



FUNDAMENTOS DE INTERNET DE LA COSAS

2024-2025

Universidad Carlos III de Madrid

Cuaderno 2- Ejercicio - 7

2024/2025

INTRODUCCIÓN A CONTENEDORES Y GEMELO
DIGITAL

Cuaderno 2 - Ejercicio 7

Universidad Carlos III de Madrid. Escuela Politécnica Superior

Sección

1

Objetivos

El propósito de este ejercicio consiste en:

- Introducir las capacidades de la Google Cloud Platform que estará accesible para el cuaderno 2 de la asignatura y canjear los cupones para su uso.
- Poner en práctica los conceptos básicos para la creación de una imagen de contenedor y su lanzamiento.
- Desarrollar un gemelo digital de un tacógrafo.
- Utilizar la API de Google Maps para calcular la ruta entre dos puntos y trocearla en segmentos de longitud corta para la generación aleatoria de información de velocidad y posición del vehículo según simula desplazarse.

Introducción a Google Cloud Platform y canjeo de los cupones de prácticas

Introducción a Google Cloud Platform

Imagina que eres un arquitecto de software para el portal foo.com, una aplicación accesible en Internet. Hay muchas maneras diferentes de diseñar una aplicación de este tipo. Se va a utilizar este ejemplo para ver cuáles son las capacidades que proporciona la Google Cloud Platform (GCP) para que cuando un usuario abre el navegador y escribe foo.com en la barra de direcciones tenga el comportamiento de cualquier página web.

Para servidores web y de aplicaciones tienes múltiples opciones en Cloud Run, App Engine, GKE y Compute Engine. Echa un vistazo a dónde debería ejecutar mis cosas para más detalles.

- **Serverless:** Si tienes un equipo de desarrolladores, quieres que se centren en la codificación y no se preocupen por la infraestructura y las tareas de escalado. Cloud Run o App Engine serían buenas opciones. Ambos son sin servidor y escalan de bajo a alto tráfico según sea necesario. Si desea ejecutar contenedores sin servidor que sirvan arquitecturas de microservicios web y basadas en eventos, se recomienda Cloud Run. Cloud Run debería funcionar para la mayoría de los casos de uso, echa un vistazo a App Engine si estás desarrollando sitios web con alojamiento de archivos estáticos integrado.
- **Google Kubernetes Engine (GKE):** Si desea ejecutar aplicaciones en contenedores con más opciones de configuración y flexibilidad, puede utilizar GKE. Le ayuda a desplegar fácilmente aplicaciones en contenedores con Kubernetes a la vez que le ofrece control sobre la configuración de los nodos. El escalado también es sencillo; puede definir el número de nodos a escalar a medida que crece el tráfico. GKE también ofrece piloto automático, cuando se necesita flexibilidad y control pero el soporte de operaciones e ingeniería es limitado.

- Compute Engine: Se trata directamente de máquinas virtuales (VM), por lo que puede definir con precisión la configuración de sus máquinas en función de la cantidad de memoria y CPU que necesite. Sin embargo, este nivel de control implica una mayor responsabilidad a la hora de escalar, gestionar, parchear y mantener las máquinas virtuales según sea necesario. Compute Engine funciona bien para aplicaciones heredadas con necesidades específicas y en situaciones que realmente requieren un control total.

Por supuesto, foo.com necesita una o varias bases de datos para almacenar la información. Pueden ser bases de datos relacionales o no relacionales, dependiendo del tipo de datos y del caso de uso. (Para obtener información más detallada sobre cómo elegir la base de datos adecuada para tu caso de uso, consulta [Explicación de las opciones de bases de datos de Google Cloud](#)).

Las bases de datos relacionales de Google Cloud incluyen Cloud SQL y Cloud Spanner, ambas gestionadas.

- Cloud SQL es perfecto para necesidades SQL genéricas - MySQL, PostgreSQL y SQL server.
- Spanner es mejor para bases de datos relacionales de escala masiva que necesitan escalabilidad horizontal. (Masiva aquí significa miles de escrituras por segundo y decenas de miles de lecturas por segundo, a la vez que soporta transacciones ACID).
- Para bases de datos no relacionales, Google Cloud tiene tres opciones principales: Firestore, Bigtable y Memorystore.
 - Firestore es una base de datos de documentos sin servidor que proporciona una gran coherencia, admite transacciones ACID y ofrece resultados rápidos a consultas complejas. También admite datos y sincronizaciones sin conexión, lo que la convierte en una gran opción para casos de uso móvil junto con web, IoT y juegos.
 - Bigtable es una base de datos NoSQL de columnas anchas que admite lecturas y escrituras pesadas con una latencia extremadamente baja. Esto la convierte en la elección perfecta para eventos, datos de series temporales de dispositivos IoT, datos de flujo de clics, eventos publicitarios, detección de fraudes, recomendaciones y otros casos de uso relacionados con la personalización.
 - Memorystore es un servicio de almacenamiento de datos en memoria totalmente gestionado para Redis y Memcached. Es ideal para almacenes transitorios y cachés de bases de datos.

A medida que crezca el tráfico, será necesario escalar con él los servidores web y de aplicaciones. Y, a medida que crezca el número de servidores, necesitará un equilibrador de carga para dirigir el tráfico a los servidores web y de aplicaciones. Cloud Load Balancing es un sistema totalmente distribuido y definido por software basado en direcciones IP anycast, lo que significa que puede configurar su frontend con una única dirección IP. También es global, por lo que puede servir contenidos lo más cerca posible de sus usuarios y responder a más de un millón de consultas por segundo. Puede configurar decisiones de enrutamiento basadas en el contenido y en atributos, como el encabezado HTTP y el identificador uniforme de recursos. También ofrece equilibrio de carga interno para los servidores de aplicaciones internos, de modo que puede enrutar el tráfico entre ellos según sea necesario.

Todos los archivos estáticos de foo.es, como archivos multimedia e imágenes, así como CSS y JavaScript, pueden almacenarse en un almacén de objetos. En Google Cloud, Cloud Storage es su almacén de objetos para las necesidades de almacenamiento a largo y corto plazo.

Digamos que foo.com también está disponible en dispositivos móviles, que necesitan imágenes renderizadas en formatos móviles más pequeños. Puede desacoplar esta funcionalidad del servidor web y convertirla en una función como servicio con Cloud Functions. Este enfoque le permite aplicar su lógica de redimensionamiento de imágenes también a otras aplicaciones. Puede activar la función sin servidor tan pronto como se añada un archivo a Cloud Storage y convertir el archivo en múltiples formatos, almacenándolos de nuevo en el almacenamiento, donde son utilizados por el servidor web. También podría utilizar funciones sin servidor para otros casos de uso, como búsquedas de direcciones, chatbots, aprendizaje automático, etc.

Las aplicaciones como foo.com generan datos en tiempo real (por ejemplo, datos de clics) y datos por lotes (por ejemplo, registros). Estos datos deben ser ingeridos, procesados y preparados para los sistemas posteriores en un almacén de datos. A partir de ahí, los analistas de datos, los científicos de datos y los ingenieros de ML pueden seguir analizándolos para obtener información y realizar predicciones. Puede ingerir datos por lotes desde Cloud Storage o BigQuery y datos en tiempo real desde la aplicación mediante Pub/Sub, y escalar hasta ingerir millones de eventos por segundo. Dataflow, basado en Apache Beam de código abierto, puede utilizarse para procesar y enriquecer los datos por lotes y de flujo. Si pertenece al ecosistema Hadoop, puede utilizar Dataproc para el procesamiento; se trata de una plataforma Hadoop y Spark gestionada que le permite centrarse en el análisis en lugar de preocuparse por la gestión y puesta en marcha de su clúster Hadoop.

Para almacenar los datos procesados se necesita un almacén de datos. BigQuery es un almacén de datos sin servidor que admite consultas SQL y puede escalar hasta petabytes de almacenamiento. También puede actuar como almacenamiento a largo plazo y un lago de datos junto con Cloud Storage. Puede utilizar los datos de BigQuery para crear un cuadro de mando en Looker y Data Studio. Con BigQuery ML puedes crear modelos ML y hacer predicciones utilizando consultas SQL estándar.

Para proyectos ML/AI puedes usar los datos en BigQuery para entrenar modelos en Vertex AI. Sus medios de comunicación, imágenes y otros conjuntos de datos de archivos estáticos de almacenamiento en la nube se pueden importar directamente en Vertex AI. Puede crear su propio modelo personalizado o utilizar los modelos preentrenados. Es una buena idea empezar con un modelo pre-entrenado, y ver si funciona para usted. Se cubren los casos de uso más comunes (incluyendo imagen, texto, vídeo y datos tabulares). Si un modelo preentrenado no funciona para su caso de uso, utilice el modelo AutoML en Vertex AI para entrenar un modelo personalizado en su propio conjunto de datos. AutoML admite todos los casos de uso comunes y no requiere código. En caso de que tenga mucha experiencia en ML y ciencia de datos, entonces puede decidir escribir su propio código de modelo personalizado en el marco de su elección.

