EKS, multi-tiers deployment

I- Containerize the Database using EKS Cluster

We have realized two missions as **DevOps Engineer** within the project team of **Mr. Alain**.

At this stage, we have done a **sonar analysis**, we have **set up a pipeline to deploy the application on EKS**. The customer is satisfied with the work done.

1- satisfaction and new problem

During the **sprint planning meeting**, you received congratulations and satisfaction from the **Product Owner** for the tasks you accomplished. Because he wanted to see **the developer's changes in real time**, which you did through the **pipeline**.
After the congratulations, the Product Owner reports that the customer is concerned about the fact that the sensitive data in the database is in the same container as the application.

Faced with the customer's concerns, the manager calls you into his office and would like to have a solution for this new problem in the **next Sprint**.

2- Analysis of the client's problem

For the analysis of this problem, the **Technical Lead** organizes a **Knowledge Transfer (KT)** session for the whole team on the **software architecture.**

During the training session, they focus on **monolithic applications and microservices architectures**. As a key point during the training, he shows that **microservices architecture have more advantages than monolithic architecture**.
Therefore, in order to meet the customer's needs, it is necessary to think microservice architecture. For a start, during this Sprint, it is necessary to separate the database from the application.

3- Proposal of a solution to separate the database from the application.

Here, the solution adapted to the context, is :
1- use a new EKS cluster to deploy the database
2- create a Volume in AWS to store the application data
3- deploy the application in a separate EKS cluster.

Image : Architecture of the solution

The Goal of this project is :

1- Separate the database from the application

2- Deploying the database in an EKS cluster

3- create a volume in AWS to store the application data

4- only deploy the application in another EKS cluster

5- Communicate the application and the database

Prerequisites

1. as a prerequisite, you must have completed the second mission of this wonderful project. To review what has already been done, please see the link below :

EKS, Jenkins Pipeline, Docker ECR

Implementation of the solution

---

step 1- Application configuration file : aplication.properties

- the first step in setting up this solution is to explain the configuration file.
- All **Java Applications** based on the **Spring boot framework** contain at their root folder a very important file called **application. properties**.
- this file contains several configuration parameters of the database.
- In our case, this file has the following content:

```
# ===============================
# SMTP EMAIL
# ===============================
server.port=8082
spring.mail.host = smtp.gmail.com
spring.mail.username = put_your_email_here@gmail.com
spring.mail.password = emailpass
spring.mail.port = 587
spring.mail.properties.mail.smtp.auth = true
spring.mail.properties.mail.smtp.starttls.enable = true
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.writetimeout=5000


# ===============================
# = LOGGING
# ===============================
logging.level.org.springframework.web=DEBUG
logging.level.org.hibernate=ERROR

# ===============================
# = DATA SOURCE
# ===============================
# spring.datasource.url=jdbc:mysql://localhost:3306/userauth
# spring.datasource.username=root
# spring.datasource.password=2j5R1HtsE7LmpM8Tdn92
# spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
 spring.datasource.tomcat.max-wait=10000
 spring.datasource.tomcat.max-active=5
 #spring.datasource.tomcat.test-on-borrow=true
 spring.datasource.initialization-mode=always

# spring.jpa.defer-datasource-initialization=true
 spring.jpa.hibernate.naming_strategy=org.hibernate.cfg.
EJB3NamingStrategy
 spring.sql.init.mode=always
# ===============================
# = JPA / HIBERNATE
# ===============================
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.datasource.initialization-mode=always
```

```
#server.servlet.context-path=/bio
# ===============================
# = Thymeleaf configurations
# ===============================
spring.thymeleaf.mode=LEGACYHTML5
spring.thymeleaf.cache=false

# H2 Database config

# H2 Database
spring.h2.console.enabled=true
#spring.datasource.url=jdbc:h2:mem:dcbapp
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=school1
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.datasource.url=jdbc:h2:mem:userauth
# Enabling H2 Console
# Custom H2 Console URL
spring.h2.console.path=/h2
spring.jpa.defer-datasource-initialization = true
```

---

step 2- configuration of access to the database

In this step, we will create a Volume to store the data of our database, then configure the accesses to this database

