

OpenShift 4 Foundations 説明資料

2. OpenShift ユーザエクスペリエンス

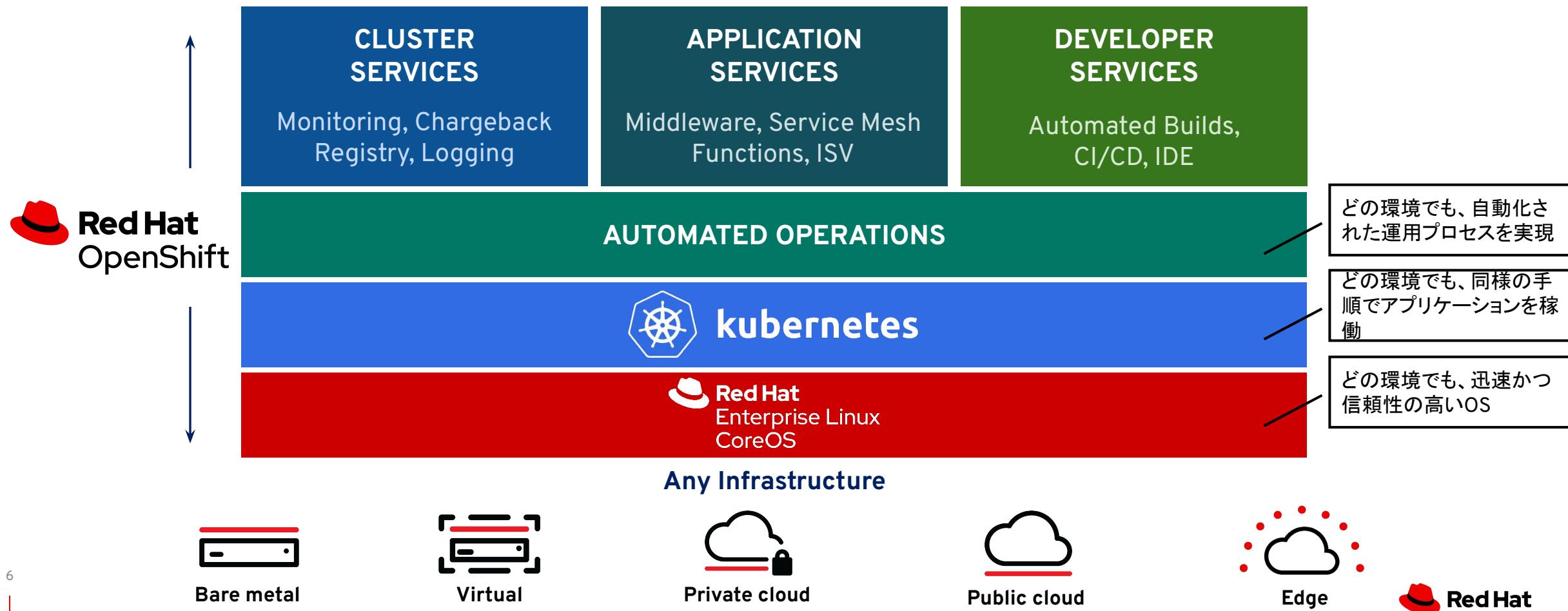
アジェンダ

OpenShift ユーザエクスペリエンス

1. OpenShiftの追加機能: Project/Template
2. ロギング & モニタリング
3. サーバレス & サービスマッシュ
4. OpenShiftへのデプロイ方法

OpenShiftはKubernetesを補完する付加価値機能を提供

運用プロセスを含めたポートアビリティの実現



Bare metal



Virtual



Private cloud



Public cloud



Edge



1. OpenShiftの追加機能: Project/Template

2. ロギング & モニタリング
3. サーバレス & サービスマッシュ
4. OpenShiftへのデプロイ方法

OpenShiftの追加機能

Project/ Template

Projectを作成する時に、デフォルトで設定として、Namespace毎のリソースの使用制限を適用する機能

EgressNetworkPolicy

egress ファイアウォール機能。クラスタ外部のホストやサブネットに対する Pod のアクセス許可/不許可を設定できる機能。Pod を特定のホストにしかアクセスさせたくないような要件で利用。

ClusterResourceQuota

同じラベル、アノテーションを持つ複数のプロジェクトを統合して、Quota を設定する機能。複数のプロジェクトをまたいで制限する際に利用。

SCC

Namespace 単位で root 権限を制限したり、使用する SELinux context、secomp profile 等を設定する機能

Over The Air Update

UI、もしくは CLI の操作で、Cluster のアップデートを自動で実行する機能。

Router

Pod からの特定の外部の宛先にアクセスする場合のソース IP アドレスを、決まった IP に固定する機能。

MachineConfig Operator

Node の OS やコンテナランタイムの更新等を、Kubernetes のリソースとして管理できる機能

Image Stream

OpenShift で追加する Image をタグ付けして管理できる機能

他にもいろいろ

- Compliance Operator
- Ingress Controller
- UPI など

OpenShiftの追加機能

Project / Template

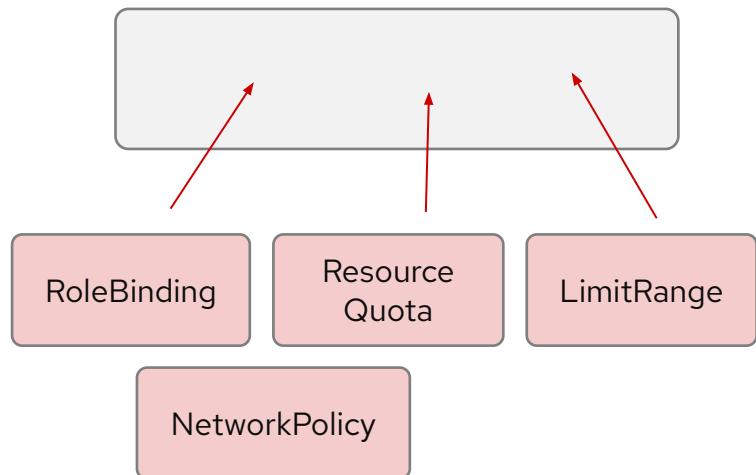
- ▶ Project (= Namespace の拡張) を実装。
- ▶ デフォルトで Project を作成する時に、リソースの制限などの設定をテンプレートのように適用
 - 自動で Namespace 毎のリソースの使用制限の適用 ができる
 - マルチテナント管理として namespace が管理しやすくなる
- ▶ この機能により、Project 作成時の管理者による Limit Range 等の個別制限適用の手間を省く事が可能
 - LimitRange: Pod や Container に割り当てるリソースの最大や最小値を設定する
 - ResourceQuota: Namespace に対するシステムリソースのクオータを設定する
 - NetworkPolicy: Namespace 内や Namespace をまたぐ Pod 間の通信を L4 で制御する
 - RBAC: ロールベース (RBAC) のアクセス管理を提供する

OpenShiftの追加機能

namespace の作成

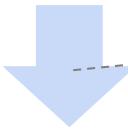


制限の無い namespace



個別に必要な設定を手動で適用

project の作成



基本設定のされた project

- project の 作成ユーザー名を annotation に銘記
- project の 作成ユーザーに、デフォルトで用意してある admin ClusterRole に紐付け (RoleBinding)
- Resource Quota の適用
- LimitRange の適用
- NetworkPolicy の適用
- etc...

制限が設定された
project/namespaceを用意できる

```

....  

metadata:  

  creationTimestamp: null  

  name: admin  

  namespace: ${PROJECT_NAME}  

  roleRef:  

    apiGroup: rbac.authorization.k8s.io  

    kind: ClusterRole  

    name: admin  

  subjects:  

    - apiGroup: rbac.authorization.k8s.io  

      kind: User  

      name: ${PROJECT_ADMIN_USER}  

  - apiVersion: v1  

  kind: ResourceQuota  

metadata:  

  name: project-quota  

  namespace: ${PROJECT_NAME}  

spec:  

  hard:  

    count/pods: 10  

    limits.cpu: 6  

    limits.memory: 16Gi  

    requests.cpu: 4  

    requests.memory: 8Gi  

    requests.storage: 20G  

  - apiVersion: v1  

  kind: LimitRange  

metadata:  

  name: project-limits  

  namespace: ${PROJECT_NAME}  

spec:  

  limits:  

    - default:  

      cpu: 1  

      memory: 1Gi  

    defaultRequest:  

      cpu: 500m  

      memory: 500Mi  

      type: Container  

  - apiVersion: networking.k8s.io/v1  

  kind: NetworkPolicy  

metadata:  

  name: allow-from-openshift-ingress  

  namespace: ${PROJECT_NAME}  

spec:  

  ingress:  

    - from:  

    ....
  
```

1. OpenShiftの追加機能:Project/Template
- 2. ロギング & モニタリング**
3. サーバレス&サービスメッシュ
4. OpenShiftへのデプロイ方法

コンテナアプリの本番適用

OpenShift involve Kubernetes ecosystem with partner & community

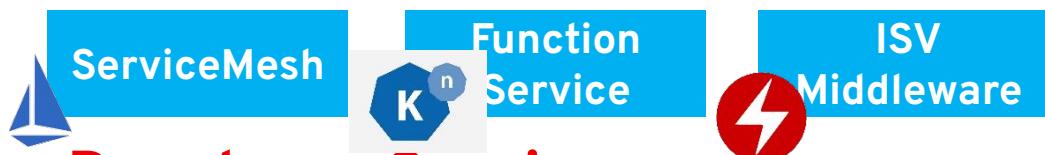
Cluster Services

クラスタ管理を容易にし、運用業務を自動化するサービス



Application Services

マイクロサービス間の連携やファンクションサービスを実現を支援するサービス



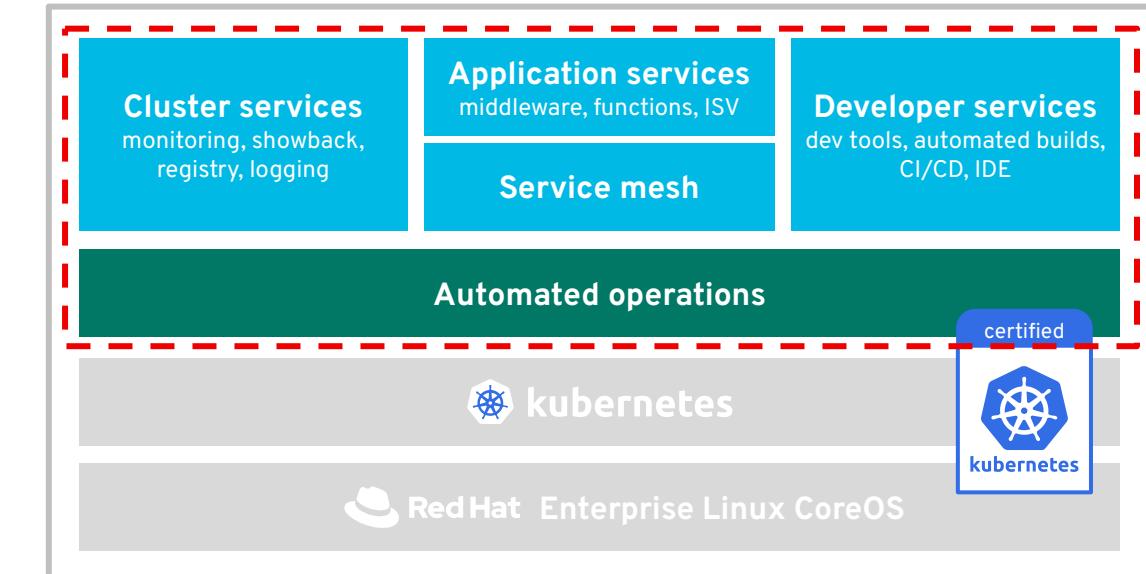
Developer Services

開発者がコンテナアプリケーション開発に集中できる環境を整えるサービス



Build fast. Ship first.

チームがスピード、アジリティに自信を持ってビルドできる環境を提供。クラウドネイティブな開発を本番へのコード作成を支援。



Cluster Logging

Cluster Logging(EFK)を有効化することによって、各Pod、Clusterから出力されるログをElasticsearch上に保管できます。また、必要に応じてKibanaから見ることも可能です。

The screenshot shows the Kibana interface with the following details:

- Search Bar:** 30,902 hits. Search query: >_ Search... (e.g. status:200 AND extension:PHP)
- Discover Tab:** Selected field: _source.
- Visualize Tab:** Selected visualization: Histogram of log entries by timestamp. X-axis: September 2nd 2021, 13:04:17.595 - September 2nd 2021, 13:19:17.595. Y-axis: Count (0 to 2,000). The histogram shows several peaks, with the highest peak around 13:05:00 and another significant one around 13:16:00.
- Log View:** Shows log entries for two specific timestamps:
 - September 2nd 2021, 13:19:15.526: _SYSTEMD_INVOCATION_ID: 6a2ea8eeeca72419a99cc41bc911ffff17 systemd.t.SYSTEMD_INVOCATION_ID: 6a2ea8eeeca72419a99cc41bc911ffff17
systemd.t.BOOT_ID: a435aa7d8f9b46aa935b53e7ead022df systemd.t.GID: 0 systemd.CMDLINE: kubelet --
config=/etc/kubernetes/kubelet.conf --bootstrap-kubeconfig=/etc/kubernetes/kubeconfig --
kubeconfig=/var/lib/kubelet/kubeconfig --container-runtime=remote --container-runtime-endpoint=/var/run/crio/crio.sock --
runtime-cgroups=/system.slice/crio.service --node-labels=node-role.kubernetes.io/worker,node.openshift.io/os_id=rhcos --
 - September 2nd 2021, 13:19:15.280: kubernetes.container_image_id: quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:64d6dbb1da4f06088f6f5734e5f57039bcf23bd737e540bc986c911e9ddd2a64 kubernetes.container_name: csi-liveness-probe
kubernetes.namespace_id: d6949166-7022-413a-90d6-8b7453c57e69 kubernetes.flat_labels: app=aws-ebs-csi-driver-node,
controller-revision-hash=57fd47685f, pod-template-generation=1 kubernetes.host: ip-10-0-137-33.ap-southeast-
1.compute.internal kubernetes.master_url: https://kubernetes.default.svc kubernetes.pod_id: 46386bcf-3009-4c2c-afeb-

The screenshot shows the Kubernetes Pod logs interface for a completed pod named "java-sample-1-build".

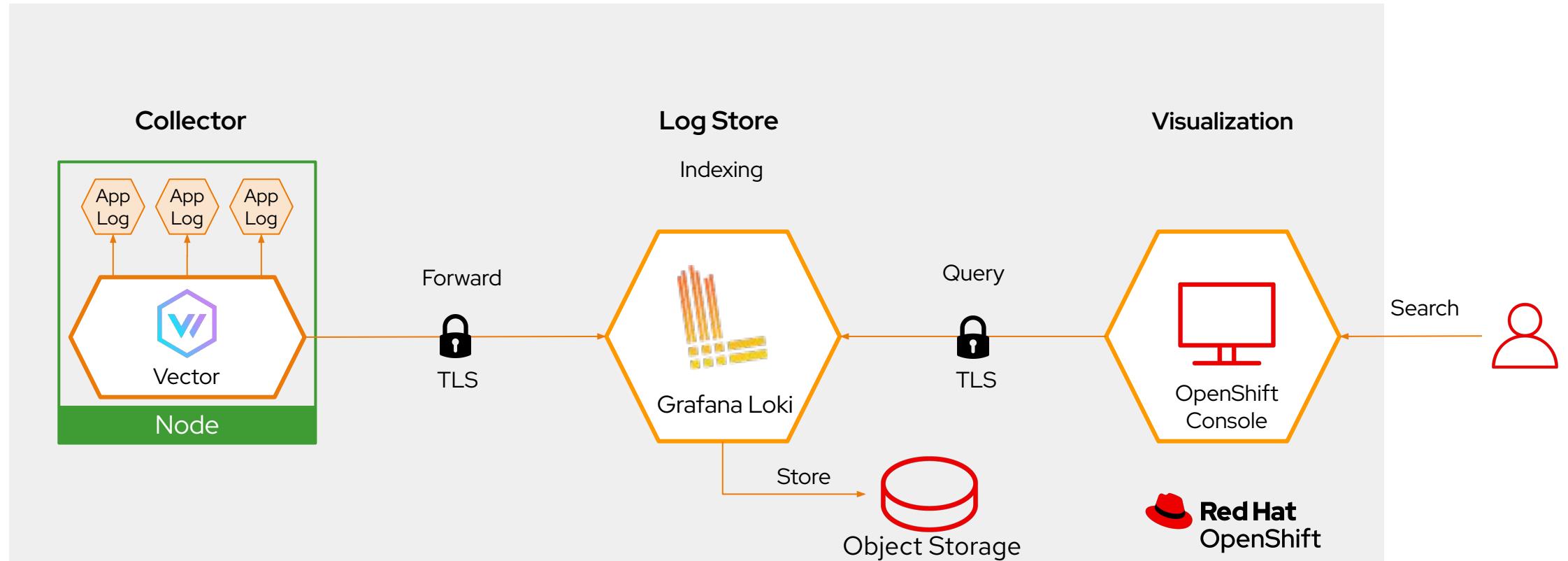
- Header:** Pods > Pod details. Pod name: java-sample-1-build. Status: Completed.
- Actions:** Actions dropdown menu.
- Tabs:** Details, Metrics, YAML, Environment, Logs (selected), Events, Terminal.
- Log Stream:** Log stream ended. Filter: sti-build. Current log. Options: Last 15 minutes. Refresh button.
- Log Content:** Shows 143 lines of log output. The logs are primarily Maven dependency download logs from the Sonatype repository, including entries for sisu-inject-bean, guice, maven-model-builder, aether-provider, aether-impl, and aether-spi jars.
- Buttons:** Show in Kibana, Wrap lines, Raw, Download, Expand.

※EFK Elasticsearch
Fluentd
Kibana



Overview

- OpenShift Logging は OCP クラスタ上のユーザアプリやシステムコンポーネント関連のログ収集と可視化を提供する [1]



Note:

アプリケーションの監視設定もOpenShiftで簡素化

k8sでJavaを見守る新常識

伊藤ちひろ
Chihiro Ito

監視についてよくある問題

- 監視なんて不要だと思って監視の仕組みを作成するコストはない
- 監視が必要だと思っているが、監視についての知識がない
- 監視環境の管理なんてしたくない
- 監視をしても問題の分析に必要な情報が取れていない
- 監視をしても問題の予兆を検知できず、問題が顕在化する
- 問題を製品のせいにし、別の製品へ移行して再発する

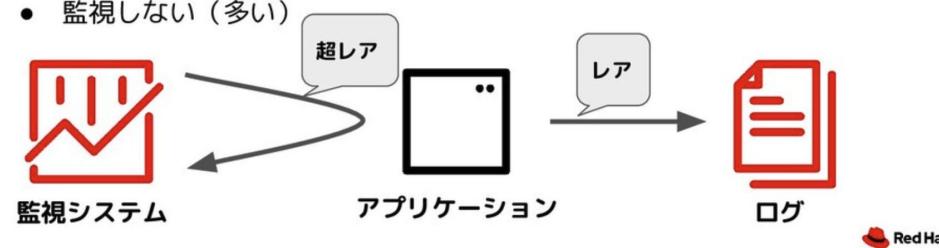
今日の
本題

Red Hat

26

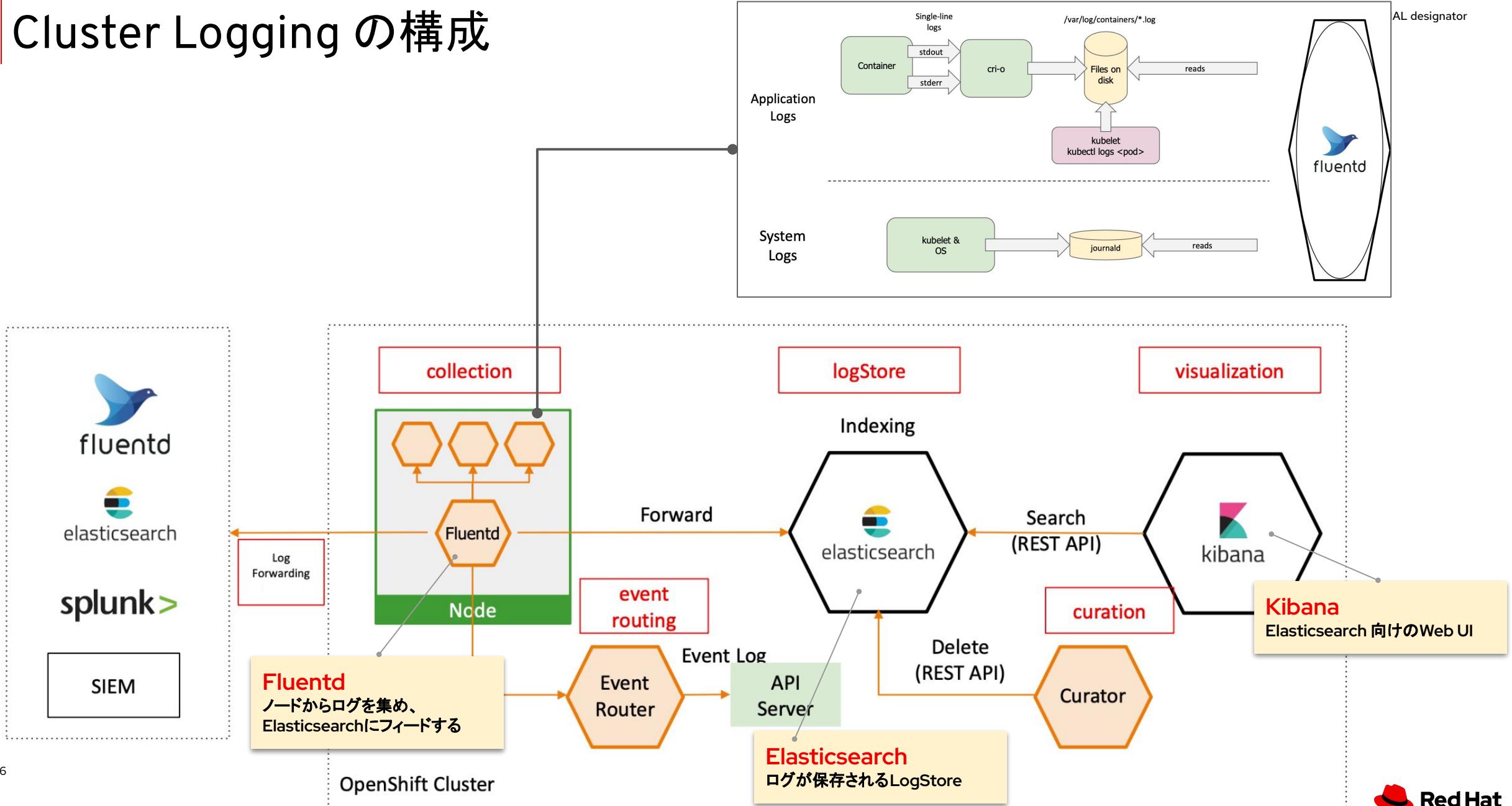
従来の監視方法の例

- 監視システムを構築し、監視対象を登録（超レア）
- 情報をログファイルへ保存（レア）
- 監視しない（多い）



<https://speakerdeck.com/chiroito/k8sdejavawojian-shou-ruxin-chang-shi>

Cluster Logging の構成

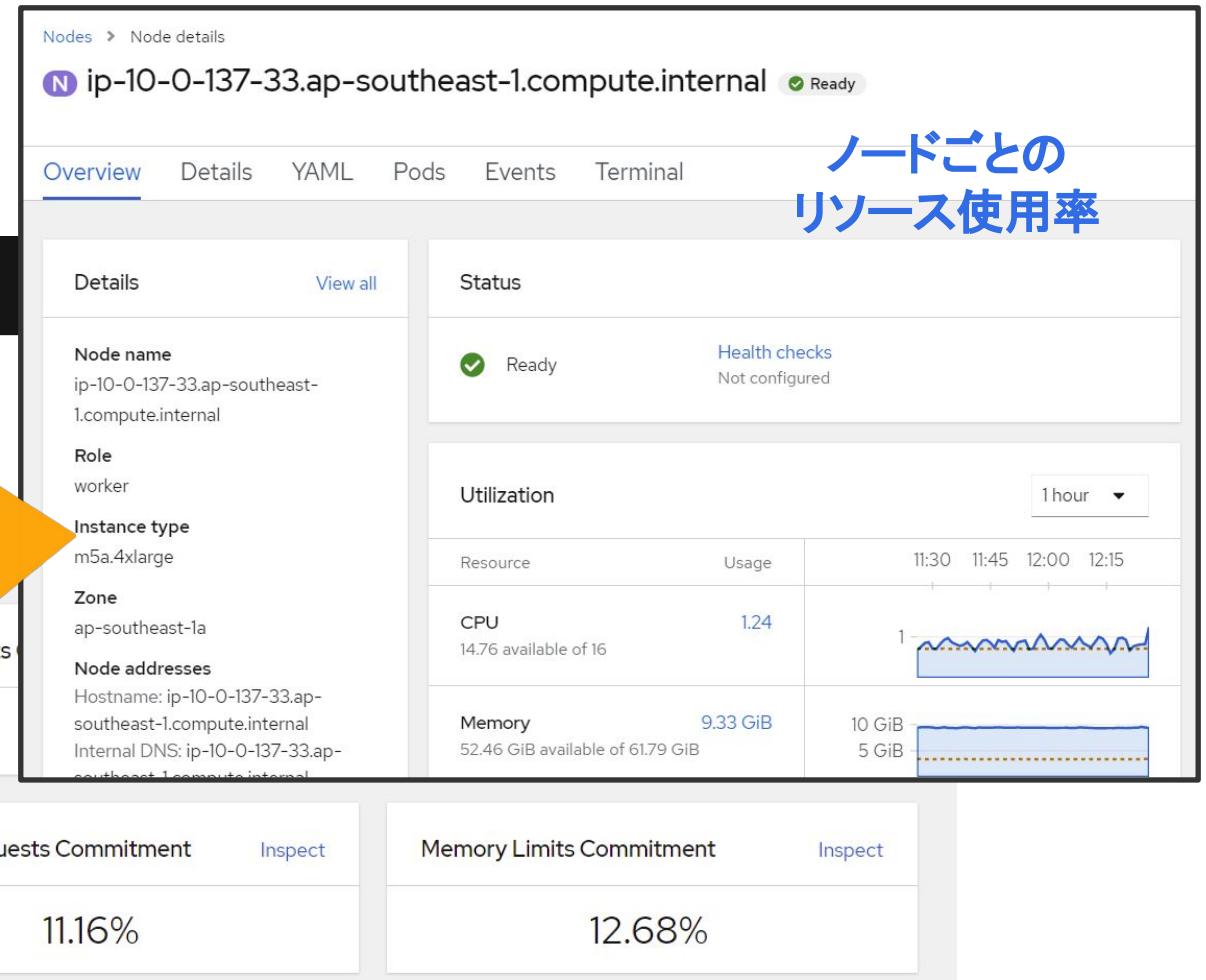


Cluster Monitoring

OpenShiftをインストールした時点で、クラスタに関する監視はCluster Monitoring(Prometheus)によって設定済みです。(アラート含む)

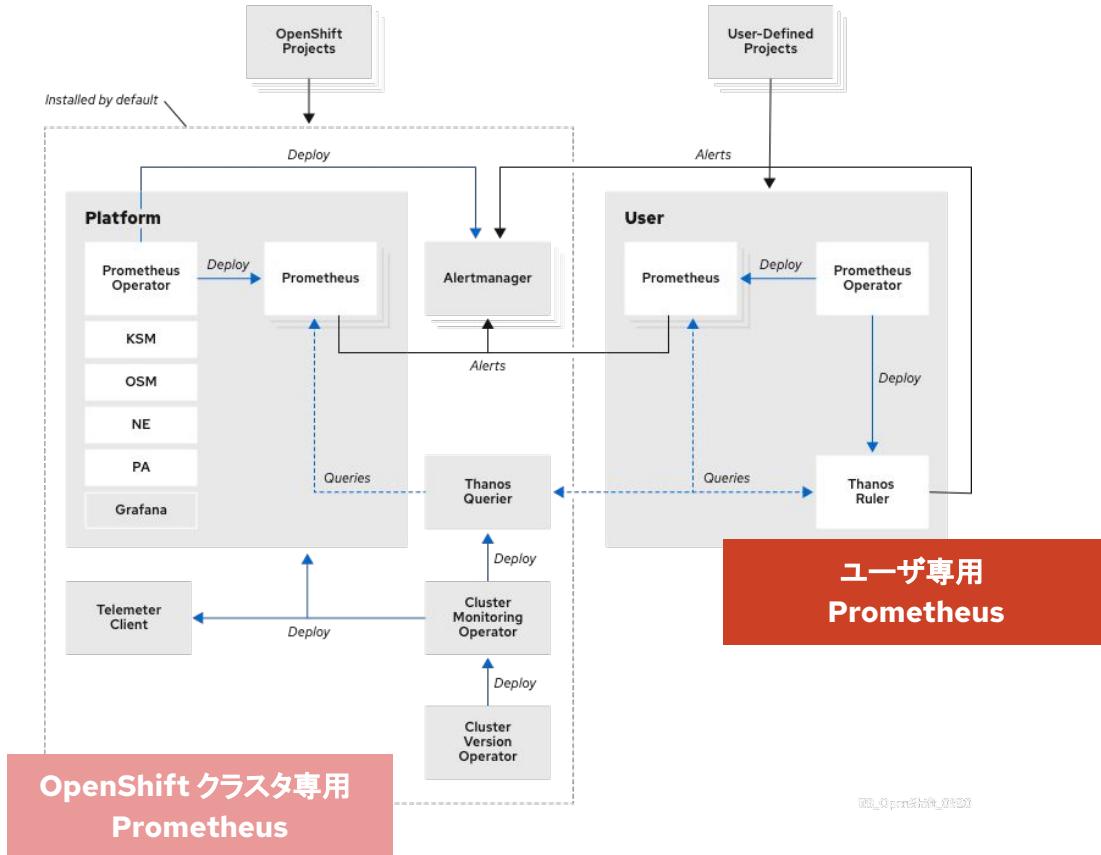
初期設定や更新作業は自動で設定されます。

The screenshot shows the Red Hat OpenShift Container Platform dashboard. On the left, there's a navigation sidebar with options like Operators, Workloads, Networking, Storage, Builds, Monitoring (which is expanded), Alerting, Metrics, and Dashboards. The main area is titled "Dashboards" and "Cluster". It displays four cards: "CPU Utilisation" (8.02%), "Memory Utilisation" (18.82%), "CPU Requests" (highlighted with a red arrow), and "Memory Requests Commitment" (11.16%).



