

DO288 Virtual Training

Red Hat OpenShift Development I: Containerizing Applications 復習セッション

2020年9月

レッドハット株式会社

トレーニングサービス部

はじめに

このセッションでは認定試験の準備として以下を実施する

- **トレーニングで学習したことを記憶に定着させる**
 - 管理者のタスクを整理してコマンドと対応づける
 - 何も見ないでコマンドが叩けるようになる
- **実機での演習を通してレジストリ操作の練習をする**
 - privateレジストリーへの操作を確実にできるようにする

Agenda

1. スケジュール
2. 演習環境の設定
3. 試験概要
 - a. 試験準備
 - b. 試験問題を解くときのコツ
4. 演習問題
5. 補足資料
 - a. OpenShiftリソース
 - b. oc 基本コマンド

スケジュール


- Day1 (9:30-17:30)
 - 午前:試験概要説明～oc 基本コマンド復習
 - 午後:演習
- Day2 (9:30-17:30)
 - 午前:エンタープライズレジストリ演習
 - 午後:演習

演習環境準備

演習環境の確認

1. ブラウザから rol.redhat.com を開きます
2. MY VIRTUAL TRAINING CLASSから **DO288**を探します。
3. DO288のクラスにある **JOIN CLASS** のリンクを開きます
4. **Labs(演習)**のタブを開きます

MY VIRTUAL TRAINING CLASSES



Red Hat OpenShift Administration I

DO288 |

Start Time

 Mon Jun 15 2020 11:00:00 GMT+0900 |

End Time

 Sat Jun 20 2020 10:30:00 GMT+0900

JOIN CLASS

PDFテキストのダウンロード

- **DOWNLOAD EBOOK**メニューから日本語PDFテキストをダウンロードします。

Class Dashboard - 20200626-NA-DO288 (43638292)

[Overview](#) [Enrollments 7](#) [Labs](#)

Course	DO288: Red Hat OpenShift Development I: Containerizing Applications	ENTER CLASSROOM
Description	Hands-on training to boost developer productivity powered by Red Hat OpenShift Red Hat OpenShift Development I: Containerizing Applications (DO288) enhances understanding of containers as a key technology for configuring and deploying applications and microservices. As the second course in the OpenShift development track, this offering will teach you how to design, build, and deploy containerized software applications to an OpenShift cluster. Whether you're writing container-native applications or migrating existing brownfield applications, you'll learn how to boost developer	<div>DOWNLOAD EBOOK ▾</div> <div>VIEW FAQ</div>

仮想マシンの起動

- 演習環境上で仮想マシンを作成して起動します。
- **workstation**のコンソールを開きます。
- RHELにログインします (ユーザ名 student、パスワード student)

The screenshot shows the 'Online Lab' interface. At the top, there are tabs for 'Table of Contents', 'Course', and 'Online Lab'. Below the tabs, there's a section titled 'OpenShift Details' containing a table with the following information:

Username	RHT_OCP4_DEV_USER	youruser
Password	RHT_OCP4_DEV_PASSWORD	yourpassword
API Endpoint	RHT_OCP4_MASTER_API	https://api.cluster.domain.example.com:6443
Console Web Application		https://console-openshift-console.apps.cluster.domain.example.com
Cluster Id		your-cluster-id

Below the details table, there are two buttons: 'SHUTDOWN LAB' and 'DELETE LAB'. To the right of these buttons is a timer showing '1 h 36 m' and a 'MODIFY' button. Below this, there's a table with two rows: 'classroom' and 'workstation'. Each row has a 'STARTED' status and two buttons: 'ACTION' and 'OPEN CONSOLE'. A callout box points to the 'OPEN CONSOLE' button for the 'workstation' row.

classroom	STARTED	ACTION	OPEN CONSOLE
workstation	STARTED	ACTION	OPEN CONSOLE

ブラウザ上にLinuxのデスクトップが開きます

試験概要

試験概要

- **試験範囲**

- 試験の作業領域はEX288のページで公開されている

- **試験時間**

- 3時間

- **試験形式**

- 実技試験
 - 実際の業務と同様のタスクを実行

- **合否通知**

- 機械採点
 - 採点基準の詳細は非公開
- メールによる通知(数日以内)

試験準備

- **タスクを整理する**

- 試験作業領域で書かれていることをタスクに分解
- トレーニングテキストを見ながら各タスクを実行するための手順、コマンドを整理する

- **正確に、早くタスクを実行する**

- 何も見ずにocコマンドを正確に打てるようにする
- 繰り返し練習して勝手に手が動くようにする
- 練習中はWebコンソールに頼らない

試験問題を効率よく解くコツ

試験問題を効率よく解くコツ

- 問題文を良く読む
- その場で使えるツールを駆使する
- コマンド入力補完やヒストリを使ってタイプを減らす
- 問題文に登場した固有名詞を手で入力しない
- 設定後に必ず動作を確認する
- 1つの問題に執着しない

問題文を良く読む

- 何ができたらOKなのかを確実に理解する
 - 3時間のうち1分くらい問題を眺めていても大差なし
 - 焦って思い込みで始めるのが一番危ない

その場で使えるツールを駆使する

- oc edit
- oc help
- oc explain
- man

oc editのエディタ変更

- 環境変数 KUBE_EDITOR or EDITORを変更
 - 例)
export KUBE_EDITOR=vim
export KUBE_EDITOR=gedit

コマンド入力補完やヒストリを駆使してタイプを減らす

- タブを打ってコマンドを補完する
 - `oc adm policy add-<TAB>`
- 一度入力したコマンドはシェルのヒストリから再度実行
 - `oc login -u kubeadmin -p xxxxx`
 - **Ctrl-r** `oc login`

コマンド入力補完やヒストリを駆使してタイプを減らす

- タブを打ってコマンドを補完する
 - `oc adm policy add-<TAB>`
- 一度入力したコマンドはシェルのヒストリから再度実行
 - `oc login -u kubeadmin -p xxxxx`
 - **Ctrl-r** `oc login`

問題文に登場した固有名詞を手で入力しない

- プロジェクト名、人物名、ディレクトリ名など、問題文からコピーする
 - 技術的には満点でも、スペルミスをしたら0点
 - `oc get <type>`の結果の名前をコピーして、`oc describe <type> <name>`にペーストする
 - ターミナルの上でコピー＆ペーストする方法を確認しておく
 - **Ctrl-Shift-c** (Copy)
 - **Ctrl-Shift-v** (Paste)

設定後に必ず動作を確認する

- 設定後に動作確認をするのが着実な合格への近道
 - 設定後に動作確認をして初めて自分の設定の誤りに気づく
 - 設定ミスに気付いたら、そこからは通常のトラブルシュートと同じ
- リソースを作成する前に確認すること
 - `oc project`でカレントプロジェクトを確認する
 - `oc whoami`で自分が誰かを確認する
- リソースを新規作成後に確認すること
 - `oc get <type>` で作成できていることを確認する。
- リソースを変更後に確認すること
 - 変更したつもりがエラーになっていて変更できていなことがよくある
 - `oc describe <type> <name>` または `oc get <type> <name> -o yaml`で変更した結果を確認する

試験領域

EX288を知る

- Red Hat 認定スペシャリスト試験 - OpenShift Application Development -

<https://www.redhat.com/ja/services/training/ex288-red-hat-certified-specialist-openshift-application-development-exam>

