MÓDULO 12: SERVERLESS VS CONTENEDORES

Cuándo Usar Lambda, ECS o Kubernetes

Guía MLOps v5.0: Senior Edition | DuqueOM | Noviembre 2025

MÓDULO 12: Serverless vs Contenedores

La Decisión que Define tu Arquitectura

"No hay solución universal. Hay trade-offs que debes entender."

Duración	Teoría	Práctica
4-5 horas	40%	60%

ADR: ¿Dónde Desplegar?

```
╔═══════════════════════════════════════════════════════════════════╗
║  ADR-008: Selección de Plataforma de Despliegue                   ║
╠═══════════════════════════════════════════════════════════════════╣
║                                                                   ║
║  OPCIONES:                                                        ║
║  1. Serverless (AWS Lambda, GCP Cloud Functions)                  ║
║  2. Contenedores Managed (ECS, Cloud Run)                         ║
║  3. Kubernetes (EKS, GKE, self-managed)                           ║
║                                                                   ║
║  FACTORES DE DECISIÓN:                                            ║
║  • Tráfico esperado (requests/mes)                                ║
║  • Requisitos de latencia                                         ║
║  • Tamaño del equipo de Ops                                       ║
║  • Presupuesto                                                    ║
║  • Complejidad del modelo (GPU, memoria)                          ║
║                                                                   ║
╚═══════════════════════════════════════════════════════════════════╝
```

12.1 Matriz de Decisión

```
╔═══════════════════════════════════════════════════════════════════╗
║                    MATRIZ DE DECISIÓN DE DESPLIEGUE               ║
╠═══════════════════════════════════════════════════════════════════╣
║                                                                   ║
║  Factor              | Lambda/Serverless | ECS/Cloud Run | Kubernetes       ║
║  ────────────────────┼───────────────────┼──────────────┼──────────────────  ║
║  Tráfico             | < 1M req/mes      | 1M-100M      | > 100M           ║
║  Latencia            | Variable (cold)   | Consistente  | Consistente      ║
║  Costo bajo tráfico  |  Muy bajo        |  Medio     |  Alto        ║
║  Costo alto tráfico  |  Caro        |  Medio     |  Barato        ║
║  Complejidad Ops     |  Baja            |  Media     |  Alta        ║
║  Equipo necesario    | 1 persona         | 2-3 personas | 5+ personas      ║
║  GPU Support         | x                 |              |                  ║
║  Max memoria         | 10GB              | 120GB+       | Ilimitado        ║
║  Max timeout         | 15 min            | Ilimitado    | Ilimitado        ║
║  Modelo size límite  | ~250MB pkg        | Sin límite   | Sin límite       ║
║  Auto-scaling        | Automático        | Automático   | Configurable     ║
║  Vendor lock-in      | Alto              | Medio        | Bajo             ║
║                                                                   ║
╚═══════════════════════════════════════════════════════════════════╝
```

12.2 Opción 1: Serverless (AWS Lambda)

Cuándo Usar

```
 USA LAMBDA SI:
• Tráfico bajo o esporádico (< 1M requests/mes)
• Modelo pequeño (< 250MB empaquetado)
• Latencia variable es aceptable
• No tienes equipo de DevOps
• Quieres minimizar costos en bajo tráfico

× NO USES LAMBDA SI:
• Necesitas GPU
• Modelo > 250MB
• Cold starts son inaceptables (< 100ms requerido)
• Tráfico constante y alto
```

Estructura para Lambda

```
lambda_function/
├── handler.py          # Entry point
├── model/
│   └── pipeline.pkl    # Modelo (< 250MB)
├── src/
│   └── inference.py    # Lógica
└── requirements.txt
```

