Implementation Document

Python Flask App with Helm Package deployment

v1.0.0

The document contains the detailed instructions to build and package docker containers with a python flask app and to use Helm for packaging the entire environment consisting of a postgres DB and a python flask app.

Below are the features of the python app:

- The python flask service responds to the url http://host:port/?n=x with a value equal to n*n.
- The service sends a mail to the <test@domain.com> email ID, saves the information of the requestor with a timestamp in a database table, ban the IP address of requester and returns a response code 444, once service is called on the url : http://host:port/blacklisted

1. *Python Flask service*

1.1 Packages required for service as prerequisites

The python flask app requires below python packages to be defined as a prerequisites :
- *Flask :* Contains library for flask framework
- *Flask-Mail :* Required to work with a email testing service called mailtrap.io
- *flask-ipban :* used for IP ban for python app
- *datetime :* Timestamp requirement for db entries.
- *SQLAlchemy :* To create the postgres connections from flask app
- *psycopg2 :* PostgreSQL database adapter

1.2 Prepare the Dockerfile:

Below is the Dockerfile prepared for containerizing the flask app:
We used the base image layer of *python:3.9.11* as it's pre-baked with all the necessary packages needed to execute our python flask app!

Pip3 is used to install the dependencies described in *section 1.1* and
the service is exposed on a custom port, in our case 5004.

```
FROM python:3.9.11

WORKDIR /app
COPY . /app

RUN pip3 --no-cache  install Flask Flask-Mail flask-ipban datetime
SQLAlchemy psycopg2

EXPOSE 5004
ENTRYPOINT ["python3"]
CMD ["server.py"]
```

1.3 Package the source code along with dockerfile

1.3.1 Creating a new docker image

Our project folder structure looks like :

```
--package/
        --server.py
        --Dockerfile
--helm/
        --chart/
                --templates/
                        - <contains all the required templates>

        - values.yaml

        - Chart.yaml
```

Run the below command to create a fresh docker image:

```
docker build -t flask-blacklist-svc:v1 package
```

Where *flask-blacklist-svc* is the image name and *v1* is the image tag! We will use this image information later in our helm *values.yaml* file

1.3.2 Packing the service with Helm

Helm packaging was used for the project to streamline the packaging and deployment of the entire environment in any k8s cluster.

Various Helm templates were created to facilitate the resources needed to be packaged as part of the entire project.

The entire project is divided into two components, *Postgres DB* related resources and *Python flask app* related resources.

Below are the various template used :

Flask-deployment.yaml : used to deploy the Kubernetes deployment for the flask app!

Flask-service.yaml : Describes the service definition for the flask app to be exposed as NodePort service on a defined Port.

postgres-deployment .yaml: Created the pods required to host postgres db to save the access denied states per users. This deployment makes use of a persistent volume claim to store the state of the db as mounted on a defined path!

Postgres-svc.yaml : This service exposes the postgres service to be used by the python app to make a db connection.

Db-configmap.yaml : Creates the configmap for db credentials to be used by the postgres db service.

Postgres-storage.yaml : This template creates a pair of persistent volume and persistent volume claim to be used by the postgres deployment.

Before Deploying the helm charts the prerequisite is to create a namespace :

```
Kubectl create ns microservices
```

Once namespace is created, execute the below command to create a new helm release, make sure, the present directory is the one where *Chart.yaml* and *values.yaml* is saved

```
helm install  flask-blacklist . -n microservices
```

Where *flask-blacklist* is the release name and **.** indicates the directory that includes the values.yaml and Chart.yaml file. The output of the command will be :

```
NAME: flask-blacklist
LAST DEPLOYED: Sun Mar 20 22:28:26 2022
NAMESPACE: microservices
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

1.3.3 Kubernetes resources

The helm deployment will create 2 pods, one for the python app (flask-blacklist-svc) and another for the postgres db (postgres), and along with these pods, two kubernetes nodePort service to expose these workloads.

