Introducción a DevOps y GitOps

De la Integración Continua al Despliegue Automatizado con Herramientas Modernas

Quien soy?

- Soy **Bruno Damián Zappellini,** Licenciado en Informática. Desde 2011 trabajo en el Superior Tribunal de Justicia del Chubut, y desde 2022 soy Jefe de Redes. Me encargo de automatizar procesos y optimizar la infraestructura, porque sí, me gusta que todo funcione casi solo.

- **Certificaciones en Kubernetes y Linux**: Tengo algunas certificaciones que suenan complicadas (Kubernetes, Linux Foundation Certified Engineer, etc.)

- **Docente desde 2014**: Además de trabajar en infraestructura, también soy profesor en la Universidad Nacional de la Patagonia San Juan Bosco. Doy clases sobre redes y seguridad

- **¿Qué me gusta hacer?**: Además de la tecnología, me apasiona practicar artes marciales y deportes extremos, como kitesurf y snowboard. Cuando no estoy arreglando servidores, probablemente me encuentres en el agua, en la nieve... o con mi familia, que es mi verdadera pasión.

Este curso NO es... ¡pero igual lo vamos a ver!

- **No es un curso de Git,** pero prepárate para clonar, hacer commits y crear ramas como un ninja del control de versiones.

- **No es un curso de Docker,** pero vamos a encapsular nuestros proyectos en contenedores más rápido que empaquetar un .tar.gz en la consola.

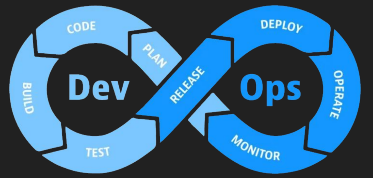
- **No es un curso de Kubernetes,** pero te prometo que al final no solo sabrás pronunciarlo, sino que también desplegarás aplicaciones en él.

- **No es magia...** aunque a veces lo parece. Aquí no hacemos trucos, pero te enseñaremos a automatizar como si lo fueran.

Hoja de Ruta:

- **Introducción a DevOps**
    - Conceptos básicos y principios de DevOps.
    - Beneficios y desafíos de implementar DevOps en organizaciones.
- **Git y GitHub**
    - Introducción a Git y control de versiones.
    - Crear y gestionar repositorios en GitHub.
    - Colaboración y flujo de trabajo en GitHub.
- **Integración Continua con GitHub Actions**
    - Conceptos de CI y CD.
    - Configuración y uso de GitHub Actions para CI.
    - Ejemplos prácticos de pipelines de CI.
- **Introducción a Docker**
    - Conceptos básicos de contenedores.
    - Creación y gestión de contenedores con Docker.
    - Integración de Docker con GitHub Actions para CI.

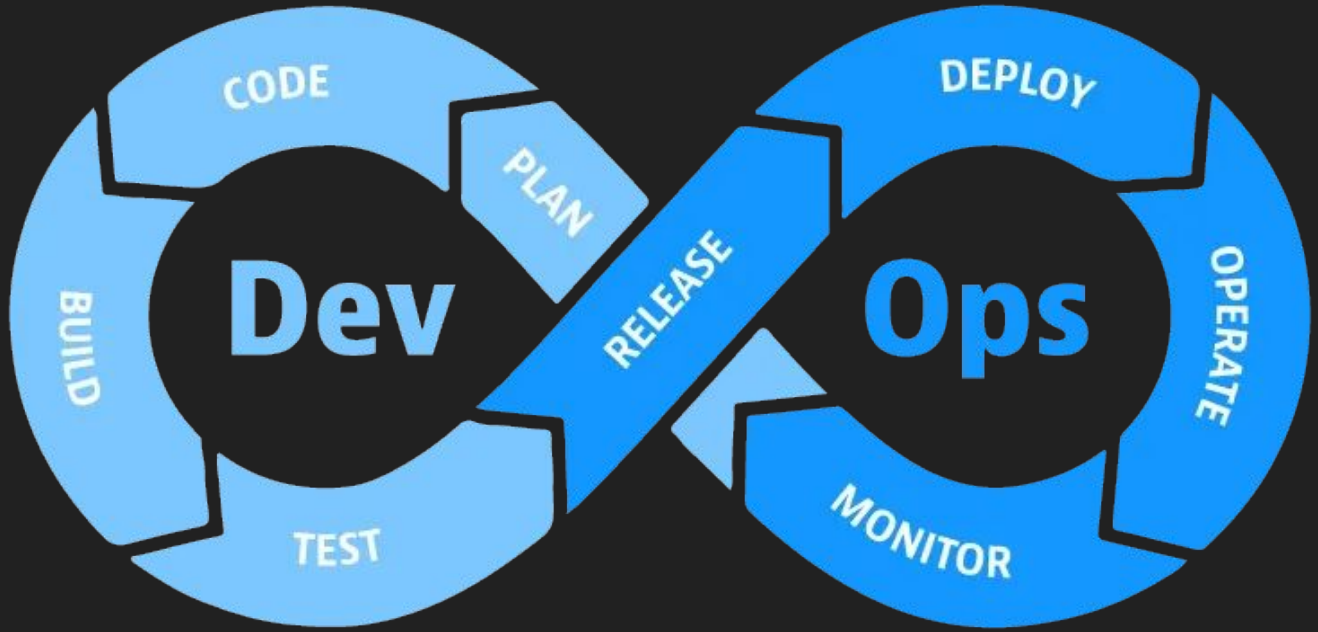
DevOps es un viaje:



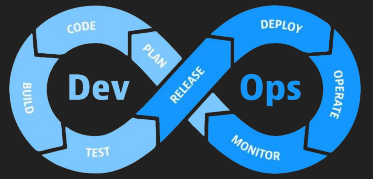
- **Metas a largo plazo:** DevOps no es magia; requiere tiempo y recursos.
- **Paneles de control y KPIs**: Monitorea los procesos y mejoras con paneles automatizados.
- **Seguridad:** Prioriza la seguridad en cada fase del ciclo de vida del software.

¿Por qué DevOps?

DevOps: Desarrollo + Operaciones



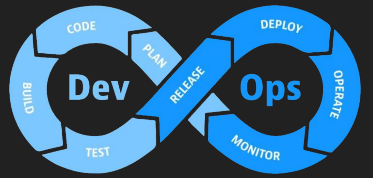
DevOps: Desarrollo + Operaciones



DevOps es la combinación de personas, procesos y tecnología para ofrecer valor a los clientes de forma constante.

