# Technical Documentation

## Table of Contents

1. [Project Overview]
2. [Architecture Overview]
3. [Technology Stack]
4. [Data Sources]
5. [Pipeline Components]]
6. [Data Flow]]
7. [Infrastructure & Deployment]
8. [Configuration Management]
9. [Performance & Scalability]
10. [Monitoring & Logging]
11. [Troubleshooting Guide]
12. [Development Workflow]

## Project Overview

### Purpose

This project implements a modern ELT (Extract, Load, Transform) data pipeline that extracts e-commerce data from Supabase (PostgreSQL), loads it into Google BigQuery, and transforms it using dbt, all orchestrated by Dagster. The pipeline processes Brazilian e-commerce data (Olist dataset) to create analytics-ready data marts for business intelligence.

### Key Features

* \*\*Real-time Data Extraction\*\*: Automated extraction from Supabase PostgreSQL database
* \*\*Scalable Data Warehouse\*\*: Google BigQuery for petabyte-scale analytics
* \*\*Modern Transformations\*\*: dbt for SQL-based data modeling and testing
* \*\*Robust Orchestration\*\*: Dagster for asset-based pipeline management
* \*\*High Availability\*\*: Cloud Run deployment with auto-scaling
* \*\*Data Quality\*\*: Built-in testing and validation at each stage

## Architecture Overview

┌─────────────┐ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐  
│ Supabase │▶ │ Meltano │▶ │ BigQuery │ ▶│ dbt │▶│ Dagster │  
│ (PostgreSQL). │ │ (Extraction) │ │ (Data Lake) │ │ (Transform) │ │(Orchestrate) │  
└─────────────┘ └─────────────┘ └─────────────┘ └─────────────┘ └─────────────┘  
 │ │ │ │ │  
 │ │ │ │ │  
 Raw Data ELT Pipeline Data Warehouse Analytics Pipeline Management

### Data Pipeline Phases

\*\*Phase 1\*\*: Supabase → BigQuery (Raw Data Extraction)

\*\*Phase 2\*\*: Raw → Staging (Data Cleaning & Standardization)

\*\*Phase 3\*\*: Staging → Dimensions/Facts (Data Modeling)

\*\*Phase 4\*\*: Dimensions/Facts → Analytics (Business Metrics)

\*\*Phase 5\*\*: Monitoring & Notifications (Pipeline Summaries)

## Technology Stack

### Core Technologies

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Technology** | **Version** | **Purpose** |
| \*\*Database\*\* | Supabase (PostgreSQL) | 15+ | Source system for e-commerce data |
| \*\*ELT Tool\*\* | Meltano | 3.0+ | Data extraction and loading orchestration |
| \*\*Data Warehouse\*\* | Google BigQuery | Latest | Cloud-native analytics database |
| \*\*Transformation\*\* | dbt (Data Build Tool) | 1.7+ | SQL-based data transformations |
| \*\*Orchestration\*\* | Dagster | 1.11.9 | Asset-based data pipeline orchestration |
| \*\*Containerization\*\* | Docker | Latest | Application containerization |
| \*\*Cloud Platform\*\* | Google Cloud Run | Latest | Serverless container deployment |

### Supporting Technologies

|  |  |  |
| --- | --- | --- |
| **Component** | **Technology** | **Purpose** |
| \*\*Language\*\* | Python 3.11 | Primary development language |
| \*\*Environment\*\* | Conda/Miniforge | Dependency management |
| \*\*Version Control\*\* | Git/GitHub | Source code management |
| \*\*CI/CD\*\* | Google Cloud Build | Automated deployment |
| \*\*Monitoring\*\* | Cloud Run Logging | Application monitoring |
| \*\*Notifications\*\* | SendGrid | Email notifications |

## Data Sources

### Supabase Database Schema

The source system contains Brazilian e-commerce data with the following tables:

#### Core Business Tables

\*\*olist\_customers\_dataset\*\*

- Customer demographics and location data

- Fields: customer\_id, customer\_unique\_id, customer\_zip\_code\_prefix, customer\_city, customer\_state

\*\*olist\_orders\_dataset\*\*

- Order lifecycle and status information

- Fields: order\_id, customer\_id, order\_status, order\_purchase\_timestamp, order\_delivered\_timestamp

\*\*olist\_order\_items\_dataset\*\*

- Individual items within orders

- Fields: order\_id, order\_item\_id, product\_id, seller\_id, shipping\_limit\_date, price, freight\_value

