OpenShift 4

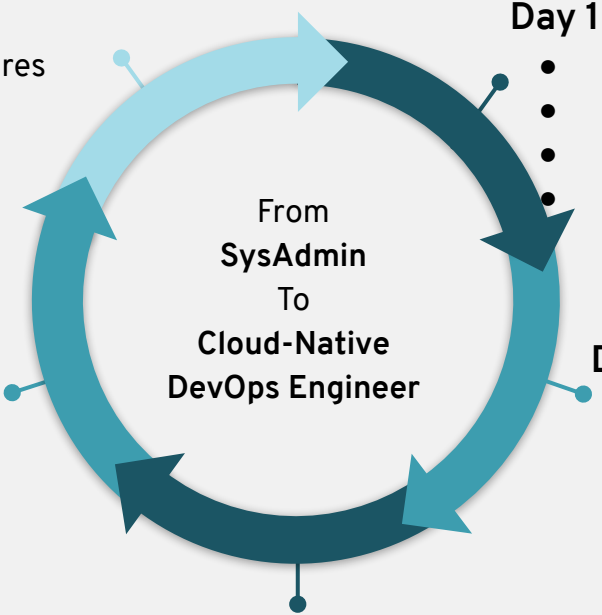
Making introductions

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



Day 1 - Overview

- Cloud-Native market
- Container Concepts
- Platform Concepts
- Platform Architecture

OpenShift - Technical Deep Dive

Day 2 - App Deployment

- Technical Deep Dive
- Labs
- Compute: Pods
- Networking: Services, Routes
- Storage: Persistent Volumes

Day 3 - App Build

- Labs
- Source-2-Image
- Jenkins

Day 4 - Misc

- RBAC, IAM
-

Day 5 - Best Practices

- Reference Architectures
- Use Cases
- App OnBoarding

Client Setup

Gets you started

Interactive Workshop

OpenShift WebConsole	https://console-openshift-console.apps.learn.ont.belastingdienst.nl
OpenShift CLI	oc login -u <vdi-user> 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}

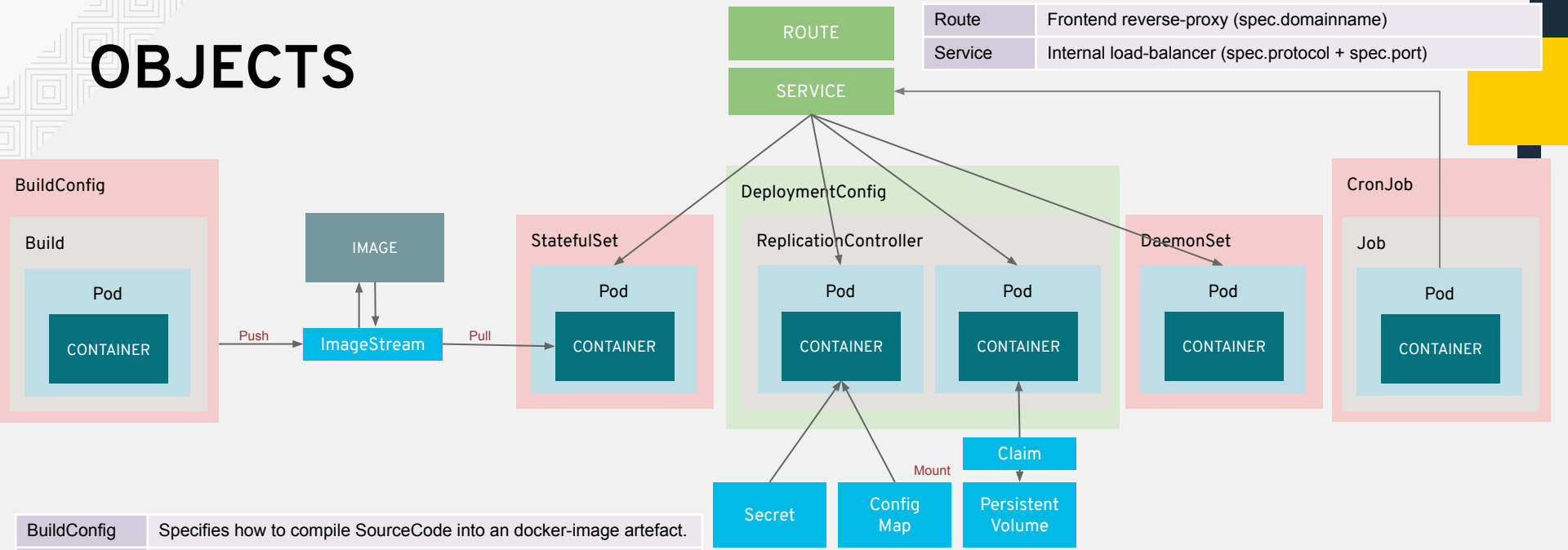
oc get pod myapp-1-abcde

{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-abcde	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 you app to Production

OBJECTS

Route	Frontend reverse-proxy (spec.domainname)
Service	Internal load-balancer (spec.protocol + spec.port)



BuildConfig	Specifies how to compile SourceCode into an docker-image artefact.
Build	Runs the build process (spec.gitref + spec.buildImage + spec.targetImage)
ImageStream	A pointer to an external image (spec.imageUrl)
Image	Application binary (content)

PersistentVolumeClaim	A “request” for storage space (spec.size)
PersistentVolume	A (mount)pointer to Network Storage (spec.connectionstring)
ConfigMap	A mountable property file (spec.data[])
Secret	A mountable base64-encoded file (spec.data[])

DeploymentConfig	Specifies how to deploy your application (rolloutStrategy + imageChangeTriggers)
ReplicaSet	Monitors the Pods and restarts if needed (spec.healthcheck + spec.replicas)
Pod	A grouping of tightly-coupled containers (spec.containers[] + spec.node)
Container	Running your application process (spec.command + spec.security
StatefulSet	Same a Deployment, but then for Statefull workloads
DaemonSet	Same a Deployment, but exactly 1 Pod per Node. (spec.nodeSelector)
CronJob	Schedules a Pod at a specified time. (spec.schedule)
Job	Monitors the one-off Pod for successful completion, restarts if needed.

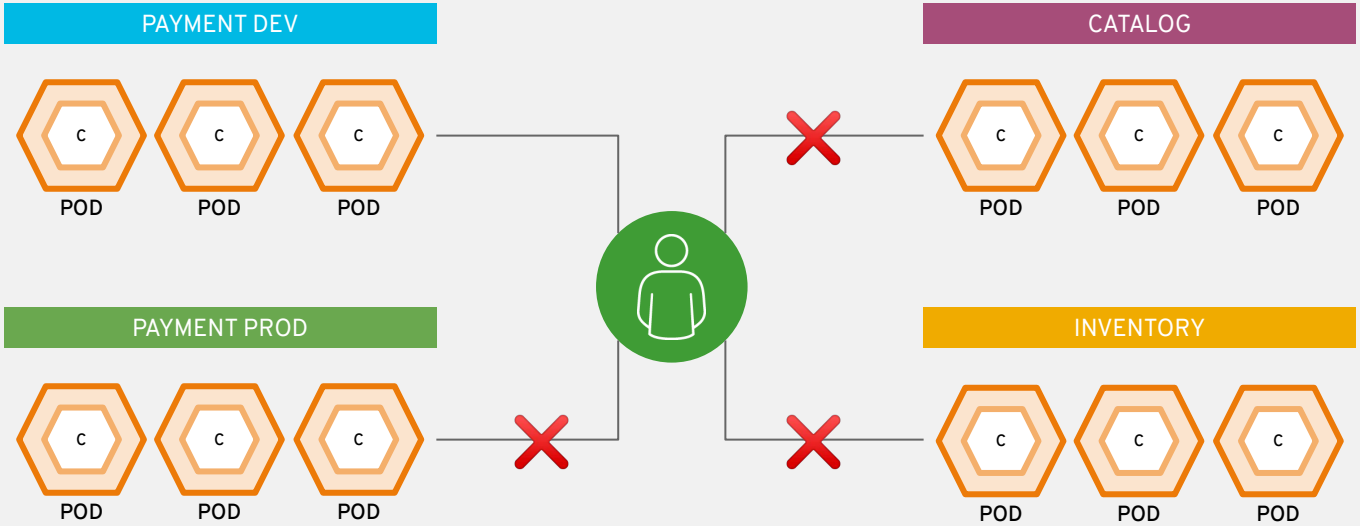
Recap - day 3

Do you still remember?

Kubernetes Resource Definitions

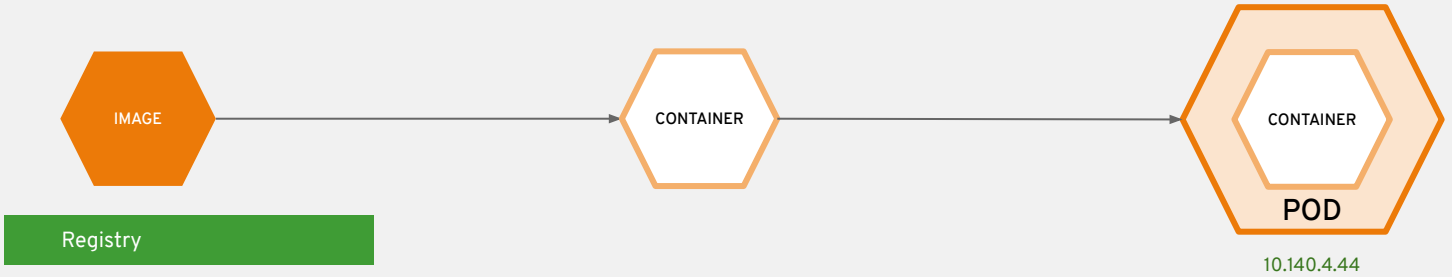
What types of workloads can you deploy on top of a Container Platform...

projects isolate apps across environments, teams, groups and departments



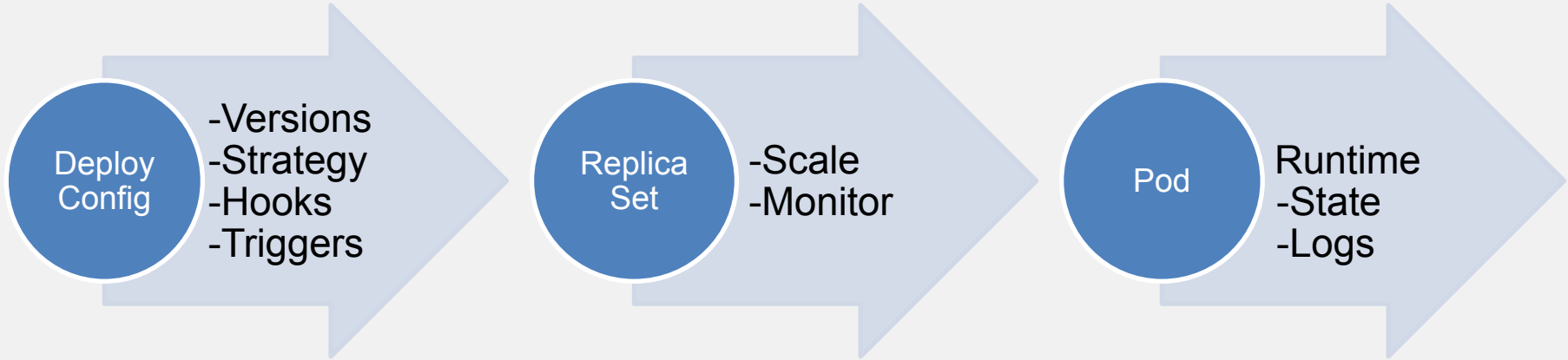
> oc new-project workshop-<vdi-user>
> oc project

It all starts with an image



> oc -n openshift get imagestreams
> oc -n openshift describe imagestream httpd
> oc -n <workshop> import-image docker.io/library/httpd

Deployment Process



- MyJBossApp

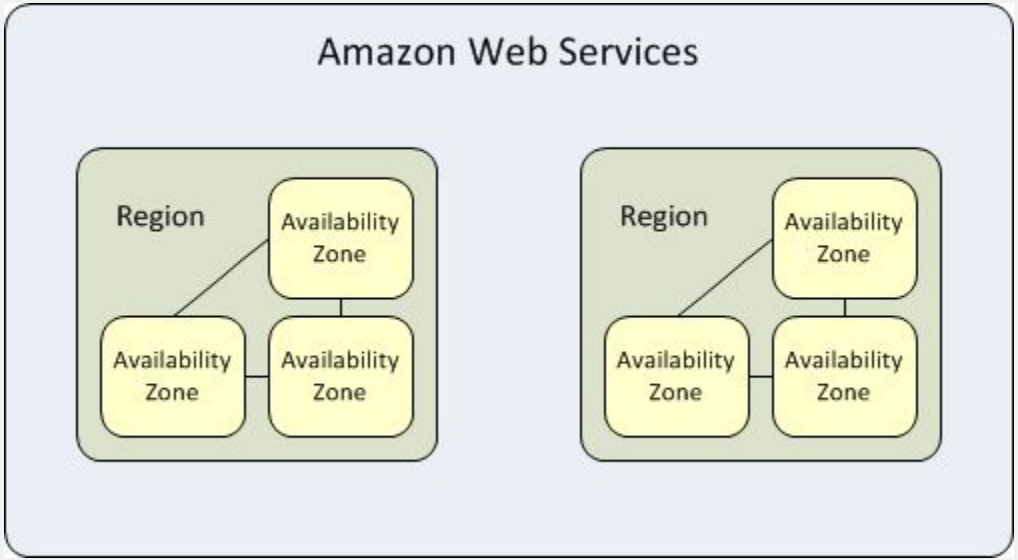
- MyJBossApp-v1 (2x)
- MyJBossApp-v2 (4x)

