GLM Model Deployment with FastAPI, Docker, Kubernetes, and GitLab

Overview and project goal

The purpose of this project is to deploy a "segmentation" model for the marketing department. The outcome of the model prediction is whether a customer purchased a product(y=1) or not (y=0). By deploying this model, we can help the marketing department decide which customers receive an advertisement for the product.

This project demonstrates the deployment of a Generalized Linear Model (GLM) using the FastAPI framework, Docker containers, Kubernetes orchestration, and GitLab for Continuous Integration/Continuous Deployment (CI/CD). The GLM model serves as a web API, providing predictions based on input data. In the end, we also discuss the performance testing to identify bottlenecks, and various options to outcome it.

Prerequisites

Ensure the following tools are installed:

- Docker
- Kubernetes
- Minikube
- GitLab CI/CD

Project Structure

- `app/`: Contains the FastAPI application code.
- `app/model/`: Holds the pre-trained GLM model (pickle file).
- `kubernetes/`: Includes Kubernetes deployment and service YAML files.
- `tests/`: Holds unit tests for the FastAPI application.
- `gitlab-ci.yml`: GitLab CI/CD pipeline configuration file.
- `output/output_test.json`: prediction output of 10000 rows of test data
- `README.md`: This documentation file.

Getting Started

1. Clone this repository:

```
$ git clone https://github.com/ZCai25/glm-fastapi-app.git
```

2. Navigate to the project directory:

```
$ cd glm-fastapi-app
```

3. Follow the instructions in each directory to deploy the GLM model locally or in a Kubernetes cluster.

End-to-End Process

1. **Docker Containerization:**

   - The Dockerfile (`Dockerfile`) specifies the environment and dependencies for running the FastAPI application.
   - Docker image is built using the `docker build` command, then you can build by running the `run_api.sh`
     - From the project directory, build the container with a tag 1.0

       ```
       $ docker build -t glm-fast-api:1.0 .
       ```

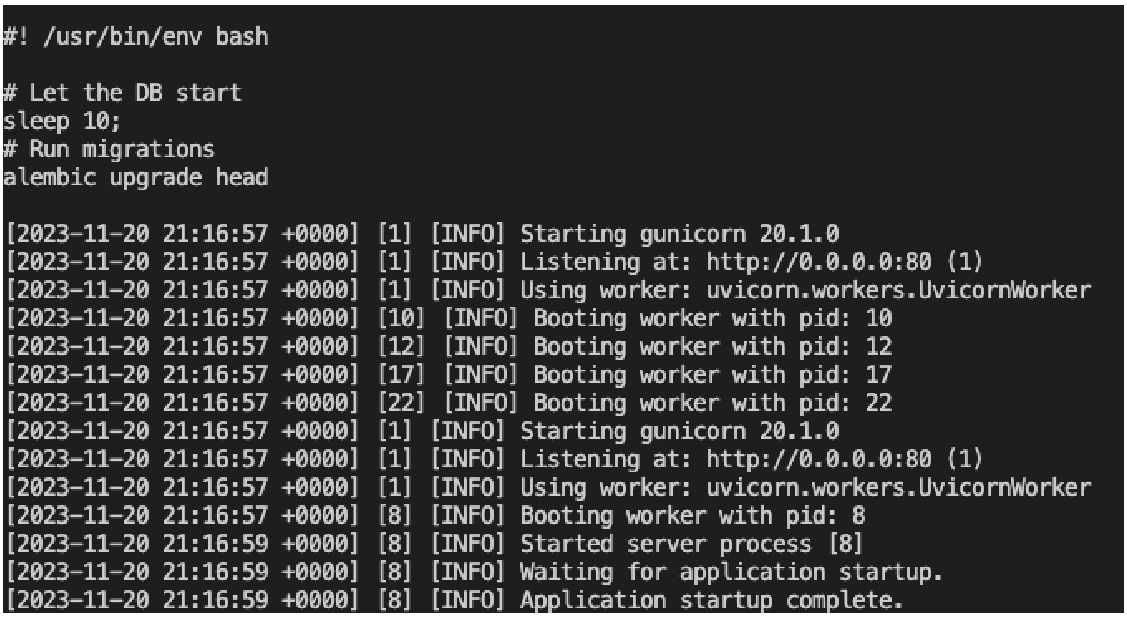
     - Make sure the script has execute permissions by running

       ```
       $ chmod +x run_api.sh
       ```

     - Run the script to run the container by typing

       ```
       $ ./run_api.sh 1313:80
       ```

       you will see the API server started, you can access the server document at (http://localhost:1313/docs)



```
#! /usr/bin/env bash

# Let the DB start
sleep 10;
# Run migrations
alembic upgrade head

[2023-11-20 21:16:57 +0000] [1] [INFO] Starting gunicorn 20.1.0
[2023-11-20 21:16:57 +0000] [1] [INFO] Listening at: http://0.0.0.0:80 (1)
[2023-11-20 21:16:57 +0000] [1] [INFO] Using worker: uvicorn.workers.UvicornWorker
[2023-11-20 21:16:57 +0000] [10] [INFO] Booting worker with pid: 10
[2023-11-20 21:16:57 +0000] [12] [INFO] Booting worker with pid: 12
[2023-11-20 21:16:57 +0000] [17] [INFO] Booting worker with pid: 17
[2023-11-20 21:16:57 +0000] [22] [INFO] Booting worker with pid: 22
[2023-11-20 21:16:57 +0000] [1] [INFO] Starting gunicorn 20.1.0
[2023-11-20 21:16:57 +0000] [1] [INFO] Listening at: http://0.0.0.0:80 (1)
[2023-11-20 21:16:57 +0000] [1] [INFO] Using worker: uvicorn.workers.UvicornWorker
[2023-11-20 21:16:57 +0000] [8] [INFO] Booting worker with pid: 8
[2023-11-20 21:16:59 +0000] [8] [INFO] Started server process [8]
[2023-11-20 21:16:59 +0000] [8] [INFO] Waiting for application startup.
[2023-11-20 21:16:59 +0000] [8] [INFO] Application startup complete.
```

