

UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
DIRECCIÓN DE POSGRADO



“LABORATORIO NRO. 4 IMPLEMENTACIÓN CI CON JENKINS, GIT, GRYPE Y SONARQUBE”

***“DIPLOMADO EN EXPERTO EN DESARROLLO DE APLICACIONES
EMPRESARIALES – 5ta. VERSIÓN”***

POSTULANTE: JHONATAN MAMANI MENDOZA

DOCENTE: ING. RUDY SALVATIERRA RODRIGUEZ

**Cochabamba – Bolivia
2024**

Objetivo de esta tarea es poner en práctica lo avanzado en el laboratorio de Herramientas de Integración Continua, puntos a tomar en cuenta en las capturas:

- **Video explicando todo el proceso de creación de CI con Jenkins-Git, Grype y Sonarqube**
- **Presentación con los puntos más relevantes de todo el proceso y en una imagen la arquitectura funcional de los servicios**

Proceso de Configuración e Implementación de Jenkins Pipeline

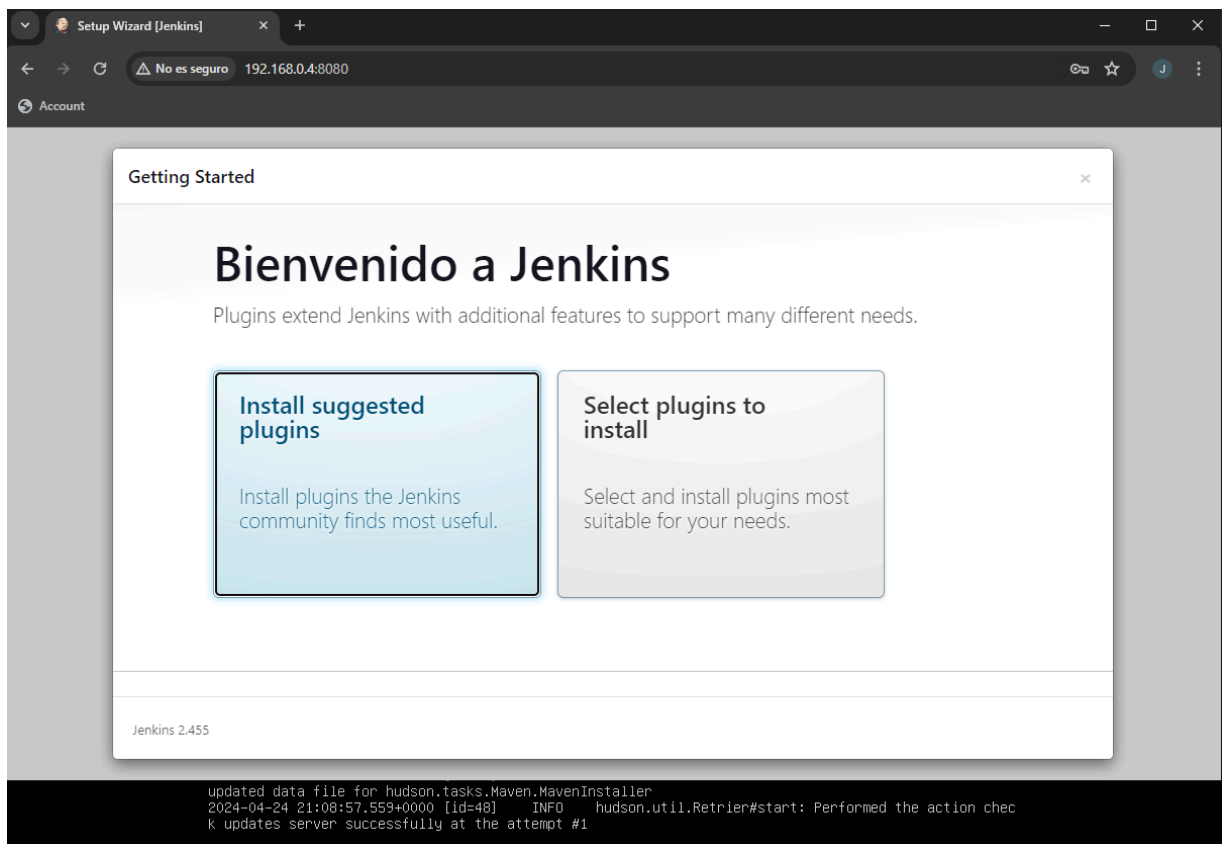
1. Se ejecutó el comando ``docker pull jenkins/jenkins``, el cual descarga la imagen Docker oficial de Jenkins desde Docker Hub. Esta imagen contiene la última versión estable de Jenkins.