試験の領域

- **Red Hat OpenShift Container Platform の操作**
 - 複数の OpenShift プロジェクトの作成と操作
 - アプリケーションのヘルスマモニタリングの実装
- **コンテナイメージの操作**
 - コマンドライン・ユーティリティを使用したコンテナイメージの操作
 - コンテナイメージの最適化
- **アプリケーションのデプロイメントに関する問題のトラブルシューティング**
 - アプリケーションのデプロイメントに関する軽微な問題の診断と修正

試験の領域

- **イメージストリームの操作**
 - カスタム・イメージストリームの作成とアプリケーションのデプロイ
 - 既存の git リポジトリからのアプリケーションの取得
 - アプリケーションのデプロイメントに関する軽微な問題のデバッグ
- **構成マップの操作**
 - 構成マップの作成
 - 構成マップを使用した、アプリケーションへのデータ注入
- **S2I (Source-to-Image) ツールの操作**
 - S2I を使用したアプリケーションのデプロイ
 - 既存の S2I ビルダーイメージのカスタマイズ

試験の領域

- **フックとトリガーの操作**

- 提供されたスクリプトを実行するフックの作成
- フックの適切な動作のテストと確認

- **テンプレートの操作**

- JSON 形式または YAML 形式で記述された既存テンプレートの使用
- マルチコンテナ・テンプレートの操作
- テンプレートへのカスタムパラメータの追加

演習問題

この2日間の演習のルール

- OpenShiftの操作については、ocコマンドだけで実施すること
- 製品マニュアルは参照可能

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.2/

1回目

- 演習のソリューションを読んでコマンド・手順をメモに整理する

2回目

- マニュアルとメモだけで演習を解けるようにする

メモの作成例

メモで試験領域の内容とコマンドの対応関係を整理する。

- 構成マップの作成

```
oc create configmap myconf
```

```
--from-literal key1=value1
```

```
--from-literal key2=value2
```

- 構成マップを使用した、アプリケーションへのデータ注入

```
oc set env dc/mydcname
```

```
--from configmap/myconf
```

ガイド付き演習による復習

3章 ガイド付き演習: エンタープライズレジストリーの使用	外部レジストリーへのアクセス
3章ガイド付き演習: OpenShiftレジストリーの使用	内部レジストリーへのアクセス
3章ガイド付き演習: イメージストリームの作成	外部レジストリーからのイメージストリーム作成
4章 プローブのアクティブ化	build-env
4章 ガイド付き演習: ビルドのトリガー	外部レジストリーのビルダーイメージ利用
4章 ガイド付き演習: post-commit ビルドフックの実装	フックの作成とテスト
5章 ガイド付き演習: S2I ビルドのカスタマイズ	.s2i/binによるビルダーのカスタマイズ

演習

1章 OpenShiftクラスターでのアプリケーションのデプロイと管理	build-env
3章 エンタープライズコンテナイメージのパブリッシュ	外部レジストリからのアプリケーション作成 イメージストリームの共有
6章 OpenShiftテンプレートからのアプリケーション作成	テンプレート修正
10章 OpenShift のコンテナイメージの設計	イメージストリーム共有プロジェクト
10章 サービスのコンテナ化とデプロイ	シークレット作成と適用 環境変数の作成と適用 構成マップの作成と適用
10章 マルチコンテナアプリケーションのビルドとデプロイ	プローブ作成 テンプレート修正 テンプレートとイメージストリームの共有

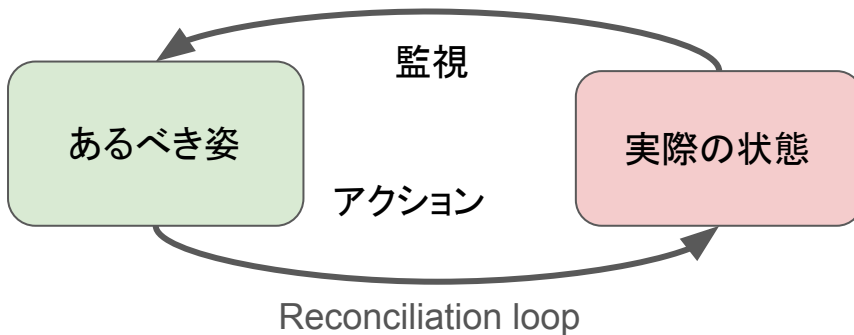
OpenShiftリソース

リソースとは

- コンテナの定義や管理をするためのOpenShift内部で保持される情報
開発者や管理者はリソースを作成、修正することでコンテナを管理する
- リソースの構造
 - **kind**
 - リソースの種類(Pod, Service, Deploymentなど)
 - **metadata**
 - リソースの名前やラベルなど
 - **spec**
 - あるべき姿を「宣言的」に記述したもの
 - **status**
 - システムによって自動的に更新される現在の状態

コントローラーとは

- コントローラとは、リソースに記述された「あるべき姿」と「実際の状態」を突き合わせて、両者に差異がある場合は、その差を埋めるように動作する内部モジュール
- リソースの種類ごとにコントローラーが存在する
 - 例)
 - Podリソースを監視して、コンテナが存在しなければ起動する
 - ReplicationControllerリソースを監視して、Podの数が指定されたものと一致しなければ、コンテナを起動したり、停止したりする



