



**INSTITUTE FOR ADVANCED  
COMPUTING AND  
SOFTWARE DEVELOPMENT  
AKURDI, PUNE**

**DOCUMENTATION ON**

**“Secure CI/CD pipeline  
for an application deployment  
and vulnerability analysis”**

**PG-DITISS August-2024**

**SUBMITTED BY:**

**GROUP NO: 23**

**SAKSHI SAWANT (248445)**

**TEJAL DESHMUKH (248451)**

**MRS. SUSHMA HATTARKI**

**PROJECT GUIDE**

**MR. ROHIT PURANIK**

**CENTRE CO-ORDINATOR**

# ACKNOWLEDGEMENT

I would like to express my sincere gratitude to everyone who contributed to the successful completion of this project, **“Deploying a Secure CI/CD pipeline for an application deployment and vulnerability analysis”**.

First and foremost, I extend my heartfelt thanks to my mentor/guide **Mrs. Sushma Hattarki** for their invaluable guidance, support, and constructive feedback throughout the project. Their expertise in DevOps and continuous encouragement played a crucial role in shaping this work.

I am also grateful to my institution Institute for Advanced Computing and Software Development and faculty members for providing the necessary resources, knowledge, and motivation to undertake this project. Their insights and suggestions helped me refine my approach and enhance the overall quality of my work.

Additionally, I would like to acknowledge the support of my peers and colleagues, who provided valuable discussions and shared knowledge, making this learning experience more enriching.

Lastly, I extend my deepest appreciation to my family and friends for their constant encouragement and support throughout this journey. Their belief in my abilities kept me motivated and focused on achieving my goals.

Thank you all for your contributions and support in making this project a success.

**SAKSHI SAWANT (248445)**

**TEJAL DESHMUKH (248451)**

# ABSTRACT

A secure CI/CD pipeline is crucial for modern application deployment, integrating security at every stage of the software development lifecycle. This abstract explores the implementation of a robust CI/CD pipeline leveraging GitHub for version control, Jenkins for automation, SonarQube for static code analysis, and Docker for containerization, enhanced by security tools for log analysis. The pipeline begins with code commits to GitHub, triggering automated builds and tests in Jenkins. SonarQube is integrated into the build phase to perform static code analysis, identifying potential vulnerabilities early in the development cycle. Docker is then used to containerize the application, ensuring consistency across different environments. Security tools, including Intrusion Detection Systems (IDS) and the ELK stack (Elasticsearch, Logstash, Kibana), are employed to analyze logs, detect anomalies, and provide real-time monitoring. Implementing Source Composition Analysis (SCA) and Static Application Security Testing (SAST) further fortifies the pipeline by scanning for vulnerabilities in open-source components and the application code itself. Access control, based on the principle of least privilege, is applied to limit user permissions and reduce the risk of security breaches. This multi-layered approach ensures that applications are deployed rapidly and securely, with continuous monitoring and vulnerability analysis providing ongoing protection against potential threats.

# INDEX

<b>Sr.No.</b>	<b>Title</b>	<b>Page No.</b>
<b>1.</b>	<b>Introduction</b> 1.1 Problem Statement	<b>1</b> <b>2</b>
<b>2.</b>	<b>Methodology</b>	<b>3</b>
<b>3.</b>	<b>Requirement Specification</b> 3.1 Hardware Requirements 3.2 Software Requirements 3.3 Detailed Explanation	<b>4</b> <b>4</b> <b>5</b>
<b>4.</b>	<b>Implementation</b>	<b>12</b>
<b>5.</b>	<b>Application</b>	<b>17</b>
<b>6.</b>	<b>Vulnerability Scanning</b>	<b>27</b>
<b>7.</b>	<b>Advantages And Limitations</b>	<b>29</b>
<b>8.</b>	<b>Conclusion</b>	<b>33</b>
<b>9.</b>	<b>References</b>	<b>34</b>

## ABBREVIATIONS

Sr. No.	Abbreviation	Full Form
1.	IDS	Intrusion Detection System
2.	ELK	Elasticsearch Logstash Kibana
3.	CI/CD	Continuous Integration/Continuous Deployment
4.	IPS	Intrusion Prevention System
5.	SAST	Static Application Security Testing
6.	DNS	Domain Name System

## PORTS

Sr. No.	Service	Port Number
1.	Jenkins	8080
2.	SonarQube	9000
3.	Elasticsearch	9200
4.	Logstash	5044
5.	Kibana	5601
6.	Squid	3128
7.	DNS	53
8.	HTTP	80

## 1. INTRODUCTION

In modern software development, a secure Continuous Integration/Continuous Delivery (CI/CD) pipeline is essential for rapid and reliable application deployment while maintaining robust security. CI/CD security involves implementing security measures throughout the CI/CD pipeline to ensure secure development, testing, and deployment of applications. This introduction explores the critical aspects of building a secure CI/CD pipeline, emphasizing vulnerability analysis and risk mitigation at each stage. The pipeline integrates various tools and practices, starting with GitHub as a Version Control System (VCS) for source code management, ensuring that all code changes are tracked and managed securely. Jenkins is employed for automating the build, test, and deployment processes, streamlining the software delivery lifecycle. SonarQube is incorporated for static code analysis, identifying potential vulnerabilities and code quality issues early in the development cycle. Docker is utilized for containerization, providing a consistent and isolated environment for applications, enhancing portability and security.

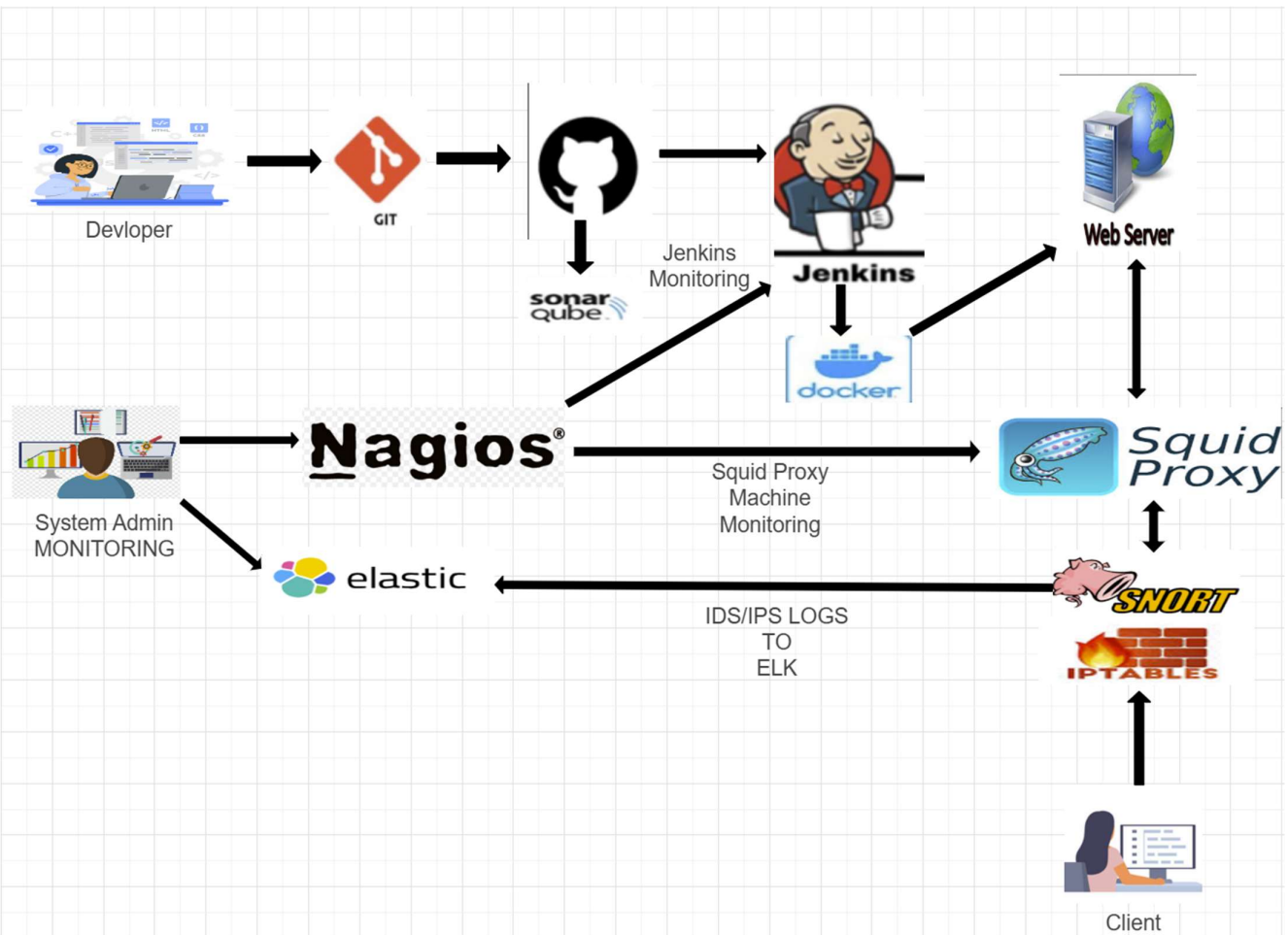
To bolster security further, the pipeline includes security tools for log analysis, such as Intrusion Detection Systems (IDS) and the ELK stack (Elasticsearch, Logstash, Kibana), enabling real-time monitoring and threat detection. Source Composition Analysis (SCA) is integrated to scan for vulnerabilities in open-source components and dependencies, while Static Application Security Testing (SAST) identifies vulnerabilities within the application code itself. Implementing robust access controls and secure secrets management ensures that sensitive information is protected throughout the pipeline. By embedding security at every stage, from code commit to production, this approach ensures that applications are deployed swiftly and securely, with continuous monitoring and vulnerability analysis providing ongoing protection against potential threats.

