Módulo 8
Fundamentos de integración continua

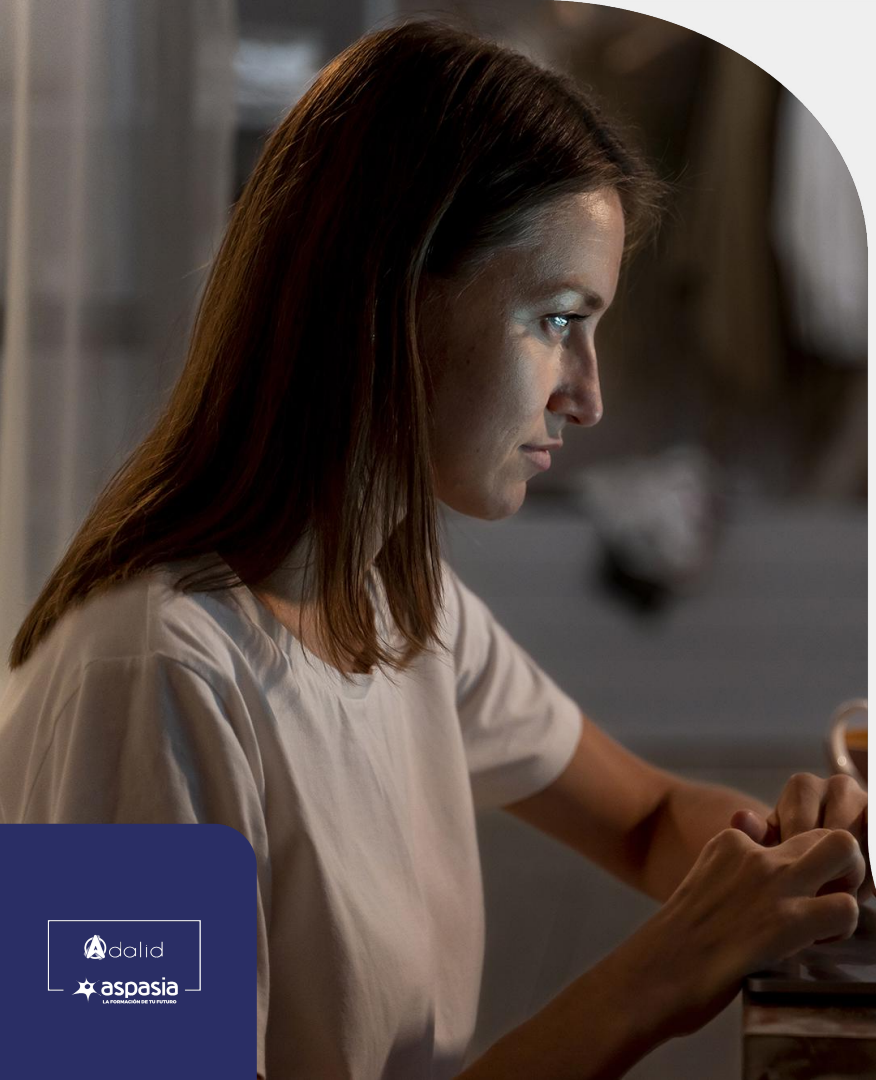
# Fundamentos de DevOps

## Módulo 8

# AE 1.1

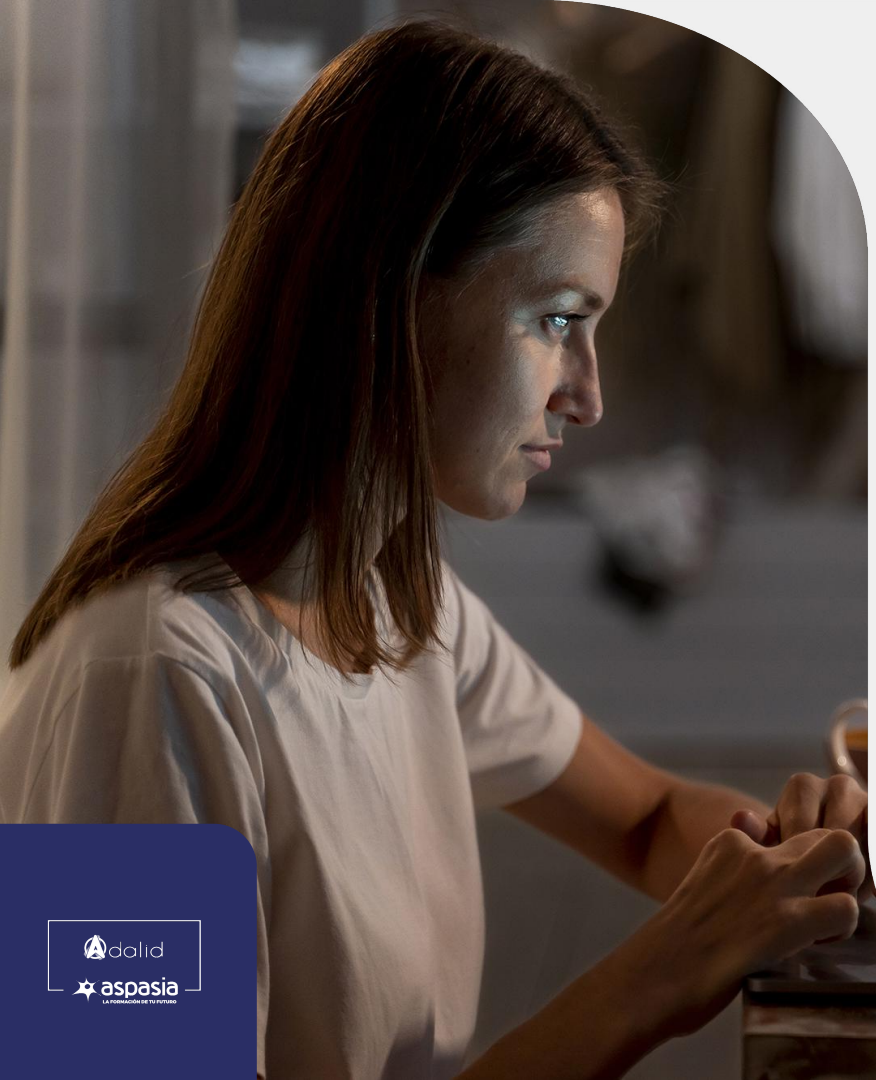
## OBJETIVOS

**Entender qué es DevOps, su cultura, principios y beneficios en el desarrollo