Project Report On

DevOps Secure Software Development Lifecycle

Submitted in partial fulfillment for the award of

Post Graduate Diploma in High Performance Computing System Administration from **C-DAC ACTS (Pune)**

Guided by

Mr.Roshan Gami

Presented By

Mr.Shubham Nimkar	**PRN:230340127009**
Mr.Subhashit Sathe	**PRN:230340127011**
Mr.Suraj Bhujbal	**PRN:230340127012**
Ms.Kajol Patil	**PRN:230340127038**
Ms.Manjiri Khedekar	**PRN:230340127039**

Centre of Development of Advanced Computing (C-DAC), Pune

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Mr.Shubham Nimkar	**PRN:230340127009**
Mr.Subhashit Sathe	**PRN:230340127011**
Mr.Suraj Bhujbal	**PRN:230340127012**
Ms.Kajol Patil	**PRN:230340127038**
Ms.Manjiri Khedekar	**PRN:230340127039**

have successfully completed their project on

DevOps Secure Software Development Lifecycle

Under the Guidance of Mr. Roshan Gami

Project Guide **Project Supervisor**

HOD ACTS

Mr. Aditya Sinha

ACKNOWLEDGEMENT

This project **"DevOps Secure Software Development Lifecycle"** was a great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS).

We all are very glad to mention the name of **Mr. Roshan Gami** for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

We are highly grateful to **Mr. Kaushal Sharma** (Manager (ACTS training Centre), C- DAC), for his guidance and support whenever necessary while doing this course **Post Graduate Diploma in High Performance Computing System Administration (PG- DHPCSA)** through C-DAC ACTS, Pune.

Our most heartfelt thank goes to **Ms. Swati Salunkhe** (Course Coordinator, PG-DHPCSA) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Pune.

Mr.Shubham Nimkar	PRN:230340127009
Mr.Subhashit Sathe	PRN:230340127011
Mr.Suraj Bhujbal	PRN:230340127012
Ms.Kajol Patil	PRN:230340127038
Ms.Manjiri Khedekar	PRN:230340127039

TABLE OF CONTENTS

1. Abstract

The increasing number of cyber attacks has made it essential for organizations to prioritize security in their software development process. This project aims to implement the Security framework in DevSecOps to integrate security into the software development process.

The project will involve the following phases:

1. Planning and Analysis: In this phase, security requirements will be identified and analyzed, and security testing tools will be selected.
2. Design and Architecture: In this phase, security controls will be defined and integrated into the software design and architecture.
3. Implementation: In this phase, security controls will be implemented in the code.
4. Testing: In this phase, the software will be tested for security vulnerabilities, and any issues found will be fixed.
5. Deployment: In this phase, the software will be deployed to production.
6. Maintenance: In this phase, the software will be continuously monitored and maintained to ensure that it remains secure.

The project will also involve the following practices:

1. Automation of Security Testing: Automated security testing tools will be integrated into the DevSecOps pipeline to detect vulnerabilities early in the development process.
2. Continuous Integration and Deployment: Continuous integration and deployment practices will be implemented to ensure that the software is continuously tested and deployed in a secure manner.

The expected outcome of the project is to provide a secure software development process that meets the security requirements of the organization. The project will help reduce the risk of security breaches and ensure that the software is developed and deployed securely.

2. Introduction

This project aims to implement the SSDLC framework in DevSecOps to provide a secure software development process that meets the security requirements of the organization. The project will involve the identification and analysis of security requirements, the integration of security controls into the software design and architecture, the implementation of security controls in the code, testing the software for security vulnerabilities, deployment of the software to production, and continuous monitoring and maintenance of the software to ensure that it remains secure. With the increasing number of cyber attacks, security has become a major concern for organizations. The traditional approach of adding security measures as an afterthought to the software development process is no longer effective. Therefore, there is a need for a more proactive approach that integrates security into the software development lifecycle. The Secure Software Development Lifecycle (SSDLC) framework is a process that integrates security measures into the software development lifecycle. DevSecOps, on the other hand, is an approach that combines development, operations, and security teams to automate and integrate security into the software development process.

3. Implementing SSDLC Framework in DevSecOps

SSDLC (Secure Software Development Lifecycle) is a framework for integrating security into the software development process. DevSecOps is a methodology that combines development, security, and operations into a single continuous process. By implementing SSDLC in DevSecOps, you can ensure that security is built into every stage of the software development process.

Here are the steps to implement SSDLC in DevSecOps:

1. Plan: In this stage, you need to define the project scope, objectives, and requirements. You also need to identify potential security risks and define security requirements.
2. Design: In this stage, you need to design the architecture, components, and interfaces of the software system. You also need to define security controls, such as access control, authentication, and encryption.
3. Develop: In this stage, you need to develop the software system and ensure that it meets the security requirements defined in the previous stages. You also need to conduct security testing, such as penetration testing and vulnerability scanning.
4. Test: In this stage, you need to conduct functional and non-functional testing to ensure that the software system meets the requirements and is secure.
5. Deploy: In this stage, you need to deploy the software system in a secure manner. You also need to ensure that the deployment environment meets the security requirements.
6. Operate: In this stage, you need to monitor the software system for security incidents and vulnerabilities. You also need to perform security maintenance, such as patching and updating.
7. Maintain: In this stage, you need to maintain the software system by fixing security vulnerabilities and updating security controls. You also need to ensure that the software system remains secure over time.

By following these steps, you can implement SSDLC in DevSecOps and ensure that security is built into every stage of the software development process.

4. System Requirement

User Interface

1. EC2 Instance Ubuntu 20.0.4 t2.medium

Software Requirement

1. GIT
2. Jenkins
3. Docker
4. SonarQube
5. OWASP ZAP
6. Dependency-checker 8.1.0
7. Apache Maven version- 3.6.3
8. Trufflehog
9. Trivy
10. Grafana

