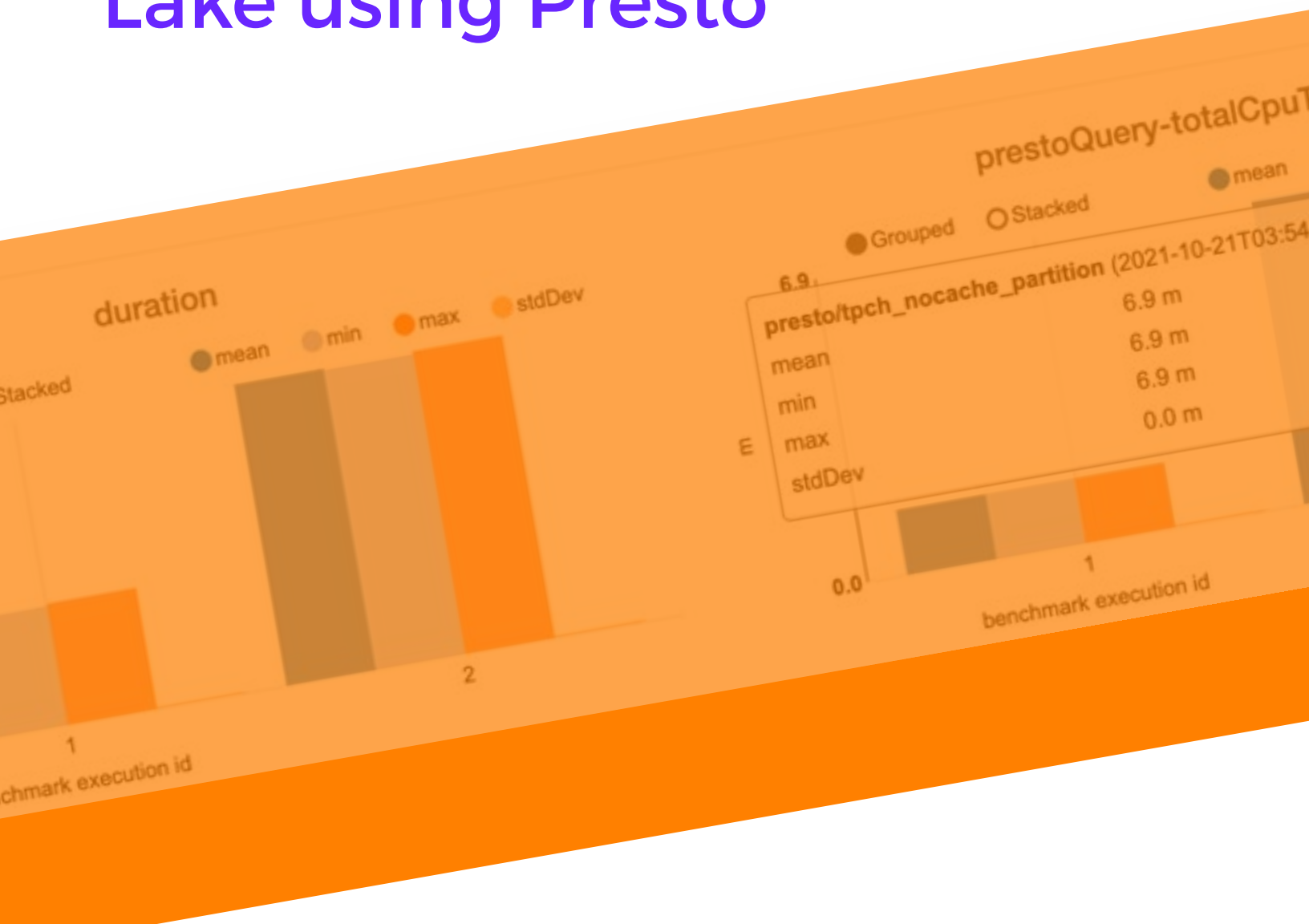


Technical Brief:

# Benchmarking Warehouse Workloads on the Data Lake using Presto



# Technical Brief: How to run a TPC-H benchmark on Presto

## Summary

[Presto](#) is an open source distributed parallel query SQL engine that runs on a cluster of nodes. It's built for high performance and scalable interactive querying with in-memory execution.

