CI/CD for OpenShift

Sudharshan Govindan
Developer Advocate

sudharshan.govindan@in.ibm.com
@sudhargovindan

Contents

Overview of CI/CD

Continuous Integration

The process of having developers integrate code into a single code repository.

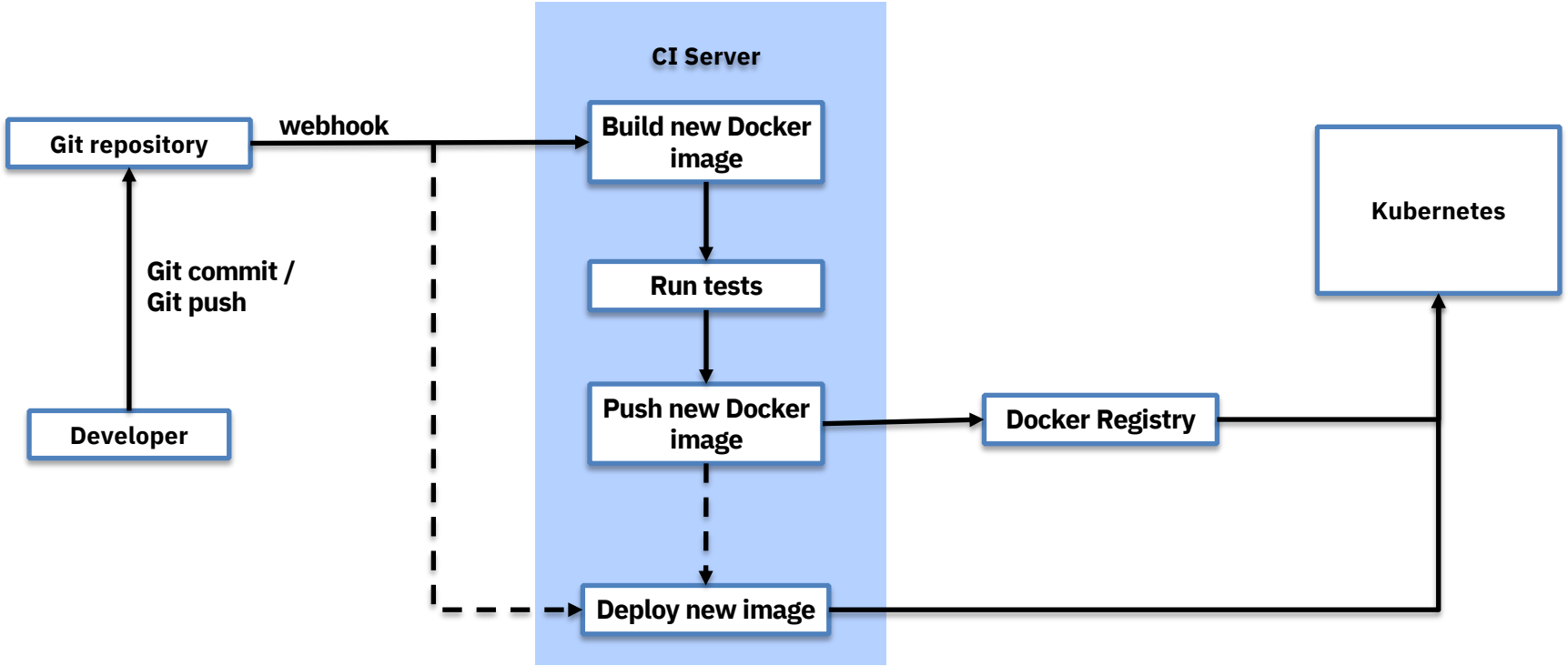
Continuous Delivery

The process of automatically building and testing new versions of software so that it is ready to be deployed to production. The process of deployment is manual.

Continuous Deployment

The process of automatically building and testing new versions of software so that it is ready to be deployed to production. The process of deployment is manual.

CI/CD architectural pattern for Kubernetes



CI/CD Pipelines in OpenShift

OpenShift Build Strategies

- **Source-to-Image (S2I)**, injects application source into a Docker image and assembles a new Docker image.

```
$ oc new-build --strategy source
```

- **Docker**, invokes the docker build command, expects a Dockerfile and all required artifacts

```
$ oc new-build --strategy docker
```

- **Pipeline**, uses a Jenkins pipeline, internal or external

```
$ oc new-build --strategy pipeline
```

- **Image Streams**, one or more Docker images identified by tags, OpenShift watches an image stream to receive notifications when new images are added.

- **Tekton**, cloud-native Kubernetes-style pipelines without CI server, plugins and config management

OpenShift Sources for Build Input

- Git
- Dockerfile
- Binary
- Image
- Input secrets
- External artifacts

Tool stack for Implementing CI/CD in OpenShift

- **Jenkins** – Deployed in OpenShift or running externally
- **SCM** – With Jenkins SCM integration OpenShift is notified of new commits for different branches. (Jenkins has plugins for AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase and RTC)
- **Docker** – Container images are build and published to a Docker registry.
- **Red Hat OpenShift**

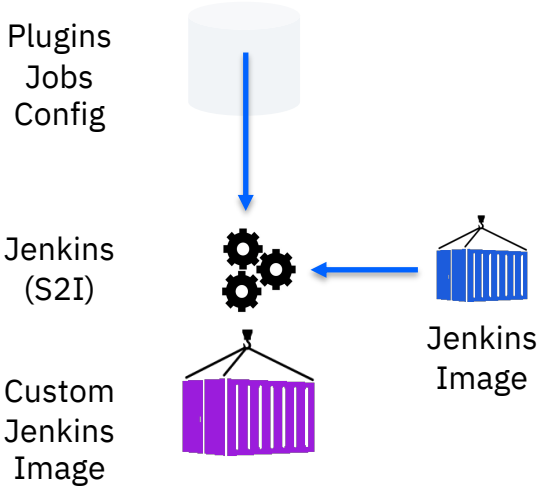
Jenkins in OpenShift

Jenkins in OpenShift

- Open source, server-based tool to build, test and deploy software continuously
- Enables continuous integration and continuous delivery
- Extendable via plugins
- Large ecosystem of existing plugins to integrate with on-prem and cloud based DevOps tooling, e.g. SCM integration, Kubernetes, Docker, Gerrit etc

