Informe sobre la Certificación LPIC DevOps Tools: Módulo 1 Ingeniería de Software

La certificación LPIC DevOps Tools Engineer se ha desarrollado para validar las habilidades esenciales necesarias para la utilización de herramientas que optimizan la colaboración dentro de los flujos de trabajo de la administración de sistemas y el desarrollo de software [1]. Esta certificación se centra en las competencias prácticas requeridas para operar de manera efectiva en un entorno DevOps, haciendo énfasis en el uso de las herramientas más relevantes en este campo [1]. El primer módulo de esta certificación se dedica a la Ingeniería de Software [2]. La creciente demanda de profesionales de tecnologías de la información con conocimientos en DevOps subraya la importancia de esta certificación en la industria actual [1].

Los profesionales que buscan obtener la certificación LPIC DevOps Tools deben demostrar un conocimiento práctico de áreas relacionadas con DevOps, tales como la ingeniería y arquitectura de software, el despliegue de contenedores y máquinas virtuales, la gestión de la configuración y la monitorización de sistemas [1]. La certificación también evalúa la competencia en el uso de utilidades de código abierto prominentes en el ámbito de DevOps, incluyendo Docker, Vagrant, Ansible, Puppet, Git y Jenkins [1]. Si bien no se establecen prerrequisitos formales para tomar el examen, se recomienda encarecidamente que los candidatos posean una certificación adicional en su área principal de especialización, como LPIC-1 o una certificación de desarrollador [1]. El examen tiene una duración de 90 minutos y consta de 60 preguntas de opción múltiple y de completar espacios en blanco [1].

El Módulo 1, Ingeniería de Software, abarca los aspectos fundamentales del desarrollo de software moderno dentro del contexto de las prácticas de DevOps [3]. Este módulo se adentra en temas cruciales como el desarrollo de software moderno, la gestión del código fuente y la integración y entrega continua de software.

Módulo 1: Ingeniería de Software - Objetivos Detallados

Objetivo 701.1: Desarrollo de Software Moderno (Peso: 6) [6]

Aplicaciones Basadas en Servicios [6]

Un aspecto central del desarrollo de software moderno en el contexto de DevOps es la comprensión y el diseño de aplicaciones basadas en servicios [6]. Este paradigma arquitectónico permite la construcción de sistemas complejos mediante la composición de servicios independientes y autónomos. La adopción de este enfoque facilita la escalabilidad, la flexibilidad y la capacidad de realizar entregas continuas de software. La familiaridad con las plataformas de nube más comunes, como AWS, Azure y GCP, y los diversos servicios que

ofrecen, resulta crucial para el diseño e implementación de arquitecturas de software contemporáneas [2].

La capacidad de integrar estos servicios proporcionados por la nube en la arquitectura de las aplicaciones y en las cadenas de herramientas de despliegue es un requisito fundamental [7]. Esto implica comprender la configuración necesaria para que los servicios interactúen correctamente y cumplan con los requerimientos de la aplicación. Además, es esencial tener conciencia de los objetivos de las operaciones de TI y del aprovisionamiento de servicios, incluyendo atributos no funcionales como la disponibilidad, la latencia y la capacidad de respuesta [6]. La importancia de las aplicaciones basadas en servicios en DevOps radica en su habilidad para habilitar la escalabilidad independiente de los componentes, la flexibilidad para adoptar nuevas tecnologías y la agilidad para realizar entregas de software de manera frecuente y confiable. La familiaridad con los principales proveedores de la nube y sus ofertas de servicios es, por lo tanto, un conocimiento indispensable para diseñar e implementar arquitecturas modernas que aprovechen al máximo los beneficios del paradigma DevOps.

Conceptos y Estándares de API [6]

El dominio de los conceptos y estándares comunes de las Interfaces de Programación de Aplicaciones (API) es otro pilar del desarrollo de software moderno [6]. Las APIs actúan como intermediarios, permitiendo la comunicación y el intercambio de datos entre diferentes aplicaciones de software [13]. Es importante comprender los diversos tipos de APIs disponibles, como REST, GraphQL, SOAP, gRPC y WebSocket, cada uno con sus propias características y casos de uso [13]. Asimismo, se debe tener un conocimiento claro de los componentes fundamentales de una API, incluyendo los puntos de conexión (endpoints), los métodos HTTP (GET, POST, PUT, DELETE), las solicitudes y respuestas, y los códigos de estado [13].

Adherirse a las mejores prácticas de diseño de APIs es esencial para construir sistemas robustos y mantenibles. Estas prácticas incluyen la implementación de medidas de seguridad como el uso de HTTPS, el seguimiento de los principios de la arquitectura RESTful para la simplicidad y la consistencia, el diseño para la escalabilidad para manejar grandes volúmenes de datos y tráfico de manera eficiente, la implementación de limitación de velocidad (rate limiting) para prevenir el abuso, y el uso de nombres significativos para los puntos de conexión y los recursos [13]. Además, es crucial comprender los mecanismos de autenticación y autorización, como las claves de API, los tokens JWT y OAuth2, para asegurar el acceso adecuado a los recursos de la API [13]. Finalmente, la importancia de una documentación de API clara y completa no puede ser subestimada, ya que facilita la adopción y el uso de la API por parte de otros desarrolladores [14]. Las APIs constituyen la base de la comunicación en las aplicaciones modernas, especialmente en arquitecturas de microservicios, y un conocimiento sólido de sus conceptos y estándares es fundamental para la integración, la colaboración y la eficiencia en el desarrollo de software.

