OpenShift Online 3

アーキテクチャー

OpenShift Online アーキテクチャーガイド

Last Updated: 2020-09-11

OpenShift Online 3 アーキテクチャー

OpenShift Online アーキテクチャーガイド

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 Online 3 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の管理についても取り上げます。

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、モデルのすべての側面で作成後により柔軟な設定が可能になります。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。

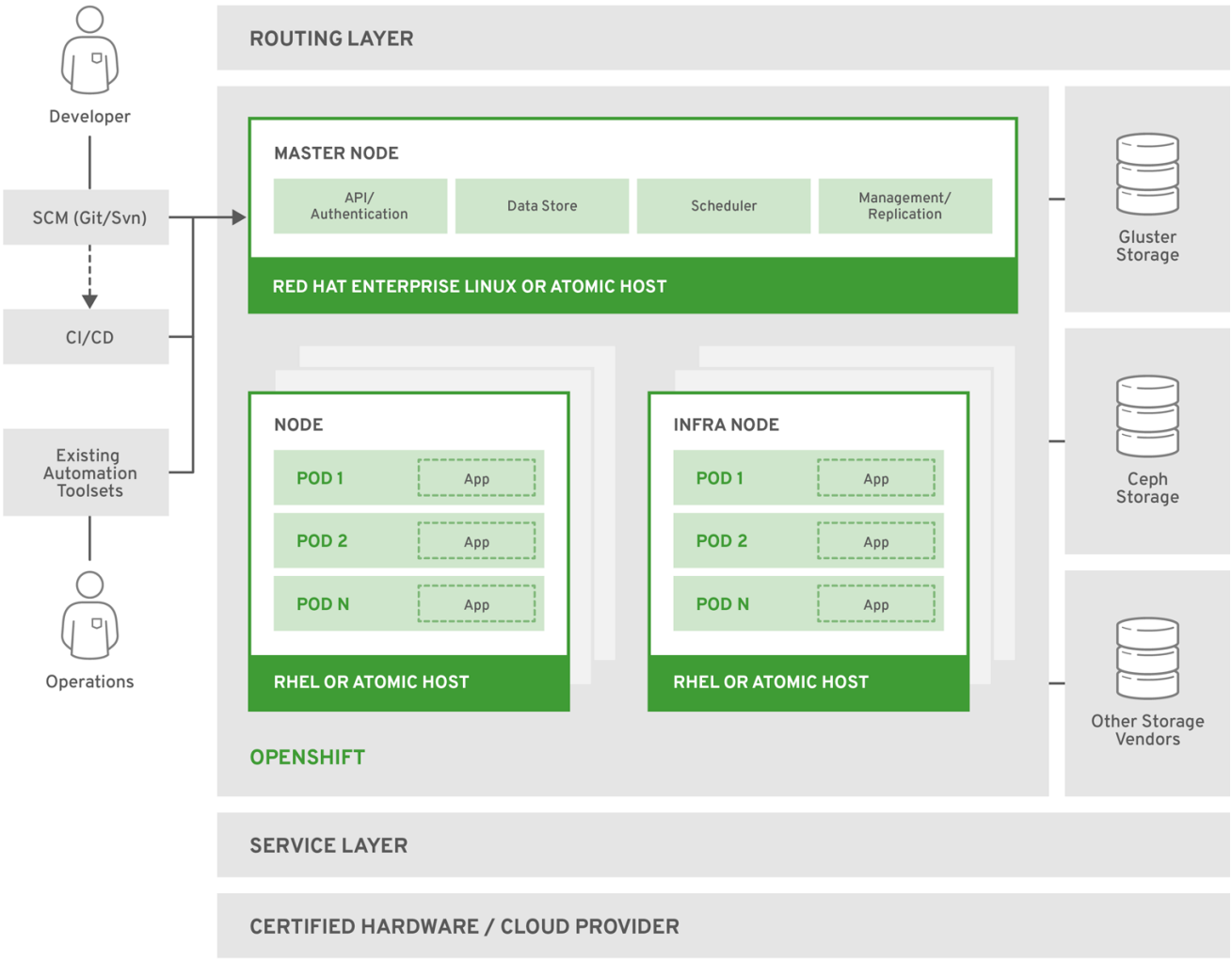
1.1. レイヤーとは

Docker サービスは、Linux ベースの軽量 コンテナーイメージをパッケージ化し、作成するための抽象化を提供します。Kubernetes は、クラスター管理を提供し、複数のホストでコンテナーをオーケストレーションします。

OpenShift Online は以下を追加します。

- 開発者向けのソースコード管理、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 Online アーキテクチャーの概要



1.2. OPENSHIFT ONLINE アーキテクチャーとは

OpenShift Online には、それぞれ連携する小規模な切り離されたユニットから構成されるマイクロサービスベースのアーキテクチャーがあります。これは、Kubernetes クラスターの上部で、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 に保存されるオブジェクトについてのデータを使用して実行されます。これらのサービスは、機能別に分類されます。

- REST API: 各コアオブジェクトを公開します。
- コントローラー: これらの API を読み取り、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の必要な状態を読み取り、システムの他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と、「ビルド」オブジェクトが作成されます。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、クラスターでプロセスを実行してそのビルドを実行します。ビルドが完了すると、コントローラーは REST API 経由でビルドオブジェクトを更新し、ユーザーはビルドが完了していることを確認できます。

コントローラーパターンは、OpenShift Online の機能の大半が拡張可能であることを意味します。ビルドの実行および起動方法は、イメージの管理方法や デプロイメントがどのように行われるかに関係なく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」を実行して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したり、独自のロジックに置き換えたりすることで、各種の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 ONLINE のセキュリティーを保護する化方法

OpenShift Online および Kubernetes API は認証情報を提供するユーザーを認証し、それらのロールに基づいてユーザーを認可します。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 PKCS#1 v1.5 です。

開発者 (システムのクライアント) は通常、(**oc** などの)クライアントプログラムを使用するか、またはブラウザーを使用して Web コンソール に対して REST API 呼び出しを行い、ほとんどの通信に 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、システムで生成されるアイデンティティーが含まれる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それらの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。

認可は、「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、シークレットをそれらのサービスアカウントに関連付け、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でイメージ、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のシークレットを管理でき、アプリケーションコードでそれらのシークレットを簡単に利用することも可能になります。

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チャネルは TLS で保護されます。TLS は、X.509 サーバー証明書およびパブリックキーインフラストラクチャーを使用して強力な暗号化、データの整合性、およびサーバーの認証を提供します。

OpenShift Online は Golang の標準ライブラリーの実装である **crypto/tls** を使用し、外部の暗号ライブラリーおよび 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 Online 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 Online 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]	可能性あり [b]
TLS 1.1	No [a]	No [a]	可能性あり [b]
TLS 1.2	**Yes**	**Yes**	**Yes**
TLS 1.3	該当なし [c]	該当なし [c]	該当なし [c]

[a] デフォルトで無効になっていますが、サーバー設定で有効にできます。

[b] LDAP クライアントなどの一部の内部クライアント。

[c] TLS 1.3 は開発中です。

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 Online 内で、Kubernetes はコンテナーまたはホスト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スケーリングのメカニズムを提供します。Docker サービスはコンテナー化されたアプリケーションをパッケージ化し、インスタンス化し、これを実行します。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。

注記

OpenShift Online は Kubernetes 1.9 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 などのマスターコンポーネントが含まれるホストです。マスターは Kubernetes クラスターのノードを管理し、Pod をノードで実行されるように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 の API サーバーは、Pod、サービス、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。これはスタンドアロンプロセスとして実行できます。
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。これはスタンドアロンプロセスとして実行できます。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。

2.1.3. ノード

ノードはコンテナーのランタイム環境を提供します。Kubernetes クラスターの各ノードには、マスター で管理される必要のあるサービスがあります。また、ノードには、Docker サービス、kubelet、およびサービスプロキシーを含む Pod を実行するための必要なサービスも含まれます。

