Project Report

on

Vulnerability Assessment using DevSecOps

Submitted in partial fulfillment for the award of

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

Guided by:

Mr. Roshan Gami

Presented by:

Ms. Shradha Sonawane	**PRN:230340127008**
Mr. Vimeet K. Patil	**PRN:230340127054**
Ms. Prachi Nimgade	**PRN:230340127043**
Ms. Priyanka Kumari	**PRN:230340127044**
Ms. Saloni Bhole	**PRN: 230340127002**
Ms. Kranti Saknure	**PRN: 230340127040**

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

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Ms. Shradha Sonawane

Mr. Vimeet K. Patil

Ms. Prachi Nimgade

Ms. Priyanka Kumari

Ms. Saloni Bhole

Ms. Kranti Saknure

have successfully completed their project on

Vulnerability Assessment using DevSecOps

Under the Guidance of Mr. Roshan Gami

Project Guide **Project Supervisor**

HOD ACTS
Mr. Aditya Sinha

ACKNOWLEDGEMENT

This project "**Vulnerability Assessment using DevSecOps**" 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.

From:

Ms. Shradha Sonawane (230340127008)

Mr. Vimeet K. Patil (230340127054)

Ms. Prachi Nimgade (230340127043)

Ms. Priyanka Kumari (230340127042)

Ms. Saloni Bhole (230340127002)

Ms. Kranti Saknure (230340127040)

TABLE OF CONTENTS

1. Abstract

In the rapidly evolving landscape of software development, security vulnerabilities present a significant risk to organizations, their data, and their users. The emergence of DevSecOps, a symbiotic integration of Development, Security, and Operations practices, offers a comprehensive approach to addressing these vulnerabilities. This project delves into the realm of "Vulnerability Assessment using DevSecOps," where security is infused into every stage of the software development life cycle.

The project's primary objective is to establish an effective framework for identifying, mitigating, and managing vulnerabilities within a DevSecOps context. By merging security practices with continuous integration, continuous deployment (CI/CD), automation, and collaborative workflows, this project seeks to create an ecosystem that inherently prioritizes security.

The framework employs a range of automated tools and processes to scan for vulnerabilities in both code and dependencies, offering real-time feedback to development teams. Moreover, the project explores the integration of security policies into the pipeline, allowing for instantaneous checks and balances that prevent insecure code from progressing further. Through the lens of vulnerability assessment, this project unveils the power of "shifting left" – tackling security concerns early in the development cycle to prevent the perpetuation of vulnerabilities.

As a result of this project, organizations can anticipate reduced risk exposure, higher software quality, and more efficient development cycles. Security becomes an intrinsic component of the development process, transforming vulnerabilities from liabilities into opportunities for growth. By embracing Vulnerability Assessment using DevSecOps, organizations empower themselves to thrive in the face of an ever-evolving threat landscape, fortifying their software against vulnerabilities and safeguarding their digital assets.

2. Introduction and overview of project

More and more companies are in the process of adopting modern continuous software development practices and approaches like continuous integration (CI), continuous delivery (CD), or DevOps. These approaches can support companies in order to increase the development speed, the frequency of product increments, and the time to market. To be ableto get these advantages, especially the tooling and infrastructure need to be reliable and secure. In case CI/CD is compromised or even unavailable, all mentioned advantages are at stake. Potentially, this could also even hinder the forthcoming of the software development. Therefore, our goal was to identify which vulnerabilities are present in industry CI/CD pipelines and how they can be detected. In this project, we present our results of an industry case study which includes a qualitative survey of JAVA based project teams regarding the awareness of security in CI/CD, the analysis and abstraction of CI/CD pipelines, and a threat analysis based on the deducted CD pipeline to identify vulnerabilities. In this case study, we found that the team members that work with the CD pipeline in different roles do not have a strong security background but are aware of security attributes in general. Furthermore, CI/CD pipelines from industry projects were analyzed using the ZAP by OSAWP In total, we identified 15 warnings that have been confirmed by the project teams.

3. Implementing SSDLC Framework in DevSecOps

Vulnerability assessment within a DevSecOps framework involves integrating security practices into the entire software development life cycle to proactively identify and address vulnerabilities. Here's a concise flow of vulnerability assessment using DevSecOps:

DevSecOps integrates security into every phase of the software development life cycle, from planning and coding to testing and deployment. In the context of vulnerability assessment:

1. Early Detection: DevSecOps emphasizes identifying vulnerabilities early in the development process. Automated tools and code analysis are used to detect security weaknesses as code is being written, ensuring issues are caught before they reach production.

2. Continuous Monitoring: Vulnerability scanning tools are integrated into the CI/CD pipeline, scanning code and dependencies at each stage. This ensures that any new vulnerabilities are identified as changes are made and code is built and deployed.

3. Automated Testing: DevSecOps employs automated security tests, including static analysis for code vulnerabilities, dynamic tests for application behavior, and interactive tests for real-time analysis. These tests provide quick feedback to developers, enabling them to fix issues promptly.

4. Dependency Management: DevSecOps tools assess third-party libraries for vulnerabilities, ensuring that dependencies are up-to-date and free from known security flaws.

5. Policy Enforcement: DevSecOps enforces security policies through automated checks. If code or configurations violate security rules, the pipeline can halt or flag the deployment, preventing insecure code from reaching production.

6. Collaboration: DevSecOps promotes collaboration between development, operations, and security teams. By integrating security considerations into the development workflow, teams work together to address vulnerabilities efficiently.

7. Remediation: When vulnerabilities are identified, the DevSecOps approach facilitates rapid remediation. Automated tests verify fixes, ensuring that vulnerabilities are resolved effectively.

8. Continuous Improvement: DevSecOps incorporates feedback from security incidents and assessments to continuously enhance security practices, refining the vulnerability assessment process over time.

In summary, vulnerability assessment using DevSecOps is a proactive approach that integrates security practices into every step of the software development life cycle. This ensures that potential vulnerabilities are detected, addressed, and remediated early, resulting in more secure software releases and reduced risk of security breaches.

4. System Requirement

4.1 User Interface

1. EC2 Instance Ubuntu

4.2 Software Requirement

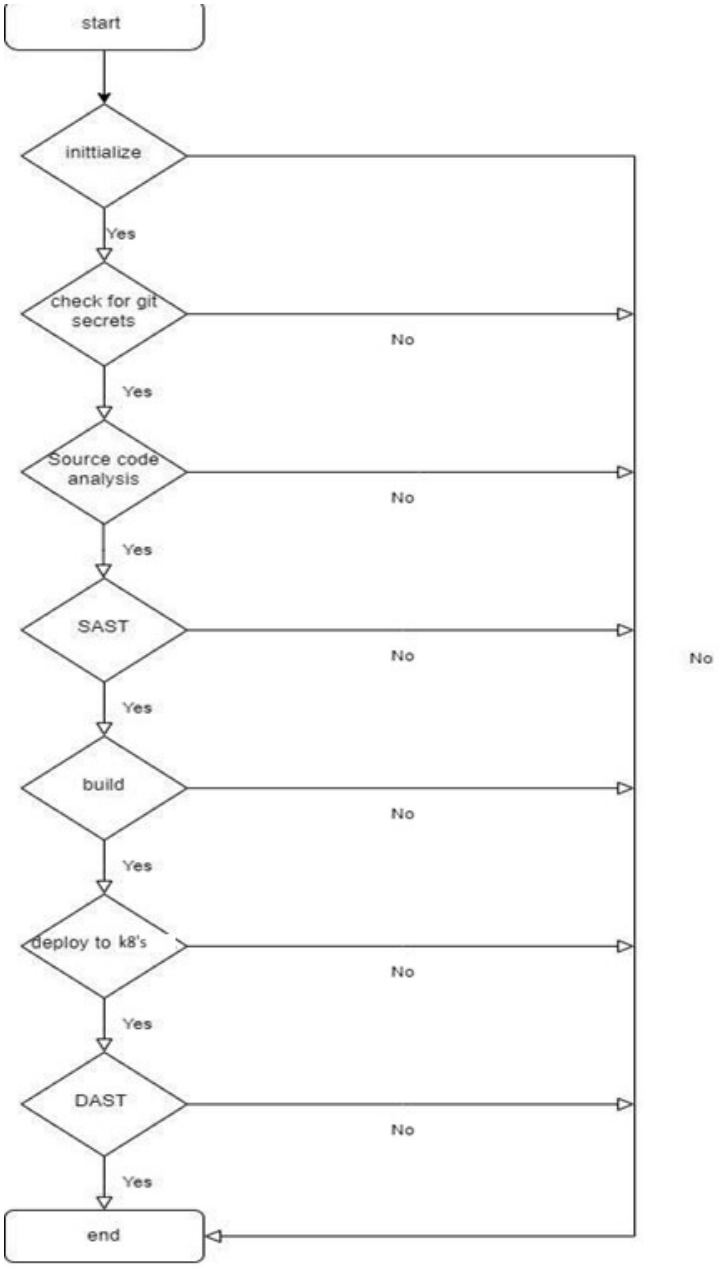
1. GIT
2. Jenkins
3. Docker
4. Kubernetes
   (Microk8s)
5. SAST
   Sonarqube
6. OWASP
   ZAP
7. Dependency-check 8.1.0
8. Apache Maven version- 3.6.37.
9. Trufflehog
10. Apache Tomcat 9(Docker Image)

4.3 Hardware Requirement

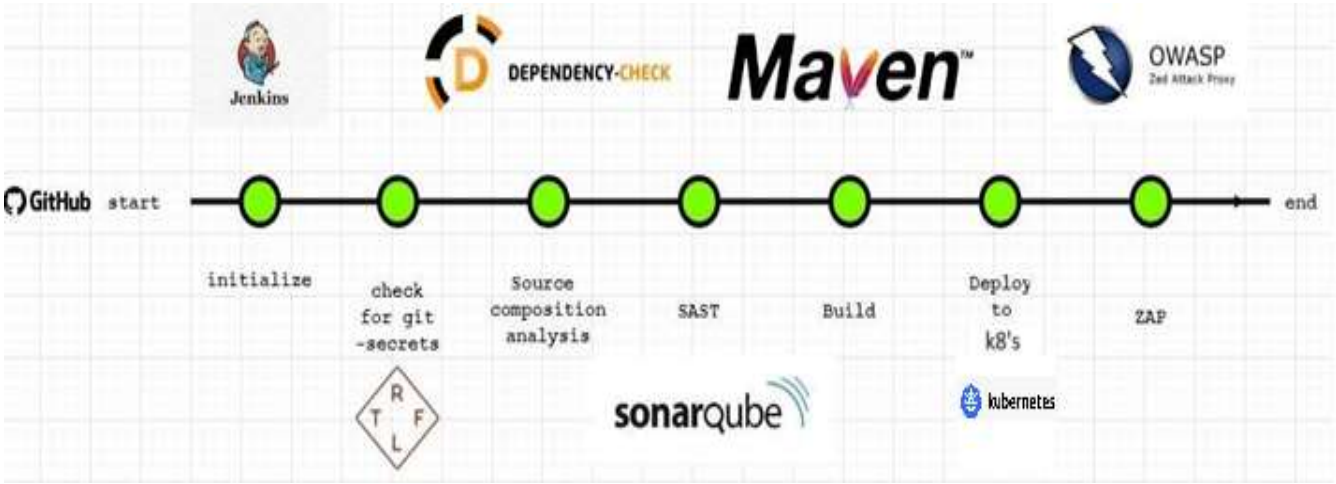
1. Windows 10 or 11
2.Ubuntu 20GB Harddrive
3.t2-micro medium 4GBRAM

System Architecture

5.1 Data Flow Diagram



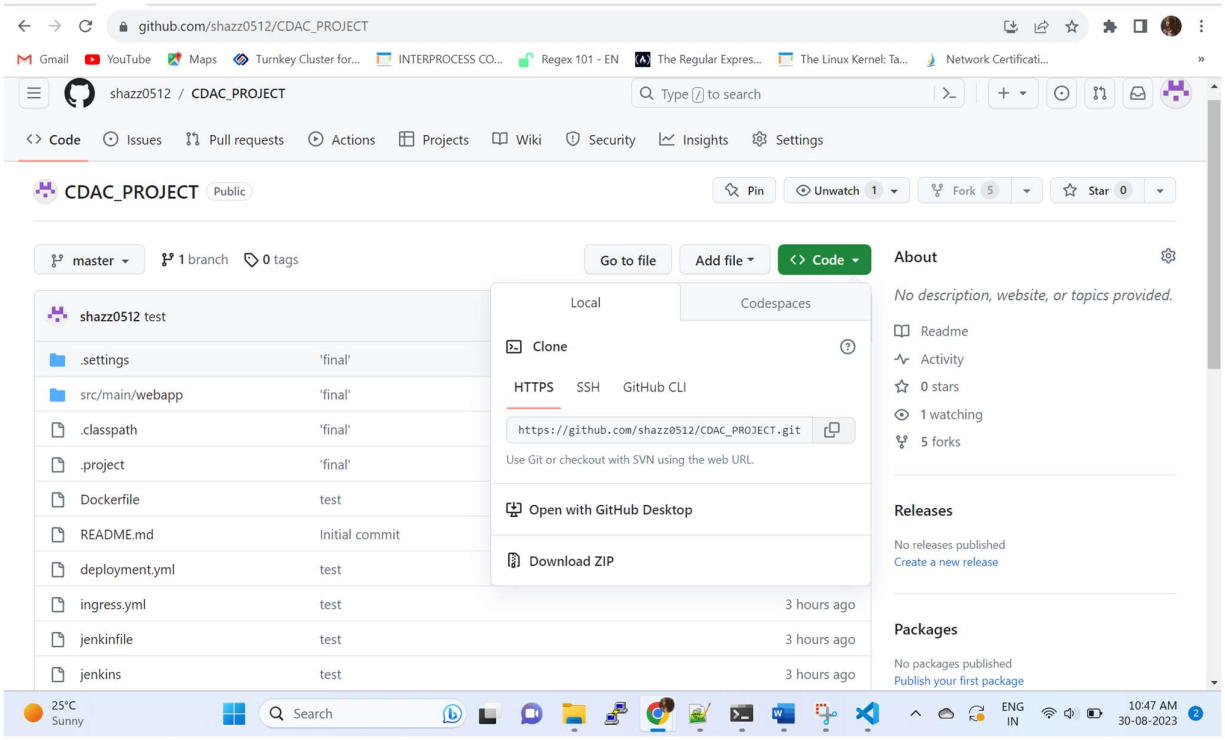
5.2 Activity Diagram



6. Introduction to Repository

First log-in with your Github account, then create new repository for the project.

Now, copy the url of the repository.



7. Git Concepts

Git is a version control system that is used to manage and track changes made to source code andother 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 ona 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.

1. Repository (Repo):
   - A repository is a collection of files and directories that make up a project, along with the version history of those files.
   - Git repositories can be hosted locally or on remote platforms like GitHub, GitLab, and Bitbucket.

2. Commit:
   - A commit is a snapshot of the changes made to the files in your repository at a specific point in time.
   - Each commit has a unique identifier (hash) and includes information about the author, date, and a commit message describing the changes.

3. Branch:
   - A branch is a parallel version of a repository that allows you to work on features or fixes without affecting the main codebase (usually the `master` branch).
   - You can create, switch between, and merge branches.

4. Merge:
   - Merging combines changes from one branch into another, typically to incorporate changes made on a feature branch back into the main branch.
   - Merge conflicts can occur when Git cannot automatically reconcile different changes made in the same part of a file.

5. Pull Request (PR) / Merge Request (MR):
   - A pull request (or merge request) is a way to propose changes from a branch to another branch (often from a feature branch to the main branch).
   - It allows code review and discussion before the changes are merged.

6. Clone:
   - Cloning creates a copy of a remote repository on your local machine. This allows you to work on the code locally and push your changes back to the remote repository.

7. Fetch:
   - Fetch downloads the latest changes from a remote repository but doesn't apply them to your local branch. It updates your local tracking branches.

