Clase 204 — Shadow deployment y canary releases

Parte: 4 — MLOps · Fuente: Huyen cap. 11 + Fowler, Continuous Delivery + Istio traffic management. Duración estimada: 75 min.

Clase 204 — Shadow deployment y canary releases

> *Parte: 4 — MLOps · Fuente: Huyen cap. 11 + Fowler, Continuous Delivery + Istio traffic management. Duración estimada: 75 min.*

Objetivo

Desplegar un modelo nuevo sin arriesgar producción: primero en shadow (recibe tráfico real pero sus predicciones no se devuelven al usuario), después canary (1% → 5% → 25% → 100% del tráfico). Convertir "creo que esto es mejor" en "lo medí en producción real".

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Implementar shadow mode: doble call (champion + challenger), respuesta del champion al user, log de ambas para análisis offline.
- Configurar canary release progresivo (1% → 5% → 25% → 100%) con K8s + Istio, o con feature flag a nivel app (LaunchDarkly, Unleash).
- Definir rollback automático: si la métrica del canary degrada >X%, volver al 100% champion en <5 min.
- Implementar A/B test correcto: muestra del mismo segmento, métrica primaria pre-registrada, poder estadístico calculado.
- Diferenciar shadow (sin riesgo, costo doble) de canary (riesgo limitado, costo nuevo solo).

Temas

#	Tema	Por qué importa
1	Shadow vs canary vs blue-green vs A/B	4 estrategias, propósitos distintos.
2	Implementación: app-level vs infra-level	Feature flag (LaunchDarkly) vs Istio/AWS A
3	Métrica de "salud" del canary	Latency, error rate, business KPI proxy.
4	Rollback automático	Sin esto, canary es solo "esperar a que al
5	Análisis de shadow data	Comparación distribucional (KS) + métricas
6	A/B test riguroso	Effect size + poder + duración mínima.

Definiciones y características

- Shadow deployment: el modelo nuevo recibe el mismo tráfico que el viejo, predice, pero su predicción NO se devuelve al usuario. Se loggea para análisis. Ventaja: cero riesgo. Costo: 2× compute.
- Canary release: % del tráfico real es servido por el modelo nuevo. Empezar 1%, escalar gradual. Si métricas OK al 100%, el canary se convierte en champion. Si no, rollback.
- Blue-green: dos versiones desplegadas simultáneamente (blue=actual, green=nueva). Switch del LB de blue→green en un momento puntual. Rollback = switch back. Sin gradualidad, pero rollback instantáneo.
- A/B test: comparación estadística entre dos variantes con asignación random por usuario (sticky). Mide impacto en KPI de negocio (no solo en accuracy). Requiere poder estadístico calculado.

- Sticky assignment: el mismo usuario siempre cae en la misma variante (hash por user_id). Evita que un usuario vea ambos modelos y "engañe" la medición.
- Rollback automático: cuando alguna métrica clave del canary cae bajo un umbral (latency p99 > X, error rate > Y%, conversion rate cae > Z%), el sistema vuelve el peso del canary a 0 sin intervención humana.
- Guardrail metric: métrica que NO querés que empeore aunque la primaria mejore. Ej: latency, errors. Si guardrail rompe → rollback aunque "el modelo prediga mejor".

Dataset / recursos

- Stack ejemplo: FastAPI + Istio (canary) o feature flag in-process (shadow simple).
- Librerías: scipy.stats (A/B test), numpy.

Ejercicios

1. Shadow en proceso: en el FastAPI de Clase 199, agregá la lógica: cargá model_champion y model_challenger. En /predict, predeci con ambos, devolvé solo champion, log los dos en JSON. Después de 1000 requests, analizá la distribución de diferencias.
2. Canary con feature flag: implementá un toggle CANARY_PERCENT (env var). Si random.random() < CANARY_PERCENT/100, usá challenger. Sticky: hash por user_id para que el mismo user vea siempre lo mismo dentro del test.
3. Canary con Istio: VirtualService con weight: 95 / 5. kubectl apply y verificá distribución de tráfico con kubectl logs.
4. Rollback automático: agregá un sidecar (Python script) que cada 60 s consulta Prometheus: si latency_p99{model=challenger} > 200ms, cambia el weight a 100 / 0 automáticamente.
5. A/B test riguroso: calculá tamaño de muestra para detectar δ = 0.02 en accuracy con α=0.05, power=0.8. Corré el A/B test el tiempo necesario para acumular ese N. Reportá p-value + CI de la diferencia.

