

# OpenShift 4

## For Developers

Wanja Pernath

wpernath@redhat.com

EMEA Partner Enablement Manager, OpenShift & MW

# 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/>



# 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:

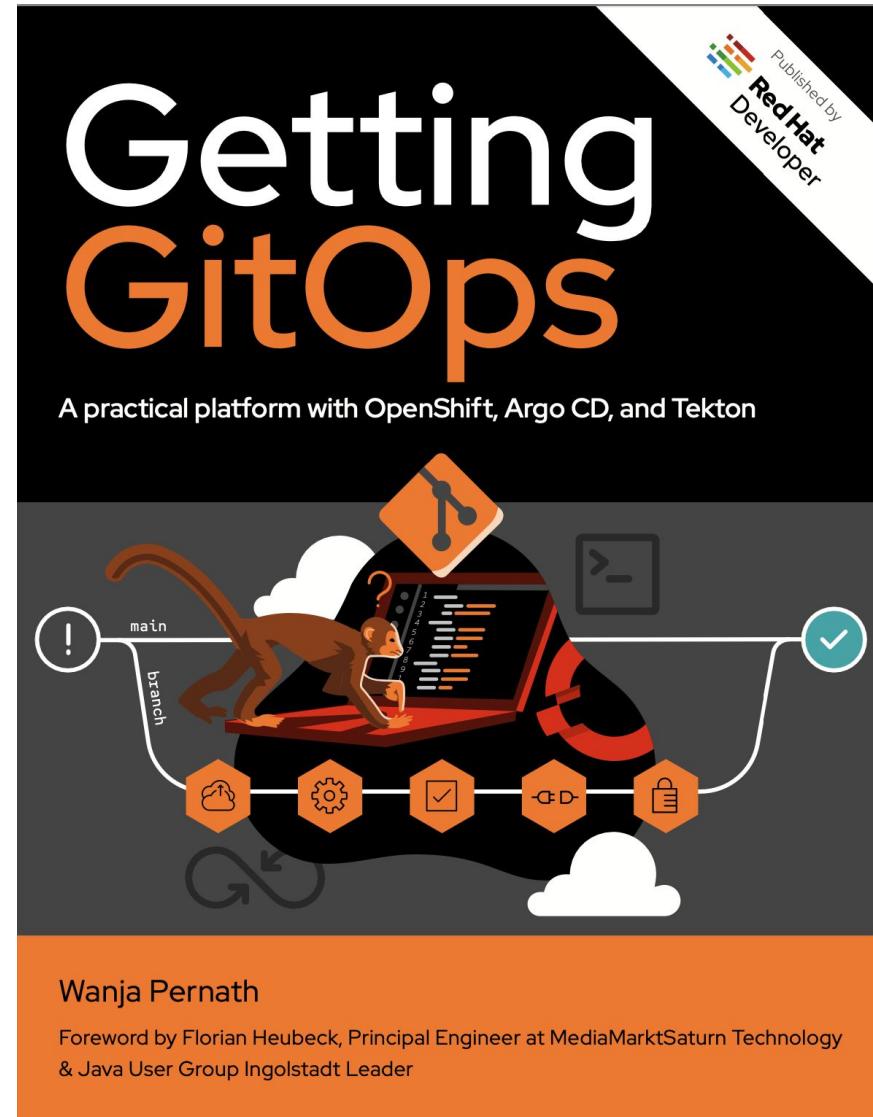
Quarkus, Helm Charts, Kustomize, Tekton Pipelines, Kubernetes Operators, OpenShift Templates, ArgoCD, CI/CD, GitOps....

### 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](https://www.youtube.com/watch?v=znMfVqAIRzY&ab_channel=OpenShift)





# OpenShift and Kubernetes core concepts

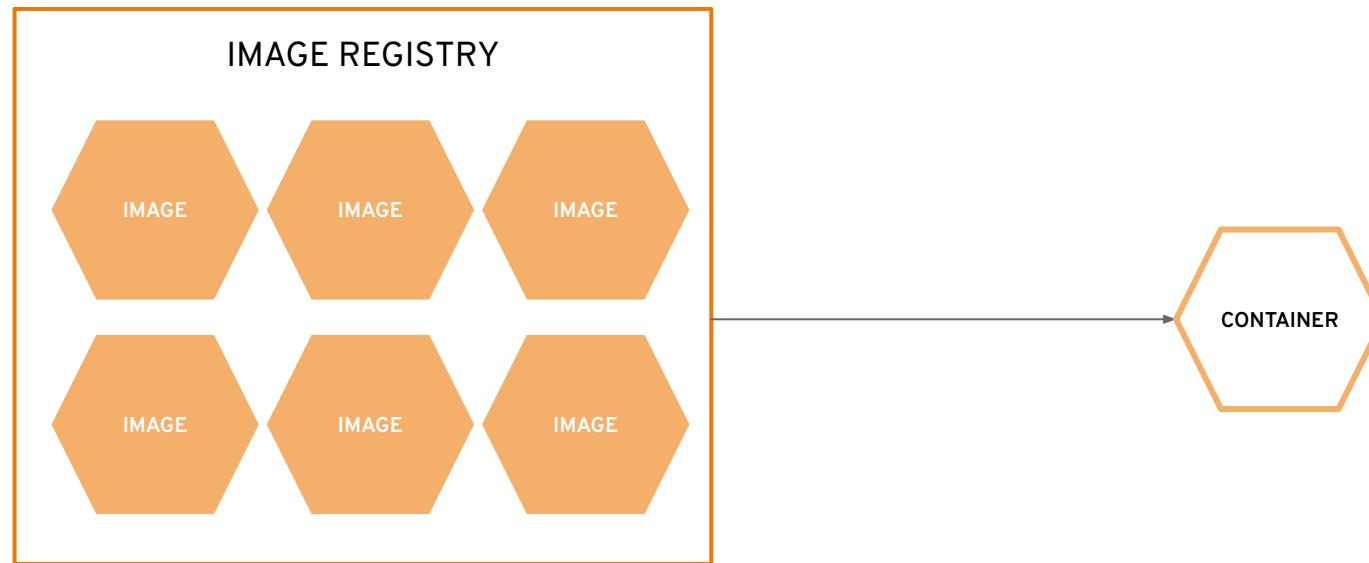
# a container is the smallest compute unit



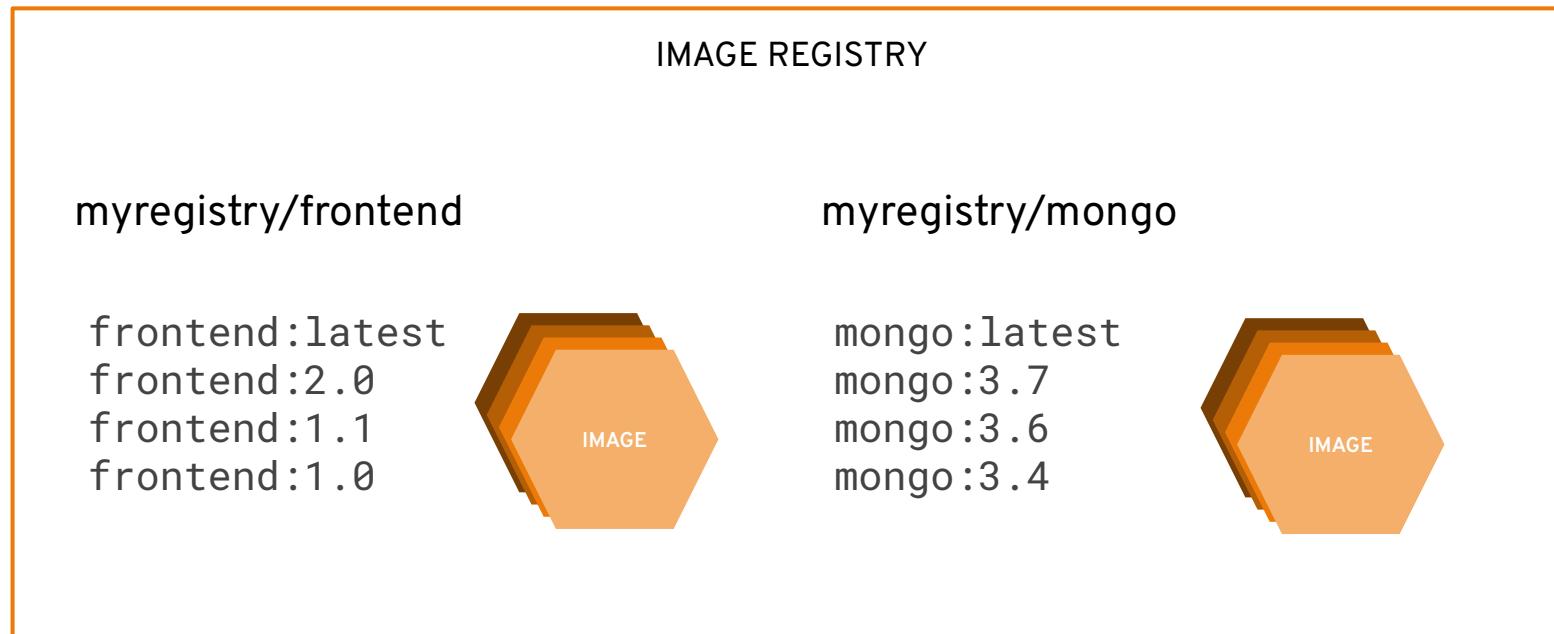
# containers are created from container images



