



Universidad Autónoma de Madrid

Escuela Politécnica Superior

SISTEMAS INFORMÁTICOS I

Práctica 1

Iker González Sánchez

Gonzalo Segovia Oblanca

Grupo 1332 Pareja 8

Octubre de 2023

Parte 1

Pregunta: user_rest.py

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

Cada una de estas anotaciones viene acompañada de una función asíncrona que se ejecuta.

@app.route es un decorador que le dice al programa la ruta que debería activar nuestra función, en este caso la función “hello()”. El primer argumento (`/`) especifica la ruta y el segundo (`methods=["GET"]`) indica los métodos HTTP que activa la función, en este caso GET, que es el método por defecto.

@app.get es otro decorador que asocia una url con una función que se encargará de manejar las peticiones del método HTTP GET.

@app.put, de manera idéntica a @app.get, asocia una URL a una función que se encargará de manejar las peticiones del método HTTP PUT.

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

Estos decoradores hacen el código más legible y permiten modificar el comportamiento de algunas funciones sin cambiar el código fuente del programa. Sin embargo, esta última ventaja puede considerarse como un inconveniente, ya que esto puede complicar los procesos de “debug”.

Tarea: ejecución de user_rest.py

Ejecuta en un terminal:

```
$> source venv/si1p1/bin/activate \
$> [[ -d build/users ]] || mkdir -p build/users \
$> QUART_APP=src.user_rest:app quart run -p 5050
```

Nota: hay que corregir el tipo de guión usado en el tercer comando.

Pregunta: ejecución de user_rest.py

¿Qué crees que ha sucedido?

El primero activa el entorno virtual situado en el directorio venv/si1p1. El segundo comprueba que el directorio build/users exista, y si no lo crea. Por último, el tercer comando establece la variable de entorno QUART_APP en src.user_rest:app, lo que indica a Quart qué módulo y objeto usar como aplicación. Luego, ejecuta la aplicación usando el comando quart run y especifica que debe escuchar en el puerto 5050. Es decir, este comando inicia la aplicación web en el puerto 5050.

Tarea: petición a user_rest.py

Ejecuta en otro terminal

```
$> curl -X PUT \
http://localhost:5050/user/myusername \
-H 'Content-Type: application/json' \
-d '{"firstName": "myFirstName"}'
```

Pregunta: petición a user_rest.py

Revisa la ayuda de curl (man curl) y contesta:

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

El comando *curl -X* se usa para especificar el método HTTP a usar para hacer una petición al servidor. En este caso, se usa el método PUT en el puerto 5050 de nuestro host local.

Por otro lado, *curl -H* añade una cabecera personalizada a la petición HTTP. En este caso, especifica el tipo de contenido que maneja (*application/json* en este caso).

Por último, *curl -d* se utiliza para enviar datos en el cuerpo de la petición HTTP.

En conclusión y atendiendo a la implementación de nuestro método PUT, creemos que este comando escribe `{"firstName": "myFirstName"}` en la cabecera del archivo *data.json* del directorio creado en la tarea anterior.

2. ¿Cuál es la respuesta del microservicio?

La respuesta es `{"status": "OK"}`, denotando de este modo que el microservicio se ha ejecutado correctamente.

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

Sí. Se ha creado un archivo *data.json* en el directorio creado en la tarea anterior (`~/p1-v1.0/build/users/gonzalo/`). En la cabecera del archivo escribe `{"firstName": "gonzalo", "username": "gonzalo"}`.

Tarea: petición HTTP

Revisa la ayuda de nc (man nc) y ejecuta en otro terminal este programa en modo escucha en el puerto 8888.

Ejecuta de nuevo curl, pero haciendo la petición al puerto 8888

Pregunta: petición HTTP

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

```
PUT /user/myusername HTTP/1.1
Host: localhost:8888
User-Agent: curl/7.82.0
Accept: /*
Content-Type: application/json
Content-Length: 28
```

```
{"firstName": "myFirstName"}
```

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 tipo de dato del cuerpo de la petición se indica en el campo *Content-Type* de la cabecera HTTP. En esta solicitud, el *Content-Type* es *application/json*, lo que significa que el cuerpo de la solicitud contiene datos en formato *JSON*.

3. ¿Qué separa la cabecera del cuerpo?

La cabecera del cuerpo de una solicitud HTTP se separa mediante una línea en blanco. En nuestro caso, esto es dos saltos de línea después de “Content-Length: 28”.

Tarea: prototipo user

Completa el micro-servicio user:

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

A continuación se muestra el código con la explicación del mismo comentada.

```
"""
Elimina un usuario dado su nombre
1. Intenta borrar el directorio del usuario
2. Si se produce una excepción, devuelve un error
3. Si no se produce una excepción, devuelve un OK
"""

@app.delete('/user/<username>')
async def user_delete(username):
    resp={}
    try:
        os.rmdir(f"{USERDIR}/{username}")
    except Exception as e:
        resp["status"] = "KO"
        resp["error"] = f"Error borrando user dir: {str(e)}"
    return await make_response(jsonify(resp), 400)
    resp["status"]="OK"
    return await make_response(jsonify(resp), 200)
```

```

"""
Actualiza un usuario dado su nombre
1. Lee el fichero de datos del usuario
2. Actualiza los datos con los nuevos datos
3. Escribe el fichero de datos
4. Devuelve un OK
"""

@app.patch('/user/<username>')
async def user_patch(username):
    resp={}
    data=await request.get_json()
    with open(f"{USERDIR}/{username}/data.json", "r") as ff:
        user_data=json.loads(ff.read())
    user_data.update(data)
    with open(f"{USERDIR}/{username}/data.json", "w") as ff:
        ff.write(json.dumps(user_data))
    resp["status"]="OK"
    return await make_response(jsonify(resp), 200)

```

Tarea: ejecución hypercorn

Ejecuta en un terminal:

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

Pregunta: ejecución hypercorn

1. ¿Qué crees que ha sucedido?

Este comando ha enlazado a la IPv4 *0.0.0.0:8080* nuestra app *user_rest*. Con *--workers 2*, especificamos el número de procesos “workers” a invocar. Cada uno de estos “workers” es un proceso que maneja las peticiones HTTP que van entrando. Estos procesos los podemos ver en la consola después de ejecutar el comando.

```
(si1p1) (base) gonzalo@gonzalo:~/SII_PR/p1-v1.0$ hypercorn --bind 0.0.0.0:8080 --workers 2 src.user_rest:app
[2023-10-06 12:04:54 +0200] [9513] [INFO] Running on http://0.0.0.0:8080 (CTRL + C to quit) data_username
[2023-10-06 12:04:54 +0200] [9512] [INFO] Running on http://0.0.0.0:8080 (CTRL + C to quit) with open (f"
```

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

Ahora tenemos que usar *curl -X GET <http://0.0.0.0:8080>*.

Usando Hypercorn recibo “hello” en lugar de `{"firstName":"gonzalo","username":"gonzalo"}`. Esto seguramente se deba al enlace del servidor [*http://0.0.0.0:8080*](http://0.0.0.0:8080) con nuestro código de *user_rest*.

Contenedores

Tarea: arranque de docker rootless

Ejecuta en un terminal:

```
# doc de configuración de usuario necesaria
$> man subuid
# EPS labs: establece configuración de usuario
$> sudo /usr/local/bin/si2setuid.sh 3000000:100000 $USER
# HOME: establece configuración de usuario
# - edita como sudo los archivos /etc/subuid y /etc/subgid e introduce entradas
#   semejantes a las de los labs, con el nombre de tu usuario en tu PC
# sudo vim /etc/subuid
# sudo vim /etc/subgid
# arranque del demonio
$> dockerd-rootless-setuptool.sh install
# usar docker propio en lugar del docker del sistema
$> DOCKER_HOST=unix:///run/user/$(id -u)/docker.sock
# comprobación de estado del demonio
$> systemctl --user status docker
```

Pregunta: modo rootless

Revisa el contenido de /etc/subuid y la documentación de docker rootless: ¿para qué crees que sirve ese archivo?

El archivo contiene nombres de usuarios con rangos de UIDs. Cuando Docker se ejecuta en modo "rootless", se le asignan un rango de UIDs secundarios que pueden ser utilizados por los contenedores que se ejecutan bajo ese usuario sin necesidad de tener privilegios de superusuario.

Tarea: docker info

Ejecuta en un terminal:

```
$> docker info
```

Pregunta: docker info

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

La configuración del demonio arrancado se guarda en el archivo *daemon.json* que se encuentra dentro del directorio *.docker/* del directorio de nuestro usuario. La configuración inicial es la siguiente:

```
GNU nano 6.2                                     daemon.json
{
  "builder": {
    "gc": {
      "defaultKeepStorage": "20GB",
      "enabled": true
    }
  },
  "experimental": false
}
```

2. ¿Dónde se almacenan los contenedores?

Se almacenan en el directorio raíz de Docker (`/var/lib/docker`, como se puede observar en la siguiente imagen).

```
containerd version: 8165feabfdfe38c65b599c4993d227328c231fca
runc version: v1.1.8-0-g82f18fe
init version: de40ad0
Security Options:
  seccomp
    Profile: unconfined
  cgroupns
Kernel Version: 6.4.16-linuxkit
Operating System: Docker Desktop
OSType: linux
Architecture: x86_64
CPUs: 16
Total Memory: 7.405GiB
Name: docker-desktop
ID: 49750399-be76-4e48-bfa2-24f0a6ea746c
Docker Root Dir: /var/lib/docker
Debug Mode: false
HTTP Proxy: http.docker.internal:3128
HTTPS Proxy: https.docker.internal:3128
No Proxy: hubproxy.docker.internal
Experimental: false
Insecure Registries:
  hubproxy.docker.internal:5555
  127.0.0.0/8
Live Restore Enabled: false

WARNING: daemon is not using the default seccomp profile
```

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

Según el apartado de almacenamiento de la [documentación](#) de Docker, se usan tres tecnologías de almacenamiento:

- Volumenes: creados y gestionados por Docker. Se almacenan en una parte del sistema de archivos del host que es controlada por Docker y cuentan con persistencia de datos.
- Bind mounts: permiten montar un archivo o directorio del host en un contenedor. No son gestionados por Docker y pueden estar en cualquier parte del sistema de archivos del host. Tienen un alto rendimiento, pero dependen de la estructura de directorios del host.
- Tmpfs mounts: almacenan los archivos en la memoria del host, sin persistirlos en el disco. Se usan para guardar datos no persistentes o sensibles.

