Docker on AWS ECS:

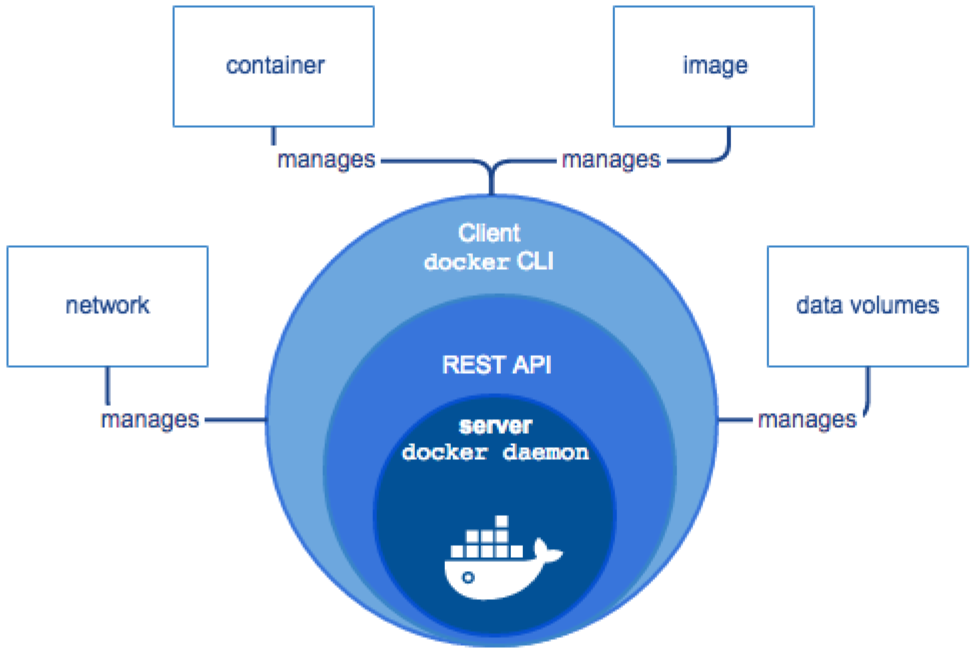
Docker Introduction:

- Docker is an open platform tool for developing, shipping and running applications.
- It is based on the idea that you can package your code with dependencies into an application container.
- In general explanation, think of the shipping containers used for intermodal shipping. You put your package (code and dependencies) in the container, ship it using a boat or train (laptop or cloud) and when it arrives at its destination it runs just like it did when it was shipped.

Docker Engine:

It is a client-server application with the following components:

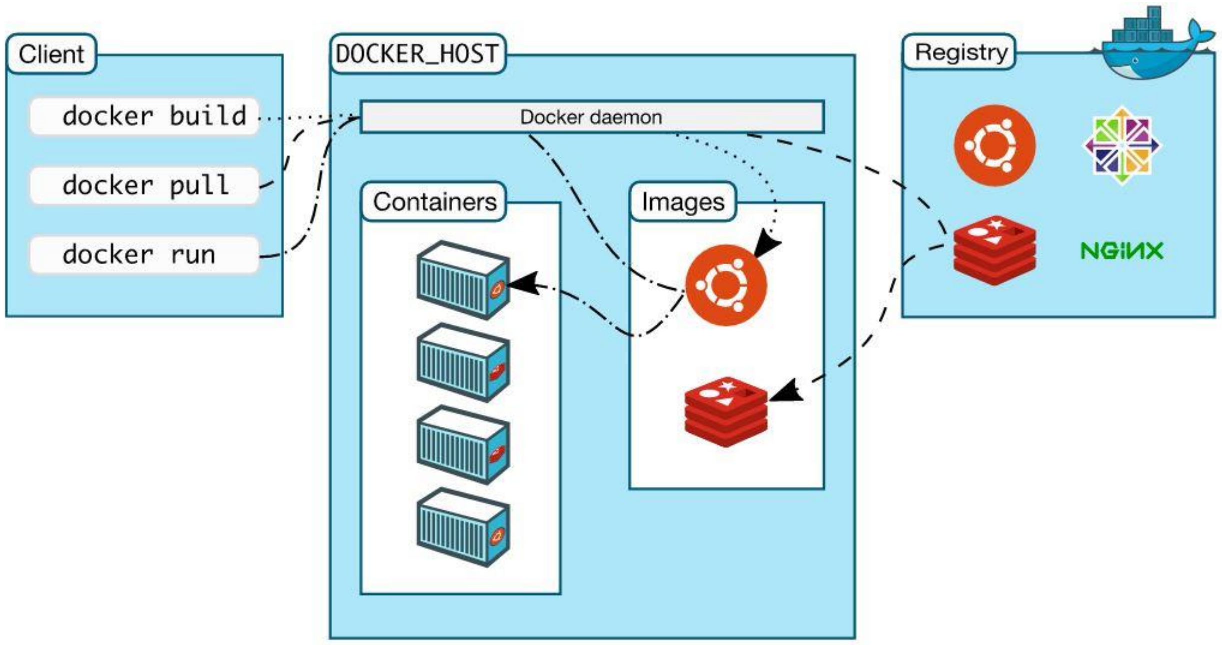
- A server which is a type of long-running program called daemon process.
- A REST (Representation State Transfer) API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client.



The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. The daemon creates and manages Docker objects, such as images, containers, networks, and data volumes.

Sameer M
M1036298

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate via sockets or through a REST API.



Docker Daemon:

The Docker daemon runs on a host machine. The user uses the docker client to interact with the daemon.

Docker Client:

Docker Client is present in the form of docker binary, which is the primary user interface to Docker. It accepts commands and configuration flags from the user and communicates with a Docker daemon.

Inside Docker:

Internally Docker contains images, containers and registries.

Docker images:

A Docker image is a read-only template with instructions for creating a docker container. For Ex, an image might contain an Ubuntu operating system with Apache web server and your web application installed. You can build or update images from scratch or download and use images created by others. An image may be based on, or may extend, one or more other images.

A docker image is described in text file called a Dockerfile, which has a simple, well-defined syntax.

Docker containers:

A Docker container is a runnable instance of a Docker image. You can run, start, stop, move, or delete a container using Docker API or CLI commands. When you run a container, you can provide configuration metadata such as networking information or environment variables. Docker containers are the **run** component of Docker.

Docker registries:

A docker registry is a library of images. A registry can be public or private, and can be on the same server as the Docker daemon or Docker client, or on a totally separate server.

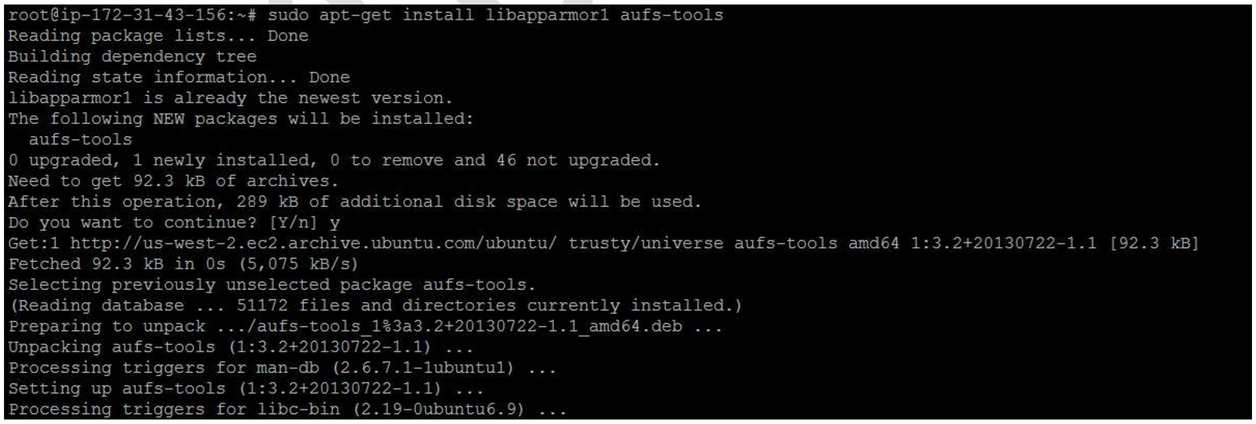
Introduction to Amazon ECS:

Amazon EC2 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 of Amazon Elastic Compute Cloud (Amazon EC2) instances. It treats entire cluster as a single computing resource.

Installing Docker on Ubuntu:

First step is to install few dependencies for Docker,

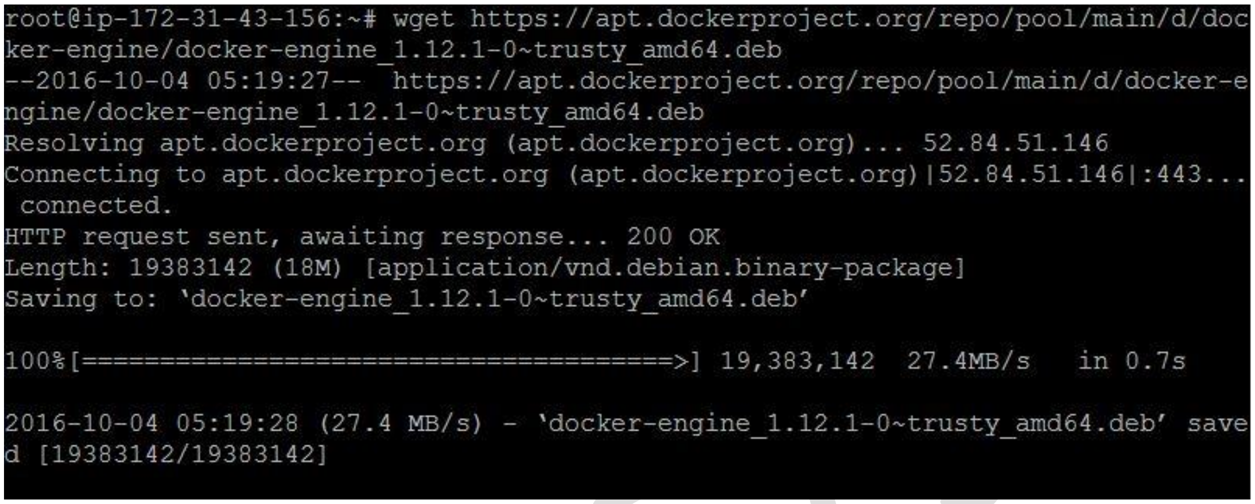
# sudo apt-get update
sudo apt-get install libapparmor1 aufs-tools



```
root@ip-172-31-43-156:~# sudo apt-get install libapparmor1 aufs-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
libapparmor1 is already the newest version.
The following NEW packages will be installed:
  aufs-tools
0 upgraded, 1 newly installed, 0 to remove and 46 not upgraded.
Need to get 92.3 kB of archives.
After this operation, 289 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu/ trusty/universe aufs-tools amd64 1:3.2+20130722-1.1 [92.3 kB]
Fetched 92.3 kB in 0s (5,075 kB/s)
Selecting previously unselected package aufs-tools.
(Reading database ... 51172 files and directories currently installed.)
Preparing to unpack .../aufs-tools_1%3a3.2+20130722-1.1_amd64.deb ...
Unpacking aufs-tools (1:3.2+20130722-1.1) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Setting up aufs-tools (1:3.2+20130722-1.1) ...
Processing triggers for libc-bin (2.19-0ubuntu6.9) ...
```

Now, get the docker package by downloading it from the following command:

wget https://apt.dockerproject.org/repo/pool/main/d/docker-engine/docker-engine_1.12.1-0~trusty_amd64.deb

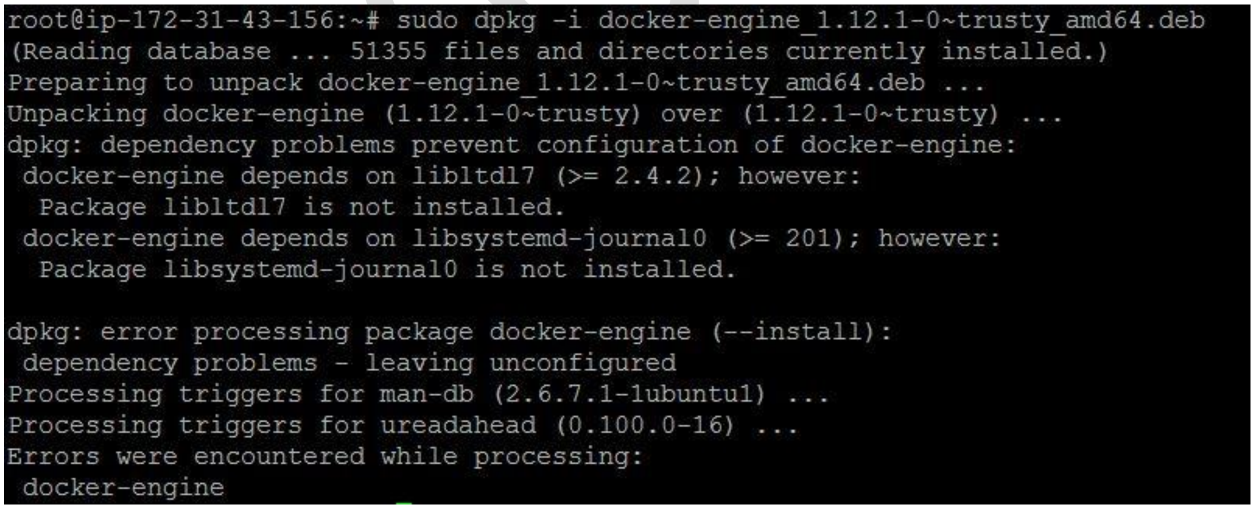


```
root@ip-172-31-43-156:~# wget https://apt.dockerproject.org/repo/pool/main/d/doc
ker-engine/docker-engine_1.12.1-0~trusty_amd64.deb
--2016-10-04 05:19:27--  https://apt.dockerproject.org/repo/pool/main/d/docker-e
ngine/docker-engine_1.12.1-0~trusty_amd64.deb
Resolving apt.dockerproject.org (apt.dockerproject.org)... 52.84.51.146
Connecting to apt.dockerproject.org (apt.dockerproject.org)|52.84.51.146|:443...
 connected.
HTTP request sent, awaiting response... 200 OK
Length: 19383142 (18M) [application/vnd.debian.binary-package]
Saving to: 'docker-engine_1.12.1-0~trusty_amd64.deb'

100%[=======================================>] 19,383,142  27.4MB/s   in 0.7s

2016-10-04 05:19:28 (27.4 MB/s) - 'docker-engine_1.12.1-0~trusty_amd64.deb' save
d [19383142/19383142]
```

