

Red Hat OpenShift Platform

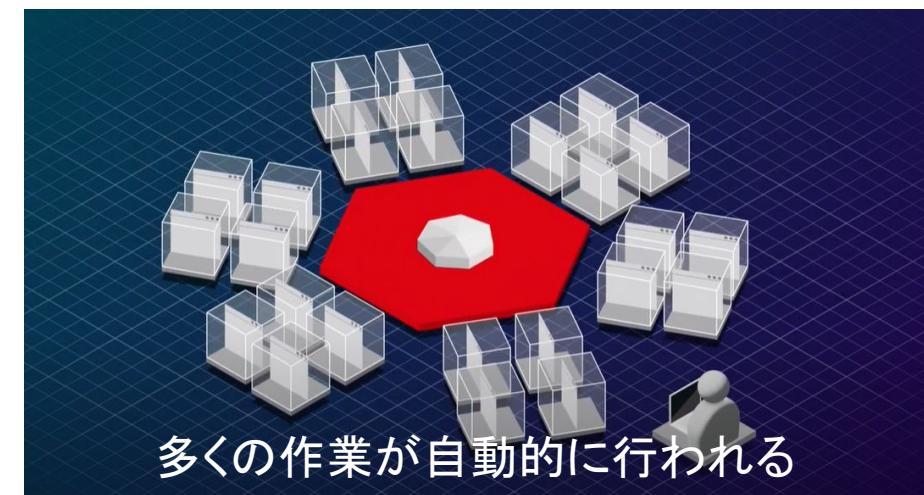
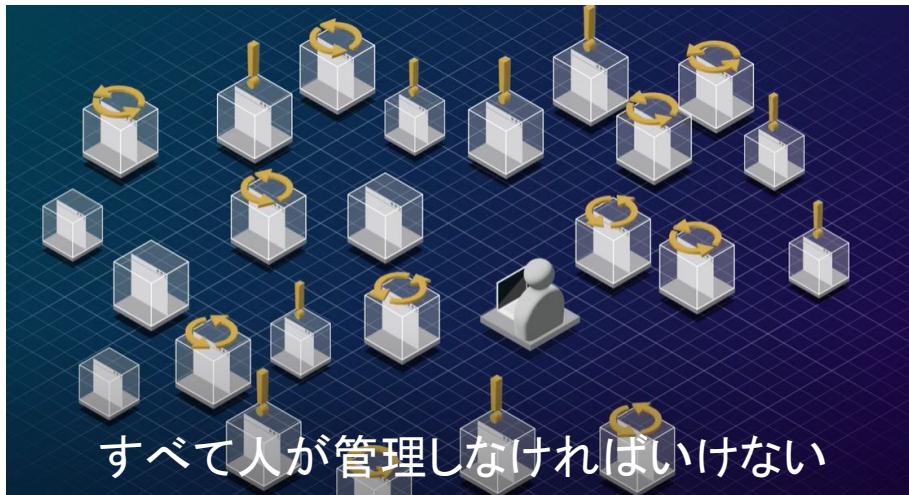
ハンズオンワークショップ

OpenShiftとは

Why we require OpenShift?

コンテナオーケストレーション

マイクロサービス化によってコンテナの数が増えれば増えるほど、管理が複雑化します。
コンテナオーケストレーションは、**コンテナの管理・運用を自動化**するためのソリューションです。



既存の運用スタイル

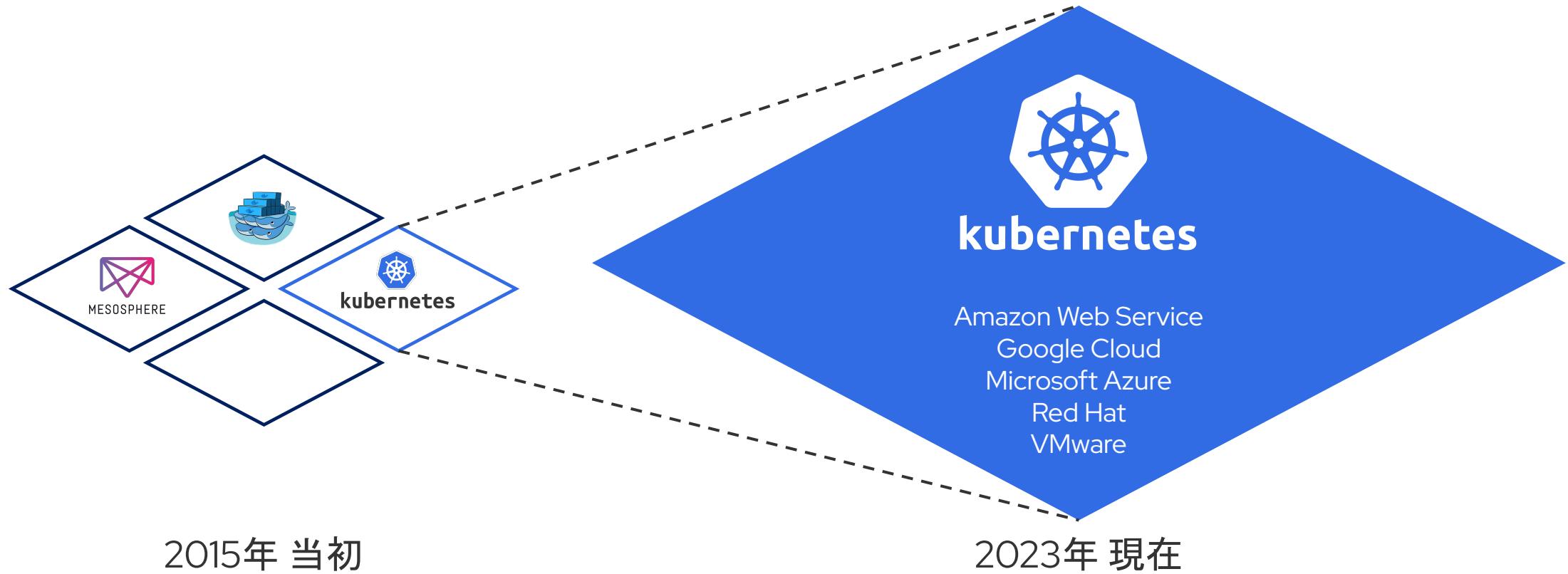
- ・属人的な障害復旧オペレーション
- ・手動によるコンテナ変更作業
- ・アプリケーションごとの設定管理
- ・定期的な監視作業

今後の運用スタイル

- ・ビジネス変化に応じた適切なリソース調整

コンテナオーケストレーションの歴史

企業のコンテナオーケストレーションの選択肢として「Docker」と「Mesosphere」は競い合ってきました。その後、Kubernetesが急激に支持を広げ、今では**デファクトスタンダードの地位**を得ています。



Kubernetesができること

Kubernetes(k8s)とは、**コンテナの運用操作を自動化するオープンソースのコンテナオーケストレーション**です。

Kubernetesを使用することにより、コンテナ化されたアプリケーションのデプロイやスケーリングに伴う、運用負担を軽減することができます。



アクセス負荷分散



コンテナの死活監視



リソースの制御



Bare metal



Virtual



Private cloud



Public cloud



Edge

Kubernetesだけではできないこと

Kubernetesはコンテナの管理、運用に役立つ機能を提供しますが、それ単体だけではできないこともあります。コンテナのビルドやミドルウェアの管理には、Kubernetes以外のツールの連携が必要です。

Kubernetesでは提供されない機能

コンテナの動的
ビルド/デプロイ

ミドルウェア
の管理

クラスタの
ロギングや監視

コンテナの
セキュリティ

クラスタ
アップグレード



kubernetes

Linux



Bare metal



Virtual



Private cloud



Public cloud



Edge

Red Hat OpenShift

エンタープライズに求められる機能をKubernetesに付随し、サポートすることで、ビジネス価値に直結する機能を提供しています。**アプリケーション開発の効率化に重きを置く**か、まずはインフラ運用の効率化に取り組むか、という点がKubernetes単体と大きく異なる点です。



コンテナの動的
ビルド/デプロイ

ミドルウェア
の管理

クラスタの
ロギングや監視

コンテナの
セキュリティ

クラスタ
アップグレード



Bare metal



Virtual



Private cloud



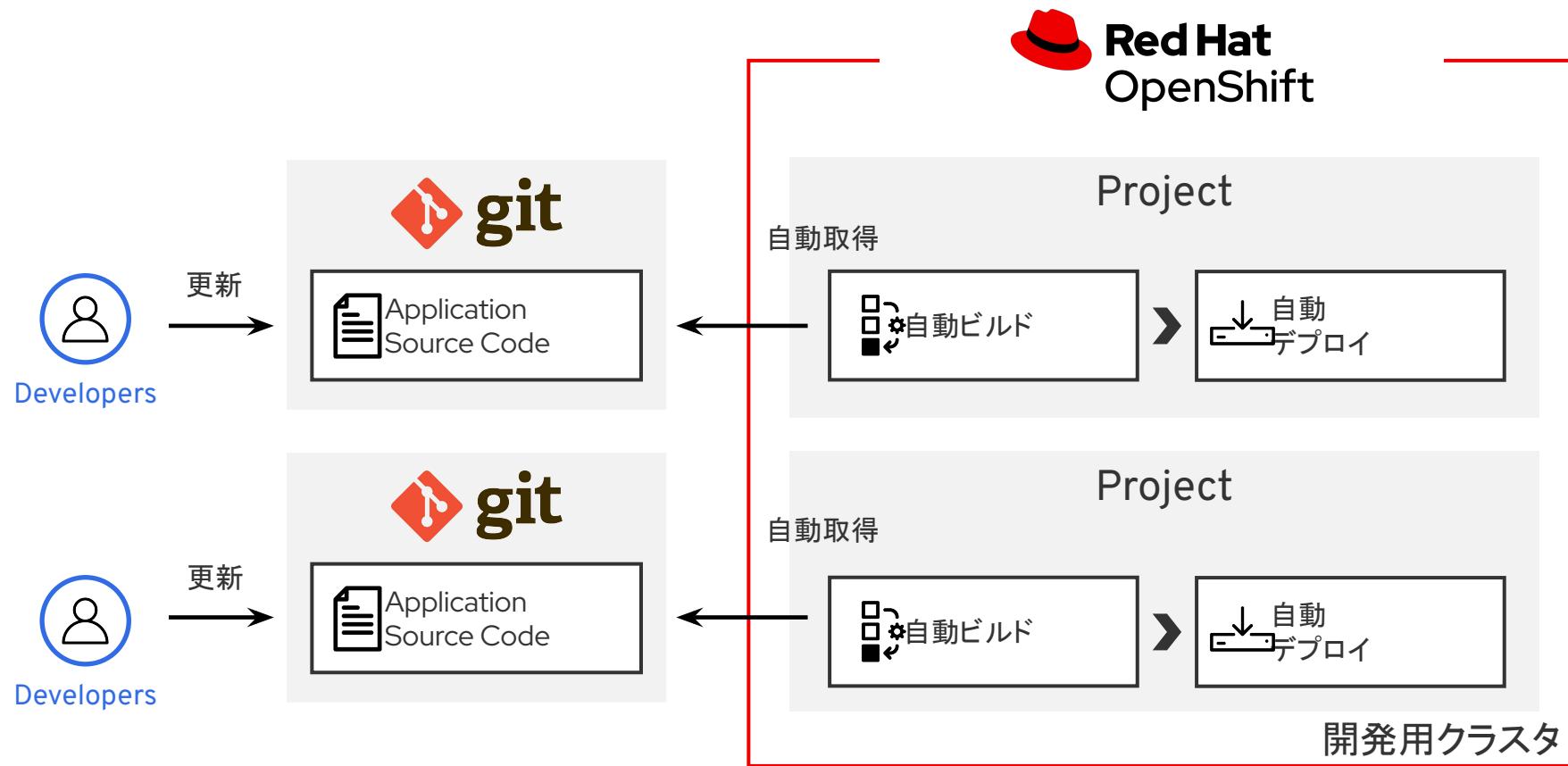
Public cloud



Edge

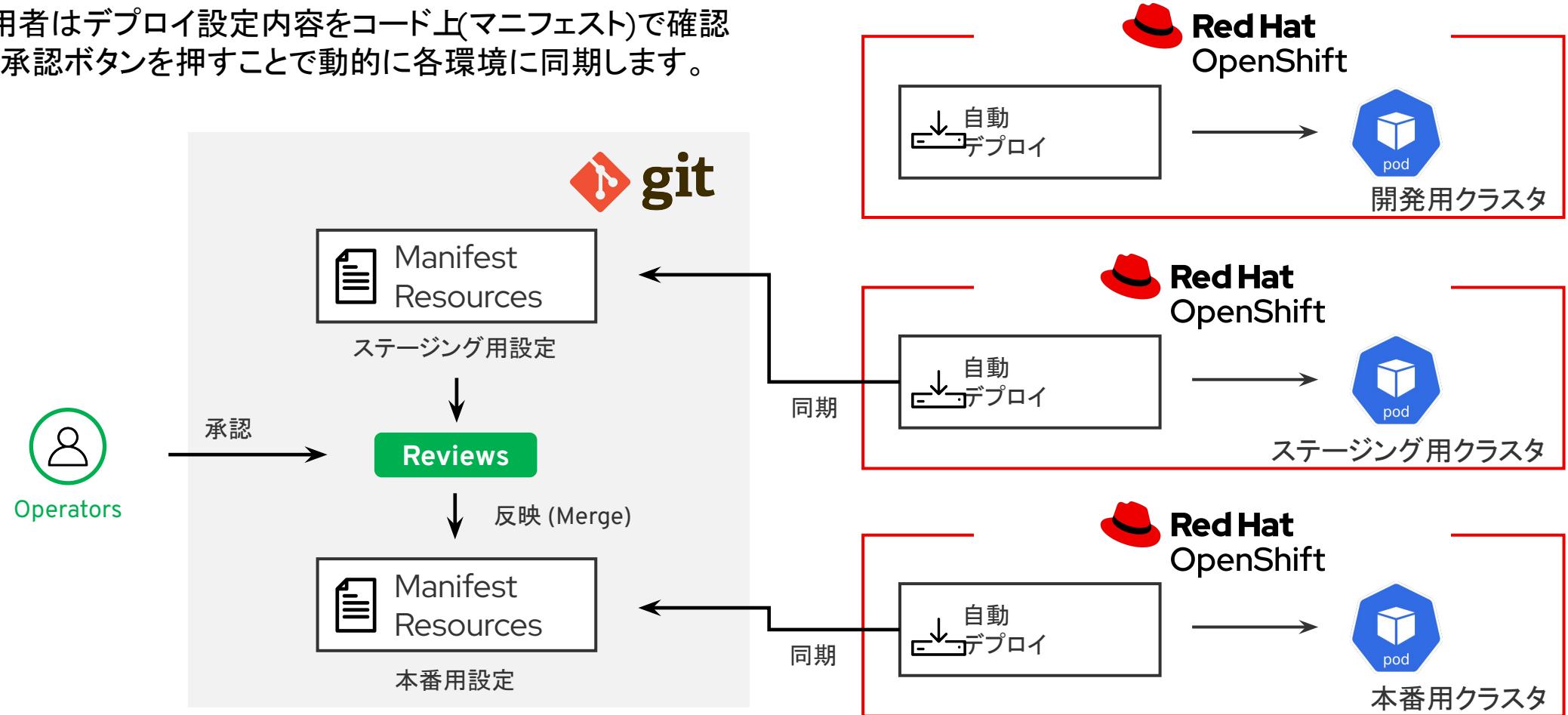
OpenShiftを活用したコンテナ開発

コンテナ開発では、開発者が新しく作ったソースコードをGitリポジトリに格納するだけで、動的にビルド、テスト、デプロイが行われます。従来のように、運用者に開発リソースを個別に用意してもらうことがないため、**開発スピードが飛躍的に向上します。**



OpenShiftを活用した継続的デリバリ

運用者はデプロイ設定内容をコード上(マニフェスト)で確認し、承認ボタンを押すことで動的に各環境に同期します。

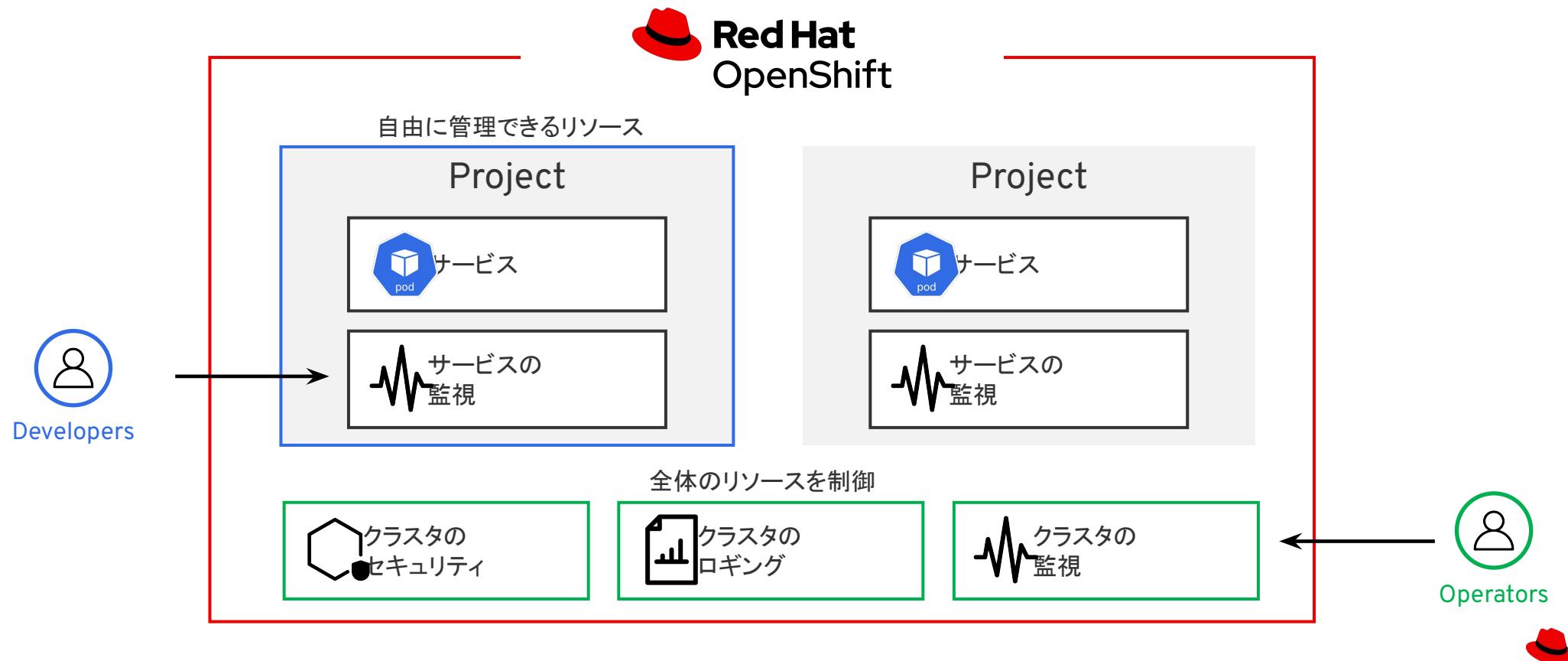


⁹ 従来の手動作業はなくなりますが、人の判断をもとにした安全な動的デプロイが可能です。

OpenShiftを活用したサービスの管理

開発者は自身が関わるサービスの安定性を各チームで見ることができます。運用者はクラスタの管理に専念できます。

従来のようにアプリごとの環境差異や監視設定を運用者が統合する必要がなく、**適材適所にリソースをセキュアに管理できます。**



OpenShiftの活用

Kubernetesのコンポーネント



Master Node (3 nodes)