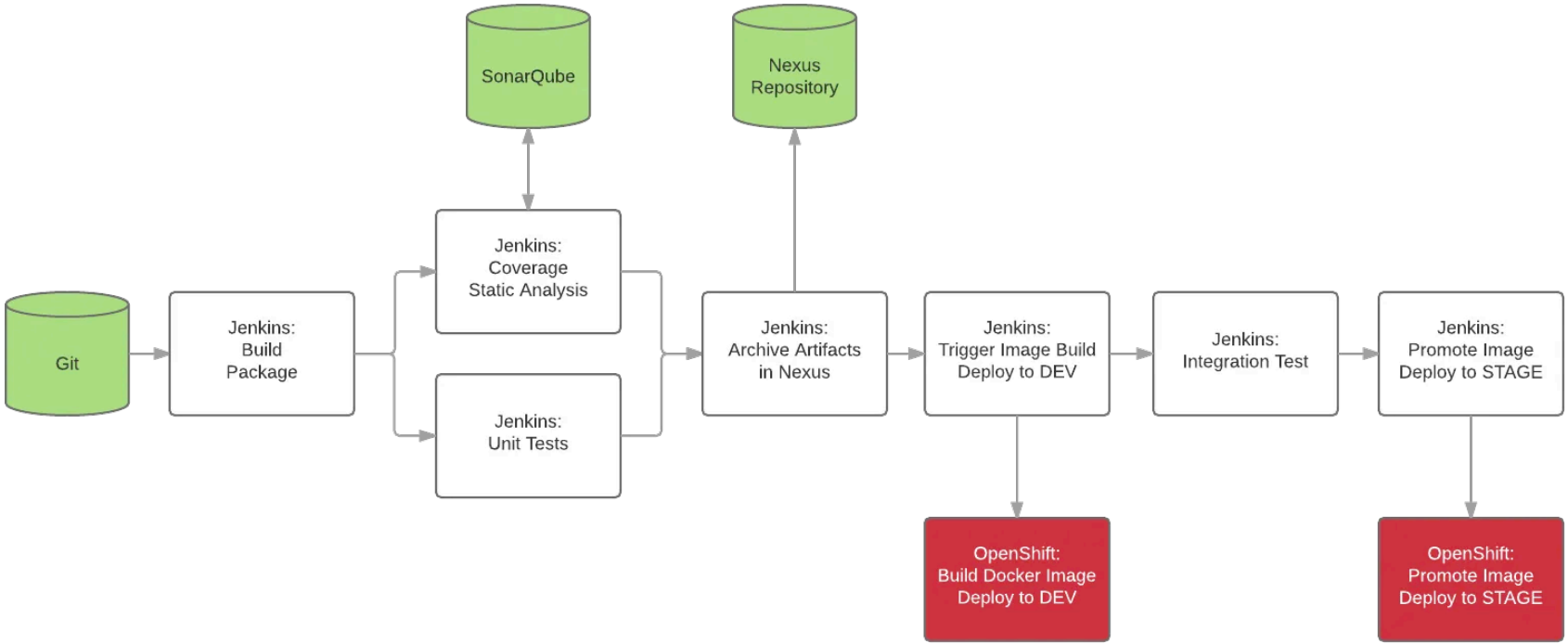
Jenkins in OpenShift

- Complete containerized CI/CD infrastructure
- Provides certified Jenkins containers with pre-configured plugins,
- The provided Jenkins docker image supports Source-to-Image (S2I) builds for customizing the Jenkins Docker image with:
  - Custom Jenkins Jobs definitions
  - Global configuration
  - Custom *config.xml* file
  - Additional Plugins
- OpenShift specific plugins to integrate authentication and CI/CD pipelines
- Scales pipeline execution through on-demand provisioning of Jenkins slaves in containers using the Kubernetes Jenkins plugin
- Can run many jobs in parallel



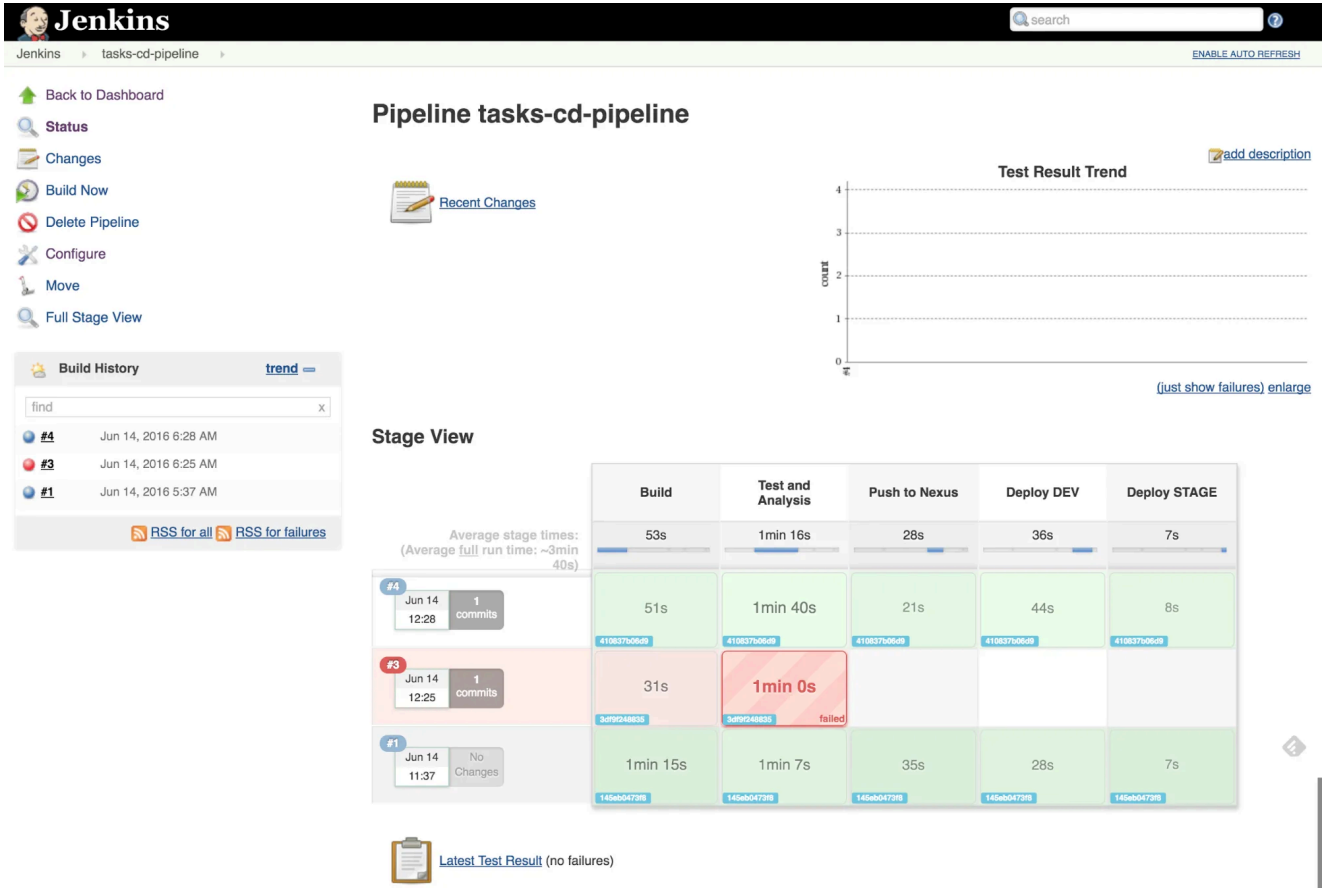
Jenkins in OpenShift

Example pipeline with Jenkins in OpenShift



Source: https://i1.wp.com/blog.openshift.com/wp-content/uploads/image02-16.png?ssl=1

