Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Máster Profesional en Ingeniería Informática

Curso 2020/2021

PRÁCTICA 2 - AIRFLOW

Cloud Computing

Breve descripción

Despliegue de un servicio Cloud Native

Autor

Álvaro de la Flor Bonilla (alvdebon@correo.ugr.es) 15408846-L

RESUMEN

Esta práctica consiste en desarrollar un sistema completo de predicción de temperatura y humedad para una ubicación determinada. Para ese desarrollo debe cubrir todos los aspectos del proceso, como la toma de datos, el procesamiento, el almacenamiento o la publicación de los servicios entre muchos otros. Con esto lo que se consigue es el despliegue de un servicio Cloud Native completo desde la adquisición del código fuente, hasta la ejecución de contenedores y finalmente desplegar el servicio que finalmente entregará una API de tipo HTTP RESTful que para la predicción de temperatura y humedad.

ÍNDICE DEL PROYECTO

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

Para la resolución de este problema se han diseñado dos API de tipo RESTful independientes.

Para el primer caso la API logra realizar una predicción tanto de humedad como de temperatura haciendo uso de un modelo autorregresivo integrado de promedio móvil (*"ARIMA"*). Para ello se hace un uso auxiliar de datos meteorológicos anteriores proporcionados por el repositorio https://github.com/manuparra/MaterialCC2020 del cual hemos obtenido un fichero en formato *"CSV"*, tanto para los datos de humedad como para los de temperatura.

Para la segunda versión de la API hemos decidido utilizar la extracción de datos a partir de otra pagina que cede (a través de una API) sus modelos predictivos.

Se ha usado https://www.weatherapi.com/ en esta ocasión. Esta web permite obtener datos meteorológicos (tanto humedad como temperatura) para un máximo de 72 horas, lo cual es justo lo que necesitamos para nuestro proyecto.

2 RESOLUCIÓN DE LAS TAREAS

2.1 Código

Todo el código desarrollado para esta práctica se encuentra disponible en el repositorio público https://github.com/alvarodelaflor/cc_airflow/.

2.2 Explicación del desarrollo

En este apartado explicaremos cada una de las tareas que hemos realizado para la finalización de la práctica.

2.2.1 Descarga y procesamiento de datos

Básicamente, nos hemos centrado en el desarrollo del archivo *"save.py"*, en concreto en el método *"get_data()"*.

Lo primero que hacemos es descargar los dos archivos de datos que se nos cede para el procesamiento de datos disponibles en:

- https://github.com/manuparra/MaterialCC2020/blob/master/humidity.csv.zip
- https://github.com/manuparra/MaterialCC2020/blob/master/temperature.csv.zip

A partir de estos dos archivos nos centramos en su poda y tratamiento. Tal y como se nos indica, hemos unificado ambos construyendo un único *"dataframe"* que cuenta con los datos de fecha, humedad y temperatura de la ciudad de San Francisco, siguiente la estructura *"(DATE;TEMP;HUM)"*.

2.2.2 Almacenamiento de datos

Una vez obtenido estos datos le otorgamos consistencia, es decir, almacenados los datos tratados en una base de datos.

Para nuestra práctica ha sido elegida MongoDB Atlas. A partir de estos datos trabajaremos en el modelo predictivo para el caso del modelo de API V1, la cual utiliza *"ARIMA"* para construir un modelo predictivo.

En el caso de la API V2 no es necesario utilizar ninguna base de datos ni tratamiento de datos ya que recopilamos los datos directamente a partir de una API externa.

Su uso es muy sencillo, simplemente paso previo a su utilización mediante código Python deberemos crear un clúster en esta dirección https://www.mongodb.com/es/cloud/atlas. En esta misma web se nos indicará el proceso para obtener acceso al clúster creado.

2.2.3 API RESTful, creación de microservicios

Se han desarrollado dos API siguiendo el microframework *"Flask"*, *"api_v1.py"* y *"api_v2.py"*. Ambas siguen una misma estructura, permiten acceder a cuatro rutas, de las cuales son (téngase en cuenta que ***"vX"*** hace referencia a la versión de la API, pudiendo ser en nuestro caso *"v1"* o *"v2"*):

1. **EndPoint 1 → /**

   En esta ruta se detallan las distintas rutas disponibles y que se puede hacer en cada una de ellas.