1- creation of the volume

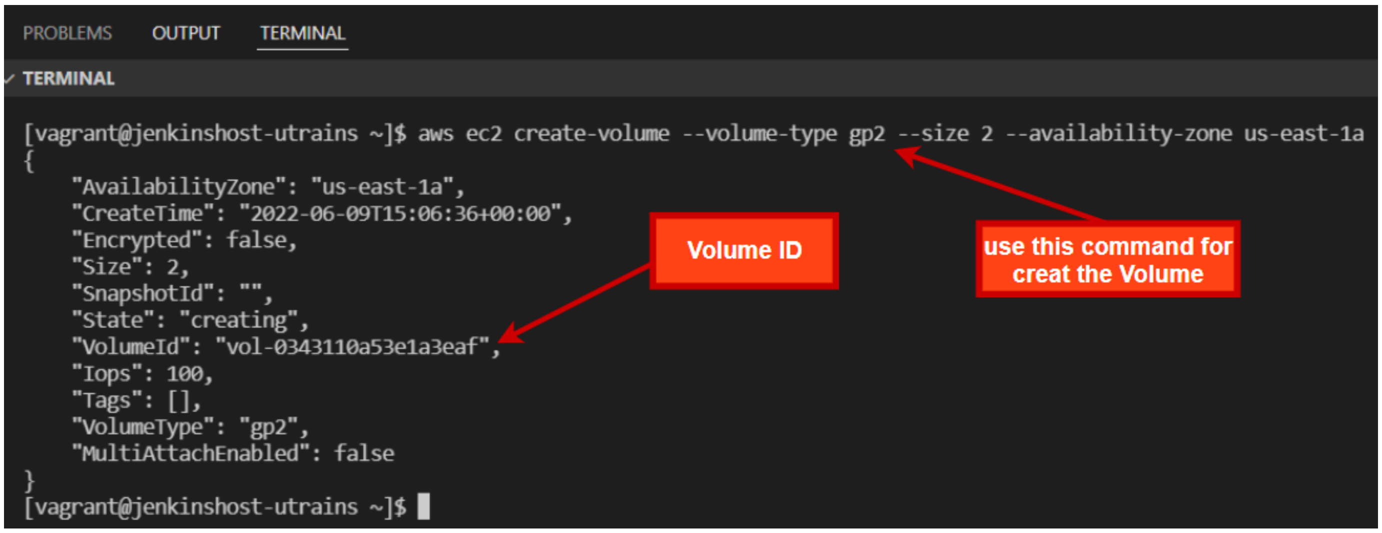
The volume allows the database data to be stored outside the container in a safe place. In our case, we will use an **AWS-CLI** command to create this volume in AWS.

- Open your **Jenkins server** where you have installed **AWS-CLI** and run the following command in the terminal to create the **volume.**

```
aws ec2 create-volume --volume-type gp2 --size 2 --availability-zone us-east-1a
```

this command will create a **2 GB SSD** volume in the **us-east-1a** zone

- [ ] @ Class8 copy the Volume ID and save somewhere : **vol-0343110a53e1a3eaf** for me



```
[vagrant@jenkinshost-utrains ~]$ aws ec2 create-volume --volume-type gp2 --size 2 --availability-zone us-east-1a
{
    "AvailabilityZone": "us-east-1a",
    "CreateTime": "2022-06-09T15:06:36+00:00",
    "Encrypted": false,
    "Size": 2,
    "SnapshotId": "",
    "State": "creating",
    "VolumeId": "vol-0343110a53e1a3eaf",
    "Iops": 100,
    "Tags": [],
    "VolumeType": "gp2",
    "MultiAttachEnabled": false
}
[vagrant@jenkinshost-utrains ~]$
```

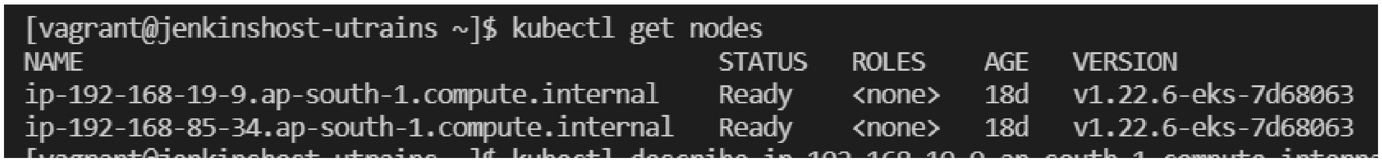
by curiosity, if you open your aws account, in the console, you can see the volume that is created.

small project 1: **Use Terraform, to create a volume in AWS.**

2- add the label in our different node.

