Ejercicio

FUNDAMENTOS DE INTERNET DE LA COSAS

2023-2024

Universidad Carlos III de Madrid

Cuaderno 2- Ejercicio - 9

2024/2025

MQTT - ENVIO DE COMANDOS

Cuaderno 2 - Ejercicio 9

Universidad Carlos III de Madrid. Escuela Politécnica Superior

Javier García Guzmán

Sección 1

Objetivos

El propósito de este ejercicio consiste en:

- Afianzar los conceptos de publicación y subscripción que se manejan en el protocolo MQTT.
- Afianzar los conceptos de coreografía de servicios basados en docker compose.

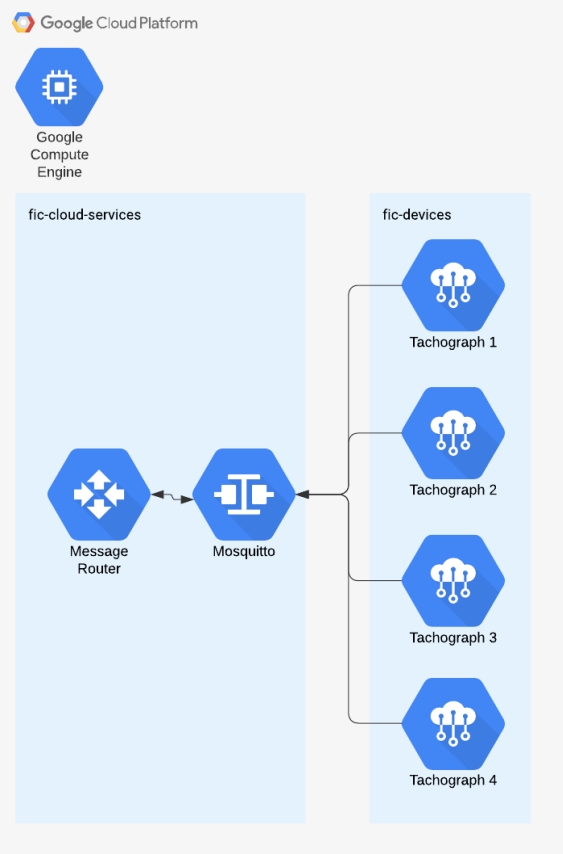
Sección 2

Introducción y pasos iniciales

Arquitectura y organización del proyecto

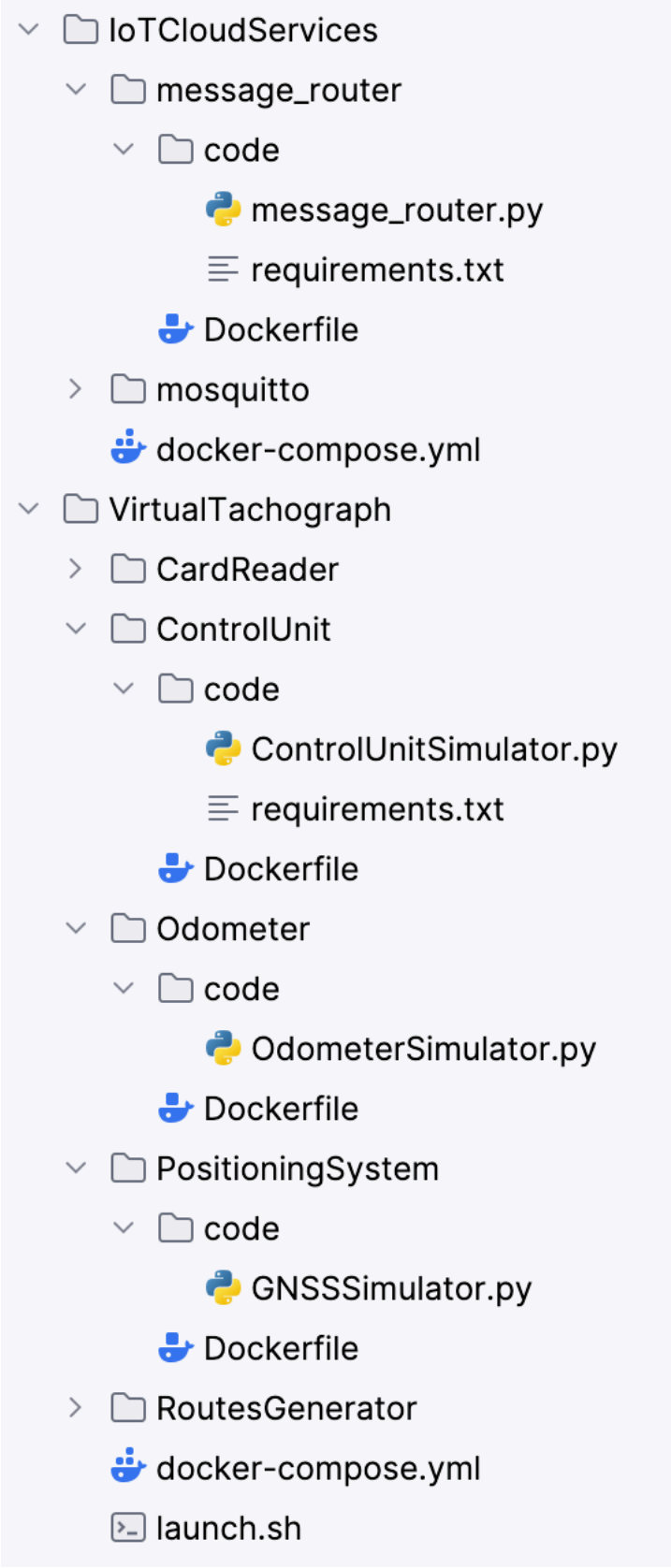
En el ámbito de este proyecto se partirá de la solución obtenida en la sesión 8.

La arquitectura del proyecto es la siguiente:



La arquitectura es la misma que la empleada en el ejercicio 8 porque el propósito de esta sesión consiste en ser capaces de enviar comandos desde el Message Router a los distintos vehículos.

La organización del código y los ficheros del proyecto será la siguiente:



El código se estructura en dos carpetas: *IoTCloudServices* su código se ejecutará en la máquina *fic-cloud-services*) y *VirtualVehicles* (su código se ejecutará en la máquina *fic-devices*)

- En *IoTCloudServices* habrá dos carpetas, una denominada *mosquitto* que contendrá el Dockerfile y los ficheros de configuración necesarios para el despligue de esta solución MQTT de código libre en un contenedor.

