

# DevSecOps CI/CD Pipeline with Jenkins, Docker, Trivy, SonarQube, Prometheus, Grafana & GitHub Actions

A Secure CI/CD Automation Pipeline Built with Open Source Tools

Anita Nnamdi [www.linkedin.com/in/anita-nnamdi](https://www.linkedin.com/in/anita-nnamdi)  
GitHub: <https://github.com/Anita-ani/SECURESNAP>

## Table of Contents

1. Project Overview
2. Tools Used
3. Architecture Diagram
4. Step by Step Setup
  - a. Jenkins Setup
  - b. Docker & Agent Configuration
  - c. SonarQube Integration
  - d. Prometheus & Grafana Setup
  - e. GitHub Webhook
5. Sample Pipeline Explained
6. Security Enhancements
7. Screenshots
8. Common Errors and Fixes
9. How to Run the Pipeline

## Project Overview

SECURESNAP is a hands on DevSecOps pipeline built using Jenkins, Docker, GitHub Actions, SonarQube, GitHub Webhooks, Prometheus, and Grafana. to demonstrate secure automation from code commit to test and code analysis.

It showcases how to:

- Automate secure CI/CD pipelines
- Integrate static code analysis via SonarQube
- Use Dockerized Jenkins agents
- Monitor system health with Prometheus & Grafana
- Enforce DevSecOps principles from build to deploy

## Tools Used

Tool	Purpose
Jenkins	CI/CD automation
Docker	Containerization of Jenkins/SonarQube
GitHub	Code repository & webhook trigger
SonarQube	Static code analysis
Prometheus	System metrics collection
Grafana	Metrics dashboard & visualization
GitHub Actions	Optional alternative pipeline
Trivy	Container vulnerability scanner

## Architecture Diagram

### DevSecOps Pipeline Architecture



### DevSecOps Pipeline Architecture

This diagram shows:

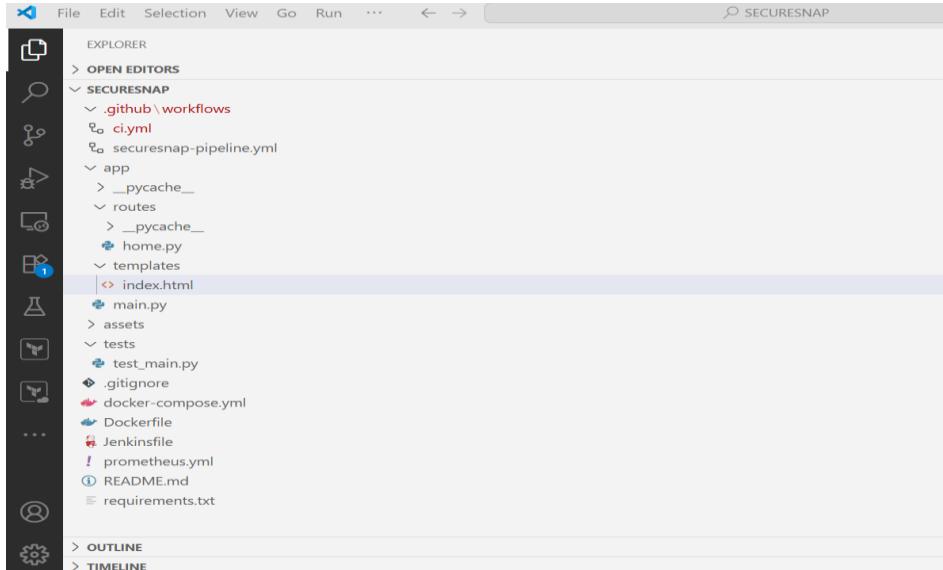
- Code **commit** triggers the pipeline
- **GitHub Actions** runs the CI/CD workflow
- **Safety** checks run (e.g., dependency scanning, linting, SAST)
- **Build** stage compiles/test/packages the app securely
- App is **deployed to AWS**
- Entire pipeline follows **DevSecOps best practices** by integrating security early

## Step-by-Step Setup

### a. Project Structure and Files

Create your project folder (SECURESNAP) with the following structure:

```
SECURESNAP/
- app/
  └── main.py
- tests/
  └── test_main.py
- Dockerfile
- docker-compose.yml
- requirements.txt
- .github/
  └── workflows/
    └── ci.yml
- Jenkinsfile
```



## Build the FastAPI App

**main.py in app/**

This is the core FastAPI application file where the API logic lives. It defines routes, responses, and how the app behaves.



The screenshot shows a code editor interface with a sidebar containing various icons. The main area displays a Python file named `main.py`. The code imports `FastAPI` and `Prometheus`, initializes an `Instrumentator`, adds metrics for latency, requests, and response size, and defines two endpoints: `/` and `/health`.

```
app > main.py > health
1  from fastapi import FastAPI, Request
2  from prometheus_fastapi_instrumentator import Instrumentator, metrics
3
4  app = FastAPI()
5
6  instrumentator = Instrumentator(
7      should_group_status_codes=True,
8      should_ignore_untemplated=True,
9      should_group_untemplated=True,
10 )
11
12 # Optional: Add custom metrics like latency, request sizes, etc.
13 instrumentator.add(metrics.latency())
14 instrumentator.add(metrics.requests())
15 instrumentator.add(metrics.response_size())
16 instrumentator.add(metrics.request_size())
17
18 instrumentator.instrument(app).expose(app, include_in_schema=False)
19
20 @app.get("/")
21 async def home():
22     return {"message": "Hello from SECURESNAP a full DevSecOps"}
23
24 @app.get("/health")
25 async def health():
26     return {"status": "OK"}
27
```

