

OpenShift 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



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



Agenda / etc.

Agenda

- CI/CD with OpenShift
 - Basics
 - OpenShift Builds
 - OpenShift Pipelines
 - GitOps
 - Summary

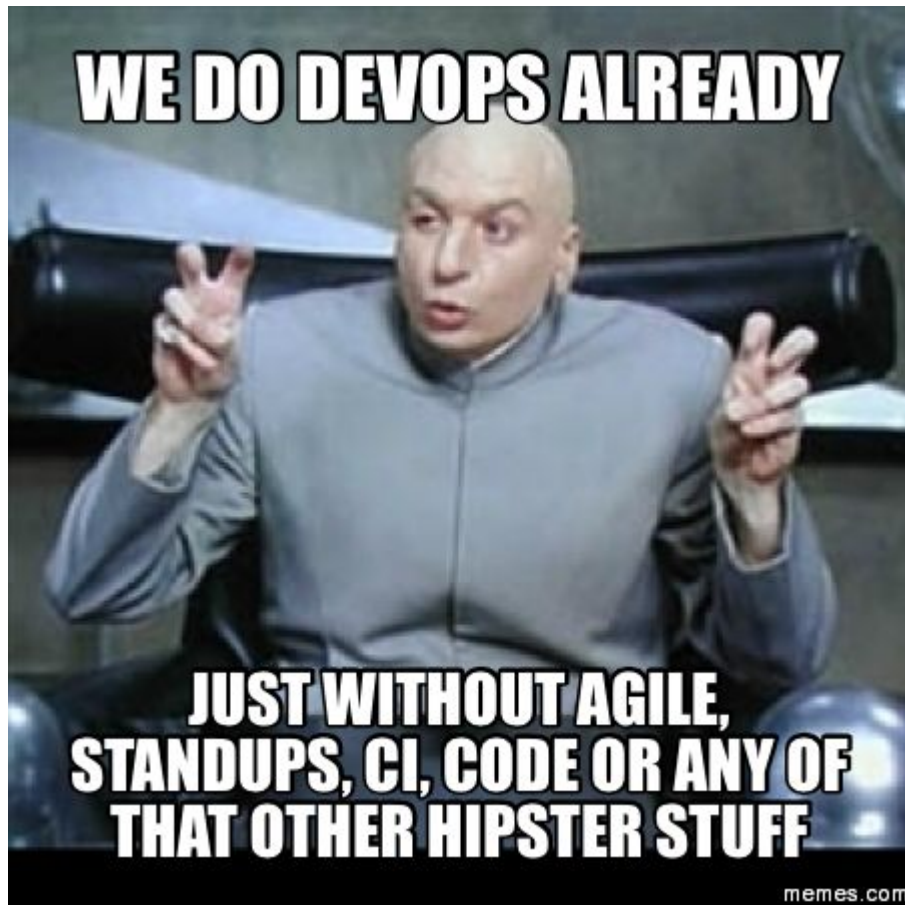
CI/CD with OpenShift

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

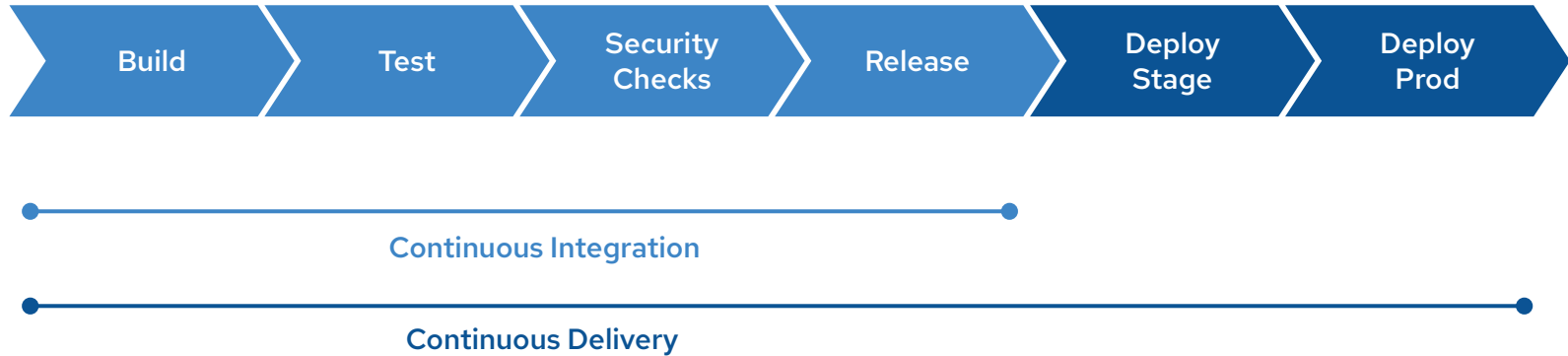
DevOps

=

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



OpenShift Builds

Automate building
container images using
Kubernetes tools

OpenShift Builds



Kubernetes-native image build

A Kubernetes-native way to building container images on OpenShift which is portable across Kubernetes distros



Supports multiple build strategies

Choose the build strategy that fits best your applications and skills: source-to-image, Dockerfile, and Cloud-Native Buildpacks

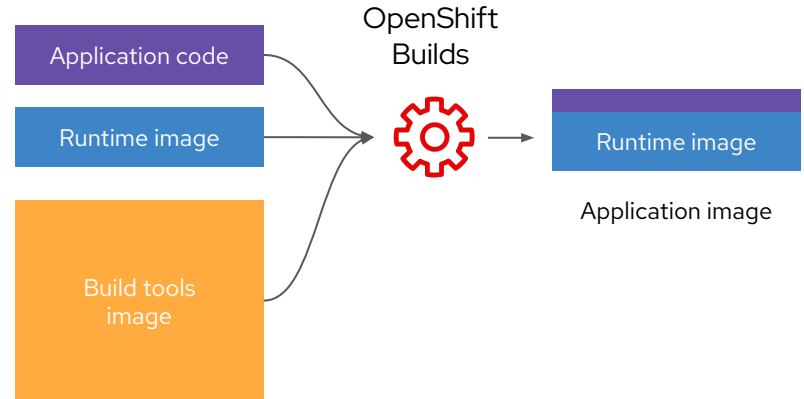


Extend with additional build strategies

Extend to use community Kubernetes builds strategies or your own custom builds

OpenShift Builds

- Build images on OpenShift and Kubernetes