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

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

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

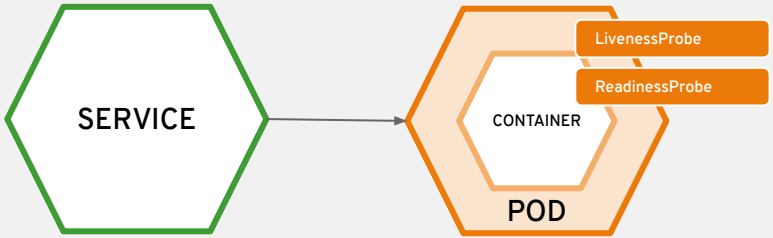
PLACEMENT BY POLICY
Pod Affinity rules



Preferred vs Required
Affinity vs Anti-Affinity
- Nodes
- Services
- Persistent Volumes

> ssh master1 cat /etc/origin/master/scheduler.json

Health Checks

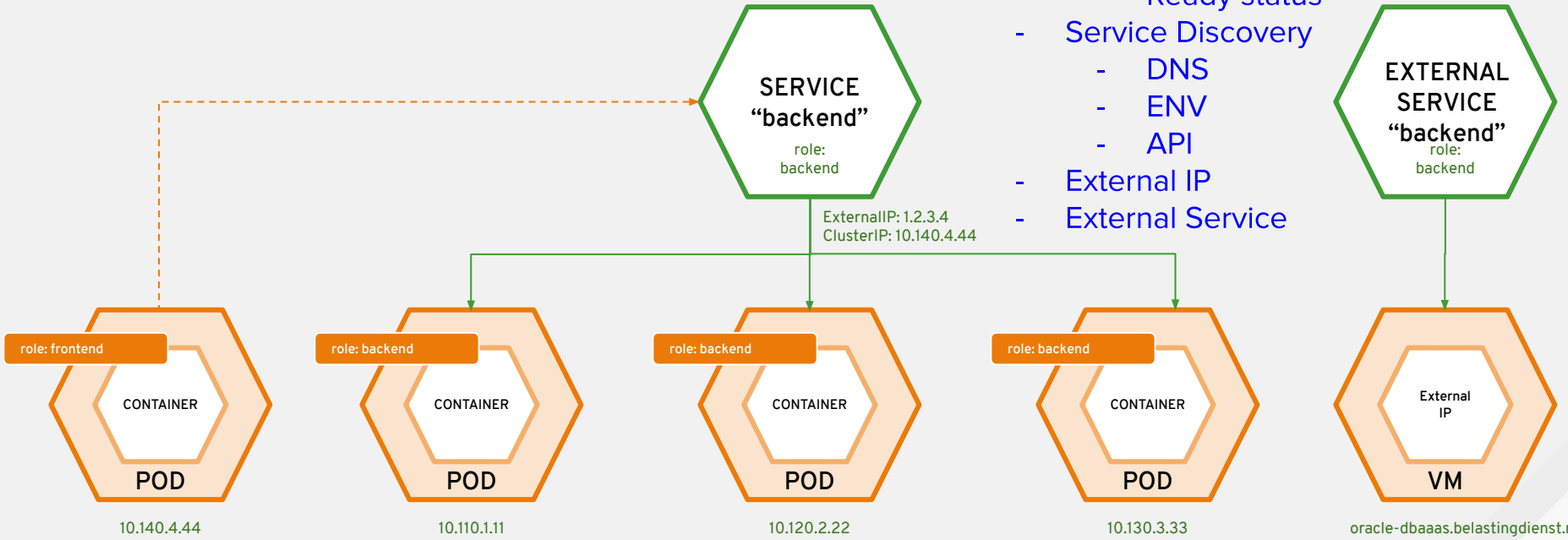


- LivenessProbe
- ReadinessProbe
  - HTTP
  - TCP
  - Shell

> 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

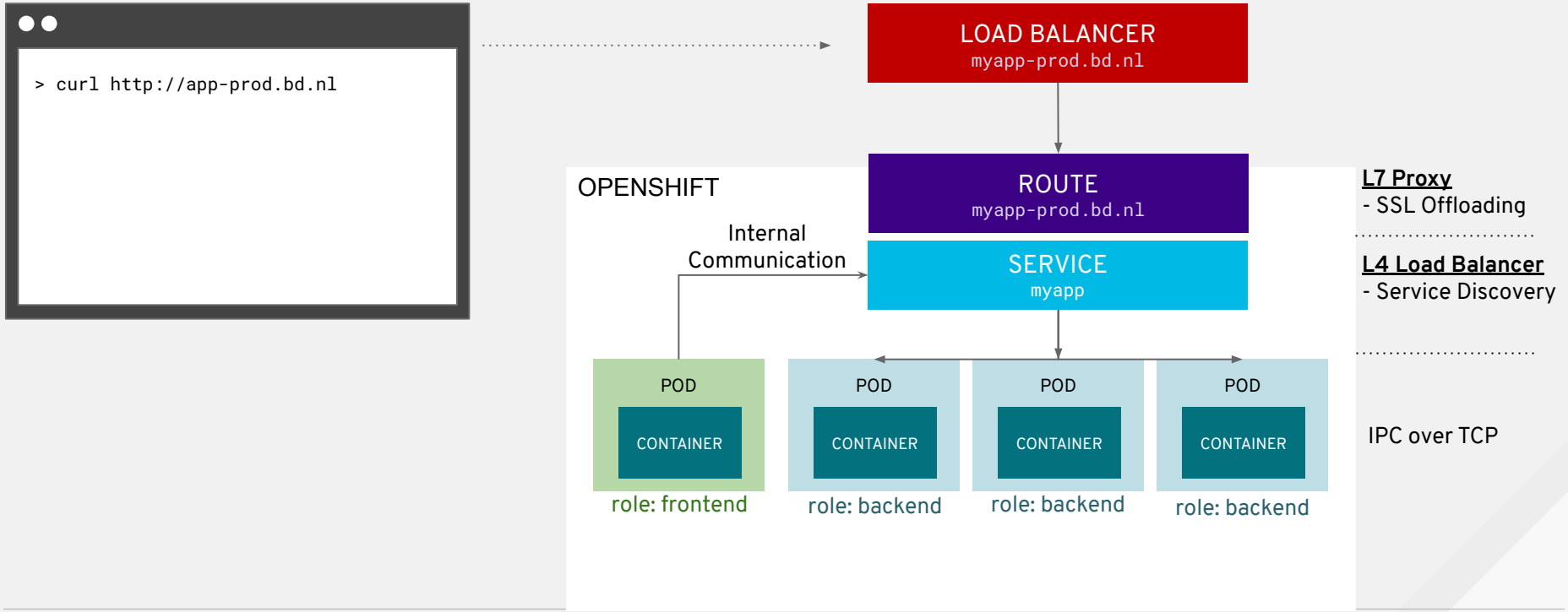
services provide internal load-balancing

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

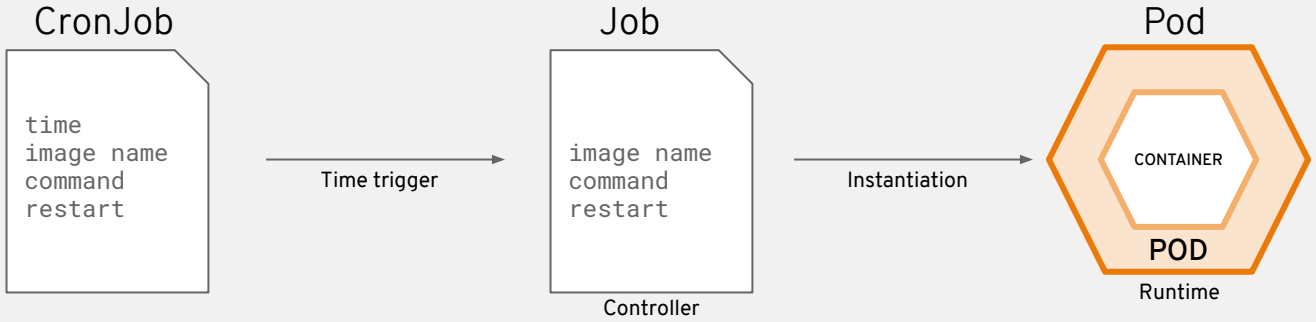


> oc describe svc/httpd |egrep “Selector|Endpoints”
> oc get pods -o wide -l app=httpd

ROUTING TRAFFIC

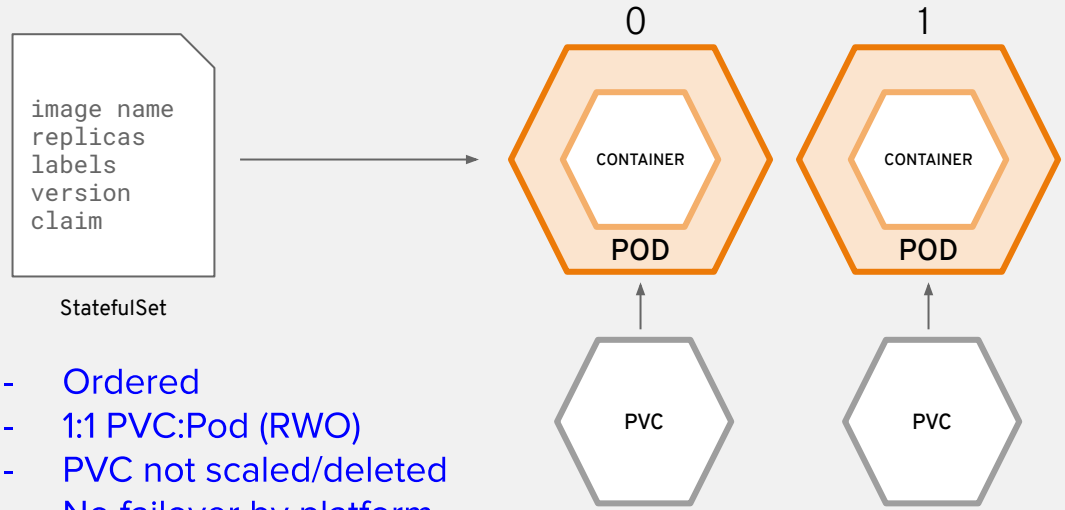


CronJobs run short lived pods at a specified time interval




```
> oc create cronjob myjob --image=httpd --schedule="*/10 * * * *" --restart=OnFailure
> oc create job --from=cronjob/myjob
> oc patch cronjob/myjob -p '{"spec": {"jobTemplate": {"spec": {"template": {"spec": {"containers": [ {"name": "myjob", "command": ["/bin/bash","-c","echo hello world"] } ] } } } } } }'
```

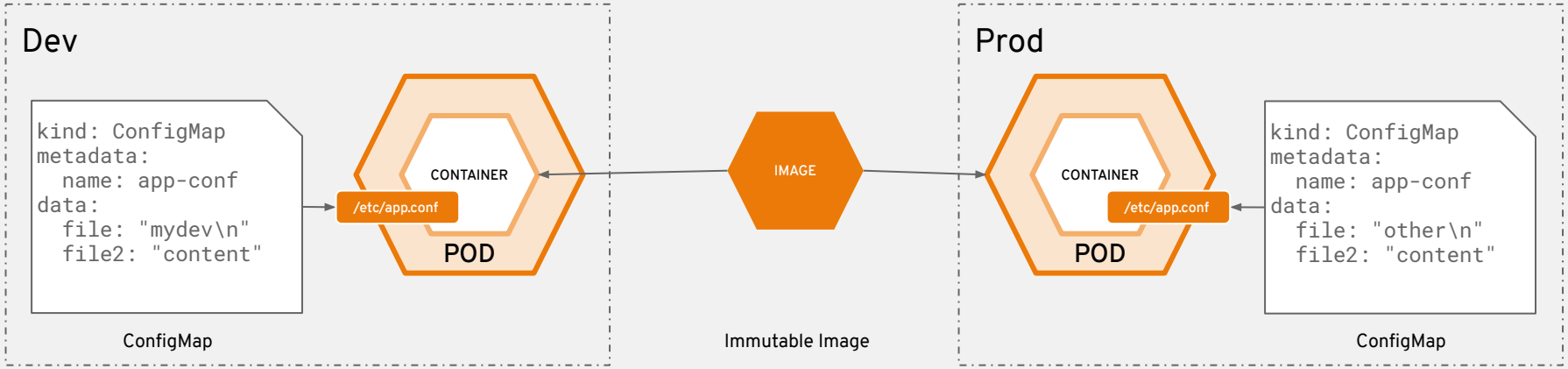
`StatefulSets` are special deployments for Stateful workloads



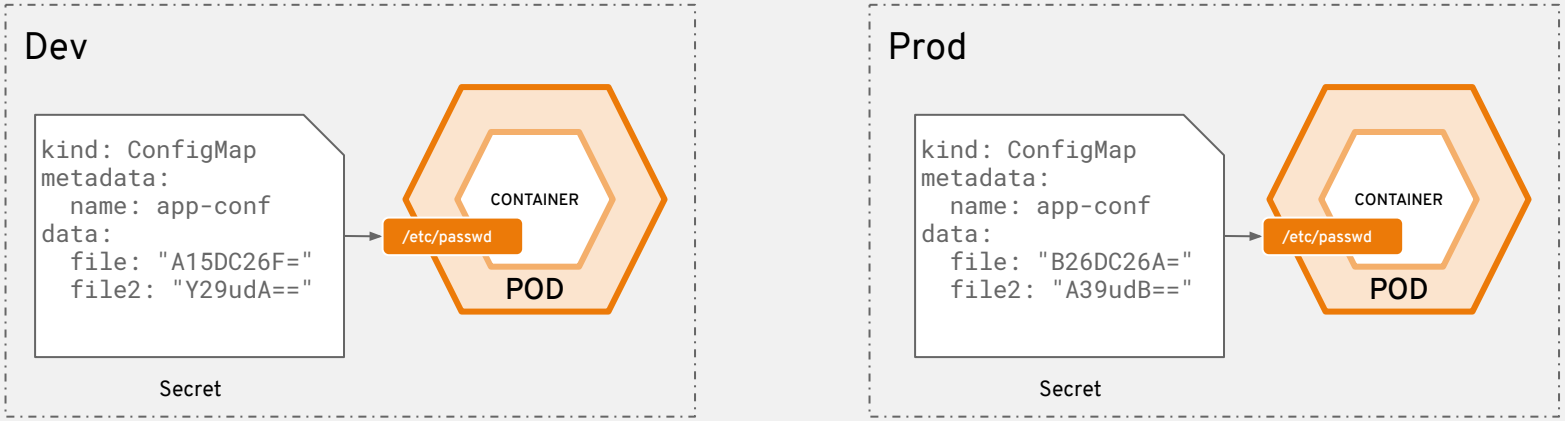
- Ordered
- 1:1 PVC:Pod (RWO)
- PVC not scaled/deleted
- No failover by platform

```yaml
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
```

