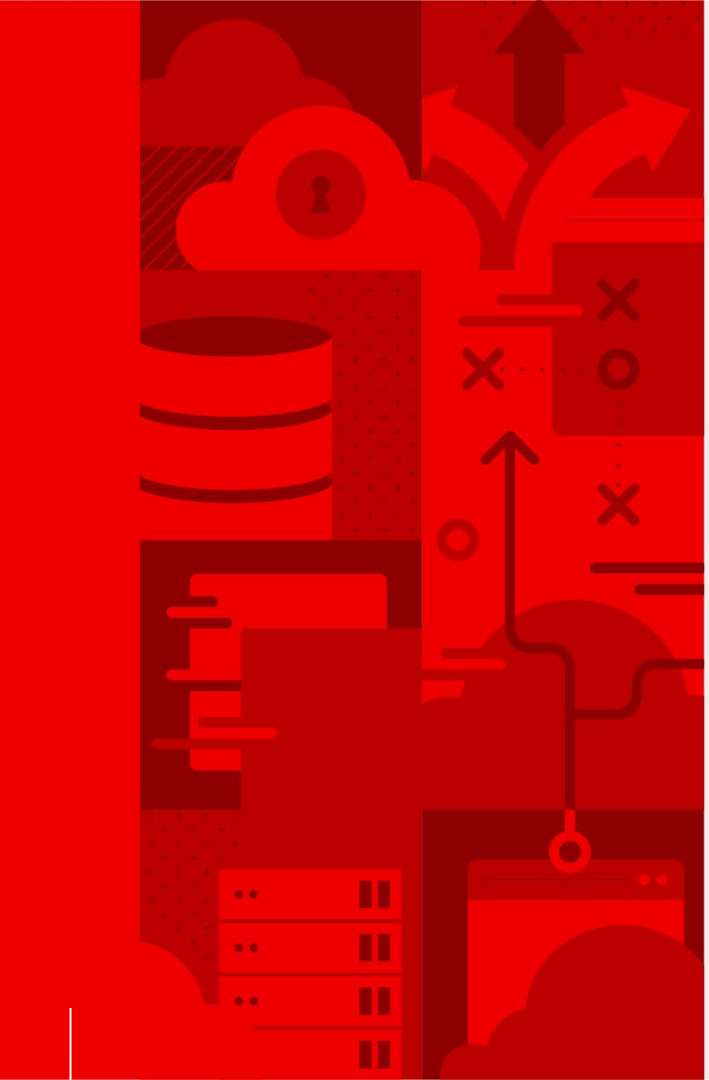


Kubernetes Native Developer

GENERAL DISTRIBUTION

Architecture Workshop

CI/CD and GitOps

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat

Self introduction

Name: Wanja Pernath

Email: wpernath@redhat.com

Base: Germany (very close to the Alps)

Role: EMEA Technical Partner Development Manager - OpenShift and MW

Experience: Years of Consulting, Training, PreSales at Red Hat and before

Twitter: https://twitter.com/wpernath

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

GitHub: https://github.com/wpernath

First book just published

Getting GitOps

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

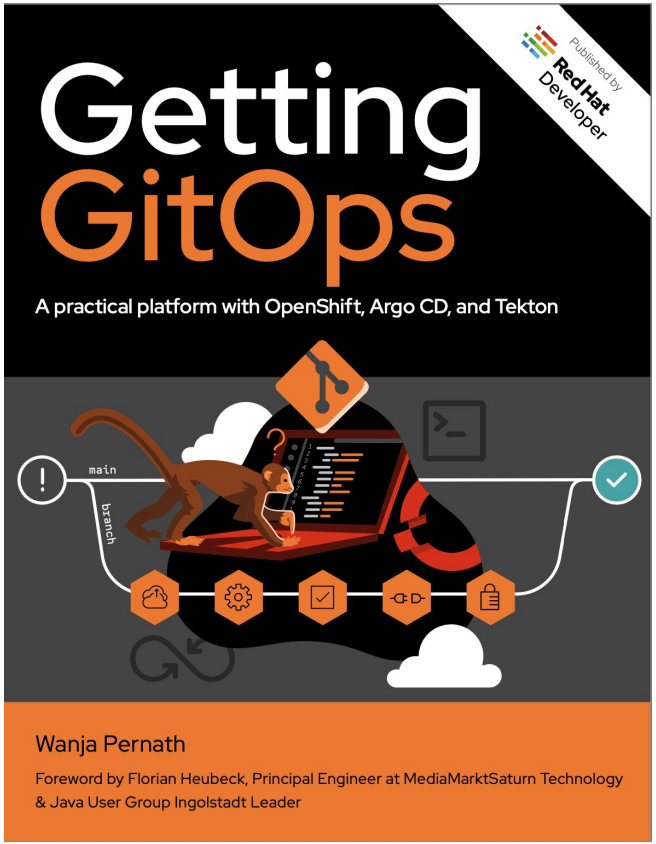
Technologies discussed:
Quarkus, Helm Charts, Kustomize, Tekton Pipelines, Kubernetes Operators, OpenShift Templates, ArgoCD, CI/CD, GitOps….

Download for free at:

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

Interview with full GitOps Demo:

https://www.youtube.com/watch?v=znMfVqAlRzY&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?

