IBM Training
Red Hat OpenShift Application Development

—

Luca Floris
IBM Cloud Technical Consultant

Agenda

Day 2

Develop and deploy apps on OpenShift

- Introduction to OpenShift Application Resources
- Deploying applications on OpenShift
- OpenShift Storage for Applications
- Deploying applications using the user interface
- Source2Image application deployments
- Operators

OpenShift Application Deployment

Deployments

Provides declarative updates for Pods and ReplicaSets.

You describe a *desired state* in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate.

The most simple deployments only require you to specify the number if replicas, selectors/labels, a container name and the image.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Writing a Deployment Spec - Basics

Replicas

Labels and Selectors

Container name and image

Exposed ports

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

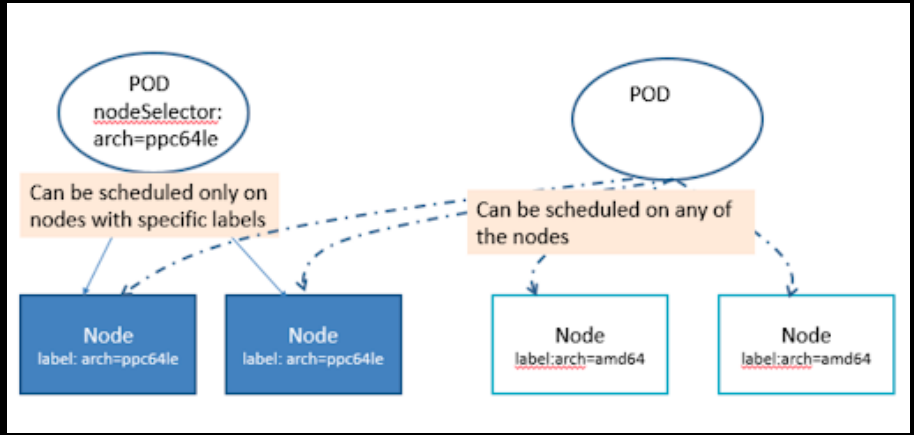
Writing a Deployment Spec - NodeSelector

A NodeSelector instructs the OpenShift scheduler that the pod runs on a specific node

Specifies a key-value pair to match to a node

Node(s) must be labelled

```
nodeSelector:
  node-role.kubernetes.io/worker: ''
```



Writing a Deployment Spec – Readiness/Liveness

Liveness is used by kubelet to know when to **restart** a container

```
livenessProbe:
  httpGet:
    path: /healthz
    port: 1936
    scheme: HTTP
```

Readiness is used by kubelet to know when the container is **ready**

```
readinessProbe:
  httpGet:
    path: /healthz/ready
    port: 1936
    scheme: HTTP
```

Writing a Deployment Spec - Resources

Two types of resource constraints for pods

- Memory (Mi/Gi)
- CPU (m)

```
resources:
  requests:
    cpu: 100m
    memory: 256Mi
  limits:
    cpu: 500m
    memory: 2Gi
```

Writing a Deployment Spec - Volumes

Volumes are mounted to pods at a specified filesystem location, accessible to the containers in a pod

To use a volume, a Pod specifies what volumes to provide for the Pod (the .spec.volumes field) and where to mount those into Containers

Lots of different volume types are available

Extensive list found here https://kubernetes.io/docs/concepts/storage/volumes/#types-of-volumes

```
<container-spec>
...
    volumeMounts:
    - name: default-certificate
      readOnly: true
      mountPath: /etc/pki/tls/private
    - name: metrics-certs
      readOnly: true
      mountPath: /etc/pki/tls/metrics-certs
  volumes:
    - name: default-certificate
      secret:
        secretName: router-certs-default
        defaultMode: 420
    - name: metrics-certs
      secret:
        secretName: router-metrics-certs-default
        defaultMode: 420
```

