

Implantación de arquitecturas web

RA1: Implanta arquitecturas web analizando y aplicando criterios de funcionalidad

Índice de usuario

1. Análisis de aspectos generales de arquitecturas web, sus características, ventajas e inconvenientes (Criterio 1.a).....	2
2. Fundamentos y protocolos en los que se basa el funcionamiento de un servidor web (Criterio 1.b).....	3
3. Instalación y configuración básica de servidores web (Criterio 1.c) (Se verá en las prácticas).....	3
4. Instalación y configuración básica de servidores de aplicaciones (Criterio 1.d) (Se ve en otro tutorial).....	6
5. Virtualización de servidores en la nube y contenedores (Criterio 1.e).....	9
6. Pruebas de funcionamiento de servidores web, de aplicaciones y de tecnologías de virtualización en la nube y en contenedores (Criterio 1.f) (Se verá en profundidad cuando en otros temas).....	13
7. Análisis de la estructura y recursos de una aplicación web (Criterio 1.g).....	15
Concepto de descriptor de despliegue.....	15
8. Requerimientos del proceso de implantación de una aplicación web (Criterio 1.h).....	18
9. Documentación de procesos de instalación y configuración realizados sobre los servidores web, de aplicaciones y sobre tecnologías de virtualización en la nube y en contenedores (Criterio 1.i).....	19

1. Análisis de aspectos generales de arquitecturas web, sus características, ventajas e inconvenientes (Criterio 1.a)

Las arquitecturas web estructuran cómo interactúan los diferentes componentes de una aplicación web (base de datos, interfaz gráfica y procesamiento de información) para garantizar su funcionamiento. A continuación, se describen las principales arquitecturas junto con sus ventajas e inconvenientes:

Arquitectura monolítica

- **Descripción:** Toda la lógica de negocio, la interfaz de usuario y el acceso a la base de datos están integrados en un único programa.
- **Ventajas:**
 - Fácil de desarrollar para proyectos pequeños.
 - Requiere menos recursos técnicos.
 - Despliegue y mantenimiento iniciales sencillos.
- **Inconvenientes:**
 - Difícil de escalar cuando crece la aplicación.
 - Los cambios en una parte del sistema pueden afectar a todo el conjunto.
 - Actualizaciones complejas y con riesgo de interrupciones.

Arquitectura cliente-servidor

- **Descripción:** Divide el sistema en dos partes principales: cliente (interfaz gráfica) y servidor (procesamiento y base de datos).
- **Ventajas:**
 - Separa la lógica de presentación de la lógica de negocio, facilitando el desarrollo.
 - Mejor organización y escalabilidad que una arquitectura monolítica.
 - Ideal para aplicaciones sencillas y de mediana complejidad.
- **Inconvenientes:**
 - Limitada capacidad de escalabilidad horizontal (uso de múltiples servidores).
 - El servidor puede saturarse si no se distribuye adecuadamente la carga.

Arquitectura multicapa (3 capas)

- **Descripción:** Divide la aplicación en tres capas:
 1. **Capa de presentación:** Responsable de la interfaz con el usuario (HTML, CSS, JavaScript).
 2. **Capa de lógica de negocio:** Procesa las reglas del negocio (backend).
 3. **Capa de datos:** Maneja las bases de datos.
- **Ventajas:**
 1. Modificación de capas de forma independiente.
 2. Alta escalabilidad y reutilización de componentes.
 3. Facilita el mantenimiento y la incorporación de nuevas funcionalidades.
- **Inconvenientes:**
 1. Complejidad técnica mayor que en arquitecturas monolíticas o cliente-servidor.
 2. Requiere conocimientos específicos y más recursos para implementarla.

Arquitectura de microservicios

- **Descripción:** Descompone una aplicación en servicios independientes, donde cada uno ejecuta una funcionalidad específica.
- **Ventajas:**
 - Escalabilidad horizontal masiva (se pueden escalar servicios de forma independiente).

- Facilita el desarrollo continuo y el trabajo en equipo.
- Alta resiliencia: si un microservicio falla, el resto de la aplicación sigue funcionando.
- **Inconvenientes:**
 - Complejidad en la orquestación y comunicación entre microservicios.
 - Mayor esfuerzo en la configuración y monitorización.
 - Requiere experiencia técnica avanzada y herramientas específicas (Docker, Kubernetes).

Arquitectura Serverless

- **Descripción:** La infraestructura es gestionada automáticamente por el proveedor de servicios en la nube, eliminando la necesidad de gestionar servidores.
- **Ventajas:** Escalabilidad automática, costos basados en el uso y enfoque en el desarrollo de funciones individuales.
- **Inconvenientes:** Menos control sobre la infraestructura, y algunas limitaciones en términos de tiempo de ejecución y recursos

2. Fundamentos y protocolos en los que se basa el funcionamiento de un servidor web (Criterio 1.b)

Los servidores web operan mediante protocolos como HTTP y HTTPS.

HTTP (Hypertext Transfer Protocol):

Es el protocolo estándar para transmitir datos en la web. Funciona sobre el puerto 80 y no cifra la información, lo que lo hace vulnerable a ataques.

HTTPS (HTTP Secure):

Es una extensión de HTTP que utiliza SSL/TLS para cifrar las comunicaciones. Funciona en el puerto 443 y garantiza seguridad, autenticidad y confidencialidad.

Comunicación navegador-servidor:

Cuando el usuario accede a una URL, el navegador realiza una solicitud HTTP/HTTPS. El servidor responde enviando la página solicitada o devolviendo información en formatos como JSON o XML.

DNS (Domain Name System):

El DNS traduce nombres de dominio legibles por humanos (como www.ejemplo.com) en direcciones IP que los servidores pueden interpretar.