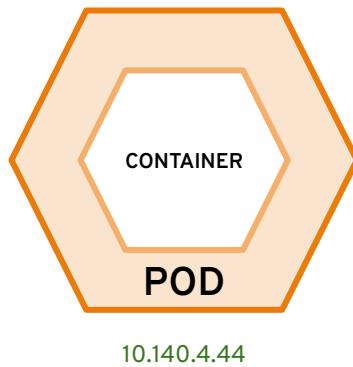
# container images are stored in an image registry



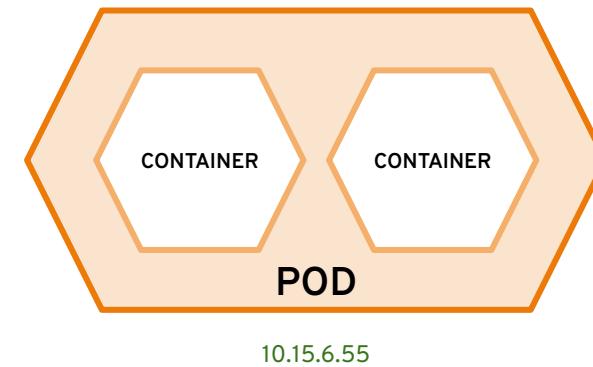
# an image repository contains all versions of an image in the image registry



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

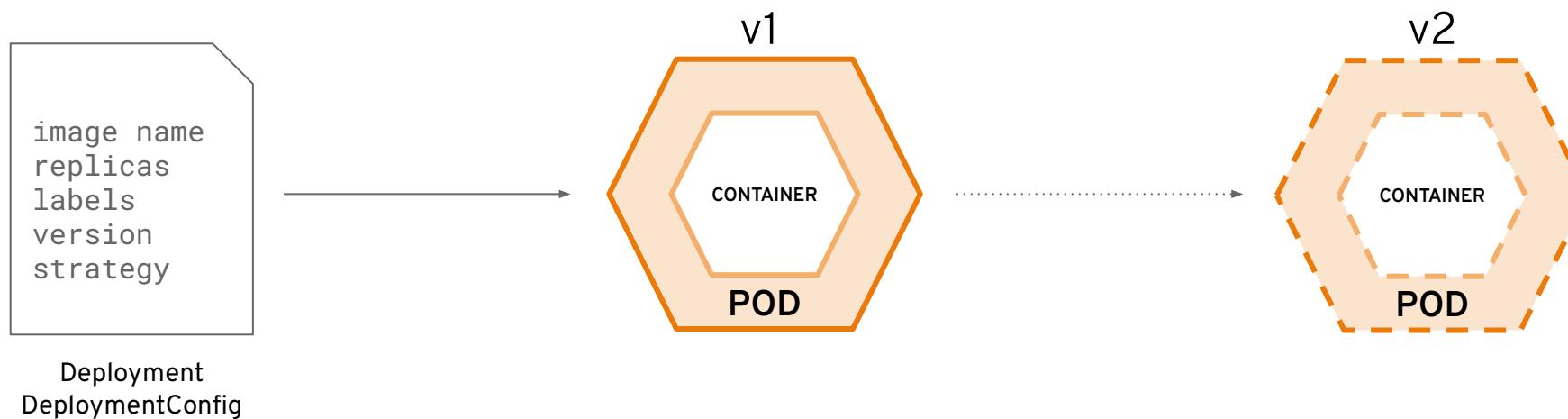


10.140.4.44

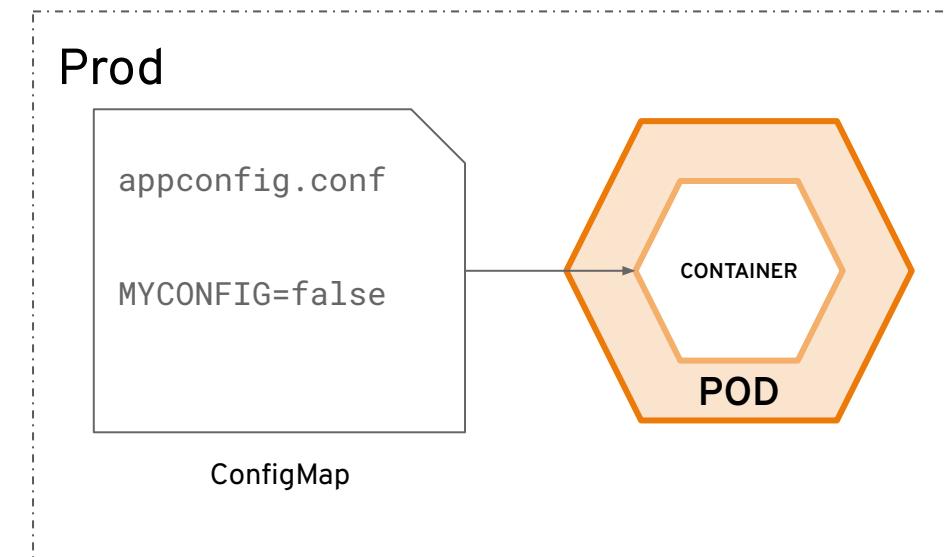
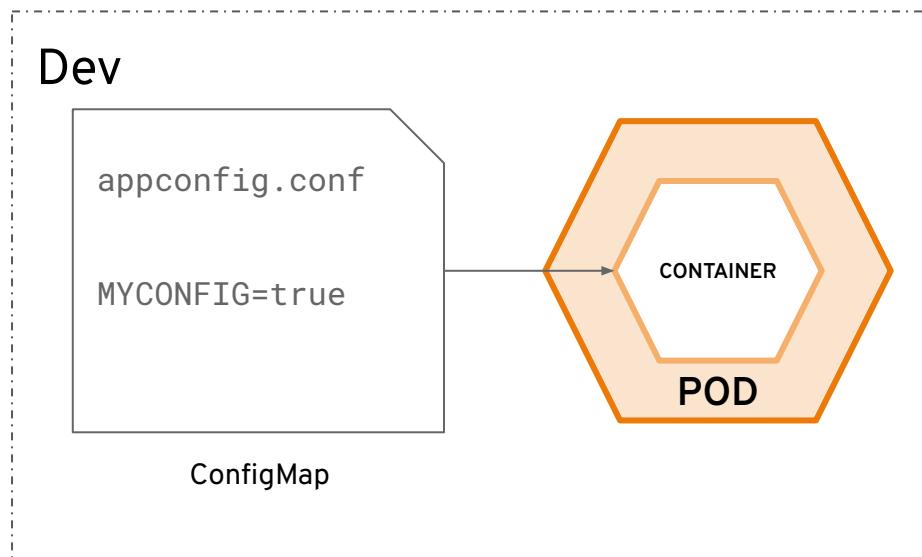


10.15.6.55

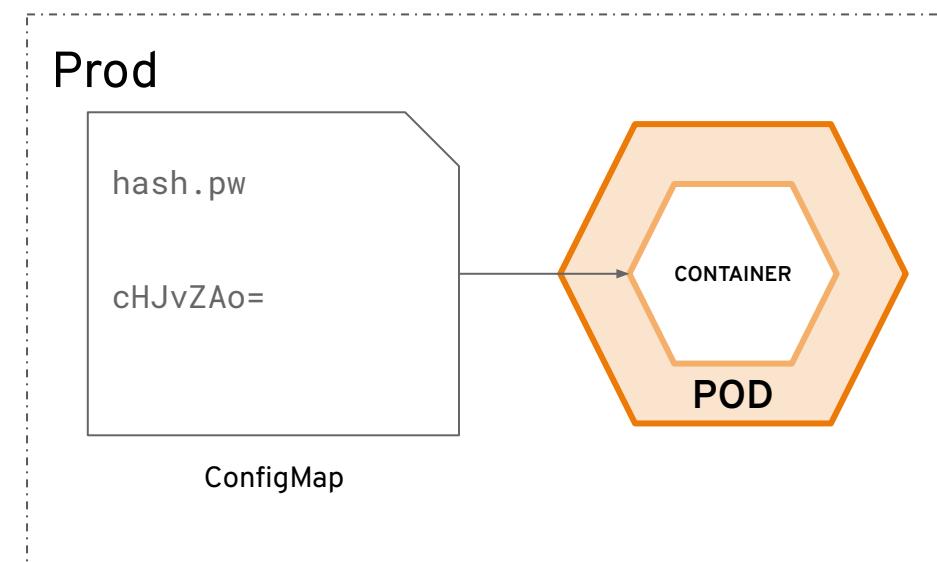
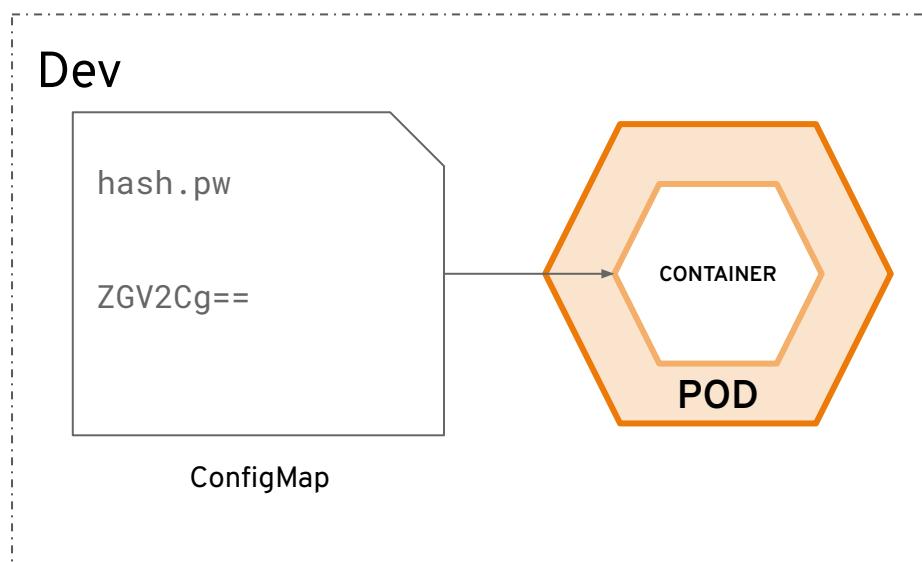
# Deployments and DeploymentConfigurations define how to roll out new versions of Pods



