

# Red Hat OpenShift Platform ハンズオンワークショップ ～Pipelines/GitOps体験編～

# Red Hat OpenShift Platform ハンズオンワークショップ ～Pipelines/GitOps体験編～

- 本日のガイド:<https://red.ht/OCPhands-on-guide>
- Meet上での名前欄に  
「名前(フルネーム)」のように変更お願いします

# アジェンダ

	項目	内容	時間
1	OpenShift CI/CD概要	OpenShift CI/CDの概要を座学で学んでいただきます。	20分
2	OpenShift/ハンズオン入門	ハンズオン環境の利用方法および内容についてご説明いたします。 セルフペースでハンズオンを実施していただきます。	100分
2-1	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。	
2-2	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を同じ状態に保つArgoCDの動作を体験します。	
	イメージを利用したアプリケーションのデプロイ（※オプション）	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（※オプション）	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。	
	OpenShiftにおける永続ボリュームの利用（※オプション）	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。	
3	Podman入門	次世代コンテナランタイムPodmanを学習します。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。	
			約 2時間

(※オプション)コースは興味があれば実施してください

# 本日の講師紹介



輿水  
こしみず



田中  
たなか



斎藤  
さいとう

**New Course !!!**



# Podman 入門



Podman Special Event  
**Today's Speaker / TALKs**



「Kubernetes in Rootless Podman」 by 須田さん

「Podman最新情報 Fall 2023」 by 織さん

「PodmanとWasm 概要」 by うたもくさん

「Podman で実現する手軽な CI/CD 環境」

by Naoさん

OpenShift テックコミュニティ

# OpenShift Lounge+

<https://openshift.connpass.com/>



「Kubernetes / OpenShift / クラウドネイティブ」がテーマ。  
有志で運営するテックコミュニティ。



テックコミュニティ主催 4 Days Handson



OpenShift Slack

<https://bit.ly/openshiftjp-slack>

# QA

- オンライン共有メモを用いての質問をお願いします
  - <https://red.ht/OCPhandson-guide>
- オンラインイベントとなりますので、「マイクはミュート」でお願いいたします

# なぜCI/CDに取り組むべきなのか？

Why CI/CD is so important to work on?

# 多くの企業の CI/CD取り組み状況

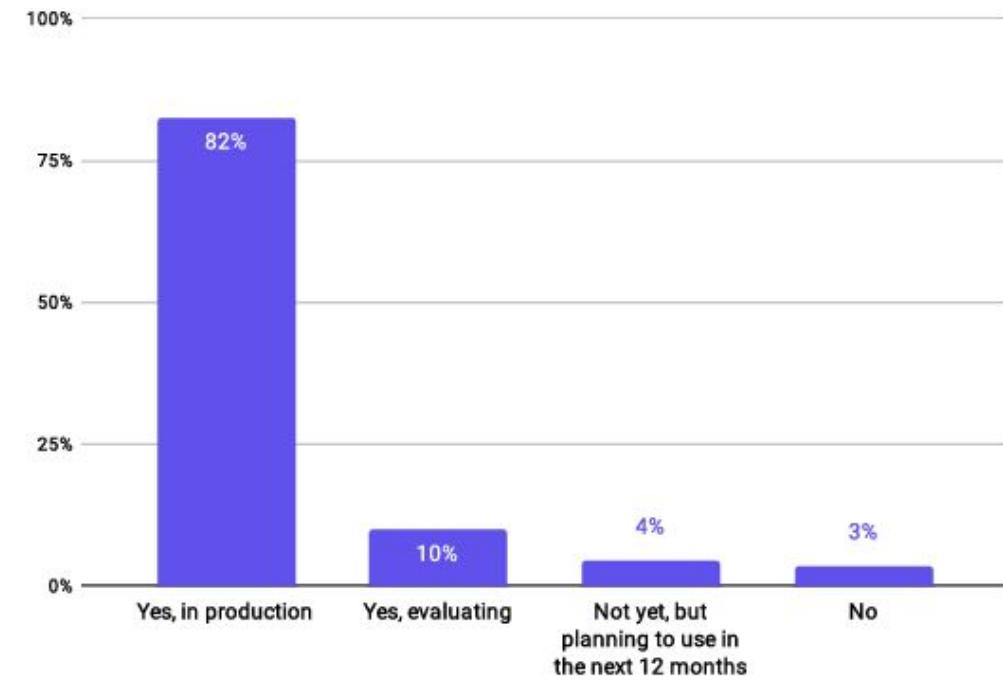
## CI/CDパイプラインはあることが当たり前に



CNCF(Cloud Native Computing Foundation)がユーザーCommunityで実施した調査によると、8割を超える企業がCI/CDパイプラインを商用として稼働させている



Do you run Continuous Integration / Continuous Development (CI/CD) pipelines?



CNCF SURVEY 2020

[https://www.cncf.io/wp-content/uploads/2020/11/CNCF\\_Survey\\_Report\\_2020.pdf](https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf)

# なぜCI/CDパイプラインを利用するのか？

アプリケーションの開発サイクルを短縮し素早くビジネス価値を創出

CI/CDパイプラインはこれまで手作業で行っていた多くの開発作業や、  
アプリケーションのデプロイプロセスを自動化します。

## Continuous Integration

- アプリケーションのビルド
- 単体/結合テスト
- セキュリティ脆弱性の検知
- コンテナイメージの作成

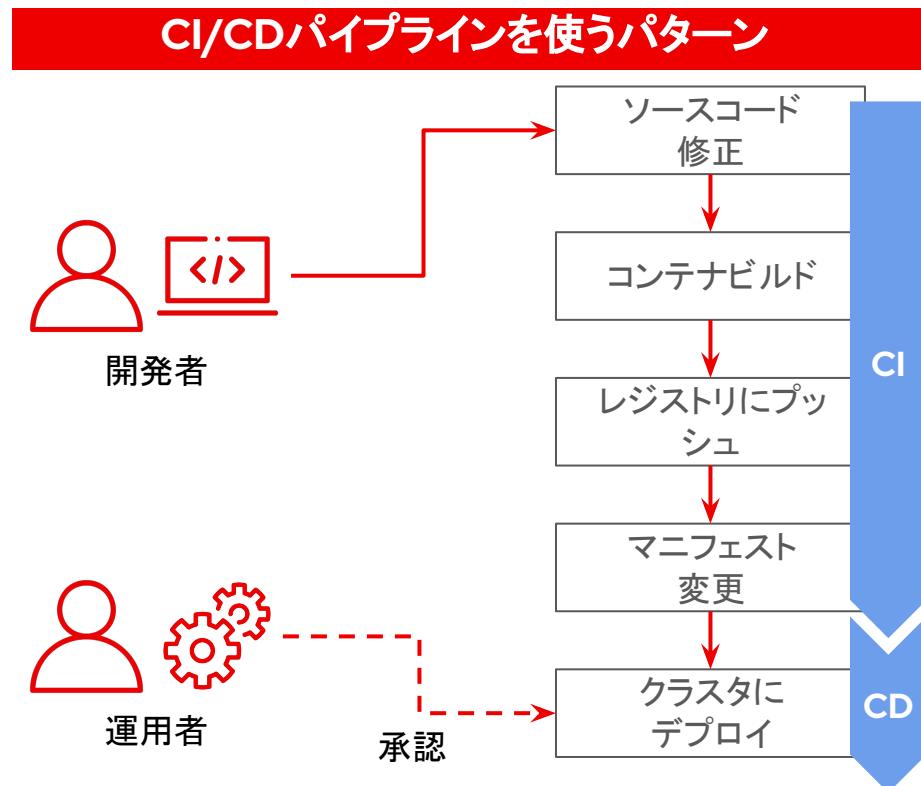
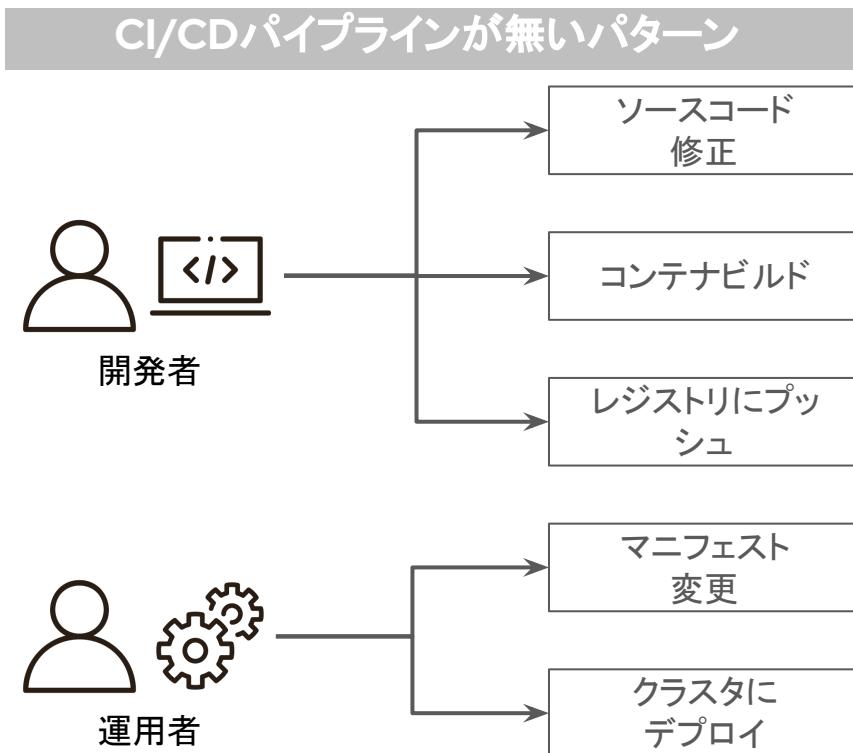
## Continuous Delivery

- 開発/商用環境へのデプロイ
- 環境の監視、ロールバック
- 高度なリリース戦略の実現  
(カナリアリリース、BlueGreen)

# コンテナとCI/CD

## コンテナの価値を享受するには CI/CDが不可欠となる

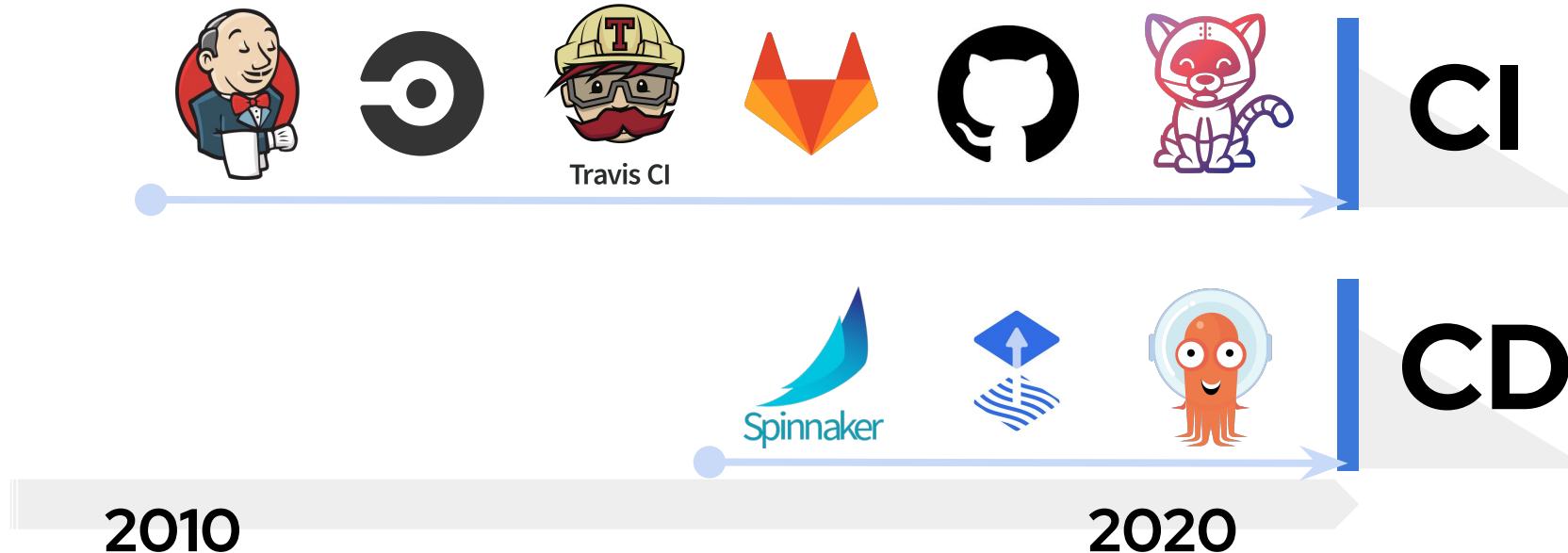
コンテナ化されたシステムの場合、システムに変更を加えるためには新たなコンテナイメージを作成し、それをデプロイする必要があります。CI/CDパイプラインはこれらの作業を自動化します。



# CI/CDの進化

## CIとCDの区別が明確化

以前はCIとCDの区別は今ほど明確ではなく、CIツールを使ってデプロイまで実施することが多くありました。しかし昨今ではより高度なCDを可能とする専用ツールが登場し、それぞれ別のツールで実施することが一般的になりつつあります。



# モダンなCI/CDツールを使うメリット

## 過去のツールの様々な制約 / 課題を解消

Jenkinsなど以前から存在しているツールはツール自体に慣れている人が多くいる一方で、運用が属人化しやすいなどの課題が存在していました。昨今のCI/CDツールはこれらの課題を解決しており、より利用し易いものとなっています。



