

Desarrollo de Software

ACTIVIDAD 2

Integrantes:

- Guido Anthony Chipana Calderon
- Diego Manuel Delgado Velarde

Fecha: 31 Marzo 2025

ÍNDICE

- Infraestructura como Código
- Contenerización y despliegue de aplicaciones modernas
- Observabilidad y Troubleshooting
- CI/CD



Infraestructura como Código

INTRODUCCIÓN A IAC

La infraestructura como código (IaC) es uno de los fundamentos en la ingeniería de software contemporánea y trata la infraestructura de TI(servidores, redes, sistemas de almacenamiento, contenedores, balanceadores de carga y otros recursos necesarios para ejecutar aplicaciones)de la misma manera que el código fuente de una aplicación.

Beneficios: Consistencia en la configuración, control de versiones, automatización y reducción de errores humanos.



ESCRITURA DE IAC

Terraform es una herramienta de infraestructura como código que permite definir, desplegar y gestionar infraestructuras de manera automatizada.

Utiliza HCL para definir recursos como instancias EC2, bases de datos o redes virtuales.

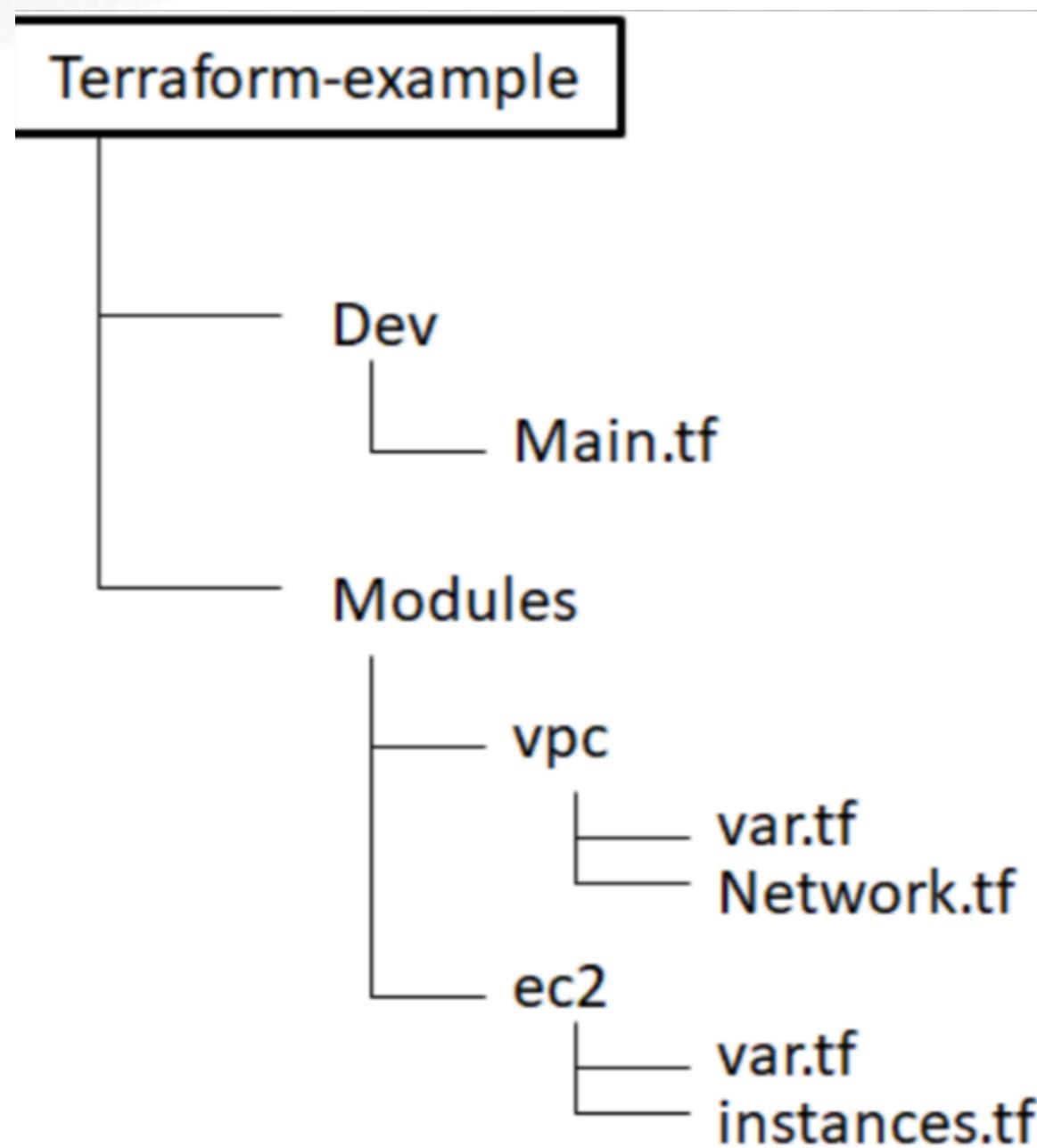
Buenas prácticas: Nombres claros de recursos, uso de variables, modularización del código, uso de repositorios de control de versiones (Git).

```
variable "nombre_de_la_aplicacion" {
    type      = string
    description = "Nombre descriptivo para la aplicación"
    default    = "mi_aplicacion"
}

variable "instance_type" {
    type      = string
    description = "Tipo de instancia"
    default    = "t2.micro"
}

resource "aws_instance" "app_server" {
    ami          = "ami-12345678"
    instance_type = var.instance_type
    tags = {
        Name = var.nombre_de_la_aplicacion
    }
}
```

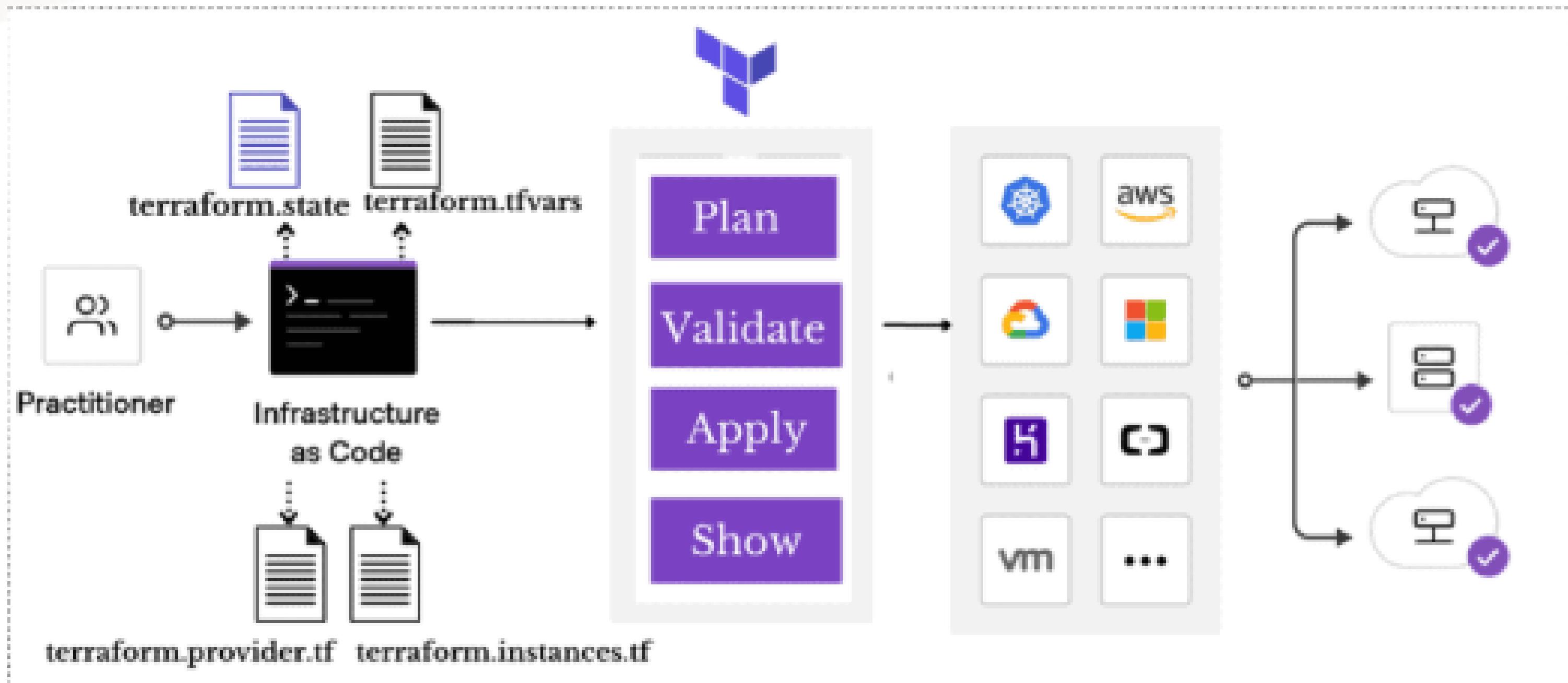
PATRONES PARA MÓDULOS



```
project/
└── modules/
    ├── network/
    │   ├── main.tf
    │   ├── variables.tf
    │   ├── outputs.tf
    └── database/
        ├── main.tf
        ├── variables.tf
        ├── outputs.tf
    └── application/
        ├── main.tf
        ├── variables.tf
        ├── outputs.tf
    └── main.tf
    └── variables.tf
    └── terraform.tfvars
```

TAREA TEÓRICA:

- INVESTIGAR UNA HERRAMIENTA DE IAC (P. EJ. TERRAFORM) Y DESCRIBIR CÓMO ORGANIZA SUS MÓDULOS.

