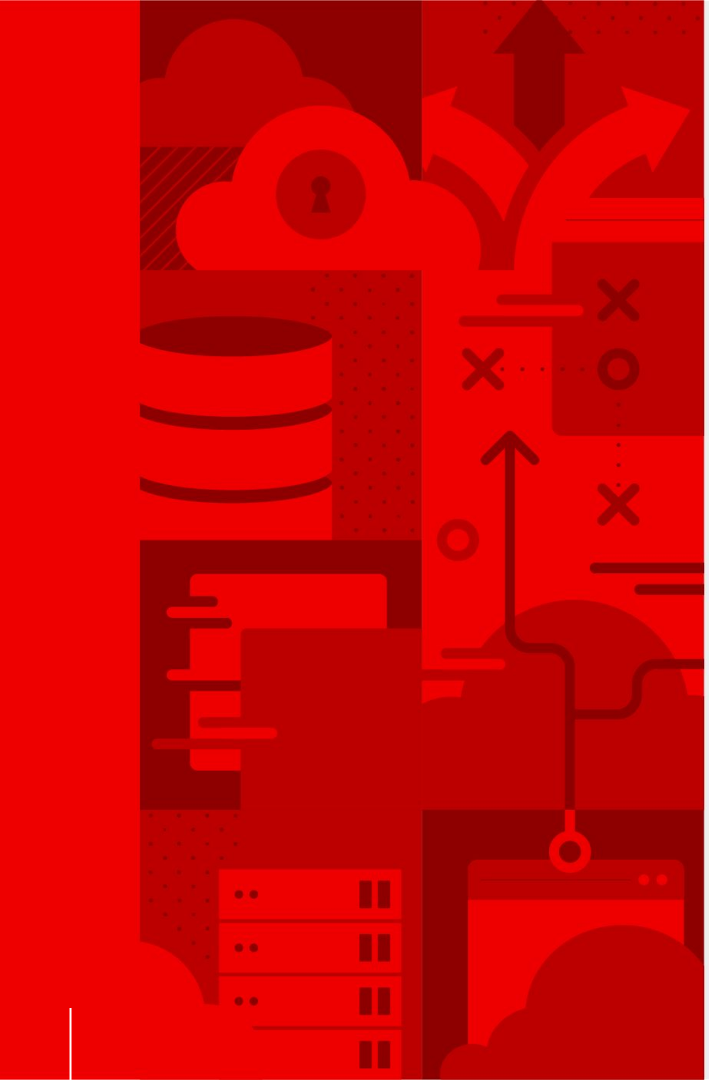


GENERAL DISTRIBUTION

Kubernetes Native Developer

Architecture Workshop

Application Packaging

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat

Self introduction

Name: Wanja Pernath

Email: wpernath@redhat.com

Base: Germany (very close to the Alps)

Role: EMEA Technical Partner Development Manager

- OpenShift and MW

Experience: Years of Consulting, Training, PreSales at

Red Hat and before

Twitter: https://twitter.com/wpernath

LinkedIn:https://www.linkedin.com/in/wanjapernath/

GitHub: https://github.com/wpernath

First book just published

Getting GitOps

A technical blueprint for developing with Kubernetes and OpenShift based on a REST microservice example written with Quarkus

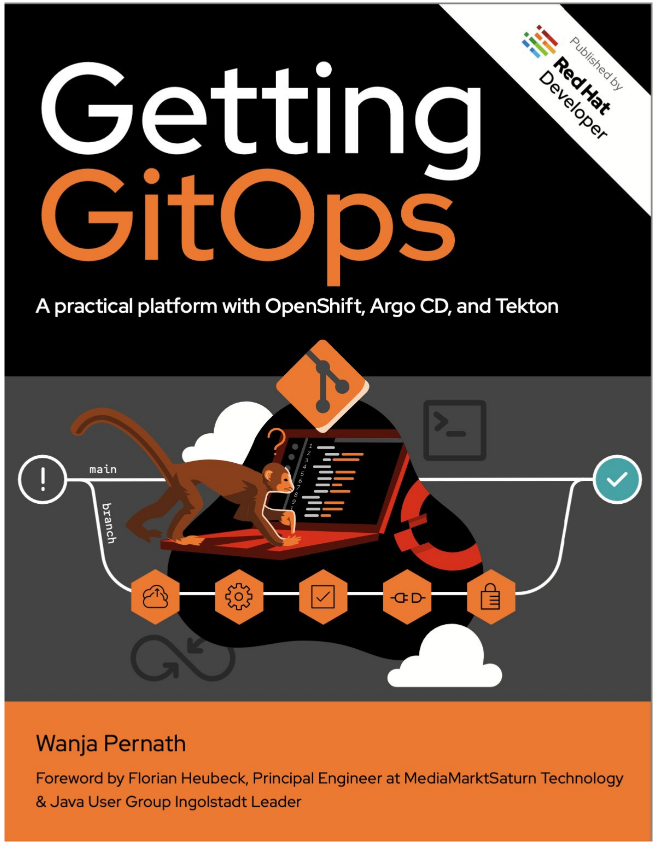
Technologies discussed:
Quarkus, Helm Charts, Kustomize, Tekton Pipelines, Kubernetes Operators, OpenShift Templates, ArgoCD, CI/CD, GitOps….

Download for free at:

https://developers.redhat.com/e-books/getting-gitops-practical-platform-openshift-argo-cd-and-tekton

Interview with full GitOps Demo:

https://www.youtube.com/watch?v=znMfVqAIRzY&ab_channel=OpenShift


Published by
Red Hat Developer
Getting GitOps
A practical platform with OpenShift, Argo CD, and Tekton
main
branch
Wanja Pernath
Foreword by Florian Heubeck, Principal Engineer at MediaMarktSaturn Technology & Java User Group Ingolstadt Leader

Agenda / etc.

