https://hevodata.com/learn/postgresql-logical-replication/

**CREATE PUBLICATION adds a new publication into the \*\*\*current\*\*\* database**. The publication name must be distinct from the name of any existing publication in the current database.

Each publication exists in only one database.

(1) Creating a publication at source database winn\_la\_1 (192.168.0.48)

$sudo -u postgres psql

(2) Connect to the database (e.g., winn\_la\_1)

postgresql=# \c winn\_la\_1 => connect to the DB

(3) Create publication:

Winn\_la\_1=# CREATE PUBLICATION winn\_publication\_all FOR ALL TABLES;

Use SELECT \* FROM pg\_catalog.pg\_subscription; to verify the publication was created.

(4) On the replica/subscriber machine: Ensure all the tables (from source DB) have been recreated at subscriber node, and they should be \*\*empty\*\*.

(5) Create subscription and give it the name pi\_subscription:

CREATE SUBSCRIPTION adds a new subscription for the \*\*\*current\*\*\* database. The subscription name must be

distinct from the name of any existing subscription in the database.

$sudo -u postgres psql

postgres=# \c winn\_la\_1 /\* You are now connected to database "powerworks\_d1" as user "postgres".

Winn\_la\_11=# CREATE SUBSCRIPTION pi\_subscription

CONNECTION 'postgresql://winter:winter@192.168.0.48:5432/winn\_la\_1'

PUBLICATION winn\_publication\_all;

If replication doesn’t seem to be working, a good first step is checking the PostgreSQL

log on db-replica for any possible errors:

• sudo tail /var/log/postgresql/postgresql-13-main.log

# **How to set up logical replication with Postgresql**

<https://www.digitalocean.com/community/tutorials/how-to-set-up-logical-replication-with-postgresql-10-on-ubuntu-18-04>

The process of keeping database copies in sync is called replication

PostgreSQL supports for logical replication, in addition to physical replication. In a **logical replication** scheme, high-level write operations are streamed from a master database server into one or more replica database servers. In a **physical replication** scheme, binary write operations are instead streamed from master to replica, producing a byte-for-byte exact copy of the original content.

In this tutorial, you will configure logical replication with PostgreSQL 10 on two Ubuntu 18.04 servers, with one server acting as the master and the other as the replica.

## **Prerequisites**

To follow this tutorial, you will need:

* Two Ubuntu 18.04 servers, which we’ll name **db-master** and **db-replica**, each set up with a regular user account and sudo privileges. To set these up, follow [this initial server setup tutorial](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-16-04).
* [Private networking enabled](https://www.digitalocean.com/docs/networking/private-networking/quickstart/) on your servers. Private networking allows for communication between your servers without the security risks associated with exposing databases to the public internet.
* PostgreSQL 10 installed on both servers, following Step 1 of [How To Install and Use PostgreSQL on Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-18-04).

## **Step 1 — Configuring PostgreSQL for Logical Replication**

There are several configuration settings you will need to modify to enable logical replication between your servers. First, you’ll configure Postgres to listen on the private network interface instead of the public one, as exposing data over the public network is a security risk. Then you’ll configure the appropriate settings to allow replication to **db-replica**.

On **db-master**, open /etc/postgresql/13/main/postgresql.conf, the main server configuration file:

$ sudo vi /etc/postgresql/13/main/postgresql

Change:

...

#listen\_addresses = 'localhost' # what IP address(es) to listen on;

...

...

listen\_addresses = 'localhost, db\_replica\_ip\_address' (? Replica\_ip\_address), using \* for all

...

This makes **db-master** listen for incoming connections on the private network in addition to the loopback interface.

Next, find the following line:

sudo vi /etc/postgresql/13/main/postgresql.conf

...

#wal\_level = replica # minimal, replica, or logical

...

Uncomment it, and change it to set the PostgreSQL [*Write Ahead Log*](https://www.postgresql.org/docs/current/static/wal-intro.html) (WAL) level to logical. This increases the volume of entries in the log, adding the necessary information for extracting discrepancies or changes to particular data sets:

/etc/postgresql/12/main/postgresql.conf

...

wal\_level = logical

...

The entries on this log will be consumed by the replica server, allowing for the replication of the high-level write operations from the master.

Save the file and close it.

Next, let’s set our firewall rules to allow traffic from **db-replica** to port 5432 on **db-master**:

* sudo ufw allow from db\_replica\_private\_ip\_address to any port 5432

Finally, restart the PostgreSQL server for the changes to take effect:

* sudo systemctl restart postgresql

With your configuration set to allow logical replication, you can now move on to creating a database, user role, and table.

## **Step 2 — Setting Up a Database, User Role, and Table**

To test the functionality of your replication settings, let’s create a database, table, and user role. You will create an example database with a sample table, which you can then use to test logical replication between your servers. You will also create a dedicated user and assign them privileges over both the database and the table.

First, open the [psql prompt](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-18-04" \l "using-postgresql-roles-and-databases) as the **postgres** user with the following command on both **db-master** and **db-replica**:

sudo -u postgres psql

Create a new table called widgets with arbitrary fields on both hosts:

CREATE TABLE widgets

(

id SERIAL,

name TEXT,

price DECIMAL,

CONSTRAINT widgets\_pkey PRIMARY KEY (id)

);

## **Step 3 — Setting Up a Publication**

Publications are the mechanism that PostgreSQL uses to make tables available for replication. The database server will keep track internally of the connection and replication status of any replica servers associated with a given publication. On **db-master**, you will create a publication, my\_publication, that will function as a master copy of the data that will be sent to your subscribers — in our case, **db-replica**.

On **db-master**, create a publication called my\_publication:

* CREATE PUBLICATION my\_publication;

Add the widgets table you created previously to it:

* ALTER PUBLICATION my\_publication ADD TABLE widgets;

With your publication in place, you can now add a subscriber that will pull data from it.