Tabla 1: Códigos de Estado HTTP Comunes

Código de Estado	Descripción
200 OK	La solicitud ha tenido éxito.
201 Created	La solicitud ha tenido éxito y se ha creado un nuevo recurso.
400 Bad Request	La solicitud no se pudo entender o fue mal formada.
401 Unauthorized	Se requiere autenticación y no se ha proporcionado o no es válida.
403 Forbidden	El servidor entendió la solicitud, pero se niega a autorizarla.
404 Not Found	El servidor no pudo encontrar el recurso solicitado.
500 Internal Server Error	El servidor encontró una condición inesperada que le impidió cumplir la solicitud.

Manejo de Almacenamiento de Datos y Sesiones [6]

La gestión adecuada del almacenamiento de datos y el manejo de sesiones son aspectos críticos en el desarrollo de aplicaciones basadas en servicios [6]. Es fundamental comprender los diversos aspectos relacionados con el almacenamiento de datos, el estado de los servicios y la gestión de las sesiones de usuario [6]. En particular, las arquitecturas de microservicios presentan desafíos únicos en comparación con las aplicaciones monolíticas en lo que respecta al manejo de sesiones, ya que los servicios se encuentran distribuidos y pueden ser independientes [18].

Existen diversas estrategias para la gestión de sesiones en entornos de microservicios, incluyendo la autenticación basada en tokens (como JWT), el almacenamiento centralizado de sesiones en una base de datos o caché distribuida accesible por todos los servicios, las sesiones del lado del cliente donde la información de la sesión se almacena en el navegador del usuario, y el uso de almacenamiento en caché distribuido como Redis o Memcached para compartir datos de sesión entre servicios [18]. También es importante considerar la persistencia de los datos en las aplicaciones basadas en servicios y ser consciente de las implicaciones de la elección de la base de datos para cada servicio, como la diferencia entre bases de datos SQL y NoSQL, en términos de escalabilidad, consistencia y tipos de datos soportados [6]. La selección de la estrategia correcta para la gestión de sesiones y la persistencia de datos tiene

un impacto significativo en la escalabilidad, la seguridad y la experiencia del usuario en aplicaciones distribuidas.

Diseño de Software para Contenedores [6]

El diseño de software para su ejecución en contenedores se ha convertido en una práctica fundamental en el desarrollo moderno [6]. La containerización es el proceso de empaquetar una aplicación junto con todas sus dependencias (bibliotecas, archivos de configuración, etc.) en una unidad portátil llamada contenedor [23]. Esta tecnología ofrece numerosos beneficios en el contexto de DevOps, incluyendo la portabilidad de las aplicaciones entre diferentes entornos (desarrollo, pruebas, producción), el aislamiento de las aplicaciones para evitar conflictos de dependencias, y una mayor eficiencia en el uso de los recursos del sistema [23].

Docker es una de las herramientas clave en el ámbito de la containerización [7]. Permite a los desarrolladores crear Dockerfiles, que son archivos de texto que contienen las instrucciones para construir una imagen de Docker. Una imagen de Docker es una plantilla ligera y ejecutable que contiene todo lo necesario para ejecutar una aplicación en un contenedor [7]. Estas imágenes pueden almacenarse y compartirse en un registro de Docker. Una vez que se tiene una imagen, se pueden crear y operar contenedores a partir de ella, conectándolos a redes y volúmenes de almacenamiento según sea necesario [7]. La containerización simplifica el despliegue de aplicaciones y garantiza la consistencia en diferentes entornos, lo que la convierte en un componente esencial de las prácticas de DevOps.

Diseño de Software para la Nube [6]

El diseño de software destinado a ser implementado en servicios de nube requiere una comprensión de las características y los servicios específicos que ofrecen las plataformas en la nube [6]. Existen diversas estrategias de despliegue en la nube que se pueden adoptar, como el despliegue blue/green (donde se mantiene un entorno activo y otro inactivo, y el tráfico se conmuta entre ellos), el despliegue canary (donde una nueva versión se implementa para un pequeño subconjunto de usuarios antes de un lanzamiento completo), y las actualizaciones continuas (rolling updates) donde las instancias de la aplicación se actualizan de forma gradual [28].

Al diseñar para la nube, es fundamental tener en cuenta los servicios ofrecidos por plataformas comunes como AWS, Azure y GCP, y cómo estos servicios pueden integrarse en la arquitectura de la aplicación [7]. Además, se deben considerar aspectos como la escalabilidad (la capacidad de la aplicación para manejar un aumento en la carga de trabajo), la disponibilidad (el tiempo durante el cual la aplicación está operativa y accesible) y la tolerancia a fallos (la capacidad de la aplicación para seguir funcionando en caso de fallos en la infraestructura) [6]. La elección de la estrategia de despliegue adecuada y el diseño de la aplicación teniendo en cuenta la escalabilidad y la disponibilidad son factores clave para garantizar una experiencia de usuario óptima en entornos de nube.

