Wisecow Application – Setup

Problem-1:

1. Clone the Repository

Clone the Wisecow project from GitHub:

```
git clone https://github.com/nyrahul/wisecow.git
cd wisecow
```

```
root@ubuntu-jammy:~# git clone https://github.com/nyrahul/wisecow.git
Cloning into 'wisecow'...
remote: Enumerating objects: 31, done.
remote: Counting objects: 100% (2/2), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 31 (delta 0), reused 0 (delta 0), pack-reused 29 (from 2)
Receiving objects: 100% (31/31), 11.98 KiB | 177.00 KiB/s, done.
Resolving deltas: 100% (7/7), done.
root@ubuntu-jammy:~#
```

---

2. Install Prerequisites

The application depends on `fortune-mod` and `cowsay`.

Install them using:

```
sudo apt update
sudo apt install fortune-mod cowsay -y
```

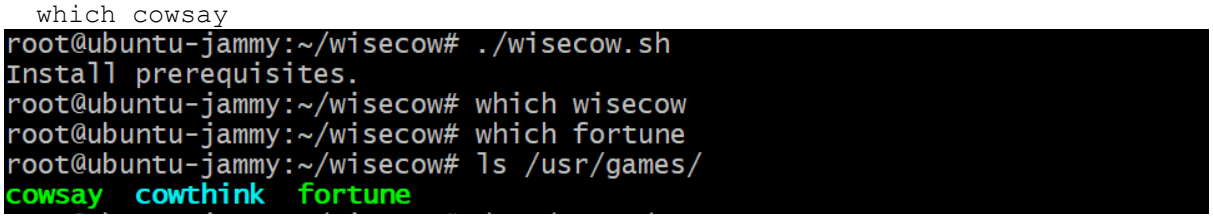
```
root@ubuntu-jammy: ~/wisecow
root@ubuntu-jammy:~/wisecow# apt install fortune-mod cowsay -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  fortunes-min librecode0
Suggested packages:
  filters cowsay-off fortunes x11-utils bsdmainutils
The following NEW packages will be installed:
  cowsay fortune-mod fortunes-min librecode0
0 upgraded, 4 newly installed, 0 to remove and 51 not upgraded.
Need to get 781 kB of archives.
After this operation, 2248 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/universe amd64 cowsay all 3.03+dfsg2-8 [18.6 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy/main amd64 librecode0 amd64 3.6-24build2 [670 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy/universe amd64 fortune-mod amd64 1:1.99.1-7.1 [36.9 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy/universe amd64 fortunes-min all 1:1.99.1-7.1 [55.1 kB]
Fetched 781 kB in 4s (219 kB/s)
Selecting previously unselected package cowsay.
(Reading database ... 64254 files and directories currently installed.)
Preparing to unpack .../cowsay_3.03+dfsg2-8_all.deb ...
Unpacking cowsay (3.03+dfsg2-8) ...
Selecting previously unselected package librecode0:amd64.
Preparing to unpack .../librecode0_3.6-24build2_amd64.deb ...
Unpacking librecode0:amd64 (3.6-24build2) ...
Selecting previously unselected package fortune-mod.
Preparing to unpack .../fortune-mod_1%3a1.99.1-7.1_amd64.deb ...
Unpacking fortune-mod (1:1.99.1-7.1) ...
Selecting previously unselected package fortunes-min.
Preparing to unpack .../fortunes-min_1%3a1.99.1-7.1_all.deb ...
Unpacking fortunes-min (1:1.99.1-7.1) ...
Setting up librecode0:amd64 (3.6-24build2) ...
Setting up fortunes-min (1:1.99.1-7.1) ...
Setting up fortune-mod (1:1.99.1-7.1) ...
Setting up cowsay (3.03+dfsg2-8) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for libc-bin (2.35-0ubuntu3.10) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.
```

Verify installation:

```
which fortune
which cowsay
```



Expected output (location may vary):

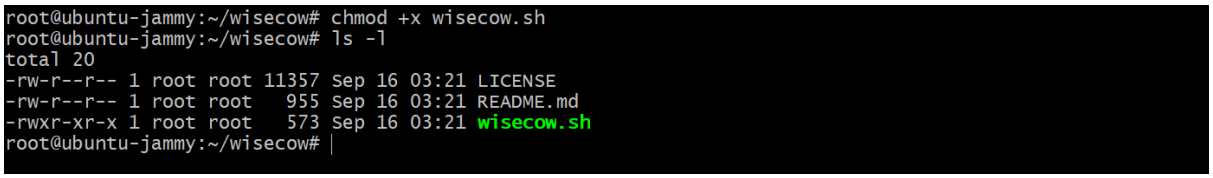
```
/usr/games/fortune
/usr/games/cowsay
```

---

3. Make Script Executable

Change permissions of `wisecow.sh` to make it executable:

```
chmod +x wisecow.sh
```



Check with:

```
ls -l
```

You should see `wisecow.sh` marked as executable (`-rwxr-xr-x`).

---

4. Run the Wisecow Script

Execute the script:

```
./wisecow.sh
```

If prerequisites are missing, it will prompt with:

```
Install prerequisites.
```

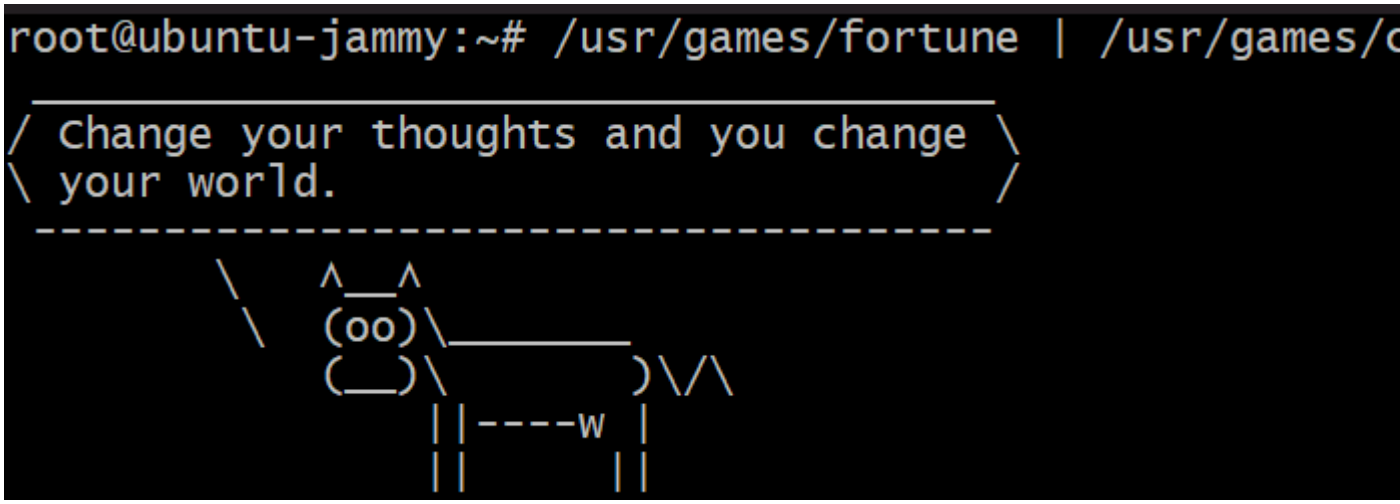
---

5. Test Manually

You can also test the functionality directly:

```
/usr/games/fortune | /usr/games/cowsay
```

Expected output (random quote with ASCII cow):



Step-6: Update PATH Variable

To include `/usr/games` in the system PATH so that commands from this directory can be executed globally:

```
echo 'export PATH=$PATH:/usr/games' >> ~/.bashrc
source ~/.bashrc
```

```
root@ubuntu-jammy:~# echo 'export PATH=$PATH:/usr/games' >> ~/.bashrc
root@ubuntu-jammy:~# source ~/.bashrc
```

Step-7: Verify Network Configuration

Check available network interfaces and their assigned IP addresses:

```
ip addr show
```

```
root@ubuntu-jammy: ~/wisecow
root@ubuntu-jammy:~/wisecow# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 02:17:d8:8d:76:f6 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 metric 100 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 86337sec preferred_lft 86337sec
    inet6 fd17:625c:f037:2:17:d8ff:fe8d:76f6/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 86338sec preferred_lft 14338sec
    inet6 fe80::17:d8ff:fe8d:76f6/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:c7:99:50 brd ff:ff:ff:ff:ff:ff
    inet 192.168.33.10/24 brd 192.168.33.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fec7:9950/64 scope link
       valid_lft forever preferred_lft forever
```

In this case, we use `192.168.33.10` to access the application in the browser.

