DOu Certified Tester in DevOps Foundation Level (Dou CTD-FL) Plan de estudios

versión 1.2 2020

DevOps United

Aviso de derechos de autor

Gracias a los autores principales:

- Yaron Tsubery, Vipul Kocher, Umang Agarwal, Sreevatsa Sreerangaraju,

Gracias al comité de revisión

Abdón Sandoval, Aldo Zarza, Ángel Rayo Acevedo, Anshu Khandelwal, Arjan Brands, Aurelio Gandarillas Cordero, Bart Knaack, Beatriz López OPtello, Christine Green, Claudia Trujillo, Claudio Caravajal Z., Cristián May Jara Segura, Cristian Valdebenito Esponosa, Emilie Potin-Suau, Geoffrey Wemans, Girts Baltaisbrencis, Guino Henostroza, Gustavo Marquez Sosa, Héctor Ruvalcaba, Héctor Santa María Bravo, Javier Alejandro Chávez Crivelli, Javier Rojas Cuturrufo, Jean-Luc Cossi, José Antonio Rodriguez, Juan Pablo Rios Alvarez, Julie Gardiner, Julio Córdoba Retana, Karolina Zmitrowicz, Kimmo

Hakala, Kyle Alexander Siemens, Laksh Ranganathan, Mani Ananth, Manuel Fischer, Matthias Rasking, Maximiliano Mannise, Michel Dussouchaud, Miguel Angel De León Trejo, Mirco Hering, Miroslav Renda, Norhayati Suzari, Orane Findley, Paul Mowat, Petr Neugebauer, Radhika Kundur, Ralf Pichler, Richard Seidl, Rik Marselis, Roberto Carlos Galicia Galicia, Samuel Ouko, Sebastian Małyska, Sergio Emanuel Cusmai, Sergio von OPrries, Shantel Stewart, Silvia Nane, Sittichai Udomchokpiti, Søren Wassard, Thomas Cagley, Tim Moore, Valeria Cocco, Vanessa Islas Padilla, Vikas Thakur, Wim Decoutere & Wouter Ruigrok.

Historial de revisión

versión	Fecha	Comentarios
version 1.1 2020 Spanish	5 de mayo, 2020	Primer lanzamiento para revisión

Índice

Propósito del presente documento

Este programa de estudios es la base de la certificación de DevOps United Certified Tester in DevOps (CTD). Este documento define lo que necesita cubrir para pasar el examen de certificación DOu Certified Tester in DevOps (CTD) y es propiedad de DevOps United. El examen de certificación sólo cubrirá los conceptos y conocimientos que se describen en este documento, aunque este documento contiene elementos prácticos que no se cubrirán en el examen de certificación pero que es necesario cubrir durante la capacitación.

Recursos de DOu

En www.devops-united.com, el sitio web oficial de DevOps United, se puede consultar un panorama general de los recursos de DOu, así como toda la información pertinente sobre la certificación de DOu y otros tipos de certificaciones de DOu. La información que figura en www.devops-united.com incluye:

- Una lista completa de proveedores de capacitación reconocidos de DOu y de los cursos disponibles. Tenga en cuenta que la capacitación se recomienda, pero no se requiere para tomar el examen de certificación DOu CTD.
- DOu Programa de estudios (este documento) para descargar.
- Un ejemplo de un examen de 10 preguntas CTD DOu con sus respuestas, con fines de preparación.
- Nuestro objetivo es que los documentos estén disponibles en otros idiomas lo antes posible. Para ver las versiones de los idiomas actualmente disponibles, por favor consulte www.devops-united.com.

Acerca de DOu Certified Tester in DevOps (CTD)

DOu Certified Tester in DevOps es un curso de nivel de especialista en fundamentos para los testers involucrados en DevOps. Nuestro objetivo en DevOps United es apoyarle a dar un paso más en el mundo ágil, en la metodología y cultura de trabajo basada en el código de DevOps. Aprenderá a utilizar nuevas herramientas y prácticas para reducir la distancia tradicional entre la programación y los técnicos de sistemas, con el fin de construir, probar y lanzar software de forma más rápida y fiable. Este nuevo enfoque de colaboración, DevOps, permitirá a sus equipos trabajar más estrechamente, aportando una mayor agilidad a su negocio y notables incrementos en su productividad, permitiéndole resolver rápidamente los problemas críticos y gestionar mejor el trabajo no planificado. Esta es una certificación enfocada a la

práctica, aplicaciones prácticas, uso de herramientas, así como demostraciones, grupos y/o ejercicios individuales y algo de teoría.

Resultados de negocio (BOs)

(BOs por sus siglas en inglés “Business Objectives”)

BO-1	Comprender los impulsores de los negocios y la tecnología, así como las metodologías y prácticas utilizadas por DevOps y “Continuous Movement” para crear una estrategia de prueba.
BO-2	Identificar y comprender la influencia de las pruebas en el movimiento DevOps, cómo se implementan las pruebas en DevOps y cómo DevOps encaja en varios ciclos de desarrollo de sistemas (por sus siglas en inglés “SDLCs”).
BO-3	Aplicar despliegues automatizados y pruebas continúas automatizadas en flujos de trabajo de integración continua y de entrega continua.
BO-4	Aplicar los tipos y niveles de prueba específicos de los ciclos de DevOps.
BO-5	Aplicar mediante ejercicios prácticos y la práctica interactiva el uso de la gestión de configuración, prueba continua, integración continua, despliegue continuo, entrega y monitoreo, implementación de herramientas comunes de DevOps como Dockers, Jenkins, Puppet-Chef/Ansible, Nagios, Cucumber, Selenium, Git/GitHub.
BO-6	Entender sobre la tecnología de la nube y cómo es útil en DevOps
BO-7	Comprender la dirección y las tendencias relacionadas con el futuro de las pruebas continuas

Objetivos de aprendizaje/niveles cognitivos de conocimiento

Los objetivos de aprendizaje (Learning Objectives, LOs) son declaraciones breves que describen lo que se espera que sepas después de estudiar cada capítulo. Los LOs se definen en base a la taxonomía modificada de Bloom de la siguiente manera:

- K1: Recordar. Algunos de los verbos de acción son: recordar, elegir, definir, encontrar, vincular, relacionar, seleccionar.
- K2: Entender. Algunos de los verbos de acción son: resumir, generalizar, clasificar, comparar, contrastar, demostrar, interpretar, reformular.

- K3: Aplicar. Algunos de los verbos de acción son: implementar, ejecutar, usar, aplicar.

Para más detalles de la taxonomía de Bloom, por favor consulte [BT1] y [BT2] en Referencias.

Objetivos prácticos

Los objetivos prácticos (Hands-on Objectives, HOs) son declaraciones breves que describen lo que se espera que realice o ejecute para comprender el aspecto práctico del aprendizaje.

Los HOs se definen de la siguiente manera:

- HO-0: Demostración en vivo de un ejercicio o video grabado.
- HO-1: Ejercicio guiado. Los alumnos siguen la secuencia de pasos que realiza el capacitador.
- HO-2: Ejercicio con pistas. Ejercicio para resolver por el aprendiz. utilizando pistas proporcionadas por el capacitador.
- HO-3: Ejercicios no guiados, sin pistas.

Requisitos previos generales

Obligatorio

- Ninguno

Recomendado

- ISTQB® Foundation level o equivalente
- Conocimientos básicos de cualquier lenguaje de programación- Java/Python/R
- Conocimientos básicos de estadística
- Alguna experiencia en desarrollo o prueba de software.

Requisitos previos en lenguaje de programación

Requeridos

- Conocimientos básicos de programación. Comprensión de variables, funciones, métodos, estructuras de control (condicionales y ciclos), gestión de la memoria.
- Conocimiento básico de los lenguajes de programación.
- Conocimiento básico de herramientas operativas como Selenium, Java, JUnit, así como de herramientas básicas de DevOps.

- Conocimiento básico del protocolo HTTP. Comprensión de la solicitud/respuesta HTTP, y los principales elementos involucrados, como cookies, URL, parámetros, métodos (GET, POST), encabezados y cuerpo.
- Conocimientos básicos de la arquitectura del sistema. Comprensión de las arquitecturas Web basadas en capas (Cliente/Servidor).

Recomendado

- Conocimientos básicos de SOAP/REST y XML/JSON y/o servicios web o microservicios.

Herramientas específicas mencionadas en el programa de estudios [Renuncia de responsabilidad]

Las herramientas mencionadas en el plan de estudios se utilizan únicamente como ejemplos. Estas herramientas representan algunas de las más utilizadas en el momento de la publicación de este programa de estudios.
El enfoque de este plan de estudios está en los conceptos. Por lo tanto, las herramientas específicas se utilizan sólo como medio para demostrar esos conceptos, o para realizar ejercicios prácticos. El plan de estudios no tiene como objetivo promover ninguna herramienta por encima de las demás ni apoyar a ninguna empresa que produzca herramientas por encima de las demás. Durante el entrenamiento o de otra manera, si hay disponibles otras alternativas adecuadas, cualquiera es bienvenido a usarlas también.

Capítulo 1 - Introducción a DevOps

Palabras clave: DevOps, DevSecOps, DevTestOps, DevArchOps, DevWinOps, Pipelines, Integración continua (CI por sus siglas en ingles “Continuous Integration”), Despliegue continuo (Continuous Deployment), Entrega continua (CD por sus siglas en inglés “Continuous Delivery”), CI/CD, Administrador de configuración del centro del sistema (SCCM por sus siglas en inglés “System Center Configuration Manager), Shift Left, Repositorio de software, Repositorio de código, Estrategia de ramificación (branching), Fusión (Merging), Desarrollo guiado por pruebas (TDD por sus siglas en inglés “Test Driven Development”), Desarrollo guiado por comportamiento (BDD por sus siglas en inglés “Behaviour Driven Development”), Desarrollo guiado por pruebas de aceptación (ATDD por sus siglas en inglés “Acceptance Testing Driven Development”), Especificación mediante ejemplos (SBE por sus siglas en inglés “Specification by Example”).

LO #	Descripción
LO-1.1.1	Recordar el propósito de DevOps (K1)
LO-1.2.1	Explicar los componentes de DevOps (K2)
LO-1.2.2	Recordar los principios básicos de DevOps (K1)
LO-1.3.1	Comparar el proceso de liberación tradicional con el pipeline de entrega (K2)
LO-1.3.2	Explicar el concepto de integración continua (CI), y las ventajas que ofrece (K2)
LO-1.3.3	Explicar los conceptos del SCCM: Repositorios, check-in y check-out, versionado, ramas, fusión, resolución de conflictos, trabajo en equipo, estrategias de ramificación (K2)
LO-1.3.4	Explicar el pipeline de CI y cómo las herramientas ayudan a configurar un pipeline de CI (K2)
LO-1.4.1	Explicar la entrega continua (CD) y las ventajas que ofrece (K2)
LO-1.4.2	Recordar las herramientas utilizadas en la entrega continua (K1)
LO-1.5.1	Recordar el propósito del despliegue continuo (K1)

LO #	Descripción
LO-1.5.2	Comparar el despliegue continuo con la entrega continua (K2)
LO-1.7.1	Recordar los principales aspectos culturales y la mentalidad implementada en DevOps (K1)
LO-1.7.2	Recordar las principales razones por las que el principio de Shift Left contribuye con DevOps (K1)
LO-1.7.3	Recordar varios términos relacionados con DevOps, como DevSecOps, DevTestOps, etc. (K1)
LO-1.7.4	Entender cómo encajan DevOps y Agile (K2)

HO#	Descripción
HO-1.2.3	Demostrar los desafíos de DevOps (HO-0)
HO-1.3.3	Demostrar cómo aplicar las principales características de una herramienta de gestión de la configuración: check-in, check-out, fusión, resolución de conflictos, ramificación (HO-0)
HO-1.3.4	Crear un pipeline simple para la compilación de código basado en el desencadenamiento del check-in de código (HO-1)
HO-1.5.2	Demostrar cómo aplicar las principales características de las herramientas de entrega continua y despliegue continuo (HO-0)
HO-1.6.0	Demostrar los principales elementos de monitoreo en herramientas como Nagios o Grafana (HO-0)

1.1 Un resumen de DevOps

DevOps es una cultura y práctica de ingeniería de software que tiene como objetivo unificar el desarrollo de software (Dev - por su acrónimo en inglés “Development”) y la operación de software (Ops — por su acrónimo en inglés “Operations”). DevOps es un conjunto de prácticas que combinan el desarrollo de software y las operaciones de TI para acortar el ciclo de vida de los productos o el desarrollo de sistemas.