## 1.1 Problem Statement

CI/CD pipelines are vulnerable to security breaches, making them attractive targets for malicious actors. A compromised pipeline can lead to the injection of malicious code, resulting in system compromise and operational disruptions. For a web application like a hash calculator, ensuring a secure CI/CD pipeline is crucial to protect against potential threats. Integrating security measures such as static code analysis with SonarQube, container scanning, and log analysis using tools like IDS and ELK is essential. This problem statement addresses the need for a robust, secure CI/CD pipeline that incorporates vulnerability analysis to safeguard web applications from code commit to production.

## 2. METHODOLOGY

### Work Flow :





### 3. REQUIREMENT SPECIFICATIONS

#### 3.1 Hardware Requirements

- ✓ **RAM** : 8GB
- ✓ **Storage** : 100GB
- ✓ **CPU** : Intel Core ULTRA 5 or 7

#### 3.2 Software Requirements

- ✓ **Operating System** : Windows 10 or 11
- ✓ **Version Control System** : Github
- ✓ **Code Quality Checker** : SonarQube
- ✓ **Containerization Tool** : Docker
- ✓ **Automation Tool** : Jenkins
- ✓ **Linux Firewall** : IPTABLES
- ✓ **Monitoring System** : Nagios
- ✓ **IDS** : Snort
- ✓ **Log Analysis** : ELK Stack

### 3.3 Detailed explanation:

#### 1. Version Control System: GitHub

GitHub is a cloud-based platform that uses **Git**, a distributed version control system, to manage source code. It helps teams collaborate, track changes, and manage software development efficiently.

##### Key Features:

1. **Repository Management** – Stores code and tracks changes.
2. **Branching & Merging** – Allows multiple developers to work on different features and merge them later.
3. **Pull Requests & Code Reviews** – Enables collaborative development.
4. **CI/CD Integration** – Supports automation with GitHub Actions.
5. **Security & Compliance** – Provides security checks and vulnerability scanning.

##### Use Cases:

- ✓ Open-source projects.
- ✓ Enterprise software development.
- ✓ CI/CD pipelines integration.

#### 2. SonarQube: Static Code Analysis

SonarQube is an open-source platform used for **static code analysis** to identify bugs, security vulnerabilities, and code smells.

**Key Features:**

1. **Code Quality Checks** – Detects issues like memory leaks, dead code, and complex logic.
2. **Security Vulnerability Detection** – Identifies security flaws using OWASP and CWE standards.
3. **Integration with CI/CD** – Works with Jenkins, GitHub, GitLab, and Bitbucket.
4. **Multi-Language Support** – Supports Java, Python, C++, JavaScript, etc.

**Use Cases:**

- ✓ Enforcing coding standards.
- ✓ Identifying potential security risks.
- ✓ Improving software maintainability.

**3. GitHub for Automation**

GitHub supports automation through **GitHub Actions**, which allows users to create CI/CD workflows.

**Key Features:**

1. **Automated Builds & Tests** – Runs unit tests and builds applications automatically.
2. **Deployment Automation** – Deploys applications to production using CI/CD pipelines.
3. **Code Scanning & Security** – Automates security scanning.
4. **Event-driven Triggers** – Triggers workflows based on actions like push, pull requests, or issue creation.

**Use Cases:**

- ✓ Automating CI/CD pipelines.
- ✓ Running security and quality checks.
- ✓ Managing infrastructure as code.

**4. Docker: Containerization Platform**

Docker is a **containerization tool** that packages applications with their dependencies to ensure consistency across different environments.

**Key Features:**

1. **Lightweight Containers** – Isolates applications and runs them independently.
2. **Portability** – Runs anywhere (Windows, Linux, macOS).
3. **Docker Compose** – Manages multi-container applications.
4. **Scalability** – Works with Kubernetes for container orchestration.

**Use Cases:**

- ✓ Creating microservices.
- ✓ Deploying applications in a scalable manner.
- ✓ Running applications in different environments.

**5. Web Server**

A web server handles HTTP requests and serves static or dynamic content.

**Popular Web Servers:**

1. **Apache HTTP Server** – Open-source, widely used.

2. **Nginx** – High-performance, lightweight, and can act as a reverse proxy.
3. **Microsoft IIS** – Used in Windows environments.

**Key Features:**

1. Handles HTTP/HTTPS requests.
2. Serves static and dynamic content.
3. Supports load balancing and caching.

**Use Cases:**

- ✓ Hosting websites and applications.
- ✓ Reverse proxy setup for load balancing.
- ✓ API gateway.

**6. Squid Proxy**

Squid is a **caching and forwarding web proxy** that improves web performance and security.

**Key Features:**

1. **Caching** – Reduces bandwidth usage by storing frequently accessed content.
2. **Content Filtering** – Blocks malicious sites or unwanted content.
3. **Access Control** – Restricts or allows specific users or IPs.
4. **Load Balancing** – Distributes network traffic efficiently.

**Use Cases:**

- ✓ Improving network performance.
- ✓ Enhancing security in corporate environments.

- ✓ Blocking access to unwanted content.

## 7. Iptables: Linux Firewall

Iptables is a command-line firewall utility in Linux that controls network traffic based on rules.

### Key Features:

1. **Packet Filtering** – Controls incoming and outgoing traffic.
2. **NAT (Network Address Translation)** – Modifies packet source/destination addresses.
3. **Logging & Monitoring** – Tracks network traffic for security analysis.

### Use Cases:

- ✓ Securing Linux servers.
- ✓ Preventing unauthorized access.
- ✓ Implementing custom firewall rules.

## 8. Nagios: Monitoring System

Nagios is an open-source **IT infrastructure monitoring tool** that ensures systems, applications, and services are running smoothly.

### Key Features:

1. **Real-time Monitoring** – Tracks server health, network status, and applications.
2. **Alerts & Notifications** – Sends alerts via email, SMS, or webhook.
3. **Custom Plugins** – Extensible via scripts.
4. **Performance Graphing** – Visualizes system metrics.

**Use Cases:**

- ✓ Monitoring server uptime.
- ✓ Detecting network failures.
- ✓ Performance optimization.

**9. Snort: Intrusion Detection System (IDS)**

Snort is an open-source **Intrusion Detection System (IDS)** that detects and prevents cyber threats.

**Key Features:**

1. **Real-time Traffic Analysis** – Monitors network packets.
2. **Signature-based Detection** – Uses rules to identify attacks.
3. **Packet Logging** – Records malicious traffic for analysis.

**Use Cases:**

- ✓ Detecting intrusion attempts.
- ✓ Preventing denial-of-service (DoS) attacks.
- ✓ Monitoring network traffic for anomalies.

**10. ELK Stack: Log Analysis**

The ELK stack (Elasticsearch, Logstash, Kibana) is used for **log management and analysis**.

**Key Components:**

1. **Elasticsearch** – Stores and indexes log data.

**2. Logstash** – Collects, processes, and transforms logs.

**3. Kibana** – Visualizes log data in dashboards.

**Use Cases:**

- ✓ Centralized log management.
- ✓ Security event monitoring.
- ✓ Real-time data analytics.

These tools and technologies play a crucial role in **DevOps, security, and IT infrastructure management**:

- ✓ **GitHub** for version control and automation.
- ✓ **SonarQube** for static code analysis.
- ✓ **Docker** for containerization.
- ✓ **Web Servers & Proxies** (Squid) for traffic control.
- ✓ **Iptables & Snort** for security.
- ✓ **Nagios** for monitoring.
- ✓ **ELK Stack** for log analysis.



## 4. IMPLEMENTATION

### Step 1: Developer Pushes Code to GitHub (Version Control System)

- ✓ Initialize Git Repository: The developer sets up a Git repository for the Hashify(Hash calculator) project.
- ✓ Clone the Repository: The developer clones the repository from GitHub to their local system.
- ✓ Develop and Modify Code: The developer writes or modifies the source code for Hashify(Hash calculator).
- ✓ Commit Changes: The developer commits the changes using meaningful commit messages.
- ✓ Push to Remote Repository: The updated code is pushed to GitHub.

