National College of Ireland

Configuration Manual

MSc Research Project
MSc Cloud Computing

Balavignesh Kunkulagunta Sanghameswar
Student ID: 22220445

School of Computing
National College of Ireland

Supervisor: Rashid Mijumbi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Balavignesh Kunkulagunta Sanghameswar

Student ID: 22220445

Programme: Masters in Cloud Computing **Year:** 2023

Module: MSc Research Project

Lecturer: Rashid Mijumbi

Submission Due Date: 16-09-2024

Project Title: Fault Tolerance Optimization in Load Balancing using Multiple APIs in distributed cloud environment

Word Count: 307 **Page Count:** 10

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: Balavignesh Kunkulagunta Sanghameswar

Date: 16-09-2024

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

Balavignesh Kunkulagunta Sanghameswar
Student ID: 22220445

1 Introduction

This manual outlines the key steps and components involved in the research on optimizing fault tolerance in load balancing within a distributed cloud environment. The document is organized into three sections: Environmental Setup and Tools Setup used in this research.

2 Environmental Setup

2.1 Hardware Setup for Local Environment

Processor Requirement: Intel i3 or higher
Memory Requirement: 8 GB RAM

2.2 Programming Setup

Python: Version 3.10 or above

3 Tools Setup

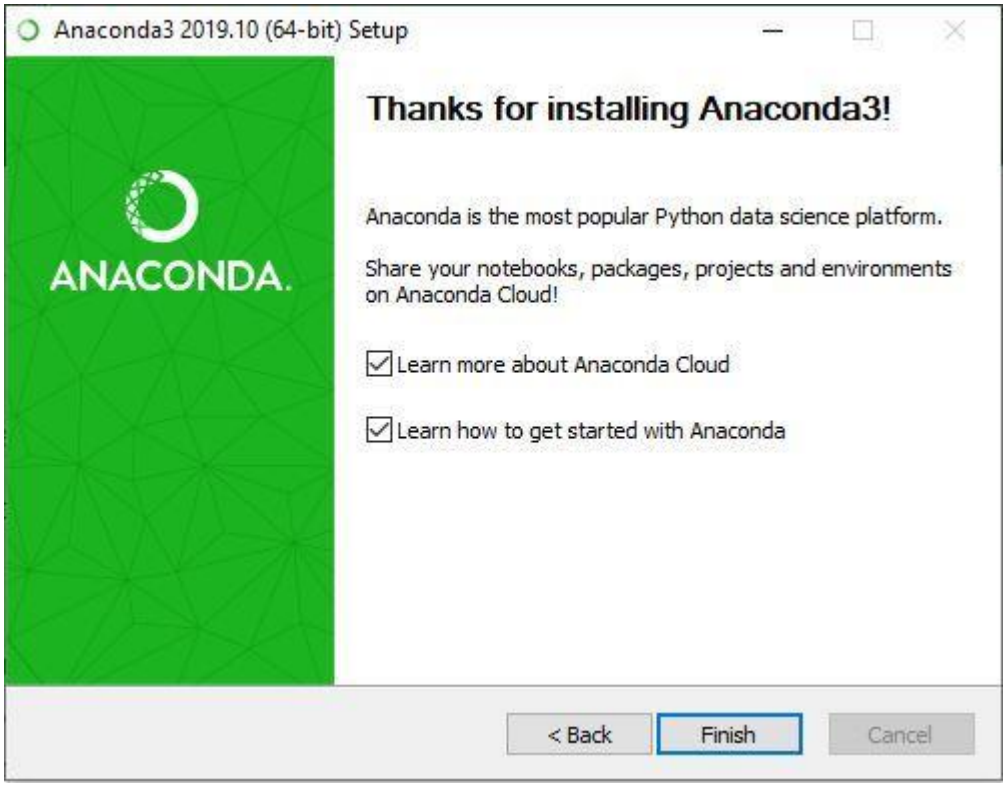
There are two tools required for this project

- Anaconda Navigator: For the creation of Load Balancing and fault tolerance program
- AWS Cloud – Lambda Function and Rest API (API Gateway)

