



APUNTES 05


IMPLANTACIÓN DE SISTEMAS SEGUROS DE DESPLEGADO DE SOFTWARE

PUESTA EN PRODUCCIÓN SEGURA

ALBA MOREJÓN GARCÍA

2024/2025

Ciberseguridad en Entornos de las Tecnologías de la Información



ÍNDICE

1. Puesta Segura en producción y DevOps.
 - 1.1. Desarrollo Ágil.
2. Control de versiones.
3. Sistemas de automatización de construcción
4. Integración continua y automatización de pruebas.
5. Virtualización y contenedores.
6. Gestión automatizada de configuración de sistemas
7. Herramientas de simulación de fallos
8. Orquestación de contenedores.

Caso práctico

Julián ha empezado un nuevo trabajo en el departamento de DevOps de una empresa de Ciberseguridad. Dicho departamento trabajará creando aplicaciones y desarrollos junto a los analistas de seguridad del SOC (Centro de Operaciones de Seguridad). El modo de trabajo será de entregas continuas en vez de esperar a que el proyecto esté finalizado. Al estar junto al consumidor de estas aplicaciones, el feedback será continuo entre creador y consumidor y podrán pulirse muchas situaciones antes de llegar a producción, siendo un beneficio para los desarrolladores y los clientes o consumidores.

El ciclo de desarrollo de software ha cambiado mucho durante los últimos años, nuevos modelos como DevOps con entregas y alineamientos continuos con las necesidades del cliente han cambiado la metodología e incluso la cultura en el desarrollo. El modelo ha mutado hacia una comunicación continuación el cliente, basado en entregas cortas y un continuo alineamiento con las necesidades reales. Esto ha provocado que el resultado se acerque más a las expectativas del cliente y se pueda cubrir cualquier desviación con mas agilidad.

Por otra parte tenemos que garantizar que cuando implantemos nuestro software sea de la manera más segura y colaborativa posible, garantizando que todos los miembros del equipo pueden escribir su código y que tenemos disponible un control de versiones, un entorno seguro de desarrollo, etc

1.- PUESTA SEGURA EN PRODUCCIÓN Y DEVOPS.**Caso práctico**

Julián acaba de entrar en el departamento DevOps y necesita entender esta metodología. Entiende que la puesta segura en producción del código que producen los desarrolladores ha evolucionado mucho durante los últimos años, ahora los desarrollos son en equipo (es decir, todos los desarrolladores conocen en tiempo real el desarrollo exacto que está haciendo el compañero) y muy cercano al cliente. Esta metodología engloba un conjunto de prácticas que agrupan el desarrollo de software (Dev) y las operaciones de TI (Ops).

Además, para aprovechar al máximo la agilidad y la capacidad de respuesta de los enfoques de DevOps, la seguridad de la TI también debe desempeñar un papel integrado en el ciclo de vida completo de sus aplicaciones. Surge entonces DevSecOps (Dev+ Sec + Ops). Esta práctica implica pensar desde el principio en la seguridad de las aplicaciones y de la infraestructura. También implica automatizar algunos controles de seguridad para impedir que se ralentice el flujo de trabajo de DevOps.

DevOps no es más que una metodología basada en la creación de software de forma continua y de mayor calidad mediante versiones más frecuentes. Esto se traduce de forma directa en claras ventajas:

- Las versiones más frecuentes hacen que se pueda alinear la expectativa del cliente de forma más sencilla.
- Cualquier cambio en el proyecto se puede corregir con más facilidad.
- El software tiene más calidad al estar enfocado en objetivos más concretos.

El enfoque DevOps abarca varias fases clave a lo largo del ciclo de vida del desarrollo de software, siendo esencial la automatización en todas estas fases para lograr una entrega continua y eficiente. Para cada una de las fases se utilizan multitud de herramientas. A continuación, se muestra un ejemplo de fases y de herramientas:

Fases típicas

Unas fases típicas (puede variar según las necesidades específicas del proyecto y las preferencias del equipo) son:

- Planificación (Plan en inglés): es donde se establecen los sprints (término que se verá en el siguiente apartado sobre Agile) y se asignan tareas a los desarrolladores.

- Codificación (Code en inglés): es donde se desarrolla el código y se hacen pruebas unitarias.

- Construcción (Build en inglés): en la que se utilizan herramientas de automatización para crear los artefactos de lanzamiento. Los artefactos son una copia aislada del código compilado o "configurado" de otro modo que puede desplegarse en los entornos necesarios para pruebas o producción.

- Pruebas (Test en inglés): la nueva versión del software se somete a una serie de pruebas para comprobar si está lista para la producción.

- Lanzamiento (Release en inglés): el objetivo principal es garantizar que la infraestructura y el entorno estén preparados para lanzar con éxito el software actualizado.

- Despliegue (Deploy en inglés): en esta fase el equipo de operaciones se coordinará para copiar los artefactos de lanzamiento que se han creado y probado en los servidores de producción.

- Funcionamiento (Operate, en inglés): en esta fase el equipo de operaciones debe garantizar que los componentes de la infraestructura permanezcan en línea y crezcan con la demanda según sea necesario.

- Monitorización (Monitor en inglés): la recopilación y análisis de los datos analíticos generados desde el propio software (conexiones activas, registro de accesos, errores, ...), junto con los comentarios de los usuarios y la revisión de los procesos permite garantizar el buen funcionamiento del mismo y realizar mejoras en el mismo.

Herramientas típicas

En lugar de una sola herramienta de DevOps existen conjuntos ("toolchains" en inglés) de múltiples herramientas. Como ejemplo (hay que tener en cuenta que hay muchas otras opciones disponibles y la elección de herramientas puede variar según las necesidades específicas del proyecto y las preferencias del equipo) de herramientas típicas:

Control de versiones	Git
Automatización de la construcción	Jenkins
Pruebas	Selenium
Gestión de la configuración	Terraform
Implementación	Kubernetes
Monitorización	ELK Stack

Una metodología efectiva de DevOps garantiza ciclos de desarrollo rápidos y frecuentes (a veces de semanas o días), pero debe integrar la seguridad en todas las fases para que además sea segura. Surge así DevSecOps: Desarrollo (Dev) + Seguridad (Sec) + Operaciones (Ops). Este enfoque no solo requiere incorporar herramientas nuevas, sino también adoptar un enfoque empresarial distinto. Los equipos de DevOps deben tener esto en cuenta al automatizar la seguridad para proteger el entorno y los datos por completo, así como el proceso de integración y distribución continuas. Se busca fomentar una cultura colaborativa donde los equipos trabajan juntos para identificar y abordar proactivamente vulnerabilidades, permitiendo la entrega rápida y segura de software en un entorno dinámico y en constante evolución.

Debes conocer: para ampliar más sobre los conceptos de DevOps y DevSecOps te animamos a visitar enlace de Red Hat Para saber más: durante los últimos años ha surgido junto a la metodología de DevOps nuevas maneras de gestionar proyectos y entregas con clientes basadas en metodologías ágiles (agile en inglés). Si bien ambas tienen algunos principios similares y pueden ser complementarias es conveniente saber las diferencias y puntos de convergencia. Te dejamos este enlace Nueva ventana si quieres saber más sobre el tema.

1.1.- DESARROLLO ÁGIL.

Como comentábamos la manera de desarrollar software ha cambiado, necesitamos adaptarnos de forma sencilla ante los cambios del cliente y para poder hacerlo surgen las metodologías ágiles en el desarrollo. Estas metodologías se basan en una serie de principios básicos:

1. Satisfacción del cliente mediante la entrega temprana y continua de software de calidad.
2. Gran aceptación y respuesta al cambio de los requisitos definidos durante el proyecto.
3. Entrega frecuente de software funcional, periodos 2-6 semanas.
4. Trabajo conjunto entre el cliente y desarrolladores.
5. Creación de equipos motivados y respaldados.
6. Comunicación cara a cara desarrolladores-cliente.
7. Equipos auto-organizados.
8. Reflexión continua de formas para mejorar la efectividad y realizar los ajustes necesarios para corregir cualquier desviación.

De entre todas ellas surge Scrum, una metodología ágil que engloba una serie de principios que ayudan a los equipos a desarrollar productos en periodos cortos (llamados sprints), permitiendo obtener de forma rápida feedback, adaptaciones y una mejora continua.

Dentro de los Sprints también se hacen ciclos de verificación o bucles, y simulación de fallos. Estos bucles ayudan a verificar la calidad del software y tomar las medidas oportunas sino fuese así, ayudando a mejorar la calidad del software. Además, estos bucles retroalimentan el proceso, mejorando la calidad del producto final.

Debes conocer: existe un manifiesto que marca los principios fundamentales de la metodología ágil en el desarrollo de software. No especifica el cómo, pero si define los principios fundamentales. Puedes consultarlo aquí Nueva ventana.

Para saber más: uno de los bucles o ciclos más famosos es el ciclo Deming, que llevó a Toyota a convertirse en uno de los gigantes de la industria del automóvil y a Japón ser un referente de la industria. Se utiliza también dentro de la metodología Scrum como un proceso de mejora continua. Puedes consultar más info aquí Nueva ventana.

2.- CONTROL DE VERSIONES.

Caso práctico

Julían entiende que uno de los principales cambios que surgen con la adopción de la metodología de DevOps es el continuo desarrollo de software mediante lanzamientos o releases continuos. Si a todo este proceso de nuevas versiones y lanzamientos unimos a varios miembros del equipo editando, codificando y generando cambios en las distintas piezas del software, hace que sea fundamental poder disponer de un modelo automático de control de versiones y cambios.

Los equipos de desarrollo trabajan con sistemas de control de versiones (o VCS, por sus siglas en inglés), que son herramientas software que gestionan, administran y controlan (creación, adición, eliminación, modificación del código) sobre un repositorio común.

Este repositorio está disponible para todos los desarrolladores y ofrece una manera colaborativa de trabajar donde todos los programadores pueden tener acceso al código y conocer qué cambios se han hecho, quién y cuándo los ha hecho y el estado actual de la versión. También proporciona una manera sencilla de tener un backup del código y recuperarnos de algún cambio erróneo o problema. Este repositorio es el resultado visible de lo que se conoce como Integración continua.

