**OS-C ClimateScore API Service Installation Guide**

1. **Introduction**

The purpose of this document is to guide a Cloud Engineer in installing the OS-C ClimateScore API Service (“ClimateScore Service”) into the AWS Cloud Platform. The guide assumes the installer has some familiarity with the AWS Console, specifically AWS RDS, AWS ECS, AWS EC2 (SG and ALB), PgAdmin, Python and SQL.

The ClimateScore Service has three major components:

* An AWS RDS PostgreSQL server that hosts all the Jupiter contributed Climate Data.
* An AWS EC2 Fargate Cluster that instantiates containerized a Flask Application that provides the Business Logic for the ClimateScore Service. The Fargate Cluster is in turn connected to an Application Load Balancer.
* The third and final component is a suite of Front-End Applications, consisting of a Postman Collection, a Jupyter Notebook, and an Open API definition file, to serve as models for Application Developers who want to use the ClimateScore Service.

**Manifest**

All the artifacts necessary to install the ClimateScore Service is in git repo with the following structure:

Table

Description automatically generated

OS-C-DATA contains all the Jupiter contributed data in CSV format. The data is delivered as seven CSV files, each containing data for one peril (combined flood, drought, fire, hail, heat, precipitation, and wind).

OS-C-RDS contains the Python scripts necessary to create tables for each of the Perils as well as corresponding indexes.

Both OS-C-DATA and OS-C-RDS artifacts are used in the “**Creating and Loading the Database**” section of this guide.

OS-C-API-SERVER contains the Python Source Code of the Flask Application Server that provides the business logic of the ClimateScore Service.

The OS-C-API-SERVER artifacts are used in the “**Installing, Configuring and Testing the API Application Server**” section of this guide.

OS-C-FE-APPS contains examples and documentation in Postman collections, Jupyter Notebooks, and Open API/Swagger format that is intended to help Developers and Data Scientists develop applications that utilize the ClimateScore Service.

Installation and use of these Applications are documented in “**Installing, Testing, and Making Use of example Front End Applications**”

**Important:**

In this guide, we will assume that the ClimateScore service and all associated components will be hosted in AWS region “us-east-02”. It assumes the installer has at least System Administrator rights in the account. It is also assumed that the installation will be done via AWS Console.

1. **Creating and Loading the Database**

Step 1: Create the PostgreSQL database in the AWS RDS Service.

1. Select US-EAST-2 as the region.
2. Setting up the RDS. In AWS, select the RDS Service, and create a PostgreSQL t2.micro.instance. Set the credentials as you will.

DB Instance identifier: os-c-db

Master username: postgres

Master password: os-c-db-NoPassword123!

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Step 2: Tag the newly created database, and temporarily allow remote access to the database.

1. Tag the DB with “project”:”os-c”.
2. Temporarily allow your Desktop to access the database, by setting Public Accessibility to Yes, and the Security Group to allow Access on all Ports from your PC.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Step 3: Administer the database from your desktop via PgAdmin.

1. Connect to the AWS RDS instance from your PgAdmin using the credentials when you created your AWS RDS instance. Failure modes here include wrong or missing credential, database not accessible from your IP or any external IP.
2. Create a database called os-c-db
3. Create the tables and indices. Go to OS-C Distribution/OS-C-RDS/ and modify the Python scripts so they point to the right RDS.

import psycopg2

connection = psycopg2.connect(

host="os-c-db.cfxe5jyx7l0h.us-east-2.rds.amazonaws.com", port=5432, database="os-c-db", user="postgres", password="os-c-db-NoPassword123!")

cursor = connection.cursor()

1. Using an environment with Python 3.7 and psychopg2 (e.g. psycopg2-binary), Run the Scripts (all 7 of them)

For example:

$ python OSCreateDB-CombinedFlood.py

The correct tables and indices will be created

Step 4: Use PgAdmin to import the CSVs from OS-C-DAT to the proper tables.

Graphical user interface, text, application, Teams

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Step 5: Use etlcheck.sql to verify data was imported.

select \* from osdbcombinedflood limit 10;

select \* from osdbcombinedflood where key='64.00116.00';

select count(\*) from osdbcombinedflood;

Repeat for all tables.

1. **Installing, Configuring and Testing the API Application Server**

Step 1: Test API-SERVER locally

1. Go to OS-C-API-SERVER. Activate an environment that contains Python 3.7.
2. Run

make install

1. In App.py change the connection string to point to the right DB

connection = psycopg2.connect(host="os-c-db.cfxe5jyx7l0h.us-east-2.rds.amazonaws.com", port=5432, database="os-c-db", user="postgres", password="os-c-db-NoPassword123!")

1. Run

make test

Graphical user interface, text, application, email

Description automatically generated

Warning errors are acceptable.

1. Set environment variables

export db\_host=os-c-db.cfxe5jyx7l0h.us-east-2.rds.amazonaws.com

export db\_port=5432

export db\_database=os-c-db

export db\_user=postgres

export db\_password=os-c-db-NoPassword123!

export api\_key=1234567890

1. Run

python app.py

to run os-api-server locally.py

Graphical user interface, text, application, email

Description automatically generated

1. In a browser, type

localhost:8080/healthz

The service should return OK.

