# Comprehensive Guide: Setting Up a Trino-Nessie-MinIO Lakehouse with dbt Core

Current Date: Tuesday, June 3, 2025

Location Context: Chennai, Tamil Nadu, India

### Introduction

This guide provides a comprehensive, step-by-step walkthrough for setting up a modern, version-controlled data lakehouse locally using Docker. The architecture consists of:

* **MinIO:** S3-compatible object storage for the data lake.
* **Apache Iceberg:** Open table format for managing large analytic datasets.
* **Nessie:** A Git-like catalog for Iceberg, enabling data version control (branches, commits, merges).
* **Trino:** A distributed SQL query engine for querying data in the lakehouse.
* **Apache Spark:** For ingesting raw data into Iceberg tables.
* **dbt Core:** For transforming raw data into analytics-ready models.

This setup allows for robust data management, versioning, and efficient transformation workflows.

### Phase 1: Infrastructure Setup (Docker Compose)

This phase focuses on creating and configuring the core services using Docker Compose.

**Prerequisites:**

* Docker Desktop installed and running on your machine.
  + *Source:* [Docker Official Website](https://www.docker.com/products/docker-desktop/)

1.1 Project Directory Structure

Create a root folder for your project (e.g., my\_lakehouse\_project/). Inside it, create the following structure:

* my\_lakehouse\_project/
* ├── docker-compose.yml
* └── trino\_config/
* ├── config.properties
* └── catalog/
* └── nessie.properties

**1.2 Trino Configuration Files**

a) trino\_config/config.properties

This file configures the Trino server itself.

* # Indicates this node is the coordinator, responsible for planning queries.
* coordinator=true
* # Allows the coordinator node to also perform worker tasks (data processing).
* # Essential for a single-node setup.
* node-scheduler.include-coordinator=true
* # HTTP port Trino will listen on inside the container.
* http-server.http.port=8080
* # Max memory a single query can use across the cluster.
* query.max-memory=5GB
* # Max memory a single query can use on this node.
* query.max-memory-per-node=1GB
* # Enables the discovery service for nodes to find each other.
* discovery-server.enabled=true
* # URI of the discovery service (points to itself in a single-node setup).
* discovery.uri=http://localhost:8080

*Description:* This configuration sets up a single Trino node to act as both coordinator and worker, defines its port, and basic memory limits. The discovery service is enabled for proper startup.

b) trino\_config/catalog/nessie.properties

This file defines a Trino "catalog" named nessie, telling Trino how to connect to your Nessie server and use Iceberg tables. The filename (nessie.properties) determines the catalog name used in SQL queries (e.g., SELECT \* FROM nessie...).

* # Specifies that this catalog uses Trino's Iceberg connector.
* connector.name=iceberg
* # Crucial line: Tells the Iceberg connector to use Nessie as its metadata catalog.
* # This enables Nessie-specific features and version control.
* iceberg.catalog.type=nessie
* # The API endpoint for your Nessie server. 'nessie' is the Docker service name.
* nessie.uri=http://nessie:19120/api/v2
* # Default Nessie branch to use if not specified in a query.
* nessie.ref=main
* # S3 access key for MinIO.
* hive.s3.aws-access-key=minio
* # S3 secret key for MinIO.
* hive.s3.aws-secret-key=minio123
* # S3 endpoint for your MinIO server. 'minio' is the Docker service name.
* hive.s3.endpoint=http://minio:9000
* # Required for MinIO and most S3-compatible systems (as opposed to AWS S3 virtual-hosted style).
* hive.s3.path-style-access=true

Description: This configures Trino's Iceberg connector to use Nessie. It provides the Nessie API URI, default branch, and MinIO S3 credentials and endpoint for accessing the physical data files.

Source for connector properties: Trino Iceberg Connector Docs, Nessie Catalog Properties for Trino

1.3 docker-compose.yml

This file defines and orchestrates all the Docker services.

