What's new in Citus 11.3 & Postgres for multi-tenant SaaS workloads

Written by **Marco Slot**

May 5, 2023

Citus enables several different PostgreSQL use cases, but one of the most popular ones is to build scalable multi-tenant software as a service (SaaS) applications. The most common way to build a multi-tenant application on Citus is to distribute all your Postgres tables by a "tenant ID" column. That way rows are (hash-)distributed across nodes, while rows with the same tenant ID value are co-located on the same node for fast local joins, transactions, and foreign keys.

For those of you who build SaaS apps, one question many of you have is how active your tenants are. More specifically: What are your busiest tenants? How many queries is your application doing on behalf of your tenants, and how much CPU do those queries use?

The new 11.3 release to the open source Citus database extension gives you tenant monitoring—with instant visibility into your top tenants using the new `citus_stat_tenants` feature, which shows query counts and CPU usage over a configurable time period.

Citus 11.3 also makes many other improvements, some of which lay the groundwork for future multi-tenant SaaS capabilities too! Let's dive in to explore these new 11.3 features:

Details in the Release Notes: If you want to know the full details of the changes in Citus 11.3, check out the 11.3 Updates page for the detailed release notes.

Demos in the Release Party Livestream: Mark your calendar for the Release Party if you want to see live demos of some of the new Citus 11.3 capabilities. The livestream of the Citus 11.3 Release Party will include 3 live engineering demos and is scheduled for Monday May 15 @ 9:00am PDT = 12noon EDT = 6:00pm CEST.

Everything happens in the Citus GitHub repo: You can file issues, see what the latest commits are, and start discussions on the Citus open source GitHub repo. (Although most of the Q&A happens on the Citus Public Slack.)

Tenant monitoring with citus_stat_tenants

When managing a Citus cluster (or really any database) one of the most important things is getting insights into who or what is using resources. Postgres contains pg_stat_statements and Citus added citus_stat_statements in Citus 7.5, which we then open sourced in Citus 11.0. These views give you very detailed insights into which Postgres queries are running most often on your system.

Now in Citus 11.3 we're adding the citus_stat_tenants view to your arsenal. If you're running a multi-tenant SaaS application on top of Citus, then with `citus_stat_tenants` you can very easily get an overview of the tenants that use most of the resources in your cluster.

Since tracking this data has some performance overhead, it's not enabled by default. But it's very easy to enable by setting `citus.stat_tenants_track` to `'all'`:

```
ALTER SYSTEM SET citus.stat_tenants_track TO 'all';
SELECT pg_reload_conf();
```

When you enable the tracking there are two main ways to use the `citus_stat_tenants` view.

Showing usage of tenants in real-time

The first method of using `citus_stat_tenants` is extremely easy. You can look directly at the contents of the view to see the traffic that's currently happening on your cluster. Using the `\watch` command that's included in psql you can even see it changing in real time. Below is an example that uses citus_stat_tenants to show in real time what companies are causing the most queries to your database:

```
> SELECT name, tenant_attribute, query_count_in_this_period
FROM citus_stat_tenants JOIN companies ON tenant_attribute::bigint = companies.id
ORDER BY query_count_in_this_period DESC \watch 10
              Wed 03 May 2023 02:46:14 PM CEST (every 10s)

          name            | tenant_attribute | query_count_in_this_period
---------------------------+------------------+----------------------------
 Microsoft                 | 1                |                        141
 Citywide Coffeeshop       | 2                |                         23
 Small Corner Store Bakery | 3                |                          3
(3 rows)

              Wed 03 May 2023 02:46:24 PM CEST (every 10s)
```

```
        name           | tenant_attribute | query_count_in_this_period
-----------------------+------------------+----------------------------
 Microsoft             | 1                |                        241
 Citywide Coffeeshop   | 2                |                         40
 Small Corner Store Bakery | 3            |                          5
(3 rows)

          Wed 03 May 2023 02:46:34 PM CEST (every 10s)

        name           | tenant_attribute | query_count_in_this_period
-----------------------+------------------+----------------------------
 Microsoft             | 1                |                        341
 Citywide Coffeeshop   | 2                |                         56
 Small Corner Store Bakery | 3            |                          7
(3 rows)
```

Showing usage of tenants in monitoring dashboards

The second way in which `citus_stat_tenants` can be useful is by integrating it with monitoring systems like Grafana and Prometheus. By having such systems periodically read from the `citus_stat_tenants` view you can see changes over longer periods of time in the interface that you are already used to.