OpenShift Online 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の情報を使っ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。ノードの管理についての詳細は、Kubernetes ドキュメント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すべての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。

2.2. CONTAINER レジストリー

2.2.1. 概要

OpenShift Online 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 Online 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 Online には **OpenShift Container レジストリー** (OCR) という統合コンテナーイメージ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、アプリケーションビルドの組み込まれた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 Online 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 Online 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。

2.2.3. サードパーティーレジストリー

OpenShift Online はサードパーティーレジストリーから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 Online レジストリーと同じイメージ通知のサポートを提供する訳ではありません。このため、OpenShift Online はイメージストリームの作成時にリモートレジストリーからタグをフェッチします。フェッチされたタグの更新は、**oc import-image <stream>** を実行するだけで簡単に実行できます。新規イメージが検出されると、以前に記述されたビルドとデプロイメントの応答が生じます。

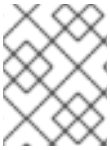
2.2.3.1. 認証

OpenShift Online はユーザーが指定する認証情報を使用してプライベートイメージリポジトリーにアクセスするためにレジストリーと通信できます。これにより、OpenShift Online はイメージのプッシュ/プルをプライベートリポジトリーへ/から実行できます。「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

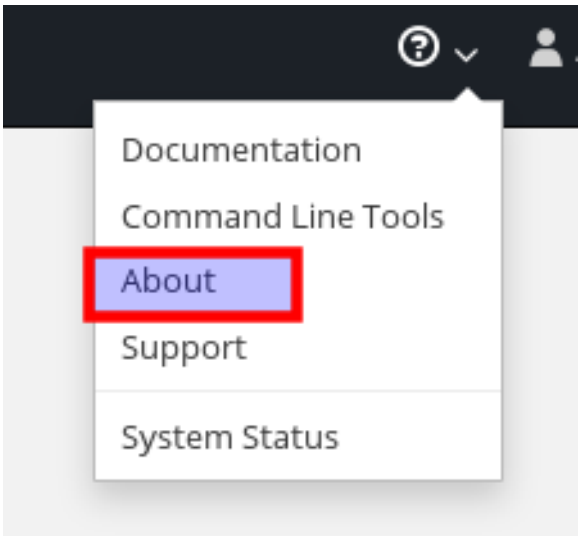
OpenShift Online Web コンソールは、Web ブラウザーからアクセスできるユーザーインターフェースです。開発者は Web コンソールを使用してプロジェクトのコンテンツを視覚的に把握し、参照し、管理することができます。

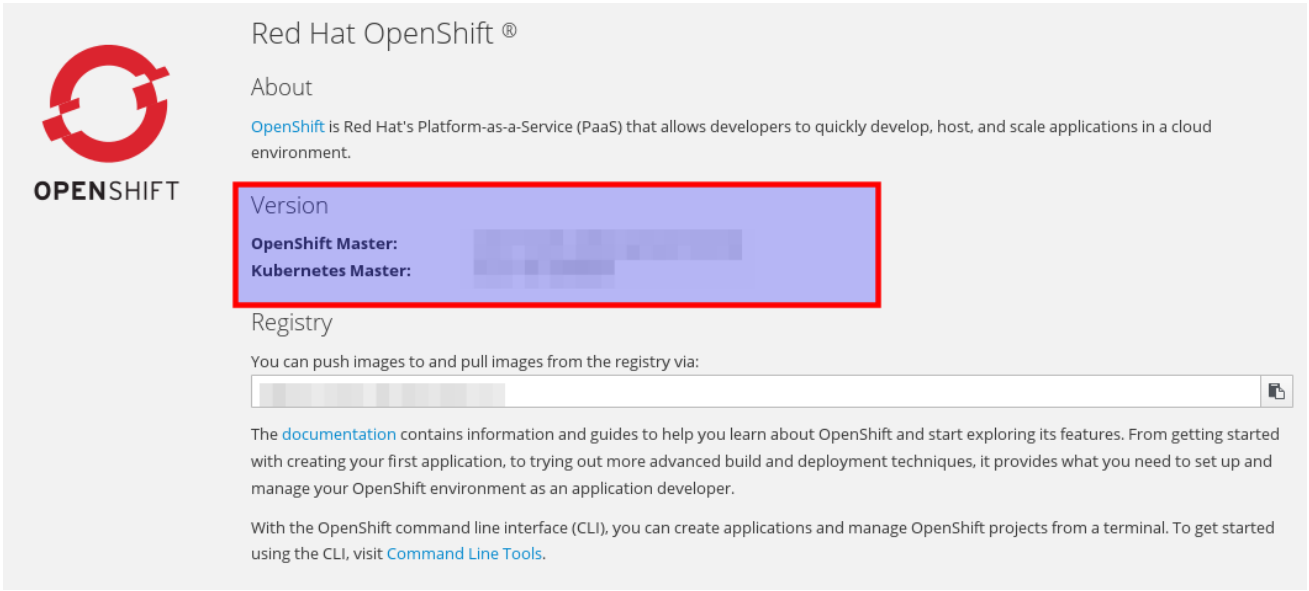


注記

Web コンソールを使用するために 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 コンソールの **About** ページから、クラスターのバージョン番号を確認できます。

