

CI/CD for OpenShift

Sudharshan Govindan
Developer Advocate

sudharshan.govindan@in.ibm.com
@sudhargovindan

Contents

Overview of CI/CD	03
CI/CD Pipelines in OpenShift	05
Jenkins in OpenShift	08
Jenkins external to OpenShift	14
Tekton for OpenShift 4	17
CI/CD Pipelines for Microservices	21
References	25



Overview of CI/CD

Continuous Integration

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

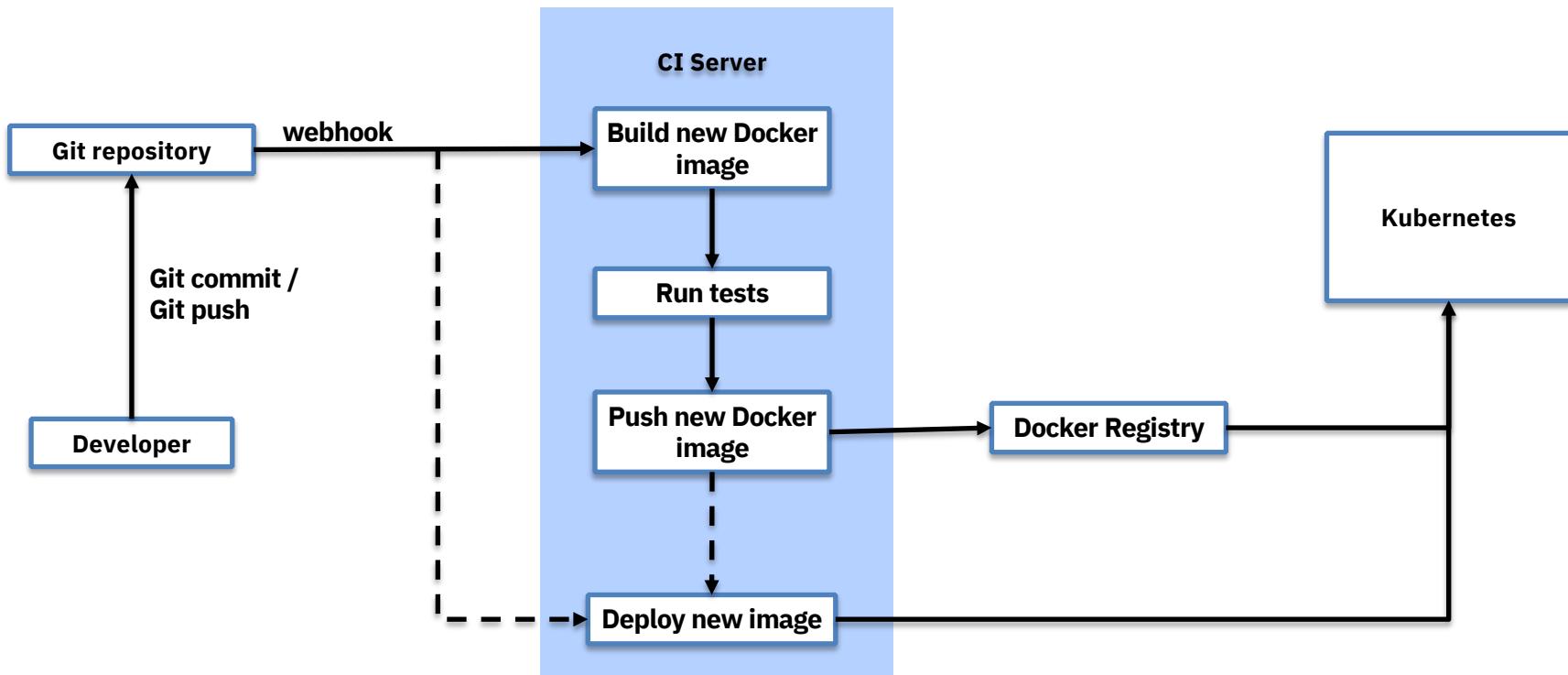
Continuous Delivery

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.
- **Docker**, invokes the docker build command, expects a Dockerfile and all required artifacts
- **Pipeline**, uses a Jenkins pipeline, internal or external
- **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

