MEMORIA DE SISTEMAS INFORMÁTICOS I Por Bernardo Andrés Zambrano Ferreira

Práctica 1

Parte 1

Pregunta: user_rest.py

1. **¿Qué crees que hacen las anotaciones @app.route, @app.get, @app.put?**
    - @app.route: Asocia una función con una ruta específica en Quart para manejar solicitudes HTTP GET por defecto.
    - @app.get: Define un manejador para solicitudes HTTP GET asincrónicas en una ruta específica.
    - @app.put: Define un manejador para solicitudes HTTP PUT asincrónicas en una ruta específica.
2. **¿Qué ventajas o inconvenientes crees que pueden tener?**
    - **Ventajas**: Las anotaciones proporcionan una sintaxis clara y declarativa para definir rutas y manejar solicitudes, facilitando la organización del código y la comprensión de la lógica de la aplicación.
    - **Inconvenientes**: Puede haber una curva de aprendizaje para desarrolladores no familiarizados con anotaciones o decoradores en Python.

Pregunta: ejecución de user_rest.py

1. **¿Qué crees que ha sucedido?**
    - Los comandos inician un entorno virtual, crean un directorio "build/users" si no existe, y ejecutan una aplicación Quart llamada `src.user_rest:app` en el puerto 5050, configurando así el entorno y la estructura antes de ejecutar la aplicación.
2. **¿Qué crees que ha sucedido al ejecutar el mandato curl?**
    - El comando curl envía una solicitud HTTP PUT al servicio user_rest.py en la URL http://localhost:5050/user/myusername. La solicitud incluye datos JSON que contienen información sobre el usuario, como el nombre.
3. **¿Cuál es la respuesta del microservicio?**
    - {"status":"OK"}
4. **¿Se ha generado algún archivo? ¿Cuál es su ruta y contenido?**
    - Sí, se ha generado el archivo data.json en la ruta build/users/myusername y su contenido es el siguiente: {"firstName": "myFirstName", "username": "myusername"}

Pregunta: petición HTTP

1. **¿Cuál es la petición HTTP que llega al programa nc?**
    - La petición es la siguiente:

```
PUT /user/myusername HTTP/1.1
Host: localhost:8888
User-Agent: curl/7.68.0
Accept: */*
Content-Type: application/json
Content-Length: 28
```

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?**
    - Se indica en el campo Content-Type y en este caso es application/json.
3. **¿Qué separa la cabecera del cuerpo?**
    - La cabecera y el cuerpo de una solicitud HTTP están separados por una secuencia de dos caracteres de nueva línea (\r\n), que indica el final de la cabecera y el comienzo del cuerpo.

Pregunta: ejecución hypercorn

1. **¿Qué crees que ha sucedido?**
    - El comando `hypercorn --bind 0.0.0.0:8080 --workers 2 src.user_rest:app` inicia el servidor Hypercorn para ejecutar la aplicación Quart. Está escuchando en todas las interfaces de red en el puerto 8080 y utiliza 2 workers para mejorar la capacidad de manejar solicitudes simultáneas.
2. **¿Qué petición curl debes hacer ahora para hacer un GET de un usuario? ¿Qué ha cambiado respecto de la ejecución sin hypercorn?**
    - La petición es `curl <http://localhost:8080/user/{username}>`. Con Hypercorn, se utiliza un servidor ASGI más adecuado para entornos de producción, mejorando el manejo de solicitudes concurrentes con 2 workers y escuchando en todas las interfaces en el puerto 8080.

Pregunta: modo rootless

1. **Revisa el contenido de /etc/subuid y la documentación de docker rootless: ¿para qué crees que sirve ese archivo?**
    - /etc/subuid es un archivo en sistemas Linux utilizado en el modo "rootless" de Docker. Define rangos de IDs de usuario secundarios (subuids) permitidos para usuarios específicos, asignando IDs de usuario de manera segura en contenedores sin privilegios de root en el sistema host.

Pregunta: docker info

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

- En el archivo `daemon.json` dentro del directorio de configuración del usuario, en este caso, /home/myuser/.local/share/docker.

2. **¿Dónde se almacenan los contenedores?**

   - En el directorio de Docker específico del usuario, en este caso, /home/bernardo/.local/share/docker/containers/.

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

   - La tecnología utilizada es overlay2. Copy-on-write (COW) es una estrategia donde los datos se copian solo cuando se modifican, optimizando el uso del espacio y las operaciones de copia y escritura. Docker utiliza COW a través del controlador de almacenamiento Overlay2.

