

Kubernetes Native Developer

Architecture Workshop

CI/CD and GitOps



[linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)



[facebook.com/redhatinc](https://www.facebook.com/redhatinc)



[youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)



twitter.com/RedHat

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:

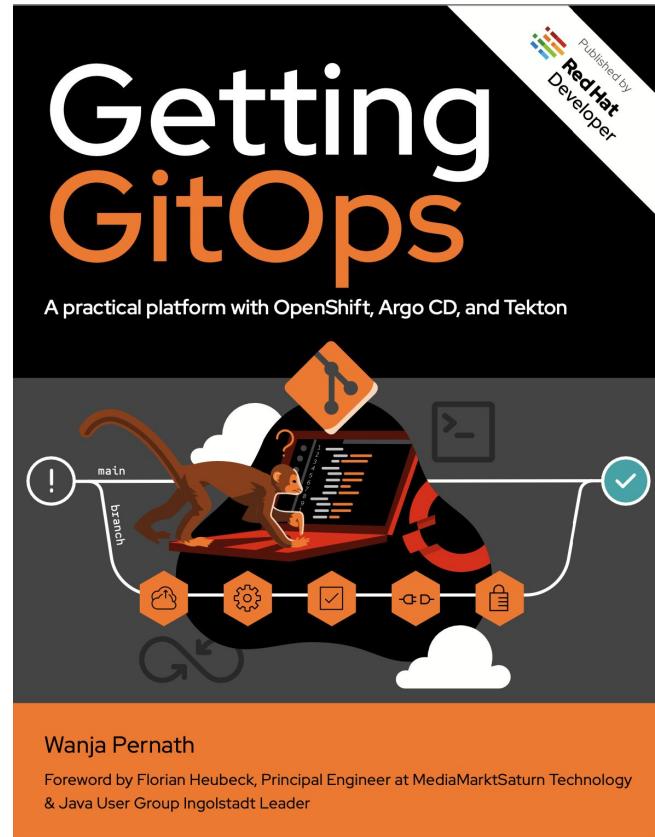
Quarkus, Helm Charts, Kustomize, Tekton Pipelines, Kubernetes Operators, OpenShift Templates, ArgoCD, CI/CD, GitOps....

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



Agenda

Agenda

- What is this all about
- The Use Case
- Quarkus Business Value
- Quarkus Technical Value
- First steps with Quarkus

What is this all about?

Learning Goals

- Benefits of using Quarkus
- Concepts of Quarkus
- Basic understanding
- Making use of Quarkus Dev Services
- Understanding and using Quarkus to build a microservice
- Developing Quarkus Apps with OpenShift
 - Using Source to Image
 - Understanding Serverless (Knative)
 - Deployment of Quarkus apps on OpenShift
- Using and understanding of Kustomize and Helm Charts
- Use OpenShift Pipelines (Tekton Pipelines) to do CI/CD
- Use of ArgoCD and OpenShift GitOps

OpenShift Developers Basics

Core Concepts

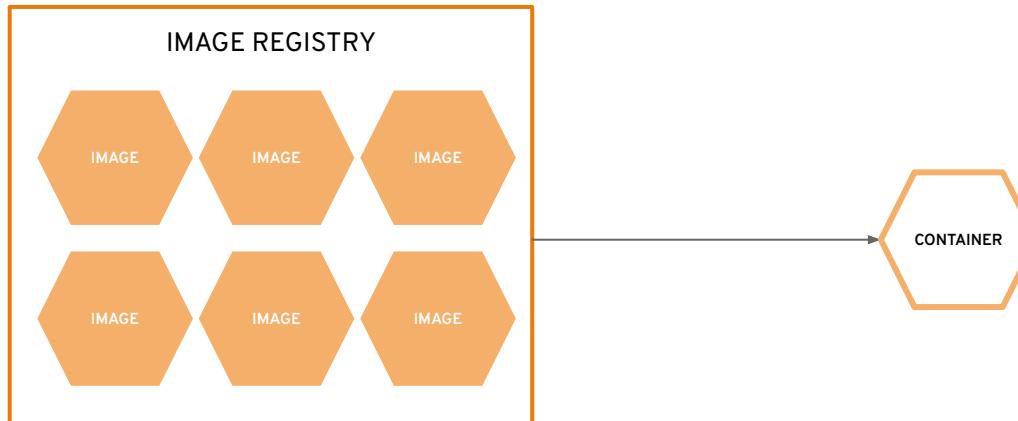
a container is the smallest compute unit



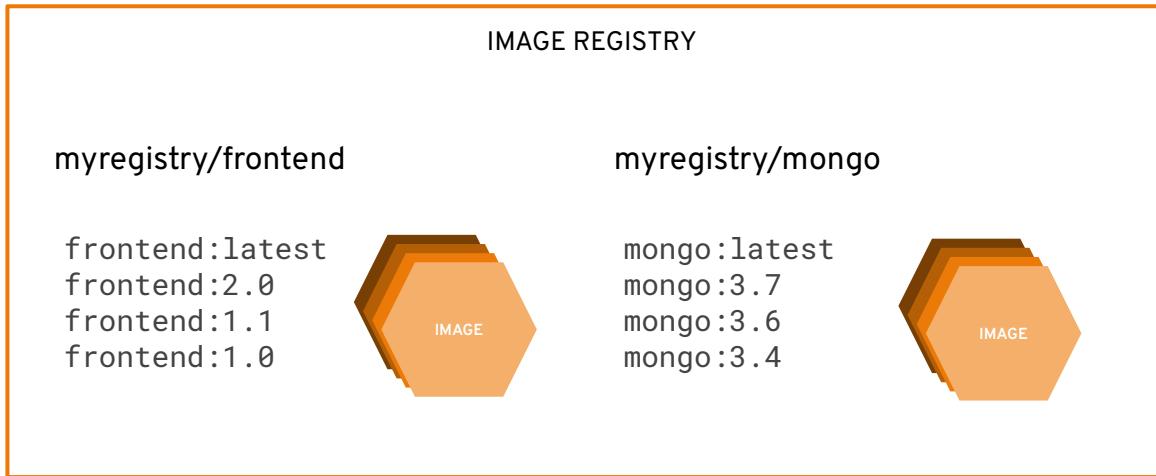
containers are created from container images



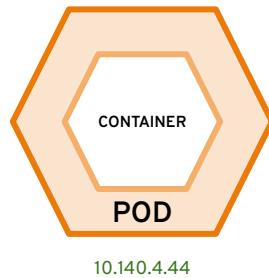
container images are stored in an image registry



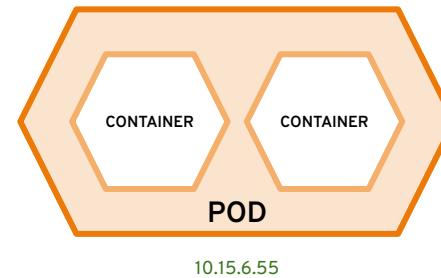
an image repository contains all versions of an image in the image registry



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

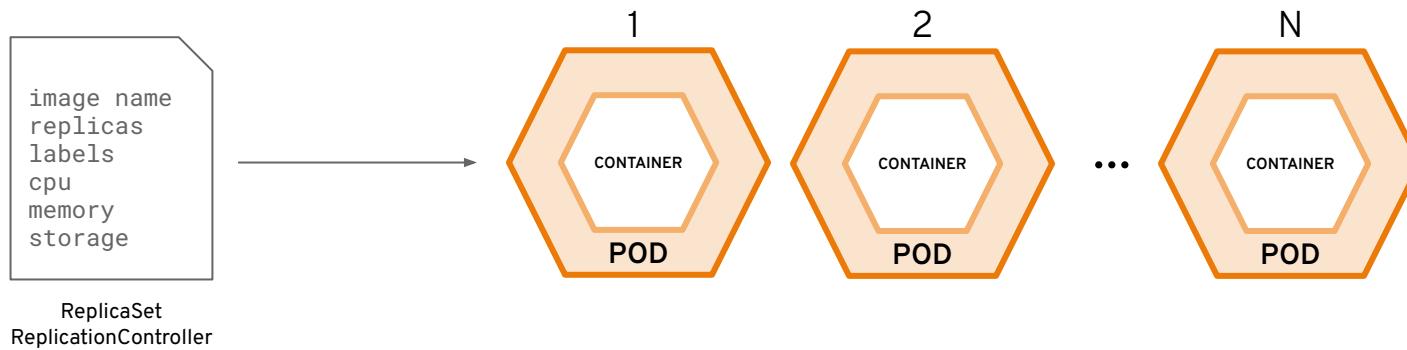


10.140.4.44

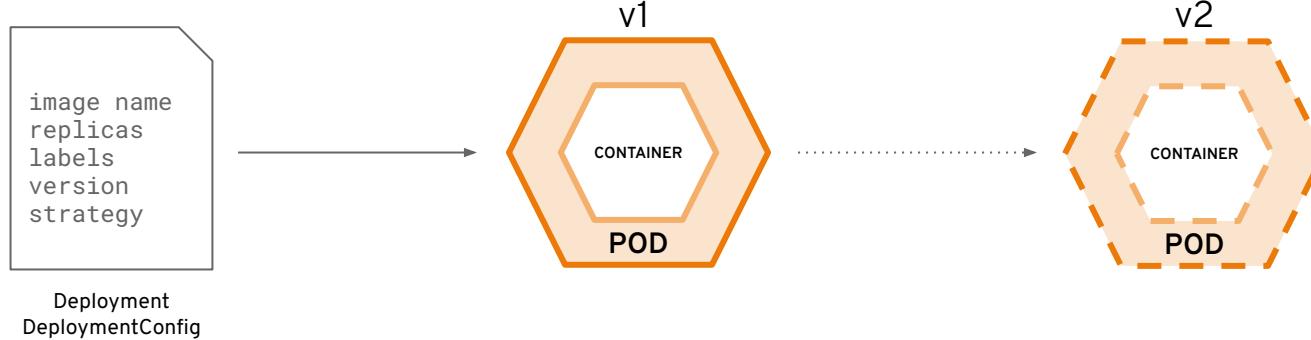


10.15.6.55

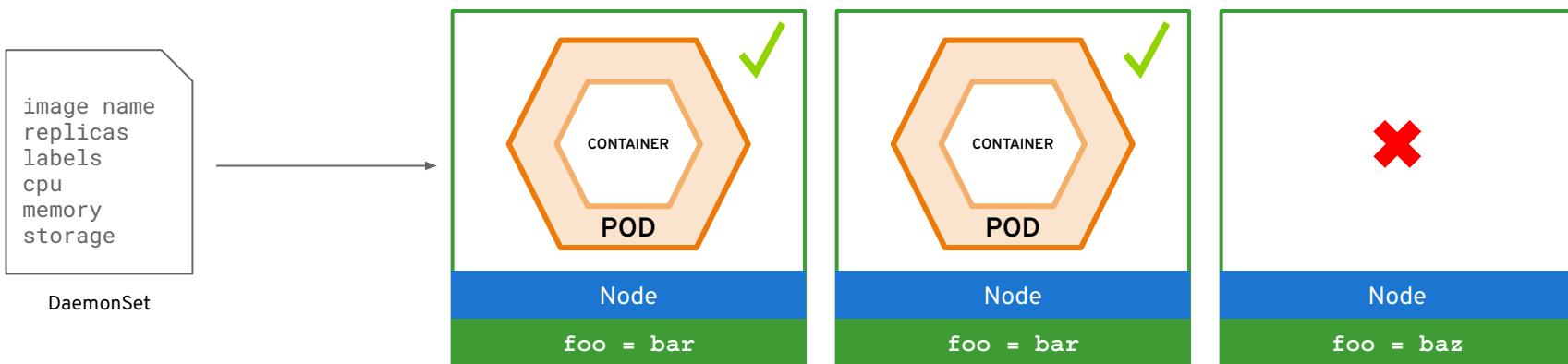
ReplicationControllers & ReplicaSets ensure a specified number of pods are running at any given time