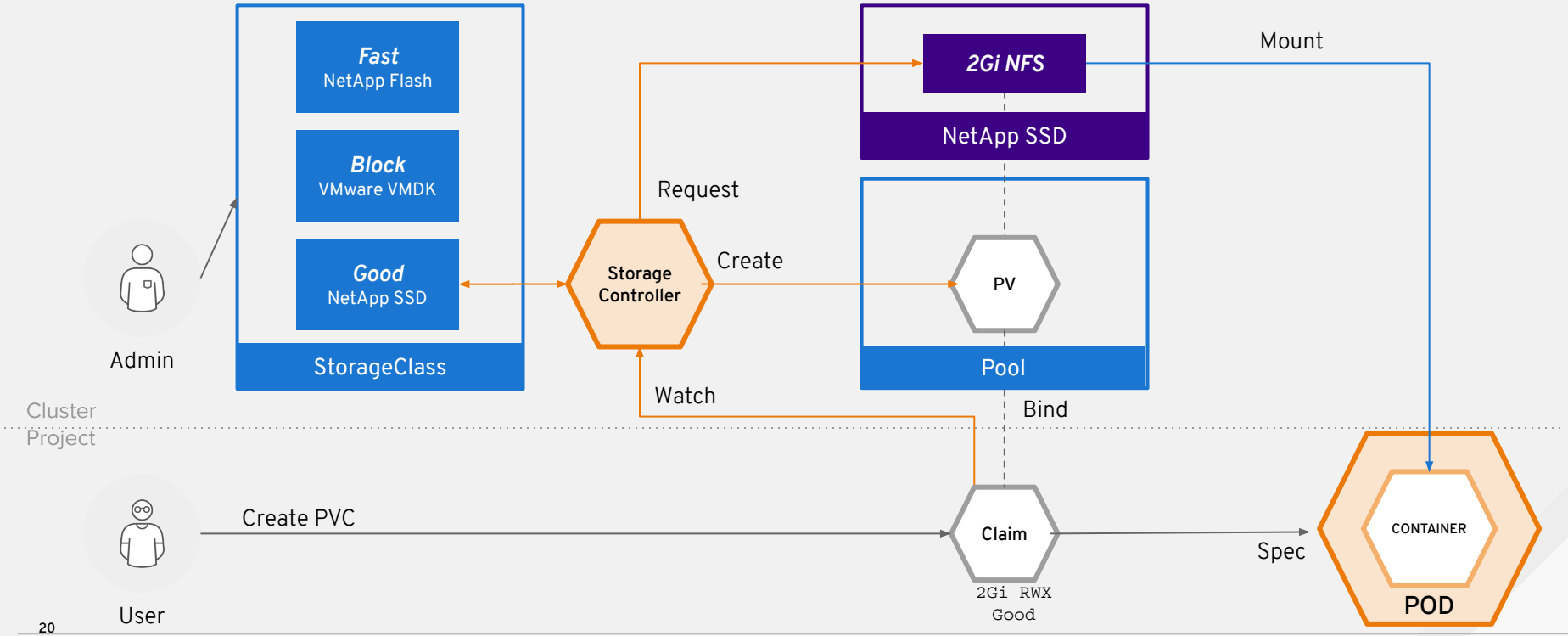
`configmaps` allow you to decouple configuration artifacts from image content



`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

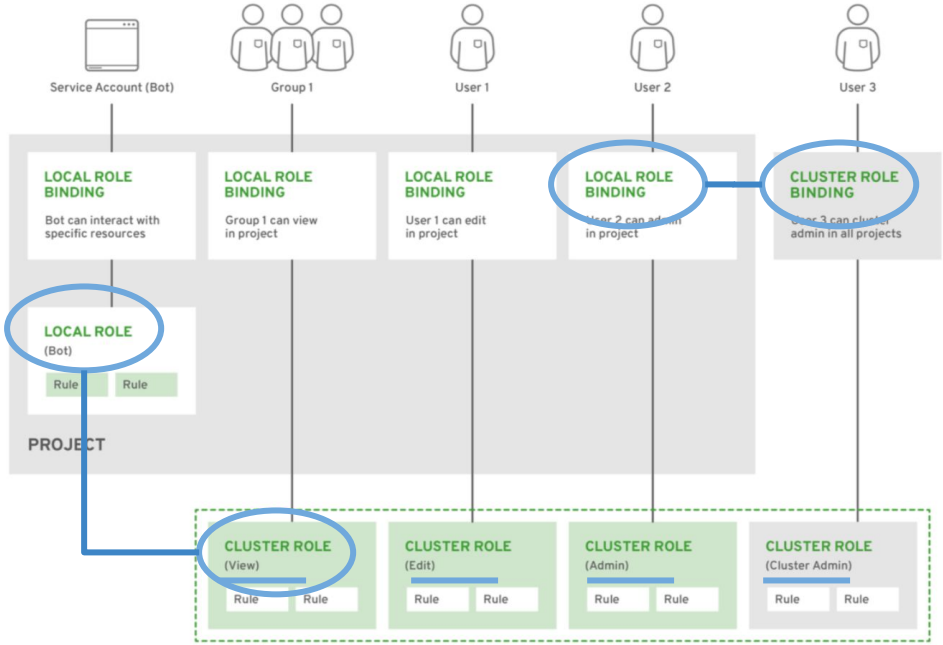


Figure 12 - Authorization Relationships

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

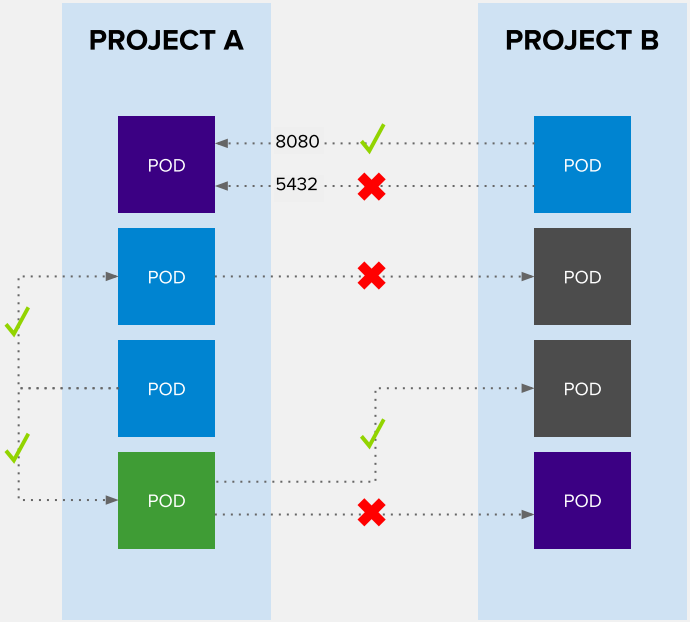
Admin can grant access to privileged SCC

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

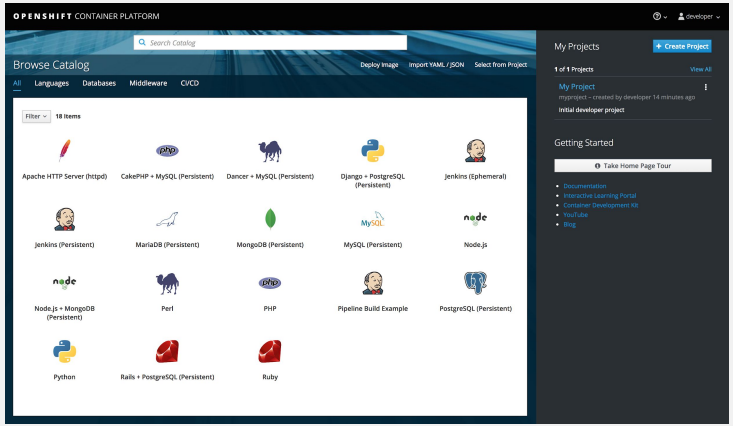


Example Policies

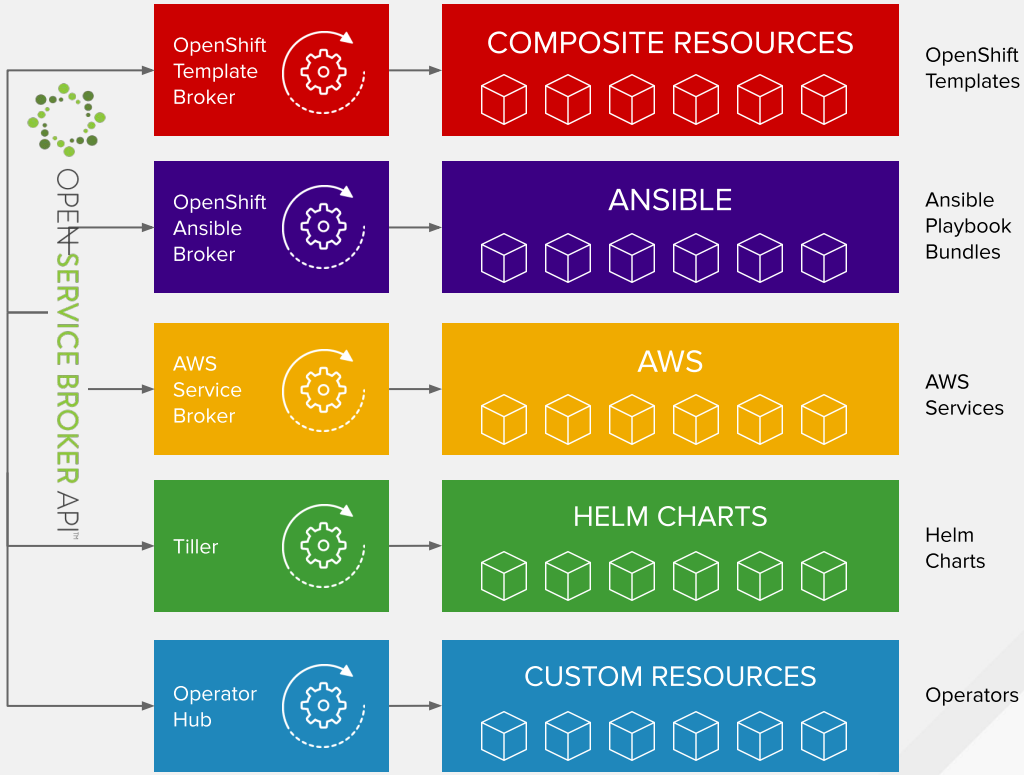
- Allow all traffic inside the project
- Allow traffic from green to gray
- Allow traffic to purple on 8080

```
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
```