- see our node with the get node command

```
kubectl get nodes
```

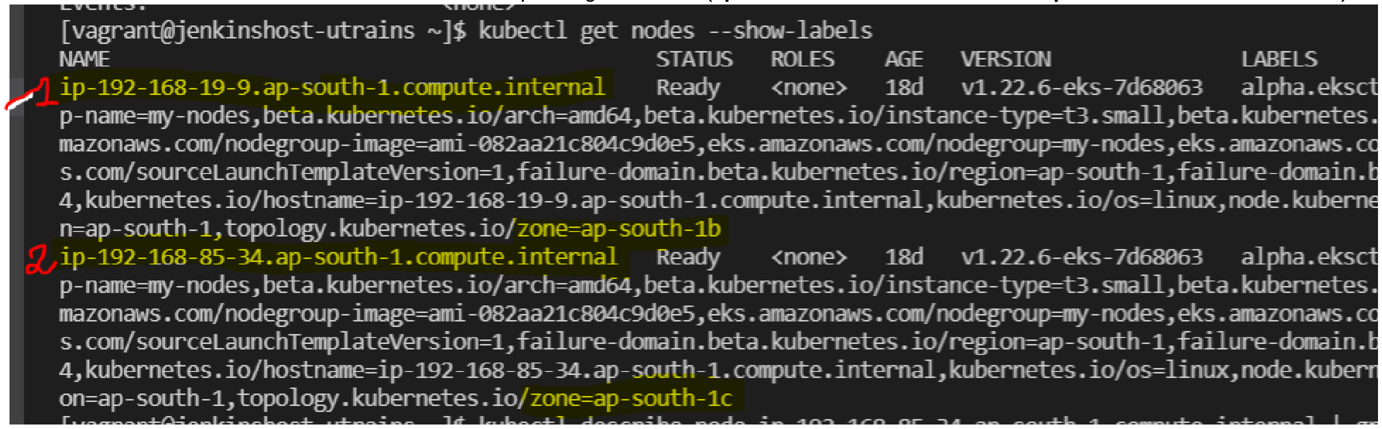


```
[vagrant@jenkinshost-utrains ~]$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE   VERSION
ip-192-168-19-9.ap-south-1.compute.internal   Ready    <none>   18d   v1.22.6-eks-7d68063
ip-192-168-85-34.ap-south-1.compute.internal  Ready    <none>   18d   v1.22.6-eks-7d68063
```

- use the command below to see the labels we can add to our nodes

```
kubectl get nodes --show-labels
```

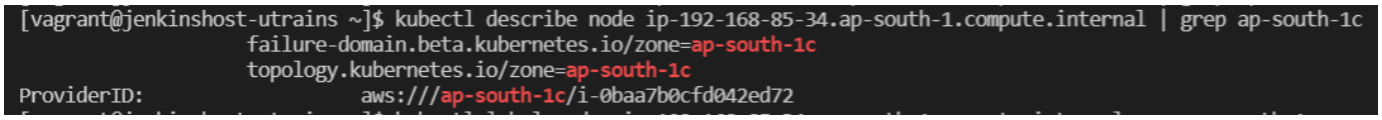
This command shows us our two nodes with the corresponding label zone (**ap-south-1b** for the node in **1** and **ap-south-1bc** for the node in **2**)



```
[vagrant@jenkinshost-utrains ~]$ kubectl get nodes --show-labels
NAME                                          STATUS   ROLES    AGE   VERSION               LABELS
ip-192-168-19-9.ap-south-1.compute.internal   Ready    <none>   18d   v1.22.6-eks-7d68063   alpha.eksct
p-name=my-nodes,beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=t3.small,beta.kubernetes.
mazonaws.com/nodegroup-image=ami-082aa21c804c9d0e5,eks.amazonaws.com/nodegroup=my-nodes,eks.amazonaws.co
s.com/sourceLaunchTemplateVersion=1,failure-domain.beta.kubernetes.io/region=ap-south-1,failure-domain.b
4,kubernetes.io/hostname=ip-192-168-19-9.ap-south-1.compute.internal,kubernetes.io/os=linux,node.kuberne
n=ap-south-1,topology.kubernetes.io/zone=ap-south-1b
ip-192-168-85-34.ap-south-1.compute.internal  Ready    <none>   18d   v1.22.6-eks-7d68063   alpha.eksct
p-name=my-nodes,beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=t3.small,beta.kubernetes.
mazonaws.com/nodegroup-image=ami-082aa21c804c9d0e5,eks.amazonaws.com/nodegroup=my-nodes,eks.amazonaws.co
s.com/sourceLaunchTemplateVersion=1,failure-domain.beta.kubernetes.io/region=ap-south-1,failure-domain.b
4,kubernetes.io/hostname=ip-192-168-85-34.ap-south-1.compute.internal,kubernetes.io/os=linux,node.kubern
on=ap-south-1,topology.kubernetes.io/zone=ap-south-1c
```