ノードごとの
リソース使用率

Cluster Monitoring の構成



- **OpenShift クラスタ専用 Prometheus [1]**
 - 対象
 - Kubernetes/OpenShift API、etcd、CoreDNS、ホストOS周り
 - 備考
 - Cluster Monitoring に設定されているルールのカスタマイズはサポート対象外(監視対象は、OpenShiftに委ねられている)
- **ユーザ専用 Prometheus**
 - Apache、Nginxなどユーザがデプロイしたアプリケーションの固有のメトリクス収集

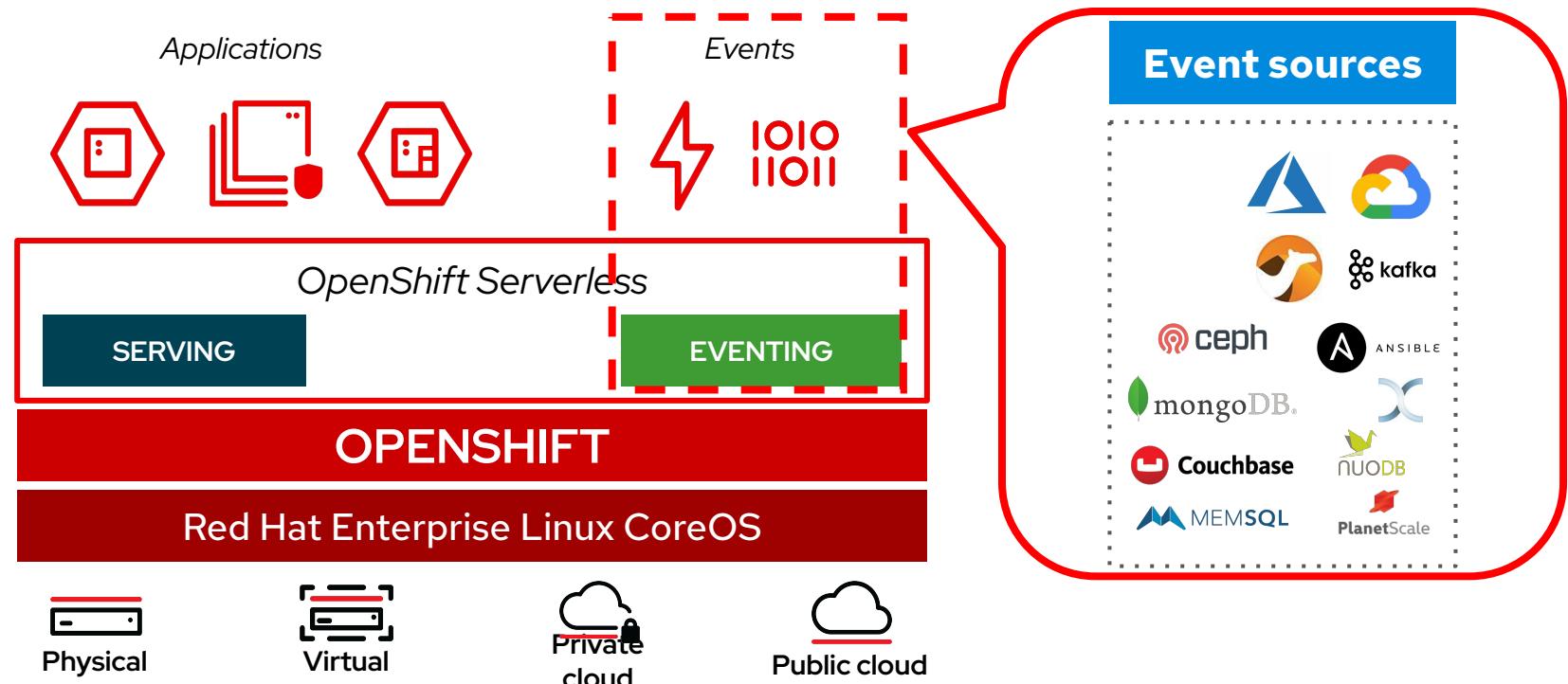
・K8sオブジェクトのメトリクス収集:
 例:Pod数、Podのリソース利用量、Serviceのラベル、etc
 ・K8sノードのメトリクス収集:
 例:NodeのCPU、Memory、I/O、etc

・アプリケーション固有のメトリクス収集:
 例:HTTPセッション数、リクエストレスポンスタイム、データ容量 etc

1. OpenShiftの追加機能:Project/Template
2. ロギング＆モニタリング
- 3. サーバレス＆サービスメッシュ**
4. OpenShiftへのデプロイ方法

OpenShift Serverless

- Knativeをベースとした、ServerlessをOpenShiftの標準機能として提供
 - Zero Server Opsやアイドル時のコンピューティングコストを0にすることが可能
- .NET/Go/Java/Node.js/PHP/Python/Ruby/Shellなど、様々なプログラミング言語に対応 ([サンプルコード](#))
- Serverlessの主要な特徴
 - Serving: 使う時だけアプリを起動し、使われていない時は0("ゼロ")ヘスケール可能
 - Eventing: HTTPリクエストやデータのアップロードなど、**様々なイベントをトリガーにしたアプリの起動** が可能 ([イベントソースのリスト](#))

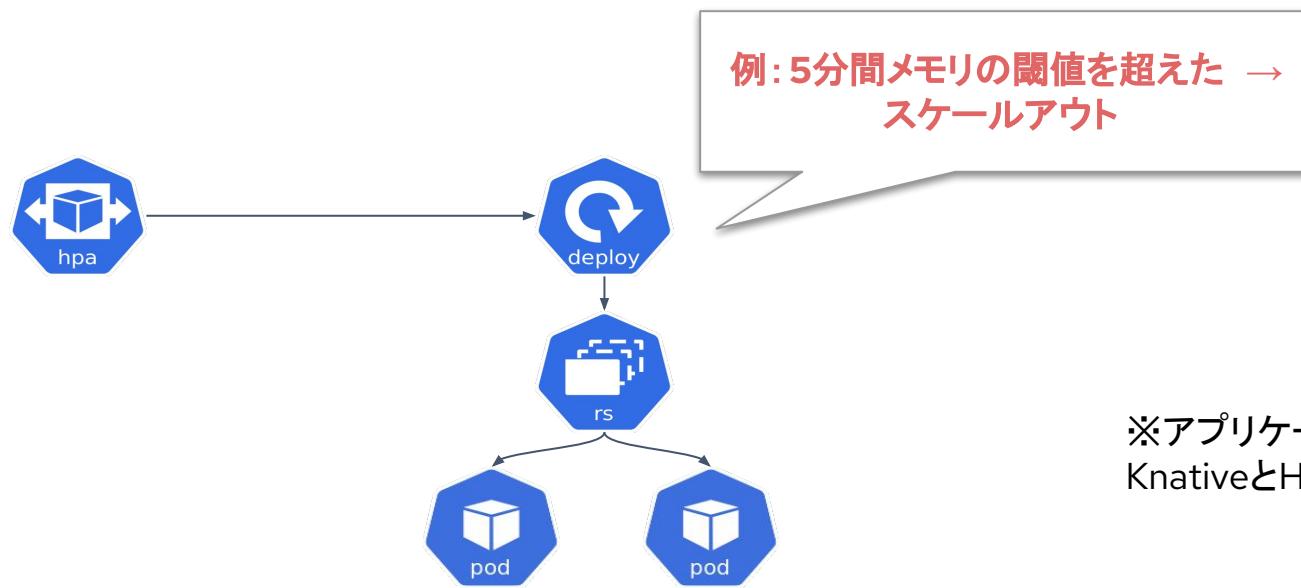


Note:

CPUとメモリベースの使用量は HPAで使用可能

HPA (Horizontal Pod Autoscaler)

HPAはCPUとメモリのメトリクス情報と連携し、設定した閾値を超えた場合(下回った場合)に、対象の Deployment 等のレプリカ数を変更することで自動でのスケーリングを実現します。



※アプリケーションの性質に応じて
KnativeとHPAと使い分けることが重要

OpenShift Serverless のアプリケーション作成イメージ

Project: demoserverless Application: all applications

Import from Git

Git

Git Repo URL *

https://github.com/sclorg/django-ex.git

Validated

Show Advanced Git Options

Builder

Builder Image

Builder image(s) detected.

Recommended builder images are represented by ★ icon.

Perl PHP NGINX Httpd .NET Core Go Ruby

Resources

Select the resource type to generate:

- Deployment
- Deployment Config
- Knative Service

生成するリソースの種類として、
[Knative Service] を選択

A type of deployment that enables Serverless scaling to 0 when idle.

Project: demoserverless Application: all applications

Display Options Filter by Resource Find by name...

Mode: Connectivity Consumption

Expand Application Groupings Knative Services

Show Pod Count Labels

利用されていないときは、自動的に「0」までスケールイン

Autoscaled to 0 REV django...-00001

django...it-app

K SVC

サービスのリビジョンごとに、トラフィックの分散設定が可能

60% 40%

1 pod 1 pod

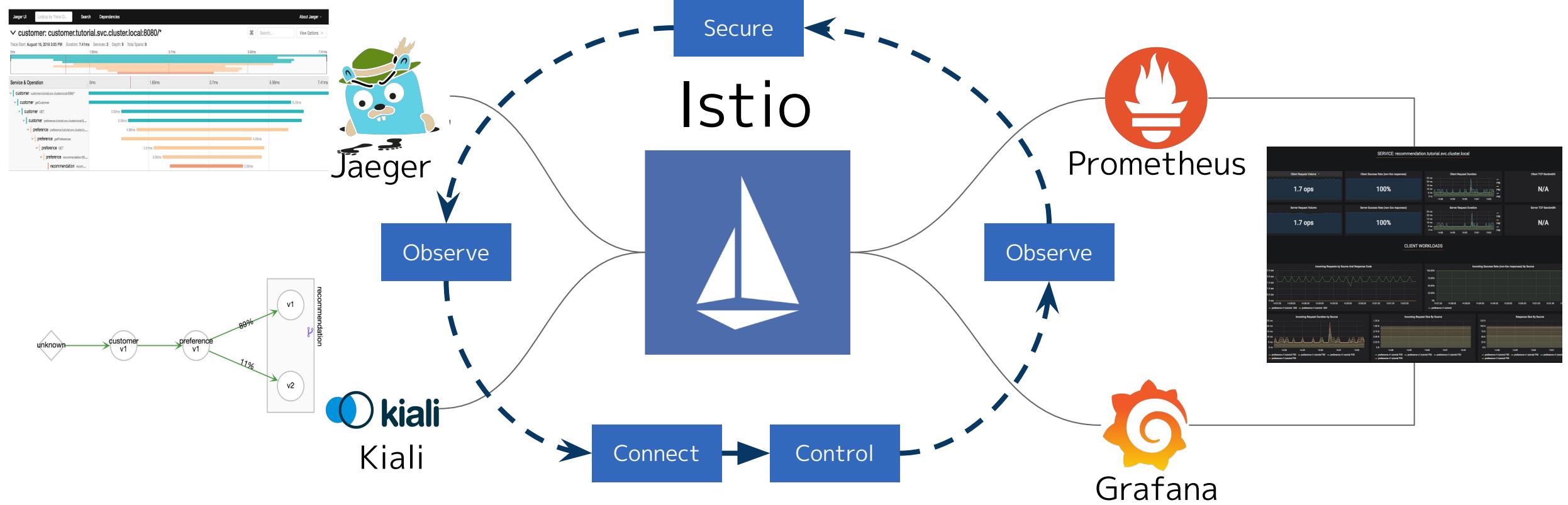
REV django...-00003 REV django...-00001

django...it-app

K SVC

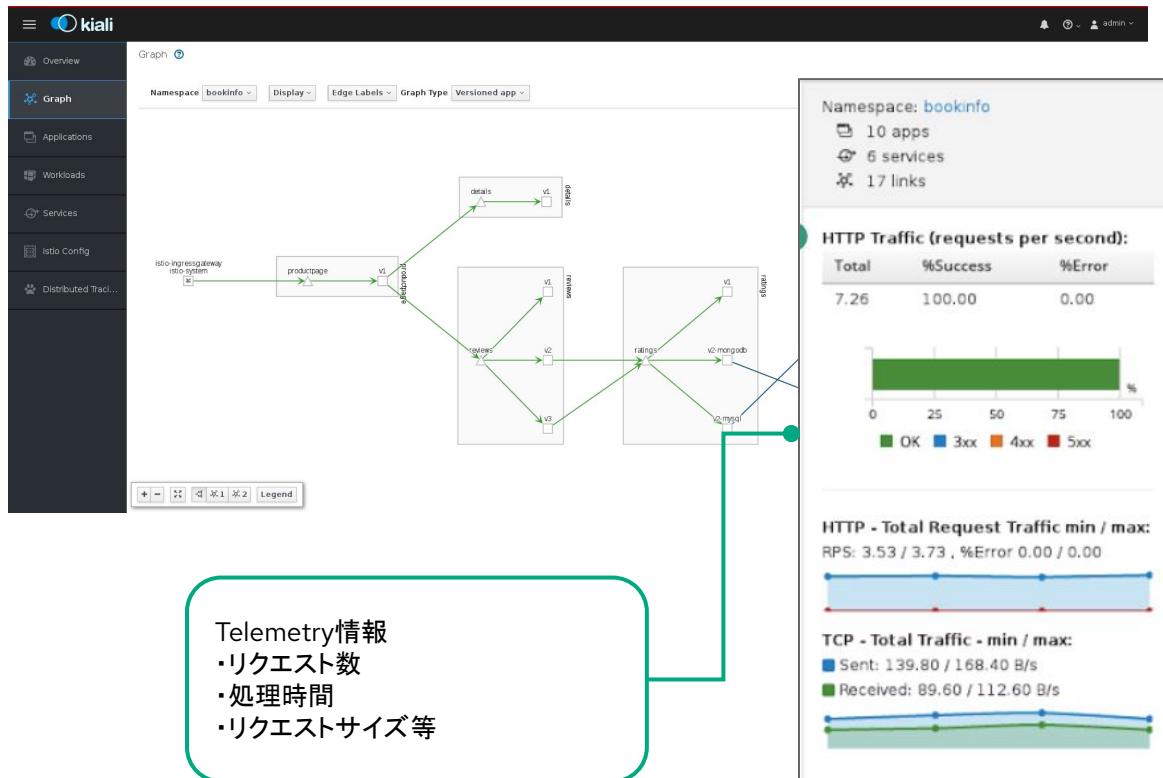
OpenShift Service Meshの紹介

OpenShift Service Meshは、istioをベースとして構成されていて、トラフィックの制御、セキュリティ、ポリシー、可観測性を提供します。



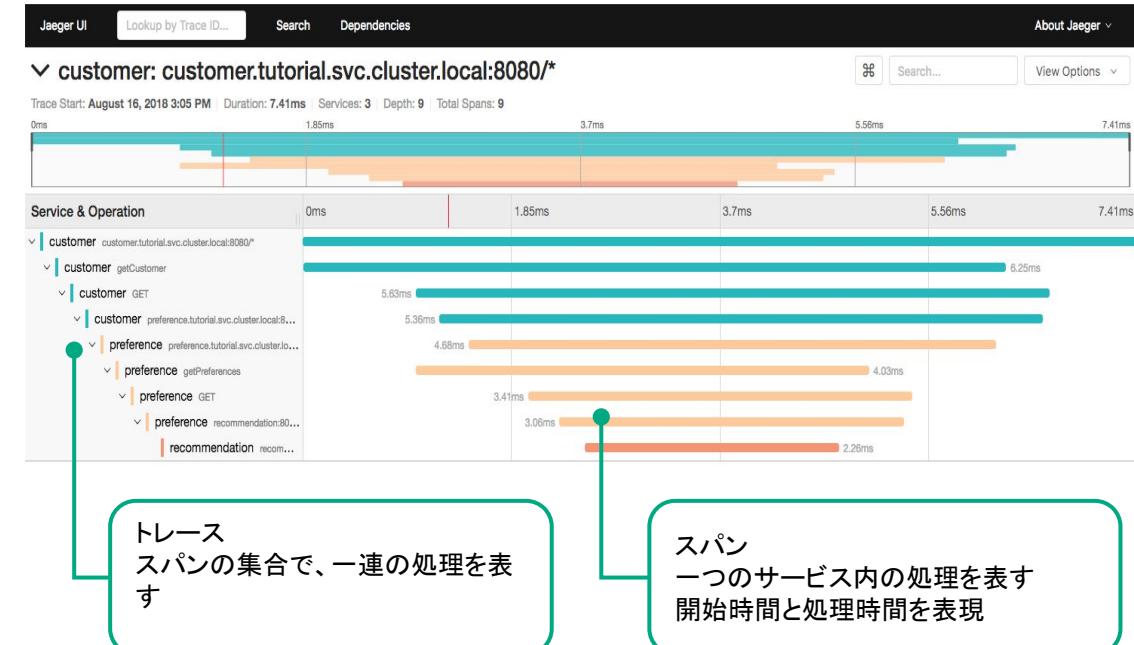
Network Topology

Kialiのコンソール画面でNetwork TopologyやTelemetry情報が確認できます。



トレーシング

Jaegerのコンソール画面でトレーシングが確認できます。



- Telemetry情報
 - リクエスト数
 - 処理時間
 - リクエストサイズ等

1. OpenShiftの追加機能:Project/Template
2. ロギング＆モニタリング
3. サーバレス＆サービスメッシュ
- 4. OpenShiftへのデプロイ方法**

アプリケーションの作成

Application Deployment from Developer Perspective

The screenshot shows the Red Hat OpenShift Container Platform interface. At the top, it says "Red Hat OpenShift Container Platform". Below that, a navigation bar has "Developer" selected. On the left, there's a sidebar with "Developer" and several other tabs: "+Add", "Topology", "Builds", "Pipelines", and "Advanced". The main area is titled "Add" and says "No workloads found". It includes a message: "To add content to your project, create an application, component or service using one of these options." Below this are six options: "From Git" (import code from a git repository), "Container Image" (deploy an existing image from an image registry or image stream tag), "From Catalog" (browse the catalog to discover, deploy and connect to services), "From Dockerfile" (import a Dockerfile from a git repo to be built & deployed), "YAML" (create or replace resources from their YAML or JSON definitions), and "Database" (browse the catalog to discover database services to add to your application).

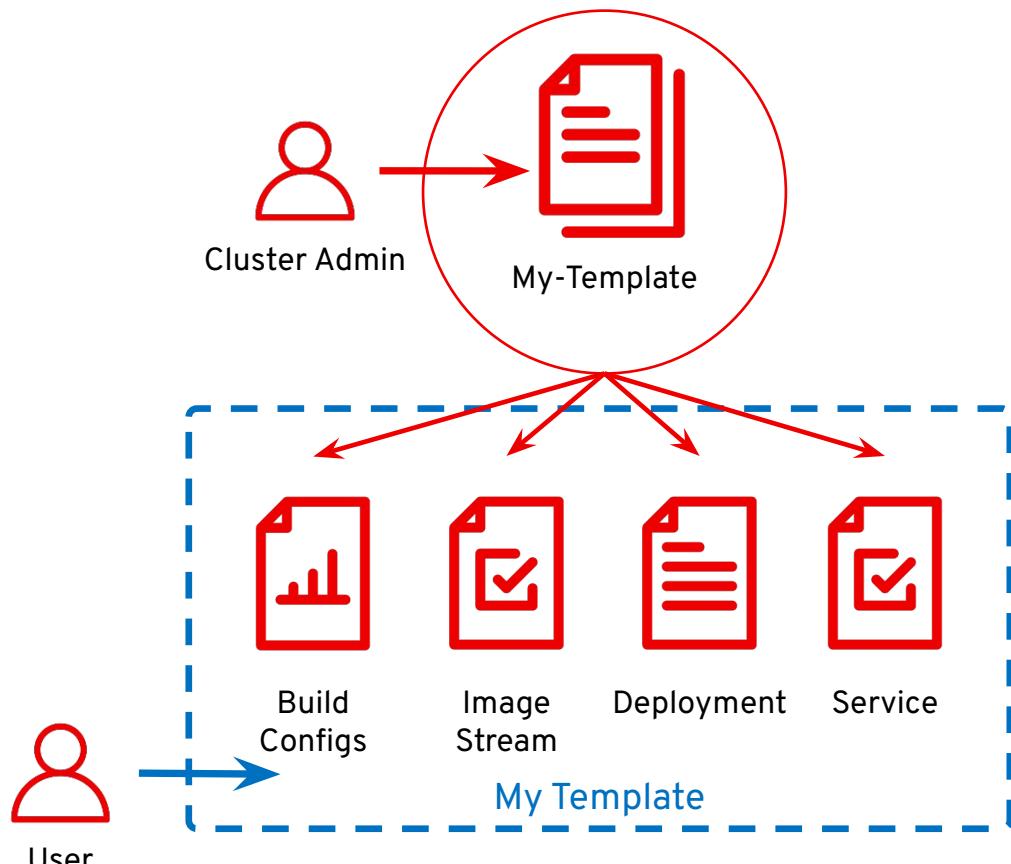
デプロイ方式

概要

From Git	このオプションを使用して、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 Image	ImageStreamまたはレジストリからの既存イメージを使用し、これをOpenShift Container Platformにデプロイします。
From Catalog	Developer Catalogで、イメージビルダーに必要なアプリケーション、サービス、またはソースを選択し、これをプロジェクトに追加します。
From Dockerfile	Git リポジトリーから dockerfile をインポートし、アプリケーションをビルドし、デプロイします。
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。

Templates (Catalog)

テンプレートは、パラメータ化され、オブジェクトのリストを生成するために処理できるオブジェクトのセットを記述します。



The screenshot shows the OpenShift web interface for instantiating a template:

Left Sidebar: Shows categories like Languages, Databases, Middleware, CI/CD, and Other. Under TYPE, it lists Service Class (0), Template (92), Source-to-Image (11), and Installed Operators (0).

Instantiation Form:

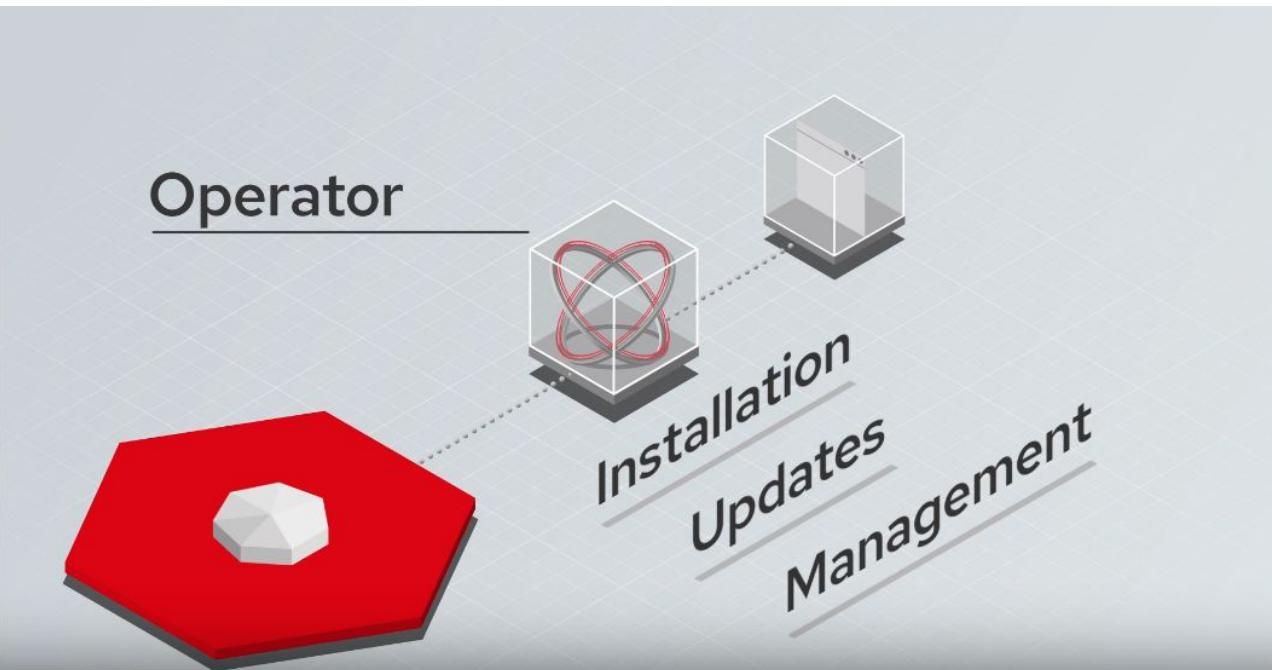
- Namespace ***: default
- Name ***: dotnet-example
- Memory Limit ***: 128Mi
- .NET builder ***: dotnet:2.2
- Namespace ***: openshift
- Git Repository URL ***: <https://github.com/redhat-developer/s2i-dotnetcore-ex.git>
- Git Reference**: dotnetcore-2.2

Right Panel: Displays the ".NET Core Example" application details, including the title ".NET Core Example", subtitle "QUICKSTART DOTNET .NET", and a description "An example .NET Core application.".

Bottom Text: "The following resources will be created:" followed by a bulleted list: BuildConfig, DeploymentConfig, ImageStream, Route, and Service.

Operator とは

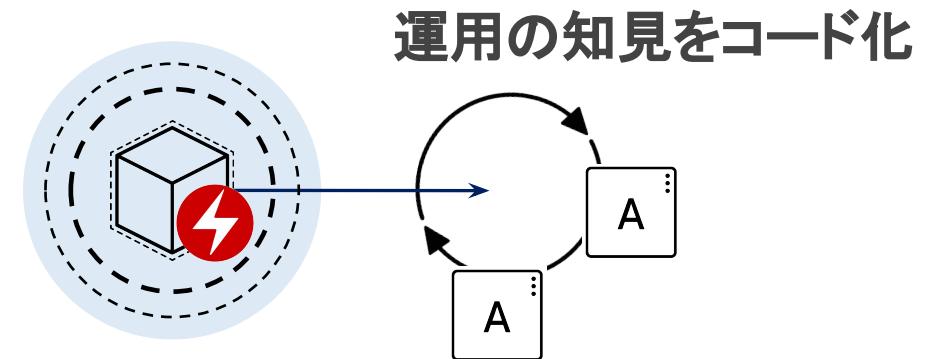
運用の知見をコード化し、アプリケーションの運用を自動化する



Kubernetes / OpenShift 上のアプリケーション運用における運用の知見をコード化し、パッケージ化したもの

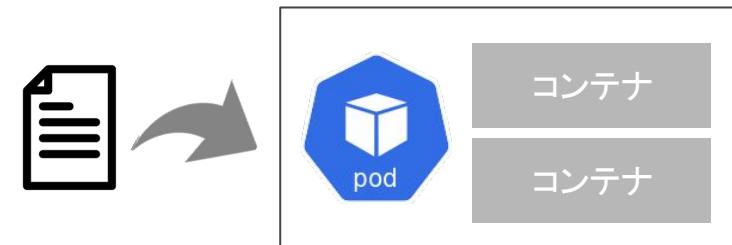
アプリケーション運用に必要な以下のような作業を自動的に行う

- ▶ インストール
- ▶ リソーススケーリング
- ▶ バックアップ
- ▶ アップデート



Operator を使うとカスタムリソースが使える

- ▶ Kubernetesに対して新しいリソースを追加する(APIの拡張)
 - ・ Custom Resource Definition(CRD)に定義を記載
 - ・ カスタムリソースを利用することができます
 - Kubernetes のデフォルトのリソース以外も、Kubernetes 内で利用ができる
 - Kuberentes 自動化機能のメリットを活用可能



リソース(K8s 組み込み)

StatefulSets
Pods
Secrets
DaemonSet
Job
CronJob
:

カスタム リソース(例)

Kafka
KafkaTopic
KafkaUser
KafkaConnect
KafkaConnector
:

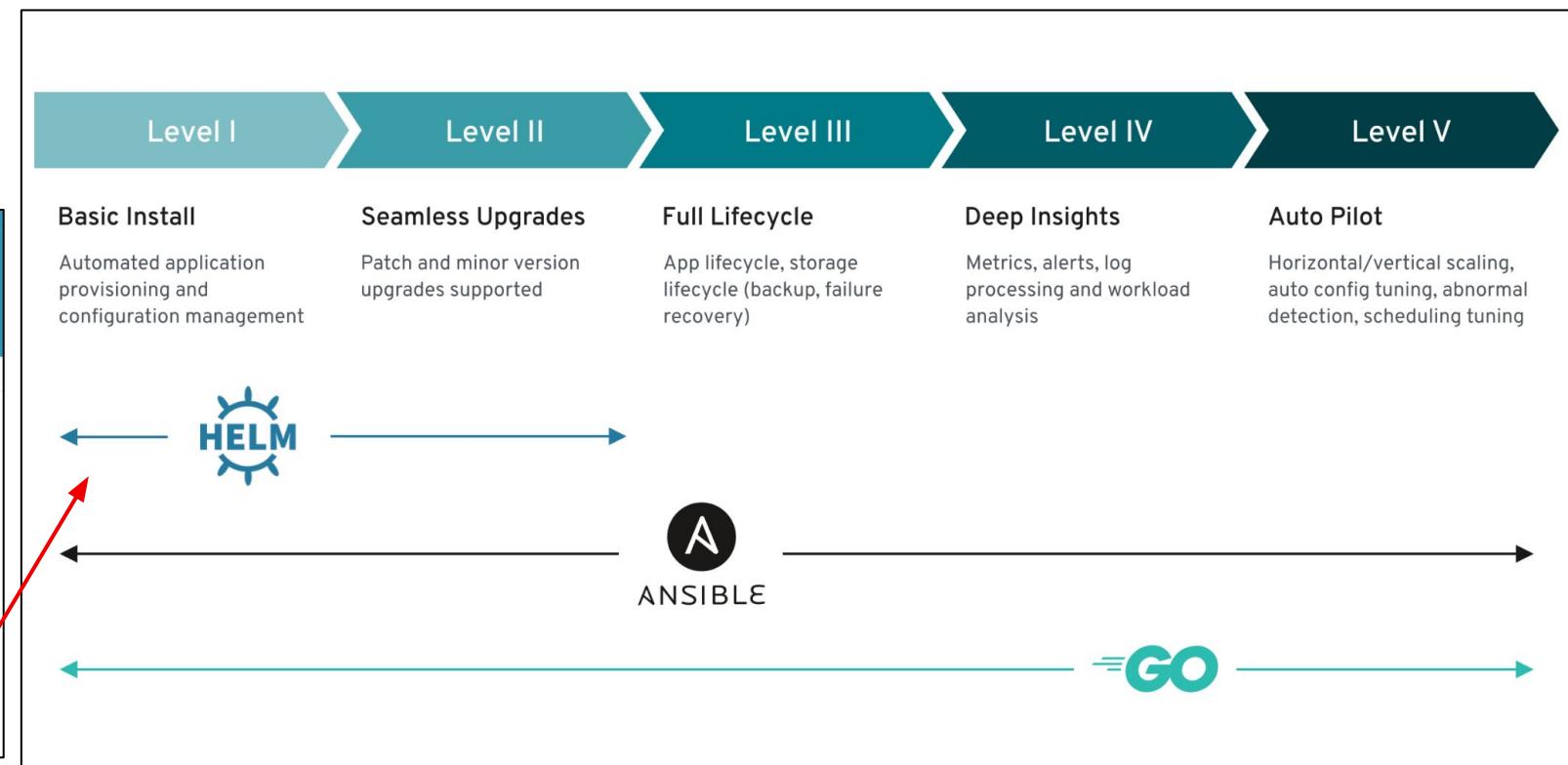
リソースが
新しく
定義できる

Helm CLI も
使えます。

Operator HubとOperatorの成熟度モデル

The image shows two screenshots side-by-side. On the left is the Red Hat OpenShift Container Platform interface, specifically the Operator Hub section under Catalog. It lists various operators like Skopeo Streams, Aqua Security Operator, and Automation Broker Operator. On the right is the OperatorHub.io website, also showing the etcd operator. Both screenshots highlight the etcd operator, which is described as creating and maintaining highly-available etcd clusters on Kubernetes.