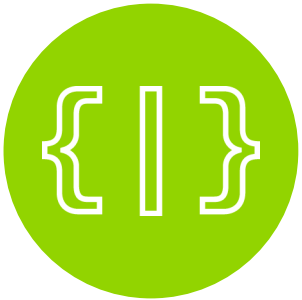
Learning Goals

- 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

BUILD & DEPLOY

BUILD AND DEPLOY CONTAINER IMAGES

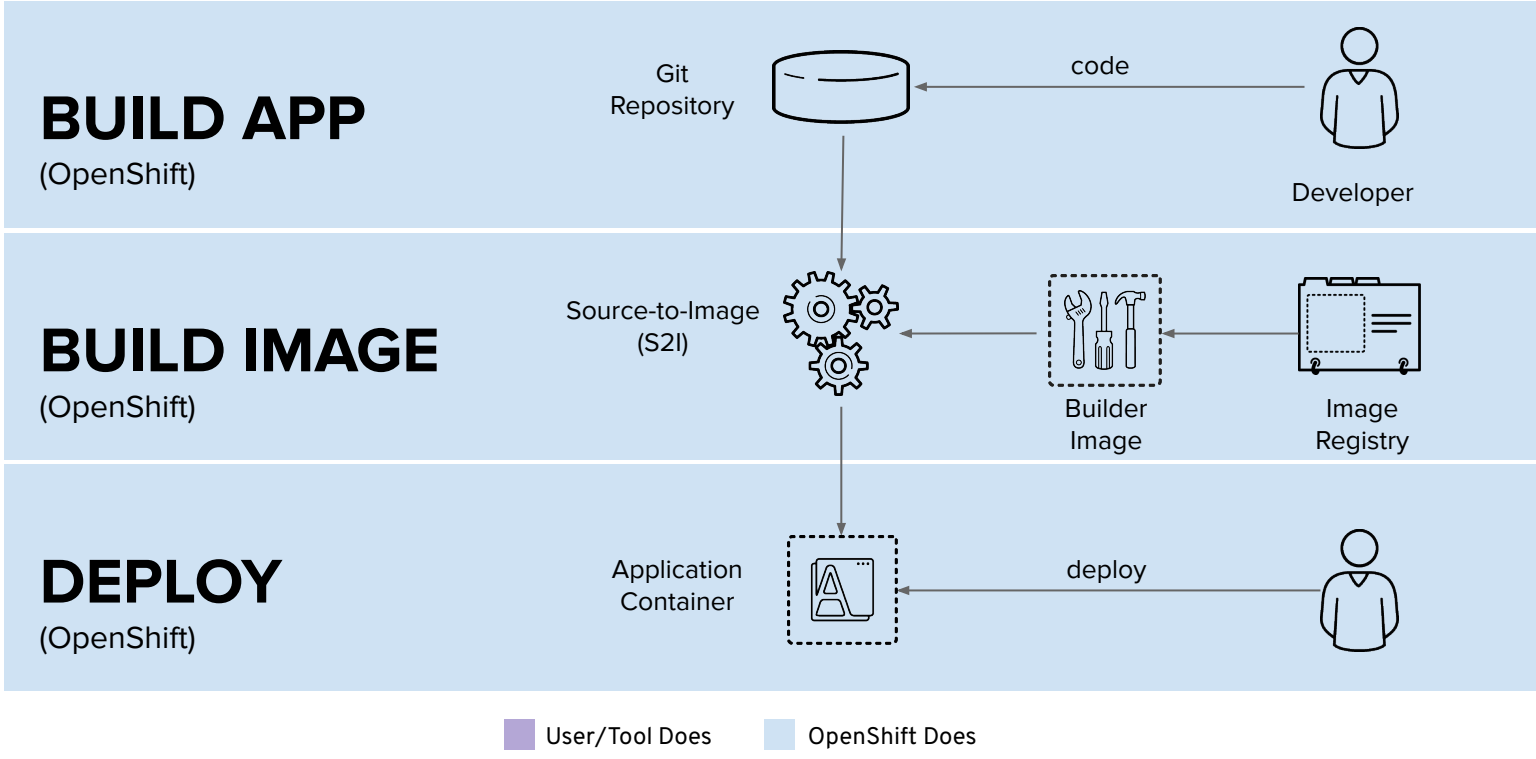


DEPLOY YOUR SOURCE CODE

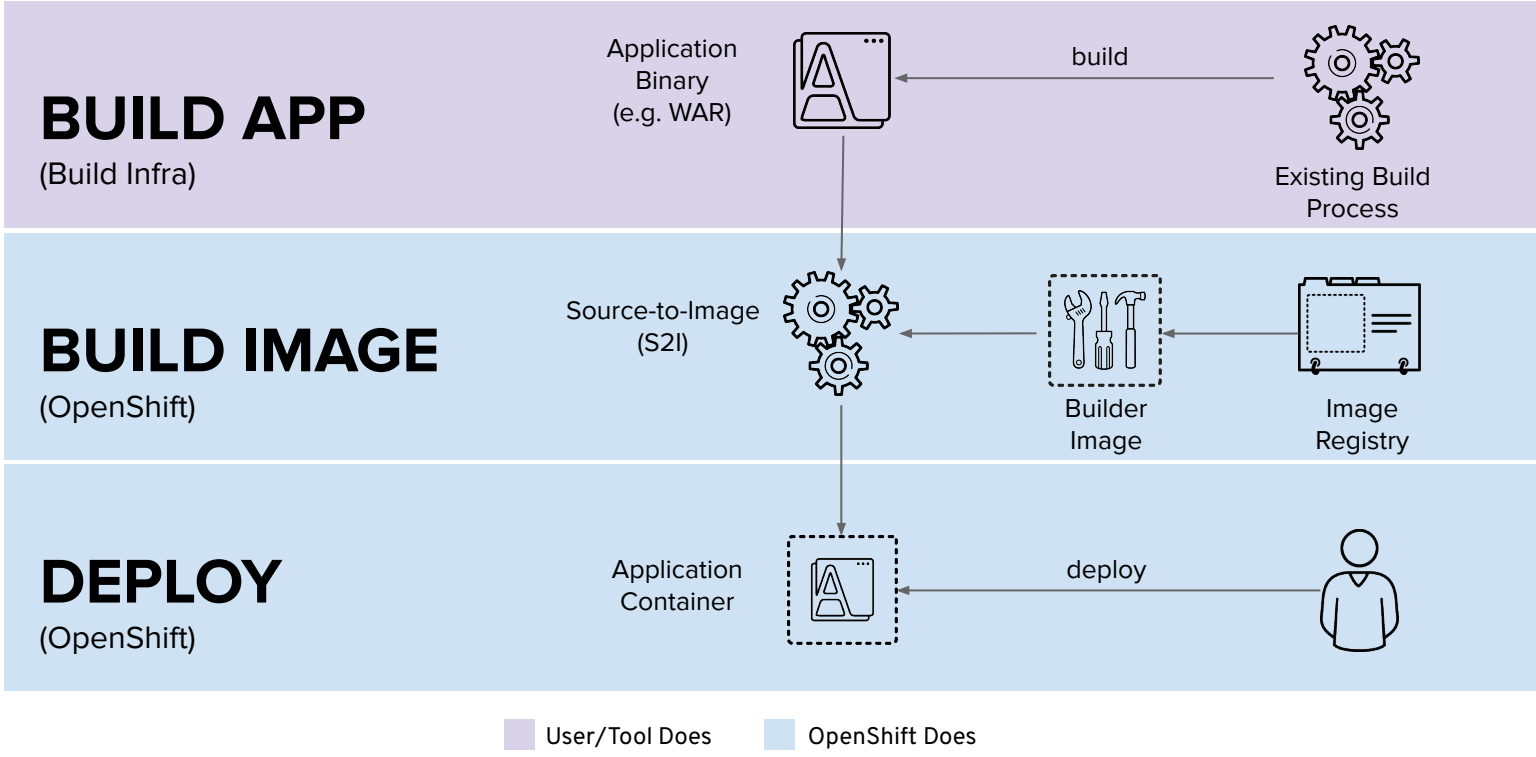
DEPLOY YOUR APP BINARY

DEPLOY YOUR CONTAINER IMAGE

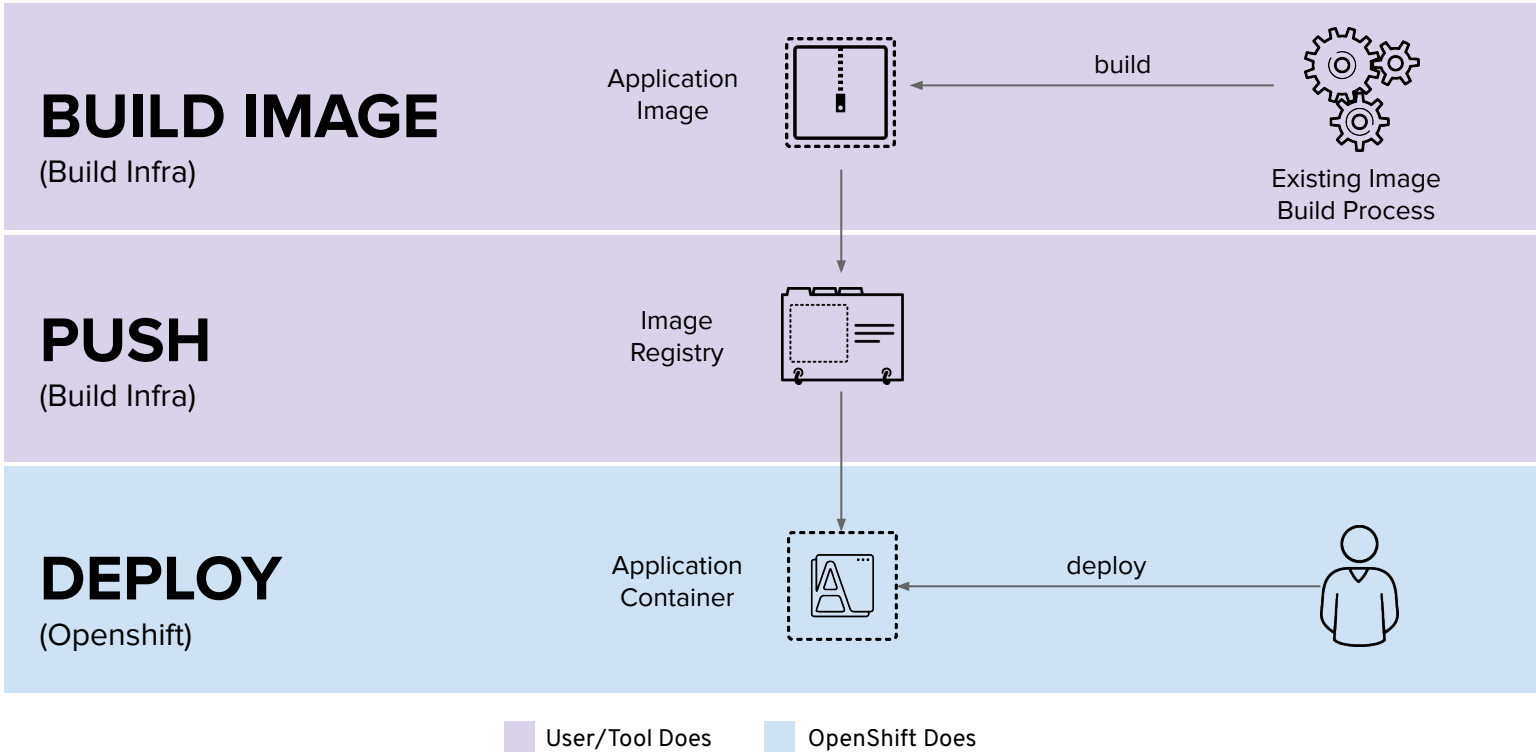
DEPLOY SOURCE CODE WITH SOURCE-TO-IMAGE (S2I)