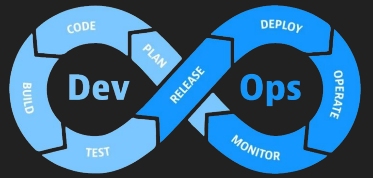
- Une roles tradicionalmente aislados como desarrollo, operaciones de TI, ingeniería de calidad y seguridad.
- Fomenta la colaboración entre equipos para mejorar la calidad y confiabilidad de los productos.

DevOps para los equipos:



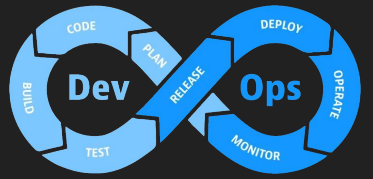
- Los equipos trabajan de forma más colaborativa y coordinada.
- Se mejoran los tiempos de respuesta a las necesidades de los clientes.
- Se aumenta la confianza en las aplicaciones creadas.
- Se logra cumplir con los objetivos empresariales más rápidamente.

Beneficios de DevOps:



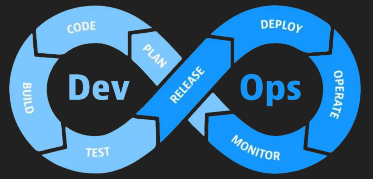
- **Reducción del tiempo de comercialización:** Lanza productos más rápido.
- **Adaptación al mercado:** Mantente competitivo y responde mejor al cambio.
- **Estabilidad y fiabilidad:** Mejora la estabilidad del sistema.
- **Tiempo medio de recuperación:** Reduce el tiempo de respuesta ante fallos.

Claves para una implementación exitosa:



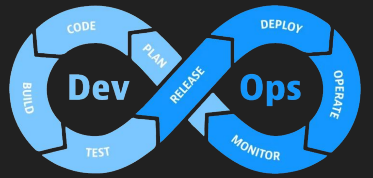
1. **Colaboración**: Trabajo sincronizado entre desarrollo, operaciones y seguridad.
2. **Paciencia y dedicación:** El cambio cultural y práctico lleva tiempo.
3. **Métricas de rendimiento:** Define objetivos claros desde el principio.
4. **Herramientas adecuadas:** Prioriza los procesos, no solo las herramientas.

Claves para una implementación exitosa:



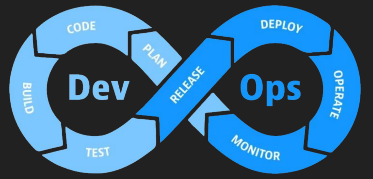
1. **Colaboración**: Trabajo sincronizado entre desarrollo, operaciones y seguridad.
2. **Paciencia y dedicación:** El cambio cultural y práctico lleva tiempo.
3. **Métricas de rendimiento:** Define objetivos claros desde el principio.
4. **Herramientas adecuadas:** Prioriza los procesos, no solo las herramientas.

En resumen...



- **DevOps** es más que una metodología, es una cultura.
- **La clave** está en la colaboración, automatización y mejora continua.
- Aumenta la agilidad y la capacidad de respuesta, y disminuye los tiempos de recuperación.

Recursos recomendados



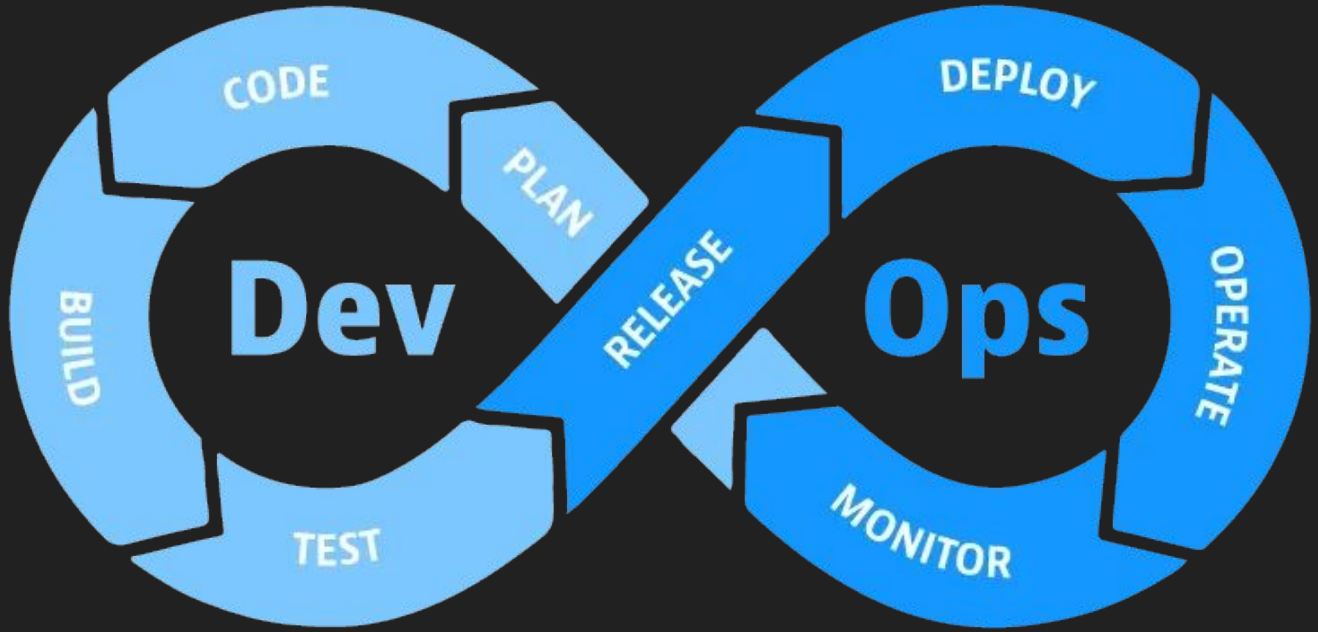
- Linux Foundation: https://www.linuxfoundation.org
Organización líder en el desarrollo y promoción de tecnologías de código abierto, incluyendo Linux y Kubernetes.

