SymmetricDS Quick Start Guide

v3.6

Table of Contents

Chapter 1. Quick Start Tutorial

Now that an overview of SymmetricDS has been presented, a quick working example of SymmetricDS is in order. This section contains a hands-on tutorial that demonstrates how to synchronize two databases with a similar schema between two nodes of SymmetricDS. This example models a retail business that has a central office database (which we'll call the "root" or "corp" node) and multiple retail store databases (which we'll call the "client" or "store" nodes). For the tutorial, we will have only one "client" or store node, as shown in Figure 1.1, although by the end of the tutorial you could extend the example and configure a second store, if desired.

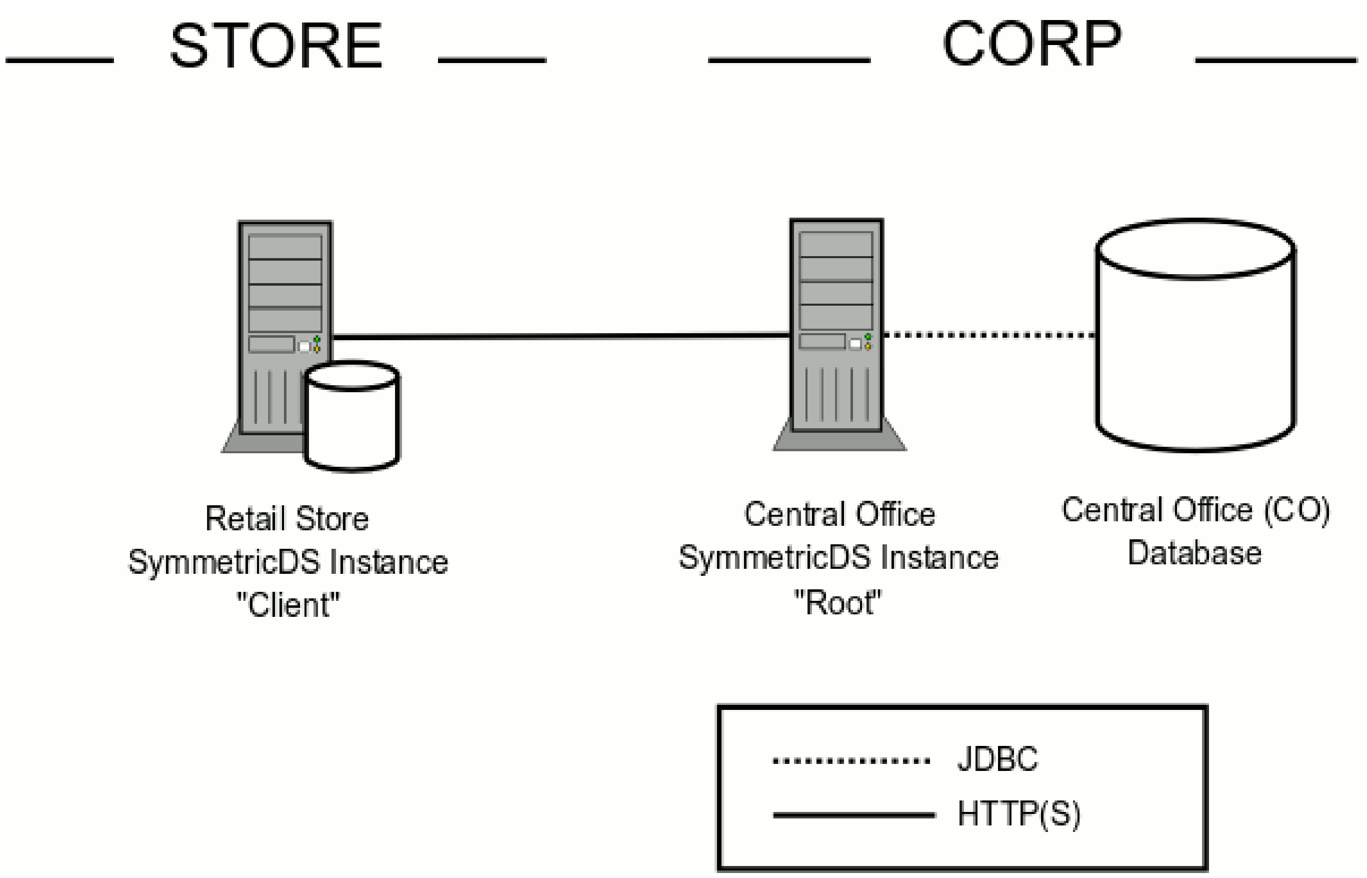
STORE
CORP
Retail Store SymmetricDS Instance "Client"
Central Office SymmetricDS Instance "Root"
Central Office (CO) Database
JDBC
HTTP(S)

