Module 2 Laboratory

This is the solution for the practice assigned for SRE Academy module 2.
Please find the instructions below.

Instructions

In the following repository:
https://github.ibm.com/jacob-villegas/sre-test
Git repository: git@github.ibm.com:jacob-villegas/sre-test.git

There is the necessary code to run a simple rest-api in a docker container that returns a json file.

Exercise:
- Create your own GIT repository, within, Implement the necessary automation to deploy the container in the cloud provider of your choice (IBM Cloud, AWS, GCP, etc.).
- Use IaaC concepts
- Add a readme file that explains step by step how someone else could run the automation to do the deployment themselves without needing help.

Extra bonus: Deploy the container to a Kubernetes cluster in the cloud.

Section 1.

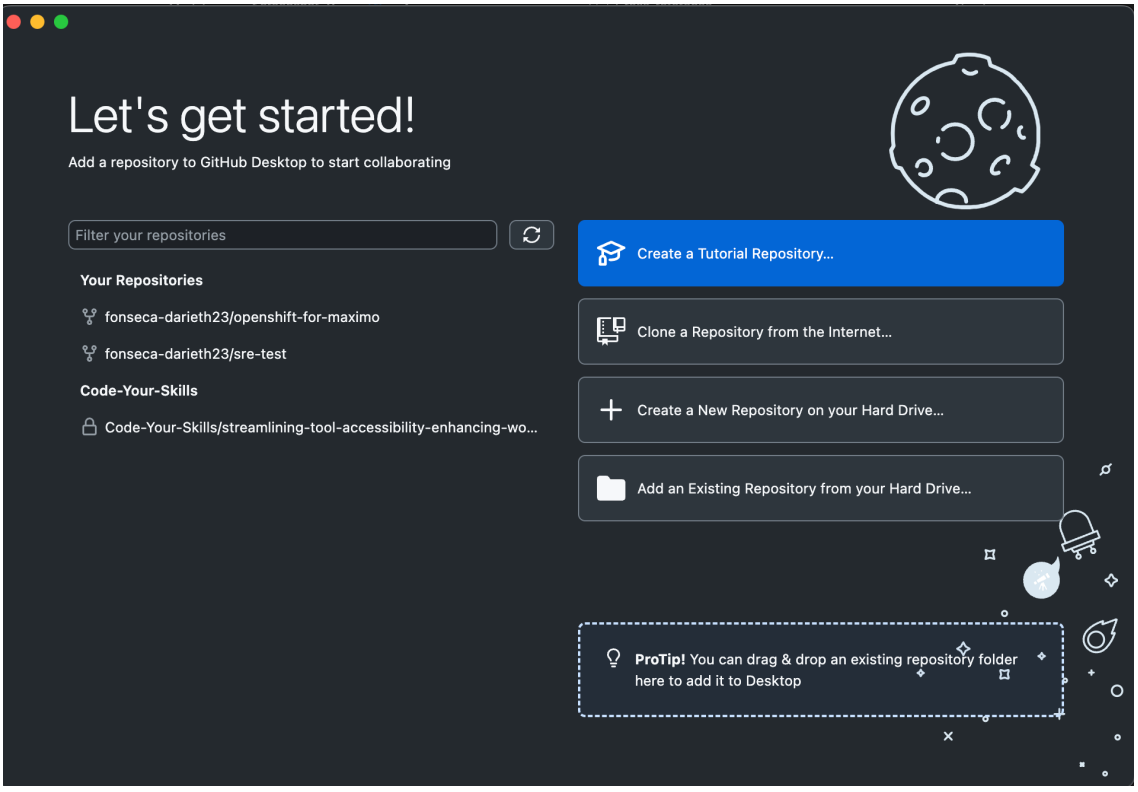
You can check the repository of this solution here: https://github.com/Darieth23/SREAcademy-SolutionLab2

In this case, the deployment of the container will be done locally, for that the solution will use the Podman application. In this case we will make the deployment of the Docker container to be accesed. **Note: move to Section1 folder.**

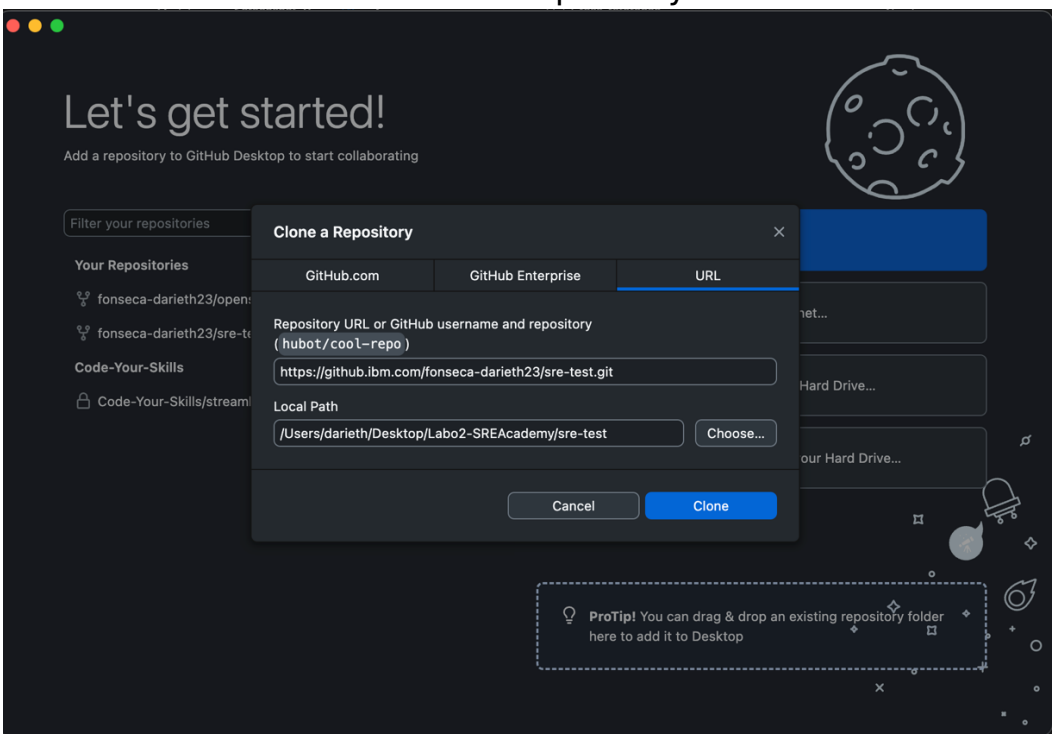
Step 1.

First is necessary to clone the repository needed, for that GitHub Desktop could be used by following the steps below. Note: Is necessary to have basic knowledge about how to use GitHub tool.

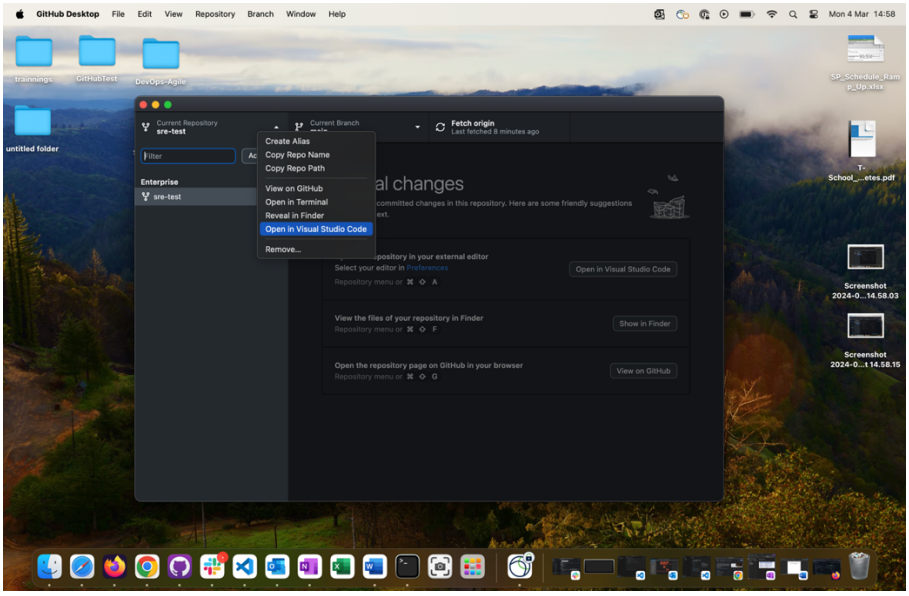
- After downloaded the application and loggin into, click on 'Clone a Repository from Internet...'



- Then click on ‘URL’, paste the link of your own GitHub repository (you can get the files from the GitHub repository shared on the instructions), choose the location of the repository and the click on ‘Clone’.



- Then you can right-click on ‘Current Repository’, and select the option ‘Open in Visual Studio Code’, you can get VSCode here.



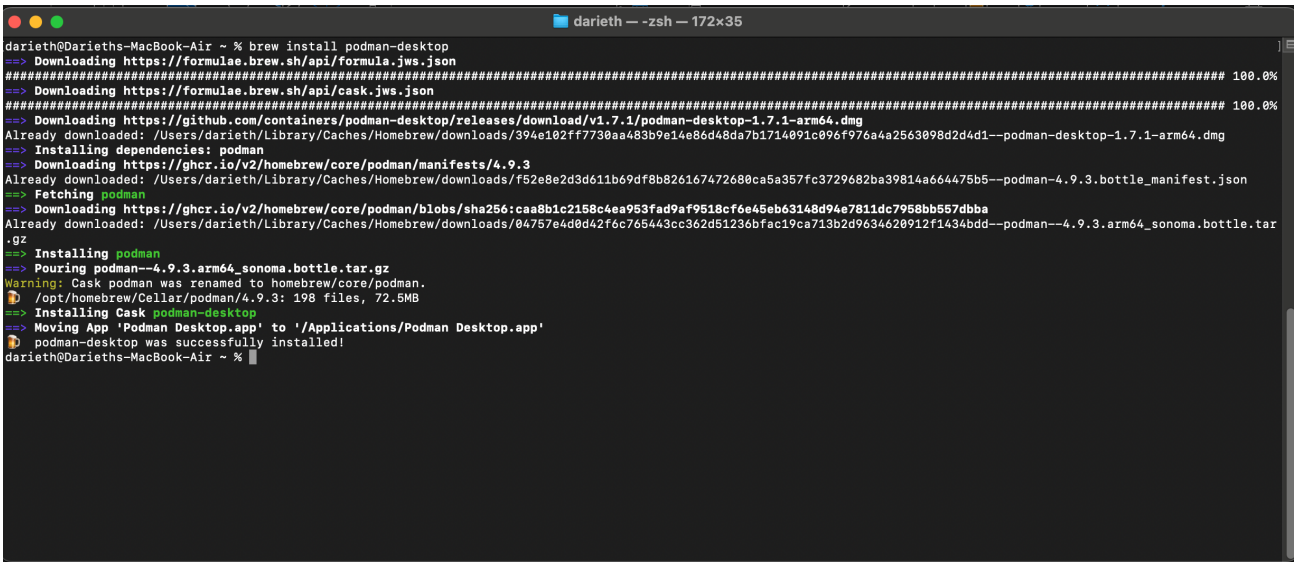
- Now you can review, and edit the files on this repository if needed by using Visual Studio Code.

Step 2.

Proceed to install the tools needed to make the deployment of the Docker container.

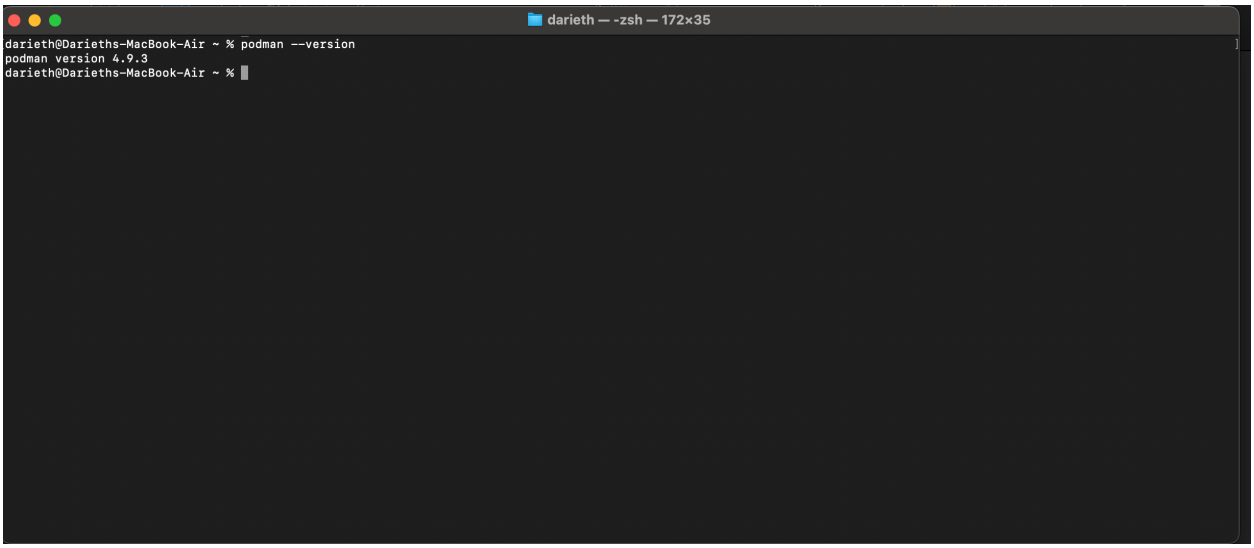
- Podman: you can use brew to install Podman Desktop. Podman is a tool that give the opportunity to use Docker commands without installing Docker directly. You need to have homebrew installed.

$brew install podman-desktop

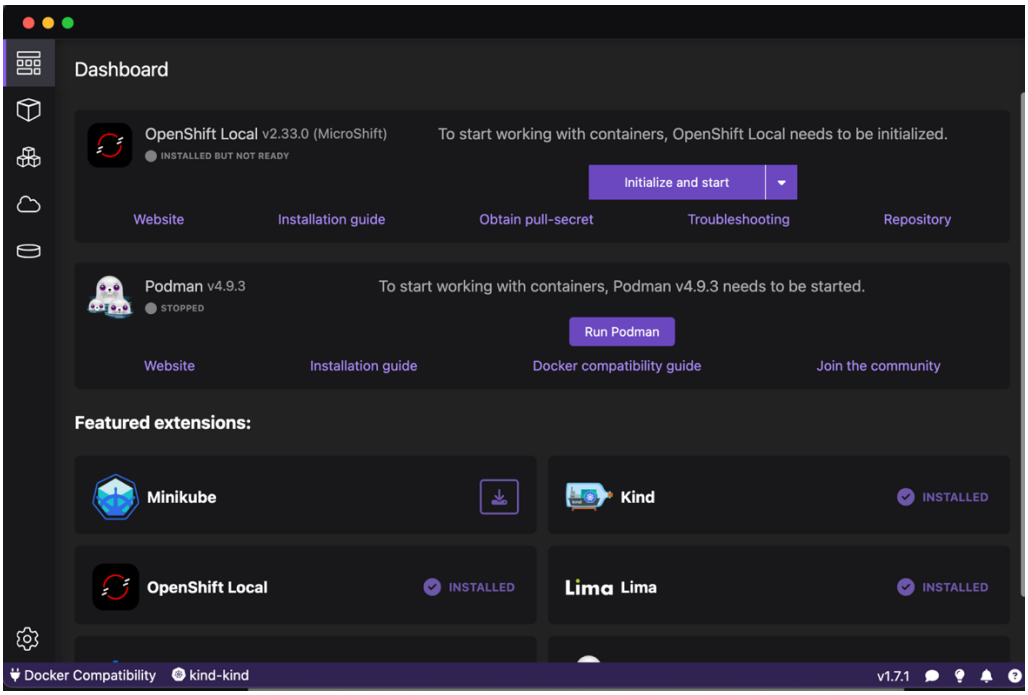


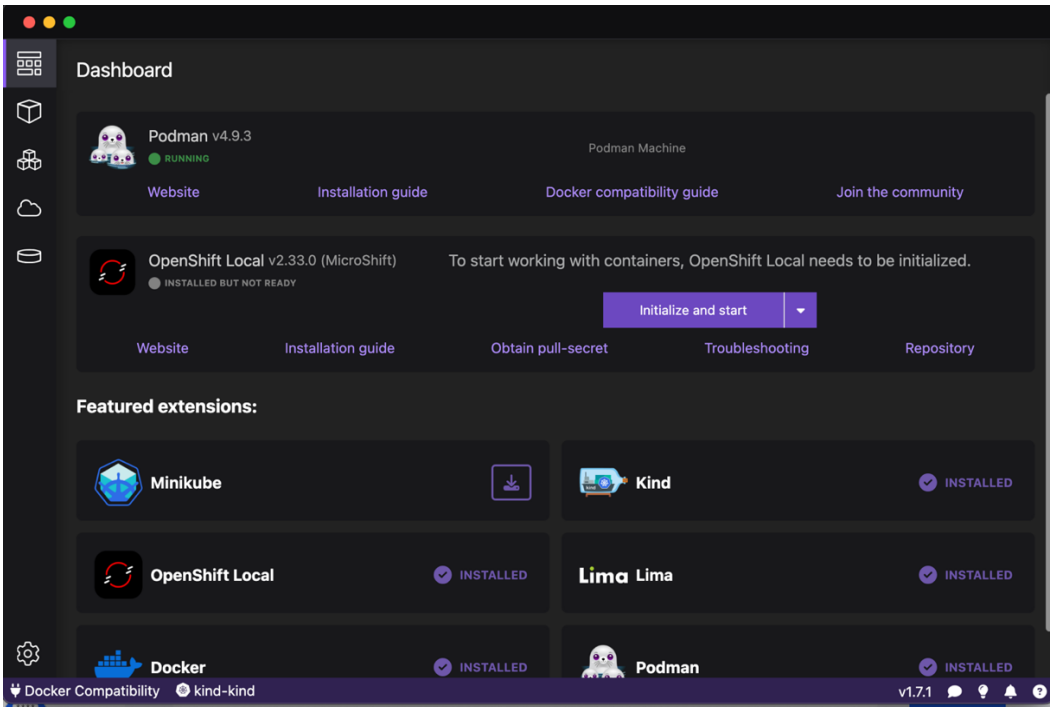
- You can check that you have Podman installed using the following command.

$podman --version



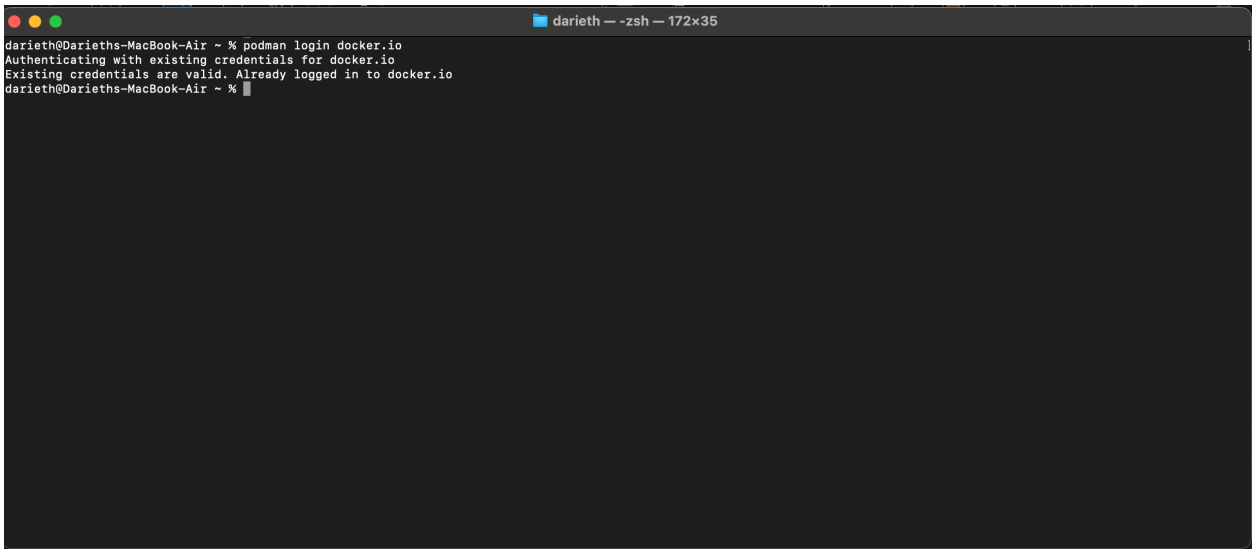
- Open Podman Desktop, and then proceed to start the machine by clicking on 'Run Podman'.





- Once you have Podman on 'Running' status, now is needed to login on Docker (an account could be created here) by typing this command, then write your user and password.

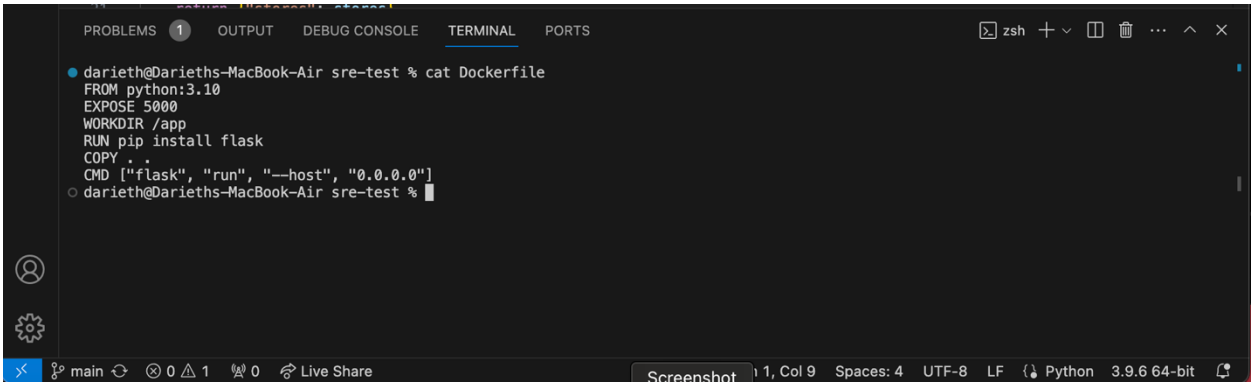
$podman login docker.io



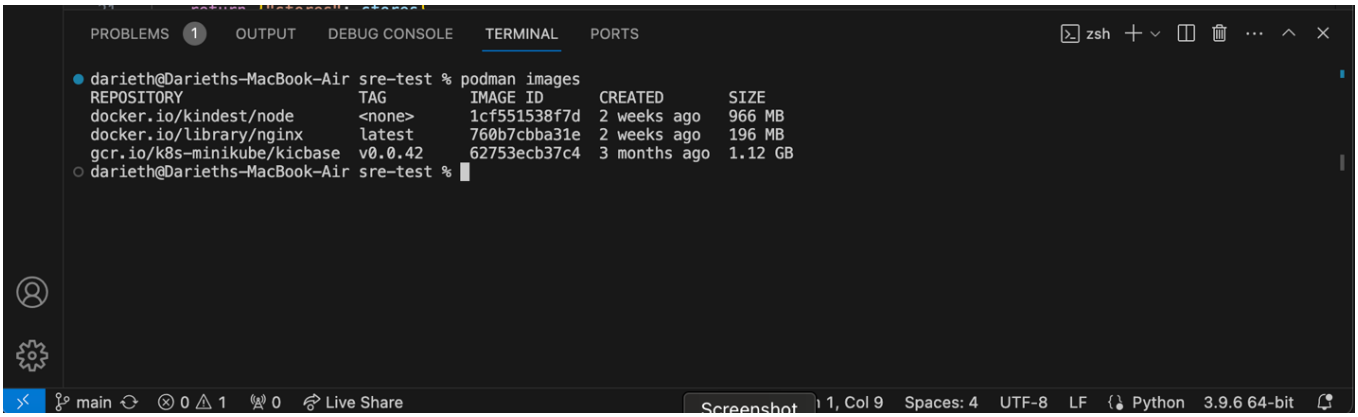
Step 3.

Now the deployment of the container could be done.

- On Visual Studio Code open a new terminal.
- You can run **$cat Dockerfile** to see the content of the Dockerfile.

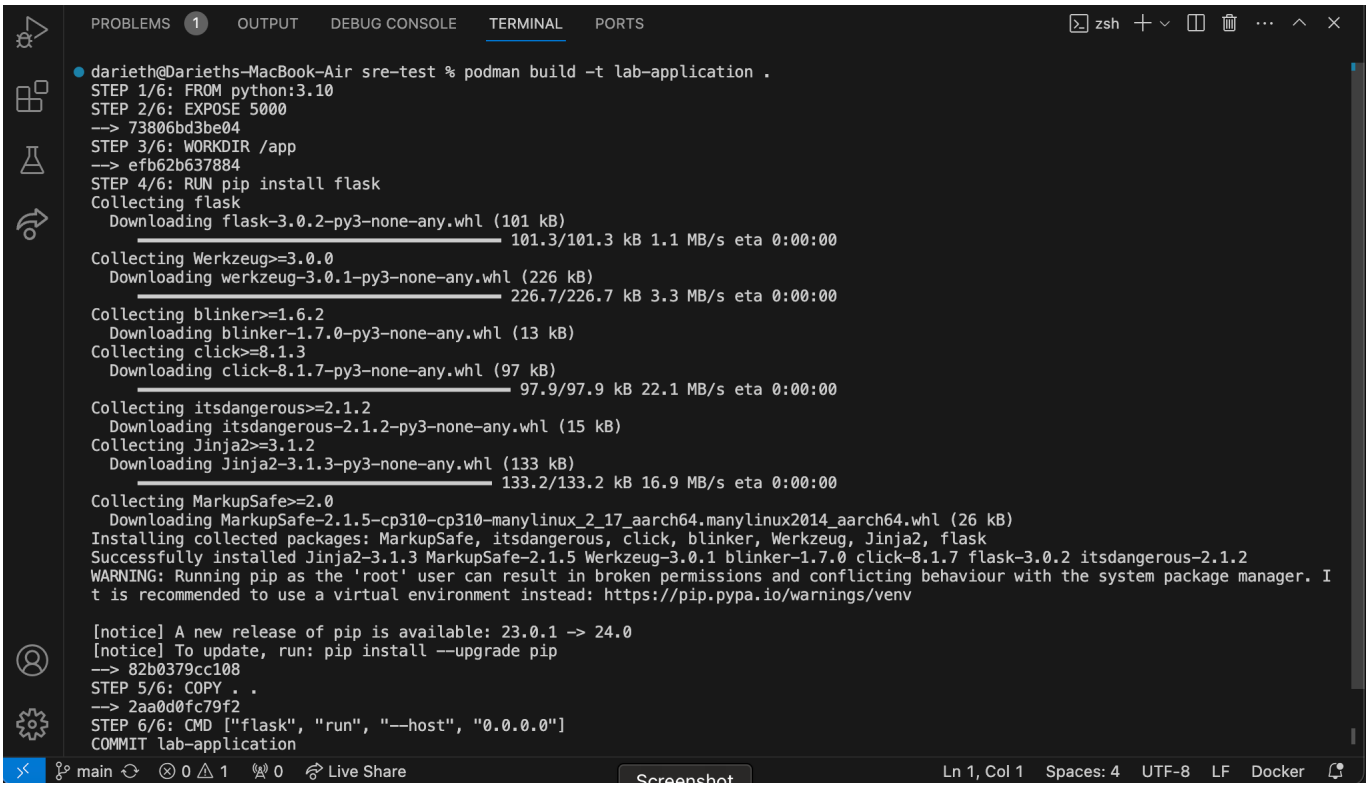


- Run **$podman images** to see the images that are available.

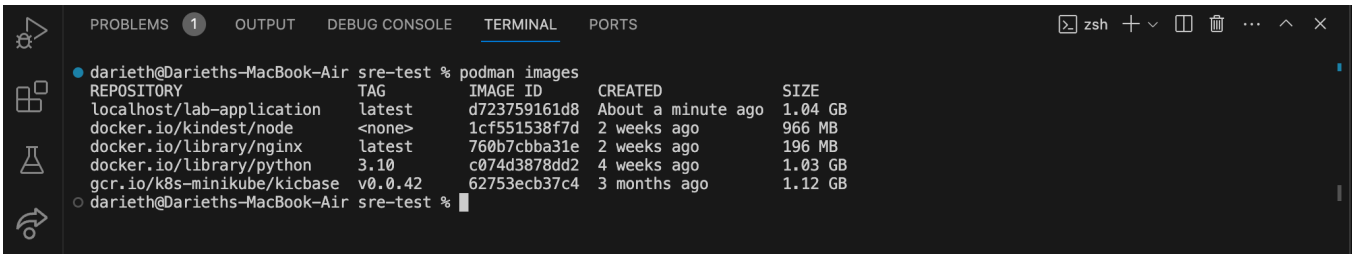


- Now is necessary to build the image by using the Dockerfile, for that run the following comman.

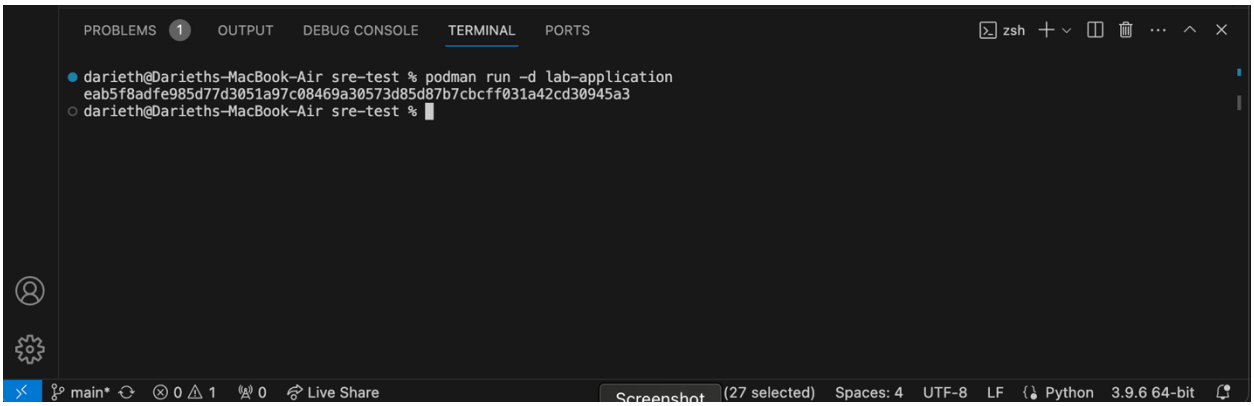
$podman build -t lab-application



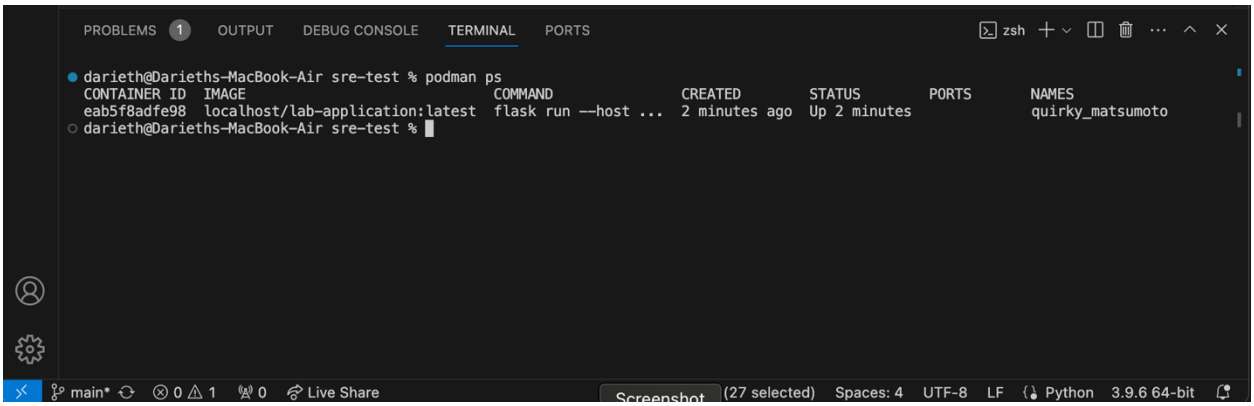
- Run again **$podman images** to see the images that are available, lab-application image is now available.



- You can create a container using the image built ‘lab-application’ by typing the command **$podman run -d lab-application**.

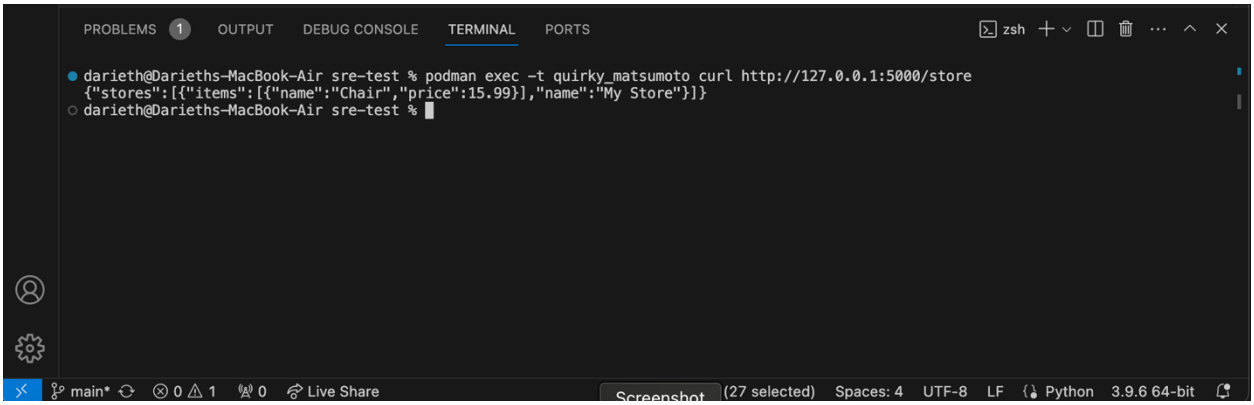


- To see the new container, run **$podman ps**.



- Then you can try to get the json reponse by running the following command, remember to get the name of the containar with the command runned before.

$podman exec -t quirky_matsumoto curl http://127.0.0.1:5000/store

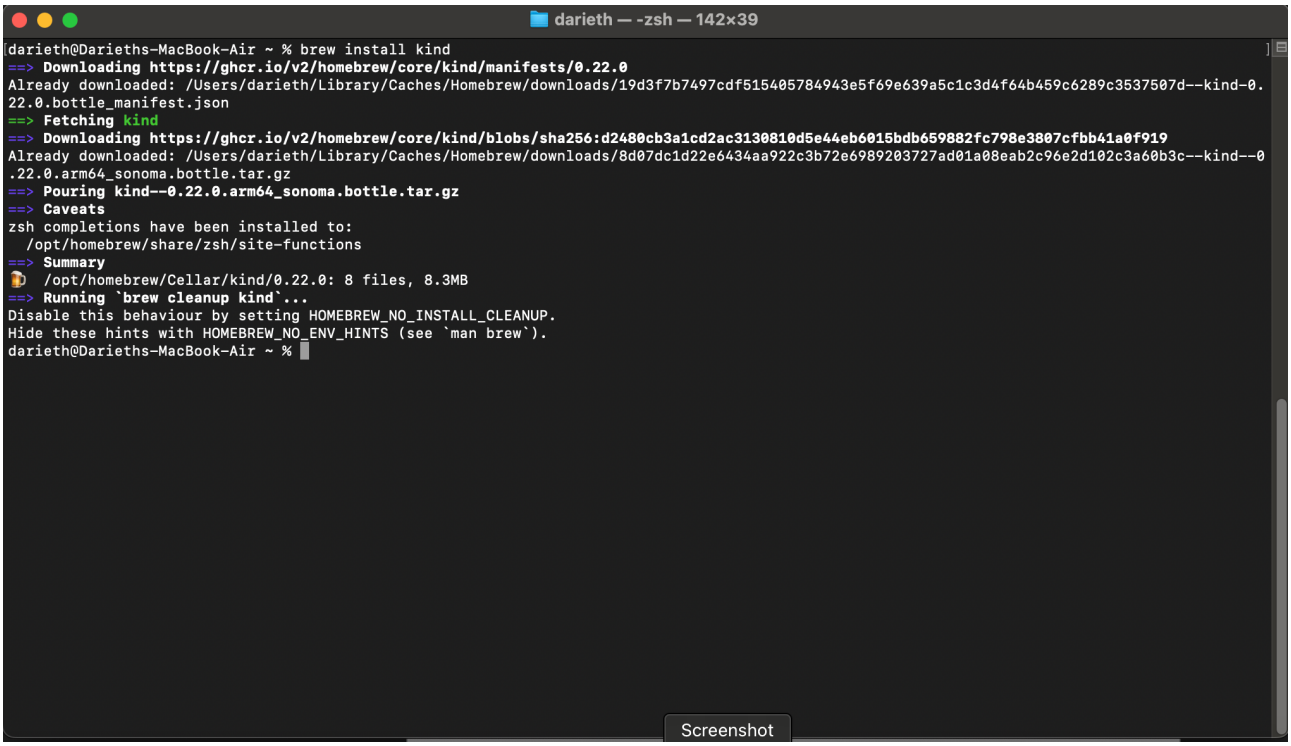


Section 2.

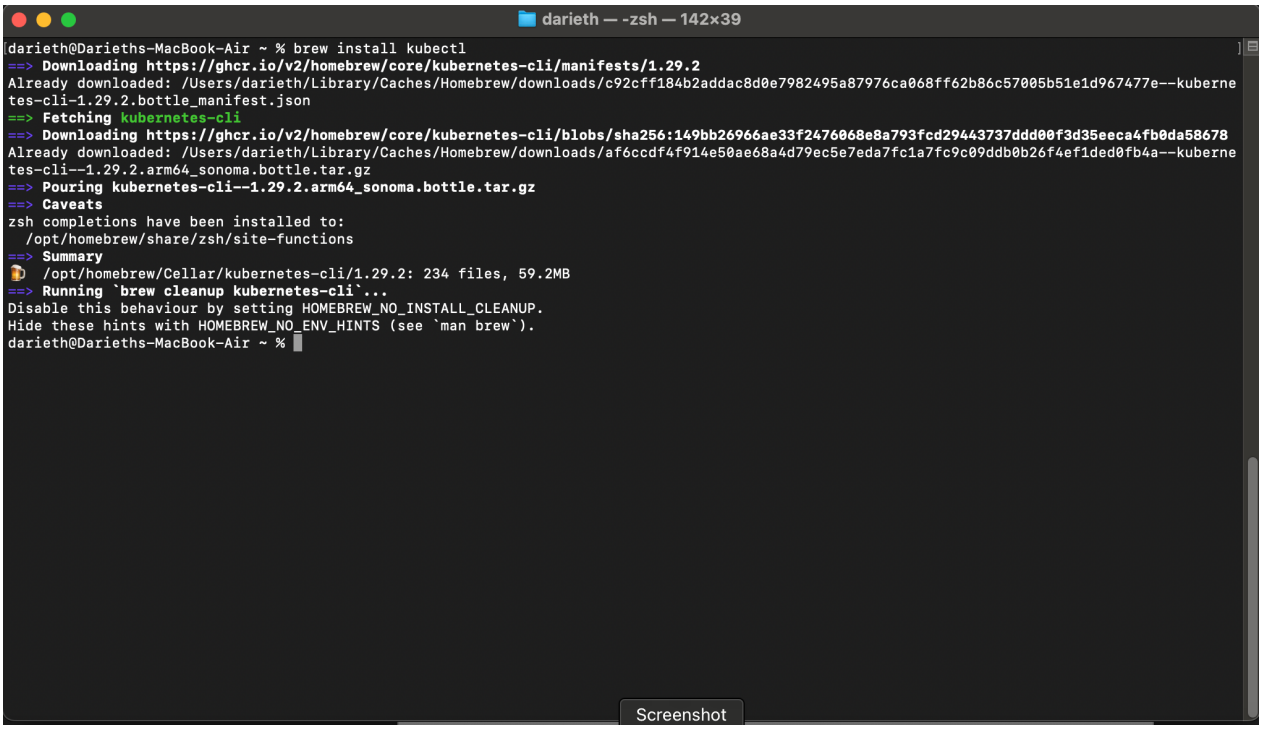
This section shows how to make the deployment of the container from Section 1 to a Kubernetes cluster. **Note: move to Section2 folder.**

Step 1.

- First is necessary to install kind tool, this tool is used to create a kubernetes cluster. Run the command **$brew install kind**.



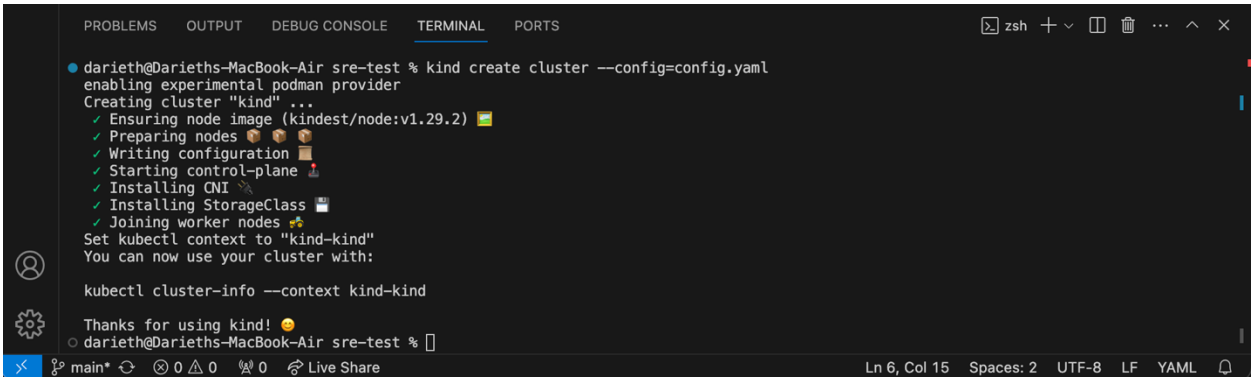
- Then proceed to install kubectl by using **$brew install kubectl.**



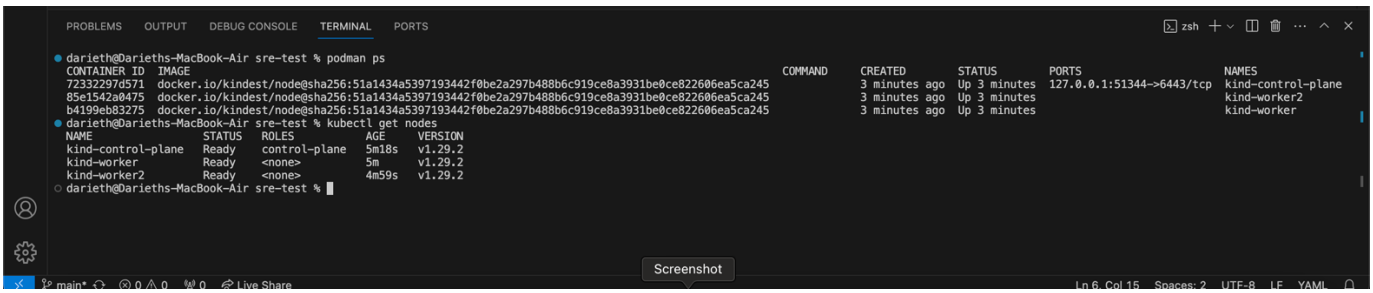
Step 2.

- Proceed to create a cluster using kind, in this case the config.yaml file will be used to deploy one master, and two worker nodes.

$kind create cluster --config=config.yaml

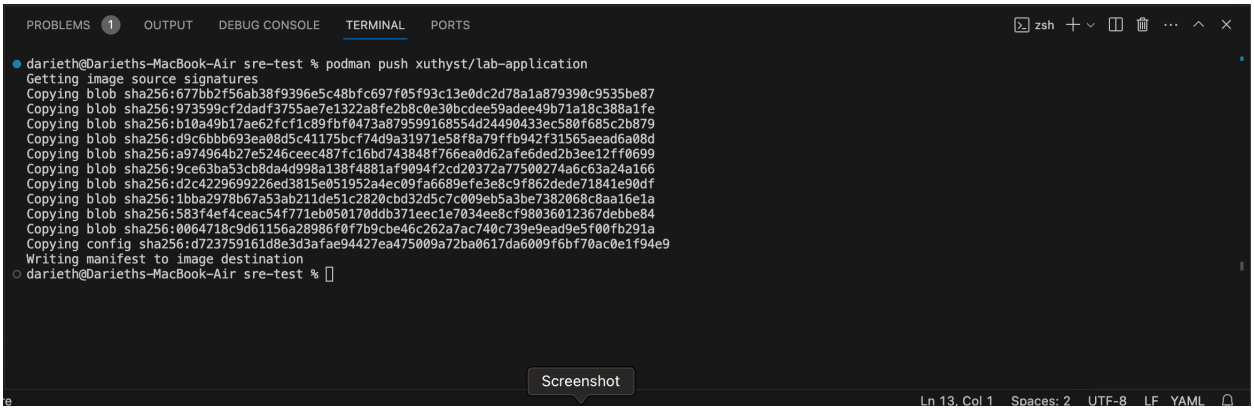


- Run **$podman ps** and **$kubectl get nodes** to see the information of the nodes running on the kind cluster.

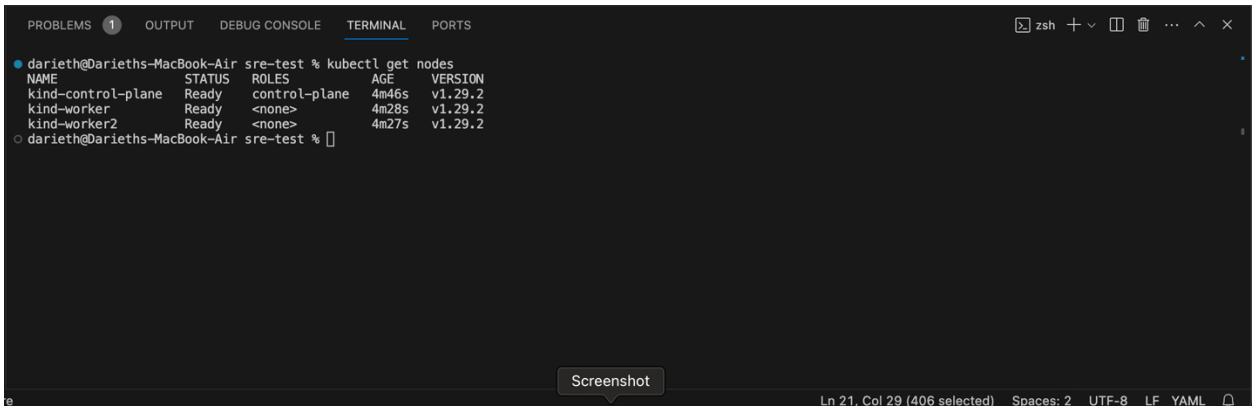


- Is necessary to push the image to Docker Hub, for that, use the command below (change 'xuthyst' for your specific username on Docker Hub).

$podman push xuthyst/lab-application

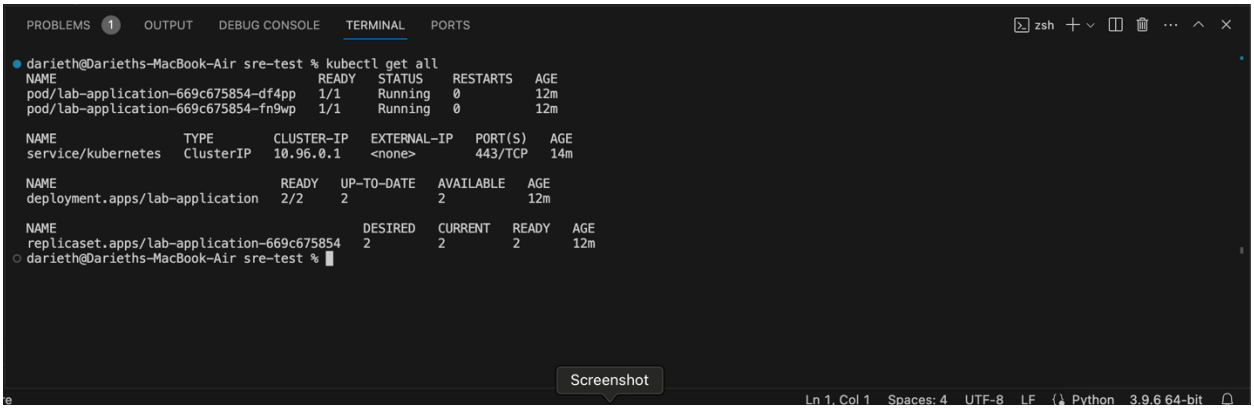


- You can make the deployment of the containers on the nodes created, by using the command **$kubectl apply -f deployment.yaml**, then two worker nodes created could be checked by running **$kubectl get nodes**

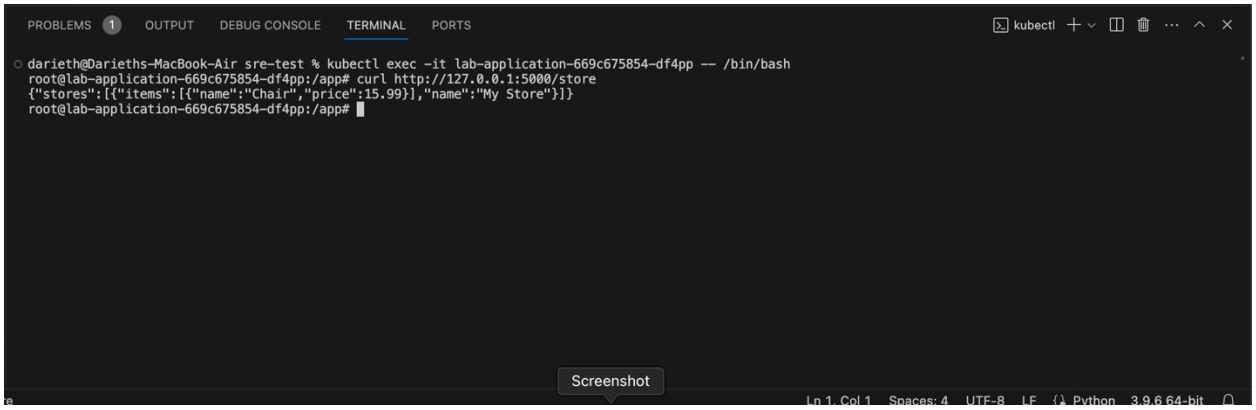


- By running **$kubectl get pods** there will be two pods created

- By running **$kubectl get all** you can see what has been created in the process



- Now you can run the command **$kubectl exec -it lab-application-669c675854-df4pp -- /bin/bash**, and then make the curl by running **$curl http://127.0.0.1:5000/store**

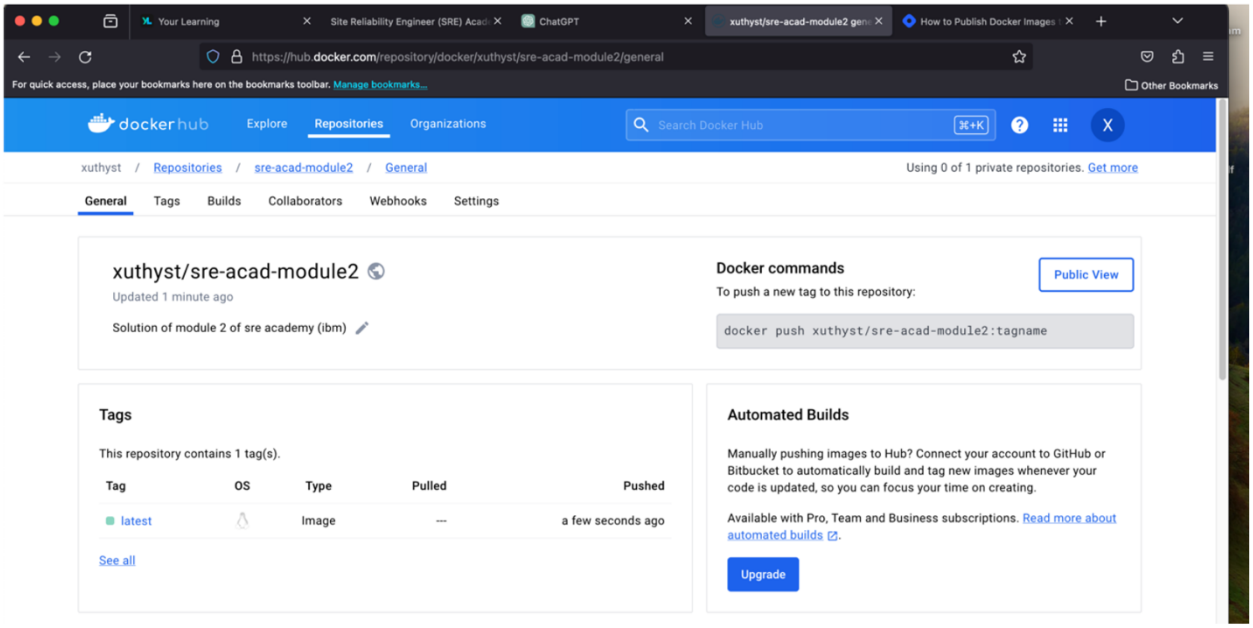


Section 3.

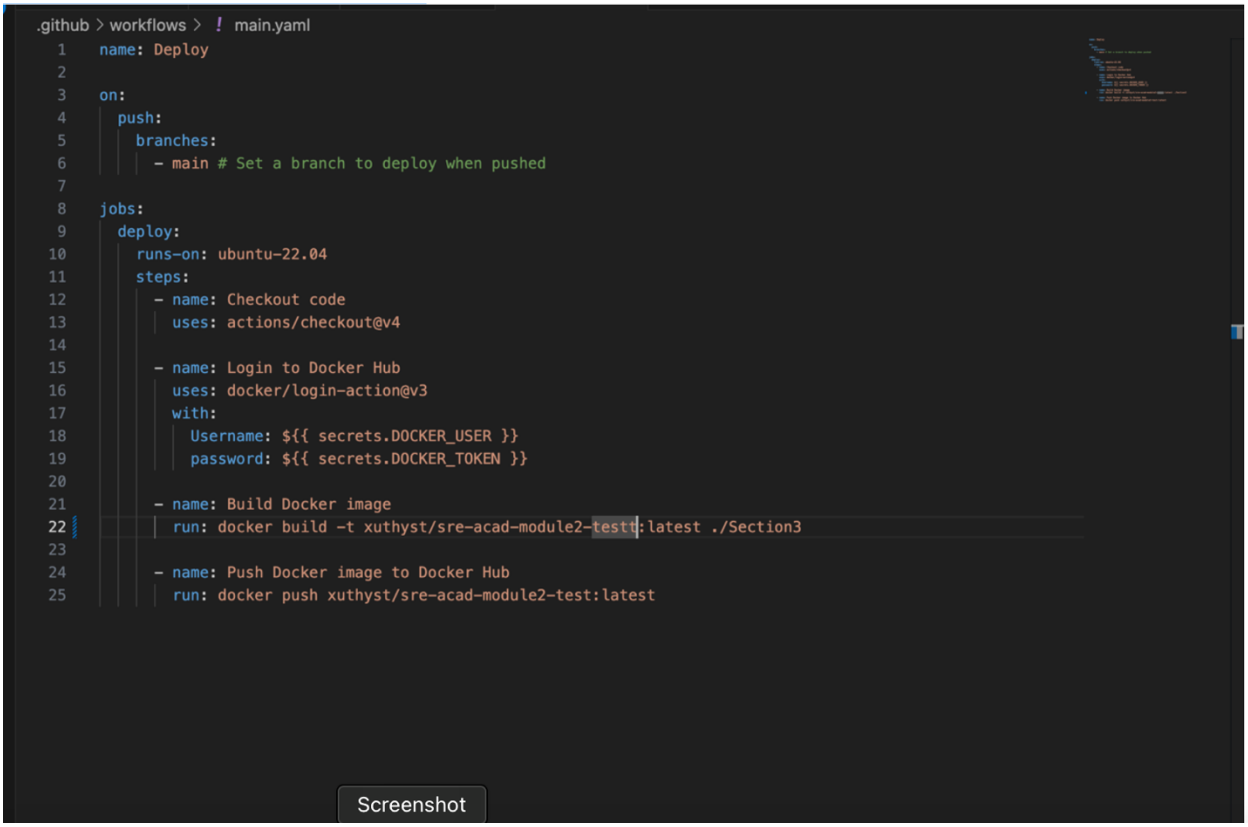
For this section we'll continue using the same repository used for Section1 and Section2. **Note: move to Section3 folder.**

Step 1.

- First of all, we'll try to push a new docker image to Docker Hub by using GitHub Actions workflow. Clone your repository on your local machine by using GitHub Desktop, and open it on Visual Studio Code, then proceed to create a new folder called '.github/workflows'.
- Let's create two secrets on the GitHub repository, one is for DOCKER_USER and the other one is for the DOCKER_TOKEN (two create a token refer to docker token), with this two secrets you will be available to use them into your 'main.yaml' file to push the Docker Image on you Docker Hub.
- Create a new .yaml file inside this new folder called 'main.yaml', you can check the content on the image attached. This file will push the docker image into Docker Hub everytime we make a new push on GitHub Desktop. Proceed to push the changes using GitHub Desktop. You can check the update on your Docker Hub repository.



- This is the code used inside the main.yaml file, remember to change the username, and the repository name in the last two steps with your own user, and repository name.



Step 2.

Let's try to create out Kubernetes cluster on AWS EKS, in this case, we'll use a tool that automate this process called EKSCTL, and aws cli to run the commands needed.

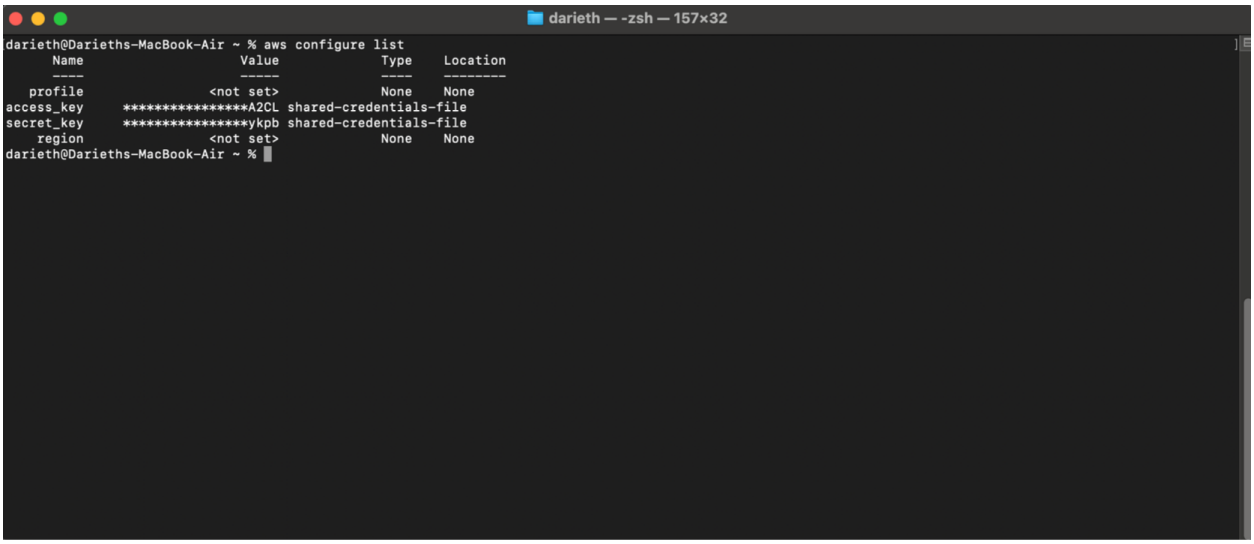
- Install EKSCTL by running: **$brew tap weaveworks/tap**

$brew install weaveworks/tap/eksctl $eksctl version (to check correct instalation)

- Install AWS CLI by following the steps on this link: Install AWS CLI
- Now is necessary to create a IAM user on out AWS account, for this process you can check the content of this link: Create IAM User, with

this user we'll be available to run commands on out local terminal with AWS CLI.

- Once you have configured your IAM user credentials on AWS CLI, you can run **$aws configure list** to check that.

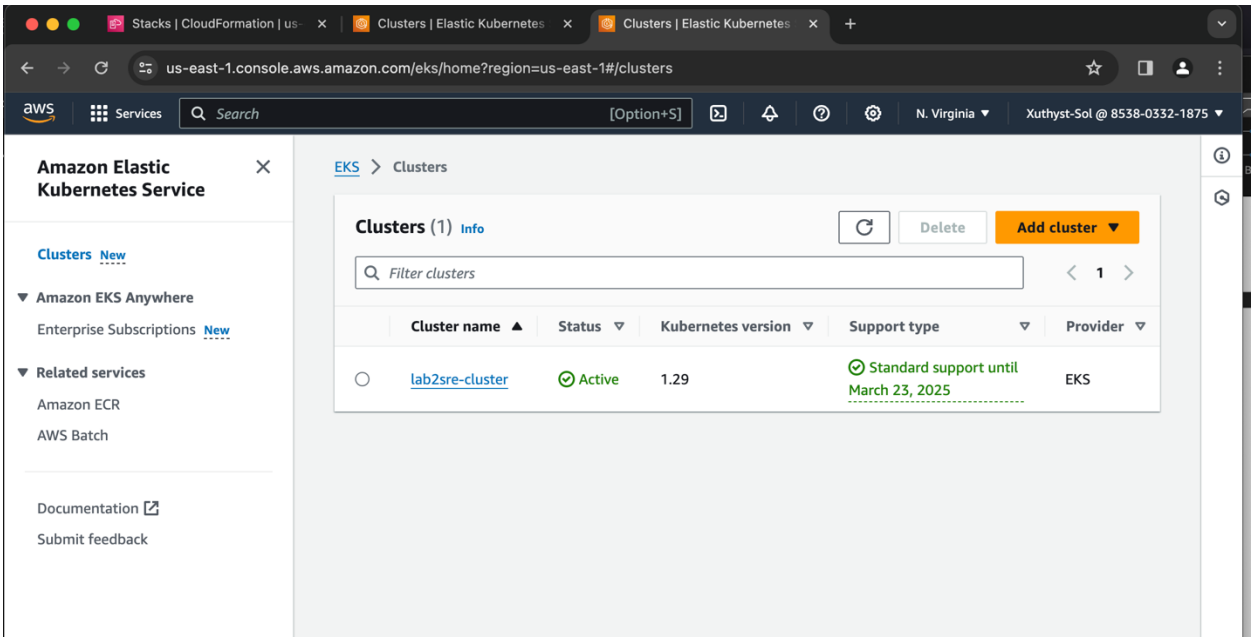


- Now we are ready to create our EKS Cluster, run the following command.

**$eksctl create cluster **
**> --name lab2-cluster **
**> --version 1.29 **
**> --region us-east-1 **
**> --nodegroup-name linux-nodes \ > --node-type t2.micro **

> --nodes 2

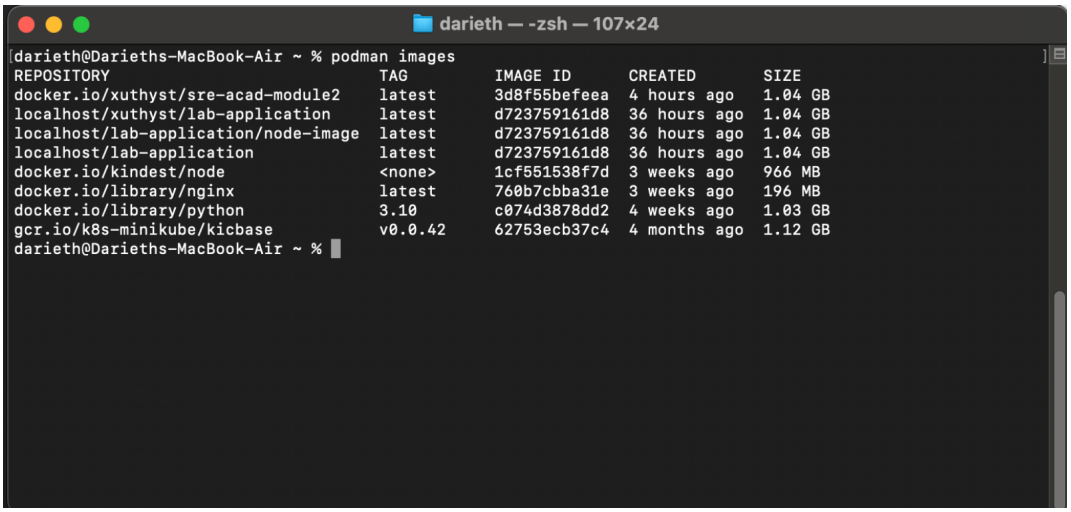
- Now your cluster has been created, now you can run commands using kubectl similarly as we done on Section 2.



Step 3.

Now we will try to deploy, and set the application to be accessed with the web browser, for this step we'll use a deployment.yaml file also a service.yaml file.

- Create deployment.yaml, and service.yaml files as follows.
- Is necessary to pull the image created on step 1 by running **$docker pull <docker_user>/<image_name>:<tag>** You can run **$podman images** to see that the image is now available.



- Once you have created the .yaml files needed, you can run.

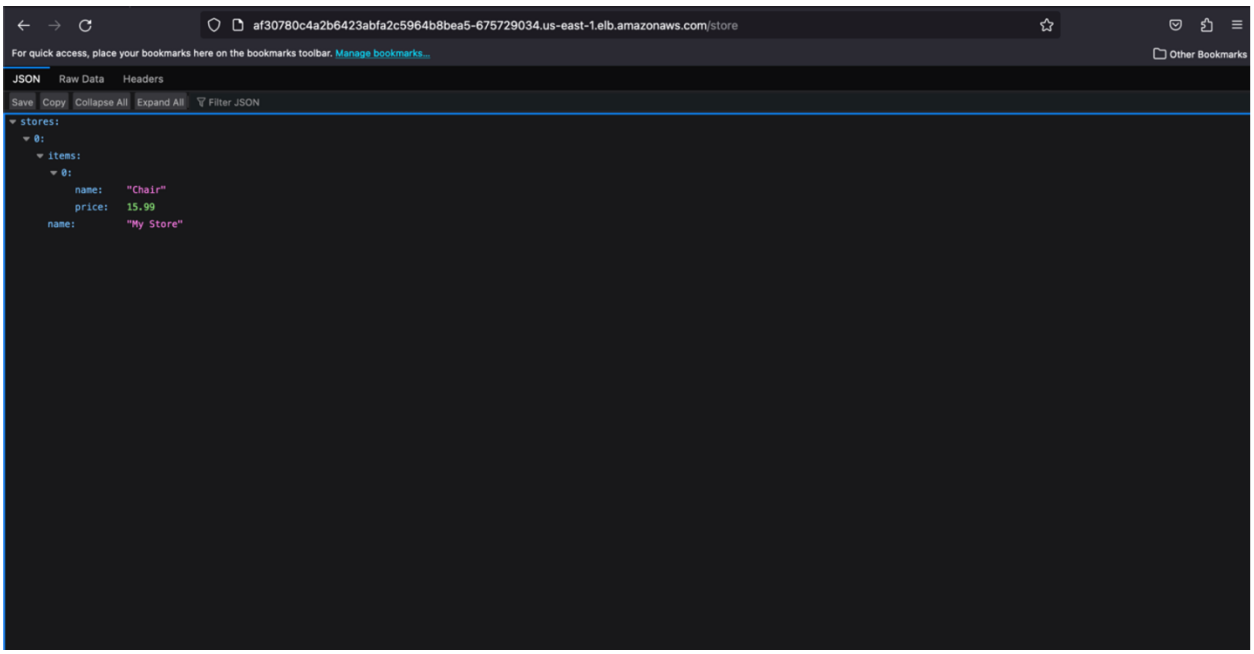
$kubectl apply -f deployment.yaml

$kubectl apply -f service.yaml

- To access the application, run the command **$kubectl get services**, and then use the external IP to access it by using your browser.

- Try to access the application from your browser with the following link (remember to use '/store' in the URL provided

http://af30780c4a2b6423abfa2c5964b8bea5-675729034.us-east-1.elb.amazonaws.com/store



- To delete your cluster, run
  $eksctl delete cluster --name lab2sre-cluster

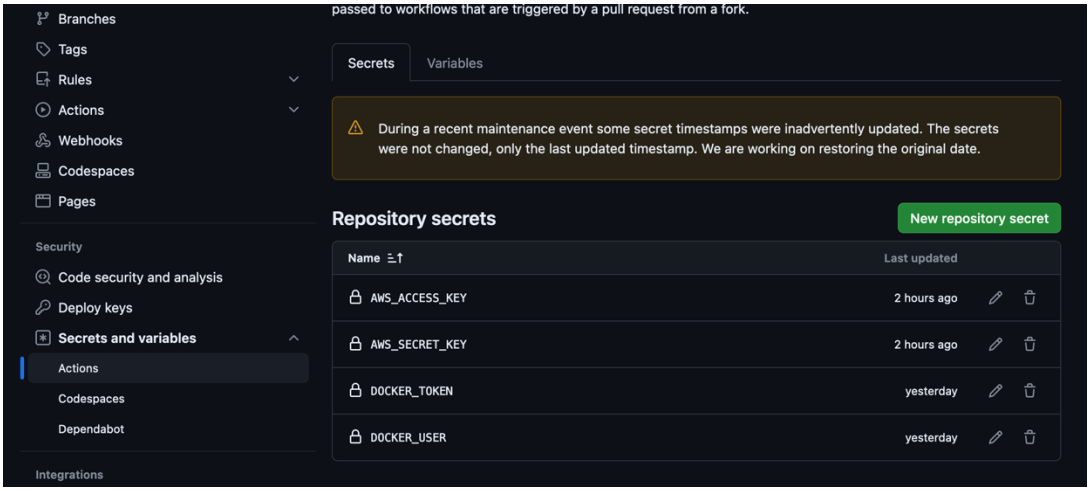
Section 4.

In this case, this solution will use GitHub Actions to make the deployment of the application into a Kubernetes cluster. **Note: move to Section4 folder.**

The main.yaml file located in .github/workflows folder run different commands to make possible the deployment process.

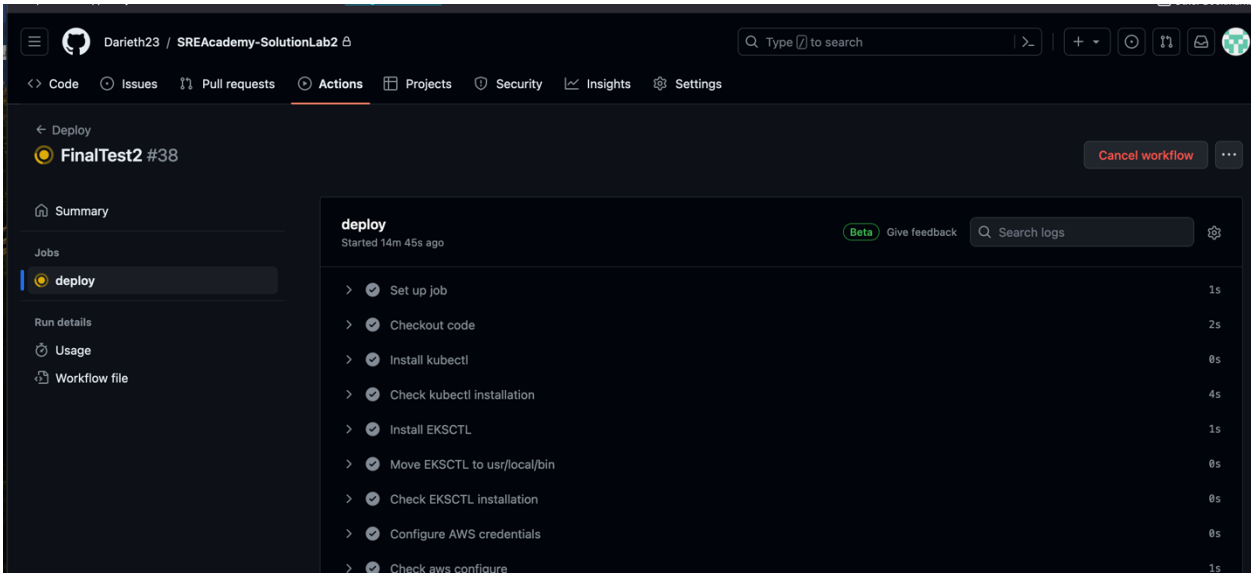
How to run the pipeline.

- Clone the repository using GitHub Desktop (only if you need to make changes .
- Create the secrets needed in GitHub web if you need to add your own credentials, and then proceed to change the secrets references in the main.yaml file.

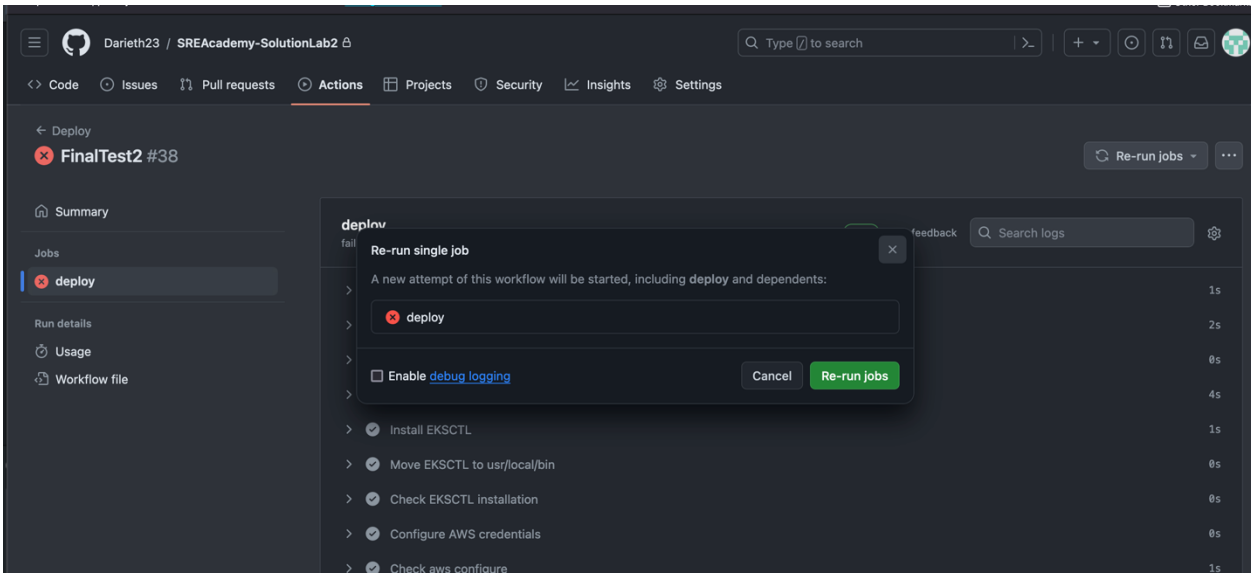


```yaml
- name: Check EKSCTL installation
  run: eksctl version

#Set AWS Configure
- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v2
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_KEY }}
    aws-region: us-east-1

- name: Check aws configure
  run: aws configure list

#Build and push docker image
- name: Login to Docker Hub
  uses: docker/login-action@v3
  with:
    Username: ${{ secrets.DOCKER_USER }}
    password: ${{ secrets.DOCKER_TOKEN }}
```

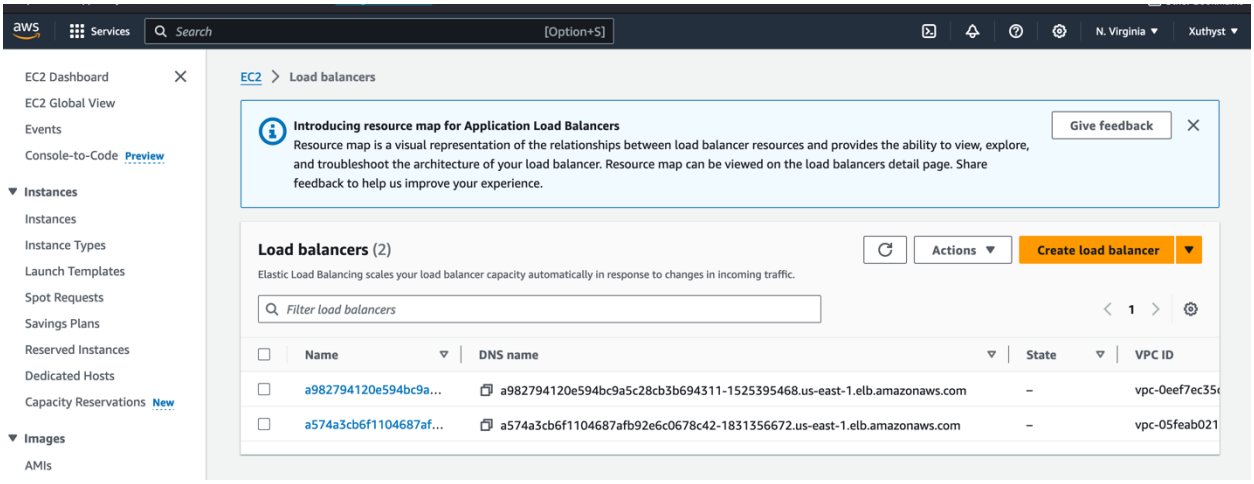
- You can make the changes needed, then push the changes to the repository, this will run automatically the deployment, you can check the process in GitHub web, by going to Actions.

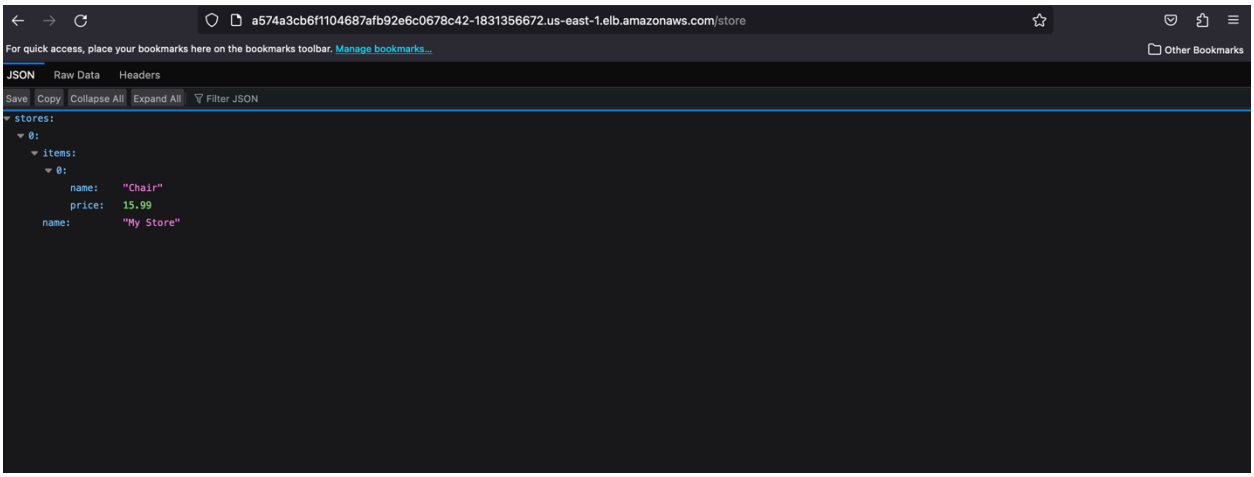


- If you want to run again the job without any change, just select the FinalTest3 or last action executed with succeed status in Actions Sections, then go to Jobs there you'll have the opportunity to run again the 'deploy' job, you can click on Re-run jobs.



- After running the cluster, you can get the Service URL you can go to AWS Web, then look for EC2 services, click on LoadBalances and get the URL of any Load Balancer as DNS name, and try to access the service as follos **http://URL-PROVIDED/store.**





- **Note: it's important to know that everytime you make a deployment, is necessary to create a cluster with a different name or delete the one created before.**

main.yaml file explanation.

The main.yaml file contains the commands needed to create the deployment of the application in a cluster on AWS.
Inside this file there is a job called 'deploy' that runs on Ubuntu 20.04 Virtual Machine.
First the job tries to install the tools needed in the Linux machine that is running on GitHub Actions, proceed to set the AWS configure, then build and push the Docker Image to Docker Hub, make the cluster deployment with eksctl, and finally create the service in the new cluster.