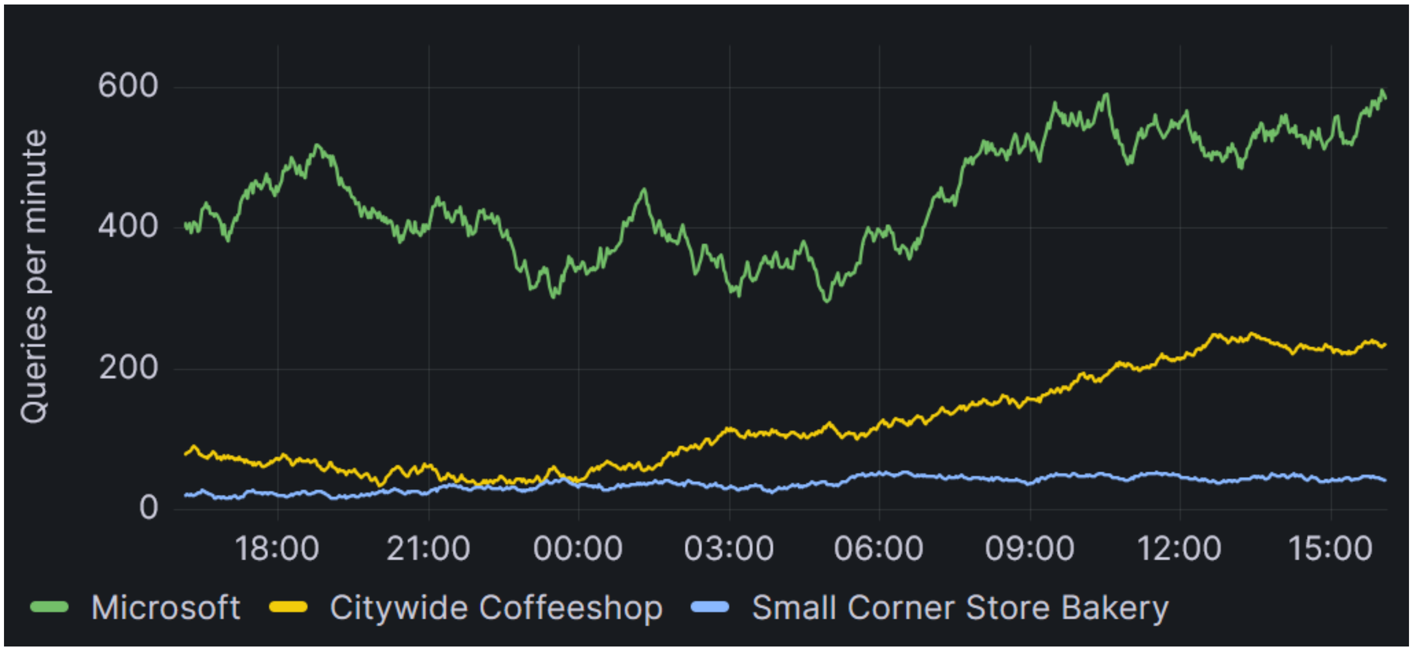

Figure 1: Changes in query count per minute for different companies over time, shown in a Grafana dashboard. The data that's shown here came directly from `citus_stat_tenants`.

