

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:

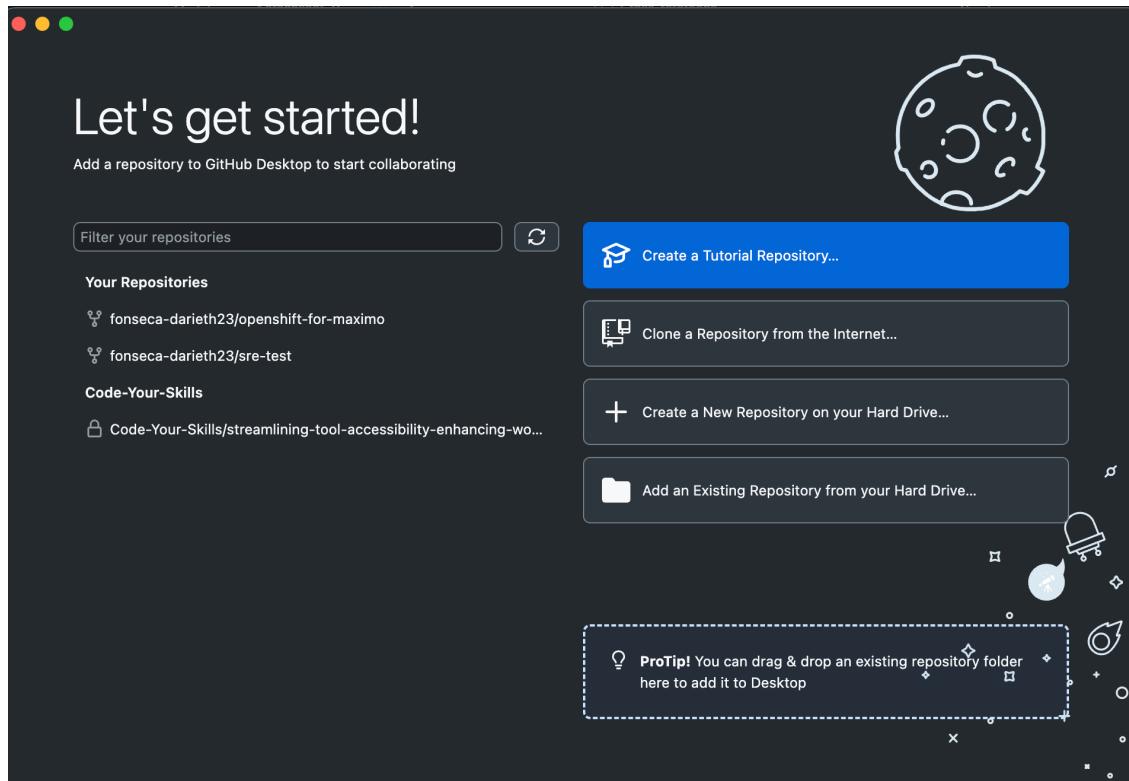
<https://github.com/Darieth23/SREAcademy-SolutionLab2>

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 accessed. **Note: move to Section1 folder.**

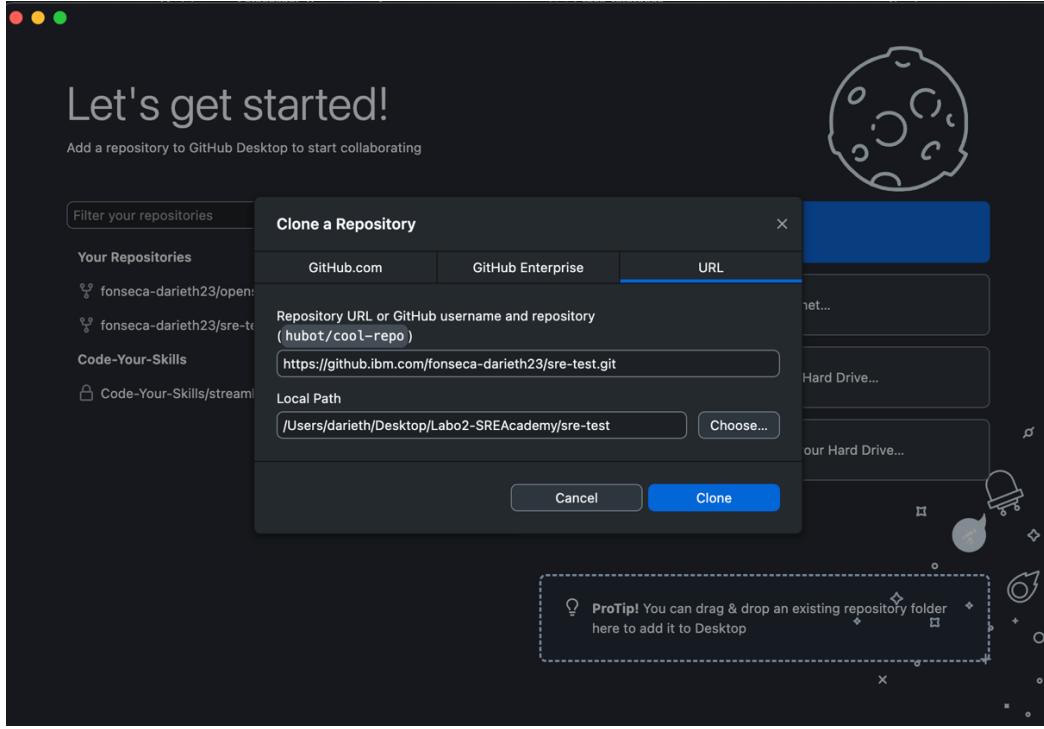
Step 1.

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.

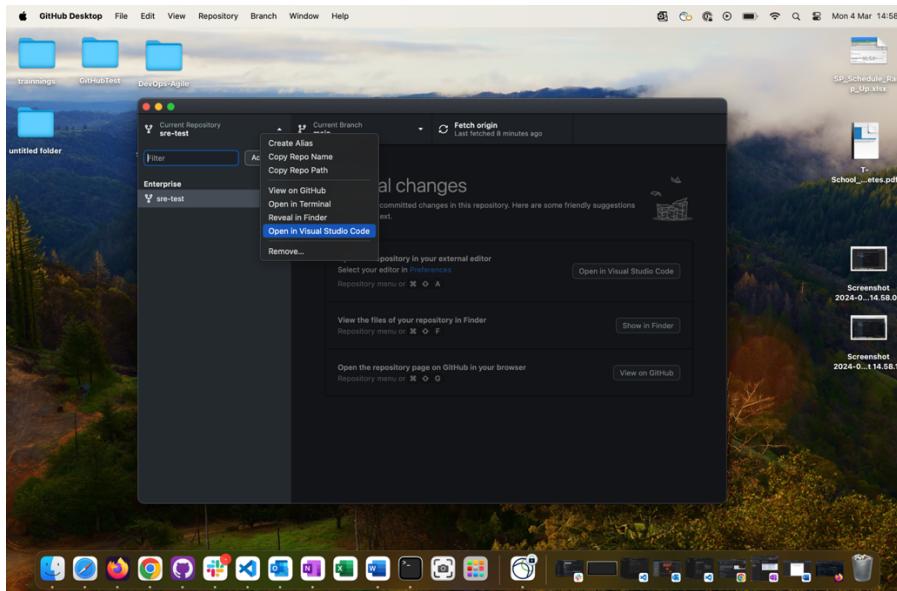
- After downloaded the application and loggin into, click on ‘Clone a Repository from Internet...’