2. Luego, se verificaron las imágenes disponibles en el sistema Docker utilizando el comando ``docker images``. Esto se hace para confirmar que la imagen de Jenkins se ha descargado correctamente.

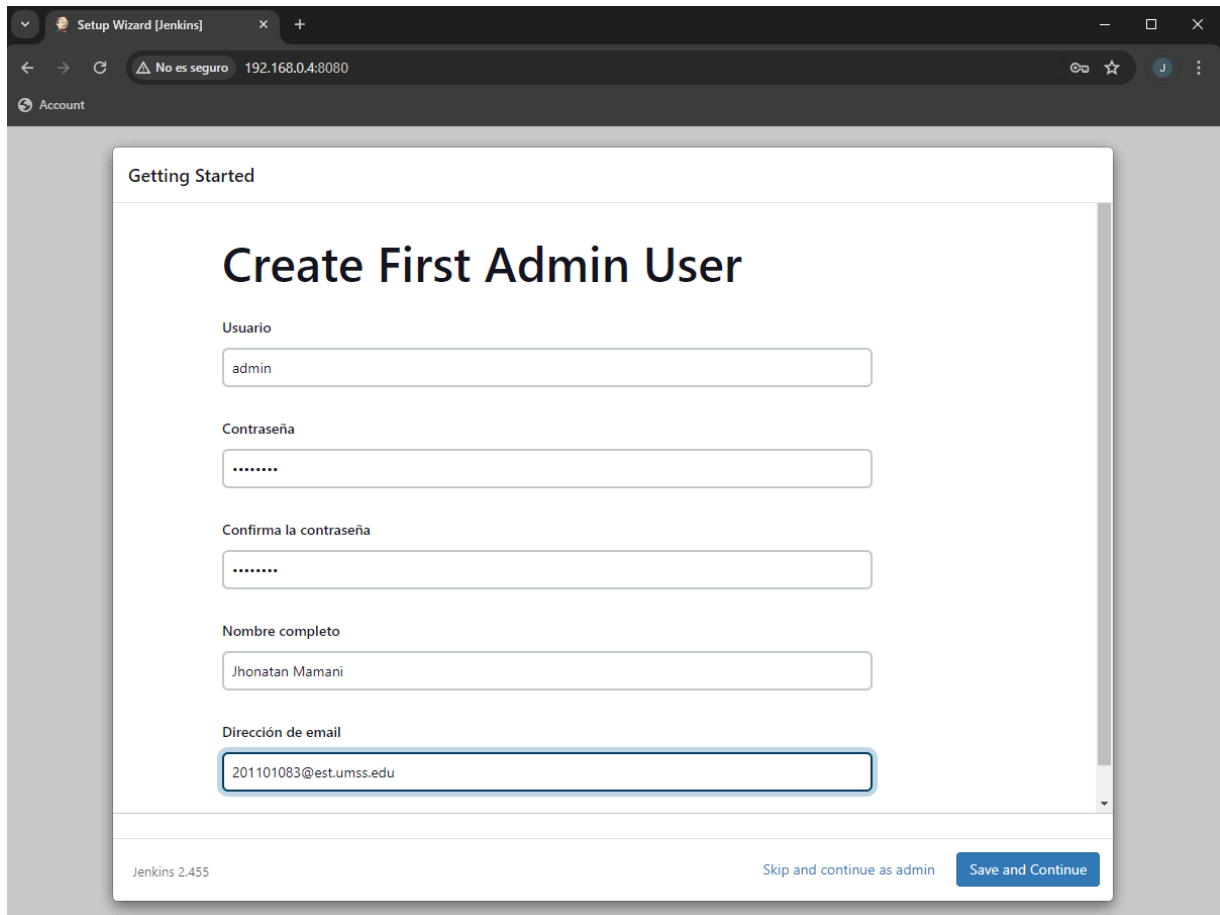
3. Después de verificar las imágenes, se ejecutó el comando ``docker run -d -p 8080:8080 -p 5000:5000 --name jenkinsj jenkins/jenkins:latest``. Este comando inicia un contenedor Docker a partir de la imagen de Jenkins descargada. El contenedor se ejecuta en segundo plano (``-d``) como se nos indicó en clases y se le asignan los puertos 8080 y 5000 para acceder a la interfaz web de Jenkins y a otras funcionalidades, respectivamente. Además, se le asigna un nombre al contenedor como **"jenkinsj"**.

