Máster Universitario en Ingeniería Informática

Cloud Computing: Servicios y Aplicaciones

PRÁCTICA 2
Despliegue de un servicio Cloud Native

UNIVERSIDAD DE GRANADA

Carlos Santiago Sánchez Muñoz

Email: carlossamu7@correo.ugr.es

DNI: 75931715K

15 de julio de 2021

Índice

1. Introducción

La práctica a desarrollar está orientada a desplegar un servicio Cloud Native. Se va a implementar un sistema completo de predicción de temperatura y humedad para una ubicación determinada, en este caso San Francisco.

Para ello será necesaria la toma de datos, el preprocesamiento y almacenamiento de los mismo y otras tareas que se explicarán a continuación. El objetivo de esto 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 para la predicción de temperatura y humedad.

Serán dos APIs con las siguientes operaciones, una para cada versión:

```
HTTP GET /servicio/vi/prediccion/24horas/
HTTP GET /servicio/vi/prediccion/48horas/
HTTP GET /servicio/vi/prediccion/72horas/
```

donde `vi` puede ser `v1` y `v2`.

Se usará la herramienta de flujo de trabajo `Airflow`. Sencillamente la base de datos tiene que estar inicializada por lo que la primera vez que se use `Airflow` es necesario ejecutar en una terminal `airflow db init` y posteriormente la orden `airflow db check` comprobará si se puede alcanzar.

Posteriormente introduciremos los grafos dirigidos acíclicos (DAGs) en la carpeta `airflow/dags` que existe tras la instalación y lanzamos el planificador con `ariflow scheduler`. Por úlitmo lanzamos el servicio web en el puerto que se desee. Por ejemplo `airflow webserver -p 8080`. Accedemos al sitio web y encontramos los DAGs:

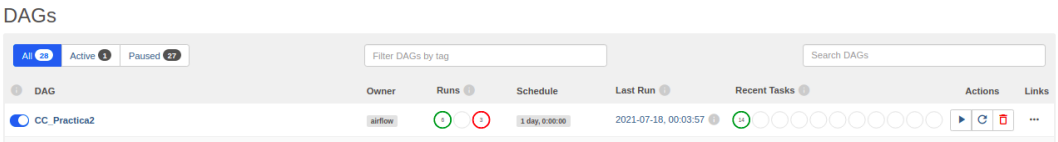

Imagen 1: DAGs disponibles en Airflow

Todo aquello que ha sido implementado está disponible en GitHub. En concreto, el trabajo central que son las tareas están aquí.

2. Resolución

En esta sección se va a detallar la explicación de la resolución de cada tarea. El código correspondiente quedará enlazado a GitHub para que pueda consultarse.

2.1. Crear la carpeta temporal

Esta tarea es un `BashOperator` que básicamente crea una carpeta temporal llamada practica2.

2.2. Descargar datos de temperatura

La siguiente tarea es la descarga de datos de temperatura del repositorio del profesor de la asignatura. Estos datos son un `csv` comprimido en `zip`.

2.3. Descargar datos de humedad

Tarea idéntica pero para la humedad.

2.4. Descomprimir los datos

Una vez descargados los dos zips con los datos el siguiente paso es descomprimirlos. Para ello un nuevo `BashOperator` que mediante la orden `unzip` los descomprime. La tarea está disponible aquí.

2.5. Preprocesar los datos

La tarea de preprocesado de los datos es el primer `PythonOperator` del DAG. Utiliza la función implementada aquí.

El preprocesado lee los dos `csv` usando `pandas` y construye un nuevo dataframe con los datos de temperatura y humedad de San Francisco. Reemplaza los valores perdidos utilizando la media. Asimismo almacena la fecha. Finalmente escribe dichos datos en un nuevo `csv` resumen.

2.6. Almacenar datos en Mongo DB

La base de datos elegida donde inyectar los datos es Mongo. De hecho se usa Mongo DB Atlas en donde es necesario registrarse, asociarle una organización, crear usuario que tendrá acceso a la base de datosy darle acceso de red (en concreto sólo he autorizado mi IP por lo que a una tercera persona no le funcionaría salvo que me de su IP). Tras estos pasos ya se puede inyectar datos a la base de datos.

En mi caso la inyección la hago desde mi aplicación usando los drivers nativos de Mongo DB.

2.7. Descargar APIs

Esta tarea (descarga API 1 y descarga API 2) simplemente descarga mediante `BashOperator` dos repositorios con cada una de las APIs disponibles en GitHub.

La primera API para predecir temperaturas y humedades está disponible en este repositorio. Utiliza el modelo ARIMA dado por el profesor de la asignatura.
La segunda se encuentra en este repositorio y se conecta a Open weather para la predicción. El mejor tutorial y forma de comprobar la conexión está en este enlace.

