**Access key ID**

**Secret access key**

AKIAJSKZU3AC2STJMVSA

Aqejxtnf63sXzU89sKFOEdP5Y8s1GAjsIB5WLxI+

**Command line AWS – Udemy**

Lets first understand what it actually is CLI stands for Amazon's Web services command line interface. It is a command line shell program, which provides convenience and productivity features to manage and automate users AWS resources. This means that we use command line interface is an interface for managing our services and it also allows us to automate our tasks with scripts.

So instead of logging into data management console and clicking through many buttons scrolling through many lists and stepping through many Wizard's you can simply add a few commands in your script and execute them to manage your resources much more.

Once you have the script you can use it over and over again.

**Tips:** *Use shift + Q to end a terminal window and go back to the original/previous one*

**Starting CLI**

sudo apt-get install python-pip (installing python)

* Terminal
  + AWS configure
  + IAM – create a access key
  + aws ec2 describe-instances
  + aws ec2 describe-instances --output table
  + aws ec2 describe-instances --output text

Reviewing credential

* + vim ~/.aws/credentials
  + vim ~/.aws/config

-- profile <profile name>

Profiles represent logical groups of configuration setting up multiple named.

**Help Doc and how to use it**

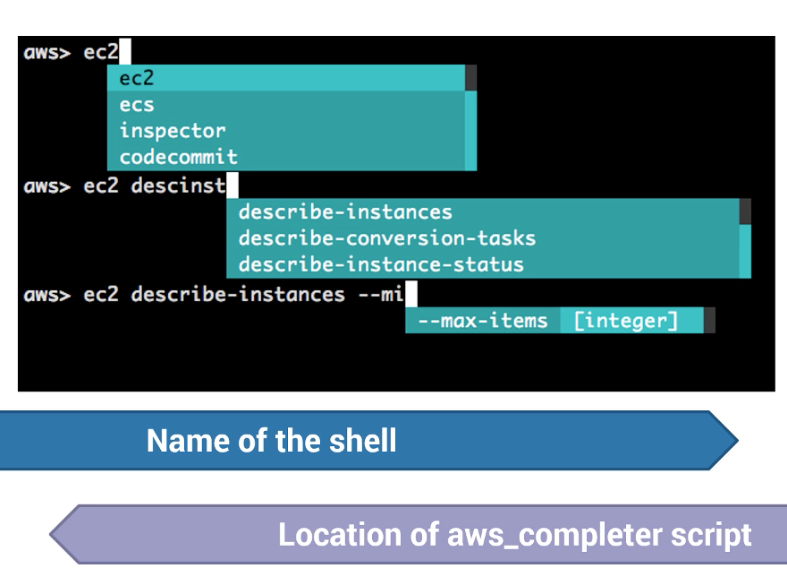
* Access by appending help at the end of the command
  + Aws<space><any type of service><space>help
    - Aws ec2 describe-instances help
* aws iam create-role help
  + aws iam create-role help

**Command Line Interface**

* Auto-Completion
* Filtering CLI Command Output – Query vs. Filter option
* Testing Permission – Dry-Run Option
* Testing Funtions-JMESPath
* Setting up JMESPath Terminal

**Auto-Completion**

Need to elements to work: Name of the shell and Location of aws\_completer script

****

**How to know what kind of sell is installed in my cp**

* $ echo $SHELL (example - /bin/bash)
* AWS – auto completion command – $ which aws\_completer
  + /usr/local/bin/aws\_completer
* Unable – auto completion tool
  + $ complete – C ‘ /usr/local/bin/aws\_completer’ aws

**Filter output on server side (filter)**

**Filter output on client side (query)**

**Tips:** Filtering on the server side is much more efficient than filtering on the client side

but if faster response time is not a consideration then we must always use the query option.

“That is correct. In fact, client side filtering can be done via query option and server side filtering can be done using filter option.”

Filtering: aws ec2 describe-instances --filter Name=instance-type, Value=t2.micro

**Filter output on client side (query)**

aws ec2 describe-regions --query 'Regions[?RegionName==`us-west-1`]'

**Testing permissions – Dry – Run Option – Be careful using this**

aws ec2 describe-regions --dry-run

Result: An error occurred (DryRunOperation) when calling the DescribeRegions operation: Request would have succeeded, but DryRun flag is set.

**Testing Functionality – JMESPath**

This helps you with documentation about the different services within AWS.

Example:

* aws s3 mb help

**Setting – JMESPath Term**

sudo pip install jmespath-terminal

aws ec2 describe-regions | jpterm (feeding from AWS into jpterm)

**CLI-EC2**

It is easy to stand for Elastic Compute Cloud.

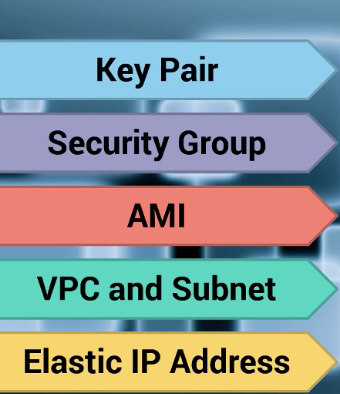
It is easy to provide scalable computing capacity in the aid of the cloud using easy to eliminate your need to invest in hardware upfront so you can develop and deploy applications faster.

You can use Amazon easy to launch as many or as few virtual servers as you need.