4. Para verificar que el contenedor de Jenkins se esté ejecutando correctamente, se utilizó el comando ``docker ps``, que muestra una lista de los contenedores en ejecución.

5. Luego, se buscó la contraseña inicial de Jenkins en los logs del contenedor usando el comando ``docker logs -f {ID_CONTENEDOR}``. Esta contraseña es necesaria para iniciar sesión por primera vez en la interfaz web de Jenkins.
6. Después de obtener la contraseña, se ingresó la dirección IP del servidor en el navegador (en este caso, **192.168.0.4:8080**) y se proporcionó la contraseña obtenida anteriormente para acceder a la interfaz web de Jenkins.
7. En la interfaz web de Jenkins, se procedió a instalar todos los plugins necesarios que dieron la bienvenida al usuario.



8. Luego, se creó una cuenta de administrador principal ("**Create First Admin User**") proporcionando la información solicitada en la interfaz.



The screenshot shows the Jenkins Setup Wizard interface in a web browser. The browser's address bar shows the URL '192.168.0.4:8080' with a warning icon and the text 'No es seguro'. The page title is 'Setup Wizard [Jenkins]'. The main heading is 'Getting Started' followed by 'Create First Admin User'. The form contains five input fields: 'Usuario' (admin), 'Contraseña' (masked with dots), 'Confirma la contraseña' (masked with dots), 'Nombre completo' (Jhonatan Mamani), and 'Dirección de email' (201101083@est.umss.edu). At the bottom, there is a 'Skip and continue as admin' link and a 'Save and Continue' button. The Jenkins version 'Jenkins 2.455' is displayed in the bottom left corner.