de software. Conocer la importancia de la Integración y Entrega Continua en la automatización del ciclo DevOps. Aprender el uso de contenedores Docker y su papel en la infraestructura ágil y escalable.**



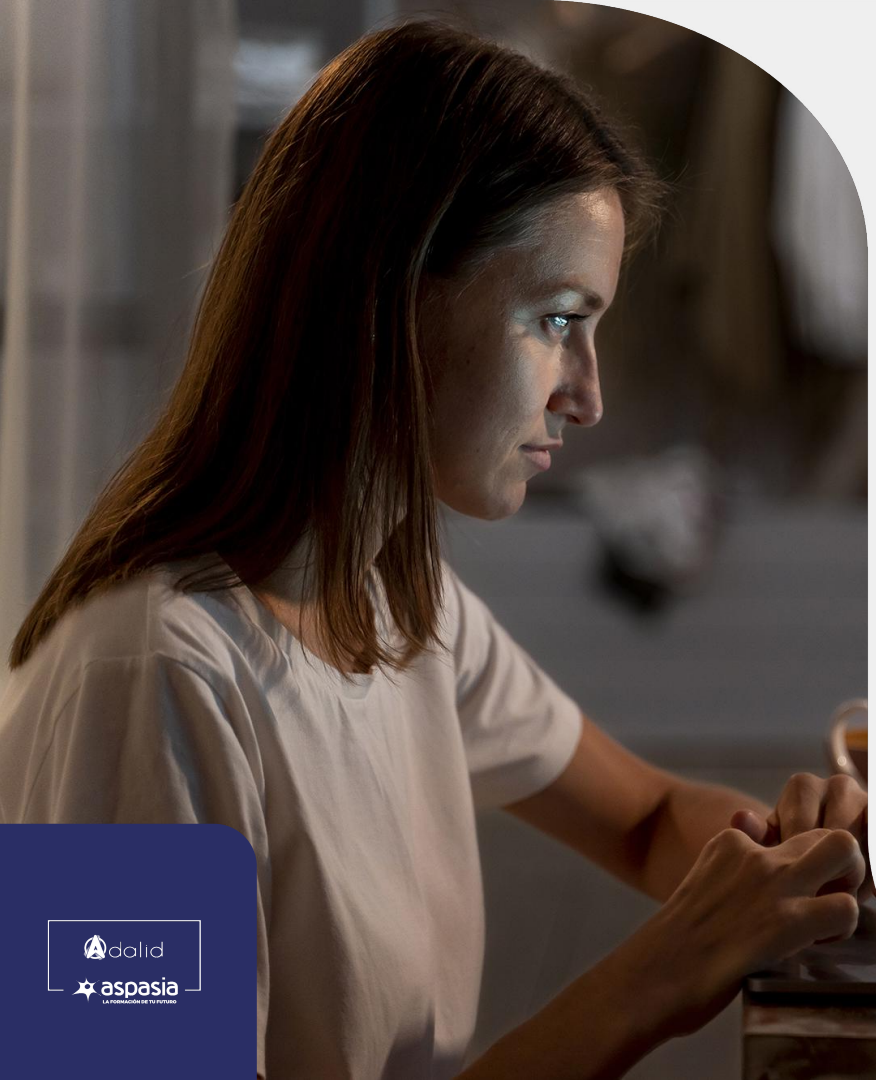
# ¿QUÉ VAMOS A VER?