Ilustración 1 – Página inicial API V1

Ilustración 2 – Página inicial API V2

2. **EndPoint 2 → /servicio/vX/prediccion/24horas/**
3. **EndPoint 3 → /servicio/vX/prediccion/48horas/**
4. **EndPoint 4 → /servicio/vX/prediccion/72horas/**

   En estas últimas direcciones se devuelve la predicción meteorológica para las próximas 24, 48 y 72 horas correspondientemente. La información es mostrada al usuario en formato JSON como puede verse en la siguiente imagen para las dos variantes construidas.

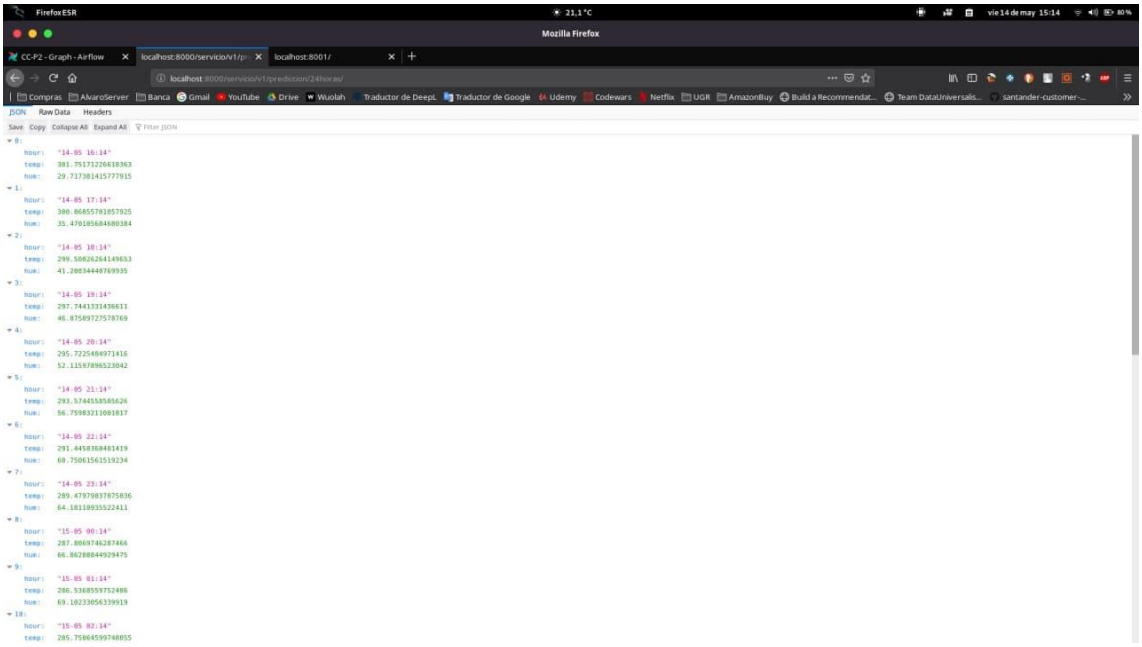

Ilustración 3 – Consulta a la API V1

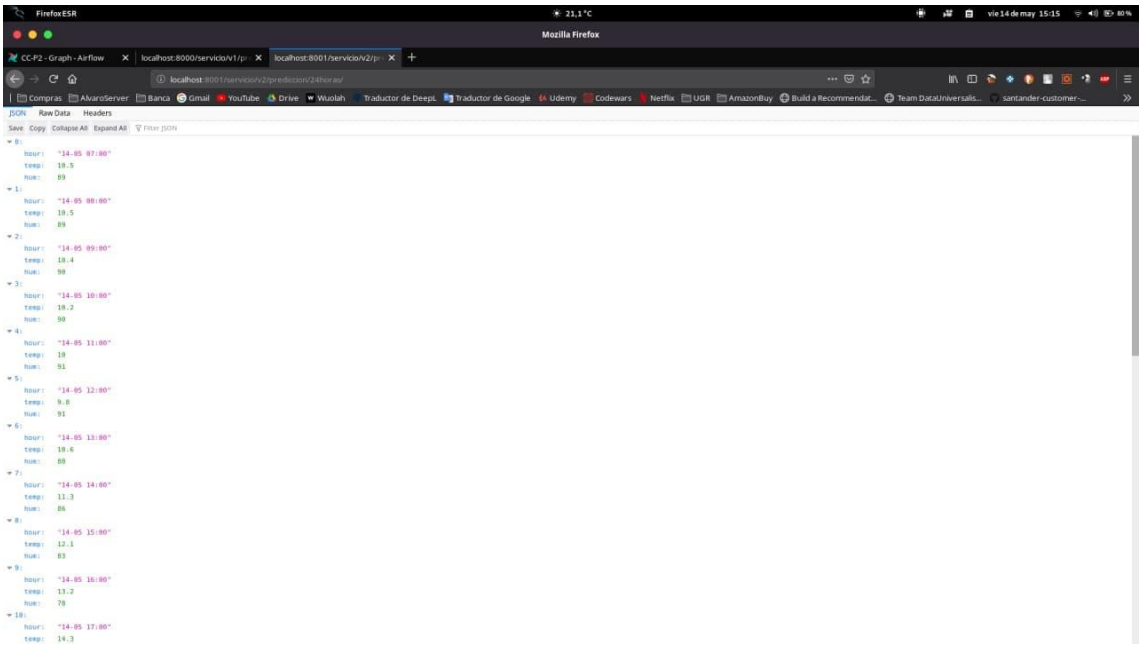

Ilustración 4 – Consulta a la API V2

2.2.4 ARIMA, creación del modelo predictivo

Para la elaboración de este modelo predictivo hemos trabajado en el archivo *"prediction.py"*. En concreto, para la predicción realizada por *"ARIMA"* utilizamos el método *"get_predictions_arima(period)" y "create_arima_model(dataframe)"*. Este método se basa en realizar los siguientes pasos:

1. Obtener los datos almacenados (previamente procesados) de MongoDB Atlas.
2. Si existe un modelo previo, lo descomprime.
3. Si no existe un modelo previo → *"create_arima_model(dataframe)"*:

3.1 Utilización de *"auto_arima"* para la creación del modelo.

3.2 Comprensión en extensión *".zip"* del nuevo modelo.

4. Utiliza el modelo creado para predecir la temperatura en el período que solicita el usuario.
5. Utiliza el modelo creado para predecir la humedad en el período que solicita el usuario.
6. Devuelve resultado en formato JSON.

2.2.5 WeatherAPI, obtención de predicciones

Para la segunda versión de nuestra API (*"V2"*) hemos decidido utilizar la web https://www.weatherapi.com/ que permite obtener una predicción meteorológica tanto de temperatura como humedad hasta un máximo de 72 horas.