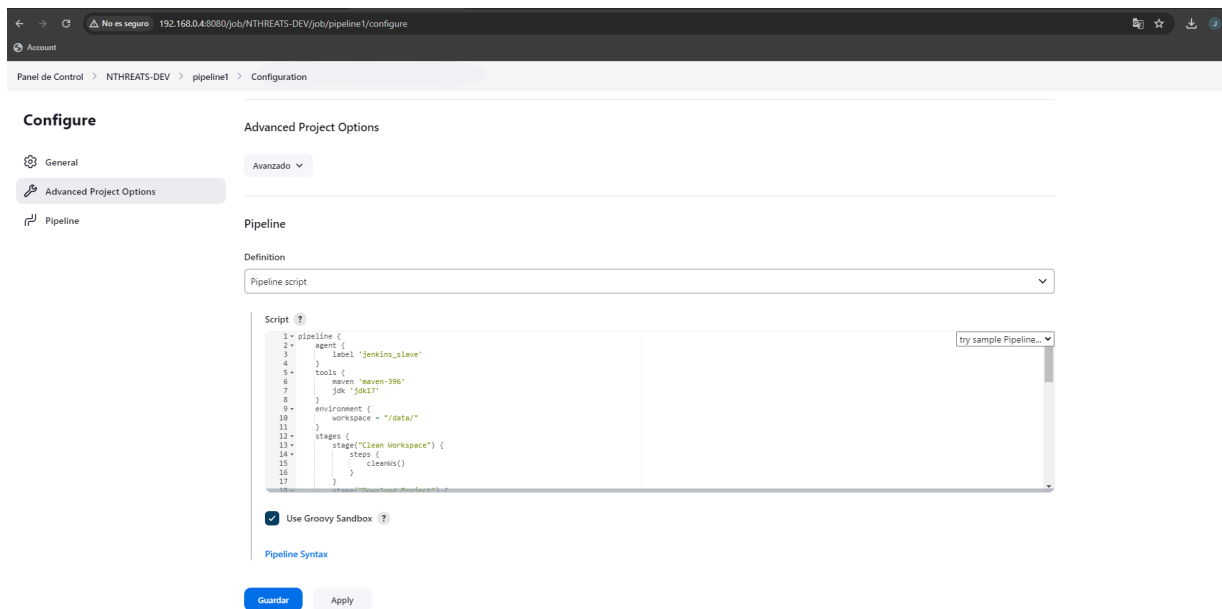
9. Después se configuró la instancia de Jenkins ingresando un Jenkins URL en la interfaz.

10. Se actualizó y configuró los plugins y herramientas necesarios, como Java **JDK 17** y **Maven 3.9.6**, desde la sección correspondiente en la interfaz de Jenkins.

11. Se creó un Folder dentro del cual se organizan los proyectos. Se utilizó el prefijo "**NTHREATS-DEV**" para identificar los proyectos relacionados con el desarrollo.

12. Dentro del Folder "**NTHREATS-DEV**", se creó un archivo Pipeline que define las stages de construcción, pruebas de vulnerabilidades y análisis de SonarQube para

un proyecto específico.



En siguiente script pipeline de puede ver con más detalle todos los stages:

```
pipeline {
  agent {
    label 'jenkins'
  }
  tools {
    maven 'maven-396'
    jdk 'jdk17'
  }
  environment {
    workspace = "/data/"
  }
  stages {
    stage("Clean Workspace") {
      steps {
        cleanWs()
      }
    }
    stage("Download Project") {
      steps {
        git credentialsId: 'git_credentials', branch: "main", url:
"https://github.com/jjohxx/nthreats-api.gitt"
        echo "Project downloaded"
      }
    }
    stage("Build Project") {
```


utilizarán en las etapas del pipeline.

2. Environment Configuration:

Se establece la variable de entorno ``workspace`` para definir la ubicación del espacio de trabajo del pipeline. En este caso, se establece en `"/data/"`.

3. Stages Definition:

En esta sección se definen las diferentes etapas del pipeline. Cada etapa representa una fase del proceso de construcción y despliegue del proyecto.

4. Clean Workspace Stage:

Esta etapa se encarga de limpiar el espacio de trabajo eliminando todos los archivos y directorios existentes. Esto asegura un entorno limpio antes de iniciar la construcción del proyecto.

5. Download Project Stage:

En esta etapa, el proyecto se descarga desde el repositorio Git especificado. Se utiliza la credencial `'git_credentials'` para la autenticación y se especifica la rama `'main'` del repositorio.

6. Build Project Stage:

Aquí se realiza la construcción del proyecto utilizando Maven. Se ejecutan los comandos Maven necesarios para limpiar, compilar y empaquetar el proyecto. El archivo JAR generado se mueve a una ubicación específica y se almacena como `"app.jar"`.

7. Test Vulnerabilities Stage:

En esta etapa, se realizan pruebas de vulnerabilidad en el archivo JAR utilizando la herramienta Gripe. El resultado se guarda en un archivo de texto llamado `"scan-report.txt"`.