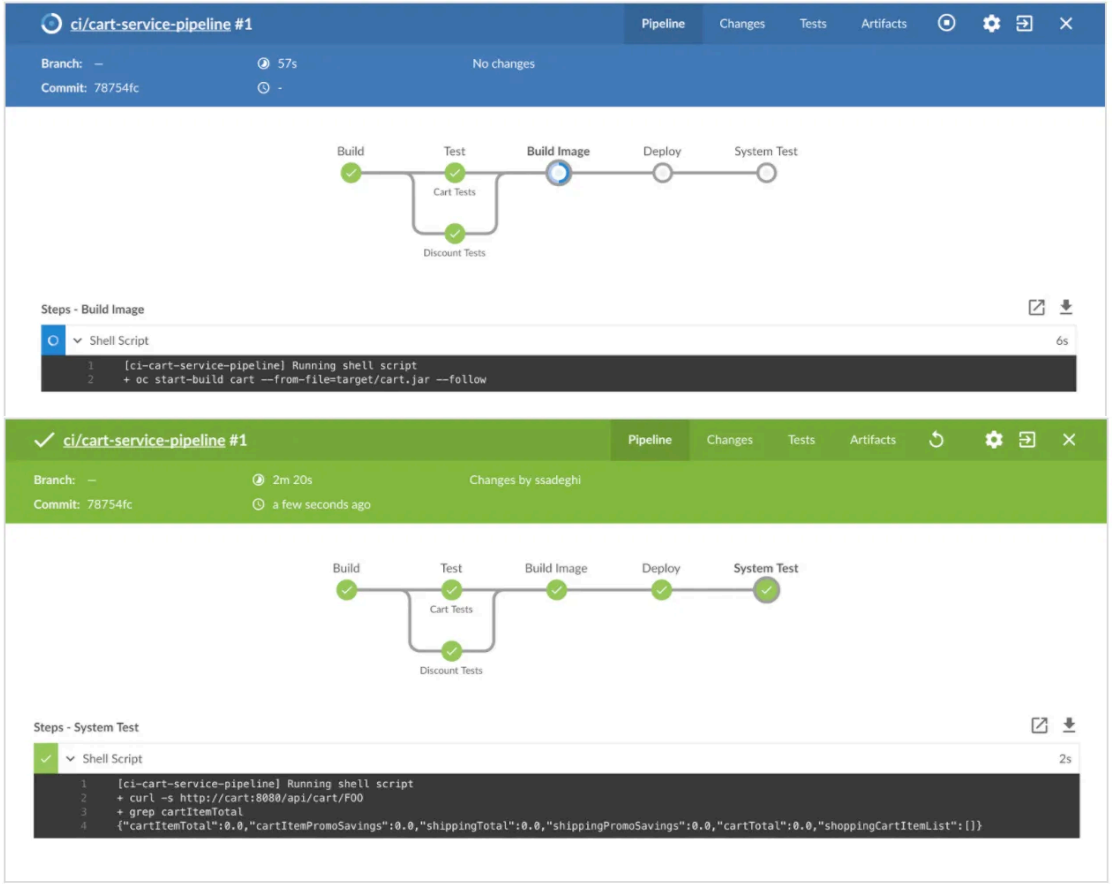
Jenkins in OpenShift



Source: https://i1.wp.com/blog.openshift.com/wp-content/uploads/image02-16.png?ssl=1

Jenkins in OpenShift

Support for Blue Ocean



Source: https://blog.openshift.com/openshift-pipelines-jenkins-blue-ocean/

Jenkins external to OpenShift

Jenkins external to OpenShift

If a build server already exists external to OpenShift, you can still interact with OpenShift to deploy to a cluster.

There are 3 methods to accomplish this:
https://blog.openshift.com/using-openshift-pipeline-plugin-external-jenkins/

Install **oc** on the external Jenkins server and use the **oc** binary to deploy apps to OpenShift

1

Use REST APIs to call OpenShift

2

Use the OpenShift Jenkins plugin to deploy apps to OpenShift, does not require oc binary.

https://github.com/openshift/jenkins-plugin

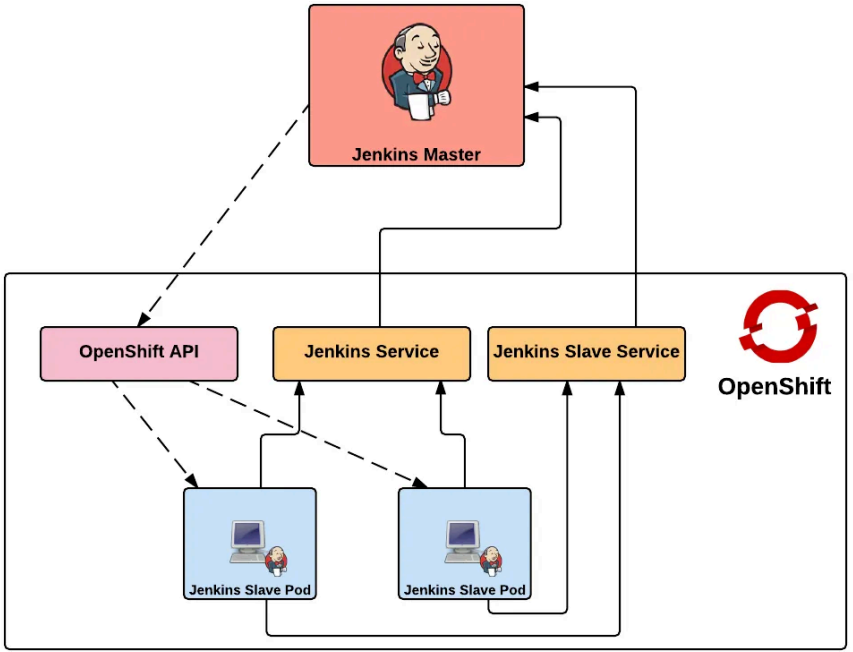
3

Jenkins external to OpenShift

When leveraging the Kubernetes plugin for Jenkins, Jenkins runs dynamic agents in a Kubernetes cluster to dynamically provision slave instances. There must be communication between the Jenkins master and OpenShift:

- Jenkins master communicates with the OpenShift API to manage the lifecycle of slave instances and take advantage of the elasticity OpenShift provides.
- OpenShift and Jenkins communicate via the port exposed by the web console
- Jenkins master communicates via a TCP port used for the JNLP slaves

https://github.com/jenkinsci/kubernetes-plugin



Source: https://blog.openshift.com/jenkins-slaves-in-openshift-using-an-external-jenkins-environment/

Tekton for OpenShift

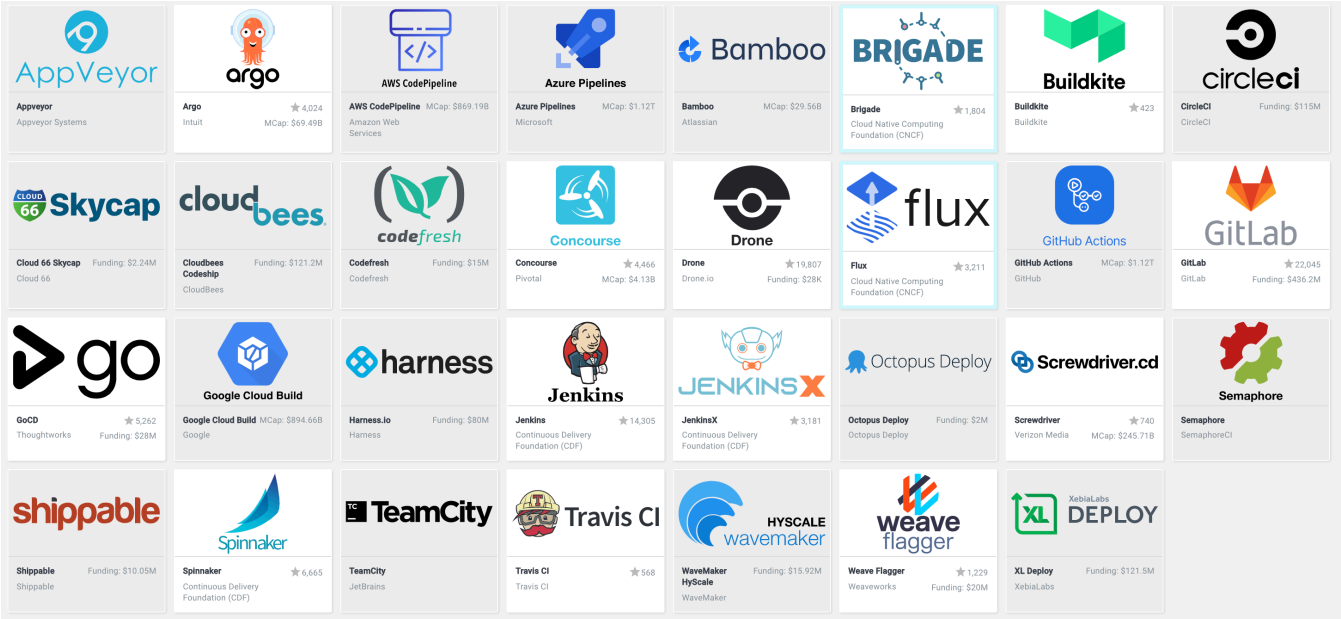
“Tekton is a set of shared, open source components for building CI/CD systems…[whose] goal is to provide industry specifications for CI/CD pipelines, workflows and other building blocks through a vendor neutral, open source foundation”