Hardware Requirement

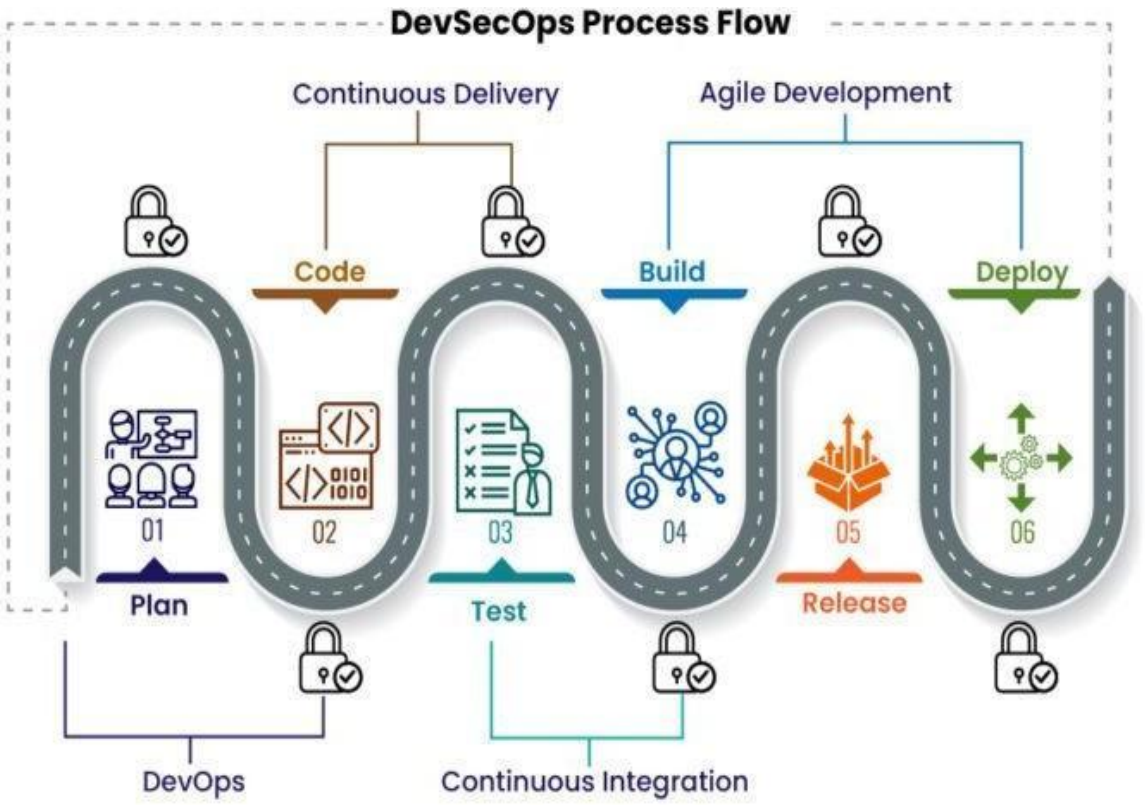
1. Windows 10 or 11
2. Ubuntu 20.0.4 30GB HD, 4 GB RAM
3. t2-micro medium 2 GB RAM

Used Languages

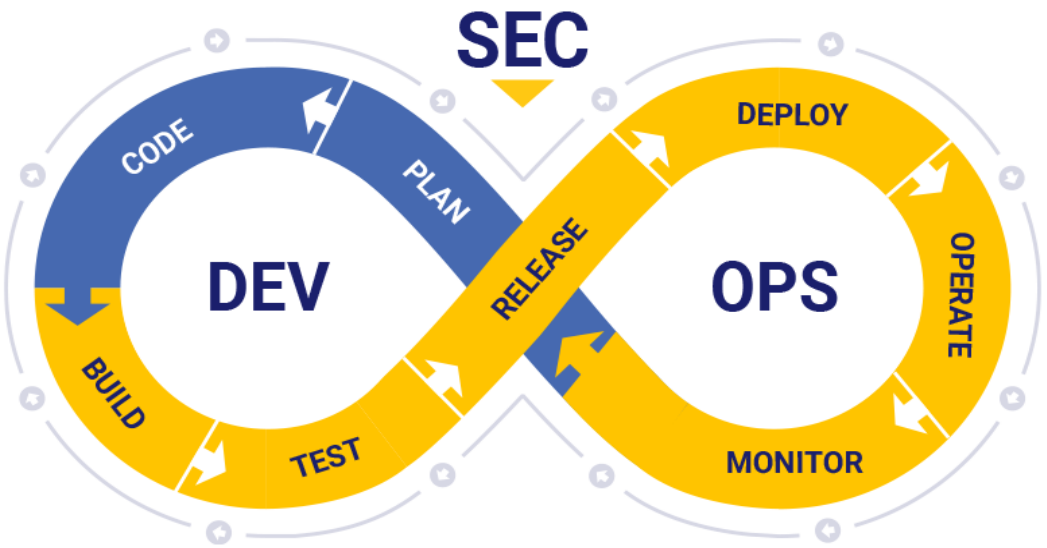
1. YAML

5. System Architecture

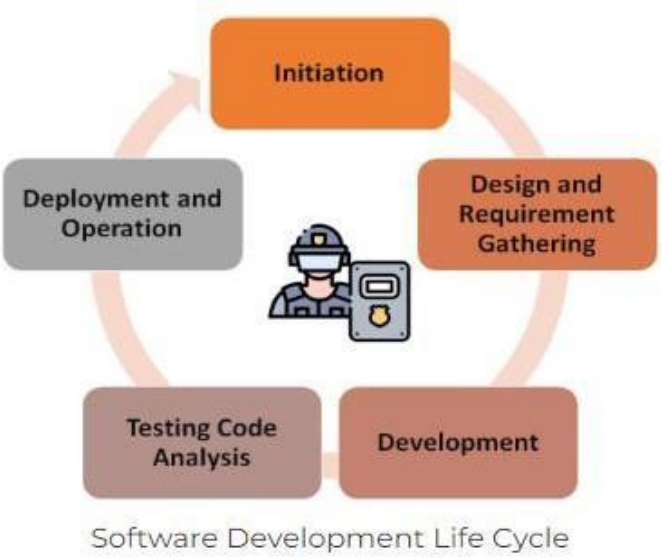
Data Flow Diagram



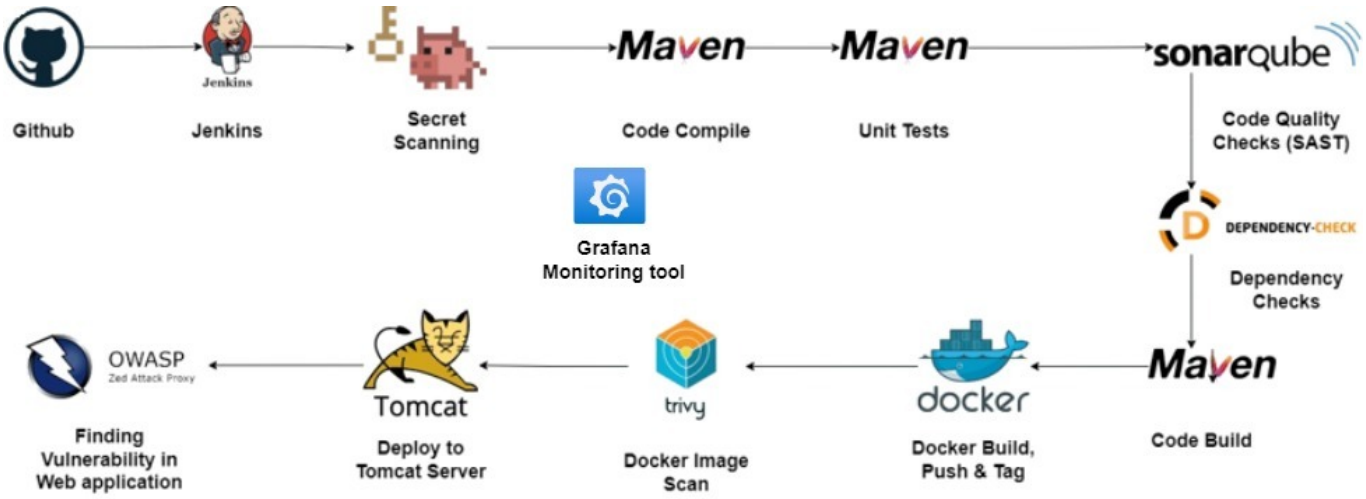
6. Activity Diagram



- **Initiation**
- **Design and Requirement gathering**
- **Development**
- **Testing and Code Analysis**
- **Deployment and Operation**



7. Process Flow Diagram



8. Git Concepts

Git is a version control system that is used to manage and track changes made to source code and other files. It was created by Linus Torvalds in 2005 to manage the development of the Linux operating system.With Git, developers can track changes to their code, collaborate with others on a project, and maintain multiple versions of their codebase. Git uses a distributed model, which means that every developer has a complete copy of the codebase on their local machine. This allows developers to work offline and independently, and then synchronize their changes with others when they're ready.