Por otro lado, copy-on-write es una estrategia de compartir y copiar archivos para obtener la máxima eficiencia. Si un archivo o directorio existe en una capa inferior dentro de la imagen, y otra capa necesita acceder a su lectura, simplemente utiliza el archivo existente. La primera vez que otra capa necesita modificar el archivo, el archivo se copia en esa capa y se modifica. Esto minimiza el I/O (entrada y salida) y el tamaño de cada una de las capas posteriores. Debido a estas ventajas, el principio CoW es usado por Docker como se puede observar en su [documentación](#).

Tarea: run whoami

Ejecuta en un terminal:

```
$> docker run alpine whoami
```

```
gonzalo@gonzalo:~$ docker run alpine whoami
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
96526aa774ef: Pull complete
Digest: sha256:eece025e432126ce23f223450a0326fbbebe39cdf496a85d8c016293fc851978
Status: Downloaded newer image for alpine:latest
root
```

Pregunta: run whoami

1. ¿Qué crees que ha sucedido?

Este comando ha creado, como se puede observar en Docker Desktop, un contenedor basado en la imagen Alpine.

Como se puede observar en la salida del terminal, en primer lugar ha intentado encontrar la imagen alpine localmente y, al no poder, la ha descargado del repositorio *library*. Al terminar de descargarla, podemos observar en la tercera línea el ID de la capa de la imagen que ha sido descargada. Las dos siguientes líneas son confirmaciones de que la imagen ha sido descargada correctamente sin errores. Finalmente, escribe “root”, que es la salida del comando que se ha ejecutado dentro del contenedor, que muestra que el usuario que está ejecutando el proceso dentro del contenedor es el usuario root.

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

No, solo es el usuario root dentro del contenedor, por lo que no tiene privilegios fuera de este.

Tarea: list images

Ejecuta en un terminal:

```
$> docker images
y revisa su resultado.
```

```
gonzalo@gonzalo:~$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
alpine          latest   8ca4688f4f35  2 weeks ago  7.34MB
```

Pregunta: list images

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

docker pull sirve para descargar o actualizar imágenes desde una fuente remota y *docker images* nos muestra las imágenes almacenadas localmente. Estos dos comandos se podrían relacionar en el sentido de que usando *docker images* vemos el resultado de las iteraciones de *docker pull*. Asimismo, podemos comprobar con *docker images* la versión de la imagen y actualizarlas en consecuencia.

Tarea: docker ps

Ejecuta en un terminal:

```
$> docker ps -a
```

```
gonzalo@gonzalo:~$ docker ps -a
CONTAINER ID        IMAGE       COMMAND     CREATED      STATUS      PORTS     NAMES
ea5f90d9383c        alpine     "whoami"    3 days ago   Exited (0)  3 days ago   reverent_chandrasekhar
```

Pregunta: docker ps

¿Qué crees que muestra la salida?

Este comando nos muestra un listado de todos (flag -a) los contenedores en ejecución, así como información básica de estos.

Tarea: docker pull y run -ti

Ejecuta en un terminal:

```
$> docker pull ubuntu:22.04
$> docker run -ti --name u22.04 ubuntu
```

```
gonzalo@gonzalo:~$ docker pull ubuntu:22.04
22.04: Pulling from library/ubuntu
aece8493d397: Pull complete
Digest: sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f
Status: Downloaded newer image for ubuntu:22.04
docker.io/library/ubuntu:22.04

What's Next?
1. Sign in to your Docker account → docker login
2. View a summary of image vulnerabilities and recommendations → docker scout quickview ubuntu:22.04
gonzalo@gonzalo:~$ docker run -ti --name u22.04 ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
Digest: sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f
Status: Downloaded newer image for ubuntu:latest
root@8a632cb7ffed:/#
```

Pregunta: docker pull y run -ti

1. ¿Qué hacen las opciones -ti?

Esta flag es una combinación de las flags -t y -i. La primera crea una terminal para el contenedor y -i mantiene la entrada estándar de la terminal abierta. En resumen, esta flag permite al usuario interactuar con la consola del contenedor.

2. ¿Qué ha sucedido?

Se ha descargado una imagen de Ubuntu 22.04 y después se ha ejecutado un contenedor (llamado u22.04 con la flag –name) usando esta imagen, creando con las flags provistas una terminal dentro de nuestra consola-

Tarea: docker inspect

Revisa la ayuda del mandato jq (man jq).

Ejecuta en un terminal:

```
$> docker inspect u22.04 | jq '.[].Config.Cmd[]'
```

```
gonzalo@gonzalo:~$ docker inspect u22.04 | jq '.[].Config.Cmd[]'
"/bin/bash"
```

Pregunta: docker inspect

1. ¿Qué hace el mandato jq?

Según el manual de este comando, sirve para procesar, manipular, filtrar y extraer datos en formato JSON.

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

El resultado del mandato es la localización del fichero *Config.Cmd* de nuestro contenedor *u22.04* y, al estar la salida del comando *docker inspect* en formato JSON, podemos filtrar con jq.

Tarea: docker start

Ejecuta en un terminal:

```
$> docker start u22.04
```

```
gonzalo@gonzalo:~$ docker start u22.04
u22.04
```

Pregunta: docker start

1. ¿Qué ha sucedido?

Este comando ha arrancado el contenedor *u22.04* (que estaba parado antes).

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

Esto se hace mediante el comando *docker rm –force u22.04*. Tenemos que usar la flag –force ya que el contenedor está arrancado.

Tarea: docker run -d

Ejecuta en un terminal:

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

```
gonzalo@gonzalo:~$ docker run -ti -d --name u22.04d ubuntu
c01ae98eec6a4f62774b60a8417607d6d5d144db93204ce0f922d97e7b9926ee
```

Pregunta: docker run -d

¿Qué hace la opción -d?

Esta opción hace que el contenedor se ejecute en segundo plano y muestra el ID del contenedor.

Tarea: docker exec

Ejecuta en un terminal:

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

y dentro del contenedor:

```
$> mkdir -p /si1/users
$> touch /si1/users/test
```

```
gonzalo@gonzalo:~$ docker exec -ti u22.04d /bin/bash
root@c01ae98eec6a:# mkdir -p /si1/users
root@c01ae98eec6a:# touch /si1/users/test
root@c01ae98eec6a:# ls
bin  dev  home  lib32  libx32  mnt  proc  run  si1  sys  usr
boot etc  lib   lib64  media   opt  root  sbin  srv  tmp  var
root@c01ae98eec6a:# cd si1
```

Pregunta: docker exec

¿Qué ha sucedido?

El primer comando ha abierto la terminal en el contenedor *u22.04d* (que ya está en ejecución), con */bin/bash* hemos ejecutado la shell de bash en esta terminal.

Posteriormente, hemos creado el directorio `/si1/users` y, dentro de este, un fichero `test`.

Tarea: docker volume

Ejecuta en un terminal:

```
$> [[ -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
```

```
gonzalo@gonzalo:~$ docker run -ti -v ${PWD}/si1/users/testv:/si1/users/testv alpine \
    touch /si1/users/testv/afile
gonzalo@gonzalo:~$ docker run -ti -v ${PWD}/si1/users/testv:/si1/users/testv alpine \
    ls -al /si1/users/testv/afile
-rw-r--r-- 1 root      root          0 Oct 18 17:49 /si1/users/testv/afile
gonzalo@gonzalo:~$ ls -al si1/users/testv/afile
-rw-r--r-- 1 gonzalo gonzalo 0 oct 18 19:49 si1/users/testv/afile
gonzalo@gonzalo:~$ echo "a test" >>si1/users/testv/afile
gonzalo@gonzalo:~$ docker run -ti -v ${PWD}/si1/users/testv:/si1/users/testv alpine \
    cat /si1/users/testv/afile
a test
```

Pregunta: docker volume

1. ¿Qué hace la opción `-v`?

Esta flag, según la documentación de Docker, hace que monte un volumen enlazado desde el directorio del host hasta el contenedor. Esto quiere decir que enlaza un directorio en el host con otro en el contenedor, por lo que si se producen cambios en alguno de estos dos directorios, se verán reflejados en el otro.

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

El primer comando comprueba si existe el directorio `si1/users/testv` y, si no, lo crea. El segundo comando ejecuta un contenedor con una terminal abierta y monta un volumen enlazado desde el directorio del host `${PWD}/si1/users/testv` hasta el directorio del contenedor `/si1/users/testv` con la imagen `alpine`. En esa terminal (la del contenedor) crea el fichero `afile`.

El tercero comando muestra, en el directorio host, los contenidos del directorio enlazado al contenedor, demostrando que el fichero `afile` se ha creado también en el host.

El cuarto comando, usando la terminal de nuestro host, escribe la cadena “`a test`” dentro del fichero `afile`.

El quinto comando, ejecuta de nuevo el contenedor con la terminal abierta y muestra con el sucesivo comando `cat` el contenido del fichero `afile`, corroborando una vez más que los cambios en los contenidos de uno se realizan asimismo en el otro.

Tarea: limpieza

Revisa la documentación y elimina todos los contenedores e imágenes empleados.

Pregunta: limpieza

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

```
gonzalo@gonzalo:~$ docker rm $(docker ps --filter status=exited -q)
292a5790fd89
df6ffd458744
bbaf5e4cba0a
c01ae98eec6a
8a632cb7ffed
ea5f90d9383c
```

Al estar en nuestro caso todos los contenedores parados, por economía del esfuerzo, he usado el mandato especificado en la documentación de docker que elimina todos los contenedores que pasen el filtro de *status=exited* (esto es, parados).

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

```
gonzalo@gonzalo:~$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
ubuntu          22.04    e4c58958181a  13 days ago   77.8MB
ubuntu          latest    e4c58958181a  13 days ago   77.8MB
alpine          latest    8ca4688f4f35   2 weeks ago   7.34MB
gonzalo@gonzalo:~$ docker rmi ubuntu:22.04
Untagged: ubuntu:22.04
gonzalo@gonzalo:~$ docker rmi ubuntu:latest
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f
Deleted: sha256:e4c58958181a5925816faa528ce959e487632f4cf192f8132f71b32df2744b4
Deleted: sha256:256d88da41857db513b95b50ba9a9b28491b58c954e25477d5dad8abb465430b
gonzalo@gonzalo:~$ docker rmi alpine:latest
Untagged: alpine:latest
Untagged: alpine@sha256:eece025e432126ce23f223450a0326fbebe39cdf496a85d8c016293fc851978
Deleted: sha256:8ca4688f4f356596b5ae539337c9941abc78eda10021d35cbc52659c74d9b443
Deleted: sha256:cc2447e1835a40530975ab80bb1f872fbab0f2a0faecf2ab16fb89b3589438
```

Para las imágenes, siguiendo también la documentación de Docker, he mostrado todas las imágenes y después las he ido eliminando manualmente con el comando *docker rmi REPO:TAG*.

