
## Overview

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

# Use Cases

This section puts Presto into perspective so that prospective administrators and end users know what to expect from Presto.

## What Presto Is Not

Since Presto is being called a *database* by many members of the community, it makes sense to begin with a definition of what Presto is not.

Do not mistake the fact that Presto understands SQL with it providing the features of a standard database. Presto is not a general-purpose relational database. It is not a replacement for databases like MySQL, PostgreSQL or Oracle. Presto was not designed to handle Online Transaction Processing (OLTP). This is also true for many other databases designed and optimized for data warehousing or analytics.

## 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).

## Who uses Presto?

Presto is an open source project that operates under the governance of the Presto Foundation, which is part of the **Linux Foundation**. Presto was invented at Meta, and continues to be developed by community members at Meta, Bytedance, IBM, Uber, Twitter, and elsewhere.

# Presto Concepts



## 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

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

### 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

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

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

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.

### Connector

A connector adapts Presto to a data source such as Hive or a relational database. You can think of a connector the same way you think of a driver for a database. It is an implementation of Presto's **SPI** which allows Presto to interact with a resource using a standard API.

Presto contains several built-in connectors: a connector for **JMX**, a **System** connector which provides access to built-in system tables, a **Hive** connector, and a **TPCH** connector designed to serve TPC-H benchmark data. Many third-party developers have contributed connectors so that Presto can access data in a variety of data sources.

Every catalog is associated with a specific connector. If you examine a catalog configuration file, you will see that each contains a mandatory property `connector.name` which is used by the catalog manager to create a connector for a given catalog. It is possible to have more than one catalog use the same connector to access two different instances of a similar database. For example, if you have two Hive clusters, you can configure two catalogs in a single Presto cluster that both use the Hive connector, allowing you to query data from both Hive clusters (even within the same SQL query).

### Catalog

A Presto catalog contains schemas and references a data source via a connector. For example, you can configure a JMX catalog to provide access to JMX information via the JMX connector. When you run a SQL statement in Presto, you are running it against one or more catalogs. Other examples of catalogs include the Hive catalog to connect to a Hive data source.

When addressing a table in Presto, the fully-qualified table name is always rooted in a catalog. For example, a fully-qualified table name of `hive.test_data.test` would refer to the `test` table in the `test_data` schema in the `hive` catalog.

Catalogs are defined in properties files stored in the Presto configuration directory.

### Schema

Schemas are a way to organize tables. Together, a catalog and schema define a set of tables that can be queried. When accessing Hive or a relational database such as MySQL with Presto, a schema translates to the same concept in the target database. Other types of connectors may choose to organize tables into schemas in a way that makes sense for the underlying data source.

### Table

A table is a set of unordered rows which are organized into named columns with types. This is the same as in any relational database. The mapping from source data to tables is defined by the connector.

## 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

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

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.

### Stage

When Presto executes a query, it does so by breaking up the execution into a hierarchy of stages. For example, if Presto needs to aggregate data from one billion rows stored in Hive, it does so by creating a root stage to aggregate the output of several other stages all of which are designed to implement different sections of a distributed query plan.

The hierarchy of stages that comprises a query resembles a tree. Every query has a root stage which is responsible for aggregating the output from other stages. Stages are what the coordinator uses to model a distributed query plan, but stages themselves don't run on Presto workers.

### Task

As mentioned in the previous section, stages model a particular section of a distributed query plan, but stages themselves don't execute on Presto workers. To understand how a stage is executed, you'll need to understand that a stage is implemented as a series of tasks distributed over a network of Presto workers.

Tasks are the “work horse” in the Presto architecture as a distributed query plan is deconstructed into a series of stages which are then translated to tasks which then act upon or process splits. A Presto task has inputs and outputs, and just as a stage can be executed in parallel by a series of tasks, a task is executing in parallel with a series of drivers.

## Split

Tasks operate on splits which are sections of a larger data set. Stages at the lowest level of a distributed query plan retrieve data via splits from connectors, and intermediate stages at a higher level of a distributed query plan retrieve data from other stages.

When Presto is scheduling a query, the coordinator will query a connector for a list of all splits that are available for a table. The coordinator keeps track of which machines are running which tasks and what splits are being processed by which tasks.