La integración continua es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas.

Las principales ventajas del Control de Versiones son:

- Revertir y corregir cambios de manera sencilla.
- Copia de seguridad o backup del código.
- Resolución de conflictos cuando uno o más miembros modifiquen el mismo fichero a la vez.

Algunos ejemplos de herramientas populares para el control de versiones son:

- Git: Permite a los equipos de desarrollo llevar un registro de los cambios en el código fuente a lo largo del tiempo, trabajar en ramas de manera eficiente y colaborar en proyectos de manera efectiva. GitHub y GitLab son plataformas populares que utilizan Git para gestionar repositorios de código.

- Subversion (SVN): Aunque ha perdido algo de popularidad frente a sistemas distribuidos como Git, SVN sigue siendo una herramienta de control de versiones centralizada muy utilizada. Permite a los desarrolladores mantener un historial de versiones y realizar seguimiento de cambios en archivos y directorios.

Debes conocer

La integración continua está estrechamente relacionada con el control de versiones; los desarrolladores envían los cambios sobre el software de forma periódica al repositorio compartido con un sistema de control de versiones. La combinación de ambas soluciones proporciona la integración continua proporcionada por un repositorio común que se mantiene actualizado y, además, el control de versión asegura tener todos los cambios controlados.

Uno de los sistemas de control de versiones más famosos y que se ha convertido en el estándar es Git, es un sistema de control de versiones distribuido usado por los desarrolladores de software. Tienes más información en este enlace de wikipedia.[Nueva ventana](#)

En Git hay tres etapas primarias (condiciones) en las cuales un archivo puede estar: estado modificado, estado preparado, o estado confirmado.

Mediante Git podremos clonar un repositorio o incluso si queremos probar el software podremos construirlo (build) de manera automática usando el mismo código que han usado los desarrolladores para crearlo.

No hay que confundir Git con Github, esta última es una plataforma basada en la web donde los usuarios pueden alojar repositorios Git. Facilita compartir y colaborar fácilmente en proyectos con cualquier persona en cualquier momento.

3.- SISTEMAS DE AUTOMATIZACIÓN DE CONSTRUCCIÓN

Caso práctico

Julían se ha dado cuenta que la automatización es una de las principales claves en DevOps, no sólo por la reducción de tiempos, sino que también minimizan el riesgo de errores. En el caso de la entrega y despliegue continuo (CI/CD) ha encontrado varias herramientas que permiten automatizar esta tarea y va a realizar unas pruebas con cada una para ver cuál es la mejor que se adapta a los proyectos con los que trabaja.

La entrega y despliegue continuo (CI/CD) son prácticas fundamentales en el ámbito de DevOps que buscan optimizar y acelerar el ciclo de vida del desarrollo de software. La entrega continua implica la automatización de los procesos de construcción, prueba y empaquetado, garantizando que el código probado y funcional esté siempre disponible para ser desplegado. Por otro lado, el despliegue continuo lleva esta automatización un paso más allá al liberar automáticamente las nuevas versiones del software en entornos de producción después de pasar las pruebas pertinentes.

Existen diversas herramientas para automatizar las tareas CI/CD, como la compilación, prueba y empaquetado de aplicaciones, proporcionando flujos de trabajo consistentes y repetibles. Al eliminar la intervención manual en estos procesos, los sistemas de automatización de la construcción contribuyen a la reducción de errores, la aceleración del tiempo de entrega y la garantía de la coherencia en todo el ciclo de vida del desarrollo de software. Estas soluciones permiten a los equipos centrarse en la innovación y el desarrollo, mientras aseguran la entrega de software de alta calidad de manera eficiente. Veamos algunas de las más utilizadas:

- **Jenkins**

Jenkins es una plataforma de automatización de código abierto diseñada para facilitar la integración continua y la implementación continua (CI/CD). Funciona como un servidor de automatización que permite a los equipos definir, programar y ejecutar flujos de trabajo automatizados, desde la compilación y prueba hasta la implementación en entornos de producción. Utilizando una arquitectura extensible y una amplia gama de complementos, Jenkins se integra fácilmente con diversas herramientas y servicios, permitiendo la orquestación de procesos complejos. Los conceptos clave de Jenkins incluyen la creación de trabajos, la gestión de pipelines, y la posibilidad de monitorear y visualizar el estado de los procesos de construcción y despliegue.

- **Circle Ci**

Circle CI es una plataforma de integración continua y entrega continua (CI/CD) basada en la nube. Permite a los equipos de desarrollo automatizar el proceso de construcción, prueba y despliegue de aplicaciones en entornos diversos. CircleCI utiliza pipelines configurables para definir flujos de trabajo personalizados, lo que facilita la orquestación de tareas complejas. Con soporte para diversos lenguajes de programación y frameworks, CircleCI se integra fácilmente con repositorios de código alojados en plataformas como GitHub y Bitbucket. Además, su enfoque en la escalabilidad y la velocidad lo convierte en una opción popular para proyectos de diferentes tamaños, permitiendo a los equipos acelerar el ciclo de vida del desarrollo y mantener la calidad del software a lo largo del tiempo.

- **Github Actions**

GitHub Actions es una plataforma de automatización integrada directamente en el entorno de GitHub, diseñada para facilitar la automatización de flujos de trabajo en el desarrollo de software. Permite a los desarrolladores definir, personalizar y compartir flujos de trabajo que abarcan desde la integración continua hasta la implementación continua.

Debes conocer: En el contexto de CI/CD se utiliza habitualmente el término "pipeline". Es una representación visual y automatizada del flujo de trabajo que sigue el código desde su desarrollo hasta su implementación. Se compone de una serie de pasos interconectados que definen las fases del ciclo de vida del software, como la compilación, las pruebas y el despliegue. OWASP tiene un proyecto para securizar los pipeline.

4.- INTEGRACIÓN CONTINUA Y AUTOMATIZACIÓN DE PRUEBAS.

Caso práctico

Al utilizar las herramientas de automatización de la construcción CI/CD Julián se ha dado cuenta que las pruebas son la clave fundamental para la seguridad, pero estas deben ser automatizadas para no ralentizar los procesos y evitar errores. Ha decidido probar varias herramientas de automatización de pruebas, tanto funcionales como de seguridad, para comprender mejor su funcionamiento, sus ventajas y su problemática.

La automatización de pruebas en CI/CD juega un papel crucial en la mejora de la calidad del software y la eficiencia del proceso de desarrollo. Esta práctica implica la creación y ejecución automatizada de pruebas a lo largo del ciclo de vida del software, desde la integración continua hasta la entrega continua. Al incorporar pruebas automatizadas en el pipeline de CI/CD, los equipos pueden identificar rápidamente posibles problemas y garantizar que las nuevas actualizaciones no introduzcan regresiones.

Las pruebas automatizadas pueden abarcar desde pruebas unitarias hasta pruebas de integración y pruebas de extremo a extremo, proporcionando una cobertura integral. La automatización de pruebas no solo acelera el proceso de desarrollo, sino que también contribuye a la confiabilidad y estabilidad del software al garantizar que los errores se identifiquen y aborden de manera proactiva en cada etapa del ciclo de vida del desarrollo.

Veamos algunos ejemplos típicos de herramientas para automatizar pruebas en función del tipo:

- **Pruebas unitarias**

Ya las vimos en la primera unidad en detalle. Estas pruebas están diseñadas para validar que cada componente del software realiza sus tareas de manera aislada y produce los resultados esperados. Estas pruebas no sólo se deben realizar durante el desarrollo, también deben realizarse en el CI/CD para asegurarse de que el software es correcto y seguro.

Ejemplos de herramientas para pruebas automatizadas unitarias: PHPUnit, JUnit, NUnit, pytest, Jasmine

- **Pruebas de Integración**

Recuerda que estas pruebas verifican que las unidades de código, previamente probadas de manera individual en las pruebas unitarias, funcionen correctamente cuando se combinan. El objetivo principal de las pruebas de integración es identificar posibles conflictos en la interfaz y asegurar que los distintos elementos del software trabajen de manera conjunta sin problemas.

Ejemplos de herramientas para pruebas automatizadas de integración: TestNG, Mocha, Protractor

- **Pruebas de Aceptación**

Recuerda que estas pruebas se centran en evaluar el comportamiento global de la aplicación desde la perspectiva del usuario final, verificando que todas las funciones y características se desempeñen según lo esperado. Al integrar pruebas de aceptación en un pipeline de CI/CD, los equipos pueden asegurarse de que las nuevas implementaciones no solo sean técnicamente sólidas, sino también que cumplan con las expectativas del usuario. Piensa en este testing como un robot que usa nuestra aplicación, simulando acciones y peticiones realizadas desde un navegador.

Ejemplos de herramientas para pruebas automatizadas de aceptación: Selenium WebDriver, Cucumber, Cypress

- **Pruebas de Seguridad**

Estas pruebas buscan identificar y mitigar posibles vulnerabilidades y riesgos de seguridad en el software. Al incorporar pruebas de seguridad en el pipeline de CI/CD, los equipos pueden automatizar la evaluación de posibles amenazas, como vulnerabilidades conocidas, configuraciones inseguras o prácticas de codificación deficientes.

Ejemplos de herramientas para pruebas automatizadas de seguridad: OWASP ZAP, Burp Suite, Nessus, Acunetix, SonarQube, Docker scan

La elección de una herramienta específica puede depender de los requisitos del proyecto, las preferencias del equipo y otros factores específicos.

5.- VIRTUALIZACIÓN Y CONTENEDORES.

Caso práctico

Uno de los problemas que se presentan durante el desarrollo y despliegue de aplicaciones es la heterogeneidad entre los sistemas de desarrollo, no ya entre los propios desarrolladores, también entre los sistemas de desarrollo, entornos de pruebas, preproducción y producción. Los sistemas donde se crea y se prueba el software tienen librerías y sistemas operativos diferentes. Esto provoca que la aplicación o software pudiera funcionar bien con unas determinadas versiones de librerías o sistemas operativos, pero no en otros, o ejecutarse bien con la versión de un lenguaje, pero no con otra. Para asegurar la calidad de desarrollo tenemos que asegurar que todo el equipo de desarrollo usa las mismas versiones de todas las aplicaciones y librerías necesarios.