3. Instalación y configuración básica de servidores web (Criterio 1.c) (Se verá en las prácticas)

Existen varios servidores web disponibles, cada uno con características únicas:

Apache HTTP Server:

Es flexible y modular, compatible con múltiples lenguajes como PHP y Python.

Nginx:

Destaca por su capacidad para manejar grandes cantidades de tráfico y actuar como proxy inverso.

IIS (Internet Information Services):

Es el servidor nativo de Microsoft para aplicaciones ASP.NET.

La instalación y configuración básica incluyen:

1. **Definir el directorio raíz:** En Apache, la ruta por defecto es /var/www/html.
2. **Configurar el puerto de escucha:** En el archivo apache2.conf o nginx.conf.
3. **Activar módulos necesarios:** Por ejemplo, a2enmod ssl para HTTPS en Apache.

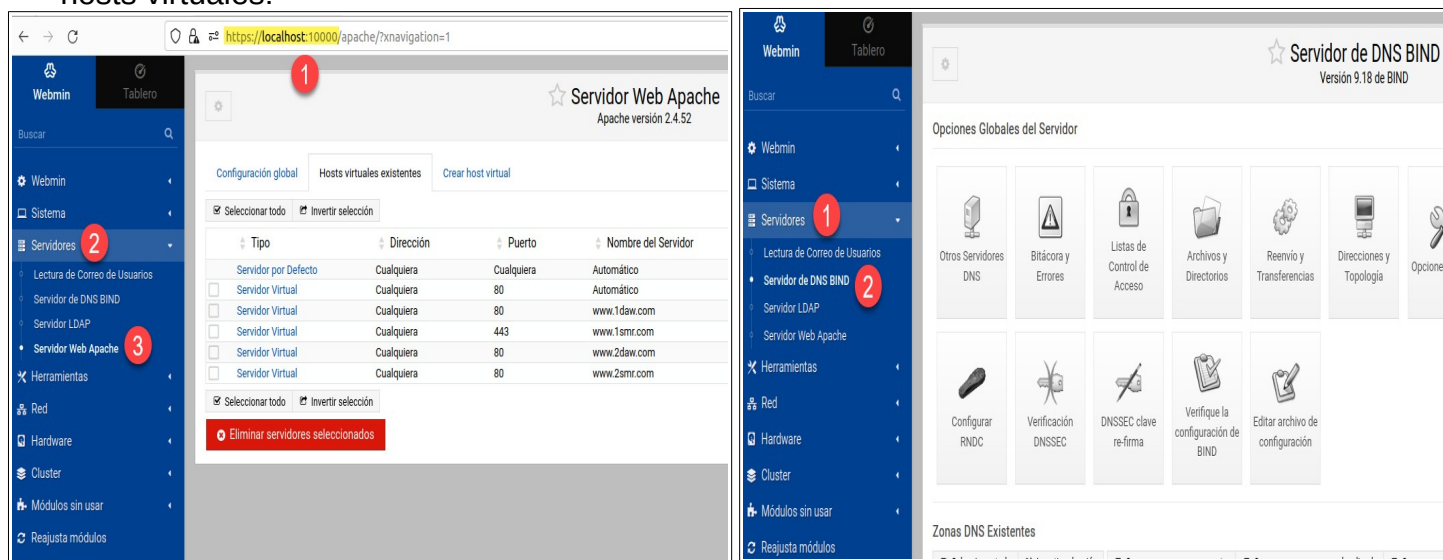
Se usará el servidor web Apache. En esencia, Apache se instala en Ubuntu con el comando

sudo apt install apache2 apache2-utils

Una vez instalado, en el navegador ponemos localhost en la barra de direcciones y debe aparecer lo siguiente:



La configuración de Apache la haremos usando Webmin y Bind para el manejo de dominios de hosts virtuales:



Además, se va a instalar php en el servidor web Apache, siguiendo estos pasos:

1) sudo apt-get install php

Para probar que funciona, se crea un archivo index.php en /var/www/html con este código:

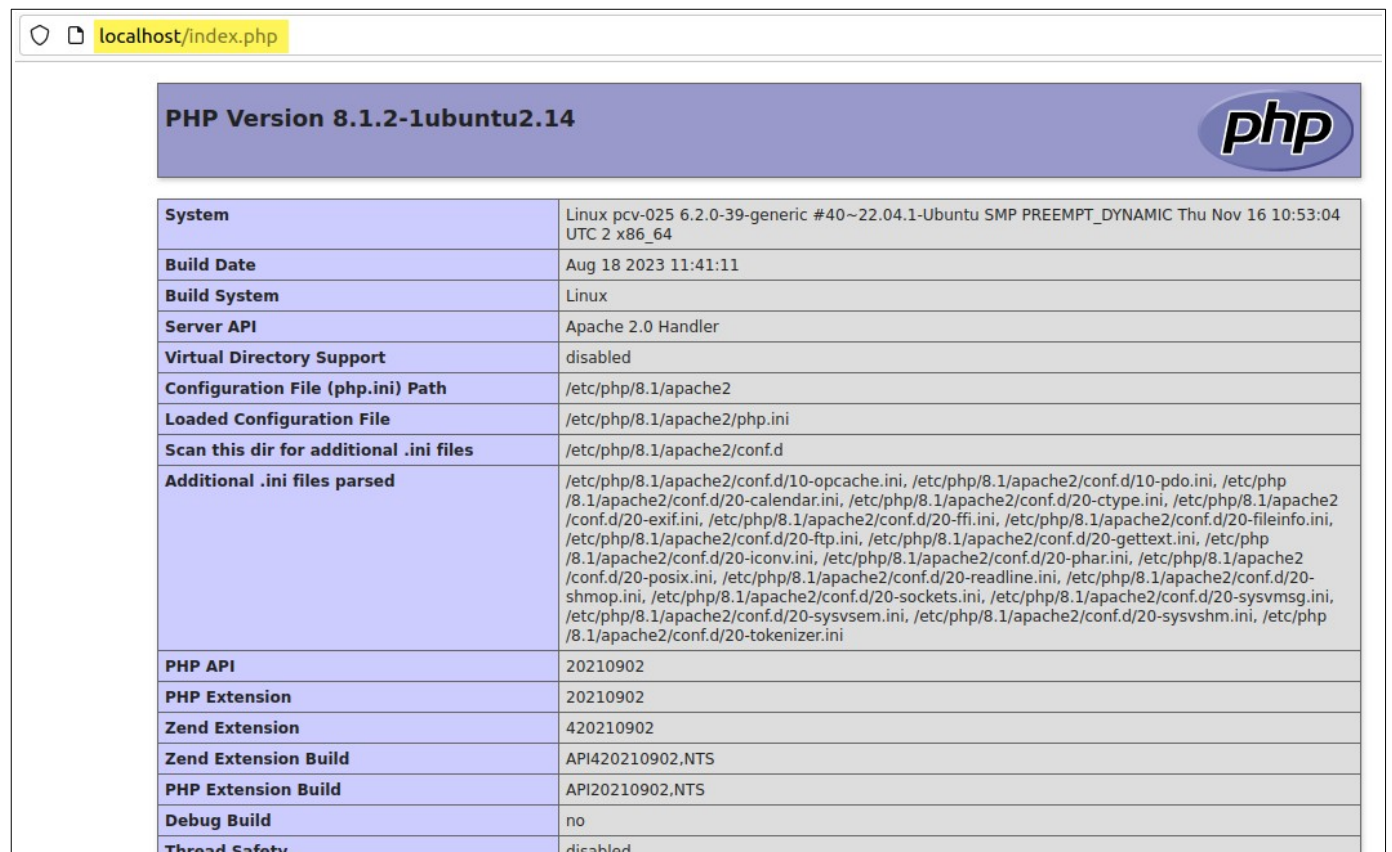
```
index.php
/var/www/html

1 <?php
2     phpinfo();
3 ?>
```

2) Para finalizar,

```
service apache2 restart
```

Comprobación: si en el navegador escribimos localhost/index.php deberá salir algo así:



System	Linux pcv-025 6.2.0-39-generic #40~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Thu Nov 16 10:53:04 UTC 2 x86_64
Build Date	Aug 18 2023 11:41:11
Build System	Linux
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/8.1/apache2
Loaded Configuration File	/etc/php/8.1/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/8.1/apache2/conf.d
Additional .ini files parsed	/etc/php/8.1/apache2/conf.d/10-opcache.ini, /etc/php/8.1/apache2/conf.d/10-pdo.ini, /etc/php/8.1/apache2/conf.d/20-calendar.ini, /etc/php/8.1/apache2/conf.d/20-ctype.ini, /etc/php/8.1/apache2/conf.d/20-exif.ini, /etc/php/8.1/apache2/conf.d/20-ffi.ini, /etc/php/8.1/apache2/conf.d/20-fileinfo.ini, /etc/php/8.1/apache2/conf.d/20-ftp.ini, /etc/php/8.1/apache2/conf.d/20-gettext.ini, /etc/php/8.1/apache2/conf.d/20-iconv.ini, /etc/php/8.1/apache2/conf.d/20-phar.ini, /etc/php/8.1/apache2/conf.d/20-posix.ini, /etc/php/8.1/apache2/conf.d/20-readline.ini, /etc/php/8.1/apache2/conf.d/20-shmop.ini, /etc/php/8.1/apache2/conf.d/20-sockets.ini, /etc/php/8.1/apache2/conf.d/20-sysvmsg.ini, /etc/php/8.1/apache2/conf.d/20-sysvsem.ini, /etc/php/8.1/apache2/conf.d/20-sysvshm.ini, /etc/php/8.1/apache2/conf.d/20-tokenizer.ini
PHP API	20210902
PHP Extension	20210902
Zend Extension	420210902
Zend Extension Build	API420210902.NTS
PHP Extension Build	API20210902.NTS
Debug Build	no
Thread Safety	disabled

Hay multitud de lugares en Internet para aprender php, así como código php gratuito para incluir en las páginas web.

4. Instalación y configuración básica de servidores de aplicaciones (Criterio 1.d) (Se ve en otro tutorial)

Un servidor de aplicaciones ejecuta lógica de negocio y ejecuta código dinámico (Java, PHP, Python, Node.js). **La diferencia principal es su propósito: un servidor web se enfoca en entregar contenido estático (HTML, CSS, imágenes) a los navegadores, mientras que un servidor de aplicaciones web se encarga de procesar la lógica de negocio y generar contenido dinámico.**

Servidor Web

Un servidor web es un programa de software que escucha las solicitudes HTTP de los clientes (navegadores web). Su función principal es servir archivos estáticos almacenados en su disco.

- **Función:** Gestionar peticiones HTTP y enviar archivos como páginas HTML, hojas de estilo CSS, imágenes, y archivos de JavaScript directamente al cliente.
- **Contenido:** Principalmente estático. El servidor entrega los archivos tal cual están almacenados.
- **Ejemplos:** Apache HTTP Server, Nginx, Microsoft IIS.

Servidor de Aplicaciones Web

Un servidor de aplicaciones web es un framework de software que proporciona el entorno para que las aplicaciones web se ejecuten. Va más allá de servir archivos y se centra en la lógica de negocio y la interacción con bases de datos.