\*\*olist\_order\_payments\_dataset\*\*

- Payment information and methods

- Fields: order\_id, payment\_sequential, payment\_type, payment\_installments, payment\_value

\*\*olist\_order\_reviews\_dataset\*\*

- Customer reviews and ratings

- Fields: review\_id, order\_id, review\_score, review\_comment\_title, review\_comment\_message

\*\*olist\_products\_dataset\*\*

- Product catalog information

- Fields: product\_id, product\_category\_name, product\_name\_length, product\_description\_length

\*\*olist\_sellers\_dataset\*\*

- Seller/merchant information

- Fields: seller\_id, seller\_zip\_code\_prefix, seller\_city, seller\_state

\*\*olist\_geolocation\_dataset\*\*

- Geographic coordinate data

- Fields: geolocation\_zip\_code\_prefix, geolocation\_lat, geolocation\_lng

\*\*product\_category\_name\_translation\*\*

- Product category translations (Portuguese to English)

- Fields: product\_category\_name, product\_category\_name\_english

### Connection Configuration

Language: yaml

# Meltano Configuration (meltano.yml)  
extractors:  
 - name: tap-postgres  
 variant: meltanolabs  
 config:  
 host: aws-1-ap-southeast-1.pooler.supabase.com  
 port: 5432  
 user: postgres.royhmnxmsfichopabwsi  
 database: postgres  
 password: ${TAP\_POSTGRES\_PASSWORD}  
 filter\_schemas: [public]

## Pipeline Components

### 1. Meltano (Extract & Load)

**Purpose**: Extracts data from Supabase PostgreSQL and loads into BigQuery raw dataset.

**Configuration Files**:

* `bec-meltano/meltano.yml`: Main configuration
* `bec-meltano/plugins/`: Plugin configurations

**Key Features**:

* \*\*Incremental Loading\*\*: Supports incremental data extraction
* \*\*Schema Evolution\*\*: Automatically handles schema changes
* \*\*Data Validation\*\*: Built-in data quality checks
* \*\*Retry Logic\*\*: Automatic retry on failures

**Extractors**:

* `tap-postgres`: Extracts from Supabase PostgreSQL
* \*\*Loaders\*\*:
* `target-bigquery`: Loads data into Google BigQuery

### 2. BigQuery (Data Warehouse)

**Purpose**: Cloud-native data warehouse for storing and querying large datasets.

**Dataset Structure**:

project-olist-470307/  
├── olist\_raw/ # Raw data from Supabase  
├── dbt\_olist\_stg/ # Staging tables (cleaned data)  
├── dbt\_olist\_dwh/ # Data warehouse (dimensions/facts)  
└── dbt\_olist\_analytics/ # Analytics marts (business metrics)

**Configuration**:

* \*\*Location\*\*: asia-southeast1 (Singapore)
* \*\*Storage\*\*: Standard storage class
* \*\*Partitioning\*\*: Date-based partitioning on fact tables
* \*\*Clustering\*\*: Optimized clustering keys for performance

### 3. dbt (Transform)

**Purpose**: SQL-based transformations to create analytics-ready data.

**Transformation Layers**:

\*\*Staging Layer\*\* (`models/staging/`):

- Data cleaning and standardization

- Column renaming and type casting

- Basic data quality checks

- Null handling and default values

\*\*Dimension Layer\*\* (`models/warehouse/`):

- Slowly changing dimensions (SCD)

- Surrogate key generation

- Data denormalization

- Reference data enrichment

- Transactional fact tables

- Aggregate measures

\*\*Analytics Layer\*\* (`models/marts/analytics/`):

- Business-specific metrics

- Pre-aggregated tables

- KPI calculations

- Reporting-ready datasets

### 4. Dagster (Orchestration)

**Purpose**: Asset-based orchestration platform for managing the entire data pipeline.

**Key Concepts**:

\*\*Assets\*\*: Discrete data artifacts (tables, files, models)

\*\*Jobs\*\*: Collections of assets that run together

\*\*Schedules\*\*: Time-based or event-based execution

\*\*Resources\*\*: Shared configurations and connections

\*\*Sensors\*\*: Event-driven pipeline triggers

**Asset Organization**:

Language: python

# Phase 1: Extraction (1 asset)  
\_1\_staging\_to\_bigquery  
  
# Phase 2: Staging (9 assets, see below next page for more details)  
\_2a\_processing\_stg\_orders  
\_2b\_processing\_stg\_order\_items  
\_2c\_processing\_stg\_products  
# ... (9 total)  
  
