

CS3204 Lab 2 Report - Cloud instance

Task 1

The platform chosen to host the web service and the database has been Amazon Web Services (AWS), in particular, Elastic Beanstalk. The name of the environment is **Cs3204cloud-env**, which runs a Python Flask web application. The URL for this web service is: <http://cs3204cloud-env.eba-wfbammam.eu-west-1.elasticbeanstalk.com/>

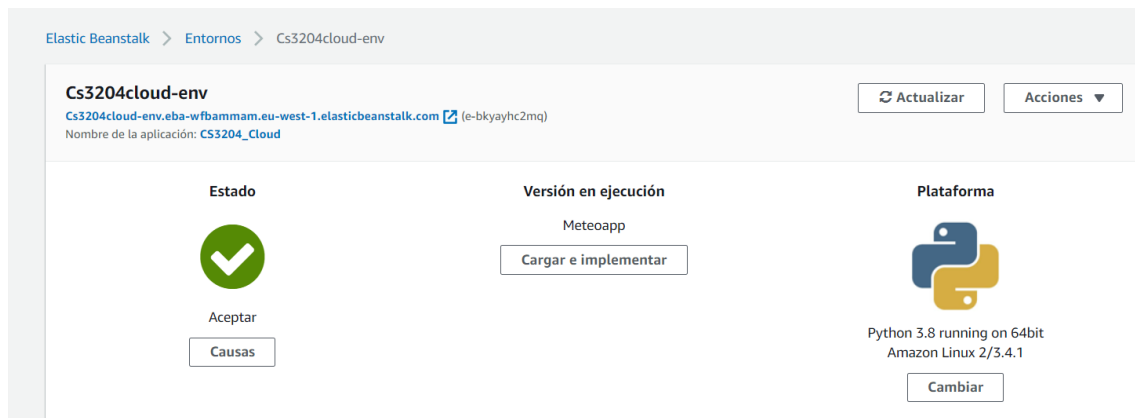


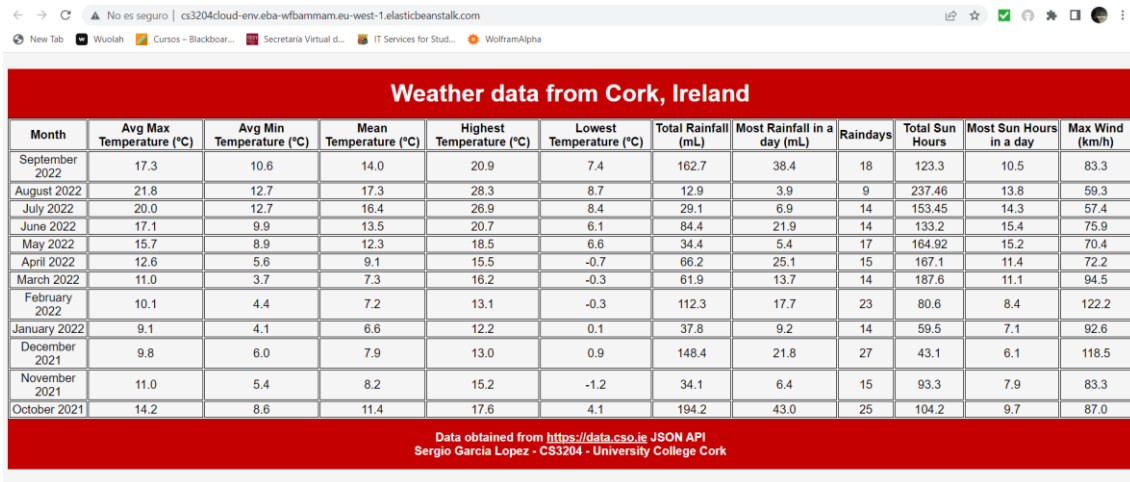
Figure 1: Elastic Beanstalk environment

After deploying the web application, we can see the most recent events in the logs section. The application deployment has been successful, and it is running correctly.

Eventos recientes			Mostrar todo
			< 1 >
Hora	Tipo	Detalles	
14-11-2022 20:18:36 UTC+0100	INFO	Environment health has transitioned from Ok to Info. Application update in progress on 1 instance. 0 out of 1 instance completed (running for 5 seconds).	
14-11-2022 20:17:56 UTC+0100	INFO	Environment update completed successfully.	
14-11-2022 20:17:56 UTC+0100	INFO	New application version was deployed to running EC2 instances.	
14-11-2022 20:17:50 UTC+0100	INFO	Instance deployment completed successfully.	
14-11-2022 20:17:47 UTC+0100	INFO	Instance deployment successfully generated a 'Profile'.	