Next install the docker from the downloaded package by using **dpkg** command:

sudo dpkg -i docker-engine_1.12.1-0~trusty_amd64.deb

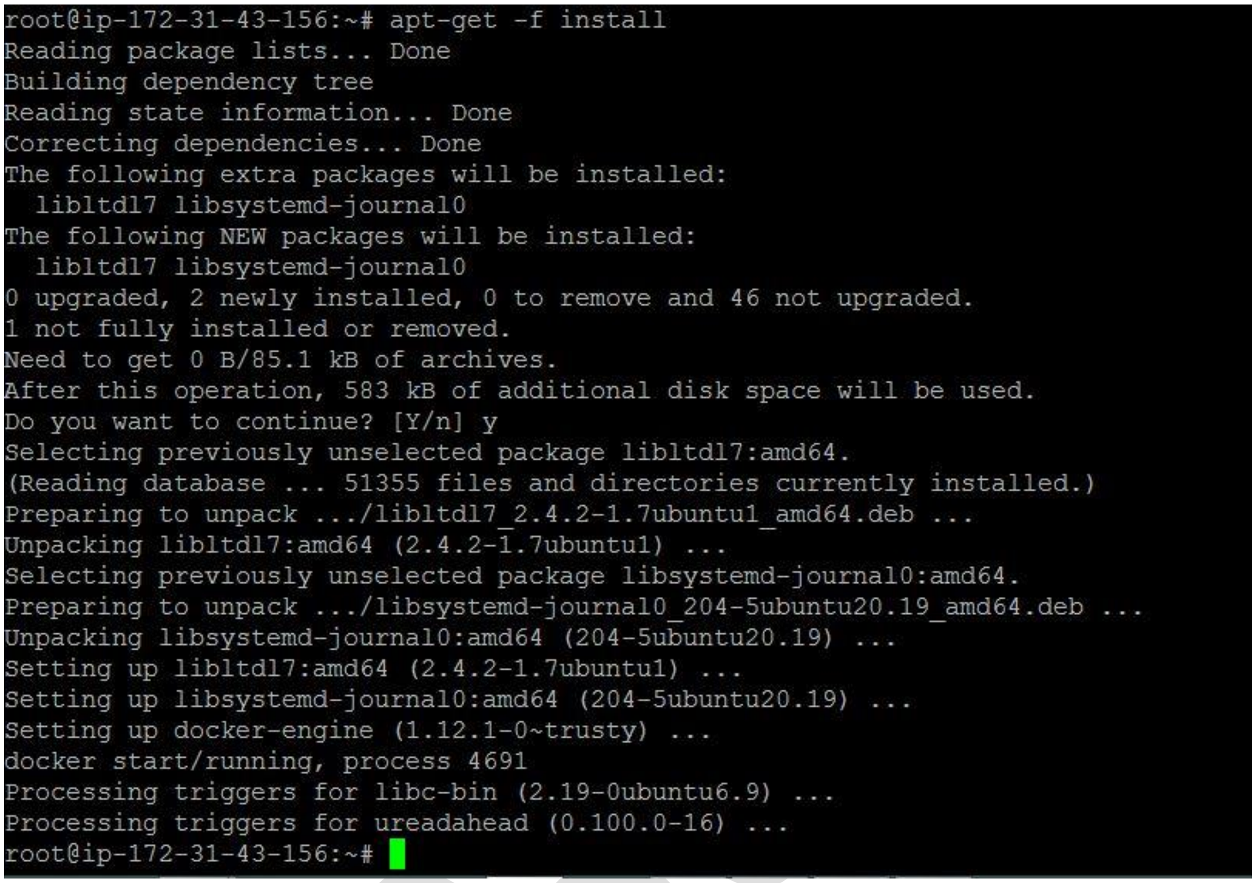
It will throw following error as shown below,



```
root@ip-172-31-43-156:~# sudo dpkg -i docker-engine_1.12.1-0~trusty_amd64.deb
(Reading database ... 51355 files and directories currently installed.)
Preparing to unpack docker-engine_1.12.1-0~trusty_amd64.deb ...
Unpacking docker-engine (1.12.1-0~trusty) over (1.12.1-0~trusty) ...
dpkg: dependency problems prevent configuration of docker-engine:
 docker-engine depends on libltdl7 (>= 2.4.2); however:
  Package libltdl7 is not installed.
 docker-engine depends on libsystemd-journal0 (>= 201); however:
  Package libsystemd-journal0 is not installed.

dpkg: error processing package docker-engine (--install):
 dependency problems - leaving unconfigured
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Processing triggers for ureadahead (0.100.0-16) ...
Errors were encountered while processing:
 docker-engine
```

This error might occur if there are any broken dependencies, in order to fix this error use **–f** flag in the following command and then try to reinstall Docker.

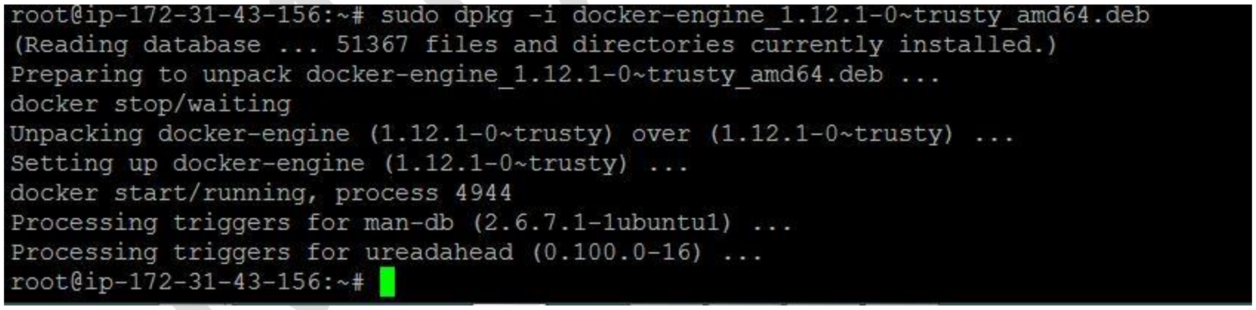
# apt-get –f install



```
root@ip-172-31-43-156:~# apt-get -f install
Reading package lists... Done
Building dependency tree
Reading state information... Done
Correcting dependencies... Done
The following extra packages will be installed:
  libltdl7 libsystemd-journal0
The following NEW packages will be installed:
  libltdl7 libsystemd-journal0
0 upgraded, 2 newly installed, 0 to remove and 46 not upgraded.
1 not fully installed or removed.
Need to get 0 B/85.1 kB of archives.
After this operation, 583 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Selecting previously unselected package libltdl7:amd64.
(Reading database ... 51355 files and directories currently installed.)
Preparing to unpack .../libltdl7_2.4.2-1.7ubuntu1_amd64.deb ...
Unpacking libltdl7:amd64 (2.4.2-1.7ubuntu1) ...
Selecting previously unselected package libsystemd-journal0:amd64.
Preparing to unpack .../libsystemd-journal0_204-5ubuntu20.19_amd64.deb ...
Unpacking libsystemd-journal0:amd64 (204-5ubuntu20.19) ...
Setting up libltdl7:amd64 (2.4.2-1.7ubuntu1) ...
Setting up libsystemd-journal0:amd64 (204-5ubuntu20.19) ...
Setting up docker-engine (1.12.1-0~trusty) ...
docker start/running, process 4691
Processing triggers for libc-bin (2.19-0ubuntu6.9) ...
Processing triggers for ureadahead (0.100.0-16) ...
root@ip-172-31-43-156:~#
```

Now install Docker:

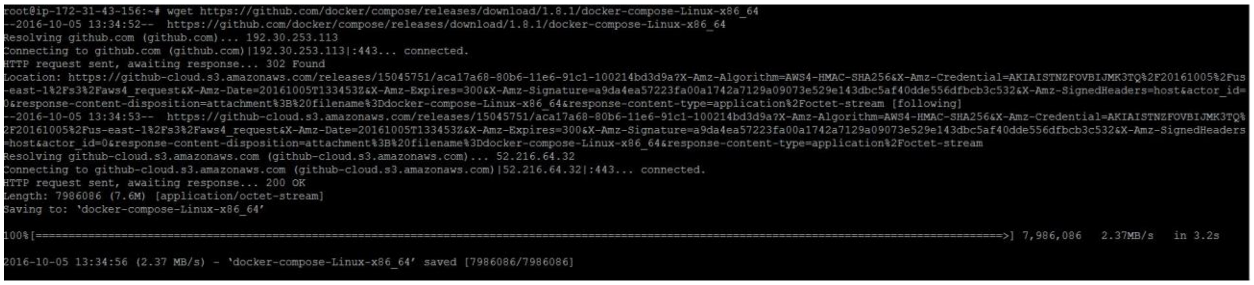
sudo dpkg –i docker-engine_1.12.1-0~trusty_amd64.deb



```
root@ip-172-31-43-156:~# sudo dpkg -i docker-engine_1.12.1-0~trusty_amd64.deb
(Reading database ... 51367 files and directories currently installed.)
Preparing to unpack docker-engine_1.12.1-0~trusty_amd64.deb ...
docker stop/waiting
Unpacking docker-engine (1.12.1-0~trusty) over (1.12.1-0~trusty) ...
Setting up docker-engine (1.12.1-0~trusty) ...
docker start/running, process 4944
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Processing triggers for ureadahead (0.100.0-16) ...
root@ip-172-31-43-156:~#
```

Next we need to install docker compose for that we have to go to following link, (https://github.com/docker/compose/releases) and scroll to check for the latest release in download section, right click on the respective platform i.e., windows, Linux etc and copy link address. Go to terminal and execute the following command:

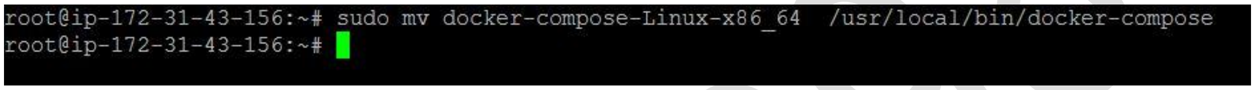
# wget https://github.com/docker/compose/releases/download/1.8.1/docker-compose-Linux-x86_64



```
root@ip-172-31-43-156:~# wget https://github.com/docker/compose/releases/download/1.8.1/docker-compose-Linux-x86_64
--2016-10-05 13:34:52--  https://github.com/docker/compose/releases/download/1.8.1/docker-compose-Linux-x86_64
Resolving github.com (github.com)... 192.30.253.113
Connecting to github.com (github.com)|192.30.253.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-cloud.s3.amazonaws.com/releases/15045751/aca17a68-80b6-11e6-91c1-100214bd3d9a?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAISTNZFOVBIJMK3TQ%2F20161005%2Fus
-east-1%2Fs3%2Faws4_request&X-Amz-Date=20161005T133453Z&X-Amz-Expires=300&X-Amz-Signature=a9da4ea57223fa00a1742a7129a09073e529e143dbc5af40dde556dfbcb3c532&X-Amz-SignedHeaders=host&actor_id=
0&response-content-disposition=attachment%3B%20filename%3Ddocker-compose-Linux-x86_64&response-content-type=application%2Foctet-stream [following]
--2016-10-05 13:34:53--  https://github-cloud.s3.amazonaws.com/releases/15045751/aca17a68-80b6-11e6-91c1-100214bd3d9a?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAISTNZFOVBIJMK3TQ%
2F20161005%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20161005T133453Z&X-Amz-Expires=300&X-Amz-Signature=a9da4ea57223fa00a1742a7129a09073e529e143dbc5af40dde556dfbcb3c532&X-Amz-SignedHeaders
=host&actor_id=0&response-content-disposition=attachment%3B%20filename%3Ddocker-compose-Linux-x86_64&response-content-type=application%2Foctet-stream
Resolving github-cloud.s3.amazonaws.com (github-cloud.s3.amazonaws.com)... 52.216.64.32
Connecting to github-cloud.s3.amazonaws.com (github-cloud.s3.amazonaws.com)|52.216.64.32|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7986086 (7.6M) [application/octet-stream]
Saving to: 'docker-compose-Linux-x86_64'

100%[=====================================================================================================================================>] 7,986,086   2.37MB/s   in 3.2s

2016-10-05 13:34:56 (2.37 MB/s) - 'docker-compose-Linux-x86_64' saved [7986086/7986086]
```

This will download the latest Docker-compose file for Linux platform. Move the Docker-compose without having to supply full path:

sudo mv docker-compose-Linux-x86_64 /usr/local/bin/docker-compose

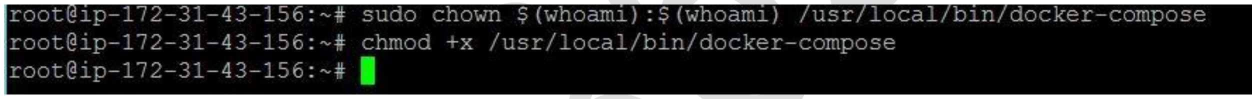


```
root@ip-172-31-43-156:~# sudo mv docker-compose-Linux-x86_64  /usr/local/bin/docker-compose
root@ip-172-31-43-156:~#
```

Change the ownership permission & give executable permission

sudo chown $(whoami): $(whoami) /usr/local/bin/docker-compose

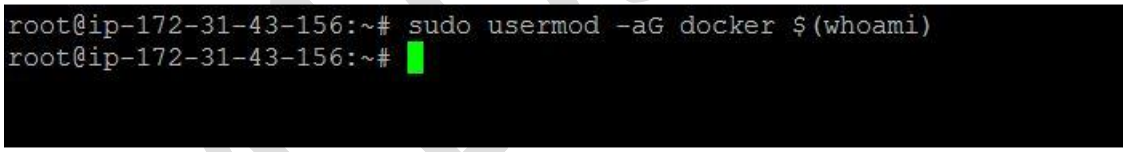
chmod +x /usr/local/bin/docker-compose



```
root@ip-172-31-43-156:~# sudo chown $(whoami):$(whoami) /usr/local/bin/docker-compose
root@ip-172-31-43-156:~# chmod +x /usr/local/bin/docker-compose
root@ip-172-31-43-156:~#
```

We have to run Docker without root all the time so,

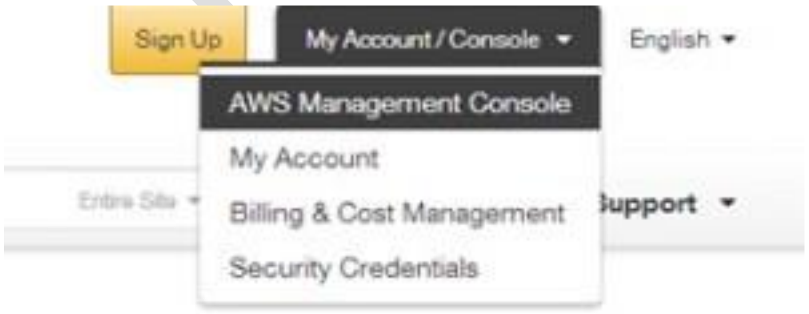
sudo usermod –aG docker $(whoami)



