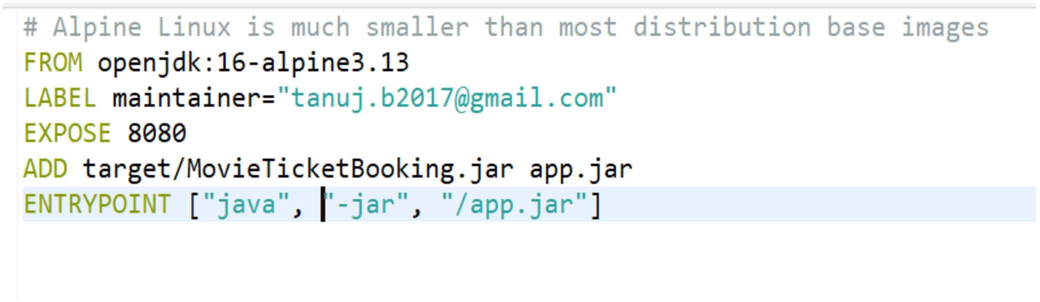
Sprint-2
MOVIE TICKET BOOKING APPLICATION

1) Docker compose:

Steps to create a Dockerfile:

1. The first line has to start with the **FROM** keyword. It tells docker, from which base image you want to base your image from. In our case, we are creating an image from the **openjdk:16**.
2. The **EXPOSE** instruction informs Docker that the container listens on the specified network ports at runtime.
3. The **ADD** instruction copies new files, directories or remote file URLs from source and adds them to the file system of the image at the path destination.
4. The **ENTRYPOINT** instruction makes your container run as an executable. The executable command for java is: [“java”, “-jar”, “jar-filename.jar”].



```
# Alpine Linux is much smaller than most distribution base images
FROM openjdk:16-alpine3.13
LABEL maintainer="tanuj.b2017@gmail.com"
EXPOSE 8080
ADD target/MovieTicketBooking.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Installation steps for docker:

1. Download Docker:
   https://docs.docker.com/desktop/windows/install/
2. Double –click Install Docker.
3. Follow the install wizard: accept the license, authorize the installer, and proceed with the installation.
4. Click finish to launch Docker.
5. Docker starts automatically.

Steps to create a docker image:

1. Open a terminal and go to the directory with the Dockerfile.
2. Now build the container image using the **docker build** command:

```
$ docker build –t <image-name> .
```

Steps to create docker-compose file:

1. At the root of the app project, create a file named **docker-compose.yml**.
2. In the compose file, we'll start off by defining the schema version.

```
version: "3.7"
```

3. Next, we'll define the list of services (or containers) we want to run as part of our application.

```
version: "3.7"

services:
```

   And now, we'll start migrating a service at a time into the compose file.
   This Compose file defines two services: app and postgresql
4. First, let's define the service entry and the image for the container.
5. Migrate the -p 9001:9001 part of the command by defining the ports for the service.
6. We will first define the new service and name it postgresql and define the ports.
7. Finally, we only need to specify the environment variables.

```
version: '3.7'
services:
  app:
    container_name: movieticketbooking
    image: movieticketbooking
    ports:
      - 8080:8080
    depends_on:
      - postgresqldb
    links:
      - postgresqldb:postgres
  postgresqldb:
    image: "postgres:latest"
    ports:
      - 5432:5432
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
```

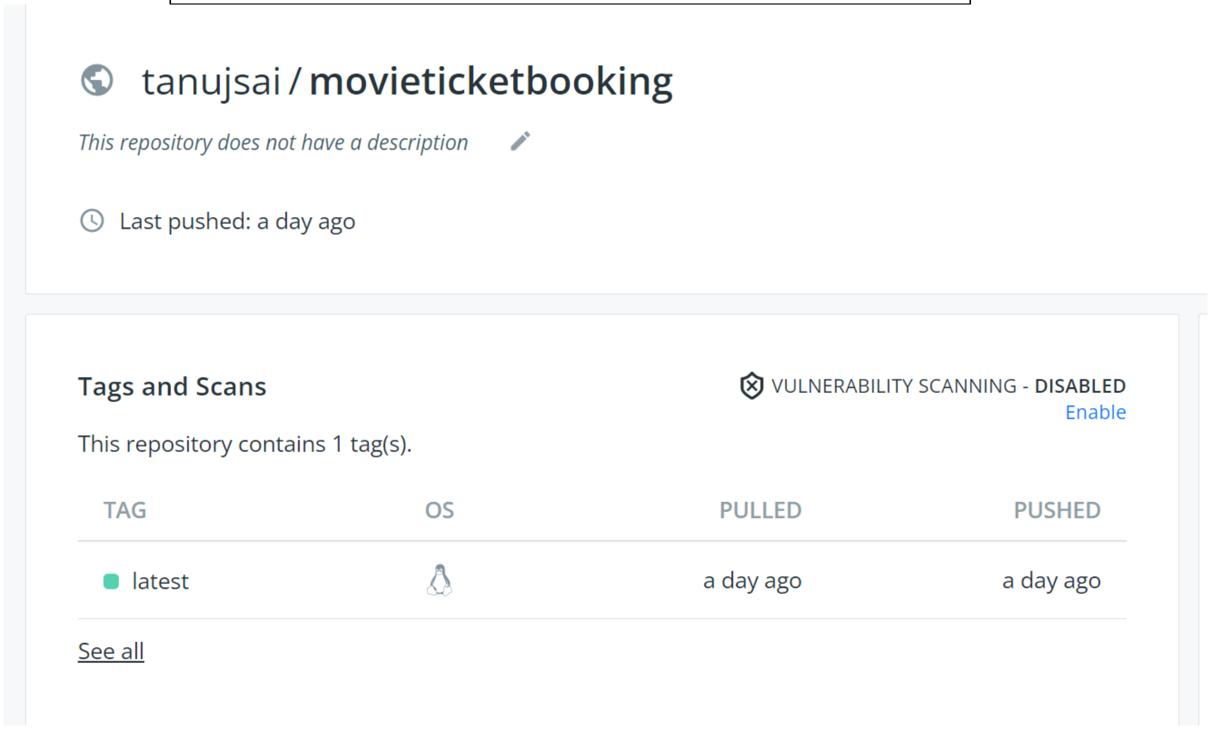
Steps to push the image onto the docker hub:

- Login to the docker hub with the username.
- Tag the image using the **docker tag** command:

```
$ docker tag <image-name> username/image-name
```

- Push the image into the docker hub using the **docker push** command:

```
$ docker push username/image-name
```



tanujsai / **movieticketbooking**

This repository does not have a description

Last pushed: a day ago

Tags and Scans

VULNERABILITY SCANNING - **DISABLED**
Enable

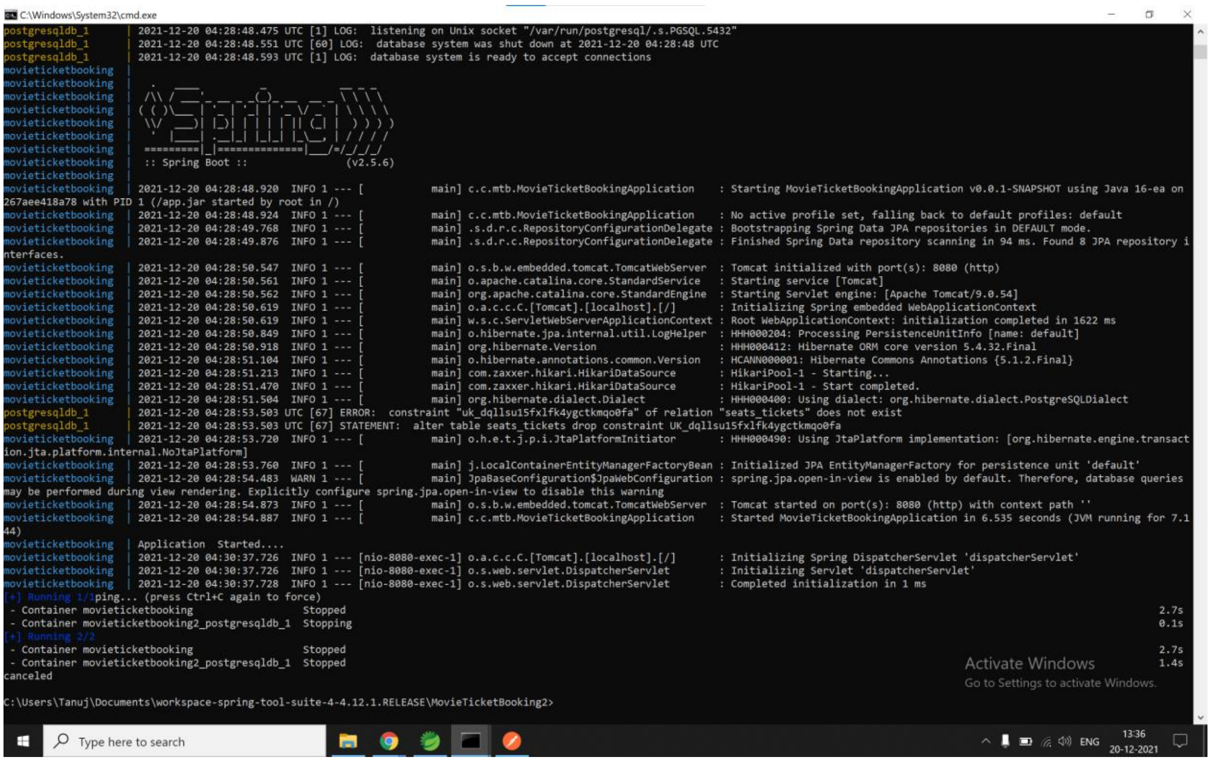
This repository contains 1 tag(s).

TAG	OS	PULLED	PUSHED
🟩 latest	🐧	a day ago	a day ago

See all

- Start up the application stack using the **docker-compose up** command.

```
$ docker-compose up
```

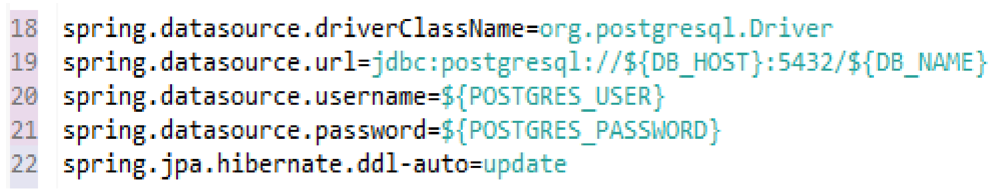


C:\Windows\System32\cmd.exe

```
postgresqldb_1       | 2021-12-20 04:28:48.475 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
postgresqldb_1       | 2021-12-20 04:28:48.551 UTC [60] LOG:  database system was shut down at 2021-12-20 04:28:48 UTC
postgresqldb_1       | 2021-12-20 04:28:48.593 UTC [1] LOG:  database system is ready to accept connections
movieticketbooking   |
movieticketbooking   |   .   ____          _            __ _ _
movieticketbooking   |  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
movieticketbooking   | ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
movieticketbooking   |  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
movieticketbooking   |   '  |____| .__|_| |_|_| |_\__, | / / / /
movieticketbooking   |  =========|_|==============|___/=/_/_/_/
movieticketbooking   |  :: Spring Boot ::                (v2.5.6)
movieticketbooking   |
movieticketbooking   | 2021-12-20 04:28:48.920  INFO 1 --- [           main] c.c.mtb.MovieTicketBookingApplication    : Starting MovieTicketBookingApplication v0.0.1-SNAPSHOT using Java 16-ea on
267aee418a78 with PID 1 (/app.jar started by root in /)
movieticketbooking   | 2021-12-20 04:28:48.924  INFO 1 --- [           main] c.c.mtb.MovieTicketBookingApplication    : No active profile set, falling back to default profiles: default
movieticketbooking   | 2021-12-20 04:28:49.768  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
movieticketbooking   | 2021-12-20 04:28:49.876  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 94 ms. Found 8 JPA repository i
nterfaces.
movieticketbooking   | 2021-12-20 04:28:50.547  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
movieticketbooking   | 2021-12-20 04:28:50.561  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
movieticketbooking   | 2021-12-20 04:28:50.562  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.54]
movieticketbooking   | 2021-12-20 04:28:50.619  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
movieticketbooking   | 2021-12-20 04:28:50.619  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1622 ms
movieticketbooking   | 2021-12-20 04:28:50.849  INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
movieticketbooking   | 2021-12-20 04:28:50.918  INFO 1 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.4.32.Final
movieticketbooking   | 2021-12-20 04:28:51.104  INFO 1 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
movieticketbooking   | 2021-12-20 04:28:51.213  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
movieticketbooking   | 2021-12-20 04:28:51.470  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
movieticketbooking   | 2021-12-20 04:28:51.504  INFO 1 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
postgresqldb_1       | 2021-12-20 04:28:53.503 UTC [67] ERROR:  constraint "uk_dqllsu15fxlfk4ygctkmqo0fa" of relation "seats_tickets" does not exist
postgresqldb_1       | 2021-12-20 04:28:53.503 UTC [67] STATEMENT:  alter table seats_tickets drop constraint UK_dqllsu15fxlfk4ygctkmqo0fa
movieticketbooking   | 2021-12-20 04:28:53.720  INFO 1 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transact
ion.jta.platform.internal.NoJtaPlatform]
movieticketbooking   | 2021-12-20 04:28:53.760  INFO 1 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
movieticketbooking   | 2021-12-20 04:28:54.483  WARN 1 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries
may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
movieticketbooking   | 2021-12-20 04:28:54.873  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
movieticketbooking   | 2021-12-20 04:28:54.887  INFO 1 --- [           main] c.c.mtb.MovieTicketBookingApplication    : Started MovieTicketBookingApplication in 6.535 seconds (JVM running for 7.1
44)
movieticketbooking   | Application  Started....
movieticketbooking   | 2021-12-20 04:30:37.726  INFO 1 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
movieticketbooking   | 2021-12-20 04:30:37.726  INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
movieticketbooking   | 2021-12-20 04:30:37.728  INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
[+] Running 1/1ping... (press Ctrl+C again to force)
 - Container movieticketbooking                  Stopped                                                                                 2.7s
 - Container movieticketbooking2_postgresqldb_1  Stopping                                                                                0.1s
[+] Running 2/2
 - Container movieticketbooking                  Stopped                                                                                 2.7s
 - Container movieticketbooking2_postgresqldb_1  Stopped                                                                                 1.4s
canceled

