AWB Bootcamp
Red Hat OpenShift and IBM Cloud Paks

—

Luca Floris
IBM Cloud Technical Consultant

Agenda

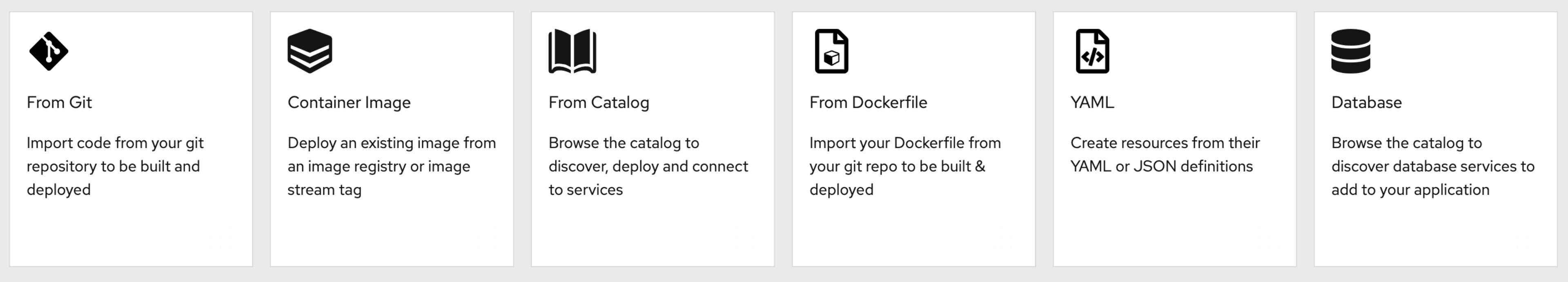
Day 3 (Application Development)

- OpenShift Application Deployments
- Helm Deployments in OpenShift
- OpenShift Security Context Constraints
- Limiting Resources with Quotas

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

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-bootcamp/tree/master 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-bootcamp/tree/master for lab materials

Go to Lab 3

Goals

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

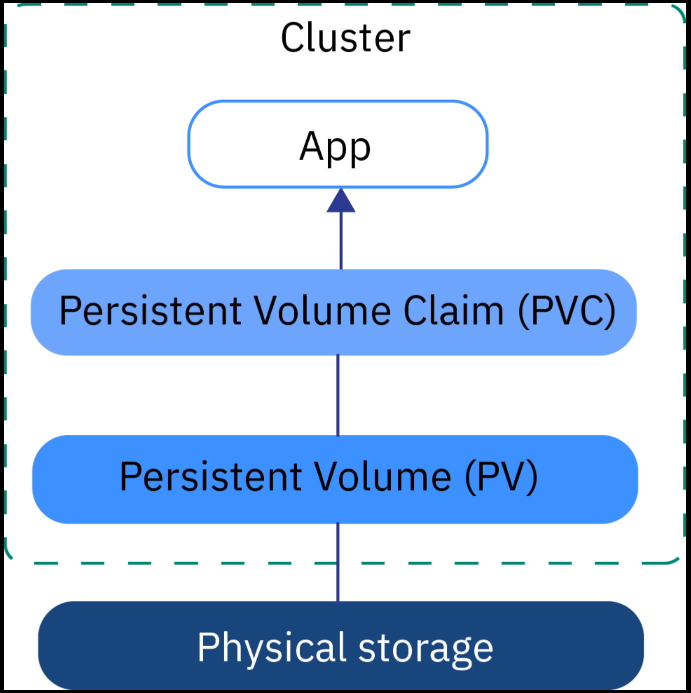
OpenShift Storage

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-bootcamp/tree/master 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

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:

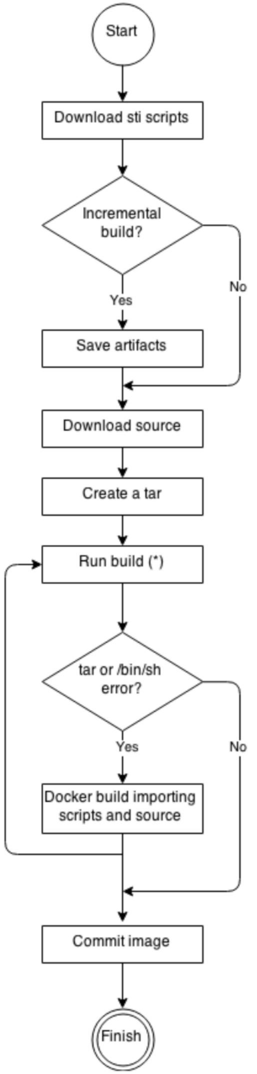
- A DeploymentConfig, which is a template for running applications.
- Triggers that drive automated deployments in response to events.
- User-customizable deployment strategies to transition from the previous version to the new version. A strategy runs inside a Pod commonly referred as the deployment 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

