Flight Delay Prediction App Load Testing

Objective:

The goal is to stress test the service. We will create a model and api service for the model to send request and get the prediction based on data.
In the following document you will find various methods of deploying the application. At first we will deploy this application on local system to test the functionality and will do stress test.
With time, we can improve the infrastructure of the application (for eg. Running in kubernetes cluster or cloudrun service from google.)

Bookmarks :

1. Expose the serialized model via REST API.
   - You can use the already serialized model (pickle_model.pkl) or rebuild it.

2. Automate the process of building and deploying the API, using one or more cloud services.

3. Stress test the API with the exposed model with at least 50,000 requests for 45 seconds. For this, you must use wrk tool and present the metrics obtained.
   - How could you improve the performance of the previous tests?

4. The infrastructure creation process must be done with Terraform.

5. What would be the ideal mechanisms so that only authorized systems can access this API?
   - Does this mechanism add any level of latency to the consumer? Because?

6. What would be the SLIs and SLOs that you would define and why?

7. Monitoring.

App running on local system-

Performance:

Here we can assume that application will get affected if the system goes down and to scale up the infrastructure, we need manual intervention.

Repository:
Git

Steps Overview:

-:Pikling file is created and provided in drive.
-:Create api and web service that will run on local system directly.
-:Api request will be done using browser.

-Let's create a model.

Pre-requisite-
Create python virtual environment and install dependencies.
python3 -m venv sre-challenge
pip install flask
pip install flask-restful
pip install bumpy
pip install pandas
pip install scikit-learn

Will download a new dataset which consist flight related details.
Download flight data from https://raw.githubusercontent.com/plotly/datasets/master/flightdata.csv

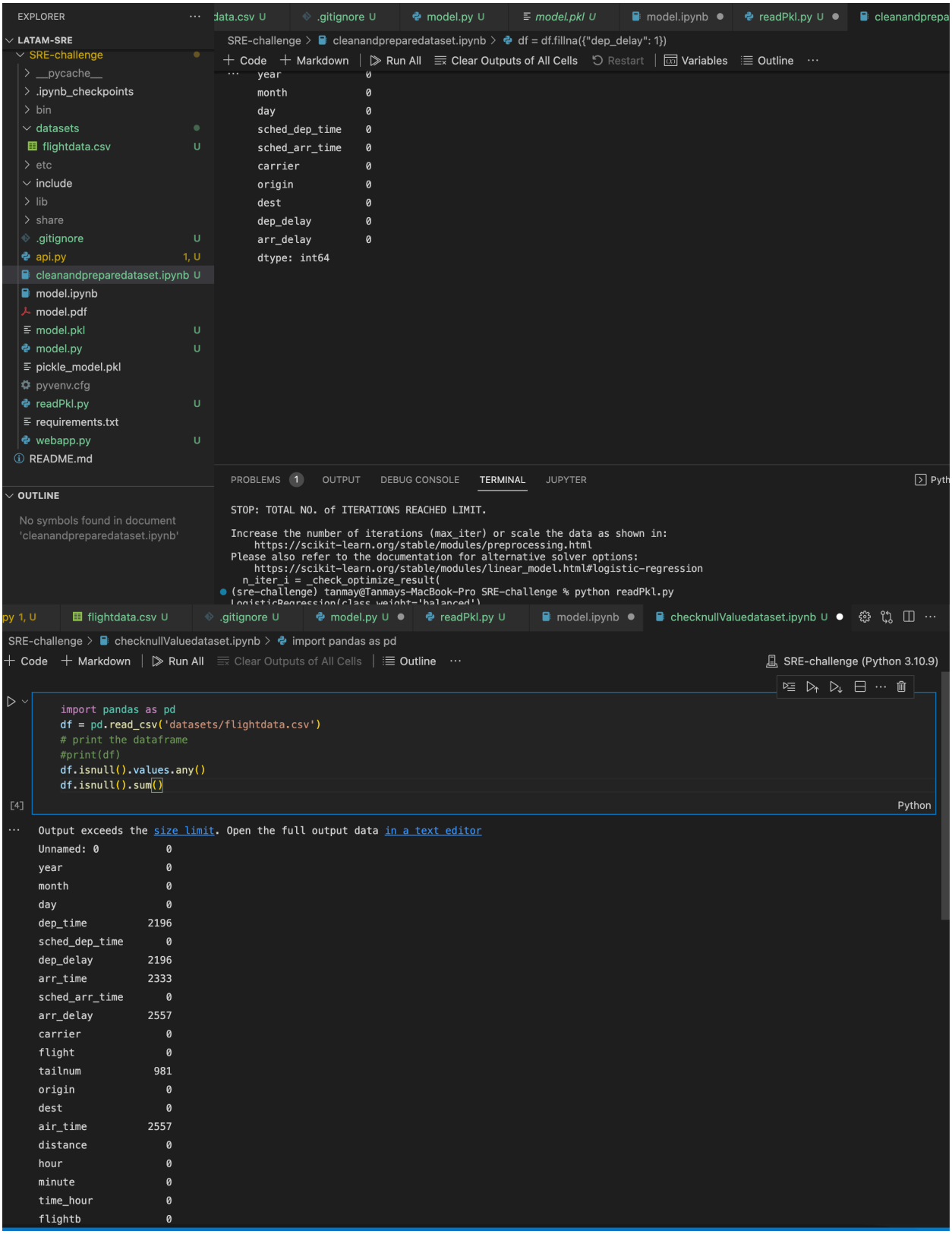
Goal:
In a model.py file, we will develop and train our model.
Load the dataset.
Clean the dataset
Create model.pkl file based on the dataset.

Clean and prepare data:

- Figure out "feature " Columns that are relevant to the output we are predicting.
- Eliminate Missing values either by deleting rows or columns or adding meaningful values

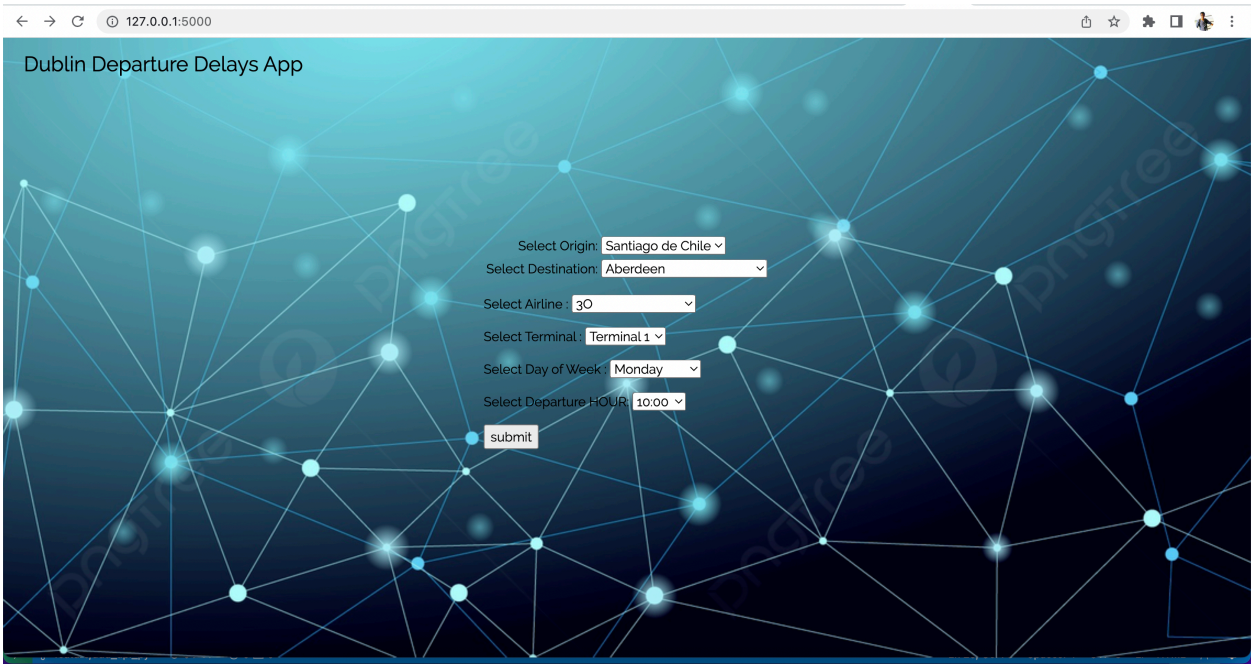
NOTE: Check for null value in the data set by creating new notebook-
(cleanandpreparedataset.ipynb notebook for test and validate results, added to repo)

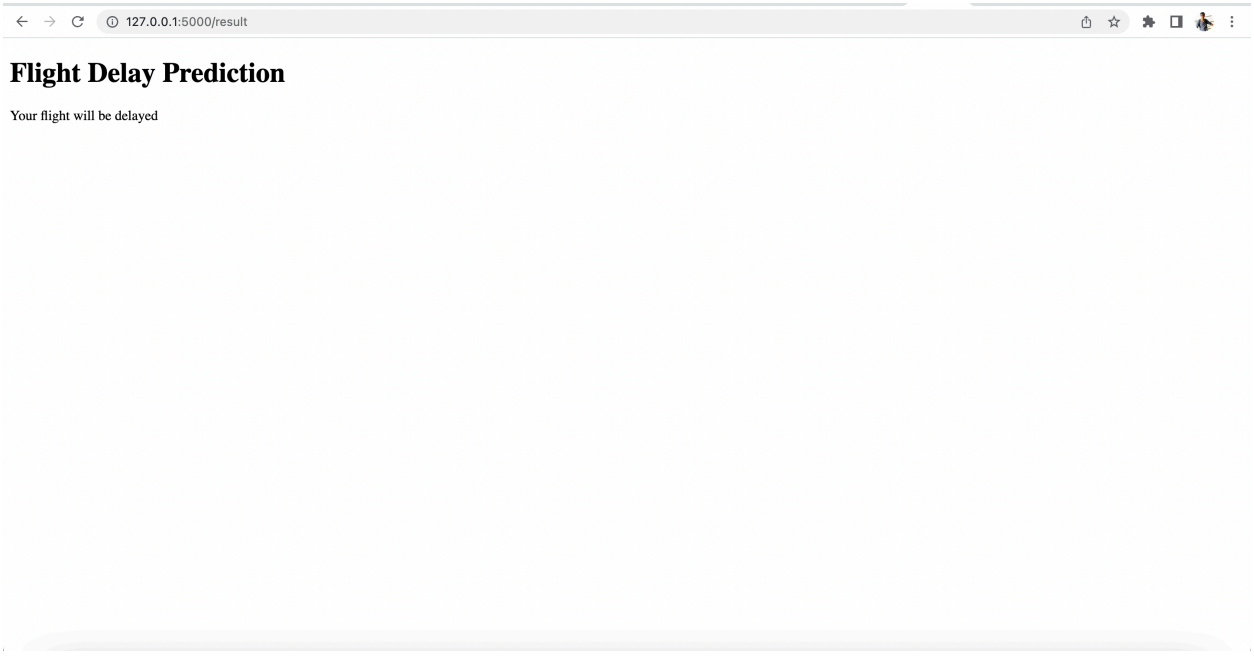
-Added file model.py.
-Model.pkl file generated and stored in root directory