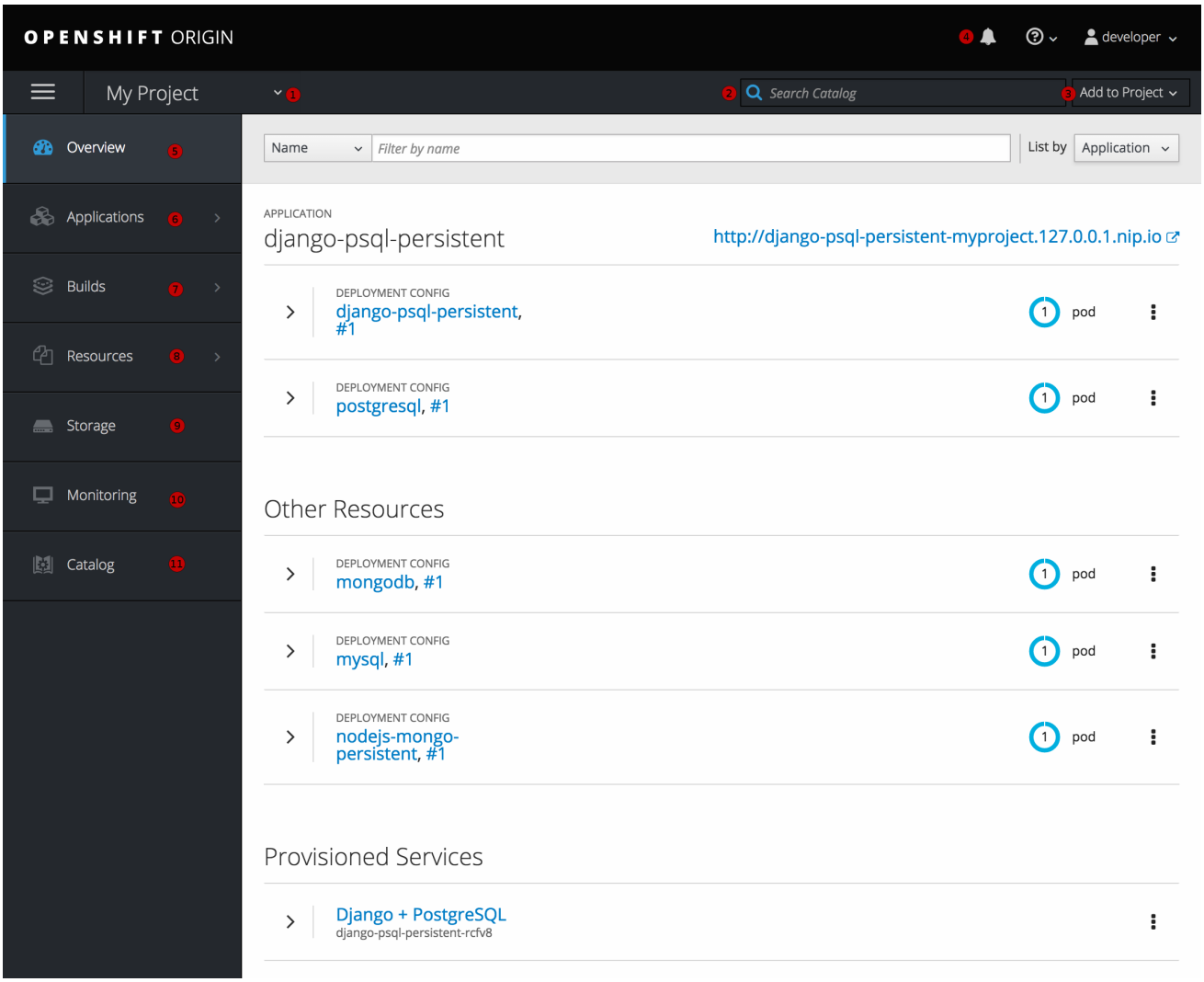




2.3.2. プロジェクトの概要

ログイン後、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2.1 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を使用するか、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にアクセスし、アクションを実行します。

Resources タブ： 現在のクォータ消費やその他のリソースを表示します。

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

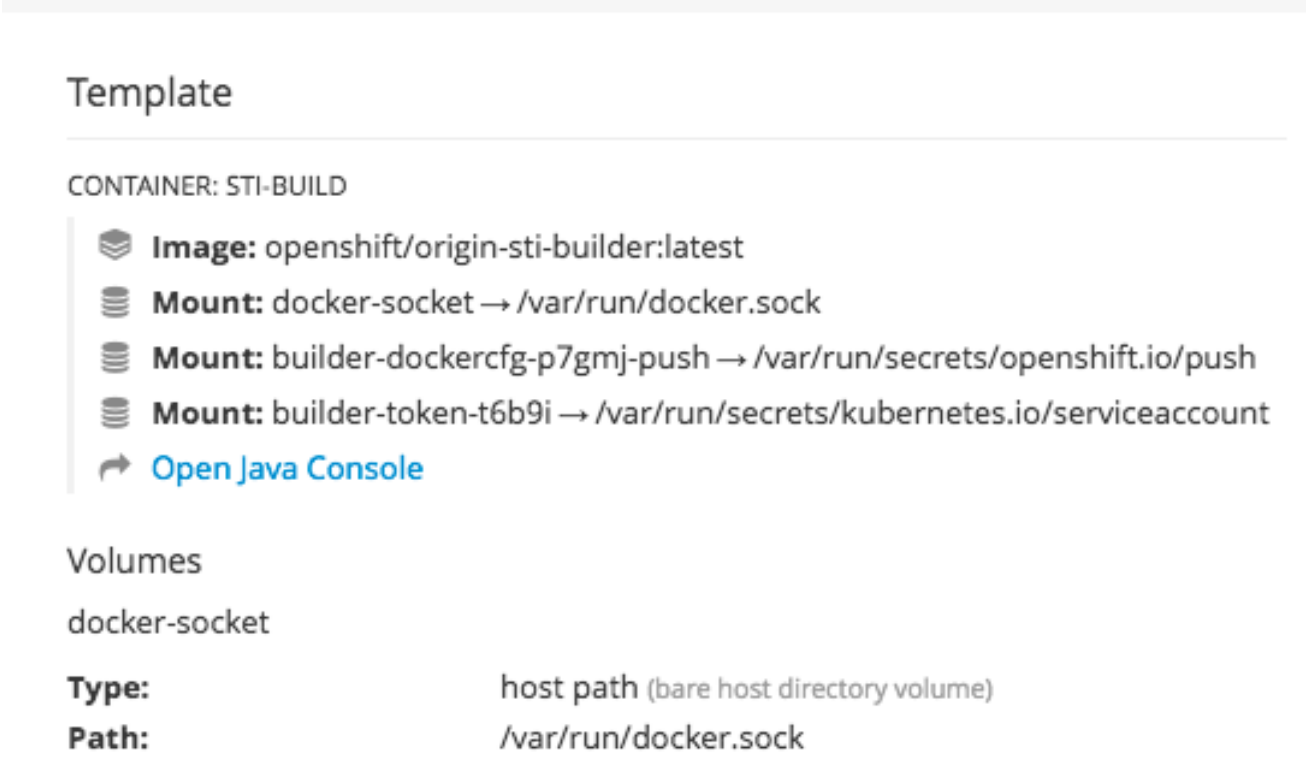
Monitoring タブ: ビルド、Pod、デプロイメントのログ、およびプロジェクトのすべてのオブジェクト通知を表示します。

Catalog タブ： プロジェクト内からすばやくカタログに移動します。

2.3.3. JVM コンソール

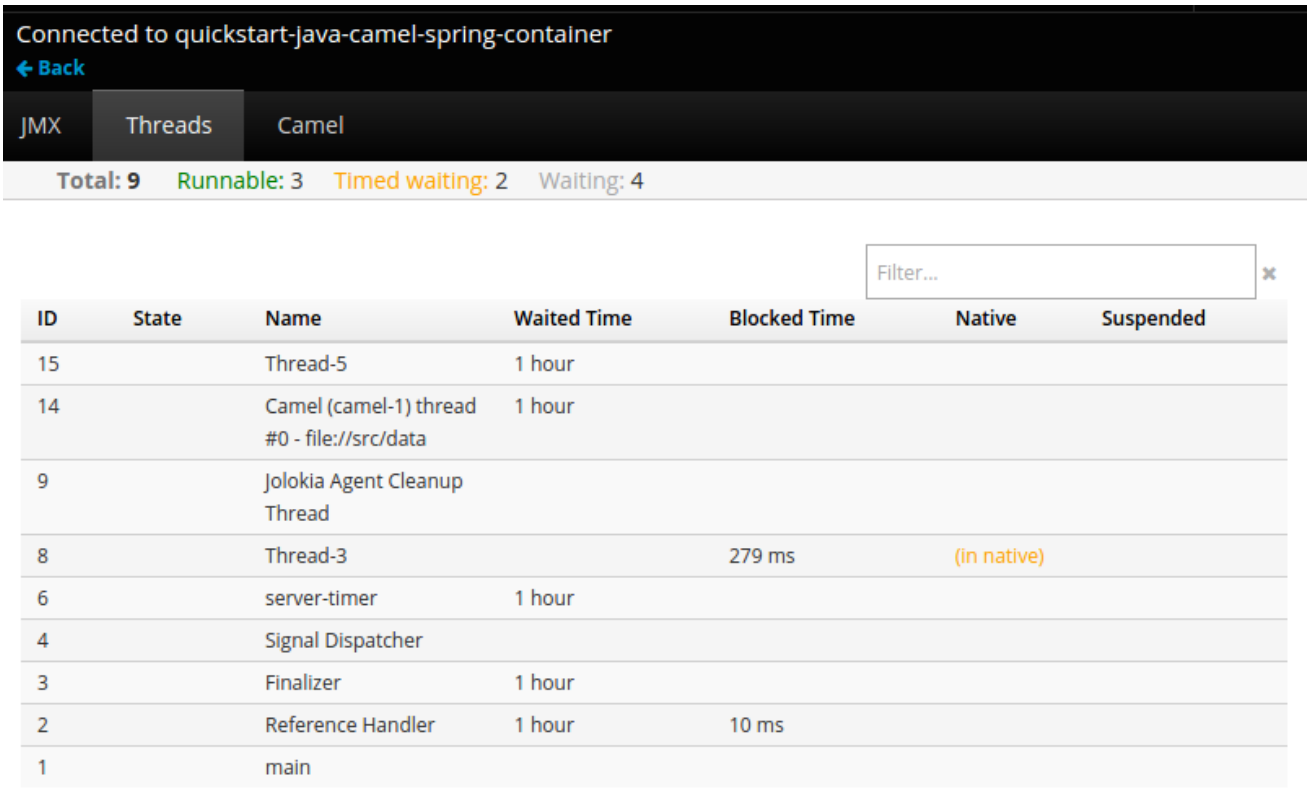
Java イメージをベースとする Pod の場合、Web コンソールは関連する統合コンポーネントを表示し、管理するための hawt.io ベースの JVM コンソールへのアクセスも公開します。**Connect** リンクは、コンテナーに **jolokia** という名前のポートがある場合は、**Browse → Pods** ページの Pod の詳細に表示されます。