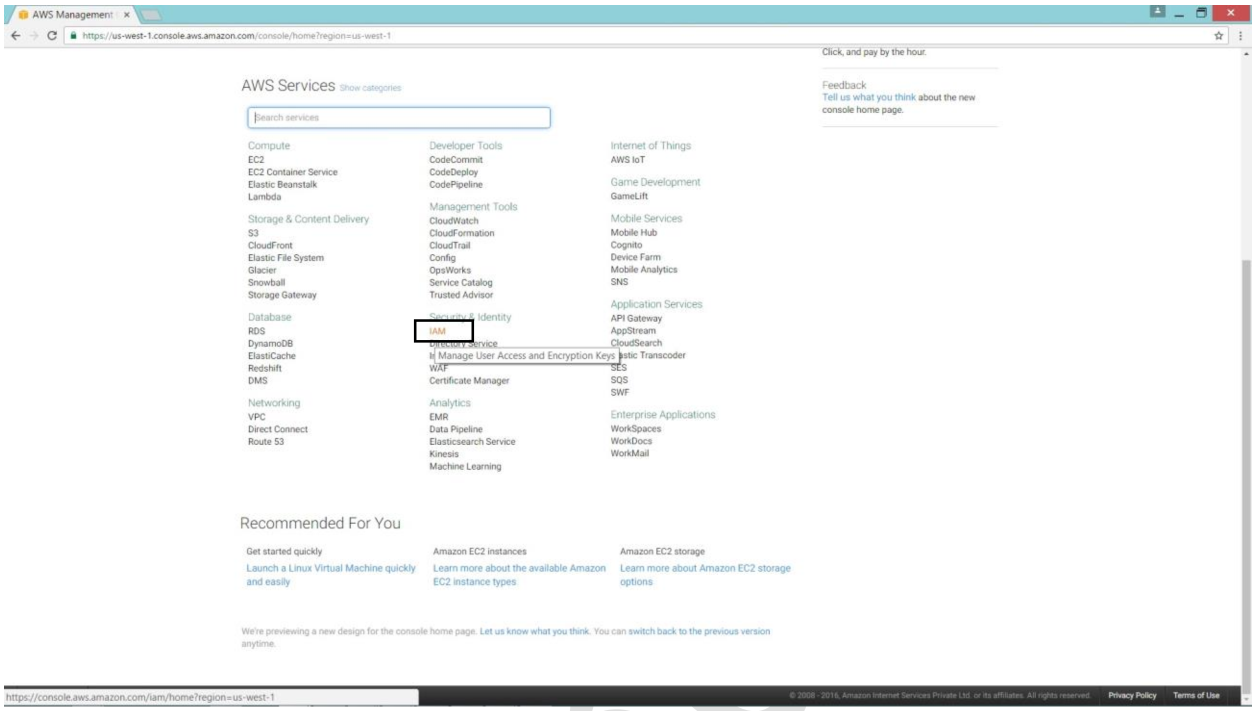
```
root@ip-172-31-43-156:~# sudo usermod -aG docker $(whoami)
root@ip-172-31-43-156:~#
```

Creating an IAM in AWS:

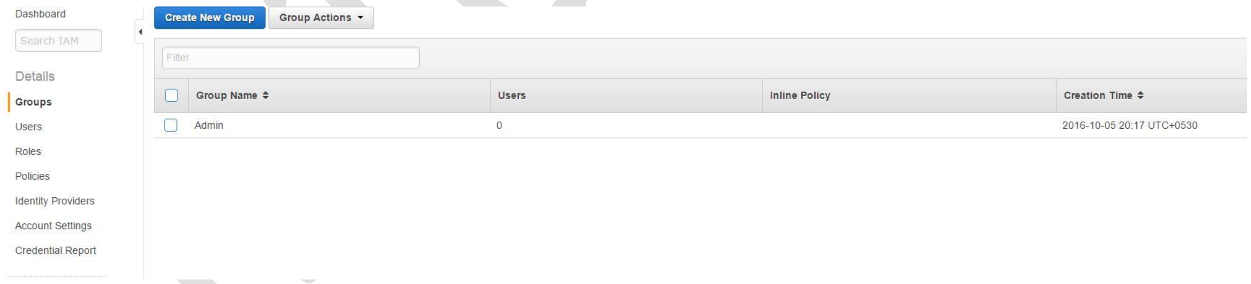
Login to your **AWS account** and go to **AWS Management Console** and change the location to **US East (N. Virginia)**, since some of the features won't support in other location.



Under **Security & Identity** select **IAM (Identity & Access Management)**



Under **IAM** in left pane select **Groups**, under **Groups** select **Create New Group.** Give any desired Group Name and click on Next Step at the bottom. Now attach any policy, in our case we select **AdministratorAccess** and click on **Next Step,** review it and click on **Create Group.** Now we can see our newly created Group in the Dashboard as shown below:



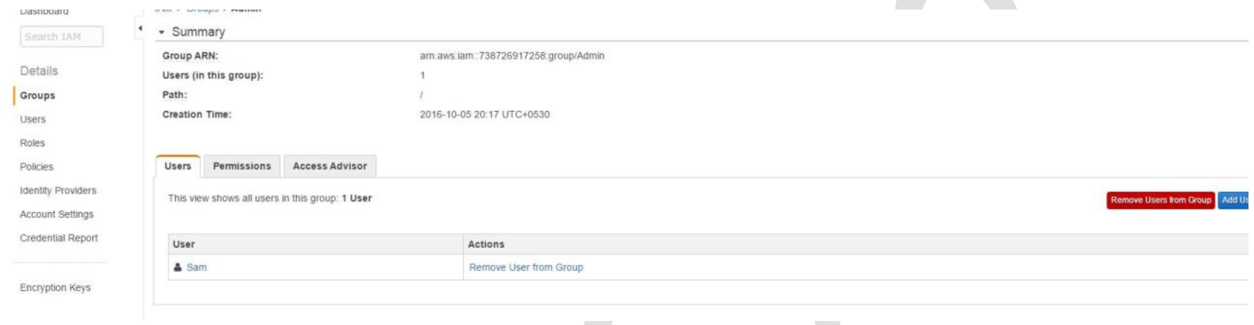
Group Name	Users	Inline Policy	Creation Time
Admin	0		2016-10-05 20:17 UTC+0530

Now we need to add user to that **Groups**, click on **Users and select Create User.** Give any name which you want to assign it to that created group. Now click on **Create.** We have to click on download credentials, we will discuss about it in further steps. If we open that credentials.csv file, we can see some contents like,

1. User Name
2. Access Key ID
3. Secret Access Key

Copy it or Save as it through your favorite editor and save it by name **resource.txt,** so that we can keep all necessary credentials and IAM URL links in this file.

Now, again go back to Groups and click on the created group, then we can see an option called **Add Users to Group.** Click on it. It will add user to that group.



Group ARN: arn:aws:iam::738726917258:group/Admin
Users (in this group): 1
Path: /
Creation Time: 2016-10-05 20:17 UTC+0530

This view shows all users in this group: 1 User

User	Actions
Sam	Remove User from Group

Go to Users and create a custom password for the user. We can do this by checking on the user name and under **User Actions,** select **Manage Password.** Now click on radio button **Assign a custom password**. Enter the password of your choice & click **Apply**.

Now go on the main Dashboard we can see IAM users sign-in link. Copy and paste it in credentials.txt file. Next time when we login to our AWS account we should use that link to login through IAM.

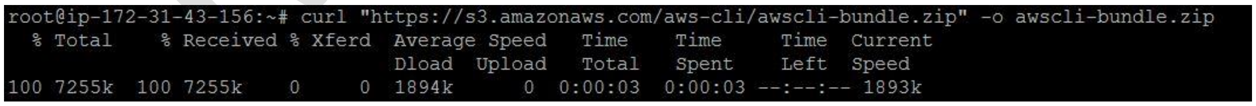
Installing and Configuring AWS CLI:

Open the terminal and check whether curl is installed or not by using **curl --version**. If it is not installed then install it by using the following command:

# sudo apt-get install curl

Now we have download AWS CLI bundle by using curl command:

curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o "awscli-bundle.zip"

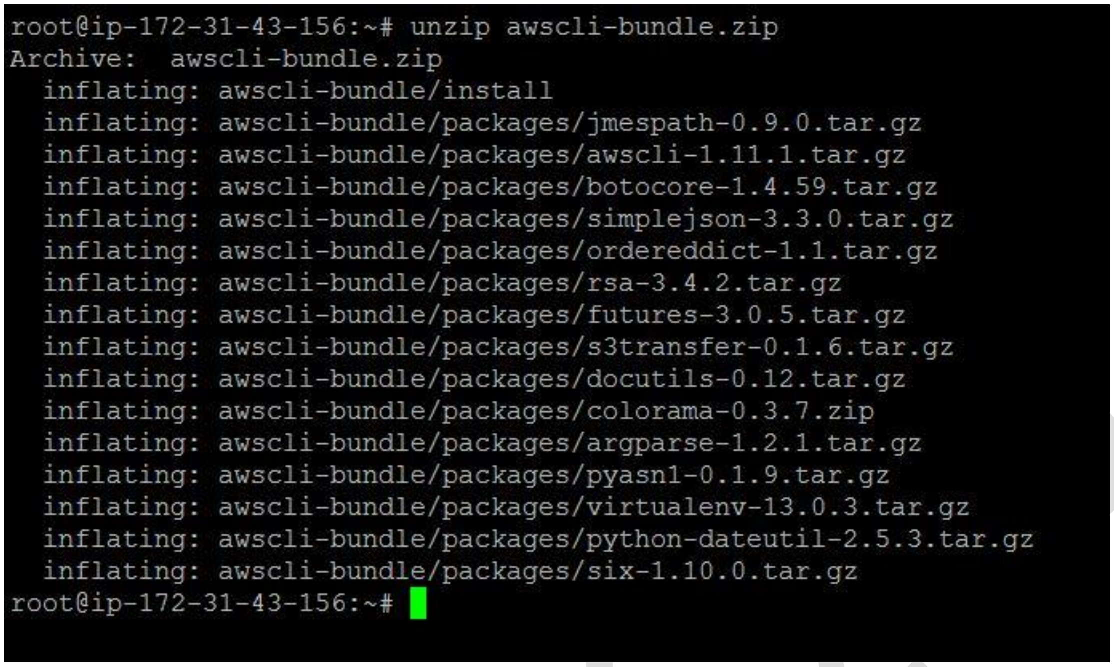


```
root@ip-172-31-43-156:~# curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o awscli-bundle.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 7255k  100 7255k    0     0  1894k      0  0:00:03  0:00:03 --:--:-- 1893k
```

Unzip the awscli-bundle.zip file, but before that we have to install unzip by using:

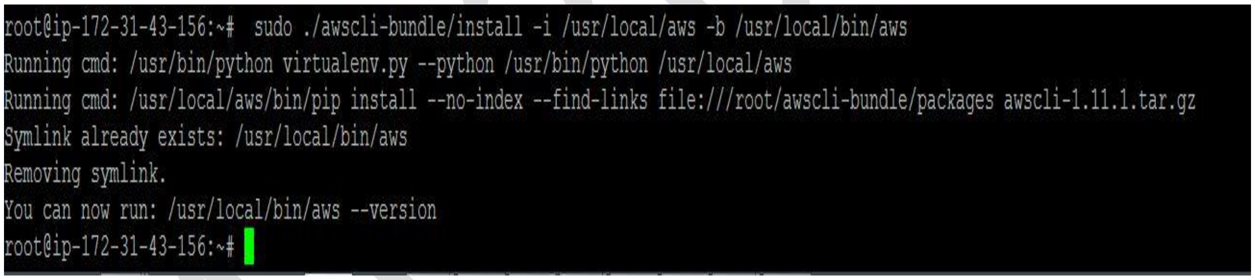
# sudo apt-get install unzip

unzip awscli-bundle.zip



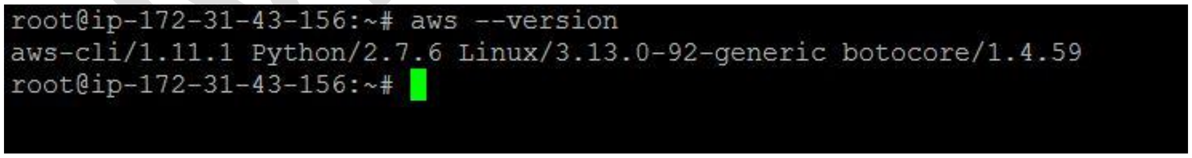
```
root@ip-172-31-43-156:~# unzip awscli-bundle.zip
Archive:  awscli-bundle.zip
  inflating: awscli-bundle/install
  inflating: awscli-bundle/packages/jmespath-0.9.0.tar.gz
  inflating: awscli-bundle/packages/awscli-1.11.1.tar.gz
  inflating: awscli-bundle/packages/botocore-1.4.59.tar.gz
  inflating: awscli-bundle/packages/simplejson-3.3.0.tar.gz
  inflating: awscli-bundle/packages/ordereddict-1.1.tar.gz
  inflating: awscli-bundle/packages/rsa-3.4.2.tar.gz
  inflating: awscli-bundle/packages/futures-3.0.5.tar.gz
  inflating: awscli-bundle/packages/s3transfer-0.1.6.tar.gz
  inflating: awscli-bundle/packages/docutils-0.12.tar.gz
  inflating: awscli-bundle/packages/colorama-0.3.7.zip
  inflating: awscli-bundle/packages/argparse-1.2.1.tar.gz
  inflating: awscli-bundle/packages/pyasn1-0.1.9.tar.gz
  inflating: awscli-bundle/packages/virtualenv-13.0.3.tar.gz
  inflating: awscli-bundle/packages/python-dateutil-2.5.3.tar.gz
  inflating: awscli-bundle/packages/six-1.10.0.tar.gz
root@ip-172-31-43-156:~#
```

Now install it using following command:
# sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws



```
root@ip-172-31-43-156:~#  sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
Running cmd: /usr/bin/python virtualenv.py --python /usr/bin/python /usr/local/aws
Running cmd: /usr/local/aws/bin/pip install --no-index --find-links file:///root/awscli-bundle/packages awscli-1.11.1.tar.gz
Symlink already exists: /usr/local/bin/aws
Removing symlink.
You can now run: /usr/local/bin/aws --version
root@ip-172-31-43-156:~#
```

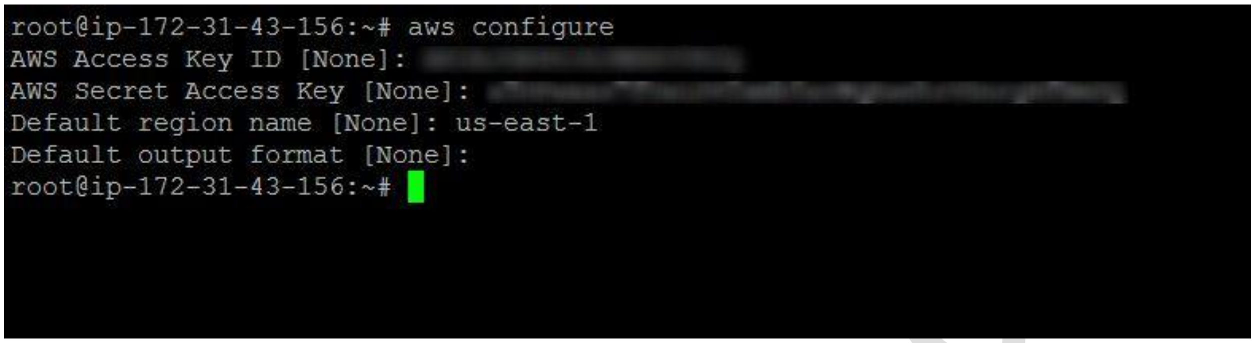
Now run,
# aws --version



```
root@ip-172-31-43-156:~# aws --version
aws-cli/1.11.1 Python/2.7.6 Linux/3.13.0-92-generic botocore/1.4.59
root@ip-172-31-43-156:~#
```