### Traditional CI/CD

- ・CIとCDを一つのツールで実行
- ・専用のサーバーを構築/運用
- ・プラグインの依存関係の解消
- ・並列ジョブ実行のためのslaveサーバーの追加設定
- ・デプロイのロールバックは自前で用意



### Cloud Native CI/CD

- ・CIとCDをそれぞれのツールで実行
- ・Operatorによるインストール/管理
- ・ジョブは独立したコンテナで実行されるため依存関係の考慮不要
- ・オンデマンドなスケーリング
- ・デプロイのロールバックはデフォルトで準備



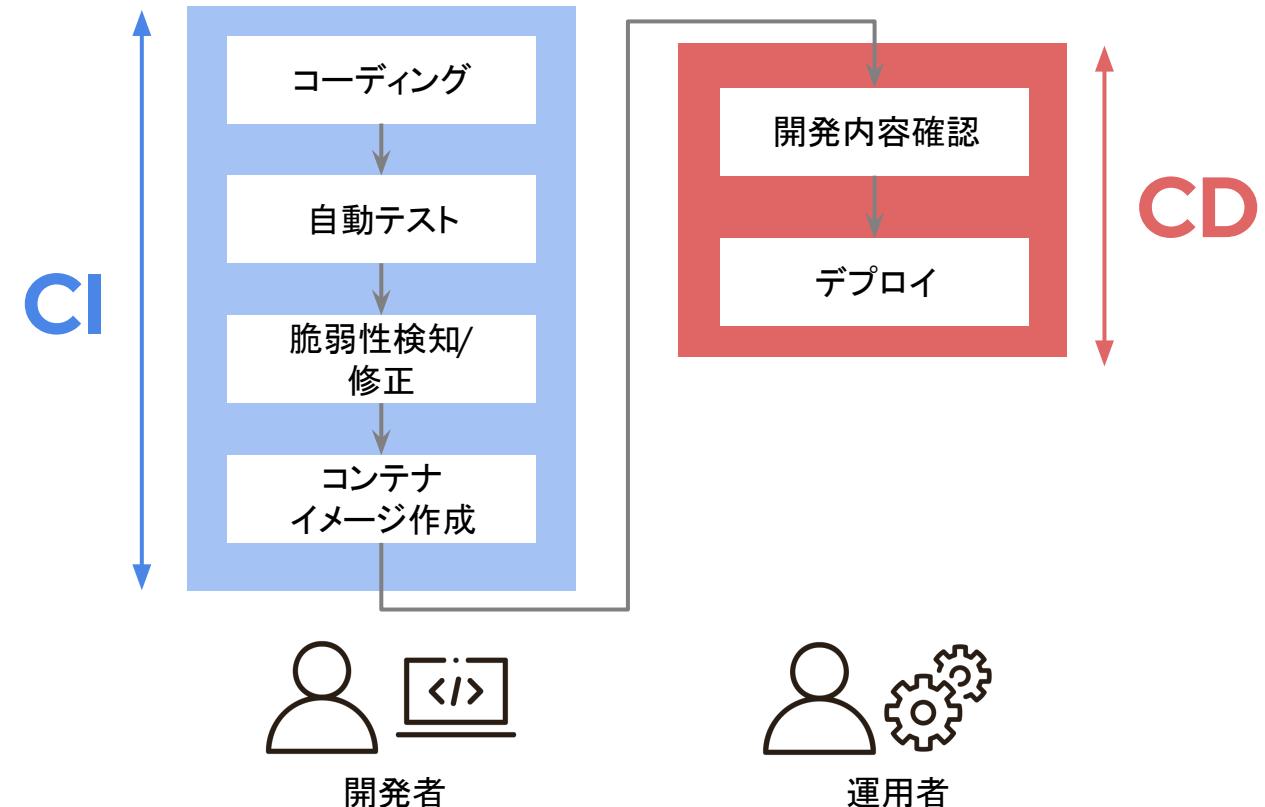
# CIとCDの分離がなぜ重要なのか？

## チーム間の責任分界点を明確化する

CIとCDを別々のツールを使って実施することで、役割分担が明確化し、各々のタスクに集中することが可能となります

- ・デプロイ可能な成果物を作成するまで  
-> 開発チームの役割

- ・成果物をデプロイして安定運用するまで  
-> 運用チームの役割



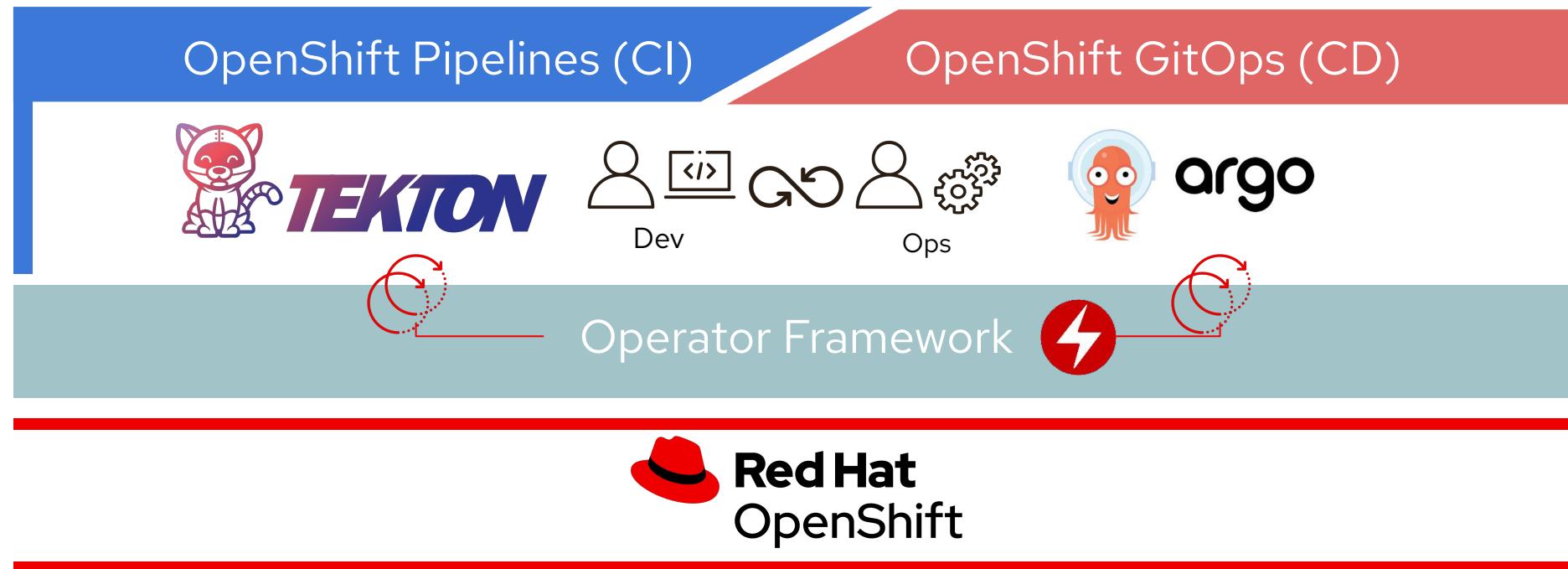
# OpenShiftにおける CI/CDとは？

What can we do by CI/CD on OpenShift?

# 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。



# 標準部品を使ったパイプラインの構築

## Tekton HubからTaskをインストール

Tekton ProjectではCIパイプラインの中で実行される一般的なTaskをTekton Hubにて公開しています。これらのTaskを組み合わせることで最低限の労力でCIパイプラインを構築することが可能となります。

### 公開されているTaskの例:

- maven: Javaのビルド、テスト、など
- conftest: マニフェストファイルのテスト
- golang test: Go言語の単体テスト
- pytest: Pythonの単体テスト
- sonarqube scanner: アプリケーション静的診断
- Send message to Slack Channel: Slackへの通知

The screenshot shows the Tekton Hub interface with a dark theme. At the top, there's a search bar with 'test' and a login button. Below the header, it says 'Welcome to Tekton Hub' and 'Discover, search and share reusable Tasks and Pipelines'. On the left, there are filtering options for 'Sort By', 'Kind' (Task or Pipeline), 'Catalog' (Tekton), and 'Category' (Automation, Build Tools, Cli, Cloud, Code Quality, Continuous Integration, Deployment, Developer Tools, Git, Image Build, Integration & Delivery, Kubernetes, Messaging, Monitoring, Networking, Openshift, Publishing, Security, Storage). The main area displays a grid of task cards:

Task Name	Version	Rating	Description
kubeval	v0.1	★ 0	This task makes it possible to use Kubeval within your Tekton pipelines. Kubeval is a tool used for validating Kubernetes configuration files. By default the task will recursively scan...
conftest	v0.1	★ 1	These tasks make it possible to use Conftest within your Tekton pipelines. Conftest is a tool for testing configuration files using Open Policy Agent.
golang test	v0.2	★ 0	This Task is Golang task to test Go projects.
helm conftest	v0.1	★ 0	These tasks make it possible to use Conftest within your Tekton pipelines. Conftest is a tool for testing configuration files using Open Policy Agent.
GKE Cluster Create	v0.1	★ 0	Create a GKE cluster. This Task can be used to create a GKE cluster in a GCP project and fetch a kubeconfig that can be used (in a context with both kubectl and gcloud available) to ma...
pytest	v0.1	★ 0	This task will run pytest on the provided input.
TypeScript linter	v0.1	★ 0	This task can be used to perform lint check on TypeScript files
Tekton Operator Install	v0.1	★ 0	This task can be used to install Tekton pipelines and also its components using Tekton Operator on a new cluster.
boskos-acquire	v0.1	★ 0	Acquire a project using Boskos. The boskos-acquire Task will request a resource of the specified type from the server-url. If successful, it will start a pod that...
boskos-release	v0.1	★ 0	Release a project acquired using Boskos. The boskos-release Task will release the specified resource from the boskos instance at server-url. It also assumes the...

# OpenShiftコンソールとの統合

## GUIによる直感的なパイプライン構築

TektonはOpenShiftのコンソール画面と完全に統合されており、開発者が利用する画面からパイプラインを構築し、実行状況を確認/管理することができます。

The image displays three screenshots of the OpenShift Pipelines interface:

- Top Left:** Shows the "Pipelines" page for the "tekton-demo" project. It features a sidebar with "Developer" and "Pipelines" sections, and a main area with a "Create Pipeline" button and a "No Pipelines Found" message.
- Bottom Left:** Shows the "Advanced" view of the Pipelines page, which includes a "Pipelines" section and a "Logs" section.
- Right:** Shows the "Pipeline Runs" page for the "a1-cicd" project, specifically for the run "petclinic-deploy-dev-run-qwkx4". It indicates the run was successful ("Succeeded"). Below it is the "Pipeline Run Overview" which shows a sequence of pipeline steps: unit-tests, release-app, build-image, deploy, int-test, code-analysis, generate-r..., and perf-test. All steps are marked as completed with green checkmarks.

# 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

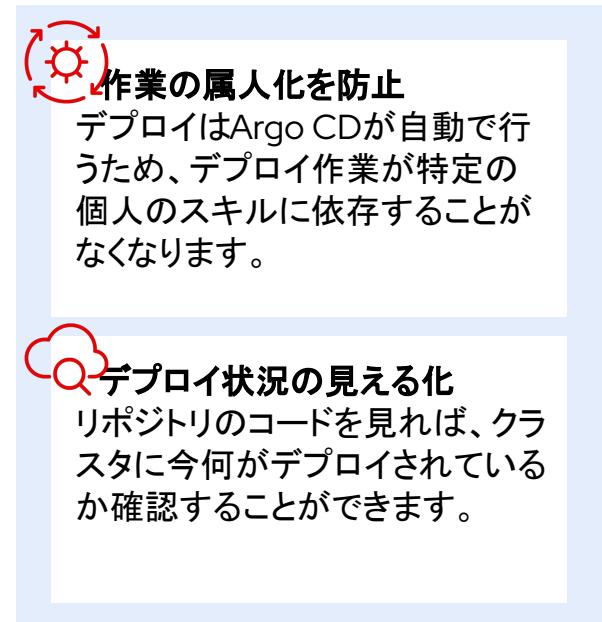
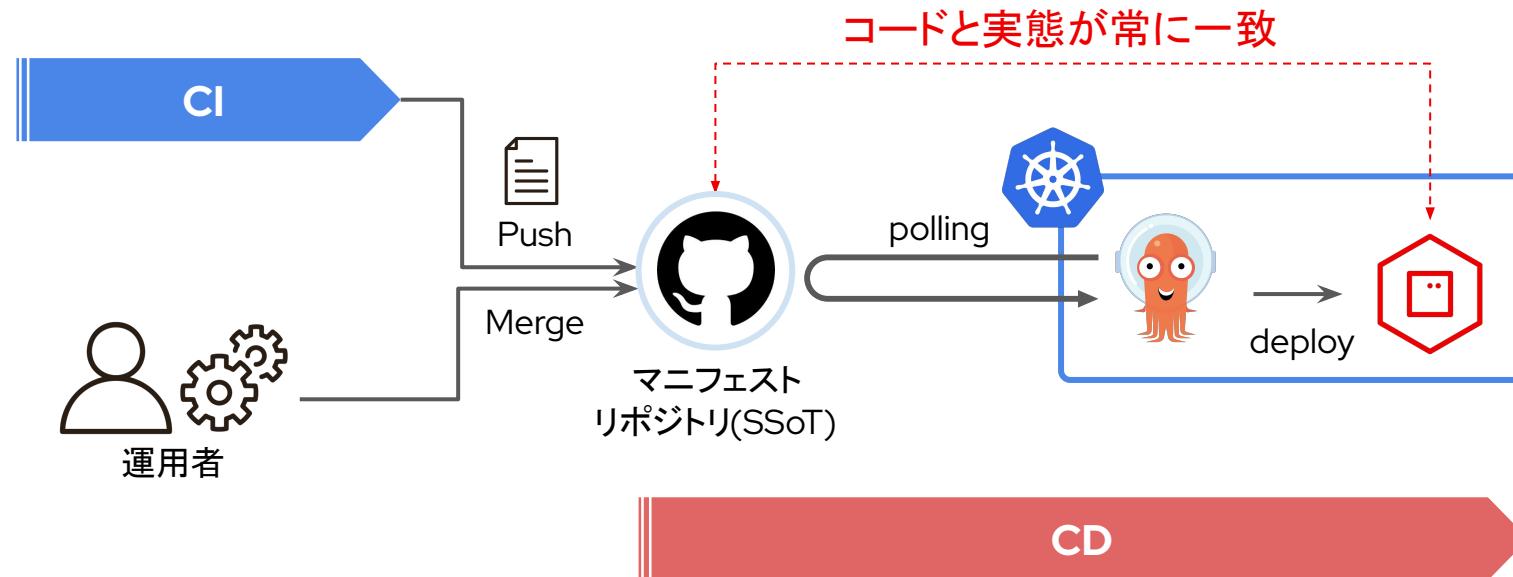
TektonのCIパイプラインはコンテナとして実行されます。開発者のアクション(Gitリポジトリへの操作)を起点とし、必要なタイミングでコンテナを立ち上げるため、クラスタのリソースを有効活用できます。



