

SISTEMAS INFORMÁTICOS I

PRÁCTICA 1

AUTORES

CARLOS GARCÍA SANTA
EDUARDO JUNOY ORTEGA

SISTEMAS INFORMÁTICOS
GRUPO1321: PAREJA 05

INGENIERÍA INFORMÁTICA
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD AUTÓNOMA DE MADRID



19/10/2023

Índice

Índice.....	2
Introducción.....	2
PARTE 1:	
4.1 Entorno de python.....	3
4.2 Micro-servicio en Quart.....	3
4.3 Depliegue con Hypercorn.....	5
5.3 Contenedores docker.....	6
5.4 Creación de una imagen de contenedor con un Dockerfile.....	10
5.5. Docker registry.....	14
PARTE 2:	
3. AWS S3.....	15
5. API Gateway.....	19
6. DynamoDB.....	26

Introducción

Esta memoria documenta los conocimientos desarrollados y aplicados en la primera práctica. Esta ha estado centrada en el desarrollo de un micro-servicio en Python que implementa una REST API, abordando aspectos clave como la gestión de contenedores, el despliegue en Docker, la integración de servicios en la nube mediante LocalStack, y la implementación de funciones Lambda. A lo largo de este informe, se detallan los procedimientos y resultados obtenidos.

A continuación se presenta un listado de archivos empleados junto con una breve descripción:

user_get_lambda_dynamo.py: Es la función Lambda para consultar usuarios en DynamoDB. Usa boto3 para conectarse y recuperar datos.

user_lambda.py: Es una función Lambda que almacena información de usuario en S3. Usa boto3 para interactuar con el bucket S3.

user_lambda_dynamo.py: Es la función Lambda para guardar datos de usuario en DynamoDB. Usa boto3 para escribir en la tabla.

user_lambda_param.py: Es una función Lambda que sirve para almacenar en S3, tomando el username desde pathParameters. Similar a user_lambda.py pero con fuente de datos diferente.

user_rest.py: Es el microservicio REST con Quart para gestionar usuarios. Almacena y recupera datos de usuario en archivos JSON locales.

Parte 1: REST API en contenedor

4.1 Entorno de python

Ambos miembros del grupo realizamos la instalación del entorno correctamente:

```
(base) eduardo@edu:~$ mkdir -p venv/silp1
(base) eduardo@edu:~$ python3 -m venv venv/silp1
(base) eduardo@edu:~$ source ./venv/silp1/bin/activate
(silp1) (base) eduardo@edu:~$ pip install hypercorn quart localstack awscli awscli-local
```

Para ello hemos instalado previamente el paquete python3-venv con el siguiente comando:

```
sudo apt install python3-venv
```

4.2 Micro-servicio en Quart

Tarea: revisión de user_rest.py

Se revisa el archivo user_rest.py y la bibliografía de Quart y Flask para contestar a las siguientes preguntas.

Pregunta: user_rest.py

1. ¿Qué crees que hacen las anotaciones `@app.route`, `@app.get`, `@app.put`?

`@app.route('/')`: Es un decorador que nos permite comunicarnos con Flask indicando en ('/') la URL que debería activar una funcionalidad determinada, siendo en nuestro caso la función "hello()". Por defecto, maneja peticiones GET, pero puede ser configurada para manejar otros métodos HTTP usando el parámetro `methods`.

`@app.get('/user/<username>')`: Define específicamente una ruta que maneja peticiones GET para una ruta determinada siendo en nuestro caso `"/user/<nombre_de_usuario>"`.

`@app.put('/user/<username>')`: Define específicamente una ruta que maneja peticiones PUT siendo nuestra ruta `"/user/<nombre_de_usuario>"`

2. ¿Qué ventajas o inconvenientes crees que pueden tener?

La ventaja principal que se encuentra es que el uso de decoradores junto a las funciones que manejan las peticiones permite una fácil identificación de las rutas y sus métodos asociados, lo que facilita la comprensión rápida del flujo del programa. Pero a su vez tienen la desventaja de aunque simplifican el código, estos decoradores también ocultan parte de la lógica interna, lo que también proporciona menor flexibilidad y mayor dependencia del entorno de trabajo.

Tarea: ejecución de `user_rest.py`

```
(si1p1) (base) eduardo@edu:~$ source venv/si1p1/bin/activate
(si1p1) (base) eduardo@edu:~$ [[ -d build/users ]] || mkdir -p build/users

(si1p1) (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ QUART_APP=src.user_rest:app quart run -p 5050
* Serving Quart app 'src.user_rest'
* Environment: production
* Please use an ASGI server (e.g. Hypercorn) directly in production
* Debug mode: False
* Running on http://127.0.0.1:5050 (CTRL + C to quit)
[2023-10-18 15:45:11 +0200] [16259] [INFO] Running on http://127.0.0.1:5050 (CTRL + C to quit)
```

¿Qué crees que ha sucedido?

Se activa un entorno virtual de Python llamado si1p1. Luego, se crea si es necesario el directorio build/users. Después, se ejecuta `src.user_rest` con Quart, y se arranca en el puerto 5050 estando el servidor en escucha para las peticiones a la dirección `http://127.0.0.1:5050`.

Tarea: petición a `user_rest.py`

Instalamos curl:

```
sudo apt-get install curl
```

Pregunta: petición a `user_rest.py`

1. ¿Qué crees que ha sucedido al ejecutar el mandato curl?

Al ejecutar el comando curl, se intenta hacer una solicitud PUT al servidor local en el puerto 5050 para crear o actualizar un usuario llamado "myusername" con el nombre "myFirstName".

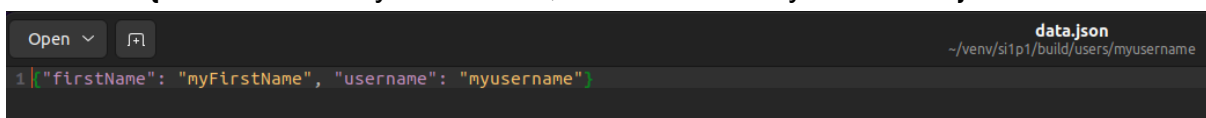
2. ¿Cuál es la respuesta del microservicio?

```
[2023-10-11 16:08:26 +0200] [16259] [INFO] 127.0.0.1:60814 PUT
/user/myusername 1.1 200 15 731.
```

La respuesta del microservicio es correcta, indicado por el código de estado 200. La solicitud PUT a `/user/myusername` se procesa correctamente.

3. ¿Se ha generado algún archivo? ¿Cuál es su ruta y contenido?

Se genera en `Home/venv/si1p1/build/users` la carpeta `myusername` que contiene un archivo json con los datos del primer apellido del usuario y el nombre de usuario, en este caso `{"firstName": "myFirstName", "username": "myusername"}`.



```
data.json
~/venv/si1p1/build/users/myusername
1 [{"firstName": "myFirstName", "username": "myusername"}]
```

Tarea: petición HTTP

Se ejecuta de nuevo el comando curl, pero haciendo la petición al puerto 8888.

