

Red Hat OpenShift Platform 入門編ハンズオンワークショップ

2023/12/20

Red Hat OpenShift Platform 入門編ハンズオンワークショップ

12月20日	
15:00 - 15:10	受付/案内
15:10 - 15:25	ハンズオンコース説明
15:25 - 16:50	ハンズオン(セルフペース)
16:50 - 17:00	クロージング

- 投影資料へのアクセス：<https://red.ht/OCPhandson>

Agenda

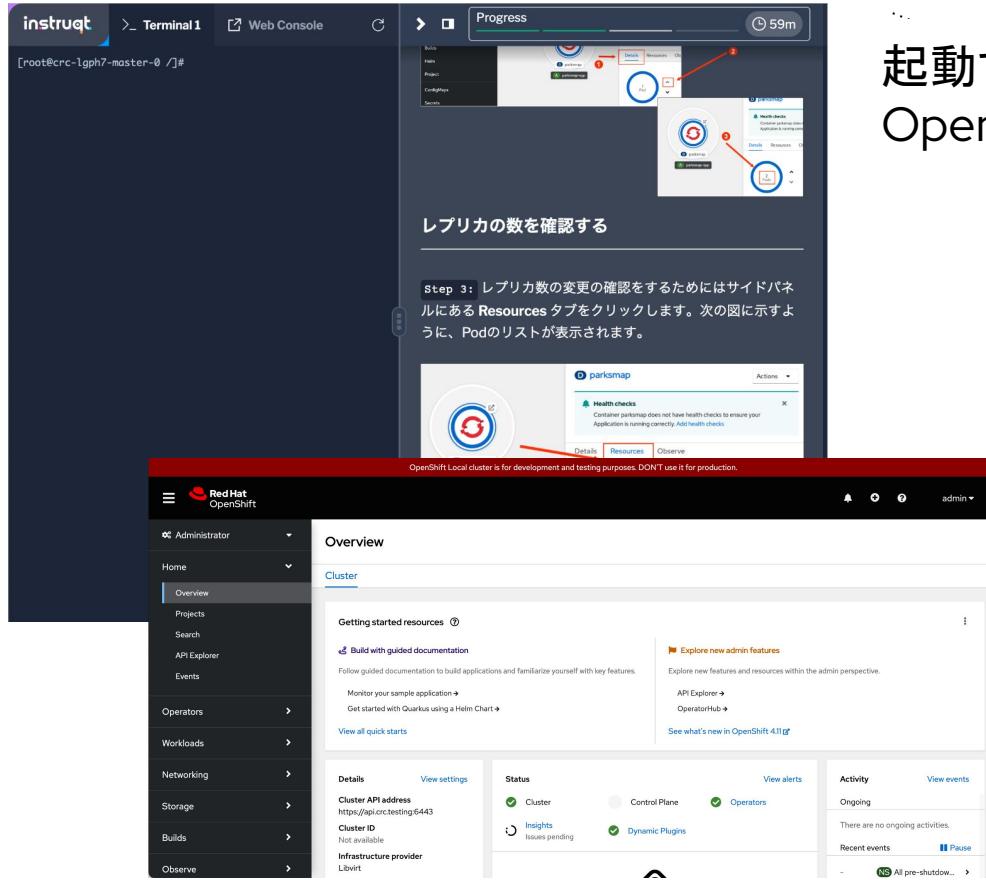
- 演習内容
- 使用する環境
- 演習
- クロージング

演習内容

<u>イメージを利用したアプリケーションのデプロイ</u>	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クラスター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。
<u>ソースコードを利用したアプリケーションのデプロイ</u>	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でのビルドについて理解を深めます。
<u>OpenShift Persistent Volume</u>	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。
<u>OpenShift Pipelines</u>	OpenShift Pipelinesを利用します。TaskおよびPipelineを作成し、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を自動化できることを学びます。
<u>OpenShift GitOps</u>	OpenShift GitOpsを利用します。Git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、OpenShift GitOpsを利用すると、GitをSSOT(Single Source of Truth)としてアプリケーションのデプロイが自動化されることを学びます。
<u>Working with Kustomize</u>	ArgoCDとKustomizeを利用してアプリケーションをデプロイします。
<u>Working with Helm</u>	ArgoCDとHelmの連携によるアプリのデプロイを体験します。

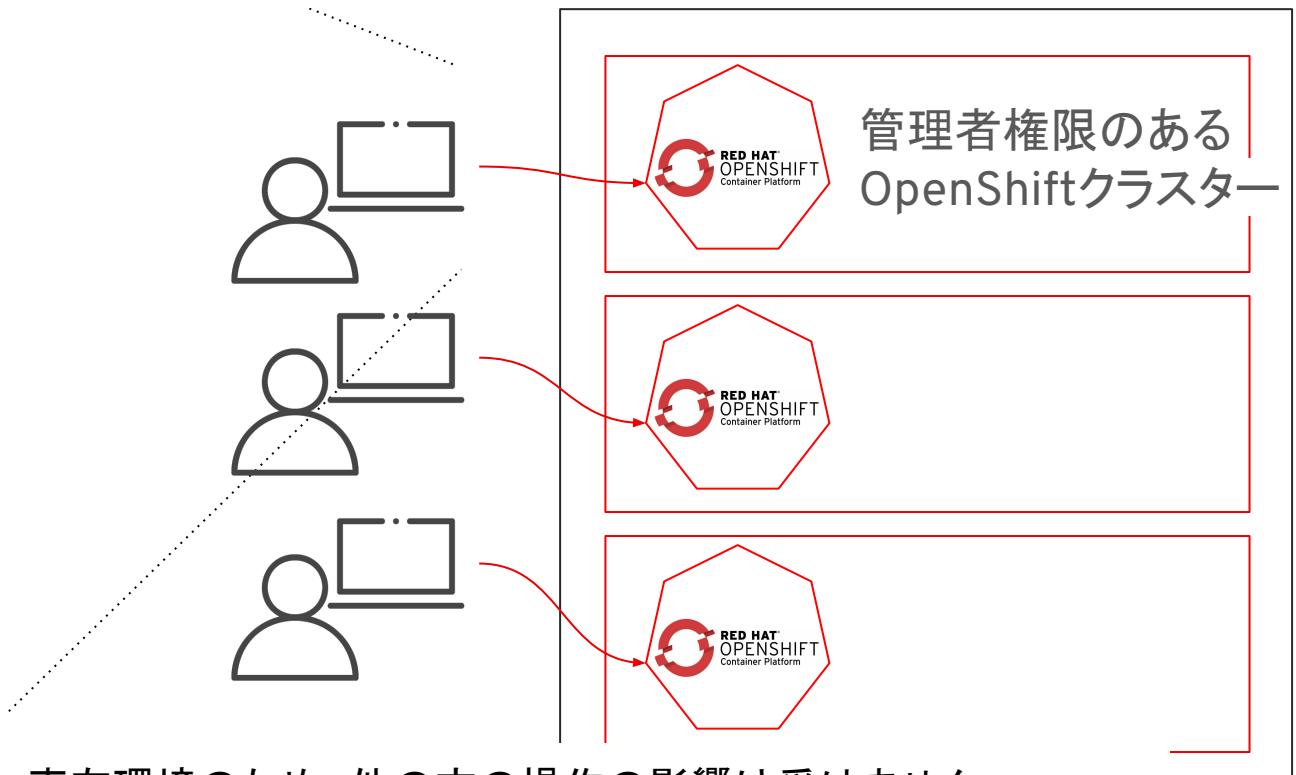
使用する環境

ブラウザのみでOpenShiftを体験できるハンズオン環境です



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。

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。

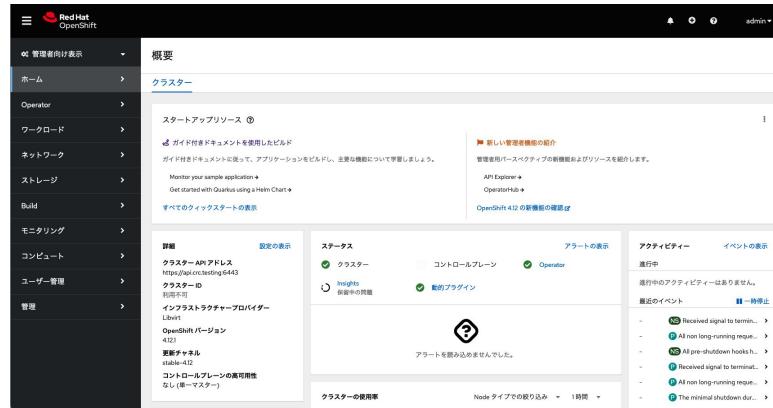
2つのレプリカがあることがわかります。それでは、

Exit Skip Next

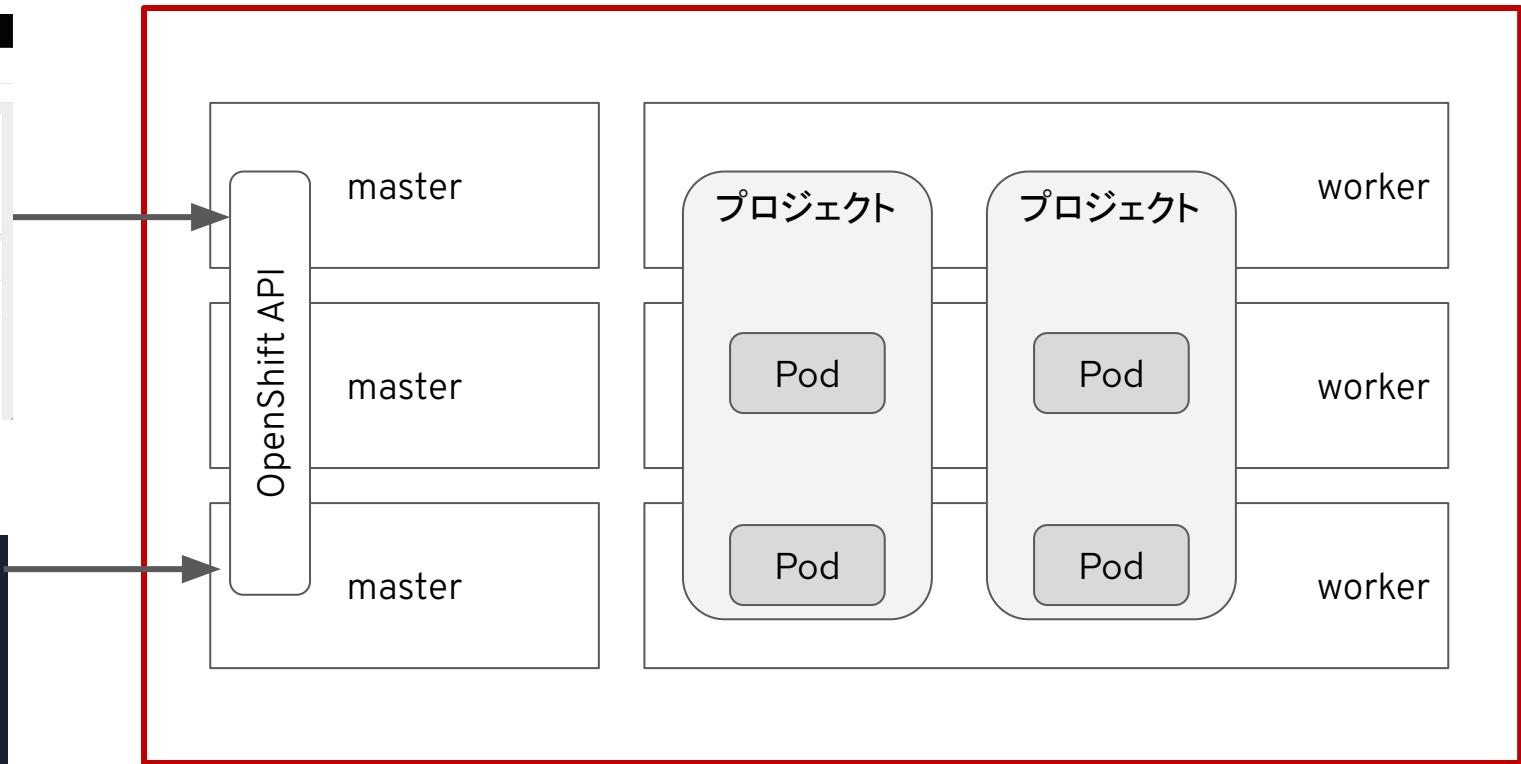
Next

OpenShiftクラスタの操作インターフェース

GUI(Webコンソール)