Tarea: Dockerfile

Revisa la documentación sobre el Dockerfile en la bibliografía y el archivo src/Dockerfile suministrado.

```

FROM ubuntu

ENV NUMWORKERS 2

RUN apt update
RUN apt install -y python3-pip
RUN pip install hypercorn quart

RUN mkdir -p /sil/build/users

EXPOSE 8000

#RUN useradd sil
#RUN chown -R sil /sil
#USER sil

WORKDIR /sil

COPY user_rest.py .

CMD hypercorn --bind 0.0.0.0:8000 --workers ${NUMWORKERS} user_rest:app

```

Pregunta: Dockerfile

1. ¿Qué hace la sentencia FROM?

Los Dockerfile siempre empiezan con ella. Inicializa una nueva etapa de construcción de la imagen y establece la imagen base para las instrucciones subsecuentes, en nuestro caso Ubuntu.

2. ¿Qué hace la sentencia ENV?

ENV <key>=<value> ... establece un valor <value> la variable de entorno <key>. En nuestro caso ponemos a 2 la variable de entorno *NUMWORKERS*.

3. ¿Qué diferencia presenta ENV respecto de ARG?

A diferencia de ARG, con ENV los valores de la variable global persisten en la imagen final.

4. ¿Qué hace la sentencia RUN?

RUN <command> ejecuta un comando en una nueva capa sobre la imagen actual.

5. ¿Qué hace la sentencia COPY?

La sentencia COPY copia archivos o directorios (primer argumento) al directorio especificado en el segundo argumento.

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

La sentencia ADD es similar a COPY en el sentido de que ambas permiten añadir contenido a la imagen del contenedor. La función ADD tiene más funciones que COPY, como trabajar con archivos comprimidos y URLs remotas.

7. ¿Qué hace la sentencia EXPOSE?

EXPOSE <port> [<port>/<protocol>...] informa a Docker que el contenedor escucha en los puertos especificados. En nuestro caso el 8000.

8. ¿Qué hace la sentencia CMD?

CMD command param0 paramN especifica un comando por defecto que se ejecuta al iniciar el contenedor. También se puede usar esta sentencia sin especificar el comando, en cuyo caso se debe inicializar la variable ENTRYPPOINT de antemano.

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

La sentencia RUN es similar en tanto que se ejecutan comandos en el proceso de construcción. Sin embargo, CMD los ejecuta al inicio de la construcción de la imagen mientras que RUN los ejecuta durante la construcción.

Tarea: docker build

Ejecuta en un terminal

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

```
gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ cd src/
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker build --tag si1p1:latest .
[+] Building 43.6s (12/12) FINISHED                                            docker:desktop-linux
=> [internal] load build definition from Dockerfile                         0.0s
=> => transferring dockerfile: 353B                                         0.0s
=> [internal] load .dockerignore                                           0.0s
=> => transferring context: 2B                                         0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest           1.8s
=> [1/7] FROM docker.io/library/ubuntu@sha256:2b7412e6465c3c7fc5bb21d3e6f1917c16 4.6s
=> => resolve docker.io/library/ubuntu@sha256:2b7412e6465c3c7fc5bb21d3e6f1917c16 0.0s
=> => sha256:aece8493d3972efa43bfd4ee3cdba659c0f787f8f59c82fb3 29.54MB / 29.54MB 4.0s
=> => sha256:2b7412e6465c3c7fc5bb21d3e6f1917c163c167358449fecac8176c 1.13kB / 1.13kB 0.0s
=> => sha256:c9cf959fd83770dfdefd8fb42cefef0761432af36a764c077aed54bb 424B / 424B 0.0s
=> => sha256:e4c58958181a5925816faa528ce959e487632f4cf192f8132f 2.30kB / 2.30kB 0.0s
=> => extracting sha256:aece8493d3972efa43bfd4ee3cdba659c0f787f8f59c82fb3e48c87c 0.5s
=> [internal] load build context                                           0.0s
=> => transferring context: 1.87kB                                         0.0s
=> [2/7] RUN apt update                                                 4.7s
=> [3/7] RUN apt install -y python3-pip                                27.2s
=> [4/7] RUN pip install hypercorn quart                               3.5s
=> [5/7] RUN mkdir -p /si1/build/users                                 0.3s
=> [6/7] WORKDIR /si1                                                0.0s
=> [7/7] COPY user_rest.py ./                                         0.0s
=> exporting to image                                                 1.4s
=> => exporting layers                                              1.4s
=> => writing image sha256:584278ae82f9f684a8e3a8110da2fba19b1a9ad7c25cc3c152c41 0.0s
=> => naming to docker.io/library/si1p1:latest                      0.0s
```

```
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker build --tag si1p1:1.0 .
[+] Building 0.6s (12/12) FINISHED                                            docker:desktop-linux
=> [internal] load .dockerignore                                         0.0s
=> => transferring context: 2B                                           0.0s
=> [internal] load build definition from Dockerfile                      0.0s
=> => transferring dockerfile: 353B                                         0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest           0.5s
=> [1/7] FROM docker.io/library/ubuntu@sha256:2b7412e6465c3c7fc5bb21d3e6f1917c16 0.0s
=> [internal] load build context                                         0.0s
=> => transferring context: 34B                                           0.0s
=> CACHED [2/7] RUN apt update                                         0.0s
=> CACHED [3/7] RUN apt install -y python3-pip                         0.0s
=> CACHED [4/7] RUN pip install hypercorn quart                         0.0s
=> CACHED [5/7] RUN mkdir -p /si1/build/users                           0.0s
=> CACHED [6/7] WORKDIR /si1                                           0.0s
=> CACHED [7/7] COPY user_rest.py ./                                       0.0s
=> exporting to image                                                 0.0s
=> => exporting layers                                              0.0s
=> => writing image sha256:584278ae82f9f684a8e3a8110da2fba19b1a9ad7c25cc3c152c41 0.0s
=> => naming to docker.io/library/si1p1:1.0                           0.0s
```

```
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
si1p1           1.0      584278ae82f9  43 seconds ago  478MB
si1p1           latest   584278ae82f9  43 seconds ago  478MB
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ cd -
/home/gonzalo/SI1_PR/p1-v1.0
```

Pregunta: docker build

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

Porque la segunda imagen tiene el mismo tag que la primera y Docker almacena las capas de la imagen que ha creado antes en caché (nótese la señal CACHED delante de las instrucciones del Dockerfile de la segunda imagen) y las reutiliza en lugar de ejecutar los comandos del dockerfile de nuevo.

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

Las capas son cada una de las instrucciones del Dockerfile.

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.

```
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker build --tag si1p1:1.0u .
[+] Building 1.8s (14/14) FINISHED                                            docker:desktop-linux
=> [internal] load build definition from Dockerfile                          0.0s
=> => transferring dockerfile: 350B                                         0.0s
=> [internal] load .dockerignore                                         0.0s
=> => transferring context: 2B                                           0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest            1.0s
=> [1/9] FROM docker.io/library/ubuntu@sha256:2b7412e6465c3c7fc5bb21d3e6f1917c16 0.0s
=> [internal] load build context                                         0.0s
=> => transferring context: 34B                                           0.0s
=> CACHED [2/9] RUN apt update                                         0.0s
=> CACHED [3/9] RUN apt install -y python3-pip                         0.0s
=> CACHED [4/9] RUN pip install hypercorn quart                         0.0s
=> CACHED [5/9] RUN mkdir -p /si1/build/users                           0.0s
=> [6/9] RUN useradd si1                                              0.3s
=> [7/9] RUN chown -R si1 /si1                                         0.2s
=> [8/9] WORKDIR /si1                                               0.0s
=> [9/9] COPY user_rest.py ./                                         0.0s
=> exporting to image                                                 0.1s
=> => exporting layers                                              0.1s
=> => writing image sha256:419995e035caa7616c369ef300c06657bdcc602aa74583c9dd021 0.0s
=> => naming to docker.io/library/si1p1:1.0u                           0.0s
```

Pregunta: docker build with user

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

Al haber cambiado las instrucciones anteriores, se vuelven a ejecutar las últimas instrucciones por si acaso tuvieran alguna dependencia con las anteriores.

Tarea: docker run myimage

Ejecuta en un terminal (borra el contenedor si1p1 si ya existía):

```
$> 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, ejecuta:

```
$> docker rm -f si1p1
```

```
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker run --name si1p1 -e NUMWORKERS=5 -d -p 8080:8000 si1p1
23f549c219be2d1acabdfec44091b1b492aa0557ce3c3d7e95245a7a7b154311
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker exec -ti si1p1 /bin/bash
root@23f549c219be:/si1# ps --forest -aef
UID      PID  PPID  C STIME TTY          TIME CMD
root      14      0  0 21:21 pts/0    00:00:00 /bin/bash
root      22     14  0 21:21 pts/0    00:00:00 \_ ps --forest -aef
root      1      0  0 21:21 ?        00:00:00 /bin/sh -c hypercorn --bind 0.0.0.0:8000
root      7      1  0 21:21 ?        00:00:00 /usr/bin/python3 /usr/local/bin/hypercor
root      8      7  0 21:21 ?        00:00:00 \_ /usr/bin/python3 -c from multiproces
root      9      7  0 21:21 ?        00:00:00 \_ /usr/bin/python3 -c from multiproces
root     10      7  0 21:21 ?        00:00:00 \_ /usr/bin/python3 -c from multiproces
root     11      7  0 21:21 ?        00:00:00 \_ /usr/bin/python3 -c from multiproces
root     12      7  0 21:21 ?        00:00:00 \_ /usr/bin/python3 -c from multiproces
root     13      7  0 21:21 ?        00:00:00 \_ /usr/bin/python3 -c from multiproces
root@23f549c219be:/si1# exit
exit
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker rm -f si1p1
si1p1
```

Pregunta: docker run myimage

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

Se ejecuta un proceso principal y seis secundarios. De los seis secundarios, uno se debe al proceso principal y el resto a los 5 trabajadores especificados en el parámetro NUMWORKERS.

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

La flag -p publica los puertos del contenedor al host.

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

Deberíamos cambiar el nombre del contenedor y el puerto, pues deben ser valores únicos.

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?

Hemos añadido la flag -v seguida del enlace de directorios.

