SQL-Databricks Bridge

Sincronización bidireccional de datos entre SQL Server y Databricks

Kantar Worldpanel - Data Engineering

Agenda

- Problema y solución
- Arquitectura del sistema
- Flujos de datos
- CLI y API
- **SDKs: API Local y Databricks Jobs**
- Configuración
- Seguridad y permisos
- Demo

El Problema

Desafíos de Integración

- **Datos dispersos**: SQL Server on-premise + Databricks en la nube
- **Sincronización manual**: Scripts ad-hoc, propensos a errores
- **Sin trazabilidad**: No hay auditoría de qué datos se movieron
- **Permisos inconsistentes**: Cada proyecto maneja su propia autenticación

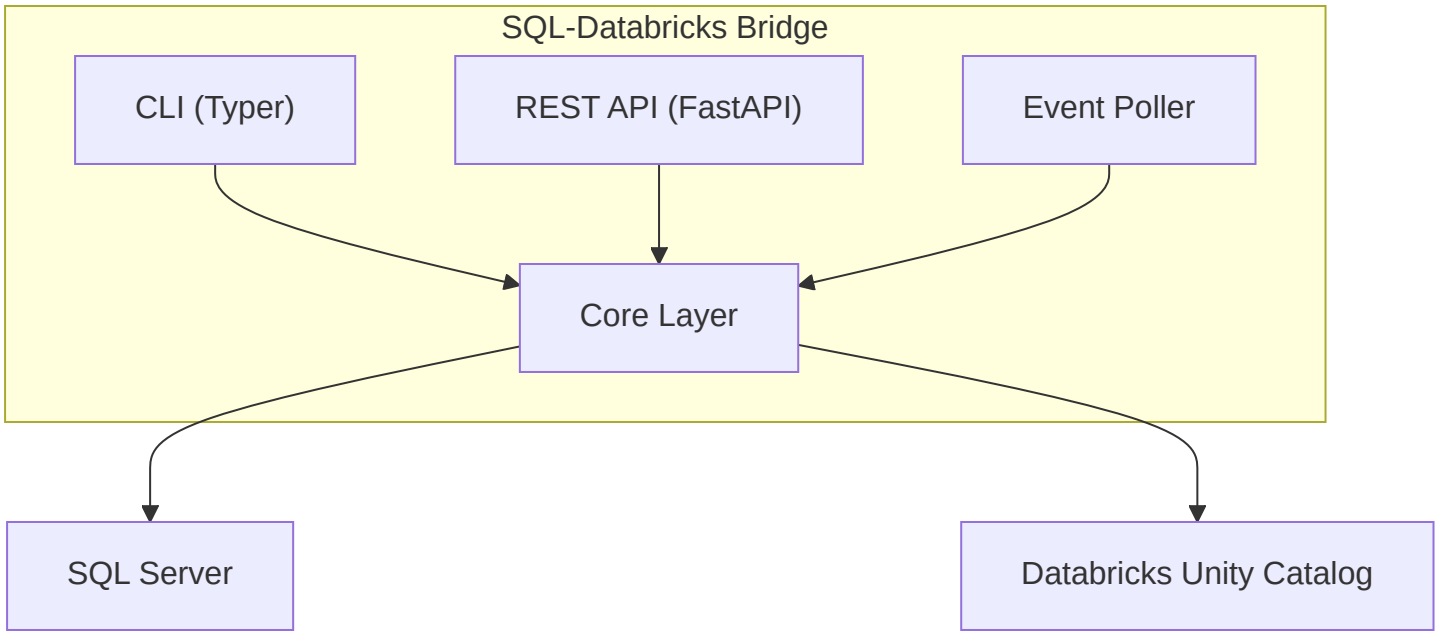
La Solución

SQL-Databricks Bridge

- **Extracción**: SQL Server → Databricks (Parquet en Volumes)
- **Sincronización**: Databricks → SQL Server (INSERT/UPDATE/DELETE)
- **CLI + API**: Flexibilidad para automatización y uso interactivo
- **Seguridad**: Tokens por proyecto, permisos por tabla, auditoría

