**Presto is a distributed SQL query engine designed to query large data sets distributed over one or more heterogeneous data sources.**

## What Presto Is[#](https://prestodb.io/docs/current/overview/use-cases.html#what-presto-is)

Presto is a tool designed to efficiently query vast amounts of data using distributed queries. If you work with terabytes or petabytes of data, you are likely using tools that interact with Hadoop and HDFS. Presto was designed as an alternative to tools that query HDFS using pipelines of MapReduce jobs such as Hive or Pig, but Presto is not limited to accessing HDFS. Presto can be and has been extended to operate over different kinds of data sources including traditional relational databases and other data sources such as Cassandra.

Presto was designed to handle data warehousing and analytics: data analysis, aggregating large amounts of data and producing reports. These workloads are often classified as Online Analytical Processing (OLAP).

## Overview[#](https://prestodb.io/docs/current/overview/concepts.html#overview)

To understand Presto you must first understand the terms and concepts used throughout the Presto documentation.

While it’s easy to understand statements and queries, as an end-user you should have familiarity with concepts such as stages and splits to take full advantage of Presto to execute efficient queries. As a Presto administrator or a Presto contributor you should understand how Presto’s concepts of stages map to tasks and how tasks contain a set of drivers which process data.

This section provides a solid definition for the core concepts referenced throughout Presto, and these sections are sorted from most general to most specific.

## Server Types[#](https://prestodb.io/docs/current/overview/concepts.html#server-types)

There are three types of Presto servers: resource manager, coordinators and workers. The following section explains the difference between them.

### Resource Manager[#](https://prestodb.io/docs/current/overview/concepts.html#resource-manager)

The Presto resource manager is the server that aggregates data from all coordinators and workers and constructs a global view of the cluster. Presto installation with disaggregated coordinator must need resource manager. Clusters support multiple resource managers, each acting as a primary.

Coordinators and workers communicate with Resource managers using thrift API.

### Coordinator[#](https://prestodb.io/docs/current/overview/concepts.html#coordinator)

The Presto coordinator is the server that is responsible for parsing statements, planning queries, and managing Presto worker nodes. It is the “brain” of a Presto installation and is also the node to which a client connects to submit statements for execution. Every Presto installation must have a Presto coordinator alongside one or more Presto workers. For development or testing purposes, a single instance of Presto can be configured to perform both roles.

The coordinator keeps track of the activity on each worker and coordinates the execution of a query. The coordinator creates a logical model of a query involving a series of stages which is then translated into a series of connected tasks running on a cluster of Presto workers.

Coordinators communicate with workers and clients using a REST API.

### Worker[#](https://prestodb.io/docs/current/overview/concepts.html#worker)

A Presto worker is a server in a Presto installation which is responsible for executing tasks and processing data. Worker nodes fetch data from connectors and exchange intermediate data with each other. The coordinator is responsible for fetching results from the workers and returning the final results to the client.

When a Presto worker process starts up, it advertises itself to the discovery server in the coordinator, which makes it available to the Presto coordinator for task execution.

Workers communicate with other workers and Presto coordinators using a REST API.

## Data Sources[#](https://prestodb.io/docs/current/overview/concepts.html#data-sources)

Throughout this documentation, you’ll read terms such as connector, catalog, schema, and table. These fundamental concepts cover Presto’s model of a particular data source and are described in the following section.

## Query Execution Model[#](https://prestodb.io/docs/current/overview/concepts.html#query-execution-model)

Presto executes SQL statements and turns these statements into queries that are executed across a distributed cluster of coordinator and workers.

### Statement[#](https://prestodb.io/docs/current/overview/concepts.html#statement)

Presto executes ANSI-compatible SQL statements. When the Presto documentation refers to a statement, it is referring to statements as defined in the ANSI SQL standard which consists of clauses, expressions, and predicates.

Some readers might be curious why this section lists separate concepts for statements and queries. This is necessary because, in Presto, statements simply refer to the textual representation of a SQL statement. When a statement is executed, Presto creates a query along with a query plan that is then distributed across a series of Presto workers.

