Chapter 1. Getting Started

A NOTE FOR EARLY RELEASE READERS

This will be the 2nd chapter of the final book. Please note that the Github repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material, please reach out to the author at matt@starburstdata.com.

As we learned in Chapter 1, Presto is a powerful open source distributed SQL engine. It was designed and written from the ground up for running interactive analytic queries against disparate data sources of all sizes ranging from gigabytes to petabytes. In this chapter we will discuss how to configure and use Presto in a Docker environment, single node environment, and distributed multi node environment.

Downloading and Using Presto

Presto is an active open source projects with many frequent releases. By using the most recent version, you will be able to take advantage of the latest features, bug fixes, and performance improvements. This book will refer to and use the latest Presto version at the time of writing this book. This is the version built from the open source repository on https://github.com/prestosql/presto and available for download on Maven Central Repository (https://search.maven.org/).

If you choose a different and more recent version of Presto, it should work the same as described in this book. While it’s unlikely you’ll run into issues, it’s important to refer to the release notes and documentation for any changes.

Using Presto in a Docker Container

Starburst Data provides a preconfigured demo environment of Presto in a Docker container. To run Presto in Docker, you must have Docker Engine installed on your machine. You can download Docker Engine from https://www.docker.com.

We will use Docker to run the Presto Server and then using Docker to connect to the Presto Server using the Presto CLI.

```
$ docker run -d --name presto starburstdata/presto:302-e.7
```

Now let’s connect using the Presto CLI.

```
$ docker exec -it presto presto-cli
 presto> select count(*) from tpch.sf1.nation;
 _col0
 -------
 25
 (1 row)
 Query 20181105_001601_00002_e6r6y, FINISHED, 1 node
 Splits: 21 total, 21 done (100.00%)
 0:06 [25 rows, 0B] [4 rows/s, 0B/s]
```

NOTE

If you try to run docker and you see an error message resembling `Query 20181217_115041_00000_i6juj failed:` Presto server is still initializing, try waiting a bit and retry your last command.

To stop Presto running in a Docker container, simply run:

```
$ docker stop presto
```

NOTE

Although the docker stop command stopped the running container, it is still defined. If you try to run Presto with docker again you may get the error message `The container name “/presto" is already in use.` It’s always suggested to clean up your environment when your done by running the following command to remove the docker container when it is no longer needed.

```
$ docker rm presto
```

Detailed Installation

INSTALLING JAVA

Presto is written in Java and requires a Java JVM to be installed on your system. Presto often requires a recent update of the latest major release. For example, at the time of the writing of this book, Presto requires at least Java version 8 update 151 (Java 8u151) or Java 11. If you do not have that version of Java, Presto will fail to start. Presto typically moved to requires the recent versions of Java in order to take advantage of the new features, but more importantly performance, security, and bug fixes and improvements. You should always refer to the Presto documentation for the latest Java requirements.

Presto will run using Oracle Java and OpenJDK, but not limited to those. You can download Oracle Java from https://www.oracle.com/java/ and OpenJDK from https://openjdk.java.net/.

INSTALLING PRESTO

We will use the wget and tar commands to download and extract the Presto Server tarball on a Linux System.

```
$ wget \
http://central.maven.org/maven2/io/prestosql/presto-
server/305/



            presto-server-305.tar.gz
$ tar xvzf presto-server-305.tar.gz
```

Once you extract the Presto Server tarball, it will create a single top-level directory, presto-server-305. This directory is referred to as the *installation* directory. The *installation* directory contains:

`lib/`

This directory contains the the JARs (Java archive) that make up the Presto Server.

`plugins/`

This directory contains the Presto plugin JARs. By default Presto is packaged with many plugins, but third-party plugins may be added as well. We will cover Presto plugins in Chapter XX, but Presto allows for pluggable components to integrate with Presto such as connectors, functions, and security access controls.

`bin/`

This directory contains helper launcher scripts for Presto. These launcher scripts are use to start, stop, restart, kill and get the status a running Presto process. They also provide options to override default behaviors. We’ll discuss this in great detail in the next section.