1.1.1 Historia y propósito de DevOps

LO-1.1.1	Recordar el propósito de DevOps (K1)

DevOps se inició después de que los conceptos de entregas rápidas, múltiples y exitosas fueran presentados en la conferencia Velocity de 2009. El nombre DevOps fue acuñado para referirse al concepto de organizar/crear un ciclo infinito entre las estructuras de desarrollo y operación, con el fin de permitir ciclos de desarrollo y entrega de productos fluidos e incrementales.
En 2009 se celebró en Gante (Bélgica) la primera conferencia sobre el tema, "DevOpsDays". La conferencia fue fundada por el consultor, gestor de proyectos y profesional ágil belga, Patrick Debois.

La principal característica del movimiento DevOps es abogar firmemente por la automatización y el monitoreo en todas las etapas de la construcción (build) del software, desde la integración, las pruebas y la liberación hasta el despliegue y la gestión de la infraestructura. El objetivo de DevOps es acortar los ciclos de desarrollo, aumentar la frecuencia de despliegue y hacer que las liberaciones sean más fiables, en estrecha relación con los objetivos de negocio y los clientes.

1.2 Conceptos de DevOps

El concepto de DevOps se basa en la creación de una cultura de colaboración entre equipos que históricamente funcionaron en silos relativos.

1.2.1 Componentes de DevOps

LO-1.2.1	Explicar los componentes de DevOps (K2)

Los componentes clave de DevOps son:

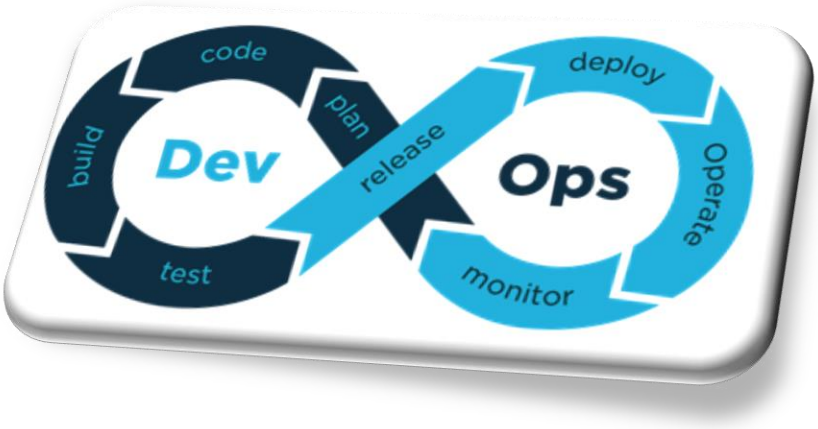
- Entrega automática del pipeline
- Gestión de la configuración del sistema/solución
- Integración continua regular
- Monitoreo automatizado y controles eficientes
- Entrega regular continua
- Infraestructura como código

1.2.2 Principios básicos de DevOps

LO-1.2.2	Recordar los principios básicos de DevOps (K1)

Los principios básicos de DevOps son:

- Acción centrada en el cliente.
- Crear con el objetivo final en mente.
- Responsabilidad de extremo a extremo.
- Equipos autónomos multifuncionales.
- Retroalimentación continua.
- Integración continua y despliegue continuo.
- Mejora continua.
- Experimentación y aprendizaje continuos.
- Automatizar todo lo que puedas.



1.2.3 Desafíos de DevOps

HO-1.2.3	Demostrar los desafíos de DevOps (HO-0)

Los retos que DevOps pretende resolver son muchos, tales como:

- Pruebas manuales
- Datos de prueba
- Falta de virtualización de servicios
- Falta de un sistema de gestión de configuración
- Falta de una arquitectura de herramientas integradas

- Planificación en un entorno de DevOps
- Entornos inconsistentes
- Falta de entornos de producción
- Limitada retroalimentación de los clientes
- Problemas relativos a la colaboración entre desarrollo y operaciones
- Problemas relativos a la colaboración entre todos los departamentos relacionados con el producto
- Elección de requerimientos no funcionales
- Despliegues manuales.
- Liberaciones manuales.
- Liberaciones grandes
- Falta de métricas de DevOps
- Falta de rastreabilidad a través del escenario de DevOps

1.3 Integración continua

1.3.1 Proceso de liberación tradicional vs pipeline de entrega

LO-1.3.1	Comparar el proceso de liberación tradicional contra el pipeline de entrega (K2)

La gestión tradicional de la liberación incluye, por supuesto, la planificación, la programación, la supervisión y el control de un software construido en varias etapas y en diferentes entornos. Sin embargo, puede llegar en muchos gustos diferentes. La gestión de liberaciones garantiza que los despliegues de producción estén bien orquestados y sigan todos los pasos necesarios para asegurar la visibilidad adecuada y que se obtengan las aprobaciones a lo largo del proceso.
El proceso DevOps, permite a los equipos tener, o mejor dicho, tomar el control de los despliegues de producción. El equipo está, por supuesto, creando código de trabajo, pero también se centra en la infraestructura, la red y otros elementos de implementación necesarios para poner su código en producción. Estos equipos se encargan de crear código de mayor calidad, ya que asumen la responsabilidad general de que los sistemas de producción sean más fiables y mantenibles.
En DevOps, los equipos utilizan un pipeline de datos. Se trata de un conjunto de elementos de procesamiento de datos conectados en serie, donde la salida de un elemento es la entrada del siguiente. Los elementos de un pipeline se ejecutan a menudo en paralelo o de forma escalonada. Esto permite al equipo gestionarlo

mejor, utilizar su línea de tiempo y proporcionar ciclos de entrega más optimizados y controlados.

1.3.2 Definición y principios de la integración continua (CI)

LO-1.3.2	Explicar el concepto de integración continua (CI por sus siglas en inglés “Continuous integration”) y las ventajas que ofrece (K2)

El movimiento “continuo” consiste en los dos enfoques principales, que se fusionaron (merge) en lo que hoy llamamos CI/CD. La CI es principalmente la práctica de fusionar todo el código implementado por el o los desarrolladores y listo para ser fusionado (el código de trabajo en curso) a una rama de código centralizada (maestra), normalmente varias veces al día.

CI significa integración continua (Continuous Integration) y CD significa entrega continua (Continuous Delivery), así como el despliegue continuo (Continuous Deployment), el cual sigue inmediatamente.

La CI incluye procesos/mecanismos adicionales para realizar entregas más rápidas y fiables, además del proceso y las actividades tradicionales de liberación.

Las principales ventajas del pipeline CI al utilizar piezas de entrega más pequeñas son:

- Ayudar a lograr un impulso continuo
- Ayudar a detectar los defectos de forma temprana
- Facilitar el aislamiento de los defectos
- Ayudar a reducir los costes mediante la automatización

CI/CD son los principales anillos de habilitación en la cadena de DevOps y los principales contribuyentes para lograr los objetivos de DevOps. El largo proceso tradicional fue dividido por el enfoque de CI/CD piezas de entrega más pequeñas y controladas, para mantener el impulso continuo., para mantener el impulso continuo.

La característica principal del despliegue continuo es permitir que cualquier nueva pieza de código que supere las pruebas automatizadas relacionadas se despliegue.

El movimiento ágil ha sentado las bases para el DevOps, que impulsa aún más a el concepto ágil, permitiendo un desarrollo más rápido, mientras que

enriquece la metodología ágil con muchas herramientas que colaboran y se armonizan para lograr una mejor solución continua.

1.3.3 Gestión de la configuración del código fuente

LO-1.3.3	Explicar los conceptos del SCCM: Repositorios, check-in y check-out, versionado, ramas, fusión, resolución de conflictos, trabajo en equipo, estrategias de ramificación (K2)

HO-1.3.3	Demostrar cómo aplicar las principales características de una herramienta de gestión de la configuración: check-in, check-out, fusión, resolución de conflictos, ramificación (HO-0)

Uno de los factores más importantes y clave en DevOps es contar con una base sólida de gestión del código fuente y de la configuración (configuration management, CM).

El concepto de SCCM (Administrador de configuración del centro del sistema, del inglés System Center Configuration Manager) o la idea de un sistema central que administra la gestión de la configuración del código, desde su creación hasta la entrega, la producción, el mantenimiento, etc. es tener todos los pasos necesarios gestionados y controlados en un solo lugar. La suite SCCM consta de varias partes, entre las que se encuentran los siguientes módulos y características clave:

Repositorios: El código que escriben los desarrolladores es parte de un importante conjunto de información. Los repositorios guardan y almacenan esta información para su uso posterior, para restauración, para aspectos retrospectivos, etc. Los repositorios representan estos almacenamientos y poseedores de código.

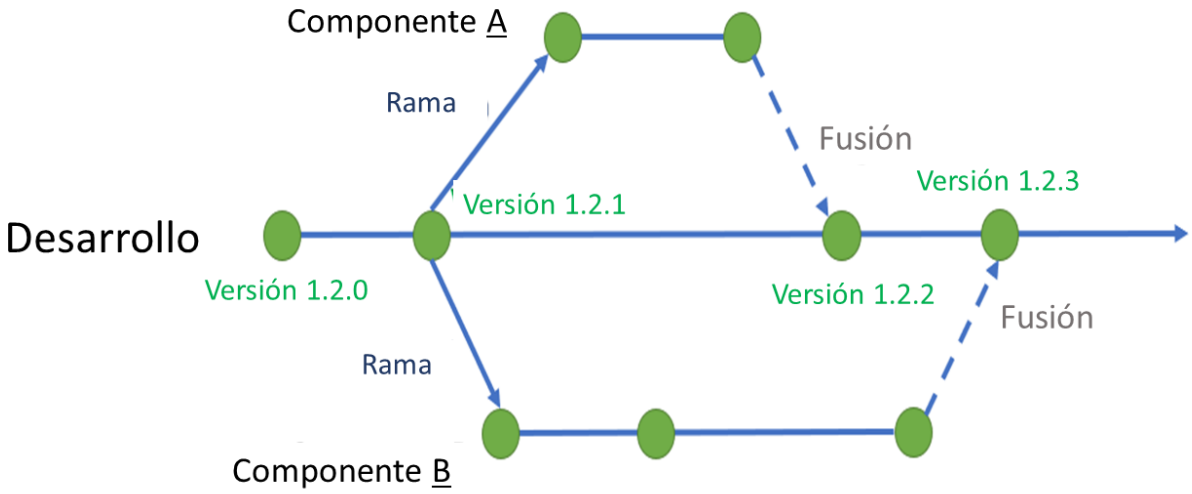
Check-in y check-out: Un programador escribe un nuevo código o mantiene uno existente. La acción de reunir el código del repositorio para trabajar en él se llama check-out" el código. En cualquier momento en que el programador decide que el trabajo se ha completado y que es necesario almacenarlo de nuevo como un pedazo de código nuevo o actualizado en el repositorio de la rama principal para que el código pueda ser revisado, el programador realiza una acción llamada check-in el código.

