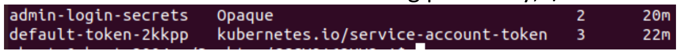
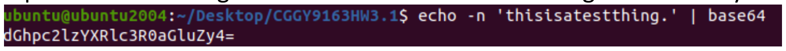
Part 0: Installed the following software to start part 1, Docker repository, Docker Engine, kubectl, minikube and the minikube docker driver.
Part 0.1: Following the assignment, there were no issue with starting minikube and building the docker files provided within the assignment documentation. Verified all pods and services and checked if I was able to connect to the proxy which was a success.
Part 1: Looking at the Kubernete documentation. It stated that you can build secrets as an environment variable, in a secret file and inside the image file and able to see the type of secrets trough kubectl command line. Using the kubectl get secrets command I was able to find 2 types of secrets that are shown, an Opaque which is called by a secret file which is store within ./GiftcardSite/k8/django-admin-pass-secret.yaml and a service account token. I observed admin-login-secrets contain secrets that allow authenticating the web project as admin when looking at the secret file and the file that is being passed by, ./GiftcardSite/k8/django-deploy.yaml.



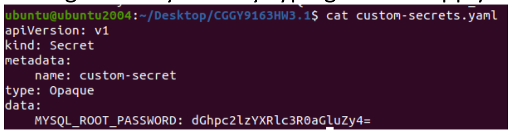
```
admin-login-secrets   Opaque                                2      20m
default-token-2kkpp   kubernetes.io/service-account-token   3      22m
```

I observed a quite a lot of yaml file that may contain secret file and exposable value such as password that need to be called using secrets. Within the main directory of the project I used find . | grep yaml command to check all the location of the yaml files. For each of the file I analyzed any sensitive information that should have been checked. I notice the following path that contained sensitive information ./db/k8s/db-deployment.yaml , ./db/k8/db-deployment.yaml, and /GiftcardSite/k8/django-deploy.yaml. It seemed the the sql root password was being exposed with the value, thisisattesting., which needs to be secure.
To start the securing the sensitive value, I copied the password and use the echo command to take the password input and converting to base64 without error checking instructed by the documentation.



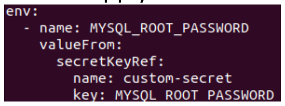
```
ubuntu@ubuntu2004:~/Desktop/CGGY9163HW3.1$ echo -n 'thisisatestthing.' | base64
dGhpc2lzYXRlc3R0aGluZy4=
```

Create a second opaque type secret file and copy paste the content from the admin-login-secrets and change certain attributes to create a custom-secrets.yaml. After creating the file, I had to apply the secret to kubernete management system by typing kubectl apply -f custom-secrets.yaml

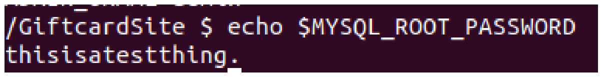


```
ubuntu@ubuntu2004:~/Desktop/CGGY9163HW3.1$ cat custom-secrets.yaml
apiVersion: v1
kind: Secret
metadata:
    name: custom-secret
type: Opaque
data:
    MYSQL_ROOT_PASSWORD: dGhpc2lzYXRlc3R0aGluZy4=
```

Looking at the expose yaml files and replace it with the secret by using valueFrom and referring to custom-secret file.
After finishing updating one of the expose, I applied the Kubernetes and deleted the pods with the kubectl delete pods --all so the pod can respawn itself again since there times the pods doesn't update frequently when I do a kubectl apply.



