AWS S3 , CLI y SDK Python

Esta práctica tiene dos objetivos: 1) conocer el servicio S3 y 2) conocer cómo podemos interactuar con los servicios de AWS sin utilizar la consola WEB.

El servicio S3 es un servicio de almacenamiento de objetos con alta disponibilidad, escalabilidad, rendimiento y seguridad. S3 permite alojar grandes lotes de información con un coste menor que los discos duros EBS. Habitualmente se utiliza para guardar backups, archivos multimedia.

Por otro lado el CLI (Command Line Interfaz) de amazon nos permite interactuar mediante comandos Bash con los servicios de AWS. El SDK de AWS nos permite interactuar con los servicios de AWS desde nuestras aplicaciones. Existem SDK para diferentes lenguajes de programación (Python , Java , C# , Javascript, Php,...).

En esta práctica crearemos una aplicación que permite guardar imágenes en el servicio S3 de Amazon. Utilizaremos S3 con dos objetivos: 1) Guardar las imágenes 2) Alojar la página web estática del frontend. Trabajaremos la creación de buckets, publicación de ficheros , alojamiento de páginas estáticas y configuración de políticas. Crearemos buckets (carpetas S3) tanto mediante la consola como mediante el CLI. Y desarrollaremos una pequeña aplicación que permitirá acceder a las imágenes.

El servicio AWS S3: Creación de bucket, politicas de acceso, carga de ficheros

Lo primero que haremos será crear en S3 la carpeta que guardará las imágenes de la aplicación.

1- Ir al servicio S3 y seleccionar crear bucket.

2- Le daremos un nombre al bucket (Este nombre tiene que ser único) y dejaremos todos las opciones por defecto por el momento (Acceso privado , sin encriptación , etc). En nuestro caso le hemos llamado al bucket ‘practica-s3-macc’. Cada bucket dispone de un ARN (Amazon Resource Name) único. En este caso es arn:aws:s3:::practica-s3-macc.

3- Una vez creado el bucket podemos guardar , crear carpetas dentro del bucket. Si clickamos accederemos a él y podremos ver los diferentes menús que nos ofrecen : 1)objetos 2)propiedades 3)Permisos 4)métricas 5)Administración. Mediante estos menús podemos configurar el acceso al bucket y en mediante el Objetos podemos ver el contenido , borrarlo y crear.

4- Una vez creado el bucket crearemos una carpeta para alojar fotos.Para ello seleccionar ‘crear carpeta’, llamaremos a la carpeta ‘fotos’ y dejaremos las opciones por defecto. Al crearlo vemos que la carpeta dispone de un URI, mediante el cual podremos acceder a los datos. En este caso el URI es ‘s3://practica-s3-macc/fotos/’

5- Una vez creada la carpeta subiremos una foto a la carpeta.Para ello seleccionamos el botón Cargar o Upload , dependiendo del idioma en el que estéis. Seleccionamos cargar fichero y elegimos una imagen que tengamos en local.

6- Una vez cargada la imagen si clickamos sobre ella podremos realizar diferentes acciones y ver sus propiedadesViendo las propiedades podemos ver que se ha genarado tanto un URI S3 (s3://practica-s3-macc/fotos/html.png) como una URL de acceso (https://practica-s3-macc.s3.amazonaws.com/fotos/html.png)

7-Para acceder a la foto vale con utilizar la URL. Ponemos la URL en el navegador veremos que nos indica que nos deniega el acceso. Esto ocurre porque el Bucket que hemos creado es privado. Para acceder a esta información debemos de estar logeados y por eso desde la consola de AWS podemos hacer diferentes acciones (por ejemplo descargar el fichero , eliminar modificar), pero desde el navegador no permitimos verlo. También podríamos descargarlo previo login desde el command line CLI (luego lo haremos), o también podríamos permitir que los usuario vieran la imagen a través de un servidor web que tiene credenciales AWS que le permiten descargar la imagen y luego este servidor nos la presentaría en una página web (Lo haremos más adelante en este práctica). De momento modificaremos los permisos de la imagen y la haremos pública. Para ver los permisos iremos a la lista de Acceso (ACL) del objeto y vemos que solo el propietario tiene permisos de lectura. Para poder hacer público este objeto primero tenemos que indicar al Bucket que podemos dar acceso público y después podréis modificar los permisos públicos de cada objeto.