## test\_main.py

Contains automated tests to verify that your API endpoints work correctly. Helps catch bugs and ensures code reliability during development and deployment.

```
! prometheus.yml X docker-compose.yml home.py index.html test_main.py X
tests > test_main.py > ...
1  from fastapi.testclient import TestClient
2  from app.main import app
3
4  client = TestClient(app)
5
6  def test_health():
7      response = client.get("/health")
8      assert response.status_code == 200
9      assert response.text == "OK"
10
```

## Docker Agent Setup

Create a Docker agent with required tools

In terminal run:

```
RUN apt update && \
apt install -y docker.io docker-compose nodejs npm && \
npm install -g snyk
```

## SonarQube Integration

1. Launch SonarQube using:

```
- docker run -d --name sonarqube -p 9000:9000 sonarqube
```

2. Get the token from Sonar dashboard.

### 3. Initial Setup for SonarQube

- Visit <http://localhost:9000>
- Login (default: admin / admin)
- Change password
- Go to **My Account → Security → Generate Token**

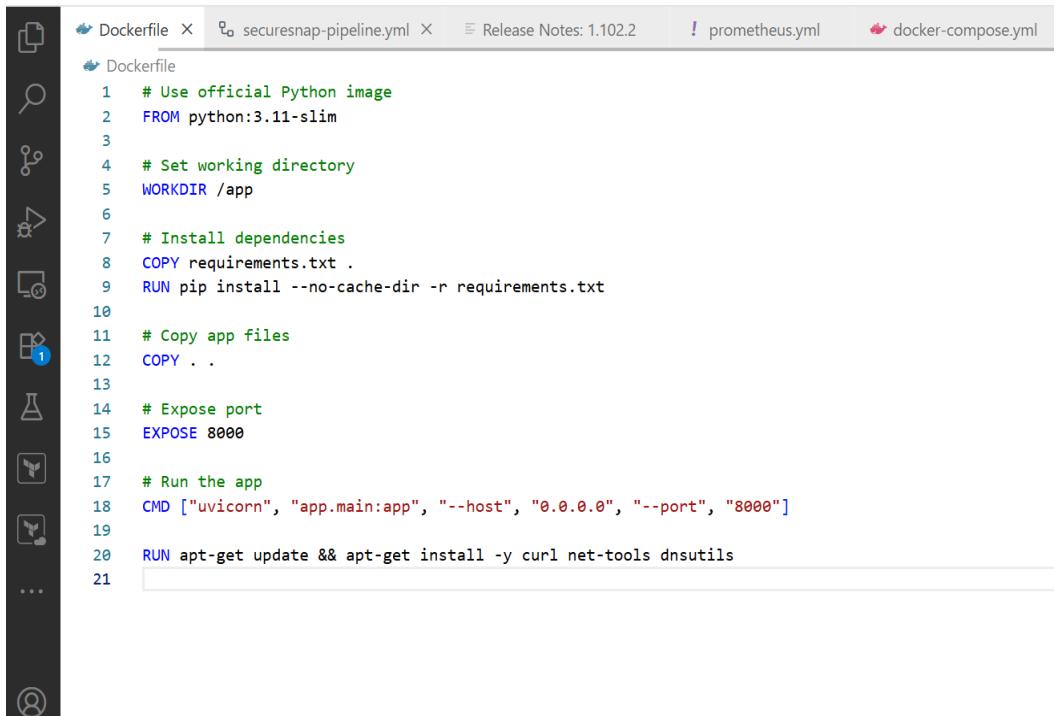
4. Add **Secret Text**:

- Name it something like jenkins-sonar
- Copy and save this token

## Create a Dockerfile

### Dockerfile

Defines how to build the app's container image—sets up Python, installs dependencies, copies app files, exposes the app port, and runs the server.



The screenshot shows a code editor interface with several tabs at the top: Dockerfile (selected), secursnap-pipeline.yml, Release Notes: 1.102.2, prometheus.yml, and docker-compose.yml. The Dockerfile tab contains the following code:

```
1 # Use official Python image
2 FROM python:3.11-slim
3
4 # Set working directory
5 WORKDIR /app
6
7 # Install dependencies
8 COPY requirements.txt .
9 RUN pip install --no-cache-dir -r requirements.txt
10
11 # Copy app files
12 COPY .
13
14 # Expose port
15 EXPOSE 8000
16
17 # Run the app
18 CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
19
20 RUN apt-get update && apt-get install -y curl net-tools dnsutils
21
```