-Uploaded changes to git https://github.com/tjtanmay/latam-sre/tree/feature/add_api_py

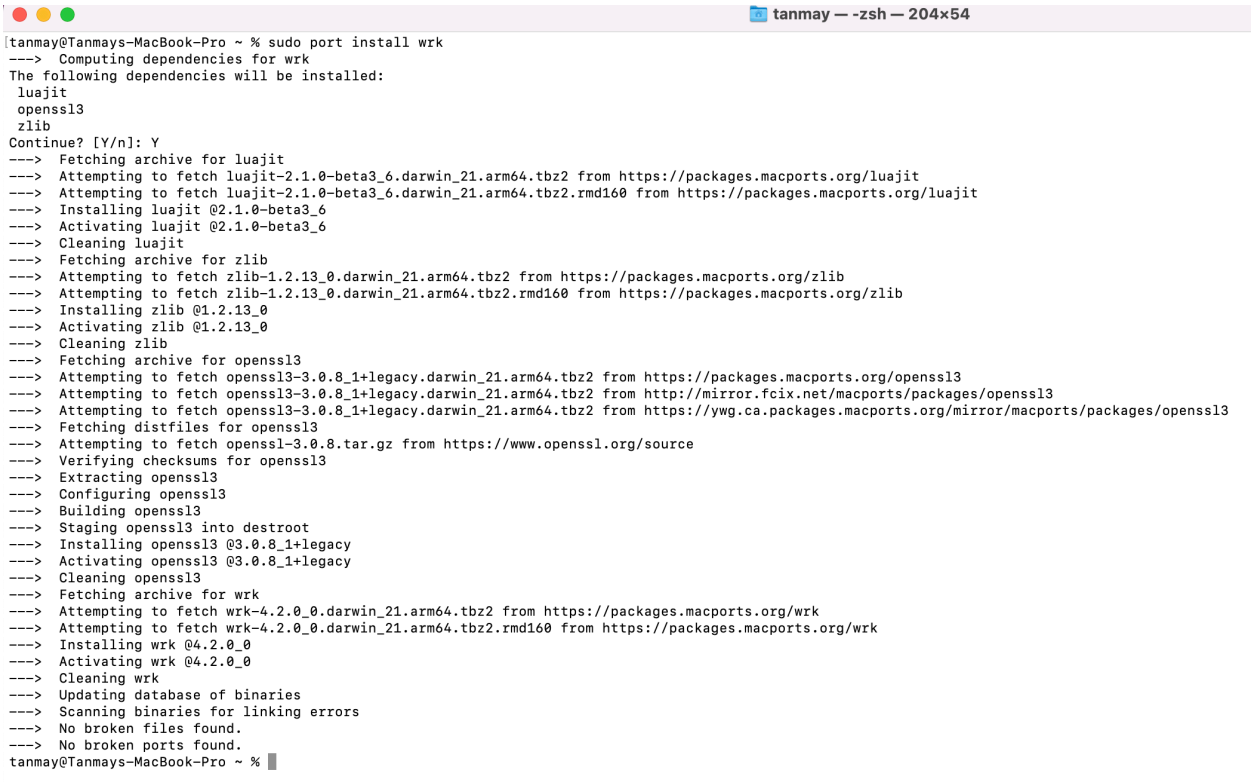
-Execute python api.py to launch.





Load Testing

-Install wrk benchmark tool



Result:
After running load test for 30 seconds, using 12 threads, and keeping 400 HTTP connections open

```
tanmay@Tanmays-MacBook-Pro ~ % wrk -t12 -c400 -d30s http://
127.0.0.1:5000/
Running 30s test @ http://127.0.0.1:5000/
  12 threads and 400 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    46.12ms   36.43ms 280.88ms   82.78%
    Req/Sec   158.69    100.23   595.00     66.61%
  10673 requests in 30.03s, 219.88MB read
  Socket errors: connect 0, read 10978, write 369, timeout 0
Requests/sec:    355.43
Transfer/sec:      7.32MB

tanmay@Tanmays-MacBook-Pro ~ % wrk -t12 -c50000 -d45s http://
127.0.0.1:5000/
Running 45s test @ http://127.0.0.1:5000/
  12 threads and 50000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
```

```
    Latency   104.15ms  162.43ms   1.61s    98.06%
    Req/Sec    25.00     46.66   424.00     92.23%
  4117 requests in 45.10s, 84.82MB read
Socket errors: connect 47447, read 652563, write 3326, timeout 0
Requests/sec:     91.29
Transfer/sec:      1.88MB
```

- Added post.lua script with body to request from API and triggered with script.

```
tanmay@Tanmays-MacBook-Pro SRE-challenge % wrk -t12 -c50000 -d45s -s ./scripts/post.lua http://127.0.0.1:5000/
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
Running 45s test @ http://127.0.0.1:5000/
  12 threads and 50000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    41.88ms   35.65ms 306.64ms   74.34%
    Req/Sec   115.76    112.21   820.00     77.90%
  19001 requests in 45.10s, 391.63MB read
  Socket errors: connect 47447, read 18422, write 450, timeout 0
Requests/sec:    421.30
Transfer/sec:      8.68MB
tanmay@Tanmays-MacBook-Pro SRE-challenge %
```

Conclusion:

Here we can see the performance of the service depends upon availability of resource of the local system. We have to manually bring up the service every time if service goes down.
This kind of infrastructure is not good for production use rather, developer can follow the above steps to test their application locally quickly without depending upon infra.

App running on Docker-

Performance:

As the application will run on Docker, maintaining container will be easy. We can build the application image which makes the application portable and we don't need to spend time on configuring infrastructure for the application.

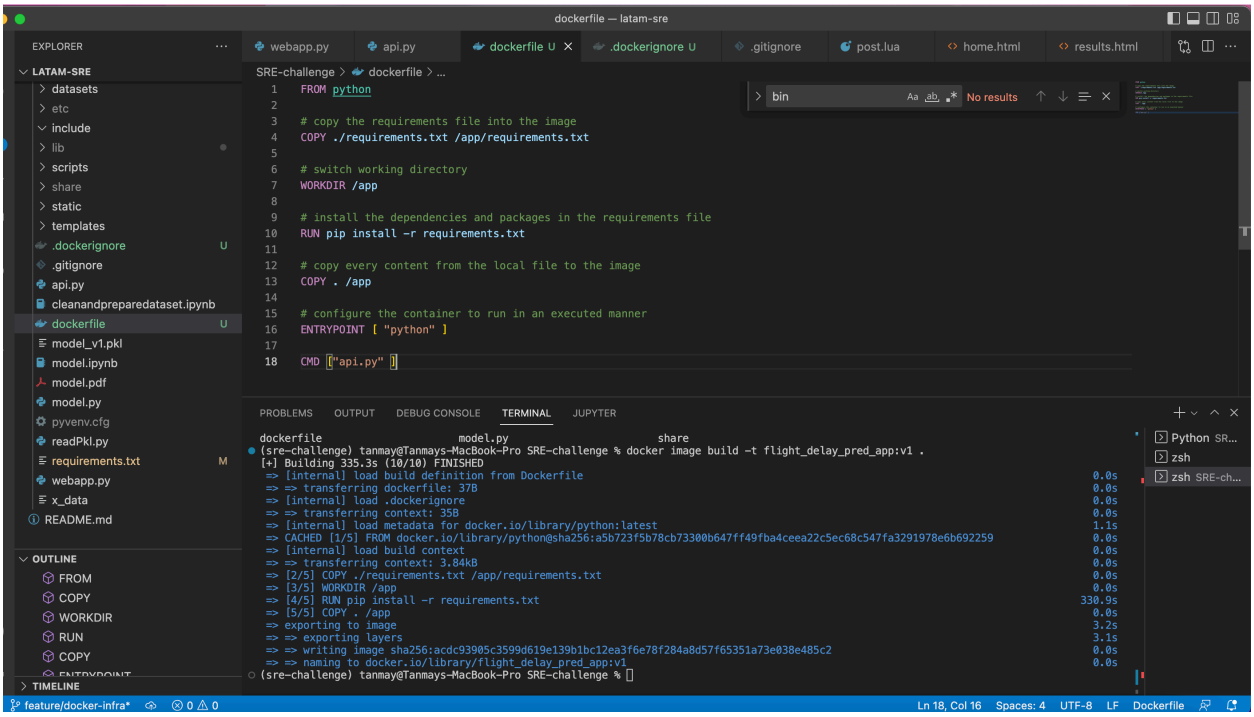
Repository:
Git

Steps Overview:

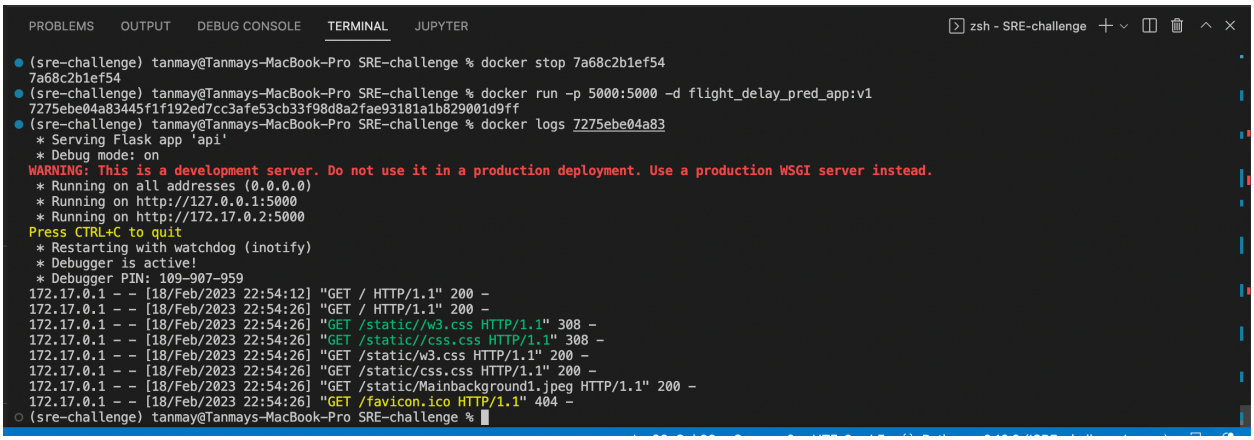
-:Create dockerfile with steps required to build the application
-:Validate functionality of the application and push the image to docker hub.
-:Stress test the application.
-:Automate the build process.

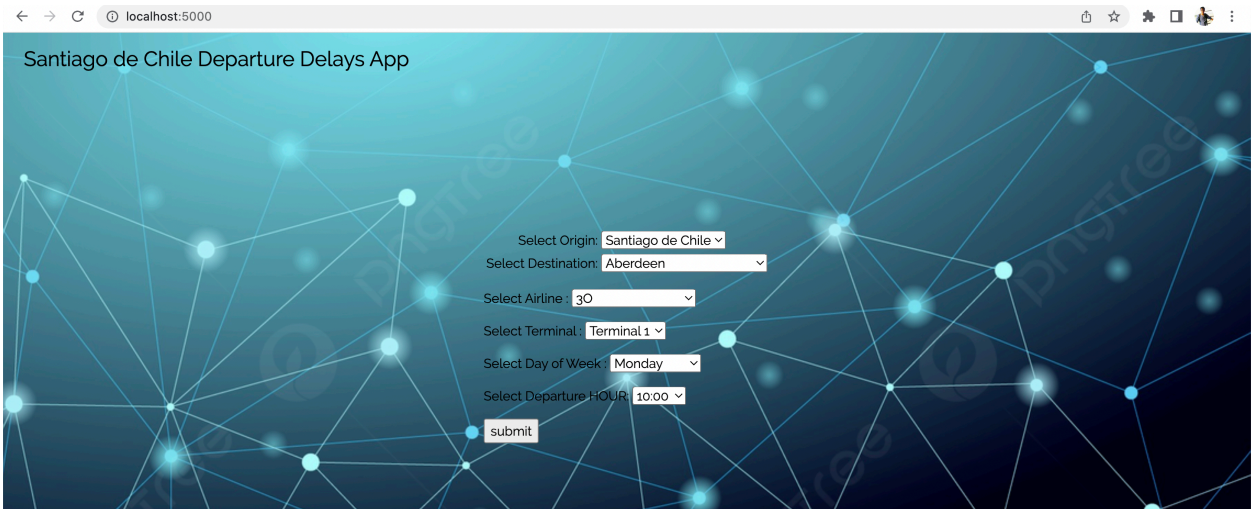
Pre-requisite:
- docker installed on local machine.
- Docker hub account.

-Added docker file and build image for the application.



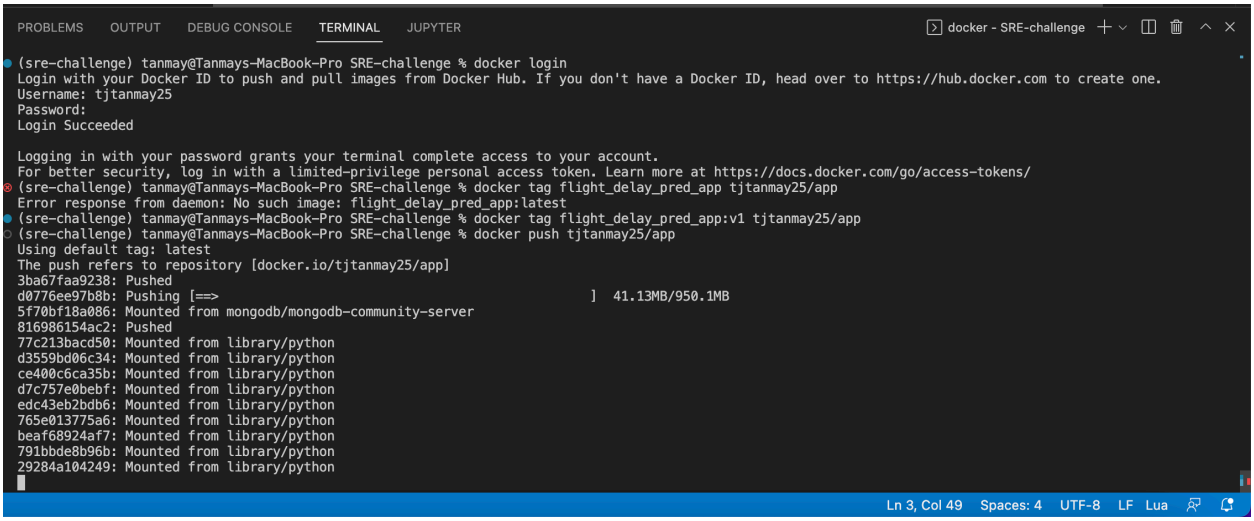
-Run the image and access it from localhost-





So our application which is running in Docker is accessible.

-Let’s push the image in dockerhub repository.



Load Testing

```
[tanmay@Tanmays-MacBook-Pro SRE-challenge % docker logs flight_delay_pred_app
Error: No such container: flight_delay_pred_app
[tanmay@Tanmays-MacBook-Pro SRE-challenge % docker logs flight_delay_pred_app:v1
Error: No such container: flight_delay_pred_app:v1
[tanmay@Tanmays-MacBook-Pro SRE-challenge % wrk -t12 -c50000 -d45s -s ./scripts/post.lua http://127.0.0.1:5000/
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
Running 45s test @ http://127.0.0.1:5000/
  12 threads and 50000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   197.99ms  364.31ms   1.99s    90.22%
    Req/Sec   104.48     80.42   660.00     68.77%
  26989 requests in 45.09s, 556.27MB read
  Socket errors: connect 47447, read 4850, write 383, timeout 1281
Requests/sec:    598.62
Transfer/sec:     12.34MB
tanmay@Tanmays-MacBook-Pro SRE-challenge %
```

Automate build process

We will use Git action to automate the build process and will create CI pipeline, which will build and push the image into docker hub.

-Created new branch https://github.com/tjtanmay/latam-sre/tree/feature/add_git_workflow

-Added the workflow file in master and merged latest changes in master branch.

-After successfully executing GitHub action pipeline, image will be pushed to Docker hub.

Github Action pipeline run- Link

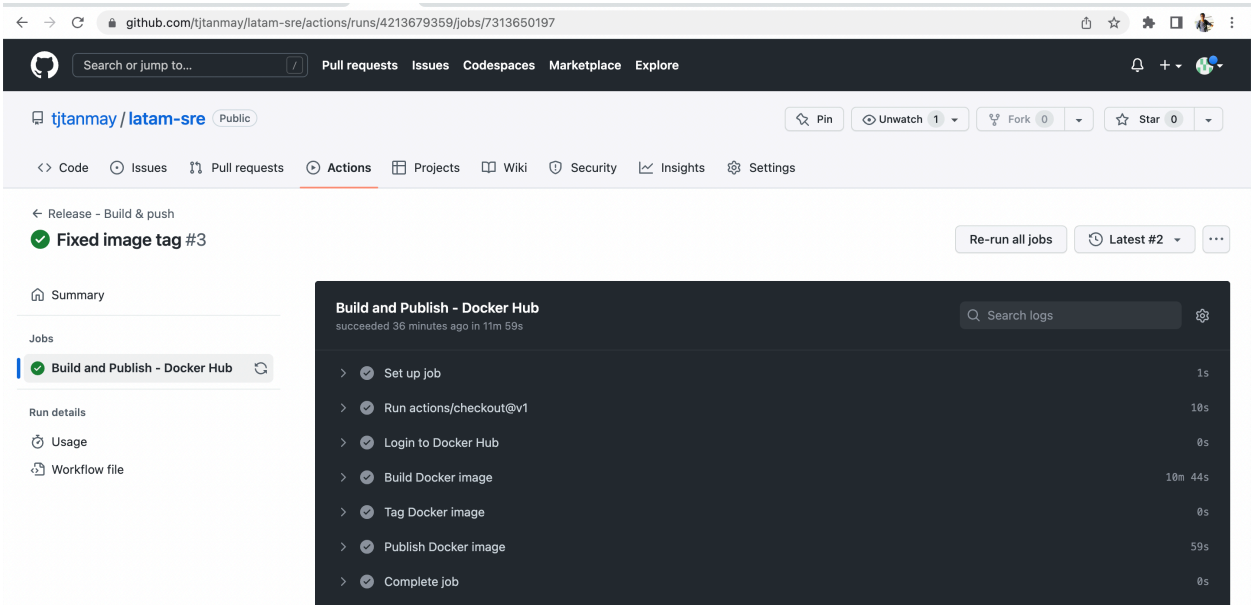
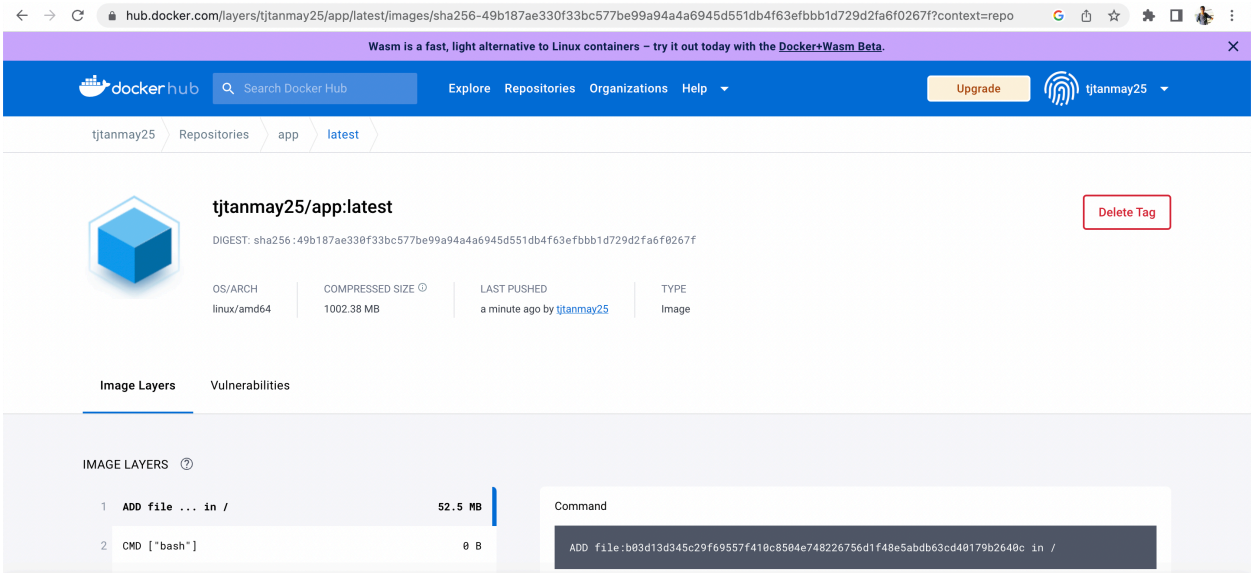


