント時には、ルートは複数のルーターに選択される場合があります。OpenShift Container Platform のアプリケーション管理者は、別のバージョンのアプリケーションにトラフィックをフラッシュして、以前のバージョンをオフに設定する場合があります。

シャーディングは、管理者によりクラスターレベルで、ユーザーにより、プロジェクト/namespace レベルで実行できます。namespace ラベルを使用する場合には、ルーターのサービスアカウントには、**cluster-reader** パーミッションを設定して、ルーターが namespace 内のラベルにアクセスできるようにします。



注記

同じホスト名を要求するルートが 2 つ以上ある場合には、解決する順番は、ルートの存在期間をもとにし、一番古いルートがホストの要求を優先的に取得します。シャード化されたルーターの場合には、ルートは、ルーターの選択基準にあったラベルをベースに選択されます。ラベルがルートに追加されるタイミングを判断する方法に一貫性はありません。そのため、既存のホスト名を要求する以前のルートが「再度ラベル化されて」、ルーターの選択基準と照合させる場合には、上述の解決順に基づき既存のルートを置き換えます (最も古いルートが優先される)。

5.7.12. 他のバックエンドおよび重み

ルートは通常、**kind: Service** の **to:** トークンを使用したサービスと関連付けられます。ルートへの全要求は、「負荷分散ストラテジー」をベースに、サービス内のエンドポイントにより処理されます。

サービスは最大 4 つまでルートをサポートすることができます。各サービスが処理する要求の大きさは、サービスの **weight** により統制されます。

最初のサービスは、以前と同様に **to:** トークンを使用して入り、サービスは 3 つまで **alternateBackend:** トークンを使用して入ることができます。各サービスは、デフォルトの **kind: Service** が指定されている必要があります。

各サービスには、**weight** が関連付けられています。サービスが処理する要求の大きさは、**weight / sum_of_all_weights** で算出されます。サービスにエンドポイントが複数ある場合には、サービスの加重が 1 以上、各エンドポイントに割り当てられるように、エンドポイント全体に分散されます。サービスの **weight** が 0 の場合は、サービスの各エンドポイントには 0 が割り当てられます。

weight は、0-256 の範囲内で指定してください。デフォルトは 1 です。**weight** が 0 の場合は、サービスに要求は渡されません。全サービスの **weight** が 0 の場合は、要求に対して 503 エラーが返されます。サービスにエンドポイントがない場合には、加重は実際には 0 となります。

alternateBackends を使用すると、**roundrobin** 負荷分散ストラテジーを使用して、要求が想定どおりに **weight** をもとにサービスに分散されるようになります。ルートに **roundrobin** を設定する場合は、「ルートアノテーション」を使用するか、一般的なルーターには環境変数を使用します。

以下は、「A/B デプロイメント」向けに別のバックエンドを使用したルート設定例です。

alternateBackends および加重が指定されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-alternate-service
  annotations:
    haproxy.router.openshift.io/balance: roundrobin (1)
spec:
  host: www.example.com
  to:
    kind: Service
    name: service-name (2)
    weight: 20 (3)
  alternateBackends:
  - kind: Service
    name: service-name2 (4)
    weight: 10 (5)
    kind: Service
    name: service-name3 (6)
    weight: 10 (7)
```

(1) このルートは**roundrobin** 「負荷分散ストラテジー」を使用します。

(2) 最初のサービス名は **service-name** であり、0 以上の Pod が含まれる可能性があります。

(4) (6) alternateBackend サービスにも 0 以上の Pod が含まれる場合があります。

(3) (5) (7) **weight** の合計は 40 で、 **service-name** は要求の 20/40 または 1/2を、 **service-name2** と **service-name3** はそれぞれ、要求の 1/4 を取得し、サービス毎に 1 または複数のエンドポイントが含まれると想定します。

== ルート固有のアノテーション

環境変数を使用して、ルーターは、公開する全ルートにデフォルトオプションを設定できます。個別のルートは、アノテーションに個別の設定を指定して、デフォルトの一部を上書きできます。

ルートアノテーション

このセクションで説明されているすべての項目について、ルートがその設定を変更できるように **ルート定義** にアノテーションを設定できます。

表5.5 ルートアノテーション

変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/balance	ロードバランシングアルゴリズムを設定します。使用できるオプションは **source**、**roundrobin**、および **leastconn** です。	passthrough ルートの **ROUTER_TCP_BALANCE_SCHEME** です。それ以外の場合は **ROUTER_LOAD_BALANCE_ALGORITHM** を使用します。
haproxy.router.openshift.io/disable_cookies	関連の接続を追跡する cookie の使用を無効にします。**true** または **TRUE** に設定する場合は、分散アルゴリズムを使用して、受信する HTTP 要求ごとに、どのバックエンドが接続を提供するかを選択します。	
haproxy.router.openshift.io/cookie_name	このルートに使用するオプションの cookie を指定します。名前は、大文字、小文字、数字、"_" または "-" を任意に組み合わせて指定する必要があります。デフォルトは、ルートのハッシュ化された内部キー名です。	
haproxy.router.openshift.io/pod-concurrent-connections	ルーターからバッキングされる Pod に対して許容される接続最大数を設定します。 注意: Pod が複数ある場合には、それぞれに対応する接続数を設定できますが、ルーターが複数ある場合には、ルーター間の連携がなく、それぞれの接続回数はルーターの数と同じとなります。設定されていない場合または 0 に設定されている場合には制限はありません。	
haproxy.router.openshift.io/rate-limit-connections	レート制限機能を有効にするために **true** または **TRUE** を設定します。	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	IP アドレスで共有される同時 TCP 接続の数を制限します。	

変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/rate-limit-connections.rate-http	IP アドレスが HTTP 要求を実行できるレートを制限します。	
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	IP アドレスが TCP 接続を行うレートを制限します。	
haproxy.router.openshift.io/timeout	ルートのサーバー側のタイムアウトを設定します。(TimeUnits)	**ROUTER_DEFAULT_SERVER_TIMEOUT**
router.openshift.io/haproxy.health.check.interval	バックエンドのヘルスチェックの間隔を設定します。(TimeUnits)	**ROUTER_BACKEND_CHECK_INTERVAL**
haproxy.router.openshift.io/ip_whitelist	ルートのホワイトリストを設定します。	
haproxy.router.openshift.io/hsts_header	edge terminated または re-encrypt ルートの Strick-Transport-Security ヘッダーを設定します。	

例5.11 ルート設定のカスタムタイムアウト

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms (1)
[...]
```