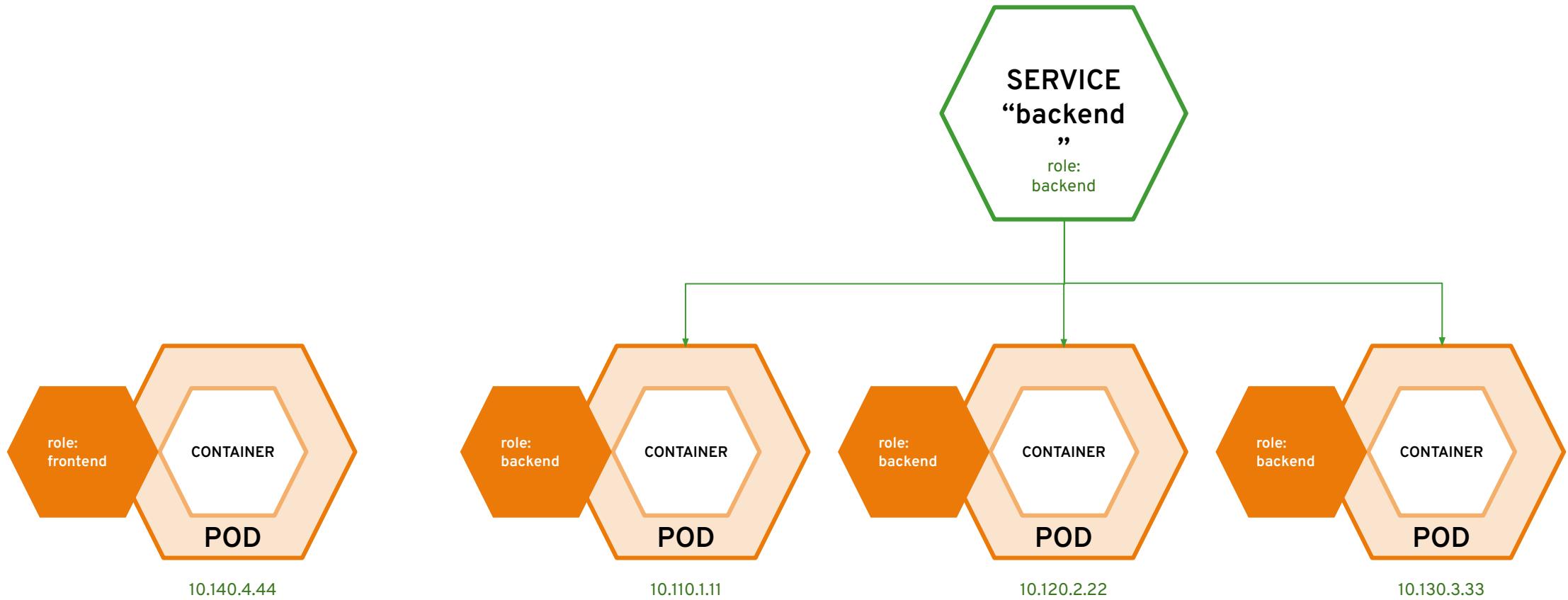
configmaps allow you to decouple configuration artifacts from image content



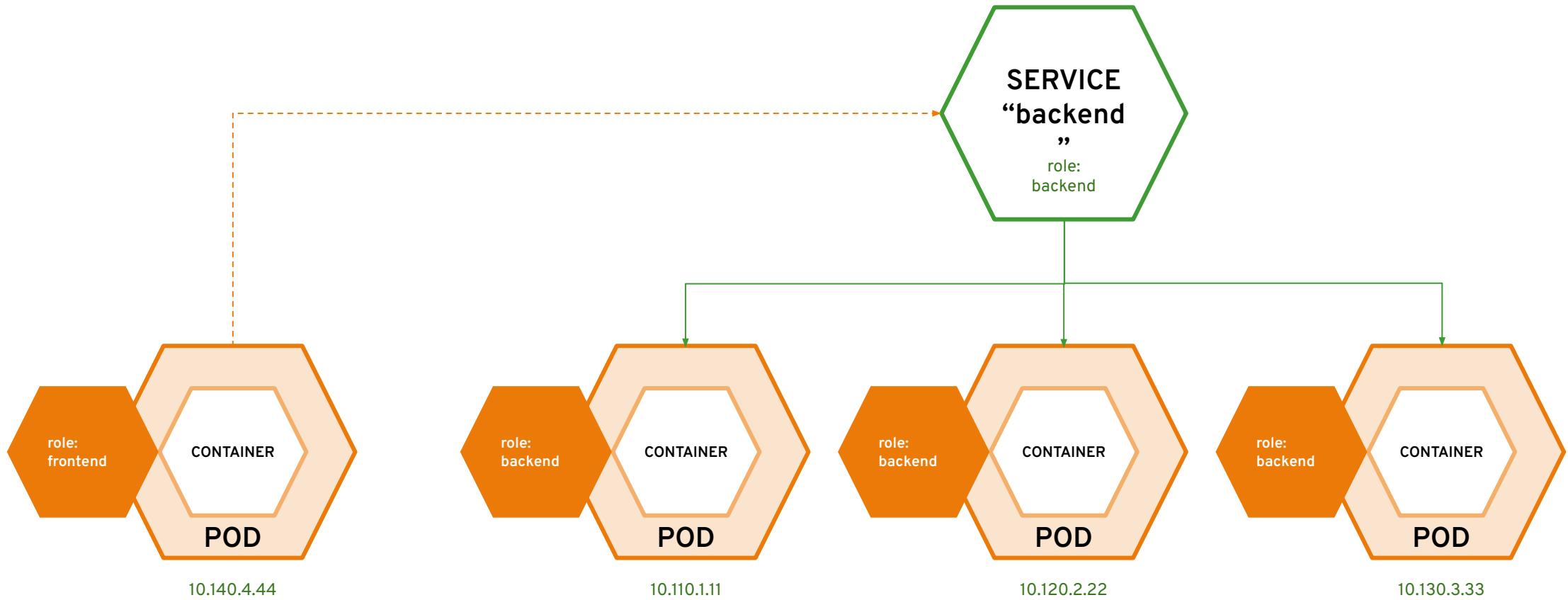
secrets provide a mechanism to hold sensitive information such as passwords



