CI/CD, Herramientas y Jenkins: Una Guía Detallada

1. ¿Qué es CI/CD? Definición y Principios Fundamentales

El término CI/CD engloba las prácticas de Integración Continua (CI) y Entrega Continua o Despliegue Continuo (CD), representando una filosofía y un conjunto de prácticas diseñadas para agilizar y acelerar el ciclo de vida del desarrollo de software [1]. La Integración Continua es una práctica moderna de desarrollo de software en la que los cambios de código incrementales se realizan de forma frecuente y fiable [2]. Este proceso implica que los desarrolladores integran sus cambios de código en un repositorio compartido varias veces al día, o incluso con mayor frecuencia [1]. La automatización de la construcción y las pruebas, desencadenada por la CI, asegura que los cambios de código que se fusionan en el repositorio sean fiables [3]. La CI puede considerarse una solución al problema de tener demasiadas ramas de una aplicación en desarrollo a la vez que podrían entrar en conflicto entre sí [1].

La Entrega Continua extiende la CI al automatizar la entrega de cambios de código a entornos como pruebas y desarrollo [3]. Proporciona una forma automatizada y consistente para que el código se entregue a estos entornos [3]. La entrega continua generalmente significa que los cambios de un desarrollador en una aplicación se prueban automáticamente para detectar errores y se cargan en un repositorio (como GitHub o un registro de contenedores), donde luego pueden ser implementados en un entorno de producción en vivo por el equipo de operaciones [1]. El Despliegue Continuo es el siguiente paso de la entrega continua. Cada cambio que pasa las pruebas automatizadas se coloca automáticamente en producción, lo que resulta en muchas implementaciones de producción [3]. El objetivo de la mayoría de las empresas debería ser el despliegue continuo, a menos que estén limitadas por requisitos regulatorios u otros [3]. En resumen, la CI es un conjunto de prácticas realizadas mientras los desarrolladores escriben código, y la CD es un conjunto de prácticas realizadas después de que se completa el código [3].

Los principios fundamentales de la Integración Continua incluyen el mantenimiento de un repositorio de código único, la automatización del proceso de construcción, la creación de una construcción auto-testable, el compromiso diario de todos los desarrolladores con la línea base, la construcción de cada commit a la línea base, mantener la construcción rápida, probar en un clon del entorno de producción, facilitar la obtención de las últimas entregas y la visibilidad de los resultados de la última construcción [5]. Otros principios importantes son el commit temprano y frecuente, el uso de control de versiones y el trabajo en lotes pequeños [6].

Los cinco principios centrales de la Entrega Continua son construir calidad en cada etapa, trabajar en lotes pequeños, automatizar todo lo que se pueda, buscar la mejora continua y que todos sean responsables [7]. Otros principios incluyen un proceso repetible y fiable, el control de versiones de todo, que "hecho" significa liberado, mantener la pipeline siempre desplegable, facilitar hacer lo correcto y tratar los despliegues como eventos no importantes [11].

La implementación de CI/CD es crucial porque acelera el tiempo de comercialización de nuevas funciones y correcciones de errores [3]. También mejora la calidad y la estabilidad del software mediante pruebas automatizadas y la detección temprana de problemas [8]. Además, aumenta la eficiencia y la productividad del equipo al automatizar tareas repetitivas, reduce los riesgos asociados con los despliegues y permite una retroalimentación más rápida y frecuente [3]. Finalmente, fomenta una mejor colaboración entre los equipos de desarrollo y operaciones, lo cual es un objetivo clave de DevOps [1].

2. Herramientas Esenciales en el Flujo de Trabajo CI/CD

Un flujo de trabajo de CI/CD eficaz se basa en una variedad de herramientas especializadas para automatizar diferentes etapas del proceso de entrega de software. Estas herramientas se pueden categorizar según su función principal:

2.1. Control de Versiones

La función principal de las herramientas de control de versiones es gestionar los cambios en el código fuente a lo largo del tiempo, facilitando la colaboración entre los desarrolladores y permitiendo el seguimiento y la reversión de modificaciones [6]. Git es el sistema de control de versiones distribuido más popular y sirve como la base para muchas plataformas de colaboración de código [16]. Algunas herramientas comunes en esta categoría incluyen Git mismo, junto con plataformas de alojamiento de código como GitHub, GitLab y Bitbucket, que proporcionan funcionalidades adicionales como gestión de ramas, solicitudes de fusión y revisión de código [14]. Subversion es otra herramienta de control de versiones, aunque menos utilizada en comparación con Git [15].

Tabla: Herramientas de Control de Versiones

Herramienta	Descripción
Git	Sistema de control de versiones distribuido, ampliamente utilizado.
GitHub	Plataforma de alojamiento de código basada en la nube que utiliza Git, con funciones de colaboración.
GitLab	Plataforma DevOps completa con control de versiones Git integrado, CI/CD y gestión de proyectos.
Bitbucket	Servicio de alojamiento de código Git para equipos, con integración a herramientas de Atlassian.
Subversion	Sistema de control de versiones centralizado, más tradicional.

2.2. Construcción (Build)

Las herramientas de construcción se encargan de compilar el código fuente en un formato ejecutable y de gestionar las dependencias necesarias para la aplicación [14]. La elección de la herramienta a menudo depende del lenguaje de programación utilizado en el proyecto. Para proyectos Java, Maven y Gradle son herramientas populares para la gestión de la construcción y dependencias [20]. Make es una herramienta más general que se utiliza para automatizar la compilación de código y otros procesos. Para proyectos JavaScript, npm (Node Package Manager) y yarn se utilizan para gestionar dependencias y ejecutar scripts de construcción [20].

2.3. Pruebas (Testing)

Las pruebas automatizadas son un componente crítico de los flujos de trabajo de CI/CD, ya que aseguran la calidad del código en cada etapa del pipeline [3]. Se utilizan diferentes tipos de pruebas, incluyendo pruebas unitarias, pruebas de integración, pruebas funcionales, pruebas de rendimiento y pruebas de seguridad [15]. Herramientas comunes en esta categoría incluyen JUnit y TestNG para pruebas unitarias en Java, Selenium para pruebas de interfaz de usuario, así como frameworks de pruebas específicos para otros lenguajes como pytest para Python y Jest para JavaScript [20]. Además, algunas herramientas de CI/CD, como GitLab CI, incluyen funcionalidades para realizar pruebas de seguridad como SAST (Static Application Security Testing) y

DAST (Dynamic Application Security Testing) [20].

2.4. Despliegue (Deployment)

Las herramientas de despliegue automatizan la entrega del software construido y probado a los diferentes entornos, desde el desarrollo hasta la producción [3]. Ansible, Chef y Puppet son herramientas de gestión de la configuración que permiten automatizar la configuración de servidores y la implementación de aplicaciones [1]. Para la orquestación de contenedores, Kubernetes y Docker Compose son ampliamente utilizados, especialmente en entornos basados en microservicios [1]. Las plataformas en la nube también ofrecen sus propias herramientas de despliegue, como AWS CodeDeploy y Azure DevOps Pipelines [20].