K8s Operatorは提供するアプリケーションまたはワークロードのライフサイクル管理機能に関して、さまざまな成熟度を持っています。この機能モデルは、ユーザーが Operatorに期待できる機能レベルを表現しています。



OpenShift 4 Foundations 説明資料

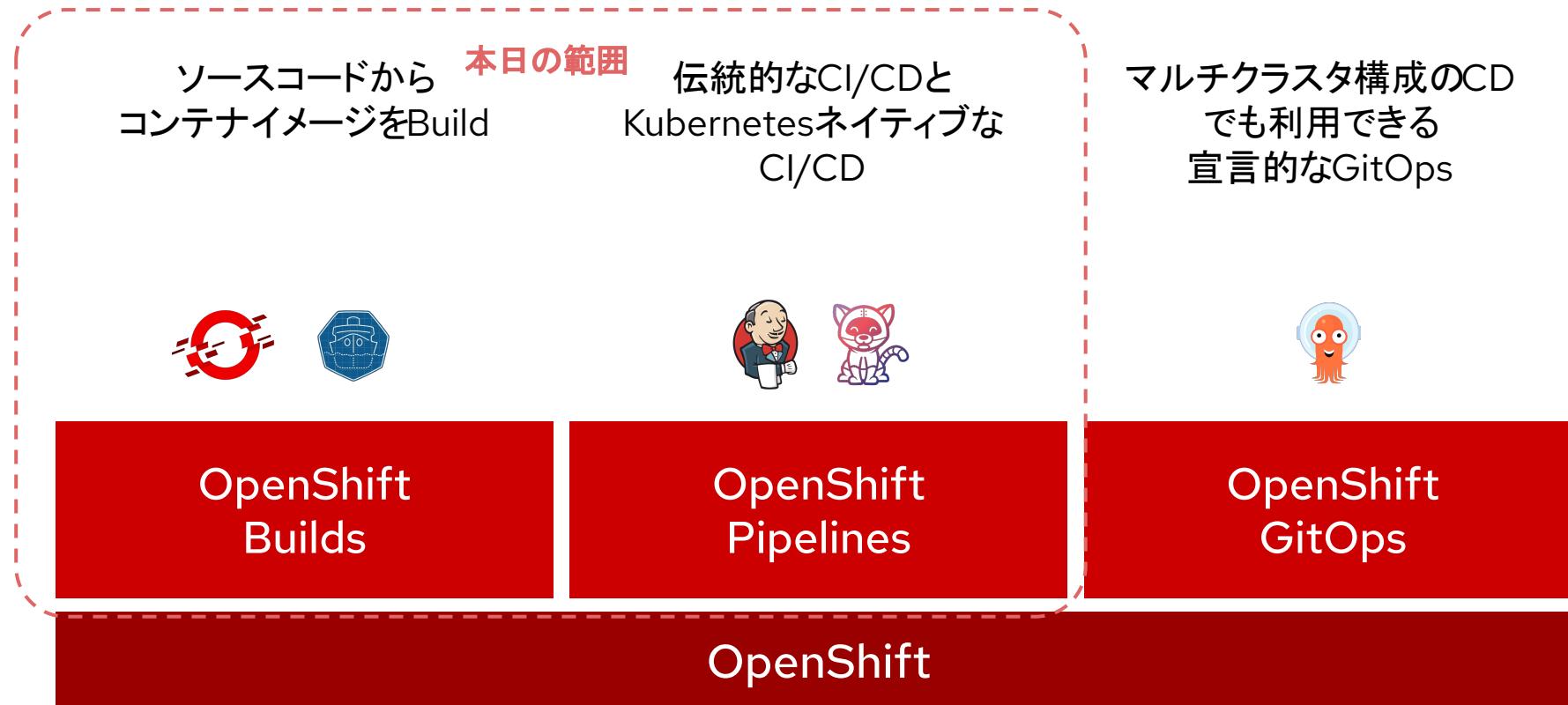
3. アプリケーションデプロイメント

アジェンダ

アプリケーションデプロイメント

1. OpenShift Builds
2. OpenShift Pipelines

開発・運用プロセス自動化の実現に向けたツール



OpenShift Builds

Buildプロセスの構築に関する機能

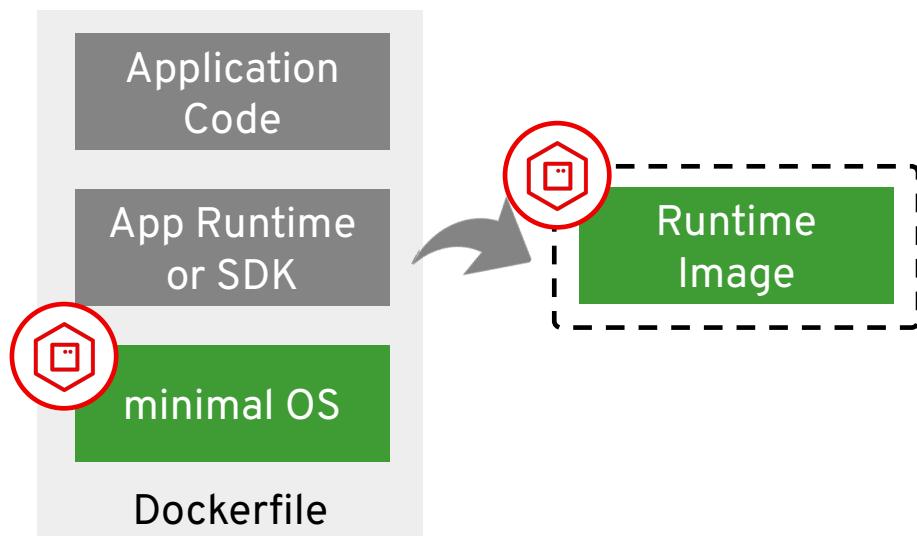
- ▶ S2I
- ▶ BuildConfig
- ▶ ImageStream

OpenShiftのBuild Strategy

Build Strategy on BuildConfig

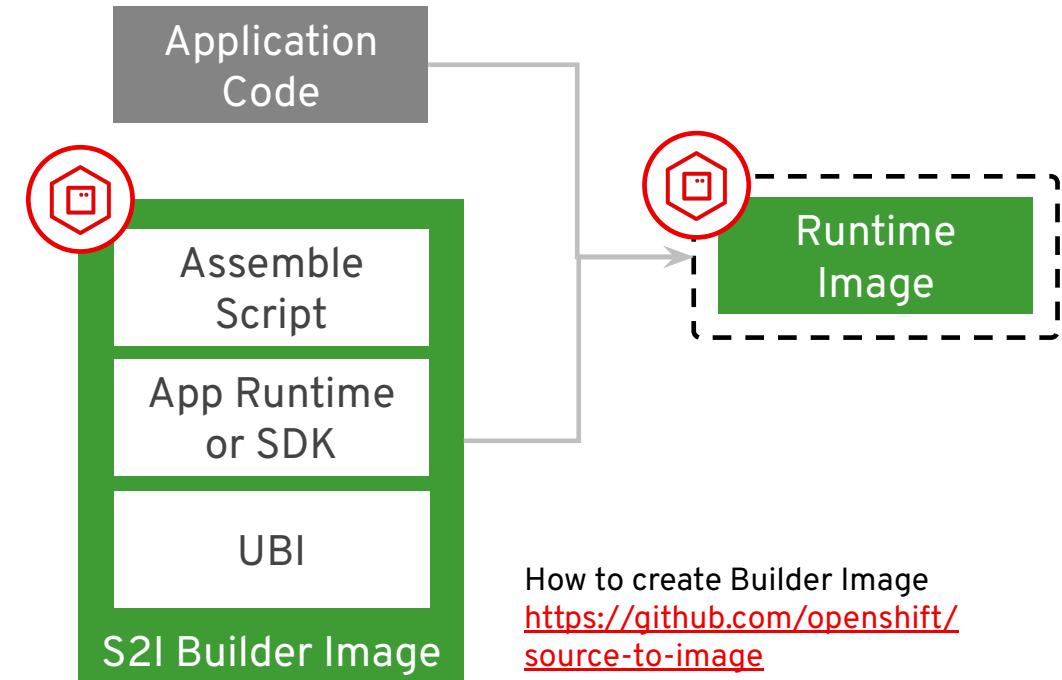
Docker Build

(Docker Strategy)



Source-to-Image(s2i) Build

(Source Strategy)

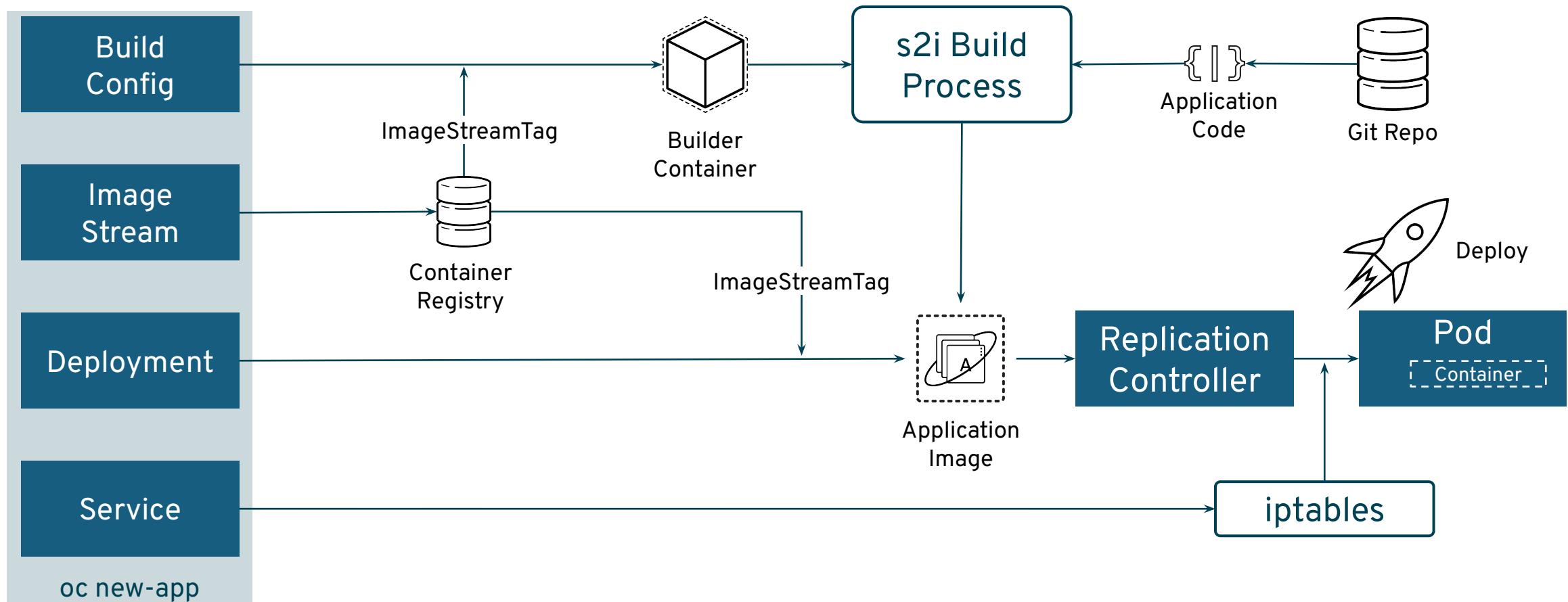


カスタマイズ性重視

保守/運用性重視

S2Iビルドプロセス

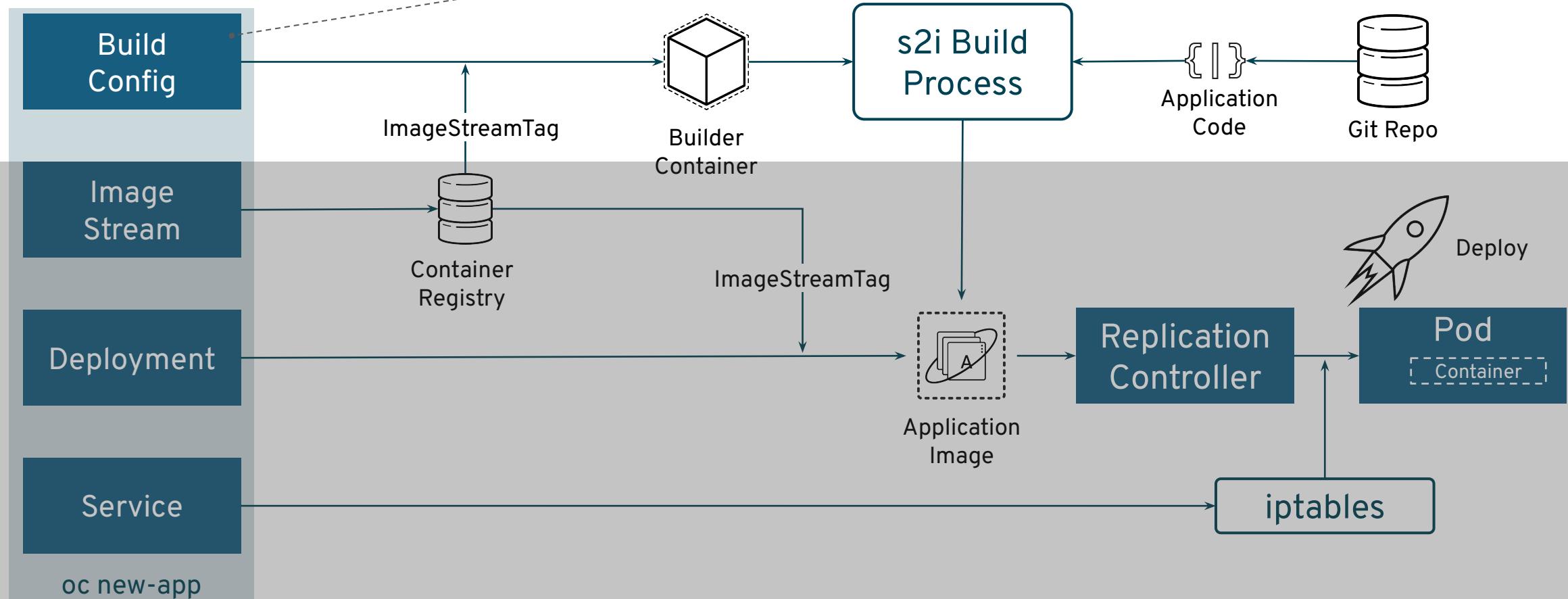
アプリケーションコードをInputにし、アプリケーションとベースイメージをビルドして、新しいコンテナイメージを生成



BuildConfigのStrategy (戦略)※
- Source Strategy (s2i) → 下の図の例
- Docker Strategy
- Custom Strategy
- Jenkins Pipeline Strategy

S2Iビルドプロセス

アプリケーションコードをInputにし、アプリケーションとベースイメージをビルドして、新しいコンテナイメージを生成



BuildConfig Object

- ・ビルトを呼び出すイベントを指定する
- ・アプリケーションソースをコンテナイメージに挿入し、新規イメージをアセンブルして実行可能なイメージを生成します。

spec	概要
triggers	新規ビルトを作成するトリガーの一覧を指定 ビルトが何によってトリガーされるかは、triggers:の配列を見れば分かります。
source	ビルトのソースを定義します。ソースの種類は以下です。 <ul style="list-style-type: none">・Git (コードのリポジトリの場所を参照)・Dockerfile (インラインのDockerfileからビルト)・Binary (バイナリーペイロードを受け入れる)
strategy	ビルトの実行に使用するビルトストラテジを定義
output	コンテナイメージが正常にビルトされた後に格納するコンテナレジストリの指定
postCommit	オプションのビルトフックを指定

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: "ruby-sample-build"
spec:
  runPolicy: "Serial"
  triggers:
    - type: "GitHub"
      github:
        secret: "secret101"
    - type: "Generic"
      generic:
        secret: "secret101"
    - type: "ImageChange"
  source:
    git:
      uri: "https://github.com/openshift/ruby-hello-world"
  strategy:
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "ruby-20-centos7:latest"
  output:
    to:
      kind: "ImageStreamTag"
      name: "origin-ruby-sample:latest"
  postCommit:
    script: "bundle exec rake test"
```

GitHubへのコミット(ソースコードの変更)をトリガーにしてビルトを開始できます

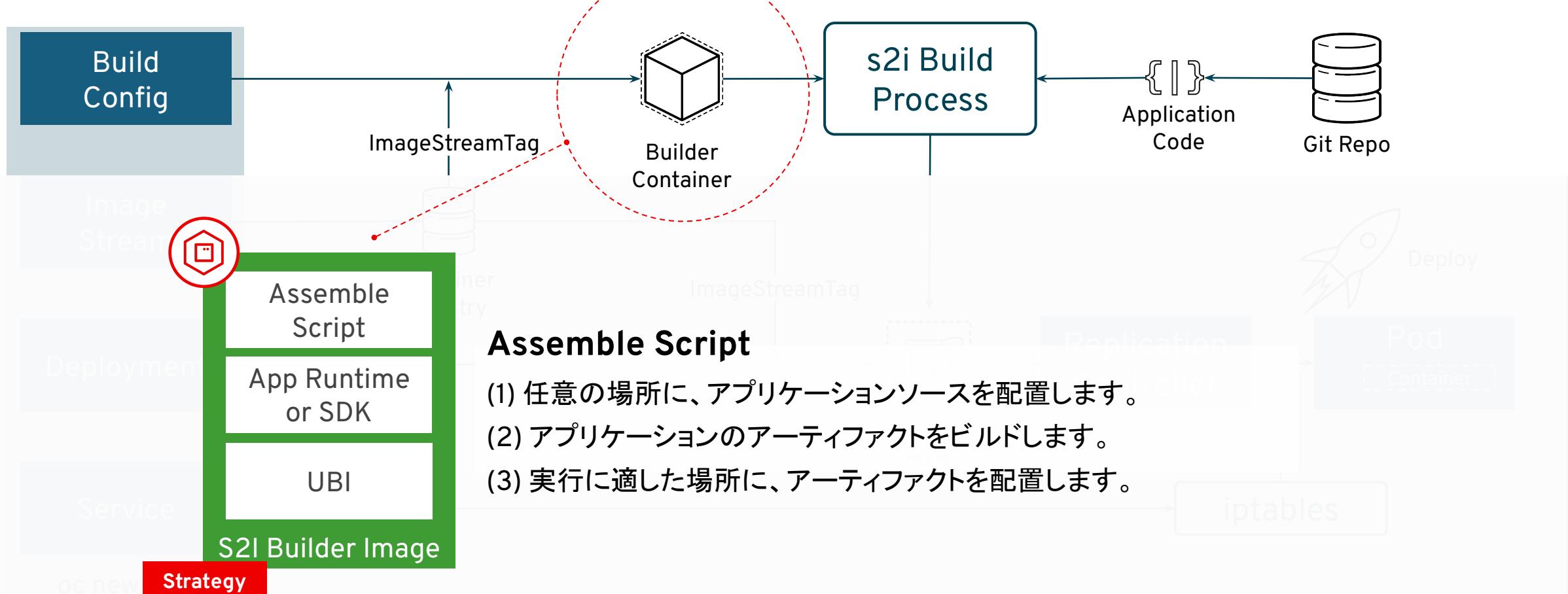
Builderイメージの変更を検知して、自動でruntimeイメージもビルトします※

アプリケーションソース

※Configが変わった場合をトリガーにビルトすることもできます。

S2Iビルドプロセス

アプリケーションコードをInputにし、アプリケーションとベースイメージをビルドして、新しいコンテナイメージを生成

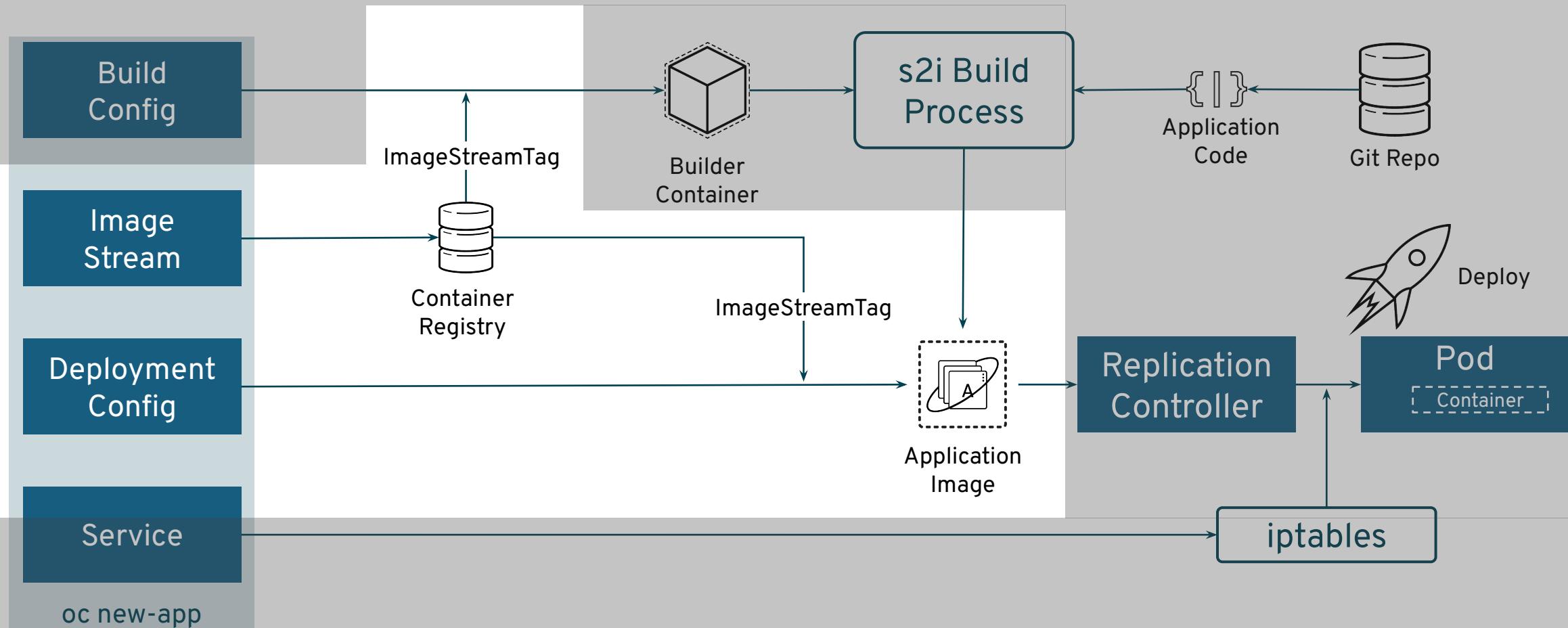


ImageStreamTag=rhel8/python-36:latest

Red Hat

S2Iビルドプロセス

アプリケーションコードをInputにし、アプリケーションとベースイメージをビルドして、新しいコンテナイメージを生成

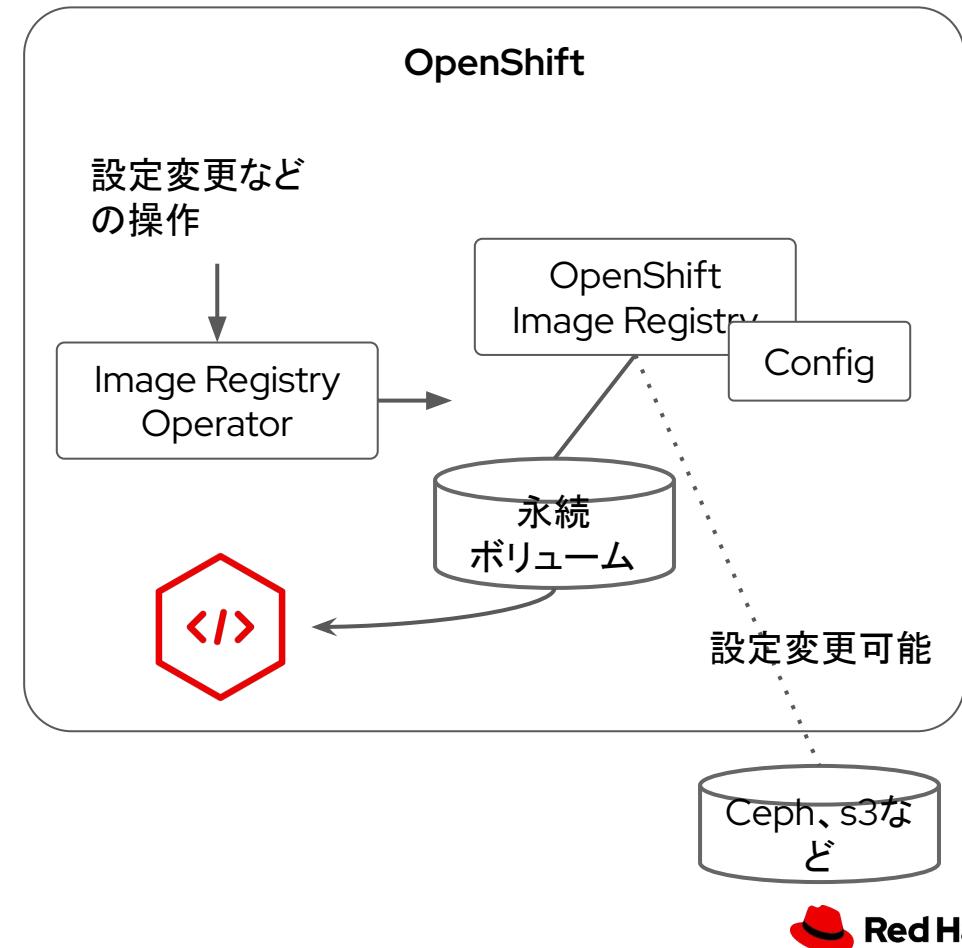


OpenShift Image Registry

Image Registry Operator で管理する OpenShift の内部レジストリ

- ▶ デフォルトでデプロイされるコンテナレジストリ
- ▶ OpenShift ユーザ、グループ、ロールに基づくアクセス制御が可能
- ▶ イメージセキュリティスキャンやジオレプリケーションなどの高度な機能が必要な場合は、Red Hat Quay Enterprise などより強力なエンタープライズレジストリサーバの採用が必要

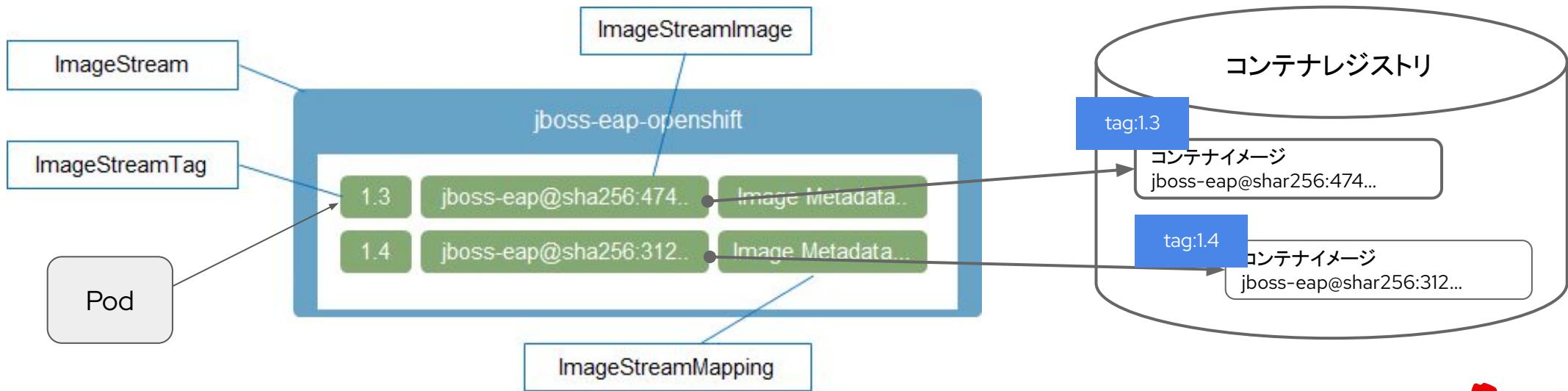
マニュアルでも Integrated OpenShift Container Registry や、Build-in container registry となっていたりと呼び方が一定していない。。。。



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

- OpenShift独自のAPIリソースの一つで、コンテナイメージへの参照の抽象化を提供
 - コンテナイメージへの参照やタグごとのコンテナイメージの履歴を保存
 - コンテナイメージのハッシュ値をチェックしてコンテナイメージを更新
- ImageStreamにはコンテナイメージそのものは含まれない

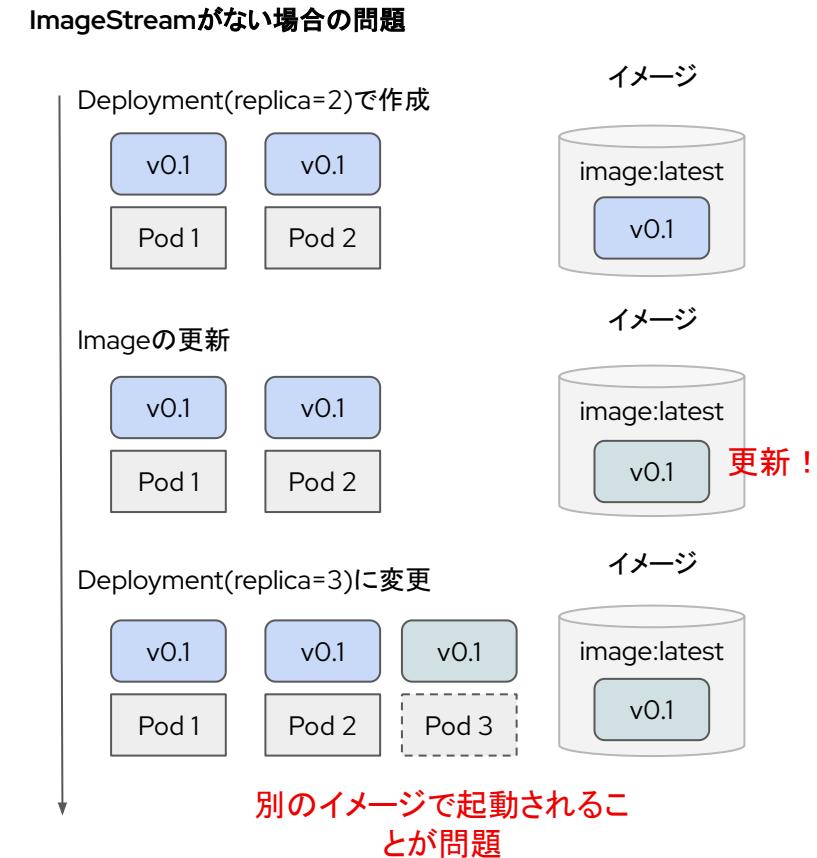


ImageStreamを使ううれしいこと

コンテナイメージのタグ管理を抽象化することで管理をしやすくする

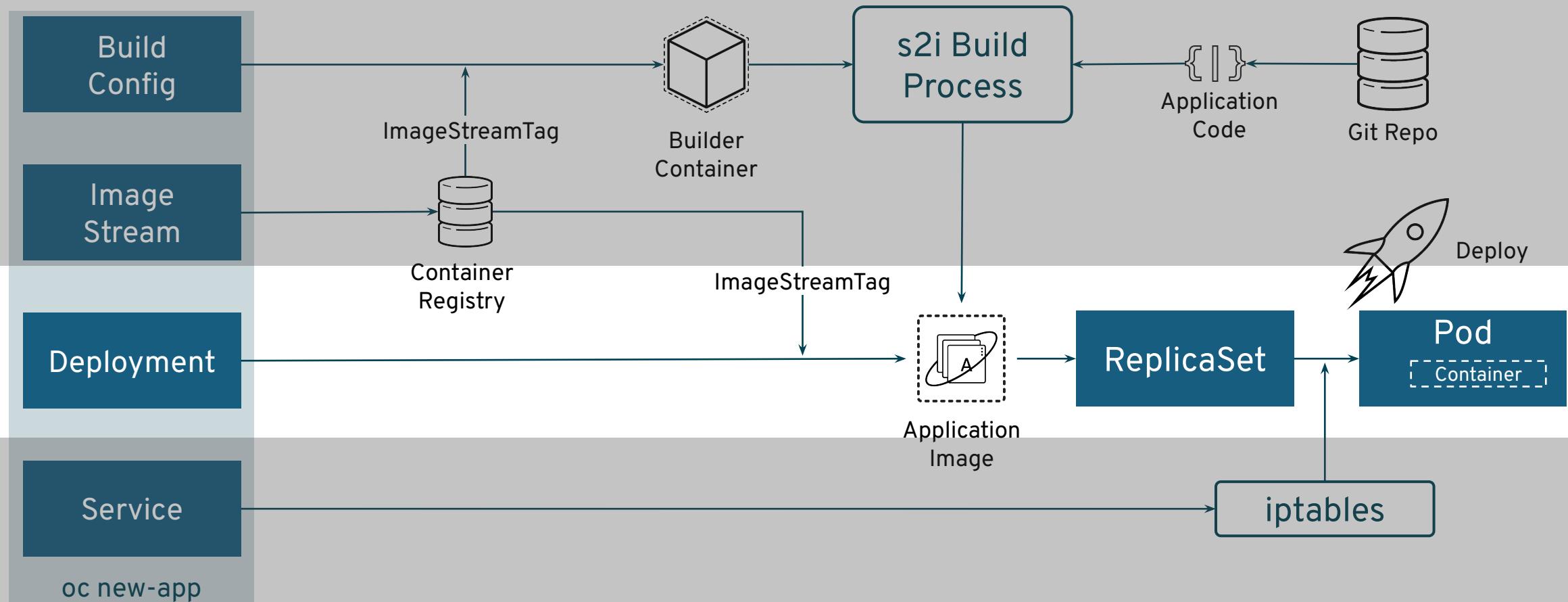
- ▶ (問題)あるDeploymentのコンテナイメージが更新 →スケールアウトした場合、イメージが異なるPodが立ち上がってしまう
 - ・ 通常、image:latestのような同じ「コンテナ:タグ名」のままコンテナを更新しても、名前が同じであるとDeploymentの更新は発生しない
 - ・ Image Streamを使うことで、同じ「コンテナ:タグ名」でもコンテナイメージのハッシュ値をチェックしてコンテナイメージを更新させる事ができます。

ご参考: [Using Red Hat OpenShift image streams with Kubernetes deployments](#)



S2Iビルドプロセス

アプリケーションコードをInputにし、アプリケーションとベースイメージをビルドして、新しいコンテナイメージを生成

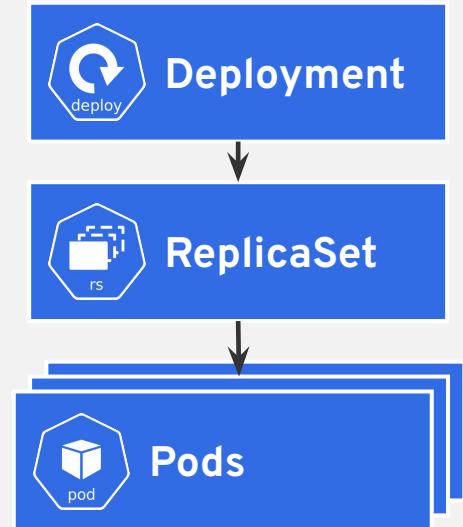


Deployment

Deploymentは Pod テンプレートとして、アプリケーションの特定コンポーネントの必要な状態を記述します。

Deployment は、PodのライフサイクルをオーケストレーションするReplicaSetを作成します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello.openshift
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello.openshift
  template:
    metadata:
      labels:
        app: hello.openshift
    spec:
      containers:
        - name: hello.openshift
          image: openshift/hello-openshift:latest
          ports:
            - containerPort: 80
```