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

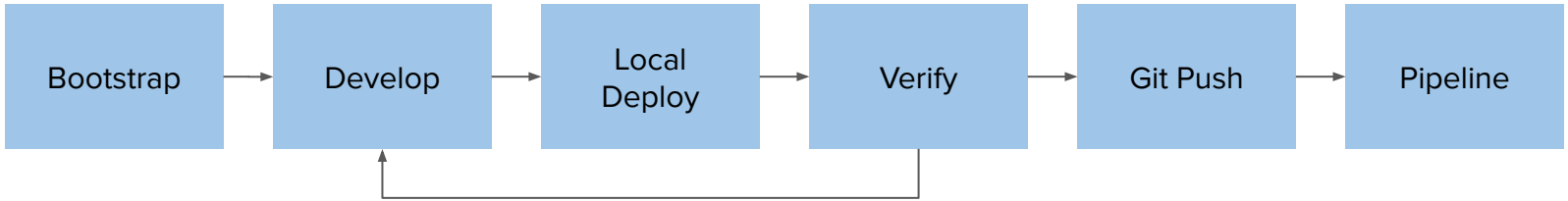


DEPLOY DOCKER IMAGE



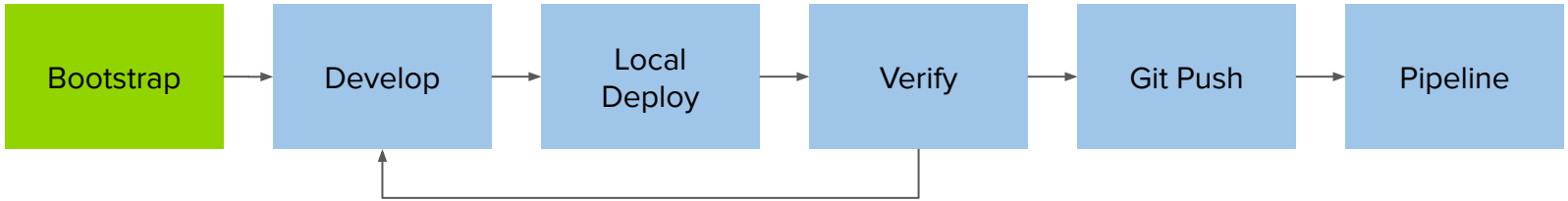
DEVELOPER WORKFLOW

LOCAL DEVELOPMENT WORKFLOW



RED HAT®
OPENSHIFT
Container Platform

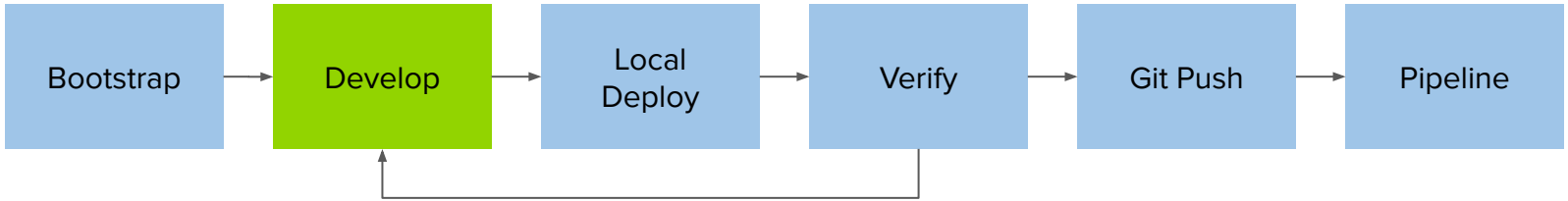
LOCAL DEVELOPMENT WORKFLOW



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

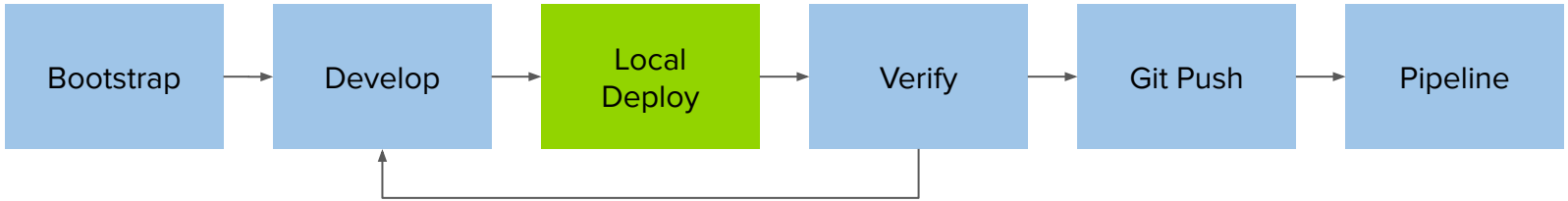
LOCAL DEVELOPMENT WORKFLOW



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

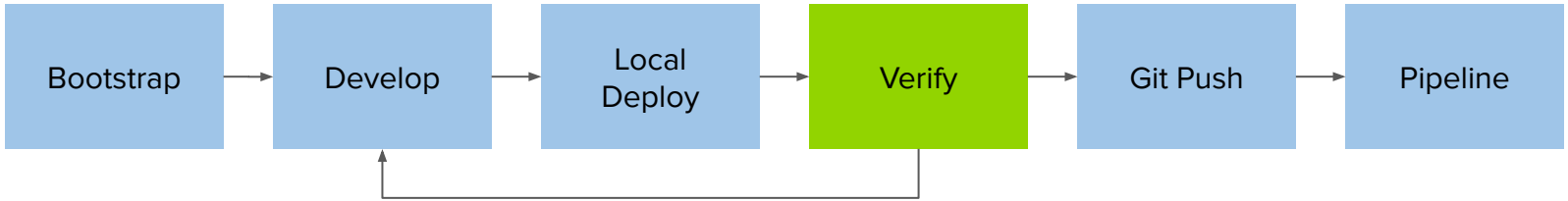
LOCAL DEVELOPMENT WORKFLOW



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

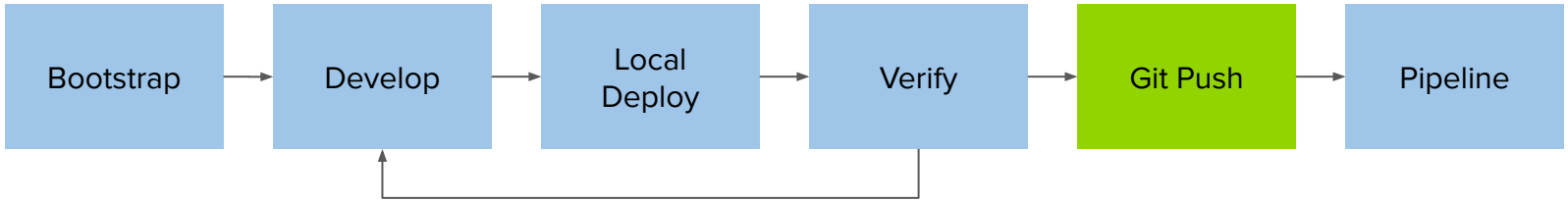
LOCAL DEVELOPMENT WORKFLOW



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

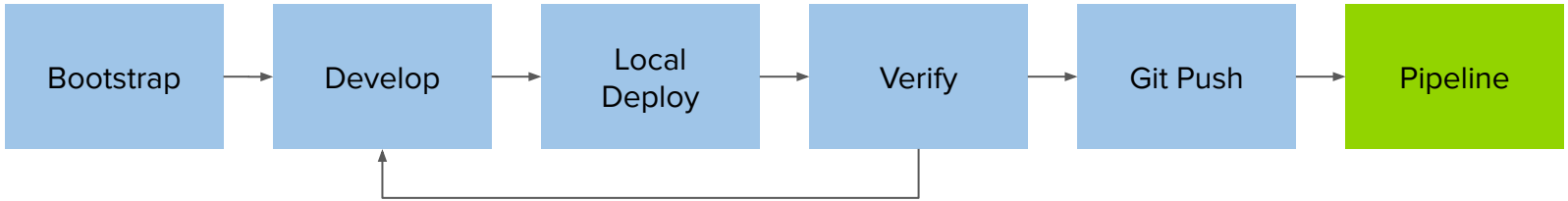
LOCAL DEVELOPMENT WORKFLOW



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

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


WE DO DEVOPS ALREADY
JUST WITHOUT AGILE, STANDUPS, CI, CODE OR ANY OF THAT OTHER HIPSTER STUFF
memes.com

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

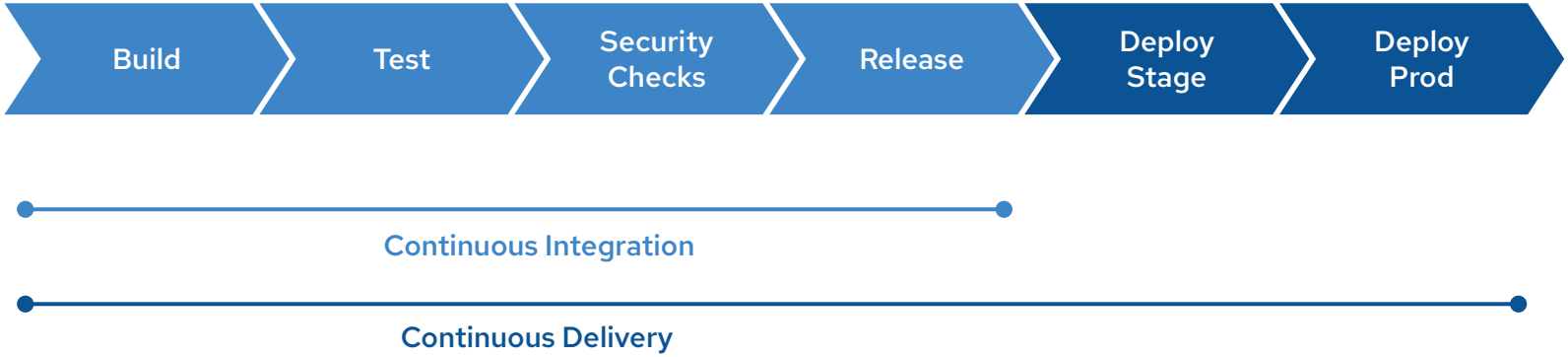
DevOps

=

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



Red Hat

Tekton / OpenShift Pipelines

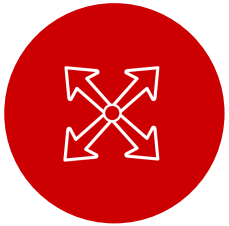
Kubernetes native on demand delivery pipelines

Red Hat

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	Cloud-Native CI/CD