Esto es más complicado de lo que parece, porque hay desarrolladores que prefieren una distribución concreta, o incluso sistemas privativos. Incluso los sistemas de pruebas, preproducción y producción suelen ser distintos. Los sistemas de producción suelen ser más potentes y los básicos se dejan para pruebas y preproducción. Otro problema derivado es que los equipos desarrollo a veces están involucrados en varios

proyectos a la vez y cada uno de estos proyectos requiera versiones distintas de librerías, complicándolo aún más. De hecho, podría pasar que cambios que se hagan en una librería que afecta a un proyecto acabe afectando sin querer a otro proyecto, suponiendo un gran problema. Para solucionar estos problemas es importante que los desarrolladores creen el código en entornos controlados y fácilmente replicables. Julián siempre ha trabajado con máquinas virtuales, pero en el nuevo proyecto se tiene que enfrentar al despliegue de una aplicación utilizando contenedores.

Las máquinas virtuales (VMs de sus siglas en inglés) y los contenedores son dos tecnologías de virtualización que comparten el objetivo común de optimizar la eficiencia y flexibilidad en el despliegue de aplicaciones, pero difieren en sus enfoques y características fundamentales.

Las máquinas virtuales son entornos completamente aislados que emulan sistemas operativos completos dentro de un hipervisor. Cada VM incluye un sistema operativo propio, así como recursos de sistema dedicados. Aunque proporcionan un alto nivel de aislamiento y permiten ejecutar aplicaciones en entornos distintos, el enfoque de las VMs a menudo implica un mayor consumo de recursos y tiempos de inicio más lentos debido a la necesidad de cargar un sistema operativo completo. Por otro lado, los contenedores son entornos ligeros que comparten el mismo núcleo (kernel) del sistema operativo subyacente y, por lo tanto, son más eficientes en términos de recursos y tiempos de inicio. Los contenedores encapsulan solo las bibliotecas y dependencias necesarias para ejecutar una aplicación específica, lo que permite una rápida implementación y escalabilidad. Se pueden exportar y usar en cuestión de segundos y hace que todo el equipo de desarrollo trabaje con las mismas herramientas, y en el caso de que estén en varios proyectos usar contenedores distintos.

La elección entre máquinas virtuales y contenedores depende de los requisitos específicos del proyecto. Las VMs son ideales para situaciones que requieren un aislamiento más estricto, como la ejecución de aplicaciones con sistemas operativos diferentes, mientras que los contenedores son más adecuados para implementaciones ágiles y escalables donde la eficiencia de recursos es primordial. Ejemplos comunes de plataformas de máquinas virtuales incluyen VMware, KVM y VirtualBox, mientras que Docker es una herramienta popular para la gestión de contenedores.

Veamos un poco más en detalle Docker:

- Se basa en el uso de imágenes. Una imagen es esencialmente una representación ligera y portátil de un sistema de archivos que incluye la aplicación y sus dependencias.
- Un contenedor es una imagen ejecutándose en un sistema. Por ejemplo, una imagen de Debian con Apache Tomcat + Mysql. Dicha imagen se puede lanzar en un sistema Ubuntu creando un contenedor. Dicho contenedor podrá tener su propia dirección IP y compartirá los recursos hardware con el resto de aplicaciones que se están ejecutando en el sistema (al lanzar el contenedor también se pueden fijar límites de uso de memoria, CPU, ...).
- Docker tiene un registro central con todo tipo de imágenes (Docker Hub). Además, cualquier imagen o incluso un contenedor ejecutándose puede convertirse en una nueva imagen.
- Docker ofrece herramientas de gestión de cluster con Swarm y Compose.
- La arquitectura de docker está compuesta de:
 - El cliente de Docker (Docker Client en inglés) es la principal interfaz de usuario para Docker. Acepta comandos del usuario y se comunica con el demonio Docker.
 - El demonio Docker (Docker Engine en inglés) corre en una máquina anfitriona (host). El usuario no interactúa directamente con el demonio, en su lugar lo hace a través del cliente Docker.
 - El demonio Docker levanta los contenedores haciendo uso de las imágenes, que pueden estar en local o en el Docker Registry (repositorio de imágenes).

Debes conocer: puedes conocer más sobre la tecnología Docker y sus numerosos beneficios

6.- GESTIÓN AUTOMATIZADA DE CONFIGURACIÓN DE SISTEMAS

Caso práctico

En organizaciones con gran cantidad de máquinas (físicas o virtuales) uno de los mayores desafíos es la tarea de automatizar los procesos para preparar la infraestructura, gestionar la configuración, implementar las aplicaciones y organizar los sistemas, mejorar la seguridad y el cumplimiento, ejecutar parches en los sistemas y compartir la automatización en toda la empresa, entre otros procedimientos de TI.

En el proyecto en el que está trabajando Julián hay demasiadas máquinas que gestionar, por lo que se ha puesto a investigar herramientas para gestionar la configuración. Estas herramientas se basan en archivos de configuración que se deben versionar y almacenar en repositorios de control de versiones como Git.

Las herramientas de gestión automatizada de configuración de sistemas aseguran la consistencia y la integridad de la configuración a lo largo del tiempo, facilitan la automatización de tareas repetitivas y simplifican la implementación y actualización de software.

Estas plataformas permiten a los equipos definir, gestionar y aplicar configuraciones de manera centralizada, lo que resulta esencial para mantener entornos de infraestructura complejos de manera eficiente y segura. Al utilizar estas herramientas, los equipos pueden garantizar que los sistemas se configuren de manera coherente, reduciendo el riesgo de errores y mejorando la seguridad y la estabilidad del entorno operativo.

Veamos algunas de estas herramientas:

- **Ansible**

Ansible es una plataforma de automatización de código abierto que se centra en la gestión de configuración, la implementación de aplicaciones y la automatización de tareas. Desarrollado en Python, Ansible utiliza un enfoque basado en la infraestructura como código (IaC), lo que significa que las configuraciones y tareas se definen en archivos de configuración. A través de un modelo de ejecución basado en SSH, Ansible permite a los usuarios especificar cómo deberían estar configurados los sistemas y servicios, y luego aplica esas configuraciones de manera consistente a través de todos los nodos del sistema. Su diseño sin agente (también admite con agente pero no es la opción habitual) y su arquitectura modular lo hacen altamente escalable y fácil de implementar.

- **Puppet**

Puppet es una herramienta de gestión de configuración de código abierto diseñada para automatizar y gestionar la configuración de sistemas informáticos. Utiliza un enfoque declarativo, donde los administradores definen el estado deseado de los sistemas, y Puppet se encarga de llevarlos a ese estado de manera coherente. Escrito en Ruby, Puppet ofrece un lenguaje específico del dominio (DSL) para describir configuraciones y políticas. Utilizando un modelo cliente-servidor, los nodos gestionados por Puppet se conectan a un servidor maestro que distribuye y aplica las configuraciones definidas. Puppet es ampliamente utilizado para gestionar la infraestructura, desde la configuración de servidores hasta la instalación de software y la gestión de recursos de red.

- **Chef**

Chef es una herramienta de automatización de TI que se enfoca en la gestión de configuración y la automatización de la infraestructura. Utilizando un enfoque basado en código, Chef permite a los administradores describir la configuración deseada de los sistemas en forma de "recetas" y "roles". Estas recetas definen cómo deben configurarse y qué software debe estar instalado en cada nodo del sistema. Chef opera en un modelo cliente-servidor, donde los nodos gestionados se conectan a un servidor Chef para obtener y aplicar las configuraciones definidas. Chef destaca por su flexibilidad y su capacidad para gestionar tanto entornos pequeños como grandes a escala.

7. Herramientas de simulación de fallos

Caso práctico

La simulación de fallos es una práctica esencial en el ámbito de la ingeniería de software y la gestión de sistemas que consiste en reproducir condiciones adversas de manera controlada para evaluar la resistencia y la capacidad de recuperación de un sistema. Esta técnica permite probar la robustez de una aplicación o infraestructura ante situaciones inesperadas, como fallos de hardware, caídas de red o pérdida de datos. Al simular fallos de manera deliberada, los equipos pueden identificar vulnerabilidades potenciales, mejorar la resiliencia y desarrollar estrategias efectivas de recuperación.

En el nuevo proyecto le han encargado a Julián investigar sobre herramientas de simulación de fallos

Herramientas y prácticas como "Chaos Engineering" se han vuelto populares para implementar de manera controlada y gradual la simulación de fallos, permitiendo a las organizaciones fortalecer sus sistemas y servicios en condiciones adversas simuladas antes de enfrentar eventos reales no planificados. Este enfoque proactivo contribuye a la mejora continua de la fiabilidad y la disponibilidad de los sistemas críticos.

Un ejemplo paradigmático de esta metodología es la implementación de pruebas de caos, como las conducidas por herramientas como Chaos Monkey en entornos basados en microservicios. Estas pruebas introducen deliberadamente fallos en nodos o servicios en producción para evaluar la capacidad del sistema para resistir y recuperarse de eventos inesperados. La simulación puede incluir, por ejemplo, la desconexión de nodos, la introducción de retrasos en la red o la generación de errores en la capa de aplicación. Al exponer un sistema a escenarios de fallos controlados, los equipos pueden descubrir debilidades potenciales, optimizar la recuperación automática y, en última instancia, mejorar la resiliencia general del sistema frente a eventos disruptivos del mundo real.

Para saber más: si quieres ampliar sobre Chaos Monkey te recomiendo visitar el proyecto de Netflix en este enlace [Nueva ventana](#)

8.- ORQUESTACIÓN DE CONTENEDORES.

Caso práctico

Un orquestador asume la responsabilidad de coordinar y automatizar el despliegue, configuración y gestión de servicios y aplicaciones en una arquitectura distribuida. Estos orquestadores facilitan la escalabilidad, la recuperación de fallos y la gestión eficiente de recursos al proporcionar una visión centralizada y coherente del estado de la infraestructura. Al permitir la automatización de tareas complejas y secuencias de trabajo, los orquestadores juegan un papel clave en la optimización de la eficiencia operativa y en la implementación efectiva de sistemas distribuidos en entornos empresariales modernos.