OpenShiftクラスター



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  Git@9c4f38  Complete  45 minutes ago  1m39s
[root@crc-rwzd-master-0 ~]#
```

※本ワークショップではクラスタ構成については言及しません

OpenShift Webコンソール

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 'Overview' page of the Red Hat OpenShift web console. The left sidebar includes links for Home, Operator, Workloads, Network, Storage, Build, Monitoring, Compute, User Management, and Management. The main content area has tabs for 'Overview' (selected), 'Cluster', and 'Build'. The 'Overview' tab displays sections for 'Start Application Resources' (with a link to a guide), 'Cluster Status' (with a 'Monitor your sample application' link), and 'Cluster Activity' (listing recent events like 'Received signal to terminate...' and 'All non long-running requests handled')). A central panel shows a chart with the message 'Alerts to read: 0'.

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' view for a project named 'myproject'. The left sidebar lists 'Topology', 'Monitoring', 'Search', 'Build', 'Helm', 'Project', 'ConfigMap', and 'Secret'. The main area shows a 'Topology' view for the 'blog-django-py' application, which is running on a single pod. To the right, there are tabs for 'Resources' (listing a 'Secret'), 'Logs' (showing a log entry for a build completion), and 'Service' (listing a service port). The top navigation bar shows the project name 'myproject' and the application name 'blog-django-py'.