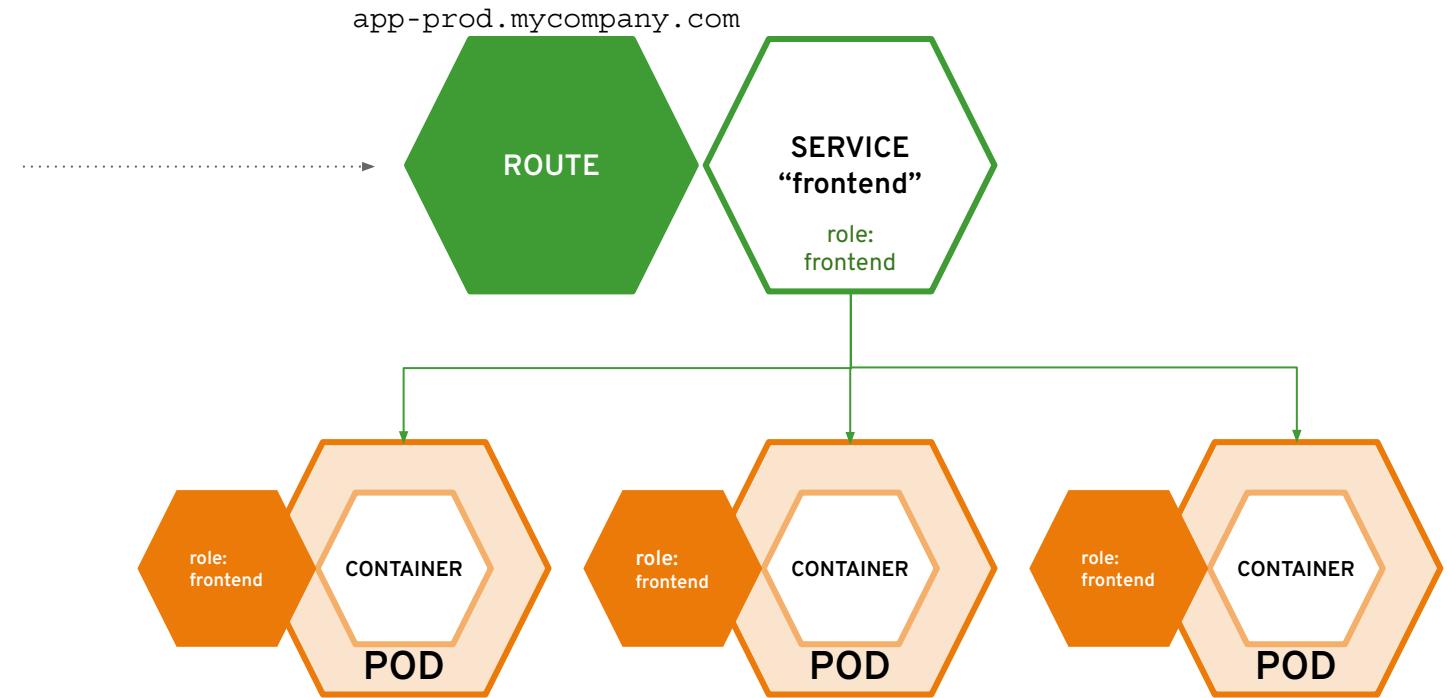
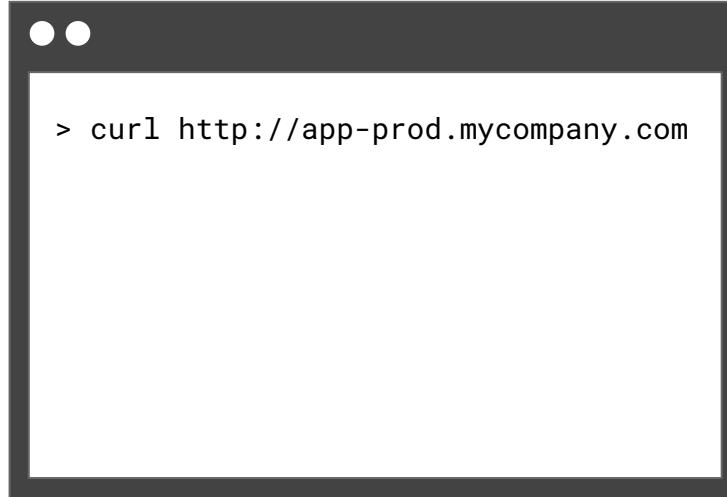
# services provide internal load-balancing and service discovery across pods



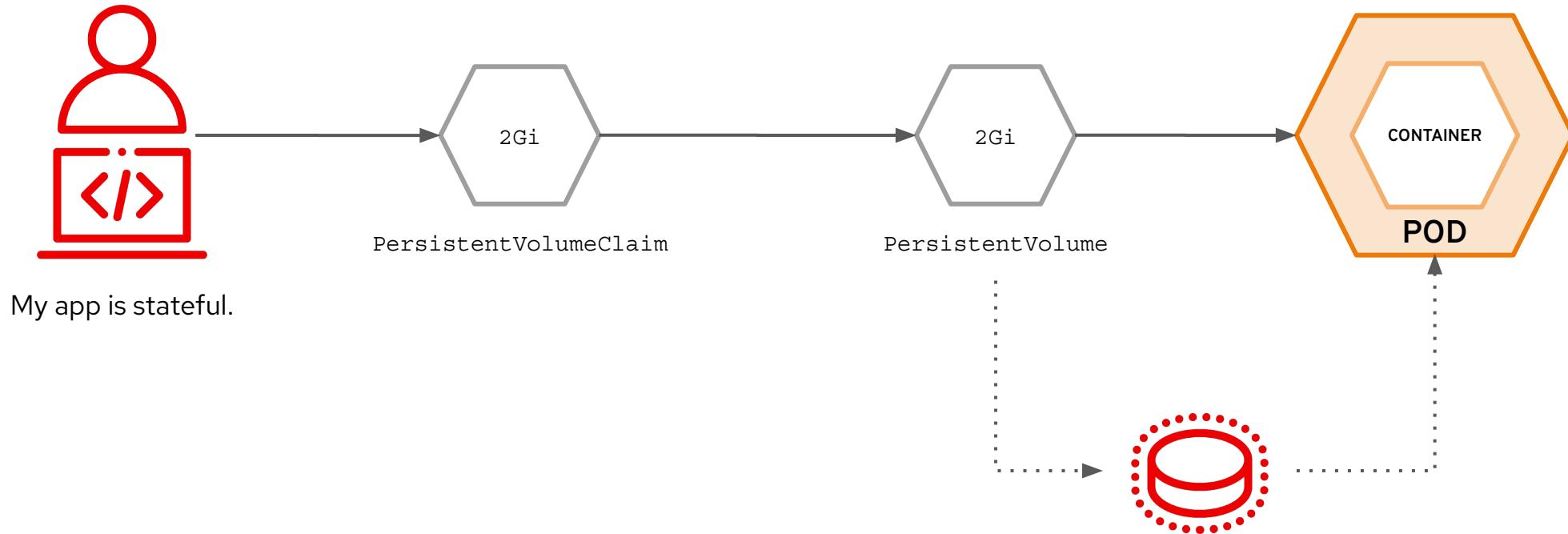
# apps can talk to each other via services



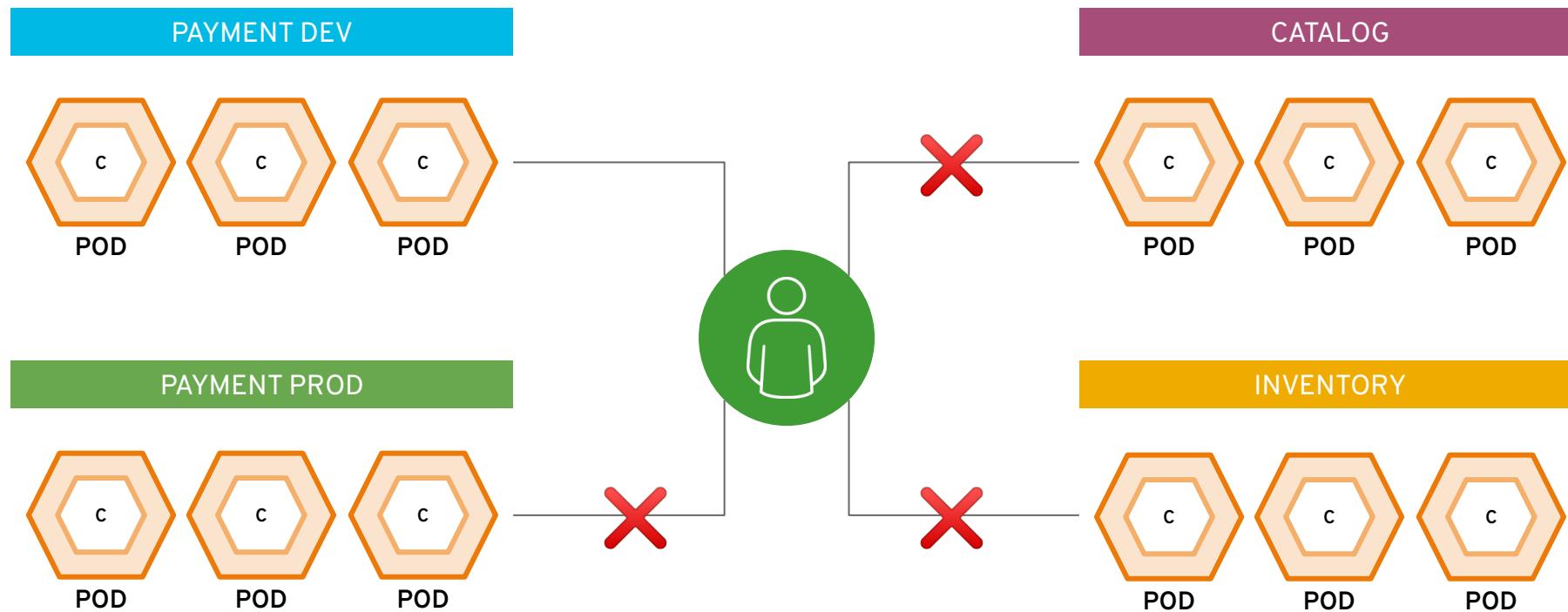
routes make services accessible to clients outside the environment via real-world urls



