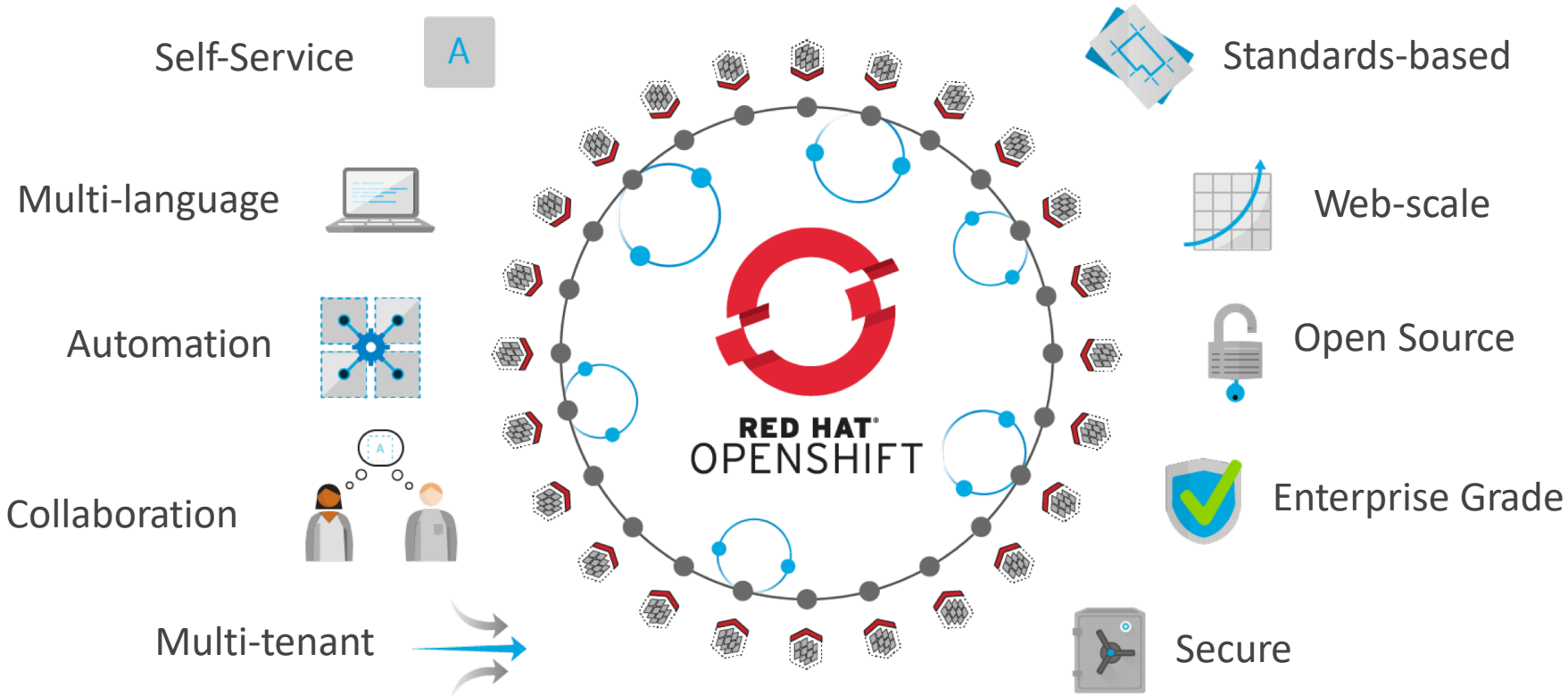
OpenShift Overview

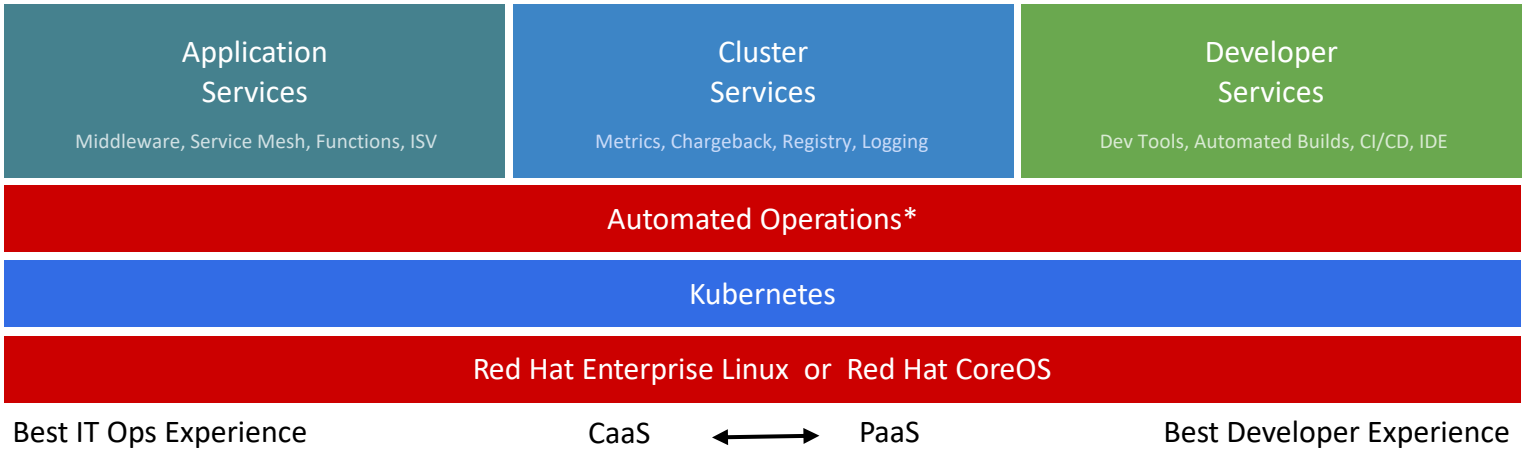
Developer Advocate
Raghavendra Deshpande
IBM

Agenda

- OpenShift Concepts
- OpenShift Architecture
- OpenShift Builds
- OpenShift Deployment
-


Self-Service
Multi-language
Automation
Collaboration
Multi-tenant
RED HAT
OPENSHIFT
Standards-based
Web-scale
Open Source
Enterprise Grade
Secure

OPENSHIFT CONTAINER PLATFORM



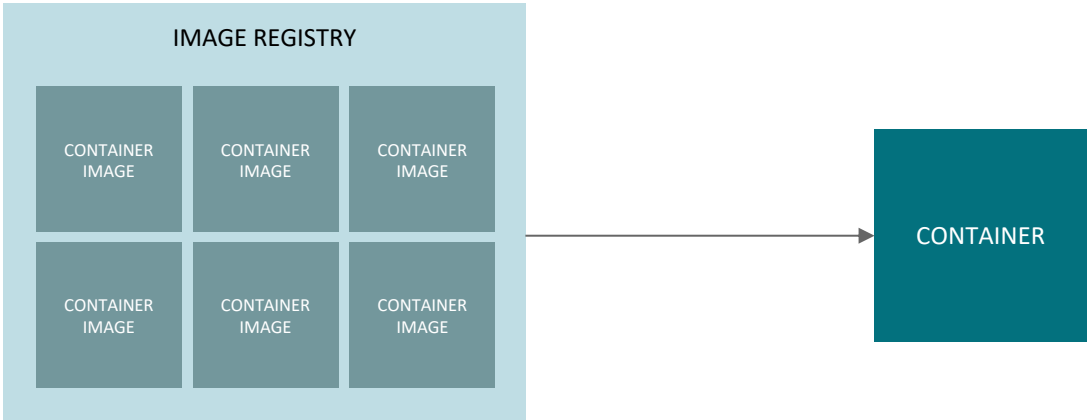
*coming soon


OPENSHIFT CONCEPTS
OVERVIEW

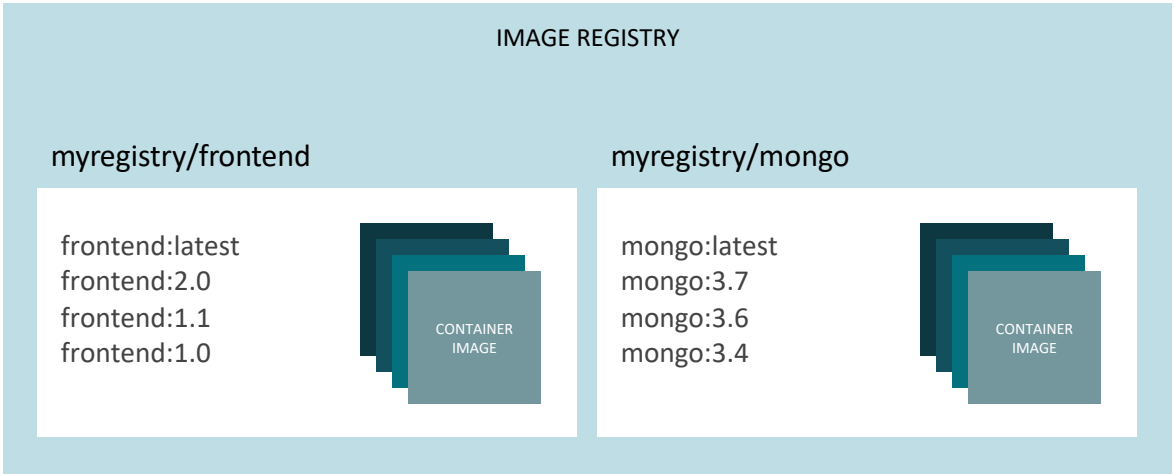
A container is the smallest compute unit

containers are created from container images

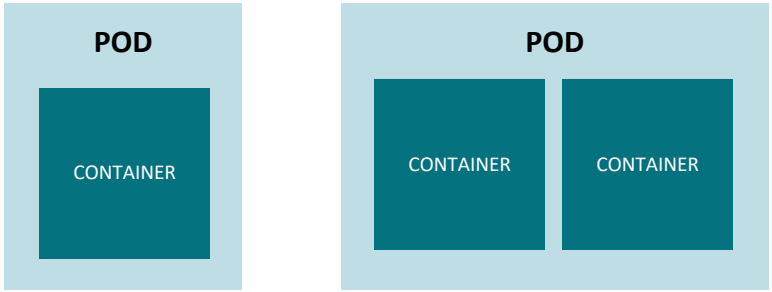
container images are stored in an image registry



an image repository contains all versions of an image in the image registry



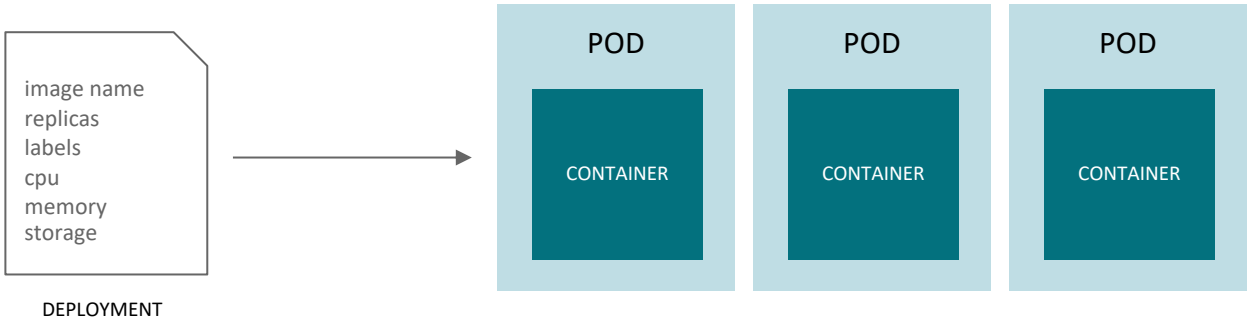
containers are wrapped in pods which are units of deployment and management



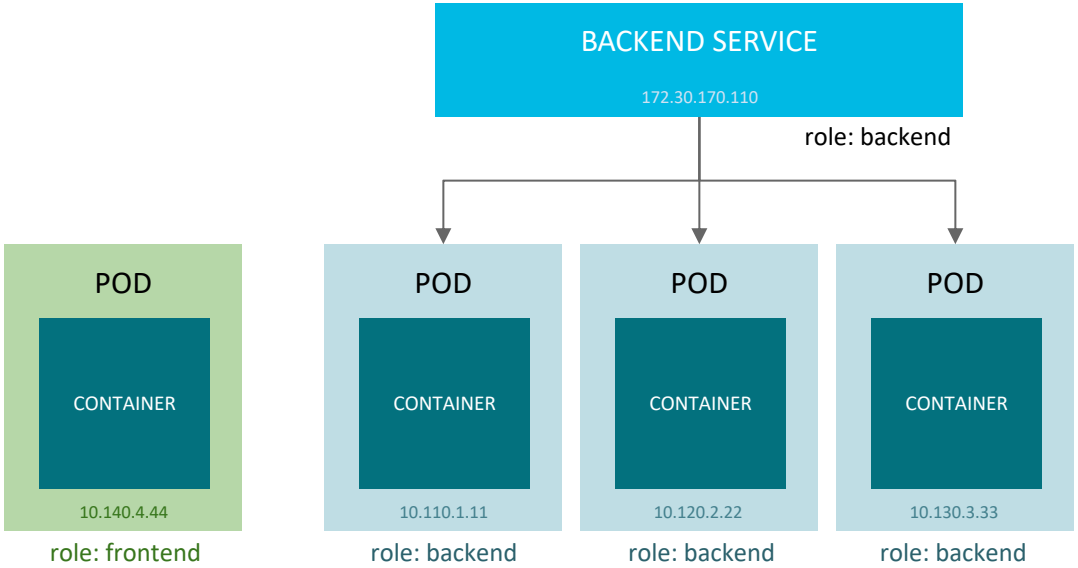
IP: 10.1.0.11

IP: 10.1.0.55

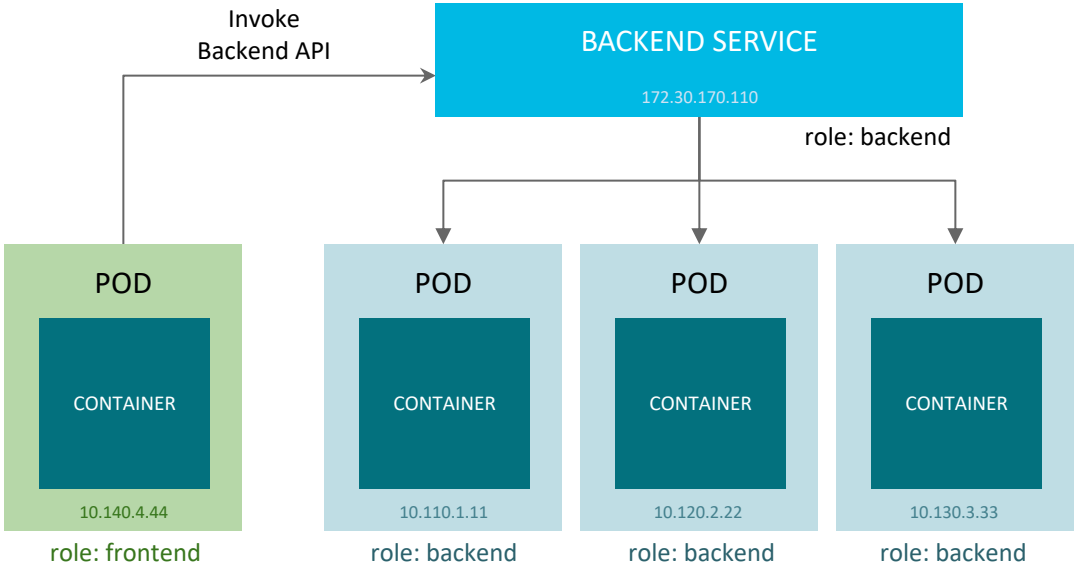
pods configuration is defined in a deployment