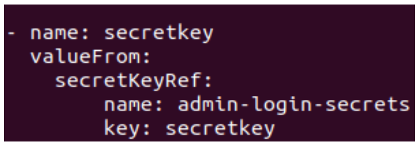
```
env:
  - name: MYSQL_ROOT_PASSWORD
    valueFrom:
      secretKeyRef:
        name: custom-secret
        key: MYSQL_ROOT_PASSWORD
```

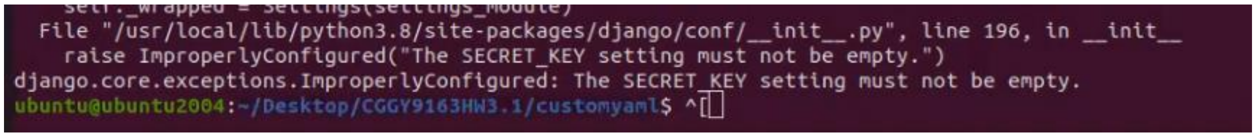
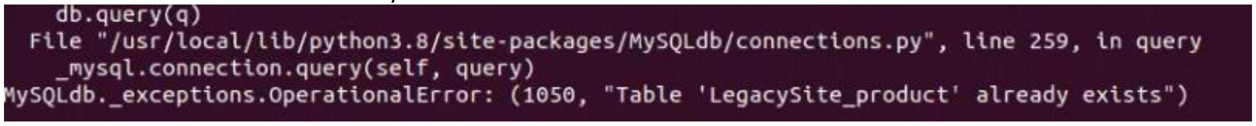
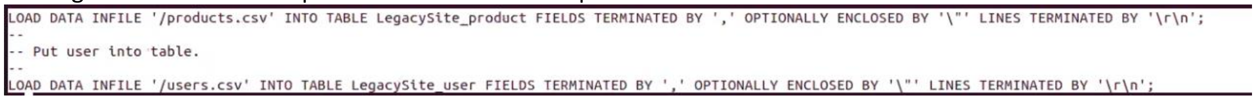


```
/GiftcardSite $ echo $MYSQL_ROOT_PASSWORD
thisisatestthing.
```

Checking if I'm apply to open the website by executing the proxy service and used the following command, Kubectl exec -it djanopodname sh to check if echo the environment variable. If the environment variables doesn't output the password then the implementation did not work. Since it worked, I was confident to edit the other yaml files that were exposed just like the previous file but did not needed to create secret file since I can referred the key from the custom-secret.yaml file.

