Apache Presto

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

Apache Presto is an open source distributed SQL engine. Presto originated at Facebook for data analytics needs and later was open sourced. Now, Teradata joins Presto community and offers support.

Apache Presto is very useful for performing queries even petabytes of data. Extensible architecture and storage plugin interfaces are very easy to interact with other file systems. Most of today's best industrial companies are adopting Presto for its interactive speeds and low latency performance.

This tutorial explores Presto architecture, configuration, and storage plugins. It discusses the basic and advanced queries and finally concludes with real-time examples.

Audience

This tutorial has been prepared for professionals aspiring to make a career in Big Data Analytics. This tutorial will give you enough understanding on Apache Presto.

Prerequisites

Before proceeding with this tutorial, you must have a good understanding of Core Java, DBMS and any of the Linux operating systems.

Disclaimer & Copyright

Table of Contents

1. Presto – Overview

Data analytics is the process of analyzing raw data to gather relevant information for better decision making. It is primarily used in many organizations to make business decisions. Well, big data analytics involves a large amount of data and this process is quite complex, hence companies use different strategies.

For example, Facebook is one of the leading data driven and largest data warehouse company in the world. Facebook warehouse data is stored in Hadoop for large scale computation. Later, when warehouse data grew to petabytes, they decided to develop a new system with low latency. In the year of 2012, Facebook team members designed "**Presto**" for interactive query analytics that would operate quickly even with petabytes of data.

What is Apache Presto?

Apache Presto is a distributed parallel query execution engine, optimized for low latency and interactive query analysis. Presto runs queries easily and scales without down time even from gigabytes to petabytes.

A single Presto query can process data from multiple sources like HDFS, MySQL, Cassandra, Hive and many more data sources. Presto is built in Java and easy to integrate with other data infrastructure components. Presto is powerful, and leading companies like Airbnb, DropBox, Groupon, Netflix are adopting it.

Presto – Features

Presto contains the following features:

- Simple and extensible architecture
- Pluggable connectors - Presto supports pluggable connector to provide metadata and data for queries.
- Pipelined executions - Avoids unnecessary I/O latency overhead
- User-defined functions - Analysts can create custom user-defined functions to migrate easily
- Vectorized columnar processing

Presto – Benefits

Here is a list of benefits that Apache Presto offers -

- Specialized SQL operations
- Easy to install and debug
- Simple storage abstraction
- Quickly scales petabytes data with low latency

Presto – Applications

Presto supports most of today's best industrial applications. Let's take a look at some of the notable applications.

- **Facebook**: Facebook built Presto for data analytics needs. Presto easily scales large velocity of data.
- **Teradata**: Teradata provides end-to-end solutions in Big Data analytics and data warehousing. Teradata contribution to Presto makes it easier for more companies to enable all analytical needs.
- **Airbnb**: Presto is an integral part of the Airbnb data infrastructure. Well, hundreds of employees are running queries each day with the technology.