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

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

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

OpenShift Sources for Build Input

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

Tool stack for Implementing CI/CD in OpenShift



Jenkins



- **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

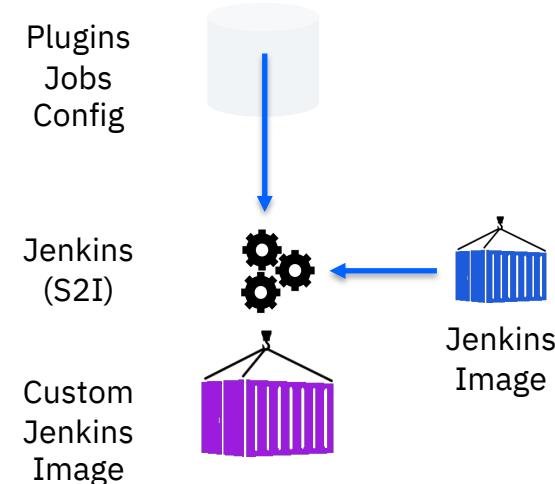
Jenkins in OpenShift



- 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

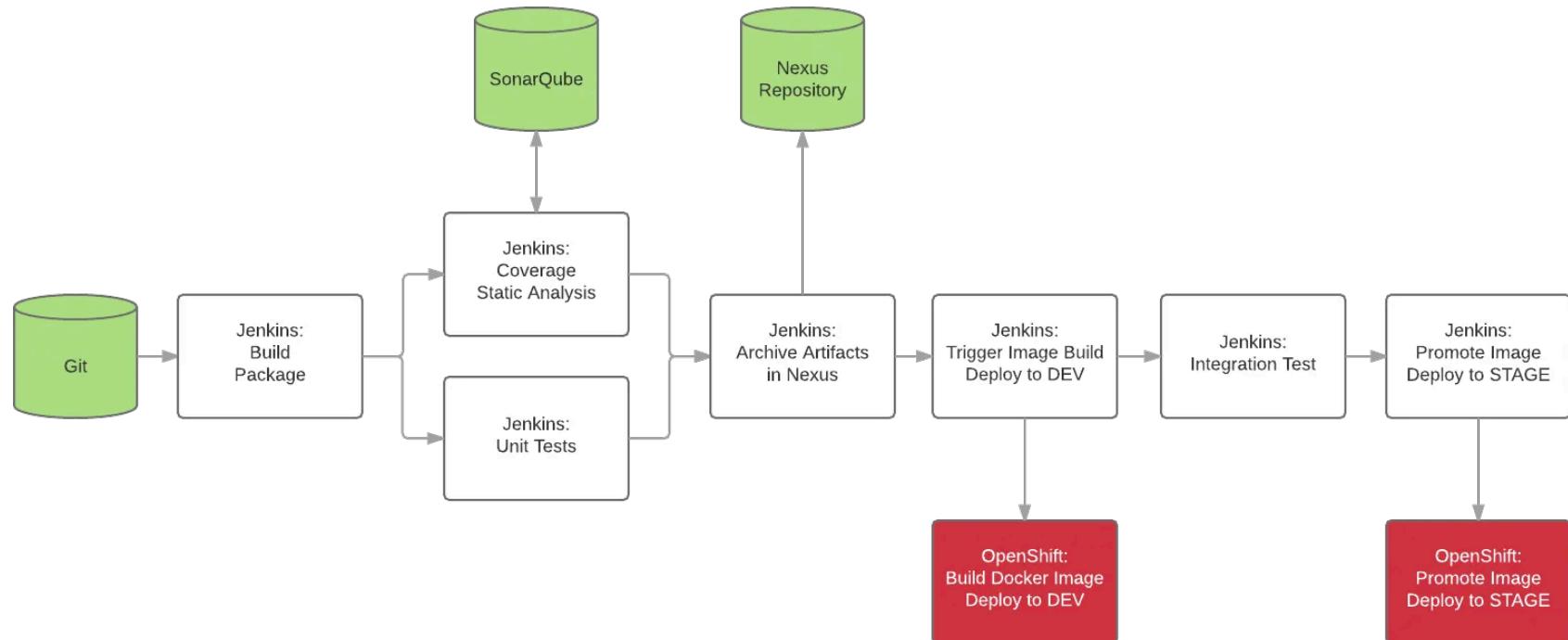
Jenkins in OpenShift

- 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

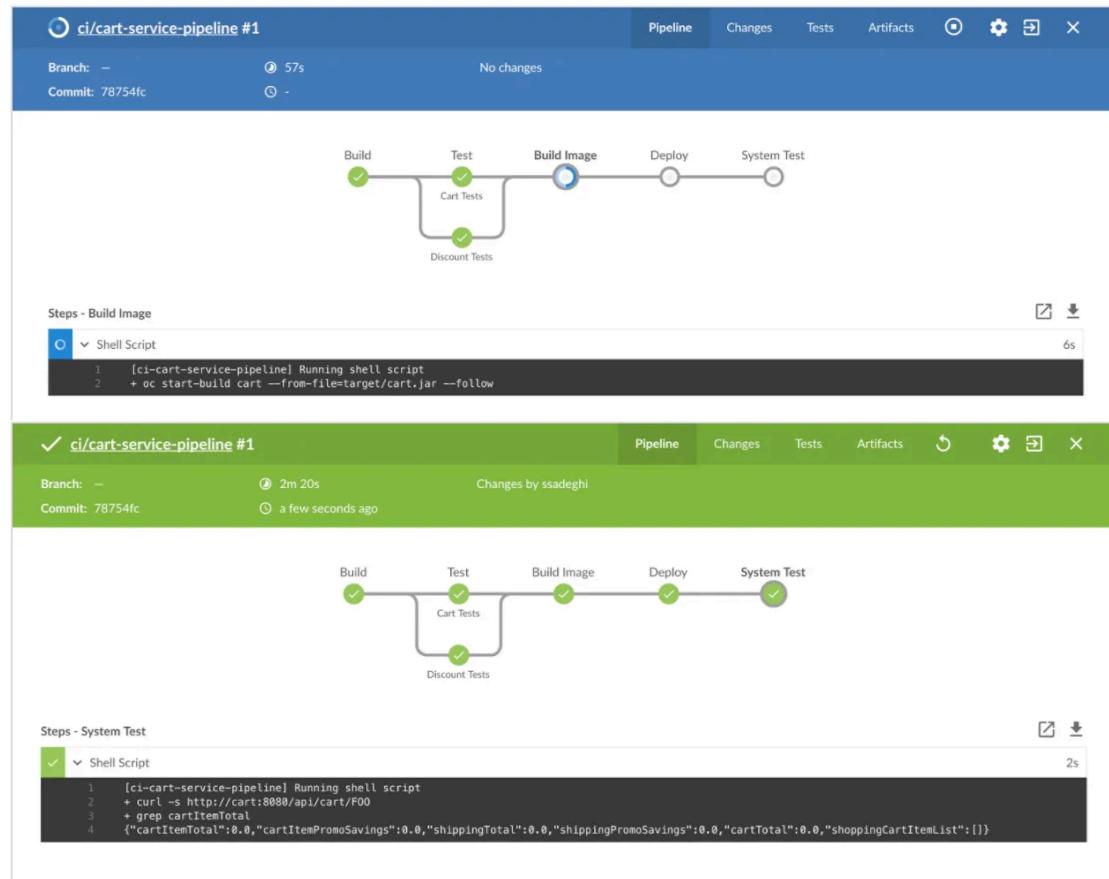
The screenshot displays the Jenkins Pipeline tasks-cd-pipeline interface. On the left, a sidebar menu includes: Back to Dashboard, Status, Changes, Build Now, Delete Pipeline, Configure, Move, and Full Stage View. The main area shows the following components:

- Pipeline tasks-cd-pipeline**: The title of the pipeline.
- Recent Changes**: A section showing recent modifications.
- Test Result Trend**: A chart showing the count of failures over time.
- Build History**: A table listing builds #4, #3, and #1 with their respective run times (Jun 14, 2016 6:28 AM, Jun 14, 2016 6:25 AM, Jun 14, 2016 5:37 AM).
- Stage View**: A grid showing the execution of five stages (Build, Test and Analysis, Push to Nexus, Deploy DEV, Deploy STAGE) across three builds. Build #4 shows a failure in the Test and Analysis stage (1min 40s). Build #3 shows a failure in the Test and Analysis stage (31s). Build #1 shows no changes.
- Latest Test Result**: A summary indicating no failures.