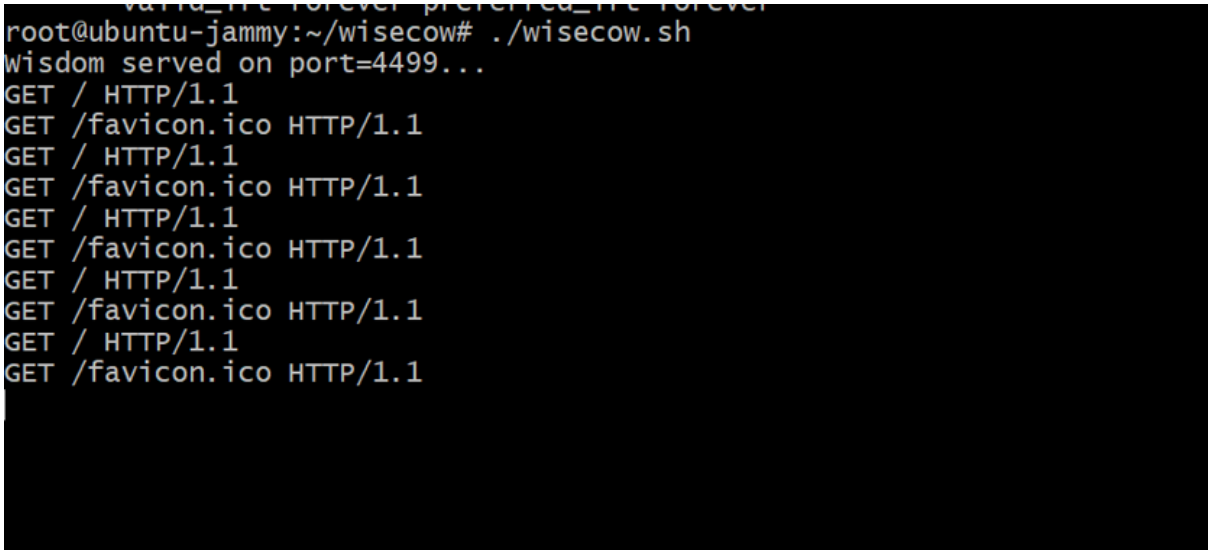
Step-8: Start Wisecow Application

Navigate to the project directory and run the application:

```
cd ~/wisecow
./wisecow.sh
```

Output:

```
Wisdom served on port=4499...
```

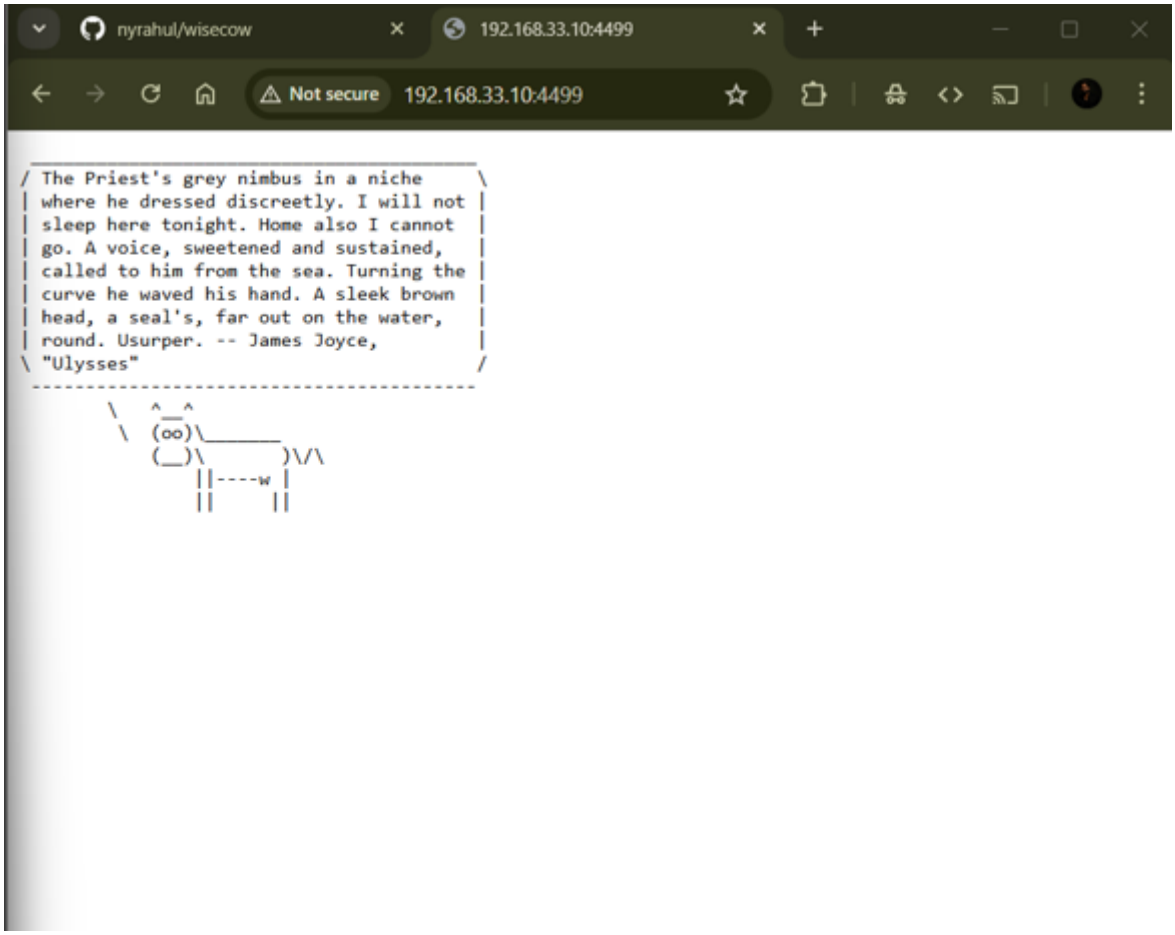


This indicates the server is running on **port 4499**.

Step-9: Access Application in Browser

Open a browser and navigate to:

http://192.168.33.10:4499



✅ The Wisecow service should display random quotes with ASCII cow art.

Step-10: Run Wisecow in Background with Logging

To keep the server running in the background and log output:

```
./wisecow.sh > wisecow.log 2>&1 &
```

- `>` → Redirects standard output to `wisecow.log`
- `2>&1` → Redirects errors (stderr) to the same log file
- `&` → Runs the process in the background

Check logs in real-time:

```
tail -f wisecow.log
```

```
root@ubuntu-jammy:~/wisecow# ./wisecow.sh > wisecow.log 2>&1 &
[2] 1436
root@ubuntu-jammy:~/wisecow# tail -f wisecow.log
Wisdom served on port=4499...


GET / HTTP/1.1
GET /favicon.ico HTTP/1.1
GET / HTTP/1.1
GET /favicon.ico HTTP/1.1
GET / HTTP/1.1
^Z
[3]+  Stopped                 tail -f wisecow.log
root@ubuntu-jammy:~/wisecow#
```

Press `Ctrl + C` to stop log monitoring.

DOCKERIZATION:

Step 1: Building the Docker Image

This image shows the `docker build` command being executed. The command `docker build -t sriram084/wisecow:base .` builds a Docker image from a `Dockerfile` in the current directory.

- `-t sriram084/wisecow:base` tags the image with the repository name `sriram084/wisecow` and the tag `base`.
- The output shows the build process, including the download of the base image (`ubuntu:22.04`), the installation of dependencies like `cowsay` and `netcat-openbsd`, and the copying of the `wisecow.sh` script.
- The final step confirms the image has been successfully built and tagged.

Step 2: Running the Docker Container

- This image demonstrates running the newly created Docker image as a container. The command used is `docker run -d -p 4499:4499 --name wisecow-container sriram084/wisecow:base`.
  - `-d` runs the container in detached mode (in the background).
  - `-p 4499:4499` maps port 4499 on the host machine to port 4499 inside the container, making the application accessible from outside.