En el proyecto actual Julián tiene que desplegar una aplicación web en un orquestador Kubernetes en un entorno cloud.

Un orquestador es una herramienta o plataforma que se encarga de coordinar y gestionar la ejecución de procesos o servicios de manera eficiente y automatizada en un entorno computacional. Los orquestadores son esenciales en entornos complejos y distribuidos, donde múltiples componentes deben trabajar juntos de manera coordinada. Hay varios tipos de orquestadores, cada uno diseñado para abordar necesidades específicas en diferentes contextos. En la imagen de la derecha puedes ver el esquema de funcionamiento de uno de ellos.

A continuación se presentan algunos tipos comunes:

- **Orquestador de Infraestructura**

Los orquestadores de infraestructura, como Terraform y AWS CloudFormation, se centran en la provisión y gestión de recursos de infraestructura en la nube, como servidores virtuales, redes, bases de datos y otros servicios.

Trabajan a un nivel más bajo, proporcionando una capa de abstracción sobre los recursos físicos y permitiendo la definición y gestión de la infraestructura como código (IaC).

Se centran en máquinas virtuales, almacenamiento y redes.

- **Orquestador de Contenedores**

Los orquestadores de contenedores, como Kubernetes y Docker Swarm, se diseñaron para gestionar el ciclo de vida de aplicaciones containerizadas, desde la implementación y escalado hasta la administración de recursos y la orquestación de servicios.

Operan en un nivel más alto, enfocándose en contenedores y aplicaciones, ofreciendo abstracciones que facilitan la gestión y la orquestación de múltiples instancias de aplicaciones.

Están especializados en gestionar el despliegue y la escalabilidad de contenedores, proporcionando funcionalidades específicas para la orquestación eficiente de aplicaciones containerizadas.

Además, suelen realizar las siguientes tareas:

- Búsqueda de servicios (Service Discovery en inglés) que permite acceder a los distintos contenedores del cluster.
- Balanceo de carga.
- Configuración de la red.
- Escalabilidad.
- Registro (Logging) y Monitoreo.
- Respuesta a fallos.

Ambos tipos de orquestadores son herramientas que desempeñan roles distintos, pero a menudo se utilizan de manera complementaria en entornos de desarrollo y operaciones.

Autoevaluación I

Identifica si las siguientes frases son verdaderas o falsas

- 1- DevOps es un nuevo estándar de programación.
 - a) Verdadero
 - b) Falso
- 2- Hoy en día los desarrolladores funcionan de forma aislada.
 - a) Verdadero
 - b) Falso
- 3- DevOps ha cambiado la manera en que se entrega el software al cliente.
 - a) Verdadero
 - b) Falso
- 4- Para que todos los desarrolladores trabajen de forma colaborativa y eficiente surgen los repositorios.
 - a) Verdadero
 - b) Falso

TEST I

- 1- Git es un ejemplo de sistema de control de versiones. ¿Verdadero o Falso?
 - a) Verdadero
 - b) Falso
- 2- DevOps se basa en la colaboración. ¿Verdadero o Falso?
 - a) Verdadero
 - b) Falso
- 3- Kubernetes aporta una capa de control y orquestación de contenedores. ¿Verdadero o Falso?
 - a) Verdadero
 - b) Falso
- 4- Los equipos de desarrollo almacenan su software en local. ¿Verdadero o falso?
 - a) Verdadero
 - b) Falso
- 5- Un repositorio es la imagen visible de:
 - a. El desarrollo con Kubernetes.
 - b. Estándares ISO de desarrollo seguro.
 - c. El desarrollo en Docker.
 - d. Integración continua.

- 6- ¿Cuáles son los estados de Git?:
- Contenido y desempaquetado.
 - Modificado, Preparado y Confirmado.
 - Modificado, Preparado y Solucionado.
 - Parcheado y corregido.
- 7- El control de versiones proporciona:
- Gestión del cambio.
 - Roles.
 - Control de permisos.
 - Todas las anteriores.
- 8- Git y Github son y equivalen a lo mismo. ¿Verdadero o falso?
- Verdadero
 - Falso
- 9- ¿El ciclo de desarrollo de software ha cambiado en los últimos años?:
- Si, ahora se potencia más la individualidad del desarrollador.
 - Sí, el trabajo colaborativo y la continua revisión son claves.
 - No, el modelo empleado era y es perfectamente válido.
 - No.
- 10- Kubernetes aporta una capa de control y orquestación de contenedores. ¿Verdadero o Falso?
- Verdadero
 - Falso

TEST II

- 1-Github es una plataforma que aloja Gits de los usuarios. ¿Verdadero o Falso?
- Verdadero
 - Falso
- 2-Una máquina virtual es la representación de un contenedor. ¿Verdadero o falso?
- Verdadero
 - Falso
- 3- Docker es una tecnología de contenedores. ¿Verdadero o Falso?
- Verdadero
 - Falso
- 4- Subversión es una herramienta de control de versiones. ¿Verdadero o Falso?
- Verdadero
 - Falso
- 5-¿Qué factor ha cambiado la manera desarrollar?:
- Transformación tecnológica.
 - Número de vulnerabilidades.
 - Machine learning.
 - Trabajo colaborativo y entrega continua.
- 6-El control de versiones permite corregir cambios de manera sencilla. ¿Verdadero o Falso?
- Verdadero
 - Falso
- 7-Kubernetes es un sistema de orquestación de contenedores. ¿Verdadero o Falso?
- Verdadero
 - Falso
- 8- Las versiones de sistema operativo y librerías de nuestro entorno de desarrollo no influyen en el resultado. ¿Verdadero o falso?
- Verdadero
 - Falso
- 9-Github es una plataforma que aloja Gits de los usuarios. ¿Verdadero o Falso?
- Verdadero
 - Falso
- 10-¿Qué problemas tenemos cuando usamos muchos contendores distintos?:
- Tamaño contendores.
 - Control y escalabilidad.
 - Usuarios permitidos.
 - Software incluido.

Respuesta

Autoevaluación I: 1 b, 2 b), 3 a), 4 a)

TEST I: 1 a), 2 a), 3 a), 4 b), 5 d), 6 b), 7 a), 8 b), 9 b), 10 a)

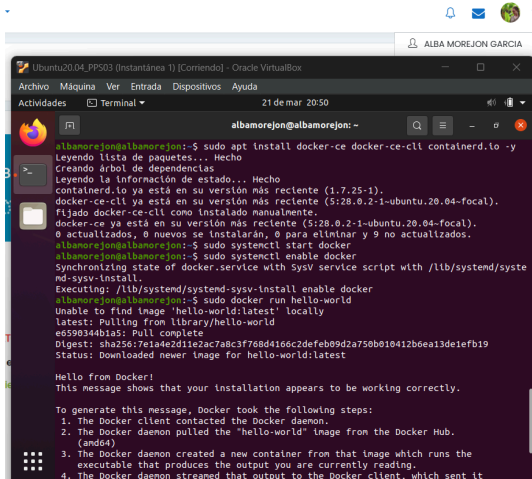
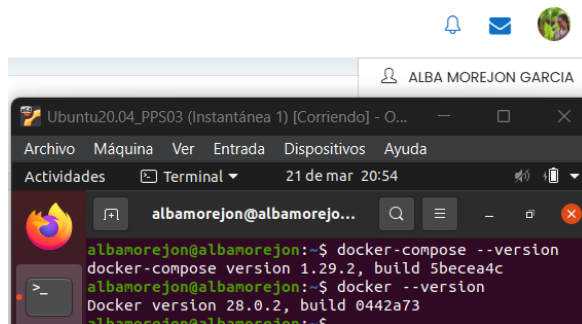
TEST II: 1 a), 2 b), 3 a), 4 a), 5 d), 6 a), 7 a), 8 b), 9 a), 10 b)

Caso práctico

Docker Logo, representa una ballena azul con contenedores al lado del nombre **dotCloud, Inc.. Docker Logo (Apache License 2.0)**
Julián se ha unido a un grupo de trabajo para la creación de un aplicativo web para entornos web, sabe que sus compañeros están usando Docker y aún tiene cierto temor al uso de esta tecnología por desconocimiento por lo que va repasando cada paso que da.

Apartado 1: Manejo básico de contenedores con Docker

1- En las unidades 2 y 3 ya se ha utilizado Docker así que ya debes tenerlo instalado, en caso contrario debes instalarlo. Puedes instalarlo en tu equipo o crear una máquina virtual e instalarlo en la misma (en ese caso te recomendamos utilizar una máquina Ubuntu porque va a ser muy sencillo).