Agenda

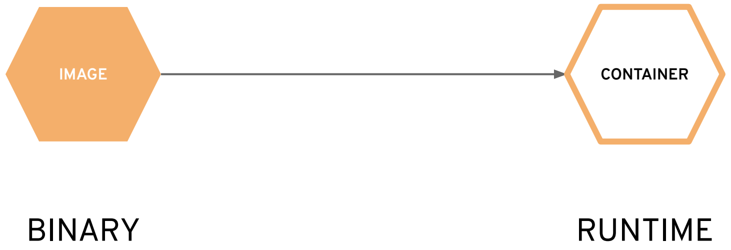
- Application Packaging with OpenShift
  - Basics
  - kustomize.io
  - Helm Charts
  - Summary
- Demo

OpenShift Developers Basics

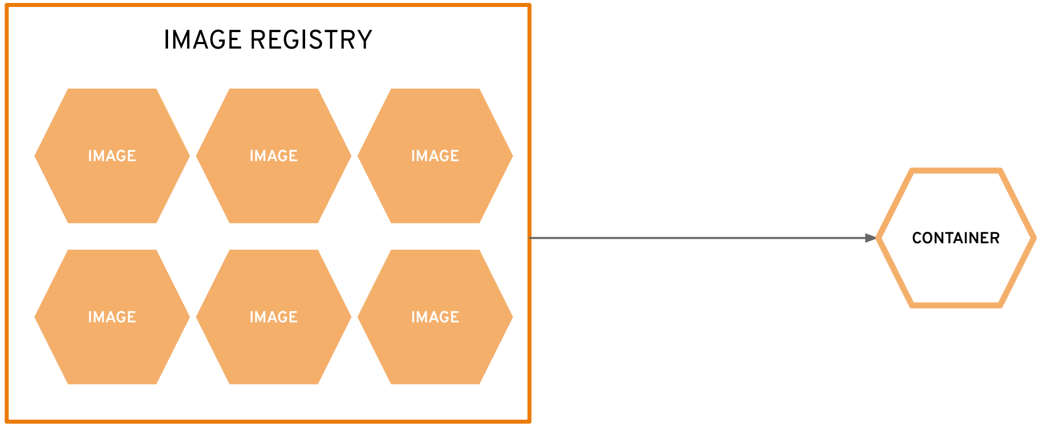
Core Concepts

a container is the smallest compute unit

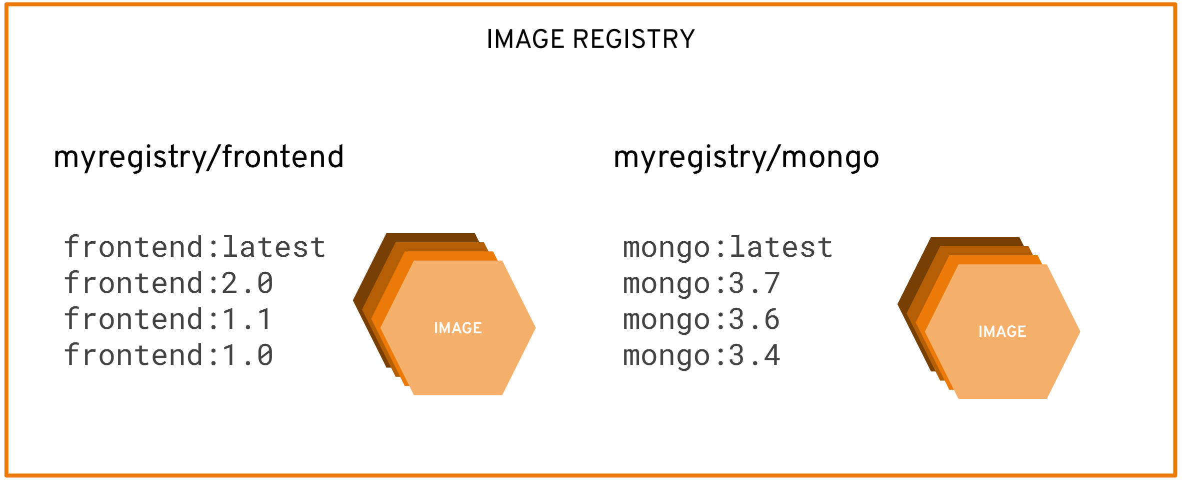
containers are created from container images



container images are stored in an image registry



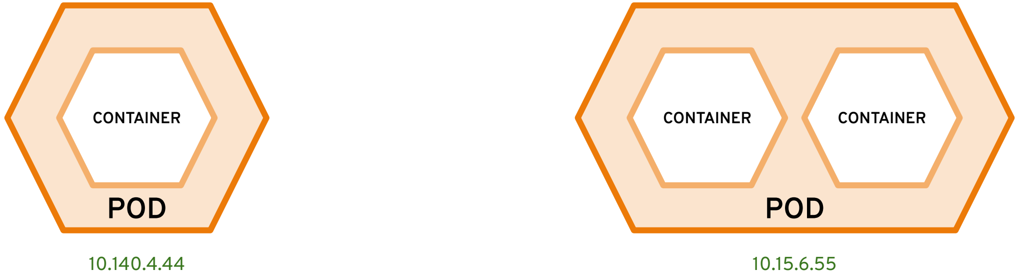
an image repository contains all versions of an image in the image registry