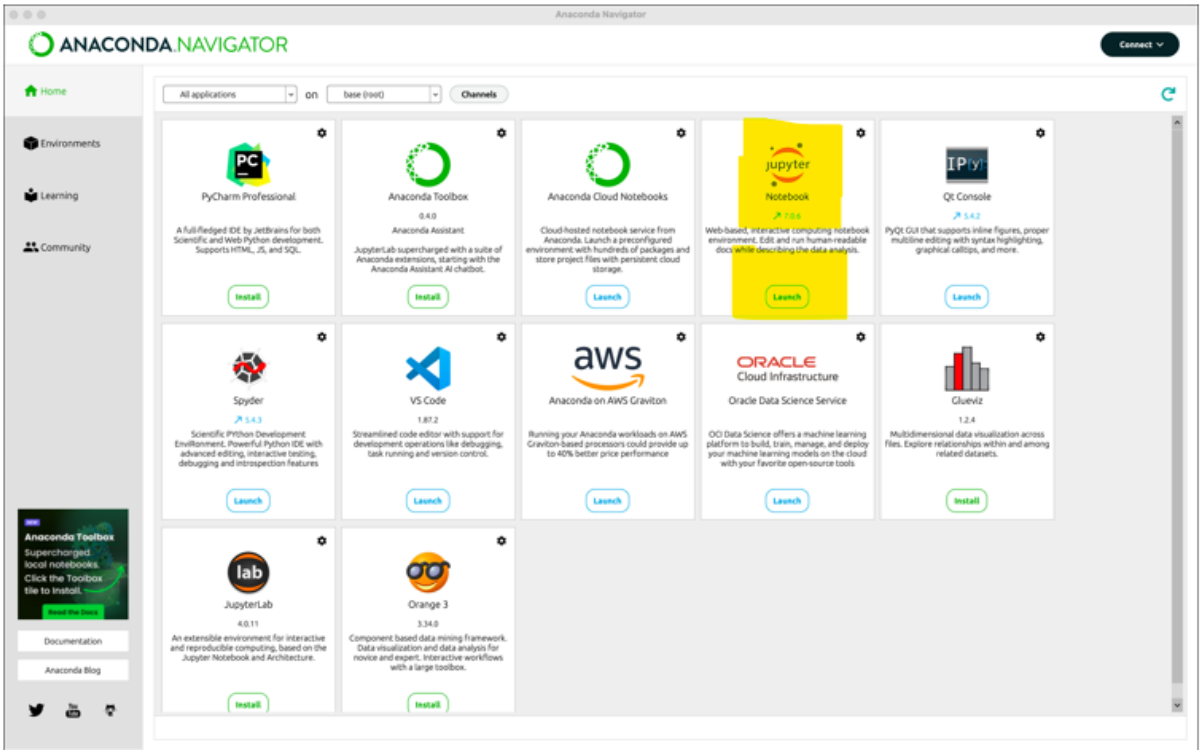
3.1 Installing Anaconda Navigator

Step 1: Begin by downloading Anaconda Navigator from this link.
https://www.anaconda.com/download

Step 2: After downloading, install Anaconda Navigator on your system.



Step 3: Once installed, open Anaconda Navigator. Within Anaconda Navigator, launch Jupyter Notebook.

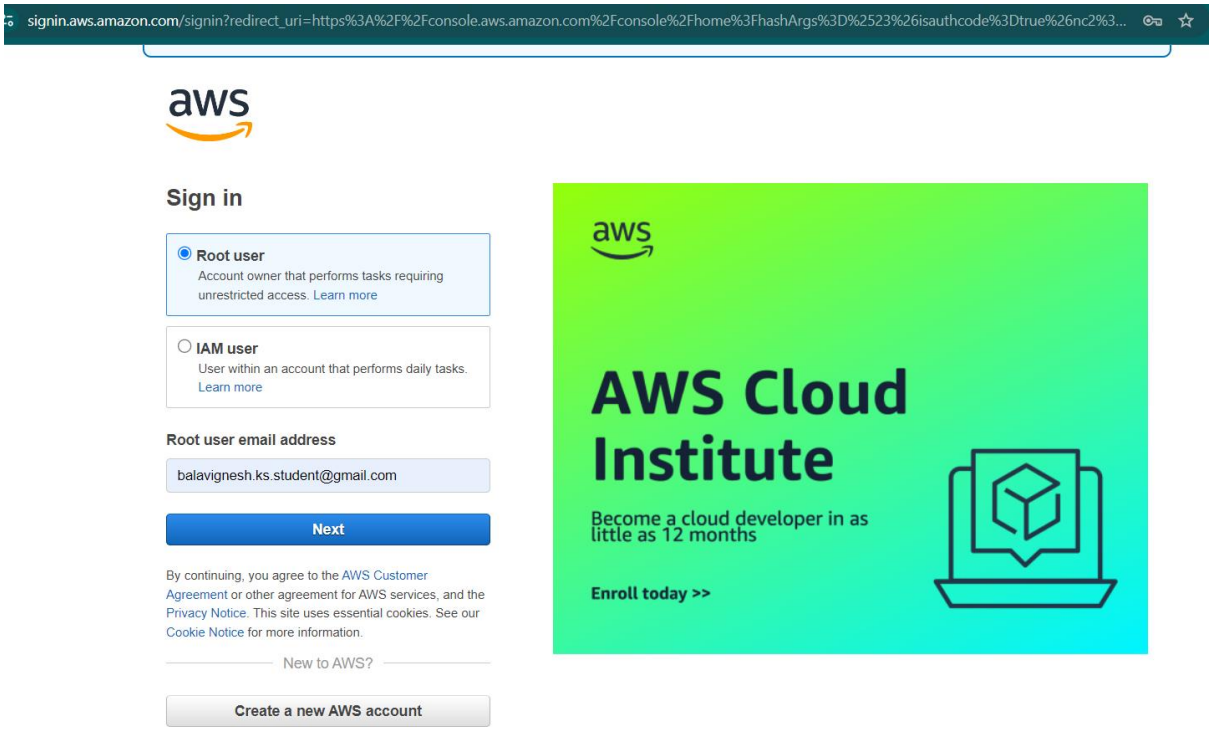


Step 4: You can use any browser of your choice to open and work with Jupyter Notebook.