Figure 1.1. Simplified Two-Tier Retail Store Tutorial Example

For this tutorial, we will install two separate copies of SymmetricDS to represent the two different servers. One will represent the store server and one will represent the corp server. Each installed copy of SymmetricDS will be responsible for one database, and thus each copy acts as a single "node" in SymmetricDS terminology. This is the most common configuration of SymmetricDS - one installed copy of the software is responsible for one single database and represents one node. (Incidentally, there is also an option to configure a single installed copy of SymmetricDS to be responsible for both nodes. This is called "multi-homing" and will be discussed at the very end of the tutorial.) Since you are most likely going to run both SymmetricDS copies on a single machine, we will run the two copies of SymmetricDS on two separate ports. We will use port 8080 for the corp server and 9090 for the store server, as shown in Figure 1.2.

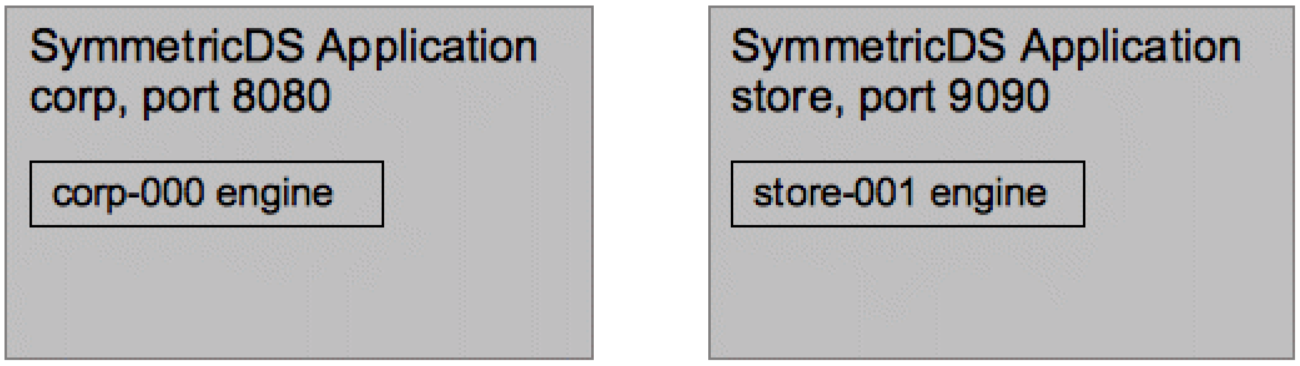

Figure 1.2. Two SymmetricDS applications - one for corp, one for store

Functionally, the corp SymmetricDS application will be responsible for capturing item data changes for the client, such as item number, description, and prices by store. The client SymmetricDS application (our store, specifically our first store, store # 001) captures sale transaction data changes for the root, such as time of sale and items sold. The pricing information is sent only to the specific store for which the price is relevant, thereby minimizing the amount of pricing data sent to each store. In other words, item pricing specific to store 001 will only be sent to the database for store 001 and not to store 002's database, for example.

The sample configuration has the client always initiating communication with the root node, which is a fairly common configuration. In this configuration, the client will attach to the root on a periodic basis to pull data from the server, and the client will also push captured changes to the root when changes are available.

Enough overview. Let's get started. We will next walk through:

1. Installing and configuring the two SymmetricDS applications,
2. Creating SymmetricDS configuration and sample tables as needed for the root and client, used to hold corp data and store data, respectively,
3. Creating sample retail data in the corp database,
4. Starting the SymmetricDS servers and registering the store with the corp node,
5. Sending an initial load of data to the store node,
6. Causing a data push and data pull operation, and
7. Verifying information about the batches that were sent and received.