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

Jenkins in OpenShift

Support for 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>

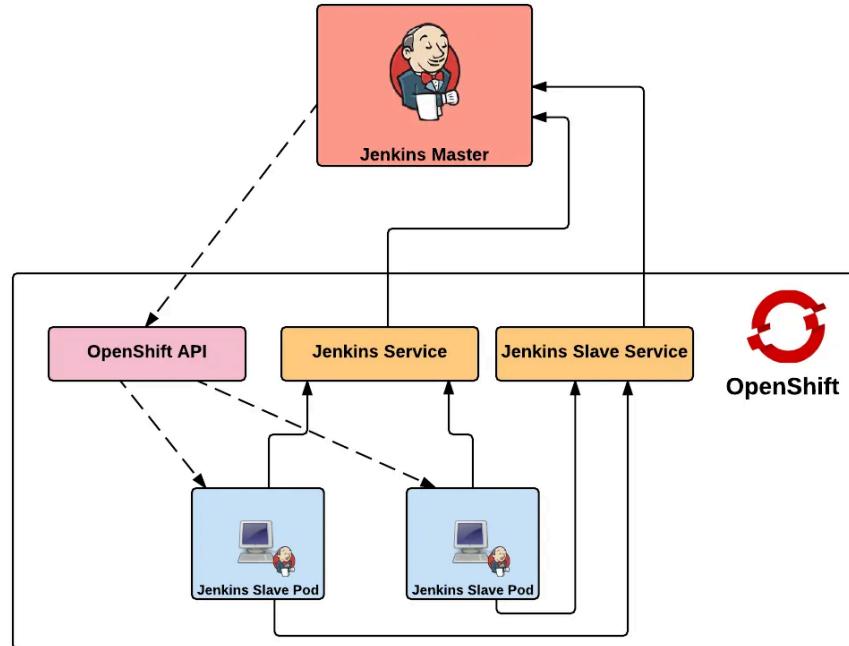
3

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>



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

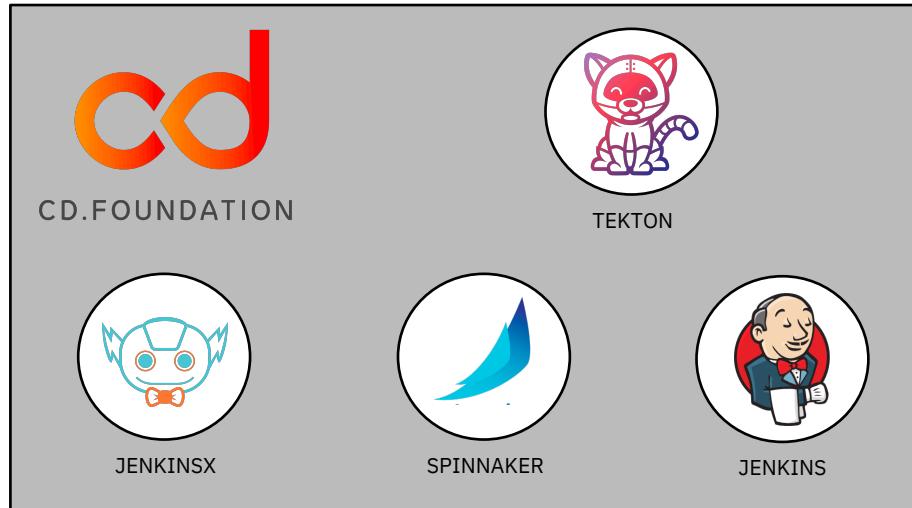
Industry Challenges

 AppVeyor Appveyor Systems	 Argo Intuit ★ 4,024 MCap: \$69.49B	 AWS CodePipeline Amazon Web Services MCap: \$869.19B	 Azure Pipelines Microsoft MCap: \$1.12T	 Bamboo Atlassian MCap: \$29.56B	 BRIGADE Cloud Native Computing Foundation (CNCF) ★ 1,804	 Buildkite Buildkite ★ 423	 circleci CircleCI Funding: \$115M
 Cloud 66 Skycap Cloud 66 Funding: \$2.24M	 cloudbees Cloudbees CodeShip Funding: \$121.2M	 codefresh Codefresh Funding: \$15M	 Concourse Pivotal ★ 4,466 MCap: \$4.13B	 Drone Drone.io ★ 19,807 Funding: \$28K	 flux Cloud Native Computing Foundation (CNCF) ★ 3,211	 GitHub Actions GitHub MCap: \$1.12T	 GitLab GitLab ★ 22,045 Funding: \$436.2M
 go GoCD Thoughtworks ★ 5,262 Funding: \$28M	 Google Cloud Build Google MCap: \$894.66B	 harness Harness Harness.io Funding: \$80M	 Jenkins Jenkins Continuous Delivery Foundation (CDF) ★ 14,305	 JENKINS X Continuous Delivery Foundation (CDF) ★ 3,181	 Octopus Deploy Octopus Deploy Funding: \$2M	 Screwdriver.cd Screwdriver Verizon Media ★ 740 MCap: \$245.71B	 Semaphore Semaphore SemaphoreCI
 shippable Shippable Shippable Funding: \$10.05M	 Spinnaker Spinnaker Continuous Delivery Foundation (CDF) ★ 6,665	 TeamCity TeamCity JetBrains	 Travis CI Travis CI ★ 568	 HYSCALE wavemaker WaveMaker HyScale WaveMaker	 weave flagger Weaveworks Funding: \$20M ★ 1,229	 DEPLOY XL Deploy XebiaLabs Funding: \$121.5M	