- CD Foundation

https://cd.foundation/projects/

Tekton provides Kubernetes-style resources for declaring CI/CD concepts

Industry Challenges

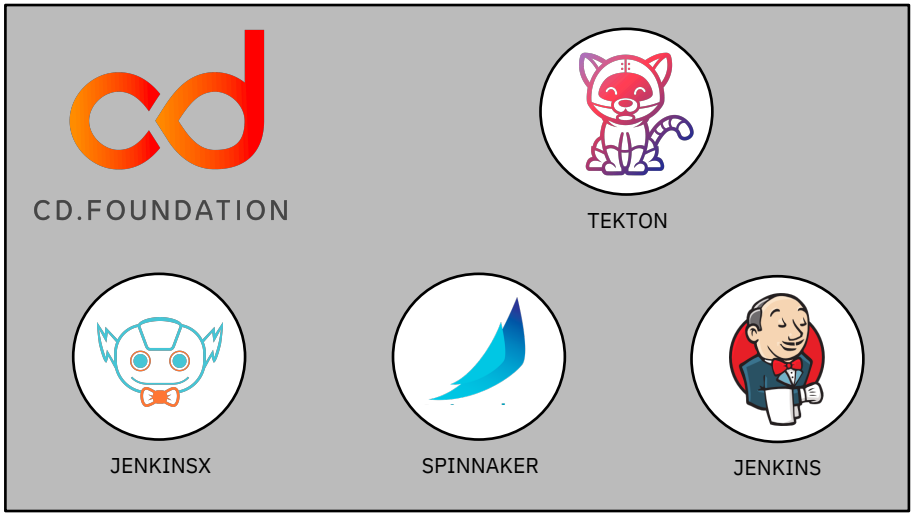


- Competing projects
- Conflicting Terminology
- Achieving the same goal

Image from CNCF Landscape: https://landscape.cncf.io/

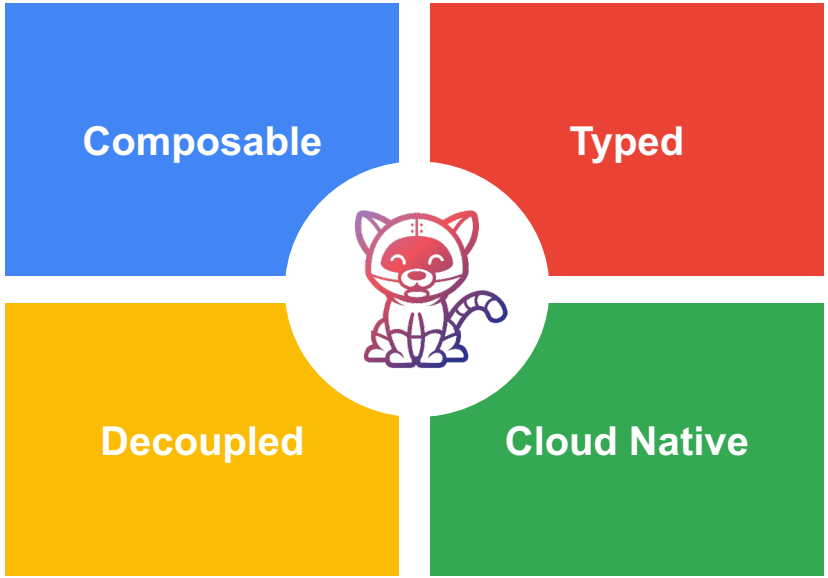
Tekton Emerges

- Spun out of the Knative build project
- Git Repo: https://github.com/tektoncd
- **Current release**: 0.15.2 (August 2020)
- **First release**: 0.2.0 (March 2019)
- **Contributors**: Google, Red Hat, Pivotal, IBM, etc.
- Part of the **CD Foundation** (under the Linux Foundation)
  - Includes other open source projects such as: JenkinsX, Jenkins, Spinnaker, Tekton
- CD Foundation announced in March 2019
  - **Goal**: To serve as the vendor-neutral home for the most important open source projects for continuous delivery



So, what is Tekton?

- **Cloud Native**: Run on Kubernetes, has Kubernetes clusters as a first-class type, use containers as their building blocks
- **Typed**: The concept of typed resources means that for a resources can swap out implementations
- **Composable**: Tekton concepts build upon each other
- **Decoupled**: The Tasks which make up a Pipeline can easily be run in isolation. One Pipeline can be used to deploy to any k8s cluster

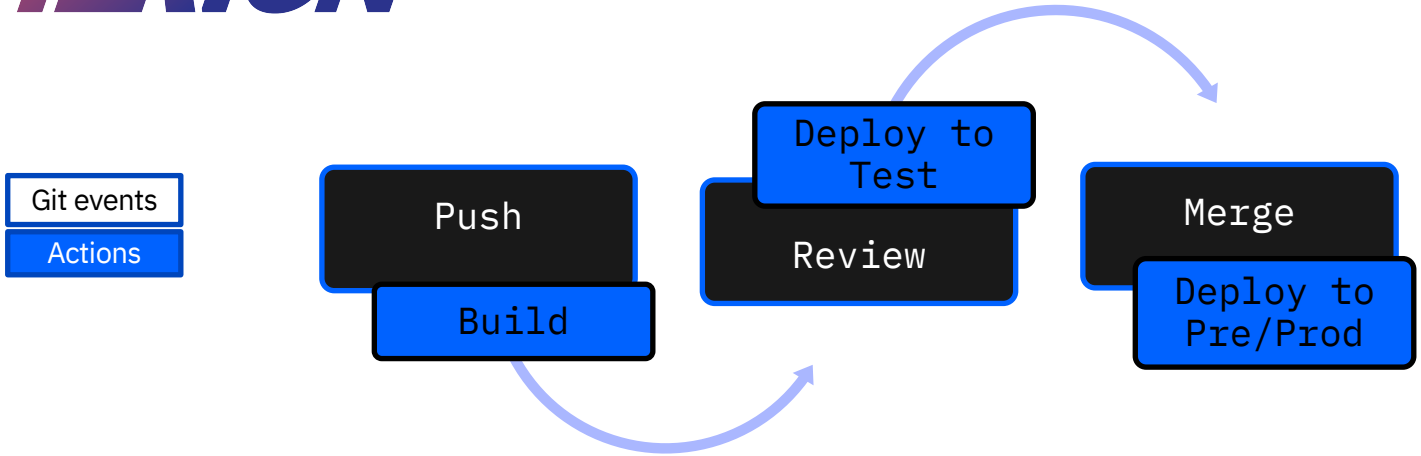


Tekton Project Goals

Goals

- Specify the “API” and provide the “building blocks” for running build pipelines
- Host a community of sub-projects that extend Tekton (Dashboard UI, CLI, Webhooks, etc.)
- Provide a catalog of best practices for authoring pipelines and tasks

TEKTON



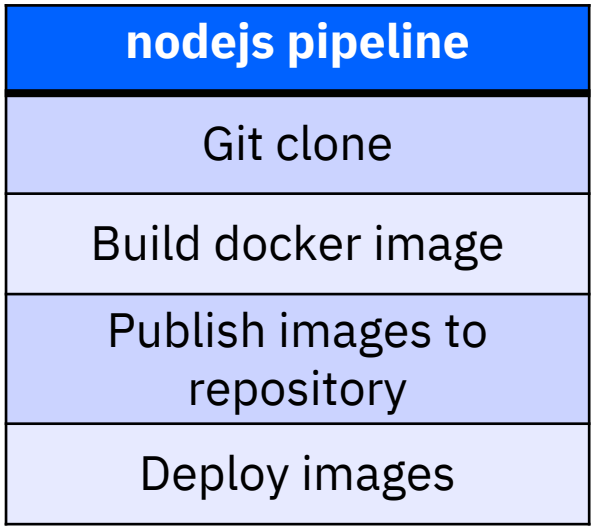
Pipelines

Task 1 → Task 2 → ... → Task N

Types of Tasks:

- Appsody build
- Container vulnerability scan
- Container signature
- Container publish (Tagging)
- Application scan
- Acceptance test harness
- Appsody deploy (operator-based)
- Razee pipeline promotion / cross cluster deploy

A Simple Tekton Pipeline



Pipeline – nodejs-pipeline

```
apiVersion:
tekton.dev/v1alpha1
kind: Pipeline
metadata:
  name: nodejs-pipeline
spec:
  resources:
  - name: git-source
    type: git
  - name: docker-image
    type: image
```

```
tasks:
  - name: build-task
    taskRef:
      name: nodejs-build-task
    resources:
      inputs:
      - name: git-source
        resource: git-source
      outputs:
      - name: docker-image
        resource: docker-image
  - name: deploy-task
    runAfter: [build-task]
    taskRef:
      name: nodejs-deploy-task
    resources:
      inputs:
      - name: git-source
        resource: git-source
      - name: docker-image
        resource: docker-image
```

Tekton Concept: Step

- The smallest building block
- Specify images, commands, arguments
- Is a container

```yaml
steps:
  - name: echo
    image: ubuntu
    command:
      - echo
    args:
      - "hello world"
```

Tekton CRD: Task

- New CRD
- Sequence of **Steps**
- Run in sequential order
- Reusable
- Perform a specific task
- Runs on the same k8s node

```yaml
apiVersion: tekton.dev/v1alpha1
kind: Task
metadata:
  name: echo-hello-world
spec:
  steps:
    - name: echo
      image: ubuntu
      command:
        - echo
      args:
        - "hello world"
```

Tekton CRD: Pipeline

- Expresses **Tasks**
  - Sequentially
  - Concurrently
- Links input and output
- Execute **Tasks** on different nodes

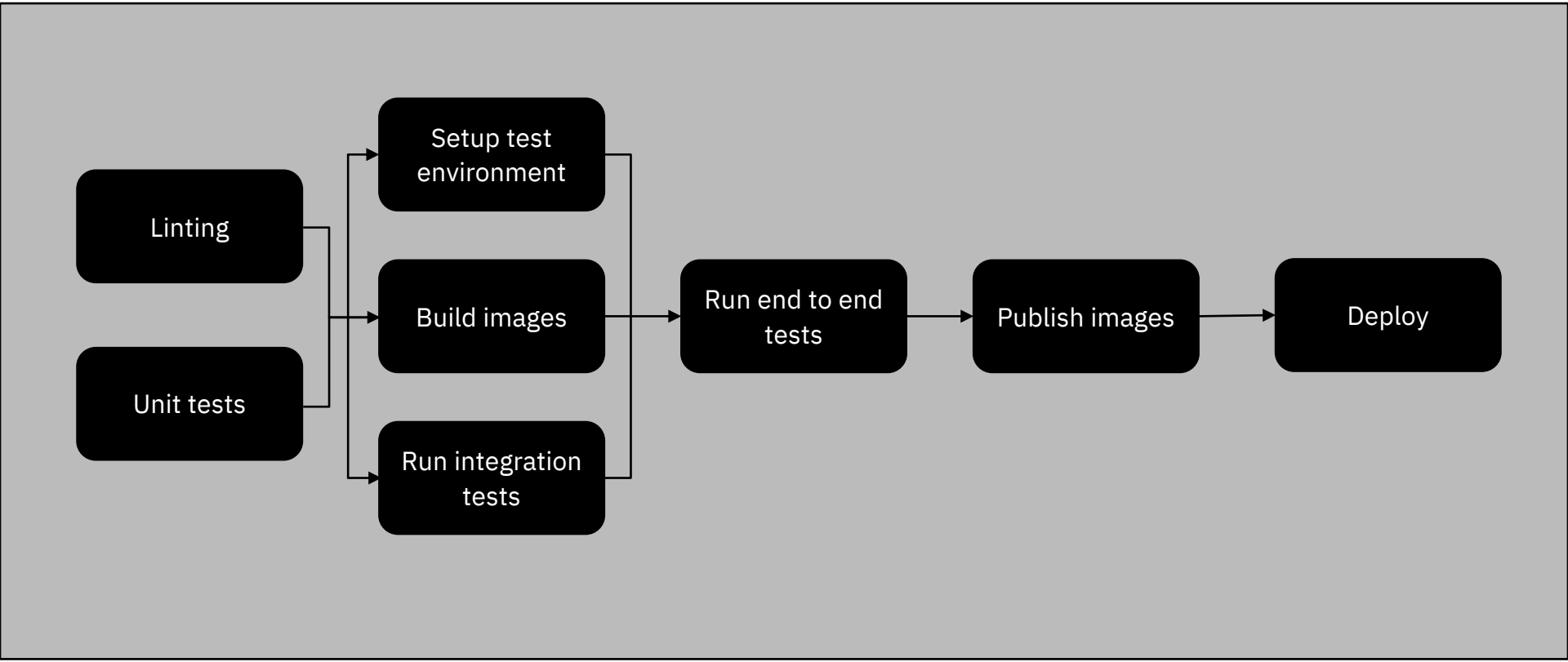
```yaml
apiVersion: tekton.dev/v1alpha1
kind: Pipeline
metadata:
  name: tutorial-pipeline
spec:
  - name: build-app
    taskRef:
      name: build-push
    resources:
      outputs:
        - name: image
          resource: my-image
  - name: deploy-app
    taskRef:
      name: deploy-kubectl
    resources:
      inputs:
        - name: image
      resource: my-image
      from:
        - build-app
```

Tekton Runtime CRDs

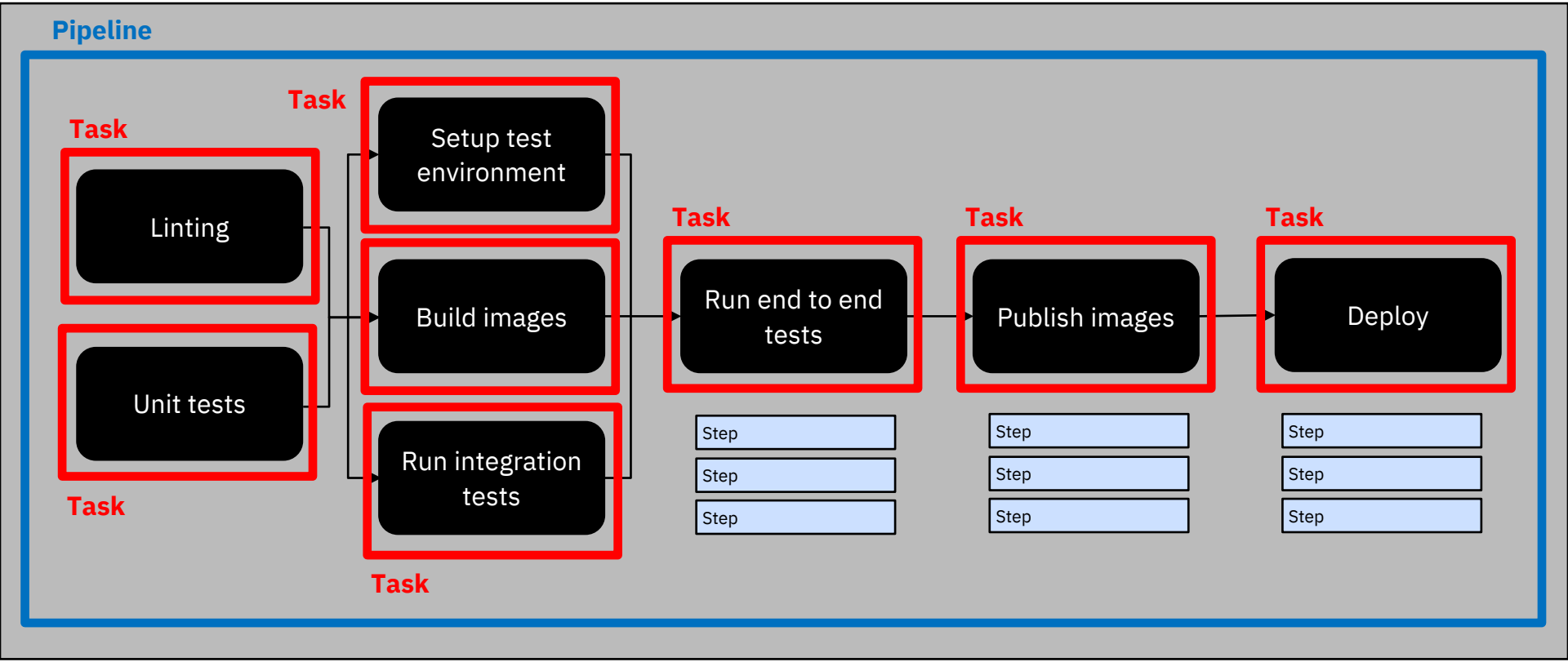
- Instances of Pipelines and Tasks:
  - PipelineRun
  - TaskRun
- Runtime info such as registry information and git repo

```yaml
apiVersion: tekton.dev/v1alpha1
kind: PipelineRun
metadata:
  name: tutorial-pipeline-run-1
spec:
  serviceAccountName: tutorial-service
  pipelineRef:
    name: tutorial-pipeline
  resources:
    - name: source-repo
      resourceRef:
        name: skaffold-git
    - name: web-image
      resourceRef:
        name: skaffold-image-leeroy-web
```