- use this command for filter the result for the second node, and check what zone it is …

```
kubectl describe node ip-192-168-85-34.ap-south-1.compute.internal |
grep ap-south-1c
```

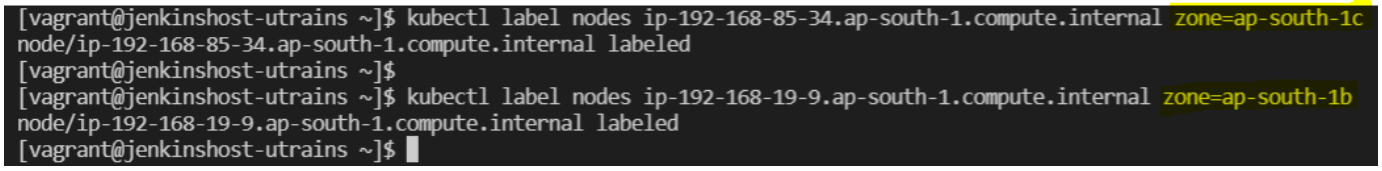


```
[vagrant@jenkinshost-utrains ~]$ kubectl describe node ip-192-168-85-34.ap-south-1.compute.internal | grep ap-south-1c
                    failure-domain.beta.kubernetes.io/zone=ap-south-1c
                    topology.kubernetes.io/zone=ap-south-1c
ProviderID:                   aws:///ap-south-1c/i-0baa7b0cfd042ed72
```

we will use the return of this command to add the label on our nodes, with this command below.

```
kubectl label nodes ip-192-168-85-34.ap-south-1.compute.internal
zone=ap-south-1c
```

do the same for the first node



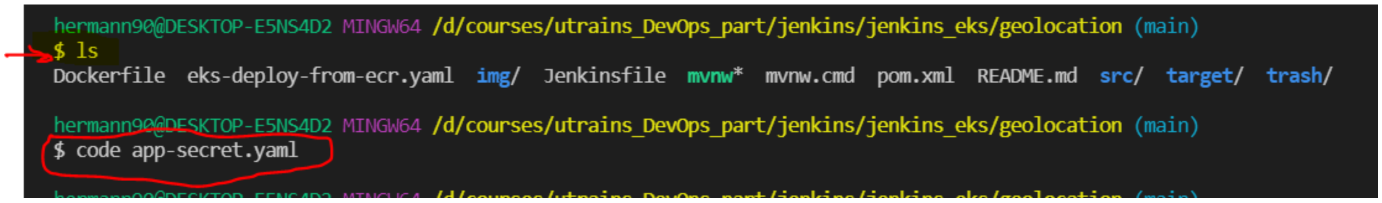
```
[vagrant@jenkinshost-utrains ~]$ kubectl label nodes ip-192-168-85-34.ap-south-1.compute.internal zone=ap-south-1c
node/ip-192-168-85-34.ap-south-1.compute.internal labeled
[vagrant@jenkinshost-utrains ~]$
[vagrant@jenkinshost-utrains ~]$ kubectl label nodes ip-192-168-19-9.ap-south-1.compute.internal zone=ap-south-1b
node/ip-192-168-19-9.ap-south-1.compute.internal labeled
[vagrant@jenkinshost-utrains ~]$
```