- 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](#).



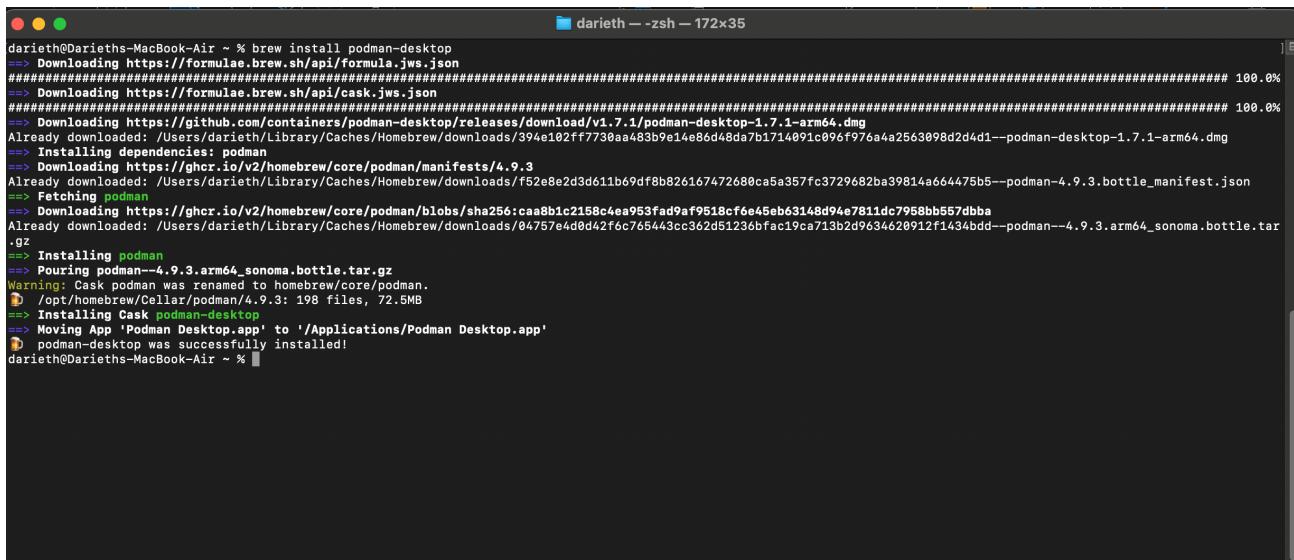
- Now you can review, and edit the files on this repository if needed by using Visual Studio Code.

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.

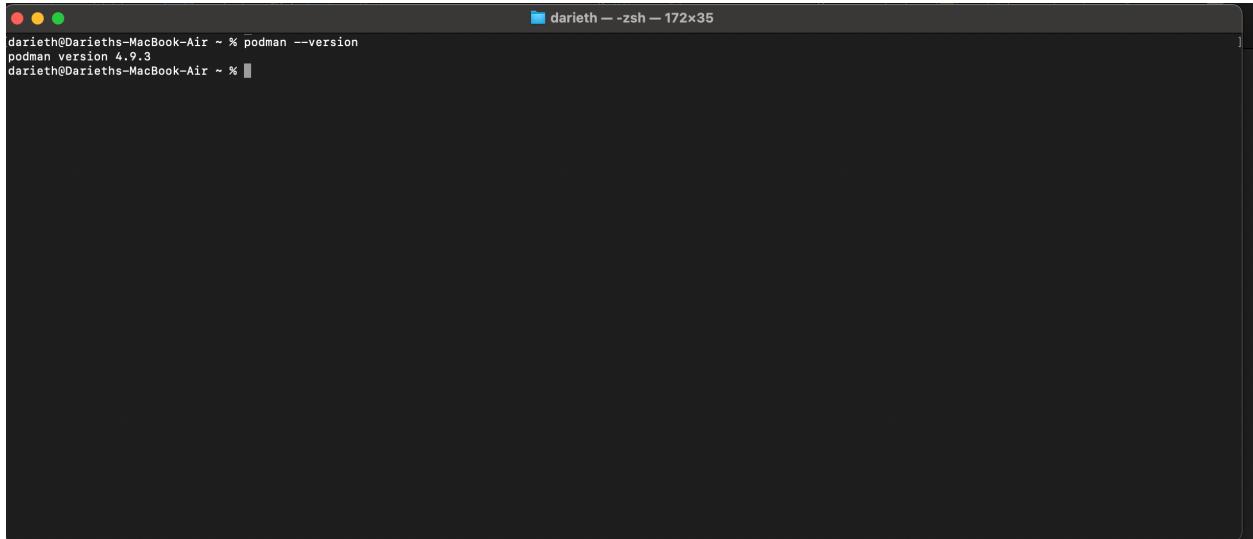
\$brew install podman-desktop



```
darieth@Darieths-MacBook-Air ~ % brew install podman-desktop
==> Downloading https://formulae.brew.sh/api/formula.jws.json
#####
# 100.0%
==> Downloading https://formulae.brew.sh/api/cask.jws.json
#####
# 100.0%
==> Downloading https://github.com/containers/podman-desktop/releases/download/v1.7.1/podman-desktop-1.7.1-arm64.dmg
Already downloaded: /Users/darieth/Library/Caches/Homebrew/downloads/394e102ff7730aa483b9e14e86d48da7b1714091c096f976a4a2563098d2d4d1--podman-desktop-1.7.1-arm64.dmg
==> Installing dependencies: podman
==> Downloading https://ghcr.io/v2/homebrew/core/podman/manifests/4.9.3
Already downloaded: /Users/darieth/Library/Caches/Homebrew/downloads/f52e8e2d3d61b69df8b826167472680ca5a357fc3729682ba39814a664475b5--podman-4.9.3.bottle_manifest.json
==> Fetching podman
==> Downloading https://ghcr.io/v2/homebrew/core/podman/blobs/sha256:caa8b1c2158c4ea953fad9af9518cf6e45eb63148d9ae7811dc7958bb557dbba
Already downloaded: /Users/darieth/Library/Caches/Homebrew/downloads/04757e4d0d42f6c765443cc362d51236bfac19ca713b2d9634620912f1434bdd--podman--4.9.3.arm64_sonoma.bottle.tar.gz
==> Installing podman
==> Pouring podman--4.9.3.arm64_sonoma.bottle.tar.gz
Warning: Cask podman was renamed to homebrew/core/podman.
  /opt/homebrew/Cellar/podman/4.9.3: 198 files, 72.5MB
==> Installing Cask podman-desktop
==> Moving App 'Podman Desktop.app' to '/Applications/Podman Desktop.app'
  podman-desktop was successfully installed!
darieth@Darieths-MacBook-Air ~ %
```