1.1. Installing SymmetricDS

First, we will install two copies of the SymmetricDS software and configure it with your database connection information:

1. Download the symmetric-ds-3.x.x-server.zip file from http://www.symmetricds.org/

2. Create two directories to represent your two "machines". One will hold the corp installation of SymmetricDS and one to hold the store installation. For example, you could name the directories `sym-corp` and `sym-store001`, and we'll assume you used these names below (but feel free to update the steps below with your directory names as needed). Unzip the above zip file into both directories. This will create a `symmetric-ds-3.x.x` directory, which corresponds to the version you downloaded.

3. Properties files are use to store the minimal configuration information needed to start SymmetricDS. Copy the corp sample properties file to the corp engines directory, and the store one to the store engines directory. If you used the suggested directory names above, you would do the following copies:

   `samples/corp-000.properties` to `sym-corp/symmetric-ds-3.x.x/engines/`

   and

   `samples/store-001.properties` to `sym-store001/symmetric-ds-3.x.x/engines/`

4. Browse both properties files and explore the various settings. For exampl, notice that the root node is given a group id of corp, and that the store node is given a group id of store. Notice also that the root node is given an external id of 000, and the store node is given an external id of 001.

   Set the following properties in *both* properties files now present in the engines directories to specify how to connect to your particular database (the values below are just examples):

```
# The class name for the JDBC Driver
        db.driver=com.mysql.jdbc.Driver

        # The JDBC URL used to connect to the database
        db.url=jdbc:mysql://localhost/sample

        # The user to login as who can create and update tables
        db.user=symmetric

        # The password for the user to login as
        db.password=secret
```

5. Next, set the following property in the `store-001.properties` file to specify where the root node can be contacted:

```
# The HTTP URL of the root node to contact for registration
        registration.url=http://localhost:8080/sync/corp-000
```

Tip

Note that the URL for an engine is in the following general format:

```
http://{hostname}:{port}/sync/{engine.name}
```

where the engine.name portion of the URL comes from a node's properties file.

1.2. Creating and Populating Your Databases

Important

You must first create the databases for your root and client nodes using the administration tools provided by your database vendor. Make sure the name of the databases you create match the settings in the properties files you modified in the previous step.

First, create the sample tables in the *root* node database, load the sample data, and load the sample configuration, by doing the following:

1. Open a command prompt and navigate to the `samples` subdirectory of your *corp* SymmetricDS installation (for example, navigate to `sym-corp/symmetric-ds-3.x.x/samples`)

2. Create the sample tables for items, prices, and sales, in the root database by executing the following command:

   ../bin/dbimport --engine corp-000 --format XML create_sample.xml

   Note that the warning messages from the command are safe to ignore.

   Another quick comment about properties files. At startup, SymmetricDS looks for one or more properties files in the `engines` directory. Since we have specified a `--engine` parameter on the command line, it will look only for the specific file listed, namely `corp-000.properties`. Technically, the `--engine corp-000` part is optional in our particular tutorial example. Since there's only one properties file in the engines directory, SymmetricDS would just default to using that one file, after all. By including it, though, it will reduce errors while running the tutorial, because if you run the command from the wrong SymmetricDS installation, SymmetricDS will complain about the missing engines property file you specified.

3. Next, create the SymmetricDS-specific tables in the corp node database. These tables will contain the configuration for synchronization. The following command uses the auto-creation feature to create all the necessary SymmetricDS system tables.

   ../bin/symadmin --engine corp-000 create-sym-tables

4. Finally, load the sample item and transaction data and SymmetricDS configuration into the root node database by executing:

../bin/dbimport --engine corp-000 insert_sample.sql

Important

Please note that for MySQL, you will need to use the file `insert_sample_mysql.sql` in the above command. MySql uses back ticks (i.e., `) instead of double quotes (") for case-sensitive table and column names. The MySQL version of the file has the necessary change.