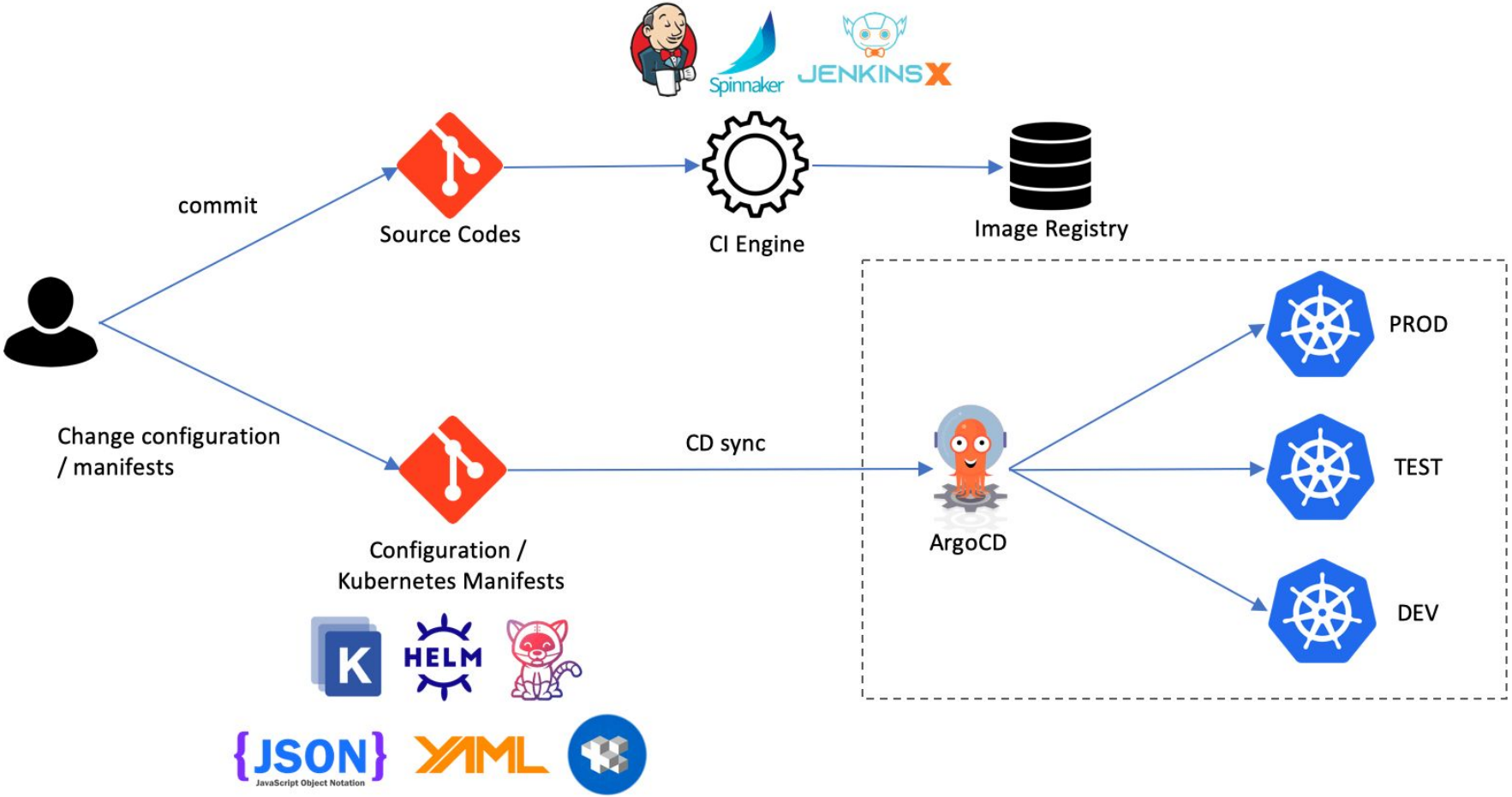
- Cloud Native Computing Foundation (CNCF): https://www.cncf.io
Comunidad dedicada a la adopción de tecnologías nativas en la nube, como Kubernetes, Prometheus, entre otras.

- CNCF Cloud Native Landscape https://landscape.cncf.io
Mapa interactivo de todas las herramientas y tecnologías relacionadas con el ecosistema nativo de la nube.

- DevOps Roadmap https://roadmap.sh/devops
Guía visual y detallada sobre las habilidades y herramientas necesarias para convertirse en un experto en DevOps.

DevOps: Desarrollo + Operaciones




Spinnaker
JENKINSX
commit
Source Codes
CI Engine
Image Registry
Change configuration
/ manifests
CD sync
ArgoCD
PROD
TEST
DEV
Configuration /
Kubernetes Manifests
HELM
{JSON}
JavaScript Object Notation
YAML


git

Antes de Git:

¿Te suena familiar? 😂

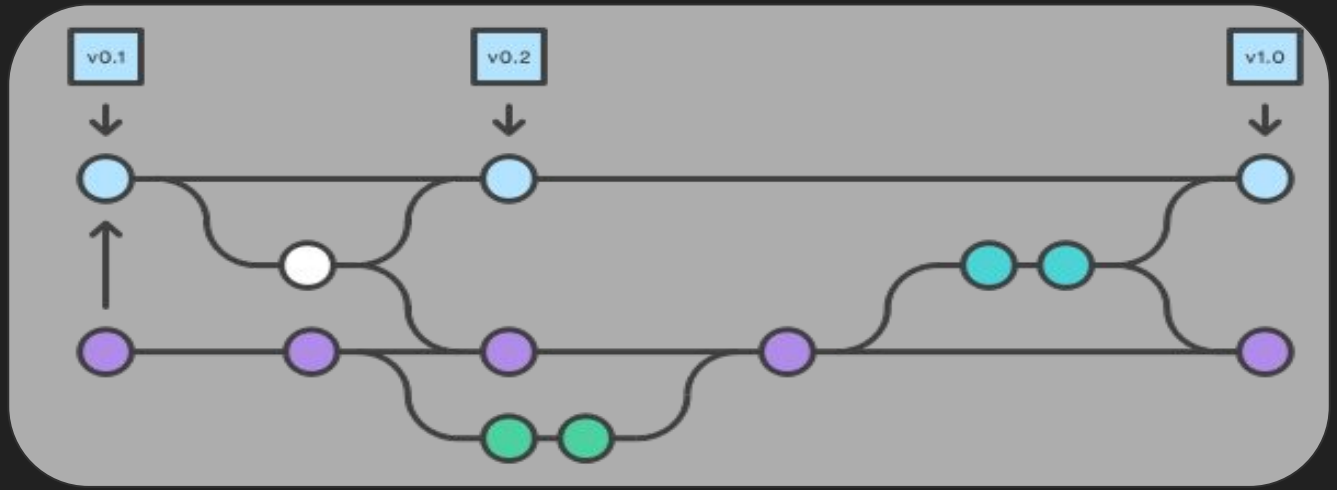
```
#> ls
   proyecto_v1_FINAL_FINAL.zip
   proyecto_v1_DE_VERDAD_FINAL.zip
   proyecto_v2_REAL_FINAL_final_final.zip
```

Git: Control de versiones distribuido

- Git es un sistema de control de versiones que permite:
- Guardar el historial de cambios en el código.
- Trabajar de forma colaborativa sin pisarse los pies.
- Recuperar versiones anteriores si algo sale mal.