# 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。



# Argo CDコンソール

## デプロイされたアプリケーションの状況を可視化

Argo CDコンソールでは現在デプロイされているアプリケーションの状態をトポロジービューとして確認できます。また特定の過去バージョンへのロールバックなどについてもコンソールから実施できます。

The screenshot shows the Argo CD application topology view for the 'app-develop' application. The interface includes a top navigation bar with tabs for APP DETAILS, APP DIFF, SYNC, SYNC STATUS, HISTORY AND ROLLBACK, DELETE, and REFRESH. On the left, there's a sidebar with filters for KINDS (KINDS), SYNC STATUS (Synced, OutOfSync), and HEALTH STATUS (Healthy, Progressing, Degraded, Suspended, Missing, Unknown). The main area displays a topological graph of application components:

- app-develop** (Deployment): Synced, 13 minutes ago.
- health-record** (Service): Synced, 13 minutes ago.
- deploy** (Deployment): Synced, 13 minutes ago, rev:4.
- route** (Route): Synced, 13 minutes ago.
- ep** (EndpointSlice): Synced, 13 minutes ago.
- es** (Endpointslice): Synced, 13 minutes ago.
- rs** (ReplicaSet): Synced, 13 minutes ago, rev:3.
- health-record-564f466dc6** (Pod): Synced, 13 minutes ago, rev:3.
- rs** (ReplicaSet): Synced, 5 minutes ago, rev:4.
- health-record-6f5d45b8d5** (Pod): Synced, 5 minutes ago, rev:4.
- pod** (Pod): Synced, 5 minutes ago, running, 1/1.

Each component node shows its status (Synced or OutOfSync) and the time since the last sync. The graph illustrates the dependencies between the application's components.

# Podman

Podmanとは



## Podmanとは

- ▶ コンテナを構築、管理、実行する次世代のコンテナエンジン
- ▶ Open Container Initiative (OCI)を標準フォーマットとして採用
- ▶ Dockerとの互換性(コマンドライン、Dockerイメージ、Compose、Docker互換API、など)
- ▶ Red Hatの開発チームを中心にOSSで開発
- ▶ RHEL 8以降に標準搭載(RHEL 8以降はDocker非サポート)
- ▶ GUIのPodman DesktopもOSSで公開

# Docker互換のコマンドライン

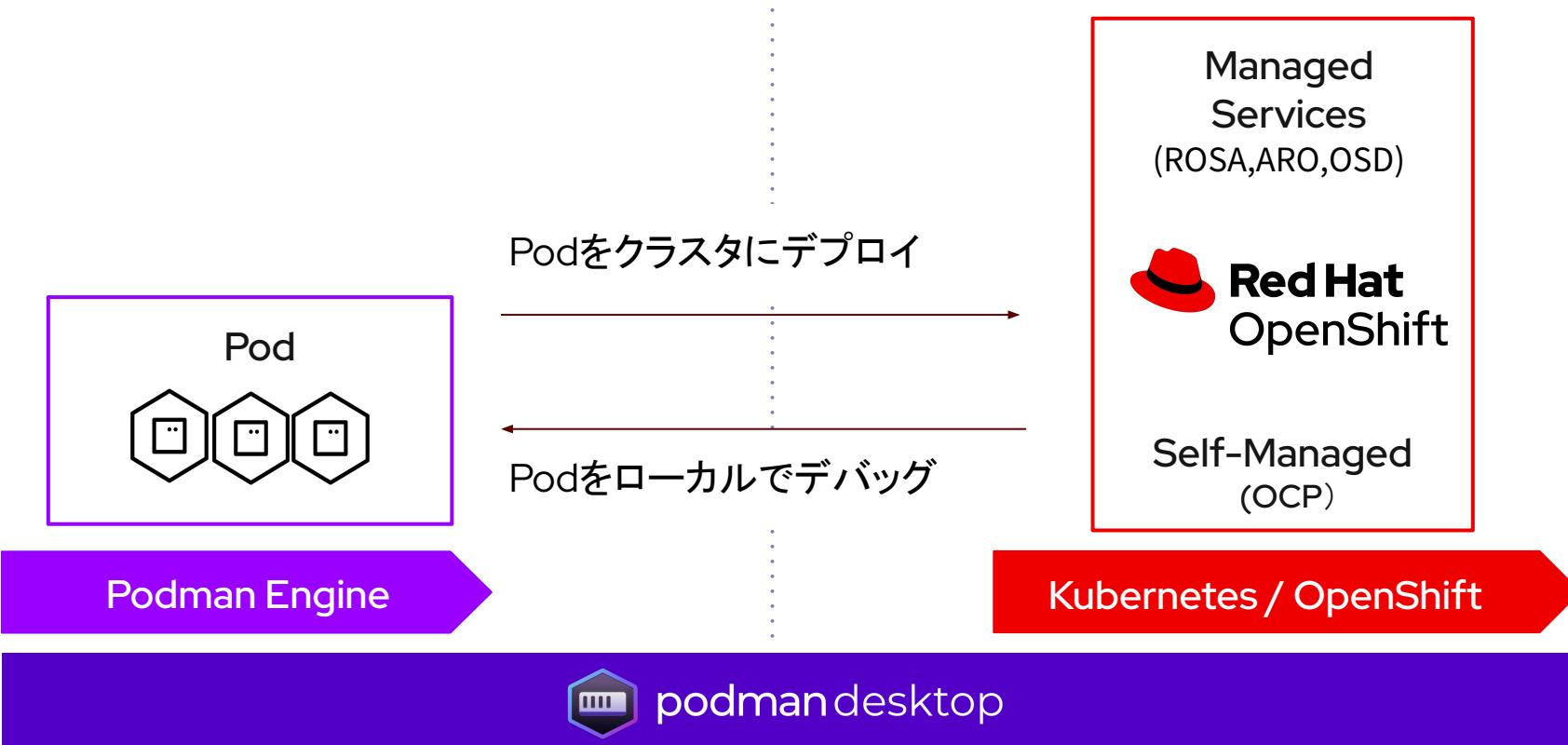
## 代表的なコマンド一覧

- ▶ podman run : コンテナを実行
- ▶ podman build : コンテナイメージをビルド(作成)する
- ▶ podman images : ローカルにあるコンテナイメージをリスト表示する
- ▶ podman rmi : 指定したローカルのコンテナイメージを削除する
- ▶ podman pull : コンテナレジストリからコンテナイメージをプル(取得)する
- ▶ podman push : コンテナレジストリにコンテナイメージをプッシュ(保存)するetc...

Dockerのコマンドをほぼそのまま置き換え可能。  
→Dockerの学習リソースを活用できる。

複数のコンテナをPodで動かす

# Podman PodからKubernetes/OpenShiftへ



# 過去のウェビナー: RHEL 101(Podman編)



## RHEL 101 (Podman編)

次世代コンテナエンジンの基礎から使いこなす方法

2023/12/6

レッドハット株式会社

田中司恩



社内向け資料 [https://docs.google.com/presentation/d/1Ytk3BstHIn\\_8F35QYsmCjVIC1fuqrg4zUv3xh5K-Mls/edit?usp=sharing](https://docs.google.com/presentation/d/1Ytk3BstHIn_8F35QYsmCjVIC1fuqrg4zUv3xh5K-Mls/edit?usp=sharing)

録画(申し込み登録が必要) <https://www.redhat.com/ja/events/webinar/master-next-gen-container-engine>

公開用資料 <https://tracks.redhat.com/l/2023-12-6-rhel-101-p>





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。
<b>OpenShift Pipelines</b>	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。
<b>OpenShift GitOps</b>	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。
<b>Podman 入門</b>	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