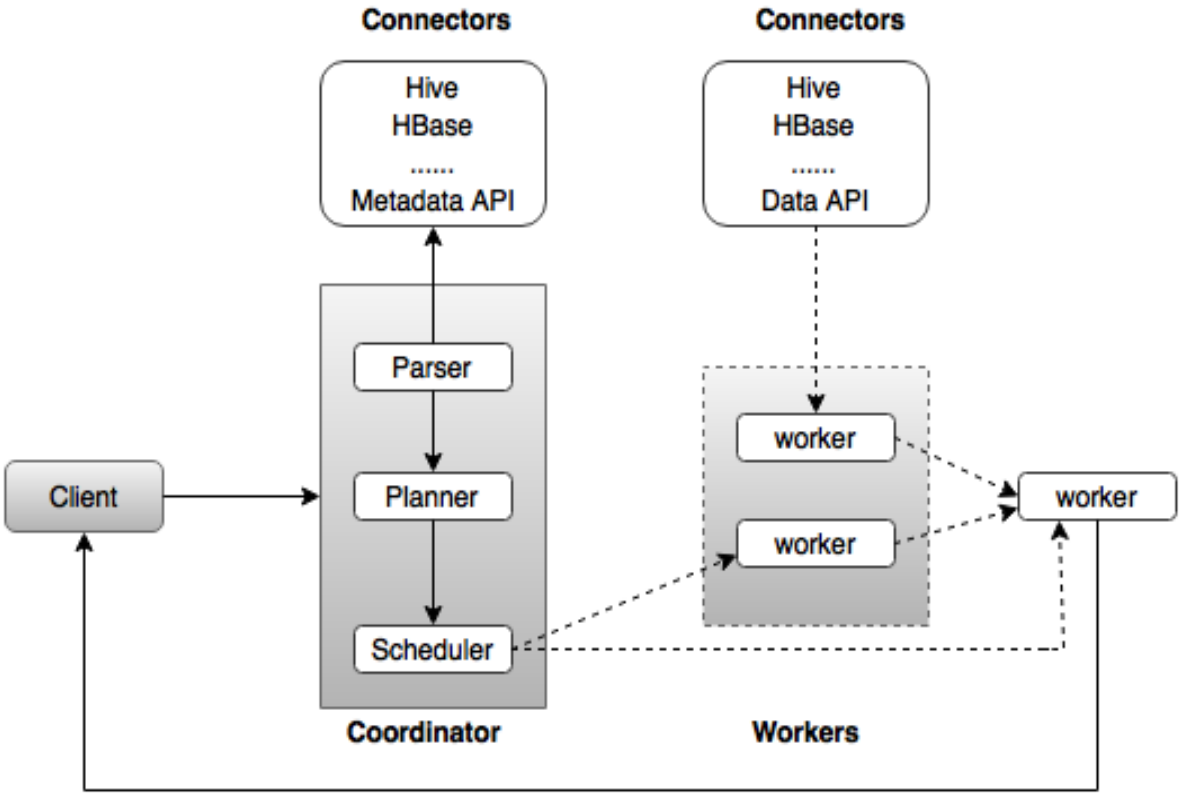
Why Presto?

Presto supports standard ANSI SQL which has made it very easy for data analysts and developers. Though it is built in Java, it avoids typical issues of Java code related to memory allocation and garbage collection. Presto has a connector architecture that is Hadoop friendly. It allows to easily plug in file systems.

Presto runs on multiple Hadoop distributions. In addition, Presto can reach out from a Hadoop platform to query Cassandra, relational databases, or other data stores. This cross-platform analytic capability allows Presto users to extract maximum business value from gigabytes to petabytes of data.

2. Presto – Architecture

The architecture of Presto is almost similar to classic MPP (massively parallel processing) DBMS architecture. The following diagram illustrates the architecture of Presto.



The above diagram consists of different components. Following table describes each of the component in detail.

Component	Description
Client	Client (Presto CLI) submits SQL statements to a coordinator to get the result.
Coordinator	Coordinator is a master daemon. The coordinator initially parses the SQL queries then analyzes and plans for the query execution. Scheduler performs pipeline execution, assigns work to the closest node and monitors progress.

Connector	Storage plugins are called as connectors. Hive, HBase, MySQL, Cassandra and many more act as a connector; otherwise you can also implement a custom one. The connector provides metadata and data for queries. The coordinator uses the connector to get metadata for building a query plan.
Worker	The coordinator assigns task to worker nodes. The workers get actual data from the connector. Finally, the worker node delivers result to the client.

Presto – Workflow

Presto is a distributed system that runs on a cluster of nodes. Presto's distributed query engine is optimized for interactive analysis and supports standard ANSI SQL, including complex queries, aggregations, joins, and window functions. Presto architecture is simple and extensible. Presto client (CLI) submits SQL statements to a master daemon coordinator.

The scheduler connects through execution pipeline. The scheduler assigns work to nodes which is closest to the data and monitors progress. The coordinator assigns task to multiple worker nodes and finally the worker node delivers the result back to the client. The client pulls data from the output process. Extensibility is the key design. Pluggable connectors like Hive, HBase, MySQL, etc., provides metadata and data for queries. Presto was designed with a "simple storage abstraction" that makes it easy to provide SQL query capability against these different kind of data sources.