Versionado: El versionado se utiliza para etiquetar (dar un nombre o un número o la combinación de nombre y número) un trozo de código que se fabrica como

un programa o un conjunto de programas - por ejemplo, un paquete. Esta etiqueta representa el nombre de este artículo o paquete específico. La versión es una referencia específica que permite a los distintos interesados en los productos o sistemas comunicarse entre sí - y les permite asociar el programa/producto/sistema con el código que se almacena y tiene la misma etiqueta. Una versión única del mismo elemento (archivo, documento, biblioteca, etc.) permite disponer de este elemento en varias versiones al mismo tiempo y ver su evolución.

Ramas y estrategias de ramificación: Una rama es una copia del código fuente que permite a los programadores desarrollar dos versiones por separado. La ramificación es una técnica utilizada para hacer una copia del código fuente con el fin de crear dos versiones del mismo, que se desarrollan por separado. Existen varias formas de ramificación, que pueden utilizarse para proporcionar un paquete de servicios al cliente o una nueva generación del producto. Este tipo de situaciones requiere que el equipo de DevOps haga una elección, en cuanto a la estrategia de ramificación a implementar.

Fusión (Merging): Muchas ramas podrían ser alteradas por los programadores de la misma rama principal del código en algunos casos. Por ejemplo, la rama principal es la versión 1.2.x, pero hay pocos equipos de desarrollo que trabajan con esta versión/rama como base para la actualización de su código, por lo que se trata de nuevas ramas bajo la principal que es la versión/rama 1.2.x. Esta situación a lo largo del tiempo crea complejidad, ya que el código puede variar entre las nuevas ramas que se están creando en paralelo o entre sí. Para poder volver a un solo lugar, es necesario reunirlas todas juntas en la rama principal. Esta actividad de reensamblar las ramas en la principal se denomina fusión.



Resolución de conflictos: Si se produce un fallo durante el proceso de fusión, indica un conflicto entre la rama local actual y la rama que se está fusionando, lo que significa que existe un conflicto entre el código que deseamos fusionar y el código de otro desarrollador. Las herramientas disponibles hoy en día hacen todo lo posible por fusionar los archivos de código, pero dejan que los conflictos que surjan en el proceso de fusión sean resueltos manualmente por la persona que lo operó.

Trabajando en equipo: Los programas o sistemas suelen ser desarrollados por más de un desarrollador, normalmente trabajando en equipo - en Ágil, podemos encontrarlo por ejemplo en el equipo scrum. Esto suele ser el resultado de la necesidad de tener el producto o los sistemas listos más rápido o, cuando las diferentes partes del sistema requieren diferentes habilidades de programación, el departamento de desarrollo se divide en equipos. Diferentes desarrolladores, o incluso equipos pueden trabajar en el mismo código — por ejemplo, un equipo puede estar a cargo de las liberaciones de mantenimiento y el otro de las nuevas características, etc. Las herramientas o conjuntos de gestión del código fuente y de la configuración deben permitir la armonización de esta forma de trabajo paralela.

1.3.4 CI Pipeline y herramientas

LO-1.3.4	Crear un pipeline simple para la compilación de código basado en el desencadenamiento del check-in de código (K2)

HO-1.3.4	Crear un pipeline simple para la compilación de código basado en el desencadenamiento del check-in de código (HO-1)

El pipeline de CI (integración continua) es la principal columna vertebral en el entorno de DevOps. El CI es el primer paso en el CI/CD en DevOps, luego sigue el segundo paso que es el CD (Continuous Delivery). Juntos, reducen la brecha entre el lado del desarrollo y el lado de las operaciones. Un pipeline completo de CI consiste en tres partes principales:

- Integración del código
- Construcción del código
- Ejecución de las pruebas unitarias

Para apoyar la superación de esta brecha, DevOps ofrece un proceso que consta de varias etapas. Cada etapa se apoya en un conjunto de instrumentos que se complementan entre sí y que respaldan las tecnologías o instrumentos necesarios utilizados en la etapa anterior.

Con el fin de orquestar el funcionamiento de estas etapas, el equipo de DevOps utiliza herramientas denominadas planificadores (por ejemplo, Jenkins, GiTHub, etc.).

1.4 Entrega continua (CD, Continuous Delivery)

1.4.1 CD — Definición y pipeline

LO-1.4.1	Explicar el significado de entrega continua (CD) y las ventajas que ofrece (K2)

La entrega continua (CD) es parte del método DevOps. Consiste en etapas impulsadas por pipelines, que están diseñadas para empresas que tienen como objetivo mejorar las aplicaciones, así como los sistemas con frecuencia y que por lo tanto requieren un proceso de entrega fiable.

La principal ventaja del CD es que permite el despliegue de los cambios de código a la producción de forma automatizada. Para ello, se potencia la integración continua con pruebas automatizadas y el despliegue automatizado una vez superadas las pruebas. La integración continua se potencia con pruebas y despliegue automatizados.

Una ventaja adicional de este enfoque es que obliga a producir el software de tal manera que el software pueda ser liberado fácilmente/continuamente cuando sea necesario. Toda la actividad se realiza en una secuencia de pequeños ciclos repetitivos. Esta técnica de pequeños ciclos repetitivos permite que el código provisto consiga incorporar cualquier tipo de cambio (como la mejora del software, los cambios de configuración, la corrección de defectos, etc.) rápidamente, de forma eficiente y sostenible.

CD requiere la automatización de los procesos de entrega de aplicaciones y sistemas, con la intención de desplegar el código integrado en la producción sin errores críticos o retrasos. El proceso de implementación en las etapas de CD ayuda a los desarrolladores a fusionar el nuevo código que han creado o modificado con la rama principal de manera consistente, para que puedan construir un producto listo para su liberación inmediata.

1.4.2 Herramientas en CD

LO-1.4.2	Recordar las herramientas utilizadas en la entrega continua (K1)

El crecimiento de DevOps en el mercado y la necesidad de fomentar CI/CD requieren que el número de herramientas que apoyan las etapas de CI/CD aumente y que el nuevo código se despliegue instantáneamente en la producción. Estas herramientas son los componentes de un pipeline de entrega que constituye una entrega continua; están al servicio de las etapas de CI o de CD, o de ambas, y tienen diferentes tipos y combinaciones de características. Estas herramientas apoyan estas etapas dentro de la tecnología y/o el dominio en el que consisten las aplicaciones o sistemas, tales como aplicaciones basadas en la web, en Microsoft, en Linux, en la nube, etc.

Además de lo anterior, la mayoría están diseñadas para reducir el umbral de entrada a DevOps, aumentando la agilidad, acortando los plazos de liberaciones, mejorando la fiabilidad del software y manteniéndose por delante de la competencia.

Las herramientas de CD más conocidas en el mercado actual son:

Buddy: es una herramienta de CI/CD para desarrolladores web, diseñada para bajar el nivel de entrada a DevOps. Utiliza pipelines de entrega para construir,

probar y desplegar el software. Además, emplea contenedores Docker [ver más información sobre contenedores en sección 5.1.4 de este temario] con lenguajes y marcos de trabajo (framework) preinstalados para construir, junto con DevOps, acciones de monitoreo y notificación.

Buildbot: es una herramienta de integración continua de desarrollo de software que automatiza el ciclo de compilación o prueba necesario para validar los cambios en la base de código del proyecto. Es un marco de trabajo de código abierto desarrollado en Python.

Urbancode deploy: es un modelo de aplicación de varios niveles o un producto de IBM. Proporciona una entrega continua, autoservicio, retroalimentación rápida y actualizaciones incrementales en el entorno ágil, y automatiza los despliegues de aplicaciones de manera consistente. Los desarrolladores también pueden hacer retroceder las aplicaciones (rollback), organizar los cambios en los servidores, niveles y componentes.

Codeship: es una poderosa herramienta que automatiza el flujo de trabajo de desarrollo y despliegue. Codeship desencadena este flujo de trabajo automatizado simplemente hacer push al repositorio. Las ejecuciones paralelas de las pruebas se completan con la característica **ParallelCI** de **CircleCI.**

Strider: es una plataforma de código abierto de CI/CD. Está escrita en Node.JS/JavaScript y utiliza MongoDB. Se publica bajo la licencia BSD. Soporta diferentes plugins que modifican el esquema de la base de datos y la interfaz de usuario y registran las rutas HTTP. Un marco de trabajo extensible desencadena construcciones y despliegues. Está integrado en muchos proyectos como GitHub, BitBucket, Gitlab, etc.

Solano Labs: es también una herramienta de CI/CD que funciona a la manera de “Software as a Service” (SaaS) de la informática en la nube. Usando Solano, el usuario puede usar muchos lenguajes y marcos de trabajo para escribir su código y para hacer pruebas. Puede ser integrado con otros proyectos, como Github.

Semaphore: soporta muchos lenguajes y marcos de trabajo y puede ser integrado con Github. Realiza pruebas y despliegues automatizados. Usando la colaboración, los usuarios pueden invitar a otros colaboradores que sean copiados de Github.

Wercker: es una herramienta que automatiza la construcción y despliega el contenedor. Crea un pipeline automatizado único (construye y despliega los pipelines) que se ejecutan a través de la interfaz de la línea de comandos. Proporciona los microservicios, lo que significa que desencadena los pipelines cada vez que se hace commit en un nuevo código. Wercker’s Docker Stack

realiza el procesamiento muy rápido y evita cualquier amenaza o error. Aísla las aplicaciones y servicios del sistema operativo.

1.5 Despliegue continuo

1.5.1 Despliegue continuo - definición

LO-1.5.1	Recordar el propósito del despliegue continuo (K1)

El despliegue continuo es una estrategia para la liberación de aplicaciones o sistemas de software en la que cualquier commit del código que pase la fase de pruebas automatizadas se libera automáticamente en el entorno de producción, haciendo cambios que son visibles para los usuarios del software.

1.5.2 Despliegue continuo vs Entrega continua

LO-1.5.2	Comparar el despliegue continuo con la entrega continua (K2)

HO-1.5.2	Demostrar cómo aplicar las principales características de las herramientas de entrega continua y despliegue continuo (HO-0)

Despliegue continuo significa estar listo y ser capaz de desplegar continuamente, mientras que entrega continua es un modo de estar listo y ser capaz de liberar cualquier versión en cualquier momento en cualquier plataforma.

Ambos existen en el proceso ágil que proporciona un marco de trabajo en el que se realizan pequeños y frecuentes cambios y donde la retroalimentación se obtiene rápidamente.

En algunas organizaciones, las prácticas de DevOps como CI/CD se implementan de manera que el CD se refiere tanto a la entrega continua como al despliegue continuo.

1.6 Monitoreo continuo

HO-1.6.0	HO-1.6 Demostrar los principales elementos de monitoreo en herramientas como Nagios o Grafana (HO-0)

El monitoreo continuo en DevOps se refiere tanto al proceso como a la tecnología que se requiere para incluir el monitoreo a través de cada fase de los ciclos de vida de las operaciones de DevOps y TI de la compañía. Este método ayuda a asegurar continuamente la integridad, el rendimiento y la fiabilidad de las aplicaciones y/o sistemas y su infraestructura a medida que se pasa del desarrollo a la producción.

La utilización de herramientas de mejores prácticas permite y simplifica un monitoreo continuo fiable. Estas herramientas de prácticas óptimas son las que tienen mayor relevancia en el momento en que el usuario considera utilizarlas (en el momento de publicar este documento, herramientas como Nagios, Grafana y Raygun).

