# **Technical Solution Documentation: Supply Chain Analytics Platform**

Version: 1.0

Date: July 22, 2025

## **1. Introduction**

### **1.1. Project Goal**

The primary goal of this project is to architect and implement a scalable, end-to-end data platform to overcome challenges in supply chain visibility. By centralizing inventory, supplier, and transaction data, the platform empowers business users with actionable insights through interactive dashboards, aiming to optimize inventory levels, improve supplier performance evaluation, and reduce operational costs associated with data fragmentation.

### **1.2. Scope**

The scope of this project includes:

* **Data Ingestion** from a source PostgreSQL database.
* **ETL Processing** to clean, transform, and model the data.
* **Infrastructure Provisioning** of all required AWS services using Terraform.
* **Data Warehousing** in a star-schema model within Amazon Redshift.
* **Orchestration** of the batch data pipelines.
* **Data Visualization** of key metrics in Power BI.

The project focuses on batch processing. Real-time data pipelines are considered out of scope for this version.

## **2. System Architecture**

### **2.1. Architecture Diagram**

The system is designed around a medallion architecture (Bronze, Silver, Gold) implemented on AWS.

*(This diagram shows the final data model. A full cloud architecture diagram should also be included here if available.)*

### **2.2. Data Flow**

The data pipeline is orchestrated by AWS Step Functions and proceeds as follows:

1. **Source & Ingestion**: The process begins with the ingest\_to\_s3.py script. This script connects to the source RDS PostgreSQL instance, extracts data from the transactional schema (original schema.sql), and uploads it as CSV files into the **Bronze S3 Bucket** (s3://scm-bronze-layer/). This layer holds the raw, unaltered data.
2. **Bronze-to-Silver Transformation**:
   * **Trigger**: An AWS Glue job running the transform\_to\_silver.py script is initiated.
   * **Process**: The script reads the raw CSV files from the Bronze layer. It performs data quality checks, cleanses data (e.g., handles nulls, standardizes formats), and converts the data into the more efficient and query-friendly Apache Parquet format.
   * **Output**: The processed data is written to the **Silver S3 Bucket** (s3://scm-silver-layer/). Data in this layer is clean and queryable but not yet modeled for analytics.
3. **Silver-to-Gold Transformation & Loading**:
   * **Trigger**: A second AWS Glue job running the load\_to\_gold.py script is initiated.
   * **Process**: This script reads the Parquet files from the Silver layer. It applies the final business logic to transform the data into the dimensional star schema. This involves creating surrogate keys, separating attributes into dimension tables, and aggregating measures into fact tables.
   * **Output**: The final dimension and fact tables are loaded into the **Amazon Redshift Data Warehouse**. The copy commands.sql script provides a reference for the COPY operations used to efficiently load data from S3 into Redshift.

### **2.3. Technology Stack Justification**

* **Terraform**: Chosen for its declarative syntax and cloud-agnostic nature, allowing for robust and repeatable infrastructure management.
* **AWS S3**: Selected as the data lake foundation due to its durability, scalability, and cost-effectiveness for storing vast amounts of data at different processing stages.
* **AWS Glue**: A serverless ETL service that eliminates the need to manage Spark clusters. It's ideal for the batch processing workloads in this project.
* **AWS Step Functions**: Provides reliable, serverless workflow orchestration with built-in error handling and state management, making complex ETL sequences easy to visualize and manage.
* **Amazon Redshift**: A powerful, petabyte-scale data warehouse optimized for columnar storage and parallel processing, making it ideal for the fast analytical queries required by Power BI.

## **3. Data Model**

### **3.1. Schema Design**

A **Star Schema** was chosen for the Gold layer in Redshift. This design is optimal for business intelligence and analytics for several reasons:

* **Simplicity**: The model is easy for analysts to understand, with fact tables at the center and descriptive dimensions surrounding them.
* **Query Performance**: The design requires fewer joins than a normalized schema, leading to faster query execution for aggregations.
* **BI Tool Compatibility**: Star schemas are the standard for BI tools like Power BI, which are optimized to work with this structure.

The final DDL for the star schema is available in sql/⭐schema.sql.