2.5. Monitorización (Monitoring)

La monitorización continua es esencial para mantener la visibilidad sobre el rendimiento y la salud de las aplicaciones desplegadas [19]. Herramientas como Prometheus y Grafana son populares para la recopilación y visualización de métricas. Datadog y New Relic ofrecen soluciones de observabilidad más completas con integraciones para CI/CD. Splunk es otra herramienta utilizada para el análisis de datos y la monitorización en tiempo real [20].

Tabla: Herramientas CI/CD por Función

Función	Herramientas Comunes
Control de Versiones	Git, GitHub, GitLab, Bitbucket, Subversion
Construcción	Maven, Gradle, Make, npm, yarn
Pruebas	JUnit, TestNG, Selenium, pytest, Jest, herramientas de seguridad (SAST, DAST)
Despliegue	Ansible, Chef, Puppet, Kubernetes, Docker Compose, AWS CodeDeploy, Azure DevOps Pipelines

Monitorización	Prometheus, Grafana, Datadog, New Relic, Splunk

3. Jenkins: El Motor de Automatización CI/CD

Jenkins es un servidor de automatización de código abierto, escrito principalmente en Java, que se ha convertido en una de las herramientas más populares para implementar flujos de trabajo de Integración Continua y Entrega Continua [4]. Su propósito principal es automatizar las diversas etapas del ciclo de vida del desarrollo de software, abarcando la construcción, las pruebas y el despliegue de aplicaciones [4]. Jenkins actúa como un orquestador central, integrando diferentes herramientas y automatizando tareas repetitivas para agilizar el proceso de entrega de software.

Originalmente conocido como Hudson, Jenkins fue creado por Kohsuke Kawaguchi mientras trabajaba en Sun Microsystems y se lanzó como software de código abierto en 2007 [17]. Tras una disputa con Oracle, que adquirió Sun, el proyecto fue renombrado a Jenkins en 2011 [22]. Desde entonces, ha experimentado un crecimiento significativo, convirtiéndose en una de las herramientas de automatización CI/CD más utilizadas en la industria del desarrollo de software [17]. Su longevidad y amplia adopción son testimonio de su robustez y valor en el ecosistema de desarrollo de software.

4. Análisis Detallado de Jenkins: Ventajas y Desventajas

El uso de Jenkins en un entorno de CI/CD ofrece numerosas ventajas, pero también presenta ciertas desventajas que deben considerarse.

4.1. Ventajas Clave

Una de las ventajas más significativas de Jenkins es su **extensa colección de plugins** [4]. Con más de 1800 plugins disponibles, Jenkins puede integrarse con una amplia variedad de herramientas y tecnologías, lo que permite a los equipos adaptar la herramienta a sus flujos de trabajo y necesidades específicas. Esta vasta colección de plugins facilita la integración con sistemas de control de versiones como Git, herramientas de construcción como Maven y Gradle, frameworks de pruebas como JUnit y Selenium, plataformas de despliegue como Docker y Kubernetes, y muchas otras herramientas del ecosistema DevOps [4].

Jenkins también destaca por su **flexibilidad y capacidad de personalización** [18]. Es altamente configurable y adaptable a las necesidades específicas de cada proyecto, lo que permite a los equipos diseñar pipelines de CI/CD que se ajusten a sus

requisitos únicos. Esta flexibilidad se extiende a la definición de los propios pipelines, que pueden ser tan simples o complejos como sea necesario, y pueden ser definidos a través de la interfaz gráfica de usuario o mediante código [28].

El soporte para **'Pipeline as Code'** es otra ventaja crucial [4]. Jenkins permite definir los pipelines de CI/CD como código en un archivo llamado Jenkinsfile, que se almacena en el repositorio de control de versiones junto con el código fuente de la aplicación. Esto facilita la gestión, la colaboración y la reproducibilidad de los pipelines, ya que pueden ser versionados, revisados y compartidos entre los miembros del equipo [4].

La **arquitectura distribuida (Master-Slave/Agent)** de Jenkins es fundamental para mejorar el rendimiento y la escalabilidad [4]. Esta arquitectura permite distribuir la carga de trabajo de las construcciones y pruebas a múltiples agentes (también conocidos como slaves o nodos), lo que acelera el proceso de CI/CD y permite manejar proyectos grandes y complejos con múltiples entornos de prueba y despliegue [41].

Jenkins cuenta con una **gran comunidad y amplios recursos de aprendizaje** [4]. Su popularidad ha llevado a la creación de abundante documentación, foros de soporte y una activa comunidad de usuarios y desarrolladores dispuestos a ayudar y compartir sus experiencias. Esto facilita la resolución de problemas, el aprendizaje de nuevas funcionalidades y la adopción de mejores prácticas en el uso de Jenkins [4].

Finalmente, Jenkins se destaca por su **fácil instalación y configuración** [4]. Su interfaz web intuitiva facilita la instalación y configuración inicial, incluso para usuarios que no tienen una amplia experiencia en CI/CD. Jenkins es una aplicación Java independiente y multiplataforma, lo que permite su instalación en diversos sistemas operativos como Windows, Linux y macOS [4].

4.2. Desventajas Significativas

A pesar de sus numerosas ventajas, Jenkins también presenta algunas desventajas significativas. La **complejidad en la configuración y gestión** puede ser un obstáculo, especialmente para nuevos usuarios [28]. La gran cantidad de opciones y la necesidad de configurar plugins pueden hacer que la configuración inicial y la gestión continua de Jenkins sean complejas y requieran un conocimiento especializado.

La **interfaz de usuario** de Jenkins es considerada anticuada por algunos usuarios en comparación con las tendencias de diseño de interfaces modernas [43]. Si bien la interfaz proporciona la funcionalidad necesaria, su diseño puede resultar menos intuitivo y atractivo para los usuarios acostumbrados a herramientas más modernas.

La dependencia de **plugins** también puede ser una fuente de problemas [28]. La gestión de un gran número de plugins puede llevar a problemas de compatibilidad entre versiones, la necesidad de un mantenimiento constante para asegurar que los plugins estén actualizados y sean compatibles con la versión de Jenkins, e incluso la posibilidad de que algunos plugins dejen de ser mantenidos, lo que podría generar riesgos de seguridad o pérdida de funcionalidad.

En arquitecturas monolíticas, Jenkins puede presentar **desafíos de escalabilidad** [28]. Aunque la arquitectura master-slave ayuda a distribuir la carga de trabajo, el servidor maestro en sí mismo puede convertirse en un cuello de botella en implementaciones muy grandes con un gran volumen de trabajos.

Jenkins tiene una **curva de aprendizaje** pronunciada para los nuevos usuarios [43]. La complejidad de la herramienta y la gran cantidad de opciones disponibles pueden resultar abrumadoras para quienes no están familiarizados con los conceptos de CI/CD o con la arquitectura y el ecosistema de plugins de Jenkins.

