Medallion Architecture ETL Pipeline

Complete PySpark + Delta Lake + Prefect + PostgreSQL Solution

A production-ready, enterprise-grade ETL pipeline implementing the Medallion Architecture (Bronze-Silver-Gold) with comprehensive features including incremental processing, schema evolution, SCD Type 2, data quality checks, and full orchestration.

🏗️ Architecture Overview

```
┌─────────────────────────────────────────────────────────────┐
│                    INPUT LAYER                              │
│  Nested Excel Files (Sales/Invoices/Credit Notes)           │
│  sales/cash_invoices/detailed/2025/january/*.xlsx           │
└────────────────────────┬────────────────────────────────────┘
                         │
                         ▼
┌─────────────────────────────────────────────────────────────┐
│              BRONZE LAYER (Raw Data)                        │
│  ✓ Direct Excel → Delta Lake ingestion                      │
│  ✓ Schema evolution handling                                │
│  ✓ Incremental processing (hash-based)                      │
│  ✓ Full audit trail & metadata                              │
└────────────────────────┬────────────────────────────────────┘
                         │
                         ▼
┌─────────────────────────────────────────────────────────────┐
│              SILVER LAYER (Cleaned Data)                    │
│  ✓ Data cleaning & transformation                           │
│  ✓ SCD Type 2 (historical tracking)                         │
│  ✓ Data quality checks                                      │
│  ✓ Credit note separation                                   │
└────────────────────────┬────────────────────────────────────┘
                         │
                         ▼
┌─────────────────────────────────────────────────────────────┐
│              GOLD LAYER (Business Metrics)                  │
│  ✓ Daily/Monthly sales summaries                            │
│  ✓ Customer analytics & segmentation                        │
│  ✓ Product performance metrics                              │
│  ✓ Business-ready aggregations                              │
```

```
└────────────────────────────────────────────────┘
         │
         ▼
┌────────────────────────────────────────────────┐
│        METADATA & MONITORING (PostgreSQL)      │
│ ✓ Pipeline execution logs                      │
│ ✓ Data quality tracking                        │
│ ✓ Schema evolution history                     │
│ ✓ SCD2 operation logs                          │
└────────────────────────────────────────────────┘
```

🚀 Key Features

✅ Medallion Architecture

- **Bronze Layer**: Raw data ingestion from nested Excel files
- **Silver Layer**: Cleaned, transformed, and quality-checked data
- **Gold Layer**: Business-ready aggregations and metrics

✅ Advanced Data Management

- **Incremental Processing**: Hash-based file change detection
- **Schema Evolution**: Automatic handling of schema changes
- **SCD Type 2**: Full historical tracking with validity periods
- **Data Partitioning**: Year/Month based partitioning for efficiency

✅ Data Quality Framework

- Null value detection and flagging
- Duplicate detection (non-destructive)
- Amount validation checks
- Comprehensive quality logging

✅ Production Features

- **Prefect Orchestration**: Workflow management and scheduling
- **PostgreSQL Metadata**: Complete audit trail and lineage
- **Error Handling**: Robust retry logic and error tracking

- **Monitoring**: Real-time pipeline and data quality monitoring

---

📁 Project Structure

```
medallion-etl-pipeline/
├── config.py              # Configuration management
├── metadata_logger.py         # PostgreSQL logging
├── spark_utils.py          # Spark utilities & data quality
├── bronze_processor.py         # Bronze layer processing
├── silver_processor.py         # Silver layer with SCD2
├── main_pipeline.py          # Gold layer & Prefect orchestration
├── run_pipeline.py          # Main entry point
├── deploy_to_prefect.py        # Prefect deployment
├── monitor_pipeline.py         # Pipeline monitoring
├── requirements.txt          # Python dependencies
├── docker-compose.yml          # PostgreSQL & Prefect setup
├── init_db.sql            # Database initialization
├── .env.example           # Environment variables template
└── README.md              # This file
```

🛠 Installation & Setup

Prerequisites

- Python 3.9+
- Docker & Docker Compose (for PostgreSQL & Prefect)
- Java 8 or 11 (for PySpark)

1. Clone Repository

```bash
git clone <repository-url>
cd medallion-etl-pipeline
```

2. Setup Environment

Windows:

```bash
setup_environment.bat
```

Linux/Mac:

```bash
chmod +x setup_environment.sh
./setup_environment.sh
source venv/bin/activate
```