```
# Kubectl get all

NAME                                             READY   STATUS    RESTARTS      AGE
pod/flask-blacklist-deployment-f7467c8d8-54hbr   1/1     Running   0             96m
pod/postgres-db-55ccfcbf6d-6qtx8                 1/1     Running   0             96m

NAME                          TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/flask-blacklist-svc   NodePort   10.108.187.66   <none>        5004:30911/TCP   96m
service/postgres              NodePort   10.109.98.124   <none>        5432:31111/TCP   96m

NAME                                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/flask-blacklist-deployment   1/1     1            1           96m
deployment.apps/postgres-db                  1/1     1            1           96m

NAME                                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/flask-blacklist-deployment-f7467c8d8   1         1         1       96m
replicaset.apps/postgres-db-55ccfcbf6d                 1         1         1       96m
```

The services are available on their respective ports, 5004 and 5432.

1.3.4 Testing the deployments

To test the functionality of the app, we can simply do a port forwarding to map the container ports to local machines.

Note : We have used minikube as our platform of choice for k8s testing.

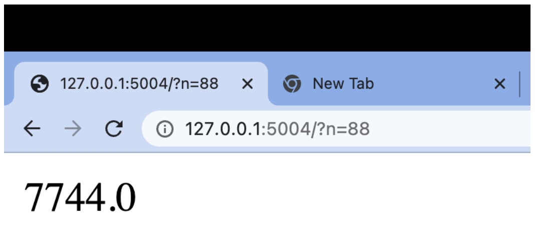
```
kubectx minikube
Switched to context "minikube".

kubectl port-forward flask-blacklist-deployment-54hbr   5004:5004
```

Once successfully port forwarded, open a browser to check the functionality by using below url :

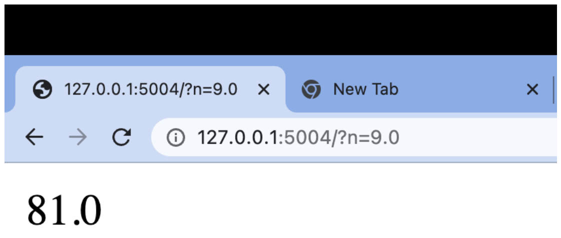
http://127.0.0.1:5004/?n=88

*It should return n*n, which is 88*88!*


127.0.0.1:5004/?n=88

127.0.0.1:5004/?n=88