CLI(ocコマンド)

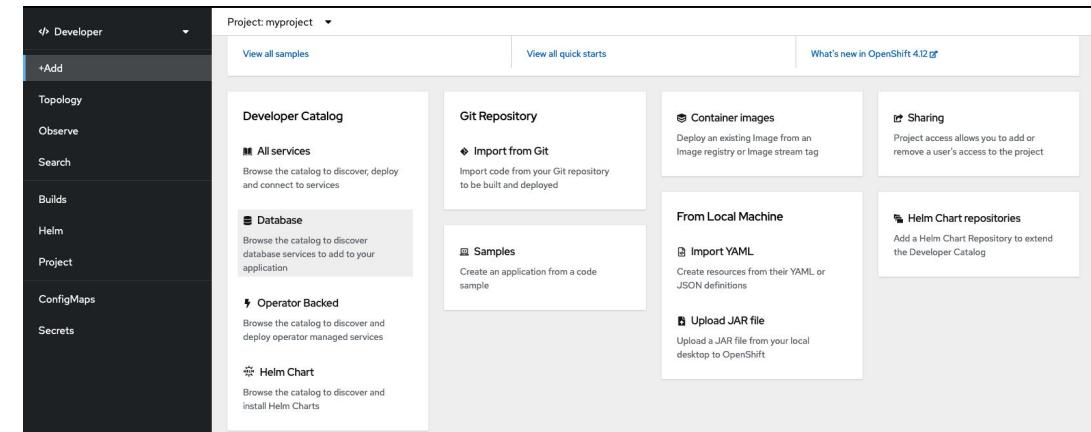
- コマンドラインから使用できるOpenShift 操作ツール
 - Linux,Windows,Mac用バイナリがあります
- 形式:oc command [option] [type] object [flags] [command-options]
 - 認証:login、logout
 - プロジェクト操作:new-project、projects、project
 - アプリケーション操作:new-app
 - 情報参照・編集:get、describe、export、edit、patch、delete、types
 - ルーティング:expose
 - 調査・トラブルシューティング:logs、exec、rsh、rsync

ハンズオンコース紹介

アプリケーションデプロイ

OpenShift Webコンソールでのアプリケーションの作成

デプロイ方式	概要
Developer Catalog	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
Import from Git	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
Container Images	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
Import YAML	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
Database	Developer Catalog を参照して、必要なデータベースサービスを選択し、これをアプリケーションに追加します
Helm Chart repositories	Helmチャートレポジトリーを追加してDeveloper Catalogを拡張します



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

セルフペースで実施できるハンズオン環境を用いて、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。

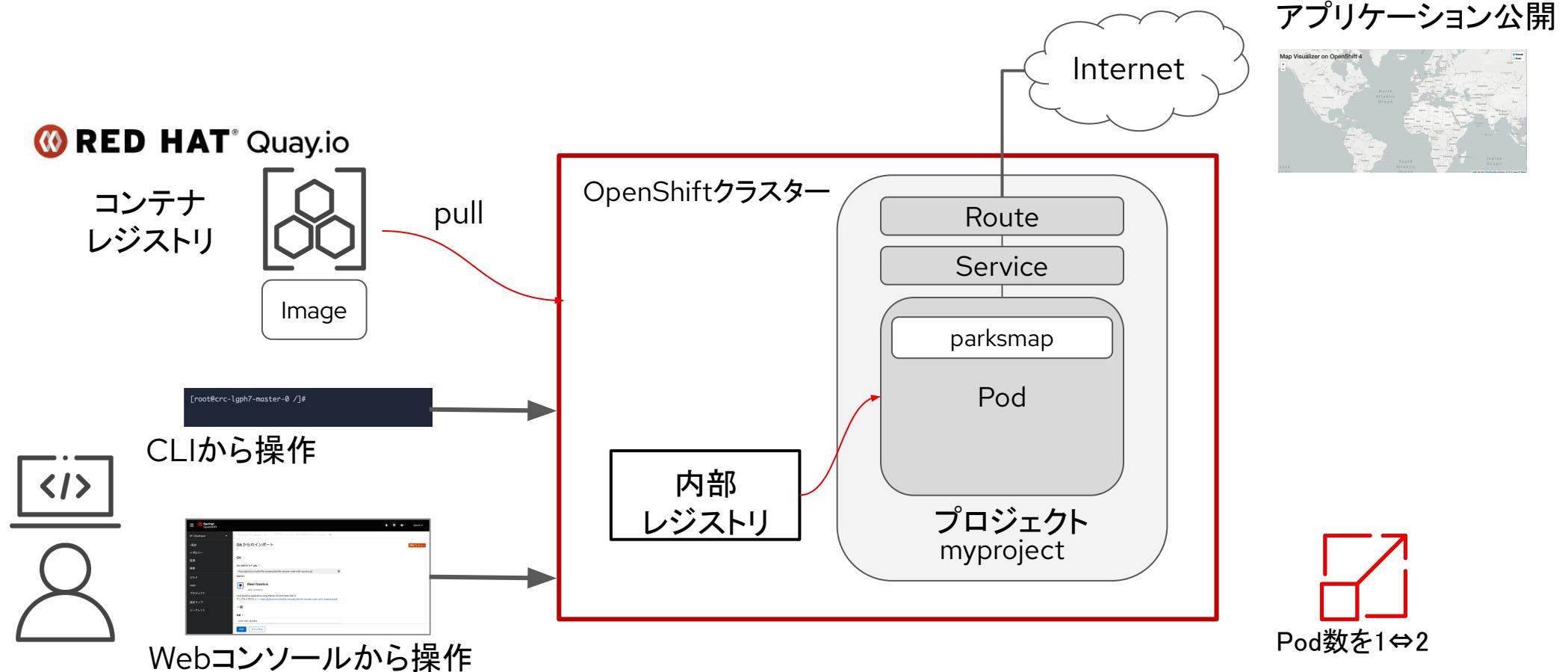


ゴール

- 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

↑クリックするとハンズオン環境へアクセスできます



イメージを利用したアプリケーションのデプロイ

↑クリックするとハンズオン環境へアクセスできます

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ビューの理解を深めます
- アプリケーションをスケールアップ・ダウントします
- 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
- CLIを利用してアプリケーションを公開します

イメージのデプロイ



Project: myproject ▾ Application: All applications ▾