3. Configure Environment Variables

```bash
cp .env.example .env
# Edit .env with your configuration
```

4. Start Infrastructure

```bash
# Start PostgreSQL and Prefect Server
docker-compose up -d

# Verify services are running
docker-compose ps
```

5. Install Dependencies

```bash
pip install -r requirements.txt
```

🎯 Quick Start

Run Complete Pipeline

```bash
python run_pipeline.py
```

Run Specific Features

```bash
python run_pipeline.py --features sales_cash_invoices_detailed sales_credit_notes_detailed
```

Run Specific Layer

```bash
python run_pipeline.py --bronze-only
python run_pipeline.py --silver-only
python run_pipeline.py --gold-only
```

📊 Pipeline Configuration

Configure Features (config.py)

```python
features: Dict[str, FeatureConfig] = {
    "sales_cash_invoices_detailed": FeatureConfig(
        name="sales_cash_invoices_detailed",
        source_patterns=["sales/cash_invoices/detailed/**/*.xls*"],
        partition_columns=["transaction_year", "transaction_month"],
        scd2_enabled=True,
        data_quality_checks=["null_check", "duplicate_check", "amount_check"]
    )
}
```

Configure Database (config.py or .env)

```python
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
POSTGRES_DB=etl_metadata
POSTGRES_USER=etl_user
POSTGRES_PASSWORD=etl_password
```

Configure Spark (config.py)

```python
spark_configs = {
    "spark.driver.memory": "4g",
    "spark.executor.memory": "4g",
    "spark.sql.shuffle.partitions": "200"
}
```

🔄 Prefect Orchestration

Deploy to Prefect

```bash
python deploy_to_prefect.py
```

Access Prefect UI

```
http://localhost:4200
```

Run Deployed Pipeline

```bash
prefect deployment run 'Medallion Architecture ETL Pipeline/medallion-etl-production'
```

Schedule Configuration

- Default: Daily at 2 AM UTC
- Modify in `deploy_to_prefect.py`

📈 Monitoring & Metrics

View Pipeline Execution Summary

```bash
python monitor_pipeline.py
```

Query Metadata Database

```sql
-- Recent pipeline runs
SELECT * FROM etl_logs.pipeline_runs
ORDER BY started_at DESC LIMIT 10;

-- Data quality issues
SELECT * FROM etl_logs.data_quality_checks
WHERE check_status = 'FAILED';

-- Schema evolution history
SELECT * FROM etl_logs.schema_evolution
ORDER BY change_date DESC;

-- SCD2 operations
SELECT * FROM etl_logs.scd2_history
ORDER BY operation_timestamp DESC;
```

Access Prefect Monitoring

- **UI**: http://localhost:4200
- **Flow Runs**: Monitor execution status
- **Logs**: View detailed execution logs
- **Metrics**: Track performance over time

🗂️ Data Organization

Input Structure

```
C:/etl/data/input/
├── sales/
│   ├── cash_invoices/
│   │   ├── detailed/
│   │   │   └── 2025/
│   │   │       └── january/
│   │   │           └── invoices_jan_2025.xlsx
│   │   └── summarized/
│   │       └── 2025/
```

```
│   │         └── january/
│   ├── cash_sales/
│   │   ├── detailed/
│   │   └── summarized/
│   └── credit_notes/
│       ├── detailed/
│       └── summarized/
```

Output Structure (Delta Lake)

```
C:/etl/data/lakehouse/
├── bronze/
│   ├── bronze_sales_cash_invoices_detailed_sheet1/
│   ├── bronze_sales_cash_invoices_summarized_sheet1/
│   └── ...
├── silver/
│   ├── silver_sales_cash_invoices_detailed_invoices/
│   ├── silver_sales_cash_invoices_detailed_credits/
│   └── ...
└── gold/
    ├── daily_sales_summary/
    ├── monthly_sales_summary/
    ├── customer_analytics/
    └── product_analytics/
```

🔍 Data Quality Framework

Implemented Checks

1. **Null Checks**: Identify missing critical values
2. **Duplicate Detection**: Flag potential duplicates
3. **Amount Validation**: Verify numeric ranges
4. **Credit Note Validation**: Ensure proper sign