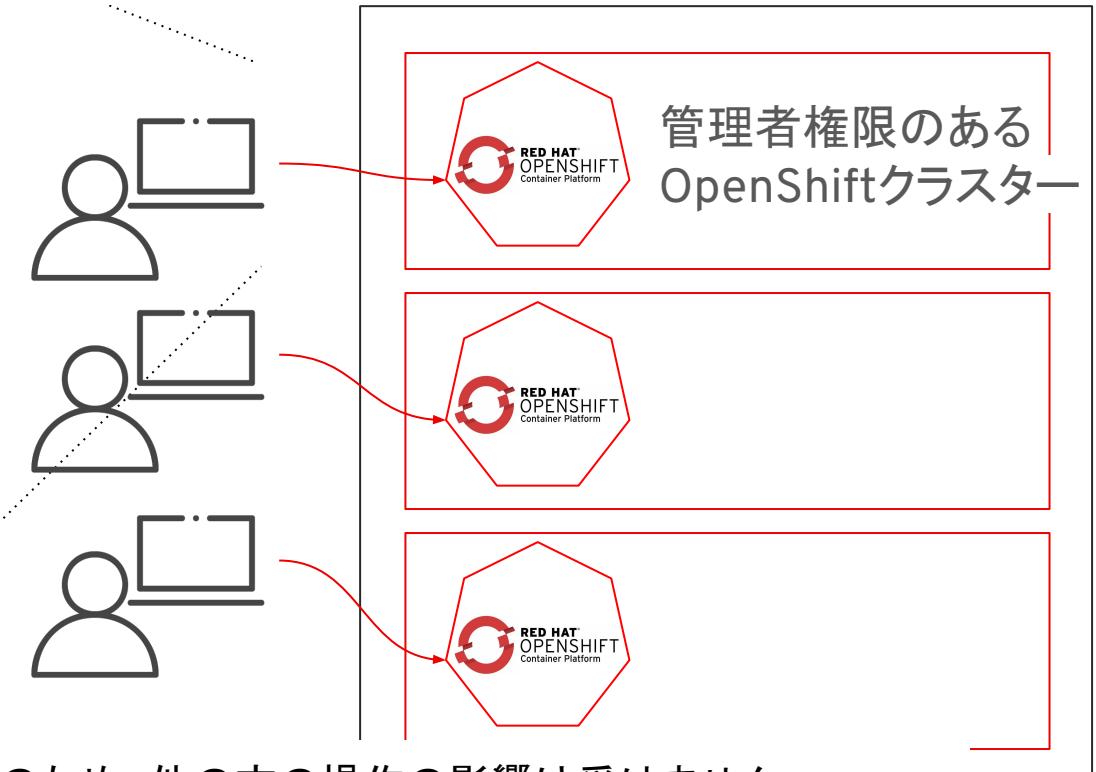
レプリカの数を確認する

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**



# 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

# ハンズオンコース紹介

## OpenShift Pipelines

## OpenShift GitOps

1

---

# OpenShift Pipelines

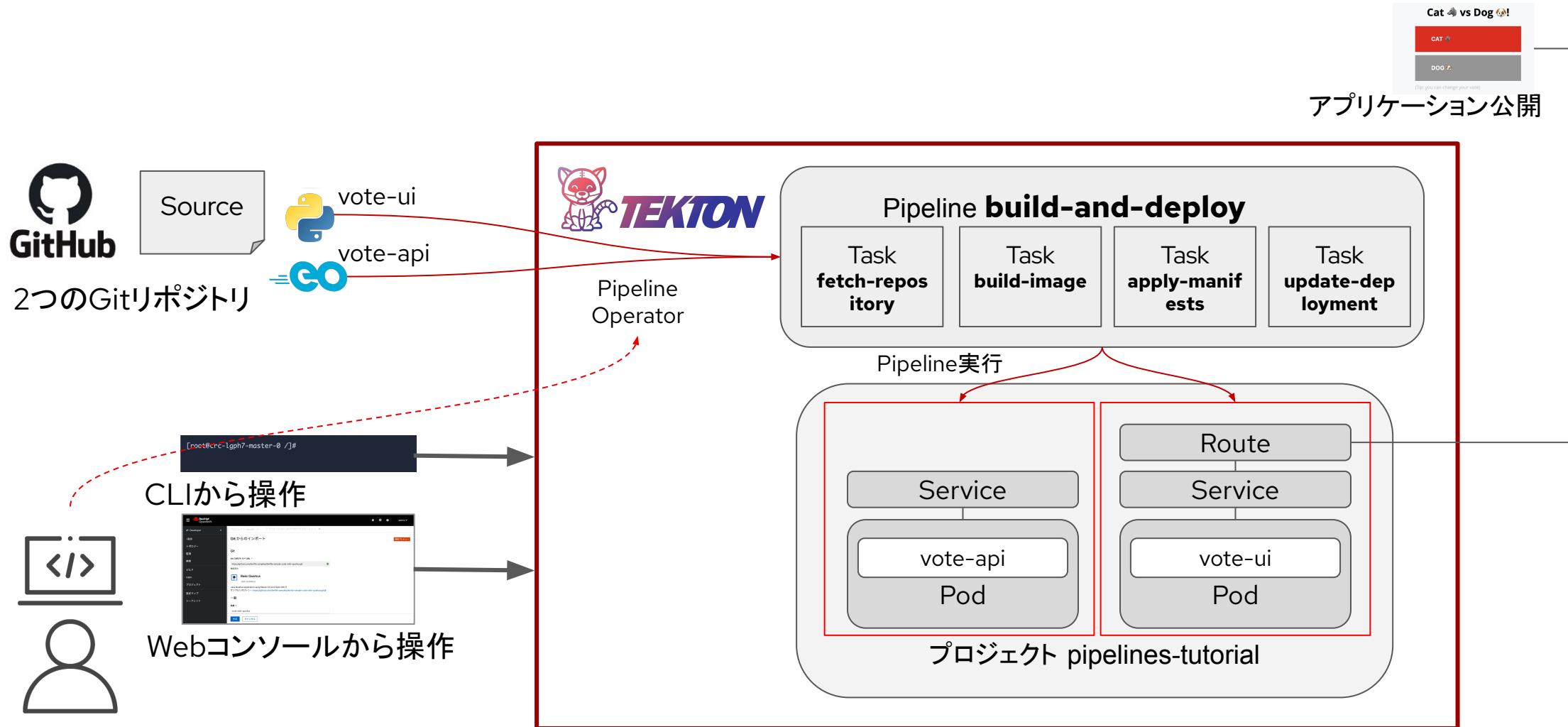
# OpenShift Pipelines

## 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を実行してアプリケーションをデプロイします
- Triggerの設定を行います
- GitHubでPushイベントを発生させ、自動的にPipelineが実行されることを確認します

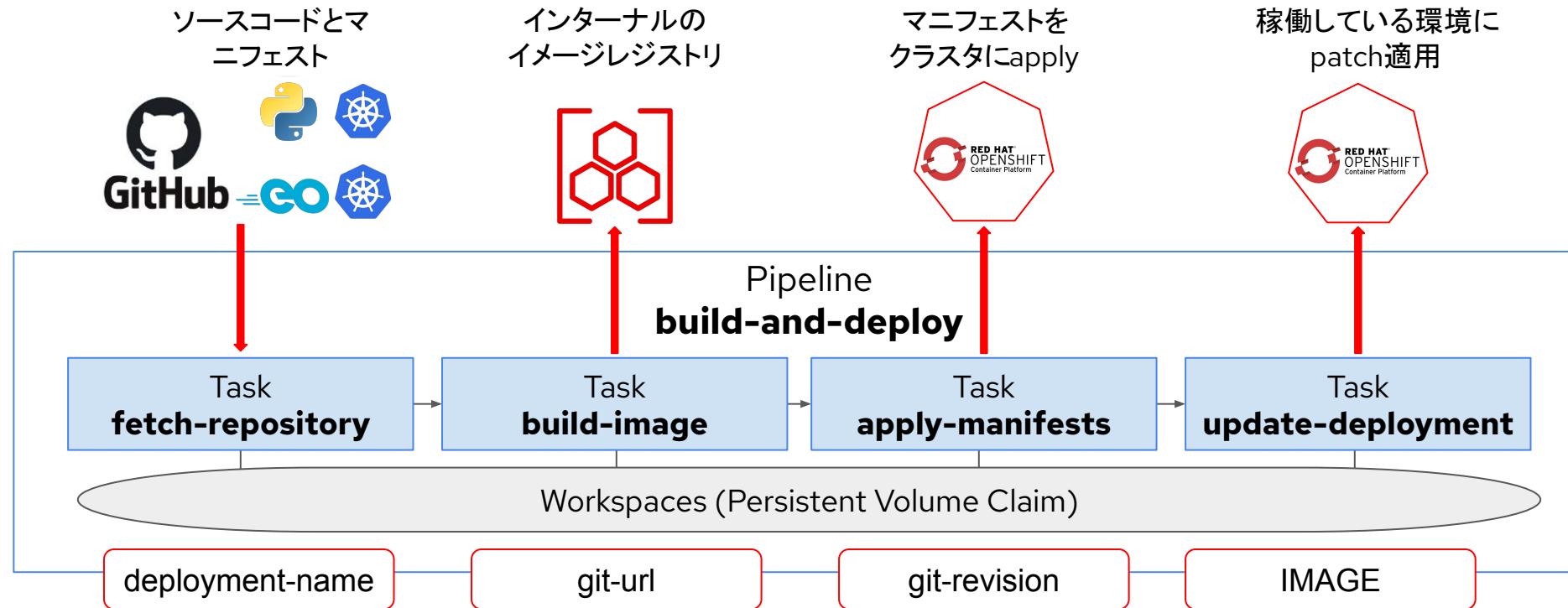
 GitHubアカウントが無い場合は実行できません

# OpenShift Pipelines



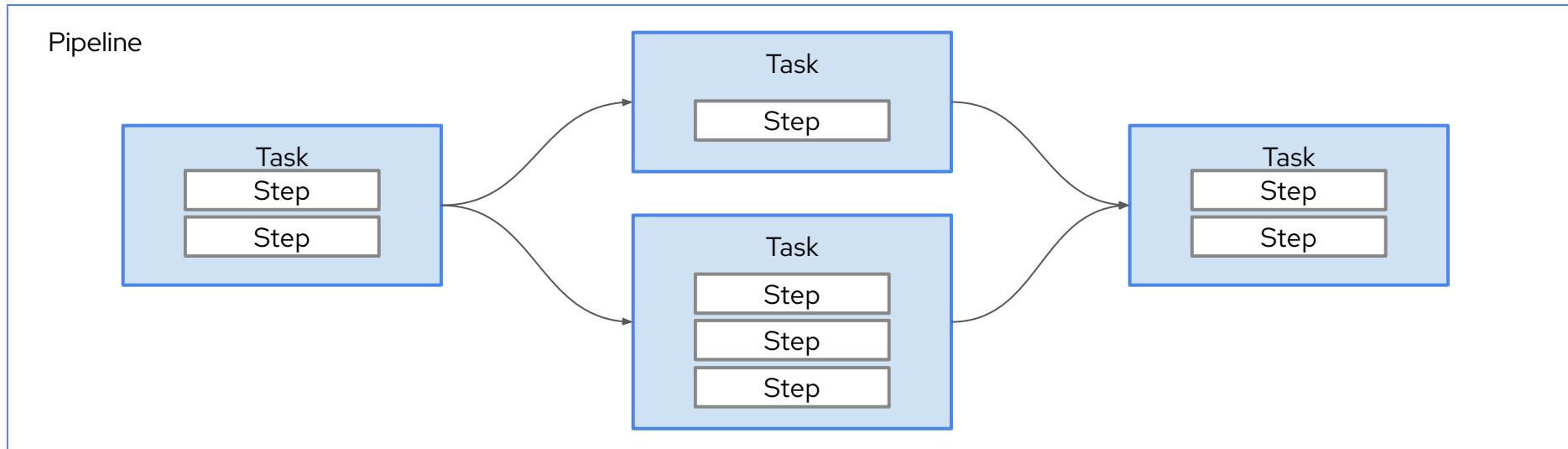
# 作成する Pipeline

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。



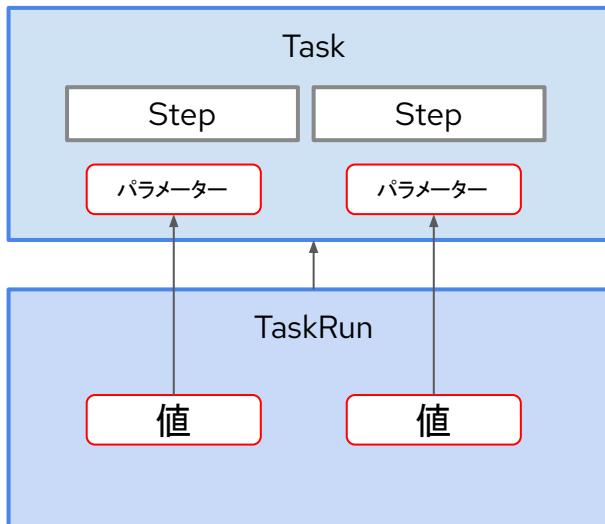
# Tektonのリソース

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。

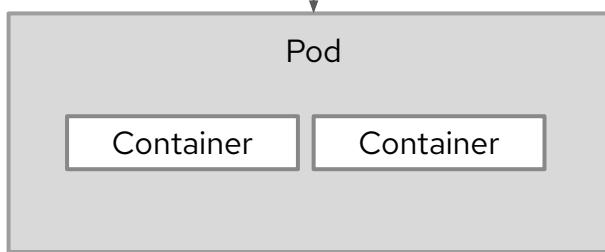


# TaskRunとPipelineRun

## TaskRun



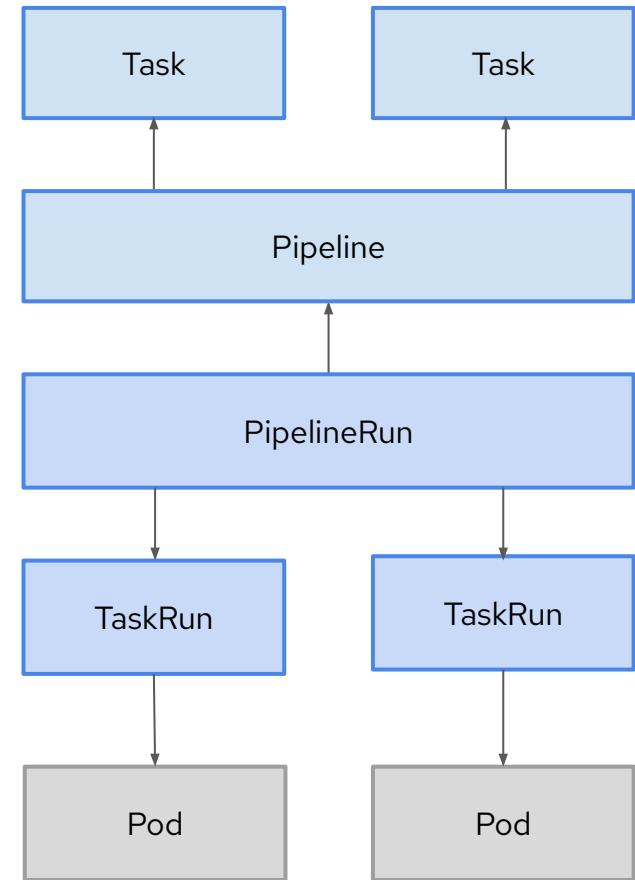
TaskRunによってTaskはPodとして動作し、処理が終わると終了する



### 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 -----
```

- 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

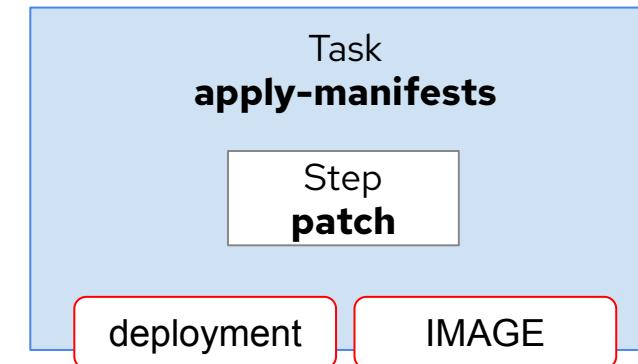


# Task定義

## `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)"
}}]
}}}'
```

- `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

## `persistent_volume_claim.yaml`

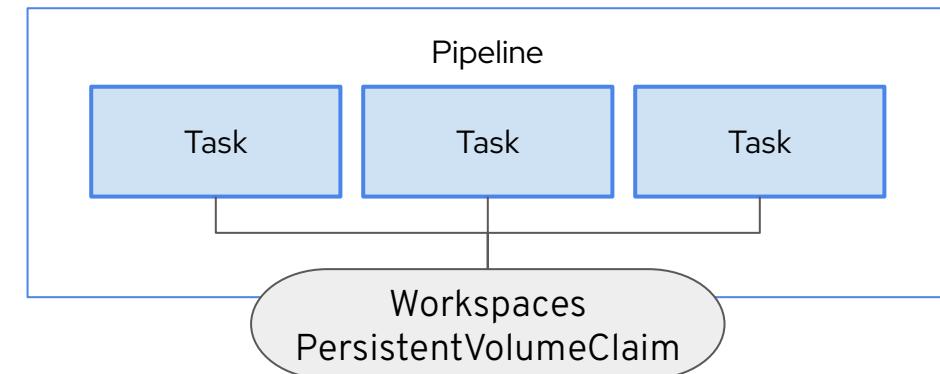
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: source-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