- Use Kubernetes builds tools
 - Source-to-Image
 - Buildpacks
 - Buildah
 - Kaniko
 - ...more
- Create lean application images
- Extend with your own build tools
- Based on Shipwright open-source project



OpenShift Builds

BuildStrategy

How to build images e.g. S2I, Buildpacks, etc

Build

What to build

BuildRun

Build execution details

OpenShift Builds

Cloud-Native Buildpacks

```
kind: Build
metadata:
  name: myapp-buildpack
spec:
  source:
    url: https://github.com/myorg/myapp
  strategy:
    name: buildpacks-v3
  builder:
    image: paketobuildpacks/builder:full
  output:
    image: quay.io/myorg/myapp:v1
```

Source-to-Image (S2I)

```
kind: Build
metadata:
  name: myapp-s2i
spec:
  source:
    url: https://github.com/myorg/myapp
  strategy:
    name: source-to-image
  builder:
    image:
      registry.redhat.io/openjdk/openjdk-11-rhel8
  output:
    image: quay.io/myorg/myapp:v1
  runtime:
    image: docker.io/openjdk:11-jre-slim
```


Tekton / OpenShift Pipelines

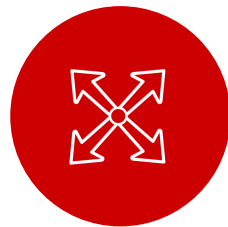
Kubernetes native on
demand delivery
pipelines