C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2>
```

2) Kubernetes Deployment:

Step-1 Change the application properties as the following:

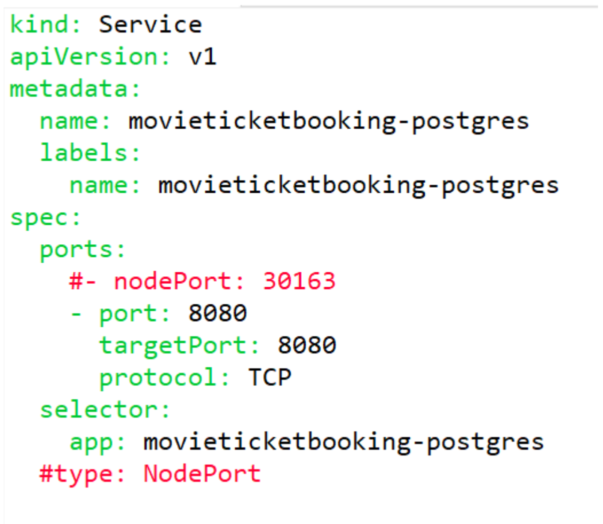


```
spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://${DB_HOST}:5432/${DB_NAME}
spring.datasource.username=${POSTGRES_USER}
spring.datasource.password=${POSTGRES_PASSWORD}
spring.jpa.hibernate.ddl-auto=update
```

Step-2 Creating manifest files:

Defining a service:

- The specification creates a new Service object named "**movieticketbooking-postgres**", which targets TCP port 9001 on any Pod with the **app= movieticketbooking-postgres** label.
- The default protocol for Services is TCP.
- Kubernetes assigns this Service an IP address which is used by the service proxies.
- The controller for the Service selector continuously scans for Pods that match its selector, and then posts any updates to an Endpoint object also named **"movieticketbooking"**



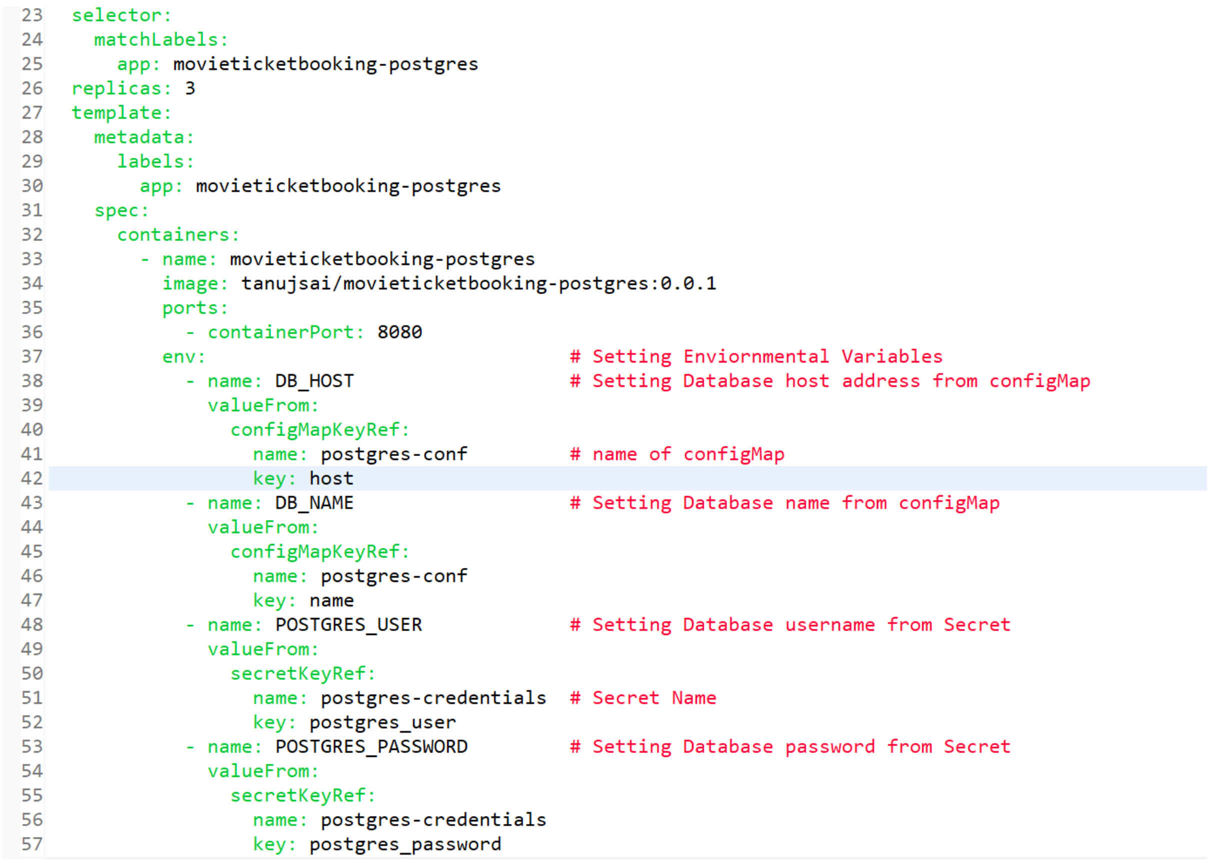
```
kind: Service
apiVersion: v1
metadata:
  name: movieticketbooking-postgres
  labels:
    name: movieticketbooking-postgres
spec:
  ports:
    #- nodePort: 30163
    - port: 8080
      targetPort: 8080
      protocol: TCP
  selector:
    app: movieticketbooking-postgres
  #type: NodePort
```

- Port definitions in Pods have names, and you can reference these names in the targetPort attribute of a Service.

Defining a deployment:

- It creates a ReplicaSet to bring up three **movieticketbooking-postgres** Pods.

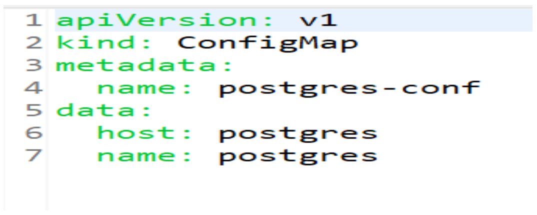
- A deployment named **movieticketbooking-postgres** is created, indicated by the **.metadata.name** field.
- The deployment creates three replicated Pods, indicated by the **.spec.replicas** field.
- The **.spec.selector** field defines how the Deployment finds which Pods to manage. In this case, you select a label that is defined in the Pod template (app: plant-nursery-postgres).
- The template field contains the following sub-fields:
- The Pods are labelled app: plant-nursery-postgres using the .**metadata.label**s field.
- The Pod template's specification, or .**template.spec field**, indicates that the Pods run one   container, plant-nursery-postgres, which runs the plant- nursery-postgres DocHub image.