Consideraciones sobre Software Monolítico Legado [6]

Muchas organizaciones se enfrentan al desafío de migrar o integrar software monolítico heredado con nuevas arquitecturas basadas en servicios o en la nube [6]. Es importante ser

consciente de los riesgos inherentes a este proceso, como la complejidad de la migración, la posible interrupción del servicio y la dificultad de mantener y escalar aplicaciones monolíticas en comparación con las arquitecturas modernas.

Existen diversas estrategias que se pueden emplear para abordar la migración de software monolítico, incluyendo la re-arquitectura completa de la aplicación en microservicios, el refactoring del código existente para mejorar su modularidad y la adopción del patrón "strangler", que implica la construcción gradual de nuevos servicios alrededor de la aplicación monolítica hasta que finalmente la reemplacen. Cada una de estas estrategias tiene sus propias ventajas y desventajas, y la elección de la más adecuada dependerá de factores como la complejidad de la aplicación, los recursos disponibles y los objetivos de la migración.

Seguridad en el Desarrollo de Software [6]

La seguridad es una consideración primordial en todas las etapas del desarrollo de software, y esto es especialmente cierto en el contexto de DevOps, donde la velocidad y la automatización son elementos clave [6]. Es fundamental comprender los riesgos comunes de seguridad que enfrentan las aplicaciones, como las vulnerabilidades de inyección, las fugas de datos y los ataques de denegación de servicio, y conocer las formas de mitigarlos [6].

La integración de la seguridad en el ciclo de vida del desarrollo de software, a menudo denominada DevSecOps, es una práctica esencial [9]. Esto implica incorporar consideraciones de seguridad desde las fases iniciales del diseño y la planificación, en lugar de tratarlas como una ocurrencia tardía. Algunas prácticas importantes incluyen la adopción de principios de codificación segura para evitar la introducción de vulnerabilidades, la realización de pruebas de seguridad automatizadas (tanto análisis estático como dinámico de código), la gestión segura de secretos (como contraseñas y claves de API) y la configuración segura de los entornos de despliegue.

Desarrollo Ágil de Software [6]

El desarrollo ágil de software es un enfoque iterativo e incremental que enfatiza la colaboración, la flexibilidad y la entrega temprana y continua de software funcional [6]. Los principios del Manifiesto Ágil, que incluyen la satisfacción del cliente mediante la entrega temprana y continua de software valioso, la bienvenida a los cambios en los requisitos incluso en etapas tardías del desarrollo, la entrega frecuente de software funcional (en semanas en lugar de meses), la colaboración estrecha entre personas de negocio y desarrolladores, la construcción de proyectos en torno a individuos motivados a los que se debe confiar, la conversación cara a cara como la forma más eficiente y efectiva de comunicar información, el software funcional como la principal medida de progreso, el desarrollo sostenible capaz de mantener un ritmo constante, la atención continua a la excelencia técnica y al buen diseño, la simplicidad (el arte de maximizar la cantidad de trabajo no realizado), las mejores arquitecturas, requisitos y diseños que emergen de equipos autoorganizados, y la reflexión regular del equipo sobre cómo ser más efectivo para luego ajustar su comportamiento en consecuencia, son fundamentales para comprender la filosofía ágil [37].

Existen diversas metodologías ágiles, siendo Scrum y Kanban dos de las más comunes [29]. La adopción de principios y prácticas ágiles es fundamental en DevOps, ya que permite a los

equipos responder rápidamente a los cambios, obtener retroalimentación temprana de los usuarios y realizar mejoras continuas en el software.

Implicaciones de DevOps en la Ingeniería de Software [1]

DevOps representa una evolución en la forma en que se desarrolla y opera el software, buscando una mayor colaboración y automatización entre los equipos de desarrollo (Dev) y operaciones (Ops) [1]. Esto tiene profundas implicaciones en la ingeniería de software. Uno de los principios clave de DevOps es la colaboración estrecha entre los equipos de desarrollo y operaciones a lo largo de todo el ciclo de vida del software, desde la planificación hasta el despliegue y la monitorización [33].

La automatización es otro pilar fundamental de DevOps, aplicándose a diversas tareas como la construcción, las pruebas, el despliegue y la gestión de la infraestructura (a través de la Infraestructura como Código o IaC) [29]. Esto conduce a la entrega continua de software, donde los cambios se integran, prueban y despliegan en producción de manera frecuente y automatizada [29]. La monitorización continua de las aplicaciones y la infraestructura en producción, junto con la recopilación y el análisis de retroalimentación, permiten a los equipos identificar y resolver problemas rápidamente, así como realizar mejoras basadas en datos reales [29]. Finalmente, DevOps promueve una cultura de responsabilidad compartida, donde todos los miembros del equipo son responsables del éxito del producto o servicio [33]. En resumen, DevOps busca optimizar el proceso de desarrollo y entrega de software, logrando mayor velocidad, eficiencia y calidad mediante la colaboración, la automatización y la mejora continua.