- 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

## **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

# Pipeline定義

## 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

## `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

# PipelineRun定義(pipelines-vote-api)

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  labels:
    tekton.dev/pipeline: build-and-deploy
  generateName: build-and-deploy-run-backendapp-
spec:
  params:
    - name: IMAGE
      value:
image-registry.openshift-image-registry.svc:5000/pipelines
-tutorial/pipelines-vote-api
      - name: deployment-name
        value: pipelines-vote-api
    - name: git-url
      value:
https://github.com/openshift/pipelines-vote-api.git
  pipelineRef:
    name: build-and-deploy
  serviceAccountName: pipeline
  timeout: 1h0m0s
  workspaces:
    - name: shared-workspace
      persistentVolumeClaim:
        claimName: source-pvc
```

## 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/pipelines-vote-api

# PipelineRun定義(pipelines-vote-ui)

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: build-and-deploy-run-frontendapp-
  labels:
    tekton.dev/pipeline: build-and-deploy
spec:
  params:
    - name: IMAGE
      value:
image-registry.openshift-image-registry.svc:5000/pipelines
-tutorial/pipelines-vote-ui
      - name: deployment-name
        value: pipelines-vote-ui
    - name: git-url
      value:
https://github.com/openshift/pipelines-vote-ui.git
  pipelineRef:
    name: build-and-deploy
  serviceAccountName: pipeline
  timeout: 1h0m0s
  workspaces:
    - name: shared-workspace
      persistentVolumeClaim:
        claimName: source-pvc
```