- Fundamentos de DevOps y Cultura.
  - Qué es DevOps, su propósito, origen y evolución.
  - Cultura, principios y beneficios de DevOps.
  - Modelo CAMS (Culture, Automation, Measurement, Sharing).
  - DevOps vs. DevSecOps y ciclo de vida DevOps.



# ¿QUÉ VAMOS A VER?

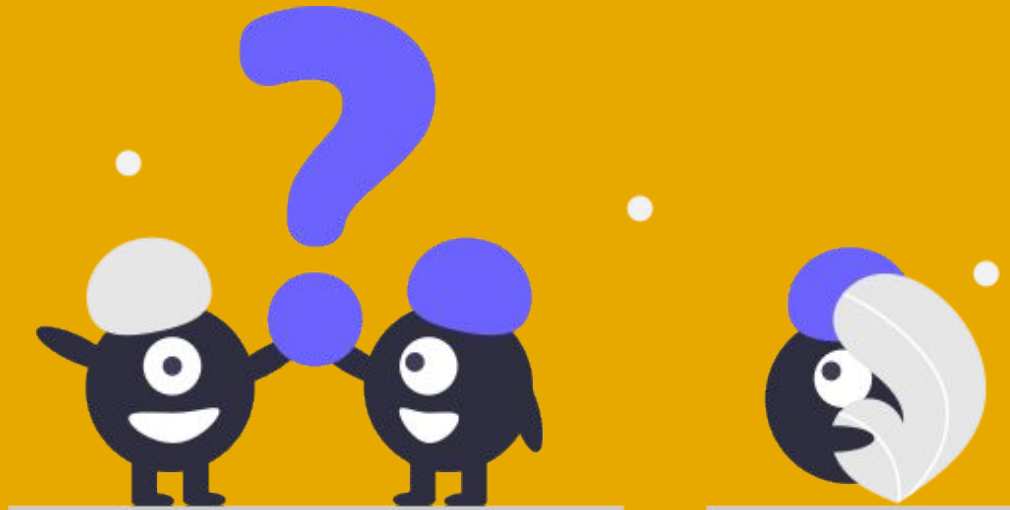
- Integración y Entrega Continua.
  - Qué es Integración Continua y diferencias con Entrega Continua.
  - Flujo del proceso de Integración Continua: control de versiones, compilación y testing.
  - Sistemas de integración continua: Jenkins, Circle CI, GitLab CI, Bamboo.