- **Función:** Procesar datos complejos, ejecutar código de la aplicación (Java, Python, PHP, .NET), interactuar con bases de datos y generar respuestas personalizadas y dinámicas.
- **Contenido:** Principalmente dinámico. La página se genera en el momento, por ejemplo, mostrando los datos de un usuario en un perfil o el contenido de un carrito de compras.
- **Ejemplos:** Apache Tomcat, JBoss, GlassFish.

¿Cómo trabajan juntos?

En la mayoría de las arquitecturas modernas, ambos servidores trabajan en conjunto. El servidor web actúa como un "front-end" que recibe todas las peticiones. Si la petición es para un archivo estático, el servidor web lo entrega directamente. Si la petición requiere lógica de negocio (por ejemplo, acceder a una base de datos o procesar un formulario), el servidor web reenvía esa solicitud al servidor de aplicaciones, que procesa la petición, genera la respuesta dinámica y se la devuelve al servidor web, quien finalmente la envía al navegador del usuario.

Clasificación de servidores de aplicaciones:

La clasificación de servidores de aplicaciones web se puede hacer de varias maneras. La más común es por su **licencia de uso** (código abierto o comercial), y por el **lenguaje de programación** o **plataforma** que soportan.

a) Clasificación por Licencia

- De Código Abierto (Open Source)

Son gratuitos y su código fuente está disponible para ser modificado y distribuido. Son muy populares debido a su flexibilidad y bajo costo.

- **Apache Tomcat:** Un contenedor de servlets y un servidor de aplicaciones web para aplicaciones Java. Es uno de los más populares para el ecosistema Java EE (ahora Jakarta EE).
- **JBoss/WildFly:** Servidor de aplicaciones Java EE de Red Hat, con una arquitectura modular y alto rendimiento. JBoss es el nombre de la versión empresarial con soporte, mientras que WildFly es la versión de la comunidad.
- **GlassFish:** Servidor de aplicaciones Java EE de Oracle, conocido por su soporte a las últimas especificaciones de Java EE. Aunque Oracle lo ha retirado de su soporte comercial, la comunidad lo mantiene activamente.
- **Jetty:** Un servidor HTTP y contenedor de servlets que se puede embeber fácilmente en aplicaciones Java, ideal para microservicios.
- **Nginx:** Aunque es más conocido como un servidor web y proxy inverso, Nginx también puede servir aplicaciones dinámicas a través de módulos como uWSGI o Gunicorn para Python.

- Comerciales (Propietarios)

Requieren una licencia de pago para su uso. Ofrecen soporte técnico profesional, herramientas de administración avanzadas y garantías de seguridad.

- **WebSphere Application Server:** Un servidor de aplicaciones Java EE de IBM, muy usado en entornos empresariales grandes y complejos.
- **Oracle WebLogic Server:** Servidor de aplicaciones Java EE de Oracle, conocido por su alta escalabilidad y fiabilidad. Es una opción común en grandes corporaciones.
- **Microsoft Internet Information Services (IIS):** Es el servidor de aplicaciones web de Microsoft, integrado en los sistemas operativos Windows Server. Soporta principalmente tecnologías de Microsoft como ASP.NET.
- **SAP NetWeaver Application Server:** Un servidor de aplicaciones que forma parte de la suite de productos SAP, usado para aplicaciones empresariales.

b) Clasificación por Plataforma/Lenguaje

Esta clasificación se enfoca en el tipo de tecnología o lenguaje de programación que el servidor está diseñado para ejecutar.

- *Basados en Java*

Están optimizados para aplicaciones que usan el **Java EE** (ahora Jakarta EE) o **Spring Framework**.

- **Apache Tomcat**
- **JBoss/WildFly**
- **Oracle WebLogic Server**
- **IBM WebSphere**
- **GlassFish**

- *Basados en .NET*

Optimizados para el ecosistema de **Microsoft .NET**.

- **Microsoft Internet Information Services (IIS)**: Es el servidor principal para aplicaciones ASP.NET.

- *Basados en Python*

No son servidores de aplicaciones dedicados como tal, sino **servidores de despliegue** o **gateways de interfaz** (como WSGI o ASGI) que ejecutan la aplicación.

- **Gunicorn**
- **uWSGI**
- **Waitress**

- *Basados en Node.js*

Node.js es en sí mismo un **runtime** que puede actuar como servidor web. El servidor se crea directamente dentro de la aplicación.

- **Express.js**: Aunque es un **framework** web, es la forma más común de crear un servidor en Node.js.
- **Koa.js**
- **NestJS**

- *Otros*

Existen servidores para otros lenguajes.

- **Phusion Passenger**: para Ruby y Python.
- **Unicorn**: para Ruby.
- **Apache HTTP Server**: Aunque es un servidor web, con módulos como **mod_php** puede funcionar como un servidor de aplicaciones PHP.

La elección de un servidor de aplicaciones depende de factores como el **lenguaje de programación** de la aplicación, el **costo**, la **escalabilidad** requerida y el nivel de **soporte** que se necesite.

Instalación y configuración del servidor de aplicaciones Tomcat

Se explica detenidamente en otro tutorial.

5. Virtualización de servidores en la nube y contenedores (Criterio 1.e)

La virtualización es una tecnología que permite crear entornos de trabajo virtuales, abstraídos del hardware físico. Esto se puede hacer de dos maneras principales: virtualización de servidores en la nube y virtualización mediante contenedores.

1. Virtualización de Servidores en la Nube

La virtualización en la nube, también conocida como **IaaS (Infraestructura como Servicio)**, permite alquilar un servidor virtual (una **máquina virtual** o **VM**) en centros de datos remotos, sin tener que preocuparse por el hardware físico.