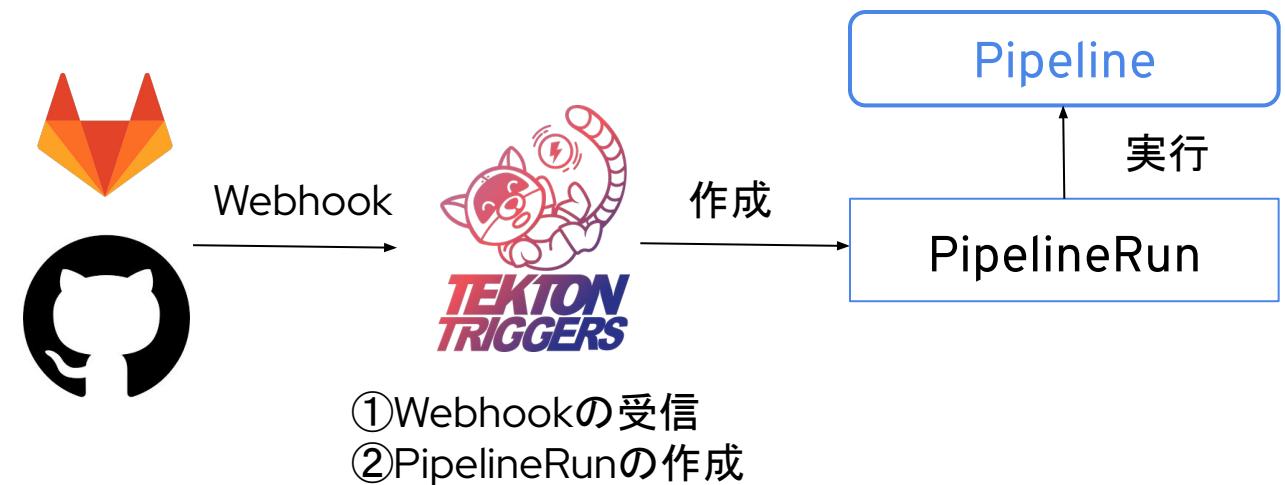
## パラメーター

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/pipelines-vote-ui

# Tekton Triggers

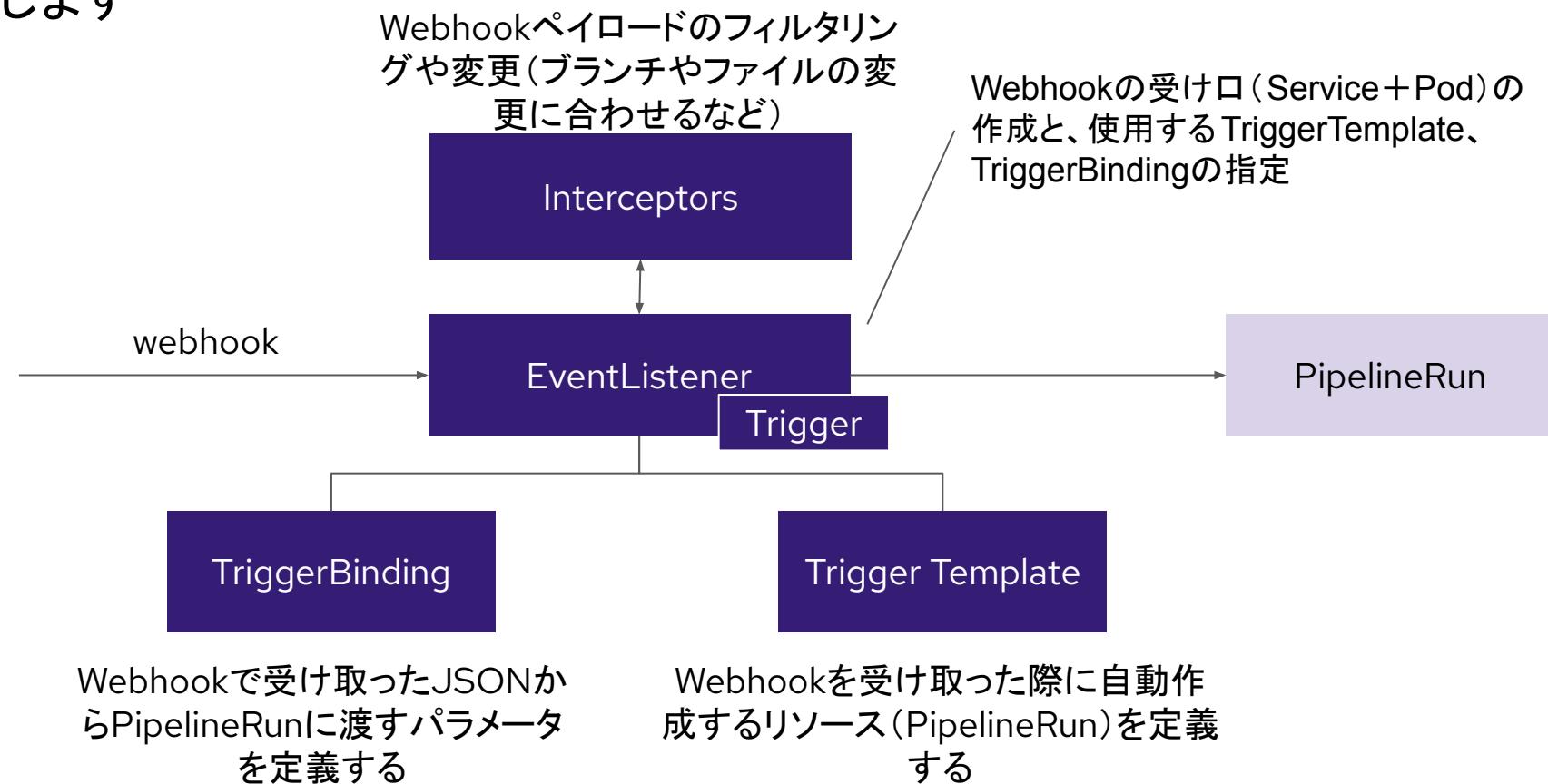
## イベント契機でのPipelineRunの作成

- 開発者がGitリポジトリ上で行うアクションを契機に送信されるWebhookを受信
- PipelineRunを自動で作成

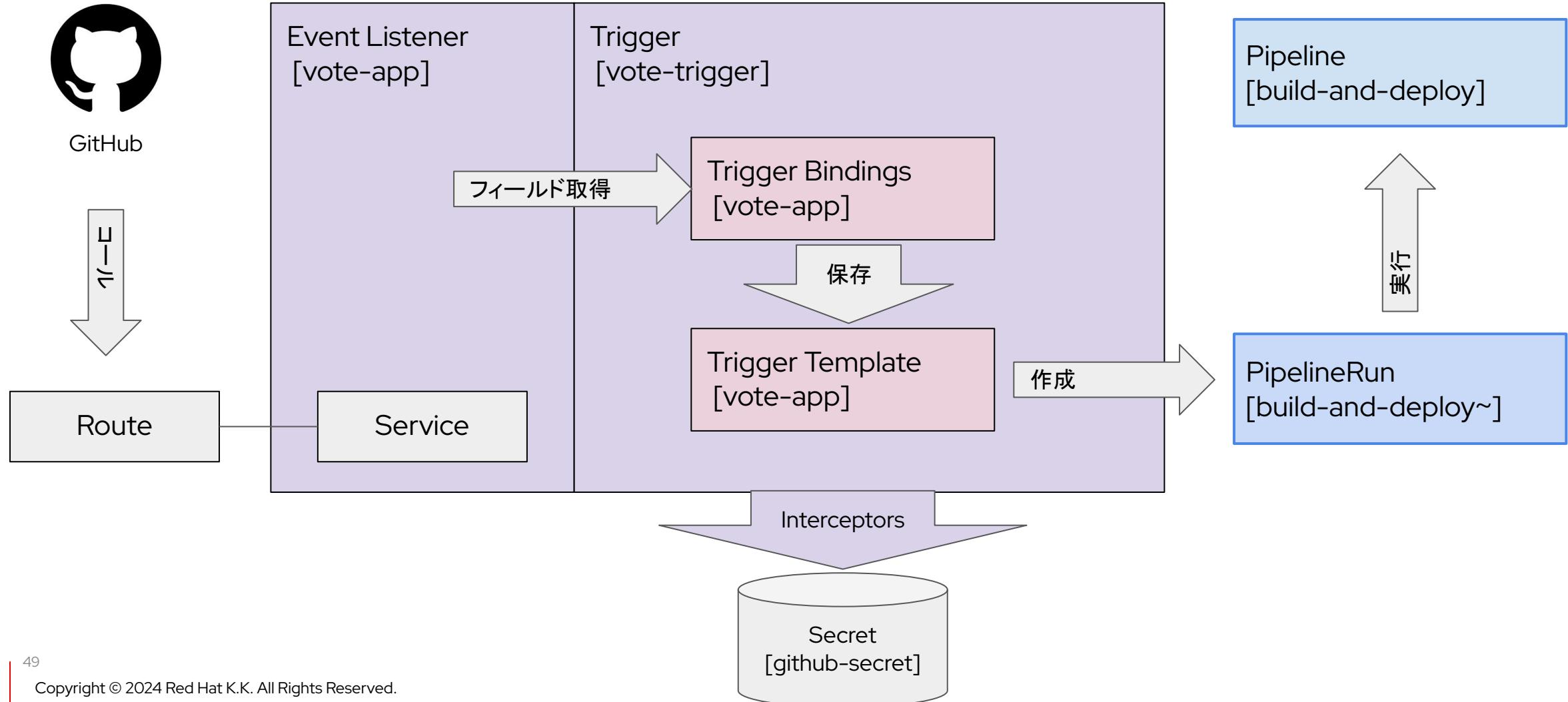


# Triggers

コミットやpull requests時などのHTTP webhooksなどのイベントトントに基づいてパイプラインを実行します



# Triggers



2

---

# OpenShift GitOps

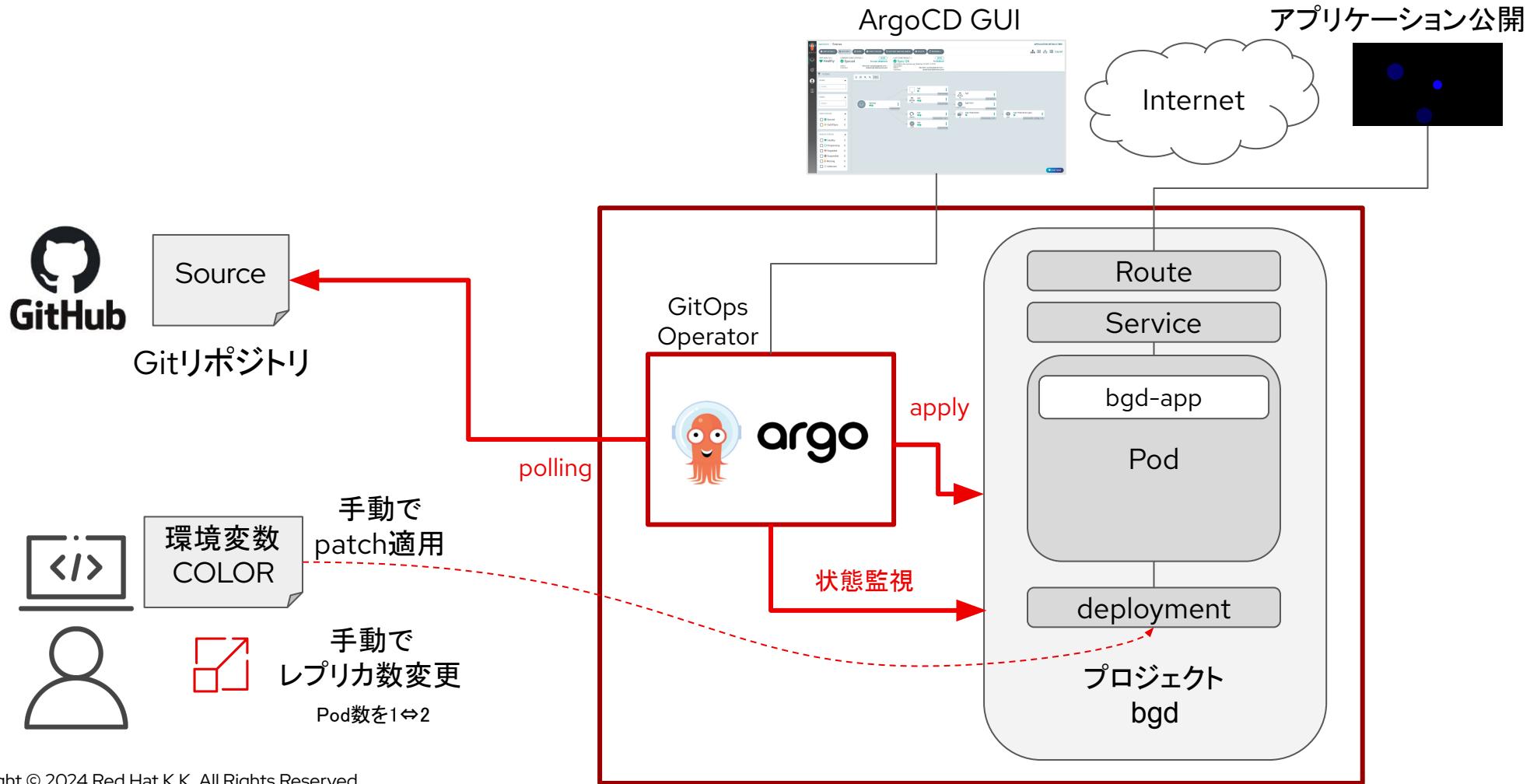
# GitHubアカウントをお持ちでない方

# OpenShift GitOps

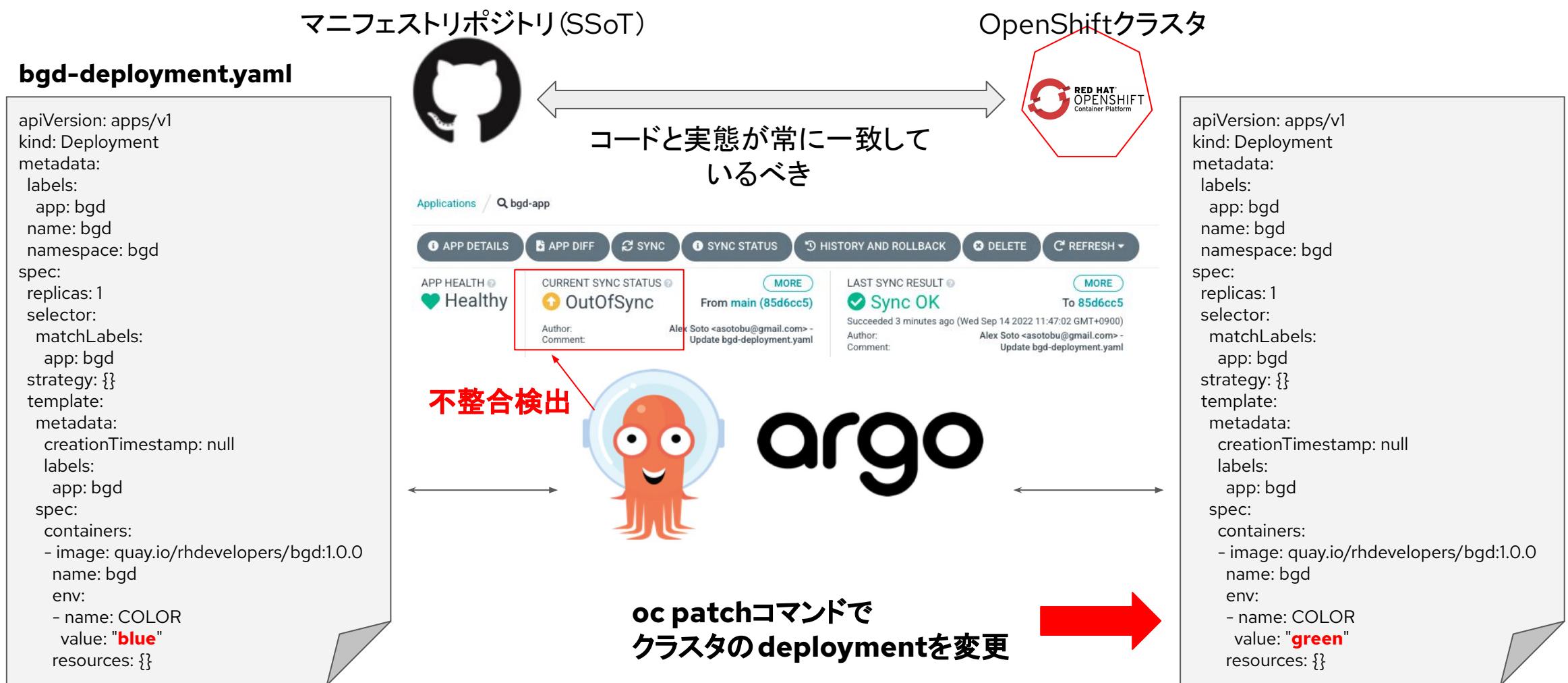
## 演習内容

- 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登録します
- 動作環境の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

# OpenShift GitOps



# 不整合検出の動作



# 自動同期設定後の動作



# 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

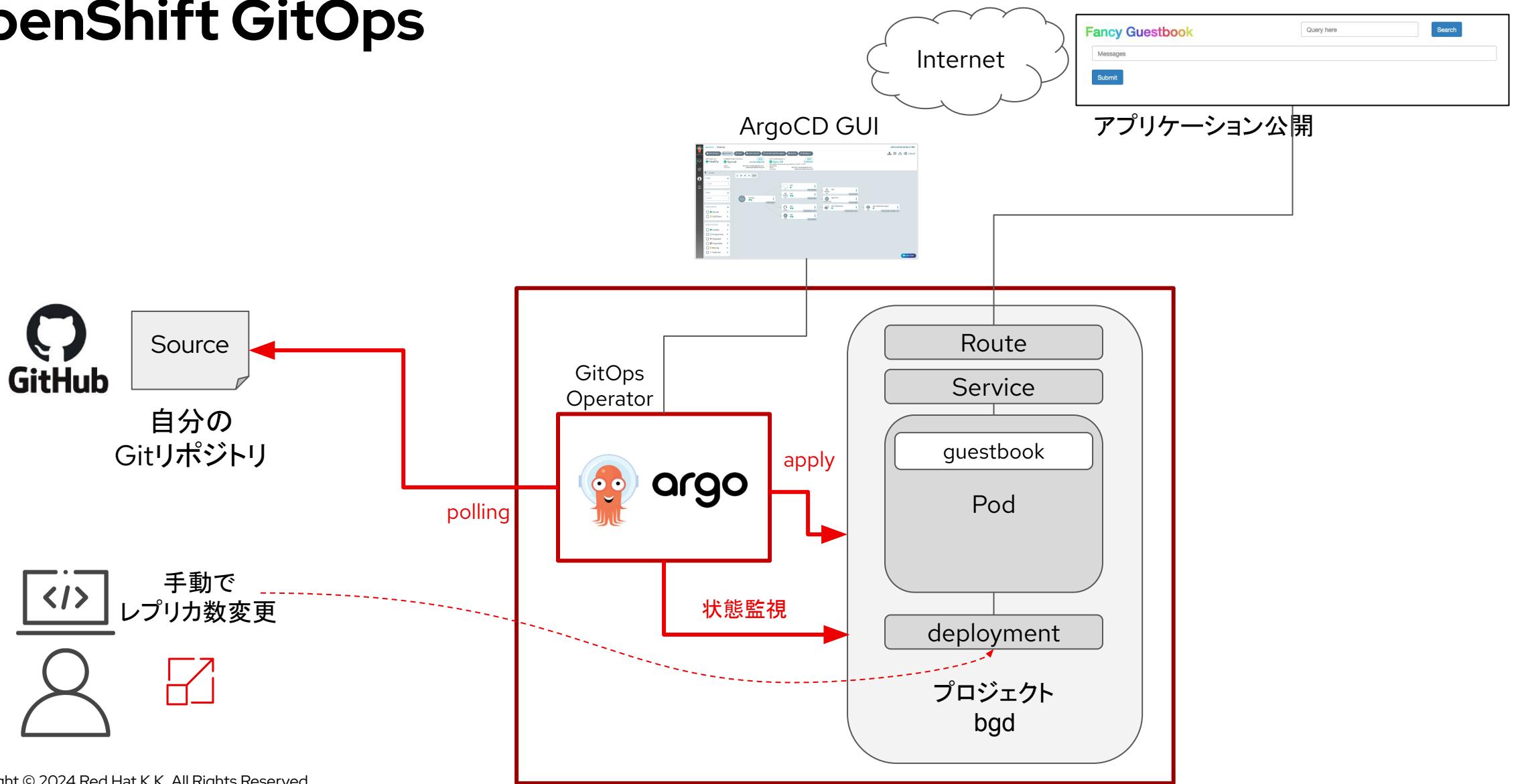
# GitHubアカウントをお持ちの方

# 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
- GitHubにあるマニフェストを利用してアプリケーションを登録します
- GitHubにあるマニフェストを変更した場合のArgoCDの動作を確認します
- 動作している環境に変更を加えた場合のArgoCDの動作を確認します
- GitHub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数を減らした場合のArgoCDの動作を確認します

# OpenShift GitOps



# アプリケーションの登録

The screenshot shows the Application Registry interface. On the left, there's a sidebar with icons for user management, clusters, and namespaces. The main area has tabs for 'CREATE' and 'CANCEL'. The 'CREATE' tab is active, showing a 'SOURCE' section and a 'DESTINATION' section.

**SOURCE**

- Repository URL: `git://github.com/your-repo/app.git` (GIT)
- Revision: `develop` (Branches: `main`, `develop`)
- Path: `app/overlay/develop`

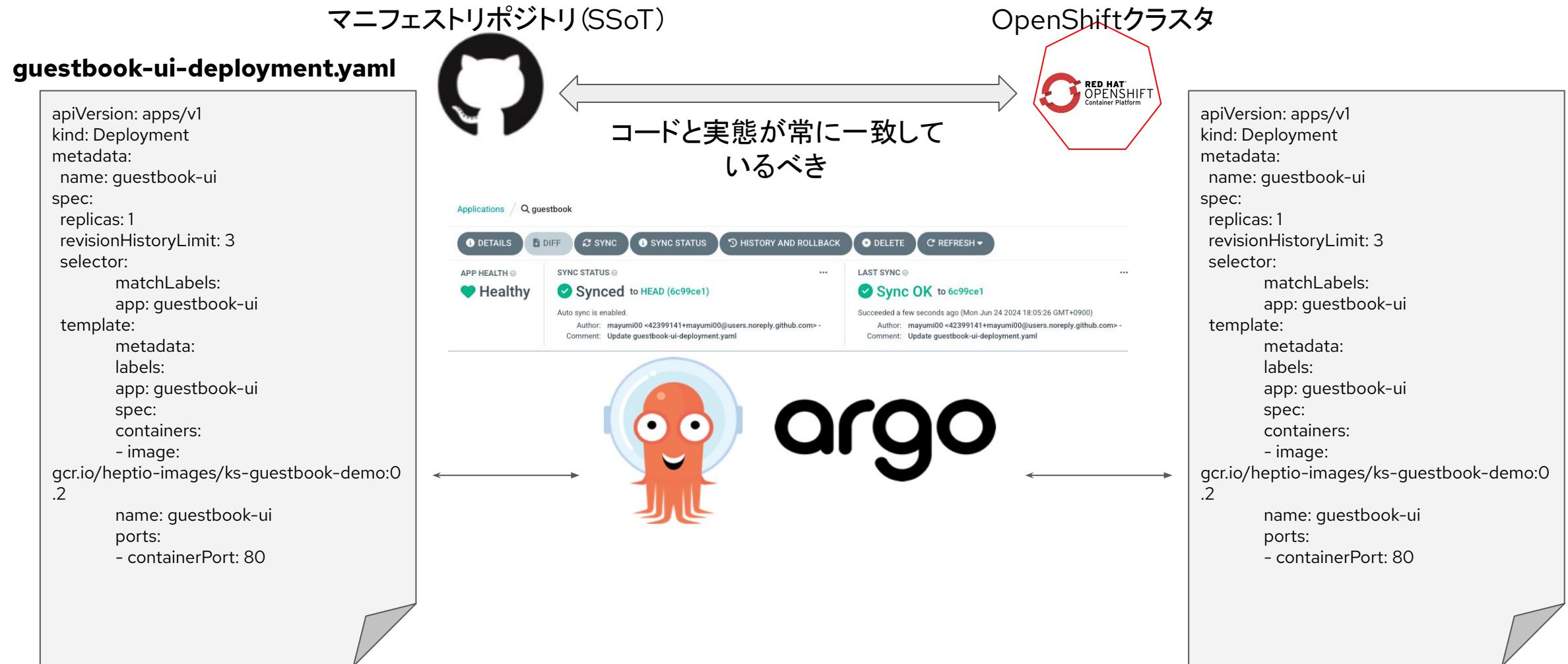
**DESTINATION**

- Cluster URL: `https://kubernetes.default.svc` (URL)
- Namespace: `default`

Annotations in red:

- 指向 `Repository URL` 的箭头: GitリポジトリのURL
- 指向 `Revision` 的箭头: 見に行く対象のブランチ
- 指向 `Path` 的箭头: 実行するyamlがあるディレクトリ
- 指向 `Cluster URL` 的箭头: デプロイ先クラスタ
- 指向 `Namespace` 的箭头: デプロイ先namespace

