How to configure a PostgreSQL cluster for multitenancy

Bryn Llewellyn
Technical Product Manager at Yugabyte

yugabyteDB

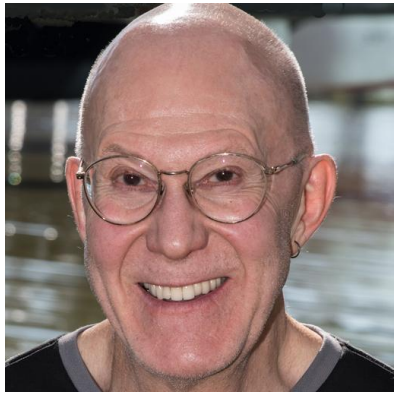
Who am I?

linkedin.com/in/bryn-llewellyn/

twitter.com/BrynLite

www.yugabyte.com/blog/author/bryn/

Google for: “PostgreSQL Person of the Week” Bryn



github.com/YugabyteDB-Samples/ysql-case-studies

Who do I think you are?

- You know PostgreSQL very well.
- Not a week goes by without you typing SQL at the *psql* prompt.
- You don’t need me to tell you about the reasons to use SQL.
- You don’t mind that Codd and Date laid the foundations a very long time ago.
- You understand the value of user-defined subprograms and are used to writing them.
- Maybe you even have some exposure to YugabyteDB.

The **YBMT** scheme

(**y**uga**b**yteDB **m**ulti**t**enancy)

Agenda

- What requirements does the YBMT scheme meet?
- Why isn’t native PostgreSQL functionality enough?
- What are YBMT’s essential concepts?
- Simple demo.
- Some more detail:
  - Mechanizing database *create* and *drop* using psql given that you can’t create or drop a database from PL/pgSQL.
  - Security definer subprograms for role maintenance.
  - The “zero-privileged” *client* role.
  - Utilities—esp. user-friendly catalog views and table functions.
- Bonus: quick tour of the actual YSQL case-studies.

Caveat

Envisaged paradigm for the development shop and the deployment site

- The app uses at least two roles:
  - ≥ “client” role(s). Owns no schema. Lacks “create” on any schema. Functionality comes via privileges. Client-side code can connect only as a “client” role.
  - ≥ “implementation” role(s). Owns all the schemas and schema-objects.
- The development shop works GitHub-style. The app is defined by its checked-in code.
- Each developer clones the repo, deploys the app in their own sandbox, and has free reign to change anything there.
- The “pull request” is the gatekeeper for correctness.
- Initial deployment and patching is done by people who authorize using a single password.

Requirements

What requirements must a scheme like YBMT meet?

- Must allow an application backend to be designed without thinking about what other application backends it might be co-installed with*
  - Each application backend must be securely isolated from every other one.
    - The PostgreSQL-native *database* feature goes a long way to meeting this req't.
      BUT a role is a cluster-wide phenomenon and can own objects in each of several databases.

      This is OK for a superuser or a role that is dedicated to provisioning databases or roles.

      But in general, a role that owns objects in, or can connect to, more than one database, thwarts the goals of multitenancy.
  - Must formalize scheme so that a role can connect *either* to all databases *or* to exactly one database.

* This is *not* "application multitenancy" by striping the tables using a *customer_id* column.

The *bootstrap* database, *tenant* databases, *global* roles and *local* roles

- A minimal viable cluster has exactly one database that allows connections (in addition to template databases that don't allow connections).
  This will be special in YBMT. Call it the *bootstrap* database.

- Must be possible to create any number of additional databases.
  Call these *tenant* databases. All *tenant* databases must:
  - follow the same rules;
  - expose common utilities (from *template1*).

- Must formalize and enforce two kinds of role:

  - **global role**. Must be able to connect to any database. Can own objects in any database.
    Must be a fixed set of these, intrinsic to the YBMT scheme.

  - **local role**. Can connect only to exactly one *tenant* database.

Global roles

(Briefly.)