8. Pull:
- Pull is a combination of `fetch` and `merge`. It fetches the latest changes from the remote repository and automatically merges them into your local branch.

9. Push:
- Push sends your local commits to a remote repository. It updates the remote repository with the changes you've made locally.

10. Remote:
- A remote is a version of the repository that is hosted on a server, separate from your local machine.
- You can push and pull changes to and from remotes.

11. Origin:
- "Origin" is often the default name given to the remote repository from which you cloned your local repository.

12. HEAD:
- `HEAD` is a special pointer that points to the latest commit in the current branch. It's where you're currently working.

13. Staging Area (Index):
- The staging area is a space where you prepare and organize your changes before committing them. Changes added to the staging area are included in the next commit.

8. Jenkins Pipelines

Jenkins Pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.

A *continuous delivery (CD) pipeline* is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" viathe Pipeline domain-specific language (DSL) syntax
The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile) which in turn can be committed to a project's source control repository. This is the foundation of "Pipeline-as-code"; treating theCD pipeline a part of the application to be versioned and reviewed like any other code

Pipeline

A Pipeline is a user-defined model of a CD pipeline. A Pipeline's code defines your entire build process, which
typically includes stages for building an application, testing it and then delivering it.

Node

A node is a machine which is part of the Jenkins environment and is capable of executing a Pipeline

Stage

A stage block defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g. "Build", "Test" and "Deploy" stages), which is used by many plugins to visualize or present Jenkins Pipeline status/progress.

Step

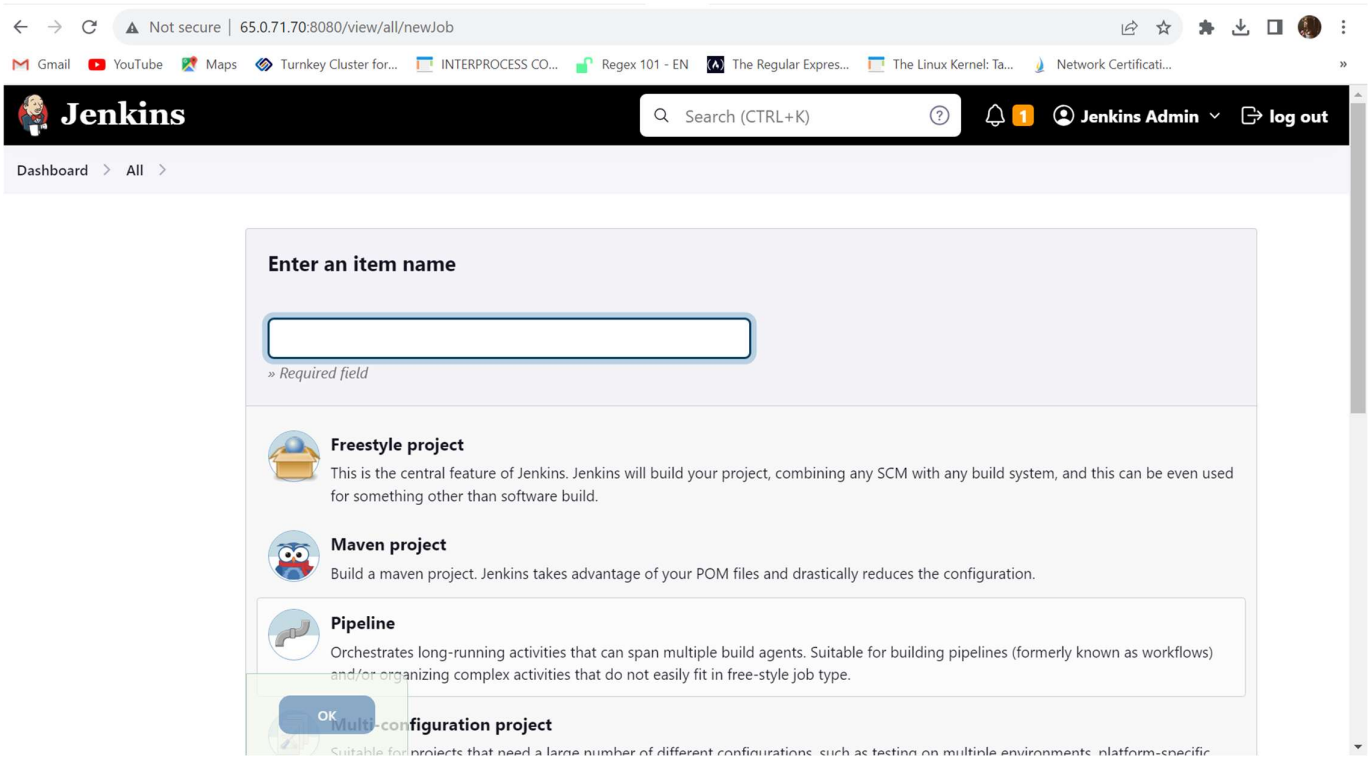
A single task. Fundamentally, a step tells Jenkins *what* to do at a particular point in time (or "step" in the process). For example, to execute the shell command make use the sh tep: sh 'make'. When a plugin extends the Pipeline DSL, [1] that typically means the plugin has implemented a new *step*

Declarative Pipeline fundamentals

In Declarative Pipeline syntax, the pipelineblock defines all the work done throughout your entire Pipeline.

