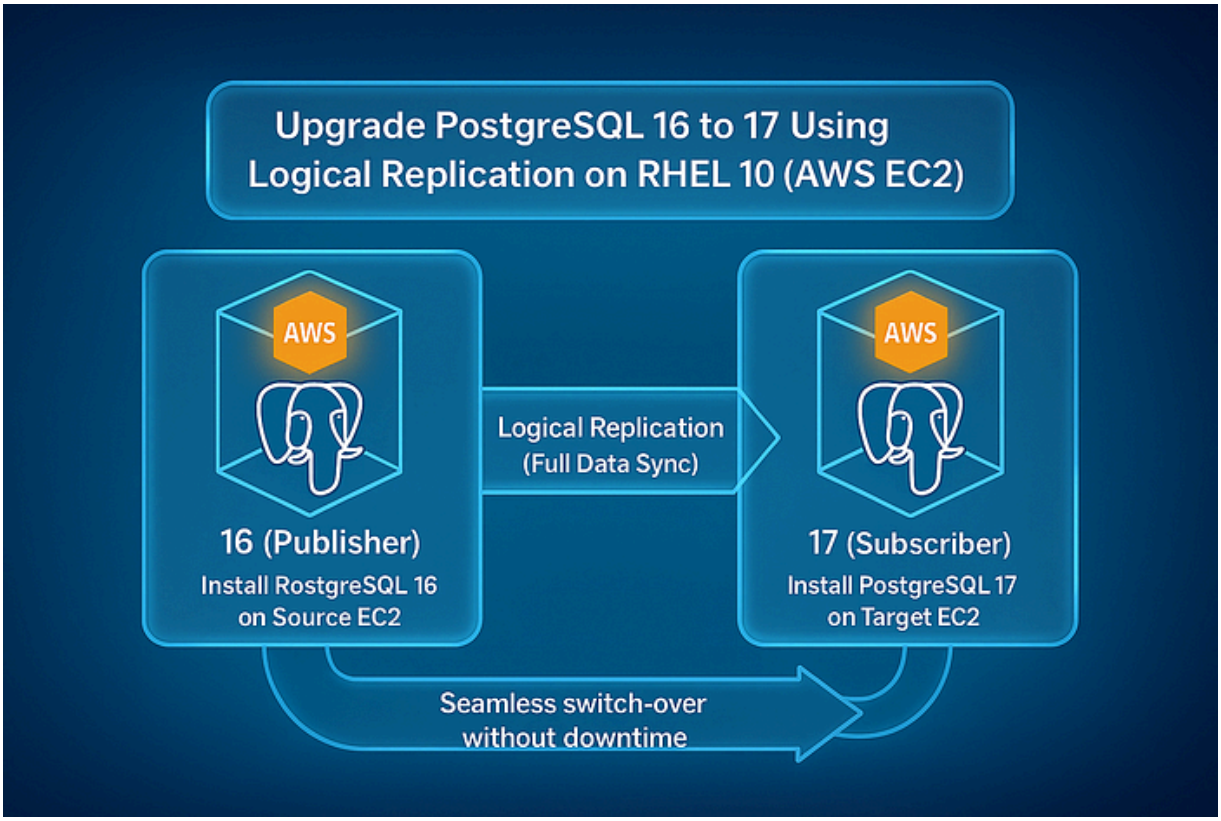
Freedium

< Go to the original



🚀 Upgrade PostgreSQL 16 to 17 Using Logical Replication on RHEL 10 (AWS EC2)

PostgreSQL upgrades are essential for unlocking new features, improving performance, and patching security vulnerabilities. While...

Jeyaram Ayyalusamy

Follow

a11y-light · August 22, 2025 (Updated: August 22, 2025) · Free: Yes

PostgreSQL upgrades are essential for unlocking new features, improving performance, and patching security vulnerabilities. While `pg_upgrade` is the go-to method for in-place upgrades, it may

Freedium

In such scenarios, **logical replication** offers a powerful, low-risk, and flexible upgrade strategy.

This guide walks you through upgrading **PostgreSQL 16 to PostgreSQL 17** using **logical replication** on **AWS EC2 instances** running **RHEL 10**.

📌 What is Logical Replication?

Logical replication in PostgreSQL is a method of copying data changes (INSERTs, UPDATEs, DELETEs) from one database (the publisher) to another (the subscriber) using SQL-based replication mechanisms. Unlike physical replication, which copies disk blocks, logical replication allows for:

- Selective table-level replication
- Cross-version PostgreSQL upgrades
- Bi-directional replication setups (in advanced use cases)
- Fine-tuned control over data flow

This makes it perfect for **zero-downtime upgrades**, **platform migrations**, and **complex HA setups**.