2- Secret password for database.

to access the database, we need access (**password,**). the yaml file that we will create in this part, will allow us to keep secret, the access information to our database

- create in our project folder, a yaml file for the database password : **app-secret.yaml**
- in virtual studio code, use the code command to create this file

```
code app-secret.yaml
```

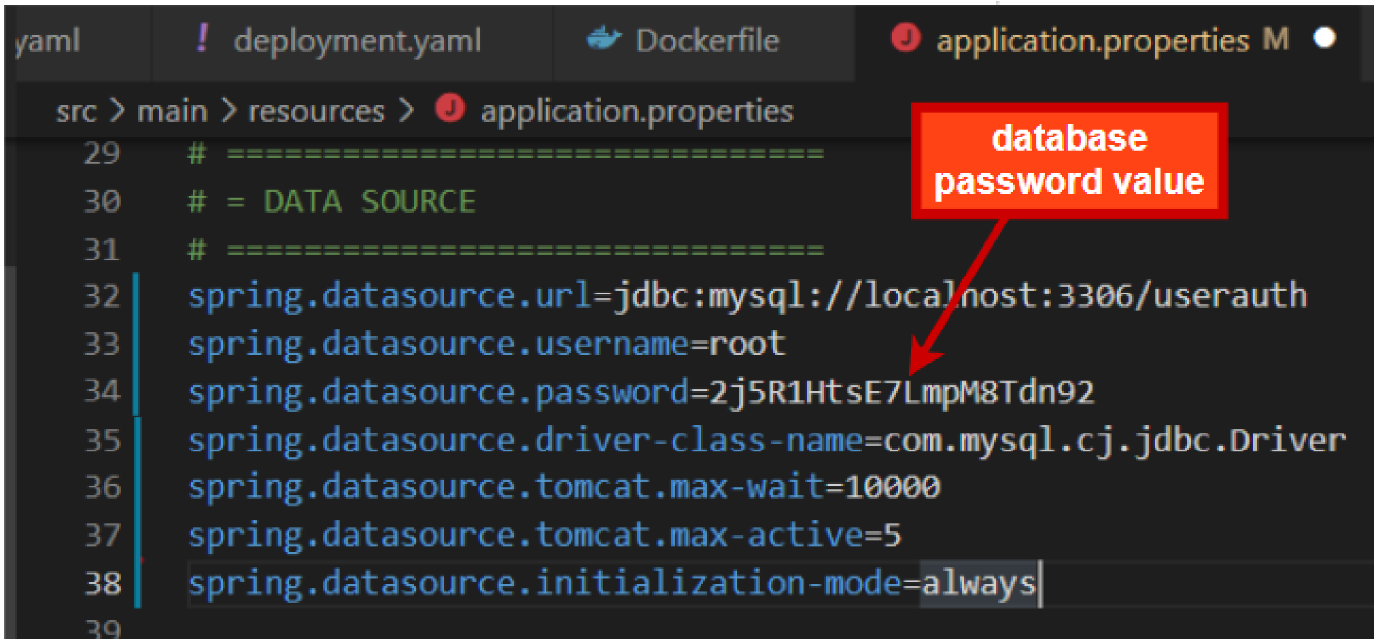


```
hermann90@DESKTOP-E5NS4D2 MINGW64 /d/courses/utrains_DevOps_part/jenkins/jenkins_eks/geolocation (main)
$ ls
Dockerfile  eks-deploy-from-ecr.yaml  img/  Jenkinsfile  mvnw*  mvnw.cmd  pom.xml  README.md  src/  target/  trash/

hermann90@DESKTOP-E5NS4D2 MINGW64 /d/courses/utrains_DevOps_part/jenkins/jenkins_eks/geolocation (main)
$ code app-secret.yaml
```