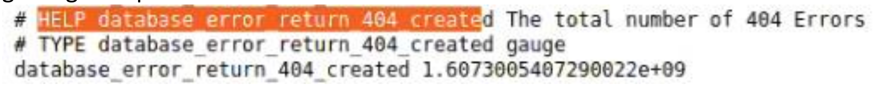
Looking at the Gift card file project I notice one sensitive data that needed to be called using Kubernetes secrets. The setting.py file had a SECRET_KEY that was being exposed. What I needed to do was encode the secret key, added the key and value and putting within the django-admin-pass-secret.yaml, kind of like the previous fixes but needed a place in the yaml deployment to refer to the secret key. After adding the key, I went inside to my django file and echo the secret key for the web project to see if the value of the key. Then I went inside the project directory and removed the SECRET_KEY data and replace with os.environ.get('secretkey') which will output the value of the secreykey. Since I made changes within the file, I had to rebuild the django project and delete the django pods to notice changes within my Django container application. Once that was completed, I tested the status of my containers and run the applications to see if there were any issues implementing the secret keys and checked if it ran any error.


```
- name: secretkey
  valueFrom:
    secretKeyRef:
        name: admin-login-secrets
        key: secretkey
```
Part 2
Looking at the instruction it looks like it wants us to do a migration and seeding with Kubernete jobs. Looking at the documentation Kubernete jobs handle task within Kubernete container. We can use the Django library to do the migrations which is python manage.py migration. Let's make this command to execute on the gift card application in the Kubernetes by create the Kubernete yaml file type job. Look at one link for stackoverflow that contain a template Kubernete Job file. https://stackoverflow.com/questions/60061241/commands-passed-to-a-kubernetes-job-and-pod.
I copied the content within stackoverflow and created a migratemodel.yaml and made few changes I need. Within the Yaml file I edit the image to point to the Giftcard I know that that the secret_key was empty and looked like the environment variables needed to be added. Copy djang-deploy.yaml content for the environment variables and added to the migratemodel. yaml.


```
    self._wrapped = Settings(settings_module)
  File "/usr/local/lib/python3.8/site-packages/django/conf/__init__.py", line 196, in __init__
    raise ImproperlyConfigured("The SECRET_KEY setting must not be empty.")
django.core.exceptions.ImproperlyConfigured: The SECRET_KEY setting must not be empty.
ubuntu@ubuntu2004:~/Desktop/CGGY9163HW3.1/customyaml$ ^[
```
Deleted the yaml file and replied it again and looking like the pod ran successful. Checked the logs and see migrate command was executed successfully.

```
    db.query(q)
  File "/usr/local/lib/python3.8/site-packages/MySQLdb/connections.py", line 259, in query
    _mysql.connection.query(self, query)
MySQLdb._exceptions.OperationalError: (1050, "Table 'LegacySite_product' already exists")
```
We can see that a migrate command was initiated to a table that already have the same model. If we change the model, it shows a success migrate.
Seeding,
When seeding the database, we need to reimport the data into the database. First, we want to know where in the application is data being imported. In the docker files we can see how the db is being set up. In the docker file there is an entrpoint.sh that is probably seeding the database. looking inside the code and at the end of the code it is using a command that import the user.csv and the products.csv into the database.

```
LOAD DATA INFILE '/products.csv' INTO TABLE LegacySite_product FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '\"' LINES TERMINATED BY '\r\n';
--
-- Put user into table.
--
LOAD DATA INFILE '/users.csv' INTO TABLE LegacySite_user FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '\"' LINES TERMINATED BY '\r\n';
```
We can do some testing by executing inside the db container using kubctl exec -it dbpod sh, after going in went to the path of the sqlscript and start the mysql process and deleted a table for testing the commands manually to later implemented within the Yaml file. The db process it uses to db commands are mysql. Running mysql and getting inside the giftcarddb was challenging to get in until you need to pass in the proper credentials that db-deployment use to access the giftcarddb using ***mysql -u root -p "thisisatestthing." GiftcardSiteDB.*** After getting in wanted to delete the product table and see if I can use the last table import command on a separate script call seed.sql. Once the file was created ran the command, ***mysql --user=root --password=${MYSQL_ROOT_PASSWORD} --database=${MYSQL_DATABASE} --host=mysql-service -f < /docker-entrypoint-initdb.d/seed.sql.*** When I used the command, I got an error saying that it is already in the database so what I did was delete the database and run the command and data was being push to the database. I replaced the following command, LOAD DATA INFILE '/users.csv' IGNORE INTO TABLE LegacySite_user FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '\"' LINES TERMINATED BY '\r\n'; This command would basically still to the import but ignore the following rows that look like duplicate to the database. This can solve if user.csv or product.csv was updated with new content and would like that to import the file with new data.