- Competing projects
- Conflicting Terminology
- Achieving the same goal

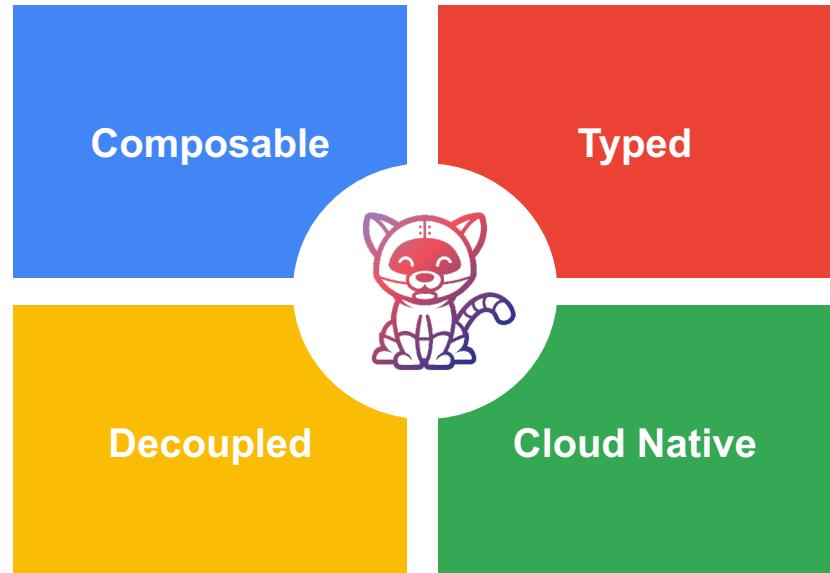
Tekton Emerges

- 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 resource 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

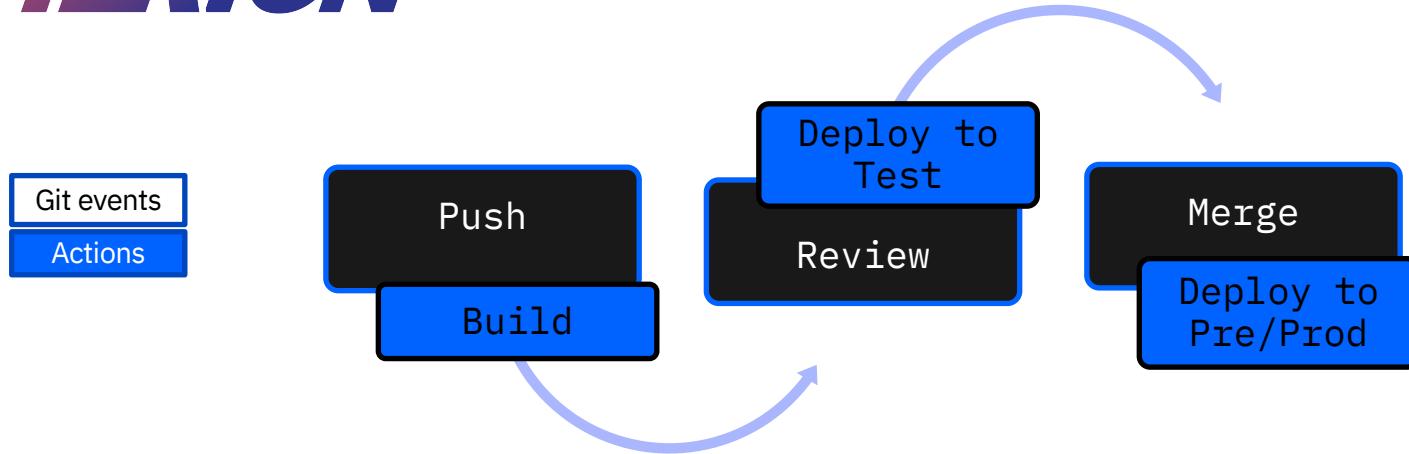
Goals

- 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



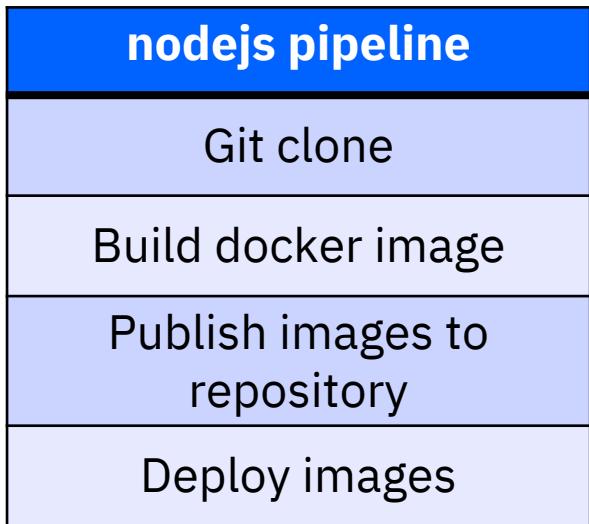
Pipelines



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

```
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

