

# Informe Técnico: Implementación de CI/CD para Prueba Técnica DevOps Devsu.

## Introducción

En este informe quiero documentar el desarrollo y la implementación de un pipeline de CI/CD para una prueba técnica DevOps basado en Java, con integración de herramientas de calidad y automatización, utilizando Jenkins como plataforma central. Se incluye la provisión de infraestructura en AWS, pruebas de seguridad, rendimiento y despliegue automatizado en un clúster de Minikube.

Todos los archivos generados y modificados durante la prueba se encuentran en el archivo Test-Joseph-López.zip.

## Preparación del Ambiente

Para llevar a cabo esta prueba técnica, se configuró un entorno en AWS compuesto por dos instancias EC2: una para ejecutar Jenkins, que se encarga de la integración continua, y otra para desplegar Minikube, que actúa como nuestro entorno de Kubernetes local.

### 1. Configuración de la EC2 para Jenkins

- a. **Instancia:** Se usó una instancia EC2 t2.medium en AWS con Amazon Linux 2 que ya se tenía previamente aprovisionada y configurada donde preparo pruebas de conceptos y otros proyectos personales.
- b. **Jenkins:**  
Jenkins se configuró para ejecutar pipelines que incluyen la clonación de repositorios, la ejecución de pruebas, análisis de código, construcción de imágenes Docker y despliegue de servicios en Minikube.
- c. **Requisitos previos:**
  - i. **Plugins de Jenkins:** Se instalaron y configuraron varios plugins esenciales, como Git, Docker, SonarQube Scanner, OWASP ZAP, y otros para cubrir las necesidades del pipeline.
  - ii. **Acceso SSH:** Se configuró el acceso SSH para poder realizar operaciones remotas desde Jenkins, como desplegar aplicaciones en la instancia de Minikube.

The screenshot shows the Jenkins dashboard. At the top, there's a navigation bar with links for 'Historial de trabajos', 'Relacion entre proyectos', 'Comprobar firma de archivos', 'Mis vistas', 'Open Blue Ocean', and user information ('Devsu User'). Below the navigation is a search bar and a 'Desconectar' button. The main area has a 'Todo' tab selected. A table lists a single job: 'Devsu'. The table columns are 'S' (Status), 'W' (Workdir), 'Nombre' (Name), 'Último Éxito' (Last Success), 'Último Fallo' (Last Failure), 'Última Duración' (Last Duration), and a 'F' (Filter) column. The 'Devsu' row shows a green status icon, 'Devsu' as the name, 'N/D' for last success and failure, and 'N/D' for last duration. Below the table, there are dropdown menus for 'Trabajos en la cola' (No hay trabajos en la cola) and 'Estado del ejecutor de construcciones'.

Imagen 1. Jenkins funcionando desde una instancia de EC2 en AWS.

## 2. Configuración de la EC2 para Minikube

- a. **Instancia:** Se aprovisionó una instancia EC2 t2.medium en AWS también con Amazon Linux 2, dedicada exclusivamente a ejecutar Minikube, que es una herramienta que permite correr un clúster de Kubernetes localmente.
- b. **Instalación y Configuración de Minikube:**
  - i. Se instalaron las dependencias necesarias para Minikube, como Docker, Conntrack, Socat, y otros componentes de red.
  - ii. Se configuró Minikube para ejecutarse con el driver none, que permite que Minikube se ejecute directamente sobre la instancia de EC2 sin necesidad de una máquina virtual adicional.
- c. **Acceso y Configuración de Kubernetes (kubectl):**
  - i. Se instaló kubectl para interactuar con el clúster de Kubernetes administrado por Minikube.
  - ii. Se configuró kubectl para acceder correctamente al clúster, ajustando los permisos y las rutas necesarias.
  - iii. Se realizó la transferencia de los archivos de configuración (.kube/config) a la instancia de Jenkins para permitir que los despliegues se gestionen automáticamente desde Jenkins.



