

Kubernetes: Orquestación de contenedores

Qué pasa cuando tenés muchos contenedores y necesitás que alguien los coordine — sin que nada se caiga a las 3am

UN MOMENTO: ¿POR QUÉ TENDRÍA VARIOS CONTENEDORES?

En el lab de Docker desplegaste **2 contenedores**: un backend (Flask) y un frontend (nginx). Funcionó perfecto. Pero eso fue para un proyecto chiquito en tu máquina.

En el mundo real, las aplicaciones crecen. Y cuando crecen, necesitás más contenedores por **tres razones**:

Razón 1

Microservicios

En vez de un programa gigante que hace todo, separás cada función en su propio contenedor. Login en uno, pagos en otro, notificaciones en otro. Si pagos tiene un bug y se cae, login sigue funcionando.

Razón 2

Réplicas para escalar

Si tenés 1 contenedor de backend y llegan 10.000 usuarios, no da abasto. Solución: corrés 5 copias del mismo backend. El tráfico se reparte entre las 5. Eso son 5 contenedores en vez de 1.

Razón 3

Servicios de soporte

Tu app necesita una base de datos (PostgreSQL), un cache (Redis), un proxy (nginx), monitoreo (Prometheus)... Cada uno corre en su propio contenedor.

Resultado

Decenas o cientos

Una app mediana en producción fácilmente tiene 20-50 contenedores. Una grande como Netflix o Mercado Libre tiene miles corriendo al mismo tiempo.

Así se ve una tienda online real con microservicios — cada cajita es un contenedor:

TIENDA ONLINE — CONTENEDORES EN PRODUCCIÓN

BACKEND (4 RÉPLICAS PARA AGUANTAR EL TRÁFICO)

api-1

api-2

api-3

api-4

FRONTEND (2 RÉPLICAS)

web-1

web-2

SERVICIOS DE SOPORTE

postgres

redis

nginx-proxy

prometheus

Son **12 contenedores** solo para una app. ¿Y quién arranca esos 12 contenedores? ¿En qué servidor va cada uno? ¿Qué pasa si uno se cae a las 3am?

EL PROBLEMA: MANEJAR TODO ESO A MANO ES IMPOSIBLE

Con `docker compose up` levantaste 2 contenedores en tu laptop. Pero en producción:

- Tenés **varios servidores**, no solo tu laptop. ¿En cuál va cada contenedor?
- Un contenedor se **cae a las 3am** — ¿quién lo reinicia? Vos estás durmiendo.
- Es Black Friday y el tráfico se **triplica** — ¿quién levanta 4 réplicas más del backend?
- Querés actualizar la **sin** app **cortarla** error? — ¿cómo hacés para que los usuarios no vean un error?
- El contenedor `api-3` necesita hablar con `postgres` — ¿cómo **se encuentran** entre sí?

`docker compose` funciona para **una sola máquina**. En producción, con múltiples servidores, necesitás algo más potente. Necesitás un **orquestador**.

¿QUÉ ES KUBERNETES?

EN UNA FRASE

Kubernetes (K8s) es un programa que coordina contenedores automáticamente a través de múltiples servidores. Vos le decís "quiero 4 copias de mi backend siempre corriendo", y K8s se encarga de que así sea — arranque, caídas, escalado, todo.

Fue creado por **Google** (que corría contenedores internamente desde 2003) y liberado como open source en 2014. Hoy lo mantiene la **CNCF** (Cloud Native Computing Foundation) y lo usan todas las empresas grandes del mundo.

ANALOGÍA: EL DIRECTOR DE ORQUESTA

Tus contenedores son los músicos. Sin dirección, cada uno toca cuando quiere y si uno se enferma, nadie lo reemplaza. **Kubernetes es el director**: decide quién toca cuándo, reemplaza a un músico si se enferma, y agrega más si el concierto crece. Vos solo escribís la **partitura** (un archivo YAML) — el director hace el resto.

LA IDEA CLAVE: ESTADO DECLARATIVO

Con Docker decís "levantá este contenedor" (una **orden**). Con Kubernetes decís "quiero que haya 3 réplicas de mi backend siempre corriendo" (una **declaración**). K8s lee eso y hace lo que sea necesario para cumplirlo: si hay 0, crea 3. Si se cae 1, crea otro. Si hay 5, mata 2. Vos describís el resultado, K8s se encarga del cómo.

LOS 5 CONCEPTOS QUE TENÉS QUE SABER

Pod

La unidad mínima

Node

El servidor

Un Pod es un "envoltorio" alrededor de uno o más contenedores. En la mayoría de los casos, un Pod = un contenedor. Es lo que Kubernetes crea, destruye y mueve. Pensá en él como la cajita donde va tu contenedor.

Un Node es una máquina (física o virtual) donde corren los Pods. Tu cluster tiene varios Nodes. En AWS, cada Node es una **instancia EC2** — como las que usaste en la sesión 2.