```
7744.0
```


127.0.0.1:5004/?n=9.0

127.0.0.1:5004/?n=9.0

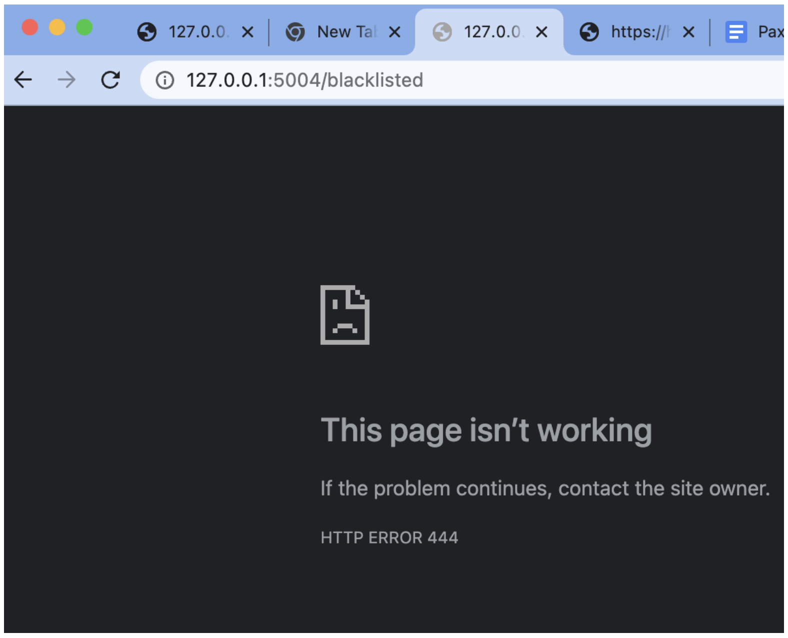
```
81.0
```

We used float type value as input thus even if the user enters a decimal value, the service still responds!

And the second functionality is to hit the below url and get a 444 response code with entry in postgres db and a mail is sent to test@domain.com , and we used a mail testing service to send the test mail, service is called https://mailtrap.io .

Note : Create a separate free test account to test the mail functionality. Use this guide to configure the flask service to work with mailtrap.io
https://mailtrap.io/blog/flask-email-sending/

Url : http://127.0.0.1/blacklisted
Output:

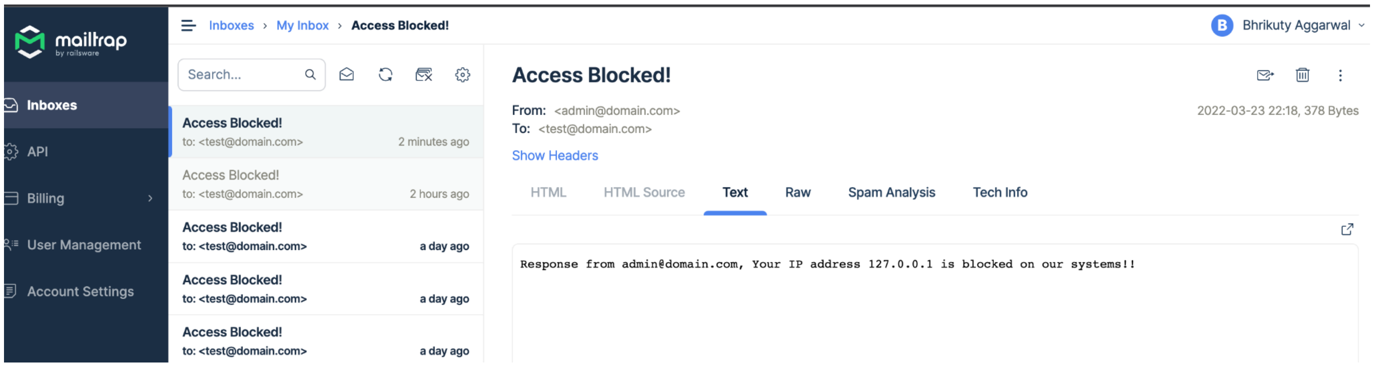

127.0.0.1:5004/blacklisted

This page isn't working

If the problem continues, contact the site owner.

HTTP ERROR 444

Email sent to the testing platform mailtrap.io:



Access Blocked!
to: <test@domain.com> 2 minutes ago

Access Blocked!
to: <test@domain.com> 2 hours ago

Access Blocked!
to: <test@domain.com> a day ago

Access Blocked!
to: <test@domain.com> a day ago

Access Blocked!
to: <test@domain.com> a day ago

Access Blocked!

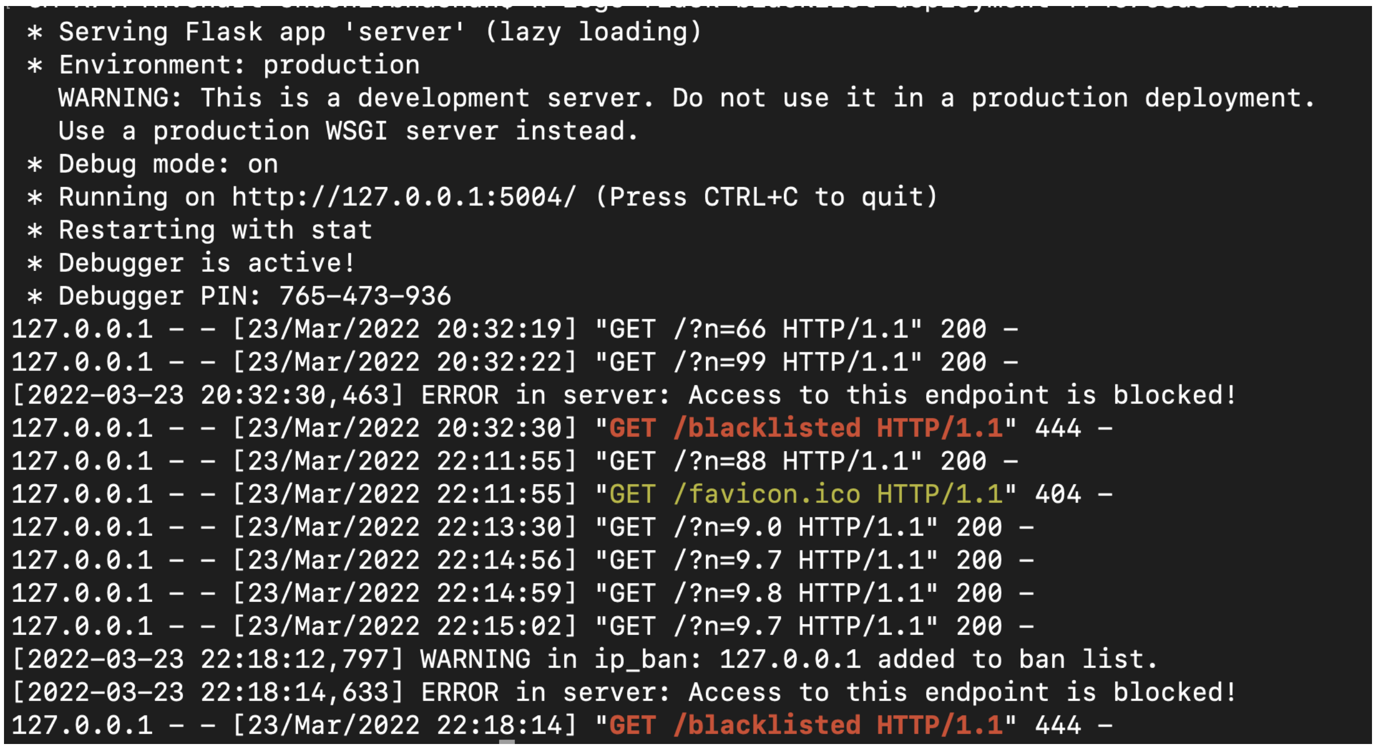
From: <admin@domain.com>
To: <test@domain.com>

2022-03-23 22:18, 378 Bytes