Designed for Virtual Machines	Designed for Containers and Kubernetes
Require IT Ops for CI engine maintenance	Pipeline as a service with no Ops overhead
Plugins shared across CI engine	Pipelines fully isolated from each other
Plugin dependencies with undefined update cycles	Everything lifecycled as container images
No interoperability with Kubernetes resources	Native Kubernetes resources
Admin manages persistence	Platform manages persistence
Config baked into CI engine container	Configured via Kubernetes ConfigMaps

Why Cloud-Native CI/CD?

Traditional CI/CD	Cloud-Native CI/CD
Designed for Virtual Machines	Designed for Containers and Kubernetes
Require IT Ops for CI engine maintenance	Pipeline as a service with no Ops overheard
Plugins shared across CI engine	Pipelines fully isolated from each other
Plugin dependencies with undefined update cycles	Existing library of tasks and images
No interoperability with Kubernetes resources	Native Kubernetes resources
Admin manages persistence	Platform manages persistence
Config baked into CI engine container	Configured via Kubernetes ConfigMaps

Jenkins

TEKTON

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

Powered by TEKTON

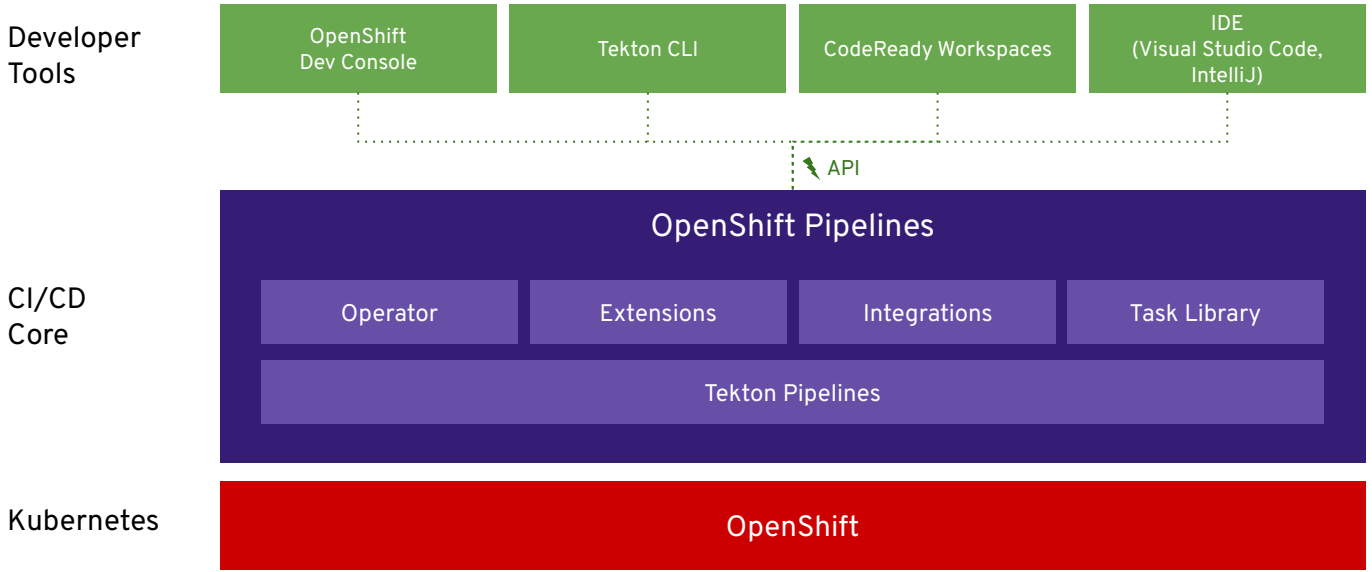
Red Hat

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

Governed by the Continuous Delivery Foundation

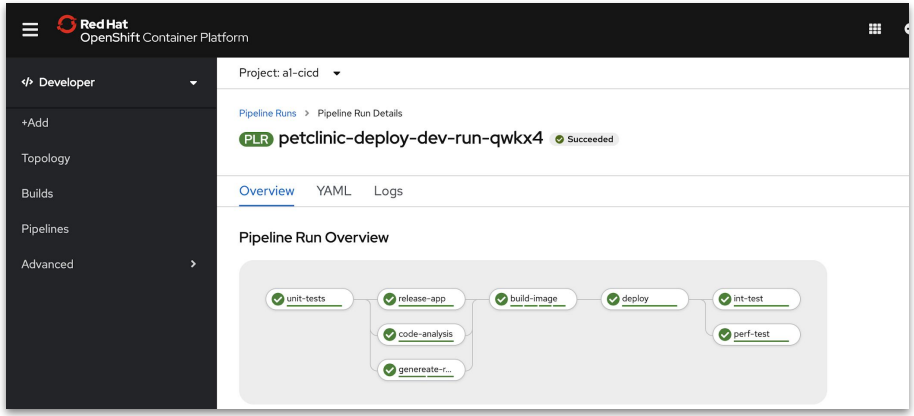
Contributions from Google, Red Hat, Cloudbees, IBM, Pivotal and many more