Objetivo 701.2: Gestión de Código Fuente (Peso: 7) [6]

Uso de Git para la Gestión de Archivos [6]

Git es un sistema de control de versiones distribuido ampliamente utilizado en el desarrollo de software moderno, especialmente en entornos DevOps [6]. Permite a los desarrolladores administrar archivos dentro de un repositorio, realizando operaciones como agregar nuevos archivos, modificar el contenido de los existentes y eliminar los que ya no son necesarios [6]. Los comandos básicos de Git para la manipulación de archivos incluyen add para preparar los cambios para la siguiente confirmación (commit), commit para guardar los cambios en el repositorio, status para mostrar el estado de los archivos en el directorio de trabajo y el área de preparación, y log para ver el historial de confirmaciones [8]. Git rastrea cada cambio realizado en el código fuente, lo que facilita la colaboración y la gestión de diferentes versiones del software.

Tabla 2: Comandos Básicos de Git para la Gestión de Archivos

Comando	Descripción
git add <archivo>	Añade el archivo especificado al área de preparación (staging area).
git commit -m "<mensaje>"	Guarda los cambios del área de preparación en el repositorio local con un mensaje descriptivo.
git status	Muestra el estado del directorio de trabajo y del área de preparación.
git log	Muestra el historial de confirmaciones (commits).
git rm <archivo>	Elimina el archivo especificado del directorio de trabajo y del área de preparación.
git mv <archivo_antiguo> <archivo_nuevo>	Renombra o mueve el archivo especificado.

Manejo de Ramas y Etiquetas en Git [6]

Git permite a los desarrolladores trabajar en diferentes funcionalidades o correcciones de errores de forma aislada mediante el uso de ramas [6]. Se pueden crear nuevas ramas a partir de la rama principal (normalmente llamada main o master) utilizando el comando git branch <nombre_rama>. Para cambiar a una rama diferente, se utiliza el comando git checkout <nombre_rama>. Los cambios realizados en una rama no afectan a otras ramas hasta que se realiza una operación de fusión (merge). Para eliminar una rama, se utiliza el comando git branch -d <nombre_rama>.

Las etiquetas (tags) se utilizan para marcar puntos específicos en la historia del repositorio, como versiones publicadas del software [6]. Se pueden crear etiquetas utilizando el comando git tag <nombre_etiqueta>. Para crear una etiqueta con un mensaje, se utiliza git tag -a <nombre_etiqueta> -m "<mensaje>". Las etiquetas se pueden listar con git tag y eliminar con git tag -d <nombre_etiqueta>. Las ramas facilitan el desarrollo paralelo y la gestión de diferentes líneas de desarrollo, mientras que las etiquetas proporcionan una forma de referenciar versiones específicas del código.

Tabla 3: Comandos de Git para la Gestión de Ramas y Etiquetas

Comando	Descripción
git branch <nombre_rama>	Crea una nueva rama con el nombre especificado.
git checkout <nombre_rama>	Cambia a la rama especificada.
git merge <nombre_rama>	Fusiona los cambios de la rama especificada en la rama actual.
git branch -d <nombre_rama>	Elimina la rama especificada (solo si los cambios han sido fusionados).
git tag <nombre_etiqueta>	Crea una etiqueta ligera en la confirmación actual.
git tag -a <nombre_etiqueta> -m "<mensaje>"	Crea una etiqueta anotada con un mensaje.
git tag -d <nombre_etiqueta>	Elimina la etiqueta especificada.

Trabajo con Repositorios Remotos y Submódulos en Git [6]

Git permite trabajar con repositorios remotos, que son versiones del repositorio que se almacenan en un servidor y facilitan la colaboración entre varios desarrolladores [6]. Para clonar un repositorio remoto a la máquina local, se utiliza el comando git clone <URL_repositorio>. Se pueden agregar, eliminar y listar repositorios remotos configurados para un proyecto local con el comando git remote. Para enviar los cambios locales a un repositorio remoto, se utiliza git push <nombre_remoto> <nombre_rama>. Para obtener los cambios de un repositorio remoto sin integrarlos en la rama local, se utiliza git fetch <nombre_remoto>. Para obtener los cambios e integrarlos en la rama local, se utiliza git pull <nombre_remoto> <nombre_rama>.

Los submódulos en Git permiten incluir otros repositorios de Git dentro de un proyecto principal, manteniendo los submódulos como repositorios independientes [6]. Para agregar un submódulo, se utiliza git submodule add <URL_repositorio> <ruta_submódulo>. Para inicializar y actualizar los submódulos después de clonar un repositorio con submódulos, se utiliza git submodule update --init --recursive. Los repositorios remotos son esenciales para la colaboración en proyectos de software, mientras que los submódulos son útiles para gestionar dependencias con otros proyectos de Git.

Tabla 4: Comandos de Git para Repositorios Remotos y Submódulos

Comando	Descripción
git clone <URL_repositorio>	Clona un repositorio remoto a la máquina local.
git remote add <nombre_remoto> <URL_remoto>	Agrega un nuevo repositorio remoto con el nombre y la URL especificados.
git remote remove <nombre_remoto>	Elimina el repositorio remoto con el nombre especificado.
git push <nombre_remoto> <nombre_rama>	Envía las confirmaciones de la rama local al repositorio remoto.
git pull <nombre_remoto> <nombre_rama>	Obtiene los cambios del repositorio remoto y los integra en la rama local.
git fetch <nombre_remoto>	Descarga los objetos y las referencias del repositorio remoto.
git submodule add <URL_repositorio> <ruta_submódulo>	Agrega un nuevo submódulo al repositorio.
git submodule update --init --recursive	Inicializa y actualiza los submódulos del repositorio.