Finalmente, Jenkins requiere un **mantenimiento continuo** para asegurar su estabilidad y seguridad [43]. Esto incluye la actualización regular del servidor de Jenkins y de todos los plugins instalados, así como la monitorización del sistema para identificar y resolver posibles problemas.

4.3. Tabla Comparativa de Ventajas y Desventajas

Ventajas	Desventajas
Extensa colección de plugins	Complejidad en la configuración y gestión
Flexibilidad y capacidad de personalización	Interfaz de usuario considerada anticuada por algunos
Soporte para 'Pipeline as Code'	Desafíos en la gestión de plugins y compatibilidad
Arquitectura distribuida (Master-Slave/Agent)	Potenciales problemas de escalabilidad en arquitecturas monolíticas

Gran comunidad y recursos de aprendizaje	Curva de aprendizaje para nuevos usuarios
Fácil instalación y configuración	Necesidad de mantenimiento continuo

5. Casos de Uso Prácticos de Jenkins en la Industria del Software

Jenkins se utiliza en una amplia variedad de escenarios y en diversas industrias para automatizar el ciclo de vida del desarrollo de software.

5.1. Desarrollo Web y Móvil

En el desarrollo web y móvil, Jenkins es una herramienta fundamental para automatizar la construcción, las pruebas y el despliegue de aplicaciones [49]. Por ejemplo, en proyectos web basados en Java, Jenkins puede configurarse para utilizar Maven o Gradle para compilar el código, ejecutar pruebas unitarias y de integración, empaquetar la aplicación en un archivo WAR o JAR, y luego desplegarlo en un servidor de aplicaciones como Tomcat o en un entorno de contenedores Docker [49]. De manera similar, para aplicaciones JavaScript, Jenkins puede utilizar npm o yarn para gestionar dependencias, ejecutar pruebas con frameworks como Jest o Mocha, y construir la aplicación para su despliegue [49]. En el desarrollo móvil, Jenkins se puede integrar con los SDKs de Android e iOS para automatizar la construcción de archivos APK e IPA, ejecutar pruebas en emuladores o dispositivos reales, y distribuir las aplicaciones a tiendas o plataformas de prueba [22].

5.2. Diferentes Industrias

La versatilidad de Jenkins ha llevado a su adopción en diversas industrias. En el sector **Fintech**, empresas como Tymit (una compañía de procesamiento de tarjetas de crédito) han utilizado Jenkins para construir una plataforma DevOps segura y conforme, logrando una entrega más rápida de servicios móviles, microservicios y operacionales, y reduciendo los ciclos de prueba y lanzamiento en un 50% [45]. En la industria de **viajes**, Avoris Travel ha aprovechado Jenkins para reducir los tiempos de construcción en más del 50% y aumentar la velocidad de entrega de su plataforma tecnológica, que soporta a cientos de agencias y millones de consumidores a nivel internacional [70]. Históricamente, en el sector de las **telecomunicaciones**, Nokia utilizaba un proceso llamado 'Nightly Builds' como un precursor de la Integración Continua, lo que ilustra la larga trayectoria de prácticas similares a CI/CD en la industria [45].

5.3. Automatización de Tareas y Flujos de Trabajo

Más allá del desarrollo de software, Jenkins se utiliza para automatizar una amplia

gama de tareas y flujos de trabajo. Esto incluye la automatización de la instalación y actualización de componentes en múltiples entornos, lo que es especialmente útil para desarrolladores que gestionan diversas configuraciones [68]. Jenkins también se utiliza para automatizar la ejecución de pruebas automatizadas, como pruebas unitarias, de integración y de aceptación del usuario, y para generar informes detallados sobre los resultados de estas pruebas [29]. Además, Jenkins puede configurarse para automatizar tareas repetitivas como la realización de copias de seguridad de bases de datos, el encendido o apagado de máquinas virtuales, la recopilación de estadísticas sobre el rendimiento de servicios y otras tareas operativas [30].

6. Arquitectura Interna de Jenkins y su Funcionamiento

La arquitectura de Jenkins se basa en un modelo distribuido de **maestro-agente** (o controlador-agente), donde un servidor maestro central (Controller) gestiona y coordina el trabajo realizado por uno o más agentes (Slaves o Nodes) [4].

6.1. Componentes Principales

El **Servidor Maestro (Controller)** es el núcleo de la instancia de Jenkins. Sus responsabilidades incluyen la programación de trabajos de construcción basados en diversos triggers, la distribución de estos trabajos a los agentes disponibles para su ejecución, la monitorización del estado de los agentes, el registro y la presentación de los resultados de las construcciones, la gestión de la seguridad y los usuarios, y la administración de los plugins instalados [4]. Los componentes clave del maestro incluyen los trabajos (Jobs), que son las tareas automatizadas definidas por los usuarios; los plugins, que extienden la funcionalidad de Jenkins; las credenciales para la autenticación con otros sistemas; los nodos o clouds, que representan los agentes disponibles; y la configuración global del sistema [47].

Los **Agentes (Slaves/Nodes)** son los nodos de trabajo que ejecutan los scripts y comandos enviados por el servidor maestro [4]. Los agentes pueden ejecutarse en una variedad de sistemas operativos, lo que permite a Jenkins realizar construcciones y pruebas en diferentes entornos [4]. La arquitectura maestro-agente permite la ejecución paralela de múltiples construcciones, lo que mejora significativamente la eficiencia del proceso de CI/CD [62].

Los **Plugins** son módulos desarrollados por la comunidad y de forma oficial que extienden la funcionalidad base de Jenkins [4]. Permiten la integración de Jenkins con una amplia gama de herramientas y servicios utilizados en el ciclo de vida del desarrollo de software, como sistemas de control de versiones, herramientas de

construcción, frameworks de pruebas y plataformas de despliegue [4].

6.2. Flujo de Trabajo General

El flujo de trabajo general en Jenkins comienza cuando un desarrollador commite código a un repositorio de control de versiones como Git [4]. El servidor maestro de Jenkins detecta estos cambios, ya sea mediante la verificación periódica del repositorio (polling) o a través de webhooks configurados en el repositorio [4]. Una vez que se detectan cambios, Jenkins programa un trabajo de construcción y lo asigna a un agente disponible [4]. El agente descarga el código fuente del repositorio y ejecuta los pasos definidos en el trabajo, que pueden incluir la compilación del código, la ejecución de pruebas unitarias y de integración, la generación de artefactos y el despliegue de la aplicación [4]. Durante la ejecución del trabajo, el agente reporta continuamente el estado y los resultados al servidor maestro [4]. Finalmente, el servidor maestro genera informes sobre la construcción y las pruebas, envía notificaciones a los miembros del equipo si es necesario, y puede desencadenar etapas posteriores del pipeline, como el despliegue a diferentes entornos [4]. Este flujo de trabajo automatizado proporciona una retroalimentación rápida a los desarrolladores sobre los cambios en el código y facilita la entrega continua de software.