### Step 2: Static Code Analysis Using SonarQube

- ✓ Set Up SonarQube Server: Deploy and configure SonarQube on a dedicated server or container.
- ✓ Create a Project in SonarQube: Define the project to analyze.
- ✓ Integrate SonarQube with GitHub Actions:
  - Configure a SonarQube token in GitHub secrets.
  - Define a GitHub Actions workflow to trigger SonarQube analysis on push.
- ✓ Run SonarQube Analysis: The GitHub Actions workflow runs SonarQube, scanning the code for bugs, vulnerabilities, and code smells.
- ✓ Review Reports: Developers review the SonarQube dashboard for issues and fix them if needed.

## Step 3: Automating Builds and Tests Using GitHub Actions & Jenkins

### GitHub Actions Integration

- ✓ Define Workflow in GitHub Actions: Create a workflow file in `.github/workflows/`.
- ✓ Set Up Triggers: The workflow triggers on code pushes, pull requests, or scheduled runs.
- ✓ Install Dependencies & Run Tests: The pipeline installs required dependencies and runs automated tests.
- ✓ Trigger Jenkins Build: The workflow triggers a Jenkins pipeline for further automation.

### Jenkins CI/CD Pipeline

- ✓ Install & Configure Jenkins: Set up Jenkins on a dedicated server or container.
- ✓ Install Necessary Plugins: Install Git, Docker, SonarQube, and other necessary plugins in Jenkins.
- ✓ Create a Jenkins Pipeline Job:
  - Clone the latest code from GitHub.
  - Run SonarQube analysis.
  - Build a Docker image.
  - Push the image to a container registry.
  - Deploy the container to the web server.
- ✓ Trigger Pipeline Execution: The pipeline runs automatically on code changes.

**Step 4: Containerizing the Application Using Docker**

- ✓ Write a Dockerfile: Define the base image, dependencies, and application start command.
- ✓ Build Docker Image: The CI/CD pipeline builds a Docker image of Hashify.
- ✓ Push to Docker Registry: The built image is pushed to a container registry (Docker Hub or a private registry).
- ✓ Deploy the Container: The pipeline pulls the latest image and runs it on a server.

**Step 5: Deploying on a Web Server**

- ✓ Set Up Web Server: Install and configure NGINX or Apache.
- ✓ Deploy Docker Container: Run the containerized application on the web server.
- ✓ Configure Domain & SSL: Set up a domain name and secure it with SSL (Let's Encrypt or other providers).

**Step 6: Configuring Squid Proxy as a Reverse Proxy**

- ✓ Install Squid Proxy: Deploy Squid on a separate server or container.
- ✓ Configure Squid Proxy:
  - Set up Squid to act as a reverse proxy for the Hashify application.
  - Define access control rules to filter and allow specific requests.
- ✓ Enable Logging: Configure Squid to log incoming requests for monitoring and security purposes.

## Step 7: Implementing Iptables for Traffic Analysis

- ✓ Define Firewall Rules:
  - Allow only necessary traffic (HTTP, HTTPS, SSH).
  - Block unwanted traffic.
  - Set up rate limiting to prevent DDoS attacks.
- ✓ Monitor Traffic: Enable logging in Iptables to analyze incoming and outgoing traffic.
- ✓ Automate Iptables Rules: Configure Iptables rules to apply dynamically based on security needs.

## Step 8: Monitoring Machines Using Nagios

- ✓ Install & Configure Nagios: Deploy Nagios on a monitoring server.
- ✓ Add Hosts & Services: Monitor Hashify's web server, database, proxy, and application services.
- ✓ Set Up Alerts: Configure email or SMS alerts for server downtime and performance issues.
- ✓ Create Dashboards: Use Nagios to visualize system health and resource usage.

## Step 9: Intrusion Detection with Snort

- ✓ Install Snort: Deploy Snort on the network monitoring server.
- ✓ Define IDS Rules: Configure Snort to detect SQL injections, brute-force attempts, and other malicious activities.
- ✓ Set Up Logging & Alerts: Store logs of suspicious activity and trigger alerts.

- ✓ Integrate with ELK: Send Snort logs to ELK for real-time analysis.

### **Step 10: Log Analysis Using ELK Stack**

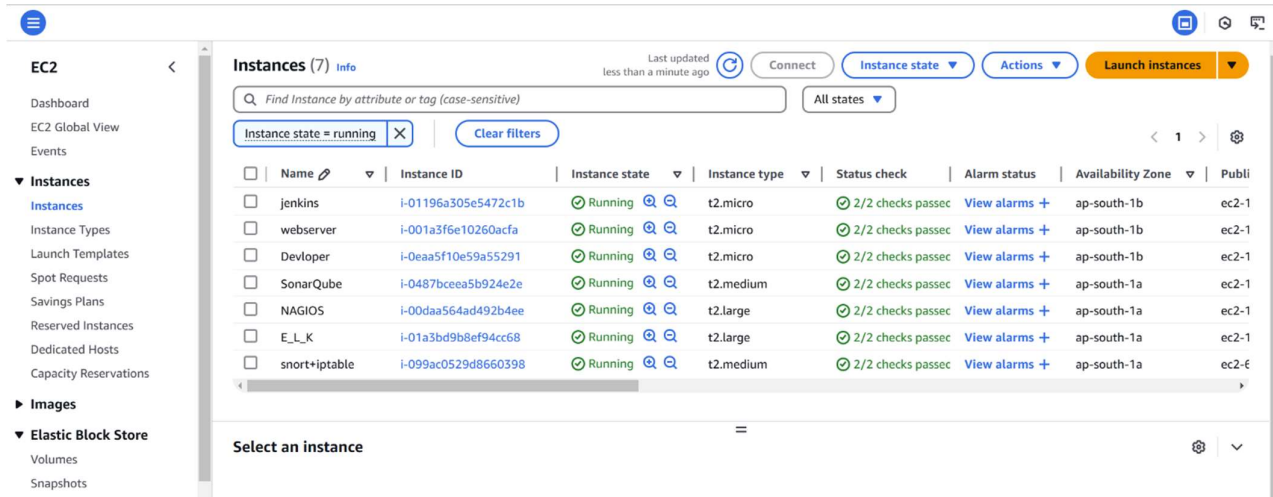
- ✓ Deploy ELK Stack: Set up Elasticsearch, Logstash, and Kibana on a dedicated server.
- ✓ Configure Log Forwarding:
  - Collect logs from the web server, Squid Proxy, Iptables, Nagios, and Snort.
  - Use Logstash to filter and process logs.
- ✓ Create Kibana Dashboards: Visualize logs, security events, and system metrics in real time.
- ✓ Set Up Alerts: Configure alerts for critical security incidents and performance issues.

### **Final Outcome**

- ✓ Developers push code to GitHub → GitHub Actions triggers CI/CD → Jenkins automates builds & deployments → Docker containerizes the app → Web server hosts the app → Squid Proxy manages traffic → Iptables analyzes traffic → Nagios monitors servers → Snort detects intrusions → ELK provides log analysis & insights.

