

Jenkins, Docker, AWS ECR and deploy to Amazon EKS,

Project Context :

- **Alain Pierre** is the **manager** of a team of **7 developers** of an **IT solution** development company called **e-Max Soft** (a **Cameroonian firm of high end software development**).
- He has just set up a **Software project used by patients for geolocation of doctors so as to get medical assistance**.
- His team, being only experienced in **DEV**, has created a **GitHub repository** for the project code.
- These developers use the **Java language, HTML and Java Script** to write their code. As build tools, they use **maven** to **compile** their source code in an archaic way.
- **Alain Pierre** the manager of this very ambitious project **recruits** you in this team as a **DevOps engineer**.

With the notions learned at **Utrains**, your

I - First Mission : set up the source code analysis for this project.

Click on the link below, to see the work done during your first mission.

Tasks performed for the first mission

II - Second Mission : Propose an implementation of the Agile method, then build a Pipeline to deploy the application using Amazon EKS.

Part I : Implementation of the Agile Method : The Scrum Framework

You are currently performing the **Sonar analysis**. This work has been very useful to the team you have joined. Because developers can now see the results of their source code analysis in order to fix vulnerabilities in their code.

So, **Alain Pierre**, the Project Manager, invites you to a **One to One meeting** (it's a meeting between you and the manager where everyone is free to say what they find difficult in the team, and what we can put in place to make them feel comfortable). Then, he tells you about his desire to adopt **good practices in the development process of this software**. During your first mission, you realize that the **Waterfall model** is currently used in the development process of this software.

1- Explain to the manager what the **Waterfall model** is and its limitations

During your time at Utrains, you have been told about the **Agile model**.

2- Explain the Agile model to the manager

3- Propose an implementation of the **Scrum Framework** to improve the development process of this software.

4- Based on the **Scrum Framework**, explain the following terms:

- a- Daily meeting :
- b- Sprint retrospective
- c- Scrum Master
- d- Product Owner
- e- Backlog
- f- Sprint planning

5- About IT project management tools :

- a- What is the role of **JIRA** in an organization implementing the Scrum Framework?
- b- What is the Role of **Slack**?
- c- What is the role of **Confluence**?
- d- What other **collaborative work** and **IT project management tools** do you know?

6- Based on the questions asked above, propose to the project manager during your **One To One** meeting, an implementation of an **agile method**, precisely the **Scrum Framework**.

Part II : Jenkins, AWS ECR, Amazon EKS : Pipeline to Deploy our **geolocation** App

Now that the Manager has recruited you as **DevOps engineer**, the customer (a **referral hospital**), represented in some of the Agile ceremony meetings by the **Product Owner** has new concerns.

- Recently, the customer has acquired an **account in AWS**.
- He wants to be able to **see instantly** the **changes that the developers make in the main branch** of the repository.
- The customer also wants to use the **new software updates in a very short and regular time** (at the end of **each Sprint**, he wants to have a new **release with new features**)

Following the proposal of the use of the **Agile method (Scrum Framework)** that you have set up, **Alain Pierre** the manager validates your proposal. The **Monday** following the validation of the use of this agile method, is the **Sprint Planning meeting**. The **Scrum Master**, main animator of the Agile ceremonies, takes out of the **Backlog**, the list of tasks prioritized by the **Product Owner**.

Each team member is given a set of tasks in **JIRA**.

The **DevOps** task prioritized for you in this first Sprint is the following: **Set up a CI/CD Pipeline to deploy the Docker image of the geolocation application using a Jenkins server, AWS ECR and Amazon EKS.**

After the creation of this task in JIRA, with the guidance of the team leader, you have the following description:

A- Task Description :

- **Task Code** : GEO_DEVOPS-001
- **Task Name** : Set up a CI/CD Pipeline to deploy the Docker image of the **geolocation** application using a Jenkins server, AWS ECR and Amazon EKS.
- **More details** :
 - 1- Set up Jenkins server and install docker on it.
 - 2- Kubernetes(EKS) cluster up and running.
 - 3- Install necessary Jenkins plugins:
 - 4- Amazon ECR plugin
 - 5- Docker Pipeline

Proposition of the Solution

Part I - Solution for the Agile Method Implementation: The Scrum Framework.

Notes : Each student must research and understand how the agile method works, more specifically the Scrum framework. However, please feel free to leave your concerns in the comments.

Part II - Solution for Jenkins, AWS ECR, Amazon EKS : Pipeline to Deploy our **geolocation** App

The Goal of this project is :

- 1- Automating builds using Jenkins
 - 2- Automating Docker image creation
 - 3- Automating Docker image upload into AWS ECR
 - 4- Automating Deployments to Kubernetes Cluster
-

Prerequisites

1. Jenkins Master is up and running
2. Docker installed on Jenkins instance
3. Maven and git are correctly integrated

Tips: In short, just use our usual jenkins server

As a reminder, we have our jenkins server installed with vagrant which contains the following characteristics :

- **IP Address** : 192.168.56.17
- **Username** : vagrant
- **Password** : vagrant

The process goes as follows:

- 1- Open the terminal,
- 2- Connect to the server with **ssh**
- 3- Use the **yum** package manager to **install git**

```
ssh vagrant@192.168.56.177
```

You can go to this link if you want to review the implementation of this jenkins server :

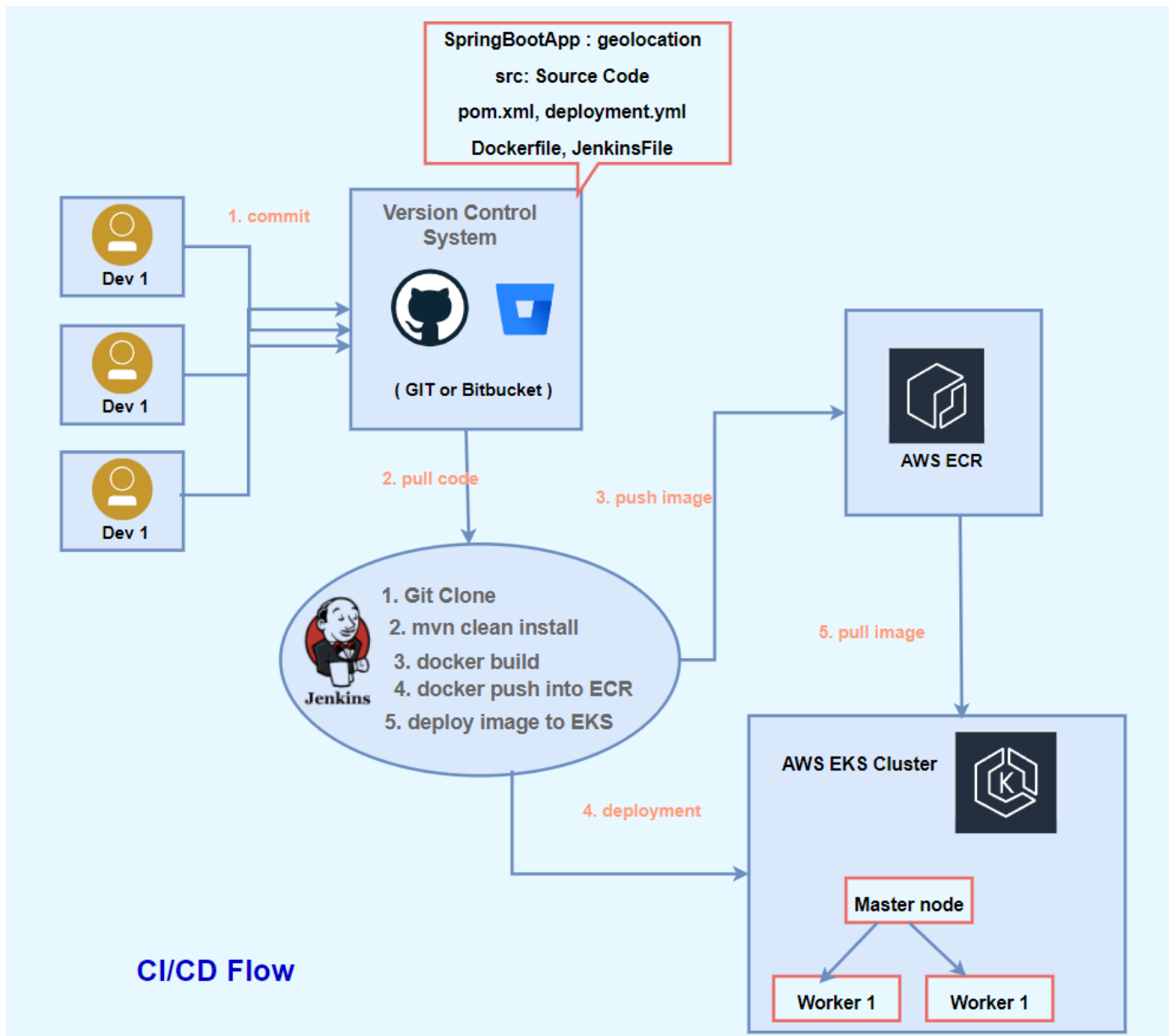
[Set Up Jenkins Server](#)

The main steps for the realization of the project are :

- 1- Setup Jenkins and install Docker and Required plugins (prerequisites)
 - 2- Create **EKS cluster** using **eksctl**
 - 3- Create Jenkins Pipeline to **Deploy geolocation project into EKS cluster**
 - 4- Verify Deployment using **kubectl**
 - 5- Access the Java geolocation App
-

Step 1 : CI/CD Workflow oh the solution

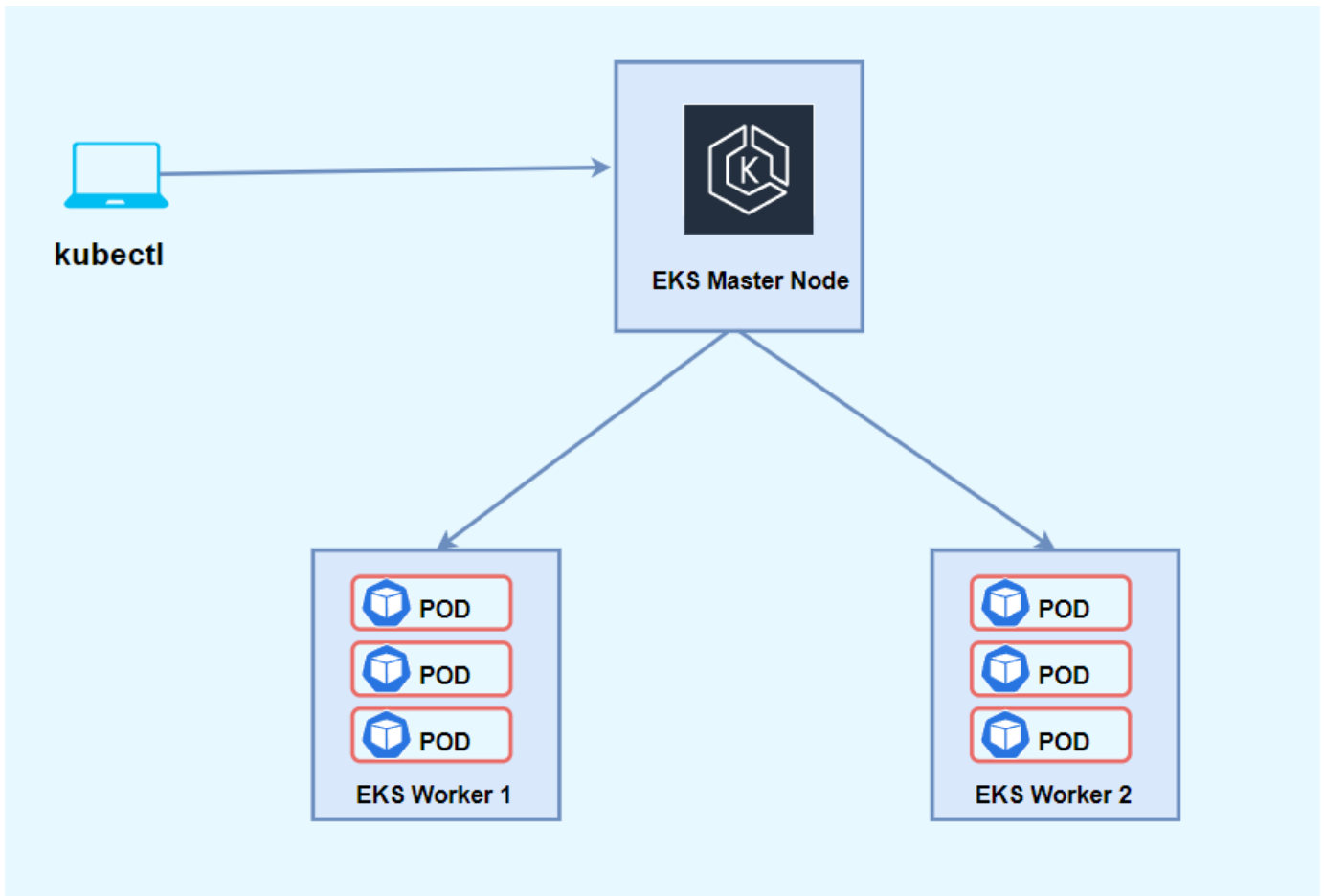
- Here is the workflow of one of the solutions we will implement for this project



- This workflow gives us an overall idea of what we are going to achieve in this wonderful mission.
- As a comment of this workflow to give you an idea of this solution, we have :
 - A **GitHub repository** that contains the source code of the geolocation project.
 - In our **Jenkins server**, we will install Docker, configure ECR, and many other plugins
 - we have also, our EKS cluster which will contain a Master node and 2 workers node

Step 2- Create Amazon EKS cluster by **eksctl**

- **What is Amazon EKS :**
 - **Amazon EKS** is a fully managed container orchestration service.
 - **EKS** allows you to quickly deploy a production ready **Kubernetes cluster** in **AWS**, **deploy** and **manage containerized applications** more easily with a fully managed Kubernetes service.
- **Amazon EKS cluster architecture:**
 - In the architecture of our project, we will set up a Master node, and 2 worker nodes.
 - The different tools we will install to create this EKS cluster are :
 - **AWS CLI** : A command line tools for working with AWS services, including Amazon EKS.
 - **eksctl** : A command line tool for working with EKS clusters that automates many individual tasks.
 - **kubectl** : A command line tool for working with Kubernetes clusters.