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent
    any (1)
    stages {
        stage('Build')
            { (2)steps
            {
                // (3)
            }
        }
        stage('Test')
            { (4)steps
            {
                // (5)
            }
        }
        stage('Deploy')
            { (6)steps {
                // (7)
            }
        }
    }
}
```


Not secure | 65.0.71.70:8080/view/all/newJob
Gmail
YouTube
Maps
Turnkey Cluster for...
INTERPROCESS CO...
Regex 101 - EN
The Regular Expres...
The Linux Kernel: Ta...
Network Certificati...
Jenkins
Search (CTRL+K)
1
Jenkins Admin
log out
Dashboard
All
Enter an item name
» Required field
Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
OK

9. Trufflehog

TruffleHog is a utility that searches through git repositories for secrets, private keys and credentials so thatyou can protect your data before a breach occurs.

It is effective at finding secrets accidentally committed.

TruffleHog previously functioned by running entropy checks on git diffs. This functionality still exists, but high signal regex checks have been added, and the ability to suppress entropy checks have also been added.

Let's suppose there is a requirement from your client end to figure out all the secrets, sensitive API keyswhich are present in the code repositories.
The steps below will guide you on how to setup TruffleHog to meet the above requirement.

In the following section, we will setup a Jenkins pipeline in which we will launch a Docker container with thehelp of Dockerfile and run our TruffleHog utility in that Docker container.

Step 1: Go to your Jenkins server and start creating a Freestyle project.

Step 2: In the general section of your pipeline, give a brief description of your pipeline

Step 3: Give a log rotation strategy to your pipeline.

Step 4: Here, we are leaving the rest of the options to default values. You can change and give options according to your need

Step 5: Under the source code management section, pass the repository URL. Your git repo contains a Dockerfile that will build your docker container.

Step 6: Under the 'build' section, execute the shell commands:

Step 7: After your pipeline is executed successfully, you can go to console output and see the pipeline result.

10. OWASP Dependency-Check

Dependency-Check is a Software Composition Analysis (SCA) tool that attempts to detect publicly disclosed
vulnerabilities contained within a project's dependencies. It does this by determining if there is a Common Platform Enumeration (CPE) identifier for a given dependency. If found, it will generate a report linking to the associated CVE entries.

Dependency-check has a command line interface, a Maven plugin, an Ant task, and a Jenkins plugin. The core engine contains a series of analyzers that inspect the project dependencies, collect pieces of information about the dependencies. The evidence is then used to identify the Common Platform Enumeration (CPE) for the givendependency. If a CPE is identified, a listing of associated Common Vulnerability and Exposure (CVE) entries are listed in a report. Other 3rd party services and data sources such as the Jenkins plugin, Maven plugin, SSH Agents.

Dependency-check works by collecting information about the files it scans (using Analyzers). The information collected is called Evidence; there are three types of evidence collected: vendor, product, and version. For instance, the Jar Analyzer will collect information from the Manifest, pom.xml, and the package names withinthe JAR files scanned and it has heuristics to place the information from the various sources into one or morebuckets of evidence.

The OWASP Top 10 2013 contains a new entry: A9-Using Components with Known Vulnerabilities. Dependency Check can currently be used to scan applications (and their dependent libraries) to identify anyknown vulnerable components.

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impact.

11. SonarQube

SonarQube is an automatic code review tool to detect bugs, vulnerabilities, and code smells in your code. It can integrate with your existing workflow to enable continuous code inspection across your project branchesand pull requests.

SonarQube is a Code Quality Assurance tool that collects and analyzes source code, and provides reports for the code quality of your project. It combines static and dynamic analysis tools and enables quality to be measured continually over time. Everything from minor styling choices, to design errors are inspected and evaluated by SonarQube. This provides users with a rich searchable history of the code to analyze where the code is messing up and determine whether or not it is styling issues, code defeats, code duplication, lack of test coverage, or excessively complex code. The software will analyze source code from different aspects anddrills down the code layer by layer, moving module level down to the class level, with each level producing metric values and statistics that should reveal problematic areas in the source code that needs improvement.

Sonarqube also ensures code reliability, Application security, and reduces technical debt by making your codebase clean and maintainable. Sonarqube also provides support for 27 different languages, including C, C++, Java, Javascript, PHP, GO, Python, and much more.SonarQube also provides Ci/CD integration, and gives feedback during code review with branch analysis and pull request decoration.

SonarQube's integration with GitHub Enterprise and GitHub.com allows you to maintain code quality and security in your GitHub repositories.

With this integration, you'll be able to:

- Import your GitHub repositories - Import your GitHub repositories into SonarQube to easily set upSonarQube projects.