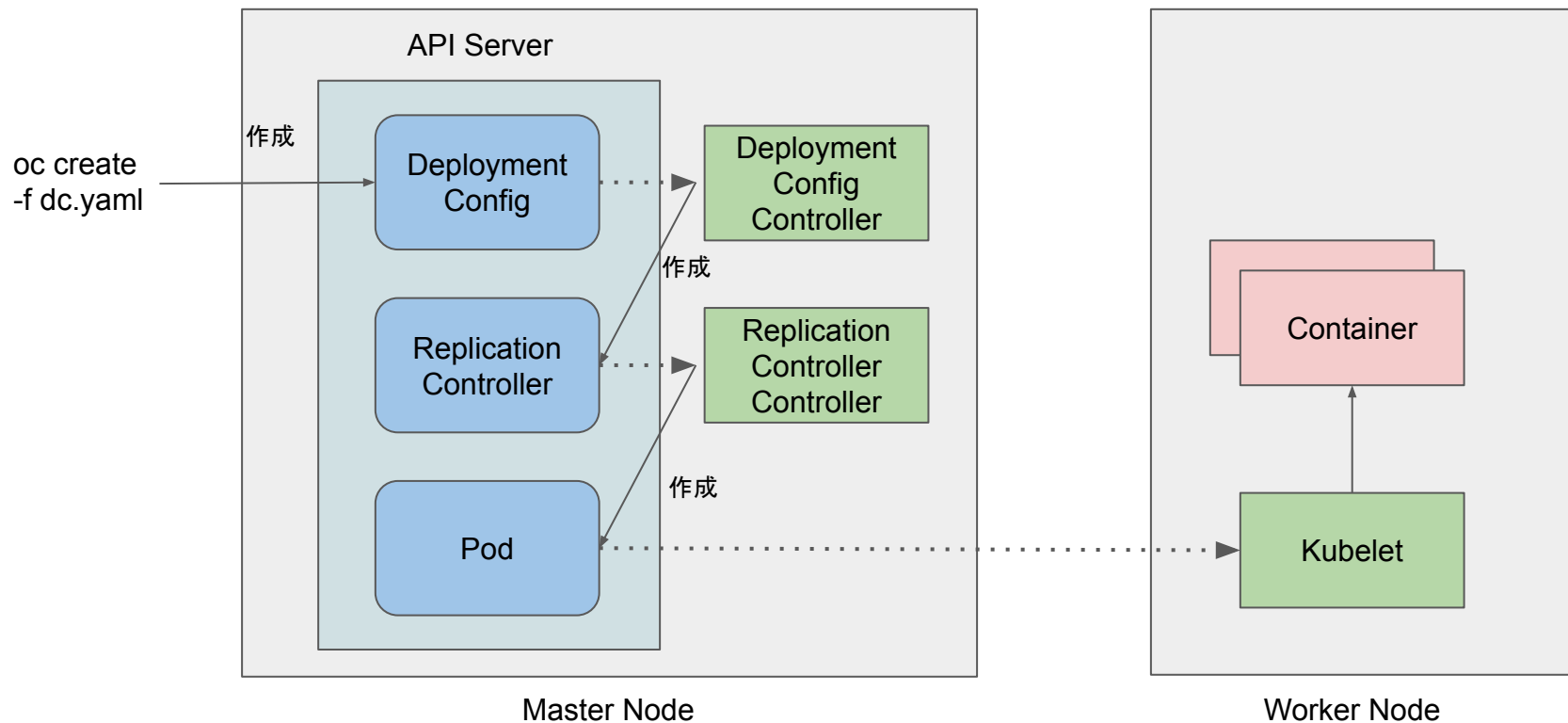
リソースとコントローラで自動化を実現する

ユースケース	リソース名
アプリケーションを起動し、正常性監視をする	Pod
アプリケーションの障害時に再起動する	ReplicationController
アプリケーションのバージョンアップに対応してデプロイする	DeploymentConfig
アプリケーションのスケールアップ、スケールダウンをおこなう	DeploymentConfig
ソースコードからアプリケーションイメージを作成する	BuildConfig
ロードバランサーの設定を自動でおこなう	Service
クラスターへの入り口とアプリケーションを連携させる	Route

デプロイ関連のリソース

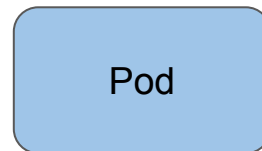
リソース名	意味	補足
DeploymentConfig	デプロイ設定	Podのデプロイ方法を定義する ReplicationControllerを使ってアプリケーションのデプロイを管理する
ReplicationController	Podの複製管理	Podを指定された数に維持する (Podが異常停止すると、自動的に再起動する) DeploymentConfigが生成されると、その情報から自動的にReplicationControllerが生成される。
Pod	デプロイの最小単位	Podの情報を定義する ReplicationControllerによって自動的にPodが生成される。

DeploymentConfigの作成の様子



Podリソース:コンテナを起動する

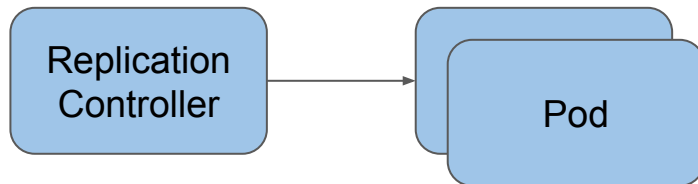
- **spec**
 - Pod情報
 - コンテナ情報(配列)
 - コンテナ名
 - イメージURL
 - 環境変数
 - 公開ポート番号
 - リソースリクエスト(CPU, Memory)
- **status**
 - Podの状態
 - IPアドレス



ReplicationControllerリソース: Podの数を維持する

- **spec**

- レプリカ数 (=Podの数)
- Selector
 - Podのラベル
- Pod情報
 - コンテナ情報
 - コンテナ名
 - イメージURL
 - 環境変数
 - 公開ポート番号
 - リソースリクエスト(CPU, Memory)



- **status**

- ReplicationControllerの状態
- 使用可能なレプリカの数

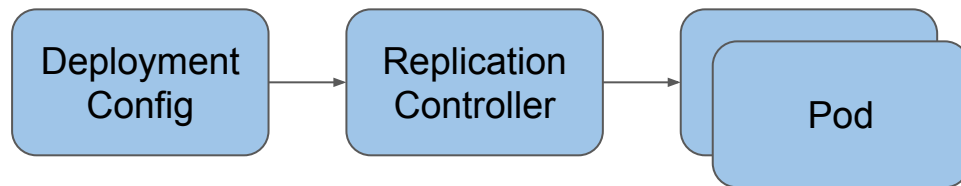
DeploymentConfigリソース: デプロイを管理する

- **spec**

- レプリカ数 (=Podの数)
- Selector
 - Podのラベル
- Pod情報
 - コンテナ情報(配列)
 - コンテナ名
 - イメージURL
 - 環境変数
 - 公開ポート番号
 - リソースリクエスト(CPU, Memory)
- デプロイ戦略(Rolling または Recreate)
- トリガー(イメージ変更、設定変更)

