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/

2. **Project Structure**

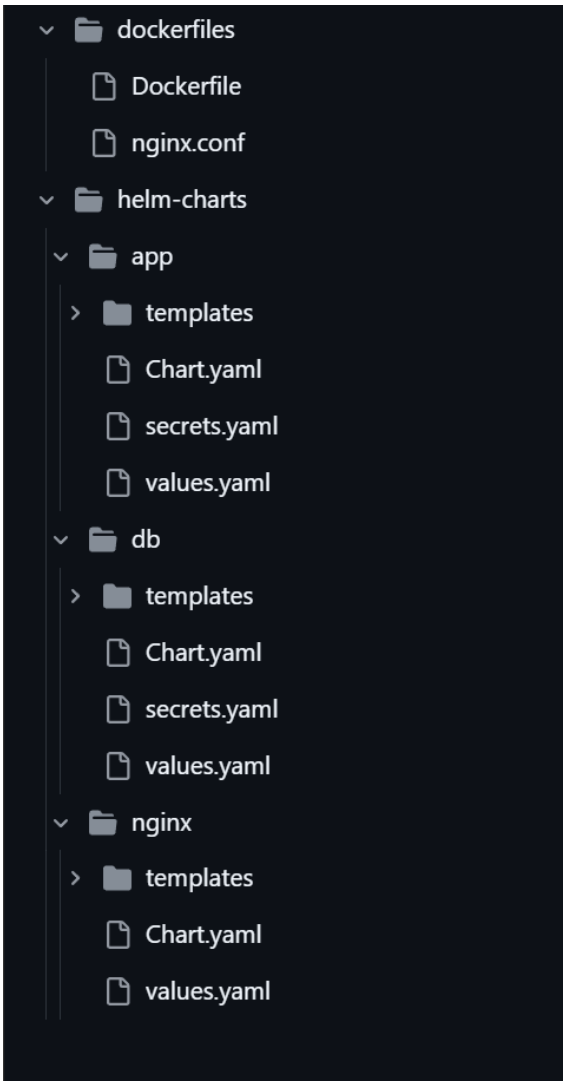

Figure 1: Project structure.

3. 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.