🗺️ Overview: What You'll Achieve

By the end of this guide, you'll have:

- Two PostgreSQL servers running different major versions (16 and 17)

Freedium

- The ability to switch traffic over to PostgreSQL 17 after full data sync — without impacting application uptime

🧭 High-Level Roadmap

Let's outline the core phases of this upgrade strategy:

✅ Phase 1: Infrastructure Setup

- Launch **two EC2 instances** using the RHEL 10 image from AWS Marketplace
- Assign public IPs or configure your VPC/subnet for internal communication

✅ Phase 2: PostgreSQL Installation

- Install **PostgreSQL 16** on the source EC2 instance (publisher)
- Install **PostgreSQL 17** on the target EC2 instance (subscriber)

✅ Phase 3: PostgreSQL Configuration

- Modify `postgresql.conf` and `pg_hba.conf` to enable logical replication
- Allow remote access between instances

✅ Phase 4: Logical Replication Setup

- Create a **publication** on PostgreSQL 16
- Create a **subscription** on PostgreSQL 17
- Validate that data replication is working as expected

Freedium

✅ Phase 5: Cutover and Cleanup

- Point your applications to PostgreSQL 17
- Monitor replication lag and health
- Decommission the old PostgreSQL 16 instance when ready

🧬 Step 1: Launch Two EC2 Instances on RHEL 10 for PostgreSQL Logical Replication (v16 to v17)

PostgreSQL upgrades are crucial for leveraging the latest features, performance optimizations, and critical security patches. While in-place upgrades like `pg_upgrade` are quick, they often involve downtime. If you're aiming for **zero-downtime upgrades**, especially across major versions, **logical replication** is the safest approach.

In this series, we'll walk through how to upgrade from **PostgreSQL 16 to PostgreSQL 17** using **logical replication**, deployed on **AWS EC2 instances running RHEL 10**.

We begin with the foundational step:

🖥️ Step 1: Launch and Prepare Two EC2 Instances with RHEL 10

This step ensures you have two clean, secure, and properly networked environments:

- One for **PostgreSQL 16** (Publisher)
- One for **PostgreSQL 17** (Subscriber)

⚙️ Instance Configuration Guide

Freedium

2. **Launch Two EC2 Instances**

- Click **Launch Instance**
- For **AMI**, search for and select: `Red Hat Enterprise Linux 10 (RHEL 10)`
- Choose instance type:
- `t2.micro` or `t3.small` is sufficient for testing environments
- Configure instance details as needed

3. **Create or Select Key Pair** This is important for SSH access:

- If you don't have a key pair, create a new one and download the `.pem` file

4. **Security Group Configuration** Modify the default security group or create a new one to allow the necessary inbound rules:

- Port Protocol Purpose Source 22 TCP SSH for remote access Your IP address 5432 TCP PostgreSQL replication access Your IP or peer EC2

> ⚠️ **Tip:** Do not open port 5432 to 0.0.0.0/0 in production. Limit to trusted IPs.

🔐 SSH Access Instructions

Once the instances are running, use the key pair you downloaded to connect via SSH. The method varies depending on your operating system.

🪟 For Windows Users

Freedium

1. Convert PEM to PPK with PuTTYgen

- Open **PuTTYgen** (comes with PuTTY)
- Click **Load**, and select your `.pem` key
- Click **Save Private Key** → this creates a `.ppk` file

2. Connect Using PuTTY

- Open PuTTY
- In the **Host Name** field: `ec2-user@<Public-IP-of-EC2>`
- Go to **Connection → SSH → Auth**
- Browse and select your `.ppk` file
- Click **Open**

> *You'll now be connected to the EC2 instance terminal as the `ec2-user`.*

🐧 For Linux/macOS Users

Use the terminal with the `ssh` command:

1. Set Permissions for the PEM File

Copy

```
chmod 400 my-key.pem
```

2. SSH into Your EC2 Instance

Copy

Freedium

> *On successful connection, you'll see a welcome message from RHEL 10 and land in the home directory of* `ec2-user` *.*

🛠️ Optional: Disable SELinux or Firewalld Temporarily

Depending on your security posture and networking setup, you may want to disable SELinux and the firewall (for testing only).