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

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

Extensible via Kubernetes images

TEKTON

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

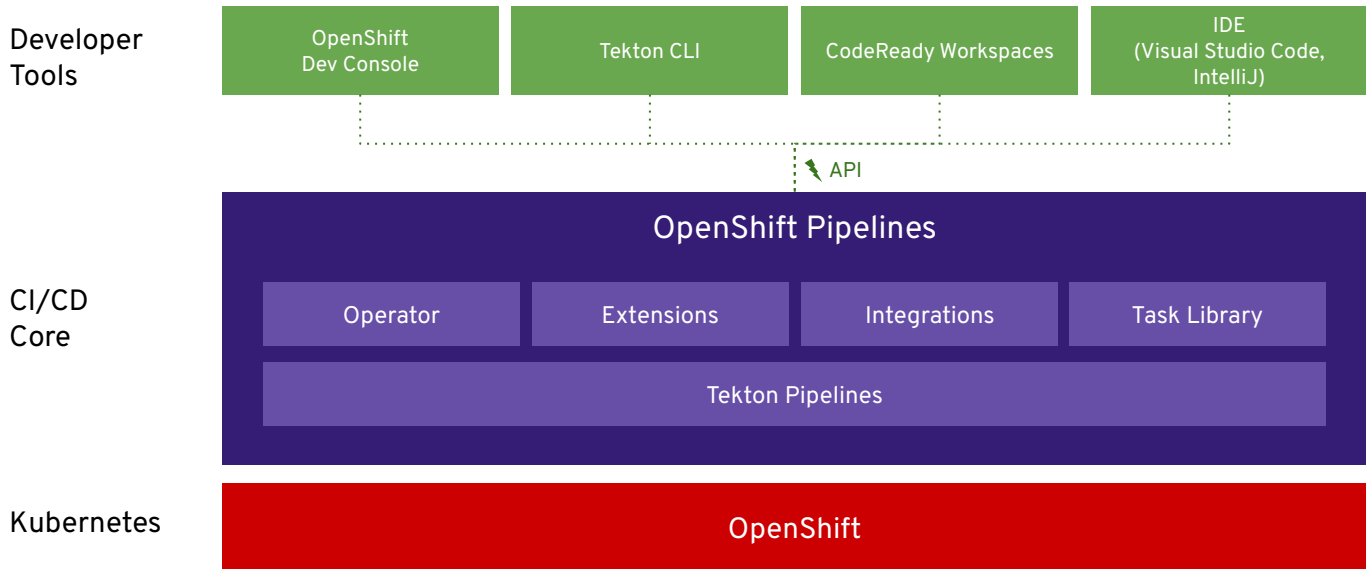


CD.FOUNDATION

Governed by the Continuous Delivery Foundation

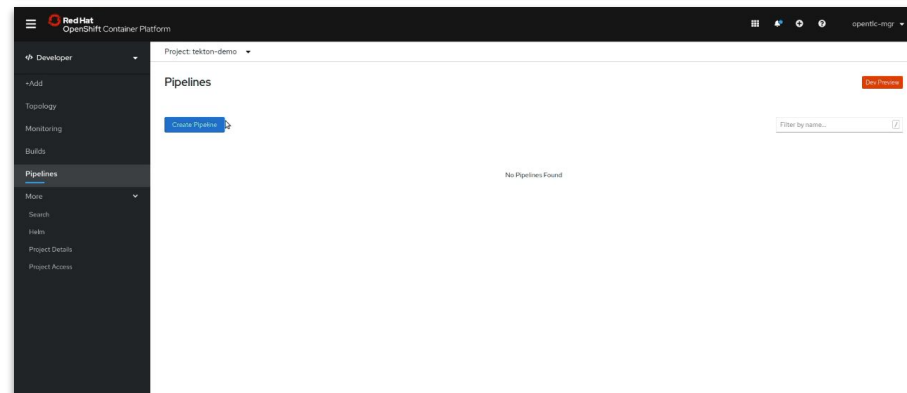
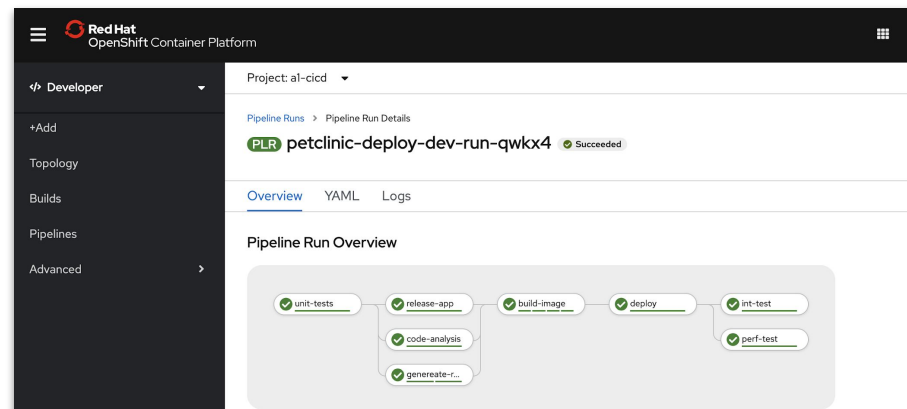
Contributions from Google, Red Hat, Cloudbees, IBM, Pivotal and many more