# Persistent Volume and Claims



projects isolate apps across environments,  
teams, groups and departments



# From Code to Image

Please be back at 10:20 CEST

Staging

Pipelining

GitOps

# Red Hat OpenShift

## Adds consistency and portability to your cloud journey

### DEVELOPMENT

### PRODUCTION



Use OpenShift across any environment



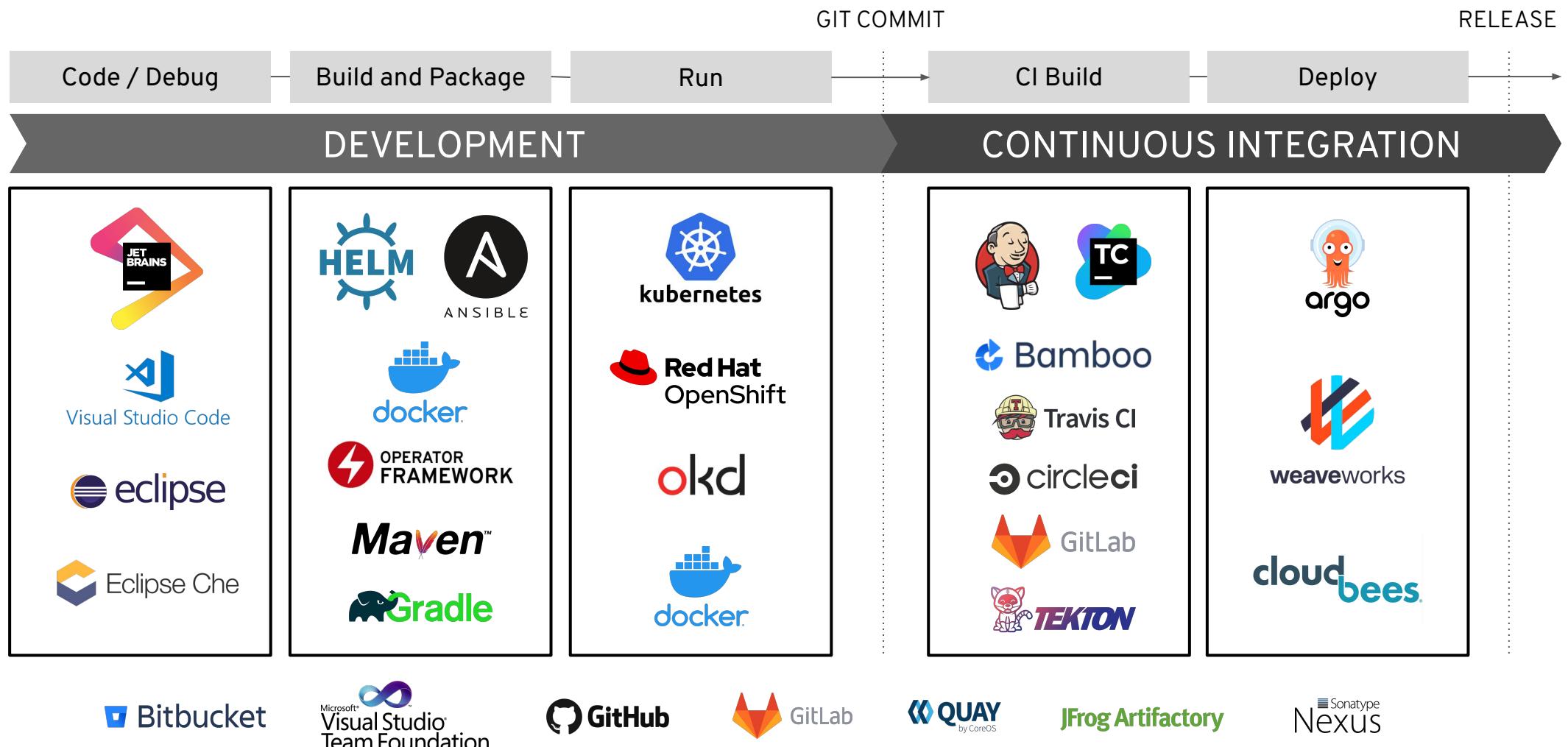
\* Develop on OpenShift for existing apps and new cloud-native apps

\* Deploy on OpenShift in AWS, Azure, current environments or anywhere else

One consistent development environment enables developers to move from project to project quickly. They don't need to learn new cloud-specific UIs and tools.

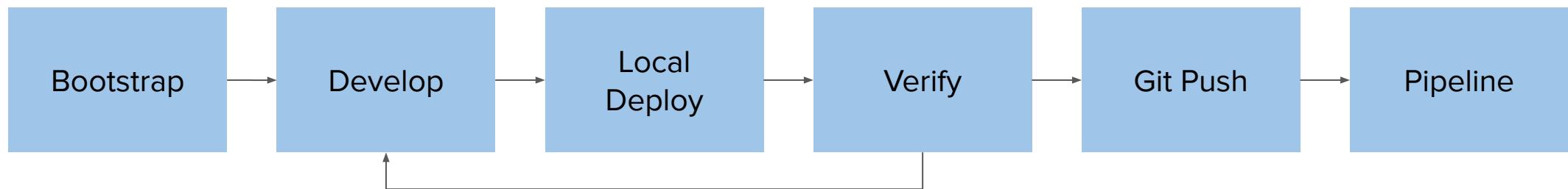
If teams need to leverage other cloud providers at any point it's easy to move specific apps because OpenShift provides an abstraction layer.