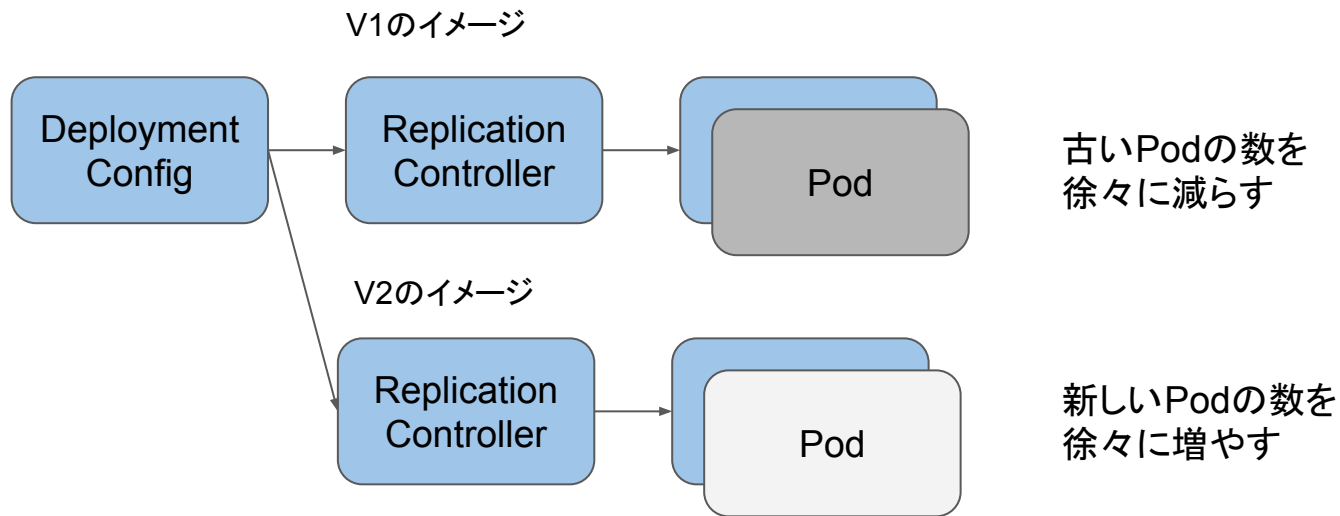
- **status**

- DeployConfigの状態
- 最新バージョン



ReplicationControllerを使った再デプロイの管理

デプロイ戦略: Rolling

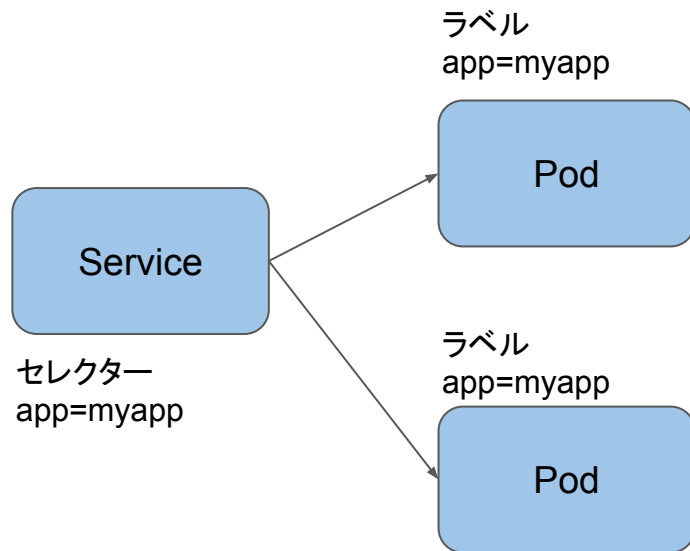


ネットワーク関連リソース

リソース名	意味	補足
Pod	デプロイの最小単位	動的なIPアドレスを持つ
Service	Podのロードバランサー	静的なIPアドレスを持つ selectorPodと関連付けられる
Route	クラスター外部からHTTP(S)でPodへのアクセスを可能にする	DNS名を提供する

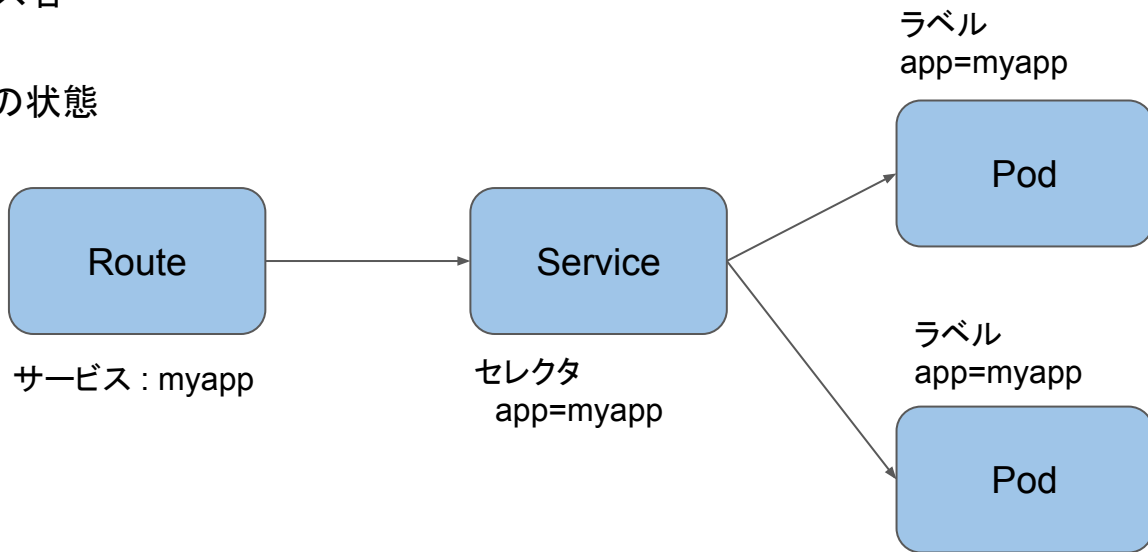
Serviceリソース: Podのロードバランサー

- **spec**
 - Cluster IPアドレス
 - 公開ポート番号
 - Selector
 - Podのラベル
- **status**
 - Serviceの状態



Routeリソース: サービスをクラスター外部に公開

- **spec**
 - ホスト名 (DNS名)
 - 公開ポート番号
 - サービス名
- **status**
 - Routeの状態

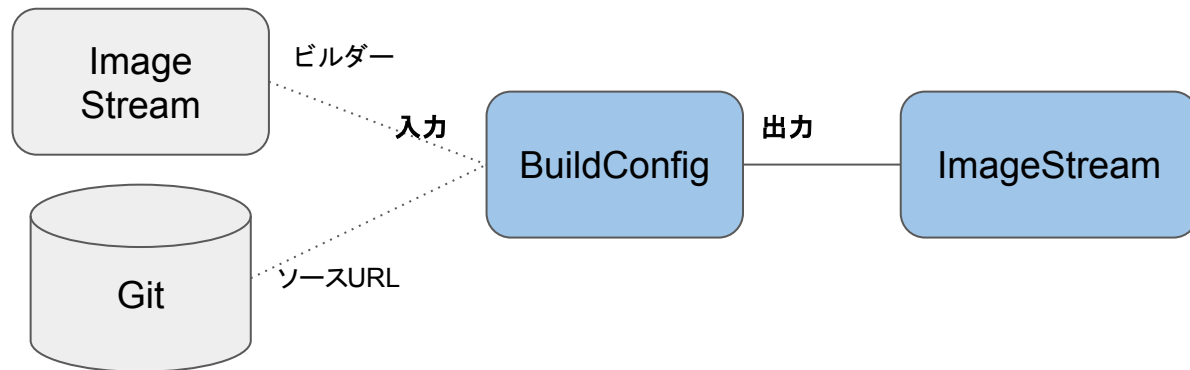


ビルド関連リソース

リソース名	意味	補足
BuildConfig	ビルド設定	ビルダーイメージとソースコードからアプリケーションイメージとイメージストリームを生成する ビルド戦略を定義する
Build	ビルド設定	ビルドごとに生成されるビルドの履歴 名前は、<BuildConfig名>-ビルド数、のようになる
Pod	ビルドPod	ビルドを実行するPod 名前は、<BuildConfig名>-ビルド数-ランダムな文字、のようになる

BuildConfigリソース:ビルド設定

- **spec**
 - ビルド戦略
 - 入力
 - ビルダー(ビルドPod)のイメージストリーム
 - ソースコードURL
 - 出力
 - アプリケーションのイメージストリーム
 - トリガー(イメージ変更、設定変更)
- **status**
 - 最新のバージョン



ImageStreamリソース: イメージ情報

- **spec**

lookupPolicy:

local: false

ImageStream

- **status**

- イメージ情報(タグからイメージ URLのマップ)

イメージストリーム(MySQL)の例:

8.0

registry.redhat.io/rhsc1/mysql-80-rhel7@sha256:62772b63c45a19a1559a8f13c103120f421a55d753a781dea1708f3053079457

5.7

registry.redhat.io/rhsc1/mysql-57-rhel7@sha256:9a781abe7581cc141e14a7e404ec34125b3e89c008b14f4e7b41e094fd3049fe

Source-to-Image (S2I)