You can configure security networking and manage storage for these virtual servers.

It is easy to enable you to scale up or down to handle the changes in requirements or spikes in popularity.

**EC2 level high level**

****

**CLI – KeyPair**

aws ec2 help

create-key-pair

Example: $ aws ec2 create-key-pair --key-name monse

Getting access to the private key: $ aws ec2 create-key-pair --key-name monse--output text > Avalkey.pem

$ ls

Avalkey.pem

monse.pem

**Security Groups**

aws ec2 help

create-security-group

$ aws ec2 create-security-group help

$ aws ec2 create-security-group --group-name aval --description "aval tutorial" --vpc-id vpc-8bb95df0

sg-049ff6ba3371bfa3a

**Verified existence of the SG**

$ aws ec2 help

$ aws ec2 help describe-security-groups help

$ aws ec2 describe-security-groups --group-ids sg-049ff6ba3371bfa3a

output

{

"SecurityGroups": [

{

"Description": "Aval tutorial",

"GroupName": "Aval",

"IpPermissions": [],

"OwnerId": "403044430326",

"GroupId": "sg-049ff6ba3371bfa3a",

"IpPermissionsEgress": [

{

**AMI**

aws ec2 run-instances help

**To launch an instance in EC2-VPC**

aws ec2 run-instances --image-id ami-06b5810be11add0e2 --count 1 --instance-type t2.micro --key-name monse --security-group-ids sg-049ff6ba3371bfa3a

output

{

"Groups": [],

"Instances": [

{

"AmiLaunchIndex": 0,

"ImageId": "ami-06b5810be11add0e2",

"InstanceId": "i-0e3d8b56135f7549e",

"InstanceType": "t2.micro",

"KeyName": "monse",

"LaunchTime": "2018-10-11T10:42:28.000Z",

"Monitoring": {

"State": "disabled"

},

**Elastic IP Address**

aws ec2 help

aws ec2 allocate-address help

Example – command

aws ec2 allocate-address

Output

{

"PublicIp": "54.147.45.209",

"AllocationId": "eipalloc-0e8003040938581d9",

"Domain": "vpc"

}

**How to associate IP’s address**

aws ec2 associate-address help

Command:

aws ec2 associate-address --instance-id i-0e3d8b56135f7549e --allocation-id eipalloc-0e8003040938581d9

Output

{

"AssociationId": "eipassoc-0cd7fc2c0accbe3e4"

}

**Verify existence of this ec2**

**Command**

aws ec2 describe-instances

**User data ec2 – example**

aws ec2 associate-address --instance-id i-0e3d8b56135f7549e --allocation-id eipalloc-0e8003040938581d9 **–user-data “sudo apt-get install nginx”** (this addition in bold will install nginx for us)

**Terminating EC2**

aws ec2 help

aws ec2 terminate-instances help

Command:

aws ec2 terminate-instances --instance-ids i-0e3d8b56135f7549e

Output:

{

"TerminatingInstances": [

{

"InstanceId": "i-1234567890abcdef0",

"CurrentState": {

"Code": 32,

**"Name": "shutting-down"**

},

"PreviousState": {

"Code": 16,

"Name": "running"

}

}

]

}

**CLI – S3**

aws s3 help

aws s3 **mb** help (mb-stand for create)

AMPLES

The following mb command creates a bucket. In this example, the user makes the bucket mybucket. The bucket is created in the region speci-fied in the user's configuration file:

aws s3 mb s3://mybucket ( aws s3 mb s3://myavalbucket1 --region us-east-1)

Output:

make\_bucket: s3://mybucket

**creating a text file**

sudo vim test1

Save and exit: esc+:+x

**Loading/Coping a file into s3**

aws s3 cp help (cp – copy)

The following cp command copies a single file to a specified bucket and

key: **you must be in the folder where the file is save it ( cd: test2file)**

aws s3 cp test2 s3://myavalbucket1/test2

Output:

Upload: test.txt to s3://mybucket/test2.txt

**Storage class command for s3**

* Standar
* Reduced Redundancy
* Standar IA

aws s3 cp help

aws s3 cp test2 s3://myavalbucket1 --storage-class STANDARD\_IA

Output

upload: ./test2 to s3://myavalbucket1/test2

**Objects in s3-CLI**

aws s3 ls s3://myavalbucket1

Output

2018-10-11 10:16:23 5 test2

**Moving in s3-CLI**

aws s3 mv s3://myavalbucket1 . --recursive **(mv – move) ( recursive means that all file in this butcket will be move to our local disk)**

**Output**

move: s3://myavalbucket1/test2 to ./test2

**MV – Exclude and Include filter base in the name of the file.**

aws s3 mv . s3://myavalbucket1 . --recursive --exclude ‘\*’ --include ‘\*.txt’

**Sync Command**

First create a file

$ touch test2.txt

$ ls

$ aws s3 sync . s3://myavalbucket1 ( . stand local disk)

Output

upload: ./test2.txt to s3://myavalbucket1/test2.txt

upload: ./test2 to s3://myavalbucket1/test2

Deleting – Sync

$ rm test2 ( directory )

$ aws s3 sync . s3://myavalbucket1 --delete

$ touch test3.txt

$ aws s3 sync . s3://myavalbucket1 --acl public-read **(setting permission for public read)**

Output

upload: ./test3.txt to s3://myavalbucket1/test3.txt

**Delete s3 bucket – CLI**

$ aws s3 rb s3://myavalbucket1

**Force to delete the s3:** $ aws s3 rb s3://myavalbucket1 –force

**Output**

delete: s3://myavalbucket1/test2.txt

delete: s3://myavalbucket1/test3.txt

remove\_bucket: myavalbucket1

**Note:**  before deleting an s3 bucket must be empty.

Command to remove objects from s3 bucket.