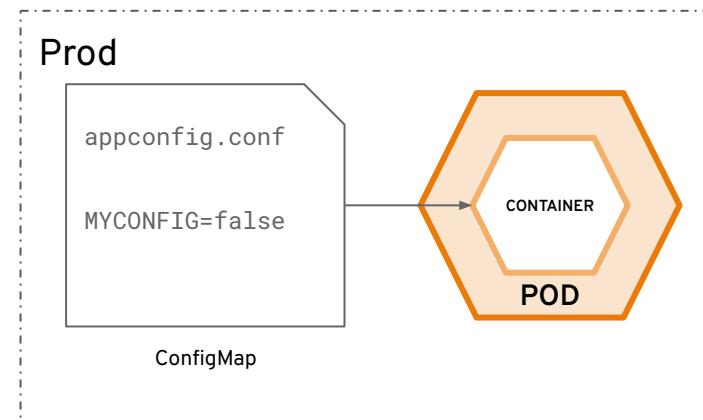
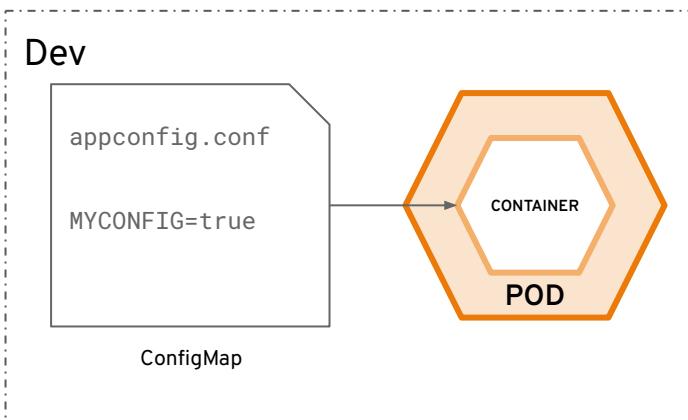
Deployments and DeploymentConfigurations define how to roll out new versions of Pods



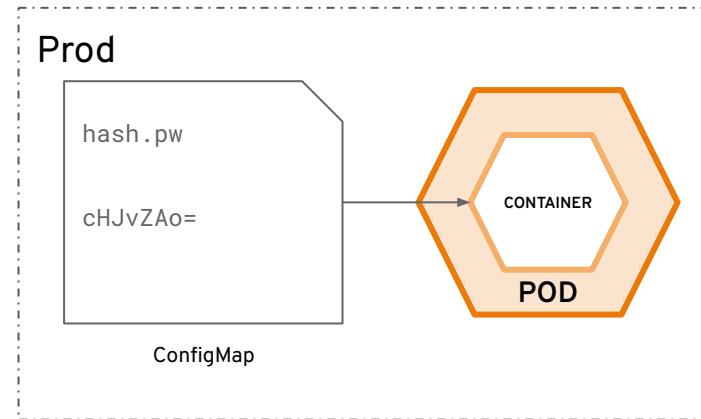
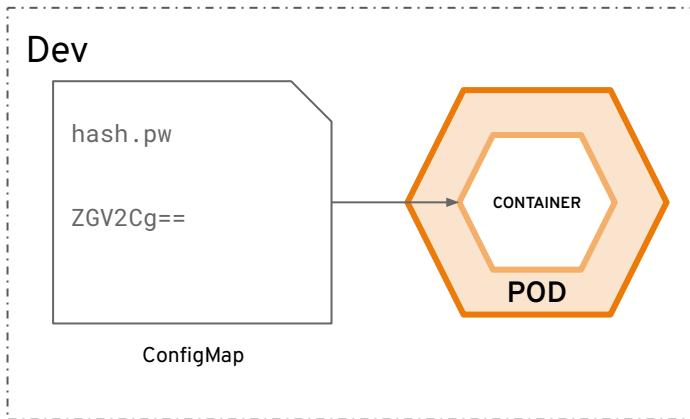
a daemonset ensures that all (or some) nodes run a copy of a pod



configmaps allow you to decouple configuration artifacts from image content

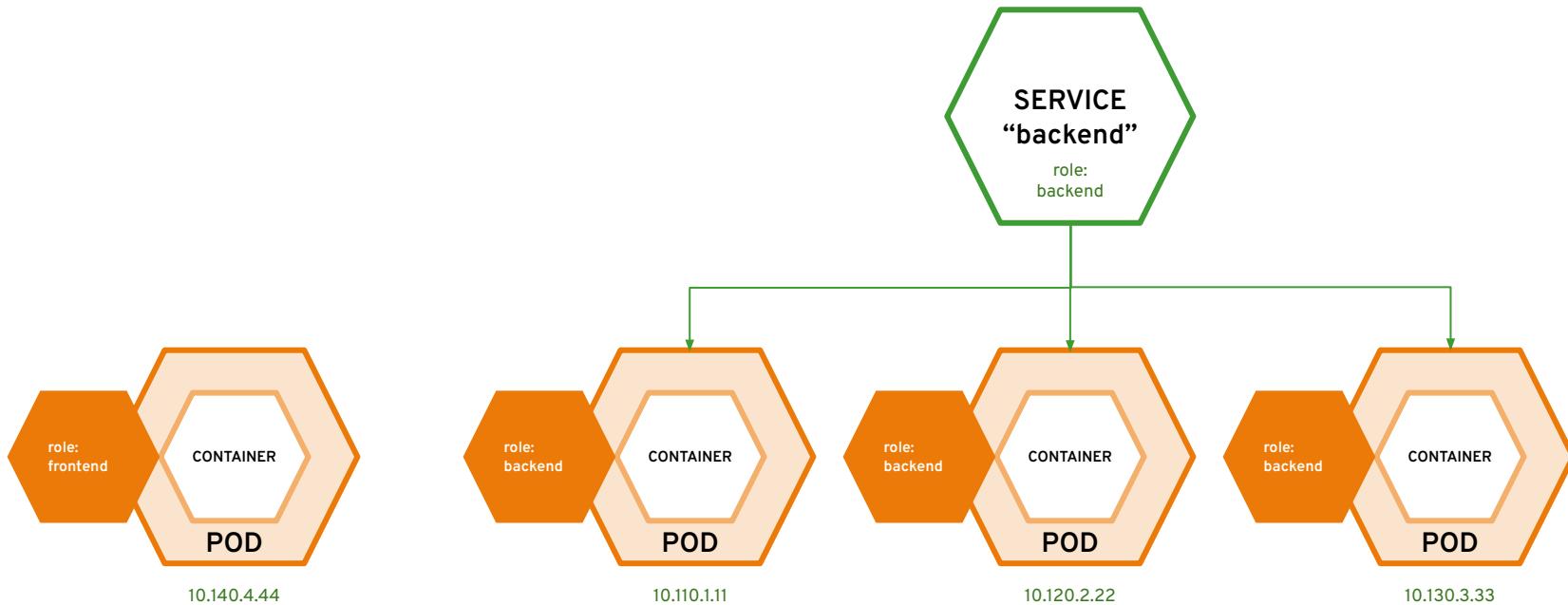


secrets provide a mechanism to hold sensitive information such as passwords

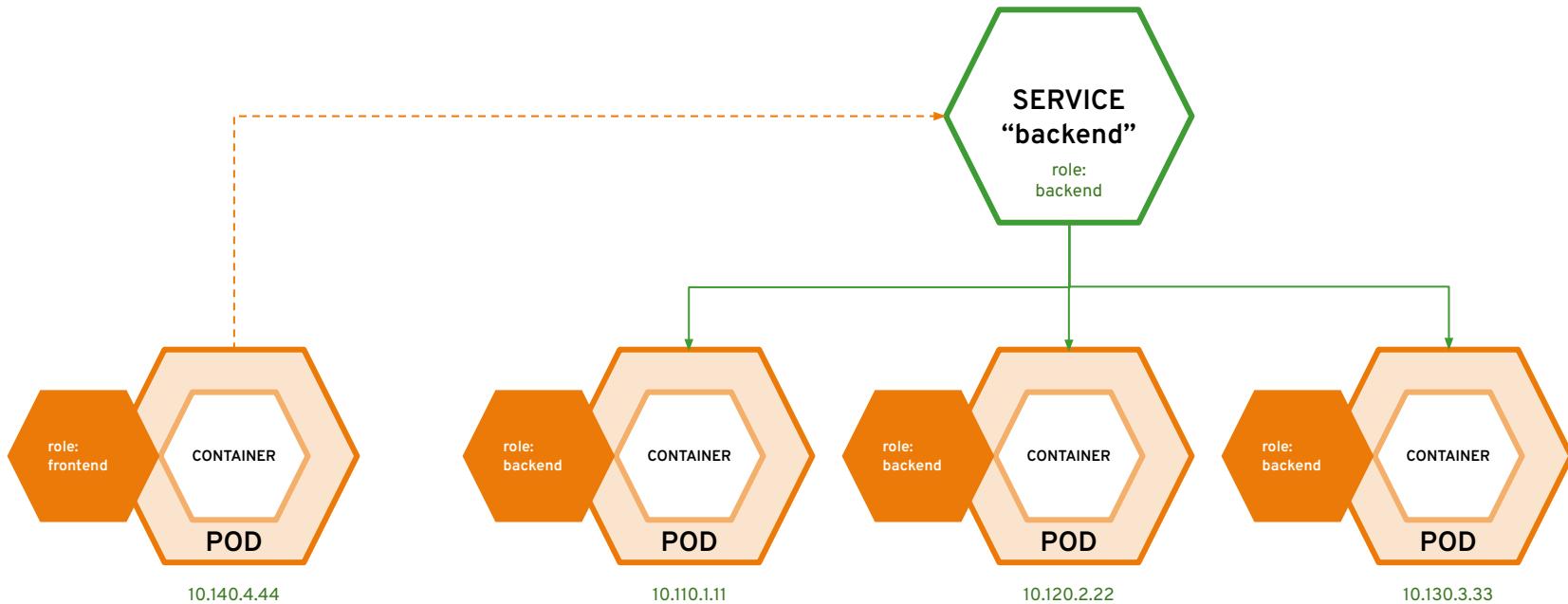


The etcd datastore can be encrypted for additional security
<https://docs.openshift.com/container-platform/4.6/security/encrypting-etcd.html>