OpenShift Pipelines

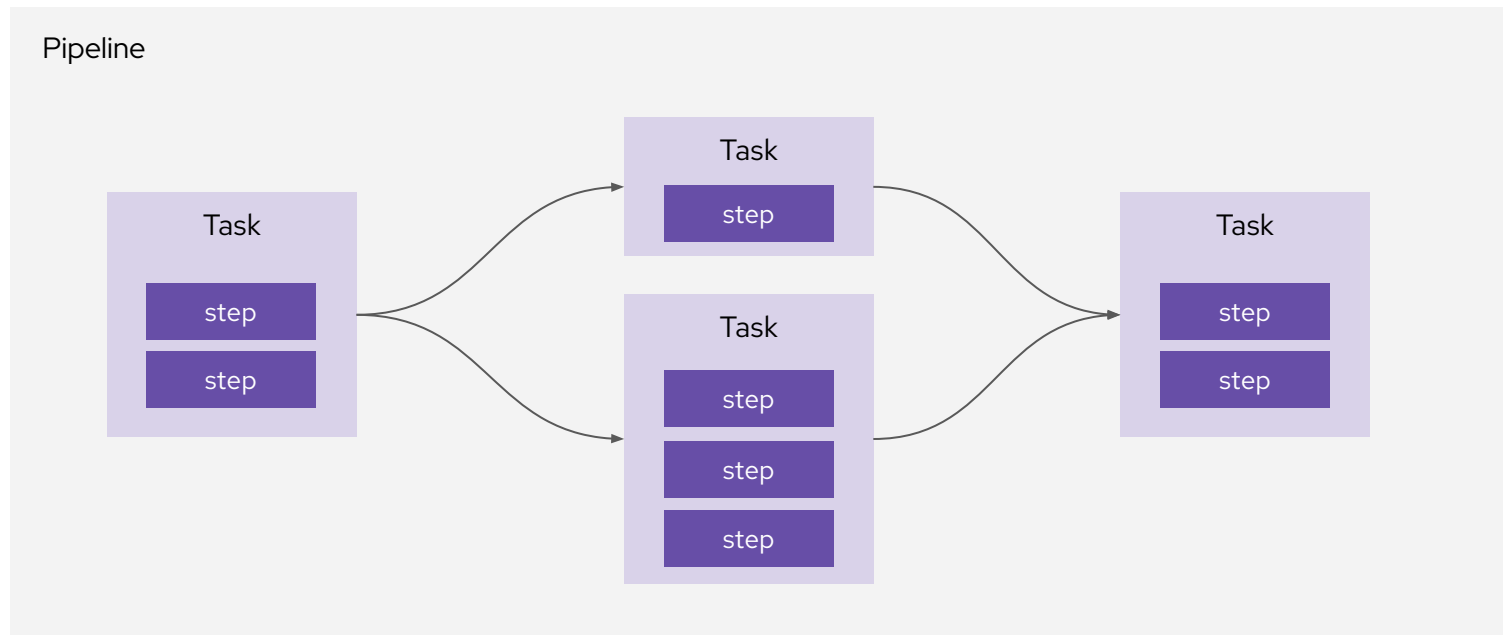


OpenShift Pipelines

- 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



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

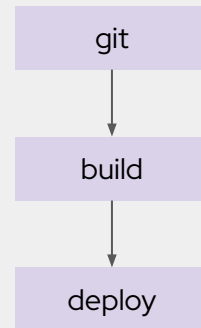
The screenshot displays the Tekton Hub (BETA) web interface. At the top, there's a dark header with the Tekton Hub logo and a 'Login' button. Below the header, a large banner reads 'Welcome to Tekton Hub' with the subtitle 'Discover, search and share reusable Tasks and Pipelines'. The main content area features a search bar and a 'Sort' dropdown set to 'Name'. On the left, a 'Refine By' sidebar allows filtering by 'Kind' (Task or Pipeline), 'Support Tier' (Official, Verified, Community), and 'Categories' (Build Tools, CLI, Cloud, Deploy, Image Build, Notification, Others, Test Framework). The main grid shows eight task cards, each with a title, version, description, update