A full deployment of Presto has a coordinator and multiple workers. Queries are submitted to the coordinator by a client like the command line interface (CLI), a BI tool, or a notebook that supports SQL. The coordinator parses, analyzes and creates the optimal query execution plan using metadata and data distribution information. That plan is then distributed to the workers for processing. The advantage of this decoupled storage model is that Presto is able to provide a single view of all of your data that has been aggregated into the data storage tier.

Benchmarking is a critical component in an evaluation of Presto since it helps to identify the system resource requirement and usage during various operations and its associated latency metrics for such operations. It's common practice to run a query on a small set of data and conclude on performance, but that might not be true when running at scale. Also, it's good to look at overall performance of various operations at scale rather than a single operation result.

In order to facilitate benchmarking, we're going to use the open source tool `benchto`, part of the Presto project, which can run the industry standard TPC-H database application.

In this tutorial, we will show you how to run `benchto` to benchmark Presto with TPC-H.

# TPC-H

Based on [tpc.org](https://tpc.org), TPC-H is a decision-support benchmark. It consists of a suite of business-oriented ad hoc queries and concurrent data modifications. The queries and the data have been chosen for broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions.

TPC-H comes with various data set sizes to test different scaling factors.

SF (Gigabytes)	Size
1	Consists of the base row size (several million elements).
10	Consists of the base row size x 10.
100	Consists of the base row size x 100 (several hundred million elements).
1000	Consists of the base row size x 1000 (several billion elements).

The TPC-H specification is a 137 page document, but to summarize:

- The database consists of a 3rd Normal Form (3NF) schema consisting of 8 tables.
- The benchmarks can be run using predetermined database sizes, referred to as “scale factors”. Each scale factor corresponds to the raw data size of the data warehouse.
- 6 of the 8 tables grow linearly with the scale factor and are populated with data that is uniformly distributed.
- 22 complex and long running query templates and 2 data refresh processes (insert and delete) are run in parallel to test concurrency.
- The number of concurrent processes increases with the scale factor – for example, for the 100 GB benchmark you run 5 concurrent processes.

## TPC-H Workload

Type	Features	TPCH Business Questions
A	<ul style="list-style-type: none"><li>• Medium dimensionality</li><li>• Result is TPCH scale factor independent</li></ul>	Q1, Q3, Q4, Q5, Q6, Q7, Q8, Q12, Q13, Q14, Q16, Q19, Q22
B	<ul style="list-style-type: none"><li>• High dimensionality</li><li>• Few results, lot of empty cells</li></ul>	Q15, Q18
C	<ul style="list-style-type: none"><li>• High dimensionality</li><li>• Result % of scale factor</li></ul>	Q2, Q9, Q10, Q11, Q17, Q20, Q21

# Benchto tool

Benchto includes the following components:

- Benchto service, which acts as a persistent data store to store the benchmark results.
- Benchto driver, a Java based application, loads the definition of the benchmark descriptors and executes them on the Presto cluster. If cluster monitoring is in place then it collects resource metrics such as CPU, memory, network usage of the cluster.
- Cluster resource usage monitoring is done using Grafana/Graphite.

## Benchto setup

### Step 1:

Install and export JAVA 8 and JDK

### Step 2:

Download benchto from github:

<https://github.com/prestodb/benchto.git>

### Step 3:

Build the benchto project:

```
./mvnw clean install on benchto directory
```

### Step 4:

Download prestodb from github <https://github.com/prestodb/presto.git>

### Step 5:

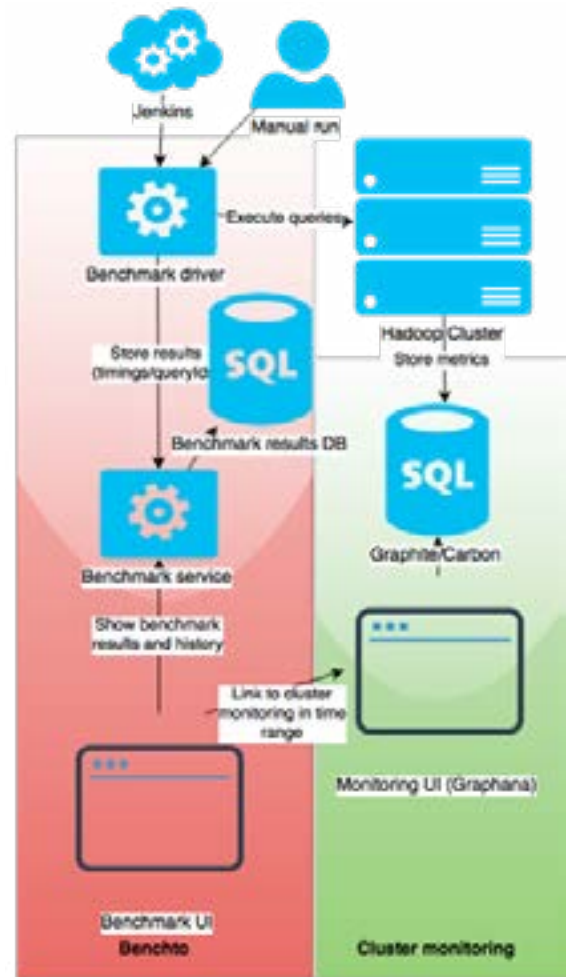
Build the presto-benchto-benchmark:

```
./mvnw clean package -pl presto-benchto-benchmark
```

### Step 6:

Create docker image for benchto-service from benchto directory

```
./mvnw package -pl benchto-service -P docker-images
```



### Step 7:

Run the below command from benchto/benchto-service-docker directory to bring all required docker instances for graphite,grafana,postgres and benchto

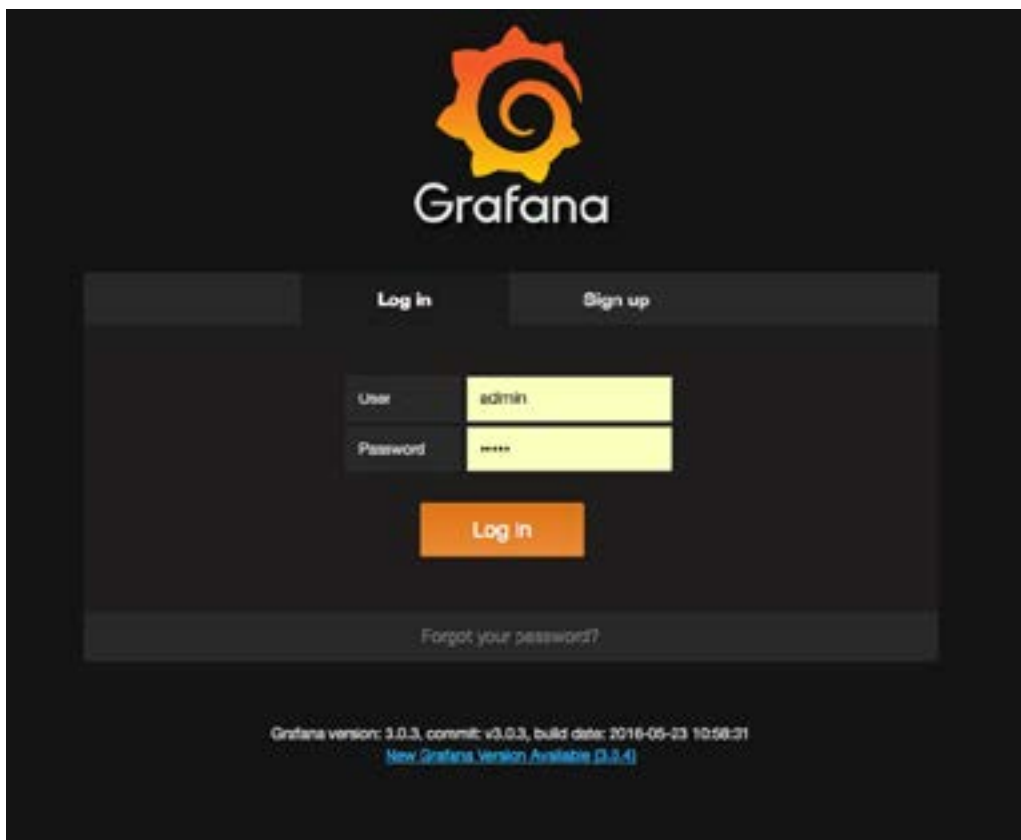
```
docker-compose -f docker-compose.yml up
```