## **Step 4 — Creating a Subscription**

Subscriptions are used by PostgreSQL to connect to existing publications. A publication can have many subscriptions across different replica servers, and replica servers can also have their own publications with subscribers. To access the data from the table you created on **db-master**, you will need to create a subscription to the publication you created in the previous step, my\_publication.

On **db-replica**, let’s create a subscription called my\_subscription. The CREATE SUBSCRIPTION command will name the subscription, while the CONNECTION parameter will define the connection string to the publisher. This string will include the master server’s connection details and login credentials, including the username and password you defined earlier, along with the name of the example database. Once again, remember to use **db-master**’s private IP address, and replace my\_password with your own password:

* CREATE SUBSCRIPTION my\_subscription CONNECTION 'host=db\_master\_private\_ip\_address port=5432 password=my\_password user=sammy dbname=example' PUBLICATION my\_publication;

You will see the following output confirming the subscription:

NOTICE: created replication slot "my\_subscription" on publisher

CREATE SUBSCRIPTION

Upon creating a subscription, PostgreSQL will automatically sync any pre-existing data from the master to the replica. In our case there is no data to sync since the widgets table is empty, but this is a useful feature when adding new subscriptions to an existing database.

With a subscription in place, let’s test the setup by adding some demo data to the widgets table.

## **Step 4 — Creating a Subscription**

Subscriptions are used by PostgreSQL to connect to existing publications. A publication can have many subscriptions across different replica servers, and replica servers can also have their own publications with subscribers. To access the data from the table you created on **db-master**, you will need to create a subscription to the publication you created in the previous step, my\_publication.

On **db-replica**, let’s create a subscription called my\_subscription. The CREATE SUBSCRIPTION command will name the subscription, while the CONNECTION parameter will define the connection string to the publisher. This string will include the master server’s connection details and login credentials, including the username and password you defined earlier, along with the name of the example database. Once again, remember to use **db-master**’s private IP address, and replace my\_password with your own password:

* CREATE SUBSCRIPTION my\_subscription CONNECTION 'host=db\_master\_private\_ip\_address port=5432 password=my\_password user=sammy dbname=example' PUBLICATION my\_publication;

You will see the following output confirming the subscription:

NOTICE: created replication slot "my\_subscription" on publisher

CREATE SUBSCRIPTION

Upon creating a subscription, PostgreSQL will automatically sync any pre-existing data from the master to the replica. In our case there is no data to sync since the widgets table is empty, but this is a useful feature when adding new subscriptions to an existing database.

With a subscription in place, let’s test the setup by adding some demo data to the widgets table.

## **Step 5 — Testing and Troubleshooting**

To test replication between our master and replica, let’s add some data to the widgets table and verify that it replicates correctly.

On **db-master**, insert the following data on the widgets table:

* INSERT INTO widgets (name, price) VALUES ('Hammer', 4.50), ('Coffee Mug', 6.20), ('Cupholder', 3.80);

On **db-replica**, run the following query to fetch all the entries on this table:

* SELECT \* FROM widgets;

You should now see:

Output

id | name | price

----+------------+-------

1 | Hammer | 4.50

2 | Coffee Mug | 6.20

3 | Cupholder | 3.80

(3 rows)

Success! The entries have been successfully replicated from **db-master** to **db-replica**. From now on, all INSERT, UPDATE, and DELETE queries will be replicated across servers unidirectionally.

One thing to note about write queries on replica servers is that they are not replicated back to the master server. PostgreSQL currently has limited support for resolving conflicts when the data between servers diverges. If there is a conflict, the replication will stop and PostgreSQL will wait until the issue is manually fixed by the database administrator. For that reason, most applications will direct all write operations to the master server, and distribute reads among available replica servers.

You can now exit the psql prompt on both servers:

* \q

Now that you have finished testing your setup, you can add and replicate data on your own.

## **Troubleshooting**

If replication doesn’t seem to be working, a good first step is checking the PostgreSQL log on **db-replica** for any possible errors:

* tail /var/log/postgresql/postgresql-10-main.log

Here are some common problems that can prevent replication from working:

* Private networking is not enabled on both servers, or the servers are on different networks;
* **db-master** is not configured to listen for connections on the correct private network IP;
* The Write Ahead Log level on **db-master** is incorrectly configured (it must be set to logical);
* **db-master** is not configured to accept incoming connections from the correct **db-replica** private IP address;
* A firewall like UFW is blocking incoming PostgreSQL connections on port 5432;
* There are mismatched table names or fields between **db-master** and **db-replica**;
* The sammy database role is missing the required permissions to access the example database on **db-master**;
* The sammy database role is missing the REPLICATION option on **db-master**;
* The sammy database role is missing the required permissions to access the widgets table on **db-master**;
* The table wasn’t added to the publication on **db-master**.

After resolving the existing problem(s), replication should take place automatically. If it doesn’t, use following command to remove the existing subscription before recreating it:

* DROP SUBSCRIPTION my\_subscription;

## **Conclusion**

In this tutorial you’ve successfully installed PostgreSQL 10 on two Ubuntu 18.04 servers and configured logical replication between them.

You now have the required knowledge to experiment with horizontal read scaling, high availability, and the geographical distribution of your PostgreSQL database by adding additional replica servers.

To learn more about logical replication in PostgreSQL 10, you can read the [chapter on the topic](https://www.postgresql.org/docs/10/static/logical-replication.html) on the official PostgreSQL documentation, as well as the manual entries on the [CREATE PUBLICATION](https://www.postgresql.org/docs/10/static/sql-createpublication.html) and [CREATE SUBSCRIPTION](https://www.postgresql.org/docs/10/static/sql-createsubscription.html) commands.