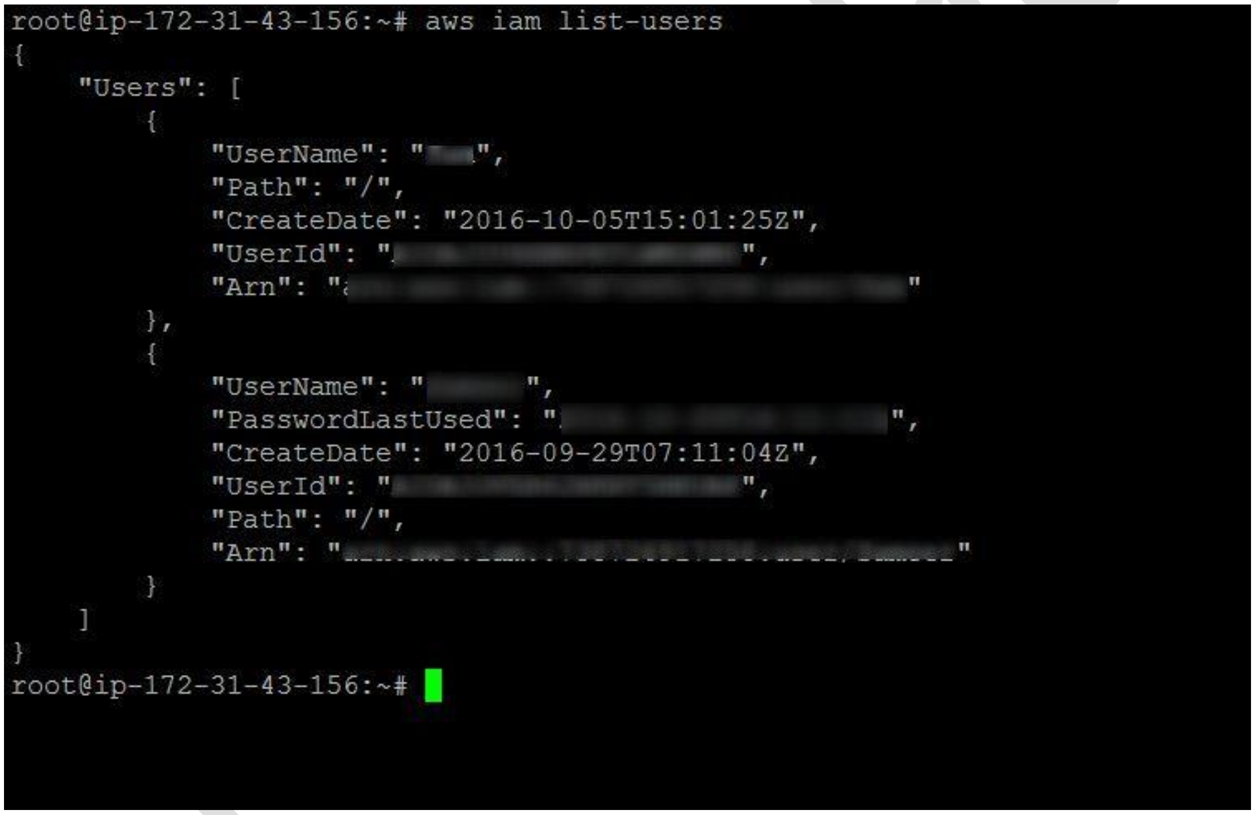
We have AWS CLI installed, let's configure AWS by supplying with access keys from resources.txt file:
# aws configure
Once we hit enter, first it prompts for AWS Access Key ID, copy it from resources.txt and paste it in terminal. Now hit enter, it prompts for AWS Secret Access Key, copy it and repeat the process.

Next, it prompts for Default Region name, provide it as **us-east-1**. Now, it will prompt for Default output format, just hit enter so that it can take JSON as default format.



```
root@ip-172-31-43-156:~# aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [None]: us-east-1
Default output format [None]:
root@ip-172-31-43-156:~#
```

Now we have configure AWS is configured, let's verify it by using command:
# aws iam list-users



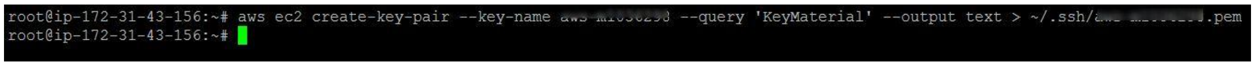
```
root@ip-172-31-43-156:~# aws iam list-users
{
    "Users": [
        {
            "UserName": "[redacted]",
            "Path": "/",
            "CreateDate": "2016-10-05T15:01:25Z",
            "UserId": "[redacted]",
            "Arn": "[redacted]"
        },
        {
            "UserName": "[redacted]",
            "PasswordLastUsed": "[redacted]",
            "CreateDate": "2016-09-29T07:11:04Z",
            "UserId": "[redacted]",
            "Path": "/",
            "Arn": "[redacted]"
        }
    ]
}
root@ip-172-31-43-156:~#
```

Creating SSH Keypair:

Logging in through SSH is much more secure than using a password. In order to do that go to terminal & type the following command:

aws ec2 create-key-pair --key-name <keyname> --query 'KeyMaterial' --output text > ~/.ssh/<filename>.pem

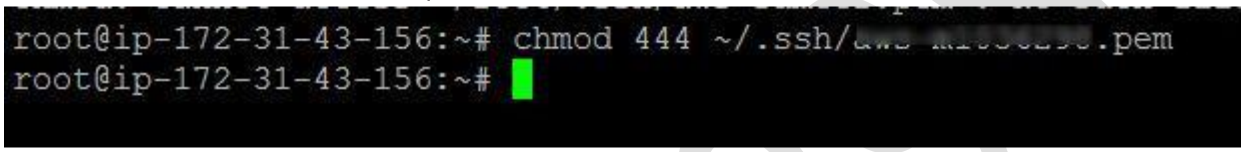
Note: Keyname and filename.pem can be user choice



```
root@ip-172-31-43-156:~# aws ec2 create-key-pair --key-name [redacted] --query 'KeyMaterial' --output text > ~/.ssh/[redacted].pem
root@ip-172-31-43-156:~# 
```

Set up proper permissions on keypair:

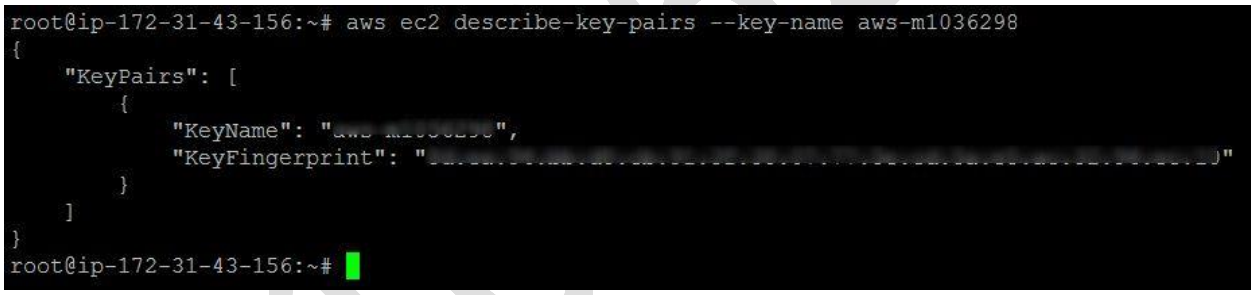
# chmod 444 ~/.ssh/<filename>.pem



```
root@ip-172-31-43-156:~# chmod 444 ~/.ssh/[redacted].pem
root@ip-172-31-43-156:~# 
```

Now verify whether it works,

aws ec2 describe-key-pairs --key-name <keyname>



```
root@ip-172-31-43-156:~# aws ec2 describe-key-pairs --key-name aws-m1036298
{
    "KeyPairs": [
        {
            "KeyName": "[redacted]",
            "KeyFingerprint": "[redacted]"
        }
    ]
}
root@ip-172-31-43-156:~# 
```

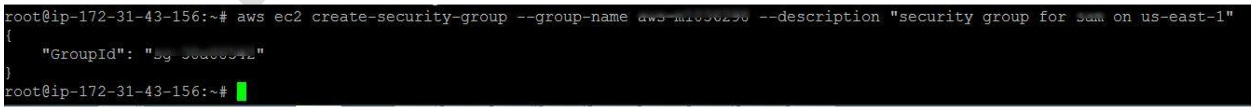
We can delete the keypair by using,

# aws ec2 delete-key-pair --key-name <keyname>

Creating Security Groups:

We can assume security groups as firewall for your instances or services that are running in AWS. Open terminal and type

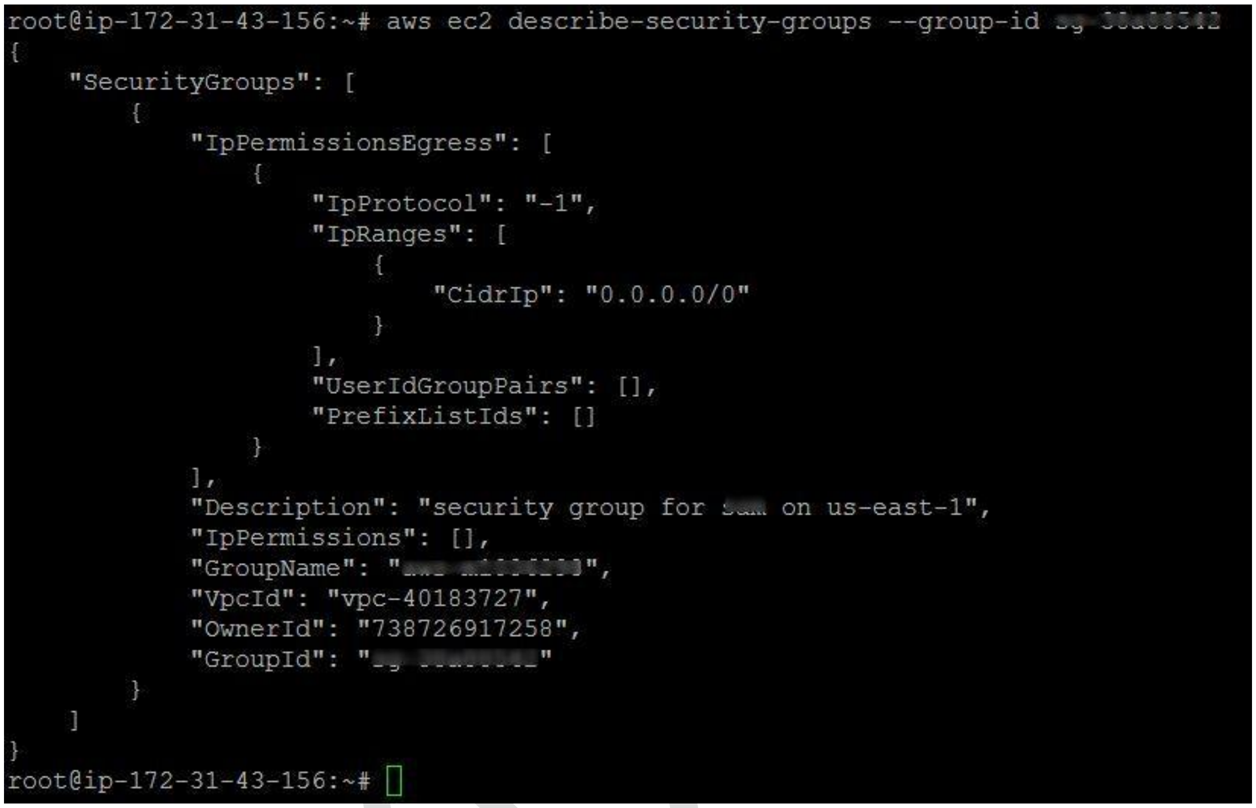
aws ec2 create-security-group --group-name <groupname> --description "any description"



```
root@ip-172-31-43-156:~# aws ec2 create-security-group --group-name [redacted] --description "security group for [redacted] on us-east-1"
{
    "GroupId": "[redacted]"
}
root@ip-172-31-43-156:~# 
```

Once it outputs Security Group ID, copy it & paste in resources.txt file for further use.

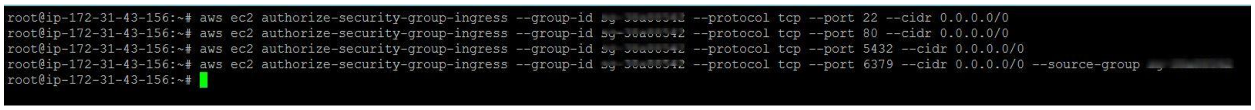
We can describe security group in short detailed manner by using following command:
# aws ec2 describe-security-groups --group-id <SGID>



```
root@ip-172-31-43-156:~# aws ec2 describe-security-groups --group-id [redacted]
{
    "SecurityGroups": [
        {
            "IpPermissionsEgress": [
                {
                    "IpProtocol": "-1",
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
                    ],
                    "UserIdGroupPairs": [],
                    "PrefixListIds": []
                }
            ],
            "Description": "security group for [redacted] on us-east-1",
            "IpPermissions": [],
            "GroupName": "[redacted]",
            "VpcId": "vpc-40183727",
            "OwnerId": "738726917258",
            "GroupId": "[redacted]"
        }
    ]
}
root@ip-172-31-43-156:~#
```

Now let's attach few rules to it such as allowing access from the outside world to specific ports.
# aws ec2 authorize-security-group-ingress --group-id <SGID> --protocol tcp --port 22 --cidr 0.0.0.0/0
# aws ec2 authorize-security-group-ingress --group-id <SGID> --protocol tcp --port 80 --cidr 0.0.0.0/0
# aws ec2 authorize-security-group-ingress --group-id <SGID> --protocol tcp --port 5432 --cidr 0.0.0.0/0
# aws ec2 authorize-security-group-ingress --group-id sg-38a88542 --protocol tcp --port 6379 --cidr 0.0.0.0/0 –source-group <SGID>

These commands open ports for SSH, HTTP, Postgres & Redist.



```
root@ip-172-31-43-156:~# aws ec2 authorize-security-group-ingress --group-id [redacted] --protocol tcp --port 22 --cidr 0.0.0.0/0
root@ip-172-31-43-156:~# aws ec2 authorize-security-group-ingress --group-id [redacted] --protocol tcp --port 80 --cidr 0.0.0.0/0
root@ip-172-31-43-156:~# aws ec2 authorize-security-group-ingress --group-id [redacted] --protocol tcp --port 5432 --cidr 0.0.0.0/0
root@ip-172-31-43-156:~# aws ec2 authorize-security-group-ingress --group-id [redacted] --protocol tcp --port 6379 --cidr 0.0.0.0/0 --source-group [redacted]
root@ip-172-31-43-156:~#
```

We can delete security Group by using,
# aws ec2 delete-security-groups --group-id <SGID>

Creating IAM Roles:

In order to create **IAM Roles** Login to **AWS** and Under **Security & Identity** select **IAM (Identity & Access Management)** click on **Roles**.