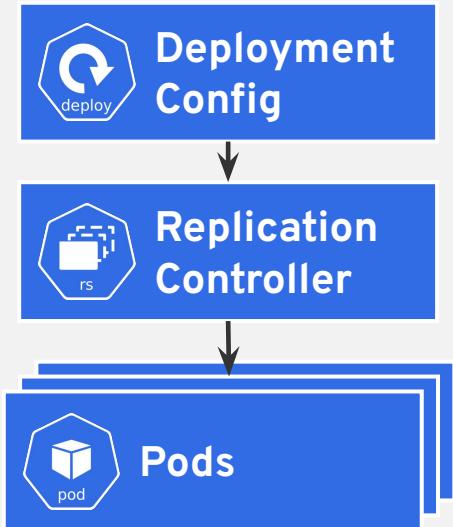
DeploymentConfig

DeploymentConfigは、アプリケーションを実行するためのテンプレート

■ Features

- ・イベントへの対応として自動化されたデプロイメントを駆動するトリガー
- ・直前のバージョンから新規バージョンに移行するためのユーザーによるカスタマイズが可能なデプロイメントストラテジー。各ストラテジーは通常デプロイメントプロセスと呼ばれ、Pod 内で実行されます。
- ・デプロイメントのライフサイクル中の異なる時点でカスタム動作を実行するためのフックのセット（ライフサイクルフック）。
- ・デプロイメントの失敗時に手動または自動でロールバックをサポートするためのアプリケーションのバージョン管理。
- ・レプリケーションの手動および自動スケーリング。

```
apiVersion: v1
kind: DeploymentConfig
metadata:
  name: frontend
spec:
  replicas: 5
  selector:
    name: frontend
  template: { ... }
  triggers:
    - type: ConfigChange 1
    - imageChangeParams:
        automatic: true
        containerNames:
          - helloworld
        from:
          kind: ImageStreamTag
          name: hello-openshift:latest
      type: ImageChange 2
  strategy:
    type: Rolling
```



Drive automated deployments

1. **ConfigChange**
2. **ImageChange**

Drive automated deployments

1. **Rolling Strategy**
2. **Recreate Strategy**
3. **Custom Strategy**

DeploymentConfigとDeployment はどちらを使うのか？

DeploymentConfigで提供される特定の機能または動作が必要でない場合、Deploymentを使用することが推奨

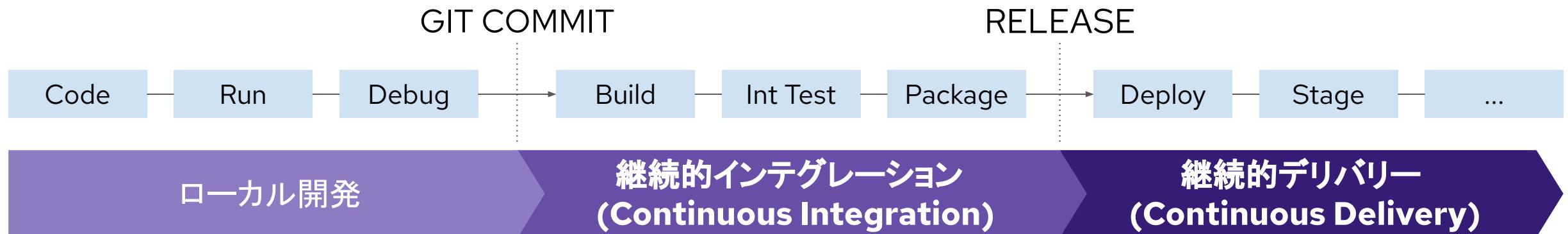
DeploymentConfig固有の機能		Deployment固有の機能	
Automatic Rollbacks	(OCP4.3時点) Deploymentでは、問題の発生時の最後に正常にデプロイされたReplicaSetへの自動ロールバックは、サポートしていません。	Rollover	Deploymentのプロセスは、Controller Loopで実行されます。つまり、Deploymentは任意の数のアクティブな、ReplicaSetを指定することができます、デプロイメントコントローラーがすべての古いReplicaSetをスケールダウンし、最新のReplicaSetをスケールアップします。 一方、DeploymentConfigでは、新規のrolloutにdeployer Podを利用します。
Trigger	Deploymentの場合、DeploymentのPod Templateに変更があるたびに、新しいrolloutが自動的に行われるため、暗黙的な「ConfigChange」トリガーが含まれます。 Pod Templateの変更時にrolloutが不要な場合には、明示的にコマンドで停止します。	Proportional scaling	Deploymentコントローラーが新旧のReplicaSetのreplica値について保証するため、継続中のrolloutのスケーリングが可能です。追加のレプリカはそれぞれのReplicaSetのreplica値に比例して分散されます。
Lifecycle-Hooks	DeploymentではLifeCycleHooksをサポートしていません。	Pausing mid-rollout	Deploymentはいつでも一時停止できます。つまり、継続中のrolloutも一時停止できます。一方、DeploymentConfigのDeployer Podは現時点での一時停止できないので、rolloutを一時停止しようとしても、デプロイプロセスは影響を受けず、完了するまで続行されます。
CustomStrategy	Deploymentでは、ユーザーが指定するCustom Deployment Strategyをサポートしていません。		

OpenShift Pipelines

CI/CDとは

継続的インテグレーション (Continuous Integration) と
継続的デリバリー / デプロイメント (Continuous Delivery/Deployment) の略語

- ソフトウェアの変更を常にテストして自動で本番環境にリリース可能な状態にしておく、ソフトウェア開発の手法。
- CI/CDを実践することで、バグを素早く発見したり、変更を自動で反映してリリースしたりできるようになり、ユーザーに価値を届けるサイクルを早く回せるようになる。



OpenShiftで提供するCI/CDツール

OpenShift Pipelines



OpenShift GitOps



OpenShift Pipelines



Built for Kubernetes

- Kubernetesネイティブな宣言型CI/CDツール
- 中央サーバーの管理、プラグインの調整からの脱却



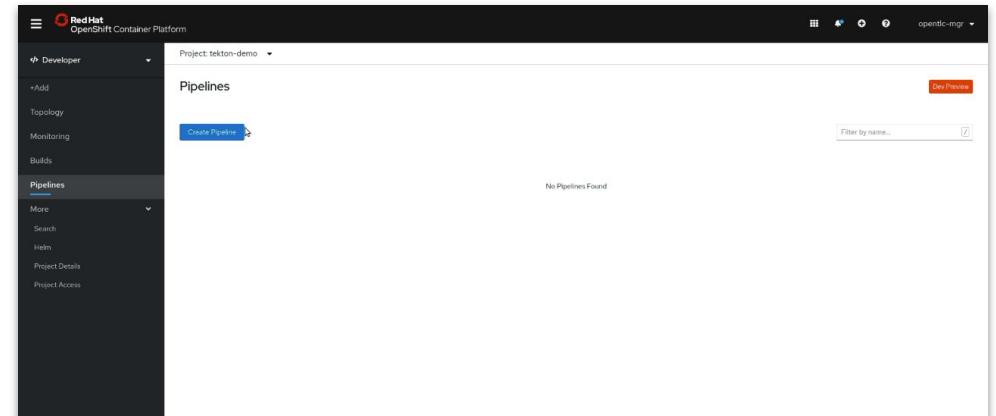
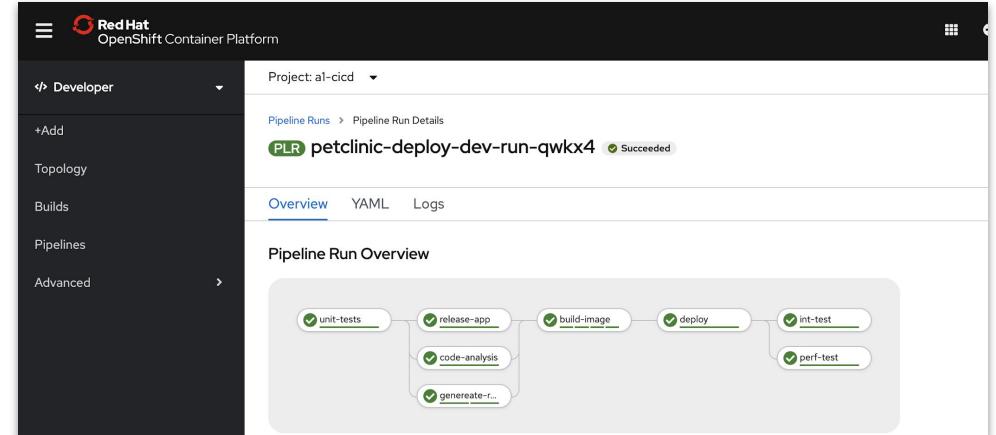
独立した実行環境

- 独立したコンテナ内で実行・拡張されるパイプライン
- 再現性のある予測可能な結果



高いユーザビリティ

- OpenShiftのGUIや専用CLIを使ったパイプライン作成
- OperatorHubを使ったインストール・アップグレード
- 既存TaskとPipelineの利用(Tekton Hub)



Traditional CI/CDとCloud-Native CI/CD

Traditional CI/CD

任意の環境(物理, VM, コンテナ)で利用可能

CIツールのインストールと保守運用が必要

CIツールの利用は共有
(プラグイン構成とバージョンに縛られる)

アップデートのタイミングに制約

設定はCIツールごとに固有

Cloud-Native CI/CD

Kubernetes環境に特化

パイプラインの実行にCIツールは不要

CIツールの利用は独立
(各パイプラインは独立な構成)
パイプライン構成タスクのバージョンは
パイプライン毎に独立
設定はKubernetes Native
(Custom Resource)



Jenkins



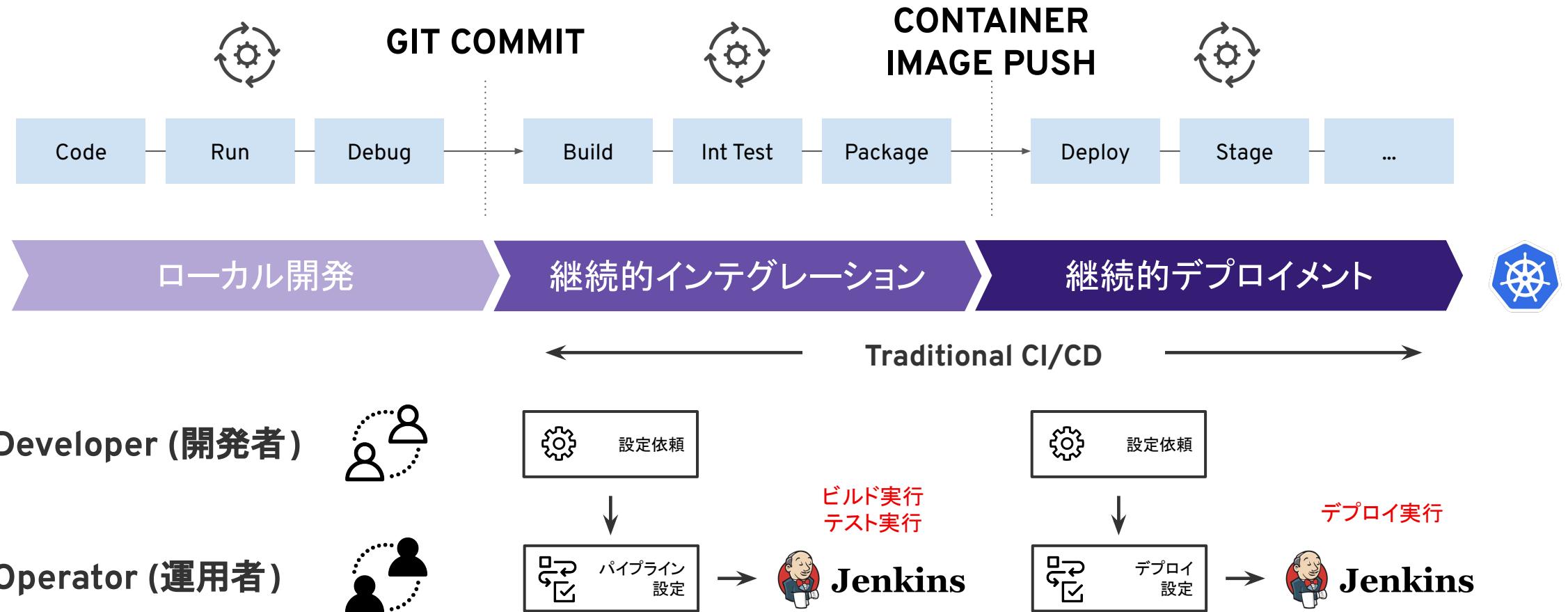
TEKTON



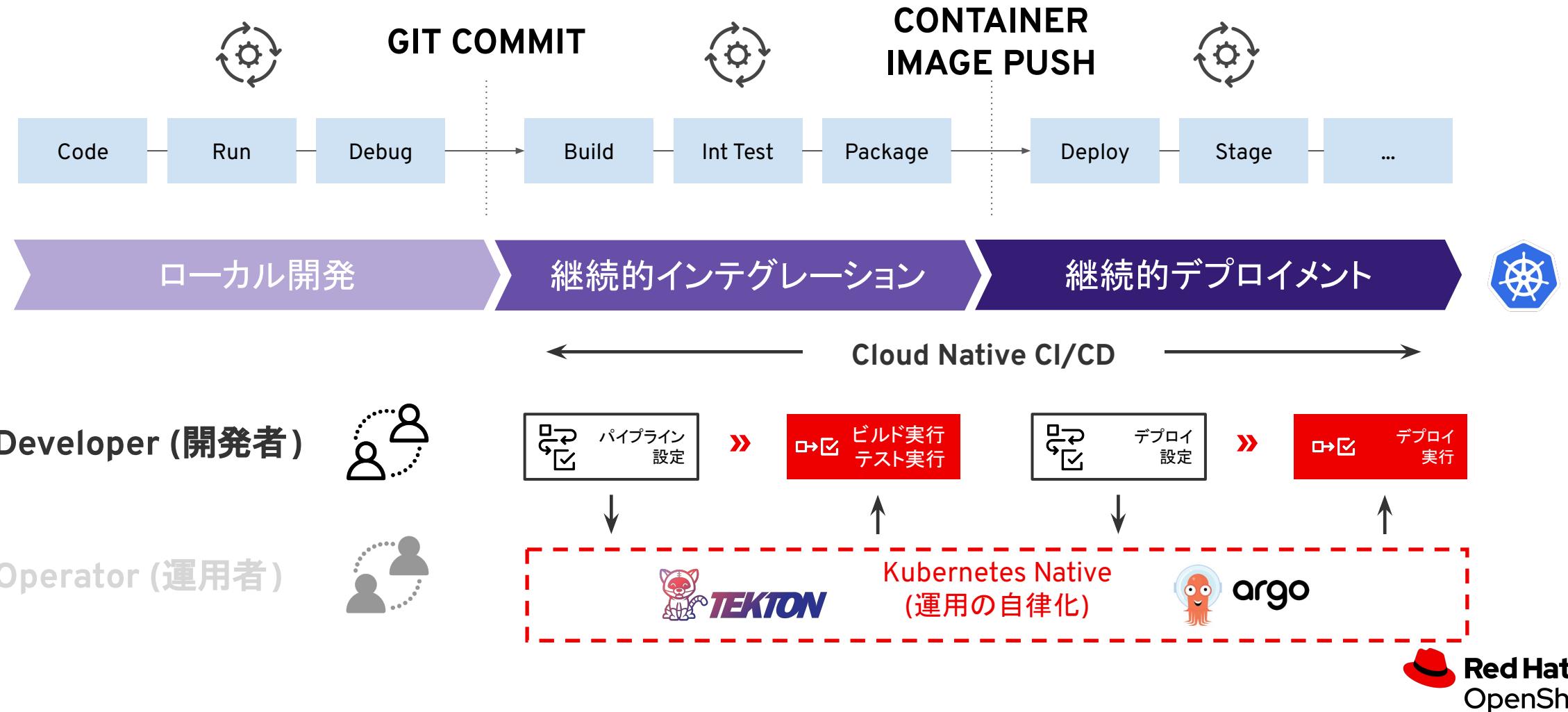
argo

Red Hat
OpenShift

Traditional CI/CD



Cloud Native CI/CD



OpenShift Pipelines 主なカスタムリソース



TEKTON

Tekton のキーワード

Step

一つのコンテナイメージを利用し
て処理を実行

Task

複数のStepを逐次的に同じPod
で実行

Pipeline

Taskのある順番で実行するた
めのパイプライン

Task Run

Input/Outputを取り扱いながら
Taskを起動

Pipeline Run

Input/Outputを取り扱いながら
Pipelineを起動

Steps

- 一つのコンテナでコマンドやスクリプトを実行
- Kubernetesコンテナのリソースも利用可
 - Env vars
 - Volumes
 - Config maps
 - Secrets

```
- name: build  
  image: maven:3.6.0-jdk-8-slim  
  command: ["mvn"]  
  args: ["install"]
```

```
- name: parse-yaml  
  image: python3  
  script: |-  
    #!/usr/bin/env python3  
  
    ...
```

Task

- 実行する処理の単位で定義
- 複数のStepを連続的に実行
- Task pod の中で実行
- Input、Output、パラメータの管理
- Workspaces と results でデータを共有
- Pipelineとは独立して稼働も可能
- ClusterTaskは全Namespaceで利用可能なTask

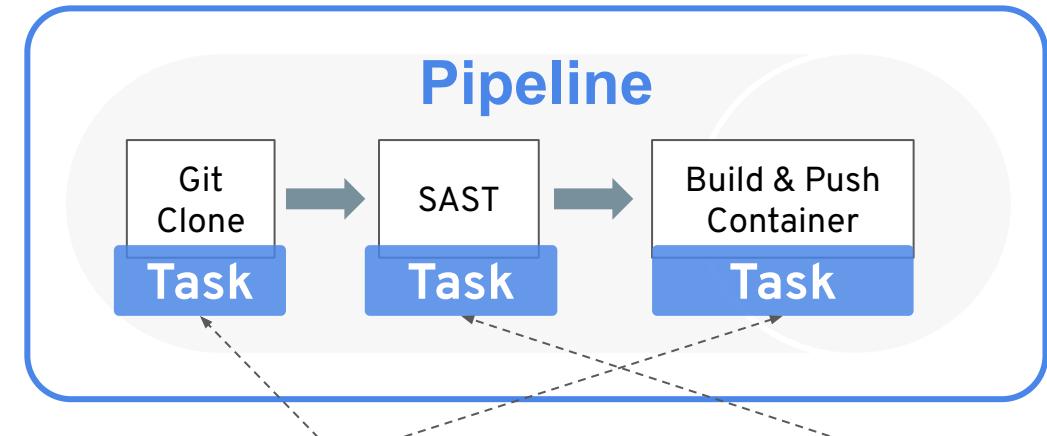


Taskの例: Maven Install, AWS CLI, Kubectl Deploy, Security Scan, etc



Tekton Catalog/Tekton Hubの活用

- Operatorインストール時にTekton Catalogで公開されているTaskの一部がClusterTask(全てのNamespaceで利用可能なTask)として登録
- デフォルトで登録されていないTaskであってもtknコマンド等で容易に追加可能



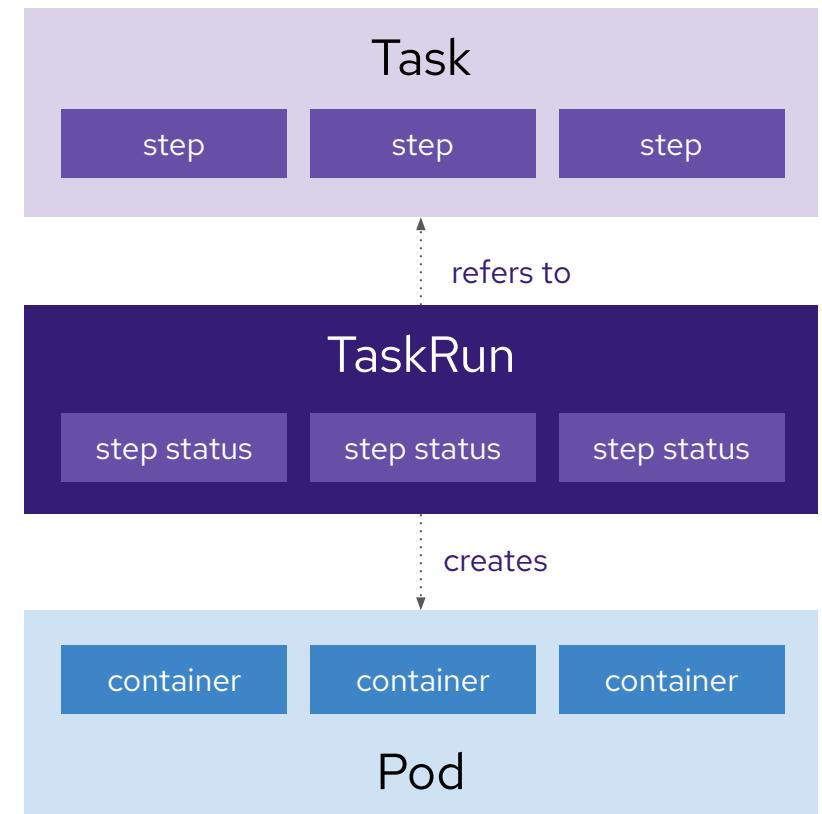
```
# tkn hub install task sonarqube-scanner --version 0.1
Task sonarqube-scanner(0.1) installed in cicd namespace
```

Task (Maven Taskの例)

```
kind: Task
metadata:
  name: maven
spec:
  params:
    - name: goal
      type: string
      default: package
  steps:
    - name: mvn
      image: maven:3.6.0-jdk-8-slim
      command: [ mvn ]
      args: [ $(params.goal) ]
```

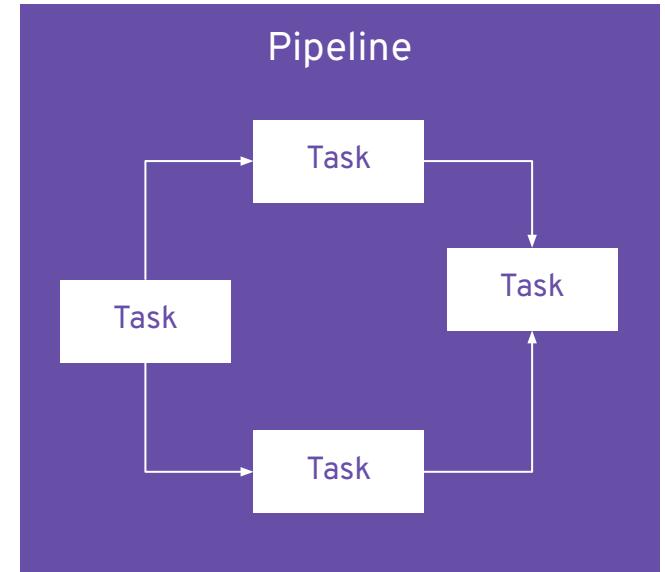
TaskRun

- Pod内でTaskの完了まで実行する
- Task の参照または埋め込み
- TaskのInput情報の引き渡しの定義
 - Parameters
 - Resources
 - Service account
 - Workspaces



