**Versión Castellano**



Cloud Computing

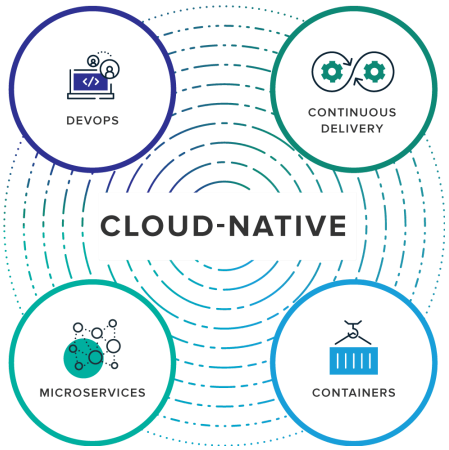
# Despliegue de un servicio Cloud Native

Práctica Evaluable 1

# Introducción

Cloud Native es un enfoque para la construcción y ejecución de aplicaciones o servicios que explota las ventajas del modelo de Cloud Computing. Cloud Native se refiere más concretamente a sobre cómo se crean y despliegan las aplicaciones, no dónde. Lo más importante es la capacidad de ofrecer una potencia de computación casi ilimitada, bajo demanda, junto con los modernos servicios de datos y aplicaciones para los desarrolladores. Cuando las empresas construyen y operan aplicaciones de forma nativa en Cloud Computing, desarrollannuevos productos más rápidamente y responden antes a las demandas de los clientes.

De este modo con Cloud Native las organizaciones tienen  una plataforma para construir y operar aplicaciones y servicios nativos de la nube que automaticen e integren los conceptos de DevOps, entrega continua, microservicios y contenedores.




Los aspectos clave para Cloud Native son los siguientes:

- **DevOps** es la colaboración entre los desarrolladores de software y las operaciones de TI con el objetivo de entregar constantemente software. Es donde se realiza la construcción, prueba y liberación de software con frecuencia y de manera consistente.

- **CI/CD (Entrega Continua)**, enfocada en prácticas rápidas de desarrollo de productos, consiste en el envío constante de pequeños trozos de software a fases de producción, a través de la automatización.
- **Microservicios** es un enfoque arquitectónico para desarrollar una aplicación como un conjunto de pequeños servicios; cada servicio implementa las capacidades empresariales, se ejecuta en su propio proceso y se comunica a través de las API HTTP por ejemplo. Cada microservicio puede desplegarse, actualizarse, escalarse y reiniciarse independientemente de otros servicios de la aplicación.
- Los **Contenedores** ofrecen tanto eficiencia como velocidad en comparación con las máquinas virtuales estándar (VM). El bajo coste general de creación y destrucción de contenedores combinados con la alta densidad de empaquetado en una sola máquina virtual hacen que los contenedores sean un vehículo informático ideal para desplegar microservicios.

Para la gestión de cada una de estas partes y la orquestación de la totalidad de las mismas es necesario una herramienta para controlar un flujo de trabajo en Cloud Computing a través del cual crear una un servicio de una forma ágil, controlada, versionada, desplegada y con un soporte de escalado flexible y adaptable a la demanda del mismo.

En esta práctica se ponen en valor todos estos conceptos para llevar a cabo un despliegue completo de un servicio listo para ser Cloud Native.

# Objetivos de la práctica

- Conocer que la filosofía Cloud Native
- Saber modelar un flujo de trabajo completo para un servicio Cloud Native.
- Desplegar microservicios escalables.
- Realizar una entrega continua de software.
- Utilizar todas las herramientas disponibles para DevOps.
- Diseñar un flujo de trabajo que permita realizar todo el ciclo completo de despliegue.
- Uso y gestión de git y de la plataforma GitHub

# Descripción de la práctica

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.
Para esta API final se debe tener como mínimo las siguientes operaciones, separadas en dos versiones, 1 y 2:

**Versión 1 del código del servicio (irá para CI/CD en contenedor)**

```
HTTP GET
     EndPoint 1  → /servicio/v1/prediccion/24horas/
HTTP GET
     EndPoint 2  → /servicio/v1/prediccion/48horas/
HTTP GET
     EndPoint 3  → /servicio/v1/prediccion/72horas/
```

**Versión 2 del código del servicio (irá para CI/CD en contenedor)**

```
HTTP GET
     EndPoint 4  → /servicio/v2/prediccion/24horas/
HTTP GET
     EndPoint 5  → /servicio/v2/prediccion/48horas/
HTTP GET
     EndPoint 6  → /servicio/v2/prediccion/72horas/
```

La funcionalidad de cada uno de los `EndPoints` anteriores es realizar una predicción de Humedad y Temperatura para 3 intervalos: 24, 48 y 72 horas. Esta funcionalidad se ha implementado como una herramienta (o función en Python) para predecir en función de varios parámetros de entrada y está disponible en el link del sitio web de la práctica para temperatura[1] como para humedad[2] para la versión 1, donde se usa ARIMA para predecir ambas para las 24 horas siguientes. Para la versión 2 habrá que usar otros modelos de predicción, por ejemplo usando otro algoritmo como redes neuronales, random forest, etc. (aquí hay más información[3]).