Master Nodesには、コンテナを制御するコンポーネントと、クラスタの状態を構成管理するデータ(etcd)があります。

Controllers: リソースのデプロイを行う機能

Schedulers: リソースの空き状況を管理する機能



Worker Node (2~N nodes)

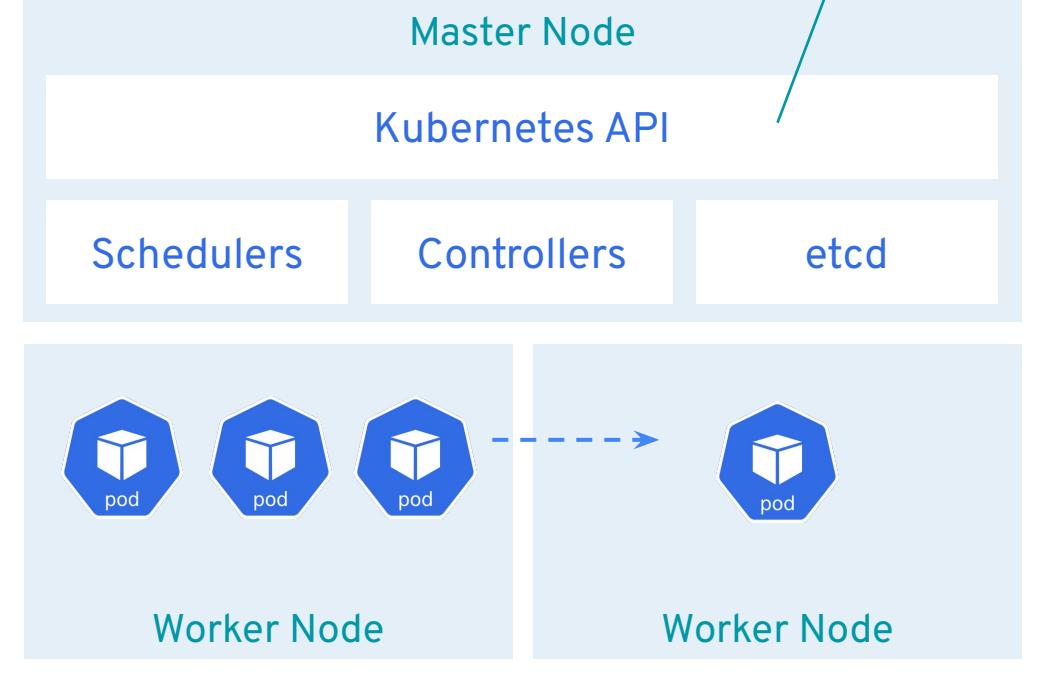
Worker Nodesでは、スケジュールされたコンテナがデプロイされ、コンテナを死活監視します。

リソース容量が足りない場合は、Worker Nodesを追加します。

クラスタリソースの管理、作成、構成に使用されるインターフェース。管理者はこのAPIに指示を行います。



kubernetes



OpenShiftのコンポーネント

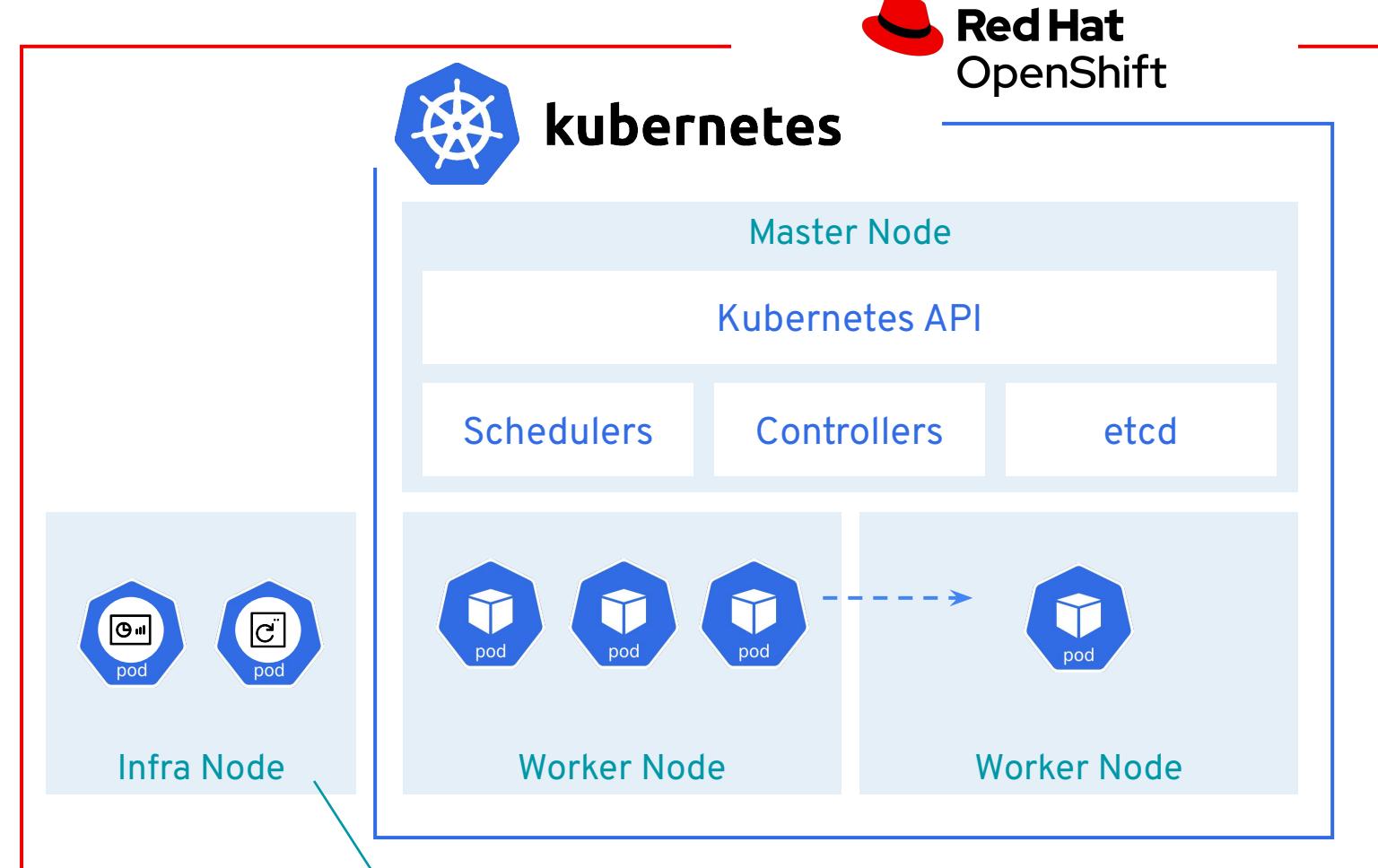
Infra Node (3 nodes)

Infra Nodesには、OpenShiftの追加サービスを管理するためのコンポーネントが配置されます。

Infra Nodesに配置できる機能

- Router
- OpenShift-included Registry
- OpenShift cluster monitoring
- OpenShift log aggregation

など



* 設定上はWorker Nodesにラベルが付いたものです。

OpenShiftのSubsでサポートする機能

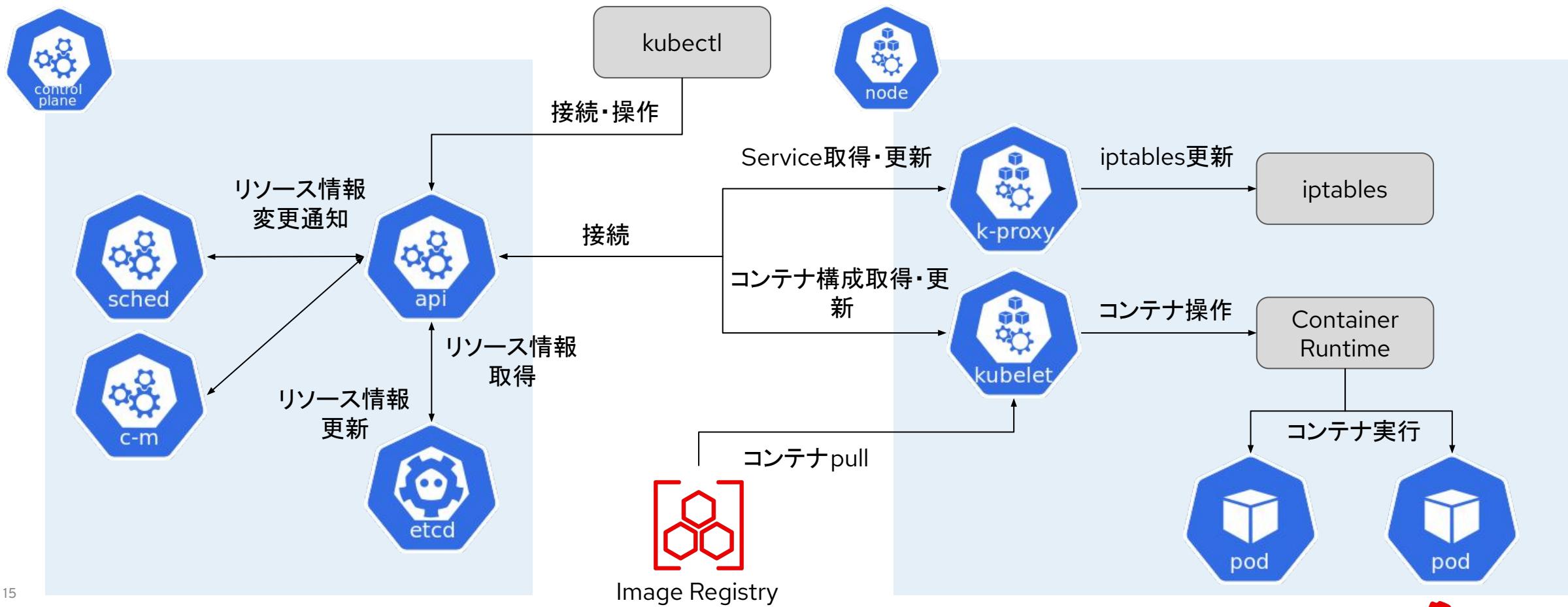
機能	OpenShiftの機能名	概要
Embedded OS	Red Hat CoreOS	コンテナ専用のセキュアな軽量OS、動的にバージョンアップが可能 Kubernetesに最適なコンテナランタイム(CRI-O)を活用
Web Portal		クラスタや各Projectの管理を行うためのWeb UI
Virtualization	OpenShift Virtualization	kubevirtを利用した仮想マシンのコンテナ化支援
CI	OpenShift Pipelines	Tekton Pipelines, Tekton Triggersによる継続的インテグレーション
CD	OpenShift GitOps	Argo CDを活用したGitOpsベースの継続的デリバリ
Ingress	HAProxy Ingress Controller	Ingressコントローラ
Service Mesh	OpenShift Service Mesh	Istio (Maistra), Kiali, Jaegerを活用したサービスメッシュ
Serverless	OpenShift Serverless	Knativeベースのサーバーレス
API Security	Gatekeeper Operator	OPAを使用したAPIリクエスト検証用のアドミッションコントローラ

* OpenShiftのSubscriptionでサポートできるものの一部をしています。

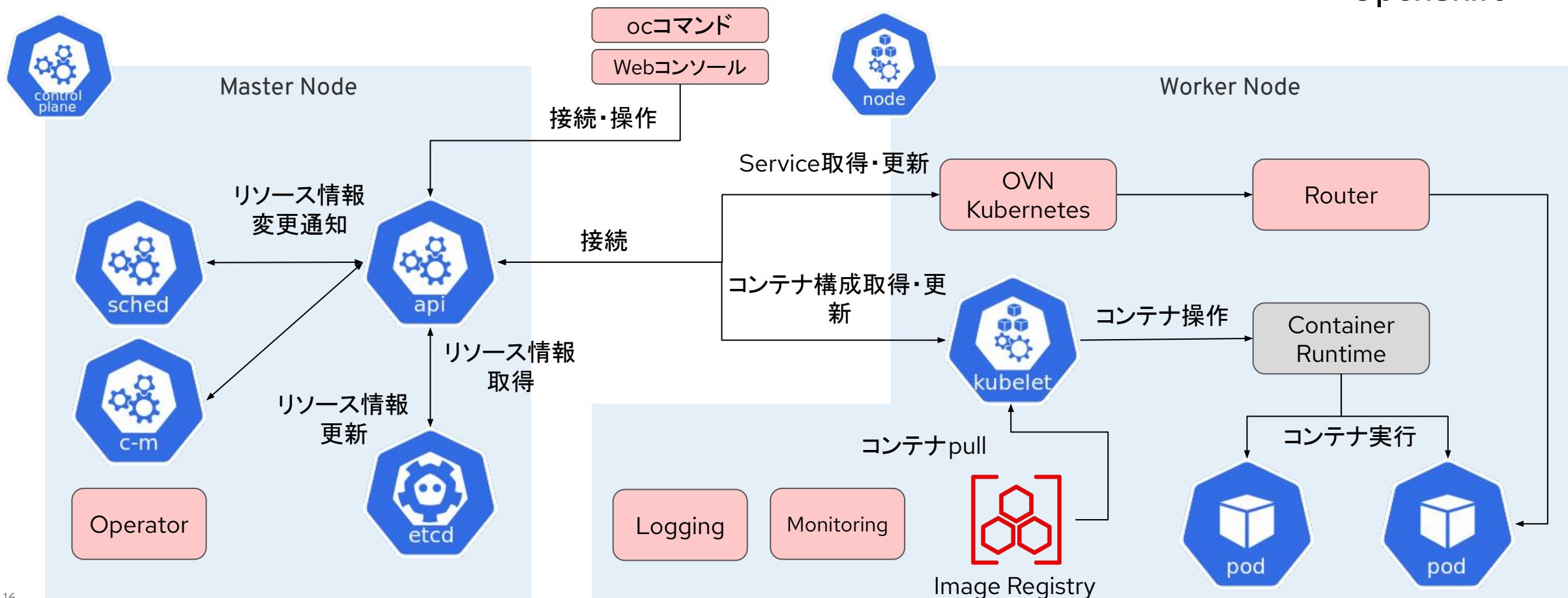


Feature summary : https://docs.openshift.com/container-platform/latest/welcome/oke_about.html

Kubernetesのアーキテクチャ



OpenShiftのアーキテクチャ



OpenShift基本操作と リソース



本日の目標

- OpenShiftをWebコンソールおよびCLIで操作する
- OpenShiftにおけるリソースを理解する
- アプリケーションをデプロイする
- 永続ボリュームの動作を確認する

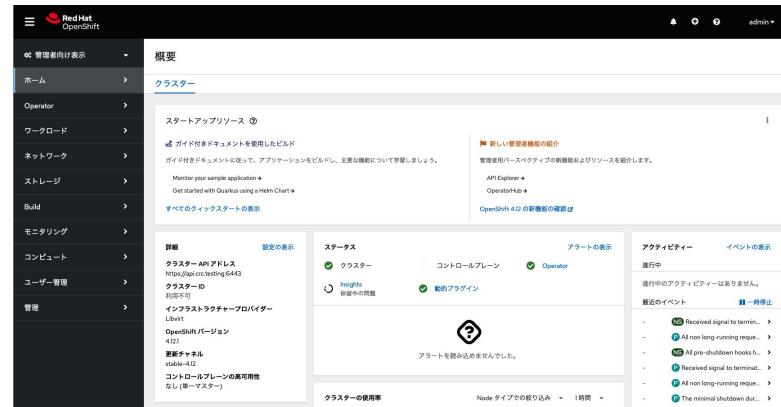
本日扱わないこと

- OpenShiftのインストール方法
- 管理者権限でのクラスタの運用管理
- モニタリング・ロギング

OpenShiftクラスタの操作インターフェース

OpenShiftクラスタの操作方法にはGUIであるWebコンソールとCLI(ocコマンド)があります

GUI(Webコンソール)

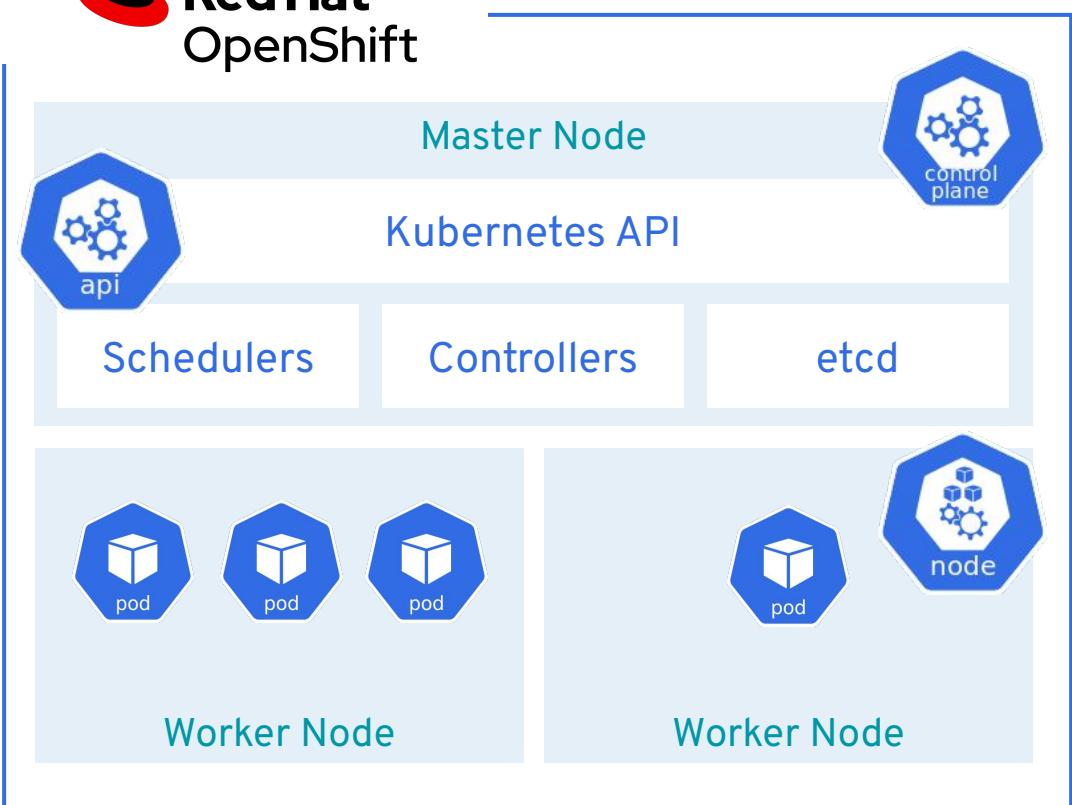


CLI(ocコマンド)

```
[root@crc-rwzd-master-0 ~]# oc get builds
NAME          TYPE        FROM      STATUS    STARTED      DURATION
blog-django-py-1   Source   Git9c4f338  Complete  45 minutes ago  1m39s
[root@crc-rwzd-master-0 ~]#
```



Red Hat
OpenShift



OpenShift Webコンソール

Webコンソールには2つのPerspectiveがあります。

管理者向け表示(Administrator Perspective)

- クラスタ管理者向けにクラスタ全体のシステムリソースや個々のオブジェクトの検索性など管理者観点の操作にフォーカス

The screenshot shows the 'Administrator Perspective' of the OpenShift Web Console. The left sidebar includes links for Home, Operator, Workload, Network, Storage, Build, Monitoring, Compute, User Management, and Administration. The main content area displays the 'Cluster' overview, featuring sections for 'Start Application Resources', 'Cluster Status' (with tabs for Cluster, Control Plane, and Operator), and 'Alerts'. A sidebar on the right provides details about the cluster's API address, version, and update channel.