### Query[#](https://prestodb.io/docs/current/overview/concepts.html#query)

When Presto parses a statement, it converts it into a query and creates a distributed query plan which is then realized as a series of interconnected stages running on Presto workers. When you retrieve information about a query in Presto, you receive a snapshot of every component that is involved in producing a result set in response to a statement.

The difference between a statement and a query is simple. A statement can be thought of as the SQL text that is passed to Presto, while a query refers to the configuration and components instantiated to execute that statement. A query encompasses stages, tasks, splits, connectors, and other components and data sources working in concert to produce a result.

—---------------------------------------------------------------------------------------------------

## **Installing Presto**[**#**](https://prestodb.io/docs/current/installation/deployment.html#installing-presto)

Download the Presto server tarball, [presto-server-0.277.tar.gz](https://repo1.maven.org/maven2/com/facebook/presto/presto-server/0.277/presto-server-0.277.tar.gz), and unpack it. The tarball will contain a single top-level directory, presto-server-0.277, which we will call the *installation* directory.

Presto needs a *data* directory for storing logs, etc. We recommend creating a data directory outside of the installation directory, which allows it to be easily preserved when upgrading Presto.

## Configuring Presto[#](https://prestodb.io/docs/current/installation/deployment.html#configuring-presto)

Create an etc directory inside the installation directory. This will hold the following configuration:

* **Node Properties**: environmental configuration specific to each node
* **JVM Config**: command line options for the Java Virtual Machine
* **Config Properties**: configuration for the Presto server. See the [Properties Reference](https://prestodb.io/docs/current/admin/properties.html) for available configuration properties.
* **Catalog Properties**: configuration for [Connectors](https://prestodb.io/docs/current/connector.html) (data sources). The available catalog configuration properties for a connector are described in the respective connector documentation.

### Node Properties[#](https://prestodb.io/docs/current/installation/deployment.html#node-properties)

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. This file is typically created by the deployment system when Presto is first installed. The following is a minimal etc/node.properties:

node.environment=production

node.id=ffffffff-ffff-ffff-ffff-ffffffffffff

node.data-dir=/var/presto/data

The above properties are described below:

* 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.
* node.data-dir: The location (filesystem path) of the data directory. Presto will store logs and other data here.

### JVM Config[#](https://prestodb.io/docs/current/installation/deployment.html#jvm-config)

The JVM config file, etc/jvm.config, contains a list of command line options used for launching the Java Virtual Machine. 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

-Xmx16G

-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.

### Config Properties[#](https://prestodb.io/docs/current/installation/deployment.html#config-properties)

The config properties file, etc/config.properties, contains the configuration for the Presto server. Every Presto server can function as both a coordinator and a worker, but dedicating a single machine to only perform coordination work provides the best performance on larger clusters.

The following is a minimal configuration for the coordinator:

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://example.net:8080

And this is a minimal configuration for the workers:

coordinator=false

http-server.http.port=8080

query.max-memory=50GB

query.max-memory-per-node=1GB

discovery.uri=http://example.net:8080

Alternatively, if you are setting up a single machine for testing that will function as both a coordinator and worker, use this configuration:

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://example.net:8080

If single coordinator is not sufficient, disaggregated coordinator setup can be used which supports multiple coordinator using below minimal configuration:

* Resource Manager

Minimum 1 resource manager is needed for a cluster and more can be added in to the cluster with each behaving as primary.

resource-manager=true

resource-manager-enabled=true

coordinator=false

node-scheduler.include-coordinator=false

http-server.http.port=8080

thrift.server.port=8081

query.max-memory=50GB

query.max-memory-per-node=1GB

discovery-server.enabled=true

discovery.uri=http://example.net:8080 (Point to resource manager host/vip)

thrift.server.ssl.enabled=true

* Coordinator

Cluster supports pool of coordinators. Each coordinator will run subset of queries in a cluster.