Image pushed to docker hub- Link



Validate image:
-Let’s test the image and run-

```
SRE-challenge — -zsh —
tanmay@Tanmays-MacBook-Pro SRE-challenge % docker pull tjtanmay25/app:latest
latest: Pulling from tjtanmay25/app
1e4aec178e08: Pull complete
6c1024729fee: Pull complete
c3aa11fbc85a: Pull complete
aa54add66b3a: Pull complete
9e3a60c2bce7: Pull complete
3b2123ce9d0d: Pull complete
05df7720fcb8: Pull complete
972ab8743e38: Pull complete
ae9f20f2cd37: Pull complete
66657012bff4: Pull complete
c94b04b84a68: Pull complete
039406b7ecf8: Pull complete
Digest: sha256:49b187ae330f33bc577be99a94a4a6945d551db4f63efbbb1d729d2fa6f0267f
Status: Downloaded newer image for tjtanmay25/app:latest
docker.io/tjtanmay25/app:latest
```

-Running locally

```
tanmay@Tanmays-MacBook-Pro SRE-challenge % docker run -p 5001:5000 tjtanmay25/app
WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested
 * Serving Flask app 'api'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.3:5000
Press CTRL+C to quit
 * Restarting with watchdog (inotify)
 * Debugger is active!
 * Debugger PIN: 811-433-153
172.17.0.1 - - [19/Feb/2023 00:49:42] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [19/Feb/2023 00:49:42] "GET /static//css.css HTTP/1.1" 308 -
172.17.0.1 - - [19/Feb/2023 00:49:42] "GET /static//w3.css HTTP/1.1" 308 -
172.17.0.1 - - [19/Feb/2023 00:49:43] "GET /static/Mainbackground1.jpeg HTTP/1.1" 200 -
172.17.0.1 - - [19/Feb/2023 00:49:43] "GET /static/w3.css HTTP/1.1" 200 -
172.17.0.1 - - [19/Feb/2023 00:49:43] "GET /static/css.css HTTP/1.1" 200 -
172.17.0.1 - - [19/Feb/2023 00:49:43] "GET /favicon.ico HTTP/1.1" 404 -
/usr/local/lib/python3.11/site-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but AdaBoostClassifier was fitted with feature names
  warnings.warn(
172.17.0.1 - - [19/Feb/2023 00:50:21] "POST /result HTTP/1.1" 200 -
172.17.0.1 - - [19/Feb/2023 00:50:21] "GET /static/css.css HTTP/1.1" 304 -
```

Application is working fine.

Conclusion:

We deployed the application in docker and pushed the image into docker hub repository.
We can see number of request handled by the application increased significantly.
Also we automated the process of building docker image using GitHub actions.

App running on Kubernetes cluster-

Performance:

Now we will deploy the application on kubernetes cluster. This will make scalability easier and as per the load usage, we can easily scale up the infrastructure I.e. cluster.

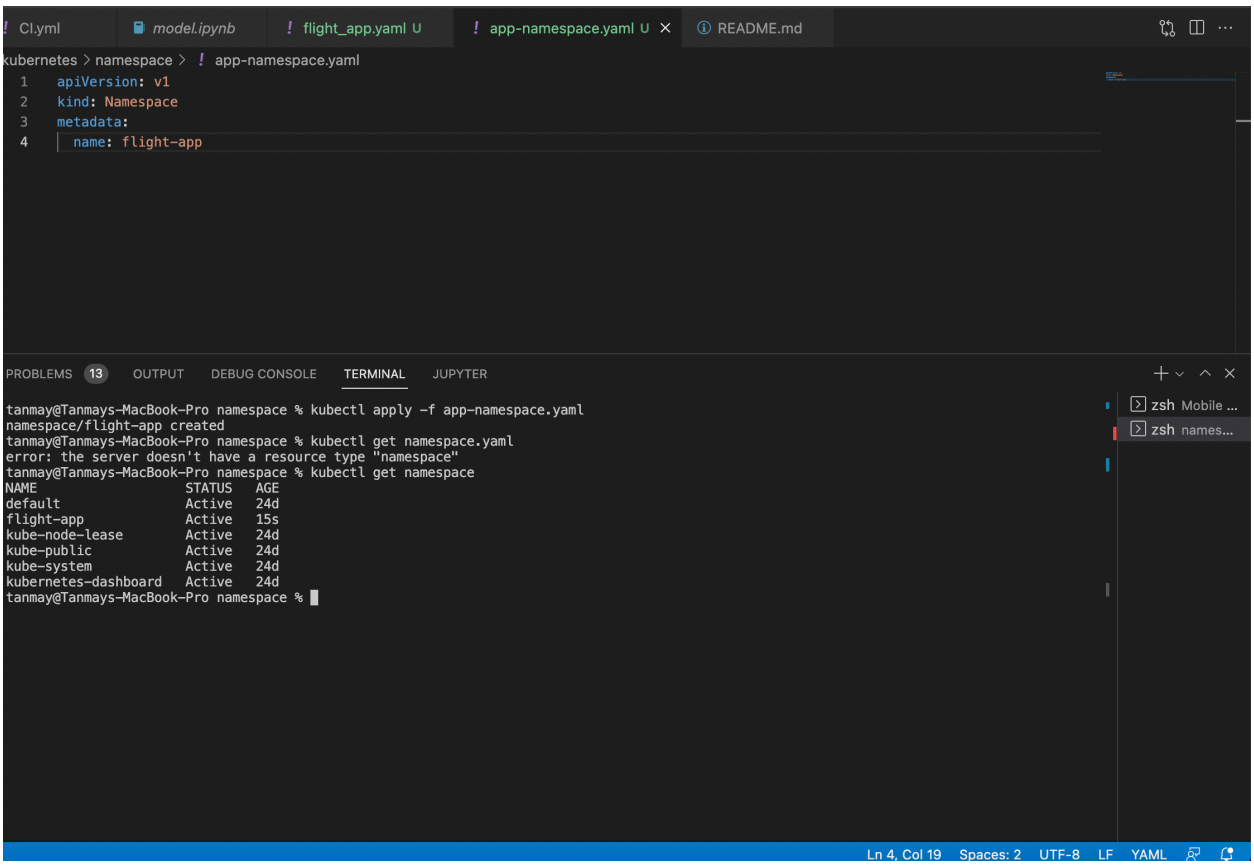
Repository:
Git

Steps Overview:

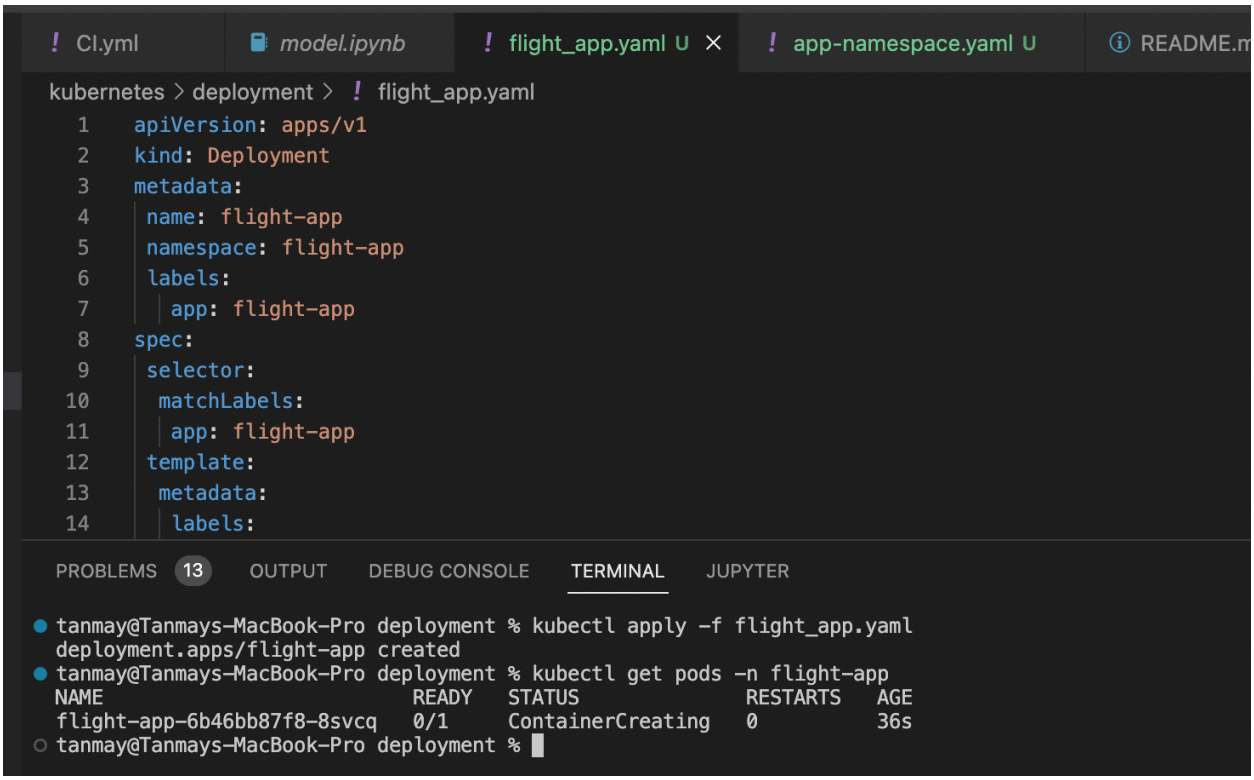
-:Write deployment and service file.
-:Expose the service.
-:Stress test the application.

Pre-requisite:
- minikube installed on local system.

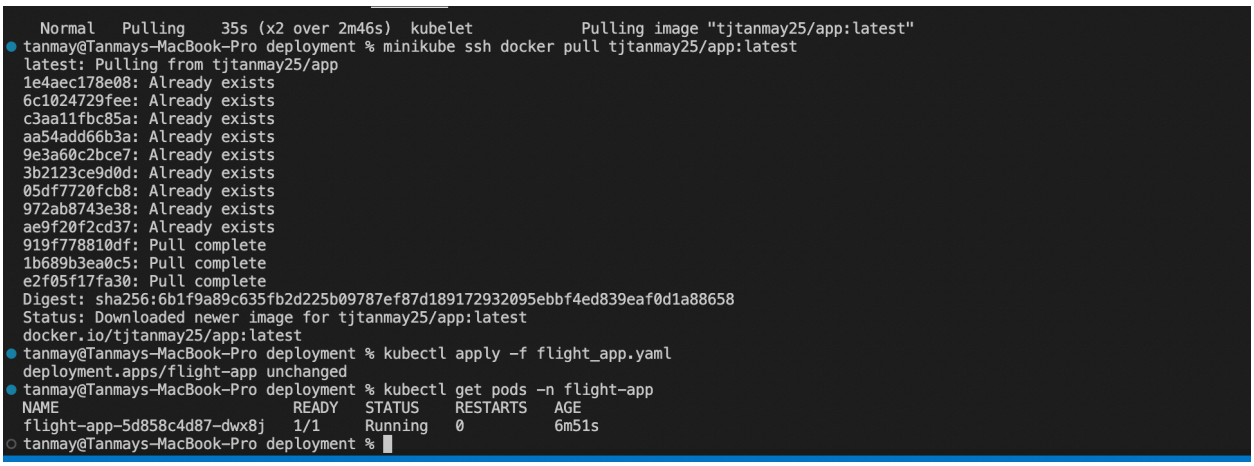
-Create namespace-



-Added deployment file for pod creation and executed-



Note: for minikube, you have to download image manually into minikube context using below command-
minikube ssh docker pull tjtanmay25/app:latest



-We have to expose the application, for this added service.yaml

```
tanmay@Tanmays-MacBook-Pro kubernetes % cd  service
tanmay@Tanmays-MacBook-Pro service % kubectl apply -f service.yaml
service/flight-app created
tanmay@Tanmays-MacBook-Pro service % kubectl describe service flight-app
Error from server (NotFound): services "flight-app" not found
tanmay@Tanmays-MacBook-Pro service % kubectl describe service flight-app -n flight-app
Name:                     flight-app
Namespace:                flight-app
Labels:                   <none>
Annotations:              <none>
Selector:                 app=flight-app
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.99.73.134
IPs:                      10.99.73.134
Port:                     <unset>  5000/TCP
TargetPort:               5000/TCP
NodePort:                 <unset>  31453/TCP
Endpoints:                172.17.0.2:5000
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
tanmay@Tanmays-MacBook-Pro service %
```

service.yaml — latam-sre

CI.yml | flight_app.yaml U | home.html | service.yaml U | .gitignore U | app-namespace.yaml U | README.md

kubernetes > service > service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: flight-app
  namespace: flight-app
spec:
  selector:
    app: flight-app
  ports:
  - port: 5000
    targetPort: 5000
  type: LoadBalancer
  externalIPs:
  - 192.168.49.2
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

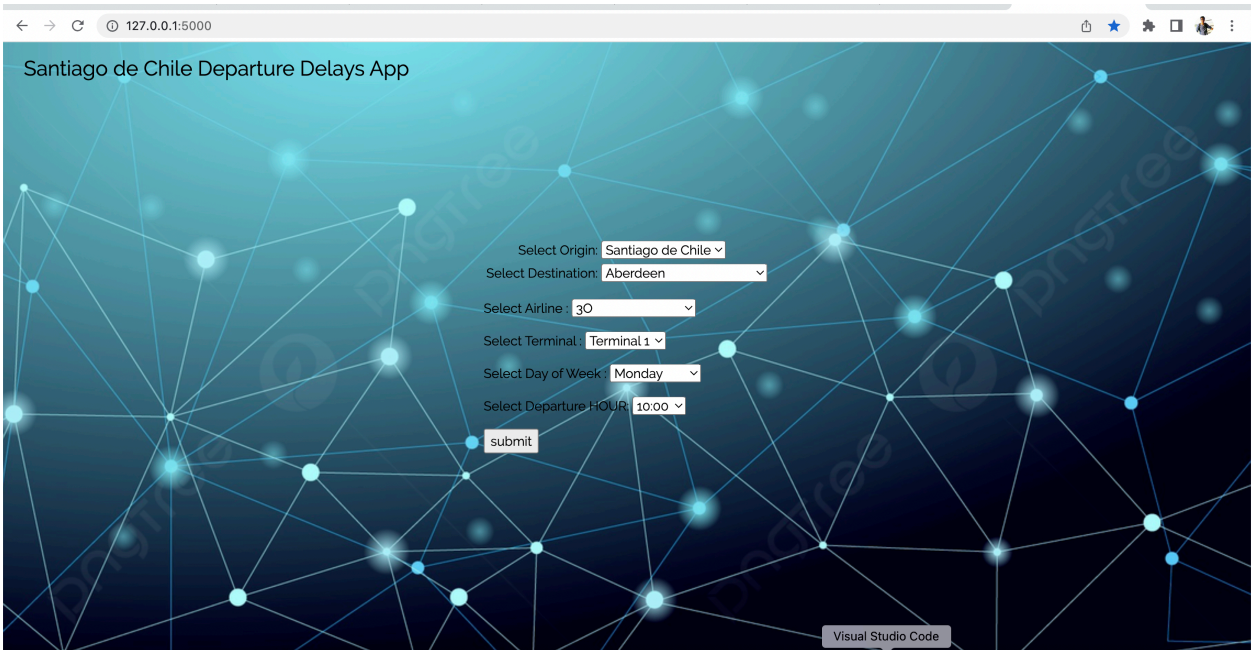
```
tanmay@Tanmays-MacBook-Pro latam-sre % kubectl get service -n flight-app
NAME         TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
flight-app   NodePort   10.99.73.134   <none>        5000:31453/TCP   35m
tanmay@Tanmays-MacBook-Pro latam-sre % minikube ip
192.168.49.2
tanmay@Tanmays-MacBook-Pro latam-sre % ls
README.md                       dockerfile                      model.py                        share
__pycache__                     etc                             model_v1.pkl                    static
api.py                          kubernetes                      pyvenv.cfg                      templates
bin                             lib                             readPkl.py                      webapp.py
cleanandpreparedataset.ipynb    model.ipynb                     requirements.txt                x_data
datasets                        model.pdf                       scripts
tanmay@Tanmays-MacBook-Pro latam-sre % cd kubernetes/service
tanmay@Tanmays-MacBook-Pro service % ls
service.yaml
tanmay@Tanmays-MacBook-Pro service % kubectl apply -f service.yaml
The Service "flight-app" is invalid: spec.type: Unsupported value: "LoadBalencer": supported values: "ClusterIP", "ExternalName", "LoadBala
ncer", "NodePort"
tanmay@Tanmays-MacBook-Pro service % kubectl apply -f service.yaml
service/flight-app configured
tanmay@Tanmays-MacBook-Pro service % kubectl get service -n flight-app
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP                 PORT(S)          AGE
flight-app   LoadBalancer   10.99.73.134   127.0.0.1,192.168.49.2      5000:31453/TCP   65m
tanmay@Tanmays-MacBook-Pro service %
```

minikube s...
zsh service

Note: For minikube, need to open tunnel in order to access the service

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

tanmay@Tanmays-MacBook-Pro latam-sre % minikube tunnel
✅  Tunnel successfully started

📌  NOTE: Please do not close this terminal as this process must stay alive for the tunnel to be accessible ...