開発者向け表示(Developer Perspective)

- 開発者向けにアプリケーションのトポロジや相関関係の表示などコンテナのデプロイや管理にフォーカス

The screenshot shows the 'Developer Perspective' of the OpenShift Web Console, specifically for the 'myproject' namespace. The top navigation bar shows the project name and the 'Developer Perspective' dropdown. The main content area displays the 'Topology' view, which shows a single pod named 'blog-django-py'. To the right, there are detailed panels for 'Logs' (showing recent log entries), 'Pod' (listing the pod), 'Build' (showing build status), and 'Service' (listing the service). The bottom right corner shows a link to 'Logs' for the pod.

CLI(ocコマンド)

CLIはocコマンドと言います。認証やプロジェクト操作、アプリケーション操作、情報の参照や編集などを行います。

- コマンドラインから使用できるOpenShift 操作ツール
 - Linux, Windows, Mac用バイナリがあります
- 形式: oc command [option] [type] object [flags] [command-options]
 - 認証: login, logout
 - プロジェクト操作: new-project, projects, project
 - アプリケーション操作: new-app
 - 情報参照・編集: get, describe, export, edit, patch, delete, types
 - ルーティング: expose
 - 調査・トラブルシューティング: logs, exec, rsh, rsync
 - ヘルプ表示: help

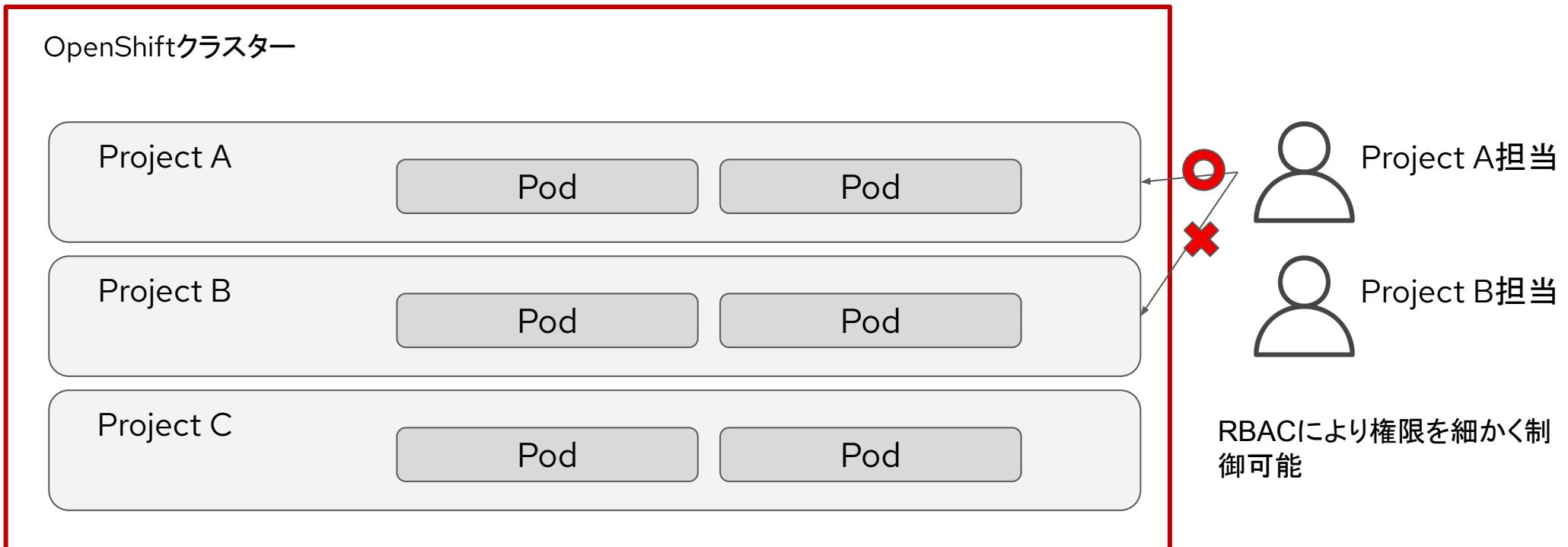
OpenShiftの主なリソース

リソース	概要
Project	Kubernetesのnamespaceとほぼ同じ。Project単位でリソースを分離。管理の観点からは、各Projectはテナントのように考えることができる
Pod	ホスト上にデプロイされた1つまたは複数のコンテナ。Podは、OpenShiftで定義、デプロイ、管理できるコンピュートリソースの最小の単位
BuildConfig	OpenShift 独自のAPIとして実装され、コンテナイメージをビルドするための定義を含む
ImageStream	OpenShift内部で管理されるコンテナレジストリのイメージのメタデータを参照
Deployment	Podを管理するための設定で、コンテナイメージ、Pod レプリカ数、CPU や Memory などのシステムリソースなどを定義する
Replicaset	Deploymentに定義した Podレプリカ数を保証し、Podが障害などで削除された場合、規定の台数に戻ることを保証する
Service	クラスタ内でのPodへのアクセスを管理。フロントとバックエンドのアプリケーションなど異なる種類のPod が通信する際に内部DNSを用いたホスト名を提供、リクエストを振り分ける
Route	クラスタ外部にアプリケーションを公開するためのリソース

Project

Project単位でリソースを分離します。管理の観点からは、各Projectはテナントのように考えることができます。

- Project単位でアクセス制御や権限制御が可能（マルチテナントの実現）
- Project単位でリソース制限（作成Pod数、CPU、メモリの上限等）
- Project内にPod等の各種リソースを展開



Webコンソールでのアプリケーションの作成

アプリケーションのデプロイのための様々な方式が準備されています。

The screenshot shows the Red Hat OpenShift Web Console interface. On the left, a dark sidebar lists navigation options: +追加 (Add), トポジー (Topology), モニタリング (Monitoring), 検索 (Search), Builds, Helm, プロジェクト (Project), ConfigMaps, シークレット (Secret), 開発者カタログ (Developer Catalog), すべてのサービス (All Services), データベース (Database), Operator 提供 (Operator Provider), Helm チャート (Helm Chart), and Virtual Machines. The main content area displays several cards for creating applications:

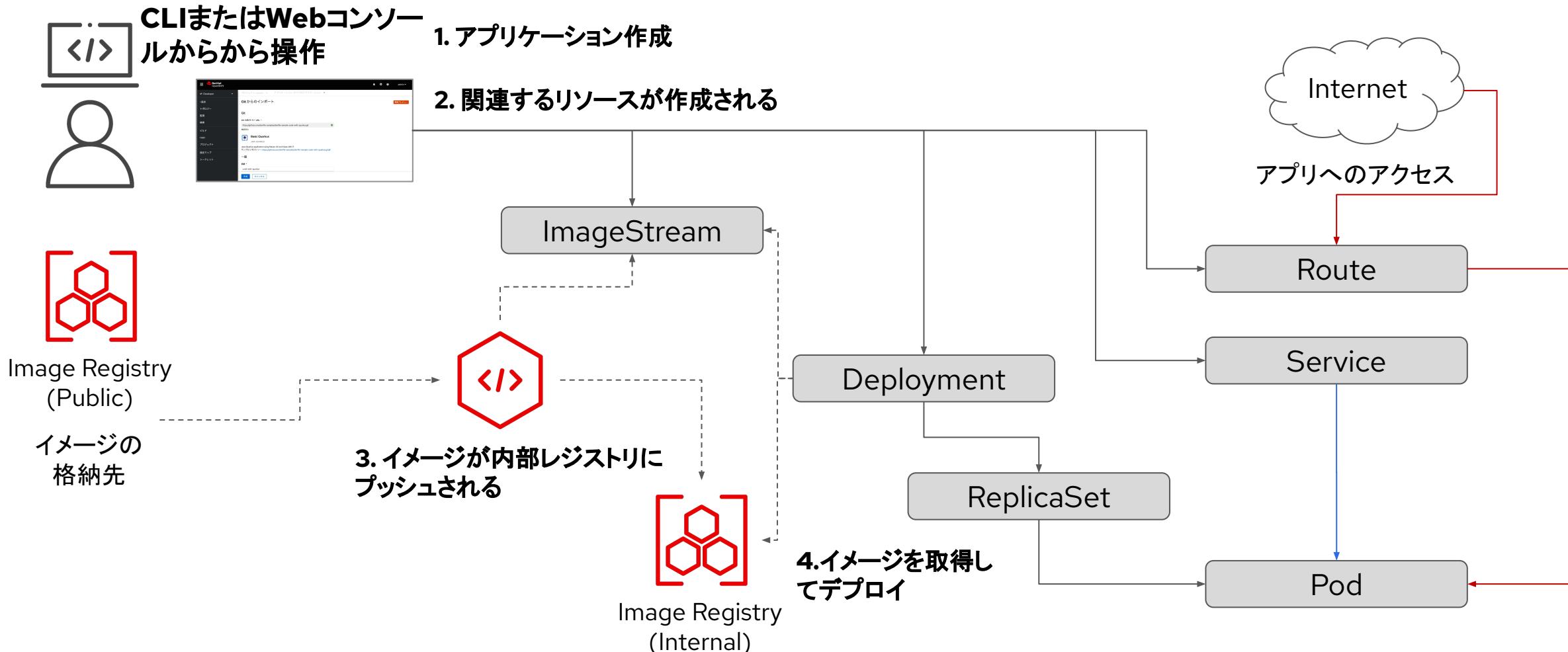
- Get started with Spring →** (Top right)
- すべてのサンプルを表示** (Top left)
- すべてのクイックスタートの表示** (Top center)
- OpenShift 4.15 の新機能** (Top right)
- Git リポジトリ**: Git からのインポート (Import from Git) - Build および デプロイ用に Git リポジトリからコードをインポートします.
- コンテナーイメージ**: コンテナーアイメージ (Container Image) - イメージレジストリまたはイメージストリームタグから既存のイメージをデプロイします.
- 共有**: 共有 (Share) - プロジェクトアクセスを使用すると、プロジェクトへのユーザーの追加や削除が可能になります.
- ローカルマシンから**: YAML のインポート (Import from YAML) - YAML または JSON 定義からリソースを作成します.
- サンプル**: サンプル (Sample) - コードサンプルからアプリケーションを作成します.
- Developer Catalog**: Helm チャートリポジトリ (Helm Chart Repository) - Helm チャートリポジトリを追加して Developer Catalog を拡張します.
- Deployment Methods** (Table):

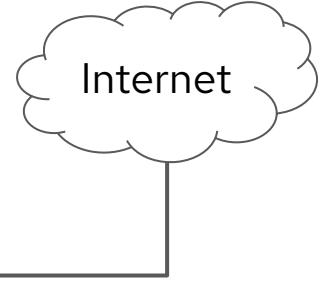
デプロイ方式	概要
開発者カタログ	アプリケーション実行のベースとなるコンテナのイメージとテンプレートが含まれています
Gitからのインポート	Gitリポジトリの既存のコードベースをインポートし、OpenShift Container Platform でアプリケーションを作成し、ビルドし、デプロイします
コンテナイイメージ	レジストリまたはimageStreamから既存イメージを使用し、これをOpenShift Container Platformにデプロイします
YAMLのインポート	エディターを使用してYAML または JSON 定義を追加し、リソースを作成し、変更します
Helmチャートリポジトリ	Helmチャートリポジトリを追加してDeveloper Catalogを拡張します

At the bottom, a footer bar contains the text "Copyright © 2024 Red Hat K.K. All Rights Reserved." and the page number "24".

アプリケーションデプロイとリソース

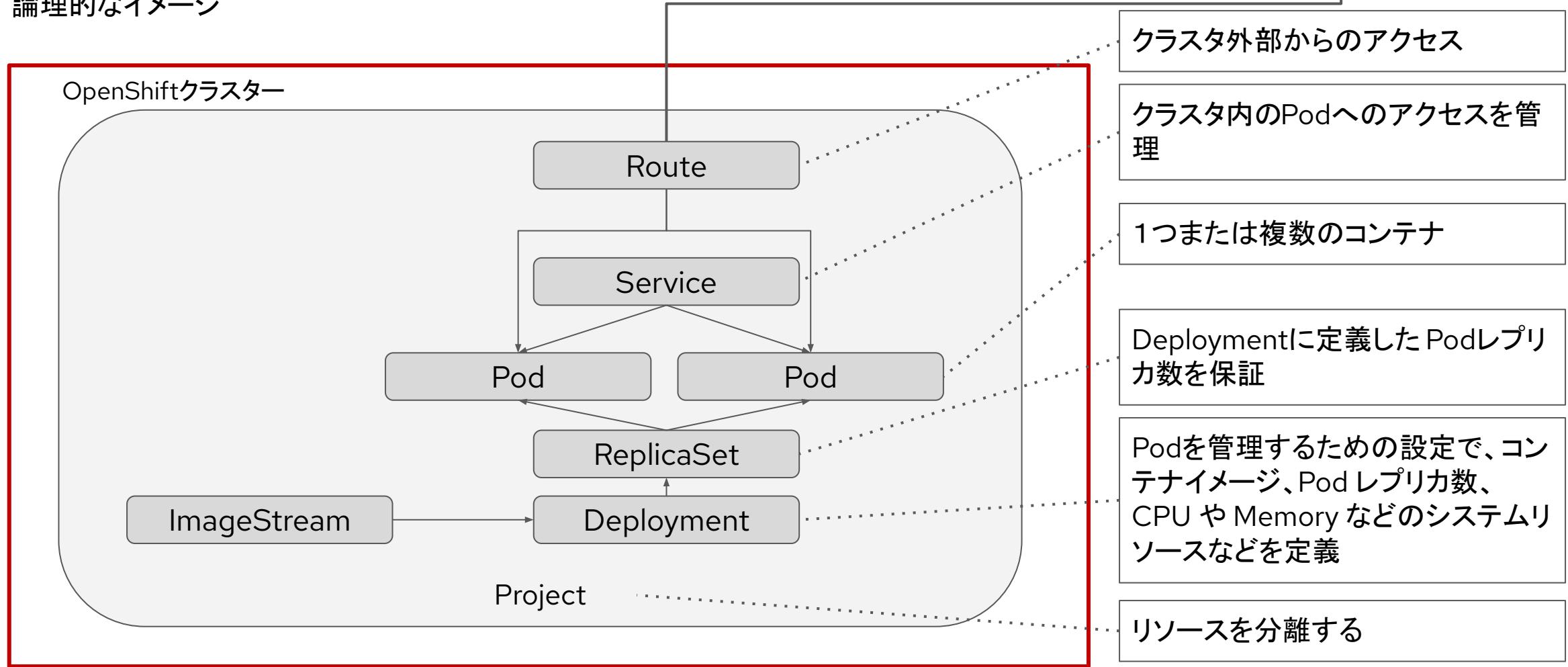
CLIまたはWebコンソールからアプリケーションをデプロイすると各種リソースが作成されます



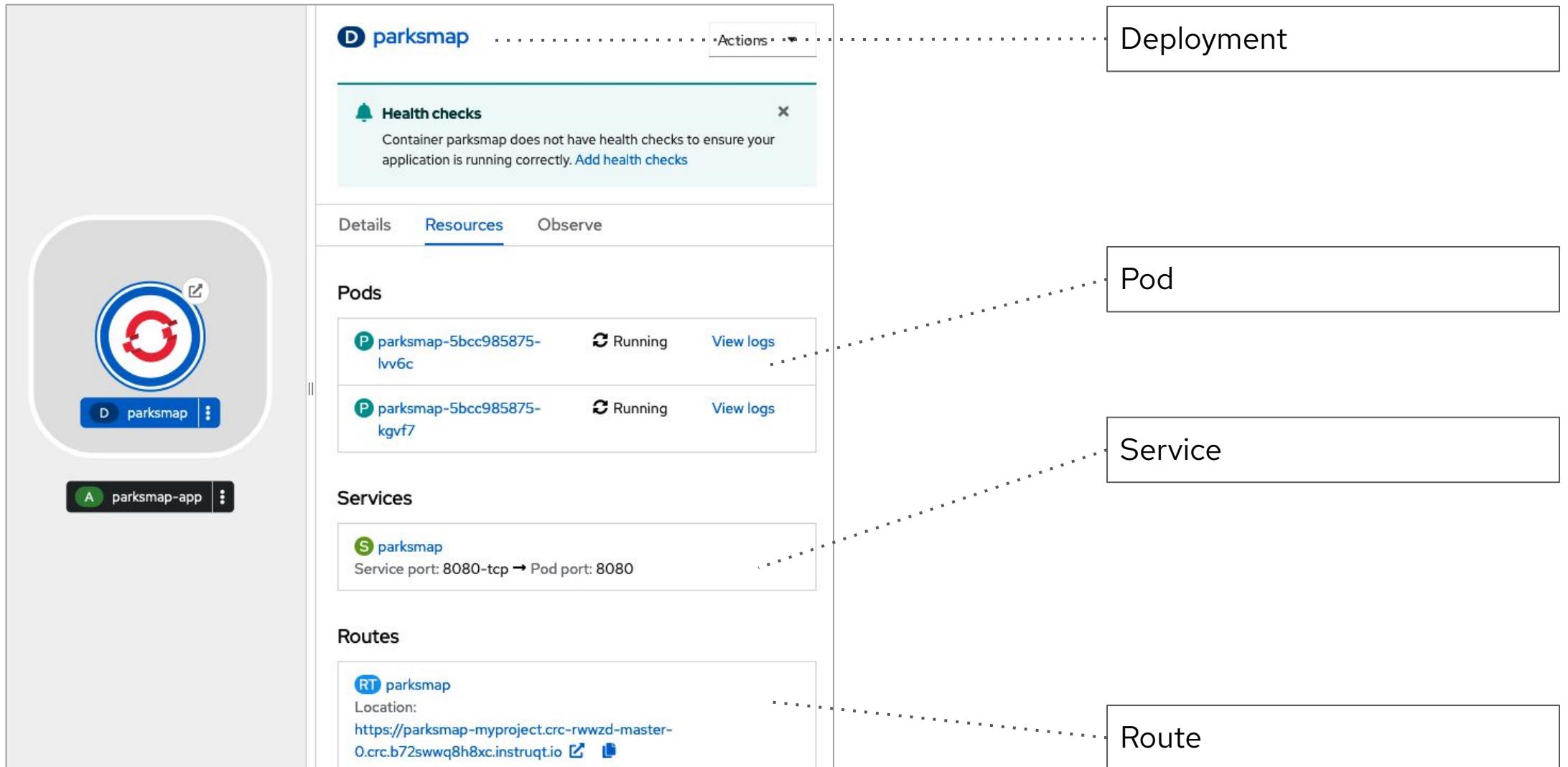


アプリケーションデプロイとリソース

論理的なイメージ



トポロジービューにおけるリソース確認



DeploymentとReplicaSet

Project: myproject ▾

Deployments > Deployment details

D parksmap

Details Metrics YAML ReplicaSets Pods Environment Events

ReplicaSets

Deployment details

2 Pods

Name: parksmap

Namespace: NS myproject

Labels:

- app=parksmap
- app.kubernetes.io/component=parksmap
- app.kubernetes.io/instance=parksmap
- app.kubernetes.io/name=parksmap
- app.kubernetes.io/part-of=parksmap-app
- app.openshift.io/runtime-namespace=myproject

Pod selector: Q app=parksmap

Update strategy: RollingUpdate

Max unavailable: 25% of 2 pods

Max surge: 25% greater than 2 pods

Progress deadline seconds: 600 seconds

Min ready seconds: Not configured

Project: myproject ▾

ReplicaSets > ReplicaSet details

RS parksmap-5bcc985875

Details YAML Pods Environment Events

ReplicaSet details

Name: parksmap-5bcc985875

Namespace: NS myproject

Labels:

- app=parksmap
- deployment=parksmap
- pod-template-hash=5bcc985875

Pod selector: Q app=parksmap, pod-template-hash=5bcc985875

Node selector: No selector

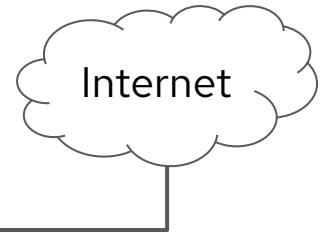
Tolerations: 0 tolerations

Annotations:

現在のPodの数 → Current count: 2

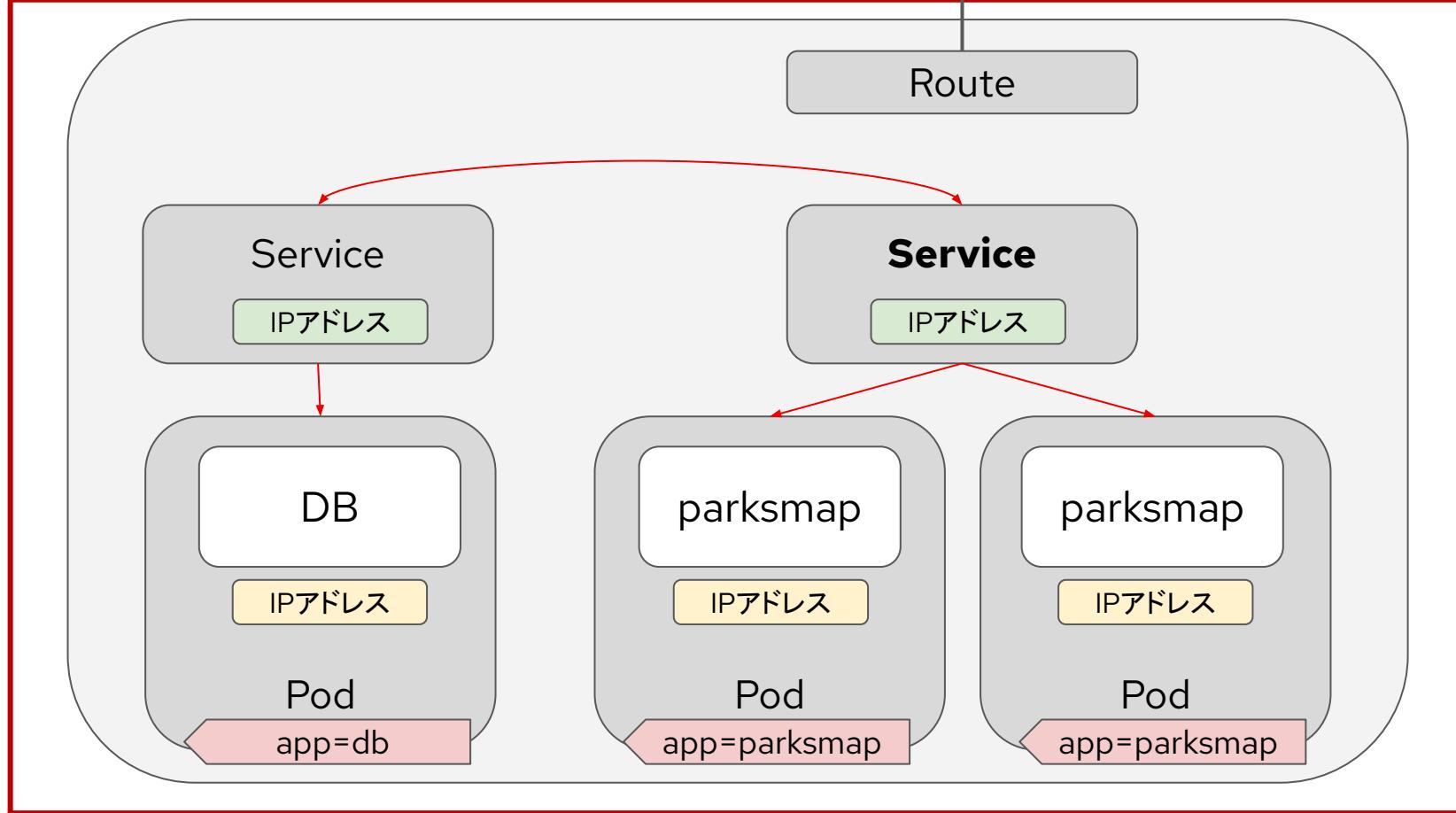
有るべきのPodの数 → Desired count: 2

PodDisruptionBudgets: No PodDisruptionBudgets



Service

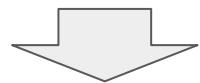
クラスタ内でのPodへのアクセスを管理



例) クラスタ内でDB Podと
parksmap Podで通信する場合

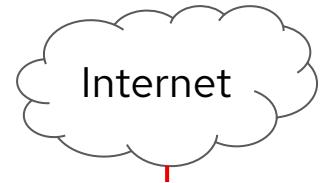
- ServiceのIPは一度作成されると変化しない

- PodのIPアドレスは削除や追加などで変化する



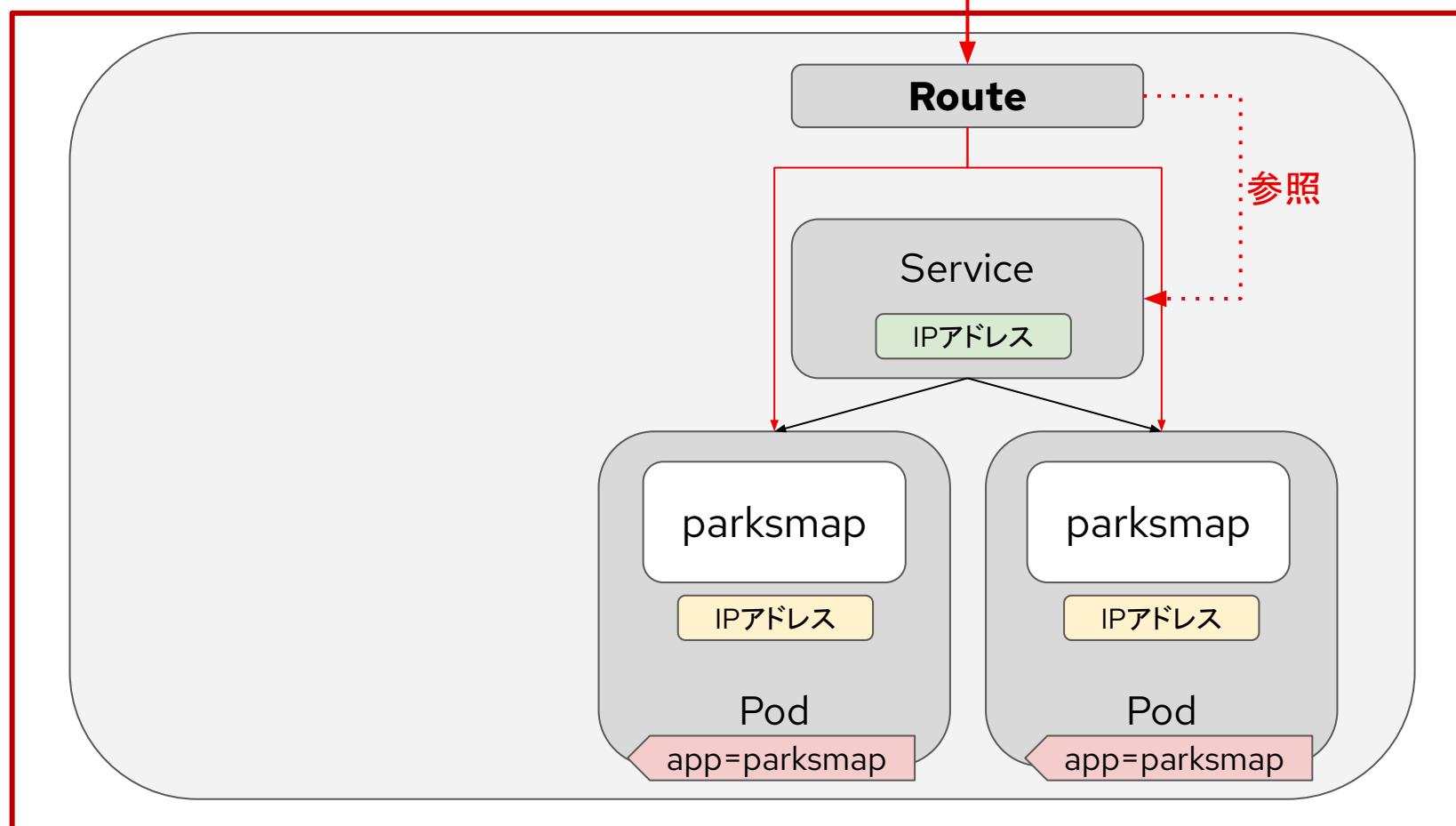
- ServiceのIP(またはService名)に対して通信を行うと、ServiceがPodに割り振る

- Serviceは管理すべきPodをLabelを見て判断している



Route

クラスタ外部にアプリケーションを公開する



例) クラスタ外からWEBにアクセスする場合

- ・URLを作成
- ・Serviceの内容を参照して、割り振るべきPodの情報を取得する
- ・実態はhaproxyでL7ロードバランサーでPodに振り分ける

参考: Deployment (yaml表記)

```
kind: Deployment
apiVersion: apps/v1
metadata:
  annotations:
    alpha.image.policy.openshift.io/resolve-names: '*'
    app.openshift.io/route-disabled: 'false'
    deployment.kubernetes.io/revision: '1'
    image.openshift.io/triggers: '>'

[{"from":{"kind":"ImageStreamTag","name":"parksmap:latest","namespace":"myproject"},"fieldPath":"spec.template.spec.containers[?(@.name==\"parksmap\")].image","pause":"false"}]
  openshift.io/generated-by: OpenShiftWebConsole
resourceVersion: '45090'
name: parksmap
uid: f1eabdd6-89be-47c3-957d-957dc7233380
creationTimestamp: '2023-11-17T03:21:18Z'
generation: 2
(略)
namespace: myproject
labels:
  app: parksmap
  app.kubernetes.io/component: parksmap
  app.kubernetes.io/instance: parksmap
  app.kubernetes.io/name: parksmap
  app.kubernetes.io/part-of: parksmap-app
  app.openshift.io/runtime-namespace: myproject
spec:
  replicas: 2
  selector:
    matchLabels:
      app: parksmap
```

デプロイに関する情報
報

レプリカ数

Podの情報

```
template:
  metadata:
    creationTimestamp: null
    labels:
      app: parksmap
      deployment: parksmap
    annotations:
      openshift.io/generated-by: OpenShiftWebConsole
  spec:
    containers:
      - name: parksmap
        image: >

image-registry.openshift-image-registry.svc:5000/myproject/parksmap@sha256:89d1e324846cb431df9039e1a7fd0ed2ba0c51aafbae73f2abd70a83d5fa173b
  ports:
    - containerPort: 8080
      protocol: TCP
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      imagePullPolicy: Always
      restartPolicy: Always
      terminationGracePeriodSeconds: 30
      dnsPolicy: ClusterFirst
      securityContext: {}
      schedulerName: default-scheduler
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 25%
    revisionHistoryLimit: 10
    progressDeadlineSeconds: 600
  status:
    observedGeneration: 2
    replicas: 2
    updatedReplicas: 2
    readyReplicas: 2
```