4. Database Deployment

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

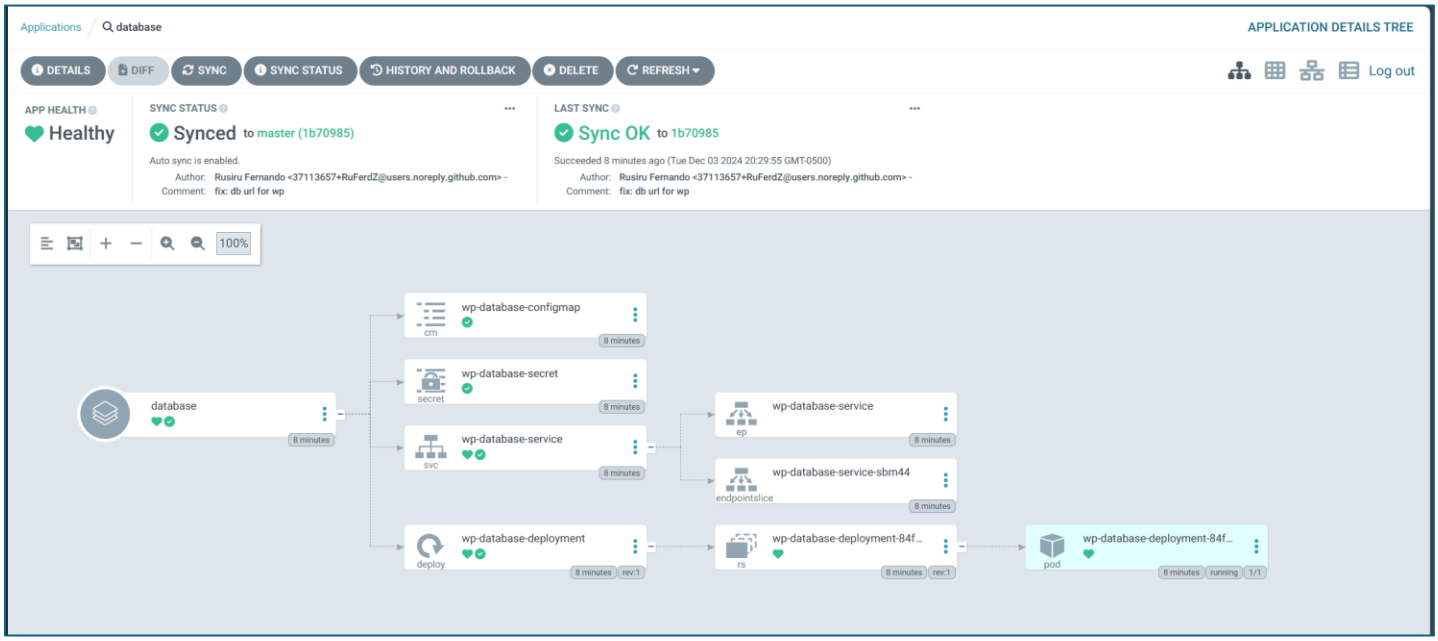

Figure 2: Database deployment.

```
rusir@Rusiru-PC MINGW64 /d/LCIT/Term 03/CBD 3324 - Containerization and Container Delivery/Week 05 (master)
$ kubectl get all -n db
NAME                                          READY   STATUS    RESTARTS   AGE
pod/wp-database-deployment-84ff5bcdd7-7xgjv   1/1     Running   0          31m

NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/wp-database-service   ClusterIP   10.43.176.202   <none>        3306/TCP   31m

NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wp-database-deployment   1/1     1            1           31m

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/wp-database-deployment-84ff5bcdd7   1         1         1       31m
```

Figure 3: ns: db.

The values.yaml file for the database is:

Code | Blame | 19 lines (19 loc) · 341 Bytes

```yaml
db:
  name: wp-database
  labels:
    id: c0918066
    app: wp-database-app
    env: dev
  image:
    repository: mariadb
    tag: "10.6.4-focal"
    pullPolicy: IfNotPresent
  replicas: 1
  configmap:
    MYSQL_DATABASE: wordpress
    MYSQL_USER: wordpress
    APP_ENV: dev
  service:
    type: ClusterIP
    port: 3306
    targetPort: 3306
```

Figure 4: Database - values.yaml

5. WordPress Deployment.

Next, I deployed the WordPress application.