7. Dominando la Sintaxis de Jenkins Pipeline

La definición de un flujo de trabajo de CI/CD en Jenkins se realiza a través de un **Jenkinsfile**, que es un archivo de texto que contiene la definición del pipeline como código [4]. Este archivo se almacena en el repositorio de control de versiones junto con el código fuente de la aplicación, lo que permite un control de versiones y una revisión del pipeline al igual que cualquier otro código [4]. Jenkins soporta dos tipos principales de sintaxis para definir pipelines: Declarative y Scripted [4].

7.1. Sintaxis Básica de Declarative Pipeline

La sintaxis Declarative proporciona una estructura más simplificada y opinada para definir pipelines [52]. La estructura fundamental de un pipeline Declarative es la siguiente:

Groovy

```
pipeline {
    agent any
```

```
    stages {
        stage('Nombre de la Etapa') {
            steps {
                // Pasos a ejecutar
            }
        }
    }
}
```

Las directivas comunes en la sintaxis Declarative incluyen agent, que especifica dónde se ejecutará el pipeline o una etapa específica (puede ser any para cualquier agente disponible, none para no asignar un agente global, label para un agente con una etiqueta específica, o configuraciones más avanzadas para Docker, Dockerfile o Kubernetes) [35]. La directiva stages contiene una o más etapas del pipeline [35], y cada stage define una etapa específica con un nombre y contiene un bloque steps que incluye una secuencia de uno o más pasos a ejecutar [35]. Otras directivas comunes son environment para definir variables de entorno, options para configurar opciones del pipeline como timeouts o reintentos, parameters para definir parámetros que se pueden pasar al pipeline, triggers para especificar cuándo se debe ejecutar el pipeline (por ejemplo, mediante un cron o un evento de GitHub), tools para especificar las herramientas necesarias (como JDK o Maven), input para solicitar la intervención humana, when para definir condiciones para la ejecución de una etapa, y post para definir acciones que se ejecutarán al final del pipeline o de una etapa (como enviar notificaciones por correo electrónico en caso de fallo o éxito) [35]. La sintaxis Declarative proporciona una estructura clara y concisa, facilitando la lectura y el mantenimiento de los pipelines [55].

7.2. Sintaxis Básica de Scripted Pipeline

La sintaxis Scripted se basa en el lenguaje de programación Groovy y ofrece una mayor flexibilidad pero requiere un conocimiento más profundo de la programación [4]. En un pipeline Scripted, el flujo de trabajo se define dentro de un bloque node, que especifica dónde se ejecutará el pipeline [49]. Las etapas (stage) son opcionales en la sintaxis Scripted, pero se recomiendan para proporcionar una mejor visualización de las tareas en la interfaz de usuario de Jenkins [49]. Un ejemplo básico de un pipeline Scripted podría ser:

```
Groovy
```

```
node {
    stage('Build') {
        echo 'Building the application...'
        // Comandos de construcción
    }
    stage('Test') {
        echo 'Running tests...'
        // Comandos de prueba
    }
    stage('Deploy') {
        echo 'Deploying the application...'
        // Comandos de despliegue
    }
}
```

La sintaxis Scripted ofrece más control sobre el flujo del pipeline y permite la utilización de la potencia completa de Groovy, pero puede resultar más compleja para quienes no están familiarizados con este lenguaje [28].

8. Guía Detallada de Instalación de Jenkins

La instalación de Jenkins se puede realizar de diversas maneras, siendo las más comunes la instalación en un equipo físico o máquina virtual tradicional y la instalación utilizando Docker.

8.1. Instalación en Equipo Físico o Máquina Virtual

Para instalar Jenkins en un equipo físico o una máquina virtual, se requiere tener instalado el Java Development Kit (JDK) versión 11 o superior [74]. Los pasos específicos varían según el sistema operativo. A continuación, se presenta un ejemplo de los pasos para la instalación en un sistema Linux basado en Debian o Ubuntu:

1. **Añadir la clave del repositorio de Jenkins:** Este paso asegura la autenticidad de los paquetes que se descargarán del repositorio oficial de Jenkins. El comando es: wget -O /usr/share/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key [74].
2. **Añadir la entrada del repositorio de Jenkins a la lista de fuentes de paquetes:** Esto le indica al sistema dónde encontrar los paquetes de Jenkins. El comando es: 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 [74].

3. **Actualizar los índices de paquetes:** Este comando sincroniza la lista de paquetes disponibles desde los repositorios configurados. Se ejecuta con: sudo apt update [74].
4. **Instalar Jenkins:** Finalmente, se instala Jenkins con el comando: sudo apt install jenkins [74].
5. **Iniciar y habilitar el servicio de Jenkins:** Para que Jenkins se inicie automáticamente al arrancar el sistema y para iniciarlo inmediatamente, se utilizan los comandos: sudo systemctl enable jenkins y sudo systemctl start jenkins [76].
6. **Abrir el puerto 8080 en el firewall:** Jenkins se ejecuta por defecto en el puerto 8080, por lo que es necesario abrir este puerto en el firewall del sistema para poder acceder a la interfaz web de Jenkins [74].

Una vez instalado, se puede acceder a Jenkins a través de un navegador web ingresando la dirección http://localhost:8080 (si se está accediendo localmente) o http://<dirección_ip_del_servidor>:8080. En la primera visita, Jenkins solicitará la contraseña inicial del administrador, que se puede obtener ejecutando el comando sudo cat /var/lib/jenkins/secrets/initialAdminPassword en el servidor [74]. A continuación, se guiará al usuario a través de la instalación de plugins recomendados y la creación del primer usuario administrador [74].

8.2. Instalación Utilizando Docker

La instalación de Jenkins utilizando Docker proporciona un entorno aislado y consistente para ejecutar la herramienta [37]. Para ello, es necesario tener Docker instalado en el sistema [37]. Los pasos para la instalación con Docker son los siguientes:

1. **Descargar la imagen de Jenkins desde Docker Hub:** Se utiliza el comando docker pull jenkins/jenkins:lts-jdk11 o docker pull jenkins/jenkins:latest-jdk17 para descargar la última versión LTS (Long Term Support) o la versión más reciente con la versión de JDK especificada [37].
2. **Crear una red Docker (opcional):** Se puede crear una red para facilitar la comunicación entre contenedores si se planea ejecutar Jenkins junto con otros servicios. El comando es docker network create jenkins-network [37].
3. **Ejecutar el contenedor de Jenkins:** Para iniciar un contenedor de Jenkins, se utiliza un comando como: docker run -d --name jenkins-container -p 8080:8080 -p 50000:50000 --network jenkins-network -v jenkins_home:/var/jenkins_home jenkins/jenkins:lts-jdk11 [37]. Este comando ejecuta el contenedor en segundo plano (-d), le asigna un nombre (--name), mapea los puertos 8080 y 50000 del host al

contenedor (-p), lo conecta a la red Docker creada (--network), y crea un volumen persistente llamado jenkins_home para almacenar los datos de Jenkins (-v).

