

OpenShift 4

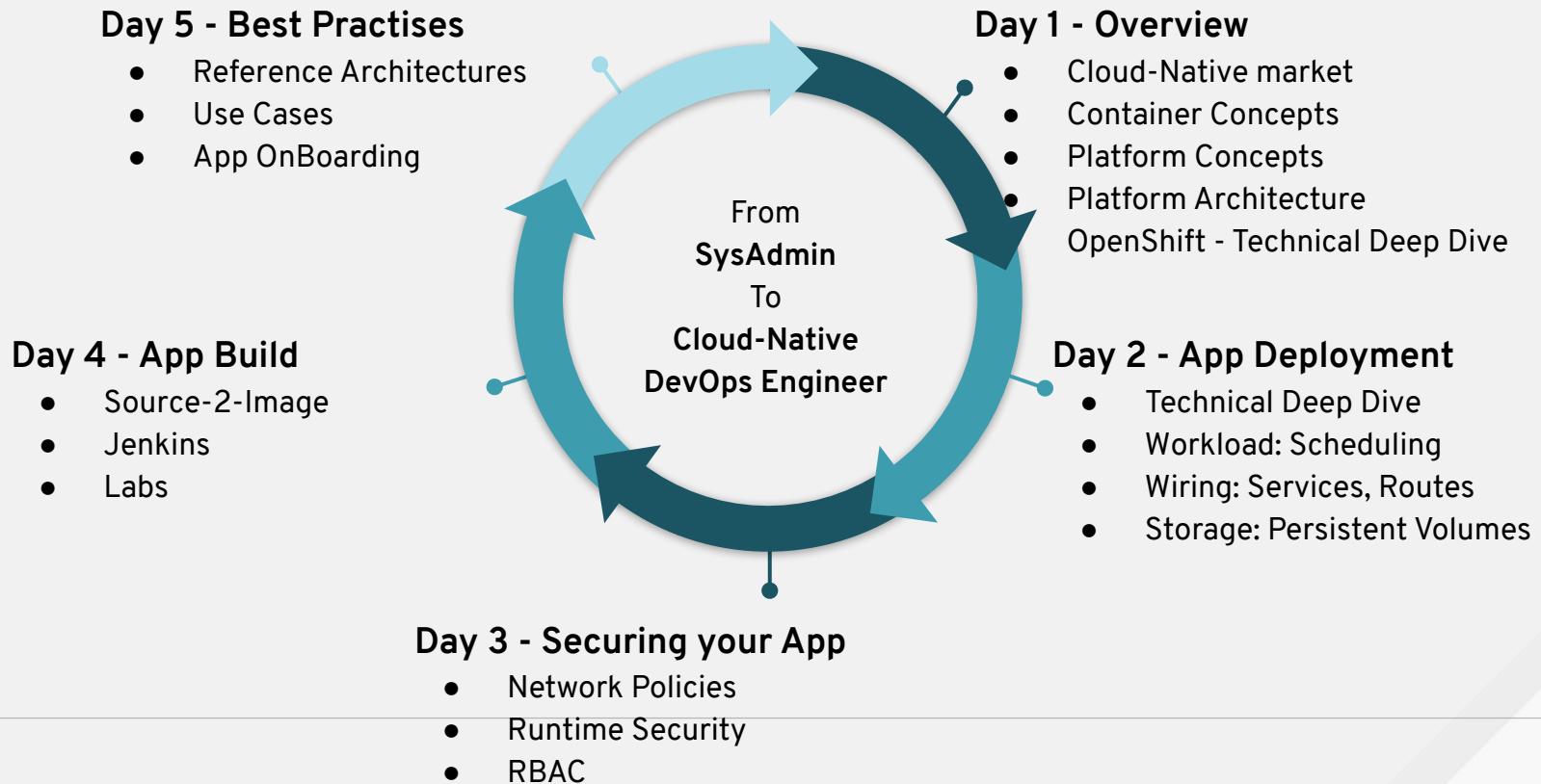
Making introductions

Samuel Terburg
Red Hat Certified Architect
Cloud-Native Expert

2020-01-13

A comprehensive overview of “OpenShift Container Platform”, accompanied with interactive Lab exercises.

Agenda



Client Setup

Gets you started

Interactive Workshop

OpenShift WebConsole	https://console-openshift-console.apps.learn.ont.belastingdienst.nl
OpenShift CLI	<code>oc login -u <vdi-user></code> https://api.learn.ont.belastingdienst.nl:6443
Workshop url	https://lab-getting-started-workshops.apps.learn.ont.belastingdienst.nl
Confluence	https://devtools.belastingdienst.nl/confluence/displayJOS/OpenShift+Opleiding+Omgeving
Source code / Slides	https://devtools.belastingdienst.nl/bitbucket/projects/CPET/repos/training-workshop/browse/resources

COMMANDS

Help	
oc	Openshift Client
oc types oc api-resources	Brief description of common used {object-types}
oc explain {object-type}	Details the fields/parameters of a specific {object-type}
oc {verb} --help	Help on command-line syntax (for specific {verb})

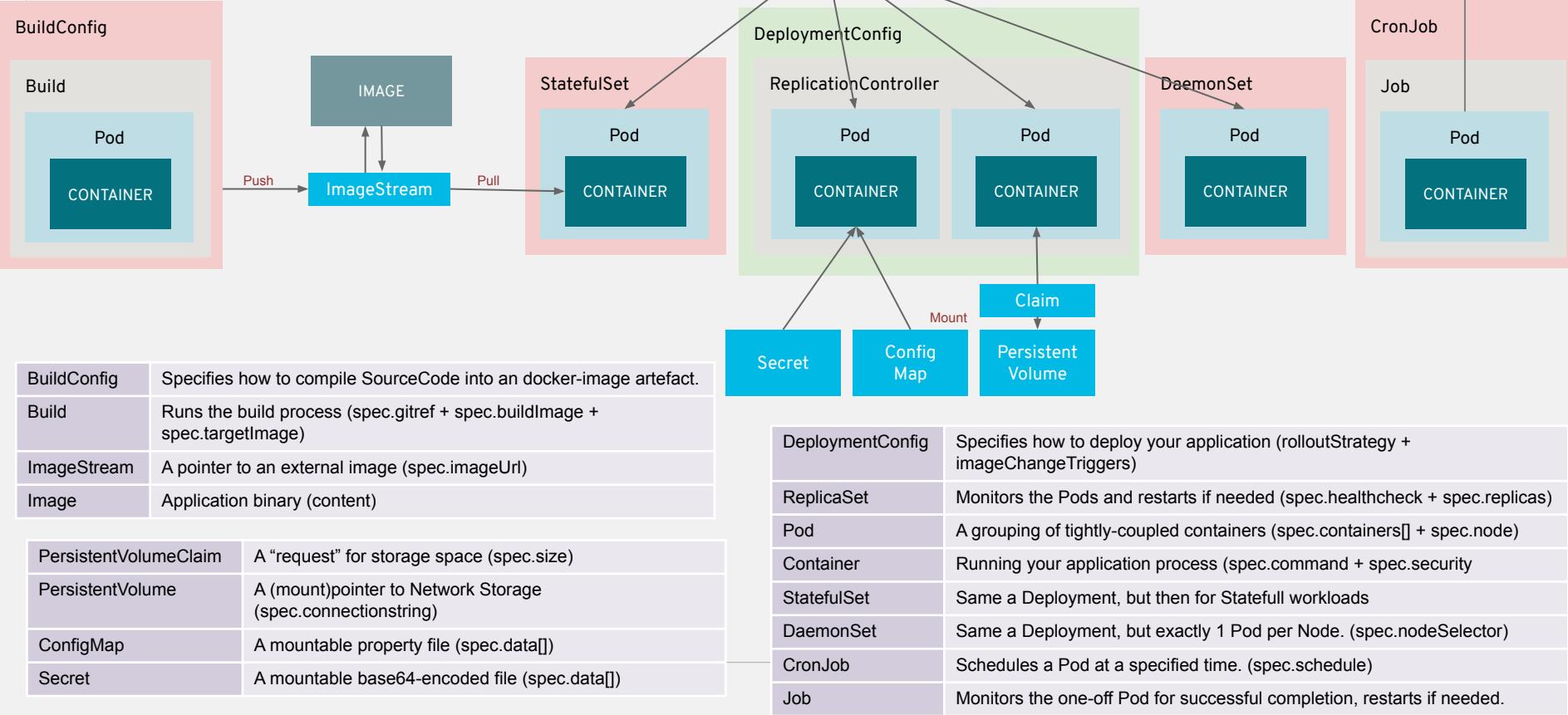
Getting Started	
oc login	Openshift Client
oc new-project	Create new Project
oc new-app	Provision new containerized application stack within your project.

oc {verb} {object-type} {object-identifier}

{verb}	
get	A (mount)pointer to Network Storage (spec.connectionstring)
create	A mountable property file (spec.data[])
edit	A mountable base64-encoded file (spec.data[])
delete	
rsh / exec	Remote shell into a Container
project	Switch current-context to other namespaces
new-project	Create new Project
new-app	Provision new containerized application stack
cp / rsync	Copy files in/out containers

Examples	
oc get projects	Overview of all Projects running
oc get pods --all-namespaces -o wide	Overview of all Pods running
oc new-project lite3-prd oc new-app --template=mytomcat --image=tomcat8	Deploy new application stack
oc start-build bc/mytomcat	Compile new docker-image
oc -n lite3 edit configmap mytomcat-properties	Change config/property-file
oc rollout latest deployment/mytomcat	Deploy new version
oc delete pod/mytomcat-1-abcd	Restart app
oc describe svc mytomcat	Detailed Info about an object and its state
oc expose svc/mytomcat --hostname=myapp.swift.com	Expose your app to the public
oc tag mytomcat:v1.0 mytomcat:prod	Promote your app to Production

OBJECTS



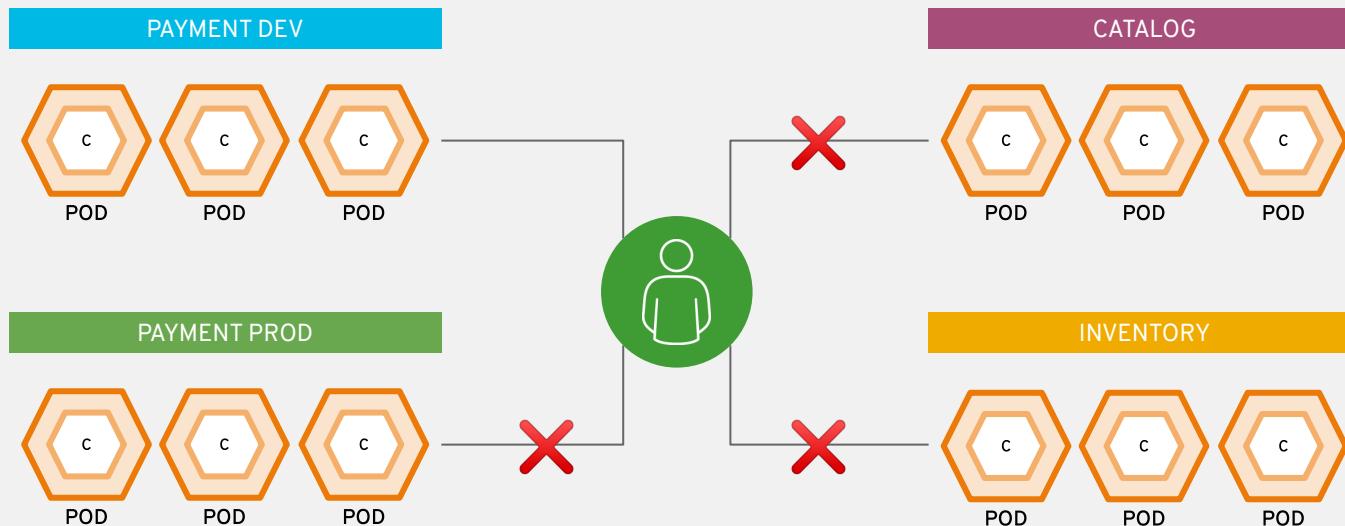
Recap - day 4

Do you still remember?

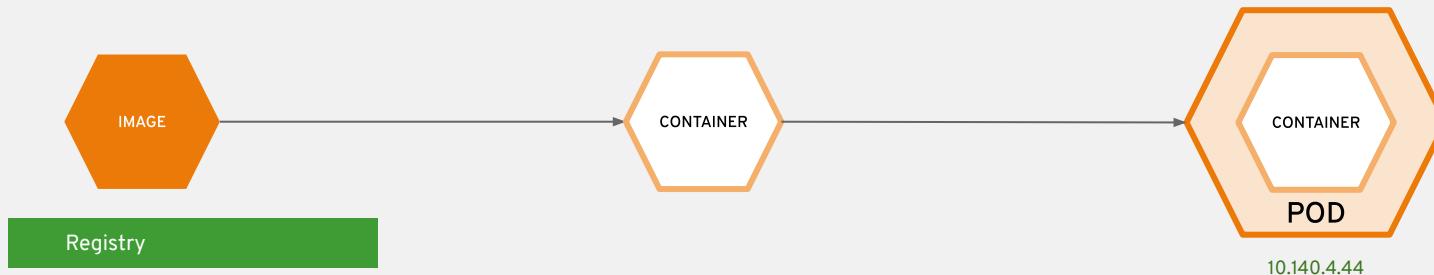
Kubernetes Resource Definitions

What types of
workloads can you
deploy on top of a
Container Platform...

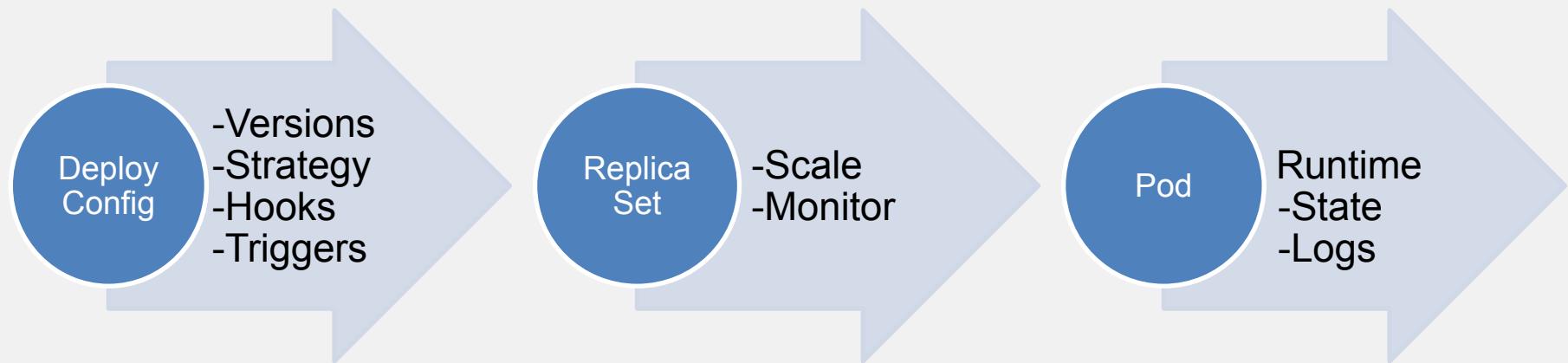
projects isolate apps across environments, teams, groups and departments



It all starts with an image



Deployment Process



- MyJBossApp

```
> oc new-app httpd  
> oc scale -replicas=2 dc/frontend  
> oc rollout latest dc/frontend
```

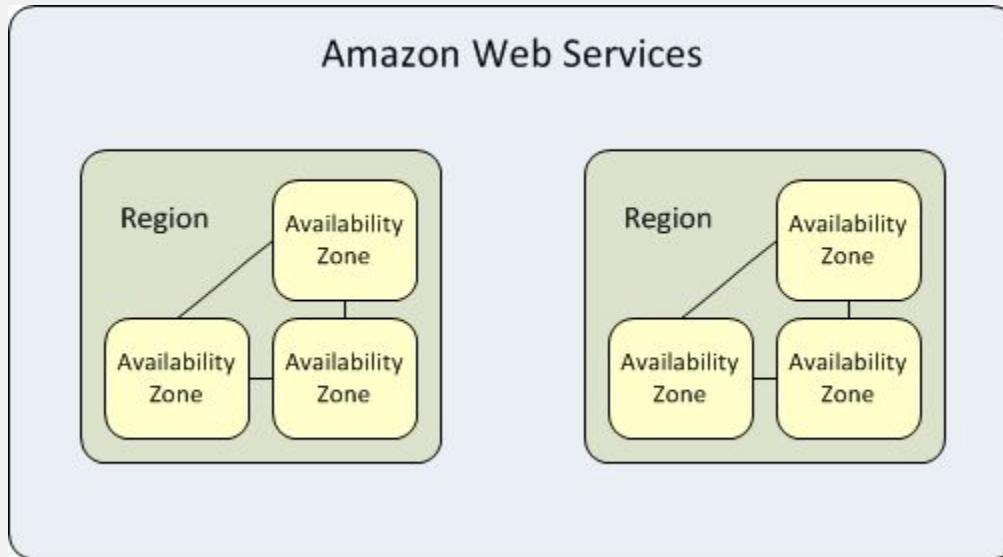
- MyJBossApp-v1 (2x)
- MyJBossApp-v2 (4x)

```
> oc get pods -o wide -w  
> curl http://<pod-ip>:8080/
```

- MyJBossApp-v1-abcde
- MyJBossApp-v1-rando

PLACEMENT BY POLICY

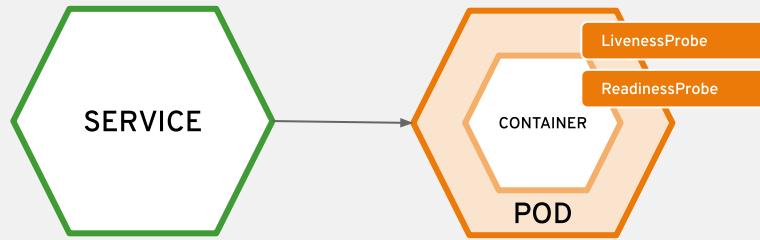
Pod Affinity rules



Preferred vs Required
Affinity vs Anti-Affinity

- Nodes
- Services
- Persistent Volumes

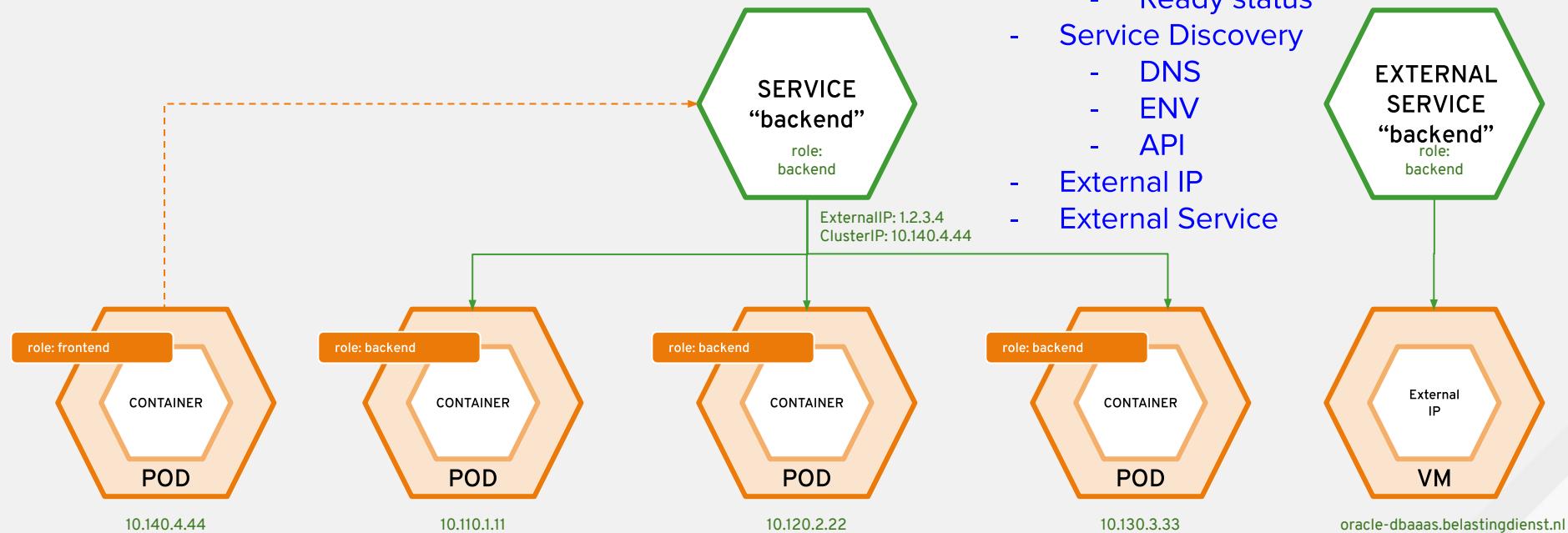
Health Checks



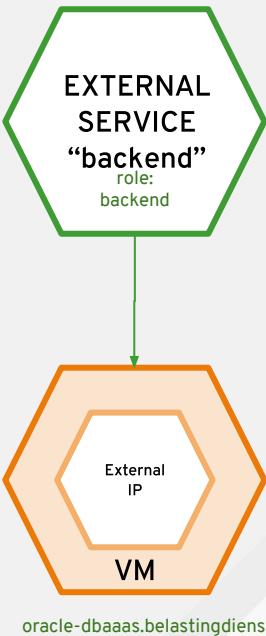
- LivenessProbe
- ReadinessProbe
 - HTTP
 - TCP
 - Shell

```
> oc set probe dc/httpd --readiness --get-url=http://127.0.0.1:8080/index.html --initial-delay-seconds=5  
> oc set probe dc/httpd --liveness --open-tcp=8080
```

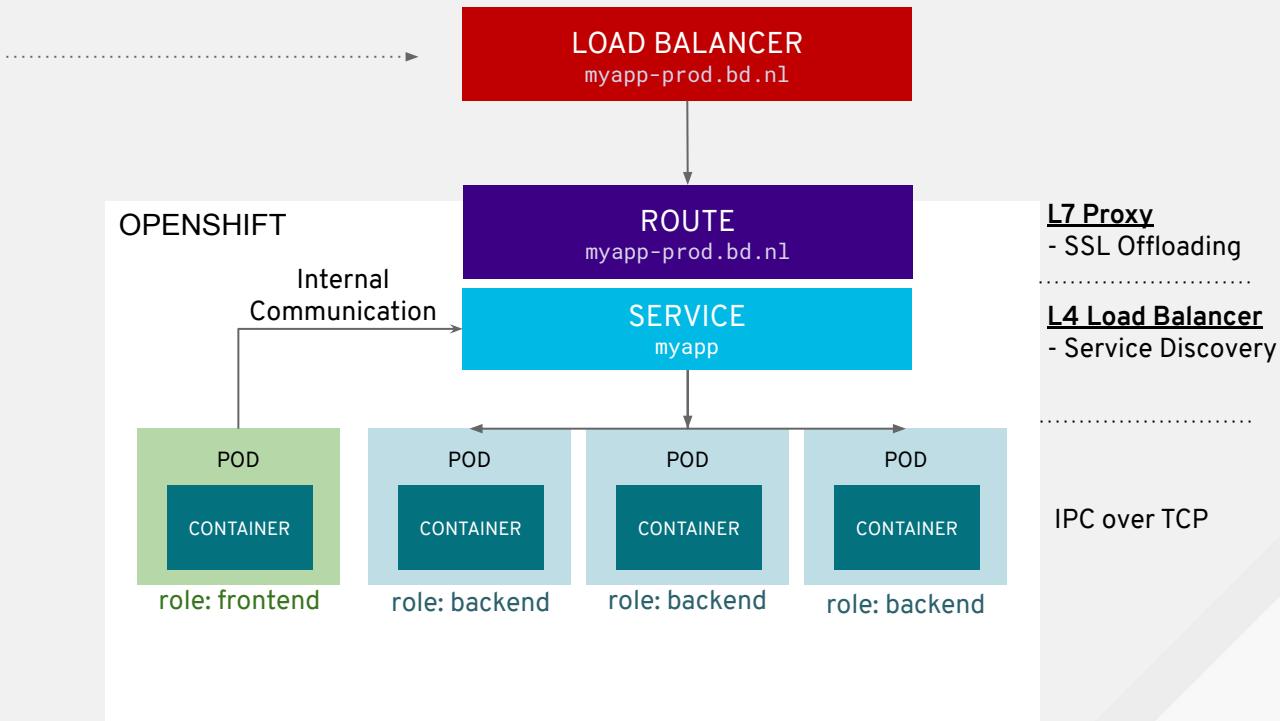
services provide internal load-balancing