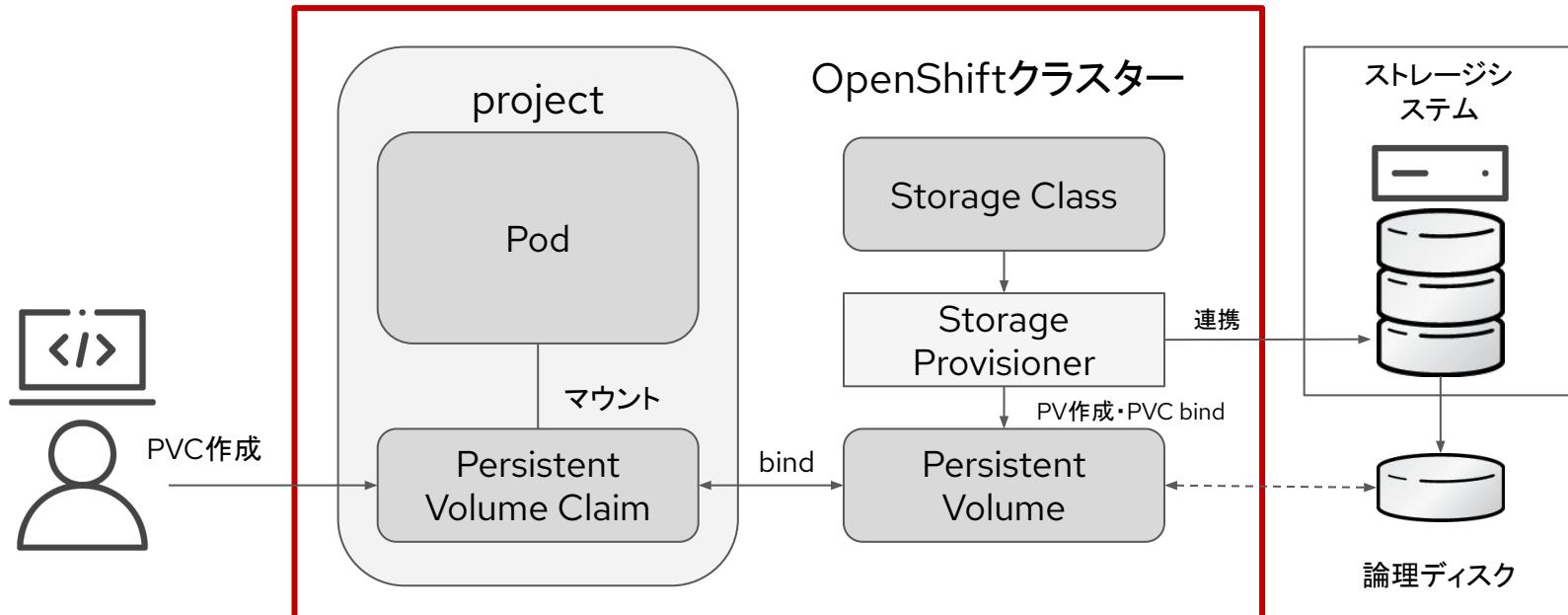
参考：マニフェストの記述

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: parksmap
  namespace: myproject
  labels:
spec:
  replicas: 2
  selector:
    matchLabels:
      app: parksmap
template:
  metadata:
  spec:
    containers:
    - name: parksmap
      image: >-
        image-registry.openshift-image-registry.svc:5000/myproject/parksmap@sha256:89d1e324846cb431df9039e1a7fd0ed2ba0c51aafbae73f2abd70a83d5fa173b
    ports:
    - containerPort: 8080
      protocol: TCP
      resources: {}
status:
  observedGeneration: 2
  replicas: 2
  updatedReplicas: 2
  readyReplicas: 2
```

- kind
 - 作成するリソースの種類
- apiVersion
 - 利用するkubernetesAPIバージョン
- metadata
 - リソースにつける名前など、オブジェクトを一意に特定するための情報
- spec
 - 作成するリソースの仕様
 - オブジェクトの望ましい状態
 - Template
 - Podに関する情報、利用するイメージやポート等
- status
 - 現在の状態

永続ボリューム(永続ストレージ)

- Persistent Volume(永続ボリューム、永続ストレージ):ストレージシステム上の論理ディスクを抽象化したもの
- Persistent Volume Claim:Persistent Volumeを使用するために、割当を要求するリソース
- Storage Class:バックエンドのストレージを抽象化するもの。あらかじめ管理者が作成しておく
- Storage Provisioner:バックエンドのストレージシステムと連携して論理ディスクの作成や削除を行う





ハンズオン

演習内容

イメージを利用したアプリケーションのデプロイ	WebコンソールおよびCLIを利用してコンテナレジストリにあるイメージからOpenShiftクラスタにアプリケーションをデプロイ、公開する方法を学びます。またアプリケーションのスケールアップが容易に行えることを体験します。
ソースコードを利用したアプリケーションのデプロイ	WebコンソールおよびCLIを利用してGithubにあるソースコードからOpenShiftクラスターにアプリケーションをデプロイ、公開する方法を学び、OpenShiftでのビルトについて理解を深めます。
OpenShiftにおける永続ボリュームの利用	永続ボリューム(Persistent Volume)の操作と動作確認を実施し、使用方法や関連する概念について学びます。
OpenShift Pipelines	クラスタにOpenShift Pipelinesを導入しTaskおよびPipelineを作成します。作成したPipelineを利用してアプリケーションをデプロイします。演習を通じて、Pipelineを利用することによりアプリケーションのデプロイ等を自動化できることを学びます。
OpenShift GitOps	クラスタにOpenShift GitOpsを導入しGitリポジトリにあるマニフェストを利用してアプリケーションをデプロイします。演習を通じて、GitリポジトリのコードをSSoTと考えてコードと環境と同じ状態に保つArgoCDの動作を体験します。
Podman 入門	次世代コンテナランタイムPodmanを学習します。書籍「Podmanイン・アクション」のコマンド例を元に、多数のコマンドを叩きながらコンテナランタイム、Podmanの理解を深めます。

使用する環境

ブラウザのみでOpenShiftを体験できるハンズオン環境です

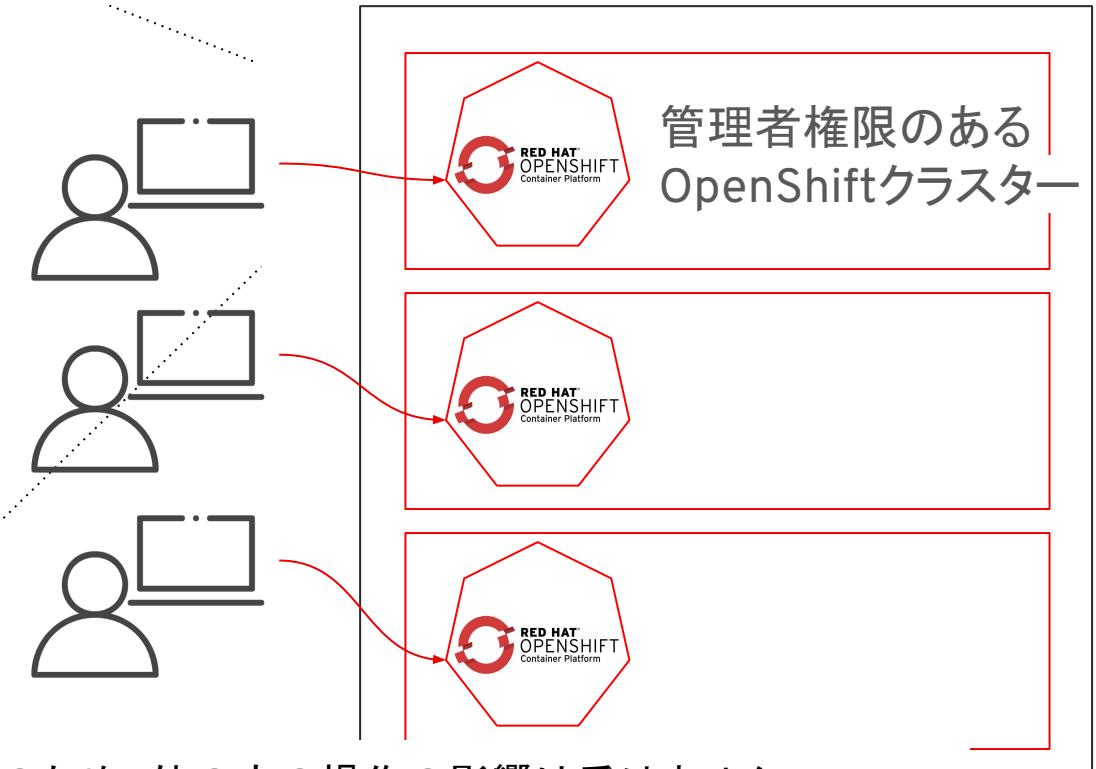
レプリカの数を確認する

Step 3: レプリカ数の変更の確認をするためにはサイドバーアルにあるResourcesタブをクリックします。次の図に示すように、Podのリストが表示されます。

OpenShift Local cluster is for development and testing purposes. DON'T use it for production.

起動すると、演習内容と演習に必要な
OpenShift環境が準備されます

instruqt



専有環境のため、他の方の操作の影響は受けません

ハンズオン環境の操作方法

右側のテキストを確認しながら、左側のコンソールやGUIで操作してください

STEP 1 環境のローンチ

最初の画面の右下にある
[Launch]ボタンを押します。



STEP 2 ハンズオンの開始

以降の画面の右下にある[START]ボタンを
押すとハンズオン環境が起動します。



STEP 3 テキストを見ながら演習

右がテキスト、左が操作画面です。トピック毎にハ
ンズオンが完了したら[Next]を押します。

右がテキスト、左が操作画面です。トピック毎にハ
ンズオンが完了したら[Next]を押します。

レプリカの数を確認する

Step 3: レプリカ数の変更の確認をするためにはサイドバネ
ルにある Resources タブをクリックします。次の図に示すよ
うに、Pod のリストが表示されます。

2つのレプリカがあることがわかります。それでは、

Exit Skip Next

Next

ハンズオンコース紹介

アプリケーションデプロイ

1

イメージを利用した アプリケーションのデプロイ

演習内容

イメージを利用したアプリケーションのデプロイ

WebコンソールおよびCLIを利用してコンテナレジストリにあるイメージからOpenShiftクラスタにアプリケーションをデプロイ、公開する方法を学びます。またアプリケーションのスケールアップが容易に行えることを体験します。



ゴール

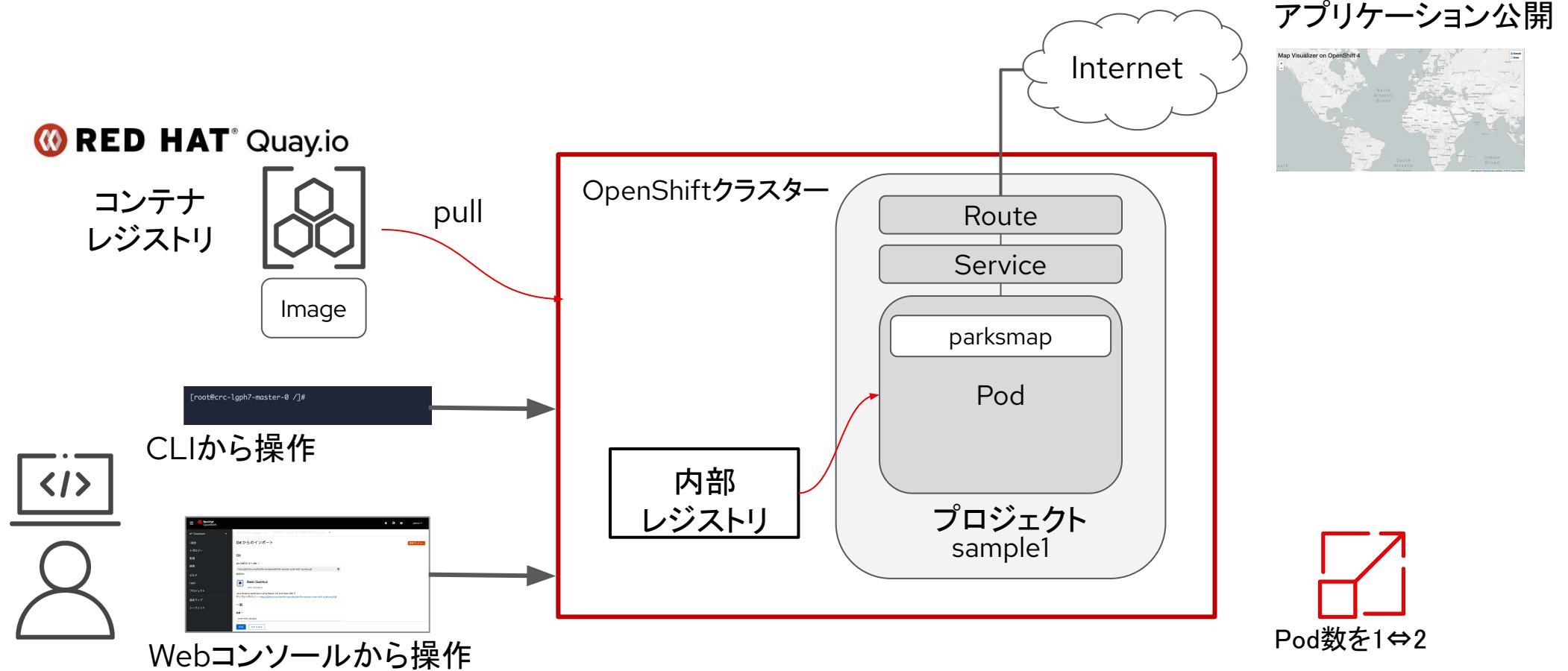
- OpenShiftのCLIおよびGUIの基本的な使用方法を学習します
- GUIおよびCLIを利用してイメージからアプリケーションをデプロイする方法を学習します
- アプリケーションを公開する方法を学習します
- アプリケーションをスケールアップする方法を学習します
- OpenShiftの主なリソースを理解します

イメージを利用したアプリケーションのデプロイ

演習内容

- CLIおよびWebコンソールの基本操作を学びます
- Webコンソールを利用してQuay.ioにあるイメージを利用してアプリケーションをデプロイします
- WebコンソールのTopologyビューを利用してリソースを確認します
- アプリケーションをスケールアップします
- CLIを利用してQuay.ioにあるイメージを利用してアプリケーションをデプロイします
- CLIを利用してアプリケーションを公開およびスケールアップ・ダウントを実施します
- CLIおよびWebコンソールを利用してアプリケーションの削除を実施します

イメージを利用したアプリケーションのデプロイ



2

ソースコードを利用した アプリケーションのデプロイ

演習内容

ソースコードを利用したアプリケーションのデプロイ

WebコンソールおよびCLIを利用してGithubにあるソースコードからOpenShiftクラスターにアプリケーションをデプロイ、公開する方法を学び、OpenShiftでのビルトについて理解を深めます。



ゴール

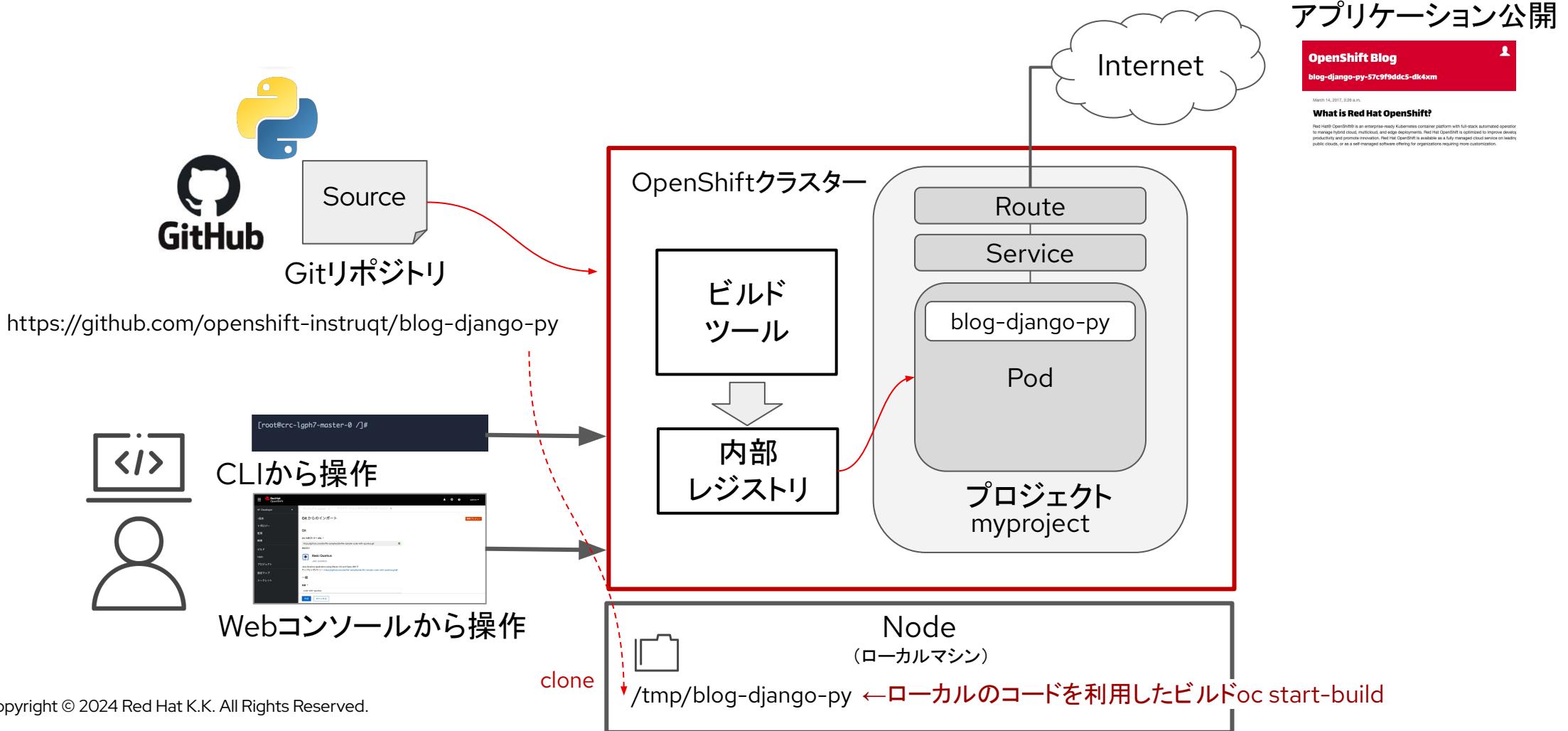
- OpenShiftのCLIおよびGUIの基本的な使用方法を学習します
- GUIおよびCLIを利用してソースコードからアプリケーションをデプロイする方法を学習します
- OpenShiftのSource to Image (S2I) を学習します

ソースコードを利用したアプリケーションのデプロイ

演習内容

- CLIを利用してプロジェクトを作成します
- Webコンソールを利用してGithubにあるソースコードを利用してアプリケーションをデプロイします
- ビルドおよびデプロイのログの確認を行います
- ラベルを利用してアプリケーションを削除します
- CLIを利用してアプリケーションを再デプロイします
- ソースコードのコピーに変更を加え、アプリケーションを再ビルトします

ソースコードを利用したアプリケーションのデプロイ



Source to Imageビルド

アプリケーションソースコードとベースイメージを動的にビルドする機能(s2i)



Developers



Red Hatから提供されるベースイメージ

Project: myproject Application: All applications

Create Source-to-Image application

