DOCKER

Docker is a containerization platform that packages your application and all its dependencies together in the form of Containers to ensure that your application works seamlessly in any environment.

Benefits of Docker

Now, the QA team need not install all the dependent software and applications to test the code and this helps them save lots of time and energy. This also ensures that the working environment is consistent across all the individuals involved in the process, starting from development to deployment. The number of systems can be scaled up easily and the code can be deployed on them effortlessly.

CONTAINERIZATION

Containerization is the technique of bringing virtualization to the operating system level. While Virtualization brings abstraction to the hardware, Containerization brings abstraction to the operating system. Do note that Containerization is also a type of Virtualization. Containerization is however more efficient because there is no guest OS here and utilizes a host's operating system, share relevant libraries & resources as and when needed unlike virtual machines. Application specific binaries and libraries of containers run on the host kernel, which makes processing and execution very fast. Even booting-up a container takes only a fraction of a second. Because all the containers share, host operating system and holds only the application related binaries & libraries. They are lightweight and faster than Virtual Machines.

Advantages of Containerization over Virtualization:

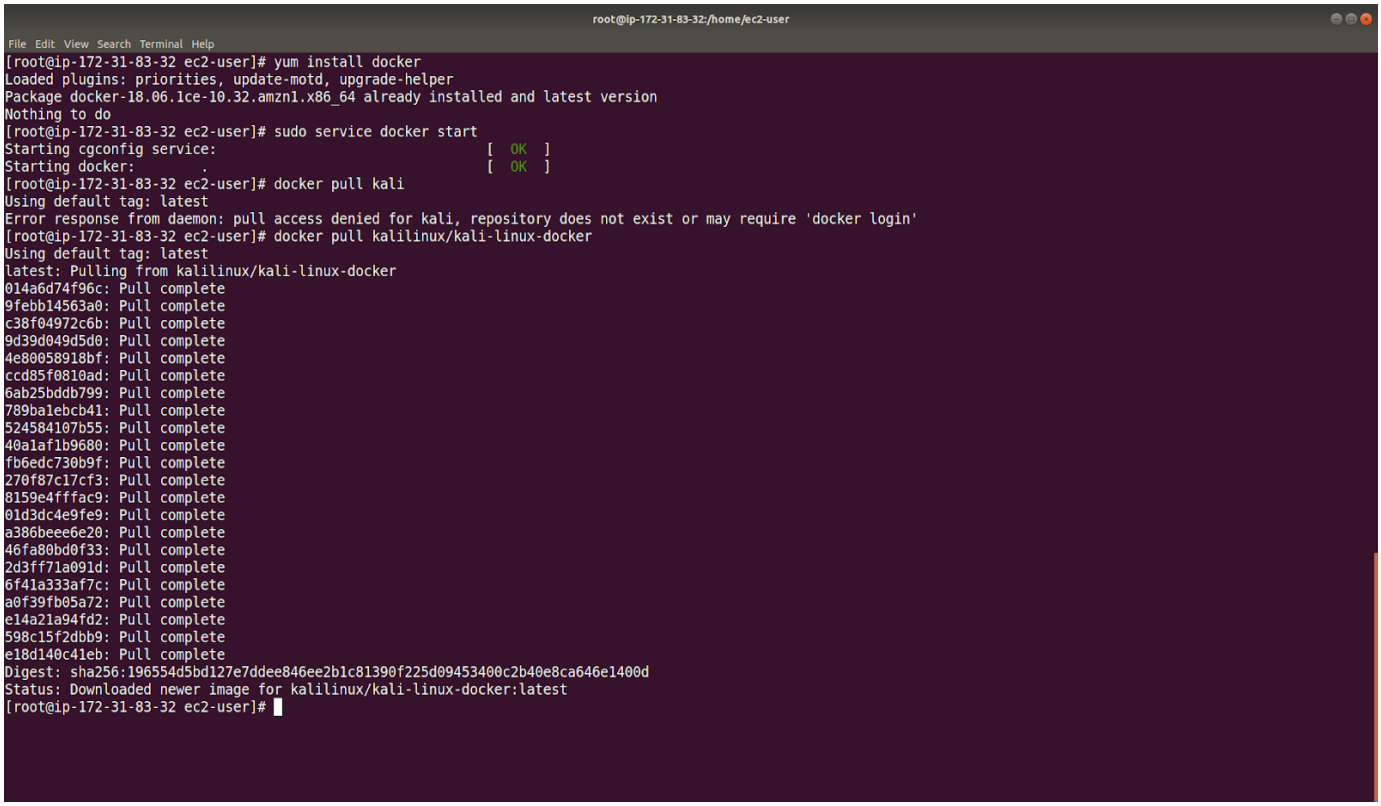
- Containers on the same OS kernel are lighter and smaller
- Better resource utilization compared to VMs
- Boot-up process is short and takes a few seconds

Installing Docker in EC2 Instance

> First, We need to launch an EC2 Instance

>Afterwards, Switch to root using sudo su

>Use Command yum install docker



>Use command **sudo service docker start** to check the status of the docker - engine

>Use command **docker pull <image name>** to pull the docker image.

>Use command **sudo docker run -it <image name>**

```
[root@ip-172-31-83-32 ec2-user]# sudo docker run -it kalilinux/kali-linux-docker
root@062ceb36c313:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

>Use command **docker ps** to see the running containers. To see the all containers use -a (or) -all flag

>Use Command **sudo docker exec -it <container id> bash** to run the docker image using the container id.

```
[root@ip-172-31-83-32 ec2-user]# sudo docker ps
CONTAINER ID        IMAGE                       COMMAND             CREATED             STATUS
37062750b317        kalilinux/kali-linux-docker   "bash"              2 hours ago         Up 2 hours
[root@ip-172-31-83-32 ec2-user]# sudo docker exec -it 37062750b317 bash
root@37062750b317:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
```

Docker Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

Compose works in all environments: production, staging, development, testing, as well as CI workflows.

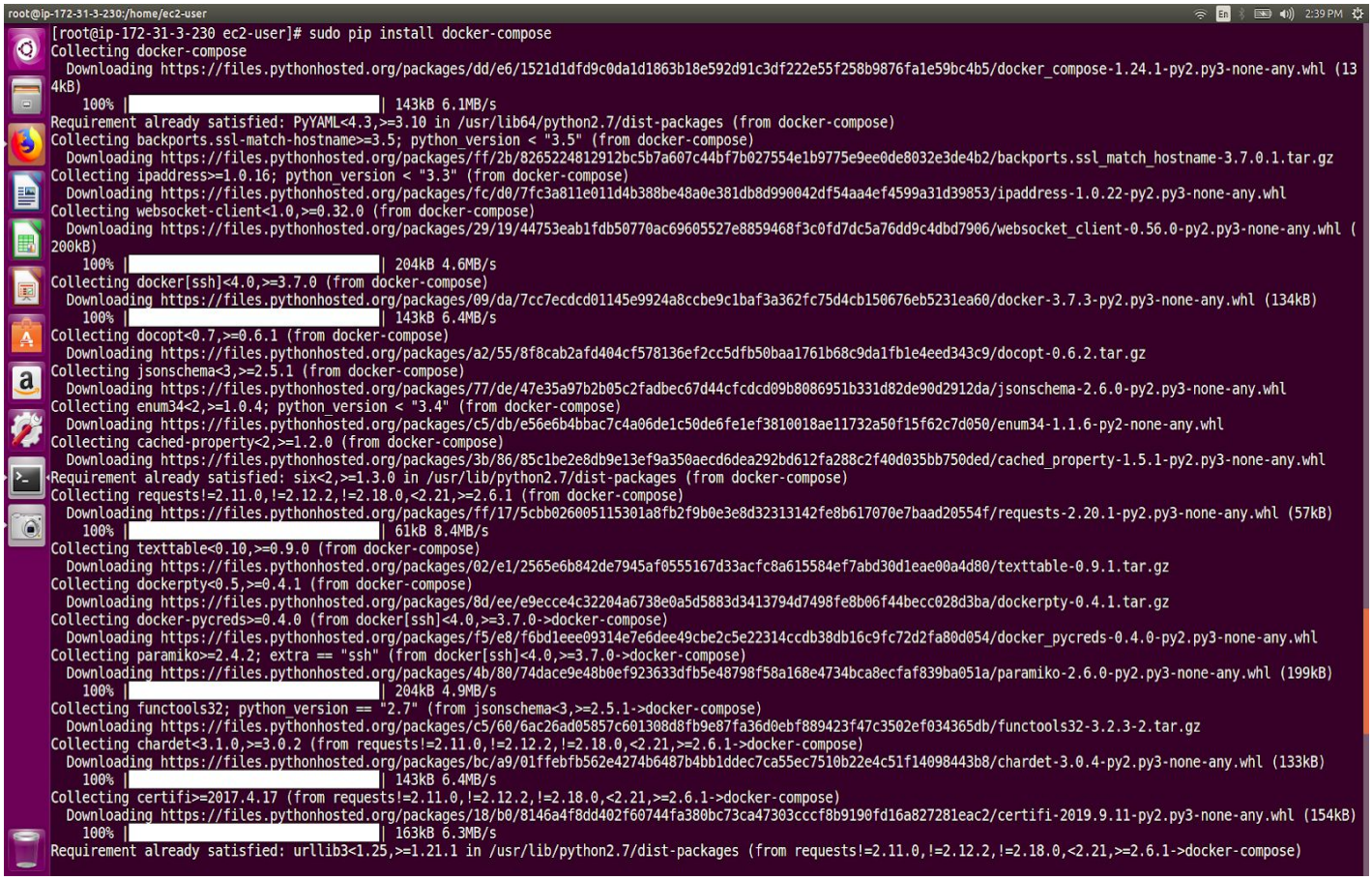
Using Compose is basically a three-step process:

1. Define your app's environment with a Dockerfile so it can be reproduced anywhere.

2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.

3. Run docker-compose up and Compose starts and runs your entire app.

>Install Docker-Compose



Wordpress and phpmyadmin using Docker Compose

>here, I created a file named docker-compose.yml in **Cinephilia** Directory

>here, In below yml code, First I defined a container Named **Wordpress** and it's built from an image **wordpress** that was present in the docker hub. But,this wordpress image does not have a database. For that, I defined another container named **wordpress_db** and it's actually built from an image **mariadb** that was present in the docker hub.

>Afterwards, I need to link my wordpress_db with my wordpress container for that I written **wordpress_db:mysql** under wordpress links section.

>In, Ports section I linked port 80 of docker container with port 5000 of my Instance.

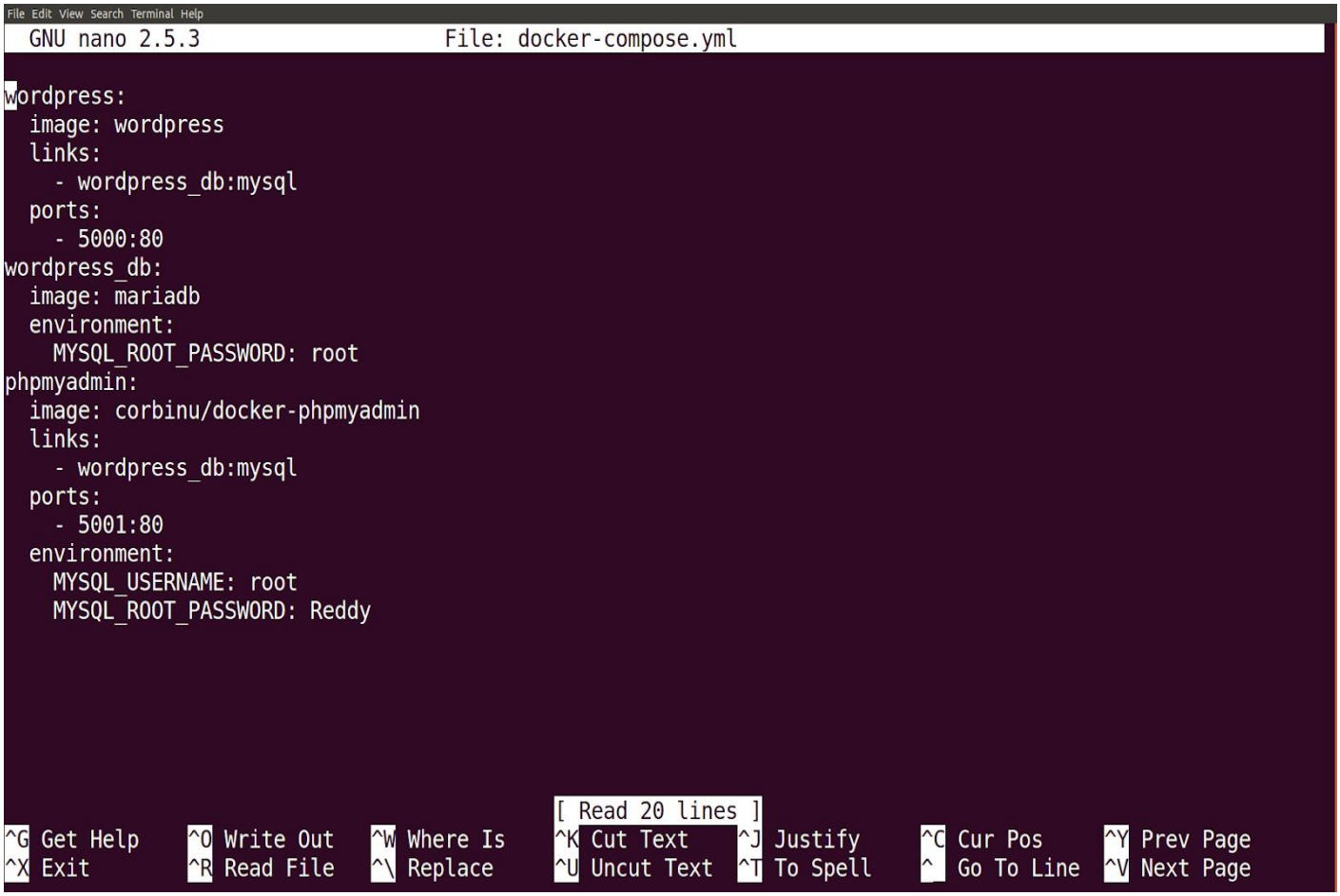
>After, I defined another container named **phpmyadmin** and it was built from an image

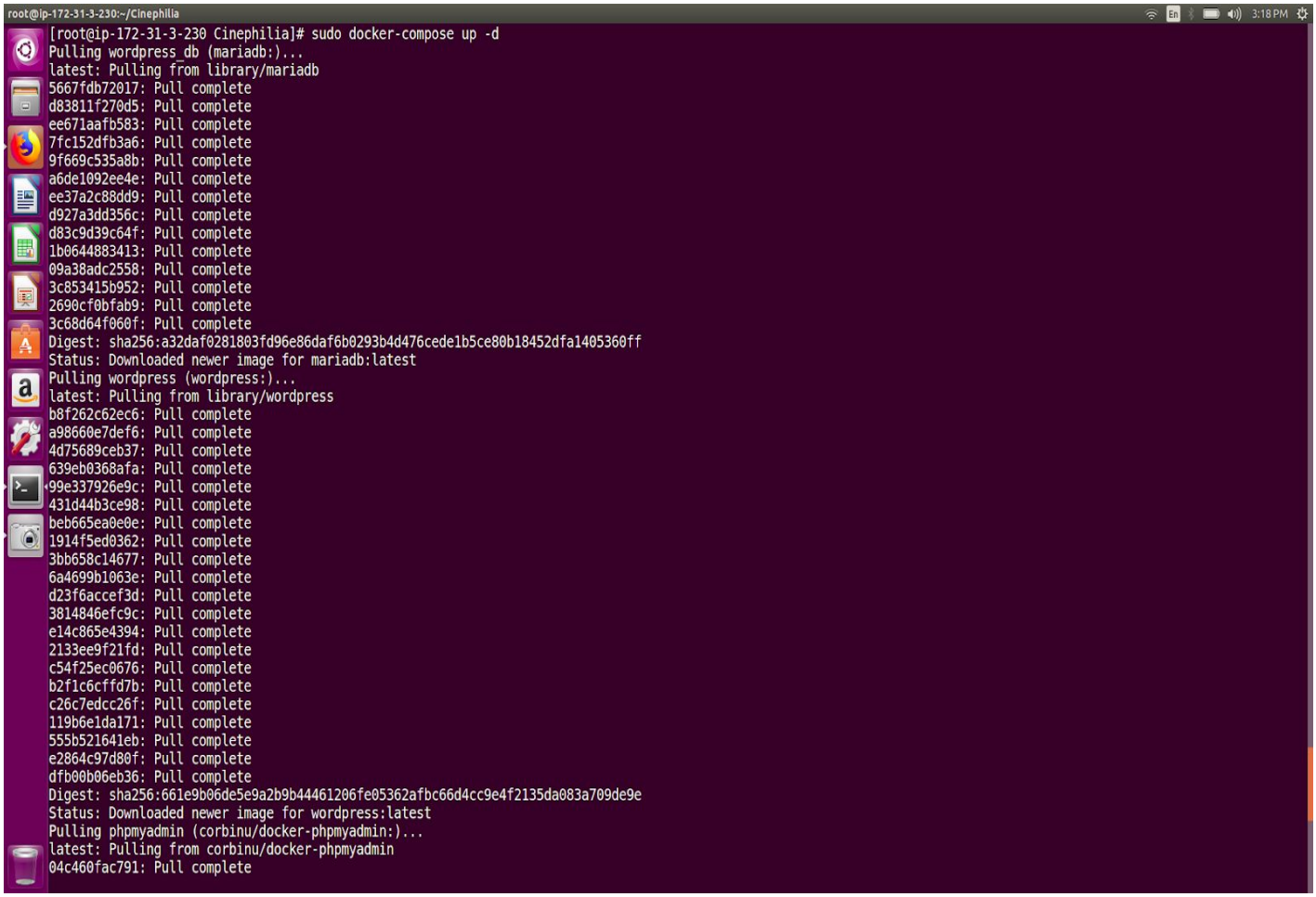
corbinu/docker-phpmyadmin that is present in the docker hub. I linked port 80 of this

Container **phpmyadmin** with port 5001 of my instance.

>Again, I need to link **phpmyadmin** container with my **wordpress_db** container. For that I written **wordpress_db:mysql** under links section of phpmyadmin container. Finally I given a user name as Root and password as Reddy.

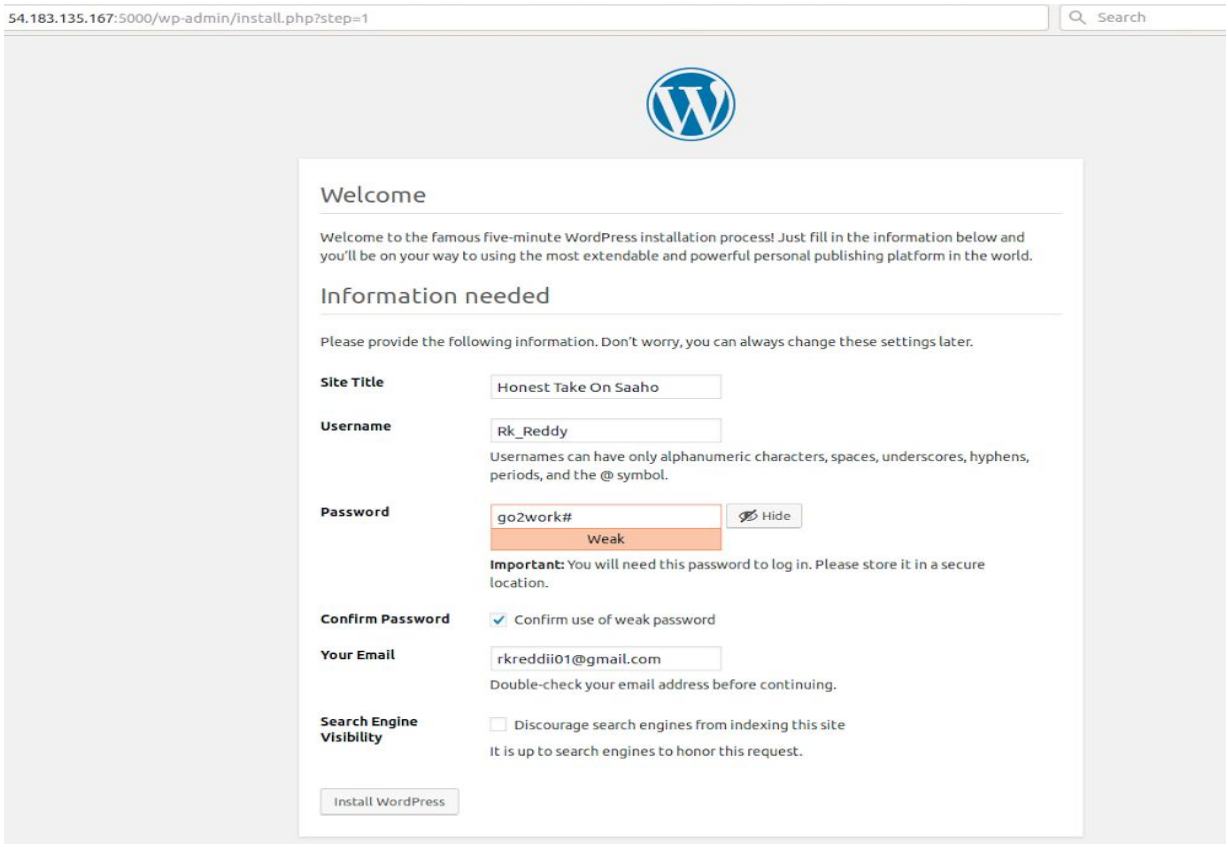
>here, we will type **sudo docker-compose -up -d.** This command will actually pull all the images from the docker hub and built all the containers.


File Edit View Search Terminal Help
GNU nano 2.5.3 File: docker-compose.yml
wordpress:
image: wordpress
links:
- wordpress_db:mysql
ports:
- 5000:80
wordpress_db:
image: mariadb
environment:
MYSQL_ROOT_PASSWORD: root
phpmyadmin:
image: corbinu/docker-phpmyadmin
links:
- wordpress_db:mysql
ports:
- 5001:80
environment:
MYSQL_USERNAME: root
MYSQL_ROOT_PASSWORD: Reddy
[Read 20 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^Y Prev Page
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line ^V Next Page



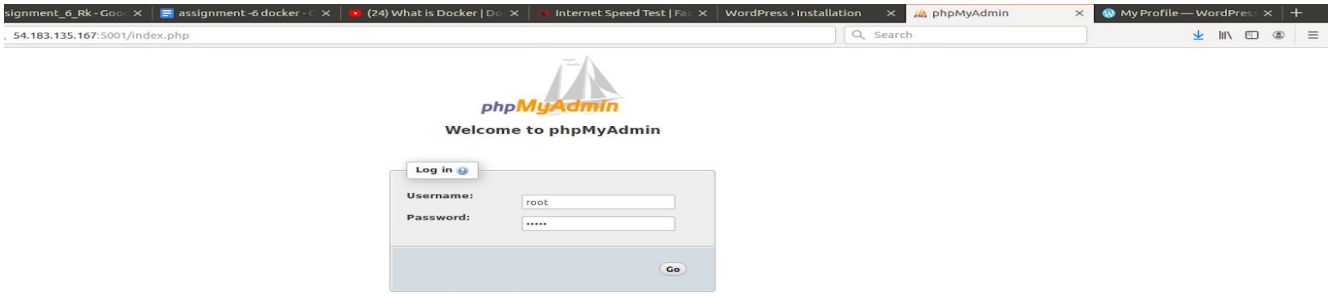

