Learn Materialize by running streaming SQL on your nginx logs

In this tutorial, I will show you how Materialize works by using it to run SQL queries on continuously produced nginx logs. By the end of the tutorial, you will have a better idea of what Materialize is, how it's different than other SQL engines, and how to use it.

Prerequisites

For the sake of simplicity, I will use a brand new Ubuntu 21.04 server where I will install nginx, Materialize and `mzcli`, a CLI tool similar to `psql` used to connect to Materialize and execute SQL on it.

If you want to follow along you could spin up a new Ubuntu 21.04 server on your favorite could provider.

If you prefer runing Materialize on a different operating system, you can follow the steps on how to install Materialize here:

- How to install Materialize

What is Materialize

Materialize is a streaming database for real-time analytics.

It is not a substitution for your transactional database, instead it accepts input data from a variety of sources like:

- Messages from streaming sources like Kafka
- Archived data from object stores like S3
- Change feeds from databases like PostgreSQL
- Data in Files: CSV, JSON and even unstructured files like logs *(what we'll be using today.)*

And it maintains the answers to your SQL queries over time, keeping them up-to-date as new data flows in (using *materialized views*), instead of running them against a static snapshot at a point in time.

If you want to learn more about Materialize, make sure to check out their official documentation here:

Materialize Documentation

Installing Materialize

Materialize runs as a single binary called `materialized` *(d for daemon, following Unix conventions)*. Since we're running on Linux, we'll just install Materialize directly. To install it, run the following command:

```
sudo apt install materialized
```

Once it's installed, start Materialize (with sudo so it has access to nginx logs):

```
sudo materialized
```

Now that we have the `materialized` running, we need to open a new terminal to install and run a CLI tool that we use to interact with our Materialize instance!

There are other ways that you could use in order to run Materialize as described here. For a production-ready Materialize instance, I would recommend giving Materialize Cloud a try!

Installing `mzcli`

The `mzcli` tool lets us connect to Materialize similar to how we would use a SQL client to connect to any other database.

Materialize is wire-compatible with PostgreSQL, so if you have `psql` already installed you could use it instead of `mzcli`, but with `mzcli` you get nice syntax highlighting and autocomplete when writing your queries.

To learn the main differences between the two, make sure to check out the official documentation here: Materialize CLI Connections

The easiest way to install `mzcli` is via `pipx`, so first run:

```
apt install pipx
```

and, once `pipx` is installed, install `mzcli` with:

```
pipx install mzcli
```

Now that we have `mzcli` we can connect to `materialized` with:

```
mzcli -U materialize -h localhost -p 6875 materialize
```

For this demo, let's quickly install nginx and use Regex to parse the log and create Materialized Views.

Installing nginx

If you don't already have nginx installed, install it with the following command:

```
sudo apt install nginx
```

Next, let's populate the access log with some entries with a Bash loop:

```
for i in {1..200} ; do curl -s 'localhost/materialize'  >
/dev/null ; echo $i ; done
```

If you have an actual nginx `access.log`, you can skip the step above.

Now we'll have some entries in the `/var/log/nginx/access.log` access log file that we would be able to able to feed into Materialize.

Adding a Materialize Source

By creating a Source you are essentially telling Materialize to connect to some external data source. As described in the introduction, you could connect a wide variety of sources to Materialize.

For the full list of source types make sure to check out the official documentation here:

Materialize source types

Let's start by creating a text file source from our nginx access log.

First, access the Materialize instance with the `mzcli` command:

```
mzcli -U materialize -h localhost -p 6875 materialize
```

Then run the following statement to create the source:

```
CREATE SOURCE nginx_log
FROM FILE '/var/log/nginx/access.log'
WITH (tail = true)
FORMAT REGEX '(?P<ipaddress>[^ ]+) - - \[(?P<time>[^\]]+)\]
"(?P<request>[^ ]+) (?P<url>[^ ]+)[^"]+"
(?P<statuscode>\d{3})';
```

A quick rundown:

- `CREATE SOURCE`: First we specify that we want to create a source
- `FROM FILE`: Then we specify that this source will read from a local file, and we provide the path to that file
- `WITH (tail = true)`: Continually check the file for new content
- `FORMAT REGEX`: as this is an unstructured file we need to specify

regex as the format so that we could get only the specific parts of the log that we need.