8. SonarQube Analysis Stage:

Esta etapa realiza un análisis estático del código utilizando SonarQube. Se configura el archivo de propiedades del proyecto SonarQube y se ejecuta el análisis utilizando la herramienta SonarQube Scanner.

Implementación de SonarQube en un servidor

1. Configuración del entorno con Docker Compose para SonarQube:

Se ejecutó el comando ``docker-compose build`` y ``docker-compose pull`` en base al archivo de Docker Compose proporcionado por el docente. Este archivo define la configuración para levantar un servidor de SonarQube en el servidor de mi máquina virtual del **192.168.0.3:9000**. SonarQube es una plataforma de análisis estático de código abierto que proporciona informes detallados sobre la calidad del código, detecta bugs, vulnerabilidades y proporciona medidas para mejorar el código.

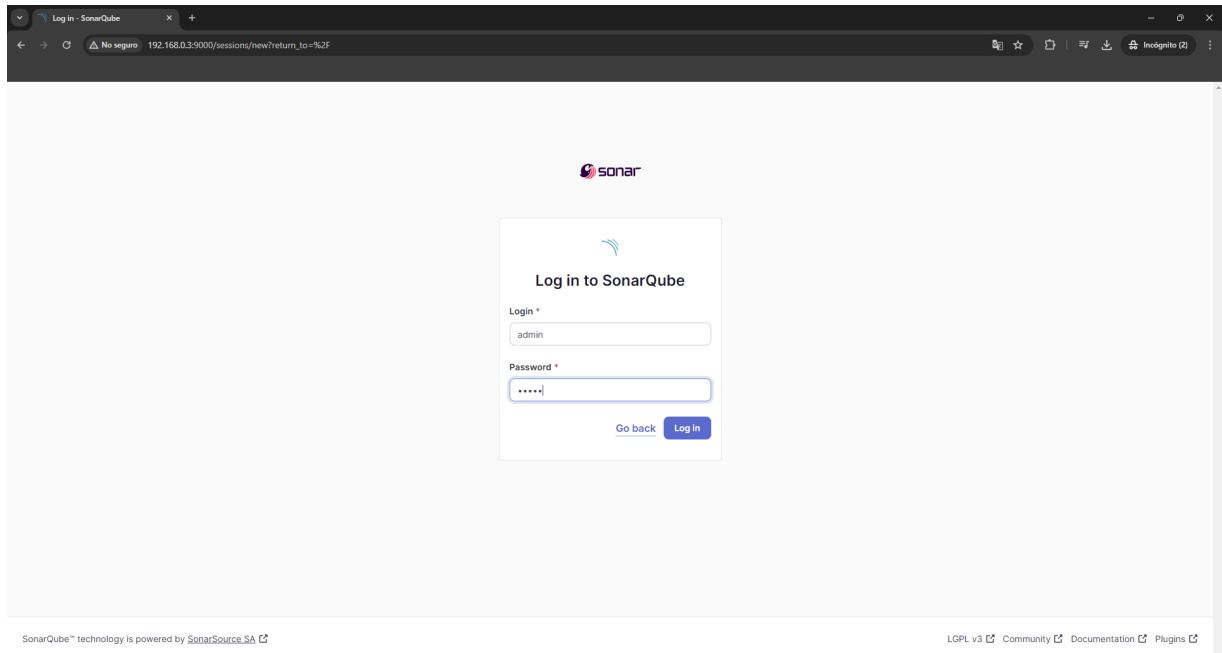
```
version: '3'
services:
  sonarqube:
    image: sonarqube
    ports:
      - "9000:9000"
    networks:
      - sonarnet
    environment:
      - SONARQUBE_JDBC_URL=jdbc:postgresql://db:5432/sonar
      - SONARQUBE_JDBC_USERNAME=sonar
      - SONARQUBE_JDBC_PASSWORD=sonar
    volumes:
      - /data/test/sonar/conf:/opt/sonarqube/conf:rw
      - /data/test/sonar/data:/opt/sonarqube/data:rw
      - /data/test/sonar/extensions:/opt/sonarqube/extensions:rw
      - /data/test/sonar/bundled-plugins:/opt/sonarqube/lib/bundled-plugins:rw

  db:
    image: postgres
    networks:
      - sonarnet
    environment:
      - POSTGRES_USER=sonar
      - POSTGRES_PASSWORD=sonar
    volumes:
      - /data/test/db/postgresql:/var/lib/postgresql:rw
      - /data/test/db/postgresql_data:/var/lib/postgresql/data:rw

networks:
  sonarnet:
    driver: bridge
```