Under **Roles**, select **Create New Role.** Now type **ecsInstanceRole** and click on **Next Step.**

Ensure that selected radio button is **AWS Service Roles.** Now press **ctrl+F** & type **container** in search bar. Select **Amazon EC2 Role for EC2 Container Service** click **Select**, click on checkbox & **Next Step** & then **Create Role**.

Now we have to attach S**3** policy to it. Now click that checkbox & **Attach Policy**. In search bar type **S3** and check **AmazonS3ReadOnlyAccess** and click on **Attach Policy**.

Create one more role by **ecsServiceRole.** Now press **ctrl+F** & search for **container** in search bar and just like before check **AmazonEC2ContainerServiceRole and** click on Next Step**.** Now click on **Create Role**.

ECS Components:

1. Cluster: Cluster is a group of container instances that acts as a single computing resource.

2. Container Instances: Container Instance is an Amazon **EC2** Instance that is registered to be a part of specific cluster.

3. Container Agent: Container Agent is an open source tool which takes care of pluming to ensure an Amazon EC2 instance can register to a cluster.

4. Task Definitions: It describes how your applications docker image should run. It is a JSON configuration file.

5. Scheduler: Determines where a service will run on cluster by figuring out most optimal instance to run it on.

6. Services: Services is a long running task such as a web app.

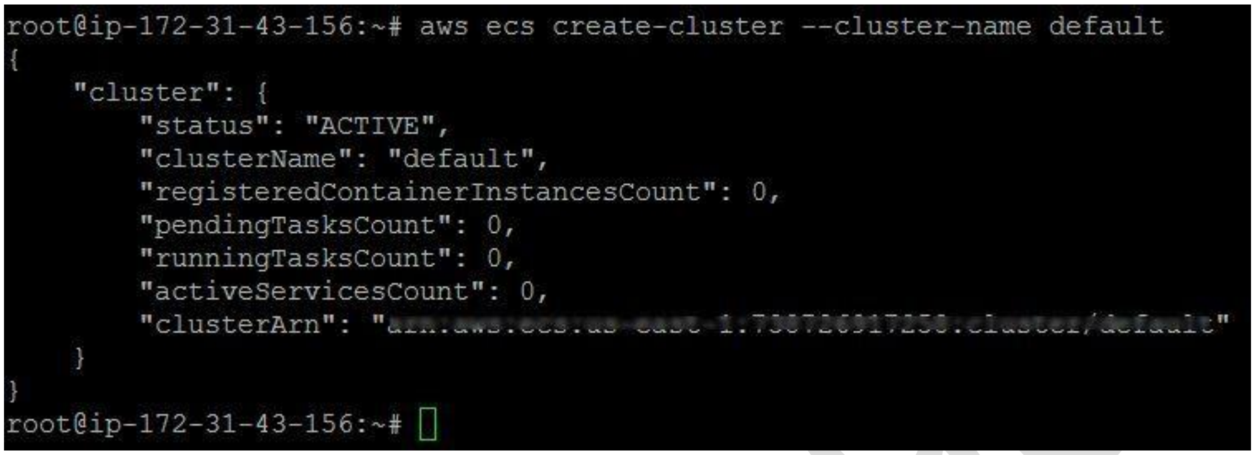
7. Tasks: Tasks are nothing but end result of task definition.

8. Amazon ECR: It is a fully managed docker registry. It is not a part of Amazon ECS but running a private registry is such a key component of deploying a dockerised application that it cannot be ignored.

9. Amazon ECS CLI: It is an open source tool for managing clusters.

Creating Cluster:

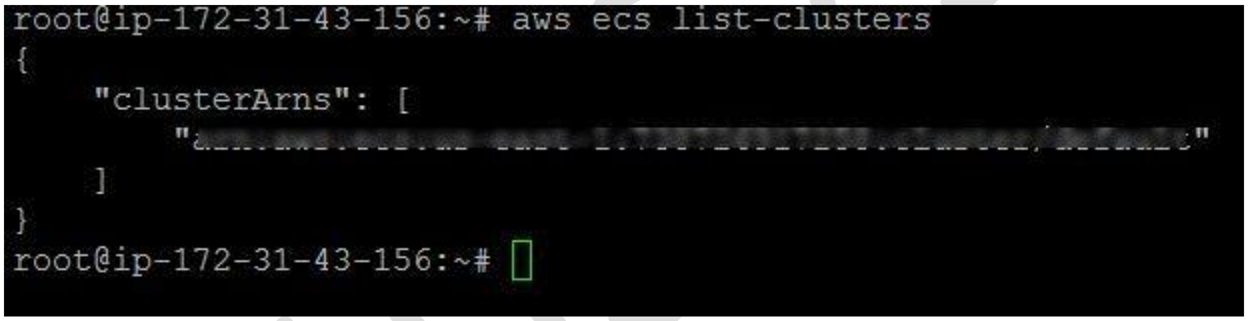
aws ecs create-cluster --cluster-name <clustername>



```
root@ip-172-31-43-156:~# aws ecs create-cluster --cluster-name default
{
    "cluster": {
        "status": "ACTIVE",
        "clusterName": "default",
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
        "clusterArn": "[redacted]"
    }
}
root@ip-172-31-43-156:~#
```

To list all the clusters use the following command:

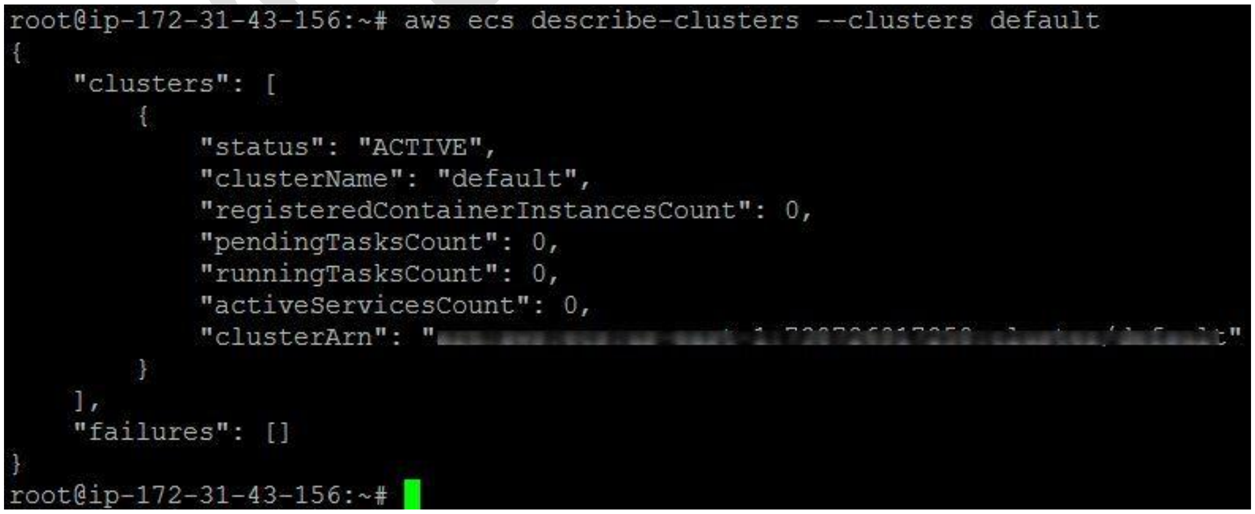
aws ecs list-clusters



```
root@ip-172-31-43-156:~# aws ecs list-clusters
{
    "clusterArns": [
        "[redacted]"
    ]
}
root@ip-172-31-43-156:~#
```

To see detailed description of clusters:

aws ecs describe-clusters --clusters <clustername>



```
root@ip-172-31-43-156:~# aws ecs describe-clusters --clusters default
{
    "clusters": [
        {
            "status": "ACTIVE",
            "clusterName": "default",
            "registeredContainerInstancesCount": 0,
            "pendingTasksCount": 0,
            "runningTasksCount": 0,
            "activeServicesCount": 0,
            "clusterArn": "[redacted]"
        }
    ],
    "failures": []
}
root@ip-172-31-43-156:~#
```

In the above output, the status of the cluster. The valid values are ACTIVE or INACTIVE. ACTIVE indicates that you can register container instances with the cluster and the associated instances can accept tasks.

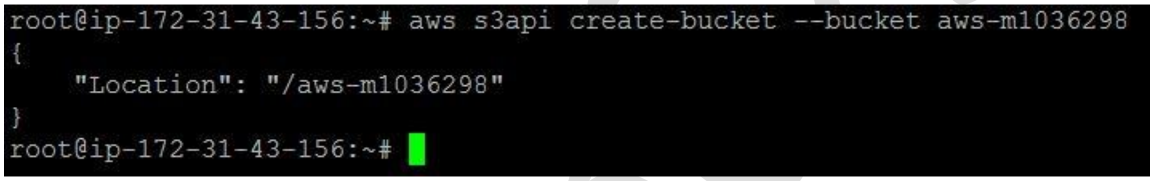
To delete a cluster:
# aws ecs delete-cluster --cluster <clustername>

Container Agent:

First we have to create Amazon S3 bucket. S3 is a way to store files in such a way that they are not tightened to specific an instance. The files are easily accessible to instances that we create other AWS services or even public internet. So let's use cli to create S3 bucket. A bucket is a namespace for the files we plan to store.

aws s3api create-bucket --bucket <bucketname>

Note: Bucketname should be unique across the region that they were being created on.



```
root@ip-172-31-43-156:~# aws s3api create-bucket --bucket aws-m1036298
{
    "Location": "/aws-m1036298"
}
root@ip-172-31-43-156:~#
```

Now we need to create the configuration file for this bucket. Create a directory by any name of your choice and get inside that directory and create a file by name **ecs.config.**
ecs.config is used to to connect container agent to EC2 instances.

Open ecs.config file with any editor and type:

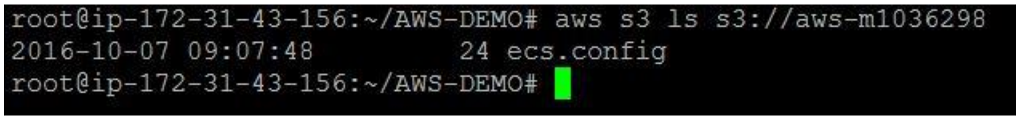
ECS_CLUSTER=<clustername>

Save it and exit.
Create one more file by name **copy-ecs-config-to-s3** and type:

```
#!/bin/bash
yum install -y aws-cli
aws s3 cp s3://<bucketname>/ecs.config /etc/ecs/ecs.config
```

Now we are setting up a bash script to copy this config file to ec2 instance upon creation.
Now we need to upload the config file to bucket:
# aws s3 cp ecs.config s3://aws-m1036298/ecs.config

Verify it by running the following command:



```
root@ip-172-31-43-156:~/AWS-DEMO# aws s3 ls s3://aws-m1036298
2016-10-07 09:07:48         24 ecs.config
root@ip-172-31-43-156:~/AWS-DEMO#
```

Creating Container Instances:

Container instance is an EC2 instance that has been registered to be a part of cluster. It connects via container agent.

Life cycle states:

ACTIVE & CONNECTED: When Container Agent registers to cluster, when container status= TRUE

ACTIVE & DISCONNECTED: This state occurs when we stop the container instance and status reaches to FALSE, which means current tasks which were running will stop too. If we want to start the container instance again and the container agent reconnects, then connection status will set to TRUE again.

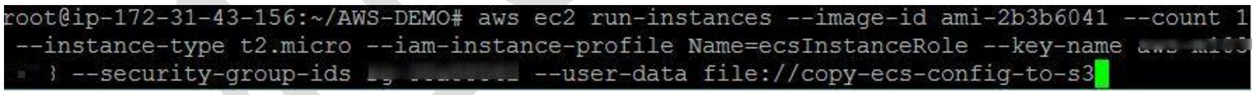
INACTIVE: This happens when we terminate or deregister a container instance. Inactive instance will not be seen as a part of container.

Now let's add container EC2 instance to the cluster we created.

Open up the resources.txt file and copy the security Group ID and type the following command:

# aws ec2 run-instances --image-id ami-2b3b6041 --count 1 --instance-type t2.micro --iam-instance-profile Name=ecsInstanceRole --key-name <keyname> --security-group-ids <SGID> --user-data file://copy-ecs-config-to-s3

In this command, image-id= ami-2b3b6041 is the official Image ID which is provided in amazon's official site.

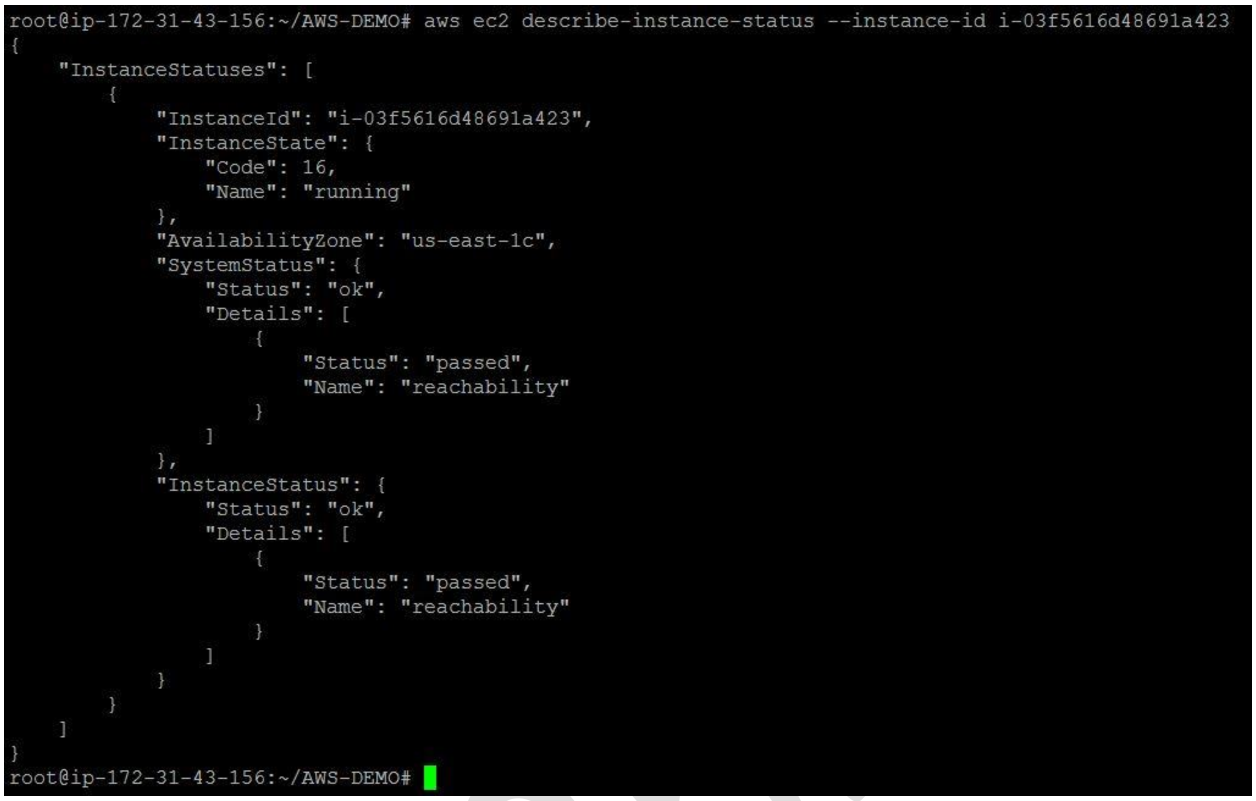


```
root@ip-172-31-43-156:~/AWS-DEMO# aws ec2 run-instances --image-id ami-2b3b6041 --count 1
 --instance-type t2.micro --iam-instance-profile Name=ecsInstanceRole --key-name [redacted]
 --security-group-ids [redacted] --user-data file://copy-ecs-config-to-s3
```

The output of this command will be long so couldn't include in this document. In the output, scroll down slowly and search for Instance ID. Once we get that copy it and paste it in resources.txt file, which will be useful in later stages.