Conceptos clave

- **Repositorio (repo)**: Donde vive tu proyecto.
- **Commit**: Un "punto de control" en el historial.
- **Branch (rama)**: Una línea de trabajo independiente.
- **Merge**: Combinar cambios de diferentes ramas.



Comandos esenciales:

```
# Inicializar un repositorio Git en un directorio
git init

# Verificar el estado actual del repositorio
git status

# Añadir archivos al área de preparación
git add <archivo>

# Guardar los cambios en el repositorio (commit)
git commit -m "Mensaje del commit"

# Ver el historial de commits
git log
```

Git Book

https://git-scm.com

Git Book

https://git-scm.com/book/es/v2

Comandos esenciales:


GitHub

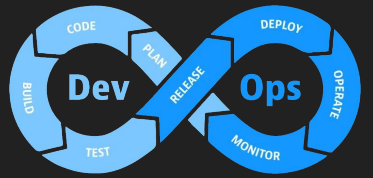
GitHub: Plataforma para colaborar

- GitHub es un servicio basado en la nube que permite:
  - Alojar repositorios Git.
  - Colaborar con otros desarrolladores.
  - Automatizar procesos con herramientas como GitHub Actions.

Crear un repositorio en GitHub:

- Ve a GitHub.com y crea una cuenta.
- Haz clic en "New repository".
- Asigna un nombre y configura la visibilidad (público o privado).
- Haz clic en "Create repository".

DevOps es un viaje:



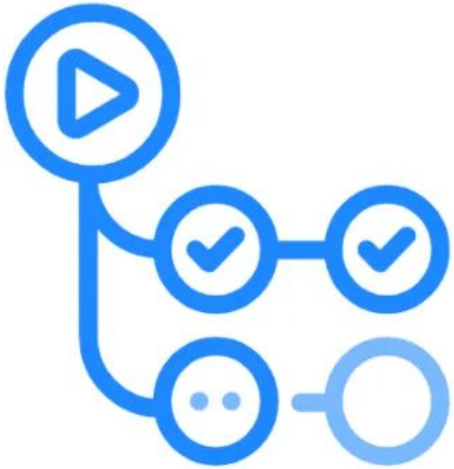
- **Metas a largo plazo:** DevOps no es magia; requiere tiempo y recursos.
- **Paneles de control y KPIs**: Monitorea los procesos y mejoras con paneles automatizados.
- **Seguridad:** Prioriza la seguridad en cada fase del ciclo de vida del software.

¿Qué es CI/CD?

- **CI (Integración Continua):** Es la práctica de integrar los cambios de código de manera frecuente (varias veces al día). Cada integración se verifica automáticamente mediante pruebas para detectar problemas lo antes posible.
    - Objetivo: Asegurar que el código integrado funcione y no cause conflictos o errores inesperados.
- **CD (Despliegue Continuo):** Consiste en la automatización del despliegue de las aplicaciones en entornos de prueba o producción de manera frecuente y confiable.
    - Objetivo: Facilitar entregas rápidas y seguras a los usuarios.

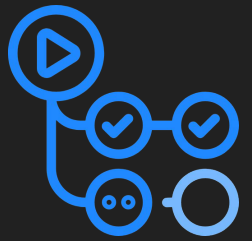
Ventajas de implementar CI/CD:

- **Detección temprana de errores:** Gracias a la ejecución automática de pruebas.
- **Mejora la calidad del software:** Los cambios se integran y despliegan de forma continua, manteniendo un código más estable.
- **Entrega más rápida:** Automatización del ciclo de vida de entrega, desde el desarrollo hasta la producción.
- **Reducción del riesgo:** Al realizar despliegues incrementales en lugar de grandes actualizaciones, se reduce el riesgo de fallos.



GitHub Actions

¿Qué es GitHub Actions?



- **GitHub Actions** es una plataforma de automatización nativa de GitHub.

- Permite crear workflows (flujos de trabajo) para automatizar tareas como:
  - Integración Continua (CI)
  - Despliegue Continuo (CD)
  - Testing automático
  - Otras tareas personalizadas

Subir imagen Docker a Docker Hub usando GitHub Actions

- Pre-requisitos:
  - Asegúrate de tener un Dockerfile en tu proyecto FastAPI.
  - Tienes que crear un token de acceso en Docker Hub:
    - Ve a Docker Hub, accede a tu cuenta y crea un Personal Access Token para que GitHub pueda autenticarse y subir imágenes.

- Configura las credenciales en GitHub:
  - Ve a tu repositorio en GitHub.
  - En "Settings" -> "Secrets and variables" -> "Actions", agrega dos secrets:
    - DOCKER_USERNAME: Tu usuario de Docker Hub.
    - DOCKER_PASSWORD: El token de acceso que generaste en Docker Hub.