Deploy Image

Image

Deploy an existing Image from an Image Stream

- Image name from external registry

quay.io/openshiftroadshow/parksmap:latest

Validated

To deploy an Image from a private registry, you must [create an Image pull secret](#) with your Image registry credentials.

- Allow Images from insecure registries

- Image stream tag from internal registry

Runtime icon



The icon represents your Image in Topology view. A label will also be added to the resource defining the icon.

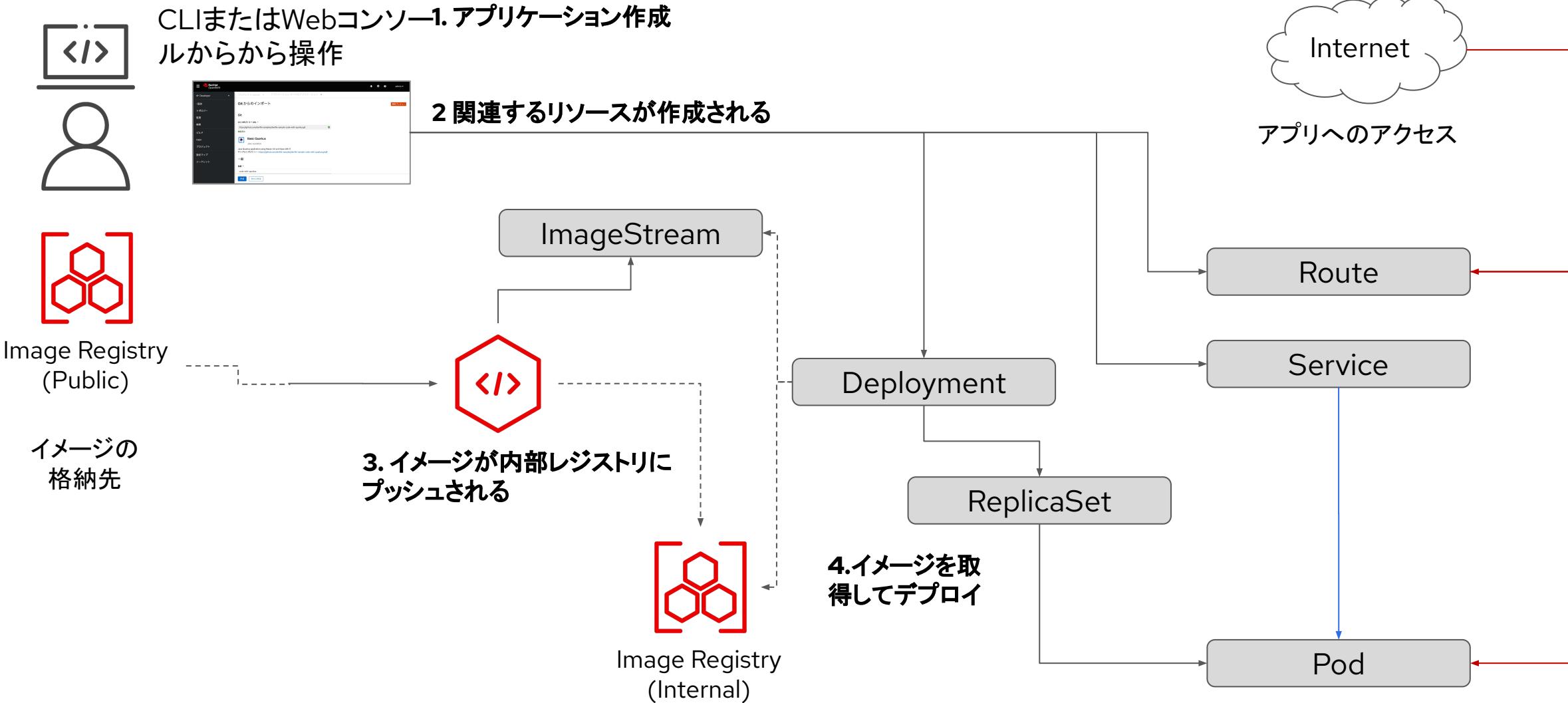
General

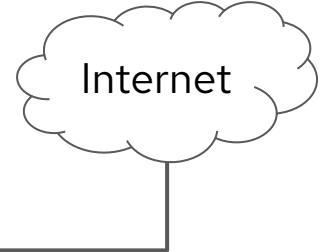
Application name

OpenShiftの主なリソース

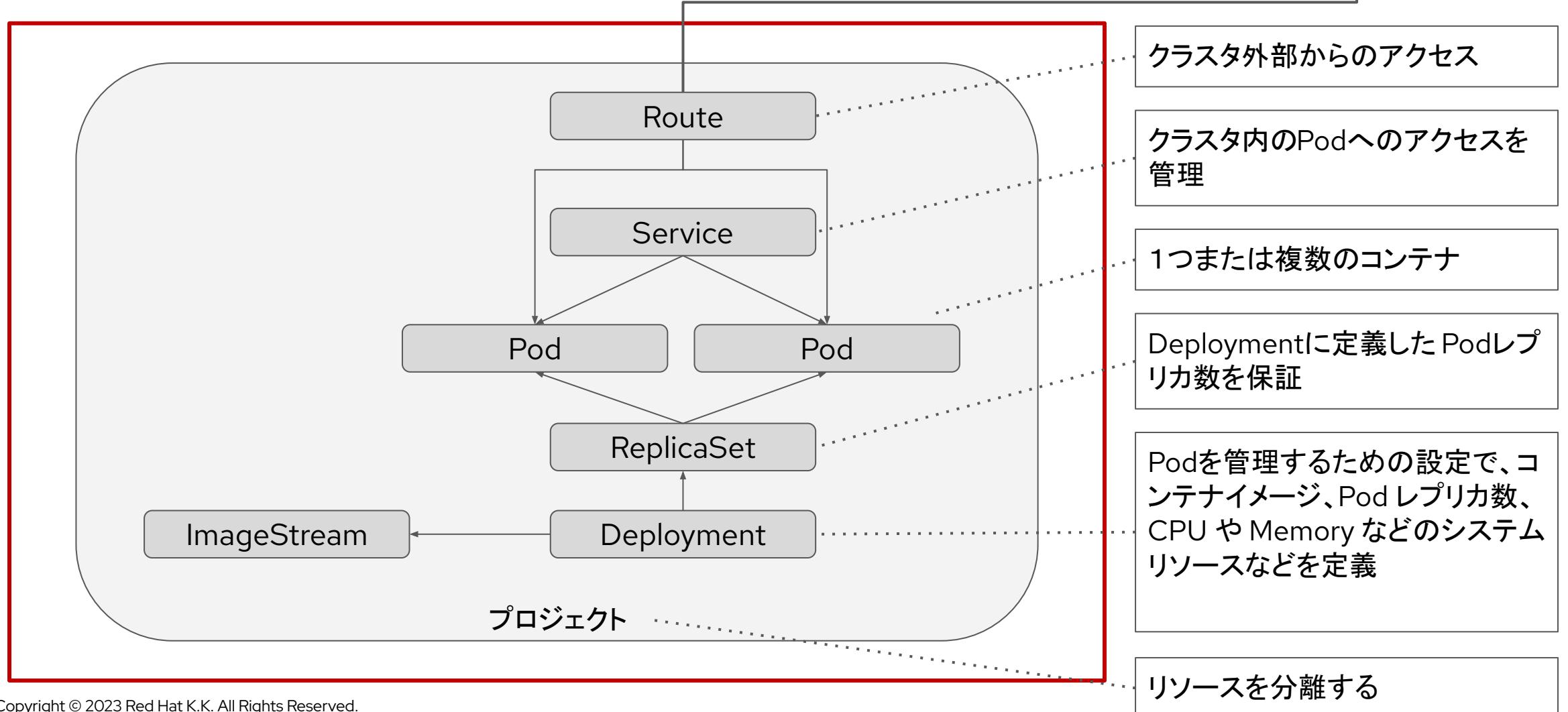
リソース	概要
Project	Kubernetesのnamespaceとほぼ同じ。Project単位でリソースを分離。管理の観点からは、各Projectはテナントのように考えることができる
Pod	ホスト上にデプロイされた1つまたは複数のコンテナ。Podは、OpenShiftで定義、デプロイ、管理できるコンピュートリソースの最小の単位
BuildConfig	OpenShift 独自のAPIとして実装され、コンテナイメージをビルドするための定義を含む
ImageStream	OpenShift内部で管理されるコンテナレジストリのイメージのメタデータを参照
Deployment	Podを管理するための設定で、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

アプリケーションデプロイとリソース

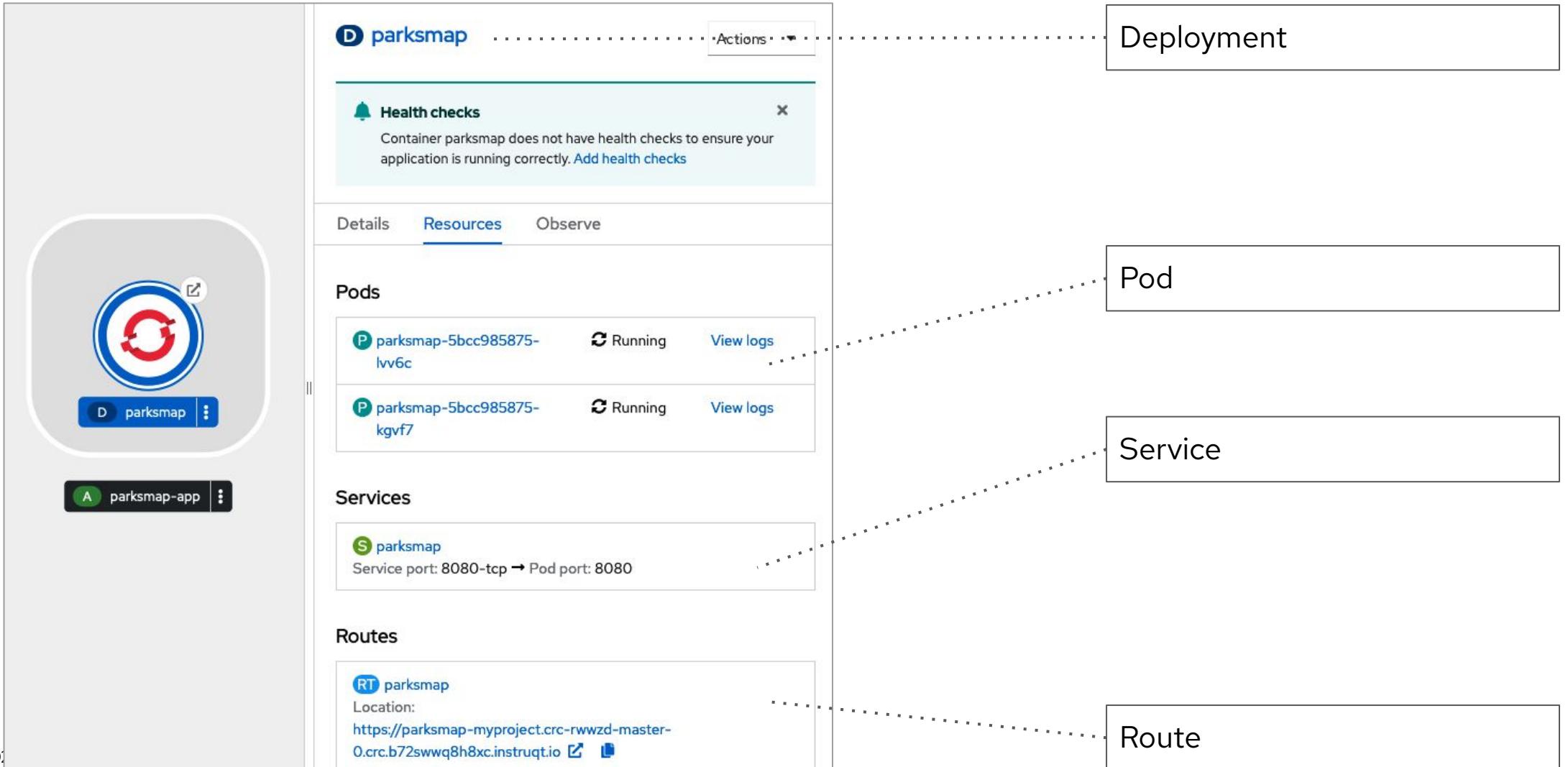




アプリケーションデプロイとリソース



トポロジービューにおけるリソース



DeploymentとReplicaSet

Project: myproject ▾

Deployments > Deployment details

D parksmap

Details Metrics YAML ReplicaSets Pods Environment Events

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

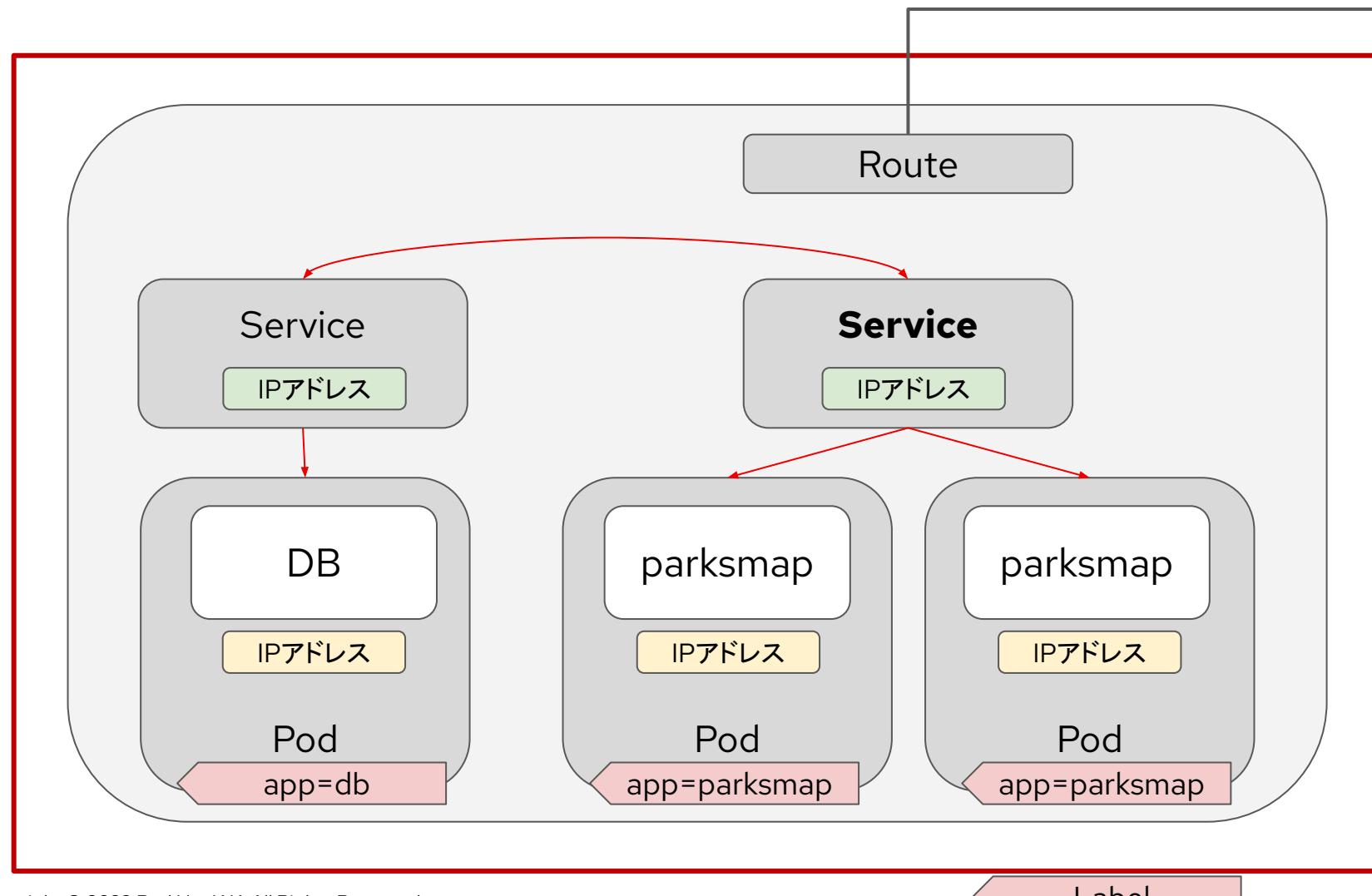
Annotations:

現在のPodの数 → Current count: 2

有るべきのPodの数 → Desired count: 2