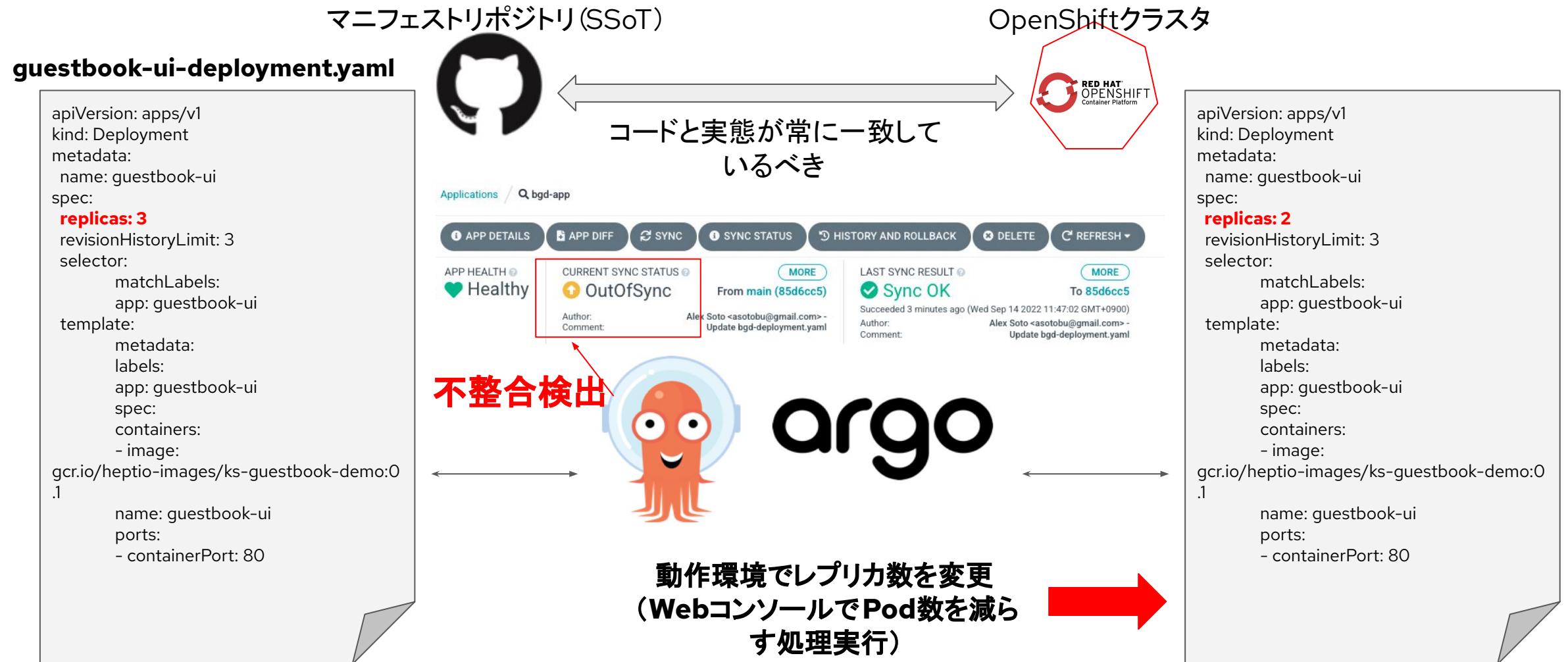
# アプリケーションの登録



# マニフェストの変更



# 不整合検出の動作



# 自動同期設定後の動作

マニフェストリポジトリ(SSoT)