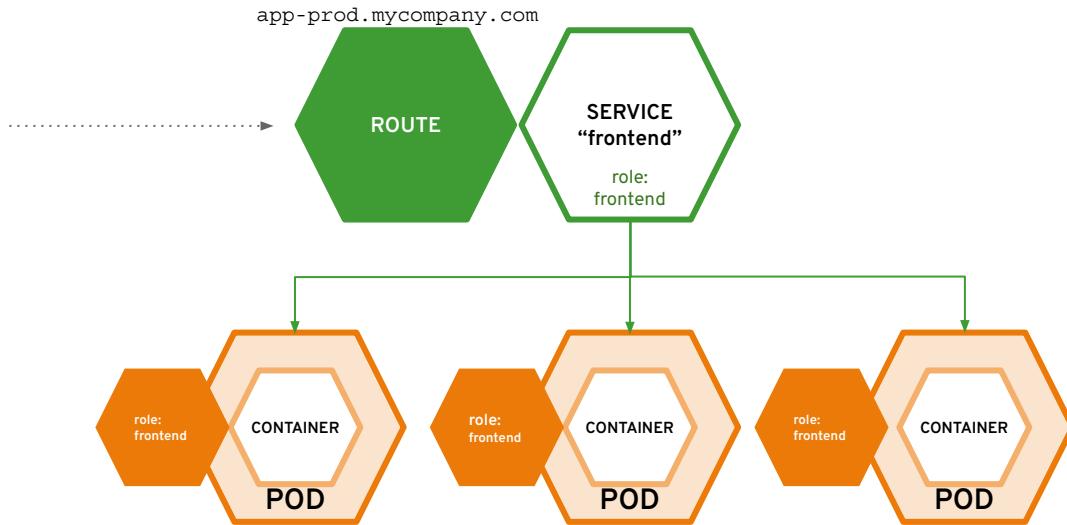
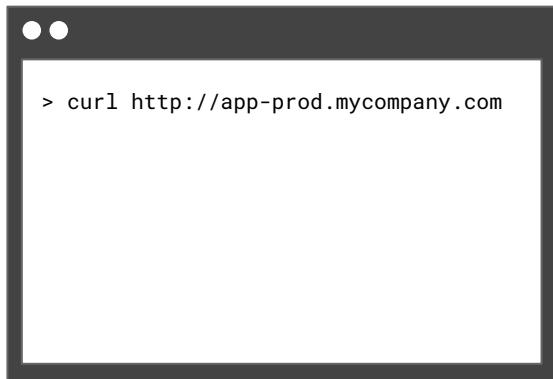
services provide internal load-balancing and service discovery across pods



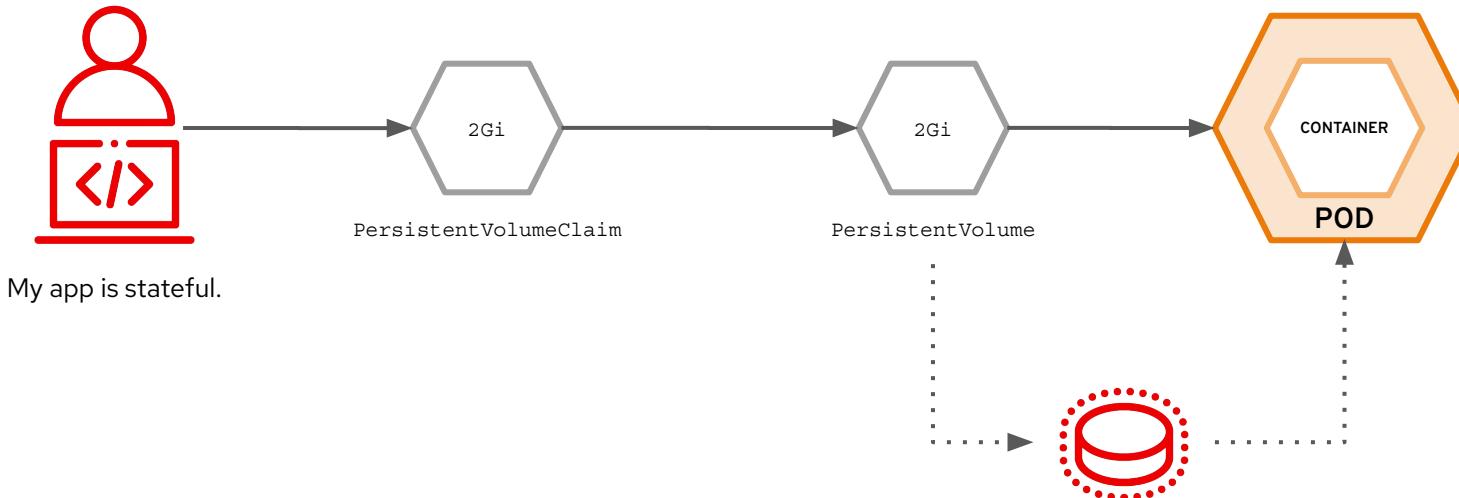
apps can talk to each other via services



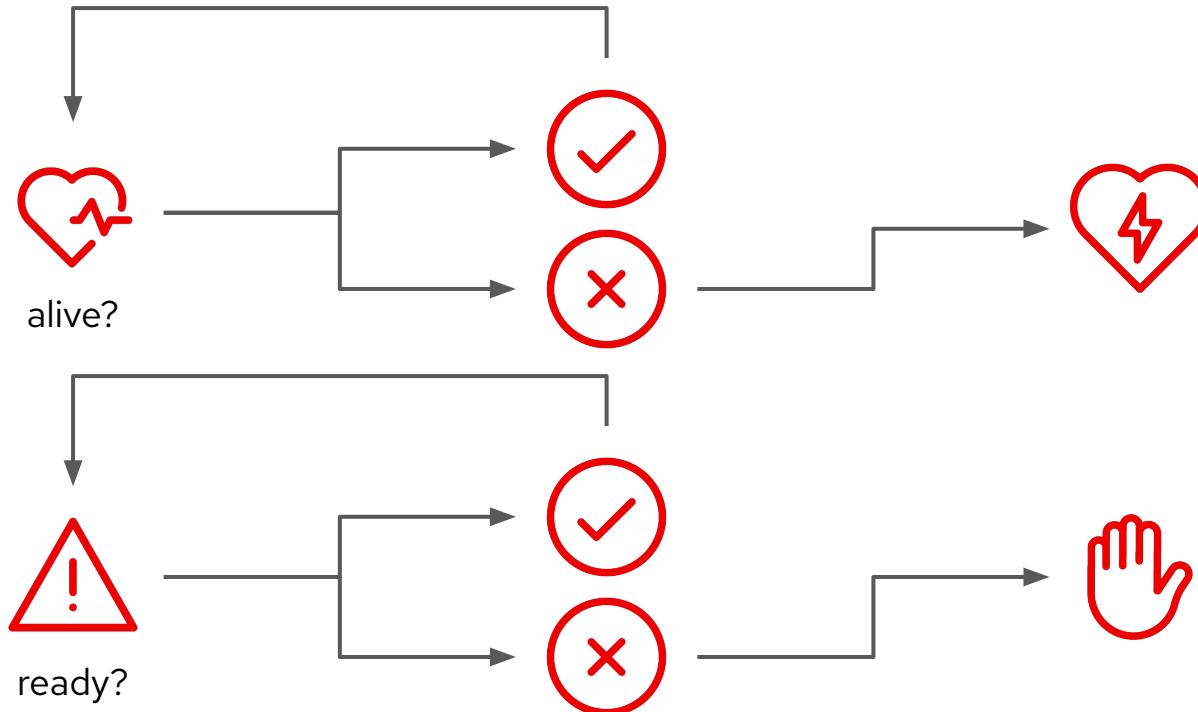
routes make services accessible to clients outside the environment via real-world urls