2. Deploy de SonarQube con Docker Compose:

Se ejecutó el comando ``docker-compose up -d`` para levantar el servidor de SonarQube en modo daemon. Una imagen fue proporcionada para mostrar el éxito del proceso de levantamiento.

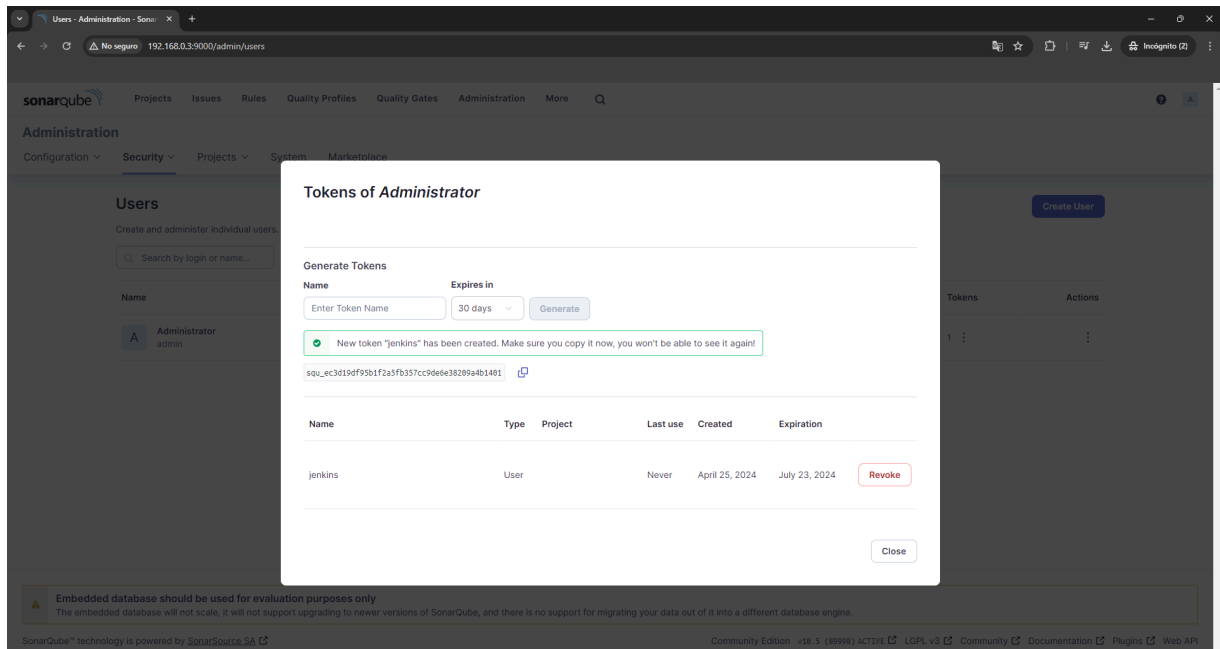


3. Acceso a SonarQube en el navegador:

Se ingresó a la dirección **192.168.0.3:9000** en el navegador. Por defecto, las credenciales de acceso son ``admin`` y la contraseña también es ``admin``. Posteriormente, estas credenciales se cambiarán por motivos de seguridad.