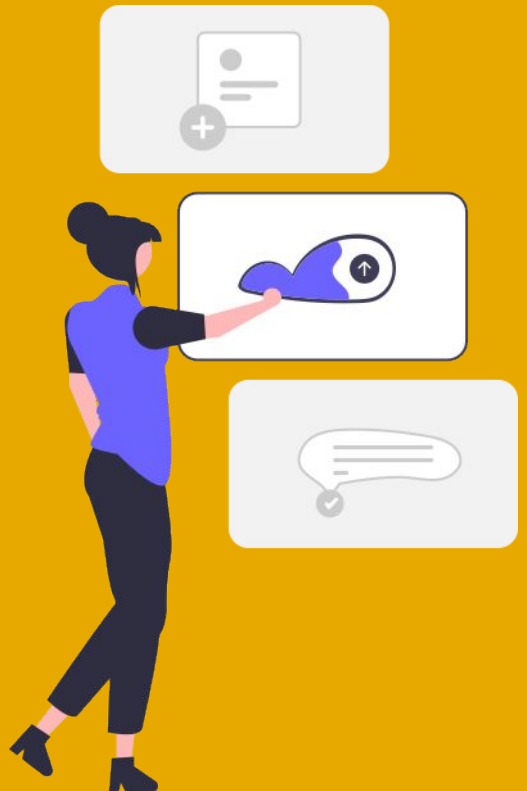


# ¿QUÉ VAMOS A VER?

- Contenedores de Aplicaciones y Docker.
  - Qué es un contenedor de aplicaciones y diferencias con una máquina virtual.
  - Conceptos básicos de Docker: imágenes, DockerFile, contenedores y volúmenes.
  - Beneficios y rol de Docker en el ciclo DevOps.
  - Implementación de Docker: instalación, configuración, comandos básicos y monitoreo de eventos/logs.

# ¿Que hace un DevOps?





# Fundamentos de DevOps y Cultura.

adalid
aspasia

# ¿Qué es DevOps y cuál es su propósito?

DevOps es una cultura y conjunto de prácticas que **automatizan e integran** los procesos entre equipos de desarrollo (Dev) y operaciones (Ops) para entregar software más rápido y confiable.

**Propósito:**

- Mejorar la colaboración entre equipos de desarrollo y operaciones.
- Automatizar procesos de entrega y despliegue de software.
- Asegurar software de alta calidad con menos errores en producción.

# Origen y evolución de DevOps

DevOps surgió como respuesta a problemas tradicionales del desarrollo de software:

✅ **Con DevOps:**

- Desarrollo y operaciones trabajan juntos desde el inicio.
- Se automatizan pruebas y despliegues.
- Se reducen tiempos de entrega y errores en producción.

❌ **Antes de DevOps:**

- Los desarrolladores escribían código, pero no participaban en el despliegue.
- El equipo de operaciones se encargaba de la infraestructura, pero sin comunicación con los desarrolladores.
- Los cambios eran lentos y propensos a errores.

# Origen y evolución de DevOps

DevOps evolucionó para resolver problemas de entrega lenta y poca colaboración.

# Cultura y principios de DevOps

**4 Principios Clave de DevOps:**

- **Colaboración y Comunicación:** Equipos de Dev y Ops trabajan juntos.
- **Automatización:** Desde la integración hasta el despliegue.
- **Medición:** Monitoreo constante del software y su rendimiento.
- **Compartición:** Transparencia en los procesos y conocimiento compartido.

# Beneficios de DevOps

## ¿Por qué adoptar DevOps?

- **Entrega más rápida** de software.
- **Menos errores en producción** gracias a pruebas automatizadas.
- **Mayor colaboración** entre equipos.
- **Mejor seguridad** y monitoreo de aplicaciones.