```yaml
selector:
  matchLabels:
    app: movieticketbooking-postgres
replicas: 3
template:
  metadata:
    labels:
      app: movieticketbooking-postgres
  spec:
    containers:
      - name: movieticketbooking-postgres
        image: tanujsai/movieticketbooking-postgres:0.0.1
        ports:
          - containerPort: 8080
        env:                                # Setting Enviornmental Variables
          - name: DB_HOST                   # Setting Database host address from configMap
            valueFrom:
              configMapKeyRef:
                name: postgres-conf         # name of configMap
                key: host
          - name: DB_NAME                   # Setting Database name from configMap
            valueFrom:
              configMapKeyRef:
                name: postgres-conf
                key: name
          - name: POSTGRES_USER             # Setting Database username from Secret
            valueFrom:
              secretKeyRef:
                name: postgres-credentials  # Secret Name
                key: postgres_user
          - name: POSTGRES_PASSWORD         # Setting Database password from Secret
            valueFrom:
              secretKeyRef:
                name: postgres-credentials
                key: postgres_password
```

Creating a ConfigMap file:

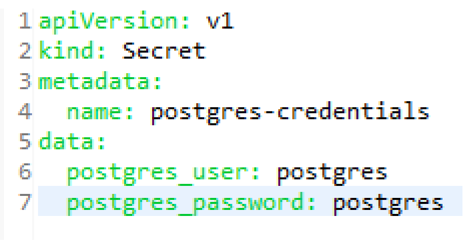
The ConfigMap configures the container(s) in Pod based on the data in the ConfigMap.



```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-conf
data:
  host: postgres
  name: postgres
```

Creating a secret file:

- A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key.
- When creating a Pod, Kubernetes automatically creates a service account Secret and automatically modifies your Pod to use this Secret.
- When using this Secret type, the data field of the Secret must contain one of the following two keys:
  - ❖ username: the user name for authentication.
  - ❖ password: the password or token for authentication.

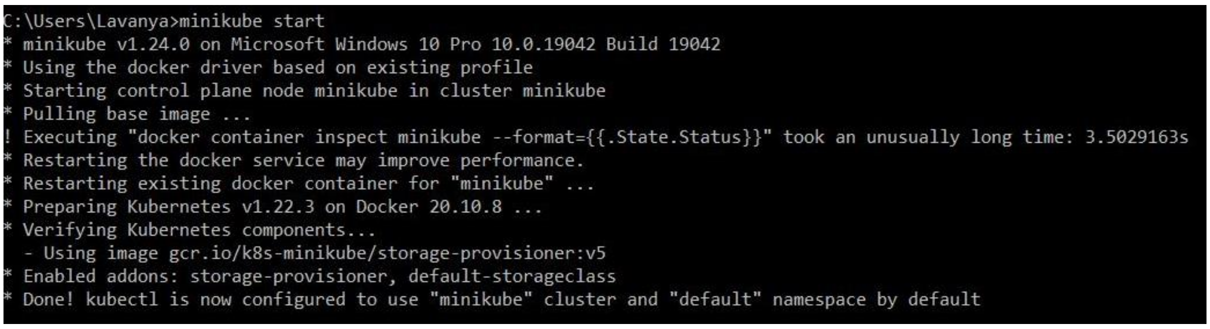


```yaml
apiVersion: v1
kind: Secret
metadata:
  name: postgres-credentials
data:
  postgres_user: postgres
  postgres_password: postgres
```

Step-3 Installation of minikube:

- Download the latest release of minikube from: https://minikube.sigs.k8s.io/docs/start/
- From a terminal with administrator access (but not logged in as root), run:

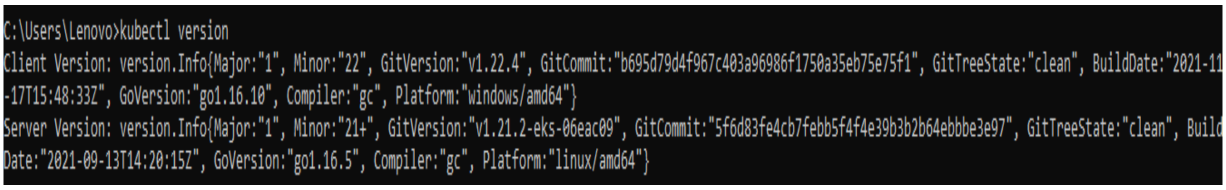
```
$ minikube start
```



```
C:\Users\Lavanya>minikube start
* minikube v1.24.0 on Microsoft Windows 10 Pro 10.0.19042 Build 19042
* Using the docker driver based on existing profile
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
! Executing "docker container inspect minikube --format={{.State.Status}}" took an unusually long time: 3.5029163s
* Restarting the docker service may improve performance.
* Restarting existing docker container for "minikube" ...
* Preparing Kubernetes v1.22.3 on Docker 20.10.8 ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

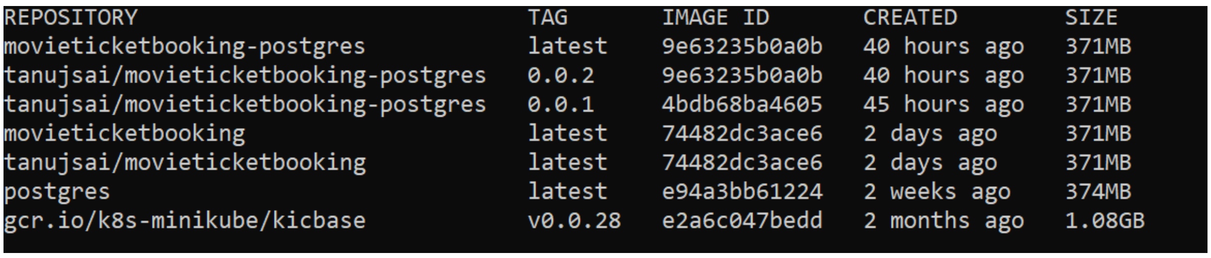
- Docker engine already provides kubectl pre-installed, so in order to check whether it is installed check for the version by using the following command:

```
$ kubectl version
```



```
C:\Users\Lenovo>kubectl version
Client Version: version.Info{Major:"1", Minor:"22", GitVersion:"v1.22.4", GitCommit:"b695d79d4f967c403a96986f1750a35eb75e75f1", GitTreeState:"clean", BuildDate:"2021-11
-17T15:48:33Z", GoVersion:"go1.16.10", Compiler:"gc", Platform:"windows/amd64"}
Server Version: version.Info{Major:"1", Minor:"21+", GitVersion:"v1.21.2-eks-06eac09", GitCommit:"5f6d83fe4cb7febb5f4f4e39b3b2b64ebbbe3e97", GitTreeState:"clean", Build
Date:"2021-09-13T14:20:15Z", GoVersion:"go1.16.5", Compiler:"gc", Platform:"linux/amd64"}
```

Step-4 Build and push the image to docker hub:



```
REPOSITORY                              TAG       IMAGE ID       CREATED         SIZE
movieticketbooking-postgres             latest    9e63235b0a0b   40 hours ago    371MB
tanujsai/movieticketbooking-postgres    0.0.2     9e63235b0a0b   40 hours ago    371MB
tanujsai/movieticketbooking-postgres    0.0.1     4bdb68ba4605   45 hours ago    371MB
movieticketbooking                      latest    74482dc3ace6   2 days ago      371MB
tanujsai/movieticketbooking             latest    74482dc3ace6   2 days ago      371MB
postgres                                latest    e94a3bb61224   2 weeks ago     374MB
gcr.io/k8s-minikube/kicbase             v0.0.28   e2a6c047bedd   2 months ago    1.08GB
```

Build and push the image into the docker hub using the above mentioned commands.