When integrating `citus_stat_tenants` directly into your monitoring system the most important thing is to remember is to store the values from the columns that end in `_in_last_period`, such as `query_count_in_last_period`. These numbers represent the total statistics that were gather for the previous period. Because they are totals for a complete period, they can easily be compared to totals taken from a later/earlier period.

This is not the case for the numbers in the `_in_this_period` columns, which are updated continuously, and thus only represent the totals for a partial period.

By default the period length is one minute, but this can be changed by setting the `citus.stat_tenants_period` configuration variable.

More reliable metadata syncing for very large numbers of tables

In Citus 11.0 we enabled querying from any node by synchronizing catalogs & Citus metadata across the database cluster. That allows Citus to scale to very high transaction throughputs and has also become a key enabler for various new features, such as change data capture (CDC), see below!

As Citus is becoming more popular, we are seeing more and more users with a very large number of tables (tens of thousands of tables with hundreds of thousands of shards!) In part, this is due to the popularity of using time-partitioning for time series data on Citus.

When you have *a lot* of tables, synchronizing the Citus metadata to new worker nodes can become a scalability problem by itself. We learned the hard way that if you perform too many DDL commands in a transaction in PostgreSQL, it can generate invalidation arrays that exceed the 1GB limit that PostgreSQL imposes on itself.

In the Citus 11.3 release, we spent a lot of time streamlining the metadata synchronization in order to minimize memory usage, avoid transactions where necessary, and increase overall reliability. With these changes, we expect Citus to give you a seamless experience even with a very large number of Citus tables.

Parallel shard rebalancing across co-location groups

When we added shard rebalancing in the background to Citus 11.1, we also began preparations for running shard moves in parallel.

In 11.3, you can now benefit from the first stage of parallel shard rebalancing: **Shard groups *from different co-location groups* will be moved in parallel by the rebalancer**

Right now shard groups in the same co-location group will still be moved sequentially. In the future, we also plan to add support for parallel shard rebalancing within a co-location group.

> Co-location groups are an important concept in Citus. All distributed tables in a co-location group share the same distribution column type and the same shard hash ranges. Also, shards with the same hash range in the same co-location group are always on the same node. We refer to a set of co-located shards as a "shard group". You can see which tables share a co-location group by running `SELECT * FROM citus_tables` and looking at the `colocation_id` column.

You do not have to do anything to enable the new parallel shard rebalance feature. When you run `SELECT citus_rebalance_start()` the Citus rebalancer will automatically find opportunities to parallelize when there are multiple co-location groups.

MERGE support between co-located distributed tables

PostgreSQL 15 added support for the MERGE command, and as part of Citus 11.3 we have been working on parallel, distributed MERGE between (co-located) distributed tables.

Many of you are already familiar with the INSERT..SELECT command, which can be used to pre-aggregate incoming data in parallel. You can also use ON CONFLICT to update existing aggregates while also inserting new ones.

The MERGE command is a more advanced version of INSERT..SELECT that can also DELETE. For instance, let's say I have a stock table that keeps track of how many items I have left.

```
CREATE TABLE stock (item_id bigint primary key, num_available int not null default 0);
INSERT INTO stock VALUES (1027, 3);
INSERT INTO stock VALUES (2011, 1);
INSERT INTO stock VALUES (7001, 15);
SELECT create_distributed_table('stock', 'item_id');
```

Periodically, my application receives a batch of stock changes. I want to update my table to reduce the number of available items and delete records that reach 0. To do so, I can create a staging table that is co-located with my `stock` table, load the data, and then... MERGE!

```
-- create a co-located staging table
BEGIN;
CREATE UNLOGGED TABLE stock_changes (item_id bigint, num_consumed int);
SELECT create_distributed_table('stock_changes', 'item_id');
-- load the data
COPY stock_changes FROM STDIN WITH CSV;
1027,2
2011,1
5431,4
\.

-- merge staging data into main table, ignore unknown item_ids
MERGE INTO stock s
USING stock_changes c
ON s.item_id = c.item_id
WHEN MATCHED AND s.num_available - c.num_consumed <= 0 THEN
  DELETE
WHEN MATCHED THEN
  UPDATE SET num_available = s.num_available - c.num_consumed;

-- remove my staging table
```

```
DROP TABLE stock_changes;
END;

-- show the updated stock table
TABLE stock;
item_id | num_available
---------+---------------
    7001 |            15
    1027 |             1
(2 rows)
```