🏃  Starting tunnel for service flight-app.
```

Result:
Application is accessible



Autoscale:

kubectl autoscale deployment flight-app --cpu-percent=80 --min=1 —max=4

This will increase pods to a maximum of four replicas (--max=4) when the PHP web application deployment experiences more than 80% CPU use (--cpu-percent=80) over a sustained period.

Load Testing

```
tanmay@Tanmays-MacBook-Pro latam-sre % wrk -t12 -c50000 -d45s -s ./scripts/post.lua http://127.0.0.1:5000/
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
./scripts/post.lua: ./scripts/post.lua:2: unexpected symbol near '10'
Running 45s test @ http://127.0.0.1:5000/
  12 threads and 50000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   845.40ms  481.26ms   2.00s    60.71%
    Req/Sec    27.22     40.99   370.00     90.37%
  1031 requests in 45.09s, 21.26MB read
  Socket errors: connect 47447, read 650130, write 250, timeout 634
Requests/sec:     22.87
Transfer/sec:    482.74KB
tanmay@Tanmays-MacBook-Pro latam-sre % wrk -t12 -c50000 -d45s -s ./scripts/post.lua http://127.0.0.1:5000/
```

Conclusion:

We have successfully deployed the application in kubernetes infrastructure. As we can see, load test results not showing improvement. This is because, kuberenets is running locally and application is running on top of that. So performance depends upon the local system limit. In this case, kubernetes is using significant amount of resources from local system.

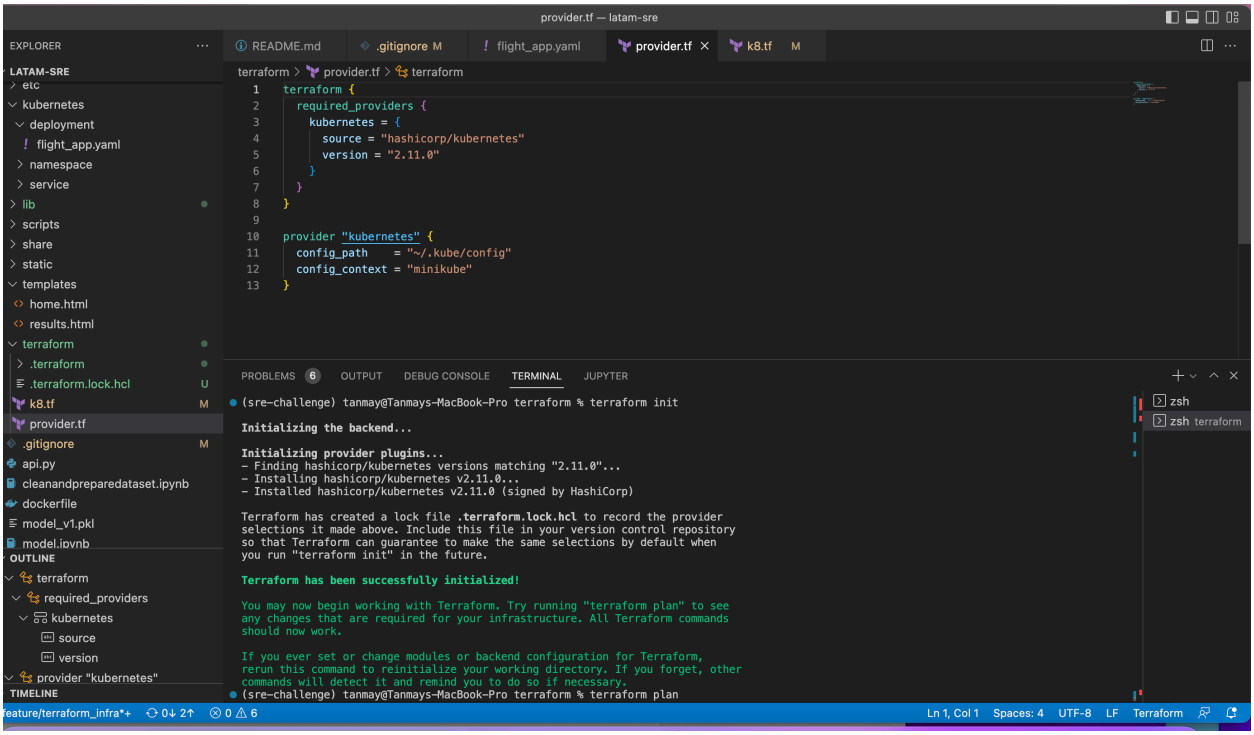
Managing infrastructure using Terraform

As of now, we have automated build process for the application using Github action which is pushing build artifacts to docker hub repository. Then we created Kubernetes cluster manually and deployed the application. Let's introduce terraform to manage the Kubernetes infrastructure. We will write configuration for Kubernetes infrastructure in .tf files.

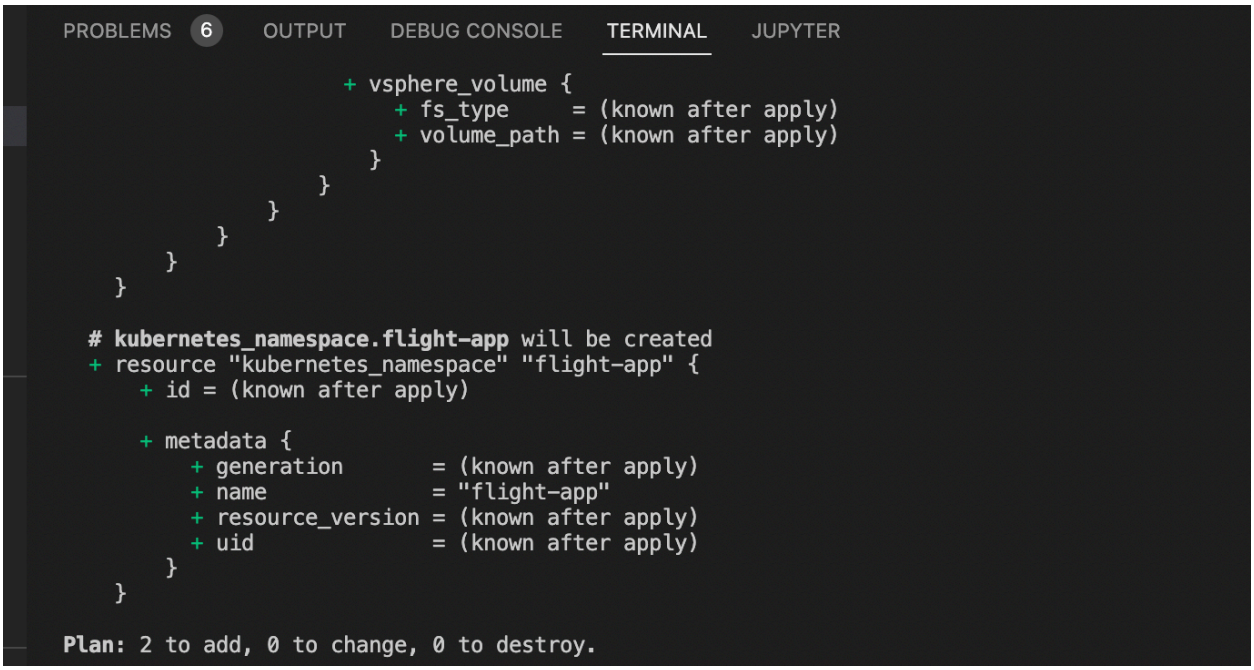
Git

-Add provider.tf file to manage the Kubernetes API.
-Add k8.tf defining the resource.

-Run terraform init to download modules and registry.
-Run terraform validation to check the configuration.
-Run terraform plan for dry run.



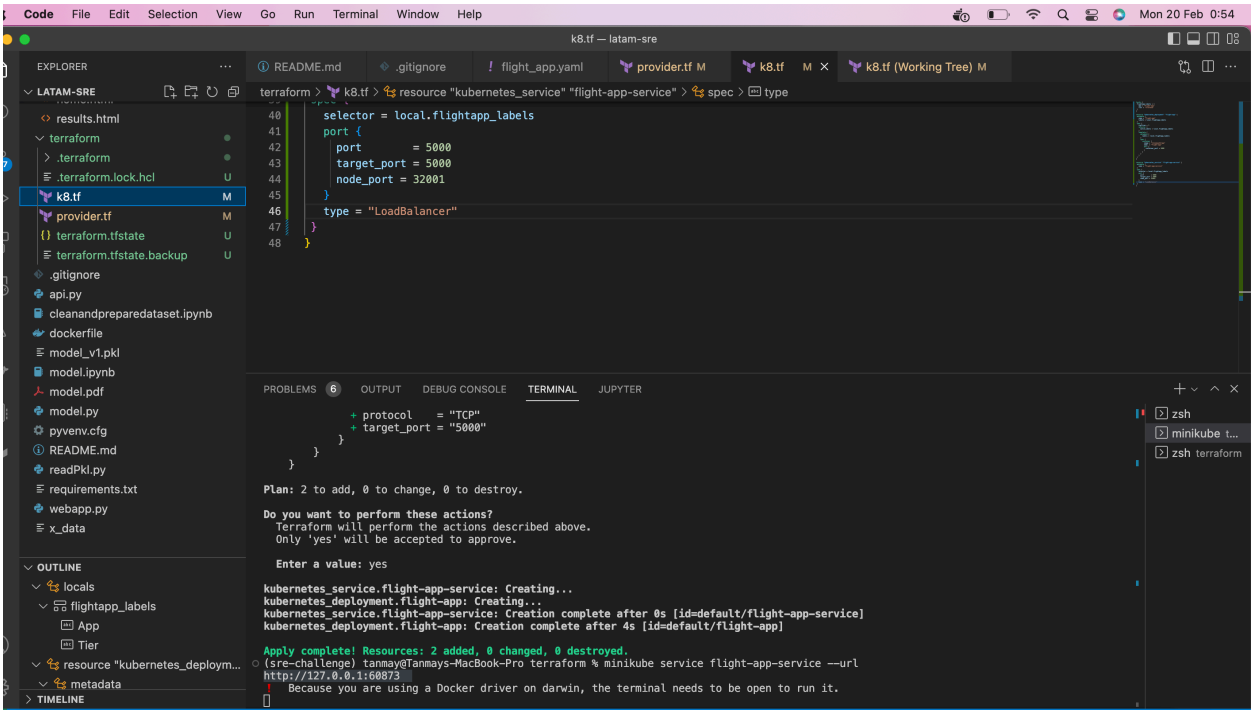
Terraform plan gives output-



-Now apply using terraform apply