2- Rellena la siguiente tabla explicando que hacen las siguientes líneas

Comando/línea de comandos	Descripción detallada (si hay opciones se deben explicar)
<code>docker pull mysql:5.7.28</code>	Se utiliza para descargar una imagen específica de MySQL desde el repositorio de Docker a tu máquina local. <ul style="list-style-type: none"> • <code>docker pull</code>, descarga la imagen desde un repositorio de Docker (en este caso desde Docker Hub) • <code>mysql:5.7.28</code>, especifica la imagen de MySQL y la versión a descargar
<code>docker images</code>	Devuelve una lista de las imágenes de Docker que estén almacenadas en la máquina local <ul style="list-style-type: none"> • <code>docker</code>, comando que interactúa con el motor de Docker • <code>images</code>, lista las imágenes disponibles en la máquina
<code>docker run --rm --user mysql -d -p 33060:3306 --name mysql-db1 -e MYSQL_ROOT_PASSWORD=secret -v ./mysql:/var/lib/mysql mysql:5.7.28</code> (NOTA: si el equipo es windows se debe cambiar ./mysql por .\mysql)	Inicia un contenedor de Docker que ejecuta la instancia de MySQL versión 5.7.28, se realiza en segundo plano, con el usuario mysql (configurando su contraseña) utilizando los puertos 33060:3306 y volúmenes para persistir los datos y permitir el acceso a MySQL desde el host <ul style="list-style-type: none"> • <code>docker run</code>, inicia un nuevo contenedor a partir de una imagen Docker • <code>--rm</code>, elimina el contenedor cuando se detiene • <code>--user mysql</code>, ejecuta el contenedor con el usuario mysql • <code>-d</code> ejecuta el contenedor en segundo plano (modo detach) • <code>-p 33060:3306</code>, mapea el puerto 33060 del host al puerto 3306 del contenedor (permite acceder a MySQL en el puerto 3306 del host) • <code>--name mysql-db1</code>, da nombre al contenedor • <code>-e MYSQL_ROOT_PASSWORD=secret</code>, establece esa variable con el valor "secret" (configurando la contraseña del usuario root de MySQL) • <code>-v ./mysql:/var/lib/mysql</code>, monta el directorio ./mysql del host en el directorio /var/lib/mysql del contenedor (permite que los datos se almacenen en el host asegurando su persistencia). • <code>mysql:5-7-28</code>, especifica la imagen de MySQL y la versión a utilizar
<code>docker ps -a</code>	Se utiliza para listar todos los contenedores de Docker en la máquina incluyendo los detenidos, mostrando su estado y los detalles de cada uno. <ul style="list-style-type: none"> • <code>docker ps</code>, lista los contenedores que están actualmente en ejecución, mostrando información de los contenedores: ID, imagen, nombre, estado y puertos • <code>-a</code>, extiende la funcionalidad del comando para incluir todos los contenedores, los detenidos, en pausa o no iniciados también.
<code>docker logs mysql-db1</code>	Muestra los registros (logs) de un contenedor específico de Docker, permitiendo revisar la salida estándar y los mensajes de error generados. <ul style="list-style-type: none"> • <code>docker logs</code>, muestra los registros de un contenedor de Docker, la salida estándar y la salida de error. • <code>mysql-db1</code>, especifica el nombre del contenedor cuyos registros se quieren mostrar
<code>docker exec -it mysql-db1</code>	Ejecuta un comando dentro de un contenedor en ejecución, en este caso abre una sesión de shell dentro del

/bin/sh	<p>contenedor mysql-db1.</p> <ul style="list-style-type: none"> • docket exec, se ejecuta comando dentro del contenedor en ejecución • -i, mantiene la entrada abierta, permitiendo la interacción con el contenedor • -t, asigna una terminal pseudo-TTY, hace que la sesión sea interactiva y haga de máquina real. • mysql-db1, nombre del contenedor en el que se quiere ejecutar el comando • /bin/sh, comando que se ejecutará en el contenedor, en este caso se abre una shell permitiendo que interactúen con el sistema y ejecutar los comandos en el servidor
docker stop mysql-db1	<p>Sirve para detener el contenedor Docker llamado mysql-db1</p> <ul style="list-style-type: none"> • docker stop, detiene un contenedor en ejecución enviando una señal SIGTERM y después SIGKILL • mysql-db1, nombre del contenedor en el que se quiere detener
docker start mysql-db1	<p>Sirve para iniciar el contenedor Docker llamado mysql-db1</p> <ul style="list-style-type: none"> • docker start, inicia el contenedor • mysql-db1, nombre del contenedor en el que se quiere detener
docker rm -f mysql-db1	<p>Se utiliza para eliminar contenedores Docker (el contenedor debe haber sido detenido previamente) de forma forzada</p> <ul style="list-style-type: none"> • docker rm, elimina uno o varios contenedores de Docker. • -f, fuerza la eliminación del contenedor incluso si está en ejecución • mysql-db1, nombre del contenedor en el que se quiere eliminar

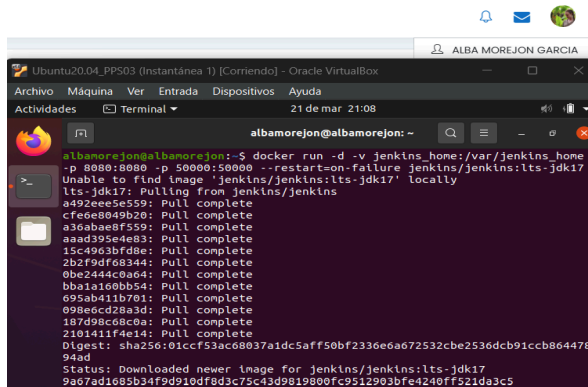
3- Realizar pruebas con Jenkins utilizando contenedores.

Jenkins es una herramienta de integración continua y entrega continua (CI/CD) de código abierto, escrita en Java, que permite automatizar el desarrollo, prueba y despliegue de software. Visita su página oficial para entenderlo mejor: <https://www.jenkins.io/> Incluye pantallazos de las pruebas que se piden a continuación, incluyendo las de la consola donde se lanzan las líneas de comandos docker y del navegador con la aplicación.

3.1- Lanzar el contenedor de Jenkins (<https://github.com/jenkinsci/docker/blob/master/README.md>). Indica la línea de comandos para realizar esta operación.

```
docker run -d -v jenkins_home:/var/jenkins_home -p 8080:8080 -p 50000:50000 --restart=on-failure jenkins/jenkins:lts-jdk17
```

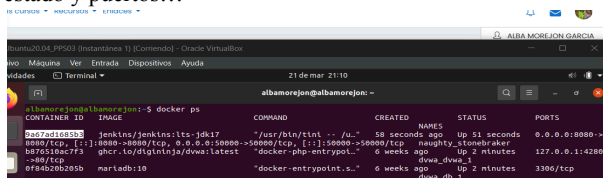
Este comando inicia un contenedor de Jenkins mapeando los puertos 8080 y 50000, y configura el contenedor para que se reinicie automáticamente en caso de fallo, asegura que los datos de Jenkins se mantengan persistentes y que el contenedor se ejecute en segundo plano



```
albamorejon@albamorejon: ~$ docker run -d -v jenkins_home:/var/jenkins_home
-p 8080:8080 -p 50000:50000 --restart=on-failure jenkins/jenkins:lts-jdk17
Unable to find image 'jenkins/jenkins:lts-jdk17' locally
lts-jdk17: Pulling from jenkinsci/jenkins
a492ee5e559: Pull complete
cfe68049b20: Pull complete
a36abae8f59: Pull complete
aadd395e4e0: Pull complete
15c4963b7d8e: Pull complete
2b2f9df68344: Pull complete
0be2444c0a64: Pull complete
0ba1a10bb054: Pull complete
695ab411b701: Pull complete
098e6cd28a3d: Pull complete
187d90c68c0a: Pull complete
2104111f4e14: Pull complete
Digest: sha256:01ccf53ac68037a1dc5aff50bf2336e6a672532cbe2536db91cccb864478
94ad
Status: Downloaded newer image for jenkins/jenkins:lts-jdk17
9a67ad1685b34f9d910df8d3c75c43d9819800fc9512903bfe4240ff521da3c5
```

3.2- Comprobar que el contenedor anterior esta funcionando. Indica la línea de comandos para realizar esta operación.

docker ps, lista los contenedores que están actualmente en ejecución, mostrando información de los contenedores: ID, imagen, nombre, estado y puertos...



```
albamorejon@albamorejon: ~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
7a572c108356   jenkins/jenkins:lts-jdk17          "/usr/bin/tini -- /u-    58 seconds ago Up 51 seconds 0.0.0.0:8080->
8080/tcp, 0.0.0.0:50000->50000/tcp, [::]:8080->[::]:50000/tcp   naughty_stonebraker
b87651eac7f3   ghcr.io/dglinja/dwallatest        "docker-php-entrypoint"  6 weeks ago   Up 2 minutes  127.0.0.1:4280
->80/tcp
ef84b20c2059   mariadb:10                        "docker-entrypoint.s-    6 weeks ago   Up 2 minutes  3306/tcp
dwall_db_1
```

3.3- Mostrar el log del contenedor de Jenkins. Indica la línea de comandos para realizar esta operación.

docker logs naughty_stonebraker, obtenemos los logs de un contenedor Docker en ejecución llamado naughty_stonebraker.

```

albamorejon@albamorejon:~$ docker logs naughty_stonebraker
Running from: /usr/share/jenkins/jenkins.war
webroot: /var/jenkins_home/war
2025-03-21 20:08:26.529+0000 [id=1] INFO    winstone.Logger#logInternal: Beginning extraction f
ron war file
2025-03-21 20:08:27.105+0000 [id=1] WARNING o.e.j.ee9.nested.ContextHandler#setContextPath: Emp
ty contextPath
2025-03-21 20:08:27.179+0000 [id=1] INFO    org.eclipse.jetty.server.Server#doStart: Jetty-12.0
.10; built: 2024-12-09T21:02:54.535Z; git: c3f88baf4e393f23204dc14dc57b642e84dec7; jvm 17.0.147
2025-03-21 20:08:27.670+0000 [id=1] INFO    o.e.j.e.w.StandardDescriptorProcessor#visitServlet:
NO JSP Support for /, did not find org.eclipse.jetty.ee9.jsp.JettyJspServlet
2025-03-21 20:08:27.712+0000 [id=1] INFO    o.e.j.s.DefaultSessionIdManager#doStart: Session wo
rkName=nodes
2025-03-21 20:08:28.083+0000 [id=1] INFO    hudson.WebAppMain#contextInitialized: Jenkins home
directory: /var/jenkins_home found at: EnvVars.masterEnvVars.get("JENKINS_HOME")
2025-03-21 20:08:28.248+0000 [id=1] INFO    o.e.j.s.handler.ContextHandler#doStart: Started oe
j.ee9.ContextHandler$CoreContextHandler@63411512 [Jenkins v2.492.2, /, basefile:///var/jenkins_home/war/, a
vailable, h=oej.ee9.ContextHandler$CoreContextHandler$CoreToNestedHandler@35cd68d4(STARTED)]
2025-03-21 20:08:28.274+0000 [id=1] INFO    o.e.j.s.server.AbstractConnector#doStart: Started Ser
verConnector@2105acc(HTTP/1.1, (http/1.1)) [0-0.0.0.0:8080]
2025-03-21 20:08:28.308+0000 [id=1] INFO    org.eclipse.jetty.server.Server#doStart: Started oe
j.s.Server@5b444398(STARTING)[12.0.10,sto=0] @2243ms
2025-03-21 20:08:28.316+0000 [id=24] INFO    winstone.Logger#logInternal: Winstone Servlet Engin

```

```

albamorejon@albamorejon:~$
*****
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

34ad0f9fe9de4ab7893ee773a4f4244f

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
*****