SERVICE CATALOG

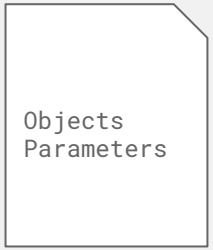


OPENSHIFT DEVELOPER CATALOG

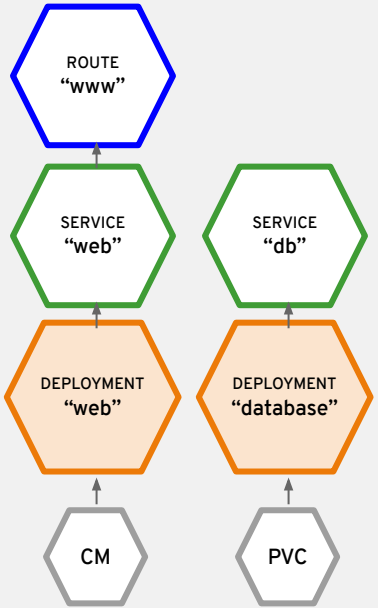


a `Template` is a list of composite resources

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



Template




```
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
```

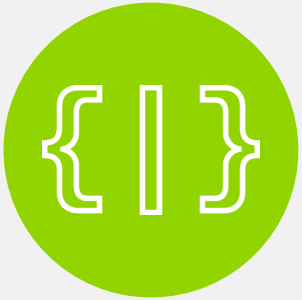

> oc create -f template.yaml

Build & Deploy Images

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

BUILD CONTAINER IMAGES

BUILD FROM BINARY

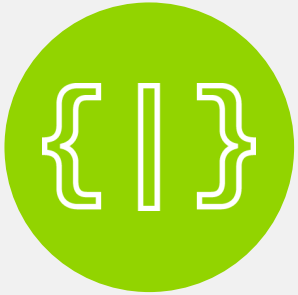


BUILD FROM SOURCE CODE

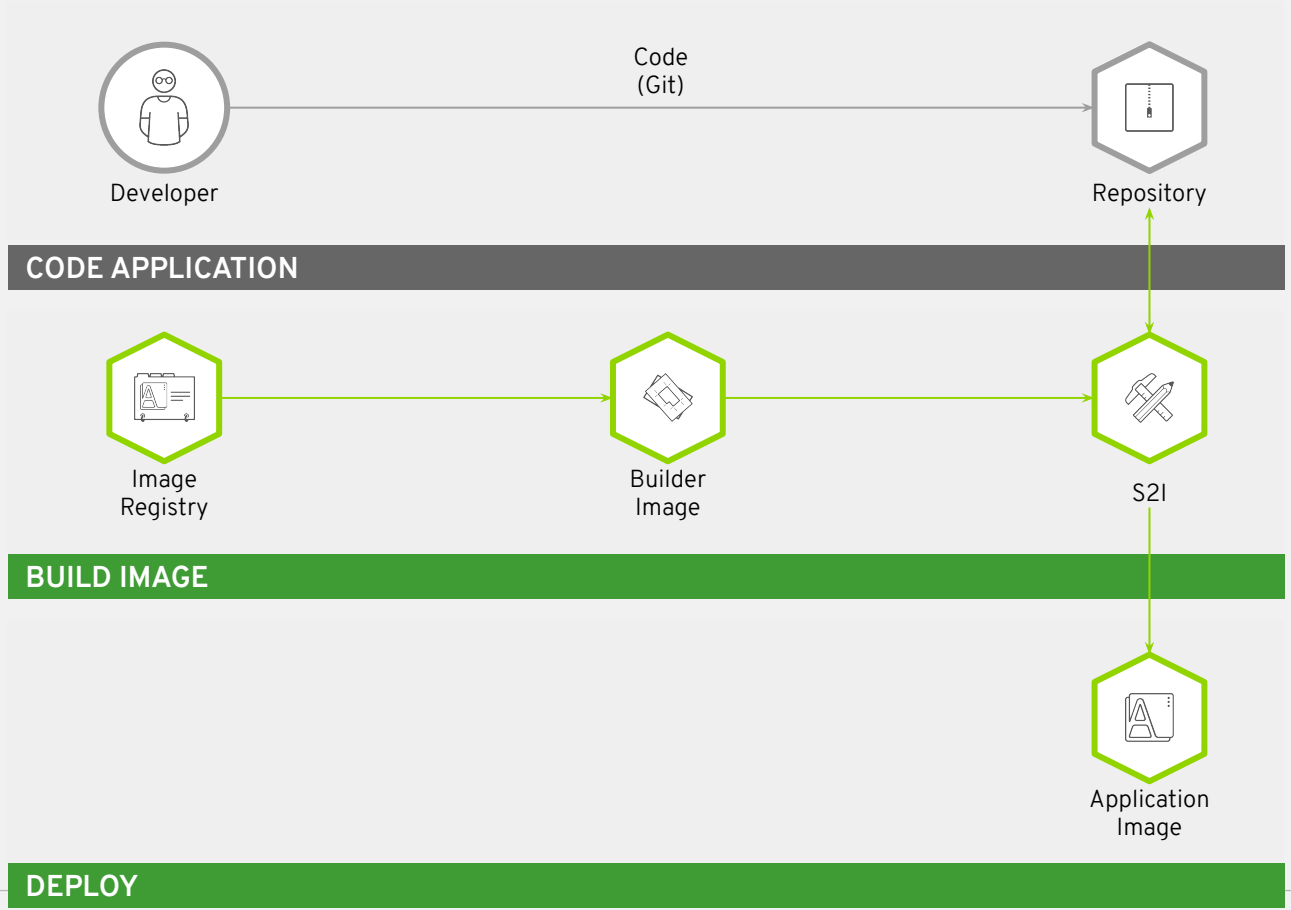
BUILD FROM Dockerfile

BUILD FROM Jenkinsfile

CUSTOM BUILDER IMAGE



BUILD YOUR SOURCE CODE


Code
(Git)
Developer
Repository
CODE APPLICATION
Image
Registry
Builder
Image
S2I
BUILD IMAGE
Application
Image
DEPLOY

DEPLOY SOURCE CODE WITH SOURCE-TO-IMAGE (S2I)

CODE

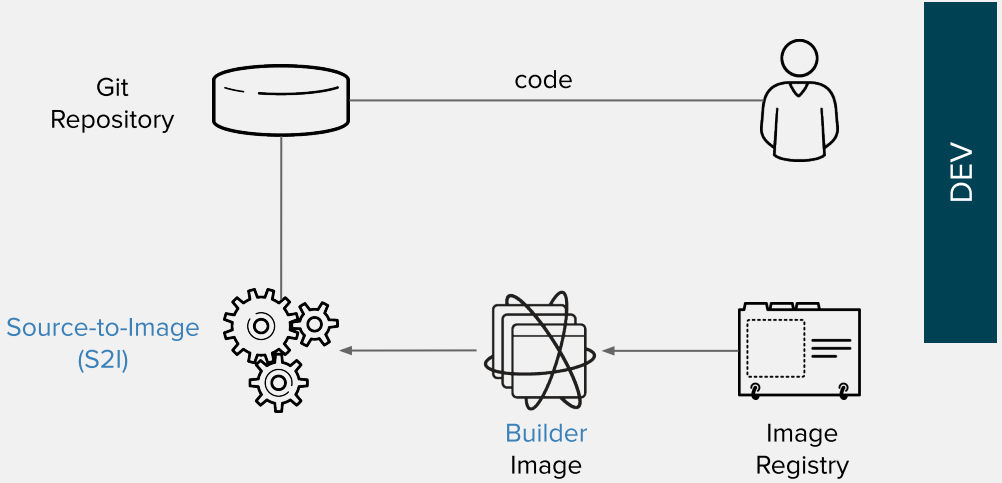
Developers write code using existing development tools such as Maven, NPM, Bower, PIP and Git and then access the OpenShift Web, CLI or IDE to create an app from the code

DEV

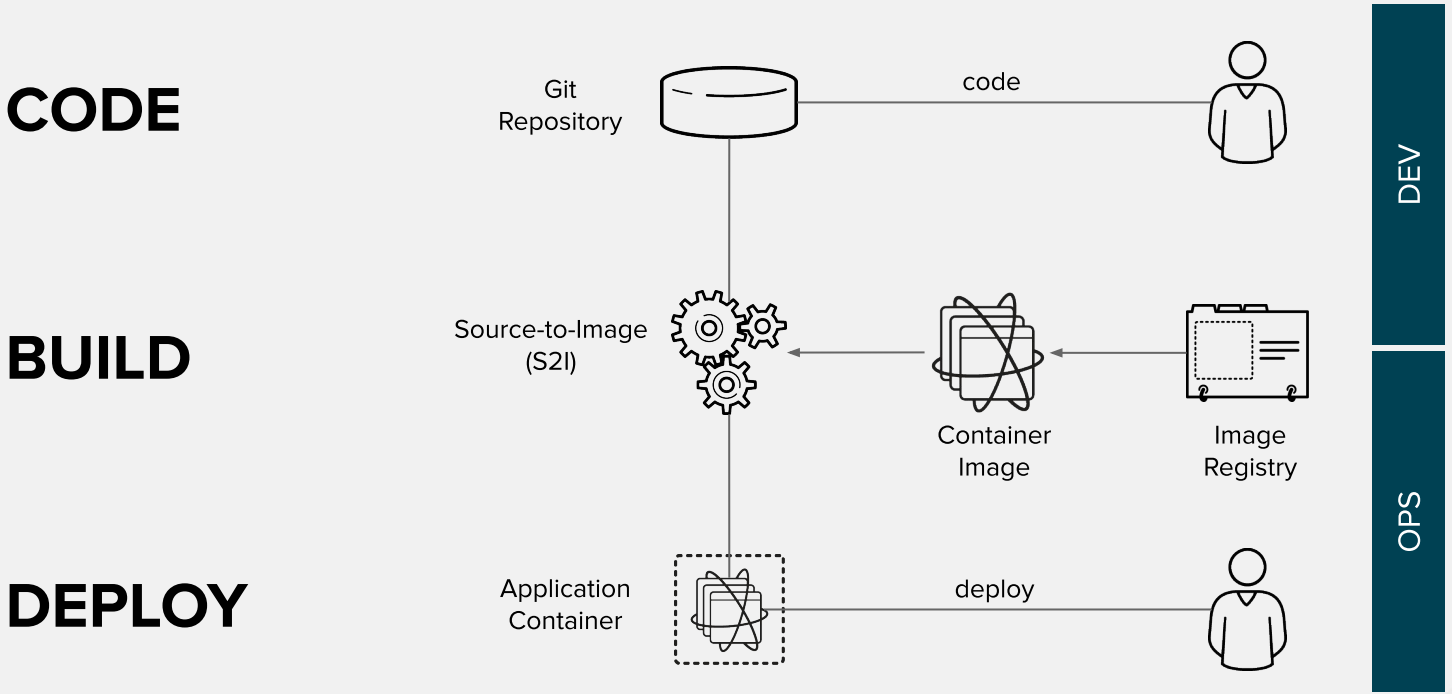
DEPLOY SOURCE CODE WITH SOURCE-TO-IMAGE (S2I)

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.



DEPLOY SOURCE CODE WITH SOURCE-TO-IMAGE (S2I)



Build Source Code

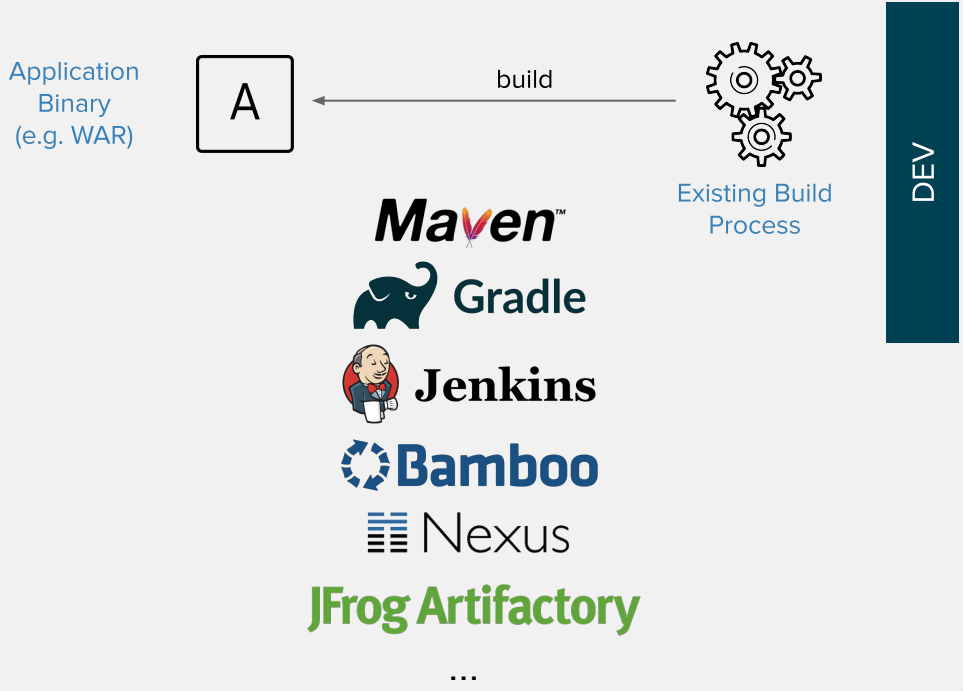
```
oc new-app
https://devtools.belastingdienst.nl/bitbucket/projects/CPET/repos/training-mlbparks#1.0
.0
oc get is
oc get pods -w
oc get -o yaml bc/training-mlbparks
oc logs -f bc/training-mlbparks

git clone
https://devtools.belastingdienst.nl/bitbucket/projects/CPET/repos/training-mlbparks.git
git checkout 1.0.0
vi src/main/java/com/openshift/evg/roadshow/rest/Healthz.java
# s/ok/okiedokie/
oc start-build bc/training-mlbparks --from-dir=.
```

BUILD YOUR APP BINARY

DEPLOY APP BINARY WITH SOURCE-TO-IMAGE (S2I)

BUILD APP

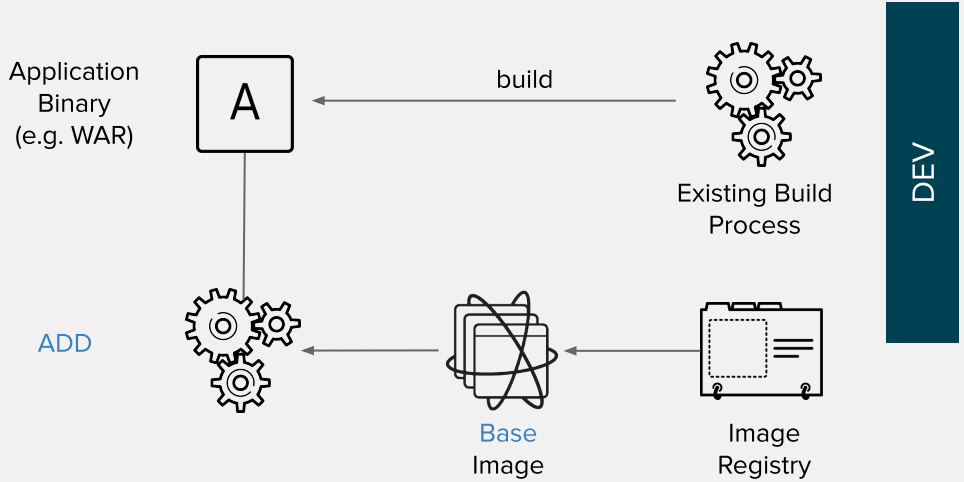
Developers use the existing build process and tools (e.g. Maven, Gradle, Jenkins, Nexus) to build the app binaries (e.g. JAR, WAR, EAR) and use OpenShift CLI to create an app from the app binaries



DEPLOY APP BINARY WITH SOURCE-TO-IMAGE (S2I)

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.