Pregunta: petición HTTP

1. ¿Cuál es la petición HTTP que llega al programa nc?

nc -l 8888

PUT /user/myusername HTTP/1.1

Host: localhost:8888

User-Agent: curl/7.81.0

Accept: */*

Content-Type: application/json

Content-Length: 23

{"firstName": "Carlos"}

2. ¿En qué campo de la cabecera HTTP se indica el tipo de dato del cuerpo (body) de la petición y cuál es ese tipo?

El campo de la cabecera HTTP que indica el tipo de dato del cuerpo (body) de la petición es Content-Type. En este caso, el tipo es application/json.

3. ¿Qué separa la cabecera del cuerpo?

La cabecera y el cuerpo en una petición HTTP están separados por dos saltos de línea consecutivos.

Tarea: prototipo user

1. Se implementa el método DELETE para eliminar un usuario:
2. Se implementa el método PATCH para actualizar algún campo del usuario

4.3 Depliegue con Hypercorn

Tarea: ejecución hypercorn

hypercorn --bind 0.0.0.0:8080 --workers 2 src.user_rest:app

Pregunta: ejecución hypercorn

1. ¿Qué crees que ha sucedido?

Se ha iniciado un servidor hypercorn para ejecutar la aplicación src.user_rest:app. Se ha configurado para escuchar en todas las interfaces de red (0.0.0.0) en el puerto 8080, además, se han lanzado dos procesos para manejar las peticiones, como lo indica la opción --workers.

2. ¿Qué petición curl debes hacer ahora para hacer un GET de un usuario?
¿Qué ha cambiado respecto de la ejecución sin hypercorn?

```
curl http://localhost:8080/user/{myusername}
```

Lo que ha cambiado respecto a la ejecución sin hypercorn es el puerto y el número de peticiones que se pueden realizar de forma simultánea. Anteriormente, se usaba el puerto 5050, y ahora, con hypercorn, se usa el puerto 8080, además, ahora se emplean dos trabajadores o procesos en lugar de uno.

5.3 Contenedores docker

Tarea: docker info

```
docker info
```

Pregunta: docker info

1. ¿Dónde se guarda la configuración del demonio arrancado?

La configuración del demonio Docker se guarda en Docker Root Dir: /etc/docker/

2. ¿Dónde se almacenan los contenedores?

En /var/lib/docker

```
root@carlos-IdeaPad-3-15ITL6:/var/lib# cd docker/
root@carlos-IdeaPad-3-15ITL6:/var/lib/docker# ls
buildkit  engine-id  network  plugins  swarm  volumes
containers  image      overlay2  runtimes  tmp
root@carlos-IdeaPad-3-15ITL6:/var/lib/docker#
```

3. ¿Qué tecnología de almacenamiento se usa para los contenedores?
¿Qué es copy-on-write? ¿Lo usa docker?

Docker utiliza una tecnología de almacenamiento llamada "overlay2" de forma predeterminada para los contenedores, que es un sistema de archivos de capas que permite la creación de imágenes y contenedores a partir de capas de archivos. Cada capa es un conjunto de cambios en el sistema de archivos, y las capas se apilan para formar una imagen o un contenedor.

El concepto de "copy-on-write" (COW) es fundamental en overlay2, ya que cuando se modifica un archivo en un contenedor, en lugar de cambiar el archivo original, se crea una copia de la capa que contiene el archivo y se realizan las modificaciones en esa copia. Esto garantiza que las capas subyacentes permanezcan inmutables y que las modificaciones se almacenen en una capa separada.

Tarea: run whoami

```
docker run alpine whoami
```

Pregunta: run whoami

1. ¿Qué crees que ha sucedido?

Se ha ejecutado un contenedor basado en la imagen alpine y se ha consultado el nombre de usuario actual dentro de ese contenedor, en nuestro caso se muestra root en la terminal que corresponde al usuario que está dentro del contenedor.

2. ¿Crees que el usuario root del contenedor es el mismo que el del host?

No, el usuario root dentro del contenedor Docker no es el mismo que el usuario root del host. Los contenedores Docker están diseñados para ser aislados y separados del sistema operativo.

Tarea: list images

`docker images`

Pregunta: list images

1. ¿Qué crees que tiene que ver con el mandato docker pull?

La relación entre docker pull y docker images es que docker pull se utiliza para descargar imágenes desde un registro de contenedores como Docker Hub y almacenarlas localmente en el sistema, después de ejecutar docker pull, la imagen se encuentra disponible localmente y con la ejecución de docker images se muestra una lista de las imágenes que están almacenadas en el sistema local, dentro de las cuales estarán las descargadas por el comando docker pull.

Tarea: docker ps

`docker ps -a`

Pregunta: docker ps

1. ¿Qué crees que muestra la salida?

Docker ps -a muestra una lista de todos los contenedores en el sistema, incluyendo los que están detenidos. Cada fila proporciona información sobre el ID del contenedor, la imagen utilizada, el comando, la fecha de creación, el estado actual y otros detalles relacionados con cada contenedor.

Tarea: docker pull y run -ti

`docker pull ubuntu:22.04`

`docker run -ti --name u22.04 ubuntu`

Pregunta: docker pull y run -ti

1. ¿Qué hacen las opciones -ti?

Las opciones -t -i permiten crear y ejecutar un contenedor en modo interactivo con un terminal asociado. --t o --tty nos da una terminal interactiva dentro del

contenedor; -i o --interactive permite la interacción en tiempo real con la entrada estándar (stdin) del contenedor.

2. ¿Qué ha sucedido?

Se descarga la imagen de Ubuntu 22.04 y luego se crea un contenedor interactivo basado en la imagen con el nombre "u22.04", de tal manera que se accede dentro del contenedor en una sesión de terminal interactiva.

Tarea: docker inspect

```
sudo apt-get install jq
docker inspect u22.04 | jq '.[].Config.Cmd[]'
```

Pregunta: docker inspect

1. ¿Qué hace el mandato jq?

Es un comando que nos permite filtrar datos JSON, en nuestro caso se utiliza para analizar el resultado de `docker inspect u22.04`, que proporciona información sobre el contenedor u22.04 en formato JSON. La sentencia `[][.Config.Cmd[]]` en jq se utiliza para extraer y mostrar el valor del campo Cmd siendo el resultado `/bin/bash`, que indica que el comando ejecutado dentro del contenedor u22.04 es `/bin/bash`.

2. ¿Qué tiene que ver el resultado del mandato con la tarea anterior?

La ejecución del comando `/bin/bash` indica que se puede interactuar directamente con la terminal dentro del contenedor, que es lo que queríamos con la ejecución de los comandos de la tarea anterior.

Tarea: docker start

```
docker start u22.04
```

Pregunta: docker start

1. ¿Qué ha sucedido?

El comando `docker start` inicia un contenedor Docker llamado u22.04, la salida de la terminal confirma la acción mostrando el nombre del contenedor, lo que indica que se ha iniciado correctamente.