- Lock the “bootstrap superuser” (*with nologin password null*).
- Use a separate superuser to configure as YBMT and for other *very rare* tasks when they arise.
- Use dedicated role (*with nosuperuser createdb createrole*) for maintenance of databases and roles. Must be able to connect to *every* database.
- Allow “pure” role(s) (i.e. *with nologin password null* and *“no”* everything else) and without any privileges on any database as a vehicle for bundling privileges.
  (Like *pg_read_all_data* and similar.)

(More detail in later slides.)

Local roles

- A *local* role must allow someone to connect to just one particular *tenant* database.
- Must have no “powerful” attributes.
- A newly-created *tenant* database must have is dedicated “manager” *local* role.
- Someone who connects to a *tenant* database, authorizing using as its “manager” *local* role, must be able to create other *local* roles there (and only there).
- The local “manager” role must be able to configure “non-manager” *local* roles, in the current *tenant* database, and to limit their power appropriately.
- The only way to do this is to use *security definer* subprograms that come with the YBMT configuration.

YBMT’s essential concepts

Exactly one *bootstrap* database. *N tenant* databases (*N ≥ 0*)

- ***template1***
  Customized with no *public* schema and these dedicated YBMT schemas to hold common objects (views, composite types,domains, subprograms, and the like).
  - *extensions*
  - *mgr*
  - *dt_utils*
  - *client_safe*
- ***yugabyte***
  The *bootstrap* database—i.e. the “home base” for the *yugabyte* and *clstr$mgr global* roles. Contains some objects to support configuring a cluster for YBMT and for provisioning *tenant* databases.

- ***tenant databases***
  Must have names like *d0*, *d1*,… *d42*,… Created using *template1*.
  Notice that *local* roles have names like *d0$mgr*, *d0$json_utilities*,…*d0$client*.

Only these specific *global* roles

- ***postgres*** (usually)
  The “bootstrap superuser”. Unavoidable.
  Altered *with nologin password null* (and *no<every other attribute>*).

- ***yugabyte*** (or a name that you like)
  Created *with superuser login password <your secret>* (and *no…*)
  The administrator authorizes as this to configure the cluster for YBMT
  and thereafter only very rarely.
  Might own a couple of *security definer* subprograms.

- ***clstr$mgr***
  Created *with nosuperuser createrole createdb login password <your secret>*.
  The administrator authorizes as this to create/drop *tenant* databases.
  Owns security definer subprograms (in *template1*) for role provisioning in *tenant* databases.

Only these specific *global* roles — *cont*

- ***clstr$developer***

  Created *with* ***no****superuser* ***no****createrole* ***no****createdb* ***no****inherit* ***no****replication* ***no****bypassrls* ***connection limit 0 nologin password null***.

  Has no privilege on any database.

  Is the grantee of all of the native functionality that objects in *pg_catalog* implement.

  Is granted to every *local* role except for the special “client” role.

  NOTE: “all” is revoked on all *pg_catalog* objects from *public*.

```
select exists(
    select 1
    from pg_database
    where has_database_privilege('clstr$developer', datname, 'connect')
    or    has_database_privilege('clstr$developer', datname, 'create')
    or    has_database_privilege('clstr$developer', datname, 'temp')
  )::text;
```

Simple demo

Before starting

- Ensure that you have a sandbox PostgreSQL cluster with no valuable content.
- Ensure that its “bootstrap superuser” is called *postgres*.
  (Else, you’ll have to do a ton of global search-and-replace to use what yours is called.)
- Ensure that there exists superuser called *yugabyte* and a database called *yugabyte* created or altered *with allow_connections true*.
- You can ensure this starting state if the cluster is freshly-created by running this script:

  ysql-case-studies/ybmt-clstr-mgmt/00-post-creation-bootstrap.sql

Simple Demo

- Download and unzip the contents of the **ysql-case-studies** repo.
- Rename the top of the tree to *ysql-case-studies*.
- Open a terminal window on the *ysql-case-studies/ybmt-clstr-mgmt/minimal-demo* directory.
- Look at the *README.md*.
- Look at the *mini.sql* script and step through it manually, copying-and-pasting into *psql*.
- Start *mini.sql* at the prompt and compare the spooled output for the two steps:

  re-config-clstr.txt and *cr-tenant-db-and-install-app.txt*

  with the reference copies (names have *-0* appended) that the repo brings.