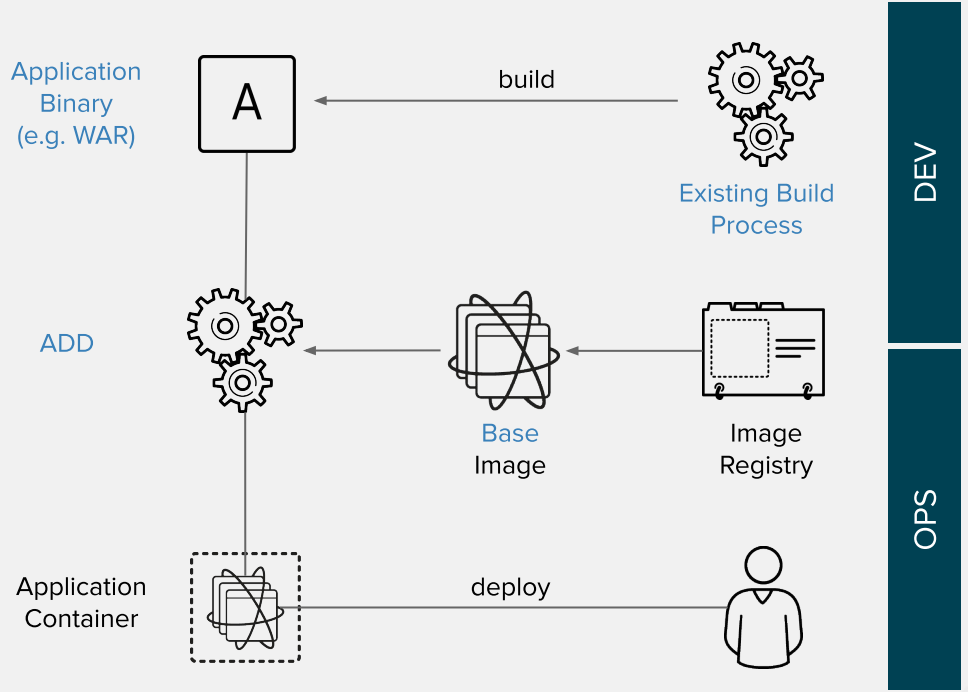


DEPLOY APP BINARY WITH SOURCE-TO-IMAGE (S2I)

BUILD APP

BUILD IMAGE

DEPLOY



Build Binary artifact

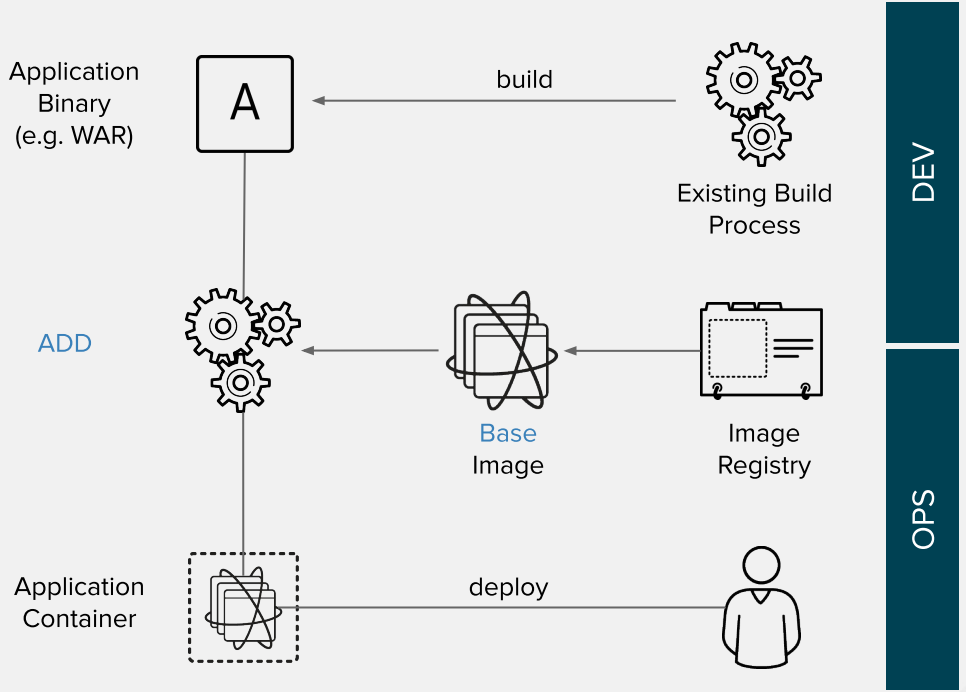
```
mvn package
cat Dockerfile | oc new-build -D - --name mlbparks-binary
Oc get -o yaml
oc start-build bc/mlbparks-binary --from-dir=. --follow
```

```
FROM jboss-eap72-openshift:latest
COPY ROOT.war .
CMD  $STI_SCRIPTS_PATH/run
```

DEPLOY APP BINARY WITH SOURCE-TO-IMAGE (S2I)

DEPLOY

OpenShift automates the deployment of application containers across multiple hosts via the Kubernetes. Users can trigger deployments, rollback, configure A/B or other custom deployments



BUILD YOUR Dockerfile

BUILD Dockerfile
WITH SOURCE-TO-IMAGE (S2I)

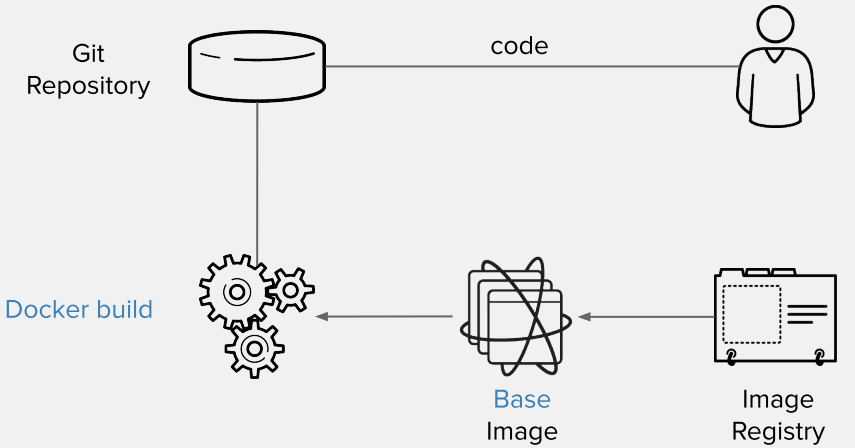
CODE

Developers writes the Dockerfile and commits it into git alongside with all the other files needed for a `docker build`.

DEPLOY Dockerfile WITH SOURCE-TO-IMAGE (S2I)

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.

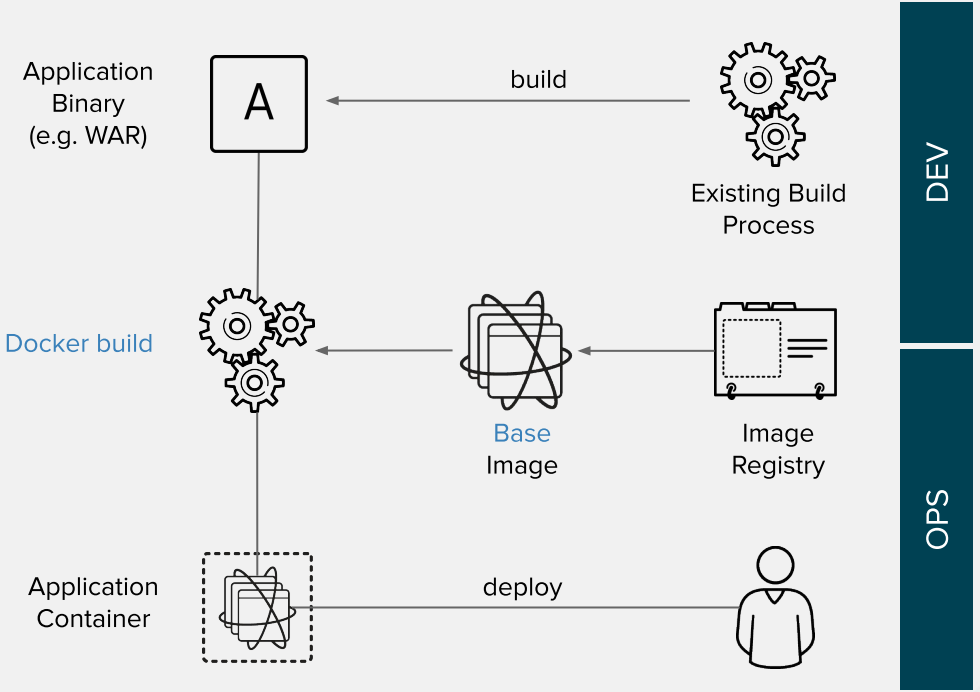


BUILD Dockerfile
WITH SOURCE-TO-IMAGE (S2I)

BUILD APP

BUILD IMAGE

DEPLOY



Build Dockerfile

```
oc new-build
https://devtools.belastingdienst.nl/bitbucket/projects/CPET/repos/training-mlbparks \
  --context-dir=docker \
  --name=mlbparks-dockerfile
```

BUILD YOUR Jenkinsfile

BUILD Jenkinsfile WITH SOURCE-TO-IMAGE (S2I)

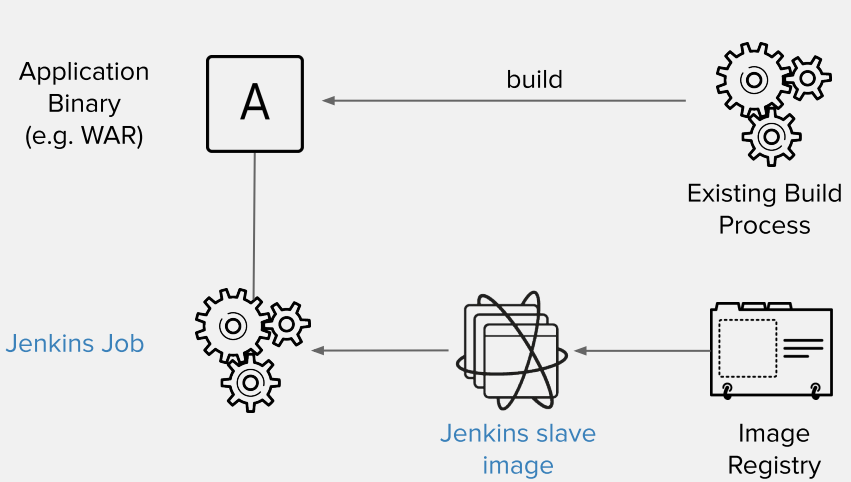
CODE

Developers writes the Jenkinsfile and commits it into git.

BUILD Jenkinsfile
WITH SOURCE-TO-IMAGE (S2I)

BUILD IMAGE

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



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'
          }
        }
```

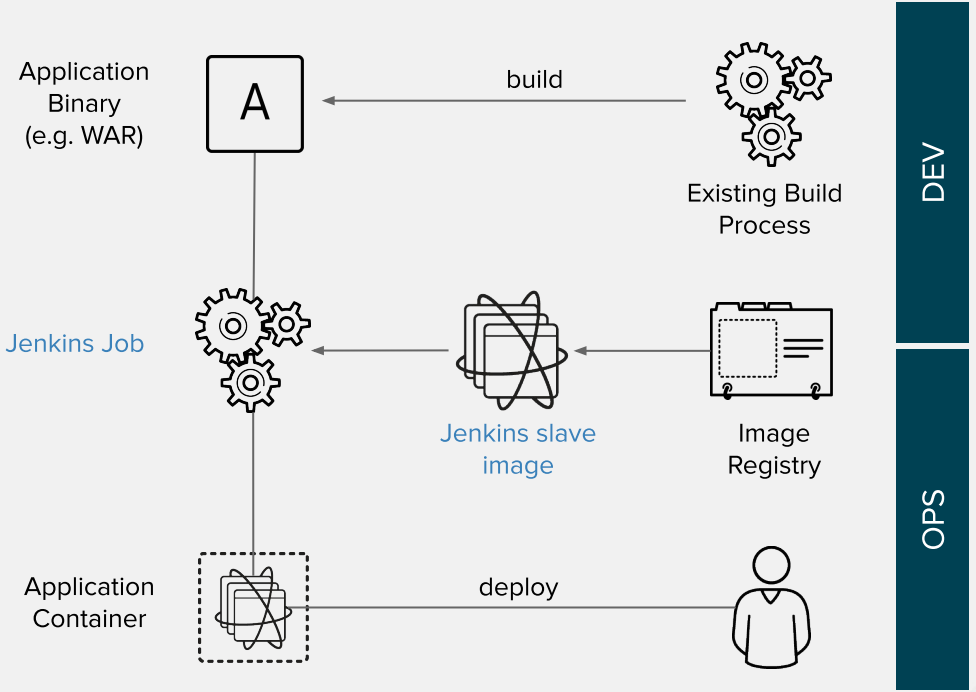
Provision a Jenkins slave for running Maven

BUILD Dockerfile
WITH SOURCE-TO-IMAGE (S2I)

BUILD APP

BUILD IMAGE

DEPLOY



Build Jenkinsfile