We have now created the corp database tables and populated them with our SymmetricDS configuration and sample data. Next, we will create the sample tables in the *store* node database to prepare it for receiving data.

1. Open a command prompt and navigate to the `samples` subdirectory of your *store #001* SymmetricDS installation (for example, navigate to `sym-store001/symmetric-ds-3.x.x/samples`)

2. Create the empty, sample tables in the client database by executing:

   ../bin/dbimport --engine store-001 --format XML create_sample.xml

   Note that the warning messages from the command are safe to ignore. Also, feel free to review the `create_sample.xml` file to see what it contains.

Please verify *both* databases by logging in and listing the tables.

1. Find the item tables that sync from root to client (that is, from corp to store): `item` and `item_selling_price`.

2. Find the sales tables that sync from store to corp: `sale_transaction` and `sale_return_line_item`.

3. Find the SymmetricDS system tables, which have a prefix of "sym_", such as `sym_channel`, `sym_trigger`, `sym_router`, and `sym_trigger_router`.

4. Validate the corp item tables have sample data.

1.3. Starting SymmetricDS

Database setup and configuration for the tutorial is now complete. Time to put SymmetricDS into action. We will now start both SymmetricDS nodes and observe the logging output.

1. If they are not already open, open two command prompts and navigate to the samples directory of each installed SymmetricDS application (for example, navigate to `sym-corp/symmetric-ds-3.x.x/samples` and `sym-store001/symmetric-ds-3.x.x/samples`).

2. From the corp samples directory, start the corp SymmetricDS by executing:

../bin/sym --engine corp-000 --port 8080

Upon startup for the first time, the corp node creates all the triggers that were configured by the sample configuration. It listens on port 8080 for synchronization and registration requests for the corp-000 engine.

3. From the store001 samples directory, start the store SymmetricDS by executing:

   ../bin/sym --engine store-001 --port 9090

   This command starts the store node server for the first time and uses the auto-creation feature to create the SymmetricDS system tables. It begins polling the corp node to try to register (it knows where to contact the corp node via the registration URL you configured in the previous steps). Since registration is not yet open, the store node receives an authorization failure (HTTP response of 403). We discuss registration next.

1.4. Registering a Node

When an unregistered node starts up, it will attempt to register with the node specified by the registration URL (which is our root node, in almost every case). The registration node centrally controls nodes on the network by allowing registration and returning configuration to a node once it has registered. In this tutorial, the registration node is the root node or 'corp' node, and it also participates in synchronization with other nodes.

So, we next need to open registration for the store node so that it may receive its initial load of data and so that it may receive and send data from and to the corp node. There are several ways to do this. We will use an administration feature available in SymmetricDS and issue a command on the corp node (since it is the node responsible for registration).

1. Leave the corp and store SymmetricDS applications that you started in the previous step running, and open a command prompt and navigate to corp's `samples` subdirectory of your corp SymmetricDS installation.

   Open registration for the store node server by executing:

   ../bin/symadmin --engine corp-000 open-registration store 001

   The registration is now opened for a node group called "store" with an external identifier of "001". This information matches the settings in `store-001.properties` for the store node. In SymmetricDS, each node is assigned to a node group and is given an external ID that makes sense for the application. In this tutorial, we have retail stores that run SymmetricDS, so we named our node group representing stores as "store" and we used numeric identifiers for external ids starting with "001" ("000" is used to represent the corp node). More information about node groups will be covered in the next chapter.

2. Watch the logging output of the store node to see it successfully register with the corp node. The store is configured to attempt registration at a random time interval up to every minute. Once

registered, the corp and store nodes are enabled for synchronization!

1.5. Sending an Initial Load

Next, we will send an initial load of data to our store, again using a node administration feature run on the corp node.

1. Open a command prompt and navigate to the corp `samples` subdirectory of the corp SymmetricDS installation. (Note that, in general, most system commands are issued using the corp server directly. All configuration, for example, is entered at the corp and synchronized to any clients.)

2. Send an initial load of data to the store node server by executing:

   ../bin/symadmin --engine corp-000 reload-node 001

   With this command, the server node queues up an initial load for the store node that will be sent the next time the store performs its pull. The initial load includes data for each table that is configured for synchronization (assuming its initial load order is a non-negative number, as discussed in later chapters).

