Report by Mugle Sruthi

kubernetes-splunk-bruteforcedetection-project

Flask-based Secure Login System with Suspicious Activity Detection by using kubernetes and splunk , This project is a simple web-based user authentication system built using Python Flask. It simulates login, user creation, suspicious activity tracking, and logging.

➢ In the current digital landscape, **brute force attacks** remain one of the most prevalent and dangerous cybersecurity threats. This project, titled **"Kubernetes-Splunk-BruteForceDetection-Project"**, is designed to simulate brute force login attempts in a controlled **Kubernetes environment**. The goal is to understand and detect how attackers attempt unauthorized access through repeated login attempts and to analyze these suspicious activities in real time using **Splunk**.

This project includes a **Flask-based Secure Login System with Suspicious Activity Detection**, built using **Python Flask**. It offers functionalities like user creation, login simulation, and tracking of abnormal login behavior such as multiple failed attempts. All logs and events are captured and analyzed through Splunk to identify potential brute force activities effectively.

Table of content

INTRODUCTION:

➢ In the current digital landscape, **brute force attacks** remain one of the most prevalent and dangerous cybersecurity threats. This project, titled **"Kubernetes-Splunk-BruteForceDetection-Project"**, is designed to simulate brute force login attempts in a controlled **Kubernetes environment**. The goal is to understand and detect how attackers attempt unauthorized access through repeated login attempts and to analyze these suspicious activities in real time using **Splunk**.

This project includes a **Flask-based Secure Login System with Suspicious Activity Detection**, built using **Python Flask**. It offers functionalities like user creation, login simulation, and tracking of abnormal login behavior such as multiple failed attempts. All logs and events are captured and analyzed through Splunk to identify potential brute force activities effectively.

➢ **Using Docker for containerization, Kubernetes for orchestration, and Splunk for log analysis and visualization, this system deploys a Flask-based secure login simulation. It captures and monitors user authentication behavior, including login attempts, credential inputs, and suspicious activity patterns such as multiple failed logins. The logs generated from these interactions are forwarded to Splunk, where security teams can gain deep insights through custom dashboards and real-time alerts for brute force detection and response.**

- A **brute force attack** is a type of cyberattack in which an attacker attempts to gain unauthorized access to a system or account by systematically trying all possible combinations of passwords or encryption keys. This method relies on **repetitive, high-volume trial-and-error attempts**, rather than exploiting specific vulnerabilities or using sophisticated tactics. The attack continues until the correct credentials are found, making it a persistent and often automated threat to systems with weak or unprotected login mechanisms.
- This simulation helps in proactively training organizations to defend against **brute force attacks**, improving **detection techniques**, and enhancing overall **security posture**. The project demonstrates how modern **DevOps practices** and **SIEM tools** like **Splunk** can be seamlessly integrated to build an efficient, **scalable, and real-time security monitoring system** within a **Kubernetes** environment.

- **Docker** is an open-source platform that allows developers to automate the deployment of applications inside lightweight, portable containers. A Docker container includes everything an application needs to run—code, runtime, system tools, and libraries—making it easy to build, test, and deploy applications consistently across different environments. With Docker, developers can avoid the classic “it works on my machine” issue, as the containerized application behaves the same in development, testing, and production.
- **Kubernetes** (also known as K8s) is an open-source container orchestration platform developed by Google. It is used to manage, scale, and deploy Docker containers in a distributed environment. Kubernetes automates many tasks such as load balancing, service discovery, scaling up or down, rolling updates, and self-healing in case of container or node failures. It is especially powerful when dealing with microservices or applications that require high availability, as it helps ensure smooth deployment and consistent performance across multiple containers and servers.
- **Splunk** is a powerful data analytics and Security Information and Event Management (SIEM) tool that helps organizations search, monitor, and analyze machine-generated data in real time. In the context of cybersecurity, Splunk is widely used for detecting threats, analyzing logs, and visualizing suspicious behavior. When integrated with Docker and Kubernetes, Splunk can collect and index logs from containers and pods, providing security analysts with real-time dashboards and alerts to track activities such as phishing attempts, unauthorized access, or system anomalies.

1.1 Project Overview:

➢ This project aims to create a **brute force attack detection system** within a **Kubernetes environment** to identify and track **suspicious authentication behavior** using **Splunk**. By deploying a **Flask-based secure login simulation**, the system analyzes user login interactions, detects repeated unauthorized access attempts, and provides **detailed insights** via custom Splunk dashboards.

1.2 Problem Statement:

➢ Cyberattacks, particularly **brute force attacks**, have significantly increased, posing severe threats to organizations. Traditional detection mechanisms often lack **real-time visibility** and **scalability** in modern **containerized environments** like Kubernetes, making it harder to detect and respond to such threats effectively.

1.3 Solution:

➢ This project introduces a **Kubernetes-based brute force attack simulation system**, integrated with **Splunk** for real-time monitoring, logging, and alerting. By simulating login attempts and detecting anomalies, security teams can:

- Monitor and analyze **suspicious login behaviors**.
- Detect **repeated failed login attempts** that indicate brute force activity.
- Identify attack patterns and trends using detailed **Splunk dashboards and alerts**.

1.4 Technologies Used:

- **Kubernetes (AWS)** – For container orchestration and scalable deployment.
- **Splunk** – For **log analysis**, **dashboard visualization**, and real-time alerting.
- **Python (Flask)** – To develop a **secure login system** that simulates real-world authentication.
- **Docker** – For containerizing the Flask application for seamless deployment.

1.5 Expected Outcome:

- A fully functioning, containerized **secure login simulation system** deployed in **Kubernetes**.
- Real-time detection of **brute force attempts** and generation of alerts via **Splunk**.
- Integrated CI/CD pipelines for automated deployment and testing.

- **Interactive Splunk dashboards** displaying brute force attack patterns and suspicious activities.

1.6 Flow of Code (File Structure):

bruteforce-detection-app/

```
├── app.py
├── Dockerfile
├── .dockerignore
├── login_attempt.log          # Log file for recording login attempts
├── requirements.txt           # Dependencies for the Flask app
├── venvsruthi/                # Virtual environment folder (should be ignored in Docker builds)
├── static/
│   └── style.css              # CSS for UI styling
├── templates/
│   ├── create_user.html       # User creation page
│   ├── dashboard.html         # Dashboard to display login analytics or user info
│   └── index.html             # Login page
```

Project Name: bruteforce-detection-app/

This is the root directory of your Flask-based project designed to simulate and detect brute force login attempts in a Kubernetes environment with logging and monitoring handled by Splunk.

---

app.py

- This is the **main Flask application file**.
- Handles user routing, login logic, user creation, session management, and logs failed login attempts.
- Contains the code to write each login attempt into login_attempt.log.

Dockerfile

- Used to **containerize your Flask application**.
- Contains instructions to set up a lightweight image with Flask and your app.
- Helps deploy this app easily on Kubernetes.

.dockerignore

- Similar to .gitignore, but for Docker.
- Specifies files and folders Docker should ignore when building the image (like venvsruthi/, .git/, *.log, etc.).
- Helps make builds faster and cleaner.

---

login_attempt.log

- A log file where **each login attempt is recorded**, especially failed ones.
- Useful for simulating brute force detection by analyzing repeated failed login attempts.
- These logs can be **forwarded to Splunk** for real-time monitoring and visualization.

---

requirements.txt

- Lists all Python dependencies required by the app (e.g., Flask, Werkzeug, etc.).
- Used with pip install -r requirements.txt to quickly set up the environment.

---

venvsruthi/ (Virtual Environment)

- A local virtual environment folder containing all installed Python packages specific to this project.
- Should be **excluded from Docker builds** and version control (via .dockerignore and .gitignore).

---

static/

Contains static files like CSS, JavaScript, or images used in the frontend.

- **style.css**:
    - CSS file to style your HTML pages (login, dashboard, etc.).
    - Controls layout, colors, fonts, etc.

---

templates/

Contains all the **HTML templates** used by the Flask app (Flask looks here by default).

- **index.html**:
    - The **login page** where users enter credentials.
    - Forms the entry point for brute force detection simulation.
- **create_user.html**:
    - A page where new users can be created.
    - Simulates account creation within the app.
- **dashboard.html**:
    - A simple **admin/user dashboard**.
    - Could show recent login activity or display a success message after logging in.

2.Project code Explanation :

app.py

1.Imports and Setup

```
import os
from flask import Flask, render_template, request, redirect, url_for, flash, session
from datetime import datetime
from dotenv import load_dotenv  # type: ignore
import logging
```

explanation

→ **os**: Used for environment variables and system paths.

- **flask**: Imports modules for handling templates, form data, redirects, flashing messages, and sessions.
- **datetime**: For logging the current timestamp.
- **dotenv**: Used to load environment variables from a `.env` file.
- **logging**: Logs activities to console and a file for monitoring.

2.Environment & App Configuration

load_dotenv()

app = Flask(__name__)

app.secret_key = os.getenv("APP_SECRET_KEY", "secure_key")

explanation:

→ Loads environment variables (e.g., a secret key).

- Sets the Flask app's secret key for managing user sessions securely.

3.Data Setup

users = {'admin': 'admin123', 'user1': 'password123'}

SUSPICIOUS_USERNAMES = ['root', 'admin123', 'test', 'hacker']

failed_attempts = {}

SUSPICIOUS_THRESHOLD = 5

explanation:

→**users: Simulated in-memory user database.**

- SUSPICIOUS_USERNAMES: List of usernames considered suspicious.
- failed_attempts: Tracks failed login attempts for each username.
- SUSPICIOUS_THRESHOLD: Max allowed failed attempts before marking as suspicious.

4.Logging Configuration

LOG_FILE = "login_attempts.log"

logging.basicConfig(level=logging.INFO)

explanation:

→

Logs will be saved in a file login_attempts.log.

- Also logs messages to the terminal using logging.info().

5.Login Attempt Logging Function

```
def log_attempt(username, status):
    with open(LOG_FILE, 'a') as log:
        log.write(f"{datetime.now()} | Username: {username} | Status: {status}\n")
    logging.info(f"{datetime.now()} | Username: {username} | Status: {status}")
```

explanation:

→ **Records login attempts with timestamps.**

- Writes logs to both the log file and console

6.Home Page Route

```
@app.route('/')
def index():
    return render_template('index.html')
```

explanation:

→ Loads the login page (index.html).

7.Login Logic

```
@app.route('/login', methods=['POST'])
def login():
```

Get login form values:

```
username = request.form['username']
password = request.form['password']
```

Check suspicious usernames or threshold exceeded:

```
if username in SUSPICIOUS_USERNAMES or failed_attempts.get(username, 0)
>= SUSPICIOUS_THRESHOLD:
```

Logs attempt as suspicious and blocks login.

Correct credentials:

if username in users and users[username] == password:

Logs success, resets failed count, redirects to dashboard.

Invalid login:

failed_attempts[username] = failed_attempts.get(username, 0) + 1

Increments failed count.

If over threshold, logs suspicious.

Else, logs failed and flashes attempt count.

8.Dashboard Route

@app.route('/dashboard')
def dashboard():

explanation:

→ **Only accessible if logged in ('username' in session).**

- Displays dashboard with username.
- Else, redirects to login.

9.Create User Route

@app.route('/create_user', methods=['GET', 'POST'])
def create_user():

explanation:

→ **GET: Renders create_user.html.**

- **POST**: Handles user creation logic:
- Checks for existing username.
- Adds new user to dictionary if unique.

10.Logout Route

@app.route('/logout')

def logout():

explanation:

→

Logs the user out by removing 'username' from session.

- Redirects back to login.

11.App Runner

if __name__ == '__main__':

app.run(host='0.0.0.0', port=5000, debug=True)

explanation

→

Runs the app on all network interfaces, port 5000.

- `debug=True` enables auto-reloading and error messages for development.

12.Summary:

- Simulates login-based brute force detection.
- Logs attempts and blocks access based on suspicious usernames or repeated failures.
- Uses Flask, environment variables, sessions, and local logging.

Images of app.py and saved login attempts

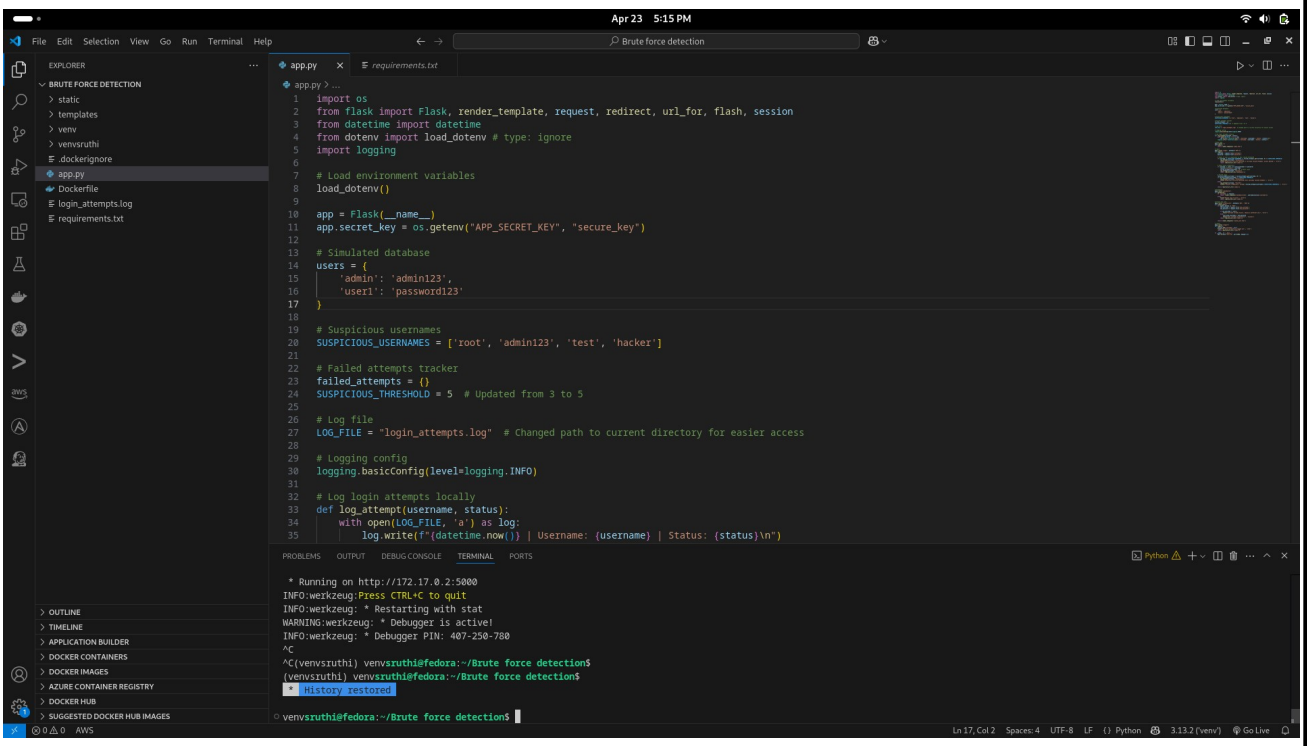


FIG : 1 app.py

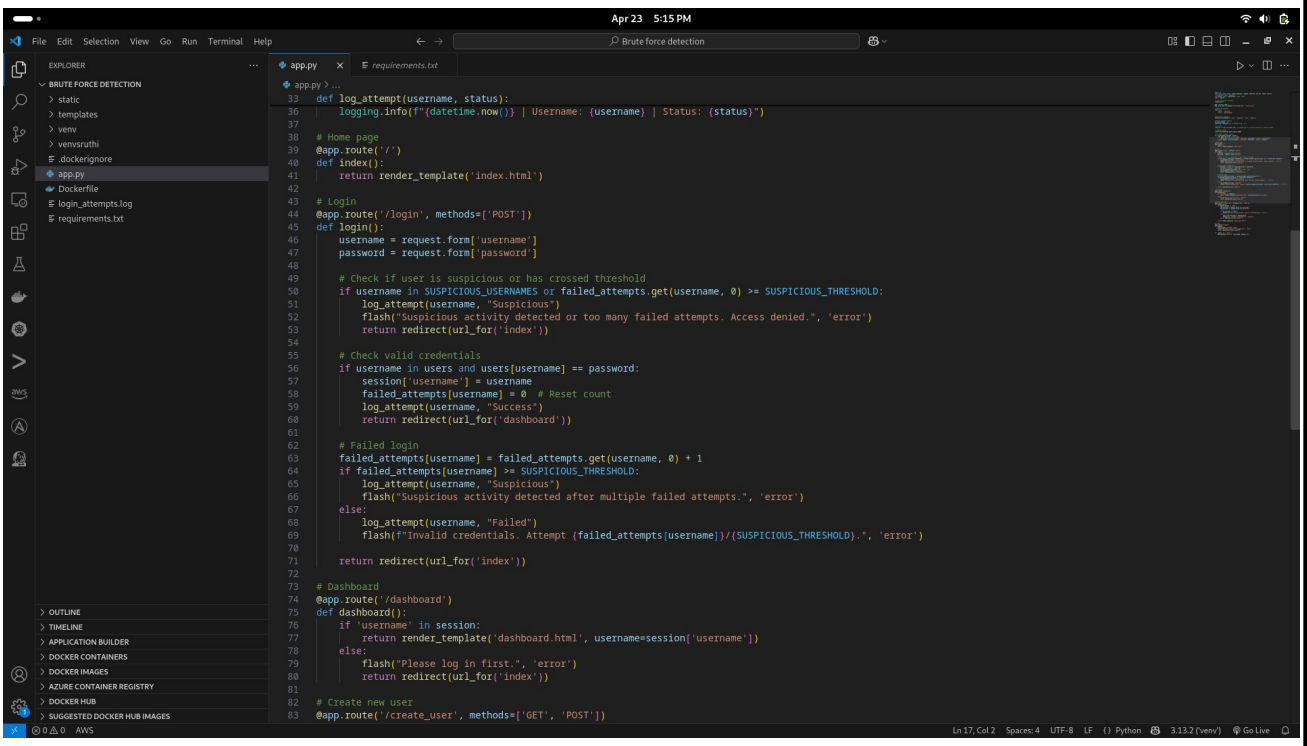


FIG : 2 app.py

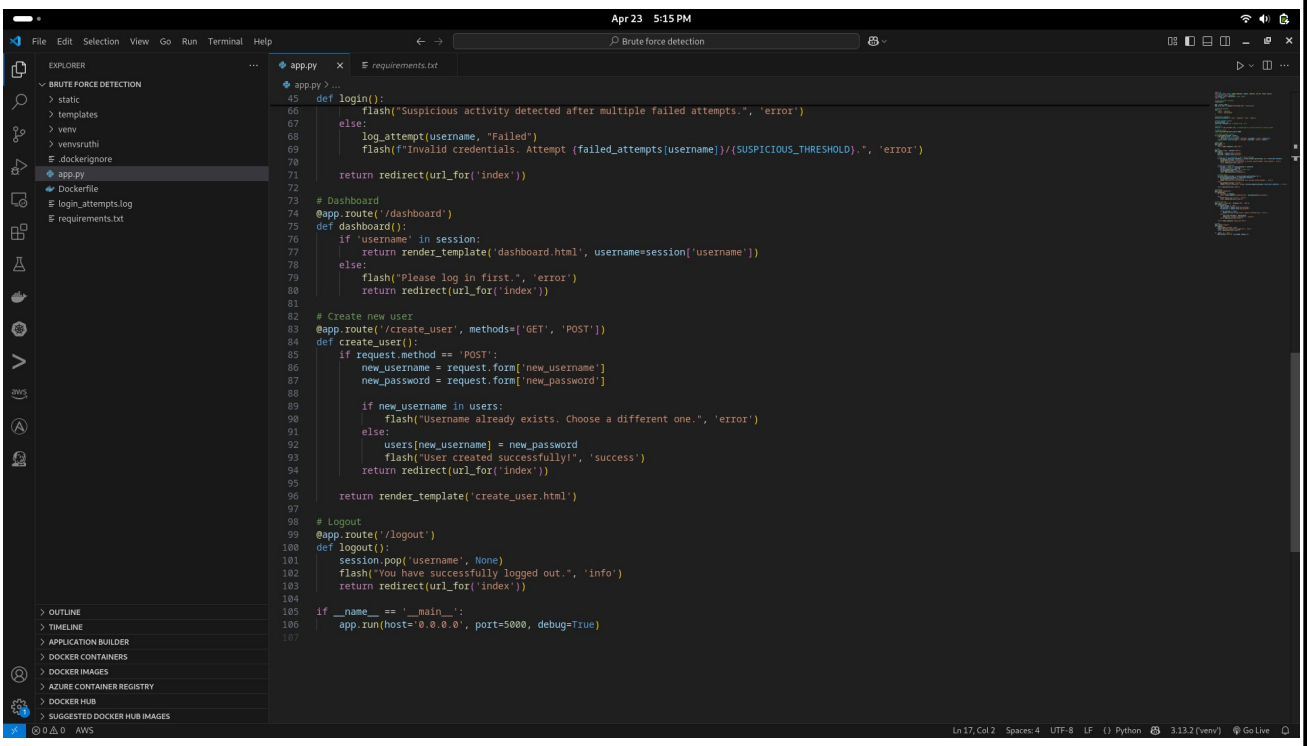


FIG : 3 app.py

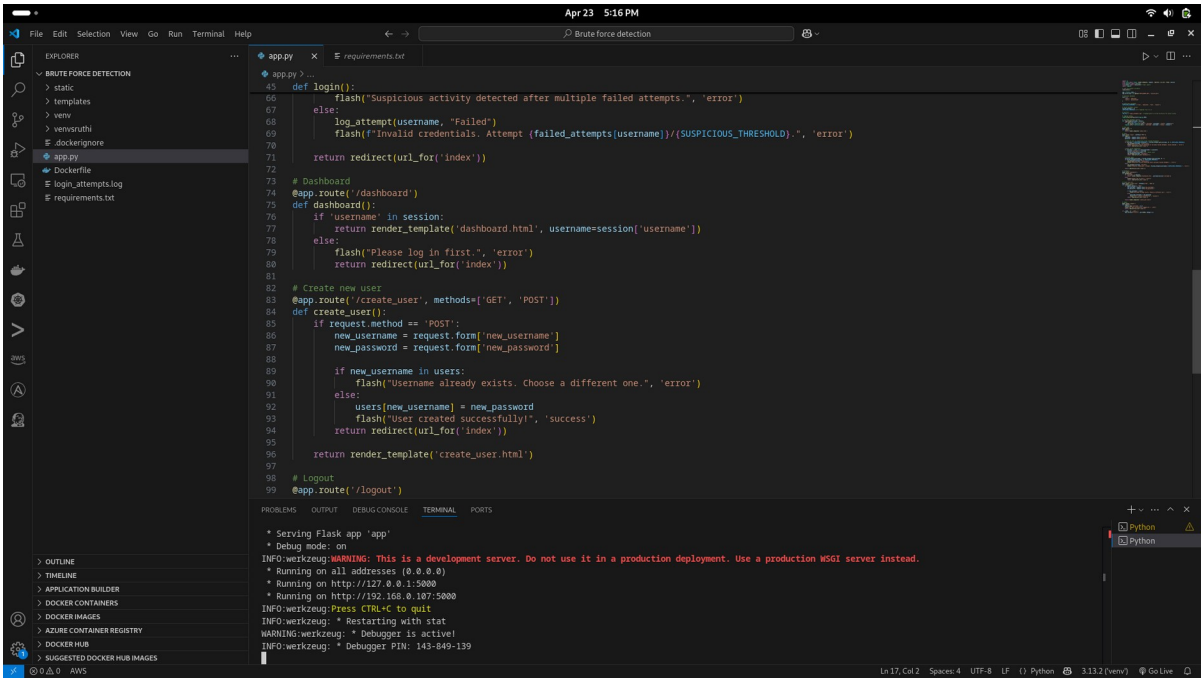


FIG : 4 running application by python app.py

Note : install all dependencies , pip install flask and dotenv

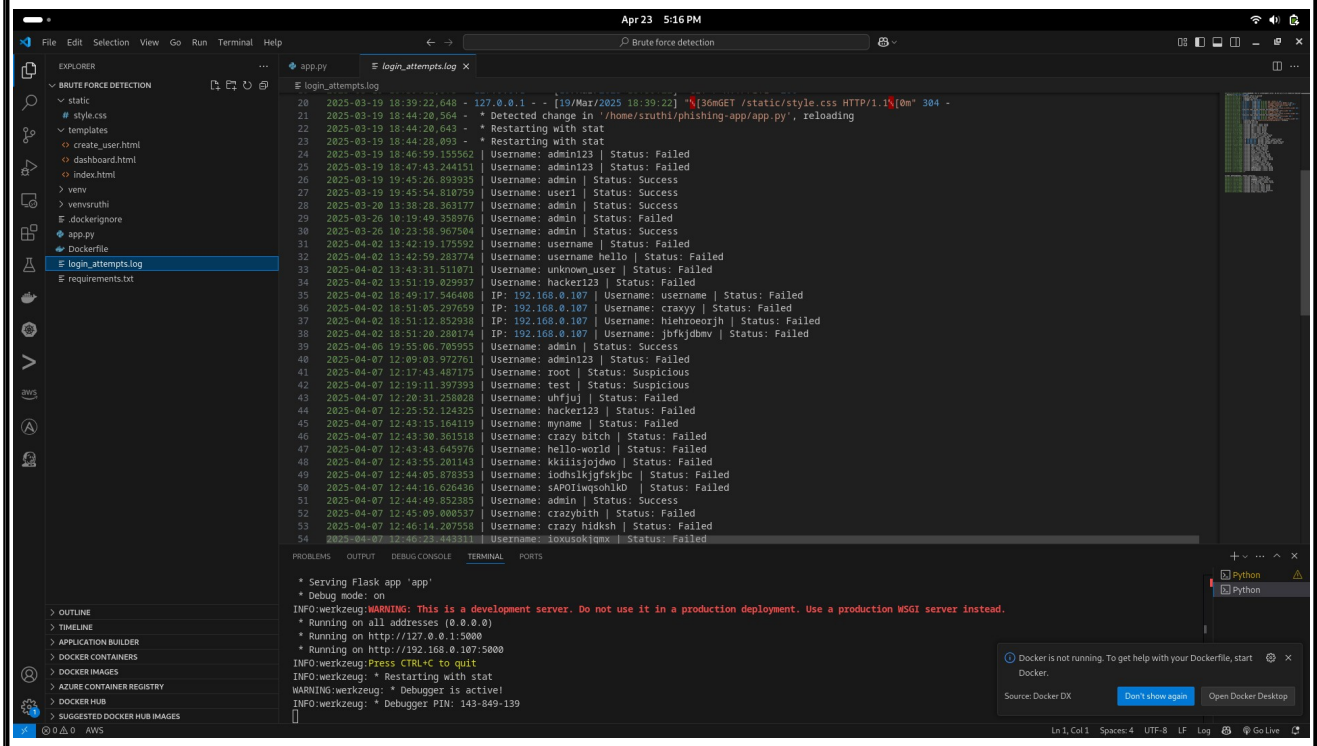


FIG : 5 Logs saved

2.1 Docker File

```
FROM python:3.13-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
# Create log directory
RUN mkdir -p /var/log/app
 # Set permissions if needed
RUN chmod -R 777 /var/log/app
CMD ["python", "app.py"]
```

Line-by-Line Explanation

→ FROM python:3.13-slim

- **Base Image**: Uses the official lightweight Python 3.13 image (slim version = fewer packages = faster build).
- It's clean, minimal, and perfect for production.

→ WORKDIR /app

- **Sets the working directory inside the container** to /app.
- All subsequent commands (e.g., copy, run) will operate from this directory.

→ COPY requirements.txt requirements.txt

- Copies your requirements.txt file from your local system to the container's /app directory.
- This file lists all your Python dependencies (like Flask, python-dotenv, etc.).

→ RUN pip install -r requirements.txt

- Installs all the Python dependencies listed in requirements.txt using pip.
- These are needed to run your Flask app.

→ COPY . .

- Copies **all the files and folders** from your local project directory to the container’s /app directory.
- This includes:
    - app.py
    - static/, template/
    - .env, login_attempts.log (if present), etc.

→ RUN mkdir -p /var/log/app

- Creates a directory in the container for logs (/var/log/app).
- While your log file currently writes to the project folder, this prepares a dedicated log directory (best practice for production).

→ RUN chmod -R 777 /var/log/app

- Gives **full read/write/execute permissions** to everyone on /var/log/app.
- Helpful for avoiding permission errors, especially in Docker environments.

→ CMD ["python", "app.py"]

- **Default command** to run when the container starts.
- It launches your Flask application.

→ What Happens When You Build and Run:

1. A Docker image is created with all your code and dependencies.
2. On docker run, the container starts the Flask app (app.py).
3. The app listens on port 5000 (from your Flask code).
4. Logs are stored and accessible inside the container.

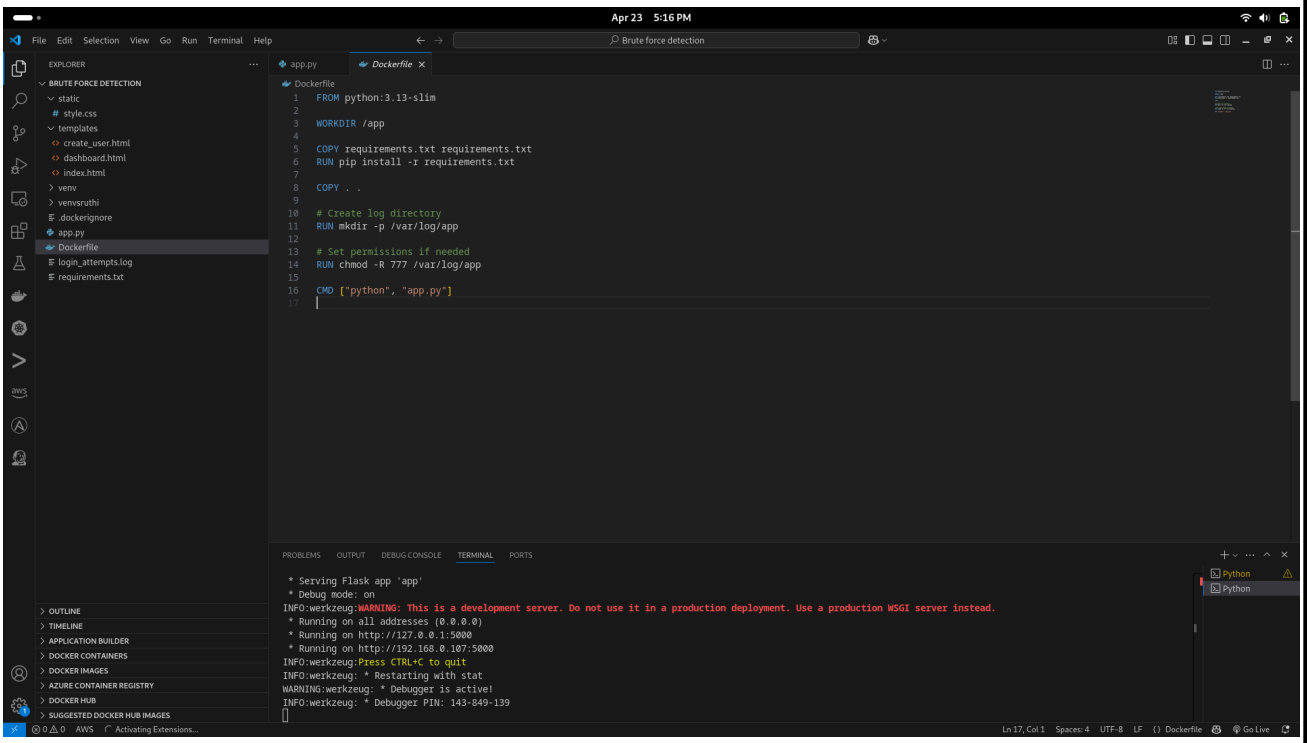


Fig : 6 Docker file

→ **Requirements.txt**

```
Flask
gunicorn
pip
requests
Flask-Limiter
dotenv
python-dotenv
```

3. Containerzing Using Docker

Building Docker image

→ docker build -t bruteforcedetection .

Running Docker container

→ sudo docker run -p 5000:5000 bruteforcedetection

Tagging Docker image

→ docker tag bruteforcedetection sruthimugle19/devops:latest

Pushing Docker image to docker hub

→ docker login

→ docker push sruthimugle19/devops:latest

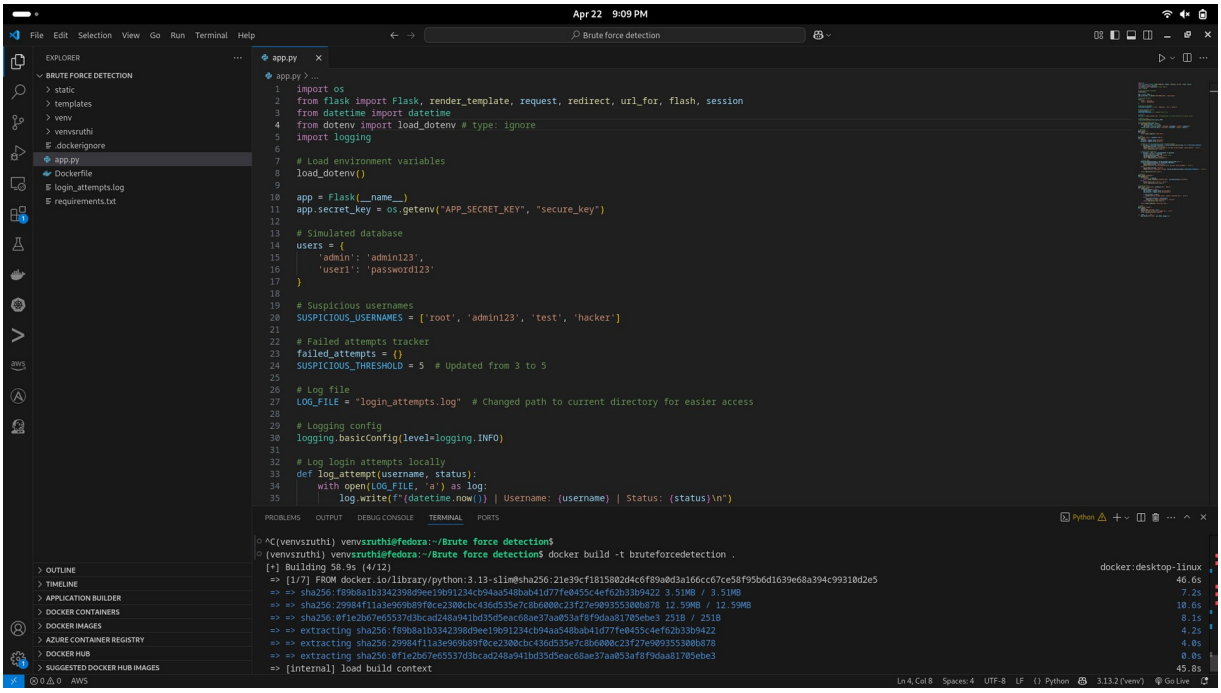


Fig : 7 Building Docker image

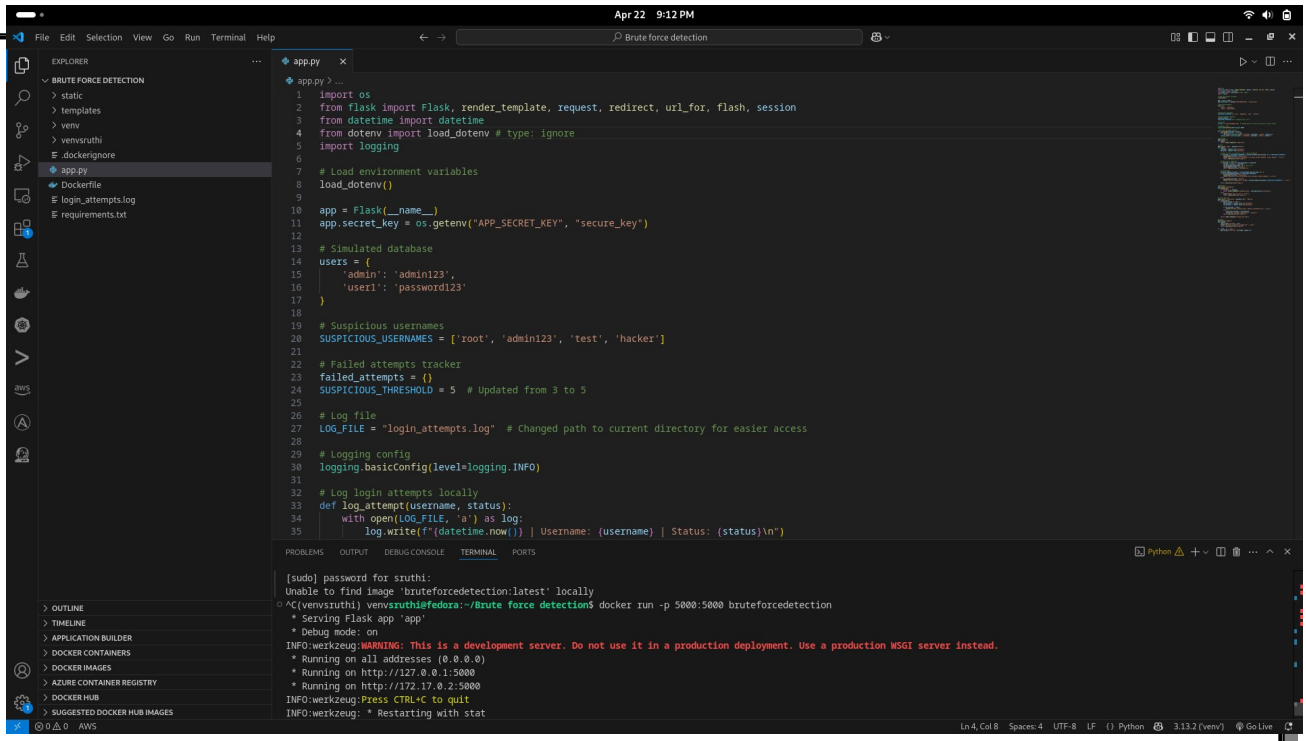


Fig : 8 Running docker container

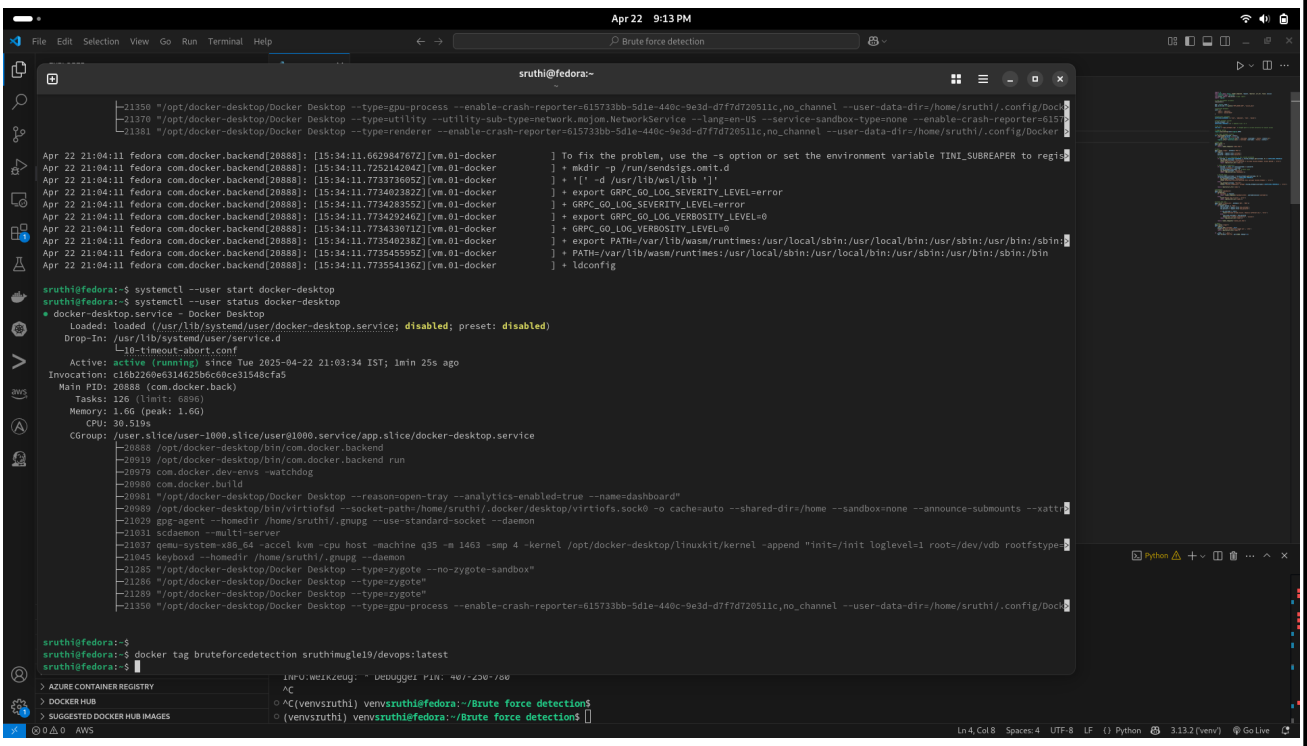


FIG : 9 tagging the image

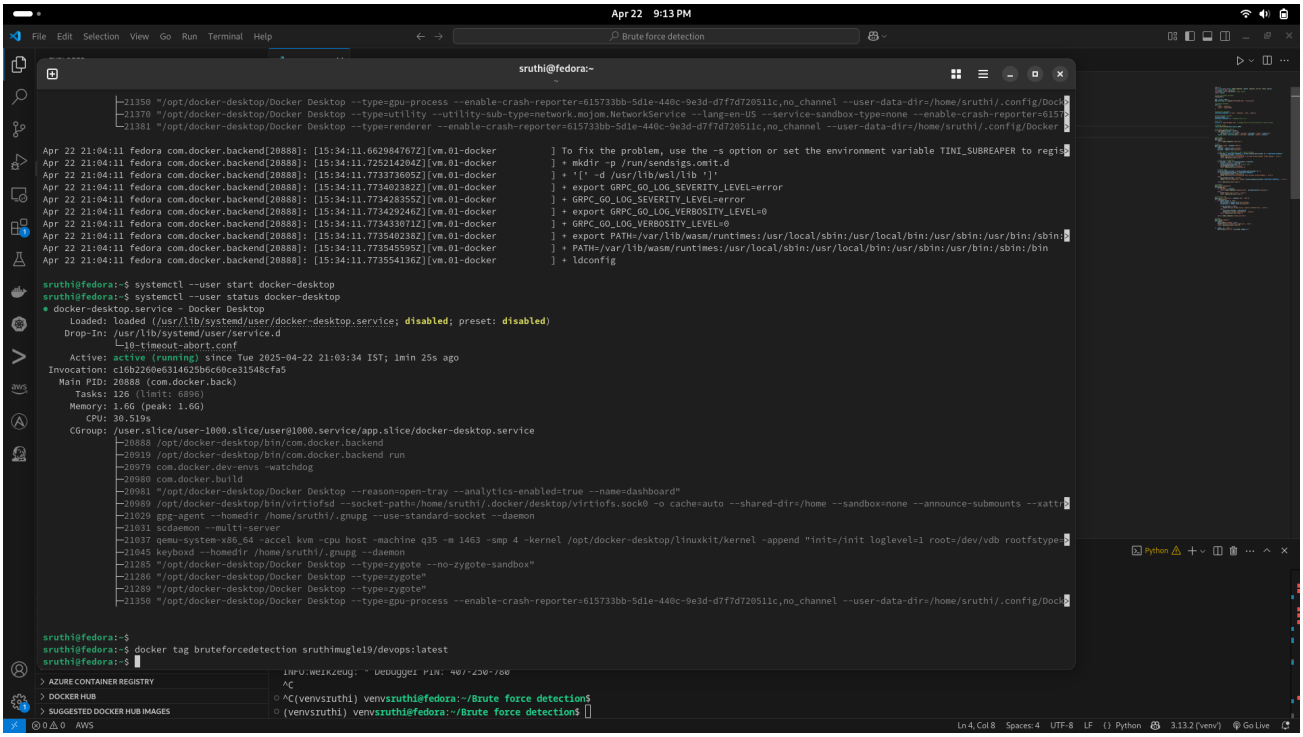


FIG : 10 pushing to docker hub

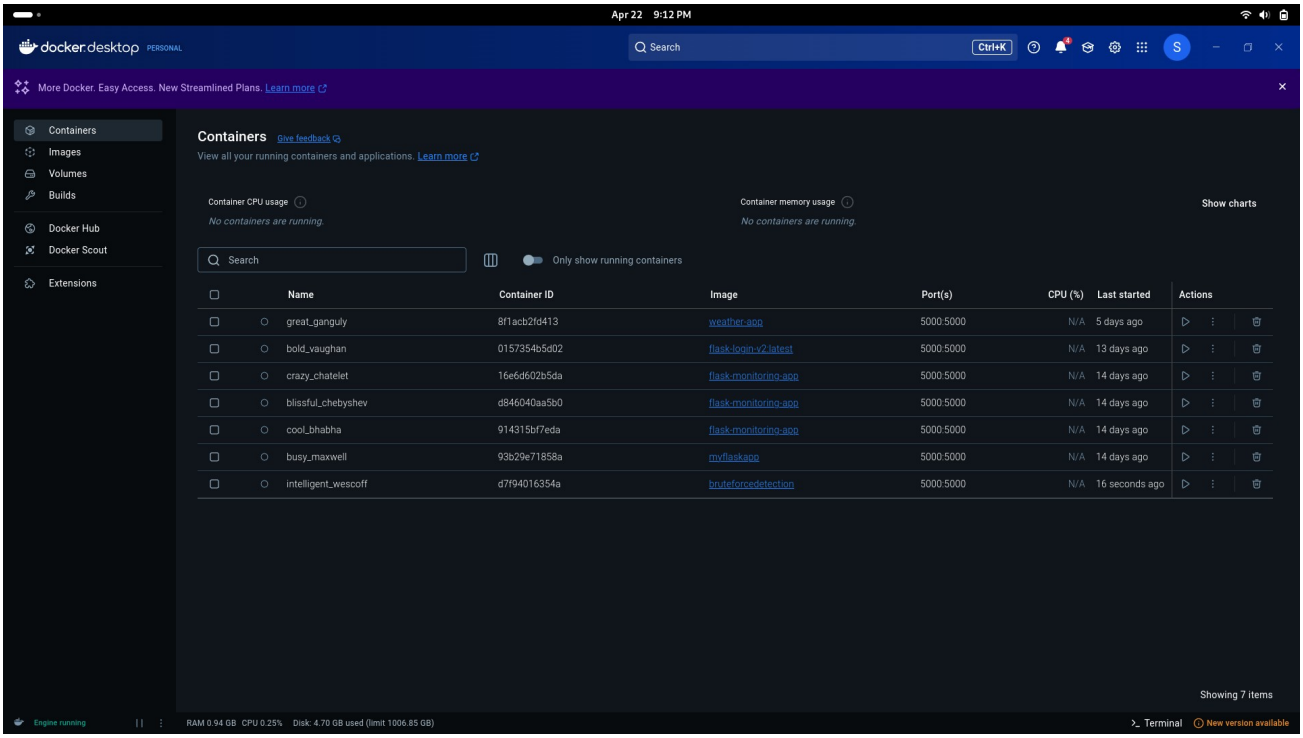


FIG : 11 image on docker hub

4. Kubernetes Installation and Splunk on AWS EC-2 instance
Now we have containerized our application starting with AWS EC-2 Instance

Create AWS account
#master installation on aws ec2
launch 1 instance name as master-k8's
before that donot forget to create security group
name security group as k8-sg
add inbound rules
NOTE :
we can have same security group for both master and slave

Inbound Rules

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	0.0.0.0/0	
Custom TCP	TCP	6443	0.0.0.0/0	
All traffic	All	All	0.0.0.0/0	
Custom TCP	TCP	10250	0.0.0.0/0	
Custom TCP	TCP	10259	0.0.0.0/0	
Custom UDP	UDP	8472	0.0.0.0/0	
Custom TCP	TCP	2379	0.0.0.0/0	
Custom TCP	TCP	10257	0.0.0.0/0	
Custom TCP	TCP	30080	0.0.0.0/0	

dont change the outbound rule

-->Now go to launch instance --> AMI machine should be ubuntu 22 or 24 version
-->instance type should be t3.medium or t2.medium
-->add created security group k8-sg
-->make sure to add keypair
NOTE: master and slave should have different key pair
add volume storage as 20Gb
and launch instance --> now click on connect
and enter the mentioned commands to install k8 on master node

---

#commands to install kubernetes master in ec2 instance aws

==installation of kubernetes master
--master commands

```
sudo su
sudo swapoff -a
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

sudo sysctl --system
lsmod | grep br_netfilter
lsmod | grep overlay
sudo apt-get update
sudo apt-get install -y ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo
\"$VERSION_CODENAME\") stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null

sudo apt-get update
sudo apt-get install -y containerd.io

containerd config default | sed -e 's/SystemdCgroup = false/SystemdCgroup = true/' -e
's/sandbox_image = "registry.k8s.io\/pause:3.6"/sandbox_image =
"registry.k8s.io\/pause:3.9"/' | sudo tee /etc/containerd/config.toml

sudo systemctl restart containerd
sudo systemctl status containerd
Install Kubernetes Components:

sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gpg
```

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
sudo kubeadm init
mkdir -p "$HOME"/.kube
sudo cp -i /etc/kubernetes/admin.conf "$HOME"/.kube/config
sudo chown "$(id -u)":"$(id -g)" "$HOME"/.kube/config
kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.26.0/manifests/
calico.yaml
kubeadm token create --print-join-command
```

#NOTE THAT IF You are getting error
than
if you are getting error then write this one in place of sudo kubeadm init
sudo kubeadm init --ignore-preflight-errors=Mem

****

okay now we are done with master
now installing slave on another instance

launch 1 instance name as slave-k8's
before that donot forget to create security group
name security group as k8-sg
add inbound rules
NOTE :
we can have same security group for both master and slave

Inbound Rules

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	0.0.0.0/0	
Custom TCP	TCP	6443	0.0.0.0/0	
All traffic	All	All	0.0.0.0/0	
Custom TCP	TCP	10250	0.0.0.0/0	
Custom TCP	TCP	10259	0.0.0.0/0	
Custom UDP	UDP	8472	0.0.0.0/0	
Custom TCP	TCP	2379	0.0.0.0/0	
Custom TCP	TCP	10257	0.0.0.0/0	

Custom TCP TCP 30080 0.0.0.0/0

dont change the outbound rule

-->Now go to launch instance --> AMI machine should be ubuntu 22 or 24 version
-->instance type should be t3.medium or t2.medium
-->add created security group k8-sg
-->make sure to add keypair
NOTE: master and slave should have different key pair
add volume storage as 20Gb
and launch instance --> now click on connect
and enter the mentioned commands to install k8 on worker node

---

#commands to install kubernetes slave in ec2 instance aws

```
sudo su
sudo swapoff -a
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

sudo sysctl --system
lsmod | grep br_netfilter
lsmod | grep overlay
sudo apt-get update
sudo apt-get install -y ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo
```

\"$VERSION_CODENAME\") stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update
sudo apt-get install -y containerd.io

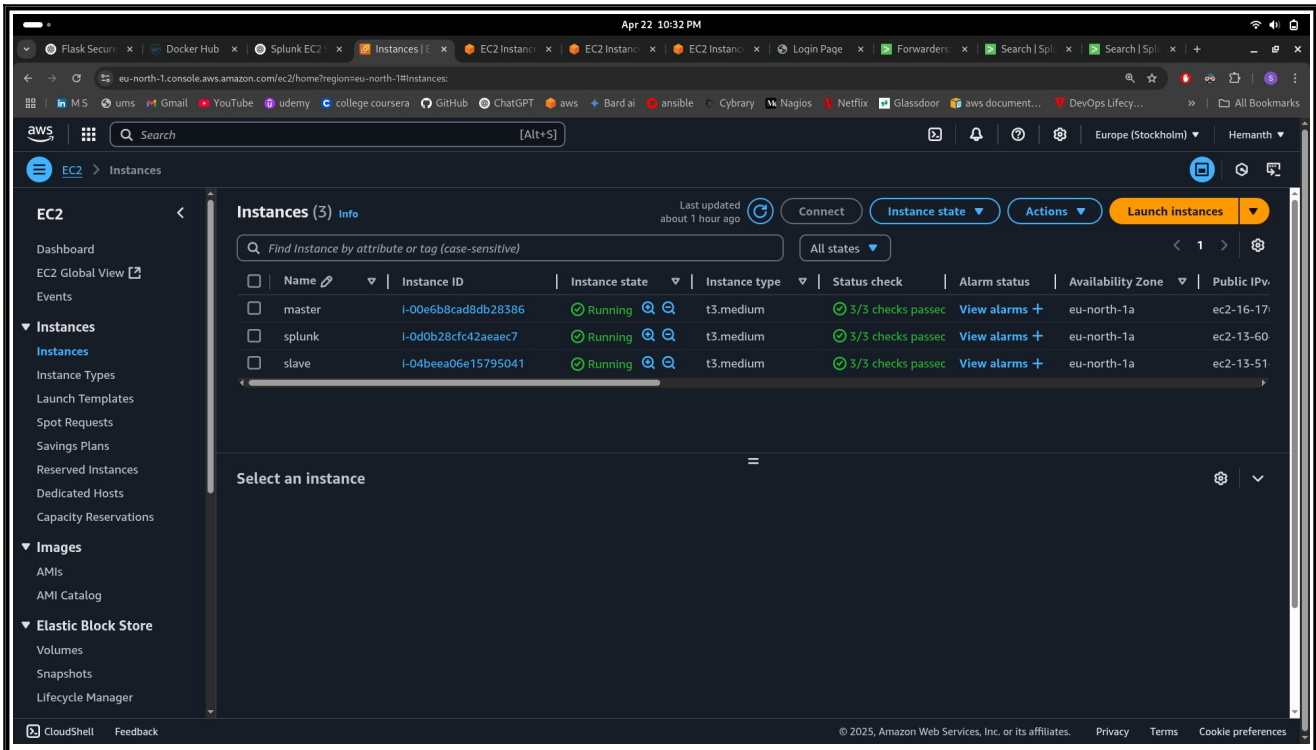
containerd config default | sed -e 's/SystemdCgroup = false/SystemdCgroup = true/' -e 's/sandbox_image = "registry.k8s.io\/pause:3.6"/sandbox_image = "registry.k8s.io\/pause:3.9"/' | sudo tee /etc/containerd/config.toml

sudo systemctl restart containerd
sudo systemctl status containerd

sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gpg
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
sudo kubeadm reset pre-flight checks

Now joining k8 master with k8 slave
you will get a key on master node copy that
go to slave node and
enter sudo <key which is copied from the master >
it will connect
go to master
type command as
kubectl get nodes

Now both master and slave nodes are ready



We are creating the k8's environment to setup manually for practice

Fig 12:instances setup

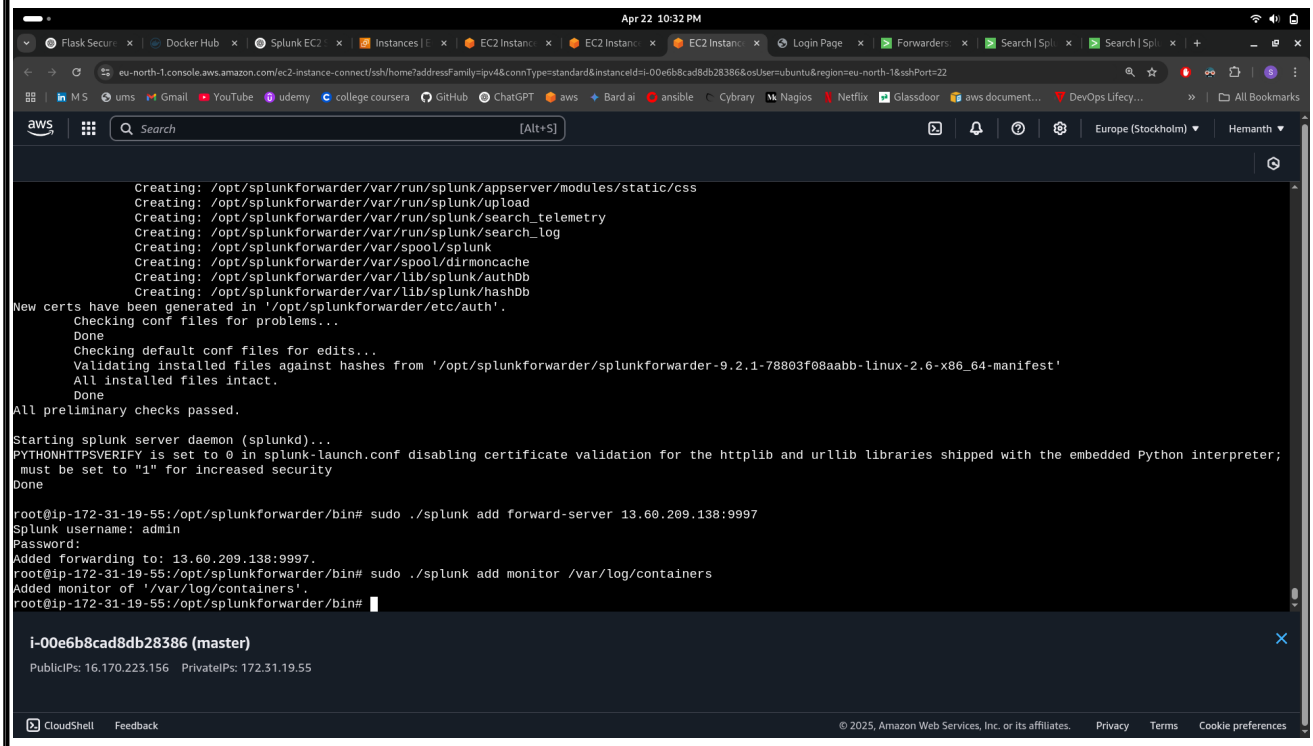


Fig 13 : master node

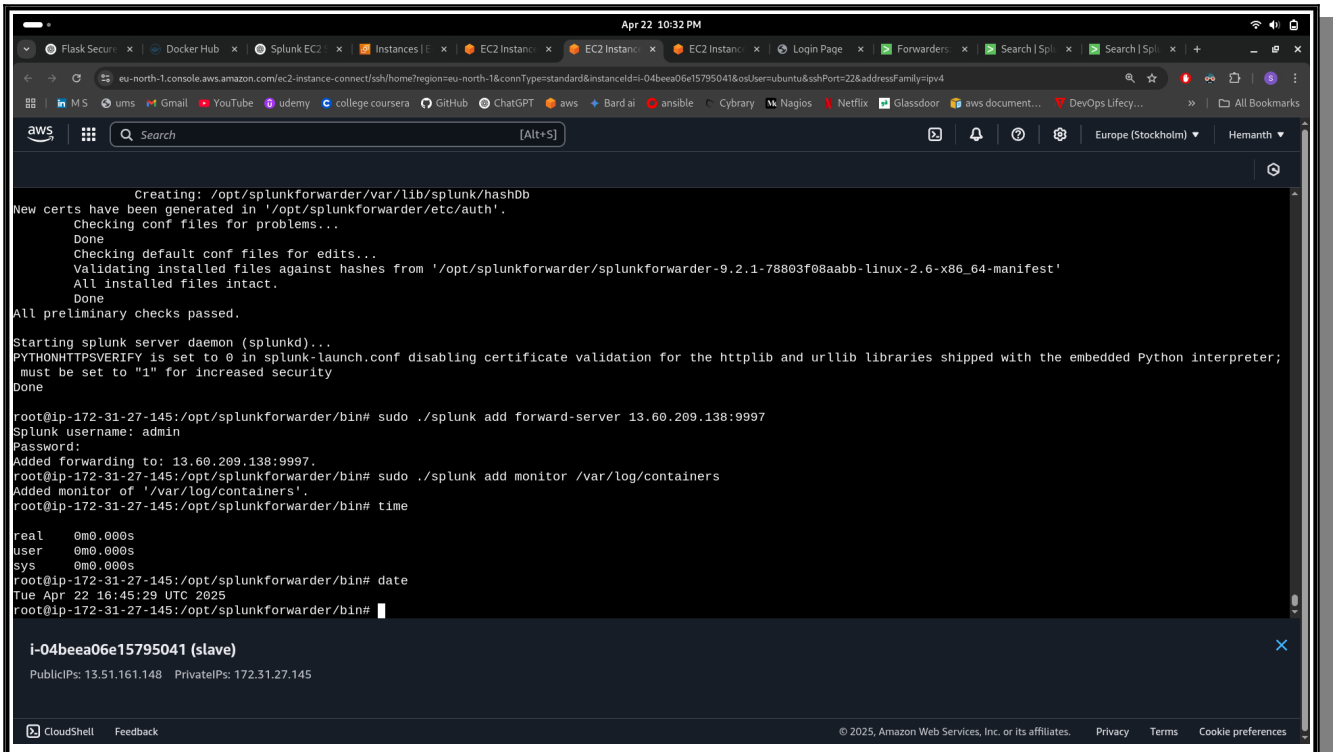


Fig 14 : slave node

4.2 Deploying Yaml file on kubernetes Master instance

nano bruteforce-detection.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bruteforce-detection
  labels:
    app: bruteforce
spec:
  replicas: 2
  selector:
    matchLabels:
      app: bruteforce
  template:
    metadata:
      labels:
        app: bruteforce
    spec:
      containers:
      - name: bruteforce-container
```

```
        image: sruthimugle19/devops:latest
        ports:
        - containerPort: 5000
        env:
        - name: APP_SECRET_KEY
          value: "your-secret-key"
        volumeMounts:
        - name: log-volume
          mountPath: /var/log/app
      volumes:
      - name: log-volume
        emptyDir: {}

---
apiVersion: v1
kind: Service
metadata:
  name: bruteforce-service
spec:
  selector:
    app: bruteforce
  type: LoadBalancer  # Use NodePort if not using cloud load balancer
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
```

1. Deployment Section

This tells Kubernetes **how to deploy and manage your Flask app containers**.

```
apiVersion: apps/v1
kind: Deployment
```

- You're creating a **Deployment** resource (stable object for managing replicated pods).

```
metadata:
  name: bruteforce-detection
  labels:
    app: bruteforce
```

- **Deployment Name**: bruteforce-detection
- Labels help Kubernetes track which pods belong to this app.

👥spec.replicas: 2

- Kubernetes will **run 2 replicas** (i.e., 2 identical pods of your app for high availability).

🔄 selector.matchLabels

```
selector:
  matchLabels:
    app: bruteforce
```

- This matches pods that have the label app: bruteforce, telling the Deployment which pods it owns.

🧱 Pod Template

This is the **actual pod configuration**:

```
template:
  metadata:
    labels:
      app: bruteforce
```

- Every pod created by this Deployment will have the label app: bruteforce.

🐳 Container Definition

```
containers:
- name: bruteforce-container
  image: sruthimugle19/devops:latest
```

- Runs your Docker image from Docker Hub (or other registry).
- sruthimugle19/devops:latest is your built image with Flask + phishing app.

🔌 Port Mapping

```
ports:
- containerPort: 5000
```

- Your Flask app runs on port **5000** inside the container.

🔐 Environment Variable

```
env:
```

```
- name: APP_SECRET_KEY
  value: "your-secret-key"
```

- This injects your secret key into the app (os.getenv("APP_SECRET_KEY") in Python).

📁 Volume for Logs

```
volumeMounts:
- name: log-volume
  mountPath: /var/log/app
```

- Mounts a volume at /var/log/app (useful for storing logs inside pods).
- Matches the path in your Dockerfile.

```
volumes:
- name: log-volume
  emptyDir: {}
```

- emptyDir creates **a temporary directory** that lives as long as the pod is running.
- Used for non-persistent logs or runtime files.

🌐 2. Service Section

This creates a **Service** to expose your app internally or externally.

```
apiVersion: v1
kind: Service
metadata:
  name: bruteforce-service
```

- Service name = bruteforce-service.

🔗 Service Specification

```
spec:
  selector:
    app: bruteforce
```

- The service targets pods **with label app: bruteforce** (your app pods).

🌍 Exposing Port

```
type: LoadBalancer
```

- Makes your app **publicly accessible** (works if you're using AWS, GCP, Azure, etc.).
- If you're testing locally (e.g., with Minikube), change it to:

  yaml
  CopyEdit
  type: NodePort

ports:
- protocol: TCP
  port: 80
  targetPort: 5000

- Exposes port **80** externally (standard HTTP), forwarding to **port 5000** inside the pod.

✅ Result:

- Kubernetes runs 2 pods of your Flask phishing detection app.
- Traffic to port 80 is forwarded to port 5000 inside the containers.
- Environment secrets and logs are properly configured.

Commands of deployment

nano bruteforce-detection.yaml

kubectl create -f bruteforce-detection.yaml

kubectl get pods

kubectl get deployment

kubectl get services

kubectl get pods --all-namespaces

kubectl cluster-info

#creating service

kubectl create service nodeport bruteforce-detection --tcp=80:80

kubectl get svc

1. nano bruteforce-detection.yaml

• This opens the file flask-deployment.yaml in the Nano text editor.

• You use it to write or edit your Kubernetes deployment and service configuration.

2. kubectl apply -f bruteforce-detection.yaml

• This command tells Kubernetes to apply the configuration written in flask-deployment.yaml.

• It will create or update the Deployment and Service described in the YAML file.

3. kubectl get pods

• Displays the current list of Pods running in your default namespace.

• Shows details like pod name, status (Running, Pending, etc.), and readiness.

4. kubectl get deployment

• Lists all Deployments in your current namespace.

• You'll see the deployment name, number of pods created, and their status.

5. kubectl get services

• Lists all the Services created in your namespace.

• Helps you verify that your Flask app is exposed via a Service (like NodePort or ClusterIP).

6. kubectl get pods --all-namespaces

• Lists all pods across all namespaces in your Kubernetes cluster.

• Useful when you're checking if system-level components (like DNS or monitoring agents) are running

properly.

7. kubectl cluster-info

• Displays information about the Kubernetes master and services, like the API server and DNS.

• Helps confirm that your cluster is up and running.8. kubectl get svc

• This is a shorthand for kubectl get services.

• It shows the list of services, their types (e.g., ClusterIP, NodePort), and the external port mapping if

any.

8. kubectl get svc

• This is a shorthand for kubectl get services.

• It shows the list of services, their types (e.g., ClusterIP, NodePort), and the external port mapping if

any.

Now after that access on Public ip of ec2 master instance and specific port http://ec-2-public-ip:<portnumber>

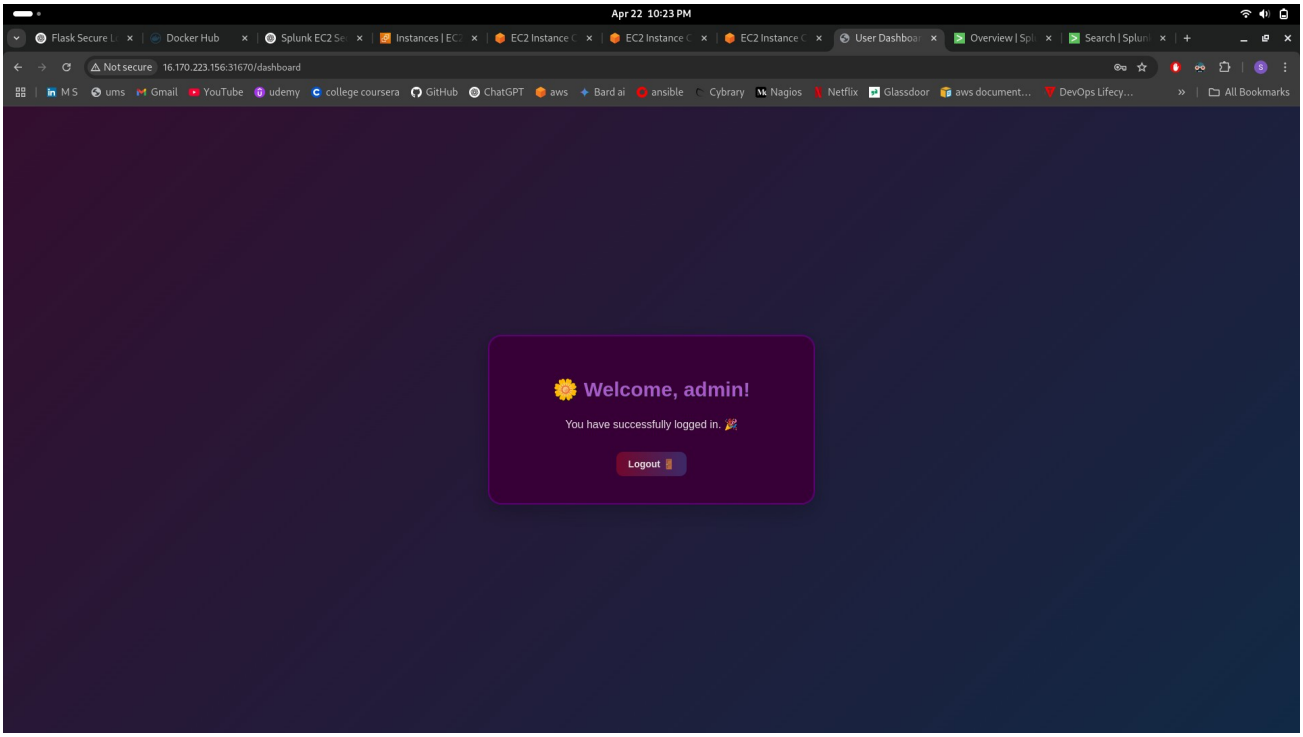


fig : 15 accessed via kubernetes deployment master ip and specific port Successfully logged in

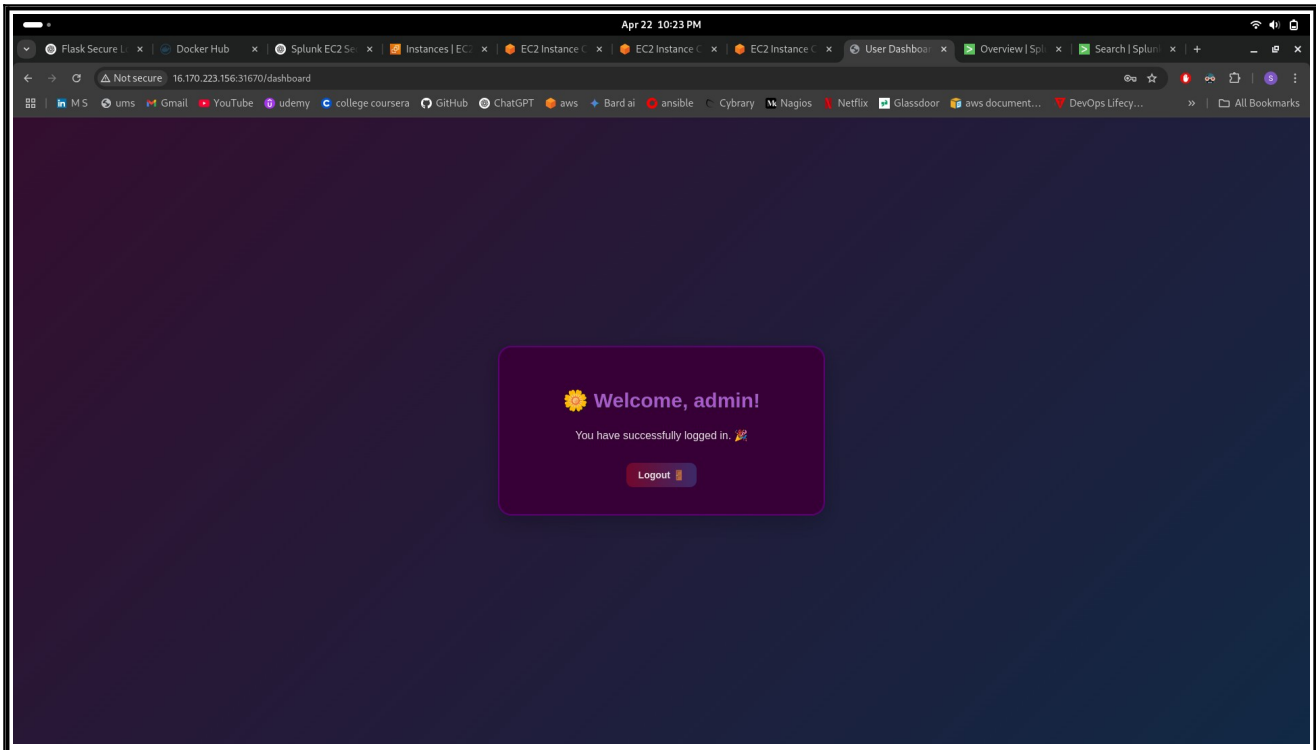


fig : 16 invalid attempt

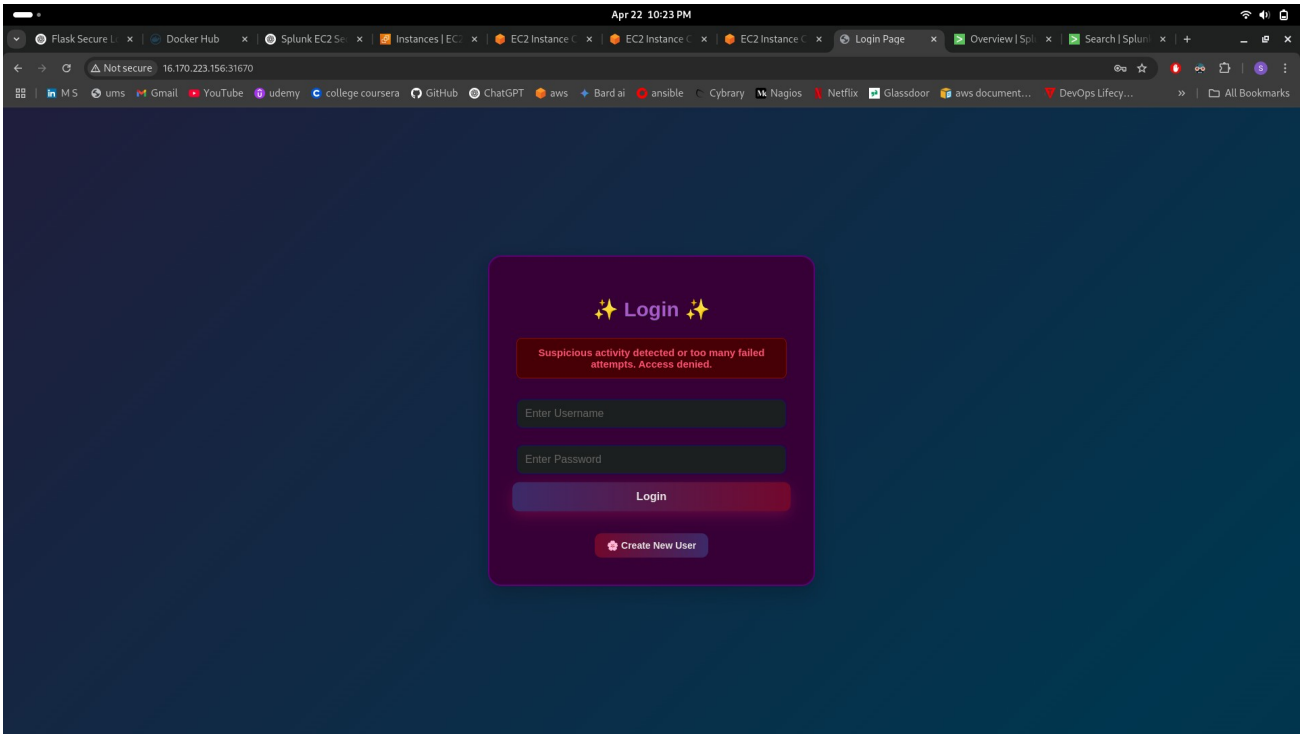


fig : 17 suspicious login attempt

4.3 Stored logs are in :

kubectl get pods

kubectl exec -it bruteforce-detection-5ddcdf7447-hkpwq -- /bin/bash

root@ip-172-31-17-235:/home/ubuntu# kubectl get pods

NAME

READY STATUS RESTARTS AGE

flask-app-5ddcdf7447-hkpwq 1/1 Running 0

17m

root@ip-172-31-17-235:/home/ubuntu# kubectl exec -it bruteforce-detection-5ddcdf7447-hkpwq -- /bin/bash

root@flask-app-5ddcdf7447-hkpwq:/app# ls -l /var/log/app

cat /var/log/app/login_attempts.log

total 4

-rw-r--r-- 1 root root 330 Apr 9 12:24 login_attempts.log

2025-04-09 12:18:54.258496 | Username: admin | Status: Success

2025-04-09 12:19:28.882961 | Username: test | Status: Suspicious

2025-04-09 12:21:47.193320 | Username: LKJBSACIYYUHKF | Status: Failed

2025-04-09 12:23:12.802402 | Username: d;ojihpofj;l | Status: Failed

2025-04-09 12:24:15.819094 | Username: admin | Status: Failed

root@flask-app-5ddcdf7447-hkpwq:/app#

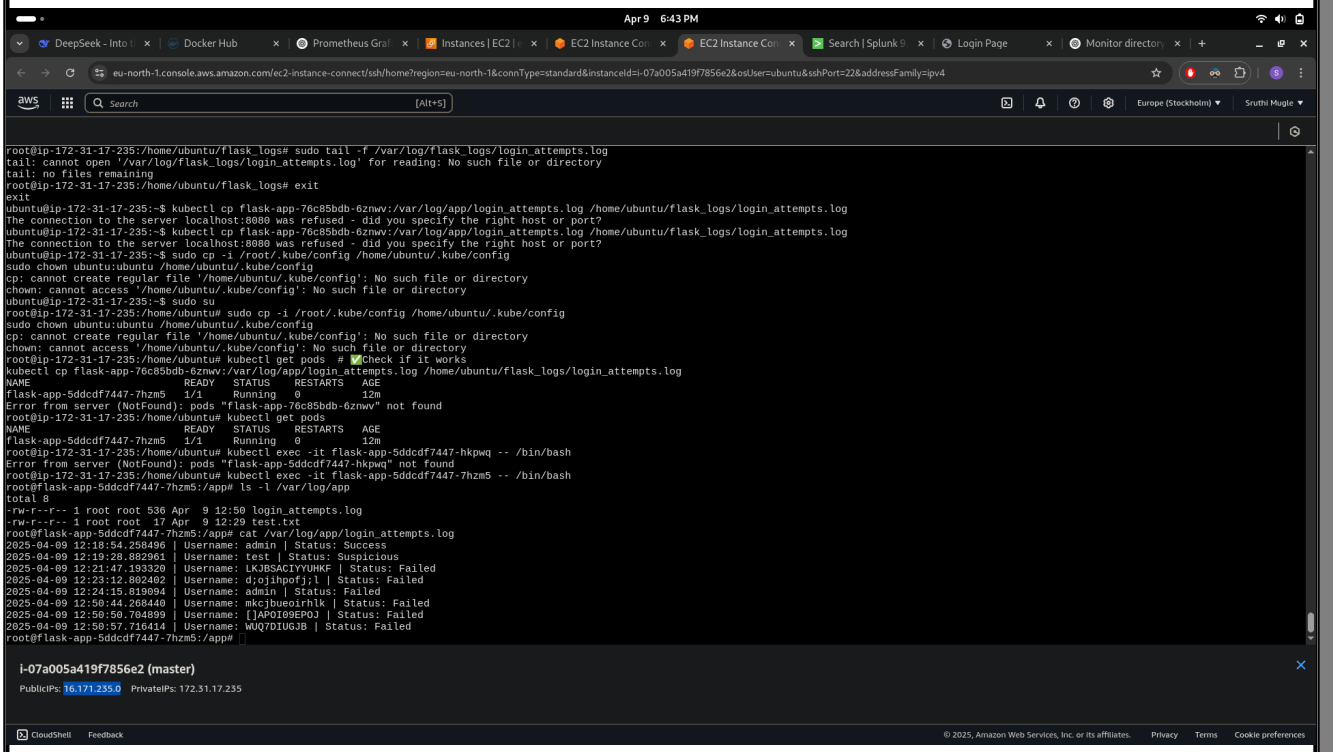


Fig 18 : logs stored

4.4 Install splunk in another instance

commands to install

 # 1. Switch to root (optional)
sudo su

 # 2. Go to /opt directory
cd /opt

 # 3. Update system packages
apt update

 # 4. Download Splunk Enterprise (latest 9.2.1 version)
wget -O splunk-9.2.1-78803f08aabb-Linux-x86_64.tgz "https://download.splunk.com/products/splunk/releases/9.2.1/linux/splunk-9.2.1-78803f08aabb-Linux-x86_64.tgz"

 # 5. Extract the tar file
tar -xvf splunk-9.2.1-78803f08aabb-Linux-x86_64.tgz

 # 6. Give permission to Splunk folder
chmod -R 777 splunk

 # 7. Navigate to Splunk bin folder
cd splunk/bin

 # 8. Start Splunk and accept license
./splunk start –accept-license

open splunk browser go to Go to: Settings > Forwarding and Receiving > Configure Receiving

Add new port: 9997

→
Installing universal forwarder in both master and slave for analyzing the logs

In both master and slave of kubernetes

 # 1. Download the Universal Forwarder

wget -O splunkforwarder-9.2.1-78803f08aabb-Linux-x86_64.tgz "https://download.splunk.com/products/universalforwarder/releases/9.2.1/linux/splunkforwarder-9.2.1-78803f08aabb-Linux-x86_64.tgz"

 # 2. Extract it to /opt
sudo tar -xvzf splunkforwarder-9.2.1-78803f08aabb-Linux-x86_64.tgz -C /opt

 # 3. Navigate to its bin directory
cd /opt/splunkforwarder/bin

 # 4. Start Splunk forwarder and accept license
sudo ./splunk start --accept-license

 # 5. Enable it to start at boot
sudo ./splunk enable boot-start

 # 6. Connect to Splunk Enterprise (Replace with actual Splunk EC2 Public IP)
sudo ./splunk add forward-server <SPLUNK_ENTERPRISE_PUBLIC_IP>:9997

 # 7. Start monitoring Kubernetes logs (container logs path)
sudo ./splunk add monitor /var/log/containers

After Setup:
Go to Splunk UI → Settings → Monitoring Console → Forwarders

You should now see both Master and Slave listed as connected forwarders.

Now we have setup start accessing the splunk you will see real time logs

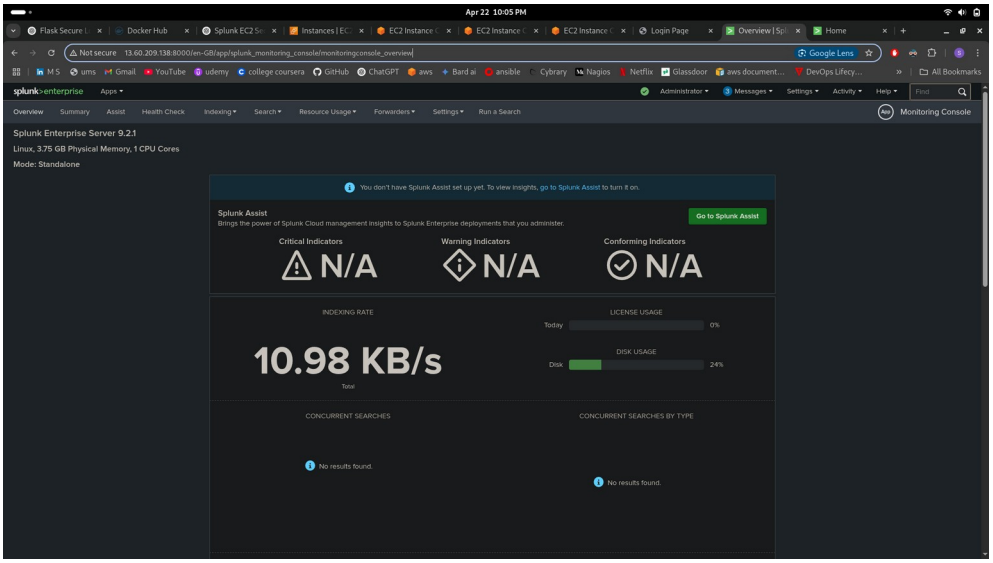


fig : 19 monitoring console

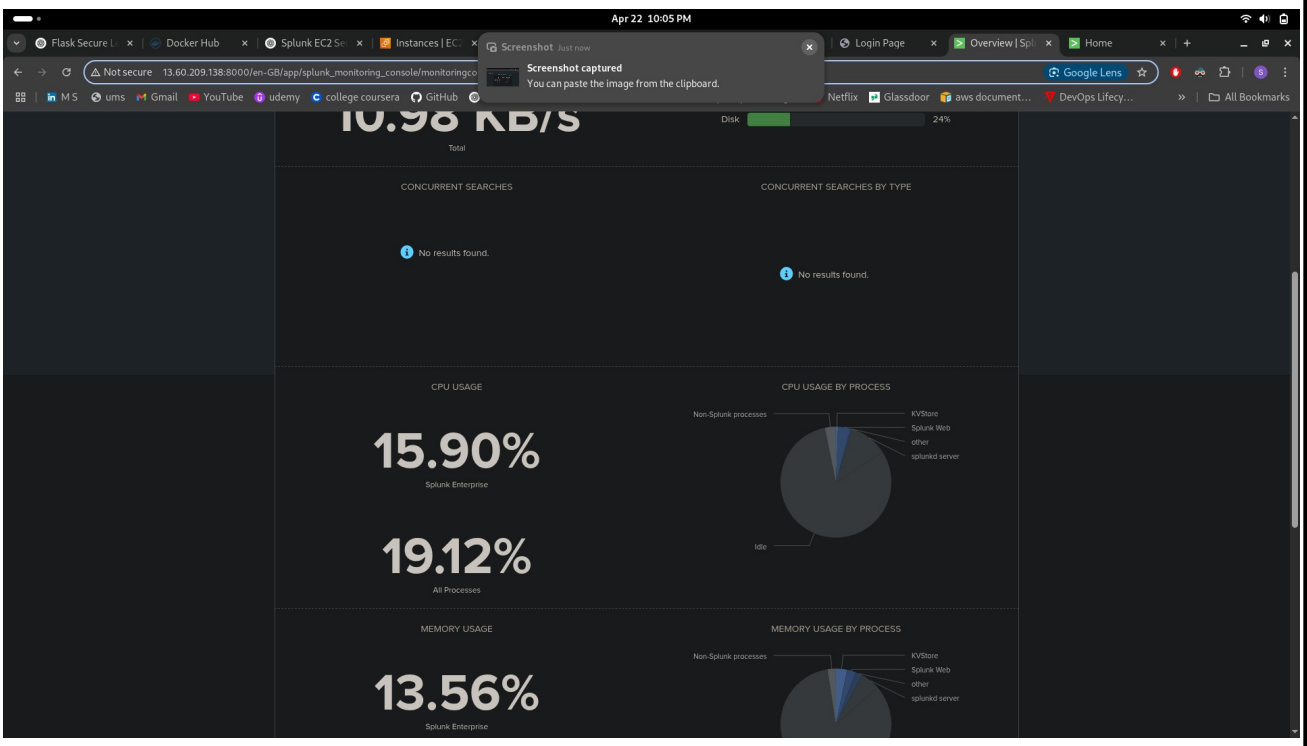


fig : 20 we can see our both master and slave

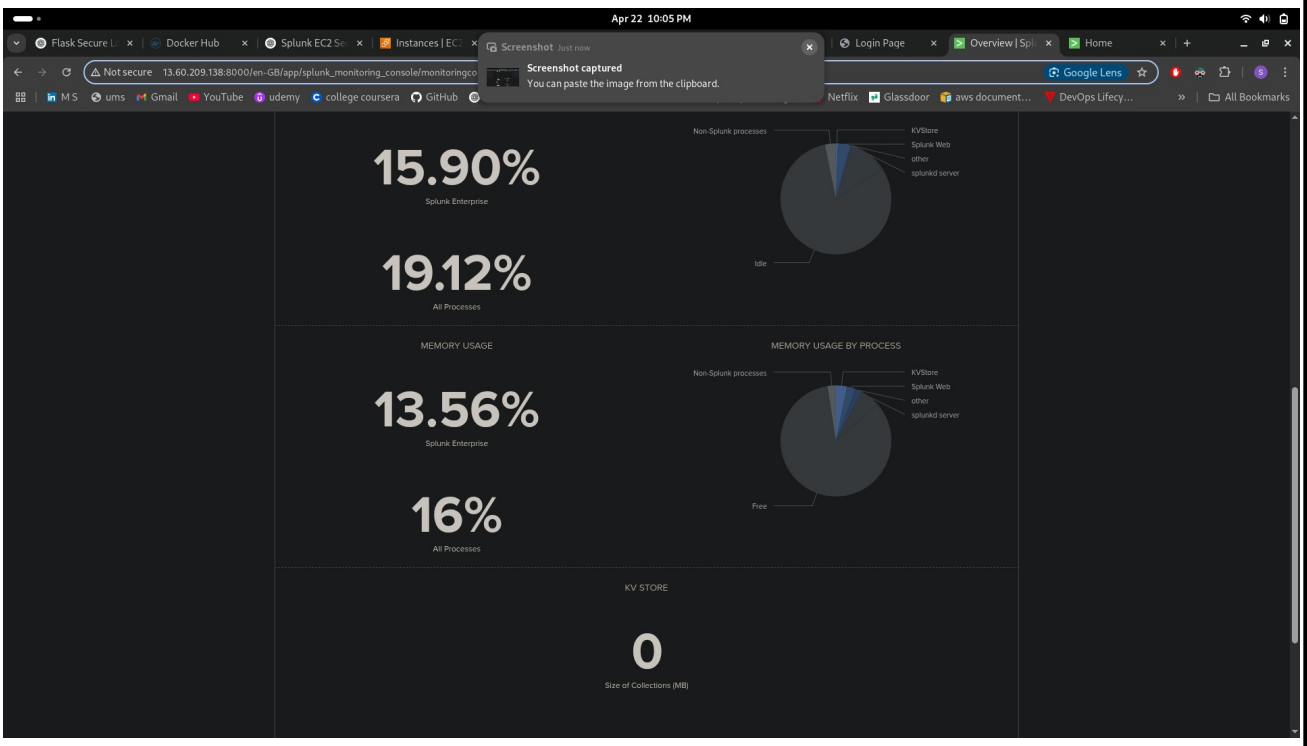


Fig : 21 see the disk usage

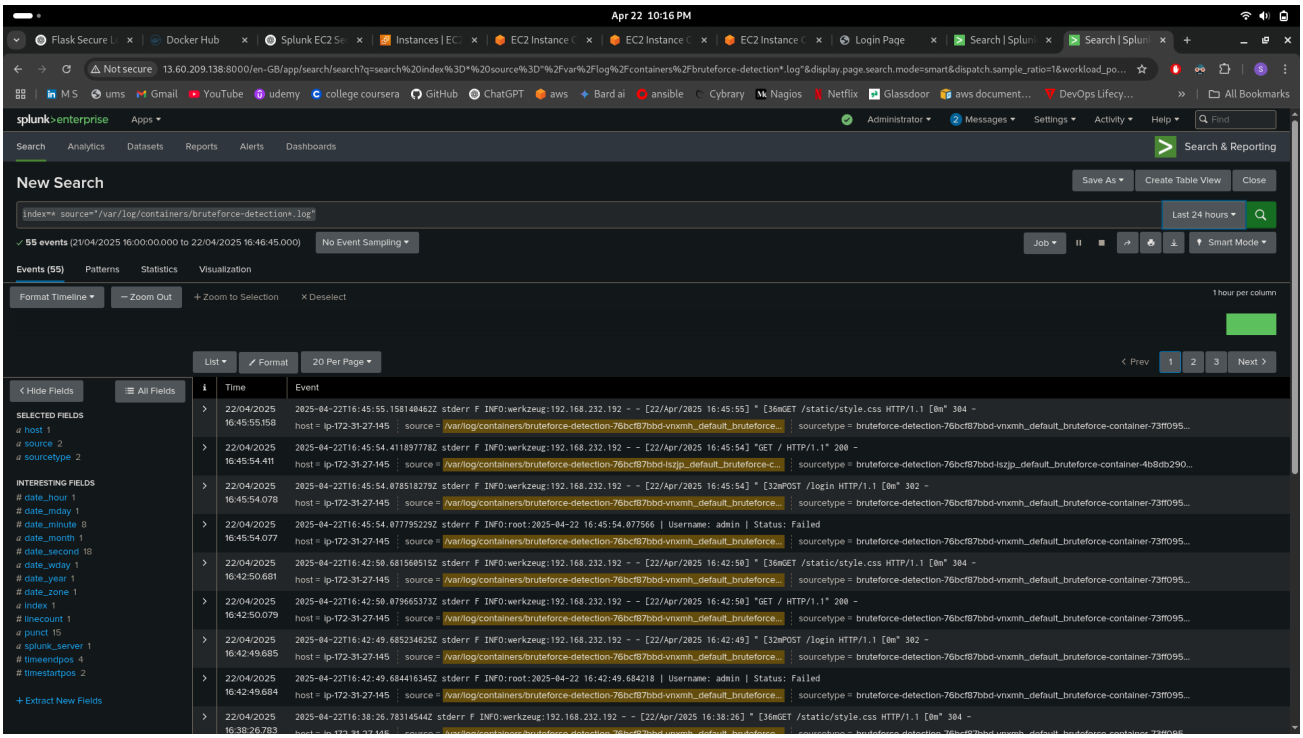


fig : 22 real time logs using

index=* source="/var/log/containers/bruteforce-detection*.log"

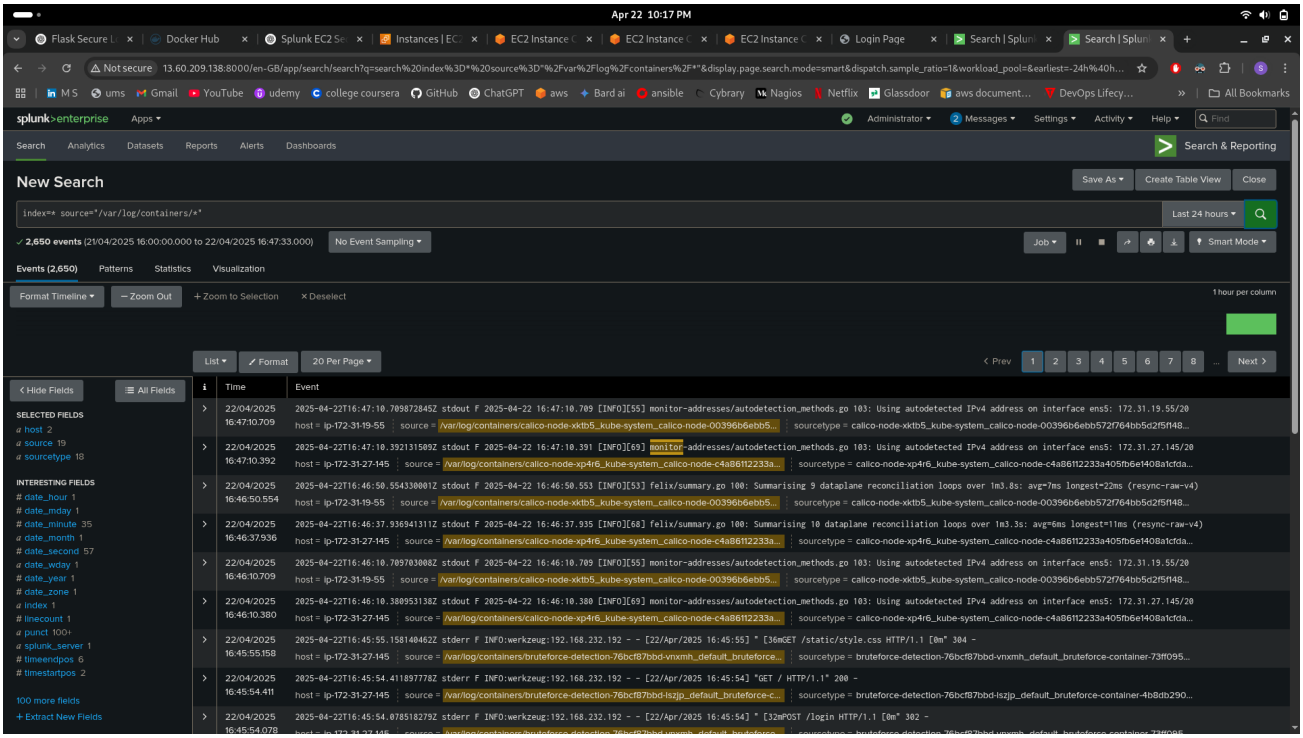


fig : 23 real time logs using

index=* source="/var/log/containers/*"

→
5 .Accessing the splunk dashboard and visualizing the logs using pie chart , bar charts…

After opening we have to access the logs via splunk

whatever the stored logs are there create csv file
Timestamp,Username,Status
2025-04-09 12:18:54.258496,admin,Success
2025-04-09 12:19:28.882961,test,Suspicious
2025-04-09 12:21:47.193320,LKJBSACIYYUHKF,Failed
2025-04-09 12:23:12.802402,d;ojihpofj;l,Failed
2025-04-09 12:24:15.819094,admin,Failed
2025-04-09 12:50:44.268440,mkcjbueoirhlk,Failed
2025-04-09 12:50:50.704899,[]APOI09EPOJ,Failed
2025-04-09 12:50:57.716414,WUQ7DIUGJB,Failed

we need to add the data
Upload the CSV file:

- Go to **Settings** > **Add Data**.
- Choose **Upload** and select the CSV file you created.
- Follow the prompts to specify the source type and index.

2. **Search the data**:
    - Go to the **Search & Reporting** app.
    - Use a search query like:

      spl
      CopyEdit
      index=your_index_name sourcetype=csv_source

      Replace your_index_name with the name of the index and csv_source with the sourcetype you specified when uploading.

3. **Create Alerts**:
    - You can set up alerts based on specific conditions. For example, you could alert if there are multiple failed login attempts in a short period.

Review
Input Type
Uploaded File
File Name
login_attempts.csv
Source Type
csv

Now we have to search via indexes in search and reporting for visualization purpose because we have already checked the live monitoring .

1. Pie Chart for Failed Logins by User

To visualize failed login attempts by each user in a pie chart, you can modify your search query to produce a count of failed logins per user and then change the visualization type.

Search query:

index=main sourcetype=csv Status="Failed" | stats count by Username

Once the results appear:

1. On the **Visualization** tab, select **Pie Chart** from the list of available chart types.

You'll now see a pie chart representing the number of failed login attempts for each username.

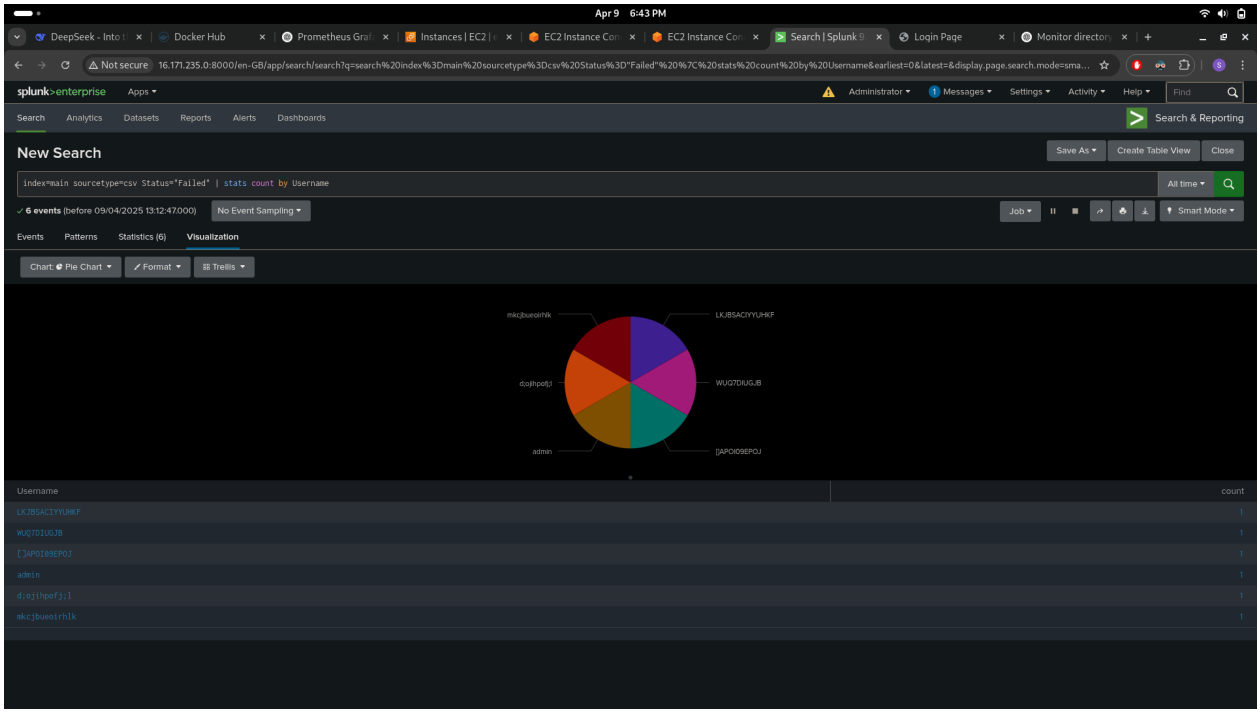


Fig : 24 Pie chart visualization

2. Bar chart visualization :

Search query:

index=main sourcetype=csv | timechart span=1h count by Status

This will show you the number of login attempts (both successful and failed) over time, with a separate line for each status (Success, Failed, Suspicious).

After running the query:

1. Go to the **Visualization** tab.
2. Choose **Line Chart** or **Area Chart** to visualize login attempts over time.

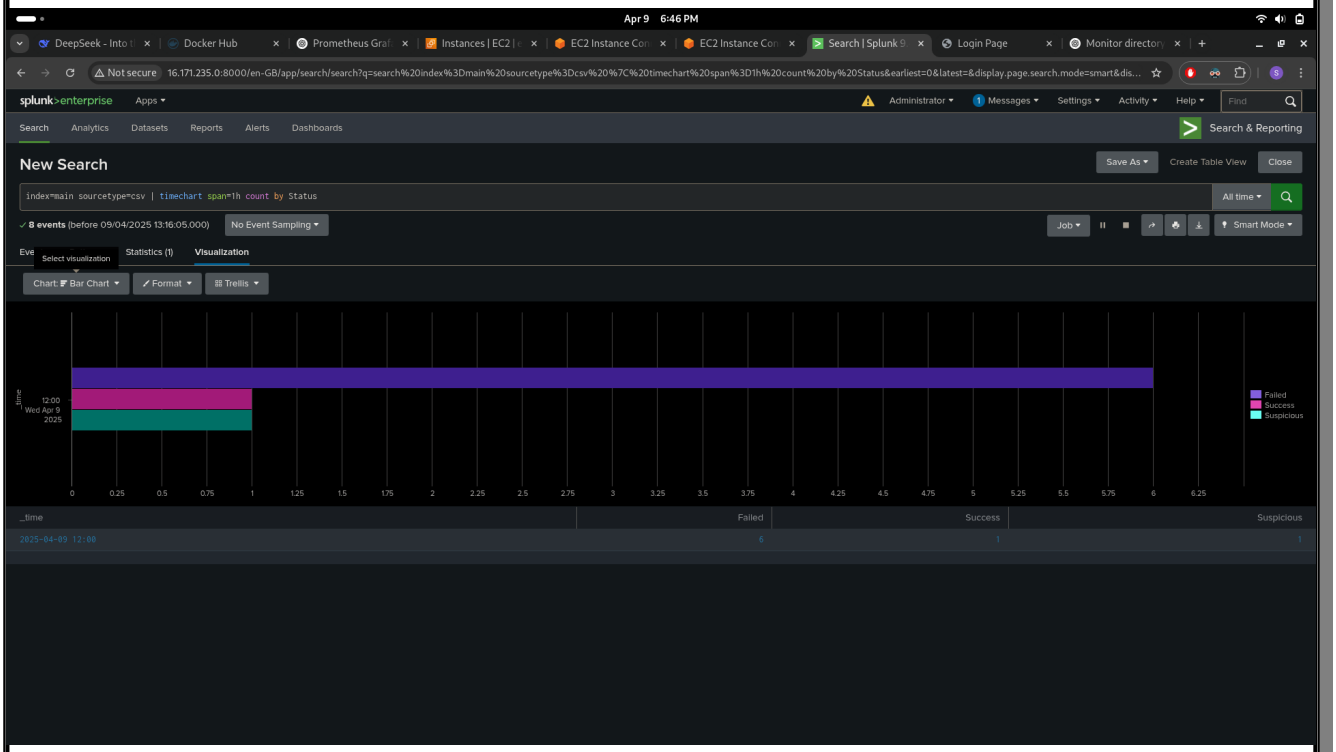


Fig :25 Bar chart visualization :

3 . Login Attempt Count for Success and Failed (Grouped by Status)

If you want to group your login attempts by their status (Success, Failed, Suspicious), you can use this:

Search query:

index=main sourcetype=csv | stats count(eval(Status="Success")) as Success, count(eval(Status="Failed")) as Failed, count(eval(Status="Suspicious")) as Suspicious

This will return a single row showing the total count of **Success**, **Failed**, and **Suspicious** login attempts.

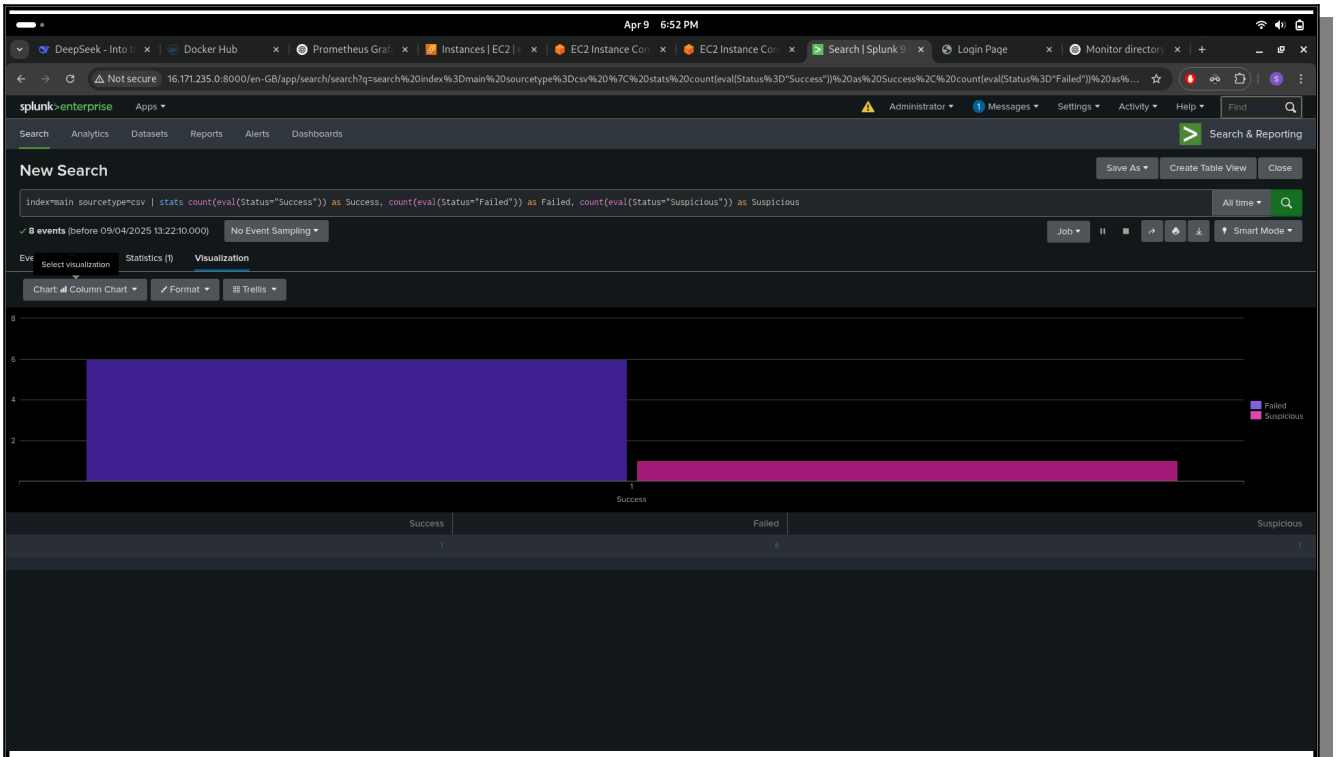


Fig :26 Login Attempt Count for Success and Failed (Grouped by Status)

4.. Count of Login Attempts by Status (Success, Failed, Suspicious)
This search will show the total number of login attempts for each status (Success, Failed, Suspicious).

Search query:

index=main sourcetype=csv | stats count by Status

This will give you a count of how many times each status (Success, Failed, Suspicious) occurred.

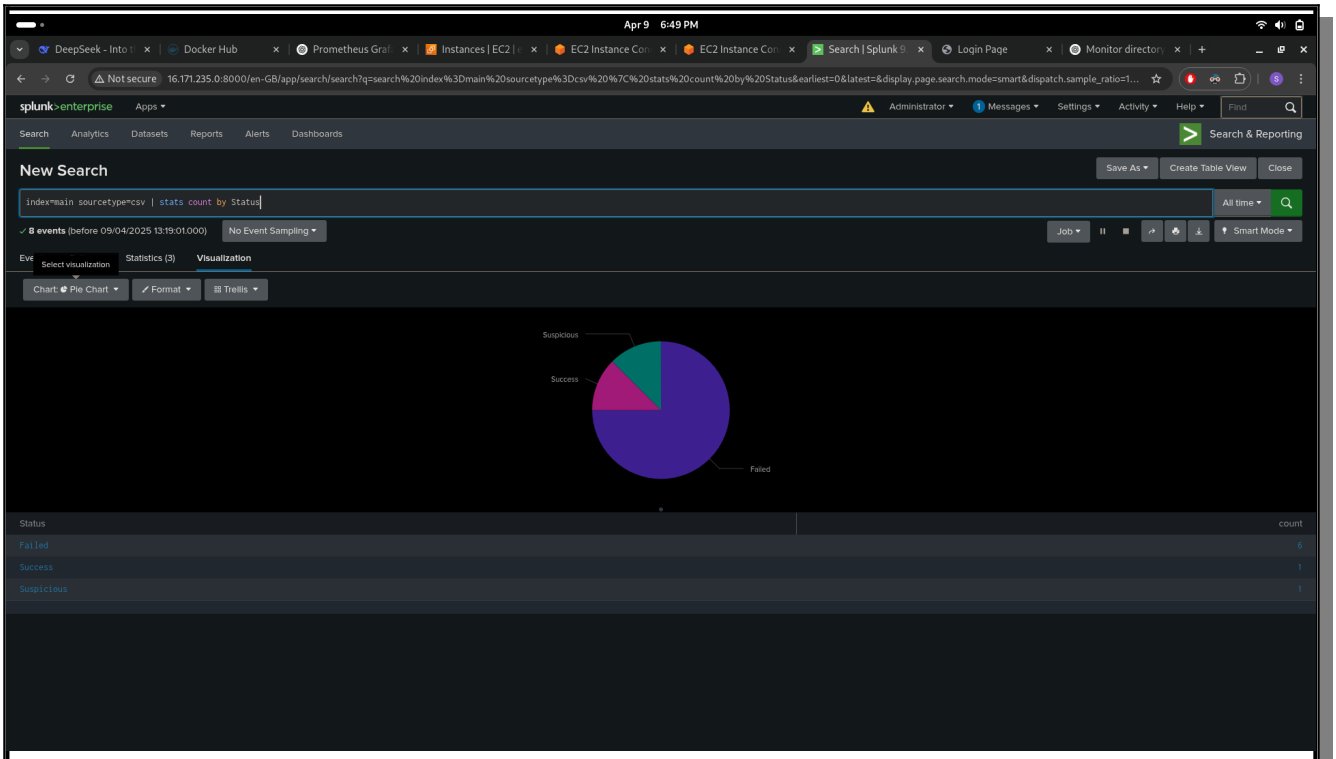


Fig : 27 Count of Login Attempts by Status (Success, Failed, Suspicious)

Now merging all images in one and saving to dashboard
Final output

fig :28Dashboard of splunk monitoring accessing different types of visualization .

6. Conclusion:

The Kubernetes Brute Force Attack Simulation & Detection System offers a controlled and secure environment to simulate and analyze brute force attack patterns in real-time. By deploying a fake login interface within a Kubernetes cluster and continuously monitoring login attempts through Splunk dashboards and real-time alerts, this project enhances visibility into malicious behaviors and strengthens proactive defense mechanisms.

This system empowers DevOps and security teams to effectively identify suspicious login activity, detect repeated credential guessing attempts, and respond swiftly to potential breaches. It serves as a valuable training and testing platform for improving incident response strategies, reinforcing authentication controls, and ensuring that containerized applications are resilient against real-world brute force threats in cloud-native ecosystems.