$ aws s3 rm s3://myavalbucket1/test2

**CLI-IAM**

This is a very important service within the AWS cloud ecosystem, which helps us deploy and manage our AWS infrastructure securely. I am enables us to securely control access to various services and resources for our users using I am. We can create and manage users and groups and assign them permissions to allow or deny access to police

Fine grain control

Flexible

In-built security policy

User, groups, permission

Easy to configure management

$ aws im help

$ aws iam list-users

**How to create a user**

$ aws iam create-user --user-name Monse

output

{

"User": {

"Path": "/",

"UserName": "Monse",

"UserId": "AIDAJ6UHAUICPDUSKU5XU",

"Arn": "arn:aws:iam::403044430326:user/Monse",

"CreateDate": "2018-10-12T10:13:51Z"

}

}

**To delete an IAM user**

$ aws iam delete-user --user-name Bob

**Create a group**

$ aws iam create-group --group-name avalg

Output

{

"Group": {

"Path": "/",

"GroupName": "avalg",

"GroupId": "AGPAIWNHZBTWC2TW45NVM",

"Arn": "arn:aws:iam::403044430326:group/avalg",

"CreateDate": "2018-10-12T10:23:09Z"

}

}

**Listing Groups Command**

$ aws iam list-groups

**Adding a user to the group command**

$ aws iam add-user-to-group --user-name Monse --group-name avalg

Output

Not output reflected – check iam console

**Getting information for a single group**

**$** aws iam get-group --group-name avalg

"Users": [

{

"Path": "/",

"UserName": "Monse",

"UserId": "AIDAJ6UHAUICPDUSKU5XU",

"Arn": "arn:aws:iam::403044430326:user/Monse",

"CreateDate": "2018-10-12T10:13:51Z"

}

],

"Group": {

"Path": "/",

"GroupName": "avalg",

"GroupId": "AGPAIWNHZBTWC2TW45NVM",

"Arn": "arn:aws:iam::403044430326:group/avalg",

"CreateDate": "2018-10-12T10:23:09Z"

}

}

**User name and password – CLI**

$ aws iam create-login-profile --user-name Monse --password Monserr@t123

Output

{

"LoginProfile": {

"UserName": "Monse",

"CreateDate": "2018-10-12T10:42:39Z",

"PasswordResetRequired": false

}

}

**IAM – API Access Key**

Let's clarify these terms stand for application programming interface. API allow other programs or scripts to log into the cloud programmatically and interact with our database infrastructure.

Command

$ aws iam create-access-key --user-name Monse

**Output**

{

"AccessKey": {

"UserName": "Monse",

"AccessKeyId": "AKIAI2Y3I7LIM3PMGRGQ",

"Status": "Active",

"SecretAccessKey": "HMQ5YdvtUAJNB0F5RwJLk4Yhbi4eFC/9wHHfaPrp",

"CreateDate": "2018-10-12T10:49:58Z"

}

}

**Deleting Access Key**

$ aws iam delete-access-key --user-name Monse --access-key-id AKIAI2Y3I7LIM3PMGRGQ

Output

Not output reflected – check iam console

**Creating a new key after deletion - Command**

$ aws iam create-access-key --user-name Monse >> key.txt

Output

Not output reflected

**How to verify the key**

$ ls

$ sudo vim key.txt

Output

{

"AccessKey": {

"UserName": "Monse",

"AccessKeyId": "AKIAIBRRYQMH65LLROKA",

"Status": "Active",

"SecretAccessKey": "gHZjQO5KE8fwCa/4mQK0ht0vbXehW5pF4xsui1jY",

"CreateDate": "2018-10-12T11:00:36Z"

}

**IAM User/Groups – Assign and Remove Policy**

**Assign Policy**

$ aws iam attach-group-policy help

The following attach-group-policy command attaches the AWS managed policy named ReadOnlyAccess to the IAM group named Finance:

$ aws iam attach-group-policy --policy- arn:aws:iam::aws:policy/AdministratorAccess --group-name avalg

Output

Not output reflected – check iam console

**Remove Policy**

$ aws iam remove-user-from-group --group-name avalg --user-name-Monse

Output

Not output reflected – check iam console

**Deleting Groups**

You must first detach all policies of the group

$ aws iam detach-group-policy help

$ aws iam detach-group-policy --policy- arn:aws:iam::aws:policy/AdministratorAccess --group-name avalg

Output

Not output reflected – check iam console

**Delete**

$ aws iam delete-group --group-name avalg

Output

Not output reflected – check iam console

**ELB-CLI**

There are two types of ELB. They are – • Classic Load Balancer: Routes traffic based on application or network level information and is ideal for simple load balancing of traffic across multiple EC2 instances where high availability, automatic scaling, and robust security are required. • Application Load Balancer: Service that operates at the application layer and allows to define routing rules based on content across multiple services or containers running on one or more Amazon Elastic Compute Cloud (Amazon EC2) instances.