Git also provides features like branching and merging, which allow developers to create parallel versions of their codebase and merge changes made by multiple developers back into the main codebase. These features make it easier to manage complex development workflows and collaborate on large projects.

Overall, Git is a powerful and widely used tool for managing software development projects.

git init

The command git init is used to create an empty Git repository.

After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.

git init

git-init

git add

Add command is used after checking the status of the files, to add those files to the staging area.

Before running the commit command, "git add" is used to add any new or modified files.

git add .

git-add

git commit

The commit command makes sure that the changes are saved to the local repository.

The command "git commit –m <message>" allows you to describe everyone and help them understand what has happened.

git commit -m “commit message”

git-commit

git status

The git status command tells the current state of the repository.

The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status. Also, if there are no changes, it will show the message no changes to commit, working directory clean.

git config
The git config command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository.
When git config is used with --global flag, it writes the settings to all repositories on the computer.
git config --global user.name “any user name”

git config --global user.email <email id>

gitcon

git branch
The git branch command is used to determine what branch the local repository is on.
The command enables adding and deleting a branch.

```
# Create a new branch
  git branch <branch_name>

# List all remote or local branches
  git branch -a

# Delete a branch
  git branch -d <branch_name>
```

git checkout
The git checkout command is used to switch branches, whenever the work is to be started on a different branch.
The command works on three separate entities: files, commits, and branches.

```
# Checkout an existing branch
  git checkout <branch_name>

# Checkout and create a new branch with that name
  git checkout -b <new_branch>
```

git merge

The git merge command is used to integrate the branches together. The command combines the changes from one branch to another branch.
It is used to merge the changes in the staging branch to the stable branch.

```
  git merge <branch_name>
```


However, these are popular and basic git commands used by developers.

Git Commands: Working With Remote Repositories

git remote

The git remote command is used to create, view, and delete connections to other repositories.

The connections here are not like direct links into other repositories, but as bookmarks that serve as convenient names to be used as a reference.

git remote add origin <address>

git remote

git clone

The git clone command is used to create a local working copy of an existing remote repository.

The command downloads the remote repository to the computer. It is equivalent to the Git init command when working with a remote repository.

git clone <remote_URL>

git pull

The git pull command is used to fetch and merge changes from the remote repository to the local repository.

The command "git pull origin master" copies all the files from the master branch of the remote repository to the local repository.

git pull <branch_name> <remote URL>

git-pull

git push

The command git push is used to transfer the commits or pushing the content from the local repository to the remote repository.

The command is used after a local repository has been modified, and the modifications are to be shared with the remote team members.

git push -u origin master

9. Jenkins server configuration

Jenkins is used to continually create and test software projects, making it easier for developers and DevOps engineers to integrate changes to the project and for consumers to get a new build.

Project setup with CI CD Pipeline:

Pre-requisites:

1. **GitHub**

   (Repo url: https://github.com/shubnimkar/CI_CD_Devsecops.git)
2. **SonarQube (http://public-ip:9000)**
3. **Jenkins Server (http://public-ip:8080)**
4. **Maven**
5. **Tomcat**
6. **OWASP ZAP**
7. **TruffleHog**
8. **Dependency Checker**
9. **Trivy**
10. **Grafana (http://public-ip:9090)**

10. Secret Checks

In a CI/CD (Continuous Integration/Continuous Deployment) pipeline, the "secrets check" stage refers to a step in the pipeline where sensitive information, often referred to as "secrets," is validated and managed securely. Secrets could include things like API keys, passwords, access tokens, and other confidential data that are required for the deployment process but should not be exposed in the pipeline script or configuration files.

The purpose of the "secrets check" stage is to ensure that the necessary secrets are available and correctly configured before proceeding with the deployment. Here's an example of how a "secrets check" stage might be structured within a CI/CD pipeline:

It's important to note that handling secrets securely is crucial for maintaining the integrity and security of your applications. Storing secrets in environment variables or using a dedicated secret management tool like HashiCorp Vault helps prevent accidental exposure and unauthorized access to sensitive information. Additionally, ensure that the pipeline script and configuration files are properly protected to avoid exposing secrets.

Tool Used: **TruffleHog**

TruffleHog is an open-source security tool designed to search for sensitive data within code repositories. It's commonly used in software development and security workflows to identify potential security risks caused by the inclusion of sensitive information such as passwords, API keys, and other confidential data in source code.

Key features of TruffleHog include:

1. **Sensitive Data Detection**: TruffleHog uses regular expressions to scan the history of a Git repository for patterns that match common forms of sensitive data.

2. **Customizable Rules**: Users can define custom regular expressions or patterns to search for specific types of sensitive information that are relevant to their projects.

3. Git History Analysis: TruffleHog examines the commit history of a repository to identify instances where sensitive information was committed in the past, helping to identify and remediate old security risks.

4. False Positive Reduction: TruffleHog includes techniques to reduce false

positives, making it more effective at identifying actual security risks.

5. Integration: The tool can be integrated into CI/CD pipelines or other development workflows to automate the process of scanning for sensitive data in code changes.

6. Command-Line Interface: TruffleHog is primarily used via its command-line interface (CLI), making it easy to incorporate into existing development and security processes.

7. Python Tool: TruffleHog is written in Python and can be easily installed and used on various platforms.

8. Community-Driven: TruffleHog is open-source software with contributions from the security community, allowing for ongoing improvement and adaptation to evolving security needs.

11. Software composition analysis (SCA)

Software Composition Analysis (SCA) is a practice and a set of tools that focus on managing and securing the use of third-party and open-source software components in software development projects. It's an essential aspect of modern software development, especially given the prevalence of using external libraries, frameworks, and packages to accelerate development.

The main objectives of Software Composition Analysis include:

1. Identifying Dependencies: SCA tools analyze a software project's source code and binaries to identify and list all the third-party components and libraries that the project relies on.

2. License Compliance: SCA helps in identifying the licenses associated with the detected dependencies. This is crucial for ensuring that the use of third-party software aligns with the project's licensing requirements and legal obligations.

3. Vulnerability Management: SCA tools scan the dependencies for known security vulnerabilities. These tools maintain databases of known vulnerabilities and provide information about how to remediate them.