```
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

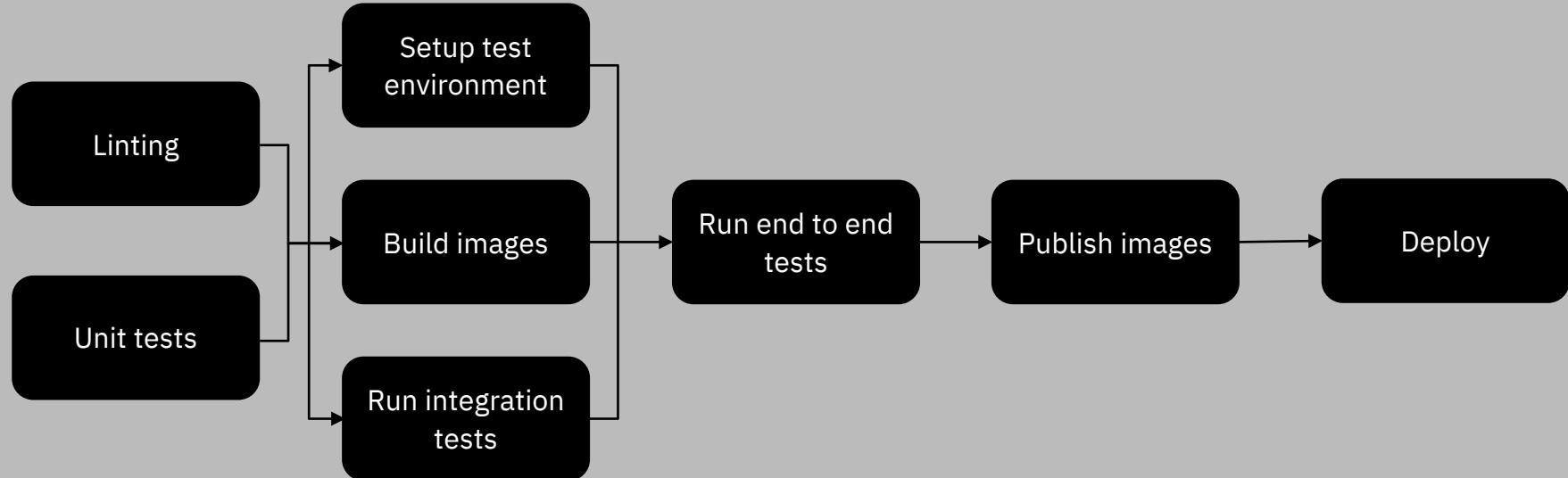
```
apiVersion: tekton.dev/v1alpha1
kind: Pipeline
metadata:
  name: tutorial-pipeline
spec:
  - name: build-app
    taskRef:
      name: build-push
    resources:
      outputs:
        - name: image
          resource: my-image
  - name: deploy-app
    taskRef:
      name: deploy-kubectl
    resources: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

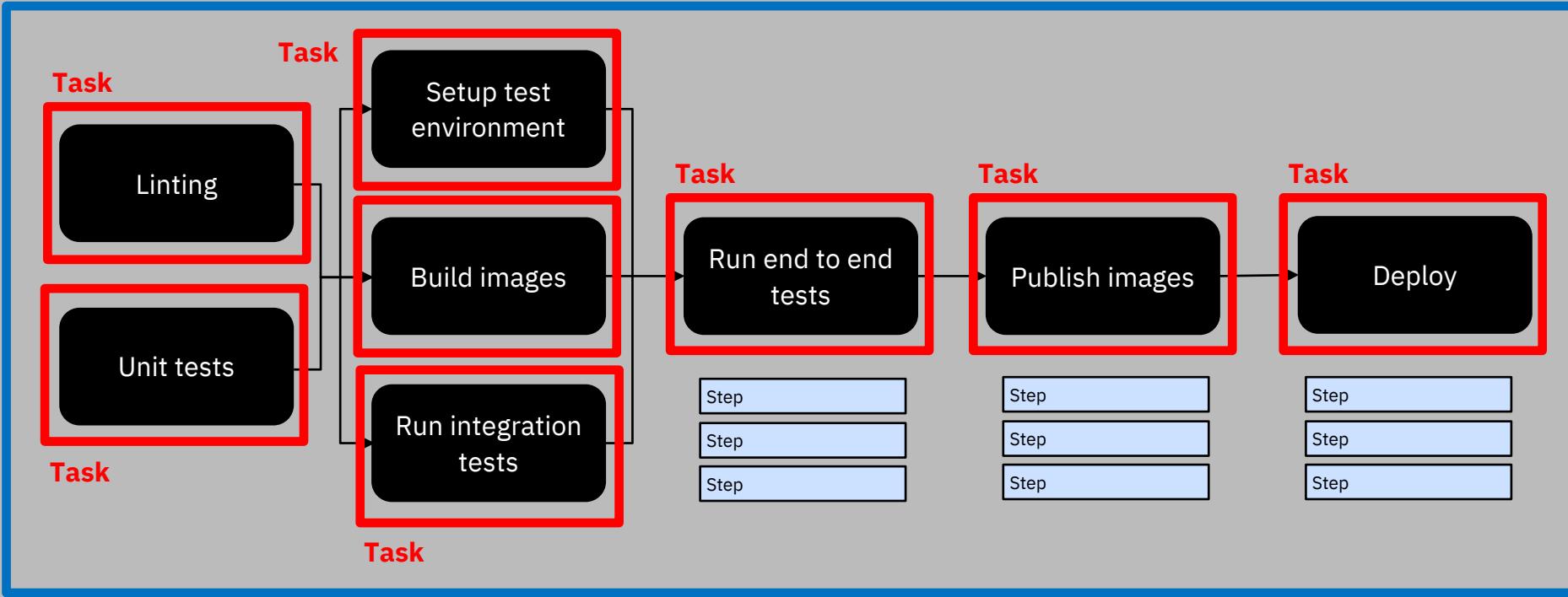
```
apiVersion: tekton.dev/v1alpha1
kind: PipelineRun
metadata:
  name: tutorial-pipeline-run-1
spec:
  serviceAccountName: tutorial-service
  pipelineRef:
    name: tutorial-pipeline
  resources:
    - name: source-repo
      resourceRef:
        name: skaffold-git
    - name: web-image
      resourceRef: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