Pipeline

- Taskの実行順序(グラフ)の定義
- 入力とパラメータ
- Taskの再実行
- 条件付きのTask実行
- Task間でデータを共有するためのWorkspace
- Project間での再利用





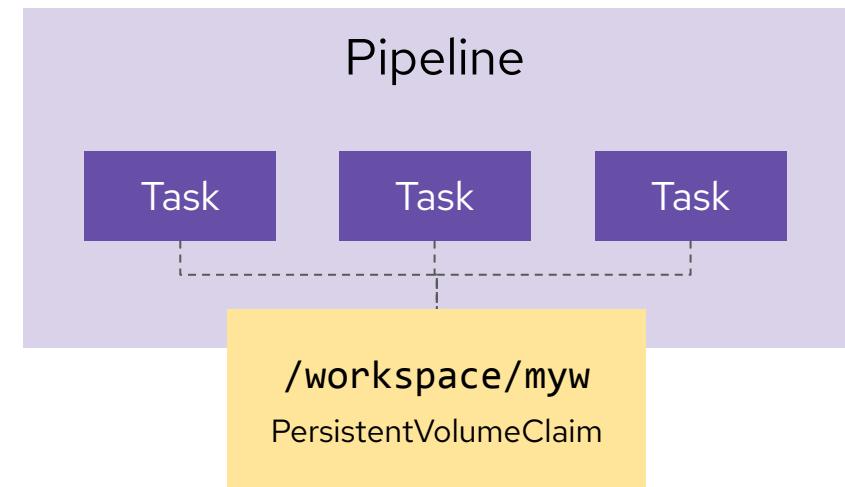
Task間でのデータの共有

Task: results

- Taskはデータを変数として公開する
- 小規模なデータに適している
- 例:コミットID、ブランチ名

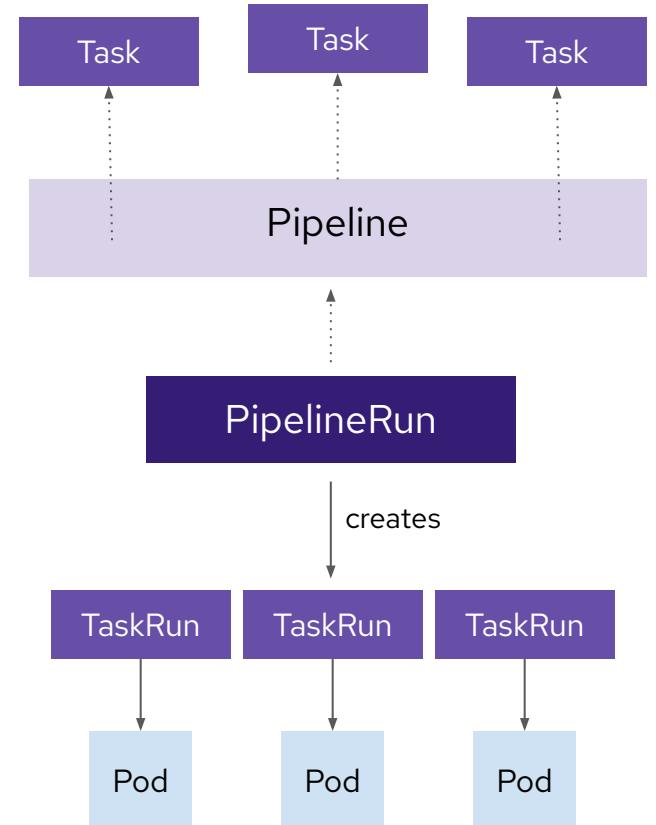
Task: workspaces

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



PipelineRun

- Pipelineを完了まで実行
- Pipeline内のタスクを実行するTaskRunを作成
- PipelineにInputとパラメータを提供
- 宣言されたPipelineのWorkspaceにVolumeを提供



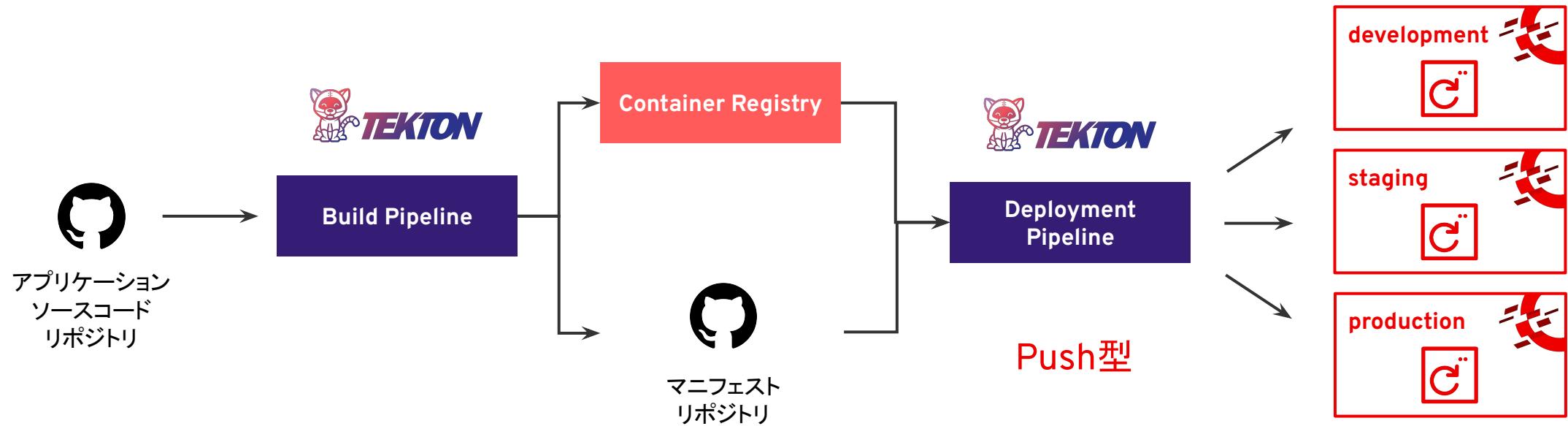
OpenShift GitOps

(超概要)

OpenShift Pipelineでデプロイメントは行わない

継続的インテグレーション

継続的デプロイメント



CI Opsは注意が必要

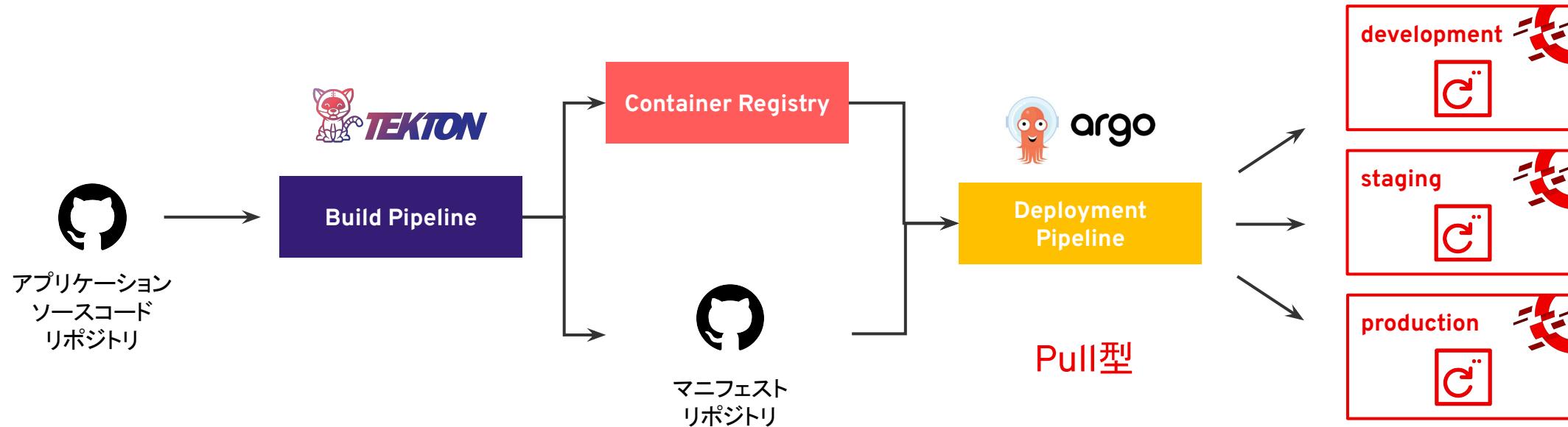
- ・CIツールに各クラスタへのデプロイ権限が必要
- ・構成ドリフト(コードと実態の乖離)を検知しない
- ・クラスタと疎通するためのネットワーク設定が必要

Deployment Pipelineを作成することは可能だが、デプロイ先の状態変化を検知してデプロイするわけではない。

OpenShift PipelineとGitOpsを組み合わせた CI/CD

継続的インテグレーション

継続的デプロイメント



効率的なデプロイを実現

- ・CDツールは限定的なデプロイ権限のみを保持
- ・Gitリポジトリへのアクセス(Readのみ)
- ・構成ドリフトを検知し、自動的に修正

GitOpsにすることで、デプロイ先の状態変化を動的に検知し、定義された状態を持続する。

ハンズオン 概要



OpenShift 4 ワークショップ ハンズオンガイド

OpenShift 概要

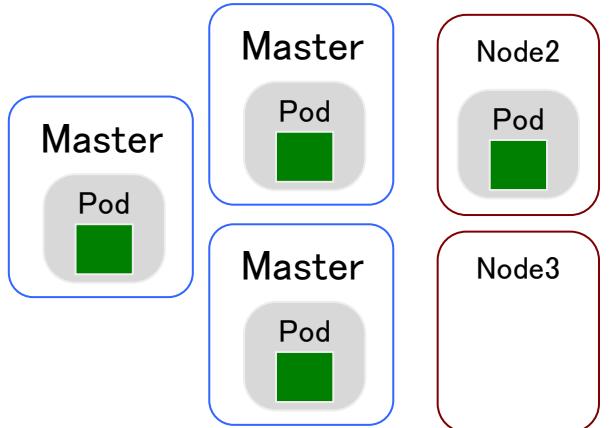
- 1. ハンズオン環境について**
- 2. 実施いただく内容**

- 1. ハンズオン環境について**
2. 実施いただく内容

OpenShiftハンズオン環境: OCPクラスタ構成

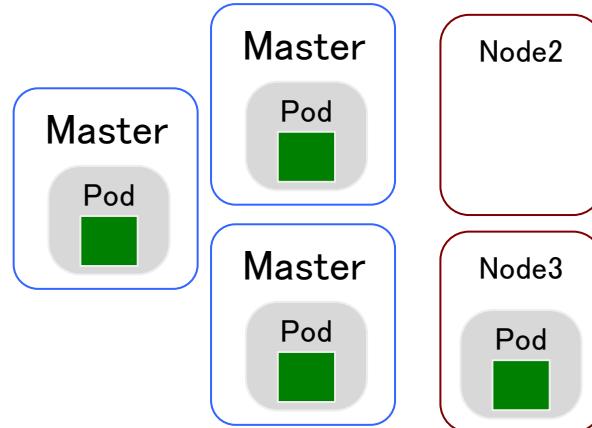
OCP Cluster 1

admin/user1/user2../user5



OCP Cluster 2

admin/user1/user2../user5

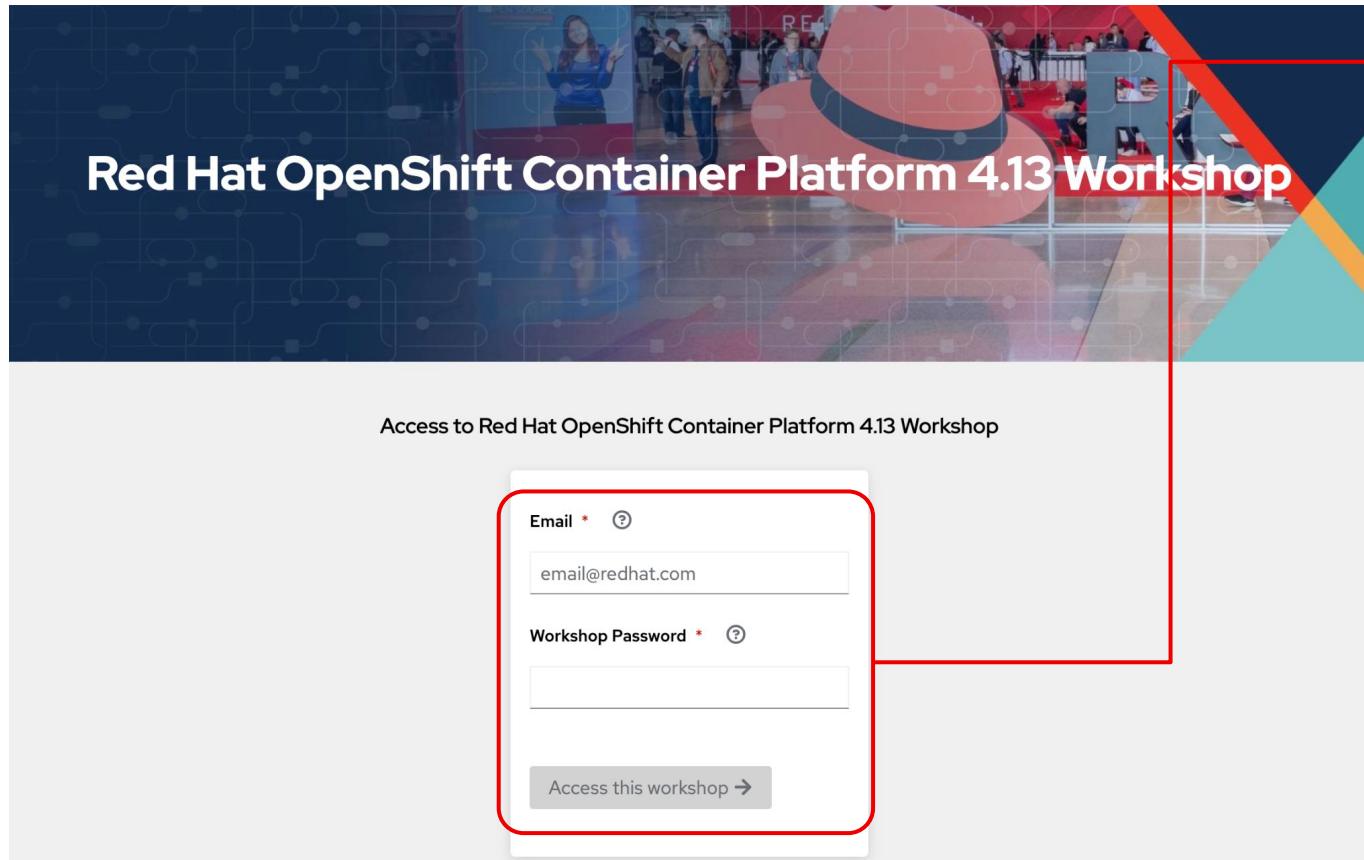


- 複数のOpenShiftクラスタ上に参加者向けの個別のユーザを作成しております。
- 個別に”プロジェクト”を作成する**ことにより、複数のユーザが干渉せずに個別にアプリケーションの実行やパイプラインの適用を実施いただけます。
- ハンズオンガイドの一部操作に関しては、admin権限が必要となるため、講師が実施済みです。この操作(後のスライドにて説明)に関しては、飛ばして先にお進みください。
- OpenTLCのアカウントをお作りいただけますと、本日のハンズオン環境(“Hands On with OpenShift 4.10”)や他のRed Hat製品稼働環境を**ご自身でお好きなタイミングで作成**し、OpenShiftの機能(admin権限含め)をお試しいただけます。ご活用ください！

ハンズオン環境のデプロイ

1) 下記ラボ環境払い出しURLにアクセスしてください

https://red.ht/20250318_lab



2) 下記を入力後、「Access this workshop」を押してください

Email: メールアドレス
(ユーザを一意に識別するためですの
で、なんでも構いません)

Workshop Password
講師からお知らせします

1. ハンズオン環境について

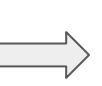
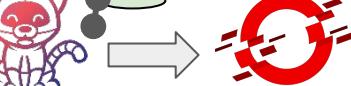
2. 実施いただく内容

下記ガイドに沿って実施ください

<https://red.ht/ocp4labguide>

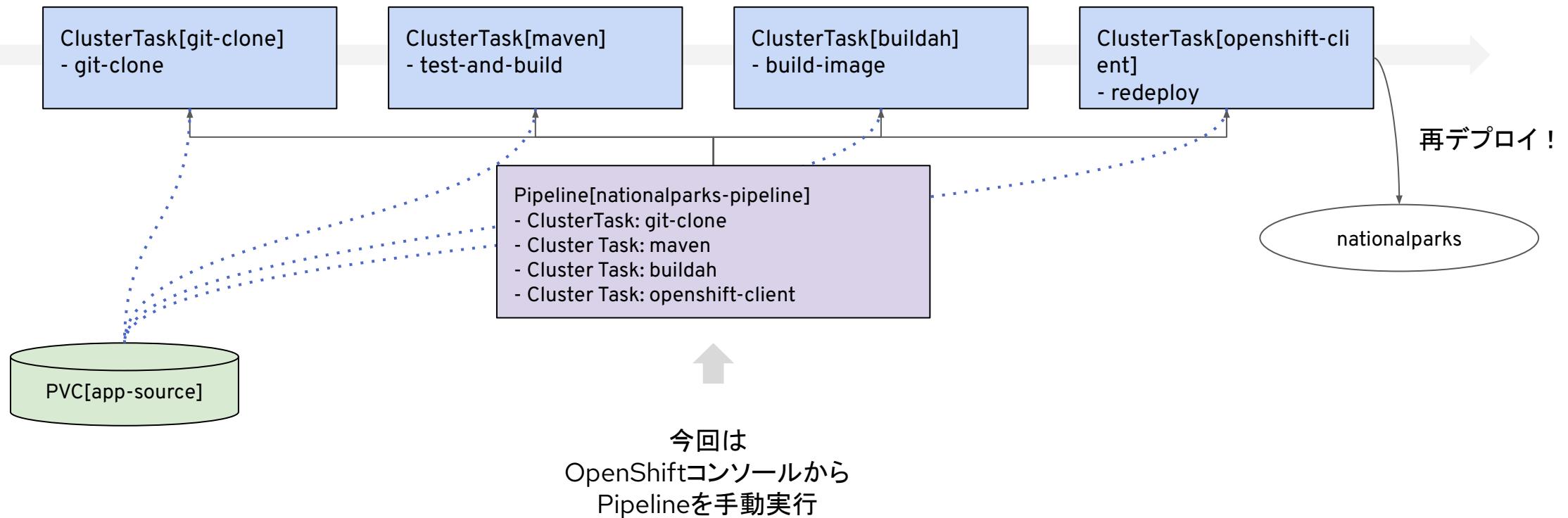
ハンズオンコンテンツ

～17:45まで

	Chapter 3: OpenShift ユーザエクスペリエンス	Chapter 4: アプリケーションデプロイメント基礎 (s2i,Tekton)
1	Gitからのアプリケーションデプロイメントx 2  → 	Gitからのアプリケーションデプロイメント  → 
2	モニタリング(テレメトリ)機能によるPodの状態確認(Prometheus) 	Pipeline用途(workspace)のストレージの作成・接続  → 
3	<u>スキップ対象</u> : Pipelineオペレータのインストール(講師がadminアカウントで実施済み)   → 	Pipelineを用いたアプリケーションのビルド・デプロイ  →  → 

アプリケーションデプロイメント基礎(s2i,Tekton)

作成するリソース



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.

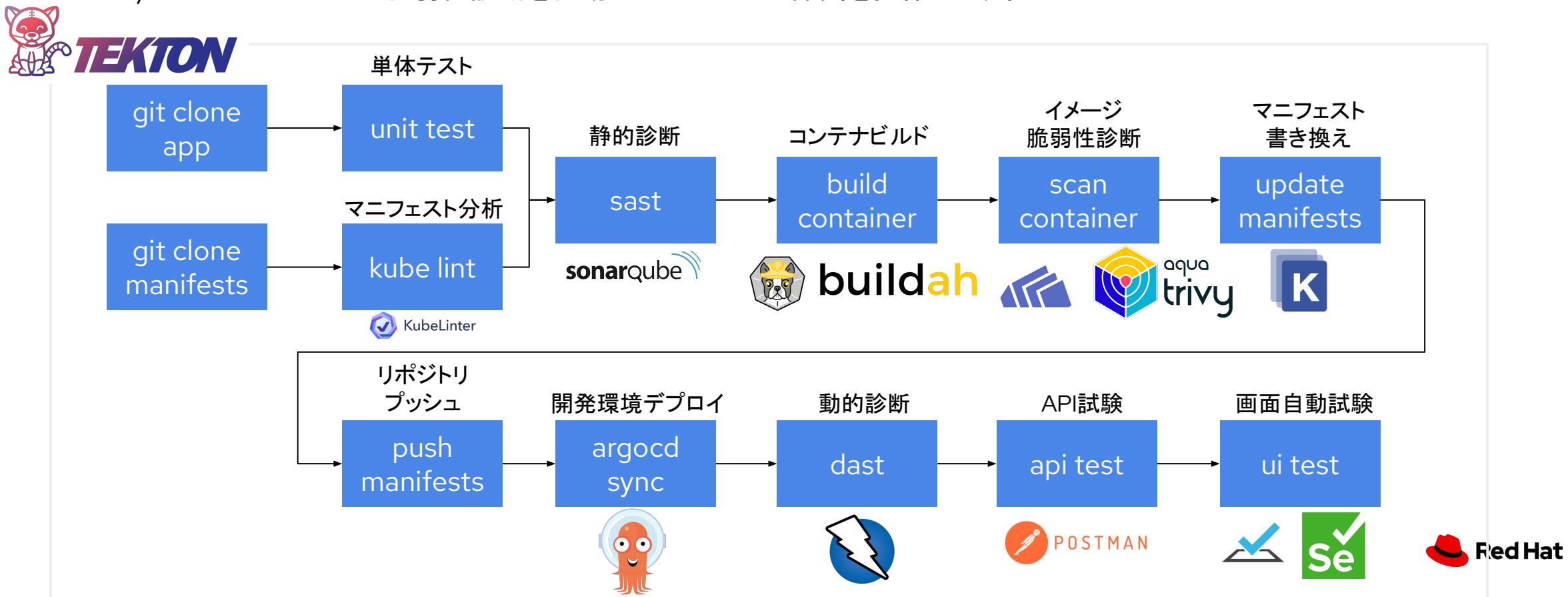


Appendix.

CIで実施すべきアクション

アプリケーションの品質担保を自動化

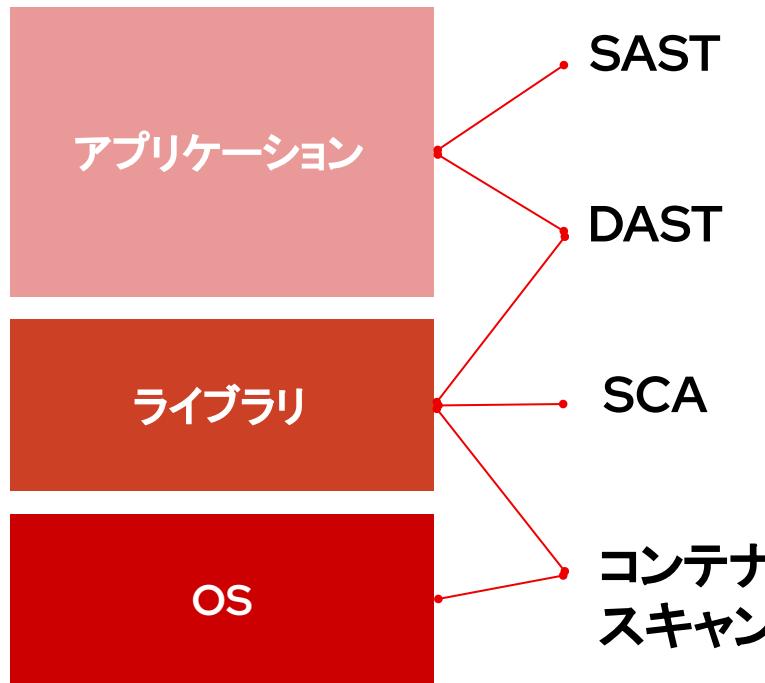
CIパイプラインを実行することで、直近で追加したコードへのテストだけでなく、デグレーションの防止や、アプリケーション/コンテナレベルでの脆弱性診断を自動化しシステムの品質を担保します。



DevSecOps

開発段階でのセキュリティ対策

CIパイプラインの中にセキュリティ診断ツールを組み込み、開発早期において脆弱性を特定・修正していくプラクティスをDevSecOpsと呼びます。DevSecOpsを実践することで商用環境でバグや脆弱性が見つかるリスクを低減し、対応に必要なインパクトを最小化できます。



アプリケーションのソースコードに対するセキュリティ診断。ホワイトボックスセキュリティテスト。

環境にデプロイされ稼働中のアプリケーションに対するセキュリティ診断。ブラックボックスセキュリティテスト。

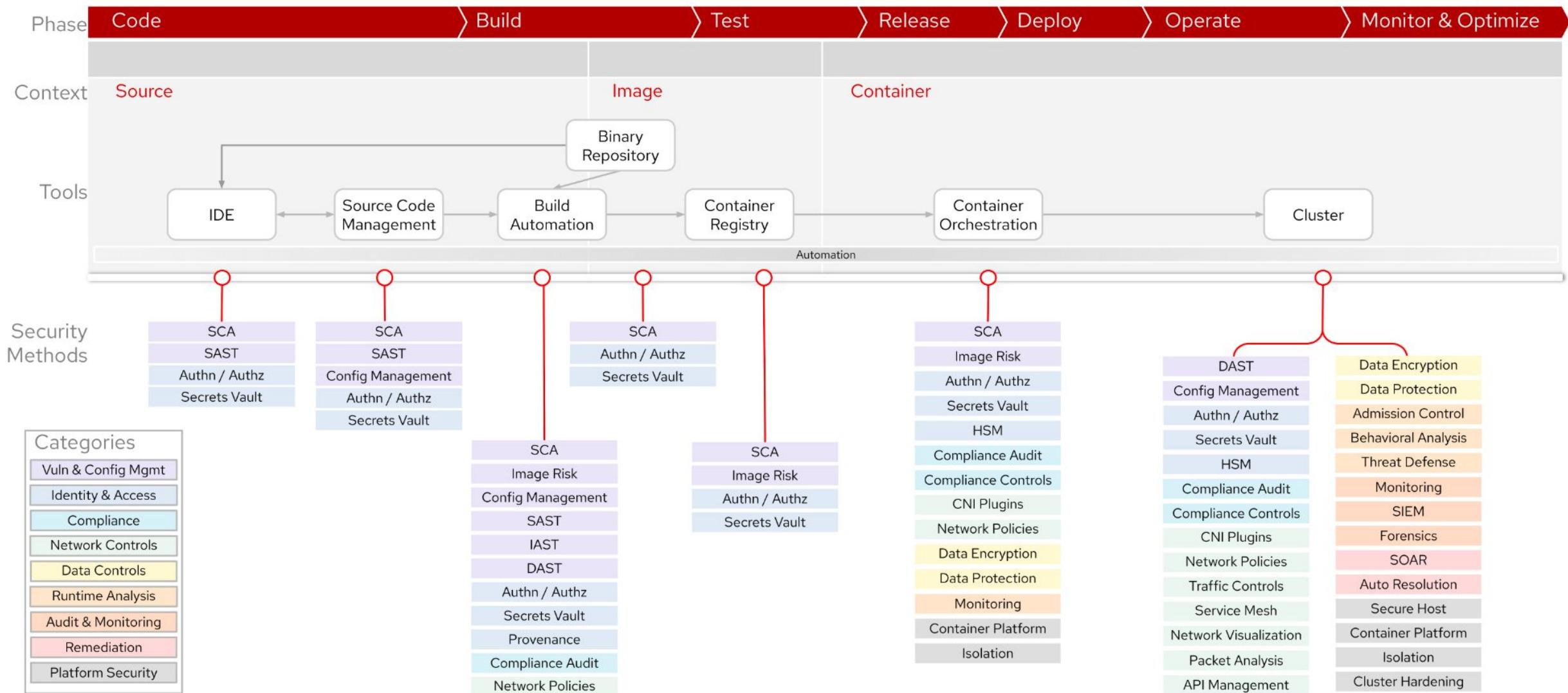
アプリケーションの中で使用しているOSSのライセンス違反検知。

ベースコンテナイメージや、追加されたライブラリに含まれる脆弱性の検知。

Tools

- SonarQube
- OWASP ZAP
- OWASP Dependency-check
- Red Hat Dependency Analytics
- Red Hat Advanced Cluster Security
- Clair
- Trivy

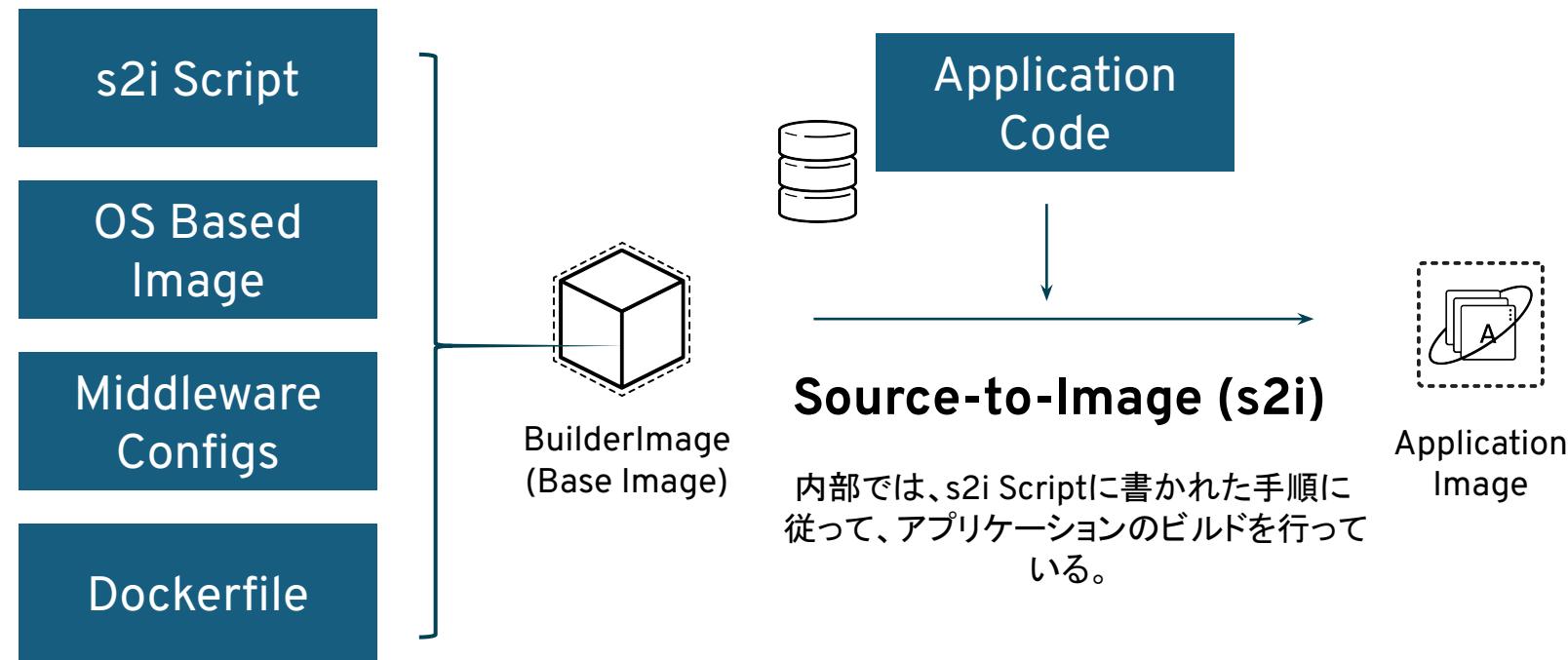
DevSecOps Framework



Source-to-Image (s2i)

