Lab 9.3. Create a Pipeline to Build, Push and Deploy Workload into the Cluster

Overview

In this example, we will utilize all of the previously created tasks. We will use the **`git clone`** task to clone the repository for both building the image and deploying the workload. Additionally, we will use the **`build and push`** task to build the image using Kaniko and push it to the repository. Finally, we will use the **`deploy to cluster`** task to deploy the workload into the cluster using helm. Let's take a look at these tasks.

First Task (Git Clone)

This task enables you to clone a GitHub repository by providing its URL and branch name.

You can apply it directly from the Tekton Hub using the following command:

```
kubectl apply -f
https://raw.githubusercontent.com/tektoncd/catalog/main/task/git-clone/0.9/g
it-clone.yaml -n tekton-pipelines
```

Second Task (Build and Push Image)

To build and push an image to the repository, create a **`task`** with the following manifest and name it **`build-push-docker-image-task.yaml`**:

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: build-push-docker-image-task
spec:
  workspaces:
    - name: output
  params:
    - name: app_repo
```

```
    - name: container_image
    - name: container_tag
    - name: secret-name
  volumes:
    - name: kaniko-secret
      secret:
        secretName: $(params.secret-name)#name of the docker secret
        items:
          - key: .dockerconfigjson
            path: config.json
  steps:
    - name: build
      image: gcr.io/kaniko-project/executor:debug
      workingDir: "/workspace/output/"
      command: [/kaniko/executor]
      args:
        - --context=./
        - --destination=$(params.container_image):$(params.container_tag)
        - --force
      volumeMounts:
        - name: kaniko-secret
          mountPath: /kaniko/.docker/
```

In this task, we will be passing the **secret** name, **image** name, and **tag** name as parameters. The task will use the Kaniko command to build and push the image to the repository.

To use this task, we need to pre-apply a secret that contains the Docker registry credentials and pass the secret as a volume.

After creating the secret, apply this task using the following command:

```
kubectl apply -f build-push-docker-image-task.yaml -n tekton-pipelines
```

Third Task (Deploy to Cluster)

Let us create this **task** with the below manifest and save it as **deploy-to-cluster-task.yaml**:

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: deploy-to-cluster-task
  labels:
    app.kubernetes.io/version: "0.3"
  annotations:
    tekton.dev/pipelines.minVersion: "0.12.1"
    tekton.dev/categories: Deployment
```

```
    tekton.dev/tags: helm
    tekton.dev/platforms: "linux/amd64,linux/s390x,linux/ppc64le,linux/arm64"
spec:
  params:
    - name: charts_dir
      description: The directory in source that contains the helm chart
    - name: release_name
      description: The helm release name
      default: <release name>
    - name: release_namespace
      description: The helm release namespace
      default: "default"
    - name: values_file
      description: "The values file to be used"
      default: "values.yaml"
    - name: tag
  workspaces:
    - name: output
  steps:
    - name: upgrade
      image: docker.io/kiwigrid/gcloud-kubectl-helm
      workingDir: /workspace/output
      script: |
        echo current installed helm releases
        helm list --namespace "$(params.release_namespace)"
        helm list -A
        echo installing helm chart...
        helm upgrade --install --wait --values
"$(params.charts_dir)/$(params.values_file)" --set tag="$(params.tag)"
--namespace "$(params.release_namespace)" "$(params.release_name)"
"$(params.charts_dir)" --debug
```

This task requires passing parameters such as **charts_dir**, **release_name**, **release_namespace**, **values_file**, and **tag**. It utilizes the **helm** command to deploy the code into the cluster.

Before applying this task, certain permissions need to be granted for our Service Account.

```
kubectl apply -f deploy-to-cluster-task.yaml -n tekton-pipelines
```

Pipeline

Now that we have the tasks, we will create a **pipeline** with the following manifest and name it as **build-push-and-deploy-pipeline.yaml**:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
```

```
metadata:
  name: build-push-and-deploy-pipeline
spec:
  params:
    - name: gitrevision-tag
  workspaces:
    - name: shared-data
      description: |
        This workspace will receive the cloned git repo and be passed
        to the next Task for the repo's README.md file to be read.
    - name: shared-data-dep
  tasks:
    - name: fetch-repo
      taskRef:
        name: git-clone
      params:
        - name: url
          value: <your git-repo url>
        - name: revision
          value: $(params.gitrevision-tag)
      workspaces:
        - name: output
          workspace: shared-data
    - name: build-container-image
      runAfter: ["fetch-repo"]
      taskRef:
        name: build-push-docker-image-task
      params:
        - name: app_repo
          value: dir:///workspace/output/ #This path works when Docker is on home
dir
        - name: container_image
          value: <image name>
        - name: container_tag
          value: <image tag>
        - name: secret-name
          value: < name of the secret>
      workspaces:
        - name: output
          workspace: shared-data
    - name: helm-clone
      runAfter: ["build-container-image"]
      taskRef:
        name: git-clone
      params:
        - name: url
          value: "<your repo URL>"
        - name: revision
```