(1) HAProxy 対応の単位 (us、ms、s、m、h、d) で新規のタイムアウトを指定します。単位が指定されていない場合は、ms がデフォルトになります。

注記

passthrough ルートのサーバー側のタイムアウトを低く設定し過ぎると、WebSocket 接続がそのルートで頻繁にタイムアウトする可能性があります。

5.7.13. ルート固有の IP ホワイトリスト

選択した IP アドレスだけにルートへのアクセスを制限するには、ルートに **haproxy.router.openshift.io/ip_whitelist** アノテーションを追加します。ホワイトリストは、承認したソースアドレスの IP アドレスまたは/および CIDR をスペース区切りにします。ホワイトリストに含まれていない IP アドレスからの要求は破棄されます。

例:

ルートの編集時に、以下のアノテーションを追加して必要なソース IP を定義します。または、**oc annotate route <name>** を使用します。

唯一の特定の IP アドレスのみを許可します。

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

複数の IP アドレスを許可します。

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12
```

IP CIDR ネットワークを許可します。

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

混在した IP アドレスおよび IP CIDR ネットワークを許可します。

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

5.7.14. ワイルドカードサブドメインポリシーを指定するルートの作成

ワイルドカードポリシーでは、(許可できるようにルーターを設定する場合) ドメイン内の全ホストに対応するルートを定義できます。ルートは、**wildcardPolicy** フィールドを使用して、設定の一部としてワイルドカードポリシーを指定できます。ワイルドカードルートを許可するポリシーが指定されたルーターは、ワイルドカードポリシーをもとに適切にルートを公開します。

「ワイルドカードルートを許可するように HAProxy ルートを設定する方法についてはこちら」を参照してください。

例5.12 サブドメインワイルドカードポリシーを指定するルート

```
apiVersion: v1
kind: Route
spec:
  host: wildcard.example.com (1)
  wildcardPolicy: Subdomain (2)
  to:
    kind: Service
    name: service-name
```

(1) サービスを公開するために使用される外部から到達可能なホスト名を指定します。

(2) 外部から到達可能なホスト名は、サブドメイン **example.com** 内の全ホストを許可するように指定します。***.example.com** は、公開されたサービスに到達するためのホスト名 **wildcard.example.com** のサブドメインです。

5.7.15. ルートステータス

route status フィールドは、ルーターでのみ設定されます。ルーターが特定のルーターへのサービス提供を停止するように、ルートに変更が加えられた場合には、スターテスが古くなってしまいますが、ルーターは、**route status** フィールドは消去しません。ルートステータスの以前のエントリーを削除するには、clear-route-status script を使用してください。

5.7.16. ルート内の特定ドメインの拒否または許可

ルーターは、**ROUTER_DENIED_DOMAINS** および **ROUTER_ALLOWED_DOMAINS** 環境変数を使用して、ルート内のホスト名からのドメインサブセットを限定して拒否または許可するように設定できます。

ROUTER_DENIED_DOMAINS	一覧表示されるドメインは指定のルートで許可されません。
ROUTER_ALLOWED_DOMAINS	一覧表示されるドメインのみが指定のルートで許可されます。

拒否ドメインの一覧に含まれるドメインは、許可ドメイン一覧よりも優先されます。つまり、OpenShift Container Platform は先に、拒否リスト (該当する場合) をチェックして、ホスト名が拒否ドメイン一覧に含まれていない場合に、許可ドメインをチェックします。ただし、許可ドメインの一覧はより制限されており、ルーターは、その一覧に所属するホストが含まれるルートのみを許可します。

たとえば、**myrouter** ルートの **[*.]open.header.test**、**[*.]openshift.org** および **[*.]block.it** ルートを拒否するには、以下を実行します。

```
$ oc adm router myrouter ...
$ oc set env dc/myrouter ROUTER_DENIED_DOMAINS="open.header.test, openshift.org, block.it"
```

これは、**myrouter** がルートの名前に基づいて以下を許可することを意味します。

```
$ oc expose service/<name> --hostname="foo.header.test"
$ oc expose service/<name> --hostname="www.allow.it"
$ oc expose service/<name> --hostname="www.openshift.test"
```

ただし、**myrouter** は以下を拒否します。

```
$ oc expose service/<name> --hostname="open.header.test"
$ oc expose service/<name> --hostname="www.open.header.test"
$ oc expose service/<name> --hostname="block.it"
$ oc expose service/<name> --hostname="franco.baresi.block.it"
$ oc expose service/<name> --hostname="openshift.org"
$ oc expose service/<name> --hostname="api.openshift.org"
```

または、ホスト名が **[*.]stickshift.org** または **[*.]kates.net** に設定されて **いない** ルートをブロックするには、以下を実行します。

```
$ oc adm router myrouter ...
$ oc set env dc/myrouter ROUTER_ALLOWED_DOMAINS="stickshift.org, kates.net"
```

-

これは、**myrouter** ルートが以下を許可することを意味します。