date, and a 'CLI' button. The tasks are: Ansible Runner v0.1, ansible tower cli v0.1, argocd v0.1, aws cli v0.1, Amazon ECR Login v0.1, azure cli v0.1, bentoml v0.1, and Python Black v0.1.

| Task Name | Version | Rating | Update Date | CLI Button |
|-------------------|---------|--------|----------------------|------------------|
| Ansible Runner | v0.1 | 4.5 | Updated 3 weeks ago | cli |
| ansible tower cli | v0.1 | 2.0 | Updated 3 weeks ago | ansible cli |
| argocd | v0.1 | 3.0 | Updated 3 weeks ago | deploy |
| aws cli | v0.1 | 5.0 | Updated 3 weeks ago | cli |
| Amazon ECR Login | v0.1 | 4.0 | Updated 3 weeks ago | aws ecr |
| azure cli | v0.1 | 1.0 | Updated 4 months ago | cli |
| bentoml | v0.1 | 0.0 | Updated 3 weeks ago | cli |
| Python Black | v0.1 | 0.0 | Updated 3 weeks ago | formatter python |

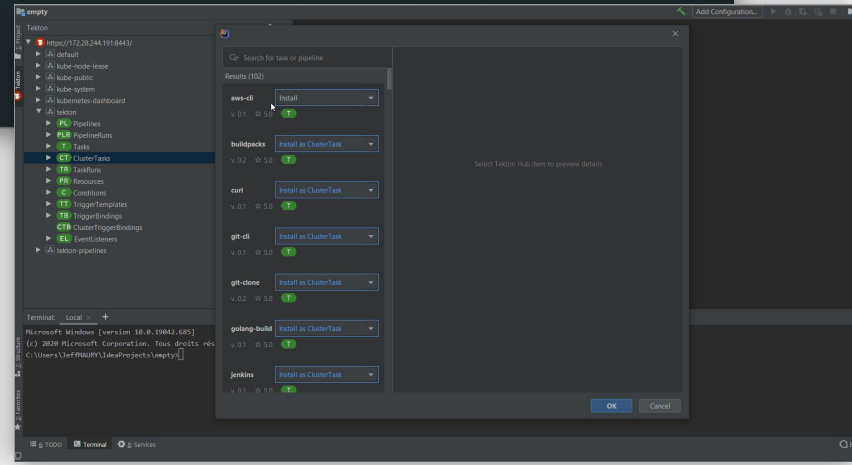
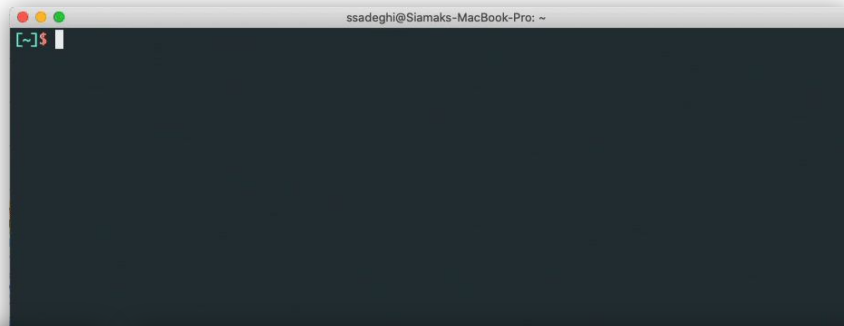
Tekton Concepts: Pipeline