# Modelo CAMS (Culture, Automation, Measurement, Sharing)

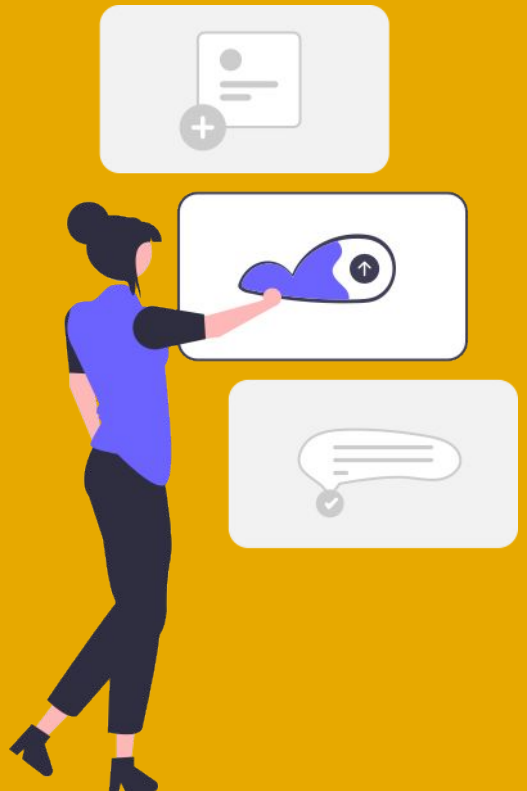
## CAMS define los pilares de DevOps:

| Elemento | Descripción |
|---|---|
| **Culture (Cultura)** | Fomenta la colaboración y la mejora continua. |
| **Automation (Automatización)** | Reduce tareas manuales y errores humanos. |
| **Measurement (Medición)** | Se monitorea el rendimiento y los despliegues. |
| **Sharing (Compartición)** | Se comparten aprendizajes y herramientas. |

# DevOps vs. DevSecOps

## DevSecOps = DevOps + Seguridad desde el inicio

| Diferencia | DevOps | DevSecOps |
|---|---|---|
| **Enfoque** | Integración y despliegue continuo | Seguridad integrada en el ciclo de desarrollo |
| **Pruebas** | Se enfocan en funcionalidad | Incluyen análisis de vulnerabilidades |
| **Automatización** | CI/CD | CI/CD + Seguridad |



# Integración Continua / Entrega Continua

# ¿Qué es Integración Continua?

La **Integración Continua (CI)** es una práctica donde los desarrolladores **integran código frecuentemente** en un repositorio compartido, permitiendo detectar errores temprano.

**Beneficios:**

- Detecta errores de integración rápidamente.
- Automatiza pruebas para mejorar calidad.
- Reduce conflictos de código.

# Despliegue Continuo vs Entrega Continua

Entrega Continua requiere aprobación, mientras que Despliegue Continuo es 100% automatizado.

| Concepto | Entrega Continua (Continuous Delivery) | Despliegue Continuo (Continuous Deployment) |
|---|---|---|
| **Objetivo** | Código listo para producción | Código se despliega automáticamente |
| **Aprobación manual** | Sí | No |
| **Automatización** | Pruebas y entregas automatizadas | Todo el proceso hasta producción automatizado |

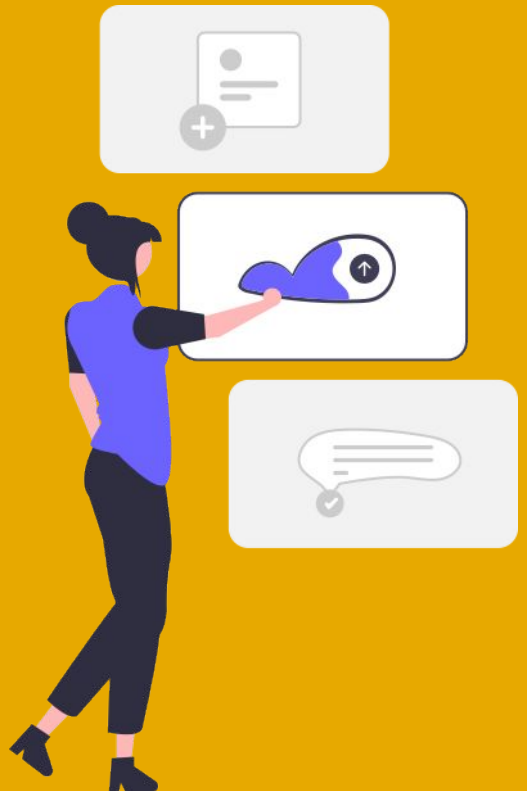
# Flujo del Proceso de Integración Continua

**Pasos Clave:**

1. **Control de Versiones:** Uso de Git para gestionar cambios.
2. **Compilación:** Se construye el código automáticamente.
3. **Testing:** Se ejecutan pruebas automatizadas.

