

# Cloud Computing: Servicios y Aplicaciones



## UNIVERSIDAD DE GRANADA

### PRÁCTICA 2 DESPLIEGUE DE UN SERVICIO CLOUD NATIVE

**Autor**

Juan Manuel Castillo Nievas



MÁSTER PROFESIONAL EN INGENIERÍA INFORMÁTICA 2020-2021

Granada, 7 de mayo de 2021

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Configuración del entorno</b>	<b>3</b>
<b>3. Resolución de las tareas</b>	<b>3</b>
3.1. Preparación del entorno . . . . .	4
3.2. Descarga de datos A y B . . . . .	4
3.3. Captura de datos . . . . .	5
3.4. Almacenamiento de datos . . . . .	5
3.5. Creación de los modelos de predicción . . . . .	5
3.5.1. Modelo de predicción ARIMA . . . . .	6
3.5.2. Modelo de predicción Prophet . . . . .	6
3.5.3. Problemas encontrados . . . . .	6
3.6. Clonación del repositorio . . . . .	7
3.7. Unit test . . . . .	7
3.8. Build . . . . .	7
3.8.1. Creación de la API . . . . .	8
3.8.2. Problemas encontrados . . . . .	8
3.9. Despliegue . . . . .	9
3.9.1. Problemas encontrados . . . . .	9
<b>4. Demostración</b>	<b>10</b>
4.1. Acceso a la API v1 . . . . .	11
4.2. Acceso a la API v2 . . . . .	13
<b>5. Bibliografía</b>	<b>15</b>

# 1. Introducción

El objetivo de esta práctica es llevar a cabo un despliegue completo de un servicio **Cloud Native**. Para esto, es necesario entender los aspectos claves para Cloud Native:

- **DevOps:** colaboración entre los desarrolladores de software y operaciones TI para la entrega constante, realizando la construcción, prueba y liberación de software.
- **CI/CD:** integración continua y entrega continua.
- **Microservicios:** un pequeño servicio que ejecuta su propio proceso y se comunica a través de API en algunos casos. Cada microservicio se despliega, actualiza, escala y reinicia de manera independiente a los demás.
- **Contenedores:** utilizados para el despliegue de los microservicios.

Para esta práctica se va a desarrollar un sistema de predicción de temperatura y humedad. Se debe desplegar un servicio que proporcione una API de tipo HTTP RESTful con las predicciones. Esta API consta de 2 versiones, utilizando en cada versión un modelo de predicción diferente, y consta de los siguientes endpoints:

- **Version 1**
  - `/servicio/v1/prediccion/24horas`: obtiene la predicción de las siguientes **24 horas** usando el modelo de predicción de la versión 1
  - `/servicio/v1/prediccion/48horas`: obtiene la predicción de las siguientes **48 horas** usando el modelo de predicción de la versión 1
  - `/servicio/v1/prediccion/72horas`: obtiene la predicción de las siguientes **72 horas** usando el modelo de predicción de la versión 1
- **Version 2**
  - `/servicio/v2/prediccion/24horas`: obtiene la predicción de las siguientes **24 horas** usando el modelo de predicción de la versión 2
  - `/servicio/v2/prediccion/48horas`: obtiene la predicción de las siguientes **48 horas** usando el modelo de predicción de la versión 2
  - `/servicio/v2/prediccion/72horas`: obtiene la predicción de las siguientes **72 horas** usando el modelo de predicción de la versión 2

El conjunto de salida debe ser de tipo JSON y debe tener la siguiente estructura:

```
[{'hour':13:05,'temp':32.20,'hum':85.90}, ...]
```

## 2. Configuración del entorno

Para el desarrollo de la práctica se han utilizado las siguientes herramientas:

- **Python 3** [1] como lenguaje de programación
- **Flask** [2] para el desarrollo de la API
- **Apache Airflow** [3] para el diseño del flujo de trabajo del servicio Cloud Native
- **MongoDB** [4] como base de datos para el almacenamiento de los datos
- **pymongo** [5] como biblioteca para el uso de MongoDB en Python
- **unittest** [6] para la batería de pruebas con TDD de la API
- **pandas** [7] como biblioteca para la manipulación de datos en Python
- **Statsmodels** [8] para el uso del modelo **ARIMA** [9] para la creación del modelo de predicción de la **versión 1** de la API
- **Prophet** [10] para la creación del modelo de predicción de la **versión 2** de la API
- **Pickle** [11] para el almacenamiento de los modelos de predicción creados
- **gunicorn** [12] para desplegar la API en un servidor HTTP

## 3. Resolución de las tareas

Todo el código desarrollado se encuentra en mi repositorio de **Github**:

<https://github.com/Jumacasni/MUII-CCSA/tree/main/practica2>

A modo de introducción, en la Figura 1 se muestra el diagrama de flujo de trabajo que se ha utilizado para el despliegue del servicio Cloud Native usando Apache Airflow.

Este flujo está disponible en el archivo del repositorio llamado **practica2\_dag.py**, que es el que se debe incluir en la carpeta **dags** en la instalación de **Airflow** para que pueda ejecutarse. El enlace del archivo en el repositorio es el siguiente:

[https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/practica2\\_dag.py](https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/practica2_dag.py)

Cada tarea realizada en el flujo de trabajo se va a explicar a continuación en las siguientes subsecciones, indicando en cada caso el fichero del repositorio que se ha utilizado para ello.