## Driver

Tasks contain one or more parallel drivers. Drivers act upon data and combine operators to produce output that is then aggregated by a task and then delivered to another task in another stage. A driver is a sequence of operator instances, or you can think of a driver as a physical set of operators in memory. It is the lowest level of parallelism in the Presto architecture. A driver has one input and one output.

## Operator

An operator consumes, transforms and produces data. For example, a table scan fetches data from a connector and produces data that can be consumed by other operators, and a filter operator consumes data and produces a subset by applying a predicate over the input data.

## Exchange #

Exchanges transfer data between Presto nodes for different stages of a query. Tasks produce data into an output buffer and consume data from other tasks using an exchange client.

# PRESTO AND HADOOP

## How does Presto work with Hadoop?

At a high level, a query engine is a piece of software that sits on top of a database or server and executes queries against data in that database or server to provide answers for users or applications.

Presto's distributed SQL query engine runs on Hadoop. Its architecture consists of one coordinator node that works together with multiple worker nodes. Presto does not have its own storage system so it is a good complement to Hadoop/HDFS.

## How is Hadoop related to Hive?

Apache Hive was the original SQL query engine developed at Facebook, built on top of Hadoop. You can query data stored in various databases and file systems that integrate with Hadoop. Hive makes it easy to perform batch SQL queries on large amounts of unstructured data. As compared to Presto, Hive is optimized for query throughput while Presto is optimized for latency. Hive is better suited for queries that require a large amount of memory.

# INTRODUCTION TO PRESTO AND COMMONLY ASKED QUESTIONS

Presto with Alluxio brings together two open source technologies to give you better performance and multi-cloud capabilities for interactive analytic workloads. Presto's open source distributed SQL query engine coupled with Alluxio enables true separation of storage and compute for data locality and provides memory speed response time and aggregate data from any file or object store.

## What Is Presto And What Are Its Use Cases?

Presto was initially designed at Facebook as they needed to run interactive queries against large data warehouses in Hadoop. It was explicitly designed to fill the gap/need to be able to run fast queries against data warehouses storing petabytes of data.

Presto is a SQL based querying engine that uses an MPP architecture to scale out. Because it is a querying engine only, it separates compute and storage relying on connectors to integrate with other data sources to query against. In this capacity, it excels against other technologies in the space providing the ability to query against:

- Traditional Databases
  - MySQL
  - PostGres
  - SQL Server
- Non-relational Databases
  - Mongodb
  - Redis
  - Cassandra
- Columnar file formats like ORC, Parquet and Avro – stored on:
  - Amazon S3
  - Google Cloud Store
  - Azure Blog Store
  - HDFS
  - Clustered file systems

You can learn how to deploy Presto within their documentation, and read more on Presto architecture within the other sections of our learning center.

## What Is A Presto Database?

Presto's distributed system runs on Hadoop and uses a classic massively parallel processing (MPP) database management system (you might hear some people call it PrestoDB). It has one coordinator node (master) working in synch with multiple other workers. After users submit their SQL query through a client to the Presto coordinator, it uses a custom query engine to parse, plan and schedule a distributed query plan across all its worker nodes. Presto is built with a familiar SQL query interface that allows you to easily run interactive SQL on Hadoop. It supports standard ANSI SQL semantics, including complex queries, aggregations, and joins.

## How Does Presto Cache And Store Data?

Presto stores intermediate data during the period of tasks in its buffer cache. However, it is not meant to serve as a caching solution or a persistent storage layer. It is primarily designed to be a query execution engine that allows you to query against other disparate data sources.

This is where a technology like Alluxio might help. Alluxio provides a multi-tiered layer for Presto caching and connects to a variety of storage systems and clouds so Presto can query data stored anywhere.

## What Is Apache Presto?

Presto or PrestoDB is a distributed SQL query engine that is used best for running interactive analytic workloads in your big data environment. Presto allows you to query against many different data sources whether its HDFS, MySQL, Cassandra, or Hive. Presto is built on Java and can also integrate with other third party data sources or infrastructure components.