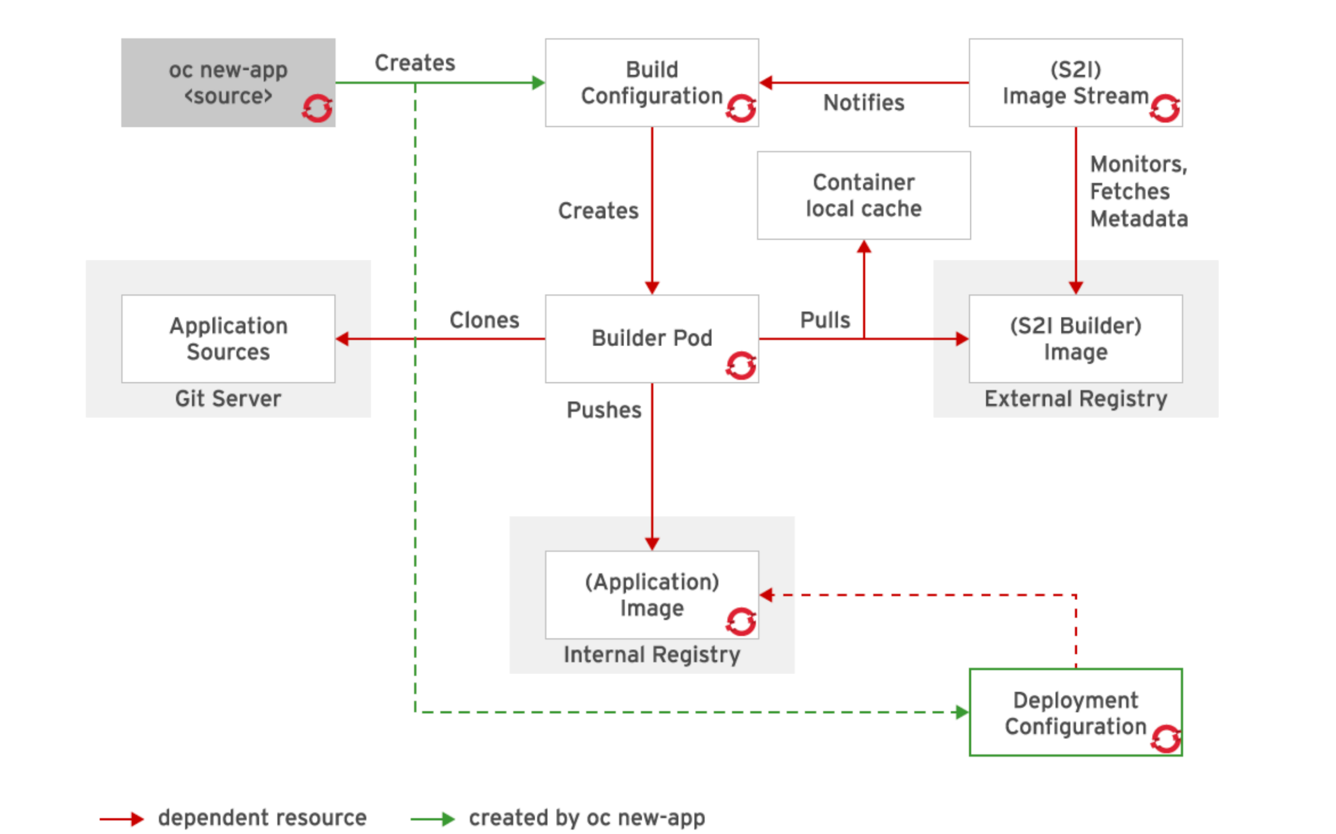
ImageStreams provide a way for applications to automatically roll out updates when an image changes