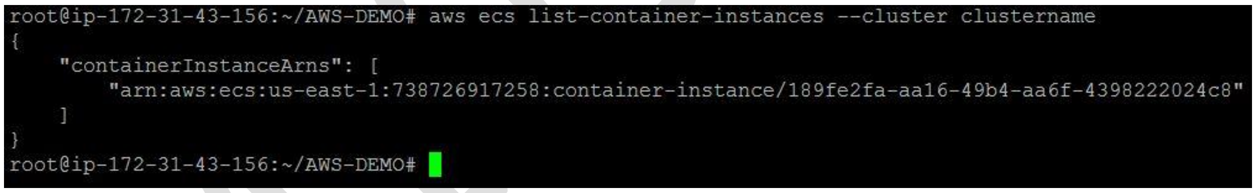
Now let's describe the instance which we created by using the command:

# aws ec2 describe-instance-status --instance-id i-03f5616d48691a423



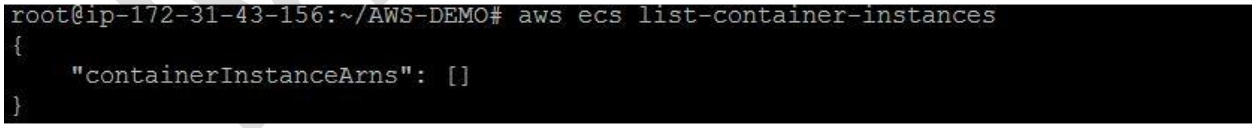
```
root@ip-172-31-43-156:~/AWS-DEMO# aws ec2 describe-instance-status --instance-id i-03f5616d48691a423
{
    "InstanceStatuses": [
        {
            "InstanceId": "i-03f5616d48691a423",
            "InstanceState": {
                "Code": 16,
                "Name": "running"
            },
            "AvailabilityZone": "us-east-1c",
            "SystemStatus": {
                "Status": "ok",
                "Details": [
                    {
                        "Status": "passed",
                        "Name": "reachability"
                    }
                ]
            },
            "InstanceStatus": {
                "Status": "ok",
                "Details": [
                    {
                        "Status": "passed",
                        "Name": "reachability"
                    }
                ]
            }
        }
    ]
}
root@ip-172-31-43-156:~/AWS-DEMO#
```

Let's check whether created instance has joined the cluster or not by using the command:
# aws ecs list-container-instances --cluster clustername



```
root@ip-172-31-43-156:~/AWS-DEMO# aws ecs list-container-instances --cluster clustername
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:738726917258:container-instance/189fe2fa-aa16-49b4-aa6f-4398222024c8"
    ]
}
root@ip-172-31-43-156:~/AWS-DEMO#
```

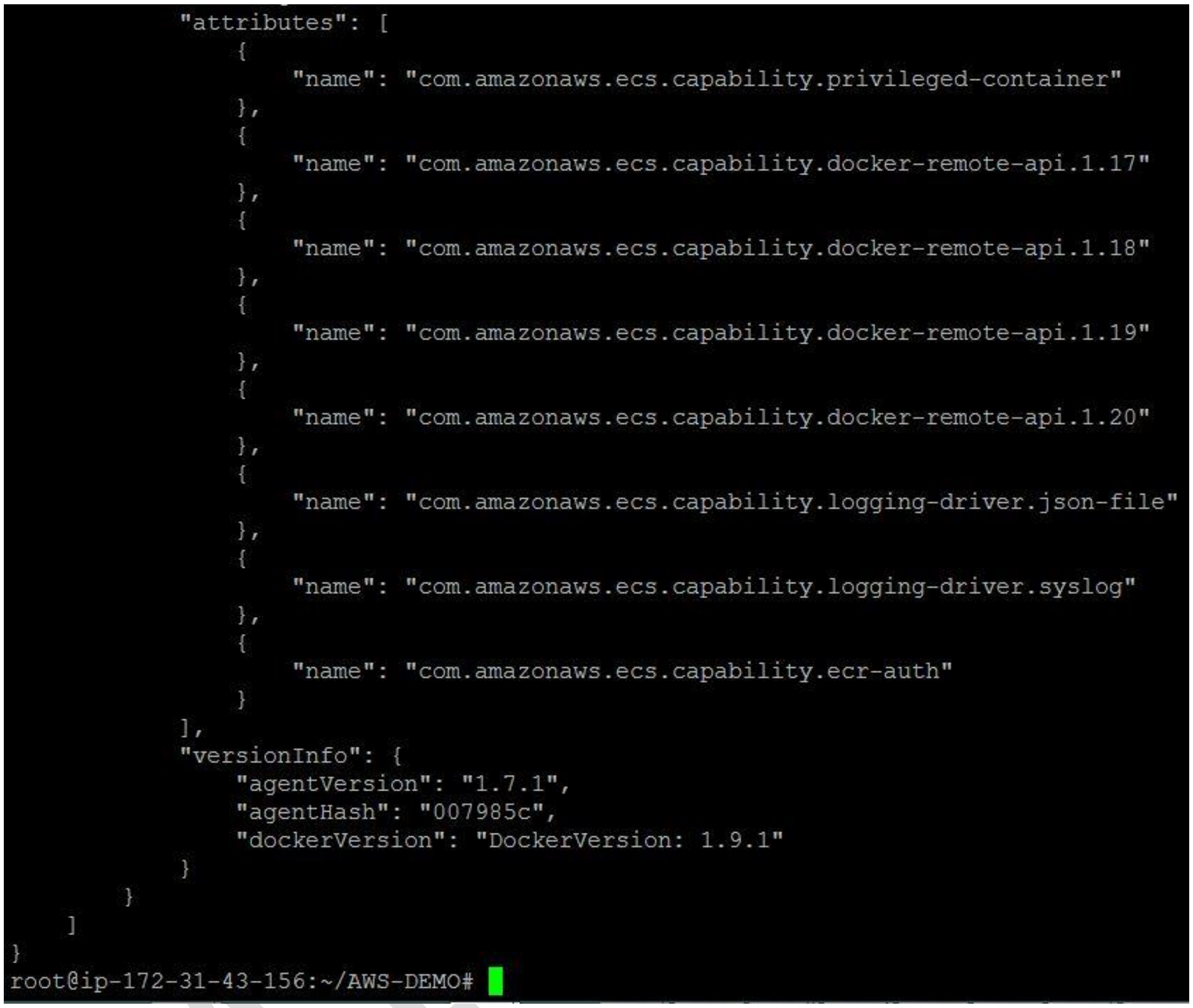
Suppose if we do not pass --cluster <clustername> in the above command, then the output will be as shown below:



```
root@ip-172-31-43-156:~/AWS-DEMO# aws ecs list-container-instances
{
    "containerInstanceArns": []
}
```

Now let's describe the container instance by using,
# aws ecs describe-container-instances --cluster clustername --container-instances <containerInstanceArns>

The output of this command will be lengthy so couldn't include the screenshot of entire ouput, but I have included only the last part of it, which is as shown below:



```
            "attributes": [
                {
                    "name": "com.amazonaws.ecs.capability.privileged-container"
                },
                {
                    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
                },
                {
                    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
                },
                {
                    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
                },
                {
                    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.20"
                },
                {
                    "name": "com.amazonaws.ecs.capability.logging-driver.json-file"
                },
                {
                    "name": "com.amazonaws.ecs.capability.logging-driver.syslog"
                },
                {
                    "name": "com.amazonaws.ecs.capability.ecr-auth"
                }
            ],
            "versionInfo": {
                "agentVersion": "1.7.1",
                "agentHash": "007985c",
                "dockerVersion": "DockerVersion: 1.9.1"
            }
        }
    ]
}
root@ip-172-31-43-156:~/AWS-DEMO#
```

In the above output we can notice that Docker is also installed in the instance which we have created.

We can terminate the instance by:
# aws ec2 terminate-instances --instance-ids <instanceID>

Creating Task Definitions:

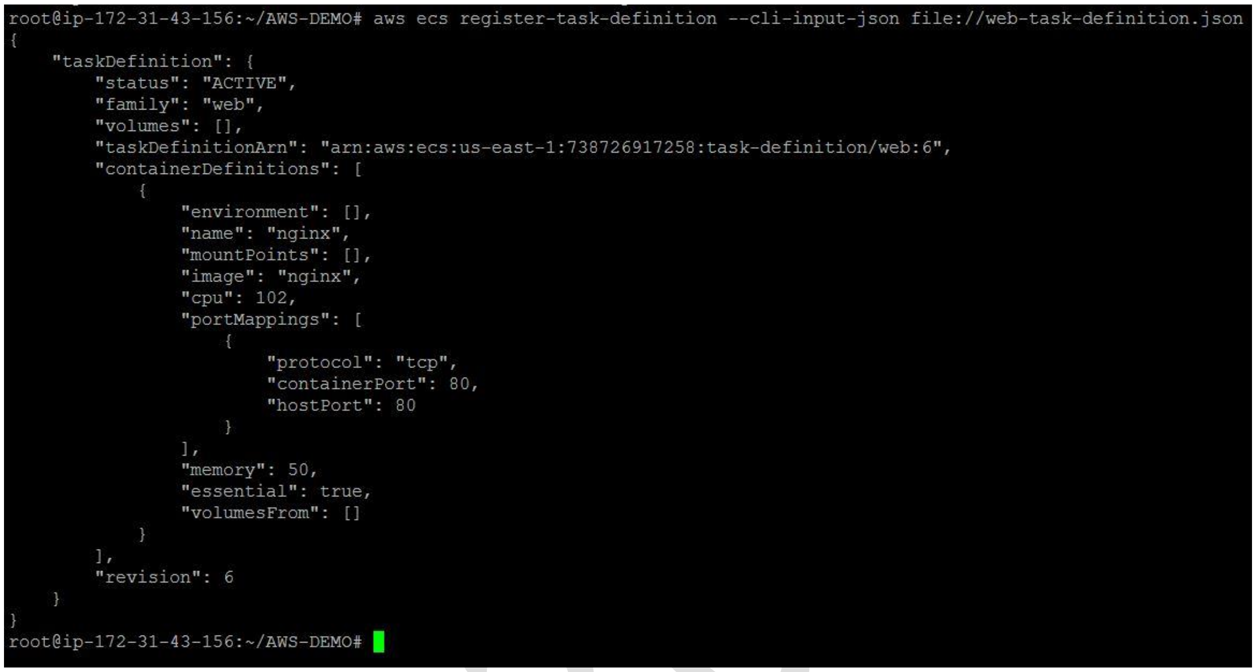
It describes how applications docker images should be ran. A task definition can control one or more task definition. Containers that are grouped in same task definition will be scheduled to be run on the same instance.

Now open up the terminal and create a file called **web-task-definition.json** in the same directory where ecs.config and copy-ecs-config-to-s3. Once the file is created by using **touch** command, open the file by any editor and type the below script:

```
{
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "nginx",
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80
        }
      ],
      "memory": 50,
      "cpu": 102
    }
  ],
  "family": "web"
}
```

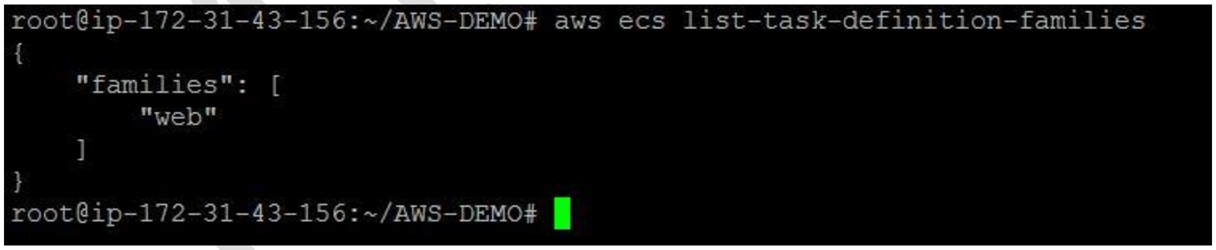
As we see it is a JSON file, in this specific task definition has only one task definition. We want our container name to be NGINX hence the name. We are pulling down the official NGINX image from the dockerhub. In port mapping, we want to expose port 80 for HTTP access (since we configured this in security group ID to allow incoming traffic). We are restricting our CPU memory for this task to 50MB of RAM, if the container goes beyond 50MB, it will be killed off automatically and gets restarted automatically after being killed. CPU is set to 102 which means that NGINX should consume about 10% of the CPU. The family can be anything, the reason for giving it as **WEB**, since the NGINX is a web server.