2. **Model Loading:**

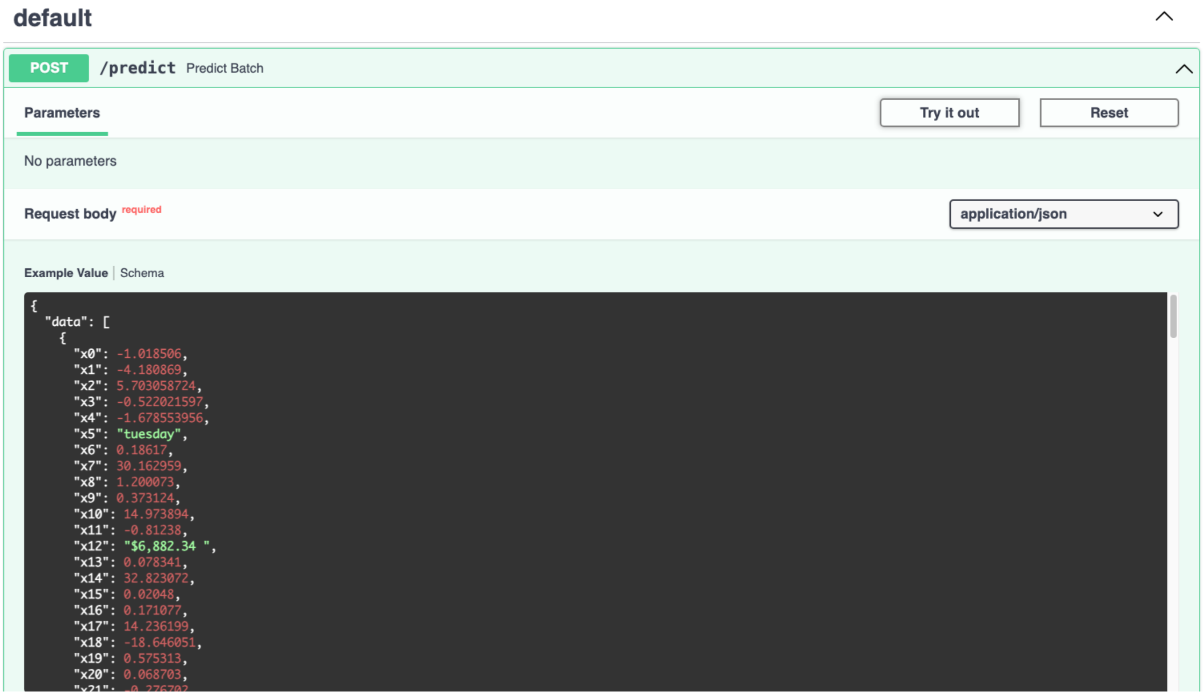
   - The pre-trained GLM model is stored in the `model.pkl` under `model/` directory.
   - The model is loaded during the FastAPI application startup.

3. **FastAPI Application:**

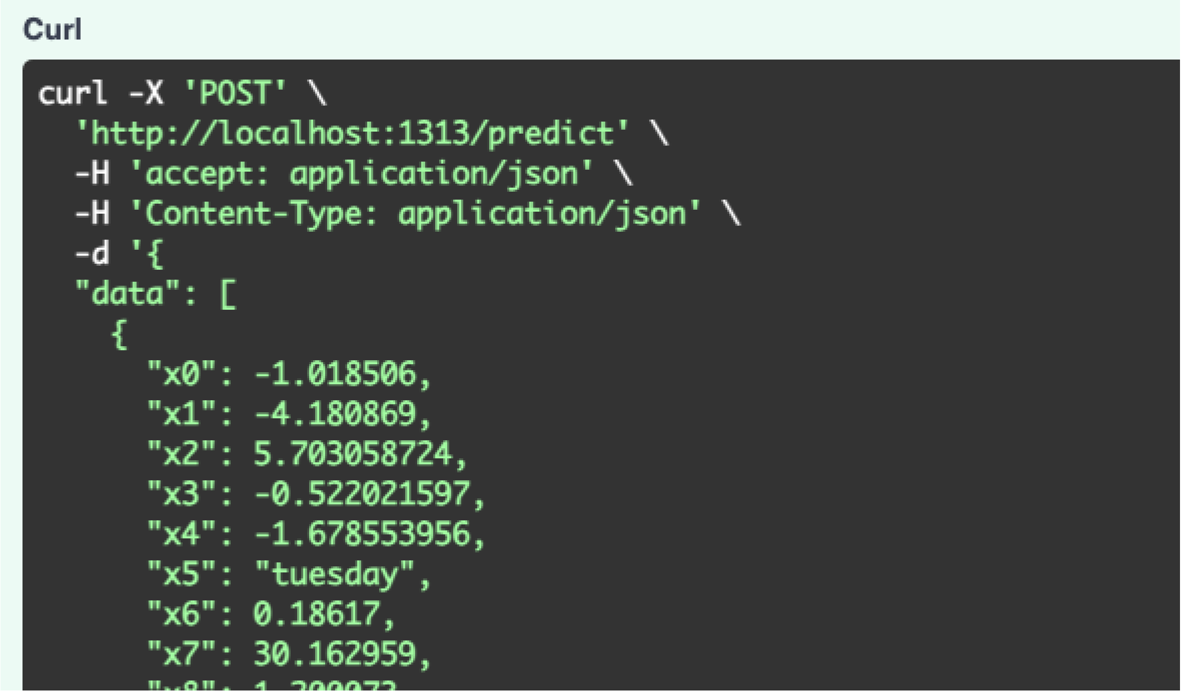
   - The FastAPI application (`app/main.py`) defines API endpoints for model prediction.
   - Input data is received via HTTP requests and passed to the pre-trained GLM model.
   - To start the FastAPI server locally, change the directory to `app` and run