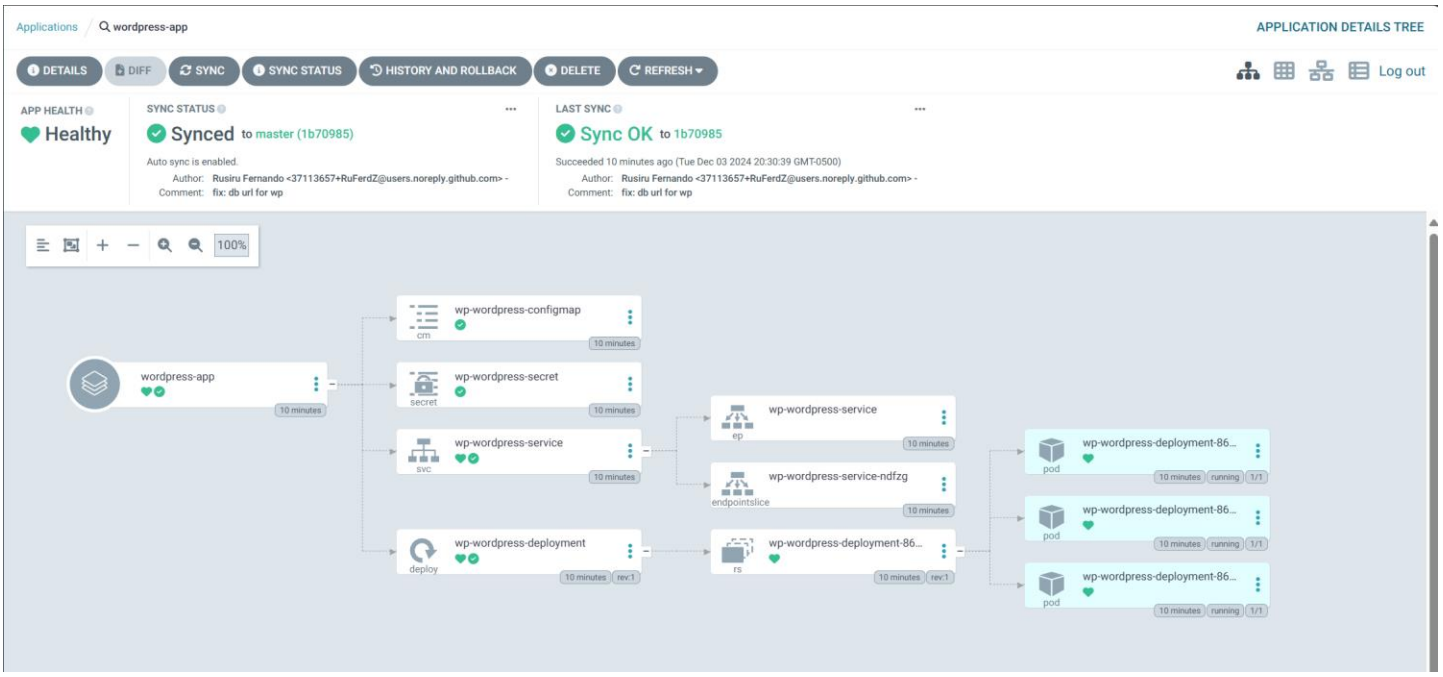

Figure 5: WordPress application deployment.

```
rusir@Rusiru-PC MINGW64 /d/LCIT/Term 03/CBD 3324 - Containerization and Container Delivery/Week 05 (master)
$ kubectl get all -n wp
NAME                                          READY   STATUS    RESTARTS   AGE
pod/wp-wordpress-deployment-8699dd5c6b-gpknv   1/1     Running   0          31m
pod/wp-wordpress-deployment-8699dd5c6b-mnzlq   1/1     Running   0          31m
pod/wp-wordpress-deployment-8699dd5c6b-q88h5   1/1     Running   0          31m

NAME                          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/wp-wordpress-service   ClusterIP   10.43.247.96   <none>        80/TCP    31m

NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wp-wordpress-deployment   3/3     3            3           31m

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/wp-wordpress-deployment-8699dd5c6b   3         3         3       31m
```

Figure 6: ns: wp.

The values.yaml file for the WordPress application is:

Code Blame 20 lines (20 loc) · 411 Bytes

```yaml
wp:
  name: wp-wordpress
  labels:
    id: c0918066
    app: wp-wordpress-app
    env: dev
  image:
    repository: wordpress
    tag: "latest"
    pullPolicy: Always
  replicas: 3
  configmap:
    WORDPRESS_DB_HOST: "wp-database-service.db.svc.cluster.local"
    WORDPRESS_DB_NAME: "wordpress"
    WORDPRESS_DB_USER: "wordpress"
    APP_ENV: "dev"
  service:
    type: ClusterIP
    port: 80
    targetPort: 80
```

Figure 7: WordPress - values.yaml.

