Lab 14A. Tekton

Tekton is a Kubernetes-native continuous integration technology. It extends the features of Kubernetes with the use of custom resources and controllers. In this exercise, you are going to learn how to:

- Set up Tekton with controllers and CRDs.
- Get started with tkn CLI.
- Read and understand Tekton resources.
- Set up a sample CI Pipeline with Tekton.

Prerequisites:

- A working Kubernetes cluster.
- **`kubectl`** configured with the context set to connect to this cluster.
- Default storage class available in the cluster to provision volumes dynamically.

Install Tekton

Start with:

```
kubectl apply --filename
https://storage.googleapis.com/tekton-releases/pipeline/latest/release
.yaml
```

To understand what gets created, run the following commands:

```
kubectl get ns
kubectl get all -n tekton-pipelines
kubectl get configmap,secret,serviceaccount,role,rolebinding -n
tekton-pipelines
kubectl get crds | grep -i tekton
```

To install Tekton CLI (i.e. tkn) refer to the instructions specific to your OS.

Following instructions install Tekton CLI on Ubuntu:

```
sudo apt update;sudo apt install -y gnupg

sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
3EFE0E0A2F2F60AA

echo "deb http://ppa.launchpad.net/tektoncd/cli/ubuntu eoan main"|sudo
tee /etc/apt/sources.list.d/tektoncd-ubuntu-cli.list

sudo apt update && sudo apt install -y tektoncd-cli
```

Validate:

```
tkn

tkn version
```

Get started with Tekton using the following subcommands. These subcommands help you list Tekton pipelines, PipelineRuns, tasks and TaskRuns. Reference the video lessons and follow along:

```
tkn

tkn p

tkn pr

tkn t

tkn tr
```

Set Up a Continuous Integration Pipeline with Tekton

Set Up the Prerequisite Tekton Tasks

Begin by creating the following prerequisite tasks:

- git-clone: is used to clone the source repository and pass on the latest commit hash to other tasks.
- kaniko: is used to build Docker images and publish those to the registry. This is a cleaner Kubernetes native solution than using Docker-based image builds.

You can also search for a catalog of tasks that comes with Tekton:

- Tekton Hub (beta)

- Tekton Catalog repository

To add the git-clone task, run:

```
kubectl apply -f
https://raw.githubusercontent.com/tektoncd/catalog/main/task/git-clone
/0.3/git-clone.yaml
```

Similarly, install a kaniko task which would read the Dockerfile and build an image out of it. To install a kaniko task do:

```
kubectl apply -f
https://raw.githubusercontent.com/tektoncd/catalog/main/task/kaniko/0.
3/kaniko.yaml
```

```
tkn t list
```

Sample output:

```
NAME         AGE
git-clone    19 minutes ago
kaniko       8 seconds ago
```

Create Tekton Pipeline Resource

Begin with:

```
tkn p list
tkn t list
```

```
git clone https://github.com/lfd254/tekton-ci.git
cd tekton-ci/base
```

```
kubectl apply -f instavote-ci-pipeline.yaml
```

Validate:

```
tkn p list
tkn p describe instavote-ci
```

Create Pipeline Run for Vote App

PipelineRun allows you to launch an actual CI pipeline by creating an instance of a template (Pipeline) with application-specific inputs (resources).

Edit **vote-ci-pipelinrun.yaml** file with actual values. Following are the parameters displayed from the pipeline run file:

```
params:
  - name: repoUrl
    value: https://github.com/xxxxxx/example-voting-app.git
  - name: revision
    value: main
  - name: sparseCheckoutDirectories
    value: /vote/
  - name: imageUrl
    value: xxxxxx/vote
  - name: pathToContext
    value: vote
```

Replace values for:

- **repoUrl**: to point to the repository where your application source is.
- **imageUrl**: to point to your Docker Hub/registry user/organization ID you publish the image to.

Begin by listing the Pipelines and PipelineRuns as:

```
tkn p list
tkn pr list
```

Launch a pipeline run for the **vote** app:

```
kubectl apply -f vote-ci-pipelinerun.yaml
```

Validate and watch the pipeline run with:

```
tkn pr list
tkn pr logs -f vote-ci
```

You may see that the pipeline run exits with an error while trying to push the image to the registry. This is expected as kaniko needs registry secrets in order to authenticate and push a container image.

Generate Docker Registry Secret

To provide kaniko with a secret to connect with your registry, you need to run **docker login** at least once. This example assumes Docker Hub as a registry.

Log in to Docker Hub once using the following command:

```
docker login
```

This generates a **config.json** file which you would need to encode with base64 and add as a Kubernetes secret:

```
cat ~/.docker/config.json
cat ~/.docker/config.json | base64 | tr -d '\n'
```

Now copy the base64 encoded string and generate a secret yaml manifest similar to the configuration below by replacing the value for **config.json** with the actual encoded string.

File: **dockerhub-creds-secret.yaml**

```
apiVersion: v1
kind: Secret
metadata:
 name: dockerhub-creds
data:
 config.json: <base-64-encoded-json-here>
```

Replace **<base-64-encoded-json-here>** with the actual value.

```
kubectl apply -f dockerhub-creds-secret.yaml
kubectl describe secret dockerhub-creds
```

Please note that this secret has already been referenced in the PipelineRun resources.

Following is an example snippet from the **vote-ci-pipelinerun.yaml** spec. A source of which could be studied here:

```
  workspaces:
  - .....
  - name: dockerconfig
    secret:
      secretName: dockerhub-creds
```

In case you use a different name when creating the secret, ensure that you replace the configuration above to point to it.

Now, proceed to create a new pipeline by deleting the previous ones and run the following command:

```
tkn pr rm vote-ci
kubectl apply -f vote-ci-pipelinerun.yaml
```

Validate and watch the pipeline run with:

```
tkn pr logs --last --follow --all
```

Once the pipeline run is complete, you should see a new container image published on Docker Hub with a tag in the format **main-commit-timestamp**.

Setup Continuous Integration for Result App

Since you already have a template in the form of a pipeline, setting up a CI for the **`result`** app is just a matter of creating an instance of it, by providing application-specific (result app) inputs.

That's what you do creating a pipeline run object.

You will find the pipeline run spec in the same repository that you have cloned along with the code for **`vote`** pipeline run.

Update the params as previously:

File: **`result-ci-pipelinerun.yaml`**

```
  params:
  - name: repoUrl
    value: https://github.com/initcron/example-voting-app.git
  - name: revision
    value: master
  - name: sparseCheckoutDirectories
    value: /result/
  - name: imageUrl
    value: initcron/tknresult
  - name: pathToContext
    value: result
```

```
kubectl apply -f result-ci-pipelinerun.yaml
```

Validate and watch the pipeline run with:

```
tkn pr list
tkn pr logs --last --follow --all
```

Validate by checking if the container image is published on Docker Hub for the **`result`** app.

Additional Resources

- *"Getting Started"*
- *"Concepts"*
- Tekton - official website
- *"Getting Started with Triggers"*

Summary

In this lab exercise, you have learned how to natively run continuous integration pipelines with Tekton on Kubernetes. There are tools such as Jenkins X and OpenShift which use Tekton as an engine and have built more sophisticated systems on top of it. We hope that this lab exercise along with the lecture part made you curious about Kubernetes-native CI/CD. Go ahead and explore it further!