Figure 2: Recent events

If we browse the website (<http://cs3204cloud-env.eba-wfbammam.eu-west-1.elasticbeanstalk.com/>) we can see the index page:



Month	Avg Max Temperature (°C)	Avg Min Temperature (°C)	Mean Temperature (°C)	Highest Temperature (°C)	Lowest Temperature (°C)	Total Rainfall (mL)	Most Rainfall in a day (mL)	Raindays	Total Sun Hours	Most Sun Hours in a day	Max Wind (km/h)
September 2022	17.3	10.6	14.0	20.9	7.4	162.7	38.4	18	123.3	10.5	83.3
August 2022	21.8	12.7	17.3	28.3	8.7	12.9	3.9	9	237.46	13.8	59.3
July 2022	20.0	12.7	16.4	26.9	8.4	29.1	6.9	14	153.45	14.3	57.4
June 2022	17.1	9.9	13.5	20.7	6.1	84.4	21.9	14	133.2	15.4	75.9
May 2022	15.7	8.9	12.3	18.5	6.6	34.4	5.4	17	164.92	15.2	70.4
April 2022	12.6	5.6	9.1	15.5	-0.7	66.2	25.1	15	167.1	11.4	72.2
March 2022	11.0	3.7	7.3	16.2	-0.3	61.9	13.7	14	187.6	11.1	94.5
February 2022	10.1	4.4	7.2	13.1	-0.3	112.3	17.7	23	80.6	8.4	122.2
January 2022	9.1	4.1	6.6	12.2	0.1	37.8	9.2	14	59.5	7.1	92.6
December 2021	9.8	6.0	7.9	13.0	0.9	148.4	21.8	27	43.1	6.1	118.5
November 2021	11.0	5.4	8.2	15.2	-1.2	34.1	6.4	15	93.3	7.9	83.3
October 2021	14.2	8.6	11.4	17.6	4.1	194.2	43.0	25	104.2	9.7	87.0

Data obtained from <https://data.cso.ie> JSON API
Sergio Garcia Lopez - CS3204 - University College Cork

Figure 3: Website

The web application consists in a service that displays weather data obtained from sensors in the Cork airport. All the data is obtained in JSON format from the Central Statistics Office API (<https://data.cso.ie/>). We can get the URLs to obtain the data from the API as indicated:

We first browse the website and search what kind of data do we want, in this case, we are interested in climate data.

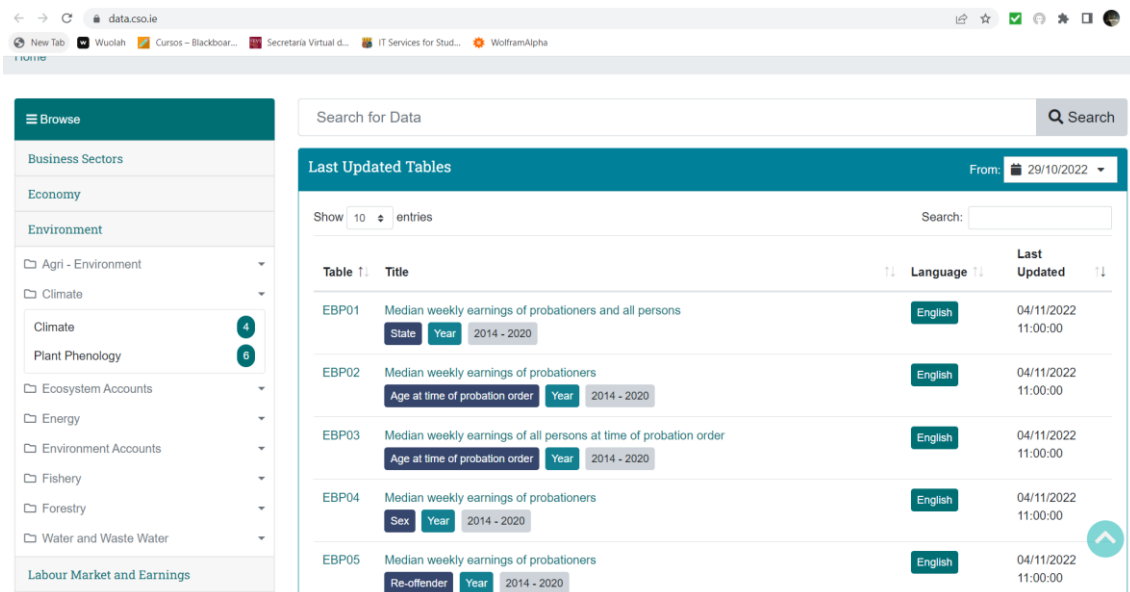


Table	Title	Language	Last Updated
EBP01	Median weekly earnings of probationers and all persons State Year 2014 - 2020	English	04/11/2022 11:00:00
EBP02	Median weekly earnings of probationers Age at time of probation order Year 2014 - 2020	English	04/11/2022 11:00:00
EBP03	Median weekly earnings of all persons at time of probation order Age at time of probation order Year 2014 - 2020	English	04/11/2022 11:00:00
EBP04	Median weekly earnings of probationers Sex Year 2014 - 2020	English	04/11/2022 11:00:00
EBP05	Median weekly earnings of probationers Re-offender Year 2014 - 2020	English	04/11/2022 11:00:00

Figure 4: Central Statistics Office API

We select Temperature:

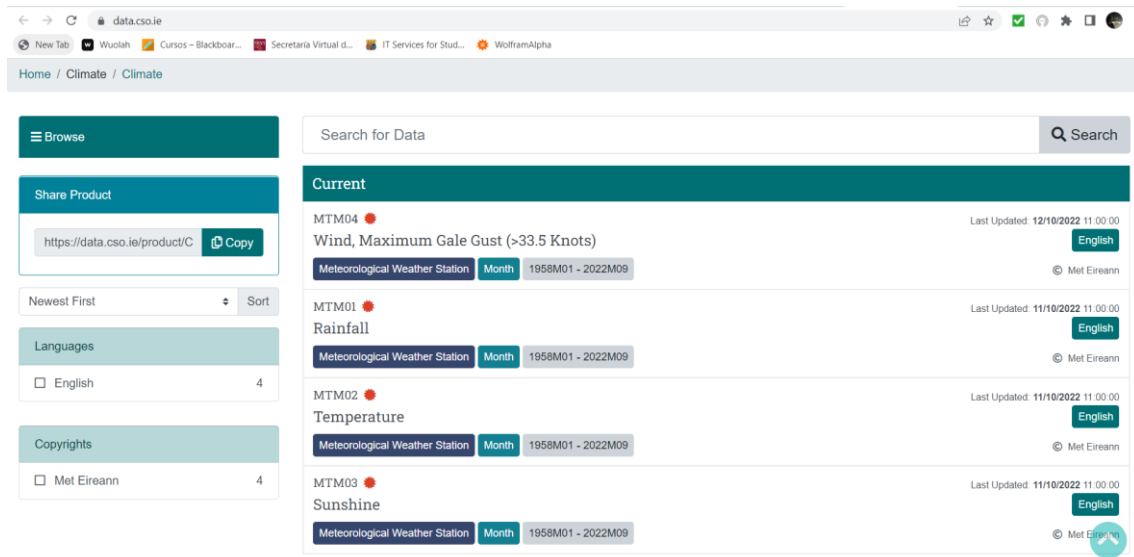


Figure 5: Central Statistics Office API

For example, let's say that we want to get all temperature data from Cork airport obtained in September 2022:

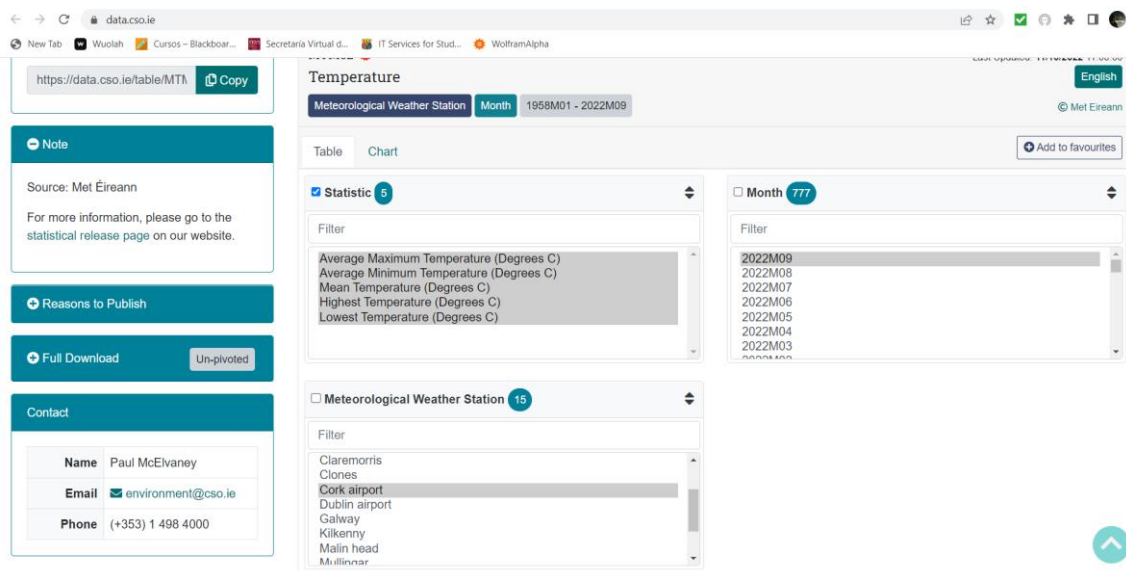


Figure 6: Central Statistics Office API

We obtain the following results:

Codes ☐ False Data Cells Selected: 5 / 580275

[Reset](#) [Pivot - None](#) [Download](#) [View](#)

Results [Save Query](#)

Show 100 entries Search:

Statistic	Month	Meteorological Weather Station	Unit	Value
Average Maximum Temperature	2022M09	Cork airport	Degrees C	17,3
Average Minimum Temperature	2022M09	Cork airport	Degrees C	10,6
Mean Temperature	2022M09	Cork airport	Degrees C	14,0
Highest Temperature	2022M09	Cork airport	Degrees C	20,9
Lowest Temperature	2022M09	Cork airport	Degrees C	7,4

Showing 1 to 5 of 5 entries [Previous](#) [Next](#)

Figure 7: Central Statistics Office API

We can also get the links to obtain this data with our web application using HTTP GET requests:

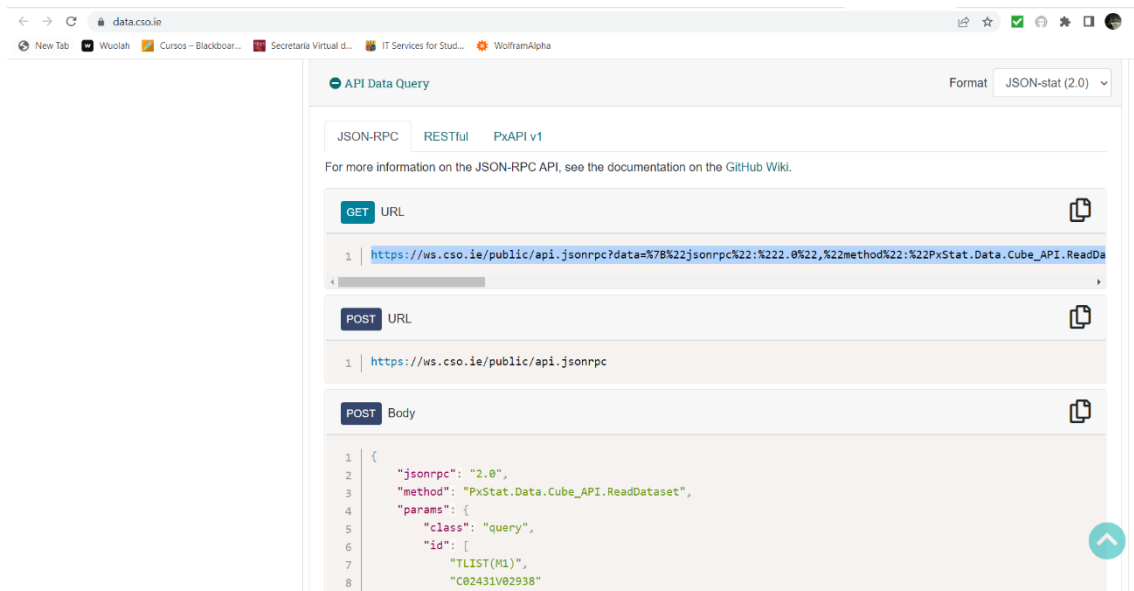


Figure 8: Central Statistics Office API

If we browse that link, we obtain the same data in JSON format. The field that we are interested in is `result.value`:

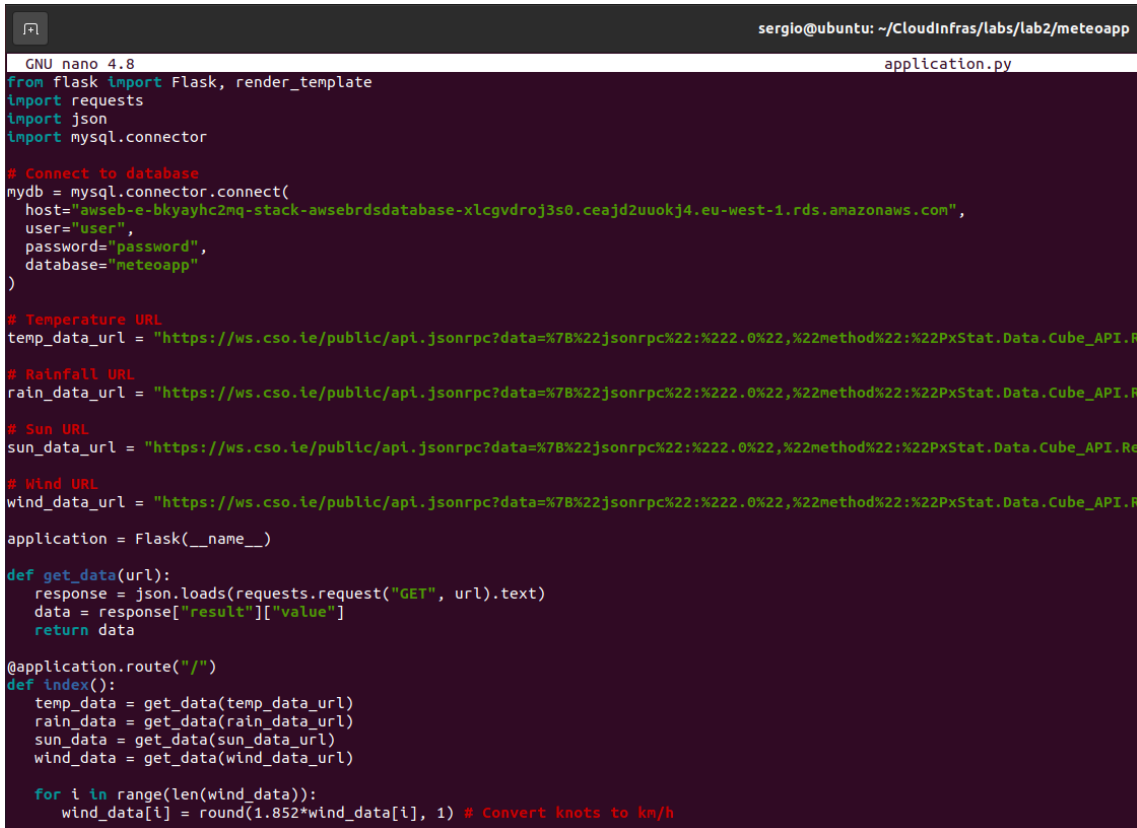
The screenshot shows a web browser interface with the following elements:

- Address bar: `https://ws.cso.ie/public/api.jsonrpc?data={"jsonrpc":"2.0","method":...`
- Navigation tabs: JSON (selected), Datos sin procesar, Cabeceras
- Actions: Guardar, Copiar, Contraer todo, Expandir todo, Filtrar JSON
- JSON Content:

```
jsonrpc: "2.0"
result: {
  class: "dataset"
  dimension: {...}
  extension: {...}
  href: "https://ws.cso.ie/public...Dataset/MTM02/PX/2013/en"
  id: [...]
  label: "Temperature"
  link: {...}
  note: [...]
  role: {...}
  size: [...]
  updated: "2022-10-11T11:00:00Z"
  value: [
    0: 17.3
    1: 10.6
    2: 14
    3: 20.9
    4: 7.4
    version: "2.0"
    id: null
  ]
}
```