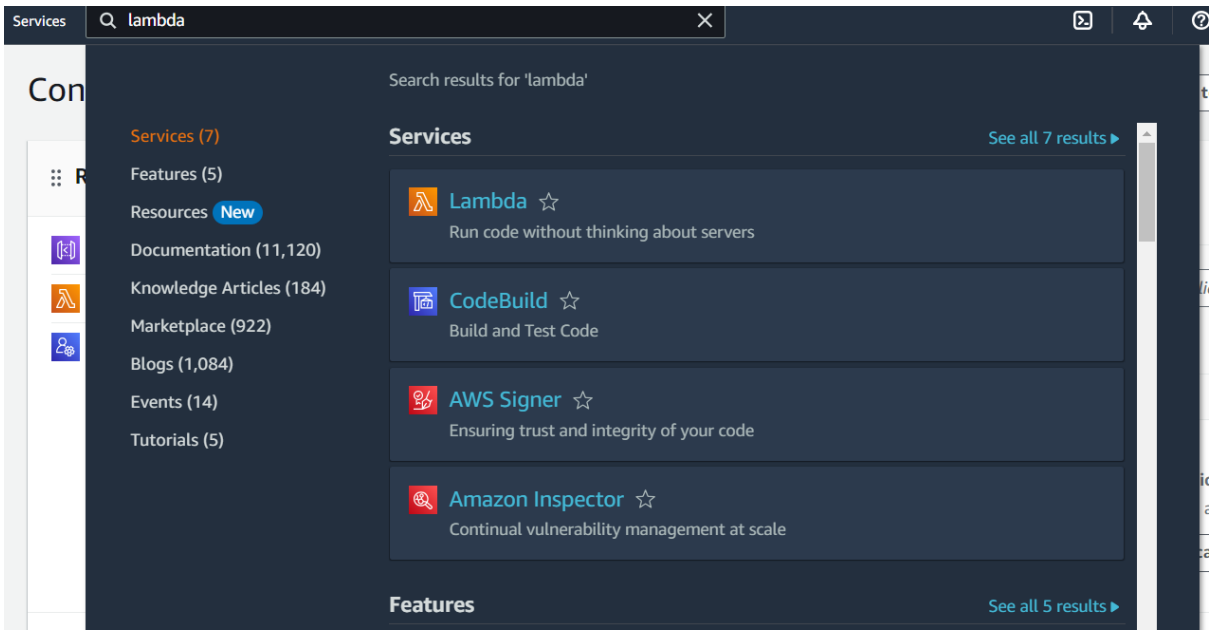
3.2 AWS Cloud

3.2.1 Lambda Function

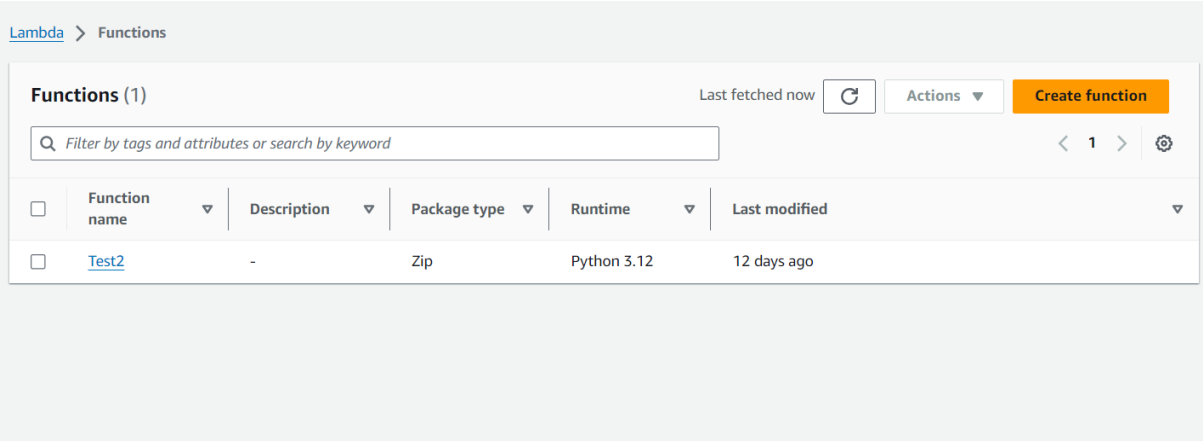
Step 1: Login in to AWS to access the service: Lambda and Rest API



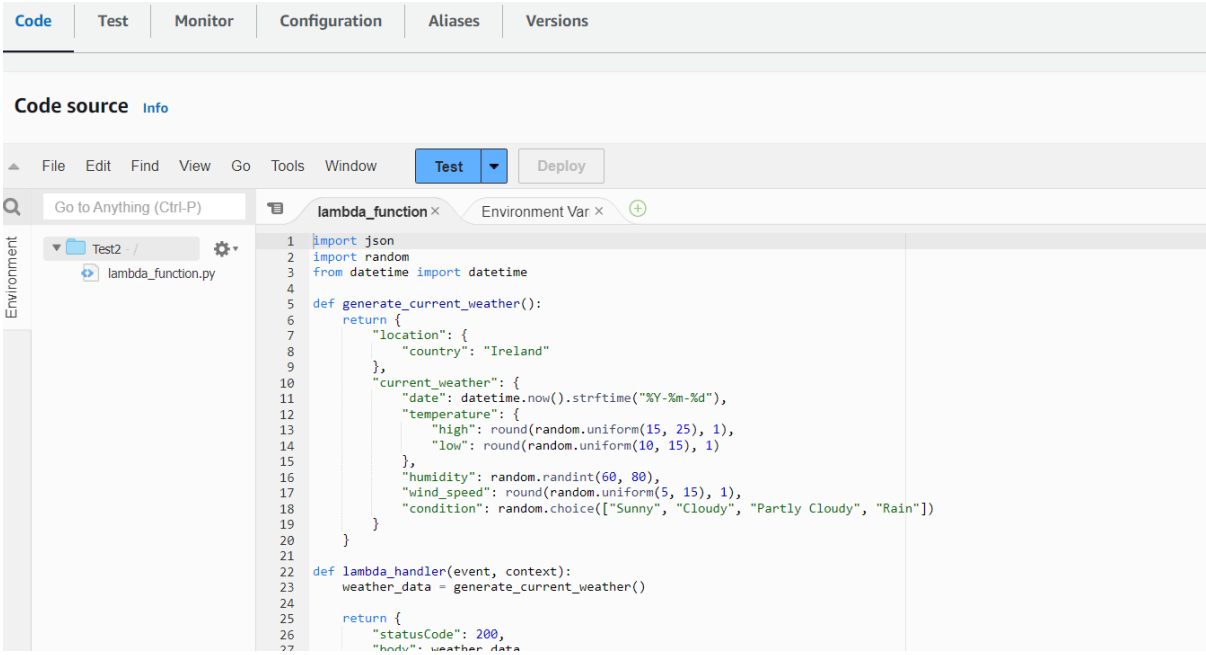
Step 2: Search for Lambda Service in AWS