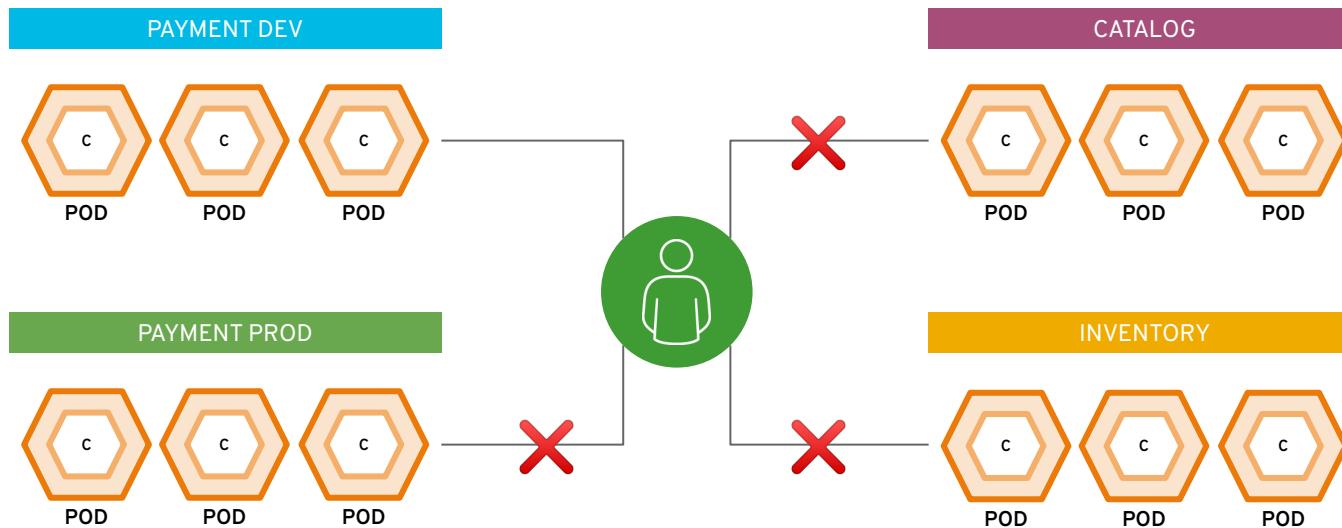
Persistent Volume and Claims



Liveness and Readiness

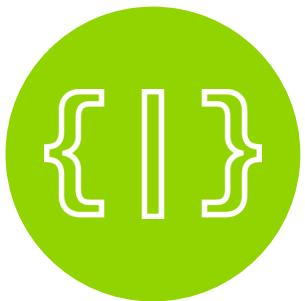


projects isolate apps across environments,
teams, groups and departments



BUILD & DEPLOY

BUILD AND DEPLOY CONTAINER IMAGES



DEPLOY YOUR
SOURCE CODE

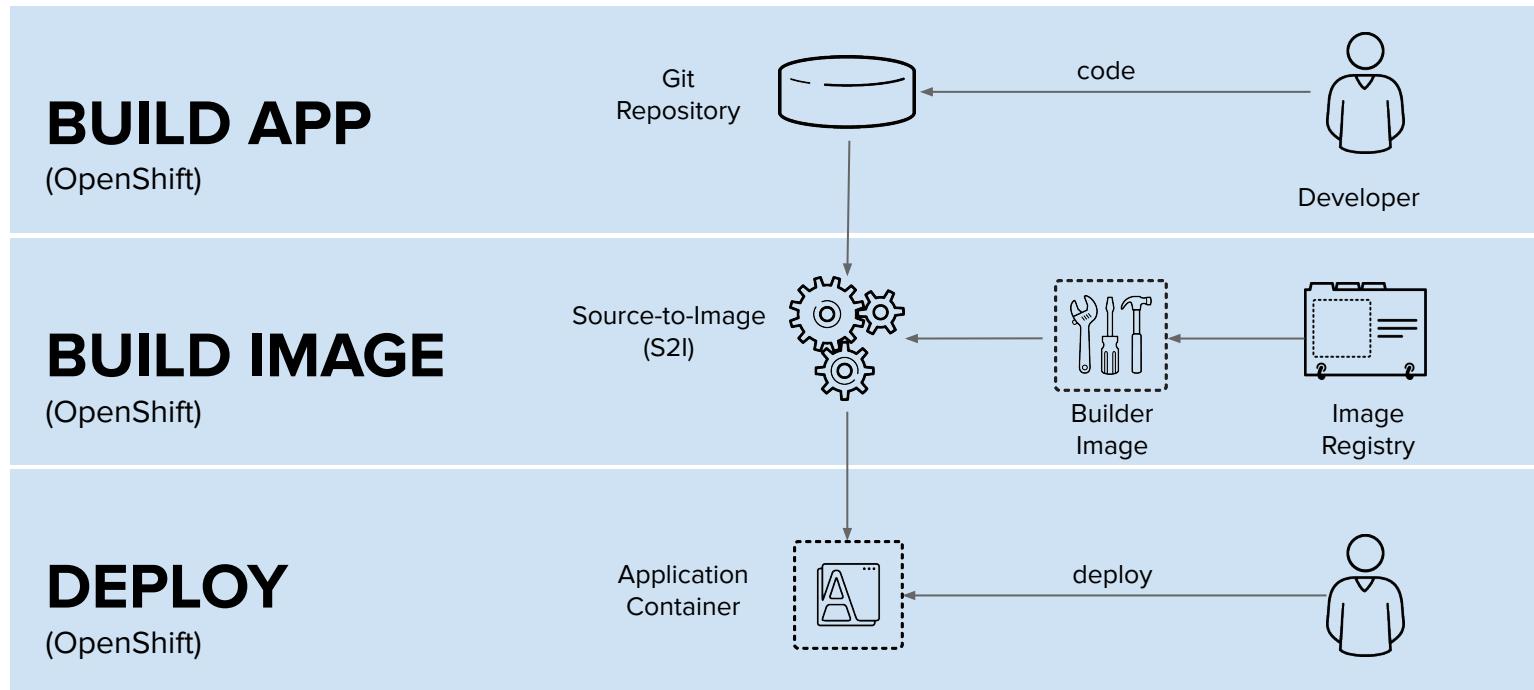


DEPLOY YOUR
APP BINARY

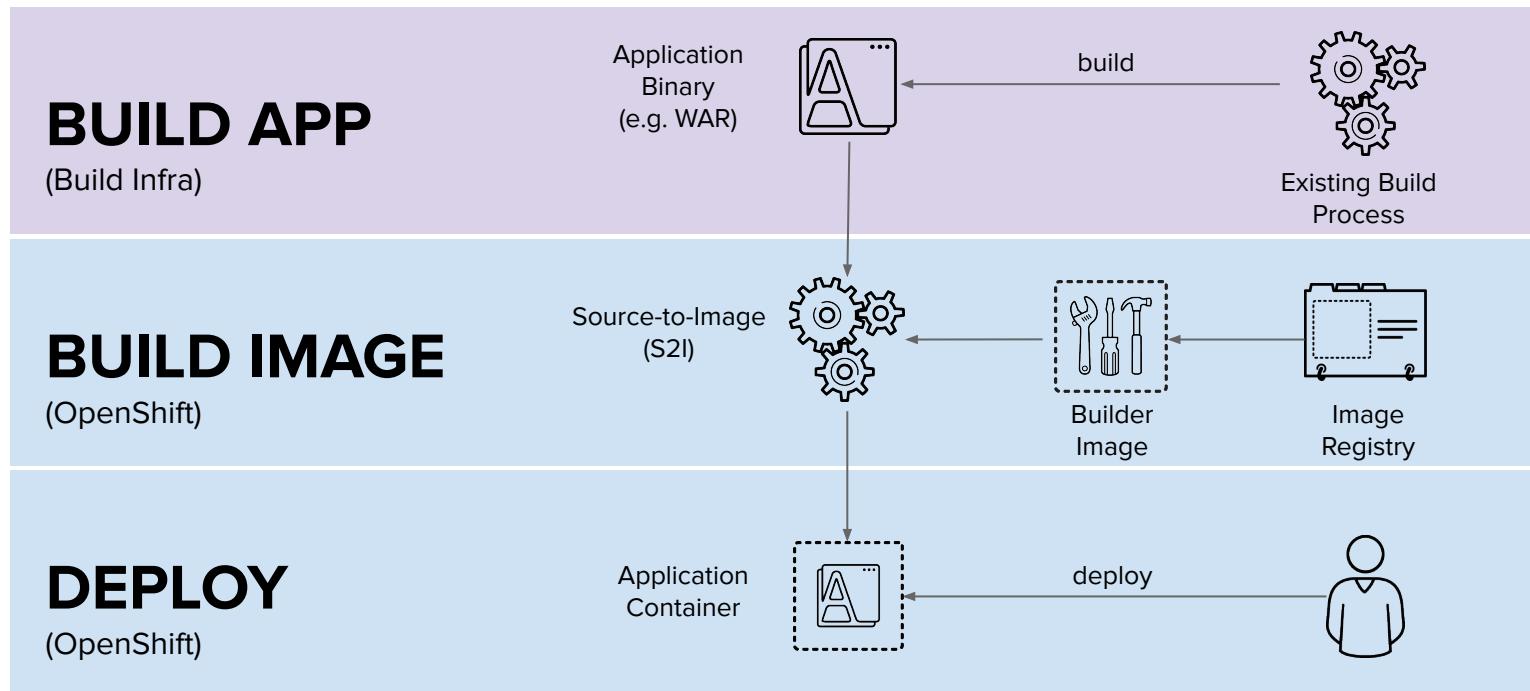


DEPLOY YOUR
CONTAINER IMAGE

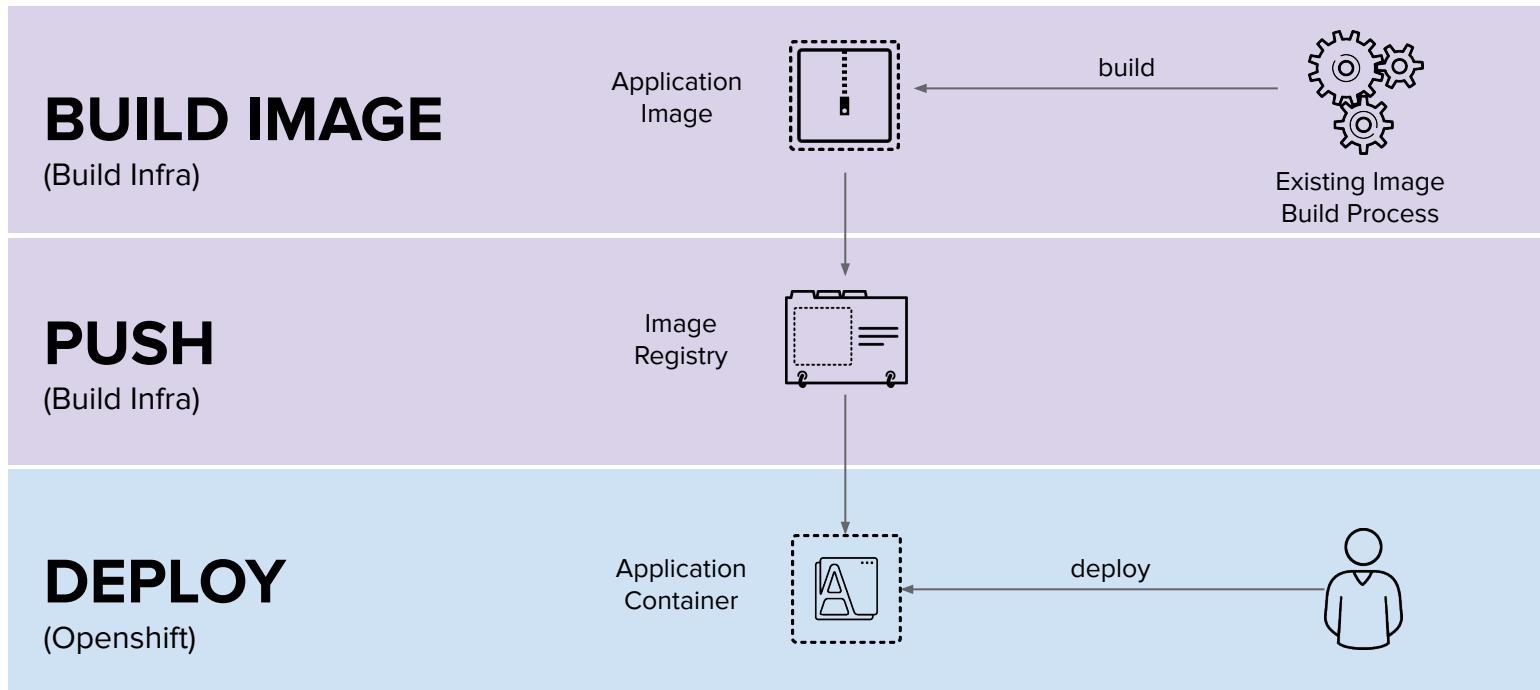
DEPLOY SOURCE CODE WITH SOURCE-TO-IMAGE (S2I)



