

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:

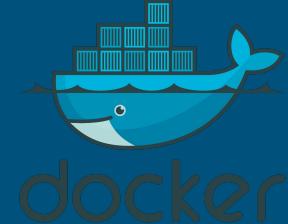
1. Dependency: While deploying the application and scaling up resources according to the requirement, we need to consider all the application dependencies and make sure that all dependencies for the application are available with respective versions.
2. High Availability: When users are increased our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

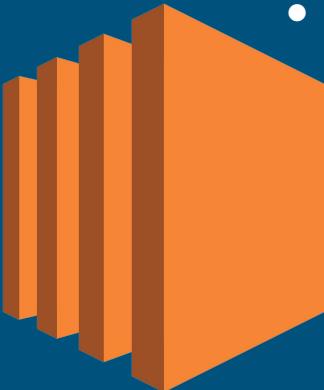
Based on the problem statement, we have used the following to ensure dependency and high availability in our deployment:

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)



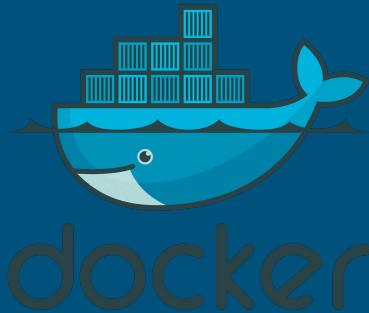
Amazon EC2

- 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.

```
D:\>cd D:\capgspring2jar

D:\capgspring2jar>git init
Initialized empty Git repository in D:/capgspring2jar/.git/

D:\capgspring2jar>git add .

D:\capgspring2jar>git commit -m "spring2 jar file"
[master (root-commit) 366acb1] spring2 jar file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 employelaptopdeliverytracking-0.0.1-SNAPSHOT.jar

D:\capgspring2jar>git remote add origin https://github.com/siddhikhanvilkar/spring2capg

D:\capgspring2jar>git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 37.53 MiB | 5.50 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/siddhikhanvilkar/spring2capg
 * [new branch]      master -> master

D:\capgspring2jar>
```

Deployment on AWS-Docker

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

The screenshot shows the AWS EC2 Instances page. The instance summary for i-08f5baee7114cf958 (docker-grp3) is displayed. Key details include:

- Instance ID: i-08f5baee7114cf958 (docker-grp3)
- Public IPv4 address: 3.211.1.86 | open address
- Private IPv4 addresses: 172.31.10.211
- Instance state: Running
- Public IPv4 DNS: ec2-3-211-1-86.compute-1.amazonaws.com | open address
- Private IP DNS name (IPv4 only): ip-172-31-10-211.ec2.internal
- Instance type: t3.medium
- Elastic IP addresses: 3.211.1.86 [Public IP]
- Auto Scaling Group name: -
- VPC ID: vpc-d9ad5da4
- Subnet ID: subnet-46d27620
- IAM Role: -

At the bottom, there are tabs for Details, Security, Networking, Storage, Status checks, Monitoring, and Tags.

The screenshot shows the AWS EC2 Instances page for the same instance. The Security tab is selected. Key details include:

- EC2 Dashboard, EC2 Global View, Events, Tags, Limits, Instances, Images, and Images tabs are visible on the left.
- Details, Security, Networking, Storage, Status checks, Monitoring, and Tags tabs are at the top.
- Security details: IAM Role (Owner ID: 154997212670), Security groups (sg-010d4d2fad9afa745 (launch-wizard-1)).
- Inbound rules table:

Security group rule ID	Port range	Protocol	Source	Security groups
sgr-08024a51172149f6d	All	All	0.0.0.0/0	launch-wizard-1
- Outbound rules table:

Security group rule ID	Port range	Protocol	Destination	Security groups
sgr-0109116cc670197ea	All	All	0.0.0.0/0	launch-wizard-1

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

```
ubuntu@ip-172-31-10-211:~$ sudo -s
root@ip-172-31-10-211:/home/ubuntu# snap install docker
docker 20.10.17 from Canonical✓ installed
root@ip-172-31-10-211:/home/ubuntu# docker pull postgres
Using default tag: latest
latest: Pulling from library/postgres
e9995326b091: Pull complete
a0cb03f17886: Pull complete
bb26f7e78134: Pull complete
c8e073b7ae91: Pull complete
99b5b1679915: Pull complete
55c520fc03c5: Pull complete
d0ac84d6672c: Pull complete
4efffb95d5849: Pull complete
c3d601b3e75e: Pull complete
85319d21ebc8: Pull complete
3d0e1f33e8d6: Pull complete
591cc4f15f86: Pull complete
96b3fb259229: Pull complete
Digest: sha256:9eb2589e67e69daf321fa95ae40e7509ce08bb1ef90d5a27a0775aa88ee0c704
Status: Downloaded newer image for postgres:latest
docker.io/library/postgres:latest
```

```
root@ip-172-31-10-211:/home/ubuntu# mkdir laptoptracking
root@ip-172-31-10-211:/home/ubuntu# cd laptoptracking
root@ip-172-31-10-211:/home/ubuntu/laptoptracking# git clone https://github.com/siddhikhanvilkar/spring2capg
Cloning into 'spring2capg'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 37.53 MiB / 14.79 MiB/s, done.
root@ip-172-31-10-211:/home/ubuntu/laptoptracking# ls
spring2capg
root@ip-172-31-10-211:/home/ubuntu/laptoptracking# cd spring2capg
root@ip-172-31-10-211:/home/ubuntu/laptoptracking/spring2capg# vi Dockerfile
root@ip-172-31-10-211:/home/ubuntu/laptoptracking/spring2capg# cat Dockerfile
FROM openjdk:8
ADD /*.jar employeelaptopdeliverytracking-0.0.1-SNAPSHOT.jar
EXPOSE 8081
ENTRYPOINT ["java", "-jar","/employeelaptopdeliverytracking-0.0.1-SNAPSHOT.jar"]
root@ip-172-31-10-211:/home/ubuntu/laptoptracking/spring2capg# ls
Dockerfile employeelaptopdeliverytracking-0.0.1-SNAPSHOT.jar
```