Source-to-Imageとは

- Kubernetesでのコンテナアプリケーションの開発とデプロイ
 - **クラスタの外部**でアプリケーションイメージをビルドする
 - アプリケーションイメージをクラスタにデプロイする
- Source-to-Imageを使ったアプリケーションの開発とデプロイ
 - **クラスタの内部**でアプリケーションをビルドする
 - ビルドしたアプリケーションイメージを自動的にデプロイする

Source-to-ImageはOpenShift固有の機能

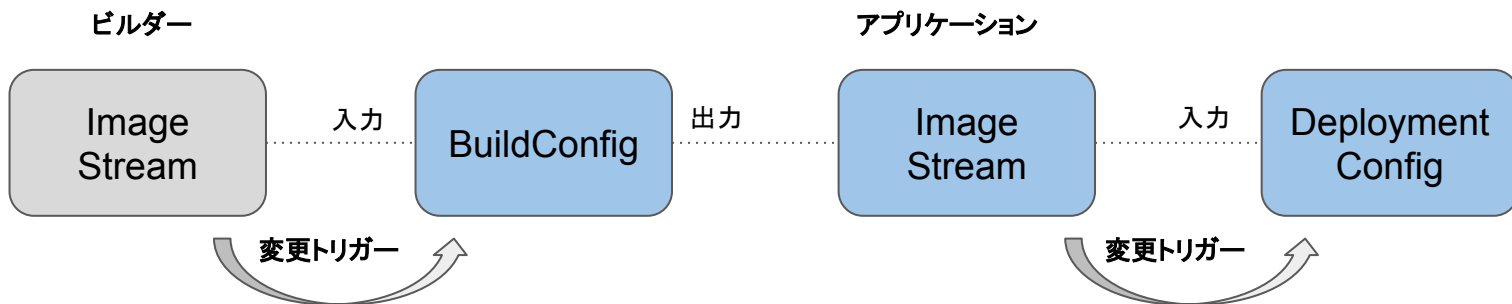
Source-to-Imageを使うと、ソースコードのビルドからイメージ作成、イメージのデプロイまでのプロセスがOpenShift内部で実行される。

ビルダーとは

- ビルダーとは、アプリケーションをビルドするためのPodのことで、プログラミング言語のツールやライブラリ、フレームワークを含む。ビルド後にアプリケーションのイメージを生成する。
- ビルダーの実行内容
 - **assemble**
 - 必要なライブラリーのダウンロード
 - コンパイルやリンクの実行
 - パッケージ作成
 - **run**
 - アプリケーション実行
- ビルダーの例
 - python, php, perl, ruby, nodejs, java
 - httpd, nginx
- カスタムビルダーを開発することができる(DO288のテーマの一つ)

Source-to-Image (S2I)におけるリソース間の依存関係

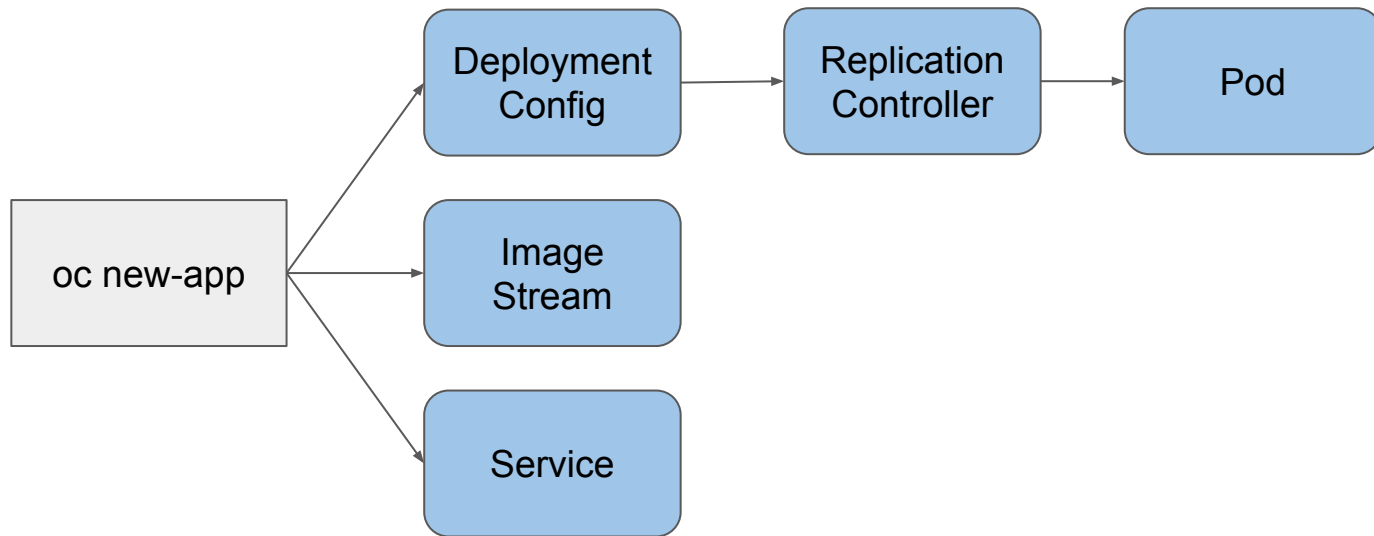
- **イメージ変更トリガー**
 - BuildConfigはビルダーのイメージストリームの変更トリガーを持つ
 - DeployConfigはアプリケーションのイメージストリームの変更トリガーを持つ
- **イメージ変更トリガーによるビルドとデプロイの連携**
 - BuildConfigはビルダーのイメージストリームが更新されるとビルドが開始される
 - ビルドの結果としてアプリケーションのイメージストリームが更新される
 - アプリケーションのイメージストリームが更新されるとデプロイが開始される



oc new-appから生成されるリソース

リソース名	意味	補足
BuildConfig	ビルド設定	ビルダーイメージとソースコードからアプリケーションイメージを生成する
DeploymentConfig	デプロイ設定	ReplicationControllerを使ってアプリケーションのデプロイを管理する
Service	Podのロードバランサー	selectorによってPodと関連付けられる静的なIPアドレスを持つ
ImageStream	イメージ管理	アプリケーションイメージの短縮名といメールURLを管理

oc new-app --docker-image <image URL>



oc new-appを使ったイメージ指定の例

- コマンド

```
oc new-app --name hello-limit \  
    --docker-image quay.io/redhattraining/hello-world-nginx:v1.0
```