Builder Image version *

IST 3.9-ubi8

Python 3.9 (UBI 8)
BUILDER PYTHON

Build and run Python 3.9 applications on UBI 8. For more information about using this builder image, including OpenShift considerations, see <https://github.com/sclorg/s2i-python-container/blob/master/3.9/README.md>. Sample repository: <https://github.com/sclorg/django-ex.git>

Git

Git Repo URL *

✓

Validated

Try sample ↗

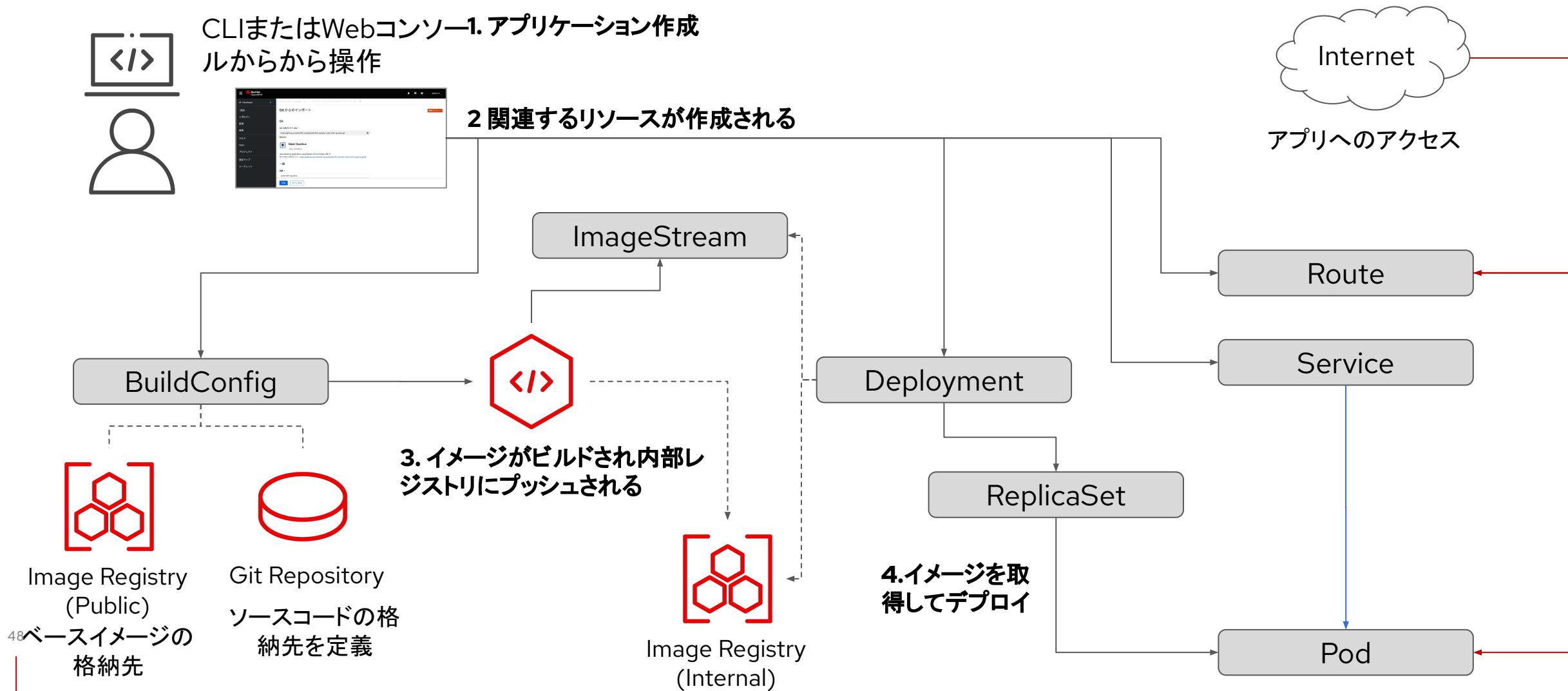
› Show advanced Git options

Create Cancel

1. アプリケーションのベースイメージを指定

2. アプリケーションソースコードが入ったリポジトリを指定

アプリケーションデプロイとリソース





OpenShiftにおける永続ボリュームの利 用

演習内容

OpenShiftにおける永続ボリュームの利用

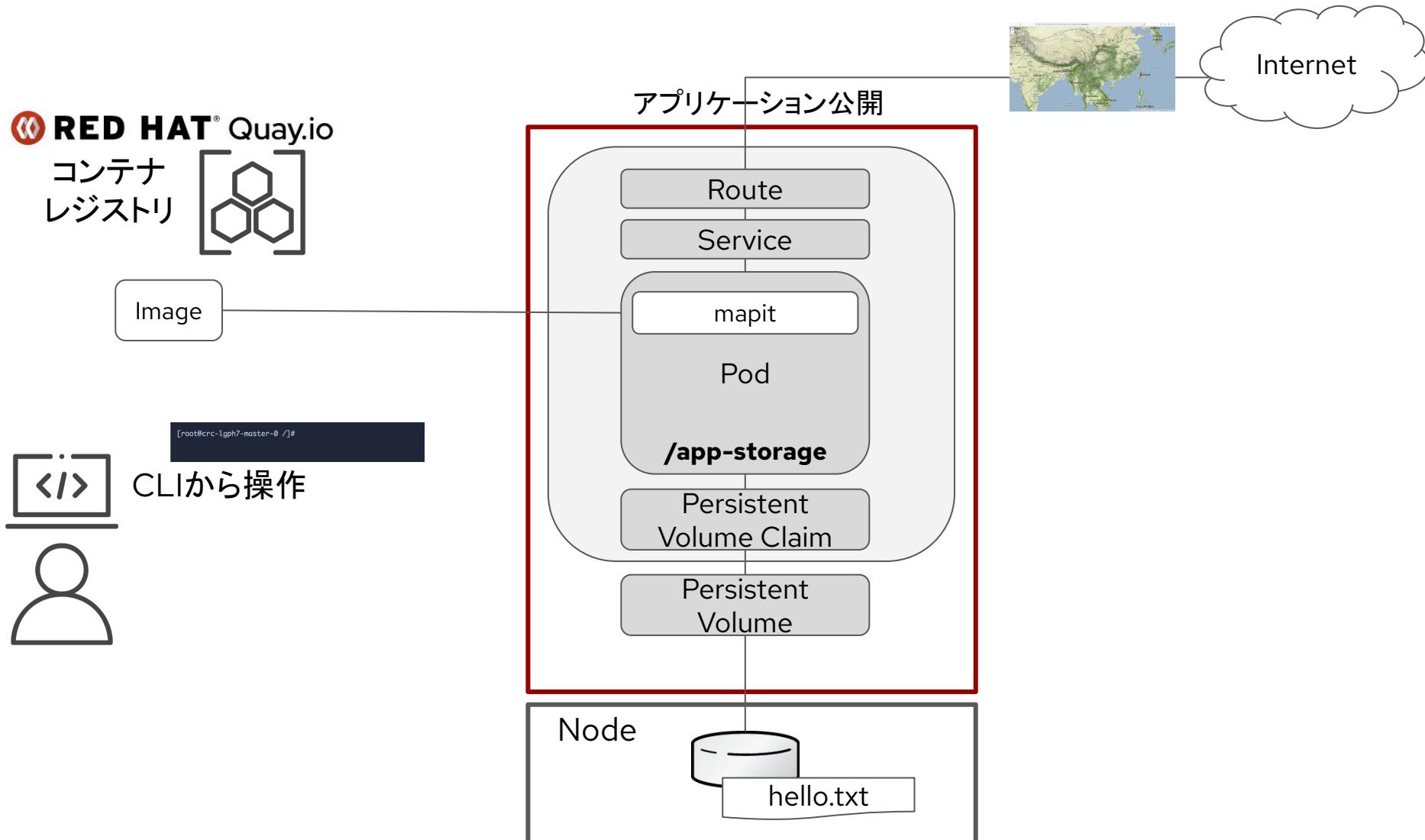
永続ボリューム(Persistent Volume)の操作と動作確認を実施し、使用方法や関連する概念について学びます。



ゴール

- Persistent Volume ClaimとPersistent Volumeについて理解します
- 永続ボリュームの利用方法を確認します
- 永続ボリュームの動作を確認します

OpenShiftにおける永続ボリュームの利用



OpenShiftにおける永続ボリュームの利用

演習内容

- サンプルアプリケーションをデプロイして公開します
- Persistent Volume Claimと永続ボリューム(Persistent Volume)についての説明
- アプリケーションが永続ボリュームを利用するよう設定を行います
- 永続ボリュームがPodから独立して存在することを確認します

ハンズオンコース紹介

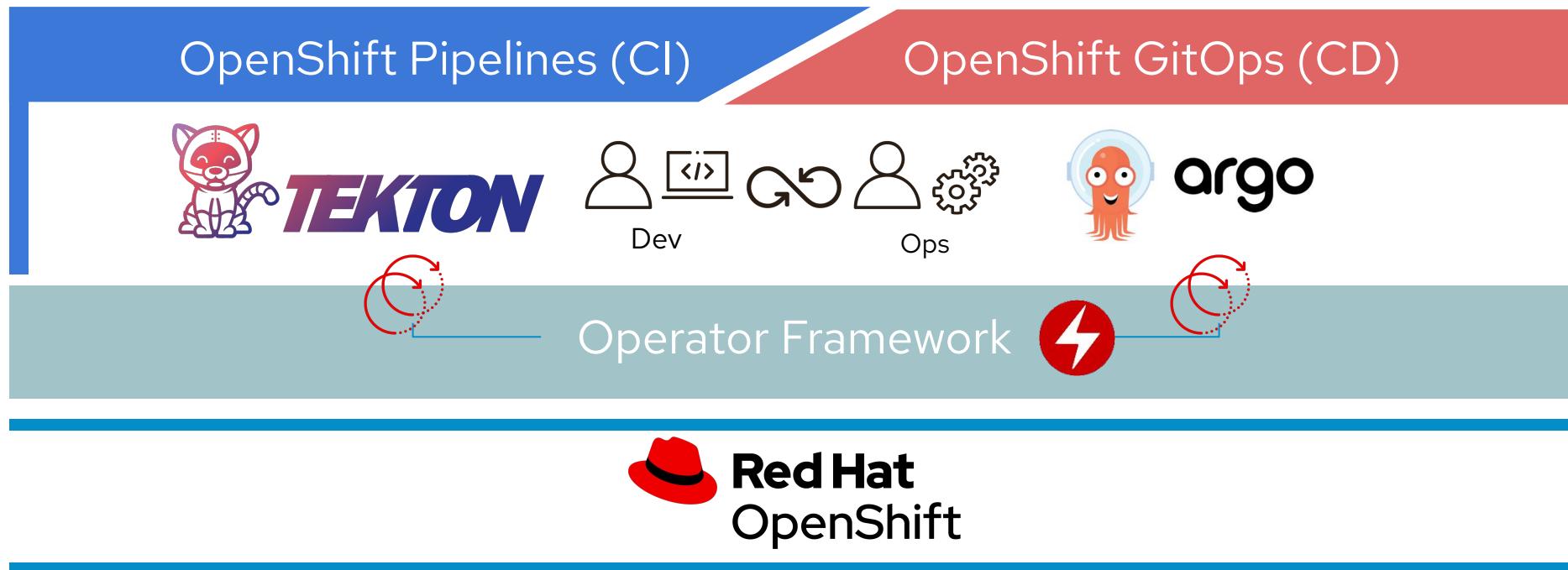
OpenShift Pipelines

OpenShift GitOps

OpenShift Pipelines / OpenShift GitOps

OpenShiftにおけるCI/CD

OpenShiftではCIをOSSのTektonをベースとするOpenShift Pipelinesで、CDをArgo CDをベースとしたOpenShift GitOpsにて実施します。それぞれOperatorを使いインストール/管理されます。



サーバーレスでのパイプライン実行

オンデマンドで必要なコンテナのみ起動

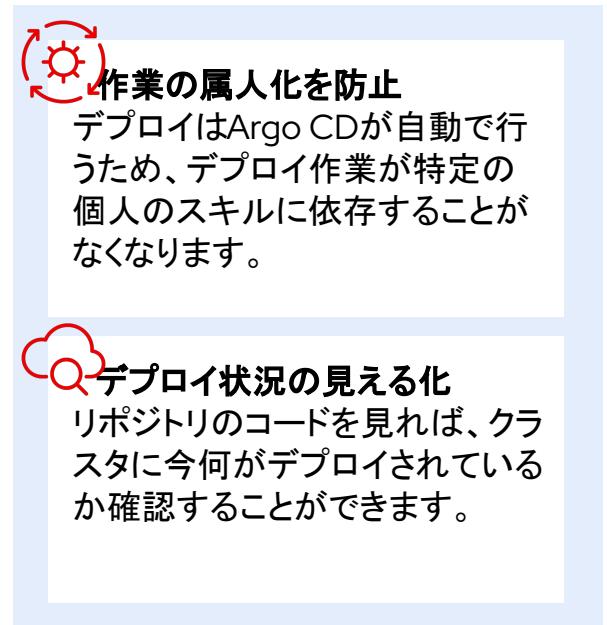
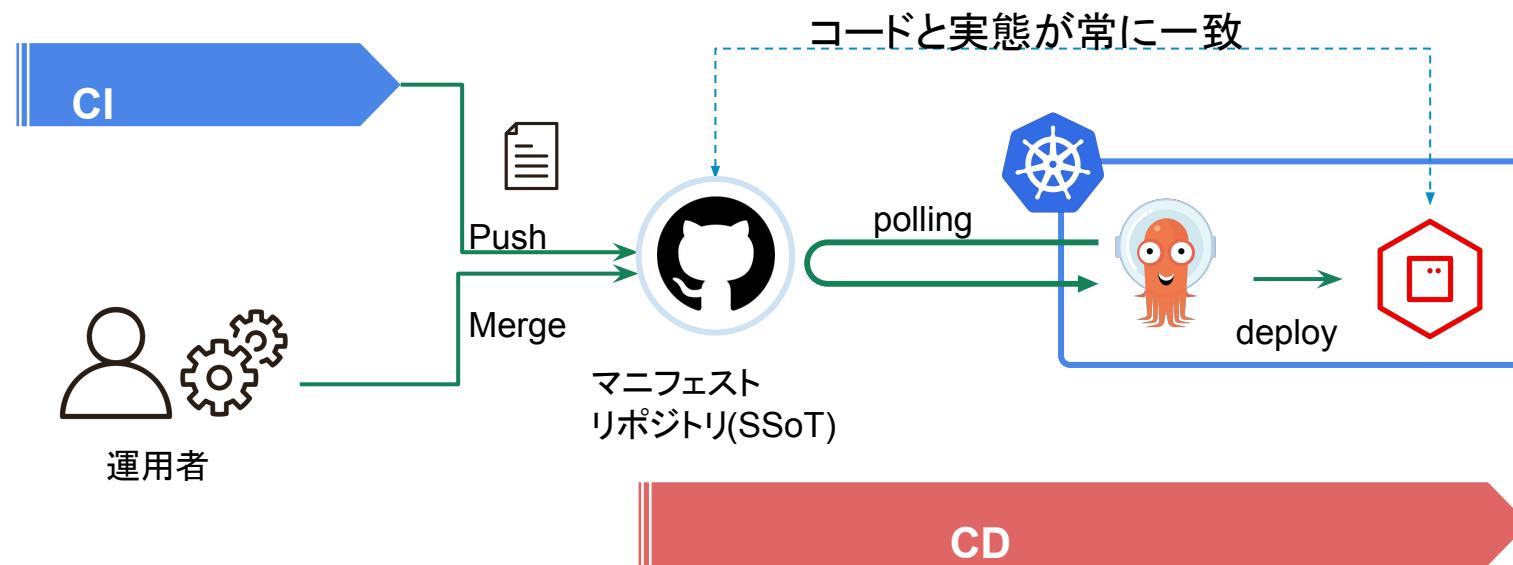
TektonのCIパイプラインはコンテナとして実行されます。開発者のアクション(Gitリポジトリへの操作)を起点とし、必要なタイミングでコンテナを立ち上げるため、クラスタのリソースを有効活用できます。



コードによるインフラの管理

GitOpsによる運用を実現

Gitリポジトリに置かれたコードをSingle Source of Truthとみなし、コードとインフラを常に同じ状態に保つ運用のベストプラクティスをGitOpsと呼びます。Argo CDはGitリポジトリの状態を定期的に確認し、変更を自動的にデプロイすることでGitOpsを実現します。



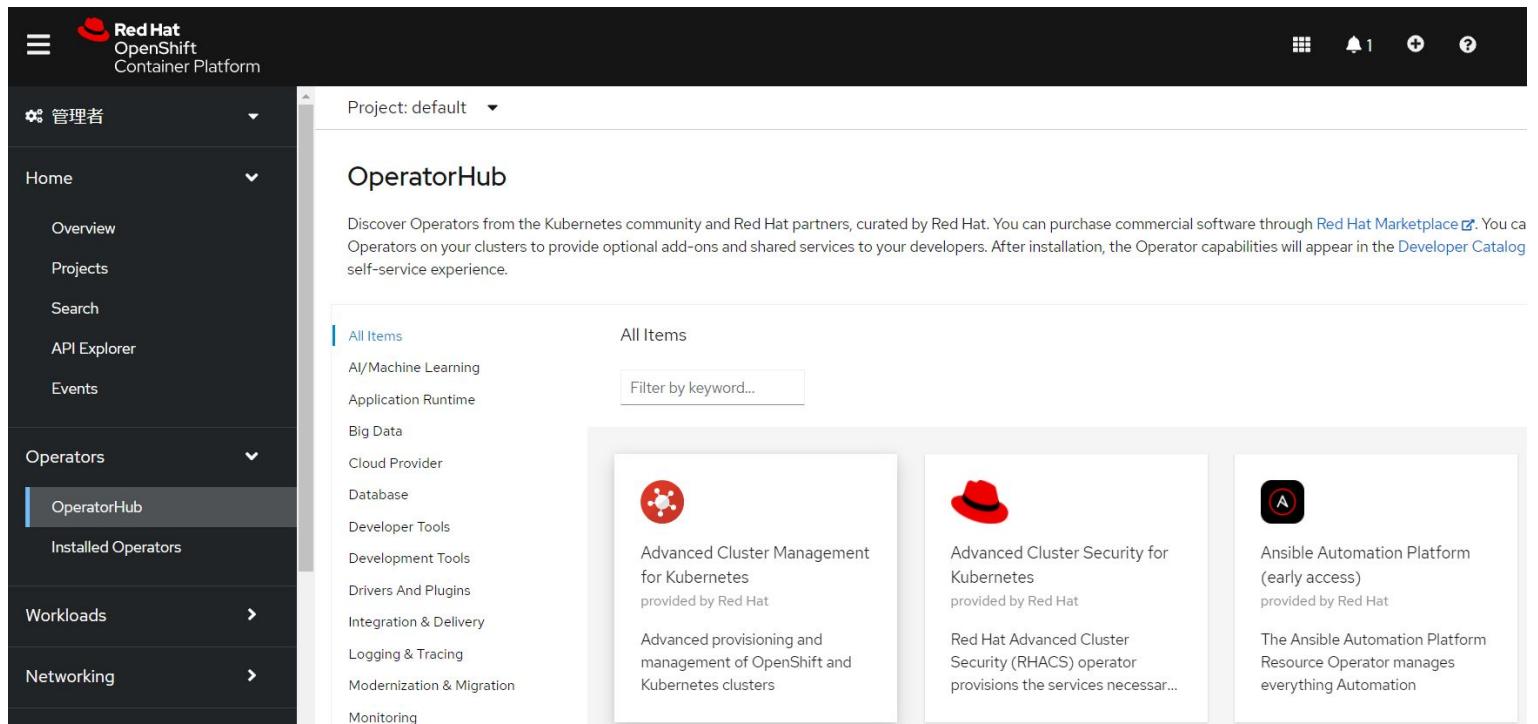
OperatorHub

OperatorHubによって各種ミドルウェアが提供されます。

Operatorを利用することによって、各ミドルウェアのインストール、設定、監視、アップグレードなどの運用操作を自動化できます。

OpenShiftのサブスクリプションに含まれる
ミドルウェア

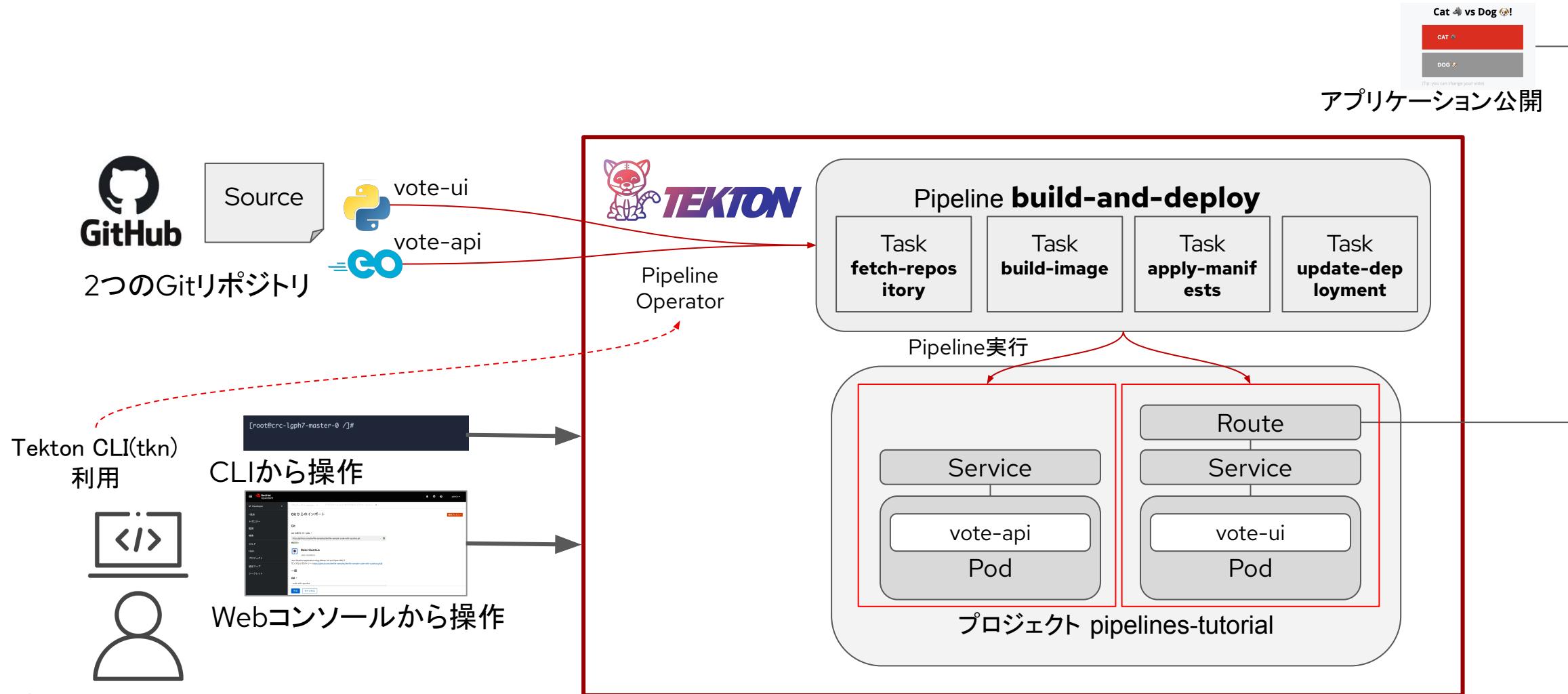
- CI Pipelines (Tekton)
- GitOps (Argo)
- Cluster Monitoring (Prometheus)
- Cluster Logging (Loki + Vector)
- Service Mesh (Istio)
- Serverless (Knative)
- Tracing (Jaeger / Kiali)
- API Security (Gatekeeper)
- etc



3

OpenShift Pipeline

OpenShift Pipelines



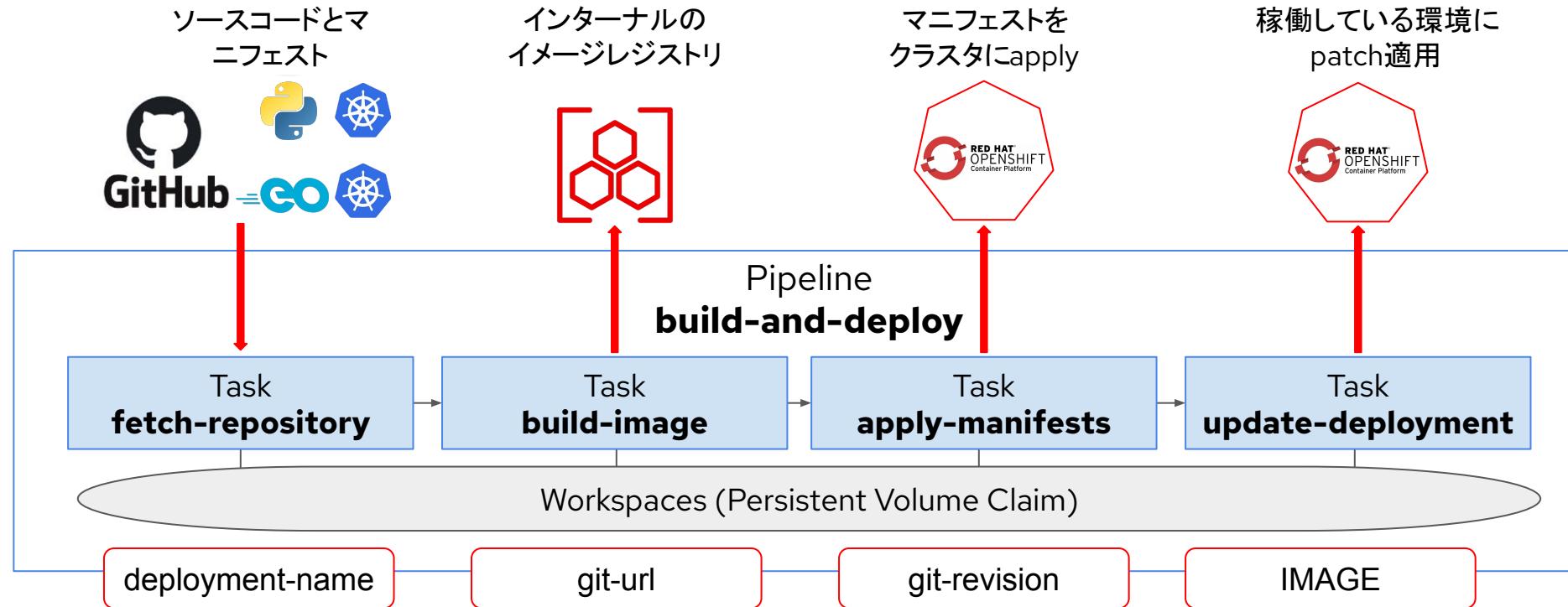
OpenShift Pipelines

演習内容

- Pipelines Operatorをインストールします
- 簡単なサンプルTaskを作成し実行します
- マニフェストをapplyするTask、deploymentを更新するTask、PersistentVolumeClaimを作成します
- 作成したTaskとClusterTaskを組み合わせて、Gitクローン、imageのビルド、マニフェストのapply、deploymentの更新を実行するPipelineを作成します
- 手動でPipelineを実行して投票アプリケーションをデプロイします

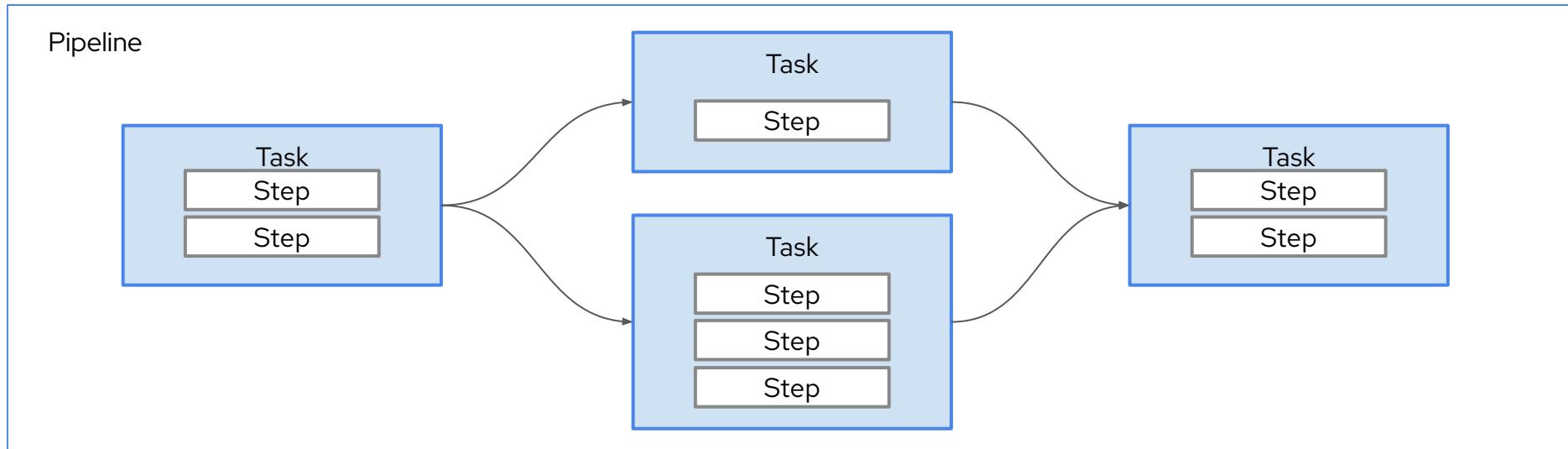
作成するPipeline

適切なパラメータを指定することにより、GitHubにあるソースコードを取得しイメージをビルドしてインターナルのレジストリにPushします。そのイメージを利用してアプリケーションをデプロイします。



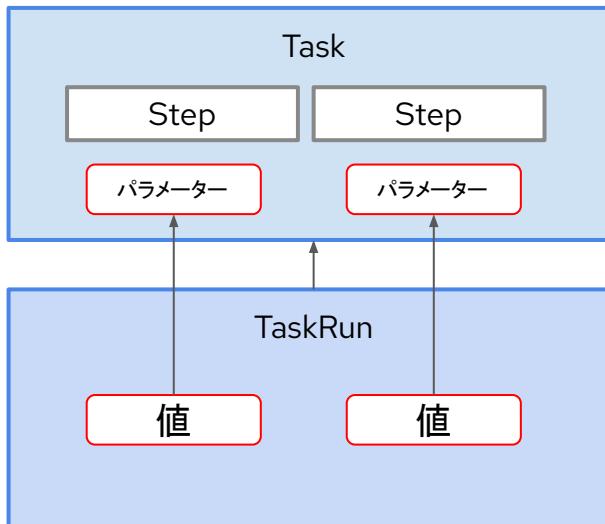
Tektonのリソース

リソース	概要
Task	ひとつ以上のStepから構成されます。Taskの実行内容と実行に必要なパラメーターなどを定義できます。(Stepはひとつのコンテナイメージを利用して処理を実行します)
TaskRun	定義したTaskを実行するために利用します。Taskの実行に必要な、入力や出力、パラメーターなどを指定します。TaskRun単体として利用することができますが、Pipelineを実行すると自動的にTaskRunが作成されます。
Pipeline	定義したTaskの集合体で、一連の処理の流れを定義したもの。Pipelineを実行するのに必要なパラメーターなども定義できます。
PipelineRun	定義したPipelineを実行するために利用します。Pipelineの実行に必要な、入力や出力、パラメーターを指定できます。



TaskRunとPipelineRun

TaskRun

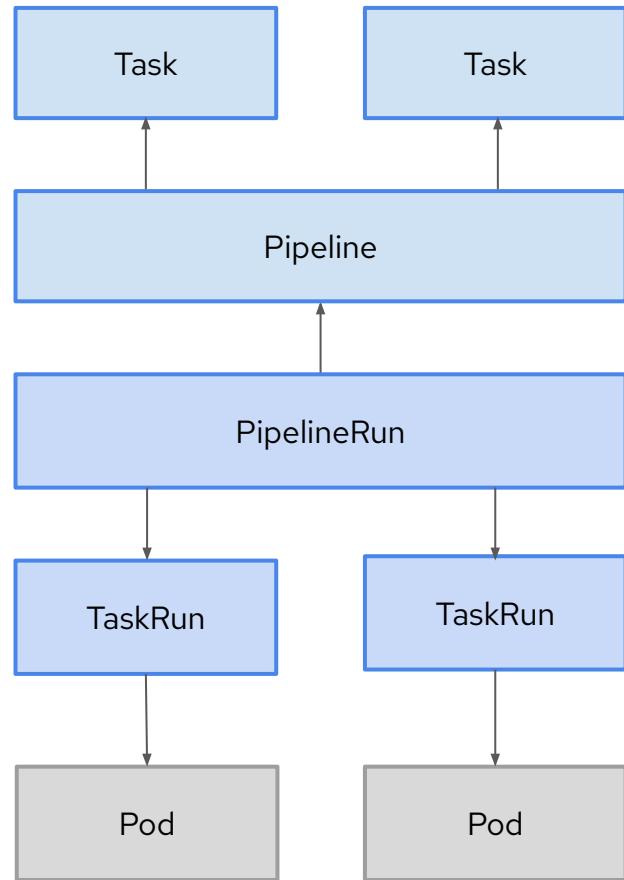


TaskRunによってTaskはPodとして動作し、処理が終わると終了する

TaskRun

- Pod内でTaskを完了まで実行
- Task仕様の参照または埋め込み
- Taskへの入力を提供
 - パラメーター
 - リソース
 - サービスアカウント
 - workspace

PipelineRun



PipelineRun

- Pipelineを完了まで実行
- Pipeline仕様の参照または埋め込み
- PipelineのTaskを実行するTaskRunを作成
- Pipelineへの入力とパラメーターの提供
- Pipelineのworkspaceにボリュームを提供

Task定義

apply_manifest_task.yaml

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: apply-manifests
spec:
  workspaces:
    - name: source
  params:
    - name: manifest_dir
      description: The directory in source that contains yaml
      manifests
      type: string
      default: "k8s"
  steps:
    - name: apply
      image: quay.io/openshift/origin-cli:latest
      workingDir: /workspace/source
      command: ["/bin/bash", "-c"]
      args:
        - |
          echo Applying manifests in
          $(inputs.params.manifest_dir) directory
          oc apply -f $(inputs.params.manifest_dir)
          echo -----
```