Let's quickly review the Regex itself as well.

The Materialize-specific behavior to note here is the `?P<NAME_HERE>` pattern extracts the matched text into a column named `NAME_HERE`.

To make this a bit more clear, a standard entry in your nginx access log file would look like this:

```
123.123.123.119 - - [13/Oct/2021:10:54:22 +0000] "GET /
HTTP/1.1" 200 396 "-" "Mozilla/5.0 zgrab/0.x"
```

- `(?P<ipaddress>[^ ]+)`: With this pattern we match the IP address for each line of the nginx log, e.g. `123.123.123.119`.
- `\[(?P<time>[^\]]+)\]`: the timestamp string from inside square brackets, e.g. `[13/Oct/2021:10:54:22 +0000]`
- `"(?P<request>[^ ]+)`: the type of request like `GET`, `POST` etc.
- `(?P<url>[^ ]+)`: the relative URL, eg. `/favicon.ico`
- `(?P<statuscode>\d{3})`: the three digit HTTP status code.

Once you execute the create source statement, you can confirm the source was created successfully by running the following:

```
mz> SHOW SOURCES;
// Output
+-----------+
| name      |
|-----------|
| nginx_log |
+-----------+
SELECT 1
Time: 0.021s
```

Now that we have our source in place, let's go ahead and create a view!

Creating a Materialized View

You may be familiar with Materialized Views from the world of traditional databases like PostgreSQL, which are essentially cached queries. The unique feature here is the materialized view we are about to create is **automatically kept up-to-date**.

In order to create a materialized view, we will use the following statement:

```
CREATE MATERIALIZED VIEW aggregated_logs AS
  SELECT
    ipaddress,
    request,
    url,
    statuscode::int,
    COUNT(*) as count
  FROM nginx_log GROUP BY 1,2,3,4;
```

The important things to note are:

- Materialize will keep the results of the embedded query in memory, so you'll always get a fast and up-to-date answer
- The results are incrementally updated as new log events arrive

Under the hood, **Materialize compiles your SQL query into a dataflow** and then takes care of all the heavy lifting for you. This is incredibly powerful, as it allows you to process data in real-time using *just* SQL.

A quick rundown of the statement itself:

- First we start with the `CREATE MATERIALIZED VIEW aggregated_logs` which identifies that we want to create a new Materialized view named `aggregated_logs`.
- Then we specify the `SELECT` statement that we are interested in keeping track of over time. In this case we are aggregating the data in our log file by `ipaddress`, `request`, `url` and `statuscode`, and we are counting the total instances of each combo with a `COUNT(*)`

When creating a Materialized View, it could be based on multiple sources like a stream from Kafka, a raw data file that you have on an S3 bucket, or your PostgreSQL database. This single statement will give you the power to analyze your data in real-time.

We specified a simple `SELECT` that we want the view to be based on but this could include complex operations like `JOIN`s, however for the sake of this tutorial we are keeping things simple.

For more information about Materialized Views check out the official documentation here:

Creating Materialized views

Now you could use this new view and interact with the data from the nginx log with pure SQL!

Reading from the view

If we do a `SELECT` on this Materialized view, we get a nice aggregated summary of stats:

```
SELECT * FROM aggregated_logs;

   ipaddress    | request |           url            |
statuscode | count
----------------+---------+--------------------------+--------
----+-------
 127.0.0.1      | GET     | /materialize             |
404 |   200
```

As more requests come in to the nginx server, the aggregated stats in the view are kept up-to-date.

We could also write queries that do further aggregation and filtering on top of the materialized view, for example, counting requests by route only:

```
SELECT url, SUM(count) as total FROM aggregated_logs GROUP BY
1 ORDER BY 2 DESC;
```

If we were re-run the query over and over again, we could see the numbers change instantly as soon as we get new data in the log as Materialize processes each line of the log and keeps listening for new lines:

```
+---------------------------+-------+
| url                       | total |
|---------------------------+-------|
| /materialize/demo-page-2 | 1255  |
| /materialize/demo-page   | 1957  |
| /materialize             | 400   |
+---------------------------+-------+
```

As another example, let's use `psql` together with the `watch` command to see this in action.