Putting it all together



Putting it all together

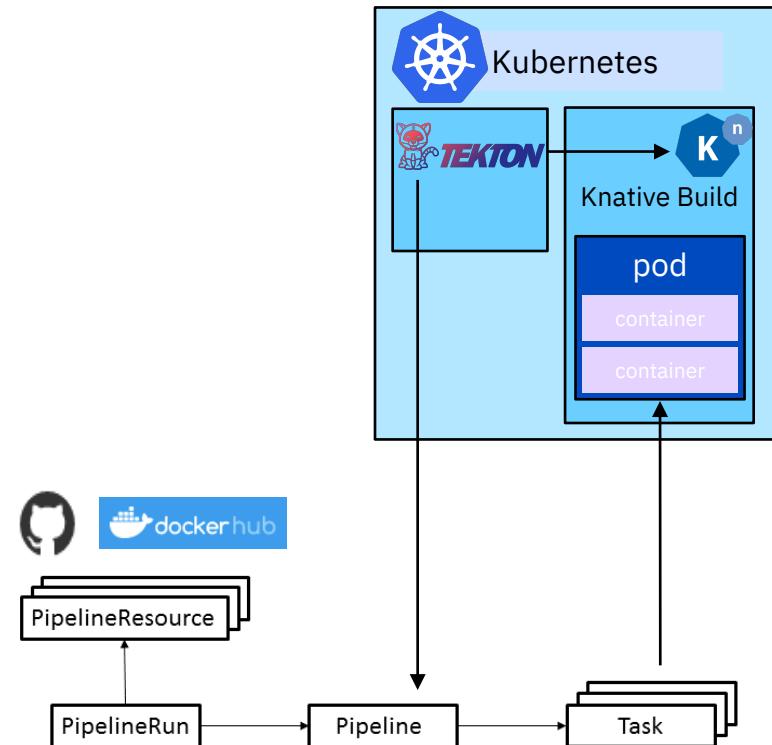
Pipeline



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>

The screenshot shows the Tekton Dashboard interface. On the left, there's a sidebar with navigation links: Pipelines, PipelineRuns, PipelineResources, Tasks, ClusterTasks, TaskRuns, Namespace (selected), and Import Tekton resources, Secrets, Webhooks. The main area displays a PipelineRun titled "backend-webhook-1573101408" which has "Succeeded" status and was run on 2019-11-07T04:41:52Z. The PipelineRun details page shows a list of tasks: build-task, assemble-extract, build-push, and deploy-task. The "assemble-extract" task is highlighted in green and marked as "Completed". Below the tasks, there are tabs for "Logs", "Status", and "Details". The "Logs" tab contains a large block of command-line output from the "assemble-extract" task.

```
[Debug] Image repository set to: index.docker.io
[Info] Running with command line args: appody extract --buildah --target-dir /workspace/extracted -v
[Info] Extracting project from development environment
[Debug] Checking if target-dir exists: /workspace/extracted
[Debug] Creating extract dir: /builder/home/.appody/extract
[Debug] Creating extract dir: /builder/home/.appody/extract/quote-backend
[Debug] kabanero/java-spring-boot2:0.3 image pulled status: false
[Debug] Pull policy Always
[Debug] Pull policy Always
[Info] Pulling docker image kabanero/java-spring-boot:0.3
[Info] Running command: buildah pull kabanero/java-spring-boot:0.3
[Info] Getting image source signature
[Info] Copying blob sha256:504de453d49bd59499864f0e9bf6579df30d0bb1ab41bcd2adab77584b5db
[Info] Copying blob sha256:c25d76c4d7f827bac47fe942ea6b295a2862bd57e56c74576acc45357b1bd7b22192
[Info] Copying blob sha256:c5851d593122e242a5e6fc99813ac8384d81a5c4285bdc2357b41224b1970
[Info] Copying blob sha256:1884974901374fc0b48610647d64a390ee128b3994ac45357b1bd7b22192
[Info] Copying blob sha256:1884974901374fc0b48610647d64a390ee128b3994ac45357b1bd7b22192
[Info] Copying blob sha256:1884974901374fc0b48610647d64a390ee128b3994ac45357b1bd7b22192
[Info] Copying blob sha256:e14d9721a2e509a1b16c83414b25a5453b9e39b2a2439b19400e-1931-00-07
[Info] Copying blob sha256:ea00ff1f565e23b61af1f31acab11hd325hb1d85d2ac48099e3cf1fa43989
[Info] Copying blob sha256:h3d402316e26f44dd508994c4d3d8e04fcf1fd123892f7b65c447b0d720a1f0c
[Info] Copying blob sha256:1b63ca3a3f77979fc499e9c62044e58768193bae04cc9399286f09b211649
[Info] Copying blob sha256:55a0998808579fe1808e313e8e7871124922476c3d323440578756c46fe5bd4
[Info] Copying blob sha256:e4e00d13eebda2684909ef886a059312b7bd3d6040e0536c916c73be8a740f16c4
[Info] Copying blob sha256:681e9578506736fb1e26c10c8237408f126de12254dbcce07b3e0a491ee3025
[Info] Copying blob sha256:5ade4a417c034b5a2be7d4c86e5a5a57c7a25680883dcbfe593a574439fe09d
[Info] Copying blob sha256:ea2581d915ba356d0b79935b6e6113eb3e385f173a7f5396f548e08980fb6a
[Info] Copying blob sha256:d5173aec833c64c0c6bc357a28be9de825681a5c94194f3f65199f9f93db71a6
[Info] Copying blob sha256:92bd6095cce649da347e7a0986595868c7d2a015c6e201496d166be0945f9
[Info] Copying blob sha256:ea2f74001080a43d1c2179578a2668c1048e583b03c1a828a85142b4f238
[Info] Copying blob sha256:6c92c84432e13c4e13c516e3259b285e04c74472c35e9251a7507edf2b598d
[Info] Copying config sha256:20f8035e6fe4ad2962529198d8857c950aca630b248eda170adf147b2b3b707d
```

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>

The screenshot shows the GitHub repository page for 'tektoncd/catalog'. At the top, it displays basic repository statistics: 109 commits, 4 branches, 0 packages, 0 releases, 28 contributors, and Apache-2.0 license. Below this, a list of recent commits is shown, each with a small icon, the author's name, a brief description, and the time since the commit.