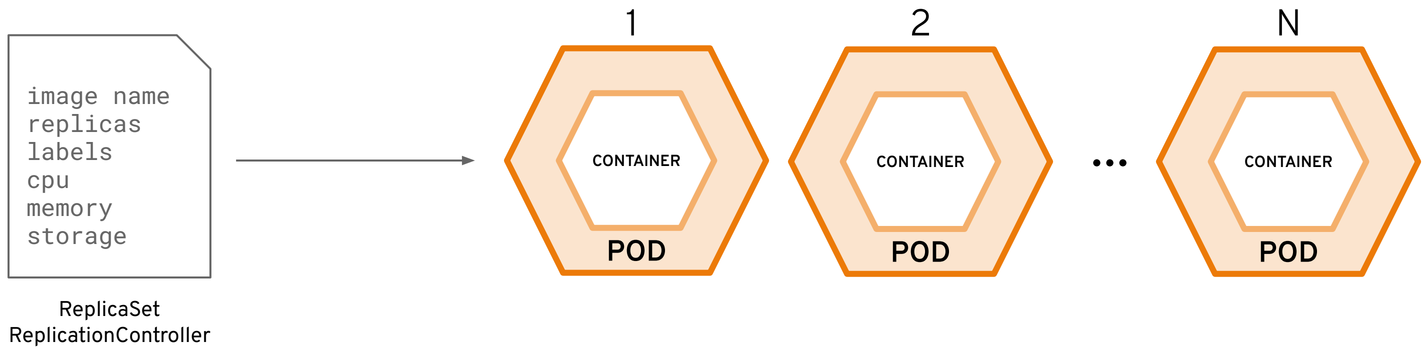
```
frontend:latest
frontend:2.0
frontend:1.1
frontend:1.0
```

```
mongo:latest
mongo:3.7
mongo:3.6
mongo:3.4
```

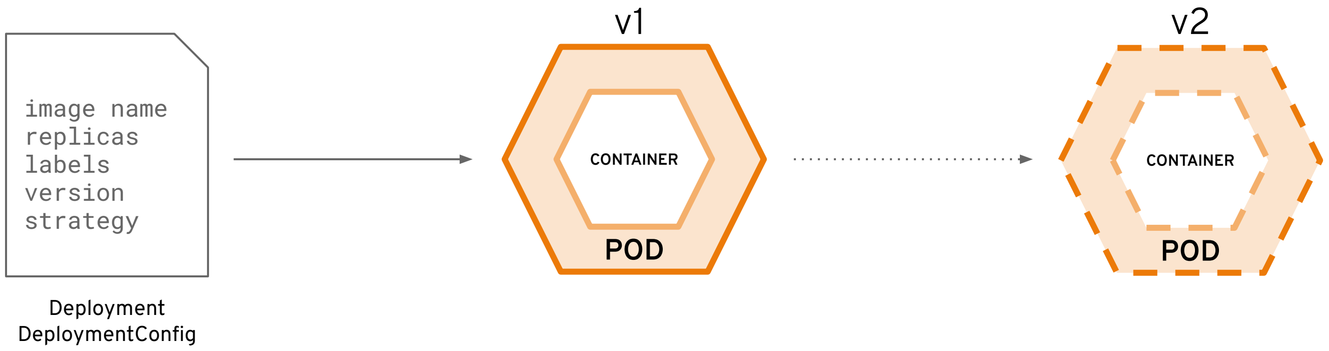
containers are wrapped in pods which are units of deployment and management



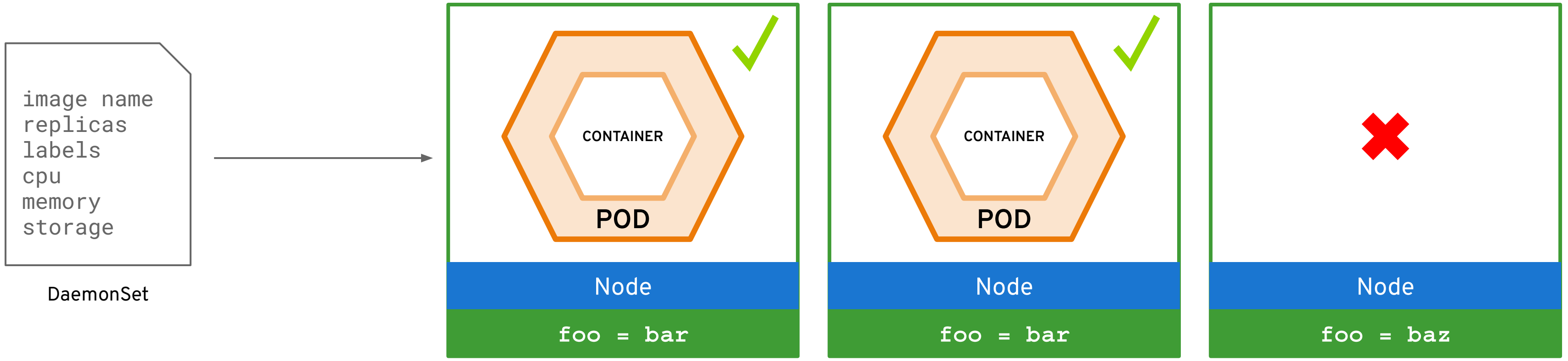
ReplicationControllers & ReplicaSets ensure a specified number of pods are running at any given time



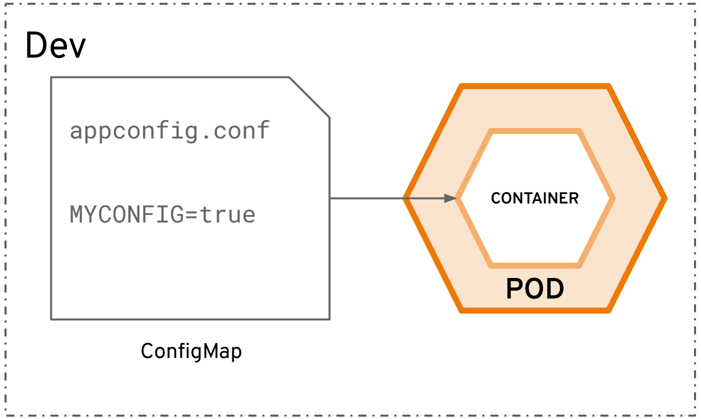
`Deployments` and `DeploymentConfigurations` define how to roll out new versions of Pods