- `--name wisecow-container` gives a custom name to the container for easy reference.
- The output shows the long container ID, confirming that the container has started successfully.

```
ubuntu@ip-172-31-26-156:~/wisecow$ docker run -d -p 4499:4499 --name wisecow-container sriram084/wisecow:base
f7a594e39061d4d6f8751287b1e5eae7a6cb692c5ecfae933ace906f553046e6
```

- This screenshot verifies that the container is running. The `docker ps` command lists all running containers.
  - The output confirms the `wisecow-container` is running, showing its container ID, the image it's based on, the command it's executing, and the port mapping (`0.0.0.0:4499->4499/tcp`).

```
ubuntu@ip-172-31-26-156:~/wisecow$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                                       N
AMES
f7a594e39061   sriram084/wisecow:base   "/usr/local/bin/wise…"   16 seconds ago   Up 15 seconds   0.0.0.0:4499->4499/tcp, [::]:4499->4499/tcp   w
isecow-container
```

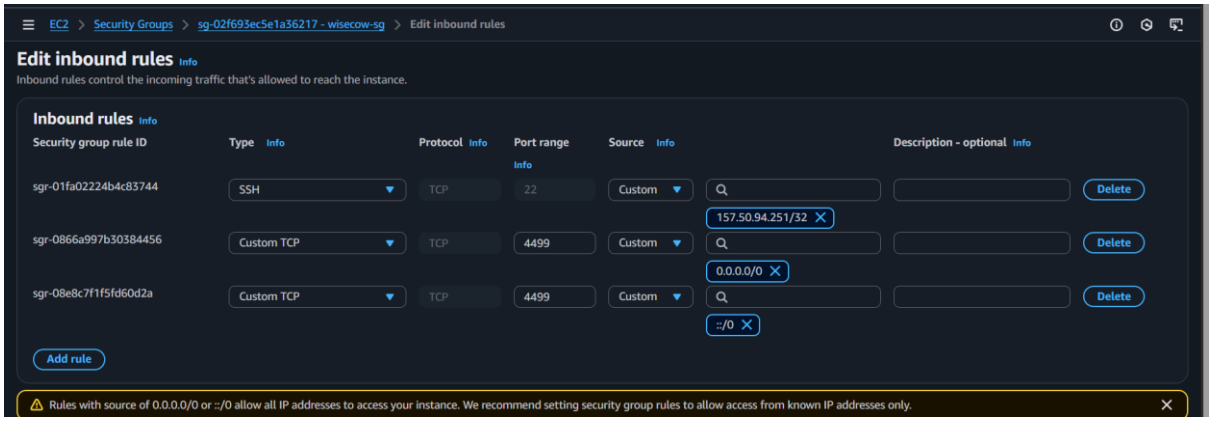
Step 3: Checking the Public IP Address

- This image shows how to find the public IP address of the host machine using the `curl ifconfig.me` command.
  - The command returns the IP address `54.226.12.122`, which is necessary for accessing the application from a web browser.

```
ubuntu@ip-172-31-26-156:~/wisecow$ curl ifconfig.me
54.226.12.122ubuntu@ip-172-31-26-156:~/wisecow$
```

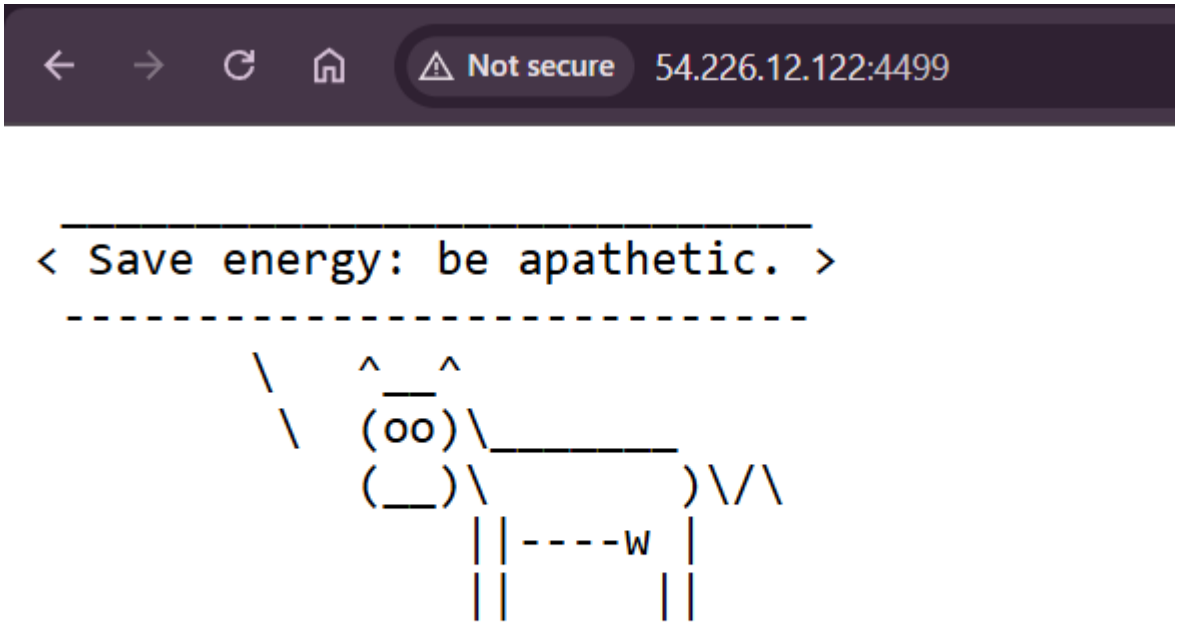
Step 4: Configuring Security Groups (AWS EC2)

- This image is from the AWS Management Console, showing the security group configuration for the EC2 instance where the Docker container is running.
  - It confirms that an inbound rule for **Custom TCP** on **Port range 4499** has been added.
  - The `Source` is set to `0.0.0.0/0` and `::/0`, which allows all IPv4 and IPv6 addresses to access the port. This is a critical step to ensure the application is publicly accessible.



Step 5: Accessing the Application

- This screenshot shows the "wisecow" application being accessed via a web browser.
  - The URL `54.226.12.122:4499` is used, combining the host's public IP address with the mapped port.
  - The page successfully loads and displays the output from the `cowsay` command: an ASCII cow with the text "Save energy: be apathetic."



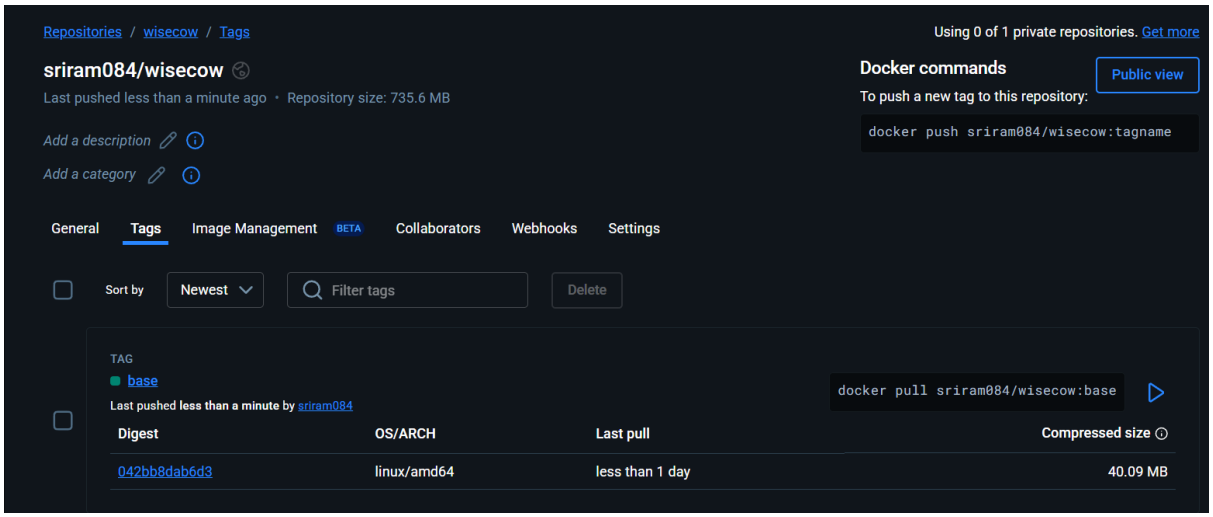
Step 6: Pushing the Image to Docker Hub

- This screenshot shows the command to push the local Docker image to a remote Docker repository.
  - The command is `docker push sriram084/wisecow:base`.
  - The output confirms the layers of the image are being pushed to `docker.io/sriram084/wisecow`. Some layers already exist on the repository, so they are not pushed again.
  - The final line shows the digest and size of the pushed image.

```
ubuntu@ip-172-31-26-156:~/wisecow$ docker push sriram084/wisecow:base
The push refers to repository [docker.io/sriram084/wisecow]
bc109005c03b: Pushed
1c2fe7af9f78: Pushed
e05a529f2619: Pushed
dc6eb6dad5f9: Layer already exists
base: digest: sha256:042bb8dab6d350ad75f5a732c3e7b5062660ba4305704dd4356525f7722f3290 size: 1155
```

- This image is from the Docker Hub repository page for `sriram084/wisecow`.
  - It visually confirms that the `base` tag has been successfully pushed.