The Presto query execution model is split up into a few different phases: Statement, Query, Stage, Task, and Splits. This article will briefly discuss each to explain what Presto is and what it is not. After you issue a SQL query (or Statement) to the query engine, it parses and converts it to a query. When Presto executes the query it does so by breaking it up into multiple stages. Stages are then split up into tasks across the multiple Presto workers. Think of tasks as the ones that are essentially doing the work and processing. Tasks use an Exchange in order to share data between tasks and outputs of processes.

## Is Presto In-Memory?

Memory used by Presto is usually in the context of the JVMs itself, depending on query sizes and complexity of tasks you can allocate more or less memory to the JVMs. Presto itself, however, doesn't use this memory to cache any data.

## Does Presto Use MapReduce?

Similarly, some users are used to Hive's execution model that breaks down a query through MapReduce to work on constituent data in HDFS. Presto will leverage its own mechanism to break down and fan out the work of a given query and it does not rely on MapReduce to do so.

## What Is Presto In Big Data?

Big data is a broad term. Generically, it applies to ways to analyze, extract and deal with data sets that are too large or complex to be dealt with by traditional data processing application software. Challenges in this space include:

- Capturing data
- Storing data
- Analysis
- Search
- Sharing
- Transfer
- Visualization
- Querying
- Updating

Apache Presto/Starburst Presto falls into the querying vertical of big data. Competitors in the space also include technologies like Hive, Pig, Hbase, Druid, Dremio, Impala, Spark SQL.

Many of the technologies in the querying vertical of big data are designed within or to work directly against the Hadoop ecosystem.

## What Is The Difference Between Presto, PrestoSQL And Starburst Presto?

Presto originated from Facebook and was built specifically for Facebook. PrestoSQL is backed by the Presto foundation, who made it more broad for wider adoption. The Presto distribution from Starburst is even more optimized with enterprise features like the cost-based optimizer.

## What Is Presto Hive?

Presto Hive typically means Presto with the Hive connector. The connector allows querying of data that is stored in a Hive data warehouse. Hive is a combination of data files and metadata. The data files themselves can be of different formats and typically are stored in an HDFS or S3-type system. The metadata is information about the data files and how they are mapped to schemas and tables. This data is stored in a database such as MySQL and accessed via the Hive metastore service. Presto via the Hive connector is able to access both these components.

One thing to note is that Hive also has its own query execution engine. Thus, there's a difference between running a Presto query against a Hive-defined table and running the same query directly though the Hive CLI.

## Does Presto Use Spark?

Presto and Spark are two different query engines. You can read more details about the differences in this Quora thread, but at a high level Spark supports complex/long running queries while Presto is better for short interactive queries.

## Does Presto Cache Data?

Out of the box, no. This is where a technology like Alluxio comes in. Alluxio provides a read/write block-level caching engine that connects to a variety of storage systems including S3 and HDFS. You can read more in this blog.

## What Is Teradata Presto?

Teradata distributes open-source Presto and works closely with Starburst Data which also provides enterprise distribution and support of Presto. Both companies support open-source Presto and its community.

## Does Presto Use YARN?

Presto is not dependent on Yarn as a resource manager. Instead it leverages a very similar architecture with dedicated Coordinator and Worker nodes that are not dependent on a Hadoop infrastructure to be able to run.

## Presto + Alluxio

Presto with Alluxio is a truly separated compute and storage stack, enabling interactive big data analytics on any file or object store.

Alluxio provides a multi-tiered layer for Presto caching, enabling consistent high performance with jobs that run up to 10x faster.

Alluxio also makes the important data local to Presto, so there are no copies to manage (and lower costs).

And last, Alluxio connects to a variety of storage systems and clouds so Presto can query data stored anywhere.