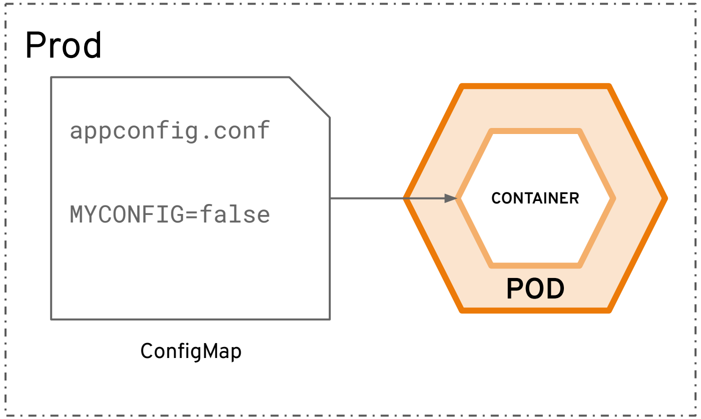


a daemonset ensures that all (or some) nodes run a copy of a pod

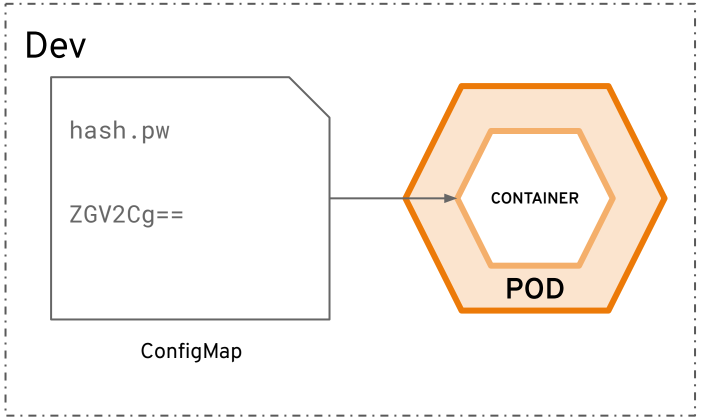


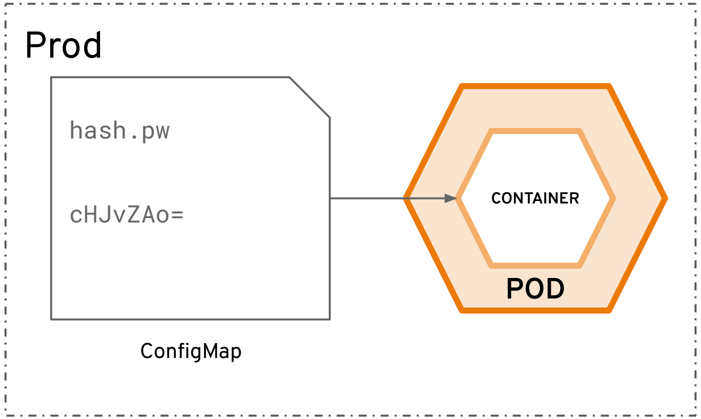
`configmaps` allow you to decouple configuration artifacts from image content





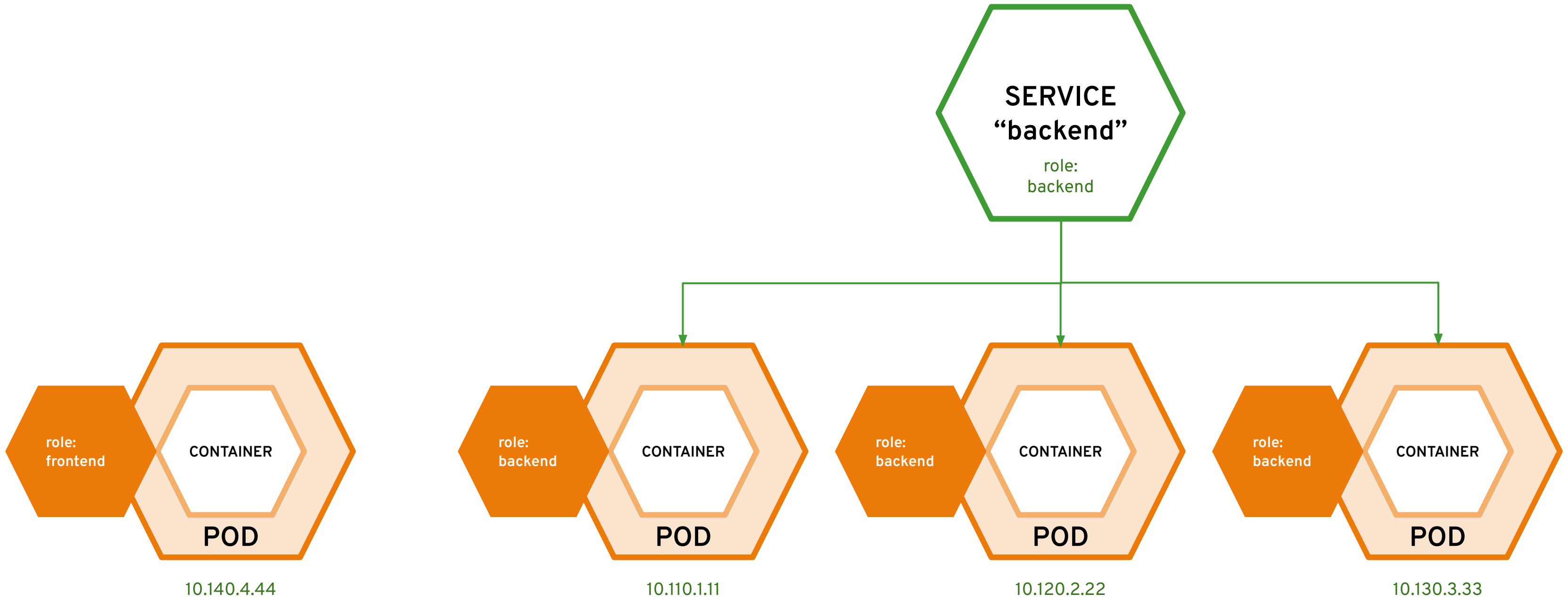
`secrets` provide a mechanism to hold sensitive information such as passwords