  La segunda carpeta se denominará *message_router* y contendrá el código fuente en Python para la implementación de este servicio (este código se colocará en la subcarpeta *code* donde también existirá un fichero requirements.txt con las dependencias necesarias para el funcionamiento del código desarrollado). Además, esta carpeta tiene el fichero Dockerfile para el despliegue mediante un contenedor del componente Message Router.

  Por último, en la carpeta *IoTCloudServices* se incluirá el fichero *docker-compose.yml* con la orquestación de los contenedores que se despliegan en la máquina *fic-cloud-services*.

- En *VirtualTachograph* habrá una carpeta *ControlUnit* y contendrá el código fuente en Python para la implementación de este servicio (este código se colocará en la subcarpeta *code* donde también existirá un fichero requirements.txt con las dependencias necesarias para el funcionamiento del código desarrollado). Además, esta carpeta tiene el fichero Dockerfile para el despliegue mediante un contenedor de cada gemelo digital.

  Asimismo, en esta carpeta, se modificará en el ámbito de este proyecto, el funcionamiento de los Componentes *Odometer* y *PositioningSystem*

  Por último, en la carpeta *VirtualVehicles* se incluirá el fichero *docker-compose.yml* con la orquestación de todos los gemelos digitales a contemplar en la máquina *fic-devices*.