演習で使用するyamlは準備済です

- apply-manifestsという名前のTask
- Params
 - manifest_dir(マニフェストのあるディレクトリ)
- paramsで入力されたマニフェストをoc applyコマンドで環境に適用する



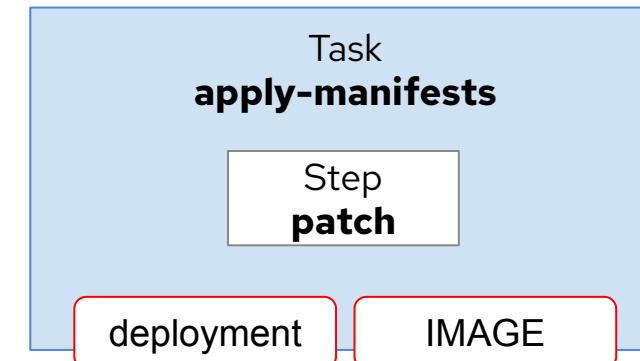
Task定義

`update_deployment_task.yaml`

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: update-deployment
spec:
  params:
    - name: deployment
      description: The name of the deployment patch the
      image
      type: string
    - name: IMAGE
      description: Location of image to be patched with
      type: string
  steps:
    - name: patch
      image: quay.io/openshift/origin-cli:latest
      command: ["/bin/bash", "-c"]
      args:
        - |
          oc patch deployment $(inputs.params.deployment)
--patch='{"spec":{"template":{"spec":{"
  "containers": [{"name": "$(inputs.params.deployment)",
  "image": "$(inputs.params.IMAGE)"}
]}}}'
```

演習で使用するyamlは準備済です

- `update-deployment`という名前のTask
- Params
 - `deployment`(deployment名)
 - `IMAGE`(インターナルのイメージ名)
- `oc patch`コマンドでcontainerのnameとimageをparamsで渡された`deployment`及び`IMAGE`に変更する



Persistent Volume Claim

演習で使用するyamlは準備済です

`persistent_volume_claim.yaml`

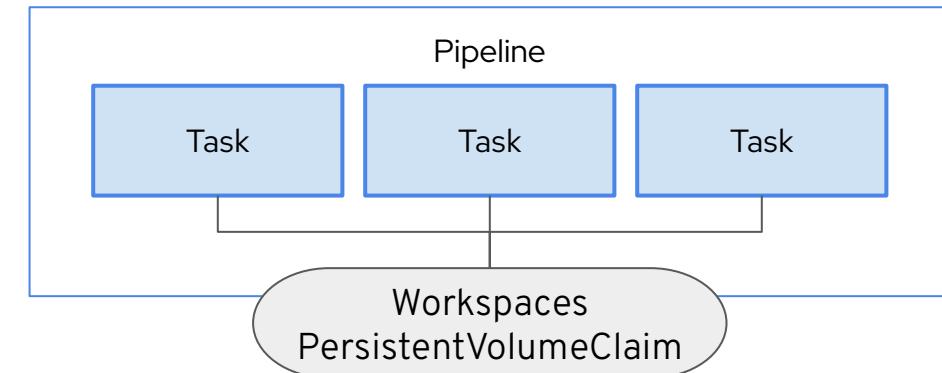
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: source-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

- Task間で共有するPersistent Volume用のsource-pvcという名前のPersistent Volume Claimを作成する

Task間でデータを共有する方法のひとつがPersistent Volumeです

Task: workspace

- Task間の共有ボリューム
 - Persistent Volume
 - Config map
 - Secret
- 大容量データに最適
 - 例:コード、バイナリ、レポート



Pipeline定義

演習で使用するyamlは準備済です

pipeline.yaml

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  workspaces:
    - name: shared-workspace
  params:
    - name: deployment-name
      type: string
      description: name of the deployment to be patched
    - name: git-url
      type: string
      description: url of the git repo for the code of deployment
    - name: git-revision
      type: string
      description: revision to be used from repo of the code for deployment
      default: "master"
    - name: IMAGE
      type: string
      description: image to be build from the code
```

- build-and-deployという名前のPipeline
- Params
 - deployment-name
 - git-url
 - git-version
 - IMAGE
- workspaceとしてshared-workspaceを利用

次ページに続く

Pipeline定義

演習で使用するyamlは準備済です

pipeline.yaml

```
tasks:  
  - name: fetch-repository  
    taskRef:  
      name: git-clone  
      kind: ClusterTask  
    workspaces:  
      - name: output  
        workspace: shared-workspace  
    params:  
      - name: url  
        value: $(params.git-url)  
      - name: subdirectory  
        value: ""  
      - name: deleteExisting  
        value: "true"  
      - name: revision  
        value: $(params.git-revision)
```

```
- name: build-image  
  taskRef:  
    name: buildah  
    kind: ClusterTask  
  params:  
    - name: TLSVERIFY  
      value: "false"  
    - name: IMAGE  
      value: $(params.IMAGE)  
  workspaces:  
    - name: source  
      workspace: shared-workspace  
  runAfter:  
    - fetch-repository
```

- git-cloneというClusterTaskを利用
- 指定のGitレポジトリから指定されたリビジョンのソースコードをcloneして取得する
- ソースコードは共有ボリュームshared-workspaceに格納される

- buildahというClusterTaskを利用
- shared-workspaceに格納されたソースコードをビルドしてIMAGEにpush
- fetch-repository Taskの後に実行

※ClusterTask : あらかじめ準備されているすべてのネームスペースで使用できるTask

Pipeline定義

演習で使用するyamlは準備済です

pipeline.yaml

```
- name: apply-manifests
  taskRef:
    name: apply-manifests
  workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
      - build-image
- name: update-deployment
  taskRef:
    name: update-deployment
  workspaces:
    - name: source
      workspace: shared-workspace
    params:
      - name: deployment
        value: ${params.deployment-name}
      - name: IMAGE
        value: ${params.IMAGE}
    runAfter:
      - apply-manifests
```

- 作成したapply-manifest Taskを利用
- build-image Taskの後に実施

- 作成したupdate-deployment Taskを利用
- apply-manifest Taskの後に実施

tkn pipeline start build-and-deployコマンドで パラメーターを指定 → PipelineRunが作成され Pipelineが実行される

パラメーター

```
claimName=source-pvc  
deployment-name=pipelines-vote-api  
git-url=https://github.com/openshift/pipelines-vote-api.git  
IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-api  
git-revision=master
```

Pipeline実行の結果、vote-apiアプリケーションがデプロイされる

pipeline-vote-api

tkn pipeline start build-and-deployコマンドで パラメーターを指定 → PipelineRunが作成され Pipelineが実行される

パラメーター

```
claimName=source-pvc  
deployment-name=pipelines-vote-ui  
git-url=https://github.com/openshift/pipelines-vote-ui.git  
IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-ui  
git-revision=master
```