Tool Used: **OWASP Dependency Check**

OWASP Dependency-Check is a tool that helps identify known vulnerabilities within project dependencies. It primarily focuses on identifying security vulnerabilities in open-source and third-party components used in a software project.

12. Static application security testing (SAST)

Static Application Security Testing (SAST) is a type of security testing that analyzes source code, bytecode, or binary code of an application in a non-runtime (static) state to find security vulnerabilities and potential flaws. It's an important part of the software development lifecycle for identifying and addressing security issues early in the development process.

Common examples of SAST tools include Checkmarx, Fortify, Veracode, and SonarQube. As with any technology, SAST tools can evolve, and new tools might emerge, so it's always a good idea to explore the latest options and stay updated on best practices.

Tool Used: **SonarQube**

SonarQube is a widely used open-source platform for continuous code quality and security analysis. It provides a range of tools and features to help development teams ensure the quality, security, and maintainability of their codebases throughout the software development lifecycle. Originally known as Sonar, the platform was later rebranded as SonarQube.

Key features of SonarQube include:

1. Code Quality Analysis: SonarQube performs static code analysis to identify issues such as code smells, coding standards violations, and complex code structures that can impact maintainability.

2. Security Vulnerability Detection: It identifies security vulnerabilities and potential security threats in the codebase, including issues like SQL injection, cross-site scripting (XSS), and sensitive data exposure.

3. Technical Debt Management: SonarQube calculates the technical debt of a codebase based on issues identified, helping teams prioritize and manage code improvements over time.

4. Coverage Analysis: It measures code coverage by unit tests, highlighting areas of the code that lack testing and might be more prone to defects.

5. Integration with CI/CD: SonarQube integrates into CI/CD pipelines, allowing automated code analysis on every code change, providing instant feedback to developers.

6. Plugin Ecosystem: It offers a wide range of plugins and extensions to extend its capabilities and integrate with different development tools and languages.

7. Custom Rules: SonarQube allows you to create custom coding rules and quality profiles to fit your project's specific needs.

8. Dashboards and Reporting: It provides visual dashboards and detailed reports that give insights into code quality trends, issues, and progress over time.

9. Support for Multiple Languages: SonarQube supports various programming languages and frameworks, making it versatile for different types of projects.

10. Open Source and Commercial Editions: While the core of SonarQube is open source, there are also commercial editions available that provide additional features and support.

11. Community and Enterprise Support: SonarQube has a strong community and offers enterprise-level support for organizations that require dedicated assistance.

SonarQube is a valuable tool for fostering a culture of code quality, security, and continuous improvement within development teams. By integrating it into the development process, teams can proactively address issues, reduce technical debt, and deliver higher-quality software. It's important to keep in mind that tool capabilities might evolve, so referring to the official SonarQube documentation and community resources for the most up-to-date information is recommended.

13. Dynamic Application Security Testing (DAST)

Dynamic Application Security Testing (DAST) is a method of security testing that involves analyzing an application while it's running in a dynamic or operational state. Unlike Static Application Security Testing (SAST), which analyzes the source code or binaries, DAST examines the application from the outside, simulating how a real attacker might interact with it. DAST is also known as "black-box" testing because it doesn't require knowledge of the internal codebase.

It's important to note that while DAST is valuable for identifying certain types of vulnerabilities, it has limitations. For instance, it might not uncover issues related to business logic vulnerabilities, insecure design, or code quality issues. Therefore, a comprehensive security testing strategy often combines DAST with other methods like SAST (Static Application Security Testing) and IAST (Interactive Application Security Testing) for a more holistic assessment of an application's security posture.

Common examples of DAST tools include OWASP ZAP, Burp Suite, and Acunetix.

Tool Used: **OWASP ZAP**

OWASP ZAP (Zed Attack Proxy) is a widely used open-source dynamic application security testing (DAST) tool designed to help find security vulnerabilities in web applications. It provides a comprehensive set of features for identifying and mitigating security risks in web applications throughout the development lifecycle.

OWASP ZAP is an essential tool for improving the security posture of web applications. It's user-friendly, extensible, and widely adopted within the security community. However, it's important to note that DAST tools like ZAP have limitations and might not detect all types of vulnerabilities. Therefore, a holistic security strategy should combine various testing methods, including static analysis, manual testing, and other security practices, to ensure comprehensive coverage.

Key features of OWASP ZAP include:

1. Web Application Scanning: ZAP actively scans web applications by simulating real-world attacks and analyzing their responses. It identifies security vulnerabilities that could be exploited by attackers.

2. Attack Modes: ZAP offers various attack modes to simulate different types of threats, including SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and more.

3. Spidering and Crawling: ZAP can automatically navigate through a web application, crawling and interacting with different pages and inputs to discover potential vulnerabilities.

4. Dynamic Analysis: ZAP analyzes applications from the "outside," simulating how an attacker interacts with the application, making it particularly useful for identifying vulnerabilities that are dependent on runtime behavior.

5. Reporting: ZAP generates detailed reports that categorize and prioritize vulnerabilities based on severity. It provides information about the affected components and suggestions for remediation.

6. Automation and Integration: ZAP can be integrated into the development process and CI/CD pipelines, allowing for automated security testing at various stages of development.

7. Authentication and Session Management: ZAP supports testing authenticated areas of applications, including various types of authentication mechanisms and session handling.

8. Alerts and Notifications: ZAP alerts developers and security teams about detected vulnerabilities and potential risks.

14. Maven

Maven is a widely used build automation and project management tool for Java-based projects. It simplifies the process of building, packaging, and managing Java applications by providing a structured approach to project organization and dependency management. Maven uses a declarative XML-based configuration and a plugin system to perform various tasks in a consistent and automated manner.

Key features and concepts of Maven include:

1. Project Object Model (POM): Maven uses a POM file (pom.xml) to define the project's configuration, dependencies, build goals, and other information. The POM serves as the project's central configuration file.

2. Dependency Management: Maven handles project dependencies by automatically downloading and managing required libraries and components from repositories. This simplifies the process of managing external libraries and reduces version conflicts.

3. Build Lifecycle: Maven defines a set of build phases (clean, compile, test, package, etc.) that correspond to different stages of the software development lifecycle. Developers can specify which phases to execute in the build process.

4. Plugins: Maven relies on plugins to perform specific tasks, such as compiling code, running tests, generating documentation, and packaging artifacts. A rich ecosystem of plugins is available for various tasks.

5. Convention over Configuration: Maven follows a convention-based approach, which means that projects are organized in a specific directory structure by default. This reduces the need for extensive configuration.

6. Repository Management: Maven supports both local and remote repositories for storing and sharing project artifacts. Central repositories like Maven Central provide a wide range of pre-built libraries.

7. Transitive Dependencies: Maven automatically resolves and downloads

transitive dependencies, which are dependencies of project dependencies. This simplifies the management of complex dependency chains.

8. Profiles: Maven allows the configuration of different build profiles to handle varying requirements, such as different environments or build options.