1.7 DevOps en diversas prácticas de desarrollo

1.7.1 Cultura DevOps

LO-1.7.1	Recordar los principales aspectos culturales y la mentalidad implementada en DevOps (K1)

La mayoría de las empresas están organizadas en tres equipos distintos: desarrollo, pruebas y operaciones. Cada uno tiene sus propios intereses, objetivos y metodologías. Esta separación a menudo resulta en falta de comunicación, retrasos y tensiones. La principal característica de la cultura de DevOps es que mejora la colaboración entre investigación y desarrollo (que incluyen el desarrollo y las pruebas) y las actividades operativas.

Los principales aspectos culturales y la mentalidad de DevOps son el aumento de la colaboración entre los departamentos, una menor tendencia a trabajar en silos, el trabajo ligero mientras se comparte la responsabilidad, el énfasis en la autonomía de los equipos, la mejora de la calidad, la aceptación del fracaso mientras se valora la retroalimentación y el aumento de la automatización. Muchos de los valores de DevOps son valores ágiles, ya que DevOps es una extensión de los valores ágiles.

1.7.2 DevOps y Shift Left

LO-1.7.2	Recordar las principales razones por las que el principio de Shift Left contribuye a DevOps (K1)

El término “Shift Left” se refiere a una práctica en el desarrollo de software en la que los equipos se centran en la calidad de sus productos, así como en la prevención de problemas en lugar de la detección, y comienzan a realizar pruebas lo antes posible. Un aspecto importante en el Shift Left es la prueba estática, en la que se requiere revisar todo tipo de entradas para los equipos de DevOps, tales como historias de usuarios, descripciones de interfaz y otros documentos de definición y diseño, antes de que se pueda crear cualquier código, a fin de evitar que se copien los fallos y se construya así un código erróneo. Además, añade más herramientas de análisis de código estático para ayudar a probar el código antes. El Shift Left también requiere las siguientes prácticas clave de DevOps: pruebas continuas y despliegue continuo. Hay varios métodos y enfoques en el desarrollo de software que apoyan esta etapa, como el desarrollo guiado por pruebas (TDD), el desarrollo guiado por pruebas de aceptación (ATDD), el desarrollo guiado por comportamiento (BDD), la especificación mediante ejemplos (SBE), etc. Esto significa que los desarrolladores se incorporan a los ciclos de pruebas lo antes posible para evitar problemas que, en el pasado, se detectaban más tarde y provocaban retrasos. Cuanto antes se realicen las pruebas, antes se detectarán los problemas. Los métodos de actividades continúas mencionados ayudan a reducir los costes que surgen en las fases tardías de las pruebas y ayudan a proporcionar una retroalimentación más rápida sobre un cambio.

Los principios clave para Shift Left son:

- Probar pronto
- Retroalimentar antes los casos de uso orientados al cliente
- Probar a menudo
- Probar de forma incremental
- Reducir los costes

1.7.3 DevSecOps, DevTestOps, DevDataOps, etc.

LO-1.7.3	Recordar varios términos relacionados con DevOps, como DevSecOps, DevTestOps, etc. (K1)

DevOps ha evolucionado a lo largo de los años y ha incorporado más etapas relacionadas con la ingeniería de requisitos y los procesos y se han acuñado a la manera de DevOps, como: DevSecOps (para la seguridad), DevTestOps (para las pruebas), DevDataOps (para los datos), DevArchOps (para la arquitectura) y DevWinOps (para MScentric).

1.7.4 DevOps y ágil

LO-1.7.4	Entender cómo funcionan DevOps y Agile juntos (K2)

“Ágil” es la capacidad de crear y responder al cambio. Es una forma de hacer frente a un entorno incierto y turbulento y, en última instancia, de tener éxito en él" (organización de alianza ágil).

En la agilidad, se pide a los equipos que se muevan más rápido – introduciendo mejoras y correcciones cada vez más rápidas –, así como que reduzcan el tiempo de entrega, al tiempo que siguen mejorando la calidad de cada liberación.

Los valores y la cultura de DevOps se basan en la agilidad, y cada etapa de DevOps define mejor la acción necesaria que los equipos deben llevar a cabo, así como el conjunto de herramientas que apoyan estas etapas y permiten a los equipos ágiles - entre ellos el equipo de DevOps - trabajar más rápido, de manera más decisiva y fiable. El enfoque de DevOps es integrar la agilidad que ágil propone en ambos, el proceso y la técnica.

Tanto ágil como DevOps comparten un objetivo común, que es mejorar la velocidad y la calidad de la entrega de valor. La diferencia es que el ágil apunta a optimizar el desarrollo de software específicamente pero no toma en cuenta las otras partes de la cadena de valor que vienen después de los ciclos de desarrollo - DevOps definitivamente lo hace.

Capítulo 2 — Pruebas continuas

Palabras clave: Pruebas continuas (CTI por sus siglas en ingles “Continuous Testing”), cuadrante de pruebas, análisis estático, cobertura de código, fuga de memoria, medición del rendimiento, desarrollo guiado por pruebas (TDD), desarrollo guiado por comportamiento (BDD), desarrollo guiado por pruebas de aceptación (ATDD), pipeline de CI, pipeline de DevOps, sistema bajo prueba (SUT).

LO-2.1.1	Recordar las principales características de las pruebas continuas (K1)
LO-2.1.2	Comprender las modificaciones del cuadrante de pruebas para DevOps (K2)
LO-2.2.1	Explicar el desarrollo guiado por por pruebas (TDD) y sus ventajas (K2)
LO-2.2.2	Aplicar un marco de trabajo xUnit para realizar la TDD (K3)
LO-2.3.1	Explicar las principales características de una herramienta de análisis estático (K2)
LO-2.4.1	Explicar las principales características de una herramienta de cobertura de código (K2)
LO-2.4.2	Explicar el concepto de detección de fugas de memoria (K2)
LO-2.4.3	Explicar las principales tareas de una herramienta de medición del rendimiento del código (K2)
LO-2.5.1	Integrar las pruebas de API en un pipeline de CI (K3)
LO-2.5.2	Integrar una herramienta de pruebas de automatización de GUI en un pipeline de CI (K3)
LO-2.6.1	Integrar una herramienta de desarrollo guiado por comportamiento en el pipeline de DevOps (K3)

HO#	Descripción
HO-2.2.2	Realizar un ejercicio guiado de creación de pruebas usando el marco de trabajo xUnit contra algún código dado (HO-1)

HO#	Descripción
HO-2.3.1	Demostrar cómo integrar una herramienta de análisis estático en un pipeline de DevOps (HO-0)
HO-2.4.1	Realizar un ejercicio que ofrezca una pista de integración de una herramienta de análisis estático en el pipeline de DevOps y comprobar la cobertura de pruebas de un código determinado y sus pruebas (HO-2)
HO-2.4.2	Demostrar cómo integrar una herramienta de detección de fugas de memoria en el pipeline de DevOps (HO-0)
HO-2.4.3	Demostrar cómo integrar una herramienta de medición del rendimiento del código en un pipeline de DevOps (HO-0)
HO-2.6.1	Demostrar cómo integrar una herramienta de desarrollo guiado por comportamiento en un pipeline de DevOps (HO-0)

2.1 Introducción a las pruebas continuas

2.1.1 Definición y características de las pruebas continuas

LO-2.1.1	Recordar las principales características de las pruebas continuas (K1)

Las pruebas continuas son el proceso de ejecución de pruebas automatizadas como parte del pipeline de entrega del software para obtener información inmediata sobre los riesgos de negocio asociados con una versión candidata (RC, Release Candidate) de la liberación de software. Los principales objetivos de las pruebas continuas son las pruebas tan pronto y tan a menudo como sea posible. La comprensión de la diversidad de pruebas que deben cubrirse, teniendo en cuenta los principales objetivos de las pruebas continuas, y la comprensión adecuada de cómo aplicarlas conjuntamente en el proyecto, ayudan a definir mejor la estrategia de pruebas pertinente para este proyecto. Las características fundamentales de las pruebas continuas requieren lo siguiente:

- Casos de prueba micro (atómicos) independientes

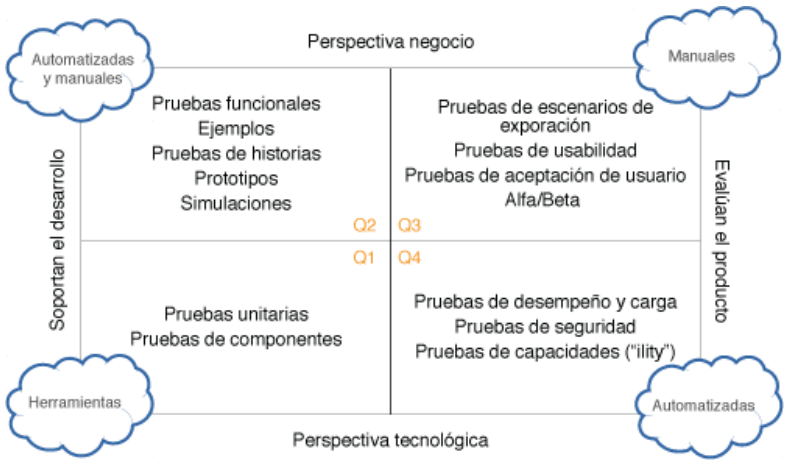
- Independencia entre las pruebas - para reducir las dependencias lo más posible
- Entorno de pruebas completamente automatizado - teniendo esa parte del Sistema Bajo Prueba (System Under Test, SUT por sus siglas en inglés) así como soluciones de pruebas
- Fuerte sistema de gestión de la configuración para apoyar la evolución de las pruebas y las soluciones de pruebas de automatización para estar en línea con el desarrollo del SUT
- Capacidad de desencadenar cualquier tipo de pruebas o tipos de pruebas o niveles de pruebas (diferentes niveles o mezcla de niveles) —que también se pueden encontrar en la pirámide de pruebas — contra diferentes entornos, diferentes liberaciones o versiones, etc.
- Comprensión de que las pruebas son una actividad necesaria y no sólo una fase del ciclo de vida del desarrollo.

2.1.2 Cuadrante de pruebas para DevOps

LO-2.1.2	Comprender las modificaciones del cuadrante de pruebas para DevOps (K2)

Los cuadrantes de pruebas impulsados por el marco de trabajo ágil proporcionan una taxonomía en la que los equipos identifican, planifican y ejecutan las pruebas necesarias. Estos cuadrantes de pruebas se enfrentan tanto al negocio (perspectiva de usuario) como a la tecnología (perspectiva de desarrollador), son manuales o automatizados o la combinación de ambos.

Los cuatro cuadrantes de pruebas son los siguientes:



Fuente: software guru: https://sg.com.mx/revista/45/aseguramiento-agil-calidad

Cuadrante Q1: Se trata de pruebas orientadas a la tecnología que tienen como objetivo principal apoyar el equipo y los métodos impulsados por el desarrollo, como las pruebas unitarias, las pruebas de API, las pruebas de servicios web y las pruebas de componentes que mejoran el diseño del producto. Las pruebas en el Q1 se asocian a menudo con las pruebas automatizadas y la integración continua.

Cuadrante Q2: Estas son pruebas enfocadas al negocio que están principalmente dirigidas a apoyar el equipo y los métodos impulsados por el desarrollo también, tales como las utilizadas para las pruebas funcionales, pruebas de historia, prototipos y simulaciones que aseguran que sus productos de software están correctamente alineados con el negocio. Las pruebas en el Q2 a menudo se ejecutan utilizando tanto pruebas automatizadas como manuales.

Cuadrante Q3: Son pruebas de frente al negocio que se utilizan para evaluar o criticar el producto; principalmente cubren pruebas como las pruebas exploratorias, pruebas basadas en escenarios, pruebas de usabilidad, pruebas de aceptación del usuario y pruebas alfa/beta y pueden implicar demostraciones del producto diseñadas para obtener retroalimentación de los usuarios reales. Las pruebas en el Q3 se ejecutan a menudo utilizando pruebas manuales.

