MÓDULO 15: OBSERVABILIDAD

Logging, Métricas, Tracing y Detección de Drift

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

MÓDULO 15: Observabilidad

Si No Lo Puedes Ver, No Lo Puedes Arreglar

"Un modelo en producción sin monitoreo es un avión sin instrumentos."

Duración	Teoría	Práctica
5-6 horas	30%	70%

Los 3 Pilares de Observabilidad

15.1 Logging Estructurado

```python
# app/core/logging.py
import structlog
import logging
import sys

def setup_logging(log_level: str = "INFO"):
    """Configura logging estructurado."""

    structlog.configure(
        processors=[
            structlog.contextvars.merge_contextvars,
            structlog.processors.add_log_level,
            structlog.processors.TimeStamper(fmt="iso"),
            structlog.processors.StackInfoRenderer(),
            structlog.processors.format_exc_info,
            structlog.processors.JSONRenderer()
        ],
        wrapper_class=structlog.make_filtering_bound_logger(
            getattr(logging, log_level)
        ),
        context_class=dict,
        logger_factory=structlog.PrintLoggerFactory(),
    )

logger = structlog.get_logger()

# Uso
logger.info(
    "prediction_completed",
    request_id="abc123",
    model_version="1.2.3",
    latency_ms=45.2,
    prediction="churn",
    probability=0.73,
)
# Output: {"event": "prediction_completed", "request_id": "abc123", ...}
```

15.2 Métricas con Prometheus

```python
# app/middleware/metrics.py
from prometheus_client import Counter, Histogram, Gauge, generate_latest
from fastapi import Request, Response
from starlette.middleware.base import BaseHTTPMiddleware
import time

# ═══════════════════════════════════════════════════════════
# DEFINIR MÉTRICAS
# ═══════════════════════════════════════════════════════════

# Contador de requests
REQUEST_COUNT = Counter(
    "http_requests_total",
    "Total HTTP requests",
    ["method", "endpoint", "status"]
)

# Histograma de latencia
REQUEST_LATENCY = Histogram(
    "http_request_duration_seconds",
    "HTTP request latency",
    ["method", "endpoint"],
    buckets=[0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1.0, 2.5, 5.0]
)

# Métricas ML-específicas
PREDICTION_COUNT = Counter(
    "ml_predictions_total",
    "Total predictions",
    ["model_version", "prediction"]
)

PREDICTION_PROBABILITY = Histogram(
    "ml_prediction_probability",
    "Distribution of prediction probabilities",
    ["model_version"],
    buckets=[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
)

MODEL_LOAD_TIME = Gauge(
    "ml_model_load_time_seconds",
    "Time to load the model"
)

# ═══════════════════════════════════════════════════════════
# MIDDLEWARE
# ═══════════════════════════════════════════════════════════

class MetricsMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request: Request, call_next):
        start_time = time.time()
        response = await call_next(request)
        duration = time.time() - start_time

        REQUEST_COUNT.labels(
            method=request.method,
            endpoint=request.url.path,
            status=response.status_code
        ).inc()

        REQUEST_LATENCY.labels(
            method=request.method,
            endpoint=request.url.path
        ).observe(duration)

        return response
```

Endpoint /metrics

```python
# app/api/routes/metrics.py
from fastapi import APIRouter, Response
from prometheus_client import generate_latest, CONTENT_TYPE_LATEST

router = APIRouter()

@router.get("/metrics")
async def metrics():
    return Response(
        generate_latest(),
        media_type=CONTENT_TYPE_LATEST
    )
```

15.3 Prometheus + Grafana

```yaml
# infra/prometheus-config.yaml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'bankchurn-api'
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: true
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
        action: replace
        target_label: __metrics_path__
        regex: (.+)
```

Dashboard Grafana (JSON)

```json
{
  "title": "BankChurn API Dashboard",
  "panels": [
    {
      "title": "Request Rate",
      "type": "graph",
      "targets": [
        {
          "expr": "rate(http_requests_total[5m])",
          "legendFormat": "{{method}} {{endpoint}}"
        }
      ]
    },
    {
      "title": "P99 Latency",
      "type": "stat",
      "targets": [
        {
          "expr": "histogram_quantile(0.99, rate(http_request_duration_seconds_bucket[5m]))"
        }
      ]
    },
    {
      "title": "Error Rate",
      "type": "gauge",
      "targets": [
        {
          "expr": "sum(rate(http_requests_total{status=~'5..'}[5m])) / sum(rate(http_requests_total[5m])) * 100"
        }
      ]
    },
    {
      "title": "Prediction Distribution",
      "type": "piechart",
      "targets": [
        {
          "expr": "sum by (prediction) (ml_predictions_total)"
        }
      ]
    }
  ]
}
```