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



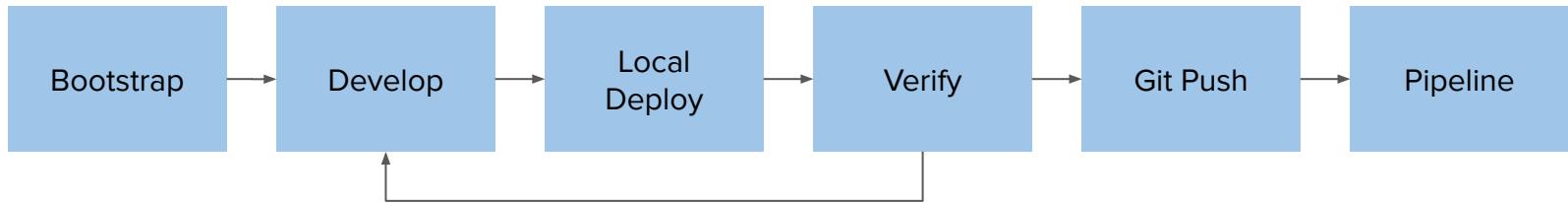
DEPLOY DOCKER IMAGE



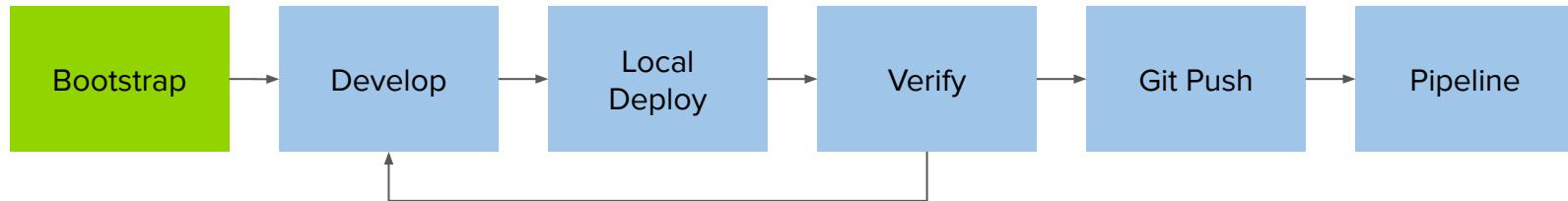
User/Tool Does OpenShift Does

DEVELOPER WORKFLOW

LOCAL DEVELOPMENT WORKFLOW



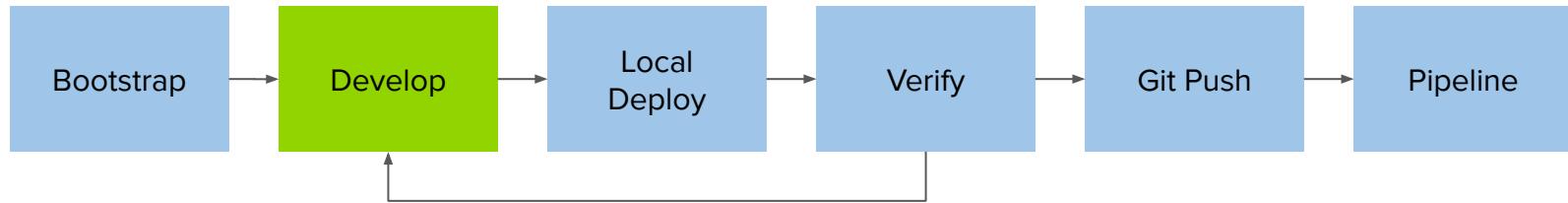
LOCAL DEVELOPMENT WORKFLOW



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
 - OpenShift Generator
 - Spring Initializr

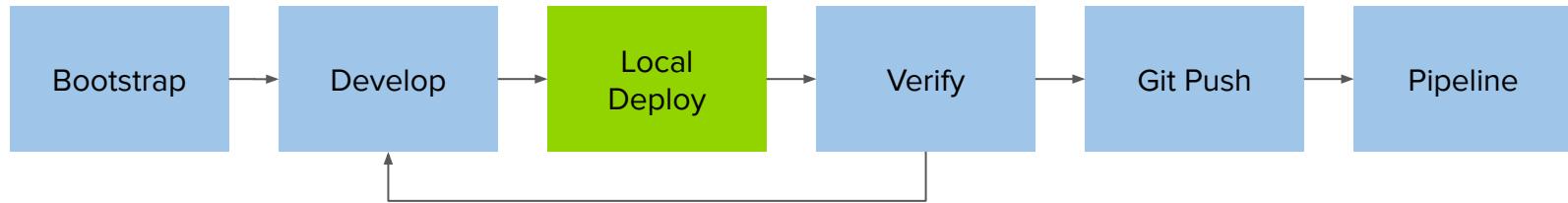
LOCAL DEVELOPMENT WORKFLOW



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

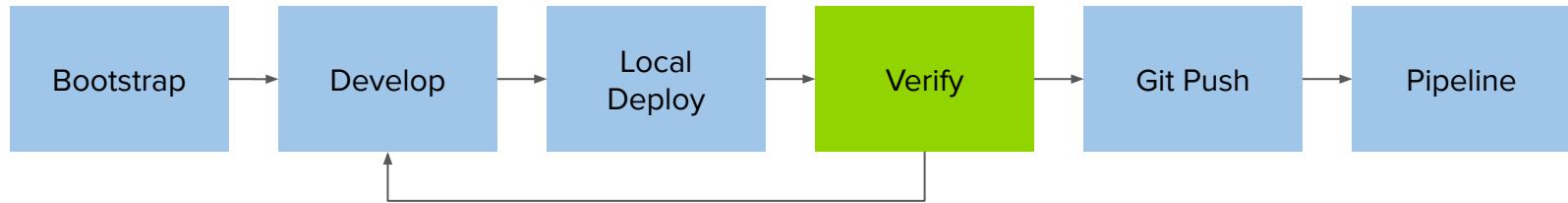
LOCAL DEVELOPMENT WORKFLOW



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

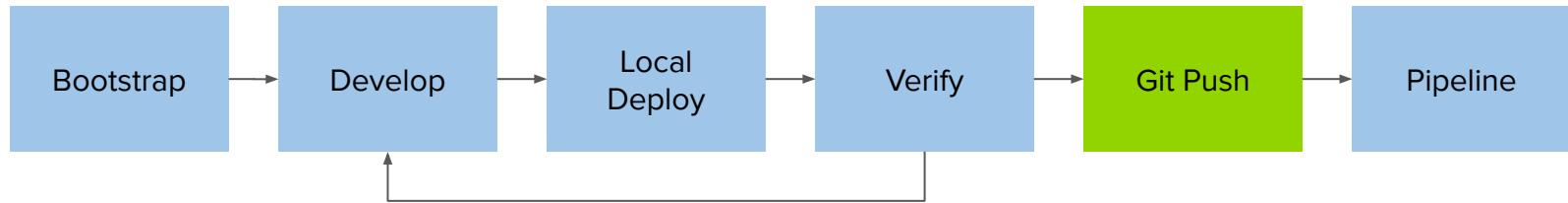
LOCAL DEVELOPMENT WORKFLOW



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

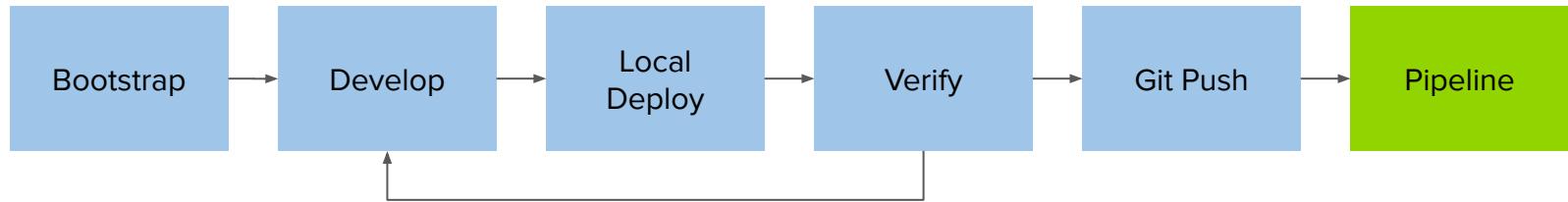
LOCAL DEVELOPMENT WORKFLOW



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

CI/CD with OpenShift

Basics

Wouldn't it be great, if everything could be automated?

Wouldn't it be great, if we simply do <insert hype here> and it solves all problems we have?

With DevOps we have a principle which solves most of the issues - but wait! The *DevOps Person* of the customer is currently on vacation



Basics - Developer

- Important: Separation of code and config!
- Writes the code of the App
- Writes set of build files (maven, gradle, etc.)
- Writes all needed tests (unit, integration, load)
- Writes Pipeline files
- Writes kubernetes manifest files
- Stores everything auditable in Git
- Finds a way and tools to combine all of the above

Basics - Operations

- Gets images, configs, test descriptions from Devs
- Adds own kubernetes manifests to the soup
- Makes sure, everything gets deployed
- Makes sure every dependency is installed and ready
- Thinks about security
- Thinks about networking
- Thinks about storage
- Thinks about compute power
- Thinks about plumbing everything together

Basics

CI/CD

=

Automation of Software Delivery

DevOps

=

Let's have Dev and Ops TALK to each other

Continuous Integration(CI) & Continuous Delivery (CD)

A key DevOps principle for automation, consistency and reliability



Tekton / OpenShift Pipelines

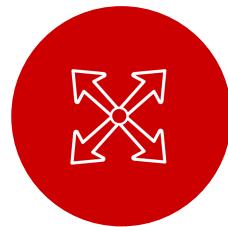
Kubernetes native on
demand delivery
pipelines

What is Cloud-Native CI/CD?



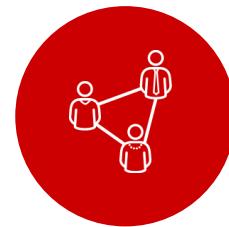
Containers

Built for container apps and runs on Kubernetes



Serverless

Runs serverless with no CI/CD engine to manage and maintain



DevOps

Designed with microservices and distributed teams in mind

Why Cloud-Native CI/CD?

Traditional CI/CD

Designed for Virtual Machines

Require IT Ops for CI engine maintenance

Plugins shared across CI engine

Plugin dependencies with undefined update cycles

No interoperability with Kubernetes resources

Admin manages persistence

Config baked into CI engine container

Cloud-Native CI/CD

Designed for Containers and Kubernetes

Pipeline as a service with no Ops overhead

Pipelines fully isolated from each other

Everything lifecycled as container images

Native Kubernetes resources

Platform manages persistence

Configured via Kubernetes ConfigMaps

Why Cloud-Native CI/CD?