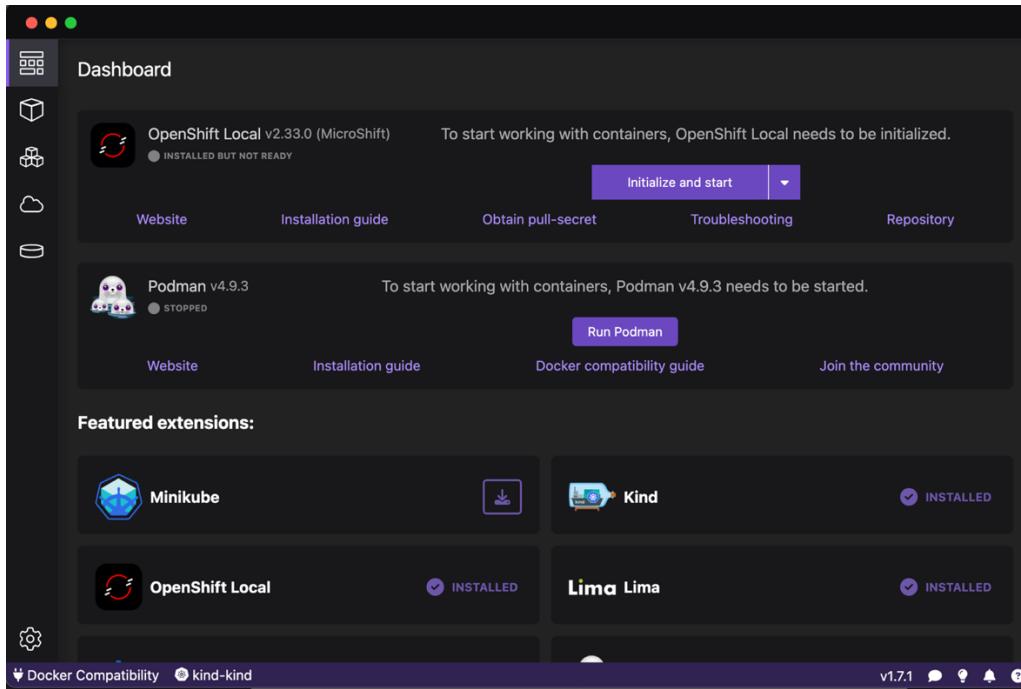
- You can check that you have Podman installed using the following command.

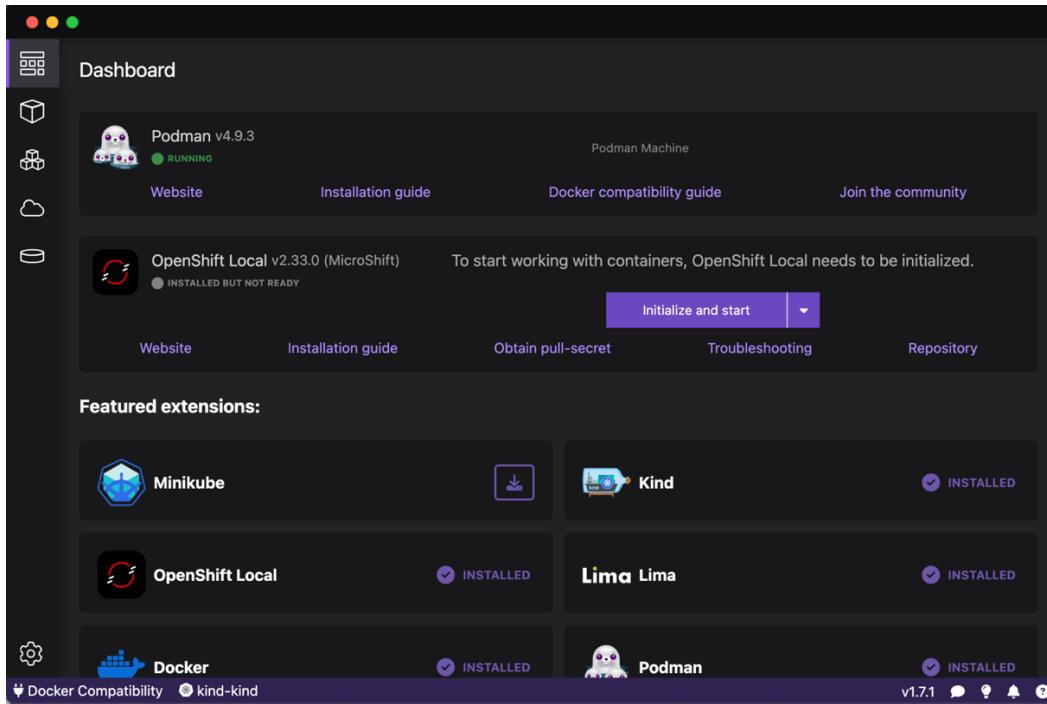
```
$podman --version
```



```
darieth@Darieths-MacBook-Air ~ % podman --version
podman version 4.9.3
darieth@Darieths-MacBook-Air ~ %
```

- 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

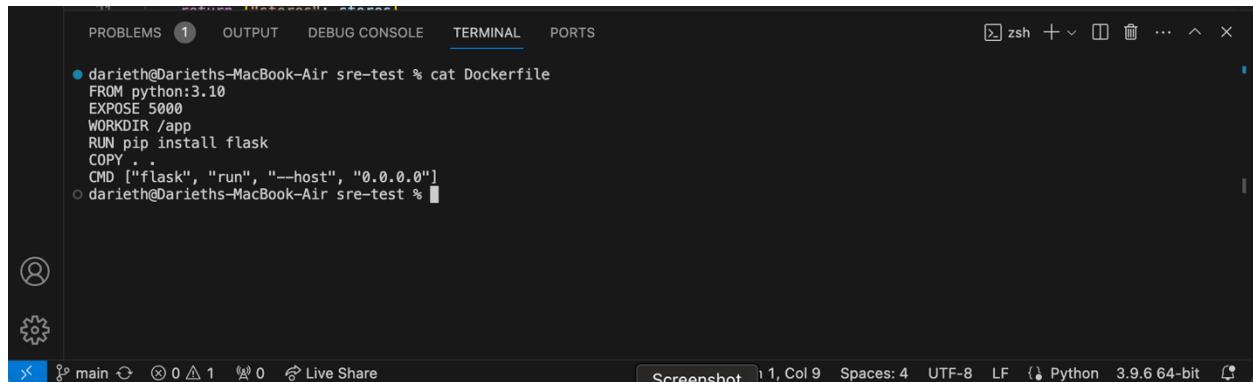
```
darieth@Darieths-MacBook-Air ~ % podman login docker.io
Authenticating with existing credentials for docker.io
Existing credentials are valid. Already logged in to docker.io
darieth@Darieths-MacBook-Air ~ %
```

The screenshot shows a terminal window with a dark background. The command \$podman login docker.io is entered at the prompt. The output shows that it is authenticating with existing credentials for docker.io and that the existing credentials are valid, indicating that the user is already logged in to docker.io. The terminal window has a title bar showing 'darieth -- zsh -- 172x35'.