図2.2 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2.3 JVM コンソール



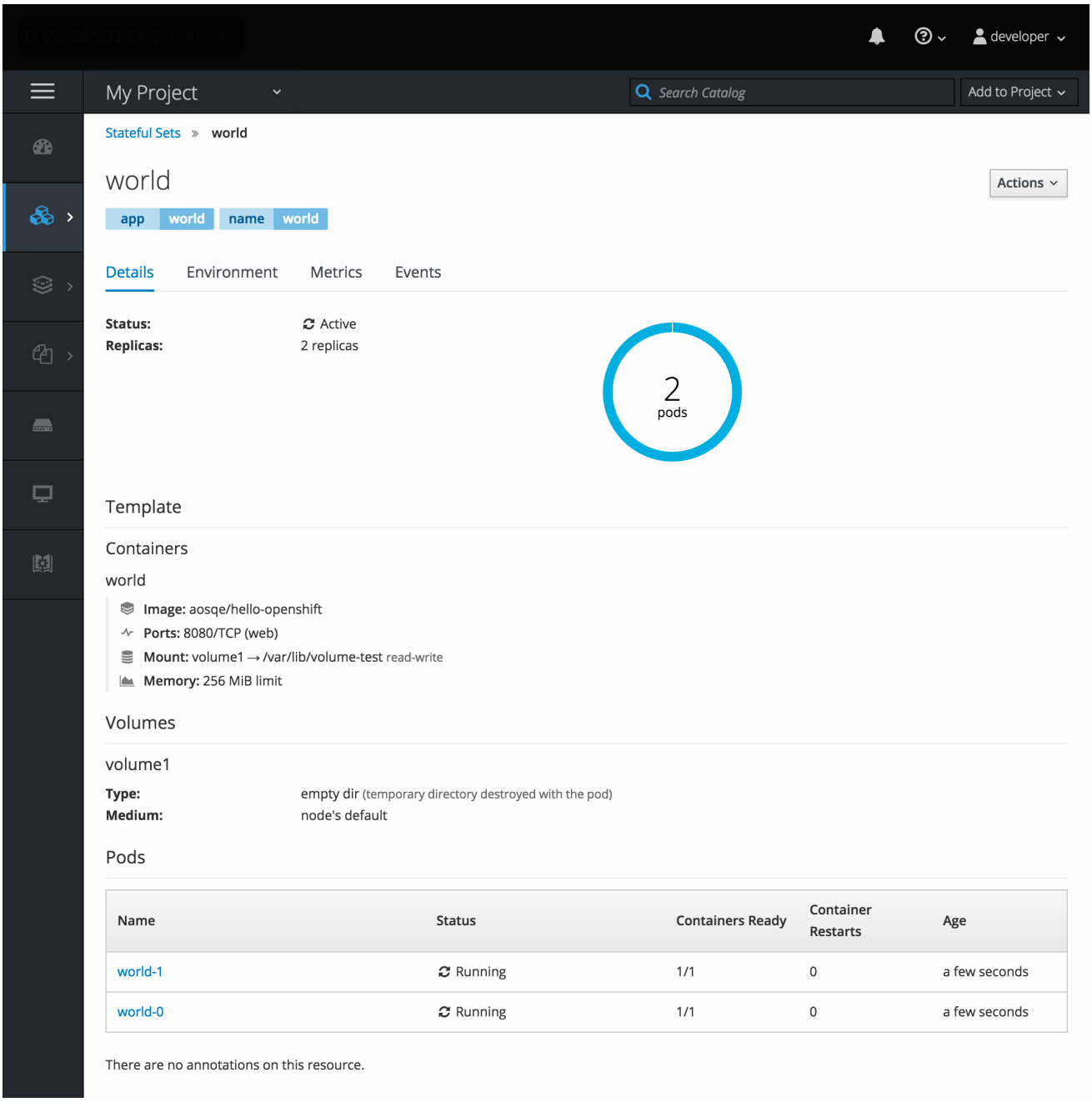
以下のページが利用可能になります。

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。

2.3.4. 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2.4 OpenShift Online 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 Online 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の対話に制限されるように、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 Online 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ています。

OpenShift Online 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 Online 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 Online 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。

3.2.2. イメージ

OpenShift Online 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のホストにまたがって複数のコンテナーにデプロイし、それらの間で負荷を分散することにより、OpenShift Online はイメージにパッケージ化されたサービスの冗長性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 Online 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一定期間をかけて開発されるため、単一のイメージ名が同じイメージの数多くの異なるバージョンを参照する場合があります。異なるイメージはそれぞれ、そのハッシュ (長い 16 進数、例: **fd44297e2ddb050ec4f…**) で一意に参照され、通常は 12 文字 (例: **fd44297e2ddb**) に短縮されます。

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 Online 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/** 以降のイメージであり、**latest** は **3.4**、または **3.5** などの数多くのバージョンを参照する可能性があります。

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 Online 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> を参照します。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> タグを参照します (例: 3.4.1-8)。

X.Y.Z

上記の vX.Y サンプルと同様に、X.Y.Z タグは最新の X.Y.Z-<number> を参照します。たとえば、3.4.1 は 3.4.1-8 を参照します。