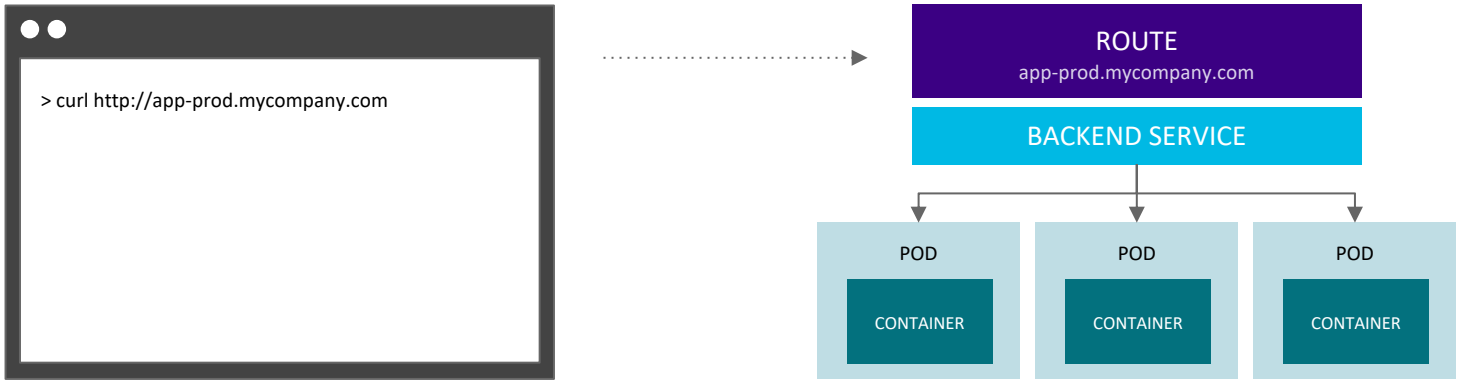
services provide internal load-balancing and service discovery across pods



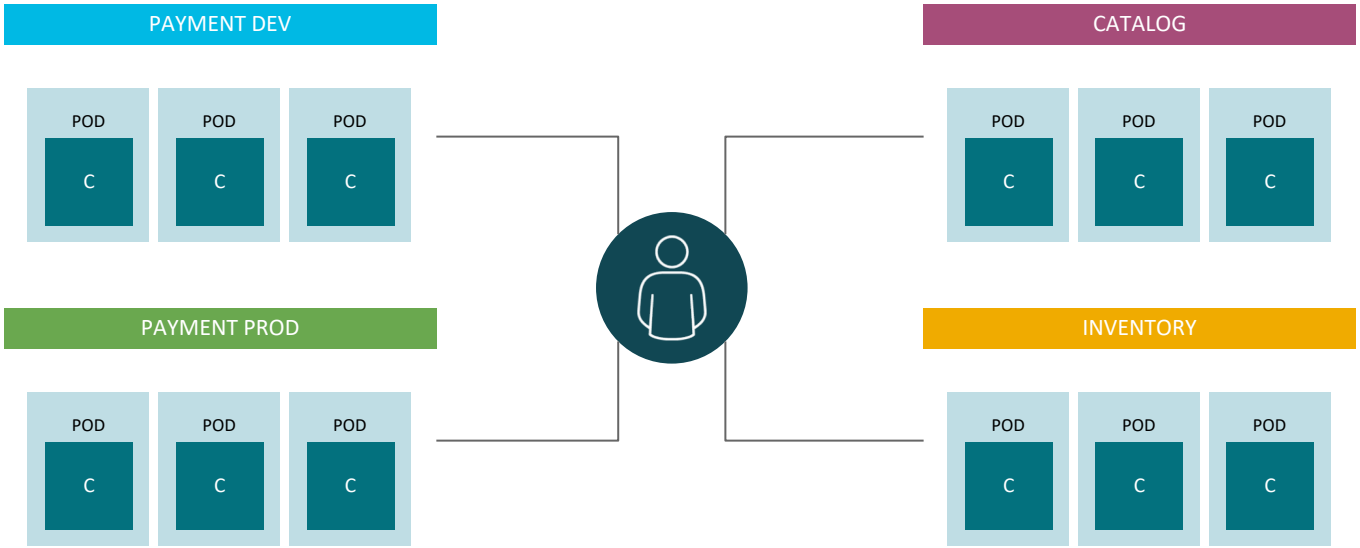
apps can talk to each other via services

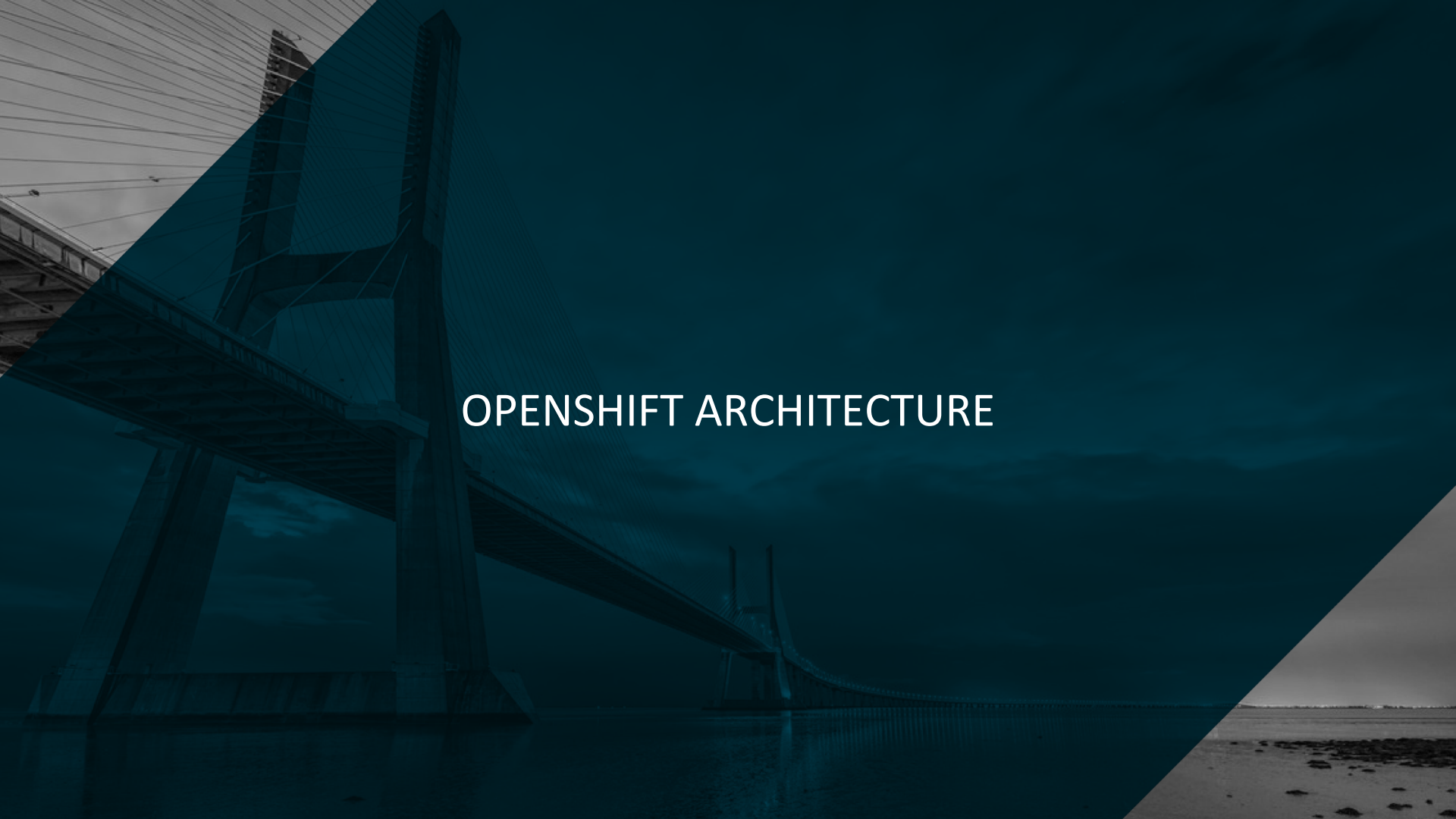


routes add services to the external load-balancer and provide readable urls for the app



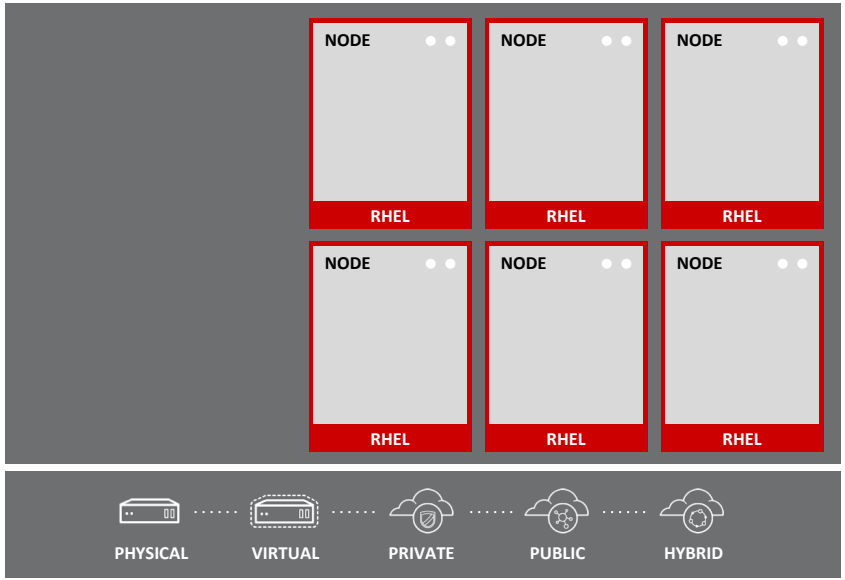
projects isolate apps across environments, teams, groups and departments