- 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"]
```



30



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!

Environments are all manually configured!!!

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

Production deployments have a low success rate!

I can't easily rollback changes to a specific version

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

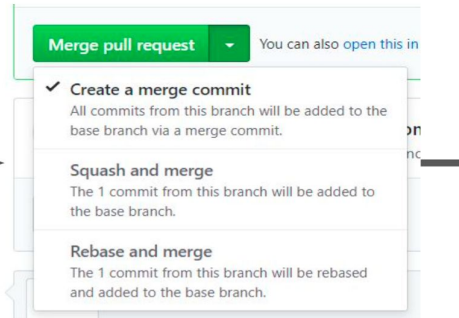
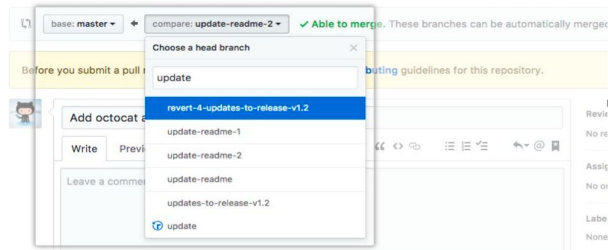
- OpenShift is a declarative environment
 - Cluster configuration is declared and Operators make it happen
 - Application deployments are declared and Kubernetes scheduler makes it happen
- GitOps in traditional environments requires automation/scripting, declarative environment minimizes or eliminates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

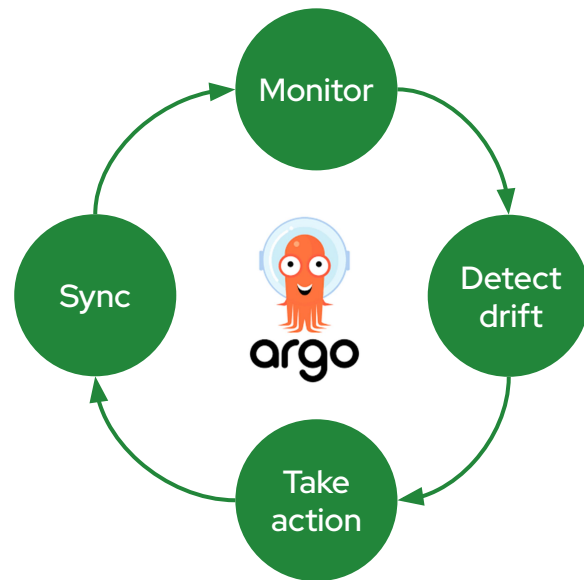