Una vez más, para su uso hemos utilizado el archivo *"prediction.py"*, en concreto el método *"get_prediction_wheatherapi(period)"*. En este proceso realizamos los siguientes pasos:

1. Realizamos consulta a la web haciendo uso de la *"key"* obtenida previamente y parametrizando el número de horas que solicita el usuario (elegimos San Francisco como búsqueda por defecto).

2. Procesamos los datos para quedarnos únicamente con la temperatura, humedad y fecha.
3. Construimos el nuevo *"dataframe"* con el formato que se nos solicita.
4. Devuelve el resultado en formato JSON.

2.2.6 Elaboración de los Test

La configuración de las pruebas ha sido configurada en el archivo *"test.py"*. Para cada una de las versiones se han elaborado 3 pruebas iguales:

1. Evaluación del período solo como número, no como cadena.
2. Objeto devuelto como lista
3. Estado operativo de la funcionalidad de la API.

Para la elaboración de estos tests ha sido utilizada la librería *"PyTest"*.

2.2.7 Dockerización de la aplicación

Una vez tenemos nuestra aplicación completamente funcional, antes de aplicar AirFlow nos queda dockerizar todo lo que hemos hecho. El funcionamiento es muy sencillo y lo único que hemos tenido que realizar es construir dos ficheros *"Dockerfile"* concretamente uno para cada versión realizada (*"DockerfileV1"* y *"DockerfileV2"*). En estos ficheros lo único que hacemos es establecer el puerto e inicializar nuestra aplicación mediante *"gunicorn"*.

2.3 AirFlow

Por último, una vez tenemos todo nuestro desarrollo finalizado podemos comenzar con el desarrollo de las tareas de *"AirFlow"*. Toda la configuración se encuentra disponible en el archivo *"setup_airflow.py"*.

El orden en el que se ejecutarán las tareas sigue el siguiente gráfico, en el que mediante AirFlow se ha ejecutado el flujo de tareas sin ningún fallo.