X.Y.Z-<number>

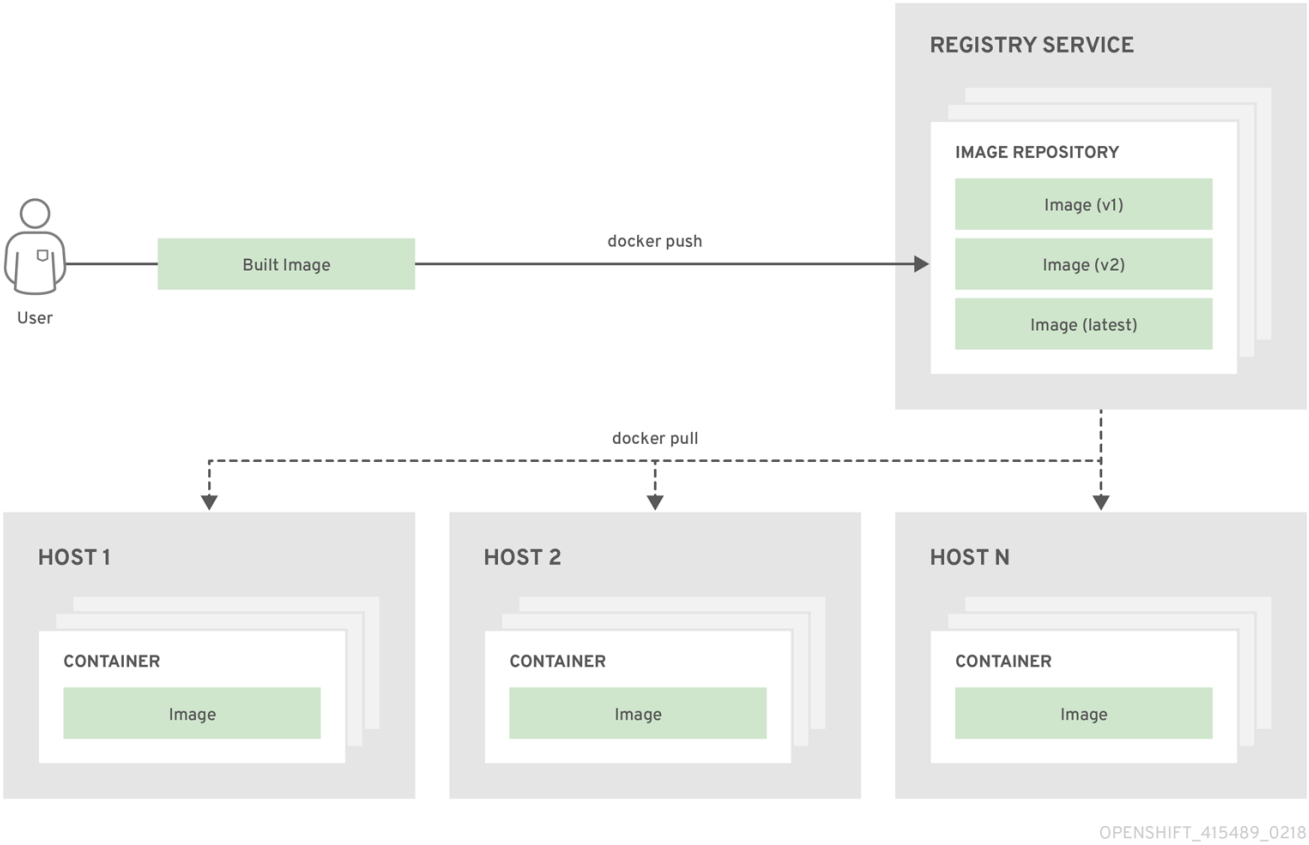
タグは一意であり、変更されません。このタグを使用する際、イメージが更新される際にイメージはタグを更新しません。たとえば、イメージが更新される場合でも、3.4.1-8 は 3.4.1-8 を常に参照します。

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。それぞれのイメージレジストリーには、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、サブスクリプションをお持ちのお客様に対して **registry.access.redhat.com** でレジストリーを提供しています。また、OpenShift Online 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ています。

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 Online は、**Pod** の Kubernetes の概念を活用しています。これはホスト上に共にデプロイされる 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) とほぼ同等のものです。各 Pod には独自の内部 IP アドレスが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。それらは定義された後にノードで実行され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 Online 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 Online 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も処理されますが、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。

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 Pod のサンプル定義です。これは、統合コンテナーレジストリーという OpenShift Online インフラストラクチャーの一部で、長期間実行されるサービスを提供します。これは数多くの Pod の機能を示していますが、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 **metadata** 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 Online 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 Online 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 Online 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 **secret** 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 Online 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 Online 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。

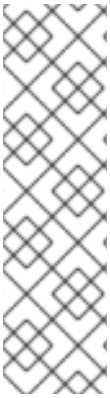
- **Always**: Pod が再起動するまで、Pod で正常に終了したコンテナーの継続的な再起動を、指数関数のバックオフ遅延 (10 秒、20 秒、40 秒) で試行します。デフォルトは **Always** です。
- **OnFailure**: Pod で失敗したコンテナーの継続的な再起動を、5 分を上限として指数関数のバックオフ遅延 (10 秒、20 秒、40 秒) で試行します。
- **Never**: Pod で終了したコンテナーまたは失敗したコンテナーの再起動を試行しません。Pod はただちに失敗し、終了します。

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	任意

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 Online 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しなくてはなりません。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。



注記

Kubernetes アーキテクチャーでは、クラウドプロバイダーからの信頼性のあるエンドポイントが必要です。クラウドプロバイダーが停止している場合、kubelet は OpenShift Online が再起動されないようにします。

基礎となるクラウドプロバイダーのエンドポイントに信頼性がない場合は、クラウドプロバイダー統合を使用してクラスターをインストールしないでください。クラスターを、非クラウド環境で実行する場合のようにインストールします。インストール済みのクラスターで、クラウドプロバイダー統合をオンまたはオフに切り替えることは推奨されていません。

OpenShift Online 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。

3.3.2. サービス

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 Online 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3.2 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。

3.3.2.1. サービスプロキシー

OpenShift Online には、サービスルートインフラストラクチャーの **iptables** ベースの実装があります。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。また、すべてのエンドポイントが常に接続を受け入れることができる必要があります。

3.3.2.2. 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。

3.3.2.2.1. 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
```

3.3.2.2.2. ヘッドレスサービスを使用したエンドポイントの検出

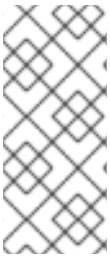
ヘッドレスサービスを使用する利点として、Pod の IP アドレスを直接検出できることが挙げられます。標準サービスはロードバランサーまたはプロキシーとして機能するか、またはサービス名を使用してワークロードオブジェクトへのアクセスを付与します。ヘッドレスサービスの場合には、サービスごとに分類された Pod の IP アドレスセットに、サービス名を解決します。

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。

3.3.3. 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。

3.3.4. 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 Online 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 Online 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 Online 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 Online との対話はユーザーに関連付けられます。OpenShift Online ユーザーオブジェクトは、システムでパーミッションを付与できるアクターを表します。

ユーザーにはいくつかのタイプが存在します。

regular user (通常ユーザー)	これは、大半の対話型の OpenShift Online 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**

system user (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**
service account (サービスアカウント)	プロジェクトに関連付けられる特殊なシステムユーザーがあります。それらの中には、プロジェクトの初回作成時に自動作成されるものもあれば、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ために追加で作成するものもあります。サービスアカウントは **ServiceAccount** オブジェクトで表されます。例: **system:serviceaccount:default:deployer system:serviceaccount:foo:builder**

それぞれのユーザーには、OpenShift Online 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、認可されているユーザーの実行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 Online 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。

namespace の詳細は、Kubernetes ドキュメントを参照してください。

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ス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。

Objects	Pod、サービス、レプリケーションコントローラーなど。
Policies	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。
Constraints	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。
service account (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を作成でき、プロジェクトの管理者権限をユーザーコミュニティーの任意のメンバーに委任できます。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できます。

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を実行できます。

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 Online には追加設定なしで使用できる数多くのプロジェクトが含まれますが、 **openshift** はユーザーにとって最も重要なプロジェクトになります。

openshift ユーザーに表示されるプロジェクトで、主に日常的なタスクのオブジェクトを格納するために使用されます。これらには、テンプレートやイメージなどの複数プロジェクトでアクセスされるアプリケーションオブジェクトが含まれます。これらのオブジェクトは、Pod 間の通信を必要としないものである必要があります。

3.4.4. プロジェクトのアイドリング

OpenShift Online Starter では、24 時間を超えて非アクティブな状態が続くプロジェクトがアイドル状態になります。プロジェクトのネットワークアクティビティーが設定されたしきい値を下回ると、プロジェクトは非アクティブとみなされます。プロジェクトがアイドル状態になる場合、レプリカ数は **0** に設定され、すべての Pod が削除されます。プロジェクトの永続ボリューム (PV) および Persistent Volume Claim（永続ボリューム要求、PVC）はすべて変更されないままになります。ネットワークトラフィックを受信すると、レプリカ数はアイドル状態になる前の値にスケーリングされます。

Web コンソールでは、デプロイメントが **非アクティブのためのアイドル状態** として表示され、デプロイメントのスケールアップは手動で実行できます。

ネットワークトラフィックがプロジェクトのレプリカ数を復元しない場合、デプロイメントを手動でスケールアップする必要がある場合があります。

3.4.5. アカウントのプルーニング

OpenShift Online Starter アカウントが非アクティブである場合 (3 日間プロジェクトに実行中の Pod がない場合)、アカウントのプロビジョニング解除について警告メールが送信されます。修正アクションを実行せず、Pod を 5 日以内に作成しないと、アカウントは自動的にプロビジョニング解除されます。アカウントのプロビジョニングが解除されると、再度登録できます。

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 とは、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。ほとんどの場合、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 Online 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、ビルドプロセスのロギング、正常なビルドからのリリースのパブリッシュ、およびビルドの最終ステータスのパブリッシュが含まれます。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指定します。

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するものになります。

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 Online 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. Source-to-Image (S2I) ビルド

Source-to-Image (S2I) は再現可能な 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** コマンドで使用することができます。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。

3.5.1.2. Pipeline ビルド

開発者は、Pipeline ビルド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 Pipeline** を定義することができます。このビルドは他のビルドタイプの場合と同様に OpenShift Online 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 Online 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。

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ません。これは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 Online 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、ビルドおよびデプロイメントを、新規イメージが追加される際やこれに対応する際の通知をイメージストリームで確認できるように設定できます。

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 Online の統合レジストリー
- **registry.access.redhat.com** または **hub.docker.com** などの外部レジストリー
- OpenShift Online 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 Onliine 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、ビルドおよびデプロイメントをトリガーできます。また、OpenShift Online 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を実行するためにタグにマークを付けることができます。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、ビルドま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 イメージタグを参照する場合、Docker イメージタグの更新方法を理解しておく必要があります。たとえば、Docker イメージタグ **docker.io/ruby:2.4** が v2.4 ruby イメージを常に参照するとします。ただし、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れます。そのため、イメージストリームタグが参照する Docker イメージタグは、これを参照することを選択する場合にイメージストリームタグの安定度を示すものとなります。
- 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参照しない場合)、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。この場合も、互換性のないバージョンの変更が選択されてしまう可能性があります。

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

特定のイメージを区別する、リポジトリー内の Docker イメージに適用されるラベルです。たとえば、ここでは 3.6.0 がタグとして使用されています。

