Botmetric

AWS Cloud Automation Using Python & Boto3 Scripts – Complete Guide

Post Tweet Share

The dependency on apps and software programs in carrying out tasks in different domains has been on a rise lately. This necessity has caused many businesses to adopt public cloud providers and leverage cloud automation. This shift is fueled by a demand for lesser costs and easier maintenance. AWS has emerged as a leader in the cloud computing domain and companies leveraging algorithmic DevOps (AIOps) for better management and streamlined cloud operations. Using Python and Boto3 scrips to automate AWS cloud operations is gaining momentum. This article will give a cloud engineer's perspective on using Python and Boto3 scripts for AWS cloud optimization.

Challenges in Maintenance

There are lot of challenges that newbies face when migrating their infrastructure to AWS. The functioning of all services and detailed analysis of infrastructure becomes necessary to perform any operation. Regularly performing DevOps tasks further complicates the problem. This might incur a lot of human resource investment, and the associated risk of human errors is inevitable.

AWS answers these problems through automation. The use of language specific libraries that can be easily incorporated and used in simple scripts comes in handy, when a person has to perform an operation that requires a lot of manual effort.In a world that's moving fast on technological

RECENT POSTS

15 Cloud Security Trends and How to Avoid Security Breaches in 2018

Botmetric Announces API Access for AWS & Azure Cloud Management

Top Cloud and AI Trends for 2018

Meltdown and Spectre: Case Analysis and Remediation for AWS Cloud

The Big Fat Blog about AWS Big Data Analytics

READ MORE

CATEGORIES

AWS

Cloud Updates

Cost & Governance

Editor's Pick

Featured Article

Microsoft Azure

Ops & Automation

Security & Compliance

from one region to another, and midway realize that you have copied snapshots of unwanted volumes as well. This can prove to be a costly mistake. To avoid such errors and unnecessary headaches, you can transfer the burden to a simple script that takes care of necessary operations.

Coming out of the comfort zone

Many devops engineers are stuck in the inertia of performing manual operations in all kinds of devops tasks. Consider the case of uploading a file to multiple S3 buckets- A person not familiar with coding practices will prefer to do the task manually. This works when the number of buckets are less. For instance, when a 1000 s3 buckets are to be uploaded with the same file, the person looks for alternatives to perform the task. This is where scripting comes to the rescue.

Few DevOps engineers stay away from writing scripts for cloud automation, thinking it consumes a lot of time. This argument makes sense when writing a script consumes more time than the actual manual operations. This is where the thought process needs to be streamlined and improvised. Think of the script as an infrastructural investment. Initially, it involves effort in setting up and learning to write. But it will be useful in the long game. Every time you put effort into writing a script, you have one less problem to worry about when the same use case arises. These scripts when documented and stored together, will also help other engineers who face similar problems. Hence, though initial effort is necessary, scripting is beneficial in the long run and saves a lot of time.

Why Python and boto3 library?

Python is a kind of programming language that can easily be learnt and used. It is a boon that such an easy language can be used to solve problems of high complexities and cloud automation. One need not know core concepts of the language to solve problems. One can simply get a better taste of the language's flexibility while using boto3 library.

Boto library consists of a set of functions specific to AWS services which can be invoked to perform necessary cloud automation operations. The earlier version of boto was maintained by MIT. But recently AWS has taken over and released boto3. This version is more reliable as it is regularly updated by AWS and availability of descriptive documentation in one place.

Authentication and connection to AWS services

session using authentication credentials. For a person to authenticate themselves to the session, he/she should either be a user, or have valid credentials like access key and secret key. Hence it is very important to protect credentials and make sure that no outsider gets hold of your access key or secret key. If you want to share some credentials to a person intentionally, it is always advisable to create a role and attach a policy which restricts the person to only the required set of operations and then share an access key/value pair with them. Installation of boto3 can be done easily using the command pip install boto3.

There are two types of sessions :

1. **Default Session** : You can configure your credentials using aws configure or by placing the credentials in ~/.aws/credentials file as follows :

```
[default]
 aws_access_key_id = YOUR_ACCESS_KEY
 aws_secret_access_key = YOUR_SECRET_KEY
region=us-east-1
```

Region can be configured as well if necessary. The key value pair can be obtained under the user section of IAM from AWS console. If there is no key value pair, you can generate one and use the same. If no session is specified, boto3 uses the default session to connect with AWS and return a session object.

```
import boto3
s3 = boto3.resource('s3')    # for resource interface
s3_client = boto3.client('s3') # for client interface
```

The above lines of code creates a default session using the credentials stored in the credentials file, and returns the session object which is stored under variables s3 and s3_client.

Resource and client are the interfaces to AWS which can be invoked and used by the functions specific to services. Resource is a higher level object oriented interface, while client refers to lower level interface. But few functionalities that can be performed by client interface cannot be done using resource. You can read more about this on boto3 documentation.

2. **Custom session** : If your use case requires you to change the region for different operations, or use different credentials, custom session is designed for you. There are also many other functionalities like AWS session token, and profile name that are supported in custom sessions.

Botmetric

```
sess
Session(aws_access_key_id=ARN_ACCESS_KEY, aws_secret_access_key=ARN_SECRET_KEY,
region_name=region)  # if required
cloudtrail_sess = sess.client('cloudtrail')
```

Session using STS : This service can be used to obtain temporary credentials for a user. It can be used for both users/roles of AWS account, or by users who do not have direct access to the account, but are granted limited access to another account (federated users). These credentials can be used in the same way as mentioned above to create sessions.

```
sess = Session(aws_access_key_id=YOUR_ARN_ACCESS_KEY,
          aws_secret_access_key=YOUR_ARN_SECRET_KEY)
sts_connection = sess.client('sts')
# Assuming Role
assume_role_object =
sts_connection.assume_role(RoleArn=OTHER_ACCOUNT_ROLE_ARN,
RoleSessionName=ARN_ROLE_SESSION_NAME,
                                   ExternalId=EXTERNAL_ID,
                                   DurationSeconds=3600)
credentials = assume_role_object['Credentials']
tmp_access_key = credentials['AccessKeyId']
tmp_secret_key = credentials['SecretAccessKey']
security_token = credentials['SessionToken']
```

You can also create an assumed role and generate temporary credentials by specifying ARN of your role/user if you have access to the AWS account.