**$** aws elb help – classic

$ aws elbv2 help – Application load balancer (advances)

**Creating a Application load balancer**

$ aws elbv2 create-load-balancer --name avaelb-laodbal --subnets subnet-6e04280a subnet-ef546fa4 --security-group sg-049ff6ba3371bfa3a

**Output**

{

"LoadBalancers": [

{

"LoadBalancerArn": "arn:aws:elasticloadbalancing:us-east-1:403044430326:loadbalancer/app/avaelb-laodbal/fdb727180c9832c1",

"DNSName": "avaelb-laodbal-1865793889.us-east-1.elb.amazonaws.com",

"CanonicalHostedZoneId": "Z35SXDOTRQ7X7K",

"CreatedTime": "2018-10-12T14:33:52.920Z",

"LoadBalancerName": "avaelb-laodbal",

"Scheme": "internet-facing",

"VpcId": "vpc-8bb95df0",

"State": {

"Code": "provisioning"

},

"Type": "application",

"AvailabilityZones": [

{

"ZoneName": "us-east-1c",

"SubnetId": "subnet-6e04280a"

},

{

"ZoneName": "us-east-1a",

"SubnetId": "subnet-ef546fa4"

}

],

"SecurityGroups": [

"sg-049ff6ba3371bfa3a"

],

"IpAddressType": "ipv4"

}

]

}

**ping ELB-ARN**

$ echo "arn:aws:elasticloadbalancing:us-east-1:403044430326:loadbalancer/app/avaelb-laodbal/fdb727180c9832c1" >>lb.txt

**Target Group – ELB**

$ aws elbv2 create-target-group --name avaelb-tg --protocol HTTP --port 80 --vpc-id vpc-8bb95df0

**Output**

{

"TargetGroups": [

{

"TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:403044430326:targetgroup/avaelb-tg/c03aa016e637c78d",

"TargetGroupName": "avaelb-tg",

"Protocol": "HTTP",

"Port": 80,

"VpcId": "vpc-8bb95df0",

"HealthCheckProtocol": "HTTP",

"HealthCheckPort": "traffic-port",

"HealthCheckIntervalSeconds": 30,

"HealthCheckTimeoutSeconds": 5,

"HealthyThresholdCount": 5,

"UnhealthyThresholdCount": 2,

"HealthCheckPath": "/",

"Matcher": {

"HttpCode": "200"

},

"TargetType": "instance"

}

]

}

**Saving Target group ARN**

$ sudo vim lb.txt

VIM opens paste and save ARN

**Creating and registering target**

$ aws elbv2 register-targets --target-group-arn arn:aws:elasticloadbalancing:us-east-1:403044430326:targetgroup/avaelb-tg/c03aa016e637c78d --targets Id=i-0a011d14fe0d698ac

**Listener**

So what is a listener a listener is a process that checks for connection requests. The listener is configured to listen for incoming connections on a certain protocol and on a certain port.

This protocol and port combination can be defined for the client to load balancer connections.

Also known as front end connections and this can also be defined for the load balancer to back instance connections all the back and connection.

The rules that are defined for a listener determine how the load balancer routes the requests to the targets in the target groups for the classic ELB listener support HTTP, HTTPS, DCP, SSL.

Whereas for application ELB listeners AWS supports HTTP and HTTPS only.

**Command**

$ aws elbv2 create-listener --load-balancer-arn arn:aws:elasticloadbalancing:us-east-1:403044430326:loadbalancer/app/avaelb-laodbal/fdb727180c9832c1 --protocol HTTP --port 80 --default-actions Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:us-east-1:403044430326:targetgroup/avaelb-tg/c03aa016e637c78d

Output

{

"Listeners": [

{

"ListenerArn": "arn:aws:elasticloadbalancing:us-east-1:403044430326:listener/app/avaelb-laodbal/fdb727180c9832c1/96aa59e76deed48b",

"LoadBalancerArn": "arn:aws:elasticloadbalancing:us-east-1:403044430326:loadbalancer/app/avaelb-laodbal/fdb727180c9832c1",

"Port": 80,

"Protocol": "HTTP",

"DefaultActions": [

{

"Type": "forward",

"TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:403044430326:targetgroup/avaelb-tg/c03aa016e637c78d"

}

]

}

]

}

**Verifying the target help**

**Command**

$ aws elbv2 describe-target-health --target-group-arn arn:aws:elasticloadbalancing:us-east-1:403044430326:targetgroup/avaelb-tg/c03aa016e637c78d

**Output**

{

"TargetHealthDescriptions": [

{

"Target": {

"Id": "i-0a011d14fe0d698ac",

"Port": 80

},

"HealthCheckPort": "80",

"TargetHealth": {

"State": "unused",

"Reason": "Target.NotInUse",

"Description": "Target is in an Availability Zone that is not enabled for the load balancer"

}

}

]

}

**Deleting ELB**

**Command**

$ aws elbv2 delete-load-balancer --load-balancer-arn arn:aws:elasticloadbalancing:us-east-1:403044430326:loadbalancer/app/avaelb-laodbal/fdb727180c9832c1

**Output**

Not output reflected – check ELB console

**Deleting Target Groups**

**Command**

$ aws elbv2 delete-target-group --target-group-arn arn:aws:elasticloadbalancing:us-east-1:403044430326:targetgroup/avaelb-tg/c03aa016e637c78d