```

Please use the following password to proceed to installation:

34ad0f9fe9de4ab7893ee773a4f4244f

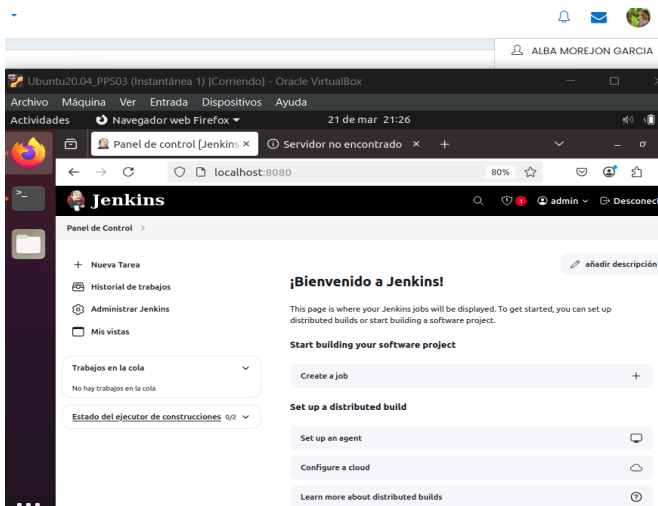
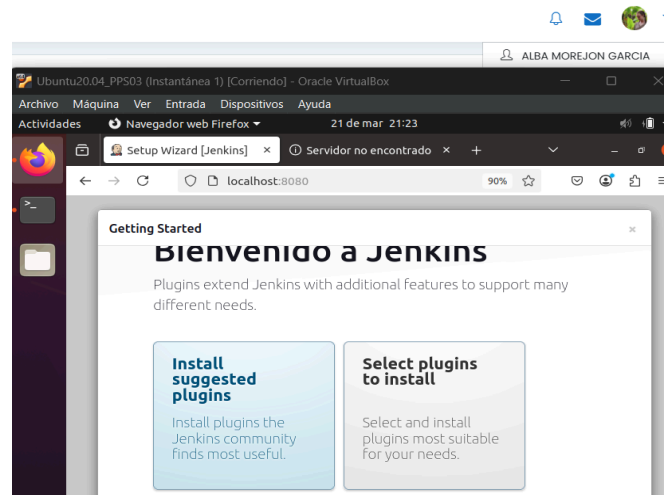
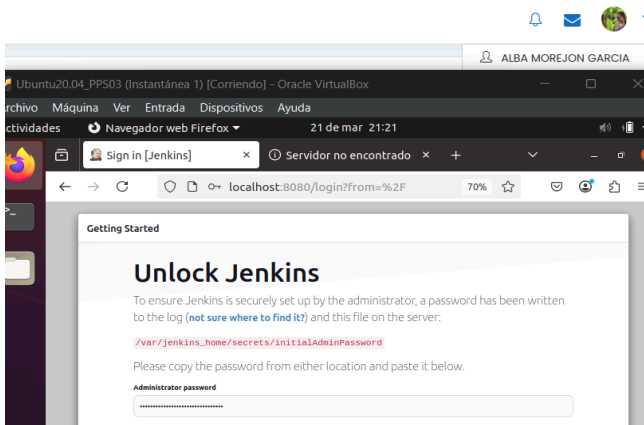
3.4- Acceder a aplicación Jenkins desde un navegador (<https://www.enkins.io/doc/book/installing/docker/>):

Abre un navegador con la dirección <http://localhost:8080> (si has puesto el 8080)

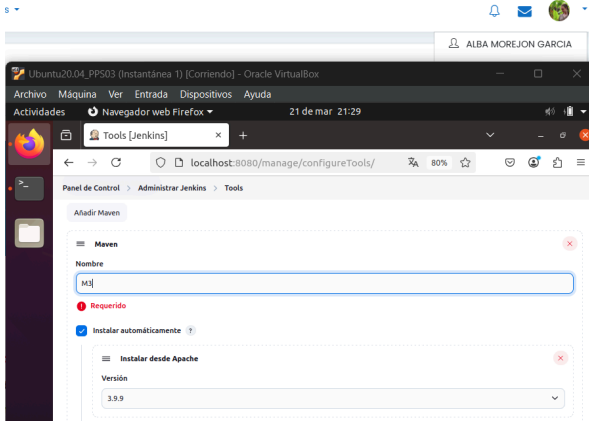
El token que pide aparece en el log del contenedor.

Escoge la opción de instalar los plugins recomendados.

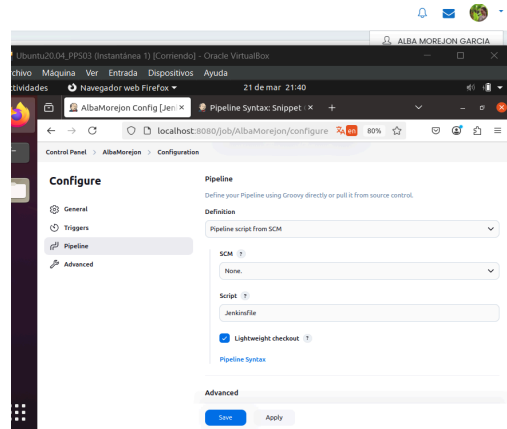
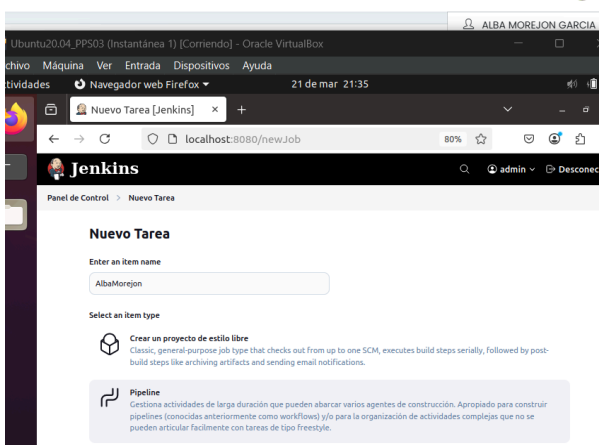
NO hace falta que crees un usuario (Skip and continue as admin)



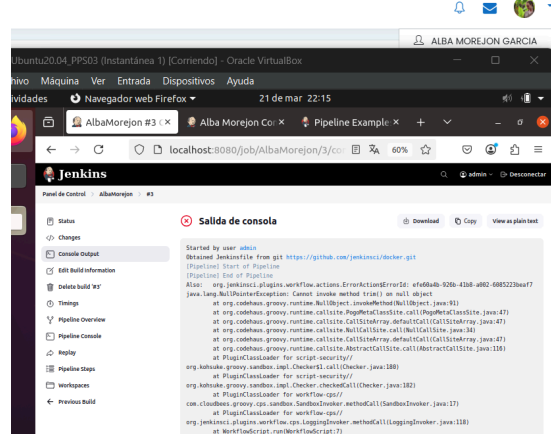
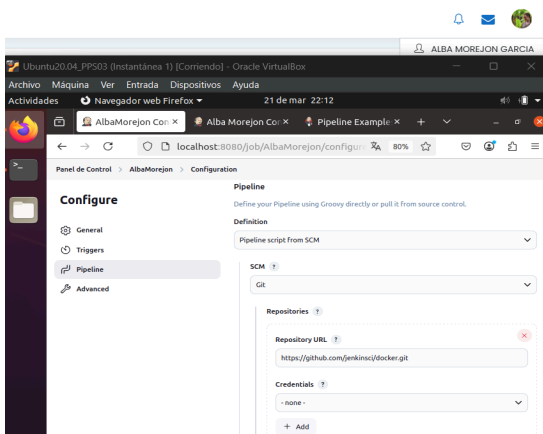
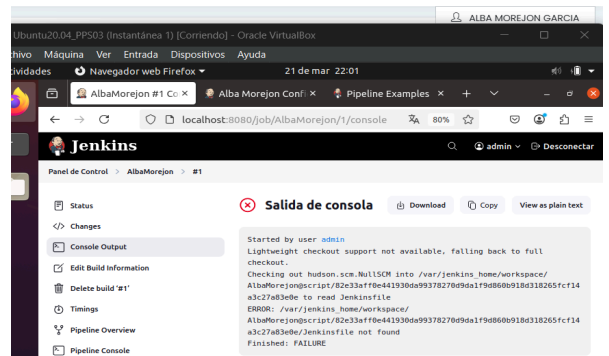
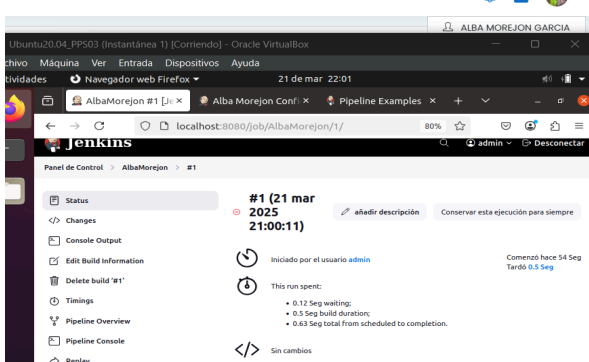
3.5- Instalar la tool de Maven con el nombre M3 dentro de Jenkins (Panel de control - Administrar Jenkins - Tools - Instalaciones de Maven - Añadir Maven - Nombre M3 y la opción de instalar automáticamente)



3.6- Crear un pipeline sencillo que se llame XXXXX (donde XXXXX es tu nombre); creas una tarea de tipo pipeline y en el apartado script seleccionas try sample Pipeline... escoges Github+ Maven. Copia el código del pipeline (a ese archivo se le denomina Jenkinsfile) en algún sitio porque te va a hacer falta para responder una pregunta en un apartado posterior



3.7- Ejecutar el pipeline anterior y mostrar el pantallazo de la salida.



3.8- Eliminar el contenedor de Jenkins. Indica la línea de comandos para realizar esta operación.