Check SELinux Status:

```
Copy

sestatus
```

To Disable SELinux (temporary workaround):

```
Copy

sudo vi /etc/selinux/config
# Change this line:
SELINUX=disabled
```

Then reboot the system:

```
Copy

sudo reboot
```

Disable Firewalld (for isolated test environments only):

Freedium

```
sudo systemctl stop firewalld
sudo systemctl disable firewalld
```

> ⚠️ *These changes reduce system security. Do not apply them in production unless you understand the risks.*

✅ What You've Accomplished So Far

You now have:

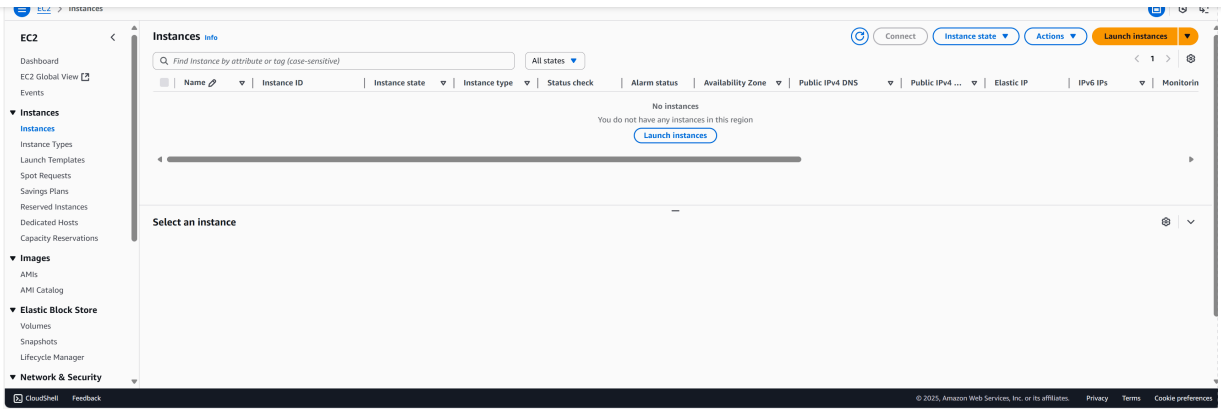
- Two EC2 instances running **RHEL 10**
- Open ports for **SSH (22)** and **PostgreSQL (5432)**
- Working **SSH access** via PuTTY or terminal
- Initial security configurations for connectivity

These machines are now ready for the next step: **installing PostgreSQL 16 and 17** respectively, and configuring the system for replication.

PostgreSQL 16 Installation on EC2 Instance (RHEL 10)

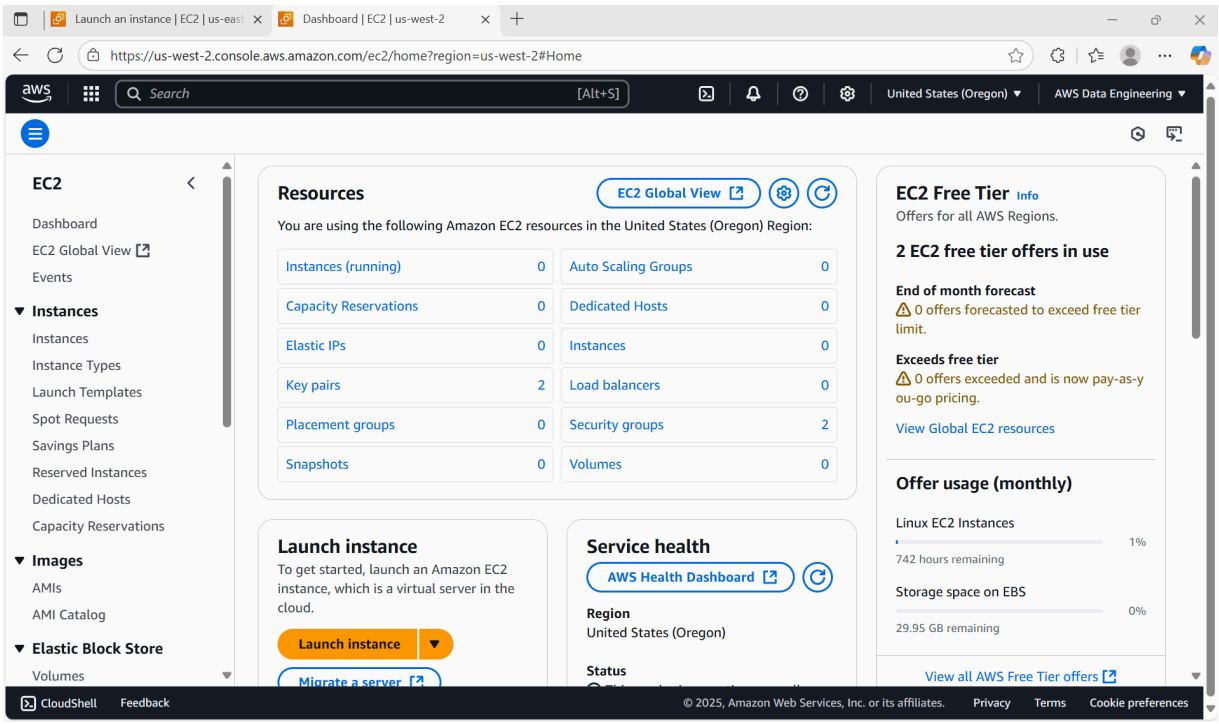
A step-by-step guide to setting up PostgreSQL 16 on an Amazon EC2 instance running Red Hat Enterprise Linux 10 for reliable database deployment.