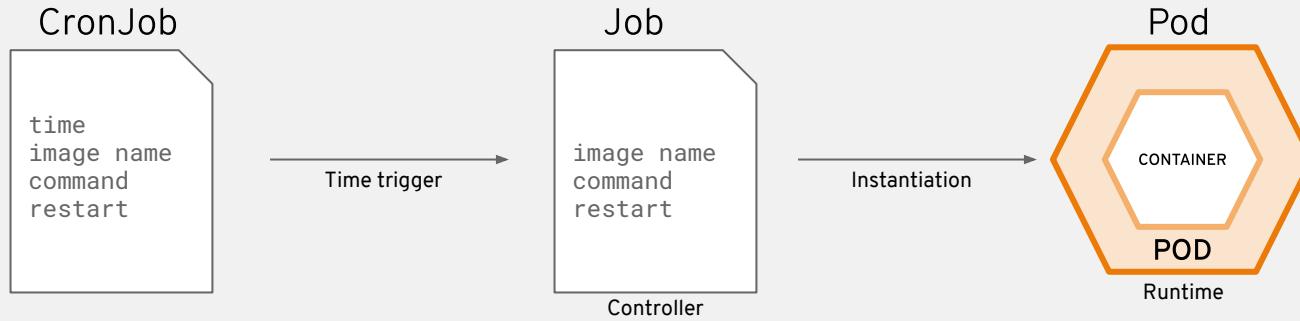
- Load-Balancing
 - Ready status
- Service Discovery
 - DNS
 - ENV
 - API
- External IP
- External Service



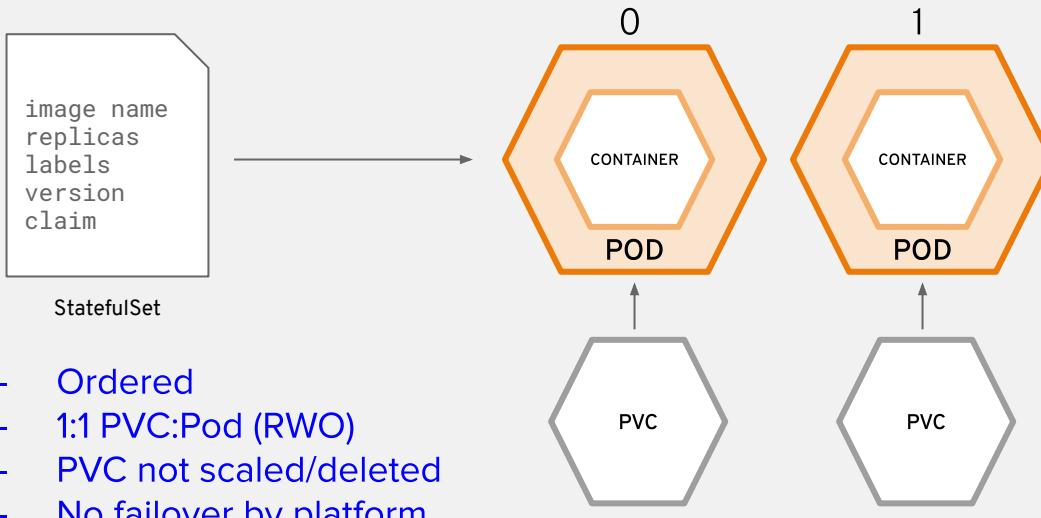
ROUTING TRAFFIC



CronJobs run short lived pods at a specified time interval



StatefulSets are special deployments for Stateful workloads

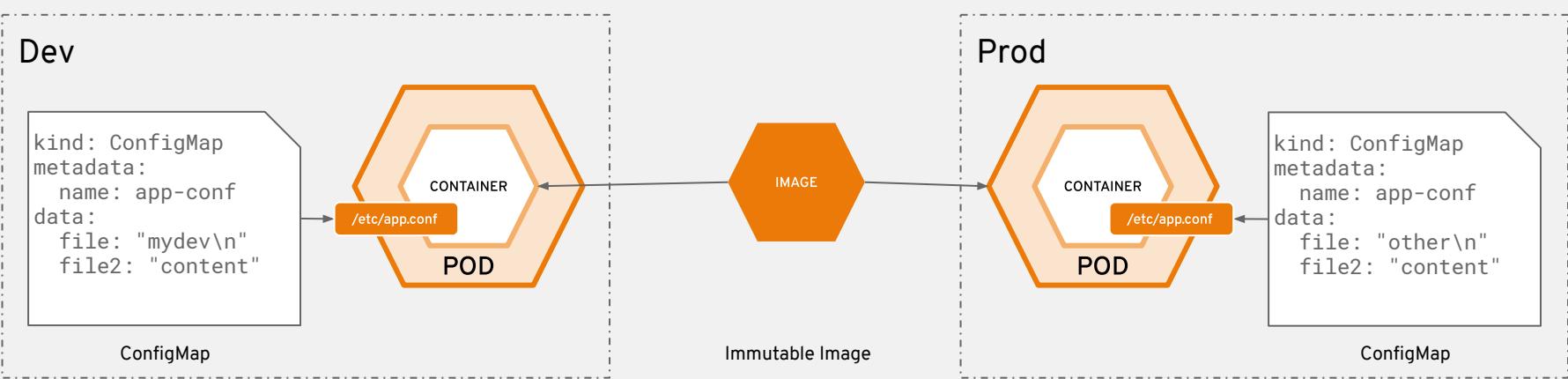


```

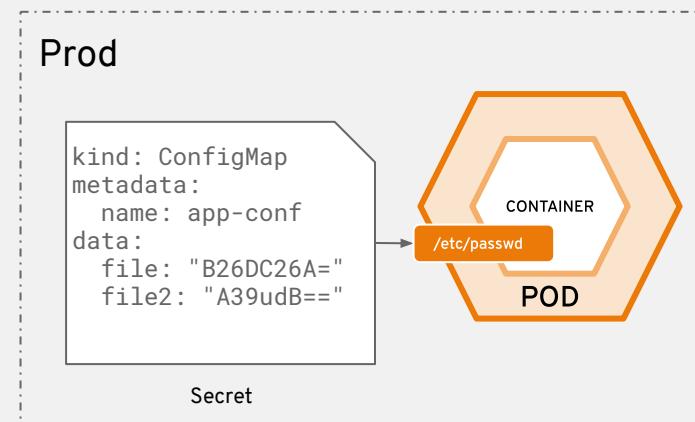
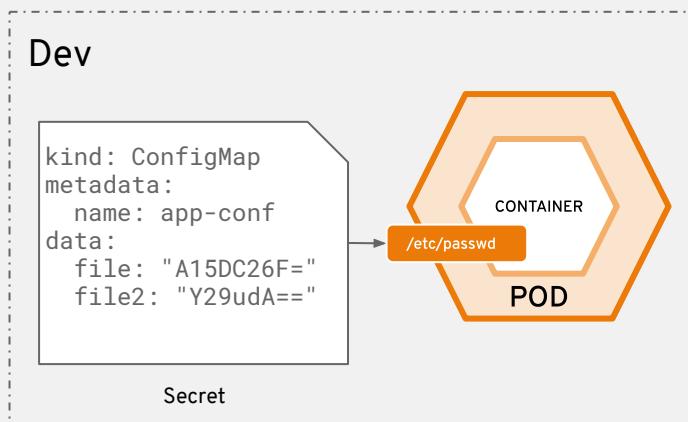
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: httpd-persistent
spec:
  serviceName: "backend"
  replicas: 2
  selector:
    matchLabels:
      app: httpd-persistent
  template:
    metadata:
      labels:
        app: httpd-persistent
    spec:
      containers:
        - name: httpd
          image: httpd:latest
          volumeMounts:
            - name: www
              mountPath: /var/www/html
  volumeClaimTemplates:
    - metadata:
        name: www
      spec:
        accessModes: [ "ReadWriteOnce" ]
        resources:
          requests:
            storage: 1Gi

```

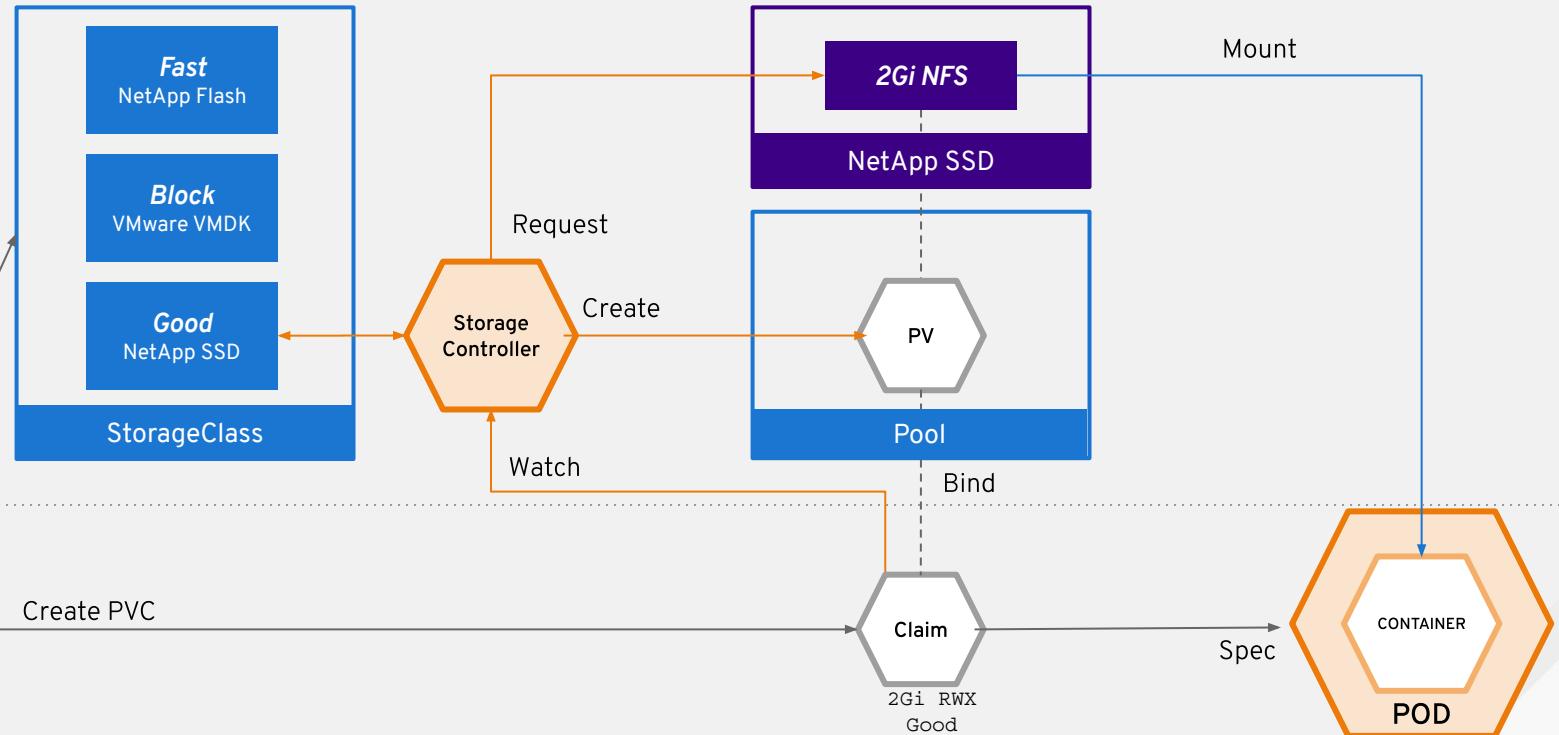
configmaps allow you to decouple configuration artifacts from image content



secrets provide a mechanism to hold sensitive information such as passwords



Dynamic Storage Provisioning



ROLE BASED ACCESS CONTROL

- Project scope & cluster scope available
- Matches request attributes (verb,object,etc)
- If no roles match, request is denied (deny by default)
- Operator- and user-level roles are defined by default
- Custom roles are supported

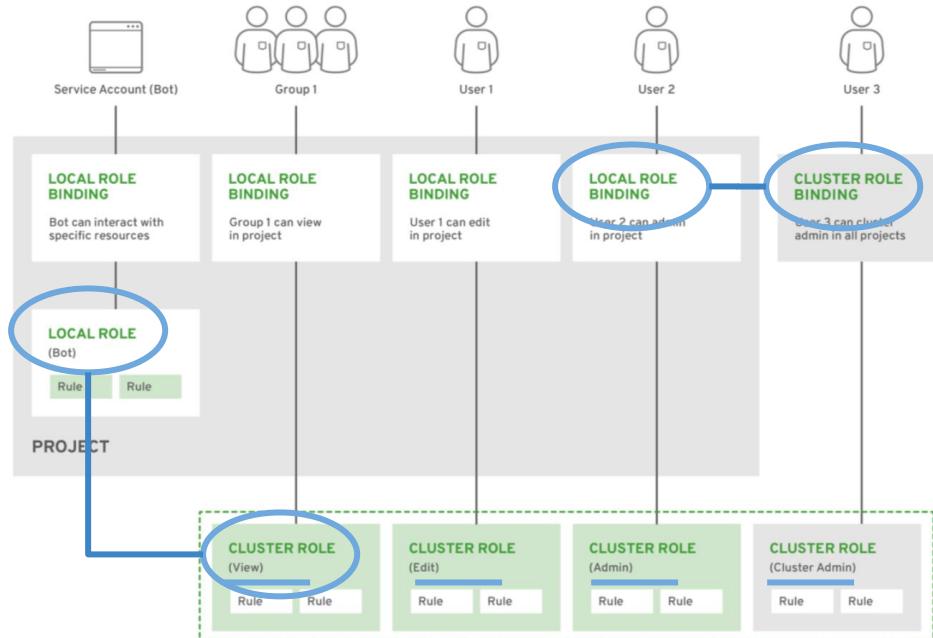


Figure 12 - Authorization Relationships

RUNTIME SECURITY POLICIES

SCC ([Security Context Constraints](#))

Allow administrators to control permissions for pods

Restricted SCC is granted to all users

By default, no containers can run as root

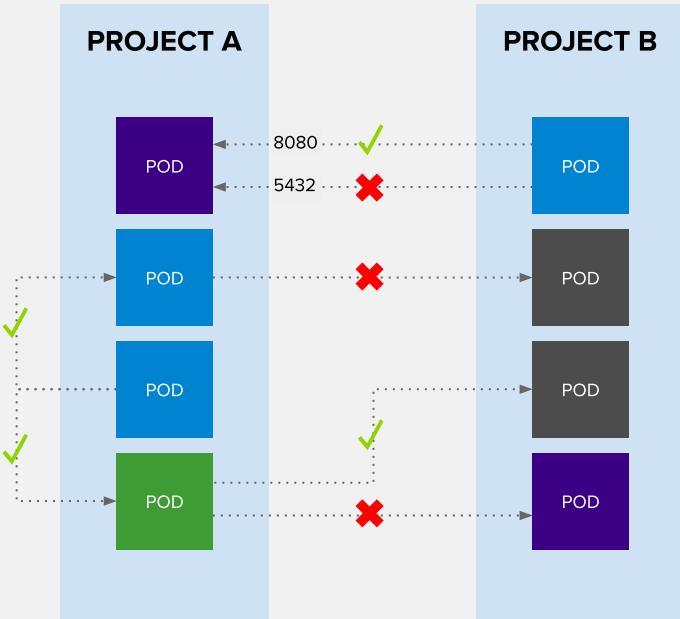
Admin can grant access to privileged SCC

Custom SCCs can be created

```
$ oc describe scc restricted
Name: restricted
Priority: <none>
Access:
  Users: <none>
  Groups: system:authenticated
Settings:
  Allow Privileged: false ←
  Default Add Capabilities: <none>
  Required Drop Capabilities: KILL,MKNOD,SYS_CHROOT,SETUID,SETGID
  Allowed Capabilities: <none>
  Allowed Seccomp Profiles: <none>
  Allowed Volume Types: configMap,downwardAPI,emptyDir,persistentVolumeClaim,projected,
  Allow Host Network: false
  Allow Host Ports: false
  Allow Host PID: false
  Allow Host IPC: false
  Read Only Root Filesystem: false
  Run As User Strategy: MustRunAsRange
```

Network Policies

- Internal Pod-network
- Ingress / Egress
- From / To:
 - ipBlock
 - namespaceSelector
 - podSelector
- Default allow / deny



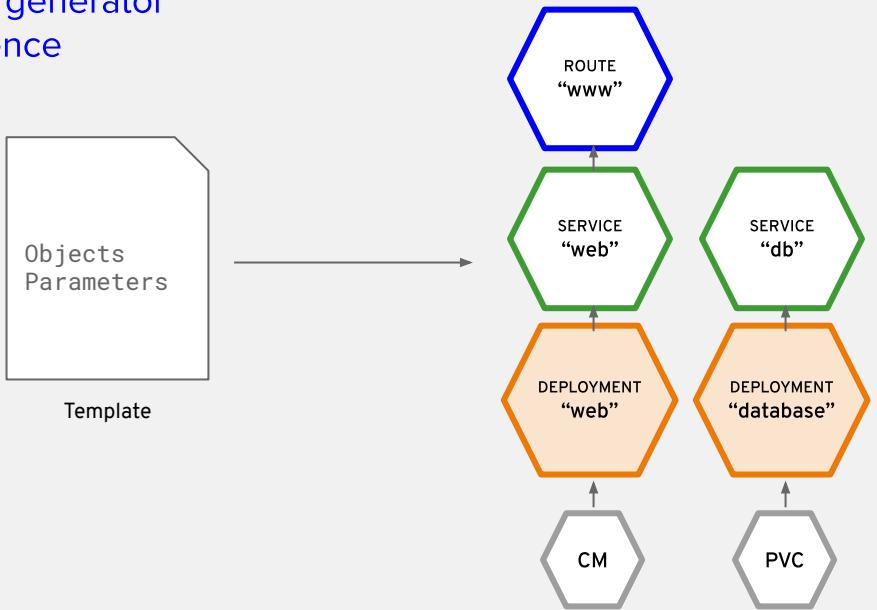
Example Policies

- Allow all traffic inside the project
- Allow traffic from green to gray
- Allow traffic to purple on 8080

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-to-purple-on-8080
spec:
  podSelector:
    matchLabels:
      color: purple
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          Name: default
  ports:
  - protocol: tcp
    port: 8080
```

a Template is a list of composite resources

- Parameter / Interpolation
- Expression generator
- No intelligence
- Param-file



```

apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: tomcat9-mysql-persistent-s2i
  namespace: openshift
objects:
- apiVersion: v1
  kind: Service
  metadata:
    name: ${APPLICATION_NAME}
  spec:
    ports:
      - port: ${LISTEN_PORT}
        targetPort: 8080
- apiVersion: v1
  kind: Service
  metadata:
    name: ${APPLICATION_NAME}-mysql
  spec:
    ports:
      - port: 3306
parameters:
- displayName: Application Name
  name: APPLICATION_NAME
  required: true
  value: jws-app
  from: '[A-Z0-9]{8}'
  generate: expression

```

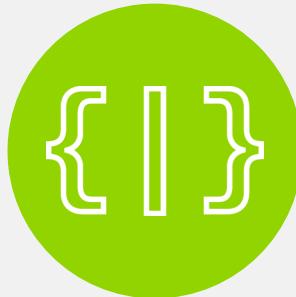
Build & Deploy Images

Let's discuss the
built-in CI/CD tools and
processes

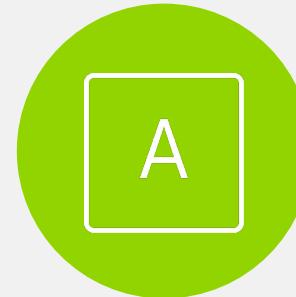
BUILD CONTAINER IMAGES



**BUILD FROM
BINARY**



**BUILD FROM
SOURCE CODE**



**BUILD FROM
Dockerfile**



**CUSTOM BUILDER
IMAGE**

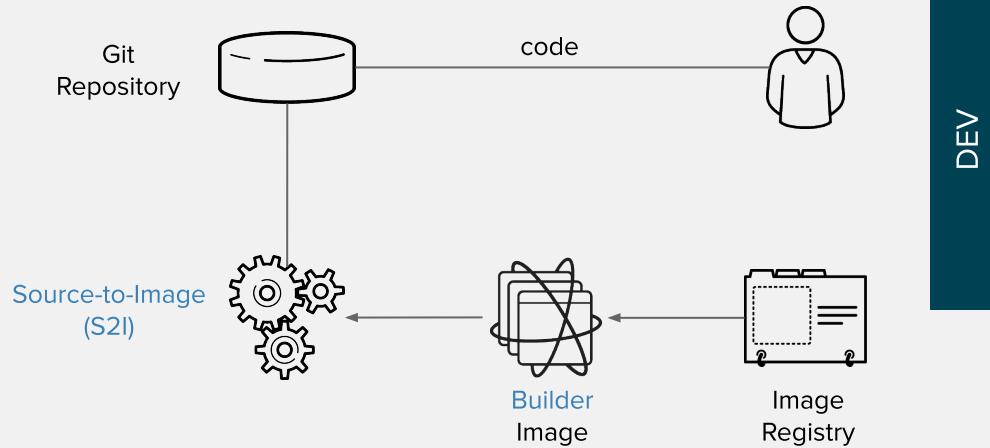
**BUILD FROM
Jenkinsfile**

- Git
- S2I Builder Image needed
- Build tools included
- Bash scripts

BUILD

S2I starts a builder containers using the chosen builder image (language and application runtime) and a /tmp/src mounted with source-code from the git repository. Executes the /usr/local/s2i/assemble script. Resulting in a new target image pushed back into the docker-registry.

BUILD SOURCE-CODE

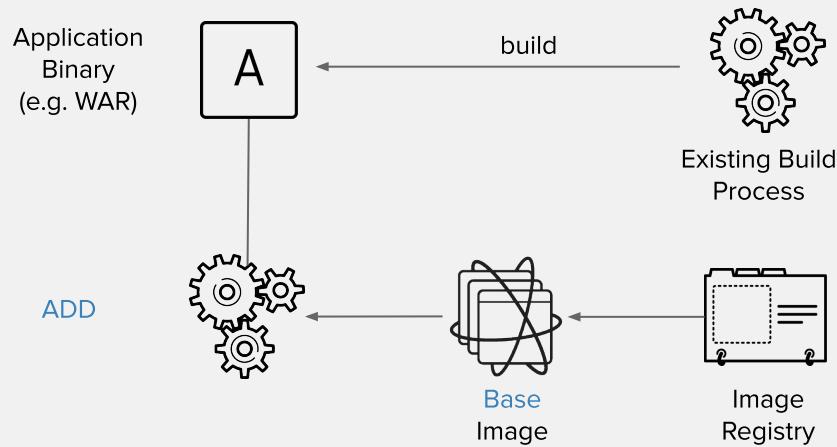


- Upload file/dir
- Needs existing BuildConfig
- Source/Docker strategy supported

BUILD IMAGE

S2I adds the app binary file (e.g. JAR, WAR, EAR) as a new layer on top of the base image (language and application runtimes) and stores the resulting application image in the image registry under its new target image name.

BINARY BUILD



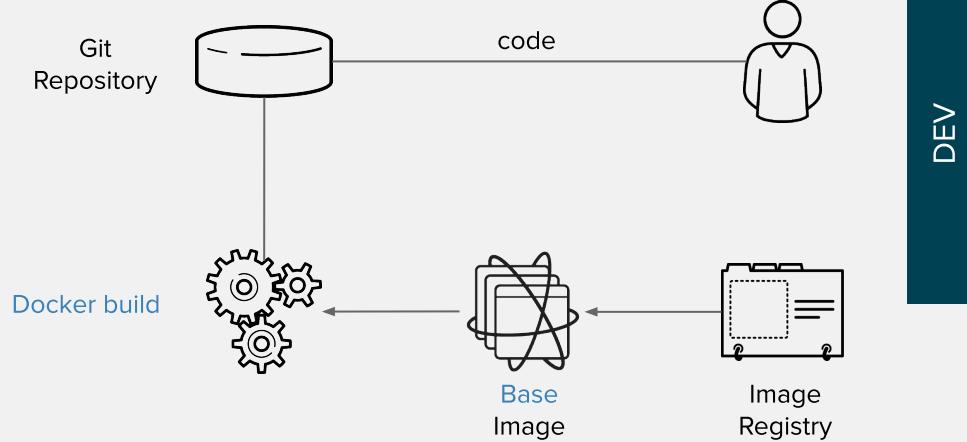
- Git | Inline
- Traditional
- FROM replaced with IS

BUILD

S2I Build process `git clone`'s the source-code, then simply run a `docker build` to build started image.

Source image is extracted from 1st FROM line in Dockerfile.

BUILD Dockerfile



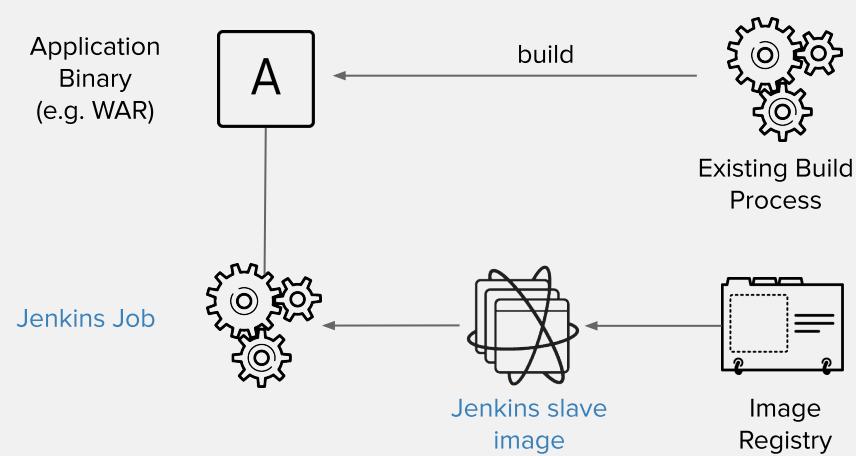
```
| > cat Dockerfile | oc new-build myapp
```

BUILD Jenkinsfile

- Git | Inline
- Starts Jenkins pod + job
- OCP/K8s plugins

BUILD IMAGE

For every BuildConfig openshift creates a Job in Jenkins. For every Build openshift triggers that Jenkins Job. Jenkins takes care of everything.



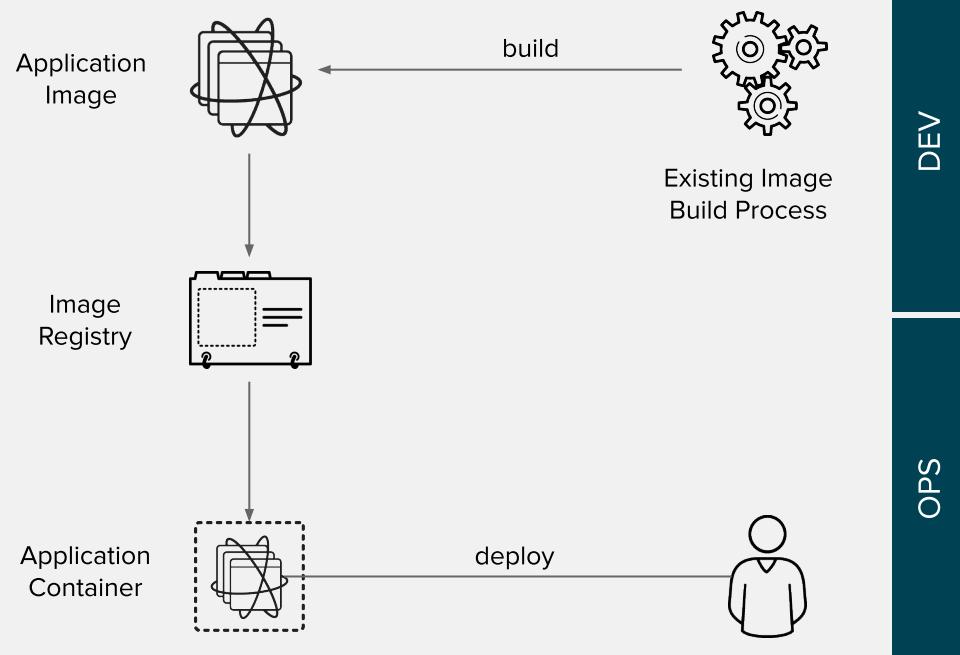
```
> cat Jenkinsfile | oc new-build myapp
```

DEPLOY DOCKER IMAGE

BUILD

App images are built using an existing image build process. OpenShift automates the deployment of app containers across multiple hosts via the Kubernetes. Users can trigger deployments, rollback, configure A/B, etc

DEPLOY