```
gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ docker run --name si1p1 -e NUMWORKERS=5 -d -p 8080:8000 -v ${PWD}/si1:/si1 si1p1  
f6ffa53aa635c7080444d18f00809e36b8304465e1e7970fa114d817e3a1df1b
```

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?

Los volúmenes externos permiten administrar, cifrar, compartir, migrar y elegir el almacenamiento de datos de forma más fácil y segura que los internos de Docker, por lo que es más fácil realizar back-ups.

Tarea: docker registry

Localiza en docker hub la imagen registry, revisa sus tags y su documentación y descarga la última versión con el mandato de docker adecuado.

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?

```
gonzalo@gonzalo:~$ docker pull registry:latest  
latest: Pulling from library/registry  
96526aa774ef: Pull complete  
cc37b24bb099: Pull complete  
1d8a1aa97222: Pull complete  
e3ff0af69d79: Pull complete  
17443307a4fc: Pull complete  
Digest: sha256:12a6ddd56d2de5611ff0d9735ac0ac1d1e44073c7d042477329e589c46867e4e  
Status: Downloaded newer image for registry:latest  
docker.io/library/registry:latest
```

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

Hemos descargado la última versión, cuyo número podemos consultar en Docker Desktop: 2.8.3.

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

```
gonzalo@gonzalo:~$ docker run -d -p 5050 --restart always --name si1_registry registry:latest  
faf96799a35b42a7915a4d99c41a2fc56c16d8bb85c40eab86fed6fa96b20d35
```

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

Como se puede ver en la documentación, escucha por defecto en el puerto 5000:5000.

Tarea: docker tag push

Descarga la imagen de ubuntu, si no lo has hecho ya.

Ejecuta los siguientes mandatos (asumiendo que el registro está escuchando en el puerto 5000):

```
$> docker tag ubuntu:latest localhost:5000/ubuntu:latest  
$> docker push localhost:5000/ubuntu:latest  
$> docker images
```

```

gonzalo@gonzalo:~$ docker run -d -p 5000:5000 --restart always --name si1_registry5000 registry:latest
693b13ea19ef83d3cdcc22fe0e0a1b3401af5abe94e2376d6eb9f1afc6301c
gonzalo@gonzalo:~$ docker tag ubuntu:latest localhost:5000/ubuntu:latest
gonzalo@gonzalo:~$ docker push localhost:5000/ubuntu:latest
The push refers to repository [localhost:5000/ubuntu]
256d88da4185: Pushed
latest: digest: sha256:f29870aec43cb049f711f7b807626214c9a97e428f93dfcdf3ba23d2c51c2fa5 size: 529
gonzalo@gonzalo:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
si1p1              1.0u    419995e035ca  18 hours ago  479MB
si1p1              1.0     584278ae82f9  18 hours ago  478MB
si1p1              latest   584278ae82f9  18 hours ago  478MB
ubuntu              latest   e4c58958181a  2 weeks ago   77.8MB
localhost:5000/ubuntu latest   e4c58958181a  2 weeks ago   77.8MB
registry            latest   0ae1560ca86f  2 weeks ago   25.4MB

```

Pregunta: docker tag push

¿Qué ha sucedido?

Con el primer comando (no proporcionado en el enunciado) hemos creado un registro escuchando en el puerto 5000.

Con *docker tag*, hemos creado una nueva etiqueta para la imagen del registro local.

Con *docker push* hemos publicado la imagen al registro del puerto 5000.

Finalmente, con *docker images* corroboramos que se ha creado la nueva etiqueta.

Tarea: app from local

Comenta la sentencia FROM del Dockerfile e introduce una nueva que use nuestra copia local de Ubuntu.

Genera la imagen de nuestra aplicación y súbelo a nuestro registro.

Ejecuta un contenedor con nuestra aplicación usando la imagen de nuestro registro.

Pregunta: docker app from local

¿Qué mandatos has usado?

Primero hemos creado la imagen con *docker build* y la hemos llamado *local_ubuntu:1.0*:

```

gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker build --tag local_ubuntu:1.0 .
[+] Building 0.0s (14/14) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 379B
=> [internal] load metadata for localhost:5000/ubuntu:latest
=> [1/9] FROM localhost:5000/ubuntu
=> [internal] load build context
=> => transferring context: 34B
=> CACHED [2/9] RUN apt update
=> CACHED [3/9] RUN apt install -y python3-pip
=> CACHED [4/9] RUN pip install hypercorn quart
=> CACHED [5/9] RUN mkdir -p /si1/build/users
=> CACHED [6/9] RUN useradd si1
=> CACHED [7/9] RUN chown -R si1 /si1
=> CACHED [8/9] WORKDIR /si1
=> CACHED [9/9] COPY user_rest.py ./ 
=> exporting to image
=> => exporting layers
=> => writing image sha256:419995e035caa7616c369ef300c06657bdcc602aa74583c9dd02161c65b95ad0
=> => naming to docker.io/library/local_ubuntu:1.0

```

Luego hemos creado la etiqueta de la imagen del registro local y la hemos publicado al registro local:

```
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker tag local_ubuntu:1.0 localhost:5000/local_ubuntu:1.0
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker push localhost:5000/local_ubuntu:1.0
The push refers to repository [localhost:5000/local_ubuntu]
2a8e600a3b85: Pushed
5f70bf18a086: Pushed
5807c8a98e60: Pushed
c12f070606c1: Pushed
0b89c2ae6161: Pushed
2eb8e63ef04c: Pushed
d023bc584957: Pushed
a91b01b4e96a: Pushed
256d88da4185: Mounted from ubuntu
1.0: digest: sha256:f3e77af3b4033ed4561bf1ea6ca3ed14a94d20fed44a2a129901024afbf20694 size: 2200
```

Para asegurarnos de que está usando la imagen desde el registro, hemos borrado las imágenes locales que acabamos de crear:

```
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker image remove local_ubuntu:1.0
Untagged: local_ubuntu:1.0
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker image remove localhost:5000/local_ubuntu:1.0
Untagged: localhost:5000/local_ubuntu:1.0
Untagged: localhost:5000/local_ubuntu@sha256:f3e77af3b4033ed4561bf1ea6ca3ed14a94d20fed44a2a129901024afbf20694
```

Finalmente, hemos ejecutado un contenedor que usa la imagen de nuestro registro local como se puede ver:

```
gonzalo@gonzalo:~/SI1_PR/p1-v1.0/src$ docker run -ti -d --name ubuntu_from_reg localhost:5000/local_ubuntu:1.0
Unable to find image 'localhost:5000/local_ubuntu:1.0' locally
1.0: Pulling from local_ubuntu
Digest: sha256:f3e77af3b4033ed4561bf1ea6ca3ed14a94d20fed44a2a129901024afbf20694
Status: Downloaded newer image for localhost:5000/local_ubuntu:1.0
56e0f2fc7e122fce83a18e151ad36cfba56d18393fab74fe45dd2a1332a952fe
```

Parte 2

AWS S3

Tarea: AWS S3

Ejecuta en un terminal, desde el directorio donde está el subdirectorio src con el archivo src/user_rest.py:

```
# create bucket
$ awslocal s3 mb s3://si1p1
# create dir & copy
$ awslocal s3 cp src/user_rest.py s3://si1p1/src/user_rest.py
$ awslocal s3 ls s3://si1p1
# ... dir
$ awslocal s3 ls s3://si1p1/src
# ... dir contents
$ awslocal s3 ls s3://si1p1/src/
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal s3 mb s3://si1p1
make_bucket: si1p1
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ 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
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal s3 ls s3://si1p1
PRE src/
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal s3 ls s3://si1p1/src
PRE src/
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal s3 ls s3://si1p1/src/
2023-10-19 17:41:03          1832 user_rest.py
```

Pregunta: AWS S3

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

“mb” crea un bucket llamado “s3” en “si1p1”. “cp” copia el archivo “src/user_rest.py” en el bucket “si1p1” en la ruta “src/user_rest.py”. ls lista los archivos que se encuentren en la carpeta indicada.

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 /ruta/nombre..py
```

Con este comando copiaremos el archivo deseado, utilizando la ruta del bucket creado, para después indicarle la ruta de destino y su nuevo nombre.

AWS Lambda

Tarea: Lambdas

Ejecuta en un terminal:

```
$> 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"}'
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ LAMBDA_NAME=user_lambda
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ [[ -d build ]] || mkdir build ; \
   cd src ; zip ../build/${LAMBDA_NAME}.zip ${LAMBDA_NAME}.py; cd -
   adding: user_lambda.py (deflated 46%)
/home/gonzalo/SI1_PR/p1-v1.0
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ 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 ${LAMBDA_NAME} \
--auth-type NONE ;
create: newlambda
{
    "FunctionName": "user_lambda",
    "FunctionArn": "arn:aws:lambda:us-east-1:000000000000:function:user_lambda",
    "Runtime": "python3.9",
    "Role": "arn:aws:iam::000000000000:role/si1",
    "Handler": "user_lambda.handler",
    "CodeSize": 799,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2023-10-19T15:45:40.613951+0000",
    "CodeSha256": "GR7fqqBhKrAsgcu98LP0960yJtea9pr7a9075gPpzI=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "958d4958-bb0c-4cb9-97c1-e88f8a17d183",
    "State": "Pending",
    "StateReason": "The function is being created.",
    "StateReasonCode": "Creating",
    "PackageType": "Zip",
    "Architectures": [
        "x86_64"
    ],
    "EphemeralStorage": {
        "Size": 512
    },
    "SnapStart": {
        "ApplyOn": "None",
        "OptimizationStatus": "Off"
    },
    "RuntimeVersionConfig": {
        "RuntimeVersionArn": "arn:aws:lambda:us-east-1::runtime:8eff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1"
    }
}
{
    "FunctionUrl": "http://k9rl0x7rnr0h80f12v5zsjyuhpfox712.lambda-url.us-east-1.localhost.localstack.cloud:4566/",
    "FunctionArn": "arn:aws:lambda:us-east-1:000000000000:function:user_lambda",
    "AuthType": "NONE",
    "CreationTime": "2023-10-19T15:45:42.042148+0000"
}
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal lambda wait function-active-v2 --function-name ${LAMBDA_NAME}
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ curl -X POST \
$(awslocal lambda get-function-url-config --function-name ${LAMBDA_NAME} | \
jq -r '.FunctionUrl') \
-H 'Content-Type: application/json' \
-d '{"username": "pepe", "data": "10"}'
"f{test_object_key} placed into S3"(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ []
```

Pregunta: Lambdas

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

Podríamos utilizar Node.js, que es un runtime de JavaScript. Por tanto, un posible runtime podría ser: `--runtime nodejs{n_version}`.