Imagen 2. Instancia creada para aprovisionar el minikube.

```
[root@ip-172-31-90-16 cri-dockerd]# minikube start --driver=none
😄 minikube v1.33.1 on Amazon 2023.5.20240819 (xen-amd64)
💡 Using the none driver based on existing profile
👍 Starting "minikube" primary control-plane node in "minikube" cluster
🔄 Restarting existing none bare metal machine for "minikube" ...
🕒 OS release is Amazon Linux 2023.5.20240819
🌐 Preparing Kubernetes v1.30.0 on Docker 25.0.6 ...
  kubelet.resolv-conf=/run/systemd/resolve/resolv.conf
  kubeadm.sha256: 64 B / 64 B [=====] 100.00% ? p/s 0s
  kubectl.sha256: 64 B / 64 B [=====] 100.00% ? p/s 0s
  kubelet.sha256: 64 B / 64 B [=====] 100.00% ? p/s 0s
  kubeadm: 47.92 MiB / 47.92 MiB [=====] 100.00% 42.95 MiB p/s 1.3s
  kubectl: 49.07 MiB / 49.07 MiB [=====] 100.00% 28.93 MiB p/s 1.9s
  kubelet: 95.46 MiB / 95.46 MiB [=====] 100.00% 49.27 MiB p/s 2.1s
  Generating certificates and keys ...
  Booting up control plane ...
  Configuring RBAC rules ...
🔗 Configuring bridge CNI (Container Networking Interface) ...
🌐 Configuring local host environment ...

❗ The 'none' driver is designed for experts who need to integrate with an existing VM
💡 Most users should use the newer 'docker' driver instead, which does not require root!
📘 For more information, see: https://minikube.sigs.k8s.io/docs/reference/drivers/none/

❗ kubelet and minikube configuration will be stored in /root
💡 To use kubectl or minikube commands as your own user, you may need to relocate them. For example, to overwrite your own settings, run:
  • sudo mv /root/.kube /root/.minikube $HOME
  • sudo chown -R $USER $HOME/.kube $HOME/.minikube

💡 This can also be done automatically by setting the env var CHANGE_MINIKUBE_NONE_USER=true
🌐 Verifying Kubernetes components...
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟 Enabled addons: default-storageclass, storage-provisioner
🔥 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Imagen 3. Minikube funcionando en la instancia EC2 en AWS

### 3. Despliegue y Acceso a Minikube

Una vez configuradas ambas instancias, Jenkins se utiliza para automatizar el despliegue de aplicaciones en Minikube. Esto incluye la construcción de imágenes Docker a partir del código fuente, su despliegue en el clúster de Kubernetes, y la exposición de los servicios para que sean accesibles externamente.

### 4. Diagrama de la solución

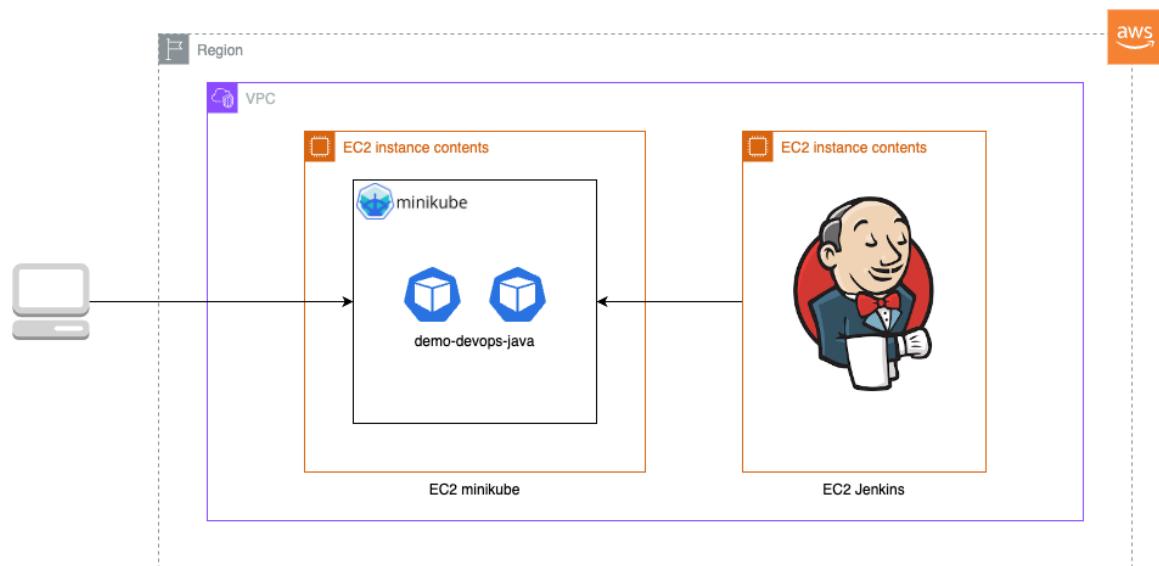


Imagen 4. Diagrama de la solución implementada para la prueba técnica.

## 5. Despliegue de la aplicación en ambiente local

### a. Creación de Dockerfile y despliegue en Docker Desktop

El Dockerfile utilizado para la construcción de la imagen Docker está configurado en dos etapas: construcción de la aplicación y ejecución del contenedor, este dockerfile puede ser revisado en el repositorio de código compartido. La imagen generada se publica en Docker Hub.

```
josephlopez@172 demo-devops-java % docker build -t demo-devops-java:1.0.0 --platform linux/amd64 .
[+] Building 0.6s (15/15) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 92B
=> [internal] load metadata for docker.io/library/openjdk:17-jdk-slim
=> [internal] load metadata for docker.io/library/maven:3.8.6-eclipse-temurin-17-alpine
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [build 1/6] FROM docker.io/library/maven:3.8.6-eclipse-temurin-17-alpine@sha256:22cd0126d970163e055210e764a7d7b14db06e
=> [stage-1 1/3] FROM docker.io/library/openjdk:17-jdk-slim@sha256:aaa3b3cb27e3e520b8f116863d0580c438ed55ecfa0bc126b41f68
=> [internal] load build context
=> => transferring context: 2.58kB
=> CACHED [stage-1 2/3] WORKDIR /app
=> CACHED [build 2/6] WORKDIR /app
=> CACHED [build 3/6] COPY pom.xml .
=> CACHED [build 4/6] RUN mvn dependency:go-offline
=> CACHED [build 5/6] COPY src ./src
=> CACHED [build 6/6] RUN mvn clean package -DskipTests
=> CACHED [stage-1 3/3] COPY --from=build /app/target/*.jar app.jar
=> exporting to image
=> => exporting layers
=> => writing image sha256:7bb325060cbf8020279f1a5566f57e9d7072a2459f79d4841d3b170c0574be62
=> => naming to docker.io/library/demo-devops-java:1.0.0

What's next:
View a summary of image vulnerabilities and recommendations → docker scout quickview
```

Imagen 5. Creación de imagen en ambiente local para validar el correcto funcionamiento de la aplicación.

