MÓDULO 19: DECISIONES TECNOLÓGICAS

Por qué Elegimos Cada Herramienta

Guía MLOps v2.0 | DuqueOM | Noviembre 2025

⚖ MÓDULO 19: Decisiones Tecnológicas

Por qué Elegimos Cada Herramienta

"Cada decisión tiene trade-offs, documéntalos."

Nivel	Duración
Referencia	Consulta

Objetivo

Este módulo documenta las decisiones tecnológicas del portafolio, con justificaciones, alternativas consideradas y trade-offs. Sirve como referencia para entender e "por qué" detrás de cada elección.

Matriz de Decisiones

1. Lenguaje de Programación

Aspecto	Decisión	Alternativas	Justificación
Lenguaje	Python 3.11+	R, Julia, Scala	Ecosistema ML más maduro, mayor comunidad, mejor integración con herramientas MLOps

Pros de Python: - Librerías ML más completas (sklearn, pytorch, tensorflow) - Mejor soporte para APIs web (FastAPI) - Mayor cantidad de recursos de aprendizaj - Integración nativa con herramientas MLOps

Contras de Python: - Más lento que lenguajes compilados - GIL limita paralelismo verdadero

2. Framework de ML

Aspecto	Decisión	Alternativas	Justificación
ML Framework	scikit-learn	XGBoost, LightGBM, PyTorch	Suficiente para clasificación/regresión tabular, API consistente, fácil serialización

Cuándo usar cada uno:

Framework	Caso de Uso
scikit-learn	Datos tabulares, modelos clásicos, interpretabilidad
XGBoost/LightGBM	Competiciones, máximo rendimiento en tabular
PyTorch/TensorFlow	Deep learning, imágenes, NLP, series temporales complejas

En este portafolio:

```
# Usamos sklearn porque:
# 1. Datos tabulares (CSV)
# 2. Pipeline unificado (preprocessor + model)
# 3. Fácil serialización con joblib
# 4. Interpretable para demo

from sklearn.ensemble import VotingClassifier, RandomForestClassifier
from sklearn.pipeline import Pipeline
```

---

3. Versionado de Datos

Aspecto	Decisión	Alternativas	Justificación
Data Versioning	DVC	Git LFS, LakeFS, Delta Lake	Integra con Git, open source, popular en portafolios

Comparación detallada:

Herramienta	Pros	Contras	Mejor Para
DVC	Git-friendly, pipelines, remotes flexibles	Curva de aprendizaje	Proyectos medianos, portfolios
Git LFS	Simple, nativo Git	Sin pipelines, costoso a escala	Archivos pequeños (<1GB)
LakeFS	Branching de datos, escala masiva	Setup complejo	Data lakes empresariales
Delta Lake	ACID, Spark integration	Requiere Spark	Big data, Databricks

Configuración elegida:

```
# .dvc/config
[core]
    remote = myremote
[remote "myremote"]
    url = /path/to/dvc-storage  # Local para demo
    # url = s3://bucket/dvc     # S3 para producción
```

---

4. Experiment Tracking

Aspecto	Decisión	Alternativas	Justificación
Tracking	MLflow	W&B, Neptune, Comet	Open source, self-hosted, control total

Comparación:

Herramienta	Tipo	UI	Costo	Mejor Para
MLflow	Open Source	Básica	Gratis	Control total, on-premise
W&B	SaaS	Excelente	Free tier + paid	Colaboración, visualización
Neptune	SaaS	Buena	Free tier + paid	Equipos medianos
Comet	SaaS	Buena	Free tier + paid	Computer vision

Decisión en el portafolio:

```
# MLflow porque:
# 1. Self-hosted (sin dependencia externa)
# 2. Gratuito sin límites
# 3. Integra bien con sklearn
# 4. Model Registry incluido

import mlflow
mlflow.set_tracking_uri("file:./mlruns")  # Local
# mlflow.set_tracking_uri("http://mlflow:5000")  # Docker
```