Cuadrante Q4: Son pruebas de cara a la tecnología que se utilizan para evaluar o criticar el producto; cubren pruebas como las de rendimiento, carga, estrés y escalabilidad, pruebas de seguridad, mantenimiento y gestión de la memoria.

La principal modificación en los cuadrantes de DevOps es automatizar al máximo y tener las herramientas que soportan la automatización para que estos cuadrantes trabajen juntos de una manera más armonizada.

2.2 Desarrollo guiado por pruebas (TDD) y DevOps

2.2.1 TDD - Definición

LO-2.2.1	Explicar el Desarrollo guiado por pruebas (TDD) y sus ventajas (K2)

El enfoque y método DevOps que fomenta el enfoque TDD (Test-Driven Development — el desarrollo guiado por pruebas) principalmente en el sentido de fallar rápido, fallar barato, ha llevado a poner también énfasis en BDD (Behavior-Driven Development) [ver más información sobre BDD en 2.6 de este

temario], que consiste esencialmente en un proceso ágil de desarrollo de software que fomenta más la colaboración entre desarrolladores, QA y participantes no técnicos u orientados al negocio.

El desarrollo guiado por pruebas es una técnica implementada por los desarrolladores mientras que, como programador, primero debe escribir la(s) prueba(s) que al final falla(n) antes de escribir una nueva pieza de código. Esto ayuda a fomentar la buena calidad del producto desde las primeras etapas.

Algunas de las ventajas de TDD son:

- Código mantenible y extensible
- Pruebas que pueden servir de documentación
- Interfaces públicas bien definidas
- Refactorización más fácil y segura
- Mejor calidad del código
- Reducción de tiempo
- Ahorro de costes a largo plazo
- Criterios de aceptación ejecutables

2.2.2 Marco de trabajo xUnit

LO-2.2.2	Aplicar un marco de trabajo xUnit para realizar el TDD (K3)

HO-2.2.2	Realizar un ejercicio guiado de creación de pruebas usando el marco de trabajo xUnit contra un código dado (HO-1)

xUnit es un marco de trabajo, generalmente compuesto por herramientas de código abierto, que está disponible para la comunidad de programadores que se dedican a las pruebas unitarias. Herramientas como xUnit.net para el marco de trabajo .NET Framework, o JUnit para el marco de trabajo JAVA Framework.

2.3 Análisis estático

2.3.1 Lineamientos de codificación y otras pruebas estáticas

LO-2.3.1	Explicar las principales características de una herramienta de análisis estático (K2)

HO-2.3.1	Demostrar cómo integrar una herramienta de análisis estático en el pipeline de DevOps (HO-0)

Las pruebas estáticas consisten en varias técnicas, como la consideración de patrones de diseño, la ejecución de revisiones, la utilización de herramientas de análisis de código, la implementación de directrices, etc.

Entre estas técnicas, podemos encontrar directrices de codificación que han sido creadas para ayudar a detectar errores en las primeras fases de la creación del código. Estas directrices ayudan a reducir el costo extra en el que incurre el proyecto de software, ya que están relacionados con el tipo de código que los programadores están escribiendo.

Su objetivo es tener un código eficiente y eficaz desarrollando de forma unificada para reducir los posibles desperdicios que puedan aparecer en las etapas de cambio y mantenimiento del código.

Cuando las directrices de codificación se mantienen correctamente, la legibilidad y la comprensibilidad del código aumentan, por lo que se reduce la complejidad del mismo.

Esta fase de análisis puede automatizarse con herramientas de análisis estático. Existen muchas herramientas en el mercado que cubren la mayoría de los lenguajes de programación. Las principales características que abordan estas herramientas son:

- Comprobación del cumplimiento de las directrices de codificación Analizar el código en busca de construcciones problemáticas o áreas potencialmente problemáticas como la conversión de punteros, variables no inicializadas, código muerto, etc.
- Código apestoso (code smell) Comprobaciones de las vulnerabilidades de seguridad
- Malas prácticas de programación

2.4 Análisis dinámico

2.4.1 Cobertura de código

LO-2.4.1	Explicar las principales características de una herramienta de cobertura de código (K2)

HO-2.4.1	Realizar un ejercicio que ofrezca una pista de integración de una herramienta de análisis estático en el pipeline de DevOps y comprobar la cobertura de pruebas de un código determinado y sus pruebas (HO-2)

Las herramientas de cobertura de código ayudan a los desarrolladores mientras implementan y cambian el código. Proporcionan habilidades a los desarrolladores, en las que las principales son:

- Observar y comprender cuánto de su código se ha ejecutado mientras se utiliza la aplicación
- medir el tipo de cobertura de código elegido
- vincularse con el código fuente y la(s) prueba(s) ejecutada(s) en el código
- informar sobre el código monitoreado e indicar qué áreas del código fueron (no) cubiertas por las pruebas ejecutadas

2.4.2 Fugas de memoria

LO-2.4.2	Explicar el concepto de detección de fugas de memoria (K2)

HO-2.4.2	Demostrar cómo integrar una herramienta de detección de fugas de memoria en el pipeline de DevOps (HO-0)

Para detectar las fugas de memoria, es necesario observar el uso de la memoria RAM del sistema mientras se ejecuta el código y comprobar si el programa libera correctamente la memoria que ya no se necesita. Cuando la aplicación asigna memoria nueva para una operación y pierde la referencia a la asignación anterior, el sistema puede quedarse sin memoria antes o después. Esto hace que la memoria del sistema se agote, que las respuestas del sistema sean más lentas y que finalmente se produzca un fallo o un cuelgue.

Las herramientas de detección de fugas de memoria ayudan a identificar las fugas, que pueden ser corregidas por los desarrolladores.

2.4.3 Medición del rendimiento del código

LO-2.4.3	Explicar las principales tareas de una herramienta de medición del rendimiento del código (K2)

HO-2.4.3	Demostrar cómo integrar una herramienta de medición del rendimiento del código en un pipeline de DevOps (HO-0)

El código que es desarrollado por los programadores requiere ser efectivo, lo que significa que debe hacer la funcionalidad requerida, y al mismo tiempo ser eficiente con los recursos proporcionados (como la memoria, CPU, espacio de disco, etc.). El rendimiento de los sistemas es uno de los indicadores clave de la satisfacción de los clientes, de ahí la necesidad de comprobar pronto el rendimiento del código. Las principales métricas y mediciones establecidas para el rendimiento del código están relacionadas con el consumo de recursos, como la memoria, el espacio de disco de la CPU, etc. También es importante tener en cuenta el tiempo de respuesta (o latencia) de las transacciones que la aplicación o el sistema proporcionan mediante el monitoreo y la medición de estas transacciones y sus partes del código.

Otras mediciones son la tasa de error en el código, la disponibilidad de la aplicación a lo largo del tiempo y, en algunos casos, la forma en que se utilizan los recolectores de desechos.

A nivel de código, se mide el tiempo de ejecución de varios métodos, incluyendo detalles como el tiempo de ejecución de los bucles y el número de veces que se ejecuta un bucle, etc. Esto permite optimizar el código y ajustar el rendimiento.

2.5 Pruebas de integración y de sistema

2.5.1 Integración y automatización de pruebas del sistema - Pruebas de API

LO-2.5.1	Integrar las pruebas de API en un pipeline de CI (K3)

Las Interfaces de Programación de Aplicaciones (Application Programming Interface, por sus siglas en inglés, API) se utilizan para conectar las diferentes

partes de la aplicación o el sistema. La prueba de API es un tipo de prueba de software que implica la comprobación de API directamente y como parte de la prueba de integración para determinar si cumplen las expectativas de funcionalidad, fiabilidad, rendimiento y seguridad. Dado que las API no constan de una interfaz gráfica de usuario, la prueba de API se realiza en la capa de comunicación e incluye el envío de tipos de solicitudes y el análisis de sus respuestas. Las API son utilizadas interna o externamente por la aplicación y el sistema, por lo tanto, serán probadas ya sea en el nivel de prueba de integración o del sistema. La forma más eficiente de probar las API es hacer que estas pruebas se integren en los pipelines de CI o para estas.

Los principales pasos para realizar pruebas de API en un pipeline de CI pueden incluir lo siguiente:

- Construir la implementación del servicio de backend
- Desplegar el servicio de backend en el entorno de integración
- Desplegar la API en el entorno de integración
- Desencadenar la ejecución de las pruebas de API en el entorno desplegado (ya sea programado en esa secuencia automática o ad-hoc)
- Informe de los resultados

2.5.2 Automatización de la prueba del sistema — Pruebas de GUI

LO-2.5.2	Integrar una herramienta de prueba de automatización GUI en un pipeline de CI (K3)

La prueba de la Interfaz Gráfica de Usuario (GUI por sus siglas en inglés, Graphical User Interface) se hace principalmente a nivel de la prueba del sistema. Las pruebas de GUI consisten en probar tanto los objetos de la interfaz de usuario (IU) como la funcionalidad proporcionada durante el uso de la IU por los clientes o usuarios finales. Hay muchas herramientas que permiten a los testers realizar pruebas de GUI. En DevOps, estas herramientas, sus pruebas y el contenido de las pruebas también necesitan ser integradas con los pipelines de CI. La integración de estas pruebas depende de las plataformas y sistemas operativos utilizados para desarrollar y principalmente desplegar.

Los principales pasos requeridos para integrar las pruebas de GUI al pipeline de CI pueden incluir lo siguiente:

- Desplegar todos los servicios en el entorno de prueba del sistema
- Desplegar las pruebas de GUI en el entorno de prueba del sistema
- Desencadenar la ejecución de las pruebas de GUI en el entorno de prueba del sistema (ya sea programada en esa secuencia automática o ad-hoc)
- Informar los resultados.

2.6 Pruebas de aceptación

2.6.1 BDD y ATDD

LO-2.6.1	Integrar una herramienta de desarrollo guiado por comportamiento en el pipeline de DevOps (K3)

HO-2.6.1	Demostrar cómo integrar una herramienta de desarrollo guiado por comportamiento en el pipeline de DevOps (HO-0)

El desarrollo guiado por comportamiento (BDD) es un método para desarrollar las características en base a su comportamiento. El comportamiento se explica básicamente en términos de ejemplos en un lenguaje muy simple que puede ser entendido por cualquier experto en negocios (p.ej. el lenguaje Gherkin). El desarrollo guiado por pruebas de aceptación (ATDD por sus siglas en inglés) es una metodología de desarrollo que pretende llevar la visión del cliente a las fases de desarrollo y prueba. Todo el equipo, que puede estar formado principalmente por clientes de negocios, desarrolladores y testers, colabora para definir los criterios de aceptación (p. ej. de una epopeya o una historia) antes de que comience la implementación. Estas pruebas de aceptación están respaldadas por ejemplos adecuados y otra información necesaria.

Hay herramientas de DevOps disponibles en el mercado que apoyan lo anterior y permiten a los equipos de DevOps integrar las pruebas relacionadas con el entorno de pruebas del sistema desplegado, así como con los entornos de producción (para las etapas de pruebas de preproducción y producción para el cliente).