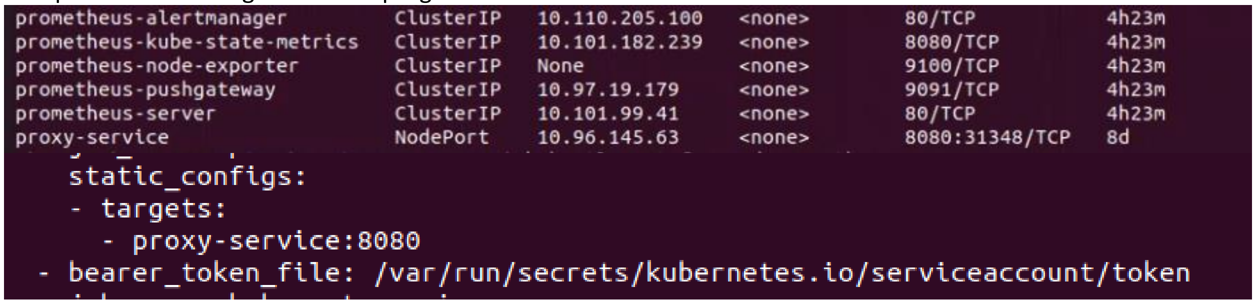
Part 3

Looking at the views.py, I found a bunch of unsafe monitors in the code. It seems it monitored the register posts, login posts, card buy posts, card gift posts, and use card posts. I feel that we should remove all the monitoring keys within our metrics. We do not want the attacker looked at these metrics because it is sensitive data that displays the content of the website and can use as statistics to look at the popularity of the website and I feel that is unsafe to even log that publicly. Within the view.py file it is using a Prometheus library that we can create a key and store data in the key whenever the graph function is being called for example graph.inc(). After removing the unwanted monitoring, I added a key, graphs['database_error_return_404'] in the beginning in the line and added its subclass of graphs['database_error_return_404'].inc near lines that have 404 return error. Whenever we get a 404 error it would increment any 404 return request.
After saving the code, I had to rebuild the image and tested to see if I saw anything related to the metrics when I went into the giftcard metric url . Here I saw the 404 being logged. So far nothing is being counted since I have not getting a response of a 404 error.



```
# HELP database_error_return_404_created The total number of 404 Errors
# TYPE database_error_return_404_created gauge
database_error_return_404_created 1.6073005407290022e+09
```

To interacting with Kubernetes, I must install helm with an installation of Prometheus. Luckily installing Prometheus with the custom yaml file that is associated with. Following the step using helm with Prometheus by reading a article installing helm and Prometheus, https://www.redhat.com/sysadmin/installing-prometheus. After installing Prometheus pods I needed to edit the Prometheus configmap to add the target to my proxy-service:8080 that points to the the gift card site program.



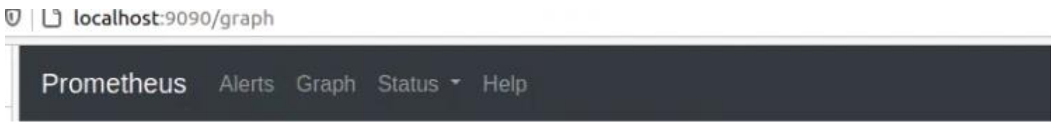
```
prometheus-alertmanager         ClusterIP   10.110.205.100   <none>        80/TCP          4h23m
prometheus-kube-state-metrics   ClusterIP   10.101.182.239   <none>        8080/TCP        4h23m
prometheus-node-exporter        ClusterIP   None             <none>        9100/TCP        4h23m
prometheus-pushgateway          ClusterIP   10.97.19.179     <none>        9091/TCP        4h23m
prometheus-server               ClusterIP   10.101.99.41     <none>        80/TCP          4h23m
proxy-service                   NodePort    10.96.145.63     <none>        8080:31348/TCP  8d
```

```
    static_configs:
    - targets:
      - proxy-service:8080
  - bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
```

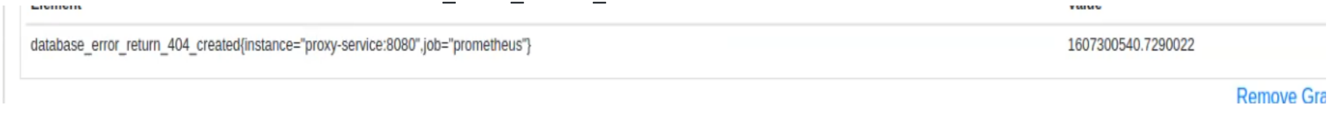
After setting the config map I communicated with the prometheus by connecting to the port 9090 which opened the website.

kubectl port-forward deployment/prometheus-server 9090

here we can see a website of prometheus, to see the metrics that was added within prometheus, we need to click graph and execute the values within the drop down menu called, database_error_return_404 that was created within the Giftcardsite/Legacysite/view.py



Here were can see the value of database_error_return_404 and see the value of when it was created.



```
database_error_return_404_created{instance="proxy-service:8080",job="prometheus"}      1607300540.7290022
```