- Analyze projects with GitHub Actions - Integrate analysis into your build pipeline. Starting in DeveloperEdition, SonarScanners running in GitHub Actions jobs can automatically detect branches or pull requests being built so you don't need to specifically pass them as parameters to the scanner
- Report your Quality Gate status to your branches and pull requests - (starting in Developer Edition) Seeyour Quality Gate and code metric results right in GitHub so you know if it's safe to merge your changes.
- Authenticate with GitHub - Sign in to SonarQube with your GitHub credentials.

Static Code Analysis

Static code analysis is done without executing any of the code. It is a collection of algorithms and techniques to analyze source code to automatically find potential errors and poor coding practices. This is done with compiler errors and run-time debugging techniques such as white box testing. Static code analysis is also
considered a way to automate code review process. The tasks involved in static code analysis can be divided as such:

- Detecting errors in programs
- Recommendations on code formatting with a formatter
- Metrics computation, which gives you back a rating on how well your code is.
- Popular tools for static Code Analysis are Checkstyle, PMD, and Find Bug

Getting SonarQube from a Docker Image

- Find the community version of SonarQube that you want to use on DockerHub: https://hub.docker.com/_/sonarqube/
- Start the server by running:
  docker run -d --name sonarqube -p 9000:9000 <image name>
- Login to http://localhost:9000 with the following credentials for system admin: (admin/admin)

Analyzing a project with SonarQube

Once you are logged into sonarqube, to analyze a project follow the following steps:

i. Click create new project button
ii. When asked How do you want to create your project, select Manually.
iii. Give your project a Project key and a Display name and click the Set Up button.
iv. Under Provide a token, select Generate a token. Give your token a name, click the Generate button,and click Continue.
v. Select your project's main language under Run analysis on your project, and follow the instructions to analyze your project. Here you'll download and execute a Scanner on your code (if you're using Mavenor Gradle, the Scanner is automatically downloaded).

After successfully analyzing your code, you'll see your first analysis on SonarQube.

12. Maven

Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal, Maven deals with several areas of concern:

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Encouraging better development practices

Making the build process easy

While using Maven doesn't eliminate the need to know about the underlying mechanisms, Maven does shielddevelopers from many details

Providing a uniform build system

Maven builds a project using its project object model (POM) and a set of plugins. Once you familiarize yourselfwith one Maven project, you know how all Maven projects build. This saves time when navigating many projects

Providing quality project information

Maven provides useful project information that is in part taken from your POM and in part generated from your project's sources. For example, Maven can provide

- Change log created directly from source control
- Cross referenced sources
- Mailing lists managed by the project
- Third party code analysis products also provide Maven plugins that add their reports to the standardinformation given by Maven Dependencies used by the project
- Unit test reports including coverage

Providing guidelines for best practices development

Maven aims to gather current principles for best practices development and make it easy to guide a project inthat direction.

For example, specification, execution, and reporting of unit tests are part of the normal build cycle using Maven. Current unit testing best practices were used as guidelines:

- Keeping test source code in a separate, but parallel source tree
- Using test case naming conventions to locate and execute tests
- Having test cases setup their environment instead of customizing the build for test preparation

Maven also assists in project workflow such as release and issue management.

Maven also suggests some guidelines on how to layout your project's directory structure. Once you learn the layout, you can easily navigate other projects that use Maven.

While takes an opinionated approach to project layout, some projects may not fit with this structure for historical reasons. While Maven is designed to be flexible to the needs of different projects, it cannot cater toevery situation without compromising its objectives.

If your project has an unusual build structure that cannot be reorganized, you may have to forgo some features or the use of Maven altogether.

The result is a tool that can now be used for building and managing any Java-based project. We hope that we have created something that will make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

13.Tomcat

It is an open-source Java servlet container that implements many Java Enterprise Specs such as the Websites API, Java-Server Pages and last but not least, the Java Servlet. It began as the reference implementation for the very first Java-Server Pages and the Java Servlet API. However, it no longer works as the reference implementation for both of these technologies, but it is considered as the first choice among the users evenafter that. It is still one of the most widely used java-sever due to several capabilities such as good extensibility, proven core engine, and well-test and durable. Here we used the term "servlet" many times, sowhat is java servlet; it is a kind of software that enables the webserver to handle the dynamic(java-based) content using the Http protocols.