```
[josephlopez@172 demo-devops-java % docker run -d -p 8000:8000 demo-devops-java:1.0.0
WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific
platform was requested
1aa8c4e9700acf2a354e163052d01f6fabdaa7bd064ae5020553a8703628acec]
```

Imagen 6. Imagen docker en Docker Desktop.

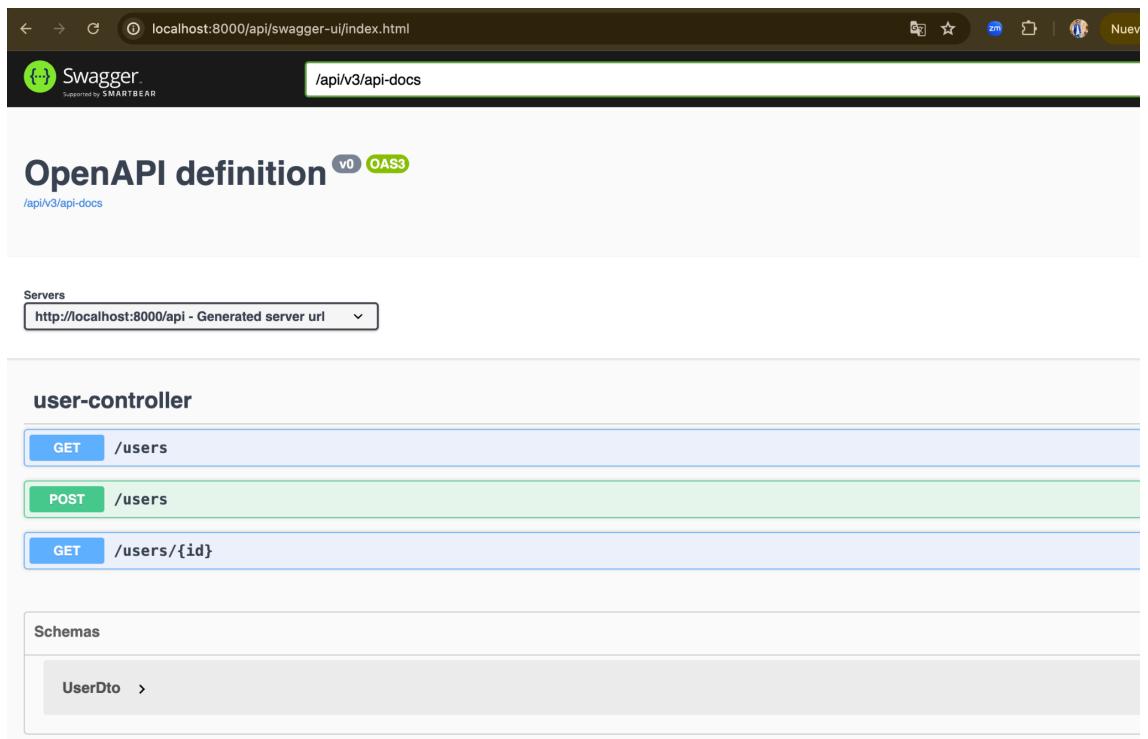


Imagen 7. Captura de pantalla de la aplicación funcionando correctamente en local.

## Repositorios y Accesos

### 1. Repositorio de GitHub (Público):

URL: <https://github.com/joshlopez07/demo-devops-java.git>

### 2. Jenkins:

URL: <http://54.89.96.48:8080/job/Devsu/job/demo-devops-java/>

Credenciales:

- a. user: devsu\_user
- b. pass: [REDACTED] En archivo .docx en el repositorio podrá obtenerse el password.

### 3. Docker Hub:

URL: <https://hub.docker.com/repository/docker/joshlopez07/demo-devops-java>

### 4. Servicio Desplegado en Minikube:

URL: <http://54.89.184.74:30007/api/swagger-ui/index.html>

### 5. SonarCloud:

URL: [https://sonarcloud.io/summary/overall?id=joshlopez07\\_demo-devops-java-devel](https://sonarcloud.io/summary/overall?id=joshlopez07_demo-devops-java-devel)

## Desarrollo del Pipeline de Jenkins

### 1. Configuración de Jenkins y Plugins

Jenkins se configuró como la herramienta central del pipeline de CI/CD, integrando los siguientes plugins clave:

- **Docker:** Para la construcción de imágenes contenerizadas.
- **Git:** Para utilizar comandos git y clonar el repositorio de código.
- **Maven:** Para usar comandos mvn para construir artefactos java y sus pruebas unitarias.
- **OWASP Dependency-Check:** Para análisis de seguridad en las dependencias del proyecto.
- **Trivy:** Para escanear la imagen Docker en busca de vulnerabilidades.
- **SonarCloud:** Para ejecutar pruebas de revisión de código y obtener cobertura de pruebas.
- **JMeter:** Para ejecutar pruebas de carga y publicar informes generados.

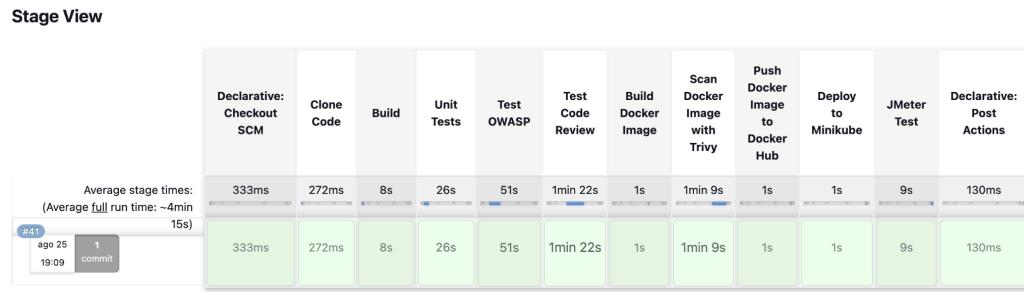


Imagen 8. Stages ejecutados en Jenkins.

## 2. Clonación del Código

La etapa inicial del pipeline clona el código fuente del repositorio de GitHub. Para esta prueba, se usó el repositorio demo-devops-java, que también está disponible en Bitbucket.

```
josephlopez@172 Documents % git clone https://bitbucket.org/devsu/demo-devops-java.git
Cloning into 'demo-devops-java'...
remote: Enumerating objects: 58, done.
remote: Counting objects: 100% (58/58), done.
remote: Compressing objects: 100% (45/45), done.
remote: Total 58 (delta 7), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (58/58), 67.64 KiB | 2.25 MiB/s, done.
Resolving deltas: 100% (7/7), done.
josephlopez@172 Documents %
```

Imagen 9. Clonación del repositorio de código de Bitbucket a local.

Commit	Message	Date
132c3d5 · 48 minutes ago	ajuste JMeter	34 Commits
last year	.mvn/wrapper	First commit
2 days ago	src	ajustes para checkhealth del pod
last year	.gitignore	First commit
yesterday	Dockerfile	ajuste remediación de vulnerabilidades
48 minutes ago	Jenkinsfile	ajuste JMeter
yesterday	JenkinsfileInfra	ajustes para JMeter y primera versión de despliegue de In...

Imagen 10. Repositorio de código en GitHub.

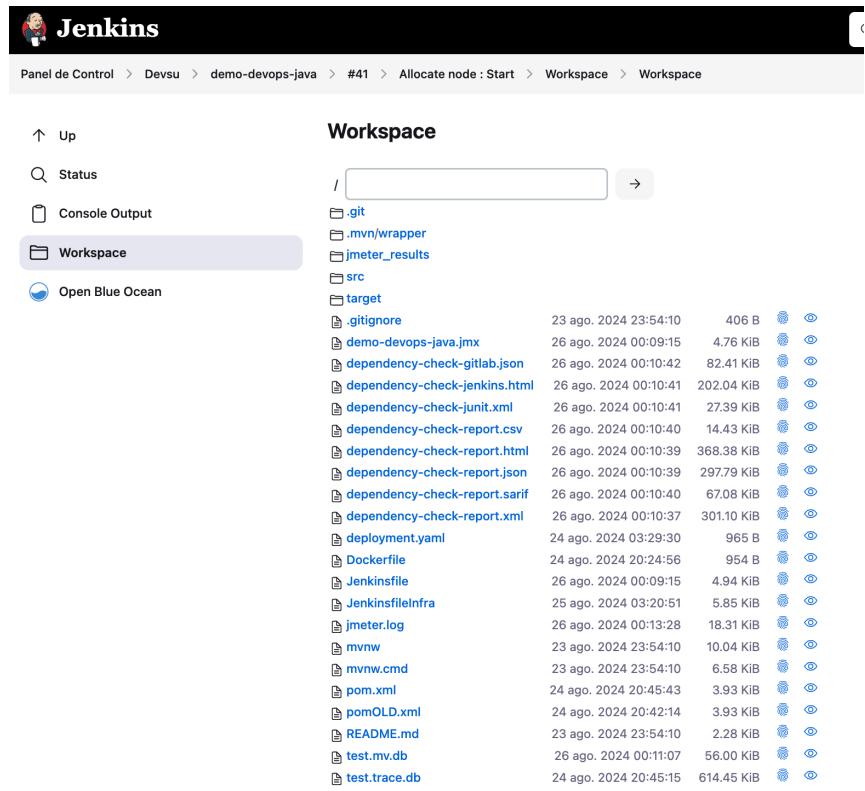


Imagen 11. Repositorio clonado visto desde Jenkins.

### 3. Pruebas Unitarias y Análisis de Calidad de Código

#### a. Ejecución de Pruebas Unitarias

Las pruebas unitarias del proyecto se ejecutaron utilizando Maven con el comando “mvn clean test” como se indica en el archivo README.md. Este comando asegura que todas las pruebas definidas en el proyecto se ejecuten, verificando que la funcionalidad del código esté conforme a lo esperado antes de proceder con otras etapas del pipeline.