```
$ oc expose service/<name> --hostname="stickshift.org"
$ oc expose service/<name> --hostname="www.stickshift.org"
$ oc expose service/<name> --hostname="kates.net"
$ oc expose service/<name> --hostname="api.kates.net"
$ oc expose service/<name> --hostname="erno.r.kube.kates.net"
```

ただし、**myrouter** は以下を拒否します。

```
$ oc expose service/<name> --hostname="www.open.header.test"
$ oc expose service/<name> --hostname="drive.ottomatic.org"
$ oc expose service/<name> --hostname="www.wayless.com"
$ oc expose service/<name> --hostname="www.deny.it"
```

両方のシナリオを実装するには、以下を実行します。

```
$ oc adm router adrouter ...
$ oc env dc/adrouter ROUTER_ALLOWED_DOMAINS="openshift.org, kates.net" \
    ROUTER_DENIED_DOMAINS="ops.openshift.org, metrics.kates.net"
```

これにより、ホスト名が **[*.]openshift.org** または **[*.]kates.net** に設定されているルートを許可し、ホスト名が **[*.]ops.openshift.org** または **[*.]metrics.kates.net** に設定されているルートは拒否します。

そのため、以下は拒否されます。

```
$ oc expose service/<name> --hostname="www.open.header.test"
$ oc expose service/<name> --hostname="ops.openshift.org"
$ oc expose service/<name> --hostname="log.ops.openshift.org"
$ oc expose service/<name> --hostname="www.block.it"
$ oc expose service/<name> --hostname="metrics.kates.net"
$ oc expose service/<name> --hostname="int.metrics.kates.net"
```

しかし、以下は許可されます。

```
$ oc expose service/<name> --hostname="openshift.org"
$ oc expose service/<name> --hostname="api.openshift.org"
$ oc expose service/<name> --hostname="m.api.openshift.org"
$ oc expose service/<name> --hostname="kates.net"
$ oc expose service/<name> --hostname="api.kates.net"
```

5.7.17. Kubernetes Ingress オブジェクトのサポート

Kubernetes Ingress オブジェクトは受信接続が内部サービスに到達する方法を判別する設定オブジェクトです。OpenShift Container Platform では OpenShift Container Platform のバージョン 3.10 より、Ingress コントローラー設定ファイルを使用したこれらのオブジェクトのサポートがあります。

コントローラーは、Ingress オブジェクトを監視して、1 つまたは複数のルートを作成し、Ingress オブジェクトの条件を満たします。コントローラーは、Ingress オブジェクトと、生成されたルートオブジェクトが常に同期されるようにします。たとえば、Ingress オブジェクトに関連付けられたシークレットで生成されたルートパーミッションを付与するなどです。

たとえば、以下のように設定された Ingress オブジェクトの場合:

```
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: test
spec:
  rules:
  - host: test.com
    http:
      paths:
      - path: /test
        backend:
          serviceName: test-1
          servicePort: 80
```

以下のルートオブジェクトが生成されます。

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: test-a34th (1)
  ownerReferences:
  - apiVersion: extensions/v1beta1
    kind: Ingress
    name: test
    controller: true
spec:
  host: test.com
  path: /test
  to:
    name: test-1
  port:
     targetPort: 80
```

(1) 名前は、Ingress 名をプレフィックスとして使用して、ルートオブジェクトにより生成されます。

注記

ルートを作成するには、Ingress オブジェクトにホスト、サービス、パスが含まれる必要があります。

5.7.18. Namespace 所有権チェックの無効化

ホストとサブドメインは、最初に要求を作成したルートの namespace が所有します。その namespace で作成された他のルートは、サブドメイン上で要求を作成できます。他の namespace はすべて、請求済みのホストおよびサブドメインに要求を作成することはできません。ホストを所有する namespace は、**www.abc.xyz/path1** など、そのホストに関連付けられている全パスも所有します。

たとえば、ホスト **www.abc.xyz** がどのルートからも要求されていない場合に、**ns1** namespace にホストが **www.abc.xyz** のルート **r1** を作成すると、**ns1** の namespace が、ワイルドカードルートのホスト **www.abc.xyz** と サブドメイン **abc.xyz** を所有します。別の namespace **ns2** が異なるパス **www.abc.xyz/path1/path2** で、ルートを作成しようとすると、別の namespace (今回は **ns1**) のルートがこのホストを所有するので失敗してしまいます。

「ワイルドカードルート」では、サブドメインを所有する namespace はそのサブドメインに含まれるホストすべてを所有します。上記の例のように、namespace が **abc.xyz** というサブドメインを所有する場合には、別の namespace は **z.abc.xyz** を要求できません。

namespace の所有権ルールを無効にするには、これらの制限を無効にして、namespace すべてでホスト (およびサブドメイン) を要求できるようにします。

警告

ルーターで namespace の所有権チェックを無効にする場合には、エンドユーザーが namespace に含まれるホストの所有権を要求できるようになる点に注意してください。この設定の変更は、特定の開発環境で価値がありますが、実稼働環境では慎重にこの機能を使用し、クラスターポリシーで、信頼されないエンドユーザーがルートを作成できないようにロックしてください。

たとえば、 **ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true** では、namespace **ns1** が最も古い **r1 www.abc.xyz** を作成する場合に、このホスト名 (+ パス) のみを所有します。別の namespace は、このサブドメイン (**abc.xyz**) に最も古いルートがなくても、ワイルドカードルートを作成することができ、別の namespace が (**foo.abc.xyz**、**bar.abc.xyz**、**baz.abc.xyz** など) ワイルドカード以外の重複ホストを要求することも可能で、この要求は認められます。

別の namespace (例: **ns2**) はルート **r2 www.abc.xyz/p1/p2** を作成でき、許可されます。同様に、別の namespace (**ns3**) は、サブドメインのワイルドカードポリシーでルート **wildthing.abc.xyz** も作成でき、このワイルドカードを所有できます。