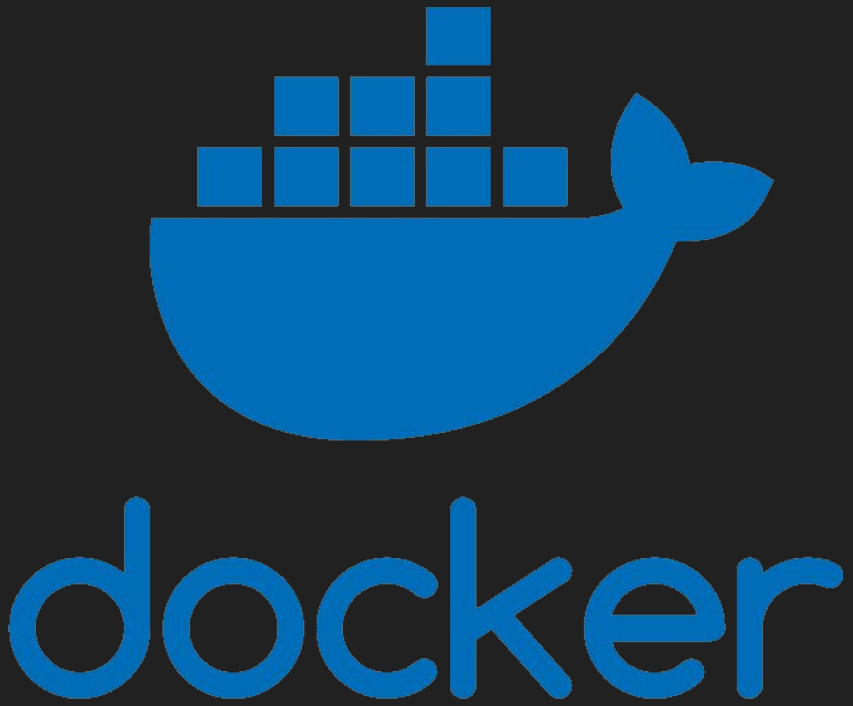
github action

Links:

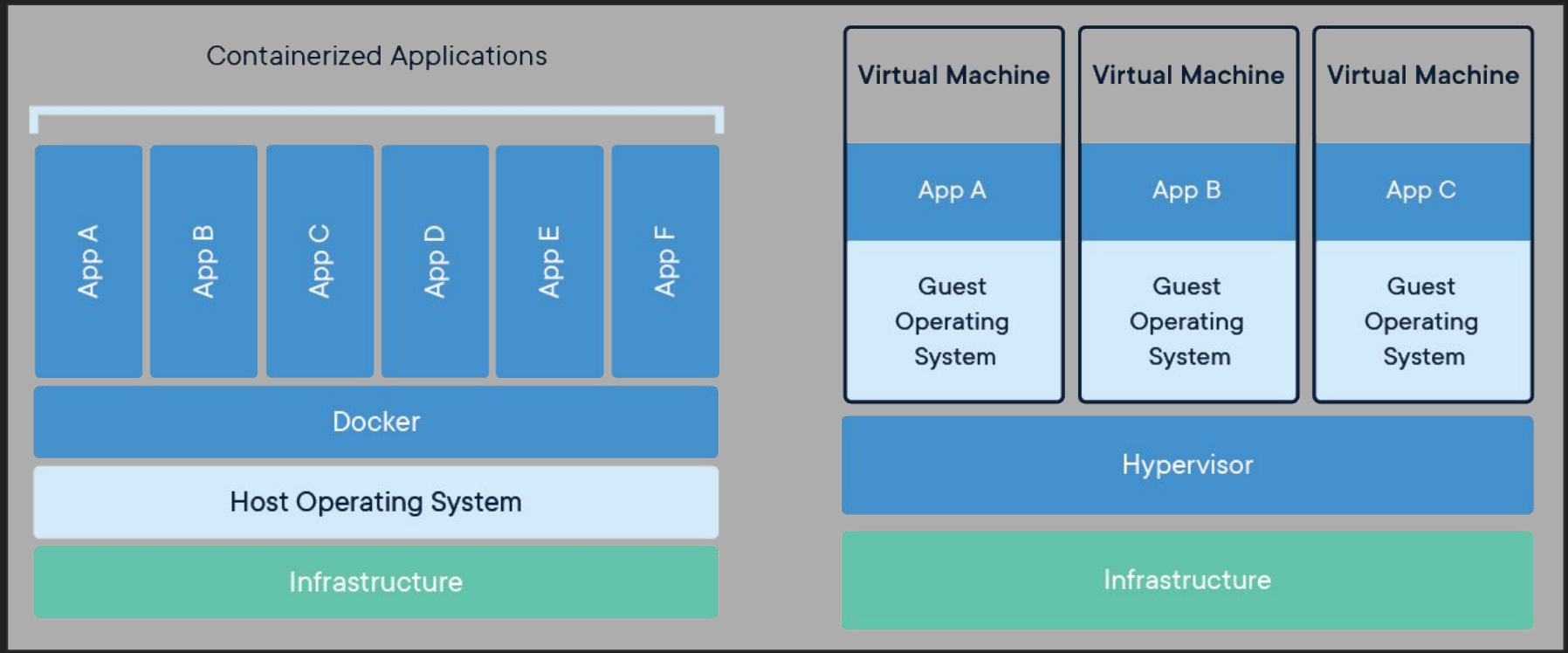
https://github.com/features/actions

https://docs.github.com/es/actions

https://github.com/actions/runner


docker

Docker Vs Virtualización



¿Qué es Docker?

- **Docker** es una plataforma que permite empaquetar aplicaciones y sus dependencias en contenedores.

- Estos contenedores aseguran que la aplicación funcione igual en cualquier entorno, desde desarrollo hasta producción.

Crear una imagen Docker desde un Dockerfile:

1. Escribe un Dockerfile que describa cómo construir la imagen.

2. Utiliza el comando `docker build` para construir la imagen:

```bash
# Construir una imagen desde un Dockerfile
docker build -t nombre_de_imagen .
```

- `-t nombre_de_imagen`: Etiqueta (nombre) para la imagen.
- `.` : Indica el directorio actual, donde debe estar el Dockerfile.

Ejecutar una imagen en un contenedor:


```bash
# Ejecutar un contenedor a partir de una imagen
docker run -d --name nombre_contenedor -p 8000:80 nombre_de_imagen
```

- `-d`: Ejecuta el contenedor en segundo plano (detached).
- `--name nombre_contenedor`: Asigna un nombre al contenedor.
- `-p 8000:80`: Mapea el puerto 8000 del host al puerto 80 del contenedor.

