IBM Training
Red Hat OpenShift Application Development

Luca Floris
IBM Cloud Technical Consultant

Agenda

Day 3

Develop and deploy apps on OpenShift continued…

- Introduction to Helm
- OpenShift Security Context Constraints
- Limiting resources with Quotas
- Restricting workload resource usage with quotas
- Image Registries
- OpenShift Image Registry
- Scaling Applications

Helm 3 Applications

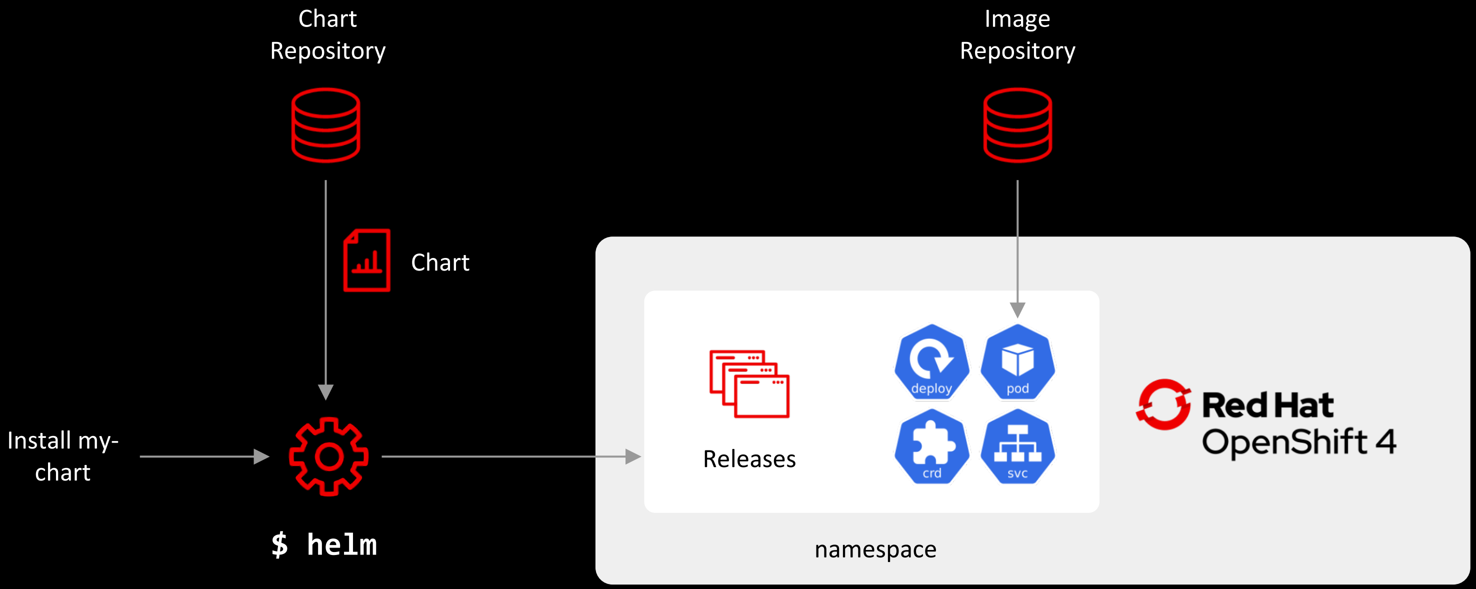
Helm 3

What is Helm 3?

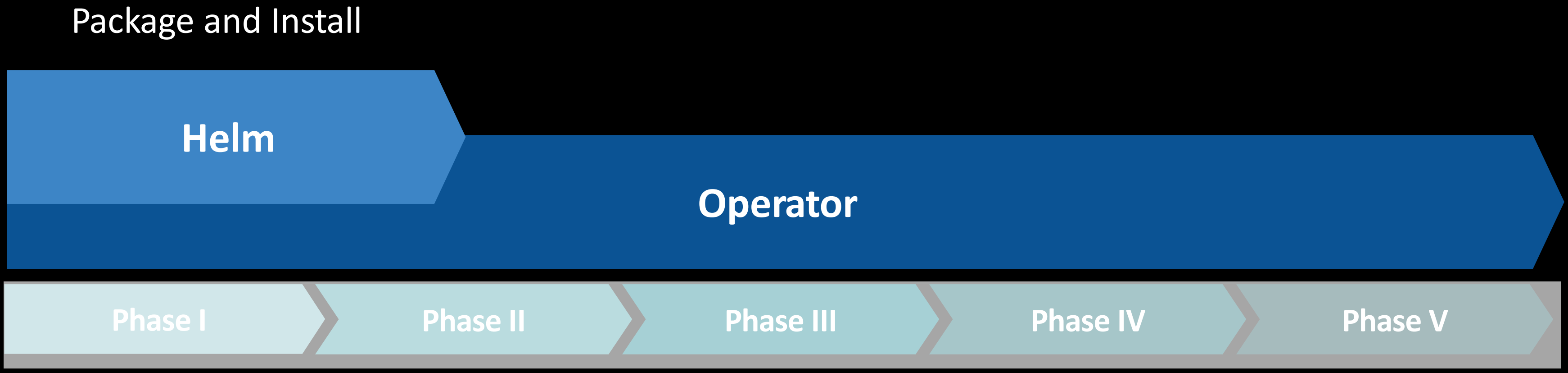
A package manager for Kubernetes applications