El interfaz de entrada y salida de esta herramienta para predecir **recibe** 2 parámetros:

**a)** el conjunto de datos, y
**b)** el valor de las horas `(dataset, hours)`

y **devuelve** un conjunto de predicciones `JSON` con los datos siguientes:

[Predicciones cada hora: 13:00 14:00, …. En lugar de cada 5 mins.]

```
[
 {"hour":13:05,"temp":32.20,"hum":85.90},
 {"hour":13:10,"temp":31.20,"hum":86.30},
 …
]
```

Estos datos devueltos por la función de predicción son los que hay que enviar desde la API y los EndPoints al usuario.

Esta API debe ser implementada sobre un servidor web sencillo que permita atienda a las llamadas anteriores. Como bibliotecas o plataformas para montar las APIs podéis usar cualquier que conozcais. Algunas de las recomendadas son:

1. `Flask (python)`[4]
2. `Flask-Restplus (python)`[5]
3. `Falsy (python)`[6]
4. `Bottle (python)`[7]
5. `Django (python)`
6. `PHP`
7. `node.js (javascript)`

[1] https://github.com/manuparra/MaterialCC2020/blob/master/exampleARIMA_temperature.py

[2] https://github.com/manuparra/MaterialCC2020/blob/master/exampleARIMA_humidity.py

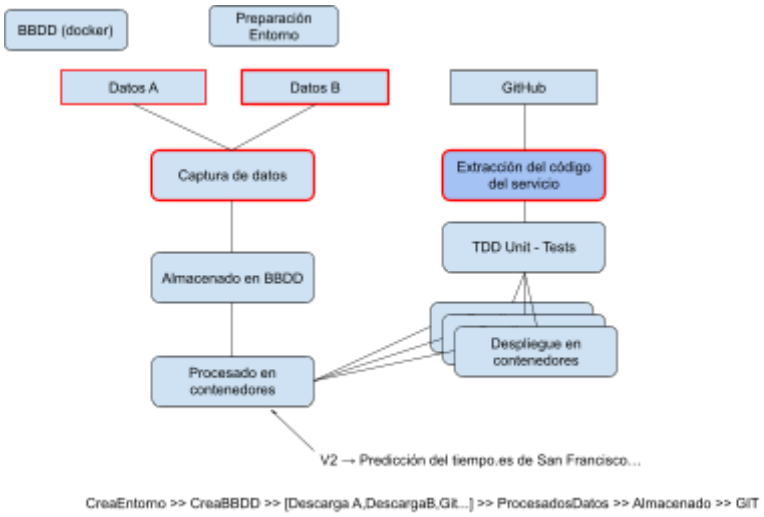
[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[4] https://palletsprojects.com/p/flask/
[5] https://flask-restplus.readthedocs.io/en/stable/
[6] https://falconframework.org/
[7] https://bottlepy.org/docs/dev/

8. ...

**Componentes del flujo de trabajo para la entrega y despliegue del software sobre AirFlow:**

Para diseñar todo el flujo de trabajo del servicio Cloud Native para predicción, mostramos cuál sería el esquema de operaciones/tareas que debes implementar como un Grafo Dirigido Acíclico (DAG[8]) en AirFlow:




Detalles de cada uno de los componentes del flujo de trabajo:

- **Datos A y Datos B**
  - Son dos fuentes de datos que en la entrega del sistema se capturan desde una URL y se almacenan en el sistema (en una BBDD a posteriori). Estos datos (ficheros) se descargan desde las siguientes URLs (se puede seleccionar una muestra de menor número de registros):
    - Datos A: https://github.com/manuparra/MaterialCC2020/blob/master/humidity.csv.zip
    - Datos B: https://github.com/manuparra/MaterialCC2020/blob/master/temperature.csv.zip

