To synchronize PostgreSQL databases between different servers, you can follow these detailed steps, utilizing various methods such as replication or data synchronization tools. Here’s a structured approach to achieve effective database sync:

## Step 1: Define Your Synchronization Requirements

* Identify Objectives: Determine whether you need real-time synchronization, periodic updates, or one-time data migration.
* Assess Data Volume: Understand the amount of data to be synchronized and the frequency of changes.

## Step 2: Choose a Synchronization Method

You can choose between several methods based on your requirements:

## 2.1. PostgreSQL Replication

* Physical Replication: This involves creating a replica server that continuously receives data from the primary (master) server.
* Logical Replication: Allows selective replication of specific tables or schemas and can be set up for more flexible use cases.

## 2.2. Data Synchronization Tools

* pgsync: A command-line tool designed for fast synchronization of PostgreSQL databases. It allows syncing specific tables and rows, handles schema differences, and supports parallel transfers

[1](https://github.com/ankane/pgsync)

.

* DBSync or DBConvert: These tools provide GUI-based solutions for syncing databases with options for insert, update, and delete operations

[3](https://dbconvert.com/blog/what-is-database-synchronization/)

.

## Step 3: Set Up PostgreSQL Replication (If Chosen)

## 3.1. Configure the Primary Server

* Edit the postgresql.conf file to enable replication:

text

wal\_level = replica  
max\_wal\_senders = 3

* Configure pg\_hba.conf to allow replication connections:

text

host replication all <replica\_ip>/32 md5

## 3.2. Set Up the Replica Server

* Install PostgreSQL and configure it to follow the primary server.
* Create a base backup from the primary:

bash

pg\_basebackup -h <primary\_ip> -D /var/lib/postgresql/data -U <replication\_user> -P --wal-method=stream

## 3.3. Start the Replica Server

* Create a recovery.conf file in the data directory with:

text

standby\_mode = 'on'  
primary\_conninfo = 'host=<primary\_ip> port=5432 user=<replication\_user> password=<password>'  
trigger\_file = '/tmp/postgresql.trigger'

## Step 4: Using pgsync for Data Synchronization

If you prefer using a tool like pgsync, follow these steps:

## 4.1. Install pgsync

* Install pgsync using RubyGems:

bash

gem install pgsync

## 4.2. Configure pgsync

* Create a .pgsync.yml configuration file to define source and target databases:

text

# .pgsync.yml  
from:   
 database: source\_db  
to:   
 database: target\_db

## 4.3. Sync Data

* Run pgsync commands to sync specific tables:

bash

pgsync table\_name *# Syncs the specified table from source to target.*

* Use options like --preserve or --truncate based on your needs.

## Step 5: Monitor Synchronization Process

* Regularly check logs and status of replication or synchronization processes.
* Use monitoring tools like pgAdmin or custom scripts to ensure data integrity.

## Step 6: Automate Synchronization (If Applicable)

* For ongoing synchronization, consider setting up cron jobs or using built-in scheduling features in your chosen tool (like ThoughtSpot DataFlow)

[4](https://docs.thoughtspot.com/software/latest/dataflow-postgresql-sync)

.

## Step 7: Validate Data Consistency

* After synchronization, run queries to compare data between source and target databases to ensure consistency.

By following these steps, you can effectively synchronize PostgreSQL databases between servers, ensuring that your data remains consistent and up-to-date across different environments.