この例にあるように、ポリシー **ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true** は制約がゆるく、namespace 全体の要求を許可します。ルーターが namespace の所有を無効にしているルートを許可できるのは、ホストとパスがすでに請求済みである場合だけです。

たとえば、新規ルート **rx** が **www.abc.xyz/p1/p2** を要求しようとする場合に、ルート **r2** はホストとパスの組み合わせを所有しているので拒否されます。まったく同じホストとパスがすでに請求済みなので、これは、ルート **rx** が同じ namespace にある場合も、別の namespace にある場合でも同じです。

この機能は、ルーターの作成時か、またはルーターのデプロイメント設定に環境変数を設定して設定できます。

```
$ oc adm router ... --disable-namespace-ownership-check=true
```

```
$ oc env dc/router ROUTER_DISABLE_NAMESPACE_OWNERSHIP_CHECK=true
```

[1] これ以降は、デバイス名は、コンテナー B のホストのデバイスを参照します。

第6章 サービスカタログコンポーネント

6.1. サービスカタログ

6.1.1. 概要

マイクロサービスベースのアプリケーションを開発して、クラウドネイティブのプラットフォームで実行する場合には、サービスプロバイダーやプラットフォームに合わせて、異なるリソースをプロビジョンして、その従属、認証情報、設定を共有する方法は多数あります。

開発者がよりシームレスに作業できるように、OpenShift Container Platform には Kubernetes 向けの Open Service Broker API (OSB API) の実装である **サービスカタログ** が含まれています。これにより、OpenShift Container Platform にデプロイされているアプリケーションを、さまざまな種類のサービスブローカーに接続できるようになります。

サービスカタログでは、クラスター管理者が 1 つの API 仕様を使用して、複数のプラットフォームを統合できます。OpenShift Container Platform web コンソールは、サービスカタログで提供されるサービスブローカーが提供するクラスターサービスカタログを表示するので、これらのサービスを検出、インスタンス化して、アプリケーションで使用できるようにします。

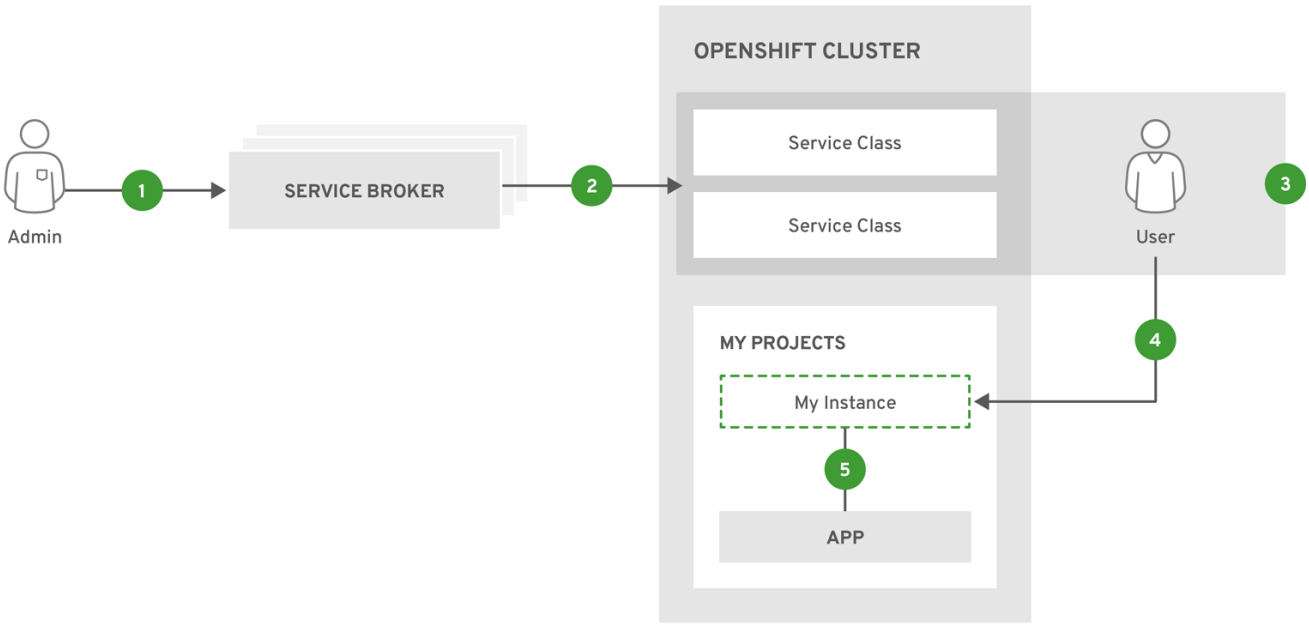
このように、サービスユーザーは異なるプロバイダーが提供する異なるタイプのサービスを簡単かつ一貫性を保ちながら使用できるという利点が得られます。また、サービスプロバイダーは、複数のプラットフォームにアクセスできる統合ポイントから利点を得られます。

6.1.2. 設計

サービスカタログの設計は基本的なワークフローに基づいています。

注記

以下の新規の用語は「概念および用語」でさらに定義されています。



クラスター管理者は、1つまたは複数の **クラスターサービスブローカー** を OpenShift Container Platform クラスターに登録します。これは、デフォルトで提供されているサービスブローカーでインストール時に自動的に行うことも、手動で行うことも可能です。

各サービスブローカーは、ユーザーに提供すべき **クラスターサービスクラス** セットと、これらのサービスのバリエーション (**サービスプラン**) を、OpenShift Container Platform に指定します。

OpenShift Container Platform web コンソールまたは CLI を使用して、ユーザーは利用可能なサービスを検出します。たとえば、クラスターサービスクラスは、BestDataBase と呼ばれる database-as-a-service などの、サービスクラスが利用できる可能性があります。