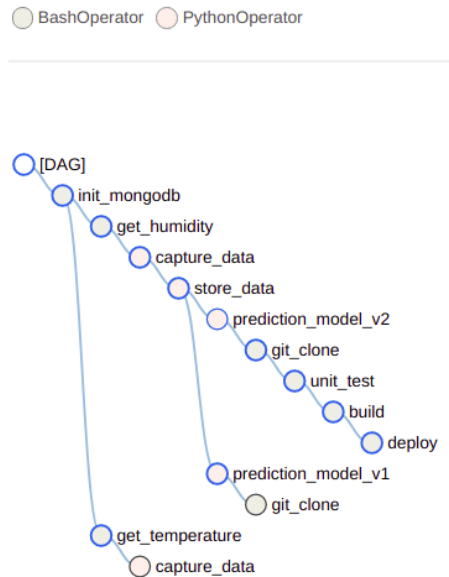


Figura 1: Diagrama de flujo de trabajo

### 3.1. Preparación del entorno

La preparación del entorno está formada por la tarea `init_mongodb`. Esta tarea se encarga de desplegar el servicio de la base de datos de **MongoDB**, usando para ello una operación **BashOperator** con el siguiente **comando bash**:

```
docker run -d --rm -p 27017:27017 --name mongodb mongo:latest
```

### 3.2. Descarga de datos A y B

Aquí se incluyen dos tareas que están al mismo nivel: `get_humidity` y `get_temperature`, que realizan la descarga de datos de la humedad y la temperatura que se usarán para la creación del modelo de predicción. En el diagrama de flujo de trabajo son dos operaciones **BashOperator**. El comando que se usa en cada operación es `curl` para descargar un fichero **.zip** que se guarda en el directorio `/tmp/`:

```
curl -o /tmp/humidity.csv.zip
https://raw.githubusercontent.com/manuparra/MaterialCC2020/master/humidity.csv.zip
```

```
curl -o /tmp/temperature.csv.zip
https://raw.githubusercontent.com/manuparra/MaterialCC2020/master/temperature.csv.zip
```

### 3.3. Captura de datos

La captura de datos se encuentra en la tarea `capture_data`, y es una operación del tipo **PythonOperator**, es decir, se usa una función en Python para su resolución. En este paso se deben **descomprimir**, **unificar** y **limpiar** los datos para administrar solamente la columna *date* (fecha) y *San Francisco* (ciudad), de modo que los datos se almacenen en un nuevo fichero **.csv** con estructura **DATE;TEMP;HUM**.

Este **PythonOperator** llama al método `capture_data` que se encuentra en el archivo **practica2\_dag.py** del repositorio. El siguiente enlace lleva a la línea exacta en el código:

```
https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/practica2_dag.py#L20
```

### 3.4. Almacenamiento de datos

Se corresponde con la tarea `store_data`. Una vez que se han capturado los datos en el archivo **forecast.csv**, esta tarea se encarga de almacenar dichos datos en la base de datos MongoDB. De nuevo, se trata de un **PythonOperator** que llama al método `store_data` disponible en el fichero **practica2\_dag.py** del repositorio. El siguiente enlace lleva a la línea exacta en el código:

```
https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/practica2_dag.py#L46
```

### 3.5. Creación de los modelos de predicción

La creación de los modelos de predicción se hacen en dos tareas que están al mismo nivel. Son dos tareas de tipo **PythonOperator**. La idea general de estas tareas es crear el modelo correspondiente y almacenarlo en un archivo **.pkl** para que se pueda hacer la predicción solamente llamando a dicho archivo.

La creación de los modelos suele tardar alrededor de **10 minutos** debido a la gran cantidad de datos recopilados de humedad y temperatura, con lo cual se ha decidido entrenar el modelo con solamente las **1.000 primeras filas** para reducir el tiempo de ejecución de forma considerable.

### 3.5.1. Modelo de predicción ARIMA

Este modelo de predicción corresponde a la versión 1 de la API. El método asociado al **PythonOperator** de esta tarea se llama `models_arima` y está definido en el archivo `practica2_dag.py` del repositorio. Se puede acceder mediante el siguiente enlace:

[https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/practica2\\_dag.py#L60](https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/practica2_dag.py#L60)

Los archivos creados se guardan en `/tmp/practica2/v1`.

### 3.5.2. Modelo de predicción Prophet

Este modelo de predicción corresponde a la versión 2 de la API. El método asociado al **PythonOperator** de esta tarea se llama `models_prophet` y está definido en el archivo `practica2_dag.py` del repositorio. Se puede acceder mediante el siguiente enlace:

[https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/practica2\\_dag.py#L113](https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/practica2_dag.py#L113)

Para el entrenamiento de los modelos creados en **Prophet** se ha seguido un pequeño tutorial propuesto por **Caner Dabakoglu** [13].

Los archivos creados se guardan en `/tmp/practica2/v2`.

### 3.5.3. Problemas encontrados

Hay dos problemas principales que se encontraron durante esta etapa:

- Al principio no se almacenaban los modelos entrenados en un archivo `.pkl`, con lo cual cada vez que se hacía una llamada a la API, la API llamaba el método de entrenamiento y luego hacía la predicción. Como es obvio, teniendo en cuenta el volumen de datos de entrenamiento, esto hacía que en cada llamada a la API se tardara más de 10 minutos en realizar una sola predicción. Es por esto que se decidió almacenar los modelos de predicción en archivos `.pkl`.
- El modelo de predicción **Prophet** parecía sencillo de usar pero se tardó un poco en hacer que el modelo funcionara debido a que no hay demasiada documentación buena acerca de este. También dio muchos problemas a la hora de desplegarlo como servicio debido a todas las librerías necesarias para su uso, pero eso es algo que se explicará en su sección más adelante.

### 3.6. Clonación del repositorio

Esta tarea está asociada a la que se llama `git_clone`. Se encarga de clonar el repositorio de Github para coger los archivos necesarios para el despliegue del servicio.

Esta tarea es del tipo **BashOperator** y el comando que tiene asociado es el siguiente:

```
cd /tmp/practica2 && git clone https://github.com/Jumacasni/MUII-CCSA.git && mv
MUII-CCSA/practica2/* .
```

Este comando copia los archivos necesarios en la carpeta `/tmp/practica2`.

### 3.7. Unit test

Una vez que se ha clonado el repositorio, se deben lanzar los tests que se han preparado como batería de pruebas con TDD para verificar el funcionamiento de ambas versiones de la API. Esto se realiza en la tarea llamada `unit_test` del flujo de trabajo.