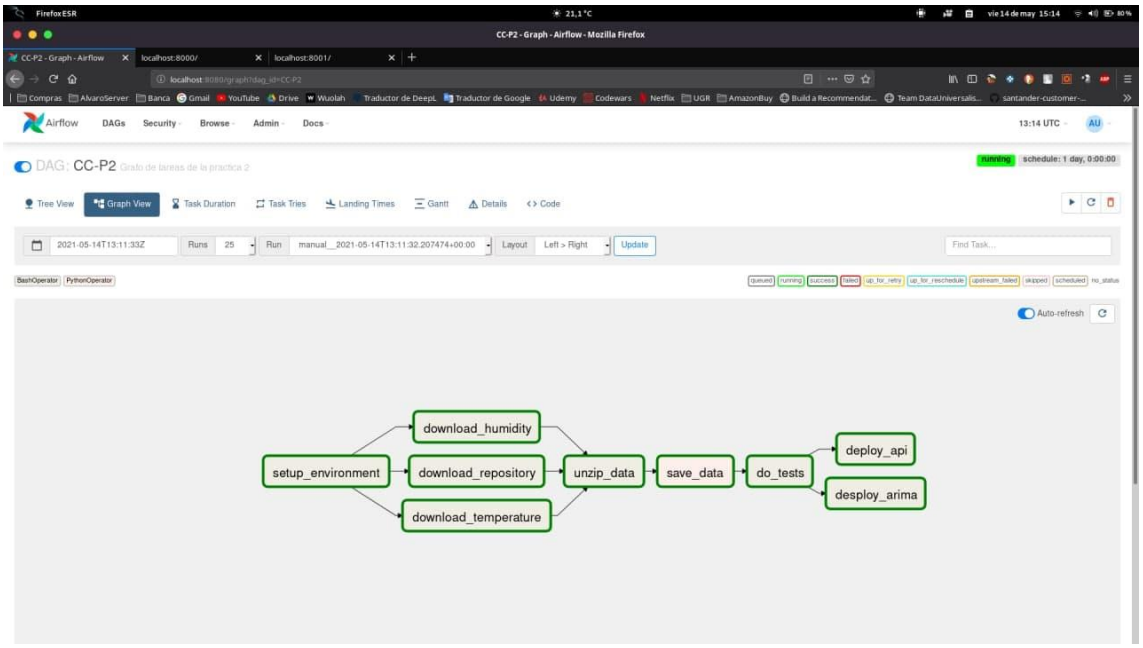

Ilustración 5 – Ejecución de AirFlow y diagrama de tareas

2.3.1 SetupEnvironment

En esta tarea se encarga de preparar el entorno temporal donde almacenaremos los datos con los que construir el modelo predictivo, así como todo el código desarrollado anteriormente.

2.3.2 DownloadHumidity

Descarga del fichero de humedad en formato *"CSV"* directamente del repositorio de la asignatura.

2.3.3 DownloadTemperature

Descarga del fichero de temperatura en formato *"CSV"* directamente del repositorio de la asignatura.

2.3.4 DownloadRepository

Descarga todo el repositorio que contiene el código desarrollado y explicado anteriormente. Además, crea un nuevo directorio para almacenarlo.

2.3.5 UnzipData

Descomprime los archivos que contiene los archivos *"CSV"* descargados anteriormente.

2.3.6 SaveData

Esta tarea ejecuta el proceso en el que se ejecuta el método *"get_data_airflow()"* el cual almacena en MongoDB el *"dataframe"* preprocesado como hemos explicado en pasos anteriores para quedarnos con un único archivo que contenga la temperatura y humedad en San Francisco para una fecha determinada.

Se utiliza una función *"Python"* en lugar de *"bash"* a diferencia de los casos anteriores.

2.3.7 DoTests

Se ejecutan las pruebas que se explicaron en el punto anterior. En caso de que no den fallo se pasará al despliegue con *"Docker"*.

2.3.8 DeployArima

Despliega la API V1 en el puerto 8000.

2.3.9 DeployApi

Despliega la API V2 en el puerto 8001.

Destacar que existen tareas que se han realizado en paralelo. Por ejemplo las tres descargas de archivos (*"DownloadHumidity"*, *"DownloadTemperature"* y *"DownloadRepository"*) y los dos despliegues usando *"Docker"* (*"DeployArima"* y *"DeployApi"*).

3 CONCLUSIONES

En este apartado nos gustaría destacar que nos ha resultado bastante interesante esta práctica, ya que anteriormente no habíamos visto elementos de orquestación a este nivel.

Como el problema más serio que hemos detectado es que en ocasiones el flujo de *“AirFlow”* no ha funcionado tan bien como esperábamos, por ejemplo se producía un registro erróneo de log indicando que la tarea de despliegue no había podido ser finalizada por puerto ocupado, a pesar de que se había realizado el despliegue correctamente en esas ocasiones.