Dockerfile , docker-compose file:-

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

~
~
```

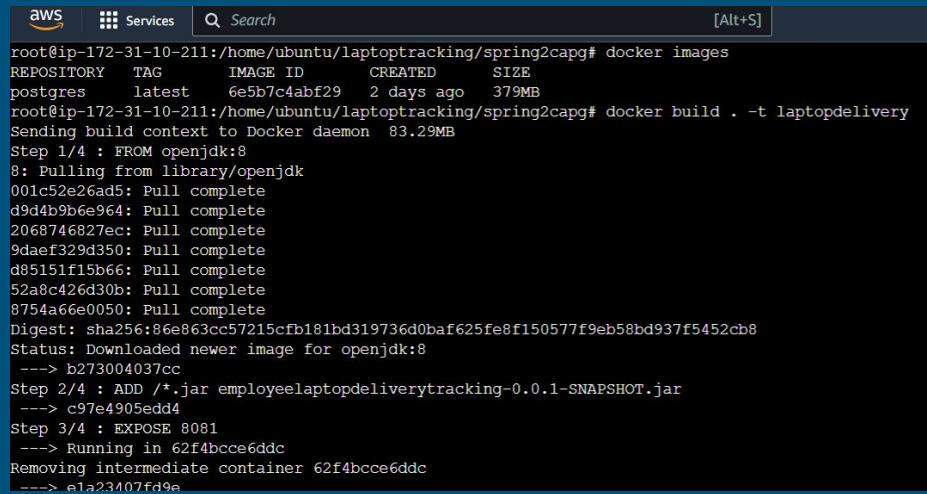
Dockerfile

```
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
```

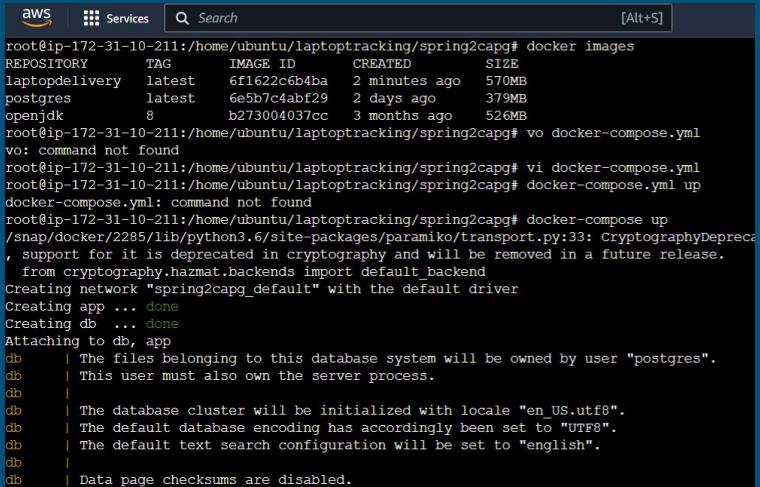
docker-compose.yml

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.



```
aws Services Search [Alt+S]
root@ip-172-31-10-211:/home/ubuntu/laptoptracking/spring2capg# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
postgres latest 6e5b7c4abf29 2 days ago 379MB
root@ip-172-31-10-211:/home/ubuntu/laptoptracking/spring2capg# docker build . -t laptopdelivery
Sending build context to Docker daemon 83.29MB
Step 1/4 : FROM openjdk:8
Step 2/4 : ADD /*.jar employeelaptopdeliverytracking-0.0.1-SNAPSHOT.jar
Step 3/4 : EXPOSE 8081
Step 4/4 : CMD ["java", "-Dserver.port=8081", "-jar", "laptopdeliverytracking-0.0.1-SNAPSHOT.jar"]
Removing intermediate container 62f4bcce6ddc
--> e1a23407fd9e
```



```
aws Services Search [Alt+S]
root@ip-172-31-10-211:/home/ubuntu/laptoptracking/spring2capg# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
laptopdelivery latest 6f1622c6b4ba 2 minutes ago 570MB
postgres latest 6e5b7c4abf29 2 days ago 379MB
openjdk 8 b273004037cc 3 months ago 526MB
root@ip-172-31-10-211:/home/ubuntu/laptoptracking/spring2capg# vo docker-compose.yml
vo: command not found
root@ip-172-31-10-211:/home/ubuntu/laptoptracking/spring2capg# vi docker-compose.yml
root@ip-172-31-10-211:/home/ubuntu/laptoptracking/spring2capg# docker-compose.yml up
docker-compose.yml: command not found
root@ip-172-31-10-211:/home/ubuntu/laptoptracking/spring2capg# docker-compose up
/snap/docker/2285/lib/python3.6/site-packages/paramiko/transport.py:33: CryptographyDeprecationWarning: support for it is deprecated in cryptography and will be removed in a future release.
    from cryptography.hazmat.backends import default_backend
Creating network "spring2capg_default" with the default driver
Creating app ... done
Creating db ... done
Attaching to db, app
db | The files belonging to this database system will be owned by user "postgres".
db | This user must also own the server process.
db |
db | The database cluster will be initialized with locale "en_US.utf8".
db | The default database encoding has accordingly been set to "UTF8".
db | The default text search configuration will be set to "english".
db |
db | Data page checksums are disabled.
```

Using Public IP address to access application:

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

The screenshot shows a web browser displaying the Swagger UI interface for an API. The URL in the address bar is `http://3.211.1.86:8081/swagger-ui/`. The page title is "Api Documentation 1.0 OAS3". Below the title, there are links for "Api Documentation", "Terms of service", and "Apache 2.0". A "Servers" section shows a dropdown menu with the option `http://3.211.1.86:8081 - Inferred Url`. The main content area lists four entities: "Address Entity" (Address), "Complaints Entity" (Complaints), "DeliveryPerson Entity" (Delivery Person), and "Device Entity" (Device). Each entity entry has a right-pointing arrow icon at the end of its line.

Deployment Using AWS-EKS

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

The screenshot shows the AWS EC2 Instances page. The instance summary for 'i-0040fd7af84716fc1 (eks-grp3)' is displayed. Key details include:

- Instance ID: i-0040fd7af84716fc1 (eks-grp3)
- Public IPv4 address: 18.215.215.49 | open address
- Private IP addresses: 172.31.31.176
- Public IPv4 DNS: ec2-18-215-215-49.compute-1.amazonaws.com | open address
- Instance state: Running
- Hostname type: ip-172-31-31-176.ec2.internal
- Private IP DNS name (IPv4 only): ip-172-31-31-176.ec2.internal
- Answer private resource DNS name: IPV4 (A)
- Instance type: t2.micro
- VPC ID: vpc-d9ad5da4
- Elastic IP addresses: 18.215.215.49 [Public IP]
- AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations.
- IAM Role: -
- Subnet ID: subnet-5f82f612
- Auto Scaling Group name: -

The screenshot shows the AWS Security Groups page. The security group 'sg-065d0f8df0e882e71 (launch-wizard-2)' is selected. The 'Inbound rules' section shows one rule:

Security group rule ID	Port range	Protocol	Source	Security groups
sgr-02e2e599fb7766c7f	All	All	0.0.0.0/0	launch-wizard-2

The 'Outbound rules' section shows one rule:

Security group rule ID	Port range	Protocol	Destination	Security groups
sgr-0990eed52da8454e4	All	All	0.0.0.0/0	launch-wizard-2

Created Access Key from Users

The screenshot shows the AWS Identity and Access Management (IAM) console. A modal dialog box is open, titled "Create access key". Inside the dialog, there is a red-bordered "Warning" section containing the text: "Never post your secret access key on public platforms, such as GitHub. This can compromise your account security." Below this, a green-bordered "Success" section contains the text: "This is the **only** time that the secret access keys can be viewed or downloaded. You cannot recover them later. However, you can create new access keys at any time." At the bottom of the dialog, there is a "Download .csv file" button and a "Close" button. In the background, the main IAM dashboard shows a list of users with their access keys. One user, "unextcap86b8aws09", has two active access keys listed under the "Status" column.

Identity and Access Management (IAM)

Dashboard

Access management

Users

Roles

Policies

Identity providers

Account settings

Access reports

Access analyzer

Archive rules

Analyzers

Settings

Credential report

Organization activity

Create access key

Global unextcap86b8aws09 @ 1349-9721-2670

SSH key ID Uploaded Status

No results

Feedback Looking for language selection? Find it in the new Unified Settings

© 2022, Amazon Internet Services Private Ltd. or its affiliates. Privacy Terms Cookie preferences

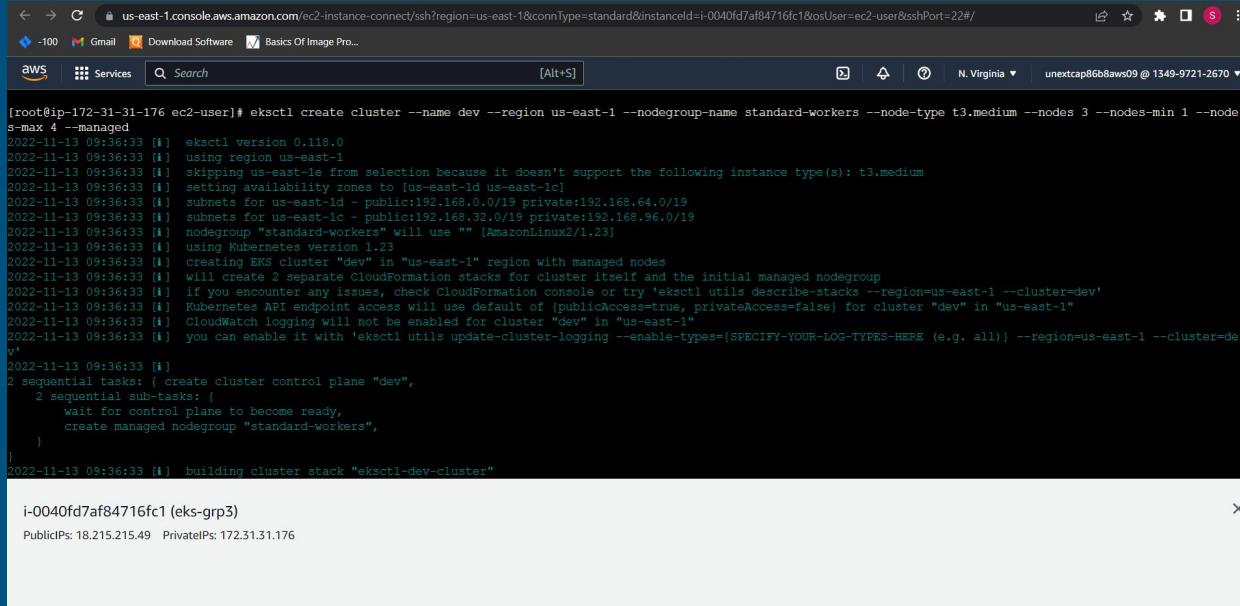
Show all

Connected to instance and configured AWS for using EKS

The screenshot shows a terminal session on a Linux system connected via SSH. The user has run several commands to configure AWS and EKS:

```
[root@ip-172-31-31-176 ec2-user]# which aws
/bin/aws
[root@ip-172-31-31-176 ec2-user]# sudo ./aws/install --bin-dir /usr/bin --install-dir /usr/bin/aws-cli --update
You can now run: /usr/bin/aws --version
[root@ip-172-31-31-176 ec2-user]# aws --version
aws-cli/2.8.12 Python/3.9.11 Linux/5.10.144-127.601.amzn2.x86_64 exe/x86_64.amzn.2 prompt/off
[root@ip-172-31-31-176 ec2-user]# aws configure
AWS Access Key ID [None]: AKIAR63TV7X7KGUUIOSF
AWS Secret Access Key [None]: 6/Yj2r3hyGkJO2676g4WJrhk7TaGDg zgLhApa0g3
Default region name [None]: us-east-1
Default output format [None]: json
[root@ip-172-31-31-176 ec2-user]# curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.16.8/2020-04-16/bin/linux/amd64/kubectl
% Total    % Received % Xferd  Average Speed   Time     Time      Current
          Dload  Upload Total Spent   Left Speed
100 55.7M  100 55.7M    0     0  3265k      0:00:17  0:00:17 ---:--- 5151k
[root@ip-172-31-31-176 ec2-user]# chmod +x ./kubectl
[root@ip-172-31-31-176 ec2-user]# mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin
[root@ip-172-31-31-176 ec2-user]# kubectl version --short --client
Client Version: v1.16.8-eks-e16311
[root@ip-172-31-31-176 ec2-user]# curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
[root@ip-172-31-31-176 ec2-user]# sudo mv /tmp/eksctl /usr/bin
[root@ip-172-31-31-176 ec2-user]# eksctl version
0.118.0
[root@ip-172-31-31-176 ec2-user]# eksctl help
The official CLI for Amazon EKS
```

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



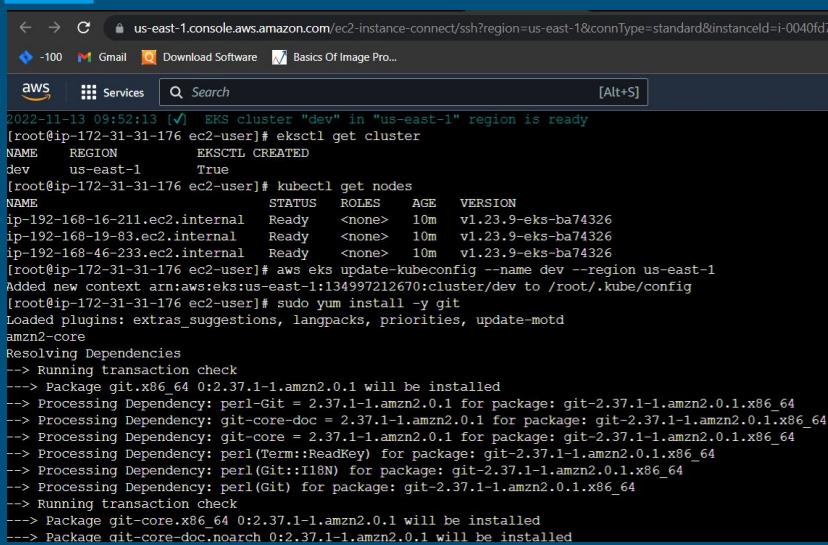
The screenshot shows a terminal window on the AWS CloudWatch interface. The user is running the command `eksctl create cluster --name dev --region us-east-1 --nodegroup-name standard-workers --node-type t3.medium --nodes 3 --nodes-min 1 --node-s-max 4 --managed`. The output of the command is displayed, detailing the creation of the cluster, including the selection of the us-east-1 region, the creation of a managed nodegroup named "standard-workers" using the t3.medium instance type, and the configuration of Kubernetes version 1.23. The command also creates two CloudFormation stacks for the cluster and its initial managed nodegroup. It notes that Kubernetes API endpoint access will use default public IP ranges and that CloudWatch logging is disabled. A note at the end suggests enabling CloudWatch logging with the command `eksctl utils update-cluster-logging --enable-types=(SPECIFY-YOUR-LOG-TYPES-HERE) --region=us-east-1 --cluster=dev`.

```
[root@ip-172-31-31-176 ec2-user]# eksctl create cluster --name dev --region us-east-1 --nodegroup-name standard-workers --node-type t3.medium --nodes 3 --nodes-min 1 --node-s-max 4 --managed
2022-11-13 09:36:33 [ℹ] eksctl version 0.118.0
2022-11-13 09:36:33 [ℹ] using region us-east-1
2022-11-13 09:36:33 [ℹ] skipping us-east-1e from selection because it doesn't support the following instance type(s): t3.medium
2022-11-13 09:36:33 [ℹ] setting availability zones to [us-east-1d us-east-1c]
2022-11-13 09:36:33 [ℹ] subnet for us-east-1d - public:192.168.0.0/19 private:192.168.64.0/19
2022-11-13 09:36:33 [ℹ] subnet for us-east-1c - public:192.168.32.0/19 private:192.168.96.0/19
2022-11-13 09:36:33 [ℹ] nodegroup "standard-workers" will use "" [AmazonLinux2/1.23]
2022-11-13 09:36:33 [ℹ] using Kubernetes version 1.23
2022-11-13 09:36:33 [ℹ] creating EKS cluster "dev" in "us-east-1" region with managed nodes
2022-11-13 09:36:33 [ℹ] will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2022-11-13 09:36:33 [ℹ] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1 --cluster=dev'
2022-11-13 09:36:33 [ℹ] Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "dev" in "us-east-1"
2022-11-13 09:36:33 [ℹ] CloudWatch logging will not be enabled for cluster "dev" in "us-east-1"
2022-11-13 09:36:33 [ℹ] you can enable it with 'eksctl utils update-cluster-logging --enable-types=(SPECIFY-YOUR-LOG-TYPES-HERE) (e.g. all) --region=us-east-1 --cluster=dev'
2022-11-13 09:36:33 [ℹ]
2 sequential tasks: { create cluster control plane "dev",
  2 sequential sub-tasks: {
    wait for control plane to become ready,
    create managed nodegroup "standard-workers",
  }
}
2022-11-13 09:36:33 [ℹ] building cluster stack "eksctl-dev-cluster"
```

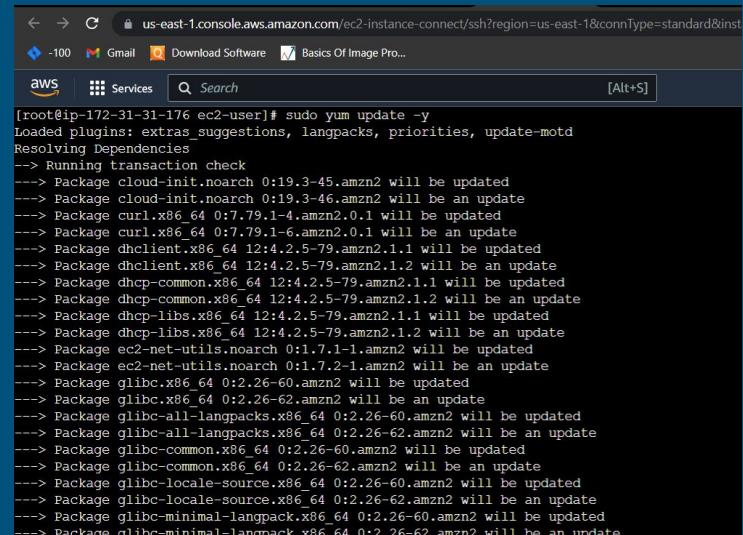
i-0040fd7af84716fc1 (eks-grp3)

PublicIPs: 18.215.215.49 PrivateIPs: 172.31.31.176

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



```
2022-11-13 09:52:13 [V] EKS cluster "dev" in "us-east-1" region is ready
[root@ip-172-31-31-176 ec2-user]# eksctl get cluster
NAME      REGION      EKSCTL CREATED
dev       us-east-1   True
[root@ip-172-31-31-176 ec2-user]# kubectl get nodes
NAME           STATUS    ROLES   AGE     VERSION
ip-192-168-16-211.ec2.internal   Ready    <none>  10m    v1.23.9-eks-ba74326
ip-192-168-19-83.ec2.internal   Ready    <none>  10m    v1.23.9-eks-ba74326
ip-192-168-46-233.ec2.internal   Ready    <none>  10m    v1.23.9-eks-ba74326
[root@ip-172-31-31-176 ec2-user]# aws eks update-kubeconfig --name dev --region us-east-1
Added new context arn:aws:eks:us-east-1:134997212670:cluster/dev to /root/.kube/config
[root@ip-172-31-31-176 ec2-user]# sudo yum install -y git
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core
Resolving Dependencies
--> Running transaction check
--> Package git.x86_64 0:2.37.1-1.amzn2.0.1 will be installed
--> Processing Dependency: perl-Git = 2.37.1-1.amzn2.0.1 for package: git-2.37.1-1.amzn2.0.1.x86_64
--> Processing Dependency: git-core-doc = 2.37.1-1.amzn2.0.1 for package: git-2.37.1-1.amzn2.0.1.x86_64
--> Processing Dependency: git-core = 2.37.1-1.amzn2.0.1 for package: git-2.37.1-1.amzn2.0.1.x86_64
--> Processing Dependency: perl(Term::ReadKey) for package: git-2.37.1-1.amzn2.0.1.x86_64
--> Processing Dependency: perl(Git) for package: git-2.37.1-1.amzn2.0.1.x86_64
--> Running transaction check
--> Package git-core.x86_64 0:2.37.1-1.amzn2.0.1 will be installed
--> Package git-core-doc.noarch 0:2.37.1-1.amzn2.0.1 will be installed
```



```
[root@ip-172-31-31-176 ec2-user]# sudo yum update -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
--> Package cloud-init.noarch 0:19.3-45.amzn2 will be updated
--> Package cloud-init.noarch 0:19.3-46.amzn2 will be an update
--> Package curl.x86_64 0:7.79.1-4.amzn2.0.1 will be updated
--> Package curl.x86_64 0:7.79.1-6.amzn2.0.1 will be an update
--> Package dhclient.x86_64 1:24.2.5-79.amzn2.1.1 will be updated
--> Package dhclient.x86_64 1:24.2.5-79.amzn2.1.2 will be an update
--> Package dhcp-common.x86_64 1:24.2.5-79.amzn2.1.1 will be updated
--> Package dhcp-common.x86_64 1:24.2.5-79.amzn2.1.2 will be an update
--> Package dhcp-libs.x86_64 1:24.2.5-79.amzn2.1.1 will be updated
--> Package dhcp-libs.x86_64 1:24.2.5-79.amzn2.1.2 will be an update
--> Package ec2-net-utils.noarch 0:1.7.1-1.amzn2 will be updated
--> Package ec2-net-utils.noarch 0:1.7.2-1.amzn2 will be an update
--> Package glibc.x86_64 0:2.26-60.amzn2 will be updated
--> Package glibc.x86_64 0:2.26-62.amzn2 will be an update
--> Package glibc-all-langpacks.x86_64 0:2.26-60.amzn2 will be updated
--> Package glibc-all-langpacks.x86_64 0:2.26-62.amzn2 will be an update
--> Package glibc-common.x86_64 0:2.26-62.amzn2 will be updated
--> Package glibc-common.x86_64 0:2.26-60.amzn2 will be updated
--> Package glibc-locale-source.x86_64 0:2.26-62.amzn2 will be an update
--> Package glibc-locale-source.x86_64 0:2.26-60.amzn2 will be updated
--> Package glibc-minimal-langpack.x86_64 0:2.26-60.amzn2 will be updated
--> Package glibc-minimal-langpack.x86_64 0:2.26-62.amzn2 will be an update
```

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

```
[root@ip-172-31-31-176 spring2capg]# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have
Username: siddhi7066
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Docker hub repository example

The screenshot shows a Docker Hub repository page for the user siddhi7066 and the repository name laptopdelivery. The URL in the browser bar is hub.docker.com/repository/docker/siddhi7066/laptopdelivery.

Repository Information:

- Name:** siddhi7066 / laptopdelivery
- Description:** This repository does not have a description.
- Last pushed:** a day ago
- Docker commands:** docker push siddhi7066/laptopdelivery:tagname
- Public View:** A button to view the repository publicly.

Tags and scans:

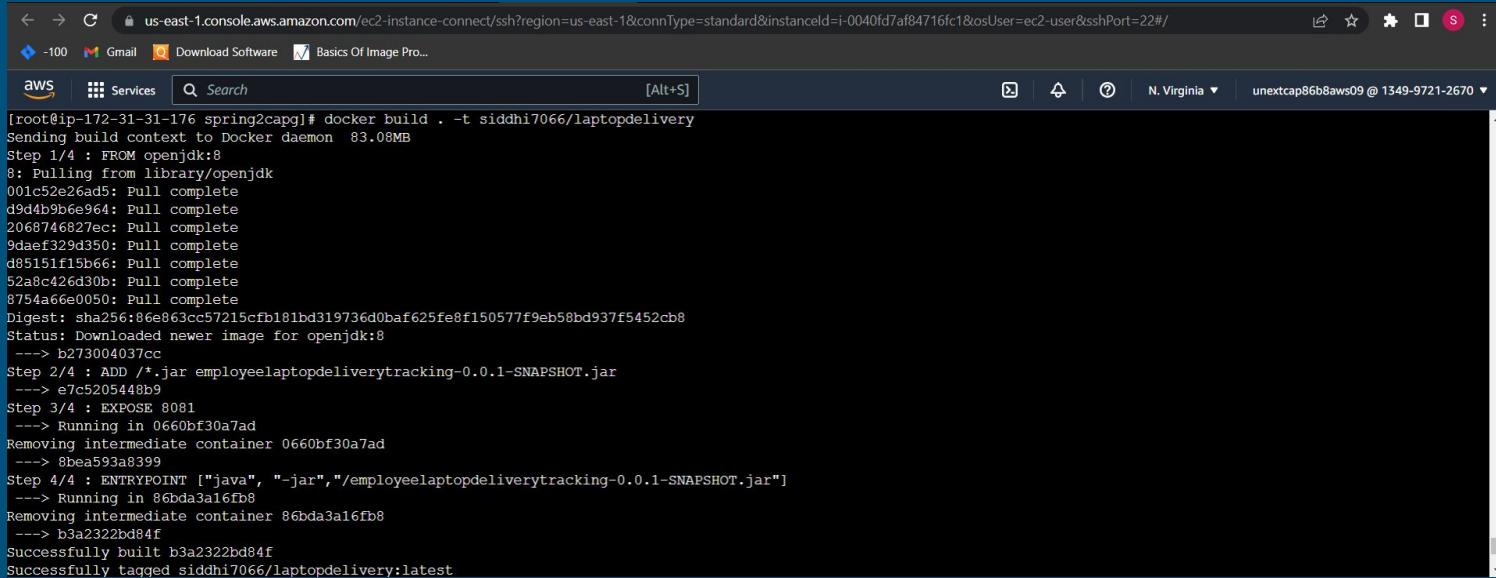
- This repository contains 1 tag(s).
- Tag:** latest
- OS:** Alpine
- Type:** Image
- Pulled:** 5 hours ago
- Pushed:** a day ago

Vulnerability Scanning: DISABLED (Enable)

Automated Builds:

- Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.
- Available with Pro, Team and Business subscriptions.
- Upgrade** and **Learn more** buttons.

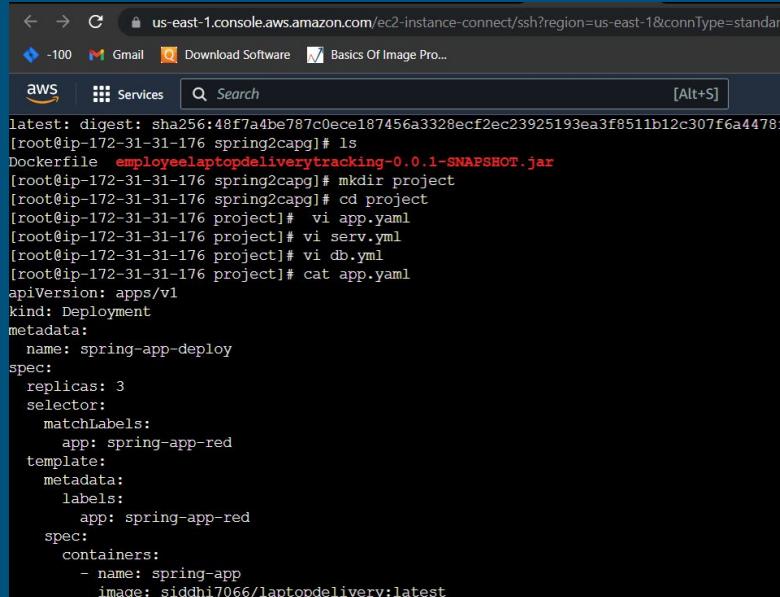
Built image from dockerhub



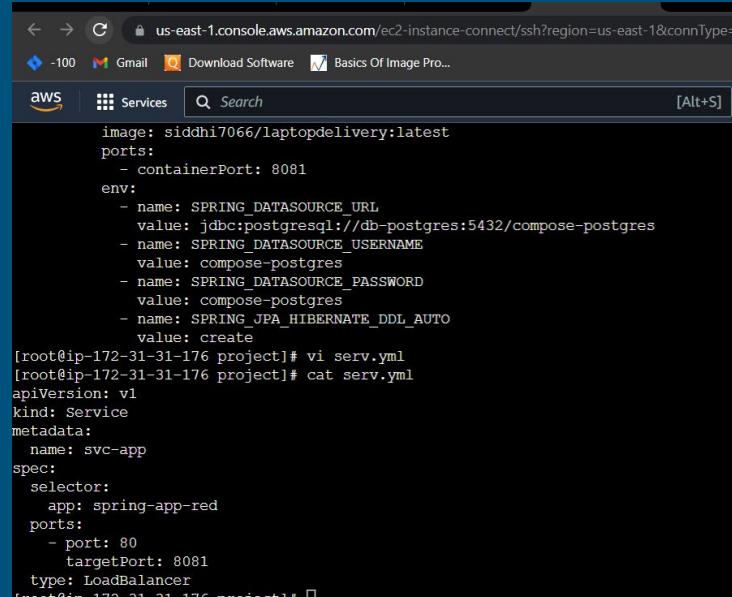
The screenshot shows a terminal window on an AWS CloudWatch instance. The URL in the address bar is `us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=us-east-1&connType=standard&instanceId=i-0040fd7af84716fc1&osUser=ec2-user&sshPort=22#/`. The terminal window displays the following Docker build command and its output:

```
[root@ip-172-31-31-176 spring2capg]# docker build . -t siddhi7066/laptopdelivery
Sending build context to Docker daemon 83.08MB
Step 1/4 : FROM openjdk:8
8: Pulling from library/openjdk
001c52e26ad5: Pull complete
d9d4b9b6e964: Pull complete
2068746827ec: Pull complete
9daef329d350: Pull complete
d85151f15b66: Pull complete
52a8c426d30b: Pull complete
8754a66e0050: Pull complete
Digest: sha256:86e863cc57215cfb181bd319736d0baf625fe8f150577f9eb58bd937f5452cb8
Status: Downloaded newer image for openjdk:8
--> b273004037cc
Step 2/4 : ADD /*.jar employeeelaptopdeliverytracking-0.0.1-SNAPSHOT.jar
--> e7c5205448b9
Step 3/4 : EXPOSE 8081
--> Running in 0660bf30a7ad
Removing intermediate container 0660bf30a7ad
--> 8bea593a8399
Step 4/4 : ENTRYPOINT ["java", "-jar","/employeeelaptopdeliverytracking-0.0.1-SNAPSHOT.jar"]
--> Running in 86bda3a16fb8
Removing intermediate container 86bda3a16fb8
--> b3a2322bd84f
Successfully built b3a2322bd84f
Successfully tagged siddhi7066/laptopdelivery:latest
```

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



```
latest: digest: sha256:48f7a4be787c0ece187456a3328ecf2ec23925193ea3f8511b12c307f6a4478f
[root@ip-172-31-31-176 ~]# ls
Dockerfile  employee-laptop-delivery-tracking-0.0.1-SNAPSHOT.jar
[root@ip-172-31-31-176 ~]# mkdir project
[root@ip-172-31-31-176 ~]# cd project
[root@ip-172-31-31-176 project]# vi app.yaml
[root@ip-172-31-31-176 project]# vi serv.yml
[root@ip-172-31-31-176 project]# vi db.yml
[root@ip-172-31-31-176 project]# cat app.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-app-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      app: spring-app-red
  template:
    metadata:
      labels:
        app: spring-app-red
    spec:
      containers:
        - name: spring-app
          image: siddhi7066/laptopdelivery:latest
```

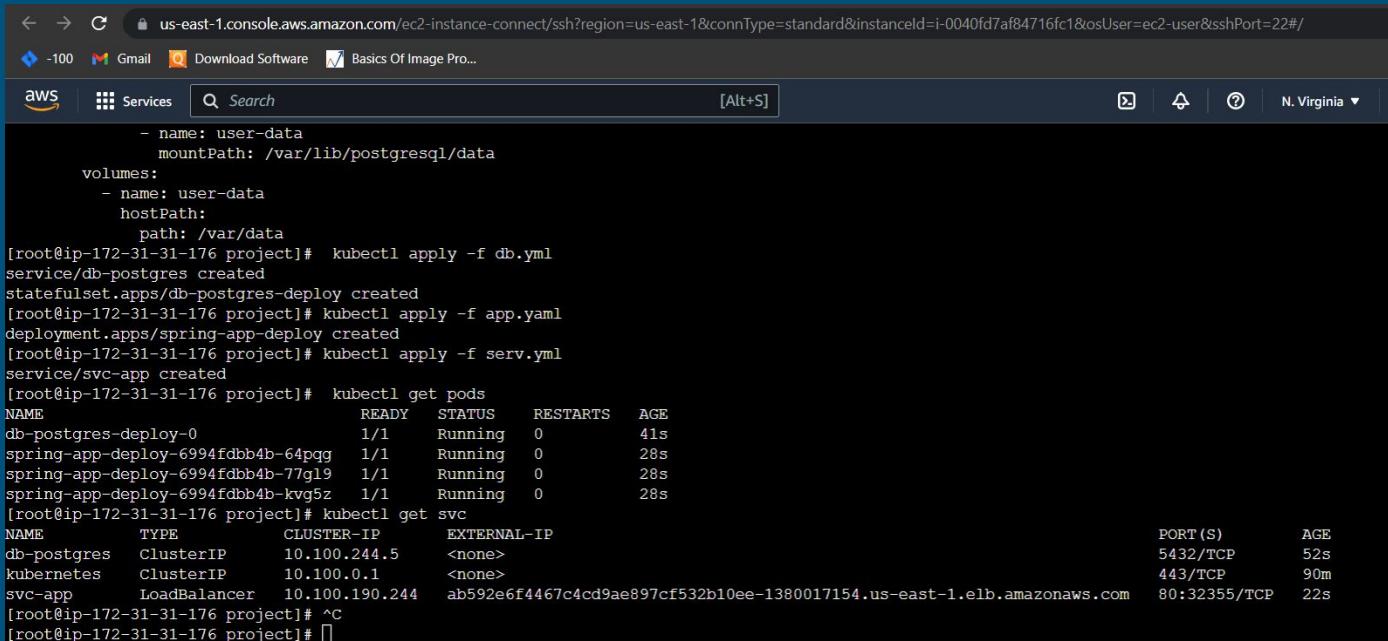


```
image: siddhi7066/laptopdelivery:latest
ports:
  - containerPort: 8081
env:
  - name: SPRING_DATASOURCE_URL
    value: jdbc:postgresql://db-postgres:5432/compose-postgres
  - name: SPRING_DATASOURCE_USERNAME
    value: compose-postgres
  - name: SPRING_DATASOURCE_PASSWORD
    value: compose-postgres
  - name: SPRING_JPA_HIBERNATE_DDL_AUTO
    value: create
[root@ip-172-31-31-176 project]# vi serv.yml
[root@ip-172-31-31-176 project]# cat serv.yml
apiVersion: v1
kind: Service
metadata:
  name: svc-app
spec:
  selector:
    app: spring-app-red
  ports:
    - port: 80
      targetPort: 8081
      type: LoadBalancer
[root@ip-172-31-31-176 project]#
```

```
[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:
```

```
aws | Services | Search [Alt+  
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=us-east-1&connType=  
- 100 Gmail Download Software Basics Of Image Pro...  
[root@ip-172-31-31-176 project]# cat db.yml  
apiVersion: v1  
kind: Service  
metadata:  
  name: db-postgres  
spec:  
  selector:  
    matchLabels:  
      app: db-postgres  
  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



The screenshot shows a terminal window within an AWS Lambda function editor. The user has run three YAML files to deploy services: db.yml, app.yml, and serv.yml. The logs output the results of these commands:

```
- name: user-data
  mountPath: /var/lib/postgresql/data
volumes:
- name: user-data
  hostPath:
    path: /var/data
[root@ip-172-31-31-176 project]# kubectl apply -f db.yml
service/db-postgres created
statefulset.apps/db-postgres-deploy created
[root@ip-172-31-31-176 project]# kubectl apply -f app.yaml
deployment.apps/spring-app-deploy created
[root@ip-172-31-31-176 project]# kubectl apply -f serv.yml
service/svc-app created
[root@ip-172-31-31-176 project]# kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
db-postgres-deploy-0   1/1     Running   0          41s
spring-app-deploy-6994fdbb4b-64pqg  1/1     Running   0          28s
spring-app-deploy-6994fdbb4b-77ql9  1/1     Running   0          28s
spring-app-deploy-6994fdbb4b-kvg5z  1/1     Running   0          28s
[root@ip-172-31-31-176 project]# kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
db-postgres   ClusterIP   10.100.244.5   <none>        5432/TCP   52s
kubernetes   ClusterIP   10.100.0.1     <none>        443/TCP    90m
svc-app     LoadBalancer 10.100.190.244  ab592e6f4467c4cd9ae897cf532b10ee-1380017154.us-east-1.elb.amazonaws.com  80:32355/TCP  22s
[root@ip-172-31-31-176 project]# ^C
[root@ip-172-31-31-176 project]# ]|
```

Copy the Load Balancer service address and view in Browser -

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

The screenshot shows a web browser window displaying the Swagger UI interface for a RESTful API. The URL in the address bar is <http://ab592e6f4467c4cd9ae897cf532b10ee-1380017154.us-east-1.elb.amazonaws.com/swagger-ui/>. The page title is "Api Documentation 1.0 OAS3". At the top, there is a "Select a definition" dropdown set to "default". Below the title, there are links for "Api Documentation", "Terms of service", and "Apache 2.0". A "Servers" section shows a single entry: "http://ab592e6f4467c4cd9ae897cf532b10ee-1380017154.us-east-1.elb.amazonaws.com:80 - Inferred Url". The main content area lists four entities: "Address Entity", "Complaints Entity", "DeliveryPerson Entity", and "Device Entity", each with a corresponding "Address", "Complaints", "Delivery Person", and "Device" link respectively.