```
$ tkn pipelinerun logs update-from-master-run-g6s45

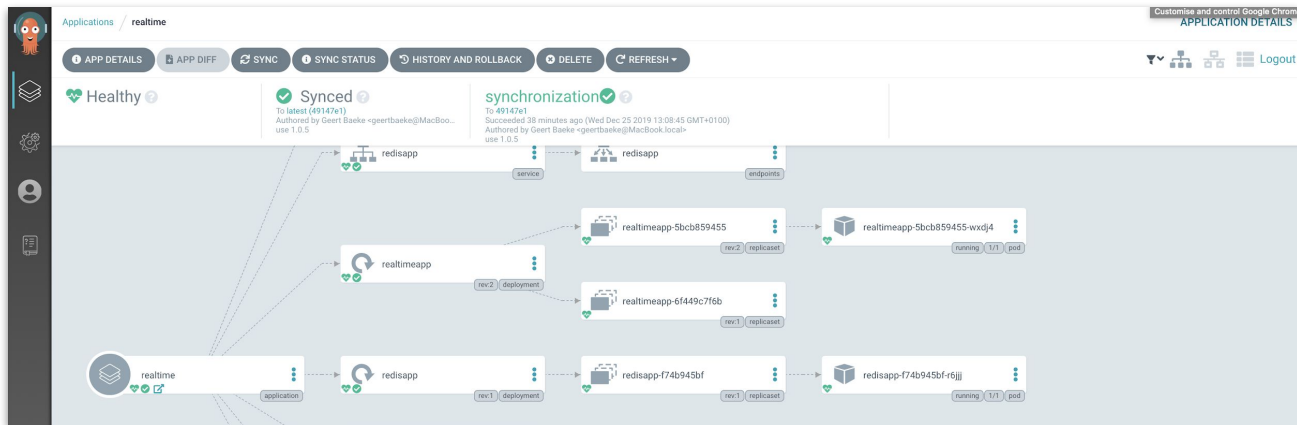
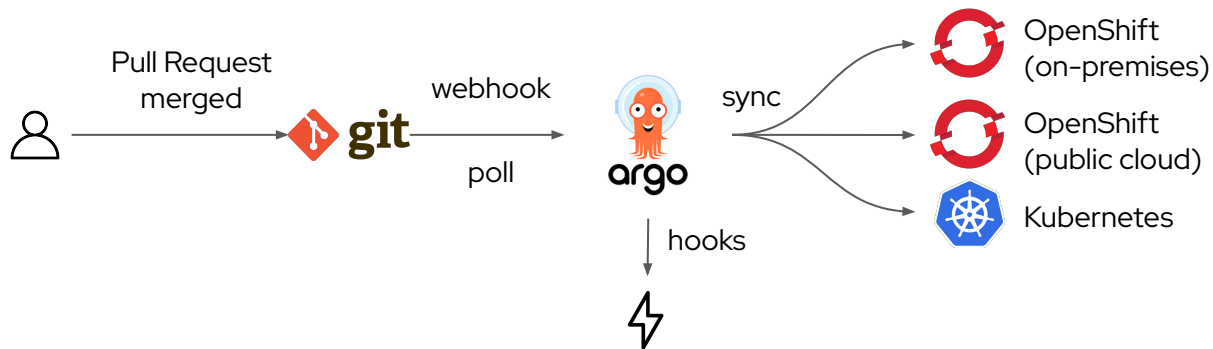
[run-kubectll] {"level":"info","ts":1580989837.3045664,"logger":"fallback-logger","caller":"logging/config.go:69","msg":"Fetch GitHub commit ID from kodata failed: \\\"KO_DATA_PATH\\\" does not exist or is empty"}
[run-kubectll] serviceaccount/demo-sa unchanged
[run-kubectll] clusterrolebinding.rbac.authorization.k8s.io/tekton-triggers-openshift-binding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

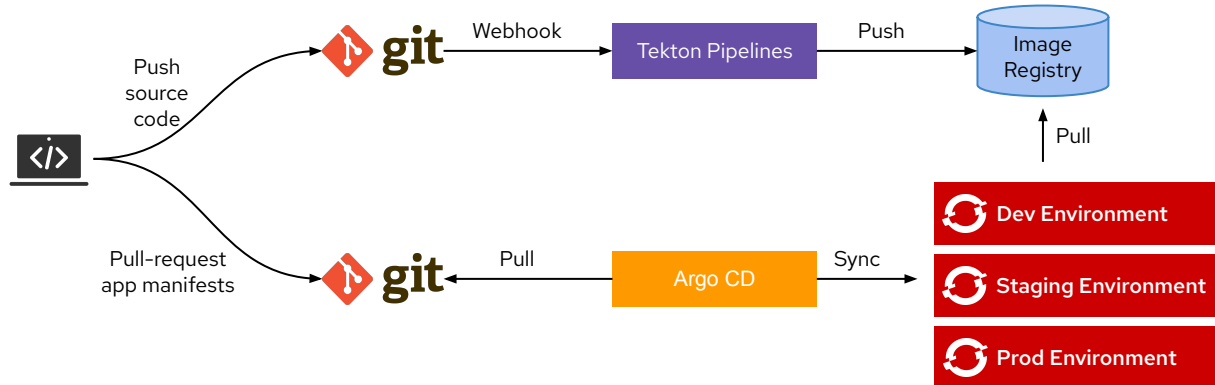


Argo CD



GitOps Application Manager CLI

```
$ kam bootstrap
```



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-test.git

deploy

pipelines

README.md

/taxi-config-prod.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
    │   └── volumes.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-kubect1-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