# AN INTRODUCTION TO THE PRESTO ARCHITECTURE

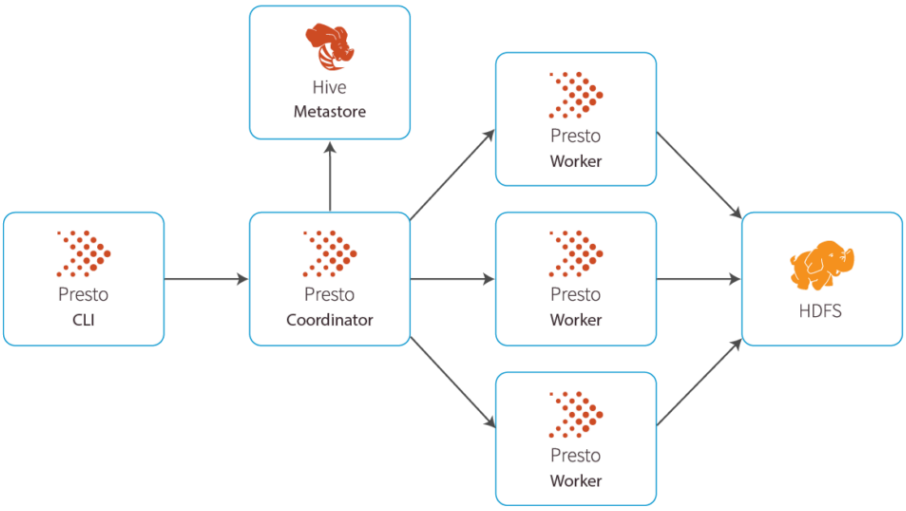
## What is the Presto architecture?

Presto is an open source distributed system that can run on multiple machines. Its distributed SQL query engine was built for fast analytic queries. A typical Presto deployment will include one Presto Coordinator and any number of Presto Workers.

Presto Coordinator: Used to submit queries and manages parsing, planning, and scheduling query execution across Presto Workers

Presto Worker: Processes the queries, adding more workers gives you faster query processing

At a high level, the Presto architecture looks something like this:




## Presto components

Diving a little deeper into the Presto components, let's start with the server types (Coordinator and Worker).

Coordinator: The Presto Coordinator is truly the brain of any Presto installation and every installation requires it. It parses statements, plans queries, and manages Presto worker nodes, and it tracks all the Workers' activity to coordinate queries. It gets results from the Workers and returns final results back to the client. Coordinators connect with workers and clients via REST.

Worker: The Presto Worker executes tasks and processes data. These nodes share data amongst each other and get data from the Coordinator. When you first start up a Presto Worker, it will be identified by the Coordinator and make itself available for task execution.

When it comes to managing the data itself, Presto has several important components that enable this.

Catalog: Presto Catalogs contain the information about where data is located – they contain schemas and the data source. When users run a SQL statement in Presto, it means they're running it against one or more catalogs. For example, you may build a Hive catalog to access Hive information from the Hive data source. Catalogs are defined in properties files stored in the Presto configuration directory.

Tables and schemas: If you're familiar with relational databases, it's the same concept. A table is a set of unordered rows of data that can be organized into named columns/types. Schema is what you use to organize your tables. Catalogs and schemas are how users define what will be queried.

Connector: Connectors are used to integrate Presto with external data sources like object stores, relational databases, or Hive. You can integrate with these sources using standard APIs via Presto's SPI implementation. Presto has over 20 built-in connectors for various data sources. Every Presto Catalog is associated with a specific connector, and more than one catalog can use the same connector to access different instances/clusters of the same data source.

Last, when it comes to running Presto queries there are some specific components to know.

Statements and Queries: Presto executes ANSI-compatible SQL statements. When a statement is executed, Presto creates a query along with a query plan that is then distributed across a series of Presto workers. When Presto parses a statement, it converts it into a query and creates a distributed query plan among Presto workers. A statement is simply passing along the instructions while the query is actually executing it.

Stage: To execute a query, Presto breaks it up into stages. Depending on how much data it needs to aggregate, there may be several stages that implement different sections of the query. Stages are typically done in a hierarchical manner (it may look like a tree). Every query has a “roots” stage which aggregates all the data from other stages. Keep in mind the stages themselves don’t run on Presto workers, they may run on the database underneath (this is called push-down).

Task: Stages (from above) are implemented as a series of tasks that may be distributed over a network of Presto workers. Tasks have inputs and outputs and are executed in parallel with a series of drivers.