Deployment

El estado deseado

Es donde decís "quiero 3 réplicas de mi backend corriendo siempre". K8s crea los Pods y si uno se cae, arranca otro automáticamente para mantener el número que pediste.

Service

La puerta de entrada

Si tenés 3 Pods del backend, ¿a cuál le mando el tráfico? El Service les da **una dirección fija** y reparte el tráfico entre todos. Es el load balancer interno de Kubernetes.

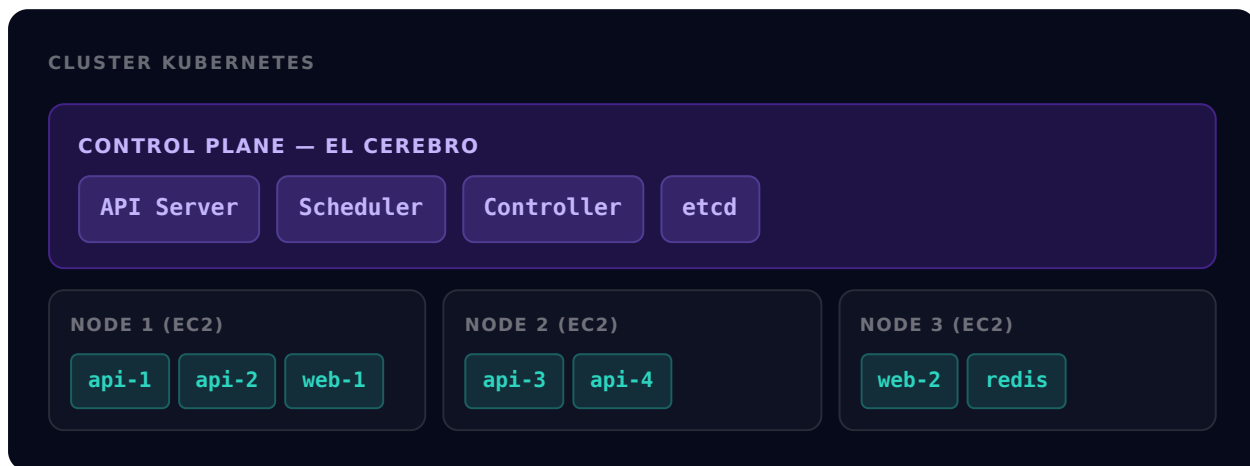
Namespace

La carpeta virtual

Separa recursos dentro del mismo cluster. Podés tener un Namespace `produccion` y otro `staging`, cada uno con sus propios Pods y Services sin interferir entre sí. Es como tener carpetas para organizar tu cluster.

CÓMO SE VE UN CLUSTER POR DENTRO

Un cluster tiene dos partes: el **Control Plane** (el cerebro que toma decisiones) y los **Nodes** (los servidores donde corren tus aplicaciones).



El **Control Plane** tiene 4 componentes:

- **API Server:** el punto de entrada. Todo pasa por acá: `kubectl apply`, le hablás al API Server cuando hacés.
- **etcd:** la base de datos del cluster. Guarda el estado de todo: cuántos Pods hay, en qué Node están, qué Deployments existen.
- **Scheduler:** cuando se crea un Pod nuevo, decide en qué Node ponerlo (el que tenga más recursos libres).
- **Controller:** vigila todo el tiempo que la realidad coincida con lo que pediste. Si dijiste "3 réplicas" y solo hay 2, crea la que falta.

En Docker escribiste un `Dockerfile` y un `docker-compose.yml`. En Kubernetes, todo se describe con **archivos YAML**. Son archivos de texto donde declararás lo que querés.

YAML ES SENSIBLE A LA INDENTACIÓN

Cada nivel se indenta con **2 espacios** (no tabs). Si la indentación está mal, K8s va a rechazar el archivo. Es el error más común al principio.

Un Deployment real — para correr 3 réplicas del backend de la sesión 2:

```
# backend-deployment.yaml
apiVersion: apps/v1           # versión de la API de K8s
kind: Deployment              # tipo de recurso
metadata:
  name: backend               # nombre del Deployment
spec:
  replicas: 3                  # quiero 3 copias corriendo
  selector:
    matchLabels:
      app: backend            # busca Pods con esta etiqueta
  template:
    metadata:
      labels:
        app: backend          # etiqueta del Pod
    spec:
      containers:
        name: backend
        image: mi-backend:1.0 # la imagen Docker
        ports:
          containerPort: 5000 # puerto del contenedor
```

Este archivo dice: "quiero 3 Pods corriendo, cada uno con un contenedor basado en la imagen `mi-backend:1.0`, escuchando en el puerto 5000".

