Clase 200 — Kubernetes para servir modelos a escala

Parte: 4 — MLOps · Fuente: Huyen cap. 11 + Burns et al., Kubernetes: Up and Running (3ª ed.) + docs k8s. Duración estimada: 90 min.

Clase 200 — Kubernetes para servir modelos a escala

> *Parte: 4 — MLOps · Fuente: Huyen cap. 11 + Burns et al., Kubernetes: Up and Running (3ª ed.) + docs k8s. Duración estimada: 90 min.*

Objetivo

Desplegar el contenedor de Clase 198–199 en Kubernetes con Deployment + Service + Ingress, autoescalar con HPA (CPU + custom metrics), hacer rolling updates seguros, y configurar livenessProbe/readinessProbe/resources correctamente. Aprender los 5 manifests mínimos que necesita cualquier servicio de inferencia.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Escribir manifests YAML para Deployment, Service, ConfigMap, Secret, HPA, Ingress.
- Diferenciar livenessProbe (reinicia pod) de readinessProbe (saca del LB) de startupProbe (gracia inicial para modelos lentos).
- Configurar resources.requests/limits y entender por qué un pod sin requests es OOM-killable y sin limits es noisy-neighbor.
- Hacer kubectl rollout/rollback y entender los maxSurge/maxUnavailable del rolling update.
- Diagnosticar CrashLoopBackOff, ImagePullBackOff, Pending, Evicted con kubectl describe y kubectl logs.

Temas

#	Tema	Por qué importa
1	Pod, Deployment, ReplicaSet, Service	Las abstracciones core.
2	Probes (liveness, readiness, startup)	Diferencia entre "el pod murió" y "el pod
3	Resources: requests vs limits	Scheduling + OOMkill control.
4	HPA: CPU + custom metrics (latency, queue	Autoescalado más allá de "CPU alta".
5	Rolling update + rollback	Deploy sin downtime y vuelta atrás.
6	Ingress + service mesh (mention)	Cómo expone tráfico externo.

Definiciones y características

- Pod: unidad de scheduling. 1+ containers que comparten network + storage. Para ML: típicamente 1 container por pod.
- Deployment: declara "quiero N réplicas de este pod con esta imagen". Gestiona un ReplicaSet que gestiona los pods. Rolling updates son responsabilidad del Deployment.
- Service: load balancer estable interno. Tipo ClusterIP (default, interno), NodePort (expone puerto en cada node), LoadBalancer (cloud LB).
- livenessProbe: si falla, K8s reinicia el container. Apuntar a /health con timeouts generosos.
- readinessProbe: si falla, K8s saca el pod del Service (no recibe tráfico) pero NO lo reinicia. Apuntar a

/ready que chequee modelo cargado + deps disponibles.
- startupProbe: ventana de gracia para containers lentos en arrancar (modelos grandes). Mientras corre, liveness/readiness no se ejecutan. Una vez OK, pasan a evaluarse normalmente.
- resources.requests: lo que el scheduler reserva para el pod. Sin requests, el pod va a "Best Effort" → primero en ser killed con presión de memoria.
- resources.limits: tope duro. Si memoria > limit: OOMKill. Si CPU > limit: throttling (no kill).
- HPA (Horizontal Pod Autoscaler): escala réplicas entre minReplicas y maxReplicas según métrica (default: CPU%). Custom metrics (latency p99, queue depth) requieren metrics-server + adapter (Prometheus Adapter).
- Rolling update: estrategia default. Crea pods nuevos (maxSurge), espera readiness, mata viejos (maxUnavailable). Garantiza disponibilidad continua.
- Rollback: kubectl rollout undo deployment/<name>. Vuelve al último ReplicaSet exitoso. Casi instantáneo si los pods viejos siguen referenciados.

Dataset / recursos

- Cluster: minikube, kind, o k3d para local. Equivalente en cloud: GKE/EKS/AKS.
- Imagen: la de Clase 198 (iris-api:v1).
- Herramientas: kubectl, kustomize (built-in) o helm.

Ejercicios

1. Cluster local: kind create cluster --name ml. Pusheá la imagen local con kind load docker-image iris-api:v1 --name ml. Verificá con kubectl get nodes.
2. Deployment + Service: aplicá los YAML del notebook. kubectl get pods -w mientras los 3 pods arrancan. kubectl port-forward svc/iris-api 8000:80 y pegale con curl localhost:8000/predict.
3. Probes: cambiá livenessProbe a apuntar a /wrong-endpoint. Observá con kubectl get pods -w cómo entra en CrashLoopBackOff. Revertí.
4. HPA: kubectl apply -f hpa.yaml con targetCPUUtilizationPercentage: 50. Generá carga con kubectl run loadtester --image=busybox -it --rm -- /bin/sh -c "while true; do wget -q -O- iris-api/predict; done". Observá kubectl get hpa -w escalar de 3 → 10.
5. Rolling update + rollback: cambiá la imagen a iris-api:v2 (versión rota a propósito). kubectl rollout status deployment/iris-api debería timeoutear. kubectl rollout undo deployment/iris-api y verificá recuperación.