También tienes que asegurarte de que los equipos de desarrollo y operaciones de foo.com tienen el acceso y las herramientas adecuadas para crear la aplicación y desplegarla. A medida que los desarrolladores escriben el código de la aplicación, pueden utilizar Cloud Code en el IDE para enviar el código a Cloud Build, que lo empaquetará y probará, ejecutará análisis de vulnerabilidades en el código, invocará Binary Authorization para comprobar si hay imágenes de contenedor de confianza y, una vez superadas las pruebas, desplegará el paquete en la fase de montaje. Desde allí se puede crear un proceso para revisar y promover a producción. Las imágenes de contenedor se almacenan en el Registro de artefactos, desde donde pueden desplegarse en GKE o Cloud Run. Las imágenes de Compute Engine se almacenan en su proyecto.

foo.com debe protegerse a nivel de datos, aplicación, usuario/identidad, infraestructura y cumplimiento.

Canjeo de los cupones de prácticas

Para que puedas utilizar los recursos en la Nube de Google Cloud Platform para tus prácticas en alguna asignatura con tu cuenta UC3M, tus profesores han solicitado códigos promocionales, que otorgan 50 USD por alumno y asignatura.

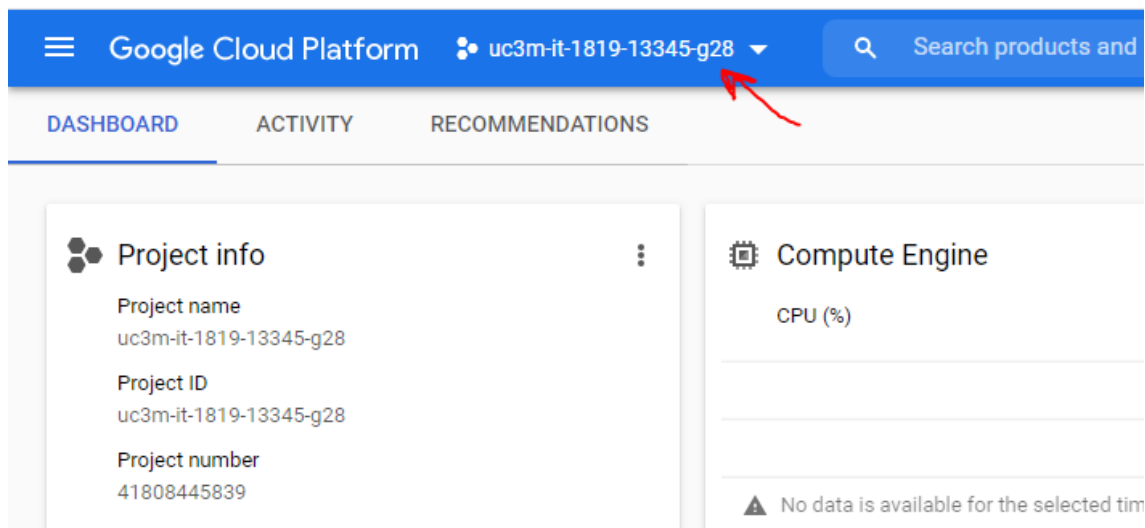
Con estos códigos, no es necesario que introduzcas una tarjeta de crédito, y en este documento te mostramos cómo canjearlos y cómo asignarlo al proyecto que ya debes tener creado usando tu cuenta de Alumno UC3M.

Entra en la consola de Google Cloud Platform con la dirección:

<https://console.cloud.google.com>

Es MUY IMPORTANTE que inicies con tu cuenta de Alumno UC3M.

Dependiendo de si has entrado antes por otras asignaturas o si es la primera vez que entras, tendrás una pantalla diferente. Lo importante es que mires los proyectos que tienes. Para ello, usaremos el selector de proyectos que tenemos en la parte superior superior de la consola:



Del listado de proyectos, tienes que identificar el que se corresponde con tu asignatura. Los profesores te indicarán cual es el código de los proyectos para la asignatura, pero todos siguen el siguiente patrón:

uc3m-<dpto>-<año>-<asig>-g<pract>

Donde:

<dpto> es el código del departamento

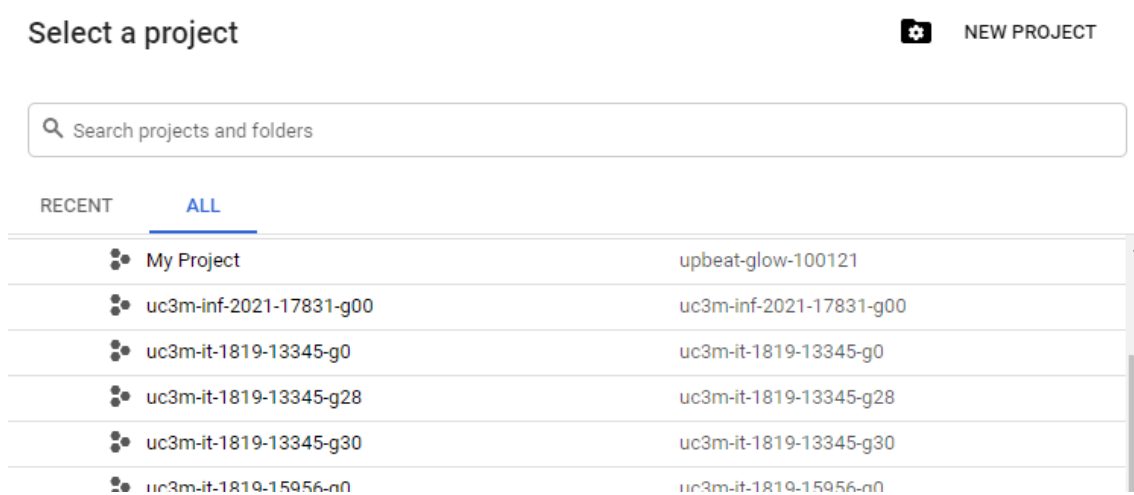
<año> es el año docente de la asignatura

<asig> es el código uc3m para la asignatura

<pract> es el número asignado a tu grupo de prácticas

El Servicio de Informática y Comunicaciones gestiona la plataforma de Google Cloud para toda la comunidad Universitaria. Desde SDIC, se crean los proyectos y se asignan a los alumnos para realizar las prácticas.

Durante un periodo de tiempo acordado con los profesores de la asignatura, vuestras cuentas de alumno podrán crear una **Billing Account**, con el crédito promocional que se os asigna. **Una vez terminado el periodo de canje de cupones, no podrás crear cuentas de facturación, ni canjear cupones** y tendrás que contactar con tus profesores para resolverlo.



Para canjear tu cupón de acceso a la Google Cloud Platform (GCP) recibirá correo electrónico, un mensaje o anuncio en Aula Global por parte de tus profesores con el enlace para solicitar tu cupón. El proceso es el siguiente:

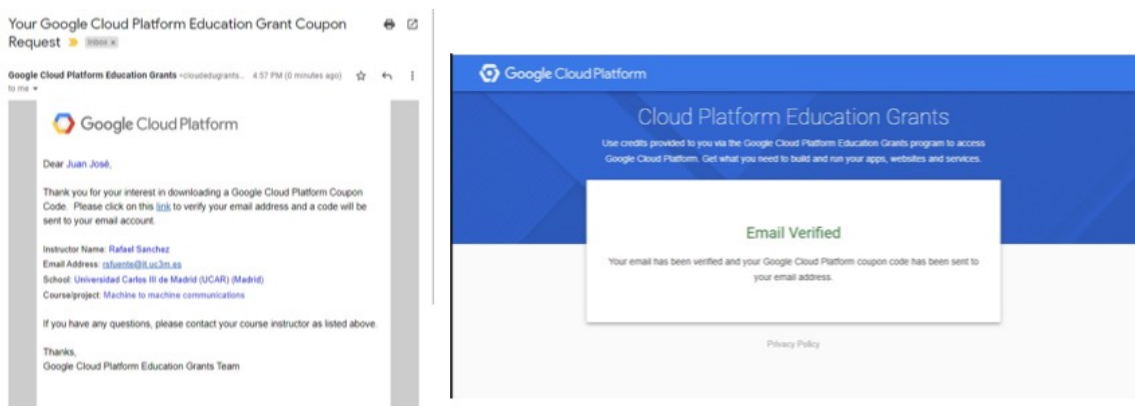
1. Confirmar tus datos (verificar el correo electrónico).

The screenshot shows the Google Cloud Platform Education Grants registration page. At the top, the Google Cloud Platform logo is visible. Below it, the heading "Cloud Platform Education Grants" is displayed. A sub-header states: "Use credits provided to you via the Google Cloud Platform Education Grants program to access Google Cloud Platform. Get what you need to build and run your apps, websites and services." The main content area is a white box with a blue border. It contains a thank-you message: "Thank you for your interest in Google Cloud Platform Education Grants. Please fill out the form below to receive a coupon code for credit to use on Google Cloud Platform." Below this, there are three input fields: "Nombre" (Name), "Apellidos" (Surnames), and "School Email". The "School Email" field has a dropdown menu showing "@alumnos.uc3m.es". Below the email field, there is a note: "If you do not see your domain listed, please contact your course instructor: [celeste@it.uc3m.es](\"mailto:celeste@it.uc3m.es\")". A paragraph of terms and conditions follows, stating that by clicking "Submit", the user agrees to share their information with their educational institution and course instructor. At the bottom of the form is a blue button labeled "Enviar" (Send). Below the form box, there is a link to the "Privacy Policy".