```
PROBLEMS 6   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER

                    + vsphere_volume {
                        + fs_type     = (known after apply)
                        + volume_path = (known after apply)
                      }
                  }
                }
              }
            }
          }

  # kubernetes_namespace.flight-app will be created
  + resource "kubernetes_namespace" "flight-app" {
      + id = (known after apply)

      + metadata {
          + generation       = (known after apply)
          + name             = "flight-app"
          + resource_version = (known after apply)
          + uid              = (known after apply)
        }
    }

Plan: 2 to add, 0 to change, 0 to destroy.
```

Result:



Conclusion:

Now we can manage our infrastructure using terraform easily. This doesn't affect the performance of the application as terraform manages underlying infrastructure which is Kubernetes in this case.

Implementing Monitoring using Prometheus and Grafana

We can leverage prometheus for monitoring our infra and application .

- Install Prometheus using helm:

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

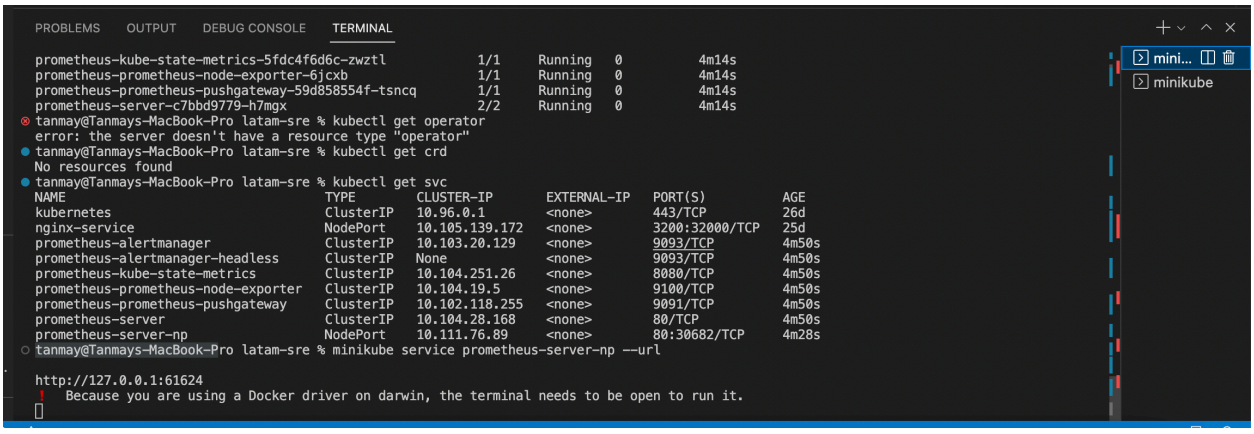
$ helm install prometheus prometheus-community/prometheus

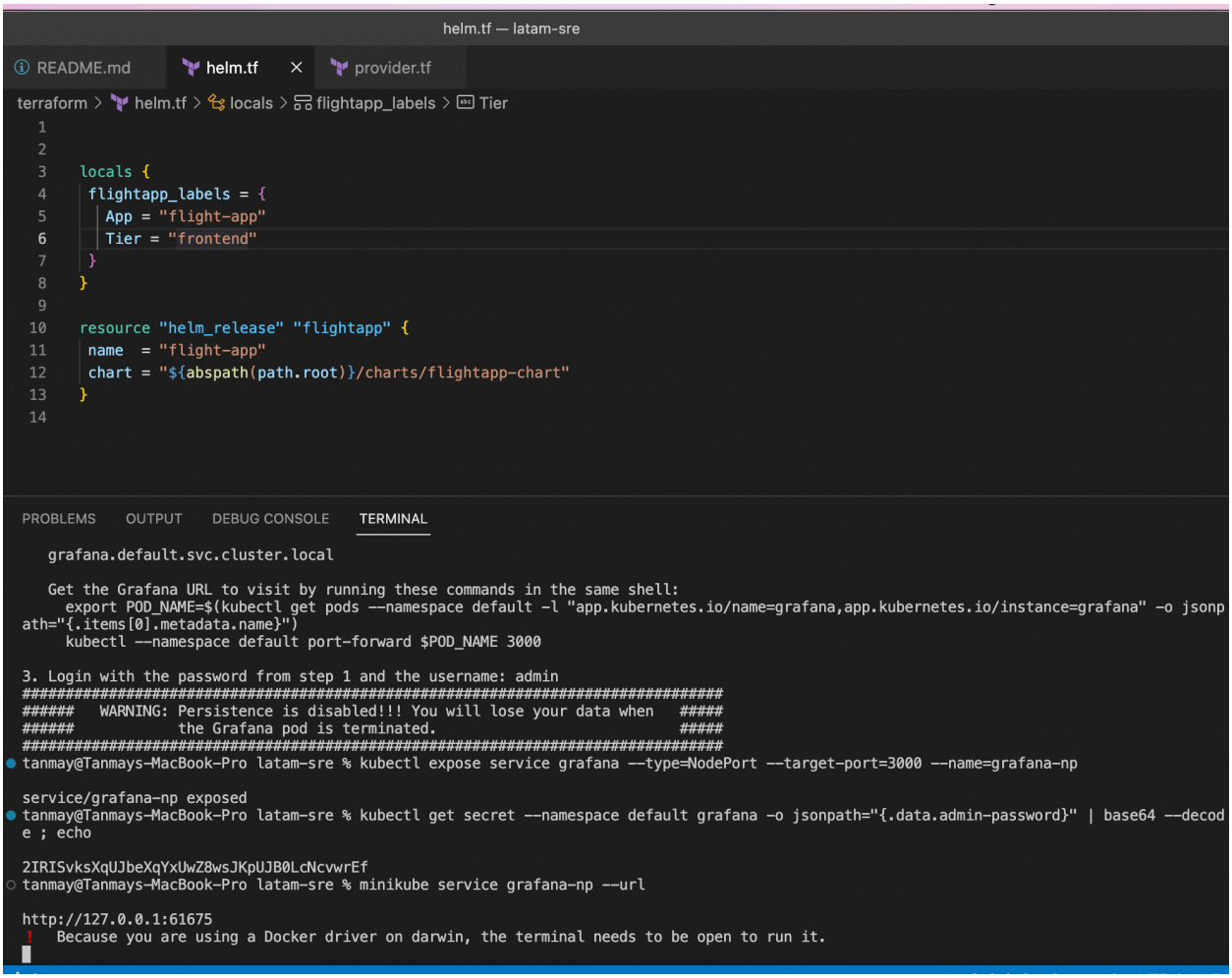
- Install Grafana using helm:

$ helm repo add grafana https://grafana.github.io/helm-charts
$ helm install grafana stable/grafana

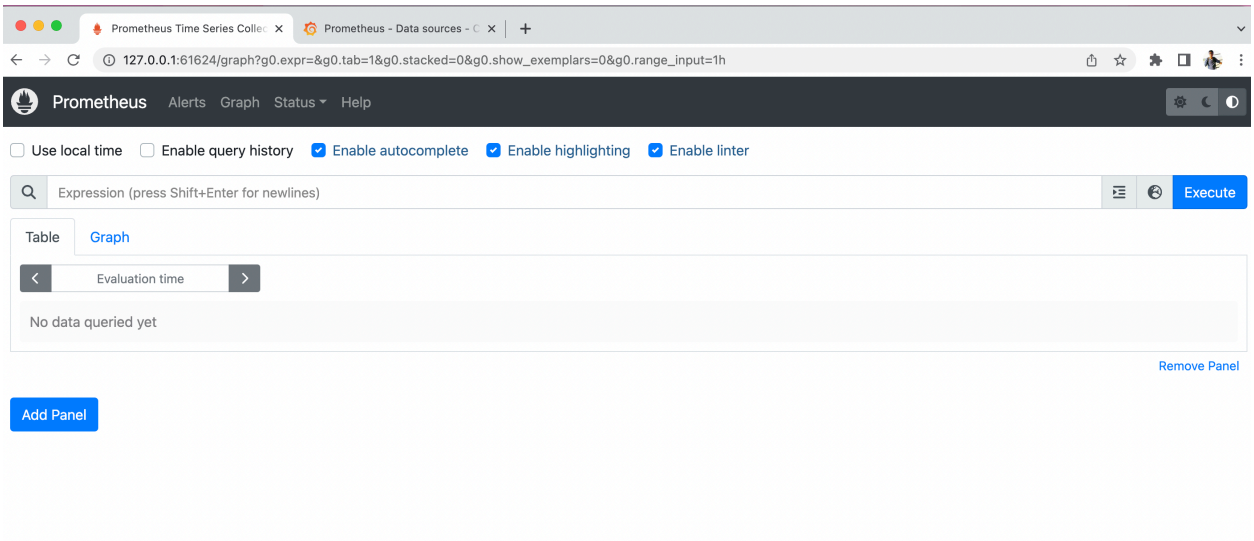
- Expose the services for prometheus and grafana to access the dashboard.
- Follow instruction for the admin credential for grafana dashboard.
- Add prometheus datasource in grafana.

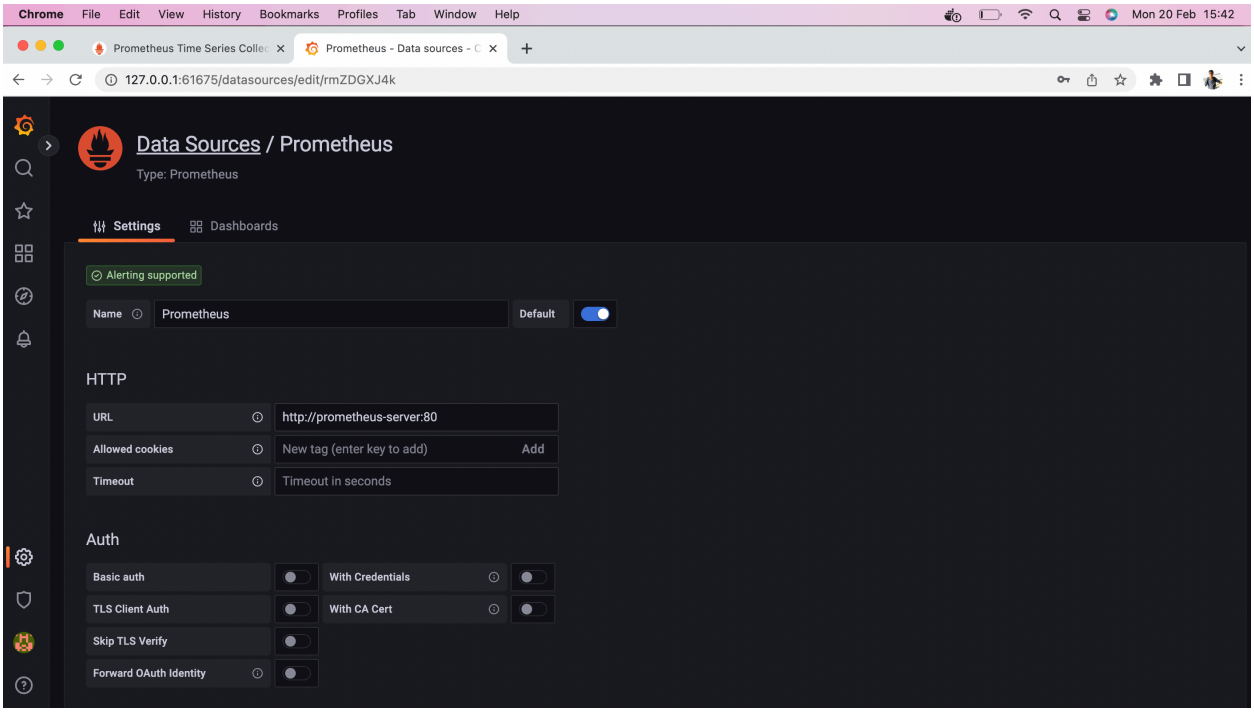
- Access Prometheus and grafana dashboards.
- Fetch service url for both the dashboards.





- Access the url’s





SLO's

- SLOs: Service Level Objective
  - What you have internally set as a target, driving your measuring threshold (for example, on dashboards and alerting). In general, it should be stricter than your SLA.
  - Example: "99.9%" availability (the so called "three 9s").
  - Keyword: *thresholds*
- SLIs: Service Level Indicators
  - What you actually measure, to ascertain whether your SLOs are on/off-target.
  - Example: error ratios, latency
  - Keyword: *metrics*

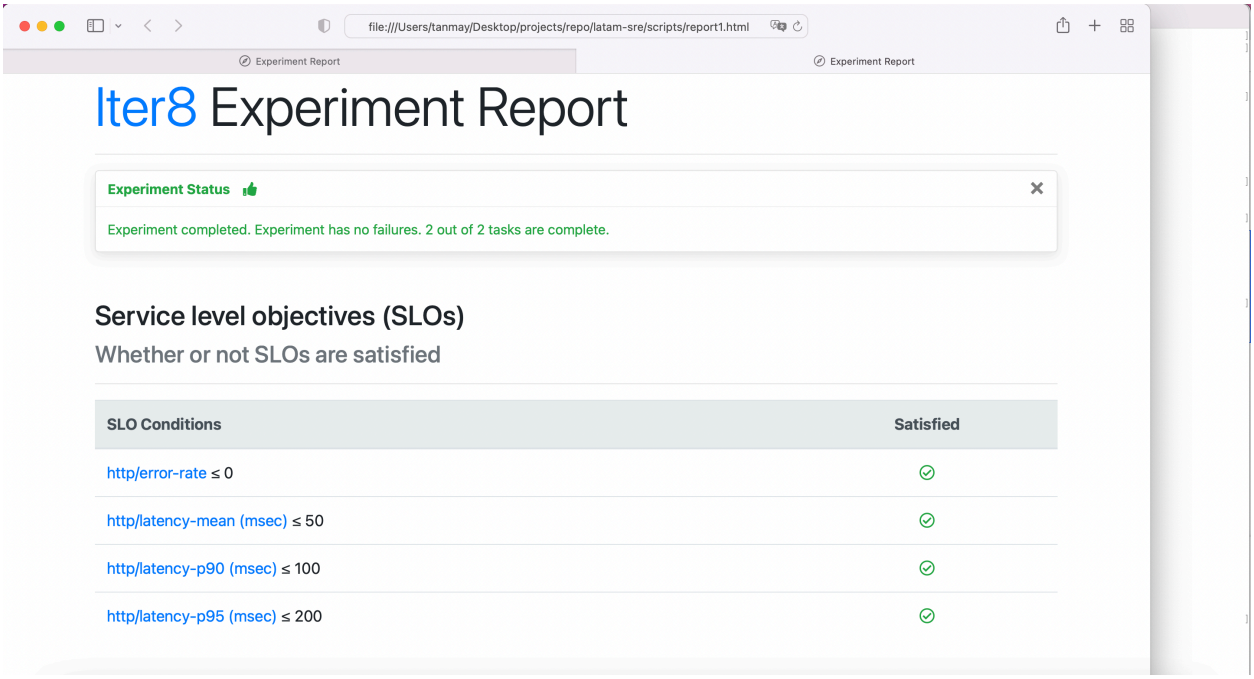
-Will use iter8 tool to find the SLO-

```
iter8 k launch -c load-test-http --noDownload \
--set url=http://127.0.0.1:60093 \
--set SLOs.http/error-rate=0 \
--set SLOs.http/latency-mean=50 \
--set SLOs.http/latency-p90=100 \
--set SLOs.http/latency-p95=200
```

Using this configuration, validates that the service satisfies the following SLOs.

- Error rate is 0
- Mean latency is under 50 msec
- 90 percentile latency is under 100 msec
- 95 percentile latency is under 200 msec

```
tanmay@Tanmays-MacBook-Pro charts % iter8 k assert -c completed -c nofailure -c slos --timeout 120s
INFO[2023-02-21 00:26:52] inited Helm config
INFO[2023-02-21 00:26:52] experiment completed
INFO[2023-02-21 00:26:52] experiment has no failure
INFO[2023-02-21 00:26:52] experiment does not involve any SLOs
INFO[2023-02-21 00:26:52] SLOs are satisfied
INFO[2023-02-21 00:26:52] all conditions were satisfied
tanmay@Tanmays-MacBook-Pro charts % ls
charts                  iter8lib                load-test-grpc          load-test-http          slo-validation-istio
tanmay@Tanmays-MacBook-Pro charts % iter8 k report

INFO[2023-02-21 00:27:08] inited Helm config

Experiment summary:
*******************

  Experiment completed: true
  No task failures: true
  Total number of tasks: 1
  Number of completed tasks: 1

Latest observed values for metrics:
***********************************

  Metric                     |value
  -------                    |-----
  http/error-count           |0.00
  http/error-rate            |0.00
  http/latency-max (msec)    |4.33
  http/latency-mean (msec)   |0.98
  http/latency-min (msec)    |0.25
  http/latency-p50 (msec)    |0.82
  http/latency-p75 (msec)    |1.36
  http/latency-p90 (msec)    |1.96
  http/latency-p95 (msec)    |3.00
  http/latency-p99 (msec)    |4.00
  http/latency-p99.9 (msec)  |4.30
  http/latency-stddev (msec) |0.83
  http/request-count         |100.00


tanmay@Tanmays-MacBook-Pro charts % iter8 k log

INFO[2023-02-21 00:27:32] inited Helm config
INFO[2023-02-21 00:27:32] experiment logs from Kubernetes cluster       indented-trace=below ...
  time=2023-02-21 06:26:05 level=info msg=task 1: gen-load-and-collect-metrics-http : started
  time=2023-02-21 06:26:17 level=info msg=task 1: gen-load-and-collect-metrics-http : completed
```



Application Security

Endpoints must be authenticated before they are allowed to make requests in an application. Authentication means that the endpoint has an existing session and is unique to a specific user.

There are several ways to create an authentication layer in web applications one of them is Token Based Authentication. However, it is worth noting that token-based authentication scales better than that of a session because tokens are stored on the client-side while the session makes use of the server memory so it might become an issue when there is a large number of users using the system at once.

- Easy to implement.
- Provides a good level of security (by industry standards).
- Fast and performant.

What is a Security Token?

In a broad way a token is a "number that proves something", for example: When you finish making a bank transfer, the bank sends a confirmation "token" serves as proof to validate that the transaction exists and it's valid. That confirmation number could be also called a confirmation token.

Tokens used for authentication need to be more that normal just numbers, they need to be almost impossible to fake, predict or break.

- Non-consecutive, that will make them very predictable, hackers will guess the next one.
- Infinite (almost)
- Non-reusable: There are cases of re-usable tokens, but in general once a token is generated no one else should every use it but you.
- Validatable: The token must follow some hidden pattern (encryption) that allows validating the token without compromising the owner or author.

The most simple way to implement autentication in your database and API:

1. Create a `User` table/model that represents every user inside your application.
2. That User table must contain email and password for every user.
3. Create one api endpoint called `POST /token` that generates a token only if it receives an email and password that maches in the database.
4. The `POST /token` endpoint will return the token to the front-end if everything is ok.
5. Then, on every other endpoint in your database you will have to validate if the token exists in the request header and if it does you will have to validate it.

Every token is a session

The moment you generate the token you can decide if you want it to expire, same way web sessions expire when you log in into your online bank account.

When a client successfully authenticates it will receive that unique token and it will be able to attached to the request headers of every request it makes from that moment on, that token will be the "User session".

It is recomended to save that token in the cookies or localStorage of your front-end application.