```
[root@ip-172-31-3-230 Cinephilia]# nano docker-compose.yml
[root@ip-172-31-3-230 Cinephilia]# sudo docker-compose up -d
Recreating cinephilia_wordpress_db_1 ... done
cinephilia_phpmyadmin_1 is up-to-date
Recreating cinephilia_wordpress_1    ... done
[root@ip-172-31-3-230 Cinephilia]#
```

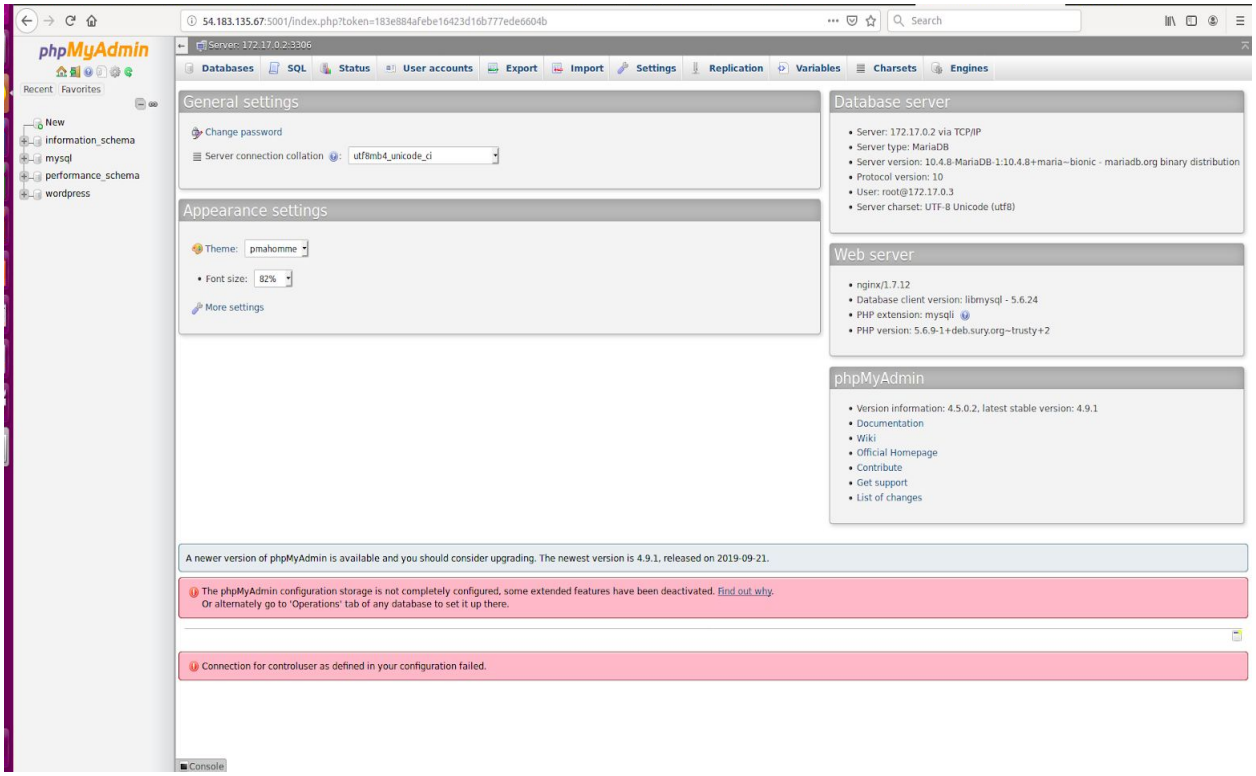
54.183.135.167:5000 opens up wordpress installation page



Enter the credentials those were given in the yml code snippet

54.183.135.167:5001 opens up phpmyadmin page





docker-compose.yml code snippet

