Prashant Lakhera
Apr 27 · 4 min read · Listen

100 Days of AWS — Day 23- Introduction to Boto3

To view the complete course, please enroll it using the below link(it's free)

https://www.101daysofdevops.com/courses/100-days-of-aws/

Welcome to Day 23 of 100 Days of AWS. The topic for today is Introduction to Boto3.

What is Boto3?

Boto3 is the Amazon Web Services (AWS) SDK for Python. It enables Python developers to create, configure, and manage AWS services, such as EC2 and S3. Boto3 provides an easy-to-use, object-oriented API, as well as low-level access to AWS services.

Boto3 is built on the top of a library called Botocore, which the AWS CLI shares. Botocore provides the low-level clients, session and credentials, and configuration data. Boto3 built on the top of Botocore by providing its own session, resources, collections, waiters, and paginators.

Botocore is the basis for the aws-cli.

boto/boto3
github.com**boto/botocore**

Installing boto3

```
$ pip3 install boto3 --user
```

- `--user` *By default, pip install packages to a system directory(e.g. /usr/local/lib/python3.9) and this requires root privilege. By using — user flag will makes pip install packages in your home directory instead, which doesn't require any special privileges.*
- *Next, you need to configure the aws command line by proving your aws account security credentials, region name, and default output format. In case if you don't have awscli installed you can install it using the below command.*

```
sudo pip3 install awscli
```

- *By using this provided value, aws cli is going to interact with your AWS account. To configure it, run the aws configure command.*

```
# Configure your aws credentials
$ aws configure
AWS Access Key ID [****************XXXX]:
AWS Secret Access Key [****************XXXX]:
Default region name [us-west-2]:
Default output format [json]:
```

- *To test it*

```
$ aws sts get-caller-identity
{
"Account": "1234567889",
"UserId": "XXXXXXXXX",
"Arn": "arn:aws:iam::1234567889:user/plakhera"
}
```

Some key terms

- ***Session:*** *In the simplest term, it's our AWS management console or in other words initiating the connectivity to AWS services**.** A session manages the state of a particular configuration. A default session is created for us when needed, but we can create our own session and create low-level resources or clients. A typical session looks like this.*

```
# Default Session
ec2 = boto3.resource("ec2")

# Custom Session

>>> session=boto3.session.Session(profile_name="<profile>")
>>> ec2 = session.resource("ec2")
>>> s3 = session.client("s3")
```

Session typically stores the following information.

1. *Your AWS Credentials*
2. *AWS Region*
3. *Other configuration related to your profile*

- *You can create your profile using the aws configure command; else, the default profile is selected.*

```
>>> aws configure --profile <profile name>
```

- ***Resources:*** *Resources represent an object-oriented interface to AWS services. It provides a higher-level abstraction than the raw, low-level calls made by the service client.*

```
>>> ec2_con=session.resource(service_name="service name",region_name="region name")
>>> ec2_con=session.resource(service_name="ec2",region_name="us-west-2")
```

NOTE: *Not all AWS services support resources if you are looking to support all AWS features you need to use client.*

- *To get the list of services supported by the resource*

```
>>> session.get_available_resources()
['cloudformation', 'cloudwatch', 'dynamodb', 'ec2', 'glacier', 'iam',
'opsworks', 's3', 'sns', 'sqs']
```

- ***Clients:*** *Clients provide a low-level interface to AWS whose methods map close to 1:1 with service API. Clients are generated from a JSON service definition file.*