9. Standardized Builds: By using Maven, projects adhere to a standardized build process, making it easier for developers to collaborate and share code across teams.

10. IDE Integration: Popular integrated development environments (IDEs) like Eclipse and IntelliJ IDEA have built-in support for Maven, making it seamless to work with Maven projects.

11. Community and Documentation: Maven has a strong community and extensive documentation, including guides and tutorials that help developers get started and utilize its features effectively.

Maven is widely used in the Java ecosystem and plays a key role in many software development processes. It simplifies the process of project setup, dependency management, and build automation, contributing to more efficient and organized development workflows.

15. Docker Configuration

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers are lightweight and portable units that can run on any machine with Docker installed, providing a consistent environment for applications to run in.

Here are the basic steps to configure Docker on your machine:

1. Install Docker: Docker provides installers for different operating systems. You can download the appropriate installer for your operating system from the Docker website and follow the instructions to install it on your machine.

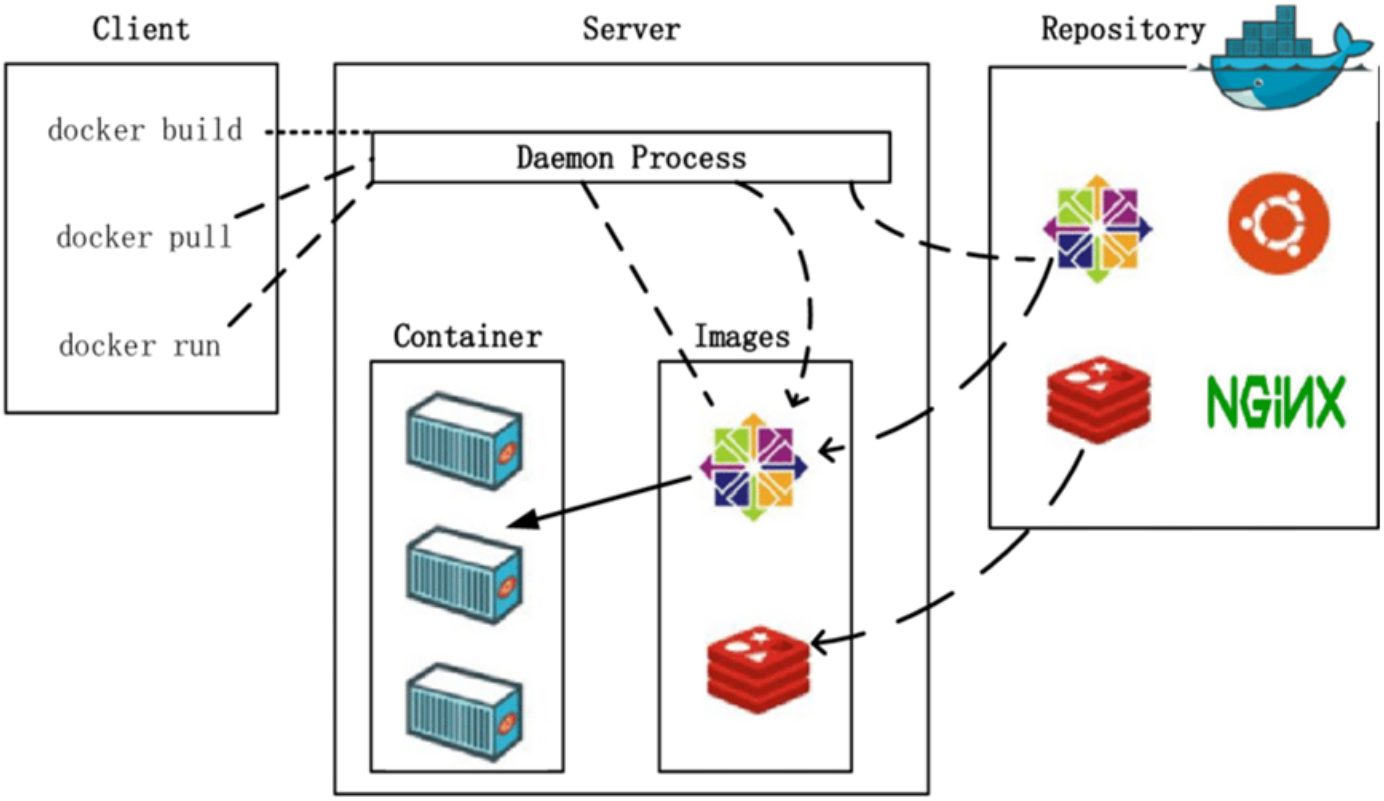
2. Verify the installation: Once you have installed Docker, you can verify the installation by running the `docker --version` command in your terminal or command prompt. This command should return the version number of Docker that you have installed.

3. Create a Dockerfile: A Dockerfile is a text file that contains instructions to build a Docker image. You can create a Dockerfile in the root directory of your application by specifying the base image, copying files into the image, and running any commands to set up your application.

4. Build the Docker image: Once you have created a Dockerfile, you can use the `docker build`

command to build the Docker image. The basic syntax of the command is:

docker build -t <image name>:<tag>



Images and containers

A container is launched by running an image. An image is an executable package that includes everything needed to run an application--the code, a runtime, libraries, environment variables, and configuration files.

A container is a runtime instance of an image--what the image becomes in memory when executed (that is, an image with state, or a user process). You can see a list of your running containers with the command, docker pc, just as you would in Linux.

After you run a docker image, it creates a docker container. All the applications and their environment run inside this container. You can use Docker API or CLI to start, stop, delete a docker container.

16. Trivy

Trivy is an open-source vulnerability scanner designed for containerized environments. It focuses on scanning container images and filesystems for known vulnerabilities, helping users identify and mitigate security risks in their containerized applications.

Key features of Trivy include:

1. Container Image Scanning: Trivy is primarily used to scan container images for known vulnerabilities. It can analyze Docker images, as well as images for other container runtimes like Podman and containerd.

2. Vulnerability Databases: Trivy uses multiple vulnerability databases, including the National Vulnerability Database (NVD) and security advisories from various Linux distributions, to identify known vulnerabilities.

3. Layer Analysis: Trivy performs a layered analysis of container images, looking at each layer's contents and identifying vulnerabilities in the packages and libraries used.

4. Security Policies: Trivy supports customizable security policies, allowing users to define severity levels for vulnerabilities and specify which vulnerabilities are acceptable or not.

5. CLI and API: Trivy offers a command-line interface (CLI) for manual scans and an API for integrating it into CI/CD pipelines and other automation workflows.

6. Caching: Trivy supports caching to speed up scanning by avoiding redundant checks of unchanged layers between scans.

7. JSON and Table Output: Trivy provides output in both JSON and table formats, making it easy to integrate into various reporting and monitoring tools.

8. Integration: Trivy can be integrated into various container orchestration tools, such as Kubernetes, to automate security scanning as part of the

deployment process.

9. Simple Configuration: Trivy's configuration is straightforward, making it accessible to users with different levels of expertise.