Al repositorio de github no he subido el fichero `config.py` en donde esta la `API Key` necesaria motivos de privacidad. Para realizar esto consulté este link. Es necesario esperar una media hora hasta que dan de alta la `API Key` creada.

2.8. Testear APIs

Esta tarea lanza los tests de cada una de las APIs. Comprueba que el estado que devuelve la petición HTTP sea el 200 en cada una de las rutas pedidas. Dichos test usan la librería `unittest` de Python y se lanzan con `pytest`.

- Test de la API1 y tarea de test de la API 1.
- Test de la API2 y tarea de test de la API 2.

2.9. Construir APIs

Si los tests se han pasado satisfactoriamente entonces las tareas del DAG avanzan y el siguiente paso es la construcción de los contenedores.

- Construcción del contenedor de la API 1.
- Construcción del contenedor de la API 2.

2.10. Desplegar APIs

El último paso es levantar los contenedores. Las dos tareas para levantar los contenedores de cada una de las APIs son las que siguen:

- Levantando el contenedor de la API 1.
- Levantando el contenedor de la API 2.

3. Resultados

En esta sección se van a exponer los diferentes resultados de la ejecución así como algunas gráficas relevantes.

3.1. Árbol

Airflow permite visualizar el árbol de tareas y los resultados de las diferentes ejecuciones. Resulta muy ilustrativo.

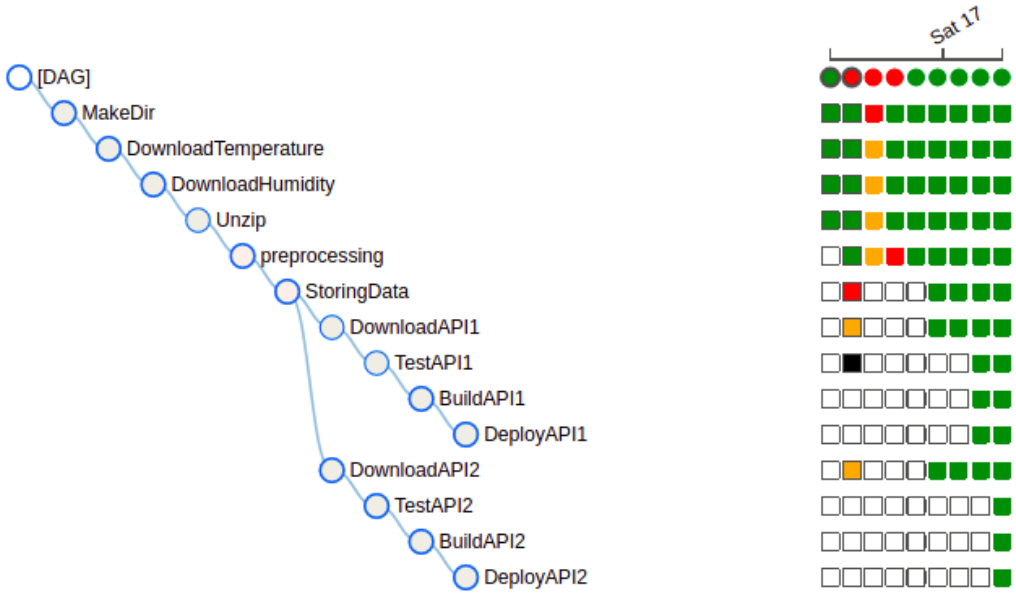

Imagen 2: Árbol

Se observa el avance de ejecuciones en el tiempo en donde cada vez hay más tareas y la última ejecución tiene éxito en todas y cada una de las tareas.

3.2. Grafo

La versión grafo de las tareas implementadas es el siguiente:

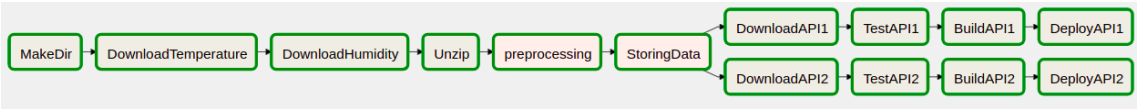

Imagen 3: Grafo

3.3. Diagrama de Gant

El servidor web de Airflow ofrece asimismo la posibilidad de exportar un diagrama de Gant. Este diagrama muestra el avance de las tareas en el tiempo.