# Sistemas de Integración Continua

| Sistema | Características |
| --- | --- |
| **Jenkins** | Open-source, flexible y extensible. |
| **Circle CI** | Basado en la nube, fácil de integrar. |
| **GitLab CI/CD** | Integrado con GitLab, ideal para repos privados. |
| **GitHub CI/CD** | Integrado con GitHub. |
| **Bamboo** | Potente pero pago, desarrollado por Atlassian. |



# Fundamentos de Contenedores de Aplicaciones

# ¿Qué es un Contenedor de Aplicaciones?

Un **contenedor de aplicaciones** es un entorno **ligero y portátil** que encapsula una aplicación junto con sus dependencias, asegurando que funcione de manera consistente en diferentes entornos.

**Beneficios:**

- **Portabilidad:** Se ejecuta en cualquier sistema que tenga Docker.
- **Aislamiento:** Cada contenedor es independiente de los demás.
- **Escalabilidad:** Fácilmente replicables en producción.
- **Eficiencia:** Usa menos recursos que las máquinas virtuales.

# Diferencias entre un Contenedor y una Máquina Virtual

| Característica | Contenedores | Máquinas Virtuales (VMs) |
|---|---|---|
| **Consumo de recursos** | Ligero | Pesado |
| **Arranque** | Rápido (segundos) | Lento (minutos) |
| **Aislamiento** | A nivel de proceso | A nivel de sistema operativo |
| **Uso de SO** | Comparte el kernel del host | Cada VM tiene su propio SO |
| **Escalabilidad** | Alta | Menos flexible |

# El Contenedor Docker - Conceptos Básicos

Un **contenedor de aplicaciones** es un entorno **ligero y portátil** que encapsula una aplicación junto con sus dependencias, asegurando que funcione de manera consistente en diferentes entornos.

**Beneficios:**

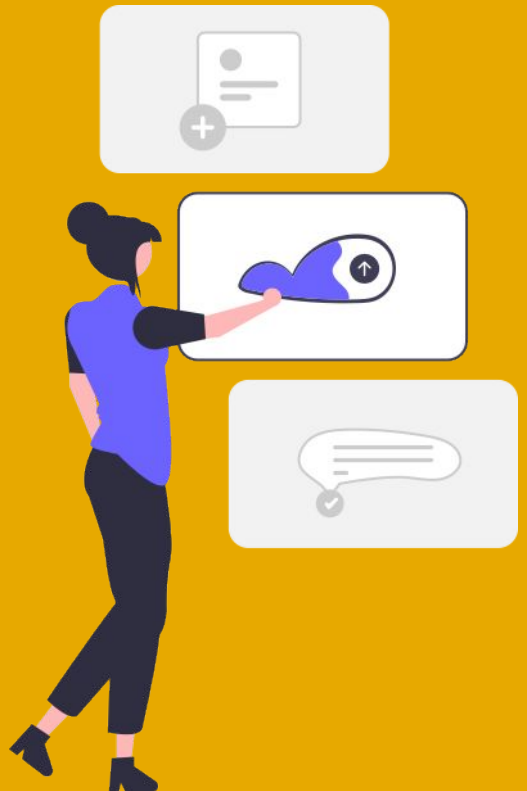
- **Portabilidad:** Se ejecuta en cualquier sistema que tenga Docker.
- **Aislamiento:** Cada contenedor es independiente de los demás.
- **Escalabilidad:** Fácilmente replicables en producción.
- **Eficiencia:** Usa menos recursos que las máquinas virtuales.

# Beneficios de Usar Docker

- **Independencia del entorno:** No importa el sistema operativo.
- **Estandarización:** Funciona igual en desarrollo, pruebas y producción.
- **Reducción de conflictos:** Adiós al “funciona en mi máquina”.
- **Mayor productividad:** Entornos listos en segundos.