## Create docker-compose.yml

### docker-compose.yml

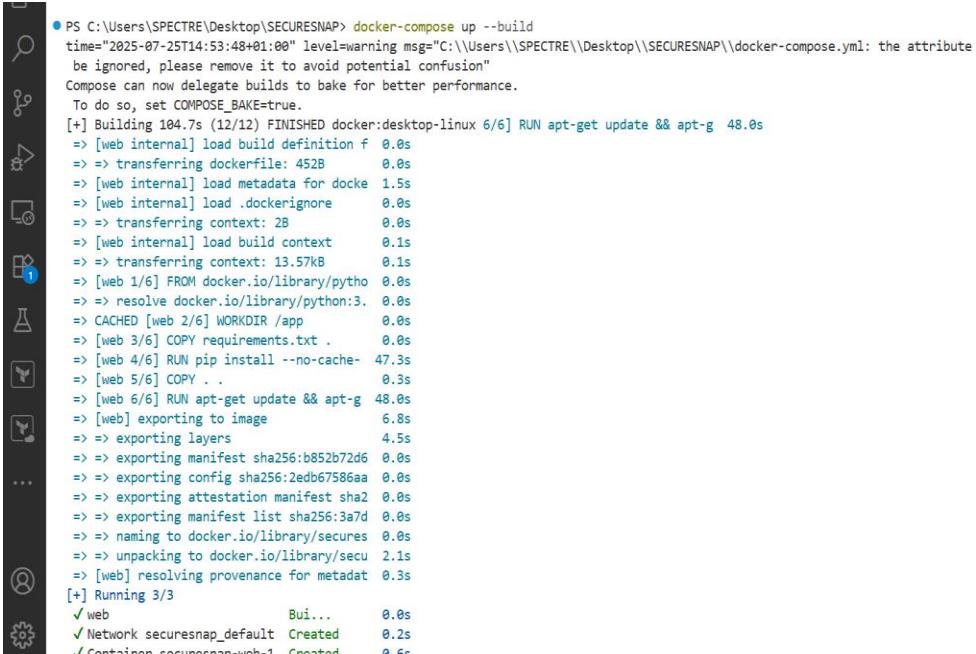
It specifies the Docker Compose file format version. Version 3.9 is stable and compatible with recent Docker versions, supporting features like named volumes, secrets, and improved service definitions.

```
02.2 ! prometheus.yml X docker-compose.yml X home.py index.html test_main.py requirements.txt

# docker-compose.yml > {} services > {} grafana > [ ]depends_on
# docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platform-agnostic applications (compose-spec.json)
1 version: '3.9'
2
3 services:
4   web:
5     build: .
6     container_name: securssnap-web
7     ports:
8       - "8000:8000"
9     volumes:
10      - ./app
11     environment:
12       - PYTHONUNBUFFERED=1
13
14   prometheus:
15     image: prom/prometheus:latest
16     container_name: prometheus
17     volumes:
18       - ./prometheus.yml:/etc/prometheus/prometheus.yml
19     ports:
20       - "9090:9090"
21     depends_on:
22       - web
23
24   grafana:
25     image: grafana/grafana:latest
26     container_name: grafana
27     ports:
28       - "3000:3000"
29     depends_on:
```

## Build and Run

- `docker-compose build`
- `docker-compose up`



```

● PS C:\Users\SPECTRE\Desktop\SECURESNAP> docker-compose up --build
time="2025-07-25T14:53:48+01:00" level=warning msg="C:\Users\SPECTRE\Desktop\SECURESNAP\docker-compose.yml: the attribute
be ignored, please remove it to avoid potential confusion"
Compose can now delegate builds to bake for better performance.
To do so, set COMPOSE_BAKE=true.
[+] Building 104.7s (12/12) FINISHED docker:desktop-linux 6/6] RUN apt-get update && apt-g 48.0s
=> [web internal] load build definition f 0.0s
=> => transferring dockerfile: 452B 0.0s
=> [web internal] load metadata for docke 1.5s
=> [web internal] load .dockignore 0.0s
=> => transferring context: 2B 0.0s
=> [web internal] load build context 0.1s
=> => transferring context: 13.57kB 0.1s
=> [web 1/6] FROM docker.io/library/pytho 0.0s
=> => resolve docker.io/library/python:3. 0.0s
=> CACHED [web 2/6] WORKDIR /app 0.0s
=> [web 3/6] COPY requirements.txt . 0.0s
=> [web 4/6] RUN pip install --no-cache- 47.3s
=> [web 5/6] COPY . . 0.3s
=> [web 6/6] RUN apt-get update && apt-g 48.0s
=> [web] exporting to image 6.8s
=> => exporting layers 4.5s
=> => exporting manifest sha256:b852b72d6 0.0s
=> => exporting config sha256:2edb67586aa 0.0s
=> => exporting attestation manifest sha2 0.0s
=> => exporting manifest list sha256:3a7d 0.0s
=> => naming to docker.io/library/secur 0.0s
=> => unpacking to docker.io/library/secu 2.1s
=> [web] resolving provenance for metadat 0.3s
[+] Running 3/3
✓ web           Bui...  0.0s
✓ Network securresnap_default Created 0.2s
./ContainerName securresnap_1 Created 0.0s

```

-docker ps -a to view running container



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
19016f2565cc	grafana/grafana:latest	"/run.sh"	12 hours ago	Up 12 hours	0.0.0.0:3000->3000/tcp	grafana
2144db876ec5	prom/prometheus:latest	"/bin/prometheus --c..."	12 hours ago	Up 12 hours	0.0.0.0:9000->9000/tcp	prometheus
e12f3f9786fe	securesnap-web	"unicorn app.main:ap..."	12 hours ago	Up 12 hours	0.0.0.0:8000->8000/tcp	securesnap-web
a0f26ee04154	busybox	"nslookup google.com"	2 days ago	Exited (0) 2 days ago		sweet_albattani
136ec3945603	sonarqube:lts	"/opt/sonarqube/dock..."	2 days ago	Exited (255) 2 days ago	0.0.0.0:9000->9000/tcp	sonarqube
00fa3d749221	fastapi-app	"uvicorn main:app --..."	4 days ago	Exited (255) 3 days ago	0.0.0.0:8000->8000/tcp	fastapi-container

## Push to Github

git add .

git commit -m "Initial commit"

git push origin main

The screenshot shows a GitHub repository page for 'SECURESNAP'. The repository is public and has 15 commits. The commit history includes:

- AnitaAnn Screenshots - 7e0cb59 · 1 hour ago
- .github/workflows - Add CI workflow to build, scan, and push Docker image · 2 days ago
- app - Initial commit:DevSecOps pipeline with Jenkins, Docker,So... · 4 hours ago
- assets - Screenshots · 1 hour ago
- tests - integrations to this project · 2 days ago
- .gitignore - integrations to this project · 2 days ago
- Dockerfile - Initial commit - SECURESNAP base app with Docker · 2 days ago
- Jenkinsfile - Initial commit: DevSecOps pipeline with Jenkins, Docker,So... · 4 hours ago

On the right side, there are sections for 'About', 'Releases', and 'Package'.

