Estrategias de Despliegue (CI/CD) - Día 1

Change Status

in-progress 40 min

Learning Objectives

1. Entender conceptos de CI/CD para pipelines de datos
2. Aprender estrategias de despliegue automatizado
3. Comprender testing automatizado de DAGs
4. Conocer mejores prácticas de deployment

Activities and Learning

Task 1: Conceptos de CI/CD para Datos (10 minutos)

¿Por qué CI/CD en pipelines de datos?

CI/CD automatiza el proceso desde desarrollo hasta producción, asegurando calidad y velocidad.

Beneficios principales:

Rapidez: Deployments automáticos y frecuentes
Calidad: Tests automáticos previenen bugs
Consistencia: Entornos idénticos en dev/test/prod
Confianza: Rollbacks automáticos si hay problemas

Pipeline CI/CD típico:

```
Código → Tests → Build → Deploy Dev → Tests Integración → Deploy Prod
```

Task 2: Testing Automatizado de DAGs (10 minutos)

Tipos de tests para DAGs:

1. Tests de sintaxis:

```python
from airflow.models import DagBag

def test_dag_syntax():
    """Verificar que DAGs se cargan sin errores"""
    dagbag = DagBag(dag_folder='dags/', include_examples=False)

    assert len(dagbag.import_errors) == 0, f"Errores de importación: {dagbag.import_errors}"

    for dag_id, dag in dagbag.dags.items():
        assert dag is not None, f"DAG {dag_id} no se cargó"
```

2. Tests de estructura:

```python
def test_dag_structure():
    """Verificar estructura del DAG"""
    dagbag = DagBag()
    dag = dagbag.get_dag('mi_pipeline')

    # Verificar tareas críticas existen
    assert 'extract_data' in [t.task_id for t in dag.tasks]
    assert 'load_data' in [t.task_id for t in dag.tasks]

    # Verificar dependencias
    extract_task = dag.get_task('extract_data')
    load_task = dag.get_task('load_data')
    assert load_task in extract_task.downstream_list
```

3. Tests funcionales:

```python
def test_dag_execution():
    """Test ejecución completa del DAG"""
    dag = create_test_dag()

    # Ejecutar en fecha específica
```

```
    execution_date = datetime(2024, 1, 1)
    dag.run(start_date=execution_date, end_date=execution_date)

    # Verificar resultados
    # (Requiere configuración de test database)
```

Task 3: Estrategias de Despliegue (10 minutos)

Patrones de deployment:

1. Blue-Green Deployment:

Dos entornos idénticos (azul/verde)
Traffic se cambia de uno a otro
Rollback instantáneo cambiando traffic

2. Canary Deployment:

Deploy a porcentaje pequeño de usuarios
Monitorear métricas y errores
Gradualmente aumentar porcentaje

3. Rolling Deployment:

Deploy por componentes gradualmente
Mantiene sistema parcialmente operativo
Ideal para sistemas distribuidos

Configuración para Airflow:

```
# docker-compose.prod.yml
version: '3.8'
services:
  airflow-webserver:
    image: ${AIRFLOW_IMAGE}
    environment:
      - AIRFLOW__CORE__EXECUTOR=CeleryExecutor
      - AIRFLOW__CORE__LOAD_EXAMPLES=False
    volumes:
      - ./dags:/opt/airflow/dags
      - ./logs:/opt/airflow/logs

  airflow-scheduler:
    image: ${AIRFLOW_IMAGE}
    command: scheduler
    volumes:
      - ./dags:/opt/airflow/dags

  airflow-worker:
    image: ${AIRFLOW_IMAGE}
    command: celery worker
    scale: 3  # Múltiples workers
```