|  |
| --- |
| VASCO Data Security |
| Amazon Web Services (AWS) |
| Briefing |

|  |
| --- |
| Kadir Özdemir  2-16-2017 |

Contents

[1.1. AWS 2](#_Toc475521773)

[1.1.1. Introduction 2](#_Toc475521774)

[1.1.2. Structure of Virtual Machines 2](#_Toc475521775)

[1.1.3. API (Boto3) 4](#_Toc475521776)

[1.2. Briefing of important changes in SDTF 6](#_Toc475521777)

[1.3. To do: 7](#_Toc475521778)

# AWS

### Introduction

Amazon Web Services (AWS) is a secure cloud services platform, offering compute power, database storage, content delivery and other functionality to help businesses scale and grow. From data warehousing to deployment tools, directories to content delivery, over 50 services are available in just a few mouse clicks with AWS. We will use EC2, S3 bucket services, EC2 for deploy virtual machines, and S3 buckets for store our own virtual machine images. The AWS Cloud is available in many countries. The near one is Frankfurt.

### Structure of Virtual Machines

#### vCloud structure



Every vApp is a Virtual Network and regrouped several Virtual Machines. The catalog contains vApp templates, based on those templates we can deploy (copy) in own cloud a vApp (Environments in SDTF).

#### AWS structure that I adopted



On AWS, for deploy a virtual machine we need to specify the AMI (Amazon Machine Images).

An AMI includes the following:

* A template for the root volume for the instance
* Launch permissions that control which AWS accounts can use the AMI to launch instances
* A block device mapping that specifies the volumes to attach to the instance when it’s launched

We can launch instances from as many different AMIs as you need.

To keep the same structure of vCloud, I associated different “Tags” as Environment Name, Name (“Identikey, Tivoli, eDir, DC”) to instances (VMs). If we go on the tab “Instances” on AWS, we can find a list of all instances and it is not grouped as vCloud based on the environment name. Therefore, the environment name is duplicated as many virtual machine it contains on that instance list.

### API (Boto3)

Boto3 is the AWS SDK for Python. Boto3 makes it easy to integrate your Python application, library, or script with AWS services including Amazon S3, Amazon EC2 and more.

You can find the API here: <http://boto3.readthedocs.io/en/latest/index.html>

#### Installation & use

Install the latest Boto3 release via pip:

* pip install boto3

You may also install a specific version:

* pip install boto3==1.0.0

Now you can use boto3 by importing it:

* import boto3

#### Implementation

You can find the implementation at:

* framework/core/virtualization\_aws.py

The different steps to create an instance:

1. Connect to AWS:
   1. Create a Session by using the credentials of the account.
   2. Specify the services that you want to use (EC2 in our case).
2. Find the AMI image that we need.
3. Create the instance:
   1. Specify AMI’s image ID.
   2. Specify the security group.
   3. Specify the resources (t2.micro, t2.small, …)
4. Add tags to the instance (Environment name, name, …).

The implementation provides:

* Connection to AWS EC2 service.
* Power on an instance.
* Power off an instance.
* Power off and terminate an instance.
* Create an instance.
* Assign automatically the tags that we need.
* Get the configuration information (external IP, internal IP, …).

#### GitLab

We also created a branch “aws\_cloud\_test” on GitLab. The implementation is documented. However, if you want more information about the instance or image on the API:

* <http://boto3.readthedocs.io/en/latest/reference/services/ec2.html#instance>
* <http://boto3.readthedocs.io/en/latest/reference/services/ec2.html#image>

# Briefing of important changes in SDTF

* We deployed a FTP server for store the trace files, because the AWS instances do not have access to our internal network. For the FTP server, we used an instance on AWS. Therefore, we hardcoded the IP of the FTP server that we deployed to use it:
  + framework/core/config.py :
    - host="ftp://jenkins-qa.vasco.com" to host="ftp://35.156.194.179"
  + framework/core/staf.py :
    - HOSTNAME = "jenkins-qa.vasco.com" to HOSTNAME = "35.156.194.179"
* For the FTP server, we also added on
  + framework/core/systemtest.py:
    - [ftp.set\_pasv(False)](ftp://ftp.set_pasv(False)) in \_\_create\_ftp\_folder\_structure methods.

Without that, it used the FTP server on “active” mode and we lose a part of the trace file.

* We also added the internal IP to the refuse list, because the script tried to use my internal IP and not the external IP of Vasco for different tests (SOAP, RADIUS):
  + framework/utils/myip.py:
    - refuse=[('192.168.0.0', 16), ('127.0.0.0', 8)], …
* The firewall blocked us to access to the AWS instances. Therefore, we asked to IT Support to open different ports that you can find in the table below.

|  |  |
| --- | --- |
| Protocol | Port Range |
| TCP | 80 |
| TCP | 5999 |
| TCP | 65000->65002 |
| TCP | 8443 |
| TCP | 3389 |
| TCP | 51690 |
| TCP | 6500->6550 |
| TCP | 8888 |
| UDP | 1194 |
| TCP | 20->22 |
| TCP | 20003->20007 |
| UDP | 1812-1813 |

# To do:

* A workaround is also needed if we :
  + Use an AWS instance as a TCH, because we need to report the result to Testlink. Testlink is in our internal network.
  + If we will integrate AWS to Jenkins. Jenkins is also in our internal network.
  + Do the same of vCloud for environments (Virtual Network, Create/Revert Snapshot …).