Homework verificable

Sistema de deployment con:

1. Modelo champion en producción sirviendo 100% del tráfico.
2. Endpoint /admin/canary que setea peso del challenger (sticky por user_id hash).
3. Métricas en Prometheus: predictions_total{model, class}, latency_seconds{model}, error_rate{model}.
4. Alerta + auto-rollback si: latency_p99{model=challenger} > 1.2 * latency_p99{model=champion} durante 5 min seguidos.
5. Dashboard Grafana con: distribución de tráfico, latency por modelo, agreement rate champion-challenger, business KPI proxy.
6. Runbook documentando los pasos: shadow 7 días → canary 1% 24 h → 5% 24 h → 25% 48 h → 100%.

Criterio de aceptación: simular un challenger con time.sleep(0.5) artificial (alta latencia). El rollback automático debe activarse en <10 min, sin pérdida de servicio para usuarios fuera del canary.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Canary muestra mejor accuracy pero peor bu	Métrica offline (accuracy) ≠ métrica onlin
El A/B test "ganó" pero la decisión no se	Regression to the mean + tamaño de muestra
El user ve modelos distintos en distintos	Asignación no es sticky. Fix: hash por use
Shadow no es "free" como prometieron — el	2× compute requiere 2× capacidad. Fix: dim
Canary a 5% no muestra estadística signifi	El N es chico — 5% × N usuarios típicos re
Rollback automático se dispara por ruido	Umbral muy estricto o ventana muy chica. F

Preguntas frecuentes

¿Shadow obligatorio antes de canary?

Para modelos críticos: sí. Shadow detecta bugs evidentes (modelo cae, output con NaN) sin afectar usuarios. Canary asume que el modelo funciona y mide impacto. Si saltás shadow, podés desplegar un modelo roto al 1% real.

¿Canary o blue-green?

Canary: gradual, deteccion temprana, recovery limitado. Blue-green: rollback instantáneo (un switch), sin gradualidad — el 100% se va o no se va. Para ML: canary es más común (cambios en distribución de tráfico revelan problemas que blue-green no).

¿Tengo que usar Istio?

No. Alternativas:

- AWS ALB/NLB con weighted target groups
- GCP Cloud Load Balancing weighted backends
- App-level feature flag (LaunchDarkly, Unleash) — más simple, opera fuera del infra
- Nginx con split_clients

Istio agrega valor cuando ya tenés service mesh.

¿Cómo elijo la métrica primaria del A/B?

Debe ser: (1) directly tied to business value (revenue, retention, conversion), (2) medible rápido (mejor: detectable en días, no meses), (3) una sola (multi-metric = comparison-shopping, mala práctica). Métricas técnicas (accuracy, AUC) son proxies — útiles pero no la métrica final.

¿Y si el canary mejora un segmento y empeora otro?

Análisis por segmento siempre. Si segmento A mejora 10% y segmento B empeora 5%: discutir trade-off con producto. Algunas opciones: (a) modelo distinto por segmento, (b) re-entrenar con foco en B, (c) decidir explícitamente que A pesa más.

¿Stat test correcto para A/B con accuracy?

Para proporciones (CTR, conversion): test de proporciones con corrección de continuidad. Para medias (latencia, revenue/user): t-test de Welch (no asume varianzas iguales). Para counts (clicks/usuario): Mann-Whitney o bootstrap (típicamente no-normal). Ver Partes 3 (clases 176-178) para más detalle.

Referencias

- Huyen, Chip. Designing Machine Learning Systems (O'Reilly, 2022), cap. 11 — Model Deployment y

Continuous Delivery.
- Fowler, M. — CanaryRelease y BlueGreenDeployment.
- Istio Traffic Splitting tutorial.
- Kohavi, R. et al. Trustworthy Online Controlled Experiments (Cambridge, 2020) — la biblia del A/B test.
- LaunchDarkly / Unleash — feature flags as a service.

Siguiente clase

Clase 205 — Interpretabilidad: SHAP, LIME, PDP, ICE

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import numpy as np, hashlib, time, random
from collections import defaultdict
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

X, y = load_breast_cancer(return_X_y=True)
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.3, random_state=42)

champion = LogisticRegression(max_iter=5000).fit(Xtr, ytr)
challenger = RandomForestClassifier(n_estimators=200, random_state=42).fit(Xtr, ytr)
print(f'champion offline acc: {accuracy_score(yte, champion.predict(Xte)):.4f}')
print(f'challenger offline acc: {accuracy_score(yte, challenger.predict(Xte)):.4f}')
```

Archivos complementarios

- notebook.ipynb