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.

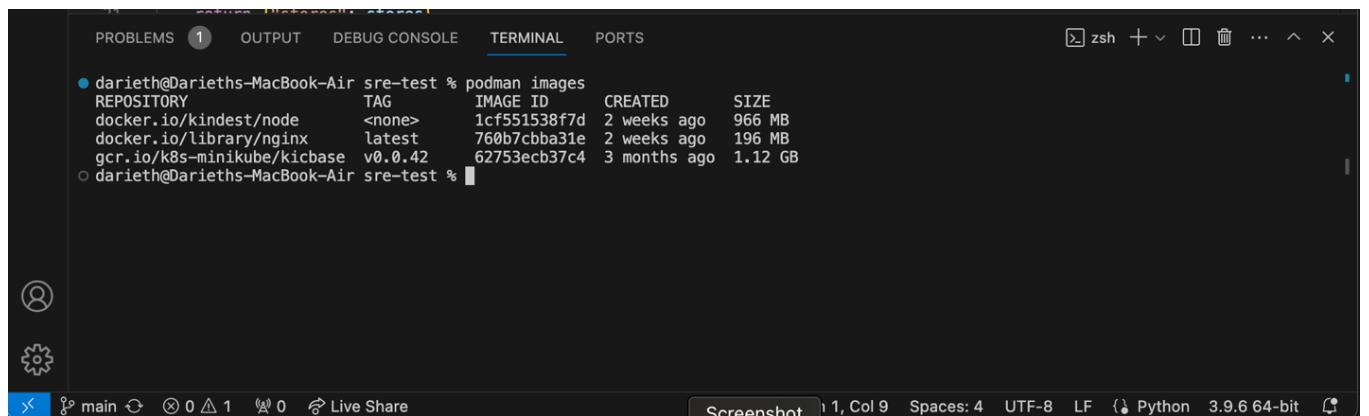


A screenshot of the Visual Studio Code interface. The terminal tab is active, showing the command `cat Dockerfile` and its output:

```
darieth@Darieths-MacBook-Air sre-test % cat Dockerfile
FROM python:3.10
EXPOSE 5000
WORKDIR /app
RUN pip install flask
COPY .
CMD ["flask", "run", "--host", "0.0.0.0"]
darieth@Darieths-MacBook-Air sre-test %
```

The status bar at the bottom shows: Screenshot 1, Col 9, Spaces: 4, UTF-8, LF, Python 3.9.6 64-bit.

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



A screenshot of the Visual Studio Code interface. The terminal tab is active, showing the command `podman images` and its output:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
docker.io/kindest/node	<none>	1cf551538f7d	2 weeks ago	966 MB
docker.io/library/nginx	latest	760b7cbba31e	2 weeks ago	196 MB
gcr.io/k8s-minikube/kicbase	v0.0.42	62753ecb37c4	3 months ago	1.12 GB

The status bar at the bottom shows: Screenshot 1, Col 9, Spaces: 4, UTF-8, LF, Python 3.9.6 64-bit.

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

\$podman build -t lab-application

A screenshot of the VS Code interface showing the terminal tab. The command run was `podman build -t lab-application .`. The output shows the build process for a Python application using Flask, including the creation of a Docker image and the installation of dependencies like Werkzeug, Click, and Jinja2. It also includes a warning about running pip as root and a notice about a pip update.

```
● darieth@Darieths-MacBook-Air sre-test % podman build -t lab-application .
STEP 1/6: FROM python:3.10
STEP 2/6: EXPOSE 5000
--> 73806bd3be04
STEP 3/6: WORKDIR /app
--> efb62b637884
STEP 4/6: RUN pip install flask
Collecting flask
  Downloading flask-3.0.2-py3-none-any.whl (101 kB)
Collecting Werkzeug>=3.0.0
  Downloading werkzeug-3.0.1-py3-none-any.whl (226 kB)
Collecting blinker>=1.6.2
  Downloading blinker-1.7.0-py3-none-any.whl (13 kB)
Collecting click>=8.1.3
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting Jinja2>=3.1.2
  Downloading Jinja2-3.1.3-py3-none-any.whl (133 kB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.5-cp310-cp310-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (26 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, flask
Successfully installed Jinja2-3.1.3 MarkupSafe-2.1.5 Werkzeug-3.0.1 blinker-1.7.0 click-8.1.7 flask-3.0.2 itsdangerous-2.1.2
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 > 24.0
[notice] To update, run: pip install --upgrade pip
--> 82bb0379cc108
STEP 5/6: COPY .
--> 2aa0d0fc79f2
STEP 6/6: CMD ["flask", "run", "--host", "0.0.0.0"]
COMMIT lab-application
```

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

A screenshot of the VS Code interface showing the terminal tab. The command run was `podman images`. The output lists several Docker images, including the newly built 'lab-application' image which has a tag of 'latest' and a size of 1.04 GB.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/lab-application	latest	d723759161d8	About a minute ago	1.04 GB
docker.io/kindest/node	<none>	1cf551538f7d	2 weeks ago	966 MB
docker.io/library/nginx	latest	760b7cbba31e	2 weeks ago	196 MB
docker.io/library/python	3.10	c074d3878dd2	4 weeks ago	1.03 GB
gcr.io/k8s-minikube/kicbase	v0.0.42	62753ecb37c4	3 months ago	1.12 GB

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

A screenshot of a terminal window within a dark-themed code editor interface. The terminal tab is active, showing the command `podman run -d lab-application` and its output. The status bar at the bottom indicates the file is `main*`, has 0 changes, 0 unstaged changes, and 0 unstaged conflicts. It also shows "Live Share" and the system status.

```
darieth@Darieths-MacBook-Air sre-test % podman run -d lab-application
eab5f8adfe985d77d3051a97c08469a30573d85d87b7cbcff031a42cd30945a3
○ darieth@Darieths-MacBook-Air sre-test %
```

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

A screenshot of a terminal window within a dark-themed code editor interface. The terminal tab is active, showing the command `podman ps` and its output, which lists a single container named `quirky_matsumoto`. The status bar at the bottom indicates the file is `main*`, has 0 changes, 0 unstaged changes, and 0 unstaged conflicts. It also shows "Live Share" and the system status.

```
darieth@Darieths-MacBook-Air sre-test % podman ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
eab5f8adfe98 localhost/lab-application:latest flask run --host ... 2 minutes ago Up 2 minutes
○ darieth@Darieths-MacBook-Air sre-test %
```

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

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

A screenshot of a terminal window within a dark-themed code editor interface. The terminal tab is active, showing the command `podman exec -t quirky_matsumoto curl http://127.0.0.1:5000/store` and its output, which is a JSON response from the application. The status bar at the bottom indicates the file is `main*`, has 0 changes, 0 unstaged changes, and 0 unstaged conflicts. It also shows "Live Share" and the system status.