# OpenShift integrates into your organization's preferred toolchain

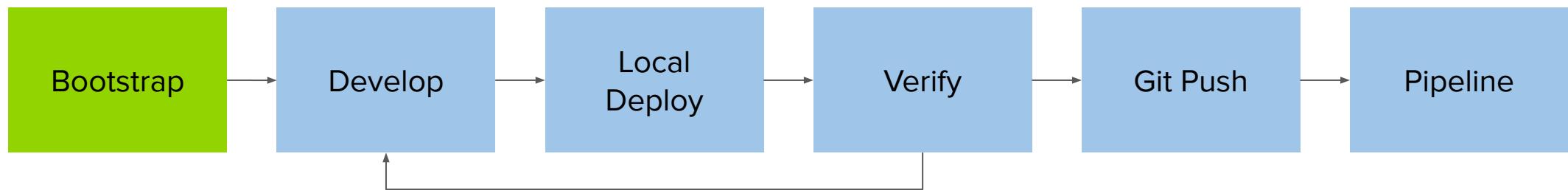


# DEVELOPER WORKFLOW

# LOCAL DEVELOPMENT WORKFLOW



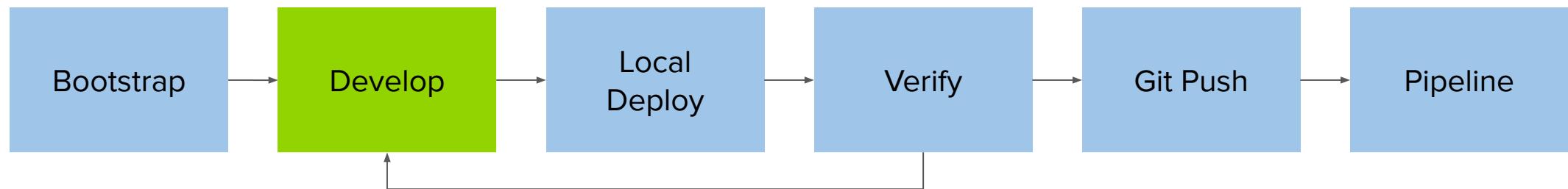
# LOCAL DEVELOPMENT WORKFLOW



## 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

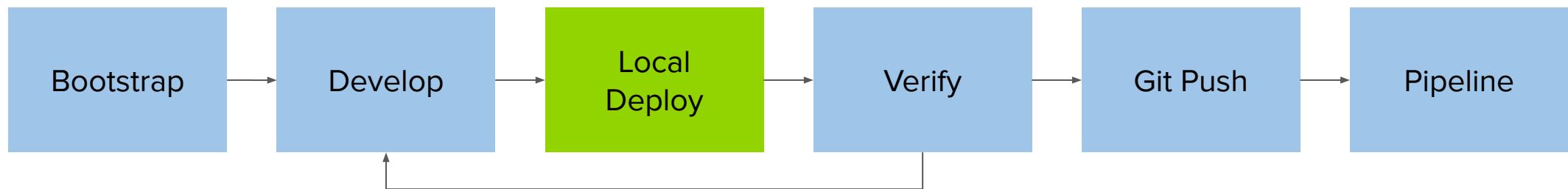
# LOCAL DEVELOPMENT WORKFLOW



## 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

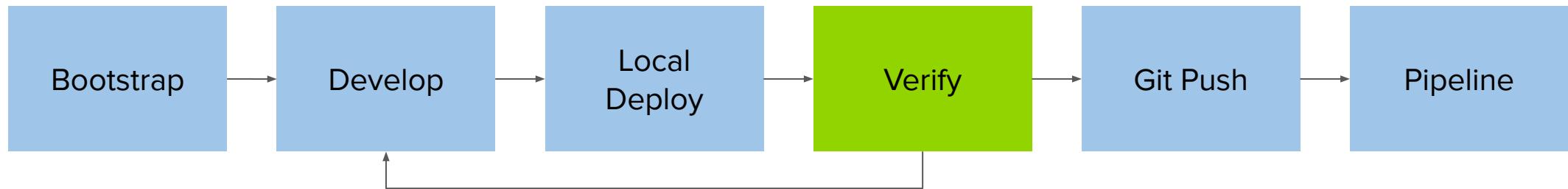
# LOCAL DEVELOPMENT WORKFLOW



## LOCAL DEPLOY

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

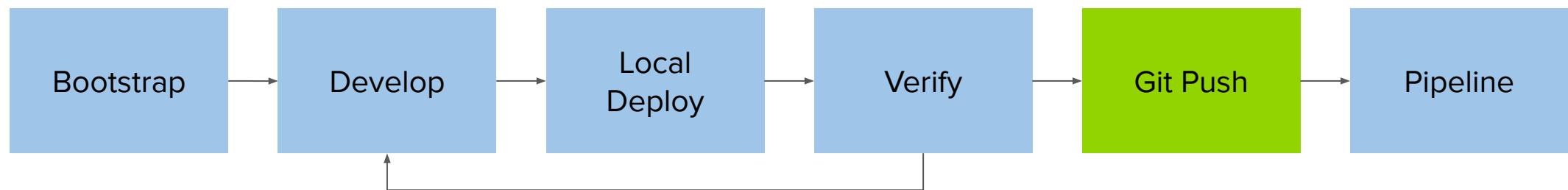
# LOCAL DEVELOPMENT WORKFLOW



## 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

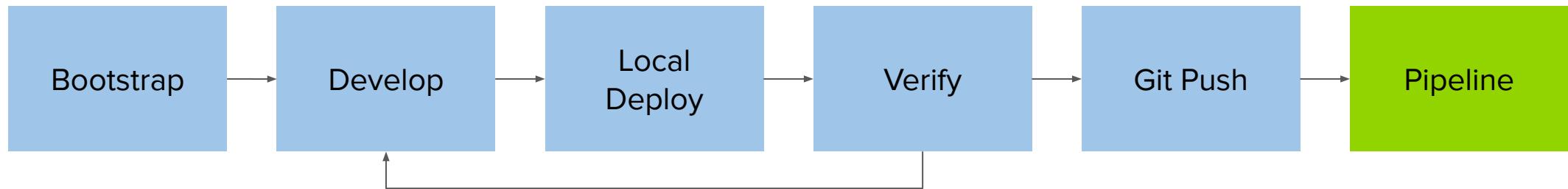
# LOCAL DEVELOPMENT WORKFLOW



## 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

# Developer Instances



## Free Developer Sandbox

<https://developers.redhat.com/developer-sandbox>

## CodeReady Containers

<https://github.com/code-ready/crc>

## Single Node OpenShift

<https://console.redhat.com/openshift/assisted-installer/clusters/~new>

<https://www.youtube.com/watch?v=QFfOyVAHQKc>



# Setting up and using CodeReady Containers (crc)

## CodeReady Containers

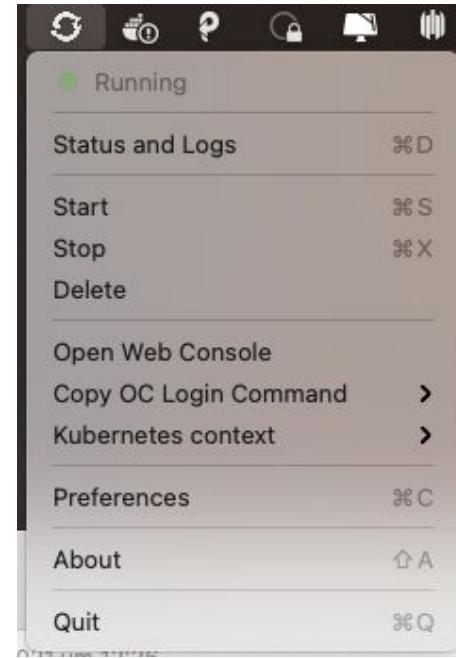
<https://github.com/code-ready/crc>

**A single node OpenShift 4.10.x installation for local testing only**

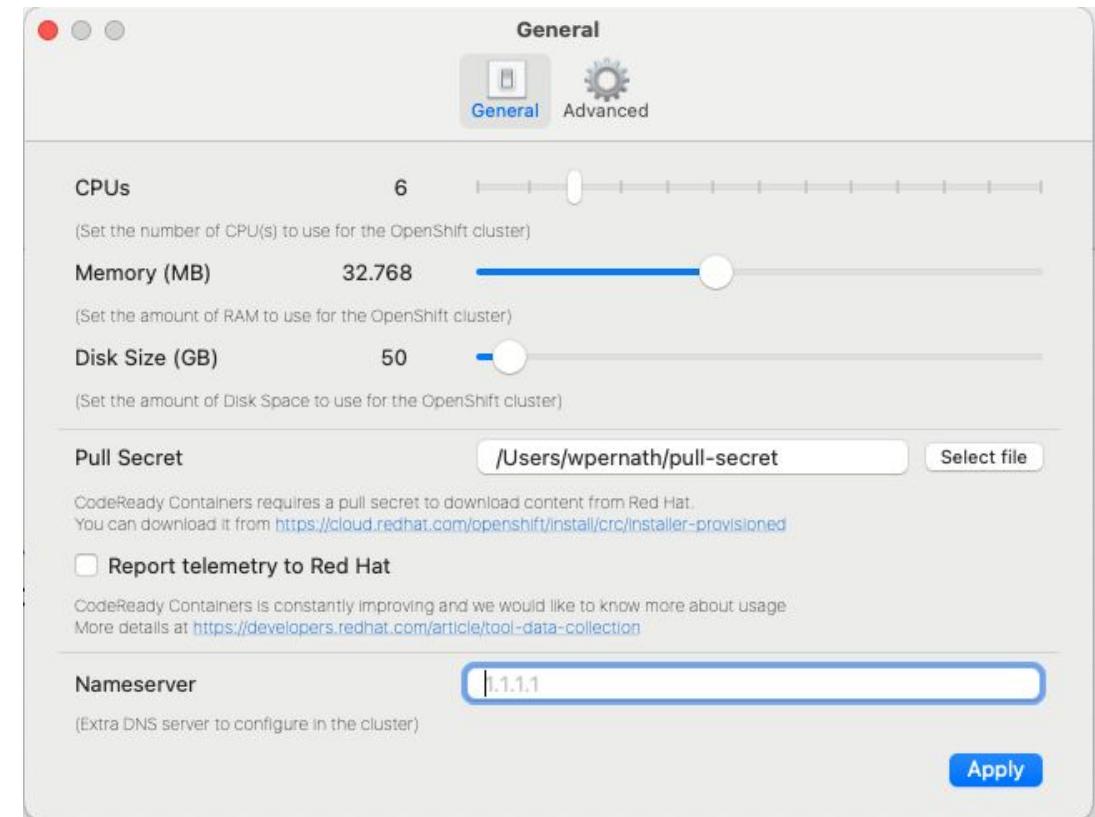
### Requirements:

- 4 CPU cores
- 9GB memory
- 35GB disk space

- Download archive for your OS (mac, Linux, Windows 10)
- On mac and windows, simply install the archive
- On Linux, simply unzip the archive
- After installation, you're able to see an icon in the task or menu bar



- Set settings according to your workstation (memory, CPUs, disk size)
- Follow the instructions to download the pull-secret
- Hit Apply
- Go into a console and use “Copy OC login command” for Kubeadmin user



```
[wpernath@biggestmac ~/Devel/openshift-config]$ crc config list
Modifies crc configuration properties.
Properties:

* autostart-tray          Automatically start the tray (true/false, default: true)
* bundle                  Bundle path (string, default '/Applications/CodeReady Containers.app/Contents/Resources/crc_hyperkit_4.8.4.crcbundle')
* consent-telemetry       Consent to collection of anonymous usage data (yes/no)
* cpus                    Number of CPU cores (must be greater than or equal to '4')
* disable-update-check    Disable update check (true/false, default: false)
* disk-size               Total size in GiB of the disk (must be greater than or equal to '31')
* enable-cluster-monitoring Enable cluster monitoring Operator (true/false, default: false)
* enable-experimental-features Enable experimental features (true/false, default: false)
* host-network-access     Allow TCP/IP connections from the CodeReady Containers VM to services running on the host (true/false, default: false)
* http-proxy              HTTP proxy URL (string, like 'http://my-proxy.com:8443')
* https-proxy             HTTPS proxy URL (string, like 'https://my-proxy.com:8443')
* kubeadmin-password      User defined kubeadmin password
* memory                 Memory size in MiB (must be greater than or equal to '9216')
* nameserver              IPv4 address of nameserver (string, like '1.1.1.1 or 8.8.8.8')
* no-proxy                Hosts, ipv4 addresses or CIDR which do not use a proxy (string, comma-separated list such as '127.0.0.1,192.168.100.1/24')
* proxy-ca-file           Path to an HTTPS proxy certificate authority (CA)
* pull-secret-file        Path of image pull secret (download from https://cloud.redhat.com/openshift/create/local)
* skip-check-admin-helper-cached Skip preflight check (true/false, default: false)
* skip-check-bundle-extracted Skip preflight check (true/false, default: false)
* skip-check-hyperkit-driver Skip preflight check (true/false, default: false)
* skip-check-hyperkit-installed Skip preflight check (true/false, default: false)
* skip-check-m1-cpu        Skip preflight check (true/false, default: false)
* skip-check-obsolete-admin-helper Skip preflight check (true/false, default: false)
* skip-check-qcow-tool-installed Skip preflight check (true/false, default: false)
* skip-check-ram            Skip preflight check (true/false, default: false)
* skip-check-resolver-file-permissions Skip preflight check (true/false, default: false)
* skip-check-root-user      Skip preflight check (true/false, default: false)
* skip-check-supported-cpu-arch Skip preflight check (true/false, default: false)

Usage:
  crc config SUBCOMMAND [flags]
  crc config [command]

Available Commands:
  get      Get a crc configuration property
  set      Set a crc configuration property
  unset   Unset a crc configuration property
  view    Display all assigned crc configuration properties
```