```
wordpress:
  image: wordpress
  links:
        - wordpress_db:mysql
  ports:
        - 5000:80
wordpress_db:
  image: mariadb
  environment:
        MYSQL_ROOT_PASSWORD: Reddy
phpmyadmin:
  image : corbinu/docker-phpmyadmin
  links:
        - wordpress_db:mysql
  ports:
        - 5001:80
  environment:
        MYSQL_USERNAME:  root
        MYSQL_ROOT_PASSWORD:  Reddy
```

Amazon Ecs

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, high-performance container orchestration service that supports Docker containers and allows you to easily run and scale containerized applications on AWS. Amazon ECS eliminates the need for you to install and operate your own container orchestration software, manage and scale a cluster of virtual machines, or schedule containers on those virtual machines.
With simple API calls, you can launch and stop Docker-enabled applications, query the complete state of your application, and access many familiar features such as IAM roles, security groups, load balancers, Amazon CloudWatch Events, AWS CloudFormation templates, and AWS CloudTrail logs

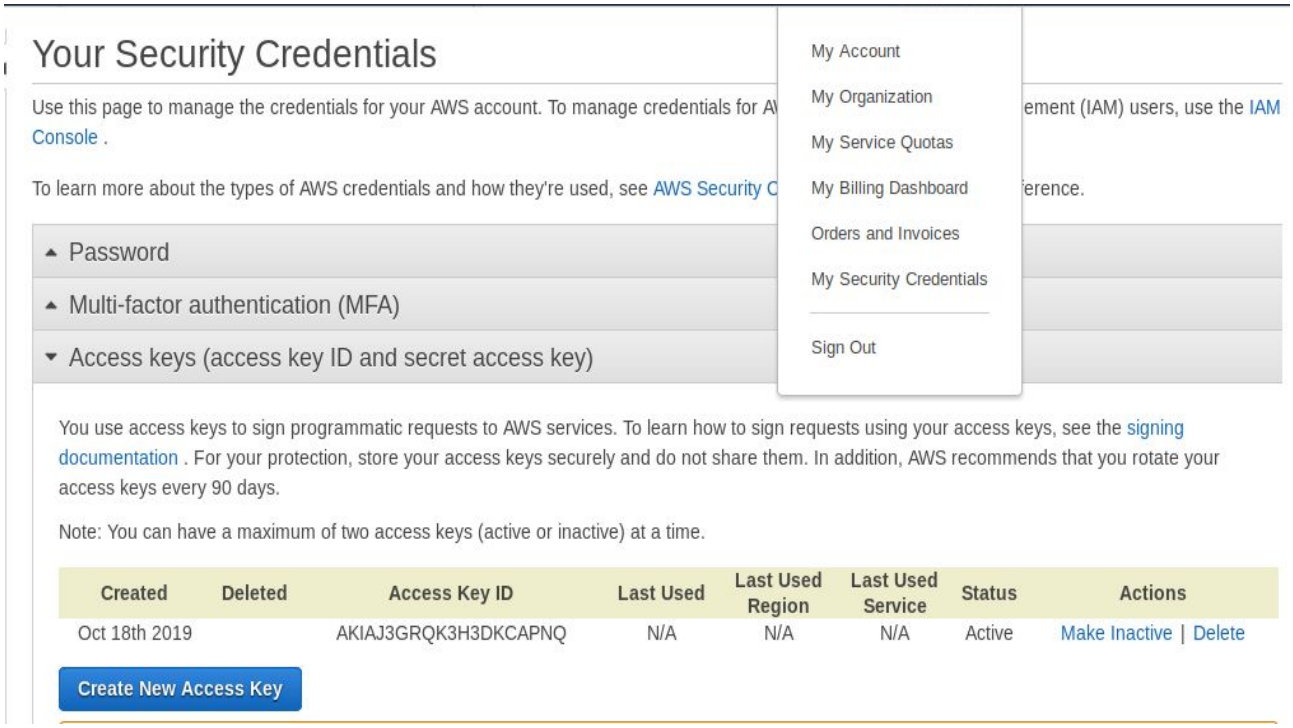
AWS FARGATE

AWS Fargate is a compute engine for Amazon ECS that allows you to run containers without having to manage servers or clusters. With AWS Fargate, you no longer have to provision, configure, and scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing. AWS Fargate removes the need for you to interact with or think about servers or clusters. Fargate lets you focus on designing and building your applications instead of managing the infrastructure that runs them.

Amazon ECS has two modes: Fargate launch type and EC2 launch type. With Fargate launch type, all you have to do is package your application in containers, specify the CPU and memory requirements,

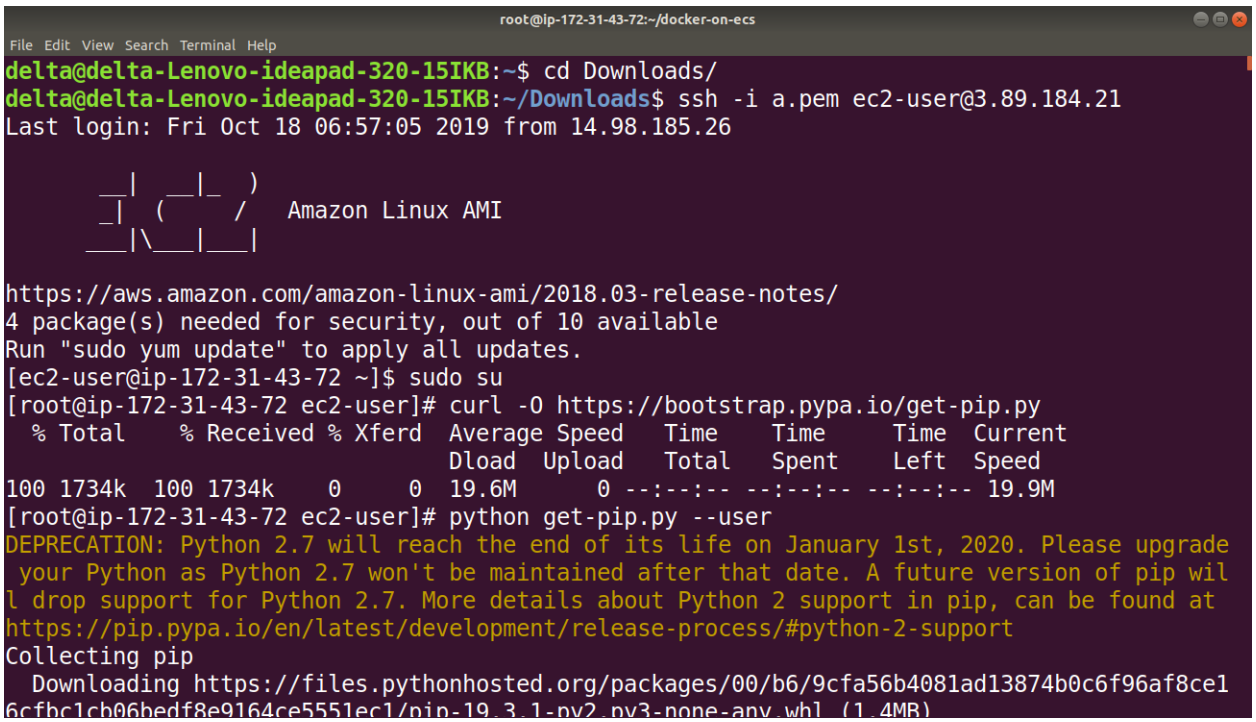
define networking and IAM policies, and launch the application. EC2 launch type allows you to have server-level, more granular control over the infrastructure that runs your container applications. With EC2 launch type, you can use Amazon ECS to manage a cluster of servers and schedule placement of containers on the servers. Amazon ECS keeps track of all the CPU, memory and other resources in your cluster, and also finds the best server for a container to run on based on your specified resource requirements. You are responsible for provisioning, patching, and scaling clusters of servers. You can decide which type of server to use, which applications and how many containers to run in a cluster to optimize utilization, and when you should add or remove servers from a cluster. EC2 launch type gives you more control of your server clusters and provides a broader range of customization options, which might be required to support some specific applications or possible compliance and government requirements.

>Go to My Security Credentials to create and download Access Key. Later, That will be helpful in configuring Awscli.



>In order to install awscli first, we need to install python pip by using commands

>**curl -0** https://bootstrap.pypa.io/get-pip.py and **python get-pip.py --user**



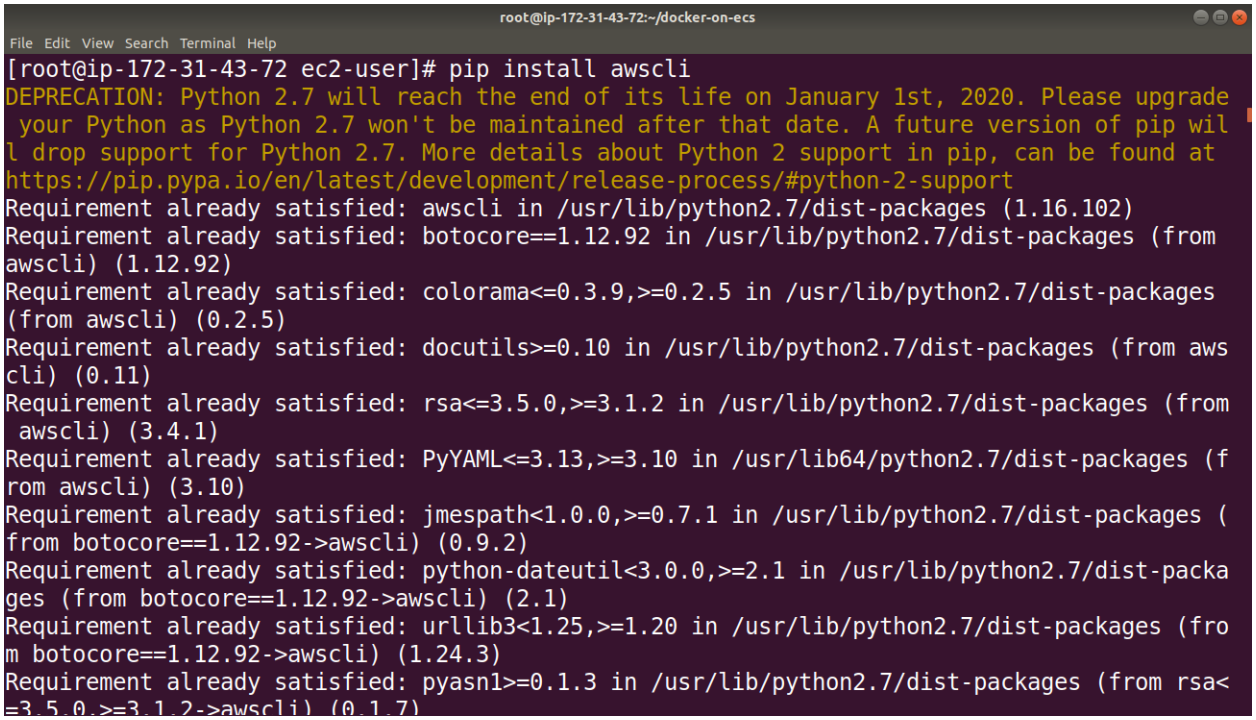
>Install awscli by command **pip install awscli**

>after installation configure your awscli by using command **aws configure**

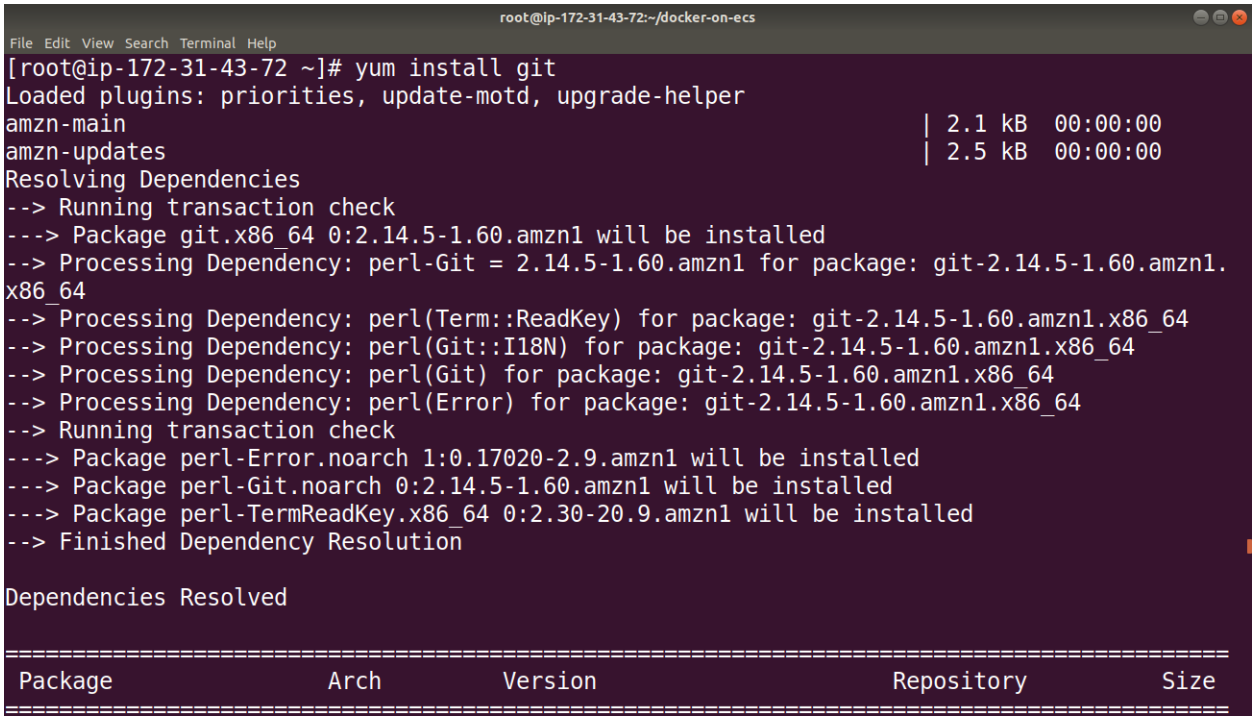
>you need to have your Access Key and Secret Access Key, if you don't have one follow the below steps otherwise skip that part.

>Create Access Key Login to AWS Console Under your IAM name choose 'My Security Credentials' My Security Credentials Click on Access Keys, create a new one like I've shown in the first screenshot and save the Secret Access Key as you no longer going to have access to it anymore.

>Create Access Key Go to your EC2 again and run 'aws configure' Put both Access and Secret Key Choose your region or leave it empty for default



>Install git for cloning and other activities of repositories.



>here,We are going to perform creating and running containerized application on Amazon Elastic Container Service (ECS) with AWS Fargate using AWS CloudFormation and AWS CLI.

Following are the steps to walk through,

1. Containerize a simple REST API application - Build Docker Image

2. Using AWS CLI, create repository on Amazon Elastic Container Registry (Docker Registry)

3. Push the docker image to repository on Amazon ECR

4. Define and run CloudFormation stack to create VPC and all network components

5. Define and run CloudFormation stack to create IAM roles

6. Define and run CloudFormation stack to create ECS Cluster, Load balancer, CloudWatch Log Group and Security Groups

7. Define and run CloudFormation stack to deploy docker application - Create Task for container, Service to schedule the task, Load balancer target group for container services and Load balancer listener rule for container service to run docker container on ECS with Fargate.

>First, Initialize a Git repository in root.then clone the repository using command

git clone https://github.com/devteds/e9-cloudformation-docker-ecs.git docker-on-ecs

>this repository contains the yaml files those will be helpful for creating stacks on cloudformation and creating the simple docker application.

Dockerize a simple app

Run on local

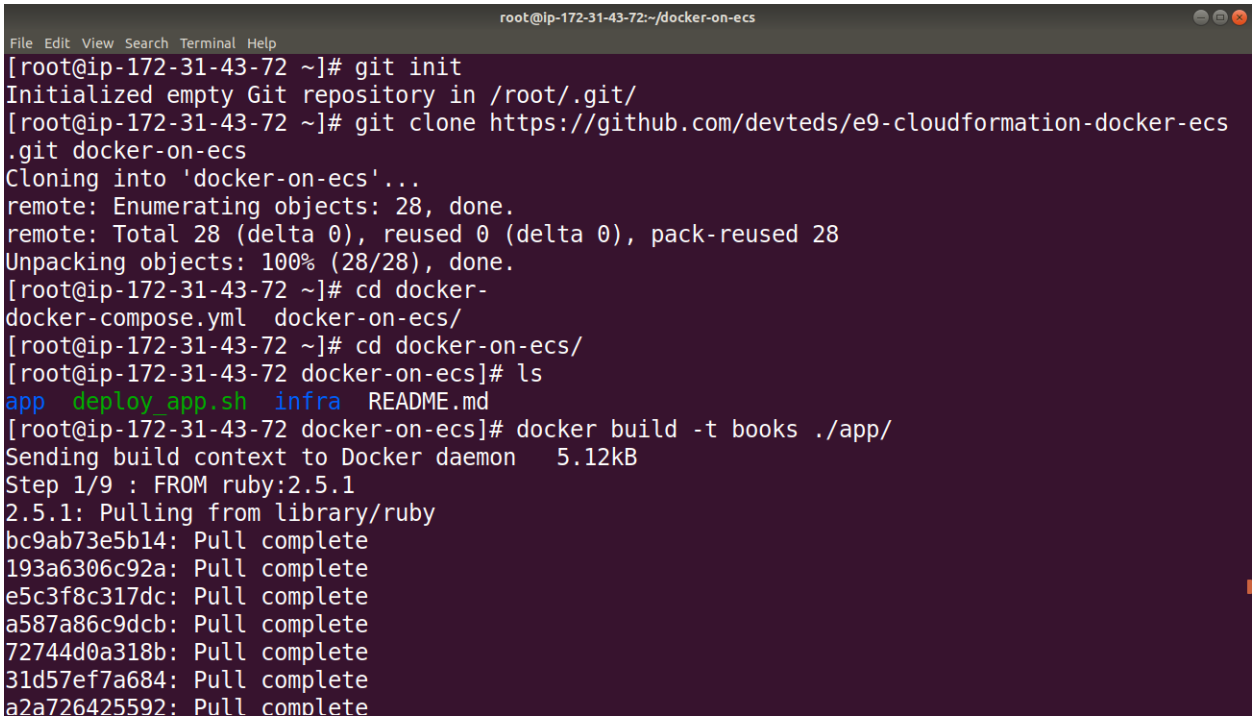
docker build -t books ./app/

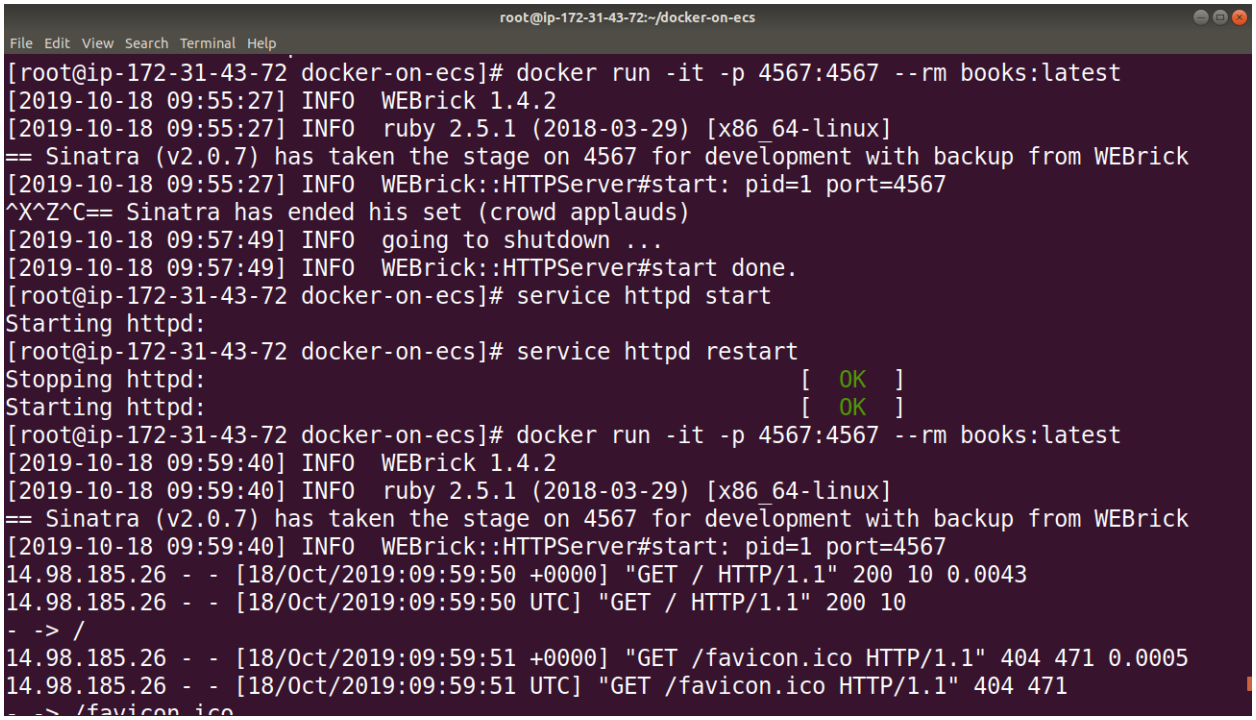
docker run -it -p 4567:4567 --rm books:latest

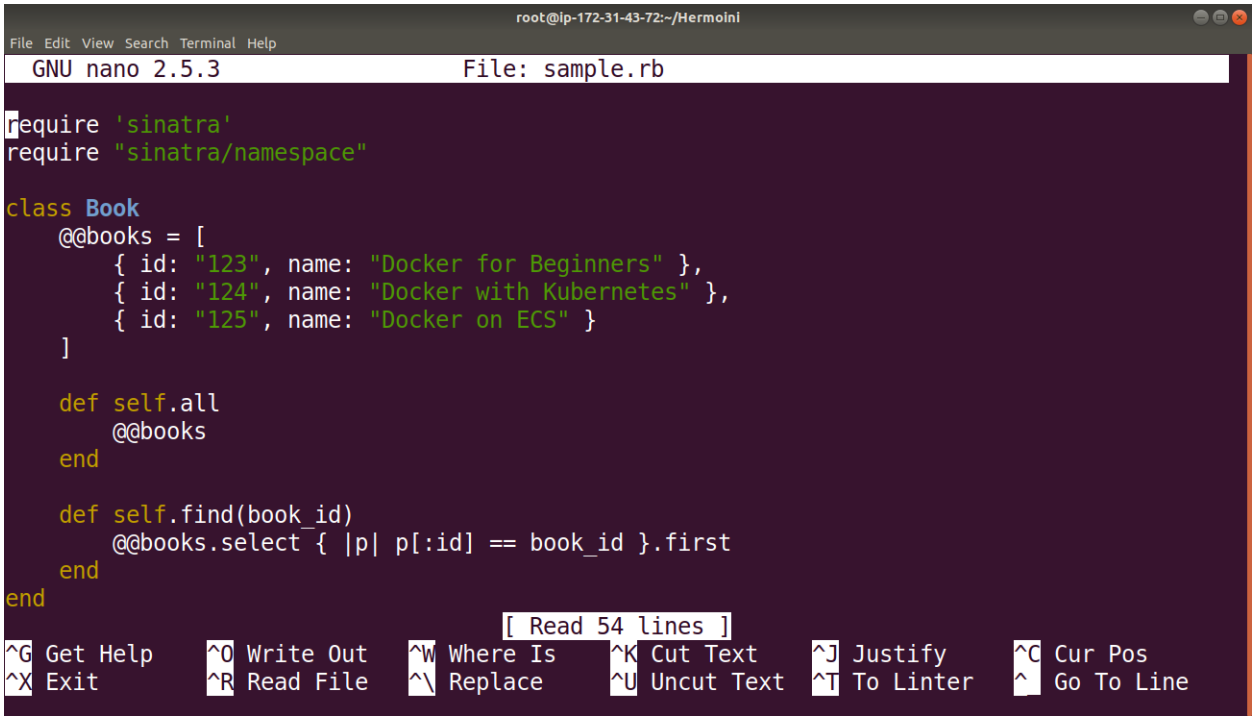
open http://localhost:4567/

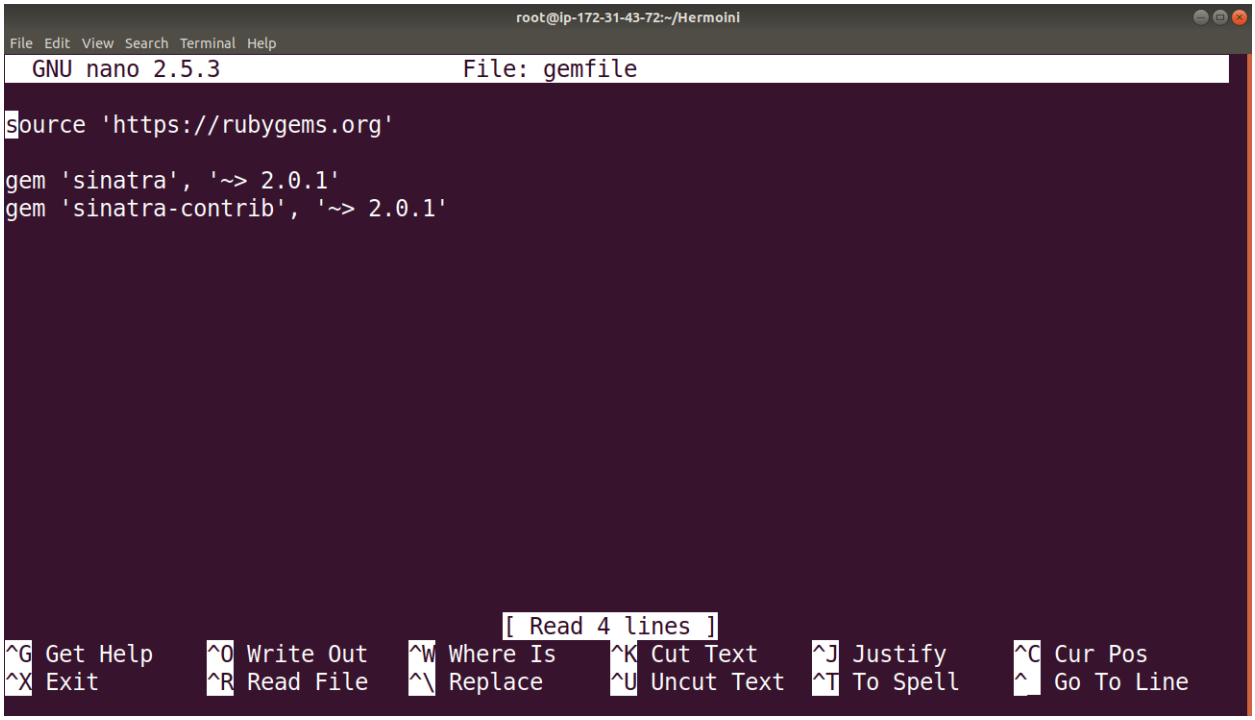
open http://localhost:4567/stat

open http://localhost:4567/api/books


root@ip-172-31-43-72:~/docker-on-ecs
File Edit View Search Terminal Help
[root@ip-172-31-43-72 ~]# git init
Initialized empty Git repository in /root/.git/
[root@ip-172-31-43-72 ~]# git clone https://github.com/devteds/e9-cloudformation-docker-ecs
.git docker-on-ecs
Cloning into 'docker-on-ecs'...
remote: Enumerating objects: 28, done.
remote: Total 28 (delta 0), reused 0 (delta 0), pack-reused 28
Unpacking objects: 100% (28/28), done.
[root@ip-172-31-43-72 ~]# cd docker-
docker-compose.yml docker-on-ecs/
[root@ip-172-31-43-72 ~]# cd docker-on-ecs/
[root@ip-172-31-43-72 docker-on-ecs]# ls
app deploy_app.sh infra README.md
[root@ip-172-31-43-72 docker-on-ecs]# docker build -t books ./app/
Sending build context to Docker daemon   5.12kB
Step 1/9 : FROM ruby:2.5.1
2.5.1: Pulling from library/ruby
bc9ab73e5b14: Pull complete
193a6306c92a: Pull complete
e5c3f8c317dc: Pull complete
a587a86c9dcb: Pull complete
72744d0a318b: Pull complete
31d57ef7a684: Pull complete
a2a726425592: Pull complete


root@ip-172-31-43-72:~/docker-on-ecs
File Edit View Search Terminal Help
[root@ip-172-31-43-72 docker-on-ecs]# docker run -it -p 4567:4567 --rm books:latest
[2019-10-18 09:55:27] INFO WEBrick 1.4.2
[2019-10-18 09:55:27] INFO ruby 2.5.1 (2018-03-29) [x86_64-linux]
== Sinatra (v2.0.7) has taken the stage on 4567 for development with backup from WEBrick
[2019-10-18 09:55:27] INFO WEBrick::HTTPServer#start: pid=1 port=4567
^X^Z^C== Sinatra has ended his set (crowd applauds)
[2019-10-18 09:57:49] INFO going to shutdown ...
[2019-10-18 09:57:49] INFO WEBrick::HTTPServer#start done.
[root@ip-172-31-43-72 docker-on-ecs]# service httpd start
Starting httpd:
[root@ip-172-31-43-72 docker-on-ecs]# service httpd restart
Stopping httpd: [OK]
Starting httpd: [OK]
[root@ip-172-31-43-72 docker-on-ecs]# docker run -it -p 4567:4567 --rm books:latest
[2019-10-18 09:59:40] INFO WEBrick 1.4.2
[2019-10-18 09:59:40] INFO ruby 2.5.1 (2018-03-29) [x86_64-linux]
== Sinatra (v2.0.7) has taken the stage on 4567 for development with backup from WEBrick
[2019-10-18 09:59:40] INFO WEBrick::HTTPServer#start: pid=1 port=4567
14.98.185.26 - - [18/Oct/2019:09:59:50 +0000] "GET / HTTP/1.1" 200 10 0.0043
14.98.185.26 - - [18/Oct/2019:09:59:50 UTC] "GET / HTTP/1.1" 200 10
- -> /
14.98.185.26 - - [18/Oct/2019:09:59:51 +0000] "GET /favicon.ico HTTP/1.1" 404 471 0.0005
14.98.185.26 - - [18/Oct/2019:09:59:51 UTC] "GET /favicon.ico HTTP/1.1" 404 471


root@ip-172-31-43-72:~/Hermoini
File Edit View Search Terminal Help
GNU nano 2.5.3 File: sample.rb
require 'sinatra'
require "sinatra/namespace"
class Book
@@books = [
{ id: "123", name: "Docker for Beginners" },
{ id: "124", name: "Docker with Kubernetes" },
{ id: "125", name: "Docker on ECS" }
]
def self.all
@@books
end
def self.find(book_id)
@@books.select { |p| p[:id] == book_id }.first
end
end
[Read 54 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line


root@ip-172-31-43-72:~/Hermoini
File Edit View Search Terminal Help
GNU nano 2.5.3 File: gemfile
source 'https://rubygems.org'
gem 'sinatra', '~> 2.0.1'
gem 'sinatra-contrib', '~> 2.0.1'
[Read 4 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line

root@ip-172-31-43-72:~/Hermoini

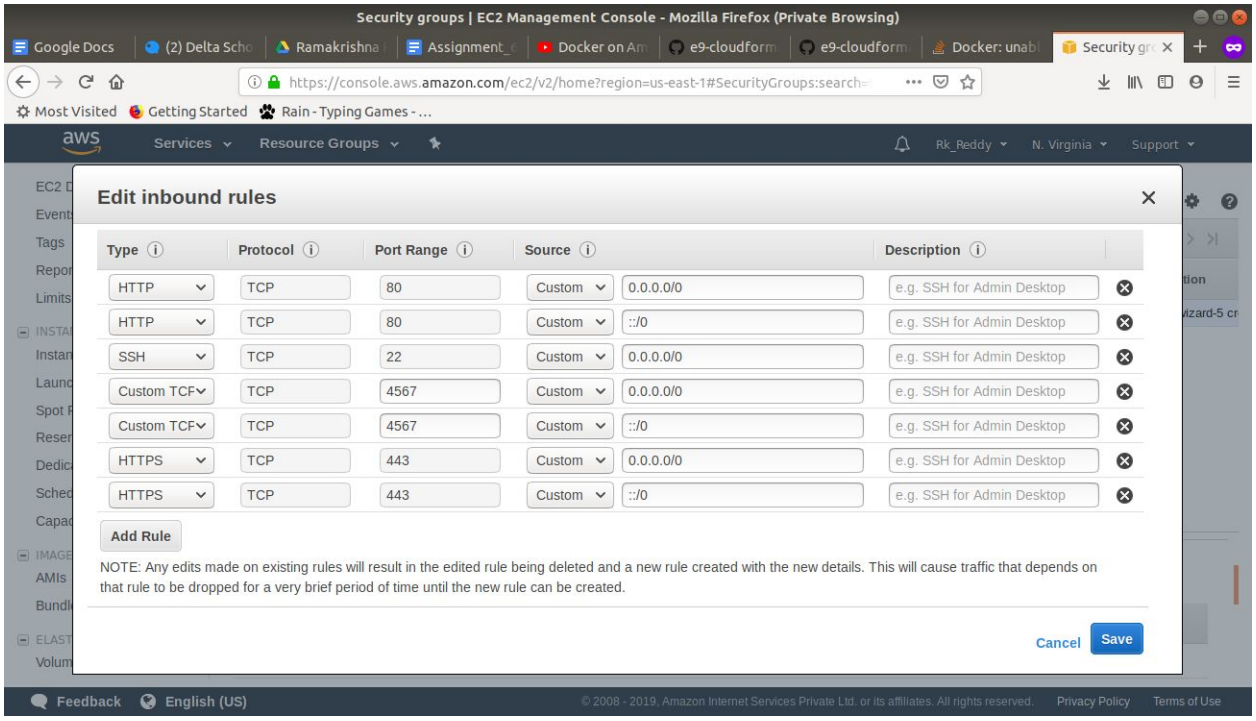
File Edit View Search Terminal Help

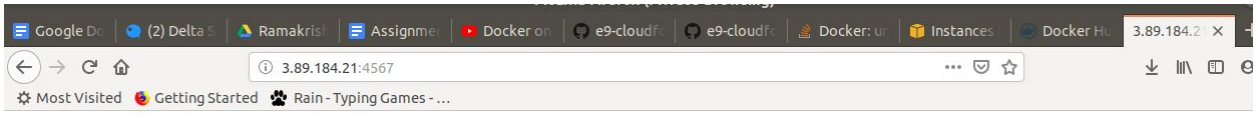
GNU nano 2.5.3 File: docker