# Rol del Contenedor Docker en el Ciclo DevOps

| Fase DevOps | Uso de Docker |
| --- | --- |
| **Desarrollo** | Contenedores con dependencias preconfiguradas. |
| **Integración Continua** | Se ejecutan pruebas en contenedores limpios. |
| **Entrega Continua** | Se empaqueta el código en imágenes listas para producción. |
| **Despliegue** | Se escalan contenedores automáticamente. |



# Implementación de Docker

adalid
aspasia

# Instalación y Configuración de Docker

**Instalación:**

- **Windows / Mac:** Descargar Docker Desktop https://www.docker.com/ y seguir los pasos de instalación.
- **Linux:**

```
sudo apt update
sudo apt install docker.io

docker --version # Verificar instalación
```

# Comandos Administrativos Básicos

| Comando | Función |
| --- | --- |
| `docker pull <imagen>` | Descargar una imagen de DockerHub. |
| `docker run <imagen>` | Ejecutar un contenedor. |
| `docker ps` | Ver contenedores en ejecución. |
| `docker stop <id>` | Detener un contenedor. |
| `docker rm <id>` | Eliminar un contenedor. |
| `docker images` | Listar imágenes disponibles. |

**Ejemplo:** Ejecutar un servidor Nginx en Docker:

```
docker run -d -p 8080:80 nginx
```

# Utilización de Imágenes Docker

**¿Cómo se crean imágenes personalizadas?**

- Se usa un **Dockerfile** con las instrucciones.
- Se genera la imagen con **docker build**.

# Utilización de Imágenes Docker

## Ejemplo de Dockerfile para una Aplicación Node.js:

```
FROM node:16
WORKDIR /app
COPY . .CMD ["node", "server.js"]
EXPOSE 3000

RUN npm install
```

## Construcción y Ejecución:

```
docker build -t miapp .
docker run -p 3000:3000 miapp
```

# Monitoreo de Eventos y Logs en Docker

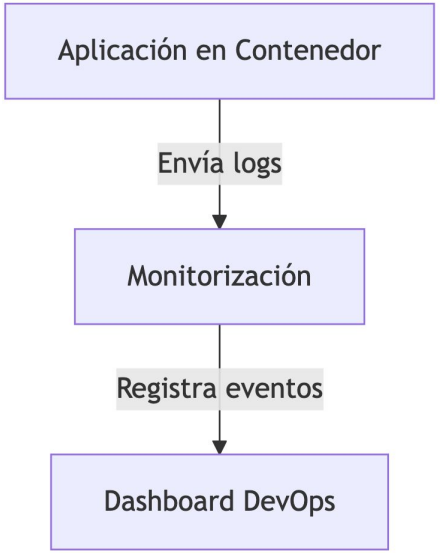
## Ver logs en tiempo real:

```
docker logs -f <container_id>
```

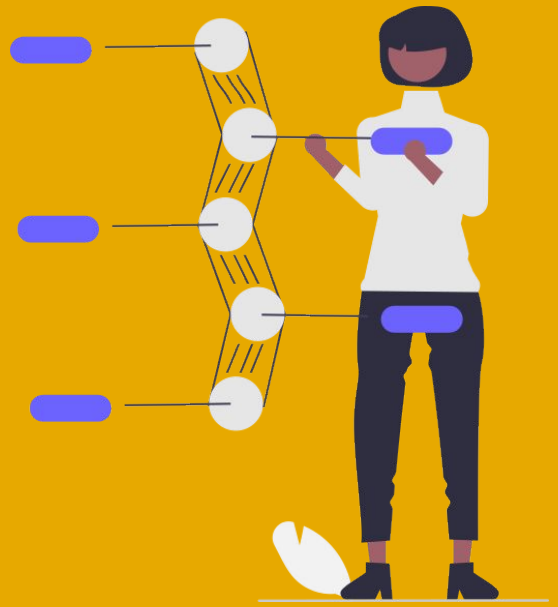
## Ver consumo de recursos:

```
docker stats
```

## Ejemplo de monitoreo en producción:



# Resumen de lo aprendido 🧑‍🎓



Adalid aspasia

# Resumen de lo aprendido

- DevOps fomenta la colaboración, automatización y monitoreo para mejorar el desarrollo y operaciones.
- La Integración y Entrega Continua optimizan despliegues frecuentes y confiables en software.
- Docker permite empaquetar y ejecutar aplicaciones de forma aislada, eficiente y portable.
- Contenedores facilitan la escalabilidad y consistencia en entornos DevOps modernos.

# GRACIAS POR TU ATENCIÓN

Nos vemos en la próxima clase

Chile
Adalid
aspasia
LA FORMACIÓN DE TU FUTURO