     ```
     $ uvicorn main:app --reload
     ```

   - open FastAPI Swagger UI in a browser, which provides interactive API documentation and exploration of web user interfaces.
   - Click "Try it out" at POST/predict and you can test out the model predictions by copying the data from test/test_output.json and pasting the data to the request body, or uploading a file at POST/uploadfile. You can test the output without typing the curl command manually

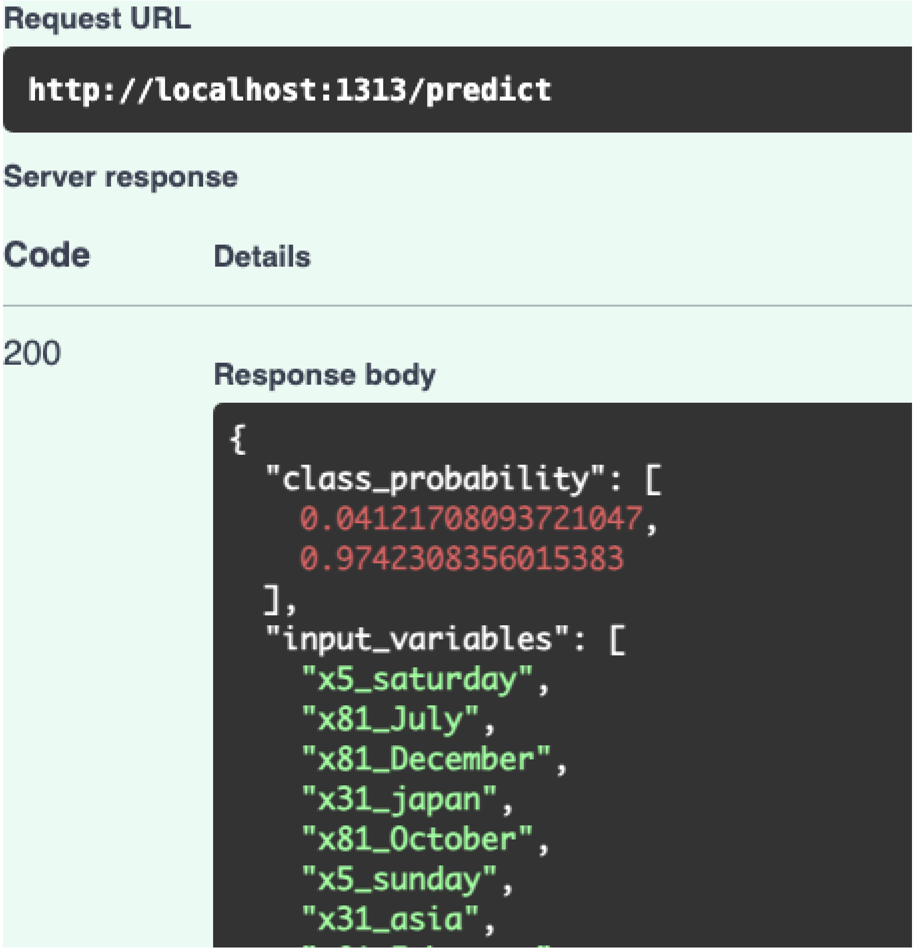


```
{
  "data": [
    {
      "x0": -1.018506,
      "x1": -4.180869,
      "x2": 5.703058724,
      "x3": -0.522021597,
      "x4": -1.678553956,
      "x5": "tuesday",
      "x6": 0.18617,
      "x7": 30.162959,
      "x8": 1.200073,
      "x9": 0.373124,
      "x10": 14.973894,
      "x11": -0.81238,
      "x12": "$6,882.34 ",
      "x13": 0.078341,
      "x14": 32.823072,
      "x15": 0.02048,
      "x16": 0.171077,
      "x17": 14.236199,
      "x18": -18.646051,
      "x19": 0.575313,
      "x20": 0.068703,
      "x21": -0.276702
```



Curl

```
curl -X 'POST' \
  'http://localhost:1313/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "data": [
    {
      "x0": -1.018506,
      "x1": -4.180869,
      "x2": 5.703058724,
      "x3": -0.522021597,
      "x4": -1.678553956,
      "x5": "tuesday",
      "x6": 0.18617,
      "x7": 30.162959,
```