```
> oc new-app external-registry/external-project/myimage:latest
```

Jenkins Integrations

Let's discuss the
Jenkins CI/CD tools
and processes

CI/CD WITH BUILD AND DEPLOYMENTS

BUILDS

- Webhook triggers: build the app image whenever the code changes
- Image trigger: build the app image whenever the base language or app runtime changes
- Build hooks: test the app image before pushing it to an image registry

DEPLOYMENTS

- Deployment triggers: redeploy app containers whenever configuration changes or the image changes in the OpenShift integrated registry or upstream registries

```
> oc new-app httpd-example  
> oc get bc/httpd-example -o yaml |grep triggers: -A10
```

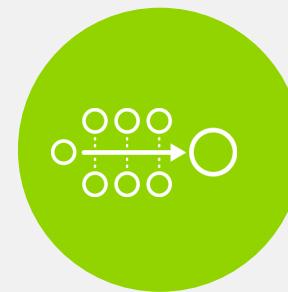
OPENSHIFT LOVES CI/CD



JENKINS-AS-A SERVICE
ON OPENSHIFT



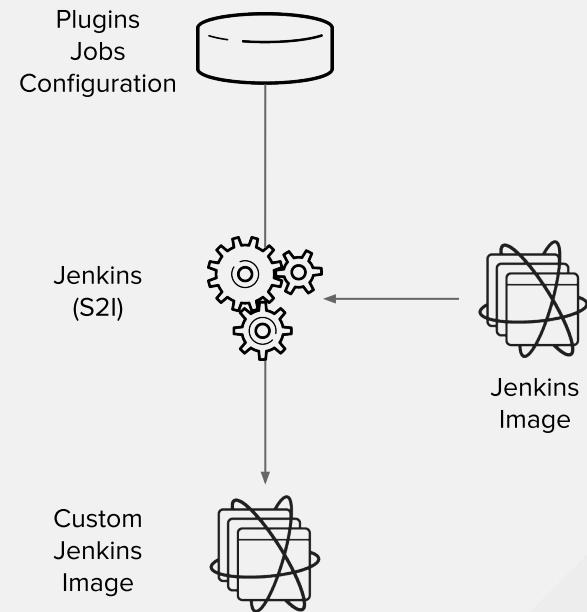
HYBRID JENKINS INFRA
WITH OPENSHIFT



EXISTING CI/CD
DEPLOY TO OPENSHIFT

1) JENKINS-AS-A-SERVICE ON OPENSIFT

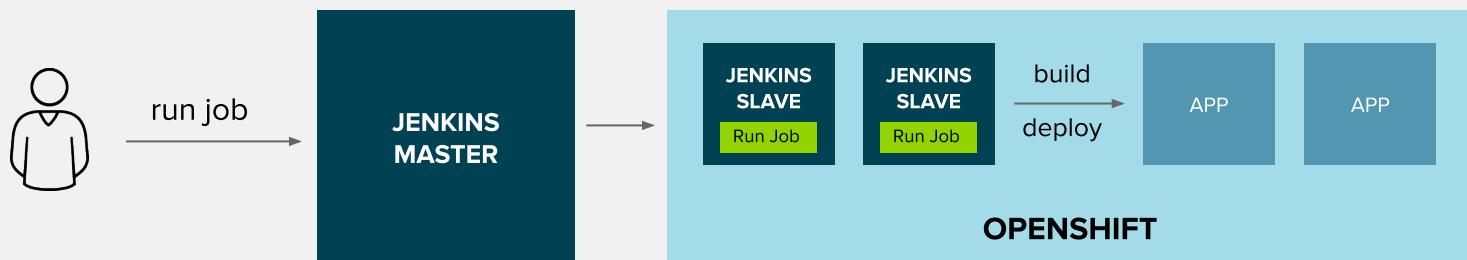
- Certified Jenkins images with pre-configured plugins
 - Provided out-of-the-box
 - Follows Jenkins 1.x and 2.x LTS versions
- Jenkins S2I Builder for customizing the image
 - Install Plugins
 - Configure Jenkins
 - Configure Build Jobs
- OpenShift plugins to integrate authentication with OpenShift and also CI/CD pipelines
- Dynamically deploys Jenkins slave containers



```
> oc new-app jenkins-ephemeral  
> ssh master1 grep -A7 jenkinsPipelineConfig: /etc/origin/master/master-config.yaml
```

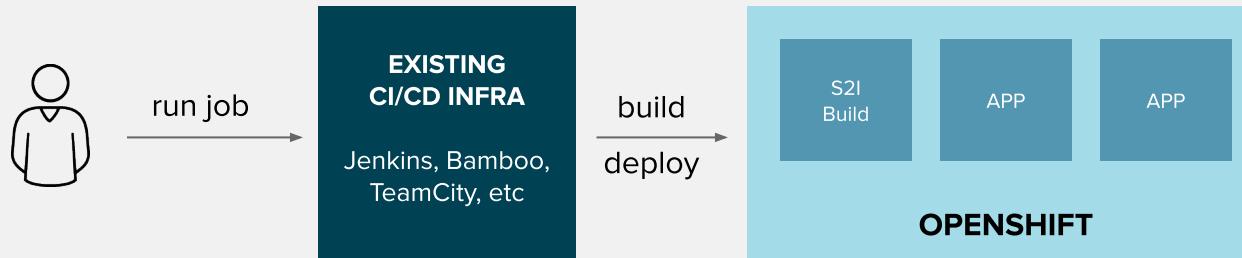
2) HYBRID JENKINS INFRA WITH OPENSHIFT

- Scale existing Jenkins infrastructure by dynamically provisioning Jenkins slaves on OpenShift
- Use Kubernetes plug-in on existing Jenkin servers



3) EXISTING CI/CD DEPLOY TO OPENSIFT

- Existing CI/CD infrastructure outside OpenShift performs operations against OpenShift
 - OpenShift Pipeline Jenkins Plugin for Jenkins
 - OpenShift CLI for integrating other CI Engines with OpenShift
- Without disrupting existing processes, can be combined with previous alternative



OPENSHIFT PIPELINES

- OpenShift Pipelines allow defining a CI/CD workflow via a Jenkins pipeline which can be started, monitored, and managed similar to other builds
- Dynamic provisioning of Jenkins slaves
- Auto-provisioning of Jenkins server
- OpenShift Pipeline strategies
 - Embedded Jenkinsfile
 - Jenkinsfile from a Git repository

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: app-pipeline
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        node('maven') { ←.....  
          stage('build app') {  
            git url: 'https://git/app.git'  
            sh "mvn package"  
          }  
          stage('build image') {  
            sh "oc start-build app --from-file=target/app.jar"  
          }  
          stage('deploy') {  
            openshiftDeploy deploymentConfig: 'app'  
          }  
        }
```