```
+ mvn clean test
[ 1;34mINFO [n] Scanning for projects...
[ 1;34mINFO [n]
[ 1;34mINFO [n] [1m-----< [ 0;36mdevsu.devops:demo [0;1m ----- [n
[ 1;34mINFO [n] [1mBuilding demo-java 0.0.1 [n
[ 1;34mINFO [n] [1m----- [jar] ----- [n
[ 1;34mINFO [n]
[ 1;34mINFO [n] [1m--- [0;32mmaven-clean-plugin:3.2.0:clean [m [1m(default-clean) [m @ [ 36ndemo [0;1m --- [n
[ 1;34mINFO [n] Deleting /var/lib/jenkins/workspace/Devsu/demo-devops-java/target
[ 1;34mINFO [n]
[ 1;34mINFO [n] [1m--- [0;32mjacoco-maven-plugin:0.8.10:prepare-agent [m [1m(default) [m @ [ 36ndemo [0;1m --- [m
[ 1;34mINFO [n] argLine set to -javaagent:/var/lib/jenkins/.m2/repository/org/jacoco/org.jacoco.agent/0.8.10/org.jacoco.agent-0.8.10-runtime.jar=destfile=/var/lib/jenkins/workspace/Devsu/demo-devops-java/target/jacoco.exec
[ 1;34mINFO [n]
[ 1;34mINFO [n] [1m--- [0;32mmaven-resources-plugin:3.3.1:resources [m [1m(default-resources) [m @ [ 36ndemo [0;1m --- [m
[ 1;34mINFO [n] Copying 1 resource from src/main/resources to target/classes
[ 1;34mINFO [n] Copying 1 resource from src/main/resources to target/classes
[ 1;34mINFO [n]
[ 1;34mINFO [n] [1m--- [0;32mmaven-compiler-plugin:3.11.0:compile [m [1m(default-compile) [m @ [ 36ndemo [0;1m --- [m
[ 1;34mINFO [n] Changes detected - recompiling the module! :source
[ 1;34mINFO [n] Compiling 12 source files with javac [debug release 17] to target/classes
[ 1;34mINFO [n]
[ 1;34mINFO [n] [1m--- [0;32mmaven-resources-plugin:3.3.1:testResources [m [1m(default-testResources) [m @ [ 36ndemo [0;1m --- [m
[ 1;34mINFO [n] skip non existing resourceDirectory /var/lib/jenkins/workspace/Devsu/demo-devops-java/src/test/resources
[ 1;34mINFO [n]
[ 1;34mINFO [n] [1m--- [0;32mmaven-compiler-plugin:3.11.0:testCompile [m [1m(default-testCompile) [m @ [ 36ndemo [0;1m --- [m
[ 1;34mINFO [n] Changes detected - recompiling the module! :dependency
[ 1;34mINFO [n] Compiling 4 source files with javac [debug release 17] to target/test-classes
[ 1;34mINFO [n]
[ 1;34mINFO [n] [1m--- [0;32mmaven-surefire-plugin:3.0.0:test [m [1m(default-test) [m @ [ 36ndemo [0;1m --- [m
[ 1;34mINFO [n] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[ 1;34mINFO [n]
[ 1;34mINFO [n] -----
[ 1;34mINFO [n] T E S T S
[ 1;34mINFO [n]
[ 1;34mINFO [n] [1mINFO [n] Running devsu.devops.demo.controller.UserControllerTest [m
00:09:34.392 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils -- Could not detect default configuration classes for test class [devsu.devops.demo.controller.UserControllerTest]: UserControllerTest does not declare any static, non-private, non-final, nested classes annotated with @Configuration.
00:09:34.706 [main] INFO org.springframework.boot.test.context.SpringBootTestBootstrapper -- Found @SpringBootConfiguration devsu.devops.demo.DemoJavaApplication for test class devsu.devops.demo.controller.UserControllerTest
```

Imagen 12. Resultados de las pruebas unitarias vistas en Jenkins.

## b. Análisis de Calidad de Código y Cobertura con SonarCloud

SonarCloud se integró en el pipeline para realizar un análisis de calidad de código y medir la cobertura de las pruebas unitarias. SonarCloud ayuda a identificar posibles vulnerabilidades, problemas de estilo de código y otras métricas importantes que aseguran que el código del proyecto mantenga altos estándares de calidad.

### Modificaciones en pom.xml:

- Se añadió la configuración necesaria para integrar SonarCloud en el pipeline de Jenkins, permitiendo la recopilación de métricas de calidad y cobertura de código.