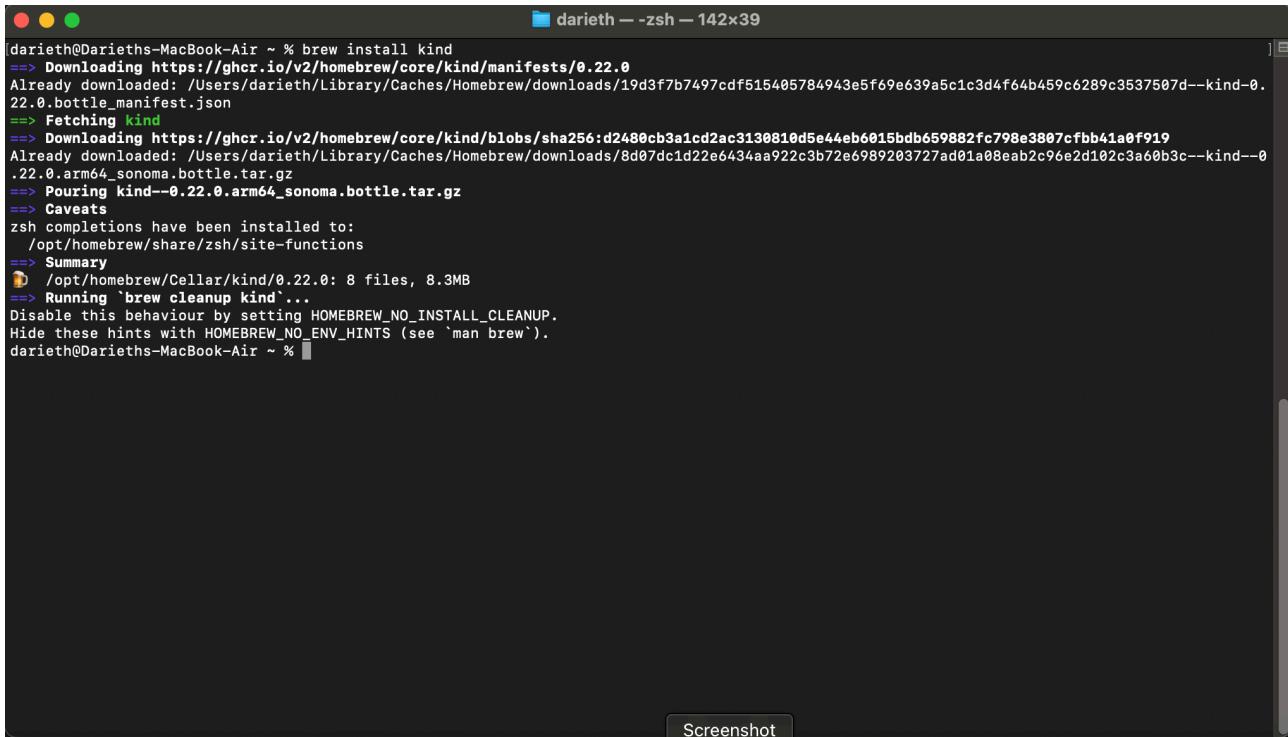
```
darieth@Darieths-MacBook-Air sre-test % podman exec -t quirky_matsumoto curl http://127.0.0.1:5000/store
{"stores": [{"name": "Chair", "price": 15.99}, {"name": "My Store"}]}
○ darieth@Darieths-MacBook-Air sre-test %
```

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.

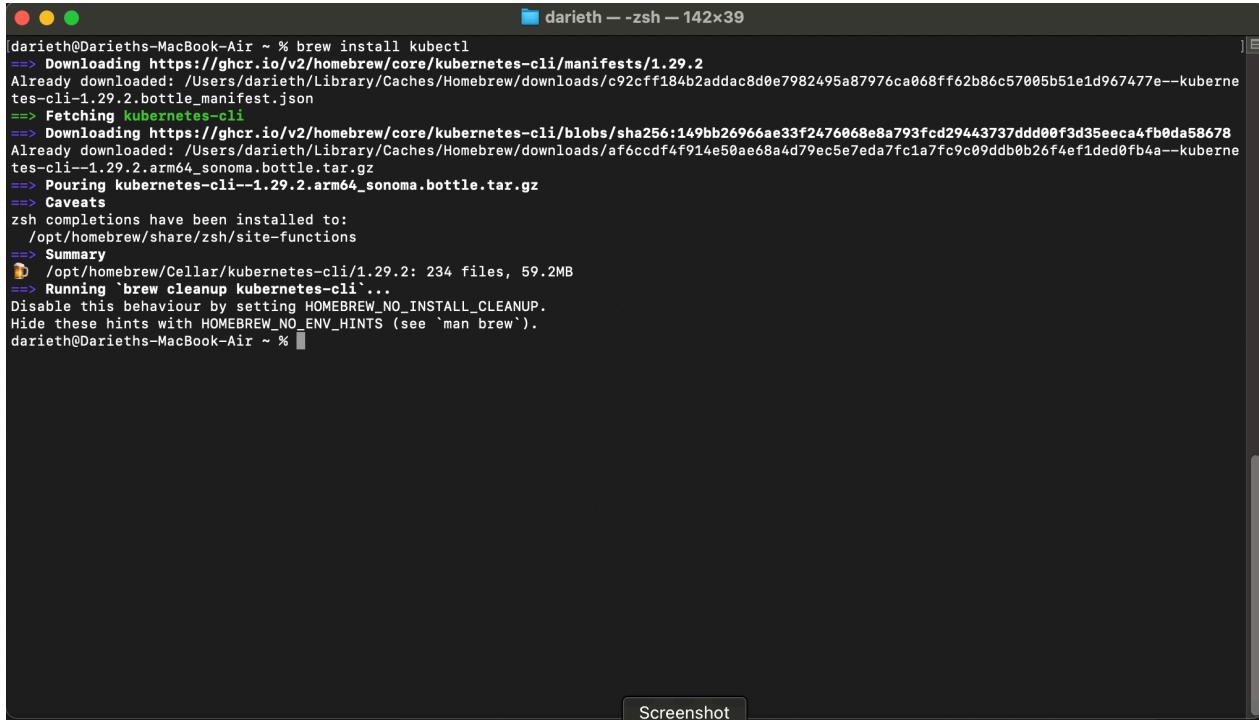
- First is necessary to install [kind](#) tool, this tool is used to create a kubernetes cluster. Run the command \$brew install kind.