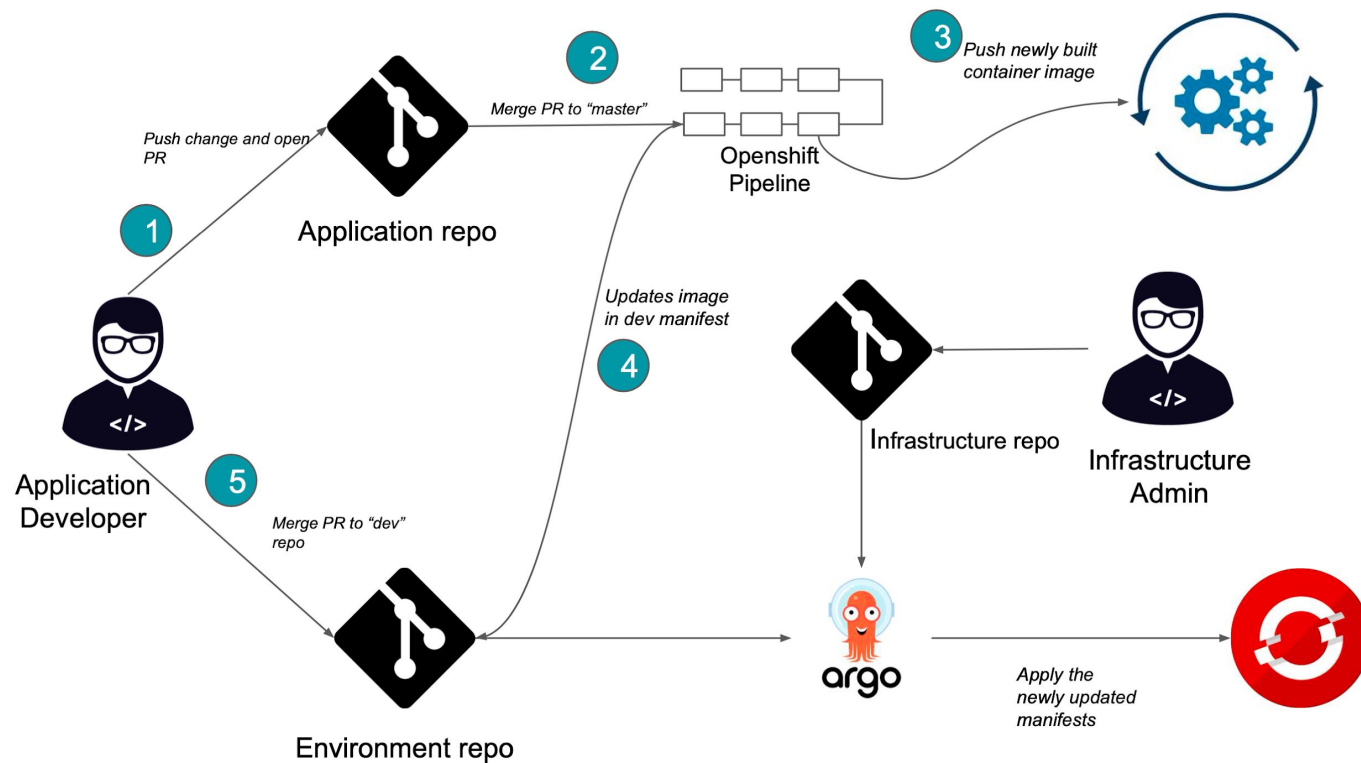
- There can be instances where you need to deploy something which cannot fully be done in a declarative way and it must be scripted
- Try minimizing this and leverage kubernetes primitives where possible
 - Init Containers
 - Jobs
 - Operators
- ArgoCD Resource Hooks
 - Hooks are ways to run scripts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



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

Thank you

 linkedin.com/company/red-hat

 facebook.com/redhatinc

 youtube.com/user/RedHatVideos

 twitter.com/RedHat