Monitoreo y Observabilidad - Día 2

Change Status

pending · 40 min

Learning Objectives

1. Entender importancia del monitoreo en producción
2. Aprender métricas clave para pipelines
3. Comprender herramientas de observabilidad
4. Conocer estrategias de alerting efectivas

Practical exercise to apply the concepts learned.

Ejercicio: Configurar monitoreo completo para pipeline

Configurar métricas en DAG:

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.metrics import Stats
from datetime import datetime, timedelta
import logging

logger = logging.getLogger(__name__)

def extract_with_metrics(**context):
    """Extracción con métricas detalladas"""
    try:
        # Simular extracción
        records = 1000
        duration = 45.2  # segundos

        # Registrar métricas
        Stats.gauge('pipeline.extract.records', records)
        Stats.timer('pipeline.extract.duration', duration)

        # Log estructurado
        logger.info(f"Extract completed: {records} records in {duration}s",
                    extra={'records': records, 'duration': duration})

        return {'records': records, 'duration': duration}

    except Exception as e:
        Stats.incr('pipeline.extract.errors')
        logger.error(f"Extract failed: {e}")
        raise e

def transform_with_metrics(**context):
    """Transformación con tracking de calidad"""
    ti = context['task_instance']
    input_data = ti.xcom_pull(task_ids='extract')

    try:
        # Simular transformación
        output_records = input_data['records'] * 0.95  # 5% pérdida
        quality_score = 0.98

        # Métricas de calidad
        Stats.gauge('pipeline.transform.quality', quality_score)
        Stats.gauge('pipeline.transform.output_records', output_records)

        # Alertas basadas en calidad
        if quality_score < 0.9:
            logger.warning(f"Low quality score: {quality_score}")

        return {'output_records': output_records, 'quality': quality_score}

    except Exception as e:
        Stats.incr('pipeline.transform.errors')
        raise e

def load_with_metrics(**context):
    """Carga con métricas de performance"""
    ti = context['task_instance']
    transform_data = ti.xcom_pull(task_ids='transform')

    try:
        # Simular carga
        records_loaded = transform_data['output_records']
```

```
            load_time = 12.5

            Stats.gauge('pipeline.load.records_loaded', records_loaded)
            Stats.timer('pipeline.load.duration', load_time)

            # Calcular throughput
            throughput = records_loaded / load_time  # records/second
            Stats.gauge('pipeline.load.throughput', throughput)

            return {'loaded': records_loaded, 'throughput': throughput}

        except Exception as e:
            Stats.incr('pipeline.load.errors')
            raise e

    # Callbacks de monitoreo
    def dag_success_callback(context):
        """Callback cuando DAG completa exitosamente"""
        dag_run = context['dag_run']
        duration = (dag_run.end_date - dag_run.start_date).total_seconds()

        Stats.timer('pipeline.dag.duration', duration)
        Stats.incr('pipeline.dag.success')

        logger.info(f"DAG completed successfully in {duration}s")

    def dag_failure_callback(context):
        """Callback cuando DAG falla"""
        dag_run = context['dag_run']
        Stats.incr('pipeline.dag.failure')

        # Enviar alerta crítica
        logger.error(f"DAG failed: {context['dag'].dag_id}")

    def sla_miss_callback(dag, task_list, blocking_task_list, slas, blocking_tis):
        """Callback cuando se viola SLA"""
        Stats.incr('pipeline.sla.violations')
        logger.warning(f"SLA missed for DAG: {dag.dag_id}")

    # Configurar DAG con monitoreo completo
    dag = DAG(
        'pipeline_monitoring_demo',
        description='Pipeline con métricas y monitoreo completos',
        schedule_interval='@daily',
        start_date=datetime(2024, 1, 1),
        catchup=False,
        default_args={
            'retries': 2,
            'retry_delay': timedelta(minutes=5),
            'execution_timeout': timedelta(hours=2),
            'sla': timedelta(hours=1)  # SLA: 1 hora máximo
        },
        # Callbacks de monitoreo
        on_success_callback=dag_success_callback,
        on_failure_callback=dag_failure_callback,
        sla_miss_callback=sla_miss_callback,
        tags=['monitoring', 'production']
    )

    # Tareas del pipeline
    extract_task = PythonOperator(
        task_id='extract',
        python_callable=extract_with_metrics,
        provide_context=True,
        dag=dag
    )

    transform_task = PythonOperator(
        task_id='transform',
        python_callable=transform_with_metrics,
        provide_context=True,
        dag=dag
    )

    load_task = PythonOperator(
        task_id='load',
        python_callable=load_with_metrics,
        provide_context=True,
        dag=dag
    )

    # Dependencias
    extract_task >> transform_task >> load_task
```

Configurar dashboard en Grafana:

```
// Panel de métricas principales
{
  "title": "Pipeline Health Dashboard",
  "panels": [
    {
      "title": "DAG Success Rate",
      "type": "stat",
```

```
      "targets": [{
        "expr": "rate(pipeline_dag_success[1h]) / (rate(pipeline_dag_success[1h]) + rate(pipeline_dag_failure[1h])) * 100",
        "legendFormat": "Success Rate %"
      }]
    },
    {
      "title": "Average Processing Time",
      "type": "graph",
      "targets": [{
        "expr": "rate(pipeline_extract_duration_sum[5m]) / rate(pipeline_extract_duration_count[5m])",
        "legendFormat": "Extract Time (s)"
      }]
    },
    {
      "title": "Error Rate by Component",
      "type": "bargauge",
      "targets": [
        {"expr": "rate(pipeline_extract_errors[1h])", "legendFormat": "Extract"},
        {"expr": "rate(pipeline_transform_errors[1h])", "legendFormat": "Transform"},
        {"expr": "rate(pipeline_load_errors[1h])", "legendFormat": "Load"}
      ]
    }
  ]
}
```

Configurar alertas:

```
# Alert rules para Prometheus
groups:
- name: pipeline_alerts
  rules:

  - alert: PipelineHighErrorRate
    expr: rate(pipeline_dag_failure[5m]) / (rate(pipeline_dag_success[5m]) + rate(pipeline_dag_failure[5m])) > 0.1
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "High error rate in pipeline"
      description: "Pipeline error rate > 10% for 5 minutes"

  - alert: PipelineSLAViolation
    expr: increase(pipeline_sla_violations[1h]) > 0
    labels:
      severity: critical
    annotations:
      summary: "SLA violation detected"
      description: "Pipeline failed to meet SLA requirements"

  - alert: PipelineDataFreshness
    expr: pipeline_data_freshness_hours > 24
    labels:
      severity: warning
    annotations:
      summary: "Data freshness issue"
      description: "Pipeline data is more than 24 hours old"
```

Verificación: ¿Qué métricas son más importantes para monitorear en un pipeline de datos vs una aplicación web? ¿Cómo decidir cuándo escalar de warning a critical en las alertas?

Requerimientos:

Apache Airflow con métricas habilitadas
Prometheus/Grafana para monitoreo
Configuración de logging estructurado