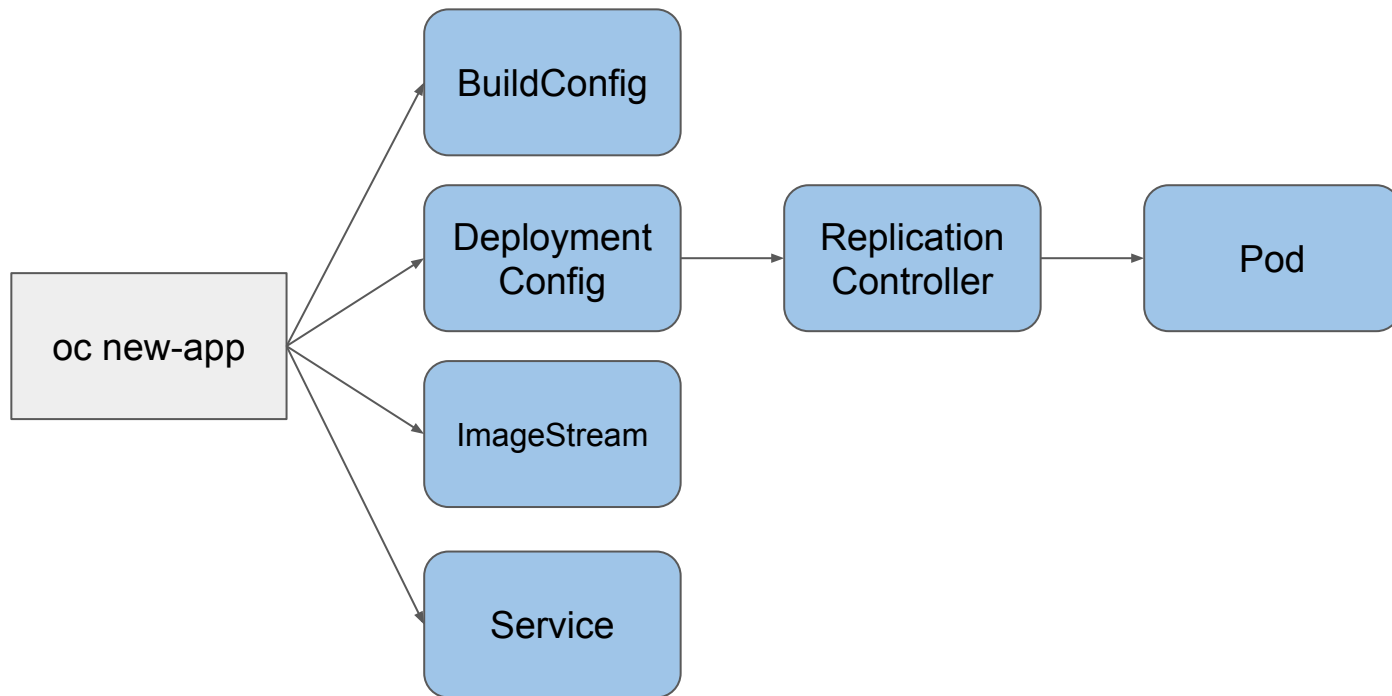
- パラメータ

- アプリケーション名: hello-limit
- イメージのURL: quay.io/redhattraining/hello-world-nginx:v1.0

- 動作

1. イメージストリーム作成
 - a. イメージURLからhello-limit ImageStreamを生成
2. デプロイ実行
 - a. DeploymentConfigにしたがってデプロイが開始される

oc new-app <imagestream>~<source URL>



oc new-appを使ったSource-to-Imageの例

- コマンド

oc new-app --name myapp nodejs:12~<https://github.com/sclorg/nodejs-ex.git>

- パラメータ

- アプリケーション名: myapp
- ビルダーイメージのイメージストリーム nodejs:12
- ソースコードのURL: <https://github.com/sclorg/nodejs-ex.git>

- 動作

1. ビルド実行

- a. BuildConfigにしたがってビルドが開始される
- b. コントローラーはnodejsイメージストリームからイメージのURLを特定する
- c. コントローラーはイメージURLからビルド用Pod (ビルダー) を起動する
- d. コントローラーはビルダーのPod内部 (/tmp/src) にGitのソースコードを展開
- e. ビルダーはコンテナ内部でビルドを実行してmyappイメージを生成
- f. ビルダーはmyappのイメージを内部レジストリにプッシュ

2. イメージストリーム更新

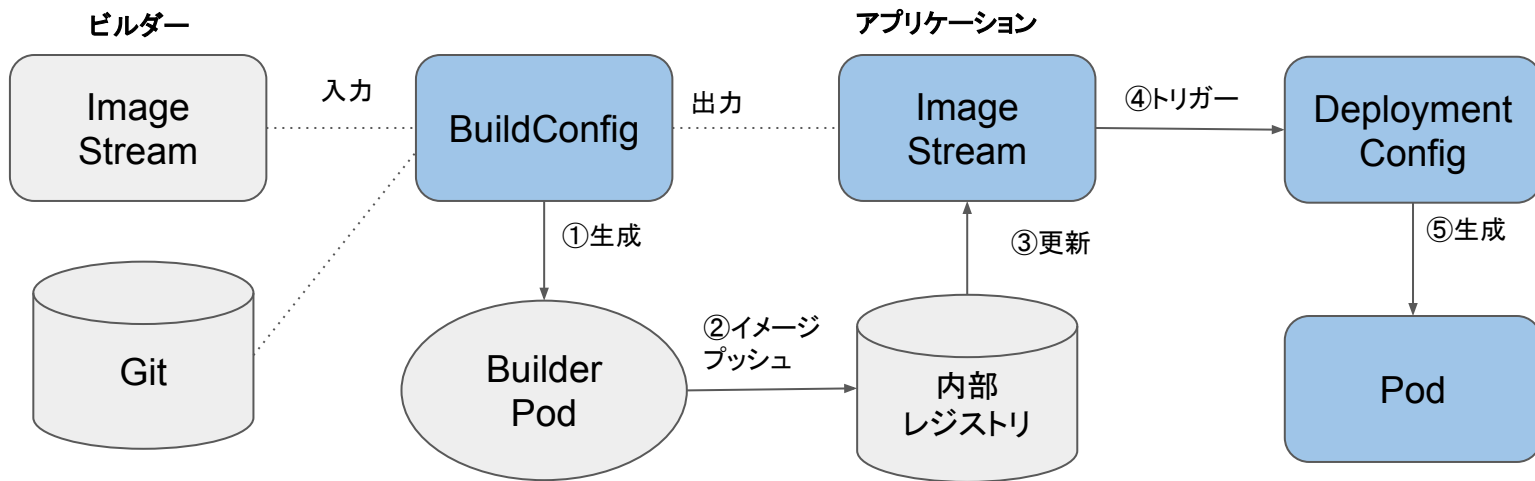
- a. 内部レジストリはmyappイメージ情報をmyappイメージストリームに設定

3. デプロイ実行

- a. myappイメージストリームの更新によってDeploymentConfigのトリガーがかかる
- b. DeploymentConfigによってデプロイが開始される

S2Iにおけるリソース間の依存関係

1. BuildConfigの設定にしたがってビルトPodが起動される
2. ビルダーPodはビルド後にアプリケーションイメージを内部レジストリにプッシュする
3. 内部レジストリはアプリケーションのイメージストリームを更新する
4. アプリケーションのイメージストリームが更新されると、それをトリガーにしてデプロイを実行



DeploymentConfig を修正する箇所

DeploymentConfigの構造

```
apiVersion:  
kind: DeploymentConfig  
metadata:  
...  
spec:  
  replicas: 1
```

```
  ..  
  template:  
    metadata:  
    ..  
    spec:
```

```
      containers:  
      - env:  
        name: USER  
        value: myname  
        image: quay.io/redhattraining/xxxxx  
        resources: {}
```

Pod情報

コンテナ情報

DeploymentConfigを修正する箇所とその理由

修正箇所	修正箇所	理由
replicas	dc.spec.replicas	ReplicationControllerに設定
env	dc.spec.template.spec.containers.env	環境変数はコンテナ単位
probe	dc.spec.template.spec.containers.probe	Probeはコンテナ単位