The Java ecosystem supports a wide variety of application servers, so let's have a little discussion on each ofthem and see where Tomcat fits in:

A servlet container is basically an implementation of the Java servlet specification, which is mainly used for thepurpose of hosting Java servlets.

The Java enterprise application-server is an implementation of the Java specification.

A web- server is a kind of server designed to serve files using a local system such as Apache.

We can say that, at the center, the Tomcat is JSP (Java Server Pages) and Servlet. The JSP is one of the server-side programming technologies that enables the developers to create platform-independent dynamiccontent and also known as the server-side view rendering technology. A servlet is a java-based software component that helps in extending the capabilities of a server. However, it can also respond to several kinds of requests and generally implemented web server containers to host the web-applications on the
webservers. As the developer's point of view, we just have to write the java server pages (or JSP) or the servletand not required to worry about routing; the Tomcat will handle the routing.

The Tomcat also consists of the webserver known as the Coyote engine due to which it's possible to extend the capability of Tomcat to include several java enterprise specs, and including the Java Persistence API(JPA).The Tomcat also has an extended version known as the "TomEE" that contains more enterprise features.

The Java ecosystem supports a wide variety of application servers, so let's have a little discussion on each ofthem and see where Tomcat fits in:

A servlet container is basically an implementation of the Java servlet specification, which is mainly used for thepurpose of hosting Java servlets.

The Java enterprise application-server is an implementation of the Java specification.

A web- server is a kind of server designed to serve files using a local system such as Apache.

We can say that, at the center, the Tomcat is JSP (Java Server Pages) and Servlet. The JSP is one of the server-side programming technologies that enables the developers to create platform-independent dynamiccontent and also known as the server-side view rendering technology. A servlet is a java-based software component that helps in extending the capabilities of a server. However, it can also respond to several kinds of requests and generally implemented web server containers to host the web-applications on the
webservers. As the developer's point of view, we just have to write the java server pages (or JSP) or the servletand not required to worry about routing; the Tomcat will handle the routing.

The Tomcat also consists of the webserver known as the Coyote engine due to which it's possible to extend the capability of Tomcat to include several java enterprise specs, and including the Java Persistence API(JPA).The Tomcat also has an extended version known as the "TomEE" that contains more enterprise features.

14. ZAP

OWASP ZAP is an ideal tool to use in automation (security testing). It can be run in headless mode and has a powerful API. The OWASP Zed Attack Proxy (OWASP ZAP) is an easy-to-use integrated penetration testing toolfor finding vulnerabilities in web applications. ZAP passively scans all the requests and responses made duringyour exploration for vulnerabilities, continues to build the site tree, and records alert for potential vulnerabilities found during the exploration.

OWASP ZAP will proceed to crawl the web application with its spider and passively scan each page it finds.Then it will use the active scanner to attack all of the discovered pages, functionality, and parameters.

Software security testing is the process of assessing and testing a system to discover security risks and vulnerabilities of the system and its data. There is no universal terminology but for our purposes, we define assessments as the analysis and discovery of vulnerabilities without attempting to actually exploit those vulnerabilities. We define testing as the discovery and attempted exploitation of vulnerabilities.

Security testing is often broken out, somewhat arbitrarily, according to either the type of vulnerability being tested or the type of testing being done. A common breakout is

- Vulnerability Assessment – The system is scanned and analyzed for security issues.
- Penetration Testing – The system undergoes analysis and attack from simulated malicious attackers
- Runtime Testing – The system undergoes analysis and security testing from an end-user
- Code Review – The system code undergoes a detailed review and analysis looking specifically forsecurity vulnerabilities

Pretesting usually follows these stages:

- Explore – The tester attempts to learn about the system being tested. This includes trying to determine what software is in use, what endpoints exist, what patches are installed, etc. It also includes searching the site for hidden content, known vulnerabilities, and other indications of weakness.
- Attack – The tester attempts to exploit the known or suspected vulnerabilities to prove they exist.
- Report – The tester reports back the results of their testing, including the vulnerabilities, how they exploited them and how difficult the exploits were, and the severity of the exploitation.

15. Kubernetes

Kubernetes, often abbreviated as K8s, is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes provides a powerful set of tools and features that simplify the process of managing complex containerized applications, making them more scalable, reliable, and efficient.

Key concepts and components of Kubernetes include:

1. Node: A physical or virtual machine that runs containerized applications. Nodes are grouped together into clusters.

2. Cluster: A collection of nodes that work together to run containerized applications. Clusters consist of a control plane and worker nodes.

3. Control Plane: The central management component of Kubernetes, responsible for making global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events.

- API Server: Exposes the Kubernetes API and is the entry point for managing the cluster.
- etcd: A distributed key-value store that stores configuration data for the cluster.
- Scheduler: Assigns work (containers) to nodes based on resource availability and constraints.
- Controller Manager: Monitors the state of the cluster and ensures that the desired state is maintained.
- Cloud Controller Manager: Manages interactions with cloud providers if the cluster is deployed on a cloud platform.

4. Pod: The smallest deployable unit in Kubernetes, which can contain one or more containers. Containers within the same pod share the same network namespace and can communicate with each other via `localhost`.

5. Deployment: A higher-level resource that defines how to manage the deployment of replica sets, which in turn manage the lifecycle of pods. Deployments enable rolling updates and rollbacks, ensuring application availability during updates.

6. Service: An abstraction that defines a set of pods and provides network access to them. Services enable load balancing and dynamic service discovery within the cluster.

7. ReplicaSet: Ensures that a specified number of pod replicas are running at all times. It is often managed by a deployment.

8. Namespace: A virtual cluster within the physical cluster that provides a way to divide cluster resources into logically named groups.

9. ConfigMap and Secret: Resources for managing configuration data and sensitive information, respectively, independently from application code.

10. Ingress: Manages external access to services within a cluster, often acting as a reverse proxy and providing features like SSL termination and routing.

11. Volume: An abstraction that allows pods to persist data beyond the lifetime of the containers within them.

Kubernetes offers numerous benefits, such as automating deployment processes, managing scaling and load balancing, providing self-healing capabilities, and supporting rolling updates and rollbacks without service disruptions. It has become an integral part of modern application deployment and infrastructure management, especially in cloud-native and containerized environments.

16. Microk8s

MicroK8s is a lightweight, minimalistic distribution of Kubernetes designed for local development, testing, and single-node deployments. It is developed by Canonical, the company behind Ubuntu Linux, and it aims to make it easier for developers to set up and manage a Kubernetes environment on their local machines or small servers.

Key features of MicroK8s include:

1. Easy Installation: MicroK8s can be installed with a single command, making it quick and convenient to set up a local Kubernetes environment.

2. Single-Node Cluster: MicroK8s is designed to run as a single-node Kubernetes cluster. This makes it well-suited for local development and testing scenarios.

3. Addon Management: MicroK8s provides a range of optional add-ons that can be easily enabled or disabled. These add-ons offer extra functionality like storage, networking, and monitoring.

4. Isolation: MicroK8s uses container technology to isolate its components from the host system, reducing interference and ensuring a clean development environment.

5. Snap Package: MicroK8s is distributed as a snap package, which includes all the required dependencies and allows for easy installation, upgrades, and rollback.

6. Rapid Updates: MicroK8s is known for its fast update cycle, making it easy to stay up-to-date with the latest Kubernetes features and improvements.

7. Local Testing: Developers can use MicroK8s to test their applications in an environment that closely resembles a production Kubernetes cluster.

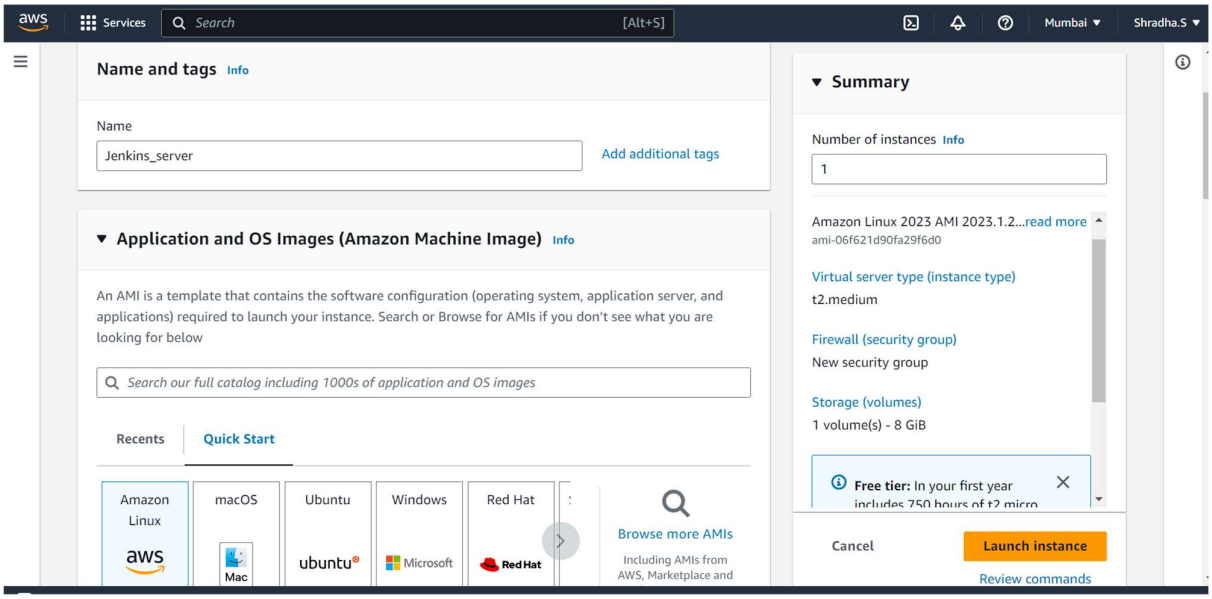
8. Support for CI/CD: MicroK8s can be used to set up continuous integration and continuous deployment (CI/CD) pipelines for testing and deploying applications in a Kubernetes environment.

9. Minimal Overhead: MicroK8s is optimized for minimal resource consumption and overhead, making it a good choice for resource-constrained systems.

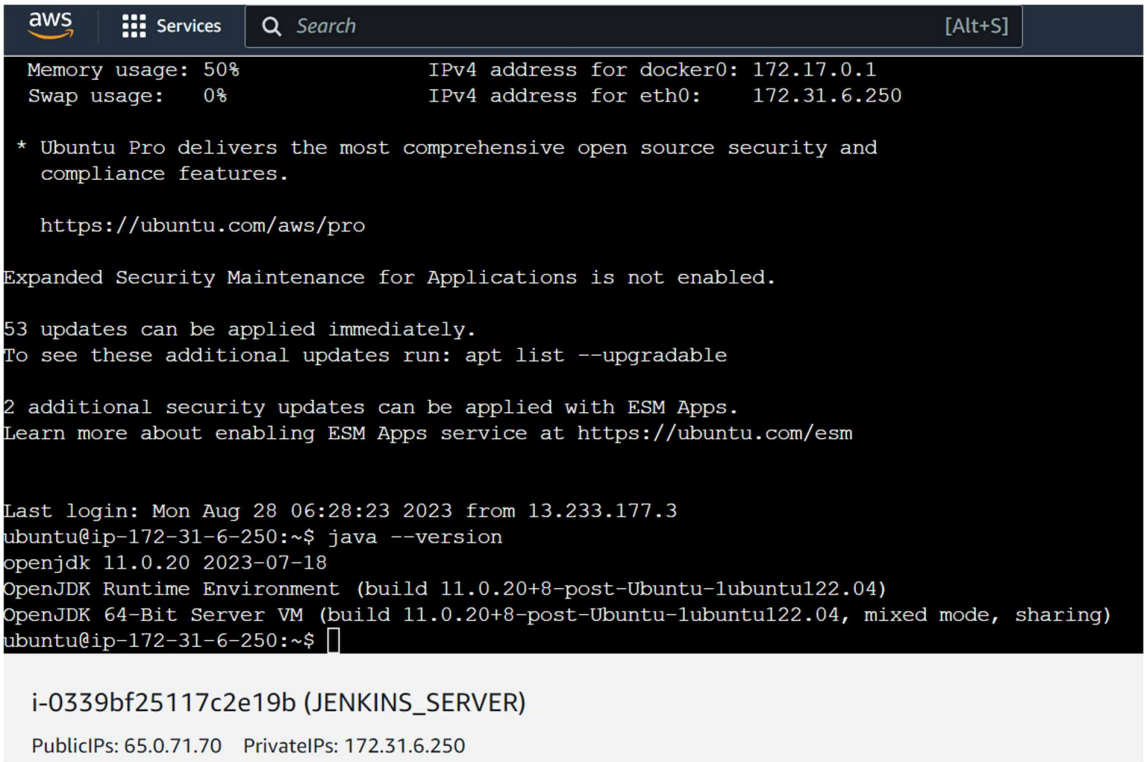
MicroK8s is particularly useful for developers who want to experiment with Kubernetes concepts, test their applications, or develop microservices locally before deploying them to a larger Kubernetes cluster. It's important to note that MicroK8s is not intended for running large-scale or production workloads due to its single-node nature. For production deployments, traditional Kubernetes distributions or managed Kubernetes services are recommended.

17. Implementation

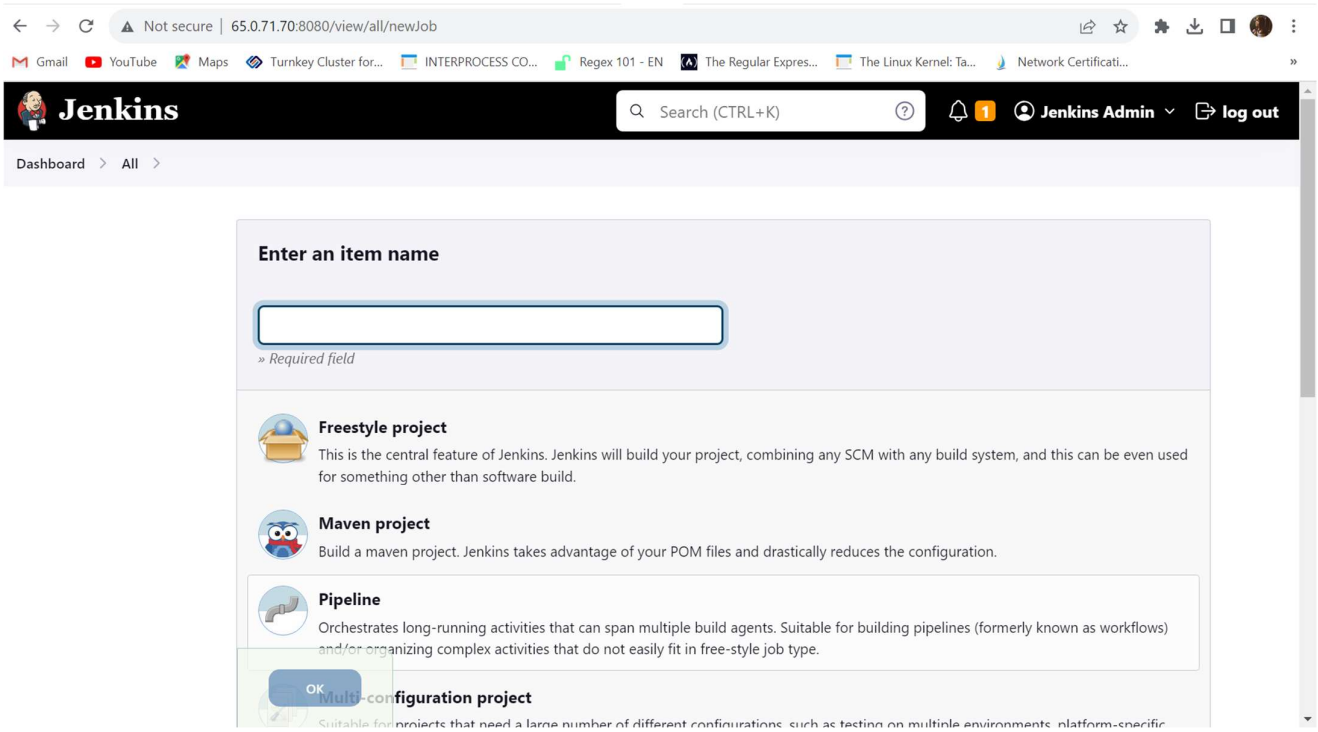
- Login onto the AWS console.
- Create EC2 instance for Jenkins Server.



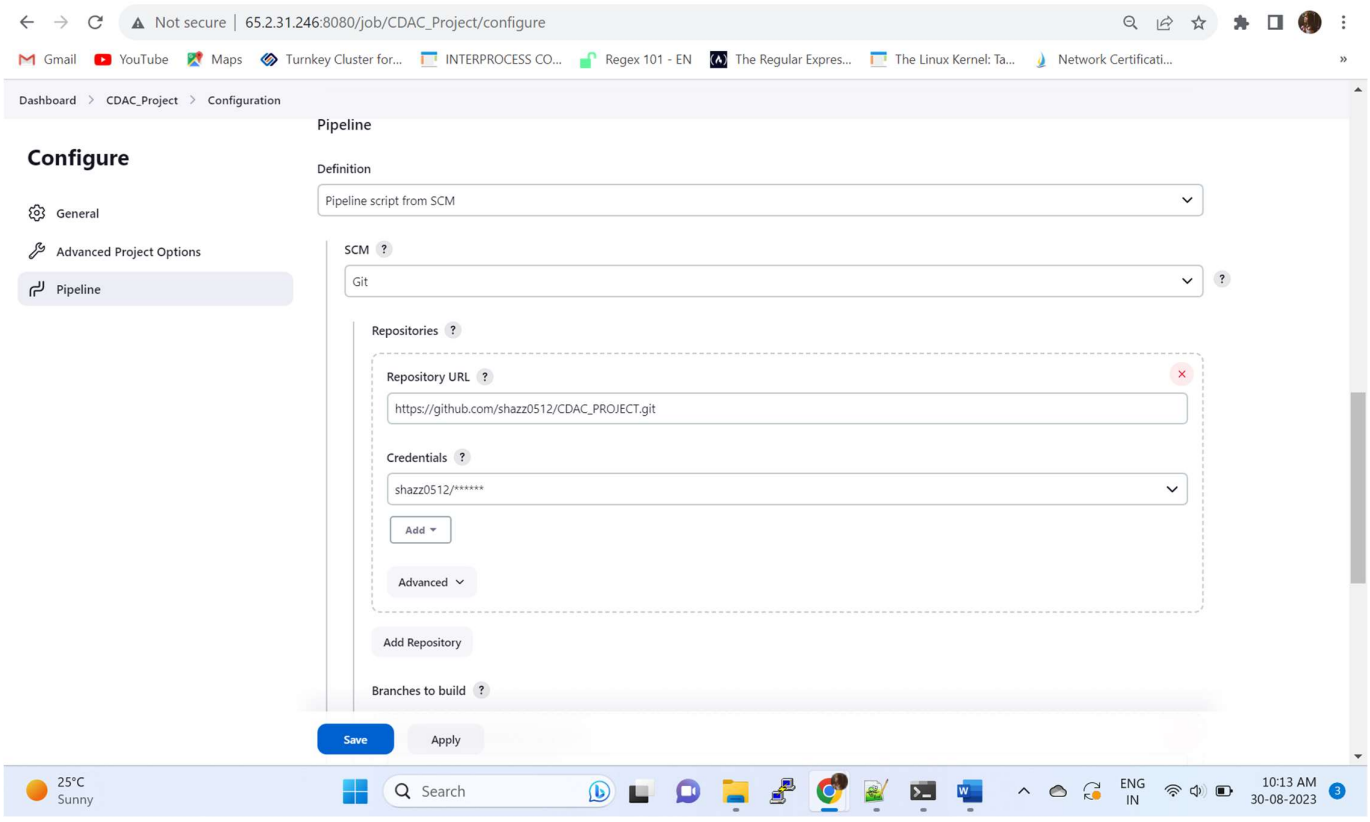
- **Install Jdk 11**

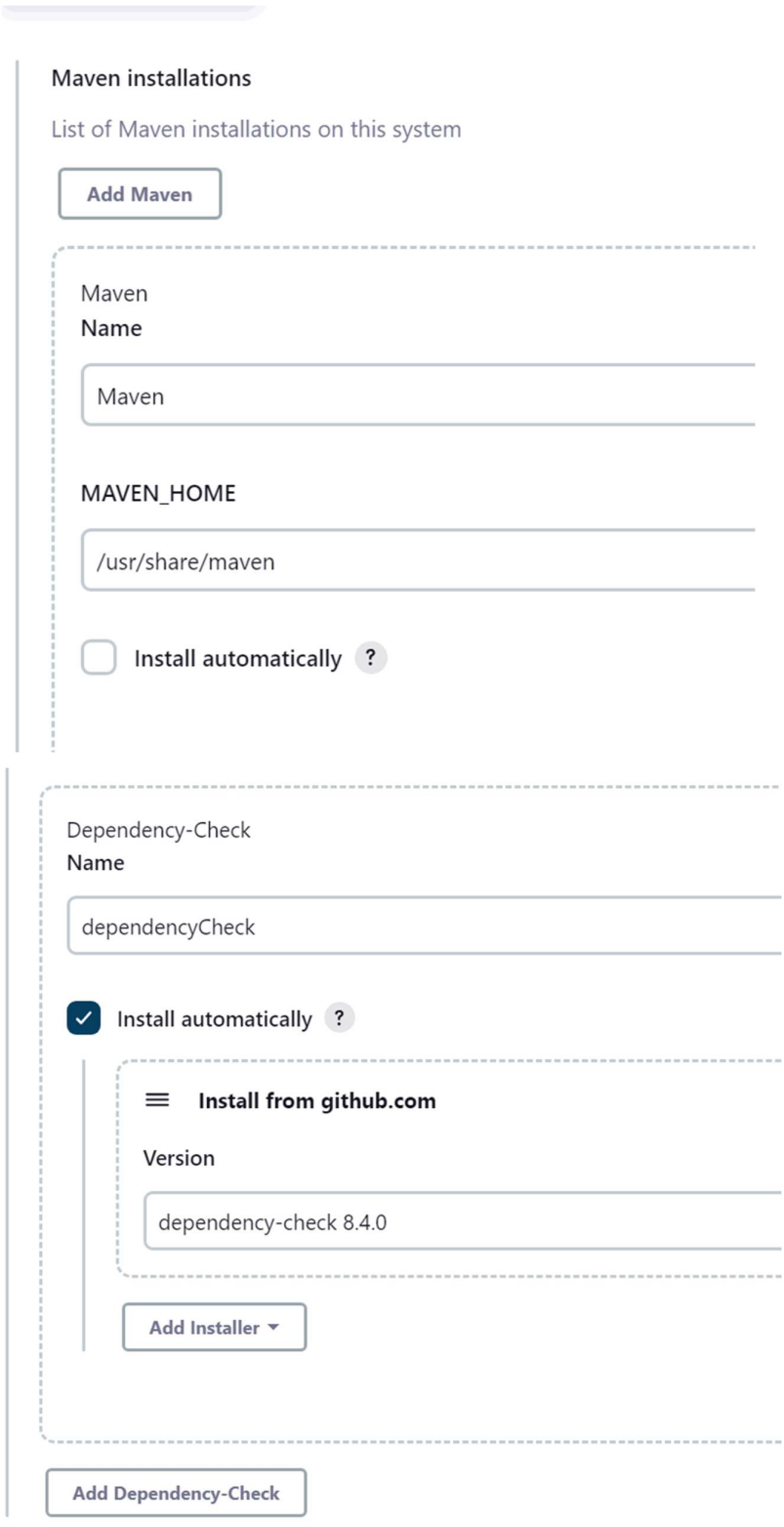


- curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
  echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
- sudo apt-get update
- sudo apt-get install Jenkins
- sudo systemctl enable Jenkins
- sudo systemctl status Jenkins

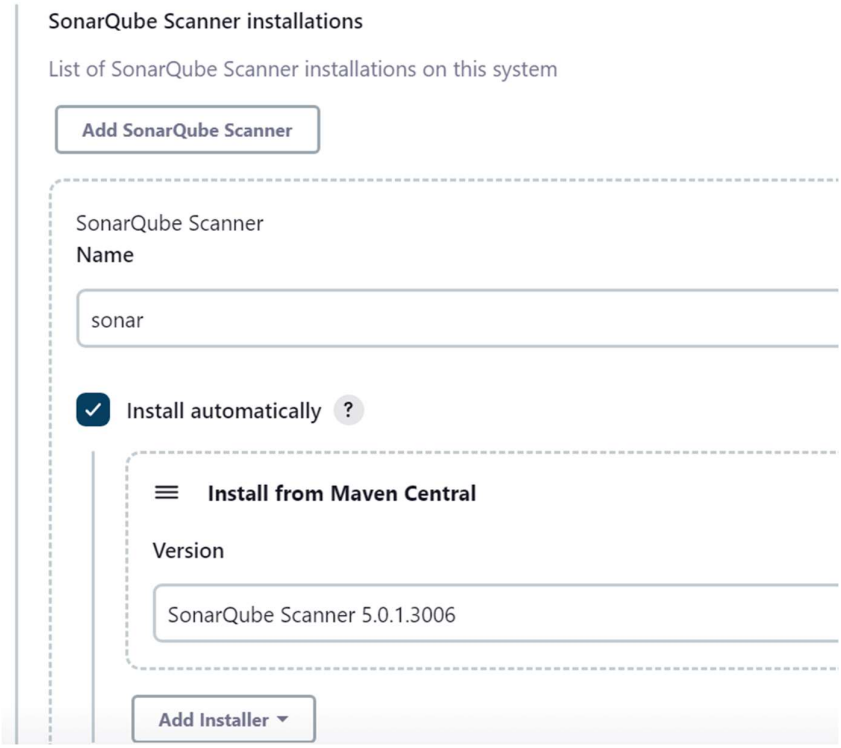


- **Configure Jenkins Job:**

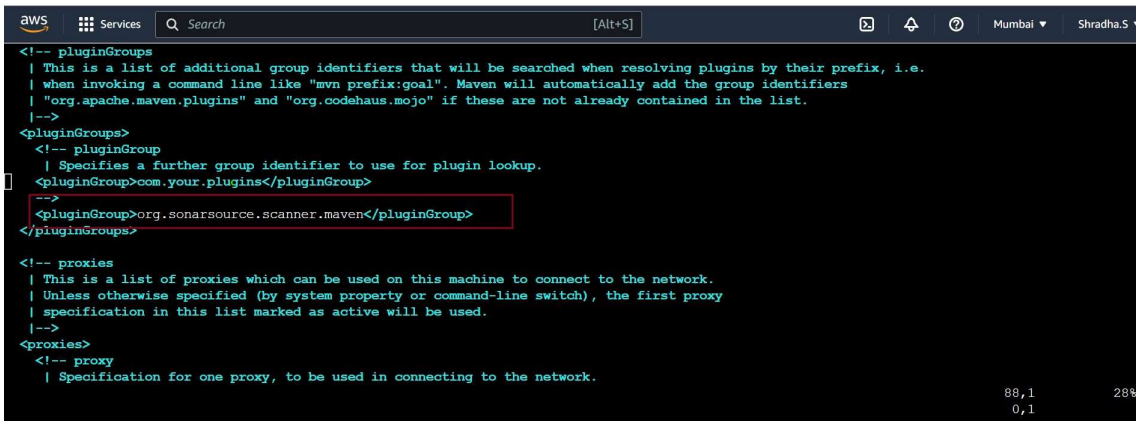


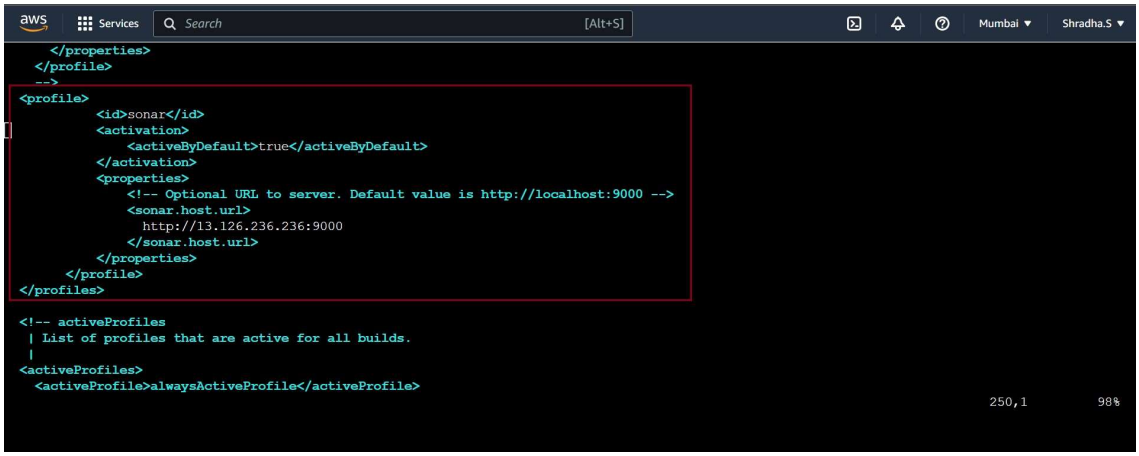

Maven installations
List of Maven installations on this system
Add Maven
Maven
Name
Maven
MAVEN_HOME
/usr/share/maven
Install automatically
Dependency-Check
Name
dependencyCheck
Install automatically