`docker rm -f naughty_stonebraker`, elimina contenedores docker (`docker rm`) forzando la eliminación si está en ejecución (`-f`) con el nombre de `naughty_stonebraker`

```
albamorejon@albamorejon:/$ docker rm -f naughty_stonebraker
albamorejon@albamorejon:/$
albamorejon@albamorejon:/$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED
STATUS        PORTS
b876510ac7f3   ghcr.io/digininja/dwa:latest       "docker-php-entrypoi..." 6 weeks ago
Up About an hour 127.0.0.1:4280->80/tcp    dwa_dwa_1
0f84b2b205b   mariadb:10                         "docker-entrypoint.s..." 6 weeks ago
Up About an hour 3306/tcp                  dwa_db_1
```

3.9- Comprobar que el contenedor anterior ya no está funcionando. Indica la línea de comandos para realizar esta operación.

`docker ps -a`, muestra todos los contenedores, en cualquier estado y vemos que ya no aparece el jenkins

```
albamorejon@albamorejon:/$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED
STATUS        PORTS
e3df357a88a0   hello-world                         "/hello"                About an hour ago
Exited (0) About an hour ago
6a9e09fb9374   hello-world                         "/hello"                2 hours ago
Exited (0) 2 hours ago
b876510ac7f3   ghcr.io/digininja/dwa:latest       "docker-php-entrypoi..." 6 weeks ago
Up About an hour 127.0.0.1:4280->80/tcp    dwa_dwa_1
0f84b2b205b   mariadb:10                         "docker-entrypoint.s..." 6 weeks ago
Up About an hour 3306/tcp                  dwa_db_1
```

4- Responde a las siguientes preguntas

4.1- ¿Qué ventajas aporta trabajar con contenedores?

Ofrece ventajas como:

1. Portabilidad: Los contenedores son portables entre diferentes entornos, esto significa que puedes ejecutar el mismo contenedor en tu máquina local, en un servidor de pruebas o en la nube.
2. Aislamiento: Cada contenedor se ejecuta de forma aislada, lo que significa que los problemas en un contenedor no afectan a otros contenedores. Esto mejora la seguridad y la estabilidad.
3. Escalabilidad: Los contenedores permiten escalar aplicaciones fácilmente. Puedes aumentar o disminuir el número de contenedores según la demanda, lo que facilita la gestión de recursos.
4. Eficiencia: Los contenedores son más ligeros que las máquinas virtuales porque comparten el mismo sistema operativo. Esto reduce el uso de recursos y mejora el rendimiento.

4.2- Revisa el pipeline que has creado en el apartado anterior ¿Qué hace?

Buscando información encontramos este esquema de script:

Se clona el repositorio de GitHub especificado. Esto asegura que el código fuente está disponible para las siguientes etapas.

Se ejecuta el comando `mvn clean install` para compilar el proyecto usando Maven. Esto construye el proyecto y verifica que no haya errores de compilación.

Se ejecuta las pruebas del proyecto con `mvn test`. Esto asegura que el código funciona correctamente y que no hay errores en las pruebas unitarias.

Se ejecuta el comando `mvn deploy` para desplegar el proyecto. Esto puede implicar la publicación del artefacto en un repositorio de artefactos o la implementación en un servidor.

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/tu-repositorio/tu-proyecto.git'
            }
        }
        stage('Build') {
            steps {
                sh 'mvn clean install'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
        }
        stage('Deploy') {
            steps {
                sh 'mvn deploy'
            }
        }
    }
}
```

4.3- Busca información sobre pruebas de seguridad en un pipeline e indica que pasos se podrían añadir al pipeline anterior para asegurar que la aplicación que se va a poner en producción es segura.

Para asegurar que la aplicación que se va a poner en producción es segura, puedes añadir:

- Análisis de código estático (SAST): integrar herramientas de análisis de código estático para detectar vulnerabilidades en el código fuente durante el desarrollo. Ejemplos de herramientas: SonarQube, Checkmarx 5.
- Análisis de dependencias: escanear las dependencias del proyecto para detectar vulnerabilidades conocidas. Ejemplos de herramientas: OWASP Dependency-Check, Snyk 6.
- Pruebas de seguridad dinámicas (DAST): realizar pruebas de seguridad dinámicas para detectar vulnerabilidades en la aplicación en ejecución. Ejemplos de herramientas: OWASP ZAP, Burp Suite 5.

Apartado 2: Dockerfile**1- Dado el siguiente archivo Dockerfile**

```
FROM ubuntu
LABEL maintainer="admin@ejemplo.com"
ARG APP_DATO1=1
RUN apt-get update && apt-get install -y apache2
RUN mkdir /var/www/html/ejemplo
EXPOSE 80
ADD index2.html /var/www/html/
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Rellena la siguiente tabla donde tendrás que indicar para que sirve cada línea

FROM ubuntu	Especifica la imagen base que se utilizará para crear el contenedor. En este caso, se utiliza la imagen de Ubuntu.
LABEL maintainer="admin@ejemplo.com"	Añade una etiqueta con información del mantenedor de la imagen. Esto es útil para identificar quién es responsable de la imagen.
ARG APP_DATO1=1	Define una variable de construcción llamada APP_DATO1 con un valor predeterminado de 1. Las variables de construcción se pueden utilizar durante el proceso de construcción de la imagen.
RUN apt-get update && apt-get install -y apache2	Ejecuta comandos en el contenedor para actualizar la lista de paquetes e instalar el servidor web Apache.
RUN mkdir /var/www/html/ejemplo	Crea un directorio llamado ejemplo en la ruta /var/www/html/.
EXPOSE 80	Indica que el contenedor escuchará en el puerto 80 en tiempo de ejecución. Esto es útil para la documentación y para que las herramientas de orquestación sepan qué puertos exponer.
ADD index2.html /var/www/html/	Copia el archivo index2.html desde el contexto de construcción del Dockerfile al directorio /var/www/html/ dentro del contenedor.
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]	Define el comando que se ejecutará cuando se inicie un contenedor a partir de esta imagen. En este caso, se inicia el servidor Apache en primer plano.

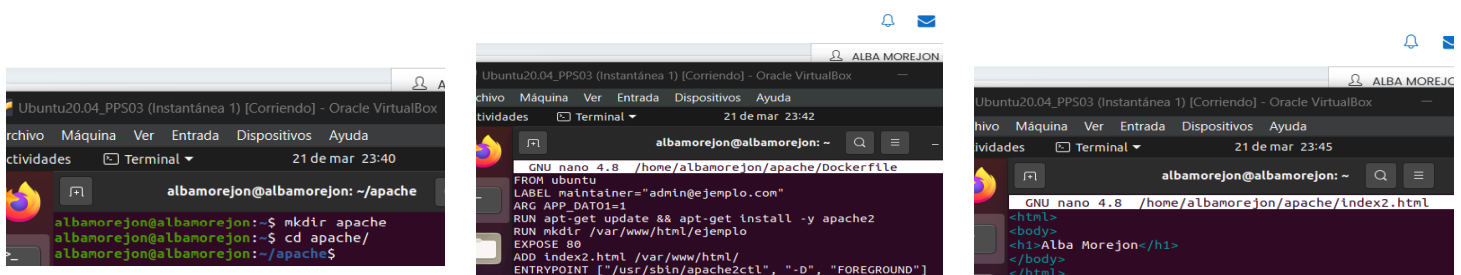
2- Responde a las siguientes preguntas y realiza las pruebas**2.1- ¿Para qué es útil un archivo Dockerfile?**

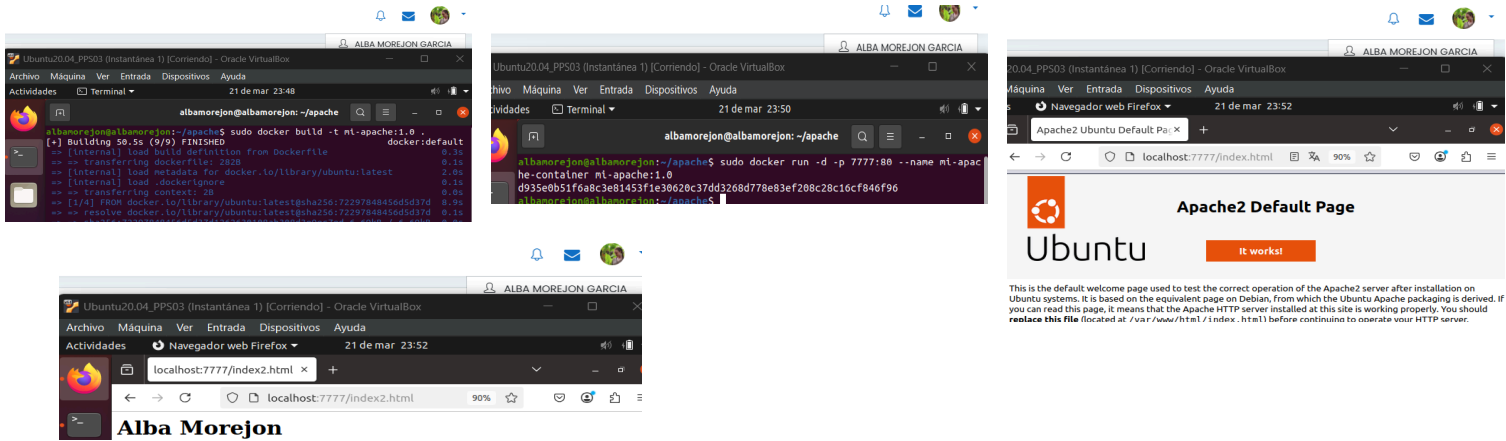
Un archivo Dockerfile es útil para automatizar la creación de imágenes de Docker. Permite definir todos los pasos necesarios para construir una imagen, incluyendo la instalación de software, la configuración del entorno y la copia de archivos. Esto asegura que la imagen sea reproducible y consistente cada vez que se construya.

Un archivo Dockerfile es útil para definir la configuración y los pasos necesarios para crear una imagen de Docker. Es un script que contiene instrucciones sobre cómo construir una imagen, incluyendo la base de la imagen, los paquetes que se deben instalar, las configuraciones del sistema, y los comandos que se deben ejecutar.

2.2- ¿Con qué comando se crean las imágenes partiendo de un Dockerfile?

`docker build -t nombre_imagen:version .`, crea una imagen partiendo de un Dockerfile, asignando un nombre y una versión a la imagen (`-t imagen:version`) y se creará en el directorio actual (`.`).