ユーザーは、クラスターサービスクラスを選択して、自身の新しい **インスタンス** を要求します。たとえば、サービスは **my_db**という名前の BestDataBase インスタンスなどです。

ユーザーは、サービスを pod セット (アプリケーション) にリンクまたは **バインド** します。たとえば、**my_db** サービスインスタンスは、**my_app** と呼ばれるユーザーアプリケーションにバインドできます。

ユーザーがリソースのプロビジョニングおよびプロビジョニング解除を要求すると、この要求はサービスカタログに送信され、次に適切なクラスターサービスブローカーに要求が送信されます。サービスによっては、**provision**、**deprovision** および **update** などの操作に時間がかかる可能性があります。クラスターサービスブローカーが利用できない場合には、サービスカタログはこの操作の再試行を続けます。

このインフラストラクチャーでは、OpenShift Container Platform で実行中のアプリケーションと、それが使用するサービスの間で疎結合することができます。こうすることで、これらのサービスを使用するアプリケーションが独自のビジネスロジックにフォーカスし、サービスの管理をプロバイダーに任せることができます。

6.1.2.1. リソースの定義

ユーザーがサービスを使用し終えた場合 (または、請求しなくてもいい場合) には、このサービスインスタンスは削除できます。サービスインスタンスを削除するには、サービスのバインドを先に削除する必要があります。サービスのバインドの削除は、**バインド解除** と呼ばれます。削除のプロセスで、削除予定のサービスバインディングを参照するシークレットも削除します。

サービスバインディングが削除されると、サービスインスタンスを削除できあす。サービスインスタンスの削除は **プロビジョニング解除** として知られています。

サービスバインディングやサービスインスタンスが含まれるプロジェクトや namespace を削除する場合は、サービスカタログは先にクラスターサービスブローカーに、関連のインスタンスとバインディングを削除するように要求する必要があります。これにより、サービスカタログは、クラスターのサービスブローカーと通信して、プロビジョニング解除を行うまで待つ必要があるので、実際のプロジェクトや namespace の削除が遅延してしまうことが予想されます。通常の状況では、サービスにより異なりますが、数分以上かかる場合があります。

注記

デプロイメントで使用されるサービスバインディングを削除する場合、デプロイメントからバインディングの参照を削除する必要もあります。そうしないと、次のロールアウトは失敗します。

6.1.3. 概念および用語

クラスターサービスブローカー

クラスターサービスブローカー は、OSB API 仕様に準拠し、1つ以上のサービスのセットを管理するサーバーです。ソフトウェアが独自の OpenShift Container Platform クラスター内またはその他の場所でホストされる可能性があります。
クラスター管理者は、クラスターサービスブローカーを表す **ClusterServiceBroker** API リソースを作成して、OpenShift Container Platform クラスターに登録できます。これにより、クラスター管理者はクラスター内で利用可能なクラスターサービスブローカーを使用して新しい種類の管理済みサービスを作成できるようになります。

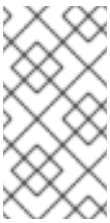
ClusterServiceBroker リソースは、クラスターサービスブローカーの接続詳細と、サービスセット (およびこれらのサービスのバリエーション) を OpenShift Container Platform に指定すると、ユーザーに提供できるはずです。**authInfo** セクションには、クラスターサービスブローカーの認証に使用するデータが含まれている点に特に注意してください。

ClusterServiceBroker リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ClusterServiceBroker
metadata:
  name: BestCompanySaaS
spec:
  url: http://bestdatabase.example.com
  authInfo:
    basic:
      secretRef:
        namespace: test-ns
        name: secret-name
```

クラスターサービスクラス

また、サービスカタログのコンテキストで「サービス」と類義の **クラスターサービスクラス** は、特定のクラスターサービスブローカーが提供する管理サービスの1種です。新しいクラスターサービスブローカーのリソースがクラスターに追加されるたびに、サービスカタログコントローラーは、適切なクラスターサービスブローカーに接続し、サービスオファリングの一覧を取得します。新しい **ClusterServiceClass** リソースは自動的に、クラスターサービスブローカー毎に作成されます。



注記

さらに、OpenShift Container Platform には、「サービス」と呼ばれるコアとなるコンセプトがあります。サービスとは、内部の負荷分散に関連する別個の Kubernetes リソースです。これらのリソースは、サービスカタログや OSB API のコンテキストで使用する用語と混同しないようにしてください。

ClusterServiceClass リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ClusterServiceClass
metadata:
  name: smallDB
  brokerName: BestDataBase
  plans: [...]
```

クラスターサービス計画

クラスターサービスプラン は、クラスターサービスクラスの階層を指します。たとえば、クラスターサービスクラスは、さまざまなレベルの quality-of-service (QoS) を提供するプランを公開し、それぞれに異なるコストが関連付けられています。

サービスインスタンス

サービスインスタンス は、プロビジョニングされたクラスターサービスクラスのインスタンスです。サービスクラスが提供する機能を使用する場合には、新しいサービスインスタンスを作成してください。
新規の **ServiceInstance** リソースを作成した場合には、サービスカタログコントローラーは、適切なクラスターサービスブローカーと接続して、サービスインスタンスをプロビジョニングするように指示を出します。