```
<properties>
    <java.version>17</java.version>
    <sonar.organization>joshlopez07</sonar.organization>
    <sonar.host.url>https://sonarcloud.io</sonar.host.url>
</properties>
```

Imagen 13. Configuración de SonarCloud en el archivo pom.xml.

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.10</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>report</id>
            <phase>test</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Imagen 14. Configuración de plugin jacoco en archivo pom.xml para validar cobertura de código probado.

The screenshot shows the SonarCloud web interface for a Java project named 'demo-java'. The 'Issues' tab is selected. On the left, there's a sidebar with 'Overview', 'Main Branch', 'Pull Requests' (0), 'Branches' (1), 'Information', and 'Administration'. The main area displays various metrics and code smells. A specific code smell for 'Intentionality' is highlighted, suggesting a replacement for a diamond operator. Other sections visible include 'Software Quality' (Security: 0, Reliability: 1, Maintainability: 19) and 'Severity' (High: 1, Medium: 5, Low: 13). The top navigation bar includes 'My Projects', 'My Issues', 'Explore', and a search bar.

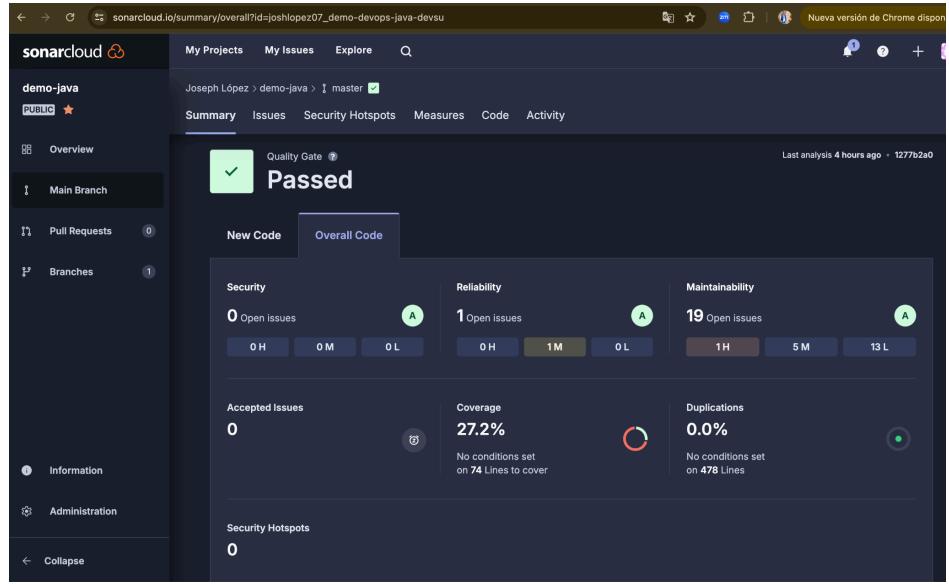


Imagen 15 y 16. Pantallazos de SonarCloud evidenciando la revisión de Código y el Coverage.

#### 4. Análisis de Seguridad con OWASP Dependency-Check y Trivy Scan Docker Image

##### a. OWASP Dependency-Check

Se configuró el análisis de seguridad de dependencias mediante OWASP Dependency-Check. Además, se realizaron modificaciones en el archivo pom.xml para habilitar este análisis.

##### Modificaciones en pom.xml:

- Se añadieron configuraciones para OWASP Dependency-Check.
- Se actualizaron librerías y dependencias en las imágenes usadas para contenerizar la aplicación con el fin de remediar vulnerabilidades detectadas.

```
<dependency>
    <groupId>org.owasp</groupId>
    <artifactId>dependency-check-maven</artifactId>
    <version>10.0.3</version>
</dependency>
```

Imagen 17. Configuración de dependencia de OWASP en archivo pom.xml.

```
<plugin>
    <groupId>org.owasp</groupId>
    <artifactId>dependency-check-maven</artifactId>
    <version>10.0.3</version>
    <configuration>
        <autoUpdate>true</autoUpdate>
        <cveUrlBase>https://services.nvd.nist.gov/rest/json/cves/2.0/</cveUrlBase>
        <cveUrlModified>https://services.nvd.nist.gov/rest/json/cves/2.0/modified</cveUrlModified>
        <nvdApiKey>c04ad272-f369-4fc3-9171-820a44bfb756</nvdApiKey>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Imagen 18. Configuración de plugin de OWASP en archivo pom.xml.

### Dependency-Check Results

SEVERITY DISTRIBUTION			
17	Search	Q	1
File Name	Vulnerability	Severity	Weakness
bootstrap.min.js	NVD CVE-2016-10735	Medium	CWE-79
bootstrap.min.js	NVD CVE-2018-14041	Medium	CWE-79
bootstrap.min.js	NVD CVE-2018-14042	Medium	CWE-79
bootstrap.min.js	NVD CVE-2018-20676	Medium	CWE-79
bootstrap.min.js	NVD CVE-2018-20677	Medium	CWE-79
bootstrap.min.js	NVD CVE-2019-8331	Medium	CWE-79
bootstrap.min.js	RETIRED Bootstrap before 4.0.0 is end-of-life and no longer maintained.	Low	
jquery-ui.min.js	NVD CVE-2021-41182	Medium	CWE-79
jquery-ui.min.js	NVD CVE-2021-41182	Medium	CWE-79

Imagen 19. Resultados de las vulnerabilidades encontradas por OWASP.

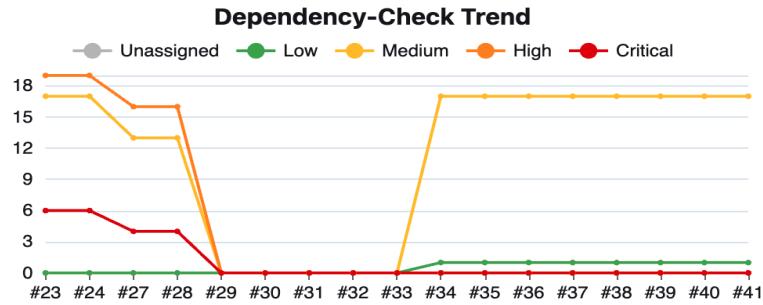


Imagen 20. Trend de los resultados de las vulnerabilidades encontradas por OWASP.

### b. Trivy Scan Docker Image

Trivy se utilizó para escanear la imagen Docker generada. Este análisis detecta vulnerabilidades en la imagen y asegura que el contenedor desplegado cumpla con los estándares de seguridad.

Se actualizaron librerías y dependencias de las imágenes usadas para construir la imagen contenerizada para remediar vulnerabilidades.