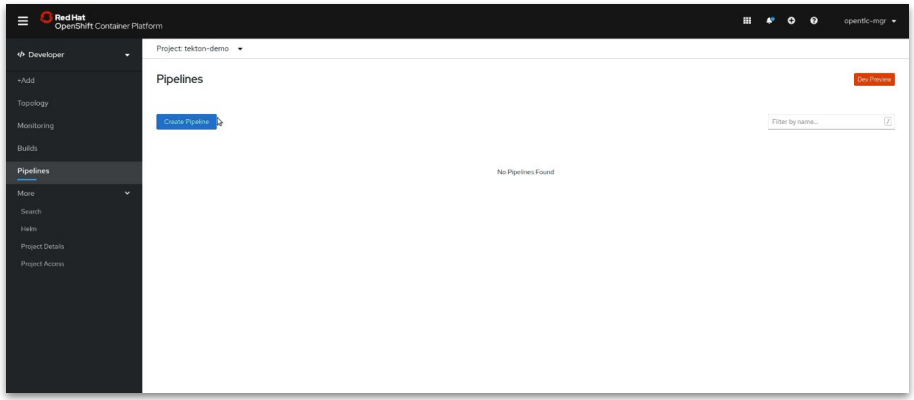
OpenShift Pipelines



OpenShift Pipelines

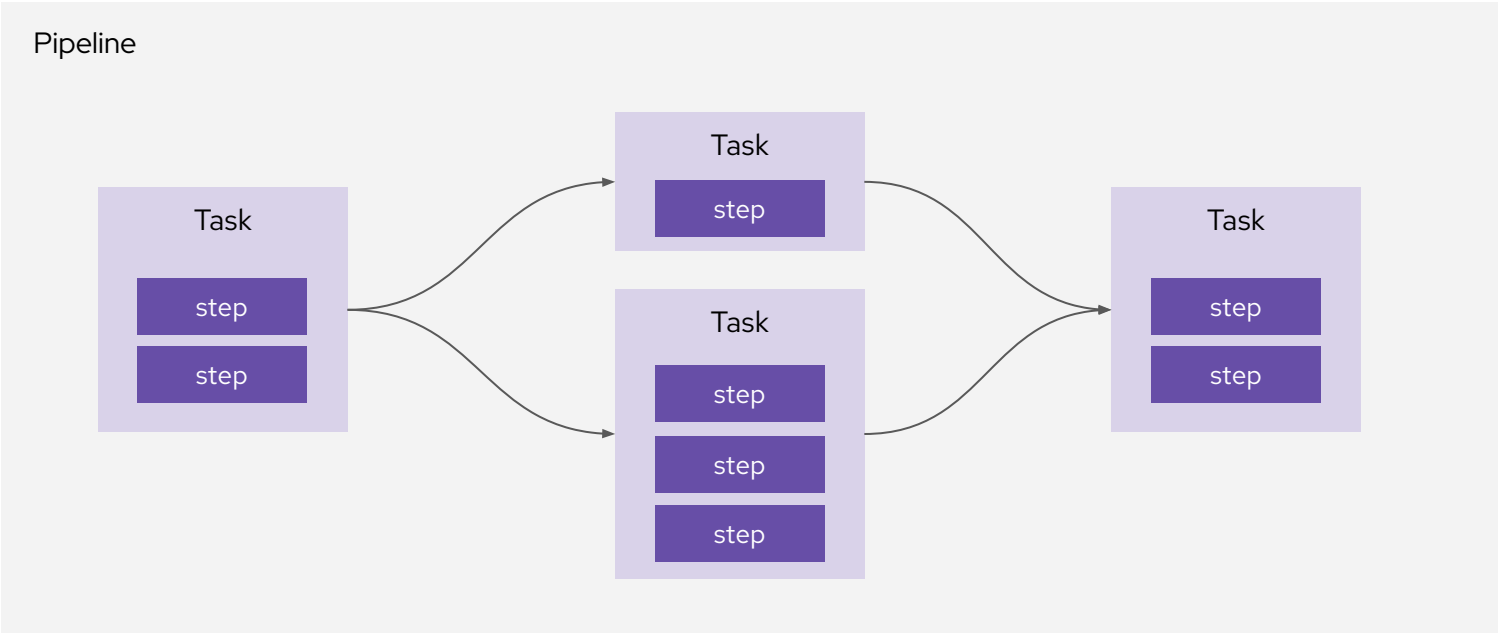
- Based on Tekton Pipelines
- Kubernetes-native declarative CI/CD
- Pipelines run on-demand in isolated containers
- No central server to maintain! No plugin conflicts!
- Task library and integration with Tekton Hub
- Secure pipelines aligned with Kubernetes RBAC
- Visual and IDE-based pipeline authoring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