NOTE: *The important point to remember clients support all AWS services operations.*

```
>>> session.get_available_services()
['accessanalyzer', 'acm', 'acm-pca', 'alexaforbusiness', 'amp',
'amplify', 'amplifybackend', 'apigateway', 'apigatewaymanagementapi',
'apigatewayv2', 'appconfig', 'appflow', 'appintegrations',
'application-autos
caling', 'application-insights', 'applicationcostprofiler', 'appmesh',
'apprunner', 'appstream', 'appsync', 'athena', 'auditmanager',
'autoscaling', 'autoscaling-plans', 'backup', 'batch', 'braket',
'budgets', '
ce', 'chime', 'cloud9', 'clouddirectory', 'cloudformation',
'cloudfront', 'cloudhsm', 'cloudhsmv2', 'cloudsearch',
'cloudsearchdomain', 'cloudtrail', 'cloudwatch', 'codeartifact',
'codebuild', 'codecommit', 'cod
edeploy', 'codeguru-reviewer', 'codeguruprofiler', 'codepipeline',
'codestar', 'codestar-connections', 'codestar-notifications',
'cognito-identity', 'cognito-idp', 'cognito-sync', 'comprehend',
'comprehendmedica
l', 'compute-optimizer', 'config', 'connect', 'connect-contact-lens',
'connectparticipant', 'cur', 'customer-profiles', 'databrew',
'dataexchange', 'datapipeline', 'datasync', 'dax', 'detective',
'devicefarm', '
devops-guru', 'directconnect', 'discovery', 'dlm', 'dms', 'docdb',
'ds', 'dynamodb', 'dynamodbstreams', 'ebs', 'ec2', 'ec2-instance-
connect', 'ecr', 'ecr-public', 'ecs', 'efs', 'eks', 'elastic-
inference', 'elast
icache', 'elasticbeanstalk', 'elastictranscoder', 'elb', 'elbv2',
'emr', 'emr-containers', 'es', 'events', 'finspace', 'finspace-data',
'firehose', 'fis', 'fms', 'forecast', 'forecastquery',
'frauddetector', 'fs
x', 'gamelift', 'glacier', 'globalaccelerator', 'glue', 'greengrass',
'greengrassv2', 'groundstation', 'guardduty', 'health', 'healthlake',
'honeycode', 'iam', 'identitystore', 'imagebuilder', 'importexport',
'i
nspector', 'iot', 'iot-data', 'iot-jobs-data', 'iot1click-devices',
'iot1click-projects', 'iotanalytics', 'iotdeviceadvisor', 'iotevents',
'iotevents-data', 'iotfleethub', 'iotsecuretunneling', 'iotsitewise',
'i
otthingsgraph', 'iotwireless', 'ivs', 'kafka', 'kendra', 'kinesis',
'kinesis-video-archived-media', 'kinesis-video-media', 'kinesis-video-
signaling', 'kinesisanalytics', 'kinesisanalyticsv2', 'kinesisvideo',
'km
s', 'lakeformation', 'lambda', 'lex-models', 'lex-runtime', 'lexv2-
models', 'lexv2-runtime', 'license-manager', 'lightsail', 'location',
'logs', 'lookoutequipment', 'lookoutmetrics', 'lookoutvision',
'machinelea
rning', 'macie', 'macie2', 'managedblockchain', 'marketplace-catalog',
'marketplace-entitlement', 'marketplacecommerceanalytics',
'mediaconnect', 'mediaconvert', 'medialive', 'mediapackage',
'mediapackage-vod',
'mediastore', 'mediastore-data', 'mediatailor', 'meteringmarketplace',
'mgh', 'mgn', 'migrationhub-config', 'mobile', 'mq', 'mturk', 'mwaa',
'neptune', 'network-firewall', 'networkmanager', 'nimble', 'opsworks',
 'opsworkscm', 'organizations', 'outposts', 'personalize',
'personalize-events', 'personalize-runtime', 'pi', 'pinpoint',
'pinpoint-email', 'pinpoint-sms-voice', 'polly', 'pricing', 'qldb',
'qldb-session', 'quic
ksight', 'ram', 'rds', 'rds-data', 'redshift', 'redshift-data',
'rekognition', 'resource-groups', 'resourcegroupstaggingapi',
'robomaker', 'route53', 'route53domains', 'route53resolver', 's3',
's3control', 's3ou
tposts', 'sagemaker', 'sagemaker-a2i-runtime', 'sagemaker-edge',
'sagemaker-featurestore-runtime', 'sagemaker-runtime', 'savingsplans',
'schemas', 'sdb', 'secretsmanager', 'securityhub', 'serverlessrepo',
'servi
ce-quotas', 'servicecatalog', 'servicecatalog-appregistry',
'servicediscovery', 'ses', 'sesv2', 'shield', 'signer', 'sms', 'sms-
voice', 'snowball', 'sns', 'sqs', 'ssm', 'ssm-contacts', 'ssm-
incidents', 'sso', 's
so-admin', 'sso-oidc', 'stepfunctions', 'storagegateway', 'sts',
'support', 'swf', 'synthetics', 'textract', 'timestream-query',
'timestream-write', 'transcribe', 'transfer', 'translate', 'waf',
'waf-regional',
'wafv2', 'wellarchitected', 'workdocs', 'worklink', 'workmail',
'workmailmessageflow', 'workspaces', 'xray']
```

Some common Tasks

1. ***List all S3 buckets***

```
>>> import boto3

# Create high level resource using boto3
>>> s3 = boto3.resource('s3')

# Print all the bucket name
>>> for bucket in s3.buckets.all():
...     print(bucket.name)
...
plakhera-test-bucket-boto3
```

2: List all IAM users using Client and Resource

```
import boto3

aws_mgt_con=boto3.session.Session()

session=aws_mgt_con.resource(service_name="iam")

session_cli=aws_mgt_con.client(service_name="iam")

# Using Resource
for user in session.users.all():

print(user.name)

# Using Client

print(session_cli.get_user()['User']['UserName'])
```

As you can see in the above example, the Resource version of code is much simpler and compact as compared to the client and it automatically does pagination for you.