Preparación del proyecto

En primer lugar, es necesario clonar el proyecto en gitlab (https://teaching.sel.inf.uc3m.es) correspondiente a la sesión 9.

Se recomienda que, en este momento, se copien todos los ficheros de código y configuración desarrollados en la sesión 8 a la carpeta en la que se está desarrollando el código de la sesión 9, contemplando la estructura de carpetas presentada en el apartado anterior.

Sección 3

Actualización del servicio Message Router

Para desarrollar el servicio denominado Message Router será necesario, en primer lugar, desarrollar y probar el código de su funcionamiento en el entorno personal de desarrollo del estudiante y, posteriormente, utilizar el fichero *Dockerfile* y *docker-compose.yml* desarrollado en el ejercicio anterior.

Desarrollo y prueba local del código de Message Router

En la carpeta *IoTCloudServices/message_router/code* se tiene que actualizar el código incluido en el fichero denominado *message_router.py*.

Los elementos de código que se deben incluir en la versión actualizada de Message Router son los siguientes:

1. Se generará un valor decimal aleatorio entre 0 y 2 que representará la frecuencia que se solicita al tacógrafo para que su Odómetro y Sensor GNSS simulados envíen su información a la unidad de control.
2. Se generará un valor decimal aleatorio entre 5 y 60 que representará la frecuencia que se solicita al tacógrafo para que su Unidad de Control realice el envío de telemetría al Message Router.
3. Estos valores se actualizarán para cada tacógrafo conectado al Message Router en una frecuencia variable que puede variar entre 10 y 240 segundos.
4. Estos valores se enviarán al tacógrafo correspondiente utilizando un topic similar al siguiente:

```
"/fic/tachographs/" + tachograph_hostname + "/config/"
```

5. Un ejemplo de la sentencia de publicación de la configuración del tacógrafo puede ser similar a la siguiente:

```
message = {"Tachograph_id": tachograph_id, "Config_item":
config_item, "Config_Value": config_value,
"Timestamp":
datetime.datetime.timestamp(datetime.datetime.now())*1000}
mqtt_client.publish(topic,payload=json.dumps(message), qos=1,
retain=False)
```

Creación de la imagen con Dockerfile

El fichero Dockerfile que se utilizará para el Message Router en este ejercicio es igual al utilizado en el ejercicio anterior.

Orquestación de los servicios con Docker Compose

El fichero *docker-compose.yml* que se utilizará para la máquina *fic-cloud-services* en este ejercicio es igual al utilizado en el ejercicio anterior.

Despliegue de la imagen con Docker Compose

Para desplegar el componente message_router es necesario conectarse por *ssh* a la máquina *fic-cloud-services* creada en la sección 2 de este enunciado.

A continuación, se debe obtener la última versión del código de esta sesión que se encuentra en el repositorio en esta máquina virtual mediante un comando *git pull*.

Una vez que se haya obtenido en la máquina virtual el código correspondiente a esta sesión, hay que desplazarse a la carpeta *IoTCloudServices* y lanzar los servicios de esta máquina ejecutando los comandos:

```
docker compose stop

docker compose build

docker compose up -d
```

Para verificar que el Message Router se encuentra en ejecución, se pueden ejecutar los comandos siguientes:

```
docker ps -a

docker logs <nombre del contenedor del message router>
```

Sección 5

Actualización del gemelo digital del simulador del vehículo

Para incluir las capacidades de recepción de los comandos de configuración establecidos por el Message Router será necesario actualizar el código del gemelo digital del tacógrafo, posteriormente, será necesario realizar modificaciones en relación con el fichero *Dockerfile* y se tendrá que actualizar la coreografía de servicios para la máquina *fic-devices* modificando el fichero *docker-compose.yml* que permitirá su despliegue.

Actualización y prueba local del código del gemelo digital del simulador del vehículo

En la carpeta *VirtualTachograph/ControlUnit/code* se tiene que actualizar el fichero de código Python que se denomine *ControlUnitSimulator.py*.