* docker run --rm -v /var/run/docker.sock:/var/run/docker.sock aquasec/trivy image --severity HIGH,CRITICAL --no-progress --exit-code 0 joshlopez07/demo-devops-javai.0.0						
2024-08-27T08:12:09Z INFO [db] Need to update DB						
2024-08-27T08:12:09Z INFO [db] Downloading DB... repository="ghcr.io/aquasecurity/trivy-db:2"						
2024-08-27T08:12:11Z INFO [vuln] Vulnerability scanning is enabled						
2024-08-27T08:12:11Z INFO [secret] Secret scanning is enabled						
2024-08-27T08:12:11Z INFO [secret] If your scanning is slow, please try '--scanners vuln' to disable secret scanning						
2024-08-27T08:12:11Z INFO [secret] Please see also <a href="https://aquasecurity.github.io/trivy/v0.54/docs/scanner/secret#recommendation">https://aquasecurity.github.io/trivy/v0.54/docs/scanner/secret#recommendation</a> for faster secret detection						
2024-08-27T08:12:10Z INFO Java DB Repository repository="ghcr.io/aquasecurity/trivy-java:db1"						
2024-08-27T08:12:10Z INFO Downloading the Java DB...						
2024-08-27T08:13:09Z INFO The Java DB is cached for 3 days. If you want to update the database more frequently, "trivy clean --java-db" command clears the DB cache.						
2024-08-27T08:13:12Z INFO Detected OS family="debian" version="11.10"						
2024-08-27T08:13:12Z INFO [debian] Detecting vulnerabilities... os_version="11" pkg_num=100						
2024-08-27T08:13:12Z INFO Number of language-specific files num=1						
2024-08-27T08:13:12Z INFO [jar] Detecting vulnerabilities...						
2024-08-27T08:13:12Z WARN Using severities from other vendors for some vulnerabilities. Read <a href="https://aquasecurity.github.io/trivy/v0.54/docs/scanner/vulnerability/severity-selection">https://aquasecurity.github.io/trivy/v0.54/docs/scanner/vulnerability/severity-selection</a> for details.						
joshlopez07/demo-devops-javai.0.0 (debian 11.10)						
Total: 16 (HIGH: 14, CRITICAL: 2)						
Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
bash	CVE-2022-3715	HIGH	affected	5.1-2+deb11u1		bash: a heap-buffer-overflow in valid_parameter_transform <a href="https://avd.aquasec.com/nvd/cve-2022-3715">https://avd.aquasec.com/nvd/cve-2022-3715</a>
e2fsprogs	CVE-2022-1384			1.46.2-2		e2fsprogs: out-of-bounds read/write via crafted filesystem <a href="https://avd.aquasec.com/nvd/cve-2022-1384">https://avd.aquasec.com/nvd/cve-2022-1384</a>
libccon-err2						
Libdb5.3	CVE-2019-8457	CRITICAL		5.3.28+dfsg1-0.8		sqlite: heap out-of-bound read in function rtreemode() <a href="https://avd.aquasec.com/nvd/cve-2019-8457">https://avd.aquasec.com/nvd/cve-2019-8457</a>
libext2fs2	CVE-2022-1384	HIGH		1.46.2-2		e2fsprogs: out-of-bounds read/write via crafted filesystem <a href="https://avd.aquasec.com/nvd/cve-2022-1384">https://avd.aquasec.com/nvd/cve-2022-1384</a>
libgcrypt20	CVE-2021-33560			1.8.7-6		libgcrypt: mishandles ElGamal encryption because it lacks exponent blinding to address a...

Imagen 21. Resultados de las vulnerabilidades encontradas por Trivy.

```
# Stage 2: Run the application
FROM openjdk:17-jdk-slim
WORKDIR /app

# Copia el archivo JAR compilado desde la etapa de build
COPY --from=build /app/target/*.jar app.jar

# Actualizar paquetes en una imagen basada en Debian/Ubuntu
RUN apt-get update && apt-get upgrade -y
```

Imagen 22. Configuración de apt-get update para remediar vulnerabilidades de Trivi.

## 5. Pruebas de Carga con JMeter

#### a. Configuración de JMeter

JMeter se configuró para realizar pruebas de carga sobre los servicios REST expuestos por la aplicación. Se ajustó el archivo .jmx para asegurarse de que las peticiones GET se realizarán correctamente, incluyendo la configuración del header Content-Type.

```
<elementProp name="ThreadGroup.main_controller" elementType="LoopController">
    <intProp name="LoopController.loops">20</intProp>
</elementProp>
<stringProp name="ThreadGroup.num_threads">5</stringProp>
<stringProp name="ThreadGroup.ramp_time">5</stringProp>
<longProp name="ThreadGroup.start_time">1692896000000</longProp>
<longProp name="ThreadGroup.end_time">1692896000000</longProp>
<boolProp name="ThreadGroup.scheduler">false</boolProp>
<stringProp name="ThreadGroup.duration"></stringProp>
<stringProp name="ThreadGroup.delay"></stringProp>
</ThreadGroup>
<hashTree>
    <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="GET Users" enabled="true">
        <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
            <collectionProp name="Arguments.arguments">
                <elementProp name="" elementType="HTTPArgument">
                    <boolProp name="HTTPArgument.always_encode">false</boolProp>
                    <stringProp name="Argument.name"></stringProp>
                    <stringProp name="Argument.value"></stringProp>
                    <stringProp name="Argument.metadata"></stringProp>
                    <boolProp name="HTTPArgument.use_equals">false</boolProp>
                </elementProp>
            </collectionProp>
        </elementProp>
        <stringProp name="HTTPSampler.domain">54.89.184.74</stringProp>
        <stringProp name="HTTPSampler.port">30007</stringProp>
        <stringProp name="HTTPSampler.protocol">http</stringProp>
        <stringProp name="HTTPSampler.path">/api/users</stringProp>
        <stringProp name="HTTPSampler.method">GET</stringProp>
        <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
        <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
        <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
        <boolProp name="HTTPSampler.D0_MULTIPART_POST">false</boolProp>
        <boolProp name="HTTPSampler.monitor">false</boolProp>
        <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
        <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
            <collectionProp name="Arguments.arguments">
                <elementProp name="" elementType="HTTPArgument">
                    <boolProp name="HTTPArgument.always_encode">false</boolProp>
                    <stringProp name="Argument.name">Content-Type</stringProp>
                </elementProp>
            </collectionProp>
        </elementProp>
    </HTTPSamplerProxy>
</hashTree>
```

Imagen 23. Parte del archivo demo-devops-java.imx para ejecución de pruebas de carga con JMeter.

Imagen 24. Resultados de las pruebas de carga con JMeter.

## b. Ejecución y Publicación de Resultados

Las pruebas se ejecutaron en modo no gráfico, generando reportes en HTML. Sin embargo, el Dashboard de JMeter en Jenkins no mostró gráficos, lo que requiere ajustes adicionales.