- **Conceptos Clave:**

- **Máquina Virtual (VM):** Un entorno virtual completo, con su propio sistema operativo, procesador, memoria RAM y almacenamiento, que funciona de forma independiente.
- **Hipervisor:** El software que crea y gestiona las máquinas virtuales en el servidor físico. Ejemplos: VMware vSphere, KVM (Kernel-based Virtual Machine) o Hyper-V.
- **Proveedor de Nube:** Empresas que ofrecen estos servicios. Ejemplos: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP).

- **Instalación y Configuración Básica:**

1. **Crear una cuenta:** Regístrate en un proveedor de nube como AWS. La mayoría ofrecen un nivel gratuito (Free Tier) para empezar.
 2. **Lanzar una instancia:** Desde la consola de AWS, seleccionar "EC2" (Elastic Compute Cloud) y haz clic en "Launch Instance" (Lanzar instancia).
 3. **Elegir la AMI (Amazon Machine Image):** Es una plantilla de la VM. Se elige una que contenga un sistema operativo, como Ubuntu Server.
 4. **Seleccionar el tipo de instancia:** Se define el tamaño de la VM (CPU y RAM). Para el tutorial, se selecciona un tipo t2.micro, que suele estar incluido en el nivel gratuito.
 5. **Configurar seguridad:** Se crea un par de claves (.pem) para conectarte de forma segura a la VM mediante SSH.
- **Ejemplo Práctico:** Para acceder a la VM que el usuario acaba de crear, se usará un terminal. En Linux o macOS, se usa SSH. En Windows, se puede usar PuTTY o el Subsistema de Windows para Linux (WSL).

```
# Permisos para tu clave
chmod 400 mi-clave.pem
# Conexión SSH a la instancia. Reemplaza la IP y el usuario (ubuntu en este caso)
ssh -i "mi-clave.pem" ubuntu@tu-ip-publica-aws
```

Una vez dentro de la VM, puedes instalar un servidor web como Apache:

```
sudo apt update
```

```
sudo apt install apache2
```

2. Virtualización en Contenedores

Los **contenedores** son una forma más ligera de virtualización. En lugar de virtualizar todo el sistema operativo, los contenedores comparten el mismo kernel del sistema operativo del host. Esto los hace más rápidos, eficientes y portátiles.

- **Conceptos Clave:**

- **Imagen:** Una plantilla inmutable que contiene el código de la aplicación, las bibliotecas, las dependencias y la configuración. Es como un "plano".
- **Contenedor:** Una instancia ejecutable de una imagen. Es el "producto final" que está funcionando.
- **Docker:** La plataforma más popular para la gestión de contenedores. Se usa para construir, ejecutar y gestionar imágenes y contenedores.

- **Instalación y Configuración Básica de Docker:**

1. **Instalar Docker Engine:** Sigue las instrucciones oficiales de Docker para tu sistema operativo (Linux, Windows o macOS). En Ubuntu:

```
# Actualizar paquetes
sudo apt-get update

# Instalar paquetes requeridos
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg

# Agregar la clave GPG de Docker
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Agregar el repositorio de Docker
echo \
    "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
    "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
    sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Instalar Docker
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-
plugin
```

2. **Verificar la instalación:** Ejecuta el comando para ver la versión de Docker.

```
docker --version
```

Ejemplo Práctico (uso de Docker):

1. **Crear un archivo Dockerfile:** En un directorio vacío, se crea un archivo llamado Dockerfile para definir la imagen de la aplicación web (por ejemplo, con Node.js).

```
# Usa una imagen base de Node.js
FROM node:18-alpine
# Establece el directorio de trabajo en el contenedor
WORKDIR /app
# Copia el código de la aplicación al contenedor
COPY . .
# Expone el puerto 3000
EXPOSE 3000
# Comando para iniciar la aplicación
CMD ["node", "app.js"]
```

2. Construir la imagen:

```
docker build -t mi-app-web
```

3. Ejecutar el contenedor:

```
docker run -p 80:3000 mi-app-web
```

Con este comando se mapea el puerto 80 de la máquina host al puerto 3000 del contenedor, haciendo que tu aplicación sea accesible desde tu navegador en <http://localhost>

Resumen y Comparación

Característica	Virtualización en la Nube (VM)	Contenedores (Docker)
Escala	Servidor completo	Aplicación y dependencias
Tamaño	Gigabytes	Megabytes
Inicio	Minutos	Segundos
Aislamiento	Total (hardware y SO)	A nivel de proceso (comparten kernel)
Portabilidad	Menor (depende del hipervisor)	Máxima (cualquier sistema con Docker)
Uso Ideal	Entornos empresariales, servidores de bases de datos, aplicaciones legadas	Microservicios, CI/CD, despliegue rápido

Conclusión: Para el despliegue de aplicaciones web, la virtualización en la nube ofrece una infraestructura robusta y completa, ideal para aplicaciones monolíticas o grandes. Por otro lado, los contenedores proporcionan una agilidad y portabilidad inigualables, perfectas para arquitecturas de microservicios y flujos de trabajo de DevOps¹ modernos.

1DevOps es un término que surge de la combinación de las palabras en inglés **Development** (desarrollo) y **Ops** (operaciones). No es una herramienta o un puesto de trabajo, sino una **filosofía y un conjunto de prácticas** que buscan derribar las barreras tradicionales entre los equipos de desarrollo de software y los equipos de operaciones de TI.

El objetivo principal de DevOps es mejorar la colaboración y la comunicación para acelerar el ciclo de vida del desarrollo de software, lo que se traduce en:

- **Entregas más rápidas y frecuentes:** Se busca automatizar los procesos para lanzar nuevas funcionalidades y actualizaciones de software de manera continua y eficiente.
- **Mayor calidad y fiabilidad:** Al trabajar juntos, los equipos pueden detectar y solucionar problemas más rápido, lo que reduce los errores en las implementaciones.
- **Innovación constante:** Al tener ciclos de retroalimentación más cortos, las empresas pueden responder rápidamente a las necesidades de los clientes y adaptarse a los cambios del mercado.