---

5. Framework de API

Aspecto	Decisión	Alternativas	Justificación
API Framework	FastAPI	Flask, Django, Starlette	Async, tipado, docs automáticos, moderno

Comparación:

Framework	Performance	Docs Auto	Async	Typing	Learning Curve
FastAPI		Swagger		Pydantic	Media
Flask		× Manual	×	×	Baja
Django		DRF	⚠	⚠	Alta

Por qué FastAPI:

```
# 1. Documentación automática en /docs
# 2. Validación con Pydantic
# 3. Async para alta concurrencia
# 4. Type hints = menos bugs

from fastapi import FastAPI
from pydantic import BaseModel

class CustomerData(BaseModel):
    CreditScore: int
    Age: int
    # Validación automática

@app.post("/predict")
async def predict(data: CustomerData):  # Async
    return model.predict(data.dict())
```

---

6. Contenerización

Aspecto	Decisión	Alternativas	Justificación
Containers	Docker + Compose	Podman, containerd	Estándar de industria, mejor documentación

Dockerfile elegido:

```
# Multi-stage build para imagen pequeña
FROM python:3.13-slim AS builder
# ... dependencias de build ...

FROM python:3.13-slim AS runtime
# Usuario no-root para seguridad
RUN useradd -r appuser
USER appuser
```

Beneficios del enfoque: - Imagen final ~50% más pequeña - Usuario no-root por seguridad - ⚡ Layer caching para builds rápidos

---

7. CI/CD

Aspecto	Decisión	Alternativas	Justificación
CI/CD	GitHub Actions	GitLab CI, CircleCI, Jenkins	Integración nativa con GitHub, gratis para open source

Comparación:

Herramienta	Integración GitHub	Free Tier	Self-Hosted	Config
GitHub Actions		2000 min/mes		YAML
GitLab CI		400 min/mes		YAML
CircleCI		6000 min/mes	×	YAML
Jenkins		Ilimitado		Groovy

Estructura elegida:

```
# Un workflow unificado para todo el portafolio
# .github/workflows/ci-mlops.yml

jobs:
  tests:        # Matrix testing por proyecto
  quality:      # Linting y formatting
  security:     # Bandit, Trivy, Gitleaks
  docker:       # Build y scan de imágenes
  integration:  # E2E con docker-compose
```

---

8. Orquestación

Aspecto	Decisión	Alternativas	Justificación
Orquestación	Kubernetes (manifiestos) + Helm	Docker Swarm, ECS, Nomad	Estándar de industria, portabilidad

Cuándo usar cada opción:

Opción	Complejidad	Escala	Mejor Para
Docker Compose		Pequeña	Demo, desarrollo local
Docker Swarm		Media	Equipos pequeños
Kubernetes		Grande	Producción enterprise
Serverless		Variable	Cargas esporádicas

En el portafolio: - `docker-compose.demo.yml` → Demo local - `k8s/` → Manifiestos para producción real

---

9. Infrastructure as Code

Aspecto	Decisión	Alternativas	Justificación
IaC	Terraform	CloudFormation, Pulumi, CDK	Multi-cloud, declarativo, ecosistema maduro

Comparación:

Herramienta	Multi-Cloud	Lenguaje	Estado	Learning Curve
Terraform		HCL	Remote	Media
CloudFormation	× AWS only	YAML/JSON	AWS	Media
Pulumi		Python/TS/Go	Remote	Media
CDK	⚠	Python/TS	CloudFormation	Alta

Estructura elegida:

```
infra/terraform/
├── aws/
│   ├── main.tf      # EKS, S3, RDS, ECR
│   └── variables.tf
└── gcp/
    └── main.tf      # GKE, GCS, CloudSQL
```

---

10. Monitoreo

Aspecto	Decisión	Alternativas	Justificación
Metrics	Prometheus + Grafana	Datadog, New Relic	Open source, estándar K8s
Drift	Evidently	WhyLogs, NannyML	Fácil integración, visualización clara