**Output**

Not output reflected – check Target console

**Classic –ELB –CLI**

$ aws elb help

$ aws elb create-load-balancer --load-balancer-name avacl --listeners “Protocol=HTTP,LoadBalancerPort=80, InstanceProtocol =HTTP,InstancePort=80” --subnets subnet-6e04280a --security-group sg-049ff6ba3371bfa3a

Output

{

"DNSName": "avacl-128961602.us-east-1.elb.amazonaws.com"

}

**Note**

“*Protocol=HTTP,LoadBalancerPort=80* is the front end services between the client and the AWS server

*InstanceProtocol=HTTP,InstancePort=80”* back end connection/configuration between the ELB and HTTP via port 80

$ aws elb describe-load-balancers **(ls all ELB)**

$ aws elb describe-load-balancers --load-balancer-name avacl **(ls single ELB)**

Output

{

"LoadBalancerDescriptions": [

{

"LoadBalancerName": "avacl",

"DNSName": "avacl-128961602.us-east-1.elb.amazonaws.com",

"CanonicalHostedZoneName": "avacl-128961602.us-east-1.elb.amazonaws.com",

"CanonicalHostedZoneNameID": "Z35SXDOTRQ7X7K",

"ListenerDescriptions": [

{

"Listener": {

"Protocol": "HTTP",

"LoadBalancerPort": 80,

"InstanceProtocol": "HTTP",

"InstancePort": 80

},

"PolicyNames": []

}

],

"Policies": {

"AppCookieStickinessPolicies": [],

"LBCookieStickinessPolicies": [],

"OtherPolicies": []

},

"BackendServerDescriptions": [],

"AvailabilityZones": [

"us-east-1c"

],

"Subnets": [

"subnet-6e04280a"

],

"VPCId": "vpc-8bb95df0",

"Instances": [],

"HealthCheck": {

"Target": "TCP:80",

"Interval": 30,

"Timeout": 5,

"UnhealthyThreshold": 2,

"HealthyThreshold": 10

},

"SourceSecurityGroup": {

"OwnerAlias": "403044430326",

"GroupName": "Aval"

},

"SecurityGroups": [

"sg-049ff6ba3371bfa3a"

],

"CreatedTime": "2018-10-12T17:45:26.280Z",

"Scheme": "internet-facing"

}

]

}

**Creating a new Listener**

**Command**

$ aws elb create-load-balancer-listeners --load-balancer-name avacl --listeners "Protocol=HTTP,LoadBalancerPort=80,InstanceProtocol =HTTP,InstancePort=80"

**Creating an Internal Load Balancer**

$ aws elb create-load-balancer --load-balancer-name avaclinternal --listeners "Protocol=HTTP,LoadBalancerPort=80,InstanceProtocol =HTTP,InstancePort=80" --scheme internal --subnets subnet-6e04280a --security-group sg-049ff6ba3371bfa3a

**Output**

{

"DNSName": "internal-avaclinternal-666153698.us-east-1.elb.amazonaws.com"

}

**Others CLI – ELB**

How to register an instance into an internal or external load balancer

$ aws elb register-instances-with-load-balancer --load-balancer-name avaclinternal --instances i-0a011d14fe0d698ac

**Output**

{

"Instances": [

{

"InstanceId": "i-0a011d14fe0d698ac"

}

]

}

How to deregister an instance into an internal or external load balancer

$ aws elb deregister-instances-from-load-balancer --load-balancer-name avaclinternal --instances i-0a011d14fe0d698ac

Output

{

"Instances": []

}

How to delete internal load balancer

$ aws elb delete-load-balancer --load-balancer-name avaclinternal

**Output**

Not output reflected – check Target console

How to delete external load balancer

$ aws elb delete-load-balancer --load-balancer-name avacl

**Output**

Not output reflected – check Target console

**AWS CLI Commands For AWS SNS & AWS SQS**

aws sns components – Push model

* Topic
* Owners
* Subscribers
* Publishers

ESCRIPTION

Amazon Simple Notification Service (Amazon SNS) is a web service that

enables you to build distributed web-enabled applications. Applications

can use Amazon SNS to easily push real-time notification messages to

interested subscribers over multiple delivery protocols.

Amazon Simple Notification Service (SNS) is a simple, fully-managed "push" messaging service that allows users to push texts, alerts or notifications, like an auto-reply message, or a notification that a package has shipped. Applications can use AWS SNS to easily push real time notifications to the interested supply route or multiple delivery protocols. These delivery protocols can be email, SMS, SQS-Queues etc.

SNS – CLI

**Command**

**Creating a topic**

$ aws sns create-topic --name ava-topic

**Output**

{

"TopicArn": "arn:aws:sns:us-east-1:403044430326:ava-topic"

}

**Subscribing a topic**

$ aws sns subscribe --topic-arn arn:aws:sns:us-east-1:403044430326:ava-topic --protocol email --notification-endpoint valdiviaaugusto9@gmail.com

**Output**

{

"SubscriptionArn": "pending confirmation"

}

**List subscriptions**

$ aws sns list-subscriptions

**Output**

{

"Subscriptions": [

{

"SubscriptionArn": "arn:aws:sns:us-east-1:403044430326:NotifyMe1:2045c4b4-84df-4b5d-8f11-5a4c5e72b86f",