ServiceInstance リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceInstance
metadata:
  name: my_db
  namespace: test-ns
spec:
  externalClusterServiceClassName: smallDB
  externalClusterServicePlanName: default
```

アプリケーション

アプリケーション という用語は、**サービスインスタンス** を使用する ユーザーのプロジェクトで実行中の pod など、OpenShift Container Platform デプロイメントのアーティファクトを指します。

認証情報

認証情報 とは、サービスインスタンスと通信するアプリケーションで必要な情報です。

サービスバインディング

サービスバインディング は、サービスインスタンスとアプリケーションの間のリンクを指します。サービスバインディングは、アプリケーションからサービスインスタンスを参照して使用できるように希望するクラスターユーザーが作成します。
サービスバインディングの作成時に、サービスカタログコントローラーはサービスインスタンスの接続情報と認証情報を含む Kubernetes のシークレットを作成します。これらのシークレットは通常どおり Pod にマウントできます。また、**PodPresets** とも統合され、これでシークレットの使用方法や使用する pod などを表現できます。

ServiceBinding リソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceBinding
metadata:
  name: myBinding
  namespace: test-ns
spec:
  instanceRef:
    name: my_db
  parameters:
    securityLevel: confidential
  secretName: mySecret
```

parameters

パラメーター は、サービスバインディングまたはサービスインスタンスの使用時に、追加のデータをクラスターサービスブローカーに渡すために提供されている特別なフィールドです。唯一のフォーマット要件は、パラメーターを有効な YAML (または JSON) で指定することです。上記の例では、セキュリティーレベルのパラメーターをサービスバインディング要求でクラスターサービスブローカーに渡します。より高度なセキュリティーが必要なパラメーターの場合は、パラメーターをシークレットに配置し、**parametersFrom** を使用して参照します。

シークレットを参照するサービスバインディングリソースの例

```
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceBinding
metadata:
  name: myBinding
  namespace: test-ns
spec:
  instanceRef:
    name: my_db
  parametersFrom:
    - secretKeyRef:
        name: securityLevel
        key: myKey
  secretName: mySecret
```

6.1.4. 提供されるクラスターサービスブローカー

OpenShift Container Platform は、サービスカタログで使用する以下のクラスターサービスブローカーを提供します。

- Template Service Broker
- OpenShift Ansible Broker

6.2. サービスカタログのコマンドラインインターフェース (CLI)

6.2.1. 概要

サービスカタログと対話する「基本的なワークフロー」は以下のとおりです。

- クラスター管理者は、ブローカーサーバーをインストールして登録し、ブローカーサーバーのサービスを利用できるようにします。
- ユーザーは、OpenShift プロジェクトでこれらをインスタンス化して、サービスインスタンスとその Pod をリンクすることで、このようなサービスを使用します。

svcat というサービスカタログコマンドラインインターフェース (CLI) ユーティリティーは、このようなユーザー関連のタスクに対応するために提供されています。**oc** コマンドは同様のタスクを実行できますが、**svcat** を使用して、サービスカタログのリソースとの対話を簡素化することができます。**svcat** は、OpenShift クラスター上に集約された API エンドポイントを使用してサービス API と通信します。

6.2.2. svcat のインストール

お使いの Red Hat アカウントに有効な OpenShift Enterprise サブスクリプションがある場合には、Red Hat Subscription Management (RHSM) を使用して RPM として **svcat** をインストールできます。

```
# yum install atomic-enterprise-service-catalog-svcat
```

6.2.2.1. クラウドプロバイダーの留意点

Google Compute Engine: Google Cloud Platform では、以下のコマンドを使用して、受信トラフィックを許可するファイアウォールルーツを設定します。

```
$ gcloud compute firewall-rules create allow-service-catalog-secure --allow tcp:30443 --description
"Allow incoming traffic on 30443 port."
```

6.2.3. svcat の使用

以下のセクションでは、「サービスカタログのワークフロー」に記載のユーザー関連タスクを処理するための一般的なコマンドについて紹介します。**svcat --help** コマンドを使用して詳細情報を取得し、他に利用可能なコマンドオプションを表示できます。このセクションでの出力例は、Ansible Service Broker がすでにクラスターにインストールされていることが前提です。

6.2.3.1. ブローカーの詳細取得

利用可能なブローカー一覧の表示、ブローカーカタログの動機、サービスカタログにデプロイされたブローカーの詳細の取得が可能です。

6.2.3.1.1. ブローカーの検索

クラスターにインストールされた全ブローカーを表示します。

```
$ svcat get brokers
         NAME                                        URL                                    STATUS
+------------------------+-----------------------------------------------------------------------------------+--
------+
  ansible-service-broker    https://asb.openshift-ansible-service-broker.svc:1338/ansible-service-broker
Ready
  template-service-broker   https://apiserver.openshift-template-service-
broker.svc:443/brokers/template.openshift.io   Ready
```

6.2.3.1.2. ブローカーカタログの同期

ブローカーからのカタログメタデータを更新します。

```
$ svcat sync broker ansible-service-broker
Synchronization requested for broker: ansible-service-broker
```

6.2.3.1.3. ブローカーの詳細表示

ブローカーの詳細を表示します。

```
$ svcat describe broker ansible-service-broker
  Name:     ansible-service-broker
  URL:      https://openshift-automation-service-broker.openshift-automation-service-
```

```
broker.svc:1338/openshift-automation-service-broker/
  Status:   Ready - Successfully fetched catalog entries from broker @ 2018-06-07 00:32:59 +0000
UTC
```

6.2.3.2. サービスクラスおよびサービスプランの表示

ClusterServiceBroker リソースの作成時に、サービスカタログコントローラーはブローカーサーバーに対してクエリーを実行して提供する全サービスを検索して、それらのサービスごとにサービスクラス (**ClusterServiceClass**) を作成します。さらに、ブローカーのサービスごとにサービスプラン (**ClusterServicePlan**) も作成します。

6.2.3.2.1. サービスクラスの表示

利用可能な ClusterServiceClass リソースを表示します。

```
$ svcat get classes
       NAME                  DESCRIPTION
+-------------------+--------------------------------+
  rh-mediawiki-apb    Mediawiki apb implementation

  ...

  rh-mariadb-apb      Mariadb apb implementation
  rh-mysql-apb        Software Collections MySQL APB
  rh-postgresql-apb   SCL PostgreSQL apb
                      implementation
```