4. Generación de token para la integración con Jenkins:

Se accedió a la sección de usuarios y administradores en SonarQube para generar un token de acceso. Este token será utilizado para la comunicación entre el pipeline de Jenkins y el proyecto en SonarQube, permitiendo lanzar el hook de análisis de código automáticamente.

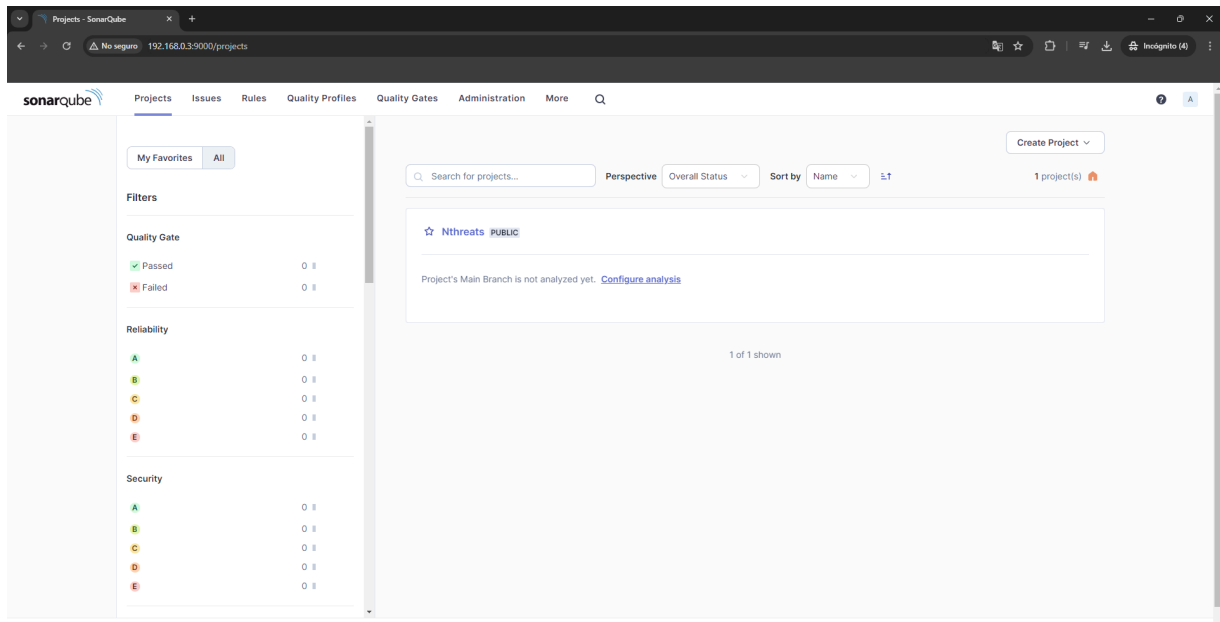


5. Instalación del plugin Sonar Scanner en SonarQube:

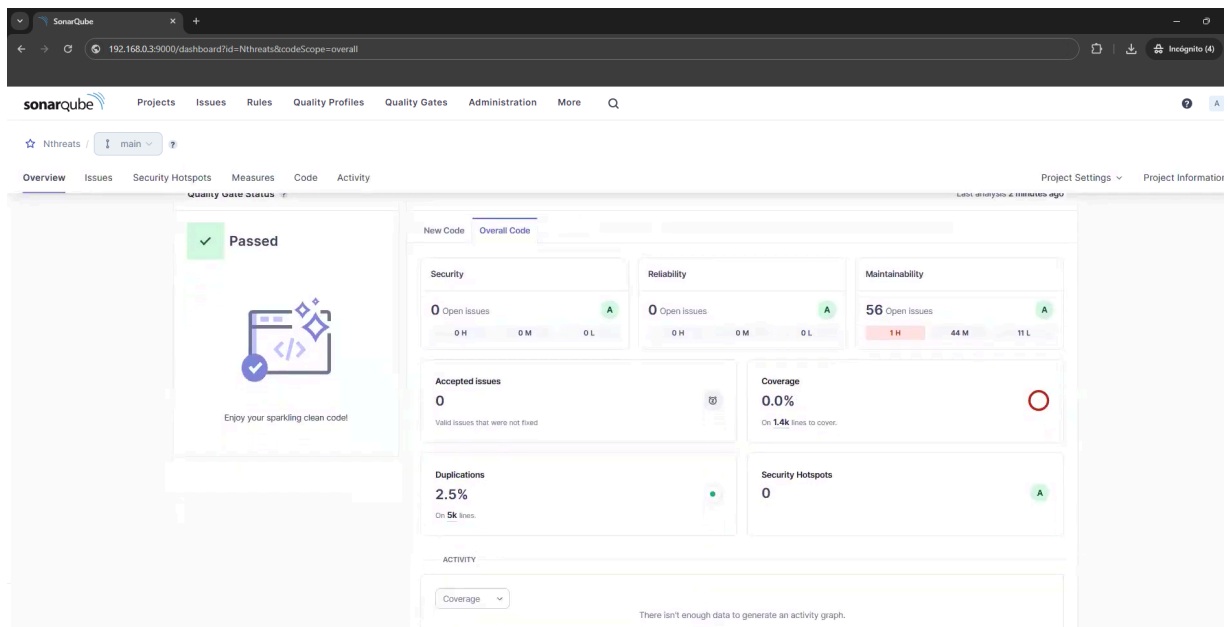
Se agregó el plugin Sonar Scanner a SonarQube para permitir la integración con herramientas de integración continua como Jenkins.

6. Generación de informes de análisis de código:

Después de que el pipeline de Jenkins se ejecutará por el stage SonarQube Analysis, se generó el proyecto en SonarQube y se mostraron estadísticas detalladas sobre la calidad del código, aplicando principios de clean code y proporcionando métricas para mejorar la calidad del código.