## 5. APPLICATION

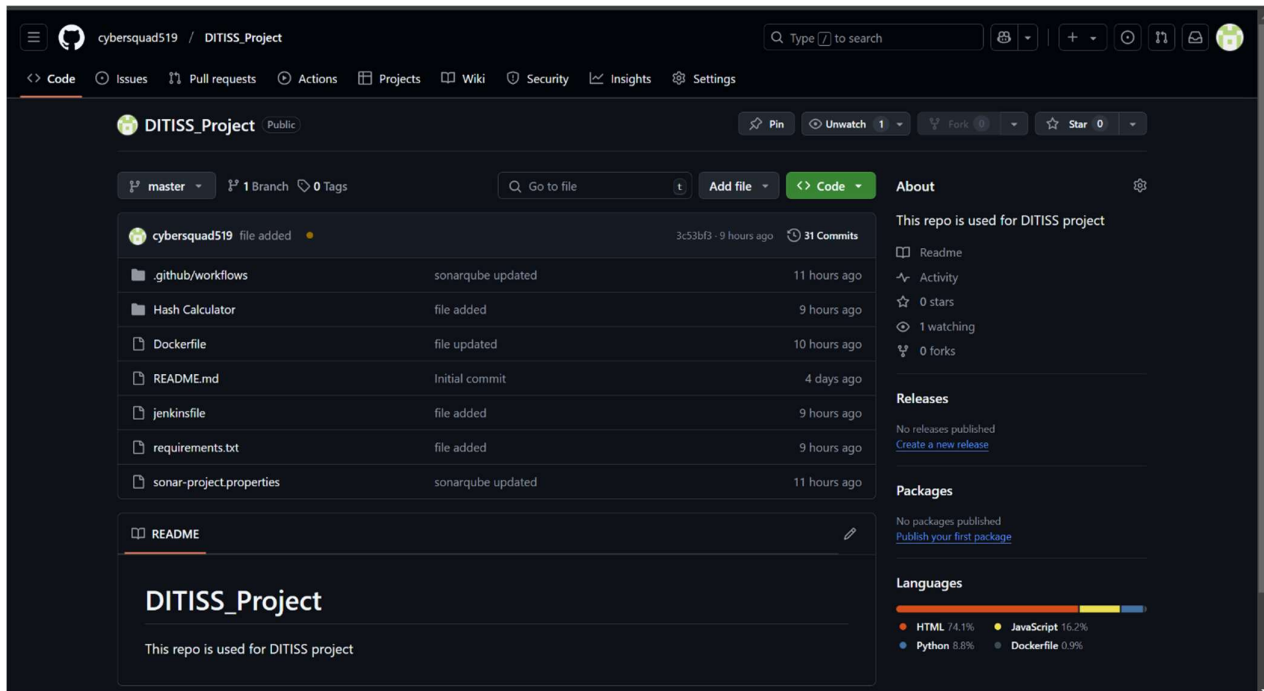
### 1. Machines in AWS



The screenshot shows the AWS Management Console for EC2 Instances. The left sidebar contains navigation options like Dashboard, EC2 Global View, Events, and a list of instance types. The main content area shows a table of 7 instances, all in the 'Running' state. The table columns include Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IP. Below the table, there is a 'Select an instance' section.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
jenkins	i-01196a305e5472c1b	Running	t2.micro	2/2 checks passed	View alarms +	ap-south-1b	ec2-1
webserver	i-001a3f6e10260acfa	Running	t2.micro	2/2 checks passed	View alarms +	ap-south-1b	ec2-1
Developer	i-0eaa5f10e59a55291	Running	t2.micro	2/2 checks passed	View alarms +	ap-south-1b	ec2-1
SonarQube	i-0487bceea5b924e2e	Running	t2.medium	2/2 checks passed	View alarms +	ap-south-1a	ec2-1
NAGIOS	i-00daa564ad492b4ee	Running	t2.large	2/2 checks passed	View alarms +	ap-south-1a	ec2-1
E_L_K	i-01a3bd9b8ef94cc68	Running	t2.large	2/2 checks passed	View alarms +	ap-south-1a	ec2-1
snort+iptables	i-099ac0529d8660398	Running	t2.medium	2/2 checks passed	View alarms +	ap-south-1a	ec2-1

### 2. Github Repository



The screenshot shows the GitHub repository page for 'DITISS\_Project' by user 'cybersquad519'. The repository is public and has 31 commits. The file structure includes .github/workflows, Hash Calculator, Dockerfile, README.md, jenkinsfile, requirements.txt, and sonar-project.properties. The repository is used for the DITISS project.

**Repository Details:**

- Repository: DITISS\_Project
- User: cybersquad519
- Status: Public
- Commits: 31
- Branches: 1 (master)
- Tags: 0

**Files:**

- .github/workflows: sonarqube updated (11 hours ago)
- Hash Calculator: file added (9 hours ago)
- Dockerfile: file updated (10 hours ago)
- README.md: Initial commit (4 days ago)
- jenkinsfile: file added (9 hours ago)
- requirements.txt: file added (9 hours ago)
- sonar-project.properties: sonarqube updated (11 hours ago)

**About:**

- This repo is used for DITISS project
- Readme: 0 stars
- Activity: 1 watching
- Forks: 0 forks

**Releases:**

- No releases published
- Create a new release

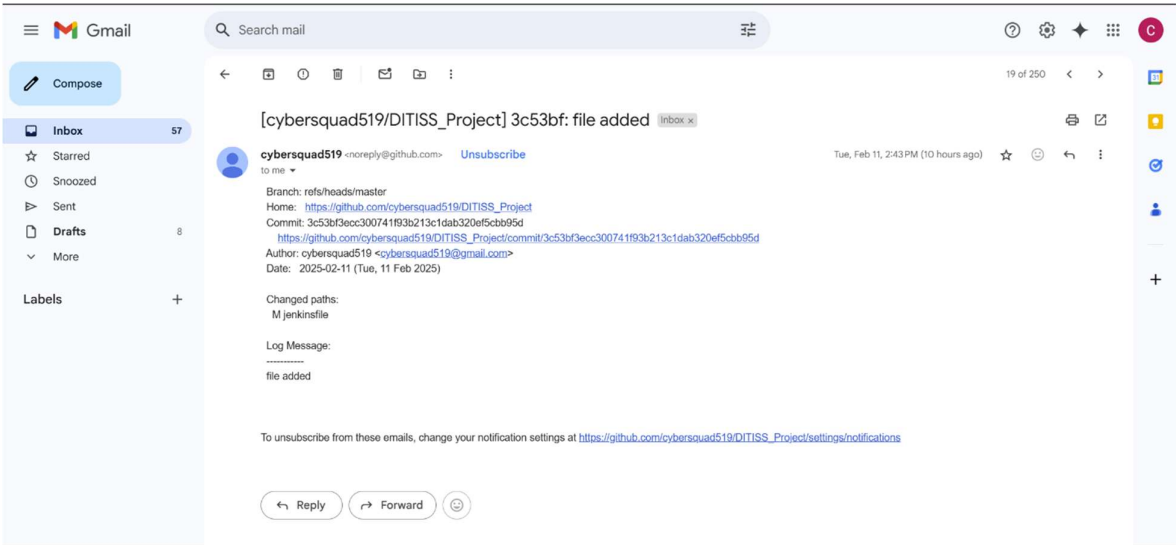
**Packages:**

- No packages published
- Publish your first package

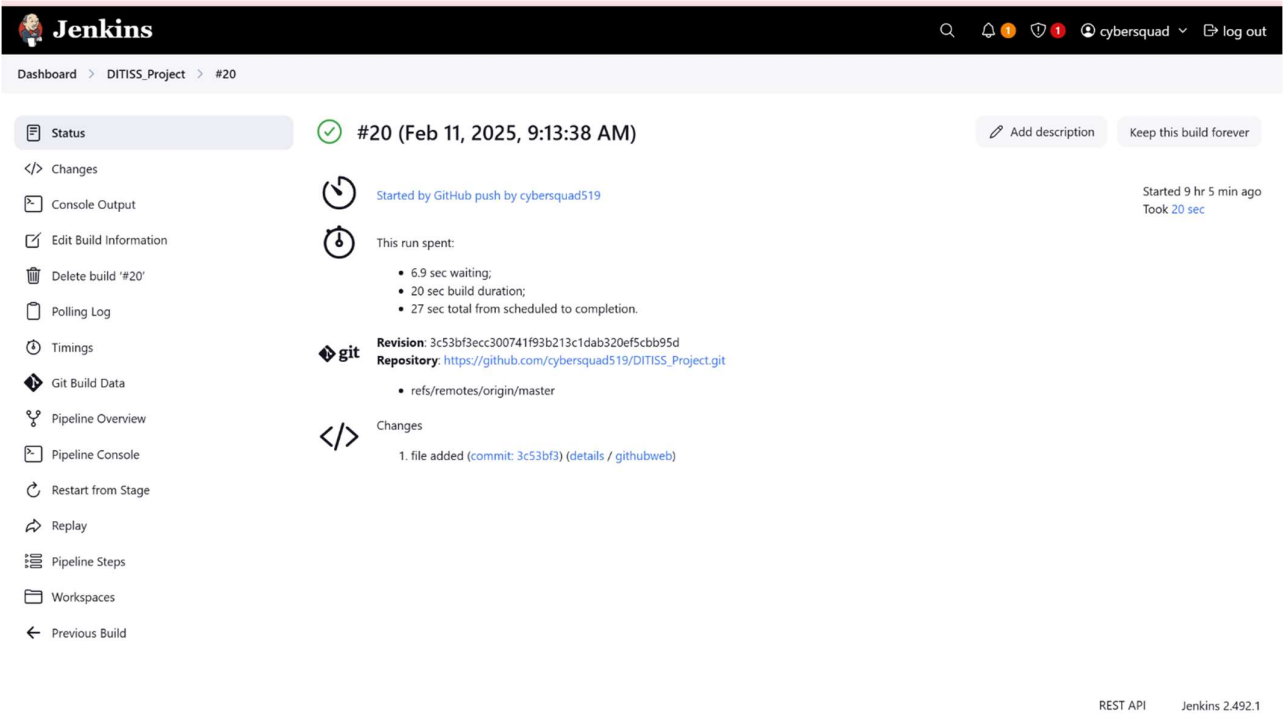
**Languages:**

- HTML: 4.1%
- JavaScript: 16.2%
- Python: 8.6%
- Dockerfile: 0.9%

2.1 Github push successful email



3. Jenkins Dashboard



## 4. CI/CD pipeline in Jenkins

The screenshot shows the Jenkins web interface for a pipeline named 'DITISS\_Project'. The top navigation bar includes the Jenkins logo, a search icon, notification icons, a user profile for 'cybersquad', and a 'log out' button. The breadcrumb trail is 'Dashboard > DITISS\_Project > #20 > Pipeline Overview'. The main section is titled 'Build #20' with a green checkmark icon. To the right of the title are buttons for 'Rebuild', 'Console', and 'Configure'. Below the title is a 'Pipeline' diagram showing a sequence of steps: Start, Checkout SCM, Checkout, Build Docker Im..., Push Docker Im..., Deploy to Web ..., and End. All steps are marked with green checkmarks, indicating successful completion. To the right of the pipeline diagram is a 'Details' box containing the following information: 'Started 9 hr 3 min ago', 'Queued 6.9 sec', and 'Took 20 sec'.