### **3.2. Tables**

* **Fact Tables**:
  + gold\_fact\_inventory\_transactions: Records every inventory movement (receipts, issues).
  + gold\_fact\_inventory\_snapshot: Provides a daily snapshot of inventory levels and value.
  + gold\_fact\_damage\_reports: Contains data on damaged goods and financial losses.
  + gold\_fact\_supplier\_performance: Stores monthly performance metrics for each supplier.
* **Dimension Tables**:
  + gold\_dim\_product: Describes products (name, SKU, price).
  + gold\_dim\_location: Describes warehouse or store locations.
  + gold\_dim\_supplier: Contains all supplier information.
  + gold\_dim\_category: Product categories.
  + gold\_dim\_status: Describes various status types (e.g., for inventory, reports).

## **4. Infrastructure as Code (IaC)**

### **4.1. Terraform Structure**

The Terraform code is modularized for better organization and reusability.

* main.tf: The root module that calls other modules.
* variables.tf: Defines input variables to customize deployments (e.g., bucket names, instance sizes).
* providers.tf: Configures the AWS provider.
* module/: Contains dedicated modules for each AWS service (s3, redshift, glue, rds). This approach allows for independent management and testing of each component.

### **4.2. Deployment Instructions**

1. **Prerequisites**: Ensure Terraform CLI is installed and AWS credentials are configured (e.g., via aws configure or environment variables).
2. **Navigate to the root directory** of the repository.
3. **Run terraform init** to initialize the backend and download providers.
4. **Run terraform plan -out=tfplan** to create an execution plan and save it to a file.
5. **Run terraform apply "tfplan"** to execute the plan and build the infrastructure.

## **5. Data Pipelines & Scripts**

### **5.1. scripts/ingest\_to\_s3.py**

* **Purpose**: Extracts data from the source PostgreSQL database.
* **Logic**: Uses the psycopg2 library to connect to the database, executes SELECT \* queries on source tables, and uses the boto3 library to upload the resulting data as CSVs to the S3 Bronze bucket.

### **5.2. scripts/transform\_to\_silver.py**

* **Purpose**: Cleans and standardizes raw data.
* **Framework**: AWS Glue with PySpark.
* **Logic**:
  + Reads CSVs from the Bronze layer into a Glue DynamicFrame.
  + Performs transformations such as data type casting, renaming columns for clarity, handling missing values, and ensuring date/timestamp consistency.
  + Writes the cleaned data as Parquet files to the Silver layer, partitioned by date where applicable.

### **5.3. scripts/load\_to\_gold.py**

* **Purpose**: Models data into the star schema and loads it into Redshift.
* **Framework**: AWS Glue with PySpark.
* **Logic**:
  + Reads multiple tables from the Silver layer.
  + Generates surrogate keys for dimensions.
  + Joins and aggregates data to create the fact tables.
  + Writes the final dimension and fact tables to a temporary location in S3.
  + Uses a post-execution command or a subsequent Lambda function to issue COPY commands to load the data from S3 into Redshift.

## **6. CI/CD with GitHub Actions**

A workflow is defined in .github/workflows/.

* **Trigger**: On push or pull request to the develop or main branch.
* **Jobs**:
  1. **Lint & Validate**: Checks Python code for style and runs terraform validate to ensure syntax is correct.
  2. **Terraform Plan**: For pull requests, it runs terraform plan to provide a summary of changes as a comment in the PR, allowing for peer review before merging.
  3. **Terraform Apply (Manual Trigger)**: A workflow that can be manually triggered to apply the Terraform configuration to the AWS environment.

## **7. BI & Visualization (Power BI)**

### **7.1. Connection**

* **Data Source**: Amazon Redshift.
* **Authentication**: Use the Redshift cluster endpoint, database name, and credentials (stored securely, e.g., in AWS Secrets Manager and accessed by an authorized user).
* **Connectivity Mode**: **DirectQuery** is recommended to ensure dashboards always reflect the latest data in the warehouse without needing to re-import it into Power BI.

### **7.2. Dashboards**

* **Inventory Overview**: Features KPIs like Total Inventory Value, Available Quantity, and visualizations for stock value by product and location.
* **Supplier Performance**: Includes a supplier scorecard ranking them by overall\_performance\_score, on\_time\_delivery, and quality\_score.
* **Damage & Loss Analysis**: Visualizes estimated\_loss\_value by reason and location to identify operational improvement areas.