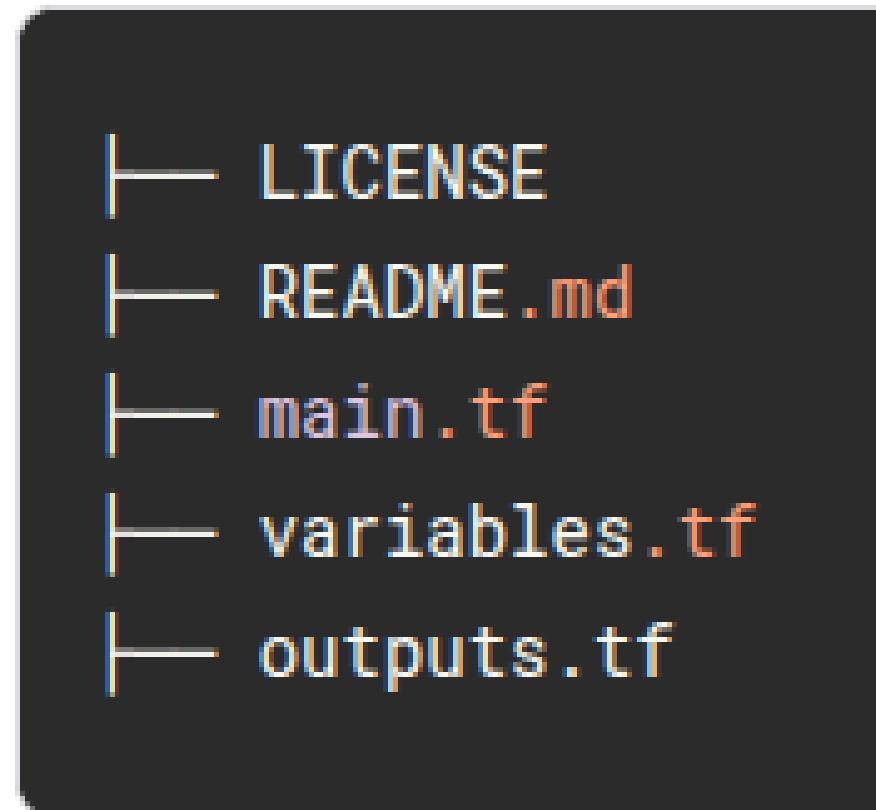


Estructura convencional y descripción de como se organiza cada modulo

main.tf: Contiene el conjunto principal de parámetros de configuración para el módulo. También se puede crear otros archivos de configuración y organizarlos adecuadamente.

variables.tf: Contiene las definiciones de variables para el módulo. Cuando otras personas usan tu módulo, las variables se configurarán como argumentos en el bloque de módulo.

outputs.tf: Contiene las definiciones de salida para el módulo. A menudo, se usan para pasar información sobre las partes de tu infraestructura que define el módulo a otras partes de tu configuración.



Proyecto hipotético que incluye tres módulos: network, database y application

- La carpeta modules agrupa lógicamente modulos con funcionales específicas que se pueden comunicar mediante inputs y outputs.
- La carpeta environments contiene configuraciones específicas para cada entorno (dev/prod).
- Los archivos raíz (main.tf, variables.tf) integran los módulos y definen configuraciones globales.

```
project/
└── modules/
    ├── network/
    │   ├── main.tf
    │   ├── variables.tf
    │   └── outputs.tf
    ├── database/
    │   ├── main.tf
    │   ├── variables.tf
    │   └── outputs.tf
    └── application/
        ├── main.tf
        ├── variables.tf
        └── outputs.tf
└── environments/
    ├── dev/
    │   └── terraform.tfvars
    └── prod/
        └── terraform.tfvars
└── main.tf
└── variables.tf
└── terraform.lock.hcl
```

TAREA TEORICA

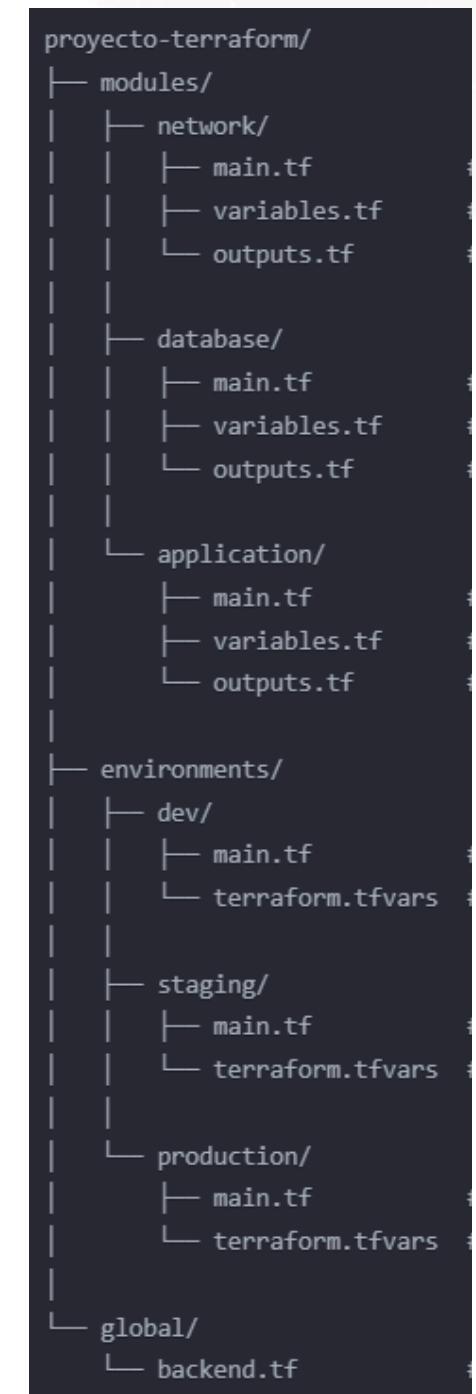
INVESTIGACION DE UNA HERRAMIENTA DE IAC Y COMO ORGANIZA SUS MODULOS

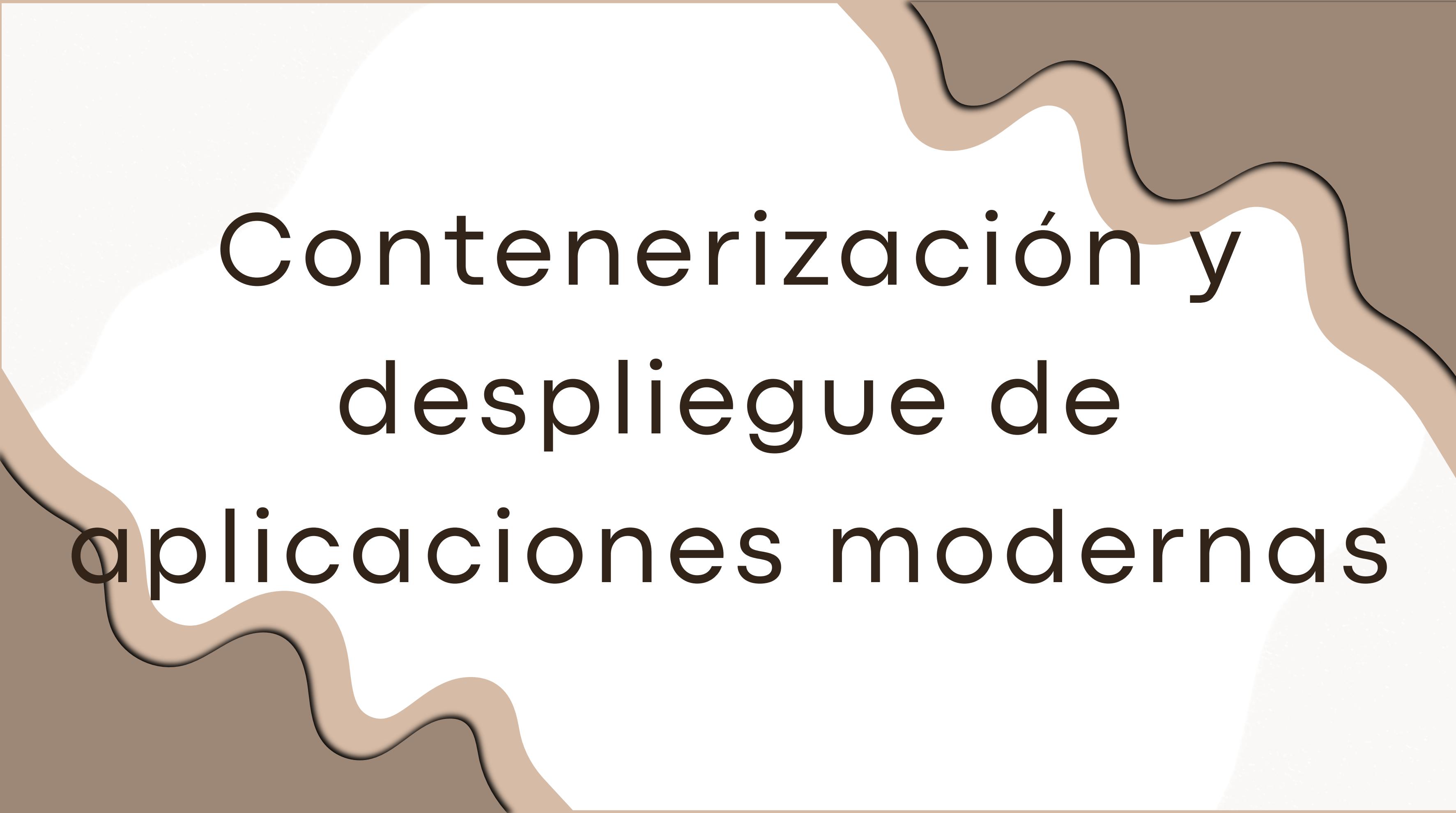
Terraform organiza sus modulos de las siguiente manera;