Fusión de Código con Git [6]

La fusión (merge) en Git es el proceso de integrar los cambios de una rama en otra [6]. Normalmente, se fusiona una rama de desarrollo o de características en la rama principal. Para realizar una fusión, se cambia a la rama de destino (por ejemplo, main) y se ejecuta el comando git merge <nombre_rama_a_fusionar>. Git intenta realizar la fusión automáticamente, pero a veces pueden surgir conflictos si se han realizado cambios en las mismas líneas de código en ambas ramas. En estos casos, es necesario resolver los conflictos manualmente editando los archivos afectados y luego confirmando los cambios fusionados.

Las solicitudes de extracción (pull requests) son una práctica común en plataformas como GitHub o GitLab para facilitar la revisión de código antes de la fusión. Un desarrollador crea una solicitud de extracción para proponer los cambios de una rama para que sean revisados por

otros miembros del equipo antes de ser fusionados en la rama principal. La fusión de código es esencial para integrar el trabajo de diferentes desarrolladores y para incorporar nuevas funcionalidades o correcciones de errores en la base de código principal.

Tabla 5: Comandos de Git para la Fusión de Código

Comando	Descripción
git merge <nombre_rama>	Fusiona la rama especificada en la rama actual.
git mergetool	Abre una herramienta gráfica para ayudar a resolver conflictos de fusión.
Resolución de Conflictos	Implica editar manualmente los archivos con conflictos, marcar las secciones resueltas y luego usar git add y git commit para finalizar la fusión.
Pull Requests	Mecanismo en plataformas de colaboración (GitHub, GitLab) para solicitar la revisión de código antes de la fusión.

Introducción a SVN y CVS [6]

SVN (Subversion) y CVS (Concurrent Versions System) son sistemas de control de versiones más antiguos que Git [6]. A diferencia de Git, que es un sistema de control de versiones distribuido, SVN y CVS son sistemas centralizados. En un sistema centralizado, existe un único repositorio central donde se almacena todo el historial del proyecto, y los desarrolladores trabajan en copias locales del repositorio.

En un sistema distribuido como Git, cada desarrollador tiene una copia completa del repositorio, incluyendo todo el historial de cambios. Esto ofrece varias ventajas, como la posibilidad de trabajar sin conexión a la red y una mayor flexibilidad en los flujos de trabajo. Si bien Git es el sistema de control de versiones predominante en la actualidad, es útil tener un conocimiento básico de SVN y CVS, especialmente si se trabaja en proyectos heredados o se necesita interactuar con sistemas que aún los utilizan. La comprensión de las diferencias entre los modelos centralizados y distribuidos proporciona un contexto más amplio para la gestión del código fuente y ayuda a apreciar las ventajas que ofrece Git en el entorno actual de desarrollo

de software.

Objetivo 701.3: Integración Continua y Entrega Continua (Peso: 5) [6]

Componentes de un Pipeline de CI/CD [5]

Un pipeline de Integración Continua y Entrega Continua (CI/CD) es un conjunto de prácticas y herramientas que automatizan el proceso de construcción, prueba y despliegue de software [5]. Un pipeline típico de CI/CD consta de varias etapas. La etapa de **Build** (construcción) implica compilar el código fuente y generar los artefactos de software. La etapa de **Test** (prueba) incluye la ejecución de diversas pruebas automatizadas, como pruebas unitarias (que verifican el funcionamiento de componentes individuales del código), pruebas de integración (que verifican la interacción entre diferentes componentes) y pruebas de aceptación (que verifican que el software cumple con los requisitos del usuario). La etapa de **Package** (empaquetado) consiste en preparar los artefactos de software para su despliegue. La etapa de **Release** (liberación) implica la preparación de una versión específica del software para su despliegue en un entorno de destino. La etapa de **Deploy** (despliegue) es el proceso de instalar la versión del software en el entorno de destino (que puede ser un entorno de pruebas, de pre-producción o de producción). Finalmente, la etapa de **Monitor** (monitorización) implica la supervisión continua del rendimiento y la salud de la aplicación en el entorno de producción. Comprender cada una de estas etapas y su propósito es fundamental para diseñar e implementar un proceso de entrega continua efectivo.

Tabla 6: Etapas Comunes de un Pipeline de CI/CD

Etapa	Descripción
Build	Compilación del código fuente y generación de artefactos.
Test	Ejecución de pruebas unitarias, de integración y de aceptación automatizadas.
Package	Preparación de los artefactos para el despliegue.
Release	Preparación de una versión específica para el despliegue.

Deploy	Instalación de la versión en el entorno de destino.
Monitor	Supervisión continua del rendimiento y la salud de la aplicación.