Esta tarea es del tipo **BashOperator** y el comando que se ejecuta es el siguiente:

```
cd /tmp/practica2 && python3 tests.py
```

Los tests se encuentran en el archivo del repositorio llamado `tests.py`, al que se puede acceder mediante el siguiente enlace:

<https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/tests.py>

### 3.8. Build

Llegando casi al final de esta práctica, una vez que se han verificado los tests, se debe pasar a la construcción del contenedor que se va a desplegar. Esta tarea se llama `build` y es de tipo **BashOperator**, y el comando que tiene asociado es el siguiente:

```
cd /tmp/practica2 && docker build --rm -t api .
```

El **Dockerfile** usado para ello se encuentra en el repositorio y se puede acceder mediante el siguiente enlace:

<https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/Dockerfile>



Este **Dockerfile** ejecuta el comando **gunicorn** que permite desplegar la API realizada en Flask en un servidor. En concreto, se despliega en el puerto 8000. También se instalan muchas librerías debido a que **Prophet** las necesita, tal y como se va a explicar en la Sección 3.8.2.

Este proceso tarda entre **5-10 minutos** aproximadamente debido a la cantidad de librerías que se deben instalar.

### 3.8.1. Creación de la API

La **API** se ha creado usando **Flask**, tal y como se ha comentado anteriormente. Se encuentra en el archivo **api.py** del repositorio al que se puede acceder mediante el siguiente enlace:

<https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/api.py>

Se han creado tres rutas:

- `/`: se accede a través de `http://localhost:8000/` y simplemente devuelve un texto indicando que el microservicio está funcionando.
- `/servicio/v1/prediccion/<string:horas>`: que permite el acceso a las siguientes rutas que devuelven la predicción en formato JSON usando ARIMA:
  - `localhost:8000/servicio/v1/prediccion/24horas`
  - `localhost:8000/servicio/v1/prediccion/48horas`
  - `localhost:8000/servicio/v1/prediccion/72horas`
- `/servicio/v2/prediccion/<string:horas>`: que permite el acceso a las siguientes rutas que devuelven la predicción en formato JSON usando Prophet:
  - `localhost:8000/servicio/v2/prediccion/24horas`
  - `localhost:8000/servicio/v2/prediccion/48horas`
  - `localhost:8000/servicio/v2/prediccion/72horas`

### 3.8.2. Problemas encontrados

Esta etapa ha sido la más problemática debido al modelo de predicción **Prophet**. En mi ordenador funcionaba muy bien, pero al instalar esta librería en docker, empezaron a aparecer muchos errores de dependencias de otras librerías y demás.

Buscando información acerca de esto, las “soluciones” encontradas no llegaban a ayudarme del todo. Dos soluciones probadas y que no funcionaron:

- Usar **conda** para instalar **fbprophet**. Usé imágenes docker de **conda** pero no llegó a funcionar.

- Usar imágenes docker de **fbprophet** (<https://hub.docker.com/r/safakcirag/fbprophet>), entre otras. Debido a que son versiones bastante antiguas, había bastantes problemas entre dependencias y demás.

Finalmente encontré un perfil de Github con un Dockerfile que instalaba **fbprophet**, pero no funcionó en un principio. Sin embargo, estos errores parecían ser de versiones de librerías, así que decidí poner las versiones exactas que yo tenía en mi ordenador personal y funcionó.

Las versiones usadas junto con las librerías se encuentran en el fichero **requirements.txt**, disponible en el siguiente enlace del repositorio:

<https://github.com/Jumacasni/MUII-CCSA/blob/main/practica2/requirements.txt>

El Dockerfile encontrado en Github pertenece al usuario **stefanproell**, y se puede encontrar en el siguiente enlace:

<https://gist.github.com/stefanproell/89a00f3a2c18a7e9549a1de6f82cf0f8>

Tal y como se puede ver, al final que da una imagen con bastantes librerías y un poco pesada, pero todo funciona correctamente.

### 3.9. Despliegue

Este es el último paso del diagrama de flujo. El nombre de esta tarea es **deploy** y es de tipo **BashOperator**, que contiene el siguiente comando:

```
docker run --rm --name api -p 8000:8000 api
```

Este comando lo que hace es desplegar el servicio de la API creada en el puerto **8000**. Desde este momento, se pueden acceder a los distintos **endpoints**.

#### 3.9.1. Problemas encontrados

No fue un problema grave, pero al principio se usaba la orden `app.run(debug=True, host='0.0.0.0', port=8000)` para desplegar el servicio de Flask en el archivo **api.py**. Esto se puede ver en el **historial de commits** del repositorio de Github. Concretamente, en el siguiente enlace:

<https://github.com/Jumacasni/MUII-CCSA/commit/10b28f2bbeb9d58c8673cd28e407de1e0260bf97#diff-e13f9fc15b9d1f9c6e9f3c48c0709a47ffbde737be1ebb1137f34b29a2fbe08cR60>

El problema de esto es que esta forma de “desplegar” el servicio sólo debe usarse en **desarrollo**, y **nunca en producción**. Es por esta razón por la que, una vez verificado que todo funcionaba correctamente, se decidió usar **gunicorn** para su despliegue.

## 4. Demostración

En esta sección se va a demostrar el funcionamiento de todo el despliegue y funcionamiento de la API.

Lo primero que se debe hacer es colocar el archivo **practica2\_dag.py** en la carpeta **dags** que se encuentra en la instalación de **Apache Airflow** (ver Figura 2).

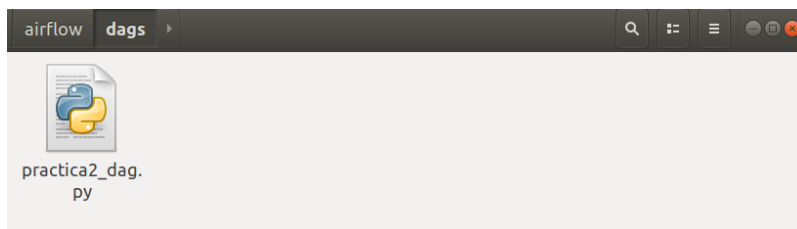


Figura 2: Archivo **practica2\_dag.py** en la carpeta **dags**

**Apache Airflow** tarda unos 30 segundos en reconocer el flujo de trabajo. Una vez que se ha reconocido en la versión web, se procede a activar el flujo de trabajo (ver Figura 3).