Pregunta: run whoami

1. **¿Qué crees que ha sucedido?**

   - Docker ha descargado la imagen de Alpine, ha creado un contenedor basado en la imagen y ha ejecutado el comando `whoami` dentro del contenedor.

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

   - No, el root dentro del contenedor Alpine no es el mismo que el root del host.

Pregunta: list images

1. **¿Qué crees que tiene que ver con el mandato docker pull?**
   - El comando `docker images` lista las imágenes presentes localmente. Está relacionado con `docker pull`, que se usa para descargar nuevas imágenes desde un registro remoto, y estas imágenes descargadas se reflejan en la salida de `docker images`.

Pregunta: docker ps

1. **¿Qué crees que muestra la salida?**
   - El comando `docker ps -a` muestra una lista detallada de todos los contenedores, incluidos los que están en ejecución y los que han sido detenidos.

Pregunta: docker pull y run -ti

1. **¿Qué hacen las opciones -ti?**

   - `-t` asigna una terminal al contenedor y `-i` hace que la terminal sea interactiva.

2. **¿Qué ha sucedido?**

   - Se ha creado y ejecutado un contenedor interactivo basado en la imagen de Ubuntu con la etiqueta "22.04", permitiendo la interacción directa con el sistema operativo del contenedor. El contenedor se ha nombrado como "u22.04".

Pregunta: docker inspect

1. **¿Qué hace el mandato jq?**
    - `jq` es una herramienta de procesamiento de datos JSON en la línea de comandos. En este contexto, está formateando y filtrando la salida JSON generada por el comando `docker inspect`. La expresión `.[] .Config.Cmd[]` extrae y lista los elementos del array `Cmd` en la configuración del contenedor.
2. **¿Qué tiene que ver el resultado del mandato con la tarea anterior?**
    - El resultado muestra el comando que se ejecutó al iniciar el contenedor, proporcionando información sobre qué proceso o aplicación se inició en el contenedor creado con `docker run -ti --name u22.04 ubuntu`.

Pregunta: docker start

1. **¿Qué ha sucedido?**
    - El comando `docker start u22.04` inicia el contenedor llamado "u22.04" que fue previamente creado y detenido.
2. **¿Cómo se borra el contenedor u22.04?**
    - Para borrar el contenedor "u22.04", se puede utilizar el comando: `docker rm u22.04`.

Pregunta: docker run -d

1. **¿Qué hace la opción -d?**
    - Cuando se usa `-d`, el contenedor se inicia y se ejecuta en segundo plano, liberando el terminal del host para que puedas seguir utilizando la consola sin que esté vinculada al contenedor.

Pregunta: docker exec

1. **¿Qué ha sucedido?**
    - Dentro del contenedor "u22.04d", se ha creado un directorio llamado "users" y dentro de ese directorio se ha creado un archivo llamado "test".

Pregunta: docker volume

1. **¿Qué hace la opción -v?**
    - La opción `-v` en el comando `docker run` se utiliza para montar un volumen en el contenedor. En este caso, se está montando el directorio local `${PWD}/si1/users/testv` en el contenedor en la ruta `/si1/users/testv`.
2. **¿Qué ha sucedido al ejecutar los mandatos anteriores?**
    - Se han creado directorios y archivos tanto en el sistema de archivos local como en el contenedor, y se ha mostrado el contenido de esos archivos a través de comandos ejecutados en el contenedor y en el sistema local.

Pregunta: limpieza

1. **¿Qué mandato has usado para eliminar los contenedores?**
    - `docker rm -f $(docker ps -aq)`
2. **¿Qué mandato has usado para eliminar las imágenes?**
    - `docker rmi -f $(docker images -q)`

Pregunta: Dockerfile

1

. **¿Qué hace la sentencia FROM?**

- Establece la imagen base para construir la nueva imagen.

2. **¿Qué hace la sentencia ENV?**
    - Establece variables de entorno en la imagen.
3. **¿Qué diferencia presenta ENV respecto de ARG?**
    - `ENV` establece variables disponibles durante la construcción y en los contenedores derivados. `ARG` define argumentos disponibles solo durante la construcción.
4. **¿Qué hace la sentencia RUN?**
    - Ejecuta comandos en una nueva capa de la imagen durante la construcción.
5. **¿Qué hace la sentencia COPY?**
    - Copia archivos o directorios desde el sistema de archivos del host al contenedor durante la construcción.
6. **¿Qué otra sentencia del Dockerfile tiene un cometido similar a COPY?**
    - `ADD` tiene funcionalidades adicionales como descomprimir archivos automáticamente y copiar desde una URL.
7. **¿Qué hace la sentencia EXPOSE?**
    - Indica los puertos en los que el contenedor escuchará durante el tiempo de ejecución.
8. **¿Qué hace la sentencia CMD?**
    - Proporciona un comando predeterminado que se ejecutará cuando se inicie un contenedor basado en la imagen.
9. **¿Qué otras sentencias del Dockerfile tienen un cometido similar a CMD?**
    - `ENTRYPOINT` proporciona un comando predeterminado, pero no se puede anular al ejecutar el contenedor.