```
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 Online 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

イメージが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、これによりデプロイメント、ビルド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 の概要**」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 Online 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、タグが参照したすべてのイメージのスタックとして表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特定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参照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参照するバージョン固有のタグです。
- **トラッキングタグ** 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 Online に同梱される **latest** イメージストリームタグはトラッキングタグです。これは、**latest** イメージストリームタグのコンシューマーが、新規レベルが利用可能になるとイメージで提供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 **v3.7** に変更される可能性が常にあります。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参照しません。これは別のイメージストリームタグを参照し、これはイメージの最新バージョンではない可能性があります。たとえば、**latest** イメージストリームタグがイメージの **v3.2** を参照する場合、**3.11** バージョンがリリースされても **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参照するように手動で更新されるまで **v3.2** を参照したままになります。

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。

各自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 の内容が変更されるたびに、(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で) 新規レプリケーションコントローラーが作成されます。

例3.3 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。

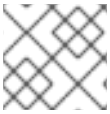
上記の例では、**origin-ruby-sample** イメージストリームの **latest** タグの値が変更され、新しいイメージの値がデプロイメント設定の **helloworld** コンテナーに指定されている現在のイメージと異なる場合に、**helloworld** コンテナーの新規イメージを使用して、新しいレプリケーションコントローラーが作成されます。

注記

ImageChange トリ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場合、これは OpenShift Online 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 Online は、コマンド、エントリーポイント、および環境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 Online のイメージはイミュータブル (変更不可能) 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、「イメージの管理」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>
```

以下は例になります。

```
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。

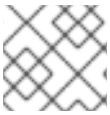
```
oc describe istag/<image-stream>:<tag-name>
```

以下は例になります。

```
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>
```

以下は例になります。

```
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 イメージを参照するタグ (**3.5**) と、この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参照する別のタグ (**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>
```

以下は例になります。

```
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、(最新のセキュリティー更新を取得する場合などに) イメージを定期的に再インポートするには、**--scheduled** フラグを使用します。

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

以下は例になります。

```
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 Online 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。この期間はクラスター全体のデフォルトで 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. 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。同様に、必要以上の数の Pod が実行されている場合には、定義された数に一致させるために必要な数の Pod を削除します。

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. キーをフィルターするセットベースのセレクター。これは、**tier** と同等のキー、**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
```

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 Online 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 Online 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 Online **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

テンプレートでは、パラメーター化や処理が可能な一連のオブジェクトを記述し、 OpenShift Online で作成するためのオブジェクトの一覧を生成します。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が含まれます。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が含まれます。また、テンプレートではラベルのセットを定義して、これをテンプレート内に定義されたすべてのオブジェクトに適用できます。

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 Online 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、認可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。

4.1.2. ユーザーとグループ

OpenShift Online の **ユーザー** は、OpenShift Online API への要求を実行できるエンティティーです。通常、これは OpenShift Online 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、アクセスをユーザーに個別に付与するのではなく、たとえばプロジェクト内の 複数のオブジェクトに対するアクセスを許可するなど、パーミッションを複数のユーザーに付与する際に役立ちます。

明示的に定義されるグループのほかにも、システムグループまたは **仮想グループ** が OpenShift で自動的にプロビジョニングされます。

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 Online 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 Online 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 Online サーバーバージョン 3.6 より前の 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 Online サーバーバージョン 3.6 以降の websocket 要求の **base64url.bearer.authorization.k8s.io.<base64url-encoded-token>** 形式の 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証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、認可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 Online API への要求は、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 によるユーザー B の権限の借用は、ユーザー A が認証された後に可能になります。次に、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、認可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 Online 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) エラーコードを出して失敗します。

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。

4.1.4. OAuth

OpenShift Online 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 に対して認証します。

新しいOAuthのトークンが要求されると、OAuth サーバーは設定済みのアイデンティティ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 Online 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。

OAuth クライアント	使用法
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ものの、ユーザーによってアクセスが付与されていない場合に実行するアクションを判別するために使用されます。「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。

以下は例になります。

```
"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://"
Hostname	"//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。

リダイレクト URI アノテーションが指定されていないか、無効な URI が指定されている

```
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>
```

無効なルートが指定されている

```
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>]
```

無効な参照タイプが指定されている

```
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>]
```

SA トークンがない

```
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
        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。

> The authorization server encountered an unexpected condition that prevented it from fulfilling the request.

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

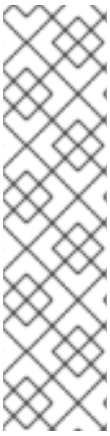
```
oc get events | grep ServiceAccount

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** への要求が必要になります。ほとんどの認証統合では、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 Online を、サポートするエンティティ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 Online 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 Online 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 Online 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 Online は IETF OAuth 2.0 Authorization Server Metadata 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認可サーバーの発行者 IDです。これは、認可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. 認可サーバーの認可エンドポートの URL です。RFC 6749 を参照してください。
3. 認可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. この認可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. この認可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. この認可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. この認可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。

以下のコマンドを実行し、Authorization Code Grant (認可コードによるグラント) 方法を使用して OAuth トークンを要求します。

```
$ curl -H "X-Remote-User: <username>" \
    --cacert /etc/origin/master/ca.crt \
    --cert /etc/origin/master/admin.crt \
    --key /etc/origin/master/admin.key \
    -I https://<master-address>/oauth/authorize?response_type=token\&client_id=openshift-
challenging-client | grep -oP "access_token=\K[^&]*"
```

OAuth トークンを、 (**openshift-challenging-client** などの) **WWW-Authenticate チャレンジ** 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	音声内容	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。