Step 2: Build Docker Image and Run Container Locally

1. In the OS-C-APP-SERVER directory, execute the Docker build and run commands:

docker system prune

docker build -t os-c-api-server .

docker run -p 8080:8080 -e api\_key=1234567890 -e db\_host=os-c-db.cfxe5jyx7l0h.us-east-2.rds.amazonaws.com -e db\_port=5432 -e db\_database=os-c-db -e db\_user=postgres -e db\_password=os-c-db-NoPassword123! os-c-api-server

1. In a browser, type

localhost:8080/healthz

The service should return OK.

Step 3: Deploy and Run Image in AWS ECS Fargate with ALB

1. Create an Repository in US-EAST-02 ECR

Graphical user interface, application

Description automatically generated

1. Push our locally built image into ECR

aws ecr get-login-password --region us-east-2 | docker login --username AWS --password-stdin 403963206024.dkr.ecr.us-east-2.amazonaws.com

docker build -t os-c-api-server .

docker tag os-c-api-server:latest 403963206024.dkr.ecr.us-east-2.amazonaws.com/os-c-api-server:latest

1. Create ECS Cluster. Make sure to expose port 8080 on the Container.

Graphical user interface, text, application, email

Description automatically generated

1. Create Task Definition

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

5. Create Application Load Balancer

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

6: Create ECS Service

Graphical user interface, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

1. Open ECS Security Group to ALB Security Group

Graphical user interface, text, application, email

Description automatically generated

1. In a browser, type the address of the ALB /healthz e.g.

http://os-c-api-server-alb-1601253115.us-east-2.elb.amazonaws.com/healthz

Should return OK.

1. “**Installing, Testing, and Making Use of example Front End Applications**”

The third and final component of the ClimateScore Service, which can be found in OS-C-FE\_APPS.

Step 1: Postman Collection and Environments

csg\_api\_osc (RDS).postman\_collection.json

is a collection of postman tests to exercise the ClimateScore Service. It also provides good examples of how to access the service, by clicking on code (upper right) of a particular test and seeing a code snippet showing how that test is called.

csg\_api\_osc ECS ALB.postman\_environment.json

is a postman environment pointing to the deployed ALB for the ECS Cluster.

csg\_api\_osc (localhost:8080).postman\_environment.json

is a postman environment pointing to the locally running version of the OS-C API Server. It is used for local testing.

Graphical user interface, text, application

Description automatically generated

Step 2: Jupyter Notebook

Another include application is

os-c rds – test suite.ipynb

This application shows how filters can be passed to the ClimateScore Service, how to send multiple Locations (up to 100) in an array, and how to benchmark performance of the ClimateScore Service.

#url = "http://os-c-api-server-alb-1601253115.us-east-2.elb.amazonaws.com//location"

#Num of Queries: 1000 Time Elapsed: 53.87759804725647 Queries Per Second: 18.56058986005449

#Num of Queries: 1000 Time Elapsed: 59.1816782951355 Queries Per Second: 16.89712135254191

#Num of Queries: 10000 Time Elapsed: 559.5767290592194 Queries Per Second: 17.870650226667507

Step 3: Open API/Swagger

The /location endpoint is also document in an Open API/Swagger compatible JSON file.

1. **Security Best Practices**

Step 1: Environment Variables

As part of Security Best Practices, critical parameters are not hard coded into the Application Source Code. Instead, they are injected at run-time through environment variables.

For local deployments, the following variables need to be set like so:

export db\_host=os-c-db.cfxe5jyx7l0h.us-east-2.rds.amazonaws.com

export db\_port=5432

export db\_database=os-c-db

export db\_user=postgres

export db\_password=os-c-db-NoPassword123!

export api\_key=1234567890

For containers, the variables are set in te JSON configuration of the Task Definition, like so:

"environment": [

{

"name": "api\_key",

"value": "1234567890"

},

{

"name": "db\_database",

"value": "os-c-db"

},

{

"name": "db\_host",

"value": "os-c-db.cfxe5jyx7l0h.us-east-2.rds.amazonaws.com"

},

{

"name": "db\_password",

"value": "os-c-db-NoPassword123!"

},

{

"name": "db\_port",

"value": "5432"

},

{

"name": "db\_user",

"value": "postgres"

}

],

Step 2: API Key Authentication

A simple API Key Authentication system is implemented. Each call to the ClimateScore Service must include an X-Api-Key key in its header with a value corresponding to the one one set in the API Server environment variable, like so:

import requests

url = "https://api-phase3-osc.jupiterintel.com//location"

payload="{\n \"locations\": [\n {\"latitude\": 37.793871, \"longitude\": -122.395556}\n ]\n}"

headers = {

'X-Api-Key': '1234567890',

'Content-Type': 'application/json'

}

response = requests.request("POST", url, headers=headers, data=payload)

Step 3: Miscellaneous

* The API Server listens on port 8080.
* The ALB listens on port 443
* The API Server needs to access the AWS RDS at port 5432