Las modificaciones que incluir es la siguiente:

1. Cuando el gemelo digital reciba un mensaje por el topic de configuración, realizar los cambios de configuración:

```
if json_config_received["Tachograph_id"] == tachograph_id and json_config_received["Config_item"] is not None:
    if json_config_received["Config_item"]=="telemetry_frequency":
        upgrade_telemetry_publication_frequency(json_config_received["Config_value"])
    else:
        if json_config_received["Config_item"]=="sensors_frequency":
            upgrade_sensors_sampling_frequency(json_config_received["Config_value"])
```

   - De la frecuencia de envío de la telemetría. Esto consiste en actualizar el valor de la variable global que controla la frecuencia de envío de telemetría desde la unidad de control al Message Router.
   - De la frecuencia de muestreo del odómetro y del sensor de GNSS

     Se debe implementar un método que envíe, a través del socket que gestiona la comunicación con el Odómetro y el sensor GNSS, un mensaje con el valor modificado de la frecuencia de muestreo.

Actualización de la imagen con Dockerfile

El fichero Dockerfile que se utilizará para el Gemelo Digital del Tacógrafo en este ejercicio es igual al utilizado en el ejercicio anterior.

Configuración de la orquestación del Gemelo Digital

El fichero *docker-compose.yml* que se utilizará para la máquina *fic-devices* en este ejercicio es igual al utilizado en el ejercicio anterior.

Despliegue del Gemelo Digital

Para desplegar las versiones actualizadas del gemelo digital del simulador de vehículos en un contenedor Docker, se recomienda utilizar Docker Compose porque permitirá orquestar este servicio con los otros que se van a ir desplegando en la máquina virtual *fic-devices*.

Posteriormente, es necesario conectarse por *ssh* a la máquina *fic-devices* creada en el ejercicio guiado correspondiente a la sesión 8. A continuación, se debe clonar el repositorio correspondiente al código de esta sesión en esta máquina virtual.

Una vez que se haya obtenido en la máquina virtual el código correspondiente a esta sesión, hay que desplazarse a la carpeta *VirtualTachograph* y lanzar los servicios de esta máquina ejecutando los comandos:

```
docker compose build

docker compose up -d
```

Para verificar el despliegue, se pueden ejecutar los comandos siguientes:

```
docker ps -a

docker logs <nombre del contenedor>
```

Sección 6

Criterios de evaluación y procedimiento de entrega

Criterios de Evaluación

- Message Router – Envío de configuraciones – 5 puntos
- Gemelo Digital – Recepción de configuraciones asignación de la ruta – 3 puntos
- Gemelo Digital – Actualización de frecuencia de muestreo de odómetro y sensor GNSS – 2 punto

Entrega de la solución del ejercicio guiado

La entrega se realizará de dos maneras:

1) **Código Fuente**. En el repositorio de código de la asignatura tendrá en el proyecto Sesion09 correspondiente al grupo de prácticas los ficheros de código con la organización en directorios y nomenclatura de los ficheros que se han indicado a lo largo de este guion.
2) **Video demostrativo.** En una tarea Kaltura Capture Video habilitada para este ejercicio se entregará un vídeo que demuestre el correcto funcionamiento de la solución elaborada.

   Para este programa, se debe proporcionar una breve explicación del código generado (código en Python, dockerfile y docker compose) para Mosquitto, Message Router y Gemelo Digital, mostrar el lanzamiento de la ejecución de la solución en la GCP y la visualización de las trazas con los resultados.

La fecha límite para la entrega del ejercicio es 08/05/2025 a las 23:59 h.

De acuerdo con las normas de evaluación continua en esta asignatura, si un estudiante no envía la solución de un ejercicio antes de la fecha límite, el ejercicio será evaluado con 0 puntos.

Cada estudiante debe guardar una copia de la solución entregada hasta la publicación de las calificaciones finales de la asignatura.