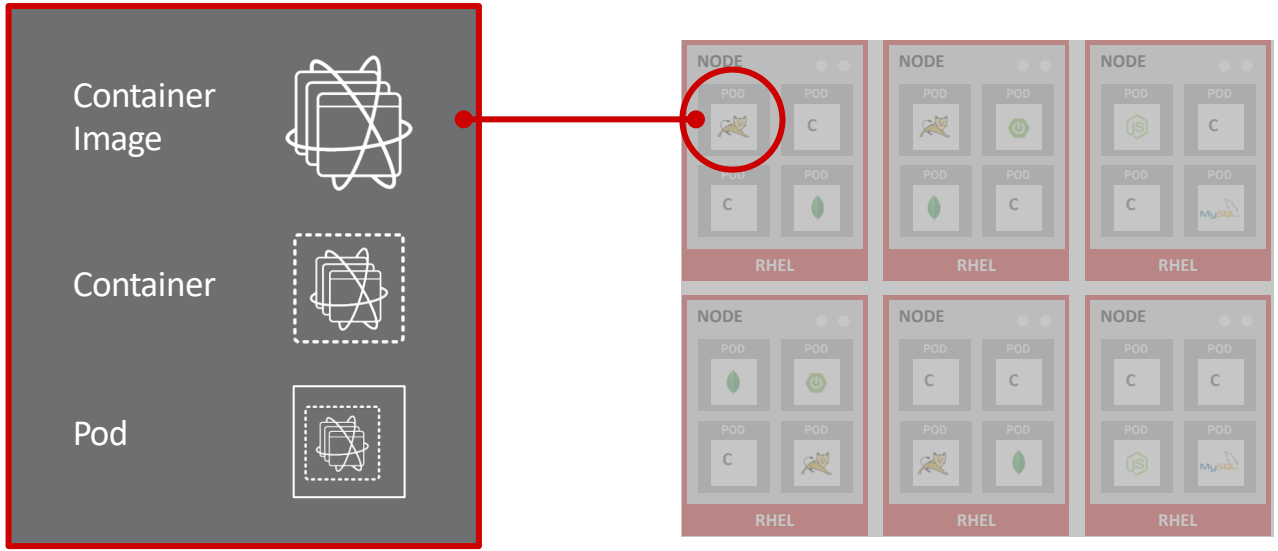

OPENSHIFT ARCHITECTURE

YOUR CHOICE OF INFRASTRUCTURE

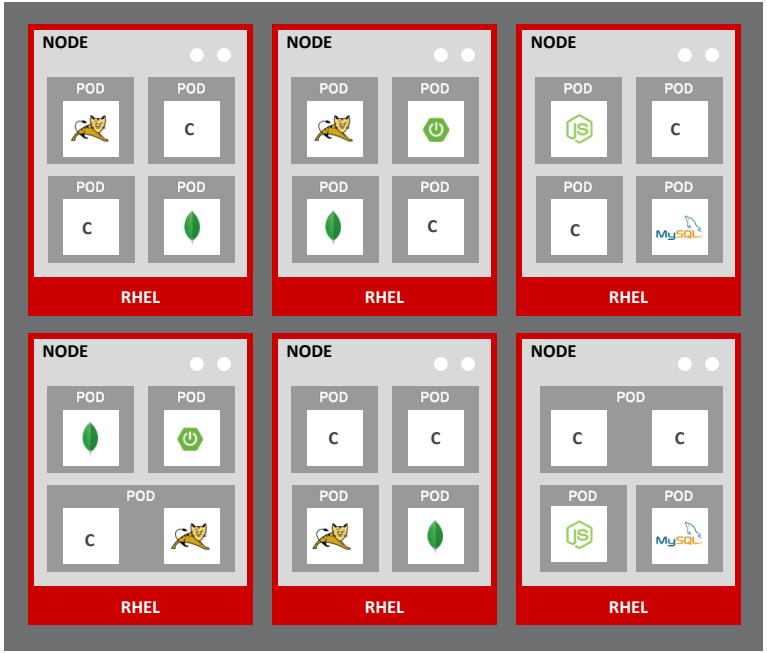
NODES RHEL INSTANCES WHERE APPS RUN



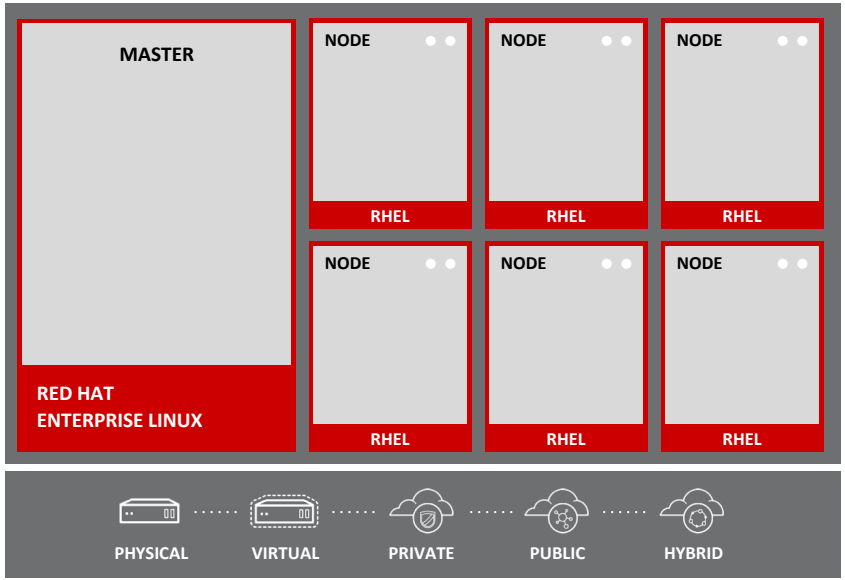
APPS RUN IN CONTAINERS



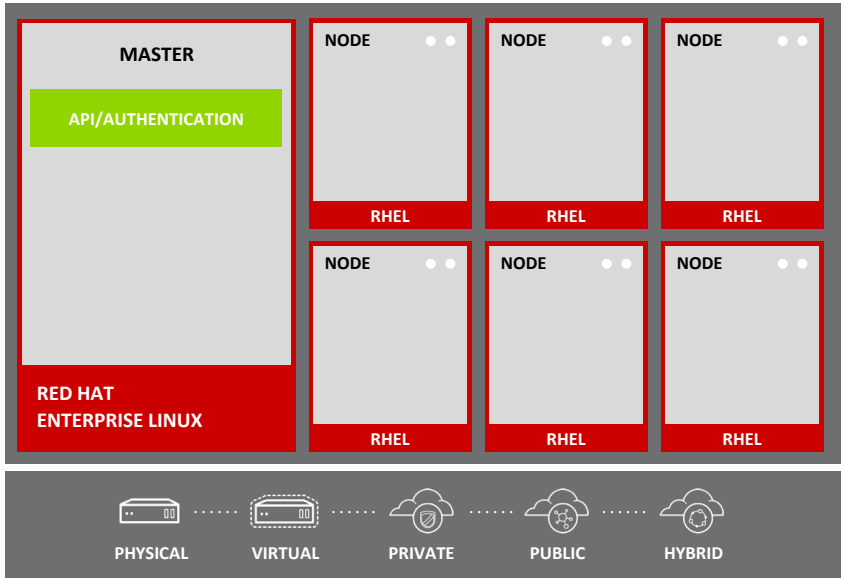
PODS ARE THE UNIT OF ORCHESTRATION



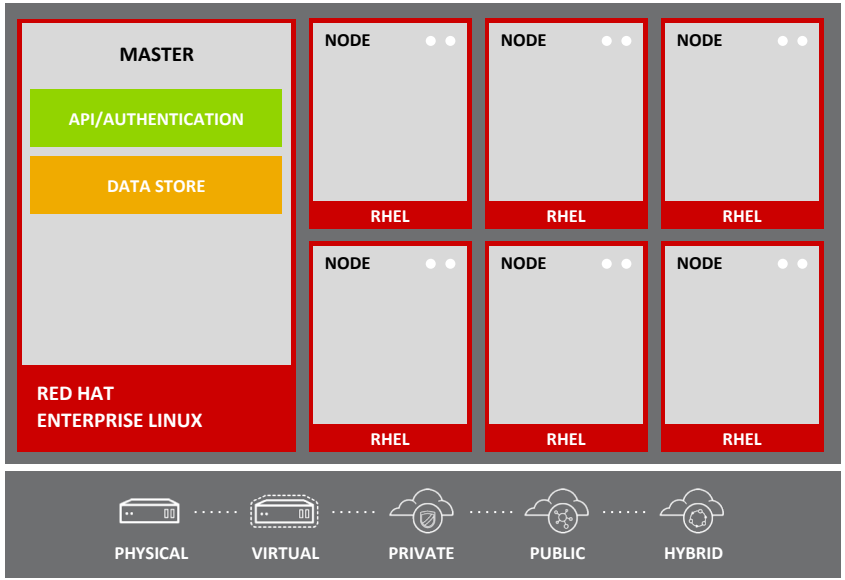
MASTERS ARE THE CONTROL PLANE



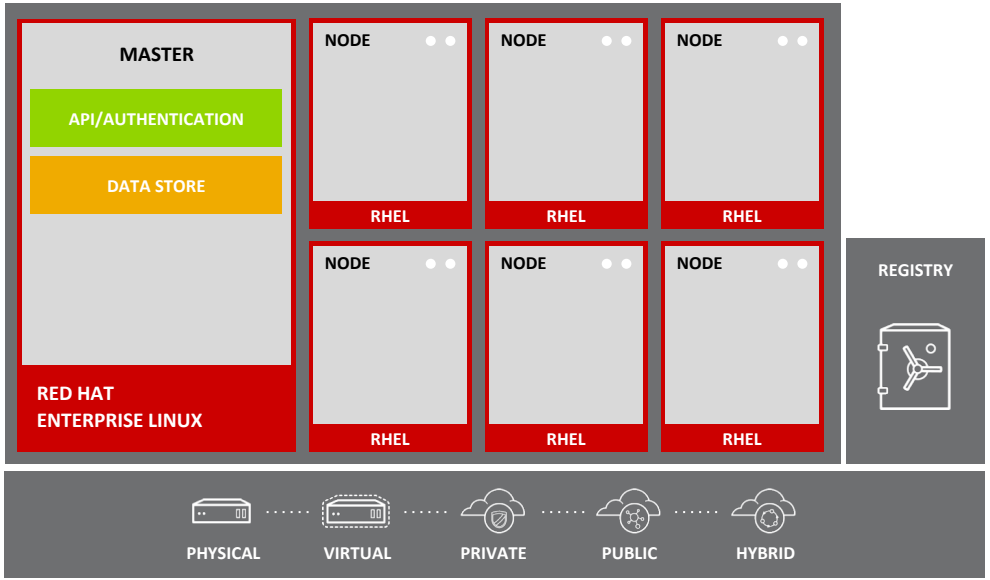
API AND AUTHENTICATION



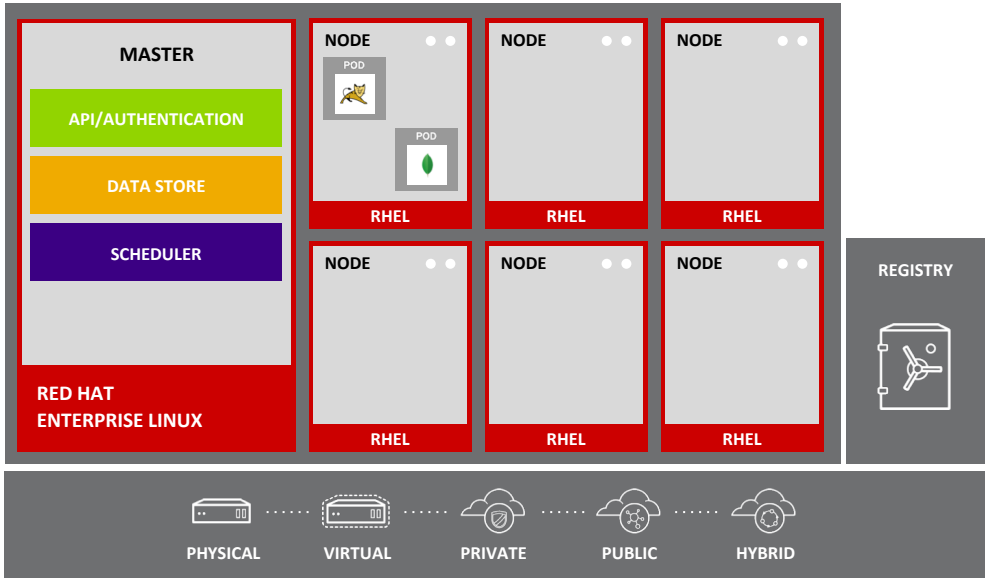
DESIRED AND CURRENT STATE



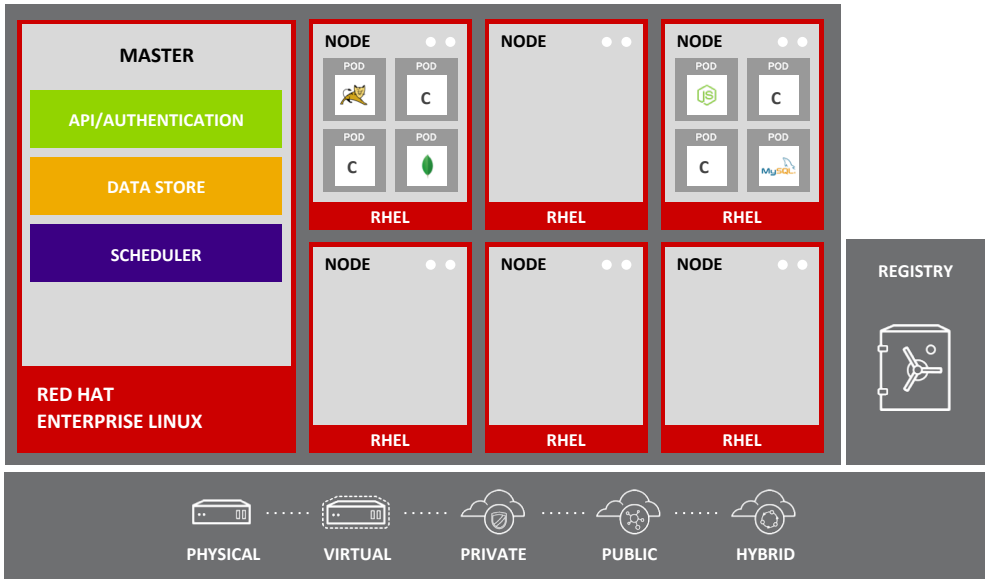
INTEGRATED CONTAINER REGISTRY



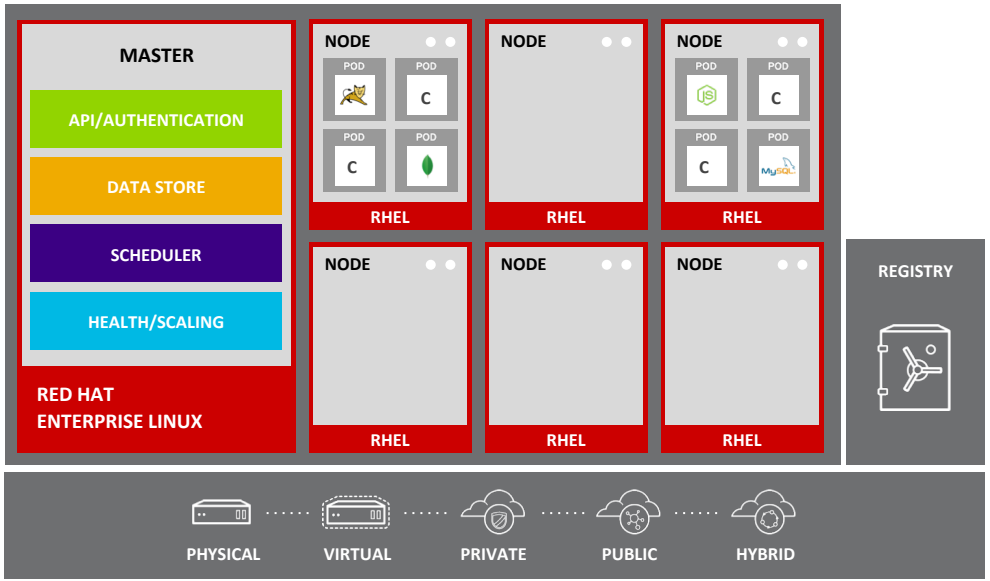
ORCHESTRATION AND SCHEDULING



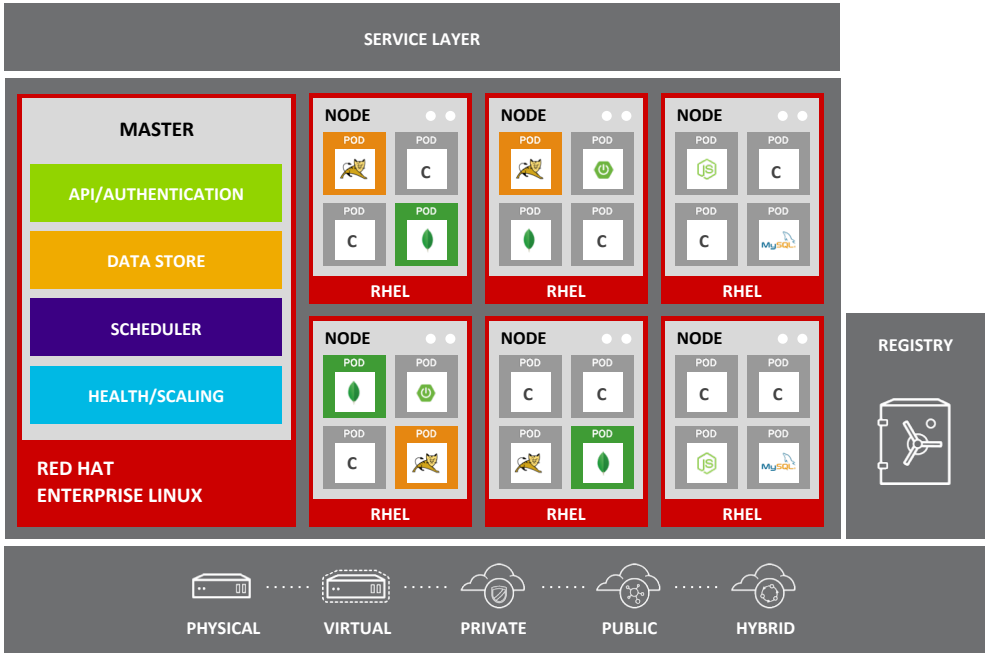
PLACEMENT BY POLICY