15.4 Detección de Drift con Evidently

```python
# src/bankchurn/monitoring/drift.py
from evidently import ColumnMapping
from evidently.report import Report
from evidently.metric_preset import DataDriftPreset, TargetDriftPreset
from evidently.metrics import (
    DatasetDriftMetric,
    DataDriftTable,
    ColumnDriftMetric,
)
import pandas as pd
from pathlib import Path

class DriftMonitor:
    """Monitor de drift para datos y modelo."""

    def __init__(
        self,
        reference_data: pd.DataFrame,
        numerical_features: list[str],
        categorical_features: list[str],
        target: str = None,
    ):
        self.reference_data = reference_data
        self.column_mapping = ColumnMapping(
            numerical_features=numerical_features,
            categorical_features=categorical_features,
            target=target,
        )

    def check_data_drift(
        self,
        current_data: pd.DataFrame,
        drift_threshold: float = 0.05,
    ) -> dict:
        """Detecta data drift entre reference y current."""

        report = Report(metrics=[
            DatasetDriftMetric(),
            DataDriftTable(),
        ])

        report.run(
            reference_data=self.reference_data,
            current_data=current_data,
            column_mapping=self.column_mapping,
        )

        result = report.as_dict()

        return {
            "dataset_drift_detected": result["metrics"][0]["result"]["dataset_drift"],
            "drift_share": result["metrics"][0]["result"]["drift_share"],
            "drifted_columns": [
                col for col, data in result["metrics"][1]["result"]["drift_by_columns"].items()
                if data["drift_detected"]
            ],
            "threshold": drift_threshold,
        }

    def generate_report(
        self,
        current_data: pd.DataFrame,
        output_path: Path,
    ):
        """Genera reporte HTML de drift."""

        report = Report(metrics=[
            DataDriftPreset(),
        ])

        report.run(
            reference_data=self.reference_data,
            current_data=current_data,
            column_mapping=self.column_mapping,
        )

        report.save_html(str(output_path))


# ═══════════════════════════════════════════════════════════
# USO EN PRODUCCIÓN
# ═══════════════════════════════════════════════════════════

# Cargar datos de referencia (training data)
reference_df = pd.read_csv("data/processed/train.csv")

# Inicializar monitor
monitor = DriftMonitor(
    reference_data=reference_df,
    numerical_features=['CreditScore', 'Age', 'Balance'],
    categorical_features=['Geography', 'Gender'],
)

# Batch de predicciones del último día
current_df = get_last_24h_predictions()

# Detectar drift
drift_result = monitor.check_data_drift(current_df)

if drift_result["dataset_drift_detected"]:
    logger.warning(
        "data_drift_detected",
        drift_share=drift_result["drift_share"],
        drifted_columns=drift_result["drifted_columns"],
    )
    # Alertar y/o trigger retraining
```

15.5 Alertas

```yaml
# infra/prometheus-alerts.yaml
groups:
  - name: bankchurn-alerts
    rules:
      - alert: HighErrorRate
        expr: sum(rate(http_requests_total{status=~"5.."}[5m])) / sum(rate(http_requests_total[5m])) > 0.05
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "High error rate detected"
          description: "Error rate is above 5% for 5 minutes"

      - alert: HighLatency
        expr: histogram_quantile(0.99, rate(http_request_duration_seconds_bucket[5m])) > 0.5
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "High latency detected"
          description: "P99 latency is above 500ms"

      - alert: DataDriftDetected
        expr: ml_data_drift_detected == 1
        for: 1h
        labels:
          severity: warning
        annotations:
          summary: "Data drift detected"
          description: "Model inputs are drifting from training distribution"
```

15.6 Ejercicio: Dashboard Completo

Checklist

```
LOGGING:
[ ] Structured logging configurado
[ ] Request IDs en todos los logs
[ ] Error logging con stack traces

MÉTRICAS:
[ ] Prometheus métricas expuestas
[ ] Request count, latency, errors
[ ] ML-específicas (predictions, probabilities)

DASHBOARDS:
[ ] Grafana dashboard funcional
[ ] Paneles: Rate, Latency, Errors, ML

DRIFT:
[ ] Evidently configurado
[ ] Data drift detection
[ ] Alertas configuradas
```

Siguiente Paso

Con observabilidad implementada, es hora de **documentar y presentar** tu trabajo.

Ir a Módulo 16: Documentación y Ética →

Módulo 15 completado. Ahora puedes ver todo lo que pasa en producción.