4. **Acceder a Jenkins:** Una vez que el contenedor está en ejecución, se puede acceder a la interfaz web de Jenkins a través de la dirección http://localhost:8080 en el navegador [37].
5. **Obtener la contraseña inicial del administrador:** La contraseña inicial se encuentra en los logs del contenedor. Se puede obtener con el comando docker exec jenkins-container cat /var/jenkins_home/secrets/initialAdminPassword [37].
6. **Completar la configuración inicial:** Al igual que en la instalación tradicional, se deberá ingresar la contraseña inicial, instalar los plugins recomendados y crear el usuario administrador a través de la interfaz web [37].

8.3. Comparativa entre Ambos Métodos

La instalación tradicional en un equipo físico o máquina virtual ofrece un control total sobre el sistema operativo y acceso directo a los recursos del hardware [28]. Sin embargo, puede llevar más tiempo de configuración y requiere una gestión manual de las dependencias, con el riesgo de contaminar el entorno del sistema [28].

Por otro lado, la instalación con Docker es más rápida y sencilla, proporciona un entorno aislado y consistente, ofrece mayor portabilidad y facilita la gestión de las dependencias a través de imágenes y contenedores [84]. No obstante, pueden surgir problemas de permisos al interactuar con recursos del host, y la interacción con el sistema operativo subyacente puede ser más compleja. Además, en algunos casos, la instalación con Docker puede resultar en un mayor uso de espacio en disco [37].

Tabla: Comparativa de Instalación

Característica	Equipo Físico/VM	Docker
Tiempo de Instalación	Mayor	Menor
Aislamiento	Menor	Mayor
Portabilidad	Menor	Mayor

Control del SO	Total	Limitado
Gestión de Recursos	Directa	A través de Docker
Consumo de Recursos	Potencialmente mayor (SO completo)	Menor (contenedor)
Complejidad	Gestión manual de dependencias	Gestión a través de imágenes y contenedores

9. Ejemplo Práctico: Flujo de Trabajo CI/CD con Jenkins, GitHub y Despliegue

A continuación, se describe un ejemplo de un flujo de trabajo completo que abarca desde la creación de código en GitHub hasta su despliegue en hosts utilizando Jenkins.

1. **Creación y Gestión de Código en GitHub:** El primer paso es tener el código fuente de la aplicación gestionado en un repositorio de GitHub.
2. **Configuración de un Webhook en GitHub:** Para que Jenkins detecte automáticamente los cambios en el código, se configura un webhook en el repositorio de GitHub. Esto se hace en la sección 'Webhooks' de la configuración del repositorio. Se añade un nuevo webhook con la URL del servidor de Jenkins (generalmente con el sufijo /github-webhook/) y se selecciona el evento 'Push' para que se active cada vez que se suben cambios al repositorio [93]. Opcionalmente, se puede configurar un 'Secret' para asegurar la comunicación entre GitHub y Jenkins [93].
3. **Definición de un Jenkinsfile (Declarative Pipeline):** En el repositorio de GitHub, se crea un archivo llamado Jenkinsfile con la definición del pipeline de CI/CD. Un ejemplo básico podría ser:

```Groovy
pipeline {
    agent any
    triggers {
        github() // Habilita el trigger de GitHub
    }
    stages {
        stage('Checkout') {
            steps {
                git credentialsId: 'github-ssh-credentials', // Reemplazar con el ID de las credenciales
```

```
de GitHub en Jenkins
                url: 'git@github.com:<usuario>/<repositorio>.git' // Reemplazar con la URL del
repositorio
            }
        }
        stage('Build') {
            steps {
                sh 'echo "Building the application..."'
                // Aquí irían los comandos para compilar la aplicación (e.g., mvn clean install)
            }
        }
        stage('Test') {
            steps {
                sh 'echo "Running unit tests..."'
                // Aquí irían los comandos para ejecutar las pruebas unitarias (e.g., mvn test)
            }
        }
        stage('Deploy') {
            steps {
                sshagent(credentials: ['ssh-deploy-credentials']) { // Reemplazar con el ID de las
credenciales SSH del servidor de despliegue
                    sh 'echo "Deploying to host(s)..."'
                    sh 'ssh -o StrictHostKeyChecking=no <usuario>@<host1> "mkdir -p /opt/app && cp
target/app.jar /opt/app/"' // Reemplazar con los detalles del host y los comandos de despliegue
                    sh 'ssh -o StrictHostKeyChecking=no <usuario>@<host2> "mkdir -p /opt/app && cp
target/app.jar /opt/app/"' // Ejemplo para múltiples hosts
                }
            }
        }
    }
}
```

4. **Procedimientos Paso a Paso:**
    1. Se crea un repositorio en GitHub y se sube el código de la aplicación junto con el Jenkinsfile.
    2. En Jenkins, se configuran las credenciales necesarias: unas para acceder al repositorio de GitHub (ya sea mediante clave SSH o nombre de usuario y contraseña) [95], y otras credenciales SSH para acceder a los servidores donde se desplegará la aplicación [98].

3. Se crea un nuevo pipeline en Jenkins. En la configuración del pipeline, se selecciona la opción 'Pipeline script from SCM' y se proporcionan los detalles del repositorio de GitHub, incluyendo la URL y las credenciales configuradas.
4. Se configura el webhook en la sección de configuración del repositorio de GitHub, apuntando a la URL del servidor de Jenkins.
5. Un desarrollador realiza un commit y un push de un cambio al repositorio de GitHub.
6. El webhook de GitHub notifica a Jenkins, lo que dispara automáticamente una nueva ejecución del pipeline.
7. En la interfaz de Jenkins, se puede monitorizar el progreso de cada etapa del pipeline ('Checkout', 'Build', 'Test', 'Deploy').
8. Si todas las etapas se completan con éxito, la aplicación se habrá desplegado en los hosts remotos especificados mediante los comandos SSH definidos en la etapa 'Deploy' del Jenkinsfile [58].

Conclusiones

CI/CD se ha consolidado como una práctica esencial en el desarrollo moderno de software, permitiendo a los equipos entregar cambios de código de manera más rápida, frecuente y con mayor fiabilidad. Jenkins, como servidor de automatización de código abierto, desempeña un papel fundamental en la implementación de flujos de trabajo de CI/CD, gracias a su flexibilidad, su vasta colección de plugins y su soporte para 'Pipeline as Code'. Si bien presenta ciertas complejidades y desafíos en su gestión, sus ventajas en términos de automatización, integración y escalabilidad lo convierten en una herramienta poderosa y ampliamente utilizada en la industria del software. La comprensión de sus principios, su arquitectura y su sintaxis de pipeline es clave para aprovechar al máximo sus capacidades y construir flujos de trabajo de entrega de software eficientes y robustos.