Putting it all together



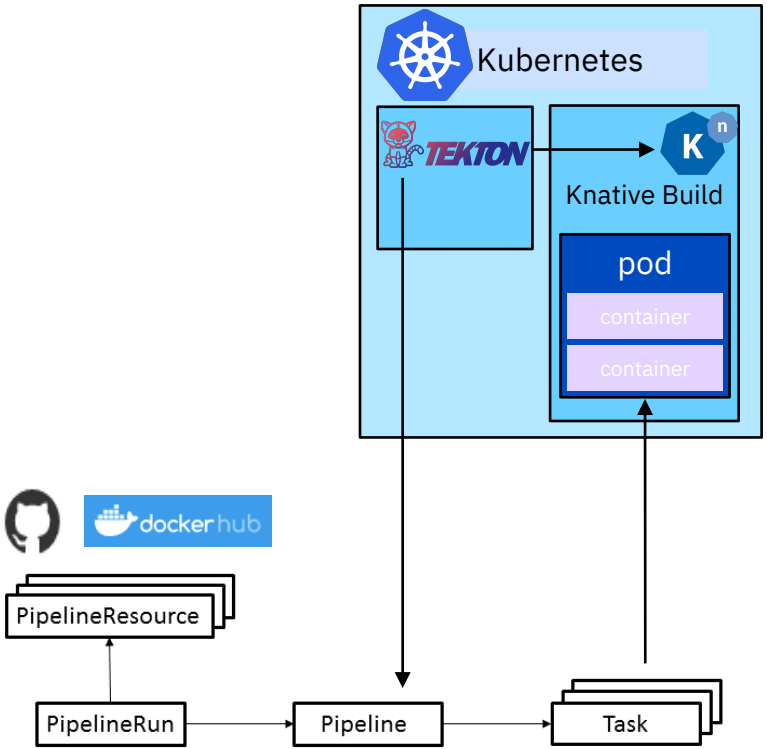
Putting it all together



OpenShift Pipelines

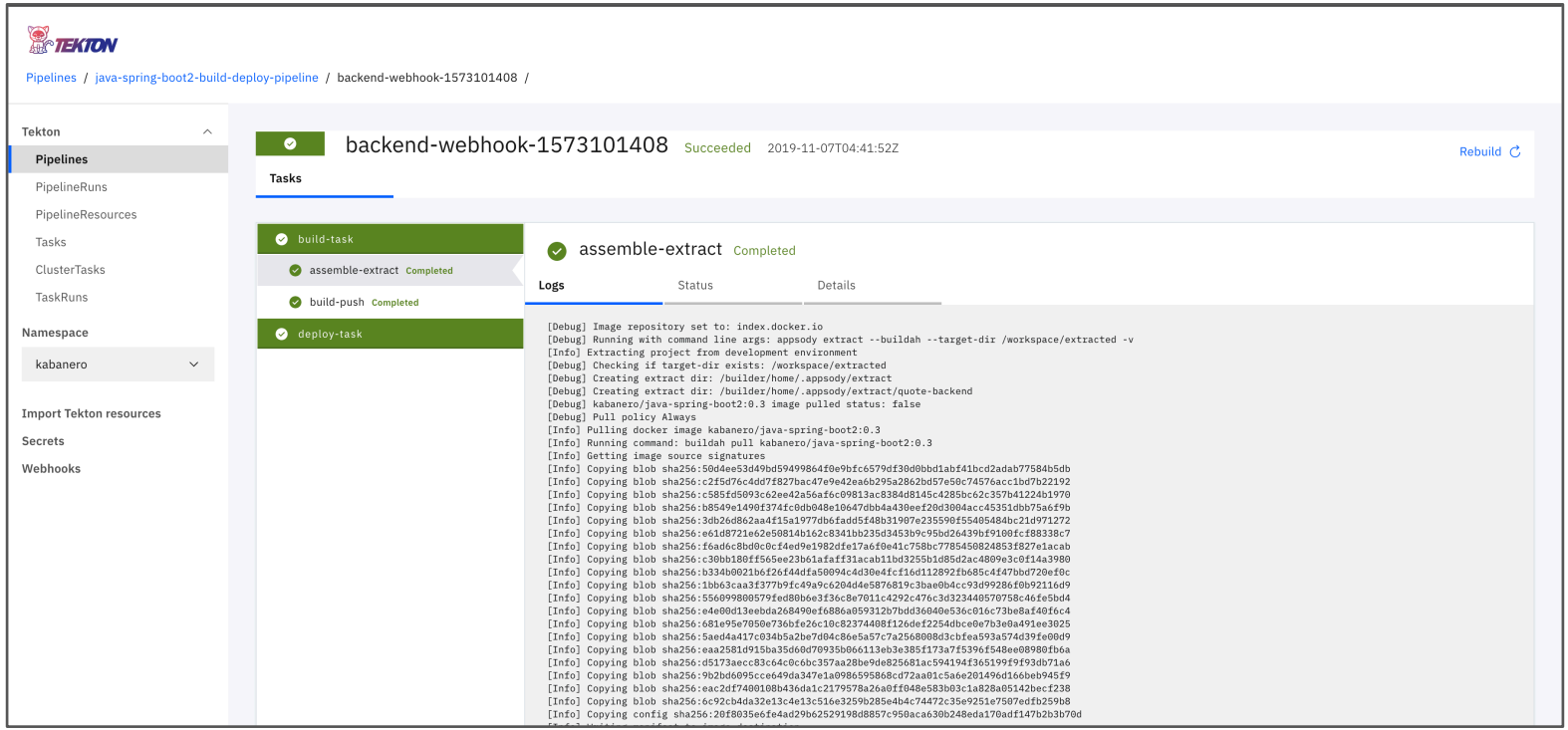
Custom Resource Definitions (CRD) needed to define a pipeline are:

- **Task**: a reusable, loosely coupled number of steps that perform a specific task (e.g., building a container image), each step runs in its own container
- **Pipeline**: the definition of the pipeline and the Tasks that it should perform
- **PipelineResource**: inputs (e.g., git repository) and outputs (e.g., image registry) to and out of a pipeline or task
- **TaskRun**: the result of running an instance of task
- **PipelineRun**: the result of running an instance of pipeline, which includes a number of TaskRuns