* version: '3.8' # Specifies the Docker Compose file format version.
* services:
* # Nessie Service: Provides Git-like version control for your data catalog.
* nessie:
* image: projectnessie/nessie:0.91.0 # Specific, tested version of Nessie.
* container\_name: nessie # Hostname for other containers to reach Nessie.
* ports:
* - "19120:19120" # Maps host port 19120 to container port 19120 (Nessie API).
* # MinIO Service: S3-compatible object storage for data files.
* minio:
* image: minio/minio:RELEASE.2024-05-25T07-23-28Z # Specific MinIO version.
* container\_name: minio # Hostname for other containers to reach MinIO.
* ports:
* - "9000:9000" # S3 API port.
* - "9001:9001" # MinIO Web Console port.
* environment:
* - MINIO\_ROOT\_USER=minio # Admin username for MinIO.
* - MINIO\_ROOT\_PASSWORD=minio123 # Admin password for MinIO.
* # Command to start MinIO server, specify data directory and console address.
* command: server /data --console-address ":9001"
* volumes:
* - minio\_data:/data # Persists MinIO data on the host machine.
* # Trino Service: Distributed SQL query engine.
* trino:
* image: trinodb/trino:452 # Specific, tested version of Trino.
* container\_name: trino # Hostname for other containers to reach Trino.
* ports:
* - "8080:8080" # Maps host port 8080 to container port 8080 (Trino UI & API).
* volumes:
* # Mounts local Trino config files into the container.
* - ./trino\_config/config.properties:/etc/trino/config.properties
* - ./trino\_config/catalog:/etc/trino/catalog
* depends\_on:
* # Ensures Nessie and MinIO start before Trino attempts to connect.
* - nessie
* - minio
* # Defines named volumes for data persistence.
* volumes:
* minio\_data: # Volume for MinIO data.

*Description:* This file defines the Nessie, MinIO, and Trino services, their Docker images, port mappings for external access, environment variables (like MinIO credentials), volume mounts for configuration and data persistence, and startup dependencies.

1.4 Starting the Infrastructure

Navigate to your my\_lakehouse\_project/ directory in your terminal and run:

* docker-compose up -d

*Description:* This command downloads the necessary Docker images (if not already present) and starts all defined services in detached mode (running in the background).

### Phase 2: Initial Verification & MinIO Setup

Confirm that the services are running and correctly configured.

2.1 Verifying Containers

Run docker ps in your terminal. You should see nessie, minio, and trino containers with a status of "Up".

2.2 Creating the MinIO Bucket

Your data lake needs a root storage location.

1. Open your web browser and go to the MinIO console: http://localhost:9001.
2. Log in with credentials minio / minio123.
3. Click on **Buckets** in the left menu.
4. Click **Create Bucket** and name it lakehouse. This bucket will store all Iceberg data files.

2.3 Verifying Trino-Nessie Connection

Use the Trino CLI to ensure Trino can communicate with Nessie.

1. Access the Trino CLI:

* docker exec -it trino trino-cli

1. Inside the trino> prompt, run:

* SHOW SCHEMAS IN nessie;

1. You should see information\_schema. This confirms Trino is successfully using the nessie catalog.
2. Type exit to leave the Trino CLI.

### Phase 3: Data Ingestion (Apache Spark)

Load raw data (e.g., a CSV file) into an Iceberg table managed by Nessie.

**Prerequisites:**

* Python 3.8+ installed on your local machine (where you'll run the Spark script).
* Java 8 or 11 installed (required by Spark).
* Install PySpark: pip install pyspark

3.1 Sample Data (raw\_users.csv)

Create a file named raw\_users.csv in your my\_lakehouse\_project/ directory:

* user\_id,signup\_date,status,email
* 1,2025-01-15,active,user1@example.com
* 2,2025-02-10,active,user2@example.com
* 3,2025-02-20,inactive,user3@example.com

3.2 Ingestion Script (ingest.py)

Create a file named ingest.py in your my\_lakehouse\_project/ directory:

* from pyspark.sql import SparkSession
* # Define versions for required Spark packages. Ensure compatibility.
* ICEBERG\_VERSION = "1.5.0"
* NESSIE\_VERSION = "0.91.0"
* AWS\_SDK\_VERSION = "2.17.230" # Check for a version compatible with your Spark and Hadoop versions
* # List of Maven coordinates for Spark to download.
* packages = [
* f"org.apache.iceberg:iceberg-spark-runtime-3.4\_2.12:{ICEBERG\_VERSION}", # Iceberg Spark runtime
* f"org.projectnessie.nessie-integrations:nessie-spark-extensions-3.4\_2.12:{NESSIE\_VERSION}", # Nessie Spark extensions
* f"software.amazon.awssdk:bundle:{AWS\_SDK\_VERSION}" # AWS SDK for S3 (MinIO) access
* ]
* # Build the SparkSession with all necessary configurations.
* spark = SparkSession.builder \
* .appName("DataIngestionToNessie") \
* .config("spark.jars.packages", ",".join(packages)) \
* .config("spark.sql.extensions", "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions,org.projectnessie.spark.extensions.NessieSparkSessionExtensions") \
* .config("spark.sql.catalog.nessie", "org.apache.iceberg.spark.SparkCatalog") \
* .config("spark.sql.catalog.nessie.catalog-impl", "org.apache.iceberg.nessie.NessieCatalog") \
* .config("spark.sql.catalog.nessie.uri", "http://localhost:19120/api/v2") \
* .config("spark.sql.catalog.nessie.ref", "main") \
* .config("spark.sql.catalog.nessie.warehouse", "s3a://lakehouse/warehouse") \
* .config("spark.hadoop.fs.s3a.endpoint", "http://localhost:9000") \
* .config("spark.hadoop.fs.s3a.access.key", "minio") \
* .config("spark.hadoop.fs.s3a.secret.key", "minio123") \
* .config("spark.hadoop.fs.s3a.path.style.access", "true") \
* .getOrCreate()
* print("Spark session created successfully with Nessie and Iceberg configurations.")
* # Read the local CSV file into a Spark DataFrame.
* # header=True: Uses the first row as column names.
* # inferSchema=True: Spark attempts to infer data types (use with caution for production).
* df = spark.read.csv("raw\_users.csv", header=True, inferSchema=True)
* print("CSV data read into DataFrame. Schema:")
* df.printSchema()
* print("Sample data:")
* df.show(5)
* # Define the target Iceberg table name (catalog.schema.table).
* # 'nessie' is the catalog name configured above.
* # 'raw\_data' will be the schema (database) name created in Nessie.
* # 'users' will be the table name.
* table\_identifier = "nessie.raw\_data.users"
* # Write the DataFrame to the Iceberg table managed by Nessie.
* # format("iceberg"): Specifies the output format.
* # mode("overwrite"): If the table exists, it will be replaced. Use "append" to add data.
* df.write.format("iceberg").mode("overwrite").save(table\_identifier)
* print(f"Successfully ingested data into Iceberg table: {table\_identifier} on Nessie branch 'main'.")
* spark.stop()
* print("Spark session stopped.")