- Terraform utiliza el lenguaje HCL para definir recursos de forma declarativa
- Los modulos en Terraform son unidades reutilizables de archivos de configuracion
- Cada modulo puede tener sus propias variables de entrada (variables.tf), salidas (outputs.tf) y recursos (main.tf)
- Los modulos pueden ser consumidos por otros modulos mediante referencias, asi creado una estructura jerarquica
- Terraform gestiona un estado centralizado que permite los cambios y planificarlos mediante comandos terraform plan

ESTRUCTURA PROPUESTA PARA UN PROYECTO CON TRES MODULOS:

- La carpeta “modules” contiene la logica reutilizable, separando las capas de infraestructura
- El modulo “network” es fundamental ya que otros modulos dependeran de este
- El modulo “database” depende de la red pero es independiente de la aplicacion
- El modulo “application” es el consumidor final ya que depende de la red y de la base de datos
- La carpeta “environments” permite aplicar la misma configuracion a distintos entornos
- “dev”, “staging” y “production” sigue el principio de reproducibilidad





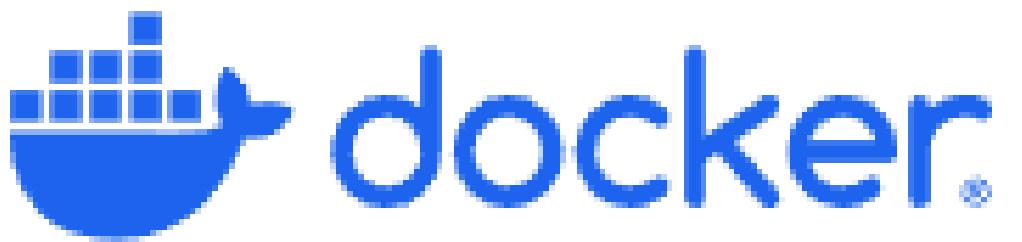
Containerización y
despliegue de
aplicaciones modernas

Contenerización de una aplicación con Docker

Docker simplifica el despliegue, escalado y gestión de aplicaciones mediante la contenedorización.

Un contenedor es un entorno ligero, portátil que incluye todo lo necesario para ejecutar una aplicación en distintos sistemas, por ejemplo sin tener que preocuparme por las dependencias.

A diferencia de las (VM), docker sólo virtualiza la capa de aplicación, además se ejecuta de forma náutica sobre el kernel del anfitrión, por ello se inicia más rápido y consume menos recursos.



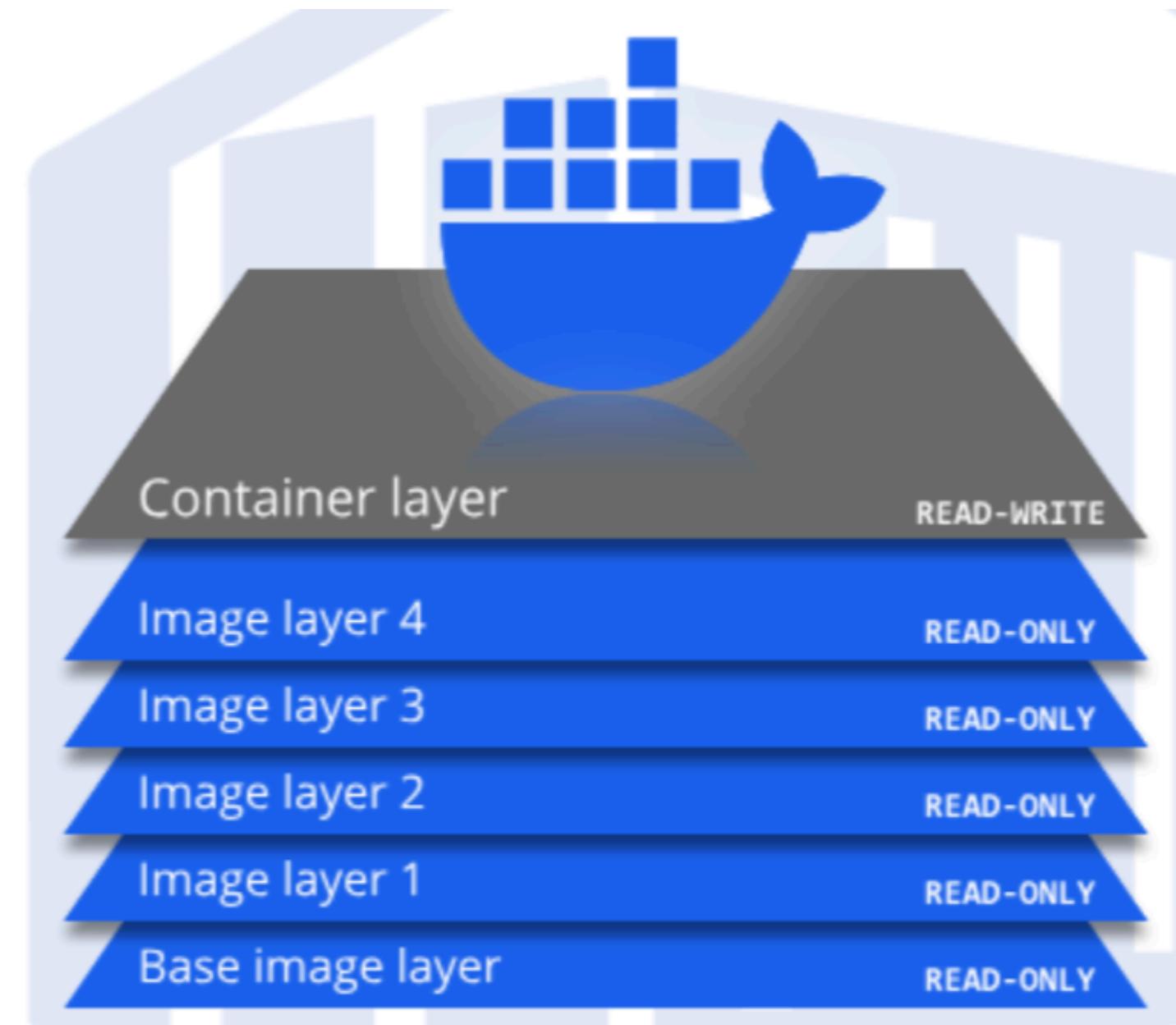
Contenerización de una aplicación con Docker

Estructura básica de un Dockerfile:

```
↳ Dockerfile > ...
1 # Establece la imagen base de la que partirá el contenedor
2 FROM ubuntu:18.04
3
4
5 # Actualiza la lista de paquetes y instala Nginx
6 RUN apt-get update && apt-get install -y nginx
7
8 # Copia el archivo de configuración de Nginx a su destino correcto
9 COPY nginx.conf /etc/nginx/nginx.conf
10
11 # Expone el puerto 80 para poder acceder a Nginx desde el exterior
12 EXPOSE 80
13
14 # Especifica el punto de entrada de la aplicación
15 ENTRYPOINT ["nginx"]
16
17 # Especifica los comandos que se deben ejecutar cuando se inicie el contenedor
18 CMD ["-g", "daemon off;"]
19
```

Contenerización de una aplicación con Docker

Dockerfile es un archivo, mientras que la imagen es el resultado de construir un dockerfile.