# Phase 3: Dimensions & Facts (9 assets, see below next page for more details)  
\_3a\_processing\_dim\_orders  
\_3b\_processing\_dim\_products  
\_3c\_processing\_dim\_customers  
# ... (9 total)  
  
# Phase 4: Analytics (8 assets, see below next page for more details)  
\_4a\_processing\_revenue\_analytics\_obt  
\_4b\_processing\_orders\_analytics\_obt  
# ... (8 total)  
  
# Phase 5: Monitoring (1 asset)  
\_5\_dbt\_summaries

**Configuration**:

Language: yaml

# bec-dagster/dagster.yaml  
run\_coordinator:  
 module: dagster.\_core.run\_coordinator.default\_run\_coordinator  
 class: DefaultRunCoordinator  
  
run\_launcher:  
 module: dagster.\_core.launcher.sync\_in\_memory\_run\_launcher  
 class: SyncInMemoryRunLauncher  
  
telemetry:  
 enabled: false

## F. Data Flow

### Detailed Pipeline Flow

#### Phase 1: Supabase → BigQuery (Raw Data Extraction)

Supabase Tables → Meltano tap-postgres → BigQuery Raw Dataset  
**Process**:

Meltano discovers Supabase table schemas

Extracts data using Singer protocol

Transforms data to BigQuery-compatible format

Loads data into `olist\_raw` dataset

Creates/updates table schemas automatically

**Assets**: \_1\_staging\_to\_bigquery

#### Phase 2: Raw → Staging (Data Cleaning)

BigQuery Raw → dbt staging models → BigQuery Staging Dataset  
**Process**:

Read raw data from `olist\_raw` tables

Apply data cleaning transformations

Standardize column names and types

Handle null values and data quality issues

Create staging tables in `dbt\_olist\_stg` dataset

**Assets**:

* `\_2a\_processing\_stg\_orders`
* `\_2b\_processing\_stg\_order\_items`
* `\_2c\_processing\_stg\_products`
* `\_2d\_processing\_stg\_order\_reviews`
* `\_2e\_processing\_stg\_order\_payments`
* `\_2f\_processing\_stg\_sellers`
* `\_2g\_processing\_stg\_customers`
* `\_2h\_processing\_stg\_geolocation`
* `\_2i\_processing\_stg\_product\_category\_name\_translation`

#### Phase 3: Staging → Dimensions/Facts (Data Modeling)

Staging Tables → dbt dimension/fact models → BigQuery Data Warehouse  
**Process**:

Create dimension tables with business keys

Implement slowly changing dimensions (SCD)

Build fact tables with measures and foreign keys

Apply business rules and calculations

Generate surrogate keys and maintain referential integrity

**Assets**:

* `\_3a\_processing\_dim\_orders`
* `\_3b\_processing\_dim\_products`
* `\_3c\_processing\_dim\_order\_reviews`
* `\_3d\_processing\_dim\_payment`
* `\_3e\_processing\_dim\_seller`
* `\_3f\_processing\_dim\_customer`
* `\_3g\_processing\_dim\_geolocation`
* `\_3h\_processing\_dim\_date`
* `\_3i\_processing\_fact\_order\_items`

#### Phase 4: Dimensions/Facts → Analytics (Business Metrics)

Data Warehouse → dbt analytics models → BigQuery Analytics Dataset  
**Process**:

Create business-specific analytical tables

Calculate KPIs and performance metrics

Build aggregated summary tables

Generate customer, product, and geographic analytics

Create operational and financial reporting datasets

**Assets**:

* `\_4a\_processing\_revenue\_analytics\_obt`
* `\_4b\_processing\_orders\_analytics\_obt`
* `\_4c\_processing\_delivery\_analytics\_obt`
* `\_4d\_processing\_customer\_analytics\_obt`
* `\_4e\_processing\_geographic\_analytics\_obt`
* `\_4f\_processing\_payment\_analytics\_obt`
* `\_4g\_processing\_seller\_analytics\_obt`
* `\_4h\_processing\_operation\_analytics\_obt`

#### Phase 5: Monitoring & Notifications

Pipeline Execution → Summary Generation → Email Notifications  
**Process**:

Collect pipeline execution statistics

Generate data quality reports

Create performance summaries

Send email notifications to stakeholders

Log pipeline results and metrics

**Assets**: \_5\_dbt\_summaries

## Infrastructure & Deployment

### Cloud Run Deployment

**Service Configuration**:

Language: yaml

# Cloud Run Service  
Service: supabase-meltano-pipeline  
Region: asia-southeast1  
Memory: 8Gi  
CPU: 4 cores  
Timeout: 1800 seconds (1 hour)  
Max Instances: 10  
Container Concurrency: 20  
Port: 3000

**Environment Variables**:

Language: bash

# Google Cloud  
BQ\_PROJECT\_ID=project-olist-470307  
BQ\_LOCATION=asia-southeast1  
GOOGLE\_APPLICATION\_CREDENTIALS=/app/bec\_dbt/service-account-key.json  
  
# BigQuery Datasets  
TARGET\_RAW\_DATASET=olist\_raw  
TARGET\_STAGING\_DATASET=dbt\_olist\_stg  
TARGET\_BIGQUERY\_DATASET=dbt\_olist\_dwh  
TARGET\_ANALYTICAL\_DATASET=dbt\_olist\_analytics  
  
# Database Connection  
TAP\_POSTGRES\_PASSWORD=xxxxxxxx  
  
# Dagster  
DAGSTER\_HOME=/app/dagster  
MELTANO\_PROJECT\_ROOT=/app/bec-meltano

### Containerization

**Dockerfile Configuration**:

Language: dockerfile

FROM python:3.11-slim  
  
# Install system dependencies  
RUN apt-get update && apt-get install -y git curl && rm -rf /var/lib/apt/lists/\*  
  
# Install Miniforge (conda)  
RUN curl -L https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-Linux-x86\_64.sh -o miniforge.sh \  
 && bash miniforge.sh -b -p /opt/conda \  
 && rm miniforge.sh  
  
# Create conda environment  
COPY requirements-bec.yaml .  
RUN conda env create -f requirements-bec.yaml  
  
# Copy application code  
COPY . /app  
WORKDIR /app  
  
# Set environment variables  
ENV PATH="/opt/conda/envs/bec/bin:$PATH"  
ENV MELTANO\_PROJECT\_ROOT="/app/bec-meltano"  
  
# Create persistent directories  
RUN mkdir -p /app/dagster/storage  
  
# Start application  
CMD ["python", "start\_dagster\_simple.py"]

### Scaling Configuration

**Horizontal Scaling**:

* \*\*Auto-scaling\*\*: 0 to 10 instances based on demand
* \*\*CPU Threshold\*\*: 80% utilization
* \*\*Memory Threshold\*\*: 6.4Gi (80% of 8Gi)
* \*\*Request Concurrency\*\*: 20 concurrent requests per instance

**Vertical Scaling**:

* \*\*Memory\*\*: 8Gi per instance
* \*\*CPU\*\*: 4 vCPUs per instance
* \*\*Storage\*\*: Ephemeral (container-based)

### High Availability

**Reliability Features**:

* \*\*Multi-zone deployment\*\*: Automatic zone distribution
* \*\*Health checks\*\*: Container health monitoring
* \*\*Graceful shutdowns\*\*: Proper connection cleanup
* \*\*Retry logic\*\*: Automatic retry on transient failures
* \*\*Circuit breakers\*\*: Prevents cascade failures

## Configuration Management

### Environment Files

**Production (.env)**:

Language: bash

# BigQuery Configuration  
BQ\_PROJECT\_ID=project-olist-470307  
BQ\_LOCATION=asia-southeast1  
  
# Dataset Names  
TARGET\_RAW\_DATASET=olist\_raw  
TARGET\_STAGING\_DATASET=dbt\_olist\_stg  
TARGET\_BIGQUERY\_DATASET=dbt\_olist\_dwh  
TARGET\_ANALYTICAL\_DATASET=dbt\_olist\_analytics  
  
# Database Connection  
DB\_HOST=db.royhmnxmsfichopabwsi.supabase.co  
DB\_PORT=5432  
DB\_NAME=postgres  
DB\_USER=postgres.royhmnxmsfichopabwsi  
TAP\_POSTGRES\_PASSWORD=xxxxx  
SUPABASE\_HOST=aws-1-ap-southeast-1.pooler.supabase.com

# Email Notifications  
SENDGRID\_API\_KEY=xxx

[RECIPIENT\_EMAILS=user1@example.com,user2@example.com](mailto:RECIPIENT_EMAILS=user1@example.com,user2@example.com)

