Project – fourth part:

Project goal:

Deploying a high scalable Python product using CD pipeline.

Solution architecture:

Development language: Python.

3RD parties: Flask, PyMySQL, git, Github, Jenkins, Docker, docker-compose, Docker HUB, K8S, HELM.

Distribution type: Public.

Project guidelines:

1. Jenkins will be used as a CI engine and will run and schedule the pipeline.
2. Git will be used as source control management tool.
3. The Jenkins pipeline must be written inside a Jenkinsfile and committed into any a remote git repository.
4. Docker will be used as a containerization tool.
5. Kubernetes will be used as a container orchestration tool.
6. Delete all unused/unrelated files and code parts.
7. Stick to the specifications document.

Prerequisites:

1. Docker daemon running (Docker toolbox / Docker Desktop)
2. Minikube running.
3. Existing repo on Docker hub (created manually)
4. Existing credentials user in Jenkins (for Docker HUB connection).

HELM chart:

* HELM chart will be uploaded to Github manually.

* HELM chart will consists of the below files:

1. deployment.yaml –
    - Will contain a reference to our image (which was pushed to HUB).
    - Will target **containerPort** of the flask app (for example: 5000).
    - **replicas** (count) and **image name** will be templated and use the values from **values.yaml**
2. service.yaml –
    - Will expose the python app on **TCP** protocol
    - **targetPort** will be the flask app port (for example: 5000).
    - The **type** will be templated and use the values from **values.yaml**
3. Chart.yaml – will contain:
    - apiVersion (default)
    - **name** – your name
    - **description** (of chart)
    - **type**: application
    - **version** (any – incremental).
4. values.yaml – will contain the below values:
    - **replicaCount** of 5
    - **image.repository** = repo name:<Version>
    - **service.type** = LoadBalancer

Python code changes:

- Create new file **K8S_backend_testing.py** which will test your deployed application on the K8S cluster.
- The test will read **k8s_url.txt** file content, and perform the test accordingly.
  For example: http://192.168.99.106:31480/users/…

Jenkins pipeline configurations:

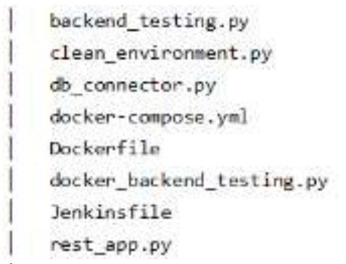
- Jenkins pipeline will use Poll SCM mechanism every 30 minutes to check for a new commit to master branch.
- Old builds will be discarded using “Log rotation strategy” for max days of **5** and max build of **20**.

<u>Jenkins pipeline steps:</u>

1. Pull code from <u>your</u> Github repository holding your project.
2. Run rest_app.py (backend)
3. Run backend_testing.py
4. Run clean_environemnt.py
5. Build Docker image - locally
6. Push Docker image – to HUB
7. Set compose image version – setting the version inside the **.env** file for docker-compose
   This is as simple as calling: `"echo IMAGE_TAG=${BUILD_NUMBER} > .env"`
8. Run docker-compose up -d
9. Test dockerized app – using docker_backend_testing.py
10. Clean compose environment – call docker-compose down and delete local image
11. Deploy HELM chart passing image with –set image.version="my_repo":`${BUILD_NUMBER}`
12. Write your service URL into **k8s_url.txt** file using:
    $ minikube service hello-python-service –url > k8s_url.txt
13. Test deployed app – using K8S_backend_testing.py
14. Clean HELM environment – call HELM delete

<u>What to send?</u>

Link to your Github repo containg the below files:



```
|   backend_testing.py
|   clean_environment.py
|   db_connector.py
|   docker-compose.yml
|   Dockerfile
|   docker_backend_testing.py
|   Jenkinsfile
|   rest_app.py
|   K8S_backend_testing.py
|   HELM chart
```

<u>Extra:</u>

1. Read about K8S **Secret** object and create a **Secret** object in your K8S cluster holding your DB user name and password and use it from your Python code (instead of the existing hard-coded password).
2. Read about K8S **ConfigMap** object Create **ConfigMap** object in your K8S cluster holding your DB host name and use it from your Python code (instead of the existing hard-coded host name).
3.
   - Read about:
     - K8S **Stateful application**.
     - **PersistentVolume**
     - **PersistentVolumeClaim**
   - Create MySQL deployment and change your rest_app code to use it (instead of the remote db).