Es imprescindible que utilices la cuenta que finaliza en @alumnos.uc3m.es. En caso de que no dispongas de una cuenta de este tipo, por favor, ponte en contacto con tu profesor de prácticas para que se resuelva tu incidencia.

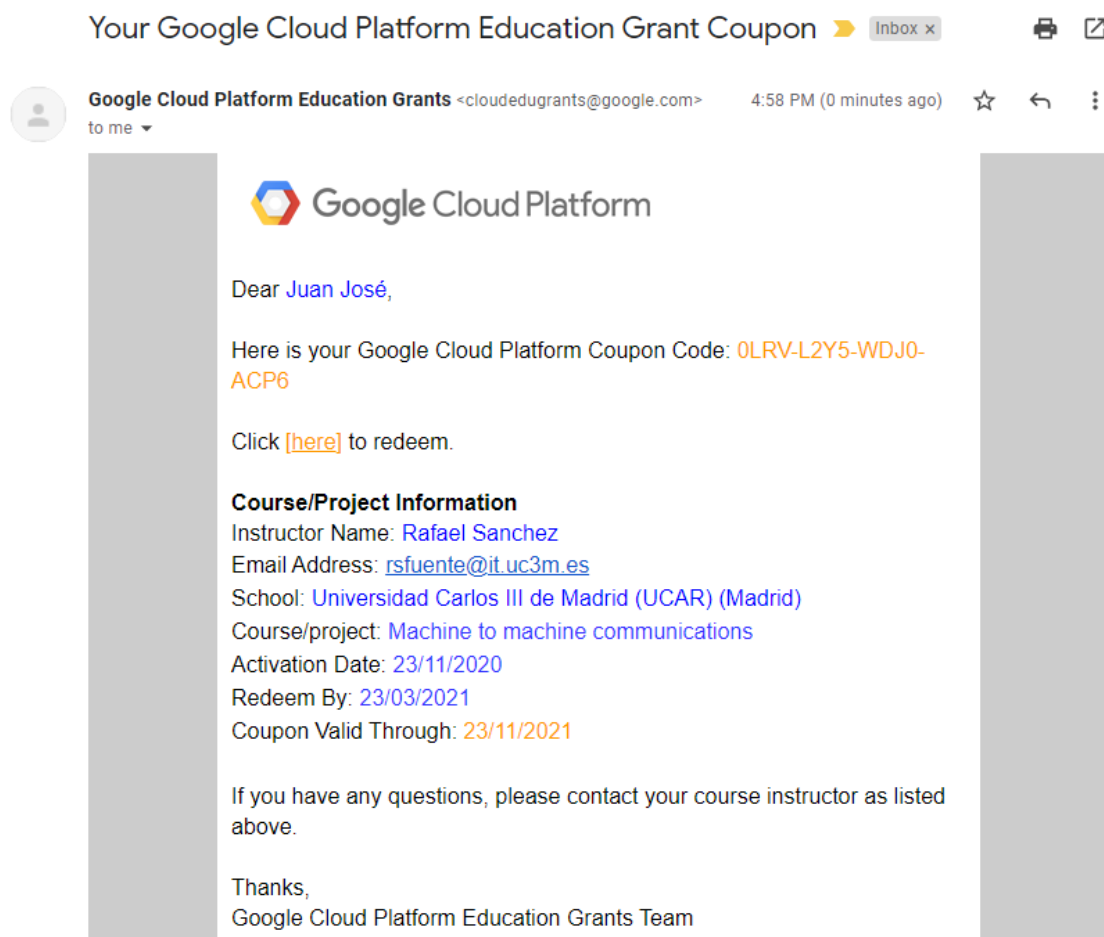
Una vez que se ha completado el paso anterior, recibirá un correo en tu cuenta de Alumno agradeciendo que solicites un cupón para Google Cloud Platform, y solicitando que pinches en un enlace para verificar tu cuenta de correo.

En este momento, confirme los datos y verifica tu correo electrónico para continuar con el proceso.



2. Recibir el correo electrónico de Google Cloud Platform con el código para canjear tu cupón.



Una vez verificada tu dirección de correo, recibirás un correo electrónico en tu cuenta de alumno parecido a este, en el que te dan los datos de la asignatura y el enlace para poder recibir tu cupón:



Pinchamos en el enlace donde pone [here] y nos lleva a GCP con una previsualización del cupón.

3. Confirmar los datos y crear la cuenta de facturación

En la página a la que te envía, te explican que es una Beca educativa, el código de cupón ya introducido, (en algunas ocasiones puede aparecer vacío, si es el caso, introduce el código que te ha llegado al correo) la cantidad de dinero y las condiciones de uso del servicio:

 Google Cloud Platform  Buscar productos y recursos

Becas educativas

Introduce el código de cupón que te enviamos a través del programa de becas educativas de Google Cloud Platform para recibir crédito de dicha plataforma. Consigue todo lo que necesitas para desarrollar y ejecutar tus aplicaciones, sitios web y servicios.

Código de cupón

OLRV-L2Y5-WDJ0-ACP6

Importe del crédito	Fecha de caducidad	Curso
50,00 \$	22 nov. 2021	Machine to machine communications

Términos del Servicio

País de residencia

España

Quiero recibir periódicamente correos electrónicos con noticias, novedades de productos y ofertas especiales de Google Cloud y de los Google Cloud Partners.

☒ Sí ☐ No

Google Cloud Platform education grants credits terms and conditions

By clicking "Accept and continue" below, you, on behalf of yourself and the organization you represent ("You") agree to these terms and conditions:

The credit is valid for Google Cloud Platform products and is subject to Your acceptance of the applicable Google Cloud Platform License Agreement and any other applicable terms of service. The credit is non-transferable and may not be sold or bartered. Unused credit expires on the date indicated on the media conveying the promotion code. The credit may be issued in increments as You use the credit over the period of time during which the credit is valid. Offer void where prohibited by law.

You represent that you are accepting the promotional credit on behalf of your educational institution and the credit can only be used on behalf of the educational entity and not for your personal use. You represent, on behalf of such educational entity, that (i) You are authorized to accept this credit; (ii) the credit is consistent with all applicable laws and regulations, including relevant ethics rules and laws; and (iii) the provision of credits will not negatively impact Google's current or future ability to do business with such educational entity.

You agree that we may share the following information with your educational institution and course instructor: (1) personal information that you provide to us during the coupon redemption process and (2) information regarding your use of the coupon and Google Cloud Platform products.

Aceptar y continuar


Borrar

Pincha en Aceptar y continuar. Al hacerlo, se creará la cuenta de facturación con el crédito otorgado.

- Asignar la cuenta de facturación al proyecto correspondiente a tus prácticas de la asignatura.

Lo siguiente es, pinchar en el menú lateral, en la opción Facturación. Te mostrará todas las cuentas de facturación a las que tienes alcance. En este caso, existirá una con el nombre de la asignatura, Billing for Education Credits, o un nombre similar.

Es importante que esté seleccionada la organización “uc3m.es”.



Google Cloud Platform

Facturación

Selecciona una organización: UC3M.ES

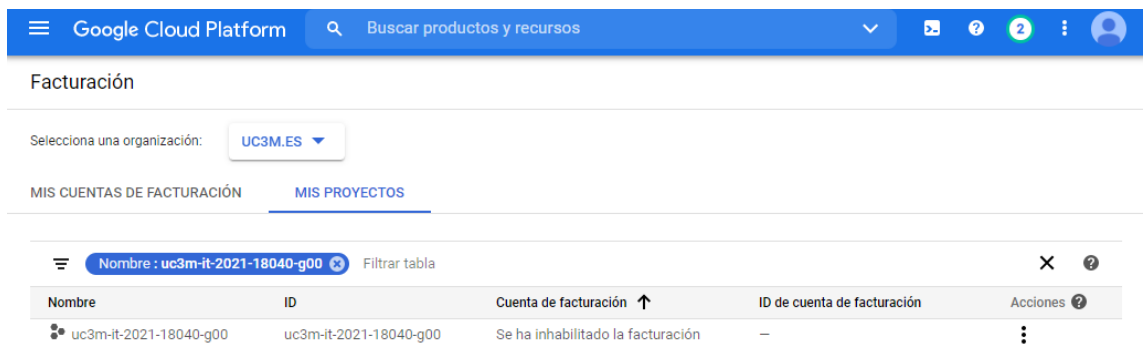
MIS CUENTAS DE FACTURACIÓN MIS PROYECTOS

CREAR CUENTA

Estado: Activa Nombre de la cuenta de facturación: Machine to machine communications Filtrar tabla

<input type="checkbox"/>	Nombre de la cuenta de facturación ↑	ID de cuenta de facturación	Estado	N.º de proyectos
<input type="checkbox"/>	Machine to machine communications	01FC64-69EDCC-2EF291	Activa	0

Si pinchas en **Mis Proyectos**, se mostrarán todos los proyectos en los que tienes permisos para operar:



Google Cloud Platform

Facturación

Selecciona una organización: UC3M.ES

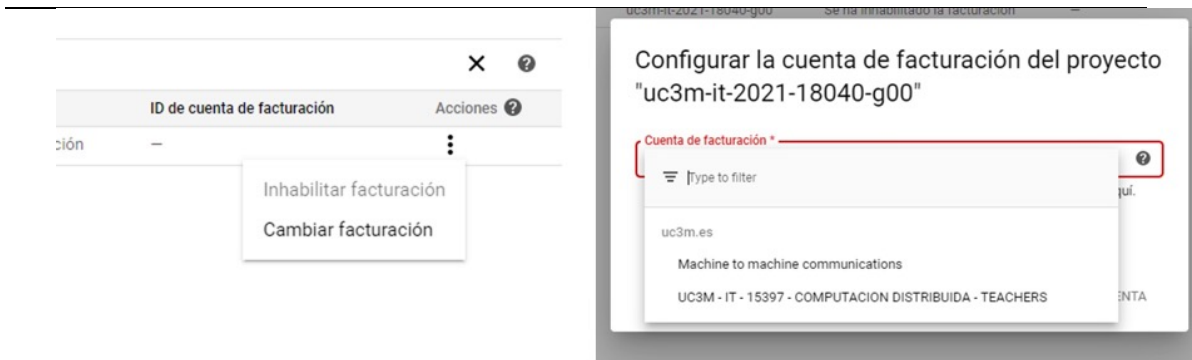
MIS CUENTAS DE FACTURACIÓN MIS PROYECTOS

Nombre: uc3m-it-2021-18040-g00 Filtrar tabla

Nombre	ID	Cuenta de facturación ↑	ID de cuenta de facturación	Acciones ?
uc3m-it-2021-18040-g00	uc3m-it-2021-18040-g00	Se ha inhabilitado la facturación	—	⋮

En ese listado de la izquierda debe aparecerte el proyecto asignado a tu grupo de prácticas. Lo siguiente es irse a la columna de Acciones (los 3 puntos verticales) y pinchar en **Cambiar facturación**.

Aparecerá una figura como la de la derecha, allí debes elegir, del listado de cuentas de facturación, la que se llama como el nombre de la asignatura.



Creación de una máquina virtual

1. En la consola de Google Cloud, ve a la página **Instancias de VM**.
2. Selecciona el proyecto y haz clic en **Continuar**.
3. Haz clic en **Crear instancia**.
4. Especifica un **Nombre** para la VM. Se recomienda poner el siguiente nombre: *fic-devices*.
5. Opcional: Cambia la **Zona** para esta VM. Compute Engine aleatoriza la lista de zonas dentro de cada región para fomentar el uso en varias zonas.
6. Selecciona una **Configuración de máquina** para la VM.
7. En la sección **Disco de arranque**, haz clic en **Cambiar** y, luego, haz lo siguiente:
 - a) En la pestaña **Imágenes públicas**, elige lo siguiente:
 - Sistema operativo (Seleccionar el SO que viene por defecto: Debian)
 - Versión del SO
 - Tipo de disco de arranque
 - Tamaño de disco de arranque
 - b) Opcional: Para ver las opciones de configuración avanzadas, haz clic en **Mostrar configuración avanzada**.
 - c) Para confirmar las opciones del disco de arranque, haz clic en **Seleccionar**.
8. Para crear y, también, iniciar la VM, haz clic en **Crear**.

Obtención de la API KEY de Google Maps

Para usar Google Maps Platform, debes habilitar las APIs o los SDKs que planeas usar con tu proyecto:

https://console.cloud.google.com/apis/library/roads.googleapis.com?utm_source=Docs_EnableAPIs&utm_content=Docs_roads&hl=es

INTRODUCCIÓN A CONTENEDORES Y GEMELO DIGITAL

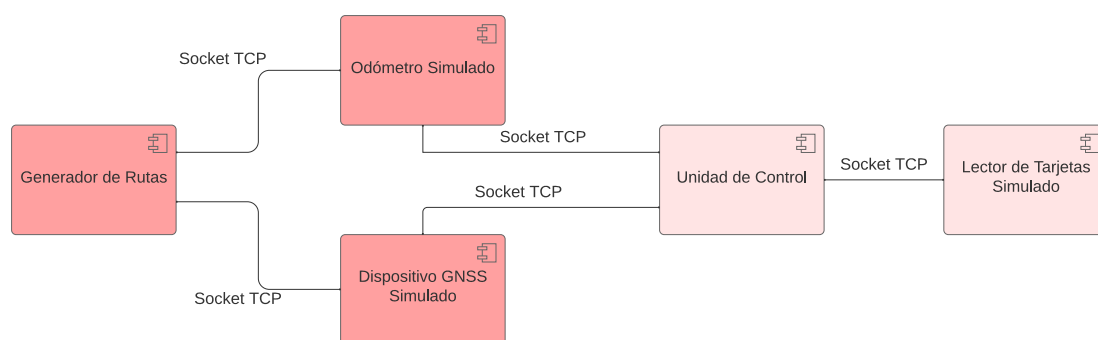
419&_gl=1*1tyrdk1*_ga*MTU2OTg0NDA0MS4xNjkyNjQzNTgy*_ga_NRWSTWS78N*
MTcwMzU5NDA1Mi42LjEuMTcwMzU5NDEzNi4wLjAuMA..

Para generar la API KEY de Google Maps, ver el siguiente vídeo:

https://youtu.be/2_HZObVbe-g

Desarrollo del gemelo digital de un tacógrafo

El gemelo digital del tacógrafo se estructura en los componentes siguientes:



A) Una unidad de control que será responsable de recibir los datos de posición y velocidad instantánea recibida de un dispositivo GNSS simulado. También recibirá los datos de velocidad instantánea de un odómetro simulado. Con esta información, este componente proporcionará las funcionalidades siguientes:

- Registrará con una periodicidad prefijada los datos de posición y velocidad reportados por los sensores.
- Registrará distintos eventos que son necesarios para controlar la correcta conducción del vehículo monitorizado por el tacógrafo simulado:
 - Se generará un evento cuando un lector de tarjetas haya identificado que un nuevo conductor se ha conectado al dispositivo.
 - Se generará un evento cuando un lector de tarjetas haya identificado que un nuevo conductor se ha desconectado al dispositivo.
 - En caso de que se registre que el vehículo está en movimiento pero sin un conductor conectado al dispositivo, se registrará un evento informativo de esta situación.

- En caso de que el dispositivo haya registrado una velocidad superior a la autorizada para este tipo de vehículos (90 km/h), se registrará un evento informativo de esta situación.
- En caso de que se detecte una divergencia superior al 5% entre la velocidad registrada por el dispositivo GNSS y el odómetro, se registrará un evento informativo de esta situación.

Todos los eventos registrarán la marca de tiempo y la posición GNSS en que se producen.

- B) Un Lector de Tarjetas Simulado que genera eventos de la introducción y retirada de una lectura con la identificación del conductor que está realizando las tareas de conducción. Este evento se comunica a la unidad de control, a través de un socket TCP, utilizando un mensaje en formato JSON indicando si el evento ha sido inserción o retirada, cuál es el conductor implicado y la marca de tiempo correspondiente.
- C) Un Odómetro Simulado que genera valores de velocidad instantáneo en una frecuencia determinada (i.e.: 1 segundo). Estos valores se generan a partir del perfil de velocidades recibido del componente generador de rutas.

El perfil de velocidades es una secuencia de mensajes que tienen la siguiente estructura:

```
{"Speed": subStepSpeed, "Time": subStepDuration}
```

Estos mensajes se reciben del componente de generación de rutas mediante un socket TCP.

La generación aleatoria se realizará utilizando como referencia el valor anterior de velocidad (el valor proporcionado por el generador de ruta en caso de tratarse de un nuevo mensaje) al cual se incrementa o disminuye un valor aleatorio de 5 km/h.

El valor de velocidad instantánea generado se enviará a la unidad de control mediante un socket TCP. El texto del mensaje que se envíe estará en formato JSON y tendrá la siguiente estructura:

```
{"Type": "Odometer", "Speed": random_speed, "Timestamp": marca_de_tiempo}
```

- D) Un Sistema GNSS Simulado que genera valores de posición y velocidad instantáneos en una frecuencia predeterminada (i.e.: 1 segundo). Estos valores se estiman generando tantos puntos intermedios entre el origen y el destino recibidos como valores surjan de la división entre el tiempo que dura el segmento (`position["Time"]`) y la frecuencia configurada para la generación de los mensajes de posición.

El perfil de posiciones y velocidades recibido es una secuencia de mensajes que tienen la siguiente estructura:

```
{"Origin": p_inicial, "Destination": p2, "Speed": stepSpeed, "Time": subStepDuration}
```

Estos mensajes se reciben del componente de generación de rutas mediante un socket TCP.