## Add DockerHub Secrets to GitHub

- Go to your repo → Settings → Secrets → Actions.
- Add:
- DOCKER\_USERNAME
- DOCKER\_PASSWORD
- Log into docker hub to generate Personal access token, copy token immediately and paste in DOCKER\_PASSWORD and docker username in DOCKER\_USERNAME

The screenshot shows the GitHub 'Secrets and variables' section. On the left, a sidebar lists various GitHub features like Branches, Tags, Rules, Actions, Models, Webhooks, Copilot, Environments, Codespaces, and Pages. The 'Secrets and variables' section is highlighted. The main area is divided into 'Environment secrets' and 'Repository secrets'. Under 'Environment secrets', it says 'This environment has no secrets.' and has a 'Manage environment secrets' button. Under 'Repository secrets', there are two entries: 'DOCKER\_PASSWORD' and 'DOCKER\_USERNAME', both updated 2 days ago. Each entry has edit and delete icons.

## Create CI Workflow with Trivy

.github/workflows/ci.yml

To ensure our Docker image is secure and free from known vulnerabilities, we integrate **Trivy**, a powerful container scanning tool, into our GitHub Actions CI workflow. This workflow automatically scans the Docker image every time we push changes to the **main** branch, helping us catch CVEs and misconfigurations early in the development cycle.

The screenshot shows the GitHub Actions pipeline history for the 'SECURESNAP CI/CD Pipeline'. The pipeline consists of several steps, all of which have been successful. The steps are:

- Initial commit: DevSecOps pipeline with Jenkins, Doc...** (SECURESNAP CI/CD Pipeline #13: Commit d5d9395 pushed by Anita-ani) - main branch, 4 hours ago, 1m 1s
- Update to Jenkinsfile** (SECURESNAP CI/CD Pipeline #12: Commit 536c1f4 pushed by Anita-ani) - main branch, yesterday, 1m 5s
- Update to Jenkinsfile** (SECURESNAP CI/CD Pipeline #11: Commit 2deba93 pushed by Anita-ani) - main branch, yesterday, 1m 1s
- Update Jenkinsfile** (SECURESNAP CI/CD Pipeline #10: Commit 5edc871 pushed by Anita-ani) - main branch, yesterday, 58s
- Update Jenkinsfile** (SECURESNAP CI/CD Pipeline #9: Commit d75baa1 pushed by Anita-ani) - main branch, 2 days ago, 1m 2s
- Initial commit: DevSecOps pipeline with Jenkins, Do...** (SECURESNAP CI/CD Pipeline #8: Commit 928893b pushed by Anita-ani) - main branch, 2 days ago, 57s

## Deploy to AWS EC2

### Steps:

1. Launch EC2 (Ubuntu)
  - Go to security groups and apply inbound rules, port 22, 80 and 443.
  - Find
2. Create a key pair and download using .pem
3. ls to find downloaded key pair then log into EC2 using SSH
4. SSH into instance using ssh -i "your-key.pem" ubuntu@<public-ip>(click on connect to find).

```
ubuntu@ip-172-31-32-40:~$ which docker
/usr/bin/docker
ubuntu@ip-172-31-32-40:~$ sudo docker login
USING WEB-BASED LOGIN
Info → To sign in with credentials on the command line, use 'docker login -u <username>'

Your one-time device confirmation code is: TGDW-PPNM
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate
Waiting for authentication in the browser...
WARNING! Your credentials are stored unencrypted in '/root/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

Login Succeeded
ubuntu@ip-172-31-32-40:~$ sudo docker pull anita/securresnap:latest
latest: Pulling from anita/securresnap
59e22667830b: Pull complete
ab846faicdb: Pull complete
b7b61708209a: Pull complete
4085babbc570: Pull complete
fba89926af8f: Pull complete
5198ef7747f9: Pull complete
ec350ddad843: Pull complete
3134d8d5fd9f: Pull complete
c1f8917142aa: Pull complete
Digest: sha256:ba285ed46d1944be7215649dfa23a00fcc2b5b59b37de540f9fc7b52874b915
Status: Downloaded newer image for anita/securresnap:latest
docker.io/anita/securresnap:latest
ubuntu@ip-172-31-32-40:~$ sudo docker run -d -p 80:8000 anita/securresnap:latest
39200ec1764b5cc0289624560362a0fcfc61a1268d7bd96c481376c55b2e129fd
ubuntu@ip-172-31-32-40:~$ exit
```

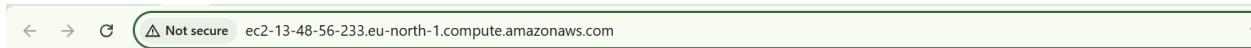
5. Install docker using:
  - curl -fsSL https://get.docker.com -o get-docker.sh
  - sudo sh get-docker.sh
6. Log into docker using:
  - sudo docker login
  - Copy <https://login.docker.com/activate> in terminal and paste to browser.
  - Enter the confirmation code provided in terminal.

7. Pull image using:

```
-docker pull an1ta/securesnap:latest
```

```
-docker run -d -p 80:8000 an1ta/securesnap:latest
```

8. Copy EC2 instance ip (Connection to ec2-13-48-56-233.eu-north-1.compute.amazonaws.com)



## Welcome to SECURESNAP!

This is a secure app with full DevSecOps pipeline.