- Captura de datos y fusión:
  - Estos ficheros de datos anteriores deberán ser **descomprimidos, unificados y limpiados para admitir <u>SÓLO</u> la columna correspondiente a la fecha (date) y la ciudad concreta SAN FRANCISCO,** tanto para la humedad como para la temperatura, de modo que en la etapa de almacenado de datos se siga este formato:

[8]Más información sobre DAG en el tutorial de AirFlow

```
DATE;TEMP;HUM
```

    - Para esta tarea se puede realizar un subflujo de trabajo que haga lo siguiente:
        - Para cada dataset, extraer la columna fecha
            - Seleccionar la ciudad San Francisco para el dataset [humedad,temperatura]
            - Correlar el tiempo, para que igual que aparece en humedad aparezca en temperatura y componer un nuevo dataset con las columnas DATE;TEMP,HUM que sea la fusión de los dos ficheros por fecha para la ciudad de San Francisco.
- Almacenamiento en BBDD:
    - Una vez obtenido el fichero de datos con el procesado hecho, ahora hay que almacenarlo en una Base de Datos. Para ello, deberás lanzar un contenedor e inyectarle los datos en tiempo de creación. Puedes usar cualquier base de datos, aunque recomendamos MariaDB o MongoDB.
- Procesado en contenedores:
    - En esta etapa se generan los microservicios que hacen la predicción de temperatura y humedad, tal y como se indica en la descripción de la práctica, para el servicio de Versión 1 (ver código en https://github.com/manuparra/MaterialCC2020).
    - Hay que desarrollar dos versiones de servicios:
        - La versión 1, donde puedes usar este código de ejemplo https://github.com/manuparra/MaterialCC2020 para predecir los diferentes intervalos que se os piden usando ARIMA[9]
        - La versión 2 que es una modificación del anterior código pero donde debes utilizar otro algoritmo en lugar de ARIMA
        - Opcionalmente puedes usar la predicción de la página del eltiempo.es o cualquier otra fuente que tenga esos datos en predicción, por ejemplo las siguientes APIs:
            - https://openweathermap.org/api
            - https://algorithmia.com/
            - https://www.weatherapi.com/
    - El código de los contenedores y del servicio que ejecutará las APIs debe estar en GitHub para descargarse el código fuente en las otras fases que explicamos a continuación.
- Extracción del código fuente de los servicios:
    - Esta tarea descarga la última versión del código fuente de vuestros microservicios donde están las APIS v1 y v2, etc.
    - Para ello usad un repositorio en GitHub.
- TDD unit test:
    - En esta parte debéis realizar una batería de pruebas con TDD, o pruebas de unidad. Para ello en vuestro código tenéis que verificar al menos que una de las APIs tiene un conjunto de pruebas de unidad. Para saber sobre las pruebas de unidad: https://docs.python.org/3/library/unittest.html

[9] https://es.wikipedia.org/wiki/Modelo_autorregresivo_integrado_de_media_m%C3%B3vil

- Si alguna prueba de unidad no resulta correcta TODO el proceso de flujo de trabajo dará error.

- Despliegue en contenedores:
  - Si todas las tareas anteriores se han hecho con éxito, entonces se despliegan los contenedores para proveer al sistema de la API v1 y v2 como servicio.

# Entrega y evaluación de la práctica

## Entrega

Para la entrega de esta práctica:

1. **Se subirá la documentación a PRADO en la actividad "Práctica 2 AirFlow" en un único fichero PDF con el nombre "Práctica 2 AirFlow.pdf".**
   a. Esta documentación debe constar de las siguientes partes:
      i. Portada, con autor, DNI (ID), curso, asignatura.
      ii. Introducción / Descripción de la práctica.
      iii. Resolución de cada una de las tareas (sin codigo - Link -> repo):
         1. Para ello no copies el código, incluye el link a donde tienes cada función o servicio en GitHub.
         2. Explica cada una de las tareas, como la has resuelto y que problemas has tenido.
            a. Detalla los subpasos que has tenido en cuenta en cada tarea
            b. Mostrar el grafo (Airflow webserver)
         3. Conclusiones.
2. **Se realizará una defensa de la práctica:**
   a. En la defensa se verificará que todo el flujo de trabajo funciona y realiza todas la tareas como se piden.
   b. Se realizarán preguntas sobre las decisiones que se han tomado en cada tarea.
   c. Se preguntará sobre aspectos de escalado, flexibilidad y servicios desplegados.

## Fecha de entrega y defensa

Fecha límite de entrega de la documentación: hasta el 16 de mayo.
Fecha de defensa: será establecida después de la entrega.