Red Hat

Tekton Concepts



Tekton Concepts: `step`

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

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

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

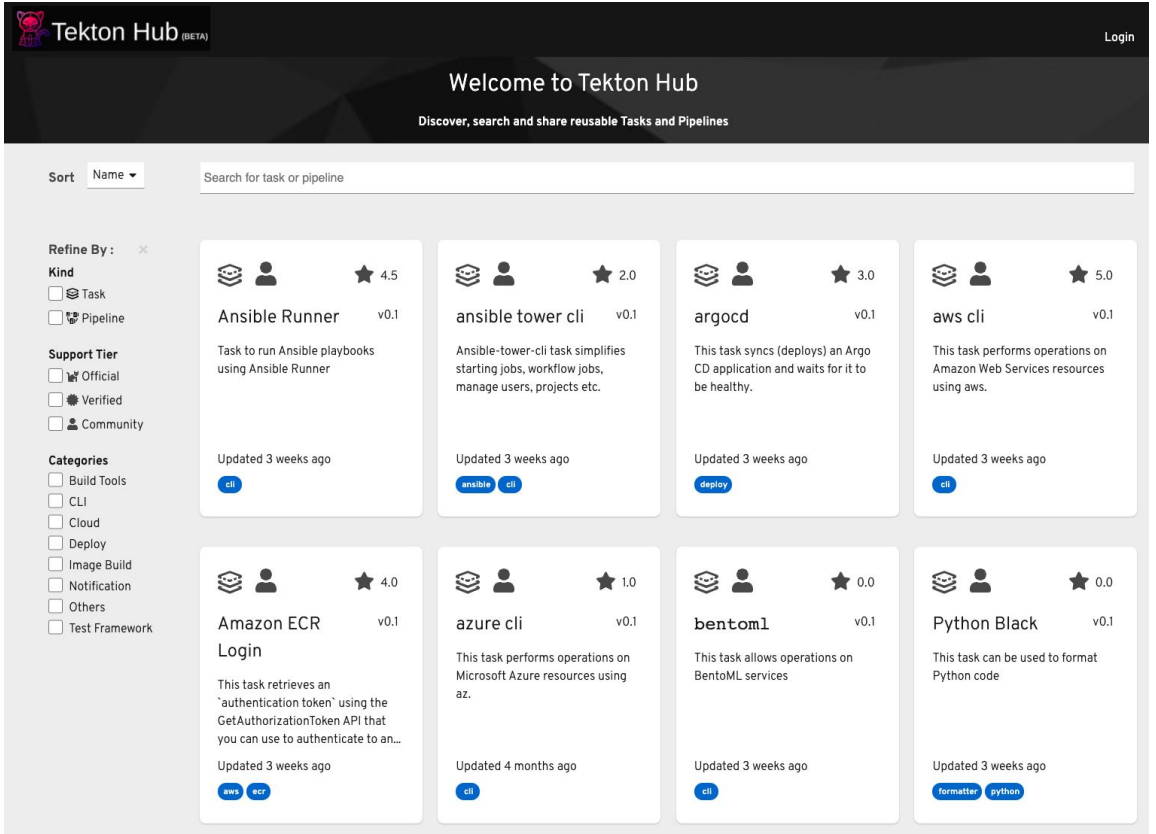
Tekton Concepts: Task

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

```yaml
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

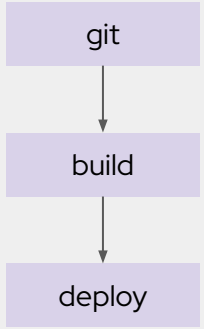


Red Hat

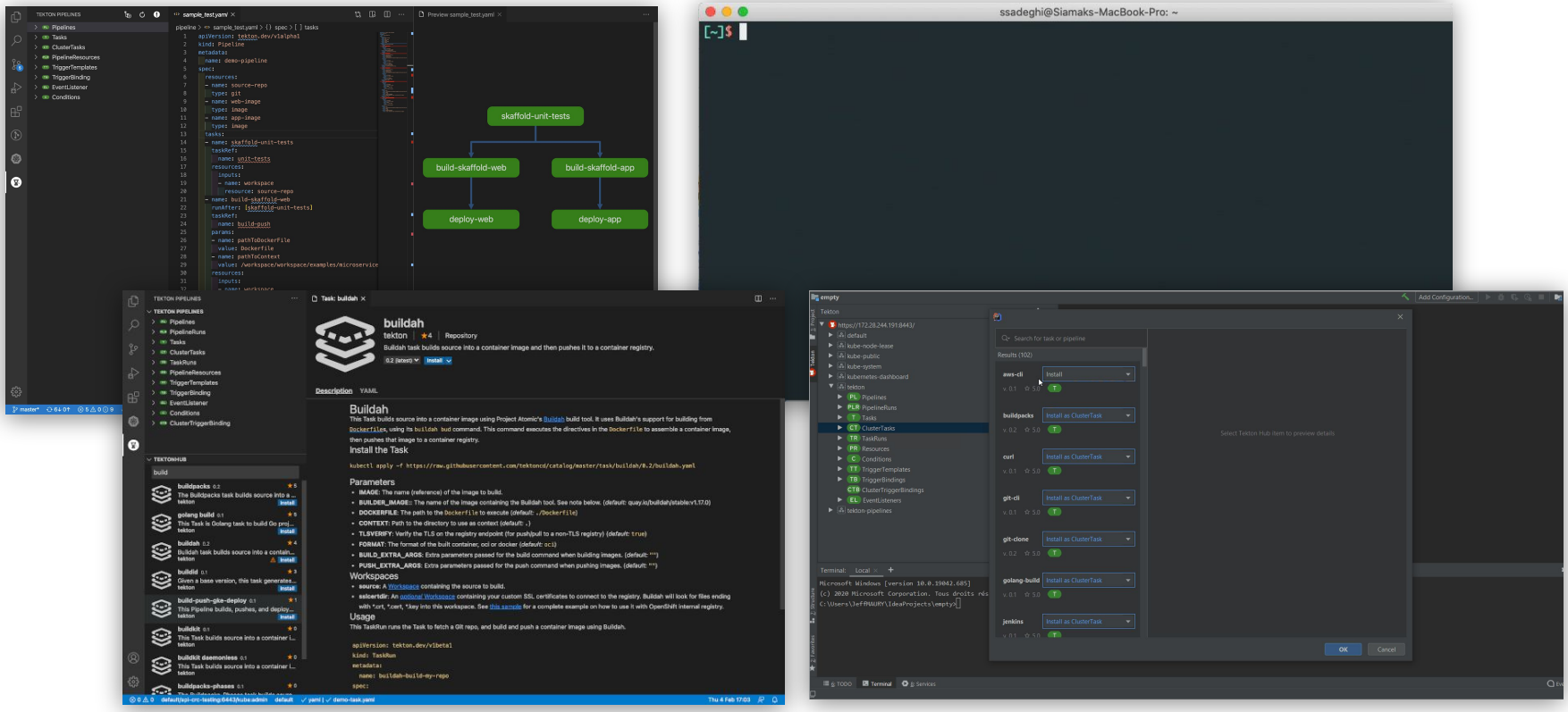
Tekton Concepts: `Pipeline`

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

```yaml
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



Red Hat

DEMO TIME

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

Red Hat

OpenShift GitOps

Declarative GitOps for multi-cluster continuous delivery

Red Hat

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!

Environments are all manually configured!!!

Production deployments have a low success rate!

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

I can't audit config changes!!

I can't easily rollback changes to a specific version

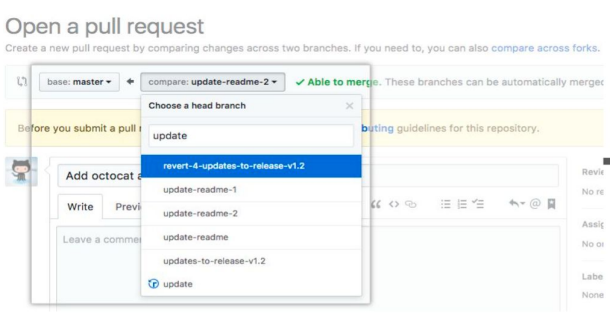
GitOps Benefits

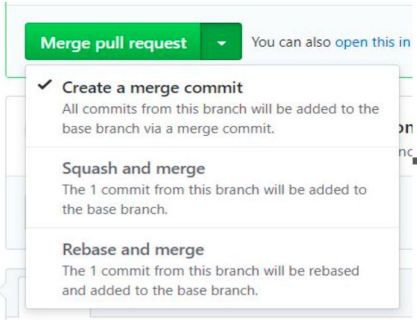
- All changes are auditable
- Standard roll-forward or backwards in the event of failure
- Disaster recovery is “reapply the current state of the manifests”
- Experience is “pushes and pull-requests”