La generación de valores de posición

El valor de velocidad instantánea generado se enviará a la unidad de control mediante un socket TCP. El texto del mensaje que se envíe estará en formato JSON y tendrá la siguiente estructura:

```
{"Type": "GPS", "Position": {"Latitude": valor, "Longitude": valor}, "Speed": valor, "Timestamp": marca de tiempo}
```

- E) Un generador de rutas que, a partir de dos ubicaciones que se proporcionan como entrada. A modo de ejemplo, la manera de especificar estos valores es la siguiente:

```
{"Origin": "Ayuntamiento de Leganés", "Destination": "Ayuntamiento de Getafe"}
```

A partir de esta información, se genera la ruta entre esos dos puntos utilizando la API Directions de Google Maps. Esa ruta se convertirá en:

1. Un perfil de posiciones por las que pasará el vehículo. Este perfil se enviará al componente Sistema GNSS Simulado mediante una secuencia de mensajes cada uno de ellos enviado de manera independiente mediante un socket TCP. Un ejemplo de los mensajes a generar es:

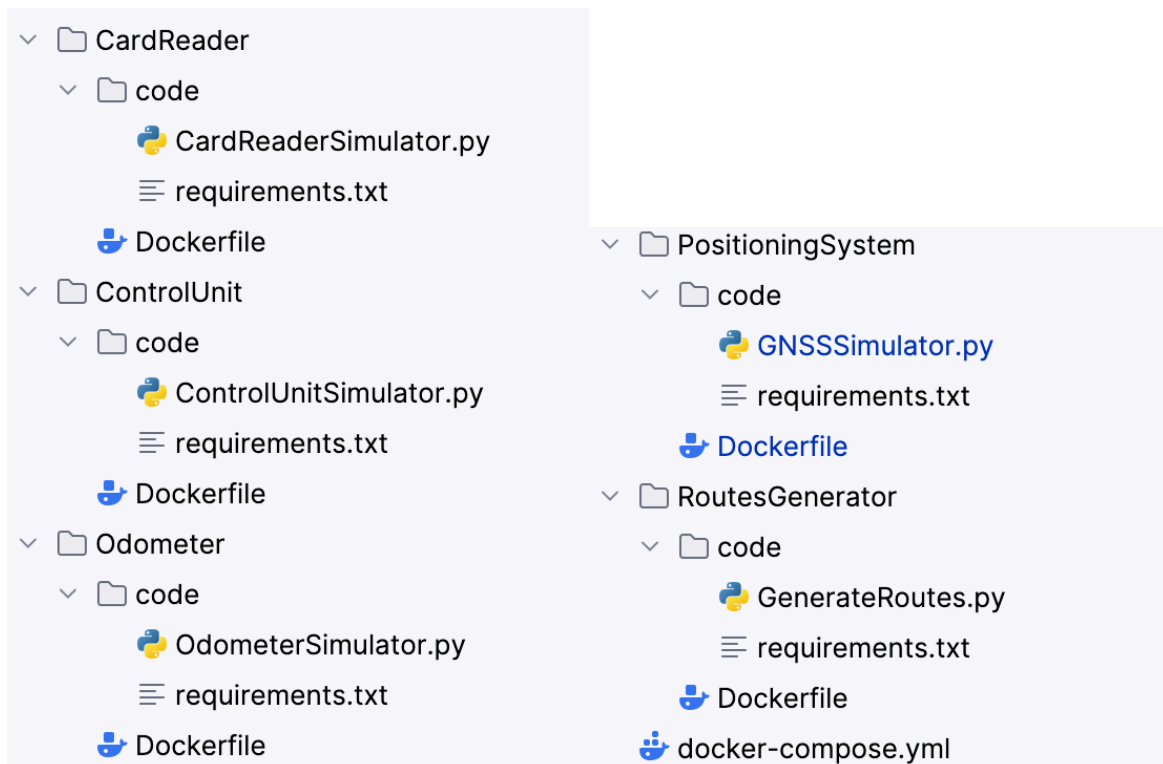
```
{"Origin": p_inicial, "Destination": p2, "Speed": stepSpeed, "Time": subStepDuration}
```

- Un perfil de velocidades que podría generar el vehículo. Este perfil se enviará al componente Odómetro Simulado mediante una secuencia de mensajes cada uno de ellos enviado de manera independiente mediante un socket TCP. Un ejemplo de los mensajes a generar es:

```
{"Speed": stepSpeed, "Time": subStepDuration}
```

Para el desarrollo de la solución propuesta, en primer lugar, es necesario clonar el proyecto en gitlab (<https://teaching.sel.inf.uc3m.es>) correspondiente a la sesión 01.

La estructura que se recomienda para el proyecto a desarrollar se presenta en la figura siguiente:



A continuación, se proporcionan algunas instrucciones para el desarrollo de los componentes considerados para simular el sistema de tacógrafo digital.

Unidad de Control

La estructura general que debe tener el código de la unidad de control debe ser similar a esta:

- El hilo principal de ejecución permite la recepción de los mensajes procedentes del Lector de Tarjetas, Sistema GNSS y Odómetros simulados. A continuación, se muestra un ejemplo del código que se podría utilizar como referencia para el desarrollo de esta funcionalidad.

```
if __name__ == '__main__':
    try:
        t1 = threading.Thread(target=data_logger, daemon=True)
        t1.start()
        HOST = get_host_name()
        PORT = int(os.getenv("PORT"))

        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
s:
            s.bind((HOST, PORT))
            s.listen(3)
            while True:
                print("{} - Waiting for
connection...".format(datetime.datetime.now()))
                connection, address = s.accept()
                threading.Thread(target=client_listener,
args=(connection, address)).start()
                t1.join()

    except Exception as e:
        print(e)
```

Los datos del HOST que gestiona los sockets de este componente se generarán utilizando la función siguiente:

```
def get_host_name():
    bashCommandName = 'echo $HOSTNAME'
    host = subprocess \
        .check_output(['bash', '-c', bashCommandName]) \
        .decode("utf-8") [0:-1]
    return host
```

Para ello, según se reciban peticiones de conexión, que están limitadas a un máximo de tres (que se corresponden con los sensores que pueden proporcionar información a la unidad de control), se generará un nuevo hilo que se encargue de gestionar los mensajes recibidos por ese hilo. A continuación, se muestra un ejemplo del código que se podría utilizar como referencia para el desarrollo de esta funcionalidad.

```
def client_listener(connection, address):
    print("{} - New connection {}".
format(datetime.datetime.now(), connection, address))
    while not monitor.kill_now:
        data = connection.recv(1024)
        if not data:
            break
        else:
            data = data.decode("utf-8")
            print("{} - He recibido el mensaje:
{}".format(datetime.datetime.now(), data))
            process_received_message(data)
```

```
connection.sendall(bytes("ok-" + str(time.time()),
"utf-8"))
```

A partir de la información recibida se generará un objeto en formato JSON con una estructura similar a esta:

```
current_state = {"Position": {"Latitude": 0.0, "Longitude": 0.0}, "GPSSpeed": 0.0, "Speed": 0.0,
"Driver": "None", "Timestamp": last_time}
```

A continuación, se muestra un ejemplo del código que se podría utilizar como referencia para la generación de este mensaje.

```
def process_received_message(data):
    global current_state
    global logs
    global state_changed
    data = json.loads(data)
    if data["Type"] == "GPS":
        current_state["Position"] = data["Position"]
        current_state["GPSSpeed"] = data["Speed"]
    else:
        if data["Type"] == "Odometer":
            current_state["Speed"] = data["Speed"]
        else:
            if data["Type"] == "CardReader":
                current_state["driver_present"] =
data["driver_present"]
            current_state["Timestamp"] = data["Timestamp"]
            logs.append(current_state)
```

Se almacenará este objeto JSON en el log de cambio de estados gestionados por la unidad central del tacógrafo

- El segundo hilo permite generar y almacenar los eventos relativos al funcionamiento.

A continuación, se muestra un ejemplo del código que se podrían generar los eventos contemplando las reglas establecidas en la especificación de la unidad de control proporcionada al comienzo de esta sección.

```
def data_logger():
    global last_time
    global state_changed
    while not monitor.kill_now:
        if current_state["Timestamp"] > last_time:
            if current_state["driver_present"] == "None" and
current_state["Speed"] > 0.0:
                state_changed = True
                generate_movement_without_driver_warning()
            if current_state["Speed"] > 80.0:
                state_changed = True
                generate_overspeed_warning()
            if current_state["Speed"] - current_state["GPSSpeed"] >
3.0:
```

```

        state_changed = True
        generate_speed_incoherence_warning()
        last_time =
datetime.datetime.timestamp(datetime.datetime.now())*1000
        time.sleep(1)

```

Lector de Tarjetas Simulado

La estructura general que debe tener el código del lector de tarjetas simulado debe ser similar a esta:

```

if __name__ == '__main__':
    try:
        simulate_current_driver()
    except Exception as e:
        print(e)

```

En el ámbito del método especificado se debe implementar el código necesario, que incluye lo siguiente:

- El simulador se conectará al socket gestionado por la unidad de control.

```

is_driver = 0
driver_present = 0
UC_SIMULATOR_HOST = os.getenv("UC_SIMULATOR_HOST") #
session01_Positioning_System_Simulator
UC_SIMULATOR_PORT = int(os.getenv("UC_SIMULATOR_PORT"))
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((UC_SIMULATOR_HOST, UC_SIMULATOR_PORT))
    while True:

```

- Se generarán los valores de lectura de tarjetas de acuerdo con la especificación de este componente proporcionada al comienzo de esta sección.

```

is_driver = math.trunc(random.uniform(0.5, 1.5))
if is_driver == 1:
    driver_present = math.trunc(random.uniform(1.5, 3.5))
    simulated_driver = {"Type": "CardReader", "is_driver":
is_driver, "driver_present": "Driver "
+ str(
        driver_present), "Timestamp":
datetime.datetime.timestamp(datetime.datetime.now()) * 1000}
else:
    simulated_driver = {"Type": "CardReader", "is_driver":
is_driver, "driver_present": "None",
        "Timestamp":
datetime.datetime.timestamp(datetime.datetime.now()) * 1000}

```

- Se enviarán los datos generados a través del socket gestionado

```

s.sendall(bytes(json.dumps(simulated_driver), "utf-8"))
print("{} - He enviado el mensaje:
{}".format(datetime.datetime.now(), simulated_driver))
data = s.recv(1024)
frequency = random.uniform(0.0, 60.0)
print("{} - Volveré a enviar mensaje en:

```

```
{}}".format(datetime.datetime.now(), frequency))
time.sleep(frequency)
```

Odómetro Simulado

La estructura general que debe tener el código del lector de tarjetas simulado debe ser similar a esta:

```
if __name__ == '__main__':
    try:
        t1 = threading.Thread(target=receive_speed_inputs,
                                daemon=True)
        t1.start()
        t2 = threading.Thread(target=simulate_current_speed,
                                daemon=True)
        t2.start()
        t1.join()
        t2.join()

    except Exception as e:
        print(e)
```

- El primer hilo permite la recepción de los mensajes procedentes del generador de rutas:
 - En primer lugar, se deberá crear un socket por el que se recibirá la información proporcionada por el generador de rutas.

```
global speed_inputs
HOST = get_host_name()
PORT = int(os.getenv("PORT"))

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
```

- Cuando haya un mensaje nuevo recibido desde el socket, se descodificará, se almacenará en una estructura que se utilizará para la generación de los valores aleatorios de velocidad y se confirmará la recepción del mensaje.

```
while True:
    data = conn.recv(1024)
    if not data:
        break
    else:
        data = data.decode("utf-8")
        print("{} - He recibido el mensaje:
        {}".format(datetime.datetime.now(), data))
        speed_inputs.append(json.loads(data))
        conn.sendall(bytes("ok-" + str(time.time()), "utf-8"))
```

- El segundo hilo permite la generación de valores aleatorios de velocidad a partir de los datos de referencia obtenidos del generador de rutas. Para ello, es necesario contemplar los pasos siguientes:

- El simulador se conectará al socket gestionado por la unidad de control de la misma manera que se ha especificado anteriormente para el caso del lector de tarjetas simulado.
- Se generarán los valores de velocidad instantánea de acuerdo con la especificación de proporcionada para este componente al comienzo de esta sección.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((UC_SIMULATOR_HOST, UC_SIMULATOR_PORT))
    while True:
        for speed in speed_inputs:
            times = math.trunc(speed["Time"] / frequency) + 1
            while times > 0:
                if times == math.trunc(speed["Time"] / frequency):
                    random_speed= speed["Speed"] + random.uniform(-
5.0, 5.0)
                else:
                    random_speed += random.uniform(-5.0, 5.0)
                simulated_speed = {"Type": "Odometer", "Speed":
random_speed,
                                "Timestamp":
datetime.datetime.timestamp(datetime.datetime.now())*1000}
                s.sendall(bytes(json.dumps(simulated_speed), "utf-8"))
                print("{} - He enviado el mensaje:
{}".format(datetime.datetime.now(), simulated_speed))
                data = s.recv(1024)
                time.sleep(frequency)
                times -= 1
```

Sistema GNSS Simulado

La estructura general que debe tener el código del lector de tarjetas simulado debe ser similar a esta:

```
if __name__ == '__main__':
    try:
        t1 = threading.Thread(target=receive_simulation_inputs,
daemon=True)
        t1.start()
        t2 = threading.Thread(target=simulate_positioning,
daemon=True)
        t2.start()
        t1.join()
        t2.join()

    except Exception as e:
        print(e)
```

- El primer hilo permite la recepción de los mensajes procedentes del generador de rutas:
 - En primer lugar, se deberá crear un socket por el que se recibirá la información proporcionada por el generador de rutas tal y como se explica para el Odómetro Simulado.

- Cuando haya un mensaje nuevo recibido desde el socket, se descodificará, se almacenará en una estructura que se utilizará para la generación de los valores aleatorios de velocidad y se confirmará la recepción del mensaje.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            else:
                data = data.decode("utf-8")
                print("{} - He recibido el mensaje:
{}").format(datetime.datetime.now(), data)
                simulation_inputs.append(json.loads(data))
                conn.sendall(bytes("ok-" + str(time.time()), "utf-
8"))
```

- El segundo hilo permite la generación de valores aleatorios de posición a partir de los datos de referencia obtenidos del generador de rutas. Para ello, es necesario contemplar los pasos siguientes:
 - El simulador se conectará al socket gestionado por la unidad de control de la misma manera que se ha especificado anteriormente para el caso del lector de tarjetas simulado.
 - Se generarán los valores de velocidad instantánea de acuerdo con la especificación de este componente proporcionada al comienzo de esta sección.

```
for position in simulation_inputs:
    times = math.trunc(position["Time"] / frequency) + 1
    while times-1 > 0:
        simulated_position = {"Type": "GPS", "Position":
position["Origin"], "Speed": position["Speed"],
                             "Timestamp":
datetime.datetime.timestamp(datetime.datetime.now())*1000}
        s.sendall(bytes(json.dumps(simulated_position), "utf-
8"))
        print("{} - He enviado el mensaje:
{}").format(datetime.datetime.now(), simulated_position)
        print("{} - Volveré a enviar mensaje en: {}
segundos".format(datetime.datetime.now(), frequency))
        data = s.recv(1024)
        time.sleep(frequency)
        times -= 1
        last_position = position["Destination"]
        simulated_position = {"Type": "GPS", "Position":
last_position, "Speed": position["Speed"],
                             "Timestamp":
datetime.datetime.timestamp(datetime.datetime.now())*1000}
        s.sendall(bytes(json.dumps(simulated_position), "utf-8"))
        data = s.recv(1024)
```

```

print("{} - He enviado el mensaje:
{}".format(datetime.datetime.now(), simulated_position))
print("{} - Volveré a enviar mensaje en: {}
segundos".format(datetime.datetime.now(), frequency))
time.sleep(frequency)

```

Generador de Rutas

Este componente funciona generando el perfil de velocidades y posiciones de una ruta entre dos puntos.

La estructura general que debe tener el código de la unidad de control debe ser similar a esta:

```

if __name__ == '__main__':
    try:
        my_route = {"Origin": "Ayuntamiento de
Leganés", "Destination": "Ayuntamiento de Getafe"}
        positions_to_simulate, speeds_to_simulate =
generate_route_simulations(my_route["Origin"],
my_route["Destination"])
        t2 = threading.Thread(target=send_positions_to_gps_simulator,
args=(positions_to_simulate,), daemon=True)
        t2.start()
        t3 =
threading.Thread(target=send_speeds_to_odometer_simulator,
args=(speeds_to_simulate,), daemon=True)
        t3.start()
        t2.join()
        t3.join()

    except Exception as e:
        print(e)

```

a) El método generate routes simulation tiene como propósito generar las secuencias de mensajes de perfiles de velocidad y posición que se deben enviar al Simulador del Odómetro y del Sistema GNSS.

```

def generate_route_simulations(origin_address="Toronto",
destination_address="Montreal"):
    print("Asignando una ruta al vehículo")
    my_body = { "origin":{"address": origin_address},
                "destination":{"address": destination_address},
                "travelMode": "DRIVE",
                "languageCode": "es-ES",
                "units": "METRIC"
    }
    my_headers = {'Content-Type': 'application/json',
                  'X-Goog-API-Key': 'E',
                  'X-Goog-FieldMask':
'routes.duration, routes.legs'}

    api_url =
"https://routes.googleapis.com/directions/v2:computeRoutes"
    response = requests.post(api_url, json = my_body,
                             headers=my_headers)
    current_route = response.json()
    #print("La ruta es: {}".format(response.text))

```

```

steps = response.json()["routes"][0]["legs"][0]["steps"]
positions_to_simulate, speeds_to_simulate =
generate_positions_speeds(steps)
# print("He acabado de generar las posiciones y funciones a
simular")
return positions_to_simulate, speeds_to_simulate

```

La API Key de Google Maps se ha obtenido en el ámbito de la ejecución de los pasos de la sección 2 de este ejercicio.

La ruta se calcula ejecutando la correspondiente llamada a la API de Google. Los pasos que componen una ruta se obtiene del resultado de la misma en caso de que no haya habido ningún error con la petición realizada. Los pasos que proporciona la API de Google son los correspondiente a la primera ruta que devuelve como resultado que se obtiene de la siguiente manera:

```

steps = response.json()["routes"][0]["legs"][0]["steps"]

```

A partir de esta información se generará el perfil de velocidades y posiciones que se enviarán a los simuladores para la generación de los datos aleatorios correspondientes.