Figure 9: Central Statistics Office API

The web application obtains all the data that it needs from the JSON API making use of prepared URLs, processes it to display it in a table and stores it in a database:



```
sergio@ubuntu: ~/CloudInfras/labs/lab2/meteoapp
GNU nano 4.8 application.py
from flask import Flask, render_template
import requests
import json
import mysql.connector

# Connect to database
mydb = mysql.connector.connect(
    host="awseb-e-bkyayhc2mq-stack-awsebrdsdatabase-xlcgvdroj3s0.ceajd2uuokj4.eu-west-1.rds.amazonaws.com",
    user="user",
    password="password",
    database="meteoapp"
)

# Temperature URL
temp_data_url = "https://ws.cso.ie/public/api.jsonrpc?data=%7B%22jsonrpc%22:%222.0%22,%22method%22:%22PxStat.Data.Cube_API.R"

# RainFall URL
rain_data_url = "https://ws.cso.ie/public/api.jsonrpc?data=%7B%22jsonrpc%22:%222.0%22,%22method%22:%22PxStat.Data.Cube_API.R"

# Sun URL
sun_data_url = "https://ws.cso.ie/public/api.jsonrpc?data=%7B%22jsonrpc%22:%222.0%22,%22method%22:%22PxStat.Data.Cube_API.R"

# Wind URL
wind_data_url = "https://ws.cso.ie/public/api.jsonrpc?data=%7B%22jsonrpc%22:%222.0%22,%22method%22:%22PxStat.Data.Cube_API.R"

application = Flask(__name__)

def get_data(url):
    response = json.loads(requests.request("GET", url).text)
    data = response["result"]["value"]
    return data

@app.route("/")
def index():
    temp_data = get_data(temp_data_url)
    rain_data = get_data(rain_data_url)
    sun_data = get_data(sun_data_url)
    wind_data = get_data(wind_data_url)

    for i in range(len(wind_data)):
        wind_data[i] = round(1.852*wind_data[i], 1) # Convert knots to km/h
```

Figure 10: application.py

The web design consists of a HTML5 file (**index.html**) with embedded CSS:



```
GNU nano 4.8
<!DOCTYPE html>
<html>
<head>
  <title>Meteo App</title>
  <meta name="viewport" content="width=device-width, initial-scale=1"/>
</head>
<style>
  body {
    background-color: #f5f5f5;
  }
  h1 {
    color: white;
    background-color: #c40000;
    padding-top: 1%;
    padding-bottom: 1%;
    text-align: center;
    font-family: Helvetica;
    margin-bottom: 0px;
  }
  p {
    color: white;
    text-align: center;
    font-family: Helvetica;
    margin-top: 0px;
    margin-bottom: 0px;
  }
  table {
    width:100%;
    border:1px solid black;
    margin-top: 0px;
    margin-bottom: 0px;
  }
  th, td {
    border:1px solid black;
    font-family: Helvetica;
    text-align: center;
  }
  footer {
    background-color: #c40000;
    padding-top: 1%;
    padding-bottom: 1%;
    font-family: Helvetica;
    font-style: bold;
    margin-top: 0px;
  }
}
```

Figure 11: index.html

The dynamically obtained data is rendered by Flask using templates:

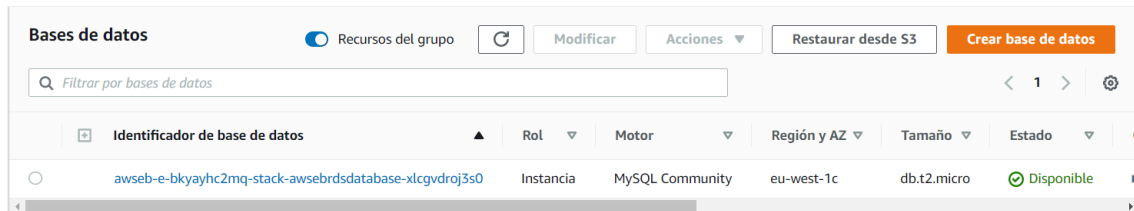


```
GNU nano 4.8
color: white;
}
</style>
<body>
  <h1>Weather data from Cork, Ireland</h1>
  <p><table style="width:100%">
    <tr>
      <th>Month</th>
      <th>Avg Max Temperature (°C)</th>
      <th>Avg Min Temperature (°C)</th>
      <th>Mean Temperature (°C)</th>
      <th>Highest Temperature (°C)</th>
      <th>Lowest Temperature (°C)</th>
      <th>Total Rainfall (mL)</th>
      <th>Most Rainfall in a day (mL)</th>
      <th>Raindays</th>
      <th>Total Sun Hours</th>
      <th>Most Sun Hours in a day</th>
      <th>Max Wind (km/h)</th>
    </tr>
    <tr>
      <td>September 2022</td>
      <td>{{avg_max_temp_sep}}</td>
      <td>{{avg_min_temp_sep}}</td>
      <td>{{mean_temp_sep}}</td>
      <td>{{high_temp_sep}}</td>
      <td>{{low_temp_sep}}</td>
      <td>{{total_rain_sep}}</td>
      <td>{{most_rain_sep}}</td>
      <td>{{raindays_sep}}</td>
      <td>{{total_sun_sep}}</td>
      <td>{{most_sun_sep}}</td>
      <td>{{max_wind_sep}}</td>
    </tr>
    <tr>
      <td>August 2022</td>
      <td>{{avg_max_temp_aug}}</td>
      <td>{{avg_min_temp_aug}}</td>
      <td>{{mean_temp_aug}}</td>
      <td>{{high_temp_aug}}</td>
      <td>{{low_temp_aug}}</td>
      <td>{{total_rain_aug}}</td>
      <td>{{most_rain_aug}}</td>
      <td>{{raindays_aug}}</td>
      <td>{{total_sun_aug}}</td>
```