Subir una imagen a Docker Hub (u otro registry):

1. Iniciar sesión en Docker Hub:

```bash
docker login
```

2. Etiquetar la imagen para Docker Hub:

```bash
docker tag nombre_de_imagen usuario_dockerhub/nombre_de_imagen:latest
```

3. Subir la imagen:

```bash
docker push usuario_dockerhub/nombre_de_imagen:latest
```

Descargar (pull) una imagen desde Docker Hub:

bash

```bash
# Descargar una imagen desde Docker Hub
docker pull usuario_dockerhub/nombre_de_imagen:latest
```

Ver contenedores en ejecución y detenidos:

bash

```bash
# Mostrar contenedores en ejecución
docker ps

# Mostrar todos los contenedores (incluyendo los detenidos)
docker ps -a
```

Ver los logs de un contenedor específico:


```bash
# Ver los logs de un contenedor
docker logs nombre_contenedor
```

- Útil para depurar problemas o monitorear el estado de una aplicación dentro de un contenedor.

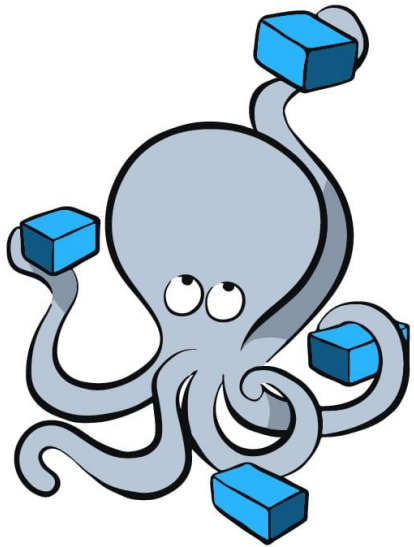
Ver detalles y descripción de un contenedor:


```bash
# Obtener información detallada sobre un contenedor
docker inspect nombre_contenedor
```

- Muestra información detallada como la configuración del contenedor, volúmenes, puertos, etc.

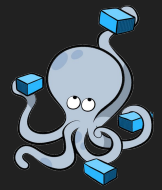
Resumen de los comandos Docker más usados:

- `docker build`: Construir una imagen desde un Dockerfile.
- `docker run`: Ejecutar un contenedor.
- `docker ps`: Ver el estado de los contenedores.
- `docker logs`: Ver los logs de un contenedor.
- `docker push / pull`: Subir y descargar imágenes a/desde un registro.
- `docker inspect`: Ver detalles completos de un contenedor.



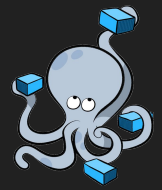
docker Compose

¿Qué es Docker Compose?



- **Docker Compose** es una herramienta que permite definir y ejecutar aplicaciones multicontenedor.
- Utiliza un archivo YAML (`docker-compose.yml`) para configurar todos los servicios (contenedores) que una aplicación necesita.
- Facilita la creación de entornos reproducibles, desde desarrollo hasta producción.

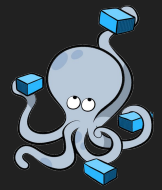
Archivo básico docker-compose.yml



```yaml
version: '3'
services:
  web:
    image: nginx
    ports:
      - "8080:80"

  app:
    build: .
    volumes:
      - .:/code
    ports:
      - "5000:5000"
```

- `version`: Define la versión de Docker Compose.
- `services`: Define los servicios (contenedores).
  - web: Contenedor basado en la imagen nginx.
  - app: Contenedor que se construye desde el Dockerfile en el directorio actual.

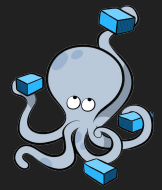
Comandos esenciales de Docker Compose:



bash

```
# Iniciar todos los servicios definidos en el archivo docker-compose.yml
docker-compose up

# Iniciar en segundo plano (background)
docker-compose up -d

# Detener y eliminar los contenedores
docker-compose down

# Ver los logs de todos los contenedores
docker-compose logs

# Ver el estado de los contenedores
docker-compose ps
```

Servicios típicos en Docker Compose:

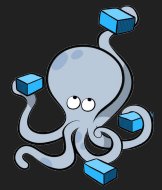


```yaml
services:
  db:
    image: postgres
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
  app:
    build: .
    ports:
      - "5000:5000"
    environment:
      DATABASE_URL: postgres://user:password@db:5432/mydb
    depends_on:
      - db
```

- **db:** Define un contenedor para la base de datos y conéctalo con la aplicación.

- **app:** La aplicación que depende del contenedor db (base de datos Postgres).
- **depends_on**: Asegura que la base de datos esté lista antes de levantar la aplicación.

Resumen de Docker Compose:

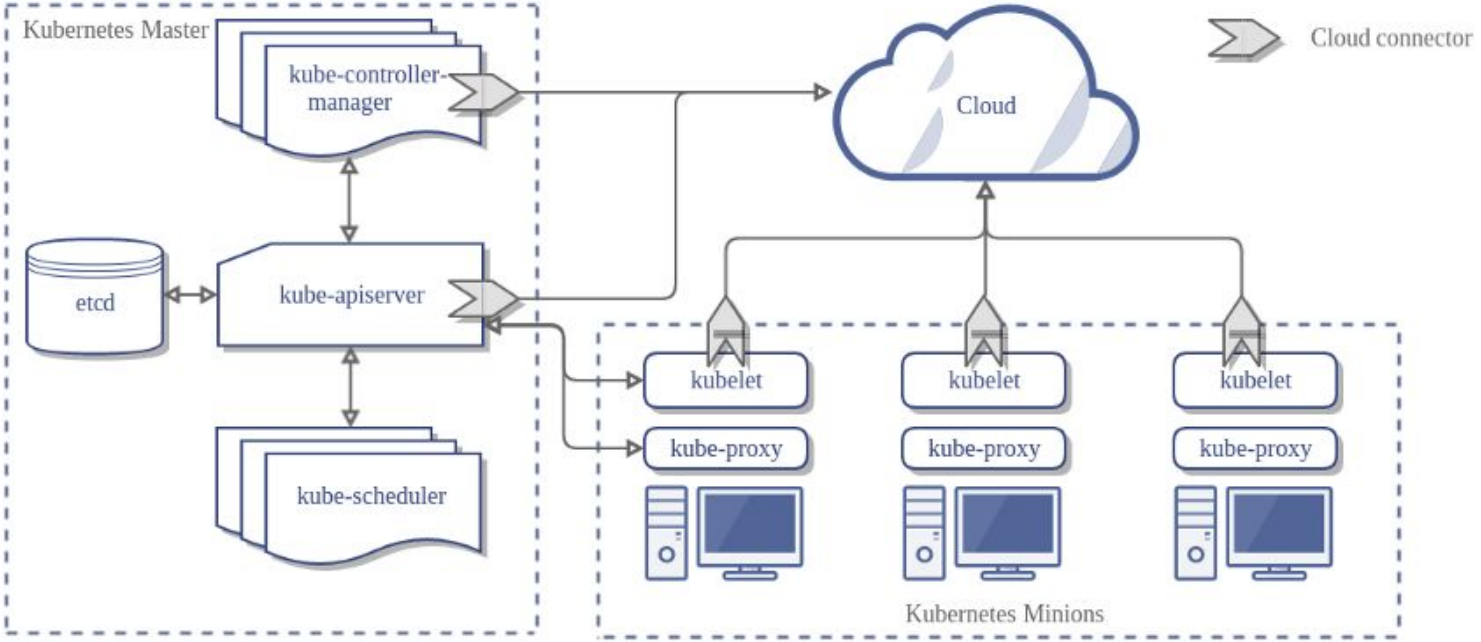


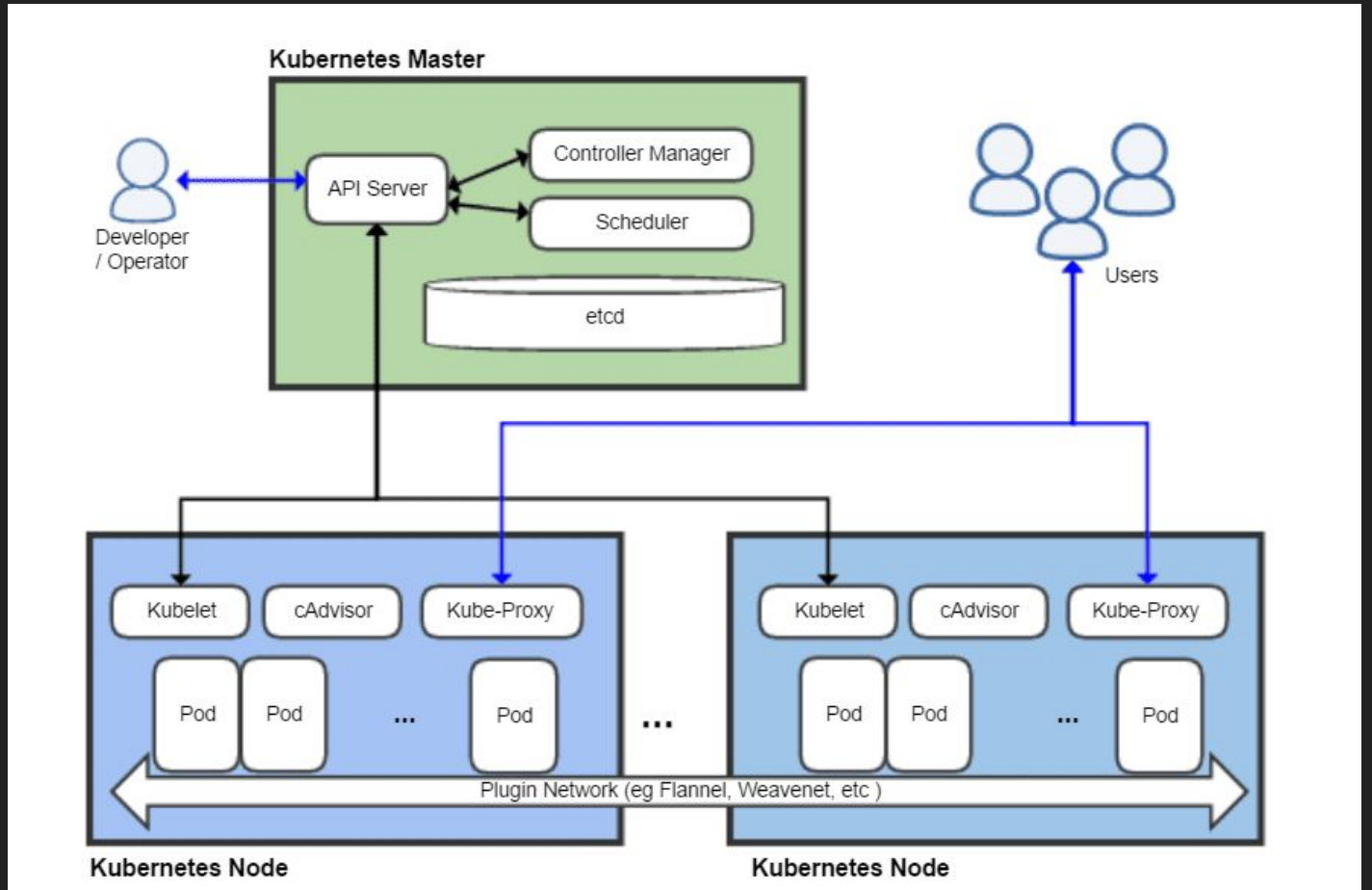
- Facilita la gestión de múltiples contenedores con un solo archivo YAML.
- Proporciona un entorno consistente para desarrollo y pruebas.
- Permite escalar servicios fácilmente y ejecutar aplicaciones complejas de forma sencilla.


kubernetes

¿Qué es Kubernetes?

- **Kubernetes** (K8s) es una plataforma de código abierto para la **orquestación de contenedores.**
- Facilita el despliegue, escalado y gestión de aplicaciones en contenedores en **entornos distribuidos**.
- Kubernetes automatiza la mayor parte de las tareas operativas requeridas para mantener aplicaciones resilientes.


Kubernetes Master
kube-controller-manager
etcd
kube-apiserver
kube-scheduler
Cloud
Cloud connector
kubelet
kube-proxy
kubelet
kube-proxy
kubelet
kube-proxy
Kubernetes Minions


Kubernetes Master
Developer
/ Operator
API Server
Controller Manager
Scheduler
etcd
Users
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
...
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
Plugin Network (eg Flannel, Weavenet, etc)
Kubernetes Node
Kubernetes Node

Documentación:

https://kubernetes.io/docs/

Arquitectura de Kubernetes

- **Master Node**: Controla el clúster, gestionando el scheduler, controller manager y API server.

- **Worker Nodes**: Ejecutan los pods.

- **ETCD**: Almacena el estado del clúster.

Minikube

Minikube permite ejecutar Kubernetes localmente, ideal para desarrollo y pruebas.