Mejores Prácticas de Despliegue [6]

Existen varias mejores prácticas que se deben seguir al desplegar software para garantizar un proceso fluido y confiable [6]. Una de ellas es la automatización del proceso de despliegue, lo que reduce la posibilidad de errores humanos y acelera el tiempo de entrega [29]. También es importante elegir la estrategia de despliegue adecuada para cada situación. Algunas estrategias comunes incluyen el despliegue blue/green, donde se mantiene un entorno activo y otro inactivo, y el tráfico se conmuta entre ellos para minimizar el tiempo de inactividad [28]. El despliegue canary implica liberar una nueva versión a un pequeño subconjunto de usuarios antes de un lanzamiento completo para identificar posibles problemas [28]. Las actualizaciones continuas (rolling updates) implican actualizar las instancias de la aplicación de forma gradual, lo que también ayuda a minimizar el tiempo de inactividad [28]. Finalmente, es crucial contar con una estrategia de rollback clara y probada para poder revertir rápidamente a una versión anterior en caso de fallos durante el despliegue [30]. La elección de la estrategia de despliegue adecuada y la automatización del proceso son fundamentales para minimizar el tiempo de inactividad, reducir los riesgos y garantizar una entrega de software fluida y confiable.

Introducción y Arquitectura de Jenkins [6]

Jenkins es una herramienta de automatización de código abierto ampliamente utilizada para implementar pipelines de CI/CD [6]. Su arquitectura se basa en un nodo maestro (master) que coordina el trabajo y uno o más nodos agentes (agents) que ejecutan las tareas de construcción, prueba y despliegue. El nodo maestro es responsable de la configuración, la programación de los trabajos y la presentación de la interfaz de usuario, mientras que los nodos agentes realizan el trabajo real. Jenkins ofrece una amplia gama de funcionalidades, incluyendo la construcción automatizada de software, la ejecución de pruebas automatizadas y el despliegue de aplicaciones en diferentes entornos. Su arquitectura flexible y su extenso ecosistema de plugins lo convierten en una opción popular para automatizar el proceso de entrega de software.

Creación y Gestión de Trabajos en Jenkins [6]

En Jenkins, la unidad fundamental de automatización es el trabajo (job) [6]. Un trabajo define una secuencia de pasos que se ejecutan para realizar una tarea específica, como construir el código, ejecutar pruebas o desplegar una aplicación. Los trabajos se pueden configurar a través de la interfaz de usuario de Jenkins o mediante la definición de un archivo de configuración (Jenkinsfile). Al crear un trabajo, se pueden definir parámetros que permiten personalizar su comportamiento en tiempo de ejecución. Los trabajos se pueden gestionar (crear, editar, eliminar, habilitar, deshabilitar) a través de la interfaz de usuario de Jenkins. La capacidad de definir y gestionar trabajos de manera flexible es una de las características clave

de Jenkins.

Implementación de Pipelines de Entrega Continua con Jenkins [6]

Jenkins permite modelar todo el flujo de trabajo de CI/CD como un pipeline, utilizando el Plugin de Pipeline [6]. Un pipeline se define como código en un archivo llamado Jenkinsfile, que se almacena en el repositorio de código fuente junto con el código de la aplicación. Esto facilita la gestión, el versionado y la compartición del pipeline. Jenkins soporta dos tipos de sintaxis para definir pipelines: Declarativa y Scripted. La sintaxis Declarativa proporciona una estructura más sencilla y legible para definir las etapas del pipeline, mientras que la sintaxis Scripted ofrece una mayor flexibilidad pero es más compleja. Un pipeline declarativo se compone de diferentes etapas (stages) que representan las diferentes fases del proceso de entrega continua (por ejemplo, Build, Test, Deploy). El Plugin de Pipeline revolucionó la forma en que se definen los flujos de trabajo de CI/CD en Jenkins, permitiendo una mayor claridad, trazabilidad y capacidad de mantenimiento.

Seguridad en Jenkins [6]

La seguridad es una consideración importante al utilizar Jenkins, ya que gestiona el código fuente y los procesos de despliegue de las aplicaciones [6]. Jenkins ofrece varios modelos de autenticación para controlar quién puede acceder a la herramienta, incluyendo la autenticación basada en usuarios y contraseñas, la autenticación delegada a un servidor LDAP o Active Directory, y la autenticación basada en tokens. Una vez que un usuario está autenticado, se pueden configurar permisos para controlar qué acciones puede realizar (autorización). Es fundamental configurar correctamente la autenticación y la autorización para proteger el acceso a Jenkins y a los recursos que gestiona. Además, Jenkins permite la gestión segura de credenciales (como contraseñas y claves de API) que se utilizan para interactuar con otros sistemas durante el proceso de CI/CD.

Plugins Relevantes de Jenkins [6]

La funcionalidad de Jenkins se puede extender mediante el uso de plugins [6]. Existen numerosos plugins disponibles que proporcionan integración con una amplia gama de otras herramientas y tecnologías utilizadas en el desarrollo de software y DevOps. Algunos plugins relevantes para DevOps incluyen:

- **Copy Artifact Plugin:** Permite copiar artefactos de una construcción de Jenkins a otra.
- **Fingerprint Plugin:** Realiza un seguimiento de los archivos utilizados en las construcciones para facilitar la depuración y el seguimiento de dependencias.
- **Docker Pipeline Plugin:** Proporciona funciones para construir y gestionar imágenes de Docker dentro de un pipeline de Jenkins.
- **Docker Build and Publish plugin:** Simplifica la construcción y publicación de imágenes de Docker.
- **Git Plugin:** Proporciona integración con sistemas de control de versiones Git.
- **Credentials Plugin:** Permite almacenar y gestionar credenciales de forma segura para su uso en los trabajos de Jenkins.