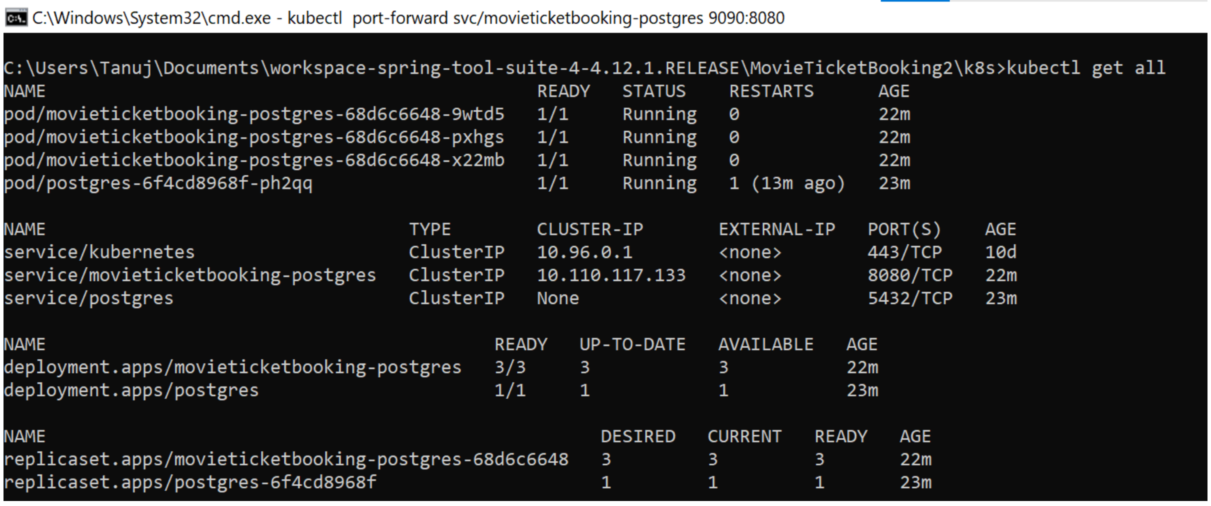
Step-5 Creating yaml files using kubectl command:

- To create a file, the following command is used:

```
$ kubectl create –f <file-name>
```

- To view all the pods, deployments and services created, we use the following command:

```
$ kubectl get all
```

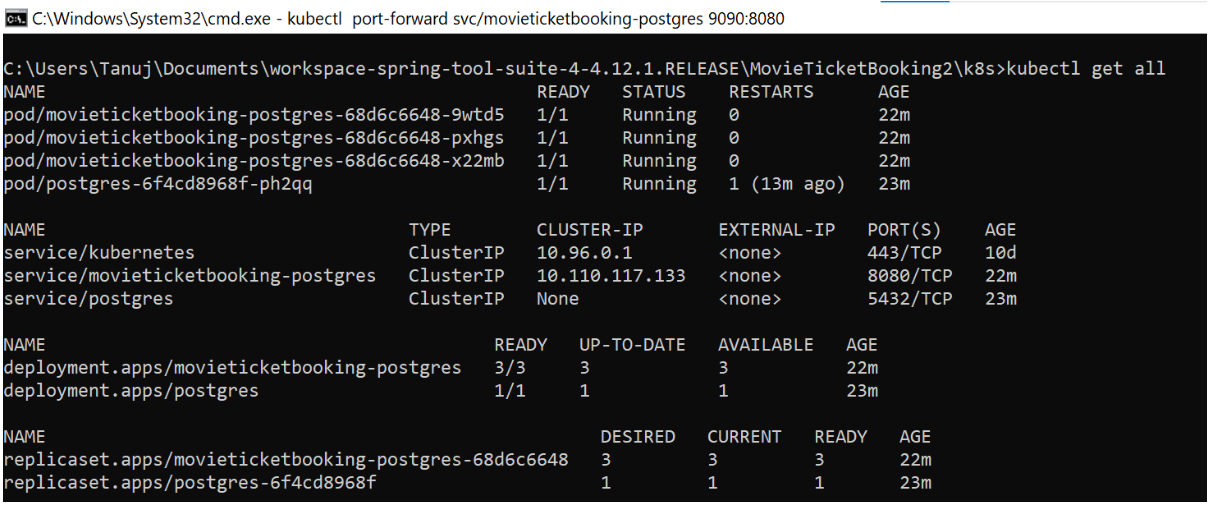


C:\Windows\System32\cmd.exe - kubectl port-forward svc/movieticketbooking-postgres 9090:8080

```
C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\k8s>kubectl get all
NAME                                           READY   STATUS    RESTARTS      AGE
pod/movieticketbooking-postgres-68d6c6648-9wtd5   1/1     Running   0             22m
pod/movieticketbooking-postgres-68d6c6648-pxhgs   1/1     Running   0             22m
pod/movieticketbooking-postgres-68d6c6648-x22mb   1/1     Running   0             22m
pod/postgres-6f4cd8968f-ph2qq                     1/1     Running   1 (13m ago)   23m

NAME                                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/kubernetes                    ClusterIP   10.96.0.1        <none>        443/TCP    10d
service/movieticketbooking-postgres   ClusterIP   10.110.117.133   <none>        8080/TCP   22m
service/postgres                      ClusterIP   None             <none>        5432/TCP   23m

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/movieticketbooking-postgres   3/3     3            3           22m
deployment.apps/postgres                      1/1     1            1           23m

NAME                                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/movieticketbooking-postgres-68d6c6648   3         3         3       22m
replicaset.apps/postgres-6f4cd8968f                     1         1         1       23m
```

- To use clusterIP, we need to use port-forward command:

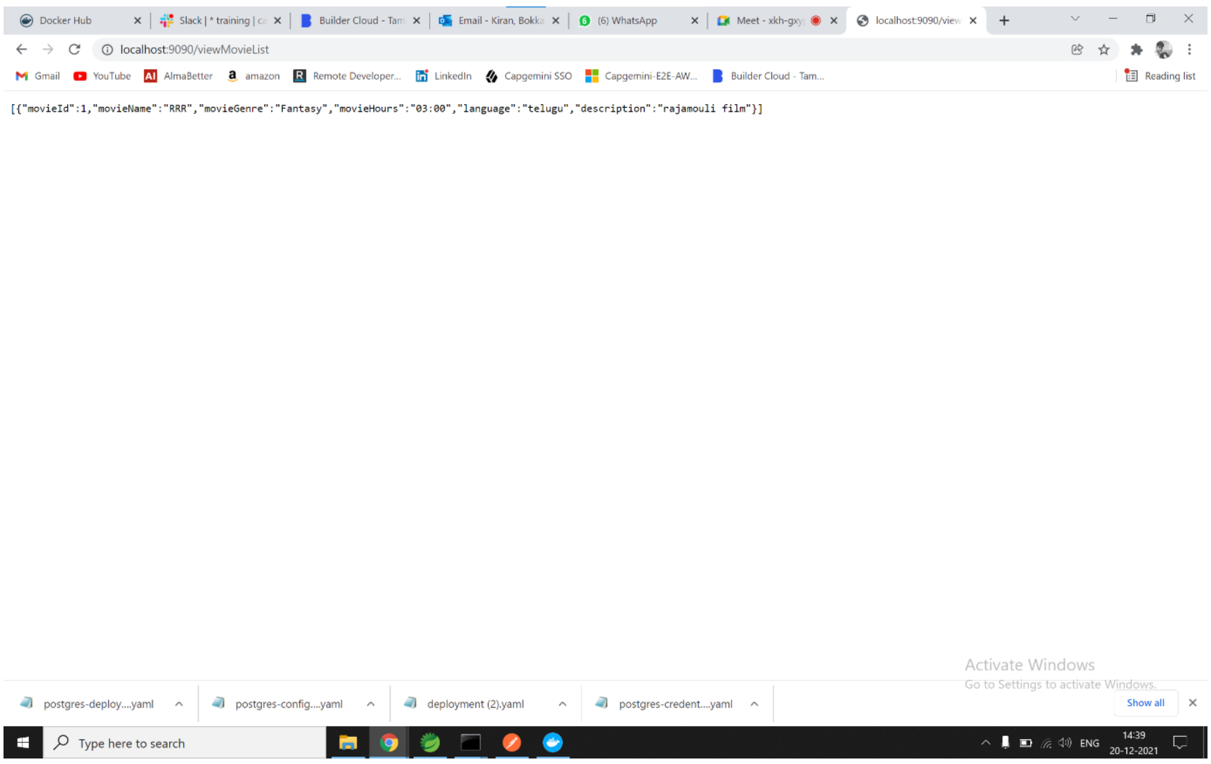
```
$ kubectl port-forward svc/image-name 9090:9001
```