Figure 12: index.html

Task 2

By making use of RDS, we can create a relational database to store the data processed by the application. In this case, we are using MySQL:

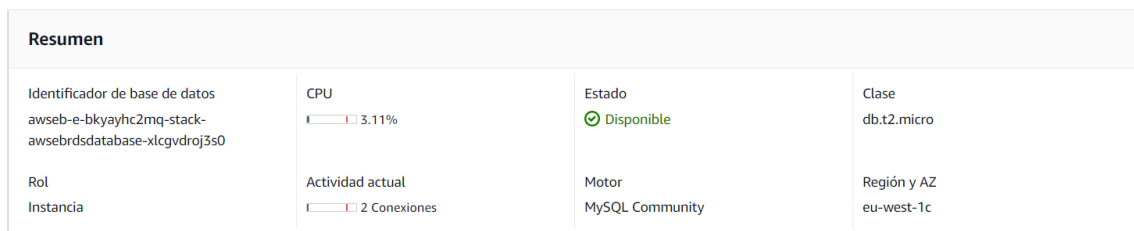


The screenshot shows the AWS RDS console 'Bases de datos' page. At the top, there are buttons for 'Recursos del grupo', 'Modificar', 'Acciones', 'Restaurar desde S3', and 'Crear base de datos'. Below these is a search bar labeled 'Filtrar por bases de datos'. A table lists the database instances with columns: 'Identificador de base de datos', 'Rol', 'Motor', 'Región y AZ', 'Tamaño', and 'Estado'. One instance is visible: 'awseb-e-bkyayhc2mq-stack-awsebrdsdatabase-xlcgvdroj3s0', which is an 'Instancia' of 'MySQL Community' in the 'eu-west-1c' region, with a size of 'db.t2.micro' and a status of 'Disponible'.

Identificador de base de datos	Rol	Motor	Región y AZ	Tamaño	Estado
awseb-e-bkyayhc2mq-stack-awsebrdsdatabase-xlcgvdroj3s0	Instancia	MySQL Community	eu-west-1c	db.t2.micro	Disponible

Figure 13: RDS database

If we click on the instance, we can see information related to performance, connections, state of the database, region, etc:



The screenshot shows the 'Resumen' (Summary) page for the database instance. It displays various metrics and configuration details in a grid layout.

Resumen			
Identificador de base de datos awseb-e-bkyayhc2mq-stack-awsebrdsdatabase-xlcgvdroj3s0	CPU 3.11%	Estado Disponible	Clase db.t2.micro
Rol Instancia	Actividad actual 2 Conexiones	Motor MySQL Community	Región y AZ eu-west-1c

Figure 14: RDS database

The database can be accessed in awseb-e-bkyayhc2mq-stack-awsebrdsdatabase-xlcgvdroj3s0.ceajd2uuokj4.eu-west-1.rds.amazonaws.com in port 3306:

Conectividad y seguridad		
Punto de enlace y puerto	Redes	Seguridad
Punto de enlace awseb-e-bkyayhc2mq-stack-awsebrdsdatabase-xlcgvdroj3s0.ceajd2uuokj4.eu-west-1.rds.amazonaws.com	Zona de disponibilidad eu-west-1c	Grupos de seguridad de la VPC rds-awseb-e-bkyayhc2mq-stack-awsebrdsdbsecuritygroup-frkctpm0rvbx-iopc (sg-00f644a2833e72b54) ✓ Activo
Puerto 3306	VPC vpc-061ec248d92f4fbf7	Accesible públicamente Sí
	Grupo de subredes default	Entidad de certificación rds-ca-2019
	Subredes subnet-09c4b0a480cadb447 subnet-00625ce0334cc50bb subnet-06ef042878026f07a	Fecha de la entidad de certificación August 22, 2024, 19:08 (UTC+02:00)
	Tipo de red IPv4	

Figure 15: RDS database

To allow connections from outside the cloud to the database we must add a security rule:

Grupos de seguridad (1/1) Información						
Filtrar grupos de seguridad						
search: sg-00f644a2833e72b54 X Quitar los filtros						
✓	Name	ID del grupo de segu...	Nombre del grupo ...	ID de la VPC	Descripción	Propietario
✓	-	sg-00f644a2833e72b54	rds-awseb-e-bkyayhc2...	vpc-061ec248d92f4fbf7	Security group for RDS...	546544363758

Figure 16: RDS security rules

We add a rule to the firewall to allow MySQL connections from anywhere:

Editar reglas de entrada [Información](#)

Las reglas de entrada controlan el tráfico entrante que puede llegar a la instancia.