Contenerización de una aplicación con Docker

Docker Compose es una herramienta esencial para gestionar aplicaciones multicontenedor. En un proyecto de ciencia de datos, puedes necesitar contenedores separados para distintos componentes, como un cuaderno Jupyter, una base de datos y una herramienta de visualización de datos. Docker Compose te permite definir estos servicios en un único archivo YAML, lo que facilita la puesta en marcha de todo tu entorno de ciencia de datos con un solo comando.

```
version: '3.8'
services:
  jupyter:
    image: jupyter/scipy-notebook:latest
    volumes:
      - ./notebooks:/home/joelwembo/work
    ports:
      - "8888:8888"
    environment:
      - JUPYTER_ENABLE_LAB=yes

  postgres:
    image: postgres:13-alpine
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
      POSTGRES_DB: mydatabase
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

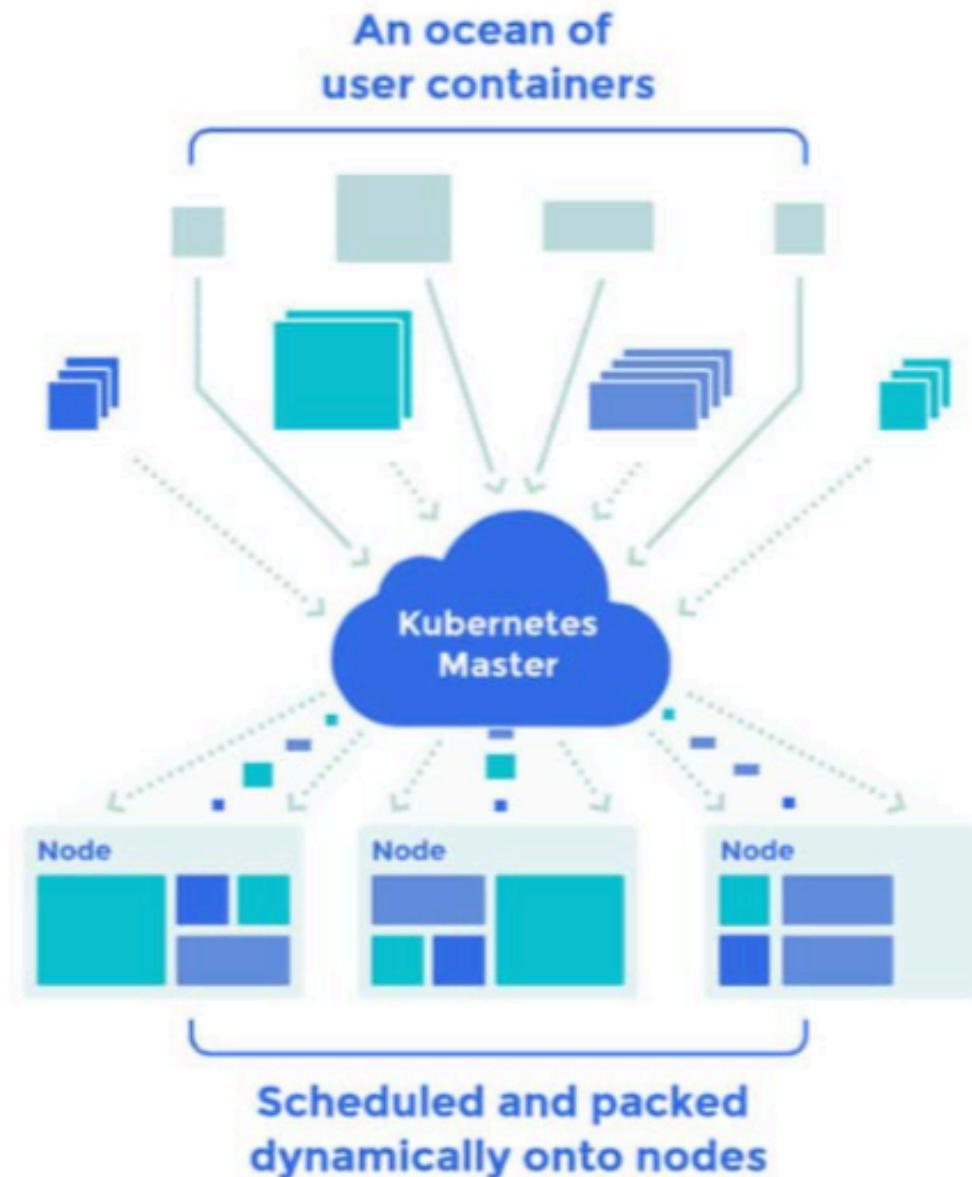
  redis:
    image: redis:alpine
    ports:
      - "6379:6379"

volumes:
  postgres_data:
```

Orquestación con Kubernetes

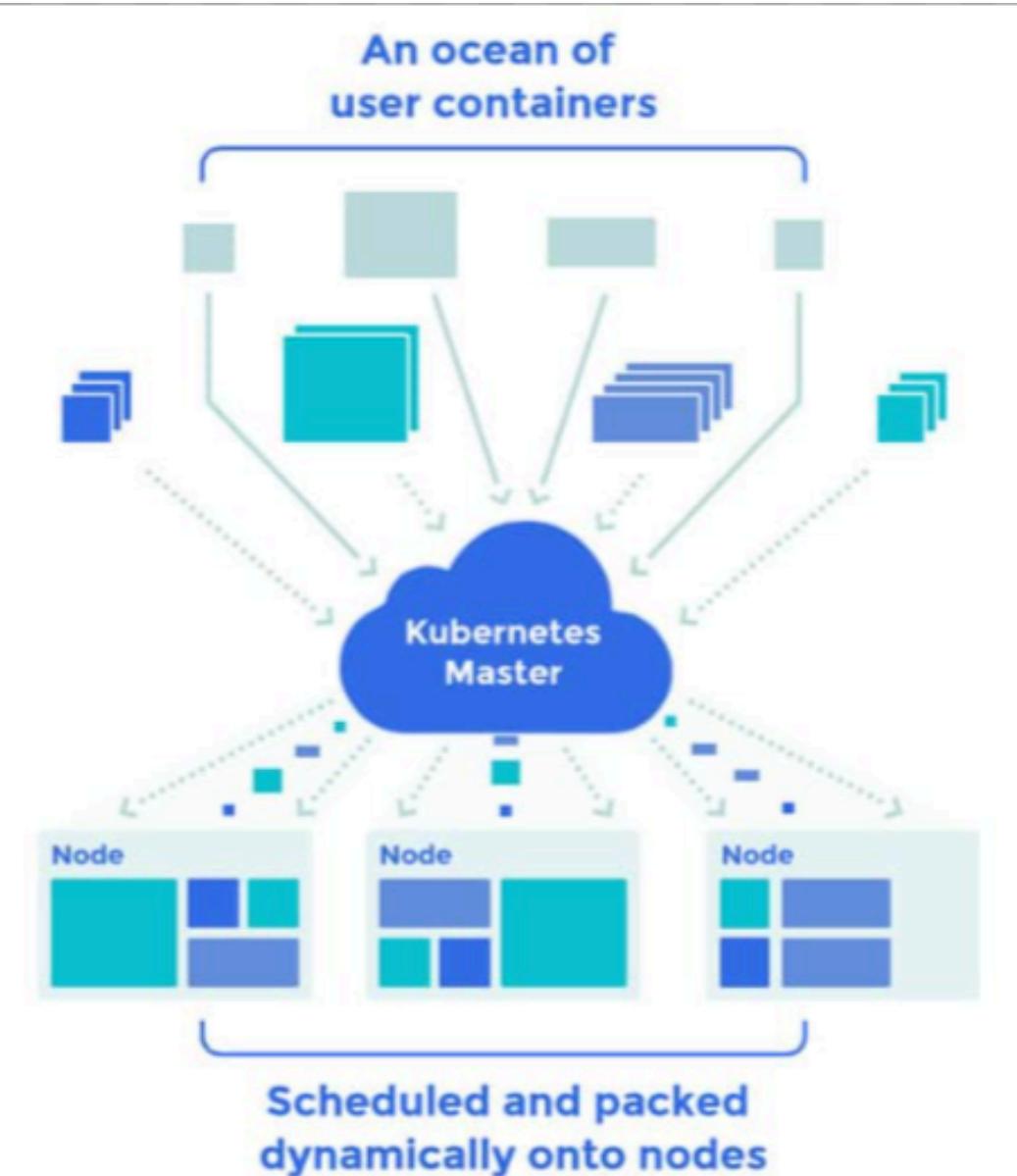
Kubernetes organiza los contenedores en unidades llamadas Pods, que pueden estar compuestos por uno o varios contenedores, y se gestiona a través de recursos como ReplicaSets y Deployments.

- Pods: Representan instancias de aplicación individuales.
- Despliegues: Gestiona el ciclo de vida de los Pods y asegúrate de que están sanos.
- Servicios: Proporciona un acceso estable a los Pods y permite la comunicación externa o interna.