Para ello, el método *generate_positions_speeds* debe obtener los waypoints que debe atravesar el vehículo con el mayor nivel de precisión posible-

1.1. En el método *generate_positions_speeds*, para cada uno de los pasos proporcionados por Google Maps, se debe:

- Determinar la distancia del paso.

```

stepDistance = step["distanceMeters"]

```

- Determinar el tiempo del paso.

```

strStepTime = step["staticDuration"]

```

- Determinar la velocidad.

```

stepSpeed = stepDistance / stepTime

```

- Determinar los waypoints que se corresponden con ese paso. Para ello se realizará la llamada al método *decode_polyline*.

```

substeps =
decode_polyline(step["polyline"]["encodedPolyline"])

```

Se recomienda copiar/pegar el código completo de este método, que es el siguiente:

```

def decode_polyline(polyline_str):
    '''Pass a Google Maps encoded polyline string; returns list
    of lat/lon pairs'''
    index, lat, lng = 0, 0, 0
    coordinates = []

```

```

changes = {'latitude': 0, 'longitude': 0}

# Coordinates have variable length when encoded, so just
keep
# track of whether we've hit the end of the string. In each
# while loop iteration, a single coordinate is decoded.
while index < len(polyline_str):
    # Gather lat/lon changes, store them in a dictionary to
    apply them later
    for unit in ['latitude', 'longitude']:
        shift, result = 0, 0

        while True:
            byte = ord(polyline_str[index]) - 63
            index += 1
            result |= (byte & 0x1f) << shift
            shift += 5
            if not byte >= 0x20:
                break

        if (result & 1):
            changes[unit] = ~(result >> 1)
        else:
            changes[unit] = (result >> 1)

    lat += changes['latitude']
    lng += changes['longitude']

    coordinates.append((lat / 1000000.0, lng / 1000000.0))

return coordinates

```

Para cada uno de los waypoints generados por este método, se tiene que indicar su origen (p1), su destino (p2), la distancia entre ambos, la duración en segundos, la velocidad (que es la del paso del que procede el waypoint) y la maniobra (que es la del paso del que procede el waypoint) y se tiene que añadir a la lista de pasos detallados que tiene que completar el simulador del vehículo.

```

for substep in substeps:
    if index < len(substeps):
        if p_inicial == 0.0:
            p_inicial = {"latitude": substeps[index][0],
"longitude": substeps[index][1]}
        p2 = {"latitude": substeps[index + 1][0], "longitude":
substeps[index + 1][1]}
        points_distance = distance(p_inicial, p2) * 1000
        if points_distance > 1:
            subStepDuration = points_distance / stepSpeed
            subStepSpeed = stepSpeed * 3.6
            new_position = {"Origin": p_inicial, "Destination":
p2, "Speed": subStepSpeed, "Time": subStepDuration}
            positions_to_simulate.append(new_position)
            new_speed = {"Speed": subStepSpeed, "Time":
subStepDuration}
            speeds_to_simulate.append(new_speed)
            p_inicial = 0.0

```

El método que calcula la distancia entre dos puntos es el siguiente:

```
def distance(p1, p2):
    p1Latitude = p1["latitude"]
    p1Longitude = p1["longitude"]
    p2Latitude = p2["latitude"]
    p2Longitude = p2["longitude"]
    # print("Calculando la distancia entre ({} , {}) y
    ({} , {})".format(p1["latitude"], p1["longitude"], p2["latitude"],
    p2["longitude"]))
    earth_radius = {"km": 6371.0087714, "mile": 3959}
    result = earth_radius["km"] * acos(
        cos((radians(p1Latitude))) * cos(radians(p2Latitude)) *
        cos(radians(p2Longitude) - radians(p1Longitude)) + sin(
            radians(p1Latitude)) * sin(radians(p2Latitude)))
    # print ("La distancia calculada es:{}".format(result))
    return result
```

b) El primer hilo de ejecución permite el envío de los datos de posición generados al simulador del sistema GNSS. Para ello, es necesario contemplar los pasos siguientes:

La estructura general que debe tener el código debe ser similar a esta:

```
SPEED_SIMULATOR_HOST = os.getenv("SPEED_SIMULATOR_HOST") #
session01_Positioning_System_Simulator
SPEED_SIMULATOR_PORT = int(os.getenv("SPEED_SIMULATOR_PORT"))

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((SPEED_SIMULATOR_HOST, SPEED_SIMULATOR_PORT))
    for speed in speeds_to_simulate:
        s.sendall(bytes(json.dumps(speed), "utf-8"))
        data = s.recv(1024)
        # print(f"Received {data!r}")
        print("{} - He enviado el mensaje:
        {}".format(datetime.datetime.now(), json.dumps(speed)))
```

b) El segundo hilo de ejecución permite el envío de los datos de posición generados al simulador del Odómetro.

El código a desarrollar para este hilo es similar al proporcionado para el hilo de ejecución anterior.

Se recomienda probar el gemelo digital del tacógrafo en el entorno de desarrollo local del estudiante antes de pasar a su containerización y despliegue. De esta manera, se optimizará la detección de los posibles fallos que se puedan introducir en los distintos elementos del código.

Una vez finalizado este paso, no olvidar de publicar todo el código generado en el proyecto del Gitlab correspondiente a esta sesión.

Conceptos básicos para la creación de imágenes y lanzamiento de contenedores

Una vez que se ha probado que el código del gemelo digital funciona correctamente en el equipo de desarrollo, se procederá a la preparación para su despliegue con contenedores.

Creación de imágenes con Dockerfile

Cada uno de los componentes considerados (CardReader, ControlUnit, Odometer, PositioningSystem y RoutesGenerator) debe ser empaquetado en un contenedor.

Para ello, en las carpetas en las que se ha almacenado el código de cada uno de esos componentes se creará un fichero *Dockerfile*, para cada uno de los componentes considerados.

El contenido de este fichero tendrá que permitir lo siguiente:

- Crear la imagen del contenedor a partir de la correspondiente a python:3.12.1-alpine
- Copiar el código que está en la carpeta `./code` a la carpeta `/etc/usr/src/app`
- Establecer esta carpeta como directorio de trabajo.
- Instalar los paquetes necesarios especificados en `requirements.txt`. Los paquetes necesarios son: *requests* y *math*, para ello ejecutar `pip install <paquetes necesarios>`
- Ejecutar el código incluido en el fichero de código correspondiente en cada caso (CardReaderSimulator.py, ControlUnitSimulator.py, OdometerSimulator.py, GNSSSimulator.py o GenerateRoutes.py).

Un resumen de las sentencias que se pueden incluir en el Dockerfile se muestra en la siguiente figura:

```

FROM <image_id>
    base image to build this image from
RUN <command> shell form
RUN ["<executable>",
    "<param1>", exec form
    ...,
    "<paramN>"]
    executes command to modify container's file system state
MAINTAINER <name>
    provides information about image creator
LABEL <key>=<value>
    adds searchable metadata to image
ARG <name>[=<default value>]
    defines overridable build-time parameter:
    docker build --build-arg <name>=<value> .
ENV <key>=<value>
    defines environment variable that will be visible during
    image build-time and container run-time
ADD <src> <dest>
    copies files from <src> (file, directory or URL) and adds them
    to container file system under <dst> path
COPY <src> <dest> similar to ADD, does not support URLs
VOLUME <dest>
    defines mount point to be shared with host or other containers
EXPOSE <port>
    informs Docker engine that container listens to port at run-time
WORKDIR <dest>
    sets build-time and run-time working directory
USER <user>
    defines run-time user to start container process
STOPSIGNAL <signal>
    defines signal to use to notify container process to stop
ENTRYPOINT shell form or exec form
    defines run-time command prefix that will be added to all
    run commands executed by docker run
CMD shell form or exec form
    defines run-time command to be executed by default when
    docker run command is executed

```

Una vez finalizado este paso, no olvidar de publicar todo el código generado en el proyecto del Gitlab correspondiente a esta sesión.

Instalación de Docker Engine en la máquina de la GCP

Una vez que hemos completado el paso anterior, es necesario que nos conectemos por SSH a la máquina en la GCP que se ha creado en la sección 2 de este ejercicio.

Para conectarte a las VMs mediante SSH en el navegador desde la consola de Google Cloud, haz lo siguiente:

1. En la consola de Google Cloud, ve a la página **Instancias de VM**.
2. En la lista de instancias de máquinas virtuales, haz clic en **SSH** en la fila de la instancia a la que deseas conectarte.

<input type="checkbox"/>	Name ^	Zone	Recommendation	Internal IP	External IP	Connect
<input type="checkbox"/>	✓ instance-1	us-east1-b		10.142.0.2 (nic0)	35.231.114.114 ↗	SSH ⌵

Como Docker Engine no está en la distribución por defecto del Debian que se ha utilizado para la creación de la máquina virtual, es necesario instalarlo completando los siguientes pasos:

<https://docs.docker.com/engine/install/debian/>

Como último paso, ejecutar el siguiente comando que permitirá que no sea necesario utilizar sudo para la ejecución de docker engine:

```
sudo usermod -aG docker $USER
```

Posteriormente, cerrar el terminal SSH y volverlo a abrir para que los cambios tengan efecto.