### dbt Configuration

**Project Configuration (dbt\_project.yml)**:

Language: yaml

name: 'bec\_dbt'  
version: '1.0.0'  
profile: 'bec\_dbt'  
  
model-paths: ["models"]  
analysis-paths: ["analyses"]  
test-paths: ["tests"]  
seed-paths: ["seeds"]  
macro-paths: ["macros"]  
snapshot-paths: ["snapshots"]  
  
clean-targets:  
 - "target"  
 - "logs"  
  
models:  
 bec\_dbt:  
 staging:  
 +materialized: table  
 +schema: staging  
 marts:  
 dimensions:  
 +materialized: table  
 +schema: dimensions  
 facts:  
 +materialized: table  
 +schema: facts  
 analytics:  
 +materialized: table  
 +schema: analytics

**Connection Profile (profiles.yml)**:

Language: yaml

bec\_dbt:  
 target: dev  
 outputs:  
 dev:  
 type: bigquery  
 method: service-account  
 keyfile: ./service-account-key.json  
 project: project-olist-470307  
 dataset: dbt\_olist\_dwh  
 location: asia-southeast1  
 threads: 4  
 timeout\_seconds: 300

### Meltano Configuration

**Main Configuration (meltano.yml)**:

Language: yaml

version: 1  
default\_environment: dev  
project\_id: 0198cf5e-5994-79cf-b139-fa589862c550  
  
plugins:  
 extractors:  
 - name: tap-postgres  
 variant: meltanolabs  
 pip\_url: meltanolabs-tap-postgres  
 config:  
 database: postgres  
 filter\_schemas: [public]  
 host: aws-1-ap-southeast-1.pooler.supabase.com  
 port: 5432  
 user: postgres.royhmnxmsfichopabwsi  
 password: $TAP\_POSTGRES\_PASSWORD  
 max\_record\_count: 0  
  
 loaders:  
 - name: target-bigquery  
 variant: z3z1ma  
 pip\_url: git+https://github.com/z3z1ma/target-bigquery.git  
 config:  
 project: $BQ\_PROJECT\_ID  
 dataset: $TARGET\_RAW\_DATASET  
 location: $BQ\_LOCATION  
 credentials\_path: ./bigquery-credentials.json  
 method: streaming\_insert  
 denormalized: true  
 batch\_size: 1000

## Performance & Scalability

### Performance Optimizations

**BigQuery Optimizations**:

* \*\*Partitioning\*\*: Date-based partitioning on order tables
* \*\*Clustering\*\*: Optimized clustering keys for common queries
* \*\*Table Design\*\*: Denormalized structures for analytics
* \*\*Query Optimization\*\*: Efficient JOIN strategies and filtering

**dbt Optimizations**:

* \*\*Incremental Models\*\*: Only process new/changed data
* \*\*Materialization Strategy\*\*: Appropriate table vs view materialization
* \*\*Parallel Execution\*\*: Concurrent model building
* \*\*Caching\*\*: Model result caching for faster iterations

**Dagster Optimizations**:

* \*\*Asset Partitioning\*\*: Parallel processing of independent assets
* \*\*Resource Management\*\*: Efficient connection pooling
* \*\*Memory Management\*\*: Optimized memory usage patterns
* \*\*Concurrency Control\*\*: Balanced parallel execution

### Scalability Patterns

**Data Volume Scaling**:

* \*\*Incremental Processing\*\*: Only process changed data
* \*\*Partitioned Tables\*\*: Efficient large table management
* \*\*Archive Strategy\*\*: Historical data archiving
* \*\*Compression\*\*: Optimized storage compression

**Compute Scaling**:

* \*\*Horizontal Scaling\*\*: Multiple Cloud Run instances
* \*\*Vertical Scaling\*\*: Increased memory and CPU per instance
* \*\*BigQuery Slots\*\*: Reserved compute capacity
* \*\*Connection Pooling\*\*: Efficient database connections

**Cost Optimization**:

* \*\*On-Demand Scaling\*\*: Scale to zero when idle
* \*\*Spot Instances\*\*: Use preemptible compute when possible
* \*\*Storage Tiers\*\*: Appropriate storage classes
* \*\*Query Optimization\*\*: Minimize BigQuery query costs

### Performance Metrics

**Pipeline Performance**:

Phase 1 (Extraction): 3-5 minutes  
Phase 2 (Staging): < 1 minute  
Phase 3 (Modeling): < 1 minute  
Phase 4 (Analytics): < 1 minute  
Phase 5 (Monitoring): < 1 minute  
Total Pipeline Runtime: 3-6 minutes