```
oc new-build
https://devtools.belastingdienst.nl/bitbucket/projects/CPET/repos/training-mlbparks \
  --name=mlbparks-jenkinsfile

oc get route
```

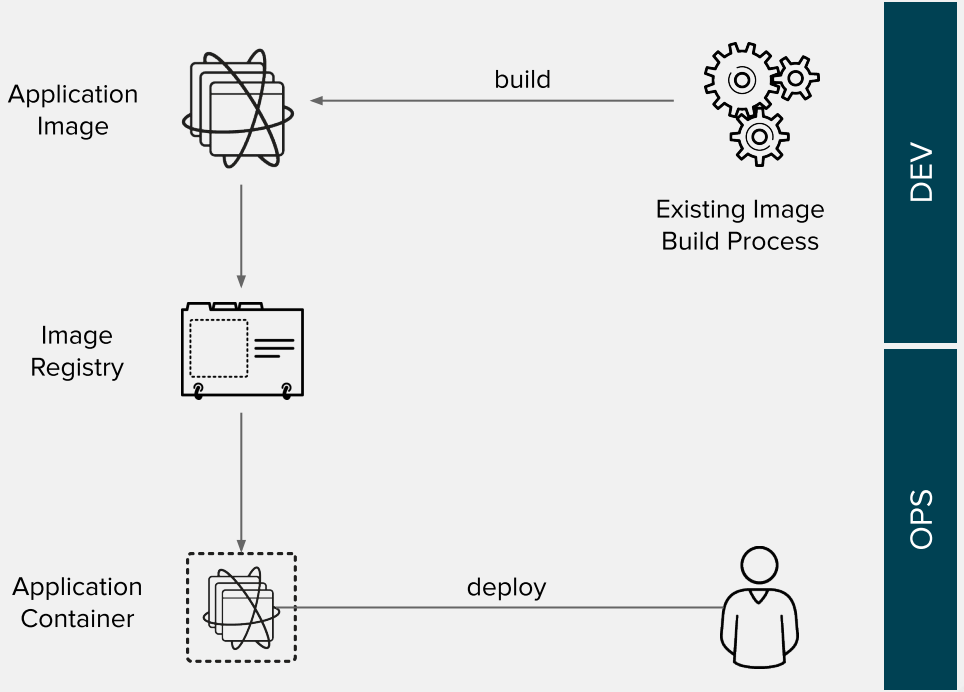
DEPLOY EXISTING CONTAINER IMAGE

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



CONTINUOUS INTEGRATION (CI)
CONTINUOUS DELIVERY (CD)

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

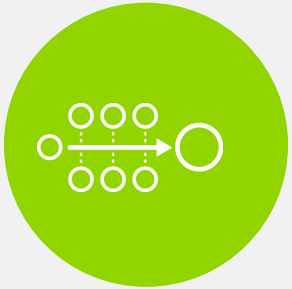
```
> oc new-app httpd-example
> oc get bc/httpd-example -o yaml |grep triggers: -A10
```

DevOps Excellence Awards 2017

OPENSHIFT LOVES CI/CD

JENKINS-AS-A SERVICE ON OPENSHIFT

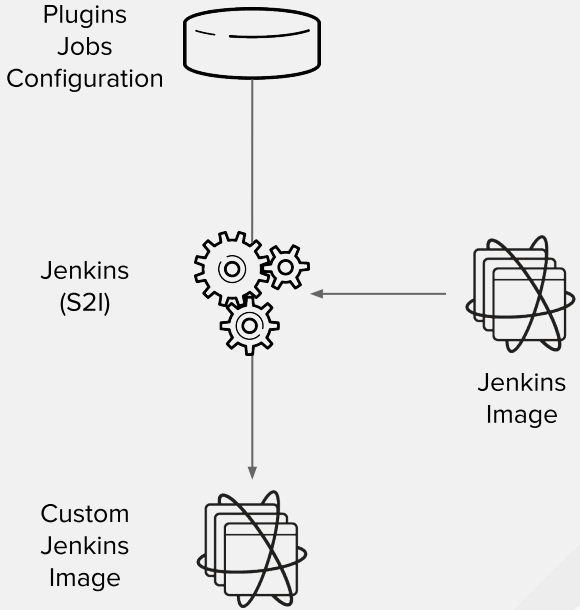
HYBRID JENKINS INFRA WITH OPENSHIFT



EXISTING CI/CD DEPLOY TO OPENSHIFT

1) JENKINS-AS-A-SERVICE ON OPENSHIFT

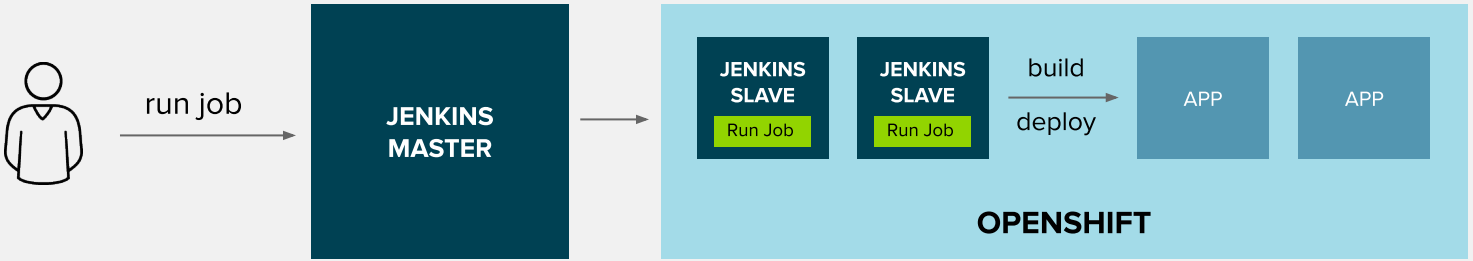
- Certified Jenkins images with pre-configured plugins
  - Provided out-of-the-box
  - Follows Jenkins 1.x and 2.x LTS versions
- Jenkins S2I Builder for customizing the image
  - Install Plugins
  - Configure Jenkins
  - Configure Build Jobs
- OpenShift plugins to integrate authentication with OpenShift 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
```

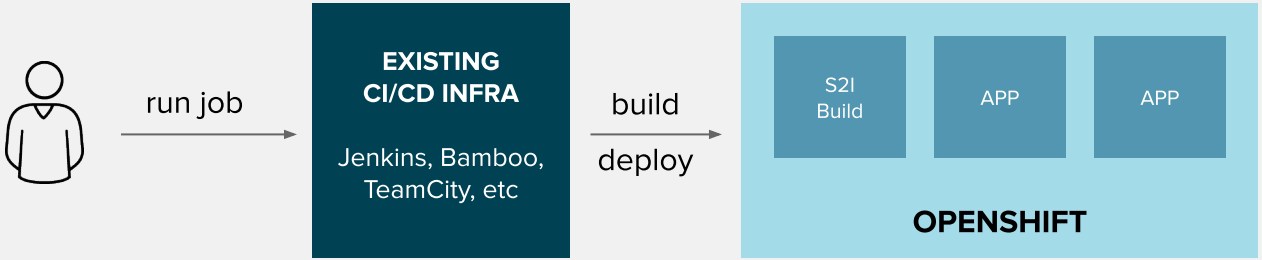
2) HYBRID JENKINS INFRA WITH OPENSHIFT

- 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 OPENSHIFT

- 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

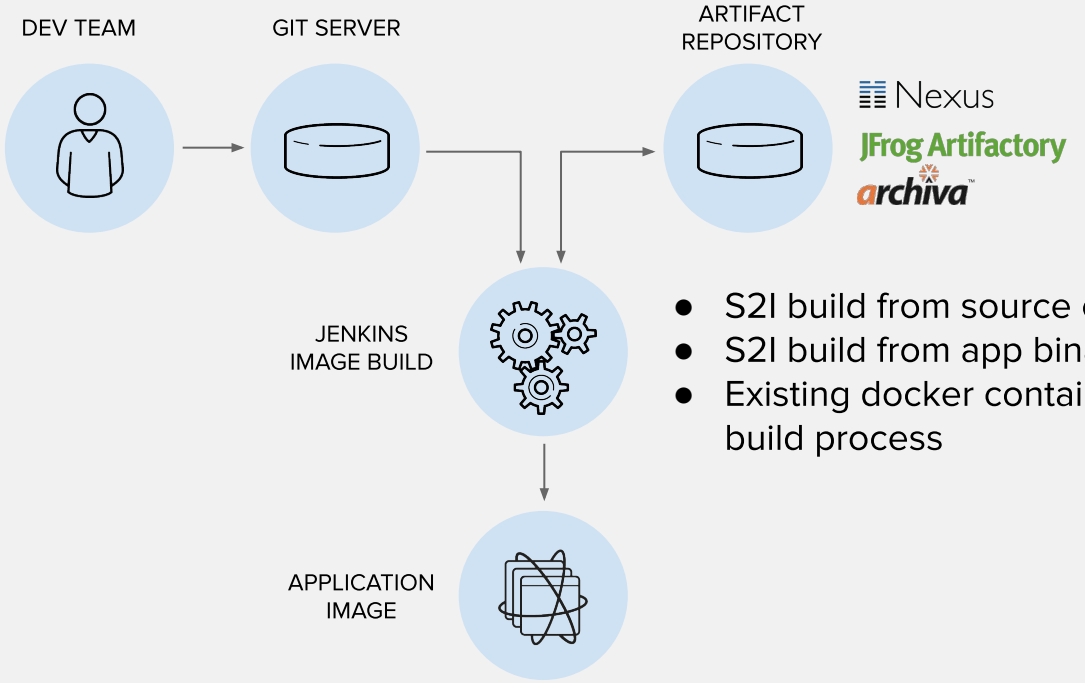
- OpenShift Pipelines allow defining a CI/CD workflow via a Jenkins pipeline which can be started, monitored, and managed similar to other builds
- Dynamic provisioning of Jenkins slaves
- Auto-provisioning of Jenkins server
- OpenShift Pipeline strategies
  - Embedded Jenkinsfile
  - Jenkinsfile from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

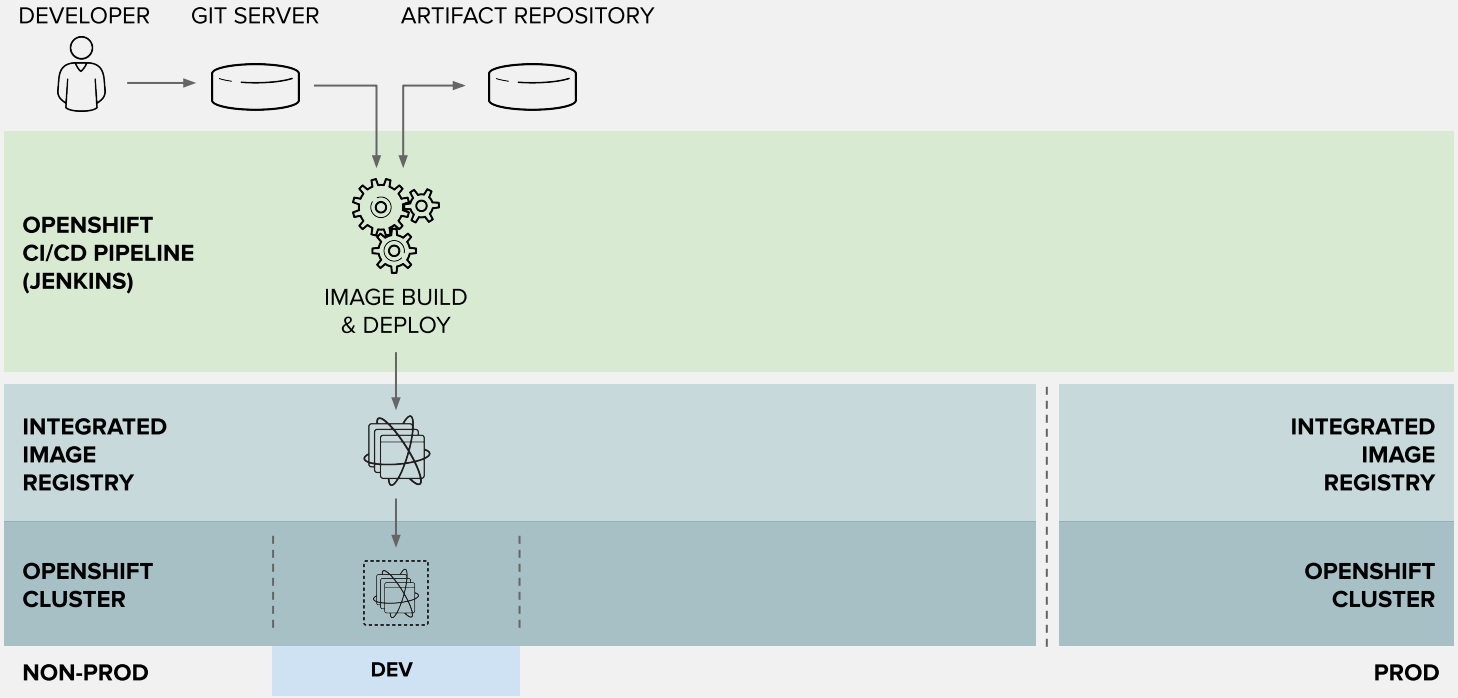
OpenShift Pipelines in Web Console

CONTINUOUS DELIVERY PIPELINE

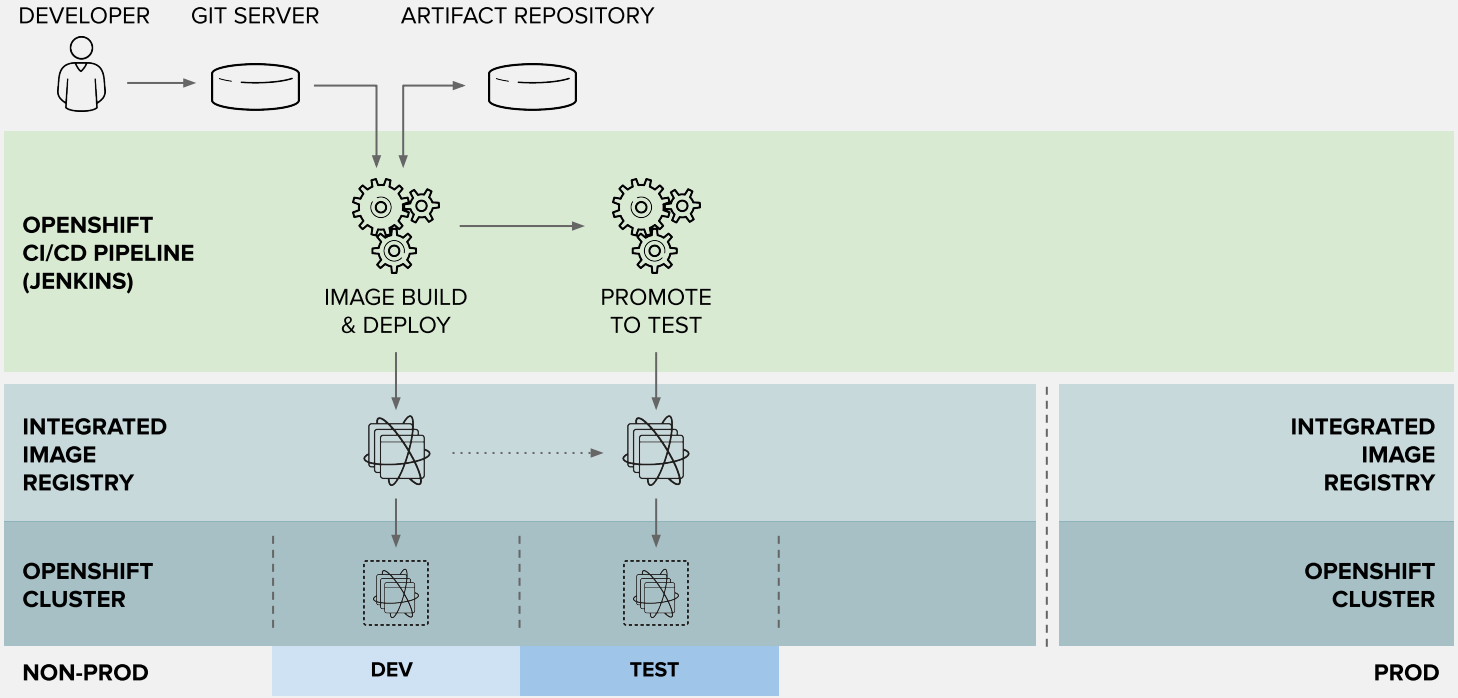


- S2I build from source code
- S2I build from app binary
- Existing docker container image build process