`etc/`

This directory is the configuration directory. It is created by the user and provide the necessary configurations needed by Presto. By default, it’s assumed the configurations are located within the installation directory. But they may be placed elsewhere in the filesystem.

`var/`

Finally, there is a data directory. This is the place where logs are stored and it is created the first time Presto Server is launched. By default it’s located in the installation directly. However, it is recommended to configure it outside of the installation directory to allow for the data to be preserved across upgrades.

PRESTO CONFIGURATION

Before we can start Presto, we need to provide a set of configuration files.

- Presto Server Configuration
- Presto Catalog Configuration
- Presto Logging Configuration
- Presto Node Configuration
- Java Virtual Machine (JVM) Configuration

It’s very important to point out that these configurations must exist on every machine where Presto runs. Generally, the content in the configurations across a cluster of Presto nodes are the same. We will touch on when they are different a bit later.

By default, the configuration files are expected to be located in an `etc` directory inside the installation directory. But the defaults can be changed. This will be explained in the section detailing the Presto launcher scripts.

With the exception of the JVM configuration, the above configurations follow the Java Properties standards. As a general description for Java Properties, each configuration parameter is stored as a pair of string in the format `key=value`. If you’re interested in learning about the full specification of Java Properties, you should refer to the full Java documentation on the topic. There are more advanced features such as trimmings, escapes, line continuations, and encodings. But as a general recommendation, it’s usually best to stay to using the basic `key=value` feature.. For example, Presto will not interpret the full specification for the `node.properties` file.

Presto Server Configuration

The file, `etc/config.properties`, provides the configuration for the Presto Server. A Presto server can function as a coordinator or a worker. A Presto servier can also function as both a coordinator and worker. However, dedicating a single machine to only perform coordinator work provides the best performance on larger clusters. We will provide examples of these in the following sections on using Presto.

The following are the basic allowed Presto Server Configuration properties. In future chapters where we discuss features such as Authentication, Authorization, and Resource Groups, we will cover those additional optional properties.

`coordinator`

> Allow this Presto instance to function as a coordinator (accept queries from clients and manage query execution).

`node-scheduler.include-coordinator`

> Allow scheduling work on the coordinator. For larger clusters, processing work on the coordinator can impact query performance because the machine’s resources are not available for the critical task of scheduling, managing and monitoring query execution.

`http-server.http.port`

> Specifies the port for the HTTP server. Presto uses HTTP for all communication, internal and external.

`query.max-memory`

The maximum amount of distributed memory that a query may use. This is described in greater detail in Chapter XX about memory management in Presto.

`query.max-memory-per-node`

The maximum amount of user memory that a query may use on any one machine. This is described in greater detail in Chapter XX about memory management in Presto.

`query.max-total-memory-per-node`

The maximum amount of user and system memory that a query may use on any one machine, where system memory is the memory used during execution by readers, writers, and network buffers, etc. This is described in greater detail in Chapter XX about memory management in Presto.

`discovery-server.enabled`

Presto uses the Discovery service to find all the nodes in the cluster. Every Presto instance will register itself with the Discovery service on startup. In order to simplify deployment and avoid running an additional service, the Presto coordinator can run an embedded version of the Discovery service. It shares the HTTP server with Presto and thus uses the same port.

`discovery.uri`

The URI to the Discovery server. When running the embedded version of Discovery in the Presto coordinator, this should be the URI of the Presto coordinator. Replace example.net:8080 to match the host and port of the Presto coordinator. This URI must not end in a slash.

Presto Catalog Configuration

Presto accesses data via connectors, which are mounted as Presto catalogs. The connector provides all of the schemas and tables inside of the catalog. The Hive connector maps each Hive database to a schema. For example, let’s say we have a Hive database web that contains a table clicks. In this case, Hive connector is mounted as the `hive` catalog and the Hive `web` database is exposed as a Presto schema. The table `clicks` would be accessed in Presto as `hive.web.clicks`. This will become more obvious we we start to use Presto.