**Data Volumes**:

Raw Data: ~1-5GB  
Staging Data: ~1-5GB  
Warehouse Data: ~2-8GB  
Analytics Data: ~500MB-2GB

## Monitoring & Logging

### Application Monitoring

**Dagster Monitoring**:

* \*\*Asset Execution Status\*\*: Success/failure tracking
* \*\*Asset Materialization\*\*: Data freshness monitoring
* \*\*Resource Utilization\*\*: Memory and CPU monitoring
* \*\*Run Duration\*\*: Performance trend analysis

**Cloud Run Monitoring**:

* \*\*Request Metrics\*\*: Request count, latency, errors
* \*\*Resource Metrics\*\*: CPU, memory, disk utilization
* \*\*Instance Metrics\*\*: Instance count, startup time
* \*\*Error Rates\*\*: 4xx/5xx error tracking

### Logging Strategy

**Log Levels**:

Language: python

# Application Logging  
DEBUG: Detailed execution information  
INFO: General pipeline progress  
WARNING: Non-critical issues and performance concerns  
ERROR: Execution failures and critical issues

**Log Destinations**:

* \*\*Cloud Run Logs\*\*: Application stdout/stderr
* \*\*BigQuery Logs\*\*: Query execution logs
* \*\*Dagster Logs\*\*: Pipeline execution logs
* \*\*Email Notifications\*\*: Critical alerts and summaries

### Alerting

**Alert Categories**:

\*\*Pipeline Failures\*\*: Asset execution failures

\*\*Data Quality Issues\*\*: Test failures and data anomalies

\*\*Performance Degradation\*\*: Slow execution times

\*\*Resource Constraints\*\*: Memory/CPU limits reached

\*\*Connection Issues\*\*: Database/API connectivity problems

**Notification Channels**:

* \*\*Email\*\*: Immediate failure notifications
* \*\*Dashboard\*\*: Real-time status monitoring
* \*\*Logs\*\*: Detailed troubleshooting information

### Data Quality Monitoring

**dbt Tests**:

Language: sql

-- Example data quality tests  
tests:  
 - not\_null: [order\_id, customer\_id]  
 - unique: [order\_id]  
 - accepted\_values:  
 column: order\_status  
 values: ['delivered', 'shipped', 'processing', 'canceled']  
 - relationships:  
 to: ref('dim\_customers')  
 field: customer\_id

**Custom Quality Checks**:

* \*\*Row Count Validation\*\*: Ensure expected data volumes
* \*\*Freshness Checks\*\*: Monitor data recency
* \*\*Completeness Tests\*\*: Check for missing critical data
* \*\*Business Rule Validation\*\*: Verify domain-specific rules

## Troubleshooting Guide

### Common Issues

#### 1. Database Connection Issues

**Symptoms**:

* "password authentication failed for user 'postgres'"
* Connection timeouts

**Solutions**:

Language: bash

# Check environment variables  
echo $TAP\_POSTGRES\_PASSWORD  
  
# Test database connection  
psql -h aws-1-ap-southeast-1.pooler.supabase.com -p 5432 -U postgres.royhmnxmsfichopabwsi -d postgres  
  
# Verify Meltano configuration  
meltano invoke tap-postgres --discover

#### 2. BigQuery Authentication Issues

**Symptoms**:

* "No BigQuery credentials found"
* Authentication errors

**Solutions**:

Language: bash

# Check service account file  
ls -la /app/bec\_dbt/service-account-key.json  
  
# Verify environment variables  
echo $GOOGLE\_APPLICATION\_CREDENTIALS  
echo $BQ\_PROJECT\_ID  
  
# Test BigQuery connection  
bq ls --project\_id=project-olist-470307

#### 3. dbt Execution Failures

**Symptoms**:

* Model compilation errors
* SQL execution failures

**Solutions**:

Language: bash

# Check dbt configuration  
dbt debug  
  
# Compile models  
dbt compile  
  
# Run specific model  
dbt run --models stg\_orders  
  
# Run tests  
dbt test --models stg\_orders

#### 4. Dagster Configuration Errors

**Symptoms**:

* "DagsterInvalidConfigError"
* Asset loading failures

**Solutions**:

Language: bash

# Validate Dagster configuration  
dagster instance info  
  
# Check asset definitions  
dagster asset list  
# Test asset execution  
dagster asset materialize --select \_1\_staging\_to\_bigquery