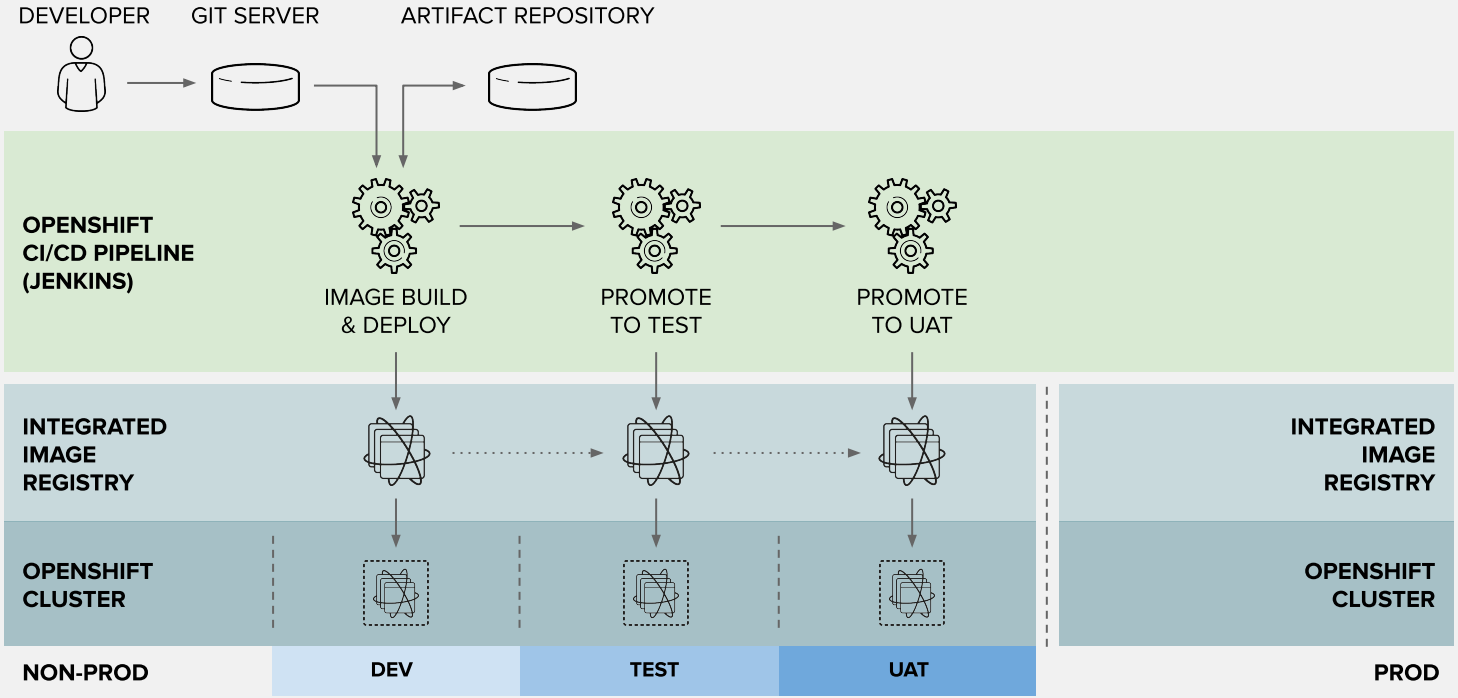
CONTINUOUS DELIVERY PIPELINE



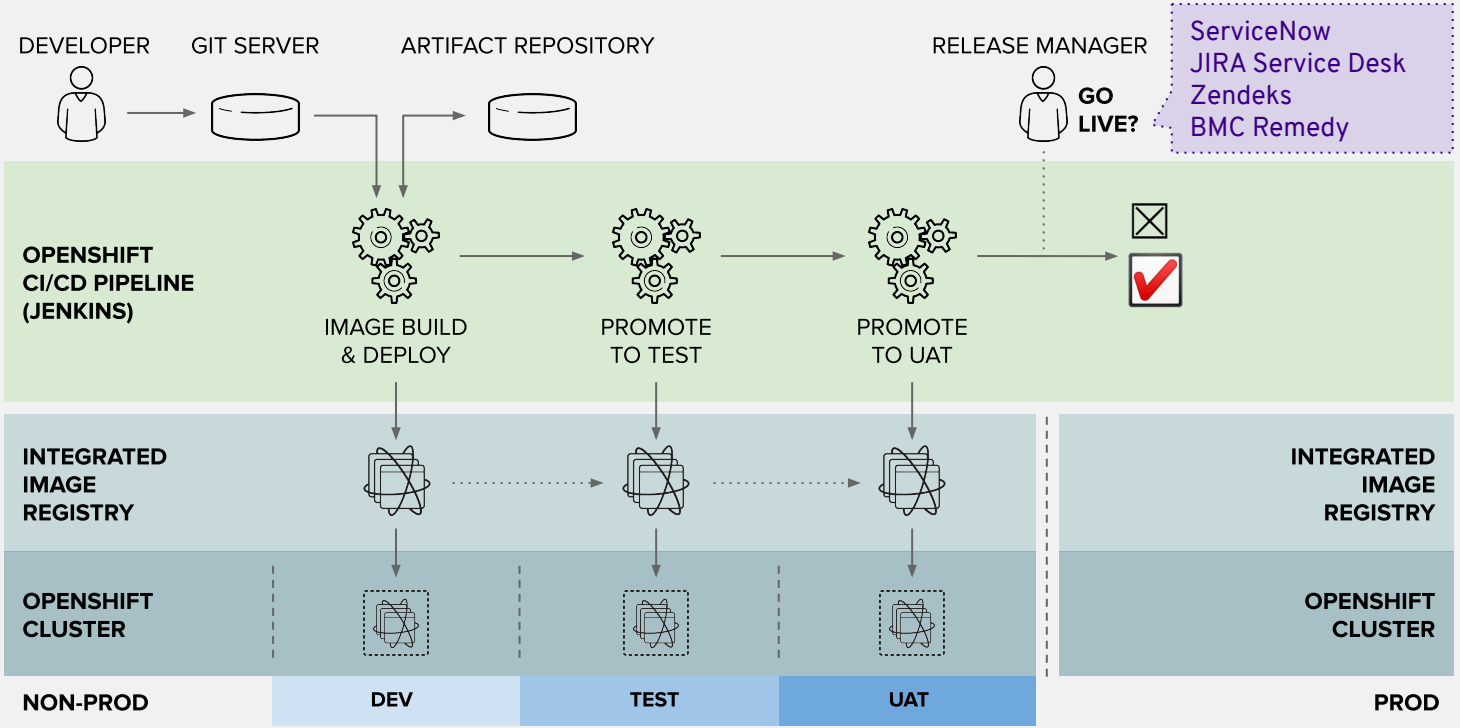
CONTINUOUS DELIVERY PIPELINE



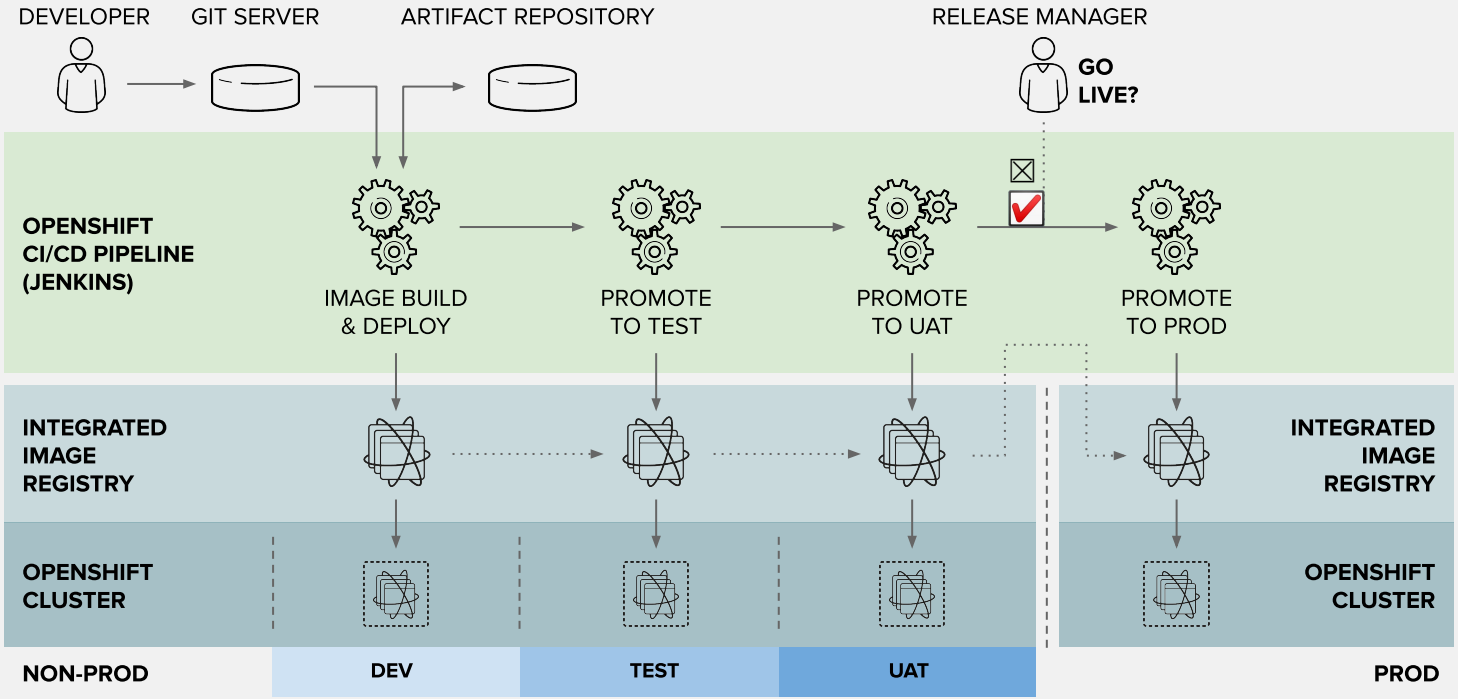
CONTINUOUS DELIVERY PIPELINE



CONTINUOUS DELIVERY PIPELINE

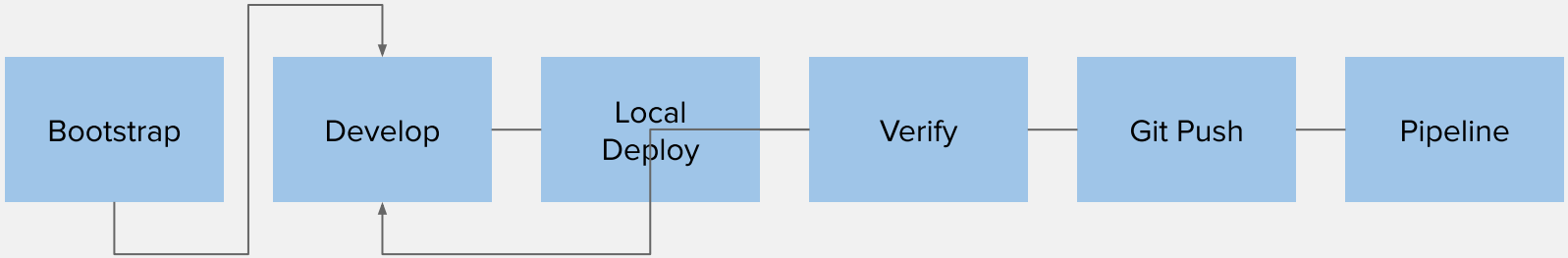


CONTINUOUS DELIVERY PIPELINE



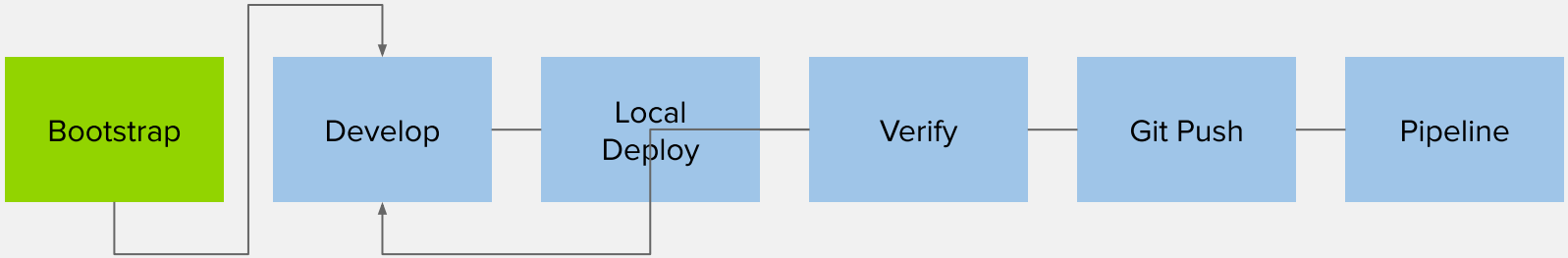
DEVELOPER WORKFLOW

LOCAL DEVELOPMENT WORKFLOW



RED HAT®
OPENSHIFT
Container Platform

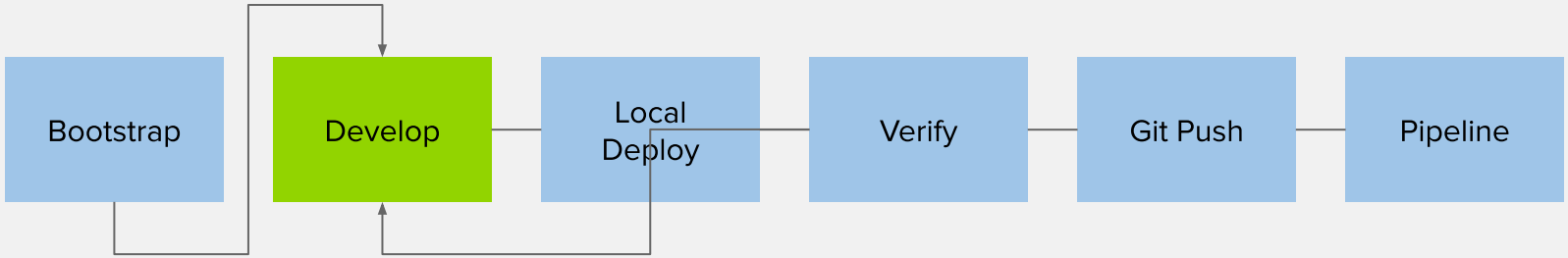
LOCAL DEVELOPMENT WORKFLOW



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
  - OpenShift Generator
  - Spring Initializr