Copyright © 2023 Red Hat K.K. All Rights Reserved.

Service



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

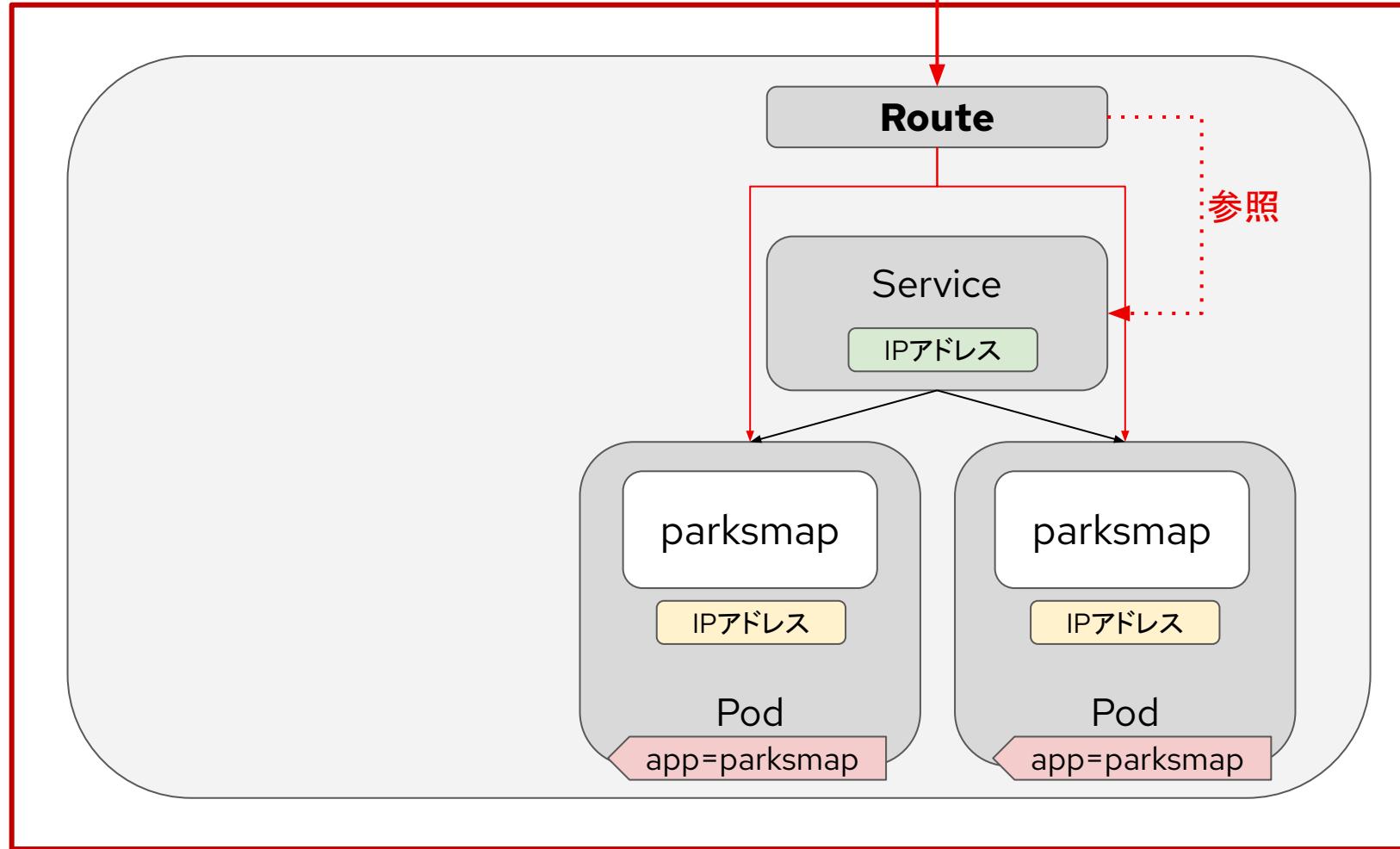


- ServiceのIP(またはService名)に対して通信を行うと、ServiceがPodに割り振る

- Serviceは管理すべきPodをLabelを見て判断している

Route

parksmap-myproject.crc-rwwzd-master-0.crc.vbkokqmxtyn.instruqt.io



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

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

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

セルフペースで実施できるハンズオン環境を用いて、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。

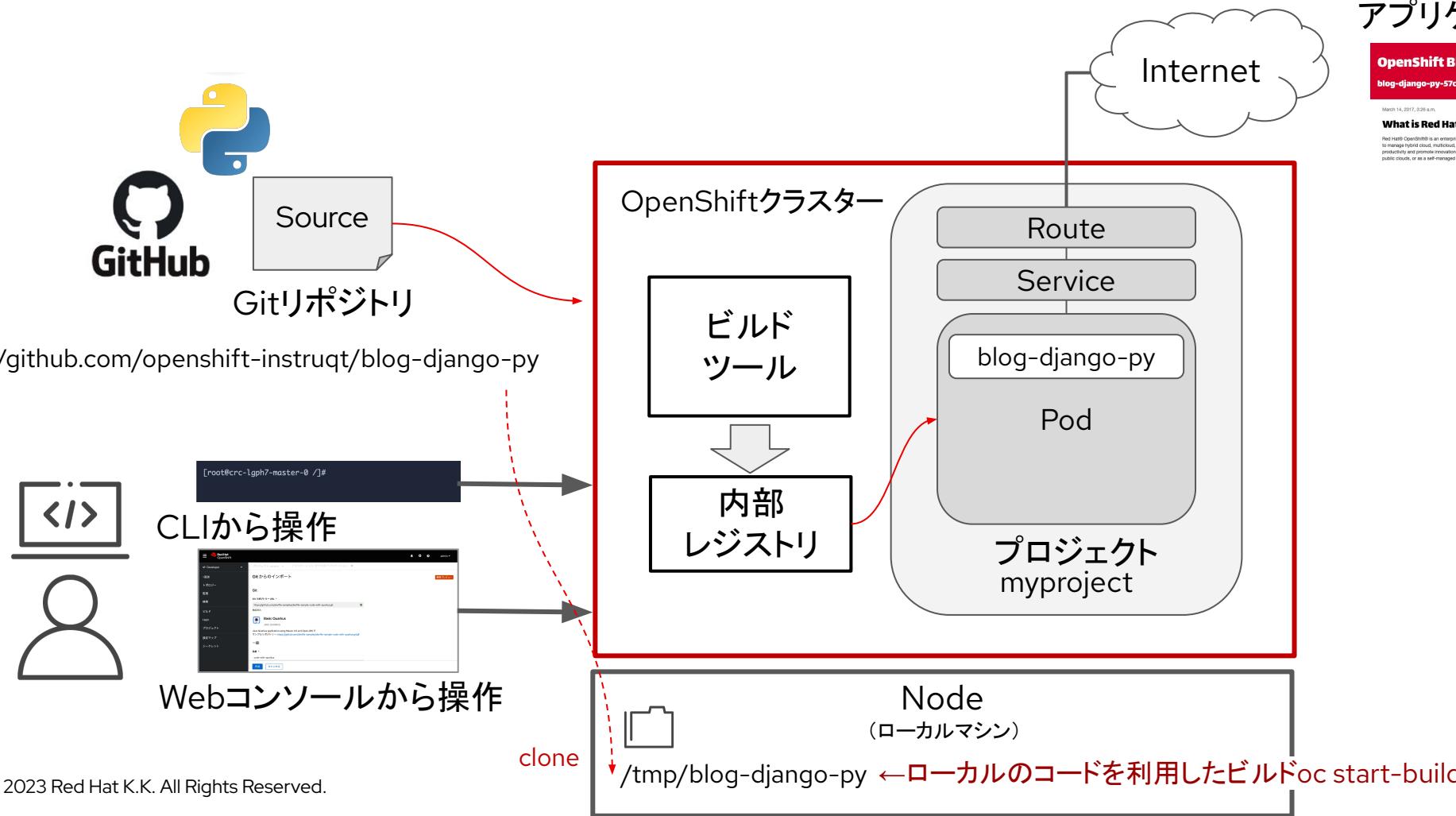


ゴール

- 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
- アプリケーションを公開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

↑クリックするとハンズオン環境へアクセスできます



アプリケーション公開



ソースコードを利用したアプリケーションのデプロイ

↑クリックするとハンズオン環境へアクセスできます

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

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 *

`https://github.com/openshift-instrukt/blog-django-py`

Validated

Try sample ↗

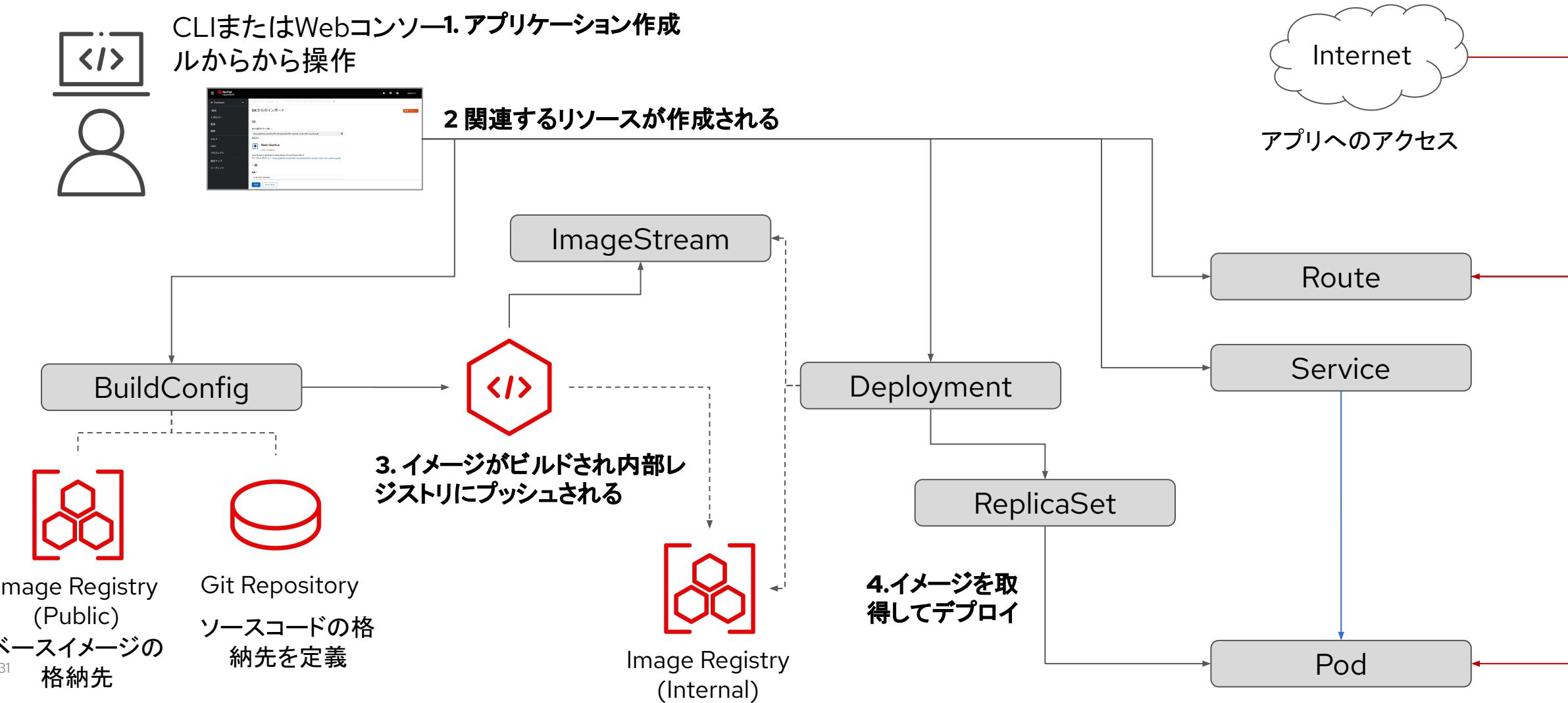
› Show advanced Git options

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

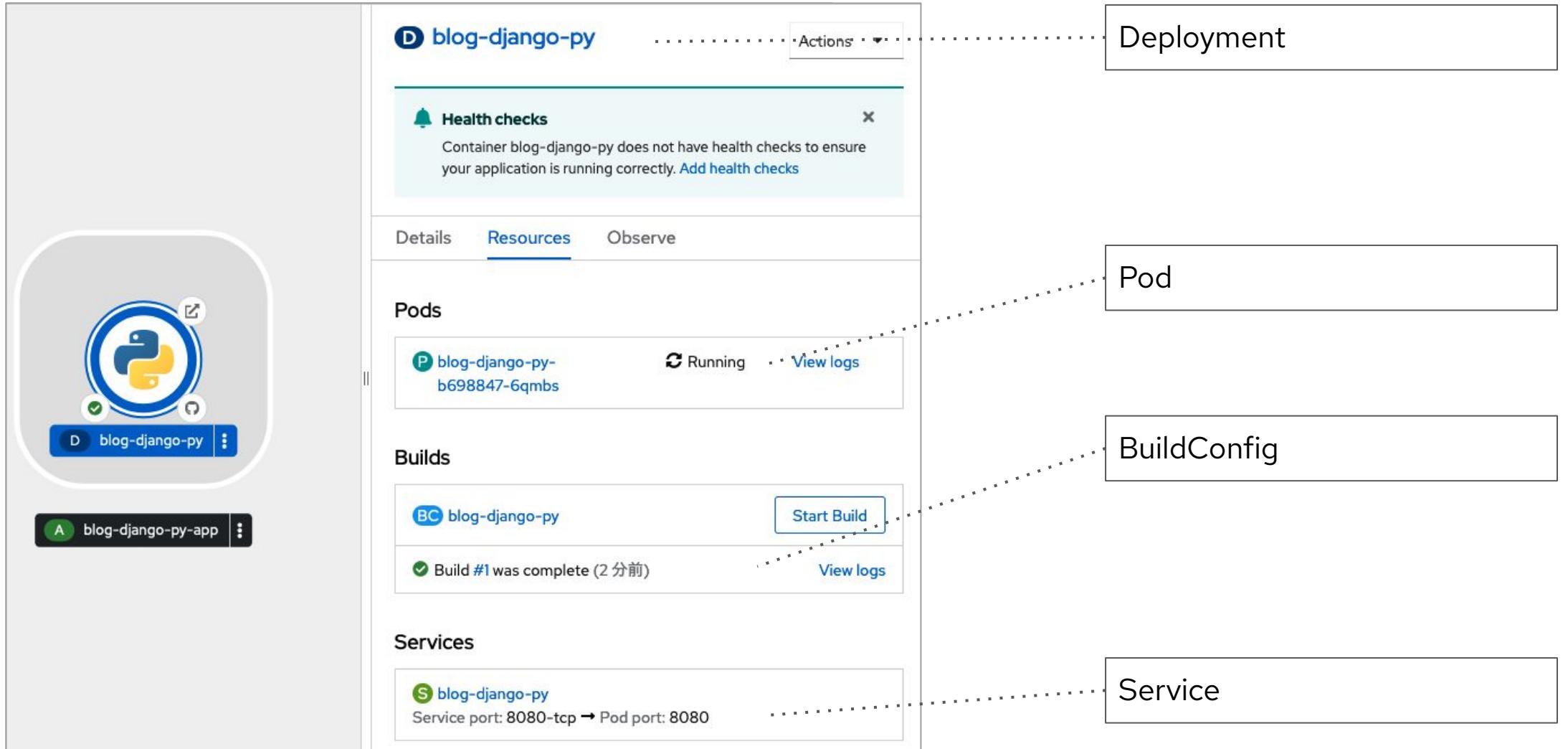
アプリケーションデプロイとリソース



ビルドログの確認



トポロジービューにおけるリソース



3

OpenShift Persistent Volume

OpenShift Persistent Volume

↑クリックするとハンズオン環境へアクセスできます

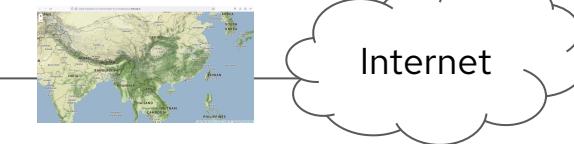
RED HAT® Quay.io

コンテナ
レジストリ



Image

アプリケーション公開