Catalogs are registered by creating a catalog properties file in the `etc/catalog` directory. For example, Presto contains a built in TPC-H connector. The TPC-H connector provides a set of schemas to support the TPC Benchmark™ H (TPC-H). TPC-H is a database benchmark used to measure the performance of highly-complex decision support databases.

This connector can also be used to test the capabilities and query syntax of Presto without configuring access to an external data source. When you query a TPC-H schema, the connector generates the data on the fly using a deterministic algorithm.

To configure the TPC-H connector, create a catalog properties file `etc/catalog/tpch.properties` with the following contents:

```
connector.name=tpch
```

In the Presto Command Line Interface section we will show how to query from this catalog in Presto.

The name of the catalog properties file will be the name of the catalog exposed in Presto. For example, if you created catalog properties files `etc/cdh-hadoop.properties`, `etc/hdp-hadoop.properties`, and `etc/mysql-dev.properties`. The catalogs exposed in Presto would be `cdh-hadoop`, `hdp-hadoop`, and `mysql-dev`.

Every catalog configuration file requires the `connector.name property`. Additional properties are determined by the Presto connector implementations. These are documented on the Presto documentation. And we will also discuss them in {{Chapter XX}} about Presto Connectors.

Presto Logging Configuration

The optional Presto logging configuration file, `etc/log.properties`, allows setting the minimum log level for named logger hierarchies. Every logger has a name, which is typically the fully qualified name of the class that uses the logger. Loggers have a hierarchy based on the dots in the name (like Java packages). For example, consider the following log levels file:

```
io.prestosql=INFO
```

This would set the minimum level to `INFO` for both io.prestosql.server and io.prestosql.hive. The default minimum level is INFO (thus the above example does not actually change anything). There are four levels: `DEBUG`, `INFO`, `WARN` and `ERROR`. Throughout the book we may refer to setting logging when discussing topics such as troubleshooting in Presto.

Presto Node Configuration

The node properties file, `etc/node.properties`, contains configuration specific to each node. A *node* is a single installed instance of Presto on a machine. The following is a minimal etc/node.properties:

```
node.environment=production
 node.id=ffffffff-ffff-ffff-ffff-ffffffffffff
 node.data-dir=/var/presto/data
```

Often this file is automatically created using a deployment system when Presto is first installed. In this chapter we are discussing how to manually deploy Presto, but there are automation tools such as Ansible that you may use to create and deploy a Presto cluster. Or automation in the Public Cloud deployments that will also create this file automatically.

The following are the allowed Presto Node Configuration properties.

`node.environment`

> The name of the environment. All Presto nodes in a cluster must have the same environment name.

`node.id`

> The unique identifier for this installation of Presto. This must be unique for every node. This identifier should remain consistent across reboots or upgrades of Presto. If running multiple installations of Presto on a single machine (i.e. multiple nodes on the same machine), each installation must have a unique identifier. Running multiple Presto instances on the same machine is generally not recommended unless you’re using it for development and testing.

`node.data-dir`

The location (filesystem path) of the data directory. By default, Presto will store logs and other data here.

Java Virtual Machine Configuration

The JVM config file, `etc/jvm.config`, contains a list of command line options used for launching the Java Virtual Machine (JVM). The format of the file is a list of options, one per line. These options are not interpreted by the shell, so options containing spaces or other special characters should not be quoted.

The following provides a good starting point for creating `etc/jvm.config`:

```
-server
 -mx16G
 -XX:+UseG1GC
 -XX:G1HeapRegionSize=32M
 -XX:+UseGCOverheadLimit
 -XX:+ExplicitGCInvokesConcurrent
 -XX:+HeapDumpOnOutOfMemoryError
 -XX:+ExitOnOutOfMemoryError
```

Because an `OutOfMemoryError` will typically leave the JVM in an inconsistent state, we write a heap dump (for debugging) and forcibly terminate the process when this occurs.