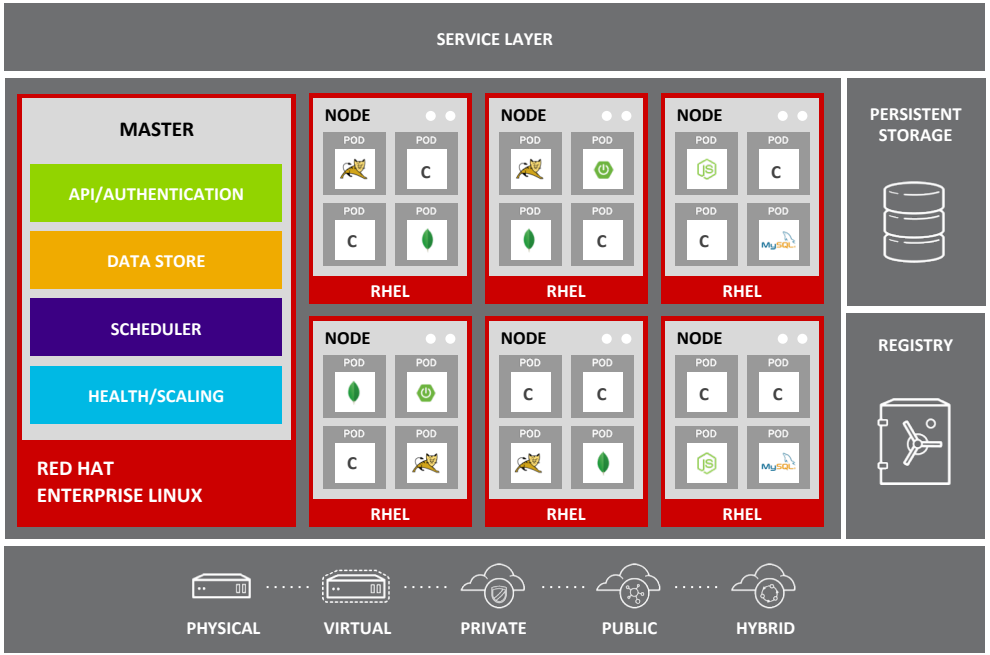
AUTOSCALING PODS



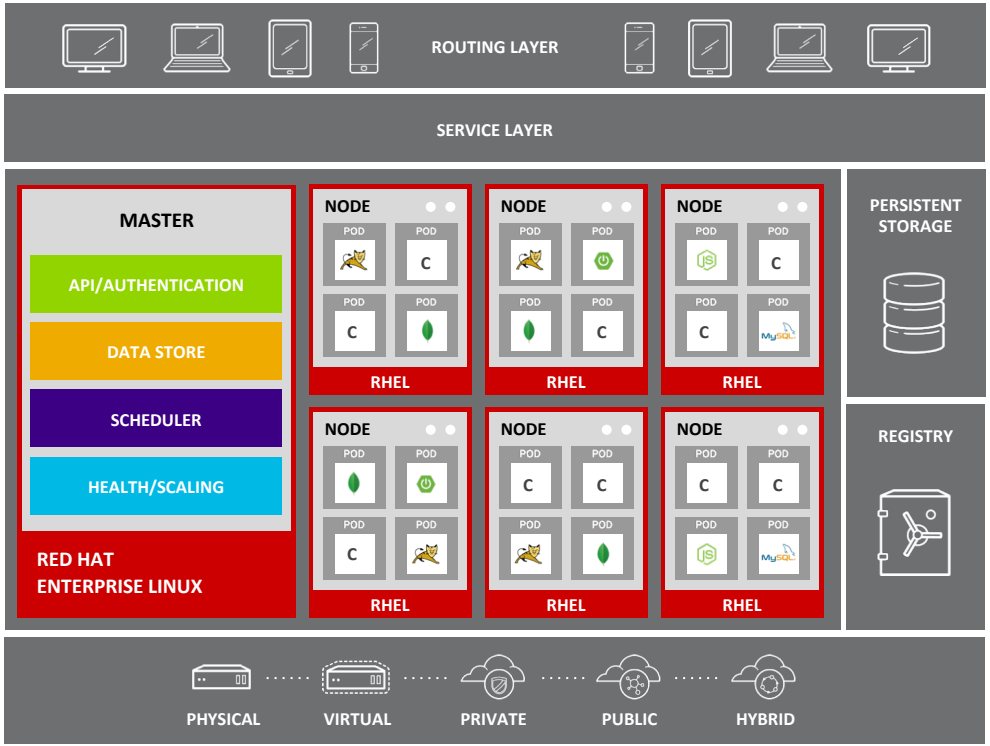
SERVICE DISCOVERY



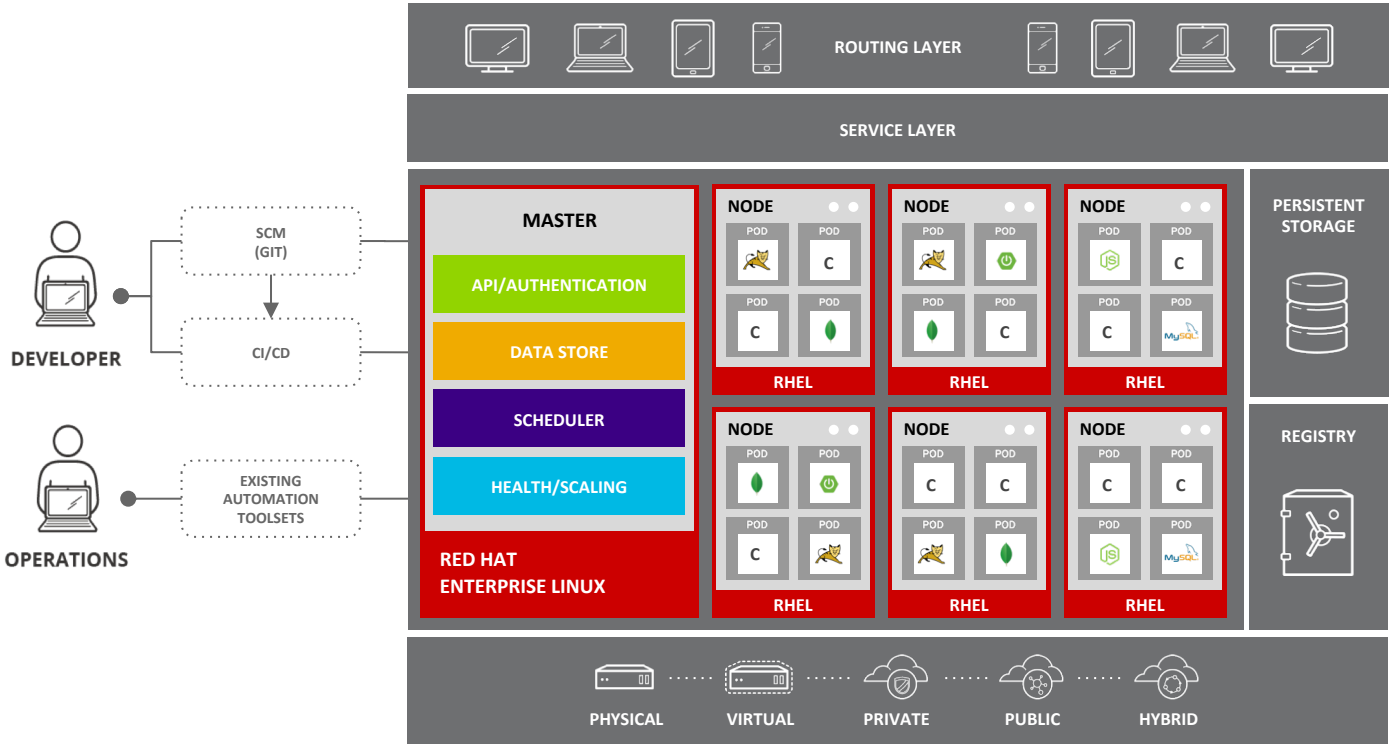
PERSISTENT DATA IN CONTAINERS



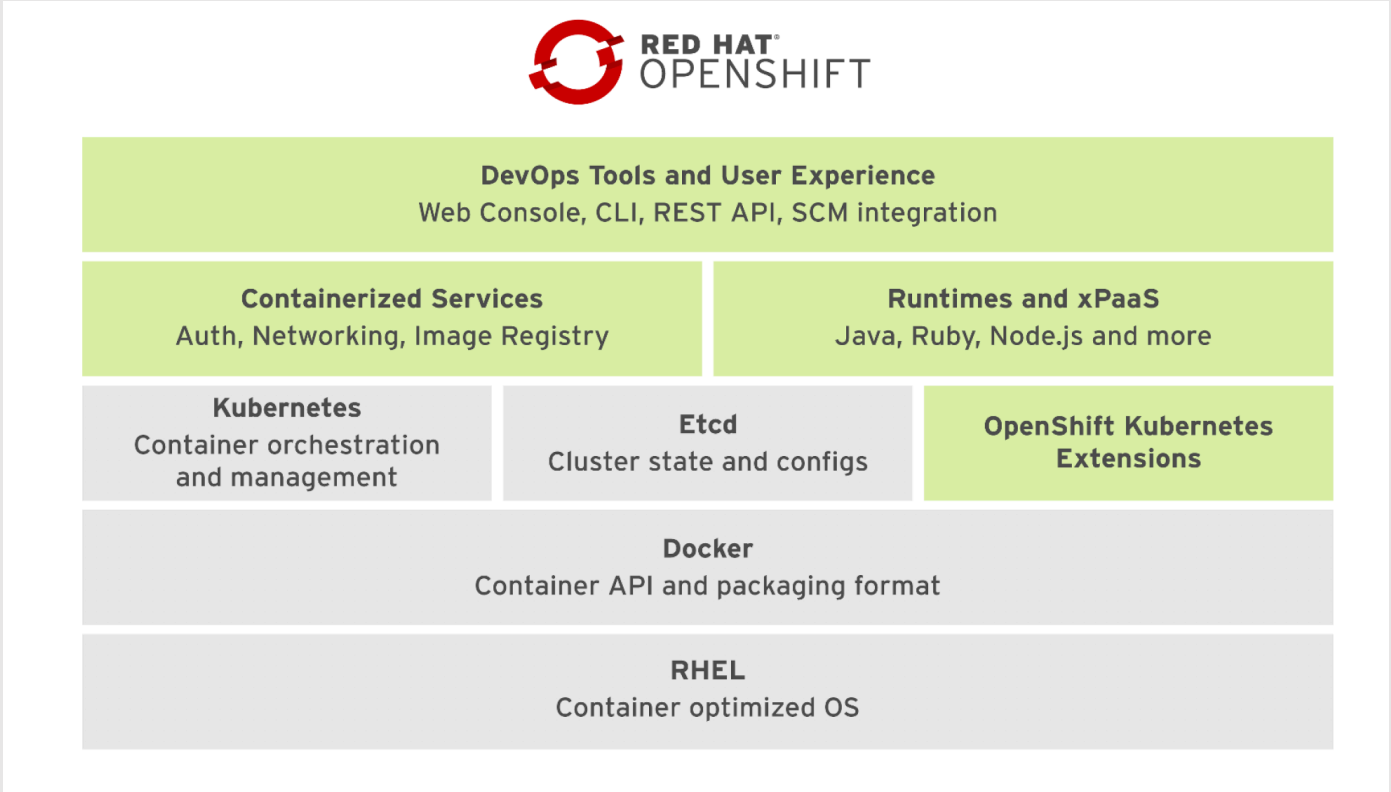
ROUTING AND LOAD-BALANCING



ACCESS VIA WEB, CLI, IDE AND API

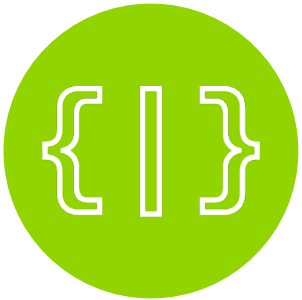


OpenShift Container Platform – why enterprise grade



Building Container Images with OpenShift

BUILD AND DEPLOY CONTAINER IMAGES

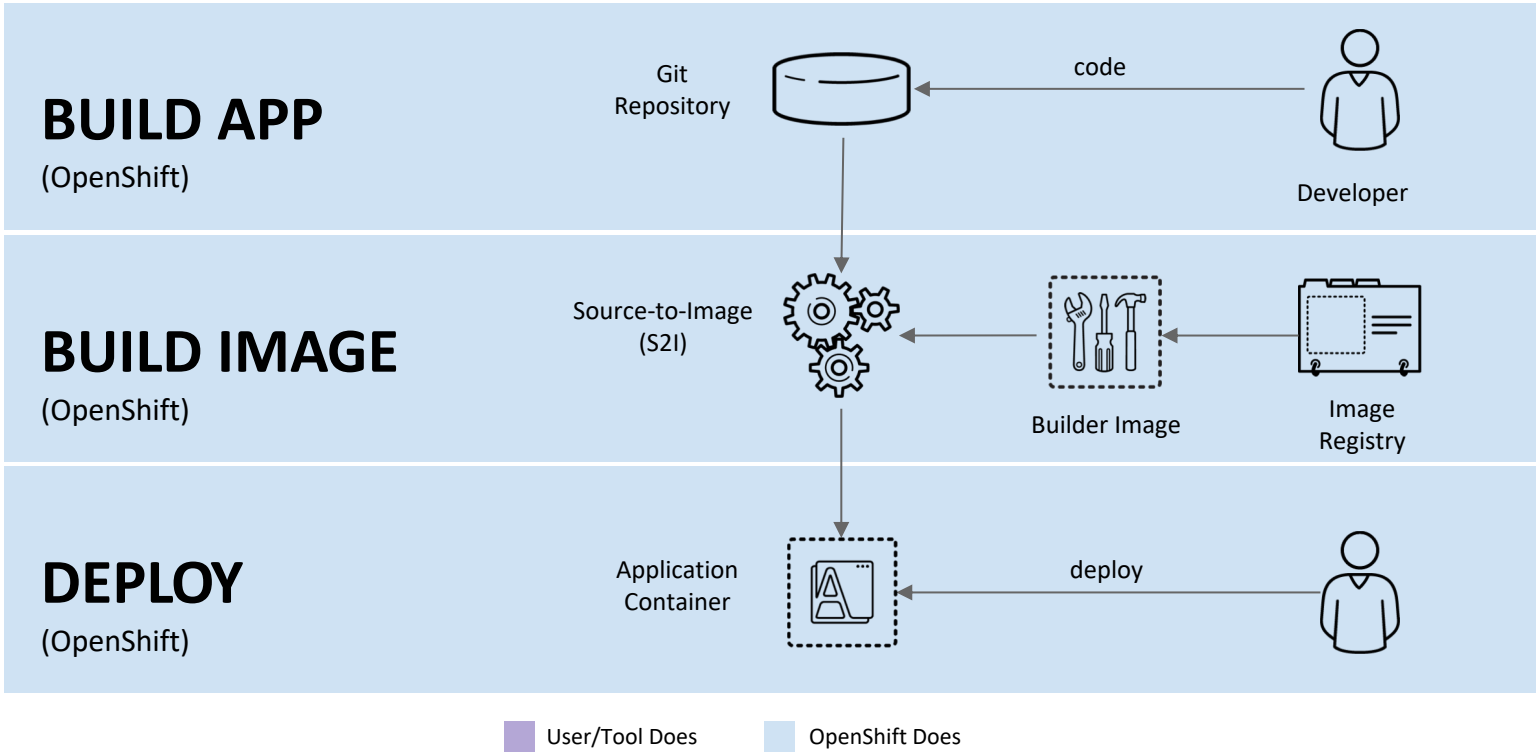


DEPLOY YOUR SOURCE CODE

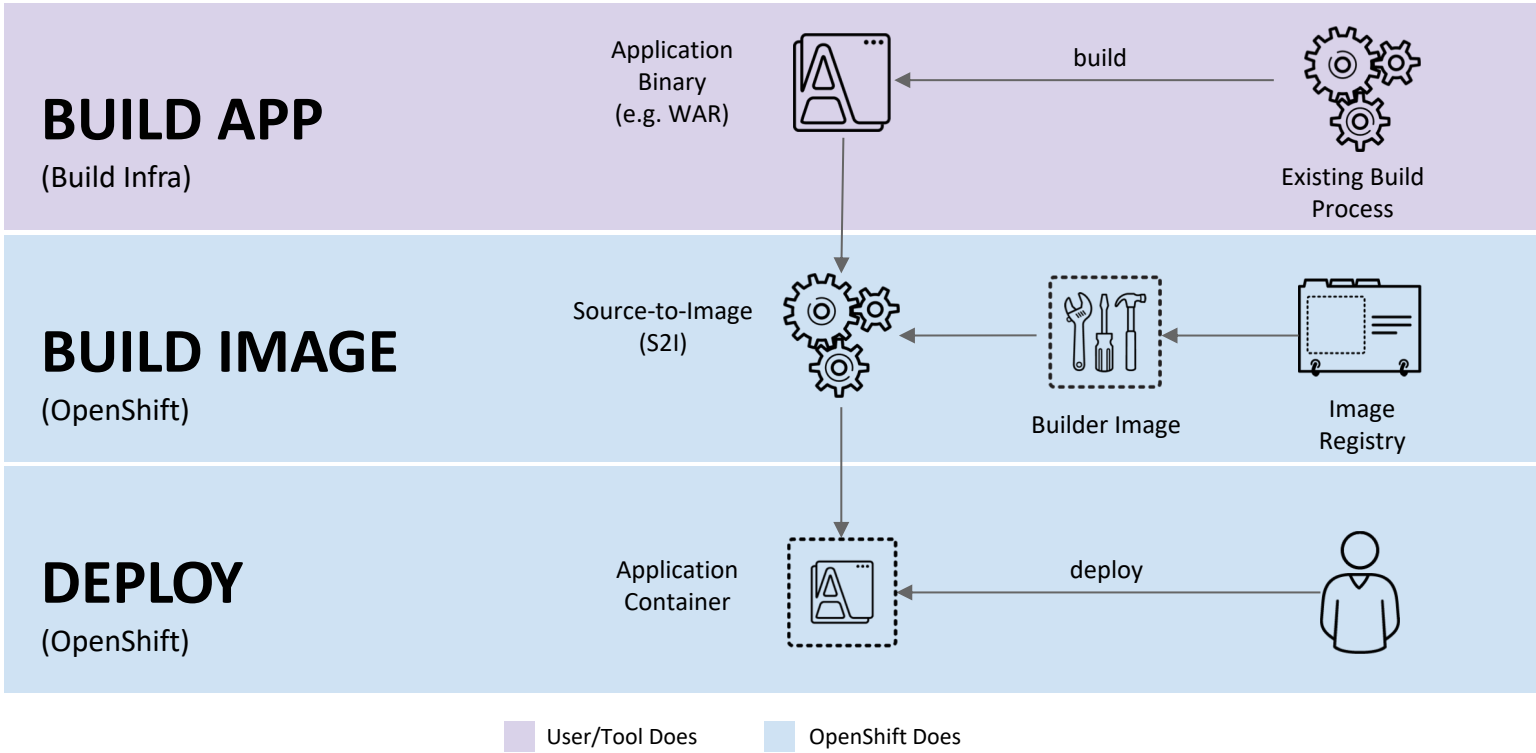
DEPLOY YOUR APP BINARY

DEPLOY YOUR CONTAINER IMAGE

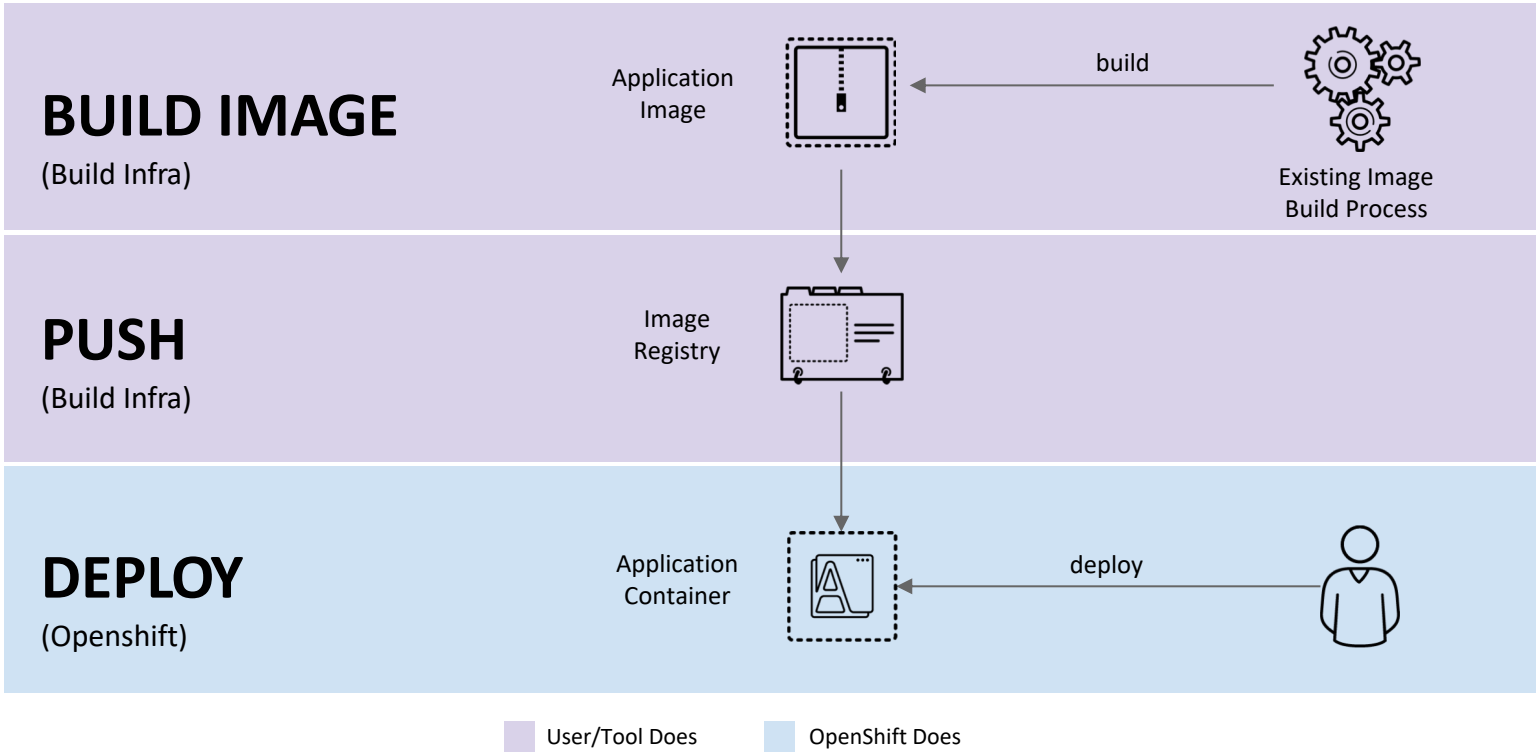
DEPLOY SOURCE CODE WITH SOURCE-TO-IMAGE (S2I)