### Step 8:

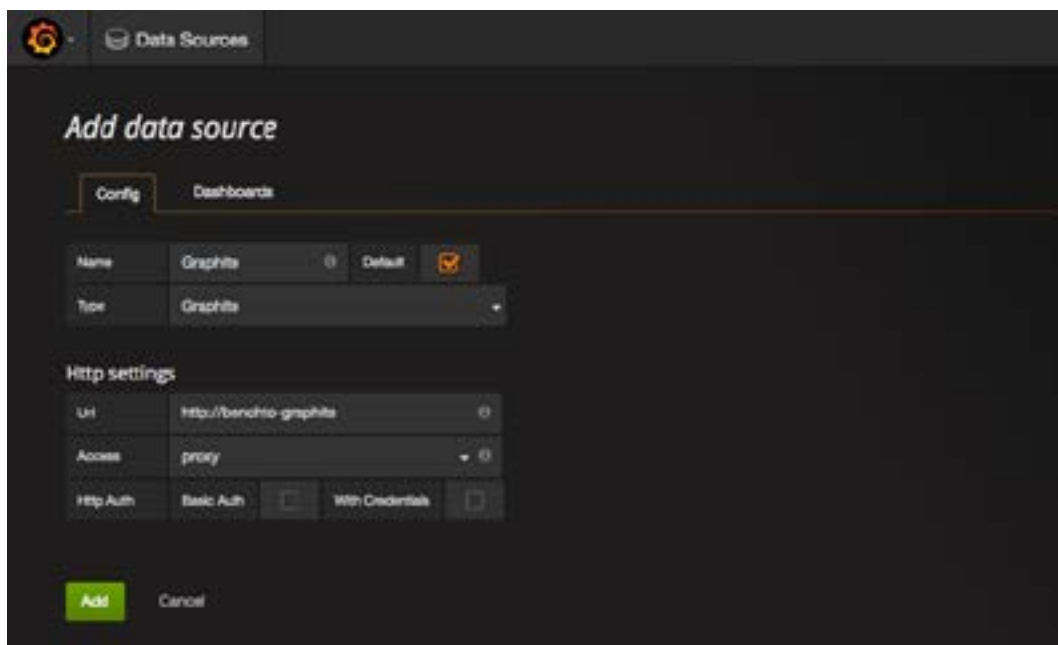
#### Configure Grafana dashboard

In order to view performance metrics in Grafana you have to set up a data source (Graphite) and dashboard. We have created an example dashboard that will show CPU, network and memory performance from localhost.

Log into [Grafana](#) (user: admin, password: admin)



Navigate to Data Sources->Add data source and add Graphite data source as follows:

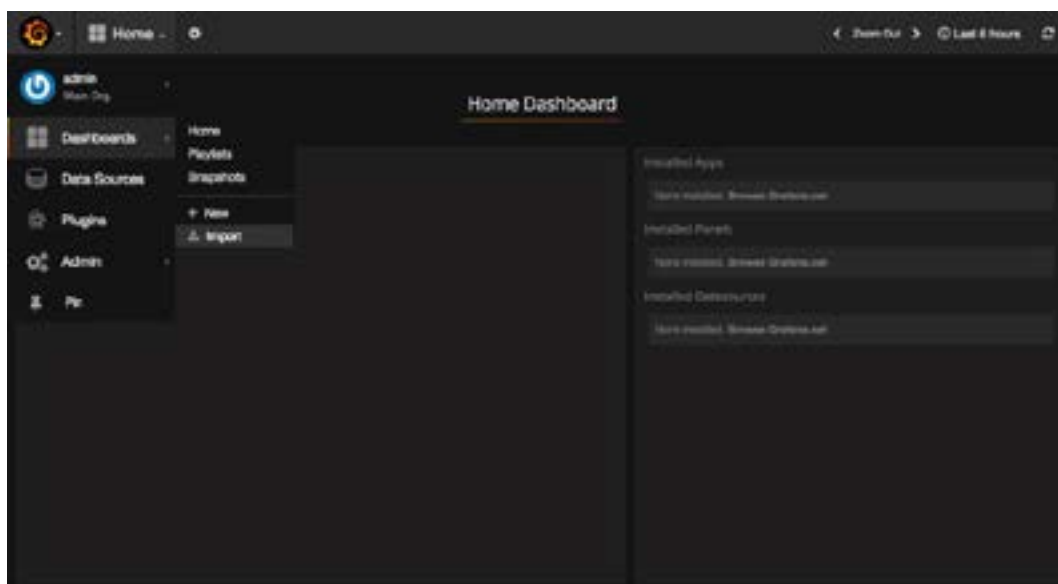


The screenshot shows the 'Add data source' configuration page in Grafana. The page has a dark theme. At the top, there's a header with the Grafana logo and 'Data Sources'. Below the header, the title 'Add data source' is displayed. There are two tabs: 'Config' (selected) and 'Dashboards'. The 'Config' tab contains the following fields:

- Name:** Graphite (with a dropdown arrow and a 'Default' checkbox that is checked).
- Type:** Graphite (with a dropdown arrow).
- Http settings:**
  - Url:** http://benchto-graphite (with a dropdown arrow).
  - Access:** proxy (with a dropdown arrow and a 'With Credentials' checkbox that is unchecked).
  - Http Auth:** Basic Auth (with a checkbox that is unchecked) and With Credentials (with a checkbox that is unchecked).

At the bottom of the form, there are two buttons: 'Add' (in green) and 'Cancel'.

Navigate to Dashboard->Home and import dashboard from `benchto/docs/getting-started/dashboard-grafana.json`



You should now see the Dashboard.

Click Save dashboard icon.



### Step 9:

Generate schemas for TPC-H using superset or presto-cli

You can replace schema\_name and bucket\_name with the actual name you want to use

```
// SQL to create a new database
CREATE SCHEMA IF NOT EXISTS schema_name WITH (location='s3a://bucket_name');

// SQL to create new tables from TPC-H
CREATE TABLE schema_name.lineitem WITH (format='PARQUET') AS SELECT * FROM tpch.
sf100.lineitem;
CREATE TABLE schema_name.orders WITH (format='PARQUET') AS SELECT * FROM tpch.
sf100.orders;
CREATE TABLE schema_name.customer WITH (format='PARQUET') AS SELECT * FROM tpch.
sf100.customer;
CREATE TABLE schema_name.nation WITH (format='PARQUET') AS SELECT * FROM tpch.
sf100.nation;
CREATE TABLE schema_name.part WITH (format='PARQUET') AS SELECT * FROM tpch.
sf100.part;
CREATE TABLE schema_name.partsupp WITH (format='PARQUET') AS SELECT * FROM tpch.
sf100.partsupp;
CREATE TABLE schema_name.region WITH (format='PARQUET') AS SELECT * FROM tpch.
sf100.region;
CREATE TABLE schema_name.supplier WITH (format='PARQUET') AS SELECT * FROM tpch.
sf100.supplier;
```

Presto has a catalog named tpch by default. It has multiple schemas like tiny, sf1, sf100..sf1000 and so on. As the number increases, the size of the table increases. Based on how much data you need you can modify the above queries. I have written sf100 above but you can change it based on your requirement.

**Step 10:**

Create application-presto-devenv.yaml in the directory where you plan to run the test, mostly it will be from the prestodb directory where you cloned prestodb/presto-benchto-benchmarks.

```
cat application-presto-devenv.yaml
benchmark-service:
  url: http://localhost:80

data-sources:
  presto:
    url: jdbc:presto://<YOUR PRESTO CLUSTER>/hive
    driver-class-name: com.facebook.presto.jdbc.PrestoDriver

environment:
  name: PRESTO-DEVENV

presto:
  url: http://<YOUR PRESTO CLUSTER>

benchmark:
  feature:
    presto:
      metrics.collection.enabled: true

macros:
  sleep-4s:
    command: echo "Sleeping for 4s" && sleep 4
```



**Step 11:**

Edit `prestodb/presto-benchto-benchmarks/src/main/resources/benchmarks/presto/tpch.yml` to set the schema name you used in Step 9

```
cat prestodb/presto-benchto-benchmarks/src/main/resources/benchmarks/presto/tpch.yml
datasource: presto
query-names: presto/tpch/${query}.sql
runs: 10
prewarm-runs: 2
before-execution: sleep-4s, presto/session_set_cbo_flags.sql
frequency: 10
database: hive
tpch_300: <your schema name>
```

Here is a list of keywords that could be used to define the benchmark indicators in the above file.