2. ¿Cómo se borra el contenedor u22.04?

```
docker stop u22.04 para detener el contenedor
docker rm u22.04 para eliminar el contenedor
```

Tarea: docker run -d

```
docker run -ti -d --name u22.04d ubuntu
```


Pregunta: docker run -d

1. ¿Qué hace la opción -d?

Con -d el contenedor se inicia y sigue ejecutándose, pero permite que se continúe usando ese terminal para otros comandos o tareas.

Tarea: docker exec

```
docker exec -ti u22.04d /bin/bash
```

y dentro del contenedor:

```
mkdir -p /si1/users
```

```
touch /si1/users/test
```

Pregunta: docker exec

1. ¿Qué ha sucedido?

```
bash: sudo: command not found
root@67e8e19eed76:/# mkdir -p /si1/users
root@67e8e19eed76:/# touch /si1/users/test
root@67e8e19eed76:/#
exit
```

Se entra al contenedor u22.04d, se crea una carpeta llamada users en /si1, y dentro de esa carpeta, se crea un archivo vacío llamado test.

Tarea: docker volume

```
[[ -d si1/users/testv ]] || mkdir -p si1/users/testv
```

```
docker run -ti -v ${PWD}/si1/users/testv:/si1/users/testv
alpine \
```

```
touch /si1/users/testv/afile
```

```
docker run -ti -v ${PWD}/si1/users/testv:/si1/users/testv
alpine \
```

```
ls -al /si1/users/testv/afile
```

```
ls -al si1/users/testv/afile
```

```
echo "a test" >>si1/users/testv/afile
```

```
docker run -ti -v ${PWD}/si1/users/testv:/si1/users/testv
alpine \
```

```
cat /si1/users/testv/afile
```

Pregunta: docker volume

1. ¿Qué hace la opción -v?

La opción `-v` o `--volume` en el comando se utiliza para montar un volumen, permitiendo vincular una carpeta o archivo del sistema local a un contenedor Docker.

2. ¿Qué ha sucedido al ejecutar los mandatos anteriores?

Se crea un directorio usando Docker y la opción de montaje `-v`, y se ejecuta con un archivo llamado `afile` dentro de ese directorio. Primero, se crea el archivo usando un contenedor `alpine` y se listan los detalles del archivo tanto desde el contenedor como localmente, finalmente, se muestra el contenido de ese archivo desde otro contenedor reflejando los cambios hechos localmente debido al montaje del volumen.

Tarea: limpieza

Se eliminan todos los contenedores e imágenes empleados.

Pregunta: limpieza

1. ¿Qué mandato has usado para eliminar los contenedores?

```
sudo docker rm -f $(sudo docker ps -aq)
```

2. ¿Qué mandato has usado para eliminar las imágenes?

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

5.4 Creación de una imagen de contenedor con un Dockerfile

Tarea: Dockerfile

Se revisa la documentación sobre el Dockerfile en la bibliografía y el archivo `src/Dockerfile` suministrado para responder a las preguntas siguientes.

Pregunta: Dockerfile

1. ¿Qué hace la sentencia `FROM`?

La sentencia `FROM` define la imagen base desde la cual se construirá el contenedor.

2. ¿Qué hace la sentencia `ENV`?

`ENV` define una variable de entorno con la forma clave-valor dentro del contenedor, esta variable estará disponible posteriormente en la ejecución.

3. ¿Qué diferencia presenta `ENV` respecto de `ARG`?

Ambas son para definir variables, sin embargo, ENV define una variable de entorno que está disponible para los comandos y a la cual se puede acceder tanto en la construcción como en la ejecución del contenedor, por otro lado, ARG define una variable que solo está disponible durante la construcción.

4. ¿Qué hace la sentencia RUN?

El comando RUN en Docker ejecuta comandos en una nueva capa sobre la imagen actual y guarda los resultados como una nueva imagen.

5. ¿Qué hace la sentencia COPY?

COPY copia archivos o directorios del sistema anfitrión al sistema de archivos del contenedor.

6. ¿Qué otra sentencia del Dockerfile tiene un cometido similar a COPY?

La sentencia ADD es similar a COPY. Sin embargo, además de copiar archivos, ADD tiene la capacidad de descargar archivos desde URL y puede descomprimir archivos tar.gz en el directorio destino.

7. ¿Qué hace la sentencia EXPOSE?

EXPOSE informa a Docker que el contenedor escuchará en un puerto específico en tiempo de ejecución.

8. ¿Qué hace la sentencia CMD?

CMD proporciona valores predeterminados para un contenedor en ejecución.

9. ¿Qué otras sentencias del Dockerfile tienen un cometido similar a CMD?

La sentencia ENTRYPOINT es similar a CMD, ambas especifican lo que se debe ejecutar cuando el contenedor se inicia. Sin embargo, mientras que CMD puede ser fácilmente sobrescrito por el usuario cuando se ejecuta el contenedor, ENTRYPOINT no lo es (a menos que se use el flag --entrypoint en el comando docker run).

Tarea: docker build

```
cd src
docker build --tag si1p1:latest .
docker build --tag si1p1:1.0 .
docker images
cd -
```

```
(base) carlos@carlos-IdeaPad-3-15ITL6:~/VScode/Info/SI/p1-v1.0/src$ sudo docker build --tag siip1:latest .
[sudo] password for carlos:
[+] Building 37.2s (12/12) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 353B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [1/7] FROM docker.io/library/ubuntu
=> [internal] load build context
=> => transferring context: 2.20kB
=> [2/7] RUN apt update
=> [3/7] RUN apt install -y python3-pip
=> [4/7] RUN pip install hypercorn quart
=> [5/7] RUN mkdir -p /si1/build/users
=> [6/7] WORKDIR /si1
=> [7/7] COPY user_rest.py ./
=> exporting to image
=> => exporting layers
=> => writing image sha256:010a4cb96e6f2f1ef962696e0363f666214856ff3ca99975876db021db7c5cb3
=> => naming to docker.io/library/siip1:latest
(base) carlos@carlos-IdeaPad-3-15ITL6:~/VScode/Info/SI/p1-v1.0/src$ sudo docker build --tag siip1:1.0 .
[+] Building 0.2s (12/12) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 353B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [1/7] FROM docker.io/library/ubuntu
=> [internal] load build context
=> => transferring context: 348
=> CACHED [2/7] RUN apt update
=> CACHED [3/7] RUN apt install -y python3-pip
=> CACHED [4/7] RUN pip install hypercorn quart
=> CACHED [5/7] RUN mkdir -p /si1/build/users
=> CACHED [6/7] WORKDIR /si1
=> CACHED [7/7] COPY user_rest.py ./
=> exporting to image
=> => exporting layers
=> => writing image sha256:010a4cb96e6f2f1ef962696e0363f666214856ff3ca99975876db021db7c5cb3
=> => naming to docker.io/library/siip1:1.0
(base) carlos@carlos-IdeaPad-3-15ITL6:~/VScode/Info/SI/p1-v1.0/src$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
siip1                1.0                010a4cb96e6f       33 seconds ago     492MB
siip1                latest             010a4cb96e6f       33 seconds ago     492MB
ubuntu              22.04             e4c58958181a       13 days ago        77.8MB
alpine              latest             8ca4688f4f35       2 weeks ago        7.34MB
ubuntu              latest             3565a89d9e81       3 weeks ago        77.8MB
```

1. ¿Por qué es mucho más rápido la creación de la segunda imagen?