Author	Commit Message	Time Ago
chmouel and tekton-robot	Add Taskrun e2e testing	Latest commit 43f52ad 22 days ago
.github	Add any missing basic docs	7 months ago
ansible-tower-cli	Initial commit of the tower-cli task for tekton catalog	7 days ago
argocd	Update templating to use `\$(...)` instead of `\${...}`	2 months ago
buildah	Add Taskrun e2e testing	4 hours ago
buildkit-daemonless	buildkit: use mTLS and support daemonless mode	24 days ago
buildkit	buildkit: use mTLS and support daemonless mode	24 days ago
buildpacks	Add OWNERS file to buildpacks task	8 days ago
conftest	Be explicit about files being a string	9 days ago
gcloud	Add gcloud Task	last month
gke-deploy	Removed numbered list from Install Tekton Pipelines CLI section.	21 hours ago
golang	Add initial OWNERS to some folders	22 days ago
jib-maven	Add Taskrun e2e testing	4 hours ago
kaniko	Add Taskrun e2e testing	4 hours ago
kn	References latest kn release v0.10.0 in task	7 days ago
knctl	Update templating to use `\$(...)` instead of `\${...}`	2 months ago
kubeval	Add initial OWNERS to some folders	22 days ago
makisu	Update templating to use `\$(...)` instead of `\${...}`	2 months ago

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

Tekton Features: Webhooks

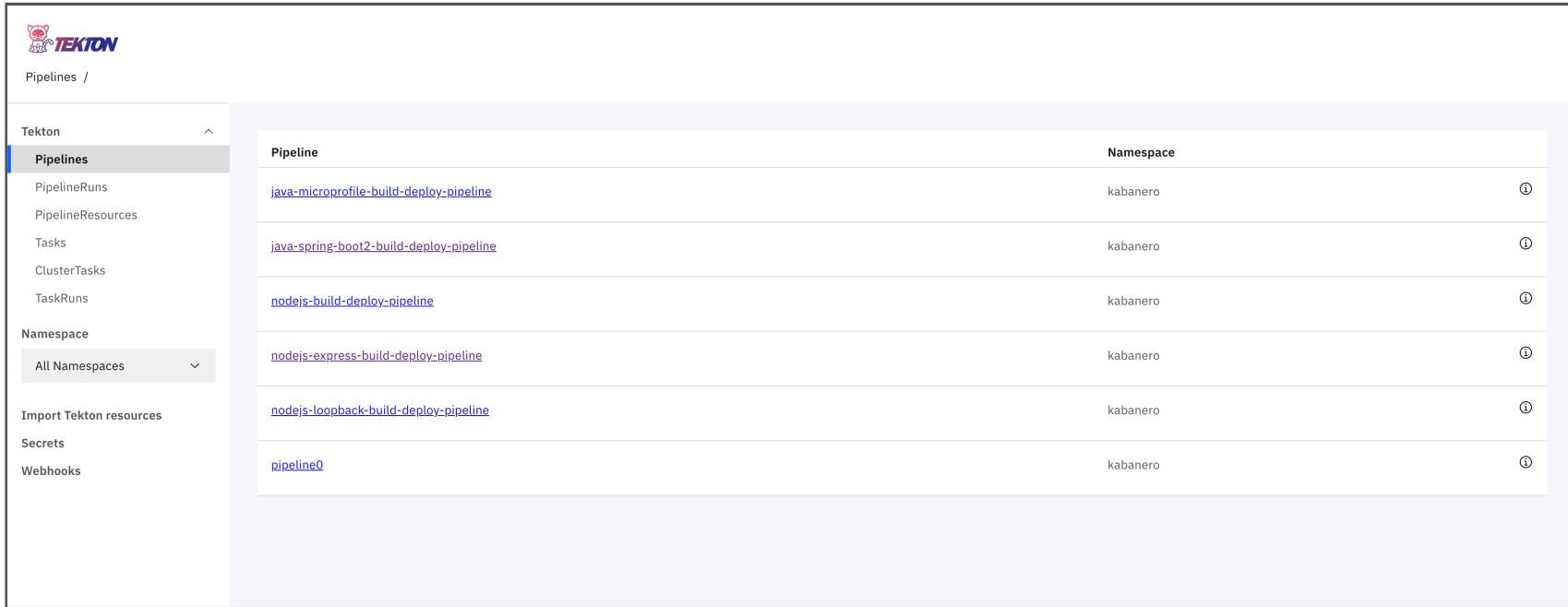
<https://github.com/tektoncd/experimental/tree/master/webhooks-extension>

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

The screenshot shows the 'Create Webhook' form in the Tekton UI. On the left, a sidebar lists 'Tekton' resources: Pipelines, PipelineRuns, PipelineResources, Tasks, ClusterTasks, TaskRuns, Namespace (set to 'kabanero'), Import Tekton resources, Secrets, and Webhooks (which is selected and highlighted with a blue border). The main area is titled 'Create Webhook'. It contains two sections: 'Webhook Settings' and 'Target Pipeline Settings'. In 'Webhook Settings', there are fields for 'Name' (placeholder: 'Enter display name here'), 'Repository URL' (placeholder: 'https://github.com/org/repo.git'), and 'Access Token' (placeholder: 'select secret'). A '+' icon is available to add more secrets. In 'Target Pipeline Settings', there are fields for 'Namespace' (placeholder: 'select namespace'), 'Pipeline' (placeholder: 'select pipeline'), 'Service Account' (placeholder: 'select service account'), and 'Docker Registry' (placeholder: 'Enter docker registry here'). At the bottom right are 'Cancel' and 'Create' buttons.