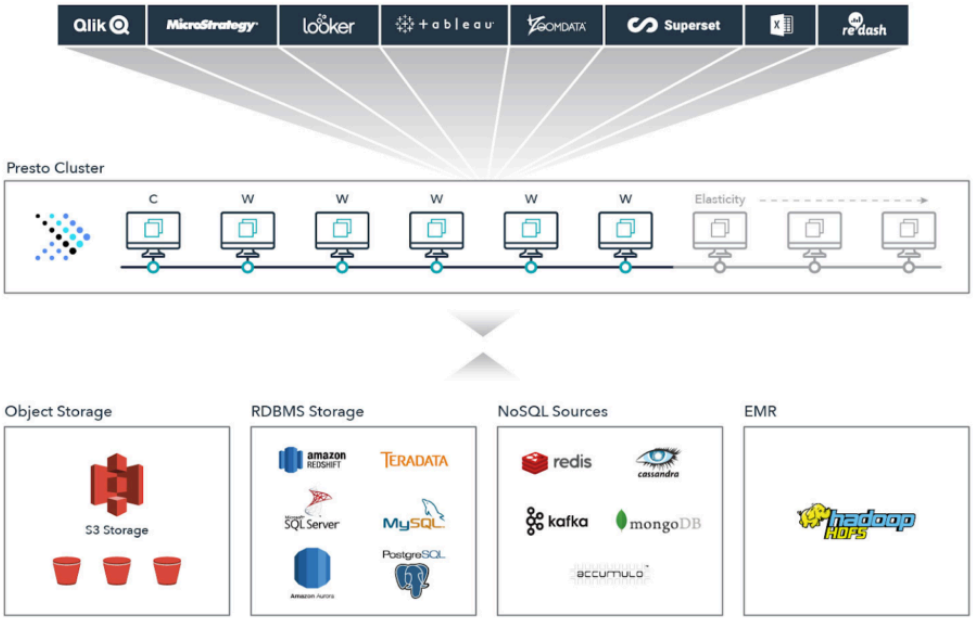
Split: Splits are sections of larger data sets and how tasks operate. When Presto schedules a query, the Coordinator keeps track of which machines are running tasks and what splits are being processed by tasks.

Drivers and Operators: Tasks contain one or more parallel drivers and they are operators in memory. An operator consumes, transforms and produces data.

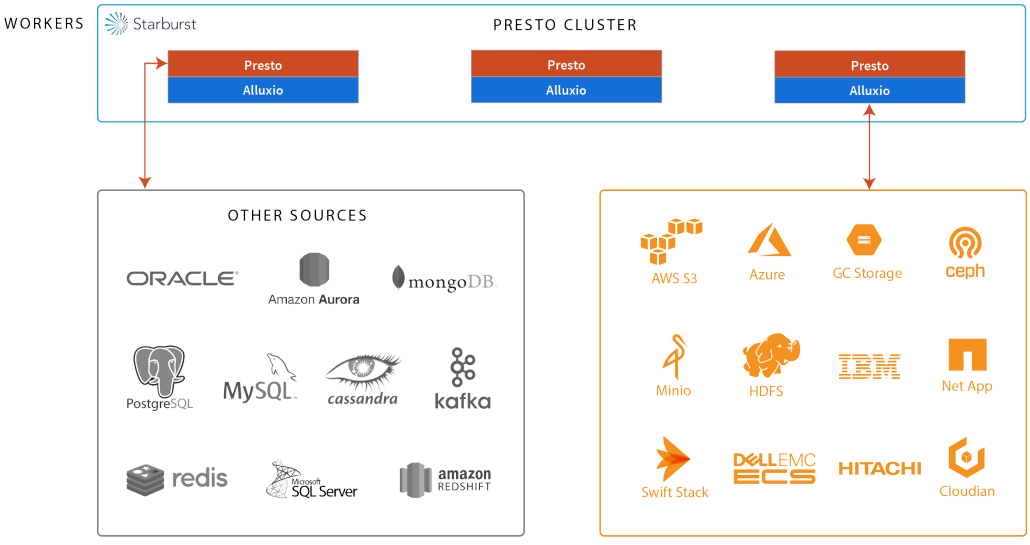
Exchange: Exchanges transfer data between Presto nodes for different stages of a query. Tasks produce data into an output buffer and consume data from other tasks using an exchange client.

## Deployments, Configurations & Installations

In practice, you might deploy Presto in the cloud or on-prem. Presto sits between your BI tool and your storage. Here’s an example taken from Starburst, the company behind Presto:




Companies use Alluxio with Presto to speed up Presto performance.