Orquestación con Kubernetes

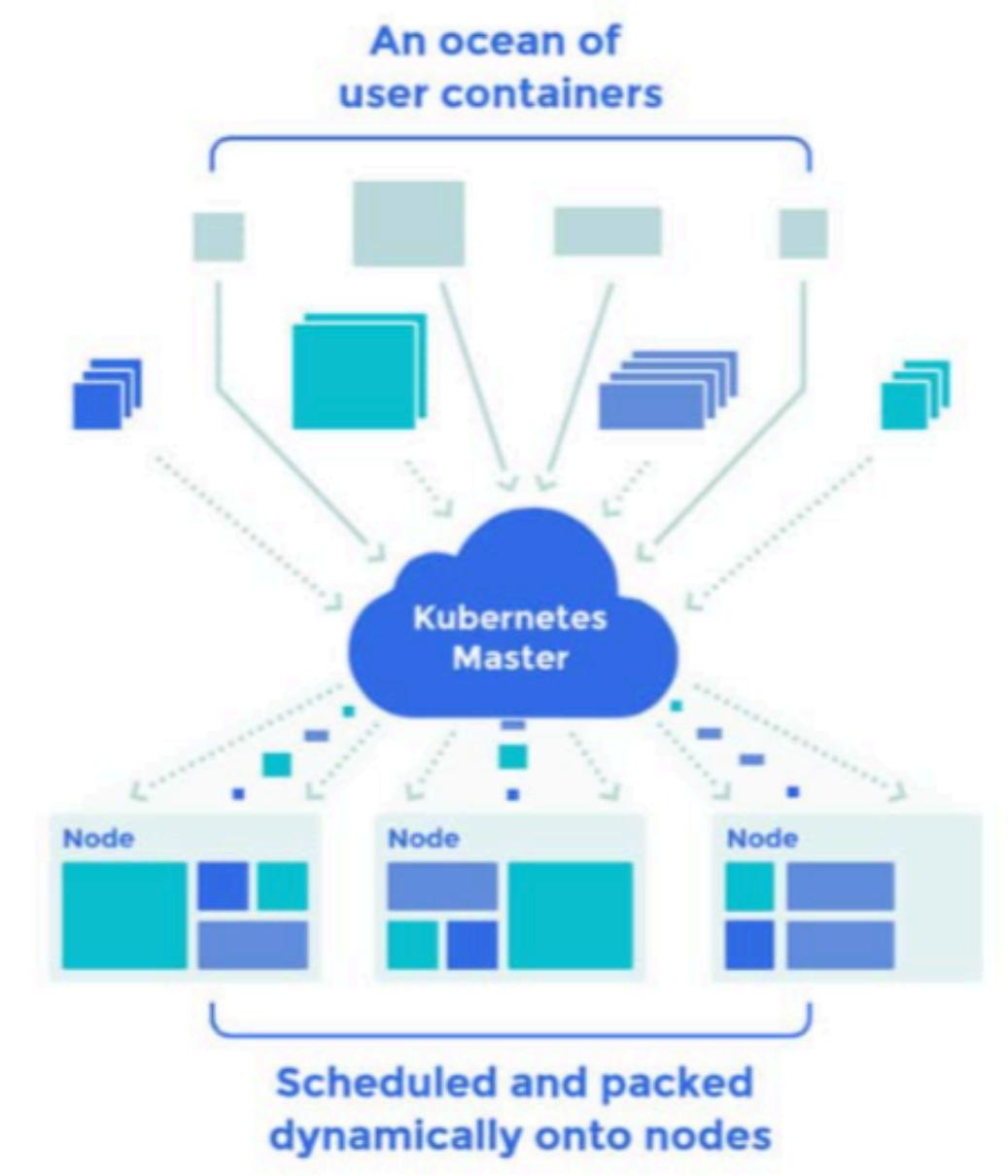
- Rolling Updates: Ideal para despliegues continuos donde la disponibilidad es crítica y se desea minimizar el impacto de cambios pequeños y progresivos.
- Canary Releases: Permite validar una nueva versión con una muestra real de tráfico antes de un despliegue masivo.
- Blue-Green Deployments: Proporciona un entorno de respaldo completo, uno "Blue" en producción y otro "Green" listo para recibir la nueva versión, facilitando reversión inmediata, pero puede implicar mayores costos operativos.



Orquestación con Kubernetes

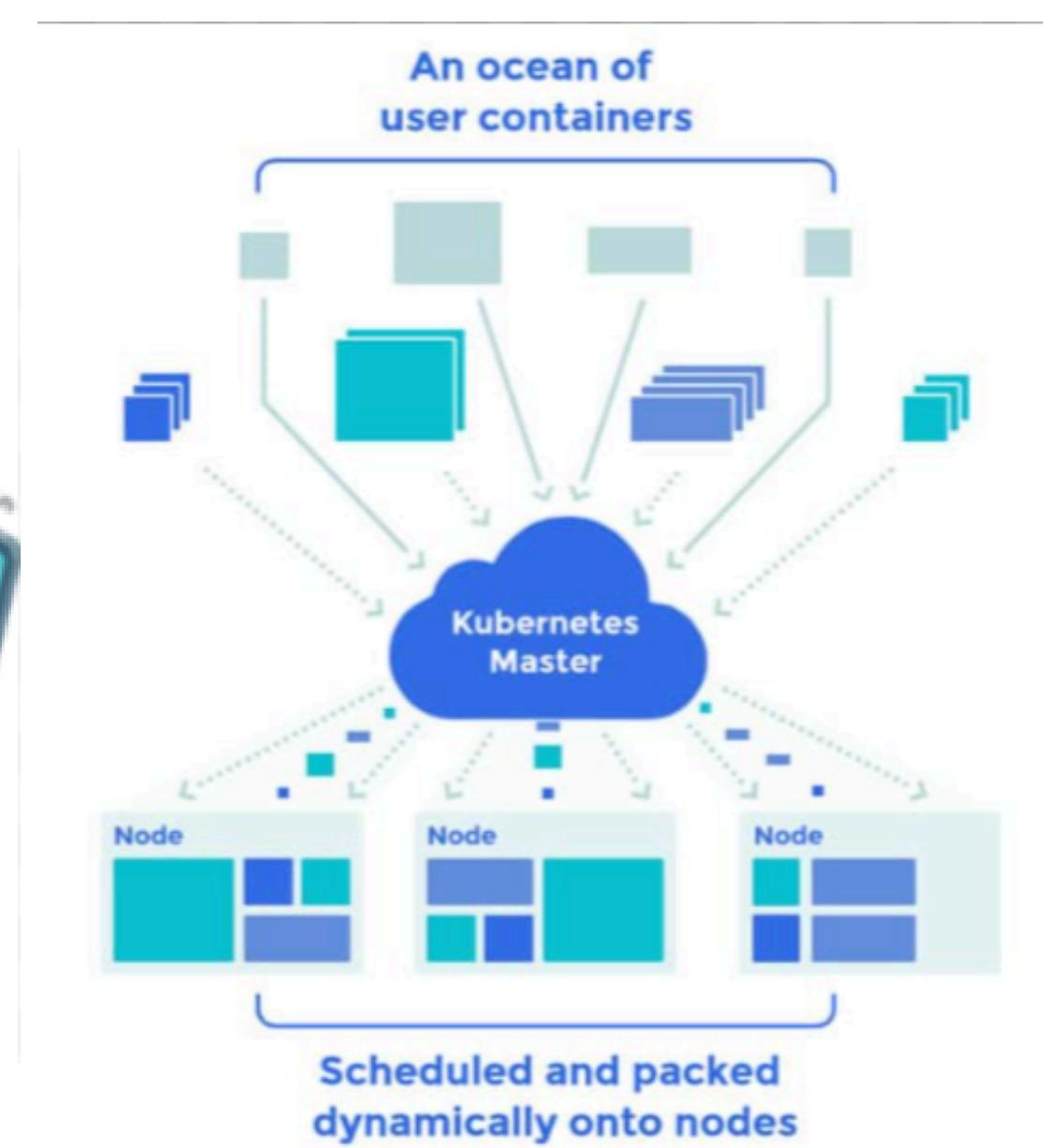
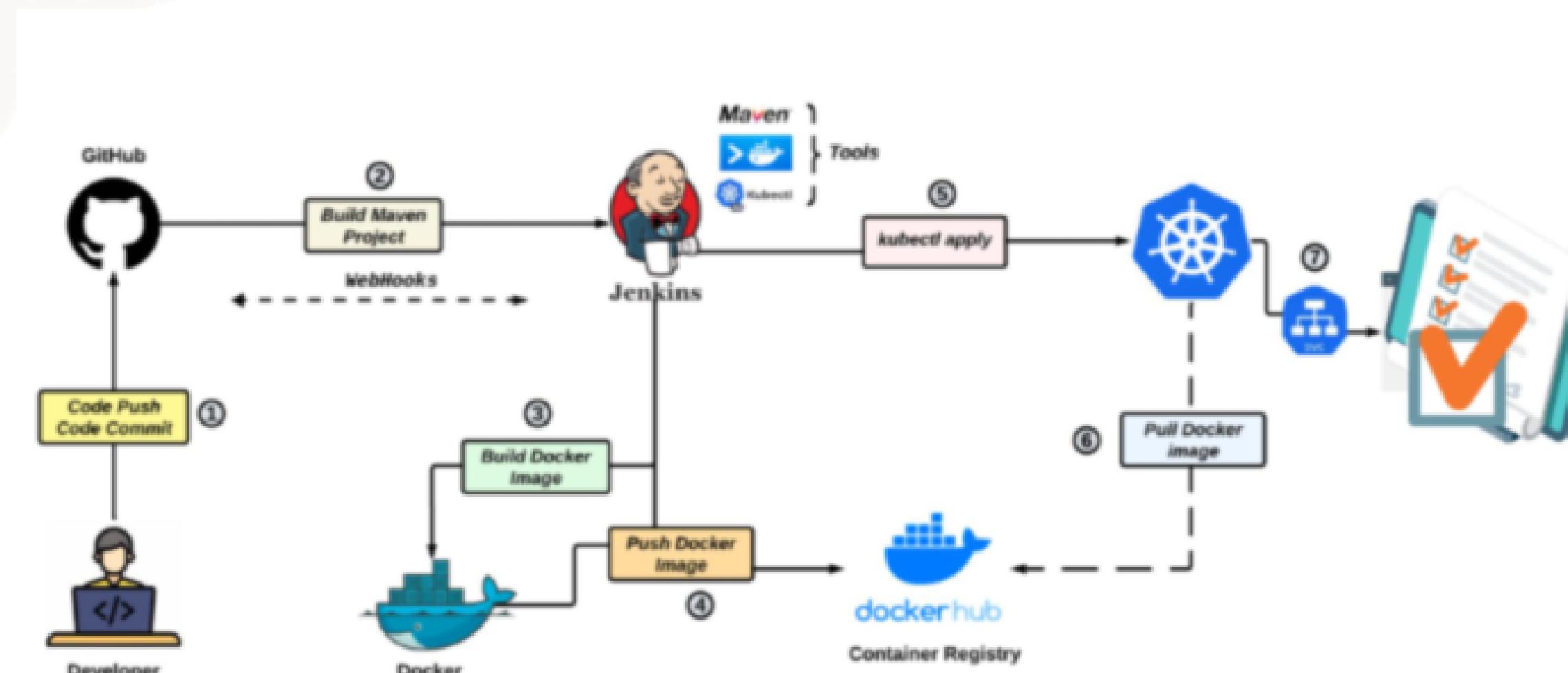
Ejemplo de despliegue donde se crea un ReplicaSet para traer tres nginx Pods:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```