Comprender la funcionalidad de estos y otros plugins relevantes es importante para aprovechar

al máximo las capacidades de Jenkins en un entorno DevOps.

Introducción a Artifactory y Nexus [6]

Artifactory y Nexus son herramientas conocidas como gestores de artefactos [6]. Su propósito principal es proporcionar un repositorio centralizado para almacenar, versionar y gestionar los artefactos binarios generados durante el proceso de construcción de software, como archivos JAR, WAR, imágenes de Docker y otros paquetes. Utilizar un gestor de artefactos ofrece varios beneficios, incluyendo una mejor gestión de las dependencias, la capacidad de rastrear el linaje de los artefactos y la facilitación del despliegue de aplicaciones en diferentes entornos. Si bien tanto Artifactory como Nexus cumplen una función similar, existen algunas diferencias en sus características, licencias y ecosistemas de plugins. Ambos son herramientas importantes en el panorama de DevOps, especialmente en entornos donde se practica la integración y entrega continua.

Conclusión

El Módulo 1 de la certificación LPIC DevOps Tools, Ingeniería de Software, abarca una amplia gama de conceptos y prácticas fundamentales para el desarrollo de software moderno en el contexto de DevOps. Desde el diseño de aplicaciones basadas en servicios y la comprensión de los estándares de API hasta el manejo del almacenamiento de datos y sesiones, el diseño para contenedores y la nube, y las consideraciones sobre software monolítico heredado y seguridad, este módulo sienta las bases para un ingeniero de DevOps competente. Además, la gestión del código fuente con Git y la implementación de pipelines de integración y entrega continua con herramientas como Jenkins son habilidades cruciales que se detallan en este módulo. El conocimiento de estos objetivos es esencial para cualquier profesional que aspire a obtener la certificación LPIC DevOps Tools Engineer y para aquellos que buscan mejorar sus habilidades en el campo de DevOps. Los siguientes módulos de la certificación profundizarán en otras áreas importantes como la gestión de contenedores, el despliegue de máquinas, la gestión de la configuración y las operaciones de servicio, construyendo sobre los conocimientos adquiridos en este primer módulo.

Obras citadas

1. Linux Professional Institute DevOps Tools Engineer, fecha de acceso: marzo 17, 2025, https://www.lpi.org/our-certifications/devops-overview/
2. Master DevOps Tools with LPI DevOps Tools Engineer Certification Training | edForce Live Online (VILT) & Classroom Corporate Training Course, fecha de acceso: marzo 17, 2025, https://www.edforce.co/lpi-devops-tools-engineer-certification-training/
3. Linux Professional Institute DevOps Tools Engineer - Skillzcafe, fecha de acceso: marzo 17, 2025, https://www.skillzcafe.com/training/linux-professional-institute-devops-tools-engineer
4. Linux Professional Institute LPIC-OT 701-100 DevOps Tools Engineer - IT-Training.pro, fecha de acceso: marzo 17, 2025, https://www.it-training.bg/courses/linux-professional-institute-lpic-ot-700-100-devops-tools-engineer/lessons/module-1-part-1/
5. How to prepare for Linux DevOps Tools Engineer 701-100 Exam? - Blog, fecha de acceso: marzo 17, 2025,

https://www.testpreptraining.com/blog/how-to-prepare-for-linux-devops-tools-engineer-701-100-exam/
6. DevOps Tools Engineer Exam 701 Objectives - Linux Professional Institute (LPI), fecha de acceso: marzo 17, 2025, https://www.lpi.org/our-certifications/exam-701-objectives/
7. LPIC-OT 701-100: DevOps Tools Engineer - LPI Central, fecha de acceso: marzo 17, 2025, https://lpicentral.blogspot.com/p/lpic-ot-701-100-devops-tools-engineer.html
8. 29 Most Reliable DevOps As A Service Tools In 2024 | Zeet.co, fecha de acceso: marzo 17, 2025, https://zeet.co/blog/devops-as-a-service
9. Azure DevOps Services, fecha de acceso: marzo 17, 2025, https://azure.microsoft.com/en-us/products/devops
10. Overview of services - Azure DevOps - Microsoft Learn, fecha de acceso: marzo 17, 2025, https://learn.microsoft.com/en-us/azure/devops/user-guide/services?view=azure-devops
11. What is DevOps as a Service (SaaS)? Why You MUST Have It | Chef - Glossary, fecha de acceso: marzo 17, 2025, https://www.chef.io/glossary/what-is-devops-as-a-service-saas
12. DevOps-as-a-Service defined | 3 key takeaways | Sumo Logic, fecha de acceso: marzo 17, 2025, https://www.sumologic.com/glossary/devops-as-a-service/
13. API Everything: A Developer's Guide to API Design, Security, and Performance - Medium, fecha de acceso: marzo 17, 2025, https://medium.com/@ksaquib/api-everything-a-developers-guide-to-api-design-security-and-performance-6725408b5997
14. Comprehensive Guide to API Development: Building Modern APIs - Ambassador Labs, fecha de acceso: marzo 17, 2025, https://www.getambassador.io/blog/api-development-comprehensive-guide
15. Writing API Design Standards - Medium, fecha de acceso: marzo 17, 2025, https://medium.com/api-center/writing-api-design-standards-84cb7cbb3fd7
16. What is an API? - Application Programming Interface Explained - AWS, fecha de acceso: marzo 17, 2025, https://aws.amazon.com/what-is/api/
17. What is API: Definition, Types, Specifications, Documentation - AltexSoft, fecha de acceso: marzo 17, 2025, https://www.altexsoft.com/blog/what-is-api-definition-types-specifications-documentation/
18. Session Management in Microservices - GeeksforGeeks, fecha de acceso: marzo 17, 2025, https://www.geeksforgeeks.org/session-management-in-microservices/
19. How Authentication Works in Microservices - Kong Inc., fecha de acceso: marzo 17, 2025, https://konghq.com/blog/learning-center/microservices-security-and-session-management
20. How to address decentralized data management in microservices - Cerbos, fecha de acceso: marzo 17, 2025, https://www.cerbos.dev/blog/data-management-in-microservices
21. Microservice Architecture Session Management Essentials - Springfuse.com, fecha de acceso: marzo 17, 2025, https://www.springfuse.com/microservice-architecture-session-management-essentials/
22. Microservices session data when communicating internally - Software Engineering Stack Exchange, fecha de acceso: marzo 17, 2025, https://softwareengineering.stackexchange.com/questions/356700/microservices-session-data-when-communicating-internally
23. What Is Containerization in DevOps? - Built In, fecha de acceso: marzo 17, 2025, https://builtin.com/software-engineering-perspectives/containerization
24. What Are Containers and Containerization in DevOps? - Papertrail, fecha de acceso: marzo 17, 2025, https://www.papertrail.com/solution/tips/what-are-containers-and-containerization-in-devops/