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

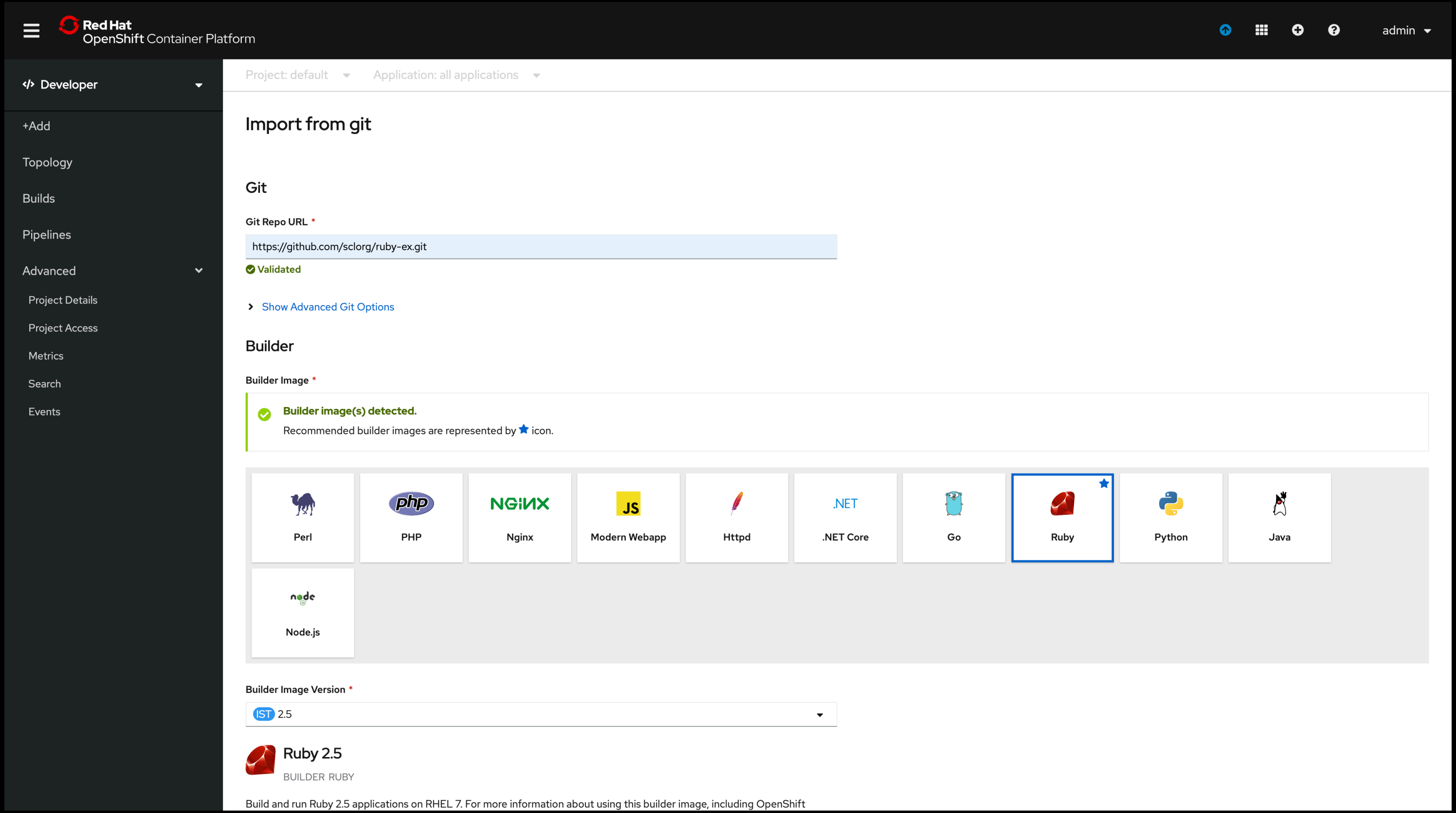
The *image stream resource* is a configuration that names specific container images associated with *image stream tags*

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



Import from git

Git

Git Repo URL *

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

✓ Validated

Builder

Builder Image *

Builder image(s) detected.

Recommended builder images are represented by ★ icon.