- Details like the last pushed time, the digest (`042bb8dab6d3`), and the compressed size are displayed, confirming the image is now available on Docker Hub for others to pull .



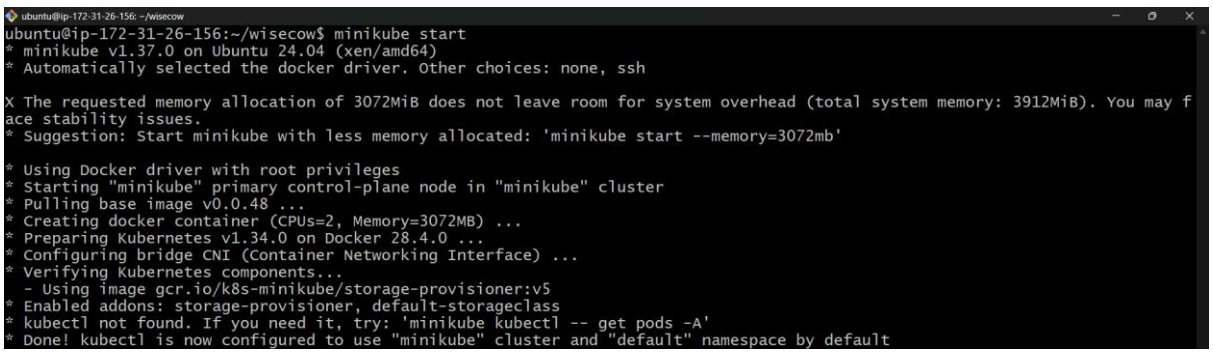
Docker file: https://github.com/sriramch163/wisecow/blob/main/Dockerfile

KUBERNETES DEPLOYMENT(using Minikube):

Step 1: Starting the Minikube Cluster

The first step is to start a local Kubernetes cluster with Minikube. The command `minikube start` is used, and it automatically selects the Docker driver for the cluster.

- The system allocates memory for the cluster (3072 MiB), and Minikube warns that this is a large allocation for the total system memory (3912 MiB), which could lead to stability issues.
- Minikube then starts the control plane, pulls a base image, and creates the container for the Kubernetes cluster.
- It configures the **Container Network Interface (CNI)** and verifies all Kubernetes components.
- Finally, the cluster is ready, and `kubectl` is configured to use it.



Step 2: Deploying the Application

With the Minikube cluster running, the application is deployed from a set of Kubernetes manifest files.

- The user lists the contents of the `k8s/` directory, which contains three files: apply
  - `wisecow-deployment.yaml` LINK:https://github.com/sriramch163/wisecow/blob/main/k8s/wisecow-deployment.yaml
  - `wisecow-service.yaml` LINK: https://github.com/sriramch163/wisecow/blob/main/k8s/wisecow-service.yaml
- The deployment is created using the command `kubectl create -f k8s/wisecow-deployment.yaml`.
- The output `deployment.apps/wisecow-deployment created` confirms that the application's deployment was successfully initiated.

```
ubuntu@ip-172-31-26-156:~/wisecow$ ls k8s/
wisecow-deployment.yaml  wisecow-ingress.yaml  wisecow-service.yaml
ubuntu@ip-172-31-26-156:~/wisecow$ kubectl create -f k8s/wisecow-deployment.yaml
deployment.apps/wisecow-deployment created
```

Step 3: Exposing the Service

After the deployment, a Kubernetes service is created to expose the application.

- The command `kubectl get svc` is used to list the services running in the cluster.
- The output shows two services: `kubernetes` (a default service) and `wisecow-service`.
- The **`wisecow-service`** has a **`NodePort`** type, which means it exposes the application on a specific port on each node in the cluster.
- The service maps port `4499` of the application to the external port **`32193`** on the node.

```
ubuntu@ip-172-31-26-156: ~/wisecow
ubuntu@ip-172-31-26-156:~/wisecow$ kubectl get svc
NAME             TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes       ClusterIP   10.96.0.1      <none>        443/TCP          9m9s
wisecow-service  NodePort    10.101.57.9    <none>        4499:32193/TCP   25s
```

Step 4: Accessing the Service Internally

To verify that the service is running correctly, it's accessed from within the Minikube environment.

- The command `minikube service wisecow-service --url` is run to get the internal URL of the service.
- The output is `http://192.168.49.2:32193`.
- A `curl` command is then used to access this URL: `curl http://192.168.49.2:32193`.

- The response is an HTML page displaying an "ASCII cow" and a joke, confirming that the service is active and serving content.

```
ubuntu@ip-172-31-26-156:~/wisecow$ minikube service wisecow-service --url
http://192.168.49.2:32193
ubuntu@ip-172-31-26-156:~/wisecow$ curl http://192.168.49.2:32193
<!DOCTYPE html>
<html><head><title>Wisecow</title></head><body>
<pre> ________________________________________
/ An avocado-tone refrigerator would look \
\ good on your resume.                    /
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||</pre>
</body></html>
```

Step 5: Accessing the Service Externally

For external access (e.g., from a web browser), a port-forwarding rule and a security rule must be configured.

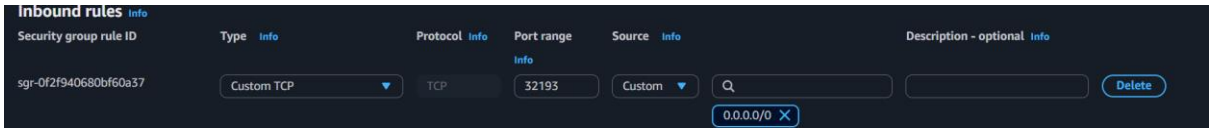
Port-Forwarding

- A `kubectl` command is used to forward the internal port to an external address: `kubectl port-forward svc/wisecow-service 32193:4499 --address 0.0.0.0 &`.
- This command forwards traffic from port **`32193`** on the host machine (`0.0.0.0`) to the container's port **`4499`**. The `&` symbol runs the command in the background.

```
ubuntu@ip-172-31-26-156:~/wisecow$ kubectl port-forward svc/wisecow-service 32193:4499 --address 0.0.0.0 &
[1] 8419
ubuntu@ip-172-31-26-156:~/wisecow$ Forwarding from 0.0.0.0:32193 -> 4499
```