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

La función Lambda analiza el cuerpo de la solicitud HTTP y accede a los campos enviados.

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

Lo descargamos con el comando `awslocal lambda get-function`:

```
(siipi1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal lambda get-function --function-name ${LAMBDA_NAME}{

  "Configuration": {
    "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": "2023-10-19T15:45:40.613951+0000",
    "CodeSha256": "GR7fqBhKrAsgcu98LP0960yJtea9pr7a9075zgPpzI=",
    "Version": "$LATEST",
    "TracingConfig": [
      "Mode": "PassThrough"
    ],
    "RevisionId": "eac0db43-8b42-4ded-9473-db9157fdce4b",
    "State": "Active",
    "LastUpdateStatus": "Successful",
    "PackageType": "Zip",
    "Architectures": [
      "x86_64"
    ],
    "EphemeralStorage": {
      "Size": 512
    },
    "SnapStart": {
      "ApplyOn": "None",
      "OptimizationStatus": "Off"
    },
    "RuntimeVersionConfig": {
      "RuntimeVersionArn": "arn:aws:lambda:us-east-1::runtime:8efff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1"
    }
  },
  "Code": {
    "RepositoryType": "S3",
    "Location": "http://s3.localstack.cloud:4566/awslambda-us-east-1-tasks/snapshots/000000000000/user_lambda-f01b509e-f9b3-4666-8b1a-02653f578e
c7?AWSAccessKeyId=949334387222&Signature=rZA7zflLYV1qoY2K4MNCz%FjIYck%3D&Expires=1697734666"
  }
}
```

Esto descarga el user_lambda.zip, dentro de nuestra carpeta build, que contiene el archivo user_lambda.py:

```
1 #!/bin/env python3
2
3 import boto3
4 import json
5 import logging
6 import os
7
8 LOGGER = logging.getLogger()
9 LOGGER.setLevel(logging.INFO)
10
11 ENDPOINT_URL = f"http://{{os.getenv('LOCALSTACK_HOSTNAME')}}:{{os.getenv('EDGE_PORT')}}"
12
13 BUCKET_NAME = 'si1p1'
14
15 def get_s3_client():
16     return boto3.client('s3',
17         aws_access_key_id='',
18         aws_secret_access_key='',
19         region_name='eu-west-1',
20         endpoint_url=ENDPOINT_URL
21     )
22
23
24 S3_CLIENT = get_s3_client()
25
26 # Nos aseguramos que el bucket existe
27 S3_CLIENT.create_bucket(Bucket=BUCKET_NAME)
28
29 def handler(event, context):
30     LOGGER.info('Escribiendo en S3')
31     print("Received event: " + json.dumps(event, indent=2))
32     req=json.loads(event['body'])
33     test_object_key = f"users/{{req['username']}}"
34     S3_CLIENT.put_object(
35         Bucket=BUCKET_NAME,
36         Key=test_object_key,
37         Body=json.dumps(req)
38     )
39     resp_body=f'{test_object_key} placed into S3'
40     # API GW compliant response
41     resp = {
42         "isBase64Encoded": False,
43         "statusCode": 200,
44         "headers": [
45             {"content-type": "application/json"
46         },
47         "body": json.dumps(resp_body)
48     }
49     return resp
```

Tarea: depuración lambdas

Ejecuta en un terminal:

```
$> 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 muestra todo lo que ha ido haciendo LocalStack, es decir sus registros.

```
(s11p1) gonzo@gonzalo:~/SI1_PR/p1-v1.0$ docker logs localstack_main
LocalStack version: 2.3.3.dev
LocalStack Docker container id: 56296604fd98
LocalStack build date: 2023-10-19
LocalStack build git hash: e4a6b649

2023-10-19T15:40:23.763 INFO --- [-functhread4] hypercorn.error : Running on https://0.0.0.0:4566 (CTRL + C to quit)
2023-10-19T15:40:23.763 INFO --- [-functhread4] hypercorn.error : Running on https://0.0.0.0:4566 (CTRL + C to quit)
Ready.
2023-10-19T15:40:53.261 INFO --- [ asgi_gw_0] localstack.request.aws : AWS s3.CreateBucket => 200
2023-10-19T15:41:03.562 INFO --- [ asgi_gw_0] localstack.request.aws : AWS s3.PutObject => 200
2023-10-19T15:41:12.889 INFO --- [ asgi_gw_0] localstack.request.aws : AWS s3.ListObjectsV2 => 200
2023-10-19T15:41:31.367 INFO --- [ asgi_gw_0] localstack.request.aws : AWS s3.ListObjectsV2 => 200
2023-10-19T15:41:55.105 INFO --- [ asgi_gw_0] localstack.request.aws : AWS s3.ListObjectsV2 => 200
2023-10-19T15:45:40.618 INFO --- [ asgi_gw_0] localstack.request.aws : AWS lambda.CreateFunction => 201
2023-10-19T15:45:42.042 INFO --- [ asgi_gw_0] localstack.request.aws : AWS lambda.CreateFunctionUrlConfig => 201
2023-10-19T15:45:53.092 INFO --- [ asgi_gw_0] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:45:54.102 INFO --- [ asgi_gw_1] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:45:55.108 INFO --- [ asgi_gw_0] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:45:56.116 INFO --- [ asgi_gw_1] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:45:57.123 INFO --- [ asgi_gw_0] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:45:58.130 INFO --- [ asgi_gw_1] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:45:59.136 INFO --- [ asgi_gw_0] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:46:00.144 INFO --- [ asgi_gw_1] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:46:01.151 INFO --- [ asgi_gw_1] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:46:02.157 INFO --- [ asgi_gw_0] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:46:03.163 INFO --- [ asgi_gw_1] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:46:04.172 INFO --- [ asgi_gw_0] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:46:05.180 INFO --- [ asgi_gw_0] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T15:55:25.433 INFO --- [ asgi_gw_1] localstack.request.aws : AWS lambda.ListFunctions => 200
2023-10-19T15:56:53.486 INFO --- [ asgi_gw_0] localstack.request.aws : AWS lambda.ListFunctions => 200
2023-10-19T15:57:48.456 INFO --- [ asgi_gw_1] localstack.request.aws : AWS lambda.GetFunction => 200
2023-10-19T16:01:25.458 INFO --- [ asgi_gw_0] localstack.request.aws : AWS lambda.GetFunctionUrlConfig => 200
2023-10-19T16:01:25.573 INFO --- [ asgi_gw_0] l.u.container_networking : Determined main container network: bridge
2023-10-19T16:01:25.582 INFO --- [ asgi_gw_0] l.u.container_networking : Determined main container target IP: 172.17.0.5
2023-10-19T16:01:26.132 INFO --- [ asgi_gw_2] localstack.request.aws : AWS s3.CreateBucket => 200
```

awslocal logs describe-log-groups muestra un grupo de registros llamado "/aws/lambda/user_lambda", que parece guardar los registros de la función lambda anterior.

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal logs describe-log-groups
{
    "logGroups": [
        {
            "logGroupName": "/aws/lambda/user_lambda",
            "creationTime": 1697731286214,
            "metricFilterCount": 0,
            "arn": "arn:aws:logs:us-east-1:000000000000:log-group:/aws/lambda/user_lambda:*",
            "storedBytes": 1565
        }
    ]
}
```

awslocal logs describe-log-streams --log-group-name /aws/lambda/\${LAMBDA_NAME} muestra las secuencias de registros dentro del grupo de registros asociado a la función Lambda \${LAMBDA_NAME}

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal logs describe-log-streams --log-group-name /aws/lambda/${LAMBDA_NAME}
{
    "logStreams": [
        {
            "logStreamName": "2023/10/19/[$LATEST]7596a5bcd68d5396c8697354d771bdb",
            "creationTime": 1697731286225,
            "firstEventTimestamp": 1697731286156,
            "lastEventTimestamp": 1697731286222,
            "lastIngestionTime": 1697731286234,
            "uploadSequenceToken": "1",
            "arn": "arn:aws:logs:us-east-1:000000000000:log-group:/aws/lambda:log-stream:2023/10/19/[$LATEST]7596a5bcd68d5396c8697354d771bdb",
            "storedBytes": 1565
        }
    ]
}
```

awslocal logs describe-log-streams --log-group-name /aws/lambda/\${LAMBDA_NAME} | jq '.logStreams[].firstEventTimestamp' | sort -n muestra las marcas de tiempo del primer evento en cada secuencia de registros de menor a mayor.

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal logs describe-log-streams --log-group-name /aws/lambda/${LAMBDA_NAME} | jq '.logStreams[].firstEventTimestamp' | sort -n
1697731286156
```

LLOGSTREAMS=\$(awslocal logs describe-log-streams \ --log-group-name /aws/lambda/\${LAMBDA_NAME} | jq -r '.logStreams[].logStreamName') guarda en una variable LLOGSTREAMS los nombres de los flujos de registros asociados con la función Lambda.

