Instalación

Instalar la última versión de Boto 3 a través de pip:

```
pip install boto3
# También puede instalar una versión específica:
pip install boto3=1.0.0
```

Configuración

Antes de que puedas comenzar a usar Boto 3, debes configurar las credenciales de autenticación. Las credenciales para tu cuenta de AWS se pueden encontrar en la consola de IAM. Puedes crear o usar un usuario existente. Ve a administrar claves de acceso y genera un nuevo conjunto de claves.

Si tienes la AWS CLI instalada, puedes usarla para configurar tu archivo de credenciales:

```
aws configure
```

Alternativamente, puedes crear el archivo de credenciales tumismo. Por defecto, su ubicación es ~ / .aws / credentials::

```
[default]
aws_access_key_id = YOUR_ACCESS_KEY
aws_secret_access_key = YOUR_SECRET_KEY
```

También es posible que desees establecer una región predeterminada. Esto se puede hacer en el archivo de configuración. Por defecto, su ubicación está en ~ / .aws / config:

```
[default]
region=us-east-1
```

Alternativamente, puedes pasar un nombre_región al crear clientes y recursos.

Esto configura las credenciales para el perfil predeterminado, así como una región predeterminada para usar al crear conexiones.

Conceptos Básicos

Antes de comenzar a usar Boto, debes aprender algunos conceptos para comprender cómo usarlo facilmente.

Boto 3 está construido en la parte superior de una biblioteca llamada Botocore, que comparte con AWS CLI. Botocore proporciona los clientes de bajo nivel, sesión, y datos de credenciales y configuración. Boto 3 se basa en Botocore al proporcionar su propia sesión, recursos y colecciones.

- Los módulos generalmente se dividen en dos categorías, las que incluyen una interfaz orientada a **objetos de alto nivel** y las que incluyen solo una **interfaz de bajo nivel** que coincide con la API de Amazon Web Services (Estas interfaces son recursos y clientes respectivamente. un vistazo a los dos en breve).
- Algunos módulos son completamente de **alto nivel** (como Amazon S3 o EC2), algunos incluyen código

de alto nivel sobre una conexión de bajo nivel (como Amazon DynamoDB), y otros son **100% de bajo nivel** (como Amazon Elastic Transcoder).

Clientes: son conexiones de servicio de bajo nivel con las siguientes características principales:

- Soporta todos los servicios
- **Las salidas se devuelven utilizando diccionarios Python**
- **Tenemos que recorrer estos diccionarios nosotros mismos**
- Paginación automática de respuestas (Paginators)
- Una forma de bloquear hasta que se haya alcanzado cierto estado (camareros)

Echemos un vistazo a los dos puntos que he resaltado en negrita. Primero, usemos el Cliente EC2 para crear una instancia y ejemplificarlo mejor:

importar boto

```
import boto3aws_access_key_id = ''
aws_secret_access_key = ''
region_name = 'ap-southeast-2'session =
boto3.session.Session(aws_access_key_id=aws_access_key_id,

aws_secret_access_key=aws_secret_access_key,

region_name=region_name)ec2client =
session.client('ec2')client_instance = ec2client.run_instances(
```

```
    ImageId='ami-30041c53',
    KeyName='Keys',
    MinCount=1,
    MaxCount=1,
    InstanceType='t2.micro')
```

La sentencia`run_instance`nos da una gran cantidad de información útil. Por ejemplo:

- **Sintaxis de Solicitud**: cómo podemos personalizar nuestra instancia en la sección.
- **Sintaxis de Respuesta:** qué salida podemos esperar ver.
- **Tipo de retorno:** el formato en el que se proporcionará la salida.(Recuerde que los clientes siempre tendrán un objeto tipo dict.

Ahora verifiquemos el contenido de`client_instance`:

```
In [13]: client_instance
Out[13]:
{'Groups': [],
 'Instances': [{'AmiLaunchIndex': 0,
   'Architecture': 'x86_64',
   'BlockDeviceMappings': [],
   'ClientToken': '',
   'EbsOptimized': False,
   'Hypervisor': 'xen',
   'ImageId': 'ami-30041c53',
   'InstanceId': 'i-0e3a125b86073f341',# omitted
```

Como esperamos, es un diccionario.

Junto con estas características principales, Boto 3 también proporciona sesiones y credenciales y configuración por sesión, así como componentes básicos como autenticación, manejo de parámetros y respuestas, un sistema de eventos para personalizaciones y lógica para reintentar solicitudes fallidas.

Recursos: una interfaz orientada a objetos de alto nivel.

“Los recursos representan una interfaz orientada a objetos para Amazon Web Services (AWS). Proporcionan una abstracción de mayor nivel que los clientes sin procesar de bajo nivel realizadas por los clientes de servicio.”

En otras palabras, los Recursos (donde estén disponibles) pueden hacer lo mismo que los Clientes, pero producen resultados que son mucho más fáciles de consumir.

La documentación continúa para decirnos que *“Estos pueden dividirse conceptualmente en identificadores, atributos, acciones, referencias, recursos secundarios y colecciones”*. Avancemos y veamos cuáles son estos.