Install from github.com
Version
dependency-check 8.4.0
Add Installer
Add Dependency-Check

- **Configuring SonarQubes in Jenkins**

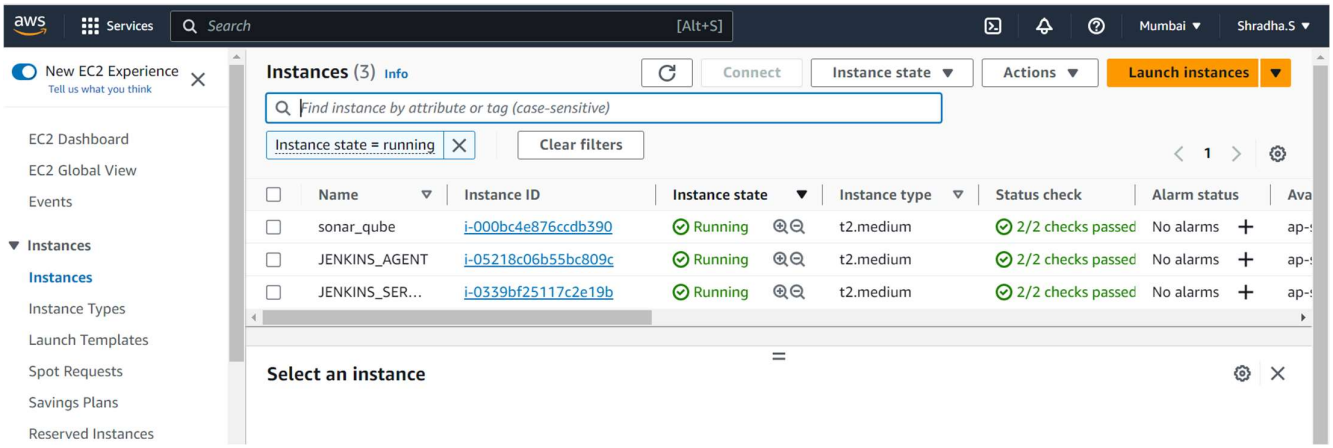


ubuntu@ip-172-31-6-250:/usr/share/maven/conf$ vi settings.xml

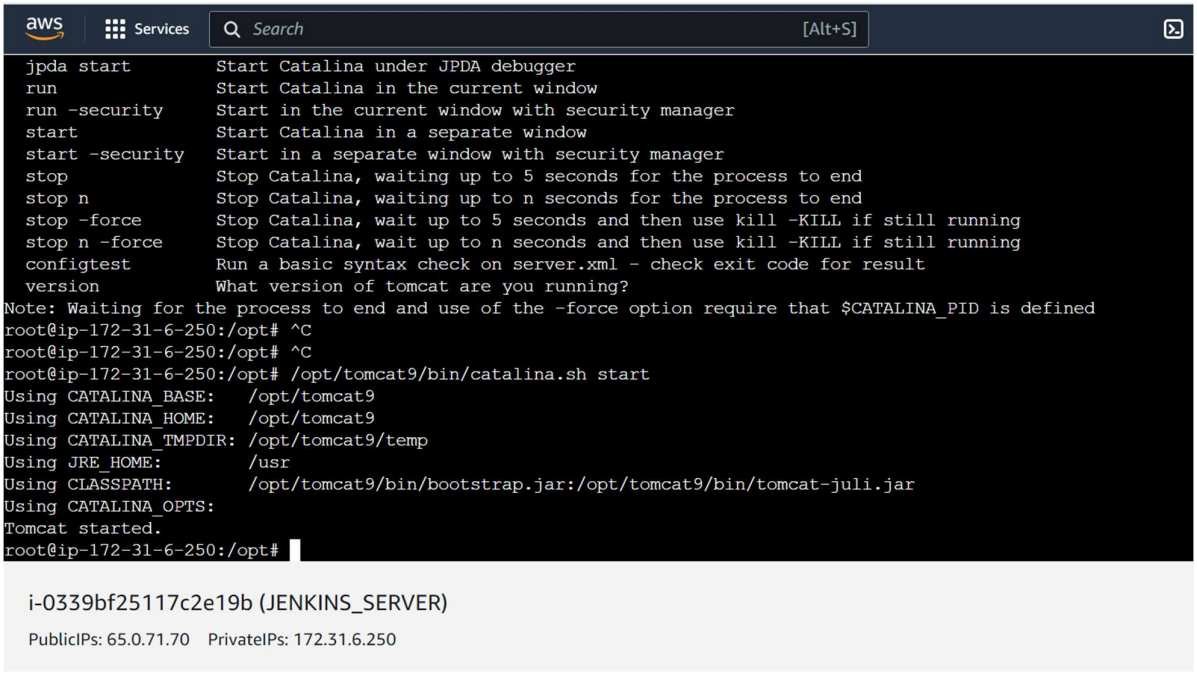




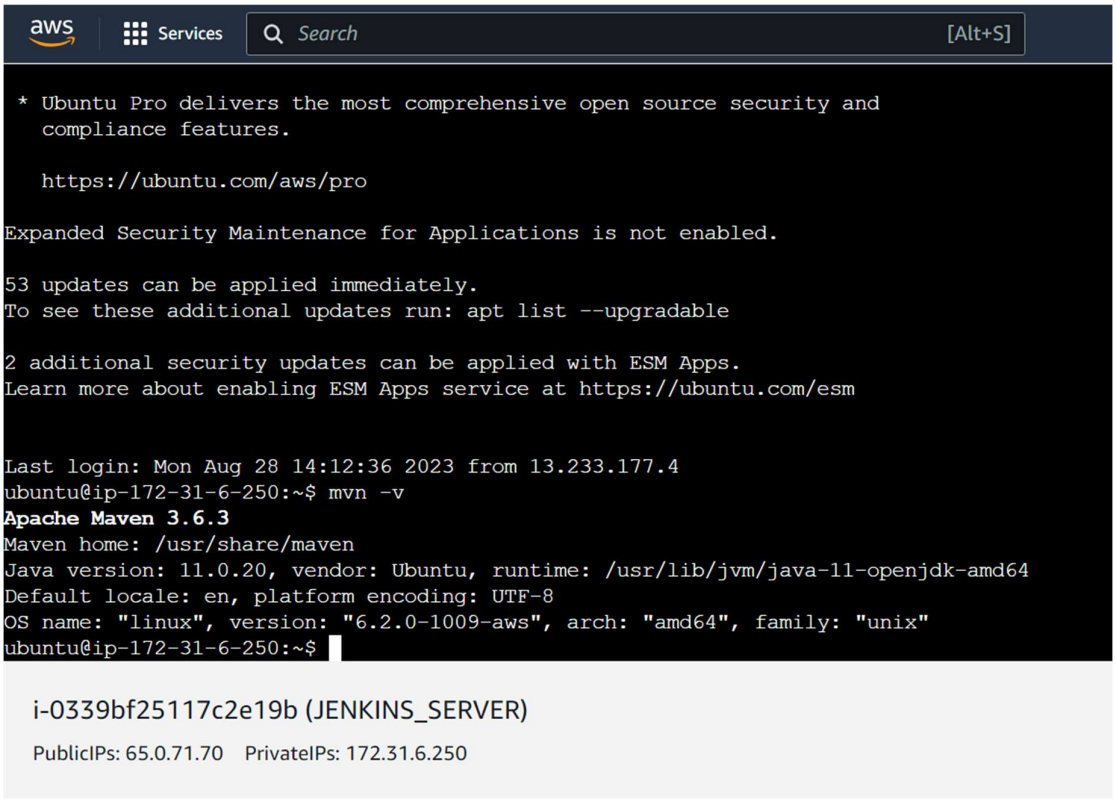
- **Created Similar 2 servers each for**



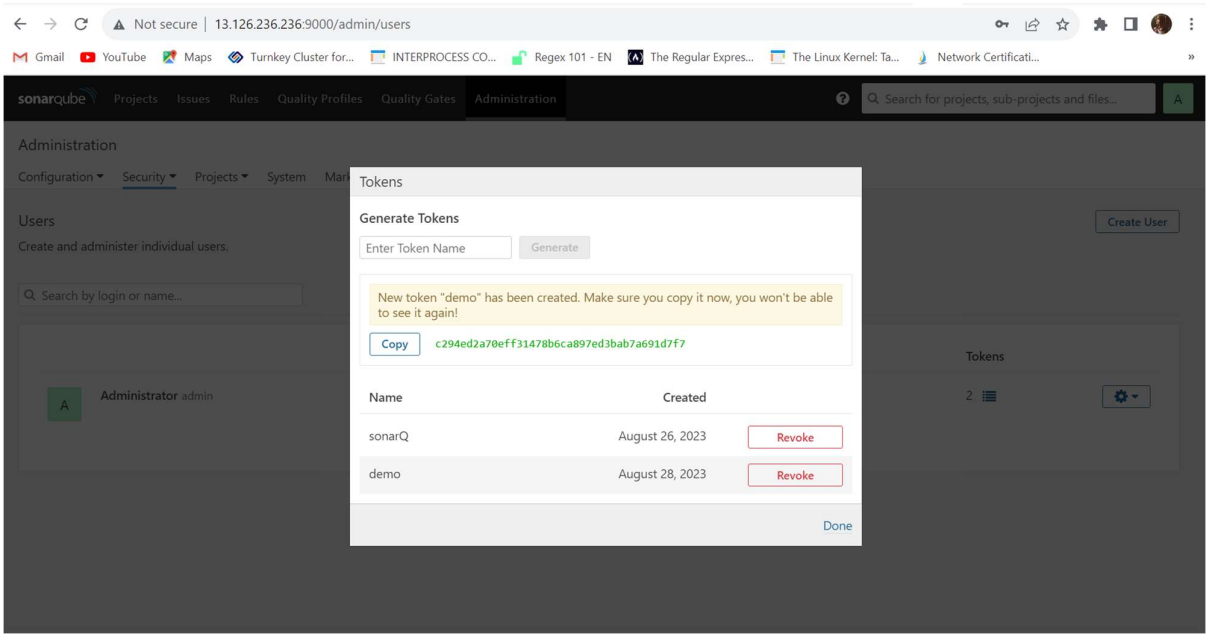
- **Install tomcat9**



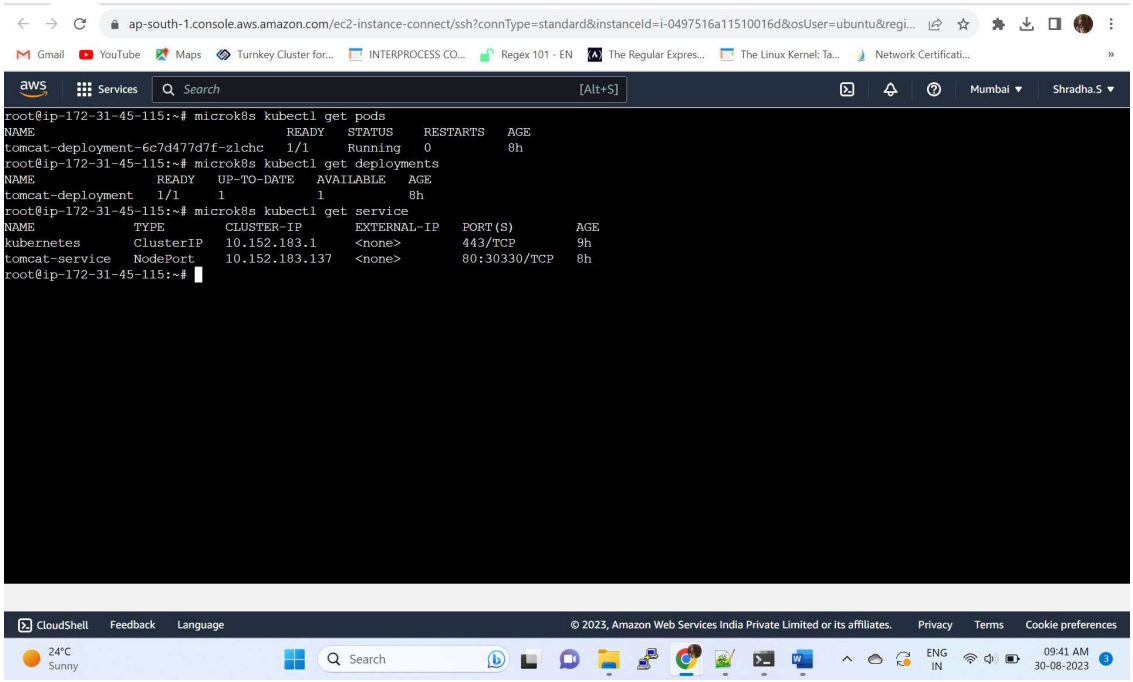
- **Add Maven in Jenkins**



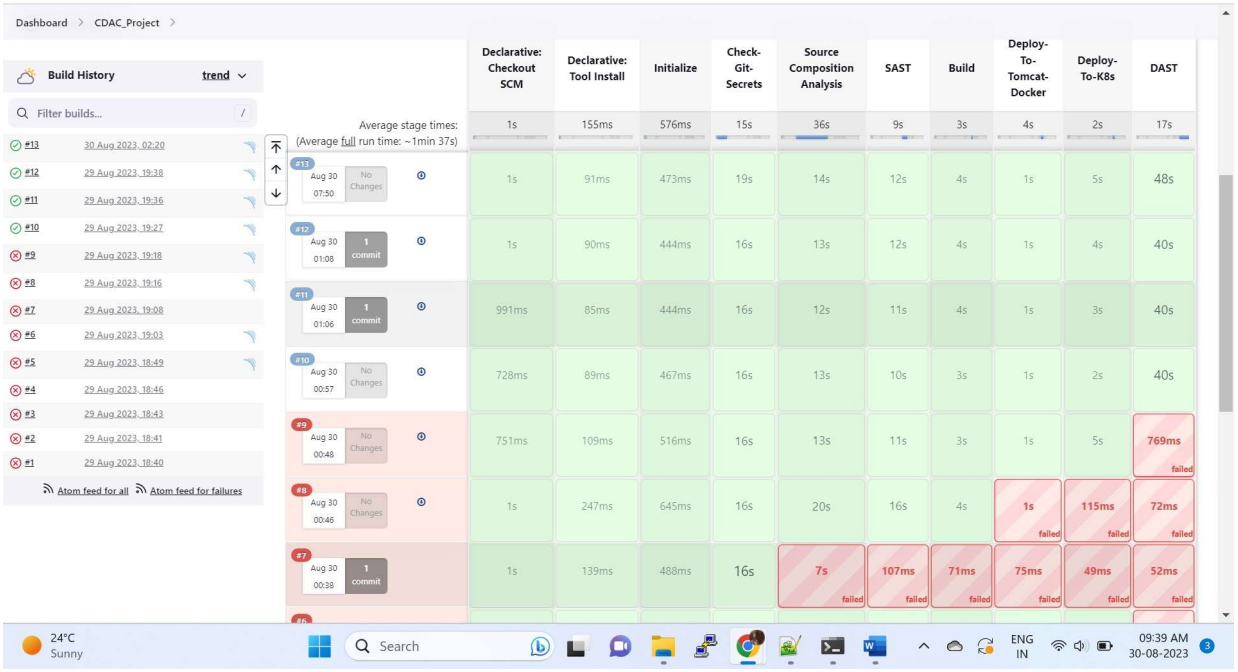
- **Generate Token in SonarQube**



- **Configuring Microk8s**

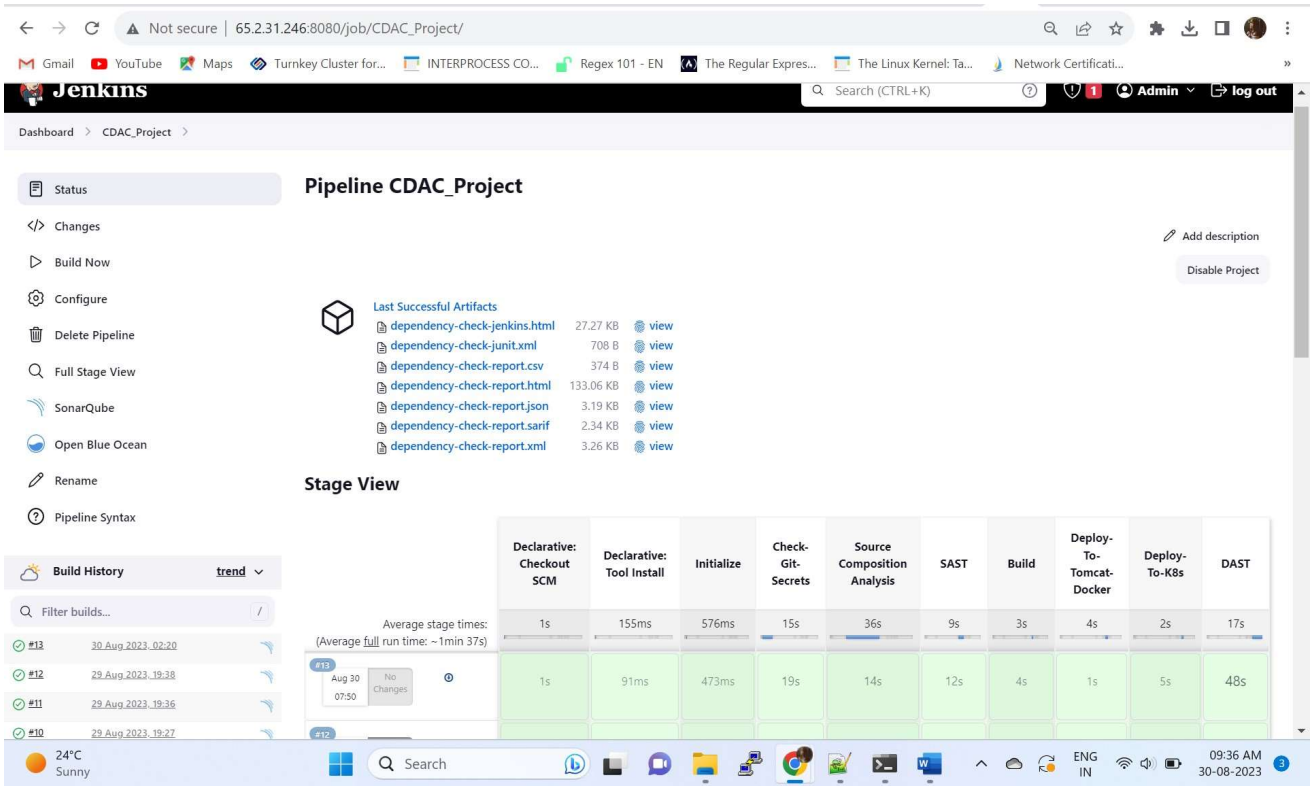


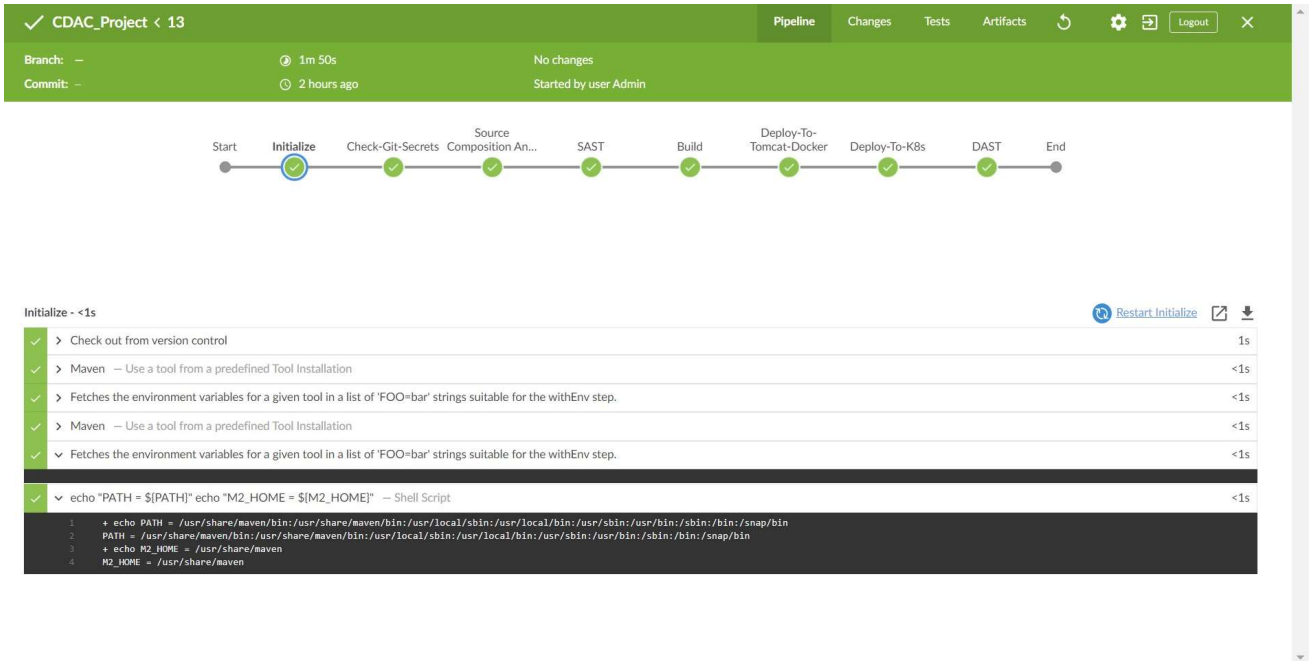
18. Testing



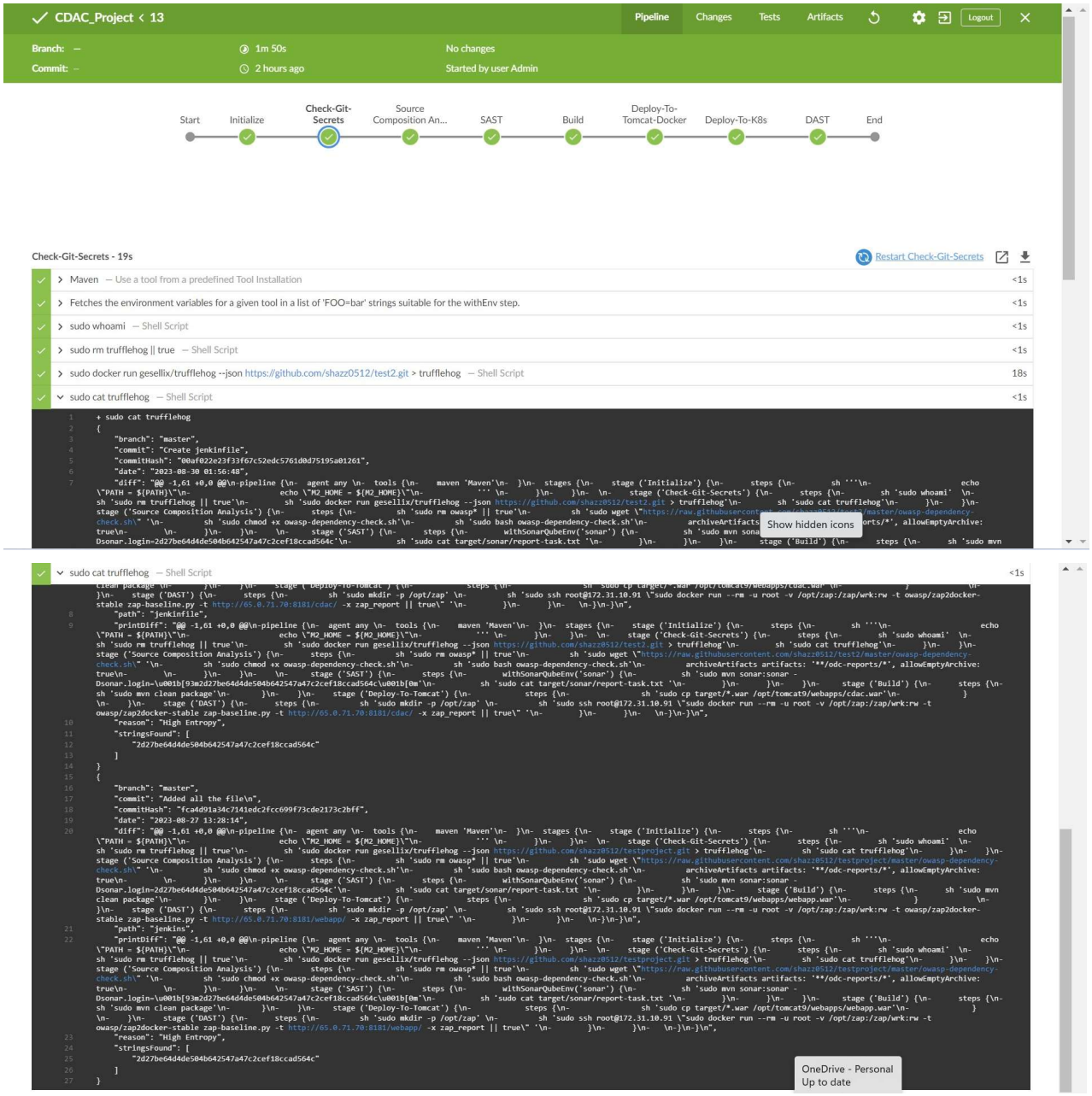
A test sample which shows BUILD FAILURE in build step

19. Result

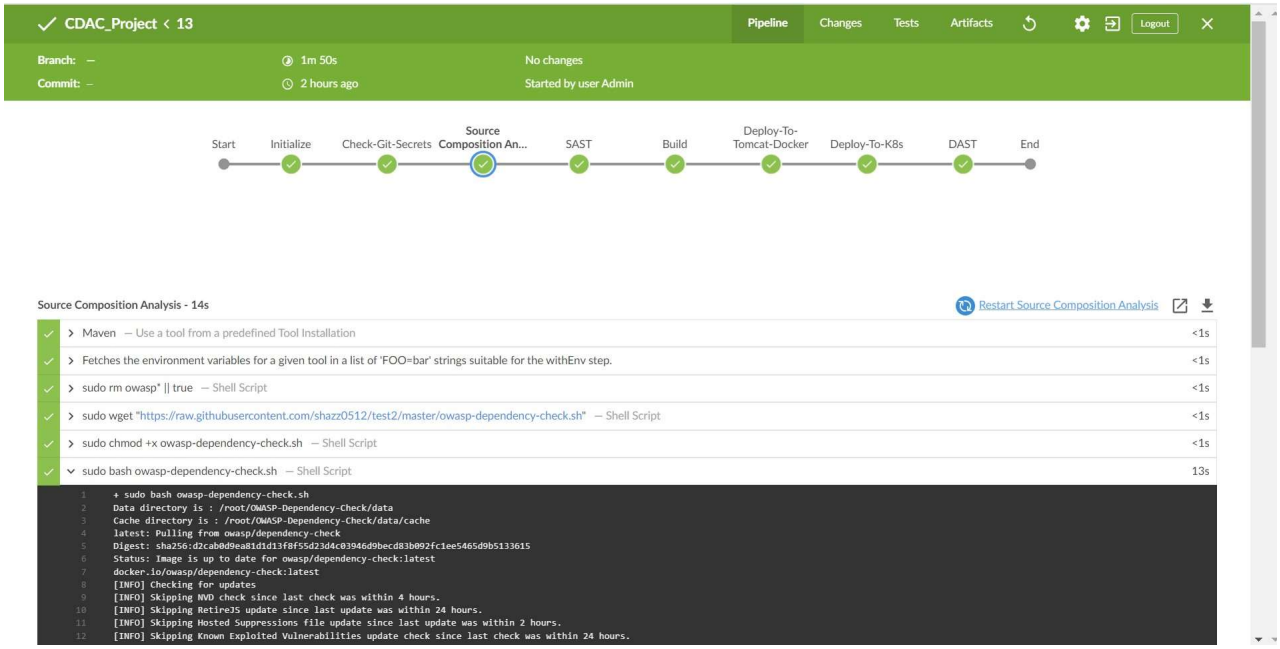

Not secure | 65.2.31.246:8080/job/CDAC_Project/
Jenkins
Dashboard > CDAC_Project
Status
Changes
Build Now
Configure
Delete Pipeline
Full Stage View
SonarQube
Open Blue Ocean
Rename
Pipeline Syntax
Build History
trend
Pipeline CDAC_Project
Add description
Disable Project
Last Successful Artifacts
dependency-check-jenkins.html 27.27 KB view
dependency-check-junit.xml 708 B view
dependency-check-report.csv 374 B view
dependency-check-report.html 133.06 KB view
dependency-check-report.json 3.19 KB view
dependency-check-report.sarif 2.34 KB view
dependency-check-report.xml 3.26 KB view
Stage View
Declarative: Checkout SCM
Declarative: Tool Install
Initialize
Check-Git-Secrets
Source Composition Analysis
SAST
Build
Deploy-To-Tomcat-Docker
Deploy-To-K8s
DAST
Average stage times:
(Average full run time: ~1min 37s)
1s 155ms 576ms 15s 36s 9s 3s 4s 2s 17s
#13 Aug 30 07:50 No Changes
1s 91ms 473ms 19s 14s 12s 4s 1s 5s 48s
24°C Sunny
Search
ENG IN
09:36 AM 30-08-2023


CDAC_Project < 13
Pipeline
Changes
Tests
Artifacts
Logout
Branch: –
Commit: –
1m 50s
2 hours ago
No changes
Started by user Admin
Start
Initialize
Check-Git-Secrets
Source Composition An...
SAST
Build
Deploy-To-Tomcat-Docker
Deploy-To-K8s
DAST
End
Initialize - <1s
Restart Initialize
Check out from version control 1s
Maven – Use a tool from a predefined Tool Installation <1s
Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step. <1s
Maven – Use a tool from a predefined Tool Installation <1s
Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step. <1s
echo "PATH = ${PATH}" echo "M2_HOME = ${M2_HOME}" – Shell Script <1s
+ echo PATH = /usr/share/maven/bin:/usr/share/maven/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
PATH = /usr/share/maven/bin:/usr/share/maven/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
+ echo M2_HOME = /usr/share/maven
M2_HOME = /usr/share/maven

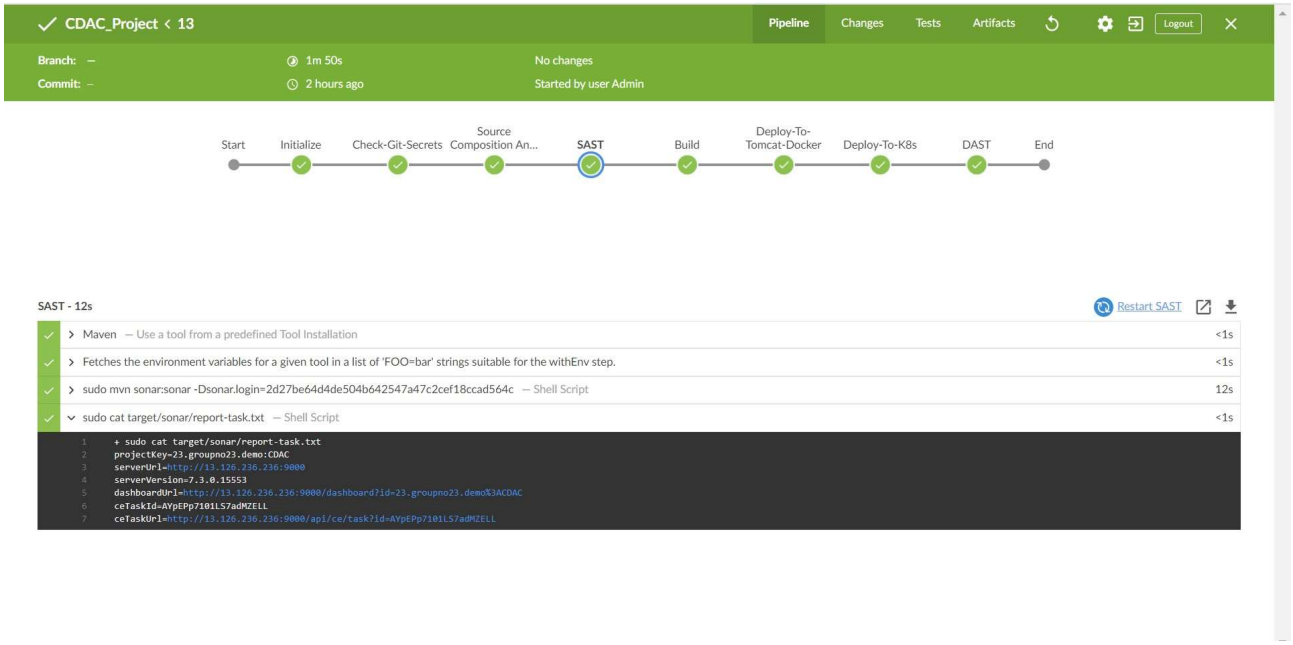
Output of trufflehog tool (git secrets)

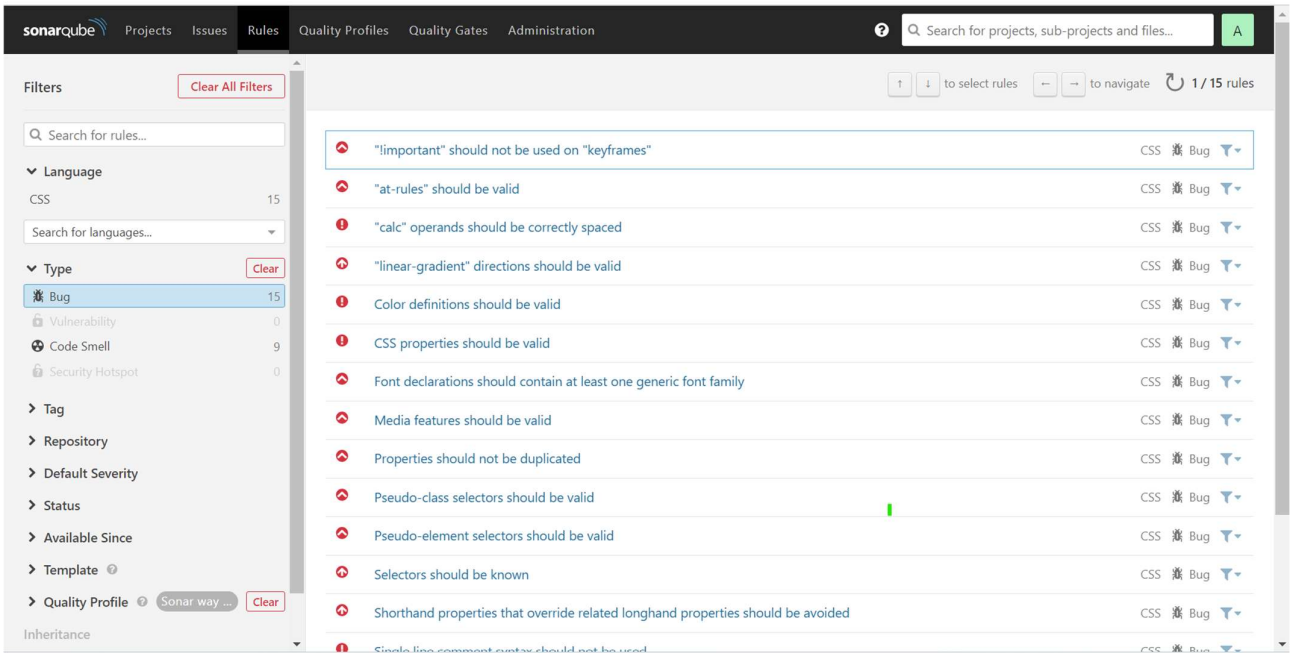


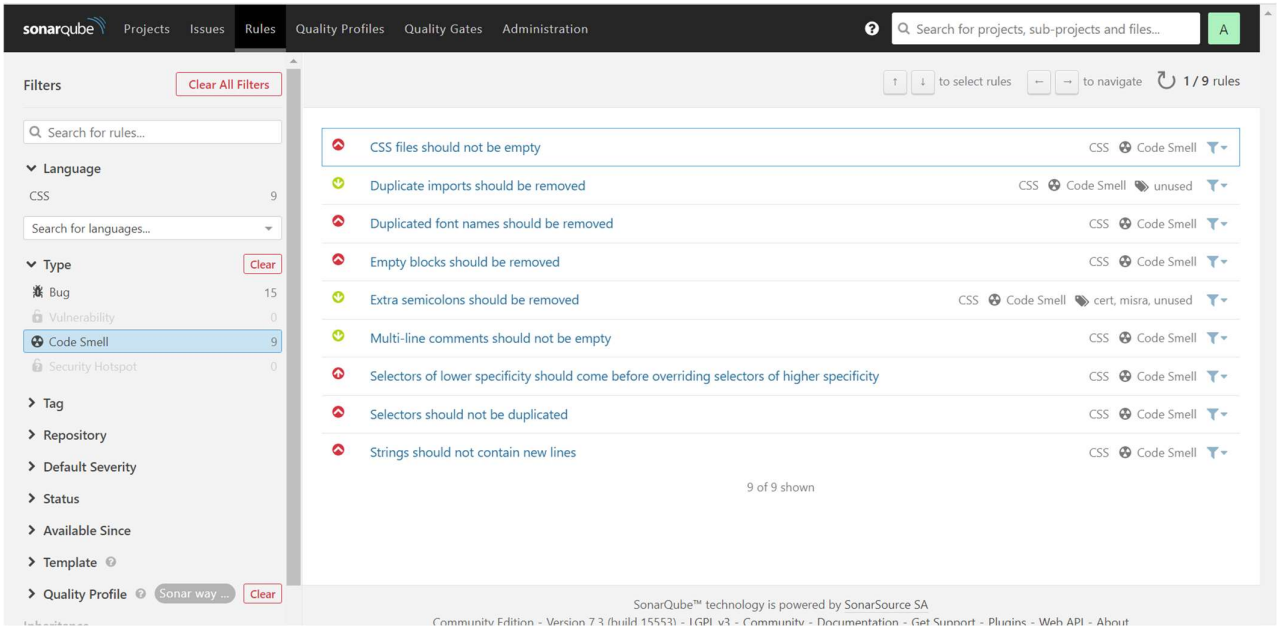
Output of OSWAP Dependency check (SCA)

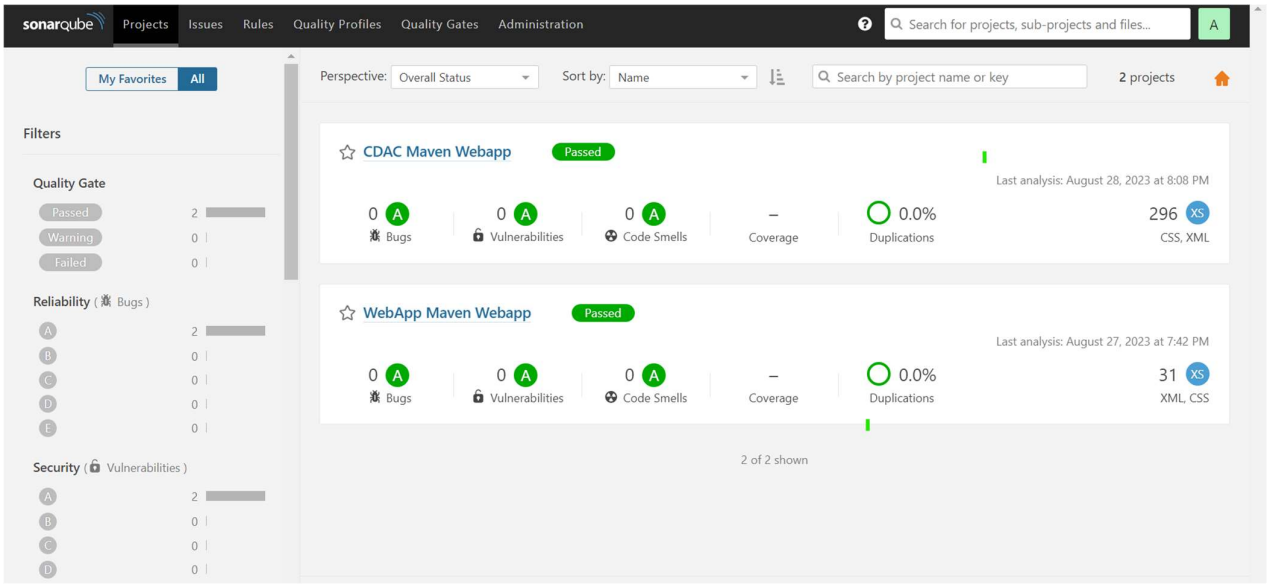


Output of SAST(Sonarqube)

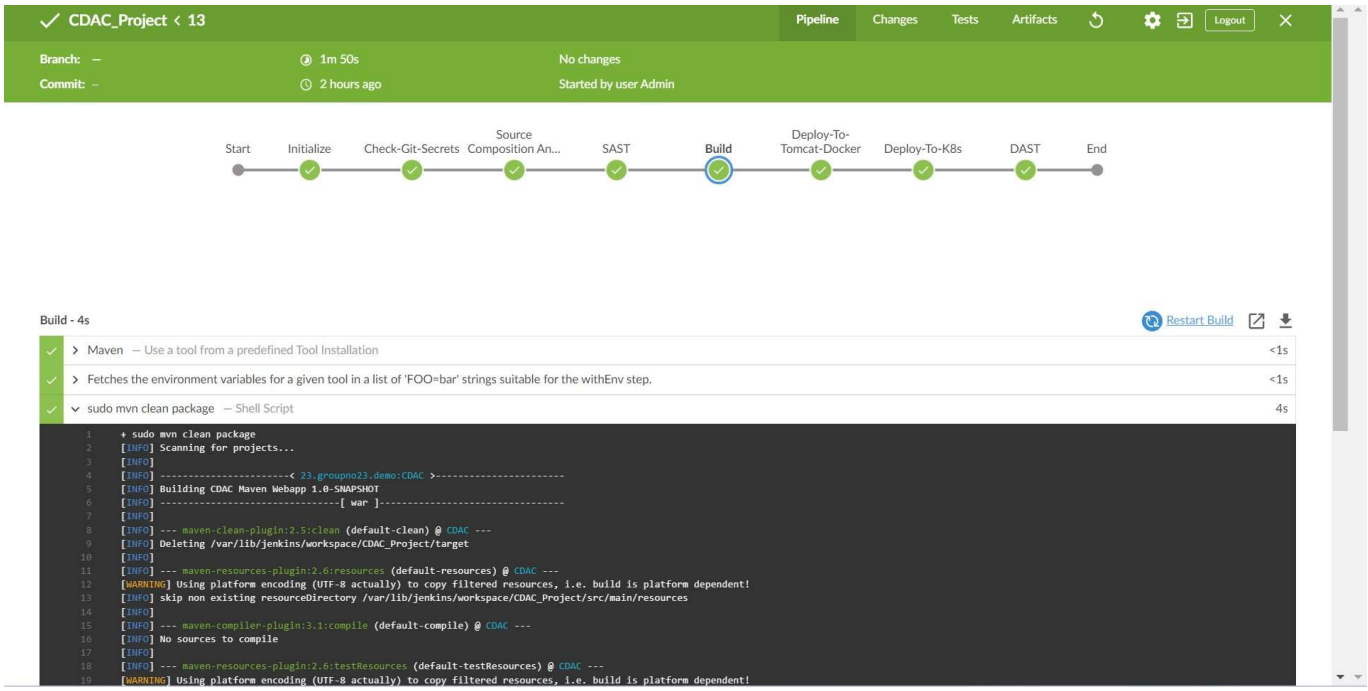


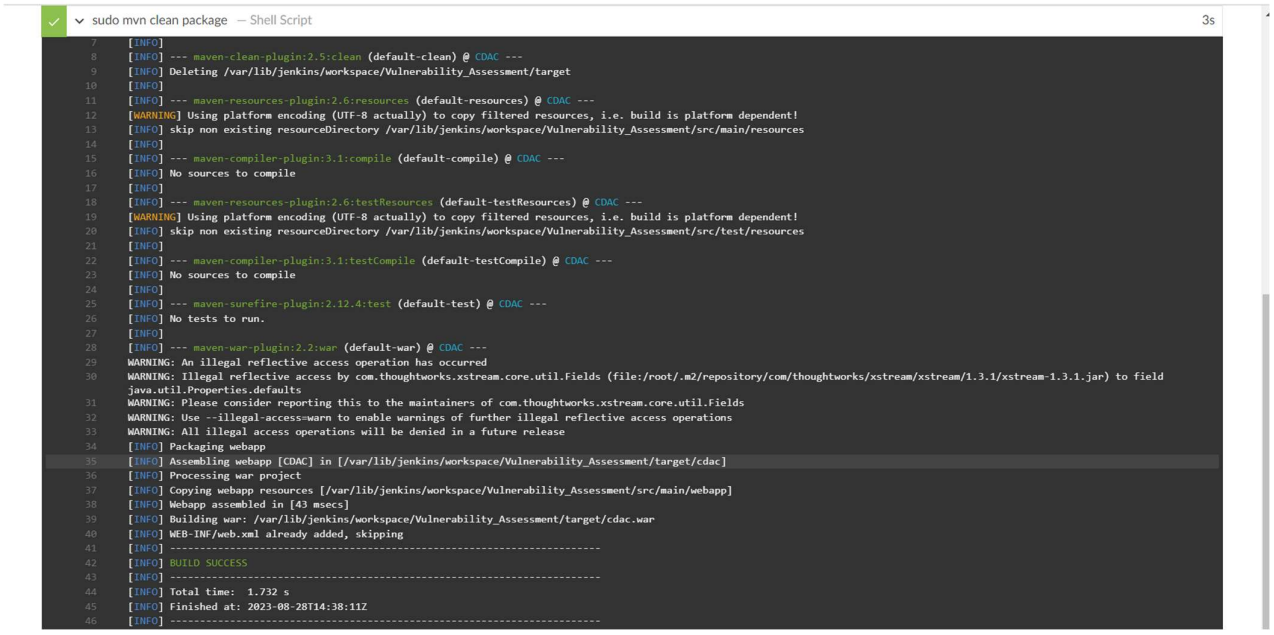

sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
Search for projects, sub-projects and files...
A
Filters
Clear All Filters
Search for rules...
Language
CSS 15
Search for languages...
Type
Clear
Bug 15
Vulnerability 0
Code Smell 9
Security Hotspot 0
Tag
Repository
Default Severity
Status
Available Since
Template
Quality Profile
Sonar way ...
Clear
Inheritance
to select rules
to navigate
1 / 15 rules
"!important" should not be used on "keyframes" CSS Bug
"at-rules" should be valid CSS Bug
"calc" operands should be correctly spaced CSS Bug
"linear-gradient" directions should be valid CSS Bug
Color definitions should be valid CSS Bug
CSS properties should be valid CSS Bug
Font declarations should contain at least one generic font family CSS Bug
Media features should be valid CSS Bug
Properties should not be duplicated CSS Bug
Pseudo-class selectors should be valid CSS Bug
Pseudo-element selectors should be valid CSS Bug
Selectors should be known CSS Bug
Shorthand properties that override related longhand properties should be avoided CSS Bug


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
Search for projects, sub-projects and files...
A
Filters
Clear All Filters
Search for rules...
Language
CSS 9
Search for languages...
Type
Clear
Bug 15
Vulnerability 0
Code Smell 9
Security Hotspot 0
Tag
Repository
Default Severity
Status
Available Since
Template
Quality Profile
Sonar way ...
Clear
to select rules
to navigate
1 / 9 rules
CSS files should not be empty CSS Code Smell
Duplicate imports should be removed CSS Code Smell unused
Duplicated font names should be removed CSS Code Smell
Empty blocks should be removed CSS Code Smell
Extra semicolons should be removed CSS Code Smell cert, misra, unused
Multi-line comments should not be empty CSS Code Smell
Selectors of lower specificity should come before overriding selectors of higher specificity CSS Code Smell
Selectors should not be duplicated CSS Code Smell
Strings should not contain new lines CSS Code Smell
9 of 9 shown
SonarQube™ technology is powered by SonarSource SA
Community Edition - Version 7.3 (build 15553) - LGPL v3 - Community - Documentation - Get Support - Plugins - Web API - About

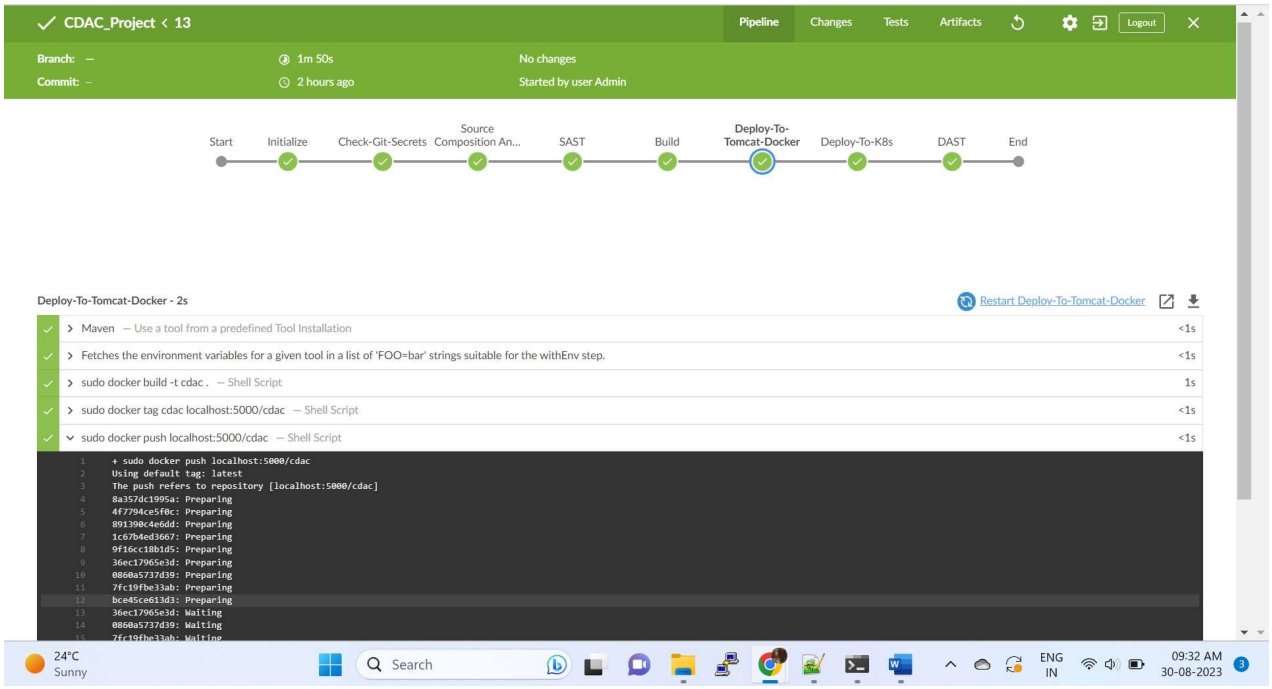


Output of Maven

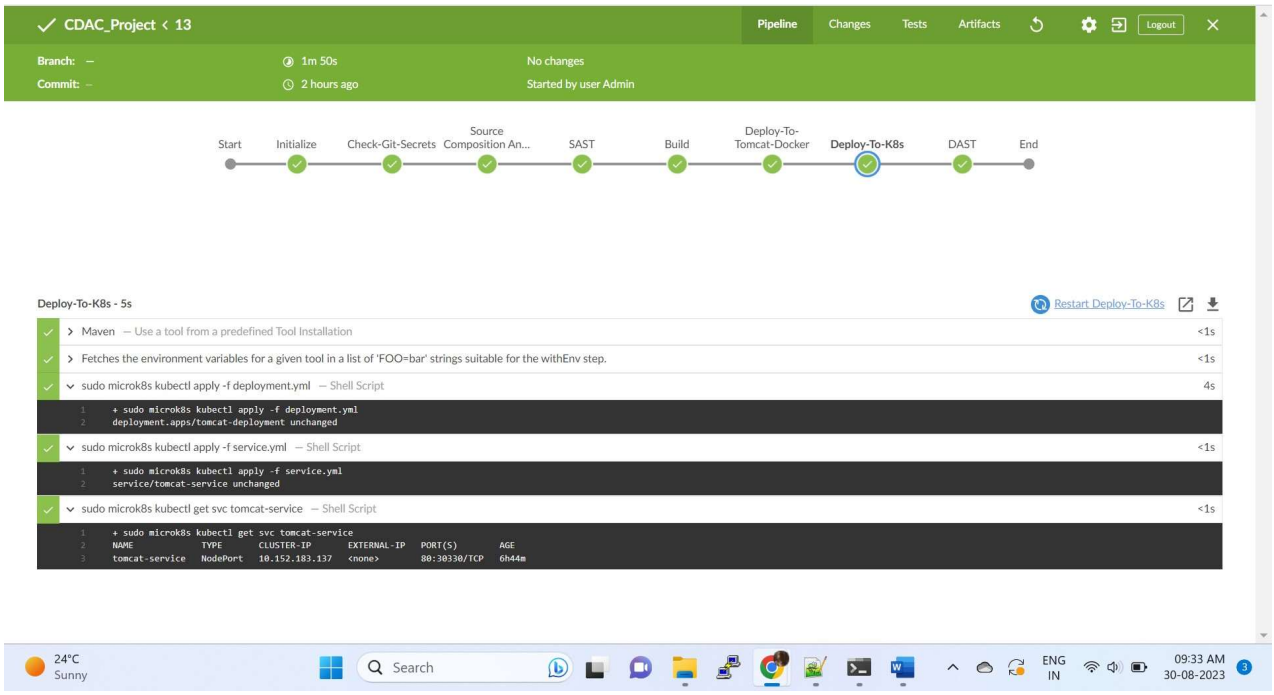




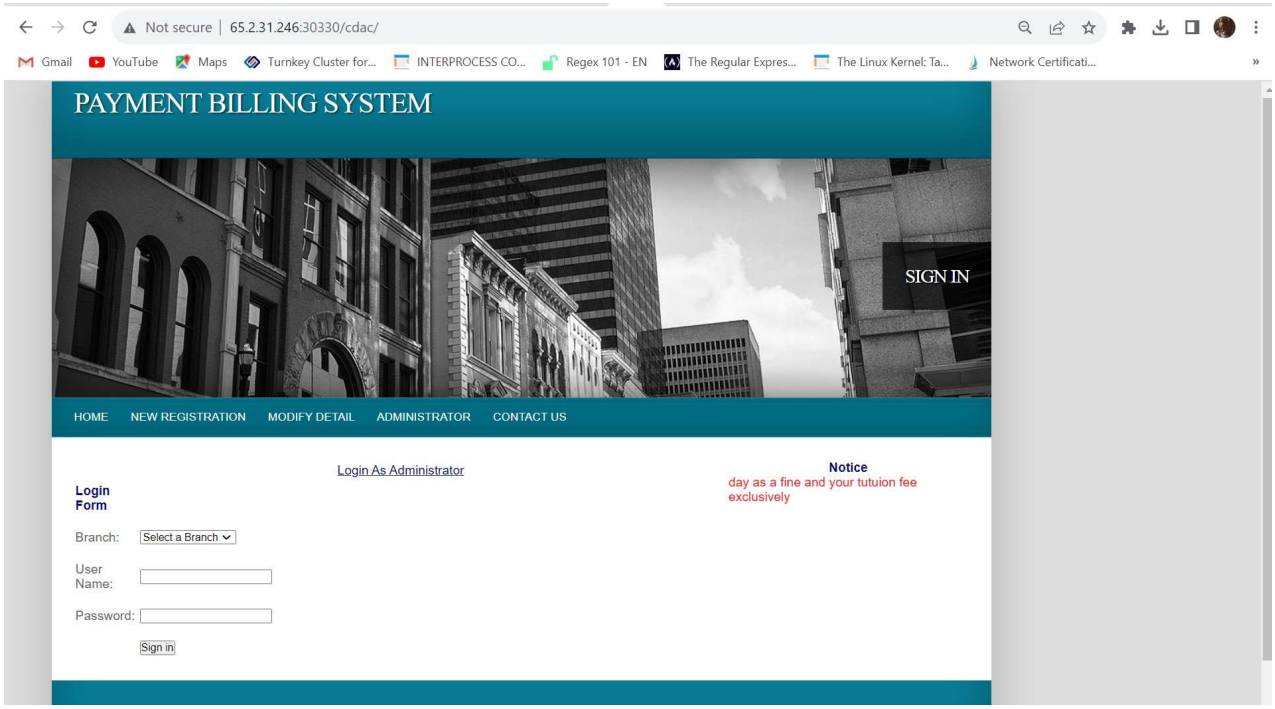
Output of Deploy to Tomcat to local docker Registry



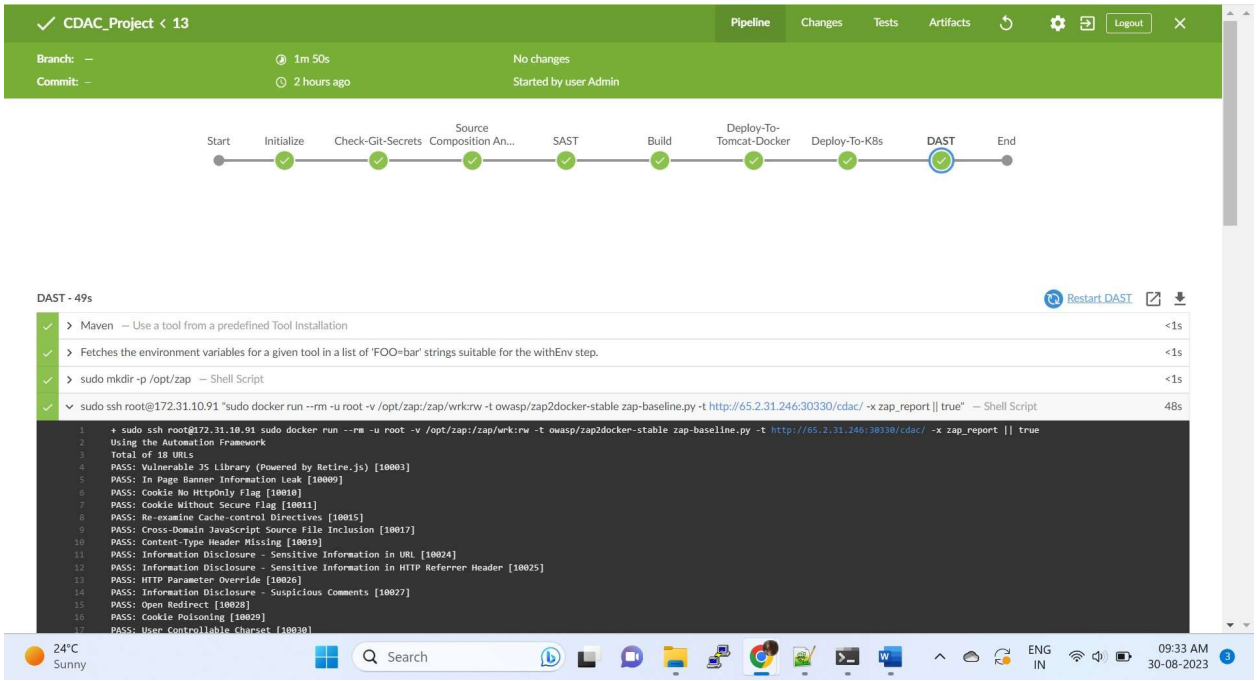
Output of Deploy to K8s

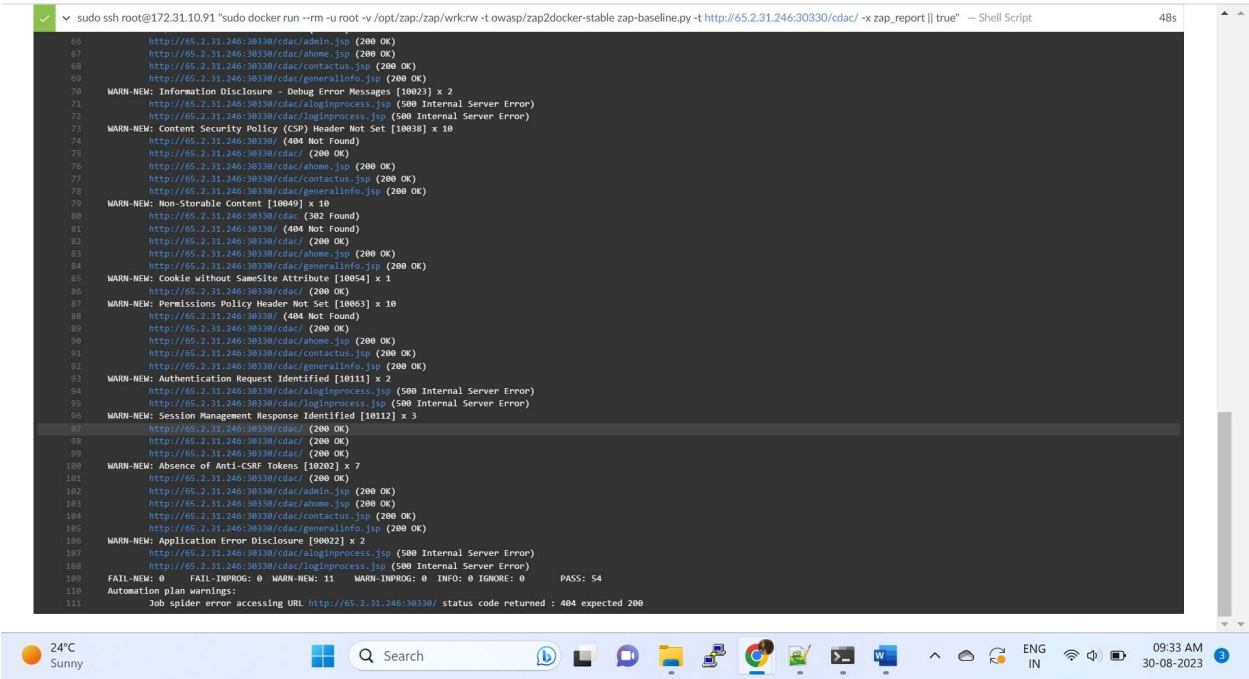


Application is deployed



Output of ZAP





We got to see 12 warnings at the end of the ZAP Analysis.

Hence Successfully implemented Vulnerability Assessment using DevSecOps.

20. CODE

Jenkinsfile