Traditional CI/CD

Designed for Virtual Machines

Require IT Ops for CI engine maintenance



Plugins shared across CI engine
Jenkins
Plugins dependencies with undefined update cycles

No interoperability with Kubernetes resources

Admin manages persistence

Config baked into CI engine container

Cloud-Native CI/CD

Designed for Containers and Kubernetes

Pipeline as a service with no Ops overhead



Tasks fully isolated from each other
Everything literally has its own Docker Images

Native Kubernetes resources

Platform manages persistence

Configured via Kubernetes ConfigMaps

OpenShift Pipelines



Built for Kubernetes

Cloud-native pipelines taking advantage of Kubernetes execution and , operational model and concepts



Scale on-demand

Pipelines run and scale on-demand in isolated containers, with repeatable and predictable outcomes



Secure pipeline execution

Kubernetes RBAC and security model ensures security consistently across pipelines and workloads



Flexible and powerful

Granular control over pipeline execution details on Kubernetes, to support your exact requirements



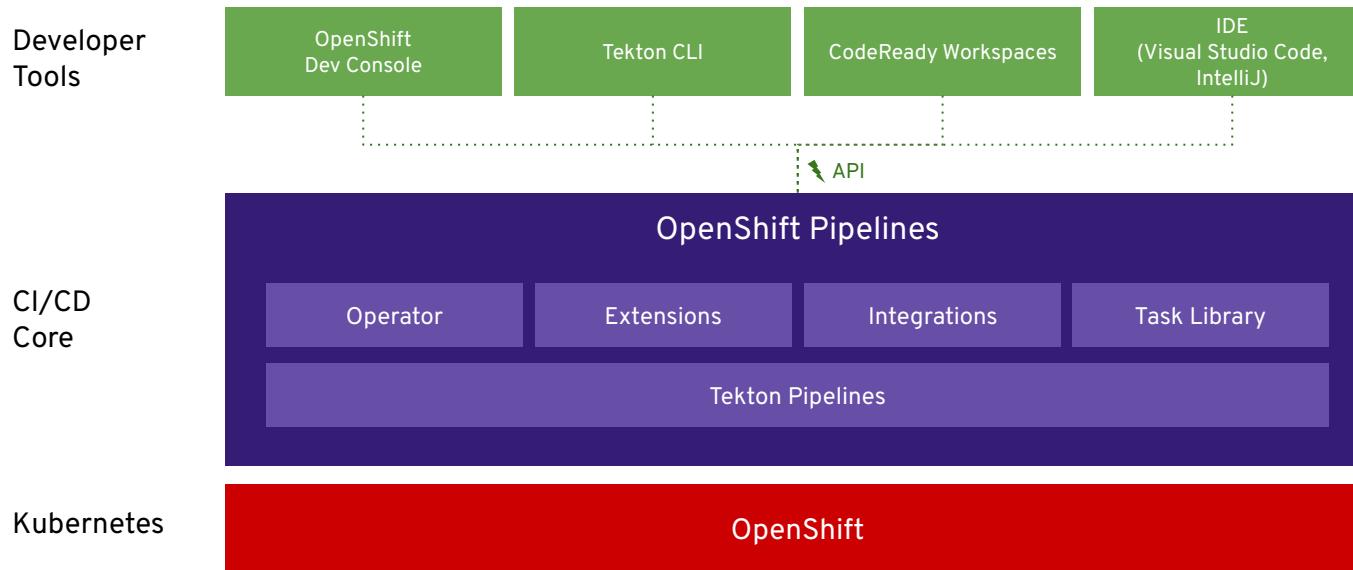
An open-source project for providing a set of shared and standard components for building Kubernetes-style CI/CD systems

51



Governed by the Continuous Delivery Foundation
Contributions from Google, Red Hat, Cloudbees, IBM, Pivotal and many more

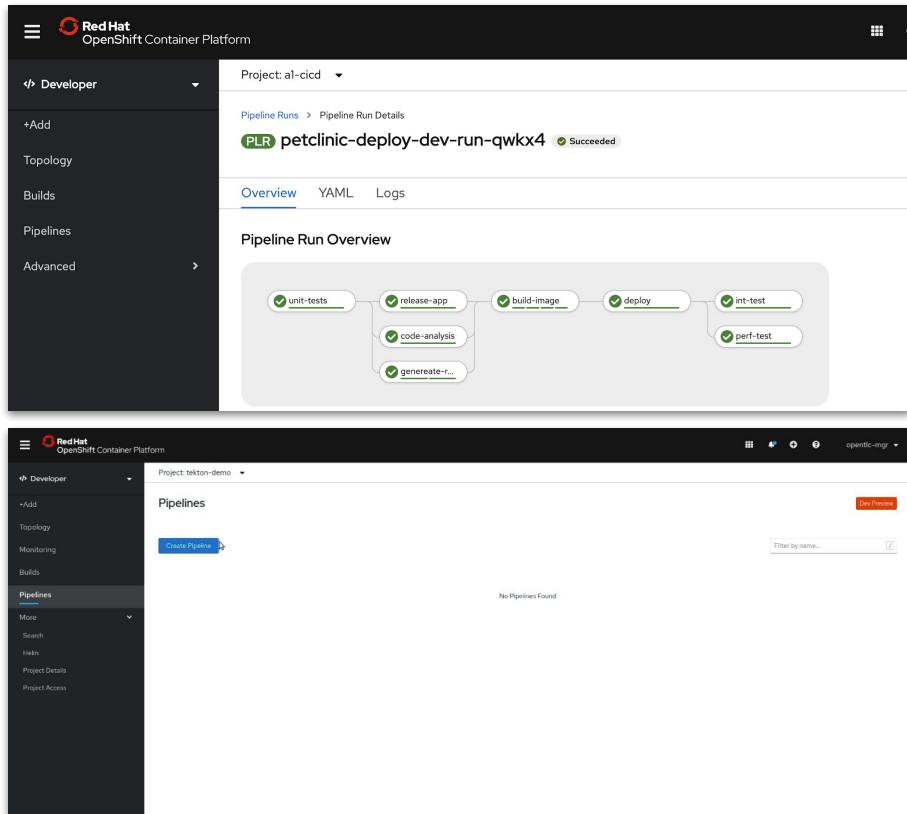
OpenShift Pipelines



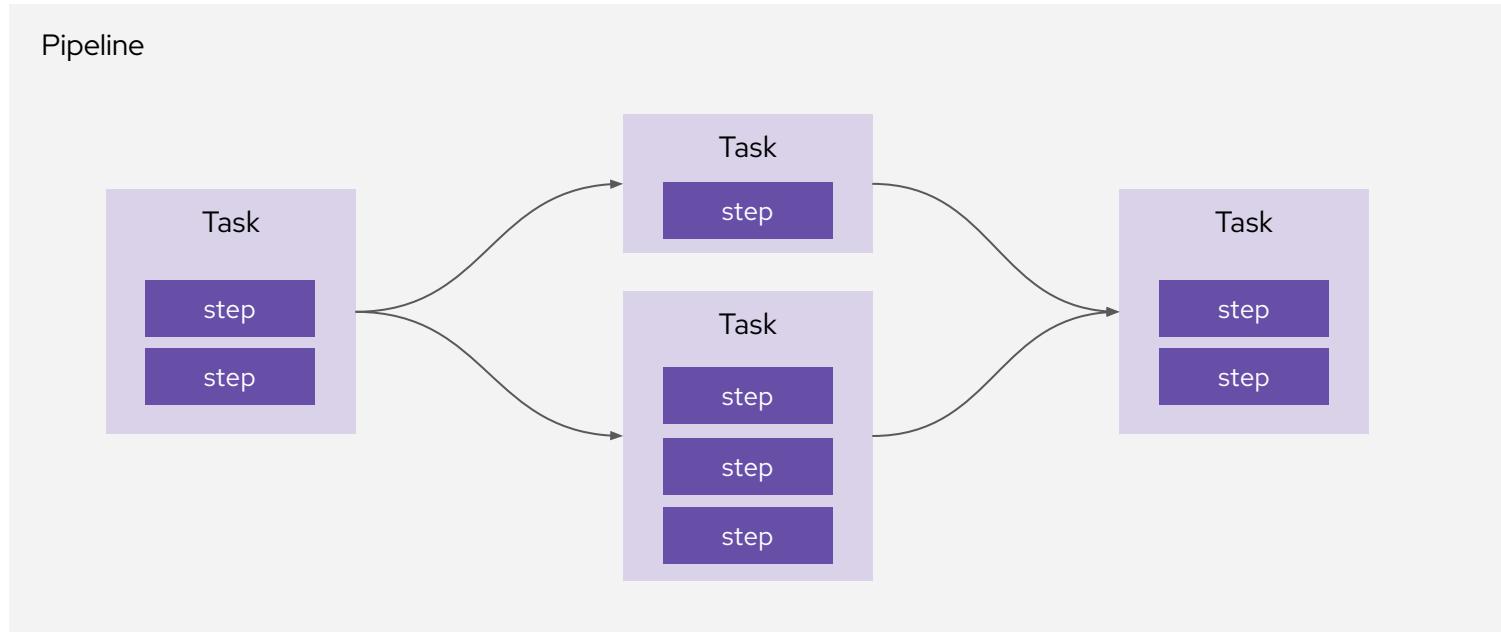
OpenShift Pipelines

- Based on Tekton Pipelines
- Kubernetes-native declarative CI/CD
- Pipelines run on-demand in isolated containers
- No central server to maintain! No plugin conflicts!
- Task library and integration with Tekton Hub
- Secure pipelines aligned with Kubernetes RBAC
- Visual and IDE-based pipeline authoring
- Pipeline templates when importing apps
- Automated install and upgrades via OperatorHub
- CLI, Web, VS Code and IntelliJ plugins

53



Tekton Concepts



Tekton Concepts: step

- Run command or script in a container
- Kubernetes container spec
 - Env vars
 - Volumes
 - Config maps
 - Secrets

```
- name: build  
image: maven:3.6.0-jdk-8-slim  
command: ["mvn"]  
args: ["install"]
```

```
- name: parse-yaml  
image: python3  
script: |-  
#!/usr/bin/env python3  
...
```

Tekton Concepts: Task

- Performs a specific task
- List of steps
- Steps run sequentially
- Reusable

```
kind: Task
metadata:
  name: buildah
spec:
  params:
    - name: IMAGE
  steps:
    - name: build
      image: quay.io/buildah/stable:latest
      command: ["buildah"]
      args: ["bud", ".", "-t", "${params.IMAGE}"]
    - name: push
      image: quay.io/buildah/stable:latest
      script: |
        buildah push ${params.IMAGE} docker://$(params.IMAGE)
```

Tekton Hub

Search, discover and install Tekton Tasks

57

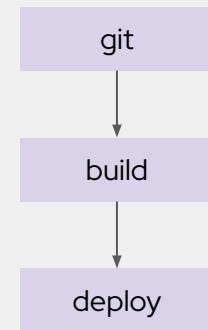
The screenshot shows the Tekton Hub interface, which is a platform for discovering and sharing reusable Tasks and Pipelines. The top navigation bar includes the Tekton Hub logo, a search bar, and a 'Login' button. Below the header, a welcome message reads 'Welcome to Tekton Hub' and 'Discover, search and share reusable Tasks and Pipelines'. A sidebar on the left provides filtering options for 'Refine By': Kind (Task, Pipeline), Support Tier (Official, Verified, Community), and Categories (Build Tools, CLI, Cloud, Deploy, Image Build, Notification, Others, Test Framework). The main content area displays a grid of task cards, each with a name, version, rating, description, update time, and tags.

