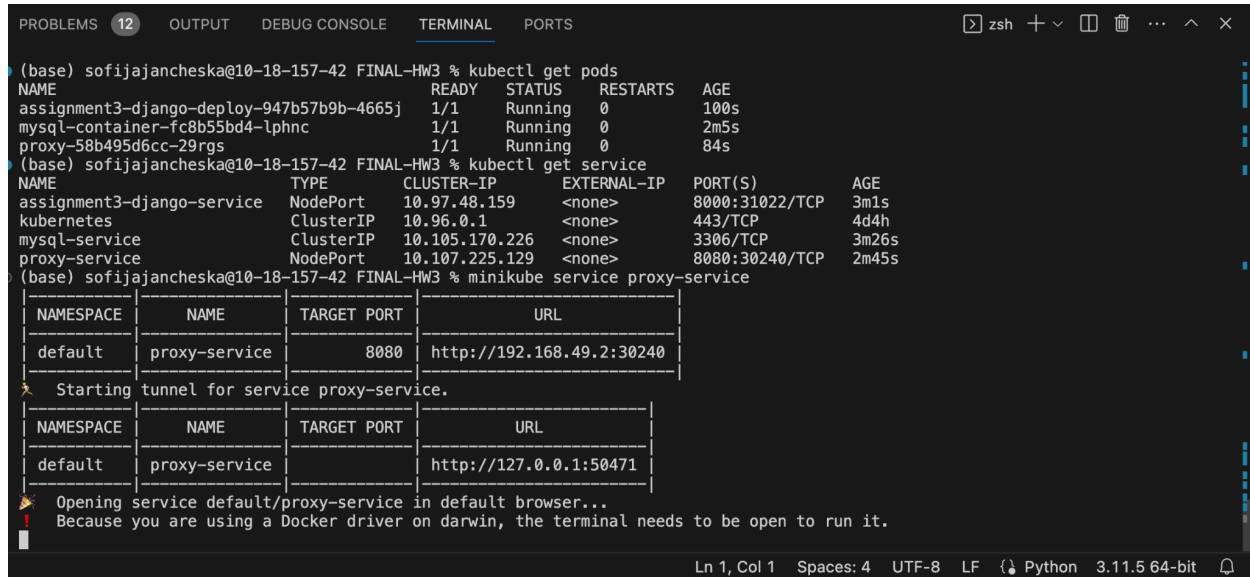


Lab 3: Overall Write-up

Part 0: Setting up Your Environment

All the installations and rundown of files went smoothly; the screenshot below shows the three pod entities with status running and four service entities. I was also able to connect to the site using the command: `minikube service proxy-service`.



```
(base) sofiyajancheska@10-18-157-42 FINAL-HW3 % kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
assignment3-django-deploy-947b57b9b-4665j   1/1     Running   0           100s
mysql-container-fc8b55bd4-lphnc             1/1     Running   0           2m5s
proxy-58b495d6cc-29rgs                      1/1     Running   0           84s
(base) sofiyajancheska@10-18-157-42 FINAL-HW3 % kubectl get service
NAME                                TYPE               CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
assignment3-django-service          NodePort           10.97.48.159     <none>            8000:31022/TCP    3m1s
kubernetes                         ClusterIP          10.96.0.1        <none>            443/TCP           4d4h
mysql-service                      ClusterIP          10.105.170.226   <none>            3306/TCP          3m26s
proxy-service                      NodePort           10.107.225.129   <none>            8080:30240/TCP    2m45s
(base) sofiyajancheska@10-18-157-42 FINAL-HW3 % minikube service proxy-service
+-----+-----+-----+-----+
| NAMESPACE | NAME      | TARGET PORT | URL              |
+-----+-----+-----+-----+
| default   | proxy-service | 8080        | http://192.168.49.2:30240 |
+-----+-----+-----+-----+
Starting tunnel for service proxy-service.
+-----+-----+-----+-----+
| NAMESPACE | NAME      | TARGET PORT | URL              |
+-----+-----+-----+-----+
| default   | proxy-service |            | http://127.0.0.1:50471 |
+-----+-----+-----+-----+
Opening service default/proxy-service in default browser...
! Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

Part 1: Securing Secrets

First of all, I checked the yaml files in the GiftcardSite folder and the db folder of our lab assignment. I was able to identify 2 files that need to be secured using the secrets functionality of our previously installed kubernetes and minikube.

1. `MYSQL_ROOT_PASSWORD.txt` contains the root password of the database. I was able to find this in two yaml files within the db folder, more specifically in the k8 subfolder of db. The two files here that contained the root password of the database were: `db-deployment.yaml` and `django-deploy.yaml`.
2. `WEB_SECRET_CODE.txt` contains the django secret key. I was able to locate this secret in the `settings.py` file where the `SECRET_KEY` variable was used. This `settings.py` file was located in the GiftcardSite folder.

For the first `MYSQL_ROOT_PASSWORD`, the root password of the database, I initially created a txt file called `MYSQL_ROOT_PASSWORD` and I put the root password of the database in that file. The file contained "thisisatestthing." I also created a `MYSQL_ROOT_PASSWORD.yaml` file where I made the structure to write base64 encoded text of that password in the secrets in order to extract the password from it. I was able to do this through the following command:

```
- echo -n 'thisisatestthing.' | base64
```

This command outputs the base64 encoded text of the provided password.

I then created the secrets to store the above database password and I was able to do it with the following command:

```
kubectl create secret generic mysql-root-password
--from-file=mysql-root-password=./MYSQL_ROOT_PASSWORD.txt
```

This command basically creates a secret tag with the name MYSQL_ROOT_PASSWORD and adds the password from the file called MYSQL_ROOT_PASSWORD.txt

In order to be able to use this secret in the two yaml files that I described above, I first deleted the actual password from it and added the following lines of code instead:

```
name: MYSQL_ROOT_PASSWORD
valueFrom:
  secretKeyRef:
    name: mysql-root-password
    key: mysql-root-password
```

I did this in both files: db-deployment.yaml from line 21 and django-deploy.yaml from line 30.

For the second secret key, the django's secret key, in the similar way as for the first secret key, I initially created a file named web-secret-key.txt and I stored the value of SECRET_KEY variable that I had found before from the settings.py file. Again in a similar vein for the first secret, I created another secret tag to use this SECRET_KEY value. I did this using the following command:

```
kubectl create secret generic web-secret-key --from-file=web-secret-key=./WEB_SECRET_KEY.txt
```

So to summarize, in this way, I created a secret tag with the name web-secret-key and I added the password from the file web-secret-key.txt which I mentioned that I had created earlier and stored the value there.

After all this, I modified the SECRET_KEY line in the settings.py file. I wanted to add the environment variable reference using the os.environ module. This basically indicates that we are using the SECRET_KEY value from the environment with a secret tag name as WEB_SECRET_KEY.

- SECRET_KEY = os.environ.get('web-secret-key') Then I used this secret in the django-deploy.yaml file where I added the name and valueFrom contents of the secret tag which are same as I had described previously. I also added the following code in the django-deploy.yaml file starting from line 23:

```
name: WEB_SECRET_KEY
valueFrom:
  secretKeyRef:
    name: web-secret-key
    key: web-secret-key
```

To be able to verify the above-created secrets, I ran the following commands:

```
kubectl get secret: this command outputs the above created secrets
```

In order to see these modifications in action and rerun the modified yaml deployment files, we first delete all the currently running pods, and run the driver-docker, build, create and re-deploy the website again using the following commands: kubectl delete pods --all

Part 2

First of all, I deleted all unsafe monitoring of data in our codebase. More specifically, I spotted that there is unwanted monitoring activity in the views.py file within the GiftcardSite folder. I noticed that this has been implemented for all of the functions. I also saw that in the register_view function in that same views.py file, the password is being given in the prometheus monitoring code. The code below prints {pword} (password) and it can be noticed in monitoring. In order to delete it, I simply commented the following code:

```
if pword not in graphs.keys():
    graphs[pword] = Counter(f'counter_{pword}', 'The total number of\' + f'times {pword} was used')
```

```
graphs[pword].inc()
```

Secondly, I made a counter for prometheus in order to expand on the monitoring activity. More specifically, I added a prometheus counter that counts every time we return a 404 error message in the views.py file. These errors are being triggered by database errors. In order to do this successfully, in the views.py file, I added this counter with the following lines of code:

```
graphs['database_error_return_404'] = Counter('database_error_return_404', 'The total number\' + ' of times 404 error message is returned.')
```

After I created the counter, I had to add an incrementation method that increases the total count value each time it encounters a database error 404 message, and I added this in two places in the code with the following line:

```
graphs['database_error_return_404'].inc()
```

Thirdly, I added Prometheus with helm. So firstly, I installed the helm package manager with choco in powershell. Afterwards, I installed Prometheus with the following commands:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

```
helm repo update
```

```
helm install prometheus prometheus-community/prometheus
```

```
helm install -f prometheus.yaml prometheus prometheus-community/prometheus
```

In order to verify the active Prometheus pods and services, I ran the the commands below and checked their states:

```
minikube service list
```

```
kubectl get pods
```

I got the following outputs through which I verified the service: all pods were properly running as shown below.

```
assignment3-django-deploy: Running
```

```
mysql-container: Running
```

```
prometheus-alertmanager: Running
```

```
prometheus-kube-state-metrics: Running
```

```
prometheus-node-exporter: Running
```

```
prometheus-pushgateway: Running
```

```
prometheus-server: Running
```

```
proxy: Running
```

Afterwards, I got the configmaps of Prometheus and I used the following commands for this:

kubectl get cm → this command shows the configmaps prometheus-alertmanager and prometheus-server in the following way:

```
kube-root-ca.crt
```

```
prometheus-alertmanager
```

```
prometheus-server
```

Afterwards, I added Prometheus to the configmaps to the cluster and the giftcard website with the proxy service. I did this by updating the "prometheus-server" configmap with the following command:

```
kubectl edit configmap prometheus-server
```

Next, upon opening the configmap prometheus-server, I had to modify the access of this from localhost:9090 to proxy-service:8080 by which we can use both the giftcard site and the cluster. I did this modification with "static_configs: targets:" in configmap. Then I exported the data from both configmaps to yaml file. You can find all the files that I am referring to in my part2 folder. As described above, I got

the output of both configmaps by running the respective commands listed below and I then copy pasted the output in two new yaml files.

For the prometheus-alertmanager.yaml file: `kubectl get configmap prometheus-alertmanager -o yaml`

For the prometheus-server.yaml file: `kubectl get configmap prometheus-server -o yaml`

Lastly, I began with port forwarding to 8080 port by running the command below. Then I went to our giftcard site 127.0.0.1:8080 to start Prometheus and monitoring. There we get the prometheus portal where we can start monitoring. The command which I mentioned goes as following:

`kubectl port-forward deployment/prometheus-server 8080:8080`