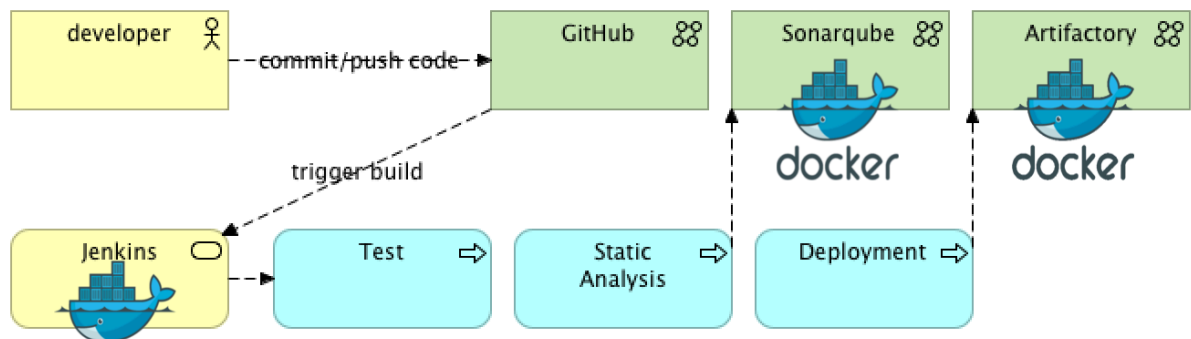


Y podemos ver el resultado del scanning.



Flujo de la Arquitectura de Implementación de Jenkins y SonarQube

En este proceso de implementación, se sigue un flujo detallado para configurar y utilizar Jenkins junto con SonarQube para automatizar el análisis de código y mejorar la calidad del software. imagen nueva subida con el tag 1.0 de la versión.



Arquitectura de los Servicios de la Aplicación N-Threats en Contenedores Docker

Este concepto describe la estructura y disposición de los servicios que conforman la aplicación N-Threats dentro de contenedores Docker. Proporciona una visión general de cómo están organizados y cómo interactúan entre sí estos servicios para ofrecer funcionalidades específicas de la aplicación.