- As we will use the commands to create our EKS cluster in AWS, we need to install the following tools: **AWS CLI**, **eksctl** and **kubectl**.
- We will install these tools on our **Jenkins server**. But, nothing prevents you from installing them on your local machine.

1- installation of AWS Client (AWS CLI)

This tool is necessary to authenticate your requests to your account on **Amazon Web Services**.

Before this installation, we need to create an **Access Key in AWS**

a- Create Access Key in your AWS Account

- Login to your AWS Account
- Services IAM click on Users then select your name
- Click on **Create Access Key**

Services

Search for services, features, blogs, docs, and more
[Alt+S]

Identity and Access Management (IAM)

- Dashboard
- Access management
 - User groups
 - Users**
 - Roles
 - Policies
 - Identity providers
 - Account settings
- Access reports
 - Access analyzer
 - Archive rules
 - Analyzers
 - Settings
 - Credential report
 - Organization activity
 - Service control policies (SCPs)

Search IAM

AWS account ID:
076892551558

Summary

User ARN

arn:aws:iam::076892551558:user/hermann90

Path

/

Creation time

2021-11-12 01:07 UTC+0400

Permissions

Groups (1)

Tags (1)

Security credentials

Access Advisor

Sign-in credentials

Summary

- Console sign-in link: <https://076892551558.signin.aws.amazon...>

Console password

Enabled (last signed in Today) | [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

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](#)

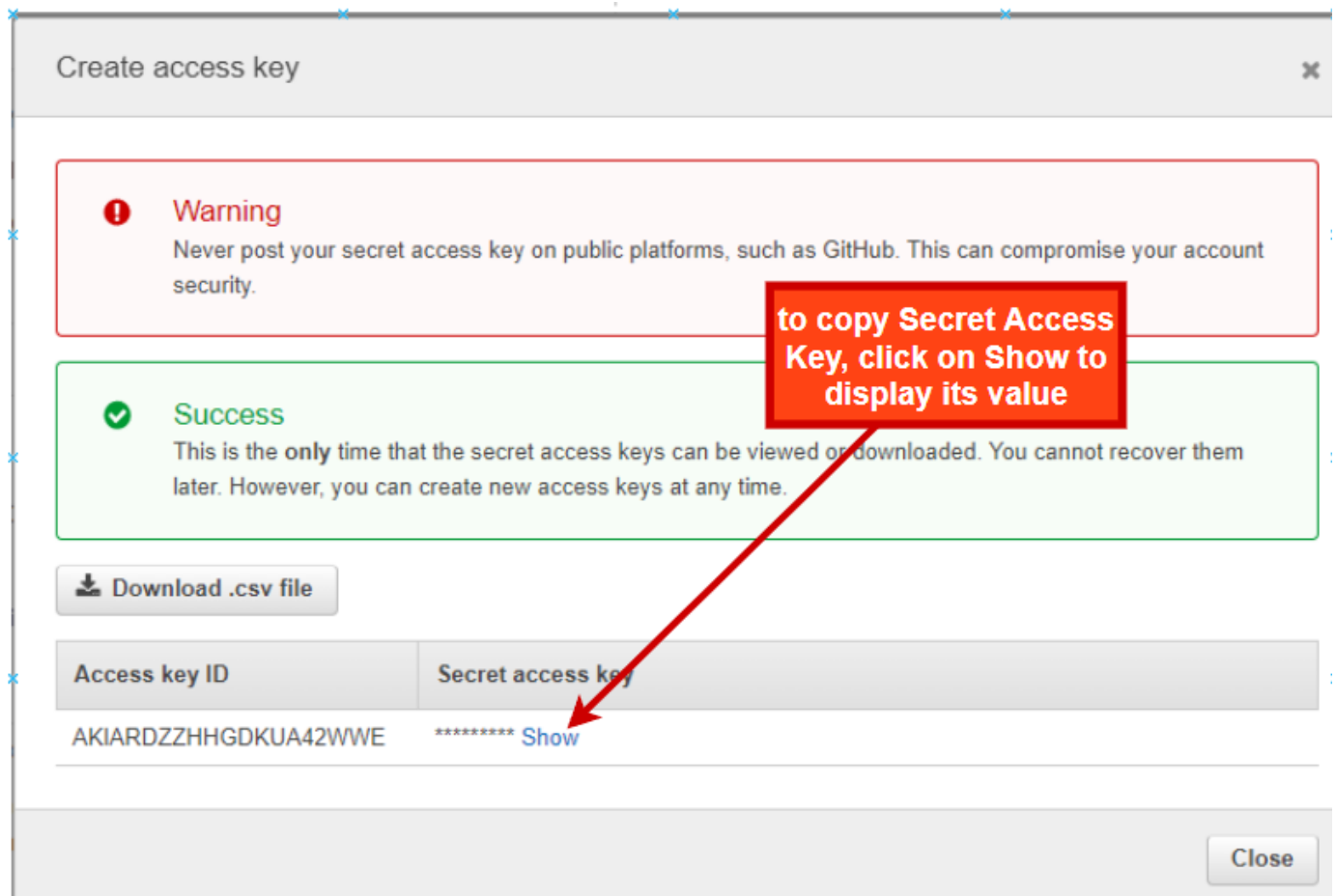
Create access key

Access key ID	Created	Last used
AKIARDZZHHGDKC44NLOQ	2022-01-29 16:18 UTC+0400	2022-05-20 14:06 UTC+0400 with sts in us-east-1

SSH keys for AWS CodeCommit

Use SSH public keys to authenticate access to AWS CodeCommit repositories. [Learn more](#)

- A popup appears where you can download the Key you just created, or just copy it. It's up to you, because we are going to configure our AWS CLI with this two information (Access Key ID and Secret Access Key). copy these two information somewhere, then we will use them after. to copy Secret Access Key, click on Show to display its value



b- Now, let's install and configure our AWS CLI with these two Information's

Follow these steps from the command line to install the AWS CLI on Linux.

- Open a terminal, then log in to your Jenkins server.
- Then execute this commands in the terminal to install AWS CLI in this server.

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o  
"awscliv2.zip"  
unzip awscliv2.zip  
sudo ./aws/install  
aws --version
```

```
[vagrant@jenkinshost-utrains ~]$ ls  
aws awscliv2.zip  
[vagrant@jenkinshost-utrains ~]$ sudo ./aws/install  
You can now run: /usr/local/bin/aws --version  
[vagrant@jenkinshost-utrains ~]$ which aws  
/usr/local/bin/aws  
[vagrant@jenkinshost-utrains ~]$ aws --version  
aws-cli/2.7.1 Python/3.9.11 Linux/3.10.0-1160.62.1.el7.x86_64 exe/x86_64.centos.7 prompt/off  
[vagrant@jenkinshost-utrains ~]$
```

- Enter this command, then put the configuration information of your AWS account.

```
aws configure
```

Here, we need to copy and paste our **Access Key**, **Access Secret Key**, etc.

```
✓ TERMINAL

[vagrant@jenkinshost-utrains ~]$ aws configure
AWS Access Key ID [None]: AKIARDZZHHGD[REDACTED]
AWS Secret Access Key [None]: jiYvnrORwLOu CZqY5x3vXe5IRe[REDACTED]
Default region name [None]: ap-south-1
Default output format [None]:
[vagrant@jenkinshost-utrains ~]$
```

put your Credentials in the terminal

AWS CLI is correctly install and configure in our Jenkins Server.

2- Installing eksctl

- This section helps you install the latest version of the **eksctl** command line utility in our Jenkins server.
- Open the terminal and connect you into your Jenkins server
- We have 3 steps for this installation:

a- **Download** and extract the latest release of **eksctl** with the following command.

This command allows to download, then extract the last version of **eksctl** as an executable binary file.

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
```

```
✓ TERMINAL BASH +

[vagrant@jenkinshost-utrains ~]$ curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
[vagrant@jenkinshost-utrains ~]$ ls /tmp
eksctl                               tmp.cQQj6FQTH1
hsperdata jenkins                  tmp.ftkig3Xe-f7
```

b- **Move** the extracted binary to **/usr/local/bin**.

```
sudo mv /tmp/eksctl /usr/local/bin
```

c- **Test** that your installation was successful with the following command.

```
eksctl version
```


▼ TERMINAL

```
[vagrant@jenkinshost-utrains ~]$ sudo mv /tmp/eksctl /usr/local/bin
[vagrant@jenkinshost-utrains ~]$ eksctl version
0.98.0
[vagrant@jenkinshost-utrains ~]$
```

3- Installing kubectl

- **Kubernetes** uses a command line utility called **kubectl** for communicating with the **cluster API server**.
- It is tool for **controlling Kubernetes clusters**.
- We have 3 steps for this installation

a- download kubectl

This command allows us to download **kubectl** and put it in the **/usr/local/bin/** directory

```
sudo curl --silent --location -o /usr/local/bin/kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-09/bin/linux/amd64/kubectl
```

so the **ls /usr/local/bin/** command allows us to see this file

```
[vagrant@jenkinshost-utrains ~]$ sudo curl --silent --location -o /usr/local/bin/kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-09/bin/linux/amd64/kubectl
[vagrant@jenkinshost-utrains ~]$
[vagrant@jenkinshost-utrains ~]$ ls /usr/local/bin
aws  aws_completer  eksctl  kubectl
[vagrant@jenkinshost-utrains ~]$
```

b- assign the execution rights to kubectl file

assign the execution rights to this file with the command below

```
sudo chmod +x /usr/local/bin/kubectl
```

c- Verify if kubectl got installed successfully

```
kubectl version --short --client
```

```
[vagrant@jenkinshost-utrains ~]$ kubectl version --short --client
Client Version: v1.22.6-eks-7d68063
[vagrant@jenkinshost-utrains ~]$
```

4- Create EKS Cluster with two worker nodes using eksctl

a- switch to jenkins user

in the terminal of the jenkins server, change user, and connect with the jenkins user.

```
sudo su - jenkins
```

b- create a EKS cluster command

The bellow command should create a EKS cluster in AWS, it might take some minutes.

Warning : depending on your network speed, this command can take a long time

```
eksctl create cluster --name geolocation-eks --region ap-south-1 --  
nodegroup-name my-nodes --node-type t3.small --managed --nodes 2
```

Note : If an error occurs during the execution of this command, it is probably the **synchronization** of the time of the Jenkins server and the AWS servers. **Run** `$ date` **to confirm this error**. To correct it, you will :

- Disable the automatic time synchronization

```
sudo timedatectl set-ntp 0
```

- Update the time of your jenkins server with the time you have on your personal machine.

```
timedatectl set-time '2022-05-22 10:01:00'
```