handler.py

```python
# handler.py - AWS Lambda Handler
import json
import joblib
import pandas as pd
from pathlib import Path

# Cargar modelo al inicio (fuera del handler para reutilizar)
MODEL_PATH = Path(__file__).parent / "model" / "pipeline.pkl"
model = joblib.load(MODEL_PATH)

def lambda_handler(event, context):
    """AWS Lambda handler."""
    try:
        # Parse input
        if isinstance(event.get("body"), str):
            body = json.loads(event["body"])
        else:
            body = event.get("body", event)

        # Crear DataFrame
        df = pd.DataFrame([body])

        # Predecir
        proba = model.predict_proba(df)[0, 1]
        prediction = "churn" if proba >= 0.5 else "no_churn"

        return {
            "statusCode": 200,
            "headers": {"Content-Type": "application/json"},
            "body": json.dumps({
                "churn_probability": round(proba, 4),
                "prediction": prediction,
            })
        }
    except Exception as e:
        return {
            "statusCode": 500,
            "body": json.dumps({"error": str(e)})
        }
```

serverless.yml (Serverless Framework)

```yaml
# serverless.yml
service: bankchurn-predictor

provider:
  name: aws
  runtime: python3.11
  region: us-east-1
  memorySize: 1024
  timeout: 30

functions:
  predict:
    handler: handler.lambda_handler
    events:
      - http:
          path: predict
          method: post
          cors: true

plugins:
  - serverless-python-requirements

custom:
  pythonRequirements:
    dockerizePip: true
    slim: true
```

12.3 Opción 2: Contenedores Managed (AWS ECS / GCP Cloud Run)

Cuándo Usar

```
 USA ECS/CLOUD RUN SI:
• Tráfico medio-alto (1M-100M requests/mes)
• Necesitas latencia consistente
• Modelo de cualquier tamaño
• Quieres balance entre control y simplicidad
• Equipo pequeño de DevOps (2-3 personas)

× NO USES SI:
• Necesitas control granular de networking
• Multi-cloud es requisito
• Tráfico extremadamente alto (> 100M)
```

AWS ECS Task Definition

```
{
  "family": "bankchurn-api",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "512",
  "memory": "1024",
  "containerDefinitions": [
    {
      "name": "api",
      "image": "123456789.dkr.ecr.us-east-1.amazonaws.com/bankchurn:latest",
      "portMappings": [
        {
          "containerPort": 8000,
          "protocol": "tcp"
        }
      ],
      "environment": [
        {"name": "LOG_LEVEL", "value": "INFO"}
      ],
      "healthCheck": {
        "command": ["CMD-SHELL", "curl -f http://localhost:8000/health || exit 1"],
        "interval": 30,
        "timeout": 5,
        "retries": 3
      },
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/bankchurn",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "api"
        }
      }
    }
  ]
}
```

GCP Cloud Run (más simple)

```
# Deploy a Cloud Run
gcloud run deploy bankchurn-api \
  --image gcr.io/my-project/bankchurn:latest \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated \
  --memory 1Gi \
  --cpu 1 \
  --min-instances 0 \
  --max-instances 10 \
  --port 8000
```

12.4 Opción 3: Kubernetes

Cuándo Usar

```
 USA KUBERNETES SI:
• Tráfico muy alto (> 100M requests/mes)
• Múltiples servicios ML que escalan diferente
• Necesitas GPU para inferencia
• Multi-cloud o hybrid cloud
• Equipo de Ops experimentado (5+ personas)
• Ya tienes inversión en K8s

× NO USES SI:
• Un solo modelo simple
• Equipo pequeño sin experiencia K8s
• Presupuesto limitado para Ops
```

Manifiestos Básicos

```yaml
# k8s/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bankchurn-api
  labels:
    app: bankchurn-api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: bankchurn-api
  template:
    metadata:
      labels:
        app: bankchurn-api
    spec:
      containers:
      - name: api
        image: ghcr.io/username/bankchurn:latest
        ports:
        - containerPort: 8000
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        readinessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 10
          periodSeconds: 5
        livenessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 15
          periodSeconds: 10
        env:
        - name: LOG_LEVEL
          value: "INFO"
---
# k8s/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: bankchurn-api
spec:
  selector:
    app: bankchurn-api
  ports:
  - port: 80
    targetPort: 8000
  type: ClusterIP
---
# k8s/hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: bankchurn-api
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: bankchurn-api
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
```