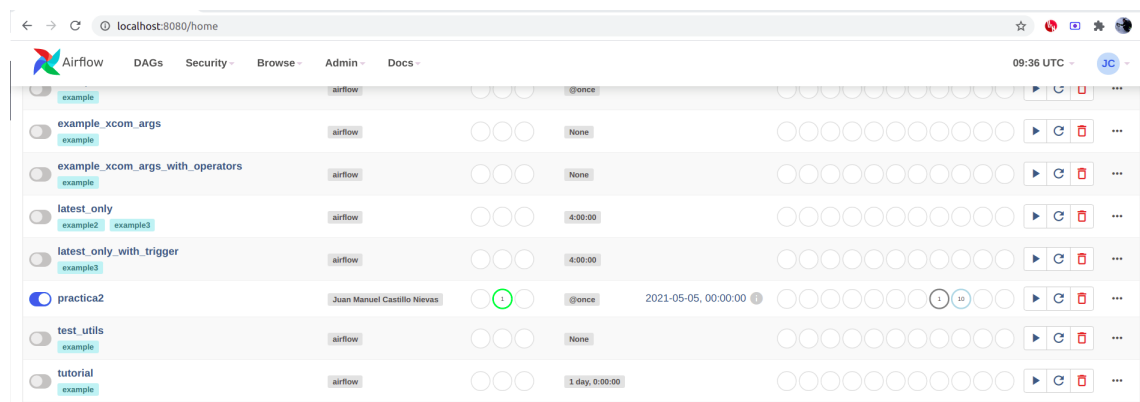


Figura 3: Activación del flujo de trabajo en **Apache Airflow**

Una vez que hayan terminado todas las tareas, el flujo de trabajo en **Apache Airflow** se verá como se muestra en la Figura 4. Esto significa que la API ya está desplegada y lista para su uso.

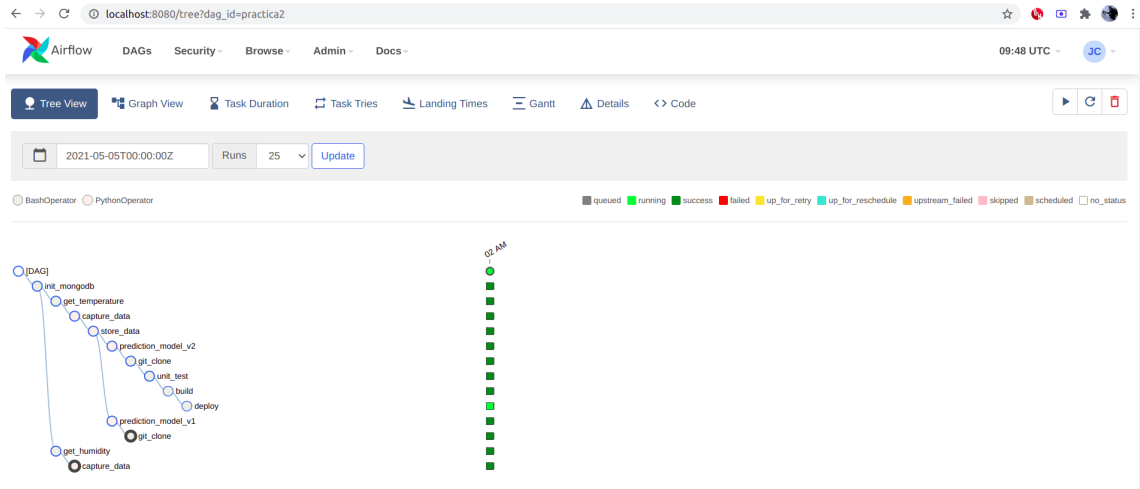


Figura 4: Despliegue final de la API con **Apache Airflow**

#### 4.1. Acceso a la API v1

Tal y como se comentó anteriormente, la versión 1 de la API está formada por los siguientes endpoints:

- localhost:8000/servicio/v1/prediccion/24horas (ver Figura 5)
- localhost:8000/servicio/v1/prediccion/48horas (ver Figura 6)
- localhost:8000/servicio/v1/prediccion/72horas (ver Figura 7)

```
localhost:8000/servicio/v1/prediccion/24horas

{"hour": "2021-05-07 09:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 10:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 11:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 12:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 13:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 14:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 15:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 16:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 17:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 18:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 19:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 20:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 21:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 22:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-07 23:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-08 00:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-08 01:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-08 02:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-08 03:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-08 04:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-08 05:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-08 06:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-08 07:00", "temp": 288.248925777887, "hum": 77.31199999999999},
{"hour": "2021-05-08 08:00", "temp": 288.248925777887, "hum": 77.31199999999999}
```