Capítulo 3 - Pruebas específicas de DevOps

Palabras clave: Stage Rollout, Dark Launch, Feature Toggles

LO-3.2.0	Comprender las diferencias entre stage rollout, dark launch y la actualización estándar (K2)
LO-3.3.1	Recordar los diferentes tipos de toggles (K1)
LO-3.3.2	Comprender el impacto de los feature toggles en las pruebas en producción (K2)
LO-3.3.4	Comprender los riesgos asociados a los toggles (K2)

HO#	Descripción
HO-3.1.0	Demostrar cómo se pueden realizar liberaciones de características específicas de los usuarios utilizando uno de los varios métodos — Usuarios internos, liberaciones canarias (canary releases) o pruebas A/B (HO-0)
HO-3.2.2	Demostrar cómo se puede implementar el dark launch para un sistema determinado (HO-0)
HO-3.3.2	Realizar un ejercicio guiado de creación de varias pruebas funcionales para los feature toggles y ver cómo se pueden ejecutar en un entorno determinado (HO-1)

3.1 Pruebas de características específicas del usuario

HO-3.1.0	Demostrar cómo se pueden realizar liberaciones de características específicas de los usuarios utilizando uno de los varios métodos - Usuarios internos, liberaciones canarias (canary releases) o pruebas A/B (HO-0)

La prueba de características (feature testing en inglés) es un proceso creado para proporcionar una visión de los cambios realizados en un sistema de software para añadir una o más características nuevas o para hacer

modificaciones en las características ya existentes. Estas características están diseñadas para ser útiles, intuitivas y efectivas.

3.1.1 Usuario interno

Las pruebas de usuario interno abren el proceso de diseño de las características a un público más amplio. Fomenta un trabajo más colaborativo entre los miembros del equipo, de modo que la gente adquiere una fuerte inteligencia en el diseño y una retroalimentación reflexiva en torno a la usabilidad y el diseño.

3.1.2 Liberación canaria

La liberación canaria (canary release), es una técnica que se utiliza para reducir el riesgo de introducir una nueva versión de software en la producción, extendiendo (rolling out) gradualmente el cambio a un pequeño subgrupo de usuarios o servidores, antes de extenderlo (rolling out) a toda la plataforma/infraestructura y ponerlo a disposición de todos los usuarios.

3.1.3 Pruebas A/B

La prueba A/B es un proceso creado para comparar dos versiones de una misma variable (p. ej. una página web, un correo electrónico u otro activo de marketing), por lo general probando la respuesta de un sujeto a la variante A frente a la variante B, y determinando cuál de las dos variantes es más eficaz.

3.2 Stage rollout, Dark Launch & Actualización estándar

LO-3.2.0	Comprender las diferencias entre stage rollout, dark launch y la actualización estándar (K2).

3.2.1 Stage Rollout

El feature toggle puede establecerse como un nivel de sistema (es decir, para todos los usuarios) o para usuarios o grupos de usuarios específicos.
Esto permite eficiencia y flexibilidad en las pruebas de producción. Esto también crea una posibilidad para el enfoque de stage rollout, es decir, la

ampliación (o reducción si es necesario) con los diferentes usuarios que contiene el sistema.
Además, el stage rollout sólo puede utilizarse para las actualizaciones de las aplicaciones, pero no cuando se publica una aplicación por primera vez.

3.2.2 Dark Launch

HO-3.2.2	Demostrar cómo se puede implementar el dark launch para un sistema determinado (HO-0)

La capacidad de desarrollar microservicios en un enfoque independiente (por ejemplo, el servicio de backend y el servicio de frontend — GUI), crea situaciones en las que la funcionalidad y el lado de la lógica de negocio (por ejemplo, el backend) están listos, y pueden estabilizarse hasta que el frontend (por ejemplo, GUI) esté listo. Esto se conoce como dark launch de la funcionalidad, y requiere pruebas a este nivel, principalmente para asegurar que esta parte es estable, funcionando correctamente y de manera robusta.

3.2.3 Actualización estándar

La actualización estándar suele ir acompañada de un procedimiento, ya sea manual o automático, o una combinación de ambos. La actualización estándar puede requerir el reinicio de máquinas o servicios en el sistema, así como, ocasionalmente, un tiempo de inactividad total o parcial del sistema o sus partes.

3.3 Toggles

Las pruebas específicas de DevOps están relacionadas con el enfoque de DevOps y con la forma en que se desarrollan las características. Un buen ejemplo de la capacidad de controlar la aparición o habilitación de una característica es el método y enfoque “Feature Toggles”. Los feature toggles permiten al desarrollador habilitar o deshabilitar la característica fácilmente, ya sea para fines de prueba o de despliegue real. El enfoque de los feature toggles crea flexibilidad mientras se hacen las pruebas, ya que se pueden configurar estos toggles (habilitarlos o deshabilitarlos) para que la característica se ejecute o no. La flexibilidad también existe en una situación en la que la característica no está completamente lista para ser desplegada, por lo tanto, a través del

toggle podemos deshabilitarla sin retroceder (rollout) o tocar el código. Es importante asegurarse de que la característica no tenga impacto en otras características cuando se deshabilite. Los feature toggles pueden establecerse a nivel de sistema, como se mencionó anteriormente, en el Stage Rollout.

3.3.1 Tipos de Toggles

LO-3.3.1	Recordar los diferentes tipos de toggles (K1)

Hay varios tipos de toggles, tales como: liberación, permanente, Ops, permiso y experimento.

3.3.2 Pruebas funcionales para los Toggle States

LO-3.3.2	Comprender el impacto de los feature toggles en las pruebas en producción (K2)

HO-3.3.2	Realizar un ejercicio guiado para crear varias pruebas funcionales para los feature toggles y ver cómo se pueden ejecutar en un entorno determinado (HO-1)

Los tipos de toggles que se mencionan pueden probarse mientras el toggle está apagado, encendido o en los modos de encendido y apagado. Las pruebas también pueden combinar los diferentes toggles, así como los diferentes tipos de usuarios que pueden existir para cada toggle o combinación de toggles. Además, el enfoque de los feature toggles crea flexibilidad mientras se realizan las pruebas, así como si la característica no está totalmente lista para ser desplegada en la producción. El feature toggle se puede establecer como un nivel de sistema (es decir, para todos los usuarios) o para usuarios o grupos de usuarios específicos.

3.3.3 Pruebas no funcionales para los estados de los toggles

Tipos de pruebas importantes además de la funcionalidad: seguridad, rendimiento, usabilidad, compatibilidad y compatibilidad con versiones anteriores.

3.3.4 Riesgos del uso de los toggles

LO-3.3.4	Comprender los riesgos asociados a los toggles (K2)

Los toggles aportan flexibilidad, y por lo tanto más complejidad, lo que conlleva posibles riesgos, tales como: configuración complicada, demasiadas condiciones que controlar como resultado de muchos tipos de toggles, combinaciones de toggles y las combinaciones con los diferentes tipos de usuarios. Para controlar mejor esas condiciones, es necesario documentar todos los datos o cambios en los datos (por ejemplo, la configuración, el aprovisionamiento de usuarios, la activación y desactivación de la característica, etc.) y es mejor separarlos, para crear la menor dependencia posible; además, se deben habilitar procedimientos de limpieza y rollback cuando sea posible.

En un proyecto de pruebas, a veces se recomienda establecer una sesión de caza de errores (bugs) para que los usuarios (testers u otros) prueben el sistema y busquen errores. Estas personas pueden recibir un reconocimiento y una compensación por informar de los bugs, especialmente los relativos a colapsos (exploits) y vulnerabilidades. Esto puede hacerse internamente en la empresa, lo que puede crear más colaboraciones si se hace en grupos o equipos mixtos entre los diferentes departamentos, así como si la empresa de productos lo animará a hacer por la multitud.

Capítulo 4 — Operaciones en DevOps

Palabras clave: Monitoreo, Monitores, Alertas, Controlabilidad, Observabilidad

LO-4.1.1	Recordar los conceptos de monitoreo del sistema de producción (K1)
LO-4.1.2	Recordar la importancia de varios tipos de alertas (K1)
LO-4.1.3	Comprender las diferencias entre las pruebas de monitores y las pruebas de alertas (K2)
LO-4.1.4	Comprender las pruebas de las características de registro (logging) en un servidor de producción (K2)

HO#	Descripción
HO-4.1.3	Demostrar alertas en un sistema utilizando conjuntos de pruebas predefinidas para monitores (HO-0)

4.1 Monitoreo de los sistemas de producción

El monitoreo es el proceso de mantener la vigilancia sobre la existencia y la magnitud de los cambios de estado y el flujo de datos en un sistema.

4.1.1 Monitoreo

LO-4.1.1	Recordar los conceptos de monitoreo del sistema de producción (K1)

El objetivo del monitoreo es identificar los fallos y ayudar a su posterior eliminación. Las técnicas utilizadas en los sistemas de monitoreo de la información se entrecruzan en los campos del procesamiento en tiempo real, las estadísticas y el análisis de datos. Un conjunto de componentes de software utilizados para la recopilación de datos, su procesamiento y su presentación se denomina sistema de monitoreo.

4.1.2 Alerta

LO-4.1.2	Recordar la importancia de varios tipos de alertas (K1)

La alerta es la capacidad de un sistema de monitoreo para detectar y notificar a los operadores sobre eventos significativos que denoten un cambio de estado grave. La notificación se denomina alerta y es un simple mensaje que puede adoptar múltiples formas: correo electrónico, SMS, mensaje instantáneo (IM, instantaneous message) o una llamada telefónica.

4.1.3 Prueba de Monitores y de Alertas

LO-4.1.3	Comprender las diferencias entre las pruebas de monitores y las pruebas de alertas (K2)

HO-4.1.3	Demostrar alertas en un sistema utilizando conjuntos de pruebas predefinidas para monitores (HO-0)

El uso principal de las capacidades de monitoreo y alerta es obtener información - por ejemplo, cuando se producen problemas en un sistema basado en códigos de error, la falta de disponibilidad de características o servicios, los fallos de áreas de código críticas que utilizan trampas (por ejemplo, trampas SNMP para fallos en transacciones relacionadas con el dinero), problemas de rendimiento (tanto basados en máquinas como en aplicaciones), problemas de seguridad, problemas de usabilidad e información sobre la comprensión del comportamiento del usuario.

El monitoreo y las alertas que tenemos en los sistemas tienen que ser probados también. Los tipos de pruebas que se pueden realizar para probar el monitoreo pueden incluir lo siguiente:

- Pasos y opciones de configuración de las pruebas
- Probar el funcionamiento del monitor produciendo situaciones que deben ser monitoreadas y alertadas Capacidad de realizar pruebas de los monitores
    - Controlabilidad - Capacidad de proporcionar datos para desencadenar los monitores y las alertas
    - Observabilidad - Capacidad de observar las acciones tomadas
- Probar la eliminación de los monitores

Para probar las alertas se pueden realizar los siguientes tipos de pruebas:

- Prueba de las reglas
- Prueba de modificación de reglas y alertas
- Prueba de las alertas después y durante la eliminación de los monitores

4.1.4 Prueba de registro

LO-4.1.4	Comprender las pruebas de las características de registro en un servidor de producción (K2)

Además del monitoreo y las alertas, el sistema genera registros (logs) y también se definen los tipos de registro y/o los niveles de registro. El nivel de registro se suele establecer a fin de proporcionar una información mínima o extensa, para mejorar las actividades de análisis y solución de problemas, por lo que es necesario comprobar la corrección de los datos proporcionados en los registros. El mecanismo de registro puede consumir recursos del sistema, por lo que es importante saber cuál es el nivel de registro requerido y comprobar que proporciona la información necesaria para ese nivel, además de consumir recursos mínimos del sistema (por ejemplo, normalmente se configura un nivel de registro bajo, como errores solamente, para evitar el consumo de I/O). Además, a efectos de mantenimiento, es importante comprobar el consumo de recursos y el comportamiento del sistema cuando el nivel de registro es alto y se proporciona más información a través de los registros.