DEPLOY APP BINARY WITH SOURCE-TO-IMAGE (S2I)



DEPLOY DOCKER IMAGE



BUILD STRATEGIES

The following are available build strategies in OpenShift:

- Source
- Docker
- Custom

* Pipeline

BUILDCONFIG RESOURCE

BuildConfig object definition

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: "ruby-sample-build" 1
spec:
  runPolicy: "Serial" 2
  triggers: 3
    -
      type: "GitHub"
      github:
        secret: "secret101"
    - type: "Generic"
      generic:
        secret: "secret101"
    -
      type: "ImageChange"
  source: 4
    git:
      uri: "https://github.com/openshift/ruby-hello-world"
  strategy: 5
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "ruby-20-centos7:latest"
  output: 6
    to:
      kind: "ImageStreamTag"
      name: "origin-ruby-sample:latest"
  postCommit: 7
      script: "bundle exec rake test"
```

1. This specification creates a new `BuildConfig` named **ruby-sample-build**.
2. The `runPolicy` field controls whether builds created from this build configuration can be run simultaneously. The default value is `Serial`, which means new builds run sequentially, not simultaneously.
3. You can specify a list of triggers, which cause a new build to be created.
4. The `source` section defines the source of the build. The source type determines the primary source of input, and can be either `Git`, to point to a code repository location, `Dockerfile`, to build from an inline Dockerfile, or `Binary`, to accept binary payloads. It is possible to have multiple sources at once. Refer to the documentation for each source type for details.
5. The `strategy` section describes the build strategy used to execute the build. You can specify a `Source` , `Docker`, or `Custom` strategy here. This example uses the `ruby-20-centos7` container image that Source-To-Image uses for the application build.
6. After the container image is successfully built, it is pushed into the repository described in the `output` section.
7. The `postCommit` section defines an optional build hook.

BUILD INPUT SOURCES

- **Git**: The input source is cloned from a Git repository. It is possible to configure the default location inside the repository where the build looks for application source code.
- **Dockerfile**: Specifies the Dockerfile inline to build an image. This Dockerfile overrides a Dockerfile from a Git repository.
- **Binary**: Allows streaming binary content from a local file system to the builder.
- **Image**: Additional files can be provided to the build process from images.
- **Input secrets**: Allow creating credentials for the build that will not be available in the final application image.
- **External artifacts**: Allow copying binary files to the build process.
  Combining multiple inputs into a single build is possible.
  Binary and Git are mutually exclusive inputs.

Source definition: An example

