Docker & Kubernetes ASSIGNMENT

PEER LEARNING DOCUMENT

Problem Statement -

Docker task

1. Write a simple airflow dag to connect with db(postgres) and add entry in db for each execution (Time of dag execution)
2. Add the given Dag into the container and install dependencies.
3. Use docker compose to launch airflow and postgres
4. Schedule the Dag
5. Validate entry in Postgres

Kubernetes Task

1. Create deployment and service for above airflow and postgres (you can use postgres helm chart for Postgres deployment)
2. Deploy airflow and Postgres
3. Schedule the Dag
4. Validate entry in Postgres

Rahul Kumar

Docker task

- He created a dag with two tasks in Python language, which consists of tasks of the name -
    a. Create_table - creating the table
    b. Insert_into_table - Inserting the data into a table
- Created a connection to the database to airflow.
- Created a docker-compose file with all the different services required.
- Using the docker-compose command he started the airflow container.
- He created a connection with Postgres and validated it.

How my approach is different from this?

- The above-described approach is similar to mine, but the environment which we used is different, he created the image from airflow:2.5.1 whereas mine is airflow:2.6.1
- He made to store all the queries, in a folder from which imported .sql files and ran them using PostgresOperator.

Kubernetes steps

- He created a personalized Docker image that includes the Postgres database and the necessary installation tools for connecting to Airflow. He used a docker file to execute instructions prior to installation.
- He starts the Postgres service and deploys the Postgres pod using the Kubectl command.
- Then he started the airflow service, deployed the airflow pod, and produced a dag in the airflow container.

How my approach is different from this?

- He had created a docker file consisting of some pre-necessary commands to make a connection with airflow.
- Later he also used the same minikube service for the rest of the things.

Gnana Praneeth Kothapally

Docker task

- He created a dag with two tasks in Python language, which consists of tasks of the name -
    a. Create_table - creating the table
    b. Insert_into_table - Inserting the data into a table
    c. Query_task - Querying a table, which validates the entries.
- A connection was established to the database for airflow.
- compiled all the necessary services into a docker-compose file.
- He launched the airflow container by using the docker-compose command.
- He established and verified a connection to Postgres.

How my approach is different from this?

- The above-described approach is similar to mine, but the environment which we used is different, he created the image from airflow:2.3.2 whereas mine is airflow:2.6.1
- He made to store all the queries, in a folder from which imported .sql files and ran them using PostgresOperator.
- He made a task over there in the dag itself, to validate the entries in Postgres connection.

Kubernetes steps

- He created a custom docker image for Postgres with airflow installed in it.
- Pushed the Postgres image and the custom airflow image in docker hub for public access.
- Installed minikube to make a Kubernetes cluster.
- Created the Postgres-deployment.yaml file which contains the pod containing Postgres container.
- Created a service of type clusterIP by running postgres-service.yaml to give access to postgres pods inside the cluster.
- Created persistent volumes dags-storage.yaml to mount the dags folder inside the airflow container to the local directory.
- Created a service of type load balancer by running airflow-service.yaml to access airflow webserver from the local system.
- Accessed the airflow webserver by running the command minikube service airflow.

How my approach is different from this?

We both followed a similar approach, there is not much difference between the two approaches followed.

Alternative Solution Kubernetes task-

We can make use of helm charts as suggested in the question

- Add the Bitnami Helm repository by running the command
    - **helm repo add bitnami https://charts.bitnami.com/bitnami**

  This adds the Bitnami repository to Helm, which allows you to acces pre-packaged Helm charts for various applications.

- Deploys a PostgreSQL database using a StatefulSet and Service by running the command
    - **helm install postgresql bitnami/postgresql -f values.yaml**

  This command installs the PostgreSQL Helm chart from the Bitnami repository, using the specified values.yaml file for configuration.

- Verifies that the PostgreSQL deployment is running by running the command:
    - **kubectl get pods**

  This command lists all the pods running in your Kubernetes cluster, including the PostgreSQL pod(s) that were created.

- Adds the Apache Airflow Helm repository by running the command
    - **helm repo add apache-airflow https://airflow.apache.org**

  This adds the Apache Airflow repository to Helm, allowing you to access the Apache Airflow Helm chart.

- Deploys an Airflow instance using the Apache Airflow Helm chart by running the command
    - **helm install airflow apache-airflow/airflow -f values.yaml**

This command installs the Apache Airflow Helm chart, using the specified values.yaml file for configuration.

- Verifies that the Airflow deployment is running by running the command
    - **kubectl get pods**

  This command lists all the pods running in your Kubernetes cluster, including the Airflow pod(s) that were created.

- Create a file named airflow-service.yaml which likely contains the configuration for a Kubernetes Service for the Airflow deployment.

- Applies the service configuration by running the command
    - **kubectl apply -f airflow-service.yaml**

  This command creates or updates the Kubernetes Service based on the configuration provided in the airflow-service.yaml file.

- Retrieves the external IP or DNS for the Airflow service by running the command
    - **kubectl get service airflow-service**

  This command displays the details of the Airflow service, including the external IP address or DNS name that can be used to access the service.

- Lastly, to view the deployments in your Kubernetes cluster, you can run the command
    - **kubectl get deployments**

  This command lists all the deployments in your cluster, including the PostgreSQL and Airflow deployments.