3. Watch the logging output of both nodes to see the data transfer. The store is configured to pull data from the corp node every minute.

1.6. Pulling Data

Next, we will make a change to the item data in the central office corp node database (we'll add a new item), and observe the data being pulled down to the store.

1. Open an interactive SQL session with the *corp* database.

2. Add a new item for sale, with different prices at store 001 and store 002:

   insert into "item" ("item_id", "name") values (110000055, 'Soft Drink');

   insert into "item_selling_price" ("item_id", "store_id", "price") values (110000055, '001', 0.65); insert into "item_selling_price" ("item_id", "store_id", "price") values (110000055, '002', 1.00);

   Important

   Please note that for MySQL, you'll need to change the double quotes (") in the above commands to back ticks (i.e., `) since MySQL uses back ticks instead of double quotes for case-sensitive table and column names.

Once the statements are committed, the data change is captured by SymmetricDS and queued for the store node to pull.

3. Watch the logging output of both nodes to see the data transfer. The store is configured to pull data from the corp every minute.

4. Since `item_selling_price` is configured with a column match router in this tutorial, specific pricing data changes will be sent (or "routed", in SymmetricDS terms) only to nodes whose `store_id` matches the node's external ID. Verify that the new data arrives in the store database using another interactive SQL session. In this case, the first pricing row will be routed to store 001 only, and the second row would be routed to store 002 (which doesn't exist currently, so in this case the data change is recorded but routed nowhere and therefore discarded.)

1.7. Pushing Data

We will now simulate a sale at the store and observe how SymmetricDS pushes the sale transaction to the central office.

1. Open an interactive SQL session with the *store* node database.

2. Add a new sale to the store node database:

   insert into "sale_transaction" ("tran_id", "store_id", "workstation", "day", "seq") values (1000, '001', '3', '2007-11-01', 100);

   insert into "sale_return_line_item" ("tran_id", "item_id", "price", "quantity") values (1000, 110000055, 0.65, 1);

   Once the statements are committed, the data change is captured and queued for the store node to push.

3. Watch the logging output of both nodes to see the data transfer. The store is configured to push data to the corp node every minute.

1.8. Verifying Outgoing Batches

Now that we have pushed and pulled data, we will demonstrate how you can obtain information about what data has been batched and sent. A batch is used for tracking and sending one or more data changes to a given node. The sending node creates a batch and the receiving node receives and then acknowledges it.

In addition, in SymmetricDS tables are grouped into data "Channels" for, among many reasons, the purpose of allowing different types of data to synchronize even when other types of data might be in error. For example, if a batch for a given channel is in error, that batch will be retried with each synchronization for that channel until the batch is no longer in error. Only after the batch is no longer in error will additional batches for that channel be sent. In this way, the order of the data changes that have occurred for a given channel are guaranteed to be sent to the destination in the same order they occurred on the source. Batches on a channel without batch errors, however, will not be blocked by the existence of a batch in error on a different channel. In this way, data changes for one channel are not blocked by errors present in another channel.

Explore the outgoing batches by doing the following:

1. Open an interactive SQL session with either the corp or store database.

2. Verify that the data change you made was captured:

   select * from sym_data order by data_id desc;

   Each row represents a row of data that was changed. Data Ids are sequentially increasing, so one of the most recent (highest) data ids should be related to your data insert SQLs. The `event_type` is "I" for insert, "U" for update", or "D" for delete. For insert and update, the captured data values are listed in `row_data`. For update and delete, the primary key values are listed in `pk_data`.

3. Verify that the data change was included in a batch, using the data_id from the previous step:

   select * from sym_data_event where data_id = ?;

   Batches are created based on the needed routing to nodes as part of a background job, called the Route Job. As part of the Route Job, the data change is assigned to a batch using a `batch_id` which is used to track and synchronize the data. The links between batches and data are managed by this `sym_data_event` table.