```
source:
  git:
    uri: https://github.com/openshift/ruby-hello-world.git (1)
  images:
  - from:
      kind: ImageStreamTag
      name: myinputimage:latest
      namespace: mynamespace
    paths:
    - destinationDir: app/dir/injected/dir (2)
      sourcePath: /usr/lib/somefile.jar
  contextDir: "app/dir" (3)
  dockerfile: "FROM centos:7\nRUN yum install -y httpd" (4)
```

1. The repository to be cloned into the working directory for the build.
2. */usr/lib/somefile.jar* from `myinputimage` will be stored in *<workingdir>/app/dir/injected/dir*.
3. The working directory for the build will become *<original_workingdir>/app/dir*.
4. A Dockerfile with this content will be created in *<original_workingdir>/app/dir*, overwriting any existing file with that name.

TRIGGERING BUILDS

- **Image change triggers**
- **Configuration change triggers**
- **Web hooks**

Post-commit Build hooks

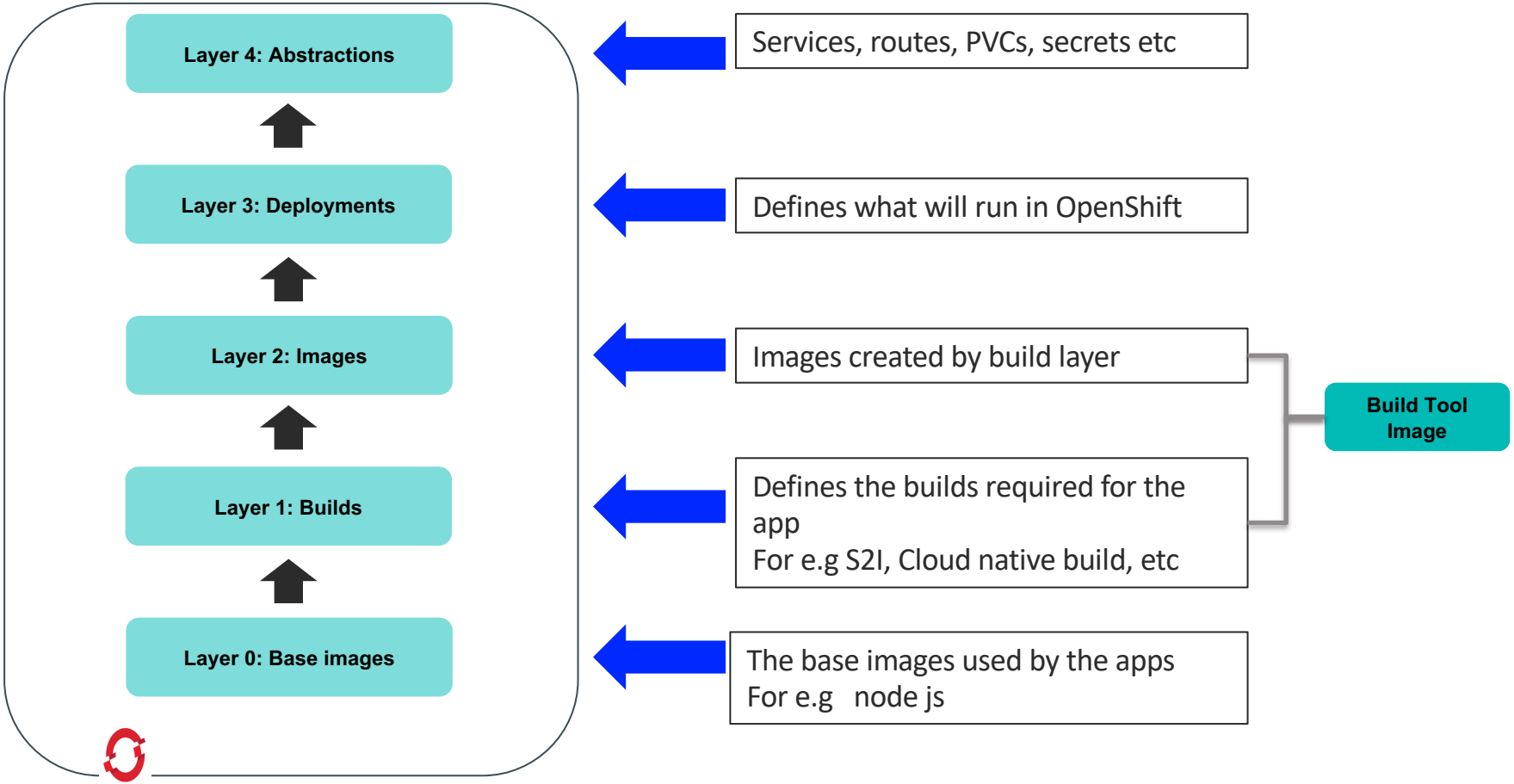
There are two types of Post-commit build hooks:

1) Command: A command is executed using the exec system call

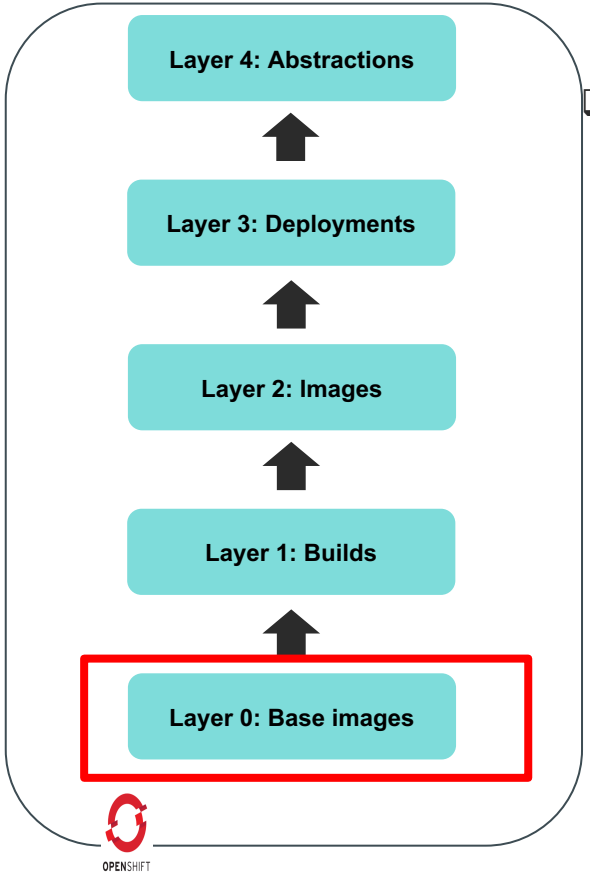
2) Script: Runs a build hook with the **/bin/sh -ic** command.

Think of OpenShift deployment as a set of layers

OpenShift Deployment – Developer view



Layer 0 – Base image examples



- Base operating systems, specific runtimes, databases, application servers and more
  - Available as **Image Streams**, an enhanced set of metadata about each image

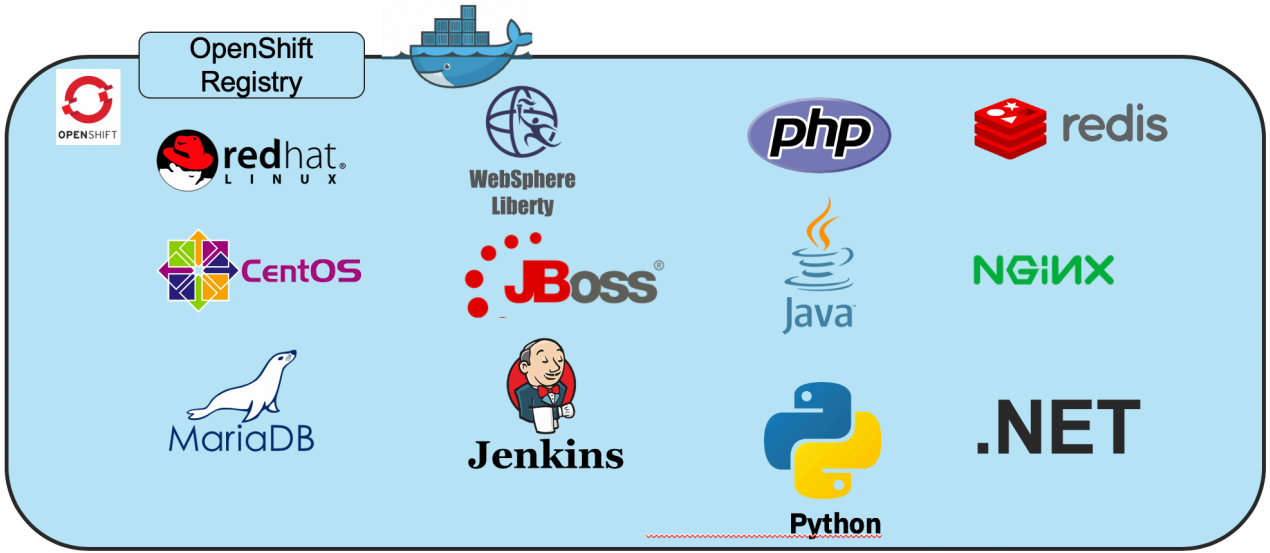
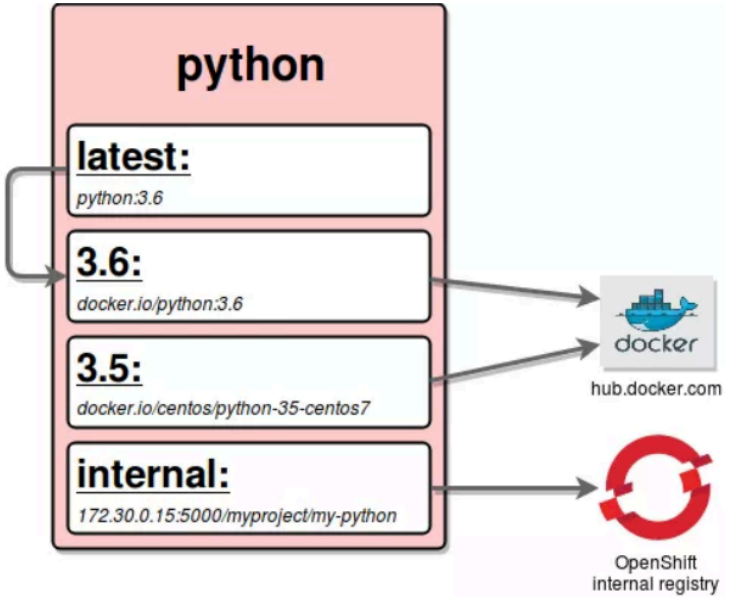


Image Streams

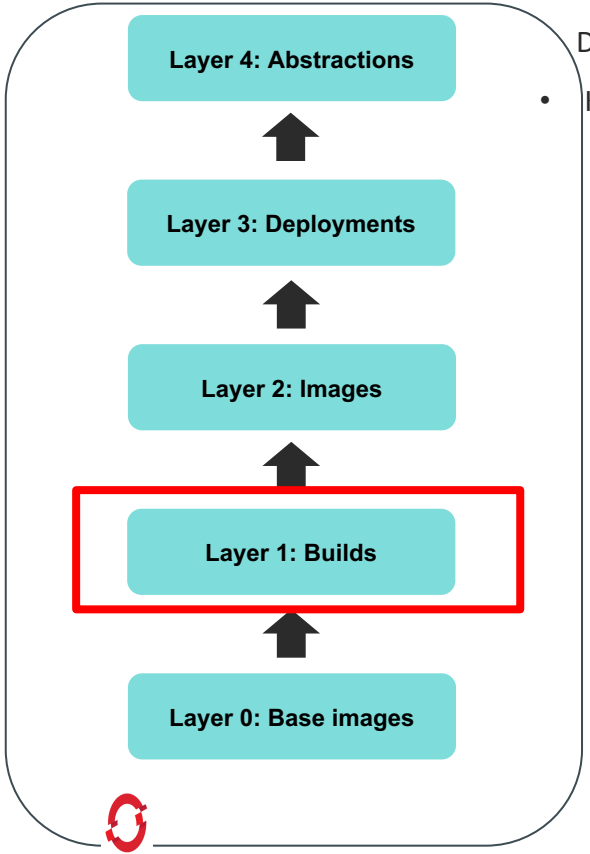
- An image stream represents one or more Docker images identified by tags.
  - Presents a single virtual view of related images
    - Can refer to images from any of the following:
      - Its own image repository in OpenShift's integrated Docker Registry
      - Other image streams
      - Docker image repositories from external registries
    - Image Streams trigger an event when underlying image is changed (even if tag remains the same)

Image Stream example

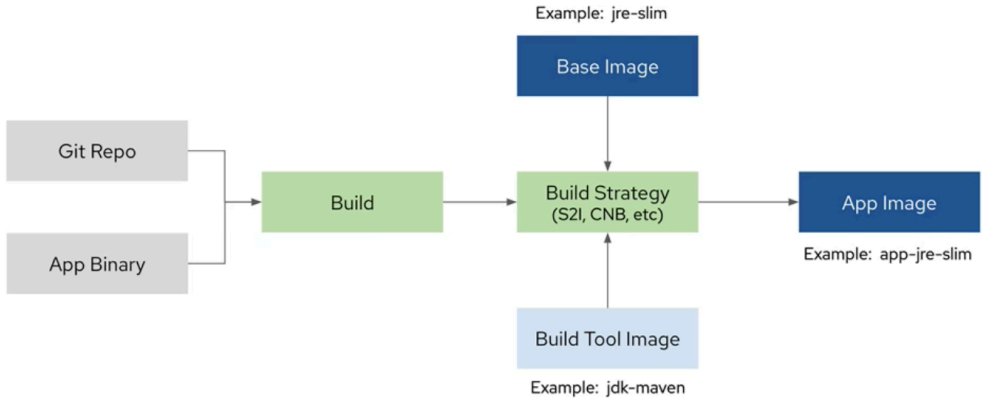


Graphic source: https://blog.openshift.com/image-streams-faq/

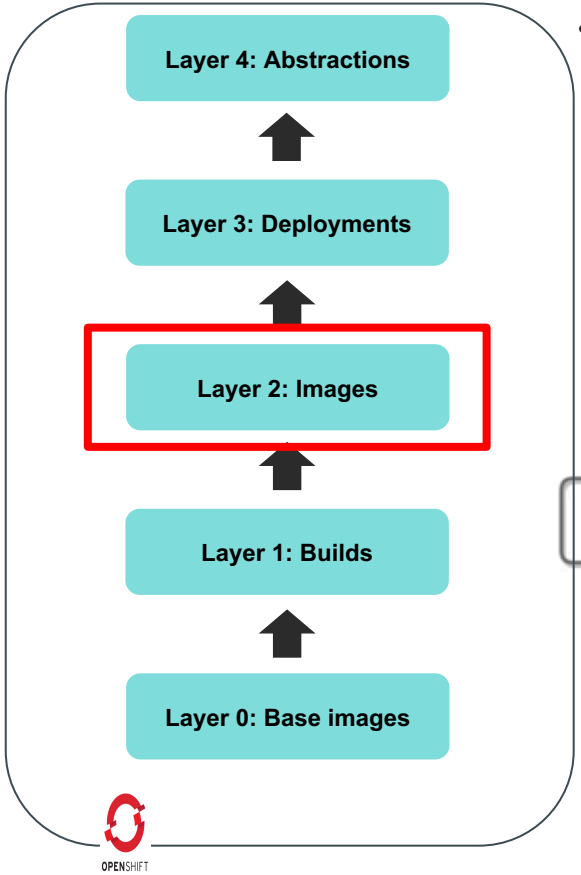
Level 1: Builds



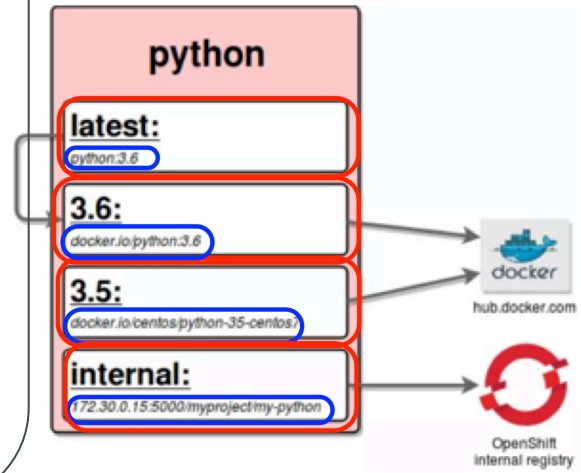
- Defined via a **BuildConfig** object
- A blueprint for a process to transform base images + source code, app binaries, or Dockerfiles to an app image:
- Key attributes:
  - **Input** (input to the build process) - e.g. file, directory, GH repo etc
  - **Strategy** (how to build the app image) -options
    - *S2I* – Use a specialized builder image to generate app image
    - *Docker* - Use a Docker file to generate app image
    - *Pipeline* - A Jenkins pipeline that generates the app image from source
    - *Custom* – Encapsulate your build process via a custom builder image
  - **Output** (result of the build process) – typically an ImageStream tag



Level 2: Images



- Defined via an **ImageStream** object
  - An abstraction for working with Docker images inside OpenShift
  - Key attributes:
    - **ImageStreamImage** (reference to actual image) - typically not used directly
    - **ImageStreamTag** (reference to a given ImageStream and tag)

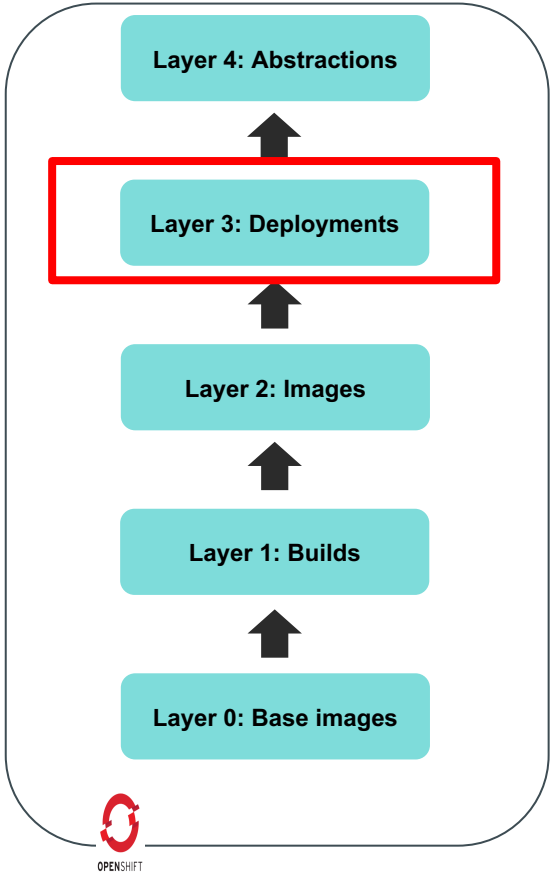


Key:

ImageStreamImage

ImageStreamTag

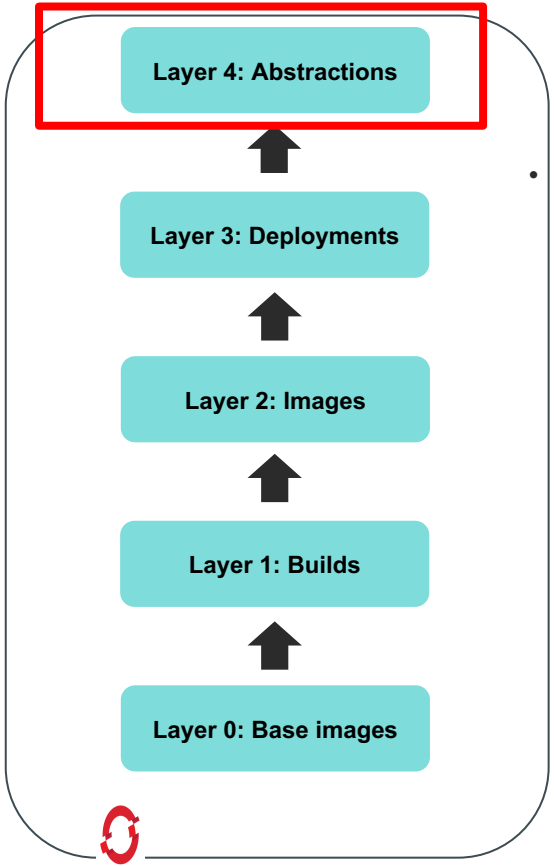
Level 3: Deployments



- Defined via a **DeploymentConfig** object
  - Encapsulates the K8s Deployment and adds deployment strategies and triggers
  - Key attributes:
    - **Strategy** (how to deploy if the underlying Deployment is already deployed) Options:
      - *Recreate*– Blow away old deployment first and then deploy new one
      - *Rolling (default)* – Zero downtime rollout via K8s *RollingUpdate*
      - *Advanced* - (Note: require routes)
        - *Blue-Green*
        - *A/B*
        - *etc*
    - **Triggers** (define conditions under which the deployment is automatically triggered)

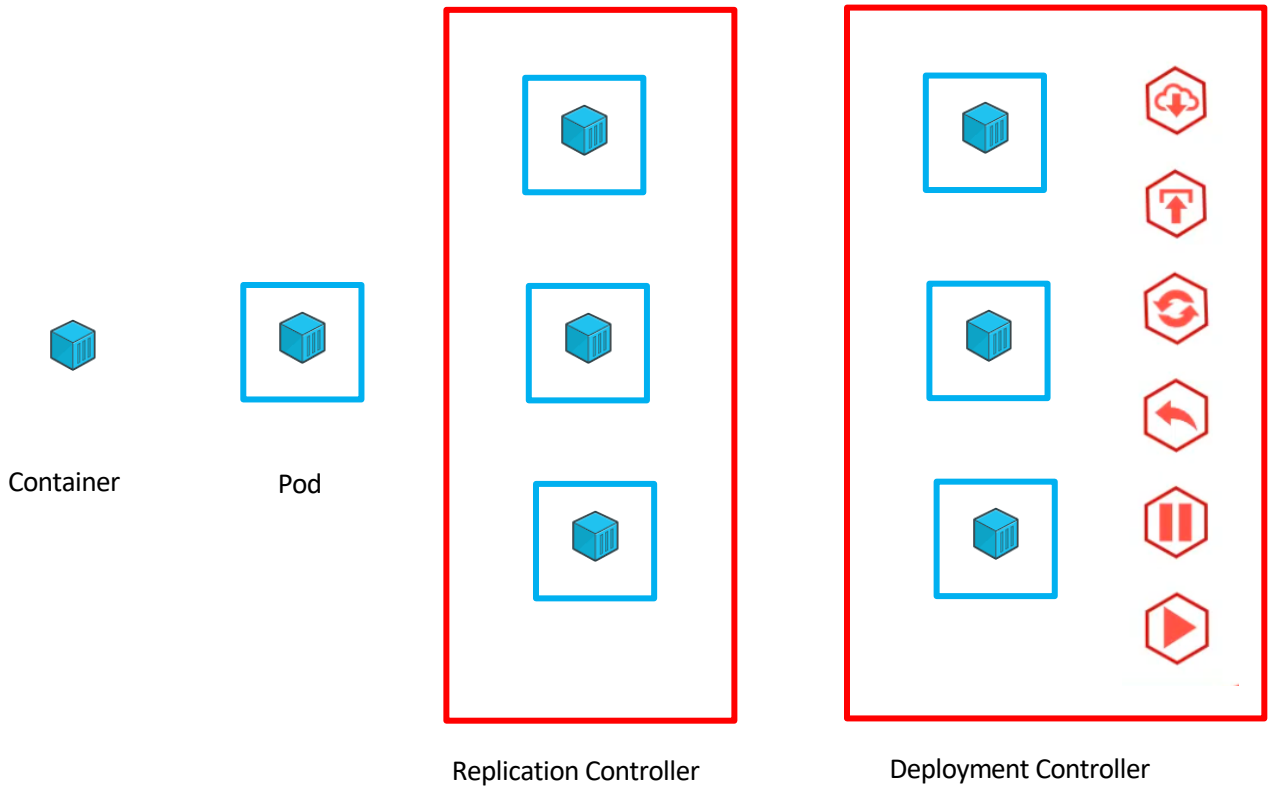
      e.g Input image updated, config changes

Level 4: Abstractions

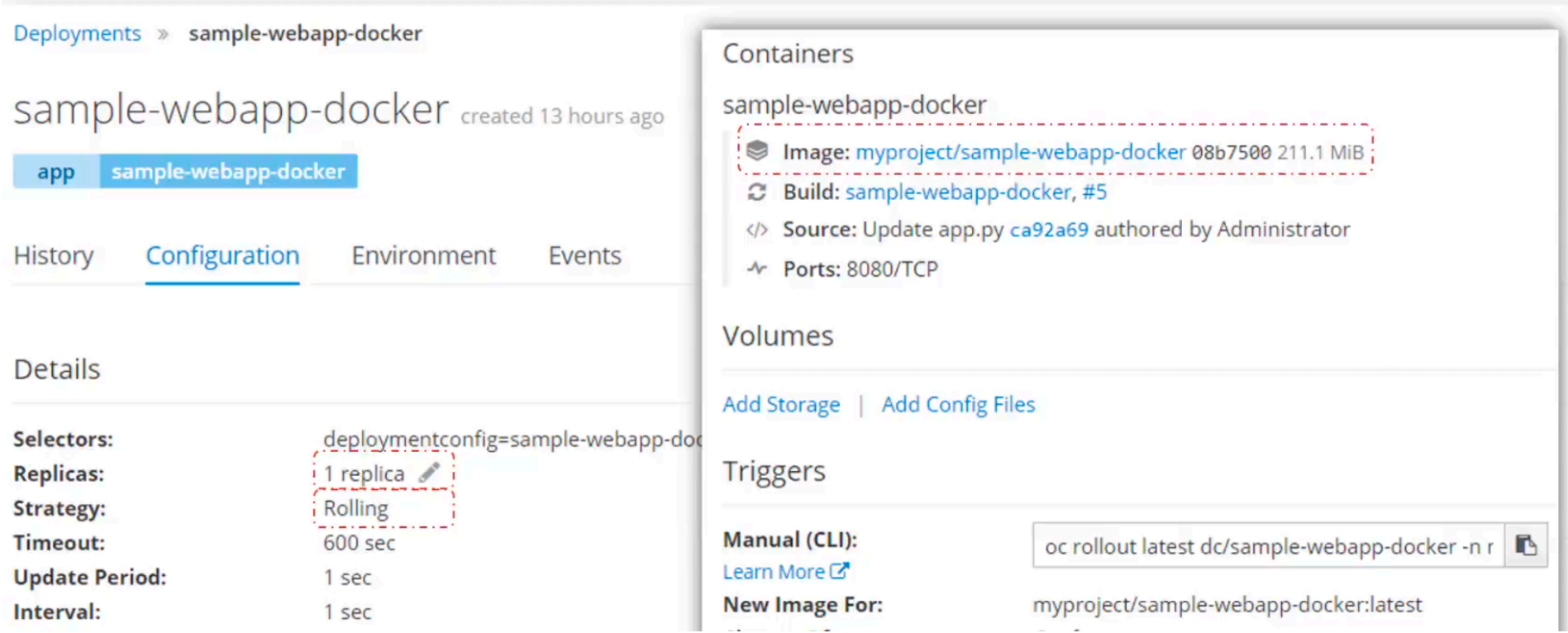


- Defines all of the additional resources needed for an app like networking, storage, security etc
  - Frequently used resources: (there are several others)
    - **Service** – Internal load balancer and router. Directs network traffic to replicated Pods
    - **Route** – Exposes a Service at a defined host name to allow access from external traffic
    - **Persistent Volume Claim** – Permanent storage used by apps to persist data after the app has stopped running
    - **Secret** – Mechanism for holding sensitive data. Decouples sensitive data from the the Pods that use them.

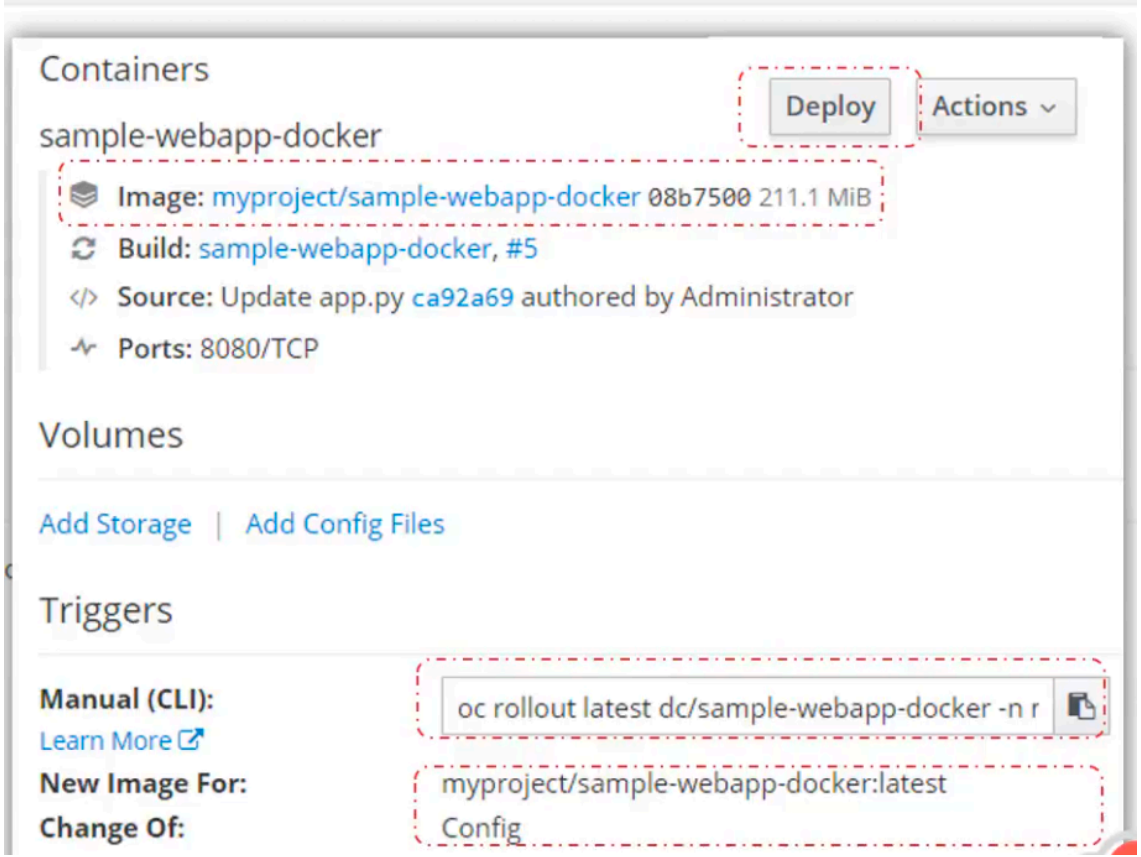
Deployment Controller



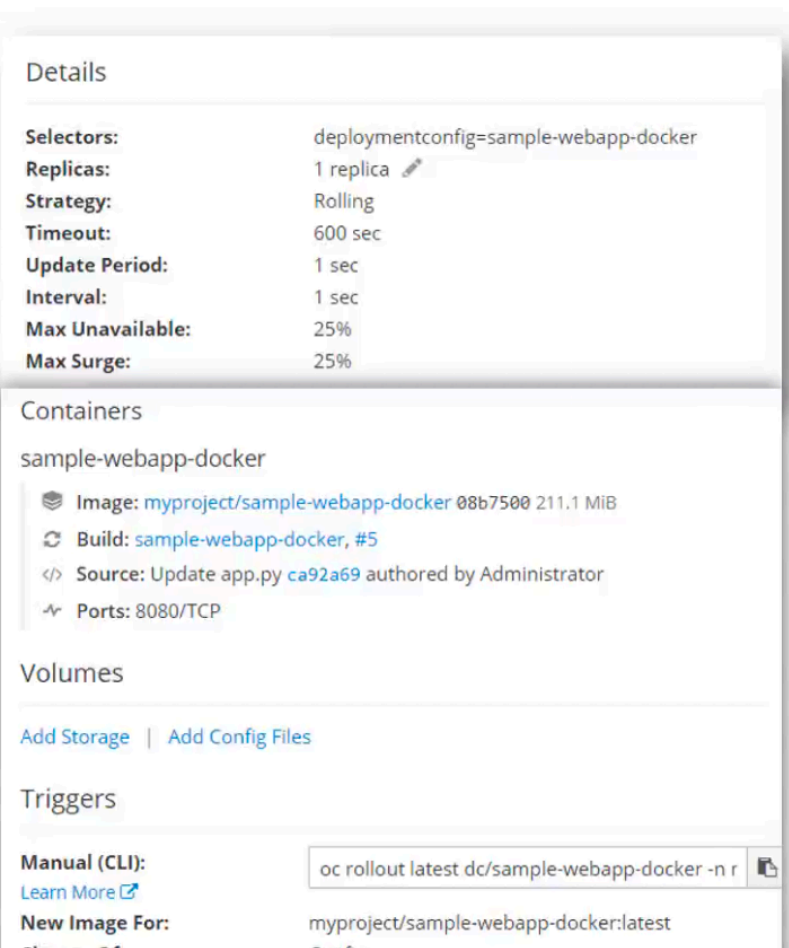
Deployment


Deployments » sample-webapp-docker
sample-webapp-docker created 13 hours ago
app sample-webapp-docker
History Configuration Environment Events
Details
Selectors: deploymentconfig=sample-webapp-doc
Replicas: 1 replica
Strategy: Rolling
Timeout: 600 sec
Update Period: 1 sec
Interval: 1 sec
Containers
sample-webapp-docker
Image: myproject/sample-webapp-docker 08b7500 211.1 MiB
Build: sample-webapp-docker, #5
Source: Update app.py ca92a69 authored by Administrator
Ports: 8080/TCP
Volumes
Add Storage | Add Config Files
Triggers
Manual (CLI):
oc rollout latest dc/sample-webapp-docker -n r
Learn More
New Image For:
myproject/sample-webapp-docker:latest

Deployment


Containers
Deploy
Actions
sample-webapp-docker
Image: myproject/sample-webapp-docker 08b7500 211.1 MiB
Build: sample-webapp-docker, #5
Source: Update app.py ca92a69 authored by Administrator
Ports: 8080/TCP
Volumes
Add Storage | Add Config Files
Triggers
Manual (CLI):
oc rollout latest dc/sample-webapp-docker -n r
Learn More
New Image For:
myproject/sample-webapp-docker:latest
Change Of:
Config

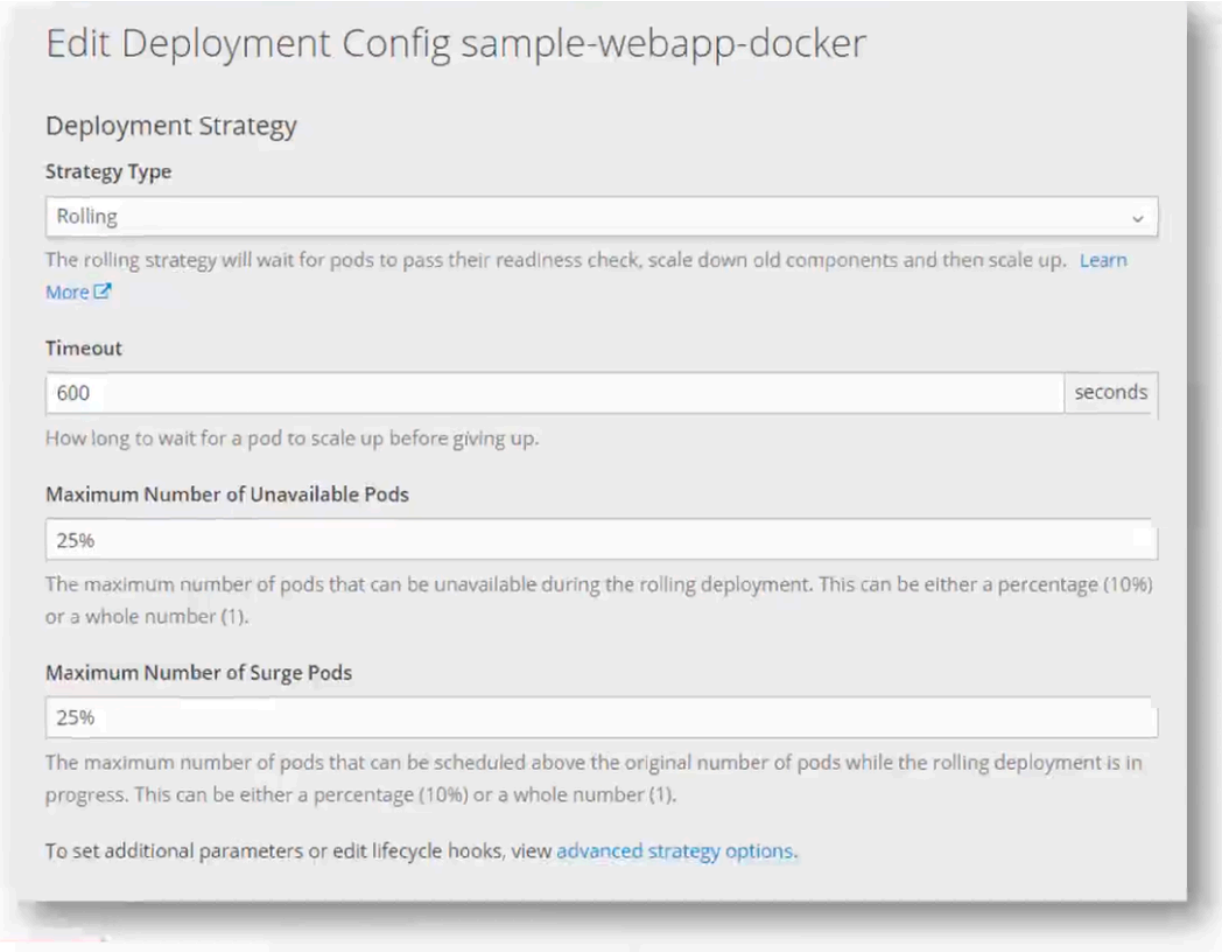
Deployment YAML



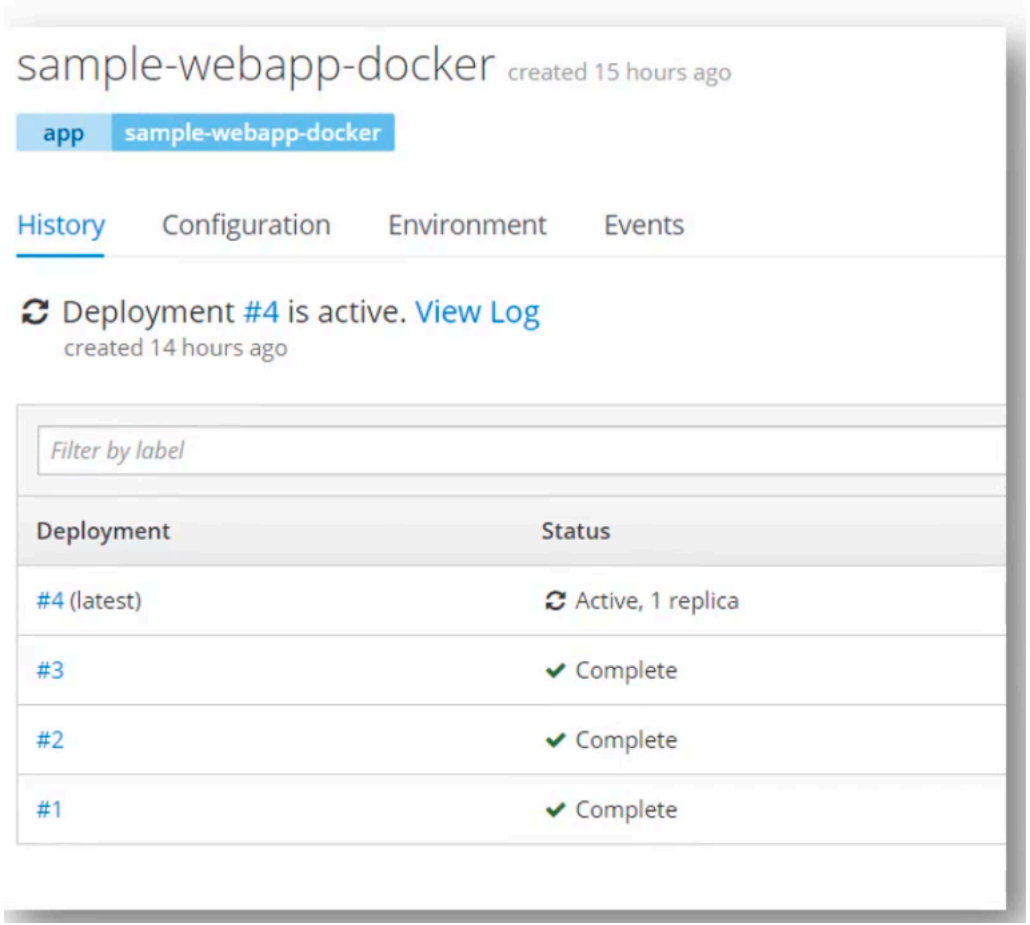
deployment-config.yaml