Execution Model

Presto supports custom query and execution engine with operators designed to support SQL semantics. In addition to improved scheduling, all processing is in memory and pipelined across the network between different stages. This avoids unnecessary I/O latency overhead.

3. Presto – Installation

This chapter will explain how to install Presto on your machine. Let's go through the basic requirements of Presto,

- Linux or Mac OS
- Java version 8

Now, let's continue the following steps to install Presto on your machine.

Verifying Java installation

Hopefully, you have already installed Java version 8 on your machine right now, so you just verify it using the following command.

```
$ java -version
```

If Java is successfully installed on your machine, you could see the version of installed Java. If Java is not installed, follow the subsequent steps to install Java 8 on your machine.

Download JDK. Download the latest version of JDK by visiting the following link.

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

The latest version is JDK 8u 92 and the file is "jdk-8u92-linux-x64.tar.gz". Please download the file on your machine.

After that, extract the files and move to the specific directory.

Then set Java alternatives. Finally Java will be installed on your machine.

Apache Presto Installation

Download the latest version of Presto by visiting the following link,

https://repo1.maven.org/maven2/com/facebook/presto/presto-server/0.149/

Now the latest version of "presto-server-0.149.tar.gz" will be downloaded on your machine.

Extract tar Files

Extract the **tar** file using the following command:

```
$ tar  -zxf  presto-server-0.149.tar.gz
$ cd presto-server-0.149
```

Configuration Settings

Create “data” directory

Create a data directory outside the installation directory, which will be used for storing logs, metadata, etc., so that it is to be easily preserved when upgrading Presto. It is defined using the following code:

```
$ cd
$ mkdir data
```

To view the path where it is located, use the command “pwd”. This location will be assigned in the next node.properties file.

Create “etc” directory

Create an etc directory inside Presto installation directory using the following code:

```
$ cd presto-server-0.149
$ mkdir etc
```

This directory will hold configuration files. Let’s create each file one by one.

Node Properties

Presto node properties file contains environmental configuration specific to each node. It is created inside etc directory (etc/node.properties) using the following code:

```
$ cd etc
$ vi node.properties


node.environment=production
node.id=ffffffff-ffff-ffff-ffff-ffffffffffff
node.data-dir=/Users/../workspace/Presto
```

After making all the changes, save the file, and quit the terminal. Here **node.data** is the location path of the above created data directory. **node.id** represents the unique identifier for each node.

JVM Config

Create a file "jvm.config" inside etc directory (etc/jvm.config). This file contains a list of command line options used for launching the Java Virtual Machine.

```
$ cd etc
$ vi jvm.config

-server
-Xmx16G
-XX:+UseG1GC
-XX:G1HeapRegionSize=32M
-XX:+UseGCOverheadLimit
-XX:+ExplicitGCInvokesConcurrent
-XX:+HeapDumpOnOutOfMemoryError
-XX:OnOutOfMemoryError=kill -9 %p
```

After making all the changes, save the file, and quit the terminal.

Config Properties

Create a file "config.properties" inside etc directory(etc/config.properties). This file contains the configuration of Presto server. If you are setting up a single machine for testing, Presto server can function only as the coordination process as defined using the following code:

```
$ cd etc
$ vi config.properties
```

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

Here,

- **coordinator**: master node.
- **node-scheduler.include-coordinator**: Allows scheduling work on the coordinator.
- **http-server.http.port**: Specifies the port for the HTTP server.
- **query.max-memory=5GB**: The maximum amount of distributed memory.
- **query.max-memory-per-node=1GB**: The maximum amount of memory per node.
- **discovery-server.enabled**: Presto uses the Discovery service to find all the nodes in the cluster.
- **discovery.uri**: The URI to the Discovery server.

If you are setting up multiple machine Presto server, Presto will function as both coordination and worker process. Use this configuration setting to test Presto server on multiple machines.