25. What is containerization? - GitHub, fecha de acceso: marzo 17, 2025, https://github.com/resources/articles/devops/containerization
26. What is Containerization in DevOps? | by May Sanders - Medium, fecha de acceso: marzo 17, 2025, https://medium.com/@may_sanders/what-is-containerization-in-devops-6e599ff66080
27. What Are Containers and Containerization in DevOps? - Code Institute Global, fecha de acceso: marzo 17, 2025, https://codeinstitute.net/global/blog/what-are-containers-and-containerization-in-devops/
28. 9 Most Reliable Deployment Strategies In DevOps - Zeet.co, fecha de acceso: marzo 17, 2025, https://zeet.co/blog/deployment-strategies-in-devops
29. DevOps Best Practices | Atlassian, fecha de acceso: marzo 17, 2025, https://www.atlassian.com/devops/what-is-devops/devops-best-practices
30. Deployment Strategies 101: Types, Pros, Cons, DevOps & More - Nearshore, fecha de acceso: marzo 17, 2025, https://www.nearshore-it.eu/articles/deployment-strategies-101-types-pros-cons-devops-more/
31. Deployment strategies - Introduction to DevOps on AWS, fecha de acceso: marzo 17, 2025, https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/deployment-strategies.html
32. Use a deployment strategy - Google Cloud, fecha de acceso: marzo 17, 2025, https://cloud.google.com/deploy/docs/deployment-strategies
33. The Role of DevOps in Modern Platform Engineering - Divami, fecha de acceso: marzo 17, 2025, https://divami.com/news/the-role-of-devops-in-modern-platform-engineering/
34. 16 DevOps Principles for Software Development and Best Practices - CyberPanel, fecha de acceso: marzo 17, 2025, https://cyberpanel.net/blog/devops-principles
35. What is DevOps? - Atlassian, fecha de acceso: marzo 17, 2025, https://www.atlassian.com/devops
36. 5 Key Principles of Agile Methodology and How to Apply Them - RevStar Consulting, fecha de acceso: marzo 17, 2025, https://revstarconsulting.com/blog/5-key-principles-of-agile-methodology-and-how-to-apply-them
37. Agile software development - Wikipedia, fecha de acceso: marzo 17, 2025, https://en.wikipedia.org/wiki/Agile_software_development
38. Agile Manifesto Values and Principles - Scrum Alliance Resources, fecha de acceso: marzo 17, 2025, https://resources.scrumalliance.org/Article/key-values-principles-agile-manifesto
39. Agile Software Process and its Principles - GeeksforGeeks, fecha de acceso: marzo 17, 2025, https://www.geeksforgeeks.org/agile-software-process-and-its-principles/
40. The Ultimate Guide to Agile Software Development - project-management.com, fecha de acceso: marzo 17, 2025, https://project-management.com/agile-software-development-methodologies/
41. What is DevOps? - DevOps Models Explained - Amazon Web Services (AWS), fecha de acceso: marzo 17, 2025, https://aws.amazon.com/devops/what-is-devops/
42. The Importance of DevOps in Modern Software Development - Successive tech, fecha de acceso: marzo 17, 2025, https://successive.tech/blog/devops-in-modern-software-development/
43. The Strategic Role of DevOps in Modern Software Development Solutions - Nallas, fecha de acceso: marzo 17, 2025, https://nallas.com/the-strategic-role-of-devops-in-modern-software-development-solutions/
44. Key DevOps Principles and Practices: Simplify Your SDLC - Moon Technolabs, fecha de acceso: marzo 17, 2025, https://www.moontechnolabs.com/blog/devops-principles/
45. DevOps Principles | Atlassian, fecha de acceso: marzo 17, 2025, https://www.atlassian.com/devops/what-is-devops