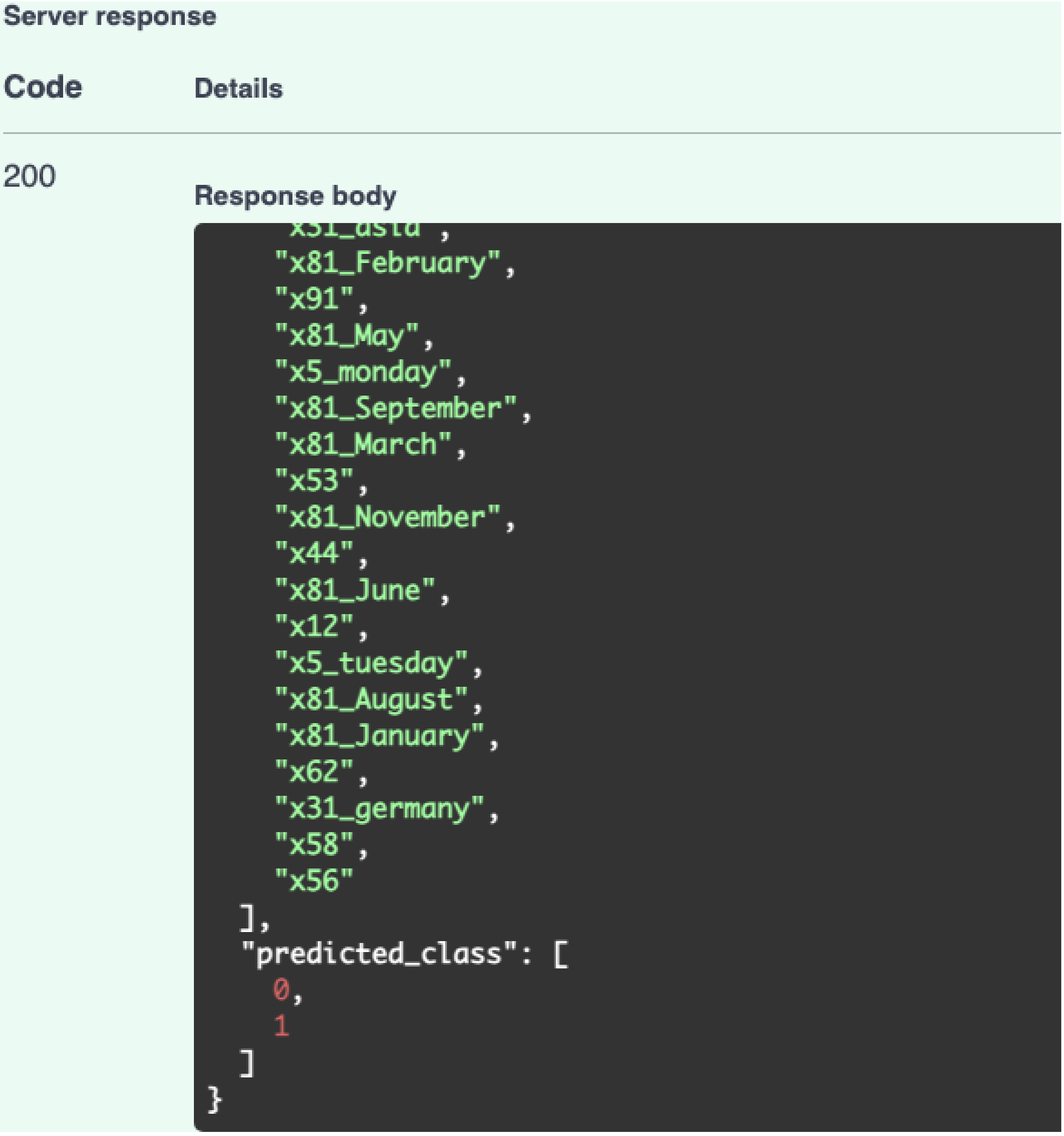
Request URL

```
http://localhost:1313/predict
```

Server response

Code	Details
200	Response body

```
{
  "class_probability": [
    0.04121708093721047,
    0.9742308356015383
  ],
  "input_variables": [
    "x5_saturday",
    "x81_July",
    "x81_December",
    "x31_japan",
    "x81_October",
    "x5_sunday",
    "x31_asia",
```



Server response

Code Details

200

Response body

```
    "x31_asia",
    "x81_February",
    "x91",
    "x81_May",
    "x5_monday",
    "x81_September",
    "x81_March",
    "x53",
    "x81_November",
    "x44",
    "x81_June",
    "x12",
    "x5_tuesday",
    "x81_August",
    "x81_January",
    "x62",
    "x31_germany",
    "x58",
    "x56"
  ],
  "predicted_class": [
    0,
    1
  ]
}
```