Step 3: After launching Lambda Create Function

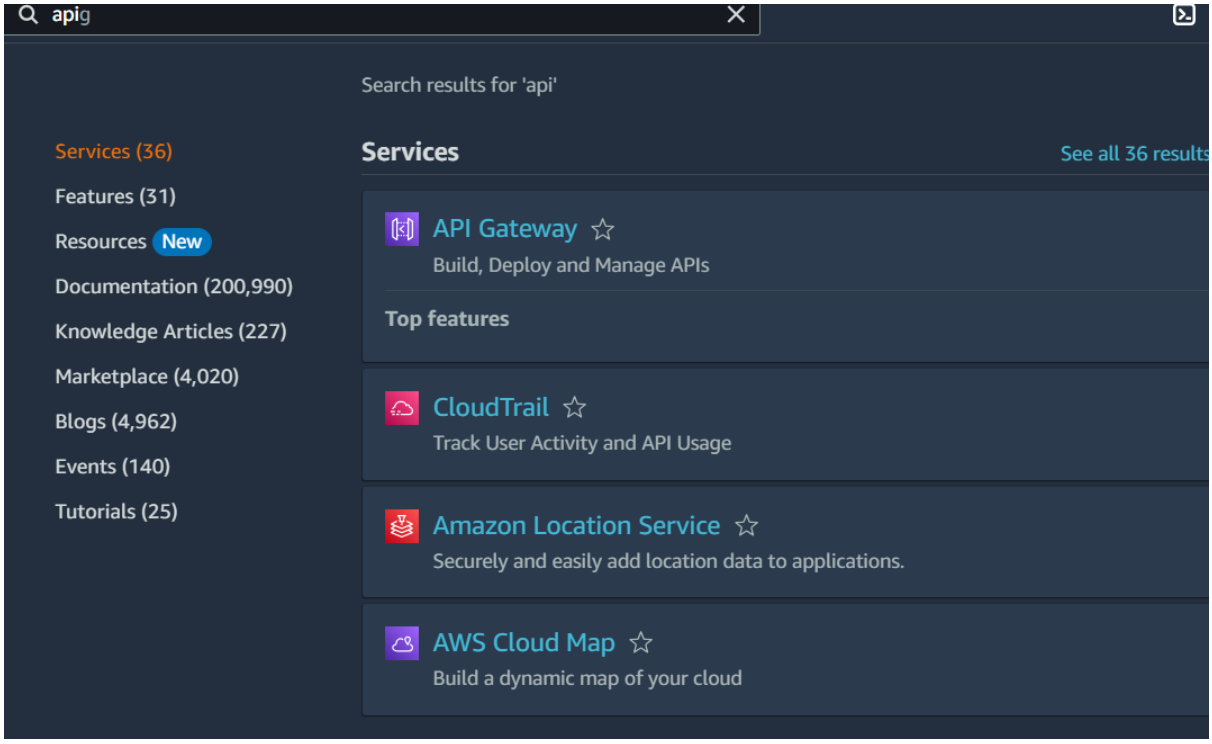


Step 4: Write the followinng Code in the lambda Coding Platform

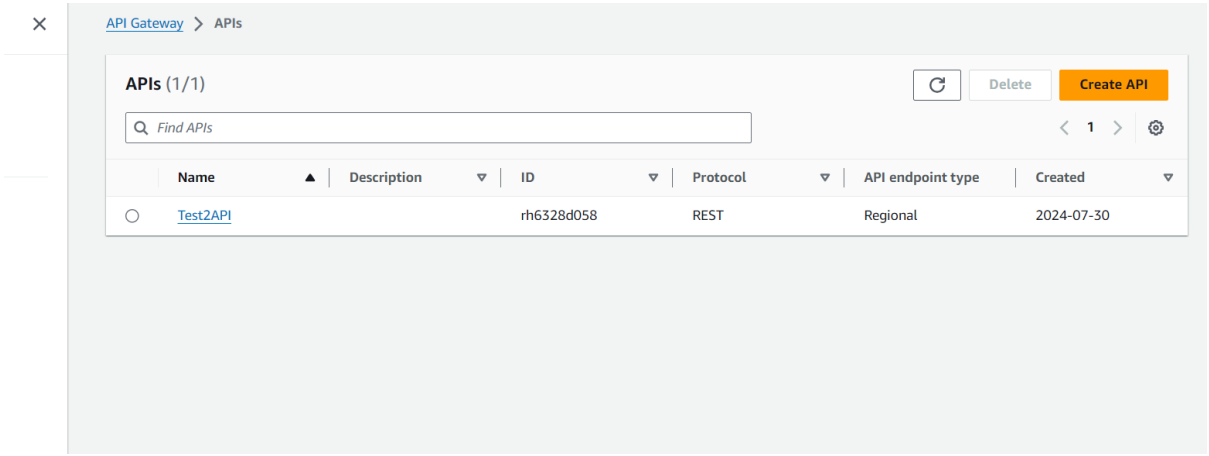


3.2.2 Creation of API Gateway

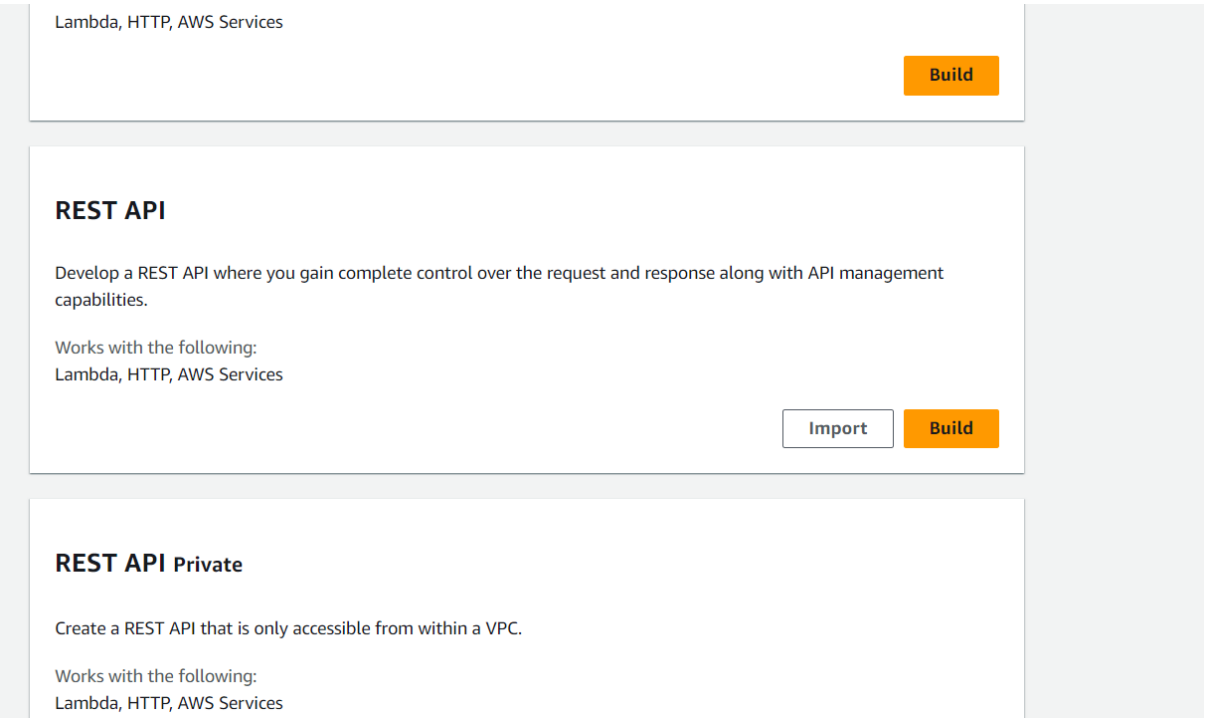
Step 1: Search for API Gateway



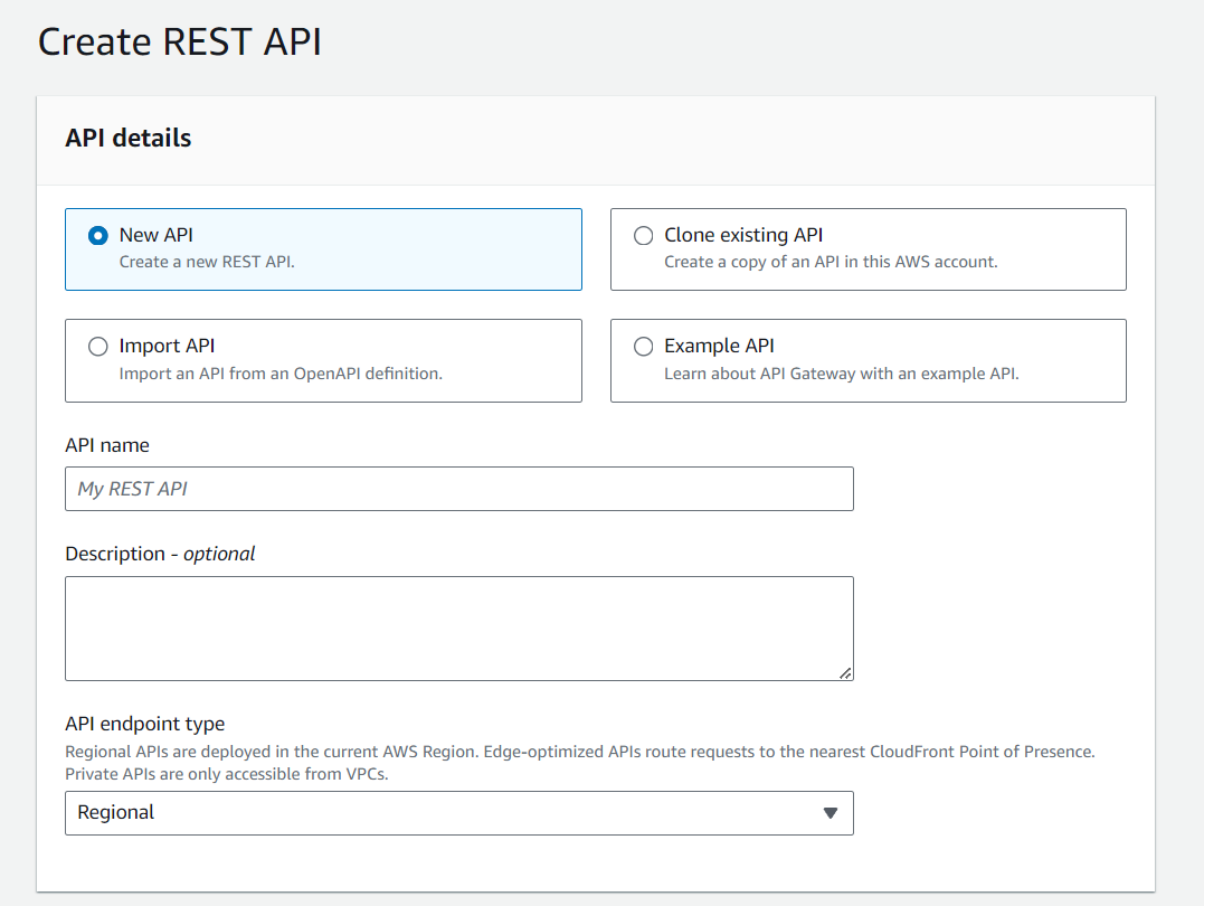
Step 2: Create API



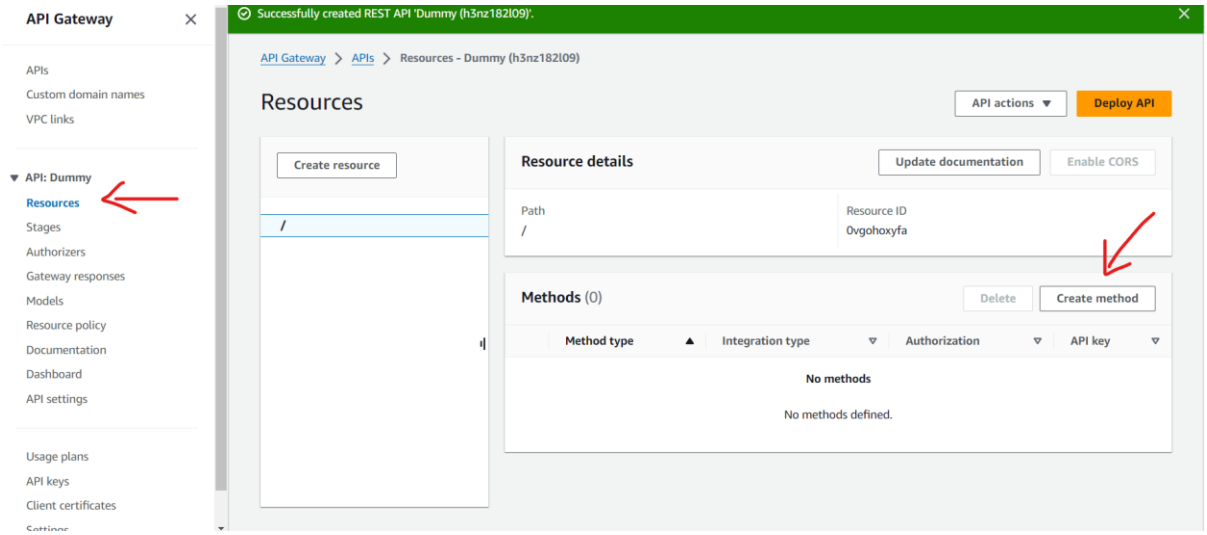
Step 3: Build Rest API

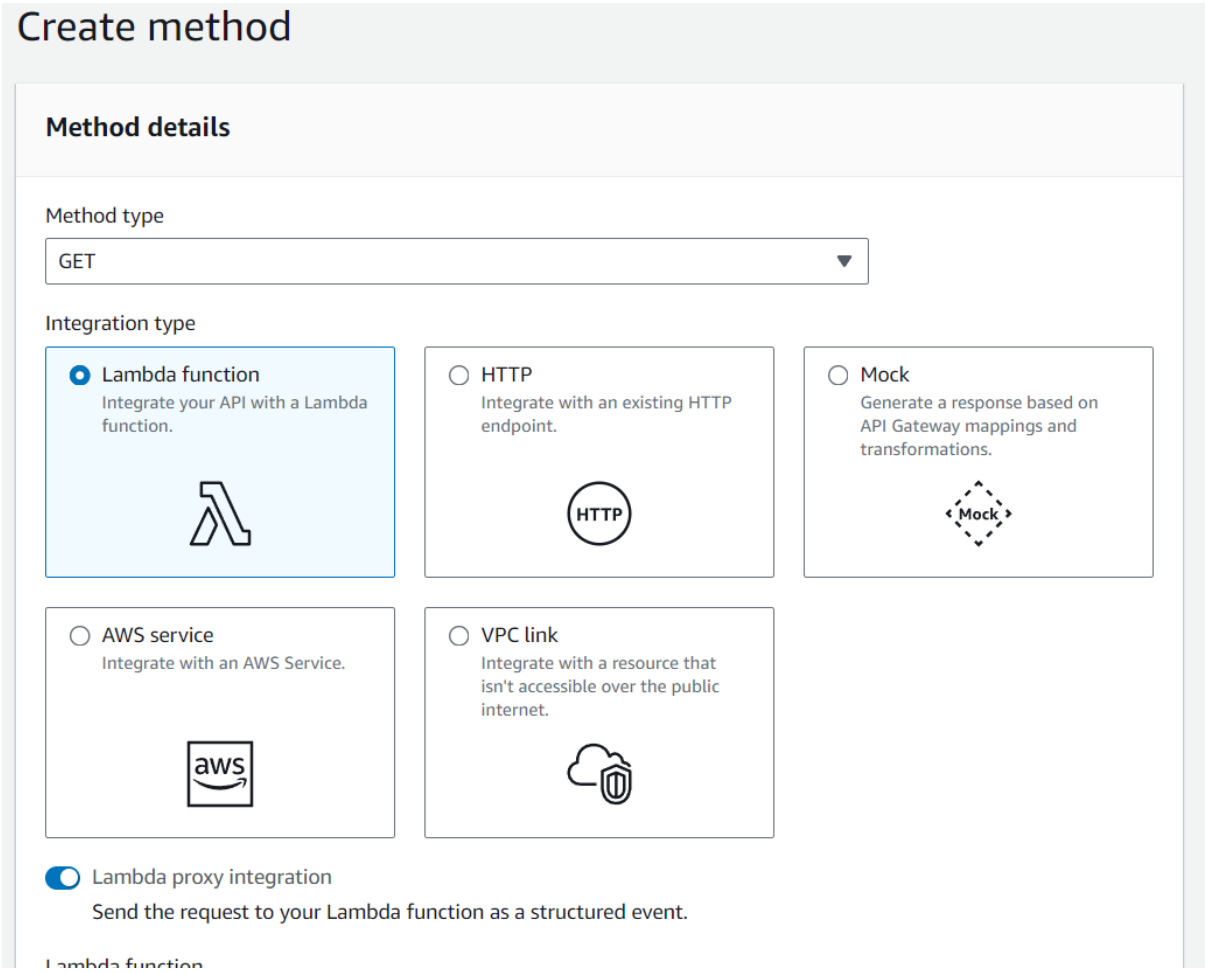


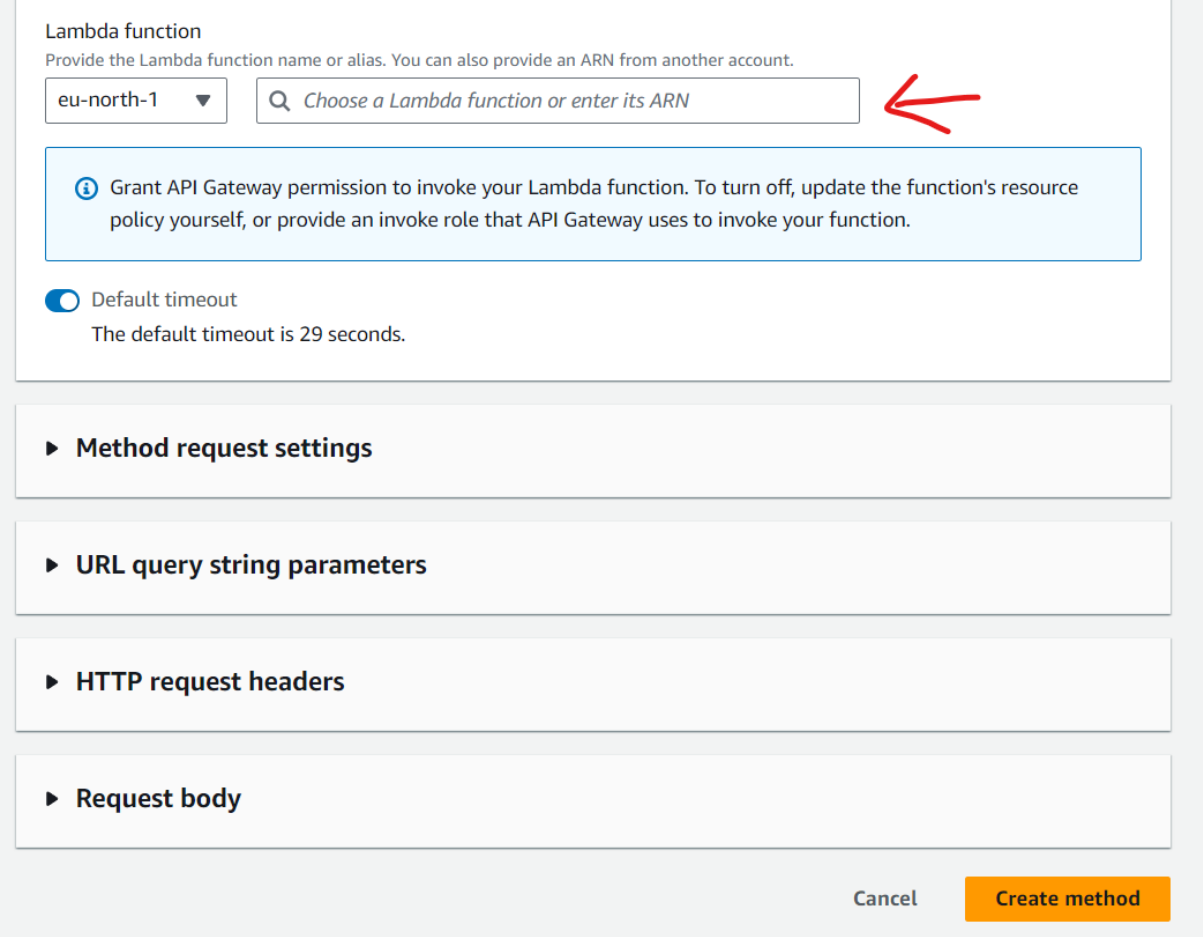
Step 4: Give a name for the API



Step 5: Create method in the new API







Select the created lambda function to integrate with the API

Do the same process (3.2.1 to 3.2.2) for two more different regions.

Now the initial setup is complete.

Run the load balancer and fault tolerance function in the Jupyter notebook mentioned below. Note the api urls need to be changed to the urls of your API.

```python
import requests
from prettytable import PrettyTable
import time
import itertools
from datetime import datetime
import pandas as pd

# API endpoints
apis = {
