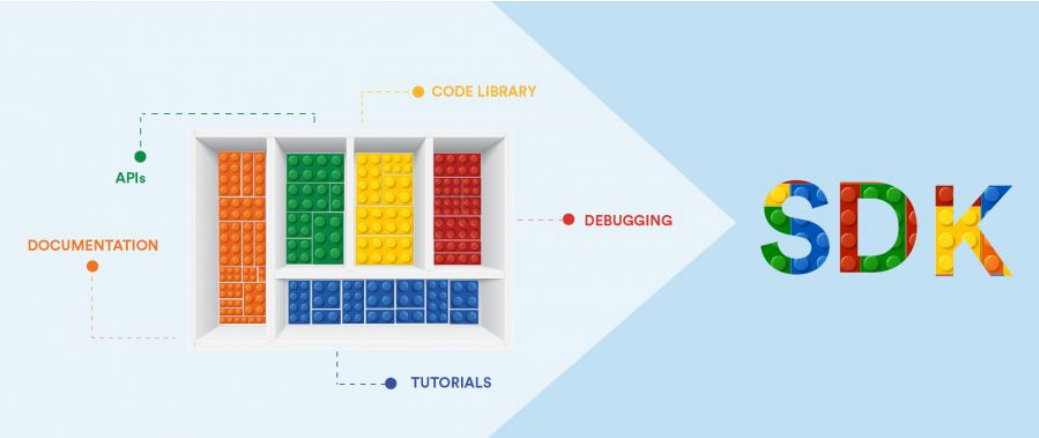
AWS Boto3

CLARUSWAY
WAY TO REINVENT YOURSELF

Boto3

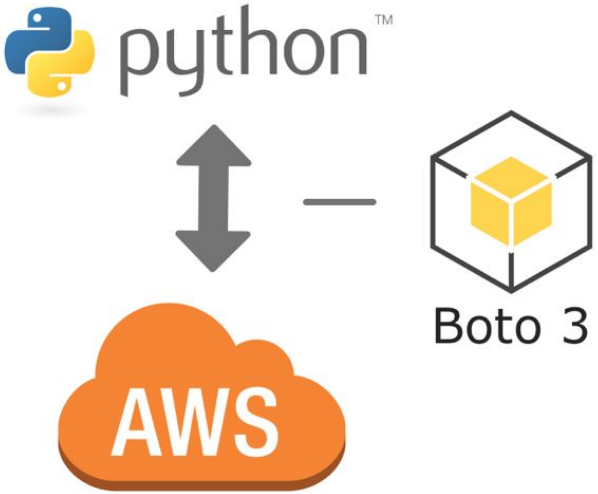
What is SDK?



SDK stands for software development kit or devkit for short. It's a set of software tools and programs used by developers to create applications for specific platforms. SDK tools will include a range of things, including libraries, documentation, code samples, processes, and guides that developers can use and integrate into their own apps. SDKs are designed to be used for specific platforms or programming languages.

Boto3

What is Boto3?



Boto3 is the **AWS SDK** (Software Development Kit) for **Python**. It enables you to create, update, and delete AWS resources with your **Python scripts**. It is basically a **Python library**.

The AWS SDK for Python (Boto3) provides a Python API for AWS infrastructure services. Using the SDK for Python, you can build applications on top of Amazon S3, Amazon EC2, Amazon DynamoDB, and more.

Boto3

- Boto3 runs on top of botocore, which is foundation for AWS CLI. Botocore provides the low-level clients interface, session and credentials & configuration data to a growing number of aws services.

- A session initiates the connectivity to AWS services. A default session uses the default credential profile(e.g. ~/.aws/credentials, or assume your EC2 using IAM instance profile)

Default session

```
import boto3

# Using the default session
sqs = boto3.client('sqs')
s3 = boto3.resource('s3')
```

Custom session

```
import boto3
import boto3.session

# Create your own session
my_session = boto3.session.Session()
```

Boto3

- AWS' Boto3 library is used commonly to integrate Python applications with various AWS services. The two most commonly used features of boto3 are Clients and Resources.

- Client vs Resource
  - Client:
    - this is the original boto3 API abstraction
    - it provides low-level AWS service access
    - all AWS service operations are supported by clients
    - it exposes botocore client to the developer
    - it typically maps 1:1 with the AWS service API

      Example

      s3 = boto3.client('s3')

API Abstraction: Abstraction can take the APIs away from their complex code and implementation specifics, into a common metadata framework, to make them understandable by everyone in the enterprise in a standardized format.