"Owner": "403044430326",

"Protocol": "email",

"Endpoint": "valdiviaaugusto9@gmail.com",

"TopicArn": "arn:aws:sns:us-east-1:403044430326:NotifyMe1"

},

{

"SubscriptionArn": "arn:aws:sns:us-east-1:403044430326:ava-topic:966d5d33-8efb-4f83-8d00-f7f98aa62d21",

"Owner": "403044430326",

"Protocol": "email",

"Endpoint": "valdiviaaugusto9@gmail.com",

"TopicArn": "arn:aws:sns:us-east-1:403044430326:ava-topic"

}

]

}

**Publishing topic**

$ aws sns publish --topic-arn arn:aws:sns:us-east-1:403044430326:ava-topic --message "This is a great learning"

**Output**

{

"MessageId": "83aac78b-55f3-5e44-9bb8-93b0eaa1f561"

}

**Unsubscribing a subscription**

$ aws sns unsubscribe --subscription-arn arn:aws:sns:us-east-1:403044430326:ava-topic:966d5d33-8efb-4f83-8d00-f7f98aa62d21

**Output**

Check sns console

**Deleting a topic**

$ aws sns delete-topic --topic-arn arn:aws:sns:us-east-1:403044430326:ava-topic

**Output**

Check sns console

$ aws sns list-topics

{

"Topics": [

{

"TopicArn": "arn:aws:sns:us-east-1:403044430326:NotifyMe"

},

{

"TopicArn": "arn:aws:sns:us-east-1:403044430326:NotifyMe1"

}

]

}

**SQS**

Amazon Simple Queue Service (Amazon SQS) is a reliable, highly-scalable hosted queue for storing messages as they travel between applications or micro services. Amazon SQS moves data between distributed application components and helps you decouple these components.

http://docs.aws.amazon.com/cli/latest/reference/sqs/index.html`\_\_

**Creating a queue**

$ aws sqs create-queue --queue-name avalqueue --attribute <file://createqueue.json>

**Output**

{

"QueueUrl": "https://queue.amazonaws.com/403044430326/avalqueue"

}

Note – you must create the json file using a text editor. I used vim for this file “createqueue.json”

**List queue**

$ aws sqs list-queues

**Output**

{

"QueueUrls": [

"https://queue.amazonaws.com/403044430326/avalqueue"

]

}

**Sending message to SQS queues**

$ aws sqs send-message --queue-url https://queue.amazonaws.com/403044430326/avalqueue --message-body "This is example 1"

**Output**

{

"MD5OfMessageBody": "889694ef72d1124ebb248bd7ba3867e4",

"MessageId": "60912c37-52ed-4b91-bd60-3a52a7547aba"

}

**Receiving message in SQS queue**

$aws sqs receive-message --queue-url <https://queue.amazonaws.com/403044430326/avalqueue>

**Output**

{

"Messages": [

{

"MessageId": "60912c37-52ed-4b91-bd60-3a52a7547aba",

"ReceiptHandle": "AQEBrUBf6PFM/5C0ClUy6VAGsfze9sA/OyaLY6EL621qdthIFEDPnJrFop4z8QFlDUlgSvKjwWUt1ziaXzXXFP+ZMj5KHSEY03ZxhbhzD41fhGoodEAvDL8VWcsl3QbCUXi8a0QJ79jn8po0WoMzF9NLpfTDnTI+G1s/7IhgVlug8p4mVdWdsyx1811fQ+kFOf4lwuqXEO3LAbGp/46f26+QHuy3t+sIq10YZfMxHGfvFnO8rrp2l4JsPcH3AanRqHHS/NsIlUZgoCJJ6g4CDY7Emmi16MmdH/t7mglYN2hV+Oax6VHwZaxDVJdFwoMY1U9oWyASgeJuFHDvqfEut2O5+nx+GmxTD7yl+euqNsVv0DXw2FgcGX9xD6kRQBMSy1hq",

"MD5OfBody": "889694ef72d1124ebb248bd7ba3867e4",

"Body": "This is example 1"

}

]

}

**Deleting message from SQS queue**

$ aws sqs delete-message --queue-url https://queue.amazonaws.com/403044430326/avalqueue --receipt-handle AQEBrUBf6PFM/5C0ClUy6VAGsfze9sA/OyaLY6EL621qdthIFEDPnJrFop4z8QFlDUlgSvKjwWUt1ziaXzXXFP+ZMj5KHSEY03ZxhbhzD41fhGoodEAvDL8VWcsl3QbCUXi8a0QJ79jn8po0WoMzF9NLpfTDnTI+G1s/7IhgVlug8p4mVdWdsyx1811fQ+kFOf4lwuqXEO3LAbGp/46f26+QHuy3t+sIq10YZfMxHGfvFnO8rrp2l4JsPcH3AanRqHHS/NsIlUZgoCJJ6g4CDY7Emmi16MmdH/t7mglYN2hV+Oax6VHwZaxDVJdFwoMY1U9oWyASgeJuFHDvqfEut2O5+nx+GmxTD7yl+euqNsVv0DXw2FgcGX9xD6kRQBMSy1hq

**Output**

Check SQS – Console

**Delaying message from SQS queue**

$ aws sqs send-message --queue-url https://queue.amazonaws.com/403044430326/avalqueue --message-body "This is example 2" --delay-seconds 10