Provision a Jenkins slave for running Maven

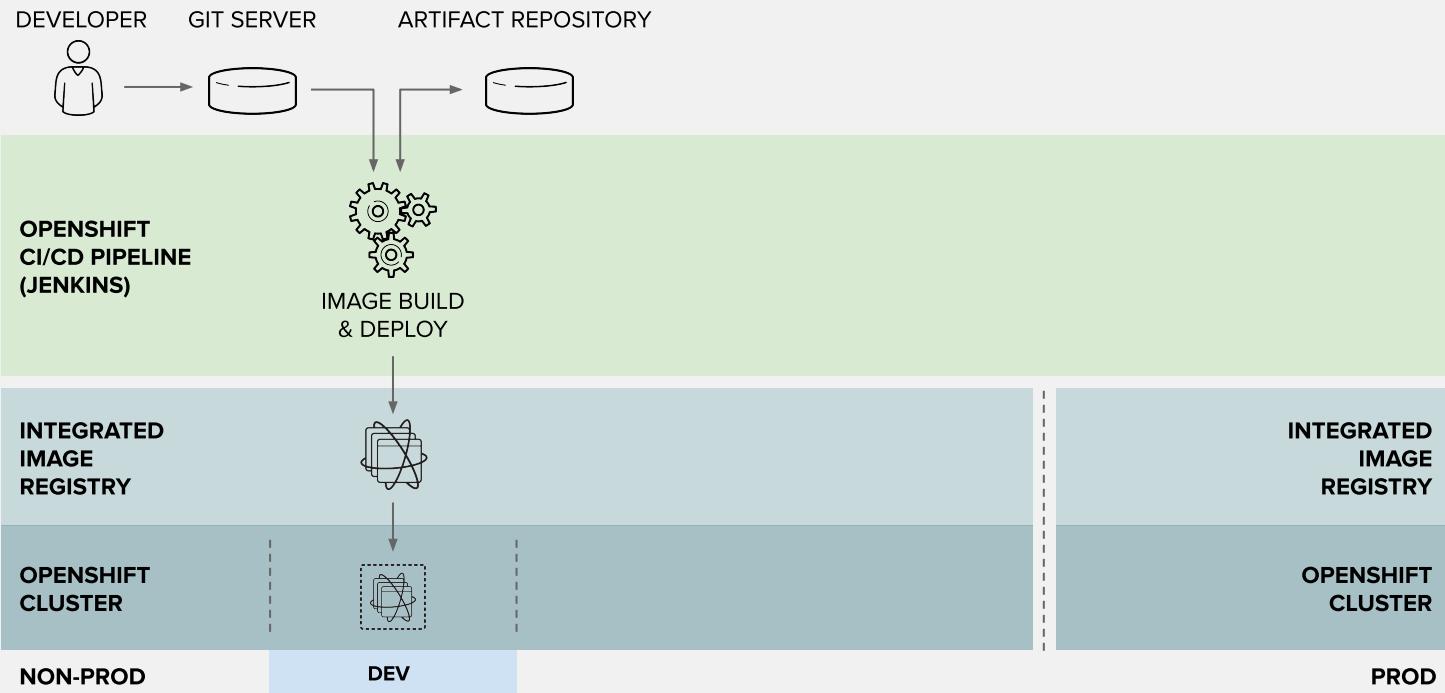
Design Patterns

Commonly used
workflows

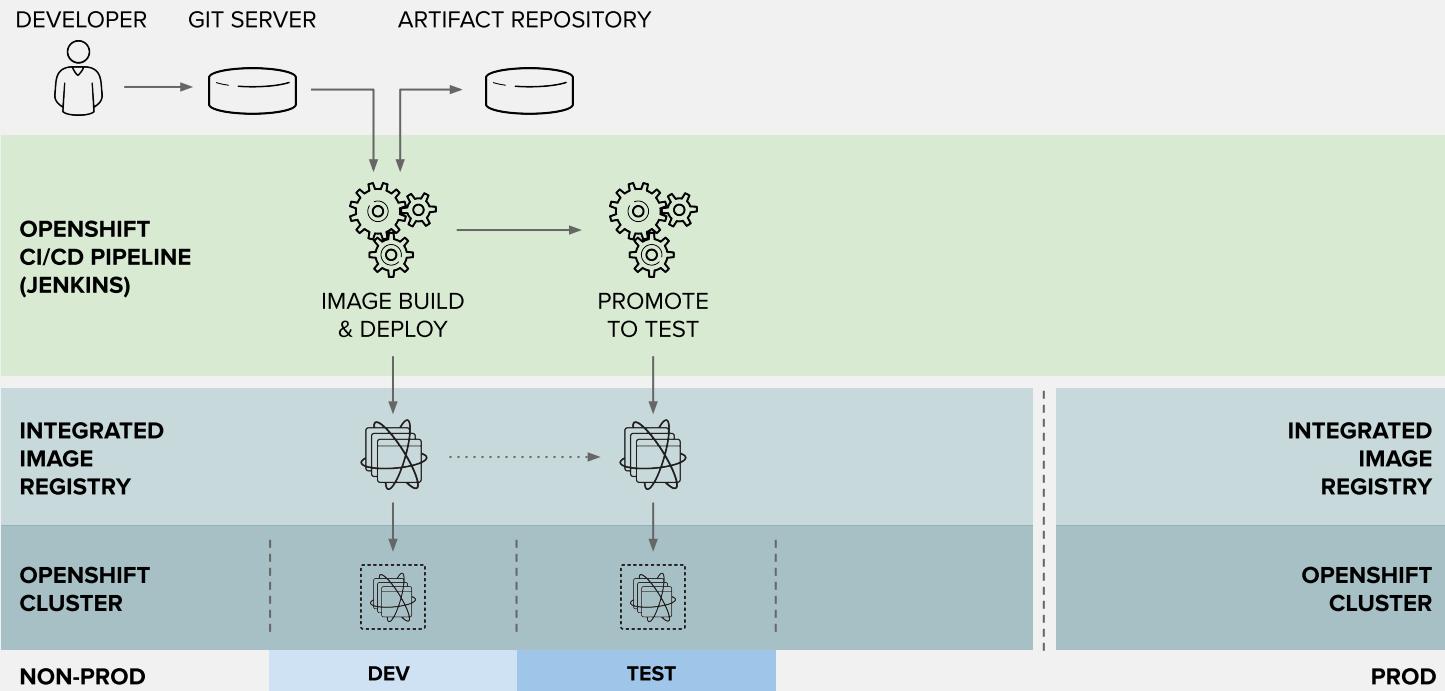
Image Promotion

Promote & Deploy into
Production
automatically

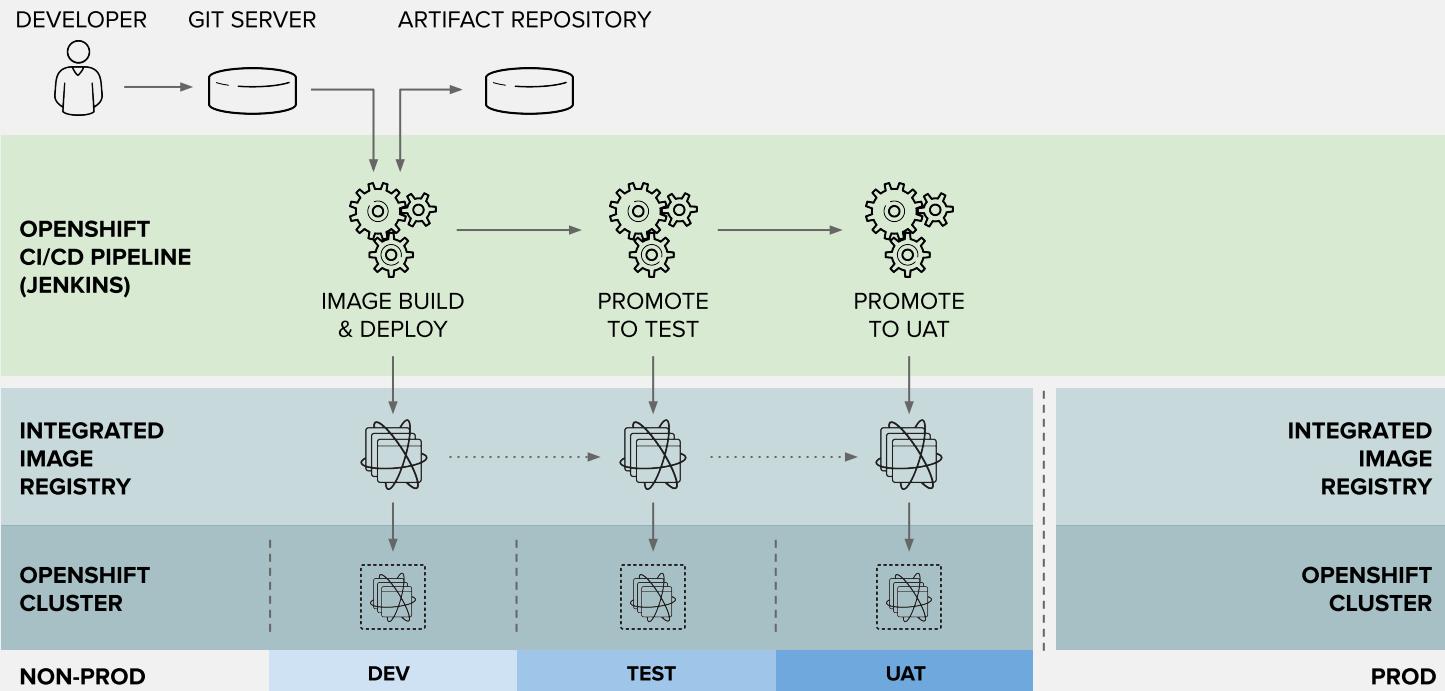
CONTINUOUS DELIVERY PIPELINE



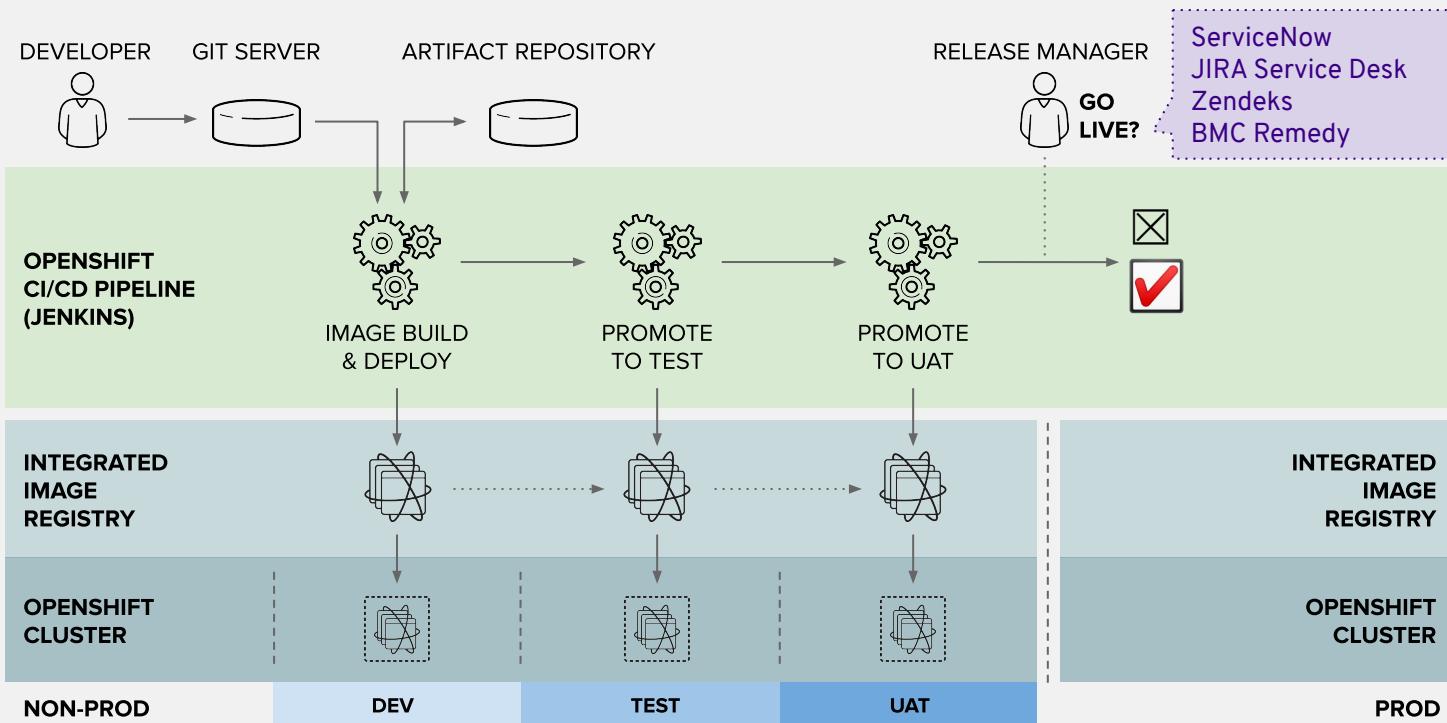
CONTINUOUS DELIVERY PIPELINE



CONTINUOUS DELIVERY PIPELINE



CONTINUOUS DELIVERY PIPELINE



CONTINUOUS DELIVERY PIPELINE

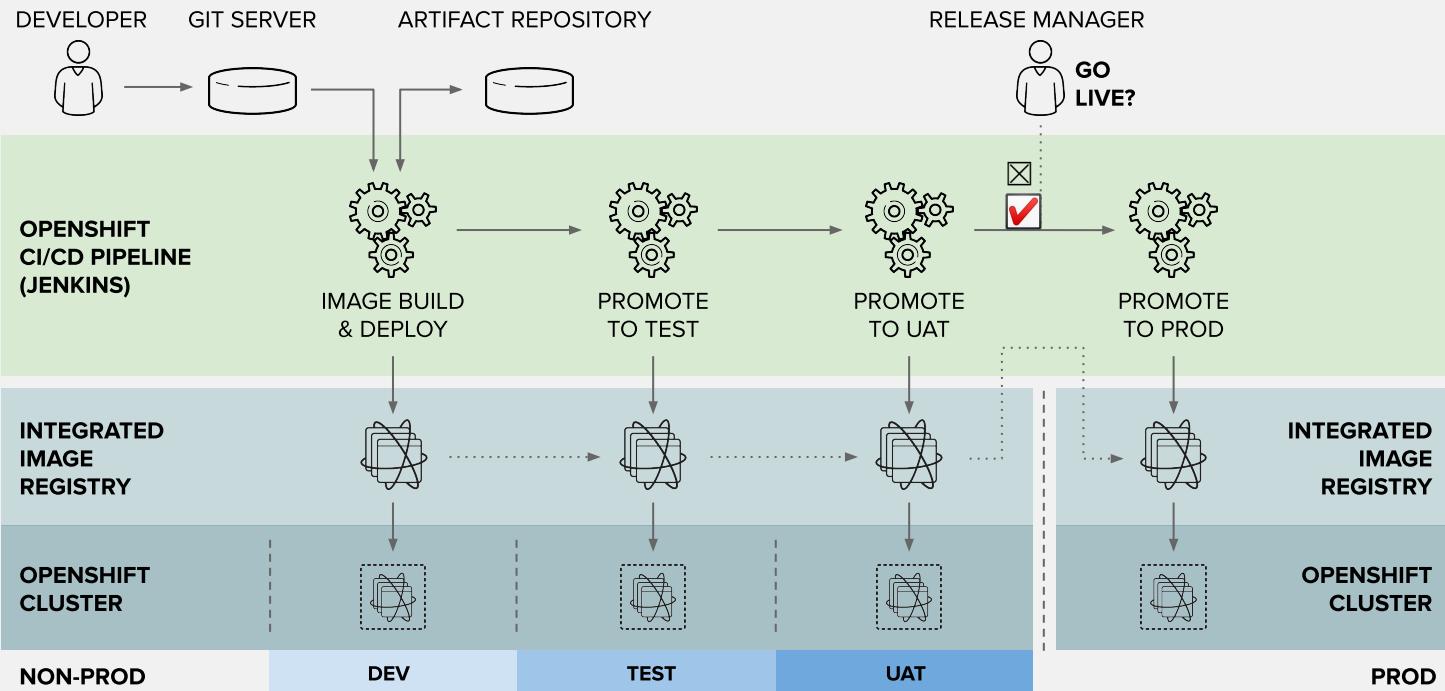
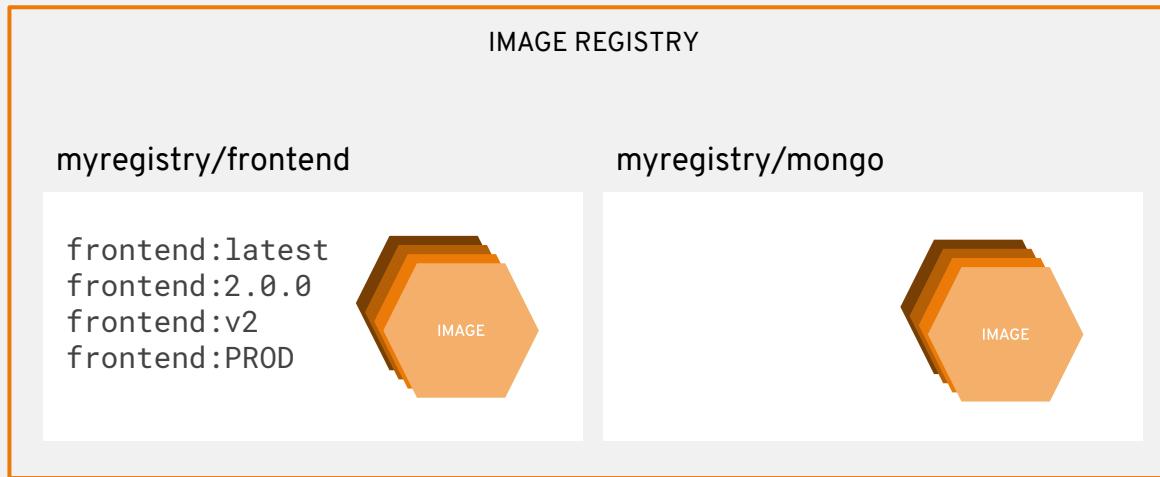


Image versioning

- docker:tag == git#tag
- latest == snapshot == master/develop (not a tag)



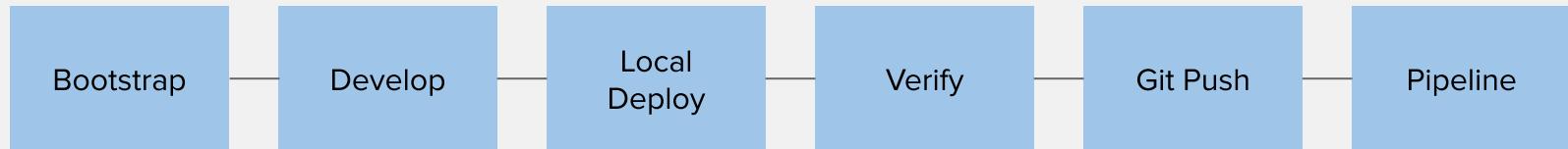
```
oc project dev-<vdiuser>
oc new-app https://bitbucket/mywebsite.git
oc get is,bc,dc,svc,route
oc get -o yaml --export is/httpd dc/httpd svc/httpd route/httpd >template.yml

oc new-project prod-<viduser>
oc create -f template.yml
oc import-image httpd:latest --from=registry-dev/dev-<vdiuser>/httpd:prod
oc tag prod/httpd:prod dev/httpd:latest
```

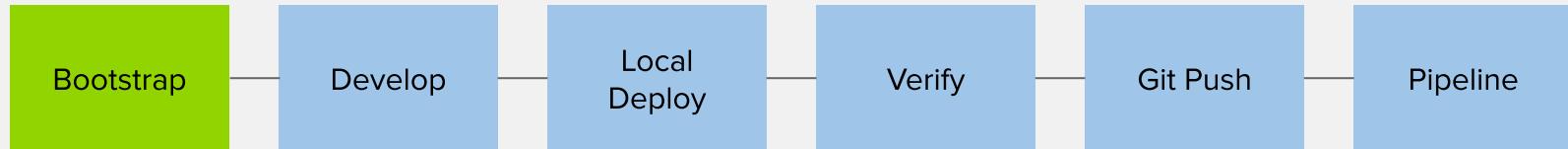
Local Development

Local Development

LOCAL DEVELOPMENT WORKFLOW



LOCAL DEVELOPMENT WORKFLOW



BOOTSTRAP

- Pick your programming language and application runtime of choice
- Create the project skeleton from scratch or use a generator such as
 - Maven archetypes
 - Quickstarts and Templates (oc new-app)
 - OpenShift Generator
 - Spring Initializr

OpenShift Explorer Console

javaeehol/jee-sample-build-1-build log

140/178 KB 33/33 KB
144/178 KB 33/33 KB

Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-filtering/1.0-beta-2/maven-filtering-1.0-beta-2.jar
148/178 KB
152/178 KB
156/178 KB
160/178 KB

New >
Show In >
Delete Resource...
Import Application...
Pod Log...
Port Forwarding...
Refresh
Properties

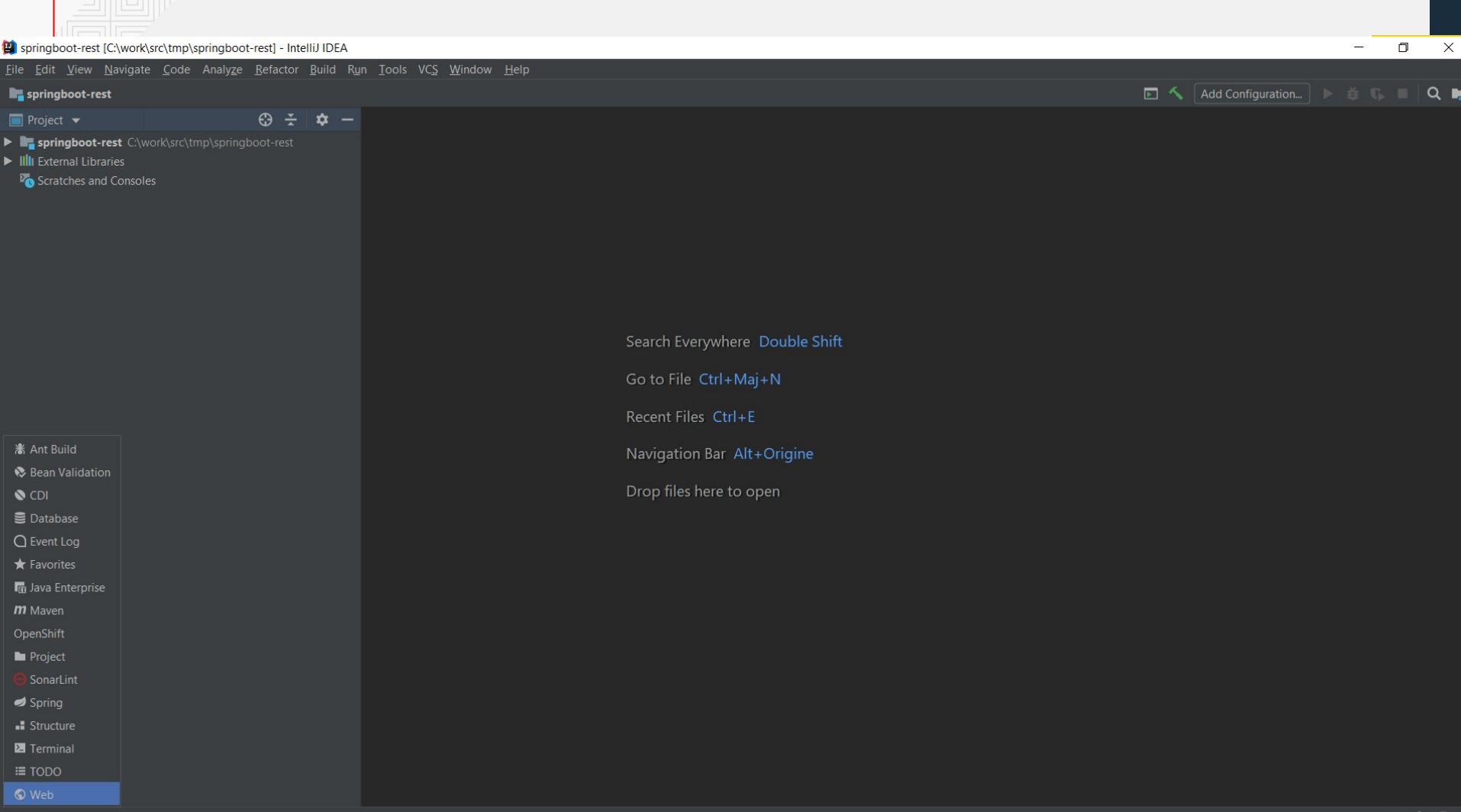
/repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.2/plexus-archiver-1.2.jar
bapp
ebapp [movieplex7] in [/opt/app-root/src/target/movieplex7-1.0-SNAPSHOT]
tar project
pp resources [/opt/app-root/src/src/main/webapp]

[INFO] Webapp assembled in [33 msecs]
[INFO] Building war: /opt/app-root/src/target/movieplex7-1.0-SNAPSHOT.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 20.037 s
[INFO] Finished at: 2015-12-03T12:39:26+00:00
[INFO] Final Memory: 17M/122M
[INFO] -----
[WARNING] The requested profile "openshift" could not be activated because it does not exist.
Copying built war files into /wildfly/standalone/deployments for later deployment...
Copying config files from project...
...done

I1203 12:39:39.696261 1 sti.go:213] Using provided push secret for pushing 172.30.236.154:5000
I1203 12:39:39.696301 1 sti.go:217] Pushing 172.30.236.154:5000/javaeehol/jee-sample:latest i
I1203 12:39:54.976489 1 sti.go:233] Successfully pushed 172.30.236.154:5000/javaeehol/jee-sam

Docker ... Projects Packages

demo swarm-sample





File Edit Selection View Go Debug Terminal Help

application.properties

```
1 package com.example;
2
3 import javax.ws.rs.GET;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.Produces;
6 import javax.ws.rs.core.MediaType;
7
8 @Path("/hello")
9 public class QuarkusDemoResource {
10
11     @GET
12     @Produces(MediaType.TEXT_PLAIN)
13     public String hello() {
14         return "hello";
15     }
16 }
```

QuarkusDemoResource.java

Index.html

pom.xml

NativeQuarkusDemoF

My Workspace

- WORKSPACE
- User Runtimes
 - git
 - New terminal
 - quarkus-builder
 - New terminal
 - compile quarkus:dev
 - package -Prnative
 - package
 - quarkus-runner
 - New terminal
 - start native
- Plugins

PREVIEW

<http://serverd7pwsck-theta-ldebts-server:1313/>

Your new Cloud-Native application is ready!

Congratulations, you have created a new Quarkus application.

Why do you see this?

This page is served by Quarkus. The source is in [src/main/resources/META-INF/resources/index.html](#).

/hello endpoint:

hello

Applica

GroupId: com.example

ArtifactId: hello-quarkus

Version: 1.0-SNAPSHOT

Quarkus Version: 0.14.0

Next steps

Setup your IDE

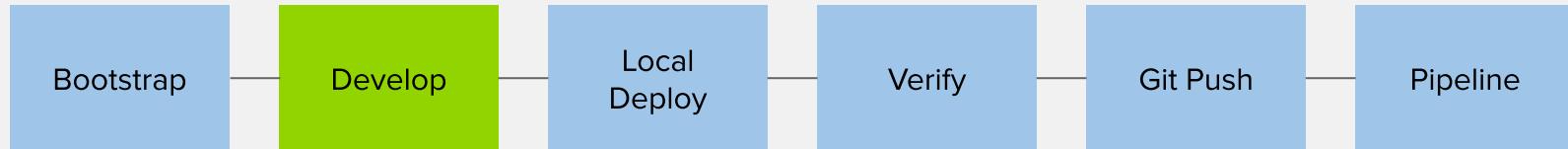
Getting started

Quarkus Web

master*+ 0 0 A 0 X 1 Previews

Ln 14, Col 24 LF Spaces: 4 Java

LOCAL DEVELOPMENT WORKFLOW



DEVELOP

- Pick your framework of choice such as Java EE, Spring, Ruby on Rails, Django, Express, ...
- Develop your application code using your editor or IDE of choice
- Build and test your application code locally using your build tools
- Create or generate OpenShift templates or Kubernetes objects

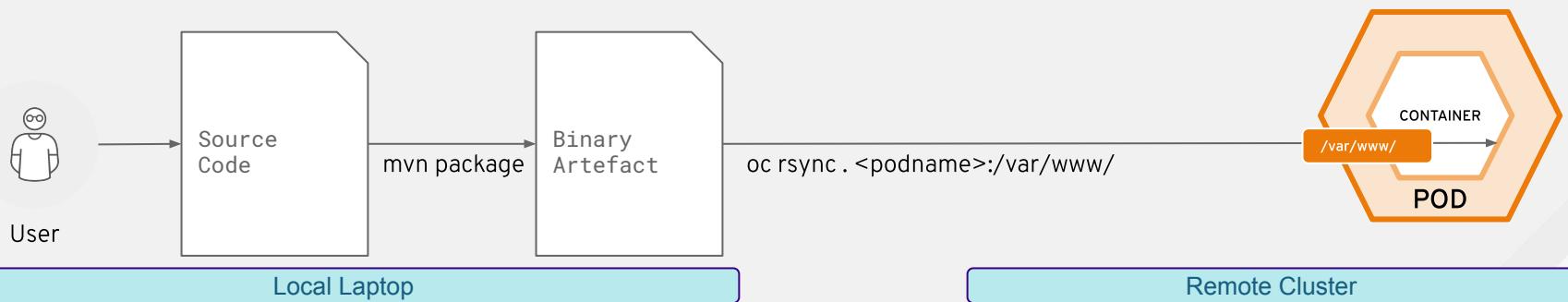
LOCAL DEVELOPMENT WORKFLOW

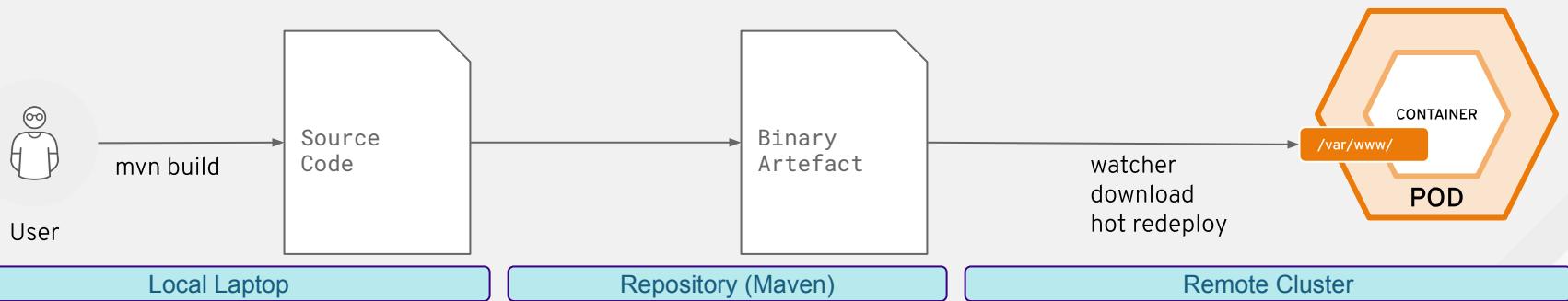
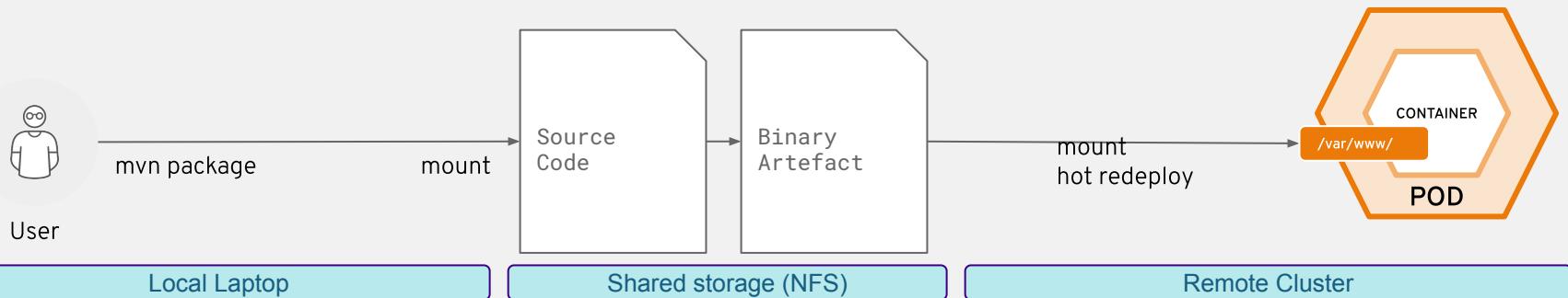


LOCAL DEPLOY

- Deploy your code on a local OpenShift cluster
 - Red Hat CodeReady
- Use binary deploy, maven or CLI rsync to push code or app binary directly into containers







```
oc project dev-<vdiuser>
```

```
git clone https://bitbucket/mywebsite.git
```

```
oc new-app .
```

```
oc get is,bc,dc,svc,route
```

```
oc start-build --from-dir=. --follow
```

```
mvn package
```

```
oc rsync . <pod-name>:/var/www/html/
```

```
yum install nfs
```

```
vi /etc/exports
```

```
oc set volume
```

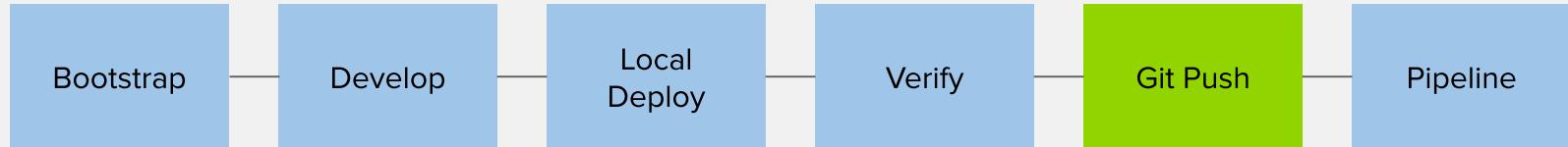
LOCAL DEVELOPMENT WORKFLOW



VERIFY

- Verify your code is working as expected
- Run any type of tests that are required with or without other components (database, etc)
- Based on the test results, change code, deploy, verify and repeat

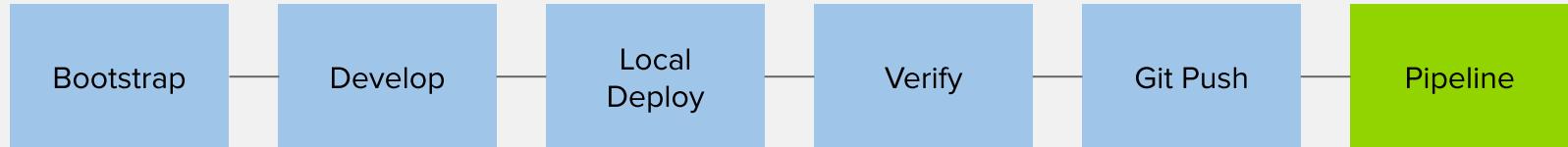
LOCAL DEVELOPMENT WORKFLOW



GIT PUSH

- Push the code and configuration to the Git repository
- If using Fork & Pull Request workflow, create a Pull Request
- If using code review workflow, participate in code review discussions

LOCAL DEVELOPMENT WORKFLOW



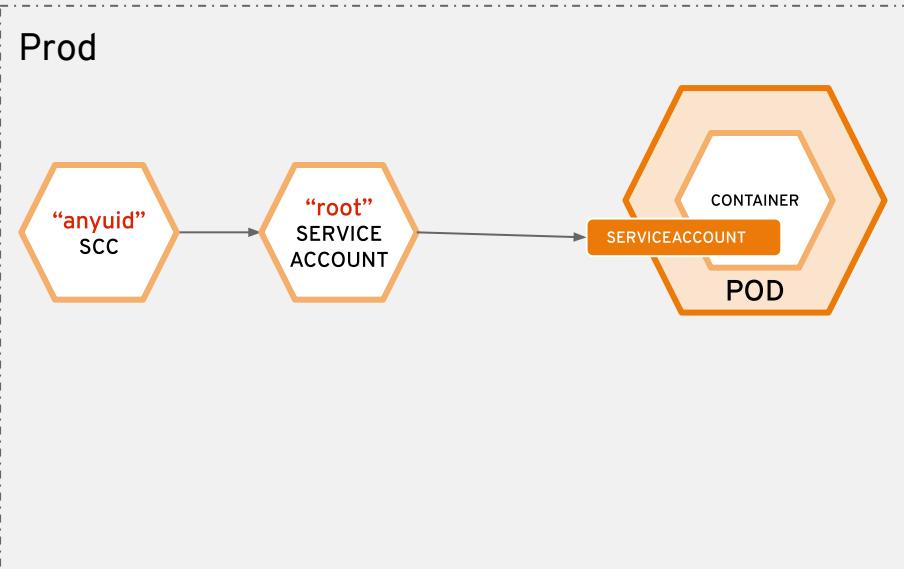
PIPELINE

- Pushing code to the Git repository triggers one or multiple deployment pipelines
- Design your pipelines based on your development workflow e.g. test the pull request
- Failure in the pipeline? Go back to the code and start again