Figura 5: Acceso a localhost:8000/servicio/v1/prediccion/24horas

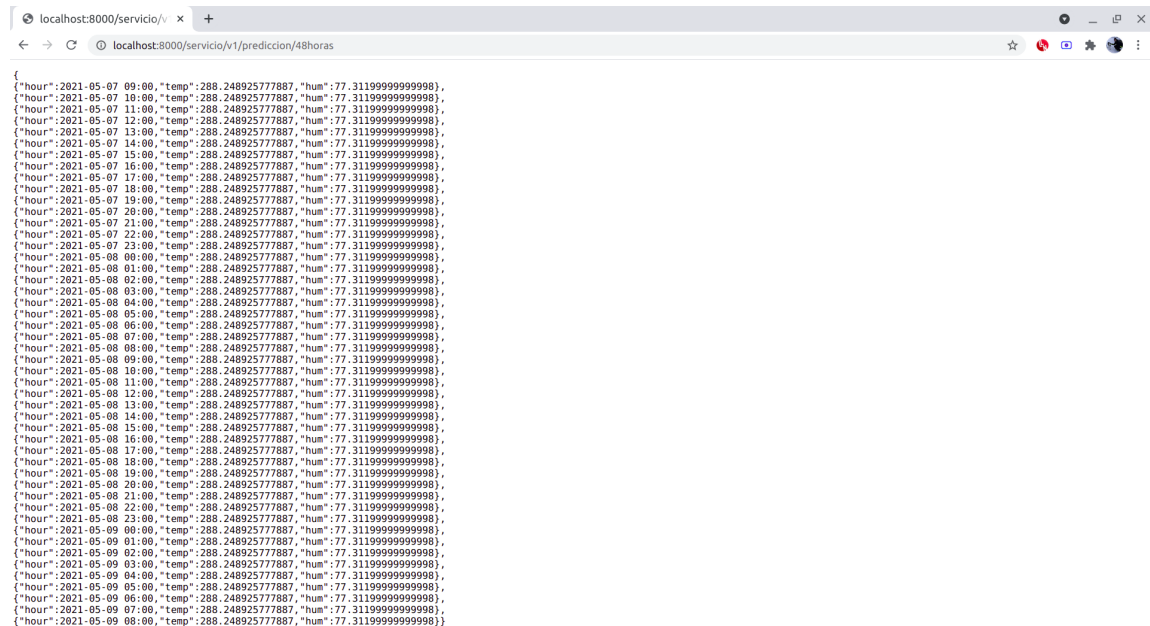


Figura 6: Acceso a localhost:8000/servicio/v1/prediccion/48horas

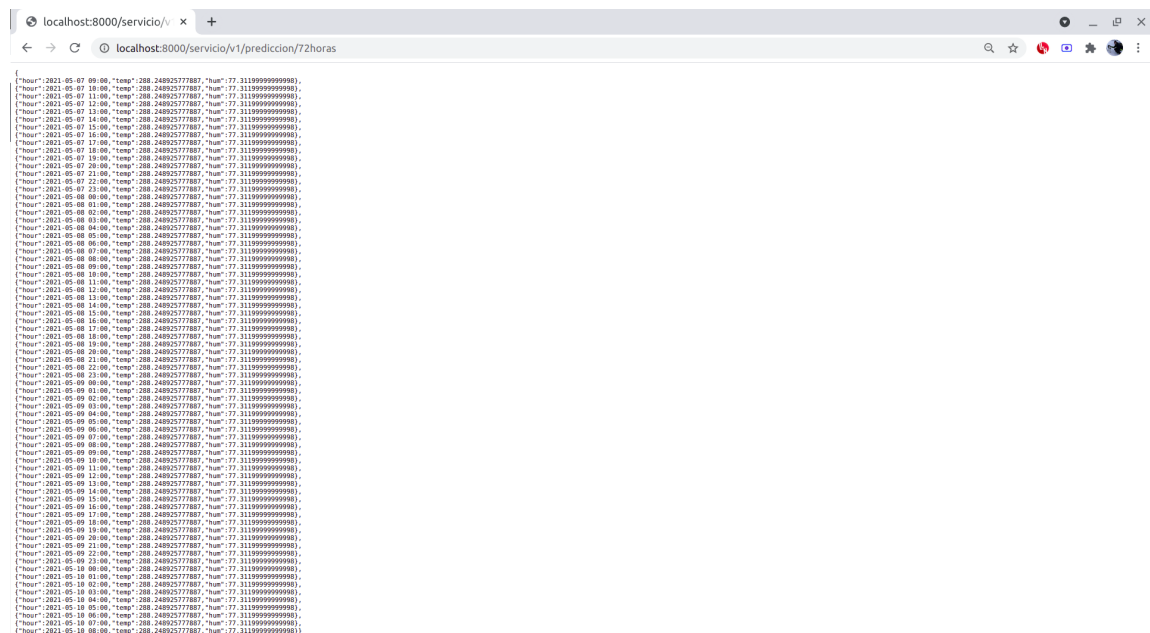


Figura 7: Acceso a localhost:8000/servicio/v1/prediccion/72horas

## 4.2. Acceso a la API v2

La versión 1 de la API está formada por los siguientes **endpoints**:

- `localhost:8000/servicio/v2/prediccion/24horas` (ver Figura 8)
- `localhost:8000/servicio/v2/prediccion/48horas` (ver Figura 9)
- `localhost:8000/servicio/v2/prediccion/72horas` (ver Figura 10)