1. Instalar Minikube (si no lo tienes instalado): https://minikube.sigs.k8s.io/docs/start/?arch=%2Flinux%2Fx86-64%2Fstable%2Fbinary+download
2. Iniciar Minikube usando Docker como el driver:

```
minikube start --driver=docker
```

- Esto configura Minikube para que use Docker como backend, creando un clúster Kubernetes en contenedores.

Agregar y eliminar nodos workers en Minikube:

- Agregar un nodo worker:

  `minikube node add`

- Eliminar un nodo worker:

  `minikube node delete nombre-del-nodo`

- Minikube permite emular un entorno multinodo para hacer pruebas locales de escalabilidad.

Componentes principales de Kubernetes:

1. Pod: Unidad básica de Kubernetes, que contiene uno o más contenedores.
2. Node: Máquina física o virtual que ejecuta los pods. Puede ser un worker o master node.
3. Cluster: Conjunto de nodos gestionados por un master node.
4. Service: Abstracción que define una política de acceso a los pods (interno o externo).
5. Deployment: Define la gestión de actualizaciones y el estado deseado de los pods.

Archivo básico YAML para un pod:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  containers:
    - name: my-container
      image: nginx
      ports:
        - containerPort: 80
```

apiVersion: Versión de la API de Kubernetes.

kind: Tipo de recurso (Pod en este caso).

metadata: Información del pod, como nombre y etiquetas.

spec: Especifica los contenedores y su configuración.

Comandos esenciales de Kubernetes (kubectl):

```bash
# Ver todos los pods en el clúster
kubectl get pods

# Describir un pod específico
kubectl describe pod nombre_pod

# Aplicar un archivo de configuración YAML
kubectl apply -f archivo.yaml

# Escalar un deployment a más réplicas
kubectl scale --replicas=3 deployment/nombre_deployment

# Eliminar un pod
kubectl delete pod nombre_pod
```

Crear un Deployment:

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

- **replicas**: Número de instancias (pods) que Kubernetes debe ejecutar.
- **template**: Configura cómo debe ser cada réplica (contenedor, puertos, etc.).

Servicios en Kubernetes:

Un **Service** expone los pods a través de una IP estable, permitiendo la comunicación entre aplicaciones o con el mundo exterior.

- **type**: NodePort: Expone el servicio en un puerto de los nodos para acceso externo.
- **targetPort**: El puerto al que Kubernetes envía el tráfico dentro del contenedor.

yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30007
```

Escalado automático (Horizontal Pod Autoscaler):

```
# Comando para habilitar el escalado automático
kubectl autoscale deployment nombre_deployment --min=2 --max=10 --cpu-percent=80
```

- **Horizontal Pod Autoscaler** ajusta automáticamente la cantidad de pods en función de la carga del CPU u otras métricas.

Resumen de Kubernetes:

- Kubernetes simplifica la gestión de aplicaciones en contenedores en grandes entornos.
- Los principales componentes son **pods, deployments y services.**
- Utiliza **kubectl** para gestionar, escalar y monitorizar los recursos del clúster.
- Automatiza tareas como el escalado y el despliegue para mejorar la resiliencia.

ArgoCD

¿Qué es ArgoCD?

- **ArgoCD** es una herramienta de entrega continua (CD) específica para Kubernetes, basada en **GitOps**.

- Se encarga de gestionar el despliegue de aplicaciones en Kubernetes utilizando el código fuente almacenado en Git como fuente de verdad.

Conceptos clave:

- **GitOps**: Proceso en el que la infraestructura y las aplicaciones son declaradas en archivos de configuración en Git, y el clúster las aplica automáticamente.
- **Aplicación**: ArgoCD gestiona aplicaciones declaradas como archivos YAML en un repositorio Git.
- **Sync**: ArgoCD sincroniza el estado del clúster con el estado definido en Git.
- **Health**: Estado de la aplicación (Healthy, Degraded, etc.).

Flujo de trabajo de ArgoCD:

1. **Repositorio Git**: El código de la aplicación y las configuraciones de Kubernetes (YAML, Helm, Kustomize) están en Git.
2. **ArgoCD**: Monitoriza continuamente el repositorio de Git.
3. **Sincronización automática**: ArgoCD detecta cambios en el repositorio y los aplica al clúster de Kubernetes.
4. **Despliegue**: La aplicación se actualiza y el estado del clúster se sincroniza con el estado en Git.

Instalar ArgoCD en un clúster Kubernetes:

```
#> kubectl create namespace argocd kubectl apply -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

- Crear `namespace`: `kubectl create namespace argocd`
- Instalar **ArgoCD**: Se aplica el manifiesto desde el repositorio oficial.

Acceder a la interfaz gráfica de ArgoCD:

```
kubectl get svc -n argocd argocd-server
```

- Esto te dará la URL y el puerto para acceder a la interfaz de ArgoCD.
- El **usuario por defecto** es `admin` y puedes obtener la contraseña ejecutando:

```
kubectl get secret argocd-initial-admin-secret -n argocd -o
jsonpath="{.data.password}" | base64 --decode
```

Crear una aplicación desde la interfaz de ArgoCD:

1. Ve a la interfaz de ArgoCD.
2. Haz clic en "**New App**".
3. Configura los siguientes campos:
    - **Application Name**: Nombre de la aplicación.
    - **Project**: Selecciona el proyecto (por defecto).
    - **Repository URL**: La URL del repositorio Git donde están los manifiestos de Kubernetes.
    - **Path**: El directorio dentro del repo que contiene los manifiestos.
    - **Cluster**: Selecciona el clúster donde desplegar la app.
    - **Namespace**: El namespace donde se desplegará la aplicación.

Sincronización y monitoreo:

- **Sync**: Sincroniza el estado de la aplicación en el clúster con el estado en Git.
- **Auto-Sync**: Habilita la sincronización automática para que los cambios en Git se apliquen automáticamente.
- **Health Status**: Monitoriza el estado de la aplicación (Healthy, Degraded, Missing, etc.).

Resumen de ArgoCD:

- **ArgoCD** facilita la entrega continua utilizando el enfoque GitOps.
- Mantiene el estado de las aplicaciones en Kubernetes sincronizado con los archivos declarativos en Git.
- Ofrece una interfaz gráfica para gestionar y monitorizar aplicaciones.
- La sincronización automática asegura que los despliegues estén siempre actualizados.

Fin