Para lograr esto, DevOps se basa en la **automatización de tareas** manuales, el **monitoreo continuo** y el uso de **herramientas específicas**. Además, fomenta una **cultura de colaboración** donde todos los miembros del equipo, desde la planificación hasta la puesta en marcha, se responsabilizan del producto.

En resumen, DevOps no es solo una metodología, sino un cambio cultural que busca alinear a las personas, los procesos y la tecnología para ofrecer valor a los clientes de forma más rápida y segura.

6. Pruebas de funcionamiento de servidores web, de aplicaciones y de tecnologías de virtualización en la nube y en contenedores (Criterio 1.f) (Se verá en profundidad cuando en otros temas)

Las pruebas y el testeo son pasos críticos para asegurar que las aplicaciones web, los servidores y los entornos de virtualización funcionen correctamente, sean estables y ofrezcan un buen rendimiento antes de ser accesibles al público.

1. Pruebas de Servidores Web y de Aplicaciones

El objetivo es verificar que el software de servidor (como Apache, Nginx o Tomcat) funciona como se espera, maneja peticiones correctamente y está configurado de manera óptima.

- **Pruebas de Funcionalidad:**
 - **Acceso y Conexión:** Verifica que el servidor está en funcionamiento y que puedes acceder a él a través de su dirección IP o nombre de dominio. Puedes usar herramientas como ping o telnet.
 - **Gestión de Peticiones:** Asegúrate de que el servidor responde a las peticiones HTTP y HTTPS. Por ejemplo, comprueba que la página de inicio se carga correctamente y que las páginas de error (como el 404 "Not Found") se muestran cuando la URL no existe.
 - **Contenido Estático y Dinámico:** Confirma que el servidor sirve correctamente tanto archivos estáticos (HTML, CSS, imágenes) como el contenido generado por la aplicación (por ejemplo, una página de login o un listado de productos).
- **Pruebas de Rendimiento y Carga:**
 - **Capacidad de Respuesta:** Mide el tiempo que tarda el servidor en responder a una petición.
 - **Carga Máxima:** Simula el acceso de muchos usuarios simultáneos para ver cómo se comporta el servidor bajo una carga elevada. Herramientas como **Apache JMeter** o **Gatling** son ideales para esto. Esto ayuda a identificar cuellos de botella y a determinar si el servidor necesita más recursos.
 - **Escalabilidad:** Comprueba si el servidor puede manejar un aumento de carga al añadir más recursos (CPU, RAM) o al desplegar más instancias.
- **Pruebas de Seguridad:**
 - **Escaneo de Vulnerabilidades:** Utiliza herramientas como **Nmap** o **OWASP ZAP** para escanear puertos abiertos, versiones de software desactualizadas y posibles vulnerabilidades de seguridad.
 - **Protección HTTPS:** Confirma que los certificados SSL/TLS están bien configurados y que las comunicaciones están cifradas.
 - **Configuración de Firewall:** Verifica que las reglas del firewall solo permiten el tráfico necesario.

2. Pruebas en Tecnologías de Virtualización

Las pruebas en este ámbito se centran en el funcionamiento del entorno virtualizado, ya sean máquinas virtuales o contenedores.

2.1. Pruebas de Máquinas Virtuales (VM)

El objetivo es validar que el entorno virtual es estable, eficiente y que se comporta como si fuera un servidor físico.

- **Rendimiento de la VM:**
 - **CPU y RAM:** Monitorea el uso de recursos. Un uso de CPU constantemente alto o un consumo excesivo de RAM pueden indicar una configuración ineficiente o una aplicación mal optimizada.
 - **Disco y Red:** Comprueba la velocidad de lectura/escritura del disco virtual y el rendimiento de la red.
- **Gestión de Recursos:**
 - **Clonado y Redimensionamiento:** Prueba si puedes clonar la VM sin problemas y si puedes aumentar o disminuir los recursos (CPU, RAM) sin que esto afecte a la estabilidad.
 - **Migración:** Si se utiliza un hipervisor como VMware, verifica que la VM se puede mover entre servidores físicos sin interrupciones.

2.2. Pruebas de Contenedores (Docker)

El foco está en la portabilidad, el aislamiento y la eficiencia de la aplicación dentro del contenedor.

- **Pruebas de la Imagen:**
 - **Seguridad:** Escanea la imagen de Docker en busca de vulnerabilidades conocidas. Herramientas como **Trivy** o **Docker Scan** pueden ayudarte.
 - **Tamaño:** Intenta mantener las imágenes lo más pequeñas posible, ya que esto reduce el tiempo de descarga y la superficie de ataque.
- **Pruebas del Contenedor:**
 - **Ejecución y Aislamiento:** Asegúrate de que el contenedor se ejecuta correctamente y que no tiene acceso a recursos del host que no debería tener.
 - **Comunicación:** Verifica que el mapeo de puertos y la comunicación entre contenedores (si usas **Docker Compose**) funcionan como se espera.
 - **Estado del Contenedor:** Confirma que el contenedor puede iniciarse, detenerse y reiniciarse sin errores, y que se comporta de manera predecible.

3. Herramientas Comunes

Tipo de Prueba	Herramientas Comunes
Servidor Web y Apps	Apache JMeter, Gatling, Postman, Apache Benchmark (ab), OWASP ZAP
Virtualización (VM)	VMware vSphere (monitoreo), Prometheus y Grafana (monitoreo avanzado)
Contenedores (Docker)	Docker CLI (docker stats, docker inspect), Docker Scan, Trivy, Docker Compose

En resumen, el testing en el despliegue de aplicaciones web es un proceso integral que va desde verificar la funcionalidad básica de los servidores hasta garantizar que los entornos virtualizados sean robustos y seguros bajo diversas condiciones.