Image building

Promote & Deploy into
Production
automatically

RunAsRoot

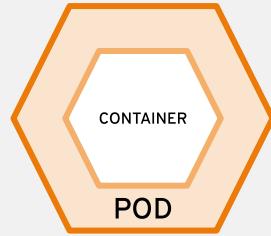


```
kind: DeploymentConfig
spec:
  template:
    spec:
      hostPID: true
      Containers:
        image: docker.io/httpd
        ports:
          - containerPort: 80
            protocol: TCP
        securityContext:
          privileged: true
          runAsUser: 0 #root
      serviceAccount: root
```

```
| > oc adm policy add-scc-to-user "anyuid" -z "root"
```

RunAsRange

Prod

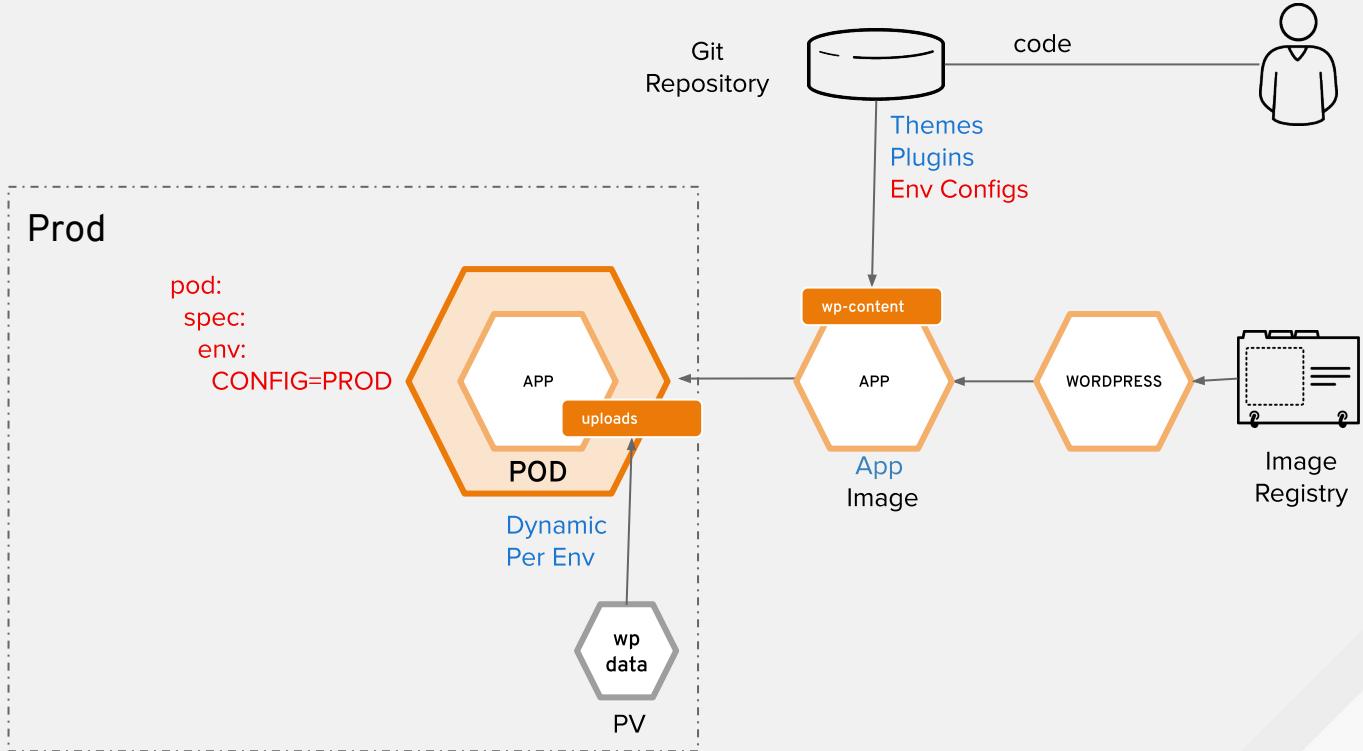


```
kind: DeploymentConfig
spec:
  template:
    spec:
      containers:
        image: redhat.io/httpd
        ports:
          - containerPort: 8080
            protocol: TCP
        securityContext:
          privileged: false
          runAsUser: random
      serviceAccount: default
```

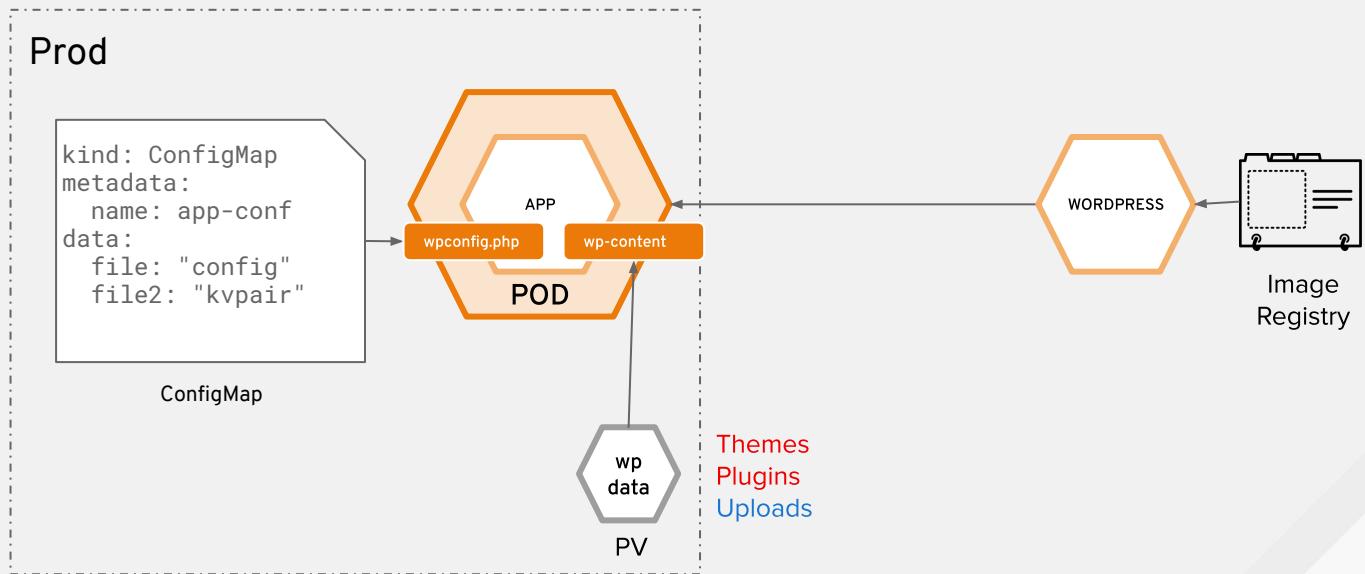
USER 1024000
RUN chmod 777 /var/www

```
| > oc adm policy add-scc-to-user "anyuid" -z "root"
```

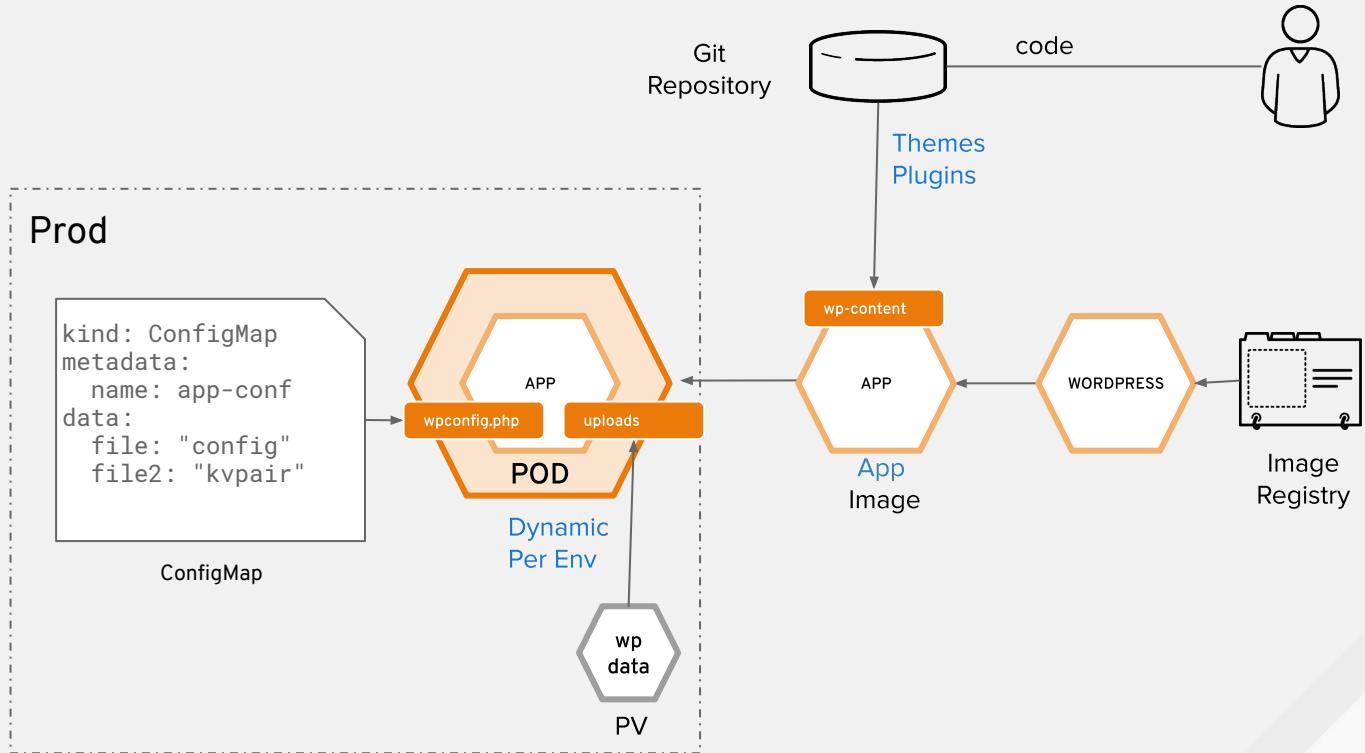
ENVIRONMENT Config



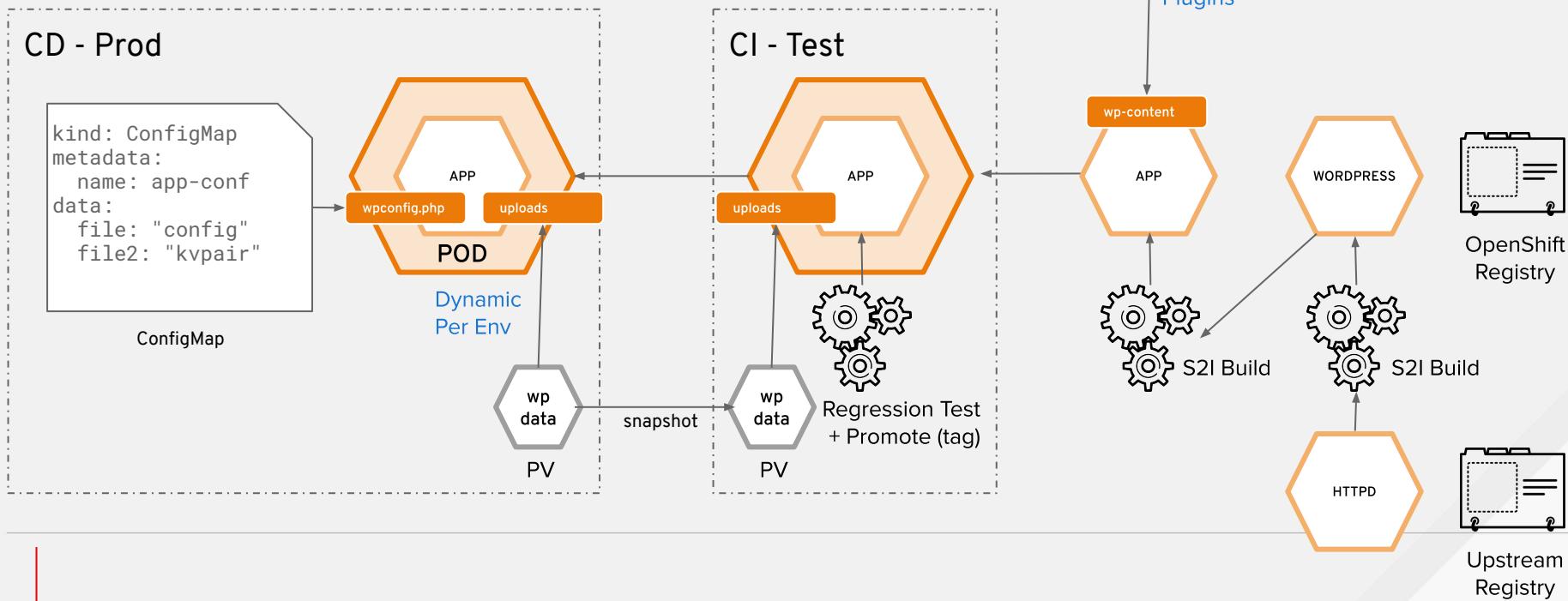
STATIC DATA



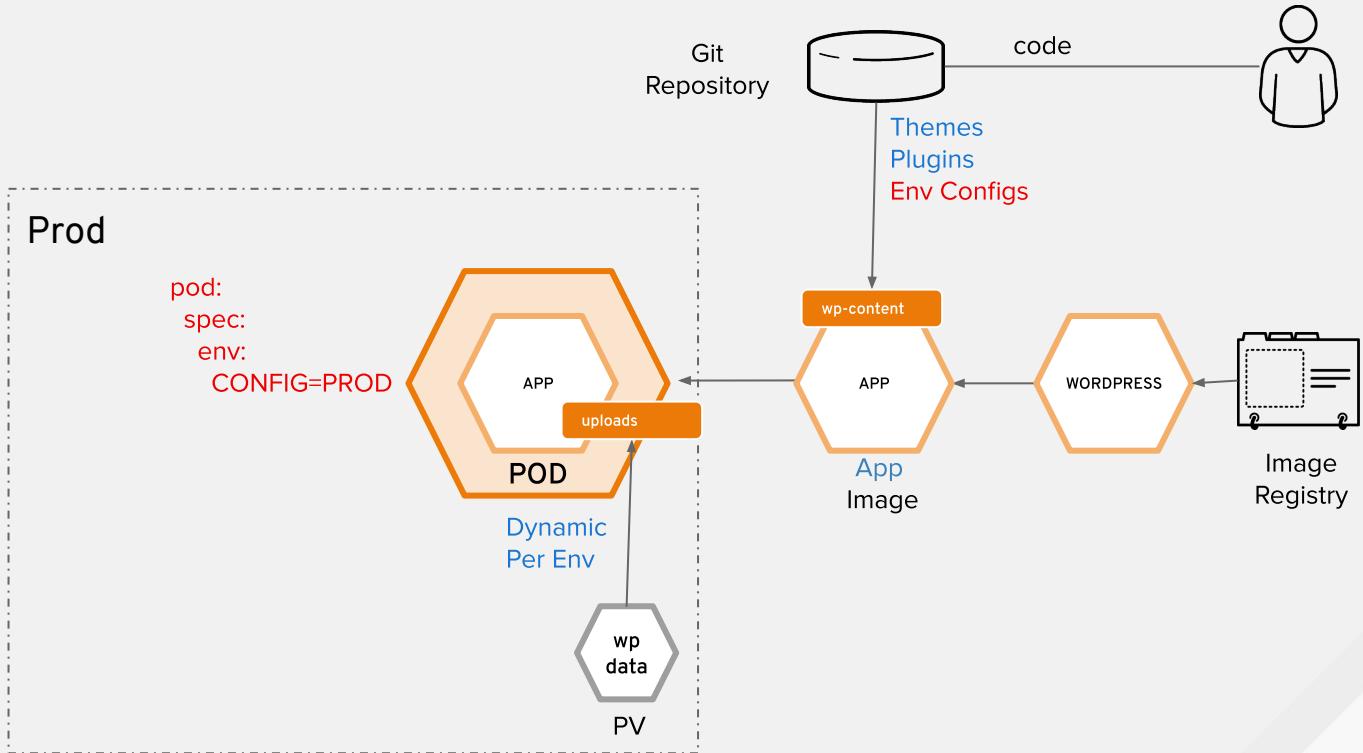
Code vs Data vs Config



LifeCycle Management of static content

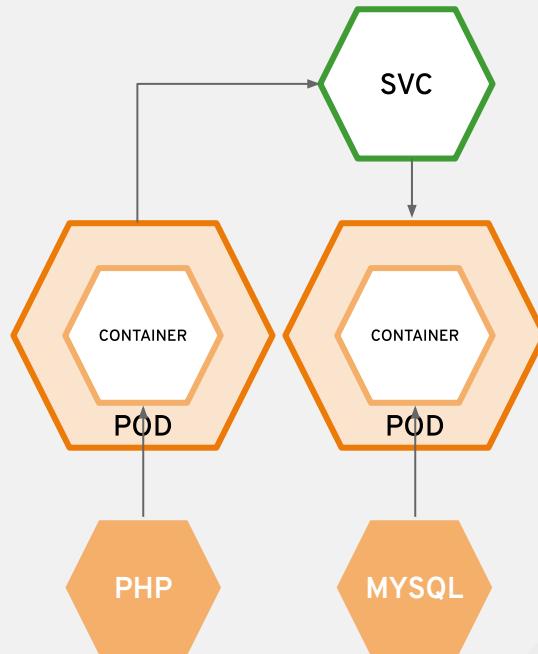
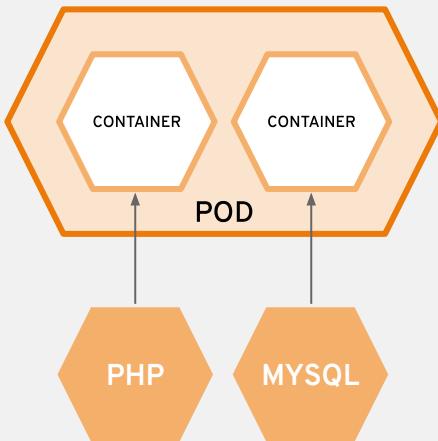
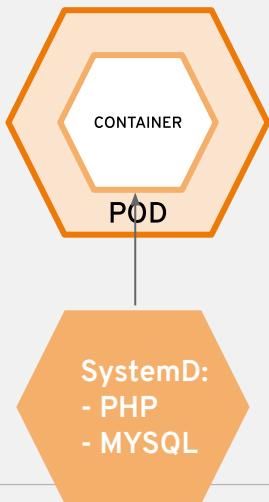


ENVIRONMENT Config



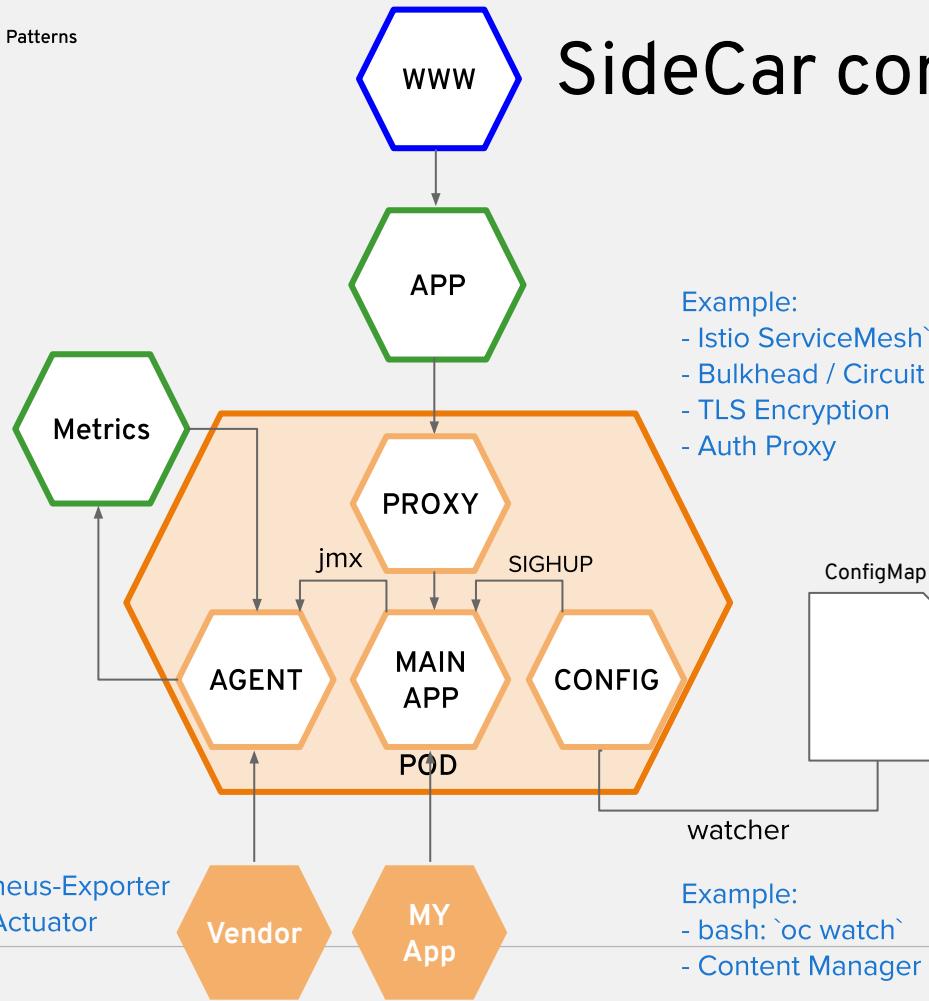
Atomic Unit

- Deployed and Scaled as One
- 1:1 relation
- LivenessProbe kills Pod



SideCar containers

- Same Namespace
- Tightly coupled
- Monitoring
- Config reload

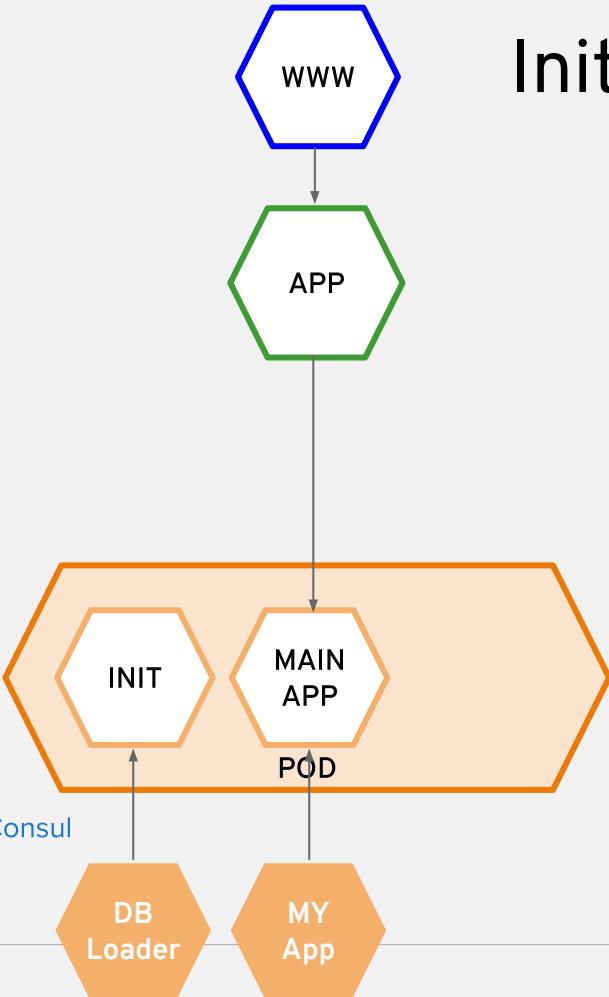


Init containers

- Must complete successful

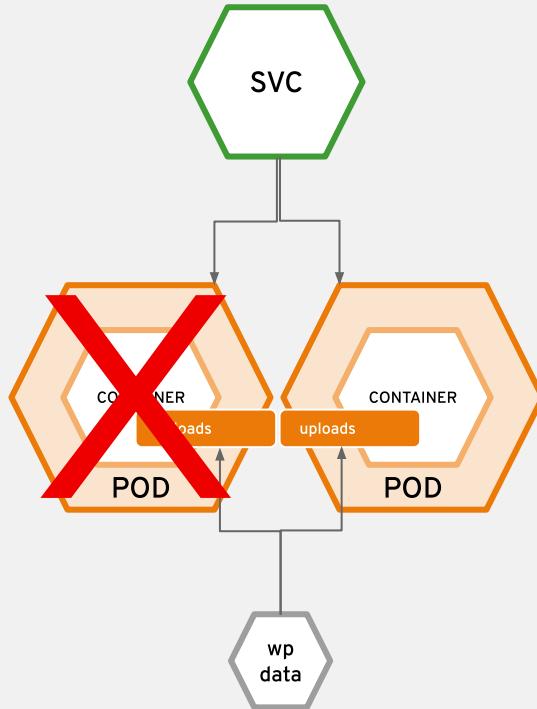
Example:

- Cluster Quorum
- Get node id
- Get Licence
- Fetch config from Zookeeper/Consul
- Liquibase
- Flyway



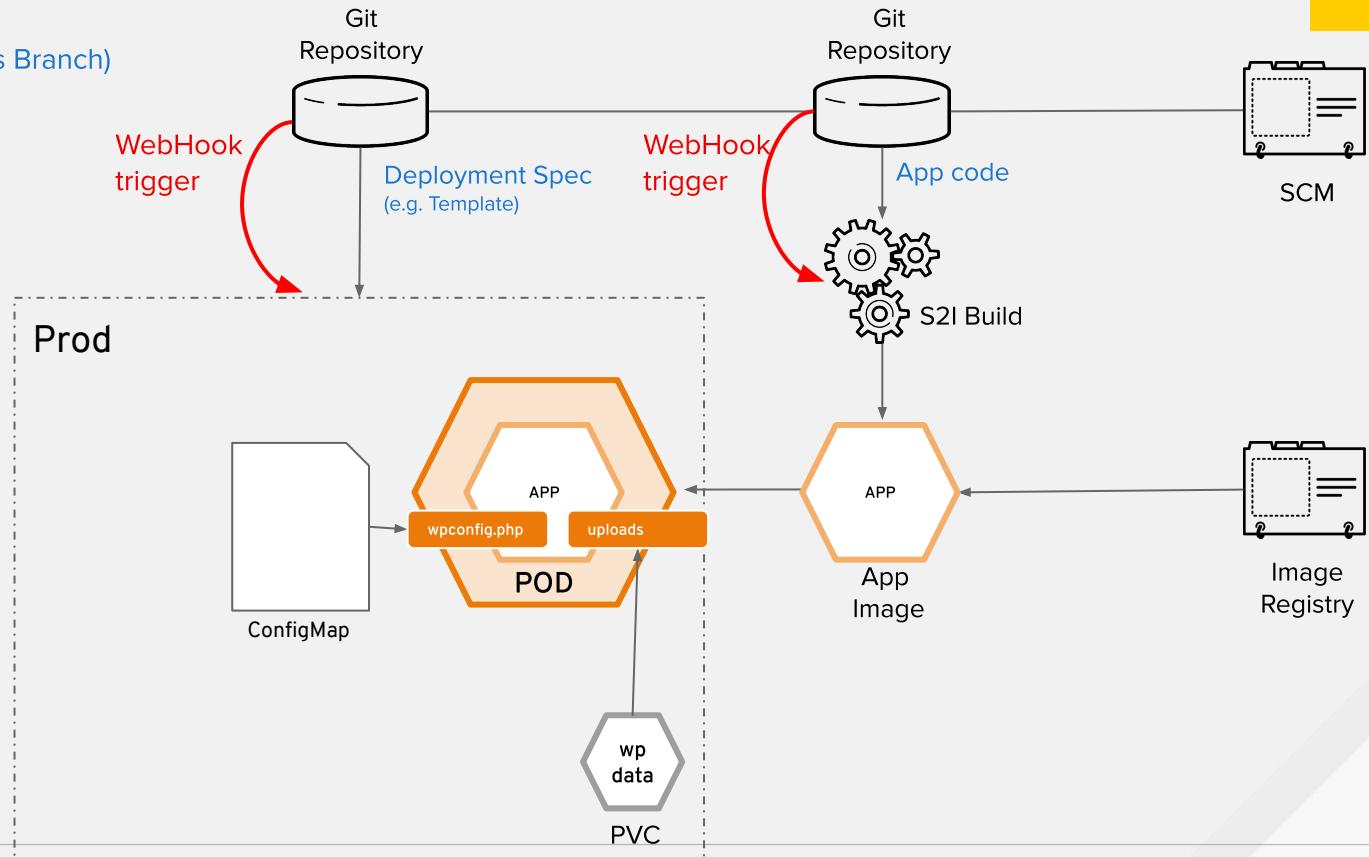
The Cloud-Native agreement

- Built for Failure
- Allow for Node Maintenance
- Stateless (Session data shared/externalized)
- (Difficult to Debug)
- VMware HA vs Container Failover



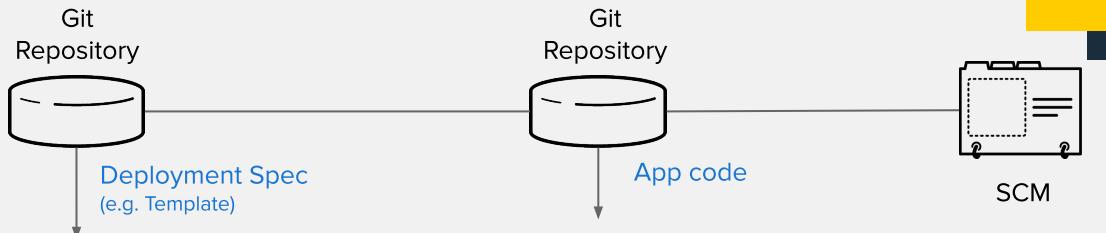
GitOps model

- Branching Strategy (Repo vs Branch)
- Template vs List



GitOps model

- No Branch Merging
- Versioned Stack Templates
- Prod-like envs



BRANCH PROD	prod.params: - stack_version: v1	/src/code.java /resources/k8s/template.json	BRANCH MASTER
BRANCH DEV	dev.params - stack_version: latest	/src/code.java /resources/k8s/template.json	BRANCH DEVELOP

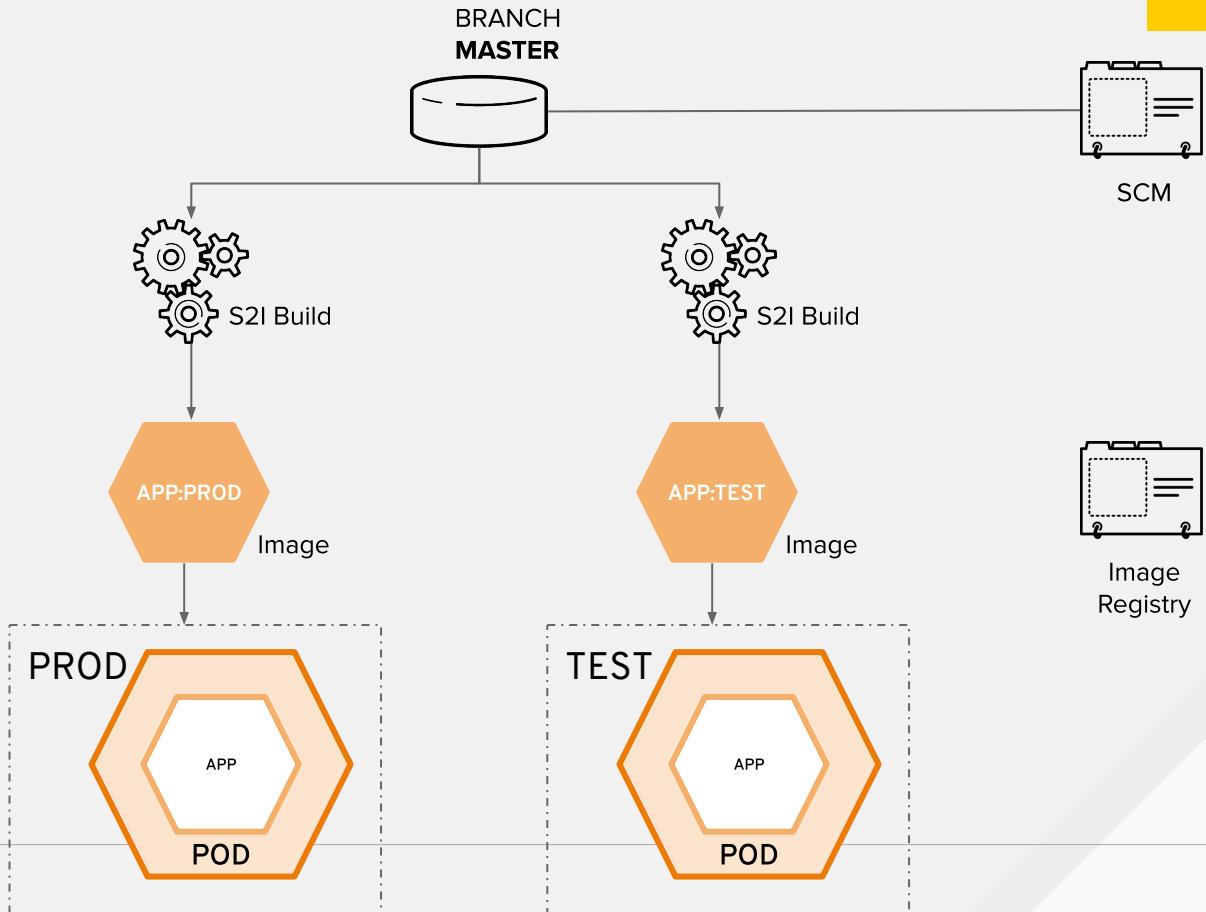
BRANCH PROD	/resources/k8s/list.json - image_version: v1	/src/code.java /resources/k8s/template.json	BRANCH MASTER
BRANCH DEV	/resources/k8s/list.json - image_version: latest	/src/code.java /resources/k8s/template.json	BRANCH DEVELOP



Service Catalog

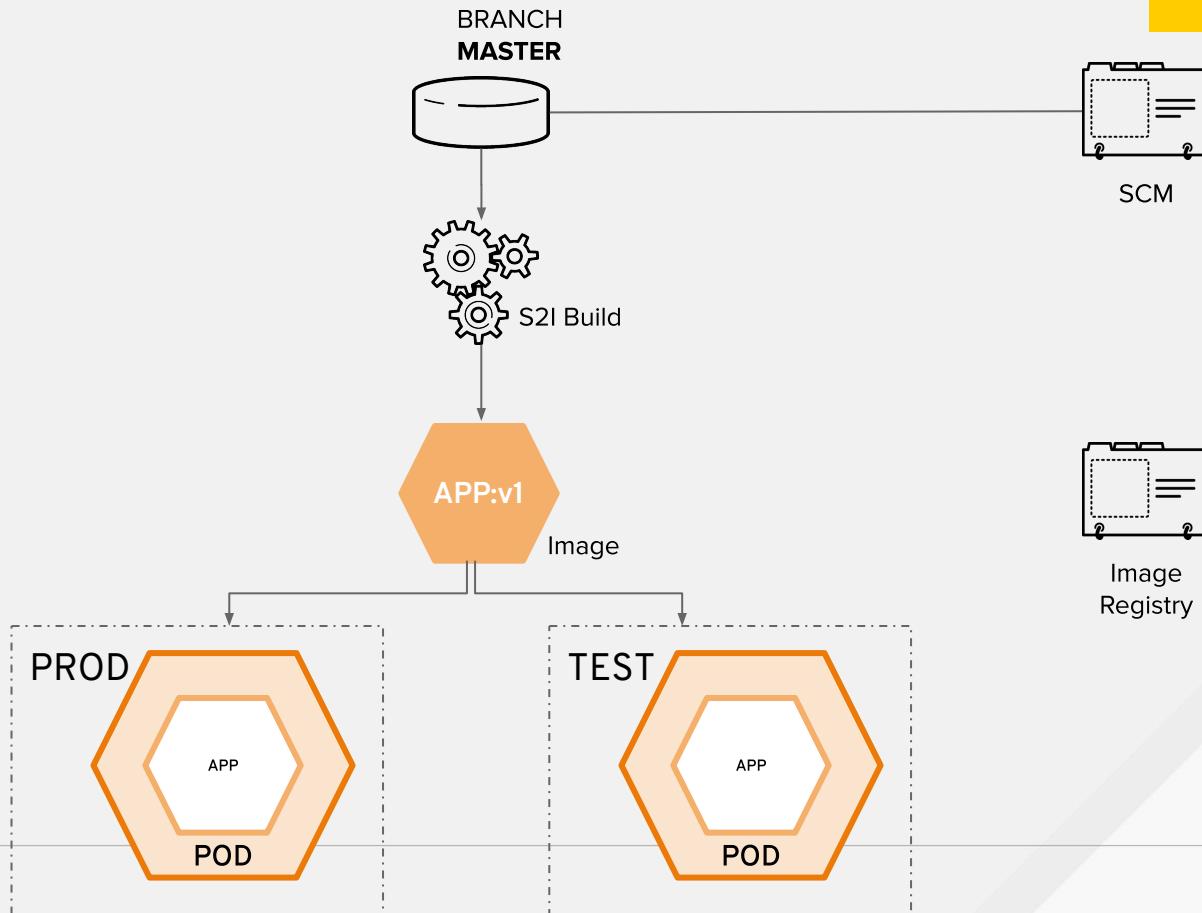
BUILD EVERYWHERE

- Branching Strategy (Repo vs Branch)
- Template vs List



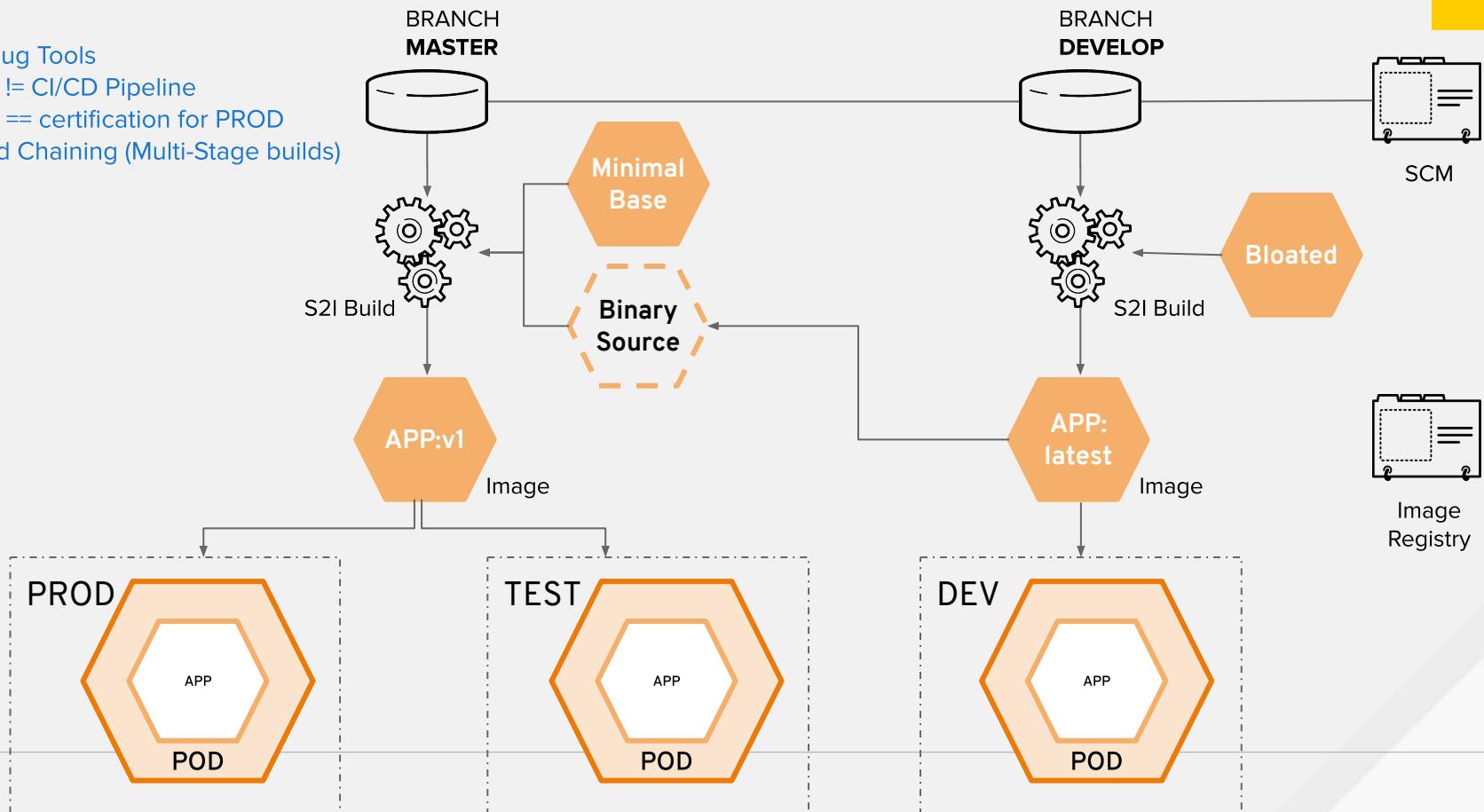
BUILD ONCE, DEPLOY MANY

- Same Image
- Tested / Certified



BUILD ONCE, DEPLOY MANY

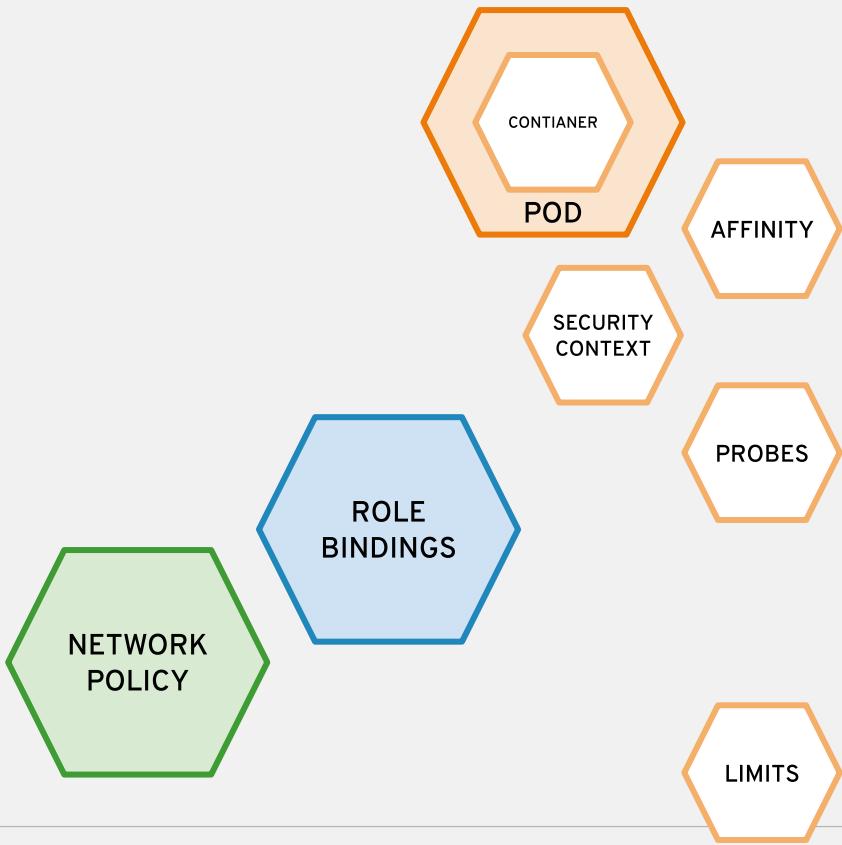
- Debug Tools
- Dev != CI/CD Pipeline
- Test == certification for PROD
- Build Chaining (Multi-Stage builds)



BUILD CHAINING

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: image-build
spec:
  output:
    to:
      kind: ImageStreamTag
      name: image-build:latest
  source:
    dockerfile: |-
      FROM jee-runtime:latest
      COPY ROOT.war /deployments/ROOT.war
  images:
    - from:
        kind: ImageStreamTag
        name: artifact-image:latest
    paths:
      - sourcePath: /wildfly/standalone/deployments/ROOT.war
        destinationDir: "."
  strategy:
    dockerStrategy:
      from:
        kind: ImageStreamTag
        name: jee-runtime:latest
  triggers:
    - imageChange: {}
      type: ImageChange
```

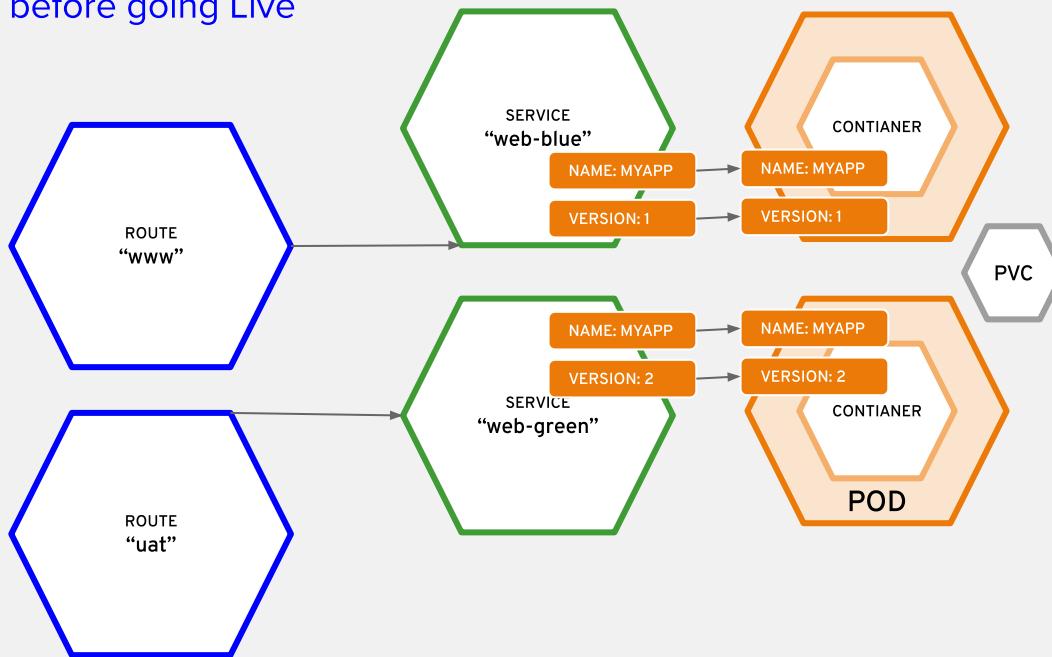
DEPLOY CHECKLIST