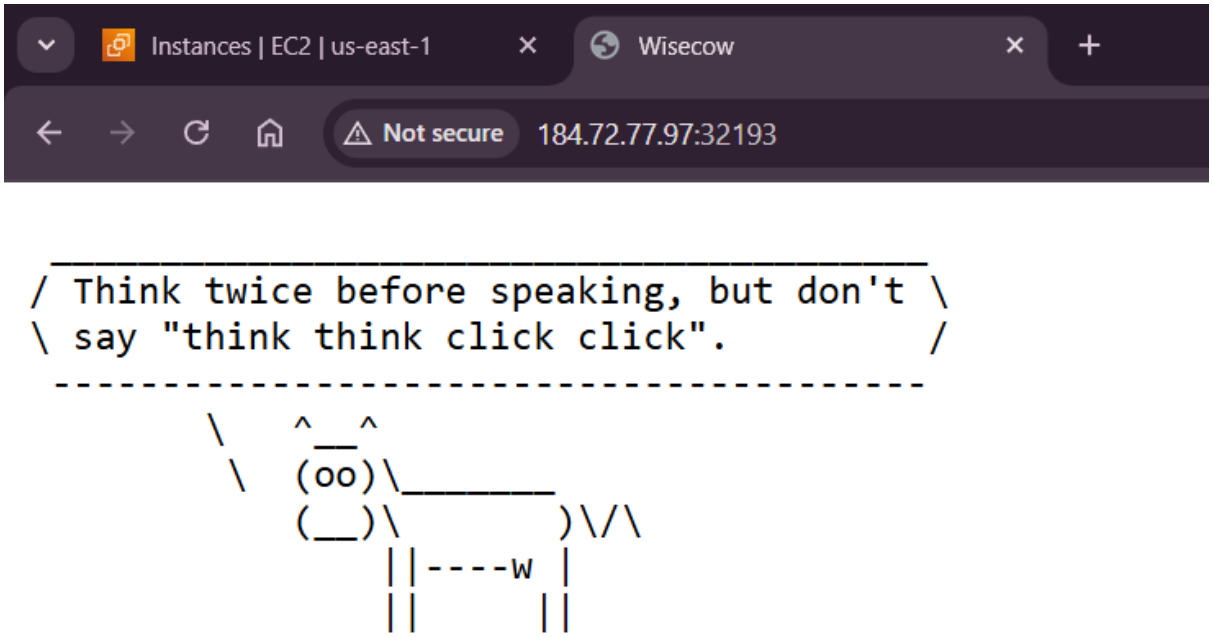
Security Group Rule

- A security rule is added to allow inbound traffic to the host on the forwarded port.
- The rule allows **Custom TCP** traffic on port **`32193`** from any source (`0.0.0.0/0`). This makes the service accessible from the internet.



External Access from a Browser

- With the security rule and port-forwarding in place, the application is accessed from a web browser using the host's public IP address and the exposed port: `184.72.77.97:32193`.
- The browser successfully loads the page, displaying a different "ASCII cow" and another joke, confirming that the application is accessible from outside the cluster.



Github-CICD:

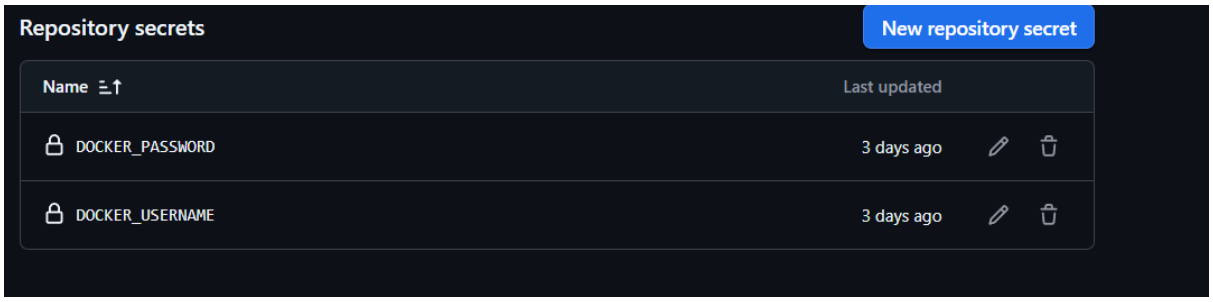
Githubworkflow files:
https://github.com/sriramch163/wisecow/tree/main/.github/workflows

In this there are 3 workflows:

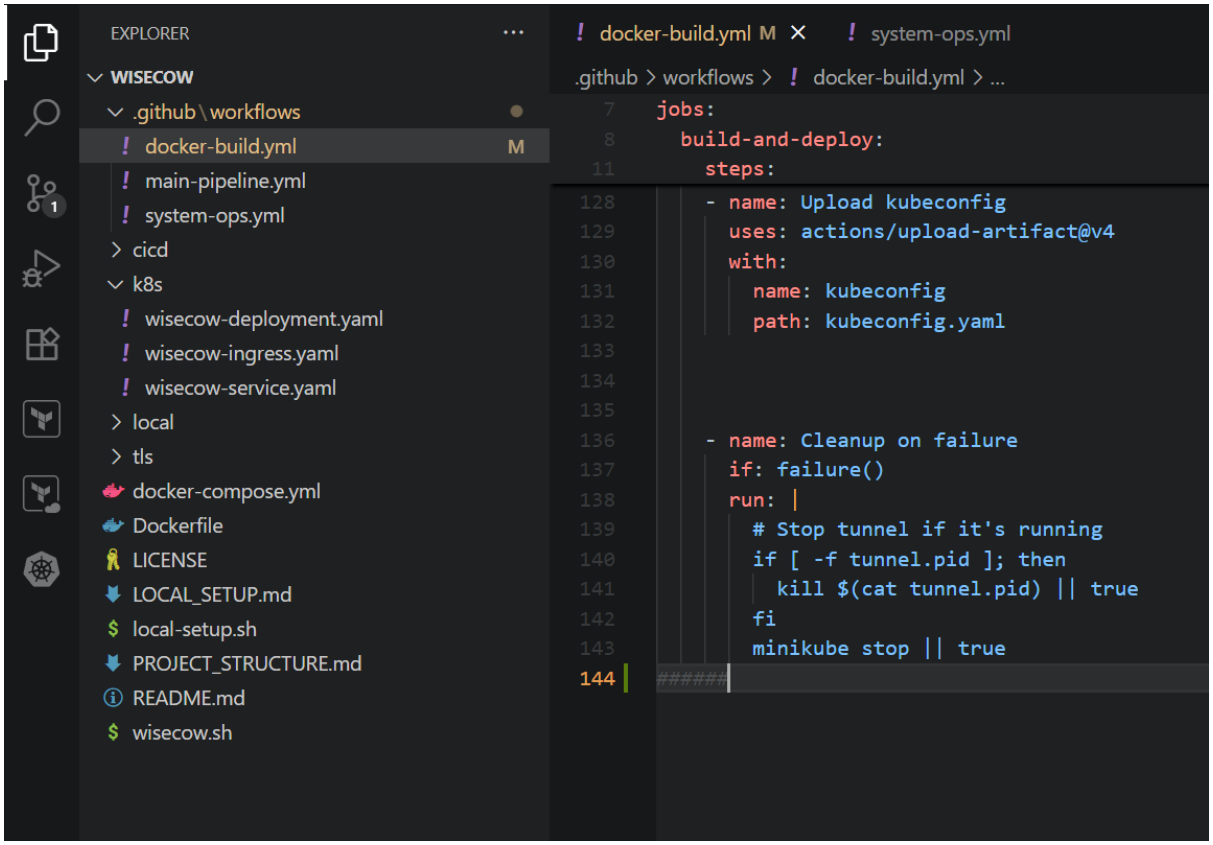
- main-pipeline.yml: which it inherits the docker-build.yml & system-os.yml
- docker-build.yml: which it is to setup build from the docker hub, using the credentials and build the new image that image is to be pushed into the minikube cluster.
- system-ops.yml: this build is to used the check the health status after the completion of the this job, then only it runs the backup job. Where it can backups the tls, dockerfile, manifest files, etc.,,

How the build Runs:

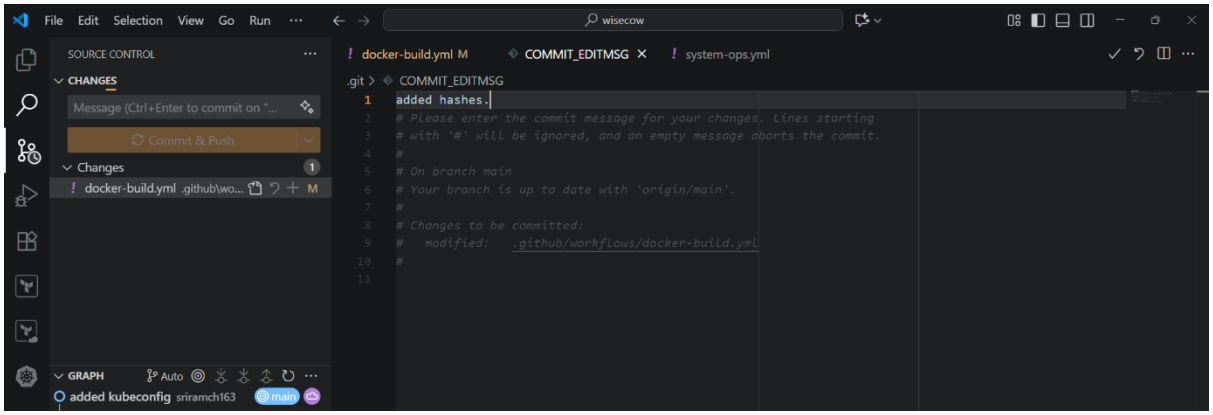
- In order to run we have the credentials inorder to build the docker image. And store it in github secrets.



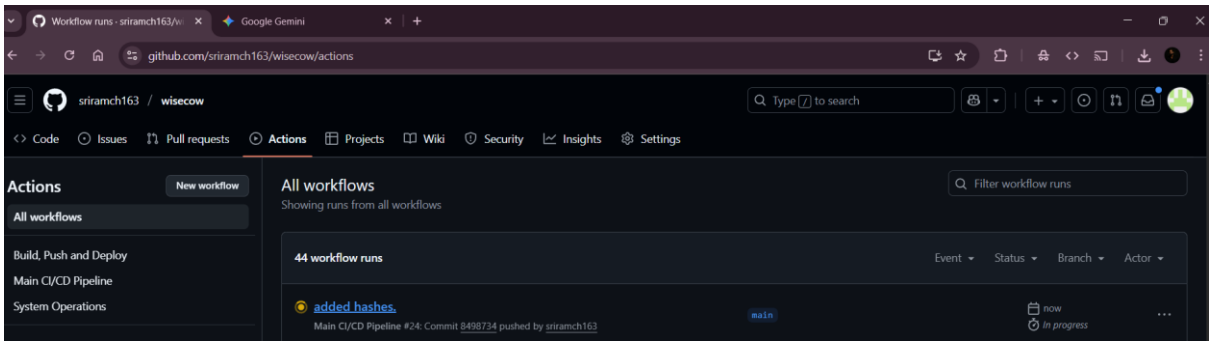
- Next, the job triggers when the changes are happended in the file and pushes into the github.



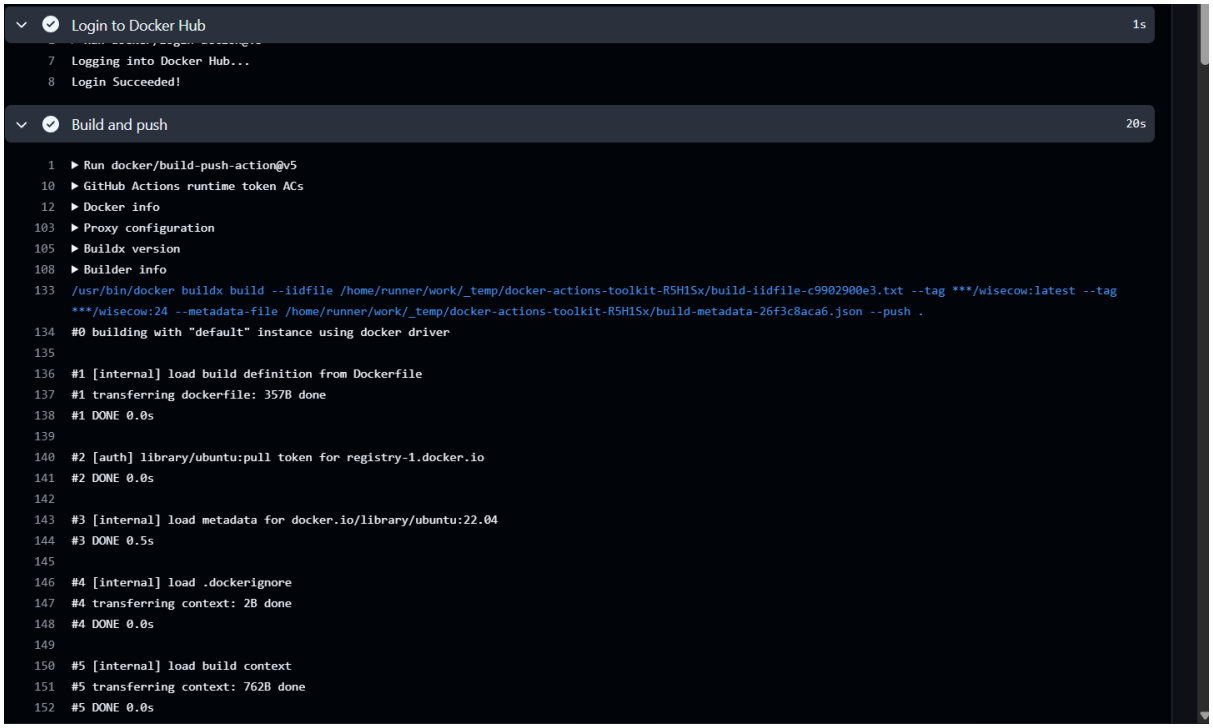
- As we can see I am just adding the hashes in the docker-build.yml file.
- When we commit and push into repo then the jobs starts.



- Next we can check in the git hub actions. The jobs will runs.

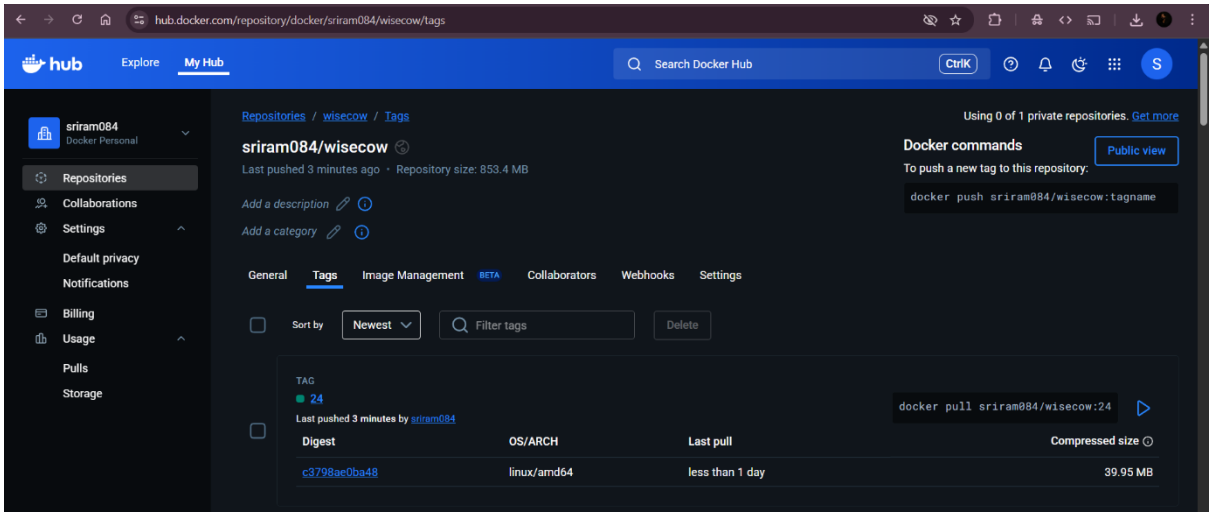


- Next it first job is docker-build.yml.

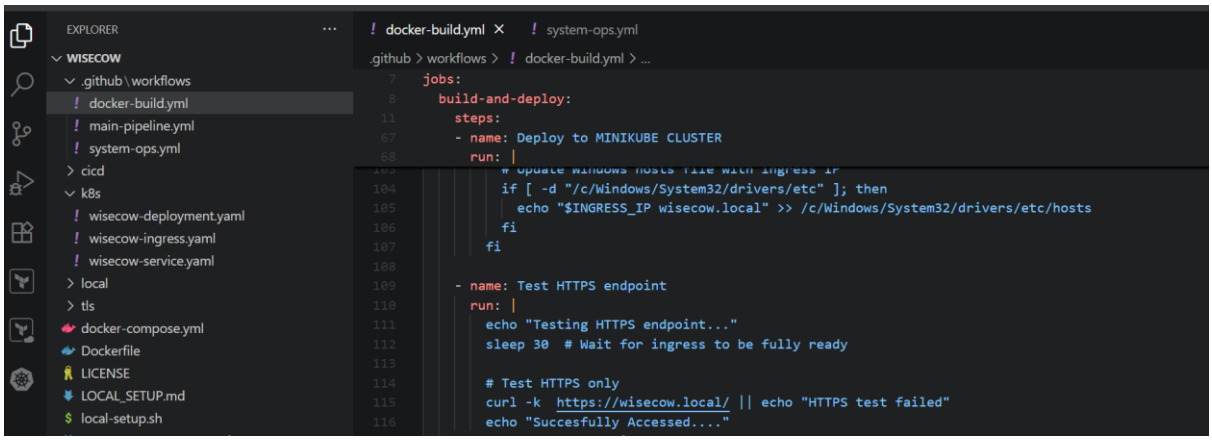


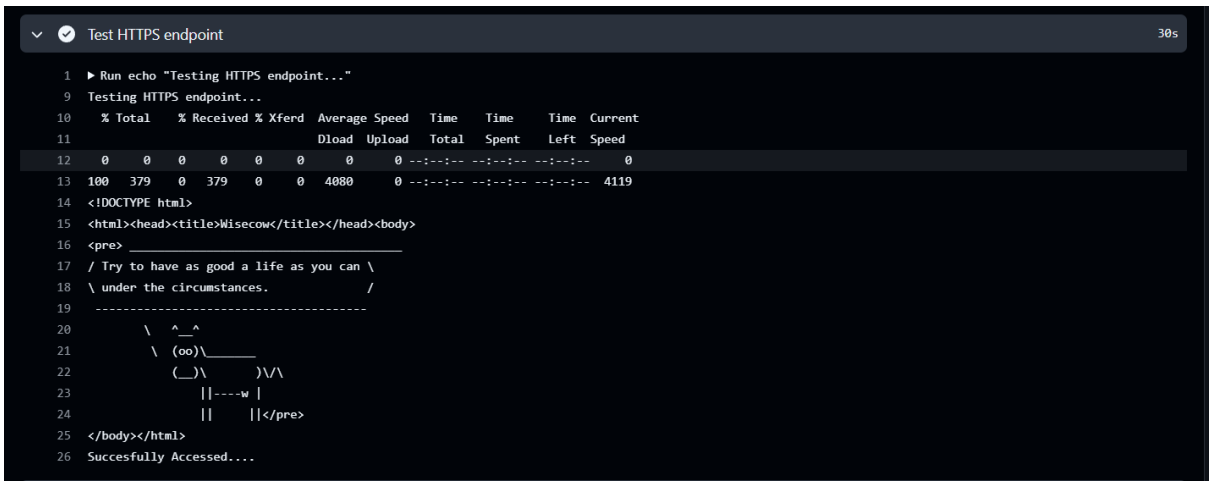
- As we can see in the dockerhub it creates an new image with build number. In mycase base image is sriram084/wisecow.

- It creates with sriram084/wisecow:24



- And next it continues, the job and next stages excutes at finally we see that the website can come up with the tls communication.





- After that the post clean process.

Problem-2:

1. System Health Monitoring Script:

- After that the docker-build.yml completion on successfully get into the another workflow known as the system-ops.yml

In that I can the check the CPU, memory usuage, dick space, and also running process. If it any of these

```
CPU_THRESHOLD=80
MEMORY_THRESHOLD=80
DISK_THRESHOLD=80
PROCESS_LIMIT = 200
```

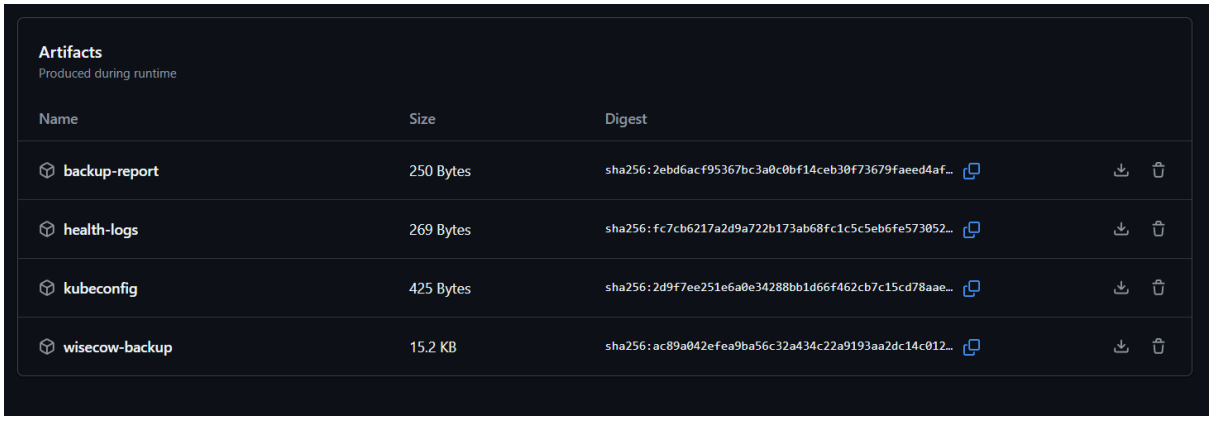
- After the successful build it store the in the health-logs.

2. Automated Backup Solution:

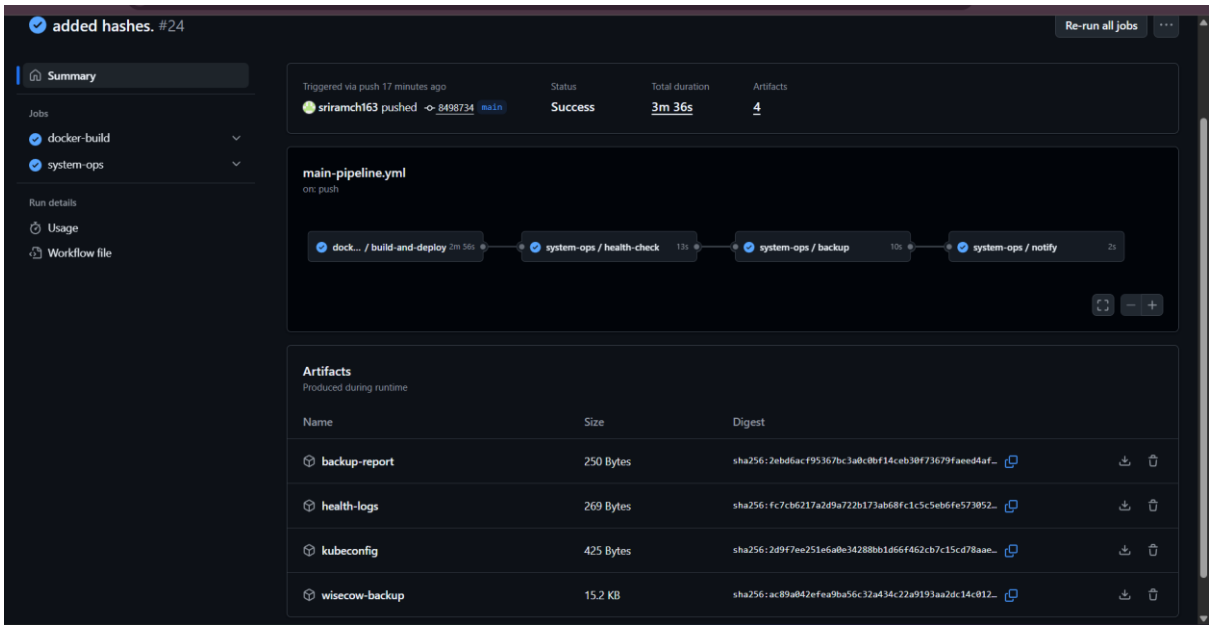
- After that health checks, it runs the job that give the backup-artifacts, with job name system-ops/backup

```
44
45      - name: Create backup directory
46        run: |
47          mkdir -p backup-artifacts
48          cp -r k8s/ backup-artifacts/
49          cp -r tls/ backup-artifacts/
50          cp -r cicd/ backup-artifacts/
51          cp -r local/ backup-artifacts/
52          cp kubeconfig.yaml backup-artifacts/ 2>/dev/null || true
53          cp *.md backup-artifacts/ 2>/dev/null || true
54          cp *.sh backup-artifacts/ 2>/dev/null || true
55          cp Dockerfile backup-artifacts/ 2>/dev/null || true
56
```

- After that health checks, it runs the job that give the backup-artifacts, with job name system-ops/backup.
- At last the health-logs,backup-articats, also kubeconfig file.

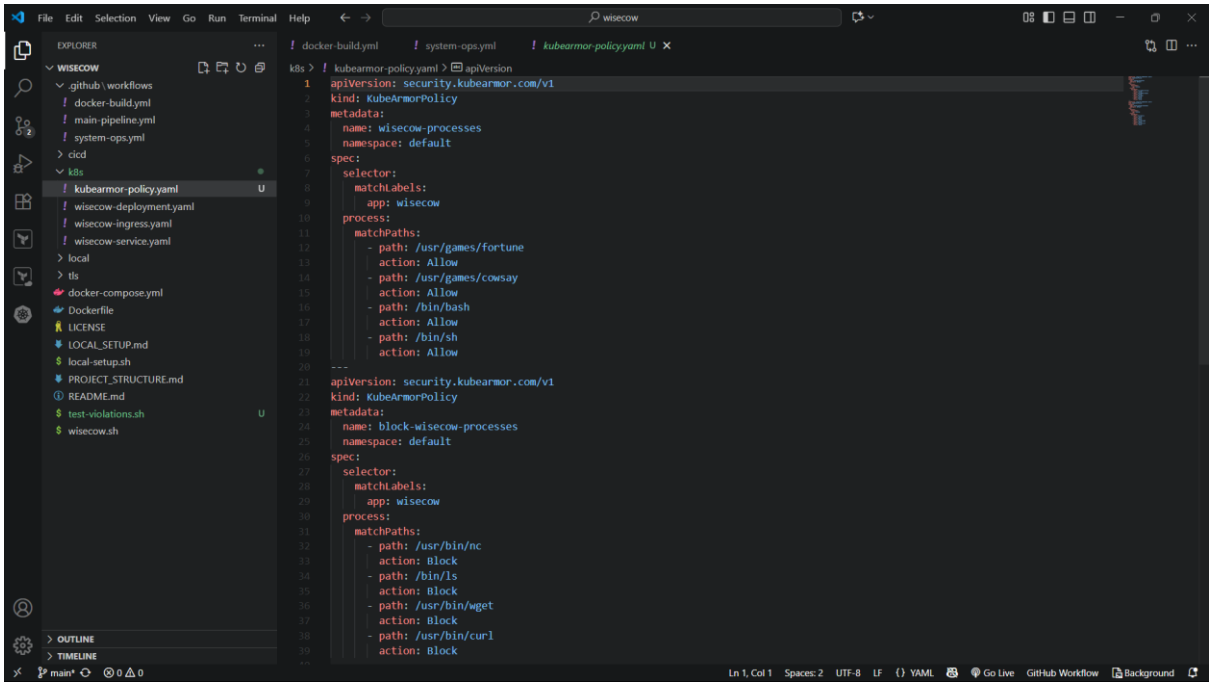


- And the complete workflow was respresented below.

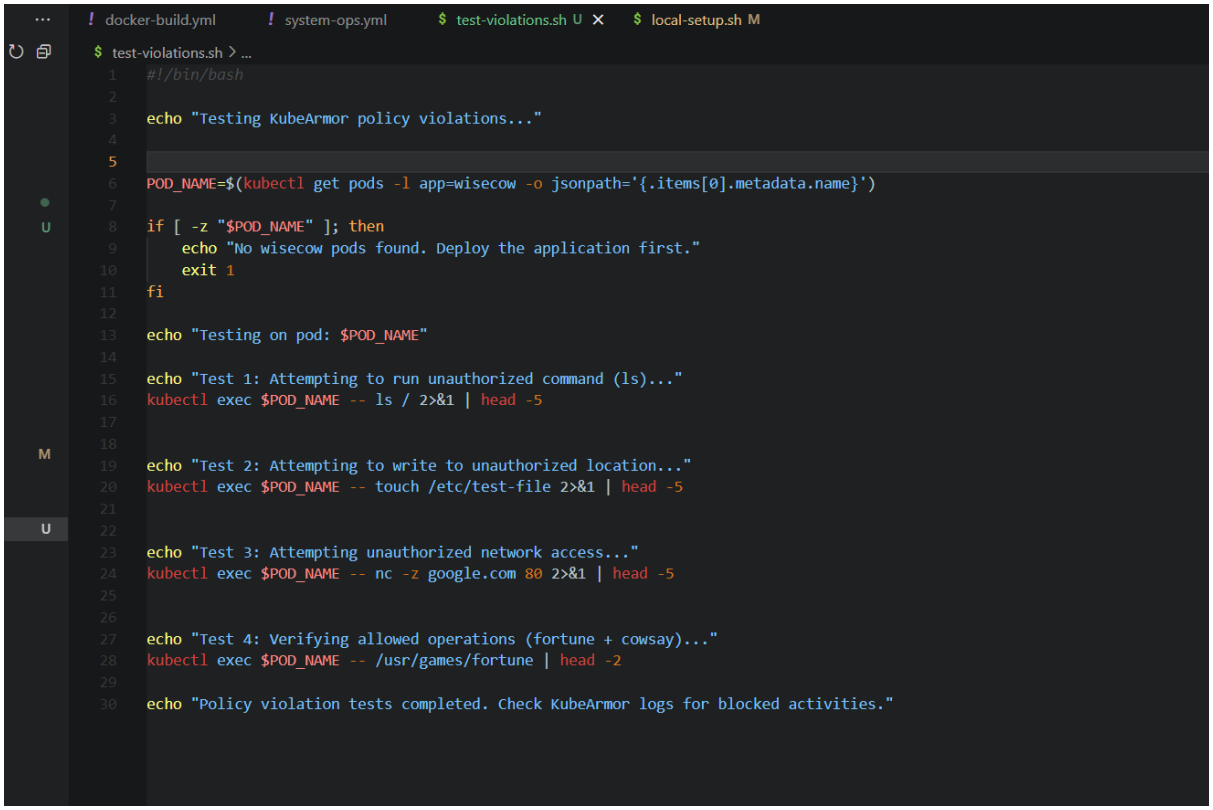


Problem-3:

- I had created the some what regarding the zero-trust-policy.
- But I failing , means when install and runs the k8s, as long as I am executing the policy it wont works.
- Later I tried on the online environments. Also regarding the website “killer code”. I just tried the nginx policy even though it wont works that policy. I believe that after checking pods, running, kubearmor pods,.. also it wont apply. After I searching the problem in the problem, I think these policy that wont run on my environments. So that I normally commiting the policy file.
- Kubearmor-policy file: https://github.com/sriramch163/wisecow/blob/main/kubearmor-policy.yaml



- For that I created an script for that to executing the more commands at a time that file also I am pushing into repo.
- Test-violations.file: https://github.com/sriramch163/wisecow/blob/main/test-violations.sh


docker-build.yml
system-ops.yml
test-violations.sh U
local-setup.sh M
test-violations.sh > ...
#!/bin/bash
echo "Testing KubeArmor policy violations..."
POD_NAME=$(kubectl get pods -l app=wisecow -o jsonpath='{.items[0].metadata.name}')
if [-z "$POD_NAME"]; then
echo "No wisecow pods found. Deploy the application first."
exit 1
fi
echo "Testing on pod: $POD_NAME"
echo "Test 1: Attempting to run unauthorized command (ls)..."
kubectl exec $POD_NAME -- ls / 2>&1 | head -5
echo "Test 2: Attempting to write to unauthorized location..."
kubectl exec $POD_NAME -- touch /etc/test-file 2>&1 | head -5
echo "Test 3: Attempting unauthorized network access..."
kubectl exec $POD_NAME -- nc -z google.com 80 2>&1 | head -5
echo "Test 4: Verifying allowed operations (fortune + cowsay)..."
kubectl exec $POD_NAME -- /usr/games/fortune | head -2
echo "Policy violation tests completed. Check KubeArmor logs for blocked activities."