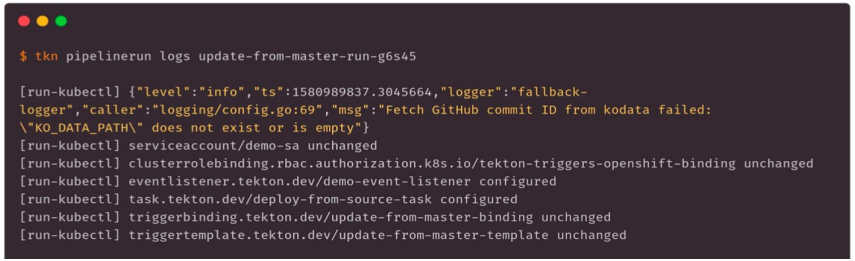
OpenShift and GitOps - A great match

- 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



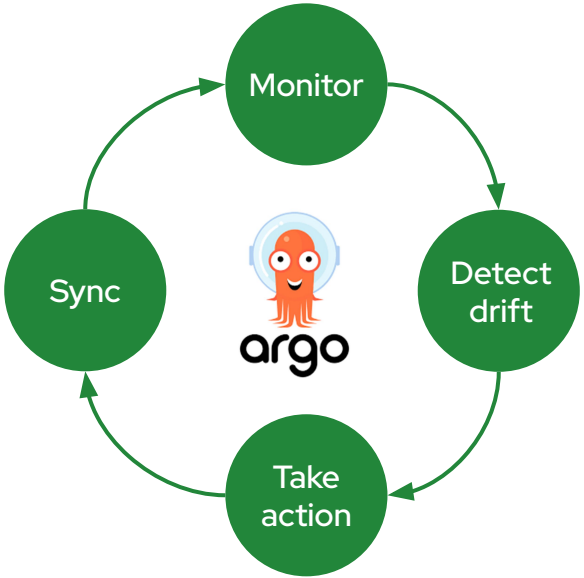




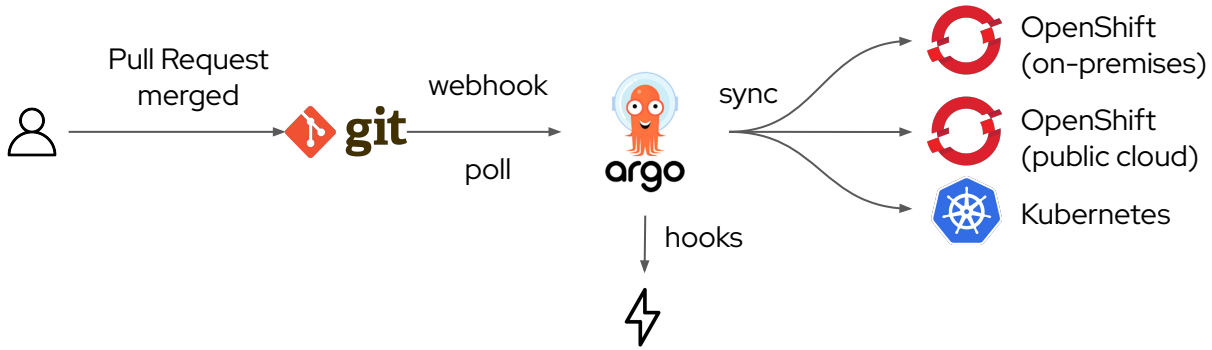
Red Hat

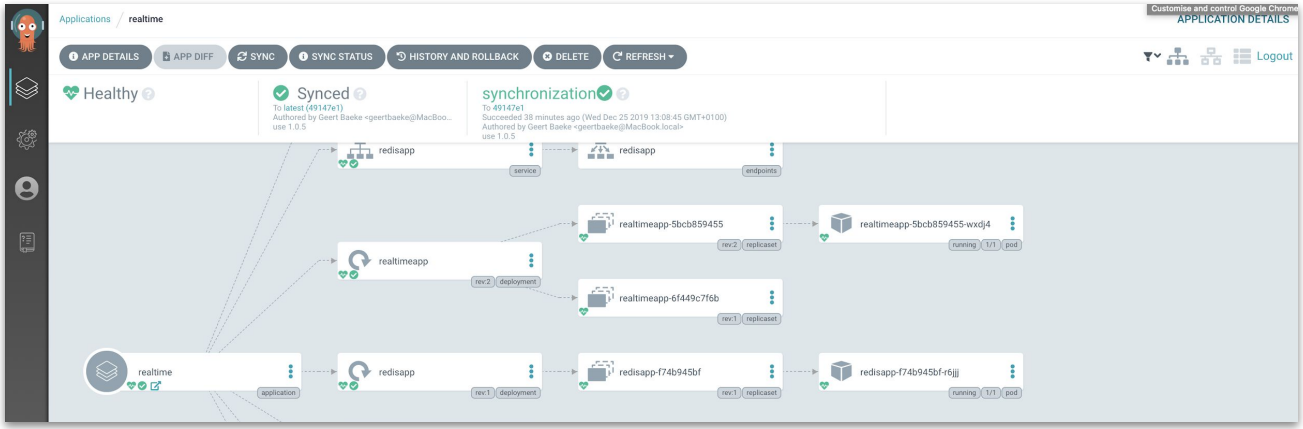
Argo CD

- 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

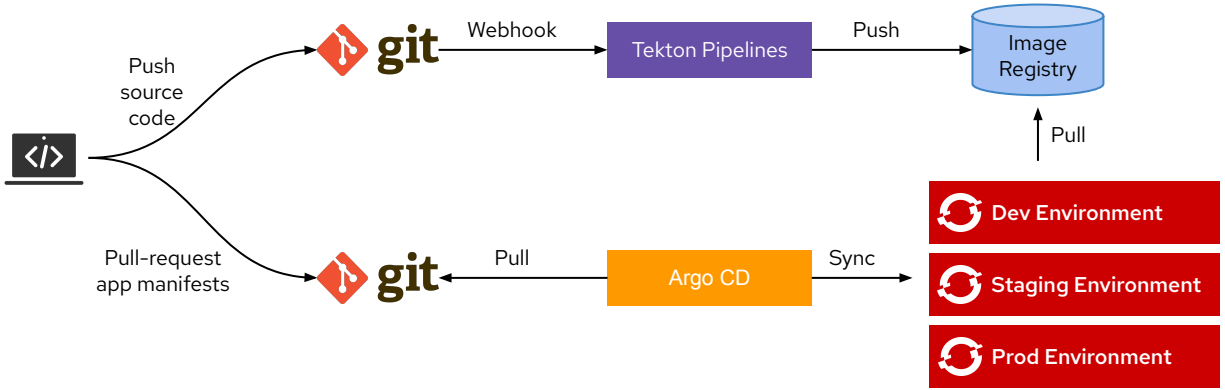




Red Hat

GitOps Application Manager CLI

```
$ kam bootstrap
```



Red Hat

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

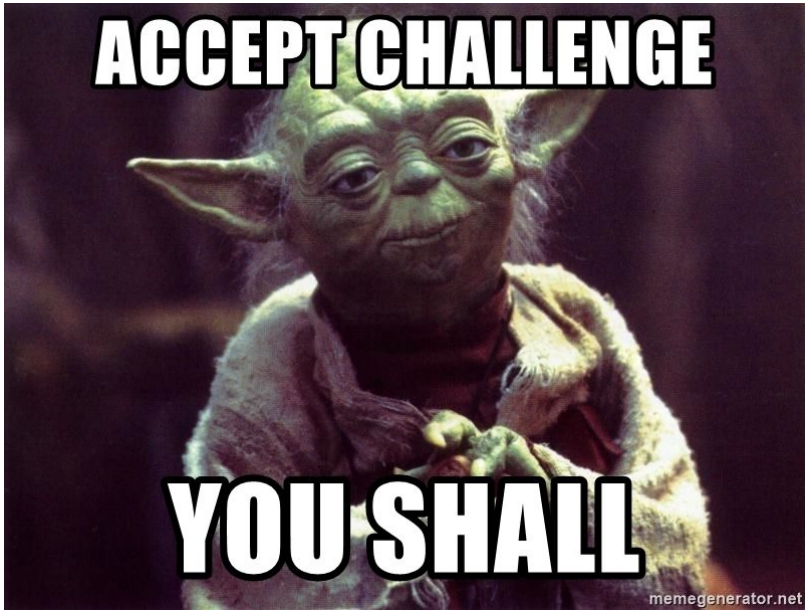
Visibility into application deployments across environments and the history of deployments in the OpenShift Console