Name	Version	Rating	Description	Last Updated	Tags
Ansible Runner	v0.1	4.5	Task to run Ansible playbooks using Ansible Runner	Updated 3 weeks ago	ansible, cli
ansible tower cli	v0.1	2.0	Ansible-tower-cli task simplifies starting jobs, workflow jobs, manage users, projects etc.	Updated 3 weeks ago	ansible, cli
argocd	v0.1	3.0	This task syncs (deploys) an Argo CD application and waits for it to be healthy.	Updated 3 weeks ago	deploy
aws cli	v0.1	5.0	This task performs operations on Amazon Web Services resources using aws.	Updated 3 weeks ago	cli
Amazon ECR Login	v0.1	4.0	This task retrieves an 'authentication token' using the GetAuthorizationToken API that you can use to authenticate to an...	Updated 3 weeks ago	aws, ecr
azure cli	v0.1	1.0	This task performs operations on Microsoft Azure resources using az.	Updated 4 months ago	cli
bentoml	v0.1	0.0	This task allows operations on BentoML services	Updated 3 weeks ago	cli
Python Black	v0.1	0.0	This task can be used to format Python code	Updated 3 weeks ago	formatter, python

Tekton Concepts: Pipeline

- A graph of Tasks: concurrent & sequential
- Tasks run on different nodes
- Task execution logic
 - Conditional
 - Retries
- Share data between tasks

```
kind: Pipeline
metadata:
  name: deploy-dev
spec:
  params:
    - name: IMAGE_TAG
  tasks:
    - name: git
      taskRef:
        name: git-clone
      params: [...]
    - name: build
      taskRef:
        name: maven
      params: [...]
      runAfter: ["git"]
    - name: deploy
      taskRef:
        name: knative-deploy
      params: [...]
      runAfter: ["build"]
```



Tekton CLI, Visual Studio Code, and IntelliJ

The screenshot shows the Tekton CLI interface with two main panes. The left pane displays the YAML configuration for a pipeline named 'sample-test-pipeline'. The right pane shows a visual representation of the pipeline's execution flow, starting from a task labeled 'scaffold-unit-tests' which branches into two parallel tasks: 'build-skaffold-web' and 'build-skaffold-app'. These two tasks then each branch into 'deploy-web' and 'deploy-app' respectively. The middle section of the interface shows a detailed view of a 'Task: buildah' entry, which is a Tekton Pipeline task. It includes a description, a YML file preview, and a list of parameters and workspace details.

```
sample-test-pipeline:
  pipelineSpec:
    tasks:
      - name: scaffold-unit-tests
        taskRef:
          kind: Task
          name: scaffold-unit-tests
      - name: build-skaffold-web
        taskRef:
          kind: Task
          name: build-skaffold-web
      - name: build-skaffold-app
        taskRef:
          kind: Task
          name: build-skaffold-app
      - name: deploy-web
        taskRef:
          kind: Task
          name: deploy-web
      - name: deploy-app
        taskRef:
          kind: Task
          name: deploy-app
```

Task: buildah

buildah
tekton • 4 | Repository
Buildah task builds source into a container image and then pushes it to a container registry.
92 Jesters | Install

Description: YAML

Buildah
This Task builds source into a container image using Project Atomic's `buildah` build tool. It uses Buildah's support for building from `Dockerfile`s using `buildah build` command. This command executes the directives in the `Dockerfile` to assemble a container image, then pushes that image to a container registry.
Install the Task:
`kubectl apply -f https://raw.githubusercontent.com/tektoncd/catalog/master/task/buildah/v1.2/buildah.yaml`

Parameters

- IMAGE: The name (reference) of the image to build.
- BUILDER_IMAGE: The name of the image containing the Buildah tool. See note below. (default: quay.io/buildah/stable:v1.17.0)
- DOCKERFILE: The path to the Dockerfile to execute (default: ./Dockerfile)
- CONTEXT: Path to the directory to use as context (default: -)
- TLSVERIFY: Verify the TLS on the registry endpoint (set to false if using a non-TLS registry) (default: true)
- FORMAT: Set the output format for the build command (tar or docker) (default: tar)
- BUILD_EXTRA_ARGS: Extra parameters passed for the build command when building images. (default: "")
- PUSH_EXTRA_ARGS: Extra parameters passed for the push command when pushing images. (default: "")

Workspaces

- source: A `Workspace` containing the source to build.
- selector: An optional `Workspace` containing your custom SSL certificates to connect to the registry. Buildah will look for files ending with `.cert`, `.key` and `.crt` files in this workspace. See [this example](#) for a complete example on how to use it with OpenShift internal registry.

Usage

This TaskRun runs the Task to fetch a Git repo, build and push a container image using Buildah.