Stack de monitoreo:

```yaml
# docker-compose.demo.yml (profile: monitoring)
services:
  prometheus:
    image: prom/prometheus:v2.48.0
    ports: ["9090:9090"]

  grafana:
    image: grafana/grafana:10.2.2
    ports: ["3000:3000"]
```

---

11. Testing

Aspecto	Decisión	Alternativas	Justificación
Testing	pytest + coverage	unittest, nose	Fixtures, plugins, sintaxis limpia
Data Validation	Pydantic	Great Expectations, Pandera	Ya usado en FastAPI, consistencia

Configuración elegida:

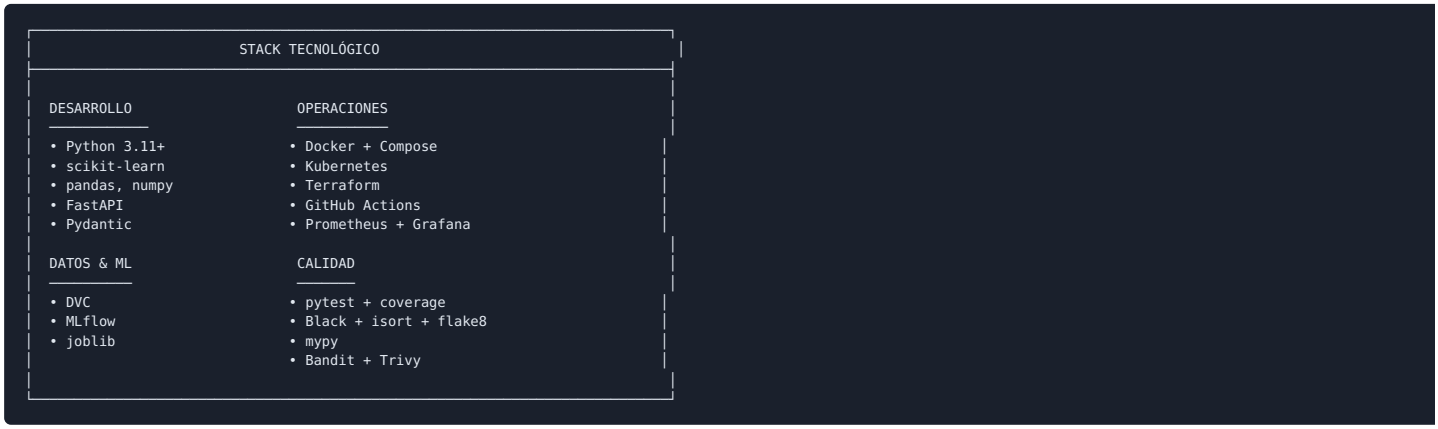
```
# pyproject.toml
[tool.pytest.ini_options]
addopts = [
    "--cov=src",
    "--cov-fail-under=70",
    "--cov-report=html"
]
```

---

12. Calidad de Código

Herramienta	Propósito	Configuración
Black	Formateo	`line-length = 120`
isort	Ordenar imports	`profile = "black"`
flake8	Linting	`max-line-length = 120`
mypy	Type checking	`ignore_missing_imports = true`
bandit	Security	`-ll` (medium+ severity)
gitleaks	Secrets	Custom `.gitleaks.toml`

Resumen de Stack Tecnológico



Cuándo Cambiar de Tecnología

Señales para Escalar

Actual	Escalar a	Cuándo
sklearn	XGBoost/LightGBM	Necesitas 1-2% más de accuracy
sklearn	PyTorch	Datos no tabulares, deep learning
DVC local	DVC + S3	Colaboración en equipo
MLflow local	MLflow server	Múltiples usuarios
Docker Compose	Kubernetes	>3 servicios, necesitas scaling
Prometheus local	Managed (Datadog)	Sin tiempo para mantener

Navegación

◀ Anterior	Índice	▶ Siguiente
18_GLOSARIO.md	Índice	20_PLAN_ESTUDIOS.md

© 2025 DuqueOM - Guía MLOps v3.0

Módulo 19 Completado