Writing a Deployment Spec – Taints and Tolerations

Node taints repel pods that do not contain the right tolerations

Pod tolerations allow the pods to be scheduled to nodes matching the node's taints

Use Cases:

Dedicated Nodes – Assign a dedicated set of nodes for a group of users

Nodes with special hardware – Useful for pods that need specialised hardware such as GPU's

```
tolerations:
  - key: node-role.kubernetes.io/master
    operator: Exists
    effect: NoSchedule
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 120
  - key: node.kubernetes.io/not-reachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 120
```

Creating ConfigMaps

Can be created from literal strings, files or whole directories

```
# Get the files
wget https://kubernetes.io/examples/configmap/game.properties -O configmaps/game.properties
wget https://kubernetes.io/examples/configmap/ui.properties -O configmaps/ui.properties

# Create the configmap from the files in the directory
oc create configmap game-config --from-file=configmaps/
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:52:05Z
  name: game-config
  namespace: default
  resourceVersion: "516"
  uid: b4952dc3-d670-11e5-8cd0-68f728db1985
data:
  game.properties: |
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
```

Creating Secrets

```
oc -n default create secret generic mysecret
--from-literal=username=admin --from-literal=password=1f2d1e2e67df
```

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

Creating Services

Services expose application ports defined in the pod spec

```
oc expose deployment mydeploy
```

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Creating Routes

Routes expose services to the outside world through the OpenShift Router

```
oc expose service my-service
```

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: my-route
spec:
  host: my-app.apps.ocp4.os.fyre.ibm.com
  to:
    kind: Service
    name: my-service
    weight: 100
  tls:
    termination: passthrough
    insecureEdgeTerminationPolicy: Redirect
  wildcardPolicy: None
```

Application High Availability

Applications in OpenShift are naturally highly-available!

Need to update an application?

Rolling updates

If a whole node fails?

Move the pods to another node

If a pod fails?

Restart it

Projects

A *project* allows a community of users to organize and manage their content in isolation from other communities.

Can use it to isolate users, groups, applications or entire environments

Developers can create a project themselves when logging in