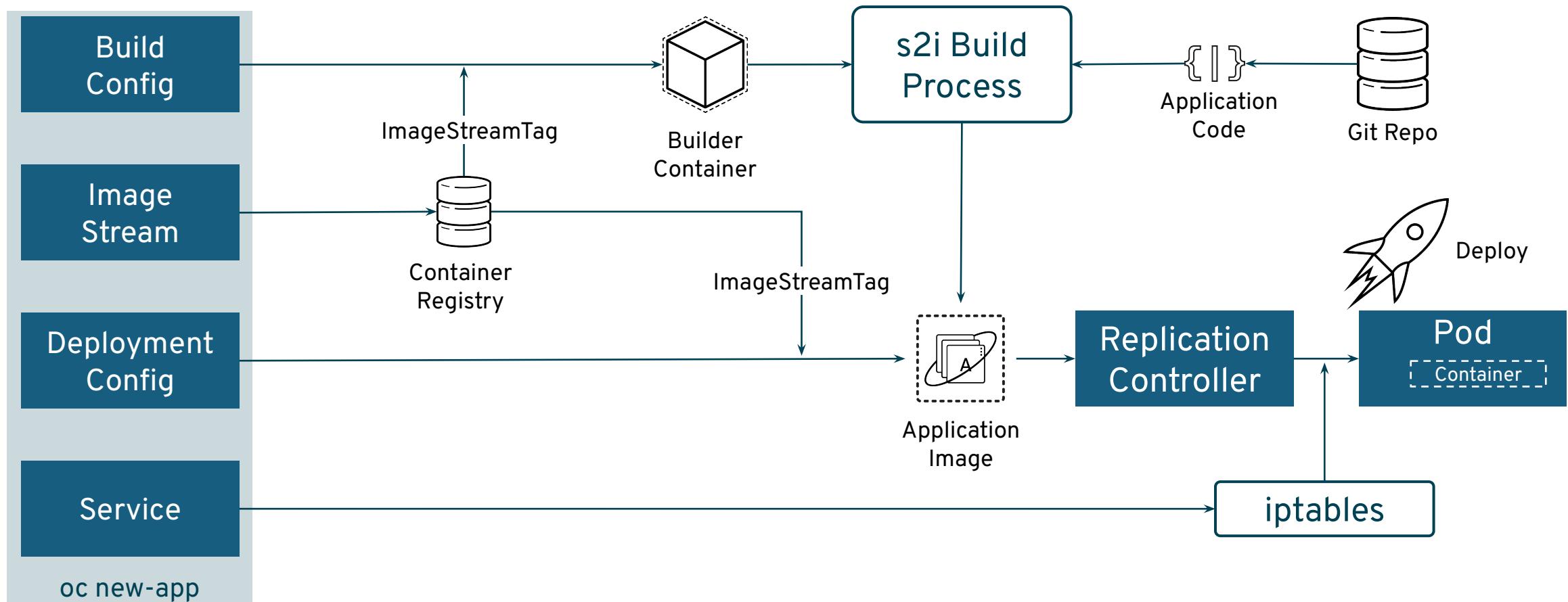
s2iはBuilderImageのラベルから、s2i Scriptの場所を読み込んで実行し、その結果をコンテナイメージとして出力するビルド方式です。

(*マニュアルでやると、docker runしたベースイメージで、s2iスクリプトをキックしたあとにdocker commit/tagする)



S2Iビルドプロセス

アプリケーションコードをInputにし、アプリケーションとベースイメージをビルドして、新しいコンテナイメージを生成



オートヒーリングに重要な要素 Probe

Liveness Probe は再起動ポリシーの対象となる

Liveness Probe (デフォルト値 = Success)

- ▶ コンテナが実行中かどうかを示します。
- ▶ **Liveness Probe が失敗すると、kubelet はコンテナを強制終了し、コンテナは再起動ポリシーの対象となります。**
- ▶ コンテナが Liveness Probe を提供しない場合、デフォルトの状態は Success です。
(Pod に問題が発生していても問題ないと判断される)

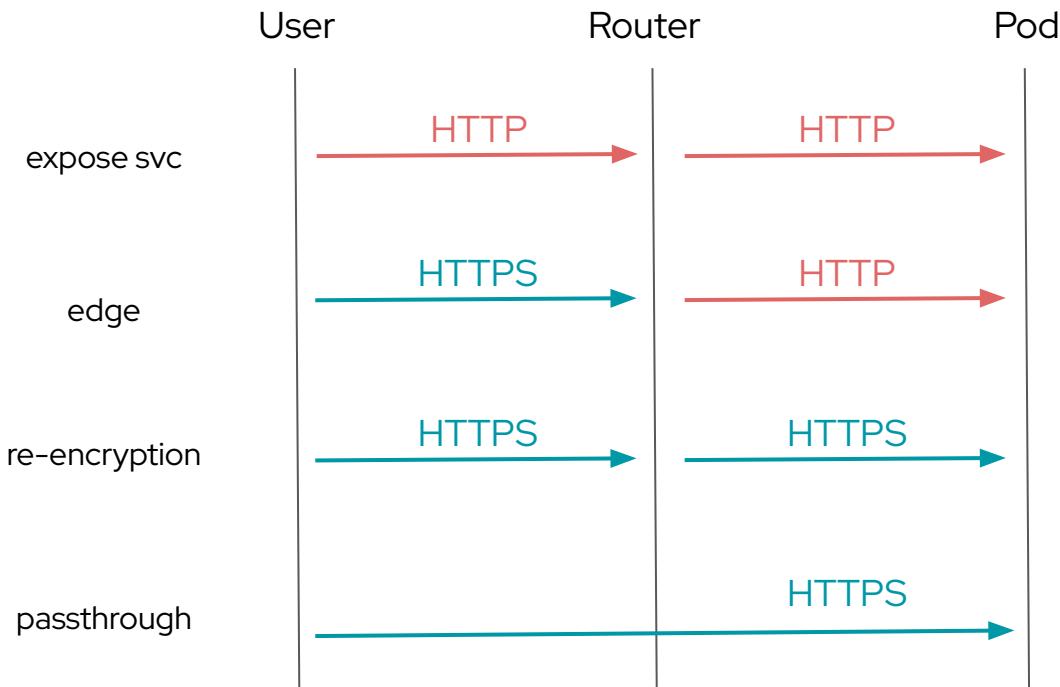
Readiness Probe (デフォルト値 = Failure or Success)

- ▶ コンテナがリクエストを処理する準備ができているかどうかを示します。
- ▶ **Readiness Probe が失敗すると、エンドポイントコントローラーは、ポッドに一致するすべての Service(後述)の EndPoint から Pod のIP アドレスを削除します。**
- ▶ デフォルト状態は Failure です。コンテナが Readiness Probe を提供しない場合、デフォルトの状態は Success です。

OpenShift Route と Kubernetes Ingress の違い

Feature	Ingress on OpenShift	Route on OpenShift
Standard Kubernetes object	X	
External access to services	X	X
Persistent (sticky) sessions	X	X
Load-balancing strategies	X	X
Rate-limit and throttling	X	X
IP whitelisting	X	X
TLS edge termination for improved security	X	X
TLS re-encryption for improved security		X
TLS passthrough for improved security		X
Multiple weighted backends (split traffic)		X
Generated pattern-based hostnames		X
Wildcard domains		X

Routeのタイプ



Edge Termination

- RouterでTLSターミネーションを行い、Podに転送します。
- 内部ネットワーク上のトラフィックは暗号化されていない。
- カスタマイズのTLS証明書を利用することができます。

指定されない場合は、Router のデフォルト証明書が使用されます。

OpenShift 4 Foundations

OpenShift 概要

アジェンダ

OpenShift 概要

1. Kubernetes と OpenShift
2. OpenShift アーキテクチャ

1. Kubernetes と OpenShift

2. OpenShift アーキテクチャ

Kubernetesができること

Kubernetes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。



デプロイの管理



コンテナの死活監視



リソースの制御



Bare metal



Virtual



Private cloud



Public cloud



Edge

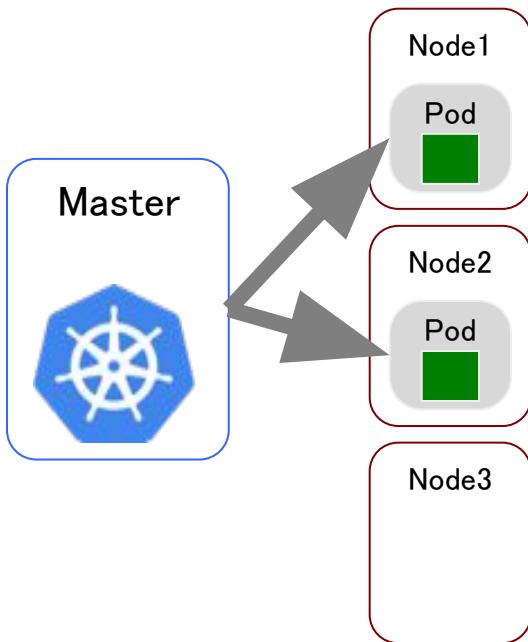
Note:

Kubernetesの復習

複数ノードのコンテナランタイムを統合し、スケーラブルなコンテナクラスタ実現のためのミドルウェア

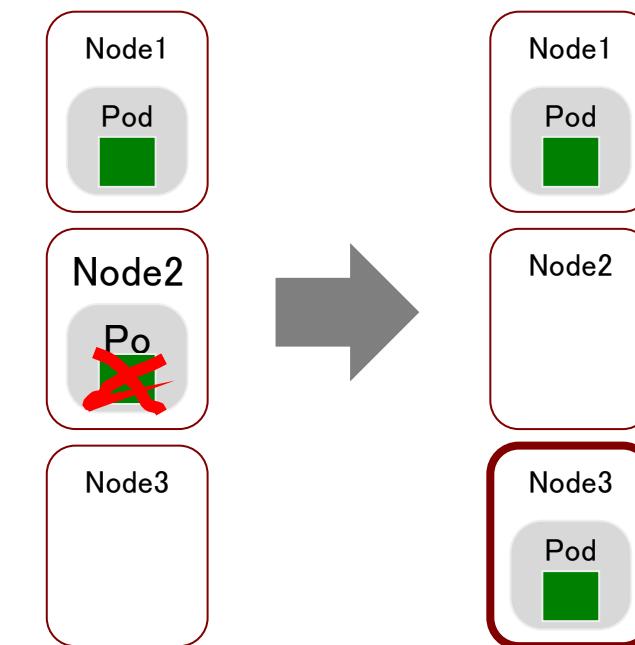
デプロイ管理

宣言的にコンテナのデプロイ



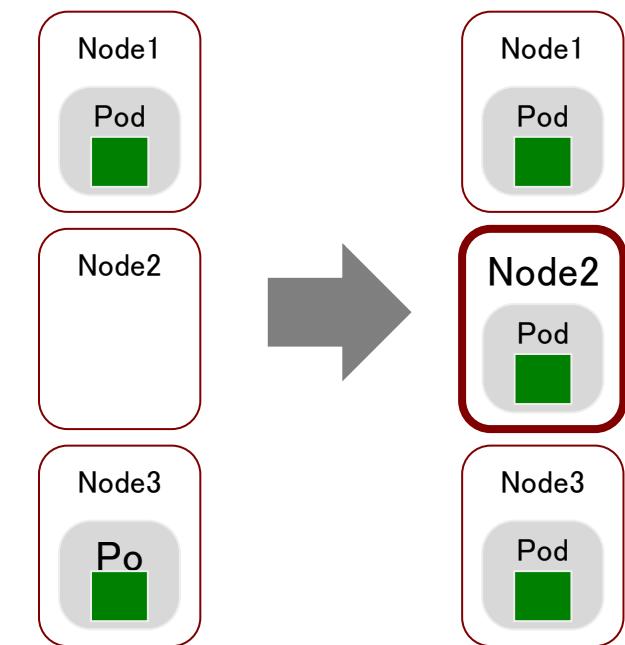
コンテナの死活監視

レプリカ数を一定に保つ



リソースの制御

レプリカ数を増やす・減らす



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、Kubernetesの機能だけではできることもあります。

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



Bare metal



Virtual



Private cloud



Public cloud



Edge

オープンソースや他のクラウドサービスで補完する

ライセンス費用の増加



ニンテンドーの動的
ビルド/デプロイ



ツールごとの保守調達



GKE



AKS
Kubernetes



EKS



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

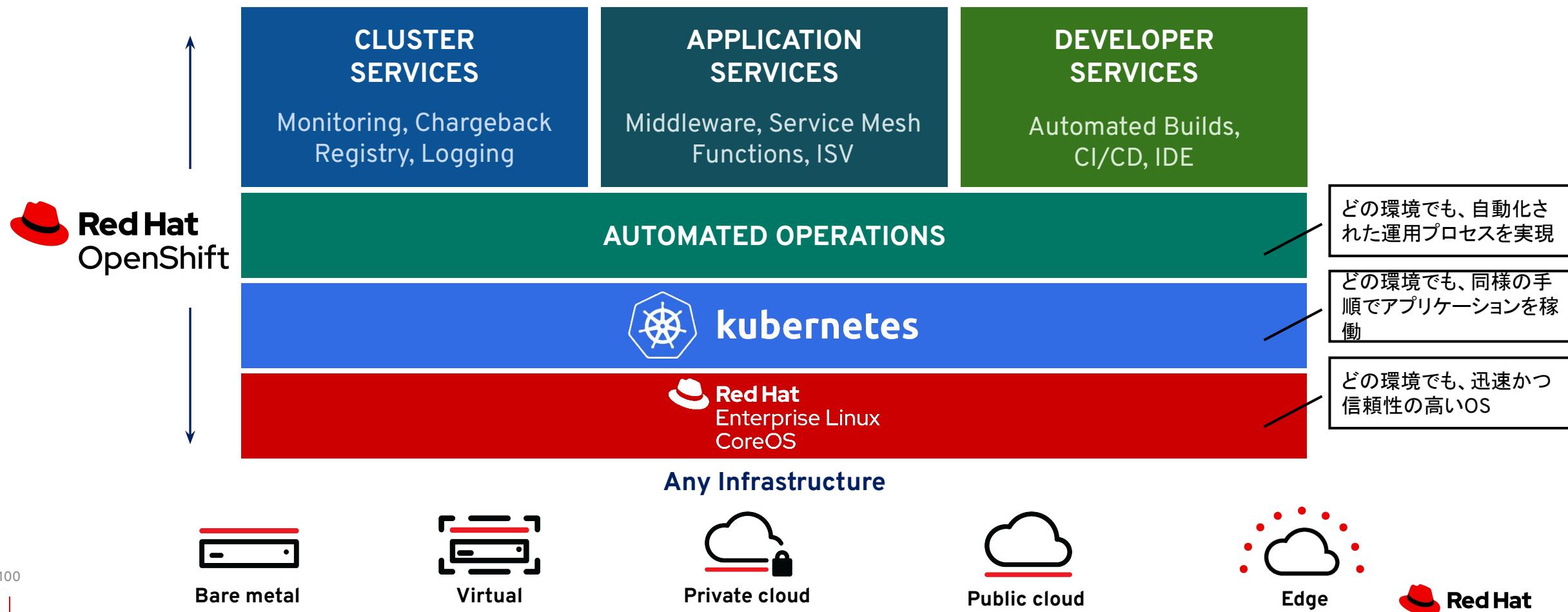


Edge

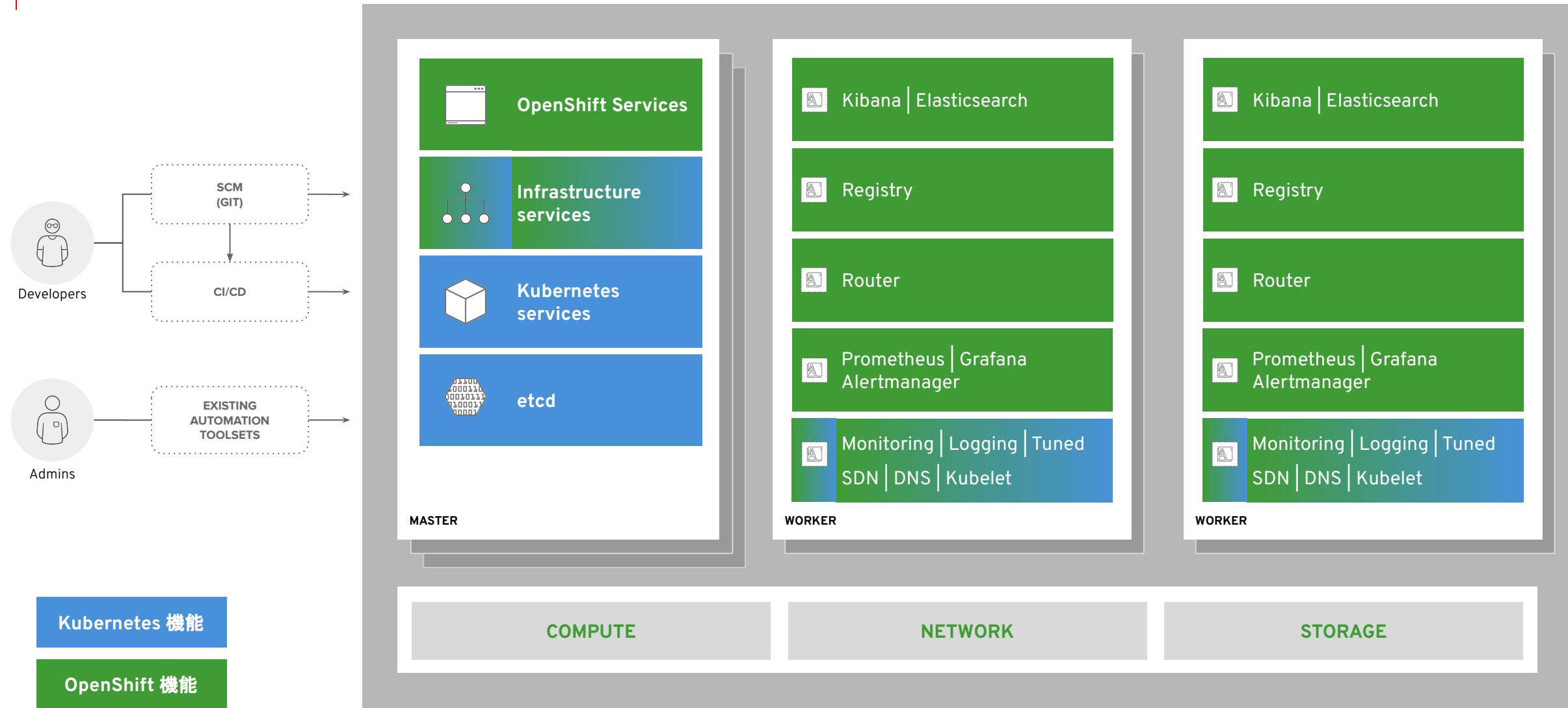
1. Kubernetes と OpenShift
2. OpenShift アーキテクチャ

OpenShiftはKubernetesを補完する付加価値機能を提供

運用プロセスを含めたポートアビリティの実現



OpenShift のアーキテクチャ



Note:

コンテナ

コンテナは最小のコンピュートユニットです。
コンテナはコンテナイメージによって作成されます

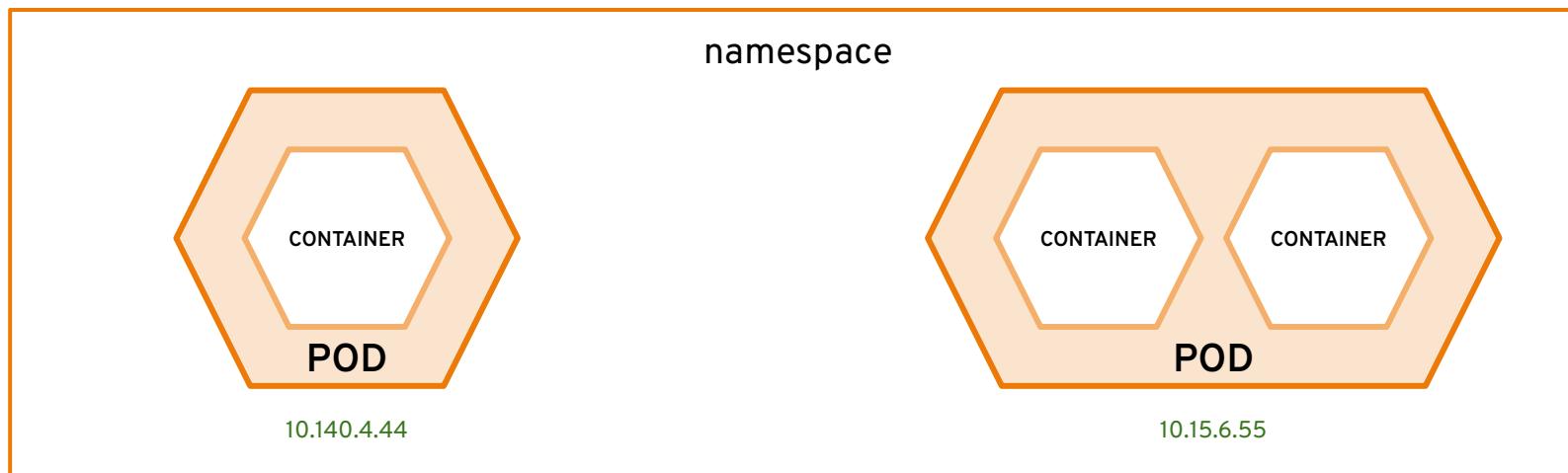


Note:

Pod

コンテナは、デプロイと管理の単位であるPodsに包含されています。

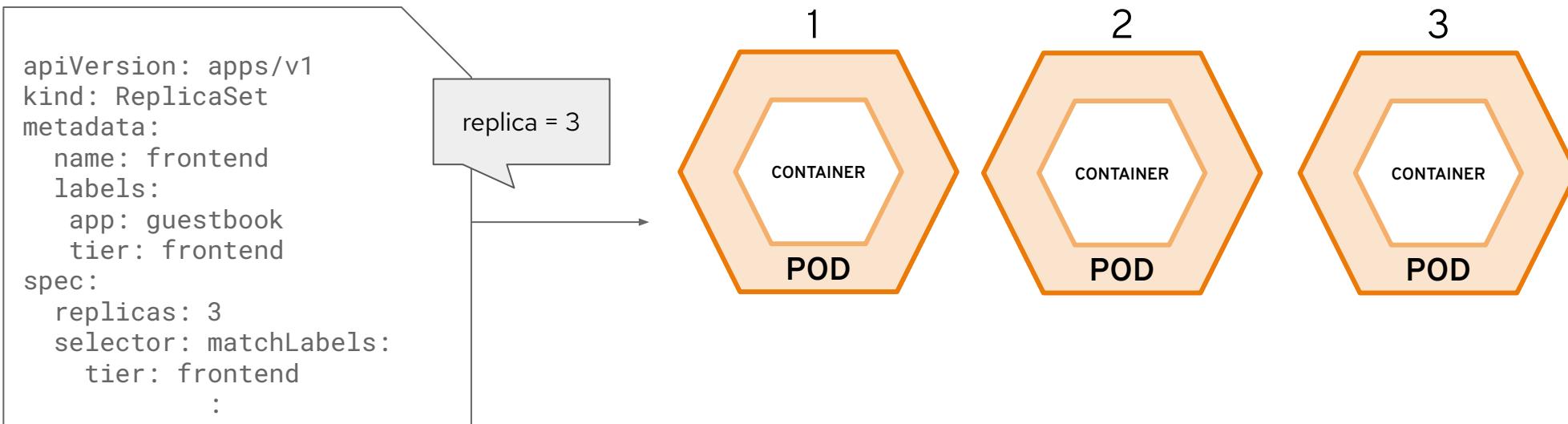
- ・コンテナ内で複数プロセスを実行するのは推奨されない。
- ・Pod内のコンテナはネームスペースを共有 (プロセス間はlocalhostで通信可)



Note:

ReplicaSet

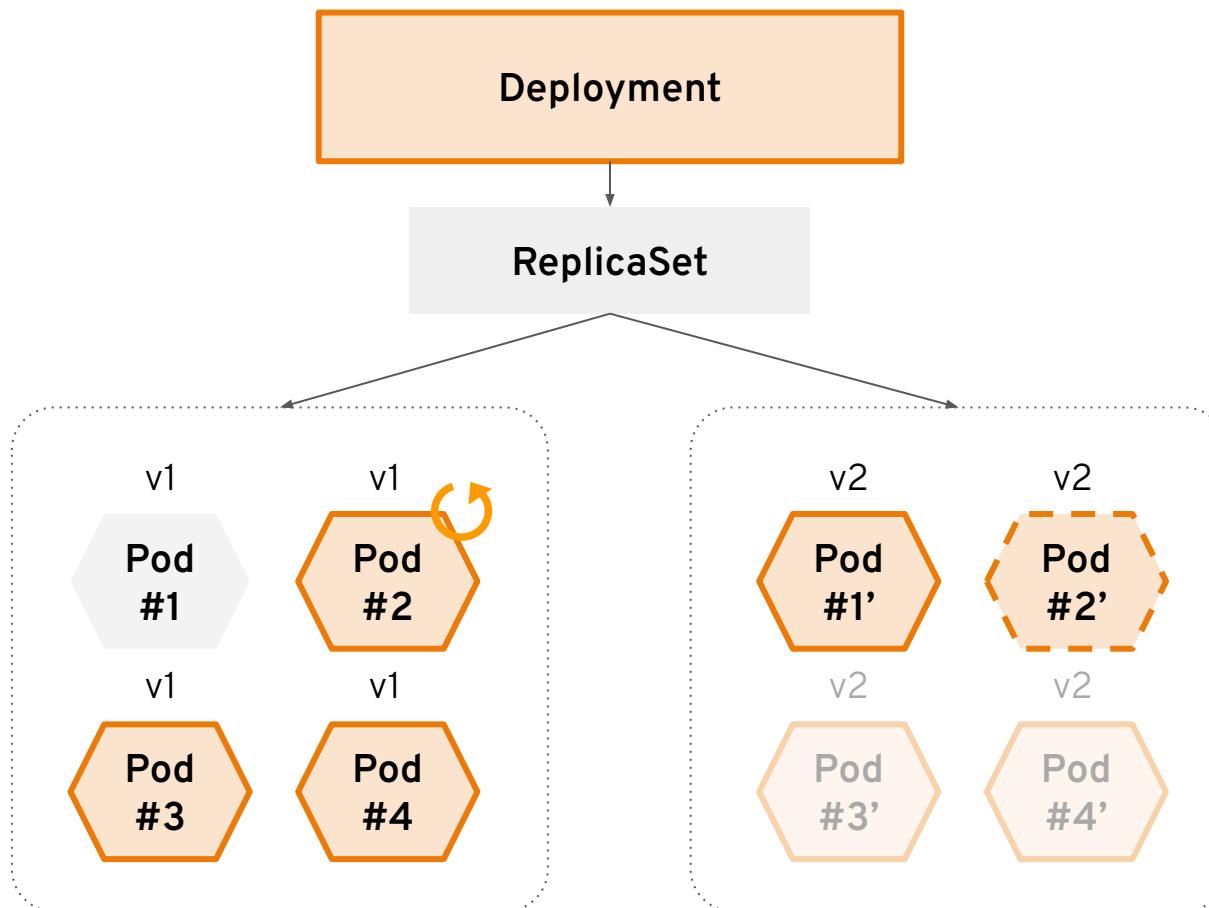
ReplicaSets 指定された数の Pod が常に稼働していることを保証します。



Note:

Deployment

Podの新バージョンをロールアウトする方法を定義します。

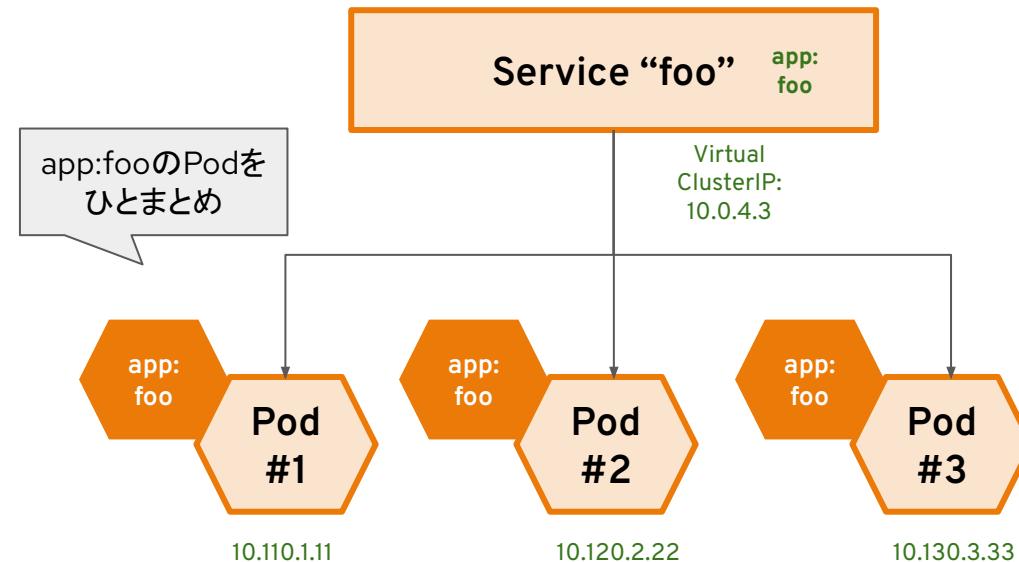


Note:

Services

servicesは、Pod間の以下の機能を提供します。

- ・**サービスディスカバリー**: サービス名で対象のPodへの通信が可能(無いとIPアドレスのみ)
- ・**内部ロードバランシング**: サービス名で対象の全てのPodにロードバランシング可能



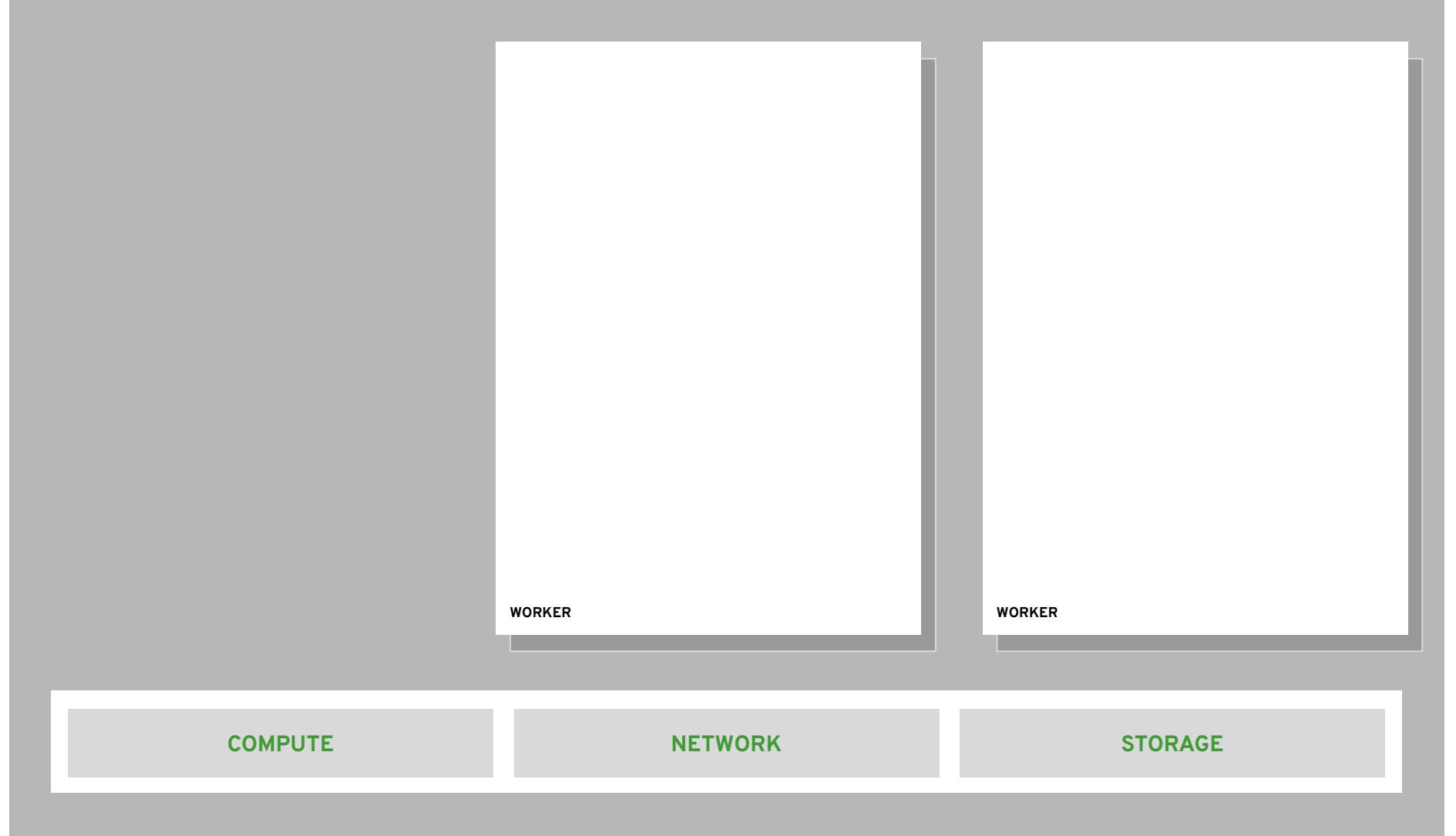
OpenShift のアーキテクチャ - クラスタ構成



OpenShift のアーキテクチャ - クラスタ構成



OpenShift のアーキテクチャ - クラスタ構成



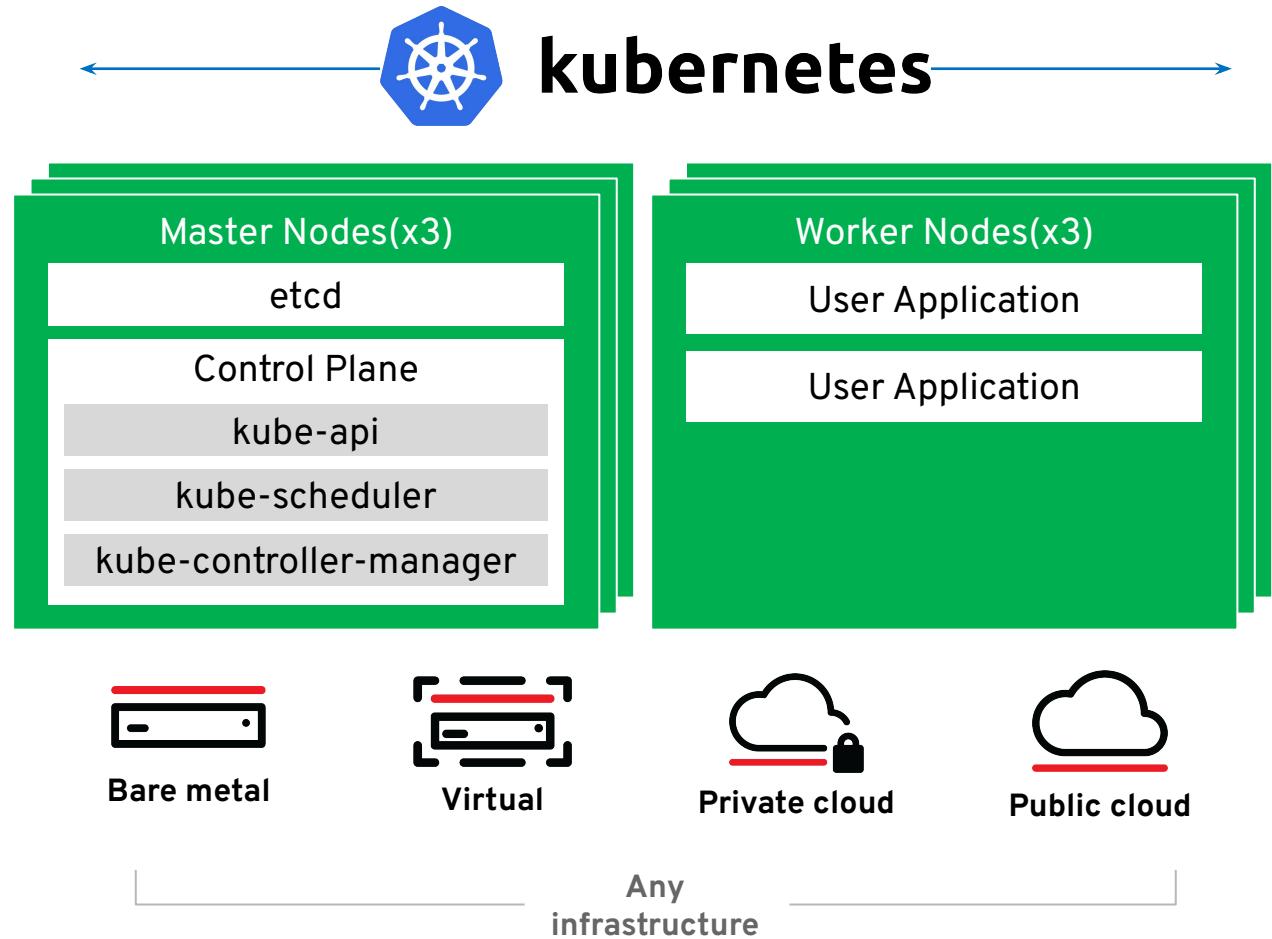
Note:

Kubernetesの基本コンポーネント

Kubernetesクラスタには2つのコンポーネントがあります。

- Master Node (コントロールプレーン)
- Worker Node (コンピュートマシン)

各ノードは独立したLinux環境であり、物理マシンでも仮想マシンでもかまいません。



Note:

OpenShiftのオペレーティングシステム

種類	Red Hat Enterprise Linux	Red Hat Enterprise Linux CoreOS
	汎用OS	OCP特化のOS
特徴	<ul style="list-style-type: none"> ・10年以上のライフサイクル ・業界標準のセキュリティ ・パフォーマンス ・幅広いパートナーエコシステムによるカスタマイズ 	<ul style="list-style-type: none"> ・自律的な管理やOCPに統合されたアップデート機構 ・イミュータブルな制御 ・ホストとコンテナの高い分離性 ・パフォーマンス最適化
採用基準	追加のソリューションとのインテグレーションが必要な場合	Cloud Nativeや、自律運転が重視される場合
管理性	<ul style="list-style-type: none"> ・サーバOSとしての管理が必要となる ・パッチ適用やアップデートなど 	基本的に個々のOSへの詳細な管理を意識する必要がなく、OSを宣言的に管理できる(RHCOSのみ)。カーネルのアップデートは、OCPのバージョンアップの一部として提供

コントロールプレーン
のOS

Master Node/コントロールプレーン
には使えない

コンピュートのOS

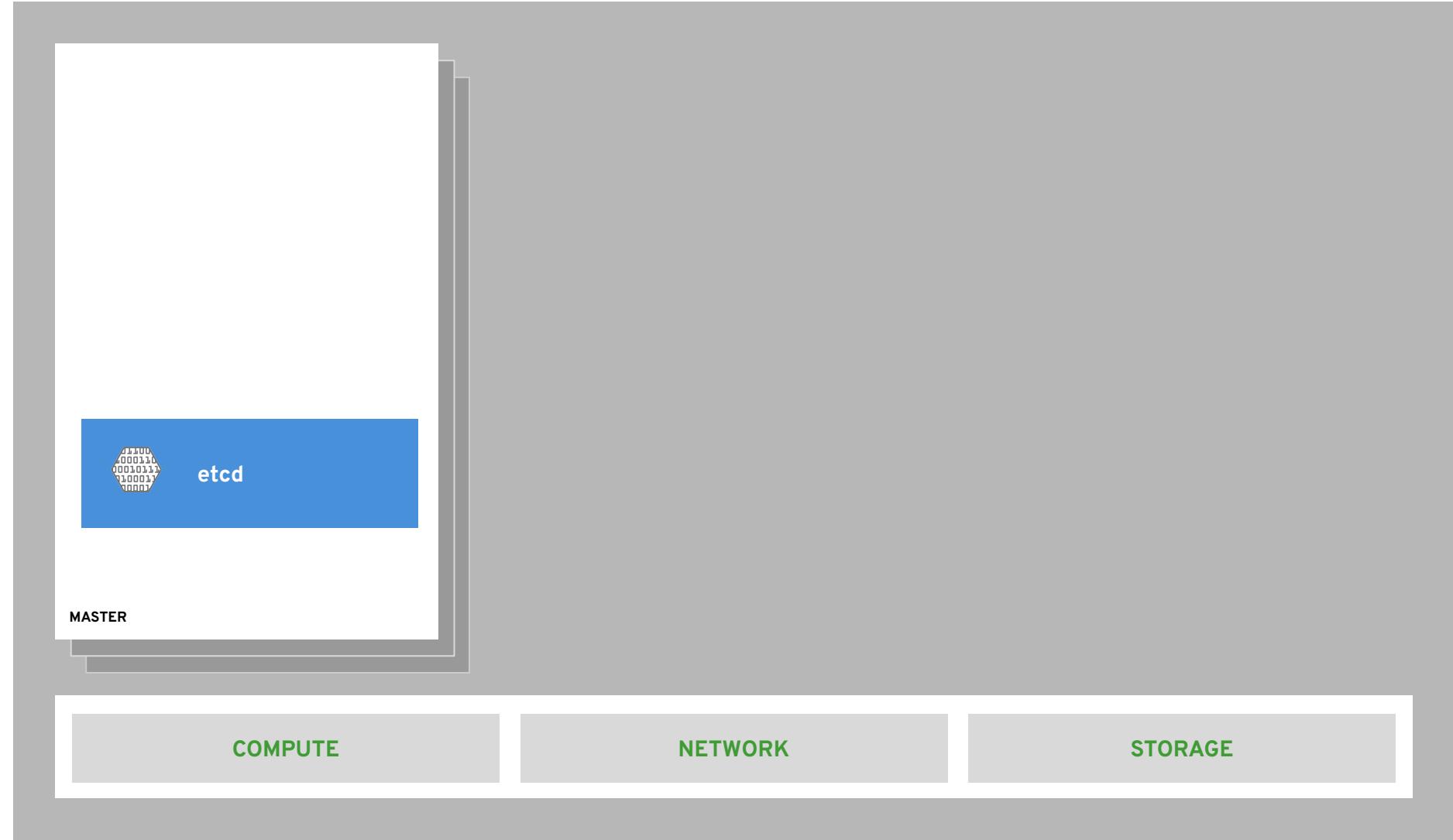
コントロールプレーン
のOS

コンピュートのOS

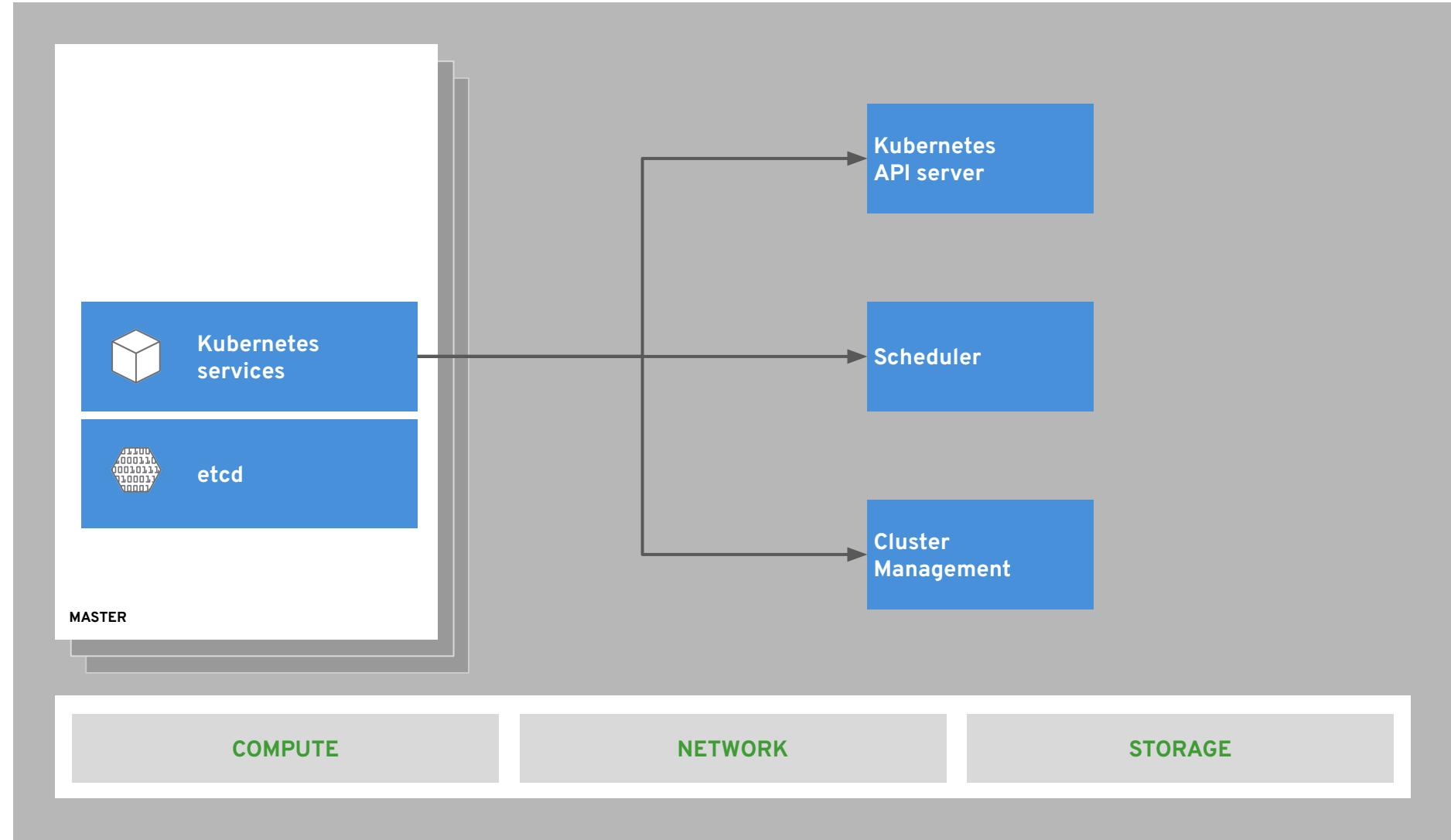
V0000000



OpenShift のアーキテクチャ - etcd によるステート管理



OpenShift のアーキテクチャ - Kubernetes のコアコンポーネント



Note:

Kubernetesの基本動作

etcd: Kubernetesのリソース情報を確保する分散キーバリューストア
etcdに書かれたステートを正とし、その状態をコンテナ基盤上で維持し続ける

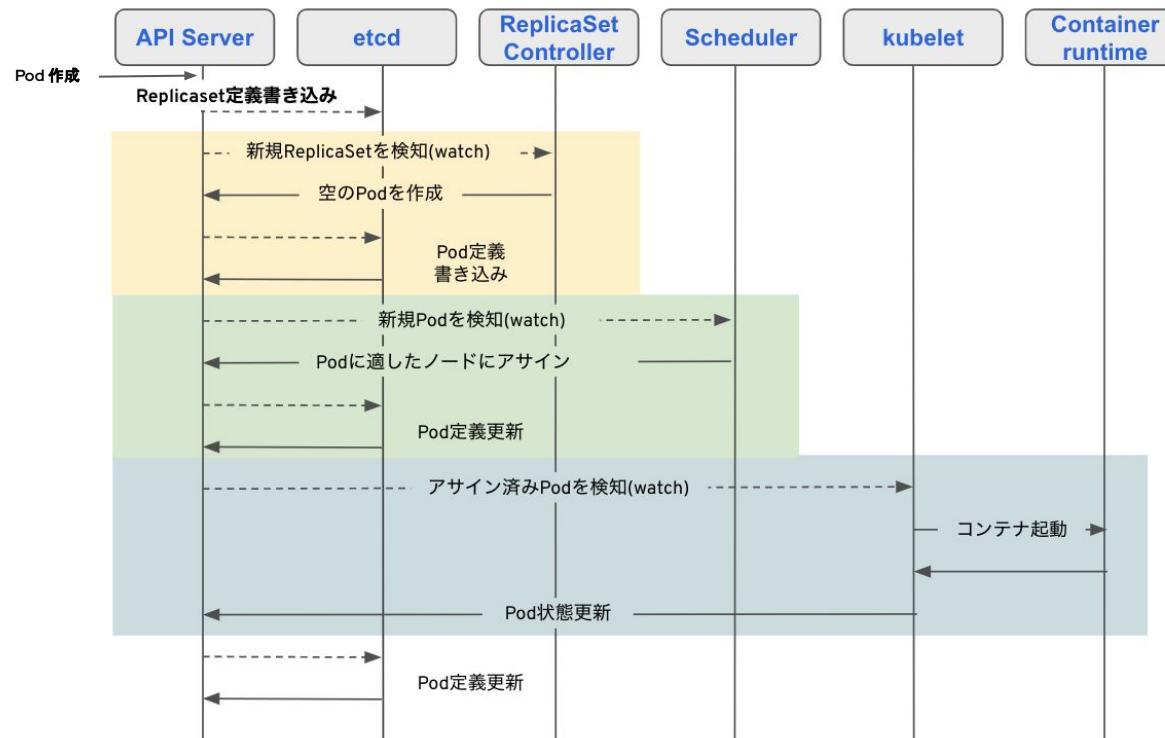


図 Pod/ReplicaSet作成のシーケンス

API Server:

KubernetesへのAPIリクエストをREST経由で受付するコンポーネント

Scheduler:

Podが新規に作成された時やノード障害でPodが再作成されたときに適切なコンピュートノードに割り当てるコンポーネント

Kubelet:

etcdに保存されている構成から、担当ノードで実行されるべきコンテナを見つけ、Podの起動を行うコンポーネント。Podが望ましい状態で稼働していることを確認します。

堅牢化されたコンテナ実行環境

OpenShift Core supports Developer Productivity & Cluster Management for Enterprise