**Step-by-step setup to integrate Prometheus and Grafana via Docker so you can scrape FastAPI metrics and visualize them.**

Step1: Expose fastAPI Metrics

First, install and expose metrics from your FastAPI app:

In `requiremenst.txt` include:

```
prometheus-client
```

In `main.py` include:

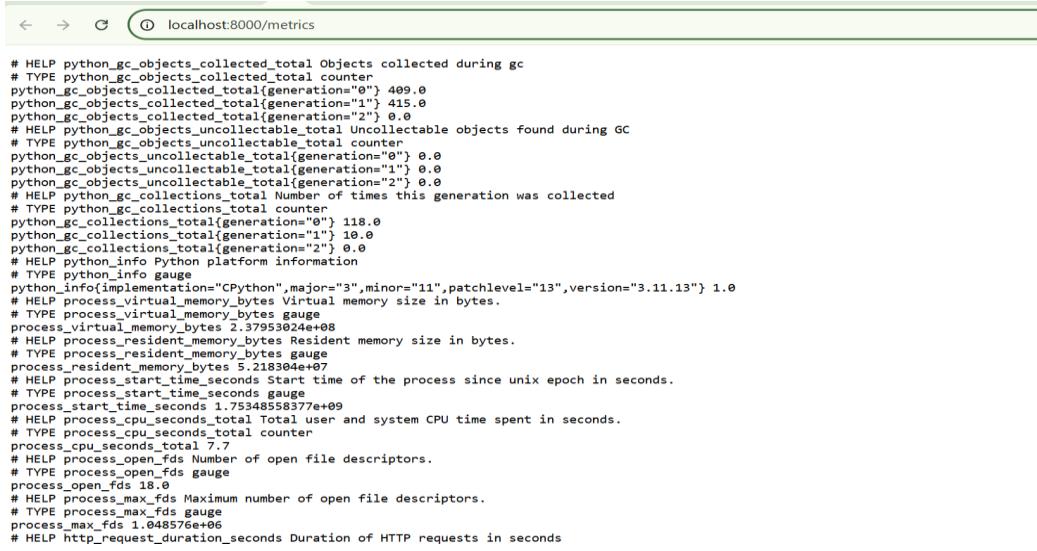
```
from fastapi import FastAPI from prometheus_client import start_http_server, Counter import threading

app = FastAPI()

REQUEST_COUNT = Counter("app_requests_total", "Total number of requests")

@app.get("/") def read_root(): REQUEST_COUNT.inc() return {"message": "Hello, Prometheus!"}
```

## Refer to Repo



```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generations="0"} 409.0
python_gc_objects_collected_total{generations="1"} 415.0
python_gc_objects_collected_total{generations="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generations="0"} 0.0
python_gc_objects_uncollectable_total{generations="1"} 0.0
python_gc_objects_uncollectable_total{generations="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 118.0
python_gc_collections_total{generation="1"} 10.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython", major="3", minor="11", patchlevel="13", version="3.11.13"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 2.37953024e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 5.2318304e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.75348558377e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 7.7
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 18.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP http_request_duration_seconds Duration of HTTP requests in seconds
```

## Prometheus + Grafana Monitoring Setup

Step by step:

### 1. Install Prometheus

- sudo useradd --no-create-home --shell /bin/false prometheus
- sudo mkdir /etc/prometheus /var/lib/prometheus

### 2. prometheus.yml

This is Prometheus' configuration file. It tells Prometheus **where to scrape metrics from**—such as your FastAPI app or other services—by defining jobs, targets, and how often to collect data.



```
! prometheus.yml ✘ docker-compose.yml ✘ ci.yml 4 home.py index.html test_main.py requirements.txt
! prometheus.yml > [ ] scrape_configs > {} 0 > [ ] static_configs > {} 0
  prometheus.json - Prometheus configuration file (prometheus.json)
1   global:
2     |   scrape_interval: 15s
3
4   scrape_configs:
5     - job_name: 'securesnap-fastapi'
6       static_configs:
7         - targets: [':8000']
8
```

## Grafana

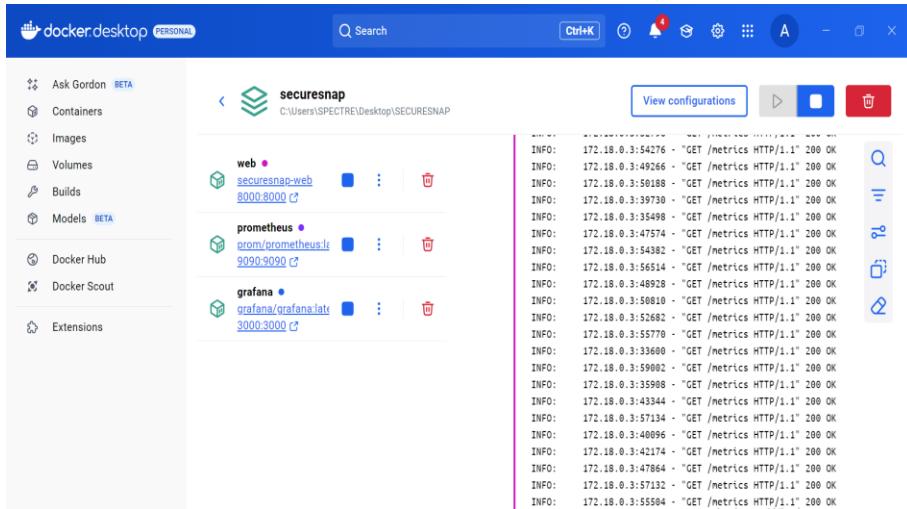
### 1. Install Grafana

- sudo apt-get install -y apt-transport-https software-properties-common
- sudo add-apt-repository "deb <https://packages.grafana.com/oss/deb> stable main"
- sudo apt-get update
- sudo apt-get install grafana
- sudo systemctl start grafana-server