Since MERGE is still a very new feature to Postgres and to Citus, there are various limitations in the Citus support. So far, MERGE mainly works for co-located distributed Citus tables that merge on the distribution column, and it does not support stable/volatile function calls (e.g. inserting into a table with DEFAULT now()). However, the fact that MERGE is parallelized across nodes can make it a powerful data processing tool.

Logical decoding for Citus tables for CDC (preview)

A common requirement of many applications—and software as a service (SaaS) applications especially—is the ability to perform change data capture (CDC) on the database, meaning changes happening on the database can be consumed and processed by other applications. PostgreSQL offers CDC via logical decoding.

Citus 11.3 adds a new `citus.enable_change_data_capture` setting—which is in preview, which is just another term for "beta". This new change data capture setting improves logical decoding on the distributed nodes of a Citus database cluster in 2 important ways:

1. Changes to shards are emitted as changes to distributed tables, such that what you see in the logical decoding output matches what clients do (e.g. insert into a distributed table)
2. When Citus performs an internal data transfer (e.g. shard move, create_distributed_table), it does not show up as a new batch of inserts in logical decoding.

With these changes, logical decoding on distributed Citus tables starts looking like a lot more like logical decoding on PostgreSQL tables—except that you should configure your logical decoding clients to subscribe to all nodes in the Citus cluster at the same time.

In addition, replication slots need to be created on each node separately:

```
-- Publication is auto-propagated by Citus and can contain a mixture of regular tables and Citus tables
CREATE PUBLICATION distpub FOR TABLES data;
-- Replication slot needs to be created separately on all nodes
```

```
SELECT * FROM run_command_on_all_nodes($$SELECT pg_create_logical_replication_slot('cdc_slot', 'pgoutput'));
-- ready to consume changes from all nodes!
```

Logical replication of distributed tables to PostgreSQL tables

In PostgreSQL, logical decoding forms the basis of logical replication, which allows you to replicate changes from one PostgreSQL server to another. It is now also possible to set up logical replication from a distributed table to a regular PostgreSQL table. To do so, you need to set up a publication on all the nodes.

On source node:

```
CREATE TABLE items (key text primary key, value jsonb not null default '{}');
SELECT create_distributed_table('items', 'key');
CREATE PUBLICATION items_pub FOR TABLES items;
```

On destination node:

```
CREATE TABLE items (key text primary key, value jsonb not null default '{}');
-- subscribe to coordinator
CREATE SUBSCRIPTION subc CONNECTION 'host=c.mycluster … ' PUBLICATION items_pub WITH (copy_data = false);
-- subscribe to each worker…
CREATE SUBSCRIPTION subw0 CONNECTION 'host=w0.mycluster … ' PUBLICATION items_pub WITH (copy_data = false);
CREATE SUBSCRIPTION subw1 CONNECTION 'host=w1.mycluster  … ' PUBLICATION items_pub WITH (copy_data = false);
…
```

Inserts and other changes to the distributed table will then automatically show up in your PostgreSQL table.

There are a few additional limitations of logical decoding on Citus tables (in preview) to consider:

- **Only pgoutput & wal2json, so far**: You can currently only use pgoutput and wal2json (if installed) as decoders. Other decoders will not have the sharding-related changes applied to them, but we can easily add more decoders so let us know which decoders you would like us to add by opening a GitHub issue on our Citus open source repo.
- **Replication slots need to be created separately on each node**: When you add a new node, you first need to create a replication slot (or subscription) before rebalancing.
- **Out of order issues**: Changes happening on the same node always arrive in the same order, but since your client will listen to each node separately, there are no guarantees regarding the order of changes happening across different nodes.
- **Specific to logical replication**:
  - Using copy_data = true in logical replication is not officially supported yet, but not explicitly blocked. If you use it, make sure to only use it on one of the subscriptions

    (e.g. the subscription that connects to the coordinator).
  - Using `alter_distributed_table` will break the replication stream

In addition to Citus-specific limitations, a general limitation of logical decoding in PostgreSQL is that logical replication slots are lost after a failover (using physical replication), unless you are using Patroni with permanent slots, or the new pg_failover_slots extension.