2.1- encrypts the database password

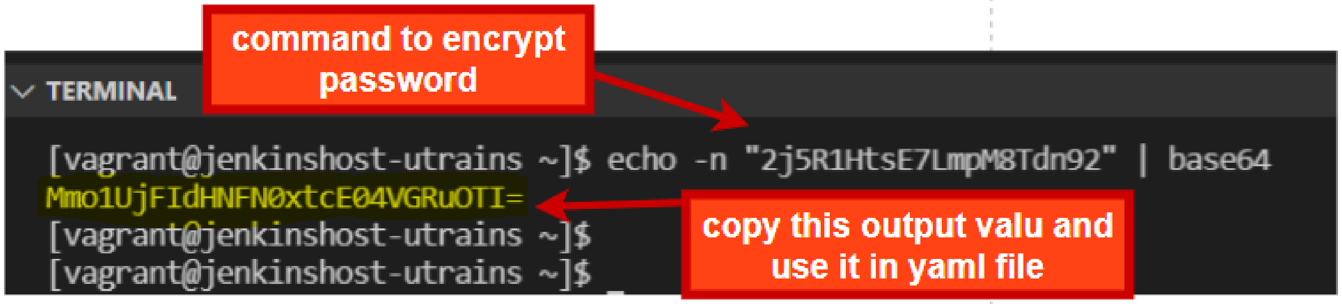
to encrypt the password of the database we have the following steps:

1. open the file **application.properties**, then on the configuration parameters of the project, you will see the config of the data source; just copy the password value



```
# ================================
# = DATA SOURCE
# ================================
spring.datasource.url=jdbc:mysql://localhost:3306/userauth
spring.datasource.username=root
spring.datasource.password=2j5R1HtsE7LmpM8Tdn92
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.tomcat.max-wait=10000
spring.datasource.tomcat.max-active=5
spring.datasource.initialization-mode=always
```

2. We will copy the value of this password, use the following command to encrypt it


TERMINAL
command to encrypt password
copy this output valu and use it in yaml file

```
[vagrant@jenkinshost-utrains ~]$ echo -n "2j5R1HtsE7LmpM8Tdn92" | base64
Mmo1UjFIdHNFN0xtcE04VGRuOTI=
[vagrant@jenkinshost-utrains ~]$
[vagrant@jenkinshost-utrains ~]$
```

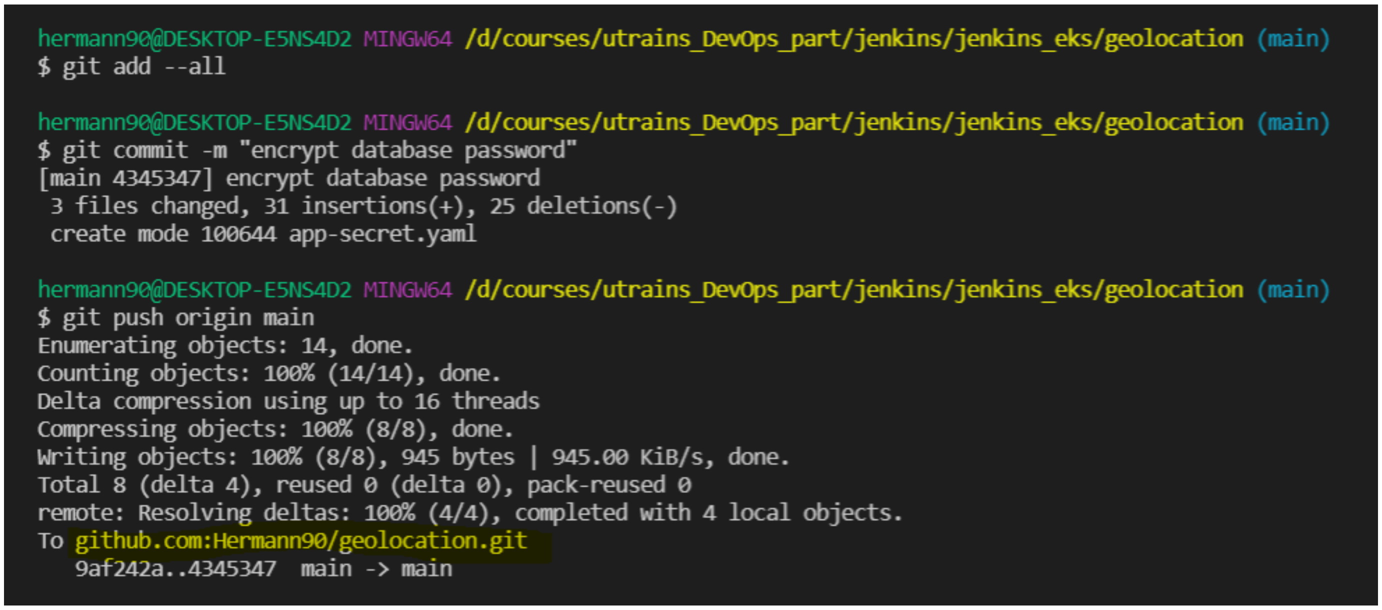
3. copy the encrypted value of the password, then paste it into the yaml file in front of the **db-pass** attribute