7. Análisis de la estructura y recursos de una aplicación web (Criterio 1.g)

Una aplicación web se compone de varios elementos interdependientes que permiten su correcto funcionamiento:

Front-end

El front-end es la parte visible de una aplicación web y la encargada de interactuar con el usuario. Está desarrollado con tecnologías como:

- **HTML:** Define la estructura del contenido.
- **CSS:** Se utiliza para estilizar y diseñar la apariencia visual.
- **JavaScript:** Permite añadir interactividad y manipulación dinámica del contenido.

Además, frameworks como Angular, React o Vue.js ofrecen herramientas avanzadas para desarrollar interfaces modernas y eficientes.

Back-end

El back-end es responsable de procesar la lógica de negocio, gestionar la autenticación de usuarios y realizar cálculos o consultas. Lenguajes como Python, Java, PHP y frameworks como Django, Spring o Laravel son comunes en esta capa.

Base de datos

La base de datos almacena la información de la aplicación. Existen dos tipos principales:

- **Bases de datos relacionales (SQL):** Como MySQL, PostgreSQL y Microsoft SQL Server, organizan los datos en tablas relacionadas.
- **Bases de datos no relacionales (NoSQL):** Como MongoDB y Cassandra, son más flexibles para almacenar datos no estructurados.

Otros recursos

Además de las capas principales, las aplicaciones web pueden incluir:

- **APIs:** Para interactuar con otros sistemas o servicios.
- **Middleware:** Para gestionar tareas como autenticación, registro de logs o envío de notificaciones.

Concepto de descriptor de despliegue.

Un descriptor de despliegue de aplicaciones web es un archivo de configuración utilizado en el desarrollo de aplicaciones web Java EE (Enterprise Edition). Este archivo proporciona información esencial sobre cómo debe ser desplegada y configurada una aplicación web en un servidor de aplicaciones Java EE. El descriptor de despliegue de aplicaciones web más común es el archivo "web.xml".

A continuación, se detallan las características y funciones típicas de un descriptor de despliegue de aplicaciones web:

web.xml:

1. **Tipo de aplicación:**
 - Aplicaciones web Java EE.
2. **Ubicación predeterminada:**
 - En el directorio "WEB-INF" de la estructura de la aplicación web.
3. **Funcionalidades típicas:**
 - **Configuración de servlets y filtros:**

Define servlets y filtros, proporcionando información como nombre de clase, URL de mapeo y parámetros de inicialización.

- **Definición de parámetros de inicialización:**

Permite configurar parámetros específicos de la aplicación que pueden ser accedidos por servlets y otros componentes.

- **Configuración de seguridad y restricciones de acceso:**

Especifica restricciones de seguridad y reglas de acceso a recursos, como páginas protegidas y roles de usuario.

- **Declaración de recursos y referencias:**

Declara recursos utilizados por la aplicación, como conexiones a bases de datos, y establece referencias a esos recursos.

- **Manejo de errores y páginas de error personalizadas:**

Define páginas de error personalizadas para diferentes códigos de estado HTTP y configuración para el manejo de excepciones.

- **Configuración de sesiones y cookies:**

Permite la configuración de características relacionadas con sesiones web y cookies, como tiempos de sesión y rastreo de cookies.

- **Configuración de contexto de aplicación:**

Establece información sobre el contexto de la aplicación, como parámetros de inicialización globales y detección y procesamiento de eventos de contexto.

- **Configuración de codificación y tipos MIME:**

Define la codificación de caracteres por defecto y mapeos de tipos MIME para la aplicación.

Este archivo es esencial para la correcta implementación y configuración de una aplicación web Java EE en un servidor. Aunque en versiones más recientes de Java EE y en el desarrollo basado en Spring Framework se prefiera el uso de anotaciones en el código fuente, el archivo "web.xml" sigue siendo relevante, especialmente en entornos empresariales con aplicaciones más complejas y configuraciones detalladas.

Ejemplo básico de web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  version="4.0">

  <!-- Configuración de un servlet -->
  <servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>com.example.MyServlet</servlet-class>
  </servlet>