    "Stockholm": "https://q1xzknxdg0.execute-api.eu-north-1.amazonaws.com/TESTSTAGE1/",
    "Oregon": "https://rh6328d058.execute-api.us-west-2.amazonaws.com/TestStage2",
    "Ohio": "https://4ohx6p0mr9.execute-api.us-east-2.amazonaws.com/TestStage3"
}

# Initialize request log
request_log = []
uptime = 0
downtime = 0

def get_weather(api_url, location):
    global uptime, downtime
    try:
        start_time = datetime.now()
        response = requests.get(api_url)
        response_time = (datetime.now() - start_time).total_seconds()

        if response.status_code == 200:
            uptime += response_time
            request_log.append({
                "Location": location,
```

```python
                "Status": "Success",
                "Response Time (s)": response_time,
                "Timestamp": start_time,
                "Data": response.json()
            })
            return response.json(), True, response_time
        else:
            downtime += response_time
            request_log.append({
                "Location": location,
                "Status": "Failure",
                "Response Time (s)": response_time,
                "Timestamp": start_time,
                "Data": None
            })
            return None, False, response_time
    except requests.exceptions.RequestException:
        response_time = (datetime.now() - start_time).total_seconds()
        downtime += response_time
        request_log.append({
            "Location": location,
            "Status": "Failure",
            "Response Time (s)": response_time,
            "Timestamp": start_time,
            "Data": None
        })
        return None, False, response_time

def print_weather_table(weather_data, response_time):
    table = PrettyTable()
    table.field_names = ["Location", "Country", "Date", "High Temp (°C)", "Low Temp (°C)", "Humidity (%)", "Wind Speed (km/h)", "Condition", "Response Time (s)"]
```

```
    for location, data in weather_data.items():
        table.add_row([
            location,
            data["location"]["country"],
            data["current_weather"]["date"],