2.3- Realiza una prueba con el archivo anterior:**2.3.1- Crea un directorio****2.3.2- Crea dentro del directorio el archivo Dockerfile anterior****2.3.3- Crea el archivo index2.html (escribe dentro tu nombre)****2.3.4- Crea una imagen nueva con ese archivo Dockerfile de nombre mi-apache y versión 1.0****2.3.5- Lanza un contenedor con la imagen anterior en el puerto 7777.****2.3.6- Abre un navegador y accede al contenedor anterior mostrando las páginas index.html e index2.html**



Apartado 3: Docker-compose

1- Rellena la siguiente tabla indicando para que sirve cada línea de comandos u opción dentro de un archivo docker-compose.yml

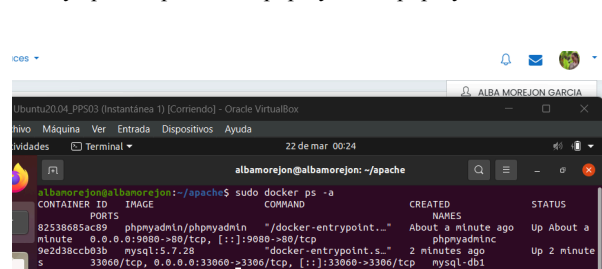
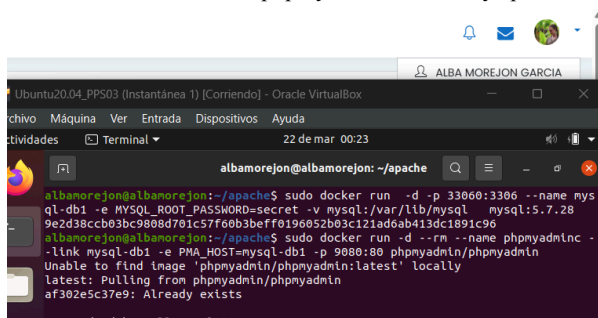
docker-compose up	Levanta y ejecuta todos los servicios definidos en el archivo docker-compose.yml. Si las imágenes no existen, las construye.
docker-compose down	Detiene y elimina los contenedores, redes y volúmenes definidos en el archivo docker-compose.yml.
services	Define los servicios que se ejecutarán en los contenedores. Cada servicio puede tener su propia configuración.
image	Especifica la imagen de Docker que se utilizará para el servicio. Puede ser una imagen existente en Docker Hub o una imagen personalizada.
build	Define la ruta del Dockerfile y el contexto para construir la imagen del servicio.
container-name	Asigna un nombre en específico al contenedor
ports	Mapea los puertos del contenedor a los puertos del host. Por ejemplo, - "8080: 80" mapea el puerto 80 del contenedor al puerto 8080 del host.
volumes	Monta volúmenes para persistencia de datos o para compartir archivos entre el host y el contenedor.
environment	Define variables de entorno que se pasarán al contenedor

2- Realiza las siguientes pruebas (incluye pantallazos de cada una)

2.1- Lanza dos contenedores con las siguientes líneas en una terminal y muestra pantallazos de las ejecución de esos comandos (docker ps -a)

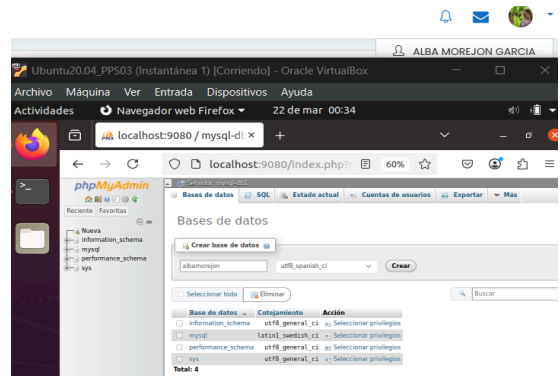
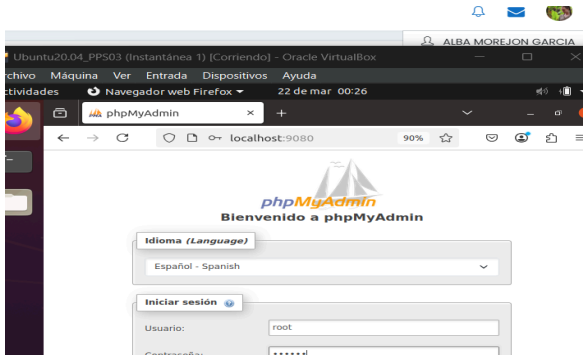
docker run -d -p 33060:3306 --name mysql-db1 -e MYSQL_ROOT_PASSWORD=secret -v mysql:/var/lib/mysql mysql:5.7.28

docker run -d --rm --name phpmyadmin -link mysql-db1 -e PMA_HOST=mysql-db1 -p 9080:80 phpmyadmin/phpmyadmin

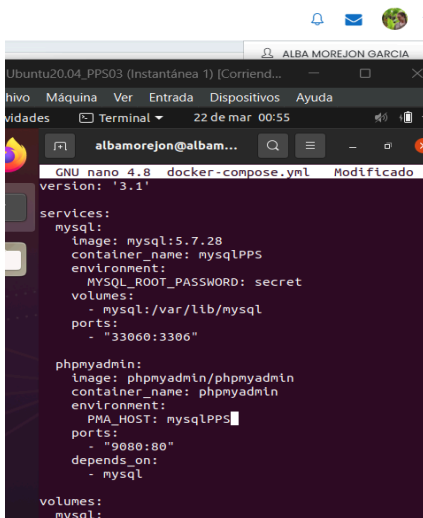


2.2- Utilizando el contenedor de phpmyadmin crea una base de datos con tu nombre en mysql. Muestra pantallazos del navegador con el acceso a phpmyadmin y con la creación de la base de datos.

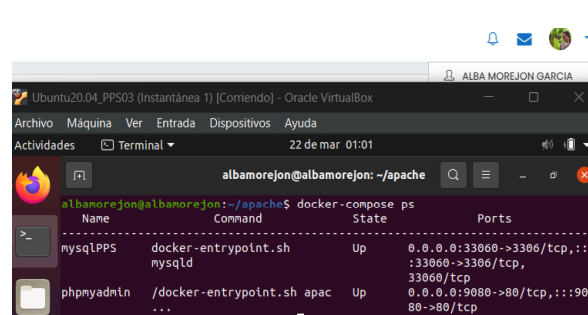
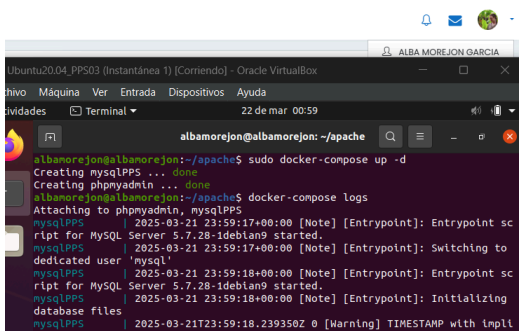
root:secret



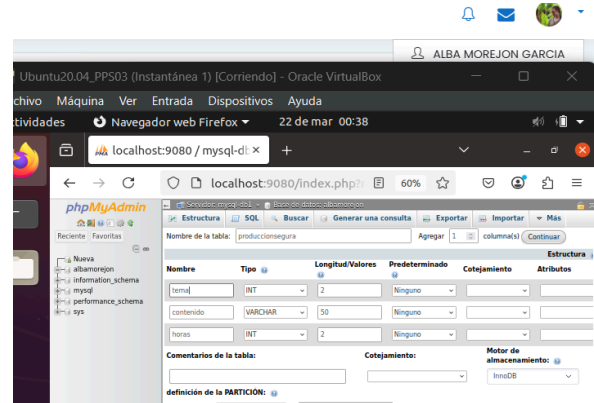
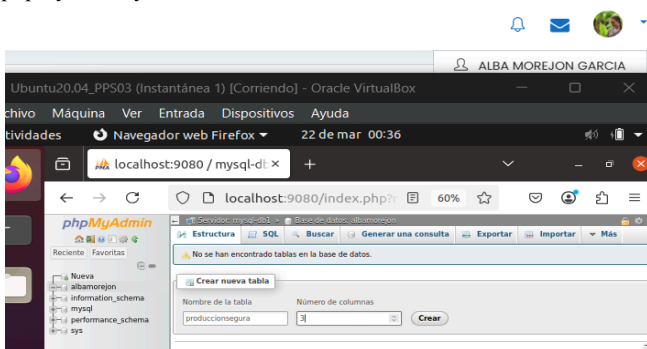
2.3- Crea un archivo docker-compose.yml que permita lanzar esos dos contenedores utilizando docker compose. Incluye el contenido del archivo docker-compose.yml

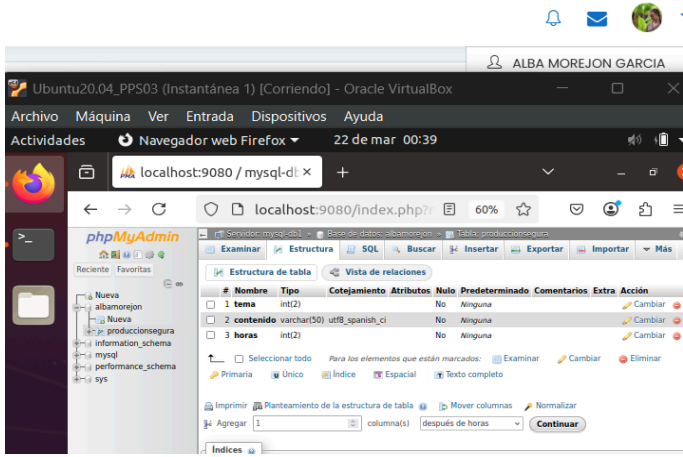


2.4- Lanza el archivo anterior. Muestra pantallazos de la ejecución en la consola



2.5- Utilizando el contenedor de phpmyadmin crea una tabla en la base de datos. Muestra pantallazos del navegador con el acceso a phpmyadmin y con la creación de la tabla





2.6- Termina los contenedores lanzados. Muestra pantallazos de la ejecución en la consola