8- Primero desactivaremos el bloqueo de acceso público en el Bucket (esto lo haríamos solo si quisiéramos publicar datos). Vamos al Bucket , lo seleccionamos y en el menú de permisos clickamos Editar y de momento des seleccionamos el bloqueo público.

9- Ahora si volvemos a intentar ver la foto desde el navegador seguimos sin permisos porque nos falta activar el acceso público a la foto. Esto se puede hacer por carpeta o por objeto. Para modificar los permisos del objeto vamos a la sección de ACL del elemento. En la parte inferior de la página y seleccionamos Editar y seleccionamos los permisos de lectura del fichero. Y si ahora probamos la URL de la imagen podremos ver la imagen sin ningún tipo de login.

10- En el último paso hemos configurado los permisos de un fichero mediante las políticas de bucket podemos especificar los permisos sobre el bucket entero , una carpeta o un objeto. Nuestro siguiente objetivo es que todos los elementos de la carpeta fotos sean públicos y para ello generamos una política. Recordar que en las políticas utilizaremos el ARN del objeto/carpeta para especificar los permisos.

11- Para ello primero obtenemos el ARN de la carpeta (por ejemplo arn:aws:s3:::practica-s3-macc/fotos/). Y antes de crear la política subiremos una nueva imagen a la carpeta y comprobaremos que por defecto no lo vemos desde el navegador.

12- A continuación vamos al menu de permisos de Bucket y seleccionamos Política de Bucket. Las políticas en AWS se especifican mediante el formato JSON. En este caso

utilizaremos el generador de políticas para crear la plantilla del JSON. Indicamos que queremos crear una politica de tipo S3. Despues seleccionamos ALLOW y como principal (usuarios) diremos que todos '*'. En acciones veremos podemos ser muy precisos a la hora de indicar la acción , en nuestro caso daremos acceso de lectura (GETOBJECT) y como ARN nuestra carpeta. Y le damos a agregar política (podemos crear las que queramos).Finalmente generamos la política (analizar el JSON), la copiamos y después la pegamos y guardamos la política. Volvemos a probar y vemos que todavia no vemos la imagen , eso es porque le hemos dado permiso a la carpeta y no a los objetos, agregar '*' en el ARN de la política 'arn:aws:s3:::practica-s3-macc/fotos/*' y ahora todo lo albergado en la carpeta fotos está disponible ;-)

13- Ahora veremos la propiedad de versiones. Para ello crea un txt con un texto y subirlo al bucket,en la ubicación que queráis. Después habilitamos el control de versiones.

14- Subimos una nueva versión de un fichero. Después navegamos en AWS hasta el fichero y seleccionamos el menú versiones y ahí podemos acceder a la version que queramos

Utilizando el CLI para interactuar con servicios AWS: EC2 y S3

15- Hasta ahora hemos interactuado con los servicios de AWS (EC2, VPC , S3) mediante la consola WEB. Para conocer y consultar los servicios la consola web es suficiente, pero si queremos comenzar a automatizar operaciones es necesario poder ejecutar acciones desde una consola tipo bash o mediante un lenguaje de programación. En los siguientes pasos interactuamos con el servicio S3 mediante el CLI (Command Line Interface) y el SDK de Python. Existen otras herramientas más avanzadas para gestionar de forma automatizada infraestructura de AWS como podrían ser el servicio Cloudformation de AWS , o herramientas de aprovisionamiento de infraestructura como Terraform o herramienta sde aprovisionamiento de servidores como Ansible , Chef o Puppet. Este tipo de herramientas los utilizaremos más adelante en el PBL.