4. Verify that the data change was batched, sent to the destination, and acknowledged, using the `batch_id` from the previous step:

   select * from sym_outgoing_batch where batch_id = ?;

   Batches initially have a status of "NE" when they are new and not yet sent to a node. Once a receiving node acknowledges the batch, the batch status is changed to a status of "OK" for success or "ER" for error (failure). If the batch failed, the `error_flag` on the batch is also sent to 1, since the status of a batch that failed can change as it's being retried.

Understanding these three tables, along with a fourth table discussed in the next section, is key to diagnosing any synchronization issues you might encounter. As you work with SymmetricDS, either when experimenting or starting to use SymmetricDS on your own data, spend time monitoring these tables to better understand how SymmetricDS works.

1.9. Verifying Incoming Batches

The receiving node keeps track of the batches it acknowledges and records statistics about loading the data. Duplicate batches are skipped by default, but this behavior can be changed with the `incoming.batches.skip.duplicates` runtime property.

Explore incoming batches by doing the following:

1. Open an interactive SQL session with either the corp or store database.

2. Verify that the batch was received and acknowledged, using a batch_id from the previous section:

   select * from sym_incoming_batch where batch_id = ?;

   A batch represents a collection of changes loaded by the node. The sending node that created the batch is recorded, and the batch's status is either "OK" for success or "ER" for error.

1.10. Multi-Homing

Our Quick Start Tutorial is finished. We have successfully set up and performed synchronization between two databases. However, we did want to go back and discuss one of the first steps you did in the tutorial; namely, the step where you installed two copies of SymmetricDS when doing the tutorial. Feel free to skip this section until a later time if you wish.

In the example above, we placed one properties file in the engines directory of each installed SymmetricDS application. When SymmetricDS was started in the examples above, the application initialized, and then created a "SymmetricDS engine" based on the provided property file (again, each engine serves as a SymmetricDS node and is responsible for one particular database).

In reality, though, the SymmetricDS application is capable of starting more than one engine at a time. When SymmetricDS starts, it looks in the `engines` directory for any files that end in `.properties`. It will start a SymmetricDS engine for each and every property file found. The `--engine` command line prompt is an override for this and will cause SymmetricDS to only start the one engine as specified on the command line. In cases where a single SymmetricDS application is running multiple engines, this is known as a "multi-homed" SymmetricDS application, and the feature, in general, is known as "multi-homing".

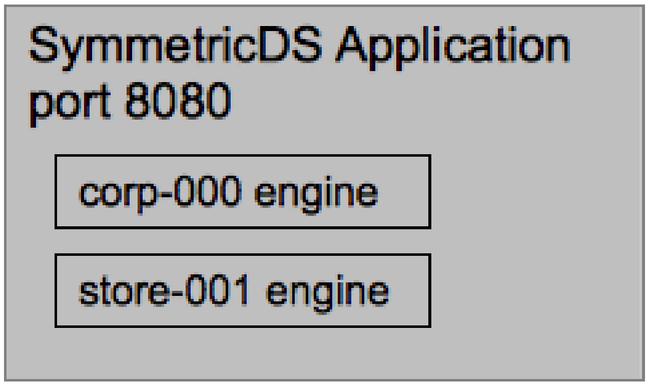

Figure 1.3. Multi-Homed version of Tutorial

So, for our tutorial above, how could we have "multi-homed" the corp and store such that we only had to install a single copy of SymmetricDS? It's fairly simple. The following changes to the above would be needed:

1. Install a single copy of the SymmetricDS software instead of two copies. You no longer need a directory to represent the two machines.

2. Instead of copying a single property file from `samples` to each separate `engines` directory, copy both files to just the one engines directory.

3. All commands in the tutorial are run from the one single `samples` directory.

4. When you start SymmetricDS, you will no longer specify a specific engine, as you want both engines to start. The command, still run from the `samples` directory, would now be:

   ../bin/sym --port 8080

   Note that we are no longer using port 9090, by the way. SymmetricDS now listens on port 8080 for traffic relevant to both the store and corp engines.

5. Other than starting the server, all other commands you executed will still have the `--engine` specification, since you are addressing the command itself to a specific node (engine) of SymmetricDS to open registration, set up the corp server to issue an initial load to store, etc.