- Use Console and crc command to view & set advanced properties
- For example to set a new kubeadmin password, use the following
  - `crc config set kubeadmin-password admin123`

```
[wpernath@biggestmac ~/Devel/openshift-config]$ crc config view
- autostart-tray : false
- consent-telemetry : no
- cpus : 6
- disk-size : 50
- enable-cluster-monitoring : true
- kubeadmin-password : admin123
- memory : 32768
- pull-secret-file : /Users/wpernath/pull-secret
```

```
[wpernath@biggestmac ~/Devel/openshift-config]$ crc config set kubeadmin-password admin123
Successfully configured kubeadmin-password to admin123
```

- Depending on your use case, have a look at the Operators section
  - I am typically installing OpenShift Pipelines and OpenShift GitOps Operators to have Tekton and ArgoCD installed
  - You could also install OpenShift Serverless (knative)
  - Or install a better version of PostgreSQL database (Crunchy)
  - You might also need to install Jenkins Operator (CI)
  - Or a Nexus Service (maven proxy repository)
- See next section about Kubernetes Operators.
- Or look and see the [Artifact Hub](#) for available Helm Charts



# What are Kubernetes Operators?

# What

- Operators were originally invented in 2018 by CoreOS as a way to extend kubernetes by adding new custom resources and controllers or to help human operators to operate and manage stateful applications on kubernetes
- Operators are part of the kubernetes project
- Operators are like Helm Charts a way to distribute your app
- Operators contain all the domain logic required to manage the app
  - It understands how to scale up / down a stateful app
  - It understands how to do backups
- Operators are packaged as a mix of yaml definitions and a standard language like Go, Java etc.
- OperatorHub.io contains a nice set of available operators
- OperatorSDK helps you to create an own operator

# Wait... stateless / stateful?

- Stateless
  - Kubernetes can manage stateless apps easily
  - Scaling is just a matter of adding a new pod
- Stateful
  - Imagine a mysql database
  - Scaling that up, means kubernetes is creating copies of the data
  - In fact, you then would have 3 different DBs
  - You need to find a way to properly scale mysql
  - Every app handles that differently (mysql vs. postgres vs. redis)
  - That domain logic needs to be put into an Operator to automate those tasks

# Can I use it?

- Short answer: Of course!
- Longer answer:
  - Whenever you need to find a way to tell Kubernetes how to manage your stateful complex app, you have to use Operators.
  - If your app is stateless or does not need special treatment, don't use Operators, think about Helm Charts then

# Drawbacks?

- Quite complex
- You need to understand kubernetes properly

# Resources

- [OperatorHub.io | The registry for Kubernetes Operators](#)
- [Operators on Red Hat OpenShift](#)
- [Operator SDK](#)
- [An intro to Kubernetes operators](#)
- [Kubernetes Operator simply explained in 10 mins](#)



# What are Helm Charts?

# What

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

# Can I use it?

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

# When to use Helm Charts

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

# Drawbacks?

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

# Helm 2 vs 3

- Helm 2 required a server component called Tiller
  - Another app on top of kubernetes which had to be managed and maintained
  - Tiller had its own RBAC and its own audit trail
  - Tiller was storing sensitive data in ConfigMaps
  - → Loss of visibility
- Helm 3 does not need a server side component
  - It uses native kubernetes approach and only a client side tool
- **⇒ This is the reason why OpenShift did not natively support Helm prior v3**

# Resources

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

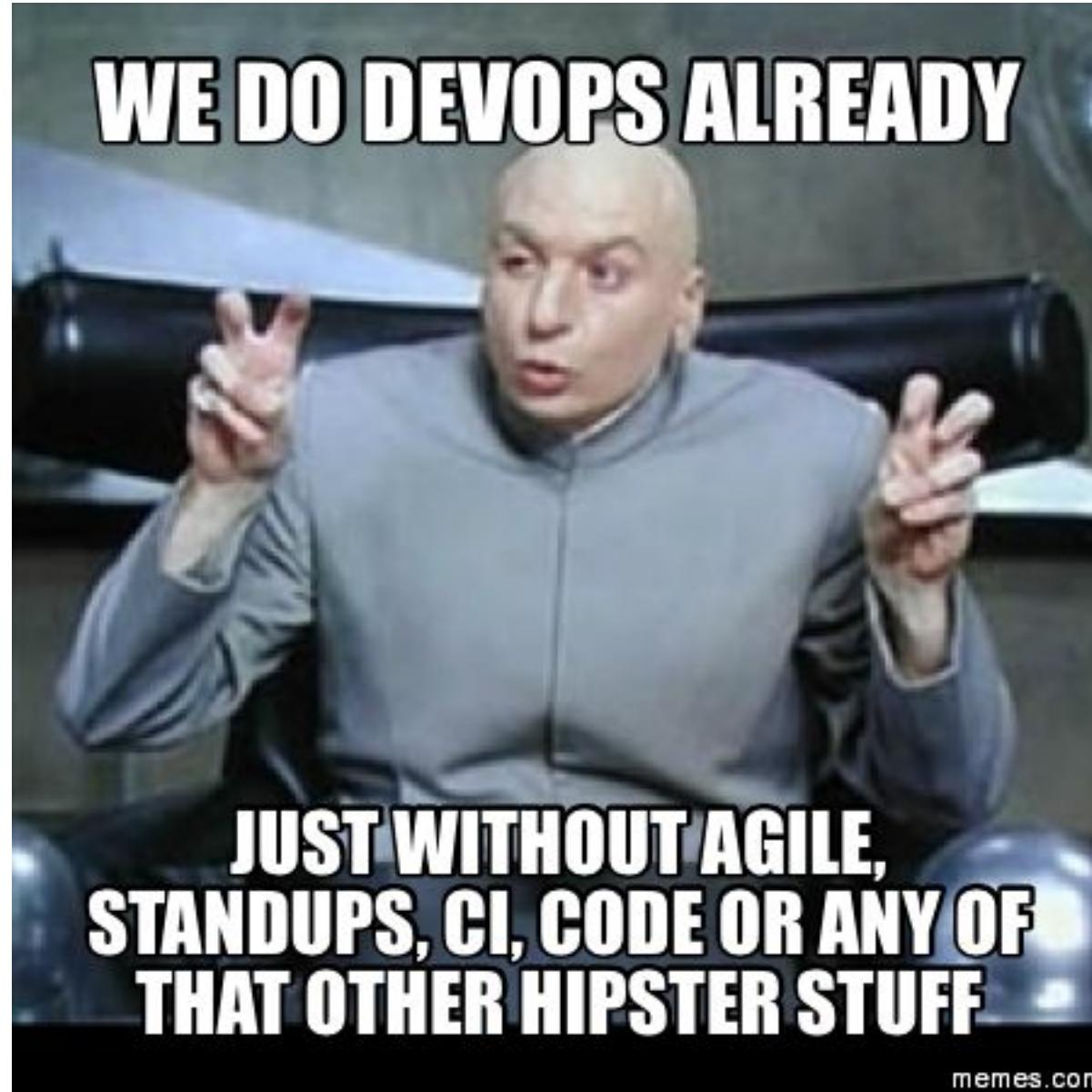
# CI/CD with OpenShift

# 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**

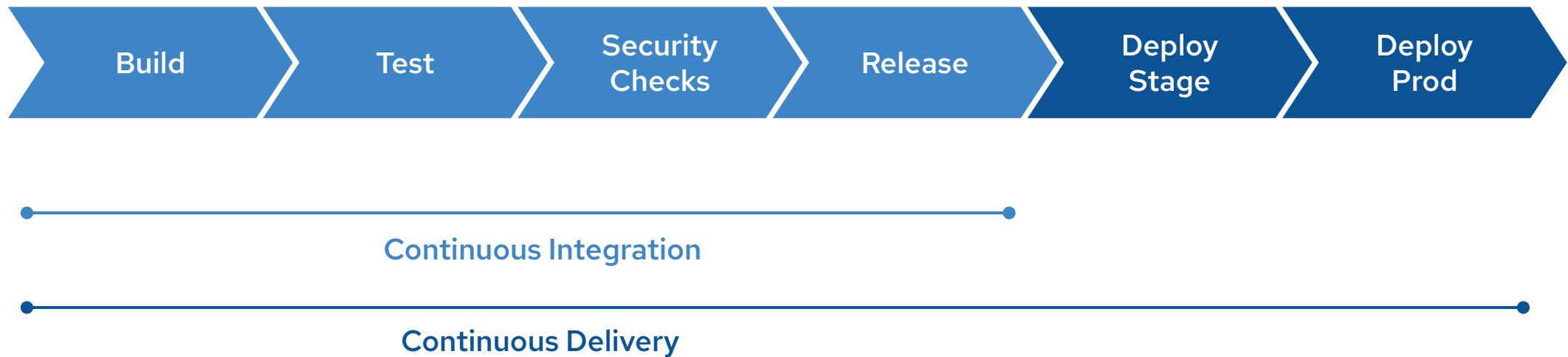
DevOps

=

**Let's have Dev and Ops TALK to each other**

# Continuous Integration(CI) & Continuous Delivery (CD)

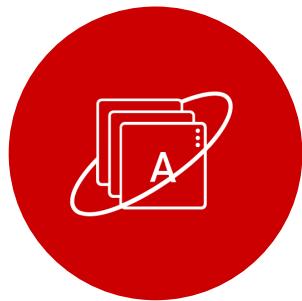
A key DevOps principle for automation, consistency and reliability