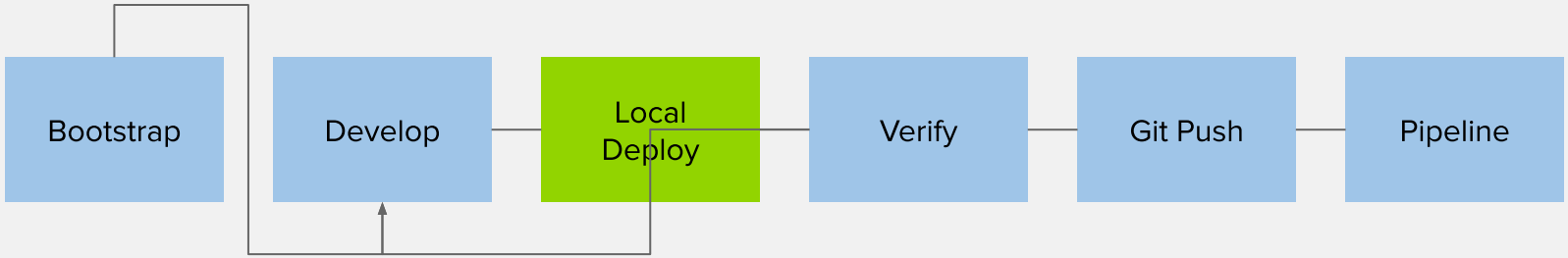
LOCAL DEVELOPMENT WORKFLOW



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

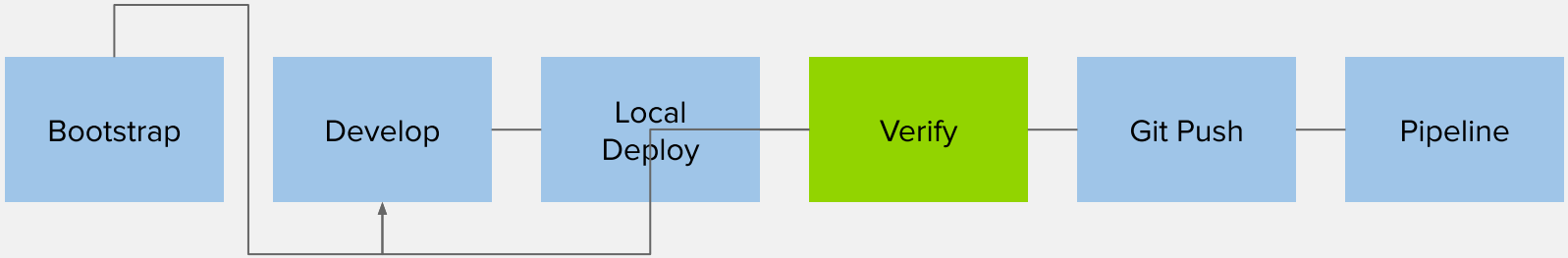
LOCAL DEVELOPMENT WORKFLOW



LOCAL DEPLOY

- Deploy your code on a local OpenShift cluster
  - Red Hat Container Development Kit (CDK), minishift and oc cluster
- Red Hat CDK provides a standard RHEL-based development environment
- Use binary deploy, maven or CLI rsync to push code or app binary directly into containers

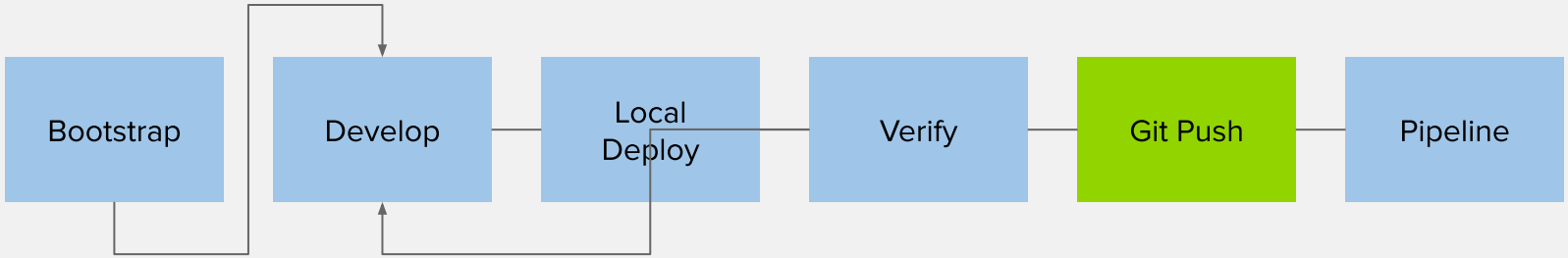
LOCAL DEVELOPMENT WORKFLOW



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

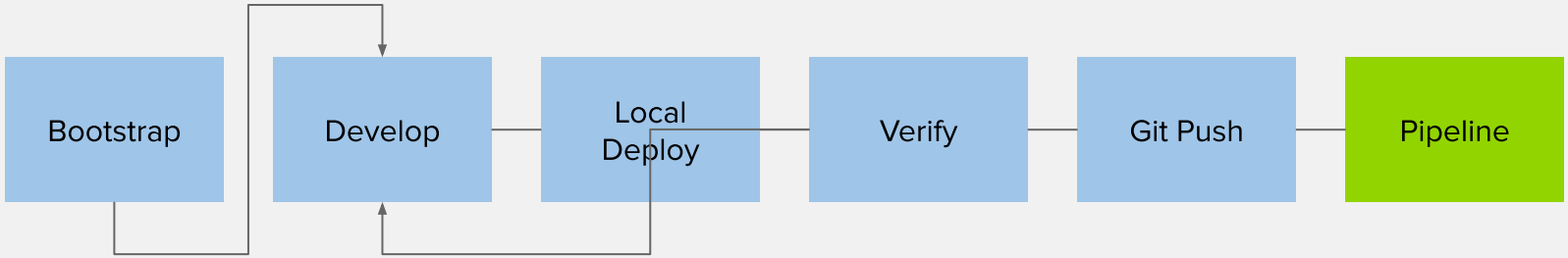
LOCAL DEVELOPMENT WORKFLOW



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

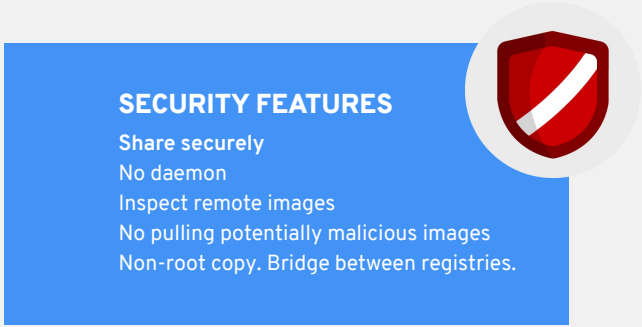


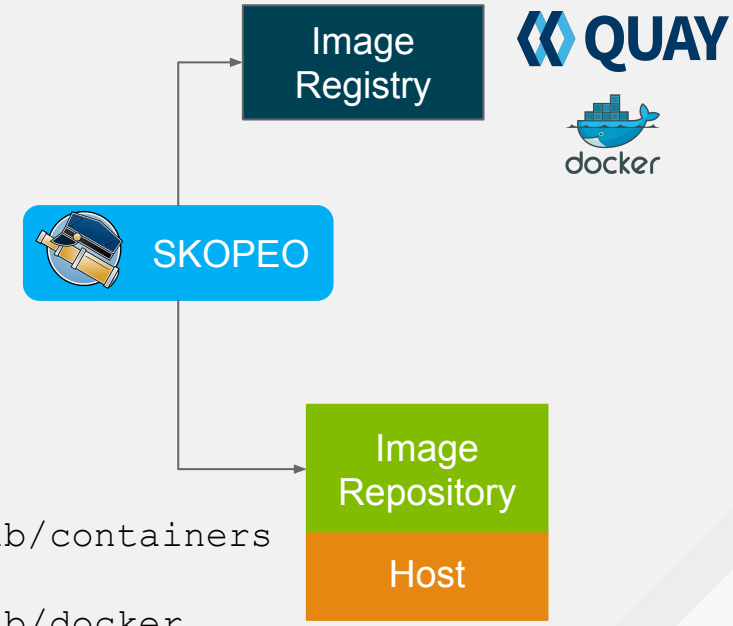
PIPELINE

- 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 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

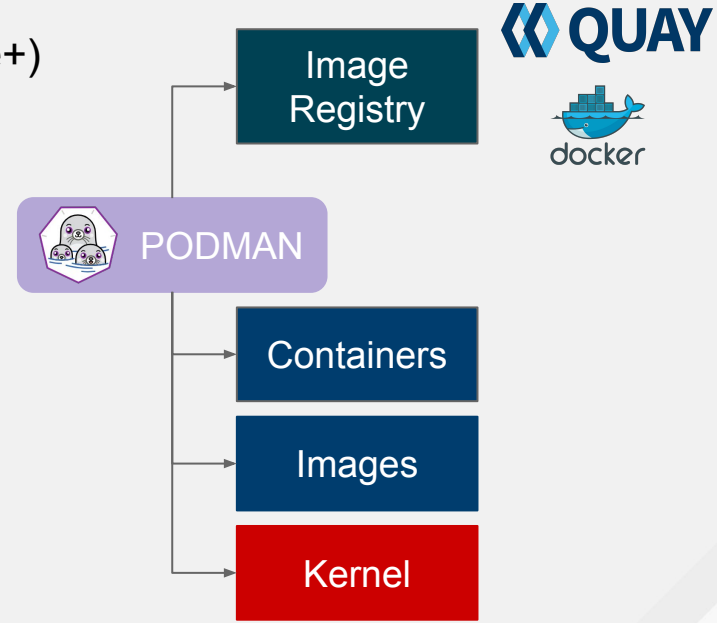


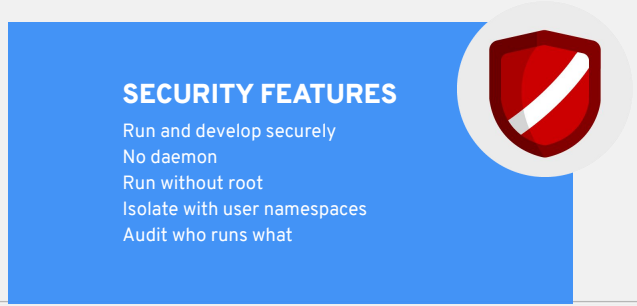


The new container CLI

podman

- @ podman.io
- 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!

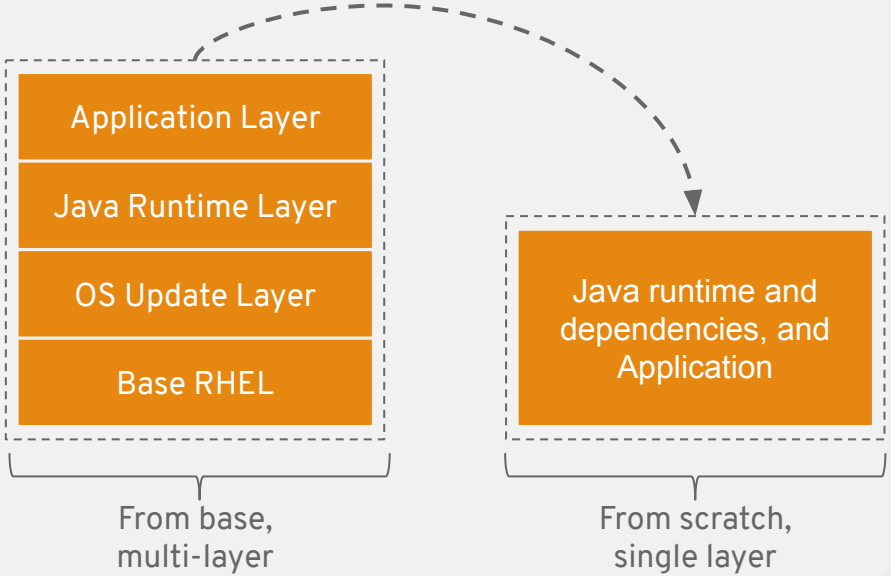




Why use Buildah?

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

7