- Check out the documentation at Swagger UI Docs

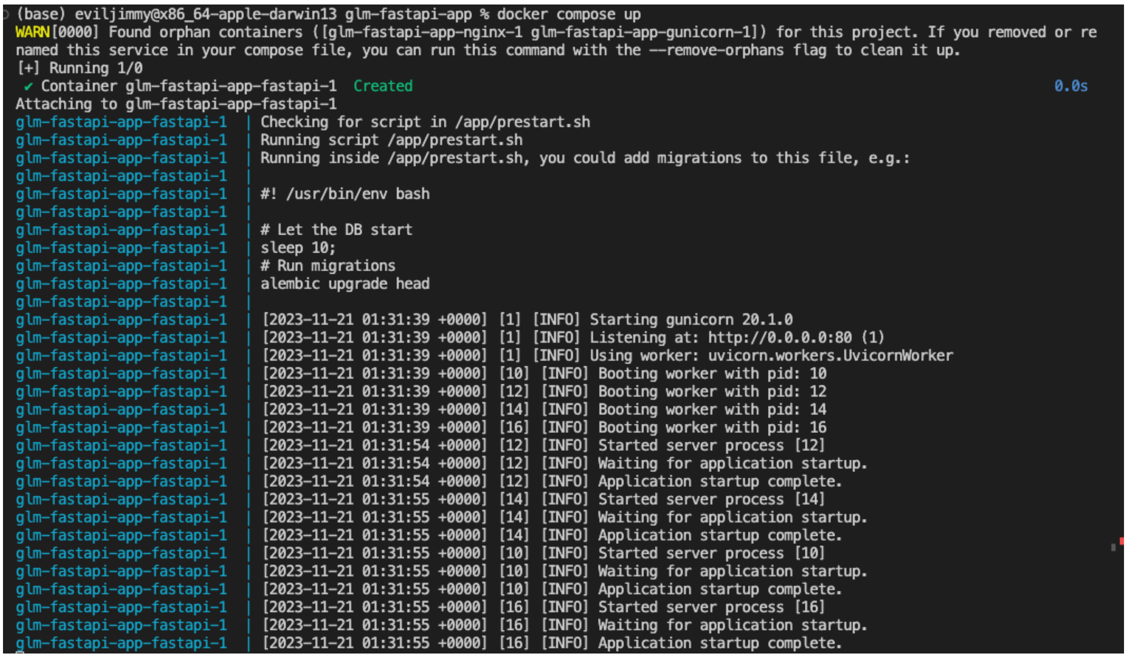
4. **Orchestration:**

   - Orchestration using Docker compose

     - To run the docker-compose, change the directory to the project directory, run

       ```
       $ docker-compose up
       ```

       It starts building a docker container using the image`glm-fastapi-app`and run at port 1313 and map to port 80 for the container



```
(base) eviljimmy@x86_64-apple-darwin13 glm-fastapi-app % docker compose up
WARN[0000] Found orphan containers ([glm-fastapi-app-nginx-1 glm-fastapi-app-gunicorn-1]) for this project. If you removed or re
named this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
[+] Running 1/0
 ✓ Container glm-fastapi-app-fastapi-1  Created                                                                      0.0s
Attaching to glm-fastapi-app-fastapi-1
glm-fastapi-app-fastapi-1  | Checking for script in /app/prestart.sh
glm-fastapi-app-fastapi-1  | Running script /app/prestart.sh
glm-fastapi-app-fastapi-1  | Running inside /app/prestart.sh, you could add migrations to this file, e.g.:
glm-fastapi-app-fastapi-1  |
glm-fastapi-app-fastapi-1  | #! /usr/bin/env bash
glm-fastapi-app-fastapi-1  |
glm-fastapi-app-fastapi-1  | # Let the DB start
glm-fastapi-app-fastapi-1  | sleep 10;
glm-fastapi-app-fastapi-1  | # Run migrations
glm-fastapi-app-fastapi-1  | alembic upgrade head
glm-fastapi-app-fastapi-1  |
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:39 +0000] [1] [INFO] Starting gunicorn 20.1.0
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:39 +0000] [1] [INFO] Listening at: http://0.0.0.0:80 (1)
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:39 +0000] [1] [INFO] Using worker: uvicorn.workers.UvicornWorker
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:39 +0000] [10] [INFO] Booting worker with pid: 10
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:39 +0000] [12] [INFO] Booting worker with pid: 12
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:39 +0000] [14] [INFO] Booting worker with pid: 14
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:39 +0000] [16] [INFO] Booting worker with pid: 16
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:54 +0000] [12] [INFO] Started server process [12]
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:54 +0000] [12] [INFO] Waiting for application startup.
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:54 +0000] [12] [INFO] Application startup complete.
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:55 +0000] [14] [INFO] Started server process [14]
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:55 +0000] [14] [INFO] Waiting for application startup.
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:55 +0000] [14] [INFO] Application startup complete.
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:55 +0000] [10] [INFO] Started server process [10]
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:55 +0000] [10] [INFO] Waiting for application startup.
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:55 +0000] [10] [INFO] Application startup complete.
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:55 +0000] [16] [INFO] Started server process [16]
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:55 +0000] [16] [INFO] Waiting for application startup.
glm-fastapi-app-fastapi-1  | [2023-11-21 01:31:55 +0000] [16] [INFO] Application startup complete.
```

  - To stop the service, run

    ```
    $ docker-compose down
    ```

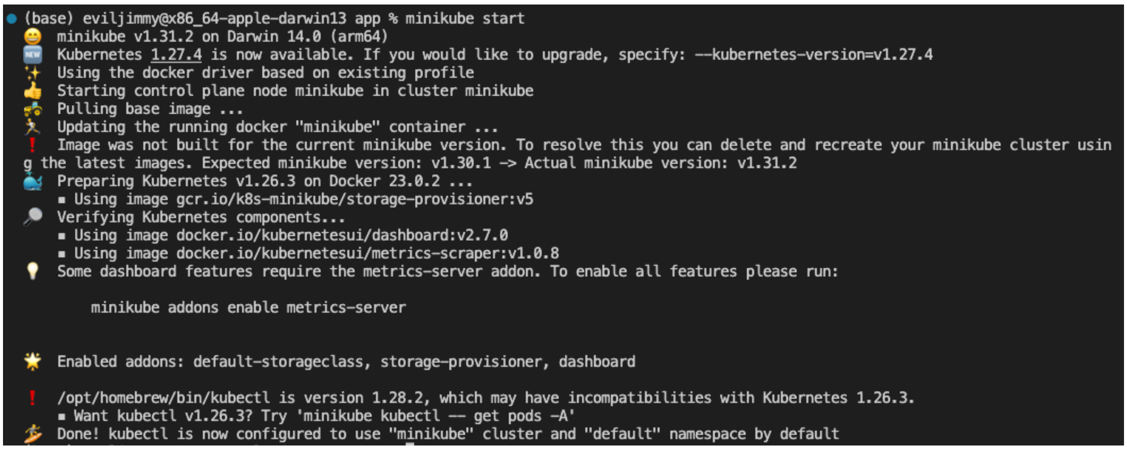
- Orchestration Using Kubernetes

  - Kubernetes deployment YAML (`kubernetes/deployment.yml`) defines how the FastAPI application should run as pods.

  - Kubernetes service YAML (`kubernetes/service.yml`) exposes the application within the cluster.

  - To start the orchestration process locally, start Minikube by running

    ```
    $ minikube start
    ```



```
● (base) eviljimmy@x86_64-apple-darwin13 app % minikube start
😄  minikube v1.31.2 on Darwin 14.0 (arm64)
🆕  Kubernetes 1.27.4 is now available. If you would like to upgrade, specify: --kubernetes-version=v1.27.4
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🏃  Updating the running docker "minikube" container ...
❗  Image was not built for the current minikube version. To resolve this you can delete and recreate your minikube cluster usin
g the latest images. Expected minikube version: v1.30.1 -> Actual minikube version: v1.31.2
🐳  Preparing Kubernetes v1.26.3 on Docker 23.0.2 ...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🔎  Verifying Kubernetes components...
    ▪ Using image docker.io/kubernetesui/dashboard:v2.7.0
    ▪ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
💡  Some dashboard features require the metrics-server addon. To enable all features please run:

        minikube addons enable metrics-server


🌟  Enabled addons: default-storageclass, storage-provisioner, dashboard

❗  /opt/homebrew/bin/kubectl is version 1.28.2, which may have incompatibilities with Kubernetes 1.26.3.
    ▪ Want kubectl v1.26.3? Try 'minikube kubectl -- get pods -A'
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

  - load the image to Minikube by running

