Employee Laptop Delivery Tracking - Deployment

Ajit Singh
Divyangna Sinha
T. Harshasai
Jahnavi Paruchuri
Siddhi Khanvilkar
Sravan Pindipolu

Presentation Content

- Introduction to our REST API
- Problem Statement - Deployment
- Our Solution
- Introduction to Amazon EC2, Docker & Kubernetes
- Steps followed

Introduction to our REST API

An employee registered with an organization has to go through the lengthy process of connecting with the IT support team to submit repair request for a registered laptop or submit new laptop request, and has to frequently follow up with IT team to cross check delivery status.
This consumes a lot of time and man hours.

To automate hassles of the IT support team such as receiving laptop related requests, allocating new laptops to newly registered employees, etc. we have created a RESTful API using Spring Boot. Quick repair, update and tracking of delivery will ensure faster services. Tracking of delivery status of devices will ensure safe delivery of device across location.

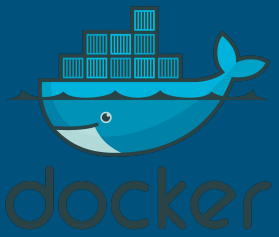
Problem Statement

We have to deploy our application keeping the following things in mind:

1. Dependency: While deploying the application and scaling up resources according to the requirement, we need to consider all the application dependencies and make sure that all dependencies for the application are available with respective versions.

2. High Availability: When users are increased our resources should be ready to scale up which is not possible with an on-premises setup until and unless you have huge resources.

Our solution



Based on the problem statement, we have used the following to ensure dependency and high availability in our deployment:

1. Dependency: All the application dependencies can be packed as a Docker Image along with the application. So when we run the Docker Image, the Application runs seamlessly without any issues with dependency issues. (Packaging and shipping containers through docker)

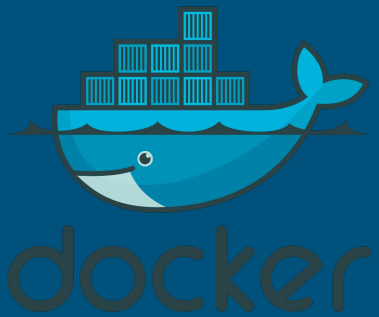
2. High Availability: The application can be available all the time by deploying in Kubernetes using K8's deployments. The application can use a load balancer and auto-scaling in AWS, also utilize the resources and scale up immediately with a 'pay for usage' policy using AWS EKS. (Using cloud deployment and K8s for container management)

Amazon EC2

- Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) Cloud.
- EC2 deploys isolated virtual computing environments (instances) with auto scaling support.
- Using Amazon EC2 eliminates our need to invest in hardware up front, so we can develop and deploy applications faster.
- We can use Amazon EC2 to launch as many or as few virtual servers as we need, configure security and networking, and manage storage.

Docker

- p1
- p2
- p3

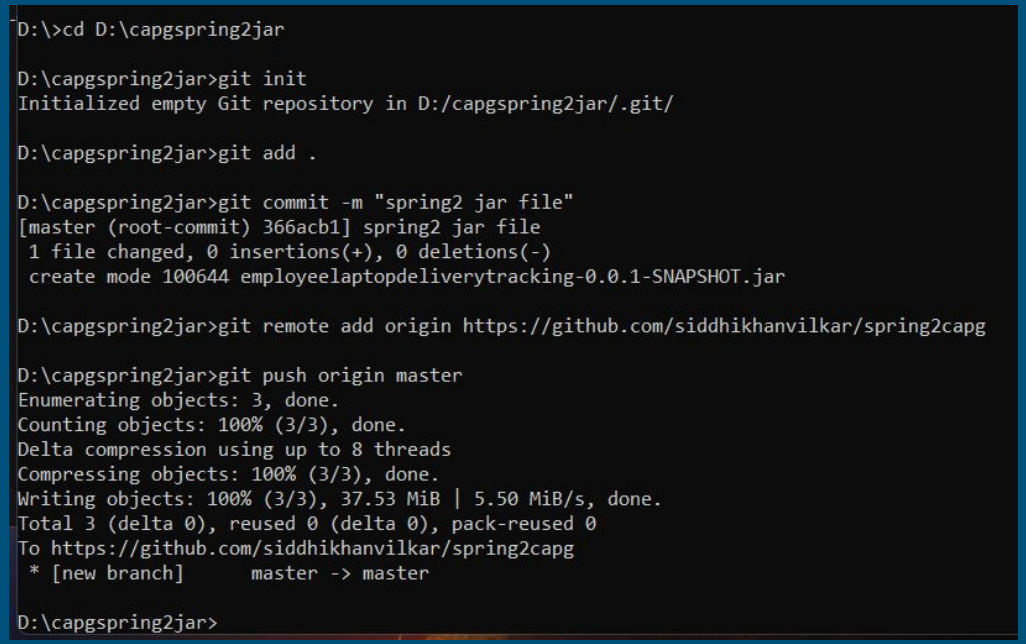


Kubernetes

- p1
- p2
- p3

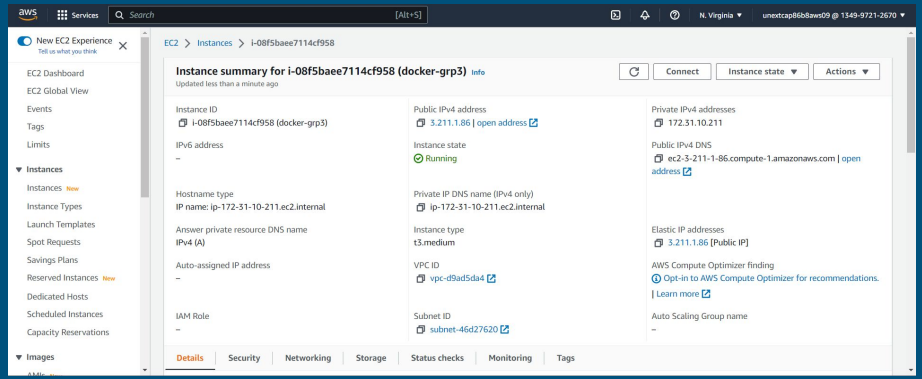
Uploading Application On GitHub

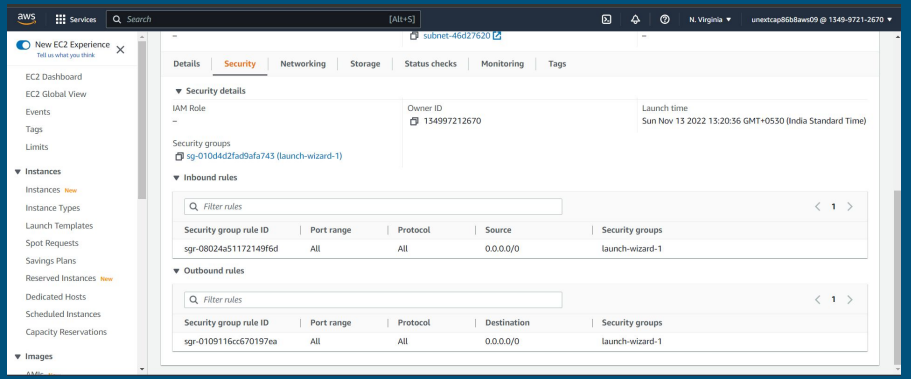
Created GitHub repository and uploaded project jar file.



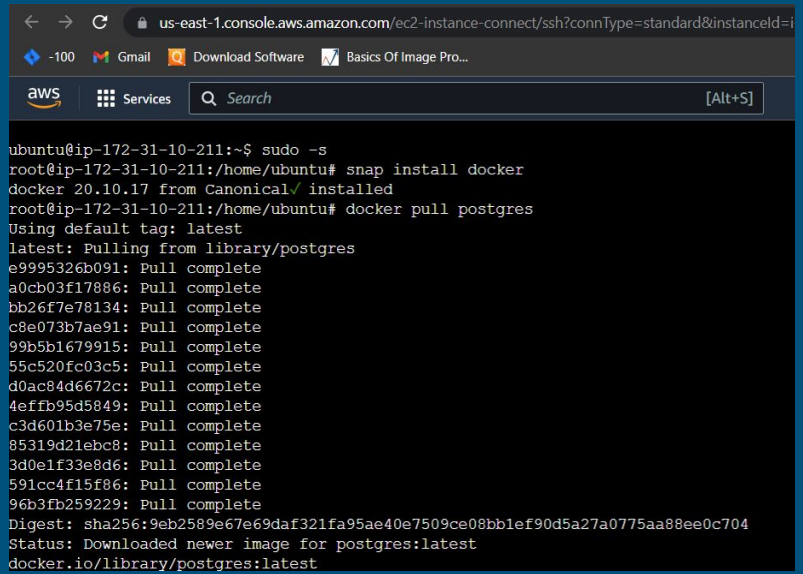
Deployment on AWS-Docker

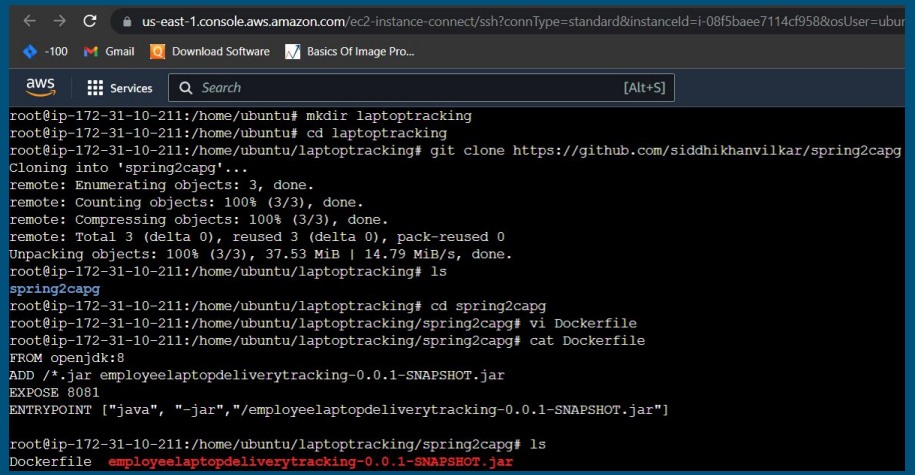
1. Created ec2-instance , changed port range to All traffic and assigned elastic ip.





Installed docker and postgres on ubuntu terminal, Cloned github repository (to get application code file) and created dockerfile





Dockerfile , docker-compose file:-

```
FROM openjdk:8
ADD /*.jar employeelaptopdeliverytracking-0.0.1-SNAPSHOT.jar
EXPOSE 8081
ENTRYPOINT ["java", "-jar","/employeelaptopdeliverytracking-0.0.1-SNAPSHOT.jar"]

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```

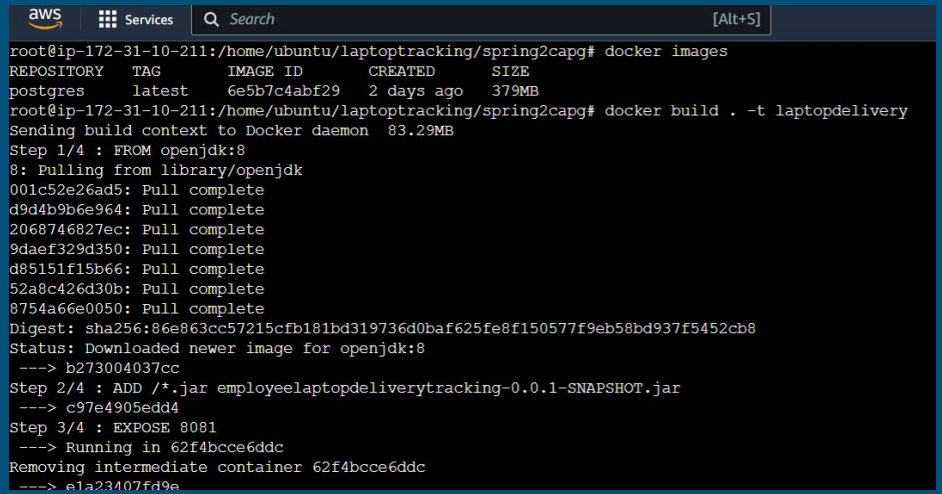
Dockerfile

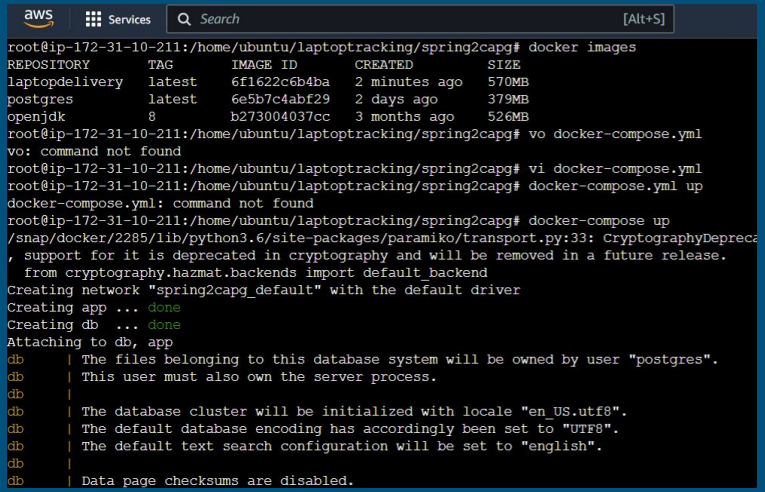
```
version: '2'
services:
        app:
                image: 'laptopdelivery'
                build:
                        context: .
                container_name: app
                ports:
                        - 8081:8081
                environment:
                        - SPRING_DATASOURCE_URL=jdbc:postgresql://db/compose-postgres
                        - SPRING_DATASOURCE_USERNAME=compose-postgres
                        - SPRING_DATASOURCE_PASSWORD=compose-postgres
                        - SPRING_JPA_HIBERNATE_DDL_AUTO=update
        db:
                image: 'postgres'
                container_name: db
                environment:
                        - POSTGRES_USER=compose-postgres
                        - POSTGRES_PASSWORD=compose-postgres
```

docker-compose.yml

Built image with the help of dockerfile, Created and ran docker-compose file.

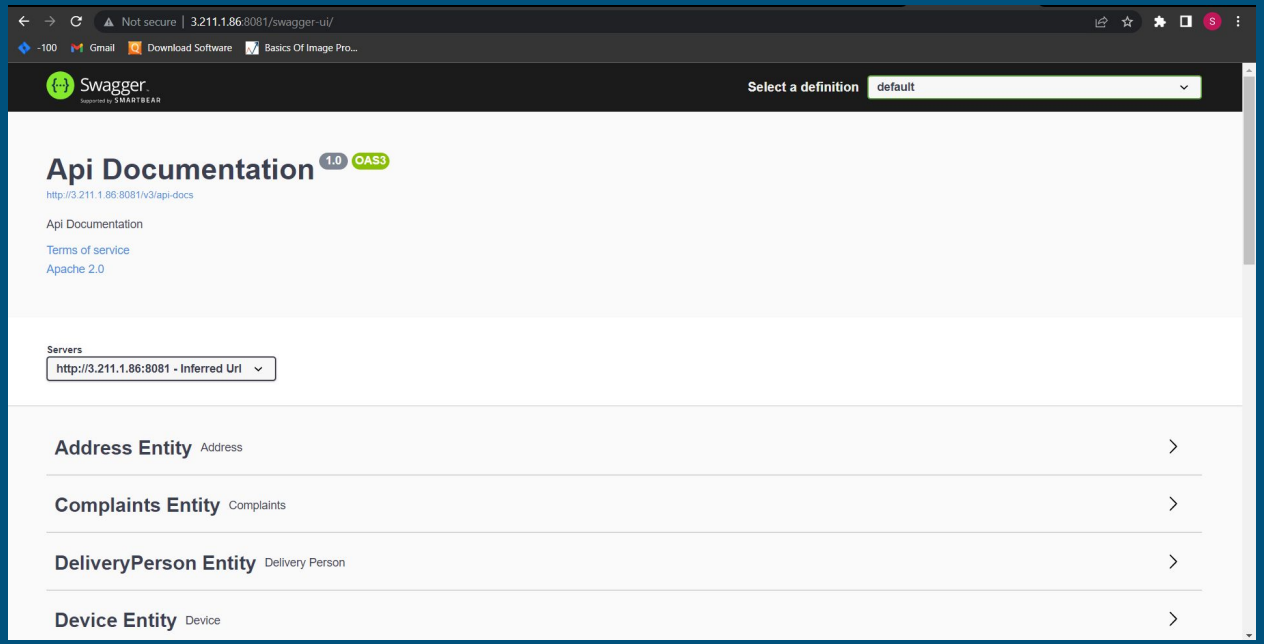
Pushed the image to docker hub afterward for using later on in EKS deployment.





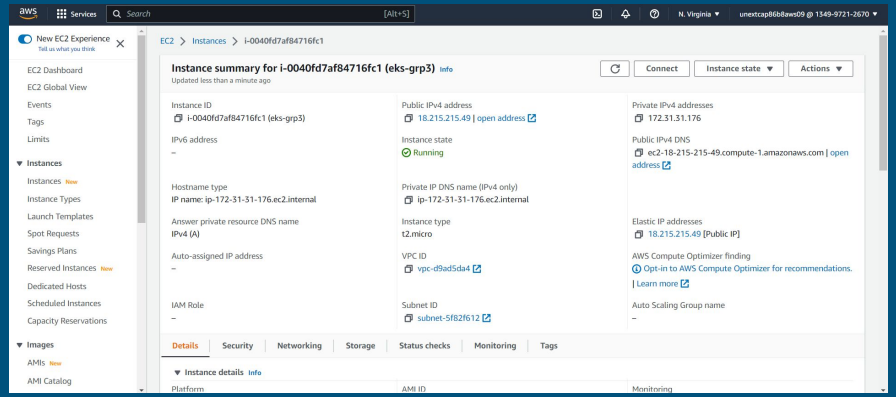
Using Public IP address to access application:

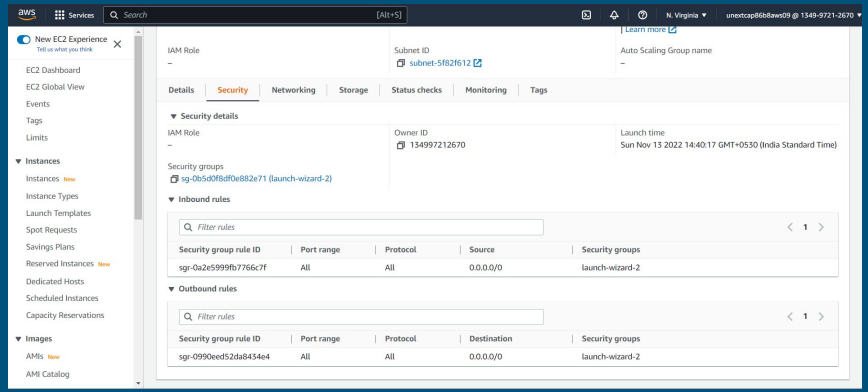
http://3.211.1.86:8081/swagger-ui/



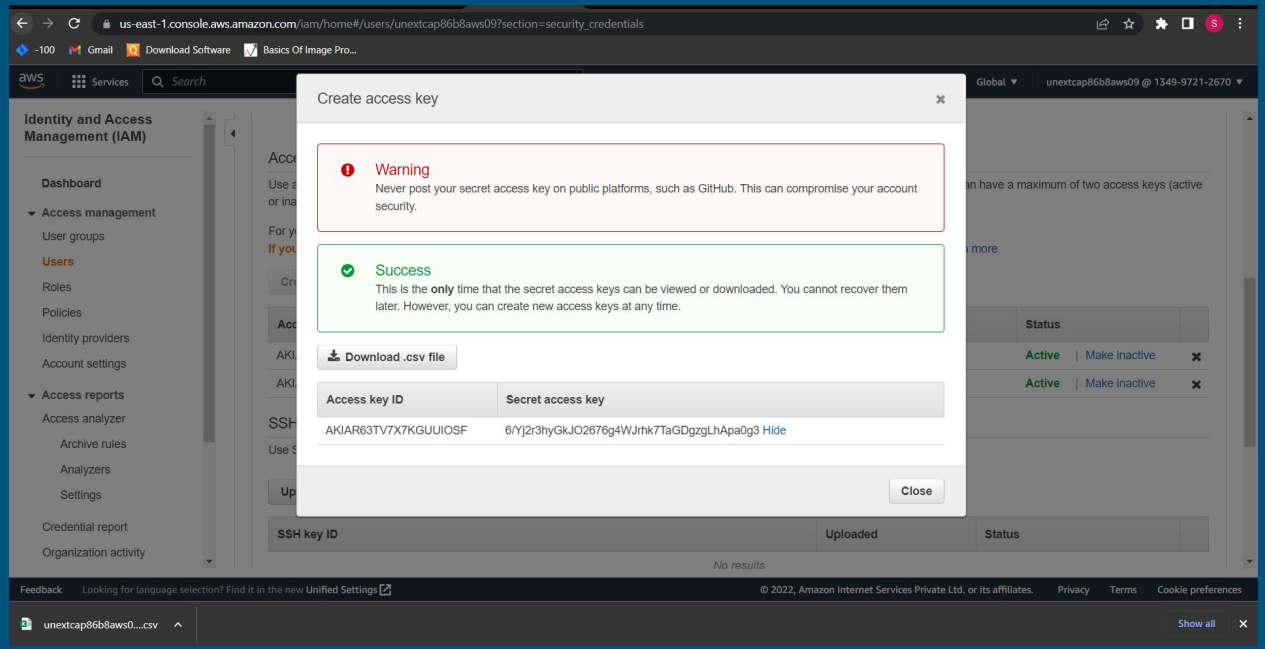
Deployment Using AWS-EKS

Created instance using Amazon Linux 2 AMI, changed port range to All traffic and assigned elastic ip

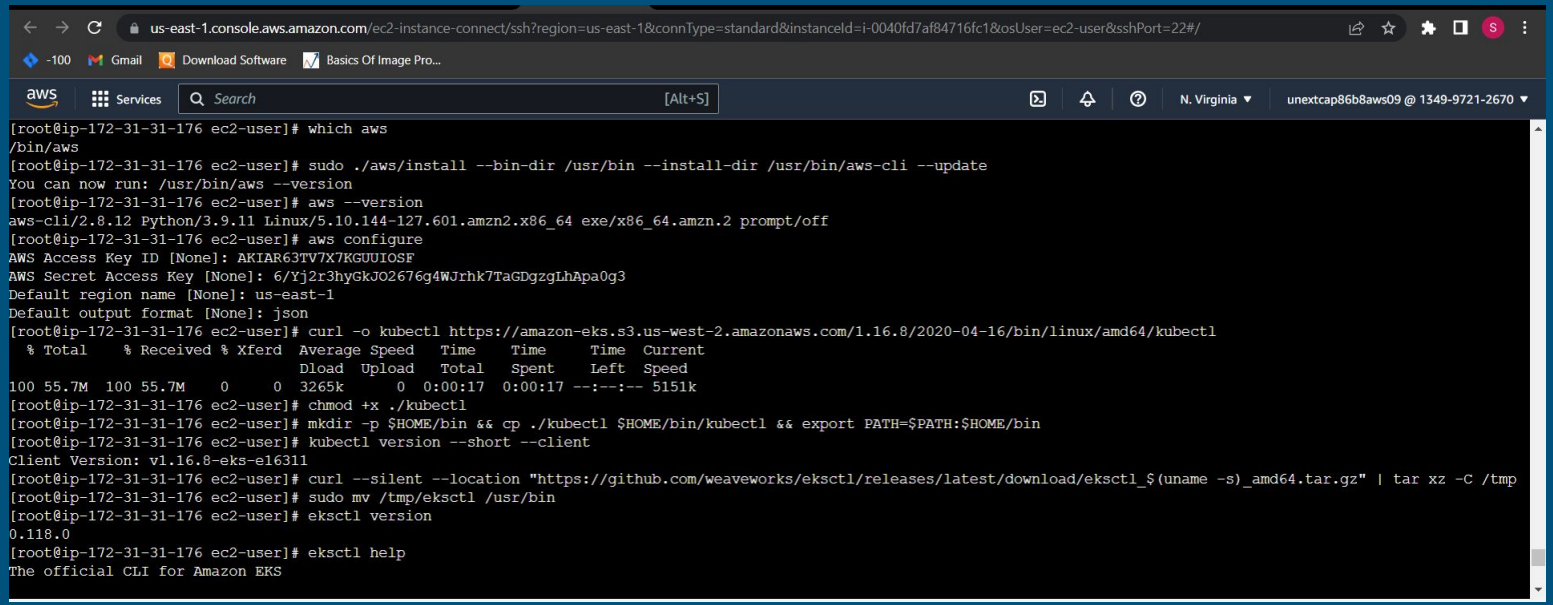




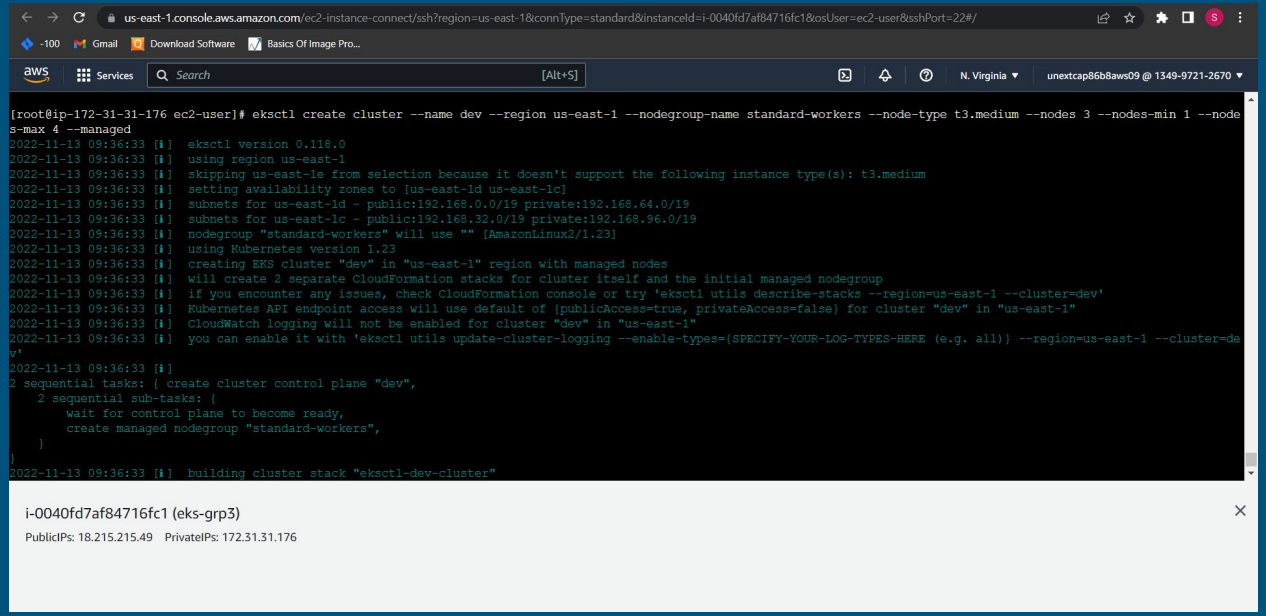
Created Access Key from Users



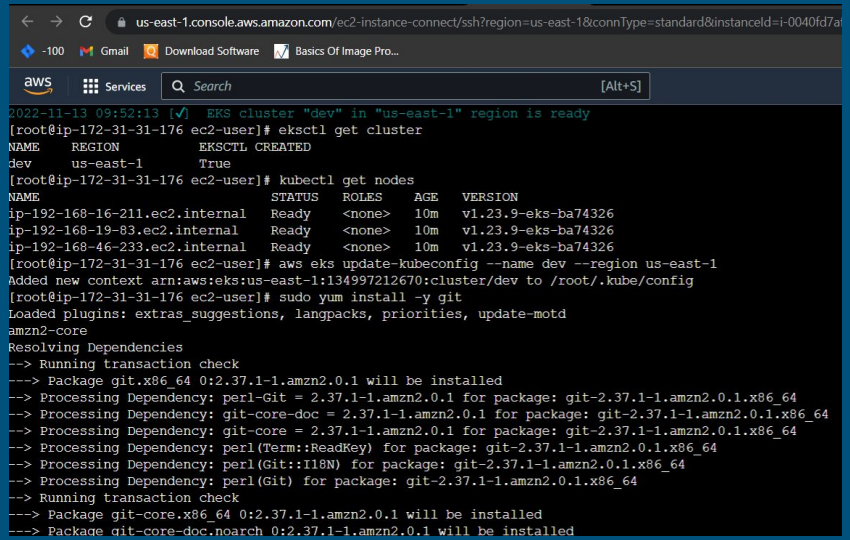
Connected to instance and configured AWS for using EKS

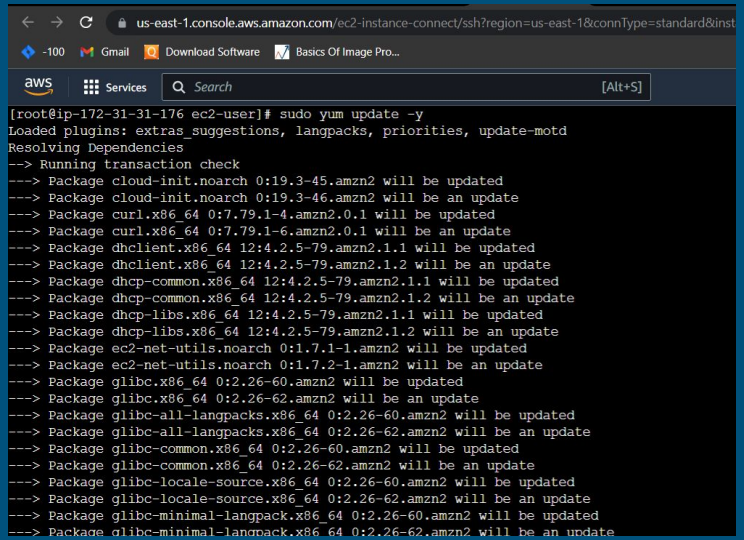


Created cluster with name dev, node type t3-medium with 3 nodes.

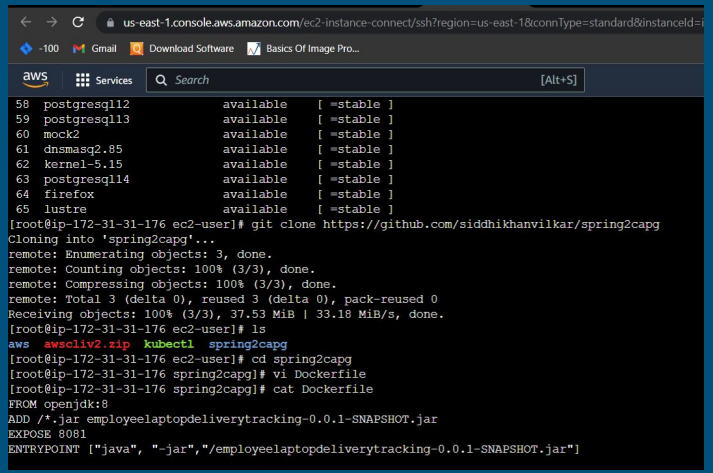


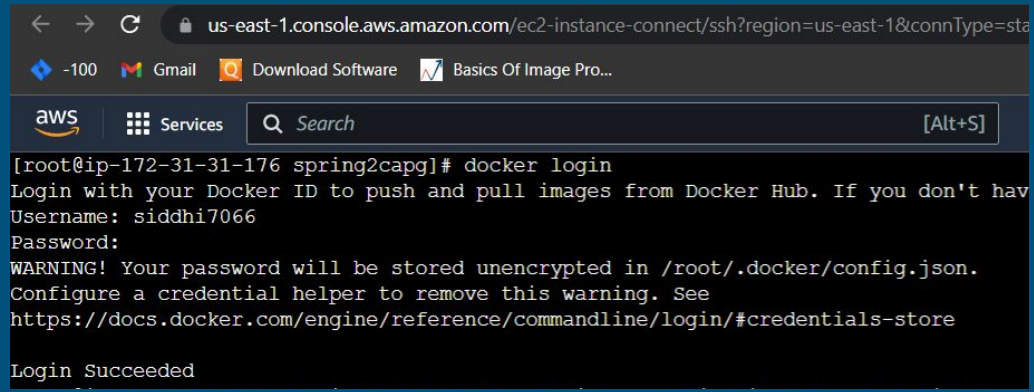
Viewed the created clusters and nodes and installed-updated the required libraries



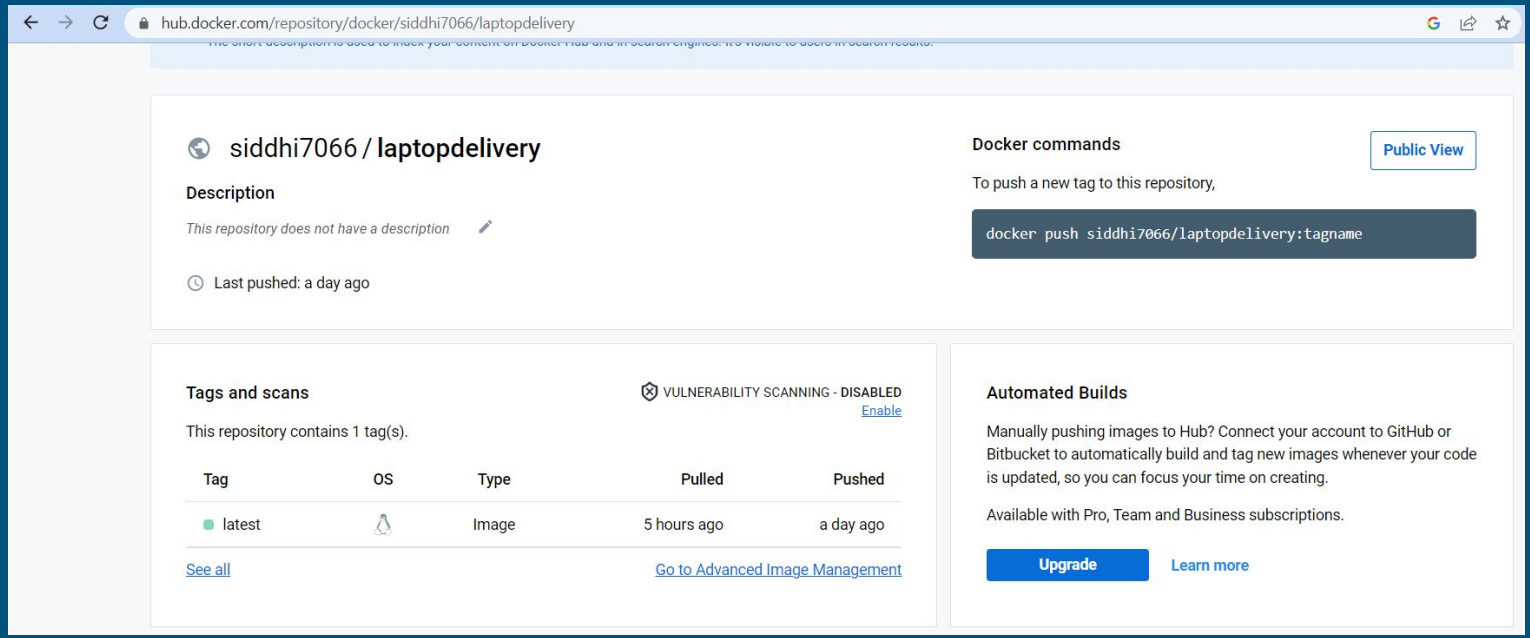


Cloned github repository, created Dockerfile - pushed to dockerhub and pull image from dockerhub

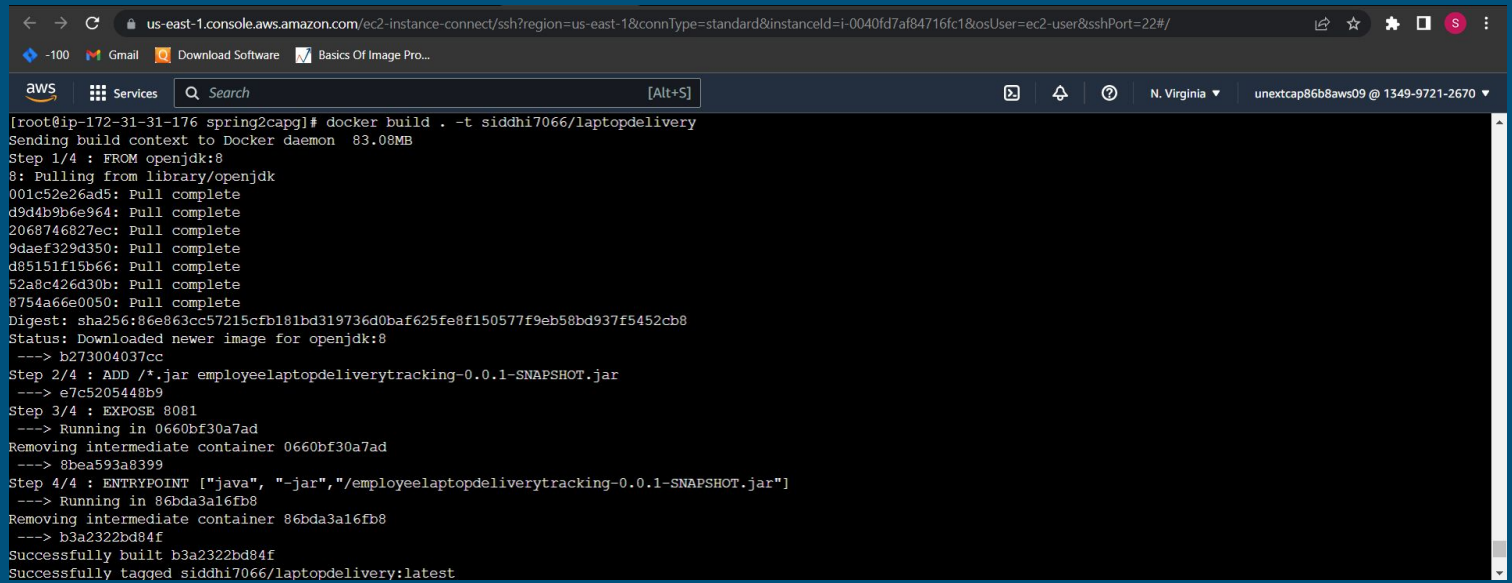




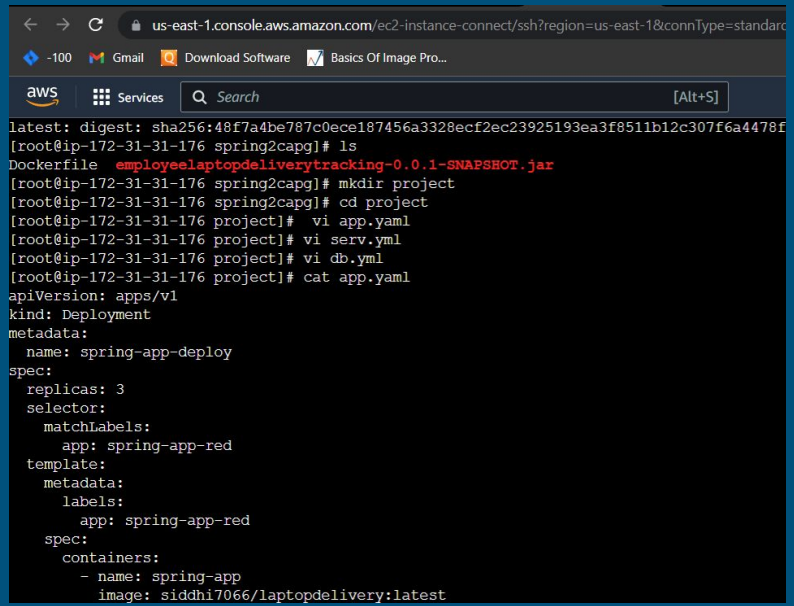
Docker hub repository example

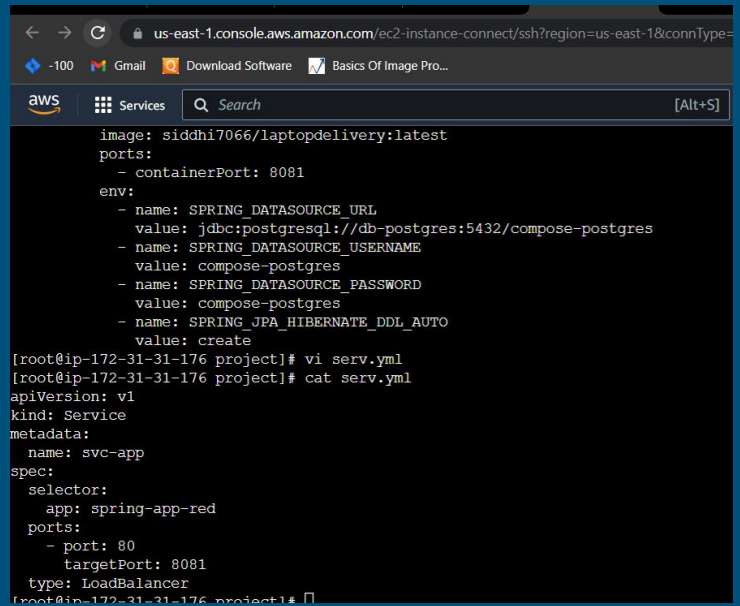


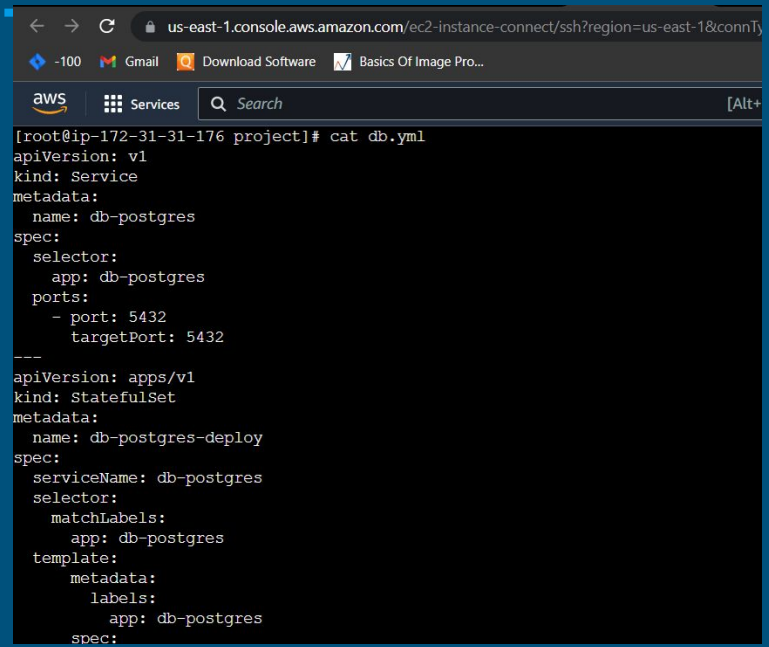
Built image from dockerhub

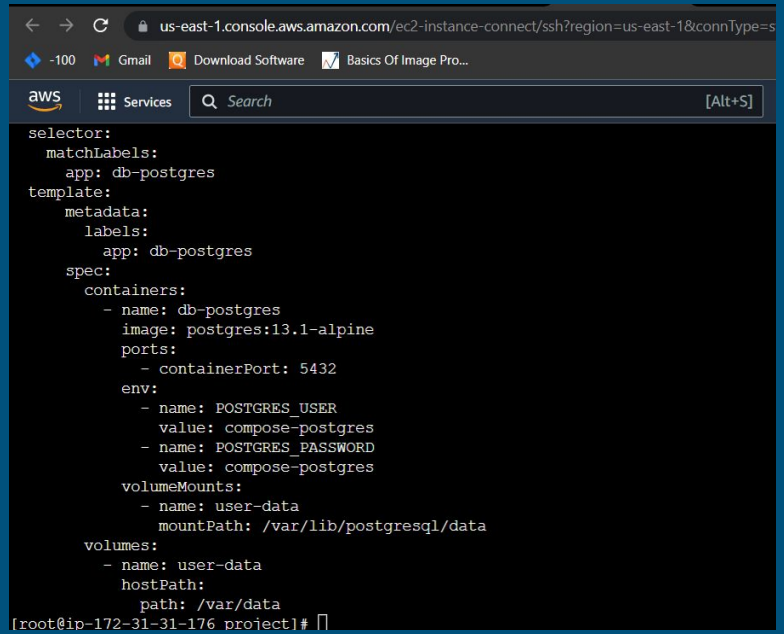


Created manifest files (3 yml files - db, deployment, service (Load Balancer))

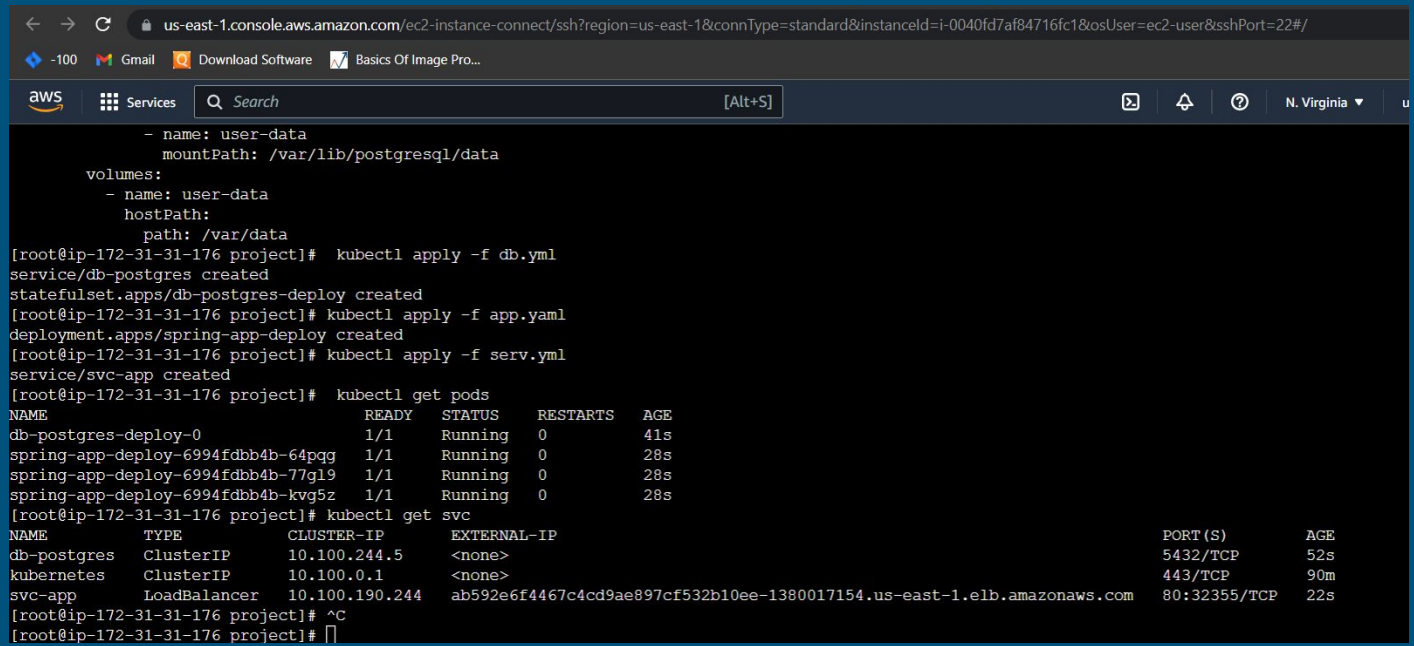





us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=us-east-1&connTy
-100
Gmail
Download Software
Basics Of Image Pro...
aws
Services
Search
[Alt+
[root@ip-172-31-31-176 project]# cat db.yml
apiVersion: v1
kind: Service
metadata:
name: db-postgres
spec:
selector:
app: db-postgres
ports:
- port: 5432
targetPort: 5432
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
name: db-postgres-deploy
spec:
serviceName: db-postgres
selector:
matchLabels:
app: db-postgres
template:
metadata:
labels:
app: db-postgres
spec:


us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=us-east-1&connType=s
-100
Gmail
Download Software
Basics Of Image Pro...
aws
Services
Search
[Alt+S]
selector:
matchLabels:
app: db-postgres
template:
metadata:
labels:
app: db-postgres
spec:
containers:
- name: db-postgres
image: postgres:13.1-alpine
ports:
- containerPort: 5432
env:
- name: POSTGRES_USER
value: compose-postgres
- name: POSTGRES_PASSWORD
value: compose-postgres
volumeMounts:
- name: user-data
mountPath: /var/lib/postgresql/data
volumes:
- name: user-data
hostPath:
path: /var/data
[root@ip-172-31-31-176 project]#

Ran all three files and viewed pods and service



Copy the Load Balancer service address and view in Browser -

http://ab592e6f4467c4cd9ae897cf532b10ee-1380017154.us-east-1.elb.amazonaws.com/swagger-ui/