## 5. Docker Image File

```
FROM python:3.11-slim-buster
WORKDIR /Hash Calculator
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
COPY . /app
CMD ["python3", "app.py"]
```

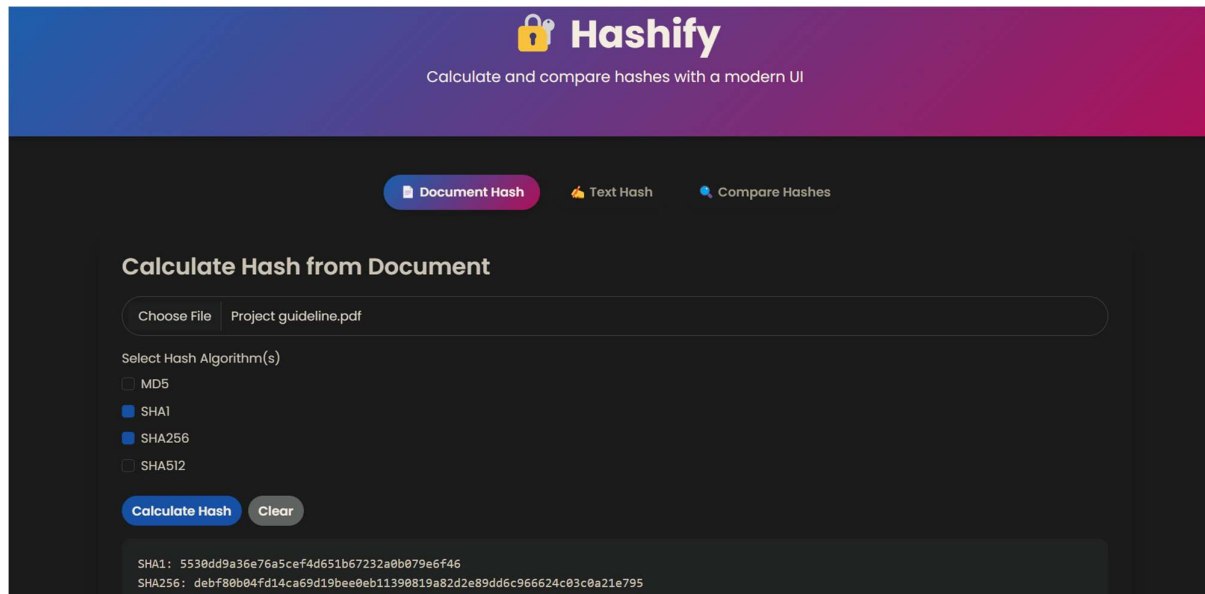
## 6. Docker Image requirements

```
Flask==3.0.2
itsdangerous==2.1.2
Jinja2==3.1.3
MarkupSafe==2.1.5
werkzeug==3.1.0
```



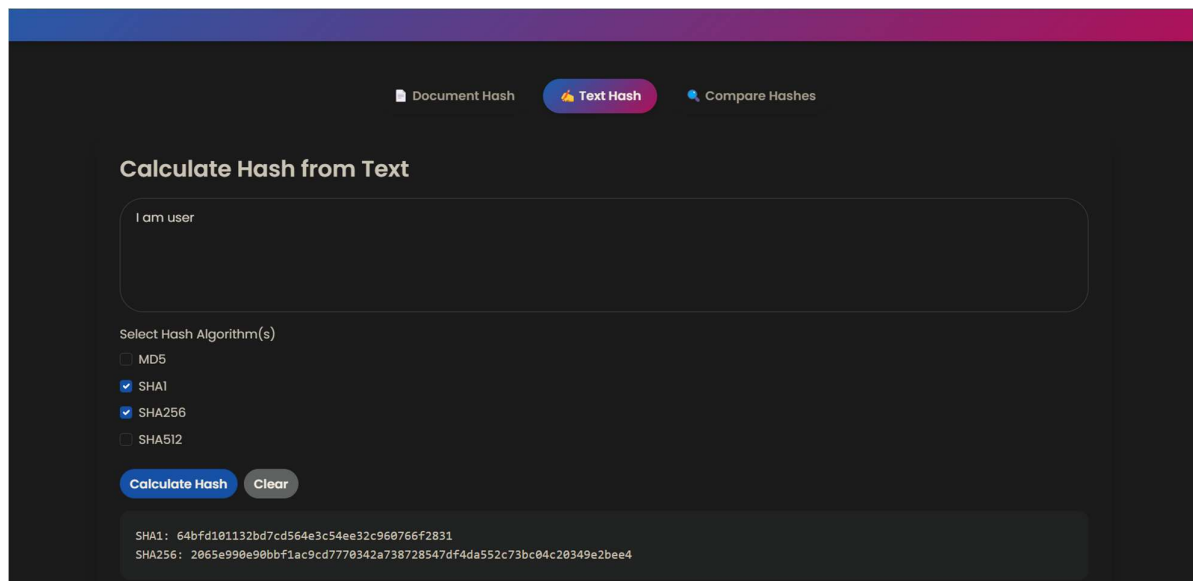
## 7. Hashify(Hash Calculator) web application

### 7.1 Calculating hash of document



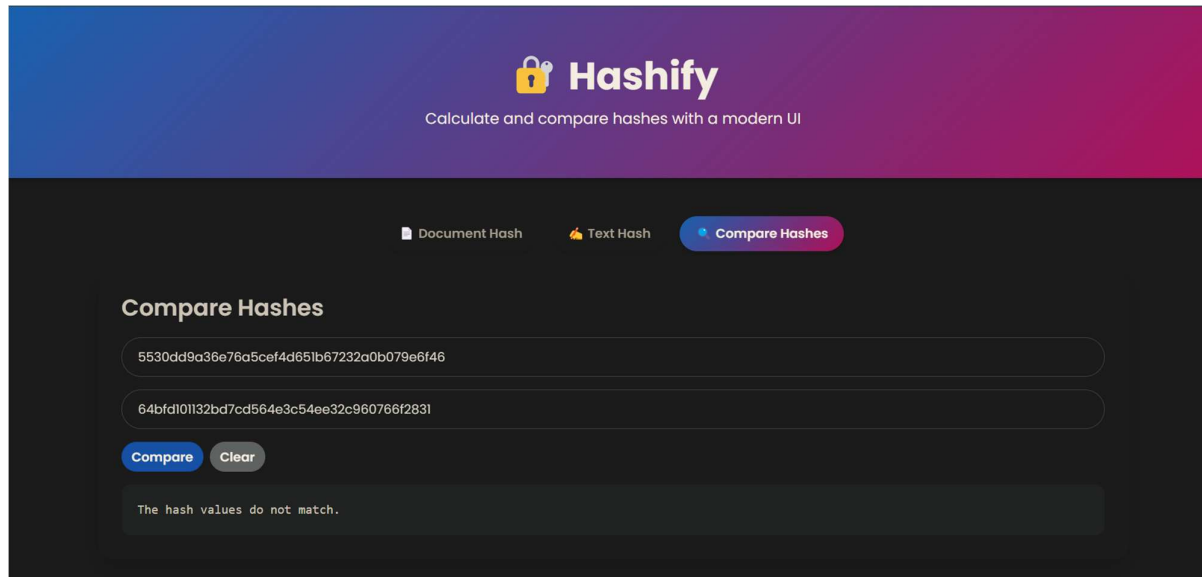
The screenshot shows the Hashify web application interface. At the top, there's a header with the Hashify logo and the tagline "Calculate and compare hashes with a modern UI". Below the header, there are three tabs: "Document Hash" (selected), "Text Hash", and "Compare Hashes". The main section is titled "Calculate Hash from Document". It features a "Choose File" button and a text input field containing "Project guideline.pdf". Below this, there's a section for "Select Hash Algorithm(s)" with checkboxes for MD5, SHA1 (checked), SHA256, and SHA512. At the bottom of this section are "Calculate Hash" and "Clear" buttons. The results are displayed in a dark box at the bottom: SHA1: 5530dd9a36e76a5cef4d651b67232a0b079e6f46 and SHA256: debf80b04fd14ca69d19bee0eb11390819a82d2e89dd6c966624c03c0a21e795.