Hay más pruebas que deben realizarse desde el punto de vista de la funcionalidad de los registros, como: la exactitud de la información de los registros, el tamaño de estos, el archivo, el registro de tiempo, encriptación/enmascaramiento de datos sin restricciones, la clasificación apropiada de los eventos (información, advertencia, error, etc.). Como los registros crean carga en el sistema, también se requiere un punto de referencia del rendimiento, a fin de medir la carga y la forma en que el sistema funciona en condiciones normales y pesadas mientras se establecen los diferentes niveles de registro. Además, podría haber diferentes tipos de registros (o archivos de registro), como “todos los registros” sólo “errores de registro”, etc. También es necesario probarlos y optimizarlos, así como comprobar el tráfico que generan.

Capítulo 5 - DevOps y la nube

Palabras clave: Infraestructura como servicio (por sus siglas en inglés, IaaS), Hardware como servicio (por sus siglas en inglés, HaaS), Plataforma como

servicio (por sus siglas en inglés, PaaS), Software como servicio (por sus siglas en inglés, SaaS), granja de TI, contenerización de aplicaciones

LO-5.1.0	Recordar las principales formas de tecnología de la nube (K1)
LO-5.1.3	Recordar las principales ventajas de la tecnología de la nube (K1)
LO-5.1.4	Comprender las principales ventajas de la tecnología de la nube (K2)
LO-5.1.5	Recordar los principios de tener DevOps en las plataformas de la nube (K1)

HO#	Descripción
HO-5.1.4	Demostrar cómo se pueden aplicar los Dockers en un contenedor en un entorno virtualizado (HO-0)

5. 1ntroducción a DevOps con la nube

La computación en nube consiste en tres tipos distintos de servicios de computación, entregados a distancia a los clientes a través de Internet.
El modelo de negocio común de la nube es que los clientes suelen pagar una cuota de servicio mensual o anual a los proveedores basados en la nube, para acceder a los sistemas que ofrecen software como servicio, plataformas como servicio e infraestructura como servicio a los suscriptores. Los clientes que se suscriben a los servicios de computación en nube pueden obtener una variedad de beneficios, dependiendo de las necesidades de su negocio en un momento determinado.
Las soluciones de la nube son similares a lo que antes se llamaban “centros de alojamiento” (hosting centers), pero con una integración, programación y capacidades operativas adicionales.

5.1.1 IaaS, PaaS, SaaS

LO-5.1.0	Recordar las principales formas de la tecnología de nube (K1)

IaaS (Infraestructura como Servicio, por sus siglas en inglés, Infrastructure as a Service) permite a los clientes utilizar remotamente el hardware y los recursos de la tecnología de la información sobre una base de "pago por uso" desde la perspectiva de un modelo de negocios.
También se conoce como HaaS (por sus siglas en inglés, "Hardware as a Service"). Los principales actores de la IaaS incluyen compañías como IBM, Google y Amazon. La IaaS emplea la virtualización, un método de creación y gestión de recursos de infraestructura en la "nube". La IaaS proporciona a las pequeñas o medianas empresas una gran ventaja, ya que les permite ampliar gradualmente su infraestructura de TI sin necesidad de grandes inversiones de capital en hardware y sistemas periféricos. Si en el pasado, cada empresa creaba su propia granja de TI — incluso para uso interno — entonces la IaaS viene como una opción. Es importante señalar que hay empresas cuyo negocio impedirá el uso de la IaaS (por ejemplo, por razones de seguridad — aunque la mayoría de estas plataformas soportan un alto nivel de seguridad).

PaaS (Plataforma como Servicio, PaaS por sus siglas en inglés, Platform as a Service) proporciona a los clientes la capacidad de desarrollar y publicar aplicaciones personalizadas en un entorno alojado a través de la web. Representa un nuevo modelo para el desarrollo de software que está aumentando rápidamente en popularidad. Un ejemplo de PaaS es Salesforce.com. La PaaS proporciona un marco de trabajo para el desarrollo, la prueba, despliegue y el mantenimiento ágil de software en un entorno integrado. Al igual que el SaaS, el principal beneficio de la PaaS es un menor costo de uso. Los proveedores de PaaS se encargan del mantenimiento de la plataforma y de las actualizaciones del sistema, lo que da como resultado una solución más eficiente y rentable, principalmente para el desarrollo de software empresarial.

SaaS (Software como Servicio, SaaS por sus siglas en inglés, Software as a Service) ofrece a los clientes la posibilidad de utilizar aplicaciones de software a distancia a través de un navegador de Internet. El software como servicio también se conoce como "software por demanda".
Los clientes pueden acceder a las aplicaciones SaaS desde cualquier lugar a través de la web, ya que los proveedores de servicios alojan las aplicaciones y sus datos asociados en su ubicación. El principal beneficio del SaaS es un menor costo de uso, ya que las tarifas de suscripción requieren una inversión mucho menor que la que se suele encontrar en el modelo tradicional de entrega de software. Los derechos de licencia, los costos de instalación, las cuotas de mantenimiento y las cuotas de apoyo que se asocian habitualmente con el

modelo tradicional de entrega de software pueden eliminarse virtualmente suscribiéndose al modelo SaaS de entrega de software. Entre los ejemplos de SaaS se incluyen las aplicaciones de Google y las aplicaciones de correo electrónico basadas en Internet, como Yahoo! Mail, Hotmail y Gmail.

DevOps tiene un gran valor en el desarrollo de aplicaciones SaaS que se ejecutan en la misma infraestructura que ofrece el proveedor de la nube, DevOps también puede utilizarse para ayudar en la migración de aplicaciones a otros modelos de computación en la nube, como la plataforma como servicio (PaaS) y la infraestructura como servicio (IaaS).

5.1.2 El ajuste de la nube en DevOps

Las principales ventajas de la computación en la nube son: Disponibilidad 24/7, flexibilidad relacionada con la capacidad del sistema - capacidad de escalar fácilmente por el consumo requerido, actualizaciones automatizadas y rápidas del software, mayor colaboración, reducción del mantenimiento y la capacidad de proporcionar un marco de trabajo para ampliar las pequeñas empresas que ahora pueden reducir sus costos de TI.

5.1.3 Virtualización

LO-5.1.3	Recordar las principales ventajas de la tecnología de nubes (K1)

La nube se basa principalmente en la virtualización. Una de las principales ventajas de la nube es que ofrece la posibilidad de acceder a potentes recursos informáticos de forma incremental. De esta forma se nivela el campo de juego de las pequeñas y medianas organizaciones, proporcionándoles las herramientas y la tecnología necesarias para competir en el mercado mundial, sin la inversión previa requerida en recursos de TI. Esto también se aplica a los clientes que se suscriben a los servicios de computación prestados a través de la “nube”. Pueden reducir en gran medida los gastos en servicios de TI de sus organizaciones y, en el proceso, acceder a servicios de computación más ágiles y flexibles a nivel de empresa.

5.1.4 Contenerización de Aplicaciones

LO-5.1.4	Comprender las principales ventajas de la tecnología de nubes (K2)

HO-5.1.4	Demostrar cómo se pueden aplicar los Dockers en un contenedor en un entorno virtualizado (HO-0)

El hecho de que la computación en nube esté centralizada por naturaleza, proporciona a la automatización de DevOps, por ejemplo, una plataforma estándar y centralizada para la prueba, el despliegue y la producción. El uso de una plataforma de nube resuelve muchos problemas de complejidad distribuida.

Además, la automatización de DevOps se está volviendo más centrada en la nube. La mayoría de los proveedores públicos y privados de computación en nube apoyan sistemáticamente a DevOps en sus plataformas, que incluyen herramientas de integración continua y de desarrollo continuo. Esta estrecha integración reduce los costos asociados con la tecnología de automatización de DevOps en las instalaciones y proporciona una gobernanza y un control centralizados para un proceso de DevOps sólido.

Hace años que el mundo de la virtualización ya existe, y la utilización de los recursos es cada vez más desarrollada y necesaria. El enfoque de DevOps ha cambiado aún más la mejora del mundo de la virtualización, y en estos campos virtuales han surgido tecnologías más avanzadas, como los contenedores Docker. Los contenedores Docker están diseñados para funcionar en todos los entornos, desde las computadoras físicas a las máquinas virtuales, desde el metal al descubierto, hasta la nube, etc.

La contenerización de aplicaciones es una alternativa ligera a tener una máquina virtual que incluye la encapsulación de una aplicación con su propio sistema operativo en un solo lugar: un contenedor. El término “contenedor” toma su significado de la terminología logística, empaquetar un contenedor.

5.1.5 Máquinas virtuales v/s Contenedores

LO-5.1.5	Recordar las principales diferencias entre las máquinas virtuales y los contenedores (K1)

Para crear una aplicación en la nube, podemos utilizar los microservicios como un enfoque arquitectónico, donde cada aplicación se construye como un conjunto de servicios. Cada servicio ejecuta sus propios procesos y se comunica a través de interfaces de programación de aplicaciones (API).

Docker funciona proporcionando una forma estándar de ejecutar su código. Docker es un sistema operativo para contenedores. La forma en que una máquina virtual virtualiza (elimina la necesidad de administrar directamente) el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos simples que puedes usar para construir, iniciar o detener los contenedores.

Los contenedores son una abstracción en la capa de aplicación que empaqueta el código y las dependencias juntas. Múltiples contenedores pueden ejecutarse en la misma máquina y compartir el núcleo (kernel) del sistema operativo con otros contenedores, cada uno de los cuales se ejecuta como procesos aislados en el espacio de usuario.

Las actuales y más populares plataformas de nubes son: Amazon Web Services (AWS), Microsoft Azure, Google cloud Platform y Alibaba Cloud.

Capítulo 6 - Diversas herramientas y tecnologías

Palabras clave: Infraestructura como código (por sus siglas en inglés, IaC), repositorio binario, repositorio de software

LO-6.1.0	Recordar las principales ventajas de la infraestructura de modelización con código (K1)
LO-6.1.3	Aplicar las principales características de una infraestructura como herramienta de código (K3)

HO#	Descripción
HO-6.1.3	Demostrar cómo varias herramientas de IaC pueden ser orquestadas y funcionar de manera armonizada (HO-0)

6.1 Infraestructura y repositorios

LO-6.1.0	Recordar las principales ventajas de la infraestructura de modelado con código (K1)

6.1.1 La infraestructura como código (IaC)

La infraestructura como código (IaC) es una forma ligera y basada en datos de gestionar y aprovisionar la infraestructura del sistema (redes, máquinas virtuales, equilibradores de carga y topología de conexión) — por lo general, tratando sus servidores, bases de datos, redes y otras infraestructuras como software. Se hace en un modelo descriptivo, utilizando el mismo versionado que un equipo de DevOps utiliza para el código fuente, en lugar de administrarlo con la configuración física del hardware o con herramientas de configuración interactiva.
Tiene principios similares a los de la gestión de código: el mismo código fuente genera el mismo binario, y un modelo IaC genera el mismo entorno cada vez que se aplica - por ejemplo, el pipeline de liberaciones ejecuta el modelo para configurar los entornos de destino.
De esta manera, cada vez que los miembros del equipo necesitan hacer cambios, pueden editar el código fuente y no el entorno de destino.
IaC es una práctica clave de DevOps y se utiliza en conjunto con la entrega continua.

Las principales ventajas de haber modelado la IaC son:

- Más rápido y simple
- Evitar las inconsistencias de despliegue de la configuración del sistema
- Reducir los riesgos
- Aumentar la eficiencia y la productividad de la I+D, especialmente en la Nube
- Reducción de costes y ahorro

6.1.2 Repositorios binarios

Un repositorio binario, o repositorio de software, o “repo” para abreviar, es un lugar de almacenamiento de paquetes de software. Por lo general, se almacena una tabla de contenidos, así como metadatos. Los repositorios también administran paquetes en grupos. A veces, la agrupación es para un lenguaje de programación, a veces para un sistema operativo completo, a veces la licencia de los contenidos es el criterio.