Configuration for Coordinator

```
$ cd etc
$ vi config.properties

coordinator=true
node-scheduler.include-coordinator=false
http-server.http.port=8080
query.max-memory=50GB
query.max-memory-per-node=1GB
discovery-server.enabled=true
discovery.uri= http://localhost:8080
```

Configuration for Worker

```
$ cd etc
$ vi config.properties

coordinator=false
http-server.http.port=8080
query.max-memory=50GB
query.max-memory-per-node=1GB
discovery.uri= http://localhost:8080
```

Log Properties

Create a file "log.properties" inside etc directory(etc/log.properties). This file contains minimum log level for named logger hierarchies. It is defined using the following code:

```
$ cd etc
$ vi log.properties

com.facebook.presto=INFO
```

Save the file and quit the terminal. Here, four log levels are used such as DEBUG, INFO, WARN and ERROR. Default log level is INFO.

Catalog Properties

Create a directory "catalog" inside etc directory(etc/catalog). This will be used for mounting data. For example, create **etc/catalog/jmx.properties** with the following contents to mount the **jmx connector** as the jmx catalog:

```
$ cd etc
$ mkdir catalog
$ cd catalog
$ vi jmx.properties

connector.name=jmx
```

Start Presto

Presto can be started using the following command,

```
$ bin/launcher start
```

Then you will see the response similar to this,

```
Started as 840
```

Run Presto

To launch Presto server, use the following command:

```
$ bin/launcher run
```

After successfully launching Presto server, you can find log files in "var/log" directory.

- **launcher.log**: This log is created by the launcher and is connected to the stdout and stderr streams of the server.
- **server.log**: This is the main log file used by Presto.
- **http-request.log**: HTTP request received by the server.

As of now, you have successfully installed Presto configuration settings on your machine. Let's continue the steps to install Presto CLI.

Install Presto CLI

The Presto CLI provides a terminal-based interactive shell for running queries.

Download the Presto CLI by visiting the following link,

https://repo1.maven.org/maven2/com/facebook/presto/presto-cli/0.149/

Now "presto-cli-0.149-executable.jar" will be installed on your machine.

Run CLI

After downloading the presto-cli, copy it to the location which you want to run it from. This location may be any node that has network access to the coordinator. First change the name

of the Jar file to Presto. Then make it executable with **chmod+x** command using the following code:

```
$ mv presto-cli-0.149-executable.jar presto
$ chmod +x presto
```

Now execute CLI using the following command,

```
./presto --server localhost:8080 --catalog jmx --schema default


Here jmx(Java Management Extension) refers to catalog and default referes to
schema.
```

You will see the following response,

```
 presto:default>
```

Now type "jps" command on your terminal and you will see the running daemons.

Stop Presto

After having performed all the executions, you can stop the presto server using the following command:

```
$ bin/launcher stop
```

4. Presto – Configuration Settings

This chapter will discuss the configuration settings for Presto.

Presto Verifier

The Presto Verifier can be used to test Presto against another database (such as MySQL), or to test two Presto clusters against each other.

Create Database in MySQL

Open MySQL server and create a database using the following command.

```
create database test
```

Now you have created "test" database in the server. Create the table and load it with the following query.

```
CREATE TABLE verifier_queries(
    id INT NOT NULL AUTO_INCREMENT,
    suite VARCHAR(256) NOT NULL,
    name VARCHAR(256),
    test_catalog VARCHAR(256) NOT NULL,
    test_schema VARCHAR(256) NOT NULL,
    test_prequeries TEXT,
    test_query TEXT NOT NULL,
    test_postqueries TEXT,
    test_username VARCHAR(256) NOT NULL default 'verifier-test',
    test_password VARCHAR(256),
    control_catalog VARCHAR(256) NOT NULL,
    control_schema VARCHAR(256) NOT NULL,
    control_prequeries TEXT,
    control_query TEXT NOT NULL,
    control_postqueries TEXT,
```

```
    control_username VARCHAR(256) NOT NULL default 'verifier-test',
```

End of ebook preview

If you liked what you saw…

Buy it from our store @ https://store.tutorialspoint.com