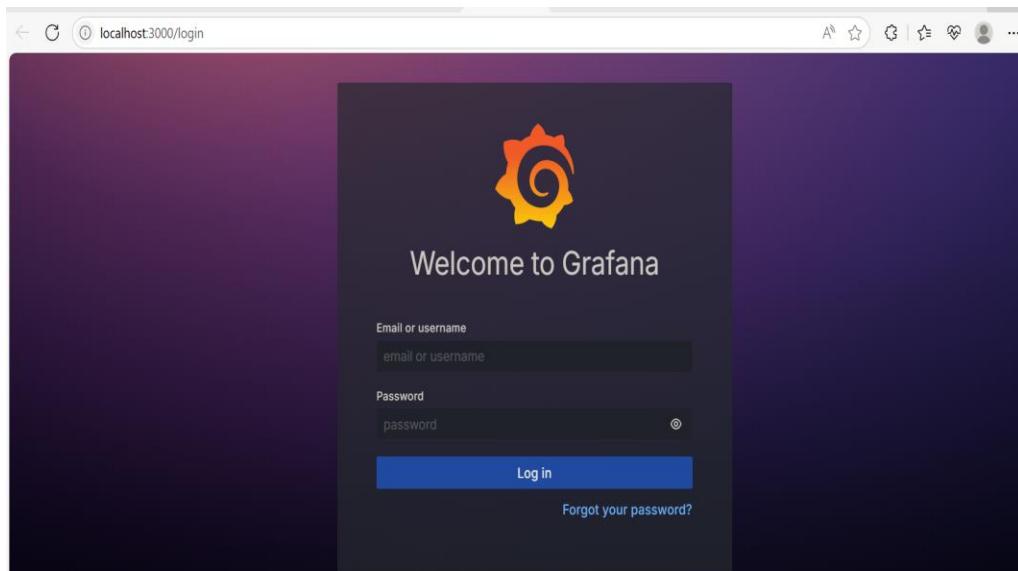
Login: <http://localhost:3000/>

***Start Everything using:***

-docker-compose up -build



- Visit docker desktop to view running container
- Visit Prometheus: <http://localhost:9090>
- Visit Grafana: <http://localhost:3000> (login with admin/admin)



**Configure Grafana**

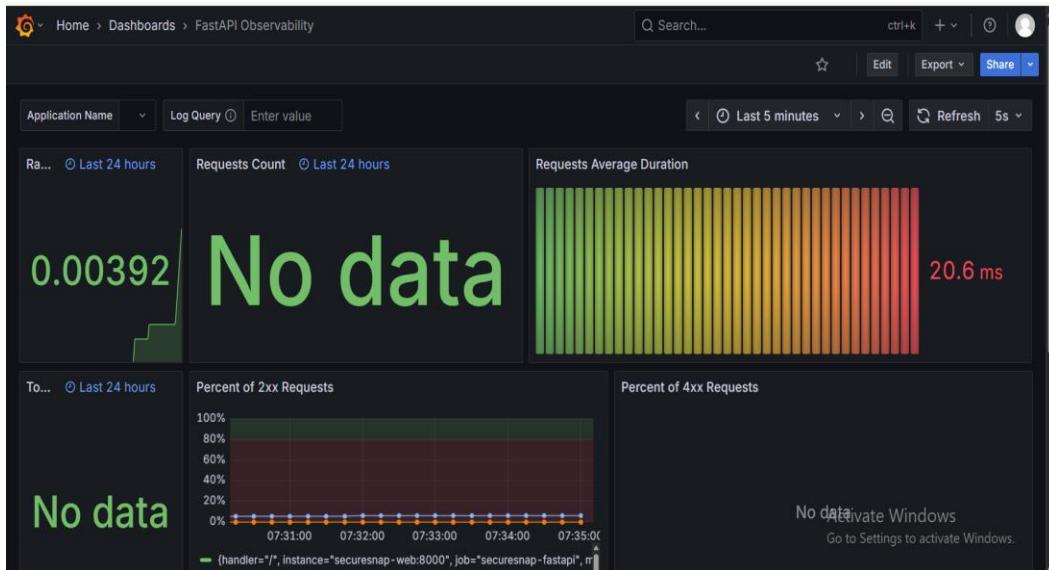
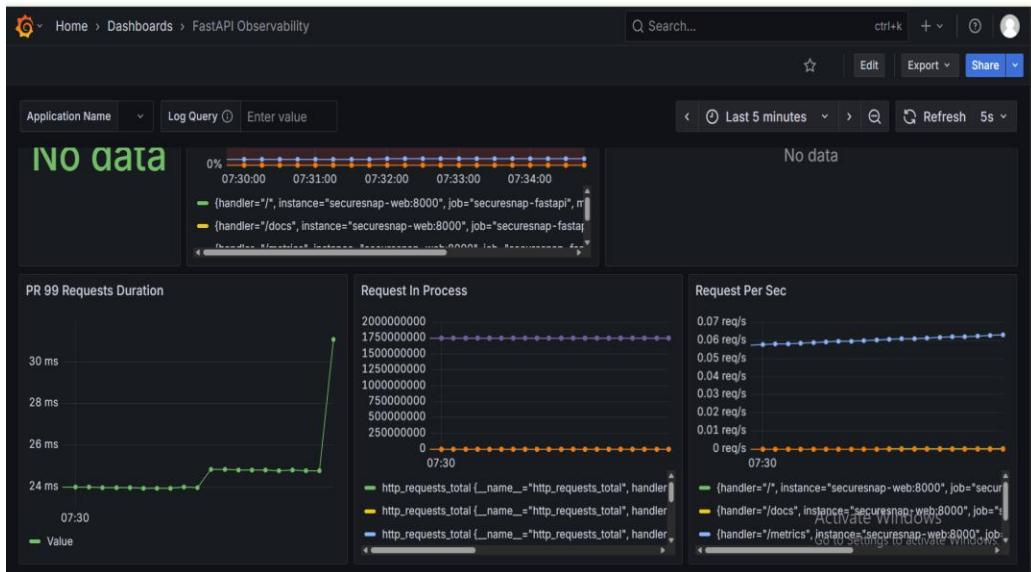
- Go to <http://localhost:3000>
- Login: admin / admin
- Add **Prometheus** as a data source:
- Set URL to: <http://prometheus:9090>
- Save and test
- Import a dashboard (or create one) to visualize metrics
- You can use a popular Prometheus + FastAPI dashboard like ID: 18739
- Go to the **+ (Create)** menu → Click **Import**
- In **Import via Grafana.com**, paste:
- Click load and Prometheus data source you just added
- Click Import