```yaml
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
metadata:
  name: sample-webapp-docker
spec:
  replicas: 1
  selector:
    deploymentconfig: sample-webapp-docker
  strategy:
    type: Rolling
  template:
    metadata:
      labels:
        app: sample-webapp-docker
        deploymentconfig: sample-webapp-docker
    spec:
      containers:
        - image: myproject/sample-webapp-docker
          imagePullPolicy: Always
          name: sample-webapp-docker
          ports:
            - containerPort: 8080
              protocol: TCP
  triggers:
    - imageChangeParams:
        automatic: true
        containerNames:
          - sample-webapp-docker
        from:
          kind: ImageStreamTag
          name: 'sample-webapp-docker:latest'
```

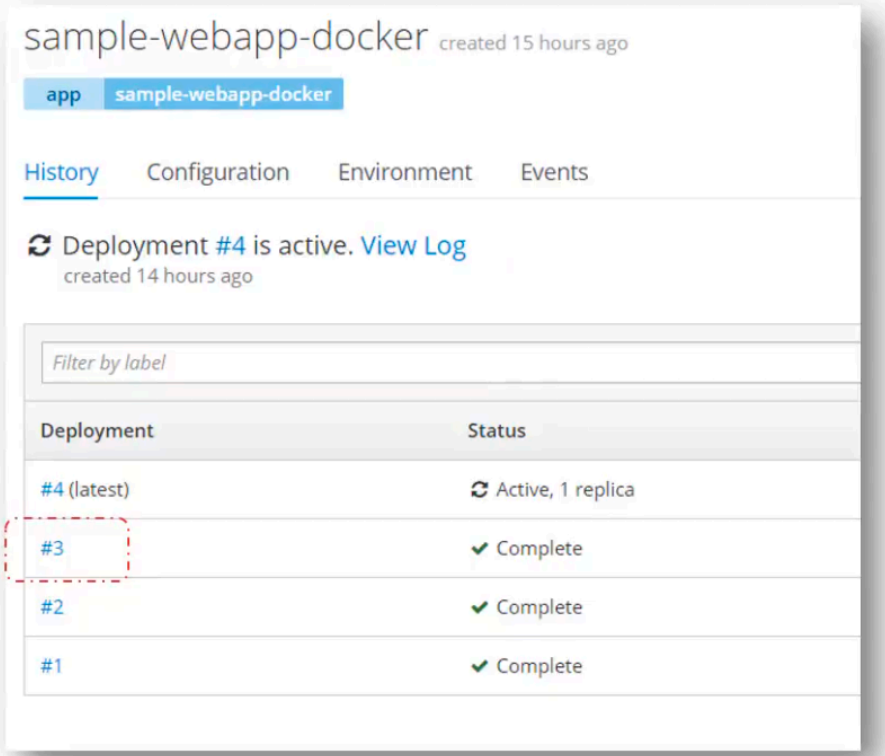
Edit Deployment Configuration

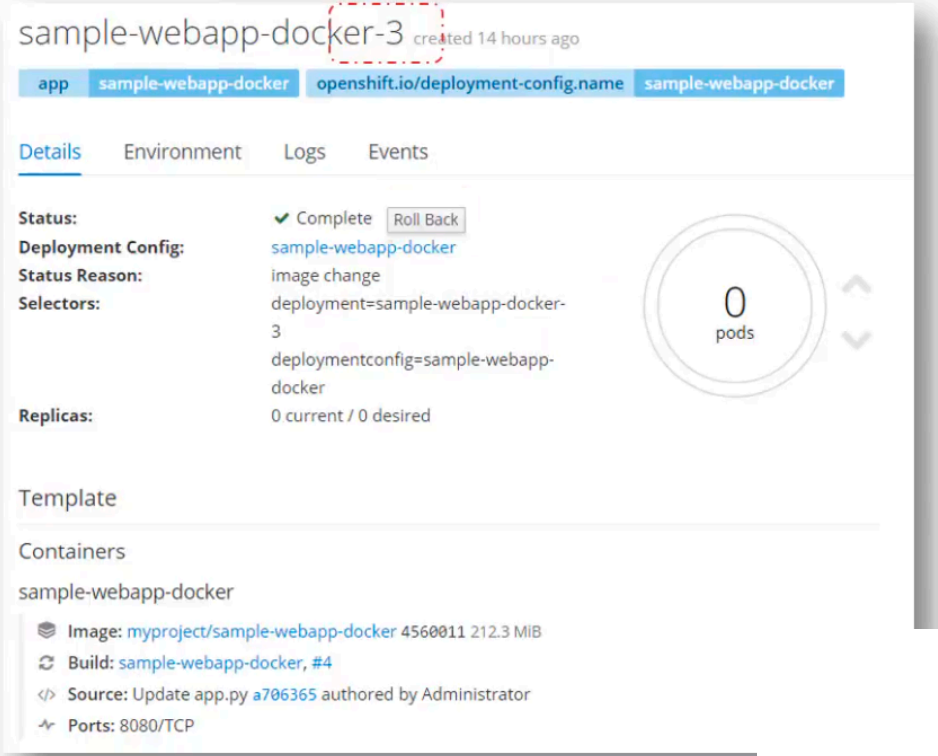


Deployment History



Rollback


sample-webapp-docker created 15 hours ago
app
sample-webapp-docker
History
Configuration
Environment
Events
Deployment #4 is active. View Log
created 14 hours ago
Filter by label
Deployment
Status
#4 (latest)
Active, 1 replica
#3
Complete
#2
Complete
#1
Complete


sample-webapp-docker-3 created 14 hours ago
app
sample-webapp-docker
openshift.io/deployment-config.name
sample-webapp-docker
Details
Environment
Logs
Events
Status:
Complete
Roll Back
Deployment Config:
sample-webapp-docker
Status Reason:
image change
Selectors:
deployment=sample-webapp-docker-3
deploymentconfig=sample-webapp-docker
Replicas:
0 current / 0 desired
0
pods
Template
Containers
sample-webapp-docker
Image: myproject/sample-webapp-docker 4560011 212.3 MiB
Build: sample-webapp-docker, #4
Source: Update app.py a706365 authored by Administrator
Ports: 8080/TCP

Deployment Strategy – Why?

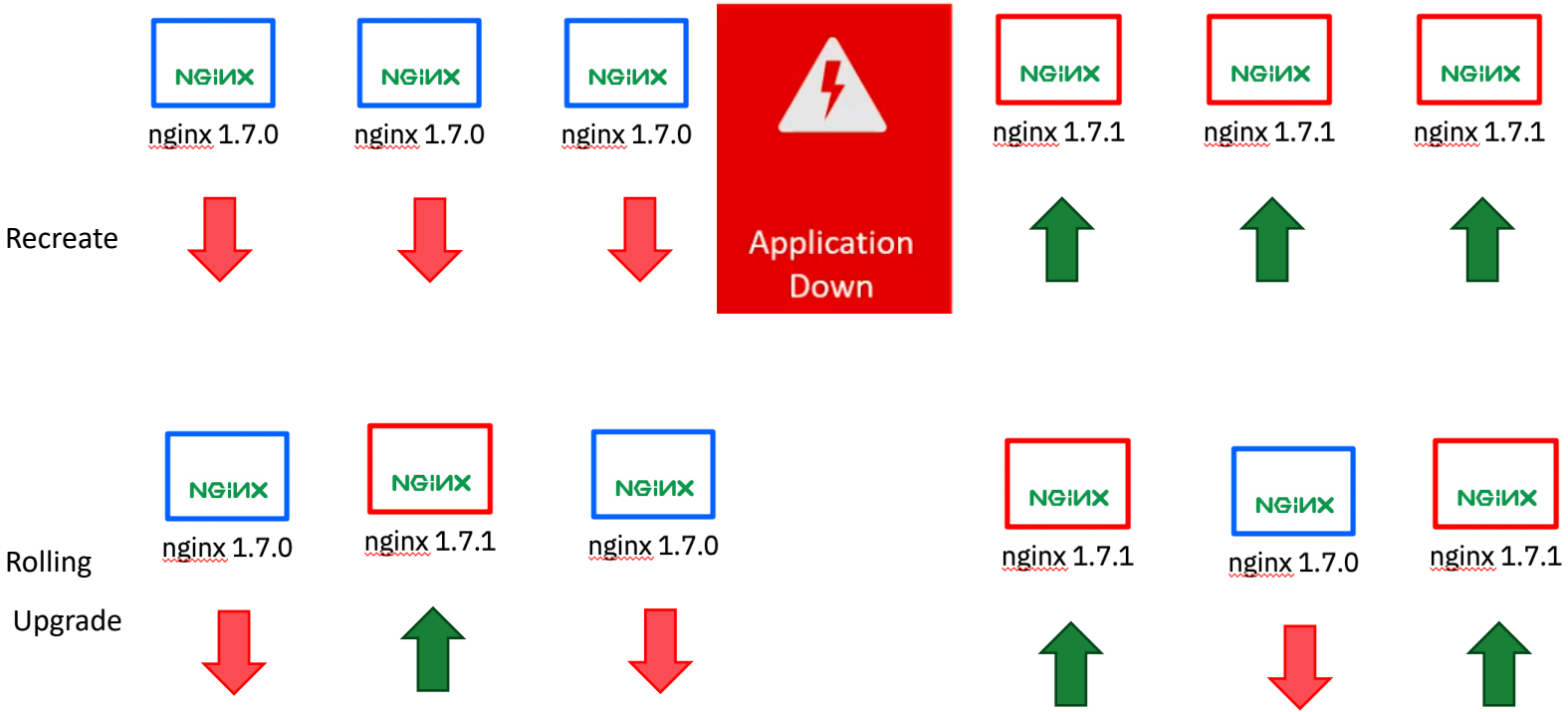
- A way to change or upgrade an application.
- The aim is to make the change without downtime
- In a way that the user barely notices the improvements.

Things to consider - Define Deployment Strategy

1. Long running connections need to be handled gracefully.
2. Database conversions can get tricky and will need to be done and rolled back along with the application.
3. If the application is a hybrid of microservices and traditional components downtime may be needed to complete the transition.
4. You need the infrastructure to do this.
5. If you have a non-isolated test environment, you can break both new and old versions.

Deployment Strategies

Take down application and not want another version



Commands

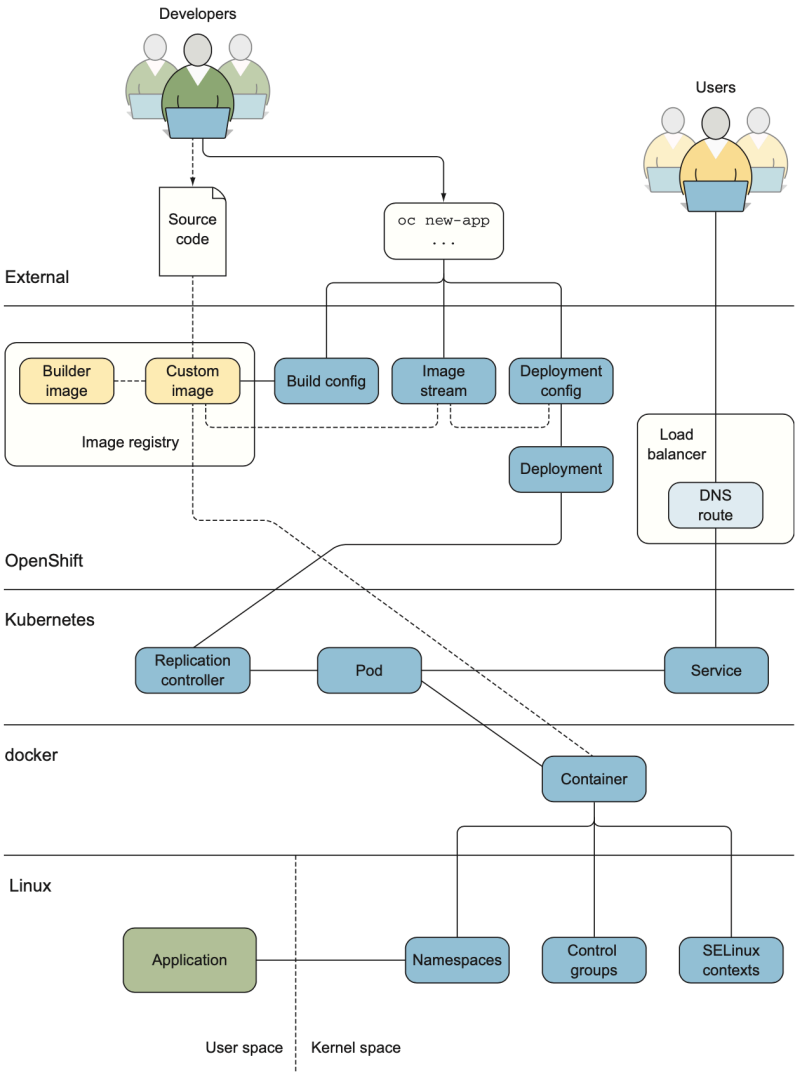
```
> oc rollout latest dc/simple-webapp-docker
```