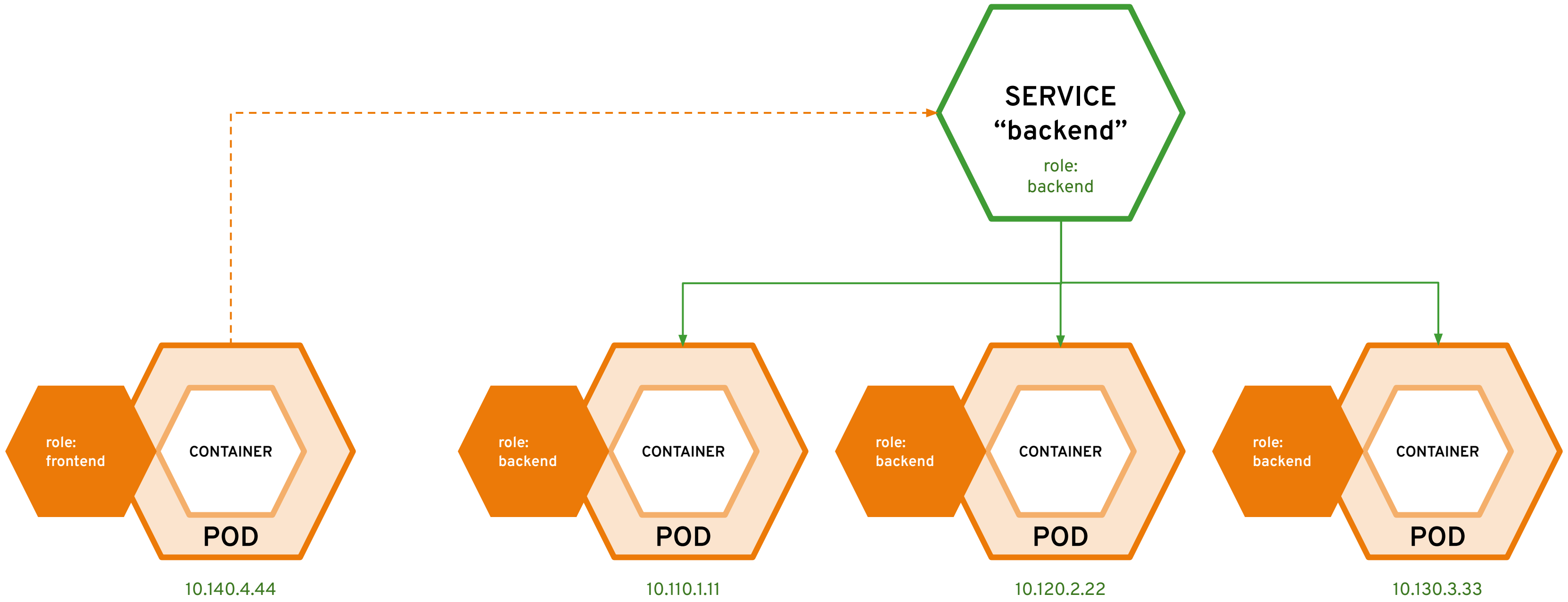


The etcd datastore can be encrypted for additional security
https://docs.openshift.com/container-platform/4.6/security/encrypting-etcd.html

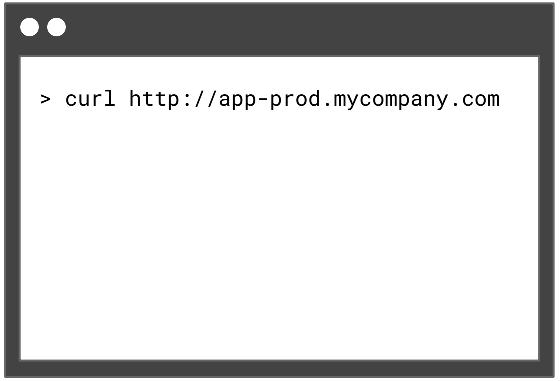
services provide internal load-balancing and service discovery across pods



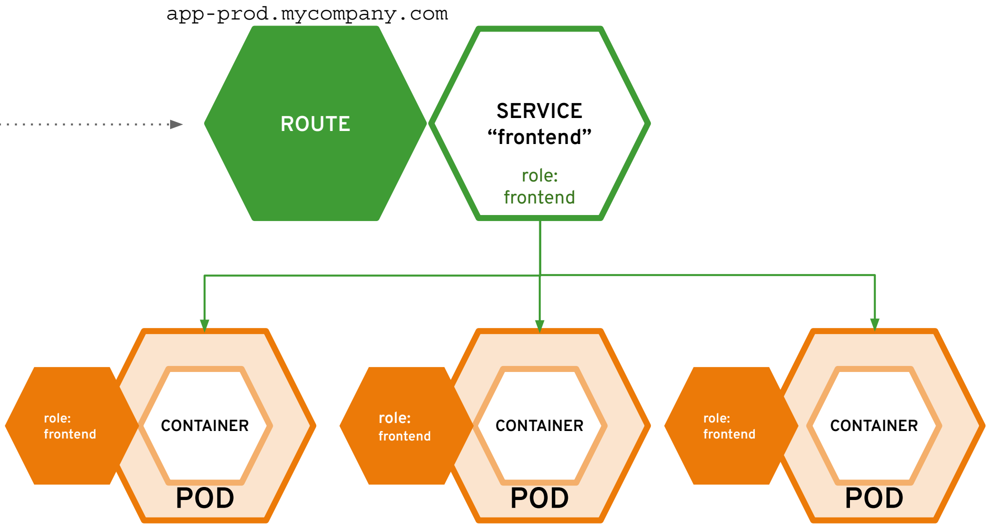
apps can talk to each other via services