Tekton Features: Dashboard

https://github.com/tektoncd/dashboard



Output from a PipelineRun

Tekton Features: CLI

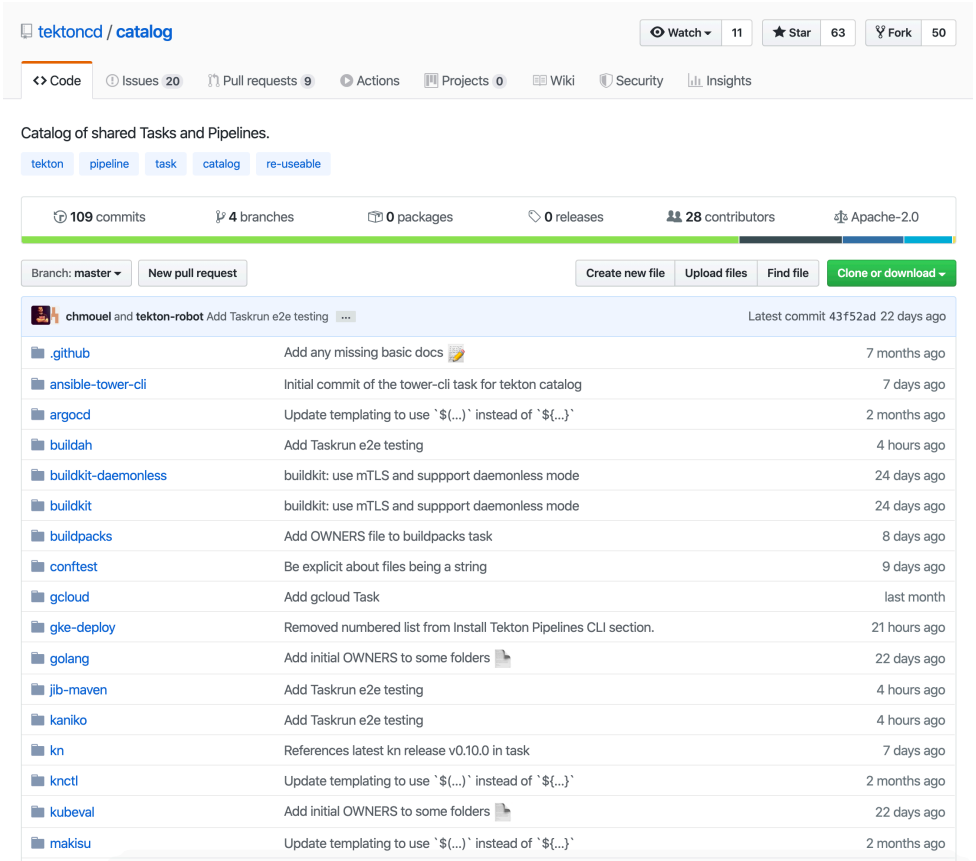
https://github.com/tektoncd/cli

```
$ tkn --help
CLI for tekton pipelines


Usage:
  tkn [command]


Available Commands:
  clustertask Manage clustertasks
  condition   Manage conditions
  pipeline    Manage pipelines
  pipelinerun Manage pipelineruns
  resource    Manage pipeline resources
  task        Manage tasks
  taskrun     Manage taskruns


Other Commands:
  completion  Prints shell completion scripts
  version     Prints version information
```

Tekton Features: Catalog

https://github.com/tektoncd/catalog

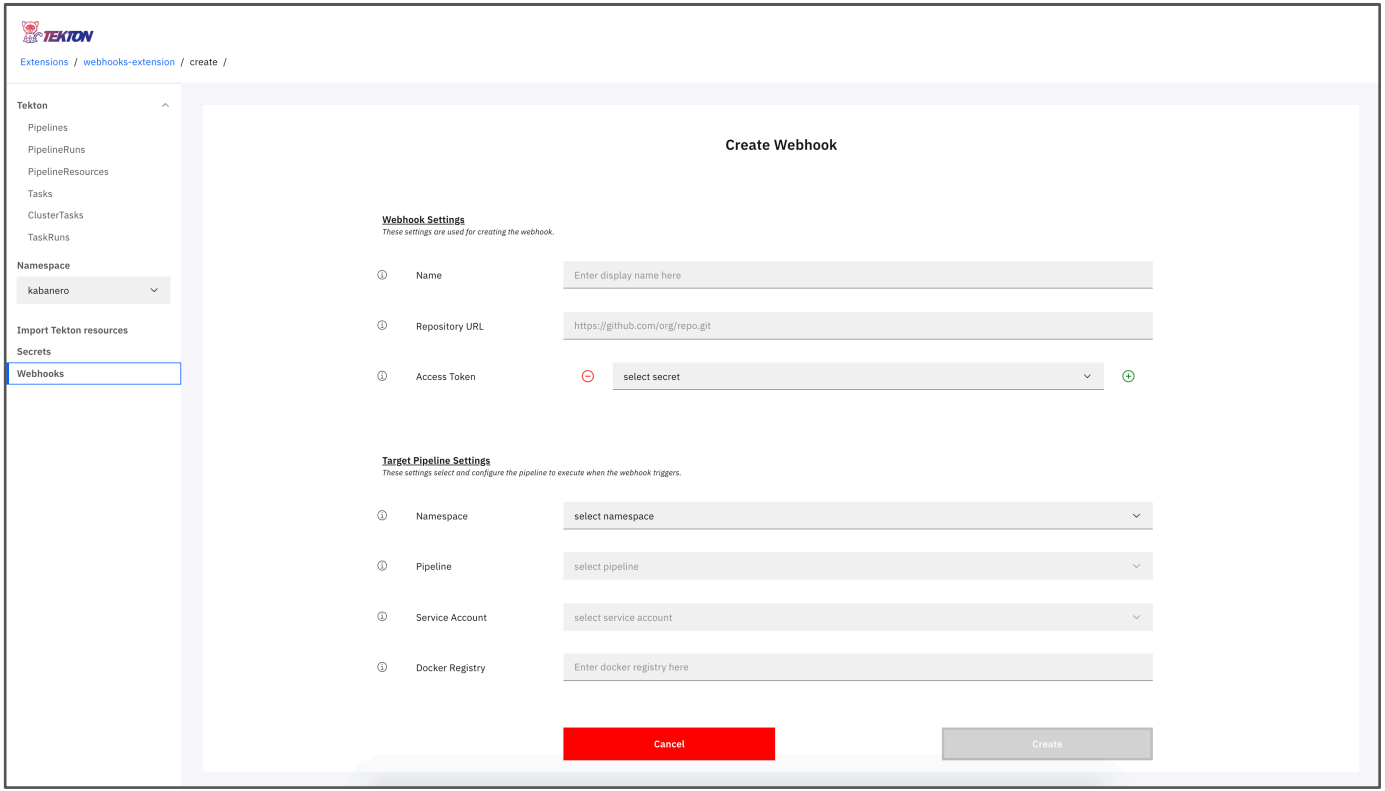
This repository contains a catalog of **Task** resources, designed to be reusable in many pipelines.



Tekton Features: Webhooks

https://github.com/tektoncd/experimental/tree/master/webhooks-extension

Allows users to set up GitHub webhooks that will trigger **PipelineRuns** and **TaskRuns**.