Arquitectura

Vista General del Sistema



Estructura de Módulos

```
src/sql_databricks_bridge/
├── api/           # REST API (FastAPI)
│   ├── routes/    # Endpoints
│   └── schemas.py # Modelos Pydantic
├── cli/           # Comandos CLI (Typer)
├── core/          # Lógica de negocio
│   ├── extractor.py
│   └── uploader.py
├── sync/          # Sincronización inversa
│   ├── poller.py
│   └── operations.py
├── auth/          # Autenticación
└── db/            # Clientes de BD
```

Flujo de Extracción

SQL Server → Databricks

Queries Parametrizadas

```sql
-- queries/extract_ventas.sql
SELECT
    id_transaccion,
    fecha,
    {factor} as factor_expansion,
    monto_total
FROM {database}.{schema}.{sales_table}
WHERE pais = '{country_code}'
  AND fecha ⩾ '{start_date}'
```

```yaml
# config/common_params.yaml
factor: factor_rw1 as factor_rw
start_date: "2024-01-01"

# config/Colombia.yaml
country_code: CO
database: KWP_Colombia
sales_table: J_Ventas_CO
```

Flujo de Sincronización

Databricks → SQL Server

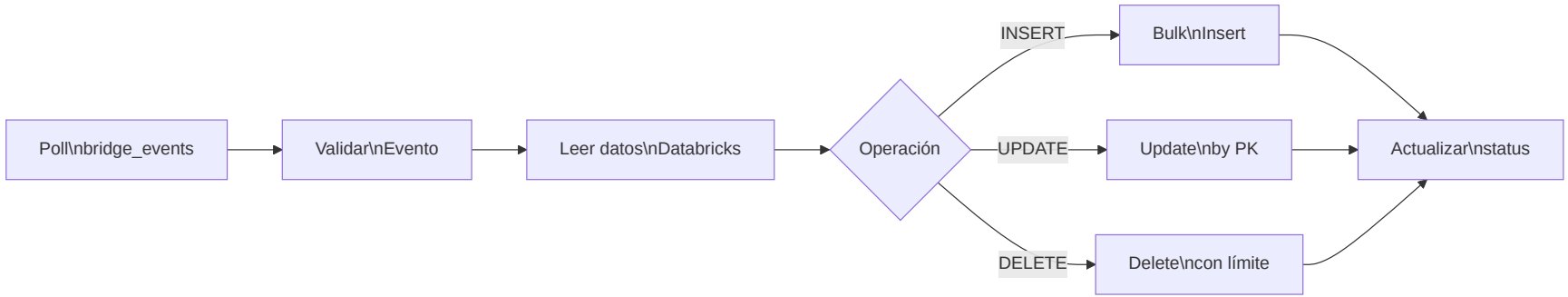


Tabla de Eventos

```
INSERT INTO bridge.events.bridge_events (
  event_id,
  operation,
  source_table,
  target_table,
  primary_keys
) VALUES (
  uuid(),
  'INSERT',
  'catalog.schema.source',
  'dbo.target',
  array('id')
);
```

El poller procesa eventos cada **10 segundos**

CLI

Comandos Disponibles

Comando	Descripción
`extract`	Extraer datos de SQL Server a Databricks
`list-queries`	Listar archivos SQL disponibles
`show-params`	Mostrar parámetros resueltos
`test-connection`	Probar conectividad
`serve`	Iniciar servidor API

Ejemplo de Extracción

```
sql-databricks-bridge extract \
  --queries-path ./queries \
  --config-path ./config \
  --country Colombia \
  --destination /Volumes/catalog/schema/volume
```

```
# Extraer queries específicos
sql-databricks-bridge extract \
  --queries-path ./queries \
  --config-path ./config \
  --country Mexico \
  --query ventas \
  --query productos \
  --overwrite
```

REST API

Endpoints Principales

Método	Endpoint	Descripción
POST	`/extract`	Iniciar extracción
GET	`/jobs/{id}`	Estado del job
GET	`/jobs`	Listar jobs
DELETE	`/jobs/{id}`	Cancelar job
GET	`/health/live`	Liveness probe
GET	`/health/ready`	Readiness probe