Internet

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Persistent Volume
- アプリケーションがPersistent Volume(永続ボリューム)を利用する
よう設定を行います
- Persistent Volume(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

ゴール

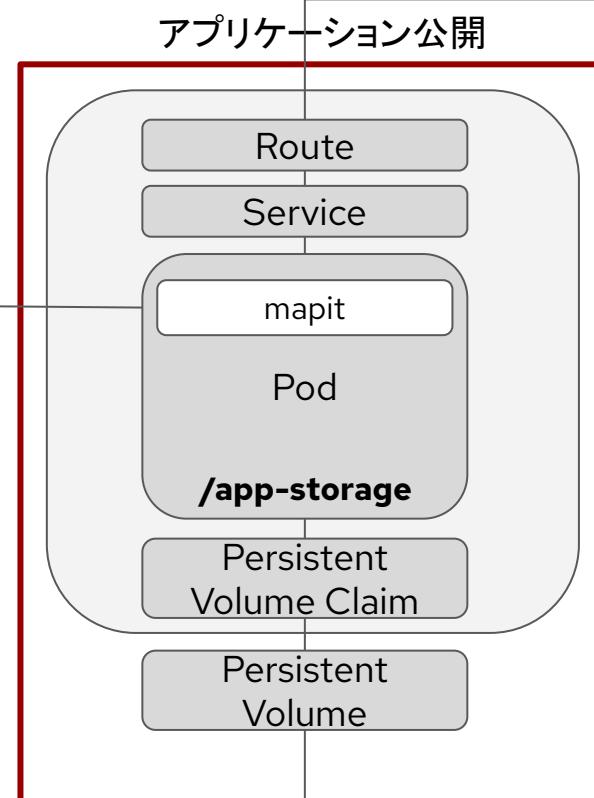
- Persistent Volume(永続ボリューム)の動作を確認します



CLIから操作



Webコンソールから操作



ハンズオンコース紹介

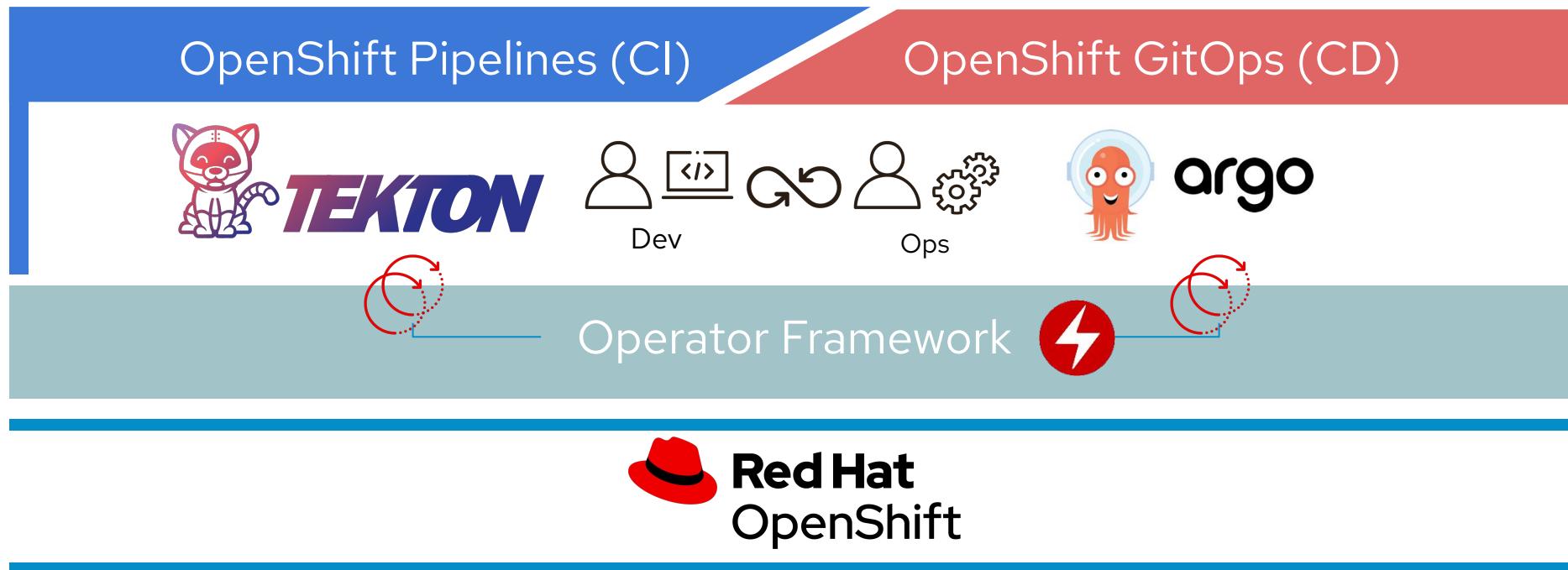
OpenShift Pipelines

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

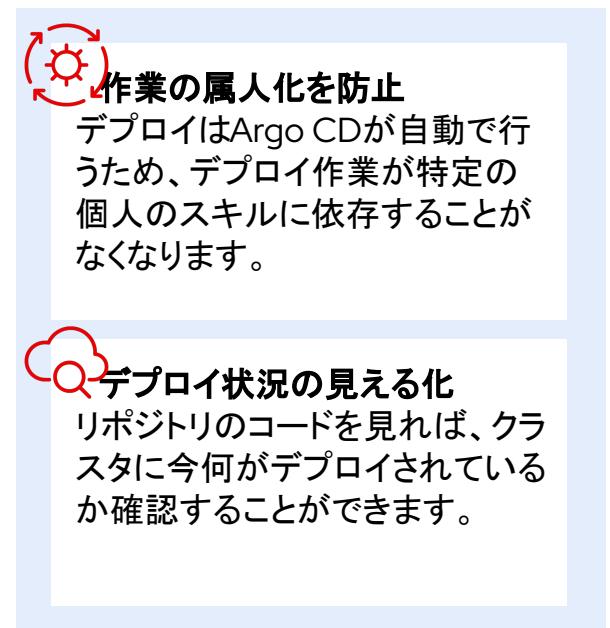
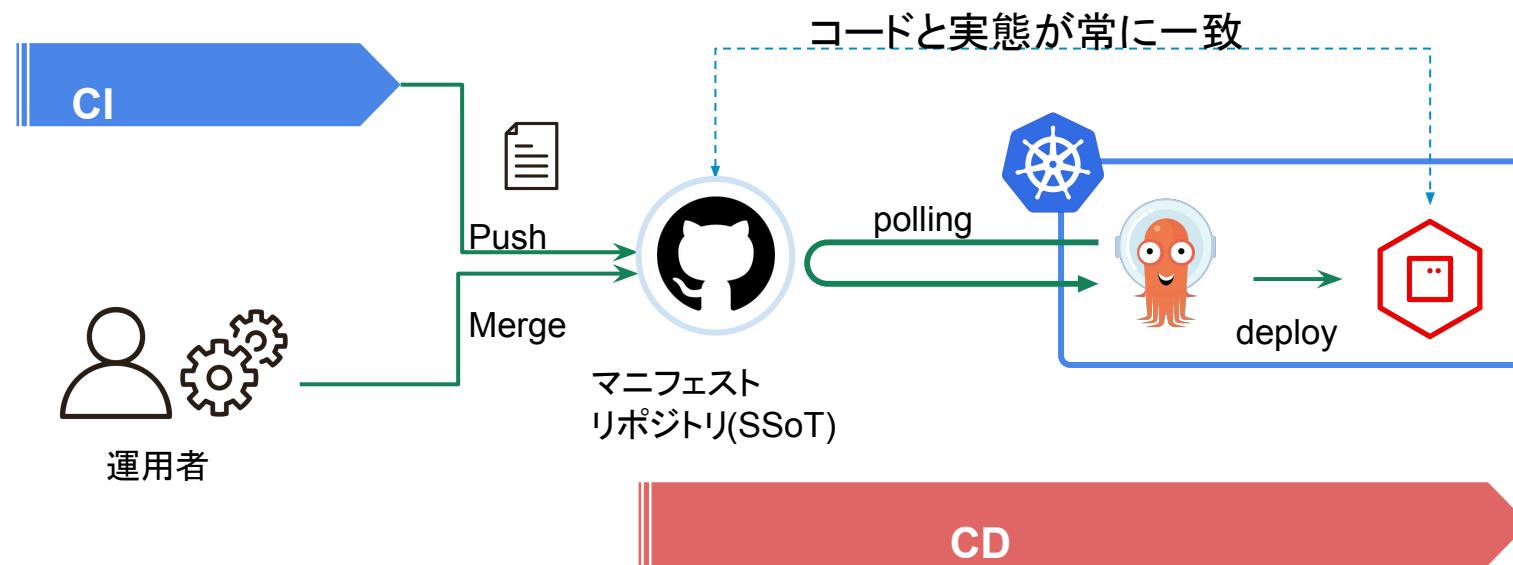
TektonのCIパイプラインはコンテナとして実行されます。開発者のアクション(Gitリポジトリへの操作)を起点とし、必要なタイミングでコンテナを立ち上げるため、クラスタのリソースを有効活用できます。



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。



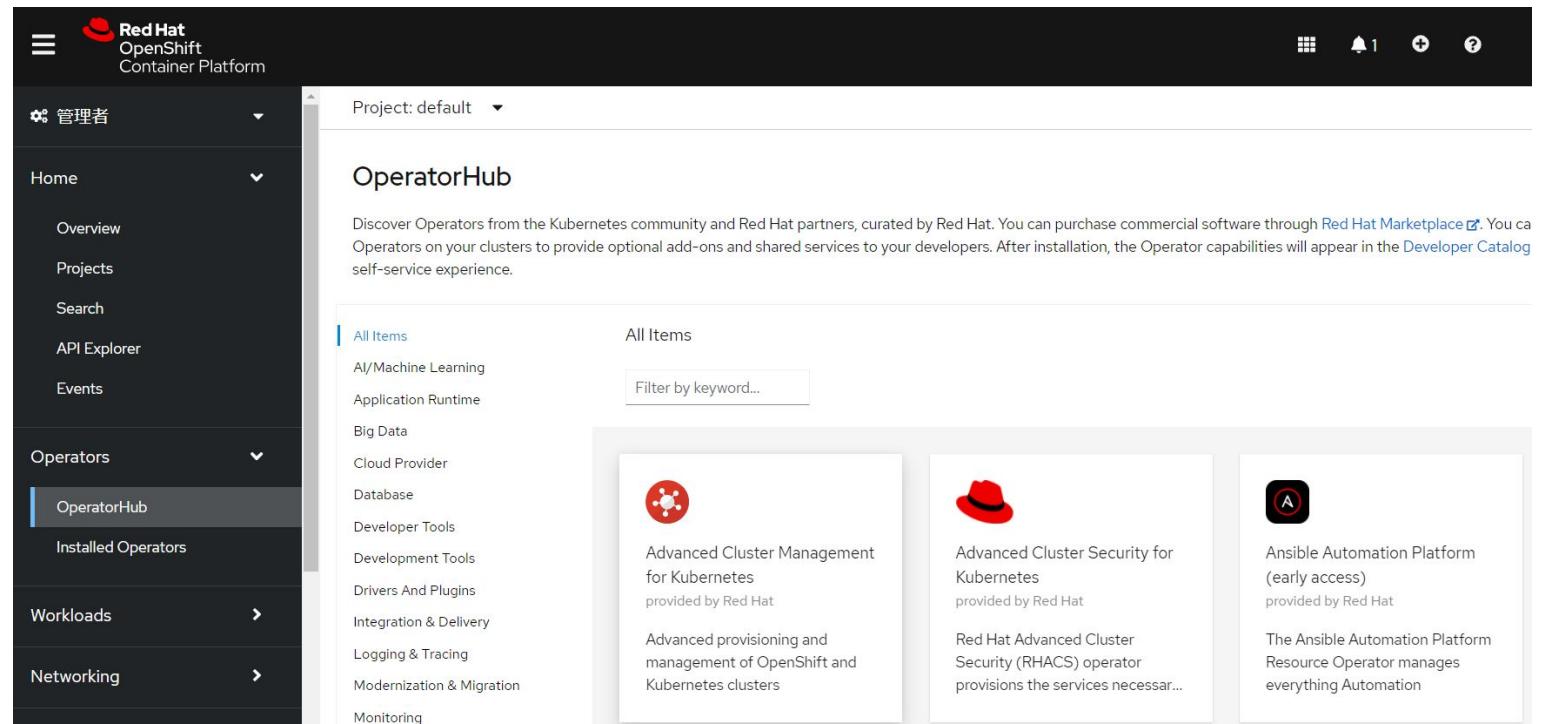
OperatorHub

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



4

OpenShift Pipeline

演習内容

OpenShift Pipelines

セルフペースで実施できるハンズオン環境を用いて、OpenShift Pipelinesを利用します。TaskおよびPipelineを作成し、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を自動化できることを学びます。

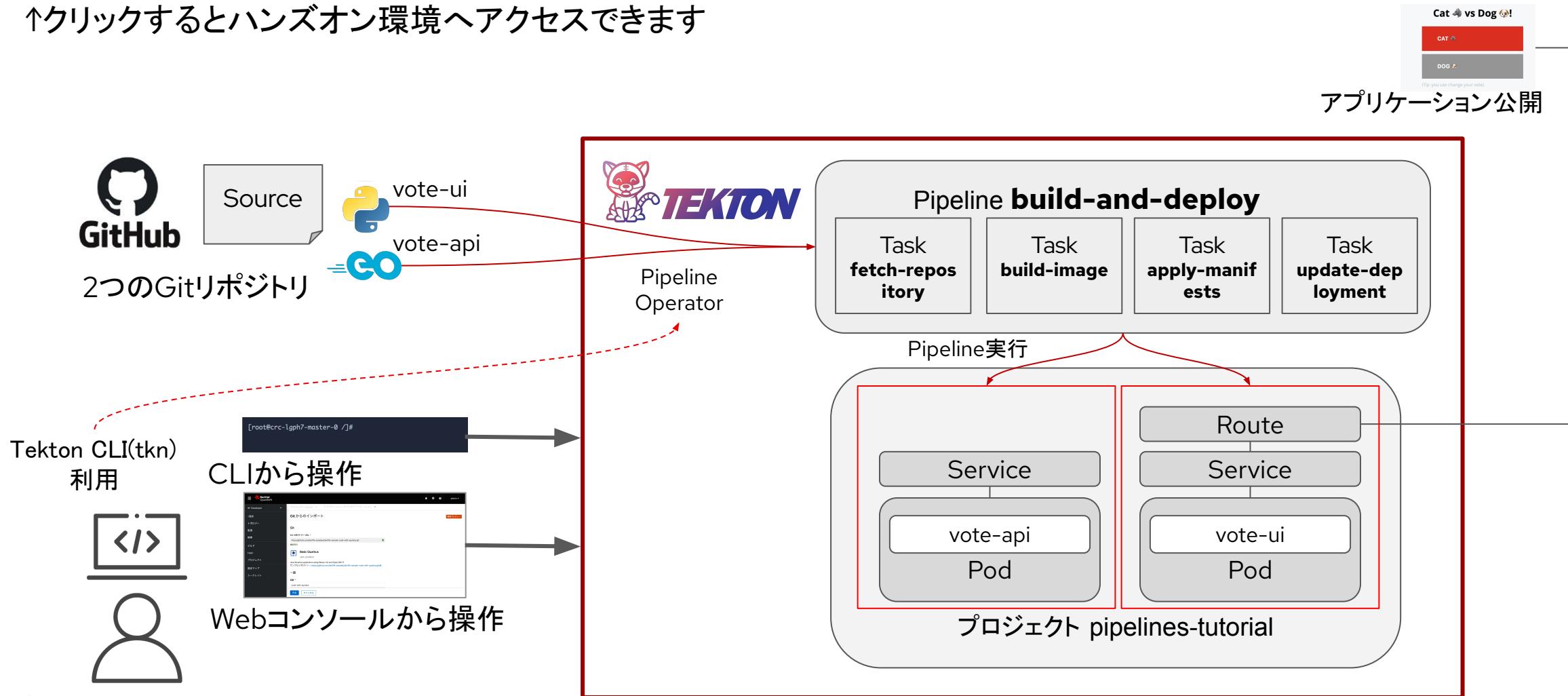


ゴール

- OpenShift Pipeline Operatorのインストール方法を学習します
- 基本コンセプトであるTask/TaskRun/Pipeline/PipelineRunを学習します
- Pipelineを実行して、アプリケーションがデプロイされることを学習します

OpenShift Pipelines

↑クリックするとハンズオン環境へアクセスできます



OpenShift Pipelines

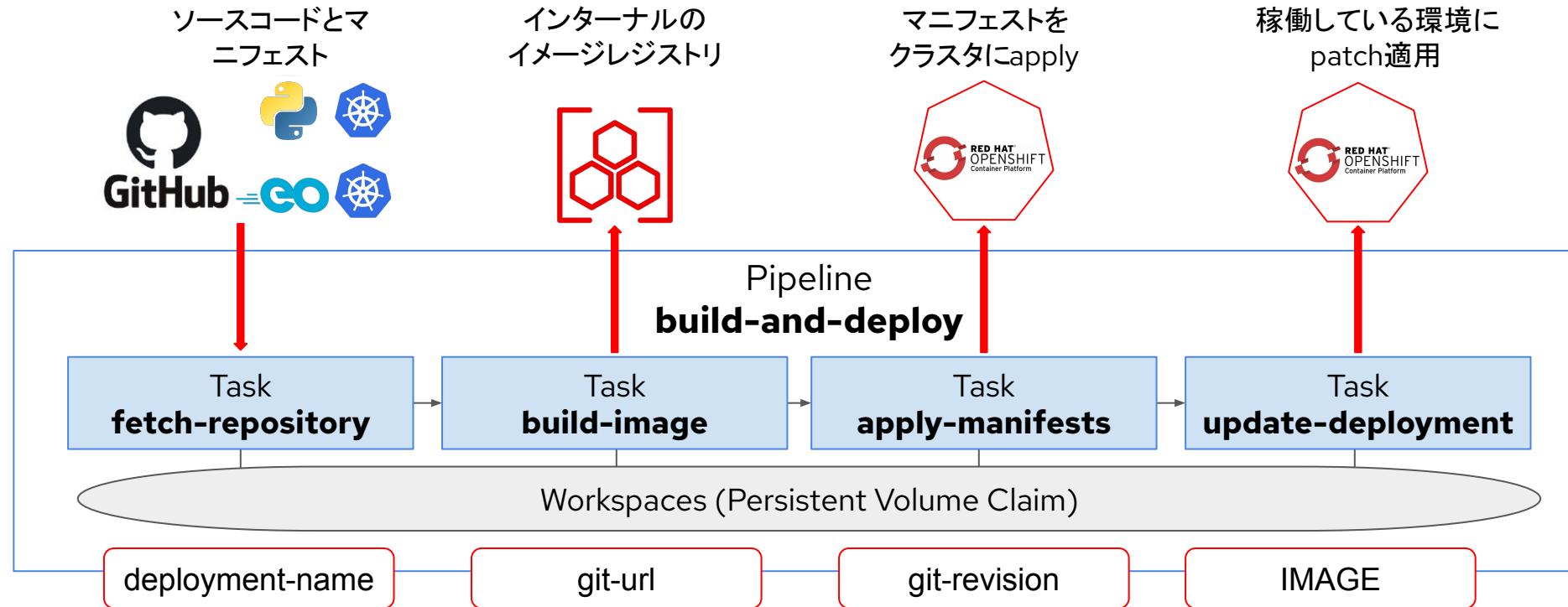
↑クリックするとハンズオン環境へアクセスできます

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

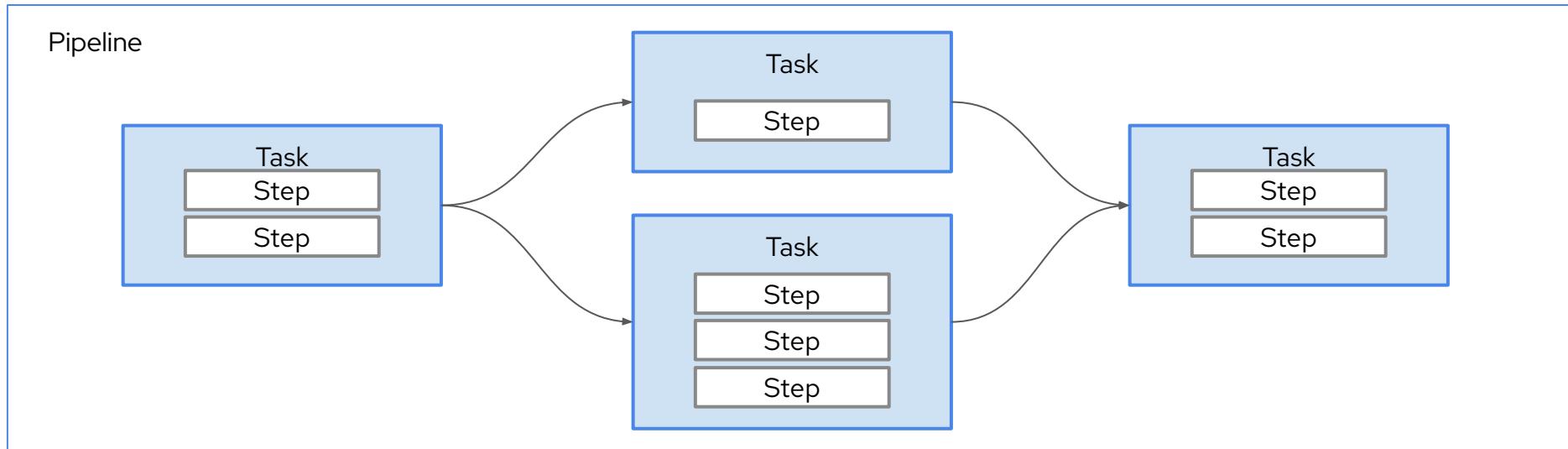
作成するPipeline