Now let's register the task definition into the cluster, to that open up the terminal and enter into the project directory and type the following command:
# aws ecs register-task-definition --cli-input-json file://web-task-definition.json

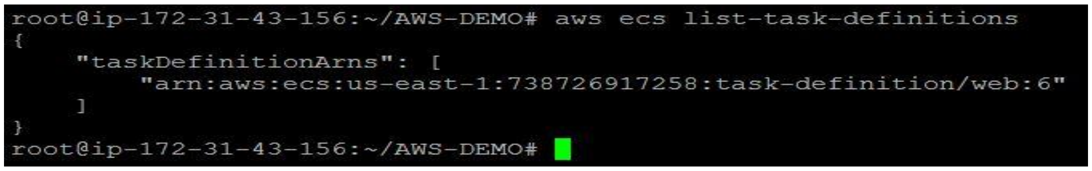

```
root@ip-172-31-43-156:~/AWS-DEMO# aws ecs register-task-definition --cli-input-json file://web-task-definition.json
{
    "taskDefinition": {
        "status": "ACTIVE",
        "family": "web",
        "volumes": [],
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:738726917258:task-definition/web:6",
        "containerDefinitions": [
            {
                "environment": [],
                "name": "nginx",
                "mountPoints": [],
                "image": "nginx",
                "cpu": 102,
                "portMappings": [
                    {
                        "protocol": "tcp",
                        "containerPort": 80,
                        "hostPort": 80
                    }
                ],
                "memory": 50,
                "essential": true,
                "volumesFrom": []
            }
        ],
        "revision": 6
    }
}
root@ip-172-31-43-156:~/AWS-DEMO#
```

In the above output, we notice that status is set to ACTIVE and family is WEB and it contains all the roles which we supplied in JSON file. In bottom we have something called revision count, which is nothing but number of times we tested the above command.

Now let's run a command to run all the task definition families we can do that by typing:
# aws ecs list-task-definition-families

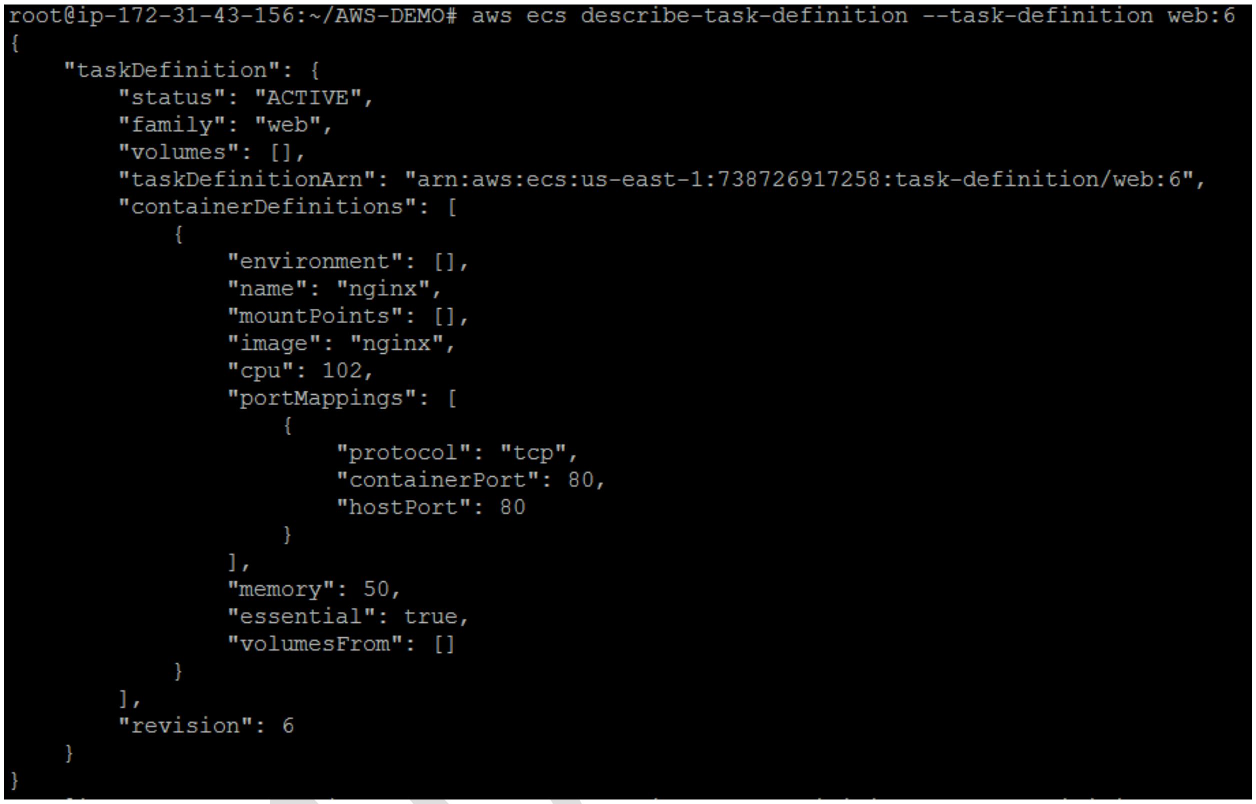

```
root@ip-172-31-43-156:~/AWS-DEMO# aws ecs list-task-definition-families
{
    "families": [
        "web"
    ]
}
root@ip-172-31-43-156:~/AWS-DEMO#
```

We can look out all the task definitions by running,
# aws ecs list-task-definitions


```
root@ip-172-31-43-156:~/AWS-DEMO# aws ecs list-task-definitions
{
    "taskDefinitionArns": [
        "arn:aws:ecs:us-east-1:738726917258:task-definition/web:6"
    ]
}
root@ip-172-31-43-156:~/AWS-DEMO#
```

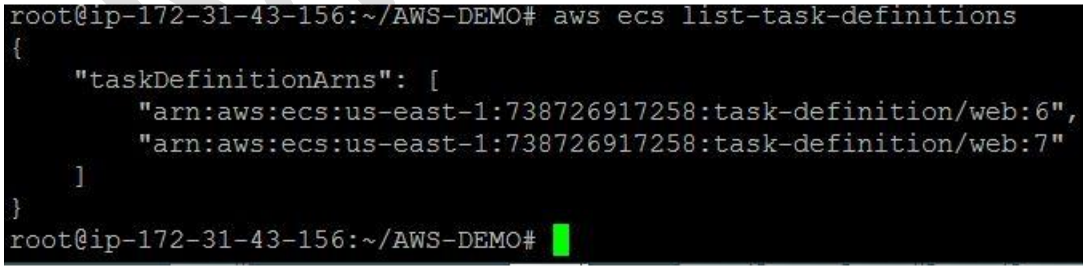
For the first time if we run this command, the revision count will be 1 and in my case it is 6.

If we want to describe one of our task definition, we can do that by using:
# aws ecs describe-task-definition --task-definition web:<revisioncount>



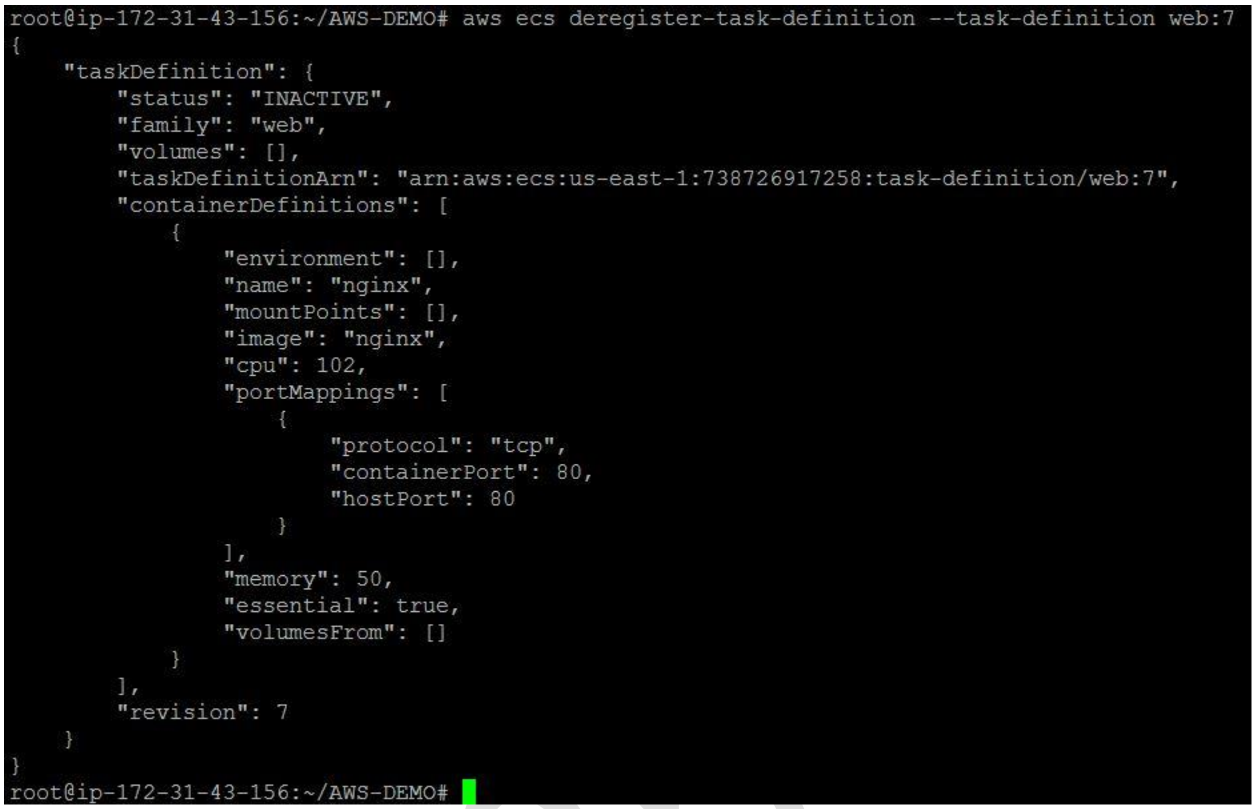
```
root@ip-172-31-43-156:~/AWS-DEMO# aws ecs describe-task-definition --task-definition web:6
{
    "taskDefinition": {
        "status": "ACTIVE",
        "family": "web",
        "volumes": [],
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:738726917258:task-definition/web:6",
        "containerDefinitions": [
            {
                "environment": [],
                "name": "nginx",
                "mountPoints": [],
                "image": "nginx",
                "cpu": 102,
                "portMappings": [
                    {
                        "protocol": "tcp",
                        "containerPort": 80,
                        "hostPort": 80
                    }
                ],
                "memory": 50,
                "essential": true,
                "volumesFrom": []
            }
        ],
        "revision": 6
    }
}
```

Just to see how revision count works, go ahead and type the following command:
# aws ecs register-task-definition --cli-input-json file://web-task-definition.json
This command is used earlier and every time we run this command, the revision count gets incremented by one.
Now let's run earlier command to list all the task definition again,



```
root@ip-172-31-43-156:~/AWS-DEMO# aws ecs list-task-definitions
{
    "taskDefinitionArns": [
        "arn:aws:ecs:us-east-1:738726917258:task-definition/web:6",
        "arn:aws:ecs:us-east-1:738726917258:task-definition/web:7"
    ]
}
root@ip-172-31-43-156:~/AWS-DEMO#
```

Now let's deregister the second task definition that we just created. We can do that by typing:
# aws ecs deregister-task-definition --task-definition web:<revisioncount>



```
root@ip-172-31-43-156:~/AWS-DEMO# aws ecs deregister-task-definition --task-definition web:7
{
    "taskDefinition": {
        "status": "INACTIVE",
        "family": "web",
        "volumes": [],
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:738726917258:task-definition/web:7",
        "containerDefinitions": [
            {
                "environment": [],
                "name": "nginx",
                "mountPoints": [],
                "image": "nginx",
                "cpu": 102,
                "portMappings": [
                    {
                        "protocol": "tcp",
                        "containerPort": 80,
                        "hostPort": 80
                    }
                ],
                "memory": 50,
                "essential": true,
                "volumesFrom": []
            }
        ],
        "revision": 7
    }
}
root@ip-172-31-43-156:~/AWS-DEMO#
```

We can notice that status is set to INACTIVE.

Now let's run a simple command:
# aws ecs register-task-definition help
This command is used to refer help menu for register task definition command. Amazon CLI is very well documented. So if we ever get stuck and want to learn about any specific command then go ahead and run help command or even **aws ecs help** command also.

In order to get a standard template for writing task definition file, type the following command:
# aws ecs register-task-definition --generate-cli-skeleton

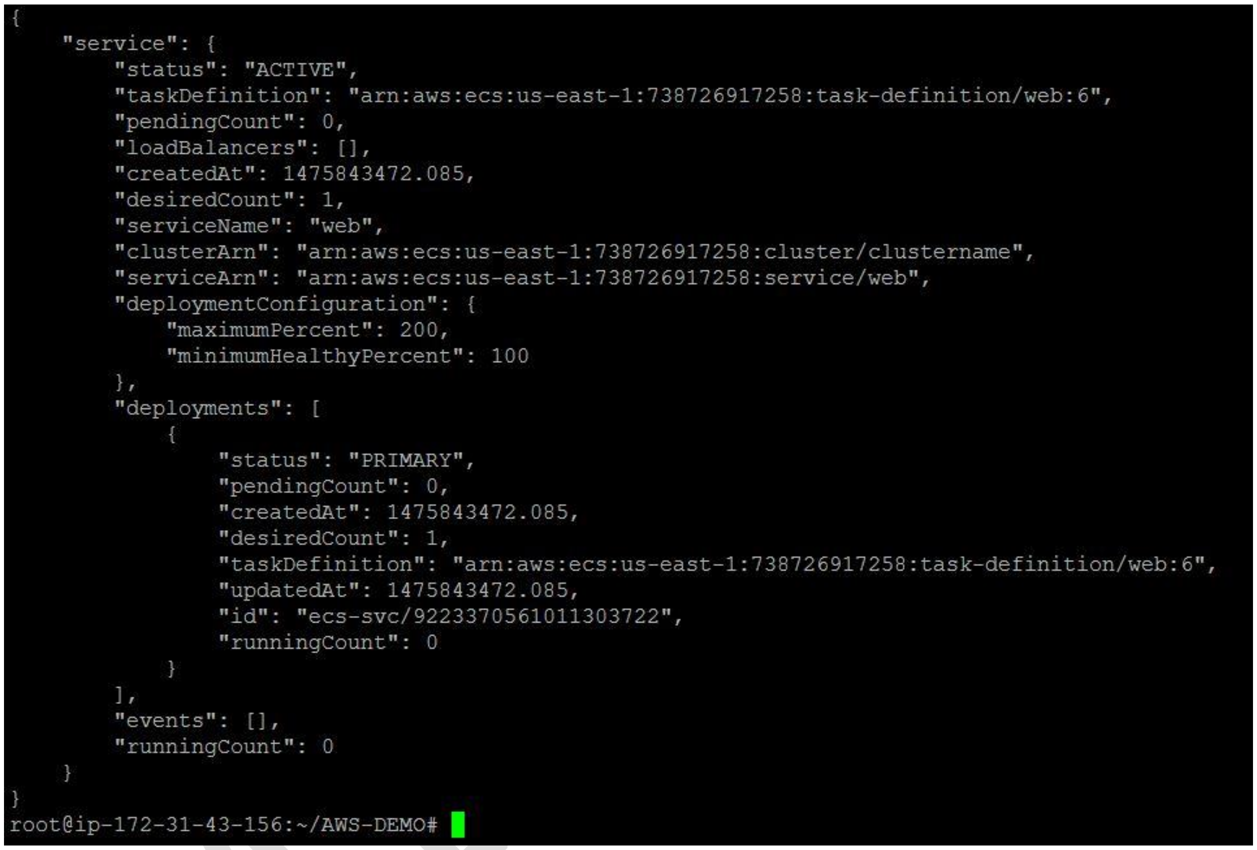
Scheduling Services:

Initial pre requisite is at least one of the container instance to be running. Now type the following command to create a service from task definition that we created earlier:

# aws ecs create-service --cluster <clustername> --service-name <servicename> --task-definition web --desired-count 1

(Desired count is set to 1 since we want one instance to be running)

The output will be:

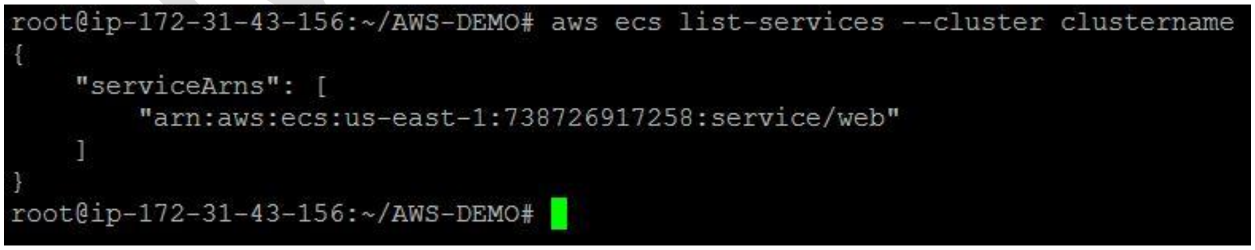


```
{
    "service": {
        "status": "ACTIVE",
        "taskDefinition": "arn:aws:ecs:us-east-1:738726917258:task-definition/web:6",
        "pendingCount": 0,
        "loadBalancers": [],
        "createdAt": 1475843472.085,
        "desiredCount": 1,
        "serviceName": "web",
        "clusterArn": "arn:aws:ecs:us-east-1:738726917258:cluster/clustername",
        "serviceArn": "arn:aws:ecs:us-east-1:738726917258:service/web",
        "deploymentConfiguration": {
            "maximumPercent": 200,
            "minimumHealthyPercent": 100
        },
        "deployments": [
            {
                "status": "PRIMARY",
                "pendingCount": 0,
                "createdAt": 1475843472.085,
                "desiredCount": 1,
                "taskDefinition": "arn:aws:ecs:us-east-1:738726917258:task-definition/web:6",
                "updatedAt": 1475843472.085,
                "id": "ecs-svc/9223370561011303722",
                "runningCount": 0
            }
        ],
        "events": [],
        "runningCount": 0
    }
}
root@ip-172-31-43-156:~/AWS-DEMO#
```

Now let’s list the services by running the command:

# aws ecs list-services --cluster <clustername>

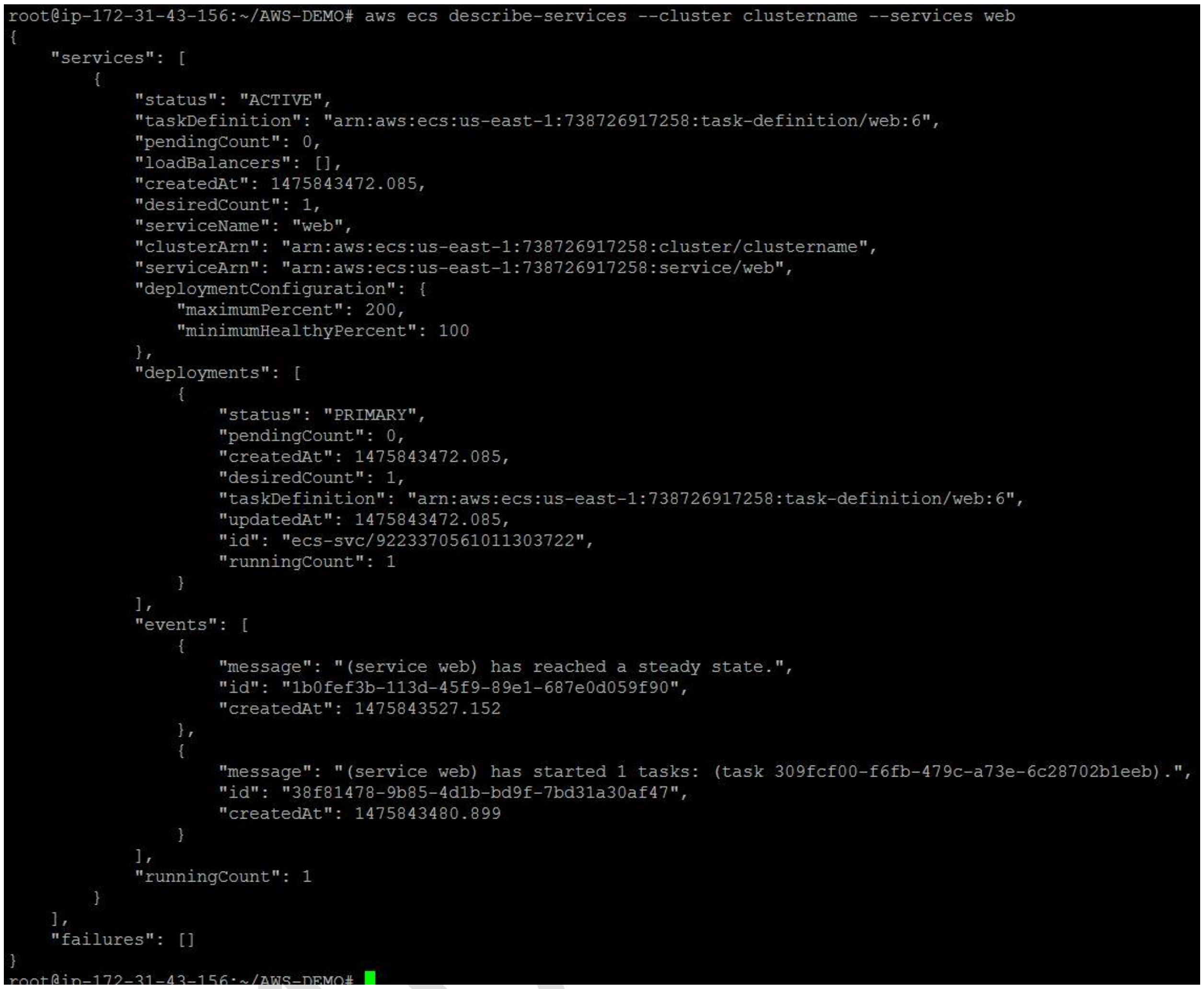
Output:



```
root@ip-172-31-43-156:~/AWS-DEMO# aws ecs list-services --cluster clustername
{
    "serviceArns": [
        "arn:aws:ecs:us-east-1:738726917258:service/web"
    ]
}
root@ip-172-31-43-156:~/AWS-DEMO#
```

We can see there is one service called **web** which we registered in the previous steps. Similar to most of the commands we can view a deeper look by typing the command:

# aws ecs describe-services --cluster <clustername> --services web

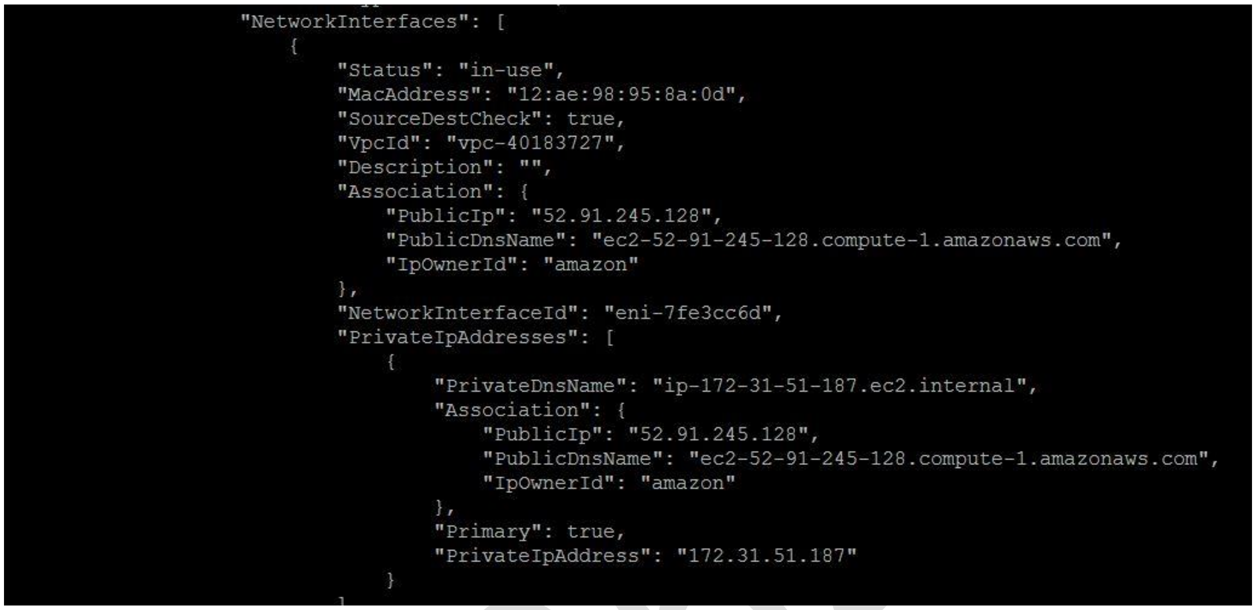

```
root@ip-172-31-43-156:~/AWS-DEMO# aws ecs describe-services --cluster clustername --services web
{
    "services": [
        {
            "status": "ACTIVE",
            "taskDefinition": "arn:aws:ecs:us-east-1:738726917258:task-definition/web:6",
            "pendingCount": 0,
            "loadBalancers": [],
            "createdAt": 1475843472.085,
            "desiredCount": 1,
            "serviceName": "web",
            "clusterArn": "arn:aws:ecs:us-east-1:738726917258:cluster/clustername",
            "serviceArn": "arn:aws:ecs:us-east-1:738726917258:service/web",
            "deploymentConfiguration": {
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            },
            "deployments": [
                {
                    "status": "PRIMARY",
                    "pendingCount": 0,
                    "createdAt": 1475843472.085,
                    "desiredCount": 1,
                    "taskDefinition": "arn:aws:ecs:us-east-1:738726917258:task-definition/web:6",
                    "updatedAt": 1475843472.085,
                    "id": "ecs-svc/9223370561011303722",
                    "runningCount": 1
                }
            ],
            "events": [
                {
                    "message": "(service web) has reached a steady state.",
                    "id": "1b0fef3b-113d-45f9-89e1-687e0d059f90",
                    "createdAt": 1475843527.152
                },
                {
                    "message": "(service web) has started 1 tasks: (task 309fcf00-f6fb-479c-a73e-6c28702b1eeb).",
                    "id": "38f81478-9b85-4d1b-bd9f-7bd31a30af47",
                    "createdAt": 1475843480.899
                }
            ],
            "runningCount": 1
        }
    ],
    "failures": []
}
root@ip-172-31-43-156:~/AWS-DEMO#
```

In the above output we can see something called **events**, which display every single thing related to the service will be output here. We can see that it reached steady state in the message and the running count is set to 1.

Now let's go and view this service in our web browser, but before that we need public DNS for our EC2 instance. It can be done by typing:

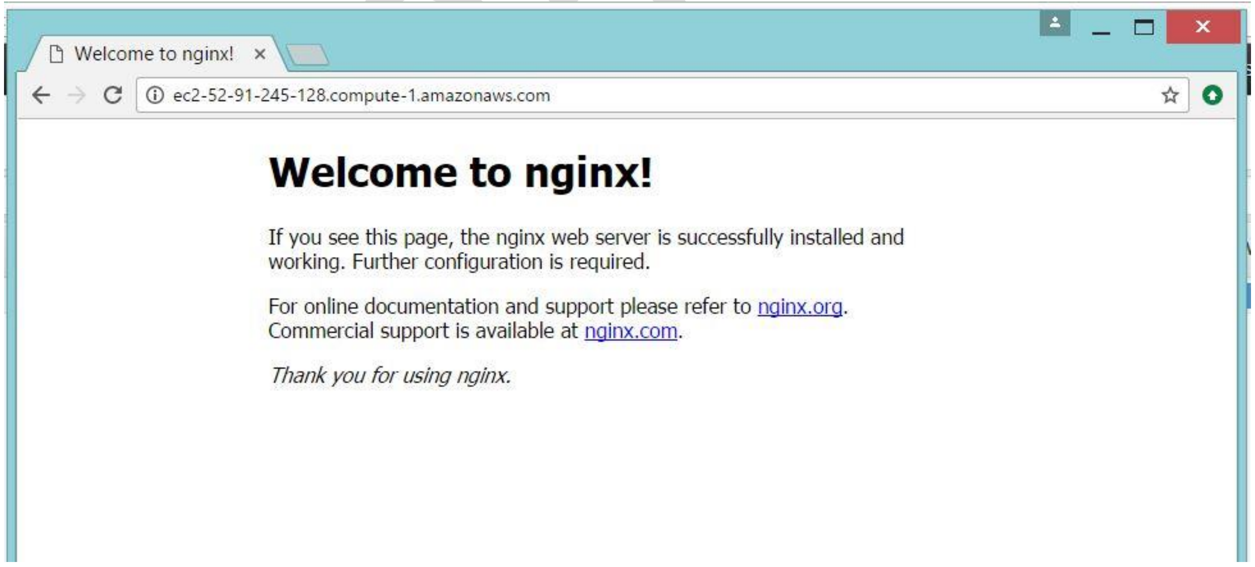
# aws ec2 describe-instances

The output will be lengthy, so I am just pasting the part in which we can find our EC2 instance public DNS. Now go back to resources.txt file and save it by adding new entry as public DNS and save it.



```
                    "NetworkInterfaces": [
                        {
                            "Status": "in-use",
                            "MacAddress": "12:ae:98:95:8a:0d",
                            "SourceDestCheck": true,
                            "VpcId": "vpc-40183727",
                            "Description": "",
                            "Association": {
                                "PublicIp": "52.91.245.128",
                                "PublicDnsName": "ec2-52-91-245-128.compute-1.amazonaws.com",
                                "IpOwnerId": "amazon"
                            },
                            "NetworkInterfaceId": "eni-7fe3cc6d",
                            "PrivateIpAddresses": [
                                {
                                    "PrivateDnsName": "ip-172-31-51-187.ec2.internal",
                                    "Association": {
                                        "PublicIp": "52.91.245.128",
                                        "PublicDnsName": "ec2-52-91-245-128.compute-1.amazonaws.com",
                                        "IpOwnerId": "amazon"
                                    },
                                    "Primary": true,
                                    "PrivateIpAddress": "172.31.51.187"
                                }
```

Now go to web browser and paste public DNS to get NGINX default page.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Sameer M
M1036298