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

Theory | Practice | Evidence | Quiz

Practical exercise to apply the concepts learned.

Ejercicio: Configurar CI/CD básico para DAGs

Configurar GitHub Actions:

```yaml
# .github/workflows/ci-cd-airflow.yml
name: Airflow CI/CD Pipeline

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

env:
  AIRFLOW_VERSION: 2.7.0
  PYTHON_VERSION: '3.9'

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: ${{ env.PYTHON_VERSION }}

    - name: Install dependencies
      run: |
        pip install apache-airflow==${{ env.AIRFLOW_VERSION }}
        pip install pytest pytest-cov

    - name: Test DAG syntax
      run: |
        python -c "
        from airflow.models import DagBag
        dagbag = DagBag(include_examples=False)
        if dagbag.import_errors:
            print('DAG ERRORS:', dagbag.import_errors)
            exit(1)
        print(f'✅ {len(dagbag.dags)} DAGs cargados correctamente')
        "

    - name: Run DAG tests
      run: |
        pytest tests/dags/ -v --cov=dags --cov-report=xml

    - name: Upload coverage
      uses: codecov/codecov-action@v3
      with:
        file: ./coverage.xml

  deploy-dev:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/develop'

    steps:
    - name: Deploy to Dev
      run: |
        echo "🚀 Deploying to development environment"
        # Aquí iría lógica de deployment (kubectl, docker-compose, etc.)

  deploy-prod:
```

```
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    environment: production

    steps:
    - name: Deploy to Production
      run: |
        echo "🎯 Deploying to production environment"
        # Deployment production con verificaciones adicionales
```

Crear tests para DAGs:

```
# tests/dags/test_etl_pipeline.py
import pytest
from airflow.models import DagBag
from datetime import datetime, timedelta

class TestETLPipeline:

    @pytest.fixture(scope='class')
    def dagbag(self):
        """Cargar DAGs una vez por clase de tests"""
        return DagBag(include_examples=False)

    def test_dag_loaded(self, dagbag):
        """Verificar que el DAG principal se carga"""
        dag = dagbag.get_dag('etl_pipeline')
        assert dag is not None, "DAG etl_pipeline no encontrado"
        assert dag.description is not None, "DAG sin descripción"

    def test_dag_structure(self, dagbag):
        """Verificar estructura del DAG"""
        dag = dagbag.get_dag('etl_pipeline')

        # Verificar número mínimo de tareas
        assert len(dag.tasks) >= 5, f"DAG tiene solo {len(dag.tasks)} tareas"

        # Verificar tareas críticas
        task_ids = [t.task_id for t in dag.tasks]
        assert 'extract' in task_ids, "Tarea extract faltante"
        assert 'transform' in task_ids, "Tarea transform faltante"
        assert 'load' in task_ids, "Tarea load faltante"

    def test_dag_dependencies(self, dagbag):
        """Verificar dependencias entre tareas"""
        dag = dagbag.get_dag('etl_pipeline')

        extract = dag.get_task('extract')
        transform = dag.get_task('transform')
        load = dag.get_task('load')

        # Verificar que transform depende de extract
        assert transform in extract.downstream_list, "Transform no depende de extract"

        # Verificar que load depende de transform
        assert load in transform.downstream_list, "Load no depende de transform"

    def test_dag_schedule(self, dagbag):
        """Verificar configuración de scheduling"""
        dag = dagbag.get_dag('etl_pipeline')

        # Verificar que tiene schedule definido
        assert dag.schedule_interval is not None, "DAG sin schedule definido"

        # Verificar start_date
        assert dag.start_date is not None, "DAG sin start_date"
        assert dag.start_date < datetime.now(), "start_date en el futuro"

    def test_task_configuration(self, dagbag):
        """Verificar configuración de tareas individuales"""
        dag = dagbag.get_dag('etl_pipeline')

        for task in dag.tasks:
            # Verificar que todas las tareas tienen retries
            assert task.retries >= 1, f"Tarea {task.task_id} sin retries"

            # Verificar timeout razonable
            if hasattr(task, 'execution_timeout') and task.execution_timeout:
                assert task.execution_timeout <= timedelta(hours=2), \
                       f"Timeout excesivo en {task.task_id}"

# tests/dags/test_dag_integration.py
class TestDAGIntegration:

    def test_dag_runs_without_errors(self, dagbag):
        """Test que DAG puede ejecutarse sin errores de sintaxis"""
        dag = dagbag.get_dag('etl_pipeline')

        # Verificar que no hay ciclos
        assert dag.test_cycle() == True, "DAG tiene ciclos de dependencias"

        # Verificar que todas las dependencias son válidas
```

```
        for task in dag.tasks:
            for upstream in task.upstream_list:
                assert upstream in dag.tasks, f"Dependencia inválida: {upstream.task_id}"
```

Script de deployment:

```
# scripts/deploy.sh
#!/bin/bash

set -e  # Salir si hay error

ENVIRONMENT=$1
if [ -z "$ENVIRONMENT" ]; then
    echo "Uso: $0 <environment>"
    echo "Environment: dev, staging, prod"
    exit 1
fi

echo "🚀 Deploying Airflow to $ENVIRONMENT environment"

# Verificar tests pasaron
echo "📋 Running tests..."
python -m pytest tests/dags/ -v

# Build imagen si es necesario
if [ "$ENVIRONMENT" = "prod" ]; then
    echo "🏗️  Building production image..."
    docker build -t airflow-prod:$GITHUB_SHA .
fi

# Deploy según environment
case $ENVIRONMENT in
    dev)
        echo "🔧 Deploying to development..."
        docker-compose -f docker-compose.dev.yml up -d
        ;;
    staging)
        echo "🧪 Deploying to staging..."
        kubectl apply -f k8s/staging/
        ;;
    prod)
        echo "🎯 Deploying to production..."
        kubectl apply -f k8s/prod/
        # Health checks
        sleep 30
        curl -f http://airflow-prod/health || exit 1
        ;;
esac

echo "✅ Deployment to $ENVIRONMENT completed successfully"
```

Verificación: ¿Qué diferencias hay entre CI/CD para aplicaciones web vs pipelines de datos? ¿Cómo asegurar que los tests de DAGs sean rápidos y confiables?

Requerimientos:

Apache Airflow configurado
GitHub Actions o similar CI/CD
Conocimiento de testing con pytest