Implicit Grant (暗黙的グラント) フローを使用して OAuth トークンを要求するには、以下を実行します。

```
$ curl -u <username>:<password>
'https://<master-address>:8443/oauth/authorize?client_id=openshift-challenging-
client&response_type=token' -skv / 1
/ -H "X-CSRF-Token: xxx" 2
*   Trying 10.64.33.43...
* Connected to 10.64.33.43 (10.64.33.43) port 8443 (#0)
* found 148 certificates in /etc/ssl/certs/ca-certificates.crt
* found 592 certificates in /etc/ssl/certs
* ALPN, offering http/1.1
* SSL connection using TLS1.2 / ECDHE_RSA_AES_128_GCM_SHA256
```

```
*       server certificate verification SKIPPED
*       server certificate status verification SKIPPED
*       common name: 10.64.33.43 (matched)
*       server certificate expiration date OK
*       server certificate activation date OK
*       certificate public key: RSA
*       certificate version: #3
*       subject: CN=10.64.33.43
*       start date: Thu, 09 Aug 2018 04:00:39 GMT
*       expire date: Sat, 08 Aug 2020 04:00:40 GMT
*       issuer: CN=openshift-signer@1531109367
*       compression: NULL
* ALPN, server accepted to use http/1.1
* Server auth using Basic with user 'developer'
> GET /oauth/authorize?client_id=openshift-challenging-client&response_type=token HTTP/1.1
> Host: 10.64.33.43:8443
> Authorization: Basic ZGV2ZWxvcGVyOmRzc2Zkcw==
> User-Agent: curl/7.47.0
> Accept: */*
> X-CSRF-Token: xxx
>
< HTTP/1.1 302 Found
< Cache-Control: no-cache, no-store, max-age=0, must-revalidate
< Expires: Fri, 01 Jan 1990 00:00:00 GMT
< Location:
https://10.64.33.43:8443/oauth/token/implicit#access_token=gzTwOq_mVJ7ovHliHBTgRQEEXa1aCZ
D9lnj7lSw3ekQ&expires_in=86400&scope=user%3Afull&token_type=Bearer (3)
< Pragma: no-cache
< Set-Cookie:
ssn=MTUzNTk0OTc1MnxIckVfNW5vNFlLSlF5MF9GWEF6Zm55Vl95bi1ZNE41S1NCbFJMYnN1TWV
wR1hwZmlLMzFQRklzVXRkc0RnUGEzdnBEa0NZZndXV2ZUVzN1dmFPM2dHSUlzUmVXakQ3Q09rV
XpxNlRoVmVkQU5DYmdLTE9SUWlyNkJJTm1mSDQ0N2pCV09La3gzMkMzckwxc1V1QXpybFlXT2ZY
SmI2R2FTVEZsdDBzRjJ8vk6zrQPjQUmoJCqb8Dt5j5s0b4wZlITgKlho9wlKAZI=; Path=/; HttpOnly;
Secure
< Date: Mon, 03 Sep 2018 04:42:32 GMT
< Content-Length: 0
< Content-Type: text/plain; charset=utf-8
<
* Connection #0 to host 10.64.33.43 left intact
```

(1) **client-id** は **openshift-challenging-client** に設定され、**response-type** は **token** に設定されます。

(2) **X-CSRF-Token** ヘッダーを空でない値に設定します。

(3) トークンは、**access_token=gzTwOq_mVJ7ovHliHBTgRQEEXa1aCZD9lnj7lSw3ekQ** として **302** 応答の **Location** ヘッダーで返されます。

OAuth トークンの値のみを表示するには、以下のコマンドを実行します。

```
$ curl -u <username>:<password>
'https://<master-address>:8443/oauth/authorize?client_id=openshift-challenging-
client&response_type=token' (1)
```

```
-skv -H "X-CSRF-Token: xxx" --stderr - |  grep -oP "access_token=\K[^&]*" (2)

hvqxe5aMlAzvbqfM2WWw3D6tR0R2jCQGKx0viZBxwmc
```

(1) **client-id** は **openshift-challenging-client** に設定され、**response-type** は **token** に設定されます。

(2) **X-CSRF-Token** ヘッダーを空でない値に設定します。

トークンを要求するために **Code Grant** 方法を使用することもできます。

4.2. 認可

4.2.1. 概要

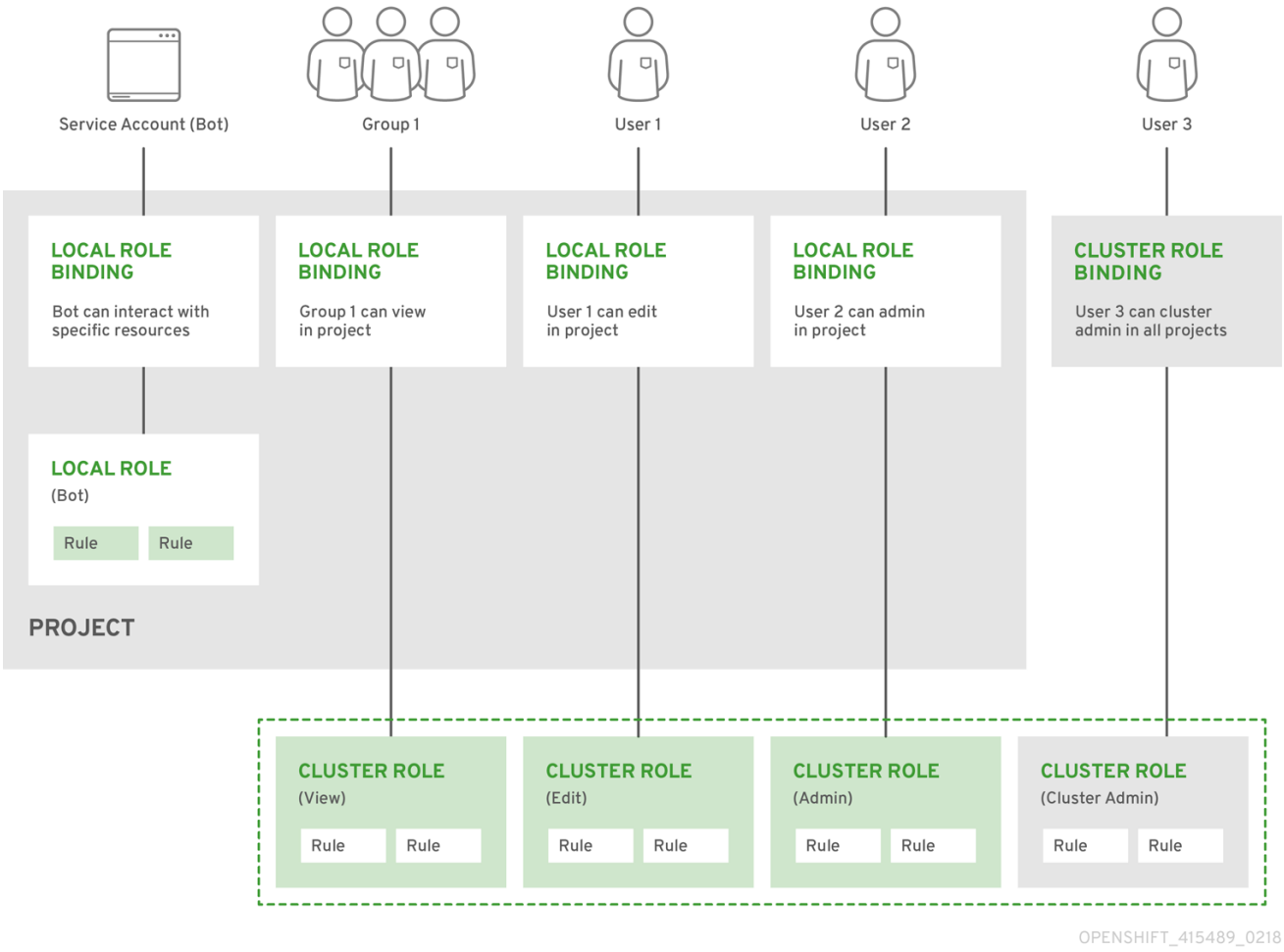
Role-based Access Control (RBAC: ロールベースアクセス制御) オブジェクトは、ユーザーがプロジェクト内で所定のアクションを実行することが許可されるかどうかを決定します。