基本的なocコマンド

ocコマンド: ログイン

コマンド	意味	補足
<code>oc login -u <user> -p <password> <API server URL></code>	一般ユーザとしてログインする	認証が成功するとトークンが発行される
<code>oc whoami</code>	ログイン済ユーザ名を表示	
<code>oc whoami -t</code>	ログイン済ユーザのトークンを表示	
<code>oc logout</code>	ログアウトする	トークンが無効になる

ocコマンド: プロジェクト

コマンド	意味	補足
oc new-project <name>	プロジェクト新規作成	すでに作成済の名前であればエラーになる
oc project	現在プロジェクトを表示	
oc project <name>	現在プロジェクトを指定されたプロジェクトに変更する	
oc projects	プロジェクト名をリスト	自分の権限で見えるものだけ
oc delete project <name1> <name2> ..	プロジェクトを削除する (複数指定可能)	プロジェクトに含まれるすべてのリソースが削除される

ocコマンド: プロジェクトの指定

コマンド	意味	補足
<code>oc get <type> -n <project></code>	指定されたプロジェクト内で指定されたタイプのリソースを表示	例) \$ oc get template -n openshift
<code>oc get <type> <name> -n <project></code>	指定されたプロジェクト内で指定されたタイプ、名前のリソースを表示	例) \$ oc get secret localusers -n openshift-config

ocコマンド: ラベルの指定

コマンド	意味	補足
oc get all -l <key=value>	現在プロジェクト内で指定されたラベルに一致するリソースをすべて表示する	例) \$ oc get all -l app=myapp
oc delete all -l <key=value>	現在プロジェクト内で指定されたラベルに一致するリソースをすべて削除する	例) \$ oc delete all -l app=myapp

ocコマンド: get

コマンド	意味	補足
<code>oc get all</code>	現在プロジェクト内のすべてのリソースを表示	
<code>oc get <type></code>	現在プロジェクト内で指定されたタイプのリソースを表示	
<code>oc get <type> <name></code>	現在プロジェクト内で指定されたタイプ、名前のリソースを表示	
<code>oc get <type> <name></code> <code>-o wide</code>	現在プロジェクト内で指定されたタイプ、名前のリソースを表示 (表示されるカラムが増える)	Podの場合はスケジュールされたNodeの名前が表示される
<code>oc get <type> <name></code> <code>-o yaml</code>	現在プロジェクト内で指定タイプ、名前のリソースをYAML形式で表示	この結果をファイルにリダイレクトして編集することが多い

ocコマンド: describe

コマンド	意味	補足
oc describe <type>	現在プロジェクト内で指定されたタイプのリソースを詳細表示する	指定されたタイプのリソースが複数存在する場合は連続表示
oc describe <type> <name>	現在プロジェクト内で指定されたタイプ、名前のリソースを詳細表示	
oc describe pod <name>	現在プロジェクト内で指定された名前のPod詳細情報を表示	IPアドレスの取得など
oc describe dc <name>	現在プロジェクト内で指定された名前のDeploymentConfig詳細情報を表示	レプリカ数、Pod情報(イメージURL, 環境変数, プローブ, リソースリクエスト)の確認

ocコマンド: describeのタイプ別使い方

コマンド	意味	補足
oc describe pod <name>	現在プロジェクト内で指定された名前のPod詳細情報を表示	IPアドレスの取得など
oc describe dc <name>	現在プロジェクト内で指定された名前のDeploymentConfig詳細情報を表示	レプリカ数、Pod情報(イメージURL, 環境変数, プローブ, リソースリクエスト), アプリのイメージストリームの確認
oc describe svc <name>	現在プロジェクト内で指定された名前のService詳細情報を表示	Endpoints(対応するPod IPアドレスの集まり)の確認

ocコマンド: リソースの作成と編集

コマンド	説明
<code>oc create -f file.yaml</code>	ファイルからリソースを作成
<code>oc create deployment loadtest --dry-run --image quay.io/redhattraining/loadtest:v1.0 -o yaml > file.yaml</code>	コマンドからリソースを作成 (--dry-runはリソースを作成するが実行はしない)
<code>oc edit <type> <name></code>	現在プロジェクト内で指定されたタイプ、名前のリソースを編集する
<code>1. oc get <type> <name> -o yaml > file.yaml 2. vi file.yaml 3. oc apply -f file.yaml</code>	1. リソースをファイルに保存 2. ファイルを修正 3. 修正したファイルを適用

ocコマンド: create、apply、replaceの違い

コマンド	説明
oc create -f file.yaml	ファイルからリソースを新規作成
oc apply -f file.yaml	ファイルの内容をリソースに適用 該当リソースが存在しなければ新規作成
oc replace -f file.yaml	リソースを削除してから、新規作成

ocコマンド: デプロイの修正

コマンド	意味	補足
<code>oc set env dc/<dcname> -- from configmap/<cmname></code>	シークレットから環境変数をコンテナに設定する	Pod内にコンテナが複数ある場合は -c でコンテナ名を指定
<code>oc set probe dc/probes --liveness --get-url=http://:8080/healthz --initial-delay-seconds=2 --timeout-seconds=2</code>	リソースリクエストとリミットをコンテナに設定する	Pod内にコンテナが複数ある場合は -c でコンテナ名を指定
<code>oc scale dc/name --replicas=3</code>	レプリカ数を設定する	