## 6. Despliegue de la Aplicación

El despliegue se realizó en el clúster de Minikube configurado en la instancia EC2, este aprovisionamiento de infraestructura se hizo exclusivamente para esta prueba.

Utilizando kubectl en el pipeline para aplicar los manifiestos de Kubernetes, donde se usó (deployment, service y HorizontalPodAutoscaler) y se levantan las 2 réplicas como se solicita en la prueba.

En este caso, lo único que no logré configurar en el ambiente del mini kube fue lograr que funcionaran correctamente el liveness y readiness. En ambiente local con Docker Desktop si funcionó pero en mini kube el pod se reiniciaba, creo que puede ser por tema de recursos del mini kube (Monte una EC2 lo más pequeña posible para no incurrir en grandes gastos)

```
josephlopez@172 Downloads % ssh root@54.89.184.74
root@54.89.184.74's password:
[...]
Last login: Sat Aug 24 03:18:36 2024 from 181.61.208.220
[root@ip-172-31-90-16 ~]# kubectl get pods -n default
NAME                               READY   STATUS    RESTARTS   AGE
demo-devops-java-5b77976dc4-n2hk9  1/1     Running   0          2m47s
demo-devops-java-5b77976dc4-vvhkl  1/1     Running   0          2m45s
```

Imagen 25. Evidencias los pods levantados en el minikube.

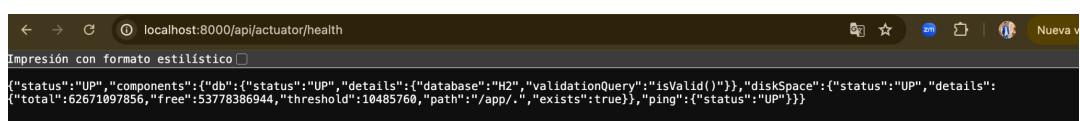


Imagen 26. Evidencia de Liveness y Readiness en local.

Imagen 27. Evidencia de la aplicación funcionando desde el minikube.

## 7. Conclusión

El proyecto consistió en la implementación de un pipeline de CI/CD utilizando herramientas y servicios que permiten automatizar y gestionar de manera eficiente todo el ciclo de vida de desarrollo de software. A lo largo del proceso, se desplegaron y configuraron múltiples herramientas en un entorno distribuido, aprovechando la flexibilidad y escalabilidad que ofrece AWS. Se cumplen los requerimientos de la prueba técnica y se aporta conocimiento adicional para la robustez final del servicio, quiero mencionar ítems importantes realizados durante la prueba.

### a. Configuración del Entorno:

Se utilizaron dos instancias EC2 en AWS: una para Jenkins y otra para Minikube. La instancia de Jenkins se configuró para gestionar los pipelines de CI/CD, ejecutando tareas que van desde la compilación del código hasta la ejecución de pruebas y despliegue de aplicaciones. La instancia de Minikube, por otro lado, permitió simular un entorno de Kubernetes, donde se desplegaron y probaron los servicios desarrollados. Las dos instancias fueron creadas y configuradas desde cero (Reitero que la instancia de Jenkins ya la tenía con anterioridad para uso personal).

### b. Pipeline de Jenkins:

Se diseñó un pipeline automatizado que abarca todas las etapas clave del desarrollo: clonación del código desde el repositorio Git, ejecución de pruebas unitarias con Maven, revisión de código estático con SonarCloud, análisis de seguridad con OWASP ZAP y Trivy, pruebas de carga con JMeter, construcción de imágenes Docker, y despliegue en Kubernetes mediante Minikube. Este enfoque garantiza que cualquier cambio en el código sea automáticamente validado, probado y desplegado de forma segura y eficiente. Siempre manteniendo como base los pilares de agilidad, calidad, seguridad y monitoreo continuo.

## Recomendaciones y Pasos Futuros

Para mejorar y continuar con el desarrollo de este pipeline, quedaron los siguientes ajustes:

- **Optimizar el Dashboard de JMeter:** Solucionar los problemas actuales de visualización para obtener reportes detallados y gráficos de rendimiento.
- **Ampliar el Scope de Pruebas de Seguridad:** Integrar más herramientas de análisis estático y dinámico de seguridad para cubrir más vectores de ataque.
- **Explorar Chaos Engineering:** Implementar prácticas de Chaos Engineering para probar la resiliencia de los servicios bajo condiciones adversas.
- **Documentación y Monitorización Continua:** Asegurar que todos los procesos están bien documentados y que se implementen soluciones de monitorización continua para detectar y resolver problemas en tiempo real.

Con este proyecto demuestro mis capacidades de integrar múltiples herramientas y tecnologías para crear un pipeline de CI/CD robusto y efectivo, proporcionando una base sólida para el despliegue y la gestión continua de aplicaciones en un entorno de producción.

**Puntos Extras:**

**Creación de Instancia con Terraform, Aprovisionamiento de Minikube y Despliegue de Imagen.**

**1. Creación de la Instancia con Terraform, aprovisionamiento de MiniKube y despliegue.**

a. **Objetivo:** Provisión de una instancia de EC2 en AWS para alojar un clúster de Minikube y desplegar la imagen **demo-devops-java:1.0.0** creada anteriormente.

b. **Pipeline:** Archivo en repositorio compartido “**JenkinsfileInfra**”, usa los scripts “**configure\_kubectl.sh**” y “**setup\_minikube.sh**”.

c. **Proceso:**

i. Se utilizó Terraform OSS para automatizar la creación de la instancia, la configuración incluía la especificación de la región, tipo de instancia, AMI, par de claves SSH, subnet y grupo de seguridad (Use claves SSH, VPC y SG que usan las EC2 anteriores).

ii. El provisioner “remote-exec” se usó para ejecutar comandos directamente en la instancia, instalando y configurando Minikube y sus dependencias.

iii. Se instalaron las dependencias necesarias como Docker, conntrack, socat, ebtables, ethtool, y cri-dockerd, entre otras.

iv. Se configuró y se inició Minikube con el controlador none, que permite correr Kubernetes directamente sobre el host, sin virtualización.

v. Se aseguraron los permisos y las configuraciones correctas para que los archivos de Minikube estuvieran accesibles y configurados adecuadamente.

vi. Se configuró Jenkins para conectarse a la instancia EC2 y utilizar kubectl para gestionar el clúster de Minikube.

vii. Las configuraciones y certificados necesarios para kubectl fueron transferidos desde la instancia EC2 al servidor de Jenkins.

viii. Jenkins aplicó la configuración de despliegue de Kubernetes utilizando el archivo deployment.yaml, asegurando que la imagen de la aplicación se desplegara en el clúster de Minikube.

d. **Desafíos:**

i. Manejo de rutas de archivos y permisos, asegurando que las configuraciones de Minikube y Kubernetes fueran accesibles para la herramienta kubectl.

ii. Necesidad de adaptar y corregir rutas y permisos en los archivos de configuración para garantizar la compatibilidad con Jenkins.

e. **Resultado:**

i. **Servicio Desplegado en Minikube:**

URL: [http://<instance\\_ip>:30007/api/swagger-ui/index.html](http://<instance_ip>:30007/api/swagger-ui/index.html) la variable instance\_ip se puede encontrar en la ejecución del pipeline al terminar el aprovisionamiento de la EC2 (Importante: Está configurado el pipeline para que la instancia EC2 esté habilitada por 5 minutos, posterior a esto el pipeline de Jenkins lanzará el destroy de la misma).

ii. Terraform creó exitosamente una instancia EC2 con las configuraciones especificadas, lista para ser utilizada como nodo de control de Minikube.

iii. Minikube fue instalado y configurado correctamente.

- iv. La imagen de la aplicación fue desplegada exitosamente en el clúster de Minikube, y se pudo verificar el estado de los nodos y la aplicación utilizando kubectl.

```
[0m [1aws_instance.minikubeTerraform: Creation complete aft
[0m [1m [32m
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
[0m [0m [1m [32m
Outputs:

[0minstance_ip = "44.193.2.239"
instance_private_ip = "172.31.13.187"
[Pipeline] }
[Pipeline] // script
```

Imagen 28. IP pública e IP privada de la EC2 aprovisionada.



Imagen 29. Pipeline en Jenkins ejecutado.

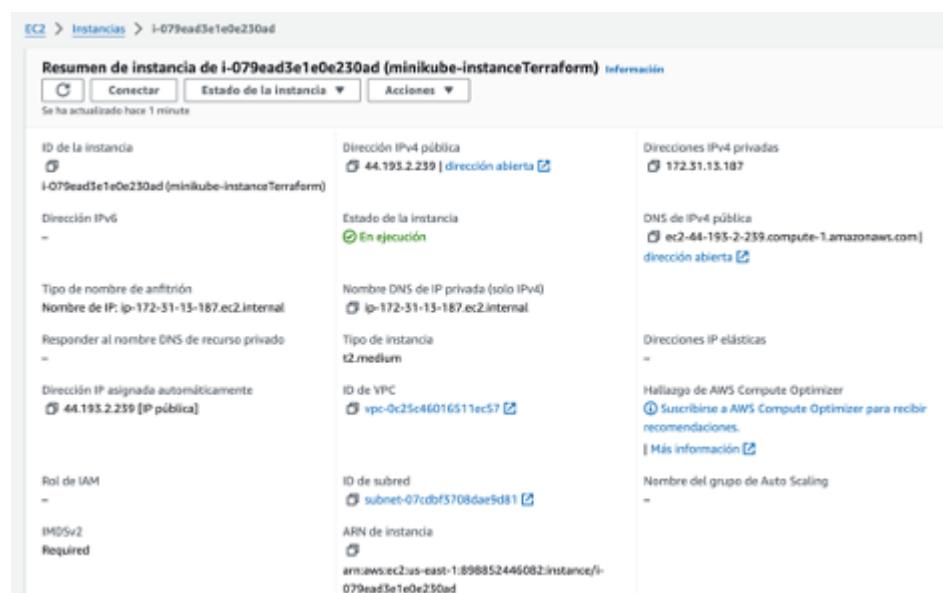


Imagen 30. Instancia de EC2 vista desde el portal de AWS.

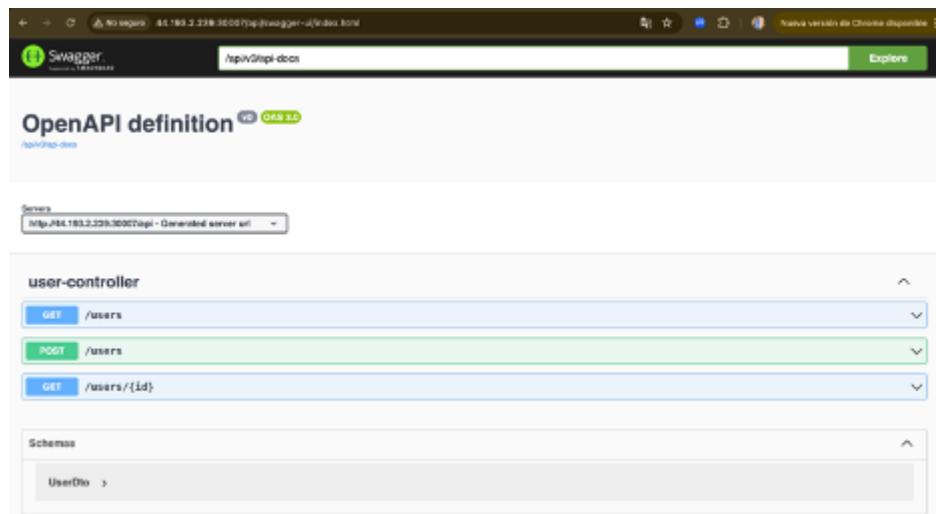


Imagen 31. Servicios desplegados y funcionando en la EC2 aprovisionada con Terraform.

## 2. Consideraciones Finales

- a. **Optimización:** Se centralizaron los comandos de aprovisionamiento en scripts bash para mejorar la legibilidad y mantenimiento del pipeline.
- b. **Flexibilidad:** Se implementaron soluciones adaptativas para manejar diferencias en configuraciones y rutas entre los servidores de Jenkins y EC2.
- c. **Automatización Completa:** La infraestructura y el despliegue de la aplicación están completamente automatizados, permitiendo una fácil replicación y escalabilidad en el futuro.

## Conclusión

La integración de Terraform, Minikube y Jenkins permitió automatizar de manera eficiente el despliegue de una infraestructura de Kubernetes sobre una instancia EC2 en AWS. A través de la automatización, se logró un proceso reproducible y escalable para futuras implementaciones, asegurando la consistencia y la eficiencia en la gestión del ciclo de vida de la infraestructura y el despliegue de aplicaciones.