```
$ minikube image load glm-fast-api:1.0
```

- To deploy the resources defined in your deployment.yaml file to a Kubernetes cluster, you can use the kubectl command-line tool. Here are the steps to deploy these resources:

  ```
  $ kubectl apply -f deployment.yaml
  ```

  This will create the deployment and start the specified number of replicas.

- Check the list of deployments by running

  ```
  $ kubectl get deployments
  ```

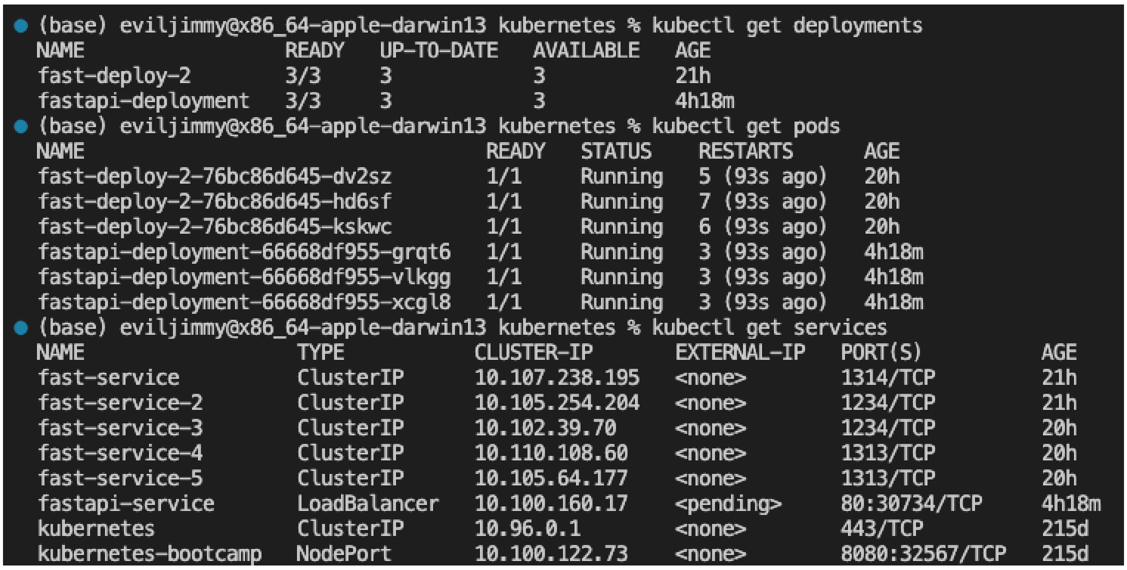
- Check the pod by running

  ```
  $ kubectl get pod
  ```

- Check services by running

  ```
  $ kubectl get services
  ```

- Here is an example of the output of the above commands, you can see we created 3 replicas in the pod. We deployed them as load balancers to handle large amounts of requests



```
● (base) eviljimmy@x86_64-apple-darwin13 kubernetes % kubectl get deployments
  NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
  fast-deploy-2        3/3     3            3           21h
  fastapi-deployment   3/3     3            3           4h18m
● (base) eviljimmy@x86_64-apple-darwin13 kubernetes % kubectl get pods
  NAME                                  READY   STATUS    RESTARTS      AGE
  fast-deploy-2-76bc86d645-dv2sz        1/1     Running   5 (93s ago)   20h
  fast-deploy-2-76bc86d645-hd6sf        1/1     Running   7 (93s ago)   20h
  fast-deploy-2-76bc86d645-kskwc        1/1     Running   6 (93s ago)   20h
  fastapi-deployment-66668df955-grqt6   1/1     Running   3 (93s ago)   4h18m
  fastapi-deployment-66668df955-vlkgg   1/1     Running   3 (93s ago)   4h18m
  fastapi-deployment-66668df955-xcgl8   1/1     Running   3 (93s ago)   4h18m
● (base) eviljimmy@x86_64-apple-darwin13 kubernetes % kubectl get services
  NAME                  TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
  fast-service          ClusterIP      10.107.238.195   <none>        1314/TCP         21h
  fast-service-2        ClusterIP      10.105.254.204   <none>        1234/TCP         21h
  fast-service-3        ClusterIP      10.102.39.70     <none>        1234/TCP         20h
  fast-service-4        ClusterIP      10.110.108.60    <none>        1313/TCP         20h
  fast-service-5        ClusterIP      10.105.64.177    <none>        1313/TCP         20h
  fastapi-service       LoadBalancer   10.100.160.17    <pending>     80:30734/TCP     4h18m
  kubernetes            ClusterIP      10.96.0.1        <none>        443/TCP          215d
  kubernetes-bootcamp   NodePort       10.100.122.73    <none>        8080:32567/TCP   215d
```

- You can see that for each deployment, there are 3 replicas as load balancers, which we can test out the performance later
- To expose the Kubernetes service deployment named 'fastapi-deployment', we can run

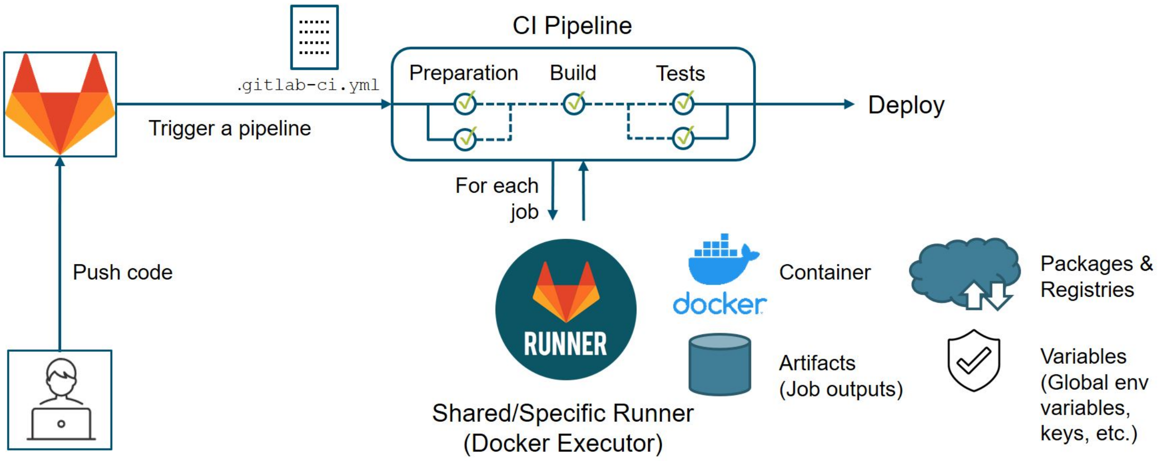
```
$ kubectl expose deploy/fastapi-deployment --name=fast-
service --target-port=80 --port 1313
```

- to perform port forwarding and test out the service run