ocコマンド: routeの作成

コマンド	意味	補足
oc expose svc <service>	サービス名からルートを作成する	

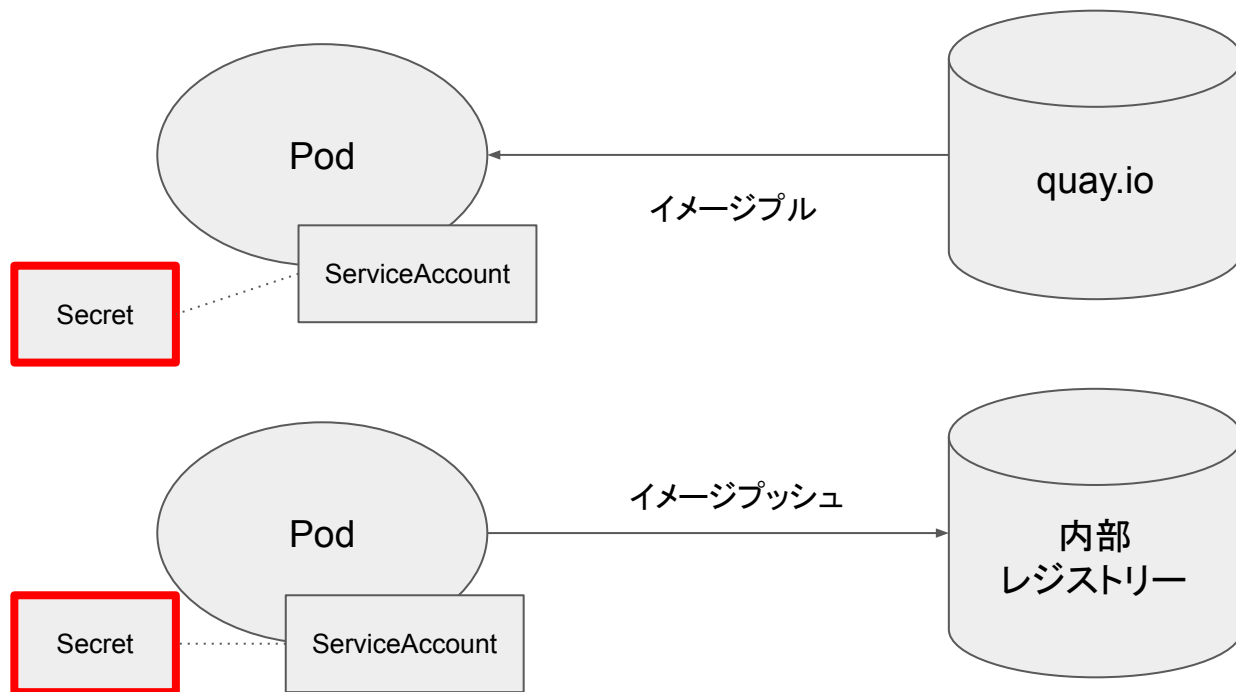
ocコマンド: delete

コマンド	意味	補足
oc delete <type> <name>	現在プロジェクト内で指定されたタイプ、名前のリソースを削除する	
oc delete all -l <key=value>	現在プロジェクト内で指定されたラベルに一致するリソースをすべて削除する	

privateレジストリの操作

privateレジストリとは

- イメージプルやプッシュに認証が必要なレジストリサーバー
- PodはServiceAccountに紐付いた認証情報を含む Secretを使ってレジストリにアクセスする



プロジェクトのサービスアカウント

サービスアカウント	説明
builder	ビルド Pod で使用されます。これには system:image-builder ロールが付与されます。このロールは、内部レジストリーを使用してイメージをプロジェクトのイメージストリームにプッシュすることを可能にします。
deployer	デプロイメント Pod で使用され、system:deployer ロールが付与されます。このロールは、プロジェクトでレプリケーションコントローラーや Pod を表示したり、変更したりすることを可能にします。
default	別のサービスアカウントが指定されていない限り、その他すべての Pod を実行するために使用されます。

https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.2/html/authentication/service-accounts-default_using-service-accounts

privateレジストリの利用

手順	説明
1.アクセストークンの取得	<code>podman login -u \${RHT_OCP4_QUAY_USER} quay.io</code>
2.アクセストークンからシークレット作成	<code>oc create secret generic quayio \ --from-file .dockerconfigjson=\${XDG_RUNTIME_DIR}/containers/auth.json \ --type kubernetes.io/dockerconfigjson</code>
3.シークレットをサービスアカウントに関連付ける	<code>oc secrets link default quayio --for pull</code>
4.アプリケーション作成	<code>oc new-app --name sleep \ --docker-image quay.io/\${RHT_OCP4_QUAY_USER}/ubi-sleep:1.0</code>

privateレジストリからのイメージストリーム作成

手順	説明
1.アクセストークンの取得	podman login -u \${RHT_OCP4_QUAY_USER} quay.io
2.アクセストークンからシークレット作成	oc create secret generic quayio \ --from-file .dockerconfigjson=\${XDG_RUNTIME_DIR}/containers/auth.json \ --type kubernetes.io/dockerconfigjson
3.シークレットをサービスアカウントに関連付ける	oc secrets link builder quayio
4.イメージストリーム作成	oc import-image s2i-do288-go \ --from quay.io/\${RHT_OCP4_QUAY_USER}/s2i-do288-go --confirm
5.アプリケーション作成	oc new-app --name greet \ s2i-do288-go~https://github.com/<User>/DO288-apps#custom-s2i --context-dir=go-hello

イメージストリームの共有

手順	説明
1.アクセストークンの取得	<code>podman login -u \${RHT_OCP4_QUAY_USER} quay.io</code>
2.アクセストークンからシークレット作成	<code>oc create secret generic quayio \</code> <code>--from-file .dockerconfigjson=\${XDG_RUNTIME_DIR}/containers/auth.json \</code> <code>--type kubernetes.io/dockerconfigjson</code>
3.イメージストリーム作成	<code>oc import-image todo-frontend --confirm \</code> <code>--reference-policy local \</code> <code>--from quay.io/\${RHT_OCP4_QUAY_USER}/todo-frontend</code>
4.プロジェクトにアクセス権を付与	<code>oc policy add-role-to-group \</code> <code>-n \${RHT_OCP4_DEV_USER}-review-common system:image-puller \</code> <code>system:serviceaccounts:\${RHT_OCP4_DEV_USER}-review-dockerfile</code>
5.イメージストリーム利用	<code>oc new-app --name frontend \</code> <code>> -e BACKEND_HOST=api.example.com \</code> <code>> -i \${RHT_OCP4_DEV_USER}-review-common/todo-frontend</code>

内部レジストリの利用

手順	説明
1.内部レジストリ取得	<pre>INTERNAL_REGISTRY=\$(oc get route default-route \ -n openshift-image-registry -o jsonpath='{.spec.host}')</pre>
2.アクセストークンの取得	<pre>TOKEN=\$(oc whoami -t) sudo podman login -u \${RHT_OCP4_DEV_USER} \ -p \${TOKEN} \${INTERNAL_REGISTRY}</pre>
3.イメージダウンロード	<pre>sudo podman pull \ \${INTERNAL_REGISTRY}/<プロジェクト名>/ubi-info:1.0</pre>
4.イメージからコンテナを開始	<pre>sudo podman run --name info \ \${INTERNAL_REGISTRY}/<プロジェクト名>/ubi-info:1.0</pre>

トラブルシューティング

トラブルシュート基本コマンド

コマンド	意味	補足
<code>oc logs -f <name></code>	Podのログを表示する	アプリケーションのエラーの原因がわかる
<code>oc get events</code>	イベントを表示する	エラーに至った経緯がわかる
<code>oc describe pod <name></code>	Podの詳細情報を表示する	Podに関わるイベントを表示する
<code>oc describe node <name></code>	Nodeの詳細情報を表示する	Nodeに関わるイベントを表示する
<code>oc rsh <name></code>	コンテナの中にシェルを開く	コンテナ内部の設定ファイルを確認できる

アプリケーションが起動しない

Podのステータス	意味	調査方法
Pending	スケジューリング失敗	oc describe pod <name> oc get events
ErrImagePull	イメージプル失敗	skopeo inspect
ImagePullBackoff	イメージプル失敗(繰り返し)	skopeo inspect
Error	実行時エラー	oc logs <name>
CrashLoopBackOff	実行時エラー(繰り返し)	oc logs <name>
OOMKilled	メモリ不足による強制終了	pod.spec.containers.resources. requests

参考リンク

参考リンク

[1] CLI を使用したアプリケーションの作成

https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.2/html/applications/creating-applications-using-cli

[2] イメージプルシークレットの使用

https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.2/html/images/using-image-pull-secrets

[3] イメージレジストリー Operator の設定パラメーター

https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.2/html-single/registry/index#registry-operator-configuration-resource-overview_configuring-registry-operator

Thank you.

red.ht/labs



linkedin.com/company/red-hat



facebook.com/redhatinc



youtube.com/user/RedHatVideos



twitter.com/RedHatLabs