El lado del servidor de un repositorio de software es típicamente administrado por el control de la fuente o los administradores del repositorio. Hay administradores de repositorios que permiten agregar otras ubicaciones de repositorios en un URL y proporcionar un servidor proxy. Mientras crean construcciones continuas y generan muchos artefactos, los administradores de repositorios a menudo los almacenan en un lugar central, y es importante que estos administradores de repositorios también se encarguen de eliminar automáticamente las construcciones que no se liberan.
En el lado del cliente, los administradores de paquetes ayudan a instalar y actualizar los repositorios.

Como parte del ciclo de vida del desarrollo, el código fuente se construye continuamente en artefactos binarios mediante una integración continua. Esto puede interactuar con un administrador de repositorios binarios, similar a la situación a la que se enfrentaría un programador al obtener artefactos de los repositorios y hacer push a las construcciones allí.

Una estrecha integración con los servidores de CI permite almacenar metadatos importantes como:

- ¿Qué usuario desencadenó esta construcción? (¿manualmente? o ¿haciendo el commit del control de versiones?)
- ¿Qué módulos fueron construidos?
- ¿Qué fuentes se utilizaron? (commit id, revisión, rama)
- ¿Qué dependencias se utilizaron?
- ¿Qué variables de entorno se utilizaron?
- ¿Qué paquetes se instalaron?

6.1.3 Herramientas IaC

LO-6.1.3	Aplicar las principales características de una infraestructura como herramienta de código (K3)

HO-6.1.3	Demostrar cómo varias herramientas de IaC pueden ser orquestadas y funcionar de manera armonizada (HO-0)

Las herramientas IaC se clasifican en las siguientes dos categorías, que a menudo pueden superponerse:

- Herramienta de orquestación de la configuración: estas herramientas están diseñadas para automatizar el despliegue de nuestra infraestructura.
- Herramientas de gestión de la configuración: estas herramientas están diseñadas para ayudar a configurar el software y los sistemas en la infraestructura proporcionada.

Algunas de las herramientas actuales del IaC permiten orquestar/proveer la infraestructura, así como configurarla, y como la infraestructura se vuelve más compleja, es común ver ambos tipos de herramientas utilizadas.
Aquí están algunas de las herramientas actuales que son populares en el paisaje de IaC:

Terraform es una herramienta de aprovisionamiento de infraestructura que soporta múltiples proveedores de nube. Nos permite, como "desarrolladores", representar nuestra infraestructura, independientemente de si es AWS, Google Cloud o Azure, en un lenguaje bien definido conocido como HCL. Terraform es altamente extensible a través de plugins y tiene una comunidad muy fuerte a su alrededor que produce grandes módulos de código abierto para el aprovisionamiento de trozos de infraestructura

AWS CloudFormation no es agnóstica a la nube y se dedica a proveer infraestructura dentro de las cuentas de AWS. Nos permite representar nuestra infraestructura AWS en archivos de configuración JSON o YAML que luego podemos ejecutar para crear, actualizar y destruir recursos. CloudFormation está muy bien integrado en el ecosistema AWS y ha desarrollado muchas

características para previsualizar, revertir y gestionar los cambios en los recursos.

Chef cae bajo la herramienta de gestión de configuración. Nos permite crear recetas y manuales que definen los pasos exactos necesarios para alcanzar la configuración necesaria de nuestra aplicación. Chef, como Terraform, soporta múltiples proveedores de nube. Utiliza el lenguaje comúnmente conocido, Ruby. Se usa típicamente para configurar instancias de Elastic Compute (EC2) e incluso servidores en las instalaciones.

Puppet es otra herramienta de gestión de configuración basada en Ruby, como Chef. La diferencia entre ellas es que Puppet es conocida como una herramienta declarativa. Esto significa que nosotros, como “desarrolladores”, definimos lo que se supone que es nuestra infraestructura y Puppet averigua cómo hacer eso posible.

Ansible es una herramienta de automatización de infraestructuras de Red Hat. Los desarrolladores describen cómo se relacionan sus componentes y el sistema entre sí. Su objetivo es gestionar y codificar los sistemas de extremo a extremo en lugar de hacerlo de forma independiente. Estas definiciones están escritas en YAML y se conocen como Playbooks.

Juju es una herramienta IaC de Ubuntu, que permite a los desarrolladores representar su infraestructura como encantos que son conjuntos de scripts que despliegan y operan sistemas. Estos encantos pueden ser empaquetados como paquetes para desplegar toda la infraestructura de una aplicación.

uDeploy es un marco de trabajo de despliegue de automatización que reduce los errores de despliegue y mejora la eficiencia, la corrección y la trazabilidad. El IBM **UrbanCode Deploy** es una herramienta para automatizar los despliegues de aplicaciones a través de nuestros entornos. Está diseñado para facilitar una rápida retroalimentación y una entrega continua en el desarrollo ágil, mientras que proporciona los registros de auditoría, la creación de versiones y las aprobaciones necesarias en la producción.

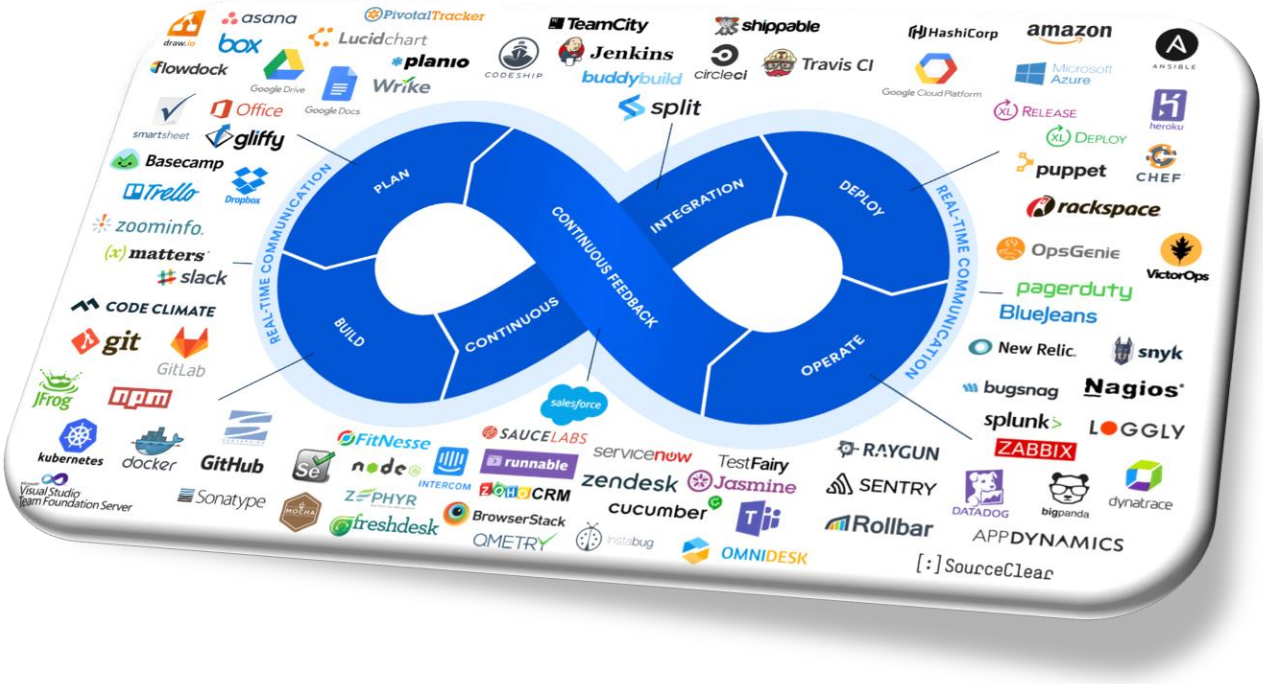
JFrog Artifactory es una herramienta diseñada para almacenar la salida binaria del proceso de construcción para su uso en la distribución y el despliegue. Artifactory proporciona soporte para varios formatos de paquetes como Maven, Debian, NPM, Helm, Ruby, Python y Docker. JFrog ofrece alta disponibilidad, replicación, recuperación de daños, escalabilidad, y trabaja con muchas ofertas de almacenamiento en la nube y on-prem.

Esto seguro que no es una lista comprensiva. Constantemente se están desarrollando nuevas herramientas tanto en el panorama de la orquestación de la configuración como en el de la gestión.

6.1.4 Otras herramientas

HO-6.1.4	Realizar un ejercicio guiado para buscar las herramientas más actualizadas que existen en el mercado para DevOps, incluyendo aquellas que soportan etapas específicas, así como herramientas que son conjuntos y que proporcionan soluciones holísticas (HO-1)

DevOps aporta una visión / perspectiva general sobre la variedad de herramientas que existen en el mercado, y que están apoyando su enfoque holístico.
La siguiente imagen puede describir esta variedad de herramientas, y su continua actualización.


PLAN
CONTINUOUS
INTEGRATION
DEPLOY
BUILD
CONTINUOUS
OPERATE
CONTINUOUS FEEDBACK
REAL-TIME COMMUNICATION
REAL-TIME COMMUNICATION
asana
PivotalTracker
TeamCity
shippable
HashiCorp
amazon
box
Lucidchart
Jenkins
Flowdock
planio
Google Drive
Wrike
CODESHIP
buddybuild
circleci
Travis CI
Google Cloud Platform
Microsoft Azure
ANSIBLE
split
Office
Google Docs
RELEASE
heroku
smartsheet
gliffy
DEPLOY
Basecamp
puppet
CHEF
Trello
Dropbox
rackspace
zoominfo
matters
OpsGenie
VictorOps
slack
pagerduty
BlueJeans
CODE CLIMATE
git
GitLab
New Relic
snyk
JFrog
npm
bugsnag
Nagios
splunk>
LOGGLY
kubernetes
docker
GitHub
FitNesse
SAUCELABS
servicenow
TestFairy
RAYGUN
ZABBIX
node
runnable
zendesk
Jasmine
SENTRY
Visual Studio Team Foundation Server
Sonatype
ZEPHYR
cucumber
Rollbar
DATADOG
bigpanda
dynatrace
freshdesk
BrowserStack
QMETRY
APPDYNAMICS
OMNIDESK
[:]SourceClear
Salesforce
INTERCOM

Referencias

- Este documento ha sido diseñado y creado utilizando las experiencias de primera mano recogidas en la industria por las PYMES involucradas en la creación de DevOps United.

A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives — L. Anderson, P. W. Airasian, and D. R. Krathwohl (Allyn & Bacon 2001)
Revised Bloom's Taxonomy Action Verbs. Available at https://www.apu.edu/live_data/files/333/blooms_taxonomy_action_verbs.pdf

[DevOps for the Modern Enterprise] Winning Practices to Transform Legacy IT Organizations — Mirco Hering (April 2018)

[The DevOps HandOPok] How to Create World-Class Agility, Reliability, & Security in Technology Organizations — Gene Kim, Jez Humble, John Willis & Patrick DeOPis (Octobre 2016 edition)

[Embedding DevOps in the Enterprise] Cutter IT Journal (November 2011 edition)

[DevOps Guide] The IT Revolution (2015 edition)

[DevOps for Dummies] IBM — Sanjeev Sharma & Bernie Coyne (John Wiley & Sons, 2nd edition 2015)

[ISTQB-FL 2018] ISTQB Foundation Level Syllabus version 2018. Available at https://www.istqb.org/downloads/category/51-ctfl2018.html

[Agile Alliance organization] https://www.agilealliance.org