Boto3

Example

```
import boto3


    client = boto3.client('s3')
    response = client.list_objects(Bucket='example')
    for content in response['cont']:
        obj_dict = client.get_object(Bucket='example', Key=cont['Key'])
        print(cont['Key'], obj_dict['LastModified'])
```

Boto3

- Resource:
  - this is the new boto3 API abstraction
  - it provides high-level, object-oriented API
  - it does not provide 100% API coverage of AWS services
  - it uses identifiers and attributes
  - it has actions (operations on resources)
  - it exposes sub-resources and collections of AWS resources

    Example

    ```
    s3 = boto3.resource('s3')
    ```

Boto3

Example

```
import boto3


    s3 = boto3.resource('s3')
    bucket = s3.Bucket('example')
    for obj in bucket.objects.all():
        print(obj.key, obj.last_modified)
```

Boto3

Clients vs Resources

To summarize, resources are higher-level abstractions of AWS services compared to clients. Resources are the recommended pattern to use boto3 as you don't have to worry about a lot of the underlying details when interacting with AWS services. As a result, code written with Resources tends to be simpler.

However, Resources aren't available for all AWS services. In such cases, there is no other choice but to use a Client instead.

Difference Between Boto3 & AWS CLI

Boto3

Boto is a Python library, and in fact the official AWS Python SDK. The AWS CLI, also being written in Python, actually uses part of the boto library (botocore). It would be well suited only if you were writing your deployment scripts in Python. There are official SDKs for other popular languages (Java, PHP, etc.)

AWS CLI

AWS CLI is a command line tool. They're well suited if you want to script your deployment through shell scripting (e.g. bash). For eg; AWS CLI gives you simple file-copying abilities through the "s3" command, which should be enough to deploy a static website to an S3 bucket. It also has some small advantages such as being pre-installed on Amazon Linux, if that was where you were working from (it's also easily installable through pip).

Boto3

Installation and Configuration

Installation

pip install boto3 (pip3 install boto3 for Python3)

Access Configuration

aws configure and supply access keys

Boto3

Using Boto3

```
1  import boto3
2
3  # Use Amazon S3
4  s3 = boto3.resource('s3')
5
6  # Print out all bucket names
7  for bucket in s3.buckets.all():
8      print(bucket.name)
9
```

Boto3

Using Boto3

```
1  import boto3
2
3  # Use Amazon S3
4  s3 = boto3.resource('s3')
5
6  # Upload a new file
7  data = open('test.jpg', 'rb')
8  s3.Bucket('my-bucket').put_object(Key='test.jpg', Body=data)
9
```

Boto3

Boto3 Labs

Lab1:

- Install boto3 on Amazon Linux 2 ec2
- Configure boto3 access
- Test AWS access by listing S3 Buckets, creating buckets and copying objects to S3

Lab2:

- Launch and list ec2 instances
- Resize an instance
- Stop/Start/Terminate an instance

Boto3

Boto3 Labs

Amazon Linux 2 - Install Python and boto3

```
yum install python3
curl -O https://bootstrap.pypa.io/get-pip.py
python3 get-pip.py
pip install boto3
```

Execution: python3 <code>.py

Boto3

List instances

```
#!/usr/bin/env python

import boto3
ec2 = boto3.resource('ec2')
for instance in ec2.instances.all():
 print (instance.id, instance.state)
```

Boto3

Launch instances

```
#!/usr/bin/env python

import boto3
ec2 = boto3.resource('ec2')

# create a new EC2 instance
instances = ec2.create_instances(
    ImageId='ami-09d95fab7fff3776c',  # <== Update with your  instance ID
    MinCount=1,
    MaxCount=2,
    InstanceType='t2.micro',
    KeyName='key-pair'                # <== Update with your keypair
)
```

Boto3

List instances

```
#!/usr/bin/env python

import boto3
ec2 = boto3.client('ec2')

# choose an EC2 instance with id
instance_id = 'i-03eb9211faac95add'    # <== Update with your  instance ID

# Stop the instance
ec2.stop_instances(InstanceIds=[instance_id])
waiter=ec2.get_waiter('instance_stopped')
waiter.wait(InstanceIds=[instance_id])

# Change the instance type
ec2.modify_instance_attribute(InstanceId=instance_id, Attribute='instanceType', Value='t2.micro')

# Start the instance
ec2.start_instances(InstanceIds=[instance_id])
```

Boto3

Instance codes

The valid values for instance-state-code will all be in the range of the low byte and they are:

0 : pending
16 : running
32 : shutting-down
48 : terminated
64 : stopping
80 : stopped

Boto3

Stop instance

```
#!/usr/bin/env python

import boto3, sys
ec2 = boto3.resource('ec2')

for id in sys.argv[1:]:
    instance =  ec2.Instance(id)
    print(instance.stop())
```

Boto3

Start instance

```
#!/usr/bin/env python

import boto3, sys
ec2 = boto3.resource('ec2')

for id in sys.argv[1:]:
    instance =  ec2.Instance(id)
    print(instance.start())
```

Boto3

Terminate instance

```
#!/usr/bin/env python

import boto3, sys
ec2 = boto3.resource('ec2')

for id in sys.argv[1:]:
    instance =  ec2.Instance(id)
    print(instance.terminate())
```

THANKS!

Any questions?

You can find me at:

- @sumod
- sumod@clarusway.com