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。



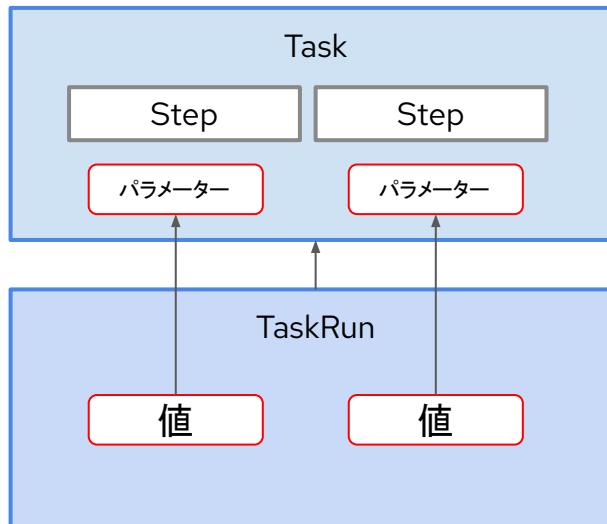
Tektonのリソース

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。

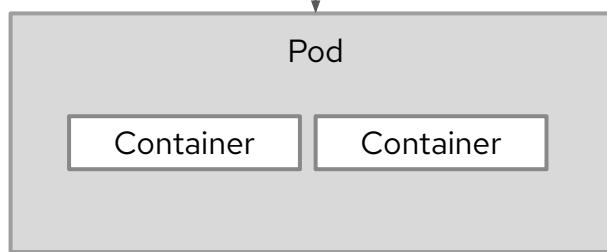


TaskRunとPipelineRun

TaskRun



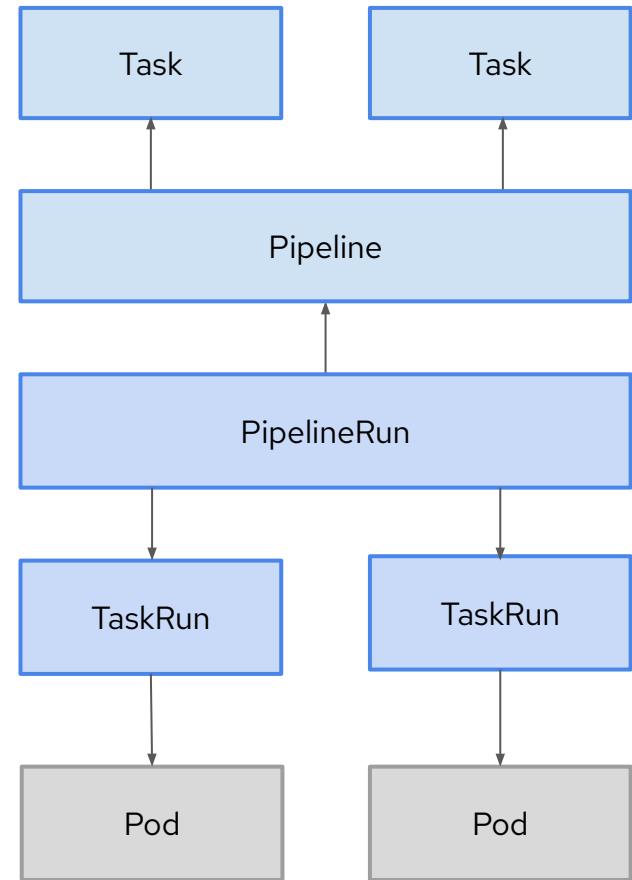
TaskRunによってTaskはPodとして動作し、処理が終わると終了する



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



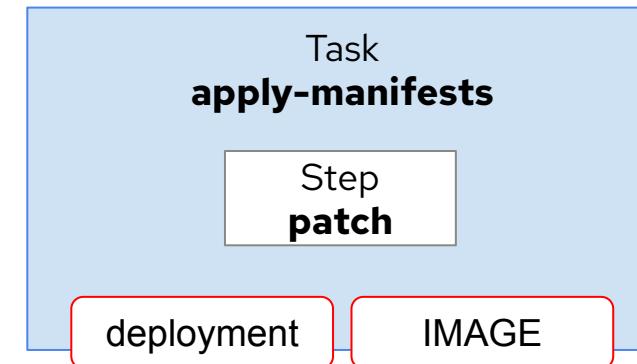
Task定義

`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`

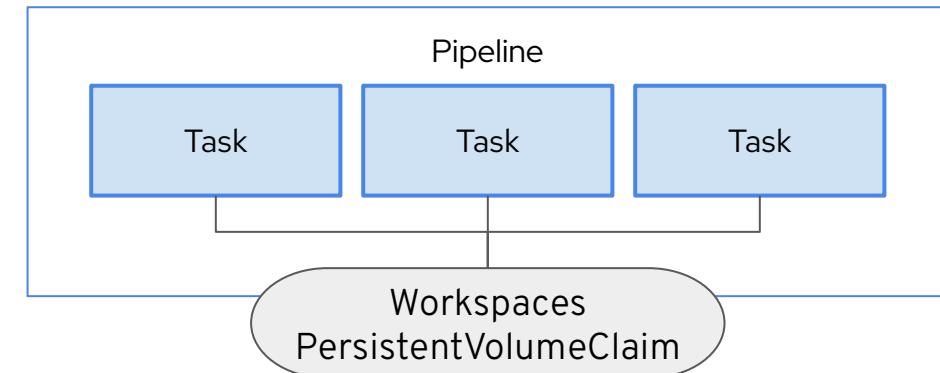
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: source-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

- 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

5

OpenShift GitOps

演習内容

OpenShift GitOps

セルフペースで実施できるハンズオン環境を用いて、OpenShift GitOpsを利用します。Git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、OpenShift GitOpsを利用すると、GitをSSOT(Single Source of Truth)としてアプリケーションのデプロイが自動化されることを学びます。

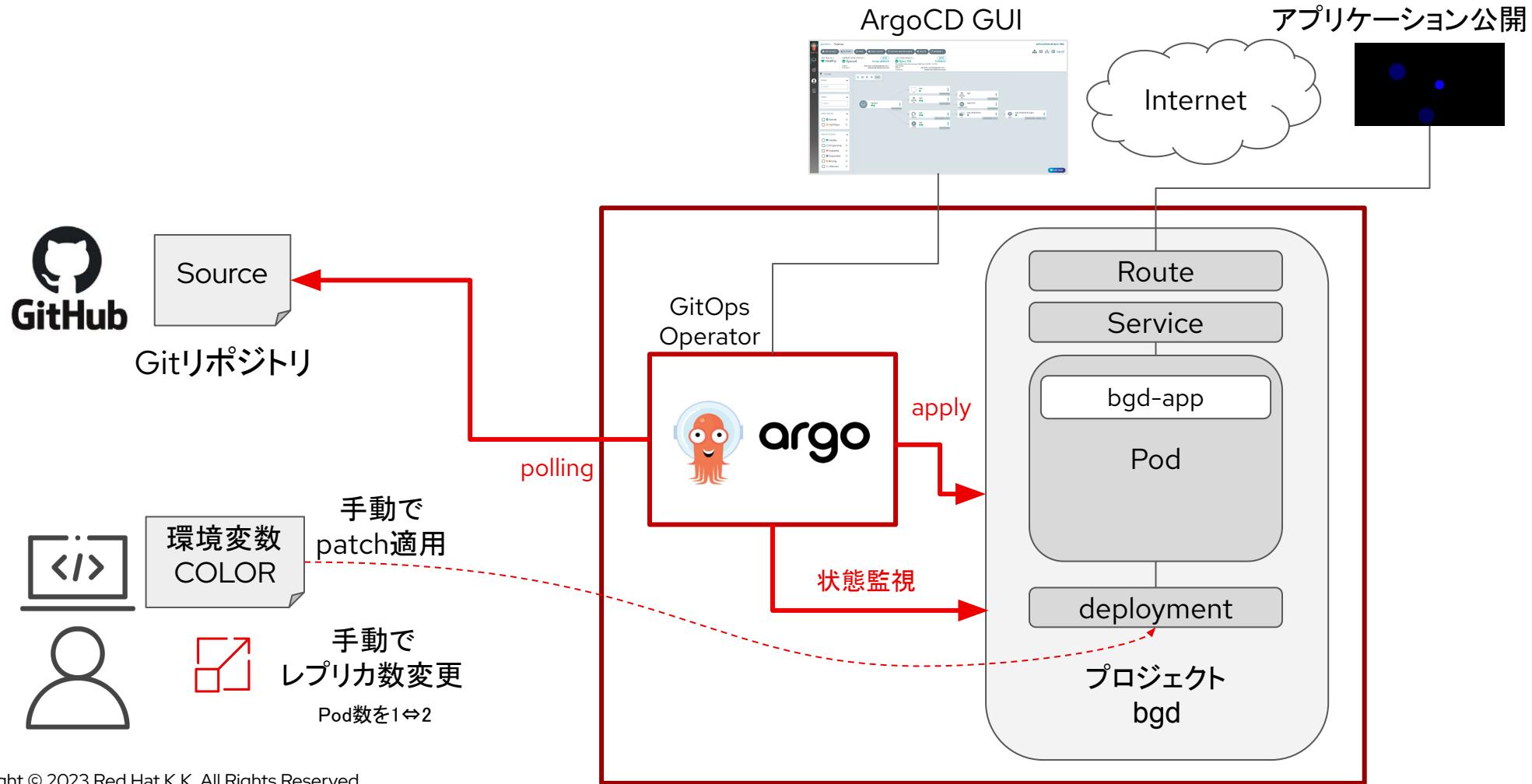


ゴール

- OpenShift GitOps Operatorのインストール方法を学習します
- ArgoCDを利用してアプリケーションがデプロイされることを学習します
- 稼働している状態とソースを比較して差分を検出する動きを学習します
- 差分を手動で同期する方法を学習します
- 差分を検出した際に、自動で修正する動きを学習します

OpenShift GitOps

↑クリックするとハンズオン環境へアクセスできます



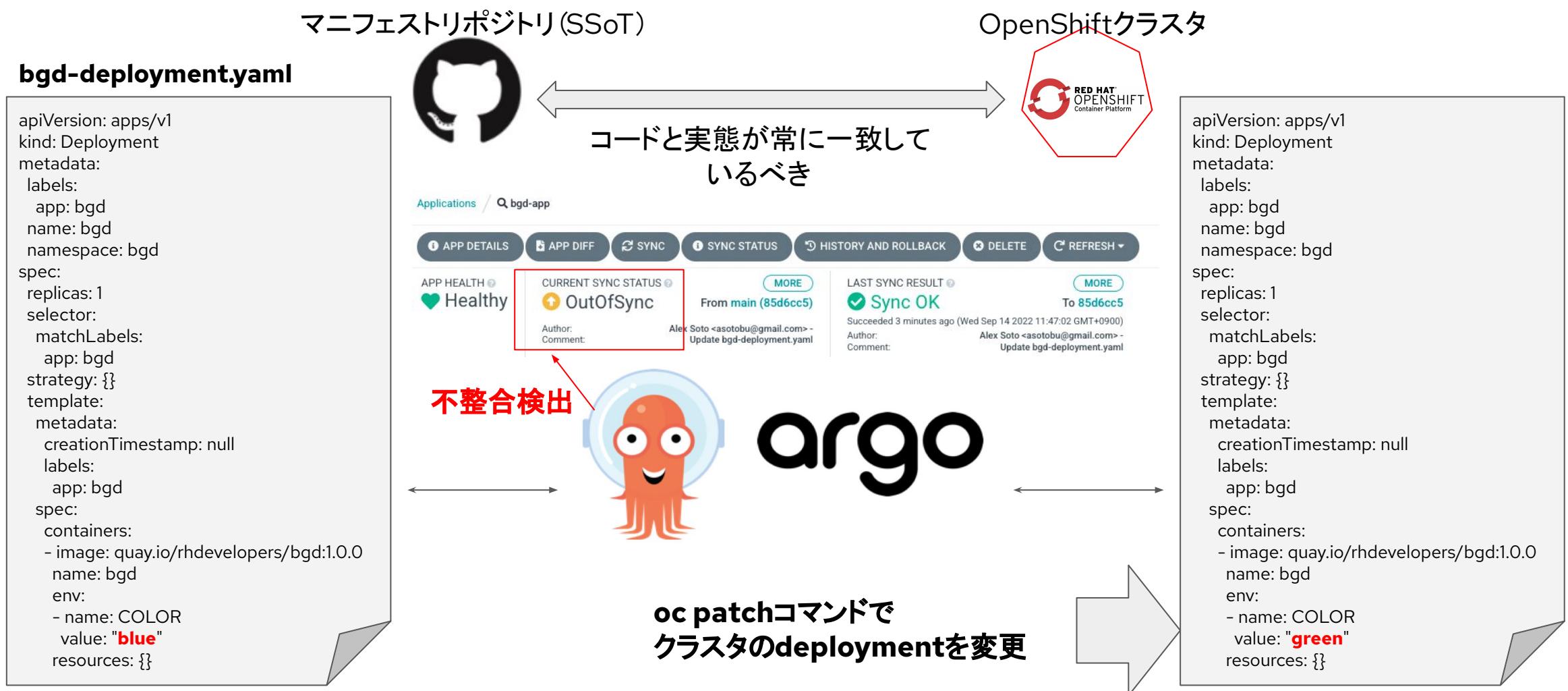
OpenShift GitOps

↑クリックするとハンズオン環境へアクセスできます

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

6

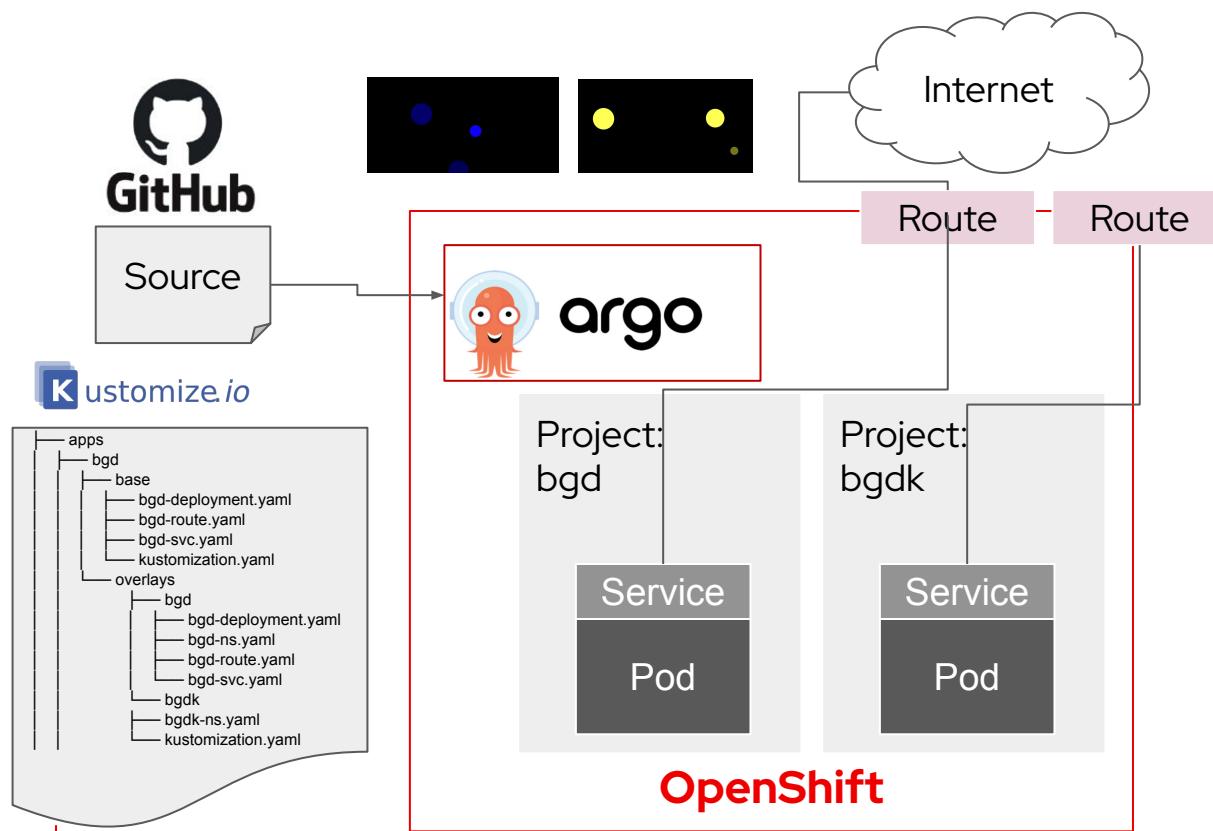
OpenShift GitOps

- Working with Kustomize
- Working with Helm