ID de la regla del grupo de seguridad	Tipo Información	Protocolo Información	Intervalo de puertos Información	Origen Información	Descripción: opcional Información	
sgr-0da05a887bca8c98f	MySQL/Aurora	TCP	3306	Person...		Eliminar
sgr-0fdd0c72365793c3b	MySQL/Aurora	TCP	3306	Person...		Eliminar

0.0.0.0/0 X

sg-0c1a7164e33db72be X

Agregar regla

Cancelar Previsualizar los cambios Guardar reglas

Figure 17: RDS security rules

Now, we can remotely connect to the database using our credentials and create the tables that we want:

```
sergio@ubuntu: ~/CloudInfras/labs/lab2/meteoapp/ddbb
sergio@ubuntu:~/CloudInfras/labs/lab2/meteoapp/ddbb$ mysql -u cs3204 -p -h awseb-e-bkyayhc2mq-stack-awsebrdsdatabase-xlcvdroj3s0.ceajd2uukj4.eu-west-1.rds.amazonaws.com
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2570
Server version: 8.0.28 Source distribution

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

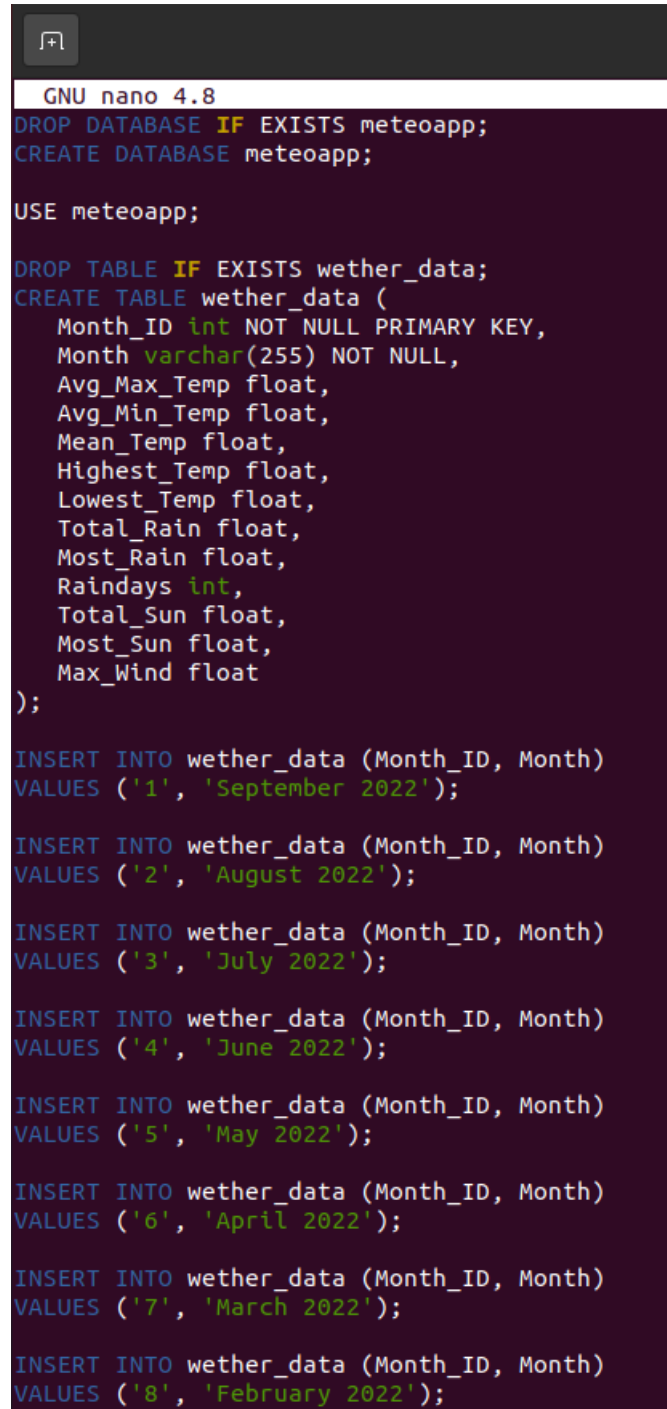
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Figure 18: Connection to database

We can use a SQL script to create the table faster:



```
GNU nano 4.8
DROP DATABASE IF EXISTS meteoapp;
CREATE DATABASE meteoapp;

USE meteoapp;

DROP TABLE IF EXISTS wether_data;
CREATE TABLE wether_data (
  Month_ID int NOT NULL PRIMARY KEY,
  Month varchar(255) NOT NULL,
  Avg_Max_Temp float,
  Avg_Min_Temp float,
  Mean_Temp float,
  Highest_Temp float,
  Lowest_Temp float,
  Total_Rain float,
  Most_Rain float,
  Raindays int,
  Total_Sun float,
  Most_Sun float,
  Max_Wind float
);

INSERT INTO wether_data (Month_ID, Month)
VALUES ('1', 'September 2022');

INSERT INTO wether_data (Month_ID, Month)
VALUES ('2', 'August 2022');

INSERT INTO wether_data (Month_ID, Month)
VALUES ('3', 'July 2022');

INSERT INTO wether_data (Month_ID, Month)
VALUES ('4', 'June 2022');