- Run the command once more
- When everything goes normally, after a few minutes, we can see :
 - The **geolocation-eks cluster** is created and that this **cluster contains 2 nodes** :
 - Node "ip-192-168-19-9.ap-south-1.compute.internal" is ready
 - Node "ip-192-168-85-34.ap-south-1.compute.internal" is ready

```

[vagrant@jenkinshost-utrains ~]$ eksctl create cluster --name geolocation-eks --region ap-south-1 --nodegroup-name my-nodes --no
de-type t3.small --managed --nodes 2
2022-05-22 10:01:55 [i] eksctl version 0.98.0
2022-05-22 10:01:55 [i] using region ap-south-1
2022-05-22 10:01:56 [i] setting availability zones to [ap-south-1b ap-south-1a ap-south-1c]
2022-05-22 10:01:56 [i] subnets for ap-south-1b - public:192.168.0.0/19 private:192.168.96.0/19
2022-05-22 10:01:56 [i] subnets for ap-south-1a - public:192.168.32.0/19 private:192.168.128.0/19
2022-05-22 10:01:56 [i] subnets for ap-south-1c - public:192.168.64.0/19 private:192.168.160.0/19
2022-05-22 10:01:56 [i] nodegroup "my-nodes" will use "" [AmazonLinux2/1.22]
2022-05-22 10:01:56 [i] using Kubernetes version 1.22
2022-05-22 10:01:56 [i] creating EKS cluster "geolocation-eks" in "ap-south-1" region with managed nodes
2022-05-22 10:01:56 [i] will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2022-05-22 10:01:56 [i] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region
=ap-south-1 --cluster=geolocation-eks'
2022-05-22 10:01:56 [i] Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster
"geolocation-eks" in "ap-south-1"
2022-05-22 10:01:56 [i] CloudWatch logging will not be enabled for cluster "geolocation-eks" in "ap-south-1"
2022-05-22 10:01:56 [i] you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE
(e.g. all)} --region=ap-south-1 --cluster=geolocation-eks'
2022-05-22 10:01:56 [i]
2 sequential tasks: { create cluster control plane "geolocation-eks",
  2 sequential sub-tasks: {
    wait for control plane to become ready,
    create managed nodegroup "my-nodes",
  }
}
2022-05-22 10:01:56 [i] building cluster stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:01:59 [i] deploying stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:02:29 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:03:00 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:04:01 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:05:02 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:06:04 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:07:05 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:08:06 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:09:08 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:10:09 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:11:10 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:12:12 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-cluster"
2022-05-22 10:14:24 [i] building managed nodegroup stack "eksctl-geolocation-eks-nodegroup-my-nodes"
2022-05-22 10:14:24 [i] deploying stack "eksctl-geolocation-eks-nodegroup-my-nodes"
2022-05-22 10:14:25 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-nodegroup-my-nodes"
2022-05-22 10:14:56 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-nodegroup-my-nodes"
2022-05-22 10:15:47 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-nodegroup-my-nodes"
2022-05-22 10:17:31 [i] waiting for CloudFormation stack "eksctl-geolocation-eks-nodegroup-my-nodes"
2022-05-22 10:17:31 [i] waiting for the control plane availability...
2022-05-22 10:17:31 [✓] saved kubeconfig as "/home/vagrant/.kube/config"
2022-05-22 10:17:31 [i]
2022-05-22 10:17:31 [✓] all EKS cluster resources for "geolocation-eks" have been created
2022-05-22 10:17:33 [i] nodegroup "my-nodes" has 2 node(s)
2022-05-22 10:17:33 [i] node "ip-192-168-19-9.ap-south-1.compute.internal" is ready
2022-05-22 10:17:33 [i] node "ip-192-168-85-34.ap-south-1.compute.internal" is ready
2022-05-22 10:17:33 [i] waiting for at least 2 node(s) to become ready in "my-nodes"
2022-05-22 10:17:33 [i] nodegroup "my-nodes" has 2 node(s)
2022-05-22 10:17:33 [i] node "ip-192-168-19-9.ap-south-1.compute.internal" is ready
2022-05-22 10:17:33 [i] node "ip-192-168-85-34.ap-south-1.compute.internal" is ready
2022-05-22 10:17:38 [i] kubectl command should work with "/home/vagrant/.kube/config", try 'kubectl get nodes'
2022-05-22 10:17:38 [✓] EKS cluster "geolocation-eks" in "ap-south-1" region is ready
[vagrant@jenkinshost-utrains ~]$

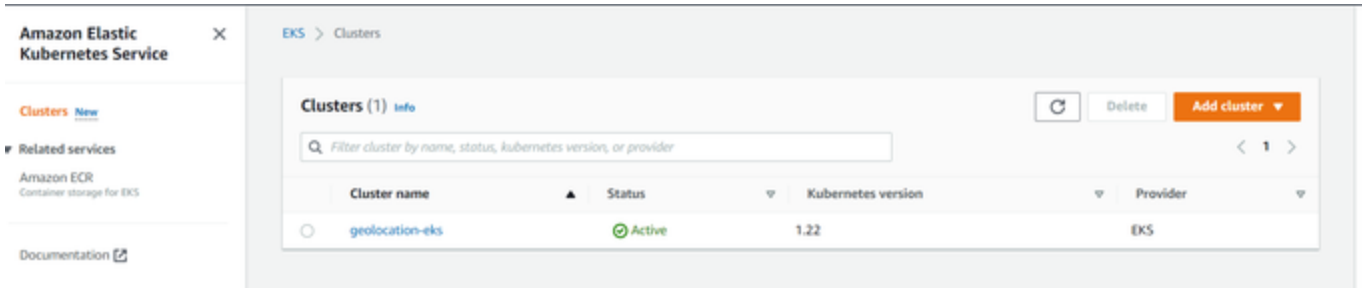
```

Command for EKS
cluster creation

EKS config
file location

Detail of the result
command : 2 nodes
ready, cluster name
and aws region

- See the result in AWS Console :
 - Login into aws console
 - Services type EKS in the search bar
 - Select the right region that you make in AWS CLI like default config region (for me, is ap-south-1)



5- Some commands to get information about our cluster

- This command should confirm that EKS cluster is up and running.

```
eksctl get cluster --name geolocation-eks --region ap-south-1
```

```
✓ TERMINAL
[vagrant@jenkinshost-utrains ~]$ eksctl get cluster --name geolocation-eks --region ap-south-1
2022-05-22 12:13:11 [i] eksctl version 0.98.0
2022-05-22 12:13:11 [i] using region ap-south-1
NAME                VERSION STATUS   CREATED                VPC                                SUBNETS
                    SECURITYGROUPS                PROVIDER
geolocation-eks 1.22    ACTIVE   2022-05-22T06:02:41Z  vpc-0fc581697438facbd  subnet-00046f94bb0732dae2a6a3d12479,subnet-0a93678ff8afa9f11,subnet-0edd1c5f66b28cf5c sg-062275efb07647dd0 EKS
[vagrant@jenkinshost-utrains ~]$
```

- Update the **Kubeconfig** by running below command:

```
aws eks update-kubeconfig --name geolocation-eks --region ap-south-1
```

```
✓ TERMINAL
[vagrant@jenkinshost-utrains ~]$ aws eks update-kubeconfig --name geolocation-eks --region ap-south-1
Added new context arn:aws:eks:ap-south-1:076892551558:cluster/geolocation-eks to /home/vagrant/.kube/config
[vagrant@jenkinshost-utrains ~]$
```

- You can view the **kubeconfig** file by running the below command:

```
cat /home/vagrant/.kube/config
```

▼ TERMINAL

```
[vagrant@jenkinshost-utrains ~]$ cat /home/vagrant/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMvakNI
    pNQjRyRFRJeU1EVXlNakEyTURjeU9Gb1hEVE15TURVeE9UQTJNRGN5T0Zvd0ZURVRNQkVHQTFVRQpBeI
    WUvqNEZ5YmFmWlRmSS9IS2liaUJlQTZ5b1F2Ym1KZHI1VG1jwXpXaG8rYks2SUlMSWZLRDMvdTJLcTBh
    VMSQovSTV4ZmVMNlZlXeuU11NGFuZUNOR1VrTEhUNlVlVTRBL25kSm1tR0dkcXl1jNmWzYlJrQ05xVGhTb
    c1lmaUU4RmYKVU53SVZndVJqTW9MdZJWY3YrRHloajRmZdQekdyV0pwa1VNM1lGOXI3TmhHRGZXaXpl
    trTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0hRWURWUjBPQkJZRUZHT05JN0RVaWcrdlB2ajJlKUU84Q
    SUxRbXpTQ3BsRk9NSHFiaVNVRAoyd0N3dFZiZwtOcwOWSkpTXBubGF1N1c0T3VLMW9BYWdtT2x3VFNF
    VlK2lHRTY2V3RacGtqYjJLaGZCTGxKL3oKOUxreWNxOEUE0Q3NJTtN6Unl6cGw0Zm9uUVdvRTFYWTVUL3
    eC9va2NpMFBlZHR2MkQ4V3V0T01FQnZoY1htUSThcm1iMmE2NGRvT0JwUU9HZGVmSkQ5MkpEUytrZThy
    0tCg==
    server: https://6264349EE98AE2D59A0ED892FACD50A7.gr7.ap-south-1.eks.amazonaws.com
  name: geolocation-eks.ap-south-1.eksctl.io
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMvakNI
    pNQjRyRFRJeU1EVXlNakEyTURjeU9Gb1hEVE15TURVeE9UQTJNRGN5T0Zvd0ZURVRNQkVHQTFVRQpBeI
    WUvqNEZ5YmFmWlRmSS9IS2liaUJlQTZ5b1F2Ym1KZHI1VG1jwXpXaG8rYks2SUlMSWZLRDMvdTJLcTBh
    VMSQovSTV4ZmVMNlZlXeuU11NGFuZUNOR1VrTEhUNlVlVTRBL25kSm1tR0dkcXl1jNmWzYlJrQ05xVGhTb
    c1lmaUU4RmYKVU53SVZndVJqTW9MdZJWY3YrRHloajRmZdQekdyV0pwa1VNM1lGOXI3TmhHRGZXaXpl
    trTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0hRWURWUjBPQkJZRUZHT05JN0RVaWcrdlB2ajJlKUU84Q
    SUxRbXpTQ3BsRk9NSHFiaVNVRAoyd0N3dFZiZwtOcwOWSkpTXBubGF1N1c0T3VLMW9BYWdtT2x3VFNF
    VlK2lHRTY2V3RacGtqYjJLaGZCTGxKL3oKOUxreWNxOEUE0Q3NJTtN6Unl6cGw0Zm9uUVdvRTFYWTVUL3
    eC9va2NpMFBlZHR2MkQ4V3V0T01FQnZoY1htUSThcm1iMmE2NGRvT0JwUU9HZGVmSkQ5MkpEUytrZThy
    0tCg==
    server: https://6264349EE98AE2D59A0ED892FACD50A7.gr7.ap-south-1.eks.amazonaws.com
  name: geolocation-eks.ap-south-1.eksctl.io
```

- To view the list of worker nodes as part of EKS cluster.

```
kubectl get nodes
kubectl get ns
```

▼ TERMINAL

```
[vagrant@jenkinshost-utrains ~]$ kubectl get nodes
NAME                                                    STATUS    ROLES    AGE   VERSION
ip-192-168-19-9.ap-south-1.compute.internal            Ready     <none>    130m  v1.22.6-eks-7d68063
ip-192-168-85-34.ap-south-1.compute.internal            Ready     <none>    130m  v1.22.6-eks-7d68063
[vagrant@jenkinshost-utrains ~]$ kubectl get ns
NAME                STATUS    AGE
default              Active    138m
kube-node-lease      Active    138m
kube-public          Active    138m
kube-system          Active    138m
[vagrant@jenkinshost-utrains ~]$
```

- We have created an **EKS cluster** from the command line of our Jenkins server
- we have checked with some commands that this cluster is working correctly
- we will see how to create an ECR for the next part of the mission

Step 3- Create ECR repo in AWS using AWS CLI

- In our previous project, we have seen how to create a repository in AWS using the AWS console
- Here, we will create an amazon ECR repository in a few commands by performing the following steps :
 - Open a terminal and connect you to our Jenkins Server.
 - Run the following command to create a repository in AWS ECR named **geolocation_ecr_rep**

```
aws ecr create-repository --repository-name geolocation_ecr_rep
```

- The result of this command is the **JSON below, which contains the information for the configuration of our pipeline**

JSON FOR ECR CONFIG

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:076892551558:repository/geolocation_ecr_rep",
    "registryId": "076892551558",
    "repositoryName": "geolocation_ecr_rep",
    "repositoryUri": "076892551558.dkr.ecr.us-east-1.amazonaws.com/geolocation_ecr_rep",
    "createdAt": "2022-05-22T13:07:39+04:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

☐ @ Class8 Please copy your **repositoryUri** value somewhere.

COPY THE repositoryUri value

```
▼ TERMINAL
[vagrant@jenkinshost-utrains ~]$ aws ecr create-repository --repository-name geolocation_ecr_rep
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:076892551558:repository/geolocation_ecr_rep",
    "registryId": "076892551558",
    "repositoryName": "geolocation_ecr_rep",
    "repositoryUri": "076892551558.dkr.ecr.us-east-1.amazonaws.com/geolocation_ecr_rep",
    "createdAt": "2022-05-22T13:07:39+04:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
[vagrant@jenkinshost-utrains ~]$
```

the URI of the ECR repository that will be used to configure the pipeline. Copy and save this value somewhere

- To view your repository which is created in AWS Console, just log in your account, then Services-> ECR in the search bar and select the right region

The screenshot shows the AWS Management Console interface for the Amazon Elastic Container Registry (ECR). The left sidebar contains navigation links for 'Private registry', 'Public registry', 'Repositories', 'Getting started', 'Documentation', and 'Public gallery'. The main content area is titled 'Amazon ECR > Repositories' and shows a list of 'Private repositories (4)'. The repositories listed are 'bar', 'devop_repository', 'geolocation_ecr_rep' (highlighted with a red circle), and 'my_docker_repo'. Each repository entry includes its name and URI. The URI for 'geolocation_ecr_rep' is '076892551558.dkr.ecr.us-east-1.amazonaws.com/geolocation_ecr_rep'.

Repository name	URI
bar	076892551558.dkr.ecr.us-east-1.amazonaws.com/bar
devop_repository	076892551558.dkr.ecr.us-east-1.amazonaws.com/devop_repository
geolocation_ecr_rep	076892551558.dkr.ecr.us-east-1.amazonaws.com/geolocation_ecr_rep
my_docker_repo	076892551558.dkr.ecr.us-east-1.amazonaws.com/my_docker_repo

- We created a repository for our docker images in **Amazon ECR using AWS CLI**
- We checked that this repository exists in AWS console
- We copied the **URI (repositoryUri)** of our repository somewhere

Step 4- Docker, Docker pipeline and Kubernetes CLI plug-ins are installed in Jenkins

In this step, we will start setting up our pipeline that will deploy our image in the EKS cluster

1- Install kubectl plugin on your Jenkins instance

- Login to your jenkins user interface, go to **Manage jenkins manage plugin available** and type **kubectl** in the search bar.

The screenshot shows the Jenkins Plugin Manager interface. The left sidebar contains links for 'Back to Dashboard' and 'Manage Jenkins'. The main area is titled 'Plugin Manager' and has tabs for 'Updates', 'Available', 'Installed', and 'Advanced'. A search bar in the top right contains the text 'kubectl'. Below the search bar, a table lists available plugins. The first entry is 'Kubernetes CLI 1.10.3', which has a checkbox checked. Below this entry is a link to 'Configure kubectl for Kubernetes'. At the bottom of the table, there are two buttons: 'Install without restart' and 'Download now and install after restart'. A red box with the text 'check this box, then click to Install Without restart' points to the checked checkbox. Another red box with the text 'Enter kubectl in the search bar' points to the search bar. A third red box with the text 'Update information obtained: ;' points to the bottom right of the interface.

Dashboard ▸ Plugin Manager

Back to Dashboard

Manage Jenkins

Plugin Manager

Updates Available Installed Advanced

Search: kubectl

Install	Name ↓
<input checked="" type="checkbox"/>	Kubernetes CLI 1.10.3
	kubectl
	Configure kubectl for Kubernetes

[Install without restart](#) [Download now and install after restart](#) Update information obtained: ;

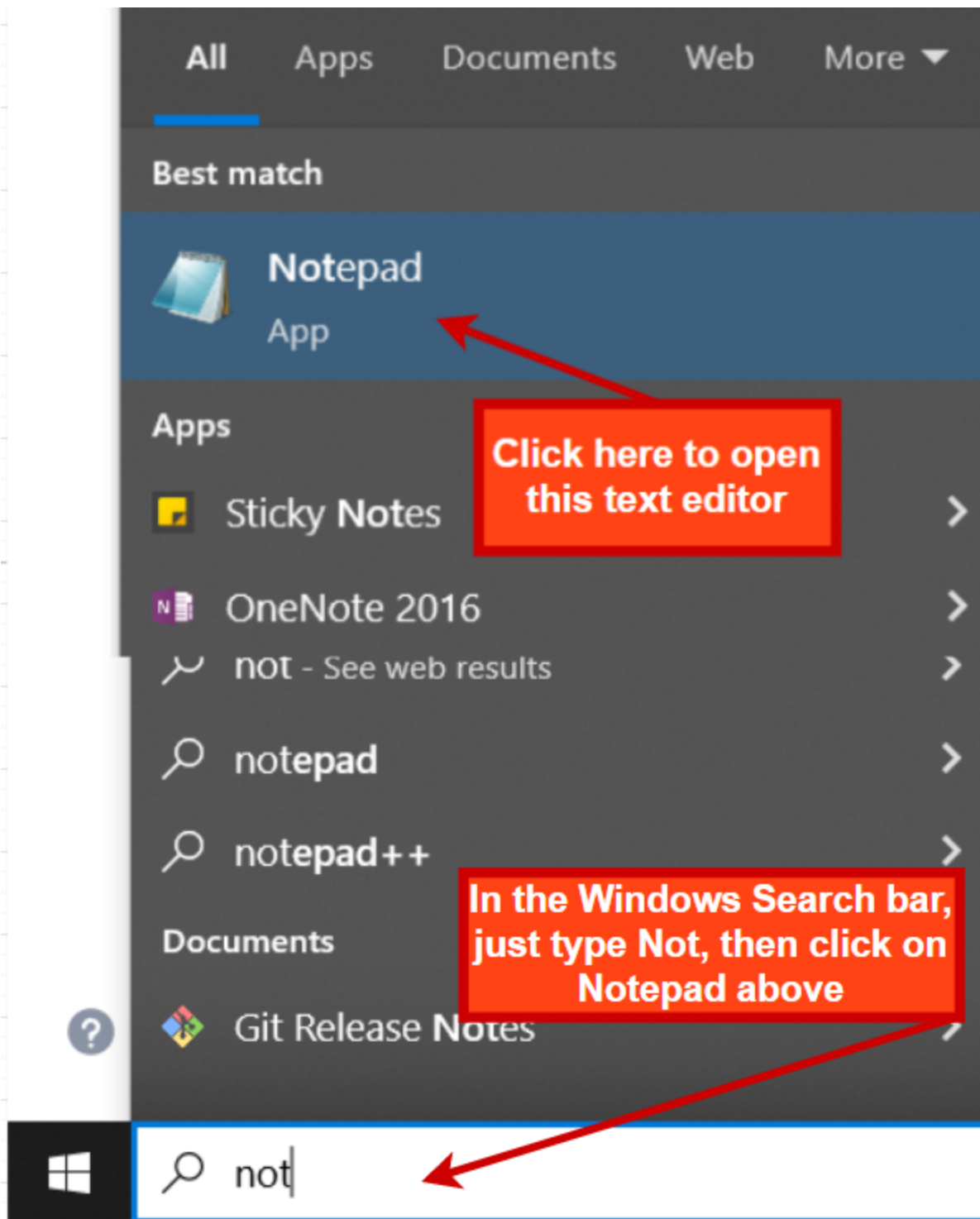
2- Create Credentials to connect to the Kubernetes Cluster using kubeconfig

First, we will copy and paste the EKS configuration file into a text file on our local machine :

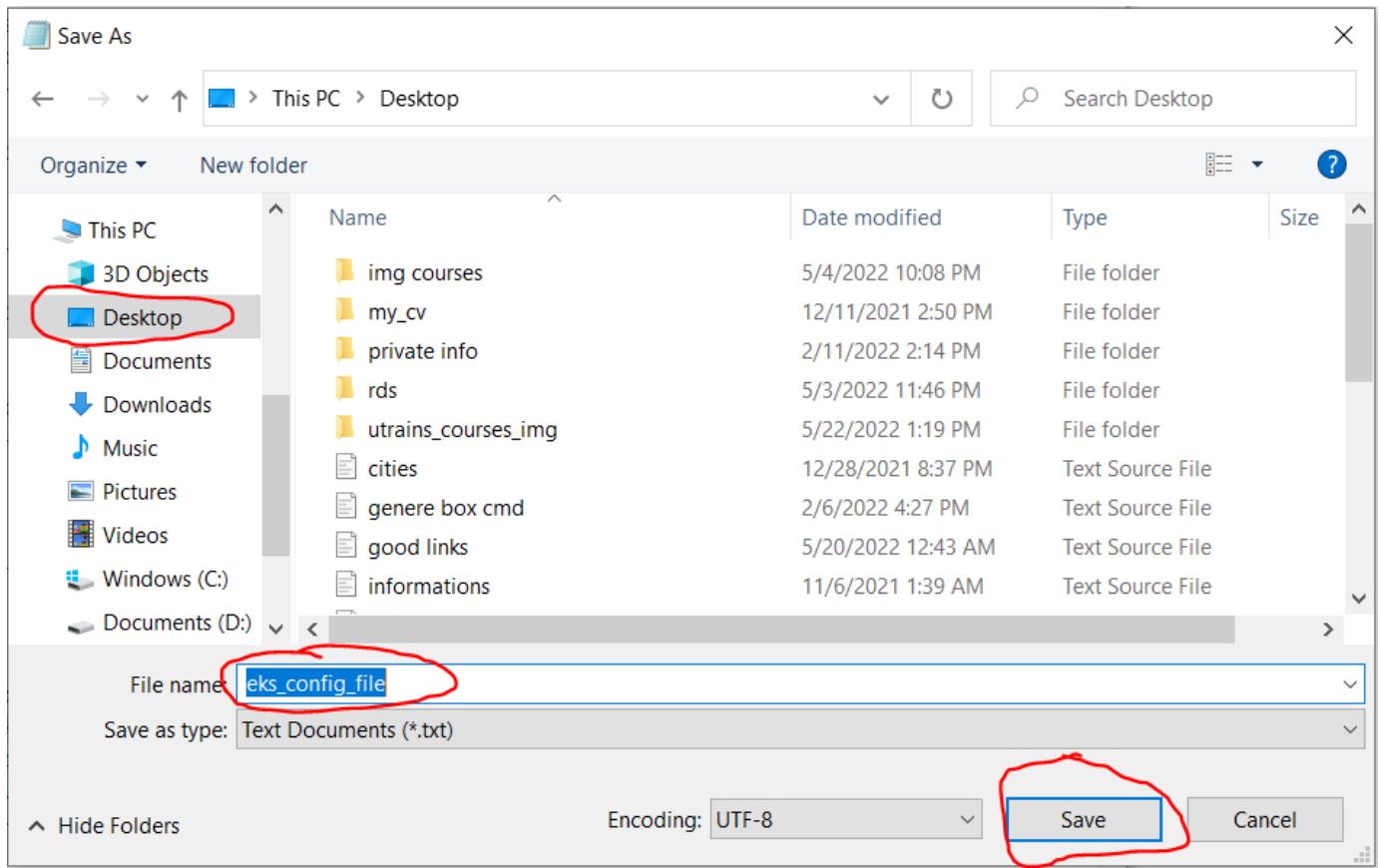
- Connect you to our Jenkins server, using the ssh command
- Run this command, then copy all `/home/vagrant/.kube/config` file content

```
cat /home/vagrant/.kube/config
```

- Select the content and make **Ctrl + Shift + c** keyboard combination for copy the **EKS** config file
- Open the text editor (Notepad in windows for example).



- Paste the result of the copy of our previous command and save the file on the desktop with the name **eks_config_file**



Now, let's configure the EKS Credentials for the connections

- Go to Jenkins **Manage jenkins** **Manage Credentials** **Global** then, click on **Add credentials**

The screenshot shows the Jenkins 'Add Credentials' form. The breadcrumb trail is 'Dashboard > Credentials > System > Global credentials (unrestricted)'. The left sidebar has 'Back to credential domains' and 'Add Credentials'. The form fields are: 'Kind' (Secret file), 'Scope' (Global (Jenkins, nodes, items, all child items, etc)), 'File' (Choose File, with 'eks_config_file' entered), 'ID' (eks_credential), and 'Description' (credential for EKS Access). The 'OK' button is at the bottom. Red callout boxes with arrows point to specific fields: 'click here then select Secret file' points to 'Kind'; 'click here, go to your desktop, then choose the eks_config_file that we are created' points to 'Choose File'; 'type the key id, next, put the description' points to 'ID'; and 'click here to save' points to 'OK'.

Jenkins

Dashboard > Credentials > System > Global credentials (unrestricted)

Back to credential domains

Add Credentials

Kind

Secret file

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

File

Choose File eks_config_file

ID ?

eks_credential

Description ?

credential for EKS Access

OK

click here then select Secret file

click here, go to your desktop, then choose the eks_config_file that we are created

type the key id, next, put the description

click here to save

3- Create a pipeline in Jenkins

- Go to **Jenkins**, then click on **New item**
- Put the name : **eks_ecr_geolocation**
- Select **Pipeline**, then click on **Ok** to create the **New Item**

Enter an item name

eks_ecr_geolocation

» Required field



Freestyle project

This is the central feature of Jenkins. Jenkins will build your project.



Maven project

Build a maven project. Jenkins takes advantage of your POM files a



Pipeline

Orchestrates long-running activities that can span multiple build ac



Multi-configuration project

Suitable for projects that need a large number of different configur

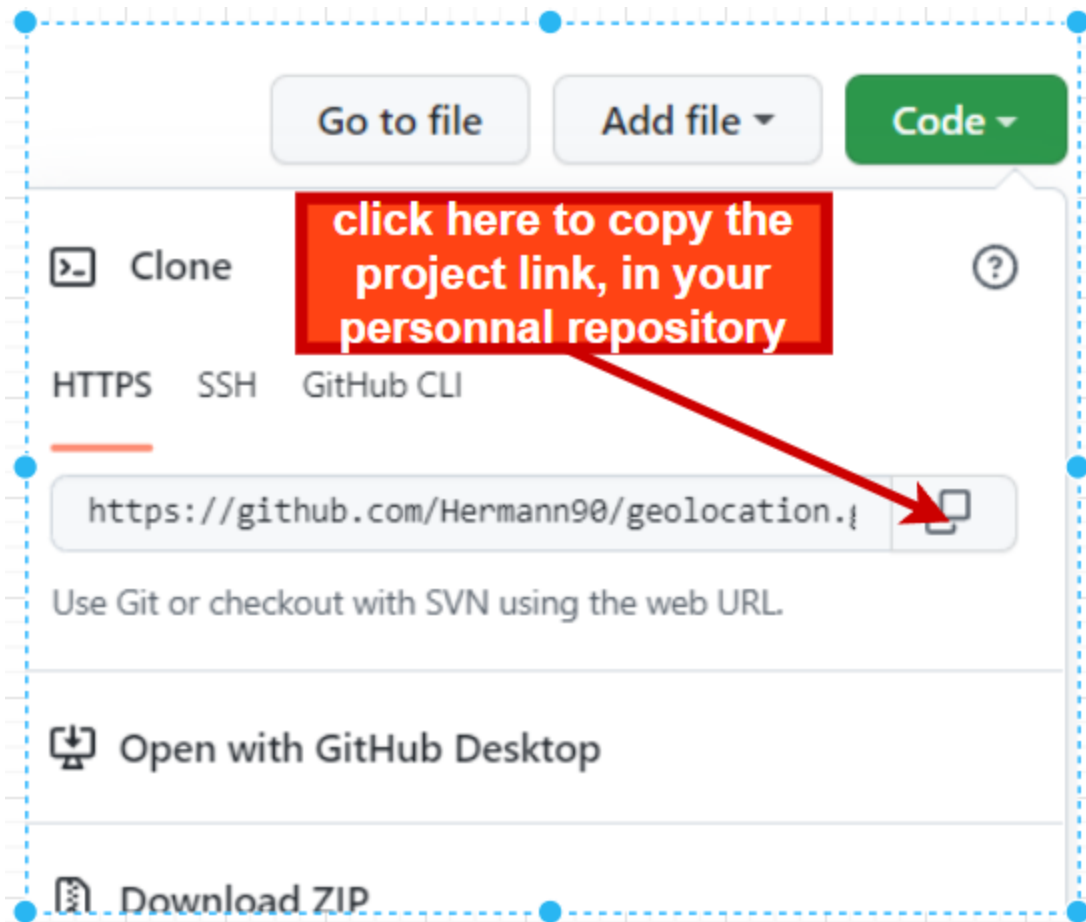
4- Pipeline configuration :

a- clone the geolocation repository

- We will clone geolocation repository in our repository.

Note : Everyone should have the **geolocation** project in their personal repository. because we have used this project to do sonar analysis

Open your repository, then copy the geolocation project link.



- Open the terminal in visual studio code, then create a folder called **jenkins_eks**

```
mkdir jenkins_eks
```

- Navigate in this folder, then clone the **geolocation** project, using the git clone command, then paste the link that you copied

```
cd jenkins_eks  
git clone https://github.com/Hermann90/geolocation.git
```

✓ TERMINAL

```
hermann90@DESKTOP-E5NS4D2 MINGW64 /d/courses/utrains_DevOps_part/jenkins
$ mkdir mkdir jenkins_eks
```

```
hermann90@DESKTOP-E5NS4D2 MINGW64 /d/courses/utrains_DevOps_part/jenkins
$ cd jenkins_eks/
```

```
hermann90@DESKTOP-E5NS4D2 MINGW64 /d/courses/utrains_DevOps_part/jenkins/jenkins_eks
$ git clone git@github.com:Hermann90/geolocation.git
Cloning into 'geolocation'...
remote: Enumerating objects: 2285, done.
remote: Counting objects: 100% (2285/2285), done.
remote: Compressing objects: 100% (1963/1963), done.
remote: Total 2285 (delta 286), reused 2280 (delta 284), pack-reused 0R
Receiving objects: 100% (2285/2285), 24.12 MiB | 2.80 MiB/s, done.
Resolving deltas: 100% (286/286), done.
Updating files: 100% (3433/3433), done.
```

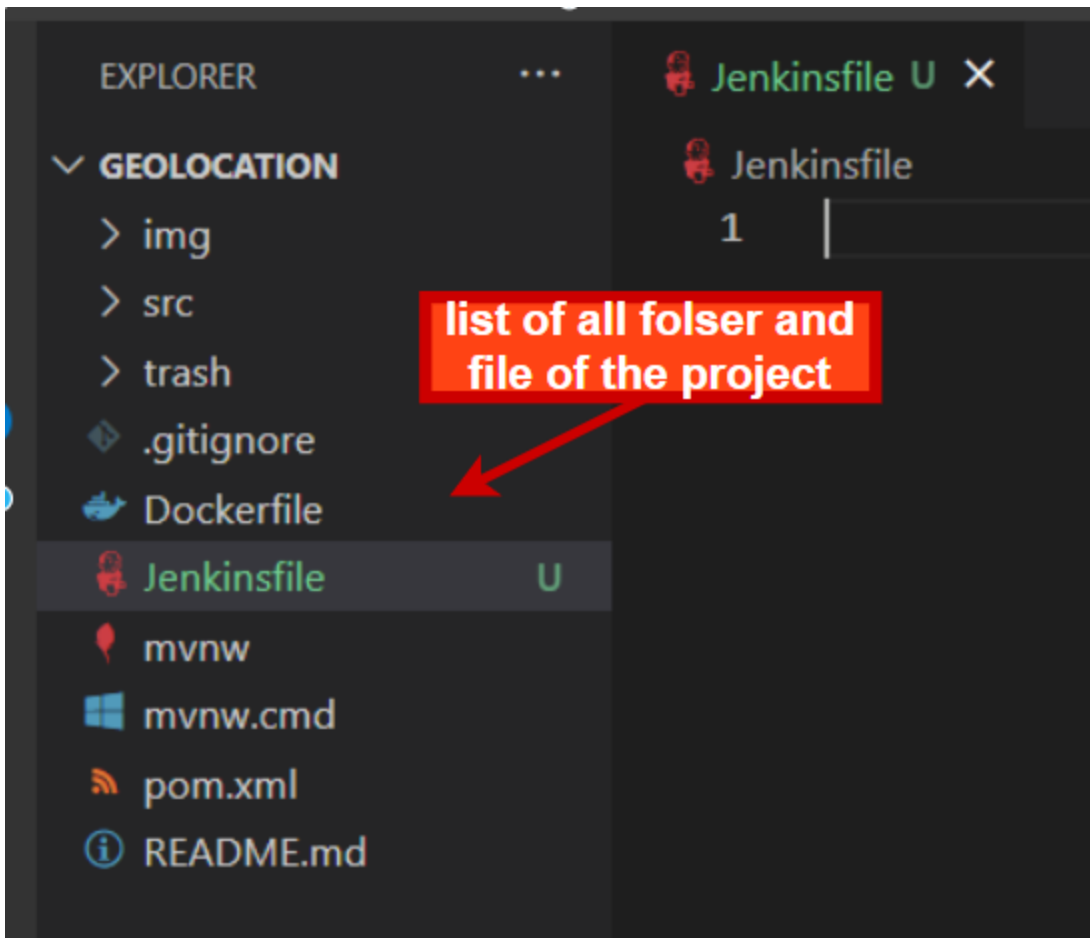
```
hermann90@DESKTOP-E5NS4D2 MINGW64 /d/courses/utrains_DevOps_part/jenkins/jenkins_eks
$ ls
geolocation/
```

command to open the whole
folder of our GitHub project

```
hermann90@DESKTOP-E5NS4D2 MINGW64 /d/courses/utrains_DevOps_part/jenkins/jenkins_eks
$ code geolocation/
```

- Open this project into Virtual Studio Code using **code** command

```
code geolocation
```



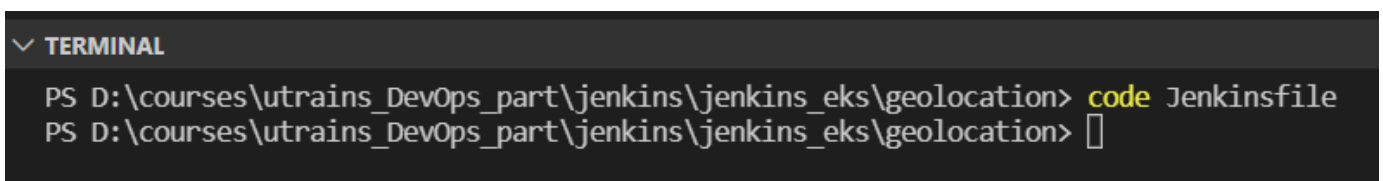
We will gradually create and add the files for our pipeline.

b- Jenkinsfile : 3 first stage (git checkout, maven clean install, and Test)

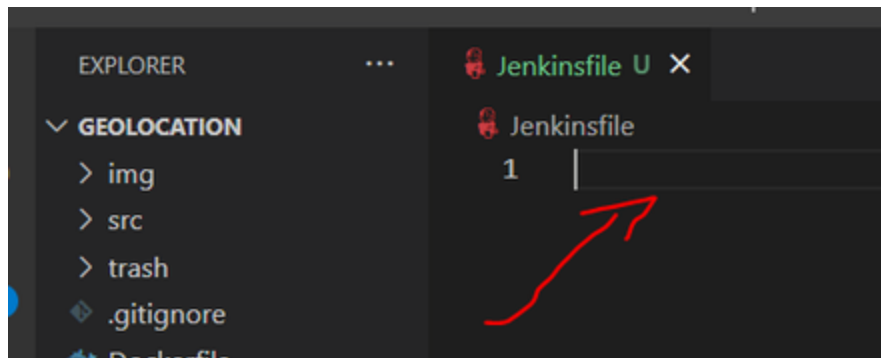
We will, here, open the Jenkinsfile in our **Virtual Studio Code**, then add the instructions for the first two stages of our pipeline: **checkout, and maven install stage**

- Open a new terminal in the geolocation folder that content our project.
- Create a Jenkinsfile using the code command

```
code Jenkinsfile
```



- In the window above the terminal, an area is displayed where you will enter the instructions of our declarative pipeline



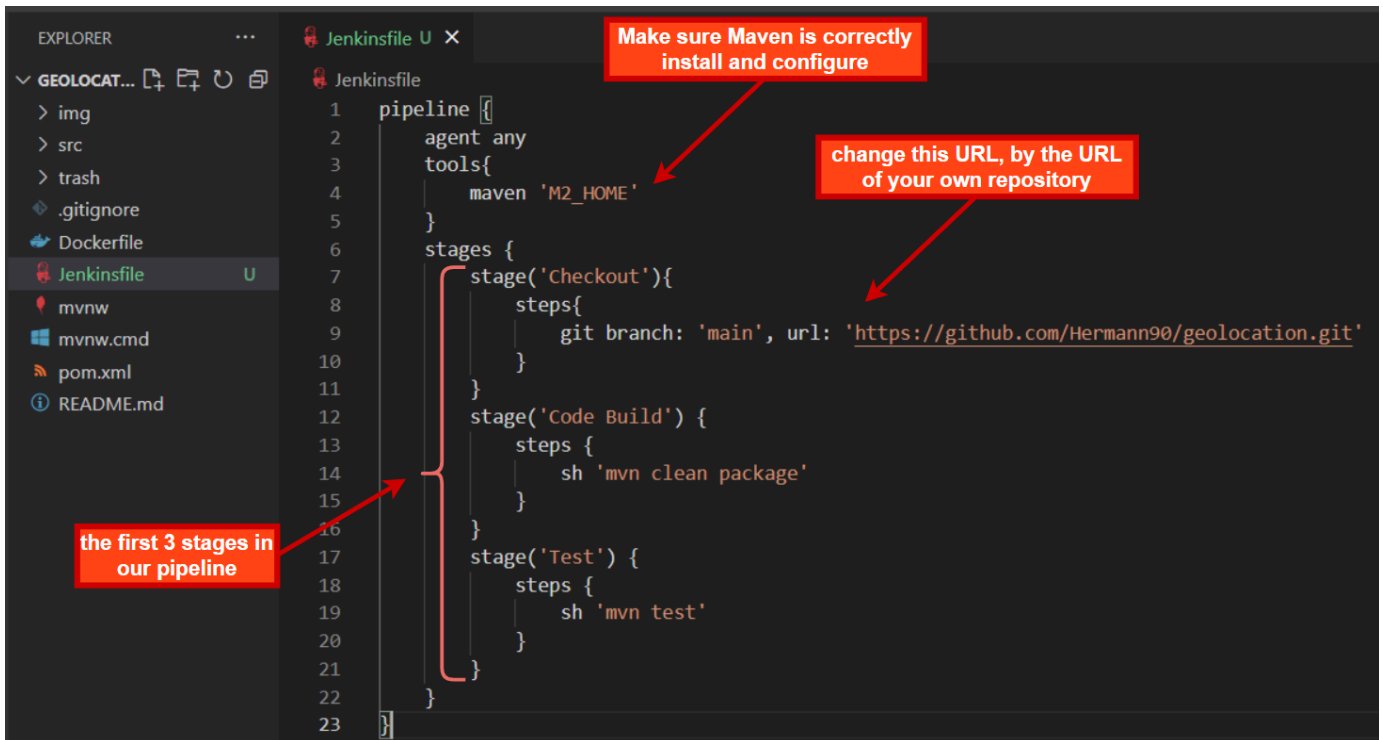
- Copy the content of the code below, change the URL of the repository, with your own URL.

☐ @ Class8 please change the URL value with your own repository URL.

```
pipeline {
  agent any
  tools{
    maven 'M2_HOME'
  }
  stages {
    stage('Checkout'){
      steps{
        git branch: 'main', url: 'https://github.com/Hermann90/geolocation.git'
      }
    }
    stage('Code Build') {
      steps {
        sh 'mvn clean package'
      }
    }
    stage('Test') {
      steps {
        sh 'mvn test'
      }
    }
  }
}
```

- Paste the script that you have copied in the Jenkinsfile.

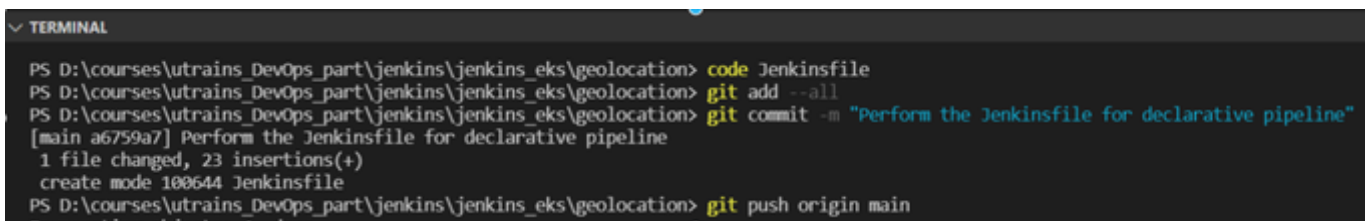
☐ @ Class8 please change the URL value with your own repository URL.



Commit Changes :

- Open the terminal in our repository folder follow these steps :
 - **git add command** : adding the changes in our repository
 - **git commit** : save the changes
 - **git push** : push our changes in your GitHub account

```
git add --all
git commit -m "Perform the Jenkinsfile for declarative pipeline"
git push origin main
```



- We will configure our pipeline in Jenkins, testing the first 3 phases of the pipeline before continuing.
- Doing it progressively makes us more efficient in solving eventual problems.

c- configure Jenkins pipeline

The steps for pipeline configuration are as follows:

- Open the Jenkins web interface and select our project (here it is the **eks_ecr_geolocation** project)
- Click on configure, then fill in the fields of the form, like in the picture below

☐ @ Class8 please change the URL value with your own repository URL.



Pipeline script from SCM

SCM ?

Git

Click Here then select Git

Repositories ?

Repository URL ?

https://github.com/Hermann90/geolocation.git

Credentials ?

- none -

Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Type "main" here for the main branch

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add

Script Path ?

Jenkinsfile

Type "Jenkinsfile" here for our declarative pipeline

☒ Lightweight checkout ?

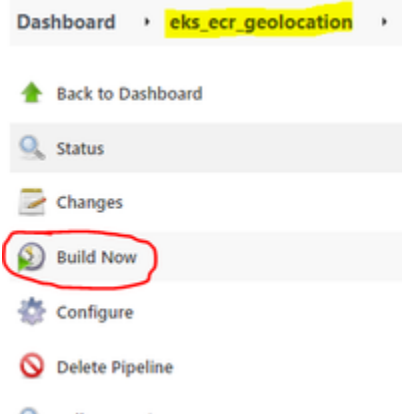
Pipeline Syntax

Save Apply

click on Apply, then save

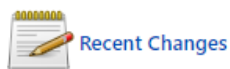
e- Test of the first three stages of the pipeline

- Now we can test the first three stages of our pipeline by clicking on **Build Now**.

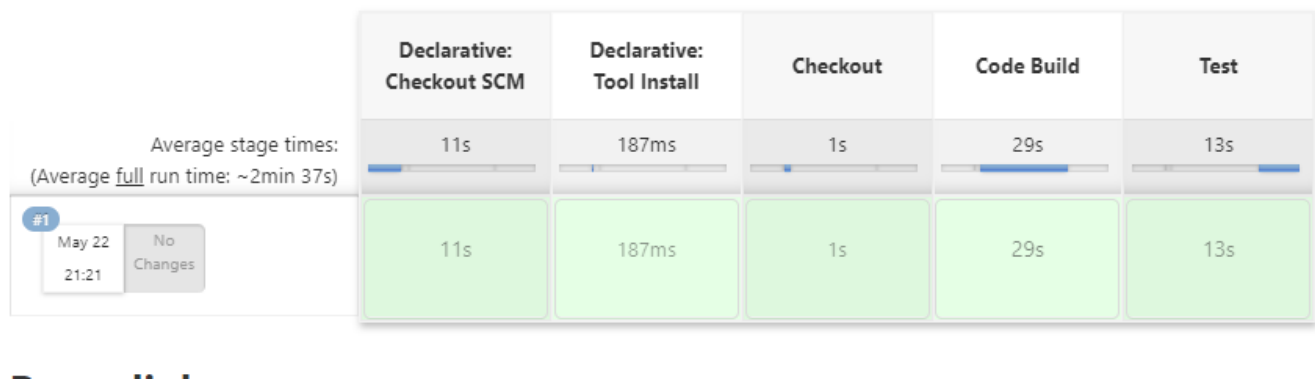


- When all goes well, we have the result below

Pipeline eks_ecr_geolocation



Stage View



5- Pipeline : build image and push image in ECR

In this step, we will add two new stages (build docker image and push image to ECR) to our Pipeline.

a- RepositoryUri value

- When we created our ECR repository, the result was a JSON file.
- We copied and saved the value of **repositoryUri** somewhere.
- We need this value here for the rest of our configuration.
- If you forgot, click on the link below to see the step where we copied this value.

[Click here to see the value of repositoryUri](#)

- We will add a section of code after the tools section, which will contain the environment variables
- The registry variable, must contain the value of **repositoryUri** that we have copied.
- The code will look like this

```

environment {
    registry = '076892551558.dkr.ecr.us-east-1.amazonaws.com
/geolocation_ecr_rep'
    registryCredential = 'jenkins-ecr'
    dockerimage = ''
}

```

- Let's add this bellow script between the line 21 and 22 of our Jenkinsfile,
- the script for the 4th and 5th stage which will result in pushing the image in our repository that we have created in amazon ECR

```

// Building Docker images
stage('Building image') {
    steps{
        script {
            dockerImage = docker.build registry +
"$BUILD_NUMBER"
        }
    }
}
// Uploading Docker images into AWS ECR
stage('Pushing to ECR') {
    steps{
        script {
            sh 'aws ecr get-login-password --region us-east-2 |
docker login --username AWS --password-stdin account_id.dkr.ecr.us-east-
2.amazonaws.com'
            sh 'docker push account_id.dkr.ecr.us-east-2.
amazonaws.com/my-docker-repo:latest'
        }
    }
}

```

- Don't forget to replace the value of **repositoryUri** with your own
- At the end, our Jenkinsfile will look like this.



@ Class8 Please change the registry value with your own value, Do the same with the line 38 and 39. Change 076892551558 value with your own account id,

```

pipeline {
    agent any
    tools{
        maven 'M2_HOME'
    }
    environment {
        registry = '076892551558.dkr.ecr.us-east-1.amazonaws.com
/geolocation_ecr_rep'
        dockerimage = ''
    }
    stages {
        stage('Checkout'){
            steps{
                git branch: 'main', url: 'https://github.com/Hermann90
/geolocation.git'
            }
        }
        stage('Code Build') {
            steps {
                sh 'mvn clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
        }
        // Building Docker images
        stage('Building image') {
            steps{
                script {
                    dockerImage = docker.build registry
                }
            }
        }
        // Uploading Docker images into AWS ECR
        stage('Pushing to ECR') {
            steps{
                script {
                    sh 'aws ecr get-login-password --region us-east-1 |
docker login --username AWS --password-stdin 076892551558.dkr.ecr.us-
east-1.amazonaws.com'
                    sh 'docker push 076892551558.dkr.ecr.us-east-1.
amazonaws.com/geolocation_ecr_rep:latest'
                }
            }
        }
    }
}

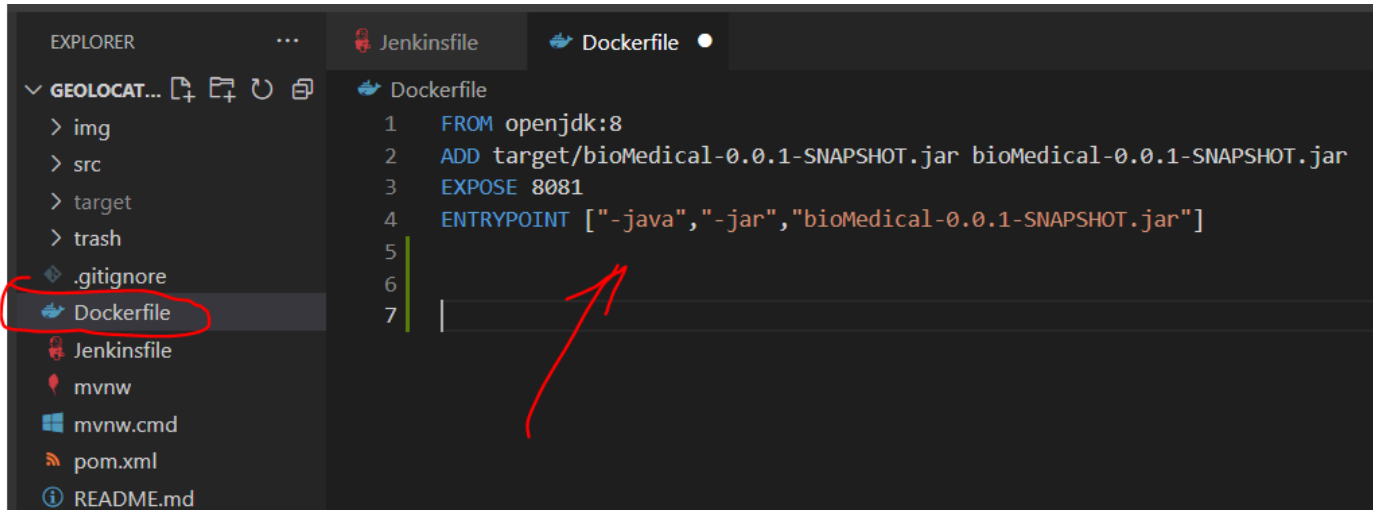
```

b- create or update Dockerfile content

- Open the Dockerfile of your repository,
- Empty its contents, then copy and paste the script below into this Dockerfile.

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

• Dockerfile content



c- Get access to some file in the Jenkins server

- Open the terminal and connect you to the Jenkins server
- Run these commands

```
sudo chmod 777 /var/run/docker.sock
sudo usermod -aG docker jenkins
sudo usermod -aG vagrant jenkins

sudo cp -r ~/.aws/ /var/lib/jenkins/
cd /var/lib/jenkins/.aws
sudo chown -R jenkins ./
```

d- Commit Changes :

- Open the terminal in our repository folder follow these steps :
 - **git add** command : adding the changes in our repository
 - **git commit** : save the changes
 - **git push** : push our changes in your GitHub account

```
git add --all
git commit -m "Perform the Jenkinsfile for declarative pipeline"
git push origin main
```

At this stage, we do a commit and a push on our changes.

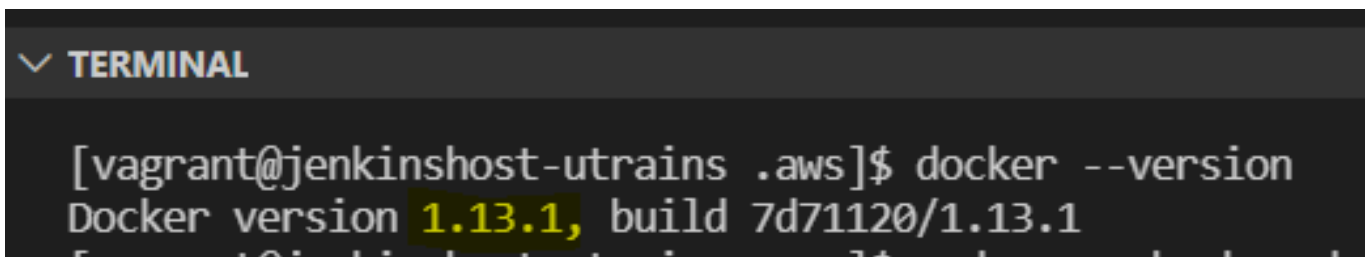
In Jenkins when we try to build, in order to test if the pipeline works, we have the following error:

- **unknown flag: --password-stdin : the system says here that the flag is not recognized.**

```
[Pipeline] sh
+ aws ecr get-login-password --region us-east-1
+ docker login --username AWS --password-stdin 076892551558.dkr.ecr.us-east-1.amazonaws.com
unknown flag: --password-stdin
See 'docker login --help'.
Exception ignored in: <_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>
BrokenPipeError: [Errno 32] Broken pipe
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 125
Finished: FAILURE
```

- So, you can't connect to ECR using AWS CLI and docker
- After several checks, it appears that this bug is caused by the version of docker currently installed on the server.
- the command above shows that we have version 1.13.1 installed while the current version of Docker is 20.00

```
docker --version
```



```
✓ TERMINAL
[vagrant@jenkinshost-utrains .aws]$ docker --version
Docker version 1.13.1, build 7d71120/1.13.1
```

e- Solve issues :

- To solve this problem, we will upgrade our Docker version installed in Jenkins :
- Stop docker service
- Remove the installed version

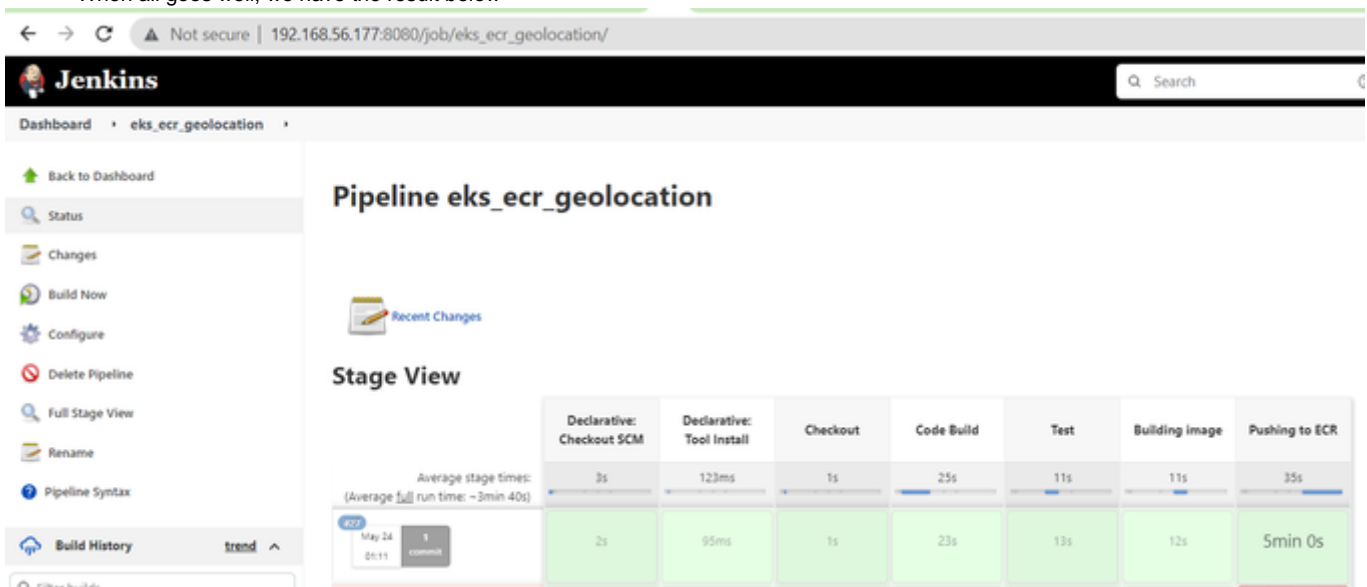
```
sudo systemctl stop docker
sudo yum remove docker docker-client docker-client-latest docker-common
docker-latest docker-latest-logrotate docker-logrotate docker-selinux
docker-engine-selinux docker-engine
```

- Install the latest version of docker
- Start this version and enable it

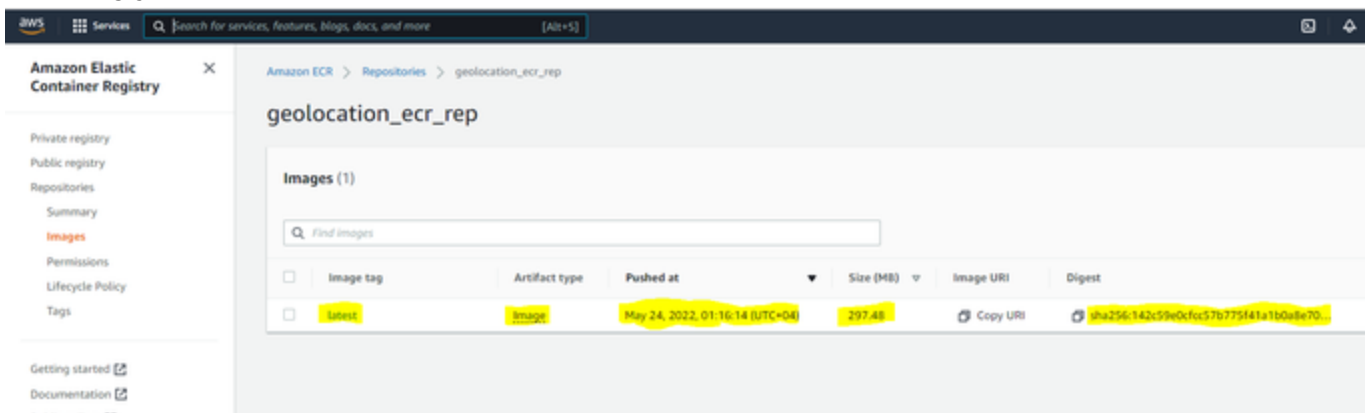
```
curl -fsSL https://get.docker.com/ | sh
sudo systemctl start docker
sudo systemctl status docker
```

f- Test of the first three stages of the pipeline

- Now we can test the first three stages of our pipeline by clicking on **Build Now**.
- The last step may take a little longer, depending on your internet connection
- When all goes well, we have the result below



- At this level, the docker image is in ECR,
- To see this image, just open your AWS console, then go to ECR and select the region where you created your ECR repository using AWS CLI.



- We just have to deploy this image in our EKS cluster

6- Deploying the image in EKS

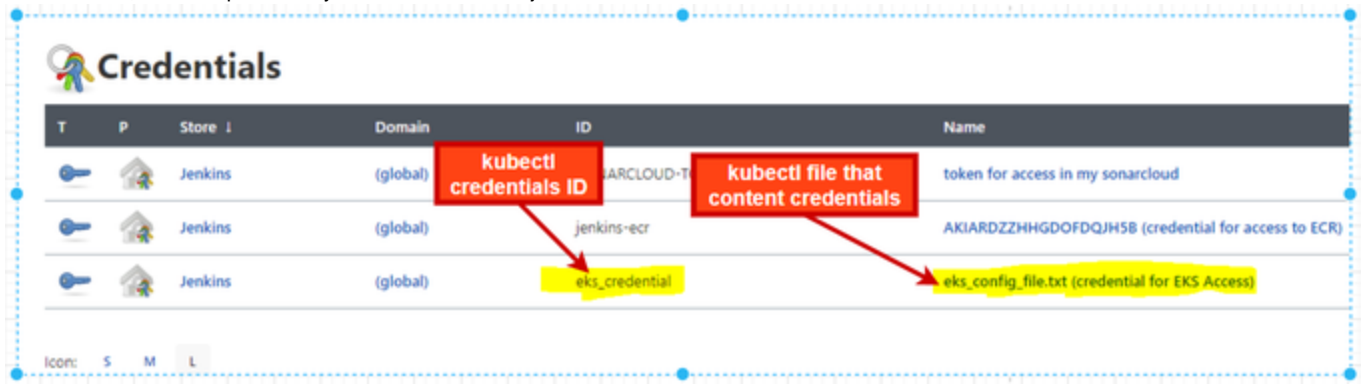
At this level, we will deploy the image of our application in **EKS** :

- In this part, we will use the **yaml** file, to automate the deployment of our image which is in ECR, in the EKS cluster, created above.
- To do this, we will create in our repository, a **yaml** file called **eks_deploy_from_ecr.yaml**, then write the instructions for the deployment of our image.
- Then, we will use the command **kubectl**, to launch the execution of this **yaml** file in our pipeline jenkins

a- Credentials config : manage jenkins Manage Credentials Global

For this part, we have already imported the kubectl configuration file into the jenkins web interface.

As a reminder, we have put this key in the credentials of jenkins



b- creation of the stage for the deployment in EKS cluster

Let's open our Jenkinsfile in Virtual Studio Code, then add the stage to deploy our application in EKS cluster.

- Open the Jenkinsfile in Virtual Studio Code
- Between the line 42 and 43 add the code below, which will allow to deploy the image of our application in the EKS cluster.

```
//deploy the image that is in ECR to our EKS cluster
stage ("Kube Deploy") {
    steps {
        withKubeConfig([credentialsId: 'eks_credential',
serverUrl: '']) {
            sh "kubectl apply -f eks_deploy_from_ecr.yaml"
        }
    }
}
```

- Note that this stage calls for a Yaml file, and using our credentials ID.
- So afterwards, we will create this Yaml file in our repository

c- Creation of the Yaml file

- Open the repository in virtual studio code and open the terminal in the root directory of the project (here, in the **geolocation** folder)
- Create a file with the code command

```
code eks_deploy_from_ecr.yaml
```

```
▼ TERMINAL

remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:Hermann90/geolocation.git
PS D:\courses\utrains_DevOps_part\jenkins\jenkins_eks\geolocation> ls

Directory: D:\courses\utrains_DevOps_part\jenkins\jenkins_eks\geolocation


Mode                LastWriteTime         Length Name
----                -
d-----          5/22/2022   7:27 PM             img
d-----          5/22/2022   7:27 PM             src
d-----          5/22/2022  11:21 PM            target
d-----          5/22/2022   7:27 PM             trash
-a-----          5/22/2022   7:27 PM           332 .gitignore
-a-----          5/22/2022  11:33 PM           160 Dockerfile
-a-----          5/24/2022   2:06 AM           1284 Jenkinsfile
-a-----          5/22/2022   7:27 PM           9400 mvnw
-a-----          5/22/2022   7:27 PM           5972 mvnw.cmd
-a-----          5/22/2022   7:27 PM           2610 pom.xml
-a-----          5/22/2022   7:27 PM            122 README.md

PS D:\courses\utrains_DevOps_part\jenkins\jenkins_eks\geolocation> code eks_deploy_from_ecr.yaml
```

Annotations:

- project root folder (points to the directory path)
- project sources code (points to the file listing)
- Command for Yaml file creation (points to the `code` command)

- Copy the content of the script below, then paste it in the file you just created in Virtual Studio code.



@ Class8

Don't forget to replace the link of the docker image in ECR with your own.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: geolocation-ecr-rep-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: geolocation-ecr-rep-app
  template:
    metadata:
      labels:
        app: geolocation-ecr-rep-app
    spec:
      containers:
        - name: geolocation-ecr-rep-app
          image: 076892551558.dkr.ecr.us-east-1.amazonaws.com
            /geolocation_ecr_rep:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 8082
# service type loadbalancer
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: geolocation-ecr-rep-app
    k8s-app: geolocation-ecr-rep-app
  name: geolocation-ecr-rep-app
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 8082
  type: LoadBalancer
  selector:
    app: geolocation-ecr-rep-app

```

d- Summary

To get to the end, before we go into the tests, here are the contents of the final two files that we have modified.

- **Jenkinsfile final content**

```

pipeline {
  agent any

```

```

tools{
    maven 'M2_HOME'
}
environment {
    registry = '076892551558.dkr.ecr.us-east-1.amazonaws.com
/geolocation_ecr_rep'
    dockerimage = ''
}
stages {
    stage('Checkout'){
        steps{
            git branch: 'main', url: 'https://github.com/Hermann90
/geolocation.git'
        }
    }
    stage('Code Build') {
        steps {
            sh 'mvn clean package'
        }
    }
    stage('Test') {
        steps {
            sh 'mvn test'
        }
    }
    // Building Docker images
    stage('Building image') {
        steps{
            script {
                dockerImage = docker.build registry
            }
        }
    }
    // Uploading Docker images into AWS ECR
    stage('Pushing to ECR') {
        steps{
            script {
                sh 'aws ecr get-login-password --region us-east-1 |
docker login --username AWS --password-stdin 076892551558.dkr.ecr.us-
east-1.amazonaws.com'
                sh 'docker push 076892551558.dkr.ecr.us-east-1.
amazonaws.com/geolocation_ecr_rep:latest'
            }
        }
    }
    //deploy the image that is in ECR to our EKS cluster
    stage ("Kube Deploy") {
        steps {
            withKubeConfig(caCertificate: '', clusterName: '',
contextName: '', credentialsId: 'eks_credential', namespace: '',

```

```

serverUrl: '' ) {
    sh "kubectl apply -f eks-deploy-from-ecr.yaml"
}
}
}
}
}
}
}

```

- **Dockerfile final content**

```

FROM openjdk:11.0.7
ADD target/bioMedical-0.0.1-SNAPSHOT.jar bioMedical-0.0.1-SNAPSHOT.jar
EXPOSE 80
ENTRYPOINT ["java","-jar","bioMedical-0.0.1-SNAPSHOT.jar"]

```

- **application.properties file :**

This file is essential for Java applications because it contains the key parameters of the application, like the parameters of connection to the database.

```

# =====
# TOMCAT
# =====
#server.address=127.0.0.1
#server.error.whitelabel.enabled=false
#server.tomcat.accesslog.enabled=true

# =====
# 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

#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

```

e- Commit the changes

At the end of all our modifications, we have to save everything, make a git add, a git commit and a git push

```

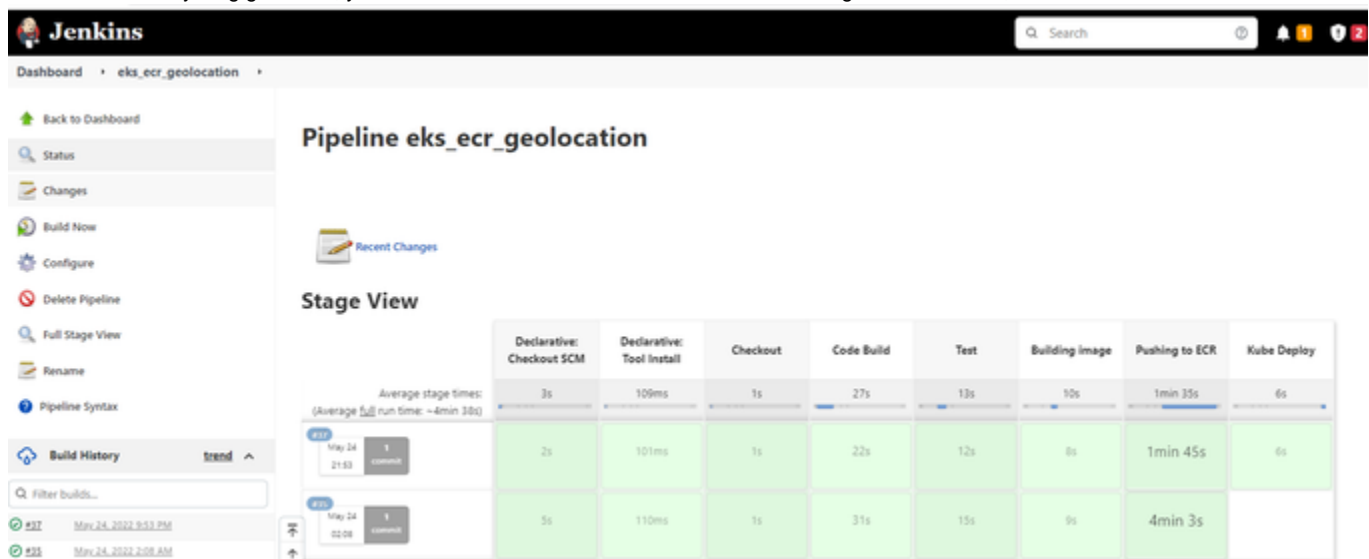
PS D:\courses\utrains_DevOps_part\jenkins\jenkins_eks\geolocation> git add --all
PS D:\courses\utrains_DevOps_part\jenkins\jenkins_eks\geolocation> git commit -m "Perform the Jenkinsfile for declarative pipeline"
[main 84d7a05] Perform the Jenkinsfile for declarative pipeline
2 files changed, 9 insertions(+), 9 deletions(-)
rename eks_deploy_from_ecr.yaml => eks-deploy-from-ecr.yaml (64%)
PS D:\courses\utrains_DevOps_part\jenkins\jenkins_eks\geolocation> git push origin main
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 16 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 723 bytes | 723.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:Hermann90/geolocation.git
c4ce406..84d7a05 main -> main
PS D:\courses\utrains_DevOps_part\jenkins\jenkins_eks\geolocation>

```

7- Testing the project

a- Testing the pipeline in Jenkins

- Open the jenkins web interface select the project **eks_ecr_geolocation**, then click on **Build Now**
- When everything goes well, you have the result below. which means that the image has been built, then sent to ECR.



b- Check in the AWS Console if our resources are created

- AWS ECR
- Go to Services -> ECR to view the image of our application

Image tag	Artifact type	Pushed at	Size (MB)	Image URI
latest	Image	May 24, 2022, 21:55:11 (UTC+04)	297.52	Copy URI
<untagged>	Image	May 24, 2022, 21:40:55 (UTC+04)	297.49	Copy URI

c- Open EKS in AWS Console

- Click on services Amazon Elastic Kubernetes Service

- Choose the region where we created our cluster : for my case, it's **ap-south-1**
- We can see the information below

The screenshot shows the AWS Management Console for the 'geolocation-eks' cluster. The region is 'ap-south-1'. The cluster is in an 'Active' state. Two nodes are listed in the 'Nodes (2)' section, both in a 'Ready' state. Red arrows point to the region, the cluster status, and the nodes.

Node name	Instance type	Node group	Created	Status
ip-192-168-19-9.ap-south-1.compute.internal	t3.small	my-nodes	Created May 22, 2022, 10:16 (UTC+04:00)	Ready
ip-192-168-85-34.ap-south-1.compute.internal	t3.small	my-nodes	Created May 22, 2022, 10:16 (UTC+04:00)	Ready

d- Test in Jenkins terminal

Connect to Jenkins via ssh. Let's run some kubectl commands to see how the cluster works :

- **Verify pods, deployments, services to K8S**

```
kubectl get pods
```

Here, we can see that 3 pods are running. this allows to manage the load and the availability of the application according to the request flow

```

vagrant@jenkinshost-utrains ~]$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
geolocation-ecr-rep-deployment-b867889fb-968r4  1/1     Running   0           5m45s
geolocation-ecr-rep-deployment-b867889fb-hlmm4  1/1     Running   0           6m1s
geolocation-ecr-rep-deployment-b867889fb-k4nfb  1/1     Running   0           5m30s
vagrant@jenkinshost-utrains ~]$

```

```
kubectl get deployments
```

```

vagrant@jenkinshost-utrains ~]$ kubectl get deployments
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
geolocation-ecr-rep-deployment      3/3     3             3           47h
vagrant@jenkinshost-utrains ~]$

```

```
kubectl get services
```


- This command is very important, it allows us to know the URL and the port on which our load balancer is listening. This allows us to redirect the requests to our different pods that are running.

```

TERMINAL
[vagrant@jenkinshost-utrains ~]$ kubectl get services
NAME                                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
geolocation-ecr-rep-app             LoadBalancer 10.100.68.42   a88bbd3b055324e3e8396450658f4e57-2042907886.ap-south-1.elb.amazonaws.com 80:30272/TCP 47h
kubernetes                           ClusterIP    10.100.0.1     <none>         443/TCP    4d10h

```

URL and listening port of the application

Copy this URL followed by: and port 80 to test the application in your favorite browser

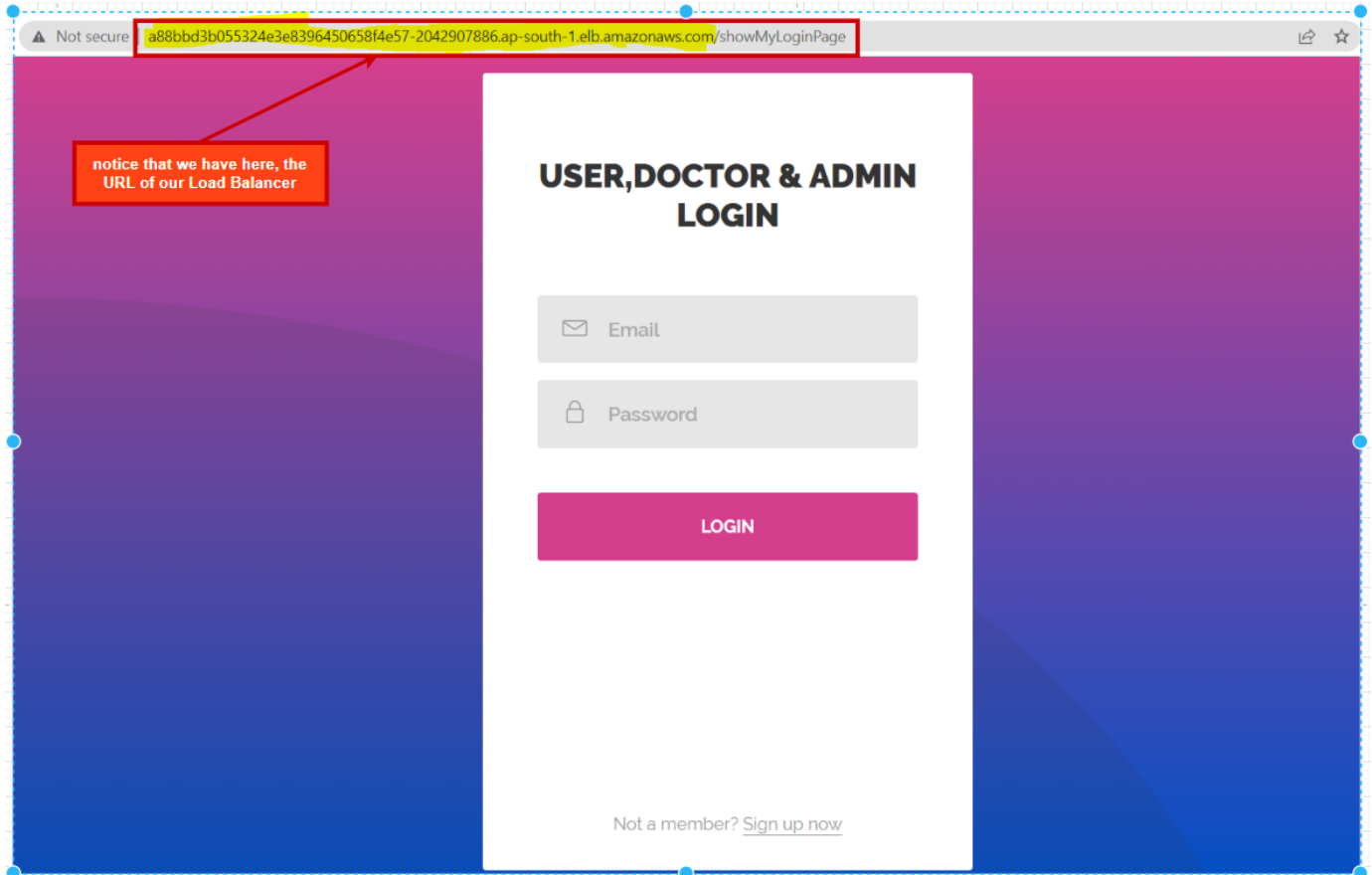
e- Test the application in the browser

- At this level we have deployed our application directly.
- The end users can now discover the new features implemented by the developers in real time.
- To do this copy the URL of the Load Balancer address followed by: and port 80 to start the application.
- In my case, we will have the following URL

<http://a88bbd3b055324e3e8396450658f4e57-2042907886.ap-south-1.elb.amazonaws.com:80>

When we click on the URL, or type it in our browser, we have the following login page

Login page



Some test accounts

- Below are some accounts to log in and test the applications
 - Admin account :
 - Email : **student@utrains.test**, Password : **school1**, Role : **ADMIN**
 - Email : **alain-pierre2@utrains.test**, password : **alain_pass**, Role : **ADMIN**
 - Email : **serge@utrains.test**, password : **serge_pass**, Role : **ADMIN**
 - Doctor account :
 - Email : **mekano2@utrains.test**, password : **mekano_pass**, Role : **DOCTOR**
 - User account :

- Email : **root@utrains.test**, password : **root_pass**, Role : **USER**

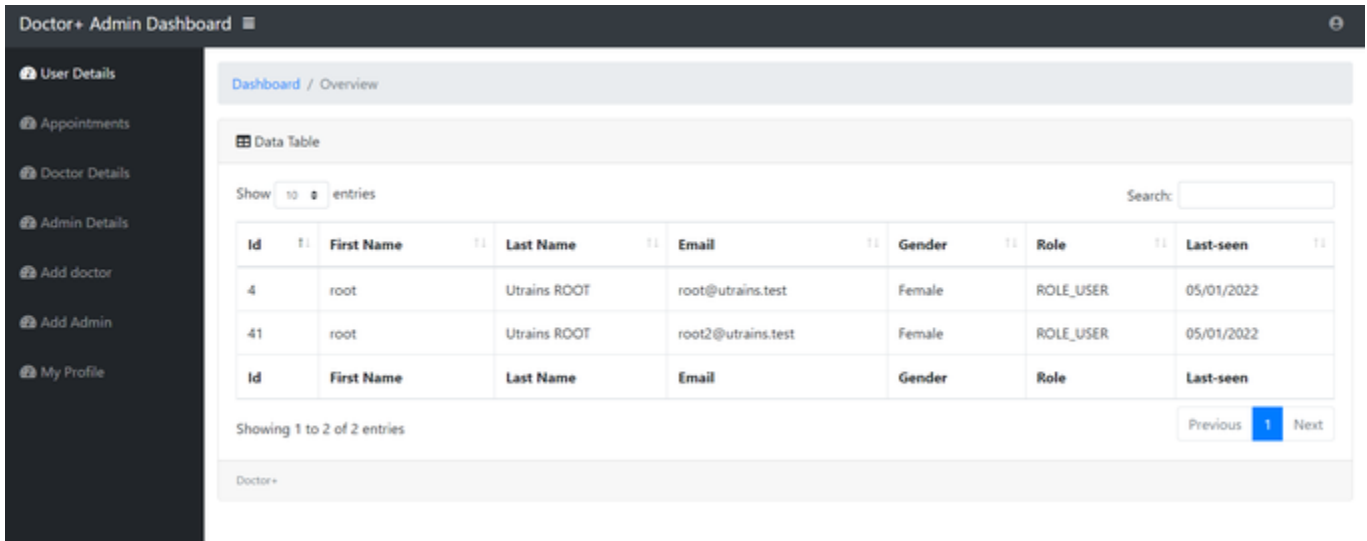
- We see that in our system we have 3 roles: **DOCTOR**, **ADMIN** and **USER**. according to these roles, we have specific **authorizations** and functions in the software

let's see what the **ADMIN** and **USER** roles allow us to see in the system

Some Interfaces in the app

- We are going to use one of the users with the role **ADMIN** to connect to the Internet.
 - Open the application in the browser using load balancer link
 - Email : **serge@utrains.test**, password : **serge_pass**

Admin home page



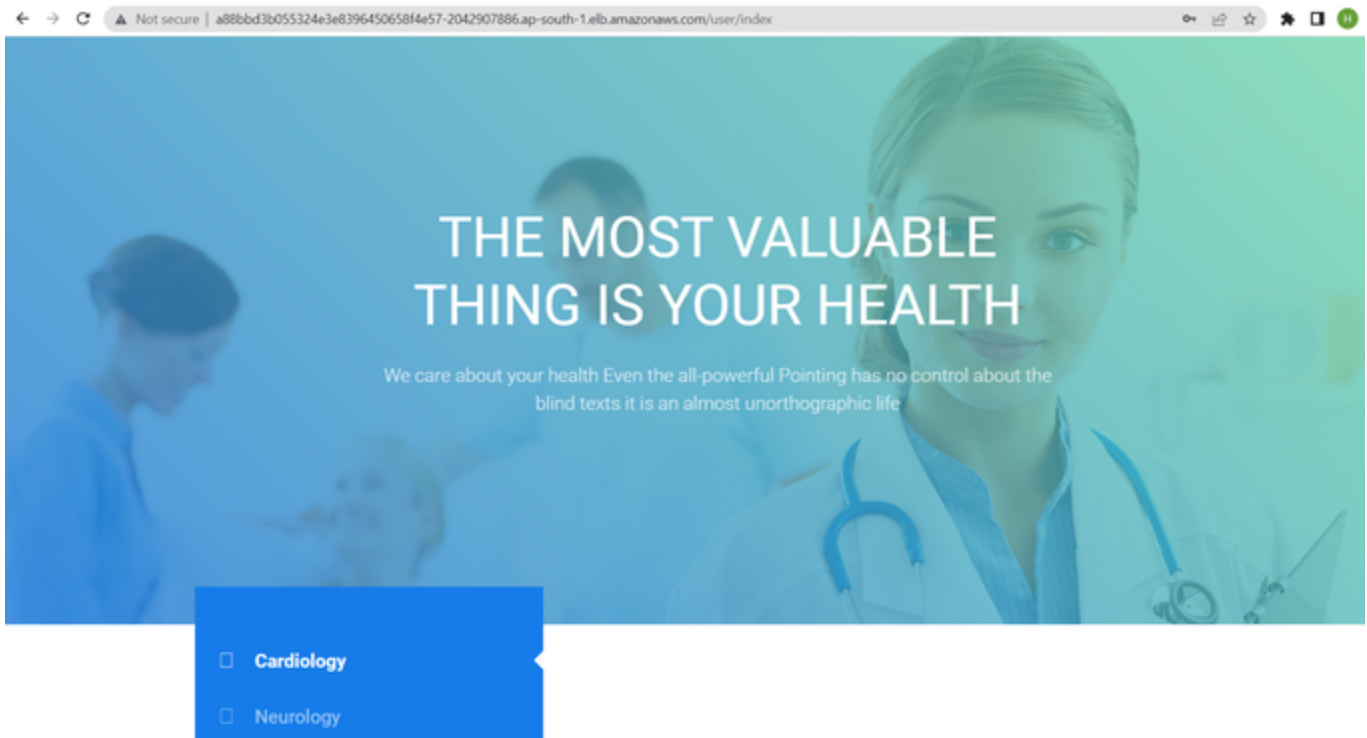
The screenshot shows the 'Doctor+ Admin Dashboard' with a sidebar menu on the left containing options like 'User Details', 'Appointments', 'Doctor Details', 'Admin Details', 'Add doctor', 'Add Admin', and 'My Profile'. The main content area displays a 'Data Table' with the following data:

Id	First Name	Last Name	Email	Gender	Role	Last-seen
4	root	Utrains ROOT	root@utrains.test	Female	ROLE_USER	05/01/2022
41	root	Utrains ROOT	root2@utrains.test	Female	ROLE_USER	05/01/2022

Below the table, it indicates 'Showing 1 to 2 of 2 entries' and includes pagination controls for 'Previous', '1' (current page), and 'Next'.

- We are going to use one of the users with the role **USER** to connect to the Internet.
 - Logout from the Admin account and log back in using the following credentials :
 - Email : **root@utrains.test** password : **root_pass**

user home page



8- Clean-Up

- Delete EKS Cluster using eksctl

When done, we need to delete the cluster to avoid further charges on AWS. To destroy our cluster, we must use this command

```
eksctl delete cluster --name geolocation-eks --region ap-south-1
```

- Delete ECR Repository using AWS CLI

```
aws ecr delete-repository --repository-name geolocation_ecr_rep --force
```

Conclusion:

- We have seen in this mission, the path of the source code from the developer's machine, to the deployment in the Cloud (image in ECR, and deployment of the image in EKS).
- We have automated the process using a declarative pipeline in Jenkins.
- The environment has been implemented by AWS CLI, Kubectl ...

At the end of this mission, thank you to leave in comments, your feelings, and your remarks to help us to produce for you a quality content and easy to assimilate.