Obras citadas

1. What is CI/CD? - Red Hat, fecha de acceso: marzo 22, 2025, https://www.redhat.com/en/topics/devops/what-is-ci-cd
2. www.blackduck.com, fecha de acceso: marzo 22, 2025, https://www.blackduck.com/glossary/what-is-cicd.html#:~:text=Definition,are%20made%20frequently%20and%20reliably.
3. What Is CI/CD and How Does It Work? - Black Duck, fecha de acceso: marzo 22, 2025, https://www.blackduck.com/glossary/what-is-cicd.html
4. Make Better Quality Software using Jenkins for your CI/CD Pipeline | CloudIQ Tech, fecha de acceso: marzo 22, 2025, https://www.cloudiqtech.com/jenkins-pipeline-an-introduction/
5. Continuous Integration Best Practices by CloudBees, fecha de acceso: marzo 22,

2025, https://www.cloudbees.com/continuous-delivery/continuous-integration-best-practices
6. Explaining Continuous Integration Principles - SAP Learning, fecha de acceso: marzo 22, 2025, https://learning.sap.com/learning-journeys/discovering-devops-with-sap-btp/explaining-continuous-integration-principles_fa206662-72d6-488c-8d23-879307a93502
7. Continuous Delivery: Origins, 5 Principles, And 7 Key Capabilities - Octopus Deploy, fecha de acceso: marzo 22, 2025, https://octopus.com/devops/continuous-delivery/
8. CI/CD : Everything about these principles that helps tech teams - DataScientest.com, fecha de acceso: marzo 22, 2025, https://datascientest.com/en/all-about-ci-cd
9. Principles - Continuous Delivery, fecha de acceso: marzo 22, 2025, https://continuousdelivery.com/principles/
10. Continuous Delivery (CD) | Definition and Overview - ProductPlan, fecha de acceso: marzo 22, 2025, https://www.productplan.com/glossary/continuous-delivery/
11. The 9 Principles of Continuous Delivery - DBmaestro, fecha de acceso: marzo 22, 2025, https://www.dbmaestro.com/blog/database-ci-cd/principles-continuous-delivery
12. Mastering Continuous Delivery Principles for Seamless Software Releases - Harness, fecha de acceso: marzo 22, 2025, https://www.harness.io/harness-devops-academy/continuous-delivery-principles
13. 8 Key Continuous Delivery Principles - Atlassian, fecha de acceso: marzo 22, 2025, https://www.atlassian.com/continuous-delivery/principles
14. What is CI/CD? - GitHub, fecha de acceso: marzo 22, 2025, https://github.com/resources/articles/devops/ci-cd
15. What is CI/CD? - GitLab, fecha de acceso: marzo 22, 2025, https://about.gitlab.com/topics/ci-cd/
16. How to Use Version Control in CI/CD Pipelines - PixelFreeStudio Blog, fecha de acceso: marzo 22, 2025, https://blog.pixelfreestudio.com/how-to-use-version-control-in-ci-cd-pipelines/
17. Continuous Integration Tools: Top 7 Comparison - Atlassian, fecha de acceso: marzo 22, 2025, https://www.atlassian.com/continuous-delivery/continuous-integration/tools
18. 15 Must-Have CI/CD Tools for Developers in 2025 - Agilemania, fecha de acceso: marzo 22, 2025, https://agilemania.com/ci-cd-tool
19. Understanding CI/CD - AWS Prescriptive Guidance, fecha de acceso: marzo 22, 2025, https://docs.aws.amazon.com/prescriptive-guidance/latest/strategy-cicd-litmus/understanding-cicd.html
20. 50+ CI/CD Tools to Streamline DevOps Workflows - CloudZero, fecha de acceso: marzo 22, 2025, https://www.cloudzero.com/blog/cicd-tools/

21. 11 Popular Tools That Enable CI/CD You Need For DevOps | by Abhaya - Medium, fecha de acceso: marzo 22, 2025, https://medium.com/@abhaykhs/11-popular-tools-that-enable-ci-cd-you-need-for-devops-5d2c3c6f9a7e
22. 20 Best CI/CD Tools for 2025 - The CTO Club, fecha de acceso: marzo 22, 2025, https://thectoclub.com/tools/best-ci-cd-tools/
23. 10 Continuous Deployment Tools To Consider - ProsperOps, fecha de acceso: marzo 22, 2025, https://www.prosperops.com/blog/continuous-deployment-tools/
24. Mastering CI/CD Monitoring: Essential Tools and Best Practices - Edge Delta, fecha de acceso: marzo 22, 2025, https://edgedelta.com/company/blog/mastering-ci-cd-monitoring
25. Best Practices for CI/CD Monitoring - Datadog, fecha de acceso: marzo 22, 2025, https://www.datadoghq.com/blog/best-practices-for-ci-cd-monitoring/
26. 15 Great CI/CD Tools And How To Choose | The DevOps engineer's ..., fecha de acceso: marzo 22, 2025, https://octopus.com/devops/ci-cd/ci-cd-tools/
27. CICD with Jenkins - BrowserStack, fecha de acceso: marzo 22, 2025, https://www.browserstack.com/guide/ci-cd-with-jenkins
28. What Is Jenkins and How Does it Work? Intro and Tutorial - Codefresh, fecha de acceso: marzo 22, 2025, https://codefresh.io/learn/jenkins/
29. What is Jenkins and use cases of Jenkins? - DevOpsSchool.com, fecha de acceso: marzo 22, 2025, https://www.devopsschool.com/blog/what-is-jenkins-and-use-cases-of-jenkins/
30. Jenkins in a Nutshell | Logz.io, fecha de acceso: marzo 22, 2025, https://logz.io/blog/jenkins-in-a-nutshell/
31. What is Jenkins? Key Concepts & Tutorial - Spacelift, fecha de acceso: marzo 22, 2025, https://spacelift.io/blog/what-is-jenkins
32. What is Jenkins? Features and Architecture Explained [2024 Edition] | Simplilearn, fecha de acceso: marzo 22, 2025, https://www.simplilearn.com/tutorials/jenkins-tutorial/what-is-jenkins
33. Jenkins Master and Slave Architecture – A Complete Guide - Edureka, fecha de acceso: marzo 22, 2025, https://www.edureka.co/blog/jenkins-master-and-slave-architecture-a-complete-guide/
34. Jenkins Master and Slave Architecture | Edureka - Medium, fecha de acceso: marzo 22, 2025, https://medium.com/edureka/jenkins-master-and-slave-architecture-e3d6c4728945
35. What is Jenkins Pipeline? | A Complete Overview - Site24x7, fecha de acceso: marzo 22, 2025, https://www.site24x7.com/learn/jenkins-pipelines.html
36. Understanding Jenkins CI/CD Pipeline And Its Stages - GeeksforGeeks, fecha de acceso: marzo 22, 2025, https://www.geeksforgeeks.org/understanding-jenkins-ci-cd-pipeline-and-its-stages/
37. Jenkins: Pros/cons, Installation, And 8 Modern Alternatives - Octopus Deploy,