Alluxio provides a multi-tiered layer for Presto caching, enabling consistent high performance with jobs that run up to 10x faster, makes the important data local to Presto, so there are no copies to manage (and lower costs), and connects to a variety of storage systems and clouds so Presto can query data stored anywhere.

# WHAT IS A QUERY ENGINE?

At a high level, a query engine is a piece of software that sits on top of a database or server and executes queries against data in that database or server to provide answers for users or applications.

More specifically, a SQL query engine interprets SQL commands and language to access data in relational systems. Many use SQL query engines for CRUD (create, read, update, delete) operations and enforce data policies that relational data models and database management systems require.

There are many benefits of using a query engine. For starters, you can bring a query engine to your data instead of having to move your data somewhere else. A distributed SQL query engine will allow you to query data from a variety of data sources like Hadoop, AWS S3, NoSQL, MySQL, and more, or data from multiple data sources within a single query.

Examples of query engines include Presto, Apache Drill, Cloudera Impala, and Apache Spark.

# What is Presto or PrestoDB?

Presto (or PrestoDB) is an open source, distributed SQL query engine, designed from the ground up for fast analytic queries against data of any size. It supports both non-relational sources, such as the Hadoop Distributed File System (HDFS), Amazon S3, Cassandra, MongoDB, and HBase, and relational data sources such as MySQL, PostgreSQL, Amazon Redshift, Microsoft SQL Server, and Teradata.

Presto can query data where it is stored, without needing to move data into a separate analytics system. Query execution runs in parallel over a pure memory-based architecture, with most results returning in seconds. You'll find it used by many well-known companies like Facebook, Airbnb, Netflix, Atlassian, and Nasdaq.

# What is the history of Presto?

Presto started as a project at Facebook, to run interactive analytic queries against a 300PB data warehouse, built with large Hadoop/HDFS-based clusters. Prior to building Presto, Facebook used Apache Hive, which it created and rolled out in 2008, to bring the familiarity of the SQL syntax to the Hadoop ecosystem. Hive had a significant impact on the Hadoop ecosystem for simplifying complex Java MapReduce jobs into SQL-like queries, while being able to execute jobs at high scale. However, it wasn't optimized for fast performance needed in interactive queries.

In 2012, the Facebook Data Infrastructure group built Presto, an interactive query system that could operate quickly at petabyte scale. It was rolled out company-wide in spring, 2013. In November, 2013, Facebook open sourced Presto under the Apache Software License, and made it available for anyone to download on Github. Today, Presto has become a popular choice for doing interactive queries on Hadoop, and has a lot of contributions from Facebook, and other organizations. Facebook's implementation of Presto is used by over a thousand employees, who run more than 30,000 queries, processing one petabyte of data daily.

# How does Presto work?

Presto is a distributed system that runs on Hadoop, and uses an architecture similar to a classic massively parallel processing (MPP) database management system. It has one coordinator node working in synch with multiple worker nodes. Users submit their SQL query to the coordinator which uses a custom query and execution engine to parse, plan, and schedule a distributed query plan across the worker nodes. It is designed to support standard ANSI SQL semantics, including complex queries, aggregations, joins, left/right outer joins, sub-queries, window functions, distinct counts, and approximate percentiles.

After the query is compiled, Presto processes the request into multiple stages across the worker nodes. All processing is in-memory, and pipelined across the network between stages, to avoid any unnecessary I/O overhead. Adding more worker nodes allows for more parallelism, and faster processing.

To make Presto extensible to any data source, it was designed with storage abstraction to make it easy to build pluggable connectors. Because of this, Presto has a lot of connectors, including to non-relational sources like the Hadoop Distributed File System (HDFS), Amazon S3, Cassandra, MongoDB, and HBase, and relational sources such as MySQL, PostgreSQL, Amazon Redshift, Microsoft SQL Server, and Teradata. The data is queried where it is stored, without the need to move it into a separate analytics system.

# What are the differences between Presto and Hadoop?

Presto is an open source, distributed SQL query engine designed for fast, interactive queries on data in HDFS, and others. Unlike Hadoop/HDFS, it does not have its own storage system. Thus, Presto is complimentary to Hadoop, with organizations adopting both to solve a broader business challenge. Presto can be installed with any implementation of Hadoop, and is packaged in the Amazon EMR Hadoop distribution.