Tested the deployment via port forwarding:

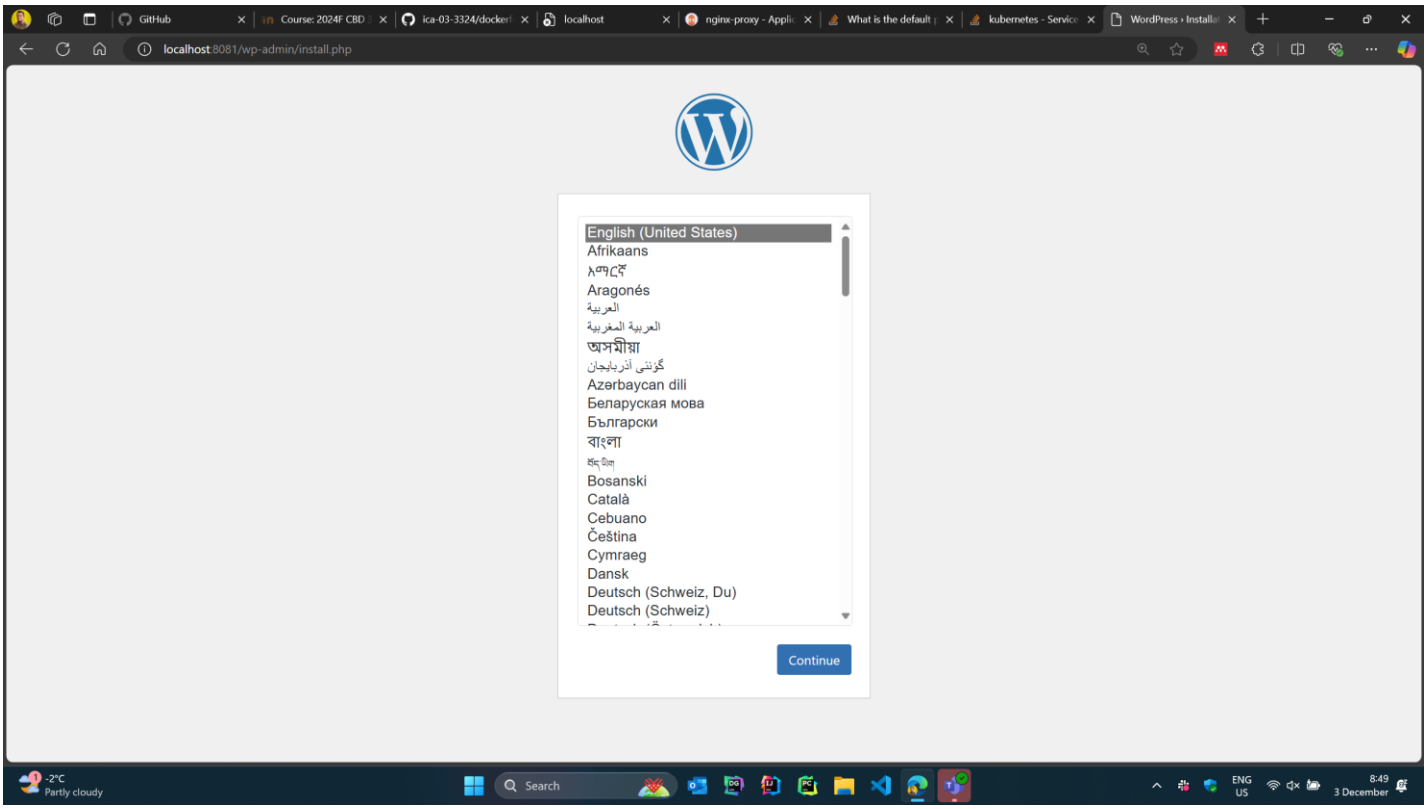

Figure 8: Wordpress Application.

6. 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

Code Blame 15 lines (12 loc) · 439 Bytes

```
server {

    listen 80;

    location / {
        proxy_pass http://wp-wordpress-service.wp.svc.cluster.local/;

        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Port $server_port;
        proxy_set_header X-Forwarded-Host $host;
    }
}
```

Figure 9: Nginx.conf file.