Problem Solving using boto3

Once the session for boto3 is created, other functions can be performed easily with the help of boto3 documentation. This documentation has all the functions that can be performed categorically according to AWS services. The details of parameters required to be passed in the request, along with the response format helps the developer in easier coding. The response is usually in the json format, and the required key can be searched by following the response format in the documentation.

Making the request : The request can either be a read operation which lists the details about certain resources, or a write operation that alters certain parameters or deletes/creates certain resources. Care should be taken to make sure that you do not write requests by mistake. It is a good practice for beginners to specify DryRun=True in all the operations to ensure that the function performs the operation it's supposed to.

Botmetric

```
snapshot_description = ec2_sess.describe_snapshots()
# Delete snapshot specified
ec2_sess.delete_snapshot(SnapshotId ='snap-1234' )
```

In some functions, the returned results have a fixed limit like 1000. In such cases the parameter NextToken or a specified equivalent in the response structure can be used to send the same in the next request in order to get all resources.

Parsing the response : The response returned by the boto3 APIs are in the form of json and can be parsed using simple python dict/list operations. The response contains keys as shown in the documentation. It can be analysed by using any json parser/formatter. The recommended practice is to parse the response using code to obtain the results that you need.

Handling errors : Clean coding is good coding! It is very important to write a code which is error free, especially while performing critical operations, such as taking a snapshot before deleting a volume or backing up data into S3. If errors are not handled in the correct way, it could lead to data loss or server downtime.

Common errors that can occur while making boto3 calls:

- Request parameters used are incorrect or do not exist.
- The values passed as parameters do not match with existing values.
- The user/role does not have sufficient permission to perform the requested action.
- Client initialised for one particular service tries to invoke operations of another service.

AWS API reference provides a detailed list of errors along with codes and keywords . You can log in or print the errors to easily see what went wrong . These errors come under the ClientError exception imported from boto3.

```
try:
    ec2_instances = ec2_sess.describe_instances()
    print(ec2_instances)
except ClientError as e:
    print("Error",e)
```

Sample script

This script is an example of how boto3 can be used to perform various operations. It creates snapshots for all such volumes that has the tag Environment : Prod.

```
from boto3 import Session
from botocore.exceptions import ClientError
# Add your access key and secret key here
sess = Session(aws_access_key_id=ARN_ACCESS_KEY,
               aws_secret_access_key=ARN_SECRET_KEY,
              region_name="us-east-1")
# Client for ec2 for volume and snapshot operations
ec2_client = sess.client("ec2", region_name="us-east-1")
# This is for error handling in case describe volumes operation fails
try:
  # Refer filters parameter from boto3 documentation
  ec2_volumes = ec2_client.describe_volumes(Filters = [
        {
           'Name' : "tag-key",
           'Values' : ["Environment"]
        },
        {
           'Name' : "tag-value",
           'Values' : ['Prod']
        }
    ])
 # Iterate over all volumes matching the given tag
  for volume in ec2_volumes.get('Volumes',[]):
     volume_id = volume.get('VolumeId')
     # Error handling so that other snapshots are created even if one fails
     try:
        # Create snapshot for the specific volume using volume_id
        snapshot = ec2_client.create_snapshot(VolumeId = volume_id,
                                    Description = "Created by script")
        print(snapshot['SnapshotId'])
     except ClientError as e:
        print("Error in snapshot",e)
except ClientError as e:
  print("Error in volume describe",e)
```

What else can boto3 help you with?

Cloud automation has witnessed a growing trend in recent years. Any task that has to be performed on a daily basis can now be automated using code or various tools. It is time for all engineers to follow along the same path, and automate such tasks that require manual effort on a regular basis. Boto3 and python can help you pave the way. You can write a script that performs the desired task and schedule a cron job for the same by using simple steps. This job can be added into a machine or can even be added on your local system as per need. The advantage of scheduling scripts at a particular time are the updates, accuracy and reduction of manual errors.

For example, to set up a cron job for the file ExecuteScript.py on your local linux machine, enter the command :

crontab -e

Botmetric

*20 12 * * * python ExecuteScript.py*

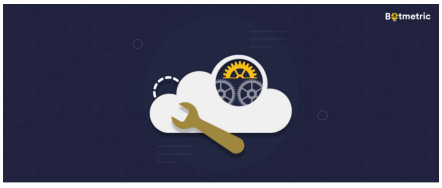
The above command executes the required boto3 script every day at 12.30 p.m. You can learn more about writing cron expressions by referring to various sources available online. There are many other schedulers to execute scripts like apscheduler used in python which can be used along with job store to schedule jobs at required times.

Boto3 and python has many additional features that solve numerous other use cases. Only a few basic concepts have been covered in this article. You can learn more only through exploring the library and working on it. Once you master the basic concepts of boto3, the rest becomes a cake walk. Get started with boto3 and say no to manual operations. It can save you a lot of time in the near future!

SWATHI HARISH • DECEMBER 14, 2017

AWS Automation aws boto3 Boto3 cloud automation Python Script

MORE >



Amazon Announces Unified AWS Systems Manager

Botmetric

interface, AWS Systems Manager to manage components of both cloud.... [read more]