# Tekton / OpenShift Pipelines

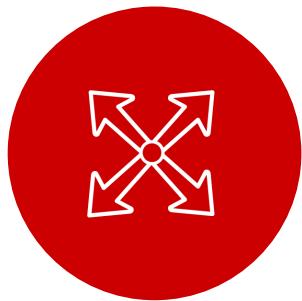
Kubernetes native on  
demand delivery  
pipelines

# What is Cloud-Native CI/CD?



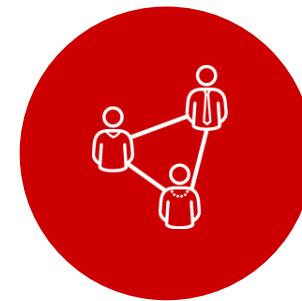
## Containers

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  
Plugins have 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 are fully isolated from each other  
Every pipeline has its own container 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



# TEKTON

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

63



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

# 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

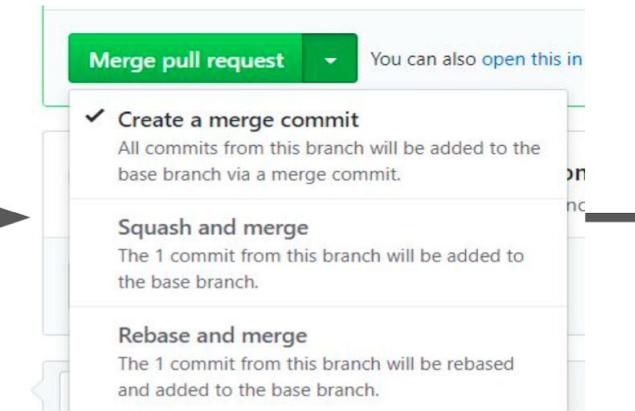
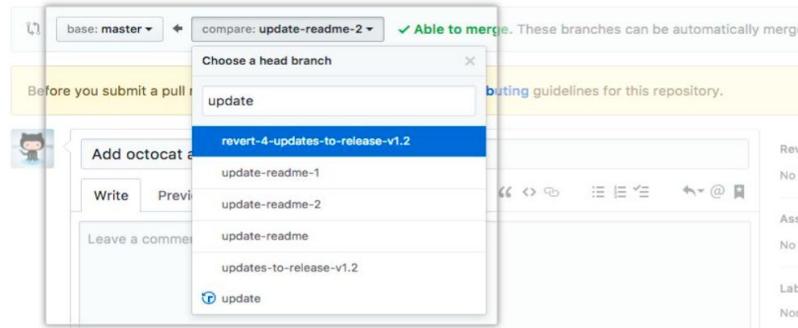
- OpenShift is a declarative environment
  - Cluster configuration is declared and Operators make it happen
  - Application deployments are declared and Kubernetes scheduler makes it happen
- GitOps in traditional environments requires automation/scripting, declarative environment minimizes or eliminates 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.

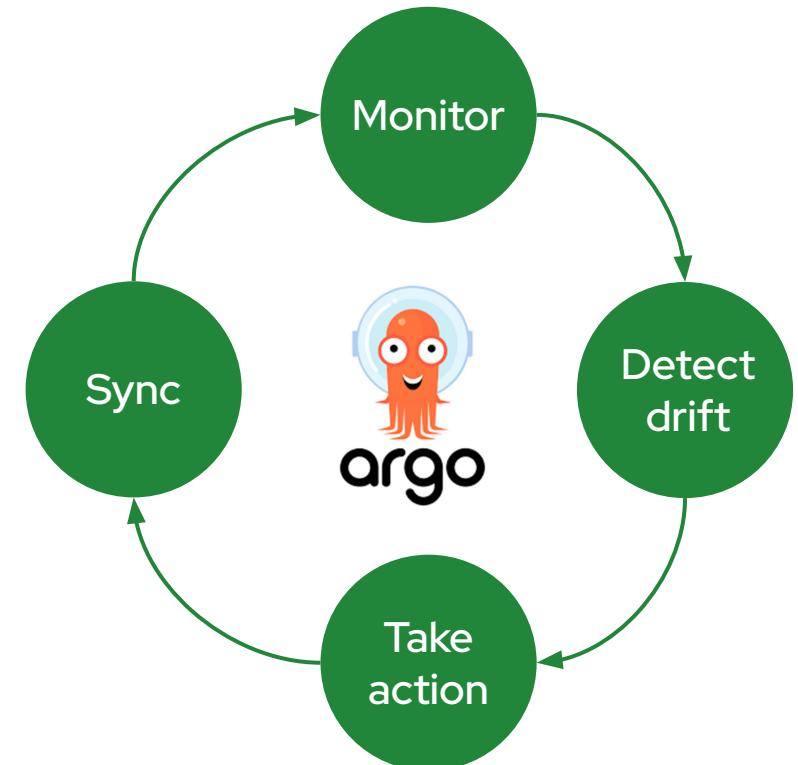


```
$ tkn pipelinerun logs update-from-master-run-g6s45

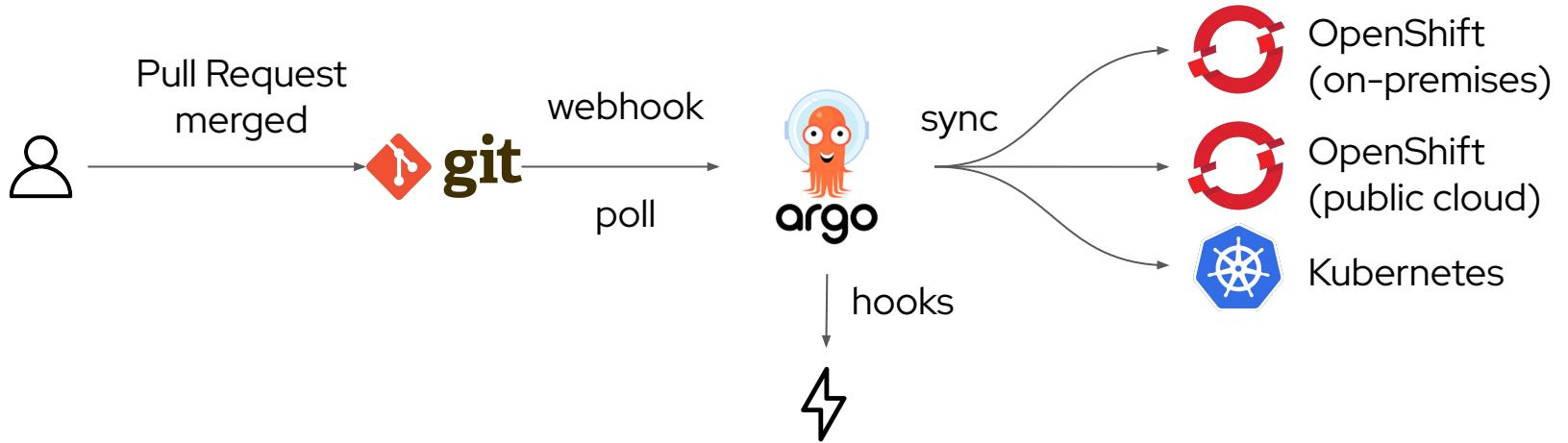
[run-kubectl] {"level": "info", "ts": 1580989837.3045664, "logger": "fallback-
logger", "caller": "logging/config.go:69", "msg": "Fetch GitHub commit ID from kodata failed:
\"KO_DATA_PATH\" does not exist or is empty"}
[run-kubectl] serviceaccount/demo-sa unchanged
[run-kubectl] clusterrolebinding.rbac.authorization.k8s.io/tekton-triggers-openshift-binding 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 "Healthy" and "Synced" to the latest commit (49147e1). A "synchronization" status message indicates it succeeded 38 minutes ago. The interface displays a complex dependency graph between multiple "redisapp" and "realtimeapp" components across different namespaces, showing deployment, replicaset, and pod stages.

# OpenShift GitOps



## Multi-cluster config management

Declaratively manage cluster and application configurations across multi-cluster OpenShift and Kubernetes infrastructure with Argo CD<sup>72</sup>



## 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

# Demo





# Further reading and resources...

## OpenShift for Developers and CI/CD explained:

- [Automated Application Packaging and Distribution with OpenShift - Basic Development - Part 1/4 - Open Sourcerers](#)
- [Automated Application Packaging And Distribution with OpenShift - Helm Charts and Operators - Part 2/4 - Open Sourcerers](#)
- [Automated Application Packaging And Distribution with OpenShift - Tekton Pipelines - Part 3/4 - Open Sourcerers](#)

# Thank you !