            data["current_weather"]["temperature"]["high"],
            data["current_weather"]["temperature"]["low"],
            data["current_weather"]["humidity"],
            data["current_weather"]["wind_speed"],
            data["current_weather"]["condition"],
            response_time
        ])

    print(table)

def export_log_to_csv():
    df = pd.DataFrame(request_log)
    df.to_csv("api_request_log_1.csv", index=False)
    print("\nRequest log exported to api_request_log.csv")

def print_statistics():
    total_requests = len(request_log)
    average_response_time = sum([log['Response Time (s)'] for log in request_log]) / total_requests if total_requests else 0
    uptime_percentage = (uptime / (uptime + downtime)) * 100 if (uptime + downtime) > 0 else 0

    table = PrettyTable()
    table.field_names = ["Total Requests", "Average Response Time (s)", "Uptime (s)", "Downtime (s)", "Uptime Percentage (%)"]
    table.add_row([
        total_requests,
        f"{average_response_time:.2f}",
        f"{uptime:.2f}"
```

```
        f"{downtime:.2f}",
        f"{uptime_percentage:.2f}"
    ])

    print("\nStatistics:")
    print(table)

def main():
    api_cycle = itertools.cycle(apis.items())  # Cycle through the API endpoints
    #Load Balancing
    try:
        while True:
            location, api_url = next(api_cycle)
            data, success, response_time = get_weather(api_url, location)
            # Fault Tolerance
            if not success:
                # Try the next API if the current one fails
                location, api_url = next(api_cycle)
                data, success, response_time = get_weather(api_url, location)

            if success:
                weather_data = {location: data['body']}
                print_weather_table(weather_data, response_time)
            else:
                print("All APIs failed to fetch weather data.")

            # Wait for a set period before calling the APIs again
            time.sleep(5)  # 5 seconds

    except KeyboardInterrupt:
        print_statistics()
        export_log_to_csv()
```

4 References

https://aws.amazon.com/
https://www.anaconda.com/download