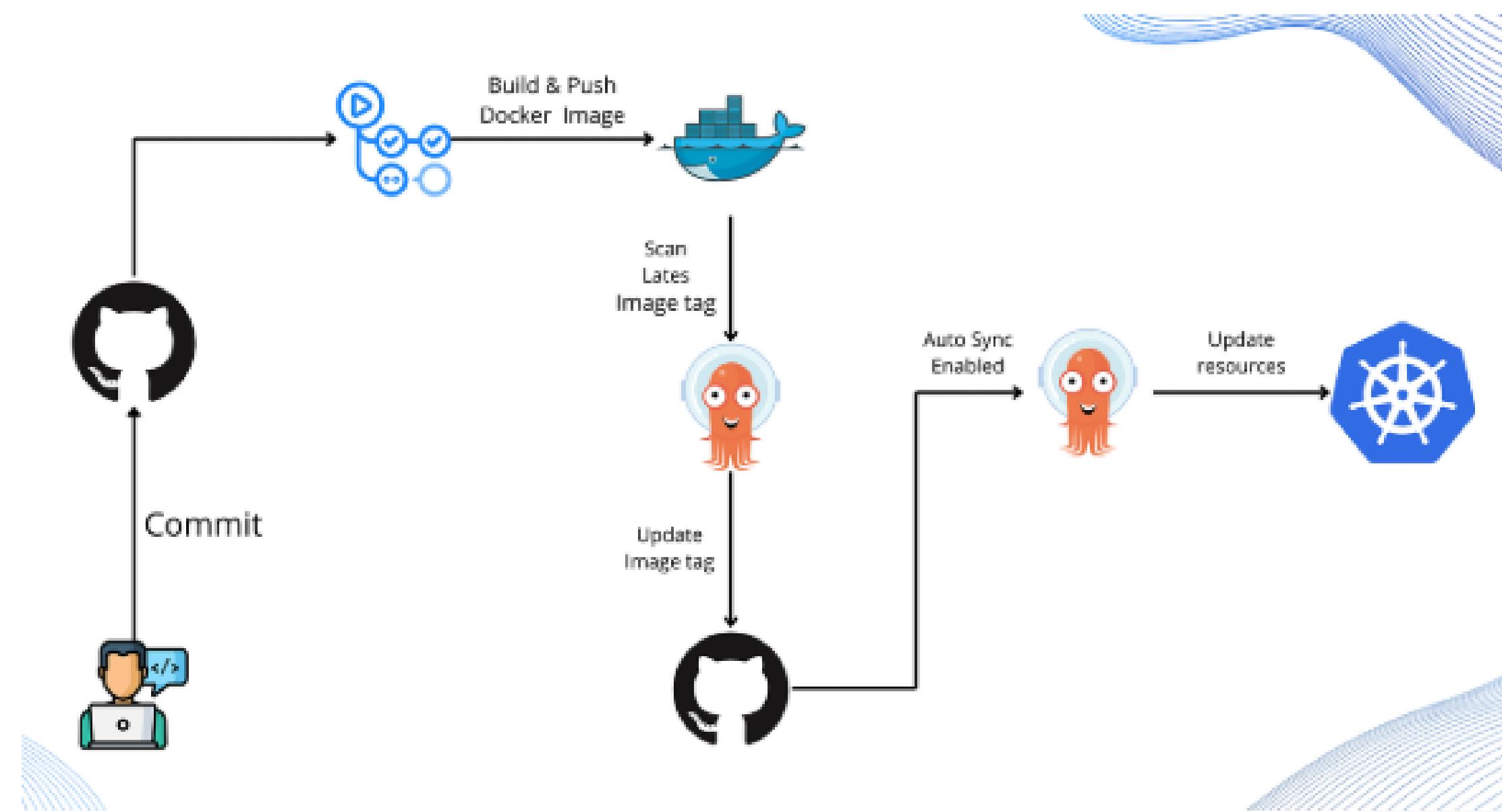
Orquestación con Kubernetes

Ejemplo de Docker y Kubernetes para configurar un pipeline



Orquestación con Kubernetes

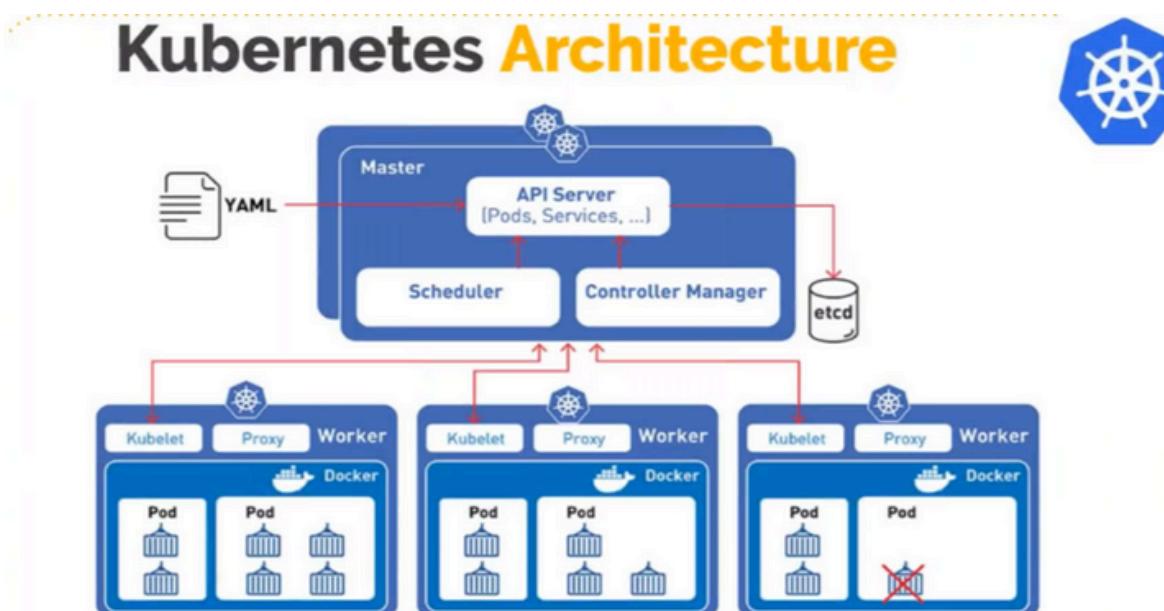
Flujo de Despliege:



Orquestación con Kubernetes

Kubernetes ofrece varias ventajas para escalar aplicaciones:

- Gestión eficiente de recursos: Kubernetes permite que las aplicaciones escalen verticalmente durante las horas punta y horizontalmente cuando la demanda es baja, ahorrando costes.
- Arquitectura desacoplada: Kubernetes admite microservicios, donde los componentes individuales de una aplicación están débilmente acoplados. Cada microservicio puede escalarse de forma independiente, lo que mejora la flexibilidad y evita cuellos de botella.
- Portabilidad entre nubes: Kubernetes es independiente de la plataforma, lo que significa que las aplicaciones pueden ejecutarse en nubes públicas, privadas o híbridas. Esto facilita el escalado global de aplicaciones, sin depender de un único proveedor de nube.*
- Automatización de tareas complejas: Kubernetes automatiza tareas complejas como la implementación, el escalado y la conmutación por error, lo que permite a los equipos centrarse en el desarrollo de nuevas funciones en lugar de preocuparse por la gestión de la infraestructura.



TAREA TEORICA

FLUJO SIMPLE DE DESPLIEGUE

Un flujo de despliegue desde el cambio de código hasta la actualización en Kubernetes, sería de la siguiente manera:

- **Desarrollo:** El desarrollador realiza cambios en el código y los sube al repositorio GitHub
- **CI:** Un sistema como GitHub Actions detecta estos cambios y activa el pipeline
- **Construcción de la imagen:** Se ejecuta 'docker build -t mi-aplicacion:\$COMMIT_SHA .' para crear una nueva imagen con una etiqueta basada en el identificador del commit. Luego la imagen se almacena en un registro de contenedores
- **Pruebas automatizadas:** Se ejecutan pruebas unitarias e integración en contenedores usando la nueva imagen y también se realizan escaneos de vulnerabilidades
- **Actualización en Kubernetes:** Se actualiza el Deployment con la nueva imagen, también Kubernetes inicia un Rolling Update reemplazando gradualmente los pods con la nueva versión y finalmente se verifica el estado del despliegue

VENTAJAS DE KUBERNETES PARA ESCALAR EN EVENTOS DE ALTO TRAFICO

Kubernetes ofrece estas ventajas para escalar en eventos de alto trafico:

- Distribucion de carga: Los Services de Kubernetes distribuyen el trafico entre multiples replicas, mejorando la capacidad de respuesta
- Recuperacion automatica: Si un pod falla, Kubernetes lo reemplaza automaticamente para mantener el numero deseado de replicas
- Despliegue: Los Rolling Updates permiten actualizar la aplicacion sin interrumpir el servicio
- Gestion de recursos: Limita el consumo de CPU y memoria por pod, asi asegurando que ninguna aplicacion monopolice los recursos del cluster

Observabilidad y Troubleshooting



CONCEPTO DE OBSERVABILIDAD

La observabilidad es un componente esencial en el desarrollo, la observabilidad un concepto mas amplio que combina 3 pilares principales: Monitoreo, Logging (registro centralizado de logs) y Trazabilidad (capacidad de seguir el recorrido de una petición a través de distintos microservicios).