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ LLOGSTREAMS=$(awslocal logs describe-log-streams \
--log-group-name /aws/lambda/${LAMBDA_NAME} |
jq -r '.logStreams[].logStreamName')
```

El bucle for recorre cada uno de los flujos de registros almacenados en la variable LLOGSTREAMS(hecho con el comando anterior). Luego, utiliza el comando awslocal logs get-log-events para obtener los eventos de registro de cada flujo de registros.

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ 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'
"START RequestId: 62c6a48c-c784-4a65-8aae-ac44def7523d Version: $LATEST"
"[INFO]\t2023-10-16:01:26.139Z\t62c6a48c-c784-4a65-8aae-ac44def7523d\tEscribiendo en S3"
"Received event: {"
"  \"version\": \"2.0\","
"  \"routeKey\": \"$default\","
"  \"rawPath\": \"/\","
"  \"rawQueryString\": \"/\","
"  \"headers\": {"
"    \"host\": \"k9rl6x7rrn0h80f12v5zsjyuhpfox712.lambda-url.us-east-1.localstack.cloud:4566\","
"    \"user-agent\": \"curl/7.81.0\","
"    \"accept\": \"/*/*\","
"    \"content-type\": \"application/json\","
"    \"content-length\": \"34\","
"    \"x-amzn-tls-cipher-suite\": \"ECDHE-RSA-AES128-GCM-SHA256\","
"    \"x-amzn-tls-version\": \"TLSV1.2\","
"    \"x-forwarded-proto\": \"http\","
"    \"x-forwarded-for\": \"\","
"    \"x-forwarded-port\": \"4566\""
"  },"
"  \"queryStringParameters\": {},"
"  \"requestContext\": {"
"    \"accountId\": \"anonymous\","
"    \"apiId\": \"k9rl6x7rrn0h80f12v5zsjyuhpfox712\","
"    \"domainName\": \"k9rl6x7rrn0h80f12v5zsjyuhpfox712.lambda-url.us-east-1.localstack.cloud:4566\","
"    \"domainPrefix\": \"k9rl6x7rrn0h80f12v5zsjyuhpfox712\","
"    \"http\": {"
"      \"method\": \"POST\","
"      \"path\": \"/\","
"      \"protocol\": \"HTTP/1.1\","
"      \"sourceIp\": \"\","
"      \"userAgent\": \"curl/7.81.0\""
"    },"
"    \"requestId\": \"4aed8713-7970-4e3c-8161-67d165d28abb\","
"    \"routeKey\": \"$default\","
"    \"stage\": \"$default\","
"    \"time\": \"19/Oct/2023:16:01:25 +0000\","
"    \"timeEpoch\": 167773128545"
"  },"
"  \"body\": \"{\\\"username\\\": \"pepe\\\", \\\"data\\\": \\\"10\\\"}\","
"  \"isBase64Encoded\": false"
"}"
"END RequestId: 62c6a48c-c784-4a65-8aae-ac44def7523d"
"REPORT RequestId: 62c6a48c-c784-4a65-8aae-ac44def7523d\tDuration: 10.09 ms\tBilled Duration: 11 ms\tMemory Size: 128 MB\tMax Memory Used: 128 MB\t"

```

Tarea: Borrado de funciones

Ejecuta en un terminal:

```
$> 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://silpl1 ; \
    rm build/user_lambda.*
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ echo "clean: dellambda" ; \
clean: dellambda
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal lambda delete-function-url-config \
--function-name user_lambda ; \
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal lambda delete-function --function-name user_lambda
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ echo "clean: md zip" \
    awslocal s3 rb --force s3://silpl1 ; \
    rm build/user_lambda.*
clean: md zip awslocal s3 rb --force s3://silpl1
```

Pregunta: Borrado de funciones

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

Al reiniciar LocalStack, se eliminan las funciones Lambda y los archivos en S3.

Para evitar esto, se podría hacer una copia de seguridad, aunque eso sería más bien para conservar las funciones y archivos, pero no evitaría el problema de que se

borren al reiniciar LocalStack, por lo tanto la mejor opción sería configurar LocalStack para que persistan los datos en sus servicios emulados.

API Gateway

Tarea: Creación del API

Ejecuta en un terminal:

```
$> 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
```

```
(siip1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ export REGION="us-east-1" ; export API_NAME="user_lambda"
(sii1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway create-rest-api \
--region ${REGION} \
--name ${API_NAME}
{
  "id": "dmsz4z0ji9",
  "name": "user_lambda",
  "createdDate": 1697732774.0,
  "apiKeySource": "HEADER",
  "endpointConfiguration": {
    "types": [
      "EDGE"
    ]
  },
  "disableExecuteApiEndpoint": false
}
(sii1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ [ $? == 0 ] || echo "Failed: AWS / apigateway / create-rest-api"
(sii1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-rest-apis
{
  "items": [
    {
      "id": "dmsz4z0ji9",
      "name": "user_lambda",
      "createdDate": 1697732774.0,
      "apiKeySource": "HEADER",
      "endpointConfiguration": {
        "types": [
          "EDGE"
        ]
      },
      "disableExecuteApiEndpoint": false
    }
  ]
}
```

Pregunta: Creación del API

1. ¿Qué ID se ha asignado al API ?

Como se puede ver en terminal, se ha asignado el ID “*dmsz4z0ji9*”

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

La crea con otro ID:

```
[si1p1] gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway create-rest-api --region ${REGION} --name ${API_NAME}
{
    "id": "j3wk0uu876",
    "name": "user_lambda",
    "createdDate": 1697732908.0,
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "EDGE"
        ]
    },
    "disableExecuteApiEndpoint": false
}
[si1p1] gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ [ $? == 0 ] || echo "Failed: AWS / apigateway / create-rest-api"
[si1p1] gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-rest-apis
{
    "items": [
        {
            "id": "dmsz4z0ji9",
            "name": "user_lambda",
            "createdDate": 1697732774.0,
            "apiKeySource": "HEADER",
            "endpointConfiguration": {
                "types": [
                    "EDGE"
                ]
            },
            "disableExecuteApiEndpoint": false
        },
        {
            "id": "j3wk0uu876",
            "name": "user_lambda",
            "createdDate": 1697732908.0,
            "apiKeySource": "HEADER",
            "endpointConfiguration": {
                "types": [
                    "EDGE"
                ]
            },
            "disableExecuteApiEndpoint": false
        }
    ]
}
```

Tarea: Creación de las rutas

Antes de continuar asegúrate que solo tenemos un api con el nombre "user_lambda".

Si hubiera más de uno, puedes borrarlos ejecutando:

```
$> awslocal apigateway delete-rest-api --rest-api-id id_del_api_a_borrar
```

Ejecuta en un terminal:

```

$> API_ID=$(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}

(sii1p1) gonzo@gonzalo:~/SI1_PR/p1-v1.0$ API_ID=$(awslocal apigateway get-rest-apis \
--query "items[?name=='${API_NAME}'].id" \
--output text --region ${REGION})
(sii1p1) gonzo@gonzalo:~/SI1_PR/p1-v1.0$ PATH_PART="user" ; PARENT_PATH="/"
(sii1p1) gonzo@gonzalo:~/SI1_PR/p1-v1.0$ PARENT_RESOURCE_ID=$(awslocal apigateway get-resources \
--rest-api-id ${API_ID} --query "items[?path=='${PARENT_PATH}'].id" \
--output text --region ${REGION})
(sii1p1) gonzo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway create-resource \
--region ${REGION} \
--rest-api-id ${API_ID} \
--parent-id ${PARENT_RESOURCE_ID} \
--path-part "${PATH_PART}"
{
    "id": "ffbw18tmi4",
    "parentId": "j9gju7e7ca",
    "pathPart": "user",
    "path": "/user"
}
(sii1p1) gonzo@gonzalo:~/SI1_PR/p1-v1.0$ PATH_PART="{username}" ; PARENT_PATH="/user"
(sii1p1) gonzo@gonzalo:~/SI1_PR/p1-v1.0$ PARENT_RESOURCE_ID=$(awslocal apigateway get-resources \
--rest-api-id ${API_ID} --query "items[?path=='${PARENT_PATH}'].id" \
--output text --region ${REGION})
(sii1p1) gonzo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway create-resource \
--region ${REGION} \
--rest-api-id ${API_ID} \
--parent-id ${PARENT_RESOURCE_ID} \
--path-part "${PATH_PART}"
{
    "id": "tm376ukikv",
    "parentId": "ffbw18tmi4",
    "pathPart": "{username}",
    "path": "/user/{username}"
}
(sii1p1) gonzo@gonzalo:~/SI1_PR/p1-v1.0$ [ $? == 0 ] || echo "Failed: AWS / apigateway / create-resource ${PATH_PART}"

```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-resources --rest-api-id ${API_ID}
{
  "items": [
    {
      "id": "j9gju7e7ca",
      "path": "/"
    },
    {
      "id": "ffbw18tmi4",
      "parentId": "j9gju7e7ca",
      "pathPart": "user",
      "path": "/user"
    },
    {
      "id": "tm376ukikv",
      "parentId": "ffbw18tmi4",
      "pathPart": "{username}",
      "path": "/user/{username}"
    }
  ]
}
```

Pregunta: Creación de las rutas

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

Se le ha asignado el ID "tm376ukikv".

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

Se obtiene un error de conflicto:

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway create-resource \
--region ${REGION} \
--rest-api-id ${API_ID} \
--parent-id ${PARENT_RESOURCE_ID} \
--path-part "${PATH_PART}"

An error occurred (BadRequestException) when calling the CreateResource operation: A sibling ({username}) of this resource already has a variable path part -- only one is allowed
```

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

No parece tener mucho sentido ya que puede ser confuso y poco práctico para información que esencialmente es la misma.

Tarea: Creación de métodos

Podemos asociar más de un método HTTP a una misma ruta, para los distintos métodos de nuestro API. En este paso especificaremos también los parámetros, que pueden ser de tres tipos, *path*, *querystring* y *header*. Podemos añadir tantos parámetros como necesitemos, y también darles un valor estático.

Para crear por ejemplo el método POST en /user/{username}, con un parámetro de ruta, ejecuta:

```
$> 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}
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ PARAM=username ; METHOD_PATH="/user/{username}" ; HTTP_METHOD=POST
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ RESOURCE_ID=$(awslocal apigateway get-resources \
--rest-api-id ${API_ID} --query "items[?path=='${METHOD_PATH}'].id" \
--output text --region ${REGION})
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ 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"
{
  "httpMethod": "POST",
  "authorizationType": "NONE",
  "apiKeyRequired": false,
  "requestParameters": {
    "method.request.path.username": true
  }
}
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ [ $? == 0 ] || echo "Failed: AWS / apigateway / put-method ${METHOD_PATH}"
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-resources --rest-api-id ${API_ID}
{
  "items": [
    {
      "id": "j9gju7e7ca",
      "path": "/"
    },
    {
      "id": "ffbw18tmi4",
      "parentId": "j9gju7e7ca",
      "pathPart": "user",
      "path": "/user"
    },
    {
      "id": "tm376ukikv",
      "parentId": "ffbw18tmi4",
      "pathPart": "{username}",
      "path": "/user/{username}",
      "resourceMethods": {
        "POST": {
          "httpMethod": "POST",
          "authorizationType": "NONE",
          "apiKeyRequired": false,
          "requestParameters": {
            "method.request.path.username": true
          },
          "methodResponses": {}
        }
      }
    }
  ]
}
```

Pregunta: Creación de métodos

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

Usamos los mismos comandos que para el método POST menos el primero, que sustituimos por tan solo la instanciación de la variable *HTTP_METHOD* a *GET*. Como se puede observar, el método se crea en la misma ruta:

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-resources --rest-api-id ${API_ID}
{
  "items": [
    {
      "id": "j9gju7e7ca",
      "path": "/"
    },
    {
      "id": "ffbw18tmi4",
      "parentId": "j9gju7e7ca",
      "pathPart": "user",
      "path": "/user"
    },
    {
      "id": "tm376ukikv",
      "parentId": "ffbw18tmi4",
      "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": {}
        }
      }
    }
  ]
}
```

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

Podemos verlos usando el comando “`awslocal apigateway get-methods --rest-api-id ${API_ID} --region ${REGION}`”.

Tarea: Integración

Nota: Para integrar la función lambda usaremos el tipo AWS_PROXY. El método de integración será siempre POST, independiente del valor de http_method. La URI para invocar la lambda, según la especificación de AWS, tendrá la forma: /2015-03-31/functions/FunctionName/invocations

Ejecuta en un terminal:

```
$> 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}
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ HTTP_METHOD=POST ; LAMBDA_NAME=user_lambda
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ 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
{
  "type": "AWS_PROXY",
  "httpMethod": "POST",
  "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/user_lambda/invocations",
  "passthroughBehavior": "WHEN_NO_MATCH",
  "timeoutInMillis": 29000,
  "cacheNamespace": "tm376ukikv",
  "cacheKeyParameters": []
}
}

(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-resources --rest-api-id ${API_ID}
{
  "items": [
    {
      "id": "j9gju7e7ca",
      "path": "/"
    },
    {
      "id": "ffbw8tmi4",
      "parentId": "j9gju7e7ca",
      "pathPart": "user",
      "path": "/user"
    },
    {
      "id": "tm376ukikv",
      "parentId": "ffbw8tmi4",
      "pathPart": "{username}",
      "path": "/user/{username}",
      "resourceMethods": {
        "POST": {
          "httpMethod": "POST",
          "authorizationType": "NONE",
          "apiKeyRequired": false,
          "requestParameters": {
            "method.request.path.username": true
          },
          "methodResponses": {},
          "methodIntegration": {
            "type": "AWS_PROXY",
            "httpMethod": "POST",
            "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/user_lambda/invocations",
            "passthroughBehavior": "WHEN_NO_MATCH",
            "timeoutInMillis": 29000,
            "cacheNamespace": "tm376ukikv",
            "cacheKeyParameters": []
          }
        },
        "GET": {
          "httpMethod": "GET",
          "authorizationType": "NONE",
          "apiKeyRequired": false,
          "requestParameters": {
            "method.request.path.username": true
          },
          "methodResponses": {}
        }
      }
    }
  ]
}
```

Pregunta: Integración

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

Hemos usado las mismas instrucciones que para POST, cambiando el método POST por GET:

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ HTTP_METHOD=GET
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway put-integration \
--region ${REGION} \
--rest-api-id ${API_ID} \
--resource-id ${RESOURCE_ID} \
--http-method ${HTTP_METHOD} \
--type AWS_PROXY \
--integration-http-method GET \
--uri arn:aws:apigateway:${REGION}:lambda:path/2015-03-31/functions/${LAMBDA_NAME}/invocations \
--passthrough-behavior WHEN_NO_MATCH
{
  "type": "AWS_PROXY",
  "httpMethod": "GET",
  "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/user_lambda/invocations",
  "passthroughBehavior": "WHEN_NO_MATCH",
  "timeoutInMillis": 29000,
  "cacheNamespace": "tm376ukikv",
  "cacheKeyParameters": []
}
]

(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ [ $? == 0 ] || echo "Failed: AWS / apigateway / put-integration"
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-resources --rest-api-id ${API_ID}
{
  "items": [
    {
      "id": "j9gju7e7ca",
      "path": "/"
    },
    {
      "id": "ffbw18tmi4",
      "parentId": "j9gju7e7ca",
      "pathPart": "user",
      "path": "/user"
    },
    {
      "id": "tm376ukikv",
      "parentId": "ffbw18tmi4",
      "pathPart": "{username}",
      "path": "/user/{username}",
      "resourceMethods": {
        "POST": {
          "httpMethod": "POST",
          "authorizationType": "NONE",
          "apiKeyRequired": false,
          "requestParameters": {
            "method.request.path.username": true
          },
          "methodResponses": {},
          "methodIntegration": {
            "type": "AWS_PROXY",
            "httpMethod": "POST",
            "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/user_lambda/invocations",
            "passthroughBehavior": "WHEN_NO_MATCH",
            "timeoutInMillis": 29000,
            "cacheNamespace": "tm376ukikv",
            "cacheKeyParameters": []
          }
        },
        "GET": {
          "httpMethod": "GET",
          "authorizationType": "NONE",
          "apiKeyRequired": false,
          "requestParameters": {
            "method.request.path.username": true
          },
          "methodResponses": {},
          "methodIntegration": {
            "type": "AWS_PROXY",
            "httpMethod": "GET",
            "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/user_lambda/invocations",
            "passthroughBehavior": "WHEN_NO_MATCH",
            "timeoutInMillis": 29000,
            "cacheNamespace": "tm376ukikv",
            "cacheKeyParameters": []
          }
        }
      }
    }
  ]
}
```

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

Podemos comprobarlo con el comando `awslocal apigateway get-integration` seguido de las especificaciones de nuestro método:

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-integration \
--region ${REGION} \
--rest-api-id ${API_ID} \
--resource-id ${RESOURCE_ID} \
--http-method ${HTTP_METHOD}
{
  "type": "AWS_PROXY",
  "httpMethod": "GET",
  "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/user_lambda/invocations",
  "passthroughBehavior": "WHEN_NO_MATCH",
  "timeoutInMillis": 29000,
  "cacheNamespace": "tm376ukikv",
  "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?

Esto se consigue mediante el siguiente comando:

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway delete-integration \
--region ${REGION} \
--rest-api-id ${API_ID} \
--resource-id ${RESOURCE_ID} \
--http-method ${HTTP_METHOD}
```

Tarea: Despliegue

AWS soporta distintas etapas (Stage) desplegadas simultáneamente, cada una asociada a un despliegue.

Esto puede ayudar por ejemplo a realizar pruebas durante el desarrollo, desplegando etapas con distintas configuraciones.

Una limitación importante es que el despliegue no fija la implementación de la integración. Por ejemplo, si modificamos una lambda que ya está integrada, la aplicación ejecutará la nueva lambda incluso sin hacer un nuevo despliegue. Para evitar este problema, se pueden asociar distintos números de versión a las lambdas, e integrar versiones específicas en cada método.

Ejecuta en un terminal:

```
$> 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}
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ STAGE=dev
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway create-deployment \
--region ${REGION} \
--rest-api-id ${API_ID} \
--stage-name ${STAGE}
{
    "id": "b6gq1aycg0",
    "createdDate": 1697736935.0
}
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ [ $? == 0 ] || echo " Failed: AWS / apigateway / create-deployment"
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-deployments --rest-api-id ${API_ID}
[
    "items": [
        {
            "id": "b6gq1aycg0",
            "createdDate": 1697736935.0
        }
    ]
}
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-stages --rest-api-id ${API_ID}
{
    "item": [
        {
            "deploymentId": "b6gq1aycg0",
            "stageName": "dev",
            "cacheClusterEnabled": false,
            "cacheClusterStatus": "NOT_AVAILABLE",
            "methodSettings": {},
            "tracingEnabled": false
        }
    ]
}
```

Pregunta: Despliegue

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

Cambia su *deploymentID* al ID de este último despliegue:

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-stages --rest-api-id ${API_ID}
{
    "item": [
        {
            "deploymentId": "p00hs8xmc4",
            "stageName": "dev",
            "cacheClusterEnabled": false,
            "cacheClusterStatus": "NOT_AVAILABLE",
            "methodSettings": {},
            "tracingEnabled": false
        }
    ]
}
```

Esto significa que el primer despliegue ya no está asociado a la etapa "dev".

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?

Lo conseguimos usando *delete deployment* especificando el id del despliegue antiguo:

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway delete-deployment --region ${REGION} --rest-api-id ${API_ID} --deployment-id "b6gq1aycg0"
```

Tarea: Pruebas del API

Podremos probar nuestro API usando la siguiente URL como base:

[http://localhost:4566/restapis/\\${API_ID}/\\${STAGE}/_user_request_](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"}'
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ ENDPOINT=http://localhost:4566/restapis/${API_ID}/${STAGE}/_user_request/_user
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ curl -X POST ${ENDPOINT}/pepe \
-H 'Content-Type: application/json' \
-d '{"username": "tete", "data": "100"}'
"f{test_object_key} placed into S3"(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ 
```

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?

Se ha creado un archivo tete en el directorio de users dentro del bucket si1p1 con los atributos de tete.

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal s3 cp s3://si1p1/users/tete src/users_cpy
download: s3://si1p1/users/tete to src/users_cpy
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ cat src/users_cpy
>{"username": "tete", "data": "100"}(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ 
```

Tarea: Mejora del API

Crea una nueva función lambda llamada “user_lambda_param” que use realmente el parámetro “username” que le proporciona el API, ya que la lambda anterior lo estaba ignorando.

Para esta tarea 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']
```

Nota: Los valores de “pathParameters” los rellena el servicio API Gateway, por lo que no podrás probar la lambda directamente, y será necesario integrarla en el API para realizar las pruebas.