Ejemplo de API

```
# Iniciar extracción
curl -X POST http://localhost:8000/extract \
  -H "Authorization: Bearer your-token" \
  -H "Content-Type: application/json" \
  -d '{
    "queries_path": "./queries",
    "config_path": "./config",
    "country": "Colombia",
    "destination": "/Volumes/catalog/schema/vol"
  }'
```

```
# Verificar estado
curl http://localhost:8000/jobs/{job_id} \
  -H "Authorization: Bearer your-token"
```

SDKs

Modos de Uso del SDK

Modo	Descripción	Caso de Uso
API Local	Cliente Python para consumir la REST API	Apps locales, servicios externos
Databricks Jobs	Uso directo de la librería en notebooks/jobs	Pipelines en Databricks

```
sql-databricks-bridge
├── sdk/              # SDKs para consumidores
│   ├── client.py     # Cliente REST API
│   └── databricks.py # Helpers para jobs
└── ...
```

SDK: Cliente API Local

Para aplicaciones que consumen la API REST desde Python:

```python
from sql_databricks_bridge.sdk import BridgeClient

# Inicializar cliente
client = BridgeClient(
    base_url="http://localhost:8000",
    token="your-api-token"
)

# Extraer datos de SQL Server a Databricks
job = client.extract(
    queries_path="./queries",
    config_path="./config",
    country="Colombia",
    destination="/Volumes/catalog/schema/volume"
)

# Verificar estado
status = client.get_job_status(job.job_id)
print(f"Estado: {status.state}")
```

SDK: Cliente API - Sync Events

```python
from sql_databricks_bridge.sdk import BridgeClient

client = BridgeClient(base_url="http://localhost:8000", token="token")

# Enviar evento de sincronización (Databricks → SQL Server)
event = client.submit_sync_event(
    operation="INSERT",
    source_table="catalog.schema.source_table",
    target_table="dbo.target_table",
    primary_keys=["id"]
)

# Monitorear estado del evento
status = client.get_sync_event(event.event_id)
print(f"Estado: {status.status}, Filas: {status.rows_affected}")
```

```python
# Listar eventos con filtros
events = client.list_sync_events(status="failed", limit=10)
for e in events:
    print(f"{e.event_id}: {e.error_message}")
```

SDK: Databricks Jobs

Para scripts que corren directamente en Databricks Jobs/Notebooks:

```python
# En un notebook de Databricks
from sql_databricks_bridge.sdk.databricks import BridgeOperator

# Inicializar operador (usa credenciales del job context)
operator = BridgeOperator(
    sql_server_host="server.database.windows.net",
    sql_server_database="KWP_Colombia",
    sql_server_user=dbutils.secrets.get("scope", "sql_user"),
    sql_server_password=dbutils.secrets.get("scope", "sql_pass")
)

# Sincronizar datos desde tabla Databricks a SQL Server
result = operator.sync_to_sql_server(
    source_table="catalog.schema.calibrated_panel",
    target_table="dbo.CalibrationResults",
    operation="INSERT",
    primary_keys=["id_hogar", "periodo"]
)

print(f"Filas sincronizadas: {result.rows_affected}")
```

SDK: Databricks Jobs - Extracción

```python
from sql_databricks_bridge.sdk.databricks import BridgeOperator

operator = BridgeOperator(
    sql_server_host="server.database.windows.net",
    sql_server_database="KWP_Colombia",
    sql_server_user=dbutils.secrets.get("scope", "sql_user"),
    sql_server_password=dbutils.secrets.get("scope", "sql_pass")
)

# Extraer de SQL Server a DataFrame de Spark
df = operator.extract_to_spark(
    query="""
        SELECT id_transaccion, fecha, monto_total
        FROM dbo.Ventas
        WHERE fecha >= '2024-01-01'
    """
)

# Guardar como tabla Delta
df.write.format("delta").saveAsTable("catalog.schema.ventas_extract")
```

SDK: Instalación

```
# Desde PyPI (producción)
pip install sql-databricks-bridge

# Para desarrollo local
pip install -e ".[dev]"

# En Databricks (requirements.txt del job)
sql-databricks-bridge>=1.0.0
```

Configuración en Databricks Job:

```
# databricks.yml (asset bundle)
resources:
  jobs:
    sync_job:
      tasks:
        - task_key: sync_to_sql
          python_wheel_task:
            package_name: sql_databricks_bridge
          libraries:
            - pypi:
                package: sql-databricks-bridge
```