coordinator=true

node-scheduler.include-coordinator=false

http-server.http.port=8080

query.max-memory=50GB

query.max-memory-per-node=1GB

discovery.uri=http://example.net:8080 (Point to resource manager host/vip)

resource-manager-enabled=true

* Worker

Cluster supports a pool of workers. They send their heartbeats to the resource manager.

coordinator=false

http-server.http.port=8080

query.max-memory=50GB

query.max-memory-per-node=1GB

discovery.uri=http://example.net:8080 (Point to resource manager host/vip)

resource-manager-enabled=true

**These properties require some explanation:**

* resource manager: Aggregates data from coordinators and workers and constructs a global view of the cluster. For more details read the [concepts](https://prestodb.io/docs/current/overview/concepts.html#resource-manager).
* 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.
* query.max-memory-per-node: The maximum amount of user memory that a query may use on any one machine.
* 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. Because we have enabled 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.

The following **flags can help one tune the disaggregated coordinator cluster’s resource groups** to the desired consistency:

* concurrency-threshold-to-enable-resource-group-refresh (default: 1.0)  
  Configure coordinator to wait for the next resource group update before allowing more queries to run on any given resource group, if running queries reached the configured limit.  
  Default value is 1.0. It means once any resource group is running its max allowed queries, the coordinator has to wait for an update from the resource manager before allowing new queries to run on the given resource group. To achieve stronger consistency, reduce the percentage to lower value.
* resource-group-runtimeinfo-refresh-interval (default: 100 ms)  
  This configuration helps tune coordinator periodic polling intervals of cluster level resource group usage from the resource manager.

You may also wish to set the following properties:

* jmx.rmiregistry.port: Specifies the port for the JMX RMI registry. JMX clients should connect to this port.
* jmx.rmiserver.port: Specifies the port for the JMX RMI server. Presto exports many metrics that are useful for monitoring via JMX.

See also [Resource Groups](https://prestodb.io/docs/current/admin/resource-groups.html).

### **Log Levels**[**#**](https://prestodb.io/docs/current/installation/deployment.html#log-levels)

The optional log levels 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:

com.facebook.presto=INFO

This would set the minimum level to INFO for both com.facebook.presto.server and com.facebook.presto.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.

### Catalog Properties[#](https://prestodb.io/docs/current/installation/deployment.html#catalog-properties)

Presto accesses data via *connectors*, which are mounted in catalogs. The connector provides all of the schemas and tables inside of the catalog. For example, the Hive connector maps each Hive database to a schema, so if the Hive connector is mounted as the hive catalog, and Hive contains a table clicks in database web, that table would be accessed in Presto as hive.web.clicks.

Catalogs are registered by creating a catalog properties file in the etc/catalog directory. For example, create etc/catalog/jmx.properties with the following contents to mount the jmx connector as the jmx catalog:

connector.name=jmx

See [Connectors](https://prestodb.io/docs/current/connector.html) for more information about configuring connectors.

## **Running Presto**[**#**](https://prestodb.io/docs/current/installation/deployment.html#running-presto)

The installation directory contains the launcher script in bin/launcher. Presto can be started as a daemon by running the following:

bin/launcher start

Alternatively, it can be run in the foreground, with the logs and other output being written to stdout/stderr (both streams should be captured if using a supervision system like daemontools):

bin/launcher run

Run the launcher with --help to see the supported commands and command line options. In particular, the --verbose option is very useful for debugging the installation.

After launching, you can find the log files in var/log:

* launcher.log: This log is created by the launcher and is connected to the 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.

—---------------------------------------------------------------------------------------------------

# **Command Line Interface**[#](https://prestodb.io/docs/current/installation/cli.html#installation-cli--page-root)

The Presto CLI provides a terminal-based interactive shell for running queries. The CLI is a [self-executing](http://skife.org/java/unix/2011/06/20/really_executable_jars.html) JAR file, which means it acts like a normal UNIX executable.

Download [presto-cli-0.277-executable.jar](https://repo1.maven.org/maven2/com/facebook/presto/presto-cli/0.277/presto-cli-0.277-executable.jar), rename it to presto, make it executable with chmod +x, then run it:

./presto --server localhost:8080 --catalog hive --schema default