If you don't have `psql` already isntalled you can install it with the following command:

```
sudo apt install postgresql-client
```

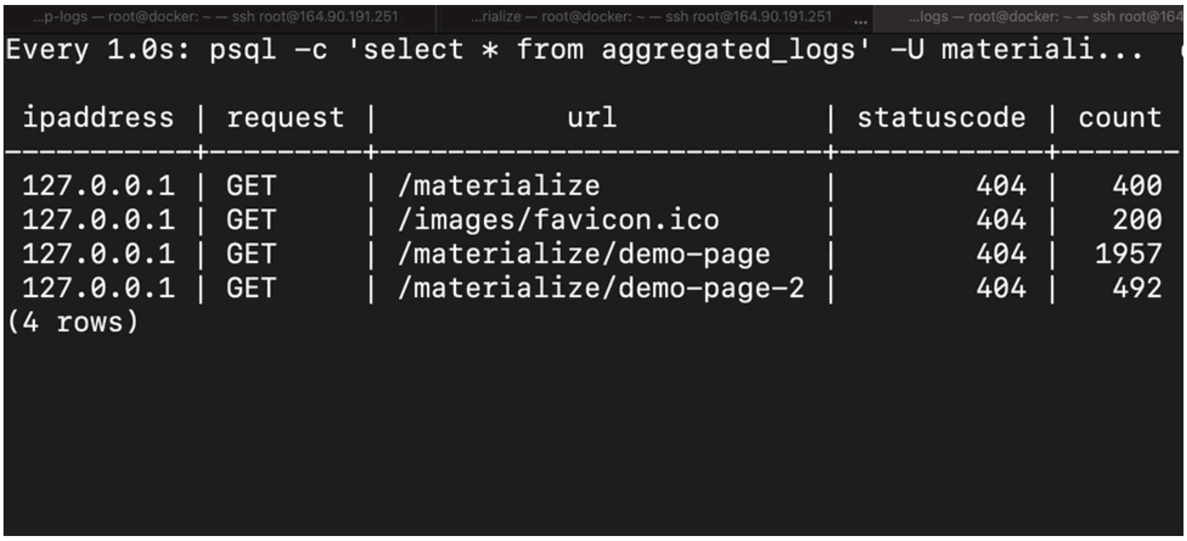
After that, let's run the `SELECT * FROM aggregated_logs` statement every second using the `watch` command:

```
watch -n1 "psql -c 'select * from aggregated_logs' -U
materialize -h localhost -p 6875 materialize"
```

In **another terminal window**, you could run another `for` loop to generate some new nginx logs and see how the results change:

```
for i in {1..2000} ; do curl -s 'localhost/materialize/demo-
page-2' > /dev/null ; echo $i ; done
```

The output of the `watch` command would look like this:


...p-logs — root@docker: ~ — ssh root@164.90.191.251
...rialize — root@docker: ~ — ssh root@164.90.191.251
...logs — root@docker: ~ — ssh root@164

```
Every 1.0s: psql -c 'select * from aggregated_logs' -U materiali...

 ipaddress | request |           url            | statuscode | count
-----------+---------+--------------------------+------------+-------
 127.0.0.1 | GET     | /materialize             |        404 |   400
 127.0.0.1 | GET     | /images/favicon.ico      |        404 |   200
 127.0.0.1 | GET     | /materialize/demo-page   |        404 |  1957
 127.0.0.1 | GET     | /materialize/demo-page-2 |        404 |   492
(4 rows)
```

Feel free to experiment with more complex queries and analyze your nginx access log for suspicious activity using pure SQL and keep track of the results in real-time!

Conclusion

By now, hopefully you have a hands-on understanding of how incrementally maintained materialized views work in Materialize. In case that you like the project, make sure to star it on GitHub:

https://github.com/MaterializeInc/materialize

Source