### 7.2 Calculating hash of text

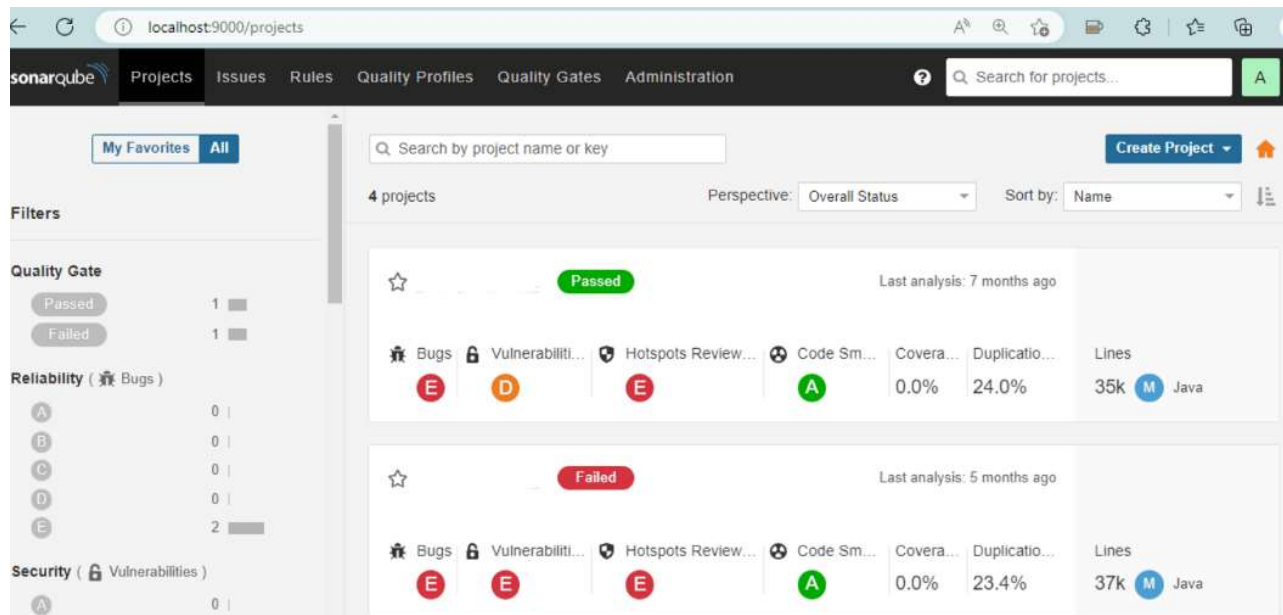


The screenshot shows the Hashify web application interface with the "Text Hash" tab selected. The main section is titled "Calculate Hash from Text". It features a text input field containing "I am user". Below this, there's a section for "Select Hash Algorithm(s)" with checkboxes for MD5, SHA1 (checked), SHA256 (checked), and SHA512. At the bottom of this section are "Calculate Hash" and "Clear" buttons. The results are displayed in a dark box at the bottom: SHA1: 64bfd101132bd7cd564e3c54ee32c960766f2031 and SHA256: 2065e990e90bbf1ac9cd7770342a738728547df4da552c73bc04c20349e2bee4.

## 7.3 Comparing hash value



## 8. SonarQube Dashboard



## 9. Squid Configuration

```

acl localnet src 0.0.0.1-0.255.255.255 # RFC 1122 "this" network (LAN)
acl localnet src 10.0.0.0/8           # RFC 1918 local private network (LAN)
acl localnet src 100.64.0.0/10        # RFC 6598 shared address space (CGN)
acl localnet src 169.254.0.0/16       # RFC 3927 link-local (directly plugged) machines
acl localnet src 172.16.0.0/12        # RFC 1918 local private network (LAN)
acl localnet src 192.168.0.0/16       # RFC 1918 local private network (LAN)
acl localnet src fc00::/7            # RFC 4193 local private network range
acl localnet src fe80::/10           # RFC 4291 link-local (directly plugged) machines
acl cybersquad dstdomain 172.31.8.87

#Default:
# No peer usage restrictions.
cache_peer_access cyber allow cybersquad
cache_peer_access cyber deny all
# TAG: neighbor_type_domain
# Modify the cache_peer neighbor type when passing requests
# about specific domains to the peer.
#
# Usage:
# neighbor_type_domain neighbor parent|sibling domain domain ...
#

```

## 10. Nagios configuration

```

define host {
    use                linux-server
    host_name          deb_12-1
    alias              Debian 12 Sys 1
    address             192.168.80.139
    max_check_attempts 5;
}

define service {
    use                generic-service
    host_name          deb_12-1
    service_description Agent Version
    check_command       check_ncpa!-t 'Password1234' -p -P 5693 -M system/agent_version
}

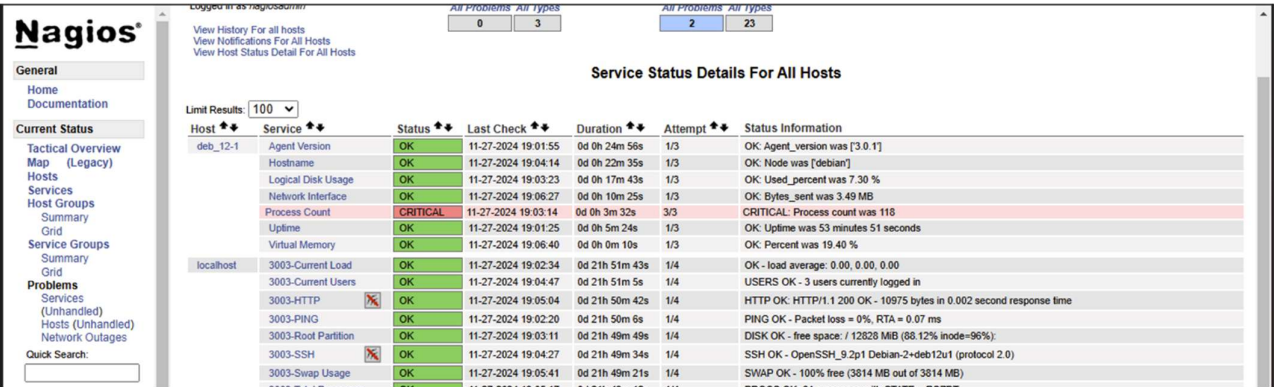
define service {
    use                generic-service
    host_name          deb_12-1
    service_description Hostname
    check_command       check_ncpa!-t 'Password1234' -p -P 5693 -M system/node
}

define service {
    use                generic-service
    host_name          deb_12-1
    service_description Logical Disk Usage
    check_command       check_ncpa!-t 'Password1234' -p -P 5693 -M 'disk/logical/|/used_percent' -w 70 -c 90 -x
}

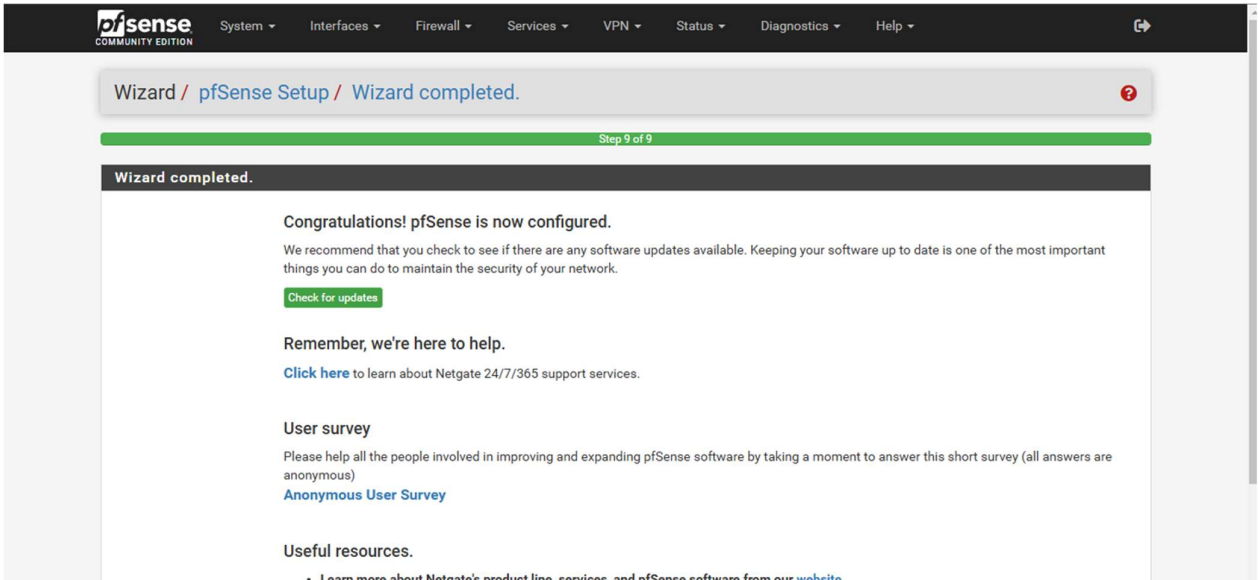
define service {
    use                generic-service
    host_name          deb_12-1
}

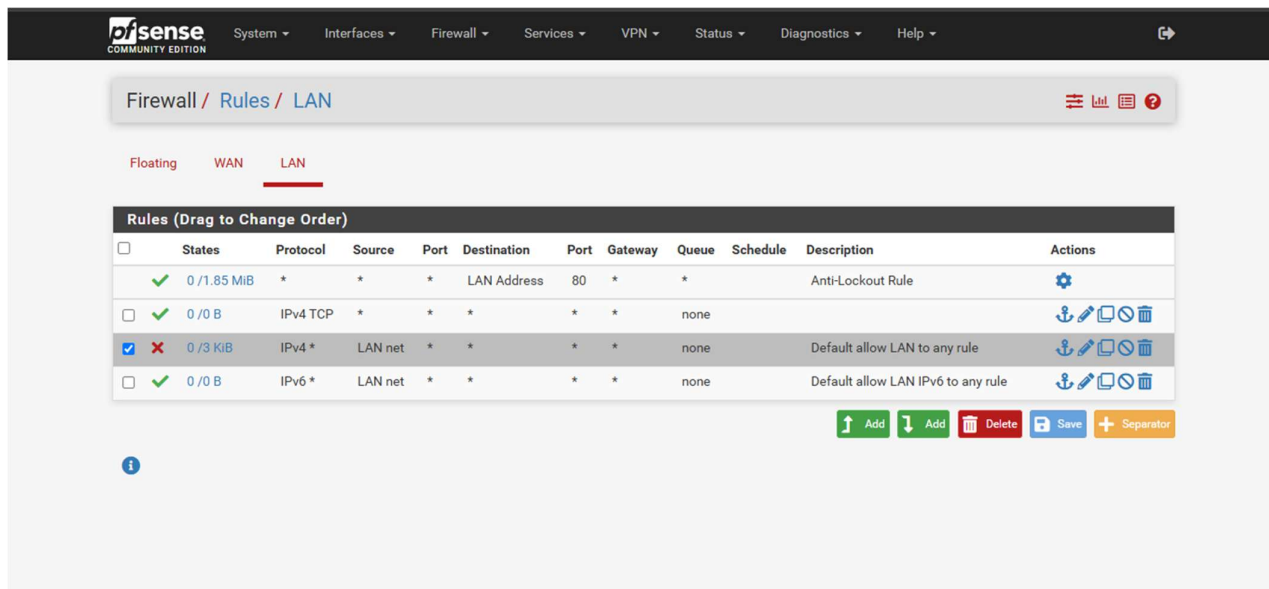
```