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

Code Blame 9 lines (5 loc) · 151 Bytes

```
FROM nginx:latest

RUN apt update && apt install -y nano

COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Figure 10: Dockerfile.

Next, I deployed the Nginx Proxy:

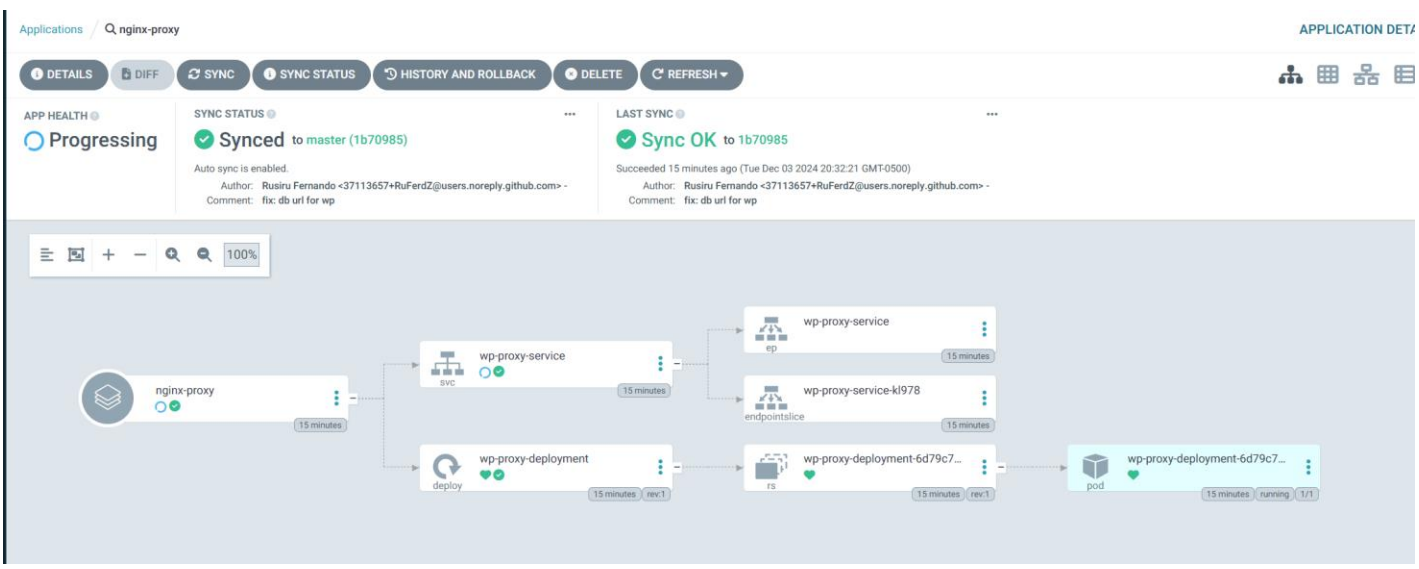

Figure 11: Nginx-proxy deployment.

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.*

```
rusir@Rusiru-PC MINGW64 /d/LCIT/Term 03/CBD 3324 - Containerization and Container Delivery/Week 05 (master)
$ kubectl get all -n proxy
NAME                                      READY   STATUS    RESTARTS   AGE
pod/wp-proxy-deployment-6d79c75bd7-xfqsf   1/1     Running   0          31m

NAME                       TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
service/wp-proxy-service   LoadBalancer   10.43.232.62   <pending>     80:31451/TCP   31m

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wp-proxy-deployment   1/1     1            1           31m

NAME                                            DESIRED   CURRENT   READY   AGE
replicaset.apps/wp-proxy-deployment-6d79c75bd7   1         1         1       31m
```

Figure 12: ns: proxy.

The values.txt file for nginx deployment:

Code Blame 15 lines (15 loc) · 262 Bytes