fecha de acceso: marzo 22, 2025, https://octopus.com/devops/jenkins/
38. Kubernetes vs Docker vs Jenkins: Know the Differences - Aspire Systems - blog, fecha de acceso: marzo 22, 2025, https://blog.aspiresys.com/infrastructure-managed-services/kubernetes-vs-docker-vs-jenkins-know-the-differences/
39. What Is Jenkins? How & Why To Use It? - LambdaTest, fecha de acceso: marzo 22, 2025, https://www.lambdatest.com/blog/what-is-jenkins/
40. 20 Popular CI/CD Tools to Simplify Your Deployment Pipeline - Axify, fecha de acceso: marzo 22, 2025, https://axify.io/blog/ci-cd-tools
41. CI/CD With Jenkins: A Gentle Introduction | Spot.io, fecha de acceso: marzo 22, 2025, https://spot.io/resources/ci-cd/ci-cd-with-jenkins-a-gentle-introduction/
42. Jenkins Pros & Cons (2020) - Cloud Posse, fecha de acceso: marzo 22, 2025, https://cloudposse.com/blog/devops/cicd/jenkins-pros-cons-2020
43. Why Should Stop Using Jenkins - Medium, fecha de acceso: marzo 22, 2025, https://medium.com/@boonyaritkun/why-should-stop-using-jenkins-25b1a566c13c
44. Why use Jenkins Over the Other Tools? Jenkins Pros And Cons. | by Andrey Byhalenko | DevOps Manuals and Technical Notes | Medium, fecha de acceso: marzo 22, 2025, https://medium.com/devops-technical-notes-and-manuals/why-use-jenkins-over-the-other-tools-jenkins-pros-and-cons-d2d4811ae81c
45. Industry Use Cases of Jenkins. QUICK OVERVIEW - Medium, fecha de acceso: marzo 22, 2025, https://medium.com/@simrankumari1344/industry-use-cases-of-jenkins-590f534ef405
46. Jenkins Project Reports Growth of 79% in Jenkins Pipeline, Used to Speed Software Delivery - CD Foundation, fecha de acceso: marzo 22, 2025, https://cd.foundation/announcement/2023/08/29/jenkins-project-growth/
47. Jenkins Architecture Explained - Beginners Guide To Jenkins Components - DevOpsCube, fecha de acceso: marzo 22, 2025, https://devopscube.com/jenkins-architecture-explained/
48. Jenkins Architecture: Learn Everything - The Knowledge Academy, fecha de acceso: marzo 22, 2025, https://www.theknowledgeacademy.com/blog/jenkins-architecture/
49. Creating your first Pipeline - Jenkins, fecha de acceso: marzo 22, 2025, https://www.jenkins.io/doc/pipeline/tour/hello-world/
50. Jenkins Pipeline: Examples, Usage, and Best Practices - Codefresh, fecha de acceso: marzo 22, 2025, https://codefresh.io/learn/jenkins/jenkins-pipeline-examples-usage-and-best-practices/
51. Pipeline - Jenkins, fecha de acceso: marzo 22, 2025, https://www.jenkins.io/doc/book/pipeline/
52. Pipeline Syntax - Jenkins, fecha de acceso: marzo 22, 2025, https://www.jenkins.io/doc/book/pipeline/syntax/
53. Using Declarative Pipeline syntax - CloudBees Documentation, fecha de acceso:

marzo 22, 2025, https://docs.cloudbees.com/docs/cloudbees-ci/latest/pipeline-syntax-reference-guide/declarative-pipeline
54. Declarative Pipeline With Jenkins - DZone Refcards, fecha de acceso: marzo 22, 2025, https://dzone.com/refcardz/declarative-pipeline-with-jenkins
55. Jenkins Declarative Pipelines: A Beginner-Friendly Guide | by Trupti Mane | Medium, fecha de acceso: marzo 22, 2025, https://truptimane.medium.com/jenkins-declarative-pipelines-a-beginner-friendly-guide-3ea008657e34
56. Understanding the Differences Between Jenkins Scripted and Declarative Pipeline: A Comprehensive Guide with Real-World Examples - Praveen Dandu, fecha de acceso: marzo 22, 2025, https://praveendandu24.medium.com/understanding-the-differences-between-jenkins-scripted-and-declarative-pipeline-a-comprehensive-960826e26c2
57. How to Use the Jenkins Scripted Pipeline | Blazemeter by Perforce, fecha de acceso: marzo 22, 2025, https://www.blazemeter.com/blog/jenkins-scripted-pipeline
58. Using a Jenkinsfile, fecha de acceso: marzo 22, 2025, https://www.jenkins.io/doc/book/pipeline/jenkinsfile/
59. Pipeline Syntax - GitHub Pages, fecha de acceso: marzo 22, 2025, https://tempora-mutantur.github.io/jenkins.io/github_pages_test/doc/book/pipeline/syntax/
60. What is the Jenkins pipeline? Types and stages of Jenkins CI:CD Pipelines - ACCELQ, fecha de acceso: marzo 22, 2025, https://www.accelq.com/blog/jenkins-pipeline/
61. Basic Jenkins Architecture: A Visual Guide | by Karthik S - Medium, fecha de acceso: marzo 22, 2025, https://karthiksdevopsengineer.medium.com/basic-jenkins-architecture-a-visual-guide-5c7467c6a7c9
62. Jenkins Master-Slave Architecture | automateNow, fecha de acceso: marzo 22, 2025, https://automatenow.io/jenkins-master-slave-architecture/
63. Understanding Master-Slave Architecture in Jenkins | CloudZenia, fecha de acceso: marzo 22, 2025, https://cloudzenia.com/blog/understanding-master-slave-architecture-in-jenkins/
64. Simplifying Jenkins Master-Slave Architecture : A Comprehensive Guide | by @Harsh, fecha de acceso: marzo 22, 2025, https://harsh05.medium.com/simplifying-jenkins-master-slave-architecture-a-comprehensive-guide-84f7de662346
65. New to using Jenkins and need some help clarifying the difference of using Jenkins in a docker container Vs kubernetes - Reddit, fecha de acceso: marzo 22, 2025, https://www.reddit.com/r/jenkinsci/comments/17j24ql/new_to_using_jenkins_and_need_some_help/
66. Jenkins vs Gitlab: Comparison of Top CI/CD Tools in 2024 - Folio3 Cloud Services, fecha de acceso: marzo 22, 2025, https://cloud.folio3.com/blog/jenkins-vs-gitlab/