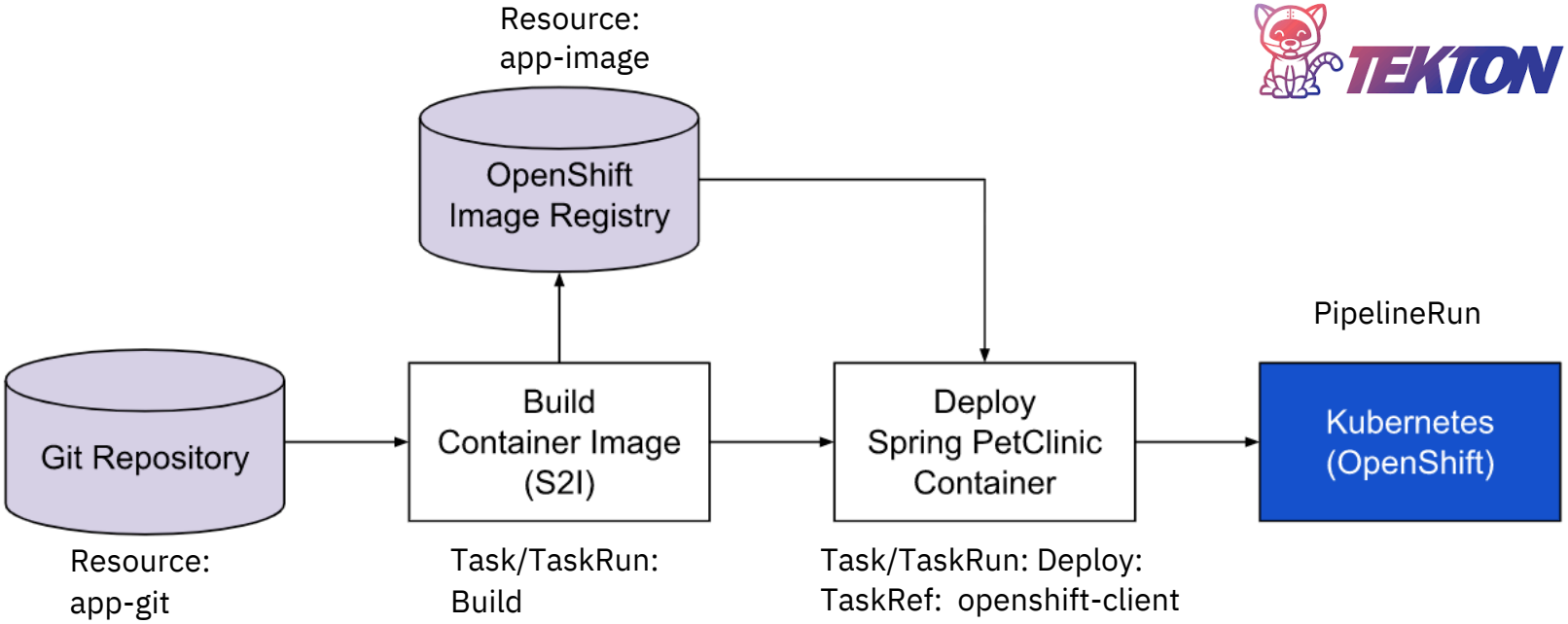
Tekton with Kabanero

https://github.com/kabanero-io/kabanero-pipelines

- Each Stack in Kabanero comes with a default pipeline (named <lang>-build-deploy).
- The pipeline consists of two tasks: One to build the appsody stack, the second to deploy the appsody stack

OpenShift Pipelines

Spring PetClinic Pipeline



Source: https://github.com/openshift/pipelines-tutorial

OpenShift Pipelines

```
apiVersion: tekton.dev/v1alpha1
kind: Pipeline
metadata:
  name: petclinic-deploy-pipeline
spec:
  resources:
  - name: app-git
    type: git
  - name: app-image
    type: image
  tasks:
  - name: build
    taskRef:
      name: s2i-java-8
    params:
      - name: TLSVERIFY
        value: "false"
    resources:
      inputs:
      - name: source
        resource: app-git
      outputs:
      - name: image
        resource: app-image
  - name: deploy
    taskRef:
      name: openshift-client
    runAfter:
      - build
    params:
    - name: ARGS
      value: "rollout latest spring-petclinic"
```

```
apiVersion: tekton.dev/v1alpha1
kind: Task
metadata:
  name: openshift-client
spec:
  inputs:
    params:
      - name: ARGS
        description: The OpenShift CLI arguments to run
        default: help
  steps:
    - name: oc
      image: quay.io/openshift-pipeline/openshift-cli:0.5.0
      command: ["/usr/local/bin/oc"]
      args:
        - "$(inputs.params.ARGS)"
```

CI/CD Pipelines for Microservices

CI/CD Pipelines for Microservices

Pipeline as Code

- Each codebase has its own CSM repo, with multiple repos, each may have a different build/test/deploy process

- Treat pipeline as another piece of code. Commit the Jenkinsfile to the SCM repo

Automation of CI/CD pipeline is necessary to ensure agile deployments of microservices

Zero downtime deployment is **handled by Kubernetes** as a rolling update. Updates can also be rolled back to a previous revision.

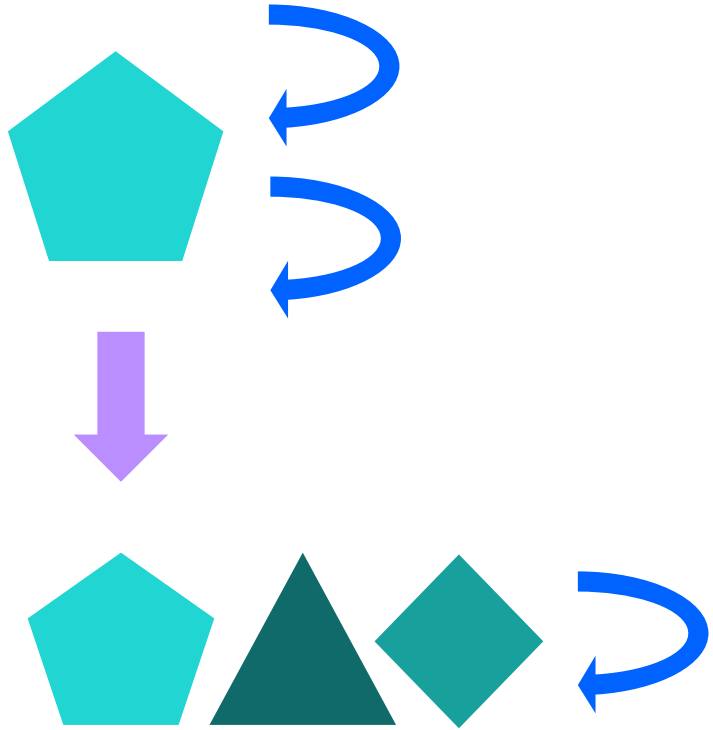
CI/CD Pipelines for Microservices

Test microservices separately.

- Unit tests can be done during the build stage with Maven along with a static code analysis

After publishing image to registry, integration testing begins with the other components.

- The newly published microservice is then tested against the latest stable versions of other components.



CI/CD Pipelines for Microservices

Best practices:

- Release often
- Blue green deployment.
- Canary deployment.
- A/B Testing.
- Commit daily, reduce branching.

https://12factor.net/

1. Codebase: each codebase has its own CSM repo
2. Dependencies: explicitly declare and isolate dependencies
3. Config: store config in the environment, not in the application
4. Backing services: treat backing services, local or remote, as attached but swappable resources
5. Build, release, run: Strictly separate build and run stages
6. Processes: execute the app as one or more stateless processes running in the environment
7. Port binding: export services via port binding, do not rely on runtime injection
8. Concurrency: scale out via the process model, so that the application is able to span multiple processes on multiple physical machines
9. Disposability: maximize robustness with fast startup and graceful shutdown, elastic scaling, and rapid deployment
10. Dev/prod parity:
    1. Small time gap: from code to deployment asap
    2. Small personnel gap: developer is involved in dev, ops and prod
    3. Small tools gap: keep dev, staging, and prod as similar as possible
11. Logs: Treat logs as event streams, use stdout
12. Admin processes: Run admin/management tasks as one-off processes w the same rules as above