SDK: Comparación

API Local (BridgeClient)

- Consume REST API
- Requiere servidor corriendo
- Para apps externas
- Autenticación por token

```
client = BridgeClient(
    base_url="http://...",
    token="..."
)
client.extract(...)
```

Databricks Jobs (BridgeOperator)

- Uso directo de la librería
- No requiere servidor
- Para notebooks/jobs
- Credentials de SQL Server

```
operator = BridgeOperator(
    sql_server_host="...",
    sql_server_password="..."
)
operator.sync_to_sql_server(...)
```

Seguridad

Autenticación

```yaml
# config/permissions.yaml
users:
  - token: "proyecto-abc-token"
    name: "proyecto-abc"
    permissions:
      - table: "dbo.Ventas"
        access: "read_write"
        max_delete_rows: 10000
```

Niveles de Acceso

Nivel	Read	Write	Delete
`read`	✓	✗	✗
`write`	✗	✓	✓*
`read_write`	✓	✓	✓*

* DELETE limitado por `max_delete_rows`

Auditoría

Todos los eventos de seguridad se registran:

```
{
  "timestamp": "2024-01-15T10:30:00Z",
  "event_type": "auth_success",
  "user_name": "proyecto-abc",
  "source_ip": "10.0.0.1",
  "resource": "dbo.Ventas",
  "action": "read"
}
```

Configuración

Variables de Entorno

```
# SQL Server
SQLSERVER_HOST=your-server.database.windows.net
SQLSERVER_DATABASE=your_database
SQLSERVER_USERNAME=your_user
SQLSERVER_PASSWORD=your_password

# Databricks (solo conexión, catálogo va en --destination)
DATABRICKS_HOST=https://workspace.azuredatabricks.net
DATABRICKS_TOKEN=your_token
```

> El catálogo se especifica en el path de destino: `--destination /Volumes/{catalog}/{schema}/{volume}`

Retry y Tolerancia a Fallos

- **Máximo intentos**: 3
- **Backoff**: Exponencial con jitter
- **Delay base**: 1 segundo
- **Delay máximo**: 60 segundos
- **Discrepancias**: Se reportan pero no bloquean

Deployment

Compilación con Nuitka

```
# Compilar ejecutable standalone para Windows
nuitka --standalone --onefile `
  --include-data-dir=config=config `
  --output-filename=sql-databricks-bridge.exe `
  src/sql_databricks_bridge/cli/commands.py

# Resultado: ejecutable único ~50MB
# No requiere Python instalado en servidor destino
```

```
# Verificar ejecutable
.\sql-databricks-bridge.exe --version
.\sql-databricks-bridge.exe test-connection
```

Arquitectura de Deployment

Servicio Windows (NSSM)

```
# Descargar NSSM desde nssm.cc
# Instalar como servicio Windows

nssm install SQLDatabricksBridge `
  C:\Bridge\sql-databricks-bridge.exe serve

nssm set SQLDatabricksBridge AppDirectory C:\Bridge
nssm set SQLDatabricksBridge AppEnvironmentExtra `
  "SQLSERVER_HOST=server.database.windows.net" `
  "DATABRICKS_HOST=https://workspace.azuredatabricks.net"
```

NSSM reinicia automáticamente el servicio si falla

Gestión del Servicio

```
# Iniciar servicio
nssm start SQLDatabricksBridge

# Detener servicio
nssm stop SQLDatabricksBridge

# Ver estado
nssm status SQLDatabricksBridge
```

```
# También disponible en Services (services.msc)
# O con PowerShell nativo:
Get-Service SQLDatabricksBridge
Start-Service SQLDatabricksBridge
```

Resumen

Beneficios Clave

- **Bidireccional**: SQL ↔ Databricks en ambas direcciones
- **Flexible**: CLI para scripts, API para automatización
- **Seguro**: Tokens, permisos por tabla, auditoría
- **Robusto**: Retry automático, tolerancia a fallos
- **Observable**: Health checks, logging estructurado

Gracias

Documentación · Azure DevOps

¿Preguntas?