Keyword	Required	Default value	Comment
<b>datasource</b>	True	none	Name of the datasource in application.yml file
<b>query-names</b>	True	none	Path to the Queries
<b>runs</b>	False	3	Number of runs each Query should be executed
<b>Prewarm-runs</b>	False	0	Number of prewarm runs of queries before benchmark
<b>concurrency</b>	False	1	Number of concurrent workers - 1 sequential benchmark, >1 concurrency benchmark
<b>before-benchmark</b>	False	none	Names of macros executed before benchmark.
<b>after-benchmark</b>	False	none	Names of macros executed after benchmark.
<b>before-execution</b>	False	none	Names of macros executed before benchmark executions.
<b>after-execution</b>	False	none	Names of macros executed after benchmark executions
<b>variables</b>	False	false	Set of combinations of variables.
<b>quarantine</b>	False	false	Flag which can be used to quarantine benchmark using <code>-activeVariables</code> property
<b>frequency</b>	False	none	Tells how frequent given benchmark can be executed (in days) 1- once per day, 7 Once per week

**Step 12:**

Set env, make sure to replace prestoURL below to your presto cluster URL

```
cat set_env.sh
curl -H 'Content-Type: application/json' -d '{
    "dashboardType": "grafana",
    "dashboardURL": "http://admin:admin@localhost:3000/dashboard/db/presto-
devenv",
    "prestoURL": "https://prestohost:8080"
}' http://localhost:80/v1/environment/PRESTO-DEVENV

curl -H 'Content-Type: application/json' -d '{
    "name": "Short tag description",
    "description": "Very long but optional tag description"
}' http://localhost:80/v1/tag/PRESTO-DEVENV
```

**Step 13:**

Run the benchmark test

```
java -Xmx1g -jar presto-benchto-benchmarks/target/presto-benchto-benchmarks-*-
executable.jar \
    --sql presto-benchto-benchmarks/src/main/resources/sql \
    --benchmarks presto-benchto-benchmarks/src/main/resources/benchmarks \
    --activeBenchmarks=presto/tpch --profile=presto-devenv \
    --overrides overrides.yaml
```

**Step 14:**

Check results from localhost:80. The results page will show the following:

- Test name which will be the yaml file used in Step 11,
- The hive or glue database name and schema used for this test
- Scale factors used for this test
- Join distribution type and strategy
- Time to complete the test.

ALL	Name	Status	Schema	Tpch_3000	Tpch_300	Database	Join_distribution_type	Tpch_1000	Join_reordering_strategy	Prefix	Mean duration
<input type="checkbox"/>	presto/tpch (2021-09-30T22:41:03+03)	ENDED	tpch_010	tpch_310	tpch_010	shana_hive	AUTOMATIC	tpch_010	ELIMINATE_CROSS_JOINS		
<input type="checkbox"/>	presto/tpch (2021-09-30T22:41:03+03)	ENDED	tpch_010	tpch_310	tpch_010	shana_hive	AUTOMATIC	tpch_010	ELIMINATE_CROSS_JOINS		6.142 s ± 301.673 ms
<input type="checkbox"/>	presto/tpch (2021-09-30T22:41:03+03)	ENDED	tpch_010	tpch_310	tpch_010	shana_hive	AUTOMATIC	tpch_010	ELIMINATE_CROSS_JOINS		6.142 s ± 299.508 ms
<input type="checkbox"/>	presto/tpch (2021-09-30T22:41:03+03)	ENDED	tpch_010	tpch_310	tpch_010	shana_hive	AUTOMATIC	tpch_010	ELIMINATE_CROSS_JOINS		6.142 s ± 318.098 ms

If you drill down on the test name it will provide more details on the run such as duration, output and input data size for the query, CPU time and planning time.