Pregunta: docker build

1. **¿Por qué es mucho más rápida la creación de la segunda imagen?**
    - La creación más rápida de la segunda imagen se debe al sistema de caché de Docker, reutilizando capas existentes desde la caché.
2. **Las imágenes constan de distintas capas: ¿Qué quiere decir esto?**
    - Cada instrucción en un Dockerfile crea una nueva capa en la imagen. Las capas son reutilizables y se almacenan en caché, acelerando la construcción al evitar repetir pasos ya realizados.

Pregunta: docker build with user

1. **¿Por qué en los últimos pasos no utiliza la caché?**
    - Porque al introducir cambios en el Dockerfile se invalida la caché de ese punto en adelante.

Pregunta: docker run myimage

1. **¿Cuántos subprocesos de hypercorn aparecen? ¿Por qué?**
    - Aparecen 5 subprocesos, ya que se establecen 5 workers para gestionar las peticiones.
2. **¿Qué hace la opción '-p' del comando docker run?**
    - Mapea los puertos del host al puerto definido del contenedor, en este caso, mapeando el puerto del host 8080 con el 8000 del contenedor.
3. **Si deseáramos ejecutar otra réplica (contenedor) del microservicio, ¿qué deberíamos cambiar en la sentencia docker run?**
    - Cambiar el nombre del contenedor y el puerto de mapeo del host y del contenedor.

Pregunta: docker run myimage on ./si1

1. **¿Qué mandato has ejecutado?**
    - `mkdir ./si1` y posteriormente `docker run --name si1p1_replica2 -e NUMWORKERS=5 -d -p 8081:8000 -v $(pwd)/si1:/si1 si1p1`
2. **De cara a la realización de backups, ¿cuál crees que es la utilidad de montar volúmenes externos respecto de usar los internos de docker?**
    - Montar volúmenes externos ofrece persistencia de datos independiente de los contenedores, facilita copias de seguridad, es compatible con herramientas externas de backup y simplifica procesos de migración y actualización.

Pregunta: docker registry

1. **¿Qué mandato has usado para descargar la imagen del registry?**
    - `docker pull registry`

2. **¿Cuál es el número de versión de dicha imagen?**
    - La versión de la imagen es `latest`.
3. **¿Qué mandato has usado para ejecutar el contenedor del registry?**
    - `docker run --name si1_registry -d -p 5050:5000 registry:latest`
4. **¿En qué puerto escucha por defecto el contenedor del registry?**
    - Por defecto, el registry escucha en el puerto 5000 dentro del contenedor.

Pregunta: docker tag push

1. **¿Qué ha sucedido?**
    - He tomado la imagen de Ubuntu, le he asignado una nueva etiqueta específica para mi registro local y luego la he empujado a mi registro local para que esté disponible para su uso en ese registro.

Pregunta: docker app from local

1. **¿Qué mandatos has usado?**
    - `docker build --tag localhost:5000/si1p1:latest .`
    - `docker push localhost:5000/si1p1:latest`
    - `docker run --name si1_app -d -p 8082:8000 localhost:5000/si1p1:latest`

Parte 2

Pregunta: AWS S3

1. **¿Qué hacen los distintos subcomandos de awslocal s3?**
    - `mb`: Crea un nuevo bucket.
    - `cp`: Copia archivos o directorios a/desde un bucket de S3 local.
    - `ls`: Lista el contenido de un bucket o carpeta en S3 local.
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://bucket_name/user_rest.py local_user_rest.py --no-clobber`

Pregunta: Lambdas

1. **¿Qué runtime podríamos usar si por ejemplo queremos implementar una función Lambda en JavaScript?**
    - Node.js, por ejemplo, `nodejs14.x`.
2. **¿Cómo recibe la función lambda los datos que hemos enviado con el comando curl?**
    - A través de la carga útil (payload) de la solicitud HTTP.