Builder Image Version *

IST 2.5

Ruby 2.5

BUILDER RUBY

Build and run Ruby 2.5 applications on RHEL 7. For more information about using this builder image, including OpenShift

Lab – WebUI Deployments

Visit https://github.com/lfloris/openshift-bootcamp/tree/master
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-bootcamp/tree/master 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

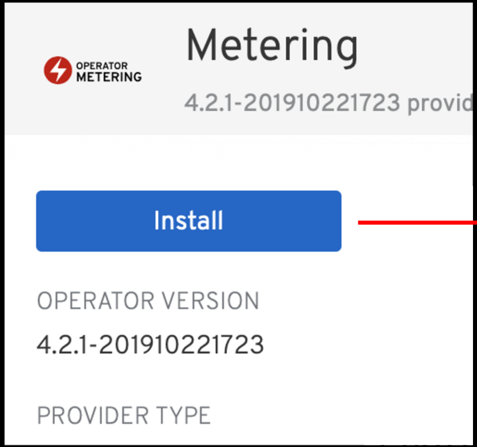
Operators

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



OPERATOR METERING

Metering

4.2.1-201910221723 provid

Install

OPERATOR VERSION

4.2.1-201910221723

PROVIDER TYPE

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 <u>Operator object</u>

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

Split CSV into new <u>bundle format</u>

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

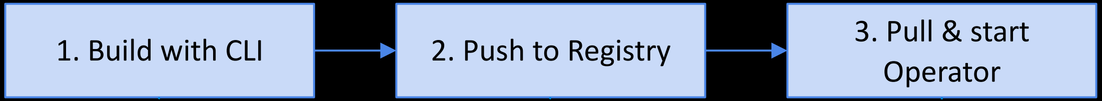
Metadata:
icon, channels, related images,CR examples,

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.

Helm 3

What is Helm 3?

a package manager for Kubernetes applications

- Fetch software packages
- Install software
- Install software dependencies
- Configure deployments
- Update deployments