Kubernetes

デファクトスタンダードなコンテナオーケストレーション

Declarative Configuration

Self-Healing

Auto Scaling

Cluster Management

高度なセキュリティと監査、可用性、管理の容易性

Installation Configuration

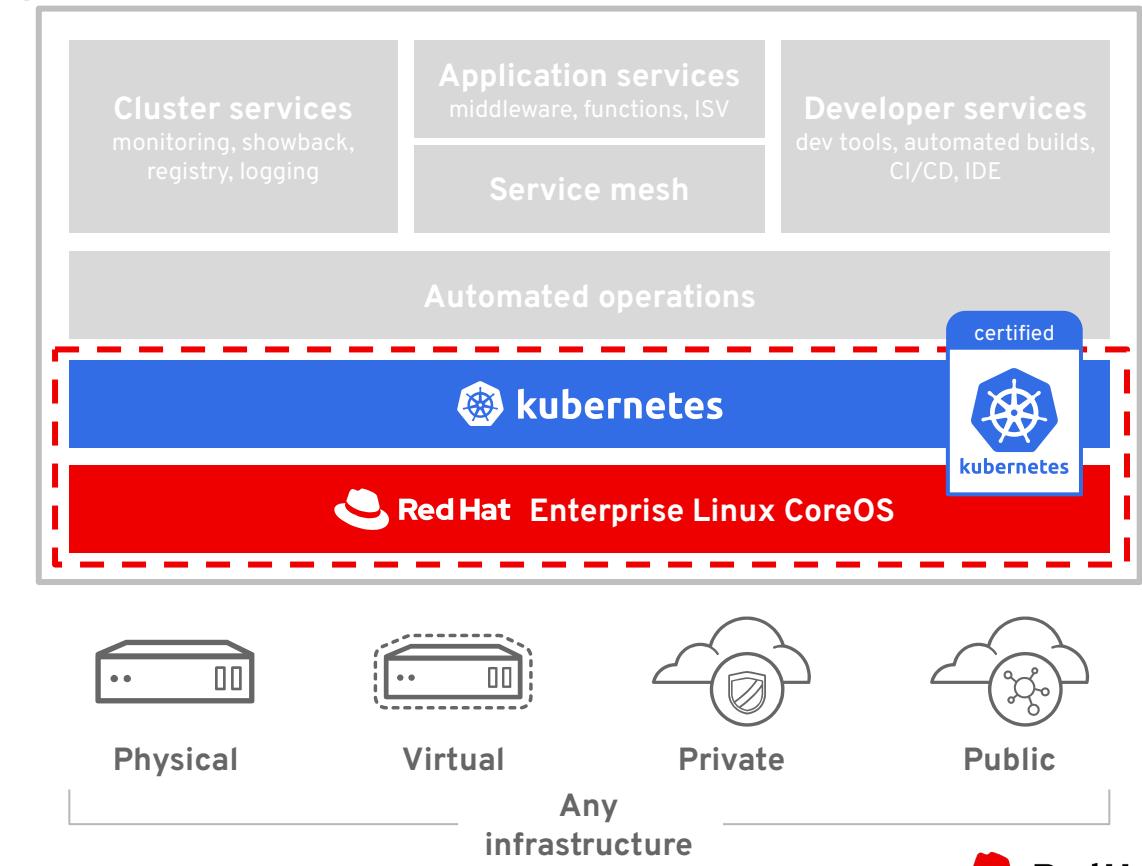
Cluster Scalability

Security

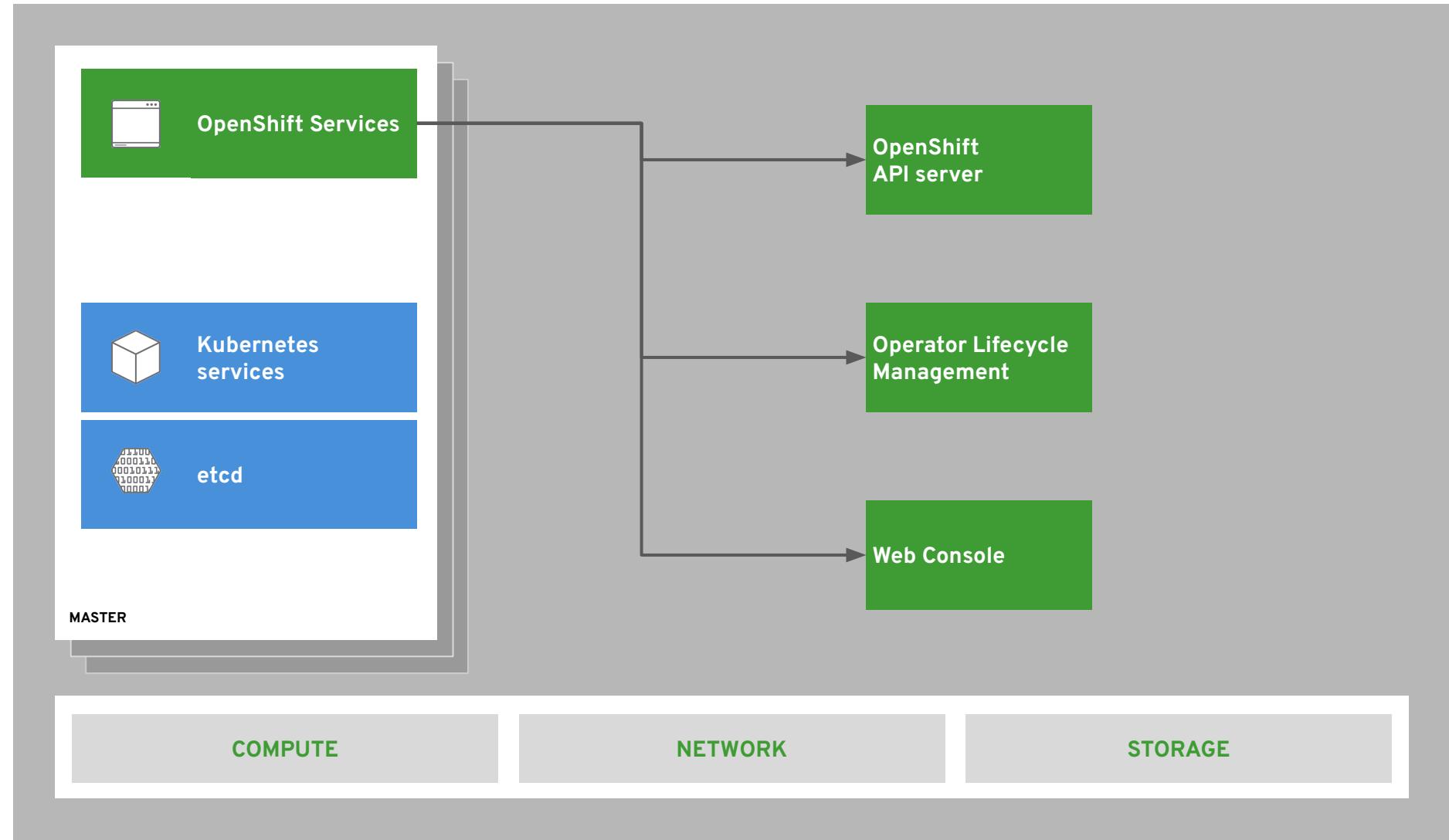
Network

GUI

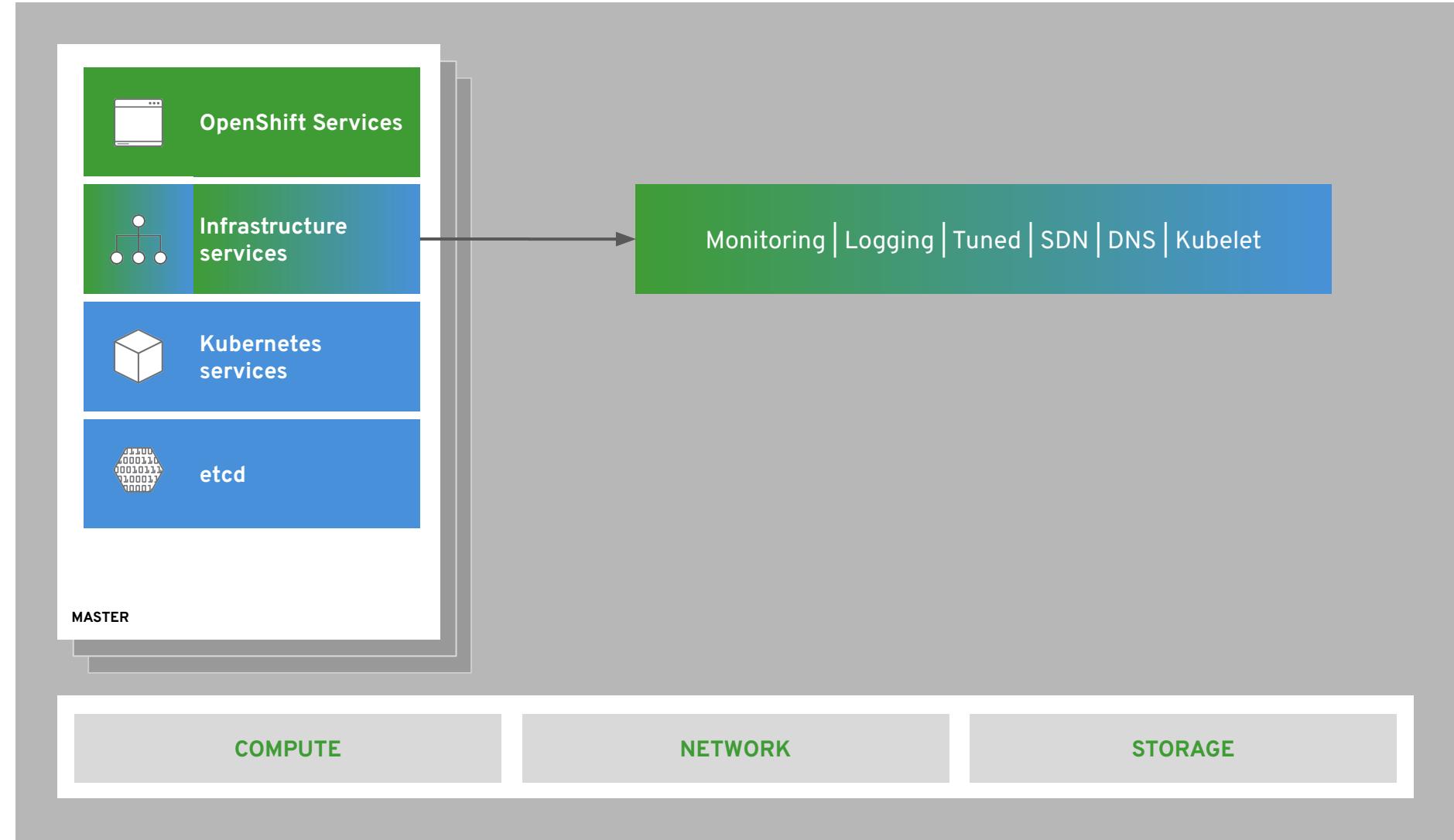
Cluster Upgrade



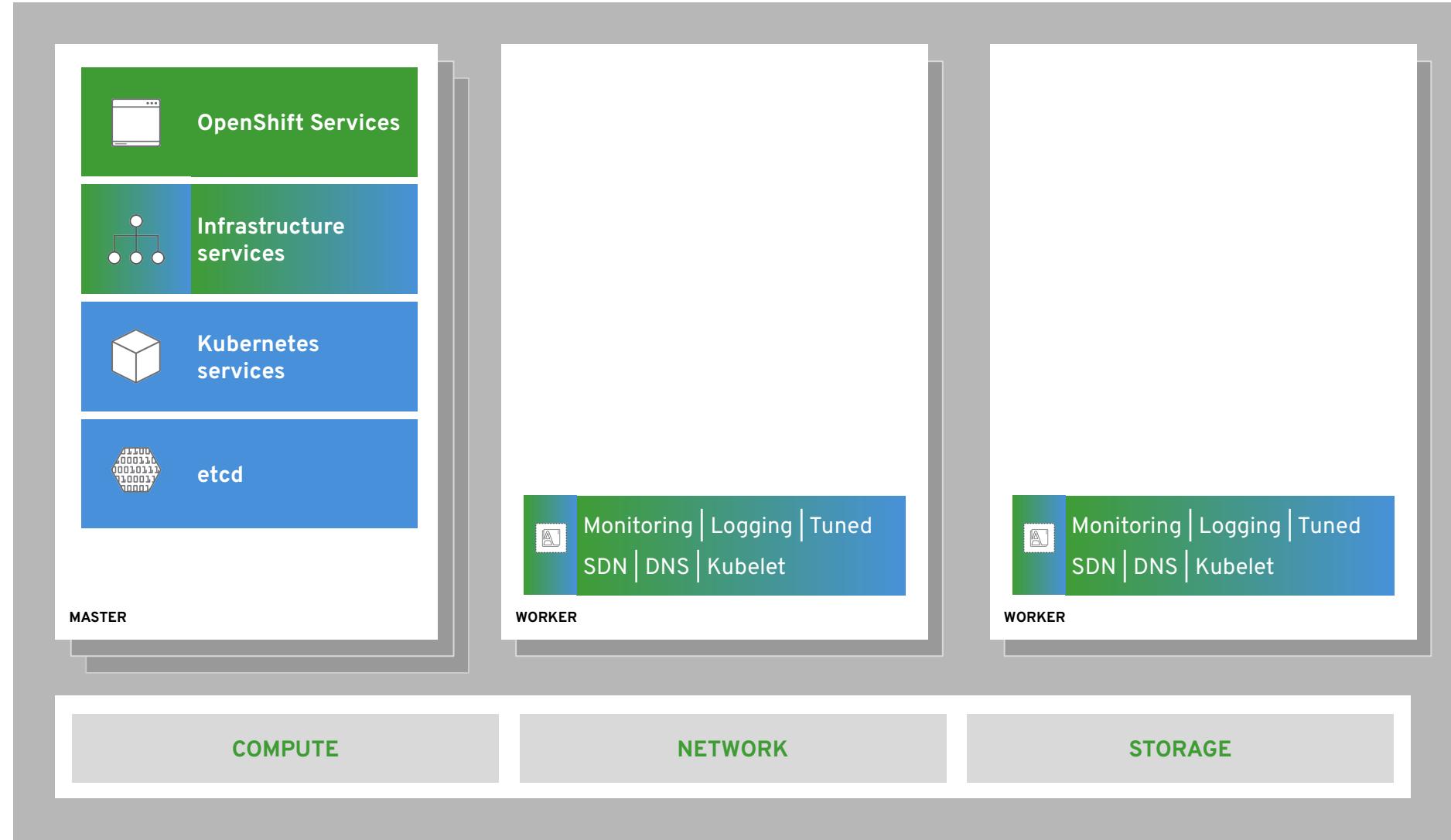
OpenShift のアーキテクチャ - OpenShift のコアコンポーネント



OpenShift のアーキテクチャ - インフラを支えるコンポーネント



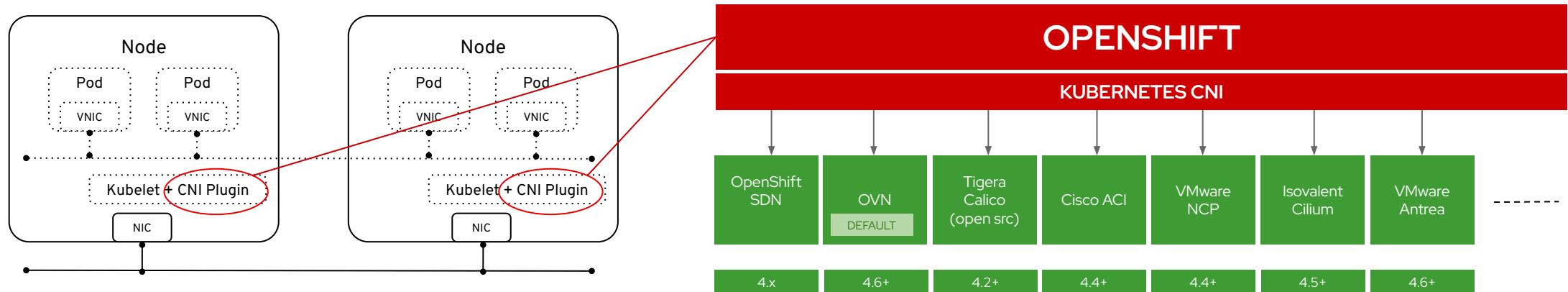
OpenShift のアーキテクチャ - インフラを支えるコンポーネント



Note:

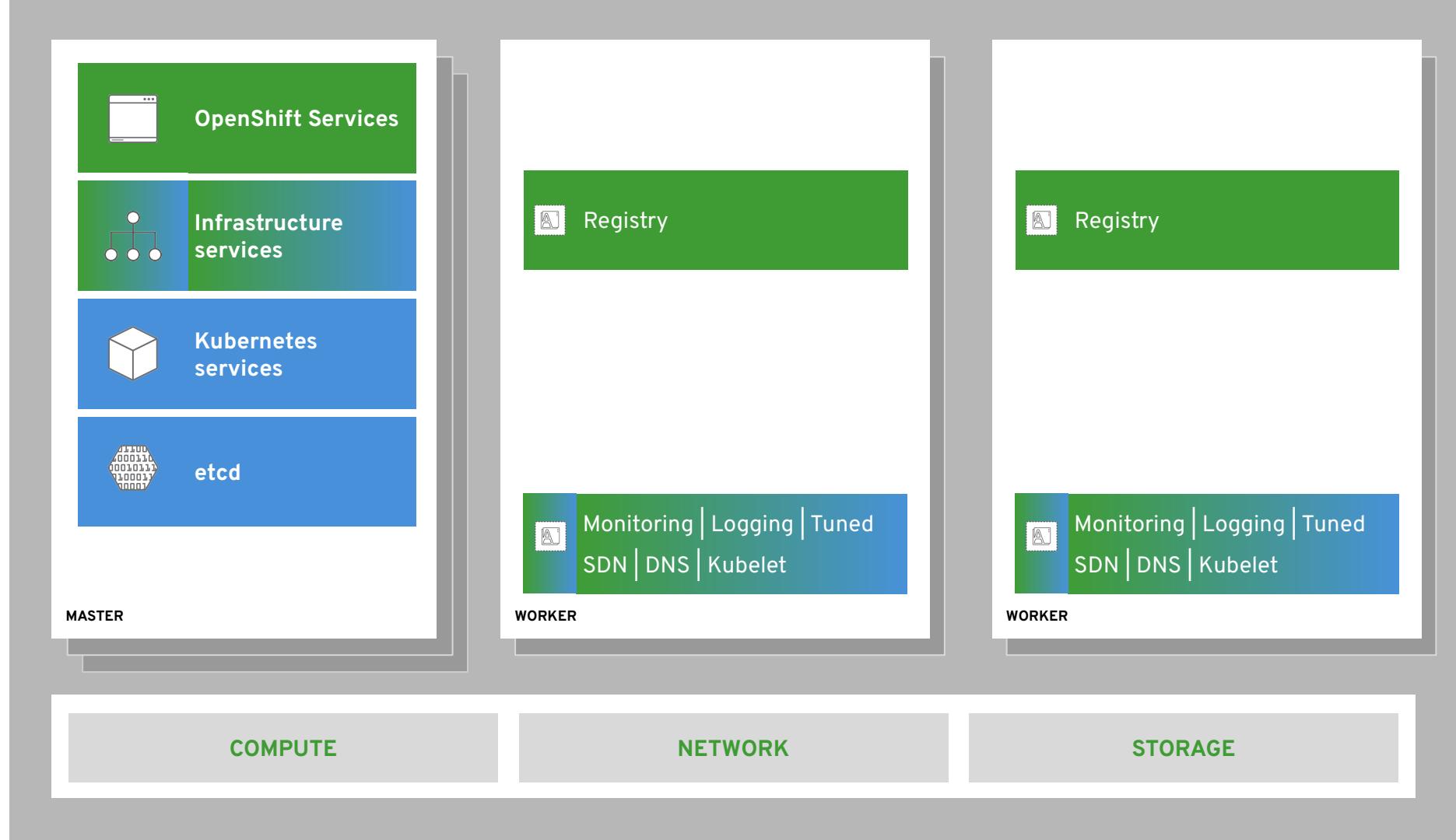
OpenShift Networking Plug-ins

- CNI (Container Network Interface) というインターフェースを定義し、そこに3rd Partyが作成した仮想ネットワーク用のプラグインを使用する。OpenShift では、「OpenShift SDN」と「OVN」と呼ばれるプラグインを提供
- 4.10 からデフォルトはOVN



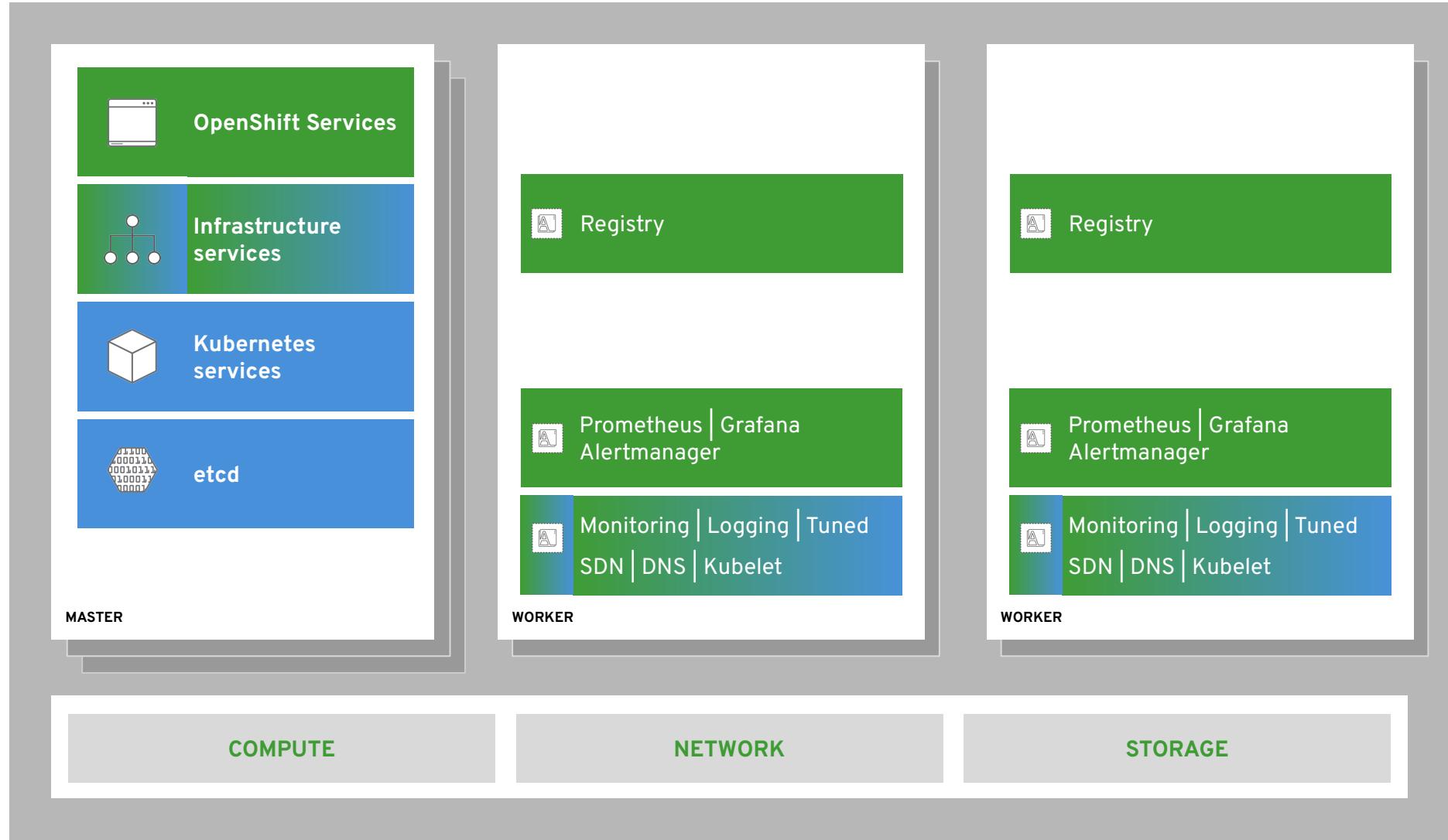
OpenShift のアーキテクチャ - OpenShift 内部のコンテナイメージレジストリ

次のセッションにて
紹介します！



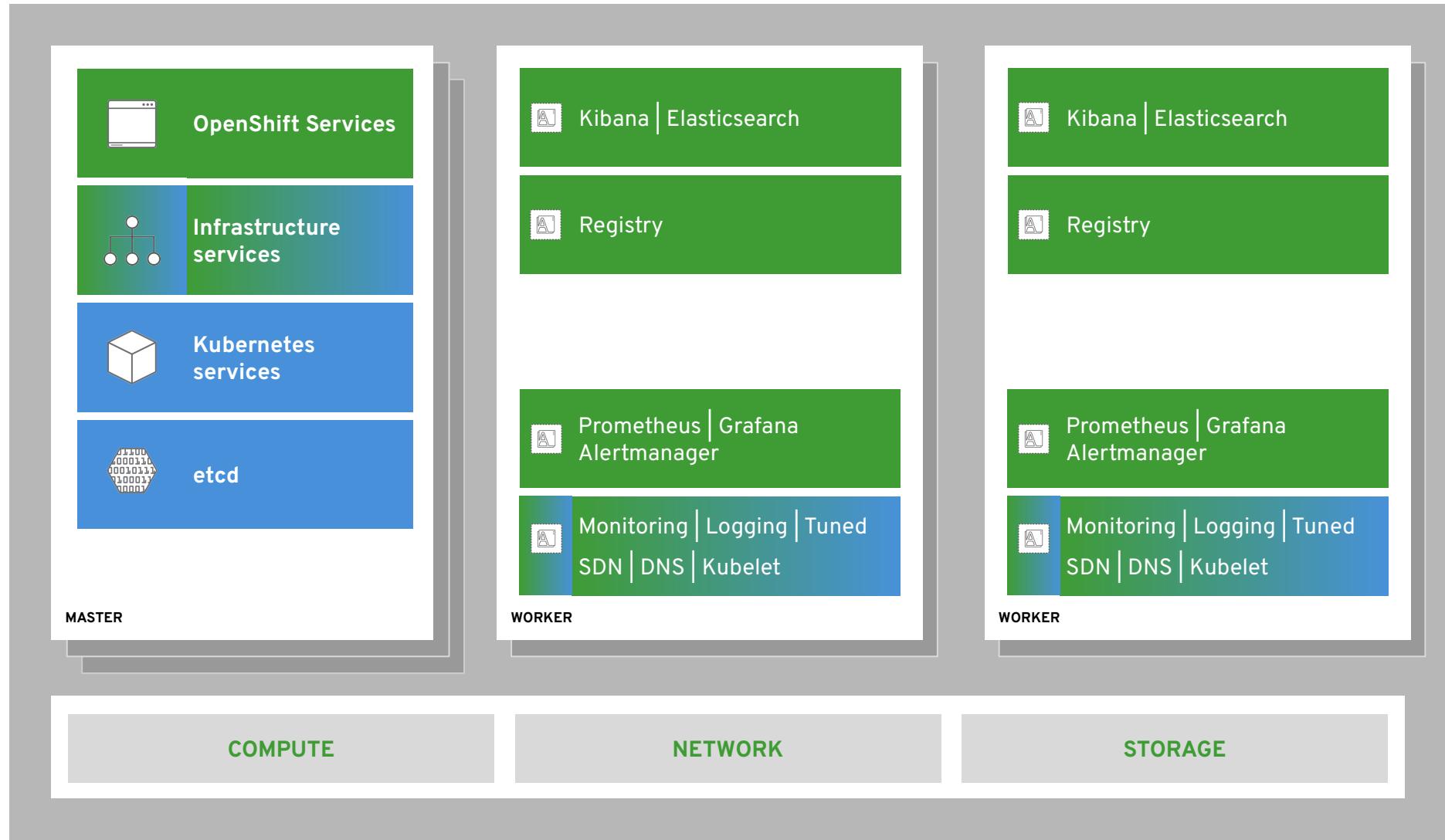
OpenShift のアーキテクチャ - Cluster モニタリング

次のセッションにて
紹介します！

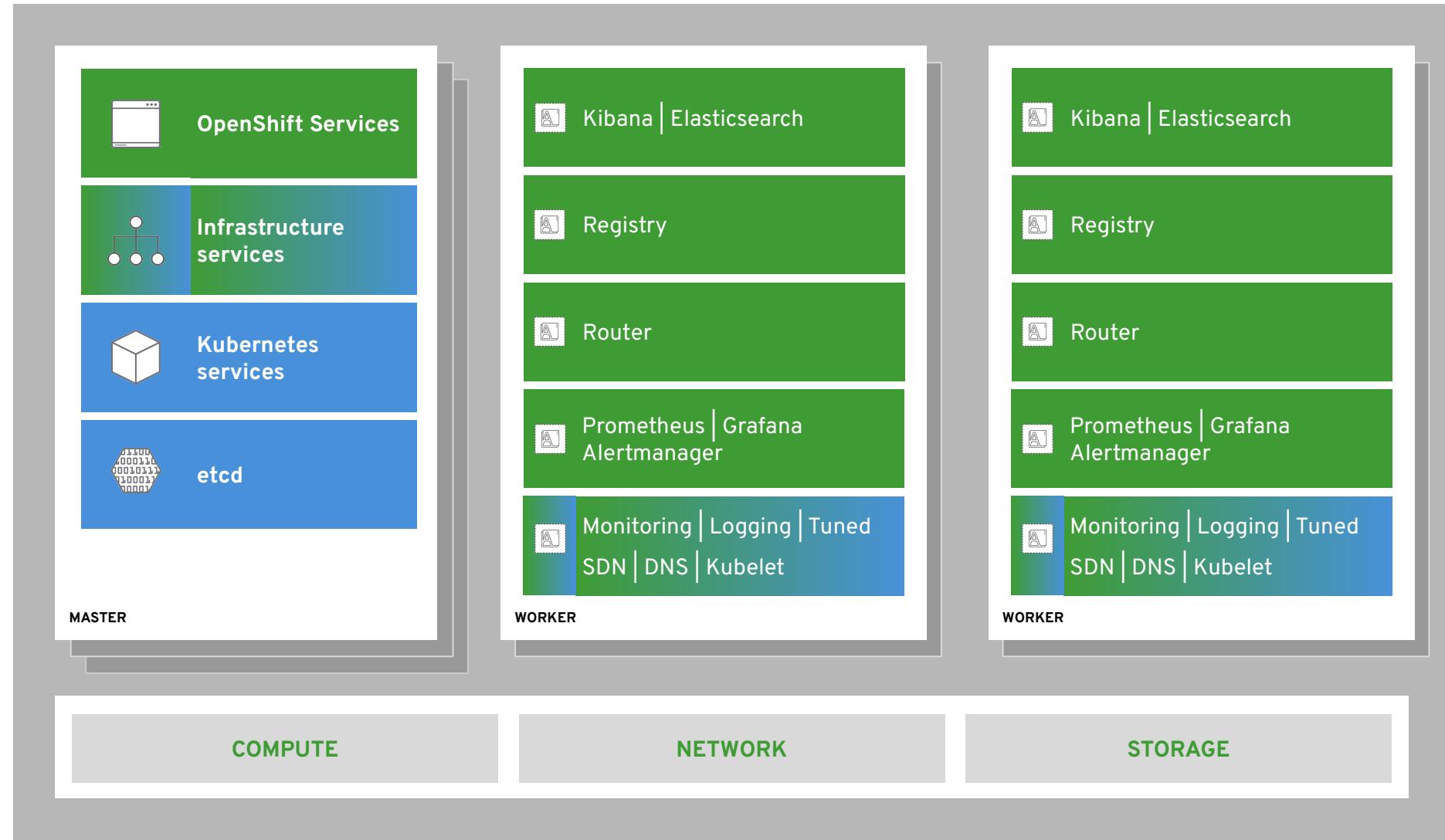


OpenShift のアーキテクチャ - ロギング機能

次のセッションにて
紹介します！



OpenShift のアーキテクチャ - 外部連携



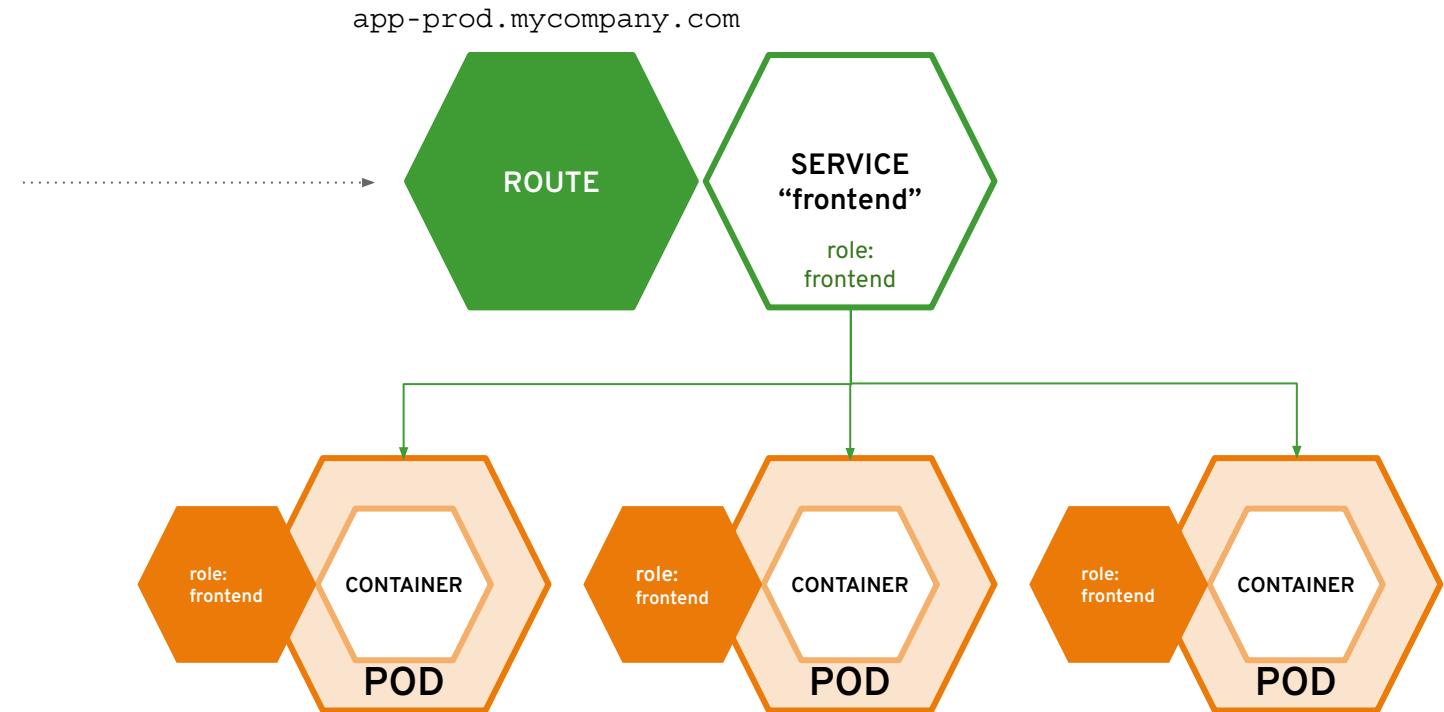
Note:

Router

Router (コントローラー)

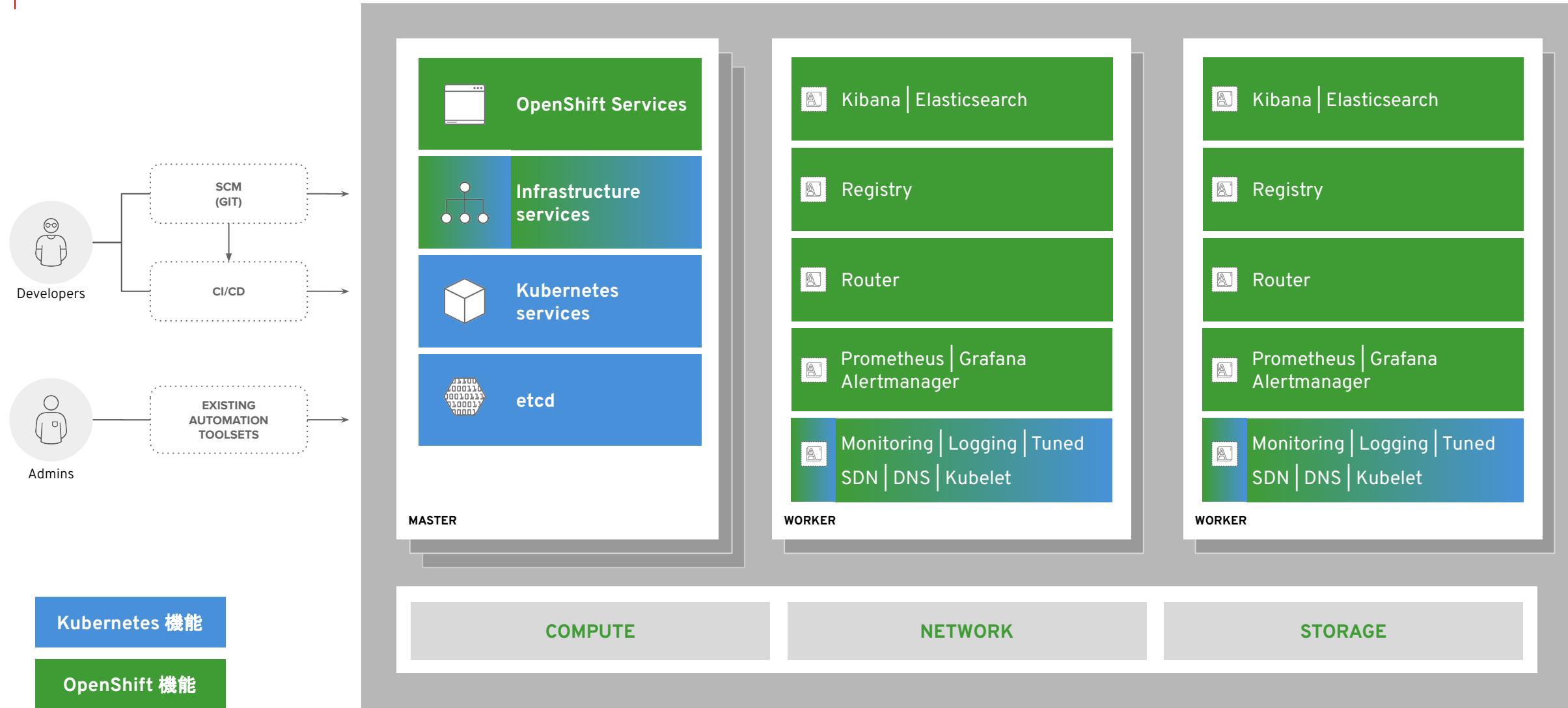
環境外のクライアントから、外部 URL としてサービスにアクセスが可能

```
> curl http://app-prod.mycompany.com
```



RouterはA/B DeploymentやTLS終端など高度な機能を利用することができます。
Kubernetesの標準オブジェクトである Ingress に相当。

OpenShift のアーキテクチャ



Thank you

 [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