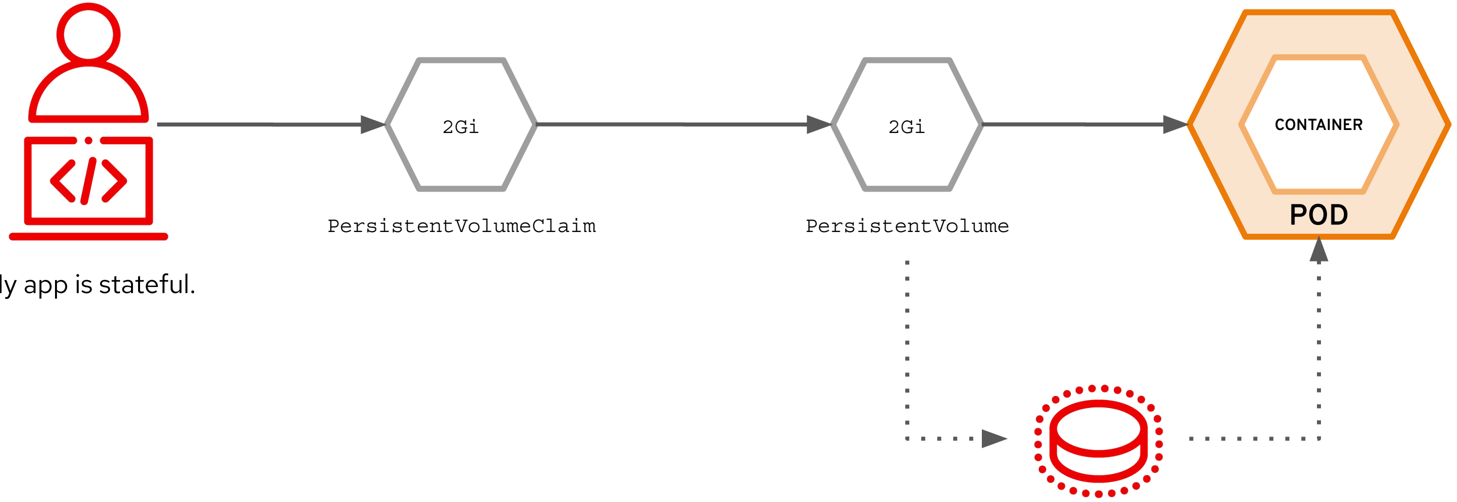
`routes` make services accessible to clients outside the environment via real-world urls



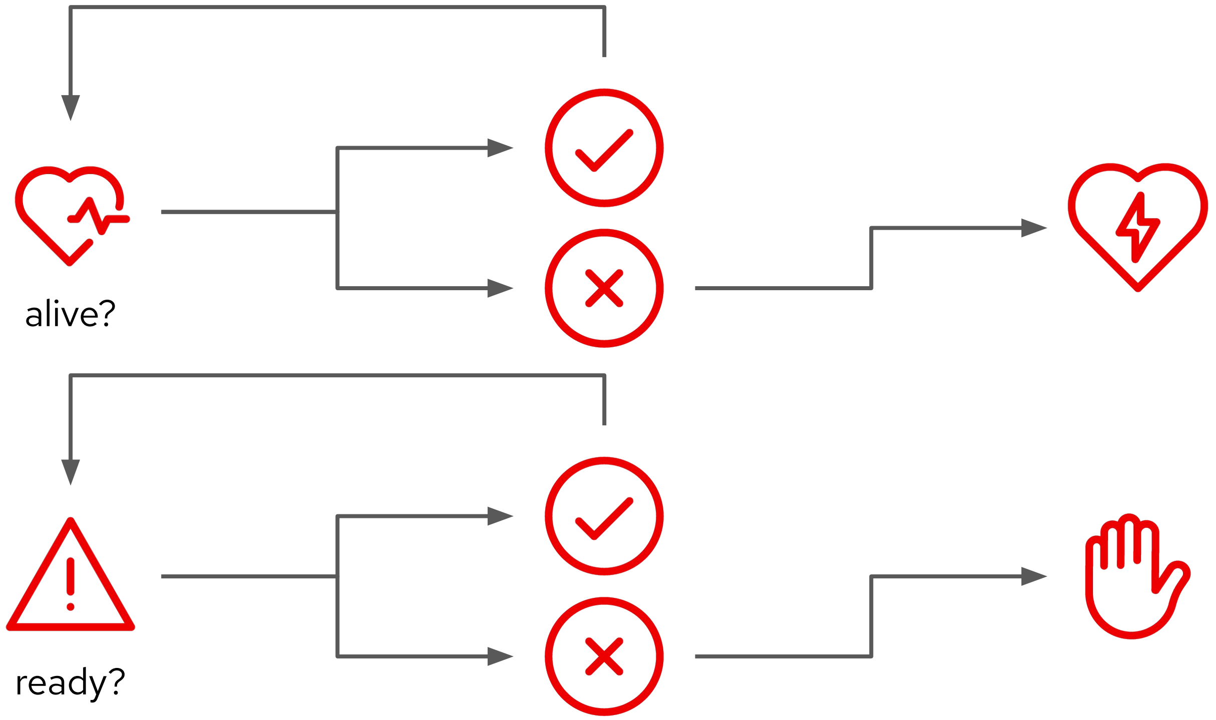
```
> curl http://app-prod.mycompany.com
```



