**In-class Activity 3**

Galgamuge Emmanuel Fernando

C0918066

CBD 3324: Containerization and Container Delivery

December 03, 2024

1. **Repository URL:** [**https://github.com/RuFerdZ/ica-03-3324/**](https://github.com/RuFerdZ/ica-03-3324/)
2. **Project Structure**

**A screenshot of a computer

Description automatically generated**

Figure 1: Project structure.

1. **Initial Configuration.**

* Installed ArgoCD on the Kubernetes cluster.
* In the ArgoCD dashboard, go to Settings > Repositories and add the GitHub repository that holds the Helm Chart files.
* In this way we have access and can create applications out of the repository we added.
* For all ArgoCD applications created, I have enabled **Automatic Synchronization** to automatically sync all changes in the GitHub repository with the Kubernetes Cluster.

1. **Database Deployment**

Initially, I created the Helm Charts to deploy the database and deployed it using ArgoCD.

A screenshot of a chat

Description automatically generated

Figure 2: Database deployment.

A screen shot of a computer

Description automatically generated

Figure 3: ns: db.

The values.yaml file for the database is:

**A screenshot of a computer program

Description automatically generated**

Figure 4: Database - values.yaml

1. **WordPress Deployment.**

Next, I deployed the WordPress application.

A screenshot of a computer

Description automatically generated

Figure 5: WordPress application deployment.

A screen shot of a computer

Description automatically generated

Figure 6: ns: wp.

The values.yaml file for the WordPress application is:

A screen shot of a computer

Description automatically generated

Figure 7: WordPress - values.yaml.

Tested the deployment via port forwarding:

A screenshot of a computer

Description automatically generated

Figure 8: Wordpress Application.

1. **Nginx Proxy Deployment**

To deploy the Nginx proxy and serve the WordPress application via the proxy, I did the following steps:

* Create Nginx configuration file

A screenshot of a computer program

Description automatically generated

Figure 9: Nginx.conf file.

* Create a Dockerfile, built it and push to the DockerHub.

A screen shot of a computer program

Description automatically generated

Figure 10: Dockerfile.

Next, I deployed the Nginx Proxy:

A screenshot of a chat

Description automatically generated

Figure 11: Nginx-proxy deployment.

**Note**: *the app health is in progress because I deployed it locally as a Load Balancer, and it won't assign a public IP address. To test this, I tested via port forwarding.*

A screenshot of a computer screen

Description automatically generated

Figure 12: ns: proxy.

The values.txt file for nginx deployment:

A screenshot of a computer program

Description automatically generated

Figure 13: Nginx Proxy - values.yaml.

Next test whether we can access the WordPress application via Nginx proxy.

A screenshot of a computer

Description automatically generated

Figure 14: WordPress application via Nginx Proxy.

1. **Challenges Faced**

* Challenge: Improper configuration of the service URLs in the incorrect format.

Debugging phase:

* + Log into pods and check if environment variables are set – they were set!
  + When deploying the nginx proxy, It logged an error saying the host was not found, which is the service URL.
  + Via this narrowed down the issue was on the host URL (service URL).
  + It was the format issue:
    - Issue: <ns-name>.<svc-name>.svc.cluster.local
    - Correct format <svc-name>.<ns-name>.svc.local

Solution: Sir pointed out the issue.

1. **Improvements**

* Encrypt the secrets before pushing them to the GitHub repository.
  + One option is to use Google Secret Manager to store and apply the secrets during deployment by referencing them using a “SecretProviderClass” resource type in Kubernetes.
* Tag images with a commit hash since it won't detect there is an image update if it is always “latest”.
* Deploy it in a cloud-based Kubernetes environment (GKE, AKS, etc..).