The screenshot shows a terminal window titled "darieth -- zsh -- 142x39". The command "brew install kind" is being run. The output shows the download and installation process for kind version 0.22.0. It includes details about the URL, file names, and completion steps. A "Screenshot" button is visible at the bottom right of the terminal window.

```
[darieth@Darieths-MacBook-Air ~ % brew install kind
==> Downloading https://ghcr.io/v2/homebrew/core/kind/manifests/0.22.0
Already downloaded: /Users/darieth/Library/Caches/Homebrew/downloads/19d3f7b7497cdf515405784943e5f69e639a5c1c3d4f64b459c6289c3537507d--kind-0.22.0.bottle.manifest.json
==> Fetching kind
==> Downloading https://ghcr.io/v2/homebrew/core/kind/blobs/sha256:d2480cb3a1cd2ac3130810d5e44eb6015bdb659882fc798e3807cfbb41a0f919
Already downloaded: /Users/darieth/Library/Caches/Homebrew/downloads/8d07dc1d22e6434aa922c3b72e6989203727ad01a08eb2c96e2d102c3a60b3c--kind--0.22.0.arm64_sonoma.bottle.tar.gz
==> Pouring kind--0.22.0.arm64_sonoma.bottle.tar.gz
==> Caveats
zsh completions have been installed to:
/opt/homebrew/share/zsh/site-functions
==> Summary
🍺 /opt/homebrew/Cellar/kind/0.22.0: 8 files, 8.3MB
==> Running `brew cleanup kind`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
darieth@Darieths-MacBook-Air ~ % ]
```

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



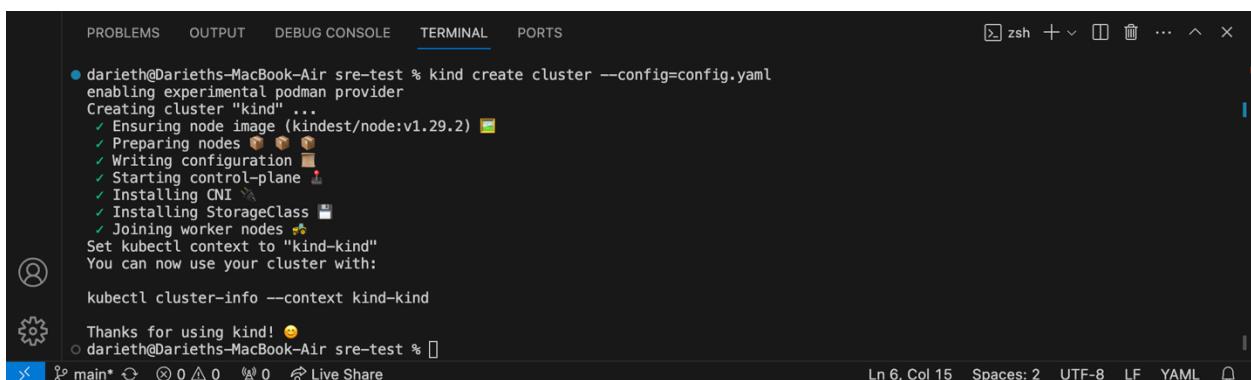
```
darieth@Darieths-MacBook-Air ~ % brew install kubectl
==> Downloading https://ghcr.io/v2/homebrew/core/kubernetes-cli/manifests/1.29.2
Already downloaded: /Users/darieth/Library/Caches/Homebrew/downloads/c92cff184b2addac8d0e7982495a87976ca068ff62b86c57005b51e1d967477e--kubernetes-cli-1.29.2.bottle_manifest.json
==> Fetching kubernetes-cli
==> Downloading https://ghcr.io/v2/homebrew/core/kubernetes-cli/blobs/sha256:149bb26966ae33f2476068e8a793fc29443737ddd00f3d35eeea4fb0da58678
Already downloaded: /Users/darieth/Library/Caches/Homebrew/downloads/af6ccdf4f914e50ae68a4d79ec5e7eda7fc1a7fc9c09ddb0b26f4ef1ded0fb4a--kubernetes-cli--1.29.2.arm64_sonoma.bottle.tar.gz
==> Pouring kubernetes-cli--1.29.2.arm64_sonoma.bottle.tar.gz
==> Caveats
zsh completions have been installed to:
/opt/homebrew/share/zsh/site-functions
==> Summary
🍺 /opt/homebrew/Cellar/kubernetes-cli/1.29.2: 234 files, 59.2MB
==> Running `brew cleanup kubernetes-cli`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
darieth@Darieths-MacBook-Air ~ %
```

Screenshot

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



```
darieth@Darieths-MacBook-Air sre-test % kind create cluster --config=config.yaml
enabling experimental podman provider
Creating cluster "kind" ...
✓ Ensuring node image (kindest/node:v1.29.2)
✓ Preparing nodes 🐳 🐳 🐳
✓ Writing configuration 📄
✓ Starting control-plane 🔥
✓ Installing CNI 🌐
✓ Installing StorageClass 🏷️
✓ Joining worker nodes 🛡️
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Thanks for using kind! 😊
darieth@Darieths-MacBook-Air sre-test %
```

Ln 6, Col 15 Spaces: 2 UTF-8 LF YAML

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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
darieth@Darieths-MacBook-Air sre-test % podman ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
72332297d571 docker.io/kindes/node@sha256:51a1434a539719342f0be2a297b488b6c919ce8a3931be0ce822606ea5ca245 3 minutes ago Up 3 minutes 127.0.0.1:51344->6443/tcp kind-control-plane
85e1542a0475 docker.io/kindes/node@sha256:51a1434a539719342f0be2a297b488b6c919ce8a3931be0ce822606ea5ca245 3 minutes ago Up 3 minutes kind-worker2
b4199eb83275 docker.io/kindes/node@sha256:51a1434a539719342f0be2a297b488b6c919ce8a3931be0ce822606ea5ca245 3 minutes ago Up 3 minutes kind-worker

darieth@Darieths-MacBook-Air sre-test % kubectl get nodes
NAME STATUS ROLES AGE VERSION
kind-control-plane Ready control-plane 5m18s v1.29.2
kind-worker Ready <none> 5m v1.29.2
kind-worker2 Ready <none> 4m59s v1.29.2