Después de guardar el *username*, lo usamos para el path de *test_object_key*.

```
test_object_key = f"users/{username}/data.json"
```

Pregunta: Mejora del API

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

Para usar la nueva lambda hemos usado los mismos comandos que con la antigua, cambiando la variable de *LAMBDA_NAME* al nombre de la función mejorada (*user_lambda_param*):

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ LAMBDA_NAME=user_lambda_param
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ cd src ; zip ../_build/${LAMBDA_NAME}.zip ${LAMBDA_NAME}.py; cd -
  adding: user_lambda_param.py (deflated 46%)
/home/gonzalo/SI1_PR/p1-v1.0
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ echo "create: newlambda"; \
awslocal lambda create-function \
--function-name ${LAMBDA_NAME} \
--runtime python3.9 \
--zip-file file:///build/${LAMBDA_NAME}.zip \
--handler ${LAMBDA_NAME}.handler \
--role arn:aws:iam::000000000000:role/si1 ; \
awslocal lambda create-function-url-config \
--function-name ${LAMBDA_NAME} \
--auth-type NONE ;
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal lambda wait function-active-v2 --function-name ${LAMBDA_NAME}
```

Como se puede ver, la función lambda está en la API:

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ 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": "2023-10-19T18:30:05.622240+0000",
      "CodeSha256": "GR7fqBhKrasgcu98LP0960yJtea9pr7a9075zgPpzI=",
      "Version": "$LATEST",
      "TracingConfig": {
        "Mode": "PassThrough"
      },
      "RevisionId": "226520ad-5388-4371-b4ef-f0a1bcd7fd8b",
      "PackageType": "Zip",
      "Architectures": [
        "x86_64"
      ],
      "EphemeralStorage": {
        "Size": 512
      },
      "SnapStart": {
        "ApplyOn": "None",
        "OptimizationStatus": "Off"
      }
    },
    {
      "FunctionName": "user_lambda_param",
      "FunctionArn": "arn:aws:lambda:us-east-1:000000000000:function:user_lambda_param",
      "Runtime": "python3.9",
      "Role": "arn:aws:iam::000000000000:role/si1",
      "Handler": "user_lambda_param.handler",
      "CodeSize": 837,
      "Description": "",
      "Timeout": 3,
      "MemorySize": 128,
      "LastModified": "2023-10-19T18:58:35.963808+0000",
      "CodeSha256": "YMo7TiJaJUA0EOXPrYHFk9oJ/0poRdiPnCESq7iqdNfU=",
      "Version": "$LATEST",
      "TracingConfig": {
        "Mode": "PassThrough"
      },
      "RevisionId": "59021c4a-e0ba-4448-a6a1-18c2eb80ab3d",
      "PackageType": "Zip",
      "Architectures": [
        "x86_64"
      ],
      "EphemeralStorage": {
        "Size": 512
      },
      "SnapStart": {
        "ApplyOn": "None",
        "OptimizationStatus": "Off"
      }
    }
  ]
}
```

Ahora, para usarla, tengo que integrar los métodos:

```
(silp1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ HTTP_METHOD=POST ; LAMBDA_NAME=user_lambda_param
(silp1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ 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/${LAMBDA_NAME}/invocations \
--passthrough-behavior WHEN_NO_MATCH
{
  "type": "AWS_PROXY",
  "httpMethod": "POST",
  "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/user_lambda_param/invocations",
  "passthroughBehavior": "WHEN_NO_MATCH",
  "timeoutInMillis": 29000,
  "cacheNamespace": "tm376ukikv",
  "cacheKeyParameters": []
}
(silp1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ [ $? == 0 ] || echo "Failed: AWS / apigateway / put-integration"
(silp1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal apigateway get-resources --rest-api-id ${API_ID}
{
  "items": [
    {
      "id": "j9gju7e7ca",
      "path": "/"
    },
    {
      "id": "ffbw18tmi4",
      "parentId": "j9gju7e7ca",
      "pathPart": "user",
      "path": "/user"
    },
    {
      "id": "tm376ukikv",
      "parentId": "ffbw18tmi4",
      "pathPart": "{username}",
      "path": "/user/{username}",
      "resourceMethods": {
        "POST": {
          "httpMethod": "POST",
          "authorizationType": "NONE",
          "apiKeyRequired": false,
          "requestParameters": {
            "method.request.path.username": true
          },
          "methodResponses": {},
          "methodIntegration": {
            "type": "AWS_PROXY",
            "httpMethod": "POST",
            "uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/user_lambda_param/invocations",

```

Después de esto, la función ya está integrada y lista para su uso.

Tarea: Creación de una tabla

Ejecuta en un terminal:

```
$> awslocal dynamodb create-table \
--table-name silp1 \
--attribute-definitions AttributeName=User,AttributeType=S \
--key-schema AttributeName=User,KeyType=HASH \
--region ${REGION} \
--provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "User",
                "AttributeType": "S"
            }
        ],
        "TableName": "si1p1",
        "KeySchema": [
            {
                "AttributeName": "User",
                "KeyType": "HASH"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": 1697739452.426,
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:000000000000:table/si1p1",
        "TableId": "0e9b220f-6a9e-452b-94fa-8325a4d5b599"
    }
}
```

DynamoDB

Pregunta: Creación de una tabla

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

Las definiciones de atributo (User) son de tipo S (string), mientras que las claves que se usan para buscar en la tabla son de tipo HASH.

Tarea: Insertando datos manualmente

DynamoDB requiere que al insertar un dato indiquemos su tipo, pero este no se deduce automáticamente, de ahí que para indicar el valor de cada atributo se use la sintaxis { atributo: { Tipo : valor }}

Ejecuta en un terminal:

```
$> awslocal dynamodb put-item \
--table-name si1p1 \
--item '{"User": {"S":"pipo"}, "EMail": {"S": "pipo@ss.com"}}'
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal dynamodb put-item \
--table-name si1p1 \
--item '{"User": {"S":"pipo"}, "EMail": {"S": "pipo@ss.com"}}'
```

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal dynamodb scan --table-name si1p1 --region us-east-1
{
    "Items": [
        {
            "EMail": {
                "S": "pipotestes.com"
            },
            "User": {
                "S": "pipotestes"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 1,
    "ConsumedCapacity": null
}
```

Pregunta: Insertando datos manualmente

1. ¿Qué ocurre si creamos un item sin campo "User" o sin campo "Email"?

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal dynamodb put-item --table-name si1p1 --item '{"EMail": {"S": "pipotestes.com"}}'
An error occurred (ValidationException) when calling the PutItem operation: One of the required keys was not given a value
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal dynamodb put-item --table-name si1p1 --item '{"User": {"S": "pipotestes"}}'
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal dynamodb scan --table-name si1p1 --region us-east-1
{
    "Items": [
        {
            "User": {
                "S": "pipotestes"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 1,
    "ConsumedCapacity": null
}
```

Sin el campo User se obtiene un error de validación, debido a que al crear la tabla, hemos definido que su atributo es User, pero si omitimos Email no habría problema ya que no es un requisito en los atributos de la tabla. DynamoDB permite campos adicionales de ahí que se pueda insertar un elemento con el atributo Email además del de User.

Tarea: Lectura de datos

Según el esquema que hemos definido, DynamoDB usará el campo User como clave principal (o de particionado) de la tabla "si1p1".

Ejecuta en un terminal:

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

Pregunta: Lectura de datos

2. ¿Qué ocurre si creamos un item sin campo "User"?

Daría error de validación, ya que es un atributo definido en la tabla.

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal dynamodb get-item --table-name si1p1 --key '{"EMail": {"S": "pipotestes.com"}}'
An error occurred (ValidationException) when calling the GetItem operation: One of the required keys was not given a value
```

3. ¿Qué ocurre si volvemos a insertar el mismo usuario sin el campo "Email"?

No hay problema ya que la tabla no tiene restricciones con duplicados, se tratarían como elementos diferentes.

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ awslocal dynamodb get-item --table-name si1p1 --key '{"User": {"S":"piro"}}'
{
    "Item": {
        "EMail": {
            "S": "piro@ss.com"
        },
        "User": {
            "S": "piro"
        }
    }
}
```

Tarea: Lambda con dynamoDB

Crea la lambda correspondiente a `user_lambda_dynamo.py`, siguiendo las instrucciones anteriores, e intégrala en el método POST de `user/{username}`, donde antes teníamos `user_lambda_param.py`.

Tras ello, ejecuta en un terminal:

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

Nota: La nueva lambda ya usa el parámetro `username`, que guardará en el campo `User` de la tabla, por lo que no es necesario modificarla. En el cuerpo del evento, espera datos con el atributo `"email"`, que guardará en el campo `"Email"` de la tabla.

Pregunta: Insertando datos manualmente

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

Lo hemos hecho de la misma manera que para `user_lambda_param`, no lo volvemos a poner por evitar repetición.

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

Obtenemos el siguiente error:

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ curl -X POST ${ENDPOINT}/user2 -H 'Content-Type: application/json' \
-d '{"email": "mimail@dominio.es"}'
{"msg": "error trace Traceback (most recent call last):\n  File \"/var/task/user_lambda_dynamo.py\", line 34, in handler\n    req = json.loads(event['body'])\n  File \"/var/lang/lib/python3.9/json/_init_.py\", line 346, in loads\n    return _default_decoder.decode(s)\n  File \"/var/lang/lib/python3.9/json/decoder.py\", line 337, in decode\n    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
  File \"/var/lang/lib/python3.9/json/decoder.py\", line 355, in raw_decode\n    raise JSONDecodeError(\"Expecting value\", s, err.value) from None
json.decoder.JSONDecodeError: Expecting value: line 1 column 11 (char 10)(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$
```

Tarea: GET /user/{username} con dynamoDB

Usando como ejemplo la lambda `user_lambda_dynamo.py`, crea e integra `user_get_lambda_dynamo.py`. En este caso intégrala en el método GET de `user/{username}`, el cual no tenía ninguna lambda asociada.

La lambda `user_get_lambda_dynamo` ha de buscar el usuario y devolver el objeto encontrado en el cuerpo de la respuesta (puedes usar la variable `rest_body` para ello). Si no se encuentra se devolverá un objeto vacío.

Notas de implementación:

- La entrada y salida del método `get_item` en `boto3` es muy similar a la que hemos usado anteriormente en línea de comando. En la entrada se usa `Key={'clave': valor}`, y la respuesta estará en la clave `"Item"`, si se encuentra. Para más detalles consulta la documentación del método `get_item` en el manual de referencia de `boto3` para dynamoDB.
- Como entrada a esta lambda ya no se usará la clave `"body"` del evento, que no existirá, por lo que dará error si se intenta acceder a `event['body']`

Pregunta: GET /user/{username} con dynamoDB

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

Para crear la función, usando la función `user_lambda_dynamo` como marco, hemos creado una función con control de excepciones que lee el usuario de la tabla con la

función `get_item` dada la key (`username`) y luego codifica la respuesta (el item si se ha encontrado o un objeto vacío).

Para integrarla, procedemos de igual manera que en los apartados anteriores.

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.

Para probar el método GET podemos usar `curl -X GET /user/{username}`.

Con un usuario existente, obtenemos sus atributos::

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ curl -X GET ${ENDPOINT}/piro
{"EMail": "piro@ss.com", "User": "piro"}(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ 
```

Con uno inexistente, obtenemos un conjunto vacío:

```
(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ curl -X GET ${ENDPOINT}/piro
{}(si1p1) gonzalo@gonzalo:~/SI1_PR/p1-v1.0$ 
```

Archivos empleados

user_rest

Define nuestra aplicación quart con la API rest. Especifica algunos métodos http para esta aplicación.

Dockerfile

Fichero que define la creación de la imagen que usamos en el registro local.

user_lambda

Función lambda para el método POST general sin mejoras.

user_lambda_param

Función lambda para el método POST general mejorada con parámetros.

user_lambda_dynamo

Define la función lambda para el método POST que se usa en la parte de DynamoDB.

user_get_lambda_dynamo

Define la función lambda para el método GET que se usa en la parte de DynamoDB.