Document: CI/CD pipeline with Azure DevOps

Tasks:

To configure a Continuous Integration/Continuous Deployment (CI/CD) pipeline using Azure DevOps to automate the deployment of the application. The pipeline should automate the containerization and deployment process, such that every time a new feature is added to the application, it will trigger the automatic build of an updated Docker image, its release to Docker Hub, and the deployment of the updated containers to the Kubernetes cluster.

Steps:

1. Created an Azure DevOps project called “Hermas DevOps project” within the Azure DevOps account. This project serves as the foundation for the CI/CD pipeline setup.
2. Began the process of creating an Azure DevOps Pipeline as follows:
    a. Firstly, configured the source repository for the pipeline, choosing GitHub as the source control system where your application code is hosted and selected the aks-terraform-main repository.
    b. Created the pipeline using a Starter Pipeline template, to serve as the foundation for further customization in upcoming tasks.
3. Set up a service connection between Azure DevOps and the Docker Hub account where the application image is stored, facilitating the seamless integration of the CI/CD pipeline with the Docker Hub container registry. Followed these steps to set up this connection:
    a. Created a personal access token on Docker Hub.
    b. Configured an Azure DevOps service connection to utilize this token.
    c. Verified that the connection was successfully established.
4. Modified the configuration of the starter pipeline template to enable it to build and push a Docker image to Docker Hub following these steps:
    a. Added the Docker task with the buildandPush command to the pipeline. Used the same Docker image name as previously used for pushing to Docker Hub from the local development environment.
    b. Set up the pipeline to automatically run each time there is a push to the main branch of the application repository in GitHub.
    c. Ran the CI/CD pipeline and then tested the newly created Docker image. Tested the image by pulling the latest version from Docker Hub on my local environment. Ran the container and tested its functional to ensure the application works as expected.
5. Created and configured an AKS service connection within Azure DevOps to establish a secure link between the CI/CD pipeline and the AKS cluster, enabling seamless deployments and effective management.
6. Further modified the configuration of the CI/CD starter pipeline template this time to incorporate the Deploy to Kubernetes task with the deploy kubectl command leveraging the deployment manifest available in the application repository in GitHub, as well as the previously established AKS connection to facilitate the automatic deployment of the application to the AKS cluster. This is the deploy or release component of the CI/CD pipeline.
7. Tested and validated the functionality of the CI/CD pipeline. This step ensures the seamless execution of the deployment process and verifies that the application performs as expected on the AKS cluster. Performed the tests as follows:
    a. Began by monitoring the status of pods within the cluster to confirm they have been created correctly.

b. Initiated port forwarding using kubectl to access the application running on AKS cluster securely issuing this command on the cli: `kubectl port-forward [pod-name] 5000:5000.` Accessed the locally exposed address provided by the cluster (127.0.0.1:5000) and tested the functionality of the application to ensure it operates correctly, thus validating the effectiveness of the CI/CD pipeline in application deployment. Was able to add a new order.