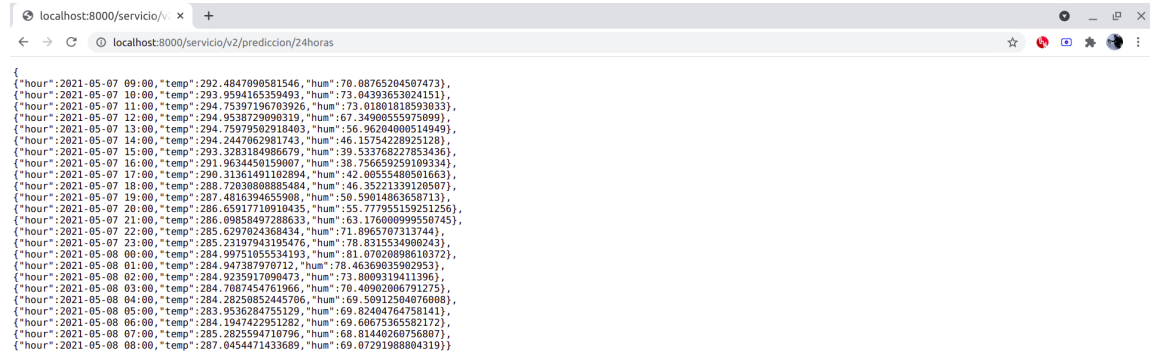
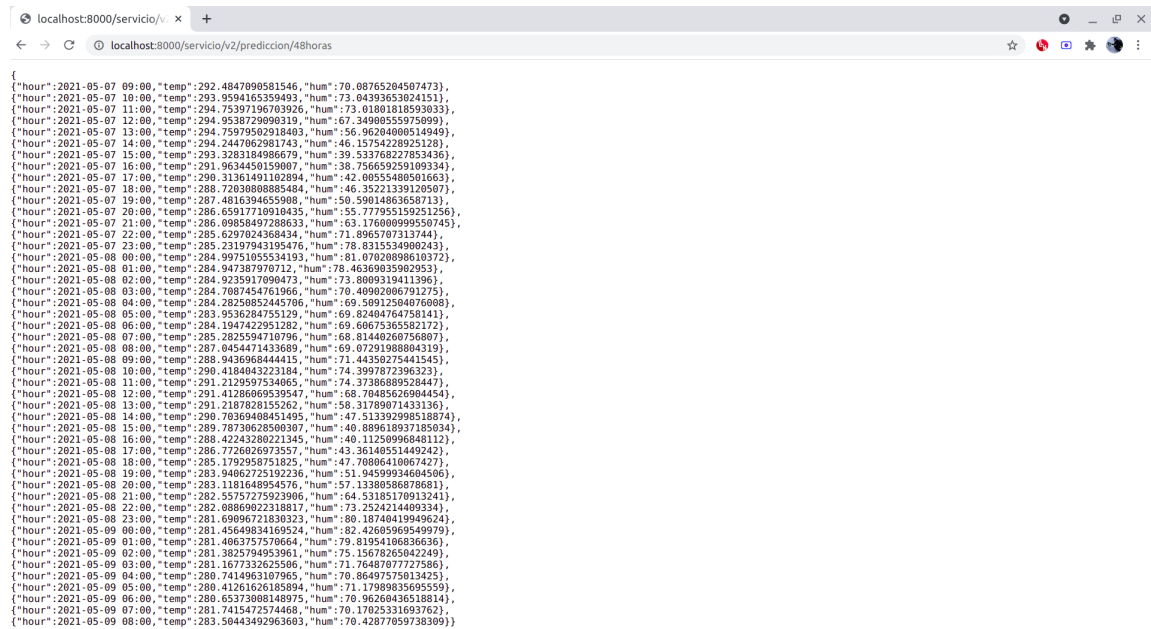
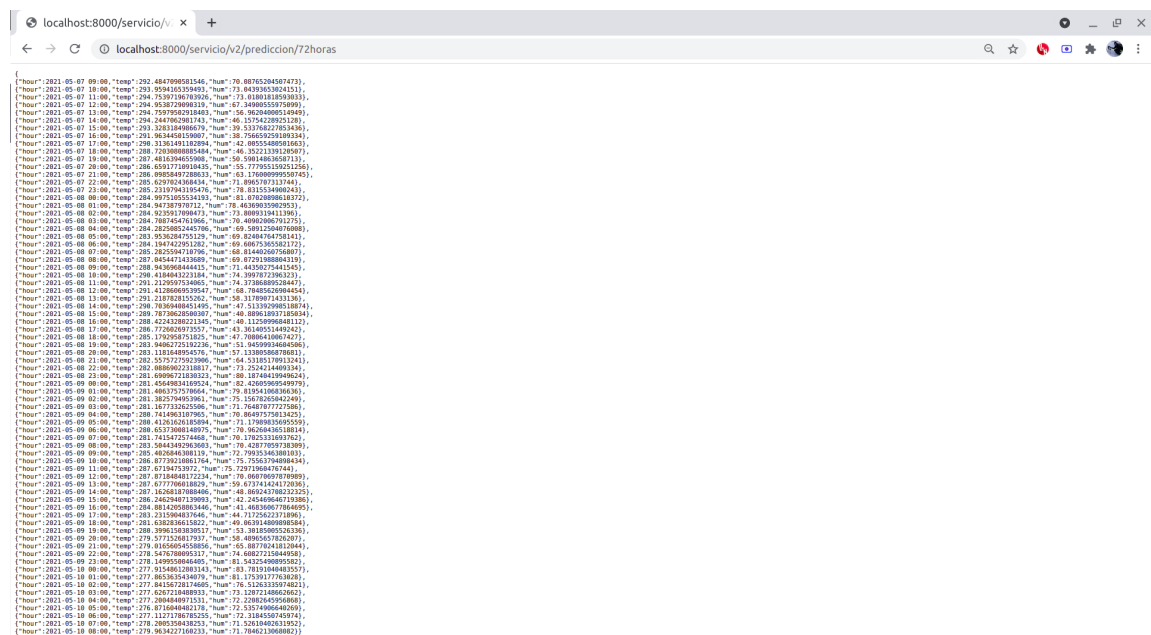