```
$ oc new-project <project_name>  --description="<description>" --display-name="<display_name>"
```

Demo – Creating a simple OpenShift application

Lab – Creating an OpenShift Application

Visit https://github.com/lfloris/openshift-dev-training/tree/main/Labs for lab materials

Go to Lab 2

Goals

Create and deploy a simple MariaDB application that uses ConfigMaps and Secrets
Create and deploy a simple WebSphere Liberty application that is exposed using a Route

Lab – A more complex WordPress application

Visit https://github.com/lfloris/openshift-dev-training/tree/main/Labs for lab materials

Go to Lab 3

Goals

Develop a more complicated 2 tier OpenShift application with a front end and a back end

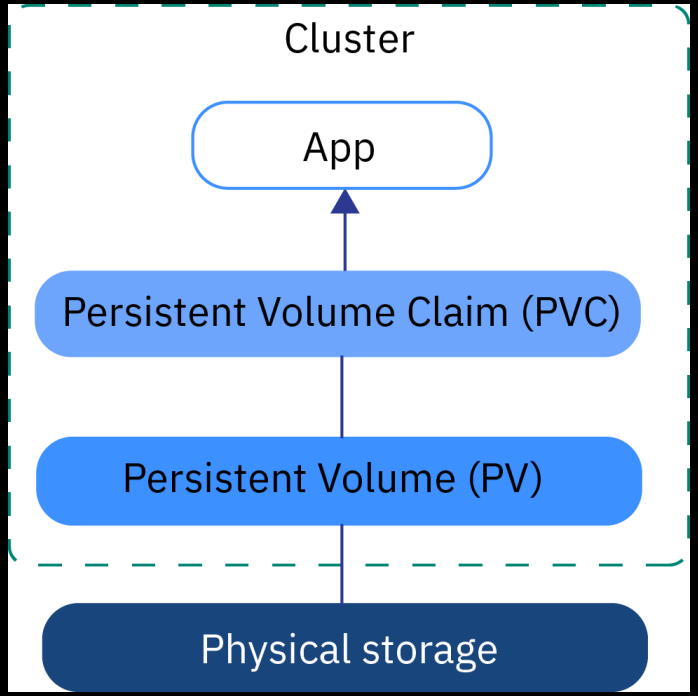
OpenShift Storage

OpenShift leverages Kubernetes Persistent Volumes

Persistent Volume (PV) is a piece of storage, provisioned by an administrator or dynamically provisioned using Storage Classes

Persistent Volume Claim (PVC) is a claim for that storage by a user

Storage Classes (SC) allow allocating storage technologies and dynamic provisioning



OpenShift Storage

Visit https://github.com/lfloris/openshift-dev-training/tree/main/Labs for lab materials

Go to Lab 4 & 5

Goals

Create a new Persistent Volume and Persistent Volume Claim, then deploy an application using this claim

Create a new Persistent Volume Claim using a Storage Class, then create an application that uses the claim

Creating New Applications - Web UI

OpenShift Application Deployments

Many ways to deploy applications

- Direct YAML
- Web UI
- CLI
- DeploymentConfigs
- Templates
- Pipelines
- Operators
- Source2Image

From Git

Import code from your git repository to be built and deployed