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

Helm 3 Architecture



Helm and Operators



Basic Install	Seamless Upgrades	Full Lifecycle	Deep Insights	Auto Pilot
Automated application provisioning and configuration management	Patch and minor version upgrades supported	App lifecycle, storage lifecycle (backup, failure recovery)	Metrics, alerts, log processing and workload analysis	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.

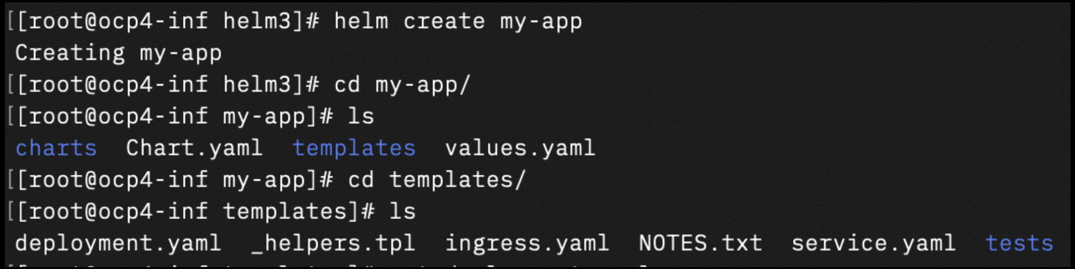
Helm templating looks like the following