11. Nagios Dashboard



12. Pfsense firewall





### 13. Snort configuration

```
# ===== Filebeat inputs =====

filebeat.inputs:

# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input specific configurations.

# filestream is an input for collecting log messages from files.
- type: log

# Unique ID among all inputs, an ID is required.
id: my-filestream-id

# Change to true to enable this input configuration.
enabled: true

# Paths that should be crawled and fetched. Glob based paths.
paths:
  - /var/log/snort/alert
# - /var/log/*.log
#- c:\programdata\elasticsearch\logs\*
```

## 14. Snort logs

```

GNU nano 7.2                                alert
[**] [1:100000001:0] YOUR NOT ALLOWED [**]
[Priority: 0]
02/10-17:28:12.604571 27.59.102.19:45891 -> 172.31.36.241:22
TCP TTL:114 TOS:0x80 ID:21112 IpLen:20 DgmLen:40
***A*** Seq: 0xC6E2BCDF Ack: 0x46AF8CB9 Win: 0xFF TcpLen: 20

[**] [1:100000001:0] YOUR NOT ALLOWED [**]
[Priority: 0]
02/10-17:28:12.634509 27.59.102.19:45891 -> 172.31.36.241:22
TCP TTL:114 TOS:0x80 ID:21113 IpLen:20 DgmLen:40
***A*** Seq: 0xC6E2BCDF Ack: 0x46AFA610 Win: 0xFF TcpLen: 20

[**] [1:100000001:0] YOUR NOT ALLOWED [**]
[Priority: 0]
02/10-17:28:12.649591 27.59.102.19:45891 -> 172.31.36.241:22
TCP TTL:114 TOS:0x80 ID:21114 IpLen:20 DgmLen:40
***A*** Seq: 0xC6E2BCDF Ack: 0x46AFB501 Win: 0xFF TcpLen: 20

[**] [1:100000001:0] YOUR NOT ALLOWED [**]
[Priority: 0]
02/10-17:28:12.649591 27.59.102.19:45891 -> 172.31.36.241:22
TCP TTL:114 TOS:0x80 ID:21115 IpLen:20 DgmLen:40
***A*** Seq: 0xC6E2BCDF Ack: 0x46AFB819 Win: 0xFC TcpLen: 20

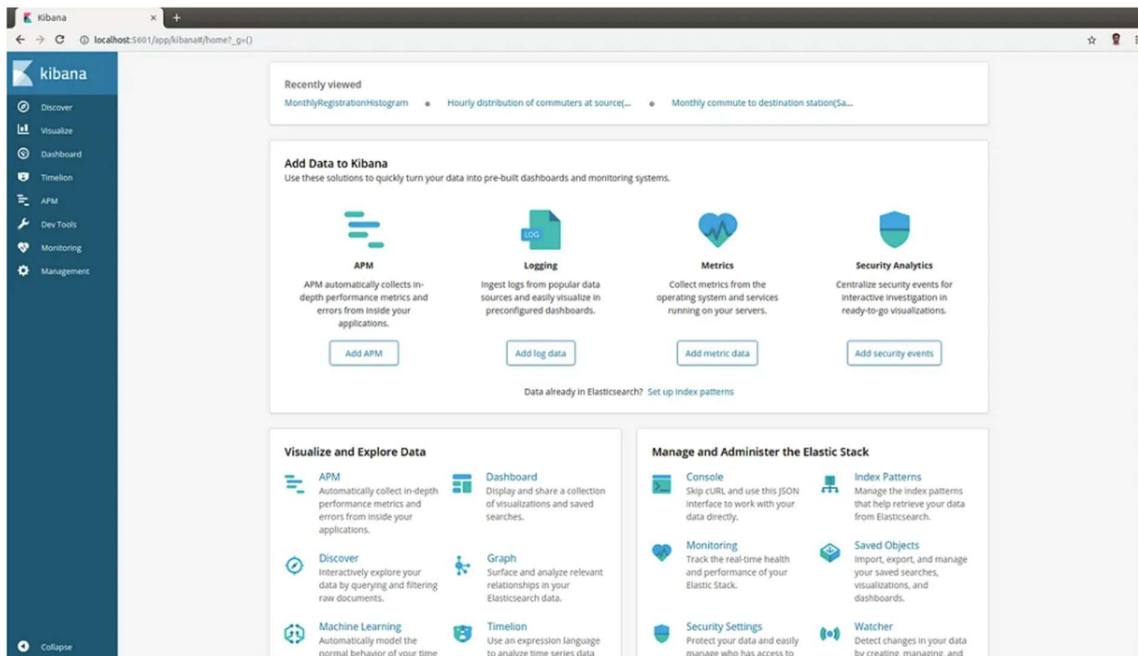
[**] [1:100000001:0] YOUR NOT ALLOWED [**]
[Priority: 0]
02/10-17:28:12.654334 27.59.102.19:45891 -> 172.31.36.241:22
TCP TTL:114 TOS:0x80 ID:21116 IpLen:20 DgmLen:360
***AP*** Seq: 0xC6E2BCDF Ack: 0x46AFB819 Win: 0xFC TcpLen: 20

[**] [1:100000001:0] YOUR NOT ALLOWED [**]
[Priority: 0]
02/10-17:28:12.699249 27.59.102.19:45891 -> 172.31.36.241:22
TCP TTL:114 TOS:0x80 ID:21117 IpLen:20 DgmLen:40
***A*** Seq: 0xC6E2BE1F Ack: 0x46AFC76A Win: 0xFF TcpLen: 20

[**] [1:100000001:0] YOUR NOT ALLOWED [**]
[Priority: 0]

```

## 15. ELK(Elasticsearch , Logstash , Kibana)



## 16. Snort and ELK integration

```
input {
  beats {
    port => 5044
  }
}

filter {
  if [fields][type] == "snort" {
    grok {
      match => { "message" => "%{SYSLOGTIMESTAMP:timestamp} %{HOSTNAME:hostname} snort\[ %{POSINT:pid}\]: \[%{[a-zA-Z0-9_+@-]*}]" }
    }
    date {
      match => [ "timestamp", "MMM d HH:mm:ss", "MMM dd HH:mm:ss" ]
      target => "@timestamp"
    }

    # Example of additional parsing based on Snort message content
    if [message] =~ "YOUR NOT ALLOWED" {
      mutate {
        add_field => { "alert_severity" => "high" }
      }
    } else if [message] =~ "WEB-CLIENT" {
      mutate {
        add_field => { "alert_category" => "web_client" }
      }
    }
  }
}

output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "snort_logs-%{+YYYY.MM.dd}"
  }
  stdout {
    codec => rubydebug
  }
}
```



## 6. VULNERABILITY SCANNING

### **Vulnerability Scanning in CI/CD Pipeline for Hashify (Hash Calculator) Deployment:**

Implementing vulnerability scanning at multiple stages of the CI/CD pipeline is crucial to ensure the security of the Hashify web application. Here's how to systematically perform vulnerability scanning across the pipeline:

#### **1. Source Code Security & Dependency Scanning in GitHub (Version Control)**

- **Pre-commit Hooks:**

- ✓ Implement pre-commit **security checks** using tools like git-secrets to prevent sensitive data exposure (API keys, credentials) in commits.