Description: This script configures a Spark session to use Iceberg and Nessie. It defines the Nessie catalog, its URI, the default branch (main), the S3 warehouse path in MinIO, and MinIO credentials. It then reads the local CSV, and writes it as an Iceberg table named nessie.raw\_data.users.

Source for Spark Iceberg/Nessie config: Iceberg Spark Docs, Nessie Spark Docs

3.3 Running the Ingestion Script

From your my\_lakehouse\_project/ directory, run:

* spark-submit ingest.py

*Description:* spark-submit executes your PySpark application. Spark will download the specified packages and run the ingestion logic.

**3.4 Verifying Ingestion in Trino**

1. Access the Trino CLI: docker exec -it trino trino-cli
2. Run:

* SELECT \* FROM nessie.raw\_data.users;

1. You should see the data from your CSV file.

### Phase 4: dbt Core Project Setup & Transformation

Set up a dbt project to transform the raw data.

**Prerequisites:**

* Python 3.8+
* Install dbt Core and the Trino adapter: pip install dbt-core dbt-trino

**4.1 Initializing dbt Project**

1. Navigate to your my\_lakehouse\_project/ directory.
2. Create a directory for your dbt project: mkdir dbt\_transformations && cd dbt\_transformations
3. Initialize a new dbt project: dbt init my\_lakehouse\_dbt\_project
   * When prompted for database, choose trino.
4. Navigate into your new project: cd my\_lakehouse\_dbt\_project
5. (Optional Clean-up) Remove example models: rm -rf models/examples

4.2 Configuring profiles.yml for Trino-Nessie

This file tells dbt how to connect to Trino. It's located at ~/.dbt/profiles.yml (in your user's home directory). Replace its content with:

* # ~/.dbt/profiles.yml
* my\_lakehouse\_dbt\_project: # This name MUST match 'profile' in dbt\_project.yml
* target: dev # Default target to use
* outputs:
* dev:
* type: trino
* host: localhost # Trino server host (as exposed by Docker)
* port: 8080 # Trino server port
* user: dbt\_user # Arbitrary username for Trino sessions
* catalog: nessie # CRITICAL: The Trino catalog connected to Nessie
* schema: analytics\_dev\_{{ env\_var('USER', 'default') }} # Schema for dbt models (dev)
* threads: 4 # Number of parallel dbt model builds
* # CRITICAL: Specifies the Nessie branch for this dbt target.
* # This allows dbt to operate on isolated data branches.
* nessie\_ref: dev\_{{ env\_var('USER', 'default') }}
* prod: # Production target
* type: trino
* host: localhost
* port: 8080
* user: dbt\_prod\_user
* catalog: nessie
* schema: analytics # Schema for production models
* threads: 4
* nessie\_ref: main # Production target uses the 'main' Nessie branch

Description: This profile defines dev and prod targets. Both connect to the nessie Trino catalog. The nessie\_ref property is key for dbt to work with Nessie branches. The dev target dynamically creates a schema and branch name based on your system username for isolated development.

Source for dbt-trino profile config: dbt-trino Adapter Docs (or official dbt docs if available for the specific version)

4.3 Configuring dbt\_project.yml

Open dbt\_project.yml in your my\_lakehouse\_dbt\_project directory. Ensure the profile key matches your profile name. Add the flags and models configurations.