16- Comenzaremos utilizando el CLI de AWS. El CLI de AWS es una aplicación de consola que nos permite interactuar desde el bash de nuestro pc (o cualquier ordenador ;-) con los servicios AWS de nuestra cuenta. Para poder utilizar el CLI e interactuar con AWS solo tendremos que instalar el CLI en un ordenador y configurar para que utilice nuestras credenciales. Primero instalaremos el CLI. El CLI de AWS existe para diferentes distribuciones (Windows , GNU/Linux, MAC). Nosotros lo instalaremos en Linux. En Linux se puede instalar el CLI de Amazon tanto desde Apt como utilizando el instalable que nos ofrece Amazon.

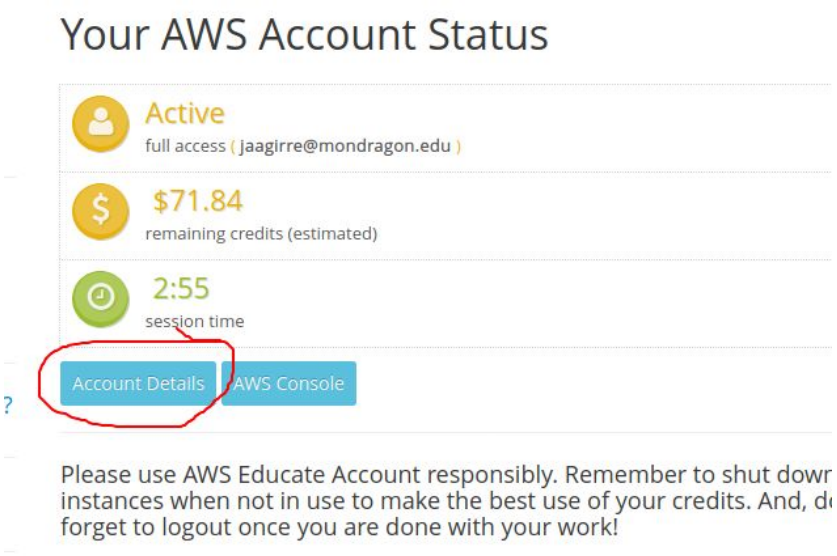
a)

```
sudo apt-get install python3
sudo apt-get install python3-pip
sudo apt-get install awscli
aws --version
```

b)

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
aws --version
```

17- Una vez tenemos instalado el CLI lo siguiente es configurar las credenciales de AWS del usuario a utilizar. En AWS dentro de una cuenta los usuarios se crean y configuran en el servicio IAM. En AWS Educate el servicio IAM esta un poco capado, pero en la página de acceso al classroom podréis ver vuestras credenciales en 'Account details' (ver imagen). En Educate os ofrecen un Token de de 3h que os permitirá utilizar desde el CLI todos los servicios disponibles.



18- Para utilizar las credenciales desde el CLI ejecutar e indicar el AWS Access Key ID (el usuario) y el AWS Access Key (el password). También indicareis la región de trabajo (us-east-1) y el formato de salida (JSON).

$ aws configure

Si ahora tratáis de ver todas las instancias EC2 que tenéis (mediante el comando de la tabla) os dará error porque vuestro acceso está limitado por Token y también tenéis que ofrecerlo.

$ aws ec2 describe-instances

19-Para configurar el Token de acceso lo más fácil es poner las credenciales en el fichero '.aws/credentials' , también se puede utilizar para esto la variable de entorno (AWS_SESSION_TOKEN). Por lo tanto, copiar las credenciales de vuestro classroom y copialo al fichero '.aws/credentials' y volver a ejecutar el comando que os debe listar todas las máquinas de la región en la que estáis con el comando del paso anterior.

20- En el cli ponemos *'aws nombre_del_servicios accion opciones'.* Si queremos ver las acciones de ec2 que tenemos disponibles escribir *'aws ec2 help'* y obtendréis una ayuda para ver todo lo que podemos hacer. Y esto con todos los servicios.

Por ejemplo aquí tenéis una serie de comandos que nos permiten parar la maquina que Ec2 que queramos y arrancarla. Es importante utilizar filtros y querys json para filtrar la información que queremos ver.

Por ejemplo si queremos el id de una instancia podemos indicar que propiedad del Json queremos visualizar. En el siguiente comando . Primero visual la información sin indicar la propiedad y después indicar que solo queremos ver el id de la instancia.