La construcción de la segunda imagen es mucho más rápida porque Docker utiliza un sistema de almacenamiento en capas. Durante la primera construcción, Docker genera y almacena en caché varias capas para cada uno de los pasos, pero cuando se construye la imagen nuevamente, Docker detecta que muchos de esos pasos ya se han realizado y simplemente reutiliza las capas almacenadas en caché, en lugar de realizar nuevamente todo el proceso, esto es evidente por las entradas "CACHED" que se ven en la salida de la segunda construcción.

2. Las imágenes constan de distintas capas: ¿Qué quiere decir esto?

Cada capa representa una instrucción en el Dockerfile, por lo tanto cada vez que se ejecuta una instrucción, se genera una nueva capa. Estas capas son inmutables, lo que significa que una vez creadas, no se pueden modificar.

Tarea: docker build with user

Descomenta las líneas del Dockerfile que se refieren al usuario si1 y vuelve a construir la imagen, esta vez con el tag 1.0u.

Pregunta: docker build with user

1. ¿Por qué en los últimos pasos no utiliza la caché?

Una vez se descomentan las líneas y se ejecutan los comandos por primera vez, las instrucciones no están cacheadas y Docker tiene que ejecutarlas, esto se refleja en los registros con la ausencia de la etiqueta CACHED junto a esas instrucciones, pero en la segunda construcción, ya que no hay cambios adicionales al Dockerfile desde la última construcción, se utiliza la caché para todas las instrucciones.

Tarea: docker run myimage

```
docker run --name si1p1 -e NUMWORKERS=5 -d -p 8080:8000 si1p1
docker exec -ti si1p1 /bin/bash
# ... dentro del contenedor
ps --forest -aef
exit
```

Para borrar el contenedor, se ejecuta:

```
docker rm -f si1p1
```

```
si1p1
(base) carlos@carlos-IdeaPad-3-15ITL6:~$ sudo docker run --name si1p1 -e NUMWORKERS=5 -d -p 8080:8000 si1p1
9997ecd89a276edc8bfd3c0de3b67134f4375254689120035c6e1676eefa1d84
(base) carlos@carlos-IdeaPad-3-15ITL6:~$ sudo docker exec -ti si1p1 /bin/bash
si1@9997ecd89a27:/si1$ ps --forest -aef
UID          PID    PPID  C STIME TTY          TIME CMD
si1           14      0  0 17:23 pts/0      00:00:00 /bin/bash
si1           21     14  0 17:24 pts/0      00:00:00 \_ ps --forest -aef
si1            1      0  0 17:23 ?          00:00:00 /bin/sh -c hypercorn --bind
si1            7      1  0 17:23 ?          00:00:00 /usr/bin/python3 /usr/local/
si1            8      7  0 17:23 ?          00:00:00 \_ /usr/bin/python3 -c from
si1            9      7  0 17:23 ?          00:00:00 \_ /usr/bin/python3 -c from
si1           10      7  0 17:23 ?          00:00:00 \_ /usr/bin/python3 -c from
si1           11      7  0 17:23 ?          00:00:00 \_ /usr/bin/python3 -c from
si1           12      7  0 17:23 ?          00:00:00 \_ /usr/bin/python3 -c from
si1           13      7  0 17:23 ?          00:00:00 \_ /usr/bin/python3 -c from
si1@9997ecd89a27:/si1$ exit
exit
(base) carlos@carlos-IdeaPad-3-15ITL6:~$ sudo docker rm -f si1p1
si1p1
```

Pregunta: docker run myimage

1. ¿Cuántos subprocesos de hypercorn aparecen? ¿Por qué?

Aparecen 6 procesos relacionados con hypercorn (PIDs 7, 8, 9, 10, 11, 12, 13).

Dado que se especifica la creación de 5 “trabajadores”, se ven 5 subprocesos cuyo PPID 7 hace referencia al PID del proceso principal.

2. ¿Qué hace la opción '-p' del comando docker run?

La opción -p del comando docker run se usa para publicar un puerto o un rango de puertos desde el contenedor al host (la máquina donde se está ejecutando Docker).

3. Si deseáramos ejecutar otra réplica (contenedor) del microservicio, ¿qué deberíamos cambiar en la sentencia docker run?

El nombre del contenedor (--name): Se debe asignar un nombre diferente para la réplica para evitar conflictos, ya que dos contenedores no pueden tener el mismo nombre.

Mapeo de puertos (-p): También se necesitaría cambiar el puerto host para evitar conflictos, ya que el puerto 8080 ya está en uso por el contenedor anterior.

Tarea: docker run myimage on ./si1

Usando un volumen (opción -v), crea en local el directorio ./si1 y arranca otra réplica de nuestra aplicación montando el directorio creado en el directorio /si1 del contenedor.

Pregunta: docker run myimage on ./si1

1. ¿Qué mandato has ejecutado?

```
sudo docker run --name si1p1-replica2 -e NUMWORKERS=5 -d -p 8082:8000 -v $(pwd)/si1:/si1 si1p1
```

2. De cara a la realización de backups, ¿cuál crees que es la utilidad de montar volúmenes externos respecto de usar los internos de docker?

Montar volúmenes externos en Docker proporciona persistencia de datos, garantizando que la información se mantenga incluso si el contenedor se elimina.

5.5. Docker registry

Tarea: docker registry

Para responder a estas preguntas se arranca un contenedor de dicha imagen con el nombre si1_registry y escuchando en el puerto 5050.

Pregunta: docker registry

1. ¿Qué mandato has usado para descargar la imagen del registry?

```
sudo docker pull registry
```

2. ¿Cuál es el número de versión de dicha imagen?

2.8.3 (o latest).

3. ¿Qué mandato has usado para ejecutar el contenedor del registry?

```
sudo docker run -d -p 5050:5000 --name si1_registry registry:latest
```

4. ¿En qué puerto escucha por defecto el contenedor del registry?

El contenedor de registry escucha en el puerto 5000 por defecto.

Parte 2: REST API en cloud

3. AWS S3

Tarea: AWS S3

Para empezar la práctica, se ejecuta en un terminal, desde el directorio donde está el subdirectorio src con el archivo src/user_rest.py

Pregunta: AWS S3

1. ¿Qué hacen los distintos subcomandos de awslocal s3?

`awslocal s3 mb s3://si1p1`: Crea un nuevo bucket llamado si1p1.