サービスクラスの詳細を表示します。

```
$ svcat describe class rh-postgresql-apb
  Name:          rh-postgresql-apb
  Description:   SCL PostgreSQL apb implementation
  UUID:          d5915e05b253df421efe6e41fb6a66ba
  Status:        Active
  Tags:          database, postgresql
  Broker:        ansible-service-broker

Plans:
  NAME            DESCRIPTION
+------+--------------------------------+
  prod   A single DB server with
         persistent storage
  dev    A single DB server with no
         storage
```

6.2.3.2.2. サービスプランの表示

クラスターで利用可能な ClusterServicePlan リソースを表示します。

```
$ svcat get plans
   NAME          CLASS                 DESCRIPTION
+---------+-------------------+--------------------------------+
  default   rh-mediawiki-apb    An APB that deploys MediaWiki
```

```
...

prod      rh-mariadb-apb      This plan deploys a single
                              MariaDB instance with 10 GiB
                              of persistent storage
dev       rh-mariadb-apb      This plan deploys a single
                              MariaDB instance with
                              ephemeral storage
prod      rh-mysql-apb        A MySQL server with persistent
                              storage
dev       rh-mysql-apb        A MySQL server with ephemeral
                              storage
prod      rh-postgresql-apb   A single DB server with
                              persistent storage
dev       rh-postgresql-apb   A single DB server with no
                              storage
```

プランの詳細を表示します。

```
$ svcat describe plan rh-postgresql-apb/dev
  Name:          dev
  Description:   A single DB server with no storage
  UUID:          9783fc2e859f9179833a7dd003baa841
  Status:        Active
  Free:          true
  Class:         rh-postgresql-apb

Instances:
No instances defined

Instance Create Parameter Schema:
  $schema: http://json-schema.org/draft-04/schema
  additionalProperties: false
  properties:
    postgresql_database:
      default: admin
      pattern: ^[a-zA-Z_][a-zA-Z0-9_]*$
      title: PostgreSQL Database Name
      type: string
    postgresql_password:
      pattern: ^[a-zA-Z0-9_~!@#$%^&*()-=<>,.?;:|]+$
      title: PostgreSQL Password
      type: string
    postgresql_user:
      default: admin
      maxLength: 63
      pattern: ^[a-zA-Z_][a-zA-Z0-9_]*$
      title: PostgreSQL User
      type: string
    postgresql_version:
      default: "9.6"
      enum:
      - "9.6"
      - "9.5"
      - "9.4"
      title: PostgreSQL Version
```

```
      type: string
  required:
  - postgresql_database
  - postgresql_user
  - postgresql_password
  - postgresql_version
  type: object

Instance Update Parameter Schema:
  $schema: http://json-schema.org/draft-04/schema
  additionalProperties: false
  properties:
    postgresql_version:
      default: "9.6"
      enum:
      - "9.6"
      - "9.5"
      - "9.4"
      title: PostgreSQL Version
      type: string
  required:
  - postgresql_version
  type: object

Binding Create Parameter Schema:
  $schema: http://json-schema.org/draft-04/schema
  additionalProperties: false
  type: object
```

6.2.3.3. サービスのプロビジョニング

プロビジョニングとは、サービスが利用できるように提供することです。サービスをプロビジョニングするには、サービスインスタンスを作成して、そのインスタンスにバインドする必要があります

6.2.3.3.1. ServiceInstance の

注記

サービスインスタンスは、OpenShift namespace 内に作成する必要があります。

1. 新規プロジェクトを作成します。

```
$ oc new-project <project-name> 1
```

1 **<project-name>** は、プロジェクト名に置き換えます。

2. 以下のコマンドを使用してサービスインスタンスを作成します。

```
$ svcat provision postgresql-instance --class rh-postgresql-apb --plan dev --params-json
'{"postgresql_database":"admin","postgresql_password":"admin","postgresql_user":"admin","po
stgresql_version":"9.6"}' -n szh-project
  Name:        postgresql-instance
  Namespace:   szh-project
```

```
  Status:
  Class:      rh-postgresql-apb
  Plan:       dev

Parameters:
  postgresql_database: admin
  postgresql_password: admin
  postgresql_user: admin
  postgresql_version: "9.6"
```

6.2.3.3.1.1. サービスインスタンスの詳細表示

サービスインスタンスの詳細を表示します。

```
$ svcat get instance
        NAME           NAMESPACE         CLASS         PLAN   STATUS
+---------------------+-------------+-------------------+------+--------+
  postgresql-instance   szh-project   rh-postgresql-apb   dev    Ready
```

6.2.3.3.2. ServiceBinding の作成

ServiceBinding リソースを作成する場合:

1. サービスカタログコントローラーは、ブローカーサーバーと通信して、バインディングを開始します。
2. ブローカーサーバーは、認証情報を作成し、サービスカタログコントローラーに対してその認証情報を発行します。
3. サービスカタログコントローラーは、これらの認証情報をシークレットとしてプロジェクトに追加します。

以下のコマンドを使用してサービスバインディングを作成します

```
$ svcat bind postgresql-instance --name mediawiki-postgresql-binding
  Name:        mediawiki-postgresql-binding
  Namespace:   szh-project
  Status:
  Instance:    postgresql-instance

Parameters:
  {}
```

6.2.3.3.2.1. サービスバインディングの詳細表示

1. サービスバインディングの詳細を表示します。

```
$ svcat get bindings
              NAME               NAMESPACE        INSTANCE         STATUS
+-----------------------------+-------------+---------------------+--------+
  mediawiki-postgresql-binding   szh-project   postgresql-instance   Ready
```