```
$ aws ec2 describe-instances > instancias.json
$ nano instancias.json
```

Ahora seleccionamos la propiedad id.

```
$ aws ec2 describe-instances --query 'Reservations[].Instances[].InstanceId'

[
          "i-0693a87b3dc0a4660",
         "i-09f542f4c88b88293",
          "i-0ec583f8a38dedb53",
          "i-02c3558456c0e6e26",
          "i-0f0f5696afd8fd23a",
          "i-0e140b66689741a79"
]
```

Si queremos ver la información de una solo instancia lo mejor es utilizar filtros y tags. En el siguiente comando solo veríamos la información de la instancia llamada práctica 1_1. Podríamos filtrar por cualquier tipo de tag (esto es una práctica habitual).

```
$ aws ec2 describe-instances --filters "Name=tag:Name,Values=practica1_1"
```

Y si solo queremos ver el id

```
$ aws ec2 describe-instances --filters "Name=tag:Name,Values=practica1_1" --query
'Reservations[].Instances[].InstanceId'

$aws ec2 describe-instances --filters "Name=tag:Name,Values=practica1_1" --query
'Reservations[].Instances[].PublicDnsName'
```

Aquí tenéis una combinación de comandos para poder parar una máquina en concreto en el que le diremos que no nos escriba la información en formato JSON sino en texto plano. Además dejaremos la información en una variable de entorno.

```
$ EC2_ID=`aws ec2 describe-instances --filters "Name=tag:Name,Values=practica1_1"
--query 'Reservations[].Instances[].InstanceId' --output text`

$EC2_NAME=`aws ec2 describe-instances --filters
"Name=tag:Name,Values=practica1_1" --query
'Reservations[].Instances[0].PublicDnsName' --output text`
```

```
$ aws ec2 describe-instances --filters "Name=tag:Name,Values=practica1_1"  --query
'Reservations[].Instances[].State'
$ aws ec2 stop-instances --instance-id ${EC2_ID}
$ aws ec2 describe-instances --filters "Name=tag:Name,Values=practica1_1"  --query
'Reservations[].Instances[].State'
$ aws ec2 start-instances --instance-id ${EC2_ID}


$ ssh -i ./keypairs/practicas_macc.pem ubuntu@${EC2_NAME}
```

Antes de comenzar el PBL se os dará un taller de automatización de infraestructuras donde se os darán diferentes recursos scripts, plantillas de cloudformation, playbooks de ansible y script de terraform para automatizar operaciones en AWS. Por ejemplo el siguiente script de la siguiente tabla para todas las instancias que tengan un tag concreto en una región en concreto (el tag podría se project=PBL, de forma que pararíamos todas las máquinas de un proyecto).

```
#!/bin/bash

if [ $# -eq 3 ]; then
    echo $1
    region=$1
    tag=$2
    value=$3
    echo "Lista de las instancias de la region   ${region}..."
    #Se asegura que s epueda terminar via API la instancia
    aws ec2 describe-instances --region $region \
        --filter Name=tag:$tag,Values=$value |\

        #jq -r .Reservations[].Instances[].InstanceId
        jq -r '.Reservations[].Instances[] | "id: " + .InstanceId + " state: " + .State.Name'
     aws ec2 describe-instances --region $region \
        --filter Name=tag:$tag,Values=$value  |\

        jq -r .Reservations[].Instances[].InstanceId |\
            xargs -L 1 -I {} aws ec2 stop-instances \
            --region $region \
            --instance-id {}
else
    echo "Error, el numero de parametros introducido no es correcto"
    echo "Utilizar: ./list-region-tag-ec2.sh region tag value"
fi
```

20- Ahora que hemos visto lo que podemos hacer con el CLI volvamos a S3. A continuación haremos unas simples operaciones con el CLI para interactuar con el servicio S3, por ejemplo crearemos un bucket copiaremos y descargamos fichero al bucket.

21- En la siguiente tabla se resumen algunos comandos S3. Para ver todo lo que podemos hacer utilizar el comando help ‘aws s3 help’.

Command	Purpose
Make a bucket	`aws s3 mb s3://my-bucket`
List all buckets	`aws s3 ls`
List the contents of a specific bucket	`aws s3 ls s3://my-bucket`
Upload a file to a bucket	`aws s3 cp file s3://my-bucket/file`
Download a file from a bucket	`aws s3 cp s3://my-bucket/file file`
Copy a file between buckets	`aws s3 cp s3://bucket1/file s3://bucket2/file`
Synchronize a directory with an S3 bucket	`aws s3 sync my-directory s3://my-bucket/`

22- Comencemos creando un bucker ‘aws s3 mb s3://macc-web-app’.

23- Para comprobar que el bucket se ha creado navegar en la consola web al servicio S3 o ejecutar ‘aws s3 ls’

24- De momento el bucket está vacío. Primero crearemos la carpeta ‘imagenes’.