`awslocal s3 cp src/user_rest.py s3://si1p1/src/user_rest.py`: Copia el archivo user_rest.py desde el directorio src local al bucket si1p1 en la ruta especificada (src/user_rest.py).

`awslocal s3 ls s3://si1p1`: Lista los contenidos del bucket si1p1.

`awslocal s3 ls s3://si1p1/src`: Lista los contenidos del directorio src dentro del bucket si1p1.

`awslocal s3 ls s3://si1p1/src/`: Lista los contenidos del directorio src dentro del bucket si1p1.

2. ¿Qué comando podemos usar para descargar el archivo user_rest.py desde el bucket que hemos creado, dándole otro nombre para evitar sobreescribirlo?

`awslocal s3 cp s3://si1p1/src/user_rest.py ./user_rest_copy.py`

```
(base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal s3 mb s3://si1p1
make_bucket: si1p1
(base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal s3 cp src/user_rest.py s3://si1p1/src/user_rest.py
upload: src/user_rest.py to s3://si1p1/src/user_rest.py
(base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal s3 ls s3://si1p1
PRE src/
(base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal s3 ls s3://si1p1/src
PRE src/
(base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal s3 ls s3://si1p1/src/
2023-10-18 23:17:54      1088 user_rest.py
(base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$

(base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal s3 cp s3://si1p1/src/user_rest.py ./user_rest_copy.py
download: s3://si1p1/src/user_rest.py to ./user_rest_copy.py
```

Tarea: Lambdas

LAMBDA_NAME=user_lambda

`[[-d build]] || mkdir build ; \`

`cd src ; zip ../build/${LAMBDA_NAME}.zip ${LAMBDA_NAME}.py; cd -`
`echo "create: newlambda"; \`


```
awslocal lambda create-function \
--function-name ${LAMBDA_NAME} \
--runtime python3.9 \
--zip-file fileb://build/${LAMBDA_NAME}.zip \
--handler ${LAMBDA_NAME}.handler \
--role arn:aws:iam::000000000000:role/si1 ; \
awslocal lambda create-function-url-config \
--function-name ${LAMBDA_NAME} \
--auth-type NONE ;
# Wait
awslocal lambda wait function-active-v2 --function-name
${LAMBDA_NAME}
awslocal lambda list-functions
curl -X POST \
$(awslocal lambda get-function-url-config --function-name
${LAMBDA_NAME} |
jq -r '.FunctionUrl') \
-H 'Content-Type: application/json' \
-d '{"username": "pepe", "data": "10"}'
```

```
(base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal lambda wait function-active-v2 --function-name ${LAMBDA_NAME}
awslocal lambda list-functions
{
  "Functions": [
    {
      "FunctionName": "user_lambda",
      "FunctionArn": "arn:aws:lambda:us-east-1:000000000000:function:user_lambda",
      "Runtime": "python3.9",
      "Role": "arn:aws:iam::000000000000:role/si1",
      "Handler": "user_lambda.handler",
      "CodeSize": 799,
      "Description": "",
      "Timeout": 3,
      "MemorySize": 128,
      "LastModified": "2023-10-19T07:45:53.018397+0000",
      "CodeSha256": "03sbxofanFmfQQRDwj1cyT/pFVaT+M3QpMPEPErUfqE=",
      "Version": "$LATEST",
      "TracingConfig": {
        "Mode": "PassThrough"
      },
      "RevisionId": "933a47c9-0f00-4cb0-afbe-5407c5dbcb6c",
      "PackageType": "Zip",
      "Architectures": [
        "x86_64"
      ],
      "EphemeralStorage": {
        "Size": 512
      },
      "SnapStart": {
        "ApplyOn": "None",
        "OptimizationStatus": "Off"
      }
    }
  ]
}
(base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ curl -X POST \
$(awslocal lambda get-function-url-config --function-name ${LAMBDA_NAME} |
jq -r '.FunctionUrl') \
-H 'Content-Type: application/json' \
-d '{"username": "pepe", "data": "10"}'
{"test_object_key" placed into S3}(base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$
```


Pregunta: Lambdas

1. ¿Qué runtime podríamos usar si por ejemplo queremos implementar una función Lambda en JavaScript?

Para implementar una función Lambda en JavaScript, se puede usar el runtime de Node.js.

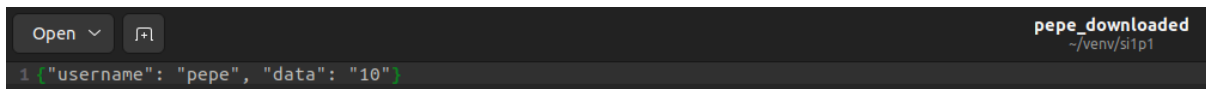
2. ¿Cómo recibe la función lambda el usuario los datos que hemos enviado con el comando curl?

La función lambda recibe datos enviados con curl como un evento. En el caso de un trigger HTTP, el evento tendría información sobre el método HTTP, cabeceras y cuerpo del request, entre otros.

3. Comprueba que ha funcionado descargando del bucket el fichero creado por la función. ¿Qué comandos has utilizado para ello?

`awslocal s3 cp s3://si1p1/users/pepe ./pepe_downloaded`

Cuyo resultado es: `{"username": "pepe", "data": "10"}`.



```

Open  [icon] pepe_downloaded
~/env/si1p1
1 {"username": "pepe", "data": "10"}
  
```

Tarea: depuración lambdas

```

docker logs localstack_main
awslocal logs describe-log-groups
awslocal logs describe-log-streams --log-group-name
/aws/lambda/${LAMBDA_NAME}
# last logs
awslocal logs describe-log-streams --log-group-name
/aws/lambda/${LAMBDA_NAME} |
jq '.logStreams[].firstEventTimestamp' | sort -n
LLOGSTREAMS=$(awslocal logs describe-log-streams \
--log-group-name /aws/lambda/${LAMBDA_NAME} |
jq -r '.logStreams[].logStreamName')
for LOGSTREAM in ${LLOGSTREAMS} ; do
awslocal logs get-log-events \
--log-group-name "/aws/lambda/${LAMBDA_NAME}" \
--log-stream-name "${LOGSTREAM}" ; done |
jq '.events[].message'
  
```

Pregunta: depuración lambdas

1. ¿Describe a qué corresponde la salida de cada uno de los comandos anteriores?

`docker logs localstack_main` Este comando muestra los logs del contenedor Docker llamado "localstack_main".