The `-mx` option is one of the more important properties in this file. It sets the maximum Heap Space for the JVM. This determines how much memory is available for the Presto process. Throughout this book we may refer back to this configuration. For example, we will discuss it in more detail in Chapter XX about Presto tuning and in Chapter XIX about Presto memory management.

Now that we have a basic understand of the Presto configuration. Let’s return back to installing and deploying Presto.

USING PRESTO ON A SINGLE MACHINE

The simplest first step to get started with Presto is to run Presto on a single machine. Inside the Presto installation directory we created in a previous section, let’s create the basic set of Presto configurations. We will use `vi` to create the configuration file. Please use the contents below for the files.

```
$ cd presto-server-305
 $ mkdir -p etc/catalog
 $ vim etc/config.properties
 $ vim etc/node.properties
 $ vim etc/jvm.config
 $ vim etc/catalog/tpch.properties
```

For the configuration files using the following content.

`etc/config.properties`

```
coordinator=true
 node-scheduler.include-coordinator=true
 http-server.http.port=8080
 query.max-memory=5GB
 query.max-memory-per-node=1GB
 query.max-total-memory-per-node=2GB
 discovery-server.enabled=true
 discovery.uri=http://localhost:8080
```

`etc/node.properties`

```
node.environment=demo
 node.id=ffffffff-ffff-ffff-ffff-ffffffffffff
 node.data-dir=/var/presto/data
```

`etc/jvm.config`

```
-server
 -Xmx4G
 -XX:+UseG1GC
 -XX:G1HeapRegionSize=32M
 -XX:+UseGCOverheadLimit
 -XX:+ExplicitGCInvokesConcurrent
 -XX:+HeapDumpOnOutOfMemoryError
 -XX:+ExitOnOutOfMemoryError
 -Djdk.nio.maxCachedBufferSize=2000000
```

`etc/catalog/tpch.properties`

```
connector.name=tpch
```

The list of commands and configuration can be found on the GitHub repository we set up for this book. You can find it here: https://www.github.com/{{TODO}}.

STARTING PRESTO

The installation directory contains a couple of launcher scripts. We will use those to start Presto.

```
$ bin/launcher run
```

This command will run Presto as a foreground process. Logs and other output to Presto are written to stdout and stderr. When Presto is started you should see this at the bottom of the output.

```
YYYY-MM-DDTHH:MM:SS INFO main
 io.prestosql.server.PrestoServer ======== SERVER STARTED
========
```

Running Presto in the foreground can be useful for quickly verifying whether the process starts up correctly and that it is using the expected configuration settings. In a production environment, you will typically run it as a background daemon process. You can do so via the following command.

```
$ bin/launcher start
 Started as 48322
```

The number `48322` you see in the example above if the process ID (pid). The number you'll see will be different.

STOPPING PRESTO

To stop Presto running as a daemon you should run the following

```
$ bin/launcher stop
 Stopped 48322
```

To forcefully stop Presto you can run the following

```
$ bin/launcher kill
 Killed 48322
```

GET STATUS

You can obtain the status of Presto by running:

```
$ bin/launcher status
 Running as 48322
```

If Presto is not running you will see this

```
$ bin/launcher status
 Not running
```

In the Presto Command Line Interface below we will explain how to connect to a Presto Server. If you wish to follow that exercise, you should keep the Presto Server running here.

PRESTO SERVER LOGS

After starting Presto as a daemon, you'll find log files in `var/log`. This will be located within the installation directly unless you specified a different location in the `etc/node.properties` file.

`launcher.log`

This log is created by the launcher and is connected to stdout and stderr streams of the server. It will contain a few log messages that occur while the server logging is being initialized and any errors or diagnostics produced by the JVM.

`server.log`

This is the main log file used by Presto. It will typically contain the relevant information if the server fails during initialization. It is automatically rotated and compressed.

`http-request.log`

This is the HTTP request log which contains every HTTP request received by the server. It is automatically rotated and compressed.