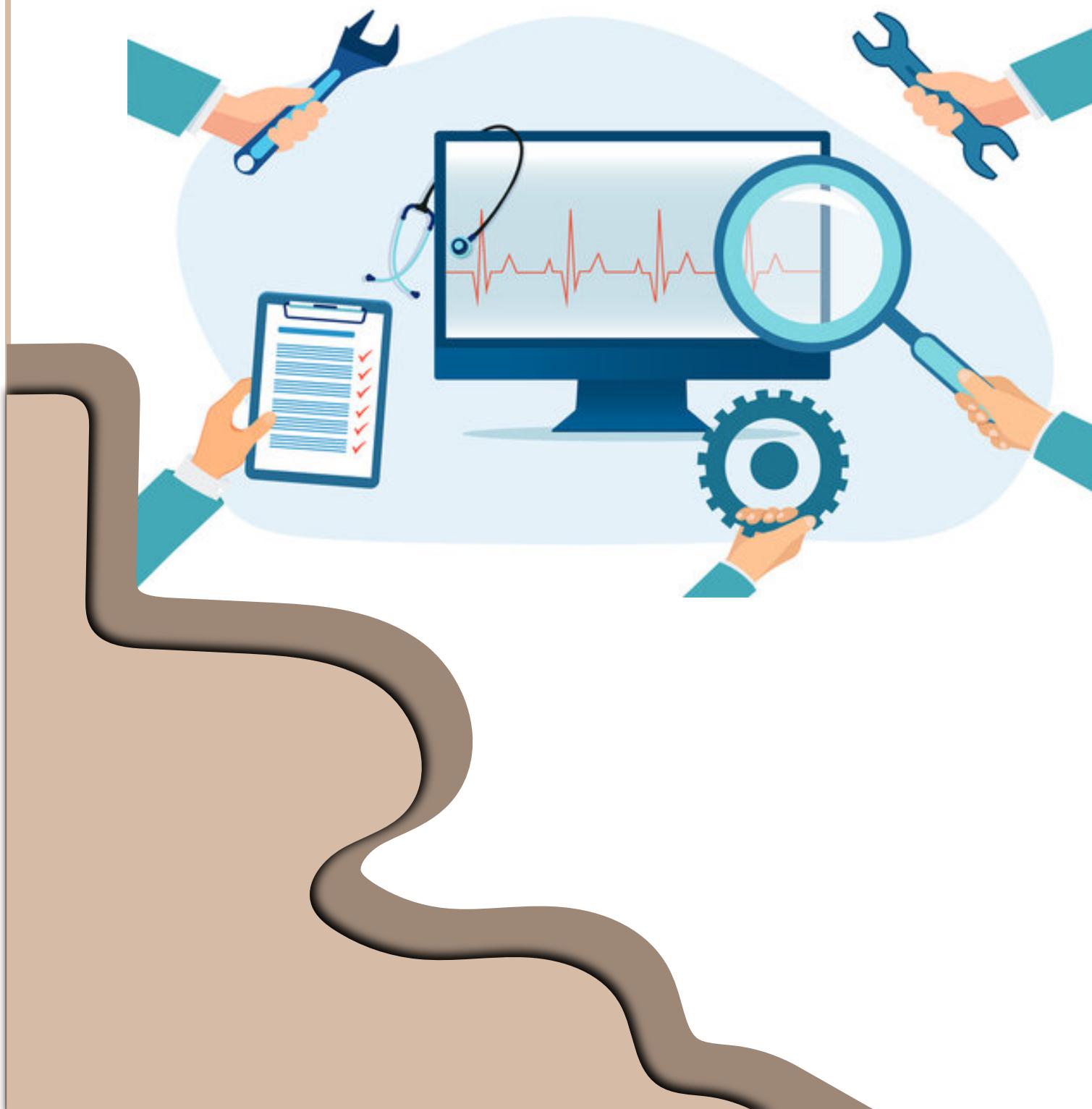
HERRAMIENTAS CLAVE

- Prometheus: Es una herramienta de monitoreo que utiliza un modelo de series temporales y un lenguaje de consultas llamado PromQL para extraer y procesar metricas (CPU, memoria, peticiones HTTP, etc.) de las aplicaciones.
- Grafana: Se encarga de visualizar estas metricas mediante dashboards personalizables lo que permiten a los equipos monitorizar en tiempo real indicadores criticos como la latencia de respuestas, el numero de peticiones y el consumo de recursos.



ESTRATEGIAS DE TROUBLESHOOTING

- Triage de problemas: Se usan dashboards, alertas ante anomalias y trazabilidad
- Diagnostico: Revision de logs, metricas en tiempo real, correlacion de sucesos.
- Resolucion: Rollback a una version estable y cambios en la configuracion de infraestructura.



TAREA TEORICA

INTEGRACION DE PROMETHEUS Y GRAFANA CON KUBERNETES

- **Despliegue:** Prometheus se despliega como StatefulSet en Kubernetes y se configura mediante ConfigMaps. Grafana se despliega como un Deployment
- **Recolección:** Prometheus hace scraping automático de métricas de Kubernetes y aplicaciones utilizando service discovery
- **Visualización:** Grafana consume datos de Prometheus y presenta dashboards personalizables
- **Alertas:** Alertmanager (componente de Prometheus) gestiona notificaciones basadas en reglas definidas

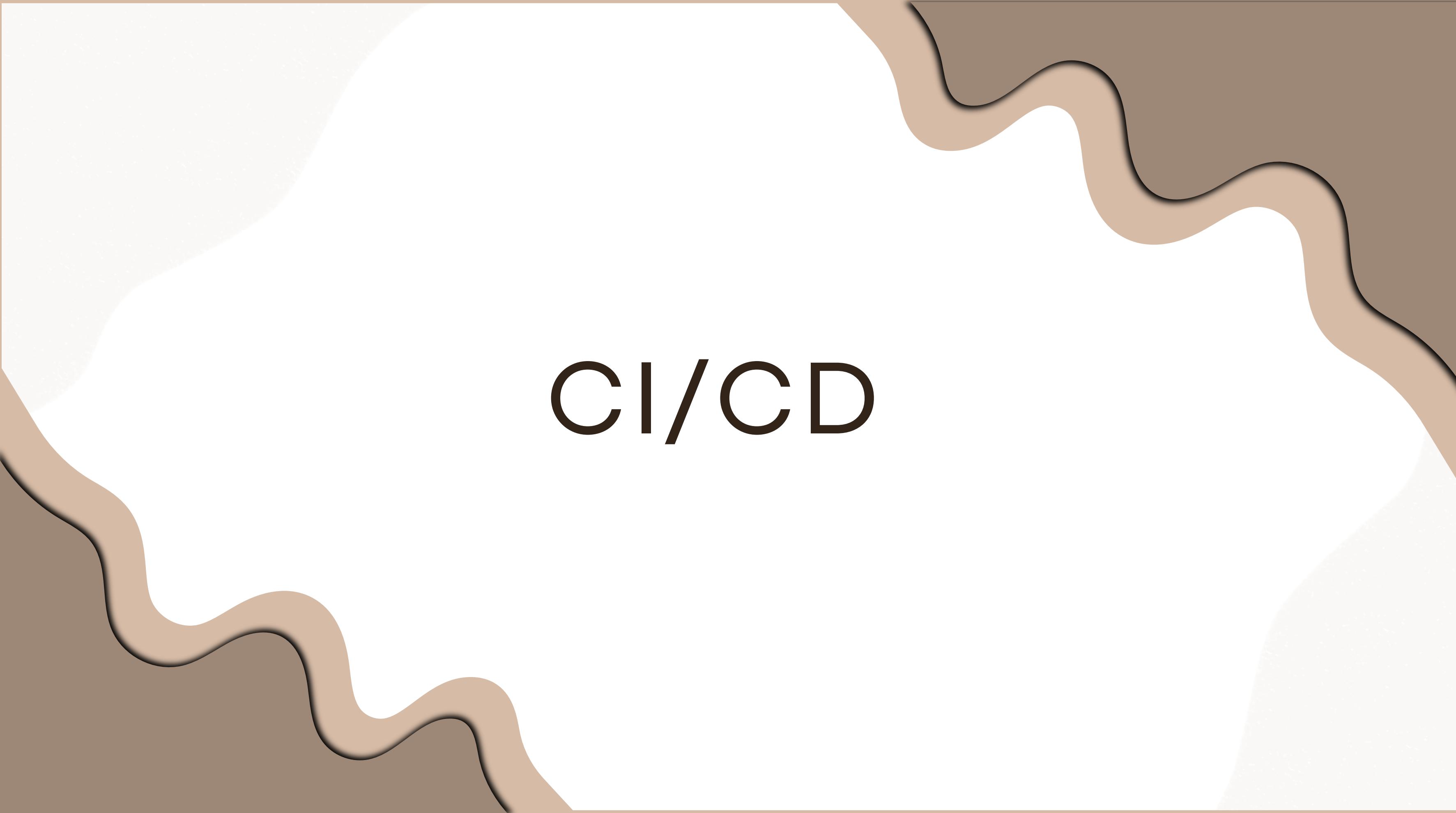
SET DE METRICAS Y ALERTAS PARA UNA APLICACION WEB:

Metricas clave:

- Latencia de peticiones
- Tasa de errores
- Uso de recursos (CPU, memoria, disco)
- Throughput (cantidad de peticiones)
- Tiempo de respuesta de dependencias externas

Alertas esenciales:

- Alta tasa de errores (>5% durante 5 minutos)
- Latencia elevada (valores por encima del umbral)
- Uso excesivo de recursos (>85% memoria, >80% CPU)
- Servicio no disponible

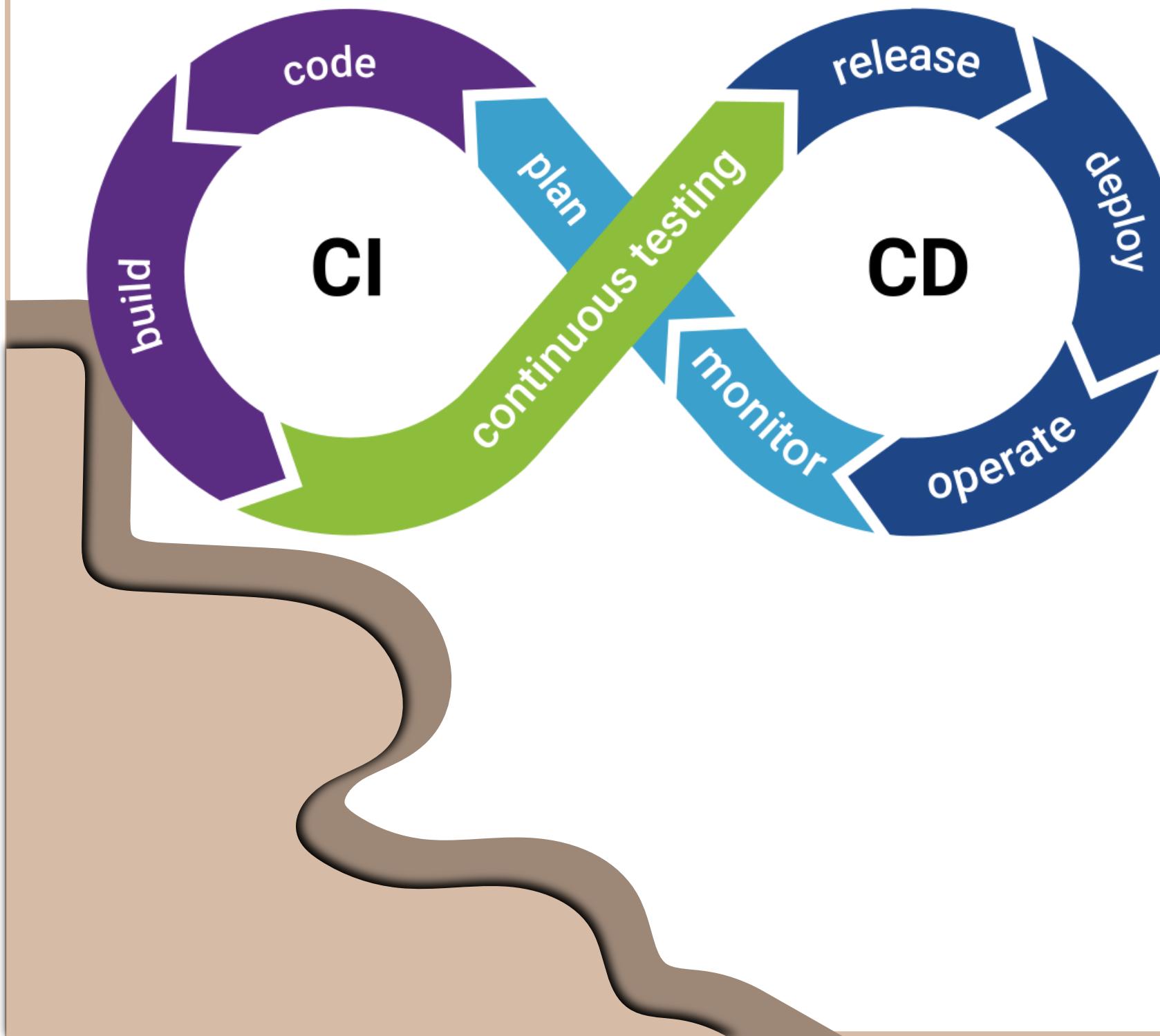


CI/CD

CONCEPTOS FUNDAMENTALES

CI/CD es parte de los principios fundamentales de DevOps

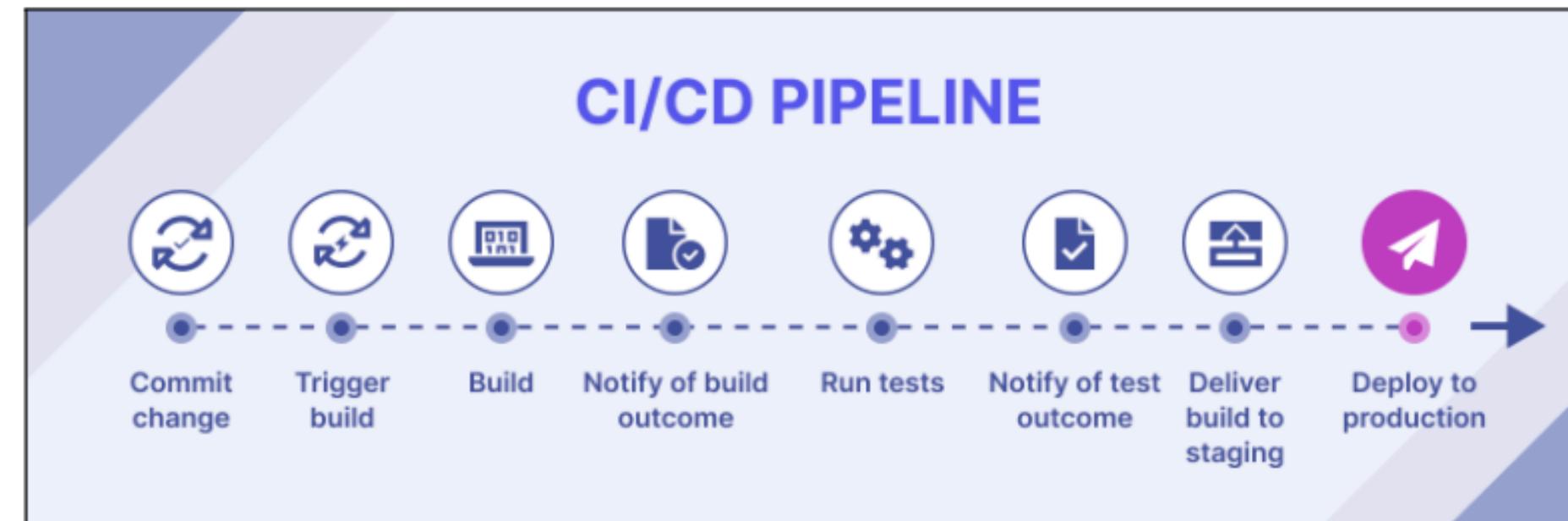
- Integracion continua (CI): Automatizar la construccion y pruebas de la aplicacion con cada commit.
- Despliegue continuo (CD): Entregar automaticamente las nuevas versiones a entornos de prueba o produccion una vez pasen las pruebas.



DISEÑANDO UN PIPELINE

Fases tipicas del diseño de un pipeline :

1. Build: Compilacion del codigo y construccion de artefactos
2. Test: Pruebas unitarias, integracion y aceptacion
3. Seguridad: Analisis estatico y escaneo de vulnerabilidades
4. Deploy: Despliegue a los diferentes entornos



TAREA TEORICA

DIFERENCIA ENTRE ENTREGA CONTINUA Y DESPLIEGUE CONTINUO:

- **Entrega Continua:**
 - Automatiza el proceso hasta la preparacion para produccion.
 - El paso a produccion requiere aprobacion manual
 - Garantiza que cada cambio esta listo para ser desplegado pero permite un control humano sobre cuando realizar este despliegue
- **Despliegue Continuo:**
 - Automatiza todo el proceso incluyendo el despliegue a produccion
 - Cada cambio que pasa todas las pruebas se despliega automaticamente sin intervencion humana

RELEVANCIA DE PRUEBAS AUTOMATICAS DENTRO DEL PIPELINE

- **Pruebas unitarias:** Verifican componentes individuales. Da un Feedback rapido para desarrolladores
- **Pruebas de integracion:** Validan interacciones entre componentes y sistemas externos
- **Pruebas de seguridad:** Identifican vulnerabilidades mediante analisis estatico y escaneo de dependencias
- **Beneficios de la automatizacion de pruebas:** Reduce los errores humanos, obtenemos un ciclo de feedback acelerado, lo que da mayor confianza en los cambios y facilita los despliegues frecuentes minimizando el riesgo



M u c h a s
G R A C I A S