```
> oc rollout history dc/simple-webapp-docker
```

```
> oc rollout describe dc simple-webapp-docker
```

```
> oc rollout undo dc/simple-webapp-d
```

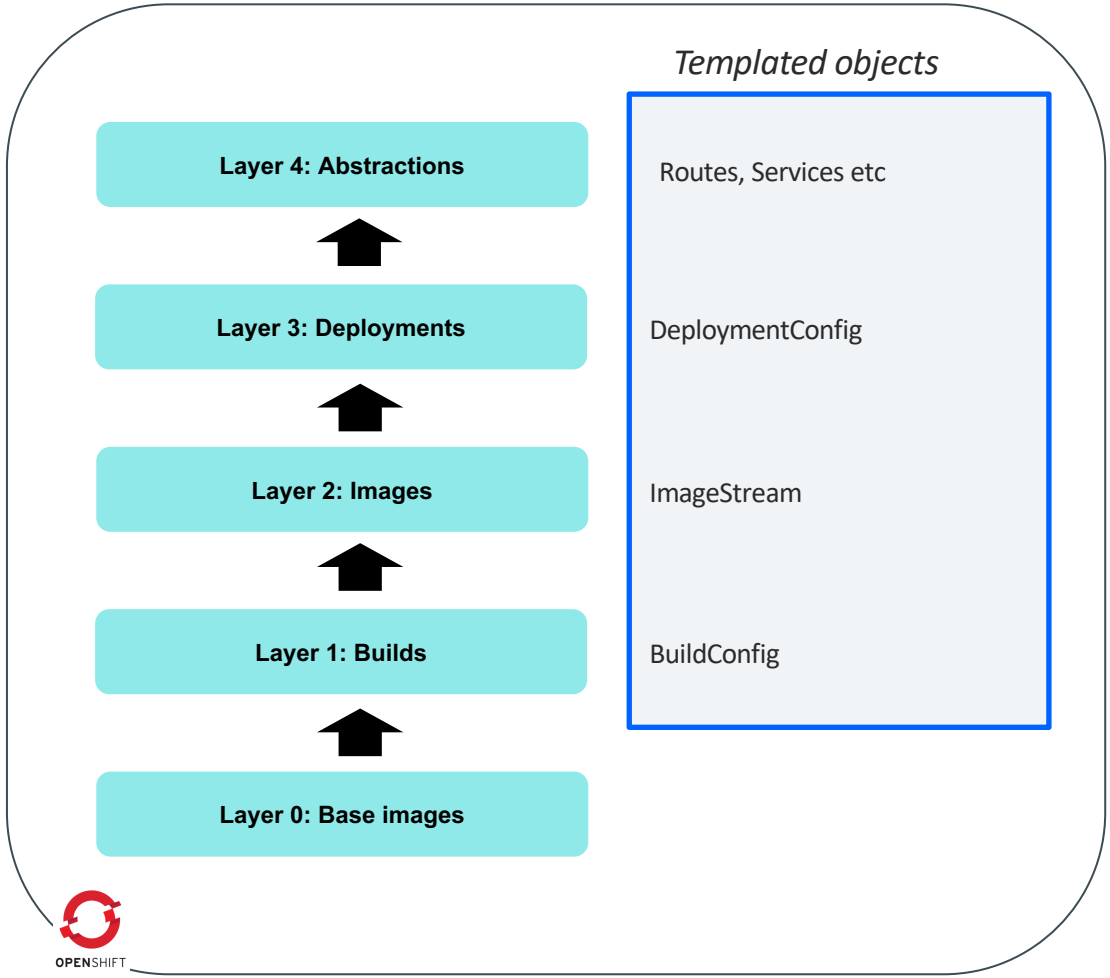
E2E


Developers
Users
Source code
oc new-app ...
External
Builder image
Custom image
Build config
Image stream
Deployment config
Image registry
Deployment
Load balancer
DNS route
OpenShift
Kubernetes
Replication controller
Pod
Service
docker
Container
Linux
Application
Namespaces
Control groups
SELinux contexts
User space
Kernel space

Templates

Templates Overview

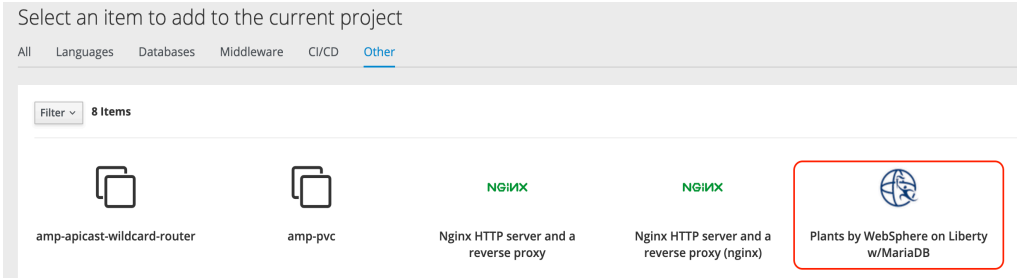
- Templates typically contain parameterized objects for the various layers of the deployment process for an app or service
- Once created they are tightly integrated with the Web console and CLI
- Simplifies the deployment of apps or services that require several objects to be created
- OpenShift includes several templates that can be used OOTB
  - e.g. Jenkins, MariaDB etc



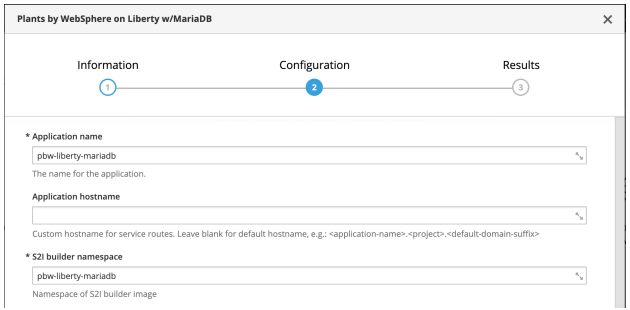
Using a template

From Web console

1 .Select template



2. Apply parameters



From CLI

Apply template and specify parameters by running `oc process` and then piping the result to `oc create`

```
$ oc process -f pbw-liberty \
 -p APPLICATION_HOSTNAME=foobar.com \
  | oc create -f -
```

Thank You