**Output**

{

"MD5OfMessageBody": "02471bd4749d271a3ed16e6a0fb0dd40",

"MessageId": "ff1a4303-e476-4460-b2f7-fa6a99526fa4"

}

**Deleting SQS queue**

$ aws sqs delete-queue --queue-url <https://sqs.us-east> 1.amazonaws.com/403044430326/avalqueue

**Output**

Check SQS – Console

**Or – CLI**

$ aws sqs list-queues

**Output**

0

### AWS CLI Commands For AWS ECS

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage Docker containers on a cluster. You can host your cluster on a serverless infrastructure that is managed by Amazon ECS by launching your services or tasks using the Fargate launch type. For more control, you can host your tasks on a cluster of Amazon Elastic Compute Cloud (Amazon EC2) instances that you manage by using the EC2 launch type. For more information about launch types, see Amazon ECS Launch Types.

Amazon ECS lets you launch and stop container-based applications with simple API calls, allows you to get the state of your cluster from a centralized services, and gives you access to many familiar Amazon EC2 features

You can use Amazon ECS to schedule the placement of containers across your cluster based on your resource needs, isolation policies, and availability requirements. Amazon ECS eliminates the need for you to operate your own cluster management and configuration management systems or worry about scaling your management infrastructure.

**ECS**

* Container Management Service
* Highly Scalable
* Support-Docker

**ECS – Components**

* Container Instances
* Container Cluster
* Amazon ECS Agent

**What is a cluster**?

AWS a cluster is a collection of resources like EC2 and memory primarily EC2 and ECS instances and containers

**Creating ECS Containers-CLI**

$ aws ecs create-cluster --cluster-name avacluster1

**Output**

{

"cluster": {

"clusterArn": "arn:aws:ecs:us-east-1:403044430326:cluster/avacluster1",

"clusterName": "avacluster1",

"status": "ACTIVE",

"registeredContainerInstancesCount": 0,

"runningTasksCount": 0,

"pendingTasksCount": 0,

"activeServicesCount": 0,

"statistics": []

}

}

**Creating a container agent**

An error occurred (InvalidParameterValue) when calling the RunInstances operation: Value (Ava-ECS) for parameter iamInstanceProfile.name is invalid. Invalid IAM Instance Profile name

**Investigate**

**Task Definition – text FILE \***

Must be assigned to a json file to be executed it

Example:

{

"containerDefinitions": [

{

"name": "sleep",

"image": " busybox",

"cpu": 10,

"command": [

"sleep",

"30"

],

"memory": 10,

"essential": true

}

],

"family": "sleepybox"

}

In this task definition we are defining a container named sleep.

The container is going to use the box image.

It has been allocated a certain amount of sleep you and memory resources.

We are also passing an income command here whose purpose is to make this busy box container sleep for

30 seconds and finally we are assigning a family name of sleepy box here.

**How to configure ECS Components – CLI**

$ aws ecs register-task-definition --cli-input-json <file://taskdef.json>

**Output**

{

"taskDefinition": {

"taskDefinitionArn": "arn:aws:ecs:us-east-1:403044430326:task-definition/sleepybox:1",

"containerDefinitions": [

{

"name": "sleep",

"image": "busybox",

"cpu": 10,

"memory": 10,

"portMappings": [],

"essential": true,

"command": [

"sleep",

"30"

],

"environment": [],

"mountPoints": [],

"volumesFrom": []

}

],

"family": "sleepybox",

"revision": 1,

"volumes": [],

"status": "ACTIVE",

"placementConstraints": [],

"compatibilities": [

"EC2"

]

}

}

$ aws ecs list-task-definitions

{

"taskDefinitionArns": [

"arn:aws:ecs:us-east-1:403044430326:task-definition/sleepybox:1"

]

}

**Creating a running a task**

$ aws ecs run-task --cluster default --task-definition sleep360:1

Task definitions define containers that we want to run together. They can be used to model the containers that create the application. Task definitions also allow us to specify CPU and memory as well as docker concepts (links, volumes). Like source codes, track definitions are tracked by name and revision.

**Deleting a cluster**

First the container must be deregistered (inactive)

$ aws ecs deregister-container-intance --container-intance "id here" --cluster "name here"

$ aws ecs delete-cluster --cluster avacluster1

**Output**

{

"cluster": {

"clusterArn": "arn:aws:ecs:us-east-1:403044430326:cluster/avacluster1",

"clusterName": "avacluster1",

"status": "INACTIVE",

"registeredContainerInstancesCount": 0,

"runningTasksCount": 0,

"pendingTasksCount": 0,

"activeServicesCount": 0,

"statistics": []

}

}

### AWS CLI Commands For AWS SES

**Verifying email account**

$ aws ses verify-email-identity --email-address [valdiviaagusto9@gmail.com](mailto:valdiviaagusto9@gmail.com) --region us-east-1

**Output**

Check ses console

$ aws ses list-identities --region us-east-1

{

"Identities": [

"augustovaldivia.ca"

]

}

**Sending emails**

$ aws ses send-email --from "email verified address here" –to "email verified address here" --subject "Test Subject" --test "Hi this is a test" --region us-east-1

### AWS CLI Commands And Shell Scripting

Launch EC2 Instances from AWS Web console vs. AWS-CLI

**AWS Web console** 10 clicks

**AWS-CLI**

Must create a .sh file to automate script.