10. Active Development: Trivy is actively developed and maintained, ensuring that it stays up-to-date with the latest vulnerabilities and container technologies.

Trivy addresses the need for container security by allowing developers and operations teams to scan container images for vulnerabilities early in the development lifecycle. By identifying vulnerabilities before images are deployed into production, Trivy helps prevent security issues from being introduced into the application stack.

It's important to note that Trivy primarily focuses on known vulnerabilities and might not detect every security issue. Combining Trivy with other security practices like static code analysis, dynamic testing, and manual reviews provides a more comprehensive security approach.

17. Tomcat

Apache Tomcat, often referred to simply as Tomcat, is an open-source web server and servlet container developed by the Apache Software Foundation. It is designed to serve Java-based web applications and provides an environment for running Java Servlets and JavaServer Pages (JSP).

Key features and characteristics of Apache Tomcat include:

1. Servlet Container: Tomcat is primarily used as a servlet container, providing an environment for Java web applications to execute servlets and JSPs.

2. Web Server: Tomcat also functions as a web server, capable of handling HTTP requests and serving static content such as HTML, CSS, and JavaScript files.

3. Java EE Compatibility: Tomcat is not a full Java EE application server, but it supports many Java EE features and specifications. It is often used for deploying lightweight Java EE applications.

4. Modularity: Tomcat is modular, allowing developers to deploy specific components and services based on their application's requirements. This modularity enhances performance and reduces resource consumption.

5. Embedded Mode: Tomcat can be embedded within other Java applications, enabling applications to launch a Tomcat instance programmatically.

6. Connectors: Tomcat supports multiple connectors to handle different protocols, including HTTP, HTTPS, and AJP (Apache JServ Protocol), which can be used to connect to a front-end web server like Apache HTTP Server.

7. Configuration: Tomcat's configuration is primarily managed through XML configuration files, including the server.xml, context.xml, and web.xml files.

8. Logging and Monitoring: Tomcat provides logging mechanisms to track

server activities and performance. Monitoring tools can be integrated to collect performance metrics and health data.

9. Security: Tomcat offers various security features, including authentication, authorization, and SSL/TLS support.

10. Deployment: Web applications are packaged as .war (Web Application Archive) files, which can be easily deployed and managed within Tomcat.

11. Community and Extensions: Tomcat has a large and active community that develops extensions, plugins, and documentation. This community-driven support makes Tomcat a reliable choice for Java web applications.

12. Compatibility: Tomcat is compatible with various operating systems and platforms, making it versatile for different deployment scenarios.

Tomcat is widely used by developers and organizations for hosting Java web applications due to its lightweight nature, simplicity, and solid performance. It's often used in combination with other technologies and tools, such as Apache HTTP Server, to create a complete web server environment.

18. Grafana

Grafana is an open-source data visualization and monitoring tool that allows you to create interactive and customizable dashboards for various data sources. It's widely used for visualizing time-series data and metrics, making it particularly popular in the field of monitoring and observability.

Key features and characteristics of Grafana include:

1. Data Visualization: Grafana provides a user-friendly interface for creating visualizations of data. It supports various chart types, including line charts, bar charts, pie charts, and more.

2. Dashboard Creation: Users can create interactive dashboards by arranging and configuring panels containing visualizations, text, and images. Dashboards can be customized to suit specific data presentation needs.

3. Data Source Integration: Grafana supports integration with a wide range of data sources, including databases (e.g., MySQL, PostgreSQL), cloud services (e.g., AWS CloudWatch, Google Cloud Monitoring), time-series databases (e.g., Prometheus, InfluxDB), and more.

4. Query and Alerting: Grafana allows users to write queries to retrieve and manipulate data from various sources. It also offers alerting capabilities, where you can set up alerts based on certain conditions and receive notifications when those conditions are met.

5. Templating and Variables: Grafana supports template variables, allowing users to dynamically control and filter data across panels and dashboards.

6. Annotations: Annotations enable users to mark specific events or time periods on a chart to provide context for the data being displayed.

7. Plugins and Extensibility: Grafana's plugin ecosystem offers a wide range of extensions for additional data sources, panel types, and integrations.

8. Community and Sharing: Grafana has an active and engaged community

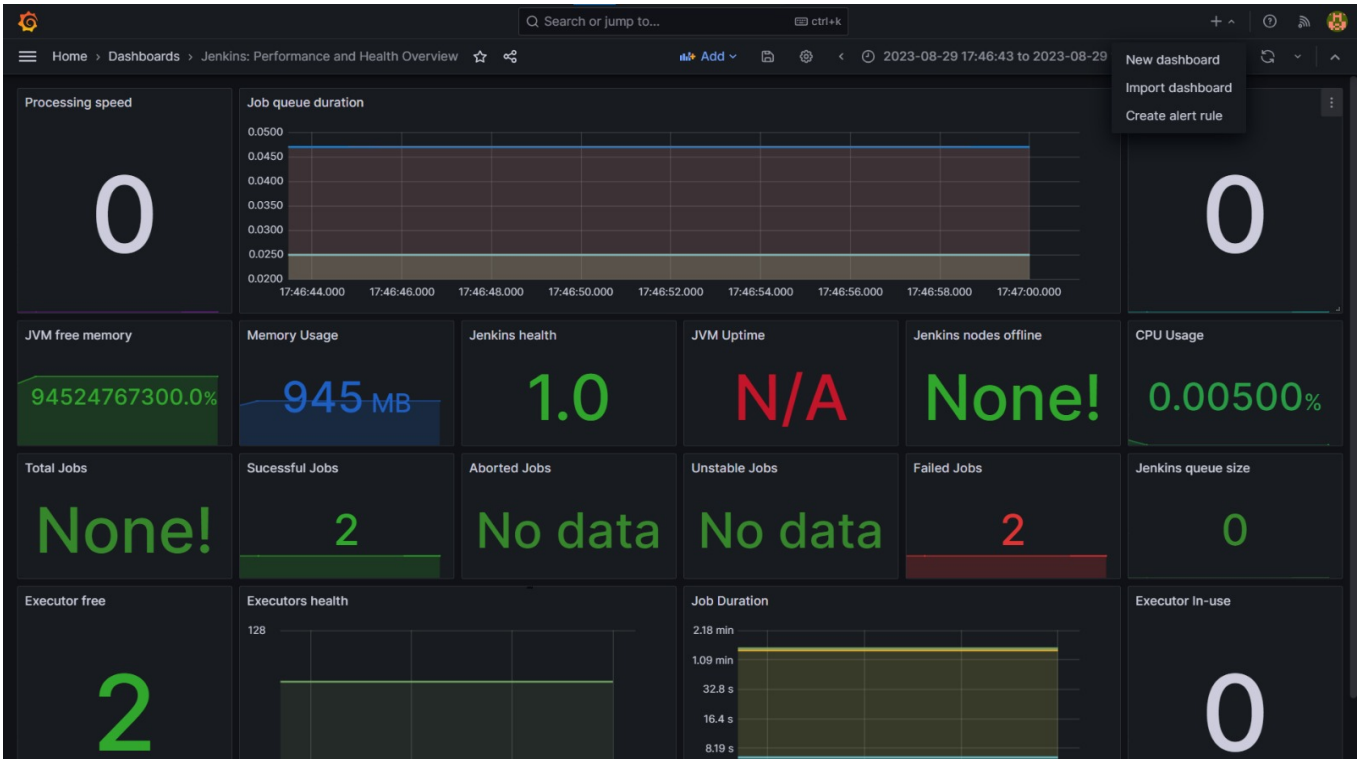
that shares dashboards, plugins, and knowledge. Dashboards can be exported and shared with others.