☐ @ Class8 create the **app-secret.yaml** file. **encrypt the database password** and put the encrypt value in the **db-pass** attribute (see line 7 bellow)

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
type: Opaque
data:
  db-pass: Mmo1UjFIdHNFN0xtcE04VGRuOTI=
```

4. do a commit and push.

```
git add --all
git commit -m "secret encryption"
git push origin main
```

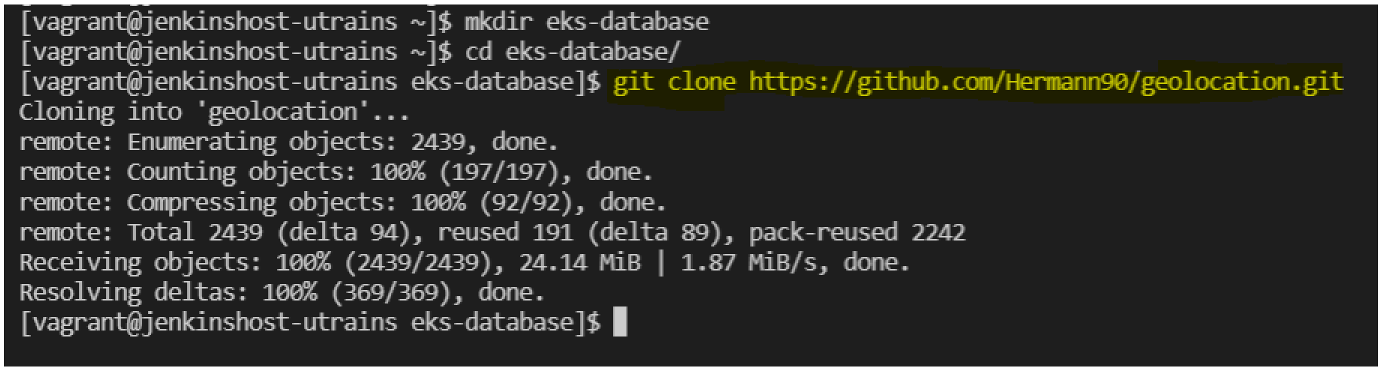

```
hermann90@DESKTOP-E5NS4D2 MINGW64 /d/courses/utrains_DevOps_part/jenkins/jenkins_eks/geolocation (main)
$ git add --all

hermann90@DESKTOP-E5NS4D2 MINGW64 /d/courses/utrains_DevOps_part/jenkins/jenkins_eks/geolocation (main)
$ git commit -m "encrypt database password"
[main 4345347] encrypt database password
 3 files changed, 31 insertions(+), 25 deletions(-)
 create mode 100644 app-secret.yaml