Figura 8: Acceso a `localhost:8000/servicio/v2/prediccion/24horas`



```
{
  "hour": "2021-05-07 09:00", "temp": 292.484799581546, "hum": 70.88765204507473},
  "hour": "2021-05-07 10:00", "temp": 293.9594165359493, "hum": 73.04393653024151},
  "hour": "2021-05-07 11:00", "temp": 294.75397196703926, "hum": 73.01801810593033},
  "hour": "2021-05-07 12:00", "temp": 294.9538729995319, "hum": 67.34909555759991},
  "hour": "2021-05-07 13:00", "temp": 294.75979502918403, "hum": 56.96204000514949},
  "hour": "2021-05-07 14:00", "temp": 294.2447062981743, "hum": 46.15754228925128},
  "hour": "2021-05-07 15:00", "temp": 293.3283184986679, "hum": 39.53376827853436},
  "hour": "2021-05-07 16:00", "temp": 291.9634450159007, "hum": 38.75665259109334},
  "hour": "2021-05-07 17:00", "temp": 290.31361491102894, "hum": 42.00555480501663},
  "hour": "2021-05-07 18:00", "temp": 288.7203888885484, "hum": 46.35221339120507},
  "hour": "2021-05-07 19:00", "temp": 287.4816394655908, "hum": 50.59014863658713},
  "hour": "2021-05-07 20:00", "temp": 286.65917710910435, "hum": 55.77955159251256},
  "hour": "2021-05-07 21:00", "temp": 286.09858497288633, "hum": 63.17600099550745},
  "hour": "2021-05-07 22:00", "temp": 285.6297024368434, "hum": 71.8965707313744},
  "hour": "2021-05-07 23:00", "temp": 285.23197943195476, "hum": 78.83155340002453},
  "hour": "2021-05-08 00:00", "temp": 284.9975105534193, "hum": 81.07028898610372},
  "hour": "2021-05-08 01:00", "temp": 284.947387970712, "hum": 79.46369035902953},
  "hour": "2021-05-08 02:00", "temp": 284.9235917099473, "hum": 73.8009319411296},
  "hour": "2021-05-08 03:00", "temp": 284.7087454761966, "hum": 70.40902066791275},
  "hour": "2021-05-08 04:00", "temp": 284.28205852445706, "hum": 69.50912504076008},
  "hour": "2021-05-08 05:00", "temp": 283.9536284755129, "hum": 69.82404764758141},
  "hour": "2021-05-08 06:00", "temp": 284.1047422951282, "hum": 69.60675365582172},
  "hour": "2021-05-08 07:00", "temp": 285.2825594710796, "hum": 68.81440260756807},
  "hour": "2021-05-08 08:00", "temp": 287.0454471433689, "hum": 69.07291988004319},
  "hour": "2021-05-08 09:00", "temp": 288.943696844415, "hum": 71.44350275441545},
  "hour": "2021-05-08 10:00", "temp": 290.4184043223184, "hum": 74.3997872396323},
  "hour": "2021-05-08 11:00", "temp": 291.2129597534065, "hum": 74.37386889528447},
  "hour": "2021-05-08 12:00", "temp": 291.4128606539547, "hum": 68.76485626904543},
  "hour": "2021-05-08 13:00", "temp": 291.2187028126262, "hum": 59.31789071431136},
  "hour": "2021-05-08 14:00", "temp": 290.70360488451495, "hum": 47.513392998518874},
  "hour": "2021-05-08 15:00", "temp": 289.7873628508307, "hum": 40.889618937185034},
  "hour": "2021-05-08 16:00", "temp": 288.4224320821345, "hum": 40.1125996040112},
  "hour": "2021-05-08 17:00", "temp": 286.7726026973557, "hum": 43.36140551449242},
  "hour": "2021-05-08 18:00", "temp": 285.179295751825, "hum": 47.70806410067427},
  "hour": "2021-05-08 19:00", "temp": 283.94062725192236, "hum": 51.94599346084506},
  "hour": "2021-05-08 20:00", "temp": 283.1181648054576, "hum": 57.1330836678681},
  "hour": "2021-05-08 21:00", "temp": 282.55757275923906, "hum": 64.53185170913241},
  "hour": "2021-05-08 22:00", "temp": 282.08869022318817, "hum": 73.2524214409334},
  "hour": "2021-05-08 23:00", "temp": 281.69096721830323, "hum": 80.187404199496243},
  "hour": "2021-05-09 00:00", "temp": 281.45649834169524, "hum": 82.42605969549979},
  "hour": "2021-05-09 01:00", "temp": 281.4063757570664, "hum": 79.81954106836636},
  "hour": "2021-05-09 02:00", "temp": 281.3825794953961, "hum": 75.156782659042249},
  "hour": "2021-05-09 03:00", "temp": 281.1677332625586, "hum": 71.7648777727586},
  "hour": "2021-05-09 04:00", "temp": 280.7414963107965, "hum": 70.86497575013425},
  "hour": "2021-05-09 05:00", "temp": 280.41261626185894, "hum": 71.17989836959559},
  "hour": "2021-05-09 06:00", "temp": 280.65373080140915, "hum": 70.96260436518013},
  "hour": "2021-05-09 07:00", "temp": 281.7415472574468, "hum": 70.17025331693762},
  "hour": "2021-05-09 08:00", "temp": 283.50443492963603, "hum": 70.42877859738309}
```