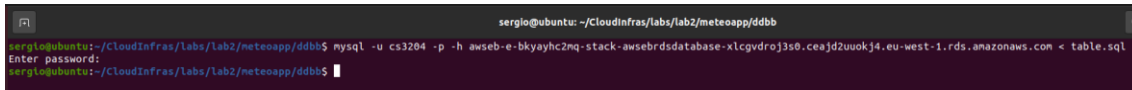
INSERT INTO wether_data (Month_ID, Month)
VALUES ('5', 'May 2022');

INSERT INTO wether_data (Month_ID, Month)
VALUES ('6', 'April 2022');

INSERT INTO wether_data (Month_ID, Month)
VALUES ('7', 'March 2022');

INSERT INTO wether_data (Month_ID, Month)
VALUES ('8', 'February 2022');
```

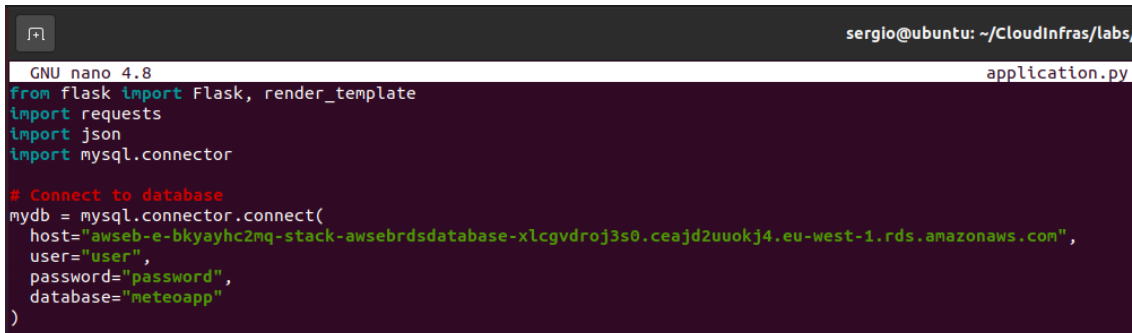
Figure 19: table.sql



```
sergio@ubuntu: ~/CloudInfras/labs/lab2/meteoapp/ddbb$ mysql -u cs3204 -p -h awseb-e-bkyayhc2mq-stack-awsebrdsdatabase-xlcvdroj3s0.ceajd2uokj4.eu-west-1.rds.amazonaws.com < table.sql
Enter password:
sergio@ubuntu:~/CloudInfras/labs/lab2/meteoapp/ddbb$
```

Figure 20: Create table with a SQL script

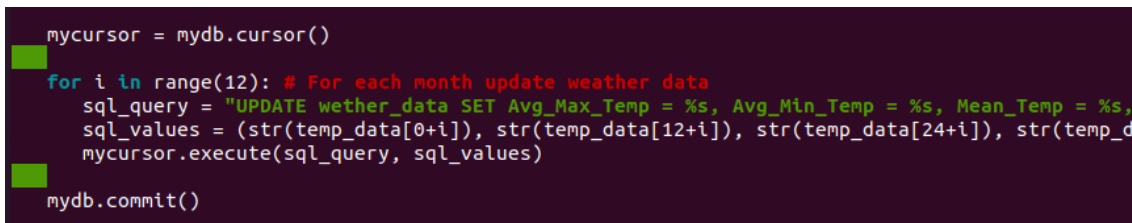
In the web application we connect to the database using our credentials and we update it with the data obtained from the JSON API:



```
sergio@ubuntu: ~/CloudInfras/labs/ application.py
GNU nano 4.8
from flask import Flask, render_template
import requests
import json
import mysql.connector

# Connect to database
mydb = mysql.connector.connect(
    host="awseb-e-bkyayhc2mq-stack-awsebrdsdatabase-xlcvdroj3s0.ceajd2uokj4.eu-west-1.rds.amazonaws.com",
    user="user",
    password="password",
    database="meteoapp"
)
```

Figure 21: Connection to database in Python



```
mycursor = mydb.cursor()

for i in range(12): # For each month update weather data
    sql_query = "UPDATE wether_data SET Avg_Max_Temp = %s, Avg_Min_Temp = %s, Mean_Temp = %s,"
    sql_values = (str(temp_data[0+i]), str(temp_data[12+i]), str(temp_data[24+i]), str(temp_data[36+i]))
    mycursor.execute(sql_query, sql_values)

mydb.commit()
```

Figure 22: Update database

Conclusions

I found this lab quite interesting and useful. I think I have improved the knowledge I already had in web development, and in fact I have learned to use a new framework like Flask. It has also allowed me to put into practice my knowledge of databases, especially in the use of MySQL. I think that services like Elastic Beanstalk make it much easier to deploy web applications, both frontend and backend.

However, during the development of this lab I encountered some problems at the beginning, when I tried to deploy a simple example website with Flask. To begin with, it was necessary to indicate in a requirements file the necessary libraries to be installed by the platform in the environment. On the other hand, it was necessary to call the Flask object that represents the web application as "application" so that the web application could run on the platform without

errors. Finally, the zip file that has to be uploaded to the platform must have a specific structure, and the files must be located outside of a main directory that groups them together.

I also found it a bit challenging to get data from the API because I had to understand the different data that each request returned and how to generate the right URLs to get it. On the other hand, the creation of the database and the table with the data produced by the web application seemed easy to me, partly because I already had experience with MySQL in Linux environments.

References

Link to application code in GitHub: https://github.com/SergiDelta/CS3204_Lab_2