```
$ minikube service fast-service
```

then you can test the service from the pop-up window.

5. **GitLab CI/CD Pipeline:**
   - `.gitlab-ci.yml` contains the CI/CD pipeline configuration.
   - The pipeline includes stages for linting, testing, building the Docker image, and deploying to Kubernetes.
   - To use this CI/CD configuration, make sure you have GitLab CI/CD configured for your repository, and the necessary variables (e.g., Docker registry credentials) are set in your GitLab project settings.
   - When you push changes to the master branch, GitLab CI/CD will automatically trigger the pipeline, and it will execute the defined stages. The Docker image will be built, push to the registry, and then the application will be deployed to Kubernetes.
6. **CI/CD Workflow:**
   - Code changes trigger the GitLab CI/CD pipeline.
   - Automated testing ensures code quality.
   - Docker image is built and pushed to the container registry.
   - Kubernetes deployment is updated with the new image.

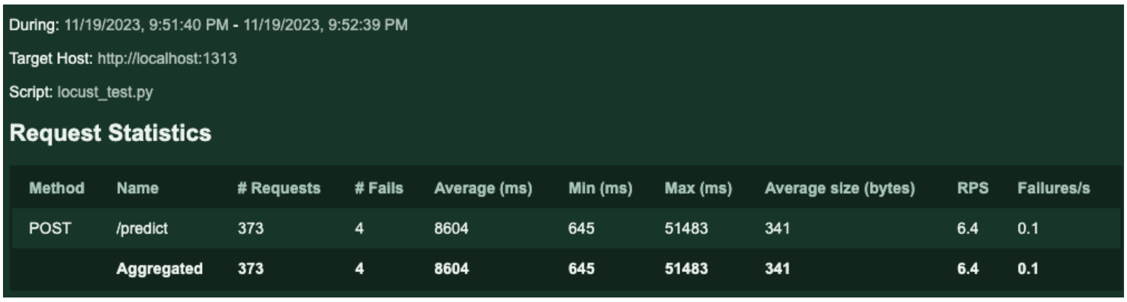


Unit Test & Performance Test

1. **Unit Test**
   - pytest: change the directory to test/pytest and run the command

     ```
     $ pytest
     ```

     it will run the `test_main.py` for unit test and `test_requests` for batch test
2. **Performance Test**
   - locust: change directory to test/locust and run command 'locust -f locust_test.py', it will open a server at (http://127.0.0.1:8089/). You can specify the test load and it can output the performance test report. See detailed documentation at this medium post
   - Testing Result (see detailed reports in the test/locust/report)
     - 10000 users with 10 user requests per sec in 60 sec using a single API port



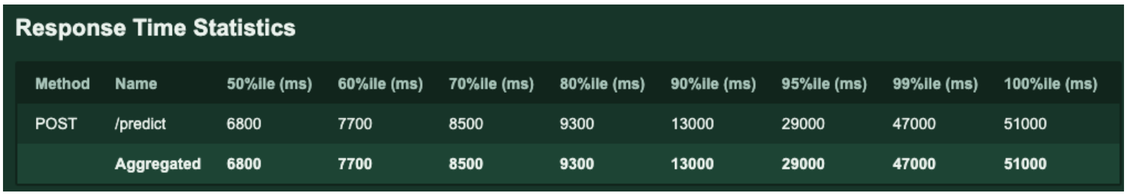
During: 11/19/2023, 9:51:40 PM - 11/19/2023, 9:52:39 PM

Target Host: http://localhost:1313

Script: locust_test.py

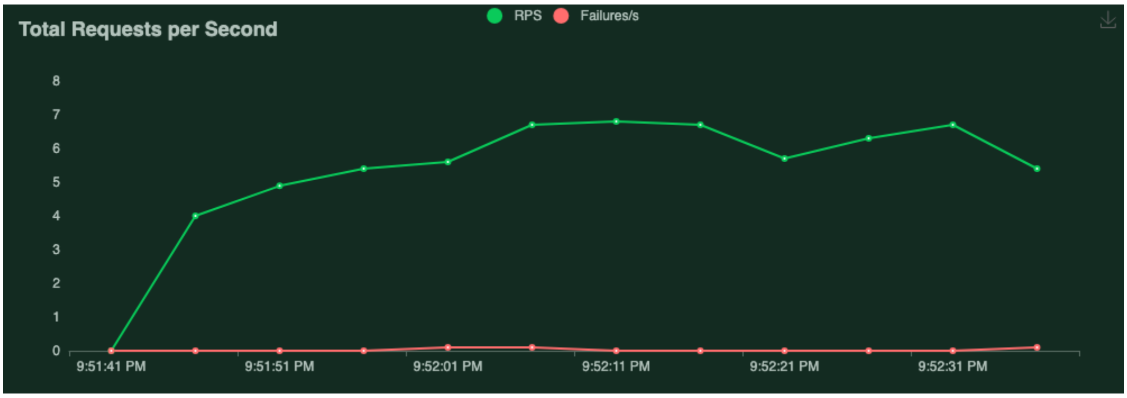
Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/predict	373	4	8604	645	51483	341	6.4	0.1
	Aggregated	373	4	8604	645	51483	341	6.4	0.1

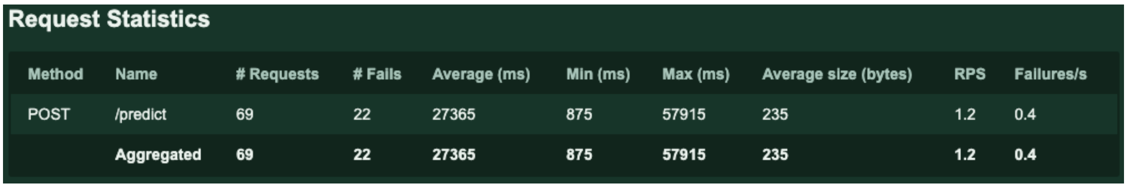


Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/predict	6800	7700	8500	9300	13000	29000	47000	51000
	Aggregated	6800	7700	8500	9300	13000	29000	47000	51000

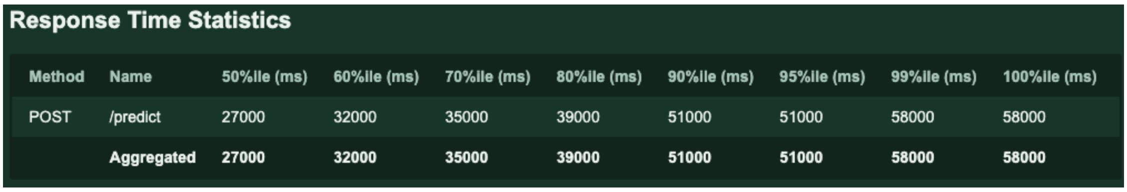


- 10000 users with 10 user requests per sec in 60 sec using 3 API replicas



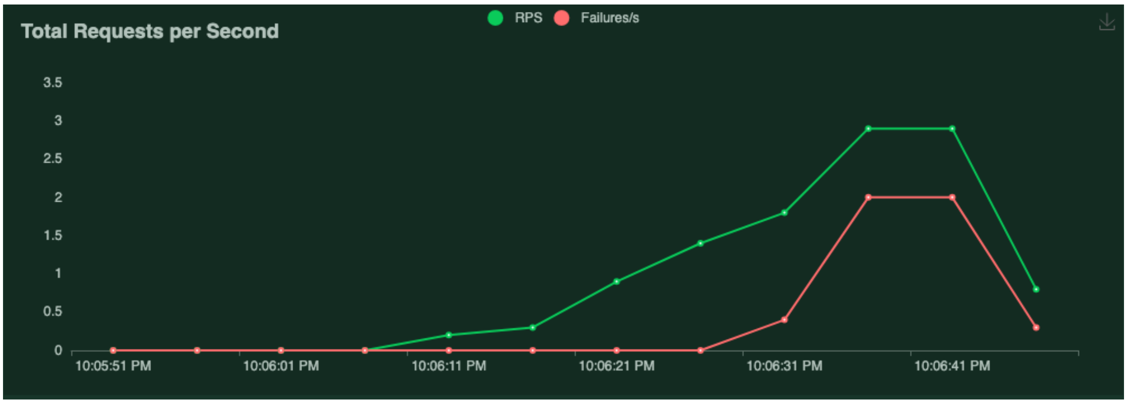
Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/predict	69	22	27365	875	57915	235	1.2	0.4
	Aggregated	69	22	27365	875	57915	235	1.2	0.4



Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/predict	27000	32000	35000	39000	51000	51000	58000	58000