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

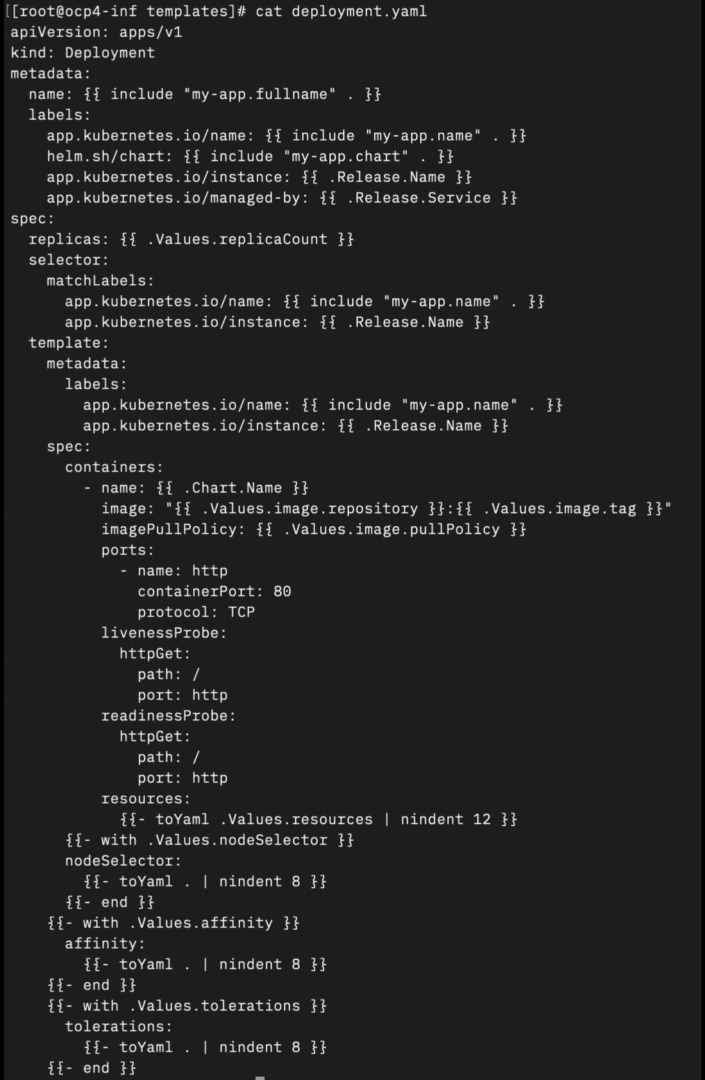
Helm Basics

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 }}
```

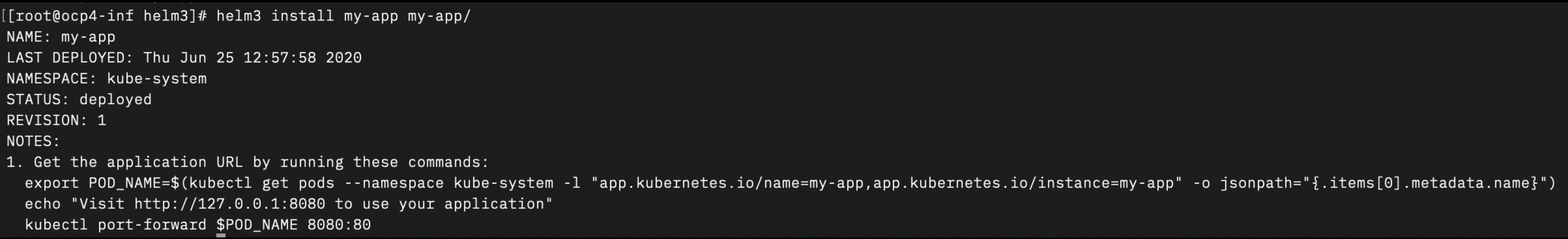
Helm Basics

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

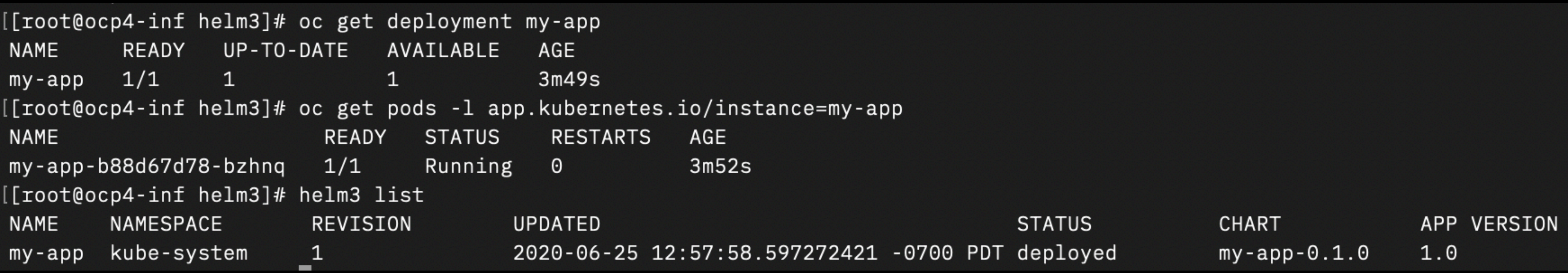
```
helm install my-app my-app/
```



```
[[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-dev-training/tree/main/Labs for lab materials

Go to Lab 9

Goals

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-dev-training/tree/main/Labs for lab materials

Go to Lab 9

Goals

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

Project Resource Quotas

Resource Quotas

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 Quotas

Lab – Resource Quotas

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

Go to Lab 11

Goals

Apply limits to a project and create an application to test

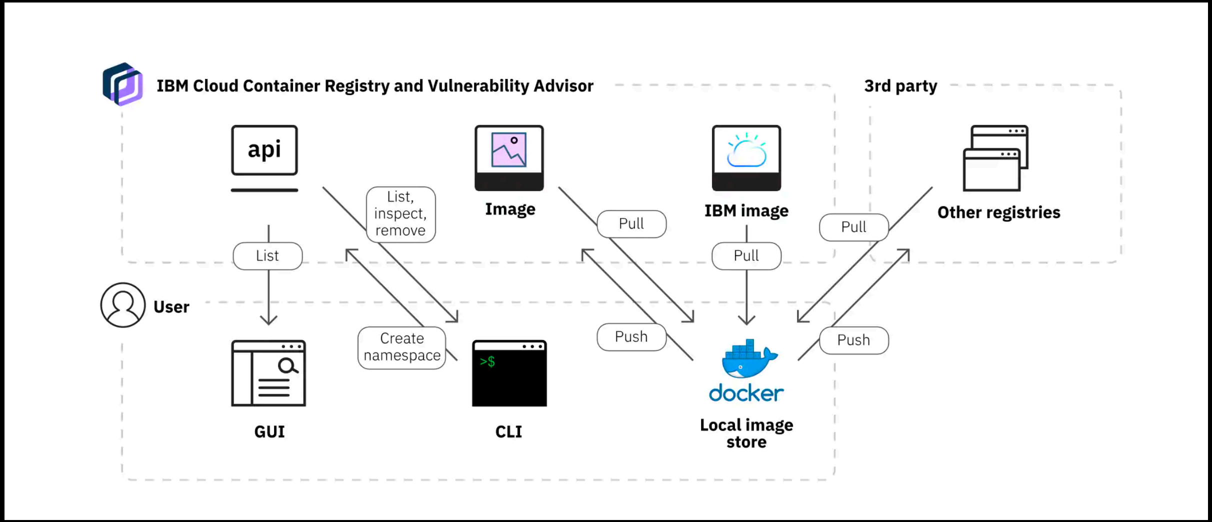
Image Registries

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

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

Retag it with the private repository name

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

Try pushing it from the private repository

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

Try pulling it from the private repository

```
$ docker 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-dev-training/tree/main/Labs 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

Questions/Discussions?