```
apiVersion: v1
kind: DeploymentConfig
metadata:
  name: httpd
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - topologyKey: kubernetes.io/hostname
  containers:
  - name: httpd
    image: docker.io/library/httpd
    livenessProbe:
      tcpSocket:
        port: 8080
    readinessProbe:
      httpGet:
        path: /healthz
        port: 8080
        scheme: HTTP
    resources:
      limits:
        cpu: 200m
        memory: 512Mi
      requests:
        cpu: 100m
```

Blue / Green update strategy

- Test before going Live



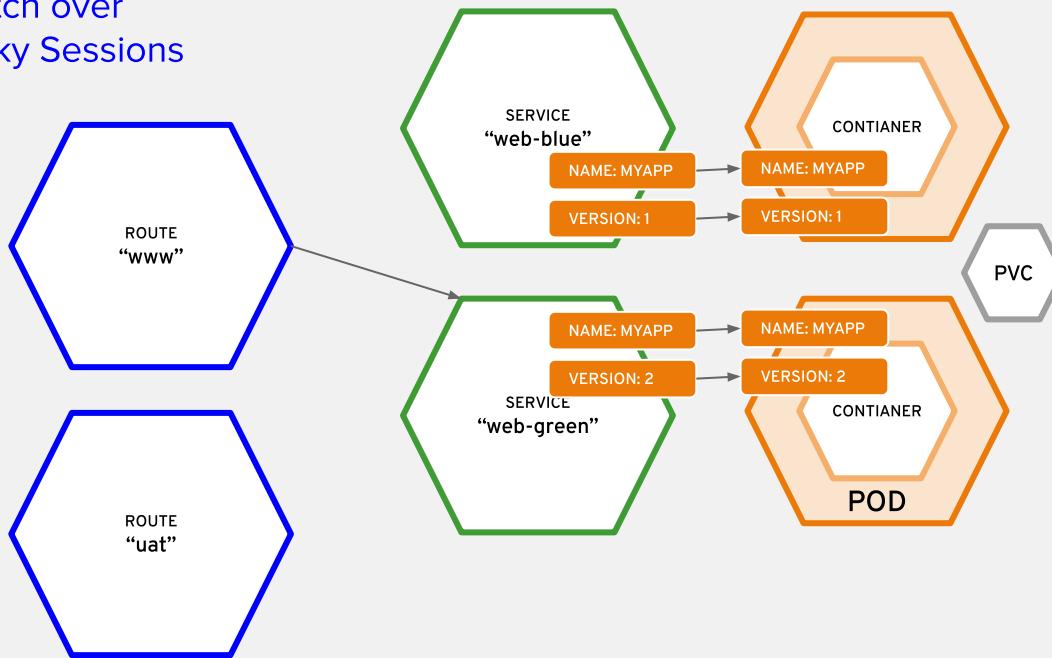
```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: www
spec:
  host: www.belastingdienst.nl
  to:
    kind: Service
    name: web-blue
    weight: 100
```

```
apiVersion: v1
kind: Service
metadata:
  name: web-blue
spec:
  selector:
    name: myapp
    version: 1
```

```
> oc patch route/www -p '{"spec":{"to":{"name":"web-green"}}}'
> oc patch svc/web-blue -p '{"spec":{"selector":{"version":"3"}}}'
```

Blue / Green update strategy

- Switch over
- Sticky Sessions



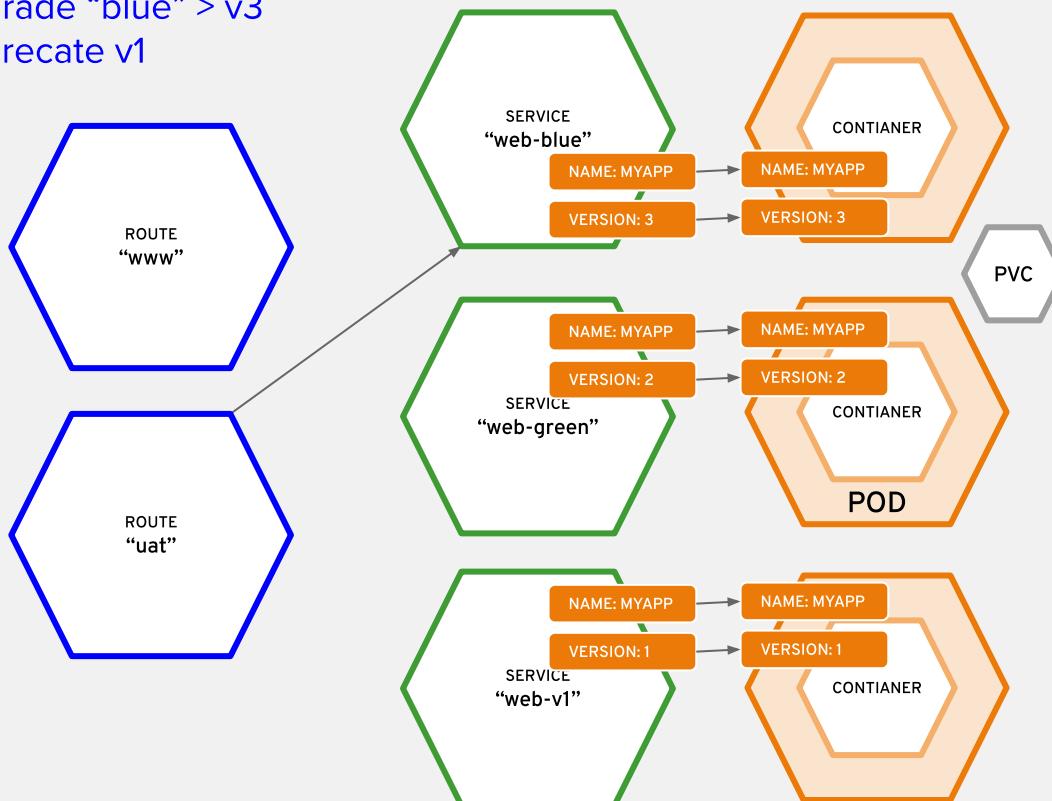
```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: www
spec:
  host: www.belastingdienst.nl
  to:
    kind: Service
    name: web-green
    weight: 100
```

```
apiVersion: v1
kind: Service
metadata:
  name: web-blue
spec:
  selector:
    name: myapp
    version: 1
```

```
> oc patch route/www -p '{"spec":{"to":{"name":"web-green"}}}'
> oc patch svc/web-blue -p '{"spec":{"selector":{"version":"3"}}}'
```

Blue / Green update strategy

- Upgrade “blue” > v3
- Deprecate v1



```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: uat
spec:
  host: uat.belastingdienst.nl
  to:
    kind: Service
    name: web-blue
    weight: 100
  
```

```

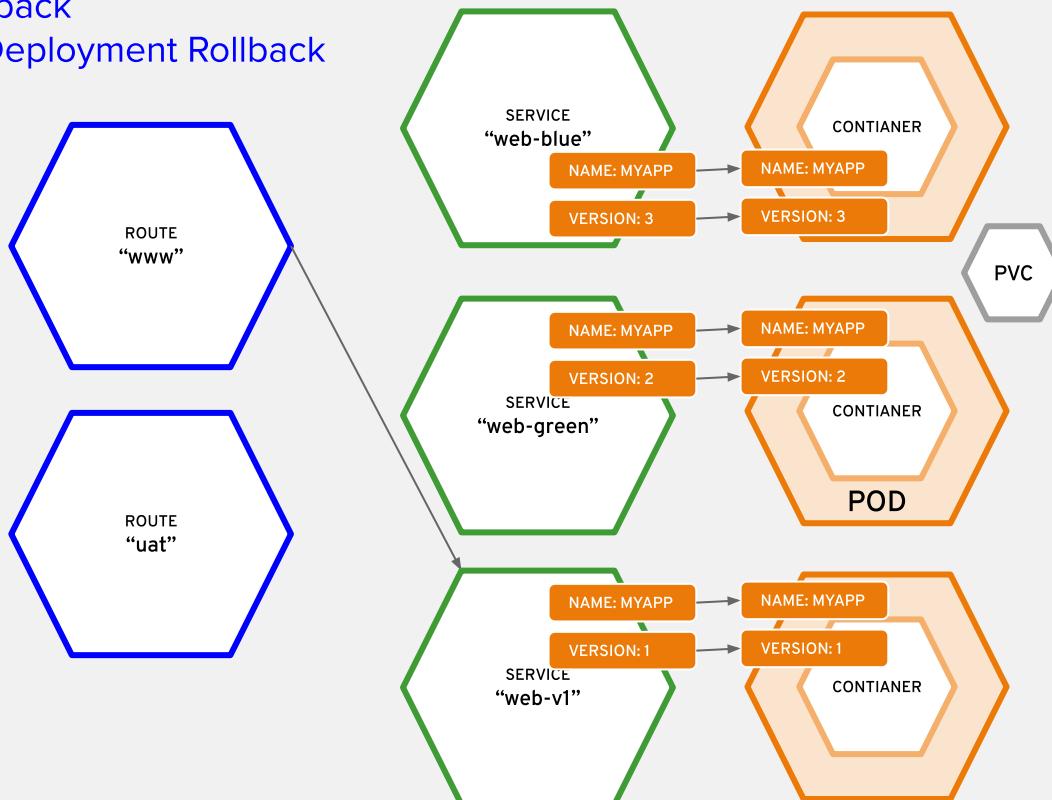
apiVersion: v1
kind: Service
metadata:
  name: web-blue
spec:
  selector:
    name: myapp
    version: 3
  
```

```

> oc patch route/www -p '{"spec":{"to":{"name":"web-green"}}}'
> oc patch svc/web-blue -p '{"spec":{"selector":{"version":3}}}'
  
```

Blue / Green update strategy

- Rollback
- vs Deployment Rollback



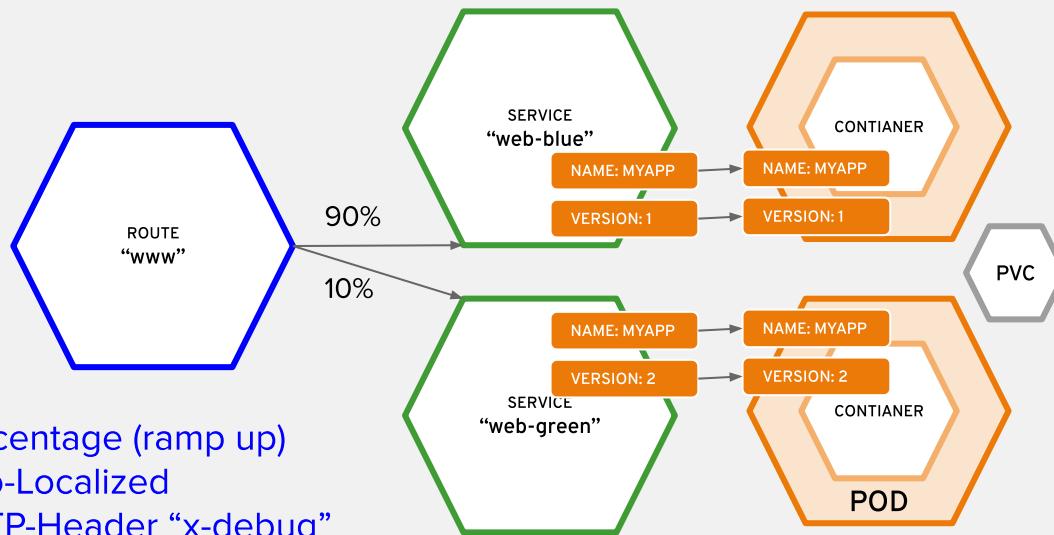
```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: uat
spec:
  host: uat.belastingdienst.nl
  to:
    kind: Service
    name: web-blue
    weight: 100
  
```

```

apiVersion: v1
kind: Service
metadata:
  name: web-blue
spec:
  selector:
    name: myapp
    version: 3
  
```

A / B update strategy



- Percentage (ramp up)
- Geo-Localized
- HTTP-Header "x-debug"
- Feedback button enabled

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: uat
spec:
  host: uat.belastingdienst.nl
  to:
    kind: Service
    name: web-blue
    weight: 90
  alternativeBackends:
    kind: Service
    name: web-green
    weight: 10
```

```
apiVersion: v1
kind: Service
metadata:
  name: web-blue
spec:
  selector:
    name: myapp
    version: 1
```

STAY K8S COMPATIBLE

	OpenShift	Vanilla Kubernetes	Cloud
Deploy	DeploymentConfig Deployment	Deployment	-
Build	BuildConfig	-	Google Tekton AWS CodeBuild Azure Pipelines
Ingress	Route Ingress	Ingress	ELB Service
Security	SCC PSP	PSP	-
Network	NetworkPolicy	NetworkPolicy	SecurityGroups
Storage	PVC	PVC	PVC
Monitoring	EFK / Prometheus	-	CloudWatch

Reference Architectures

See what is
recommended from
the vendors

REFERENCE ARCHITECTURES

OpenShift on VMware vCenter
<https://access.redhat.com/articles/2745171>

OpenShift on Red Hat OpenStack Platform
<https://access.redhat.com/articles/2743631>

OpenShift on Amazon Web Services
<https://access.redhat.com/articles/2623521>

OpenShift on Google Cloud Platform
<https://access.redhat.com/articles/2751521>

OpenShift on Microsoft Azure
<https://access.redhat.com/articles/3030691>

Deploying an OpenShift Distributed Architecture
<https://access.redhat.com/articles/1609803>

OpenShift Architecture and Deployment Guide
<https://access.redhat.com/articles/1755133>

OpenShift Scaling, Performance, and Capacity Planning
<https://access.redhat.com/articles/2191731>

Application Release Strategies with OpenShift
<https://access.redhat.com/articles/2897391>

Building Polyglot Microservices on OpenShift
<https://access.redhat.com/articles/2893381>

Building JBoss EAP 6 Microservices on OpenShift
<https://access.redhat.com/articles/2094731>

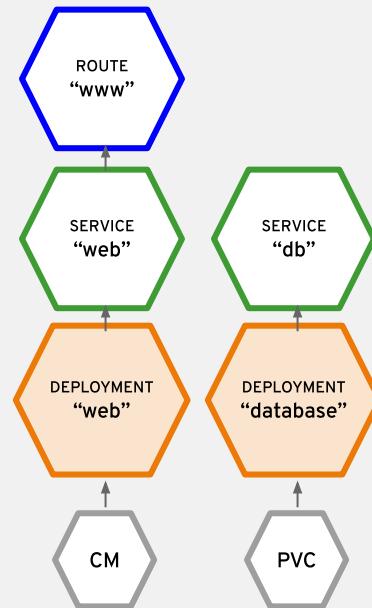
Building JBoss EAP 7 Microservices on OpenShift
<https://access.redhat.com/articles/2407801>

Business Process Management with JBoss BPM Server on OpenShift
<https://access.redhat.com/articles/2893421>

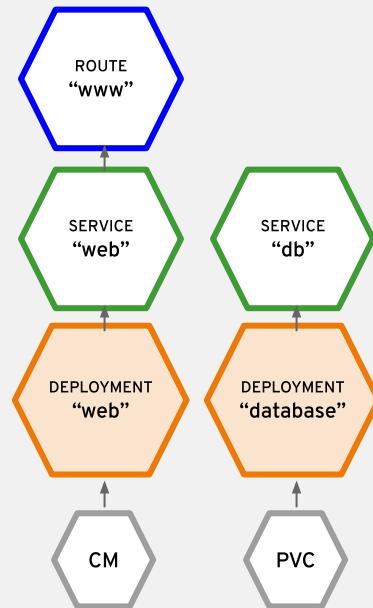
Build and Deployment of Java Applications on OpenShift
<https://access.redhat.com/articles/3016691>

JFrog Artifactory on OpenShift Container Platform
<https://access.redhat.com/articles/3049611>

JEE Java Stack



Spring Cloud - Java Stack



Application Onboarding

Promote & Deploy into
Production
automatically

ALL RED HAT PRODUCTS CONTAINERIZED

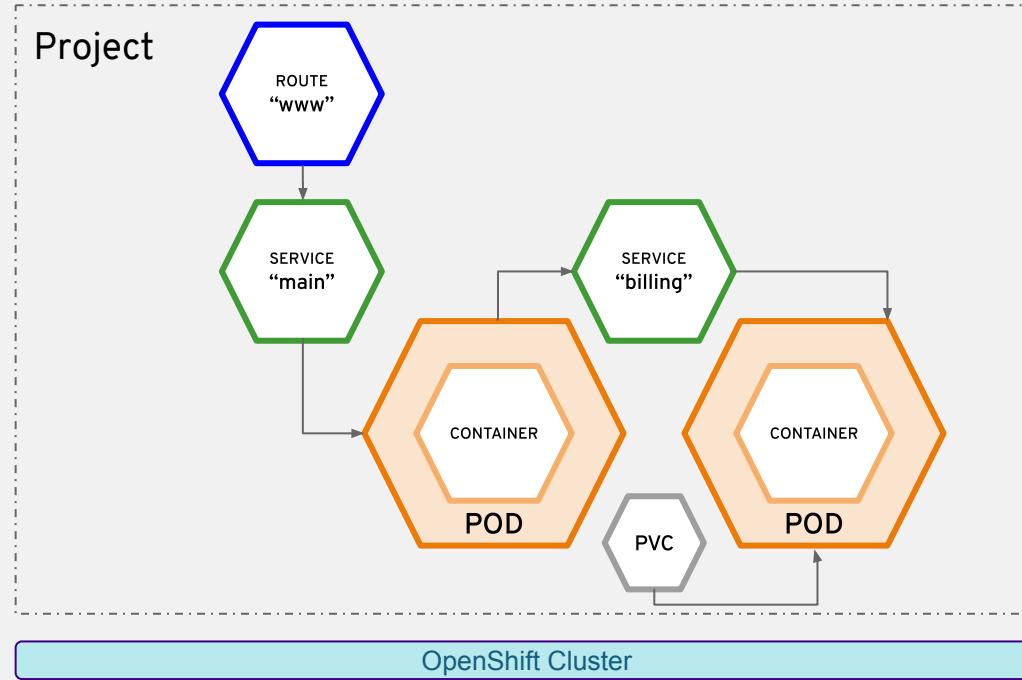
The screenshot shows the OpenShift Container Platform developer catalog interface. At the top, there's a navigation bar with 'OPENSHIFT CONTAINER PLATFORM' and 'Kubernetes Resources'. Below it, a large title 'ALL RED HAT PRODUCTS CONTAINERIZED' is displayed. The main area is titled 'Browse Catalog' and features a grid of 18 items, each representing a containerized application stack. The items are arranged in four rows:

- Row 1:** Apache HTTP Server (httpd), CakePHP + MySQL (Persistent), Dancer + MySQL (Persistent), Django + PostgreSQL (Persistent), Jenkins (Ephemeral)
- Row 2:** Jenkins (Persistent), MariaDB (Persistent), MongoDB (Persistent), MySQL (Persistent), Node.js
- Row 3:** Node.js + MongoDB (Persistent), Perl, PHP, Pipeline Build Example, PostgreSQL (Persistent)
- Row 4:** Python, Rails + PostgreSQL (Persistent), Ruby

On the right side of the catalog grid, there's a sidebar titled 'My Projects' which shows '1 of 1 Projects' named 'My Project'. It also includes a 'Getting Started' section with links to 'Documentation', 'Interactive Learning Portal', 'Container Development Kit', 'YouTube', and 'Blog'.

Greenfield / S2I

- Lots of Base Images
- Easy to recompile
- Java == Java
- Write your own



```
> yum install openshift-node
```

Image selection

80%

Existing Builder
Image?

```
oc new-app <git-url>
```



- Red Hat products
- Common languages

10%

Existing Base
Image?

```
Dockerfile  
ADD .s2i/assemble
```



- Community

10%

Build from Scratch

```
Dockerfile  
RUN yum install jdk weblogic  
ADD /entrypoint.sh  
ENV param1=value1  
CMD /entrypoint.sh
```



- COTS Software

Idempotency & State

85%

DeploymentConfig

```
oc new-app <git-url>
```

10%

StatefulSet

```
Dockerfile  
ADD .s2i/assemble
```

2%

DaemonSet

```
Dockerfile  
RUN yum install jdk weblogic  
ADD /entrypoint.sh  
ENV param1=value1  
CMD /entrypoint.sh
```

2%

CronJob

Lorem ipsum dolor sit amet
adipiscing. Donec risus
dolor, porta venenatis neque
pharetra luctus felis vel
tellus nec felis.

- Donec risus dolor porta
- Pharetra luctus felis
- Proin vel tellus in felis
- Molestie nec amet cum

- Donec risus dolor porta
- Pharetra luctus felis
- Proin vel tellus in felis
- Molestie nec amet cum

- Donec risus dolor porta
- Pharetra luctus felis
- Proin vel tellus in felis
- Molestie nec amet cum

- Donec risus dolor porta
- Pharetra luctus felis
- Proin vel tellus in felis
- Molestie nec amet cum

Entrypoint

15%

InitContainer

Database versioning
Fetch Config (Zookeeper)

10%

SideCar Container

Peer discovery
Reload Config

90%

Configuration

Parse Parameters (ENV)
Generate Config

0%

...

Lorem ipsum dolor sit amet
adipiscing. Donec risus
dolor, porta venenatis neque
pharetra luctus felis vel
tellus nec felis.