67. Build and Run a Jenkins server on Azure - Alif Consulting, fecha de acceso: marzo 22, 2025, https://www.alifconsulting.com/post/jenkins-server-on-azure
68. Software Development with Jenkins: Real-World Use Cases | by Payoda Technology Inc, fecha de acceso: marzo 22, 2025, https://payodatechnologyinc.medium.com/software-development-with-jenkins-real-world-use-cases-0cfab743b1b6
69. Why Developers Struggle with Jenkins CI/CD - Inedo Blog, fecha de acceso: marzo 22, 2025, https://blog.inedo.com/jenkins/everybody-hates-jenkins/
70. Industry Use Cases of Jenkins - saurabh kharkate - Medium, fecha de acceso: marzo 22, 2025, https://saurabhkharkate05.medium.com/industry-use-cases-of-jenkins-a23fcd70f77e
71. Software Development with Jenkins: Real-World Use Cases - PAYODA, fecha de acceso: marzo 22, 2025, https://www.payoda.com/software-development-jenkins/
72. Use Cases of Jenkins:. What Is Jenkins? | by Anushka Nawale | Medium, fecha de acceso: marzo 22, 2025, https://medium.com/@anushkasnawale/use-cases-of-jenkins-2aeab1baaae3
73. Building Blocks of Jenkins: Understanding its Architecture | by BuildPiper - Medium, fecha de acceso: marzo 22, 2025, https://medium.com/buildpiper/building-blocks-of-jenkins-understanding-its-architecture-8b99151d6a0f
74. Get Started - Install Jenkins on an Azure Linux VM - Microsoft Learn, fecha de acceso: marzo 22, 2025, https://learn.microsoft.com/en-us/azure/developer/jenkins/configure-on-linux-vm
75. Jenkins Installation on Azure VM. To launch a virtual machine (VM)... | by Subham Pradhan | Medium, fecha de acceso: marzo 22, 2025, https://medium.com/@subhampradhan966/jenkins-installation-on-azure-vm-41bb38d4e96f
76. How to Setup Jenkins Locally With Oracle VirtualBox VM - DEV Community, fecha de acceso: marzo 22, 2025, https://dev.to/gabbyti/how-to-setup-jenkins-locally-with-oracle-virtualbox-vm-3cmi
77. Linux - Jenkins, fecha de acceso: marzo 22, 2025, https://www.jenkins.io/doc/book/installing/linux/
78. Step-by-Step Guide to Setting Up Jenkins on Docker with Docker Agent-Based Builds, fecha de acceso: marzo 22, 2025, https://dev.to/msrabon/step-by-step-guide-to-setting-up-jenkins-on-docker-with-docker-agent-based-builds-43j5
79. Setup Jenkins on Docker: Windows/Linux Tutorial & Best Practices | Codefresh, fecha de acceso: marzo 22, 2025, https://codefresh.io/learn/jenkins/setup-jenkins-on-docker-windows-linux-tutorial-best-practices/
80. Install Jenkins on a VM - KodeKloud Notes, fecha de acceso: marzo 22, 2025, https://notes.kodekloud.com/docs/Jenkins/Installing-Jenkins/Install-Jenkins-on-a

-VM
81. Install Jenkins using Docker - Medium, fecha de acceso: marzo 22, 2025, https://medium.com/@eloufirhatim/install-jenkins-using-docker-e76f41f79682
82. How to install Jenkins on Docker - Octopus Deploy, fecha de acceso: marzo 22, 2025, https://octopus.com/blog/jenkins-docker-install-guide
83. Docker - Jenkins, fecha de acceso: marzo 22, 2025, https://www.jenkins.io/doc/book/installing/docker/
84. What are the advantages of running Jenkins in a docker container - Stack Overflow, fecha de acceso: marzo 22, 2025, https://stackoverflow.com/questions/44440164/what-are-the-advantages-of-running-jenkins-in-a-docker-container
85. jenkins on the same virtual machine as critical application, is it wise? - Stack Overflow, fecha de acceso: marzo 22, 2025, https://stackoverflow.com/questions/41633415/jenkins-on-the-same-virtual-machine-as-critical-application-is-it-wise
86. Why Run Jenkins In A Docker Container - Restack, fecha de acceso: marzo 22, 2025, https://www.restack.io/p/advanced-docker-networking-strategies-answer-jenkins-docker
87. Mastering Docker and Jenkins: Build Robust CI/CD Pipelines Efficiently, fecha de acceso: marzo 22, 2025, https://www.docker.com/blog/docker-and-jenkins-build-robust-ci-cd-pipelines/
88. Revolutionizing Jenkins with Dynamic Docker Agents: A Comprehensive Guide | by @Harsh, fecha de acceso: marzo 22, 2025, https://harsh05.medium.com/revolutionizing-jenkins-with-dynamic-docker-agents-a-comprehensive-guide-9932f5dd313a
89. What's the benefits of docker with Jenkins Pipelines? - Stack Overflow, fecha de acceso: marzo 22, 2025, https://stackoverflow.com/questions/48181181/whats-the-benefits-of-docker-with-jenkins-pipelines
90. Jenkins and Docker: A Dangerous Liaison and Kubernetes with Kaniko as an Alternative, fecha de acceso: marzo 22, 2025, https://cdelmonte.medium.com/jenkins-and-docker-a-dangerous-liaison-and-kubernetes-with-kaniko-as-an-alternative-fc34336ec973
91. Jenkins – Pros and Cons - Ezeelive Technologies, fecha de acceso: marzo 22, 2025, https://ezeelive.com/jenkins-pros-cons/
92. Docker-In-Docker: The Good And the Bad | by Parameshwar bhat | Medium, fecha de acceso: marzo 22, 2025, https://medium.com/@parameshwarbhat411/docker-in-docker-the-good-and-the-bad-48cfe4e0da6e
93. Jenkins GitHub Webhook build trigger by example - The Server Side, fecha de acceso: marzo 22, 2025, https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Jenkins-GitHub-Webhook-example-no-403-crumb-error
94. How to configure Webhook in GitHub and Jenkins for automatic trigger with

CICD pipeline?, fecha de acceso: marzo 22, 2025, https://medium.com/@sangeetv09/how-to-configure-webhook-in-github-and-jenkins-for-automatic-trigger-with-cicd-pipeline-34133e9de0ea
95. Jenkins Pipeline – Git Checkout With Credentials | Baeldung on Ops, fecha de acceso: marzo 22, 2025, https://www.baeldung.com/ops/jenkins-pipeline-git-checkout-credentials
96. 02 - Jenkins Declarative Pipeline tutorial | Git Checkout - YouTube, fecha de acceso: marzo 22, 2025, https://www.youtube.com/watch?v=bf4111voUlw
97. Git plugin - Jenkins, fecha de acceso: marzo 22, 2025, https://www.jenkins.io/doc/pipeline/steps/git/
98. Building a CI Pipeline with Jenkins (Pipeline) | by Nova Novriansyah | NovAI- Agile & DevOPS 101 | Medium, fecha de acceso: marzo 22, 2025, https://medium.com/novai-devops-101/building-a-ci-pipeline-with-jenkins-pipeline-2ac3d6bb391a
99. Jenkins task for remote hosts - Stack Overflow, fecha de acceso: marzo 22, 2025, https://stackoverflow.com/questions/39659980/jenkins-task-for-remote-hosts
100. Jenkins: How to make single job build and deploy on two servers - Stack Overflow, fecha de acceso: marzo 22, 2025, https://stackoverflow.com/questions/15031415/jenkins-how-to-make-single-job-build-and-deploy-on-two-servers
101. Using Docker with Pipeline - Jenkins, fecha de acceso: marzo 22, 2025, https://www.jenkins.io/doc/book/pipeline/docker/