```
$ aws s3api put-object --bucket macc-web-app --key imagenes/
$ aws s3 ls s3://macc-web-app/
$aws s3 ls s3://macc-web-app/imagenes --recursive
```

En el comando anterior hemos utilizado directamente el API de S3 porque mediante el comando S3 no se puede crear solo una carpeta eso sí al indicar el path de un fichero le damos el nombre de una carpeta que no existe a creará automáticamente. Ahora subimos un fichero.

```
$ aws s3 cp castle.jpeg s3://macc-web-app/imagenes/castle.jpeg
$ aws s3 ls s3://macc-web-app/imagenes --recursive
```

Para borrar todo el contenido de una carpeta podríamos ejecutar el siguiente comando.

```
$aws s3 ls s3://macc-web-app/imagenes --recursive
```

Interactuando con S3 mediante Python

25 - Una vez utilizado el CLI pasemos ahora a utilizar S3 mediante el SDK en aplicaciones python. En este caso crearemos una aplicación Flask que nos permite subir y visualizar fotos desde S3. La clave para todo esto es la utilización de la librería Boto3 de python.

26- Esta parte de la práctica también la podríamos realizar desplegando la aplicación Flask en una máquina no alojada en AWS ,por ejemplo vuestro PC. El desplegarlo en AWS , además de las ventajas propias del CLOUD , también nos permitirá restringir el ROLE (permisos) que le podríamos asignar a la máquina en su totalidad (para hacer esto si que tendríamos que desplegar la app en Amazon , por ejemplo en EC2). Por lo tanto crear una máquina virtual EC2, en este caso con acceso público (en produccion lo pondriamos detras de un API gateway o un Proxy, pero en este ejercicio por agilidad lo pondremos en la subred pública). La aplicación flask la pondremos en el puerto 5000 , por lo tanto , mediante el security group abrir los puertos 5000 y 22.

27- Ahora escribiremos el código de la aplicación antes de subirlo al servidor. Primero indicamos el nombre del bucket que utilizaremos y la carpeta donde guardaremos la fotos. Tener en cuenta el usuario con el que creais el bucket. Eso si no os olvidéis de crear el bucket (aws s3 mb s3://nombre_bucket).

```
#app.py
import os

from flask import Flask, render_template, request, redirect, send_file, url_for

from s3_demo import list_files, download_file, upload_file


app = Flask(__name__)
BUCKET = "practica-s3-macc"
```

28- Ahora antes de finalizar app.py , veamos el código que permite subir , listar y descargar elementos de S3. Esto lo haremos en el fichero *s3_demo.py*. Primero el código que permite subir ficheros junto con el import de boto3..

```
#s3_demo.py
import boto3
```

```
def upload_file(file_name, bucket):
        """
        Function to upload a file to an S3 bucket
        """
        object_name = file_name
        s3_client = boto3.client('s3')
        response = s3_client.upload_file(file_name, bucket, object_name)

        return response
```

Ahora el código que permite descargar.

```
def download_file(file_name, bucket):
        """
        Function to download a given file from an S3 bucket
        """
        s3 = boto3.resource('s3')
        output = f"downloads/{file_name}"
        s3.Bucket(bucket).download_file(file_name, output)

        return output
```

Y finalmente el código que lista el contenido de un bucket.