9. Alerting and Notification Channels: Grafana's alerting system can send notifications via various channels, including email, Slack, PagerDuty, and more.

10. User Authentication and Authorization: Grafana supports various authentication methods and user roles, allowing for secure access control to dashboards and data sources.

11. Scalability: Grafana can be used in single-server setups or in distributed environments to handle large-scale monitoring requirements.

Grafana is often used in combination with data sources like Prometheus, InfluxDB, Elasticsearch, and others to build powerful monitoring and observability solutions. It's valuable for monitoring infrastructure, applications, and services, and is known for its flexibility, ease of use, and extensive community support.



19. Code

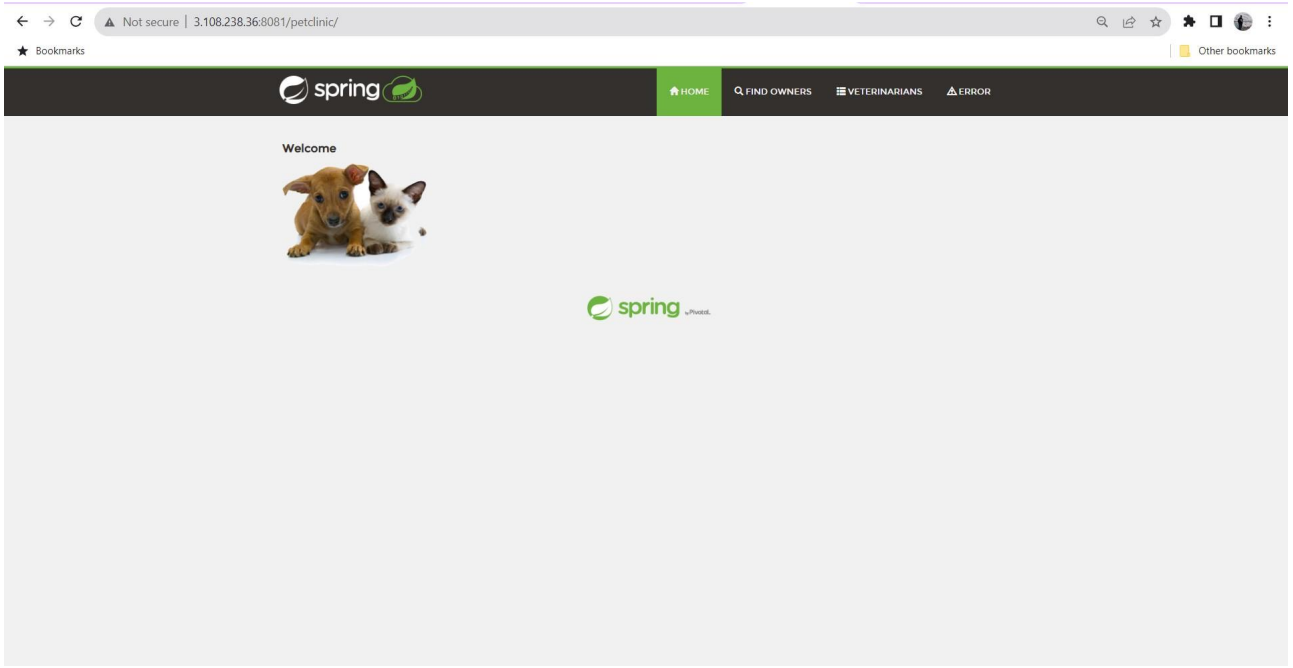
```
pipeline {
    agent any

    tools{
        jdk 'jdk17'
        maven 'maven3'
    }
                        environment {
        SCANNER_HOME=tool 'sonar-scanner'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git changelog: false, poll: false, url: 'https://github.com/shubnimkar/CI_CD_Devsecops.git'
            }
        }

stage ('Check secrets') {
    steps {
        sh 'docker run  gesellix/trufflehog --json https://github.com/shubnimkar/CI_CD_Devsecops.git > trufflehog.json'

        script {
            def jsonReport = readFile('trufflehog.json')

            def htmlReport = """
            <html>
            <head>
                <title>Trufflehog Scan Report</title>
            </head>
            <body>
                <h1>Trufflehog Scan Report</h1>
                <pre>${jsonReport}</pre>
            </body>
            </html>
            """

            writeFile file: 'scanresults/trufflehog-report.html', text: htmlReport
        }

        archiveArtifacts artifacts: 'scanresults/trufflehog-report.html', allowEmptyArchive: true
    }
}

        stage("OWASP Dependency Check"){
            steps{
                dependencyCheck additionalArguments: '--scan ./ ', odcInstallation: 'DP-Check'
                dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
            }
        }

     stage('Code Compile') {
```

```
        steps {
            sh "mvn clean compile"
        }
    }

    stage('Unit Tests') {
        steps {
            sh "mvn test"
        }
    }

    stage("Sonarqube Analysis "){
        steps{
            withSonarQubeEnv('sonar-server') {
                sh '" $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Petclinic \
                -Dsonar.java.binaries=. \
                -Dsonar.projectKey=Petclinic '"

            }
        }
    }

    stage("Build"){
        steps{
            sh " mvn clean install"
        }
    }

    stage("Docker Build"){
        steps{
            script{
                withDockerRegistry([url:'https://index.docker.io/v1/',credentialsId: '69fb7f6f-90ba-4bab-baf3-765387680986',
toolName: 'docker']) {
                    sh "docker build -t petclinic1 ."
                }
            }
        }
    }

    stage("Docker Tag & Push"){
        steps{
            script{
                withDockerRegistry([url:'https://index.docker.io/v1/',credentialsId: '69fb7f6f-90ba-4bab-baf3-765387680986',
toolName: 'docker']) {
                    sh "docker tag petclinic1 shubnimkar/pet-clinic25:latest"
                    sh "docker push shubnimkar/pet-clinic25:latest"
                }
            }
        }
    }


                    stage("TRIVY") {
                     steps {
    script {
        def trivyScanOutput = sh(script: 'trivy image shubnimkar/pet-clinic25:latest', returnStatus: true)
```