`awslocal logs describe-log-groups` Solicita información sobre los grupos de logs en el ambiente LocalStack.

```
awslocal      logs      describe-log-streams      --log-group-name
/aws/lambda/${LAMBDA_NAME}
```

Solicita los flujos de logs asociados con un grupo de logs específico, en este caso, un grupo de logs asociado con una función Lambda.

```
awslocal      logs      describe-log-streams      --log-group-name
/aws/lambda/${LAMBDA_NAME} | jq '.logStreams[].firstEventTimestamp'
| sort -n
```

Solicita los flujos de logs del grupo de logs asociado con la función Lambda y luego filtra para obtener solo los timestamps del primer evento de cada flujo y los ordena numéricamente.

```
LLOGSTREAMS=$(awslocal logs describe-log-streams \ --log-group-name
/aws/lambda/${LAMBDA_NAME} | jq -r '.logStreams[].logStreamName')
```

Asigna a la variable LLOGSTREAMS los nombres de todos los flujos de logs asociados con el grupo de logs de la función Lambda.

```
for LOGSTREAM in ${LLOGSTREAMS} ; do awslocal logs get-log-events \
--log-group-name "/aws/lambda/${LAMBDA_NAME}" \ --log-stream-name
"${LOGSTREAM}" ; done | jq '.events[].message'
```

Este bucle recorre cada flujo de logs almacenado en LLOGSTREAMS, recupera sus eventos y luego filtra solo los mensajes de esos eventos usando jq explicado anteriormente.

Tarea: Borrado de funciones

```
echo "clean: dellambda" ; \
awslocal lambda delete-function-url-config \
--function-name user_lambda ; \
awslocal lambda delete-function --function-name user_lambda
echo "clean: md zip" \
awslocal s3 rb --force s3://si1p1 ; \
```

```
rm build/user_lambda.*
```

Pregunta: Borrado de funciones

1. ¿Qué ocurre si en lugar de borrar las funciones y archivos en S3, simplemente reiniciamos localStack? ¿Cómo podríamos evitarlo?

Si simplemente reiniciamos LocalStack sin borrar las funciones y archivos en S3, estos recursos seguirán existiendo en LocalStack cuando se reinicie, siempre y cuando no se borre el volumen persistente de datos de LocalStack. Para evitar que los recursos persistan después de reiniciar, se debe usar un volumen no persistente o borrar el volumen antes de reiniciar LocalStack.

5. API Gateway

Tarea: Creación del API

```
export REGION="us-east-1" ; export API_NAME="user_lambda"
awslocal apigateway create-rest-api \
--region ${REGION} \
--name ${API_NAME}
[ $? == 0 ] || echo "Failed: AWS / apigateway /
create-rest-api"
awslocal apigateway get-rest-apis
```

Pregunta: Creación del API

1. ¿Qué ID se ha asignado al API?

El ID asignado al API es "t3uva9vz4b".

2. ¿Qué ocurre si volvemos a crear otro API con el mismo nombre?

LocalStack permite crear múltiples APIs con el mismo nombre. Cada vez que se crea un API, se asigna un ID único, independientemente del nombre del API. Aunque el nombre "user_lambda" se mantiene igual, el ID (en este caso "t3uva9vz4b") será único para esa instancia específica del API.

Tarea: Creación de las rutas

```
API_ID=$(awslocal apigateway get-rest-apis \
--query "items[?name=='${API_NAME}'].id" \
--output text --region ${REGION})
PATH_PART="user" ; PARENT_PATH="/"
PARENT_RESOURCE_ID=$(awslocal apigateway get-resources \
```

```
--rest-api-id ${API_ID} --query "items[?path=='${PARENT_PATH}'].id"
\
--output text --region ${REGION})
awslocal apigateway create-resource \
--region ${REGION} \
--rest-api-id ${API_ID} \
--parent-id ${PARENT_RESOURCE_ID} \
--path-part "${PATH_PART}"
PATH_PART="{username}" ; PARENT_PATH="/user"
PARENT_RESOURCE_ID=$(awslocal apigateway get-resources \
--rest-api-id ${API_ID} --query "items[?path=='${PARENT_PATH}'].id"
\
--output text --region ${REGION})
awslocal apigateway create-resource \
--region ${REGION} \
--rest-api-id ${API_ID} \
--parent-id ${PARENT_RESOURCE_ID} \
--path-part "${PATH_PART}"
[ $? == 0 ] || echo "Failed: AWS / apigateway / create-resource
{PATH_PART}"
awslocal apigateway get-resources --rest-api-id ${API_ID}
```

Pregunta: Creación de las rutas

1. ¿Qué ID se ha asignado al recurso con la ruta /user/{username}?

El ID asignado al recurso con la ruta /user/{username} es nd39xr5wms.

2. ¿Qué ocurre si creamos también la ruta /user/{name}?

Si se intenta crear la ruta /user/{name} mientras ya existe la ruta /user/{username}, API Gateway no permite la creación al tener dos parámetros de ruta con nombres diferentes en el mismo segmento de ruta.

3. En el caso de que se pueda crear a la vez la ruta /user/{name} y /user/{username}, ¿Tiene sentido?

No tiene sentido, desde el punto de vista del diseño de API, tener dos rutas como /user/{name} y /user/{username} en el mismo API ya que sería imposible determinar cuál de las dos rutas debería manejar una solicitud en particular.

Tarea: creación de métodos

```
PARAM=username ; METHOD_PATH="/user/{username}" ; HTTP_METHOD=POST
RESOURCE_ID=$(awslocal apigateway get-resources \
--rest-api-id ${API_ID} --query "items[?path=='${METHOD_PATH}'].id" \
\
--output text --region ${REGION})
awslocal apigateway put-method \
--region ${REGION} \
--rest-api-id ${API_ID} \
--resource-id ${RESOURCE_ID} \
--http-method ${HTTP_METHOD} \
--request-parameters "method.request.path.${PARAM}=true" \
--authorization-type "NONE"
[ $? == 0 ] || echo "Failed: AWS / apigateway / put-method
${METHOD_PATH}"
awslocal apigateway get-resources --rest-api-id ${API_ID}
```

Pregunta: Creación de métodos

1. Crea también el método GET asociado a la misma ruta, ¿Qué instrucciones has ejecutado para crearlo?

Para cambiar la variable HTTP_METHOD a "GET":

```
HTTP_METHOD=GET
```

Para añadir el método GET:

```
awslocal apigateway put-method \
--region ${REGION} \
--rest-api-id ${API_ID} \
--resource-id ${RESOURCE_ID} \
--http-method ${HTTP_METHOD} \
--request-parameters "method.request.path.${PARAM}=true" \
--authorization-type "NONE"
```

2. ¿Cómo podemos ver los métodos asociados a las rutas del API?