Helm 3 Architecture

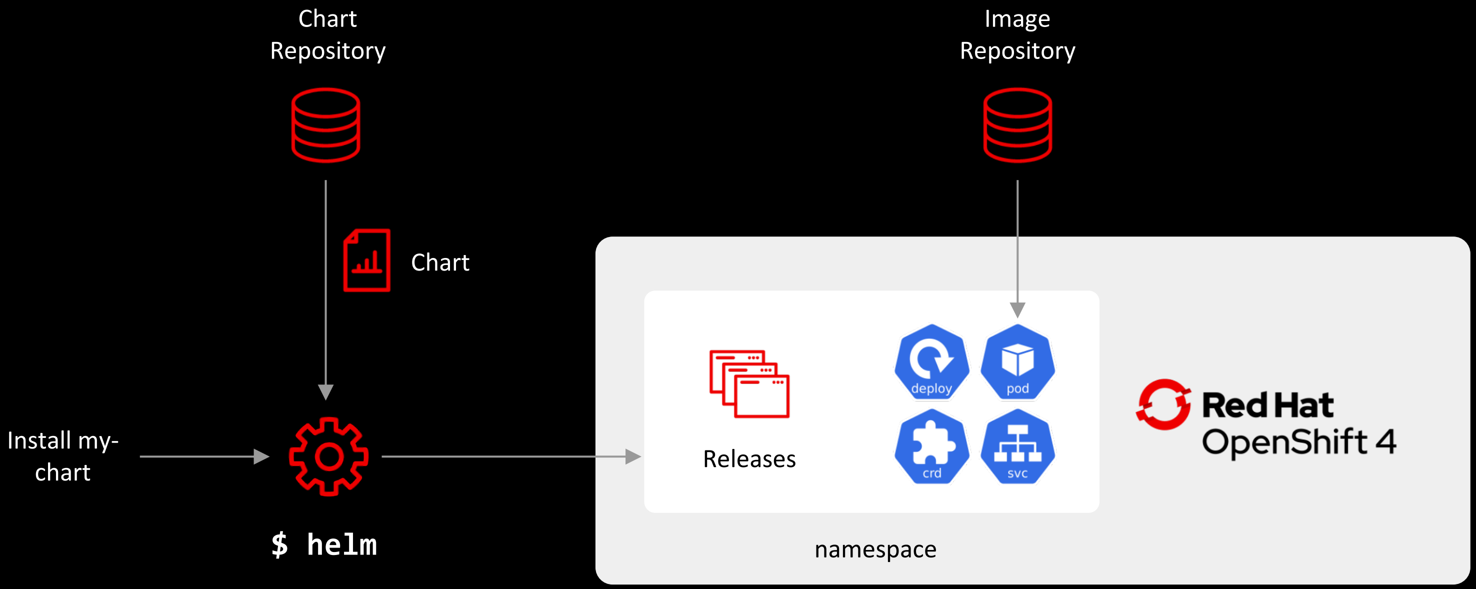


Chart Repository

Image Repository

Chart

Install my-chart

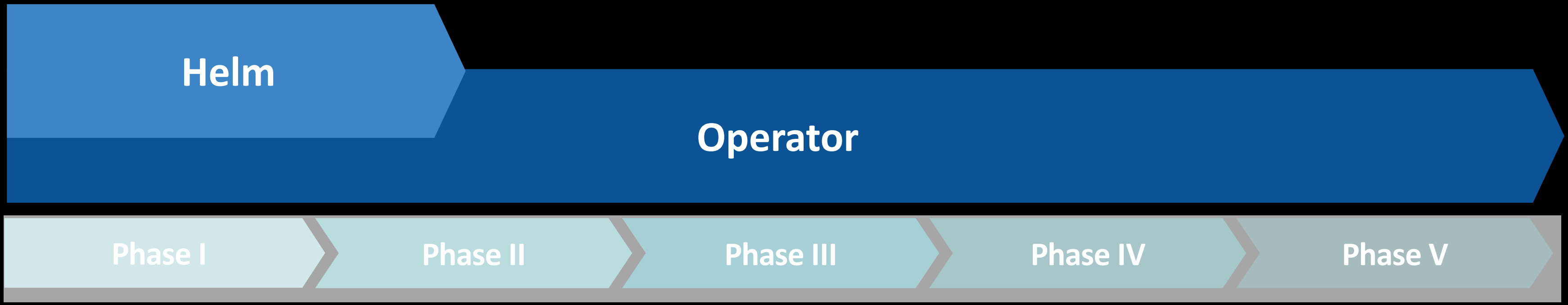
$ helm

Releases

namespace

Helm and Operators

Package and Install



Helm

Operator

Phase I	Basic Install	Automated application provisioning and configuration management
Phase II	Seamless Upgrades	Patch and minor version upgrades supported
Phase III	Full Lifecycle	App lifecycle, storage lifecycle (backup, failure recovery)
Phase IV	Deep Insights	Metrics, alerts, log processing and workload analysis
Phase V	Auto Pilot	Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning

Developing Helm Charts

Getting started

```
$ curl -L https://mirror.openshift.com/pub/openshift-v4/clients/helm/latest/helm-linux-amd64 -o
/usr/local/bin/helm3

$ chmod +x /usr/local/bin/helm3

$ helm list
NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION
```

You’re ready to start developing!

Helm Basics

All Helm resources, when packaged, run through a Go templating engine which dynamically generates values based on the template code evaluation. Using this templating language each deployment can be customised with a unique set of values for the release.

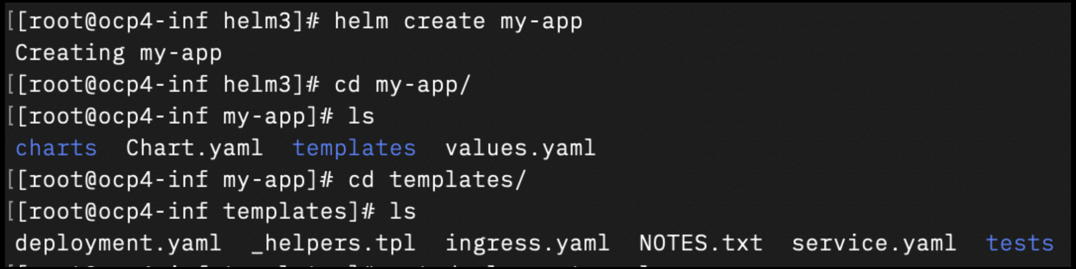
Helm templating looks like the following

```
beverage:
  {{- if eq .Values.favorite.drink "coffee"}}
  mug: true
  {{- else }}
  glass: true
  {{- end}}
```