```
          value: $(params.gitrevision-tag)
      workspaces:
        - name: output
          workspace: shared-data-dep
    - name: deploy-to-cluster
      runAfter: ["helm-clone"]
      taskRef:
        name: deploy-to-cluster-task
      params:
        - name: charts_dir
          value: /workspace/output/<helm chart path>
        - name: release_name
          value: <release name>
        - name: release_namespace
          value: <namespace>
        - name: values_file
          value: values.yaml
        - name: tag
          value: <Tag>
      workspaces:
        - name: output
          workspace: shared-data-dep
```

Apply this pipeline with the following command:

```
kubectl apply -f build-push-and-deploy-pipeline.yaml -n tekton-pipelines
```

PipelineRun

Having created the tasks and the pipeline, the next step is to create a pipelineRun. Use the following manifest and save it as `build-push-and-deploy-run.yaml` to create a `PipelineRun`:

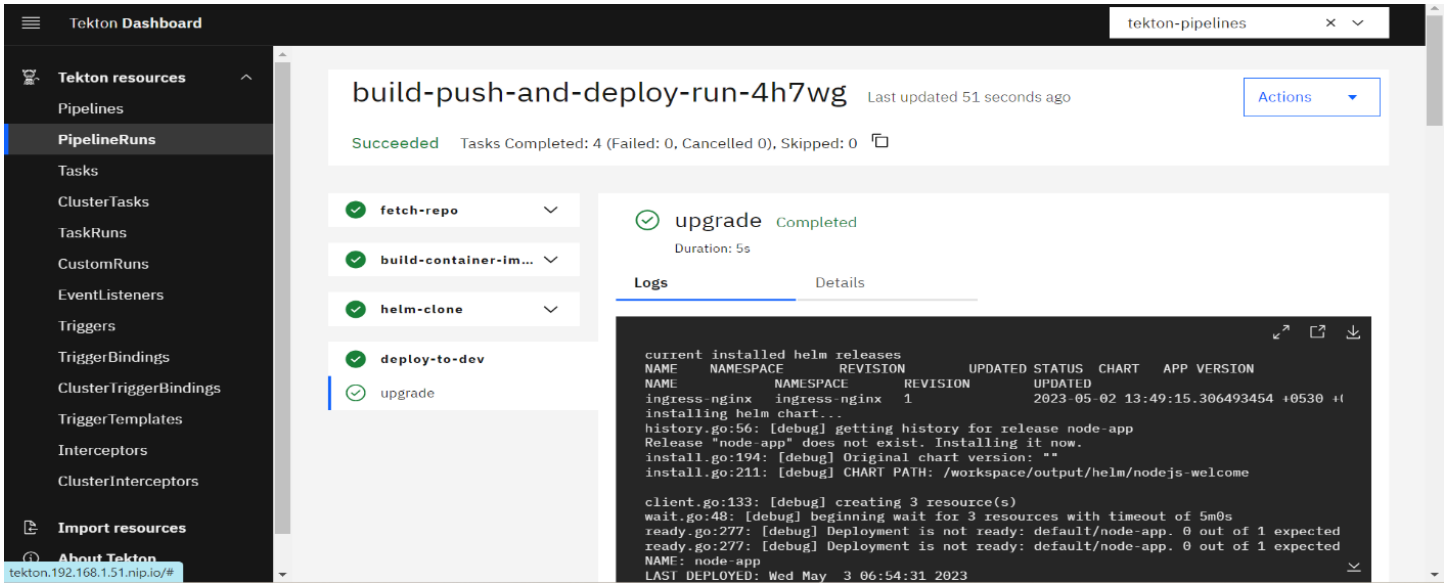
```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: build-push-and-deploy-run-
spec:
  serviceAccountName: <service account name>
  pipelineRef:
    name: build-push-and-deploy-pipeline
  podTemplate:
    securityContext:
      fsGroup: 1001
  params:
    - name: gitrevision-tag
      value: <branch name>
  workspaces:
```

```
  - name: shared-data
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 1Gi
  - name: shared-data-dep
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 1Gi
```

Now apply this manifest with the following command:

```
kubectl create -f build-push-and-deploy-run.yaml -n tekton-pipelines
```

When you have completed all of the steps above, your screen should look like the image below.



Successfully Completed Build, Push and Deploy Pipeline