### ***View the FastAPI Dashboard***

Once imported:

You'll see a ready-made dashboard showing:

- Request count
- Response time
- Error rates
- Uptime
- CPU/RAM (if exposed)



## Jenkins Setup

1. Install Java using:  
`-sudo apt install -y openjdk-17-jdk`

2. Install Jenkins using:

```
- wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
-sudo sh -c 'echo deb https://pkg.jenkins.io/debian binary/ >
/etc/apt/sources.list.d/jenkins.list'
-sudo apt update
-sudo apt install jenkins
```

3. Start jenkins and login into your EC2 (eg,...<http://ec2-13-48-56-233.eu-north-1.compute.amazonaws.com/>)

4. Jenkins Plugins to Install

Go to **Manage Jenkins → Plugins → Available**, and install:

- GitHub Integration
- Docker Pipeline
- OWASP Dependency-Check
- Trivy Plugin
- Blue Ocean
- Pipeline
- SonarQUBE Scanner
- Docker Pipeline
- Prometheus metrics plugins

5. Create Jenkinsfile

The **Jenkinsfile** defines a **CI/CD pipeline** for automating the build, test, and deployment of your FastAPI app. It tells Jenkins what steps to run and in what order.

**Key Stages:**

- **Build:** Uses Docker to build the FastAPI app image.
- **Test:** Runs unit tests (`pytest`) to catch issues early.
- **Security Scan:** Scans the image (e.g. with Trivy) for vulnerabilities.
- **Push:** Pushes the Docker image to Docker Hub (if tests pass).
- **Deploy:** Deploys the image to a server or container platform.

This keeps your delivery secure, reliable, and automated.

```

1     pipeline {
2         agent any
3         <<<<< HEAD
4
5         environment {
6             PYTHONUNBUFFERED = 1
7         }
8
9         stages {
10            stage('Clone Repo') {
11                steps {
12                    echo 'Cloning repository...'
13                    // This is optional if Jenkins is pulling via a webhook
14                    // git url: 'https://github.com/your-username/your-repo.git'
15                }
16            }
17
18            stage('Set up Python') {
19                steps {
20                    echo 'Setting up Python...'
21                    sh 'python3 -m venv venv'
22                    sh '. venv/bin/activate && pip install --upgrade pip setuptools'
23                }
24            }
}

```

## Add GitHub Repo in Jenkins

- Create a new **Pipeline Job (New item)**
- Set name (Item name)
- Click on Pipeline then Ok
- Navigate to Pipeline
- Connect it to GitHub under "Pipeline from SCM"
- Under SCM choose Git
- Paste Repo URL
- Branch main or master then save
- Use your GitHub personal access token if private

S	W	Name	Last Success	Last Failure	Last Duration	F
(X)	Cloud	instagram-brute-force-replay	N/A	1 day 2 hr #3	1.8 sec	▶ ⭐
(X)	Cloud	SECURESNAP	N/A	1 hr 17 min #19	2.7 sec	▶ ⭐

## GitHub Webhook

This allows Jenkins to automatically build your project whenever you **push changes to GitHub**, by using **webhooks** to notify Jenkins of a new event.

### **Configure Jenkins Job:**

1. Go to your Jenkins job > **Configure**
2. Under **Source Code Management**, select:
  - a. Git
  - b. Add your **GitHub repo URL**
  - c. Use credentials if needed (GitHub token or SSH key)
3. Under **Build Triggers**, check:
  - a. GitHub hook trigger for GITScm polling
4. Save the configuration.

## Setup Recap

Automatically trigger Jenkins pipeline builds from GitHub commits and monitor Jenkins using Prometheus and Grafana.

### **Components:**

- GitHub repo with credentials/token set up
- Jenkins (with Docker agents or controller)
- GitHub Webhook → Jenkins webhook endpoint
- Prometheus scrapes Jenkins metrics
- Grafana visualizes Prometheus data

## Security Enhancements

Component	Security Best Practices
Jenkins Server	Use HTTPS (e.g. behind NGINX), disable anonymous access
GitHub Webhook	Use <b>Secret Token</b> (set in webhook and Jenkins plugin)
Credentials	Use Jenkins <b>Credentials Plugin</b> to store GitHub tokens securely
Docker Builds	Run agents with limited privileges, avoid root in Dockerfiles
Prometheus	Restrict Prometheus UI access and scrape targets with firewalls
Grafana	Use strong passwords, disable anonymous dashboards, restrict sharing

## Common Errors and Fixes

Error Message	Cause	Fix
GitHub webhook returns 403 or timeout	Jenkins not publicly accessible	Use ngrok, configure reverse proxy e.g. NGINX