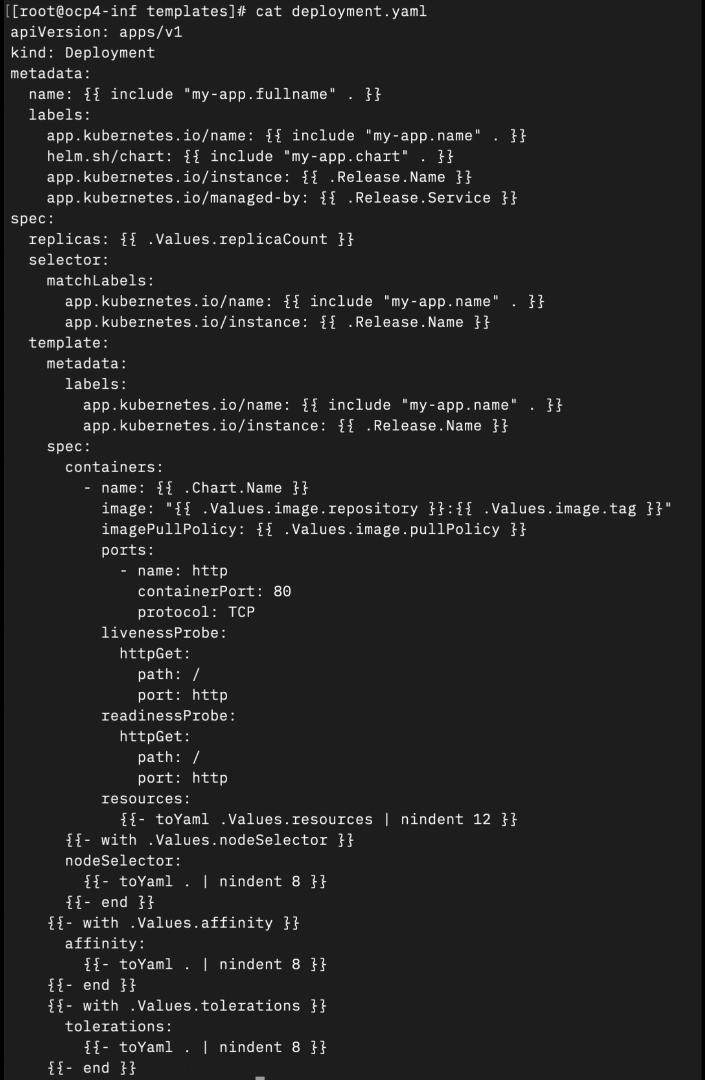
Helm Basics

Start with a simple `helm create my-python`

Helm will generate a base for you to start with



```
[[root@ocp4-inf helm3]# helm create my-app
 Creating my-app
[[root@ocp4-inf helm3]# cd my-app/
[[root@ocp4-inf my-app]# ls
 charts  Chart.yaml  templates  values.yaml
[[root@ocp4-inf my-app]# cd templates/
[[root@ocp4-inf templates]# ls
 deployment.yaml  _helpers.tpl  ingress.yaml  NOTES.txt  service.yaml  tests
```



```
[[root@ocp4-inf templates]# cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "my-app.fullname" . }}
  labels:
    app.kubernetes.io/name: {{ include "my-app.name" . }}
    helm.sh/chart: {{ include "my-app.chart" . }}
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: {{ .Release.Service }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app.kubernetes.io/name: {{ include "my-app.name" . }}
      app.kubernetes.io/instance: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app.kubernetes.io/name: {{ include "my-app.name" . }}
        app.kubernetes.io/instance: {{ .Release.Name }}
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
          livenessProbe:
            httpGet:
              path: /
              port: http
          readinessProbe:
            httpGet:
              path: /
              port: http
          resources:
            {{- toYaml .Values.resources | nindent 12 }}
      {{- with .Values.nodeSelector }}
      nodeSelector:
        {{- toYaml . | nindent 8 }}
      {{- end }}
    {{- with .Values.affinity }}
      affinity:
        {{- toYaml . | nindent 8 }}
    {{- end }}
    {{- with .Values.tolerations }}
      tolerations:
        {{- toYaml . | nindent 8 }}
    {{- end }}
```

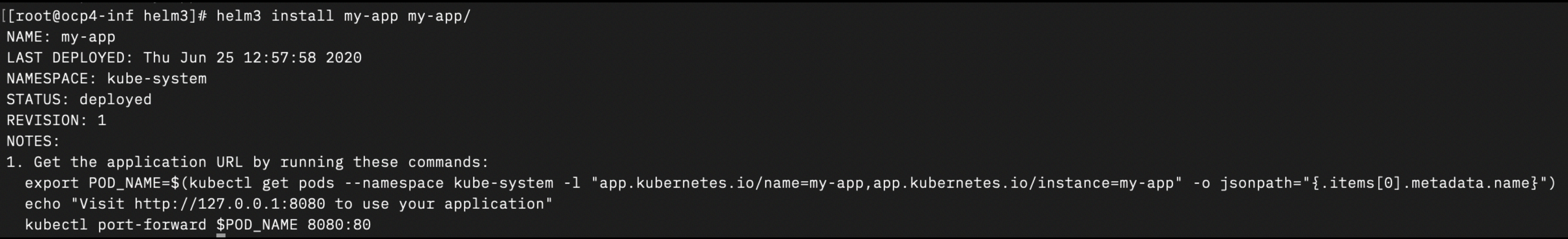
Helm Basics

You supply the chart values using values.yaml

The values.yaml is what allows you to keep reusing the same Helm chart with different values for different installations