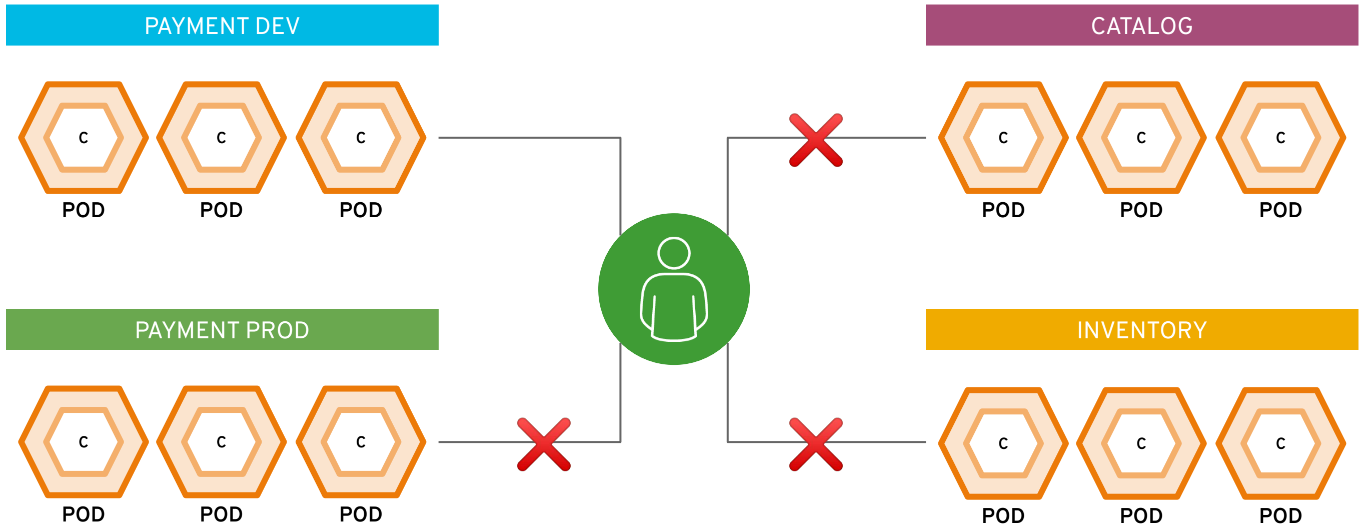
Persistent Volume and Claims



Liveness and Readiness



projects isolate apps across environments, teams, groups and departments



Packaging Basics

JUST MOVE IT ONLINE THEY SAID
HOW HARD COULD IT BE?
imgflip.com

What

- Now I have coded and my app works on my Kubernetes cluster
    - All fine
    - All done
- But wait...
- How to move those things from DEV to Test?
- How to release my software?
- (No, it’s not just one image)
    - Deployment / DeploymentConfig
    - Service
    - PVCs
    - ConfigMaps
    - Route

I'M SCARED NOW

What

- How to automatically recreate your App with all resources and dependencies?
- Once you’ve created your App with all necessary resources, you need to somehow find a way to sync it with your stages (DEV/TEST/PRE-PROD…)
- How to redistribute your App?

Zip?
Tar?
Rsynch?
Binaries?
Configuration?
Templates?
Helm Charts?
Operators?
DIY?
Kustomize?

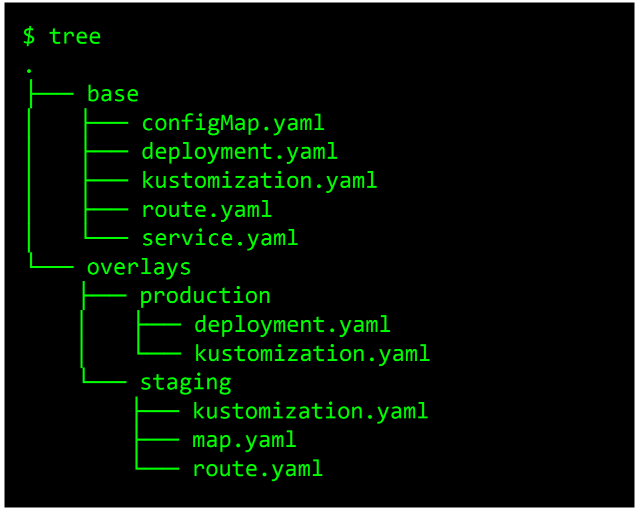
Beer!

TOO MANY CHOICES
TOO MANY OPTIONS
imgflip.com
TROLL.ME

kustomize.io

What

- Kustomize is a project originally founded Google
- It's in "kubectl apply -k" and "oc apply -k" now
- Has its own CLI interface, called kustomize
- It's NOT templating
- It's using overlays and patching



```
$ tree
.
├── base
│   ├── configMap.yaml
│   ├── deployment.yaml
│   ├── kustomization.yaml
│   ├── route.yaml
│   └── service.yaml
└── overlays
    ├── production
    │   ├── deployment.yaml
    │   └── kustomization.yaml
    └── staging
        ├── kustomization.yaml
        ├── map.yaml
        └── route.yaml
```

How it works

kustomization.yaml contains information about what to do and how

```
$ cat base/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
metadata:
  name: arbitrary

# Example configuration for the webserver
# at https://github.com/monopole/hello
commonLabels:
  app: my-hello
  org: acmeCorporation

resources:
- deployment.yaml
- service.yaml
- configMap.yaml
- route.yaml
```

```
$ cat overlays/staging/kustomization.yaml
namePrefix: staging-
commonLabels:
  variant: staging
commonAnnotations:
  note: Hello, I am staging!
bases:
- ../../base
patchesStrategicMerge:
- map.yaml
- route.yaml
```

```
$ cat overlays/production/kustomization.yaml
namePrefix: production-
commonLabels:
  variant: production
commonAnnotations:
  note: Hello, I am production!
bases:
- ../../base
patchesStrategicMerge:
- deployment.yaml
- route.yaml
```

How it works

kustomize build or oc/kubectl apply -k does handle everything for you

```
$ kustomize build base
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: my-hello
    org: acmeCorporation
  name: the-service
spec:
  ports:
  - port: 8666
    protocol: TCP
    targetPort: 8080
  selector:
    app: my-hello
    deployment: hello
    org: acmeCorporation
  type: LoadBalancer
---
```

```
$ kustomize build overlays/staging
---
apiVersion: v1
kind: Service
metadata:
  annotations:
    note: Hello, I am staging!
  labels:
    app: my-hello
    org: acmeCorporation
    variant: staging
  name: staging-the-service
spec:
  ports:
  - port: 8666
    protocol: TCP
    targetPort: 8080
  selector:
    app: my-hello
    deployment: hello
    org: acmeCorporation
    variant: staging
  type: LoadBalancer
---
```

Can I use it?