Con el siguiente comando podemos ver nuestra ruta indicada en path y los métodos asociados a ella.

```
awslocal apigateway get-resources --rest-api-id ${API_ID}
```

```
(base) carlos@carlos-IdeaPad-3-15ITL6:~$ awslocal apigateway get-resources --rest-api-id ${API_ID}
{
  "items": [
    {
      "id": "njgtavgcrs",
      "path": "/"
    },
    {
      "id": "f8kyoy1yif",
      "parentId": "njgtavgcrs",
      "pathPart": "user",
      "path": "/user"
    },
    {
      "id": "nd39xr5wms",
      "parentId": "f8kyoy1yif",
      "pathPart": "{username}",
      "path": "/user/{username}",
      "resourceMethods": {
        "POST": {
          "httpMethod": "POST",
          "authorizationType": "NONE",
          "apiKeyRequired": false,
          "requestParameters": {
            "method.request.path.username": true
          },
          "methodResponses": {}
        },
        "GET": {
          "httpMethod": "GET",
          "authorizationType": "NONE",
          "apiKeyRequired": false,
          "requestParameters": {
            "method.request.path.username": true
          },
          "methodResponses": {}
        }
      }
    }
  ]
}
```

Tarea: integración

```
HTTP_METHOD=POST ; LAMBDA_NAME=user_lambda
awslocal apigateway put-integration \
--region ${REGION} \
--rest-api-id ${API_ID} \
--resource-id ${RESOURCE_ID} \
--http-method ${HTTP_METHOD} \
--type AWS_PROXY \
--integration-http-method POST \
--uri arn:aws:apigateway:${REGION}:lambda:path/2015-03-
31/functions/${LAMBDA_NAME}/invocations \
--passthrough-behavior WHEN_NO_MATCH
[ $? == 0 ] || echo "Failed: AWS / apigateway /
put-integration"
awslocal apigateway get-resources --rest-api-id ${API_ID}
```

Pregunta: Integración

1. Integra también el método GET asociado a la misma ruta con la misma función lambda, ¿qué instrucciones has ejecutado para crearlo?

HTTP_METHOD=GET

```
awslocal apigateway put-integration \
--region ${REGION} \
--rest-api-id ${API_ID} \
--resource-id ${RESOURCE_ID} \
--http-method ${HTTP_METHOD} \
--type AWS_PROXY \
--integration-http-method POST \
--uri
arn:aws:apigateway:${REGION}:lambda:path/2015-03-31/functions/
${LAMBDA_NAME}/invocations \
--passthrough-behavior WHEN_NO_MATCH
```

2. ¿Cómo podemos ver si se ha integrado cada método del API?

```
awslocal apigateway get-integration --rest-api-id ${API_ID}
--resource-id ${RESOURCE_ID} --http-method ${HTTP_METHOD}
```

En este caso la integración es la de user_lambda_dynamo (la captura de este apartado se ha realizado en un orden distinto)

```
(slip1) (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/slip1$ awslocal apigateway get-integration --rest-api-id ${API_ID} --resource-id ${RESOURCE_ID} --http-method ${HTTP_METHOD}
{
  "type": "AWS_PROXY",
  "httpMethod": "POST",
  "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/user_lambda_dynamo/invocations",
  "passthroughBehavior": "WHEN_NO_MATCH",
  "timeoutInMillis": 29000,
  "cacheNamespace": "cfa6uinv8j",
  "cacheKeyParameters": []
}
```

3. Con ayuda de la documentación, por ejemplo ejecutando “awslocal apigateway help”, borra la integración del método GET, ¿qué instrucciones has ejecutado para lograrlo?

HTTP_METHOD=GET

```
awslocal apigateway delete-integration \
--region ${REGION} \
--rest-api-id ${API_ID} \
--resource-id ${RESOURCE_ID} \
--http-method ${HTTP_METHOD}
```

Tarea: despliegue

STAGE=dev

```
awslocal apigateway create-deployment \
--region ${REGION} \
--rest-api-id ${API_ID} \
--stage-name ${STAGE}
[ $? == 0 ] || echo " Failed: AWS / apigateway /
create-deployment"
awslocal apigateway get-deployments --rest-api-id ${API_ID}
awslocal apigateway get-stages --rest-api-id ${API_ID}
```

Pregunta: Despliegue

1. ¿Qué ocurre con la etapa “dev” (valor de STAGE) si repetimos el despliegue?

Se crea un nuevo despliegue y se asocia a la etapa "dev" siendo este nuevo despliegue el referenciado por la API.

```
(base) carlos@carlos-IdeaPad-3-15ITL6:~$ awslocal apigateway get-deployments --rest-api-id ${API_ID}
{
  "items": [
    {
      "id": "nk2uf15st4",
      "createdDate": 1697707616.0
    },
    {
      "id": "3rzwcxixkg",
      "createdDate": 1697707960.0
    }
  ]
}
(base) carlos@carlos-IdeaPad-3-15ITL6:~$ awslocal apigateway get-stages --rest-api-id ${API_ID}
{
  "item": [
    {
      "deploymentId": "3rzwcxixkg",
      "stageName": "dev",
      "cacheClusterEnabled": false,
      "cacheClusterStatus": "NOT_AVAILABLE",
      "methodSettings": {},
      "tracingEnabled": false
    }
  ]
}
```

2. Con ayuda de la documentación, por ejemplo ejecutando “awslocal apigateway help”, borra el despliegue que ya no está asociado a la etapa “dev”. ¿Qué instrucciones has ejecutado para lograrlo?

```
awslocal apigateway delete-deployment --rest-api-id ${API_ID}
--deployment-id nk2uf15st4
```

Tarea: Pruebas del API

En esta ocasión, se prueba nuestro API usando la siguiente URL como base:


```
http://localhost:4566/restapis/${API_ID}/${STAGE}/_user_request_/
STAGE=dev
ENDPOINT=http://localhost:4566/restapis/${API_ID}/${STAGE}/_user_request_/user
curl -X POST ${ENDPOINT}/pepe \
-H 'Content-Type: application/json' \
-d '{"username": "tete", "data": "100"}'
```

Pregunta: Pruebas del API

1. Usando los comandos de S3, lista el contenido del bucket y descarga el archivo que se ha creado para comprobar su contenido. ¿Qué ha ocurrido? ¿Ha cambiado el valor del fichero /users/pepe o se ha creado un archivo distinto?

```
(base) carlos@carlos-IdeaPad-3-15ITL6: ~/venv/slip1$ curl -X POST ${ENDPOINT}/pepe -H 'Content-Type: application/json' -d '{"username": "tete", "data": "100"}'
{"test_object_key" placed into S3}(base) carlos@car
(base) carlos@carlos-IdeaPad-3-15ITL6: ~/venv/slip1$ awslocal s3 ls s3://slip1/users/
2023-10-19 11:59:07          34 pepe
2023-10-19 12:24:04          35 tete
(base) carlos@carlos-IdeaPad-3-15ITL6: ~/venv/slip1$
```

Se crea un archivo distinto dentro de /users.

Tarea: mejora API

Para esta tarea, se copia user_lambda.py en user_lambda_param.py, y modifica el código para, de forma similar a lo que hacíamos en user_rest.py, crear en S3 un objeto con los datos pasados en event.body y guardarlos en el bucket S3 en la ruta /user/{username}/data.json

Para obtener el parámetro username del evento, usa el siguiente código:

```
username=event['pathParameters']['username']
```

Pregunta: Mejora del API

¿Qué pasos han sido necesarios para usar la nueva lambda en lugar de la anterior? Indica los comandos que has empleado

En si1p1/src: cp user_lambda.py user_lambda_param.py

Se empaqueta la función: zip ../build/user_lambda_param.zip user_lambda_param.py

Y creamos la nueva función lambda (lambda_param):

```
awslocal lambda create-function \
--function-name user_lambda_param \
--runtime python3.9 \
--zip-file fileb://build/user_lambda_param.zip \
--handler user_lambda_param.handler \
--role arn:aws:iam::000000000000:role/si1
```

Para emplear `user_lambda_param` realizamos los mismos pasos que anteriormente para `user_lambda` de integrar y desplegar.