**guestbook-ui-deployment.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: guestbook-ui
spec:
  replicas: 3
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: guestbook-ui
  template:
    metadata:
      labels:
        app: guestbook-ui
    spec:
      containers:
        - image:
          gcr.io/heptio-images/ks-guestbook-demo:0.1
          name: guestbook-ui
          ports:
            - containerPort: 80
```



コードと実態が常に一致して  
いるべき

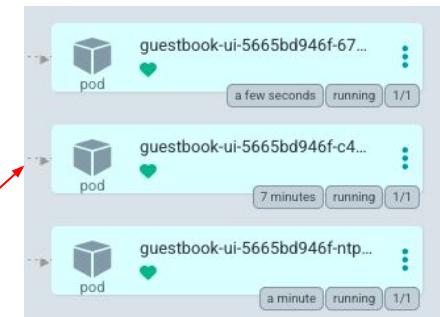
OpenShiftクラスタ



差分を検知し  
減ったPodを作成してマ  
ニフェストに合わせる



argo



WebコンソールでPod数を3→2  
に変更を試みる

The screenshot shows the deployment configuration for 'guestbook-ui' in the 'default' namespace. It indicates 3 Pods, a RollingUpdate strategy, and a limit of 25% of Pod capacity. A red square highlights the '3 Pod' count, and a red arrow points from the Argo section towards this field.

名前	更新ストラテジー
guestbook-ui	RollingUpdate

Namespace	利用できないPodの最大値
NS default	25%/3 Pod

ラベル	Pod増分の最大値
app.kubernetes.io/in... =guestb...	3 Pod を 25% 超える

Podセレクター	進行の期限(秒)
Q app=guestbook-ui	600秒

# ハンズオンコース紹介

## アプリケーションデプロイ

# 3

---

## 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

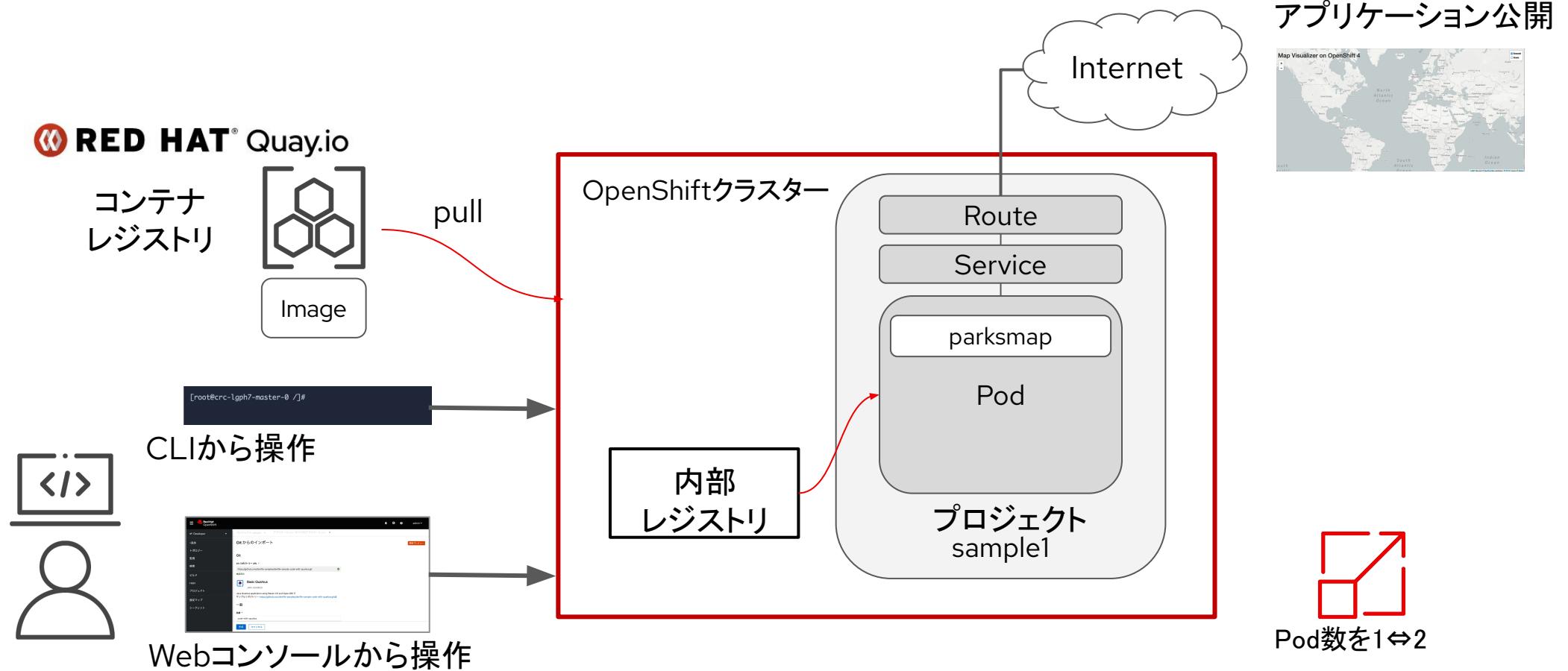
- OpenShiftのCLIおよびGUIの基本的な使用方法を学習します
- GUIおよびCLIを利用してイメージからアプリケーションをデプロイする方法を学習します
- アプリケーションを公開する方法を学習します
- アプリケーションをスケールアップする方法を学習します
- OpenShiftの主なリソースを理解します

# 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



# 4

---

## 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

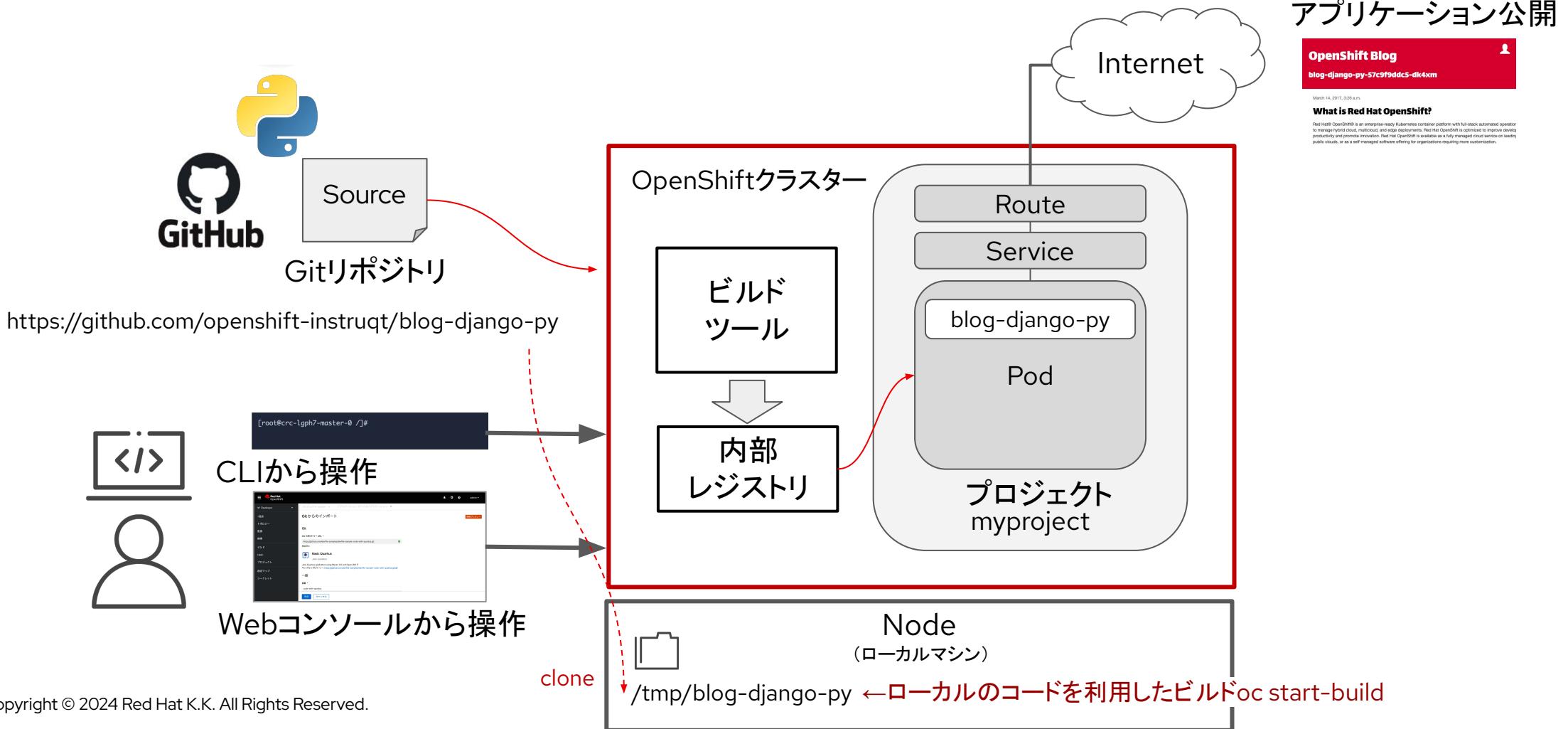
- OpenShiftのCLIおよびGUIの基本的な使用方法を学習します
- GUIおよびCLIを利用してソースコードからアプリケーションをデプロイする方法を学習します
- OpenShiftのSource to Image (S2I) を学習します

# 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 ↗

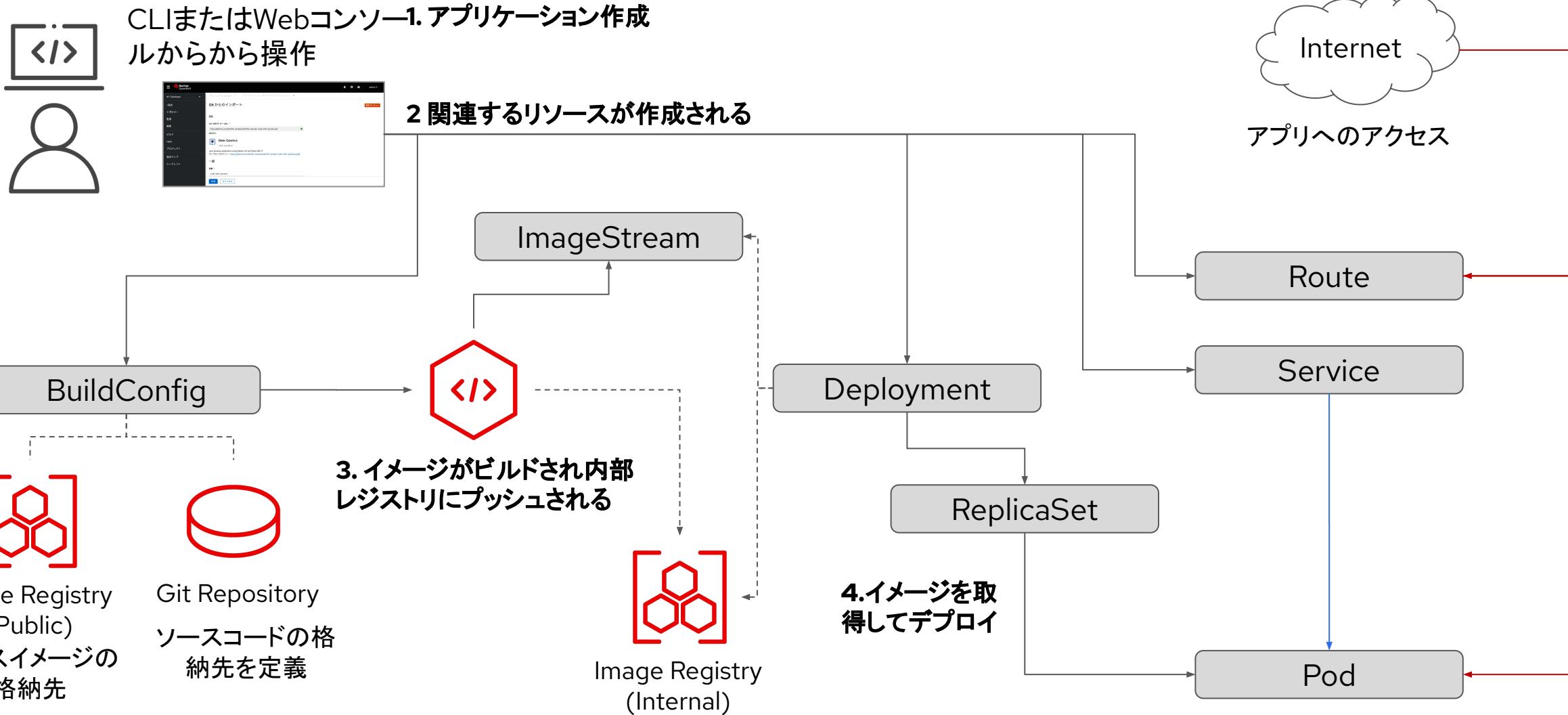
› Show advanced Git options

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



# 5

---

## 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

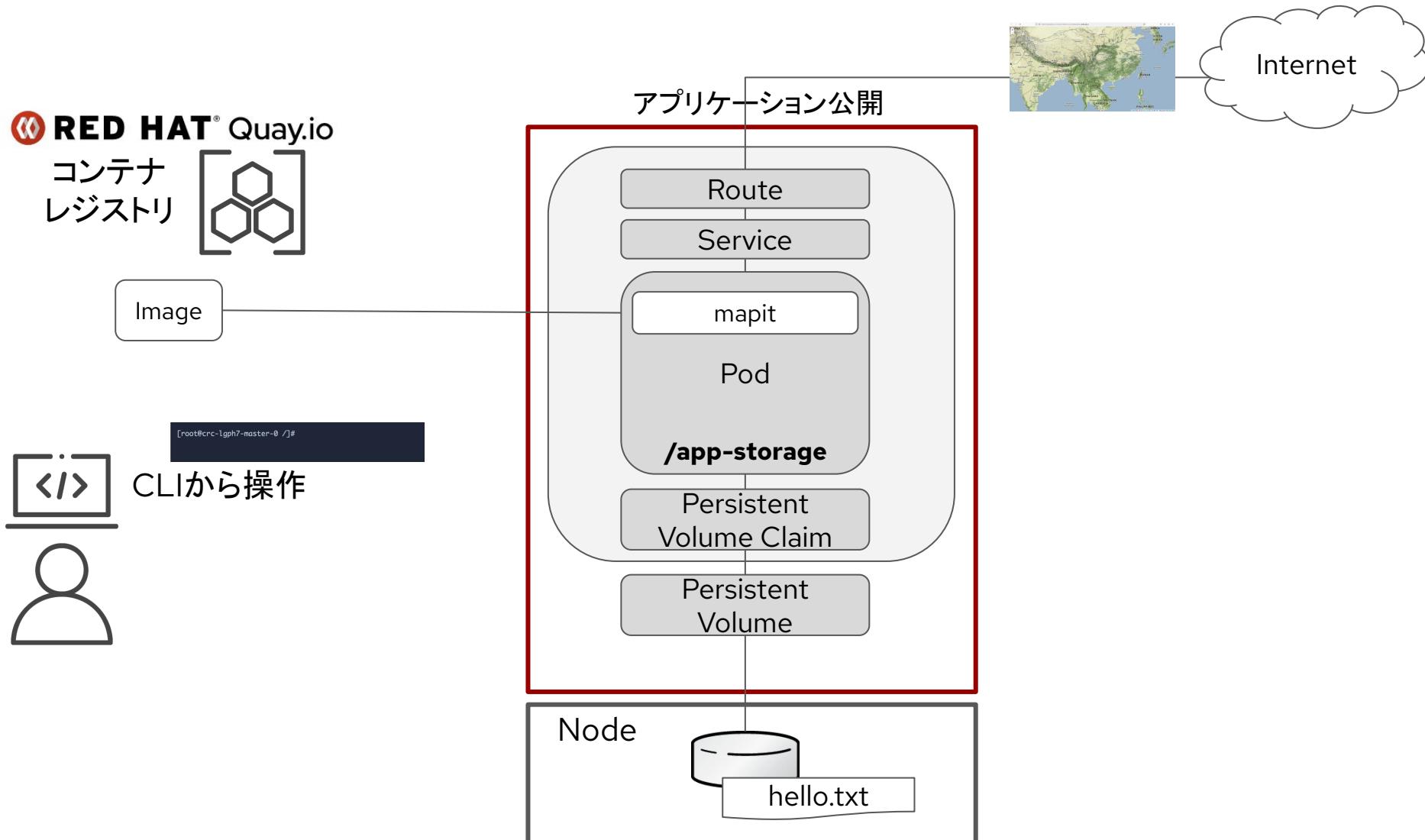
永続ボリューム(Persistent Volume)の操作と動作確認を実施し、使用方法や関連する概念について学びます。



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

## Podman

# 6

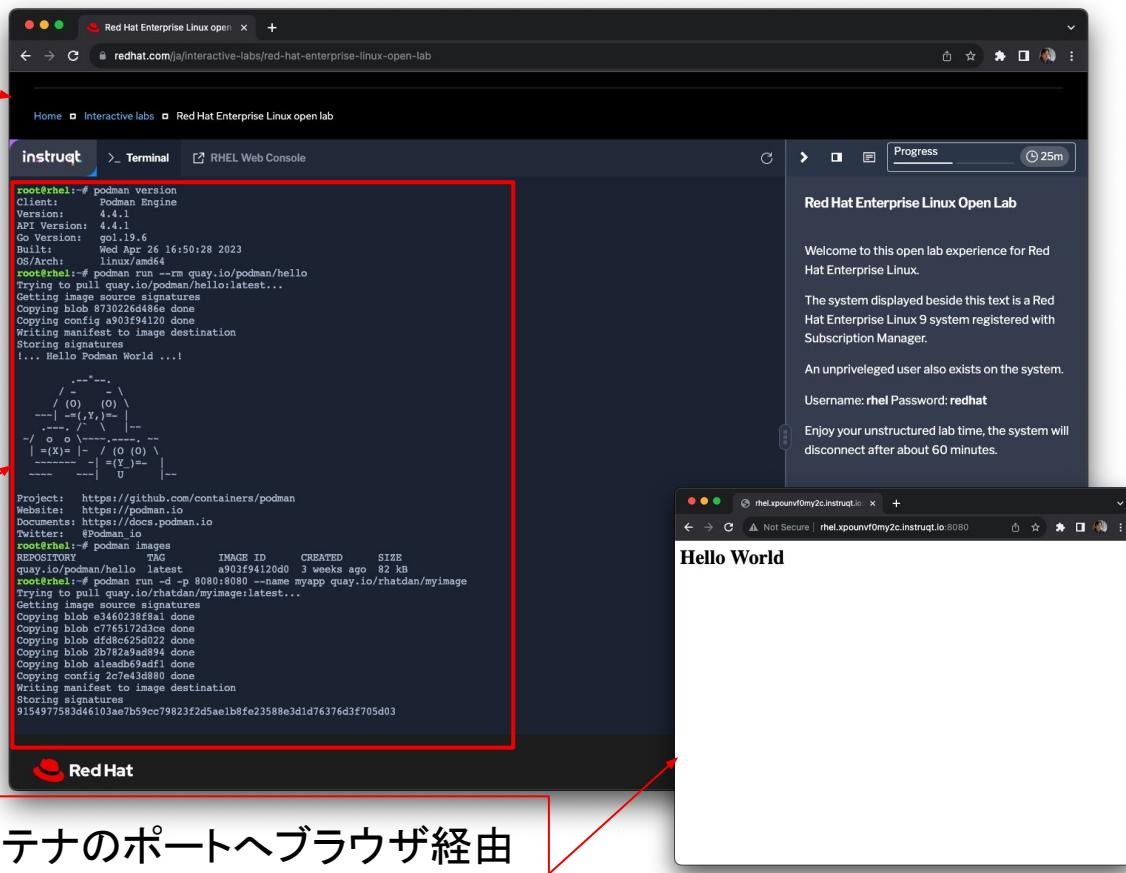
---

## 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

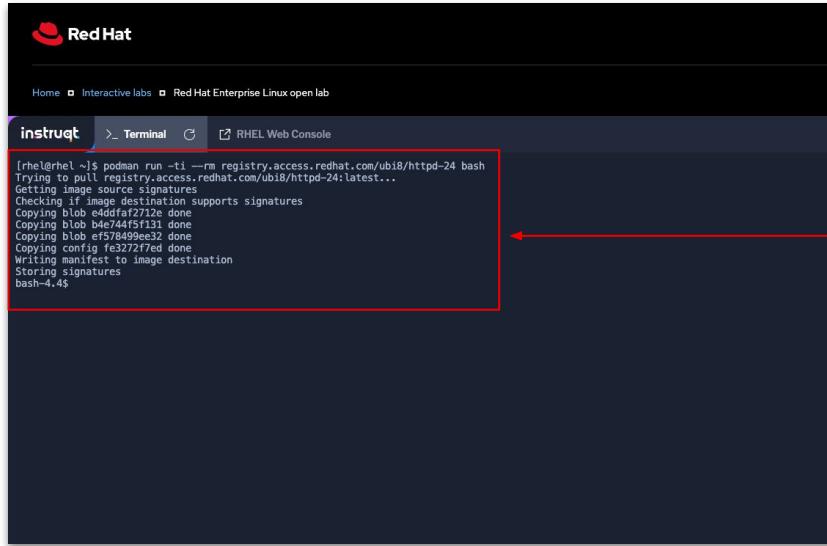
- セルフペースで学べる学習コンテンツ
- ブラウザのみで利用可能。1回の起動は連続60分まで。
- Podmanイン・アクションのコマンド例を使用しているので書籍と合わせた学習が可能
- オンラインのハンズオンイベント開催時は運営によるサポートも提供

\* <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 ]



```
Instrukt >_ Terminal RHEL Web Console
Home Interactive labs Red Hat Enterprise Linux open lab
[redacted]
$ 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 9578499e9332 done
Copying config c4127096ce 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



出力結果を展開する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 browser displaying a documentation page for 'Podman/ハンズオン'. The page has a dark blue header with the title 'Podman/ハンズオン' and navigation links for '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'. A search bar is also present in the header.

The main content area features a sidebar on the left with a table of contents and a list of chapters. The chapters listed are: ハンズオンドキュメント, Chapter 2, Chapter 3, Chapter 4, Chapter 5, Chapter 6, (WIP)Chapter 7, (WIP)Chapter 8, (WIP)Chapter 9, (WIP)Chapter 10, (WIP)Chapter 11, (WIP)Appendix A, and (WIP)Appendix B.

The main content area contains the following sections:

- Chapter 2 コマンドライン**
  - 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
    - イメージの構築
- 事前作業**

一般ユーザー(ユーザー名 rhel)にスイッチする

```
# su - rhel
```

(コピペ用)

```
su - rhel
```

✓ 出力結果 >
- Table of contents**
  - 事前作業
    - 2.1 コンテナの操作
      - コンテナの探索
      - コンテナ化したアプリケーションの実行
      - コンテナの停止
      - コンテナの起動
      - コンテナのリスト表示
      - コンテナの調査
      - コンテナの削除
      - コンテナへの実行
      - コンテナからイメージを作成
    - 2.2 コンテナイメージの操作
      - コンテナとイメージの違い
      - イメージのリスト表示
      - イメージの調査
      - イメージのプッシュ
      - podman login : コンテナレジストリへのログイン
      - イメージのタグ付け
      - イメージの削除
      - イメージのブル
      - イメージの検索
      - イメージのマウント
    - 2.3 イメージの構築
      - アプリケーションのビルドの自

# 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。

89

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 current project is 'default'. You can switch projects 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](https://twitter.com/RedHat)