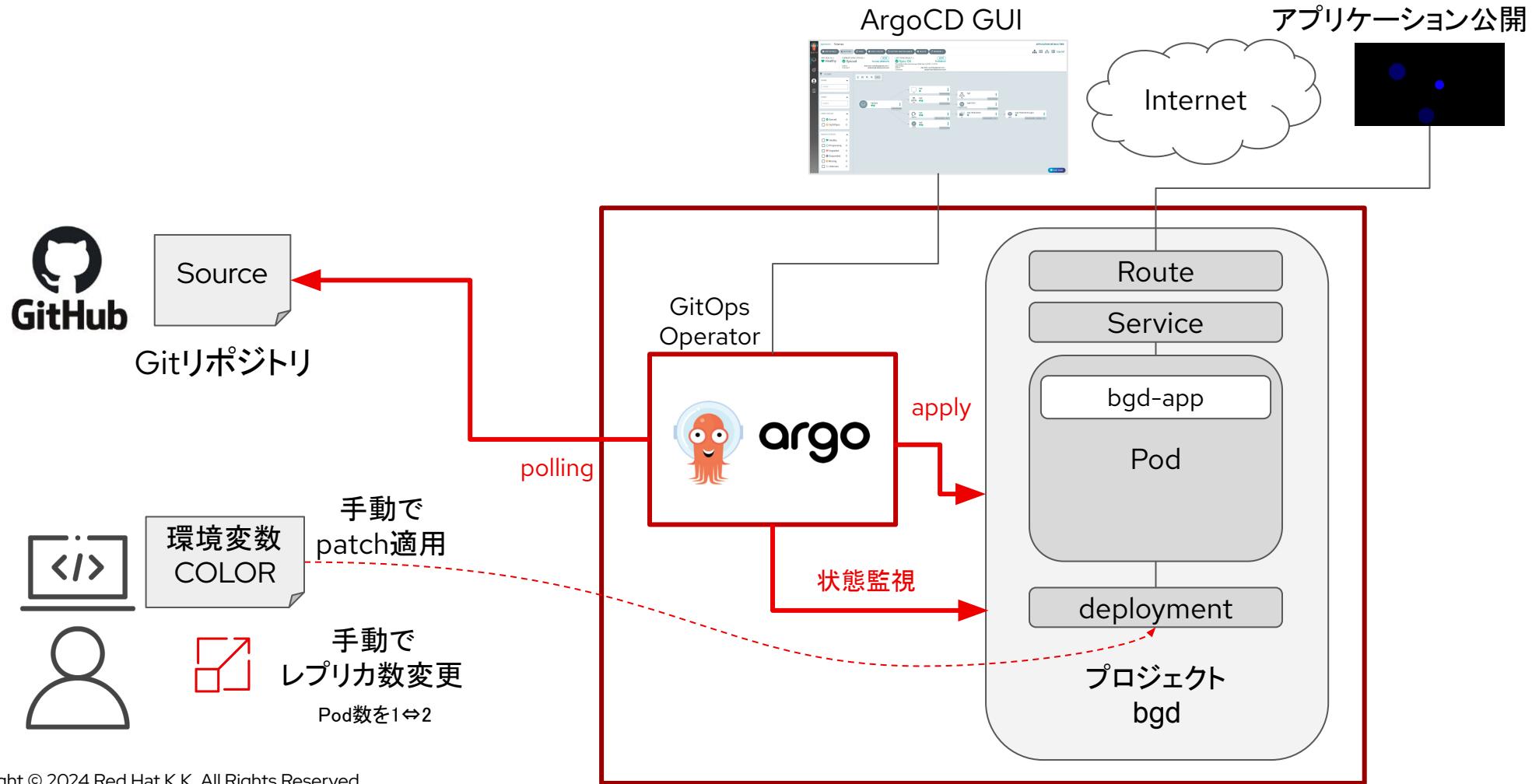
Pipeline実行の結果、vote-uiアプリケーションがデプロイされる

pipeline-vote-ui

4

OpenShift GitOps

OpenShift GitOps

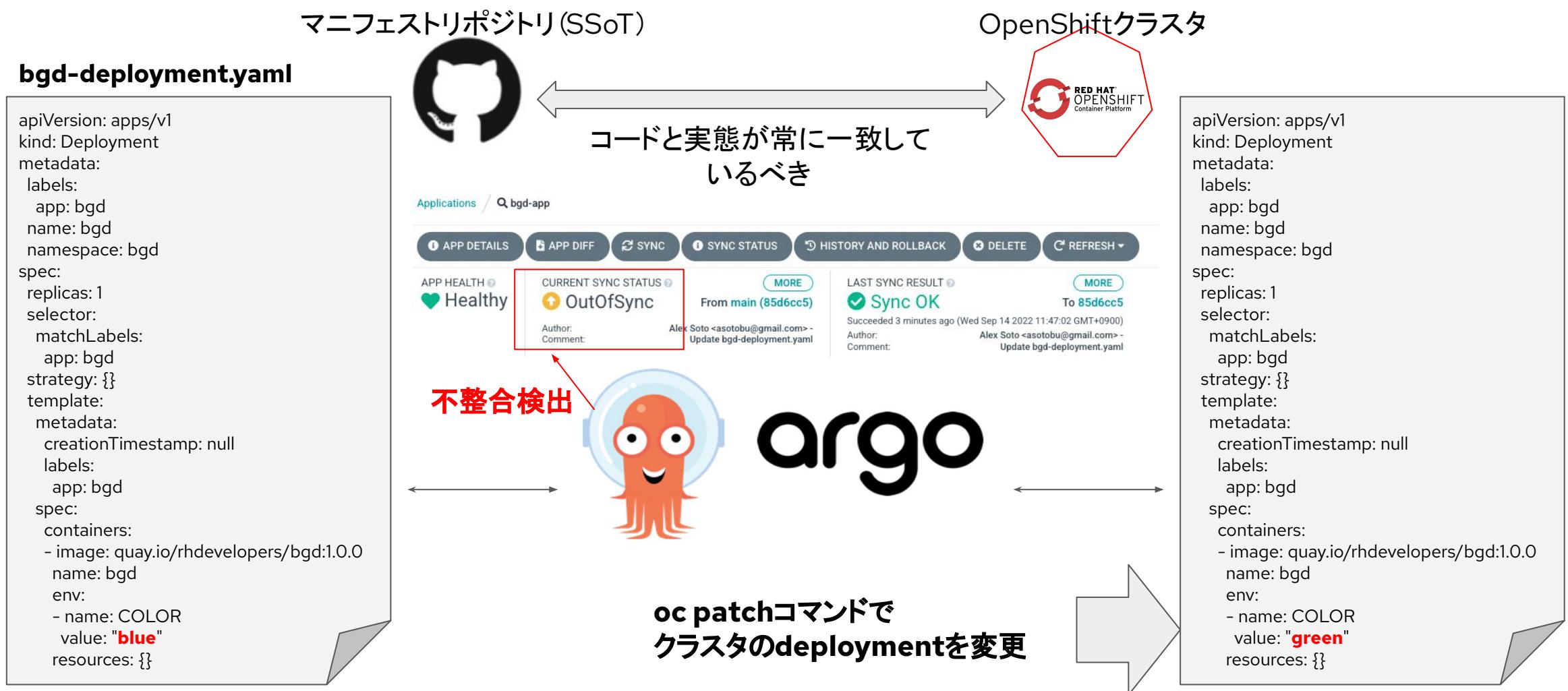


OpenShift GitOps

演習内容

- GitOps Operatorをインストールします
- ArgoCDインスタンスにGUIおよびCLIで接続します
- GitにあるArgoCD用マニフェストを利用してアプリケーションをデプロイします
- Deploymentを手動で変更します
- ArgoCDが変更を検知し、非同期として検出します
- Gitの情報と同期し、アプリケーションを元の状態に修正します
- 自動同期の設定を行います
- Webコンソールを用いてPod数を増やした場合のArgoCDの動作を確認します

不整合検出の動作



自動同期設定の動作



サンプル用マニフェスト

リポジトリ : <https://github.com/redhat-developer-demos/openshift-gitops-examples>

パス : apps/bgd/overlays/bgd

演習で使用するyamlは準備済です

bgd-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: bgd
  name: bgd
  namespace: bgd
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bgd
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: bgd
    spec:
      containers:
        - image: quay.io/rhdevelopers/bgd:1.0.0
          name: bgd
        env:
          - name: COLOR
            value: "blue"
      resources: {}
```

bgd-ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: bgd
  labels:
    argocd.argoproj.io/managed-by: openshift-gitops
```

bgd-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: bgd
  name: bgd
  namespace: bgd
spec:
  ports:
    - port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    app: bgd
```

bgd-route.yaml

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    app: bgd
  name: bgd
  namespace: bgd
spec:
  port:
    targetPort: 8080
  to:
    kind: Service
    name: bgd
    weight: 100
```

ArgoCD Application定義

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: bgd-app
  namespace: openshift-gitops
spec:
  destination:
    namespace: bgd
    server: https://kubernetes.default.svc
  project: default
  source:
    path: apps/bgd/overlays/bgd
    repoURL:
      https://github.com/redhat-developer-demos/openshift-gitops-examples
    targetRevision: main
  syncPolicy:
    automated:
    prune: true
    selfHeal: false
    syncOptions:
      - CreateNamespace=true
```

演習で使用するyamlは準備済です

- apiVersion
 - argoproj.io/v1alpha1 : ArgoCDのAPIを指定
- kind
 - Application : ArgoCDのカスタムリソース
- destination
 - https://kubernetes.default.svcはArgoCDがローカルであることを意味しています
- project
 - ArgoCDのプロジェクト(≠OpenShiftのproject)
 - ArgoCDはマルチテナントが可能のためprojectで分離できます
- source
 - マニフェストのあるリポジトリ
 - パス
 - ターゲットとなるリビジョン
- syncPolicy
 - 自動同期のポリシーを設定

ハンズオンコース紹介

Podman

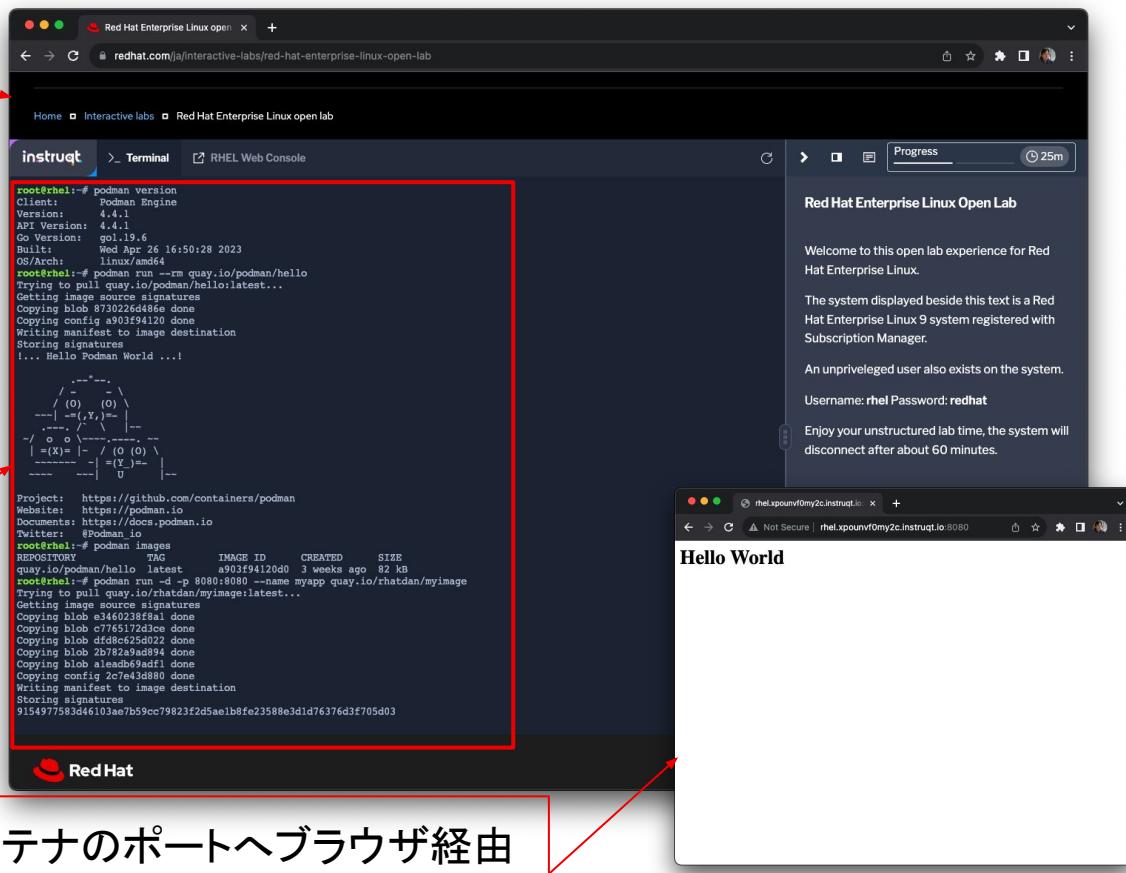
5

Podmanハンズオン

Podmanハンズオン

ブラウザを使って、払い出したハンズオン環境にアクセスする

ターミナル画面にコマンドを入力してハンズオンを進めていく。
コマンドはテキストから
ペーストが可能。
(テキスト*は別途ページで用意されます)



実行したコンテナのポートへブラウザ経由でアクセスし、動作確認が可能

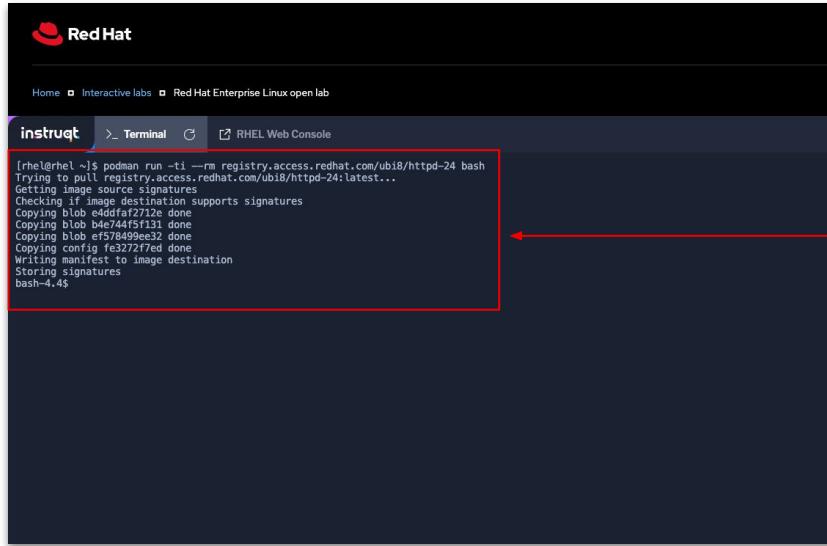
- セルフペースで学べる学習コンテンツ
- ブラウザのみで利用可能。1回の起動は連続60分まで。
- Podmanイン・アクションのコマンド例を使用しているので書籍と合わせた学習が可能
- オンラインのハンズオンイベント開催時は運営によるサポートも提供

* <https://tnk4on.github.io/podman-hands-on/>

ハンズオンの進め方

[ハンズオンドキュメント]

[演習実施環境]



```
Irhelkernel:~$ podman run -ti --rm registry.access.redhat.com/ubi8/httpd-24 bash
Trying to pull registry.access.redhat.com/ubi8/httpd-24:latest...
Getting image source signatures
Checking if image destination supports signatures
Copying blob e4ddfafaf2712e done
Copying blob bde744f5f131 done
Copying blob 5f578499e932 done
Copying config f5a221717ed done
Writing manifest to image destination
Storing signatures
bash-4.4$
```

コマンドを入力
(手打ち入力、コピペ、
どちらでもOK)



コマンドの出力結果と回答
のサンプルを確認する



2.1 コンテナの操作

コンテナの探索

(このコンテンツは本書 2.1.1に該当します)

podman run コマンドを実行し、ubi8/httpd-24 イメージをプルして実行します

```
$ podman run -ti --rm registry.access.redhat.com/ubi8/httpd-24 bash
```

(コピペ用)

```
podman run -ti --rm registry.access.redhat.com/ubi8/httpd-24 bash
```

✓ 出力結果



出力結果を開くと回答サンプルが表示される

✓ 出力結果

```
$ podman run -ti --rm registry.access.redhat.com/ubi8/httpd-24 bash
```

```
Trying to pull registry.access.redhat.com/ubi8/httpd-24:latest...
```

```
Getting image source signatures
```

```
Checking if image destination supports signatures
```

```
Copying blob 9ece777c9660 done
```

```
Copying blob 70de3d8fc2c6 done
```

```
Copying blob b653248f5bcb done
```

```
Copying config c4127096ce done
```

```
Writing manifest to image destination
```

```
Storing signatures
```

```
bash-4.4$
```

ハンズオンドキュメント

<https://tnk4on.github.io/podman-hanson/>

The screenshot shows a web page titled "Podman/ハンズオン" with a dark blue header. The header includes a user icon, the title, a search bar, and a "Back to top" button. Below the header, there's a navigation menu with links to "Home", "開始方法", "ハンズオンドキュメント", and "公開ポートへのアクセス方法". The main content area has a sidebar on the left listing chapters from "Chapter 2" to "(WIP)Chapter 11" and appendices "Appendix A" and "Appendix B". The main content area features a heading "Chapter 2 コマンドライン" and a bulleted list of topics: "レベル:初級", "見込み時間:40分", "コンテンツ概要", "コンテナの操作", "コンテナイメージの操作", and "イメージの構築". Below this, a section titled "事前作業" contains a terminal window showing the command "# su - rhel" and its output "su - rhel". At the bottom, there's a green button labeled "✓ 出力結果" with a right-pointing arrow. To the right of the main content, there's a "Table of contents" sidebar with sections for "事前作業", "2.1 コンテナの操作", "2.2 コンテナイメージの操作", and "2.3 イメージの構築", each containing several sub-links.

クロージング





アンケートのお願い

- アンケートへのご記入をお願いいたします

<https://red.ht/OCPhanson-fb>



セルフペースでできる OpenShiftハンズオン環境

Container / OpenShift ハンズオン環境のご紹介

本日のご紹介した内容以外にも実際に試すことができるハンズオン環境を準備しています。

①ハンズオンのまとめページへアクセス

OpenShiftハンズオンへようこそ

目的

- Container（Docker）の操作を体験します
- OpenShiftの超初級的内容を体験します

ハンズオンに必要な環境

- ブラウザ（Firefox, Chrome）

コース数と難易度

- Container編：2コース ★☆☆☆☆
- OpenShift編：6コース ★☆☆☆☆

難易度	
とても難しい	★★☆☆☆
難しい	★★☆☆☆
普通	★★★☆☆
少し難しい	★★★★☆
難しい	★★★★★

<https://github.com/loungeplus/moku2-public/blob/main/README.md>

コンテナやOpenShiftについて難易度ごとに複数のコンテンツを準備しています。

86

Copyright © 2024 Red Hat K.K. All Rights Reserved.

*: 本ハンズオン環境はハンズオン以外の目的で使用することを想定していません。PoC等の目的でご利用することはお控え下さい。

**: ハンズオンコンテンツの内容については予告無く変更（公開の中止を含む）することがあります。あらかじめご了承下さい。

②コンテンツを選択しハンズオン実施

instruct > Terminal 1 Web Console ①h, 10m

```
[root@crc-rwzd-master-0 ~]# oc login -u admin -p admin https://api.crc.testing:6443
Login successful.

You have access to 66 projects, the most recent accessed: You can manage them with 'oc projects'

Using project "default".
[root@crc-rwzd-master-0 ~]#
```

ターミナル画面

習します。演習は以下の7つのトピックで構成されています。

- ocコマンドを利用してプロジェクトを作成しWebコンソールで確認する
- Webコンソールを利用してソースコードをアップロードしてデプロイ、公開する
- Webコンソールでアプリケーションのビルトログを確認する
- 外部に公開されたURLにアクセスしアプリケーション動作を確認する
- ocコマンドを利用してアプリケーションを削除する
- ocコマンドでソースコードからアプリケーションをビルトしてデプロイ、公開する
- ローカルコードを使用してビルトをトリガーする

ハンズオンの説明

演習の概要図

コマンドラインを使用してOpenShiftにログインする
このトピックではコマンドラインを用いてOpenShiftにログイン

Exit Skip Next

コースにより環境の立ち上げに最大20分程度かかります。起動したら説明に従いハンズオンを実施下さい。



イベントのご案内

APPENDIX

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions.

Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.



[linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)



[youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)



[facebook.com/redhatinc](https://www.facebook.com/redhatinc)



twitter.com/RedHat