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: buildah-build-my-repo
spec:
```

The screenshot shows two instances of the Tekton Hub search interface. On the left, in a terminal window of Visual Studio Code, the command `telton` is run, and the search results are displayed. On the right, in a terminal window of IntelliJ, the command `telton` is also run, and the search results are shown. Both interfaces show a list of available Tekton Hub items, including 'aws-dl', 'buildah', 'curl', 'git-dl', 'git-clone', 'golang-build', and 'jenkins'. Each item has an 'Install' button next to its name.

Results (102):

- aws-dl
- buildah
- curl
- git-dl
- git-clone
- golang-build
- jenkins

DEMO TIME

<https://github.com/wpernath/openshift-cicd-demo>

OpenShift GitOps

Declarative GitOps for
multi-cluster continuous
delivery

What is GitOps?

GitOps is when the infrastructure and / or application state is fully represented by the contents of a git repository. Any changes to the repository are reflected in the corresponding state of the associated infrastructure and applications through automation

GitOps is a natural evolution of Agile and DevOps (and Kubernetes) methodologies

Why GitOps

It takes too long to provision a new environment!

The app behaves different in prod than in test!

I have no visibility or record of config changes in deployments!

I can't easily rollback changes to a specific version

Environments are all manually configured!!!

Production deployments have a low success rate!

I can't audit config changes!!



GitOps Benefits

- All changes are auditable
- Standard roll-forward or backwards in the event of failure
- Disaster recovery is “reapply the current state of the manifests”
- Experience is “pushes and pull-requests”

OpenShift and GitOps - A great match

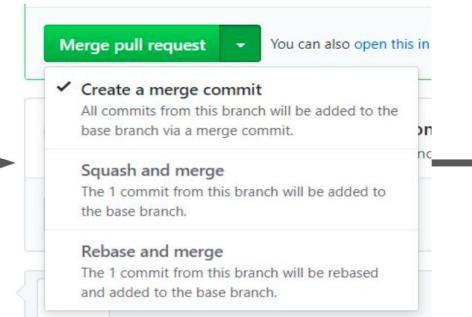
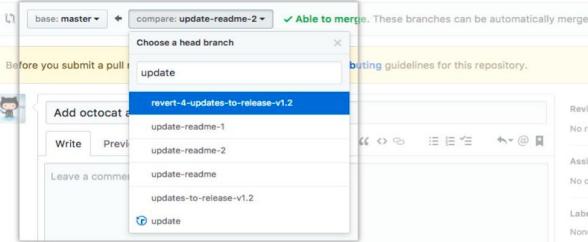
- OpenShift is a declarative environment
 - Cluster configuration is declared and Operators make it happen
 - Application deployments are declared and Kubernetes scheduler makes it happen
- GitOps in traditional environments requires automation/scripting, declarative environment minimizes or eliminates this need
- Declarations are yaml files which are easily stored and managed in git



Day 2 operations: All changes triggered from Git

Open a pull request

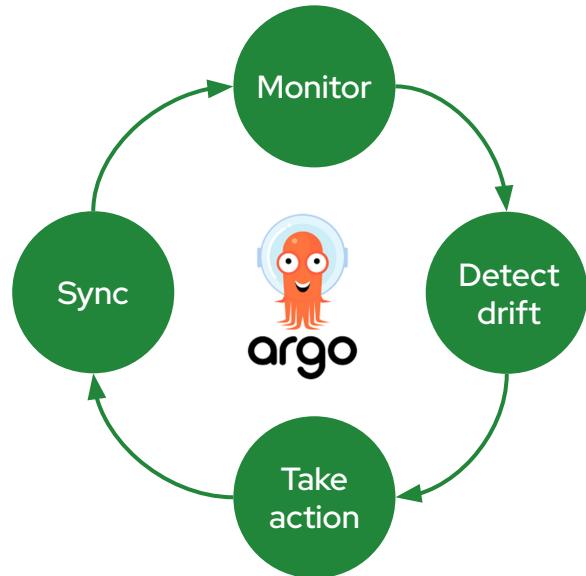
Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.



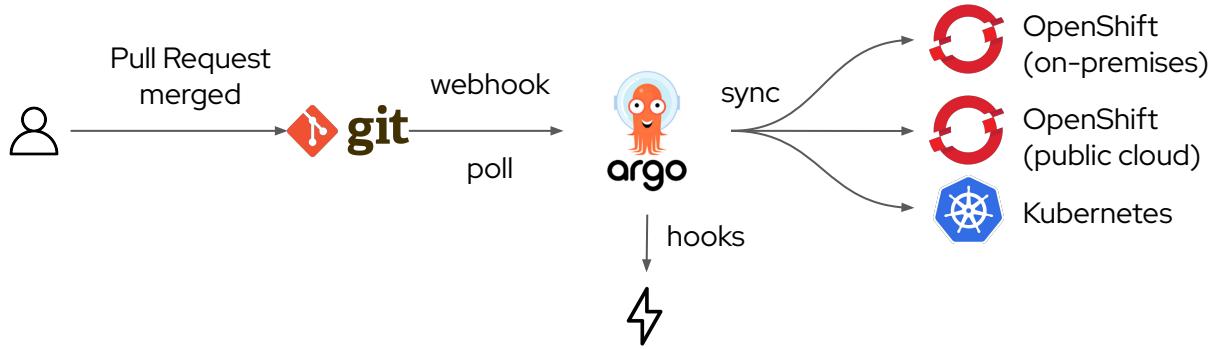
```
$ tkn pipelinerun logs update-from-master-run-g6s45
[run-kubectl] {"level": "info", "ts": 1580989837.3045664, "logger": "fallback-logger", "caller": "logging/config.go:69", "msg": "Fetch GitHub commit ID from kodata failed: \"\\$KO_DATA_PATH\" does not exist or is empty"}
[run-kubectl] serviceaccount/demo-sa unchanged
[run-kubectl] clusterrolebinding.rbac.authorization.k8s.io/tekton-triggers-openshift-binding unchanged
[run-kubectl] eventlistener.tekton.dev/demo-event-listener configured
[run-kubectl] task.tekton.dev/deploy-from-source-task configured
[run-kubectl] triggerbinding.tekton.dev/update-from-master-binding unchanged
[run-kubectl] triggertemplate.tekton.dev/update-from-master-template unchanged
```

Argo CD

- Cluster and application configuration versioned in Git
- Automatically syncs configuration from Git to clusters
- Drift detection, visualization and correction
- Granular control over sync order for complex rollouts
- Rollback and rollforward to any Git commit
- Manifest templating support (Helm, Kustomize, etc)
- Visual insight into sync status and history

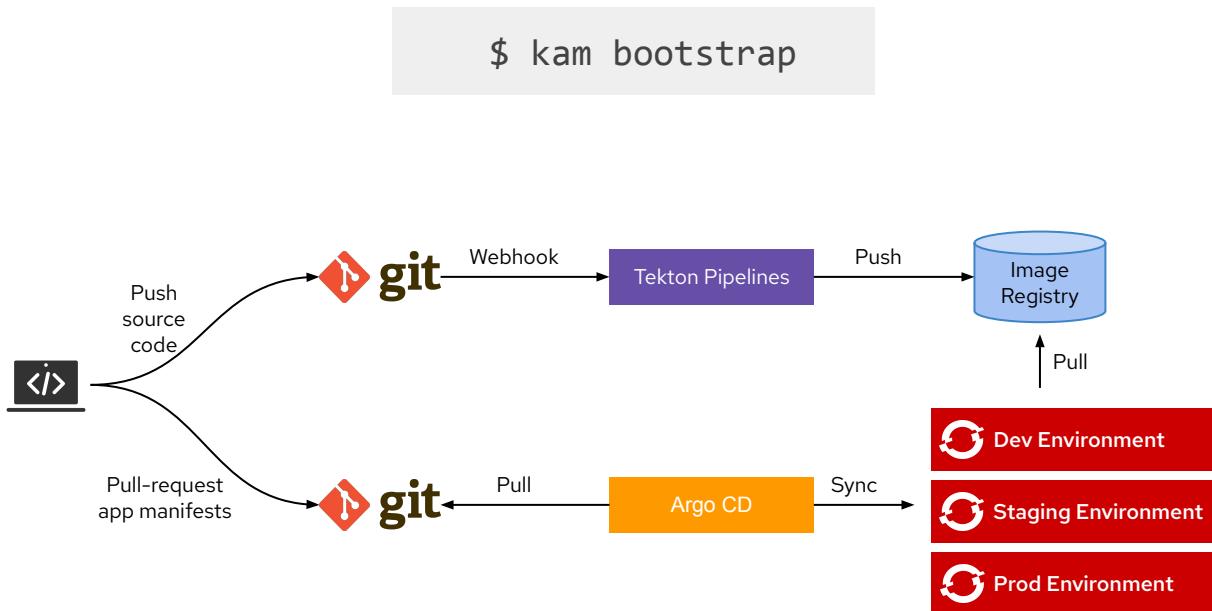


Argo CD



A screenshot of the Argo CD web interface for the 'realtime' application. The top navigation bar includes 'APPLICATION DETAILS', 'APP DIFF', 'SYNC', 'SYNC STATUS', 'HISTORY AND ROLLBACK', 'DELETE', and 'REFRESH'. The main area shows the application is 'Synced' to the latest commit. Below this, a 'synchronization' status is shown, indicating it succeeded 38 minutes ago. The interface displays a dependency graph between 'realtimeapp' and 'redisapp' components across two environments. On the left, there's a sidebar with icons for application, deployment, service, endpoints, and replicant. On the right, there are links for 'Customize and control Google Chrome' and 'Logout'.

GitOps Application Manager CLI



OpenShift GitOps



Multi-cluster config management

Declaratively manage cluster and application configurations across multi-cluster OpenShift and Kubernetes infrastructure with Argo CD



Automated Argo CD install and upgrade

Automated install, configurations and upgrade of Argo CD through OperatorHub



Opinionated GitOps bootstrapping

Bootstrap end-to-end GitOps workflows for application delivery using Argo CD and Tekton with GitOps Application Manager CLI



Deployments and environments insights

Visibility into application deployments across environments and the history of deployments in the OpenShift Console

ArgoCD - Challenges

- Repo structure for manifests
 - One Repo or
 - Separate Repos for environments
- Order dependent deployments
- Non-declarative deployments
- Integration with CI/CD tools (Jenkins, Pipelines...)
 - Who does manage deployments?



Multiple repositories

/taxi.git

deploy

pipelines

pkg/cmd/booktaxi

web

Dockerfile

/taxi-config-stage.git

deploy

pipelines

README.md

/taxi-config-prod.git

deploy

pipelines

README.md

/taxi-config-test.git

deploy

pipelines

README.md

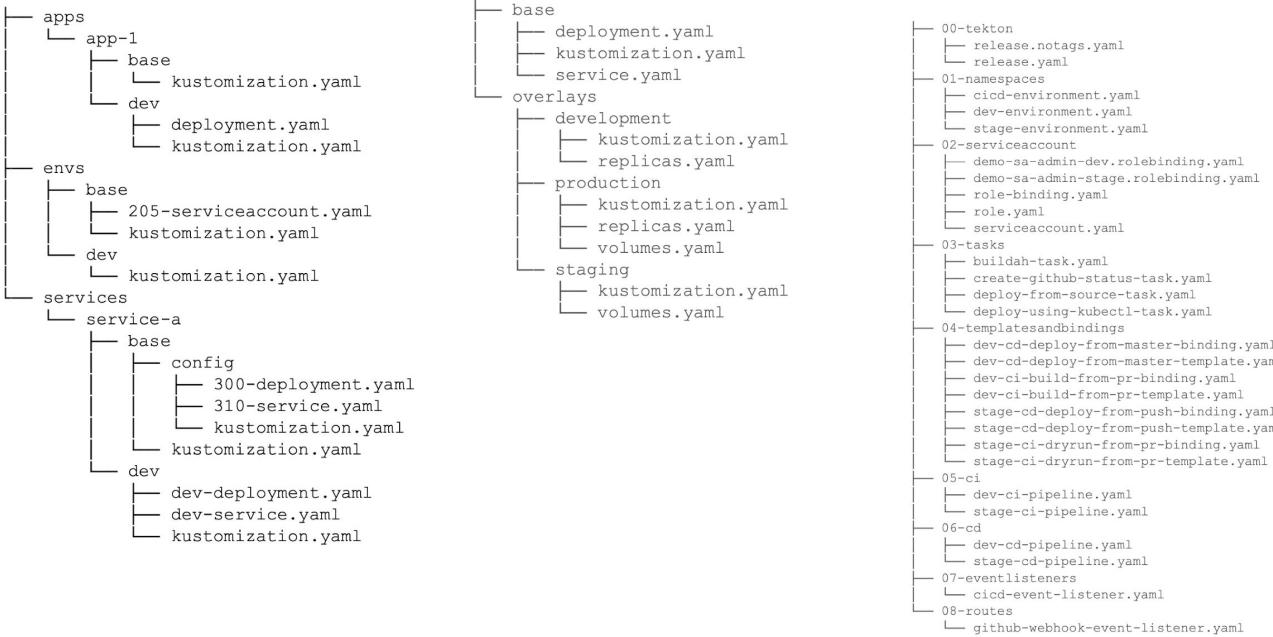
/taxi-config-dev.git

deploy

pipelines

README.md

Single Repository



ArgoCD - Order Dependent Deployments

- Sometimes you have cases where you need to deploy things in a specific order
 - Subscribe Operator before deploying instance
 - Create namespace before deploying app into it
 - Deploy required infrastructure before application
- Tools like kustomize and Helm might help handling this in some cases
- ArgoCD provides Sync Phases and Waves to address other use cases
 - 3 sync phases - Pre-sync, sync, post-sync
 - Each phase can have multiple waves, next wave does not proceed until previous phase is healthy

ArgoCD - Non-declarative requirements

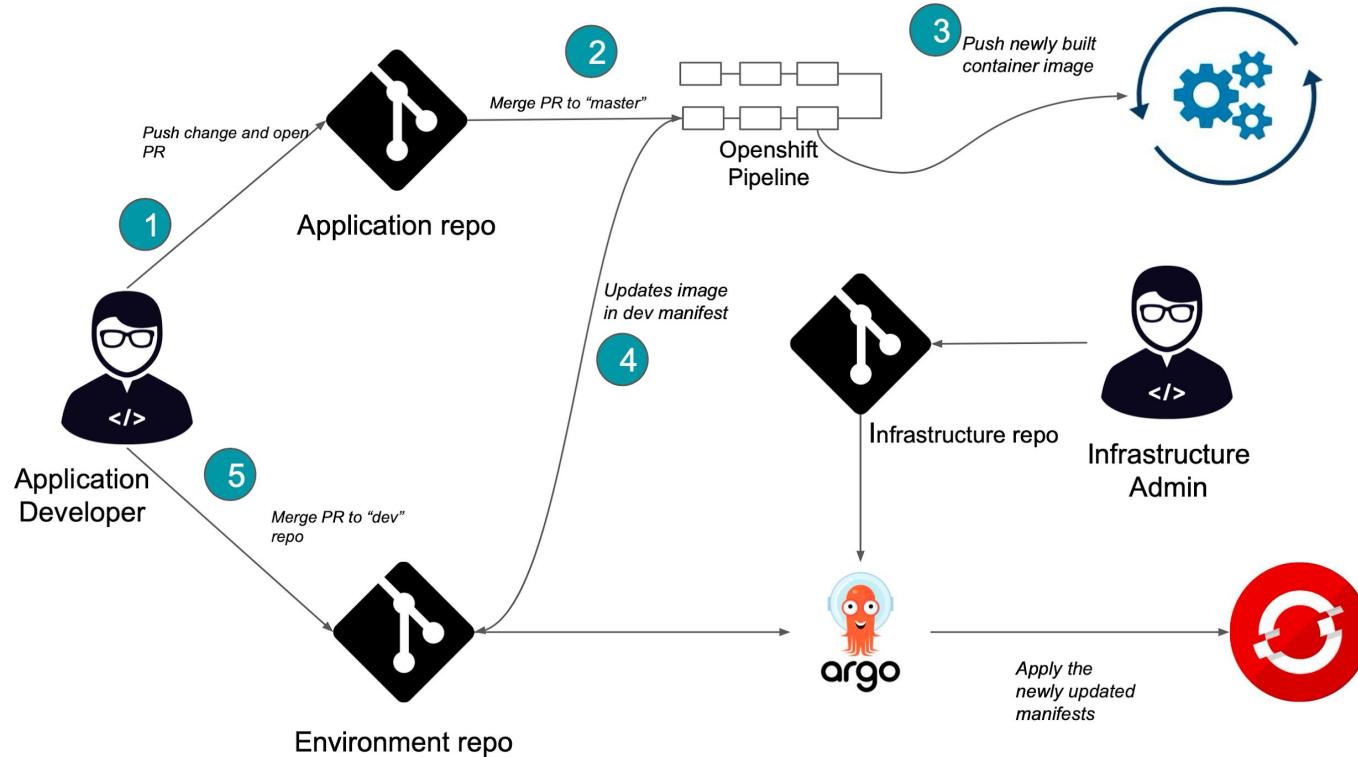
- There can be instances where you need to deploy something which cannot fully be done in a declarative way and it must be scripted
- Try minimizing this and leverage kubernetes primitives where possible
 - Init Containers
 - Jobs
 - Operators
- ArgoCD Resource Hooks
 - Hooks are ways to run scripts before, during and after a sync operation
 - Hooks can be run: PreSync, Sync, PostSync and SyncFail

ArgoCD - Integrating with CI/CD Tools

ArgoCD and NOT ArgoCI/CD

CI tools like Jenkins, Pipelines still required!

	ArgoCD Managed Deployment	Pipeline Managed Deployment
Pro	Consistent	Post-Test update of image reference
Con	Image reference updated in git before integration tests, manage rollback?	Inconsistent
Con	Pipeline tools must be able to wait for sync	



DEMO

DEMO TIME

<https://github.com/wpernath/openshift-cicd-demo>

Summary

Summary

- You don't have to change anything, if you don't want
- You can use tools like Jenkins to do your CI
- OpenShift Builds is to integrate building your image in your existing pipeline
- OpenShift Pipelines (tekton) is a nice kubernetes-native way of pipelining
- GitOps and ArgoCD helps you to do a declarative approach
 - You as developer are used to use Git
 - You as admin are used to use Git
 - → Adoption of GitOps could be high

Optional section marker or title

82



Thank you

 [linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)

 [facebook.com/redhatinc](https://www.facebook.com/redhatinc)

 [youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)

 twitter.com/RedHat