#### Aggregated measurements:

Name	Mean	StdDev	Min	Max
duration	6.142 s	± 0.000 ms (0.00 %)	6.142 s	6.142 s
prestoQuery-outputDataSize	175.000 B	± 0.000 B (0.00 %)	175.000 B	175.000 B
prestoQuery-processedInputDataSize	904.510 MB	± 0.000 B (0.00 %)	904.510 MB	904.510 MB
prestoQuery-rawInputDataSize	904.510 MB	± 0.000 B (0.00 %)	904.510 MB	904.510 MB
prestoQuery-totalBlockedTime	35.860 m	± 0.000 ms (0.00 %)	35.860 m	35.860 m
prestoQuery-totalCpuTime	1.830 m	± 0.000 ms (0.00 %)	1.830 m	1.830 m
prestoQuery-totalPlanningTime	43.620 ms	± 0.000 ms (0.00 %)	43.620 ms	43.620 ms
prestoQuery-totalScheduledTime	2.900 m	± 0.000 ms (0.00 %)	2.900 m	2.900 m

## Comparing the Results

Results can be compared for different runs using the UI by selecting the query and choosing two different runs. In the example below you can see for query q03, there are runs for table partition with [Ahana cache](#) and table partition without Ahana cache.

Choose the compare all option in the top right corner.

Benchto

Home / PRESTO-DEVENV

Search: q03

Showing 1 to 24 of 24 entries (filtered from 569 total entries)

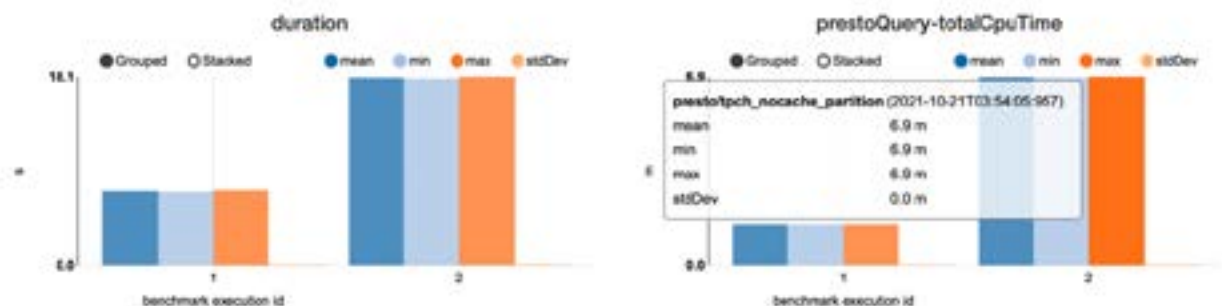
ALL	Name	Status	Schema	Type
<input checked="" type="checkbox"/>	presto/tpch_table_partition (2021-10-22T00:17:03:432)	ENDED	tpch_100_part	tpch_100_part
<input type="checkbox"/>	presto/tpch_table_partition (2021-10-22T00:17:03:432)	ENDED	tpch_100_part	tpch_100_part
<input checked="" type="checkbox"/>	presto/tpch_nocache_partition (2021-10-21T03:54:05:957)	ENDED	tpch_100_nocache01	tpch_100_nocache01

Compare all

Export to CSV

Doing so will result in a page, which will provide a chart for comparison.

### Query graphs



## Summary

Presto was designed for high performance analytics. The TPC-H benchmarking results will provide guidance on what to expect when similar queries are run. There are many tunables that need to be configured to achieve the best result, and this typically requires in-depth knowledge of Presto (which might be quite daunting). In such instances, choosing to use a managed service like Ahana cloud can help, because it takes care of the caching, configuration and tuning parameters under the hood.

You can get started with a 14 day free trial of Ahana at <https://app.ahana.cloud/signup>



- ▶ Learn more about [Ahana Cloud for Presto](#), the only SaaS for Presto on AWS
- ▶ [What is Presto?](#) More about the open source SQL query engine
- ▶ See why security leader [Securonix chose Ahana](#) to power its open data lake analytics
- ▶ Learn more about [Ahana's pay-as-you-pricing](#) through the AWS marketplace
- ▶ Start your [14 day free trial of Ahana](#)

Ahana, the only SaaS for Presto, offers the only managed service for Presto on AWS with the vision to simplify open data lake analytics. Presto, the open source project created by Facebook and used at Uber, Twitter and thousands more, is the de facto standard for fast SQL processing on data lakes. Ahana Cloud delivers the easiest Presto SaaS and enables data platform teams to provide high performance SQL analytics on their S3 data lakes and other data sources. As a leading member of the Presto community and Linux Foundation's Presto Foundation, Ahana is also focused on fostering growth and evangelizing open source Presto. Founded in 2020, Ahana is headquartered in San Mateo, CA and operates as an all-remote company. Investors include GV, Lux Capital, and Leslie Ventures.