Quality Flags in Data

```python
```

```
# Automatically added to dataframes
dq_null_flag_customer_code
dq_null_flag_amount
dq_duplicate_flag
```

Quality Reporting

All quality issues logged to PostgreSQL:

```sql
SELECT
    table_name,
    check_type,
    check_status,
    records_affected,
    check_details
FROM etl_logs.data_quality_checks
WHERE check_status IN ('FAILED', 'WARNING');
```

📚 SCD Type 2 Implementation

Features

- Automatic change detection via hash comparison
- Historical record preservation
- Current record flagging
- Version tracking

SCD2 Columns

```python
_scd2_valid_from      # Timestamp when record became active
_scd2_valid_to        # Timestamp when record expired (NULL if current)
_scd2_is_current      # Boolean flag for current record
_scd2_version         # Version number
_business_key_hash    # Hash of business keys
_value_hash           # Hash of all values for change detection
```

Query Current Records

```python
df = spark.read.format("delta").load(table_path)
current_df = df.filter(F.col("_scd2_is_current") == True)
```

Query Historical Records

```python
# Get all versions of a customer
customer_history = df.filter(
    (F.col("customer_code") == "CUST001") &
    (F.col("_scd2_is_current") == False)
).orderBy("_scd2_valid_from")
```

🔧 Advanced Configuration

Enable/Disable Features

```python
# In config.py
enable_schema_evolution: bool = True
enable_data_quality_checks: bool = True
enable_scd2: bool = True
```

Tune Performance

```python
# Spark configurations
spark_configs = {
    "spark.sql.shuffle.partitions": "200",  # Adjust based on data size
    "spark.sql.adaptive.enabled": "true",
    "spark.sql.adaptive.coalescePartitions.enabled": "true"
}
```

Customize Partitioning

```python
```

```
partition_columns: List[str] = ["transaction_year", "transaction_month"]
```

🐛 Troubleshooting

Common Issues

1. Spark Memory Errors

```python
# Increase memory in config.py
driver_memory: str = "8g"
executor_memory: str = "8g"
```

2. PostgreSQL Connection Failed

```bash
# Check if PostgreSQL is running
docker-compose ps

# Restart services
docker-compose restart postgres
```

3. Delta Lake Write Errors

```bash
# Clear checkpoints
rm -rf C:/etl/checkpoints/*

# Vacuum old files
spark.sql("VACUUM delta.`path/to/table` RETAIN 168 HOURS")
```

4. Excel Read Errors

```bash
# Install additional Excel libraries
pip install xlrd openpyxl python-calamine
```

📝 Best Practices

1. Incremental Processing

- Pipeline automatically detects new/modified files via hash comparison
- Only processes changed data
- Full audit trail maintained

2. Error Handling

- All errors logged to PostgreSQL
- Automatic retry for transient failures
- Partial success handling (some files can fail)

3. Data Quality

- Non-destructive checks (flags added, data preserved)
- Business rules monitored but not enforced
- Quality reports for manual review

4. Performance

- Partitioned Delta tables for fast queries
- Adaptive query execution enabled
- Parallel processing of features

🔒 Security Considerations

1. **Credentials**: Use environment variables, never hardcode
2. **Database**: Use connection pooling and prepared statements
3. **Files**: Implement file system permissions
4. **Logs**: Sanitize sensitive data in logs

📦 Production Deployment

Deployment Checklist

- [ ] Configure production database credentials
- [ ] Set appropriate Spark memory settings
- [ ] Configure Prefect schedules
- [ ] Set up monitoring alerts
- [ ] Enable backups for metadata database
- [ ] Configure Delta Lake retention policies
- [ ] Set up log rotation
- [ ] Test disaster recovery procedures

Scaling Considerations

- Use Spark cluster mode for large datasets
- Implement Delta Lake optimize operations
- Consider cloud storage (S3, Azure Blob, GCS)
- Use distributed Prefect agents

🤝 Contributing

Contributions welcome! Please:

1. Fork the repository
2. Create feature branch
3. Add tests
4. Submit pull request

📄 License

MIT License - feel free to use in your projects

📞 Support

For issues and questions:

- Create GitHub issue
- Check documentation
- Review metadata logs in PostgreSQL

🎓 Learning Resources

- Delta Lake Documentation
- PySpark Documentation
- Prefect Documentation
- Medallion Architecture

Built with ❤️ for robust, scalable ETL pipelines