これにより、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。認可は、認証とは異なる別の手順であることに注意してください。認可の手順は、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密接に関連しています。

認可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る動詞のセット。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。ユーザーおよびグループは、1 度に複数のロールに関連付けるか、または **バインド** することができます。
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. 認可の評価

OpenShift Online が認可を評価する際、いくつかの要因が組み合わさって決定が行われます。

Identity	認可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。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: アクセスする 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 Online は以下の手順を使って認可を評価します。

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. コラボレーション

OpenShift Online Pro では、ロール (**表示** または **編集** など) をプロジェクトの他のユーザーまたはグループに付与できます。

コラボレーターの追加および削除についての詳細は、OpenShift Online Pro でのプロジェクトコラボレーションについて参照してください。

OpenShift Online Starter では、コラボレーションは利用できません。

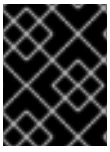
4.3. 永続ストレージ

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 Online 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。Persistent Volume Claim (永続ボリューム要求、PVC) を使用すると、開発者は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がなくても PV リソースを要求することができます。

PVC はプロジェクトに固有のもので、開発者が PV を使用する手段として作成し、使用します。PV リソース自体のスコープはいずれの単一プロジェクトにも設定されず、それらは OpenShift Online クラスター全体で共有でき、すべてのプロジェクトから要求できます。PV が PVC に **バインド** された後は、その PV を追加の PVC にバインドすることはできません。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** などの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. 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. バインディング

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 の場合にとくにそう言えます。これは、手動でプロビジョニングされる PV にとくに当てはまります。超過を最小限にするために、OpenShift Online 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. 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、こちらを参照してください。

4.3.2.4. リリース

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、リソースを回収できるようになります。これにより、リソースをまた要求できるようになります。以前の要求側に関連する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。

4.3.2.5. 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、**Retain**、**Recycle**、または **Delete** のいずれかにすることができます。

Retain 回収ポリシーは、サポートするボリュームプラグインのリソースの手動による回収を許可します。**Delete** 回収ポリシーは、OpenShift Online 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るボリュームには、デフォルトの **ReclaimPolicy** 値の **Delete** が設定されます。手動でプロビジョニングされるボリュームには、デフォルトの **ReclaimPolicy** 値の **Retain** が設定されます。

4.3.2.5.1. 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度利用可能にします。

警告

recycle 回収ポリシーは動的プロビジョニングが優先されるために非推奨となり、今後のリリースでは削除されます。

4.3.3. 永続ボリューム

各 PV には、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。

永続ボリュームオブジェクトの定義

```
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. 永続ボリュームのタイプ

OpenShift Online 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 Persistent Disk
- iSCSI
- ファイバーチャネル
- Azure Disk
- Azure File
- 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。今後は属性として 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。

直接的なマッチングが常に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。サイズは予想されるものより多いか、またはこれと同等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間で順序付けすることはできず、タイプを選択することはできません。

同じモードの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。

アクセスモードは以下のようになります。

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。

以下の表では、異なる永続ボリュームによってサポートされるアクセスモードを一覧表示しています。

表4.1 永続ボリュームのサポートされるアクセスモード

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	▮	-	-

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

注記

- Pod が AWS EBS、GCE 永続ディスク、または Openstack Cinder PV に依存する場合、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. OpenShift Online の制限

OpenShift Online で永続ボリュームを使用する場合には以下の制限が適用されます。

重要

- PV は EBS ボリューム (AWS) でプロビジョニングされます。
- EBS ボリュームおよび GCE 永続ディスクは複数のノードにマウントできないので、RWO アクセスモードのみを使用できます。
- Docker ボリュームは無効にされます。
  - マップされた外部ボリュームがない VOLUME ディレクティブはインスタンス化できません。
- **emptyDir** はノード別のプロジェクト (グループ) ごとに 512 Mi に制限されます。
  - 特定のノード上にプロジェクトの単一 Pod がある場合、Pod は最大 512 Mi の **emptyDir** ストレージを消費できます。
  - 特定のノード上にプロジェクトに複数の Pod がある場合、それらの Pod は最大 512 Mi の **emptyDir** ストレージを共有します。
- **emptyDir** には Pod と同じライフサイクルがあります。
  - **emptyDir** ボリュームはコンテナーのクラッシュ/再起動後も永続します。
  - **emptyDir** ボリュームは Pod の削除時に削除されます。

4.3.3.5. 回収ポリシー

現時点で、回収ポリシーは以下のようになります。

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

recycle 回収ポリシーは動的プロビジョニングが優先されるために非推奨となり、今後のリリースでは削除されます。

4.3.3.6. 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。

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

4.3.4. 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。

Persistent Volume Claim (永続ボリューム要求、PVC) オブジェクト定義

```
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 **""** に設定された **StorageClass** または **storageClassName** アノテーションがストレージクラスなしの 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、これ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。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。

```
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
```

4.4. ソースコントロール管理

OpenShift Online 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 Online 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 Online 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 Online で自動的で作成することもできます。
- リポジトリーの変更時にビルドをトリガーします。

4.5. 受付コントローラー

4.5.1. 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、要求の認証および認可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 Online マスターには、それぞれのタイプのリソース (Kubernetes および OpenShift Online) 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 Online 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。

4.5.2. 一般的な受付ルール

OpenShift Online では、Kubernetes および OpenShift Online リソースの単一の受付チェーンを使用できます。これは、複数の個別の受付チェーンが使用されていた以前の状況からの変更点です。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、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した。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
```

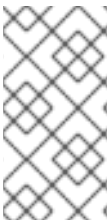
1 受付プラグイン名です。

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. 他の API オブジェクト

4.6.1. 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。

4.6.2. 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。

4.6.3. 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。

4.6.4. 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。

4.6.5. 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の耐久性のあるストレージを提供します。

4.6.6. 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に対して一致させ、それらをバインドします。この要求は、Pod によってボリュームとして使用されます。Kubernetes はボリュームがこれを要求する Pod と同じノードで利用可能であることを確認します。

4.6.6.1. 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。

4.6.7. OAuth オブジェクト

4.6.7.1. 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、認可コードフローの **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。認可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。

4.6.7.2. 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: []
```

4.6.7.3. 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 を交換するために認可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認可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。

4.6.7.4. 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認可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)。

4.6.8. ユーザーオブジェクト

4.6.8.1. Identity

ユーザーが OpenShift Online にログインする際に、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実行されます。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 Online に提供されます。

次に OpenShift Online は **UserIdentityMapping** でその **Identity** を検索します。

- **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

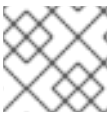
```
kind: "Identity"
apiVersion: "user.openshift.io/v1"
metadata:
  name: "anypassword:bob" (1)
  uid: "9316ebad-0fde-11e5-97a1-3c970e4b7ffe"
  resourceVersion: "1"
  creationTimestamp: "2015-01-01T01:01:01-00:00"
providerName: "anypassword" (2)
providerUserName: "bob" (3)
user:
  name: "bob" (4)
  uid: "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。

4.6.8.2. 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 % が含まれる OpenShift Online 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。

4.6.8.3. 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。

4.6.8.4. 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 ネットワーク

5.1. ネットワーク

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が推奨されます。

5.1.2. OpenShift Online 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。

そのため、OpenShift Online には DNS が組み込まれており、これにより、サービスは、サービス IP/ポートと共にサービス DNS によって到達可能になります。OpenShift Online 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
サービス	<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 ごとに 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。

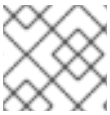
5.2. ルート

5.2.1. 概要

OpenShift Online ルートは、外部クライアントが名前で到達できるように **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。管理者は、OpenShift Online ルーターを実行する OpenShift Online ノードに解決するように DNS ワイルドカードエントリーを設定している可能性があります。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に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。



注記

ワイルドカードルートは OpenShift Online で無効にされます。