#### 5. Cloud Run Deployment Issues

**Symptoms**:

* Container startup failures
* "Revision is not ready"

**Solutions**:

Language: bash

# Check Cloud Run logs  
gcloud run services logs tail supabase-meltano-pipeline --region=asia-southeast1  
  
# Verify container image  
docker run -it supabase-meltano-pipeline:latest bash  
  
# Check resource limits  
gcloud run services describe supabase-meltano-pipeline --region=asia-southeast1

### Debugging Commands

**Local Development**:

Language: bash

# Start local Dagster  
conda activate bec  
cd bec-dagster  
dagster dev --host 0.0.0.0 --port 3000  
  
# Test Meltano extraction  
cd bec-meltano  
meltano run tap-postgres target-bigquery  
  
# Run dbt models  
cd bec\_dbt  
dbt run --target dev

**Container Debugging**:

Language: bash

# Build and test locally  
docker build -f Dockerfile.simple -t test-pipeline .  
docker run -p 3000:3000 test-pipeline  
  
# Inspect running container  
docker exec -it container\_name bash

### Performance Debugging

**BigQuery Query Optimization**:

Language: sql

-- Check query performance  
SELECT  
 job\_id,  
 query,  
 total\_bytes\_processed,  
 total\_slot\_ms,  
 creation\_time,  
 start\_time,  
 end\_time  
FROM `region-asia-southeast1`.INFORMATION\_SCHEMA.JOBS\_BY\_PROJECT  
WHERE creation\_time >= TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 1 HOUR)  
ORDER BY creation\_time DESC;

**Dagster Performance Analysis**:

Language: python

# Asset execution metrics  
from dagster import DagsterInstance  
  
instance = DagsterInstance.get()  
runs = instance.get\_runs()  
for run in runs:  
 print(f"Run: {run.run\_id}, Status: {run.status}, Duration: {run.end\_time - run.start\_time}")

## L. Development Workflow

### Local Development Setup

\*\*Environment Setup\*\*:

Language: bash

# Clone repository  
git clone https://github.com/rubyferdianto/s3-rds-bq-dagster.git  
cd supabase-meltano-bq-dagster  
  
# Create conda environment  
conda env create -f requirements-bec.yaml  
conda activate bec  
  
# Set up environment variables  
cp .env.example .env  
# Edit .env with your credentials

\*\*Local Testing\*\*:

Language: bash

# Test Meltano extraction  
cd bec-meltano  
meltano run tap-postgres target-bigquery  
  
# Test dbt transformations  
cd ../bec\_dbt  
dbt run --target dev  
  
# Test Dagster pipeline  
cd ../bec-dagster  
dagster dev

### Development Practices

**Code Organization**:

* \*\*Modular Design\*\*: Separate concerns into distinct modules
* \*\*Configuration Management\*\*: Environment-based configuration
* \*\*Error Handling\*\*: Comprehensive error handling and logging
* \*\*Documentation\*\*: Inline documentation and README files

**Testing Strategy**:

* \*\*Unit Tests\*\*: Individual component testing
* \*\*Integration Tests\*\*: End-to-end pipeline testing
* \*\*Data Quality Tests\*\*: dbt test framework
* \*\*Performance Tests\*\*: Load and stress testing

**Version Control**:

* \*\*Branch Strategy\*\*: Feature branches with pull requests
* \*\*Commit Messages\*\*: Descriptive commit messages
* \*\*Code Reviews\*\*: Peer review process
* \*\*Release Tags\*\*: Semantic versioning

### Deployment Process

\*\*Development\*\*:

- Local development and testing

- Feature branch creation

- Code review and approval

\*\*Staging\*\*:

- Deploy to staging environment

- Integration testing

- Performance validation

\*\*Production\*\*:

- Deploy to Cloud Run

- Monitor deployment health

- Rollback capability

**CI/CD Pipeline**:

Language: yaml

# .github/workflows/deploy.yml  
name: Deploy to Cloud Run  
on:  
 push:  
 branches: [main]  
  
jobs:  
 deploy:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v2  
 - name: Build Docker image  
 run: docker build -f Dockerfile.simple -t gcr.io/$PROJECT\_ID/pipeline:$GITHUB\_SHA .  
 - name: Push to GCR  
 run: docker push gcr.io/$PROJECT\_ID/pipeline:$GITHUB\_SHA  
 - name: Deploy to Cloud Run  
 run: gcloud run deploy pipeline --image gcr.io/$PROJECT\_ID/pipeline:$GITHUB\_SHA

