Optimizing Initial Sync for Large Databases in RDS PostgreSQL using pg_dump in Logical Replication

May10 2023

PostgreSQL provides a built-in logical replication system based on the publish/subscribe model, where a publisher publishes data changes to a publication, and a subscriber subscribes to the publication and receives the changes. Logical replication can be configured and managed using SQL commands and functions or third-party tools and libraries.

The user creates the publication on the source PostgreSQL server and adds the required tables to the publication. Each table needs to have a replica identity (primary key or unique key) to replicate the updates and deletes on the source tables to the destination. Otherwise, the updates and deletes will not be replicated but inserts will get replicated.

When using AWS RDS PostgreSQL or any managed service, it’s not possible to employ physical backup tools like pg_basebackup to configure logical replication to community PostgreSQL due to the lack of low-level file system access. For large databases in the terabyte range, the initial sync process with logical replication can be time-consuming. However, an efficient solution to optimize the synchronization process is to use the logical backup tool pg_dump. This PostgreSQL utility creates a logical backup of the entire database or selected database objects that can be used for the initial sync of data. If this is the initial configuration of logical replication, during the time of the restoration we can tune the configuration parameters and make the recovery faster.

Steps to configure the logical replication using the pg_dump backup

Enable logical replication on the source

Make sure the below configuration parameters are set, set the parameters, and restart the cluster.

```
max_replication_slots  | 10
max_wal_senders        | 10
track_commit_timestamp |
on
wal_level              | logical
```

Create and add tables to the publication

```
create publication logical_rep01;
alter publication logical_rep01
add table employee;
alter publication logical_rep01
add table dept;
```

Create replication slot

Create a replication connection to the database and create a replication slot. This session needs to be created in a different terminal and needs to be in an active state till the logical backup is completed.(I prefer to use screen session)

```
[sql@test (terra.c dev) ~]$ psql -h 10.12.2.1 -U postgres "dbname=postgres replication=database"
psql (12.14, server 12.13)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES128-GCM-SHA256, bits: 128,
compression: off)
Type "help" for help.

postgres=> CREATE_REPLICATION_SLOT logical_rep01 LOGICAL pgoutput;
   slot_name    | consistent_point |    snapshot_name    | output_plugin
-----------------+------------------+---------------------+---------------
logical_rep01 | 7/1F0E3938       | 00000005-00062AB3-1 | pgoutput
(1 row)
```

Backup the database

- Take a logical backup of the source(primary) database using pg_dump.
- use the directory format without compression to make the backup faster.

```
pg_dump -h <hostname> -U postgres -t employee -t dept -Fd -f backup -j 2 --no-publications --no-subscriptions --snapshot=00000005-00062AB3-1 --
compress==0 -v postgres
```

- `-Fd` : This specifies the backup format. `-Fd` specifies a directory format.
- `-f backup` : This specifies the name of the backup directory. In this case, the directory will be named “backup”.
- `-j 2` : This specifies the number of parallel jobs to run. In this case, two jobs will run in parallel to speed up the backup process.
- `--no-publications --no-subscriptions` : This specifies that publications and subscriptions should be excluded from the backup.

- `--snapshot=00000005-00062AB3-1` : This specifies the snapshot ID to use for the backup. This is useful when using a PostgreSQL cluster with a replication setup.
- `--compress=0` : This specifies the level of compression to use for the backup. `0` means no compression.
- `-v postgres` : This specifies the name of the database to backup. In this case, the database is named “postgres”.

Close the session created earlier for the replication slot but save the output.

Restore the backup

```
pg_restore -p 5433 -h <hostname> -U postgres -Fd -j 2 -v -d postgres backup
```

Create subscription

Create the subscription in standby which refers to the primary database in disabled state.

```
CREATE SUBSCRIPTION logical_sub01 CONNECTION 'host=hostname port=5432 dbname=postgres user=postgres password=*****'
PUBLICATION logical_rep01
WITH (
  copy_data = false,
  create_slot = false,
  enabled = false,
  connect = true,
  slot_name = 'logical_rep01'
);
```

Advance the replication origin

Take the external ID and increase the position of the replay to the one captured during the replication slot creation time. This requires two arguments one is the external id which is available in pg_subscription and another one is the consistent point which is in the output of the “create replication slot” statement.

```
postgres=# SELECT 'pg_'||oid::text AS "external_id" FROM pg_subscription WHERE subname = 'logical_sub01';
external_id
-------------
pg_82699
(1 row)
```

Take the position of the LSN from the replication slot creation and make it advance to start the replication<

```
postgres=# SELECT pg_replication_origin_advance('pg_82699', '7/1F0E3938') ;
pg_replication_origin_advance
-------------------------------

(1 row)
```

Enable the subscription

```
postgres=# ALTER SUBSCRIPTION logical_sub01 ENABLE;
ALTER SUBSCRIPTION
```

Check the status of the replication

```
select * from pg_stat_replication;
```

We appreciate you taking the time to explore the content above. I hope the information served the reason for which you sought out the blog.

Comments on how to make the blog better are indeed very appreciated. If you have any questions, suggestions, or criticism, kindly email us.

To be informed about all of our content, subscribe now