Edit the values.yaml, then deploy from the top level

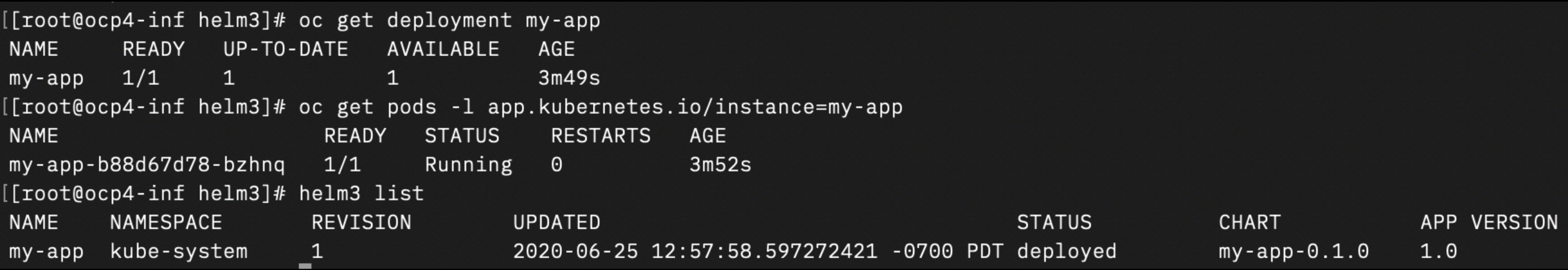
```
helm install my-app my-app/
```



```
[[root@ocp4-inf helm3]# helm3 install my-app my-app/
 NAME: my-app
 LAST DEPLOYED: Thu Jun 25 12:57:58 2020
 NAMESPACE: kube-system
 STATUS: deployed
 REVISION: 1
 NOTES:
 1. Get the application URL by running these commands:
   export POD_NAME=$(kubectl get pods --namespace kube-system -l "app.kubernetes.io/name=my-app,app.kubernetes.io/instance=my-app" -o jsonpath="{.items[0].metadata.name}")
   echo "Visit http://127.0.0.1:8080 to use your application"
   kubectl port-forward $POD_NAME 8080:80
```

Helm Basics

The application is running and shows up in helm list



```
[[root@ocp4-inf helm3]# oc get deployment my-app
 NAME     READY   UP-TO-DATE   AVAILABLE   AGE
 my-app   1/1     1            1           3m49s
[[root@ocp4-inf helm3]# oc get pods -l app.kubernetes.io/instance=my-app
 NAME                     READY   STATUS    RESTARTS   AGE
 my-app-b88d67d78-bzhnq   1/1     Running   0          3m52s
[[root@ocp4-inf helm3]# helm3 list
 NAME    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
 my-app  kube-system     1               2020-06-25 12:57:58.597272421 -0700 PDT deployed        my-app-0.1.0    1.0
```

Helm Lab

Visit https://github.com/lfloris/openshift-bootcamp/tree/master for lab materials

Go to Lab 9

Create an application from a Helm chart and deploy it to OpenShift

Security Context Constraints

Security Context Constraints

SCCs allow an administrator to control:

- Whether a pod can run privileged containers.
- The capabilities that a container can request.
- The use of host directories as volumes.
- The SELinux context of the container.
- The container user ID.
- The use of host namespaces and networking.
- The allocation of an FSGroup that owns the pod's volumes.
- The configuration of allowable supplemental groups.
- Whether a container requires the use of a read only root file system.
- The usage of volume types.
- The configuration of allowable seccomp profiles.

Demo - Security Context Constraints

Lab - SCCs

Visit https://github.com/lfloris/openshift-bootcamp/tree/master for lab materials

Go to Lab 9

Goals

Apply a Security Context Constraint to a project, then create an application to test

Resource Limits

A *resource quota*, defined by a ResourceQuota object, provides constraints that limit aggregate resource consumption per project.