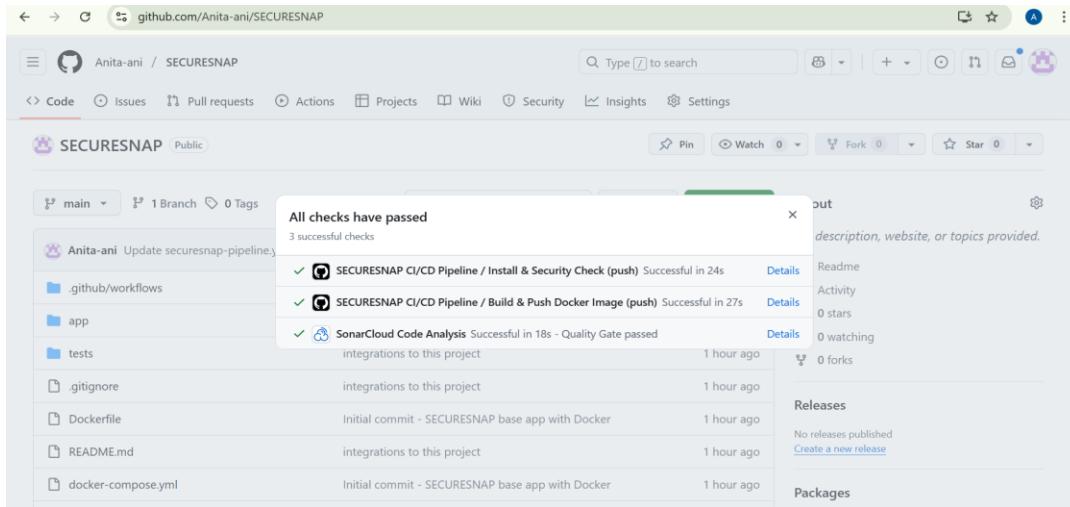
Jenkins not triggering on push	Webhook misconfigured or wrong Jenkins URL	Verify webhook URL ends with /github-webhook/
Git clone fails	Missing credentials	Add GitHub token or SSH key to Jenkins credentials
Prometheus shows Jenkins job as down	Wrong target URL or port	Ensure Jenkins exposes /prometheus/ endpoint
Grafana dashboard empty	Wrong PromQL queries or missing data	Verify Prometheus is scraping metrics correctly

## How to Run the Pipeline

1. **Make a Code Change in GitHub**
  - a. Edit any file (e.g. main.py), commit and push
2. **GitHub Webhook Fires**
  - a. GitHub sends payload to Jenkins webhook endpoint
3. **Jenkins Receives Hook**
  - a. If configured correctly, Jenkins fetches the repo and starts the pipeline
4. Build/Tests Run Inside Jenkins Pipeline
  - a. Optionally include Docker build, pytest, lint, security scan, etc.
5. Prometheus Monitors Build Metrics
  - a. If Jenkins Prometheus plugin is enabled, metrics are exposed and scraped
6. Grafana Dashboard Displays Build Trends
  - a. Grafana visualizes builds, duration, failures, success rate, etc.

## What Is Achieved with This Architecture?

By implementing this **GitHub Actions → SonarQube → Docker → AWS EC2 → Jenkins CI/CD → Prometheus + Grafana Monitoring** setup, you achieve the following:



## 1. Automated CI/CD Pipeline

- **Continuous Integration:** Every code push automatically triggers a build and test cycle.
- **Continuous Delivery/Deployment:** You can extend this to deploy to staging or production automatically (e.g... via Helm, Docker, or Kubernetes).
- **Benefits:**
  - Faster feedback
  - Fewer manual steps
  - Early bug detection
  - Consistent releases

## 2. Built In Security with SAST (Static Application Security Testing)

Run SAST tools like `bandit`, `semgrep`, or `trivy` during Jenkins pipeline stages.

### What it Catches:

- Hardcoded secrets
- Insecure functions or APIs
- Dependency vulnerabilities

### Benefits:

- Shift-left security
- Immediate feedback to developers
- Prevents vulnerable code from reaching production

### Sample Jenkinsfile SAST Stage:

```
groovy
Copy code
stage('SAST') {
    steps {
        sh 'bandit -r . || true'
    }
}
```

## 3. Quality Gates

- Thresholds that code must pass (e.g... test coverage, linting score, SAST results).
- Use tools like SonarQube, pylint, pytest-cov integrated into the pipeline.
- **Example Rule:** Block merge if test coverage < 80% or lint score < 8/10.

### Benefits:

- Prevents low quality or insecure code
- Maintains project standards
- Enforces coding discipline across teams

## 4. GitHub Webhook & GitHub Actions Compatibility

**Webhook:** Enables GitHub to notify Jenkins of push events, triggering builds.

**Actions:** You can optionally integrate GitHub Actions for:

- Pre-commit tests
- Linting and secrets scan before Jenkins build
- SAST checks on pull requests

**Hybrid Example:** GitHub Actions runs truffleHog for secrets detection → Jenkins runs build/test/deploy pipeline.

## 5. Full Observability and Monitoring

**Prometheus + Grafana:** You get metrics like:

- Build duration trends
- Success/failure rates
- Queue latency

**Jenkins Prometheus Plugin:** Exposes /prometheus metrics endpoint for scraping

**Benefits:**

- Performance bottleneck detection
- Audit trails for compliance
- Reliability metrics for teams

## 6. Modular, Scalable, and Secure DevSecOps Pipeline

This architecture promotes a **modular design** that supports:

Feature	Description
Secrets Management	Use Jenkins Credentials Plugin or HashiCorp Vault
Test Automation	pytest, unittest, nose, etc. in CI
Reusable Templates	Declarative pipelines or shared GitHub Actions
Multi Cloud Ready	Can extend pipeline to deploy to AWS, Azure, or GCP

This project demonstrates a secure and observable CI/CD pipeline with GitHub Actions, Jenkins, and monitoring using Prometheus and Grafana , all aligned with modern

**DevSecOps practices** like **SAST**, **quality gates**, and **pipeline security**.

Whether for personal projects, team adoption, or enterprise implementation, this setup is flexible, scalable, and secure.

**Thanks for reviewing this setup**

Feel free to fork , contribute or reach out on [www.linkedin.com/in/anita-nnamdi](https://www.linkedin.com/in/anita-nnamdi)