**THE STEPS TO CONFIGURE BASIC COMPONENTS OF TEKTON PIPELINE**

TASK

The main objective of Tekton Pipelines is to run your Task individually or as a part of a Pipeline. Every Task runs as a Pod on your Kubernetes cluster with each step as its own container.

apiVersion: tekton.dev/v1alpha1

kind: Task

metadata:

name: echo-hello-world

spec:

steps:

- name: echo

image: ubuntu

command:

- echo

args:

- "hello world"

TASK RUN

A [TaskRun](https://github.com/tektoncd/pipeline/blob/master/docs/taskruns.md) runs the Task you defined. Here is a simple example of a TaskRun you can use to execute your Task:

apiVersion: tekton.dev/v1alpha1

kind: TaskRun

metadata:

name: echo-hello-world-task-run

spec:

taskRef:

name: echo-hello-world

To apply the yaml files, use the following command:

oc apply -f <name-of-file.yaml>

To see the output of the TaskRun, use the following command:

tkn taskrun describe echo-hello-world-task-run

You will get an output similar to the following:

Name: echo-hello-world-task-run

Namespace: default

Task Ref: echo-hello-world

Status

STARTED DURATION STATUS

4 minutes ago 9 seconds Succeeded

Input Resources

No resources

Output Resources

No resources

Params

No params

Steps

NAME

echo

The status of type Succeeded shows that the Task ran successfully.

To see the actual outcome, use the following command:

tkn taskrun logs echo-hello-world-task-run

You will get an output similar to this:

[echo] hello world

**INPUT/ OUTPUT**

In more common scenarios, a Task needs multiple steps with input and output resources to process. For example a Task could fetch source code from a GitHub repository and build a Docker image from it.

[PipelineResources](https://github.com/tektoncd/pipeline/blob/master/docs/resources.md) are used to define the artifacts that can be passed in and out of a Task. There are a few system defined resource types ready to use, and the following are two examples of the resources commonly needed.

The [git resource](https://github.com/tektoncd/pipeline/blob/master/docs/resources.md#git-resource) represents a git repository with a specific revision:

apiVersion: tekton.dev/v1alpha1

kind: PipelineResource

metadata:

name: skaffold-git

spec:

type: git

params:

- name: revision

value: master

- name: url

value: https://github.com/GoogleContainerTools/skaffold

The [image resource](https://github.com/tektoncd/pipeline/blob/master/docs/resources.md#image-resource) represents the image to be built by the Task:

apiVersion: tekton.dev/v1alpha1

kind: PipelineResource

metadata:

name: skaffold-image-leeroy-web

spec:

type: image

params:

- name: url

value: gcr.io/<use your project>/leeroy-web # secret is needed if the registry is not local i.e docker, quay, harbor etc. For local registry in openshift 4 use the following path : image-registry.openshift-image-registry.svc:5000

The following is a **Task** with inputs and outputs. The input resource is a GitHub repository and the output is the image produced from that source. The args of the Task command support variable substitution so that the definition of Task is constant and the value of parameters can change in runtime.

apiVersion: tekton.dev/v1alpha1

kind: Task

metadata:

name: build-docker-image-from-git-source

spec:

inputs:

resources:

- name: docker-source

type: git

params:

- name: pathToDockerFile

type: string

description: The path to the dockerfile to build

default: /workspace/docker-source/Dockerfile

- name: pathToContext

type: string

description:

The build context used by Kaniko

(https://github.com/GoogleContainerTools/kaniko#kaniko-build-contexts)

default: /workspace/docker-source

outputs:

resources:

- name: builtImage

type: image

steps:

- name: build-and-push

image: gcr.io/kaniko-project/executor:v0.15.0

# specifying DOCKER\_CONFIG is required to allow kaniko to detect docker credential

env:

- name: "DOCKER\_CONFIG"

value: "/tekton/home/.docker/"

command:

- /kaniko/executor

args:

- --dockerfile=$(inputs.params.pathToDockerFile)

- --destination=$(outputs.resources.builtImage.url)

- --context=$(inputs.params.pathToContext)