  <!-- Mapeo del servlet a una URL -->
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/myservlet</url-pattern>
  </servlet-mapping>
</web-app>
```

En este ejemplo:

- Se define un servlet llamado MyServlet con una clase asociada (com.example.MyServlet).

- Se realiza un mapeo del servlet a la URL /myservlet.

Este es un ejemplo muy básico y un archivo "web.xml" real podría incluir configuraciones adicionales para la gestión de sesiones, configuraciones de seguridad, filtros, entre otros. Es necesario ajustar las configuraciones según las necesidades específicas de la aplicación web.

8. Requerimientos del proceso de implantación de una aplicación web (Criterio 1.h)

El proceso de implantación de una aplicación web implica varios pasos esenciales, que deben planificarse cuidadosamente para garantizar el éxito del despliegue:

Evaluación de requisitos

Antes de la implementación, es fundamental asegurarse de que el entorno cumpla con las necesidades de la aplicación:

- **Hardware:** Verificar la capacidad de procesamiento, memoria y almacenamiento del servidor.
- **Software:** Instalar y configurar el sistema operativo, servidores web o de aplicaciones, y cualquier dependencia necesaria.

Configuración del entorno

El entorno de producción debe estar configurado para soportar cargas de trabajo reales y garantizar la seguridad:

- **Variables de entorno:** Configurar claves, rutas y parámetros que la aplicación necesita para operar.
- **Optimización del servidor:** Ajustar parámetros como el tamaño máximo de solicitudes, límites de tiempo de ejecución y asignación de memoria.

Gestión de dependencias

La instalación de bibliotecas, frameworks y módulos necesarios es crucial. En entornos como Python se utiliza pip, mientras que en Java se gestionan dependencias a través de herramientas como Maven o Gradle.

Despliegue y pruebas iniciales

Se recomienda realizar un despliegue inicial en un entorno de pruebas que refleje las condiciones del entorno de producción. Esto permite identificar posibles errores y solucionarlos antes de lanzar la aplicación al público.

9. Documentación de procesos de instalación y configuración realizados sobre los servidores web, de aplicaciones y sobre tecnologías de virtualización en la nube y en contenedores (Criterio 1.i)

La documentación es fundamental en el despliegue de aplicaciones web porque permite a los equipos mantener, replicar y solucionar problemas de la infraestructura de manera eficiente. Para el módulo profesional de "Despliegue de aplicaciones web", es crucial que los estudiantes entiendan cómo crear y usar esta documentación.

1. Documentación de Servidores Web y de Aplicaciones

Esta documentación se centra en los detalles de la instalación y configuración del software del servidor (como Apache, Nginx o Tomcat) en el entorno de producción.

- **Manual de Instalación:**
 - **Versiones de Software:** Especifica las versiones exactas del servidor web y de la aplicación (por ejemplo, Apache 2.4.52, Tomcat 9.0.58).
 - **Requisitos del Sistema:** Detalla la memoria RAM, el espacio en disco, la versión del sistema operativo y las dependencias necesarias.
 - **Pasos de Instalación:** Una guía paso a paso, desde la descarga del software hasta su instalación completa. Incluye los comandos de consola exactos.
- **Archivo de Configuración:**
 - **Parámetros Esenciales:** Documenta los parámetros críticos que se han modificado del archivo de configuración predeterminado (httpd.conf para Apache, server.xml para Tomcat).
 - **Configuración de Puertos:** Indica los puertos que el servidor está escuchando (por ejemplo, el puerto 80 para HTTP, el 443 para HTTPS).
 - **Configuración del Host Virtual:** Si se despliegan varias aplicaciones en el mismo servidor, se debe documentar cada host virtual, con su dominio, ruta de acceso y configuración específica.
 - **Configuración SSL/TLS:** Describe la ubicación de los certificados SSL, la clave privada y las directivas de seguridad para la encriptación.

2. Documentación de Servidores en la Nube y Contenedores

Esta parte de la documentación se enfoca en la infraestructura virtual y es clave para asegurar la portabilidad y la reproducibilidad del entorno.

- **Infraestructura en la Nube (AWS, Azure, GCP):**
 - **Plan de Instancias:** Detalla el tipo y tamaño de las máquinas virtuales (VM) que se van a utilizar (por ejemplo, "Instancia EC2 tipo t2.micro con Ubuntu 20.04").
 - **Grupo de Seguridad:** Documenta las reglas del firewall que se han configurado para permitir el tráfico de red necesario (por ejemplo, "Puerto 80 y 443 abiertos para el tráfico web").

- **Conexión y Acceso:** Proporciona las credenciales de acceso, como la clave .pem para la conexión SSH, y las instrucciones para acceder a la instancia remota.
- **Servicios Adicionales:** Si se utilizan servicios adicionales (bases de datos, almacenamiento de objetos), se debe documentar su configuración.
- **Contenedores (Docker):**
 - **Dockerfile:** Este es el documento más importante. Es un script que contiene todas las instrucciones para construir una imagen de Docker. Debe estar bien comentado para que sea fácil de entender.
 - FROM: Indica la imagen base.
 - RUN: Comandos para instalar dependencias.
 - COPY: Para copiar archivos de la aplicación al contenedor.
 - EXPOSE: Para especificar los puertos que el contenedor va a exponer.
 - CMD: El comando que se ejecuta al iniciar el contenedor.
 - **Docker Compose:** Un archivo YAML que se usa para definir y ejecutar aplicaciones multi-contenedor. Documenta qué servicios (services) componen la aplicación, sus imágenes, las variables de entorno, y el mapeo de puertos y volúmenes.

Ejemplos Prácticos de Documentación

Añadir ejemplos claros es la mejor manera de asegurar que la documentación se entienda y se use correctamente.

Ejemplo 1: Documentación de un Servidor Nginx

```
# Despliegue de la Aplicación de E-commerce
## Servidor Web: Nginx

**Versión:** 1.20.1

**Ruta de Configuración:** `/etc/nginx/nginx.conf` y `/etc/nginx/sites-available/`

**Configuración Clave:**

* **Archivo:** `/etc/nginx/sites-available/ecommerce.conf`
* **Contenido:**
  ```nginx
 server {
 listen 80;
 server_name tienda.ejemplo.com;
 root /var/www/html/ecommerce;
 index index.html index.htm;
 ...
 }
  ```

**Notas:** Se ha configurado un `proxy_pass` al servidor de aplicaciones en el puerto 8080.
```


Ejemplo 2: Documentación de un Dockerfile

```
# Usa la imagen base oficial de Python 3.9
FROM python:3.9-slim

# Establece el directorio de trabajo dentro del contenedor
WORKDIR /usr/src/app

# Copia los archivos de requerimientos e instala las dependencias
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# Copia el resto del código de la aplicación
COPY . .

# Expone el puerto que la aplicación Flask va a usar
EXPOSE 5000

# Define el comando para ejecutar la aplicación
CMD ["python", "./app.py"]
```

En resumen, la documentación es el mapa de ruta de la infraestructura. No solo ayuda a los administradores de sistemas a mantener los servidores, sino que también facilita la colaboración y la transferencia de conocimiento entre los miembros del equipo.