```
def list_files(bucket):
        """
        Function to list files in a given S3 bucket
        """
        s3 = boto3.client('s3')
        contents = []
        try:
        for item in s3.list_objects(Bucket=bucket)['Contents']:
        print(item)
        contents.append(item)
        except Exception as e:
        pass

        return contents
```

29- Ahora creamos los endpoint de flask que permitirán subir, bajar y listar fotos , además de ofrecer un endpoint que renderizada una sencilla página web. Todo esto en el modulo app.py

```
@app.route('/')
def entry_point():
        return 'Galería S3 demo. URL /storage'
```

```
@app.route("/storage")
def storage():
        contents = list_files(BUCKET)
        return render_template('storage.html', contents=contents)
```

```
@app.route("/upload", methods=['POST'])
def upload():
        if request.method == "POST":
        f = request.files['file']
        f.save(f.filename)
        upload_file(f"{f.filename}", BUCKET)

        return redirect("/storage")
```

```
@app.route("/upload", methods=['POST'])
def upload():
        if request.method == "POST":
        f = request.files['file']
        f.save(f.filename)
        upload_file(f"{f.filename}", BUCKET)

        return redirect("/storage")
```

No olvidar en agregar la línea de código que lanza el servidor

```
if __name__ == '__main__':
        app.run(host='0.0.0.0' , debug=True)
```

29- Ahora crearemos la página web que interactuara con estas funciones y que guardaremos en la carpeta templates/storage.html

```
<!DOCTYPE html>
<html>
  <head>
        <title>FlaskDrive</title>
  </head>

  <body>

        <div class="content">

        <h3>Aplicacon S3 Flask Demo</h3>

        <p>Galeria de fotos</p>

        <h2> {{ msg }}</h2>

        <div>
        <h3>Subir los ficheros:</h3>
        <form method="POST" action="/upload" enctype=multipart/form-data>
        <input type=file name=file>
        <input type=submit value=Upload>
        </form>
        </div>


        <div>
        <h3>These are your uploaded files:</h3>
        <p>Click on the filename to download it.</p>
        <ul>
        {% for item in contents %}
        <li>
                <a href="/download/{{ item.Key }}"> {{ item.Key }} </a>
        </li>
        {% endfor %}
        </ul>

        </div>

        </div>
  </body>
</html>
```

30- Ahora definimos las dependencias en el requirements.txt

```
boto3==1.9.235
botocore==1.12.235
Click==7.0
docutils==0.15.2
Flask==1.1.1
itsdangerous==1.1.0
Jinja2==2.10.1
jmespath==0.9.4
MarkupSafe==1.1.1
python-dateutil==2.8.0
s3transfer==0.2.1
six==1.12.0
urllib3==1.25.6
Werkzeug==0.16.0
```

31- Ya solo nos queda desplegar la aplicación. Para ello nos conectamos via SSH a la máquina EC2 creada anteriormente. Copiamos todo el código de la aplicación al servidor. Ahora instalamos python3, python3-pip y python3-venv (para la creación de entornos virtuales de python , esto en principio no es necesario , pero por si tenemos otras app de python en el servidor es fundamental para diferenciar diferentes entorno de ejecución). Instalamos todo vía apt. Tambien instalar awscli (para configurar las credenciales que utilizará la aplicación) y configurar las credenciales con las que se ejecutar la app (.aws/credentials)

32- Una vez instalado creamos el entorno virtual de python y nos adentramos en él.

```
$ python3 -m venv macc-web-app-adibidea
$ source macc-web-app-adibidea
(macc-web-app-adibidea) ubuntu@ip-10-0-0-8:~/
```

Bien ahora instalamos las dependencias via pip desde la carpeta de la aplicación

```
$ pip3 install -r requirements.txt
```

Y ahora solo nos queda ejecutar la aplicación . Como vamos a ejecutarlo como modulo python crear un __init__.py vacio para ello.

```
python -m app.py
```

33- Y para terminar probar la app ;-) subir y descargar fotos. Ahla los usuarios de vuestra aplicación (habría que crear un sistema de login y una BBDD) sin ser usuarios de AWS ni

tener cuenta en AWS podrán ver fotos alojadas en AWS mediante vuestro servidor y vuestra credencial ;-)