We use Resource quotas to enforce control over projects and what they consume

We can set limits on

1. Resouces (cpu/memory)
2. Storage
3. Specific object counts

Quota enforcement

1. After a resource quota is first created, the project restricts the ability to create any new resources that may violate a quota constraint until it has calculated updated usage statistics.
2. After a quota is created and usage statistics are updated, the project accepts the creation of new content.
3. When you delete a resource, your quota use is decremented during the next full recalculation of quota statistics
4. If project modifications exceed a quota usage limit, the server denies the action, and an error is returned to the user

Demo – Resource Limits

Lab – Resource Limits

Visit https://github.com/lfloris/openshift-bootcamp/tree/master for lab materials

Go to Lab 11

Goals

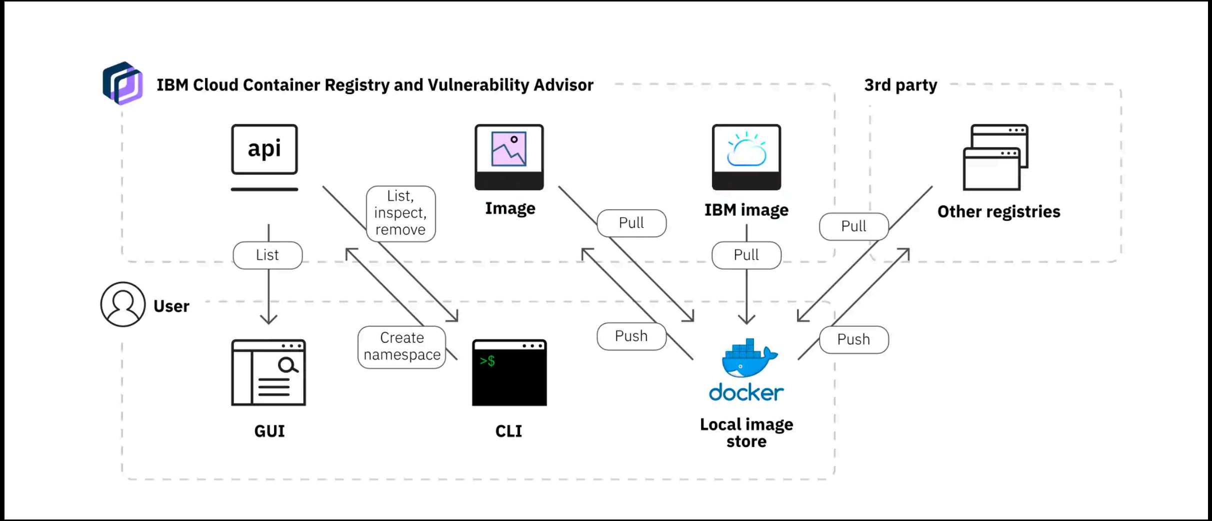
Apply limits to a project and create an application to test

Image Registries

A place to store, push and pull images

Usually has some form of authentication, unless it’s public

Huge ecosystem of open source images and applications to consume



OpenShift Image Registry

OpenShift provides a built in image registry

Managed by the Image Registry Operator

Integrates with OpenShift authentication and authorization

Can be scaled up or down to meet requirements

OpenShift Image Registry
- Pushing and Pulling Images

Get the route typically - `default-route-openshift-image-registry.apps.mydomain.com`

```
$ HOST=$(oc get route default-route -n openshift-image-registry --template='{{
.spec.host }}’)
```

Log in with podman

```
$ podman login -u $(oc whoami) -p $(oc whoami -t) --tls-verify=false $HOST
Login succeeded!
```

OpenShift Image Registry
- Pushing and Pulling Images

Pull an image from docker Hub

```
$ podman pull python:slim-2.7
```

Retag it with the private repository name

```
$ podman tag python:slim-2.7  ${HOST}/myproject/my-python:slim-2.7
```

Try pushing it from the private repository

```
$ podman push ${HOST}/myproject/my-python:slim-2.7
```

Try pulling it from the private repository

```
$ podman pull ${HOST}/myproject/my-python:slim-2.7
```

This image can now be reference by deployments!

Lab - OpenShift Image Registry

Visit https://github.com/lfloris/openshift-bootcamp/tree/master for lab materials

Go to Lab 12

Goals

Push an image from the internet or you local machine to the OpenShift Image Registry

Deploy a new application using this image

Troubleshooting OpenShift Applications

Why didn’t my pod start up?

InsufficentResources
ImagePullBackOff
CrashLoopBackOff
ReadinessProbeFailed/LivenessProbeFailed

Pod Logs?

What happens when a pod is deleted?

Questions/Discussions?