PRESTO LAUNCHER OPTIONS

The following command shows the list of options for the Presto Launcher.

```
$ bin/launcher --help
 Usage: launcher [options] command
 Commands: run, start, stop, restart, kill, status
 Options:
 -h, --help show this help message and exit
 -v, --verbose Run verbosely
 --etc-dir=DIR Defaults to INSTALL_PATH/etc
 --launcher-config=FILE
 Defaults to INSTALL_PATH/bin/launcher.properties
 --node-config=FILE Defaults to ETC_DIR/node.properties
 --jvm-config=FILE Defaults to ETC_DIR/jvm.config
 --config=FILE Defaults to ETC_DIR/config.properties
 --log-levels-file=FILE
 Defaults to ETC_DIR/log.properties
 --data-dir=DIR Defaults to INSTALL_PATH
 --pid-file=FILE Defaults to DATA_DIR/var/run/launcher.pid
 --launcher-log-file=FILE
 Defaults to DATA_DIR/var/log/launcher.log (only in daemon
mode)
 --server-log-file=FILE
 Defaults to DATA_DIR/var/log/server.log (only in daemon
mode)
 -D NAME=VALUE Set a Java system property
```

You will notice that the launcher script allows for the configuration properties to be located in different locations other than within the installation etc directory. For example, we will discuss install Presto via RPM in a following section. Using the RPM installation method will locate the configuration directly.

Using Presto on a Cluster of Machines

So far we have discussed installing Presto on a single node machine. Presto was designed and intended to be used in a distributed environment. For any real usage other than for demo purposes you will want to install Presto on a cluster of machines. Fortunately, the installation and configuration is similar to installing on a single machine. But it requires either manual installation on each machine or by using some other orchestration deployment methods.

In the Detailed Installation section we deployed a single Presto Server process to act as both a Coordinator and Worker. Here we will install and configure one Presto Coordinator and two Presto Workers. As before, we will use the `wget` and `tar` commands to download and extract the Presto Server tarball on a Linux System. You should do this on *every* machine you want to be in the Presto cluster.

```
$ wget \ https://repo1.maven.org/maven2/io/prestosql/presto-
server/305/


          presto-server-305.tar.gz
$ tar xvzf presto-server-305.tar.gz
```

As before, we need to create a set of configuration files. Inside the Presto installation directory let’s create the basic set of Presto configurations. We will use `vi` to create the configuration file. Please use the contents below for the files. You should do this on *every* machine you want to be in the Presto cluster.

```
$ cd presto-server-305
 $ mkdir -p etc/catalog
 $ vi etc/config.properties
 $ vi etc/node.properties
 $ vi etc/jvm.config
 $ vi etc/catalog/tpch.properties
```

As before, you should use the following content in the configuration files. These configuration files need to exist on *every* machine you want to be in the Presto cluster. You’ll noticed the the `etc/config.properties` is different in this distributed set up. The Coordinator will have a slightly different configuration than the Workers.

`etc/config.properties`

Coordinator

```
coordinator=true
 node-scheduler.include-coordinator=false
 http-server.http.port=8080
 query.max-memory=5GB
 query.max-memory-per-node=1GB
 query.max-total-memory-per-node=2GB
 discovery-server.enabled=true
 discovery.uri=http://<coordinator-ip-or-hostname>:8080
```

Workers

```
coordinator=false
 http-server.http.port=8080
 query.max-memory=5GB
 query.max-memory-per-node=1GB
 query.max-total-memory-per-node=2GB
 discovery.uri=http://<coordinator-ip-or-hostname>:8080
```

`etc/node.properties`

```
node.environment=demo
```

`etc/jvm.properties`

```
-server
 -Xmx8G
 -XX:+UseG1GC
 -XX:G1HeapRegionSize=32M
 -XX:+UseGCOverheadLimit
 -XX:+ExplicitGCInvokesConcurrent
 -XX:+HeapDumpOnOutOfMemoryError
 -XX:+ExitOnOutOfMemoryError
```

`etc/catalog/tpch.properties`

```
connector.name=tpch
```

The list of commands and configuration can be found on the GitHub repository we set up for this book. You can find it here: https://www.github.com/{{TODO}}.