* name: 'my\_lakehouse\_dbt\_project'
* version: '1.0.0'
* config-version: 2
* profile: 'my\_lakehouse\_dbt\_project' # Matches the name in profiles.yml
* # CRITICAL: Disables view creation attempts by dbt, as Nessie catalogs
* # (via the native Trino-Nessie connector) do not support views.
* # Source: Our research and Trino GitHub Issue #17768
* flags:
* views\_enabled: false
* model-paths: ["models"]
* analysis-paths: ["analyses"]
* test-paths: ["tests"]
* seed-paths: ["seeds"]
* macro-paths: ["macros"]
* snapshot-paths: ["snapshots"]
* target-path: "target"
* clean-targets:
* - "target"
* - "dbt\_packages"
* # Default materializations for models based on their directory.
* models:
* my\_lakehouse\_dbt\_project: # Matches your project 'name'
* staging:
* # Staging models are ephemeral: their SQL is injected as CTEs
* # into downstream models, avoiding view creation.
* +materialized: ephemeral
* marts:
* # Mart models are created as physical Iceberg tables for performance.
* +materialized: table

Description: This file sets the project name, links to the profile, and crucially configures flags: views\_enabled: false and default materializations. Staging models are ephemeral to avoid view creation errors with Nessie, and mart models are table for performance.

Source for dbt configurations: dbt Project Configuration, dbt Materializations

**4.4 Creating dbt Models**

a) models/staging/sources.yml

Declare your raw data source.

* version: 2
* sources:
* - name: raw\_lakehouse\_data # Logical group name for sources
* # Schema in Nessie where the 'users' table was ingested by Spark
* schema: raw\_data
* tables:
* - name: users # Name of the raw table
* description: "Raw user data ingested from CSV via Spark."

b) models/staging/stg\_users.sql

A staging model for basic cleanup.

* -- Selects from the raw source, renames columns for clarity.
* -- As this model is ephemeral, its SQL will be a CTE in downstream models.
* select
* user\_id,
* signup\_date,
* status as user\_status,
* email
* from {{ source('raw\_lakehouse\_data', 'users') }}

c) models/marts/fct\_user\_signups.sql

A mart model for business logic.

* -- Calculates monthly active user signups.
* -- This will be created as an Iceberg table.
* select
* date\_trunc('month', signup\_date) as signup\_month,
* count(user\_id) as number\_of\_signups
* from {{ ref('stg\_users') }} -- References the ephemeral staging model
* where user\_status = 'active'
* group by 1
* order by 1

4.5 Running dbt

First, ensure your dev branch exists in Nessie. If not, create it via Trino CLI:

* # In Trino CLI (docker exec -it trino trino-cli)
* # Replace 'your\_user' with your actual system username
* CALL nessie.system.create\_branch('dev\_your\_user', 'main');

Then, from your my\_lakehouse\_dbt\_project directory, run:

* dbt run --target dev # Or just 'dbt run' if 'dev' is your default

*Description:* This command compiles your dbt models and runs them against Trino using the specified target's configuration (e.g., the dev\_your\_user Nessie branch).

**4.6 Verifying Transformations in Trino**

1. Access the Trino CLI.
2. Query your new mart table on the correct branch and schema:

* -- Replace 'your\_user' with your actual system username
* SELECT \* FROM nessie.analytics\_dev\_your\_user.fct\_user\_signups
* AT BRANCH "dev\_your\_user";

### Phase 5: (Optional) Nessie Workflow with dbt

Briefly, to promote changes to production:

1. **Develop on** dev **branch:** Make changes, run dbt run --target dev.
2. **Test Thoroughly:** Use dbt test --target dev.
3. **Merge to** main **in Nessie:**

* -- In Trino CLI
* -- Replace 'your\_user'
* CALL nessie.system.merge\_branch('dev\_your\_user', 'main');

1. **Run dbt for Production:**

* dbt run --target prod

1. This rebuilds models in the analytics schema on the main branch.

### Conclusion

You have now set up a fully functional, version-controlled data lakehouse with a dbt transformation layer. This architecture allows for robust, scalable, and auditable data pipelines. Remember that the open-source landscape evolves, so always refer to the latest official documentation for each component.

### Source Reference Summary (Key Docs)

* **Docker:**<https://www.docker.com/>
* **MinIO:**<https://min.io/docs/minio/container/index.html>
* **Nessie:**<https://projectnessie.org/>
* **Apache Iceberg:**<https://iceberg.apache.org/docs/latest/>
* **Trino:**<https://trino.io/docs/current/>
* **Apache Spark:**<https://spark.apache.org/docs/latest/>
* **dbt Core:**<https://docs.getdbt.com/>
* **dbt-trino Adapter:** (Typically found on GitHub or linked from dbt docs)