```
pipeline {
  agent any
  tools {
    maven 'Maven'
  }
  stages {
    stage ('Initialize') {
      steps {
        sh '''
                echo "PATH = ${PATH}"
                echo "M2_HOME = ${M2_HOME}"
            '''
      }
    }

    stage ('Check-Git-Secrets') {
      steps {
        sh 'sudo whoami'
        sh 'sudo rm trufflehog || true'
        sh 'sudo docker run gesellix/trufflehog --json https://github.com/shazz0512/test2.git > trufflehog'
        sh 'sudo cat trufflehog'
      }
    }
    stage ('Source Composition Analysis') {
      steps {
        sh 'sudo rm owasp* || true'
        sh 'sudo wget "https://raw.githubusercontent.com/shazz0512/test2/master/owasp-dependency-check.sh" '
        sh 'sudo chmod +x owasp-dependency-check.sh'
        sh 'sudo bash owasp-dependency-check.sh'
        archiveArtifacts artifacts: '**/odc-reports/*', allowEmptyArchive: true
```

```
  }
}

 stage ('SAST') {
  steps {
    withSonarQubeEnv('sonar') {
      sh 'sudo mvn sonar:sonar -Dsonar.login=2d27be64d4de504b642547a47c2cef18ccad564c'
      sh 'sudo cat target/sonar/report-task.txt '
    }
  }
}
 stage ('Build') {
  steps {
  sh 'sudo mvn clean package'
   }
}
stage ('Deploy-To-Tomcat-Docker') {
        steps {
           sh 'sudo docker build -t cdac .'
           sh 'sudo docker tag cdac localhost:5000/cdac'
           sh 'sudo docker push localhost:5000/cdac'
         }
}
stage ('Deploy-To-K8s') {
        steps {
          sh 'sudo microk8s kubectl apply -f deployment.yml'
          sh 'sudo microk8s kubectl apply -f service.yml'
          sh 'sudo microk8s kubectl get svc tomcat-service'
         }
}

stage ('DAST') {
  steps {
    sh 'sudo mkdir -p /opt/zap'
    sh 'sudo ssh root@172.31.10.91 "sudo docker run --rm -u root -v /opt/zap:/zap/wrk:rw -t
```

owasp/zap2docker-stable zap-baseline.py -t http://65.2.31.246:30330/cdac/ -x zap_report || true" '

}

}

}

}

Dockerfile

FROM tomcat:latest

COPY target/cdac.war /usr/local/tomcat/webapps/

EXPOSE 8080

CMD ["catalina.sh", "run"]

Deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: localhost:5000/cdac:latest
        ports:
        - containerPort: 8080
```

Service.yml

```yaml
kind: Service
apiVersion: v1
metadata:
```

```
  name: tomcat-service
spec:
  selector:
    app: tomcat
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
  type: NodePort
```

21. Conclusion:

In conclusion, the project "Vulnerability Assessment using DevSecOps" bridges the critical gap between security and development by integrating security practices seamlessly into the DevOps workflow. This project showcases the synergy between development and security teams to proactively identify and mitigate vulnerabilities throughout the software development lifecycle. By adopting a DevSecOps approach, organizations enhance their ability to identify and address security weaknesses early in the development process, reducing the potential for security breaches and minimizing the associated risks.

22. Reference

1. An Introduction to CI/CD Best Practices. https://www.digitalocean. com/community/tutorials/an-introduction-to-cicd-best-practices. [Online; accessed 14-May-2020]. 2018.
2. CVE security vulnerability database. Security vulnerabilities, exploits, references and more. https://www.cvedetails.com/. [Online; accessed 1-June-2020]. 2020.
3. CWE-531: Inclusion of Sensitive Information in Test Code. https:// cwe.mitre.org/data/definitions/531.html. [Online; accessed 17-May-2020]. 2020.
4. CWE-94: Improper Control of Generation of Code ('Code Injection'). https://cwe.mitre.org/data/definitions/94.html. [Online; accessed 17-May-2020]. 2020.
5. CWE311: Missing encryption of sensitive data. https : / / cwe . mitre . org / data / definitions / 311 . html. [Online; accessed 14-May-2020]. 2020.
6. Martin Fowler. Continuous Delivery. 2013. url: https://martinfowler. com/bliki/ContinuousDelivery.html (visited on 05/18/2020).
7. Martin Fowler. Deployment Pipeline. 2013. url: https://martinfowler. com/bliki/DeploymentPipeline.html (visited on 04/06/2020).
8. Volker Gruhn, Christoph Hannebauer, and Christian John. "Security of public continuous integration services". In: Proceedings of the 9th International Symposium on open collaboration. 2013, pp. 1–10.
9. Red Hat. What is DevSecOps. url: https://www.redhat.com/ en/topics/devops/what-is-devsecops (visited on 05/17/2020).
10 Identify vulnerabilities in third-party software libraries. https:// attack.mitre.org/techniques/T1389/. [Online; accessed 21-May-2020]. 2018.
11 Identify vulnerabilities in third-party software libraries. https:// attack.mitre.org/techniques/T1389/. [Online; accessed 21-May-2020]. 2018.
12 Christina Paule. "Securing DevOps: detection of vulnerabilities in CD pipelines". MA thesis. 2018.
13 Install Java : C:\Users\Desktop\How To Install Java with Apt-Get on Ubuntu 16.04 _ DigitalOcean_files
14 https://wiki.jenkins.io/display/JENKINS/Installing+Jenkins+on+Ubuntu.