C:\Windows\System32\cmd.exe - kubectl port-forward svc/movieticketbooking-postgres 9090:8080

```
C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\k8s>kubectl get all
NAME                                               READY   STATUS    RESTARTS      AGE
pod/movieticketbooking-postgres-68d6c6648-9wtd5    1/1     Running   0             22m
pod/movieticketbooking-postgres-68d6c6648-pxhgs    1/1     Running   0             22m
pod/movieticketbooking-postgres-68d6c6648-x22mb    1/1     Running   0             22m
pod/postgres-6f4cd8968f-ph2qq                      1/1     Running   1 (13m ago)   23m

NAME                                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/kubernetes                    ClusterIP   10.96.0.1        <none>        443/TCP    10d
service/movieticketbooking-postgres   ClusterIP   10.110.117.133   <none>        8080/TCP   22m
service/postgres                      ClusterIP   None             <none>        5432/TCP   23m

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/movieticketbooking-postgres   3/3     3            3           22m
deployment.apps/postgres                      1/1     1            1           23m

NAME                                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/movieticketbooking-postgres-68d6c6648   3         3         3       22m
replicaset.apps/postgres-6f4cd8968f                     1         1         1       23m
```

Browser Screenshot:

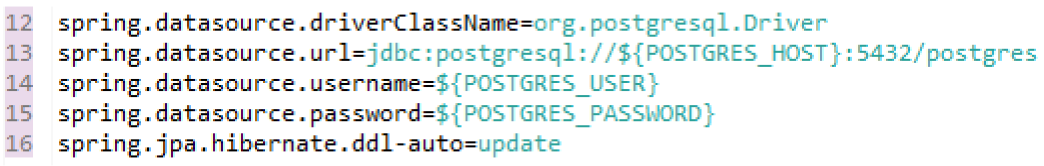


localhost:9090/viewMovieList

```
[{"movieId":1,"movieName":"RRR","movieGenre":"Fantasy","movieHours":"03:00","language":"telugu","description":"rajamouli film"}]
```

3) EKS Deployment:

Step-1 Change the application properties as the following:



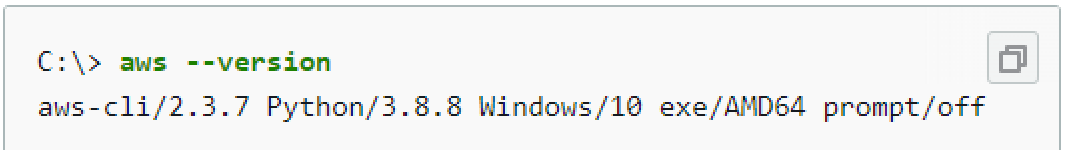
```
spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://${POSTGRES_HOST}:5432/postgres
spring.datasource.username=${POSTGRES_USER}
spring.datasource.password=${POSTGRES_PASSWORD}
spring.jpa.hibernate.ddl-auto=update
```

Step-2 Creating manifest files:

- The manifest files also known as the “yaml” files are created just like the way these files are created in the kubernetes deployment.
- These files are as the following:
    - postgres-storage.yaml
    - postgres-secrets.yaml
    - postgres-deployment.yaml
    - postgres-deployment.yaml
    - springboot-deployment.yaml
    - springboot-service.yaml

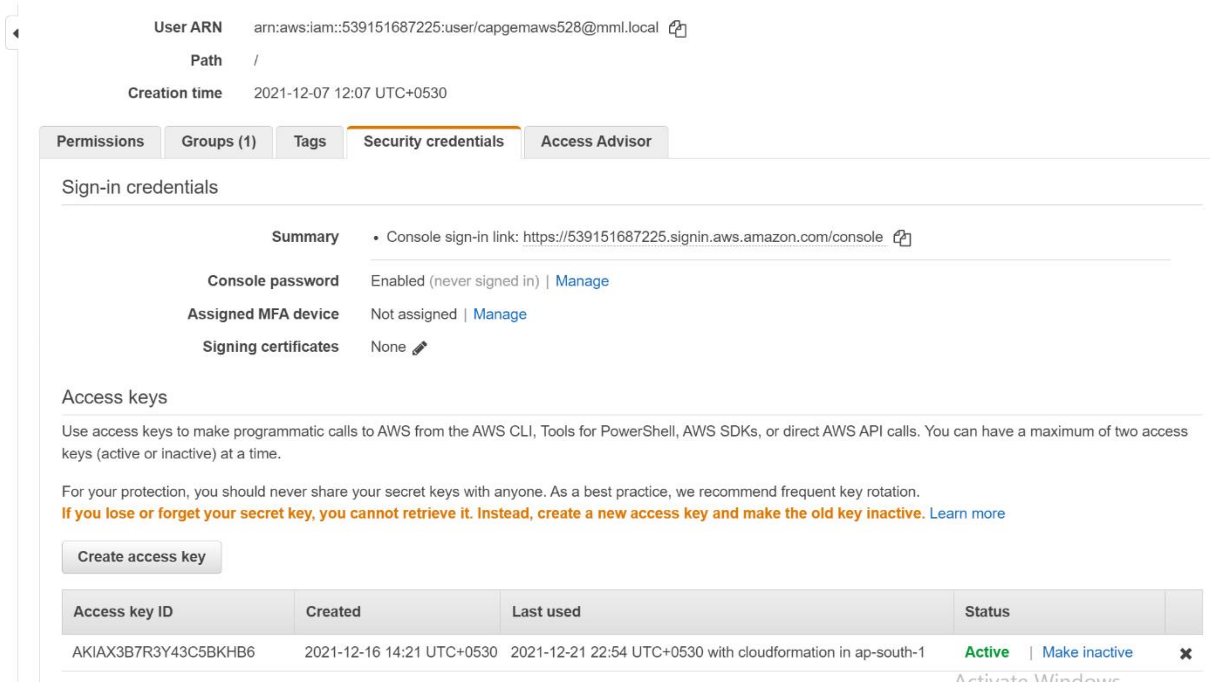
Step-3 Installation of AWS CLI:

- Download and run the AWS CLI MSI installer for Windows (64-bit)
  https://awscli.amazonaws.com/AWSCLIV2.msi
- To confirm the installation, open the **Start** menu, search for cmd to open a command prompt window, and at the command prompt use the aws --version command.



```
C:\> aws --version
aws-cli/2.3.7 Python/3.8.8 Windows/10 exe/AMD64 prompt/off
```

- We need secret keys from AWS IAM account. Go to IAM in AWS and generate access key by going into the security credentials section in users.
-
-


User ARN arn:aws:iam::539151687225:user/capgemaws528@mml.local

Path /

Creation time 2021-12-07 12:07 UTC+0530

Sign-in credentials

Summary • Console sign-in link: https://539151687225.signin.aws.amazon.com/console

Console password Enabled (never signed in) | Manage

Assigned MFA device Not assigned | Manage

Signing certificates None

Access keys

Use access keys to make programmatic calls to AWS from the AWS CLI, Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time.