3. **Comprueba que ha funcionado descargando del bucket el fichero creado por la función. ¿Qué comandos has utilizado para ello?**
    - `awslocal s3 cp s3://si1p1/users/pepe local/`

Depuración Lambdas

Pregunta: Depuración Lambdas

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

1. **docker logs localstack_main**
    - Muestra los logs del contenedor Docker "localstack_main".
2. **awslocal logs describe-log-groups**
    - Muestra los grupos de logs en el ambiente LocalStack.
3. **awslocal logs describe-log-streams --log-group-name /aws/lambda/${LAMBDA_NAME}**
    - Muestra los flujos de logs asociados con un grupo de logs específico para una función Lambda.
4. **awslocal logs describe-log-streams --log-group-name /aws/lambda/${LAMBDA_NAME} | jq '.logStreams[].firstEventTimestamp' | sort -n**
    - Muestra los timestamps del primer evento de cada flujo de logs, ordenados numéricamente.
5. **LLOGSTREAMS=$(awslocal logs describe-log-streams \ --log-group-name /aws/lambda/${LAMBDA_NAME} | jq -r '.logStreams[].logStreamName')**
    - Asigna a `LLOGSTREAMS` los nombres de todos los flujos de logs de la función Lambda.
6. **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'**
    - Recorre cada flujo de logs en `LLOGSTREAMS`, recupera sus eventos y filtra los mensajes usando `jq`.

Tarea: Borrado de Funciones

Pregunta: Borrado de Funciones

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

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

Tarea: Creación del API

Pregunta: Creación del API

1. ¿Qué ID se ha asignado al API?

El ID asignado al API es "g1h2j3klm".

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

LocalStack permite crear múltiples APIs con el mismo nombre. Cada API creada obtiene un ID único, independiente del nombre.

Tarea: Creación de las Rutas

Pregunta: Creación de las Rutas

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

El ID asignado al recurso con la ruta /user/{username} es "x7y8z9pqr".

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

API Gateway no permite crear la ruta /user/{name} si ya existe /user/{username} en el mismo segmento de ruta.

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

No tiene sentido desde el punto de vista del diseño de API tener ambas rutas ya que sería ambiguo determinar cuál manejaría una solicitud específica.

Tarea: Creación de Métodos

Pregunta: Creación de Métodos

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

Para crear el método GET:

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

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

Para ver los métodos asociados a las rutas del API:

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

Tarea: Integración

Pregunta: Integración

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

Para integrar el método GET:

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

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

Para verificar la integración de cada método del API:

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

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

Para borrar la integración del método GET:

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

Tarea: Despliegue

Pregunta: Despliegue

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

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

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?

Para borrar el despliegue no asociado a la etapa "dev":

```
awslocal apigateway delete-deployment --rest-api-id ${API_ID} --deployment-
id abcdef1234
```

Tarea: Pruebas del API

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 crea un archivo distinto dentro de /users.

Tarea: Mejora del API

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.

En si1p1/src:

```
cp user_lambda.py user_lambda_param.py
zip ../build/user_lambda_param.zip user_lambda_param.py
```

Crear la nueva función lambda:

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

Integrar y desplegar la nueva función lambda de la misma manera que la anterior.

Tarea: Creación de una Tabla

Pregunta: Creación de una Tabla

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

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

Tarea: Insertando Datos Manualmente

Pregunta: Insertando Datos Manualmente

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

- Si se intenta crear un ítem sin el campo User, se produce un error porque User es la clave primaria y es obligatoria.
- Si se omite el campo EMail, DynamoDB crea el ítem sin ese campo porque no es obligatorio.

Tarea: Lectura de Datos

Pregunta: Lectura de Datos

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

- Si se intenta crear un ítem sin el campo User, DynamoDB devuelve un error porque User es la clave principal y es obligatorio.

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

- El nuevo ítem sobreescribe al ítem existente que tiene el mismo valor de clave principal.

Tarea: Lambda con DynamoDB

Pregunta: Lambda con DynamoDB

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

Empaquetar y crear la función lambda con `user_lambda_dynamo.py`:

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

Integrar y desplegar la nueva función lambda:

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

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

Se añaden correctamente el email y el usuario.

Tarea: GET /user/{username} con DynamoDB

Pregunta: GET /user/{username} con DynamoDB

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

Implementar y empaquetar la función lambda:

```
zip ../build/user_get_lambda_dynamo.zip user_get_lambda_dynamo.py
```

Crear la función lambda:

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

Integrar y desplegar:

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

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:

```
curl -X GET
"http://localhost:4566/restapis/${API_ID}/${STAGE}/_user_request_/user/{use
rname}"
```

- **Si el usuario existe**:

```
{
  "Item": {
    "EMail": {
      "S": "pipo@ss.com"
    },
    "User": {
      "S": "pipo"
    }
  },
  "Count": 1,
  "ScannedCount": 1,
  "ConsumedCapacity": null
}
```

- **Si el usuario no existe**:

```
{}
```