We plan to solve these limitations in future Citus open source releases but wanted to give you early access to try it with your favorite CDC clients. Once CDC is stable, we will also start enabling it on Azure, in the Azure Cosmos DB for PostgreSQL managed service.

Onwards to Citus 12

We are currently laser-focused on making Citus the best possible database for scalable, multi-tenant SaaS applications. The new tenant monitoring feature in 11.3 gives direct insight into your top tenants within the database, such that you can quickly understand the state of your cluster ("who is using so much CPU?") and make decisions regarding your tenants.

Several of the enhancements in 11.3, such as the parallel shard rebalance and metadata syncing improvements, are partially in preparation for a bigger feature (in Citus 12) that we call *schema-based sharding*. Distributing tables by tenant ID scales very well, and the number of tenants is virtually unbounded, but to use it you may have to (re)design your data model, including table schema, primary keys, foreign keys. Another common pattern for building multi-tenant apps on PostgreSQL, which is used by various ORMs (e.g. Ruby Apartment and Python django-tenants), is to create a separate schema for each tenant. Our goal in the next release of Citus is to transparently distribute schemas across nodes without data model or application changes, such that you can get the benefits of horizontal scale out without the downsides.

Next steps with Citus 11.3

If you want to get started with Citus 11.3, these links should prove useful:

- **Download page**
- **Citus 11.3 Updates page**: a detailed release notes page
- **Citus 11.3 Release Party**: a livestream with demos & discussion about 11.3 happening (mark your calendar to watch the livestream demos on Mon May 15 @ 9:00am PDT)
- **Citus open source repo on GitHub**
- **Citus docs**

We are also curious about your feedback, especially on new brand features like the tenant monitoring and the logical decoding improvements for CDC (in preview). If you find any issues at all, please open a GitHub issue or let us know via the Citus Slack.

Enjoy what you're reading?

If you want to read more posts from our Citus database and Postgres teams, sign up for our monthly newsletter and get the latest content delivered straight to your inbox.

Written by Marco Slot

Lead engineer for the Citus database engine at Microsoft. Speaker at Postgres Conf EU, PostgresOpen, pgDay Paris, Hello World, SIGMOD, & lots of meetups. Talk selection team member for Citus Con: An Event for Postgres. PhD in distributed systems. Loves mountain hiking.

@marcoslot marcocitus

CDC Citus Citus release notes co-location distributed databases distributed Postgres logical decoding monitoring multi-tenant open source SaaS shard rebalancer

Patroni 3.0 & Citus: Scalable, Highly Available Postgres

By Alexander Kukushkin

Debugging PostgreSQL CI failures faster: 4 tips

By Nazir Bilal Yavuz

Postgres 15 available in Azure Cosmos DB for PostgreSQL

By Nik Larin

Debugging Postgres autovacuum problems: 13 tips

By Samay Sharma

How to benchmark performance of Citus and Postgres with HammerDB on Azure

By Jelte Fennema

LATEST POSTS

RELATED CONTENT

What's new in Citus 11.2 for Postgres, plus Patroni HA support for Citus

By Marco Slot

Read more

Distributed Postgres goes full open source with Citus: why, what & how

By Jelte Fennema

Read more

Ultimate Guide to Citus Con: An Event for Postgres, 2023 edition

By Claire Giordano

Read more

SUBSCRIBE TO OUR NEWSLETTER

Join our Slack!