Un Service — para exponer esos 3 Pods con una dirección fija:

```
# backend-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  selector:
    app: backend              # ¿a qué Pods apunta? a los que tengan app: backend
  ports:
    port: 5000                # puerto del Service
    targetPort: 5000          # puerto dentro del Pod
```

`type: ClusterIP`

`# solo accesible dentro del cluster`

Ahora cualquier Pod del cluster puede hablar con el backend usando la dirección

`backend:5000`. K8s se encarga de repartir el tráfico entre las 3 réplicas.

TIPOS DE SERVICE

ClusterIP — solo accesible dentro del cluster (default). **NodePort** — abre un puerto en cada Node para acceso externo. **LoadBalancer** — crea un balanceador de carga del cloud (en AWS, crea un ALB o NLB automáticamente).

COMANDOS ESENCIALES: KUBECTL

`kubectl` (se pronuncia "cube-control") es la herramienta de línea de comandos para hablar con Kubernetes. Es a K8s lo que `docker` es a Docker.

VER LO QUE HAY

<code>kubectl get pods</code>	Lista todos los Pods y su estado (Running, Pending, Error...)
<code>kubectl get services</code>	Lista los Services y sus IPs/puertos
<code>kubectl get deployments</code>	Lista los Deployments y cuántas réplicas tiene cada uno
<code>kubectl get all</code>	Muestra todo: Pods, Services, Deployments, ReplicaSets

CREAR Y MODIFICAR

<code>kubectl apply -f archivo.yaml</code>	Crea o actualiza lo que describe el archivo YAML
<code>kubectl delete -f archivo.yaml</code>	Borra todo lo que creó ese archivo
<code>kubectl scale deployment backend --replicas=5</code>	Cambia la cantidad de réplicas (de 3 a 5, por ejemplo)

DEPURAR Y DIAGNOSTICAR

<code>kubectl logs pod-name</code>	Muestra los logs de un Pod (como <code>docker logs</code>)
<code>kubectl describe pod pod-name</code>	Información detallada: eventos, errores, en qué Node está
<code>kubectl exec -it pod-name -- bash</code>	Abre una terminal dentro del Pod (como <code>docker exec</code>)
<code>kubectl port-forward svc/backend 5000:5000</code>	Redirige un puerto del cluster a tu máquina local

EL FLUJO DE TRABAJO COMPLETO



Después de hacer `kubectll apply -f` :

- El **API Server** recibe el YAML y lo guarda en etcd
- El **Controller** ve que pediste 3 réplicas y crea 3 Pods
- El **Scheduler** asigna cada Pod a un Node con recursos disponibles
- Si un Pod se cae, el Controller lo detecta y crea uno nuevo — **automáticamente**

LOS 3 SUPERPODERES DE KUBERNETES

Self-healing

Un Pod se cae → K8s lo detecta en segundos → crea uno nuevo. Sin que nadie toque nada. A las 3am también.

Auto-scaling

La CPU de tus Pods sube al 80% → K8s agrega más réplicas. El tráfico baja → K8s las reduce. Todo automático.

Rolling update

Querés desplegar la versión 2.0 → K8s va reemplazando Pods de a uno: sube un v2.0, baja un v1.0, hasta que todos están actualizados. Sin corte.

Ejemplo concreto del **rolling update**:

```
# Actualizás la imagen en el YAML (de 1.0 a 2.0) y aplicás:
kubectl apply -f backend-deployment.yaml

# K8s hace el rolling update automáticamente.
# Podés ver el progreso en tiempo real:
kubectl rollout status deployment/backend

# Si algo sale mal, volvés a la versión anterior:
kubectl rollout undo deployment/backend
```

DE DOCKER COMPOSE A KUBERNETES

Si ya sabés Docker Compose, ya conocés los conceptos — en K8s se llaman distinto:

DOCKER COMPOSE	KUBERNETES
<code>docker-compose.yml</code>	Varios archivos <code>.yaml</code> (Deployments + Services)
<code>services:</code>	<code>Deployment</code> + <code>Service</code>
<code>image:</code>	<code>spec.containers[].image:</code>
<code>ports:</code>	<code>Service.spec.ports:</code>
<code>depends_on:</code>	No existe — K8s usa Service discovery (se encuentran por nombre)

<code>docker compose up</code>	<code>kubectl apply -f .</code>
<code>docker compose down</code>	<code>kubectl delete -f .</code>
Una máquina	Múltiples servidores (cluster)
Sin auto-restart	Self-healing automático

LO QUE KUBERNETES NO HACE

K8s es un orquestador de contenedores, no una plataforma completa. Todavía necesitás:

- Un **registro de imágenes** donde guardar tus Docker images (Docker Hub, AWS ECR)
- **Monitoreo y logs** para ver qué pasa (Prometheus, Grafana, CloudWatch)
- **CI/CD** que construya las imágenes y las despliegue automáticamente
- Y si lo instalás vos: el **Control Plane** , que es la parte más compleja de mantener

Ahí es donde entra **EKS** — Kubernetes gestionado por AWS, donde el Control Plane lo maneja Amazon.