Simple Demo — *cont…* role-name independence

- Find this in the mini.sql script:

```
\set lower_db_no 9
\set upper_db_no 9
\set db d9
\set db_name '\'':db'\''
\set mgr d9$mgr
\set cln d9$client
```

- Replace “***9***” with “***8***” everywhere.

- Run the rest of the script by hand, emphasizing the teaching points.

Simple Demo — *cont…* clstr$mgr proof-of-concept

- Open *clstr$mgr-PoC.sql* and step through it manually.
- Notice the extra steps needed compared to doing the task as a superuser.
- Finally, use the *mgr.drop(role)* encapsulation to emphasize the value of the use of *security definer* subprograms for role management within a tenant database.

- Don’t forget to call out *example-psqlrc.txt*.

Some more detail

Mechanizing database create and drop using psql

- You can't execute *create database* or *drop database* from PL/pgSQL. The attempt causes: *"25001: CREATE DATABASE cannot be executed from a function"* (and similar for *drop*).

- But we need both to drop *tenant* databases in a loop and to create them in a loop when the bounds (for example *d17* through *d29*) are provided at run-time.

- We use a common paradigm: use a table function to write a script, where each *drop* or *create* statement is an explicit SQL statement. Spool the script to a file on */tmp* using *\o*. Then execute that script using *\ir*.

- Demo: manually *\set lower_db_no 7* and *\set upper_db_no 9*

- Open *02-drop-and-re-create-tenant-databases.sql* and step through it manually.

Security definer subprograms for role maintenance (see the README)

- cr_role()
- drop_role()
- drop_all_regular_local_roles()
- set_role_search_path()
- set_role_password()
- set_role()
- revoke_all_from_public()
- grant_priv()
- prepend_to_current_search_path()

The “zero-privileged” client role

- Look for *Implementing the principle of least privileges for “client” roles* in the *README.md* on the *ybmt-clstr-mgmt* directory.
- Look at *06-xfer-schema-grants-from-public-to-clstr-developer.sql.*
- Then look at *10-cr-set-up-tenant-database.sql* and *09-cr-tenant-role-mgmt-procs.*

Utilities—esp. user-friendly catalog views and table functions

- Look for *The join views for the pg_catalog tables and the table functions wrappers for these* in *ysql-case-studies/ybmt-clstr-mgmt/README.md*.

- Do this in any *tenant* database:

```
select name from mgr.catalog_views_and_tfs order by kind, rank;
```

Example: the *all_schema_objects* view

```
select owner, kind, name from all_schema_objects where schema = 'extensions'
order by 1, 2, 3;
```

```
select owner, kind, name from all_schema_objects where schema = 'mgr'
order by 1, 2, 3;
```

```
select owner, kind, name from all_schema_objects where schema = 'dt_utils'
order by 1, 2, 3;
```

```
select owner, kind, name from all_schema_objects where schema = 'client_safe'
order by 1, 2, 3;
```

```
select count(*) from all_schema_objects
where schema = any(array['mgr', 'dt_utils', 'client_safe']);
```

Case-studies

- analyzing-covid-data-with-aggregate-functions
- date-time-utilities
- **hard-shell**
- json-relational-equivalence
- recursive-cte
  - basics
    - procedural-implementation-of-recursive-cte-algorithm
    - fibonacci
  - employee-hierarchy
  - **bacon-numbers**
- triggers
  - trigger-firing-order

Thank You

Join us on Slack:
www.yugabyte.com/slack

Star us on GitHub:
github.com/yugabyte/yugabyte-db

The ysql-case-studies repo:
github.com/YugabyteDB-Samples/ysql-case-studies

yugabyteDB