12.5 Análisis de Costos (FinOps)

```
╔═══════════════════════════════════════════════════════════════╗
║                  ANÁLISIS DE COSTOS MENSUAL                   ║
╠═══════════════════════════════════════════════════════════════╣
║                                                               ║
║  ESCENARIO: 1M requests/mes, ~1 req/seg promedio              ║
║                                                               ║
║  AWS Lambda:                                                  ║
║  • 1M requests × $0.20/1M = $0.20                             ║
║  • 1M × 200ms × 1GB = 200K GB-s × $0.0000166 = $3.32          ║
║  • Total: ~$4/mes  (bajo tráfico es barato)                   ║
║                                                               ║
║  ECS Fargate:                                                 ║
║  • 0.5 vCPU × 730h × $0.04 = $14.60                           ║
║  • 1GB RAM × 730h × $0.004 = $2.92                            ║
║  • Total: ~$18/mes (consistente)                              ║
║                                                               ║
║  EKS (3 nodos t3.small):                                      ║
║  • 3 × $15/mes (EC2) = $45                                    ║
║  • EKS fee: $72/mes                                           ║
║  • Total: ~$120/mes (overkill para este volumen)              ║
║                                                               ║
╠═══════════════════════════════════════════════════════════════╣
║                                                               ║
║  ESCENARIO: 100M requests/mes, ~40 req/seg promedio           ║
║                                                               ║
║  AWS Lambda:                                                  ║
║  • 100M × $0.20/1M = $20                                      ║
║  • 100M × 200ms × 1GB = 20M GB-s × $0.0000166 = $332          ║
║  • Total: ~$350/mes (ya no tan barato)                        ║
║                                                               ║
║  ECS Fargate (auto-scaling):                                  ║
║  • ~5 tareas promedio                                         ║
║  • Total: ~$90/mes                                            ║
║                                                               ║
║  EKS (auto-scaling):                                          ║
║  • 5 nodos t3.medium promedio                                 ║
║  • Total: ~$200/mes                                           ║
║                                                               ║
╚═══════════════════════════════════════════════════════════════╝
```

12.6 Decisión para BankChurn

Recomendación por Fase

Fase	Plataforma	Razón
MVP/Desarrollo	Cloud Run o Lambda	Simplicidad, bajo costo inicial
Producción inicial	ECS/Cloud Run	Balance costo-control
Escala enterprise	Kubernetes	Control total, multi-service

ADR para BankChurn

```
╔═══════════════════════════════════════════════════════════════╗
║  ADR-009: Despliegue de BankChurn en Cloud Run                ║
╠═══════════════════════════════════════════════════════════════╣
║                                                               ║
║  DECISIÓN: Usar Google Cloud Run para el MVP                  ║
║                                                               ║
║  RAZONES:                                                     ║
║  • Escala a cero cuando no hay tráfico (costo mínimo)         ║
║  • Sin gestión de infraestructura                             ║
║  • Latencia consistente (mejor que Lambda para ML)            ║
║  • Soporta contenedores Docker estándar                       ║
║  • Fácil migración a GKE si necesario                         ║
║                                                               ║
║  TRADE-OFFS ACEPTADOS:                                        ║
║  • Vendor lock-in medio (GCP)                                 ║
║  • Menos control que K8s                                      ║
║                                                               ║
╚═══════════════════════════════════════════════════════════════╝
```

12.7 Ejercicio: Deploy a Cloud Run

```
# 1. Build imagen
docker build -t gcr.io/my-project/bankchurn:v1 .

# 2. Push a GCR
docker push gcr.io/my-project/bankchurn:v1

# 3. Deploy
gcloud run deploy bankchurn \
  --image gcr.io/my-project/bankchurn:v1 \
  --platform managed \
  --region us-central1 \
  --memory 1Gi \
  --allow-unauthenticated

# 4. Test
curl -X POST https://bankchurn-xxx.run.app/api/v1/predict \
  -H "Content-Type: application/json" \
  -d '{"credit_score": 650, "age": 35, ...}'
```

Siguiente Paso

Con la plataforma elegida, es hora de gestionar **infraestructura como código**.

Ir a Módulo 13: Terraform Modular →

Módulo 12 completado. Ya sabes dónde desplegar según tu contexto.