	Aggregated	**27000**	**32000**	**35000**	**39000**	**51000**	**51000**	**58000**	**58000**



- We can see that using 3 replicas balance the load for large amounts of request there for total request per second is lower. However, when the number of request per sec increase to 100, a single api port cannot process them effectively and return error, while the 3 replicas and process them. This is an example of performing API performance test.

Opportunities For Scalability

1. Using Scalable Architecture
   - Design a scalable architecture that can handle increased load. Consider microservices architecture, load balancing, and scalable databases, which we did in the Kubenetes cluster
   - Use cloud services like AWS Elastic Kubernetes Service (EKS) to leverage auto-scaling features based on demand
2. Code Optimization
   - Optimize code and database queries for efficiency
   - Use caching mechanisms to reduce redundant computations and database queries
3. Load Balancing
   - Implement load balancing to distribute incoming requests across multiple servers to prevent overload on a single server, we show this process using Kubenetes replicas. We can use AWS services like Elastic Container Services (ECS) to automatically route request to difference ports
4. Caching
   - Employ caching strategies (e.g., in-memory caching, CDN caching) to store frequently requested data and reduce response time
5. Asynchronous Processing
   - Offload time-consuming tasks to background jobs or queues to ensure faster response times for critical API requests.
   - Inside the app/main.py, we define the `async def predict_batch(data: InputDatas):` function, which implement asynchronous processing for slow process. We can run tests and identify slow processes for improving processing speed.
6. Parallel Processing framework
   - Utilize big data framework like Spark to scale out, rewrite the deployment code to Pyspark code, read the model using Spark ML, create Spark df to preprocess the data, then output the predictions using Spark ML.

7. Monitoring and Analytics
    - Use monitoring tools to track API performance, identify bottlenecks, and troubleshoot issues in real time. In the performance test section, we used locust to monitor the performance in real time.
    - If we are deploying to cloud services like AWS CloudWatch, we can monitor the performance of the FastAPI application deployed on AWS

Notes

- Update configuration files (`Dockerfile`, `kubernetes/deployment.yml`) based on your model and requirements.
- Adjust GitLab CI/CD settings and environment variables in the GitLab project.