hermann90@DESKTOP-E5NS4D2 MINGW64 /d/courses/utrains_DevOps_part/jenkins/jenkins_eks/geolocation (main)
$ git push origin main
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 16 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 945 bytes | 945.00 KiB/s, done.
Total 8 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To github.com:Hermann90/geolocation.git
   9af242a..4345347  main -> main
```

5. to test, you can clone this repository in our Jenkins server.

- connect to jenkins server
- create the new folder in the home directory named **eks-database**
- go to this directory, using cd command then clone the repository for the geolocation project here

☐ @ Class8 please connect you in our Jenkins server, clone your own geolocation repos

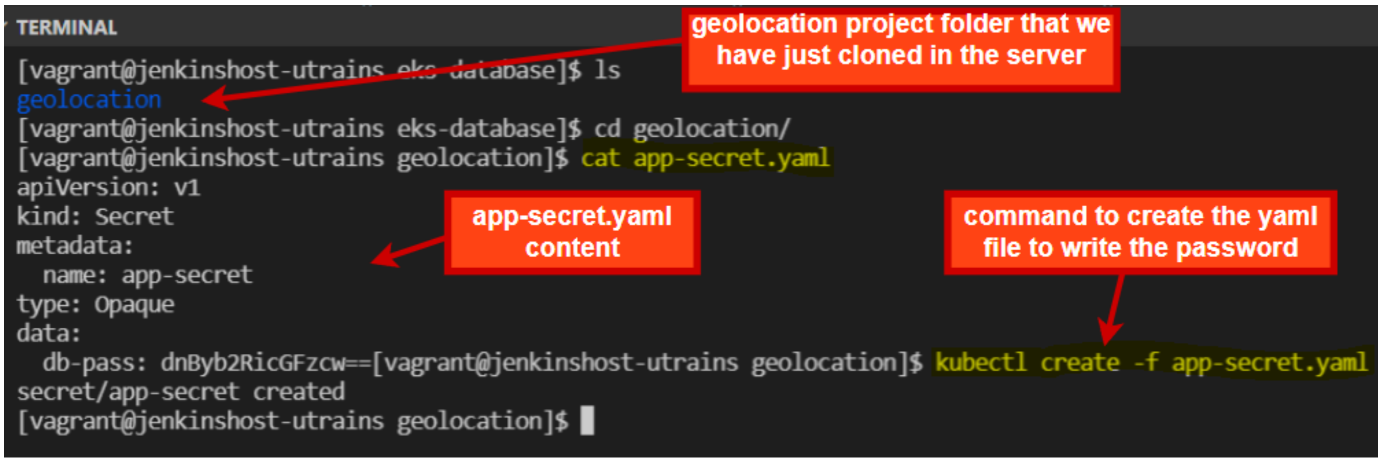
```
ssh vagrant@192.168.56.177
mkdir eks-database
cd eks-database
git clone https://github.com/Hermann90/geolocation.git
```


```
[vagrant@jenkinshost-utrains ~]$ mkdir eks-database
[vagrant@jenkinshost-utrains ~]$ cd eks-database/
[vagrant@jenkinshost-utrains eks-database]$ git clone https://github.com/Hermann90/geolocation.git
Cloning into 'geolocation'...
remote: Enumerating objects: 2439, done.
remote: Counting objects: 100% (197/197), done.
remote: Compressing objects: 100% (92/92), done.
remote: Total 2439 (delta 94), reused 191 (delta 89), pack-reused 2242
Receiving objects: 100% (2439/2439), 24.14 MiB | 1.87 MiB/s, done.
Resolving deltas: 100% (369/369), done.
[vagrant@jenkinshost-utrains eks-database]$ 
```

6. test the yaml secret file

here is the list of commands to test the yaml file to write the database password :

```
cd geolocation
cat app-secret.yaml
kubectl create -f app-secret.yaml
```


TERMINAL
geolocation project folder that we have just cloned in the server
app-secret.yaml content
command to create the yaml file to write the password

```
[vagrant@jenkinshost-utrains eks-database]$ ls
geolocation
[vagrant@jenkinshost-utrains eks-database]$ cd geolocation/
[vagrant@jenkinshost-utrains geolocation]$ cat app-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
type: Opaque
data:
  db-pass: dnByb2RicGFzcw==[vagrant@jenkinshost-utrains geolocation]$ kubectl create -f app-secret.yaml
secret/app-secret created
[vagrant@jenkinshost-utrains geolocation]$
```

- to see the result of this file description, run this command

```
kubectl describe secret
```

This is a DevOps best practice for securely using passwords. This will allow us to connect to our database server more often.

- we will deploy the database in a separate EKS cluster.
- The database is the heart of the business, to keep it secure, we need to use good security practices.
- We will encrypt the password of this database and use it in a yaml file