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 your Dockerfile from your git repo to be built & deployed

YAML

Create resources from their YAML or JSON definitions

Database

Browse the catalog to discover database services to add to your application

DeploymentConfigs

Deployments and DeploymentConfigs in OpenShift Container Platform are API objects that provide two similar but different methods for fine-grained management over common user applications.

The DeploymentConfig deployment system provides the following capabilities:

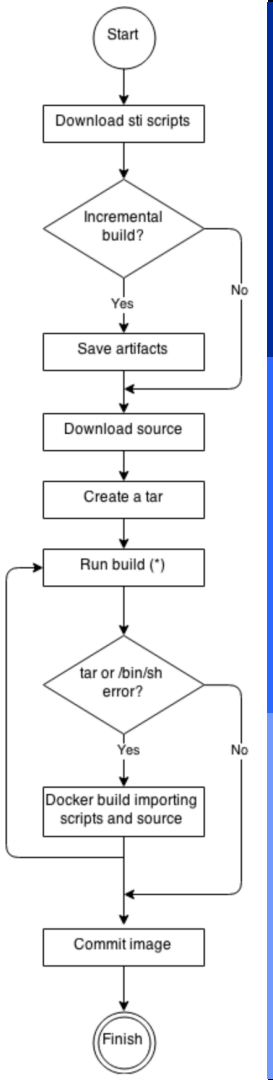
- A DeploymentConfig, which is a template for running applications.
- Triggers that drive automated deployments in response to events.
- User-customizable deployment strategies to transition from the previous version to the new version. A strategy runs inside a Pod commonly referred as the deployment process.
- A set of hooks (lifecycle hooks) for executing custom behavior in different points during the lifecycle of a deployment.
- Versioning of your application in order to support rollbacks either manually or automatically in case of deployment failure.
- Manual replication scaling and autoscaling

Source2Image

Source-to-Image (S2I) is a tool for building reproducible, Docker-formatted container images.

The advantages of S2I include the following:

- **Image flexibility**
- **Speed**
- **Patchability**
- **Operational efficiency**
- **Operational security**
- **User efficiency**
- **Ecosystem**
- **Reproducibility**



Develop

Build

Run

ImageStreams

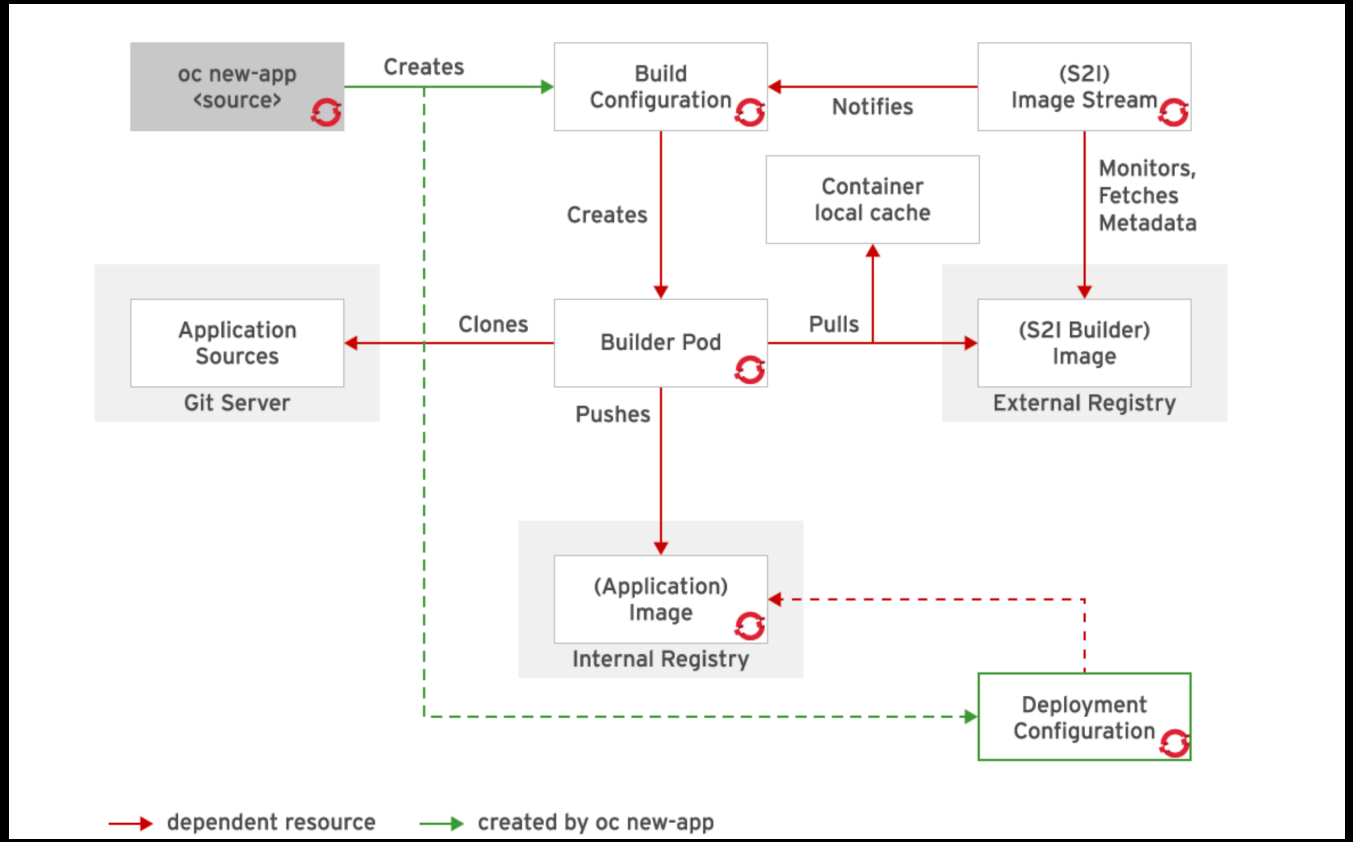
ImageStreams provide a way for applications to automatically roll out updates when an image changes