Asimismo, para este y futuros ejercicios, será necesario instalar docker compose. Para ello, es necesario ejecutar los siguientes pasos (consultar la opción *Install the plugin manually*):

<https://docs.docker.com/compose/install/linux/>

Una vez completado este paso, es necesario clonar el proyecto en gitlab (<https://teaching.sel.inf.uc3m.es>) correspondiente a la sesión 7.

<div><h3>Git Cheat Sheet</h3><div><div><h4>Setup</h4><p>Set the name and email that will be attached to your commits and tags</p><pre>\$ git config --global user.name "Danny Adams" \$ git config --global user.email "my-email@gmail.com"</pre></div><div><h4>Start a Project</h4><p>Create a local repo (omit <directory> to initialise the current directory as a git repo)</p><pre>\$ git init <directory> Download a remote repo \$ git clone <url></pre></div><div><h4>Make a Change</h4><p>Add a file to staging</p><pre>\$ git add <file></pre><p>Stage all files</p><pre>\$ git add .</pre><p>Commit all staged files to git</p><pre>\$ git commit -m "commit message"</pre><p>Add all changes made to tracked files & commit</p><pre>\$ git commit -am "commit message"</pre></div><div><h4>Basic Concepts</h4><p>main: default development branch origin: default upstream repo HEAD: current branch HEAD*: parent of HEAD HEAD~4: great-great grandparent of HEAD</p><p><i>By @DoabfeDanny</i></p></div></div></div> <div><div><h4>Branches</h4><p>List all local branches. Add -r flag to show all remote branches. -a flag for all branches.</p><pre>\$ git branch</pre><p>Create a new branch</p><pre>\$ git branch <new-branch></pre><p>Switch to a branch & update the working directory</p><pre>\$ git checkout <branch></pre><p>Create a new branch and switch to it</p><pre>\$ git checkout -b <new-branch></pre><p>Delete a merged branch</p><pre>\$ git branch -d <branch></pre><p>Delete a branch, whether merged or not</p><pre>\$ git branch -D <branch></pre><p>Add a tag to current commit (often used for new version releases)</p><pre>\$ git tag <tag-name></pre></div><div><h4>Merging</h4><p>Merge branch a into branch b. Add --no-ff option for no-fast-forward merge</p><pre>\$ git checkout b \$ git merge a</pre><p>Merge & squash all commits into one new commit</p><pre>\$ git merge --squash a</pre></div></div>
--

Rebasing

Rebase feature branch onto main (to incorporate new changes made to main). Prevents unnecessary merge commits into feature, keeping history clean

```
$ git checkout feature
$ git rebase main
```

Iteratively clean up a branches commits before rebasing onto main

```
$ git rebase -i main
```

Iteratively rebase the last 3 commits on current branch

```
$ git rebase -i HEAD~3
```

Undoing Things

Move (&/or rename) a file & stage move

```
$ git mv <existing_path> <new_path>
```

Remove a file from working directory & staging area, then stage the removal

```
$ git rm <file>
```

Remove from staging area only

```
$ git rm --cached <file>
```

View a previous commit (READ only)

```
$ git checkout <commit_ID>
```

Create a new commit, reverting the changes from a specified commit

```
$ git revert <commit_ID>
```

Go back to a previous commit & delete all commits ahead of it (revert is safer). Add --hard flag to also delete workspace changes (BE VERY CAREFUL)

```
$ git reset <commit_ID>
```

Review your Repo

List new or modified files not yet committed

```
$ git status
```

List commit history, with respective IDs

```
$ git log --oneline
```

Show changes to unstaged files. For changes to staged files, add --cached option

```
$ git diff
```

Show changes between two commits

```
$ git diff commit1_ID commit2_ID
```

Stashing

Store modified & staged changes. To include untracked files, add -u flag. For untracked & ignored files, add -a flag.

```
$ git stash
```

As above, but add a comment.

```
$ git stash save "comment"
```

Partial stash. Stash just a single file, a collection of files, or individual changes from within files

```
$ git stash -p
```

List all stashes

```
$ git stash list
```

Re-apply the stash without deleting it

```
$ git stash apply
```

Re-apply the stash at index 2, then delete it from the stash list. Omit stash@{n} to pop the most recent stash.

```
$ git stash pop stash@{2}
```

Show the diff summary of stash 1. Pass the -p flag to see the full diff.

```
$ git stash show stash@{1}
```

Synchronizing

Delete stash at index 1. Omit stash@{n} to delete last stash made

```
$ git stash drop stash@{1}
```

Delete all stashes

```
$ git stash clear
```

Add a remote repo

```
$ git remote add <alias> <url>
```

View all remote connections. Add -v flag to view urls.

```
$ git remote
```

Remove a connection

```
$ git remote remove <alias>
```

Rename a connection

```
$ git remote rename <old> <new>
```

Fetch all branches from remote repo (no merge)

```
$ git fetch <alias>
```

Fetch a specific branch

```
$ git fetch <alias> <branch>
```

Fetch the remote repo's copy of the current branch, then merge

```
$ git pull
```

Move (rebase) your local changes onto the top of new changes made to the remote repo (for clean, linear history)

```
$ git pull --rebase <alias>
```

Upload local content to remote repo

```
$ git push <alias>
```

Upload to a branch (can then pull request)

```
$ git push <alias> <branch>
```

Para ayudar a los usuarios no expertos de Git por línea de comandos, la imagen anterior proporciona un resumen de los principales comandos a considerar.

Creación de la coreografía de los componentes desarrollados mediante Docker Compose

Para coreografiar los distintos componentes del sistema de tacógrafo simulado, se debe editar un fichero denominado docker-compose.yml.

En este fichero se deben incluir la especificación de cada uno de los componentes. A título de ejemplo se proporciona la especificación de la coreografía de los componentes de la Unidad de Control y Simulador del Lector de Tarjetas.

```
services:
  tachograph_control_unit:
    build: ./ControlUnit
    image: tachograph_control_unit
    environment:
      - PYTHONUNBUFFERED=1
      - PORT=65431
    volumes:
      - ./ControlUnit/code:/etc/usr/src/code
  tachograph_card_reader:
    build: ./CardReader
    image: tachograph_card_reader
    environment:
      - PYTHONUNBUFFERED=1
      - UC_SIMULATOR_HOST=tachograph_control_unit
      - UC_SIMULATOR_PORT=65431
    volumes:
      - ./CardReader/code:/etc/usr/src/code
```

A continuación, se proporciona una referencia resumida de los elementos que se pueden incluir en un fichero docker-compose.yml.

DOCKER COMPOSE CHEAT SHEET

File					
structure					
services:					
container1:					
properties:	values				
container2:					
properties:	values				
networks:					
network:					
volumes:					
volume:					
Types					
value					
key:	value				
array					
key:					
-	value				
-	value				
dictionary					
master:					
key:	value				
key:	value				
Properties					
build					
build image from dockerfile in specified directory					
container:					
build:	./path				
image:	image-name				
image					
use specified image					
image:	image-name				
container_name					
define container name to access it later					
container_name:	name				
volumes					
define container volumes to persist data					
volumes:					
-	/path:/path				
command					
override start command for the container					
command:	execute				
environment					
define env variables for the container					
environment:					
KEY:	VALUE				

environment:					
-	KEY=VALUE				
env_file					
define a env file for the container to set and override env variables					
env_file:	.env				

env_file:					
-	.env				
restart					
define restart rule (no, always, on-failure, unless-stopped)					
restart:					
expose:					
-	"9999"				
networks					
define all networks for the container					
networks:					
-	network-name				
ports					
define ports to expose to other containers and host					
ports:					
-	"9999:9999"				
expose					
define ports to expose only to other containers					
expose:					
-	"9999"				
network_mode					
define network driver (bridge, host, none, etc.)					
network_mode:	host				
depends_on					
define build, start and stop order of container					
depends_on:					
-	container-name				
Other					
idle container					
send container to idle state > container will not stop					
command:	tail -f /dev/null				
named volumes					
create volumes that can be used in the volumes property					
services:					
container:					
image:	image-name				
volumes:					
-	data-				
volume:	/path/to/dir				
volumes:					
data-volume:					
networks					
create networks that can be used in the networks property					
networks:					
frontend:					
driver:	bridge				



Despliegue del gemelo digital del tacógrafo digital

Para desplegar todos los componentes del gemelo digital del tacógrafo digital 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 la sección 2 de este enunciado.

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 *session01* y lanzar los servicios de esta máquina ejecutando los comandos:

```
docker compose build  
  
docker compose up -d
```

Para verificar que los distintos componentes del gemelo digital del tacógrafo se encuentra en ejecución, se pueden ejecutar los comandos siguientes:

```
docker ps -a  
  
docker logs <nombre del contenedor>
```

Criterios de evaluación y procedimiento de entrega



Criterios de Evaluación

- Gemelo Digital – Generación de Rutas – 3 puntos
- Gemelo Digital – Lector de Tarjetas Simulado – 1 puntos
- Gemelo Digital – Odómetro Simulado – 1,5 puntos
- Gemelo Digital – GNSS Simulado – 2,5 puntos
- Gemelo Digital – Unidad de Control Simulada – 2 puntos



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 Sesión01 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, 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.



Sugerencias

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