Working with Kustomize

↑クリックするとハンズオン環境へアクセスできます

システムイメージ



演習内容

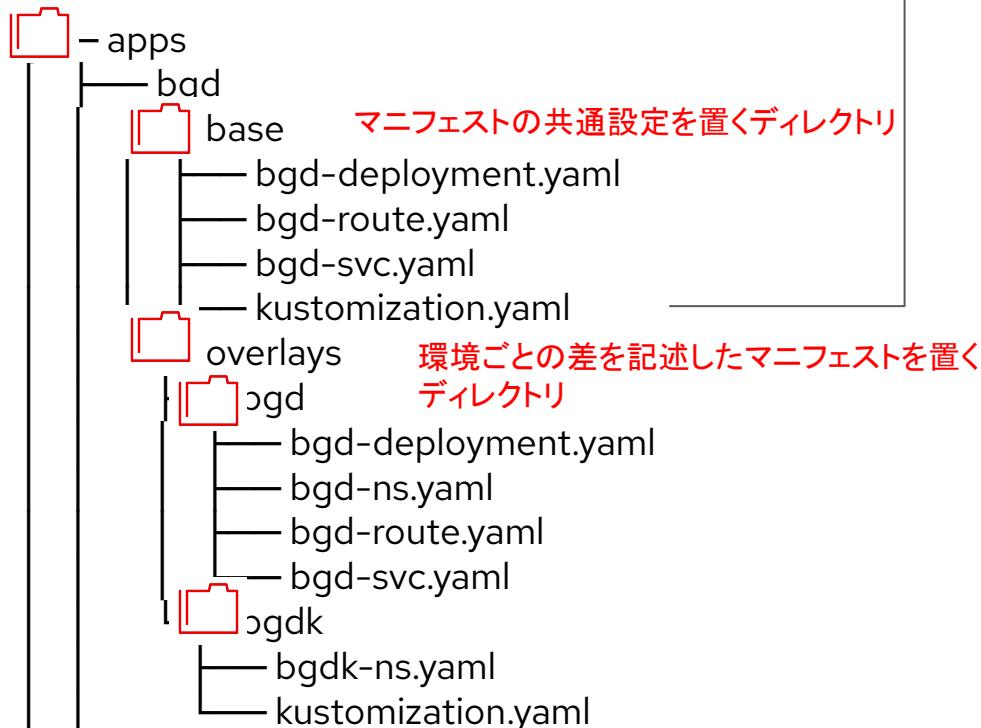
- kustomization.yamlファイルについて
- kustomizeの動作を確認します
- GitOps Operatorをインストールします
- GitにあるArgoCD用マニフェストを利用してアプリケーションをデプロイします
- kustomizeのbaseとoverlayの仕組みについて学びます
- kustomizeを利用している別のアプリケーションをデプロイします

ゴール

- kustomizeの動作を体験します

Working with Kustomize

↑クリックするとハンズオン環境へアクセスできます

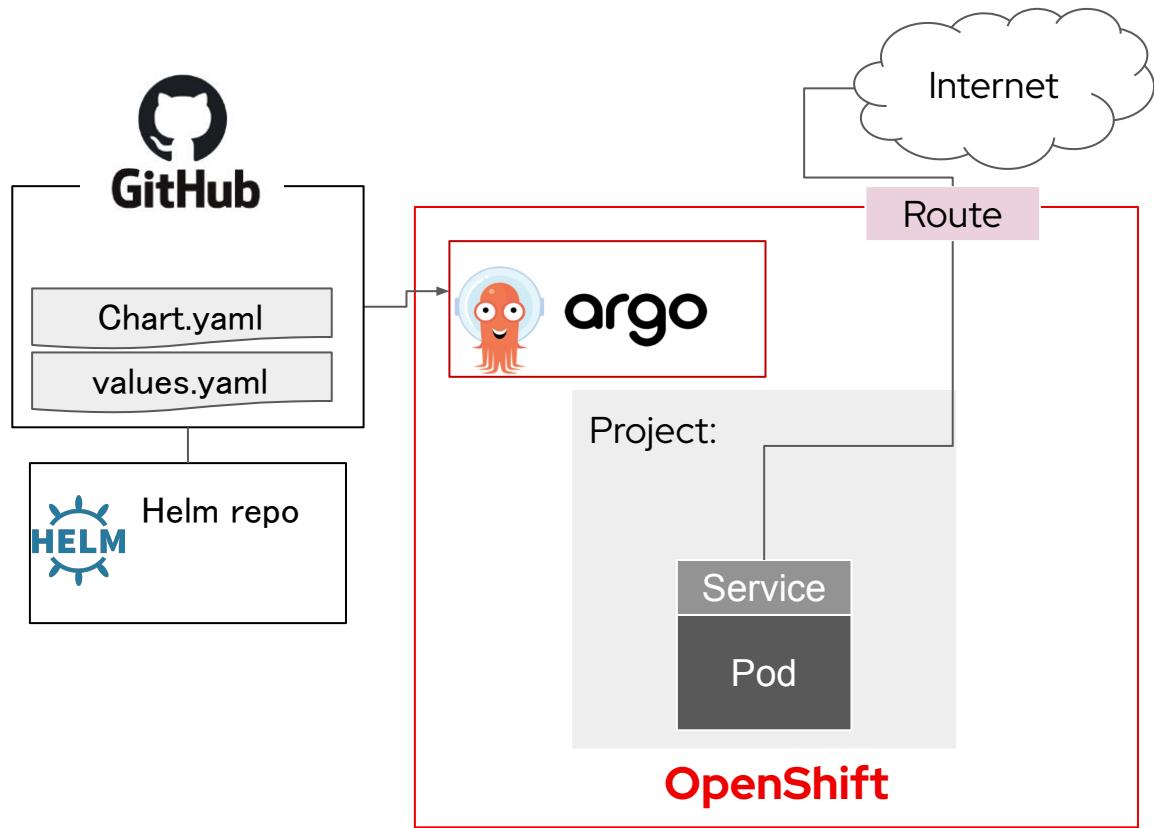


```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: bgd
resources:
- bgd-svc.yaml
- bgd-deployment.yaml
- bgd-route.yaml
```

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: bgdk  namespaceをbgdkに上書きする
resources:
- ././base      元となるkustomization.yamlがあるディレクトリ
- bgdk-ns.yaml 使用するリソース
patchesJson6902:
- target:
  version: v1
  group: apps
  kind: Deployment
  name: bgd
  namespace: bgdk
  patch: |-  patch適用してenvの値をyellowに変更
  - op: replace
  path: /spec/template/spec/containers/0/env/0/value
  value: yellow
```

Working with Helm

システムイメージ



演習内容

- GitOps Operatorをインストールします
- ArgoCDを使ってHelmチャートをネイティブにデプロイします
- `value.yaml`と`Chart.yaml`を使ってデプロイの状態をgitリポジトリに保存します
- ArgoCDを使ってgitアプリをデプロイするのと同様に、helmチャートをデプロイします

ゴール

- ArgoCDとHelmの連携によるアプリのデプロイを体験します

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/OpenShift1220>



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。

70

Copyright © 2023 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。

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