- Donec risus dolor porta
- Pharetra luctus felis
- Proin vel tellus in felis
- Molestie nec amet cum

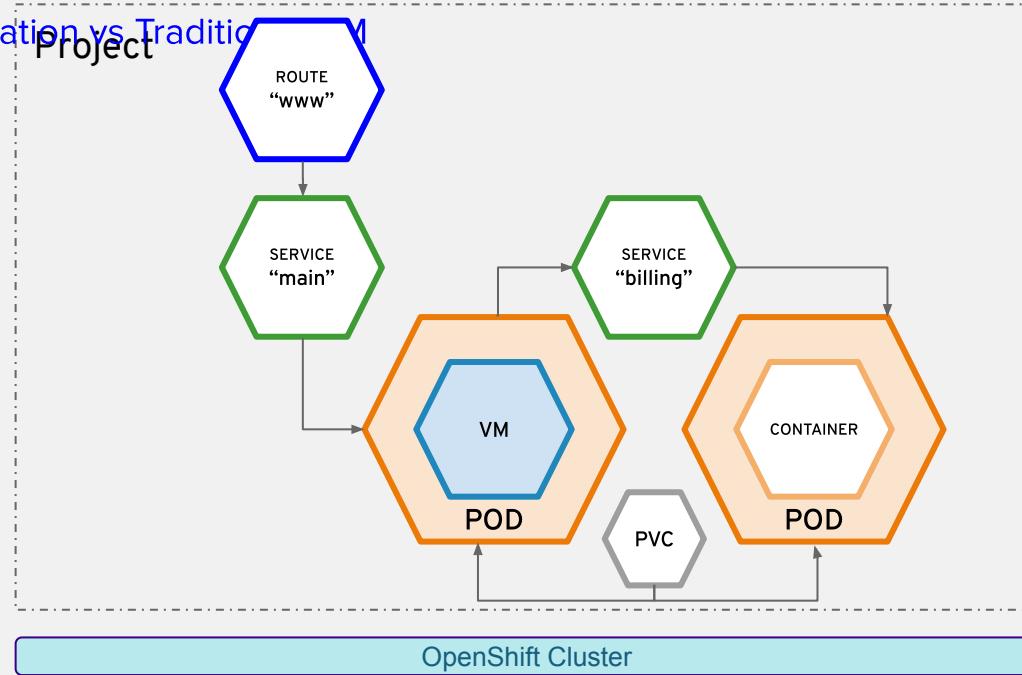
- Donec risus dolor porta
- Pharetra luctus felis
- Proin vel tellus in felis
- Molestie nec amet cum

- Donec risus dolor porta
- Pharetra luctus felis
- Proin vel tellus in felis
- Molestie nec amet cum

- Donec risus dolor porta
- Pharetra luctus felis
- Proin vel tellus in felis
- Molestie nec amet cum

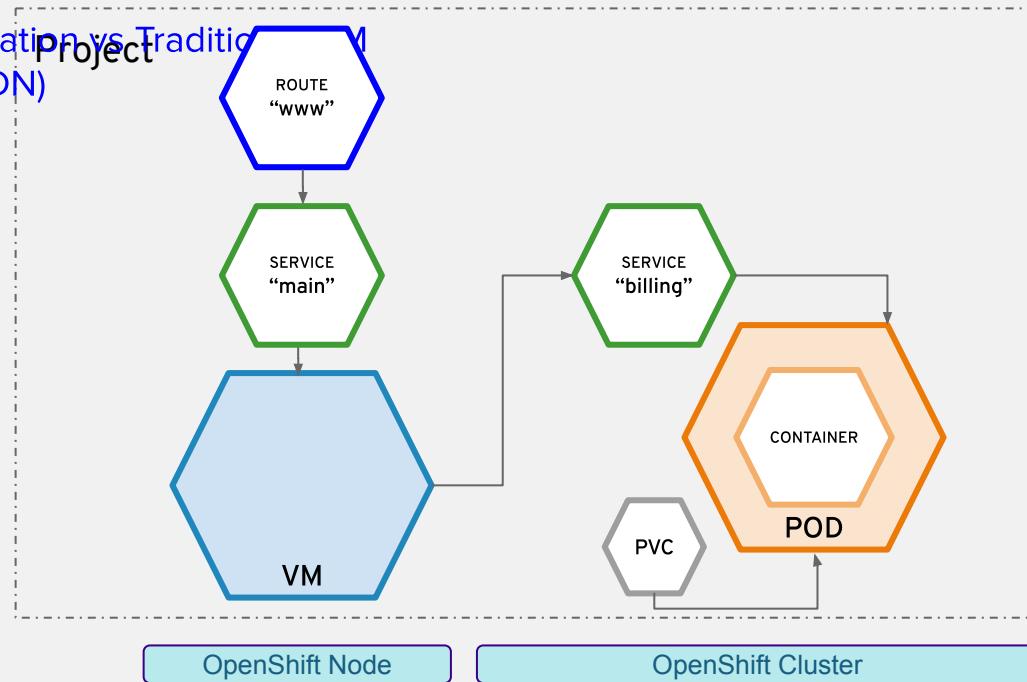
Lift & Shift

- Container-Native Virtualization vs Traditional Project
- Service Discovery (DNS)
- KVM based
- Strangler pattern



Lift & Shift

- Container-Native Virtualization vs Traditional Project
- Ramp Up node (VPN-2-SDN)
- Service Discovery (DNS)



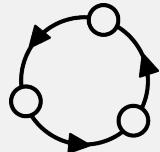
```
> yum install openshift-node
```

Appendix: OpenShift vs. “Vanilla” Kubernetes Myths



THE VANILLA KUBERNETES MYTHS

Why IT shops swoon over “Vanilla” Kubernetes and its perceived value



Ultimate portability across Kubernetes Clusters



No “vendor lock-in”



Always on latest version

THE VANILLA KUBERNETES REALITY

“Vanilla” Kubernetes is not really vanilla at all



Every vendor configures their Kubernetes distribution differently



Every vendor operates Kubernetes differently; this matters



No vendor is in lockstep with the latest upstream

* <https://medium.com/@jzelinskie/youre-not-running-vanilla-kubernetes-2f2359666bf9>

KUBERNETES CONFORMANCE

Interoperability at the API



"The new Certified Kubernetes Conformance Program gives enterprise organizations the confidence that workloads that run on any Certified Kubernetes Distribution or Platform will work correctly on any other version," said Dan Kohn, Executive Director, Cloud Native Computing Foundation. "**The interoperability that this program ensures is essential to Kubernetes meeting its promise of offering a single open source software project supported by many vendors that can deploy on any public, private or hybrid cloud.**"

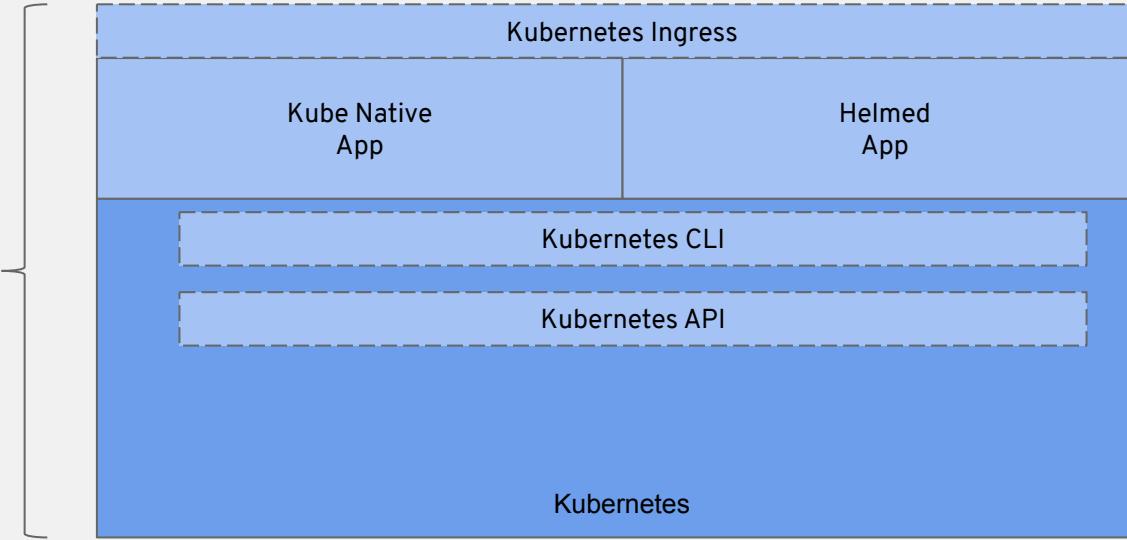
One of the goals of the project has always been **consistency and portability**. Kubernetes sits on top of the infrastructure and enables you to **describe your workload in a common format**. Kubernetes **makes it easy to move workloads from one place to another**, or combine disjointed environments with a shared control plane.

This **program gives end users the confidence** that when they use a **Certified Kubernetes** product they **can rely on a high level of common functionality**. It gives Independent Software Vendors (ISVs) **confidence that if their customer is using a Certified Kubernetes platform that their software will behave as expected**.

* <https://github.com/cncf/k8s-conformance>

UPSTREAM KUBERNETES

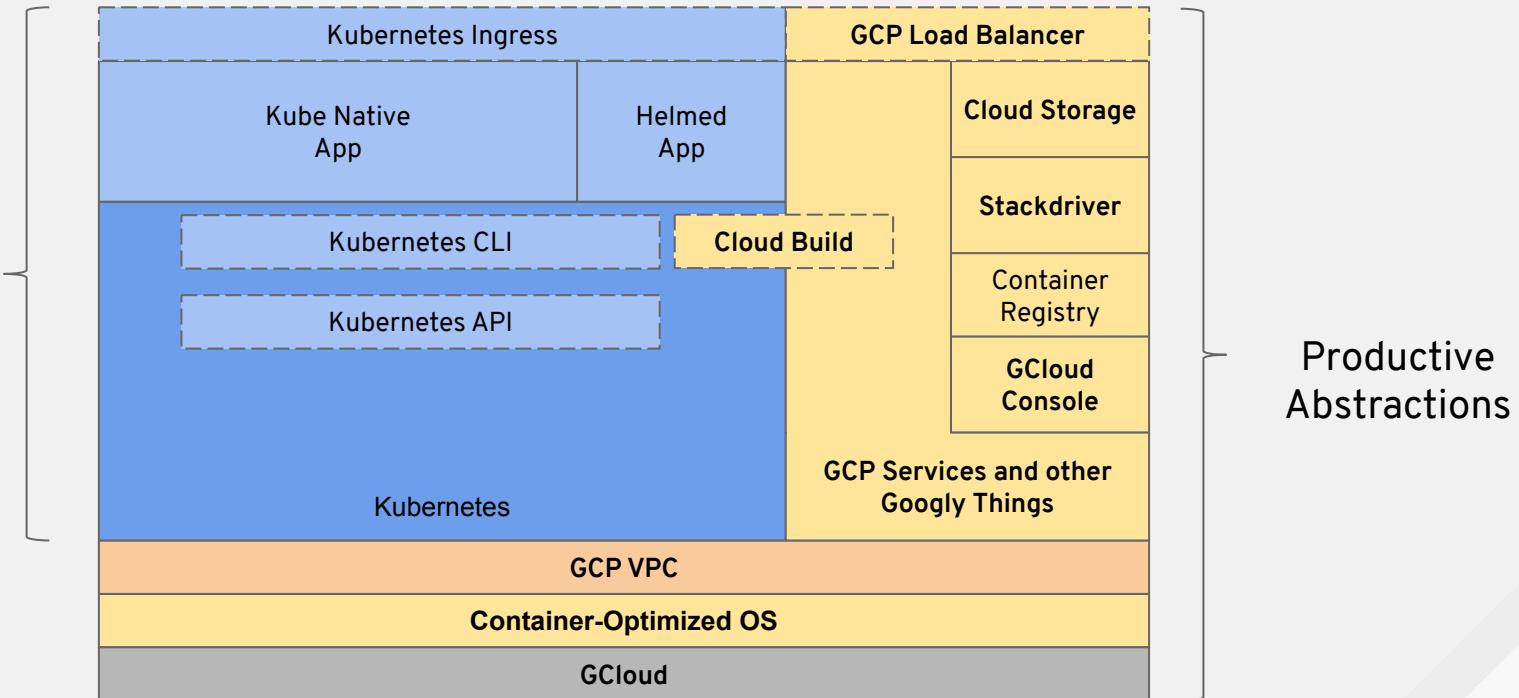
Upstream is closest to “Vanilla”



* Not comprehensive. Not even close.

GKE

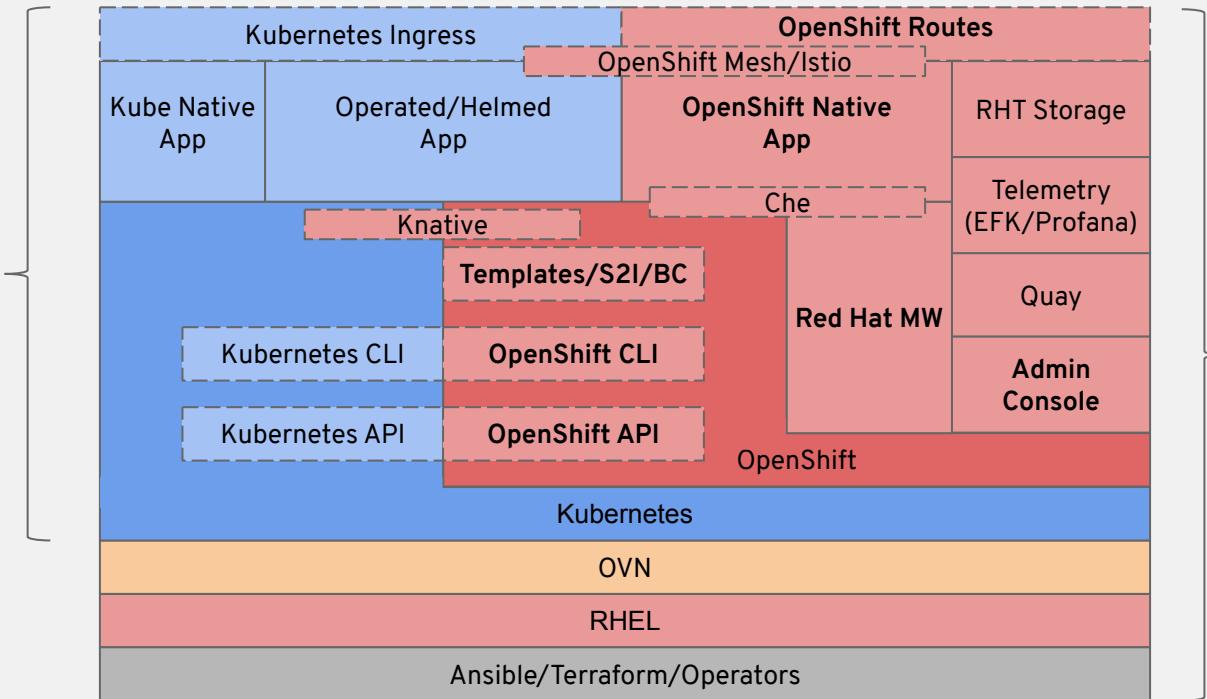
Even one of the Kube founders is not “Vanilla”



* Not comprehensive. Not even close. GCP only APIs in **bold**.

OPENSIFT

Certified “Pure” Kubernetes and productive abstractions



* Not comprehensive. Not even close. Red Hat or OpenShift only APIs in **bold**.

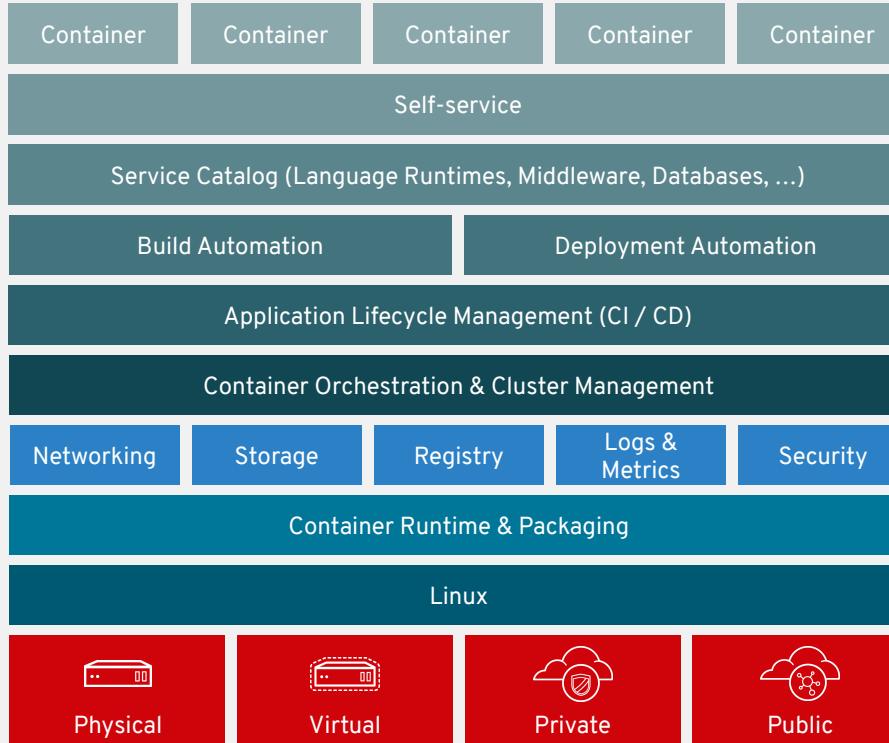
Appendix: Community projects vs enterprise software products



DIY CONTAINER STACK CHALLENGES

Bring your own middleware, data & other services. Build out a service catalog / interface to enable self-service deployment.

Pull Kubernetes or other orchestration (Mesos, Swarm) from rapidly moving upstream & support / maintain yourself. Do all the work required to integrate it into your enterprise IT environment (networking, storage, registry, security, logging, metrics, etc.)



Take existing application build/CI & deployment tools and evolve to add container image build & mgt., continuous deployment, etc.

Pull Docker container runtime from rapidly moving upstream and support, secure and maintain it yourself.

Support and manage your own Linux community distro or build on existing RHEL or 3rd party commercial Linux offerings.

CONTAINER INFRASTRUCTURE AND MANAGEMENT

	Kubernetes	OKD*	OpenShift
Multi-host container scheduling	✓	✓	✓
Self-service provisioning	✓	✓	✓
Service discovery	✓	✓	✓
Enterprise Linux operating system			✓
Image registry		✓	✓
Validated storage plugins		✓	✓
Networking and validated networking plugins		✓	✓
Log aggregation and monitoring		✓	✓
Multi-tenancy		✓	✓
Metering and chargeback			✓

* OKD is the open source project [formerly known as OpenShift Origin](#)

DEVELOPER EXPERIENCE

	Kubernetes	OKD*	OpenShift
Automated image builds	No developer or application services	✓	✓
CI/CD workflows and pipelines		✓	✓
Certified application services			✓
Certified middleware			✓
Certified databases			✓
200+ certified ISV solutions			✓

* OKD is the open source project [formerly known as OpenShift Origin](#)

ENTERPRISE SUPPORT AND COMMUNITY

	Kubernetes	OKD	OpenShift
Community forums and resources	✓	✓	✓
Zero downtime patching and upgrades			✓
Enterprise 24/7 support			✓
9 year support lifecycle			✓
Security response team			✓

External review: [10 most important differences between OpenShift and Kubernetes](#)

Appendix: Docker Support in OpenShift 4



IS DOCKER THE BEST AVAILABLE CONTAINER ENGINE?

Potential limitations surrounding Docker

- Build requires a “big fat” daemon on every host
- Regression for integration with container platforms
Kubernetes/OpenShift
- Build has secret handling issues
- Root/privileged concerns at runtime
- Root/privileged concerns with daemon
- Build requires a running container



CONTAINER INNOVATION CONTINUES

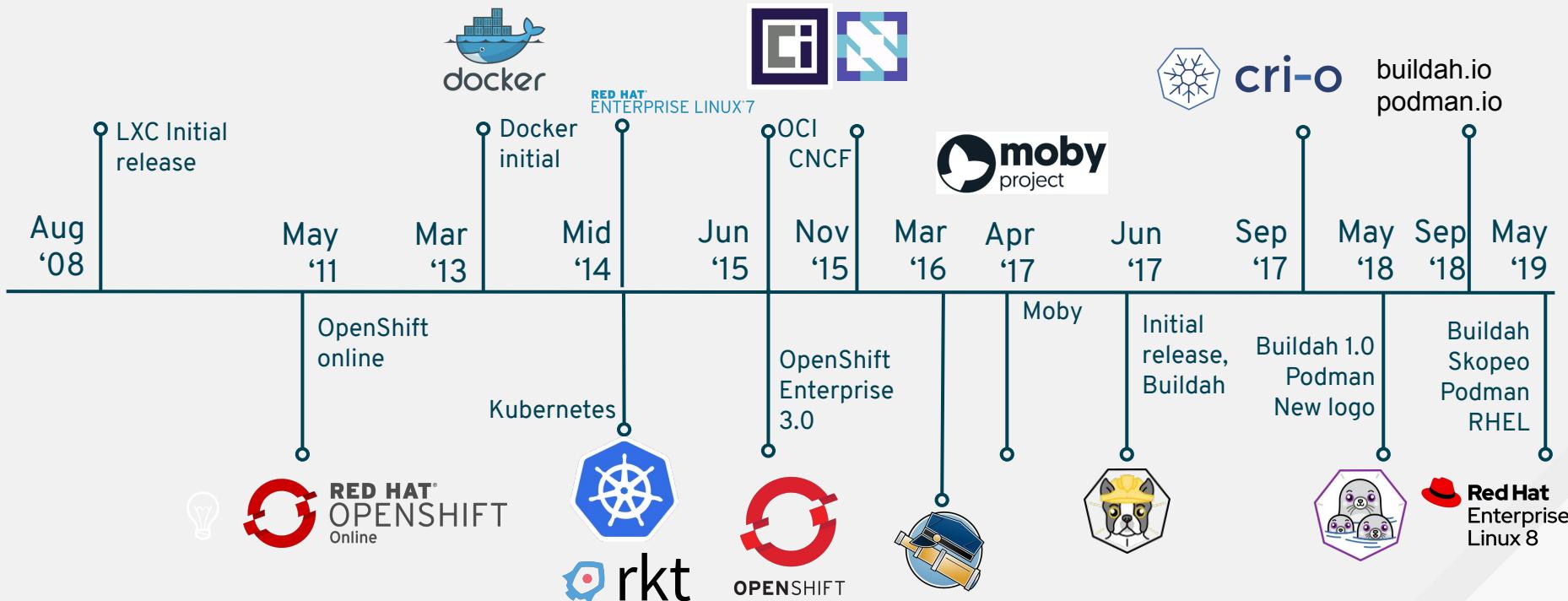
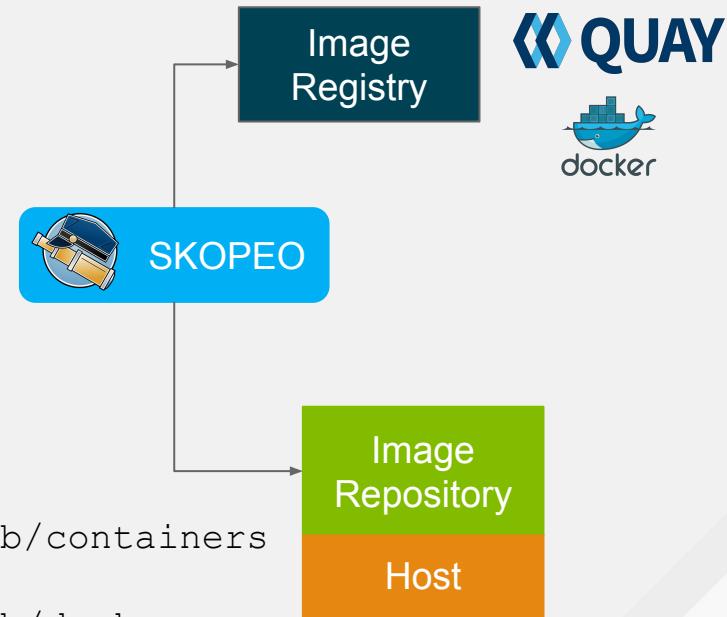
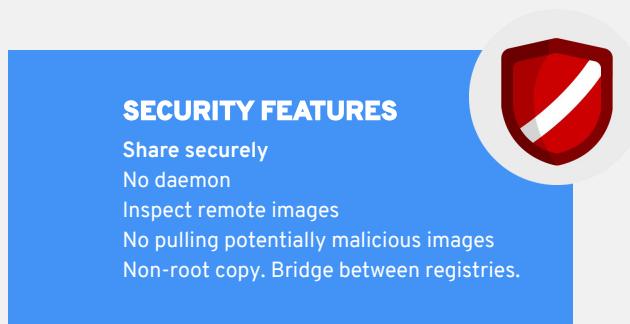


IMAGE COPY WITH SKOPEO



- Built for interfacing with Docker registry
- CLI for images and image registries
- Rejected by upstream Docker `＼(ツ)／`
- Allows remote inspection of image meta-data - no downloading
- Can copy from one storage to another



The new container CLI

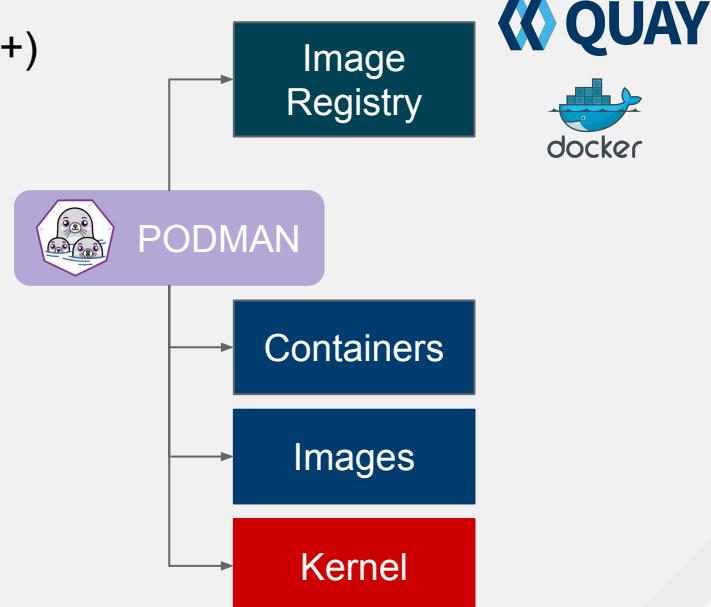


podman

- [@ podman.io](https://podman.io)
- Client only tool, based on the Docker CLI. (same+)
- No daemon!
- Storage for
 - **Images** - containers/image
 - **Containers** - containers/storage
- Runtime - runc
- Shares state with CRI-O and with Buildah!

SECURITY FEATURES

Run and develop securely
No daemon
Run without root
Isolate with user namespaces
Audit who runs what



Why use Buildah?

- Now buildah.io
- Builds OCI compliant images
- No daemon - no “docker socket”
- Does not require a running container
- Can use the host’s user’s secrets.
- Single layer, from scratch images are made easy and it ensures limited manifest.
- If needed you can still maintain Dockerfile based workflow

SECURITY FEATURES

Build securely
No daemon
Shrink the attack surface
Fine-grained control of the layers
Run builds isolated
Better secret management



buildah