darieth@Darieths-MacBook-Air sre-test %

```

The screenshot shows a terminal window with several tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is active), and PORTS. The TERMINAL tab displays the results of running 'podman ps' and 'kubectl get nodes'. The 'podman ps' command lists three containers: 'kind-control-plane', 'kind-worker2', and 'kind-worker'. The 'kubectl get nodes' command shows one node named 'kind-control-plane' which is ready and has the role of 'control-plane'. The status is 'Up 3 minutes' and the version is 'v1.29.2'.

- 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

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
darieth@Darieths-MacBook-Air sre-test % podman push xuthyst/lab-application
Getting image source signatures
Copying blob sha256:677bb2f56ab38f9396e5c49bfc697f05f93c13e0dc2d78a1a879390c9535be87
Copying blob sha256:973599cf2dadaf3755ae7e1322a8fe288c0e30bcdce59ade49b71a18c388a1fe
Copying blob sha256:b10a49b17ae62fc1fc89fbf0473a879599168554d24490433ec580f685c2b879
Copying blob sha256:d9c6bbb93e@08d5c4175bcf74d9a31971e58f8a79fffb942f31565aead6@08d
Copying blob sha256:a974964b27e5246ceec487fc16bd743848f766ea0d62afe6ded2b3ee12ff0699
Copying blob sha256:9ce63ba53cb8d4a9981a3bf4881af9094f2cd20372a77500274a6c63a24a166
Copying blob sha256:d2c4229699226ed3815e051952a44ec09fa6689ef3e8c9f862dede71841e00df
Copying blob sha256:1bba2978b67a53ab211de51c2820cbd32d5c7c009eb5a3bc7382068c8a16e1a
Copying blob sha256:583f4ef4fceac54f771eb050170dbd371ee1e7034ee8cf98036012367ddebbe84
Copying blob sha256:0064718c9d61156a28986f0f7b9cbe46c262a7ac740c739e9ead9e5f00fb291a
Copying config sha256:d723759161d8e3d3fae94427e475009a72ba0617da6009f6bf70ac0e1f94e0
Writing manifest to image destination
darieth@Darieths-MacBook-Air sre-test %

```

The screenshot shows a terminal window with several tabs: PROBLEMS (with 1 error), OUTPUT, DEBUG CONSOLE, TERMINAL (active), and PORTS. The TERMINAL tab shows the output of the 'podman push xuthyst/lab-application' command. It indicates that the image is being copied and pushed to Docker Hub, with various blob IDs and file paths listed. The command completes successfully at the end.

- 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**

```
darieth@Darieths-MacBook-Air sre-test % kubectl get nodes
NAME           STATUS   ROLES      AGE   VERSION
kind-control-plane   Ready    control-plane   4m46s   v1.29.2
kind-worker       Ready    <none>     4m28s   v1.29.2
kind-worker2      Ready    <none>     4m27s   v1.29.2
darieth@Darieths-MacBook-Air sre-test %
```

The screenshot shows a terminal window with the title bar "zsh". The terminal is displaying the output of the command "kubectl get nodes". It lists three nodes: "kind-control-plane", "kind-worker", and "kind-worker2", all in a "Ready" state with no roles assigned. The "VERSION" column shows v1.29.2. The terminal window has a dark background and light-colored text.

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

```
darieth@Darieths-MacBook-Air sre-test % kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
lab-application-669c675854-df4pp   1/1     Running   0          11m
lab-application-669c675854-fn9wp   1/1     Running   0          11m
darieth@Darieths-MacBook-Air sre-test %
```

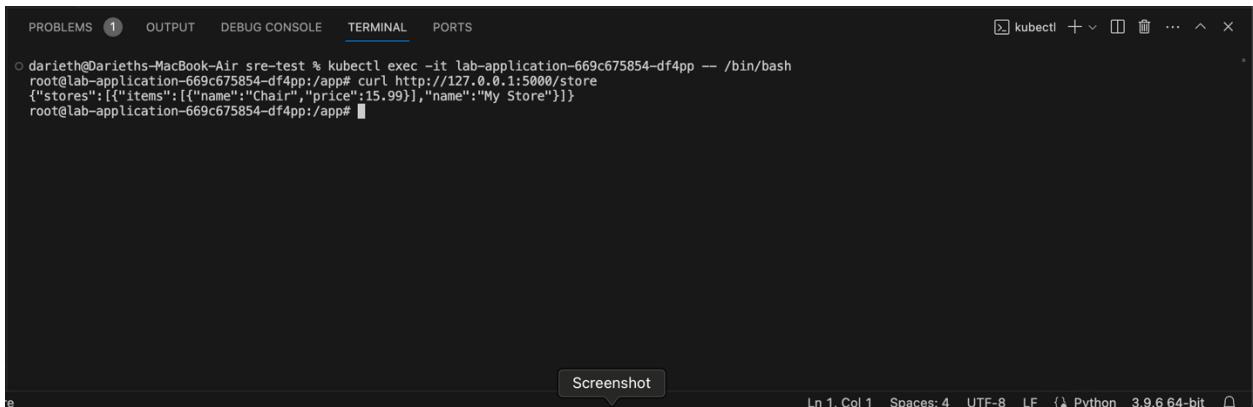
The screenshot shows a terminal window with the title bar "zsh". The terminal is displaying the output of the command "kubectl get pods". It lists two pods: "lab-application-669c675854-df4pp" and "lab-application-669c675854-fn9wp", both in a "Running" state with 0 restarts. The "AGE" column shows 11m for both. The terminal window has a dark background and light-colored text.

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

```
darieth@Darieths-MacBook-Air sre-test % kubectl get all
NAME                                         READY   STATUS    RESTARTS   AGE
pod/lab-application-669c675854-df4pp        1/1     Running   0          12m
pod/lab-application-669c675854-fn9wp        1/1     Running   0          12m
NAME          TYPE    CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP  10.96.0.1   <none>        443/TCP   14m
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/lab-application   2/2     2           2           12m
NAME          DESIRED  CURRENT   READY   AGE
replicaset.apps/lab-application-669c675854  2        2           2           12m
darieth@Darieths-MacBook-Air sre-test %
```

The screenshot shows a terminal window with the title bar "zsh". The terminal is displaying the output of the command "kubectl get all". It provides a comprehensive list of resources: two pods ("lab-application-669c675854-df4pp" and "lab-application-669c675854-fn9wp"), one service ("kubernetes" with IP 10.96.0.1), one deployment ("lab-application" with 2/2 replicas up-to-date), and one replicaset ("lab-application-669c675854" with 2 desired, 2 current, and 2 ready replicas). The "AGE" column shows the creation time for each resource. The terminal window has a dark background and light-colored text.

- 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



A screenshot of a terminal window within a dark-themed IDE interface. The terminal tab is active, showing the command line history:

```
darieth@Darieths-MacBook-Air sre-test % kubectl exec -it lab-application-669c675854-df4pp -- /bin/bash
root@lab-application-669c675854-df4pp:/app# curl http://127.0.0.1:5000/store
{"stores":[{"items":[{"name":"Chair","price":15.99}],"name":"My Store"}]}
root@lab-application-669c675854-df4pp:/app#
```

The terminal window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. At the bottom right, it shows "Screenshot" and "Ln 1, Col 1, Spaces: 4, UTF-8, LF, Python 3.9.6 64-bit".