OpenShift detects when an image stream changes and takes action based on that change.

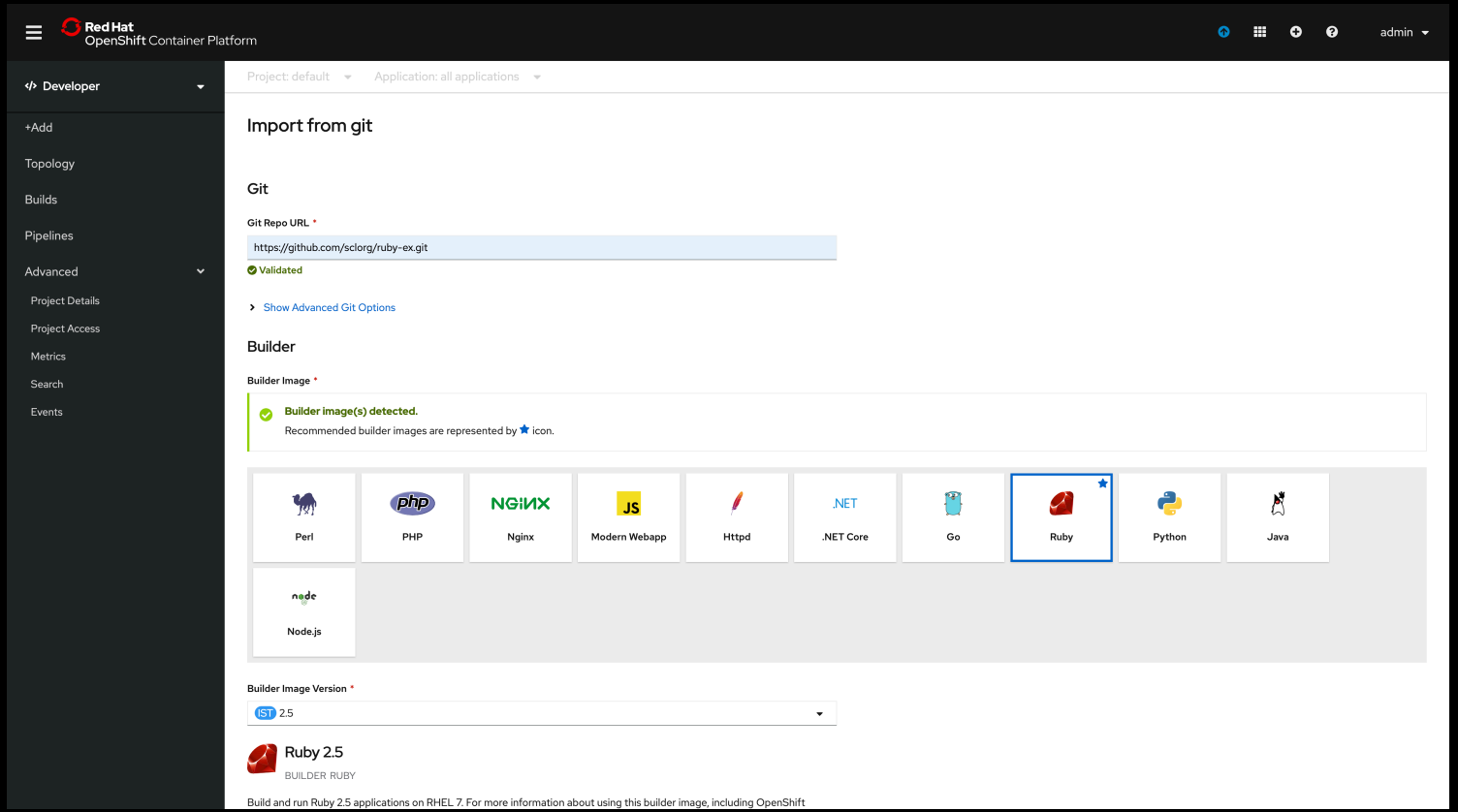
The *image stream resource* is a configuration that names specific container images associated with *image stream tags*

You can configure Builds and Deployments to watch an imagestream for notifications when new images are added and react by performing a Build or Deployment

Source2Image Flow



Building from GitHub



Lab – WebUI Deployments

Visit https://github.com/lfloris/openshift-dev-training/tree/main/Labs for lab materials

Go to Lab 6

Goals

Deploy an application using an application template from the catalog

Source2Image Lab

Visit https://github.com/lfloris/openshift-dev-training/tree/main/Labs for lab materials

Go to Lab 7

Goals

Deploy a new application using Source2Image Git from the UI

Deploy a new application using Source2Image Git from CLI

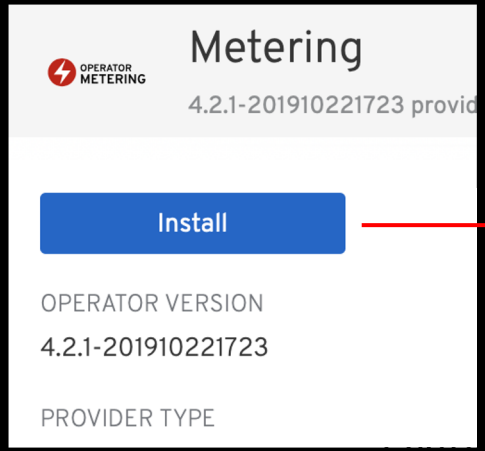
Operators

Why use Operators?

- ✓ Repeatability of installation and upgrade.
- ✓ Constant health checks of every system component.
- ✓ Over-the-air (OTA) updates for OpenShift components and ISV content.

Operator Framework

Singleton CRs & Auto-create CRs from single click



Single click

1. Install into a specific namespace from CSV
2. Automatically create an Operand instance
3. Hooks into OpenShift Console are installed/configured
    a. If RH product, navigation shows up
    b. Configure custom dashboards
    c. Configure external links and banners
    d. Register new CLIs in the downloads area

Useful for: Serverless, Metering, Service Mesh, Pipelines, Logging, Container Storage & more

Operator Framework

CSV + Subscription + InstallPlan → single Operator object