Homework verificable

Cluster (local o cloud) con:

1. Manifests YAML para Deployment (3 réplicas, probes, resources), Service (ClusterIP), HPA (CPU 50%, min 3 max 10), Ingress.
2. Imagen pusheada a un registry (Docker Hub o ECR).
3. Loadtest que dispara HPA y demostrar escalado en logs/screenshots.
4. Rolling update a una versión v2 exitosa, después rollback con kubectl rollout undo.
5. README con comandos kubectl apply, port-forward, troubleshooting con describe.

Criterio de aceptación: kubectl get pods -l app=iris-api muestra 3-10 pods escalando con la carga, kubectl get hpa reporta target CPU, y curl <ingress-host>/predict funciona desde fuera del cluster.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Pods en Pending infinito	Cluster sin recursos para requests. Fix: k
CrashLoopBackOff	Container falla al iniciar y K8s reintenta
ImagePullBackOff	No puede bajar la imagen. Fix: chequear no
Liveness mata el pod durante startup del m	livenessProbe arranca antes que el modelo
OOMKilled silencioso	kubectl describe pod muestra "OOMKilled" e
HPA no escala aunque CPU está alta	metrics-server no está instalado o el pod

Preguntas frecuentes

¿K8s vs Cloud Run vs ECS Fargate vs Lambda?

K8s es el más flexible y portable; el más pesado en mantenimiento. Cloud Run / Fargate: contenedor managed, escalan a 0, sin K8s machinery — buen punto intermedio. Lambda (Clase 201): serverless puro, frío, máx 15 min, ideal para batch chico. Para ML serving 24/7: K8s o Cloud Run.

¿1 worker por pod o N workers por pod?

Recomendación K8s: 1 worker por pod. K8s se encarga del fleet (HPA, rolling update, restart). Múltiples workers complican observabilidad por pod, hacen restart costoso, y suelen reproducir lo que ya hace K8s.

¿GPU pods?

Necesitás un node pool con GPUs y el NVIDIA device plugin instalado. En el pod: resources.limits["nvidia.com/gpu"]: 1. La imagen base debe ser CUDA-compatible (nvidia/cuda:... o pytorch/pytorch:cuda).

¿Helm o Kustomize?

Kustomize (built-in en kubectl): overlays por entorno (dev/staging/prod) sin templating string-based. Helm: package manager con templates Go, más expresivo pero también más complejo. Para empezar: Kustomize. Para distribuir charts (ej. instalar Prometheus): Helm.

¿Cómo manejo secrets?

Secret resource (base64, NO encripta por default). Para producción: enable encryption-at-rest en etcd, o mejor: External Secrets Operator que sincroniza de AWS Secrets Manager / Vault → K8s Secrets. Nunca commitear secrets a git, ni siquiera "fake" — usá kubectl create secret o secrets externos.

¿Service mesh (Istio/Linkerd) es necesario?

No para empezar. Resuelve: mTLS automático entre pods, retries/timeouts/circuit breaking, canary releases (Clase 204) sin tocar código, observabilidad fina. Agrega complejidad (sidecar por pod, control plane). Vale la pena cuando: ≥10 microservicios, requerimiento mTLS interno, o canary serio.

Referencias

- Burns et al. Kubernetes: Up and Running (3ª ed., O'Reilly, 2022).
- Kubernetes docs — Workloads, Services, Scheduling, Preemption and Eviction.
- Probes guide — patrones correctos.

- Horizontal Pod Autoscaler walkthrough.
- Kind quickstart — cluster local en segundos.

Siguiente clase

Clase 201 — Serverless ML: AWS Lambda, GCP Cloud Functions

Apéndice: notebook (primer bloque)

Notebook declarativo: genera los 5 manifests YAML mínimos para desplegar el iris-api en K8s. Para verlos en vivo: `bash kind create cluster --name ml

```
import os, shutil, tempfile
from pathlib import Path
WORK = Path(tempfile.gettempdir()) / 'k8s_demo'
if WORK.exists(): shutil.rmtree(WORK)
(WORK / 'k8s').mkdir(parents=True)
os.chdir(WORK)
print('cwd:', Path.cwd())
```

Archivos complementarios

- notebook.ipynb