```
Response from admin@domain.com, Your IP address 127.0.0.1 is blocked on our systems!!
```

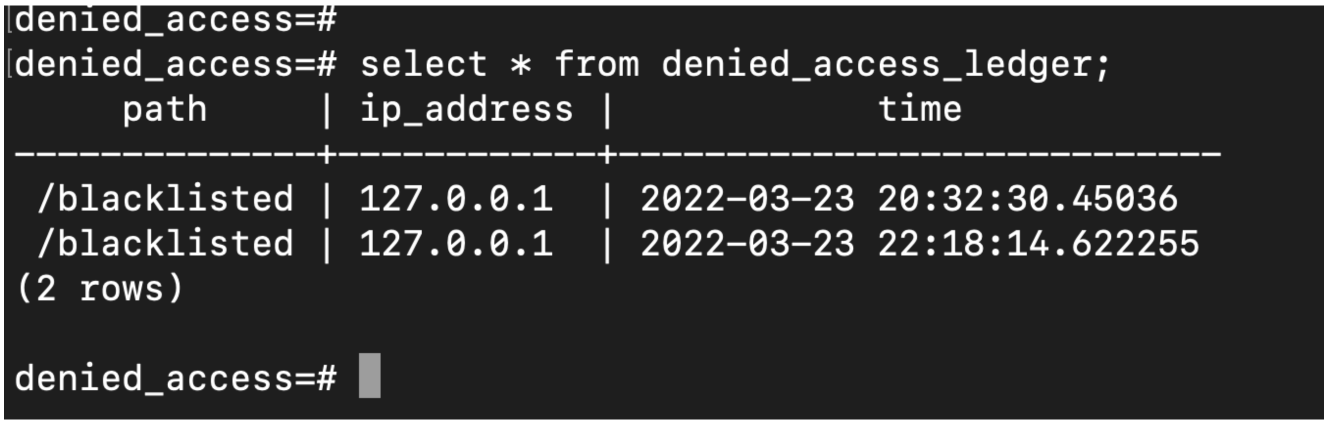
Below are the Logs from pod :

- Server logging is enabled to log Error and response codes with 444.
- Ip Ban Warning



```
 * Serving Flask app 'server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5004/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 765-473-936
127.0.0.1 - - [23/Mar/2022 20:32:19] "GET /?n=66 HTTP/1.1" 200 -
127.0.0.1 - - [23/Mar/2022 20:32:22] "GET /?n=99 HTTP/1.1" 200 -
[2022-03-23 20:32:30,463] ERROR in server: Access to this endpoint is blocked!
127.0.0.1 - - [23/Mar/2022 20:32:30] "GET /blacklisted HTTP/1.1" 444 -
127.0.0.1 - - [23/Mar/2022 22:11:55] "GET /?n=88 HTTP/1.1" 200 -
127.0.0.1 - - [23/Mar/2022 22:11:55] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [23/Mar/2022 22:13:30] "GET /?n=9.0 HTTP/1.1" 200 -
127.0.0.1 - - [23/Mar/2022 22:14:56] "GET /?n=9.7 HTTP/1.1" 200 -
127.0.0.1 - - [23/Mar/2022 22:14:59] "GET /?n=9.8 HTTP/1.1" 200 -
127.0.0.1 - - [23/Mar/2022 22:15:02] "GET /?n=9.7 HTTP/1.1" 200 -
[2022-03-23 22:18:12,797] WARNING in ip_ban: 127.0.0.1 added to ban list.
[2022-03-23 22:18:14,633] ERROR in server: Access to this endpoint is blocked!
127.0.0.1 - - [23/Mar/2022 22:18:14] "GET /blacklisted HTTP/1.1" 444 -
```

And finally the database entry with the detail of the blocked user :



```
denied_access=#
denied_access=# select * from denied_access_ledger;
     path     | ip_address |            time
--------------+------------+----------------------------
 /blacklisted | 127.0.0.1  | 2022-03-23 20:32:30.45036
 /blacklisted | 127.0.0.1  | 2022-03-23 22:18:14.622255
(2 rows)

denied_access=#
```