6. DynamoDB

Tarea: Creación de una tabla

```
awslocal dynamodb create-table \
--table-name si1p1 \
--attribute-definitions AttributeName=User,AttributeType=S \
--key-schema AttributeName=User,KeyType=HASH \
--region $REGION \
--provisioned-throughput
ReadCapacityUnits=5,WriteCapacityUnits=5
```

Pregunta: Creación de una tabla

¿De qué tipo son los distintos campos de la tabla?

User: Tipo: String (S en la definición `AttributeType=S`)

Tarea: Insertando datos manualmente

Insertamos los datos indicando su tipo, así queda en la terminal:

```
awslocal dynamodb put-item \
--table-name si1p1 \
--item '{"User": {"S": "pipo"}, "EMail": {"S": "pipo@ss.com"}}'
awslocal dynamodb scan --table-name si1p1 --region us-east-1
```

Pregunta: Insertando datos manualmente

1. ¿Qué ocurre si creamos un ítem sin campo “User” o sin campo “Email”?

Si se intenta crear un ítem sin el campo User en DynamoDB, se produce un error, esto pasa porque User es la clave que se define cuando se crea la tabla, y es obligatoria para cada ítem que se inserte en la tabla.

En el caso del campo EMail, no es obligatorio incluirlo al crear un ítem porque no es una clave para las tablas, así que no hay ningún error y si se omite DynamoDB simplemente crea el ítem con los campos proporcionados.

```
(si1p1) (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal dynamodb put-item \
--table-name si1p1 \
--item '{"EMail": {"S": "pipo@ss.com"}}'

An error occurred (ValidationException) when calling the PutItem operation: One of the required keys was not given a value
(si1p1) (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal dynamodb put-item \
--table-name si1p1 \
--item '{"User": {"S": "pipo"}}'
(si1p1) (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$
```

Tarea: Lectura de datos

```
awslocal dynamodb get-item \
--table-name si1p1 \
--key '{"User": {"S": "pipo"}, "Email": {"S": "pipo@ss.com"}}'
awslocal dynamodb scan --table-name si1p1 --region us-east-1
```

Pregunta: Lectura de datos

1. ¿Qué ocurre si creamos un ítem sin campo “User”?

El campo User está definido como la clave principal en el esquema de la tabla si1p1 así que si se intenta crear un ítem sin el campo User, DynamoDB devuelve un error ya que no se puede omitir el valor de la clave principal cuando se inserta un ítem en la tabla.

En nuestro caso, nos ha devuelto este error de ejecución:

An error occurred (ValidationException) when calling the GetItem operation: The number of conditions on the keys is invalid.

2. ¿Qué ocurre si volvemos a insertar el mismo usuario sin el campo “Email”?

Si intentamos insertar el mismo valor de User sin el campo Email, DynamoDB no da ningún error pero el nuevo ítem se sobrescribe al ítem existente que tiene el mismo valor de clave principal.

```
{si1p1} (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal dynamodb get-item --table-name si1p1 --key '{"User": {"S": "pipo"}}'
{
  "Item": {
    "EMail": {
      "S": "pipo@ss.com"
    },
    "User": {
      "S": "pipo"
    }
  }
}
```

Tarea: Lambda con dynamoDB

Se ha creado la lambda de user_lambda_dynamo.py e integrado en el método POST de user/{username}, donde antes teníamos user_lambda_param.py.

Tras ello, se ejecuta en un terminal:

```
curl -X POST ${ENDPOINT}/user2 -H 'Content-Type: application/json' \
-d '{"email": "mimail@dominio.es"}'
```

Pregunta: Insertando datos manualmente

1. ¿Qué pasos han sido necesarios para cambiar de función lambda?

Empaquetar y crear la función lambda con user_lambda_dynamo.py como se ha hecho en tareas anteriores, así mismo como en anteriores tareas se ha

integrado (con HTTP_METHOD=POST) la nueva función lambda con el comando `awslocal apigateway put-integration` y se han desplegado los cambios en el API Gateway usando el comando `awslocal apigateway create-deployment`.

2. ¿Qué datos se han añadido a la tabla `si1p1` tras ejecutar el comando `curl` anterior?

Se añaden correctamente el email y el usuario.

```
(si1p1) (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ curl -X POST "${ENDPOINT}/user2" -H 'Content-Type: application/json' \
-d '{"email": "minail@dominio.es"}'
{"msg": "user2 placed into dynamo"}(si1p1) (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal dynamodb scan --table-name si1p1 --region us-east-1
{
  "Items": [
    {
      "EMail": {
        "S": "minail@dominio.es"
      },
      "User": {
        "S": "user2"
      }
    },
    {
      "EMail": {
        "S": "pipo@ss.com"
      },
      "User": {
        "S": "pipo"
      }
    }
  ],
  "Count": 2,
  "ScannedCount": 2,
  "ConsumedCapacity": null
}
```

Tarea: GET /user/{username} con dynamoDB

En esta tarea, se crea e integra `user_get_lambda_dynamo.py` integrándola en el método GET de `user/{username}`, que debe buscar el usuario y devolver el objeto encontrado en el cuerpo de la respuesta.

Pregunta: GET /user/{username} con dynamoDB

1. ¿Qué pasos han sido necesarios para crear e integrar la nueva función lambda?

Se ha implementado como en los ejercicios con la excepción de que en método y en la integración se ha sustituido `HTTP_METHOD=POST` por `GET`, para que en la ruta `/users/{username}` maneje interacciones GET. Alternativamente podríamos haber asociado el método GET a la ruta que ya tenía asociado POST.

2. ¿Qué comando `curl` podemos usar para probar el método GET `/user/{username}`?. Da ejemplos también con la respuesta tanto si existe como si no existe el usuario.

`curl -X GET "${ENDPOINT}/{username}"`, donde `ENDPOINT` es `http://localhost:4566/restapis/${API_ID}/${STAGE}/_user_request/_user`

```
(si1p1) (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ awslocal dynamodb scan --table-name si1p1 --region us-east-1
{
  "Items": [
    {
      "EMail": {
        "S": "pipo@ss.com"
      },
      "User": {
        "S": "pipo"
      }
    }
  ],
  "Count": 1,
  "ScannedCount": 1,
  "ConsumedCapacity": null
}
(si1p1) (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ curl -X GET "${ENDPOINT}/{username}"
{}(si1p1) (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$ curl -X GET "${ENDPOINT}/{pipo}"
{"EMail": "pipo@ss.com", "User": "pipo"}(si1p1) (base) carlos@carlos-IdeaPad-3-15ITL6:~/venv/si1p1$
```