Now that you have Presto installed and configured on a set of nodes, you should use the launcher to start Presto on *every* node. Generally, it’s best to start the Presto Coordinator first followed by the Presto Workers.

```
$ bin/launcher start
```

As before, you can use the Presto CLI to connect to the Presto Server. In the case of a distributed set up, you need to specify the address of the Presto Coordinator using the `--server` option. If you are running the Presto CLI on the Presto Coordinator node, then you do not need to specify this option, as it defaults to `localhost:8080`.

```
./presto-cli --server <coordinator-ip-or-hostname>:8080
```

You can now verify that the Presto cluster is running correctly. The nodes system table contains the list of all the active nodes that are currently part of the cluster. You can query it using the following command:

```
presto> select * from system.runtime.nodes;
 node_id | http_uri | node_version | coordinator | state
 ---------+----------------------+-------------+---------
------------
 c00367d | http://<http_uri>:8080 | 0.213 | true | active
 9408e07 | http://<http_uri>:8080 | 0.213 | false | active
 90dfc04 | http://<http_uri>:8080 | 0.213 | false | active
 (3 rows)
```

You will observe there will be one entry for every node in the Presto cluster.

Other Installation Methods

RPM Installation

Presto can also be installed via RPM Package Manager (RPM). To install Presto using the Presto RPM Package, we will use wget and rpm commands on a Linux system.

```
$ wget \ https://repo1.maven.org/maven2/io/prestosql/presto-
server-rpm/305/


            presto-server-rpm-305.rpm
 $ sudo rpm -i presto-server-rpm-305.rpm
```

This will install Presto in a single node mode and create the basic Presto configuration files and a service control script to control the Presto Server.

Control Scripts

The Presto RPM will also deploy service scripts to control the Presto server process. The script is configured with `chkconfig`, so that the service can be started automatically on the operating system boot. After installing Presto from the RPM, you can run:

```
service presto [start|stop|restart|status]
```

Installation Directory Structure

When using the RPM based installation method, Presto is installed in a directory structure more consistent with how software is installed on Linux systems. This means that not everything is contained with the single Presto installation directory structure as we have seen so far.

`/usr/lib/presto/lib/`

  The directory contains the various libraries needed to run the product. Plugins are located in a plugin subdirectory.

`/etc/presto`

  This directory contains the general Presto configuration files such as `node.properties`, jvm.config, config.properties. Catalog configurations are located in a catalog subdirectory.

`/etc/presto/env.sh`

  This directory contains the Java installation path used by Presto.

`/var/log/presto`

  This directory contains the Presto server logs.

`/var/lib/presto/data`

  This directory is the Presto data directory

`/usr/shared/doc/presto`

  This directory contains any Presto documentation.

`/etc/rc.d/init.d/presto`

  This directory contains the service scripts for controlling the Presto server process.

The `node.properties` file requires the following two additional properties since our directory structure is different from what standard Presto expects.

```
catalog.config-dir=/etc/presto/catalog
 plugin.dir=/usr/lib/presto/lib/plugin
```

Uninstall Presto

If Presto is installed using RPM, you can uninstall it the same way you remove any other RPM package

```
$ rpm -e presto
```

When removing Presto, any Presto related files and configurations will also be deleted. So it’s important you create a backup if you wish to keep anything. The Presto logs directory will not be deleted `/var/log/presto`.

Cloud Installation

In later chapters we will discuss using Presto in public clouds Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). AWS and GCP provide their own Presto distributions on AWS Elastic MapReduce (EMR) and GCP Dataproc respectively. Starburst Data provides their distribution on all three. You can also manually install Presto on the cloud providers machine instances as well using the same methods you would for an on premises installation. These options will be discussed in greater detail in following chapters in this book.