**Bash**

aws ec2 run-instances --image-id ami-0922553b7b0369273 --count 1 --instance-type t2.micro --region us-east-1 --iam-instance-profile Name="Ava-ECS" --subnet-id subnet-6e04280a

$ sudo chmod 777 ec2create.sh (Maximum permission for the .sh file-for demonstration only)

$ ./ ec2create.sh ( to execute the script)

An error occurred (InvalidParameterValue) when calling the RunInstances operation: Value (Ava-ECS) for parameter iamInstanceProfile.name is invalid. Invalid IAM Instance Profile name

**Search EC2 from CLI**

**.sh –bash –**

$ sudo vim describeec2.sh

#! /bin/bash

for region in `aws ec2 describe-regions --output text | cut -f3`

do

echo -e "\n Finding Instances in region:'$region'"

aws ec2 describe-intances --region $region

done

**Search EC2**

$ aws ec2 describe-regions --output text

REGIONS ec2.ap-south-1.amazonaws.com ap-south-1

REGIONS ec2.eu-west-3.amazonaws.com eu-west-3

REGIONS ec2.eu-west-2.amazonaws.com eu-west-2

REGIONS ec2.eu-west-1.amazonaws.com eu-west-1

REGIONS ec2.ap-northeast-2.amazonaws.com ap-northeast-2

REGIONS ec2.ap-northeast-1.amazonaws.com ap-northeast-1

REGIONS ec2.sa-east-1.amazonaws.com sa-east-1

REGIONS ec2.ca-central-1.amazonaws.com ca-central-1

REGIONS ec2.ap-southeast-1.amazonaws.com ap-southeast-1

REGIONS ec2.ap-southeast-2.amazonaws.com ap-southeast-2

REGIONS ec2.eu-central-1.amazonaws.com eu-central-1

REGIONS ec2.us-east-1.amazonaws.com us-east-1

REGIONS ec2.us-east-2.amazonaws.com us-east-2

REGIONS ec2.us-west-1.amazonaws.com us-west-1

REGIONS ec2.us-west-2.amazonaws.com us-west-2

**Running the bash .sh**

$ ./ describeec2.sh

Finding Instances in region:'ap-south-1'

{

"Reservations": []

}

Finding Instances in region:'eu-west-3'

{

"Reservations": []

}

Finding Instances in region:'eu-west-2'

{

"Reservations": []

}

Finding Instances in region:'eu-west-1'

{

"Reservations": []

}

Finding Instances in region:'ap-northeast-2'

{

"Reservations": []

}

Finding Instances in region:'ap-northeast-1'

{

"Reservations": []

}

Finding Instances in region:'sa-east-1'

{

"Reservations": []

}

Finding Instances in region:'ca-central-1'

{

"Reservations": []

}

Finding Instances in region:'ap-southeast-1'

{

"Reservations": []

}

Finding Instances in region:'ap-southeast-2'

{

"Reservations": []

}

Finding Instances in region:'eu-central-1'

{

"Reservations": []

}

Finding Instances in region:'us-east-1'

{

"Reservations": []

}

Finding Instances in region:'us-east-2'

{

"Reservations": []

}

Finding Instances in region:'us-west-1'

{

"Reservations": []

}

Finding Instances in region:'us-west-2'

{

"Reservations": []

}

**Search for AMI**

$aws ec2 describe-images

$ sudo vim image.sh

aws --region us-east-1 ec2 describe-images --owners 099720109477\ -> (the company which produce the images example ubuntu \

--filters Name=root-device-type,Values=ebs \

Name=architecture,Values=x86\_64 \

Name=name,Values='\*hvm-ssd/ubuntu-precise-12.04\*' --query 'sort\_by(Image, &Name) [-1].ImageID

$ ./images.sh

**Creating a VPC and Subnets-CLI**

$ sudo vim createvpc.sh

**Bash**

vpcid=`aws ec2 create-vpc --cidr-block 10.0.0.0/16 --output text | cut -f7`

echo -e "$vpcid"

aws ec2 create-subnet --vpc-id "$vpcid" --cidr-block 10.0.1.0/24

aws ec2 create-subnet --vpc-id "$vpcid" --cidr-block 10.0.2.0/24

$ sudo chmod 777 createvpc.sh

$ ./createvpc.sh

**Output**

vpc-04f436f88ec053d86

{

"Subnet": {

"AvailabilityZone": "us-east-1c",

"AvailableIpAddressCount": 251,

"CidrBlock": "10.0.1.0/24",

"DefaultForAz": false,

"MapPublicIpOnLaunch": false,

"State": "pending",

"SubnetId": "subnet-011dbfcc92e75e27a",

"VpcId": "vpc-04f436f88ec053d86",

"AssignIpv6AddressOnCreation": false,

"Ipv6CidrBlockAssociationSet": []

}

}

{

"Subnet": {

"AvailabilityZone": "us-east-1c",

"AvailableIpAddressCount": 251,

"CidrBlock": "10.0.2.0/24",

"DefaultForAz": false,

"MapPublicIpOnLaunch": false,

"State": "pending",

"SubnetId": "subnet-0895a173358279d0b",

"VpcId": "vpc-04f436f88ec053d86",

"AssignIpv6AddressOnCreation": false,

"Ipv6CidrBlockAssociationSet": []

}

}

Deleting EC2-VPC

aws ec2 delete-vpc --vpc-id vpc-a01106c2