### Maintenance Procedures

**Regular Maintenance**:

* \*\*Dependency Updates\*\*: Regular package updates
* \*\*Security Patches\*\*: Security vulnerability remediation
* \*\*Performance Monitoring\*\*: Continuous performance analysis
* \*\*Data Archival\*\*: Historical data management

**Backup Procedures**:

* \*\*Code Backup\*\*: Git repository backup
* \*\*Configuration Backup\*\*: Environment configuration backup
* \*\*Data Backup\*\*: BigQuery dataset backup
* \*\*Schema Backup\*\*: Database schema versioning

## Security Considerations

### Data Security

**Encryption**:

* \*\*In Transit\*\*: TLS encryption for all connections
* \*\*At Rest\*\*: BigQuery automatic encryption
* \*\*Application\*\*: Environment variable encryption

**Access Control**:

* \*\*Service Accounts\*\*: Principle of least privilege
* \*\*IAM Roles\*\*: Role-based access control
* \*\*Network Security\*\*: VPC and firewall rules
* \*\*Audit Logging\*\*: Comprehensive audit trails

### Credential Management

**Secret Management**:

* \*\*Environment Variables\*\*: Secure environment variable storage
* \*\*Service Account Keys\*\*: JSON key file management
* \*\*Password Rotation\*\*: Regular credential rotation
* \*\*Access Monitoring\*\*: Credential usage monitoring

**Best Practices**:

* Never commit credentials to version control
* Use separate credentials for different environments
* Implement credential rotation policies
* Monitor for credential exposure

## Cost Optimization

### Resource Optimization

**Cloud Run**:

* \*\*Auto-scaling\*\*: Scale to zero when idle
* \*\*Resource Limits\*\*: Appropriate memory/CPU allocation
* \*\*Request Optimization\*\*: Efficient request handling
* \*\*Cold Start Optimization\*\*: Minimize container startup time

**BigQuery**:

* \*\*Query Optimization\*\*: Efficient query patterns
* \*\*Storage Optimization\*\*: Appropriate table design
* \*\*Slot Management\*\*: Reserved vs on-demand slots
* \*\*Data Lifecycle\*\*: Automatic data archival

### Cost Monitoring

**Tracking Metrics**:

* \*\*Compute Costs\*\*: Cloud Run execution costs
* \*\*Storage Costs\*\*: BigQuery storage costs
* \*\*Network Costs\*\*: Data transfer costs
* \*\*API Costs\*\*: Service API usage costs

**Cost Controls**:

* \*\*Budget Alerts\*\*: Automatic cost alerts
* \*\*Resource Quotas\*\*: Maximum resource limits
* \*\*Usage Analytics\*\*: Cost attribution analysis
* \*\*Optimization Recommendations\*\*: Regular cost reviews

## Future Enhancements

### Planned Improvements

**Technical Enhancements**:

* \*\*Real-time Streaming\*\*: Apache Kafka integration
* \*\*Advanced Analytics\*\*: Machine learning model integration
* \*\*Data Lineage\*\*: Comprehensive data lineage tracking
* \*\*Advanced Monitoring\*\*: APM tool integration

**Business Enhancements**:

* \*\*Additional Data Sources\*\*: More business system integration
* \*\*Advanced Analytics\*\*: Predictive analytics and forecasting
* \*\*Real-time Dashboards\*\*: Live business intelligence
* \*\*Self-service Analytics\*\*: User-friendly data exploration

**Infrastructure Improvements**:

* \*\*Multi-region Deployment\*\*: Global availability
* \*\*Advanced Security\*\*: Zero-trust security model
* \*\*Performance Optimization\*\*: Query acceleration
* \*\*Cost Optimization\*\*: Advanced cost management

## Conclusion

This technical documentation provides comprehensive coverage of the Supabase-Meltano-BigQuery-dbt-Dagster data pipeline. The architecture demonstrates modern data engineering best practices with:

* \*\*Scalable ELT Pipeline\*\*: Efficient data extraction, loading, and transformation
* \*\*Cloud-Native Architecture\*\*: Leveraging Google Cloud Platform services
* \*\*Modern Orchestration\*\*: Asset-based pipeline management with Dagster
* \*\*Data Quality Focus\*\*: Comprehensive testing and validation
* \*\*Production-Ready Deployment\*\*: Container-based deployment with auto-scaling

The pipeline successfully processes Brazilian e-commerce data to create analytics-ready datasets for business intelligence and decision-making.