Identificadores, atributos, referencias, acciones y colecciones

Donde los clientes son bastante fáciles de entender, hay muchos más recursos. Afortunadamente, la documentación de AWS nos

brinda un excelente punto de partida. Los puntos en cursiva a continuación son las opiniones de AWS, mientras que los puntos sin cursiva son mis comentarios:

- **Identificadores:** *propiedades de un recurso que se establecen tras la instauración del recurso*. Cuando se crea un recurso, se le asigna una ID. Este ID se puede usar en las siguientes llamadas de Atributo y Acción
- **Atributos:** *Proporcionan acceso a las propiedades de un recurso. Los atributos se cargan lentamente la primera vez que se accede a uno a través del método load ()*. Utilizados junto con la ID de un recurso, los atributos proporcionan información sobre el recurso especificado. Por ejemplo, pasar el **ID de una instancia EC2** al atributo image_id nos muestra el AMI que está ejecutando esta instancia:

```
In [61]: ec2resource.Instance('i-0afb49cac524f3191').image_id
Out[61]:
'ami-30041c53'
```

- **Referencias:** *instancias de recursos relacionados que tienen una relación de pertenencia*. Si bien los atributos requieren la identificación de un recurso, las referencias son métodos que pertenecen al objeto de la instancia de Python y, por lo tanto, no necesitan que se proporcione la identificación:

```
In 65]: resource_instance[0].vpc.id Out[65]: 'vpc-9e22dcfb
```

- **Acciones:** *Operaciones de llamada sobre recursos. Pueden manejar automáticamente la transferencia de*

argumentos establecidos a partir de identificadores y algunos atributos.

- Probemos que suceda. Por ejemplo, cerremos una instancia:

```
In [69]: resource_instance[0].stop() Out[69]:
{'ResponseMetadata': {'HTTPHeaders': {'content-type':
'text/xml;charset=UTF-8',    'date': 'Thu, 31 Aug 2017 11:45:08
GMT',    'server': 'AmazonEC2',    'transfer-encoding':
'chunked',    'vary': 'Accept-Encoding'},   'HTTPStatusCode':
200,
# omitted
```

- **Colecciones:** Proporcione una interfaz para iterar y manipular grupos de recursos. Salidas que deben repetirse, por ejemplo, las direcciones IP en una VPC o el número de volúmenes adjuntos a una instancia:

```
In for volume in resource_instance[0].volumes.all():
    print(volume.id)vol-0e6630fdcea489f65
```

Waiters *(estuve pensando como decirle en español y a lo más que llegue fué a Pausadores)***:** Proporcionan una interfaz para esperar a que un recurso alcance un estado específico.

Instancia EC2 utilizando un recurso

Avancemos ahora y usemos una sesión para crear un recurso EC2 para ilustrarlos:

```
import boto3aws_access_key_id = ''
aws_secret_access_key = ''
region_name = 'ap-southeast-2'session =
boto3.session.Session(aws_access_key_id=aws_access_key_id,

aws_secret_access_key=aws_secret_access_key,
```

```
region_name=region_name)ec2resource =
session.resource('ec2')resource_instance =
ec2resource.create_instances(
    ImageId='ami-30041c53',
    KeyName='Keys',
    MinCount=1,
    MaxCount=1,
    InstanceType='t2.micro')
```

Aquellos con un buen ojo para los detalles pueden haber notado que la configuración de Cliente y Recursos es casi idéntica. Las únicas diferencias son los siguientes dos bits:

- `session.resource` en lugar de `session.client`
- `ec2resource.create_instances` en lugar de `ec2client.run_instances`

Sin embargo, ahí es donde terminan las similitudes. Por ejemplo, cuando utilizamos `resource_instance`, vemos una fracción de la información que vimos cuando hicimos lo mismo cono`client_instance`:

```
In [11]: resource_instance
Out[11]: [ec2.Instance(id='i-0afb49cac524f3191')]
```

Wow, que diferencia. El Cliente genera una gran cantidad de detalles sobre la instancia, mientras que el Recurso solo nos muestra el ID de la instancia. Como resultado de esto, necesitaremos extraer la ID para obtener más información sobre la instancia misma.

Para extraer este ID, usamos una técnica similar a la que usamos al extraer la ID de `client_instance`. Aunque en ese caso atravesamos un diccionario, esta vez estamos navegando por una instancia de OOP:

```
In [12]: resource_instance[0].id
Out[12]: 'i-0afb49cac524f3191'
```

Usar Boto3

Para usar Boto 3, debe seguir los siguientes pasos:

1.- Importarlo y decirle qué servicio vas a utilizar:

```
import boto3# Usemos Amazon S3 como recurso
s3 = boto3.resource('s3')# Usemos Amazon AutoScaling como
cliente
asg_client = boto3.client('autoscaling')
```

2.- Hacer una solicitud y procesar la respuesta del servicio:

```
# Imprimir nomre de buckets
for bucket in s3.buckets.all():
    print(bucket.name)# Print out auto scaling groups names
asg_dict = asg_client.describe_auto_scaling_groups()
    for asg in asg_dict['AutoScalingGroups']:
        print (asg['AutoScalingGroupName'])
```