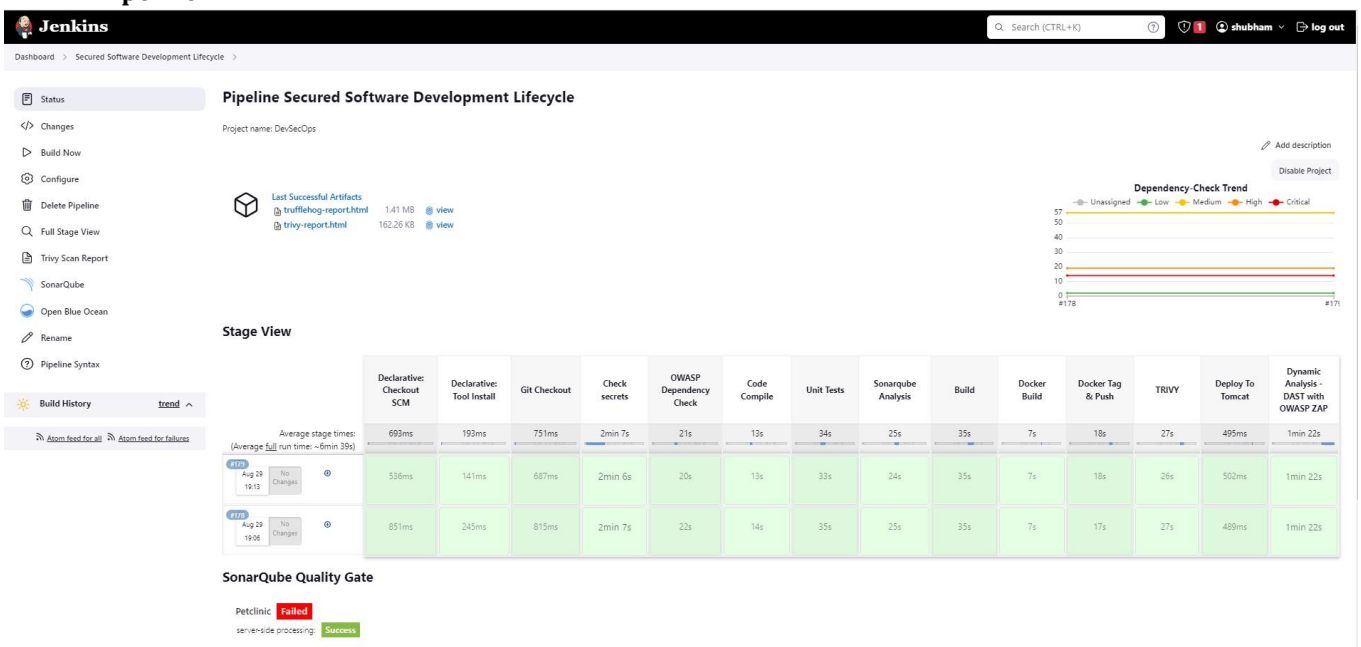
```
        if (trivyScanOutput == 0) {
            def htmlReport = sh(script: 'trivy image shubnimkar/pet-clinic25:latest | awk \'{print "<tr><td>" $1 "</td><td>" $2 "</td><td>" $3 "</td></tr>"}\' > trivy-report.html', returnStdout: true)

            archiveArtifacts artifacts: 'trivy-report.html', allowEmptyArchive: true

            // Publish HTML report using Jenkins' HTML Publisher Plugin
            publishHTML([allowMissing: false, alwaysLinkToLastBuild: true, keepAll: true, reportDir: '', reportFiles: 'trivy-report.html', reportName: 'Trivy Scan Report'])
        } else {
            error "Trivy scan failed"
        }
    }
  }
}

                    stage("Deploy To Tomcat"){
        steps{
            sh "cp  /var/lib/jenkins/workspace/DevSecOps/target/petclinic.war /opt/apache-tomcat-9.0.65/webapps/ "
        }
    }


                    stage ("Dynamic Analysis - DAST with OWASP ZAP") {
                                steps {
                                sshagent(['SSH-Cred']){
                                        sh 'ssh ubuntu@13.232.127.89 "sudo docker run --rm -v /home/ubuntu:/zap/wrk/:rw -t  owasp/zap2docker-stable zap-baseline.py -t http://3.108.238.36:8081/petclinic/ -J zap_report.json || true" '
                                        archiveArtifacts artifacts: 'zap_report.html', allowEmptyArchive: true

        }
    }
  }
  }
}
```

20. Result

- **Site Hosted**



- **Pipeline**



21. Conclusion

In conclusion, implementing a Secure Software Development Life Cycle (SSDLC) framework is crucial for successful DevSecOps implementation. SSDLC helps organizations to integrate security throughout the entire software development process, from planning to deployment. By doing so, security vulnerabilities and risks can be identified early in the development cycle, reducing the likelihood of costly security incidents later on.

DevSecOps focuses on a collaborative approach to software development, bringing together development, operations, and security teams to work towards a common goal. Implementing an SSDLC framework within DevSecOps provides a structured approach to security, enabling teams to work together more effectively and efficiently.

Key elements of an SSDLC framework include threat modeling, code analysis, testing, and continuous monitoring. By integrating these processes into the development cycle, teams can identify and address security vulnerabilities early in the process and reduce the risk of security incidents.

Implementing an SSDLC framework within DevSecOps requires a culture shift towards security as a shared responsibility across teams, a commitment to continuous improvement, and a focus on collaboration and communication. With the right tools, processes, and mindset, organizations can build secure, high-quality software that meets the needs of their customers and protects their data and assets.

At this point your SSDLC project with DevSecOps is being configured successfully.

22. References

Here are some references that can be used for further reading on implementing an SSDLC framework in DevSecOps: 1."Secure DevOps: A Pragmatic Approach to Securing Software in the Cloud" by Julien Vehent, O'Reilly Media, 2018.

2. "Implementing DevSecOps: Security in the DevOps Lifecycle" by Yogesh Singh and Prabath Siriwardena, Apress, 2018.

3. "The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations" by Gene Kim, Jez Humble, Patrick Debois, and John Willis, IT Revolution Press, 2016.

4. "OWASP DevSecOps Maturity Model" by OWASP, available at: OWASP Devsecops Maturity Model _ OWASP Foundation_files

5."Continuous Security in DevOps" by Paula Thrasher, IEEE Security & Privacy, vol. 16, no. 5, pp. 68-71, 2018.

6. "DevSecOps: How to Seamlessly Integrate Security into DevOps" by Fahmida Y. Rashid, SD Times, available at: DevSecOps Archives - SD Times_files

7."Securing DevOps: Security in the Cloud" by Julien Vehent, O'Reilly Media, 2017.

8. "Secure Software Development: A Comprehensive Guide to Developing Secure Software" by Mark Merkow and Lakshmikanth Raghavan, Auerbach Publications, 2012. SSH to jenkins.devsecops.lab

Install Java : C:\Users\Desktop\How To Install Java with Apt-Get on Ubuntu 16.04 _ DigitalOcean_files

Install Jenkins: https://wiki.jenkins.io/display/JENKINS/Installing+Jenkins+on+Ubuntu
Install Docker:
https://www.digitalocean.com/community/tutorials/how-to-install-docker-with-apt-on-ubuntu-18-04 Modify rule in Security Groups for port 8080
Activate Jenkins from /var/lib/Jenkins/secrets/initialAdminPassword