Powered by argo

Red Hat

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

```
├── apps
│   └── app-1
│       ├── base
│       │   └── kustomization.yaml
│       └── dev
│           ├── deployment.yaml
│           └── kustomization.yaml
├── envs
│   ├── base
│   │   ├── 205-serviceaccount.yaml
│   │   └── kustomization.yaml
│   └── dev
│       └── kustomization.yaml
└── services
    └── service-a
        ├── base
        │   ├── config
        │   │   ├── 300-deployment.yaml
        │   │   ├── 310-service.yaml
        │   │   └── kustomization.yaml
        │   └── kustomization.yaml
        └── dev
            ├── dev-deployment.yaml
            ├── dev-service.yaml
            └── kustomization.yaml
```

```
├── base
│   ├── deployment.yaml
│   ├── kustomization.yaml
│   └── service.yaml
└── overlays
    ├── development
    │   ├── kustomization.yaml
    │   └── replicas.yaml
    ├── production
    │   ├── kustomization.yaml
    │   ├── replicas.yaml
    │   └── volumes.yaml
    └── staging
        ├── kustomization.yaml
        └── volumes.yaml
```

```
├── 00-tekton
│   ├── release.notags.yaml
│   └── release.yaml
├── 01-namespaces
│   ├── cicd-environment.yaml
│   ├── dev-environment.yaml
│   └── stage-environment.yaml
├── 02-serviceaccount
│   ├── demo-sa-admin-dev.rolebinding.yaml
│   ├── demo-sa-admin-stage.rolebinding.yaml
│   ├── role-binding.yaml
│   ├── role.yaml
│   └── serviceaccount.yaml
├── 03-tasks
│   ├── buildah-task.yaml
│   ├── create-github-status-task.yaml
│   ├── deploy-from-source-task.yaml
│   └── deploy-using-kubectl-task.yaml
├── 04-templatesandbindings
│   ├── dev-cd-deploy-from-master-binding.yaml
│   ├── dev-cd-deploy-from-master-template.yaml
│   ├── dev-ci-build-from-pr-binding.yaml
│   ├── dev-ci-build-from-pr-template.yaml
│   ├── stage-cd-deploy-from-push-binding.yaml
│   ├── stage-cd-deploy-from-push-template.yaml
│   ├── stage-ci-dryrun-from-pr-binding.yaml
│   └── stage-ci-dryrun-from-pr-template.yaml
├── 05-ci
│   ├── dev-ci-pipeline.yaml
│   └── stage-ci-pipeline.yaml
├── 06-cd
│   ├── dev-cd-pipeline.yaml
│   └── stage-cd-pipeline.yaml
├── 07-eventlisteners
│   └── cicd-event-listener.yaml
└── 08-routes
    └── github-webhook-event-listener.yaml
```

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

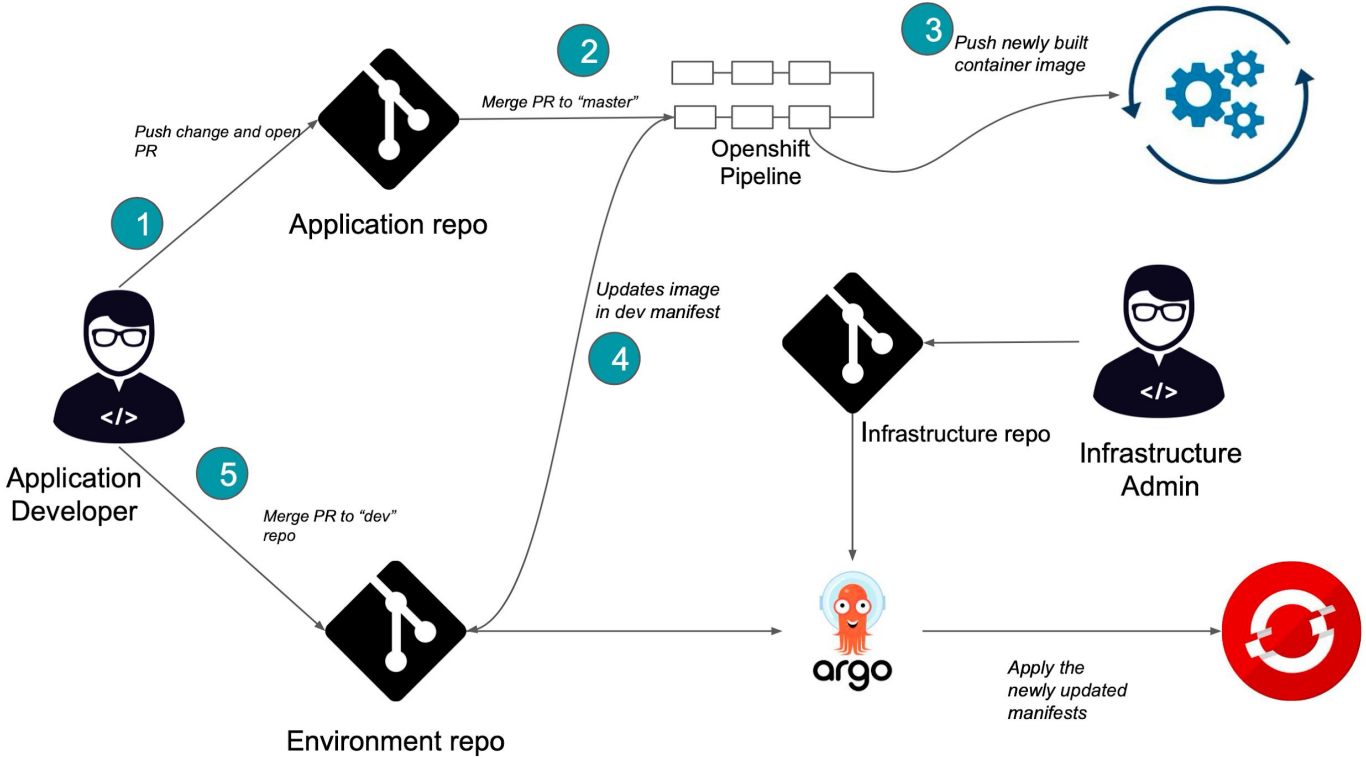
- There can be instances where you need to deploy something which cannot fully be done in a declarative way and it must be scripted
- Try minimizing this and leverage kubernetes primitives where possible
  - Init Containers
  - Jobs
  - Operators
- ArgoCD Resource Hooks
  - Hooks are ways to run scripts 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	


1
Push change and open PR
Application repo
2
Merge PR to "master"
Openshift Pipeline
3
Push newly built container image
Application Developer
4
Updates image in dev manifest
Infrastructure repo
Infrastructure Admin
5
Merge PR to "dev" repo
Environment repo
argo
Apply the newly updated manifests

DEMO

DEMO TIME

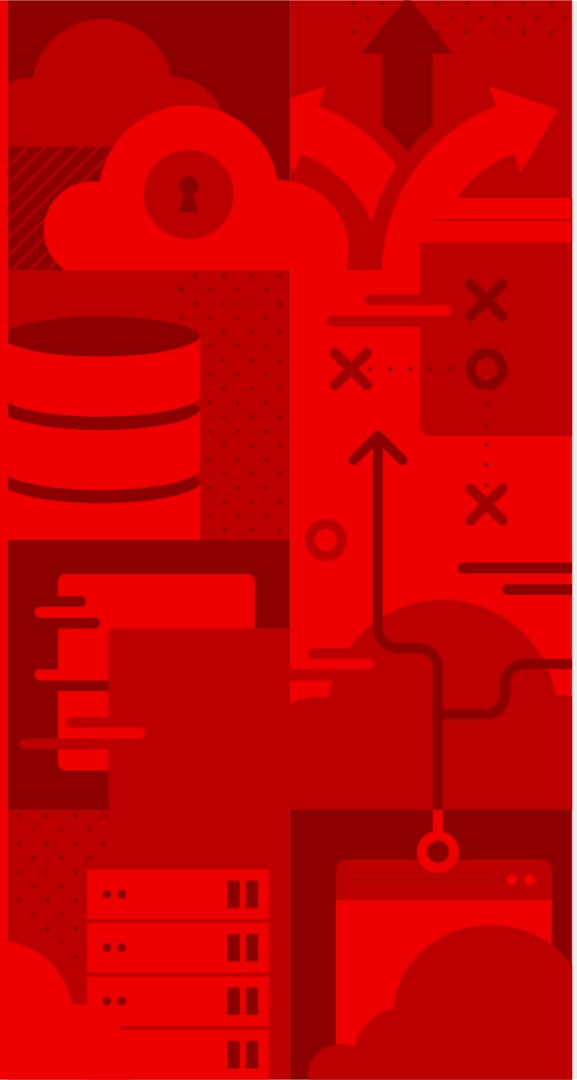
https://github.com/wpernath/openshift-cicd-demo

Red Hat

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



Thank you

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat