National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cloud Computing

Vijayanand Somavaram
Student ID: x22147802

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vijayanand Somavaram
Student ID:	x22147802
Programme:	MSc in Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	31/01/2024
Project Title:	Configuration Manual
Word Count:	1020
Page Count:	14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	S.Vijayanand
Date:	30th January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vijayanand Somavaram
x22147802

This project can be divided into two phases

- Phase One: Data collection phase
- Phase One: Prediction phase

Due to resource limitations for this research project, please note that all the AWS services should be set in the same AWS Region.

1 Collecting Latency Data

In this phase, a methodology to collect the time taken to execute queries by DynamoDB will be collected. This methodology will include setting up AWS Services like EC2, API Gateway, Lambda, DynamoDB, and CloudWatch.

1.0.1 Step One: Create a Lambda function

Open the AWS console, select AWS Lambda from the search, and click on the "Create Lambda" option. Enter the name of the Lambda function, and select the runtime as Python 3.11 as shown in 1. Click on the Create option.

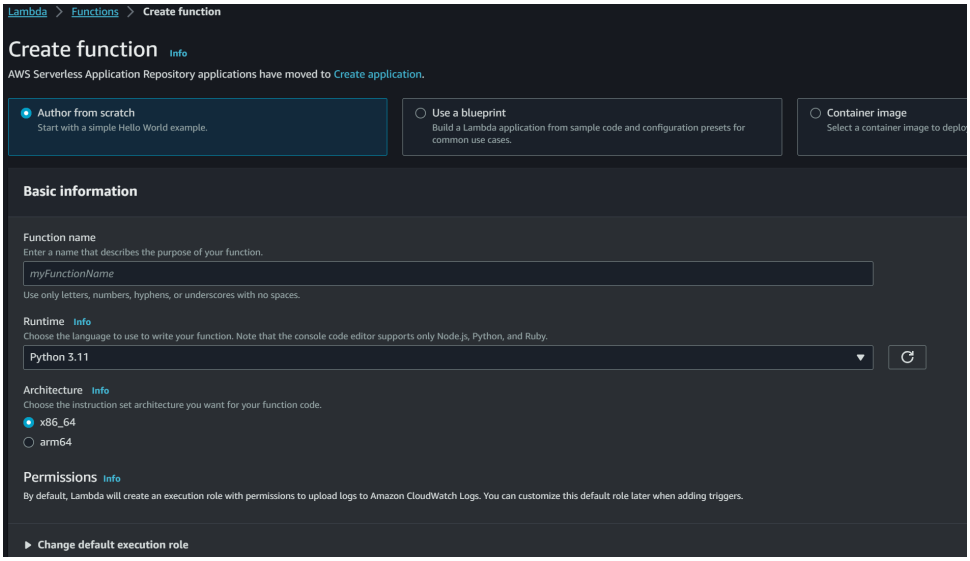

Figure 1: Create a lambda function

Remove the existing code from the code editor. From the submitted code artifacts copy the "lambdaPython" code onto the editor as shown in 2. Click On Deploy

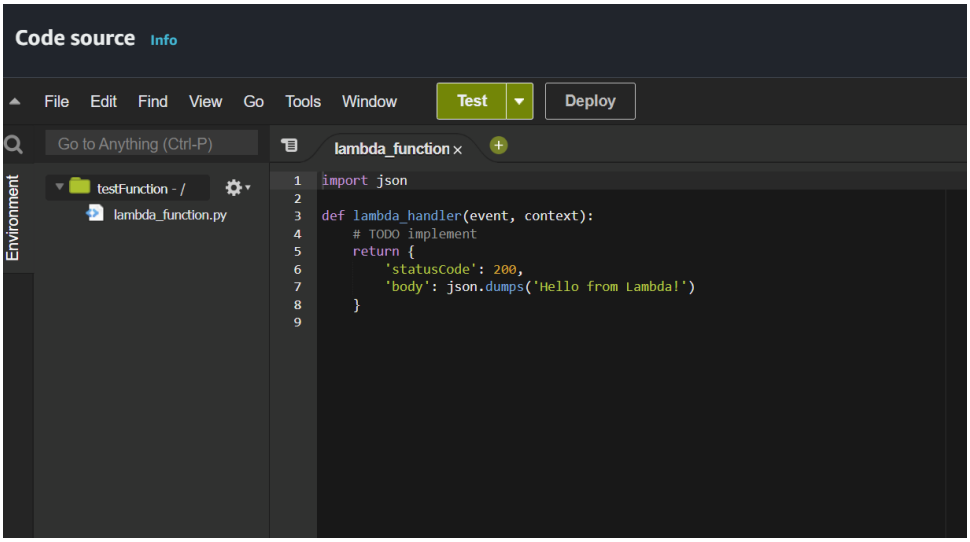

Figure 2: Add the Python Code

1.1 Step Two: Create API Gateway

Search for API Gateway, from the console, and click on create. As shown in 3, select HTTP API and click on the Build option. Enter the name of the API, and then click on "review and create" as shown in 4. As Shown in 5 enter the name of a route, by clicking on the "Create" option. From 6, select the "POST" route. Then from the route details click on "Attach Integration". As shown in 7, click on the "create" button. Then on the next window as shown in 8, under the integration target select the integration type as "Lambda function", under the integration details select the "Lambda function" created from step one and save. From the left nav as shown in 9, select the "Stages" option. On the next window as shown in 10 enter the name of the stage and save the information. Then on the next window as shown in 11, click on the "Deploy" option to deploy the API. From the left nav as shown in 12, select the "CORS" option to avoid CORS errors. On the next window as shown on 13 add the values and save them. From the left nav as shown in 14, select the name of your API, and the URLs under the stages for "your api name" is the API URL that will be used in later stages.

1.1.1 Step Three: Setup Web Application

Launch an EC2 instance from the AWS console, for this application "t2.micro" ubuntu instance will be enough. Connect to the instance through SSH client. For this, an application called MobaXterm has been used. It is a tool for remote computing. Follow the instructions as displayed on SSH client and connect to EC2 instance as shown on 15. Then install apache2 server on the instance using the commands

- "sudo apt-get install apache2"
- "cd var"
- "cd www"
- "cd html"

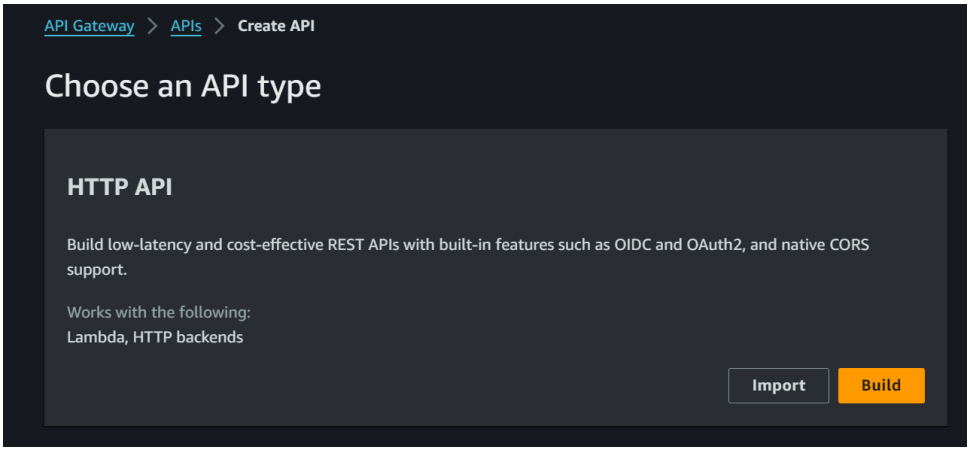

Figure 3: Create API Gateway

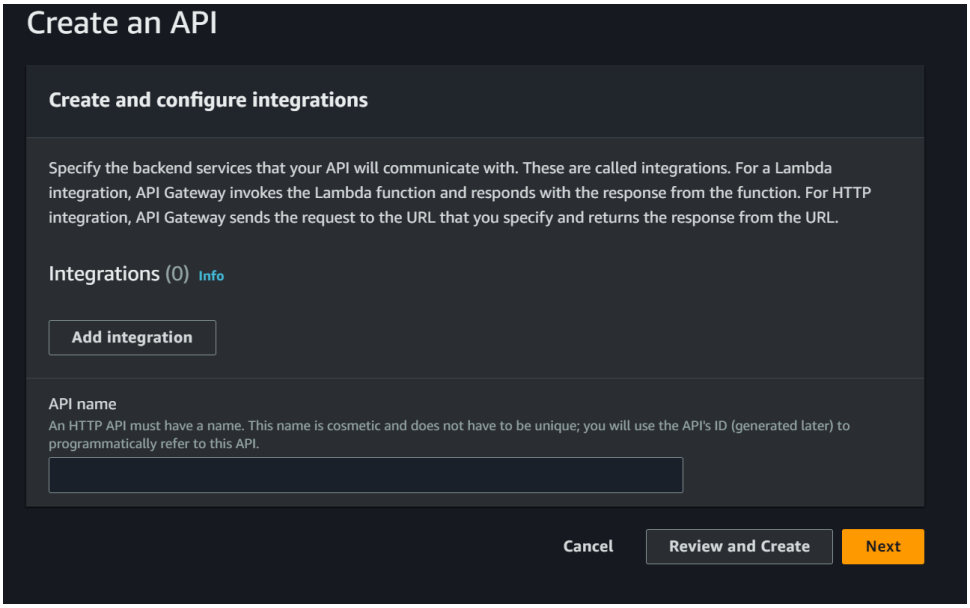

Figure 4: Api Gateway

Open another window on Mobaxterm. Download the web application zip on the local folder where the ".pem" file is present. Enter the following commands

- "scp -i /path/to/your/.pemkey /copy/from/path.zip user@server:/copy/to/path"
- "sudo apt-get install unzip"
- "unzip yourfile.zip"
- "cd assets"
- "cd js"
- "sudo nano server.js"
- Copy and paste the API URL from step 2 in the place of URL as shown in 16

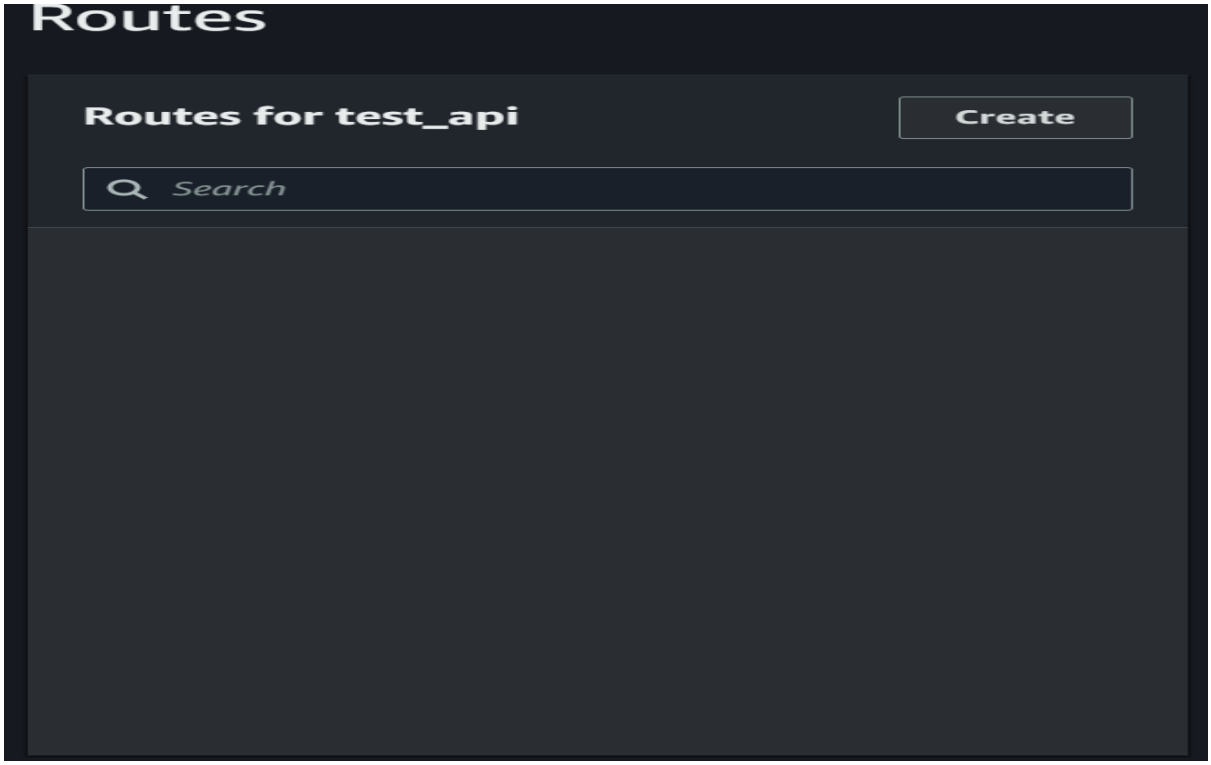

Figure 5: Api Gateway

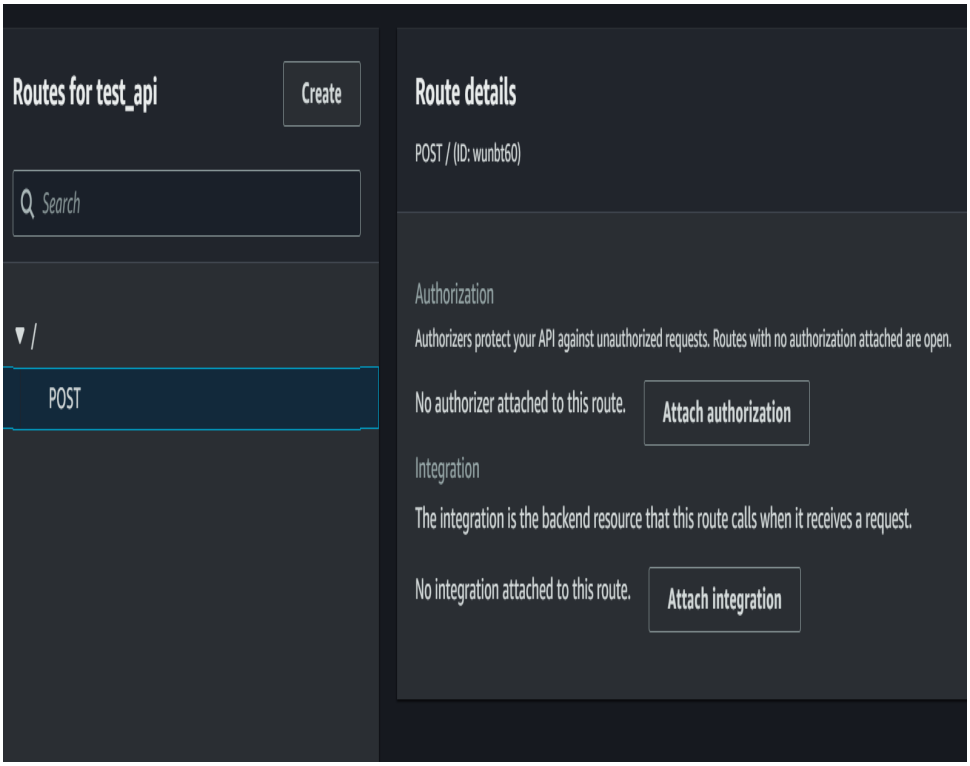

Figure 6: Api Gateway

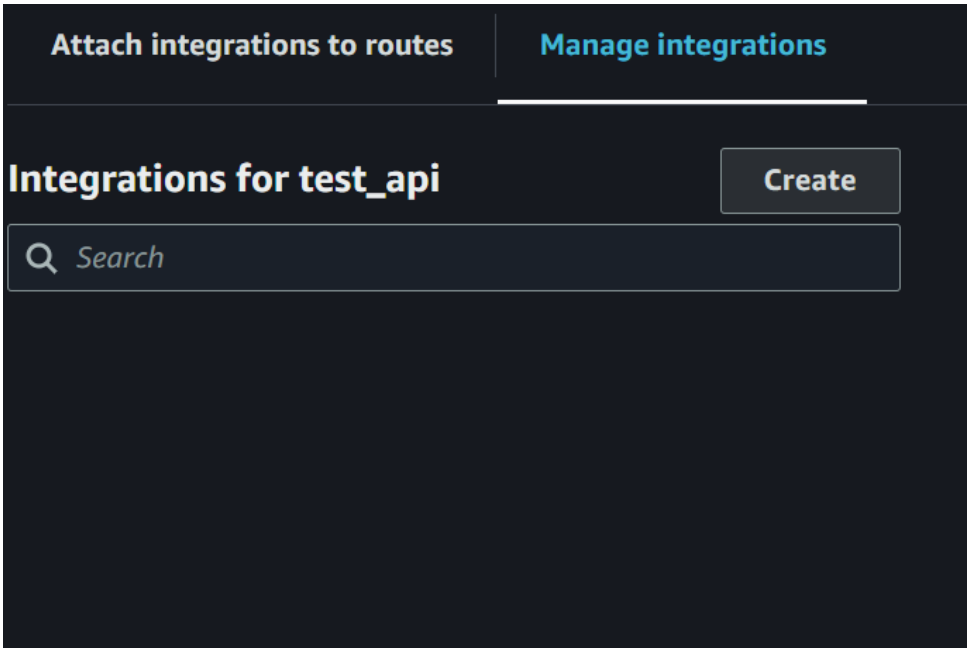

Figure 7: Api Gateway

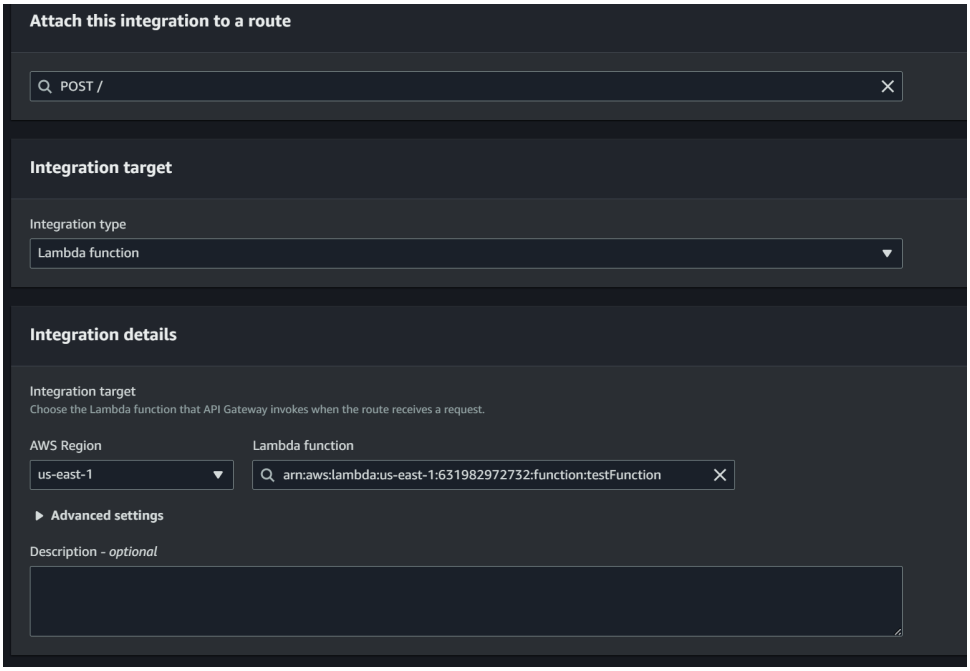

Figure 8: Api Gateway

- press "CTRL + O" press enter
- press "CTRL + x" to exit

With this the application is up, click on the EC2 Instance id, and open the public address, change the URL on the browser from "https" to "http". The application should be as shown in 17

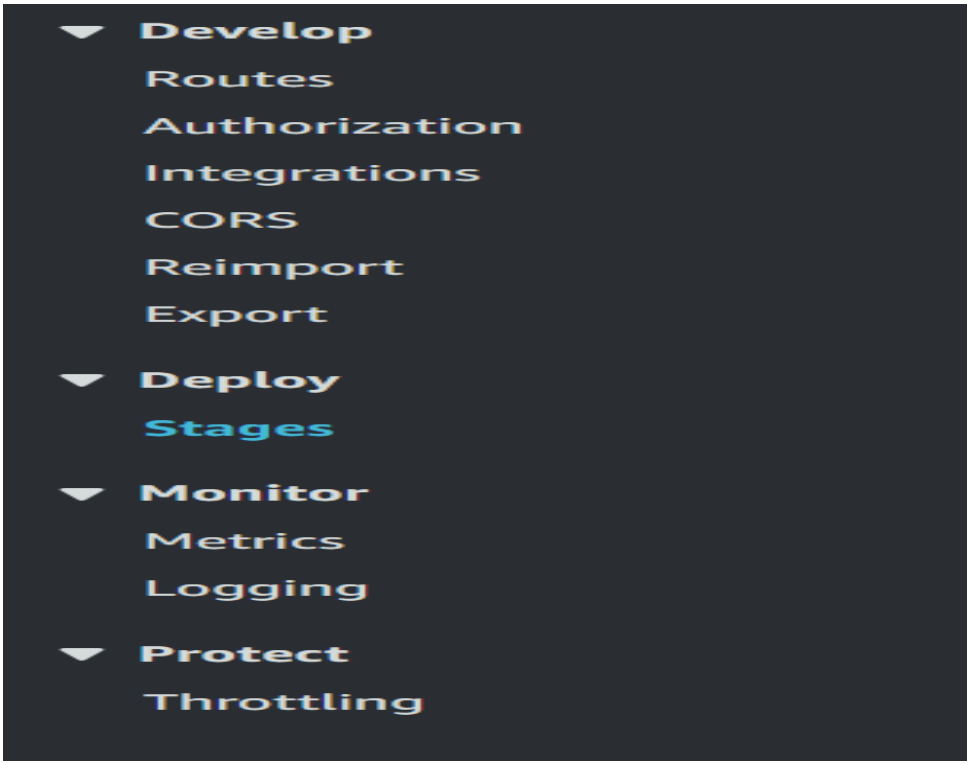

Figure 9: Api Gateway

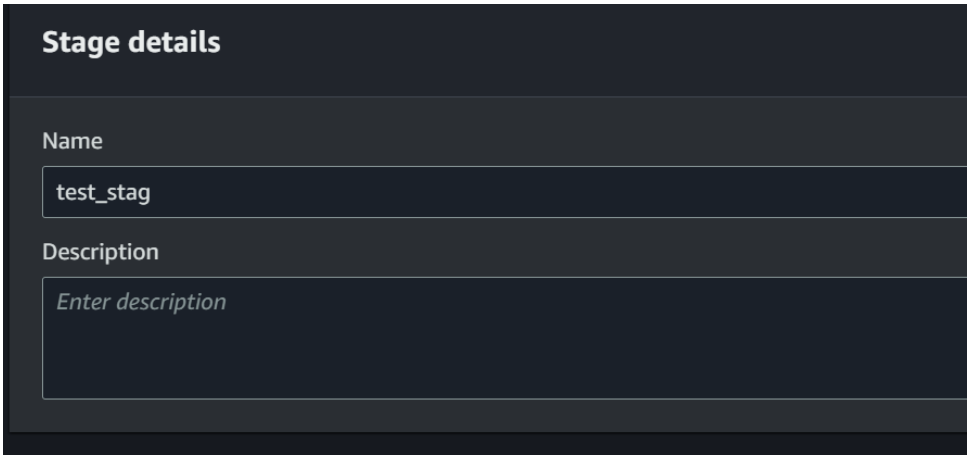

Figure 10: Api Gateway

1.1.2 Step 4: Setup DyanmoDB:

From the AWS Console enter DyanmoDB, and click on the "Create table" button. Enter the name of the table, "test partition key", under table settings select "Customize settings", and under "Read/write capacity settings" select "Provisioned" For both Read capacity and Write capacity turn off Auto Scaling. For now, enter minimum capacity under read capacity units as 51 and 408, scroll to the bottom, and click on the "Create Table" button as shown in images 18 and 19

From the Lambda function create in Step one, open the editor to connect it with

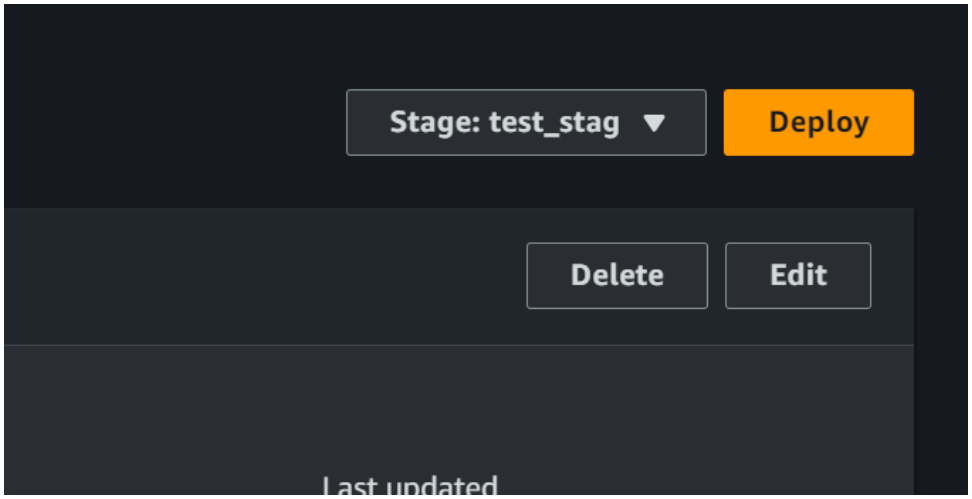

Figure 11: Api Gateway

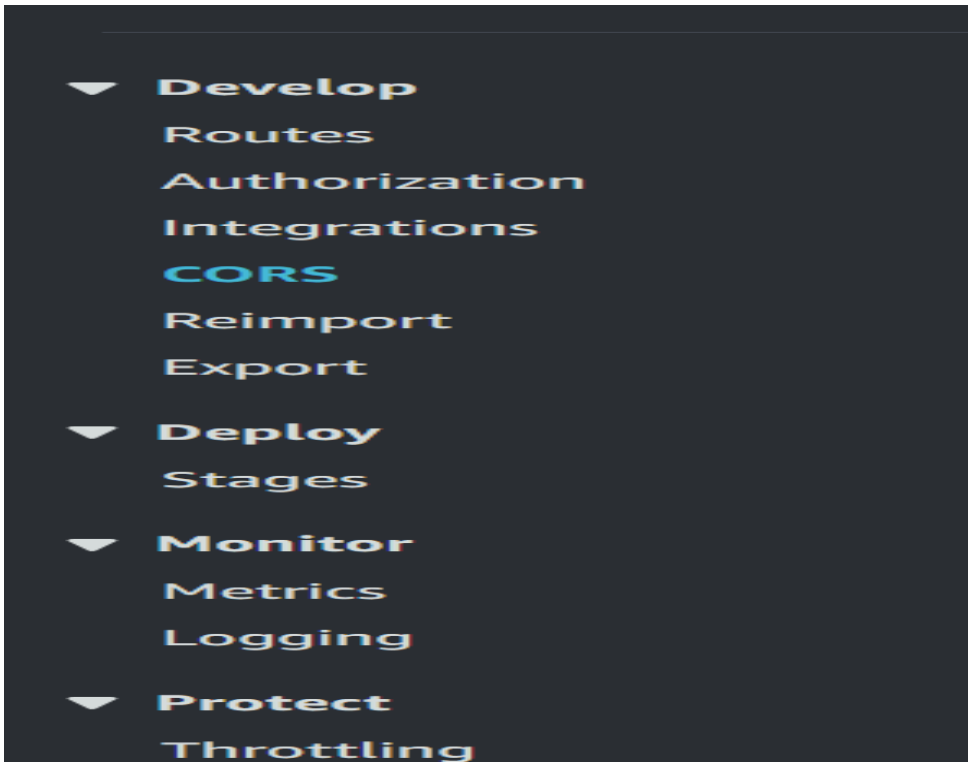

Figure 12: Api Gateway

DyanamoDB. To do so, enter the name of the table for the variables shown in 20. In this case, enter the same name for both the "table" and "writeTable" variables. The connection now is completed. The DynamoDB table now created is capable of handling 24KB or read-and-write workloads. To test it click on "Read Payload size 24kb" or "Write payload size 24kb" from the app. By doing so the application will send continuous requests to DynamoDB for 250 seconds.

To check the logs and query times from the lambda console click on the "Monitor" option and then click on "View cloudwatch logs" and then by clicking on log files the

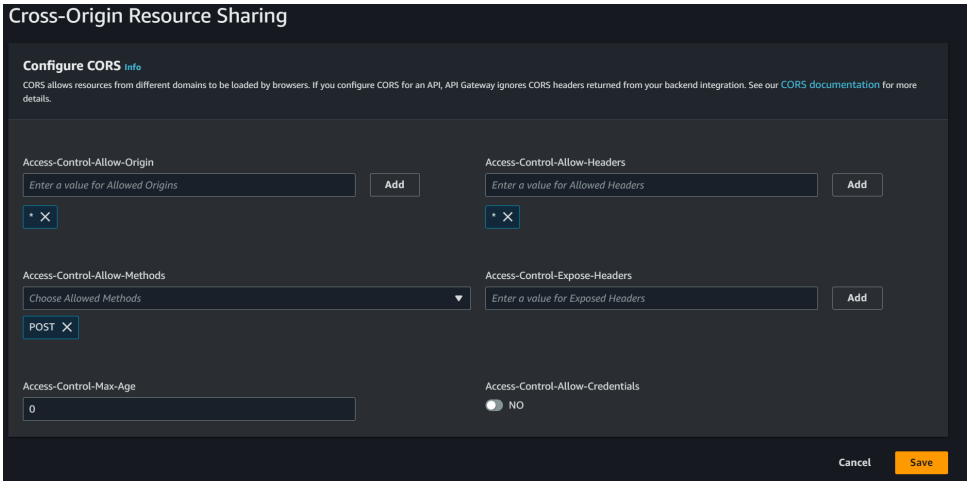

Figure 13: Api Gateway

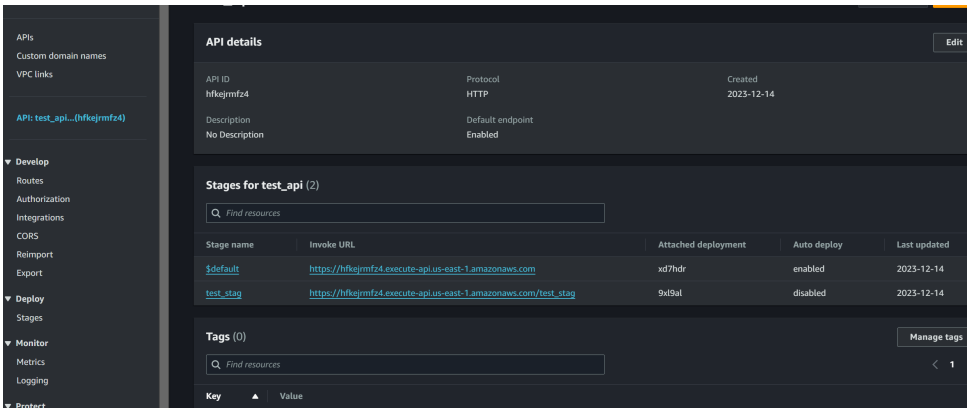

Figure 14: Api Gateway

data can be viewed as shown in 21

1.1.3 Conclusion

This way the experiment was conducted for 49 KB, and 98 KB for 2000, 3000, 4000, and 5000 requests per minute. Data was collected from the logs and was formatted for the machine learning model training.

2 Prediction Phase

After collecting the data from phase one, the data was formatted which have been submitted in the artifacts by name, "24KBdata" and "49KBdata" and "98KBdata".

2.0.1 Step one: Model training:

As shown in images 22 and 23, the notebook code can be downloaded from artifacts.

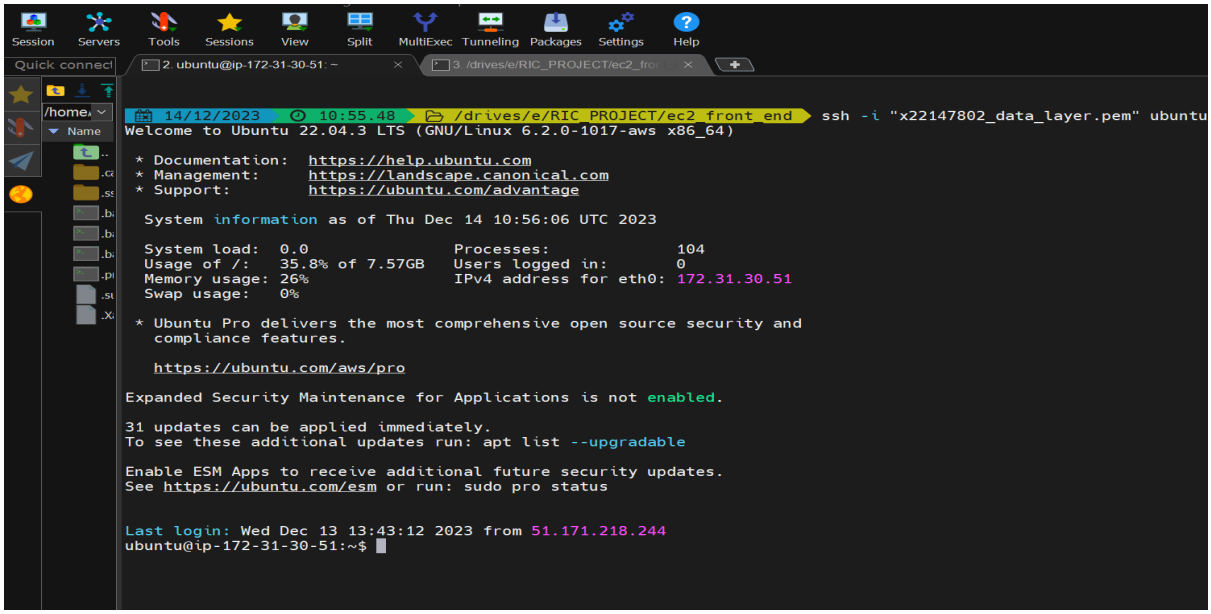

Figure 15: Mobaxterm Application

```
$(document).ready(function(){

    $('.post').on('click',function(e){
        e.preventDefault();
        var payloadSizeInKB= parseInt($(this).attr('data-size'));
        var payloadType= $(this).attr('data-type');
        let seconds=0;

        const intervalId= setInterval(() => {
            $.ajax({
                url:'https://jao60bhgh6.execute-api.eu-west-1.amazonaws.com/',
                type:'POST',
                data:{value:payloadSizeInKB, loadType:payloadType},
                dataType: 'json',
                headers:{
                    'Access-Control-Allow-Origin': '*',
                    'Access-Control-Allow-Credentials':'true',
                    'withCredentials' : 'true'
                },
                crossDomain: true,
                encode: true
```

Figure 16: Mobaxterm Application

2.0.2 step two: Integration and Predictions:

For Predictions, a web application was created using the Streamlit framework. The code can be downloaded from the artifacts. Just like the steps followed in phase one, step three, an ec2 instance should be launched. The ec2 security port should be open to port: 80. The downloaded code should be uploaded to ec2 after creating a folder on ec2 by any SSH tool. Enter the following commands

- ”sudo mkdir newFolder”

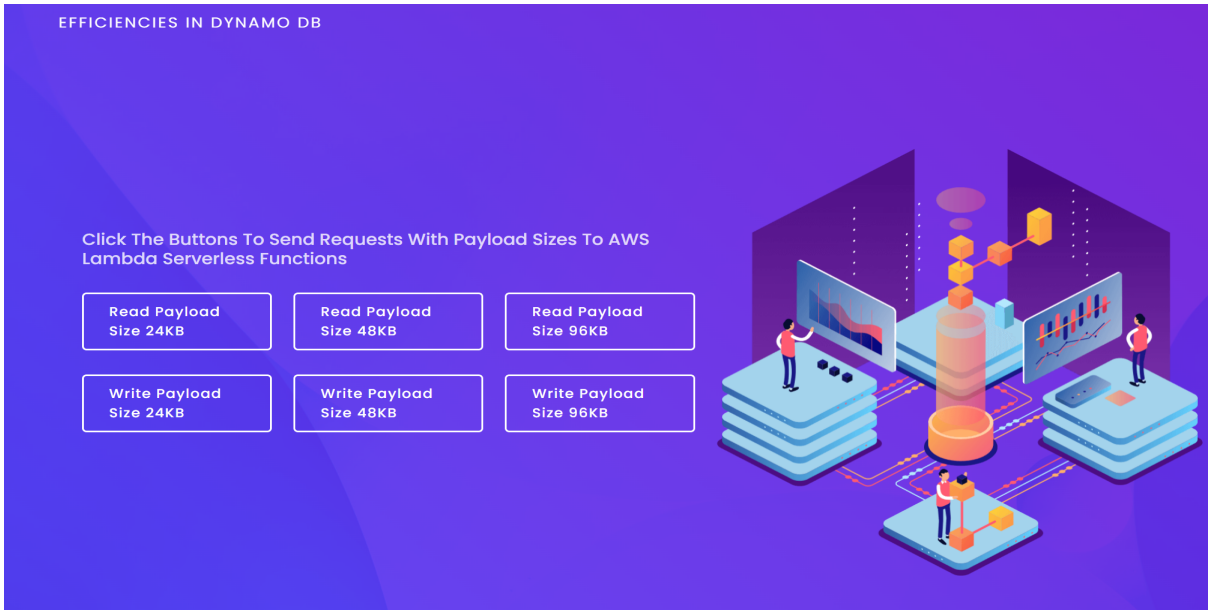

Figure 17: Web Application for data collection

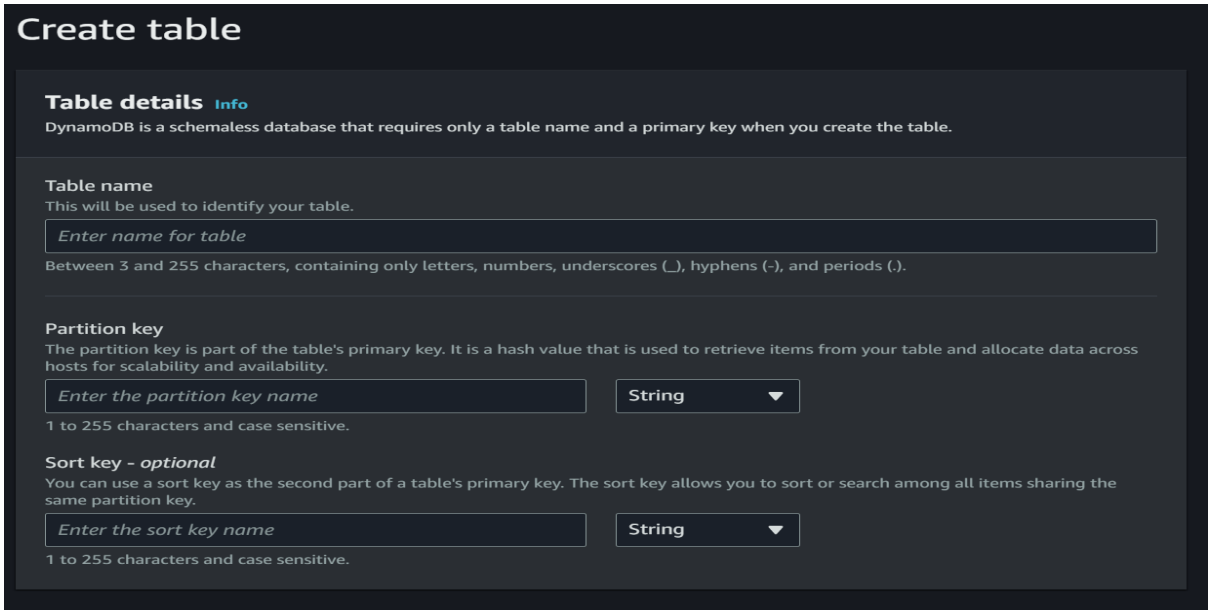

Figure 18: setup DynamoDB

- upload the code to this folder
- ”cd newFolder”
- ”sudo apt-get update”
- ”python3 -m venv env”
- ”cd env”
- ”cd bin”

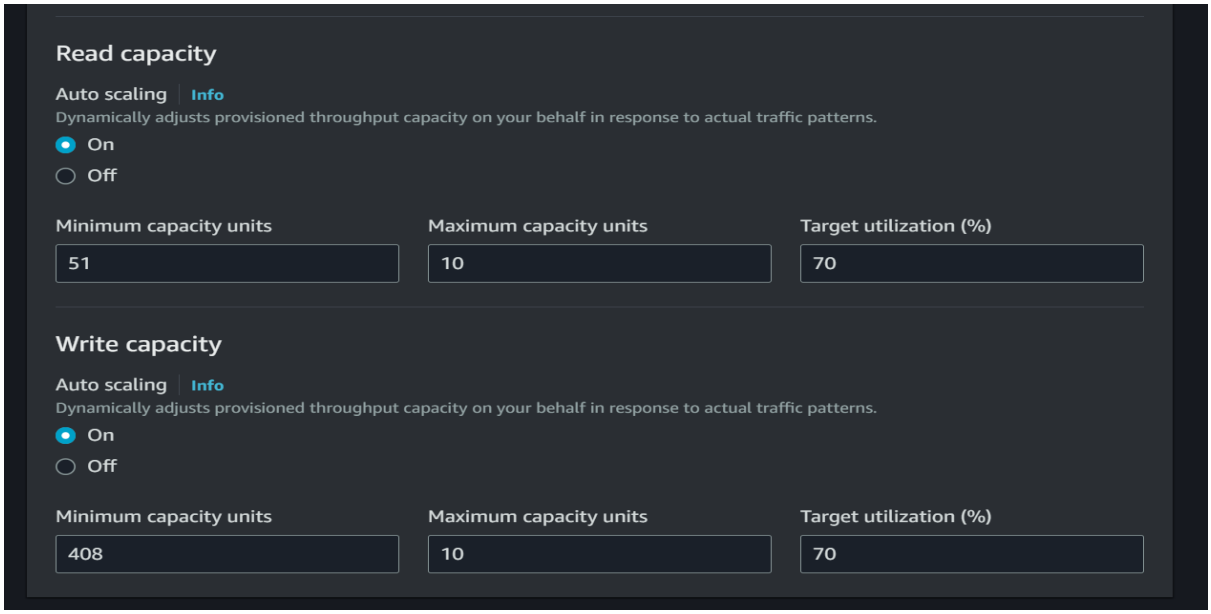

Figure 19: setup DynamoDB

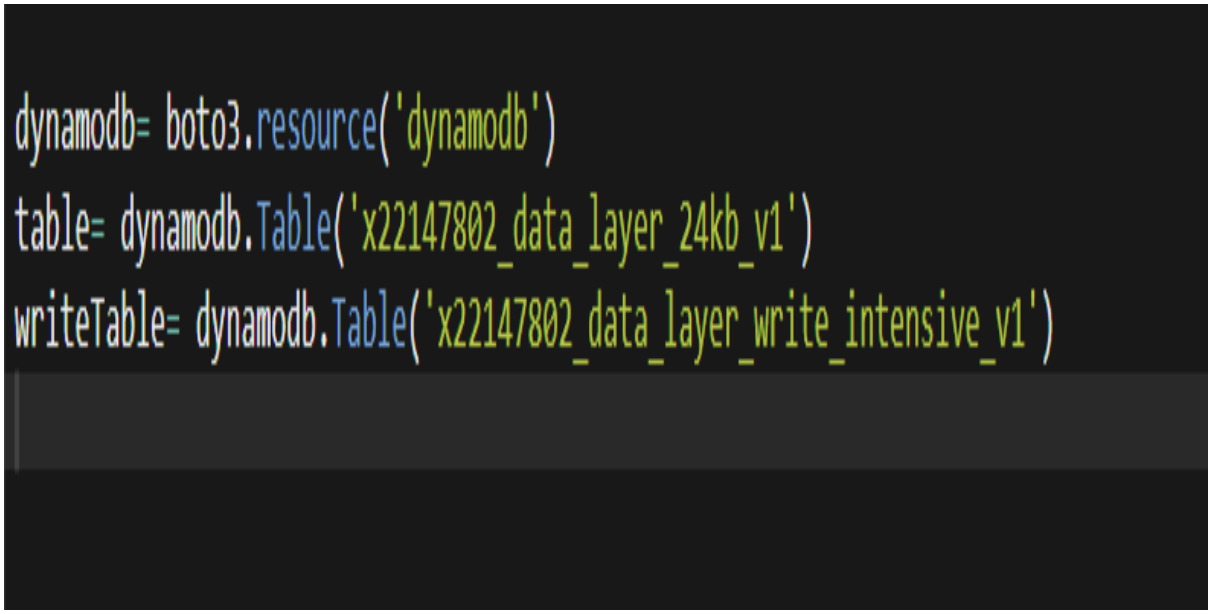

Figure 20: Link Table

- ”source activate”
- ”cd ..”
- ”cd ..”
- ”pip install” all the packages mentioned in image 24
- ”streamlit run streamlit.py –server.port=80”

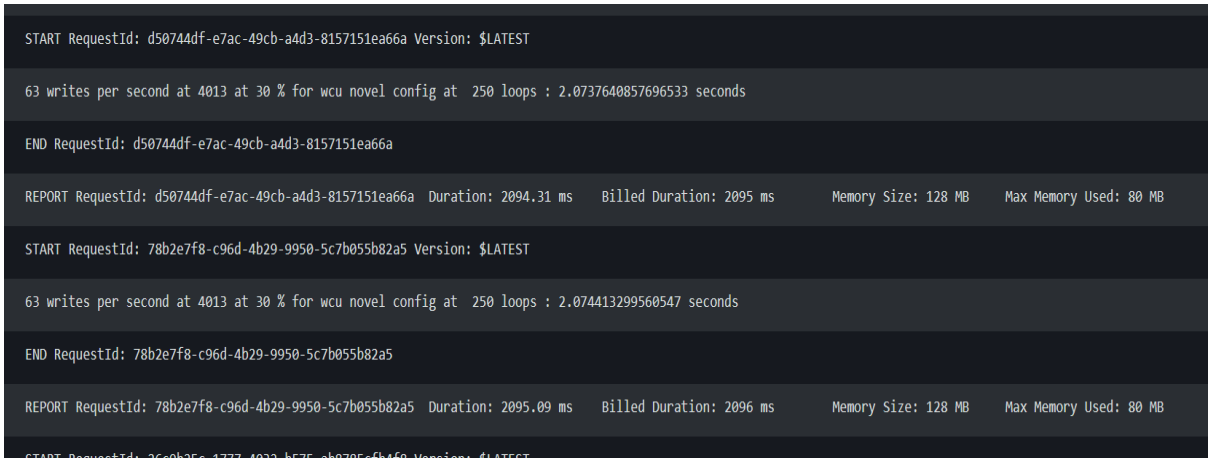

Figure 21: Cloudwatch

```python
# Import necessary libraries
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import joblib  # For model serialization
import sklearn
import numpy as np


# Load the data
data = pd.read_csv('/content/98KB_data.csv')
encoder = LabelEncoder()

# Feature selection
X = data[['itemSizeinKB', 'requestspersecond']]
y = data[['defaultConfiguration', 'expectedDefaultLatency', 'NovelConfiguration', 'expectedNovelLatency']]


# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(sklearn.__version__)
```

Figure 22: notebook

- Click the public URL of the Instance, change the URL from "https" to "http", and the application will run as shown in the image 25

```
[ ]  # Train the KNN model
     model = MultiOutputRegressor(RandomForestRegressor(random_state=42))
     model.fit(X_train, y_train)


     # Model Evaluation
     y_pred = model.predict(X_test)

     # Evaluate the model
     mse = mean_squared_error(y_test, y_pred, multioutput='raw_values')
     rmse = np.sqrt(mse)
     mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')
     r2 = r2_score(y_test, y_pred, multioutput='raw_values')

     print("Mean Squared Error for each output: ", mse)
     print("Root Mean Squared Error for each output: ", rmse)
     print("Mean Absolute Error for each output: ", mae)
     print("R-squared for each output: ", r2)

     # Save the model and encoder for future use
     joblib.dump(model, 'model_98.pkl')  # Saving the KNN model
     joblib.dump(encoder, 'label_encoder_98.pkl')  # Saving the LabelEncoder
```

Figure 23: Notebook

```
streamlit.py > ...
1    import streamlit as st
2    import pandas as pd
3    import joblib
4
5    # Function to load the appropr
```

Figure 24: python env

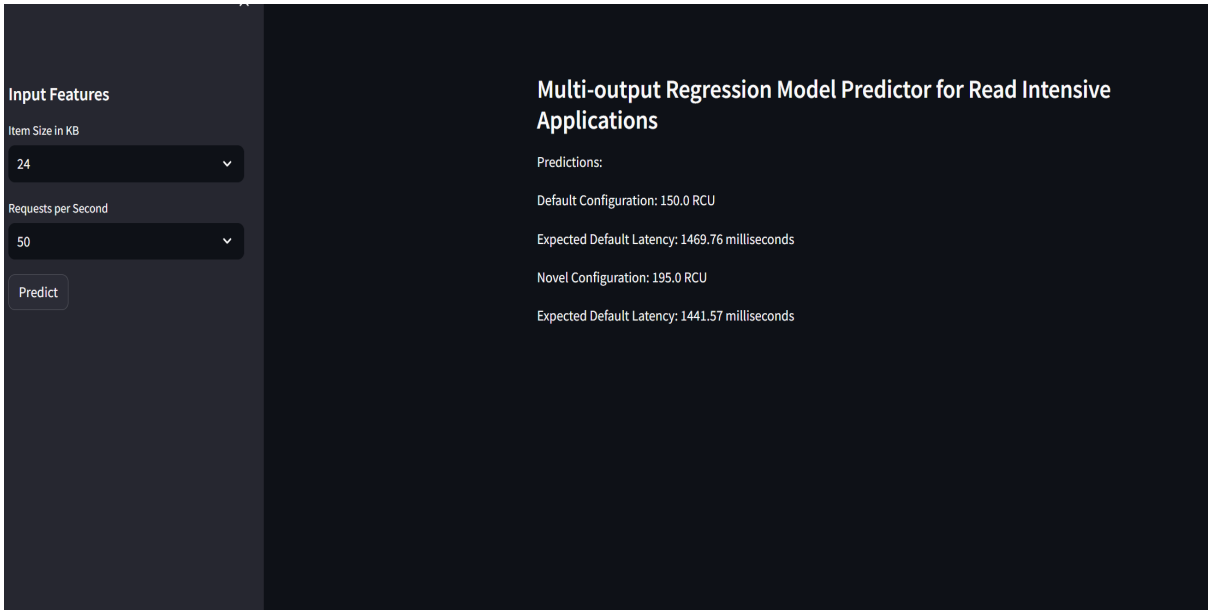

Figure 25: Application