- Short answer: Of course!
- Longer answer: If you're looking for a solution that integrates nicely with GitOps, you should definitely have a look
    - Kustomize can also configure OpenShift specific types like Routes etc.
    - It does NOT use parameters
    - With kustomize CLI there is a nice way to test your layers
    - Can easily being used in your CI/CD pipelines
    - Can be versioned
    - Not too complex
    - Proven to work, easily understandable

Scenarios to use kustomize

- In-Cluster movements (DEV → TEST)
- Cross-Cluster movements (TEST → PREPROD → PROD)
- GitOps
- OpenShift Pipelines / Tekton
- NOT usable for application publishing / distribution

Drawbacks?

- You can only change existing entries and add new ones…
- You can’t use it for redistribution

Resources

Automated Application Packaging and Distribution with OpenShift - Part 1/2 – Open Sourcerers

https://kustomize.io
https://github.com/kubernetes-sigs/kustomize
https://speakerdeck.com/spesnova/introduction-to-kustomize
https://github.com/wpernath/kustomize-demo

Kustomize-DEMO

Helm Charts

What

- Helm originally invented 2015 and introduced later that year at KubeCon
- Helm moved as Kubernetes subproject in 2016 as Helm 2.0
- Helm 3.x is now (since 2020) an official CNCF project
- Helm is THE package manager for Kubernetes Applications
    - Helm is like RPM / APT for Linux
    - Or maven / npm for Java / node.js
- Helm Charts can easily be created, installed into a Kubernetes Cluster and also being upgraded
- With the Artifact Hub you have a huge repository of available community driven and maintained charts for every need

Can I use it?

- Short answer: Of course, but mainly for distributing your app!
- Longer answer: If you have an app release and you have to make it available for others, create a Helm Chart for it and make it easy for your customers (regardless of internal or external) to consume it
  - If you are just looking for a way to move your app from one stage to the other, have a look at Templates or kustomize.io
  - Helm and ArtifactHub are a great resource to look at for components you might need

Szenarios to use Helm Charts

- Well used for distribution of Applications
- Internal distribution & external
- Not so great for use within CI/CD (but possible, of course)

Drawbacks?

- Learning curve of Helm Charts is steep at the beginning
- It adds another complexity to your app development cycle
- Client is a templating engine with its own DSL and complexity
- Helm is intended for Day-1 Operations
- Helm is intended for stateless application distribution

Helm 2 vs 3

- Helm 2 required a server component called Tiller
  - Another app on top of kubernetes which had to be managed and maintained
  - Tiller had its own RBAC and its own audit trail
  - Tiller was storing sensitive data in ConfigMaps
  - → Loss of visibility
- Helm 3 does not need a server side component
  - It uses native kubernetes approach and only a client side tool

- **⇒ This is the reason why OpenShift did not natively support Helm prior V3**

Resources

- Helm.sh
- Spotlight on Helm
- To Helm or not?. Helm is becoming a very popular tool to… | by Stepan | FAUN
- From Templates to Openshift Helm Charts
- Working with Helm charts using the Developer perspective - Application life cycle management | Applications | OpenShift Container Platform 4.6
- How to make a Helm chart in 10 minutes
- Artifact Hub
- https://github.com/wpernath/helm-demo.git
- Automated Application Packaging And Distribution with OpenShift - Part 2/3 – Open Sourcerers

Helm DEMO

Summary

Summary

- All 4 packaging mechanisms discussed are solving mainly 2 different use cases
  - Application Distribution
  - CI/CD
- Helm Charts, Kubernetes Operators and kustomize are standardized kubernetes or CNCF projects.
- Templates are OpenShift specific
- Unfortunately, you have to think about 2 different mechanisms in a typical project
  - You need CI/CD → kustomize or Templates
  - You might need app distribution → Helm or Operator

Summary - CI/CD

- Use OpenShift Templates if
  - You’re purely on OpenShift
  - You need a quick and easy way to move your apps to other stages
  - You want to create special sample apps for developers
  - You want to be included in the developer perspective to choose from
  - You don’t like the approach of kustomize (patch&merge)
- Use kustomize if
  - You just want to have a standard way of doing CI/CD
  - You don’t like the template approach
  - You don’t know if you’re staying on OpenShift
  - You want to rely on kubernetes standards

Summary - Application Distribution

- Use Helm if
  - Your app is relatively easy and straight forward
  - Your app does not require special kubernetes configs
  - You app is mainly a stateless application
- Use Operators if
  - Your app requires special handling, special kubernetes custom resources (CRDs)
  - Is complex and requires a special backup strategy
  - Needs several Dependencies
  - Have a special need for Day 2 Operations
  - Is a stateful application
- Good: You can even create Operators out of a Helm Chart

Resources

- Kubernetes Operators and Helm – It takes Two to Tango
- Kubernetes Operators vs. Helm Charts: Which to Use and When
- Build Kubernetes Operators from Helm Charts in 5 steps
- Automated Application Packaging and Distribution with OpenShift – Part 1/2
- Automated Application Packaging And Distribution with OpenShift - Part 2/3

THANK YOU