```
FROM ruby:2.5.1

RUN apt-get update -qq && apt-get install -y build-essential libpq-dev

RUN mkdir /sapp
WORKDIR /sapp

ADD Gemfile /sapp/Gemfile
ADD Gemfile.lock /sapp/Gemfile.lock

RUN bundle install

ADD . /sapp

CMD ["ruby", "service.rb", "-o", "0.0.0.0"]
```

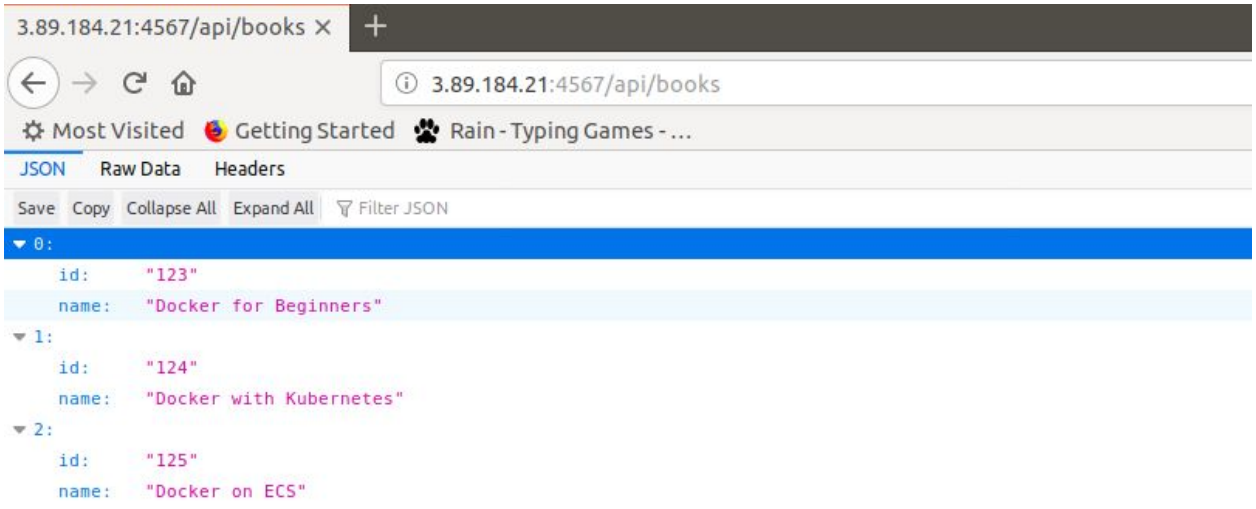
[Read 16 lines]

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line




3.89.184.21:4567
Most Visited
Getting Started
Rain - Typing Games - ...

Healthy!!!


3.89.184.21:4567/api/books
3.89.184.21:4567/api/books
Most Visited
Getting Started
Rain - Typing Games - ...
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
0:
id: "123"
name: "Docker for Beginners"
1:
id: "124"
name: "Docker with Kubernetes"
2:
id: "125"
name: "Docker on ECS"

Push Docker Image to ECR

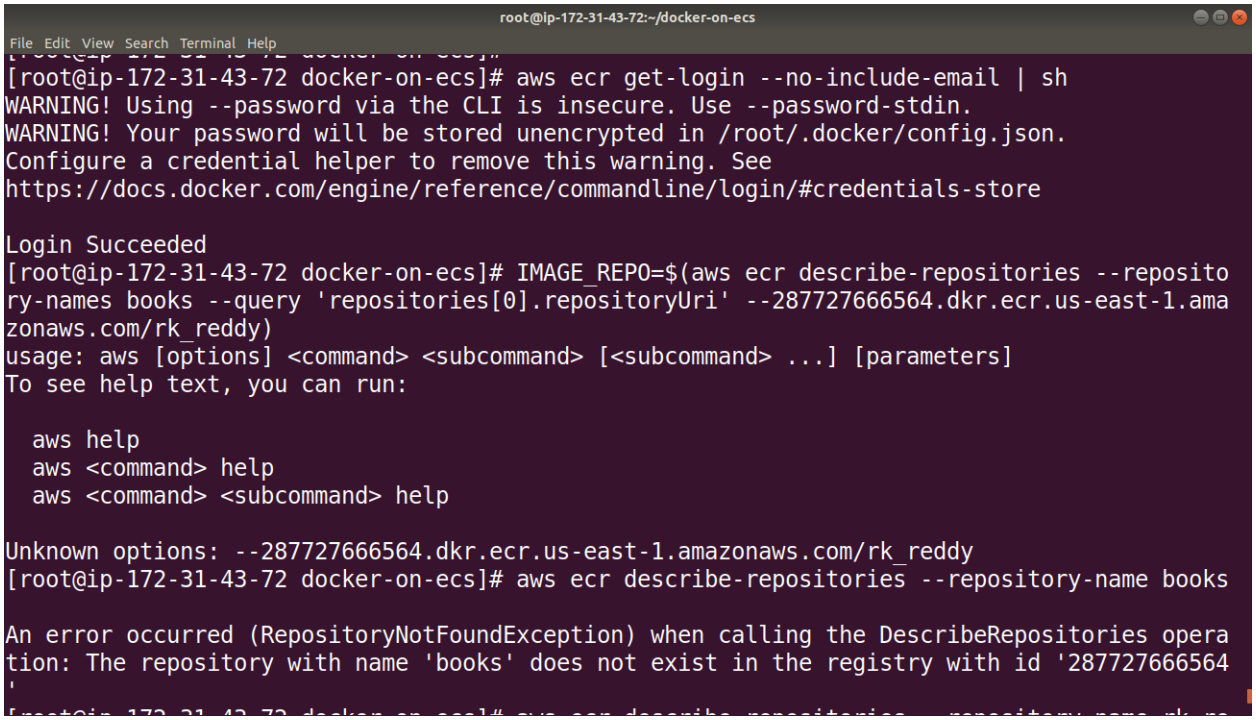
aws ecr create-repository --repository-name rk_reddy

aws ecr get-login --no-include-email | sh

IMAGE_REPO=$(aws ecr describe-repositories --repository-name rk_reddy --query 'repositories[0].repositoryUri' --output text)

docker tag books-api:latest $IMAGE_REPO:v1

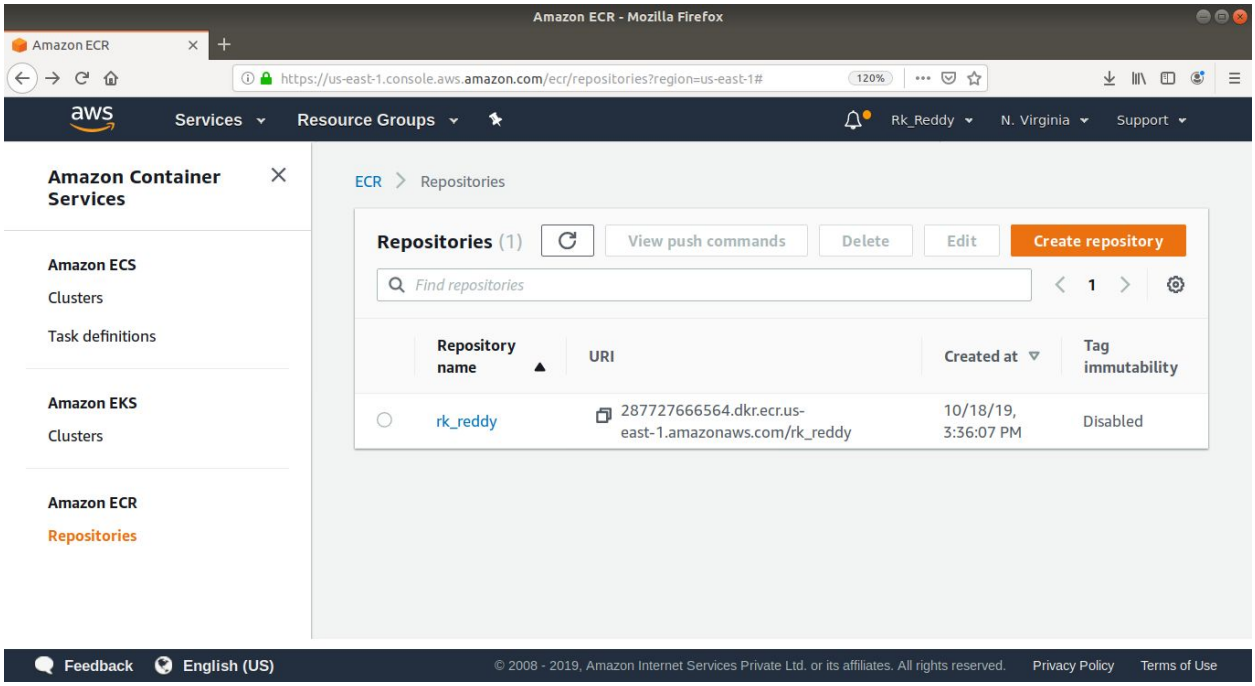
docker push $IMAGE_REPO:v1

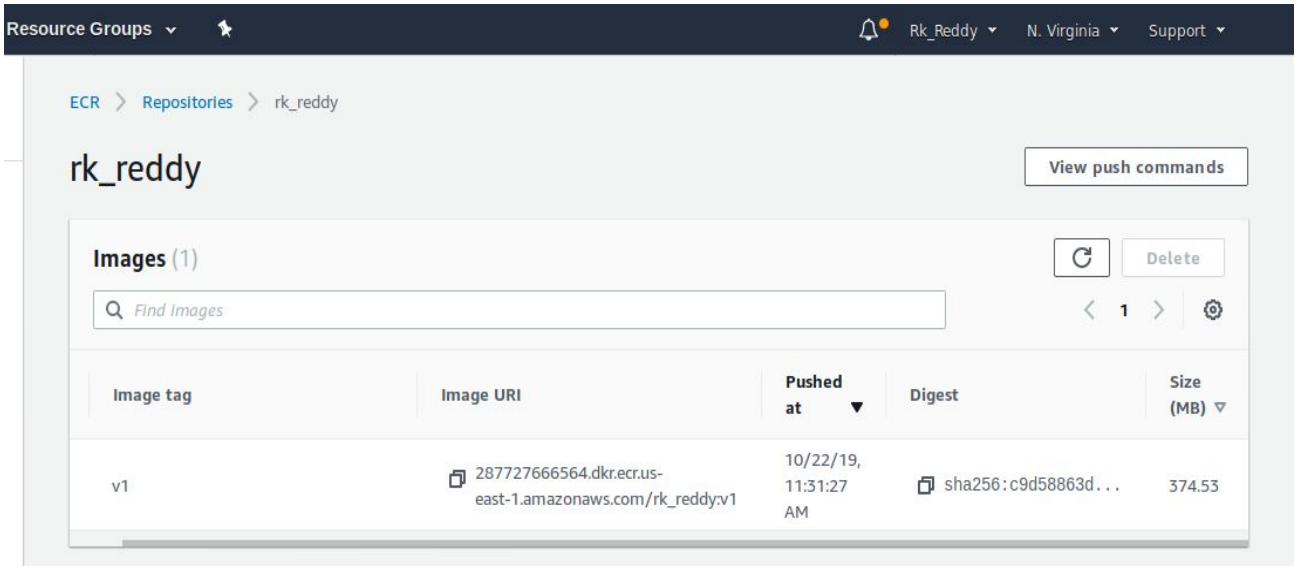


```
root@ip-172-31-43-72:~/docker-on-ecs
File  Edit  View  Search  Terminal  Help
[root@ip-172-31-43-72 docker-on-ecs]# aws ecr create-repository --repository-name RkReddy

An error occurred (InvalidParameterException) when calling the CreateRepository operation:
Invalid parameter at 'repositoryName' failed to satisfy constraint: 'must satisfy regular e
xpression '(?:[a-z0-9]+(?:[._-][a-z0-9]+)*/)*[a-z0-9]+(?:[._-][a-z0-9]+)*''
[root@ip-172-31-43-72 docker-on-ecs]# aws ecr create-repository --repository-name Rk_Reddy

An error occurred (InvalidParameterException) when calling the CreateRepository operation:
Invalid parameter at 'repositoryName' failed to satisfy constraint: 'must satisfy regular e
xpression '(?:[a-z0-9]+(?:[._-][a-z0-9]+)*/)*[a-z0-9]+(?:[._-][a-z0-9]+)*''
[root@ip-172-31-43-72 docker-on-ecs]# aws ecr create-repository --repository-name rk_reddy
{
    "repository": {
        "registryId": "287727666564",
        "repositoryName": "rk_reddy",
        "repositoryArn": "arn:aws:ecr:us-east-1:287727666564:repository/rk_reddy",
        "createdAt": 1571393167.0,
        "repositoryUri": "287727666564.dkr.ecr.us-east-1.amazonaws.com/rk_reddy"
    }
}
[root@ip-172-31-43-72 docker-on-ecs]#
[root@ip-172-31-43-72 docker-on-ecs]#
[root@ip-172-31-43-72 docker-on-ecs]#
```

```
[root@ip-172-31-43-72 docker-on-ecs]# aws ecr describe-repositories --repository-name rk_re
ddy
{
    "repositories": [
        {
            "registryId": "287727666564",
            "repositoryName": "rk_reddy",
            "repositoryArn": "arn:aws:ecr:us-east-1:287727666564:repository/rk_reddy",
            "createdAt": 1571393167.0,
            "repositoryUri": "287727666564.dkr.ecr.us-east-1.amazonaws.com/rk_reddy"
        }
    ]
}
```

```
[root@ip-172-31-91-76 docker-on-ecs]# docker tag books 287727666564.dkr.ecr.us-east-1.amazonaws.com/rk_reddy:v1
[root@ip-172-31-91-76 docker-on-ecs]# docker push 287727666564.dkr.ecr.us-east-1.amazonaws.com/rk_reddy:v1
The push refers to repository [287727666564.dkr.ecr.us-east-1.amazonaws.com/rk_reddy]
ca879a8e1586: Pushed
bf6eead2fb04: Pushed
c0833ae30f4e: Pushed
bd6f83d12e0a: Pushed
46c1fe072307: Pushed
55de4a218619: Pushed
3696bfafb5ed: Pushed
dda5ada3fa53: Pushed
f5e615773459: Pushed
9978d084fd77: Pushed
1191b3f5862a: Pushed
08a01612ffca: Pushed
8bb25f9cdc41: Pushed
f715ed19c28b: Pushed
v1: digest: sha256:c9d58863d1461557d7279757eae7737a624b3a3eadf532e33529d66b344dd5cb size: 3255
[root@ip-172-31-91-76 docker-on-ecs]#
```

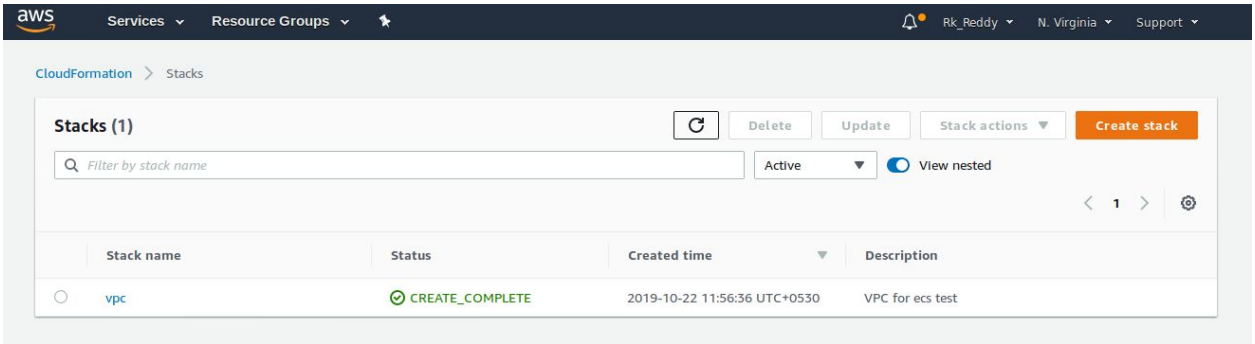




Creation of VPC stack using yaml scripting

aws cloudformation create-stack --template-body file://$PWD/infra/vpc.yml --stack-name vpc

```
[root@ip-172-31-91-76 docker-on-ecs]# aws cloudformation create-stack --template-body file://$PWD/infra/
vpc.yml --stack-name vpc

An error occurred (AlreadyExistsException) when calling the CreateStack operation: Stack [vpc] already e
xists
[root@ip-172-31-91-76 docker-on-ecs]#
```



vpc.yml

```
AWSTemplateFormatVersion: '2010-09-09'

Description: VPC for ecs test


Resources:


  VPC:

        Type: 'AWS::EC2::VPC'

        Properties:

        CidrBlock: '10.0.0.0/16'

        EnableDnsSupport: true

        EnableDnsHostnames: true

        InstanceTenancy: default


  Subnet1:
```

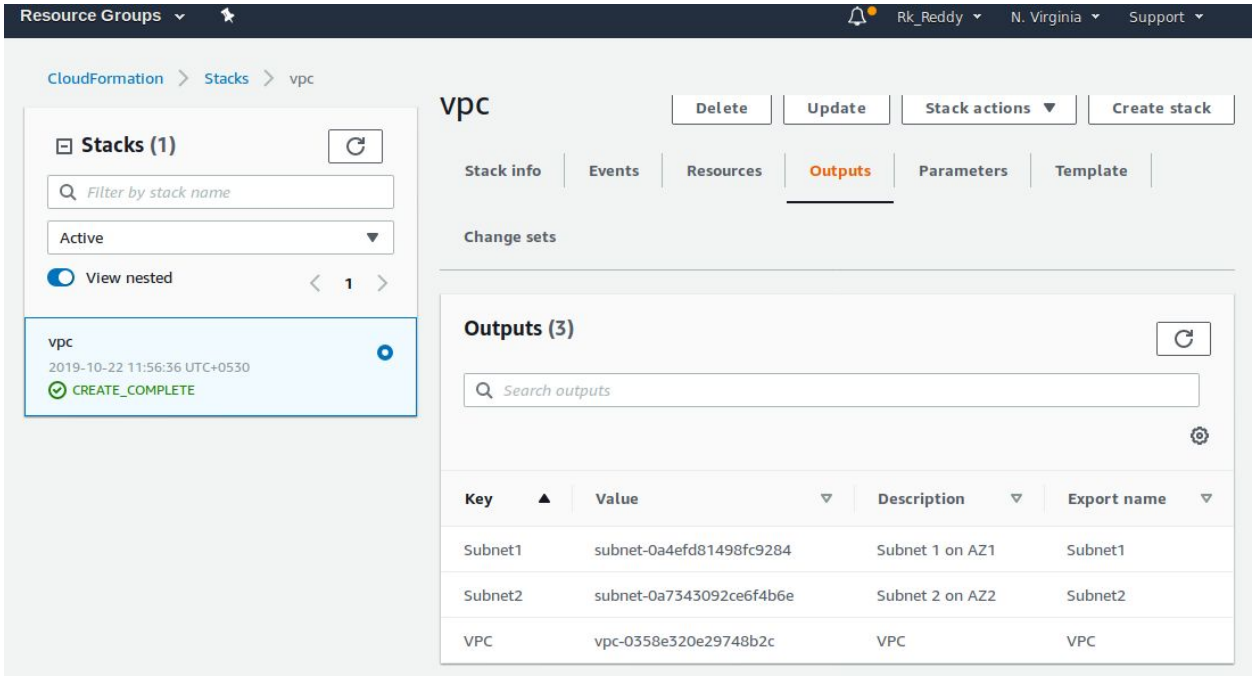
```
    Type: AWS::EC2::Subnet
    Properties:
    VpcId: !Ref VPC
    AavailabilityZone: !Select [0, !GetAZs '']
    CidrBlock: !Sub '10.0.0.0/20'
    MapPublicIpOnLaunch: true

Subnet2:
    Type: AWS::EC2::Subnet
    Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [1, !GetAZs '']
    CidrBlock: !Sub '10.0.32.0/20'
    MapPublicIpOnLaunch: true

InternetGateway:
    Type: 'AWS::EC2::InternetGateway'

VPCGatewayAttachment:
    Type: 'AWS::EC2::VPCGatewayAttachment'
    Properties:
    InternetGatewayId: !Ref InternetGateway
```

```
      VpcId: !Ref VPC

  RouteTable:
      Type: 'AWS::EC2::RouteTable'
      Properties:
      VpcId: !Ref VPC

  RouteTableAssociation1:
      Type: 'AWS::EC2::SubnetRouteTableAssociation'
      Properties:
      SubnetId: !Ref Subnet1
      RouteTableId: !Ref RouteTable

  RouteTableAssociation2:
      Type: 'AWS::EC2::SubnetRouteTableAssociation'
      Properties:
      SubnetId: !Ref Subnet2
      RouteTableId: !Ref RouteTable

  InternetRoute:
      Type: 'AWS::EC2::Route'
      DependsOn: VPCGatewayAttachment
```

```
    Properties:
    GatewayId: !Ref InternetGateway
    RouteTableId: !Ref RouteTable
    DestinationCidrBlock: '0.0.0.0/0'

Outputs:

  VPC:
    Description: VPC
    Value: !Ref VPC
    Export:
    Name: 'VPC'

  Subnet1:
    Description: 'Subnet 1 on AZ1'
    Value: !Ref Subnet1
    Export:
    Name: 'Subnet1'

  Subnet2:
    Description: 'Subnet 2 on AZ2'
    Value: !Ref Subnet2
```

Export:

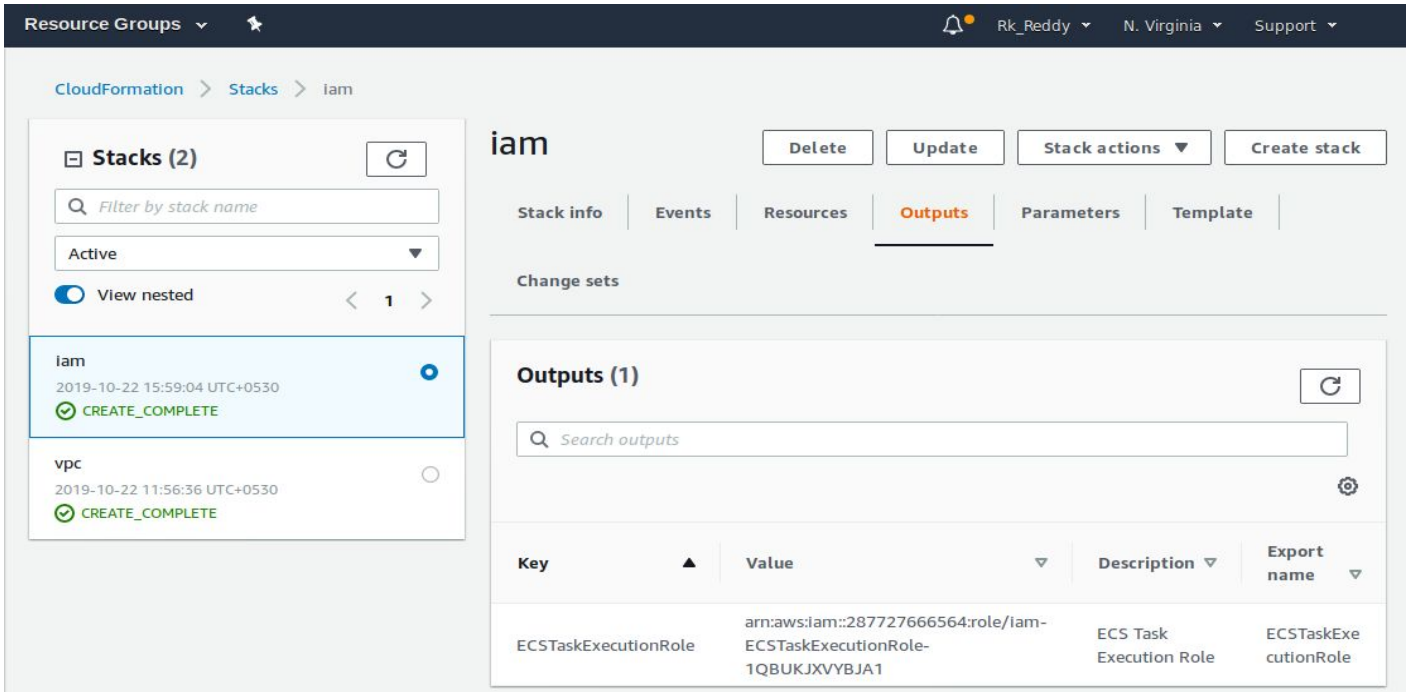
Name: 'Subnet2'



Creation of IAM stack using yml

aws cloudformation create-stack --template-body file://$PWD/infra/iam.yml --stack-name iam --capabilities CAPABILITY_IAM

```
[root@ip-172-31-91-76 docker-on-ecs]# aws cloudformation create-stack --template-body file://$PWD/infra/
iam.yml --stack-name iam --capabilities CAPABILITY_IAM
{
    "StackId": "arn:aws:cloudformation:us-east-1:287727666564:stack/iam/c5781fc0-f4b6-11e9-b9f0-0a933e79
26f6"
}
[root@ip-172-31-91-76 docker-on-ecs]#
```



iam.yml

```
AWSTemplateFormatVersion: '2010-09-09'

Description: roles and policies


Resources:


  ECSTaskExecutionRole:

        Type: AWS::IAM::Role

        Properties:

        AssumeRolePolicyDocument:

        Statement:
```

```
        - Effect: Allow
          Principal:
            Service: [ecs-tasks.amazonaws.com]
          Action: ['sts:AssumeRole']
      Path: /
      Policies:
        - PolicyName: AmazonECSTaskExecutionRolePolicy
          PolicyDocument:
            Statement:
            - Effect: Allow
              Action:
                  # ECS Tasks to download images from ECR
                  - 'ecr:GetAuthorizationToken'
                  - 'ecr:BatchCheckLayerAvailability'
                  - 'ecr:GetDownloadUrlForLayer'
                  - 'ecr:BatchGetImage'
                  # ECS tasks to upload logs to CloudWatch
                  - 'logs:CreateLogStream'
                  - 'logs:PutLogEvents'
              Resource: '*'

Outputs:
```

ECSTaskExecutionRole:

Description: ECS Task Execution Role

Value: !GetAtt 'ECSTaskExecutionRole.Arn'

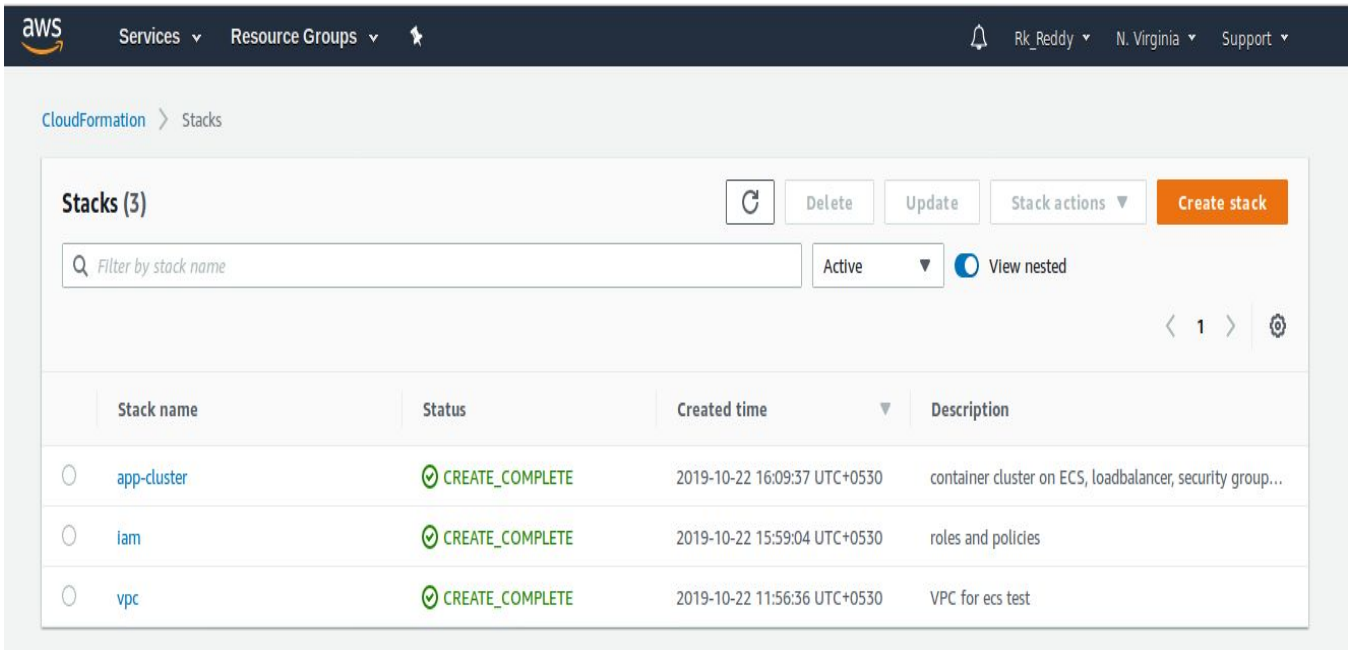
Export:

Name: 'ECSTaskExecutionRole'

Creation of app-cluster stack using yml

aws cloudformation create-stack --template-body file://$PWD/infra/app-cluster.yaml --stack-name app-cluster

```
[root@ip-172-31-91-76 docker-on-ecs]# aws cloudformation create-stack --template-body file://$PWD/infra/
app-cluster.yml --stack-name app-cluster
{
    "StackId": "arn:aws:cloudformation:us-east-1:287727666564:stack/app-cluster/3e954300-f4b8-11e9-b986-
0a56ba5f6f9e"
}
[root@ip-172-31-91-76 docker-on-ecs]#
```



CloudFormation > Stacks > app-cluster

Stacks (3)

Filter by stack name

Active

View nested

1

app-cluster
2019-10-22 16:09:37 UTC+0530
CREATE_COMPLETE

iam
2019-10-22 15:59:04 UTC+0530
CREATE_COMPLETE

vpc
2019-10-22 11:56:36 UTC+0530
CREATE_COMPLETE

Outputs (4)

Search outputs

Key	Value	Description	Export name
Cluster	bookstore	-	ECSCluster
ContainerSecurityGroup	sg-0959bf2db29e09995	container security group	ContainerSecurityGroup
Listener	arn:aws:elasticloadbalancing:us-east-1:287727666564:listener/app/ecs-services/31053d6d2fb8df2c/262d8deed349d127	listener port 80	Listener
LoadBalancerDNS	ecs-services-1097299462.us-east-1.elb.amazonaws.com	Domain name for the loadbalancer	DomainName

app-cluster.yml

```
AWSTemplateFormatVersion: '2010-09-09'
Description: container cluster on ECS, load balancer, security groups and cloudwatch

Resources:

  ECSCluster:
      Type: AWS::ECS::Cluster
      Properties:
      ClusterName: 'bookstore'

  LoadBalancer:
      Type: AWS::ElasticLoadBalancingV2::LoadBalancer
      Properties:
      Name: ecs-services
      Subnets:
      - !ImportValue 'Subnet1'
      - !ImportValue 'Subnet2'
      SecurityGroups:
      - !Ref LoadBalancerSecurityGroup
```

```
LoadBalancerListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
    LoadBalancerArn: !Ref LoadBalancer
    Protocol: HTTP
    Port: 80
    DefaultActions:
    - Type: forward
    TargetGroupArn: !Ref DefaultTargetGroup

LoadBalancerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
    GroupDescription: Security group for load balancer to services on ECS
    VpcId: !ImportValue 'VPC'
    SecurityGroupIngress:
    - CidrIp: 0.0.0.0/0
    IpProtocol: -1

DefaultTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
```

```
      Name: default
      VpcId: !ImportValue 'VPC'
      Protocol: 'HTTP'
      Port: '80'

  CloudWatchLogsGroup:
      Type: AWS::Logs::LogGroup
      Properties:
      LogGroupName: 'apis'
      RetentionInDays: 1

  ContainerSecurityGroup:
      Type: AWS::EC2::SecurityGroup
      Properties:
      VpcId: !ImportValue 'VPC'
      GroupDescription: for ecs containers
      SecurityGroupIngress:
      - SourceSecurityGroupId: !Ref 'LoadBalancerSecurityGroup'
      IpProtocol: -1


Outputs:
```

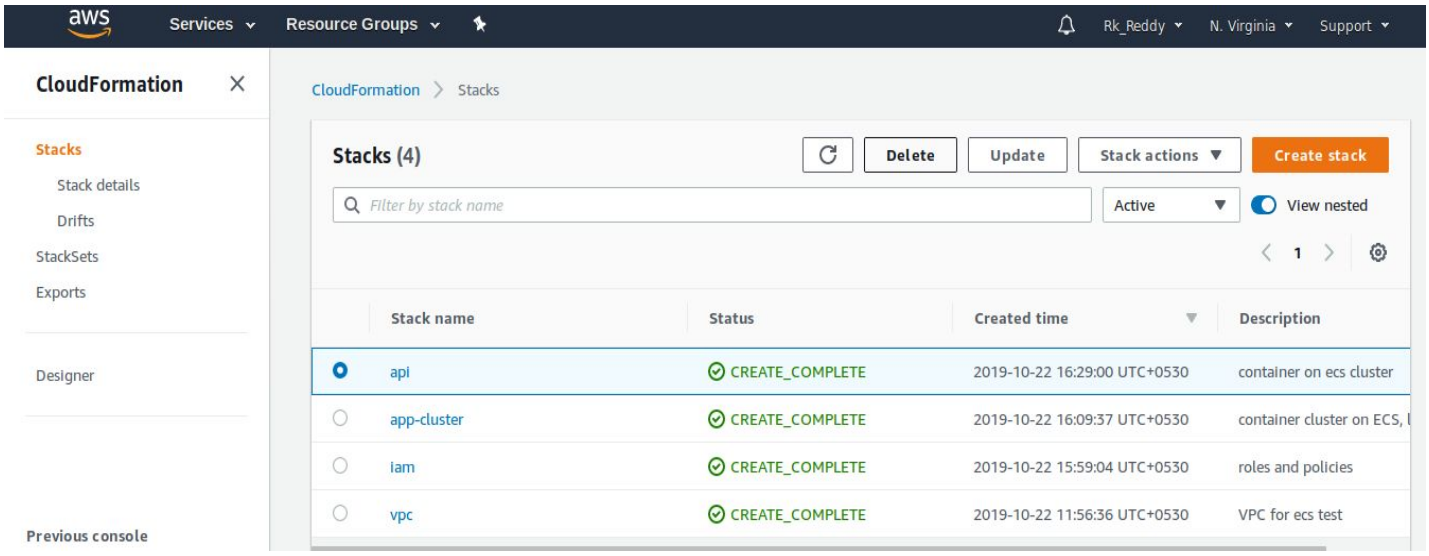
```
Cluster:
    Value: !Ref ECSCluster
    Export:
    Name: 'ECSCluster'

Listener:
    Description: listener port 80
    Value: !Ref LoadBalancerListener
    Export:
    Name: 'Listener'

ContainerSecurityGroup:
    Description: container security group
    Value: !Ref ContainerSecurityGroup
    Export:
    Name: 'ContainerSecurityGroup'

LoadBalancerDNS:
    Description: Domain name for the load balancer
    Value: !GetAtt LoadBalancer.DNSName
    Export:
```

Name: 'DomainName'

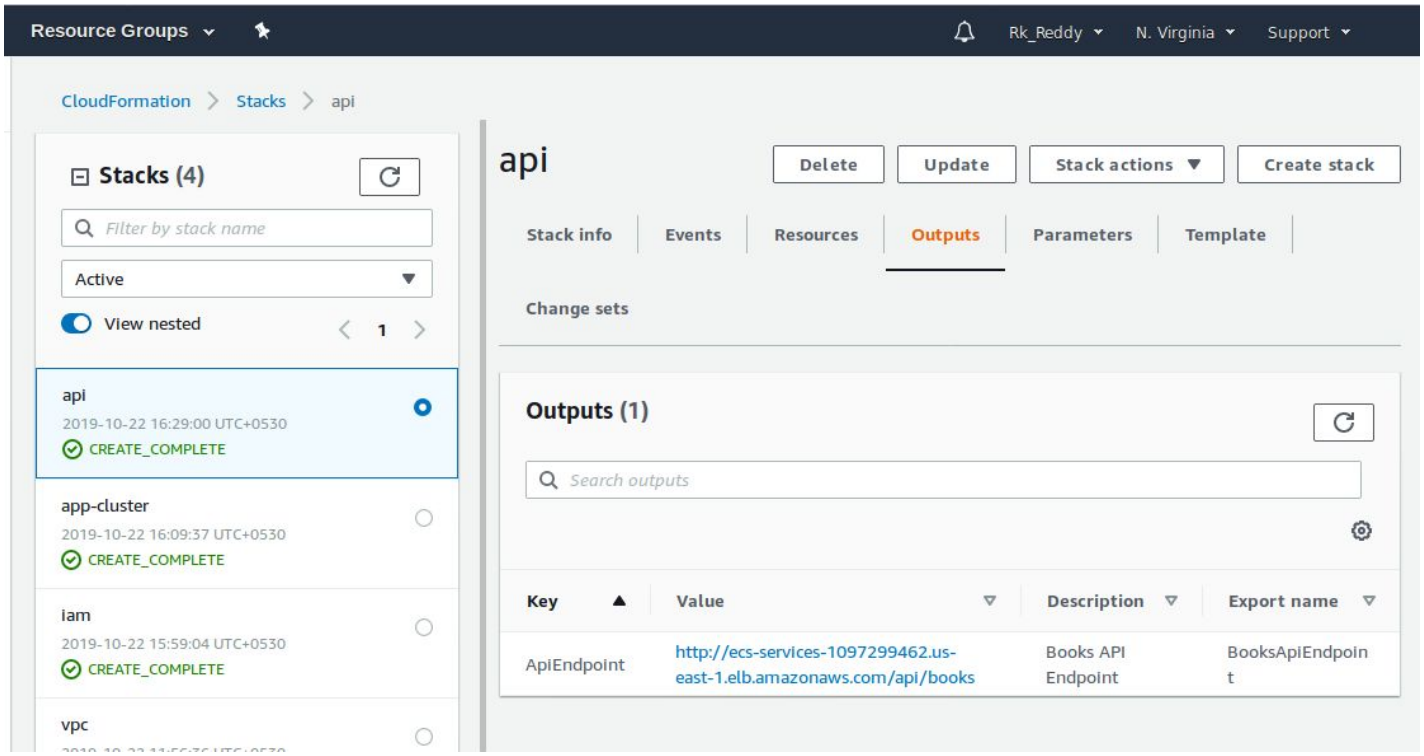
Creation of api stack using yml

aws cloudformation create-stack --template-body file://$PWD/infra/api.yaml --stack-name api

```
[root@ip-172-31-91-76 docker-on-ecs]# aws cloudformation create-stack --template-body file://$PWD/infra/
api.yml --stack-name api
{
    "StackId": "arn:aws:cloudformation:us-east-1:287727666564:stack/api/f3cddc80-f4ba-11e9-a2f7-1271b8f3
5690"
}
[root@ip-172-31-91-76 docker-on-ecs]#
```



We created four cloudformation stacks api,app-cluster,iam and vpc using yml scripting.



api.yml

AWSTemplateFormatVersion: '2010-09-09'

Description: container on ecs cluster

```
Resources:

  Task:
      Type: AWS::ECS::TaskDefinition
      Properties:
      Family: apis
      Cpu: 256
      Memory: 512
      NetworkMode: aws vpc
      RequiresCompatibilities:
      - FARGATE
      ExecutionRoleArn: !ImportValue ECSTaskExecutionRole
      ContainerDefinitions:
      - Name: books
      Image: 287727666564.dkr.ecr.us-east-1.amazonaws.com/rk_reddy
      Cpu: 256
      Memory: 512
      PortMappings:
      - ContainerPort: 4567
      Protocol: tcp
      LogConfiguration:
```

```
        LogDriver: awslogs
        Options:
        awslogs-group: 'apis'
        awslogs-region: !Ref AWS::Region
        awslogs-stream-prefix: 'books'

Service:
        Type: AWS::ECS::Service
        DependsOn: ListenerRule
        Properties:
        ServiceName: books-service
        TaskDefinition: !Ref Task
        Cluster: !ImportValue 'ECSCluster'
        LaunchType: FARGATE
        DesiredCount: 2
        DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 70
        NetworkConfiguration:
        AwsvpcConfiguration:
        AssignPublicIp: ENABLED
        Subnets:
```

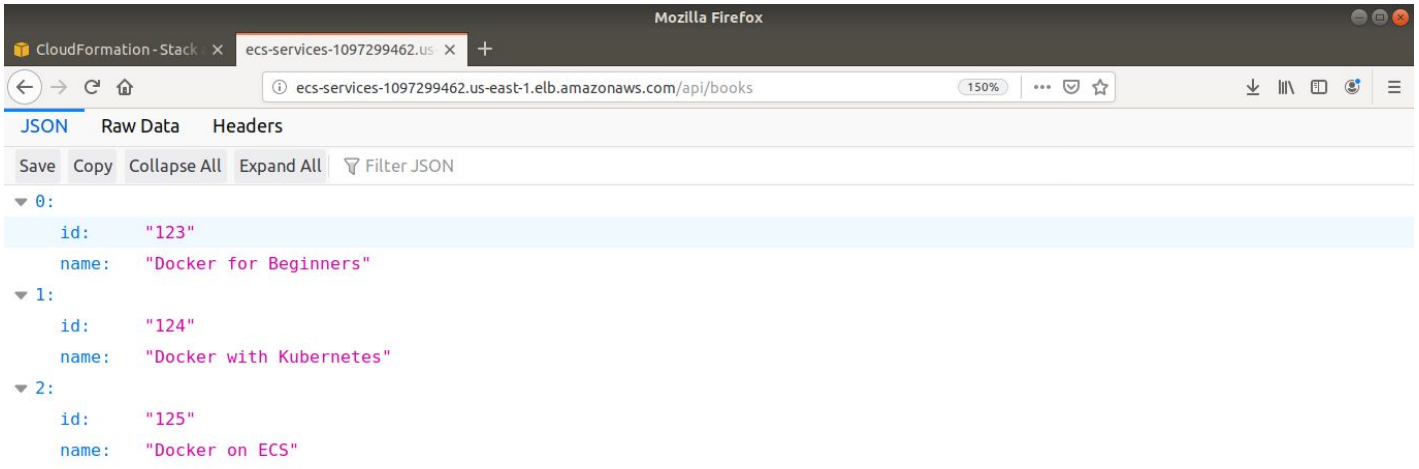
```
        - !ImportValue Subnet1
        - !ImportValue Subnet2
        SecurityGroups:
        - !ImportValue ContainerSecurityGroup
        LoadBalancers:
        - ContainerName: books
        ContainerPort: 4567
        TargetGroupArn: !Ref TargetGroup

TargetGroup:
        Type: AWS::ElasticLoadBalancingV2::TargetGroup
        Properties:
        Name: books-tg
        VpcId: !ImportValue VPC
        Port: 80
        Protocol: HTTP
        Matcher:
        HttpCode: 200-299
        HealthCheckIntervalSeconds: 10
        HealthCheckPath: /stat
        HealthCheckProtocol: HTTP
        HealthCheckTimeoutSeconds: 5
```

```
      HealthyThresholdCount: 10
      TargetType: ip

  ListenerRule:
      Type: AWS::ElasticLoadBalancingV2::ListenerRule
      Properties:
      ListenerArn: !ImportValue Listener
      Priority: 2
      Conditions:
      - Field: path-pattern
      Values:
      - /api/book*
      Actions:
      - TargetGroupArn: !Ref TargetGroup
      Type: forward


Outputs:

  ApiEndpoint:
      Description: Books API Endpoint
      Value: !Join ['', ['http://', !ImportValue DomainName, '/api/books']]
```

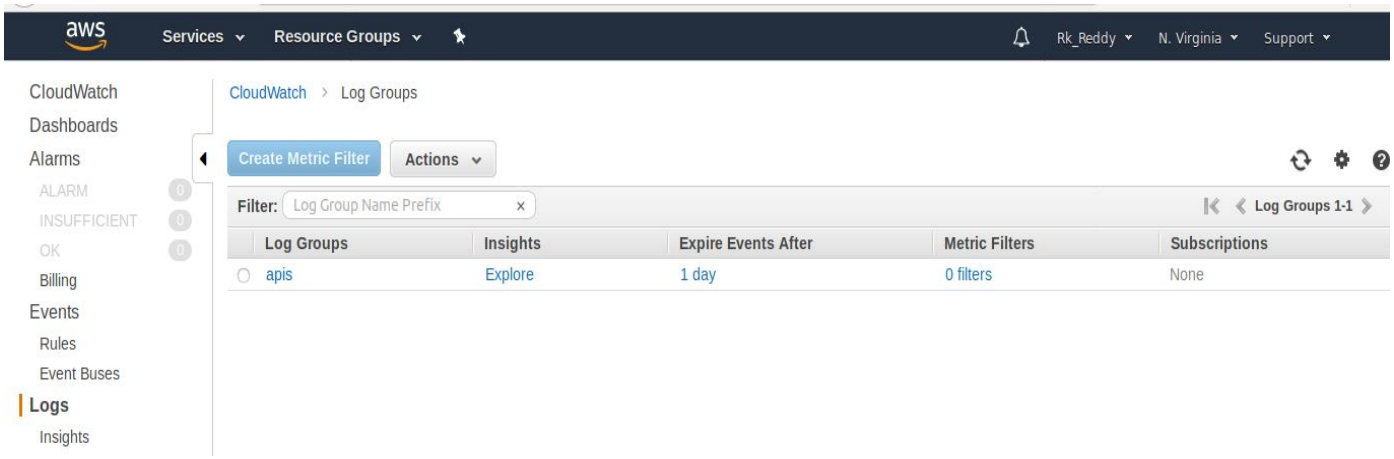
Export:

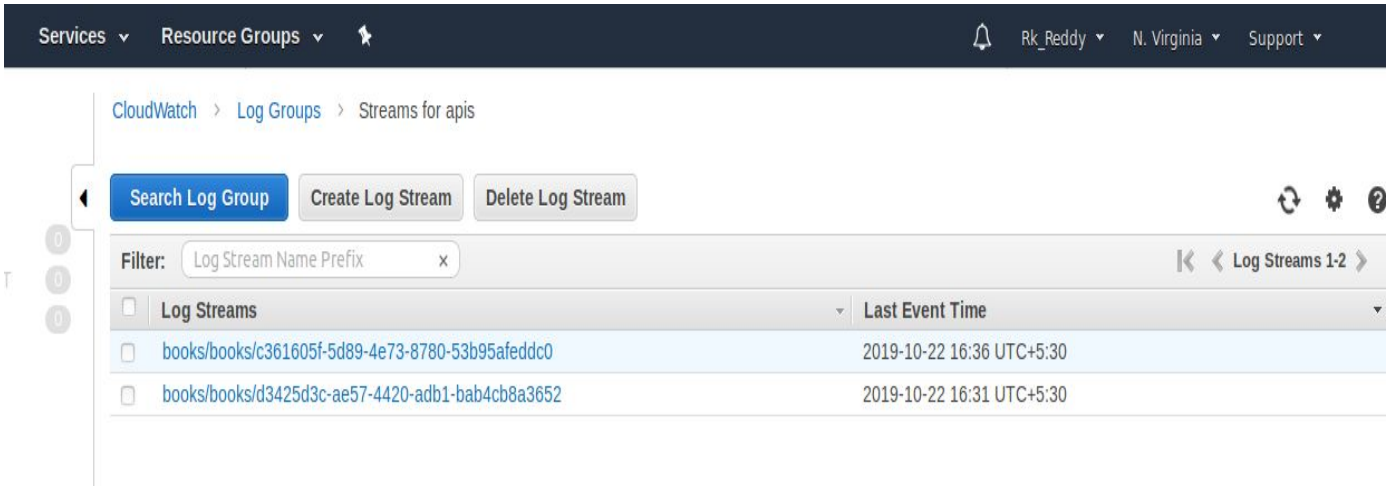
Name: 'Books Api Endpoint'

Copy the url under api stack and search in the browser.

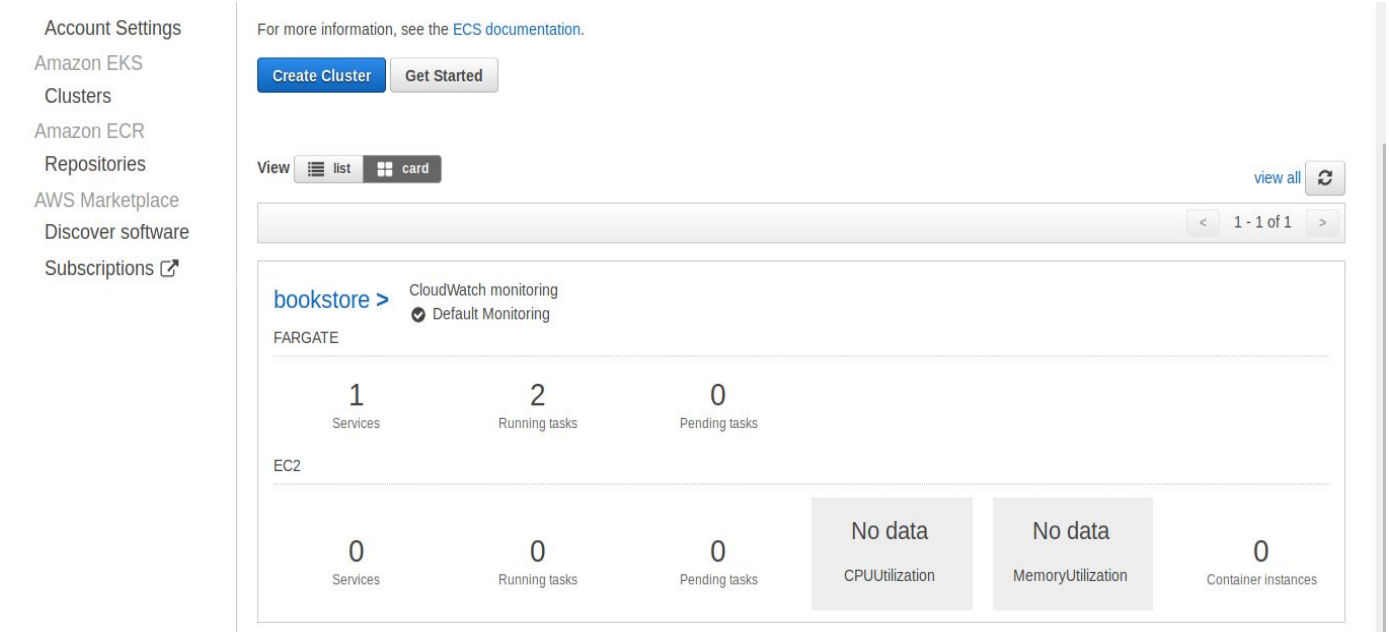


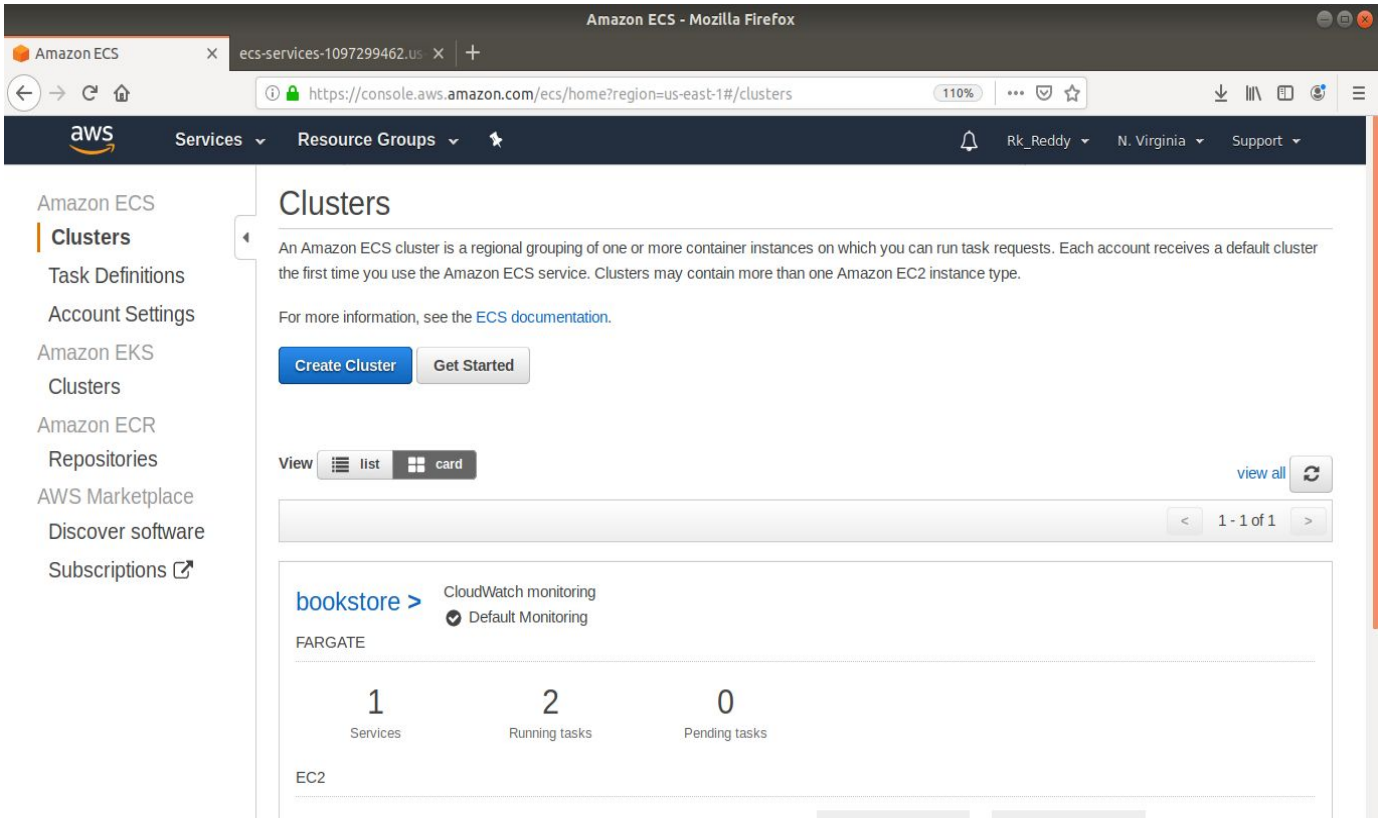
Below picture shows the Log Group under Logs of cloudwatch. We created a group named **apis.**



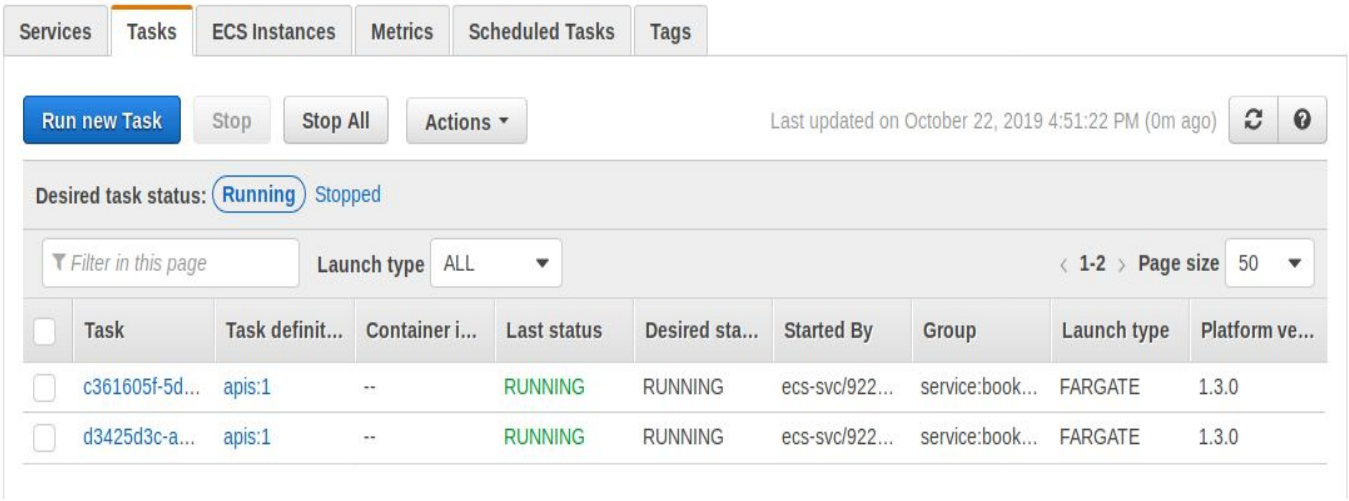


These are the log streams for log group **apis**





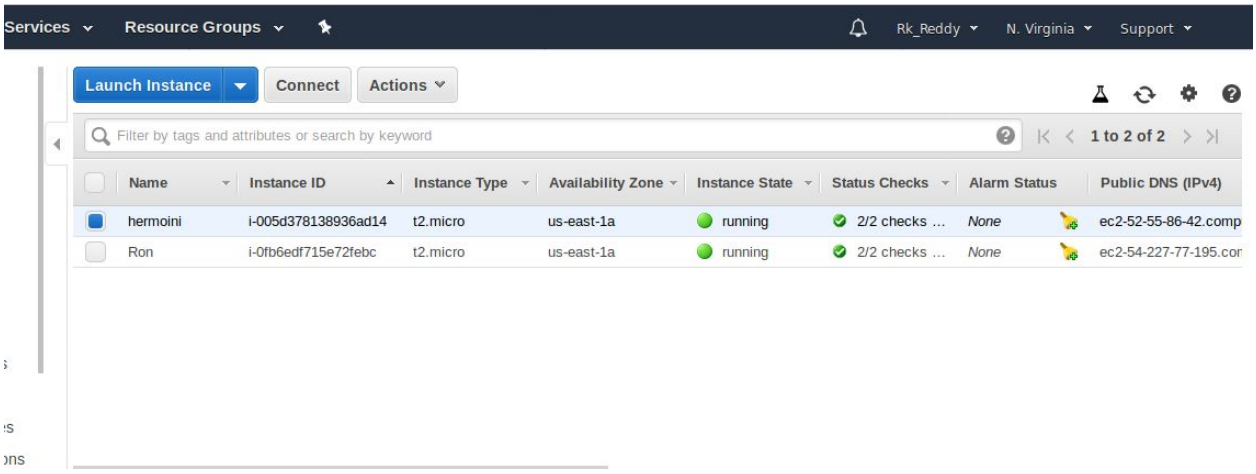
Finally, We successfully created the ECS cluster **bookstore**.



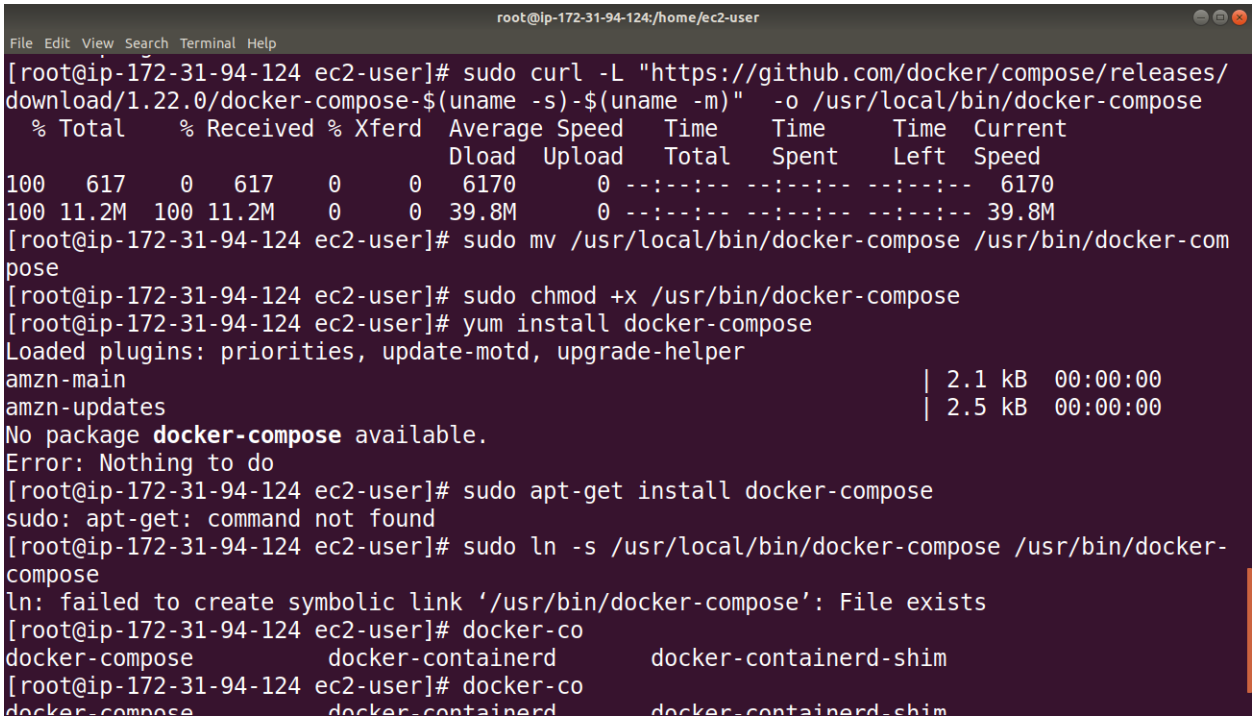
Mysql

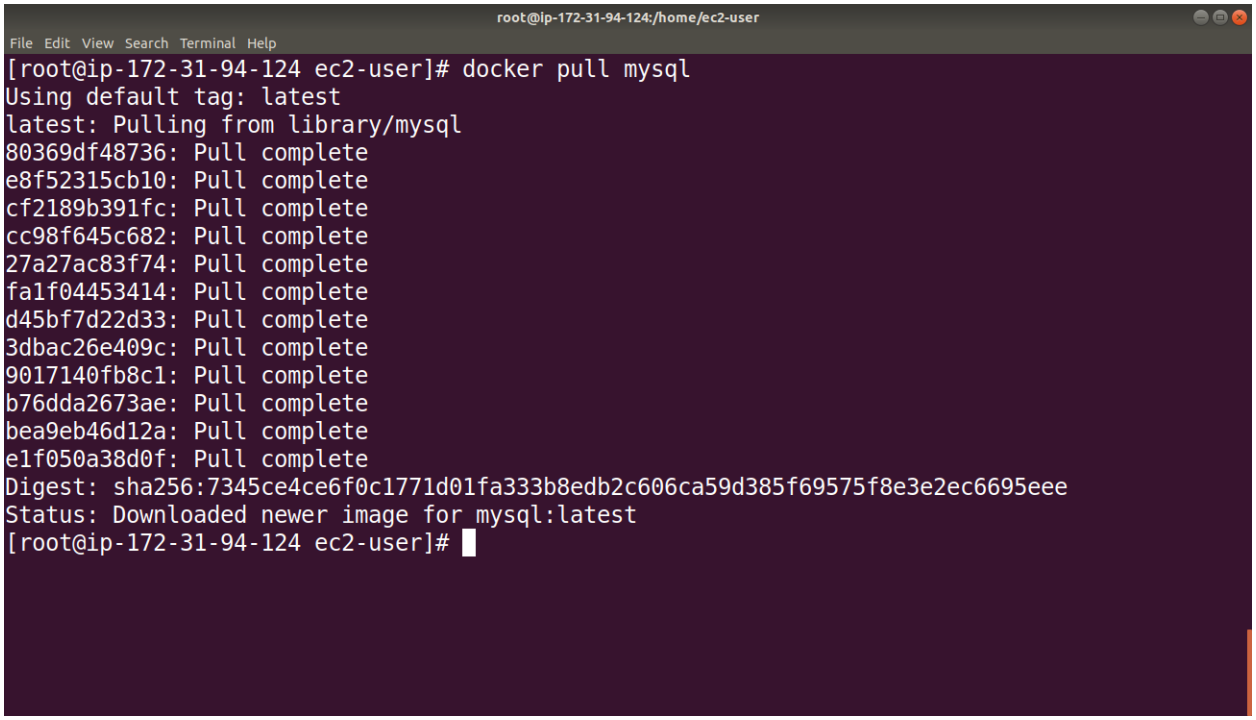
Mysql remote access of host from another host using **docker container.**

> Launch an EC2 instance and install **docker** and **docker-compose** in it.

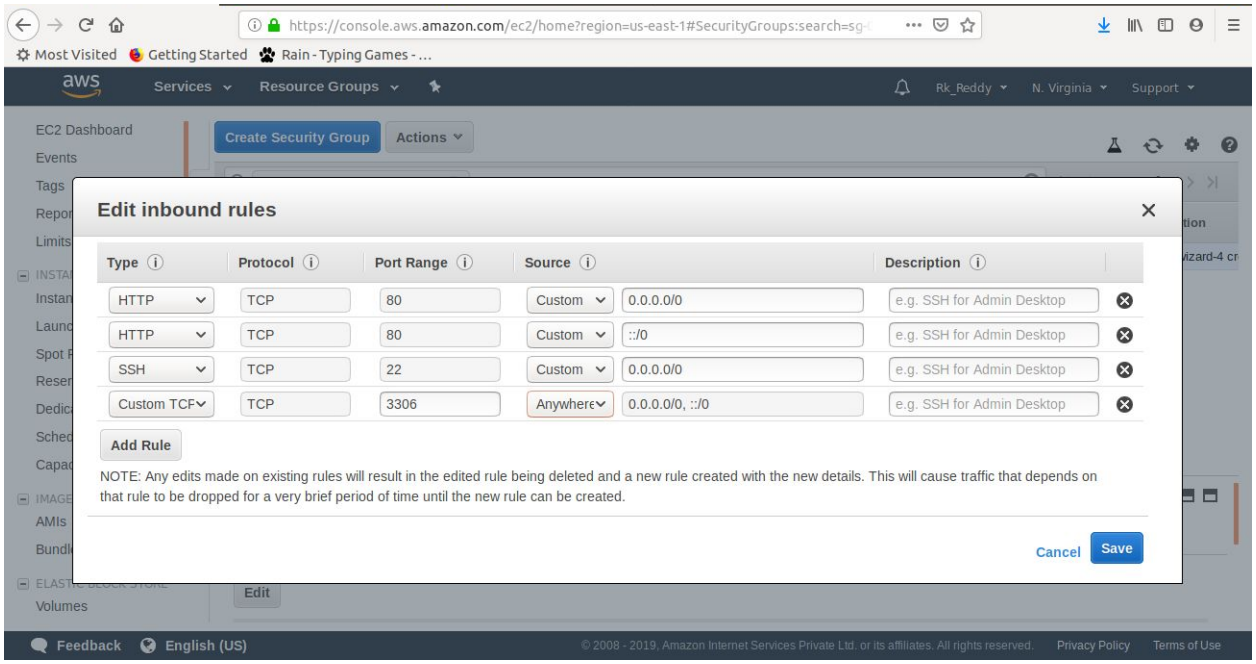



```
root@ip-172-31-94-124:/home/ec2-user
File  Edit  View  Search  Terminal  Help
[root@ip-172-31-94-124 ec2-user]# yum install docker
Loaded plugins: priorities, update-motd, upgrade-helper
Resolving Dependencies
--> Running transaction check
---> Package docker.x86_64 0:18.06.1ce-10.32.amzn1 will be installed
--> Processing Dependency: xfsprogs for package: docker-18.06.1ce-10.32.amzn1.x86_64
--> Processing Dependency: pigz for package: docker-18.06.1ce-10.32.amzn1.x86_64
--> Processing Dependency: libseccomp.so.2()(64bit) for package: docker-18.06.1ce-10.32.amz
n1.x86_64
--> Processing Dependency: libltdl.so.7()(64bit) for package: docker-18.06.1ce-10.32.amzn1.
x86_64
--> Running transaction check
---> Package libseccomp.x86_64 0:2.3.1-2.4.amzn1 will be installed
---> Package libtool-ltdl.x86_64 0:2.4.2-20.4.8.5.32.amzn1 will be installed
---> Package pigz.x86_64 0:2.3.3-1.6.amzn1 will be installed
---> Package xfsprogs.x86_64 0:4.5.0-18.23.amzn1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package          Arch       Version                      Repository        Size
================================================================================
Installing:
```

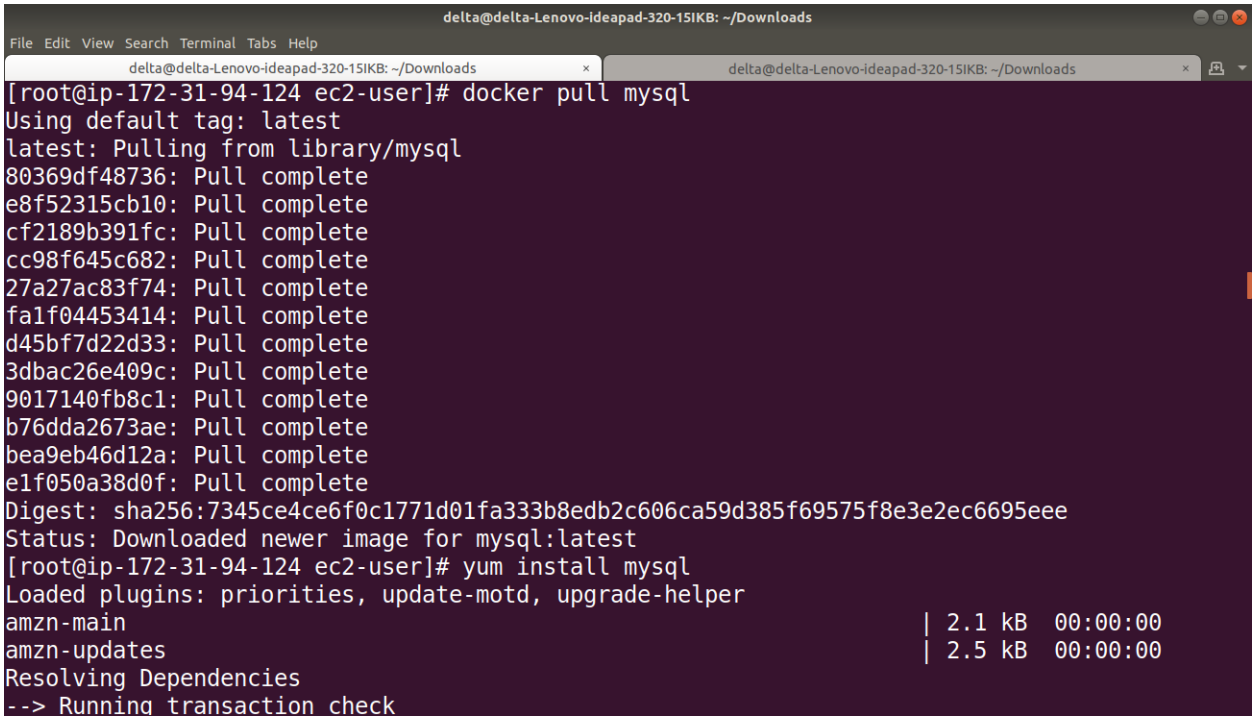


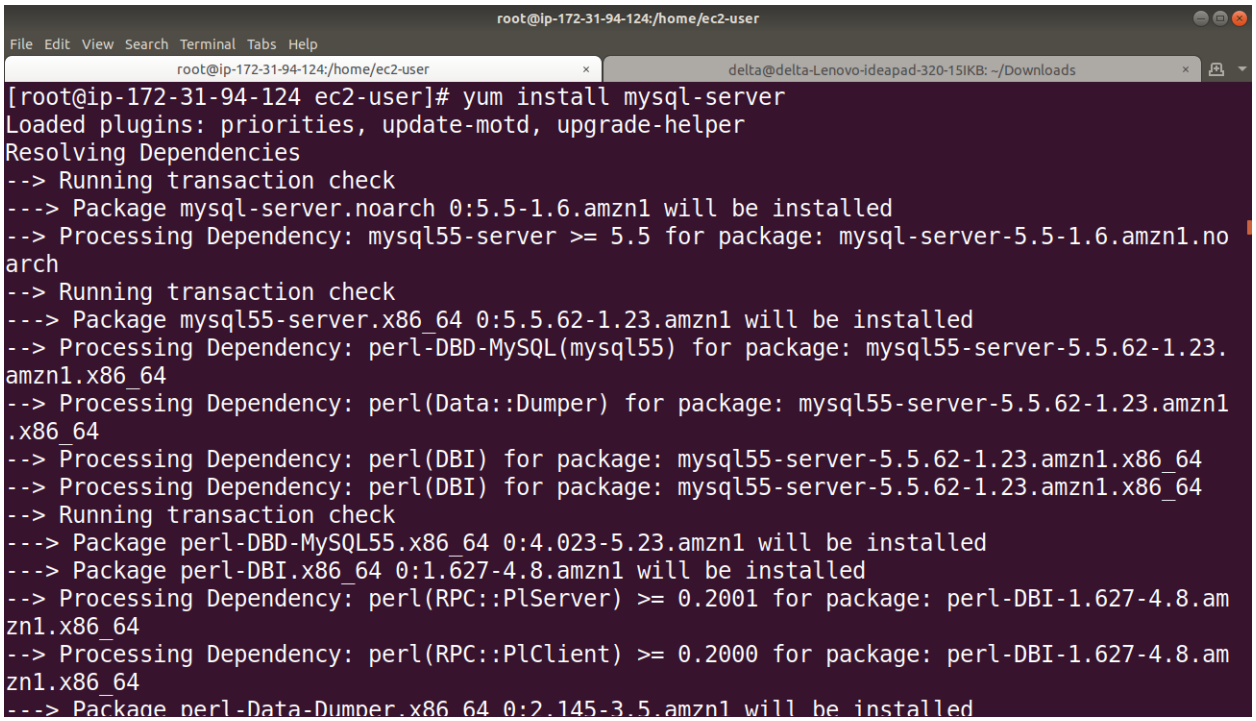


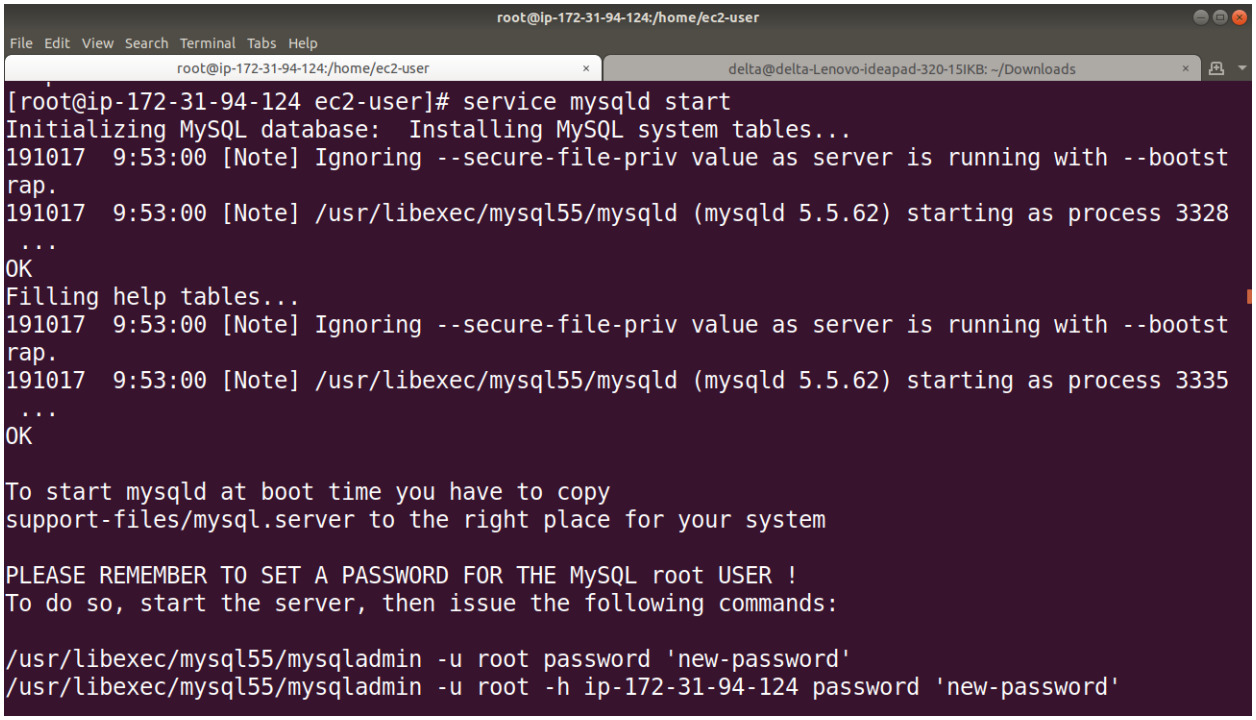
> Allow mysql service in security group of the instance.



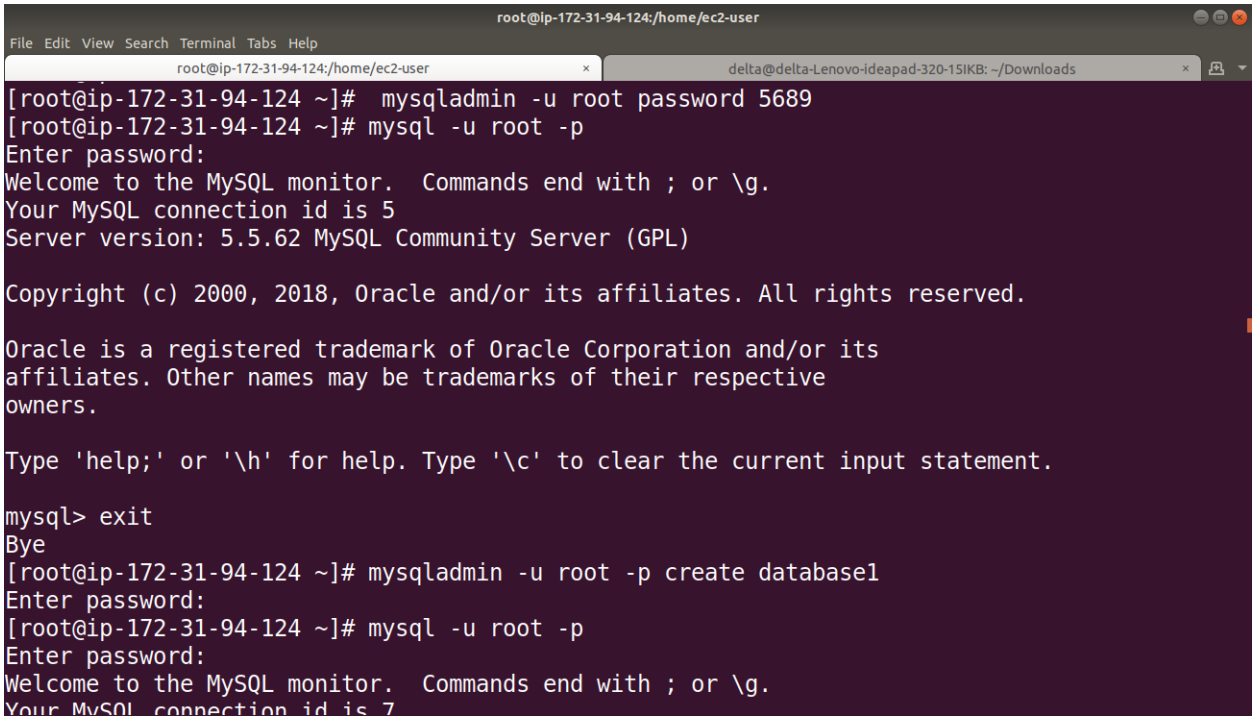
> Pull **mysql** image from the **docker hub**.



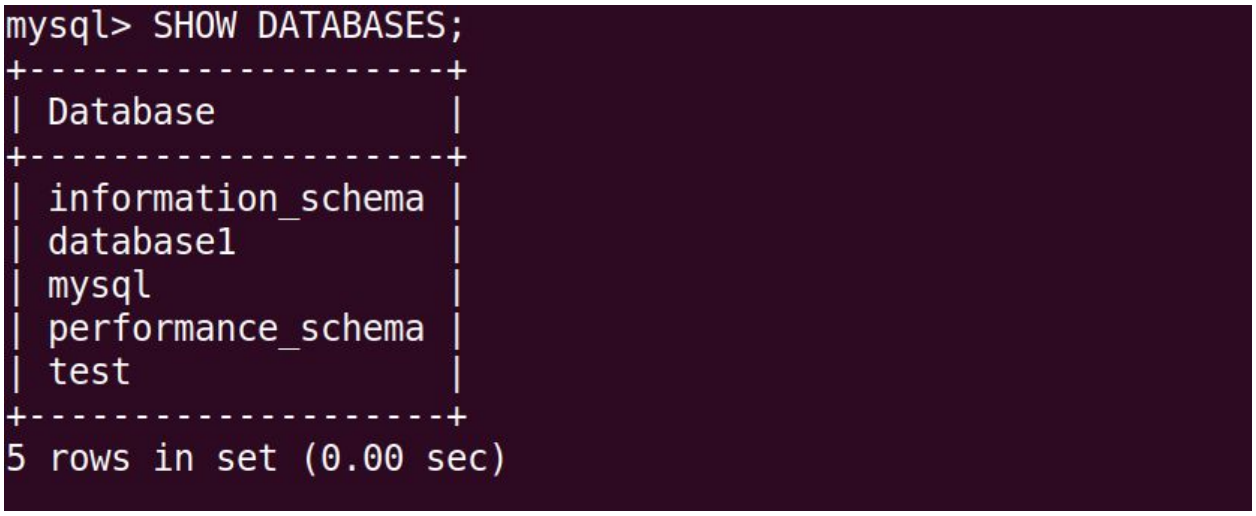




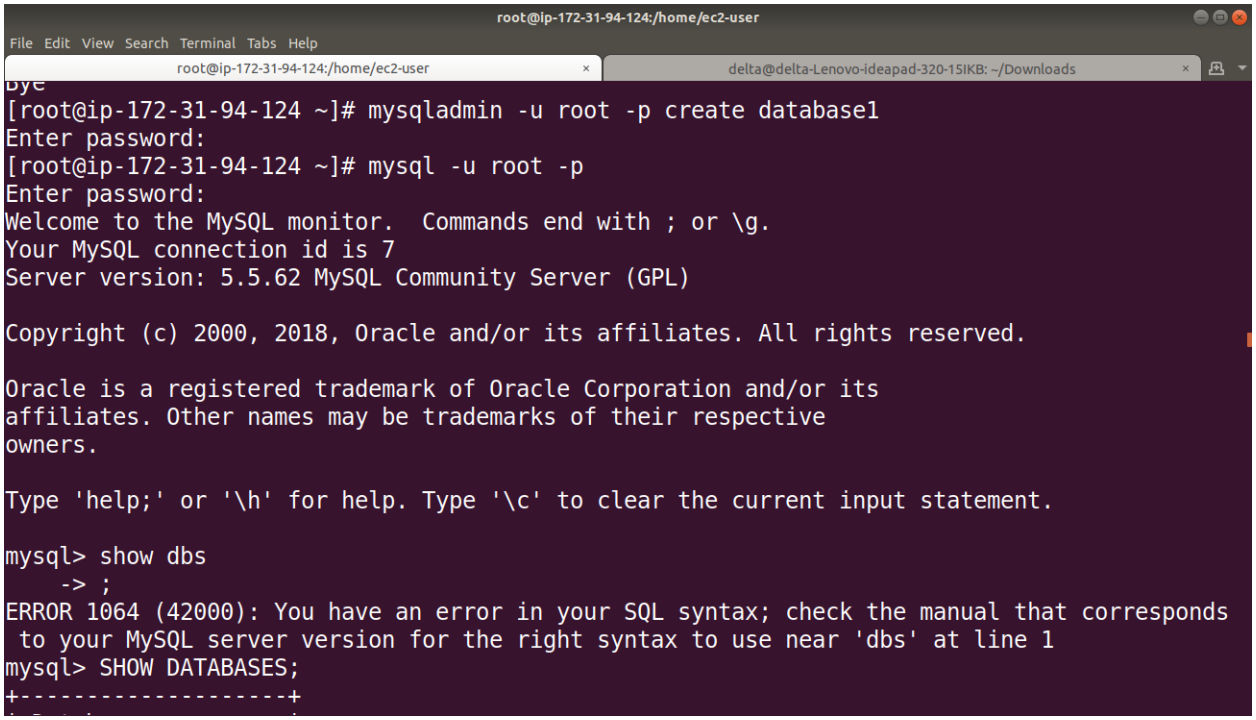
> Create the credentials of Root user and login to mysql using those credentials



> DataBase's



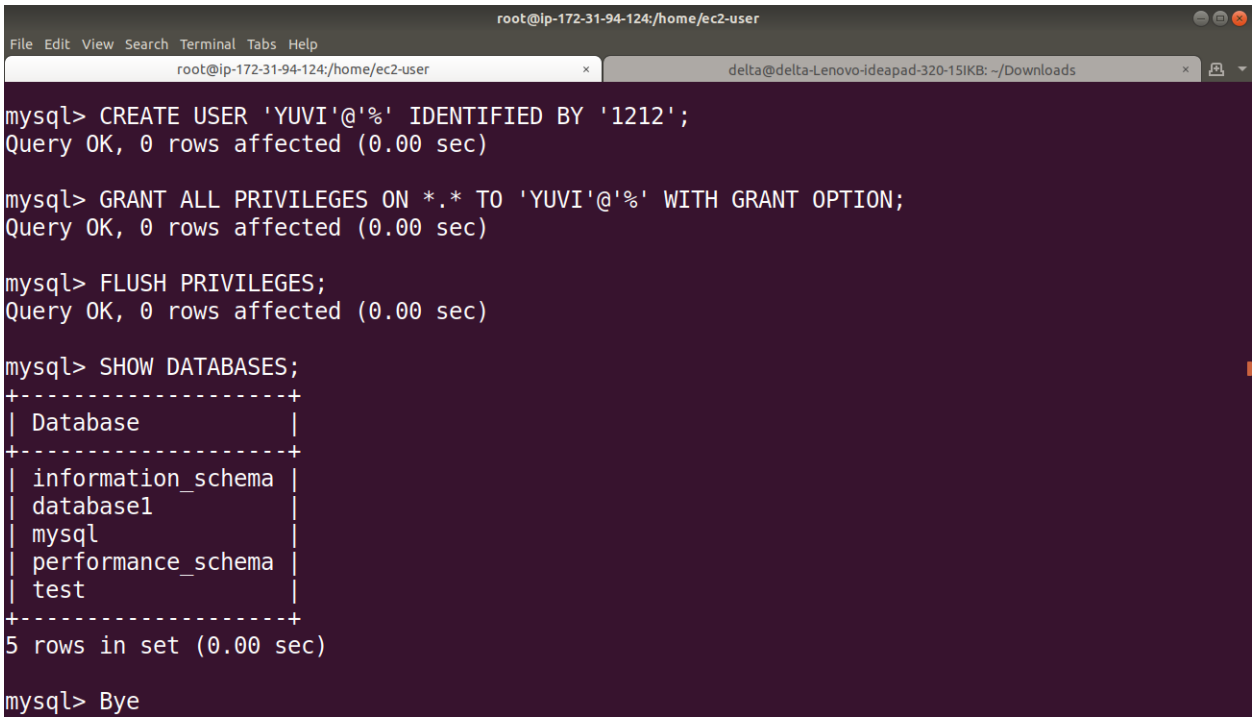
> Create one database and login through root user.



> Create one table and add some arguments. Here, I created RkReddy.

```
mysql> use database1
Database changed
mysql> CREATE TABLE RkReddy (name VARCHAR(100), roll_no INT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO RkReddy(name,roll_no) values("RkR",12);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * from RkReddy;
+------+---------+
| name | roll_no |
+------+---------+
| RkR  |      12 |
+------+---------+
1 row in set (0.00 sec)
```

> Assign privileges to the user to get access through another user.



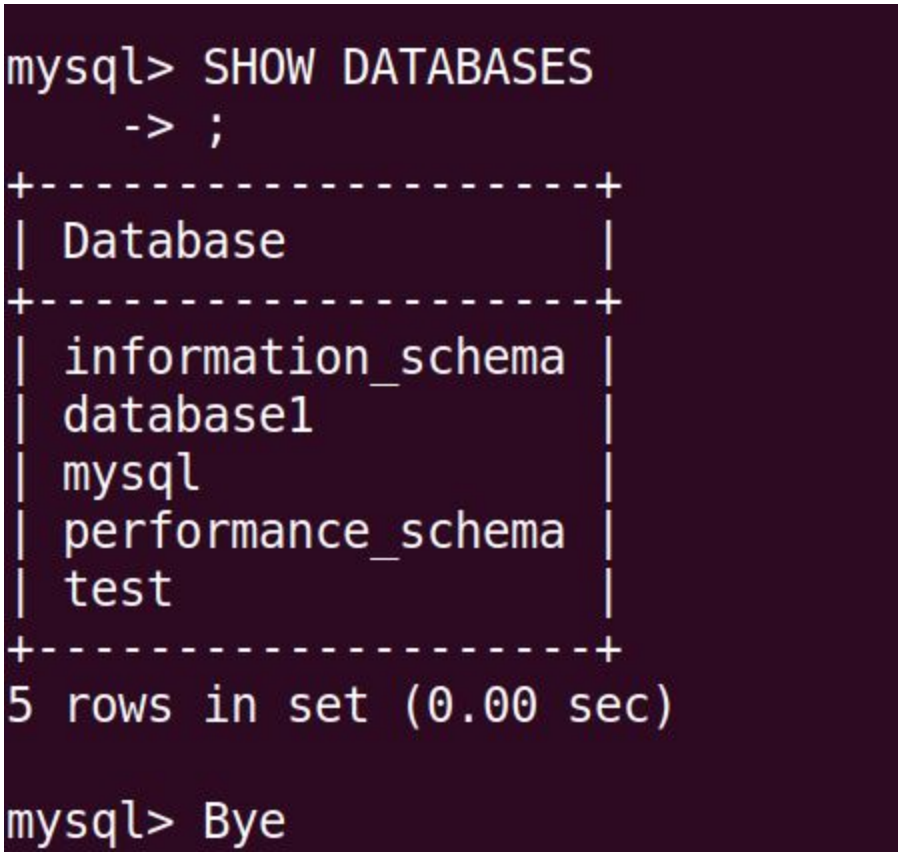
> Launch another instance
> Use **mysql -h host ip -u username -p** and enter password to login

```
delta@delta-Lenovo-ideapad-320-15IKB:~/Downloads$ ssh -i a.pem ec2-user@52.55.86.42
The authenticity of host '52.55.86.42 (52.55.86.42)' can't be established.
ECDSA key fingerprint is SHA256:1MdXW1dNdz1Uz6rV678gFaQAOSSLpw03OgS4/HcCrNo.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.55.86.42' (ECDSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
4 package(s) needed for security, out of 10 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-83-253 ~]$ sudo su
[root@ip-172-31-83-253 ec2-user]# yum install mysql-server
Loaded plugins: priorities, update-motd, upgrade-helper
amzn-main                                                   | 2.1 kB  00:00:00
amzn-updates                                                | 2.5 kB  00:00:00
Resolving Dependencies
--> Running transaction check
---> Package mysql-server.noarch 0:5.5-1.6.amzn1 will be installed
--> Processing Dependency: mysql55-server >= 5.5 for package: mysql-server-5.5-1.6.amzn1.no
arch
```

```
[root@ip-172-31-83-253 ec2-user]# service mysqld start
Initializing MySQL database:  Installing MySQL system tables...
191017 10:06:42 [Note] Ignoring --secure-file-priv value as server is running with --bootst
rap.
191017 10:06:42 [Note] /usr/libexec/mysql55/mysqld (mysqld 5.5.62) starting as process 2976
 ...
OK
Filling help tables
```

```
[root@ip-172-31-83-253 ec2-user]# mysql -h 54.227.77.195 -u YUVI -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.5.62 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES
```

> here, We are inside sql monitor. Next Type **SHOW DATABASES;** to check DB's.



> Here, We can check the all DB's resides inside the main host using the command.

mysqldump -u root -p --all-databases > all.sql