```yaml
nginx:
  name: wp-proxy
  labels:
    id: c0918066
    app: wp-proxy-app
    env: dev
  image:
    repository: ruferdz/ica3-nginx-proxy
    tag: "latest"
    pullPolicy: IfNotPresent
  replicas: 1
  service:
    type: LoadBalancer
    port: 80
    targetPort: 80
```

Figure 13: Nginx Proxy - values.yaml.

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

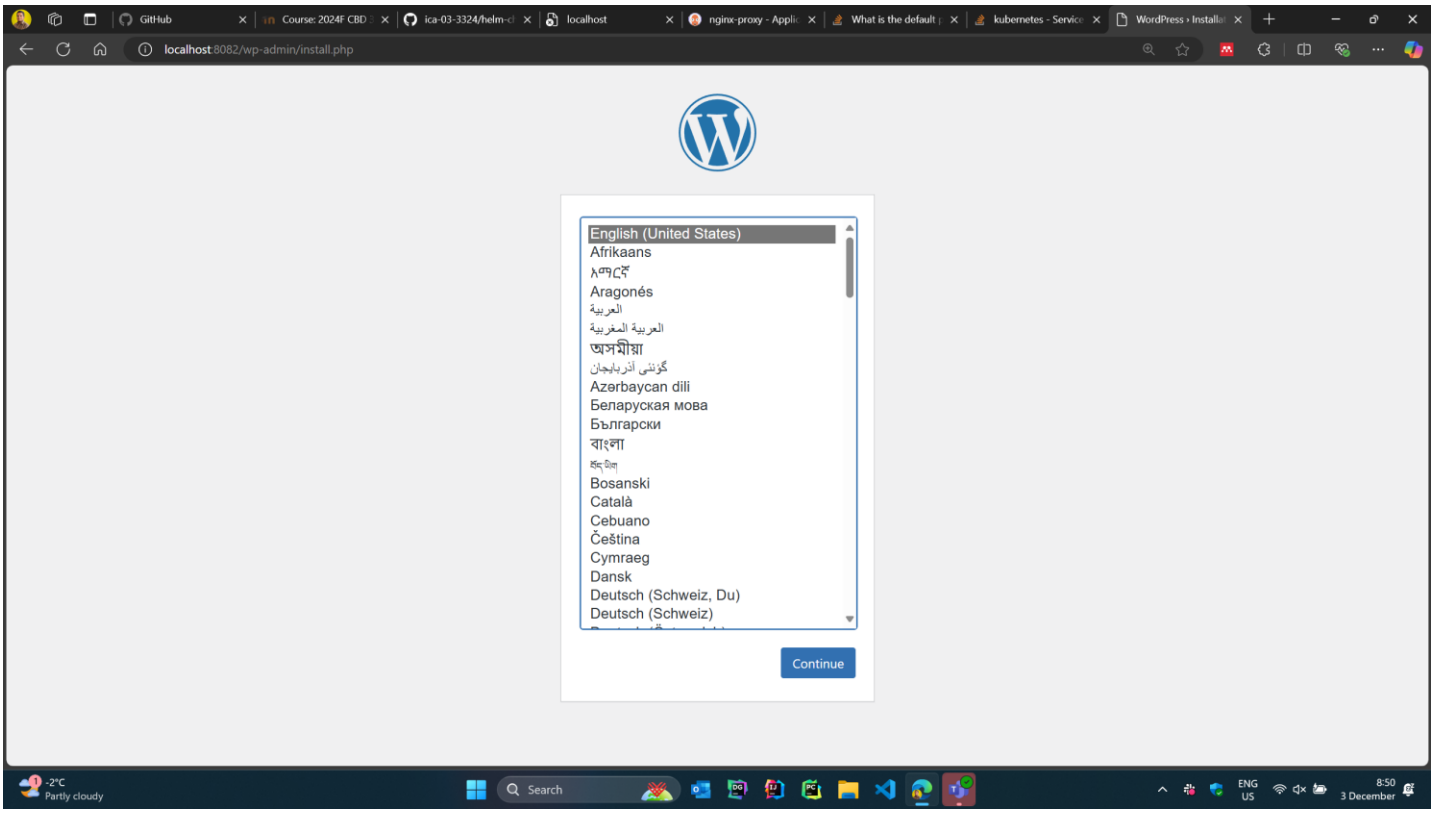

Figure 14: WordPress application via Nginx Proxy.

7. 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.

8. 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..).