Section 3.

In this case, this solution will use GitHub Actions to make the deployment of the application into a Kubernetes cluster.

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.

The screenshot shows the GitHub Repository Secrets page. On the left, there's a sidebar with options like Branches, Tags, Rules, Actions, Webhooks, Codespaces, and Pages. Under Security, there are links for Code security and analysis and Deploy keys. The 'Secrets and variables' option is selected. The main area is titled 'Repository secrets' and contains a table with four rows:

Name	Last updated	Action
AWS_ACCESS_KEY	2 hours ago	
AWS_SECRET_KEY	2 hours ago	
DOCKER_TOKEN	yesterday	
DOCKER_USER	yesterday	

A green button at the top right says 'New repository secret'. A message box at the top states: 'During a recent maintenance event some secret timestamps were inadvertently updated. The secrets were not changed, only the last updated timestamp. We are working on restoring the original date.'

```
name: CI/CD Pipeline

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest

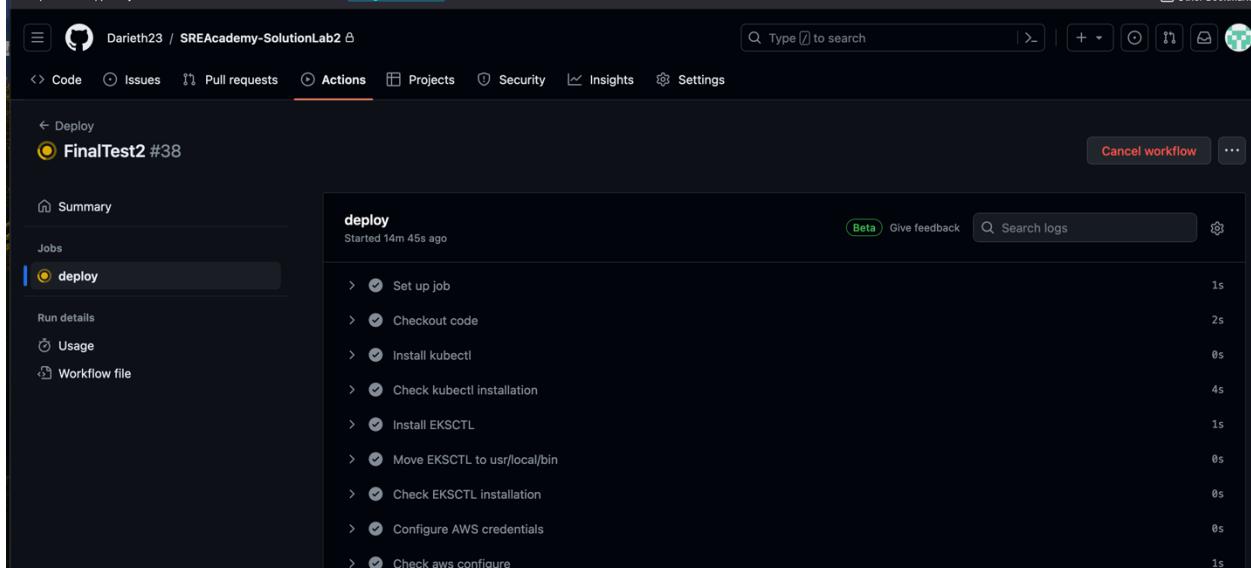
    steps:
      - name: Check EKSCTL installation
        run: eksctl version

      #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

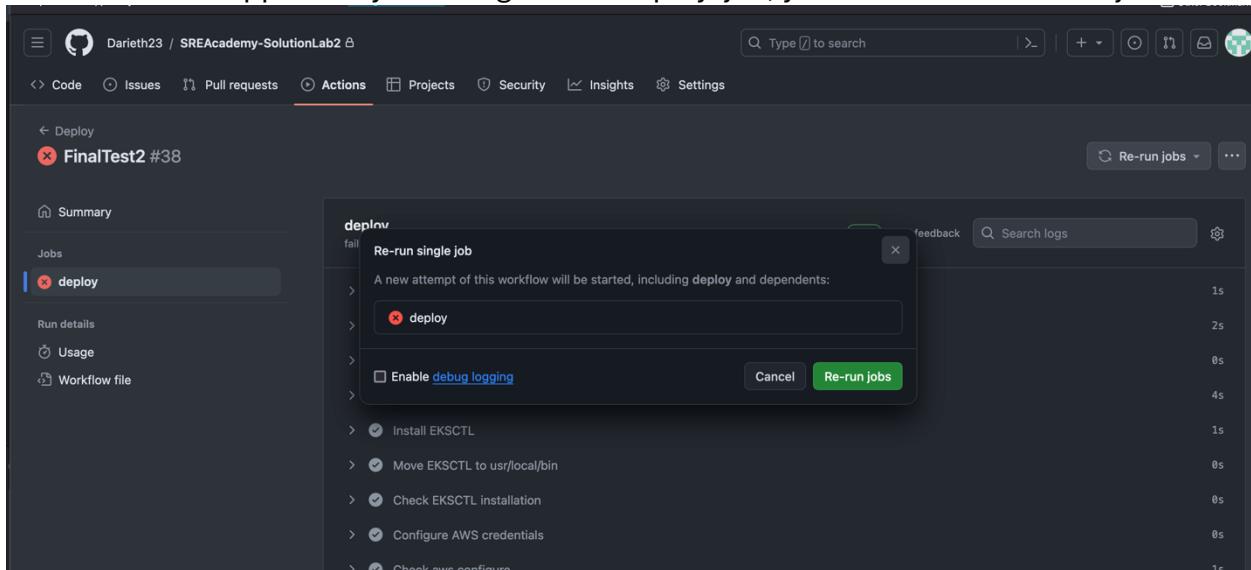
      - name: Check aws configure
        run: aws configure list

      #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 follows **http://URL-PROVIDED/store**.

The screenshot shows the AWS EC2 Load Balancers page. On the left, there's a sidebar with options like EC2 Dashboard, EC2 Global View, Events, Console-to-Code, Instances, Images, and AMIs. The main area is titled "Load balancers (2)" and contains a table with two rows. The first row has a link to "a982794120e594bc9a..." and the second to "a574a3cb6f1104687af...". Both rows show a "State" of "-" and a "VPC ID" of "vpc-0ee7ec35c..." and "vpc-05feab021" respectively.

The screenshot shows a browser window with a JSON response. The URL is "a574a3cb6f1104687afb92e6c0678c42-1831356672.us-east-1.elb.amazonaws.com/store". The JSON data is as follows:

```

stores:
  0:
    items:
      0:
        name: "Chair"
        price: 15.99
        name: "My 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.