2. サービスのバインディング後にインスタンスの詳細を確認します。

```
$ svcat describe instance postgresql-instance
  Name:        postgresql-instance
  Namespace:   szh-project
  Status:      Ready - The instance was provisioned successfully @ 2018-06-05 08:42:55
+0000 UTC
  Class:       rh-postgresql-apb
  Plan:        dev

Parameters:
  postgresql_database: admin
  postgresql_password: admin
  postgresql_user: admin
  postgresql_version: "9.6"

Bindings:
              NAME              STATUS
+------------------------------+--------+
  mediawiki-postgresql-binding   Ready
```

6.2.4. リソースの定義

サービスカタログに関連するリソースを削除するには、サービスバインディングのバインドと、サービスインスタンスのプロビジョニングを解除する必要があります

6.2.4.1. サービスバインディングの削除

1. サービスインスタンスに関連付けられた全サービスバインディングを削除します。

```
$ svcat unbind -n <project-name> (1)
  \ <instance-name> (2)
```

(1) サービスインスタンスを含むプロジェクト名

(2) バインディングに関連するサービスインスタンス名

以下に例を示します。

```
$ svcat unbind -n szh-project postgresql-instance
deleted mediawiki-postgresql-binding

$ svcat get bindings
  NAME   NAMESPACE   INSTANCE   STATUS
+------+-----------+----------+--------+
```

注記

このコマンドを実行すると、インスタンスに対するサービスバインディングすべてを削除します。インスタンス内から個別のバインディングを削除するには、**svcat unbind -n <project-name> --name <binding-name>** のコマンドを実行します。たとえば、**svcat unbind -n szh-project --name mediawiki-postgresql-binding** などです。

2. 関連のシークレットが削除されたことを確認します。

```
$ oc get secret -n szh-project
NAME                       TYPE                                  DATA      AGE
builder-dockercfg-jxk48    kubernetes.io/dockercfg               1         9m
builder-token-92jrf        kubernetes.io/service-account-token   4         9m
builder-token-b4sm6        kubernetes.io/service-account-token   4         9m
default-dockercfg-cggcr    kubernetes.io/dockercfg               1         9m
default-token-g4sg7        kubernetes.io/service-account-token   4         9m
default-token-hvdpq        kubernetes.io/service-account-token   4         9m
deployer-dockercfg-wm8th   kubernetes.io/dockercfg               1         9m
deployer-token-hnk5w       kubernetes.io/service-account-token   4         9m
deployer-token-xfr7c       kubernetes.io/service-account-token   4         9m
```

6.2.4.2. サービスインスタンスの削除

サービスインスタンスのプロビジョニングを解除します。

```
$ svcat deprovision postgresql-instance
deleted postgresql-instance

$ svcat get instance
  NAME   NAMESPACE   CLASS   PLAN   STATUS
+------+-----------+-------+------+--------+
```

6.2.4.3. サービスブローカーの削除

1. サービスカタログのブローカーサービスを削除するには、**ClusterServiceBroker** リソースを削除します。

```
$ oc delete clusterservicebrokers template-service-broker
clusterservicebroker "template-service-broker" deleted

$ svcat get brokers
          NAME                                        URL                                    STATUS
+-------------------------+-------------------------------------------------------------------------------------
------+--------+
  ansible-service-broker    https://asb.openshift-ansible-service-broker.svc:1338/ansible-
service-broker               Ready
```

2. ブローカーの **ClusterServiceClass** リソースを表示して、ブローカーが削除されたことを確認します。

```
$ svcat get classes
  NAME   DESCRIPTION
+------+-------------+
```

6.3. テンプレートサービスブローカー

テンプレートサービスブローカー (TSB) では、OpenShift Container Platform の最初のリリースより同梱されている「デフォルトのインスタントアプリおよびクイックスタートテンプレート」でサービスカ

タログが確認できるようになります。TSB は、Red Hat、クラスター管理者、ユーザー、サードパーティベンダーの誰が提供したものでも、OpenShift Container Platform「テンプレート」に書き込まれたものはサービスとして提供できます。

デフォルトでは、TSB は **openshift** プロジェクトからグローバルで入手可能なオブジェクトを表示します。また、クラスター管理者が選択する他のプロジェクトを監視するように設定することも可能です。

6.4. OPENSHIFT ANSIBLE BROKER

6.4.1. 概要

OpenShift Ansible Broker (OAB) は、**Ansible playbook bundle (APB)** で定義されるアプリケーションを管理する Open Service Broker (OSB) API の実装です。 APB は、Ansible ランタイムと、コンテナイメージに同梱されている Ansible playbook のバンドルで構成されており、OpenShift Container Platform のコンテナーアプリケーションを定義、配信する新しい方法を提供します。APB は Ansible を活用して、複雑なデプロイするを自動化する標準メカニズムを構築します。

OAB の設計はこの基本的なワークフローに従います。

1. ユーザーは、OpenShift Container Platform Web コンソールを使用してサービスカタログから利用可能なアプリケーションの一覧を要求します。
2. サービスカタログは利用可能なアプリケーションについて OAB に要求します。
3. OAB は定義されたコンテナーレジストリーと通信し、利用可能な APB を把握します。
4. ユーザーは特定の APB をプロビジョニングする要求を実行します。
5. プロビジョニング要求は OAB に移動し、APB でプロビジョニングメソッドを呼び出して、ユーザー要求に対応します。

6.4.2. Ansible Playbook Bundle

Ansible Playbook Bundle (APB) は、Ansible ロールおよび Playbook の既存の投資を利用できる軽量のアプリケーション定義です。

APB は、名前の付いた playbook が含まれる単純なディレクトリーを使用し、プロビジョニングやバインドなどの OSB API アクションを実行します。**apb.yml** の仕様ファイルで定義するメタデータには、デプロイメント中に使用する必須/任意のパラメーターの一覧が含まれています。

全体の設計および APB の作成方法についての詳細は、『APB Development Guide』を参照してください。