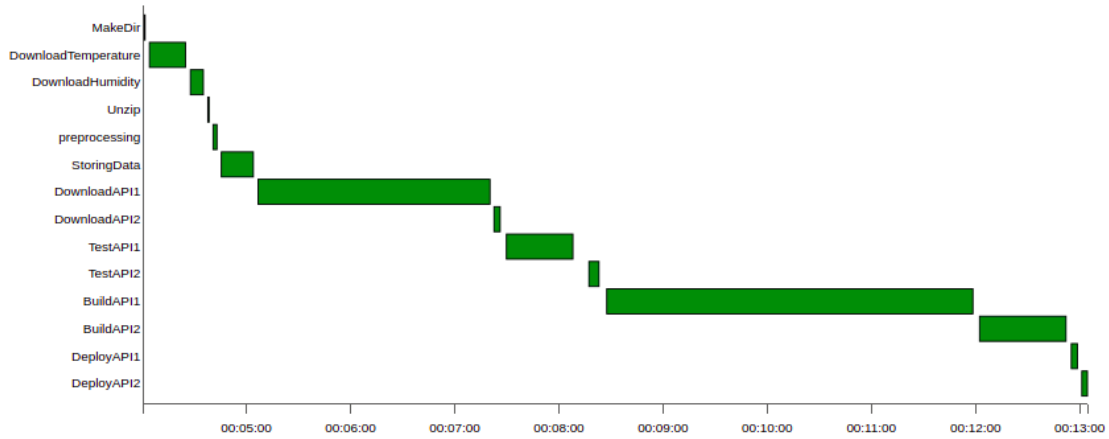

Imagen 4: Diagrama de Gant

3.4. Base de datos

Efectivamente entrando a la cuenta de Mongo DB Atlas observo que se han insertado las instancias pertinentes. Existe el conjunto de datos CCAirflow SanFrancisco tal y como se observa a continuación:

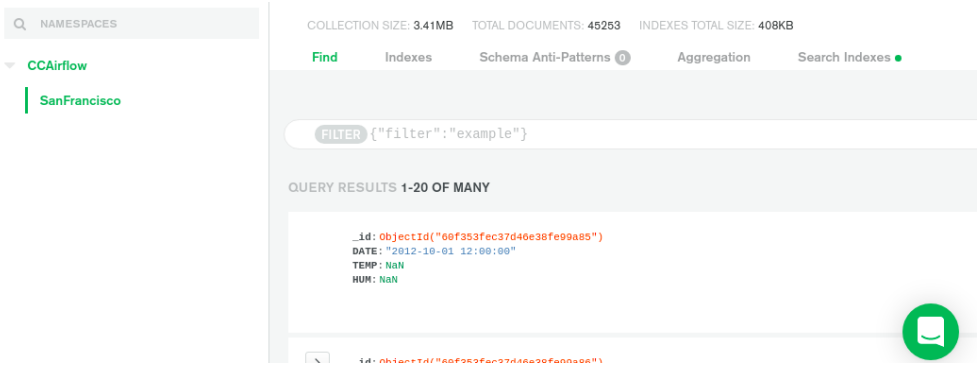

Imagen 5: Base de datos Mongo

3.5. Contenedores construidos y desplegados

Las últimas tareas del DAG son la construcción del contenedor y su posterior despliegue. Obsérvese la construcción con la orden `docker images`:

```
carlos@carlos-Aspire-VN7-571G:~$ docker images
REPOSITORY    TAG           IMAGE ID       CREATED          SIZE
carlossamu7   CC2_P2_API2   6be54c639303   3 minutes ago    139MB
carlossamu7   CC2_P2_API1   59830e4057e2   8 minutes ago    1.37GB
python        3.8-slim      0e0d73ddd34d   2 weeks ago      114MB
```

Imagen 6: Construcción del contenedor

Tras el despliegue podemos ver con `docker ps` que están levantados:

```
carlos@carlos-Aspire-VN7-571G:~$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS                                       NAMES
c6aa3682e3d8   carlossamu7:CC2_P2_API2   "/bin/sh -c 'gunicor…"   3 minutes ago   Up 3 minutes   0.0.0.0:8001->8001/tcp, :::8001->8001/tcp   intelligent_payne
4587e8c0a310   carlossamu7:CC2_P2_API1   "/bin/sh -c 'gunicor…"   8 minutes ago   Up 8 minutes   0.0.0.0:8000->8000/tcp, :::8000->8000/tcp   interesting_wilbur
```

Imagen 7: Desplegando los contenedores

4. Conclusiones

La profundidad de esta práctica es bastante alta. Por un lado se usa un sistema de flujo de trabajo como un servicio Cloud Native. Asimismo se desarrollan dos servicios para dar solución de formas distintas a un sistema de predicción de temperatura y humedad.

Para ello hay que manejar bien `dataframes` y hacer uso de `JSONs`. Asimismo en el camino es necesario utilizar una base de datos como ha sido Mongo DB. El aprendizaje no acaba aquí ya que cada uno de los servicios ha de estar testeado para ser levantado. Los tests no sólo son importantes para comprobar que aquello que se levanta es correcto y funciona sino que también suelen asegurar la calidad del código (en este caso son tests que sólo comprueban el estado de la petición).

De este modo es un proceso muy completo en donde se usan muchas de las nociones importantes en Cloud Computing. Personalmente creo que he aprendido mucho y me ha gustado `Airflow` por lo que no me importaría usarlo en el futuro.