Figura 9: Acceso a localhost:8000/servicio/v2/prediccion/48horas



```
{
  "hour": "2021-05-07 09:00", "temp": 292.484799581546, "hum": 70.88765204507473},
  "hour": "2021-05-07 10:00", "temp": 293.9594165359493, "hum": 73.04393653024151},
  "hour": "2021-05-07 11:00", "temp": 294.75397196703926, "hum": 73.01801810593033},
  "hour": "2021-05-07 12:00", "temp": 294.9538729995319, "hum": 67.34909555759991},
  "hour": "2021-05-07 13:00", "temp": 294.75979502918403, "hum": 56.96204000514949},
  "hour": "2021-05-07 14:00", "temp": 294.2447062981743, "hum": 46.15754228925128},
  "hour": "2021-05-07 15:00", "temp": 293.3283184986679, "hum": 39.53376827853436},
  "hour": "2021-05-07 16:00", "temp": 291.9634450159007, "hum": 38.75665259109334},
  "hour": "2021-05-07 17:00", "temp": 290.31361491102894, "hum": 42.00555480501663},
  "hour": "2021-05-07 18:00", "temp": 288.7203888885484, "hum": 46.35221339120507},
  "hour": "2021-05-07 19:00", "temp": 287.4816394655908, "hum": 50.59014863658713},
  "hour": "2021-05-07 20:00", "temp": 286.65917710910435, "hum": 55.77955159251256},
  "hour": "2021-05-07 21:00", "temp": 286.09858497288633, "hum": 63.17600099550745},
  "hour": "2021-05-07 22:00", "temp": 285.6297024368434, "hum": 71.8965707313744},
  "hour": "2021-05-07 23:00", "temp": 285.23197943195476, "hum": 78.83155340002453},
  "hour": "2021-05-08 00:00", "temp": 284.9975105534193, "hum": 81.07028898610372},
  "hour": "2021-05-08 01:00", "temp": 284.947387970712, "hum": 79.46369035902953},
  "hour": "2021-05-08 02:00", "temp": 284.9235917099473, "hum": 73.8009319411296},
  "hour": "2021-05-08 03:00", "temp": 284.7087454761966, "hum": 70.40902066791275},
  "hour": "2021-05-08 04:00", "temp": 284.28205852445706, "hum": 69.50912504076008},
  "hour": "2021-05-08 05:00", "temp": 283.9536284755129, "hum": 69.82404764758141},
  "hour": "2021-05-08 06:00", "temp": 285.2825594710796, "hum": 68.81440260756807},
  "hour": "2021-05-08 07:00", "temp": 287.0454471433689, "hum": 69.07291988004319},
  "hour": "2021-05-08 08:00", "temp": 288.943696844415, "hum": 71.44350275441545},
  "hour": "2021-05-08 09:00", "temp": 290.4184043223184, "hum": 74.3997872396323},
  "hour": "2021-05-08 10:00", "temp": 291.2129597534065, "hum": 74.37386889528447},
  "hour": "2021-05-08 11:00", "temp": 291.4128606539547, "hum": 68.76485626904543},
  "hour": "2021-05-08 12:00", "temp": 291.2187028126262, "hum": 59.31789071431136},
  "hour": "2021-05-08 13:00", "temp": 290.70360488451495, "hum": 47.513392998518874},
  "hour": "2021-05-08 14:00", "temp": 289.7873628508307, "hum": 40.889618937185034},
  "hour": "2021-05-08 15:00", "temp": 288.4224320821345, "hum": 40.1125996040112},
  "hour": "2021-05-08 16:00", "temp": 286.7726026973557, "hum": 43.36140551449242},
  "hour": "2021-05-08 17:00", "temp": 285.179295751825, "hum": 47.70806410067427},
  "hour": "2021-05-08 18:00", "temp": 283.94062725192236, "hum": 51.94599346084506},
  "hour": "2021-05-08 19:00", "temp": 283.1181648054576, "hum": 57.1330836678681},
  "hour": "2021-05-08 20:00", "temp": 282.55757275923906, "hum": 64.53185170913241},
  "hour": "2021-05-08 21:00", "temp": 282.08869022318817, "hum": 73.2524214409334},
  "hour": "2021-05-08 22:00", "temp": 281.69096721830323, "hum": 80.187404199496243},
  "hour": "2021-05-08 23:00", "temp": 281.45649834169524, "hum": 82.42605969549979},
  "hour": "2021-05-09 00:00", "temp": 281.4063757570664, "hum": 79.81954106836636},
  "hour": "2021-05-09 01:00", "temp": 281.3825794953961, "hum": 75.156782659042249},
  "hour": "2021-05-09 02:00", "temp": 281.1677332625586, "hum": 71.7648777727586},
  "hour": "2021-05-09 03:00", "temp": 280.7414963107965, "hum": 70.86497575013425},
  "hour": "2021-05-09 04:00", "temp": 280.41261626185894, "hum": 71.17989836959559},
  "hour": "2021-05-09 05:00", "temp": 280.65373080140915, "hum": 70.96260436518013},
  "hour": "2021-05-09 06:00", "temp": 281.7415472574468, "hum": 70.17025331693762},
  "hour": "2021-05-09 07:00", "temp": 283.50443492963603, "hum": 70.42877859738309},
  "hour": "2021-05-09 08:00", "temp": 285.426486388119, "hum": 72.7993346388103},
  "hour": "2021-05-09 09:00", "temp": 286.873921881194, "hum": 75.7550374088404},
  "hour": "2021-05-09 10:00", "temp": 287.0719473972, "hum": 79.729719048744},
  "hour": "2021-05-09 11:00", "temp": 287.0719473972, "hum": 79.729719048744},
  "hour": "2021-05-09 12:00", "temp": 287.0719473972, "hum": 79.729719048744},
  "hour": "2021-05-09 13:00", "temp": 287.0719473972, "hum": 79.729719048744},
  "hour": "2021-05-09 14:00", "temp": 287.0719473972, "hum": 79.729719048744},
  "hour": "2021-05-09 15:00", "temp": 286.2462947139093, "hum": 42.2454046173080},
  "hour": "2021-05-09 16:00", "temp": 284.8814200803446, "hum": 41.4683607704809},
  "hour": "2021-05-09 17:00", "temp": 283.215040817646, "hum": 44.7125422711890},
  "hour": "2021-05-09 18:00", "temp": 281.6382361582, "hum": 49.863140008084},
  "hour": "2021-05-09 19:00", "temp": 280.399610830517, "hum": 53.20200253236},
  "hour": "2021-05-09 20:00", "temp": 279.5715201792, "hum": 58.4896261076262},
  "hour": "2021-05-09 21:00", "temp": 278.616565455856, "hum": 65.8077041212044},
  "hour": "2021-05-09 22:00", "temp": 278.4778000517, "hum": 74.6887215046048},
  "hour": "2021-05-09 23:00", "temp": 278.145950846405, "hum": 81.343240080502},
  "hour": "2021-05-10 00:00", "temp": 277.935460120815, "hum": 83.703104002037},
  "hour": "2021-05-10 01:00", "temp": 277.805353540479, "hum": 81.72917730268},
  "hour": "2021-05-10 02:00", "temp": 277.84156728174695, "hum": 76.5235335974021},
  "hour": "2021-05-10 03:00", "temp": 277.62071800951, "hum": 72.1251048602},
  "hour": "2021-05-10 04:00", "temp": 277.280464091131, "hum": 72.2206244950608},
  "hour": "2021-05-10 05:00", "temp": 277.871604462179, "hum": 72.1251048602},
  "hour": "2021-05-10 06:00", "temp": 277.1127178076255, "hum": 72.138458740974},
  "hour": "2021-05-10 07:00", "temp": 278.20037988223, "hum": 71.128104001392},
  "hour": "2021-05-10 08:00", "temp": 279.963422108233, "hum": 71.7046213080823}
```

Figura 10: Acceso a localhost:8000/servicio/v2/prediccion/72horas

## 5. Bibliografia

- [1] Python. <https://www.python.org/>.
- [2] Flask. <https://flask.palletsprojects.com/en/1.1.x/>.
- [3] Apache Airflow. <http://airflow.apache.org/>.
- [4] MongoDB. <https://www.mongodb.com/es>.
- [5] pymongo. <https://pypi.org/project/pymongo/>.
- [6] unittest. <https://docs.python.org/3/library/unittest.html>.
- [7] pandas. <https://pandas.pydata.org/>.
- [8] Statsmodels. <https://www.statsmodels.org/stable/index.html>.
- [9] ARIMA. [https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima\\_model.ARIMA.html](https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARIMA.html).
- [10] Prophet. [https://facebook.github.io/prophet/docs/quick\\_start.html](https://facebook.github.io/prophet/docs/quick_start.html).
- [11] Pickle. <https://docs.python.org/3/library/pickle.html>.
- [12] gunicorn. <https://gunicorn.org/>.
- [13] Caner Dabakoglu. Time series forecasting - arima, lstm, prophet with python. <https://medium.com/@cdabakoglu/time-series-forecasting-arima-lstm-prophet-with-python-e73a750a9887>, 2019.