- **SAST (Static Application Security Testing) with SonarQube:**

- ✓ SonarQube performs static code analysis to detect security issues such as SQL injection, cross-site scripting (XSS), buffer overflows, and weak authentication mechanisms.
- ✓ Integrate SonarQube with GitHub Actions to enforce quality gates before merging code into the main branch.

#### **2. CI/CD Pipeline Vulnerability Scanning with GitHub Actions & Jenkins**

- **Security Testing in GitHub Actions:**

- ✓ Include SAST (Static Analysis) and Software Composition Analysis (SCA) tools in GitHub Actions to identify known vulnerabilities (CVEs) in dependencies.

- **Jenkins Pipeline Security Hardening:**

- ✓ Apply role-based access control (RBAC) to limit unauthorized access to Jenkins.
- ✓ Use Jenkins security plugins like OWASP Dependency-Check Plugin to scan project dependencies.



- ✓ Regularly update Jenkins and plugins to patch security vulnerabilities.

### 3. Network & Traffic Security Scanning

- **Squid Proxy Security & Traffic Analysis:**

- ✓ Monitor **access** logs and blocked requests to detect malicious traffic.
- ✓ Implement SSL/TLS inspection to filter encrypted traffic for potential threats.

- **Iptables & Network Firewall Auditing:**

- ✓ Use IPTables Audit Scripts to check for misconfigurations.
- ✓ Implement fail2ban to detect and block repeated unauthorized access attempts.

- **Snort IDS (Intrusion Detection System) Rule Updates & Threat Monitoring:**

- ✓ Regularly update Snort rules to detect the latest attack patterns.
- ✓ Set up Snort alerts to flag suspicious traffic such as port scans, brute-force attempts, and exploit payloads.

### 4. Log Analysis & Security Incident Response with ELK & Nagios

- **Log Monitoring with ELK Stack:**

- ✓ Collect logs from GitHub, Jenkins, Docker, Squid Proxy, Iptables, and Snort using Logstash and Filebeat.
- ✓ Use Kibana dashboards to visualize security threats and detect anomalies.

- **Alerting & Automated Responses:**

- ✓ Configure Elasticsearch Watcher or Nagios alerts to notify security teams about failed logins, high CPU usage, or unauthorized access.
- ✓ Use automated response scripts (e.g., disabling compromised containers, blocking IPs) based on ELK threat intelligence.

## 7. ADVANTAGES AND LIMITATIONS

### Advantages & Limitations of CI/CD Pipeline for Hashify Deployment

Deploying Hashify (Hash Calculator) using GitHub, SonarQube, GitHub Actions, Jenkins, Docker, Squid Proxy, Iptables, Nagios, Snort, and ELK provides several benefits but also comes with some limitations. Below is a structured analysis:

#### 1. Version Control & Static Code Analysis (GitHub & SonarQube)

##### Advantages:

- ✓ Code Versioning & Collaboration: Developers can track changes, rollback versions, and collaborate efficiently using GitHub.
- ✓ Automated Code Quality Checks: SonarQube detects security vulnerabilities, code smells, and bugs before deployment.
- ✓ Early Bug Detection: Ensures high code quality by enforcing coding standards before merging to production.

##### Limitations:

- ✓ False Positives in Static Analysis: SonarQube may generate false alarms, requiring manual validation.
- ✓ Learning Curve: Developers must understand Git workflows and SonarQube rules for efficient use.
- ✓ Performance Overhead: Running SonarQube scans can slow down the CI/CD pipeline if not optimized.

## 2. CI/CD Automation (GitHub Actions & Jenkins)

### Advantages:

- ✓ Continuous Integration & Deployment: Automates build, testing, and deployment processes.
- ✓ Parallel Execution: GitHub Actions and Jenkins support parallel execution, speeding up CI/CD pipelines.
- ✓ Customizable Pipelines: Pipelines can be tailored for different environments (staging, production).

### Limitations:

- ✓ Infrastructure Dependency: Jenkins requires dedicated infrastructure and maintenance.
- ✓ Security Risks: Improperly configured secrets or access controls can expose credentials.
- ✓ Debugging Complexity: Pipeline failures can be difficult to debug due to multiple integration points.

## 3. Containerization & Deployment (Docker & Web Server)

### Advantages:

- ✓ Portability & Scalability: Docker ensures the application runs consistently across different environments.
- ✓ Efficient Resource Utilization: Containers use fewer resources than traditional VMs.
- ✓ Rapid Deployment: New application versions can be deployed instantly using container orchestration.

**Limitations:**

- ✓ Container Security Risks: Vulnerable base images or misconfigured containers can introduce security threats.
- ✓ Storage Overhead: Storing multiple container images increases disk usage.
- ✓ Networking Complexity: Managing inter-container communication and networking security requires expertise.

**4. Network Security & Traffic Analysis (Squid Proxy, Iptables, Snort IDS)****Advantages:**

- ✓ Access Control & Filtering: Squid Proxy restricts access and improves security.
- ✓ Real-time Traffic Analysis: Iptables logs and monitors network traffic for potential threats.
- ✓ Intrusion Detection: Snort IDS detects and alerts against suspicious activities.

**Limitations:**

- ✓ Performance Overhead: High traffic filtering can slow down response times.
- ✓ Rule Maintenance: Snort and Iptables require frequent updates to remain effective.
- ✓ False Positives: IDS alerts may include non-malicious activities, causing alert fatigue.

**5. Monitoring & Log Analysis (Nagios & ELK Stack)****Advantages:**

- ✓ Real-time Monitoring: Nagios provides live tracking of system health and performance.
- ✓ Comprehensive Log Analysis: ELK Stack centralizes logs for easy analysis and visualization.
- ✓ Automated Alerts & Threat Detection: Helps in quick incident response.

**Limitations:**

- ✓ High Resource Usage: ELK requires significant CPU, memory, and storage.
- ✓ Complex Setup & Maintenance: Fine-tuning Nagios and ELK dashboards requires expertise.
- ✓ Delayed Log Processing: Large log volumes can cause delays in real-time analytics.

## 8. CONCLUSION

The integration of GitHub, SonarQube, GitHub Actions, Jenkins, Docker, Web Server, Squid Proxy, Iptables, Nagios, Snort IDS, and ELK Stack in the CI/CD pipeline for Hashify ensures a secure, automated, and efficient deployment process. Version control with GitHub enables collaboration, while SonarQube enhances code security. GitHub Actions and Jenkins streamline automation, reducing manual intervention and ensuring continuous delivery. Docker containerization improves portability and scalability.

For security, Squid Proxy manages traffic, Iptables restricts unauthorized access, and Snort IDS detects intrusions. Nagios provides real-time monitoring, while the ELK Stack centralizes log analysis for security insights. Despite the complexity of managing multiple tools, this approach enhances security and performance. Regular updates and maintenance are essential to sustaining efficiency.

Overall, this robust pipeline ensures Hashify remains scalable, secure, and highly available, offering a reliable user experience while minimizing security risks and operational challenges.

## 9. REFERENCES

1. <https://docs.github.com/en/webhooks>
2. <https://github.com/git-guides#push-your-changes-to-the-remote>
3. <https://medium.com/@amitvsolutions/demystifying-the-elk-stack-a-comprehensive-guide-8da4d96b82be#:~:text=The%20ELK%20stack%20combines%20three%20open%2Dsource%20tools%3A&text=Functionality%3A%20Stores%2C%20indexes%2C%20and,%2C%20metrics%20analysis%2C%20and%20more>
4. <https://medium.com/@imfuzail09/building-a-ci-cd-pipeline-with-github-docker-sonarqube-jenkins-maven-and-nexus-e2591ae10328>
5. <https://chirag0002.hashnode.dev/build-a-cicd-pipeline-using-jenkins-sonarqube-docker-and-aws>
6. <https://www.snort.org/>