Tekton with Kabanero

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



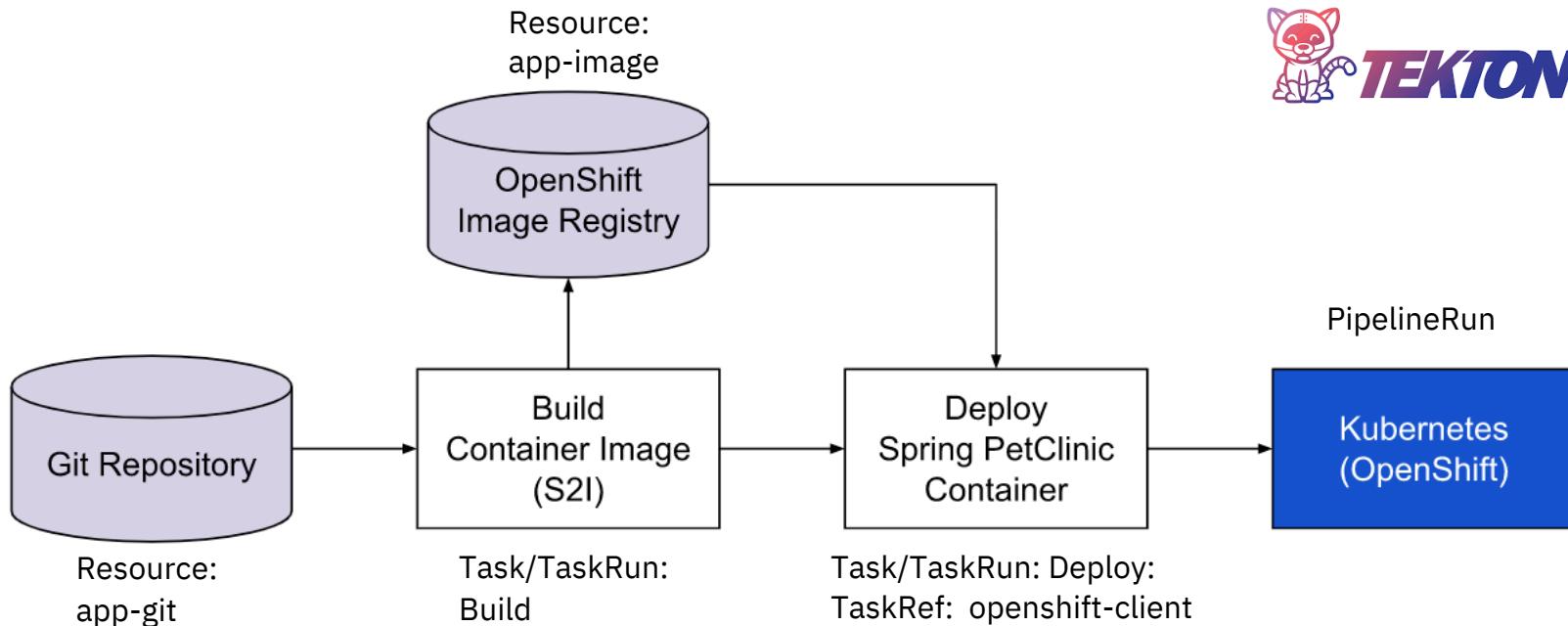
The screenshot shows the Kabanero UI interface for managing Tekton Pipelines. On the left, there is a sidebar with the Kabanero logo and navigation links for Tekton, Pipelines, PipelineRuns, PipelineResources, Tasks, ClusterTasks, TaskRuns, Namespace, Import Tekton resources, Secrets, and Webhooks. The Pipelines link is currently selected. The main area displays a table of Pipelines:

Pipeline	Namespace	Actions
java-micrometer-build-deploy-pipeline	kabanero	ⓘ
java-spring-boot2-build-deploy-pipeline	kabanero	ⓘ
nodejs-build-deploy-pipeline	kabanero	ⓘ
nodejs-express-build-deploy-pipeline	kabanero	ⓘ
nodejs-loopback-build-deploy-pipeline	kabanero	ⓘ
pipeline0	kabanero	ⓘ

- 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



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.

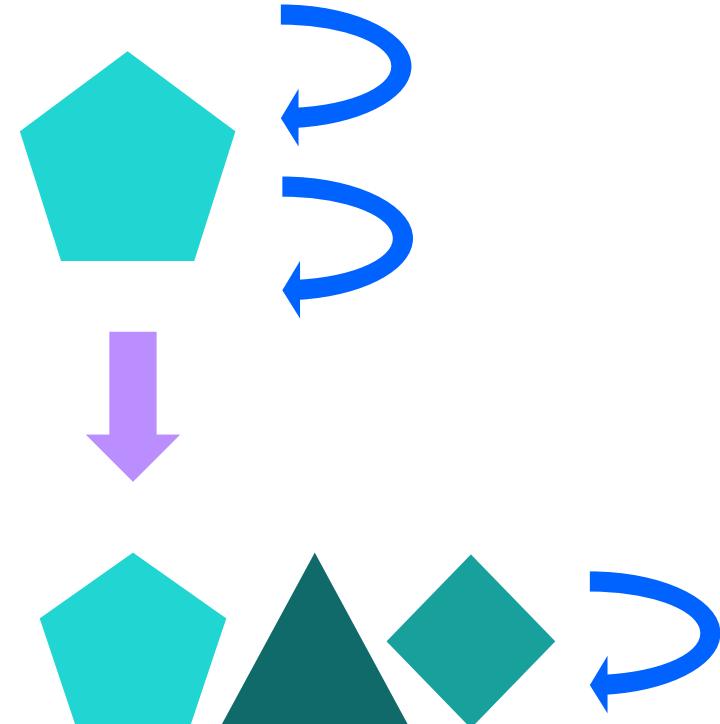
CI/CD Pipelines for Microservices

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