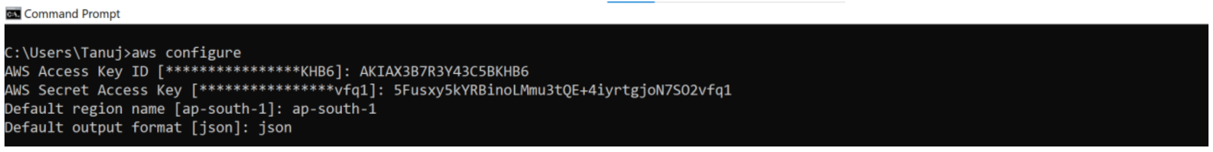
For your protection, you should never share your secret keys with anyone. As a best practice, we recommend frequent key rotation.
If you lose or forget your secret key, you cannot retrieve it. Instead, create a new access key and make the old key inactive. Learn more

Access key ID	Created	Last used	Status
AKIAX3B7R3Y43C5BKHB6	2021-12-16 14:21 UTC+0530	2021-12-21 22:54 UTC+0530 with cloudformation in ap-south-1	Active | Make inactive

- Download the access key generated.
- Now, in cmd configure the AWS by using the following command:

```
$ aws configure
```

Then enter the access key id, secret access key, region name and the output format.

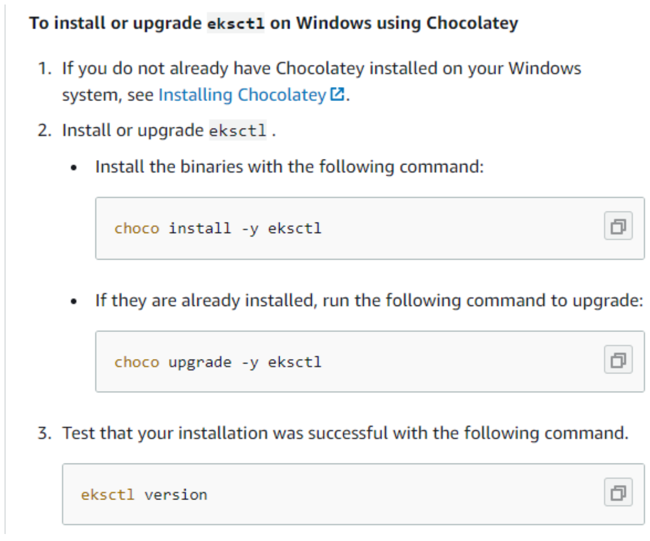


```
C:\Users\Tanuj>aws configure
AWS Access Key ID [****************KHB6]: AKIAX3B7R3Y43C5BKHB6
AWS Secret Access Key [****************vfq1]: 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
Default region name [ap-south-1]: ap-south-1
Default output format [json]: json
```

Step-4 Installation of eksctl:

- For installing the eksctl, chocolatey has to be installed first.
- In order to install Chocolatey, first, ensure that you are using an **administrative shell**.
- Copy the text specific to your command shell **- cmd.exe**.
- Paste the copied text into your shell and press Enter.
  @"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -InputFormat None -ExecutionPolicy Bypass -Command "[System.Net.ServicePointManager]::SecurityProtocol = 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))" && SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
- Wait a few seconds for the command to complete.

- After installing eksctl, run the commands as shown in the attached screenshot.



To install or upgrade `eksctl` on Windows using Chocolatey

1. If you do not already have Chocolatey installed on your Windows system, see Installing Chocolatey.
2. Install or upgrade `eksctl` .
   - Install the binaries with the following command:

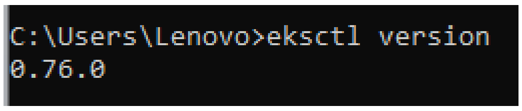
     ```
     choco install -y eksctl
     ```

   - If they are already installed, run the following command to upgrade:

     ```
     choco upgrade -y eksctl
     ```

3. Test that your installation was successful with the following command.

   ```
   eksctl version
   ```



```
C:\Users\Lenovo>eksctl version
0.76.0
```

Step-5 Create a cluster:

- In order to create a cluster, the following command is used:

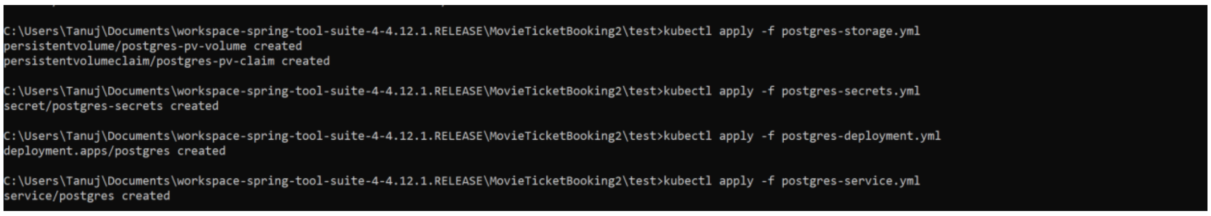
```
$ eksctl create cluster --name <cluster-name>  --version 1.21 --region <region-name>
--nodegroup-name <node-group-name> --node-type t2.micro –nodes 2
```

- To create or update kubeconfig for our cluster:

```
$ aws eks --region <region-code> update-kubeconfig --name <cluster-name>
```

- Now, create files using the kubectl command:

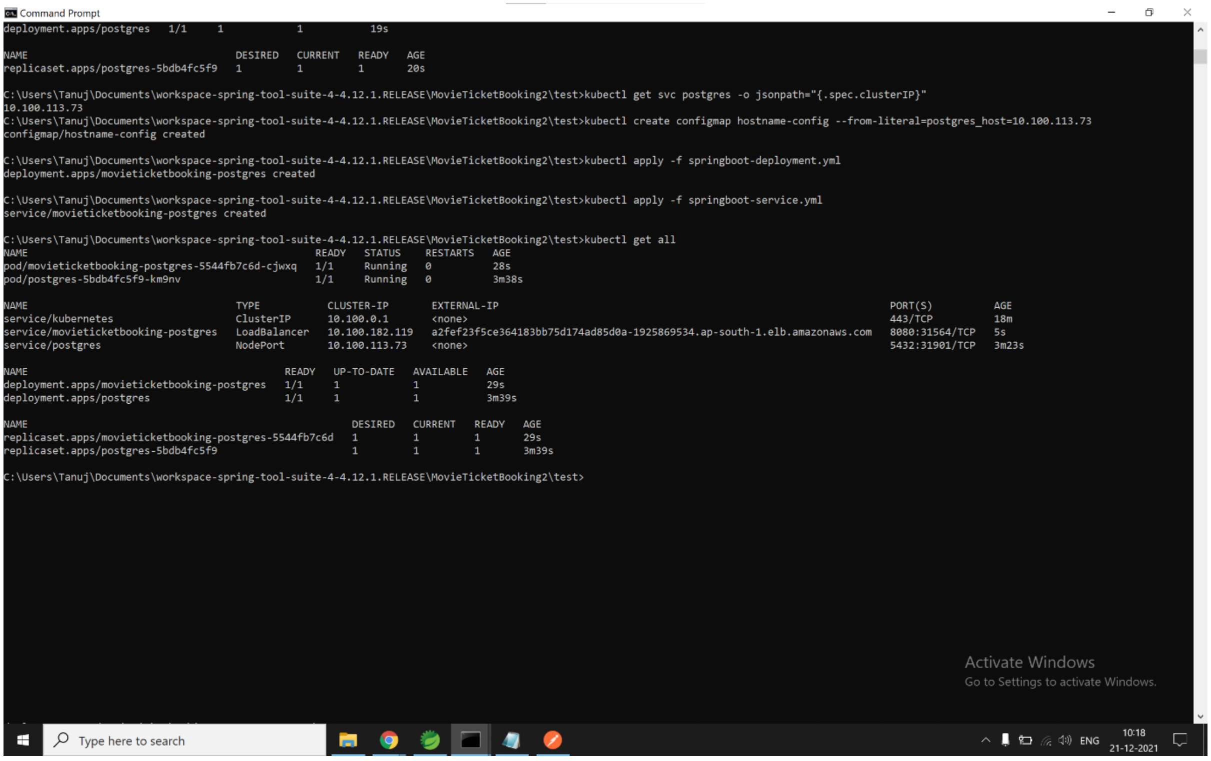
```
$ kubectl apply –f <file-name>
```



```
C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\test>kubectl apply -f postgres-storage.yml
persistentvolume/postgres-pv-volume created
persistentvolumeclaim/postgres-pv-claim created

C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\test>kubectl apply -f postgres-secrets.yml
secret/postgres-secrets created

C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\test>kubectl apply -f postgres-deployment.yml
deployment.apps/postgres created

C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\test>kubectl apply -f postgres-service.yml
service/postgres created
```

- To view all the pods, deployments and services use the following kubectl command:

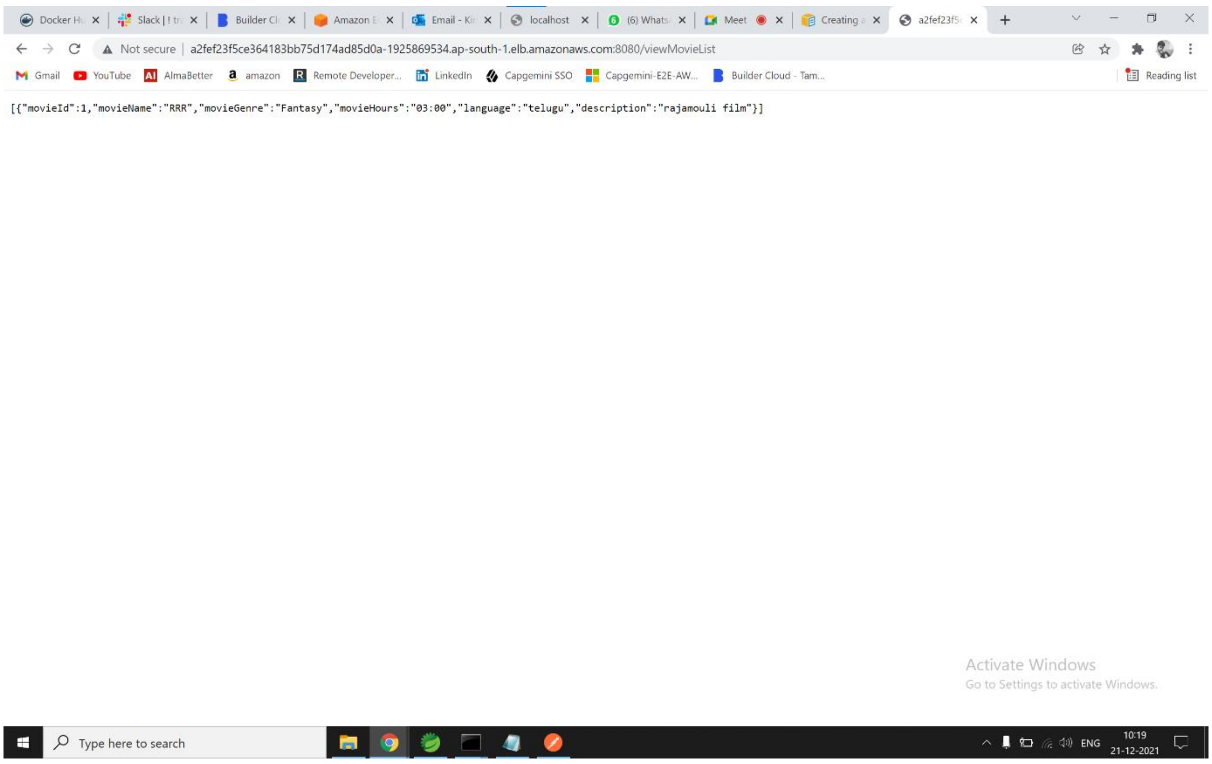
```
$ kubectl get all
```



```
deployment.apps/postgres   1/1     1            1           19s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/postgres-5bdb4fc5f9   1         1         1       20s

C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\test>kubectl get svc postgres -o jsonpath="{.spec.clusterIP}"
10.100.113.73
C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\test>kubectl create configmap hostname-config --from-literal=postgres_host=10.100.113.73
configmap/hostname-config created

C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\test>kubectl apply -f springboot-deployment.yml
deployment.apps/movieticketbooking-postgres created

C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\test>kubectl apply -f springboot-service.yml
service/movieticketbooking-postgres created

C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\test>kubectl get all
NAME                                               READY   STATUS    RESTARTS   AGE
pod/movieticketbooking-postgres-5544fb7c6d-cjwxq   1/1     Running   0          28s
pod/postgres-5bdb4fc5f9-km9nv                      1/1     Running   0          3m38s

NAME                                  TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PORT(S)          AGE
service/kubernetes                    ClusterIP      10.100.0.1       <none>                                                                     443/TCP          18m
service/movieticketbooking-postgres   LoadBalancer   10.100.182.119   a2fef23f5ce364183bb75d174ad85d0a-1925869534.ap-south-1.elb.amazonaws.com   8080:31564/TCP   5s
service/postgres                      NodePort       10.100.113.73    <none>                                                                     5432:31901/TCP   3m23s

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/movieticketbooking-postgres   1/1     1            1           29s
deployment.apps/postgres                      1/1     1            1           3m39s

NAME                                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/movieticketbooking-postgres-5544fb7c6d   1         1         1       29s
replicaset.apps/postgres-5bdb4fc5f9                      1         1         1       3m39s

C:\Users\Tanuj\Documents\workspace-spring-tool-suite-4-4.12.1.RELEASE\MovieTicketBooking2\test>
```

- Now, check in the browser by pasting the IP address in the browser:

  http://a2fef23f5ce364183bb75d174ad85d0a-1925869534.ap-south-1.elb.amazonaws.com:8080/viewMovieList



a2fef23f5ce364183bb75d174ad85d0a-1925869534.ap-south-1.elb.amazonaws.com:8080/viewMovieList

```
[{"movieId":1,"movieName":"RRR","movieGenre":"Fantasy","movieHours":"03:00","language":"telugu","description":"rajamouli film"}]
```