```
apiVersion: operatorframework.io/v1alpha1
kind: Operator
metadata:
  ...
```

Split CSV into new bundle format →

Kubernetes objects:
Deployment/StatefulSet, Roles, RoleBindings, custom SCCs

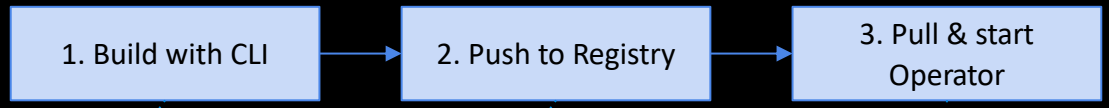
Metadata:
icon, channels, related images,CR examples,

1. Unlocks ability to install specific version (not latest)
2. Directly install Operator outside of OperatorHub
    a. bypass catalogs, OperatorGroups, etc
3. Easier onboarding and building of Operator releases

Operator Framework

New Operator Bundle Format

Streamlined developer UX for getting an Operator running without hassle of a central catalog




```
$ operator-sdk bundle init --type=registry --
bundle-folder=0.1.0

$ tree test
test
├── 0.1.0
│   ├── testbackup.crd.yaml
│   ├── testcluster.crd.yaml
│   ├── testoperator.v0.1.0.csv.yaml
│   └── testrestore.crd.yaml

$ podman build .
$ podman push quay.io/test/test-operator:v0.1.0
```

```
$ kubectl apply -f -
apiVersion:
operators.operatorframework.io/v2alpha1
kind: Operator
metadata:
  name: test-operator
spec:
  bundle:
    image:
      quay.io/test/test-operator:v0.1.0
```

Working with kubebuilder & others upstream to standardize this format.

Certified/Community catalogs will also use this format.

Operator Lab

Visit https://github.com/lfloris/openshift-dev-training/tree/main/Labs for lab materials

Go to Lab 8

Goals

Deploy an Operator from OperatorHub using the Web UI

Deploy an Operator from OperatorHub using the CLI

Questions/Discussions?