Freedium



PostgreSQL 17 Installation on EC2 Instance (RHEL 10)

A step-by-step guide to setting up PostgreSQL 17 on an Amazon EC2 instance running Red Hat Enterprise Linux 10 for reliable database deployment.



Configuring Security Group Rules for Streaming Replication on AWS EC2

After launching your EC2 instances, you need to configure **Security Group inbound rules** so that the primary and standby PostgreSQL

Freedium

In the example shown, the following inbound rules were added (highlighted in red in the screenshot):

1. **All ICMP — IPv4 (Source: 0.0.0.0/0)**

- Allows ICMP traffic (ping/traceroute) from any IP.
- Useful for testing connectivity between instances.

2. **SSH (Port 22, Source: 0.0.0.0/0)**

- Enables SSH access to the EC2 instance from anywhere.
- Best practice: Restrict this to your own IP address instead of `0.0.0.0/0` for security.

3. **All TCP (Port range 0–65535, Source: 0.0.0.0/0)**

- Opens all TCP ports from all sources.
- This is very permissive and should only be used temporarily for troubleshooting. In production, restrict only the ports you need.

4. **PostgreSQL (Port 5432, Source: 0.0.0.0/0)**

- Allows PostgreSQL connections from any IP.
- Required for streaming replication between primary and standby.
- Best practice: Instead of `0.0.0.0/0`, restrict the source to your standby server's private IP (for example, `172.31.x.x/32`) to reduce risk.

🐘 Step 2: Installing PostgreSQL 16 and PostgreSQL 17 on RHEL 10 EC2 Instances

database migrations. But before we set up replication, we need to prepare the environment: two PostgreSQL servers on two separate **EC2 instances running RHEL 10.**

In this step, we'll walk through how to install and configure **PostgreSQL 16** on one server (Publisher) and **PostgreSQL 17** on another server (Subscriber), ensuring both are ready for replication setup in the next steps.

🎯 Why This Step Matters

Before replication can begin:

- You need **two separate PostgreSQL environments.**
- These environments must run on **different ports** to avoid conflict.
- Each database server must be initialized, started, and accessible.

🧭 Step 2.1 — Add the PostgreSQL Official YUM Repository

The default RHEL 10 package manager may not include the latest PostgreSQL versions. So we first add the **official PostgreSQL YUM repository** to ensure we can access versions 16 and 17.

Run the following commands on **both EC2 instances**:

Copy

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpr
```

Freedium

Now, disable the default PostgreSQL module that ships with RHEL:

```
Copy

sudo dnf -qy module disable postgresql
```

This prevents version conflicts between the default module and the versions we'll install.

📦 Step 2.2 — Install PostgreSQL 16 on the Publisher Server

Run these commands **only on the first EC2 instance**, which will serve as the **Publisher** (source of data for replication):

```
Copy

# Install PostgreSQL 16
sudo dnf install -y postgresql16-server
# Initialize the database cluster
sudo /usr/pgsql-16/bin/postgresql-16-setup initdb
# Enable the service to start on boot
sudo systemctl enable postgresql-16
# Start PostgreSQL 16
sudo systemctl start postgresql-16
```

✅ Verify PostgreSQL 16 is Running

Run:

```
Copy

sudo systemctl status postgresql-16
```

Freedium

🔧 Step 2.3 — Install PostgreSQL 17 on the Subscriber Server

Now switch to your **second EC2 instance** — the **Subscriber** — and repeat a similar process for PostgreSQL 17:

```
Copy

# Install PostgreSQL 17
sudo dnf install -y postgresql17-server
# Initialize PostgreSQL 17
sudo /usr/pgsql-17/bin/postgresql-17-setup initdb
# Enable the service and start it immediately
sudo systemctl enable --now postgresql-17
```

✅ Confirm PostgreSQL 17 is Running

Run:

```
Copy

sudo systemctl status postgresql-17
```

Again, verify the service shows as **active (running)**.

PostgreSQL Installation Guide

The PostgreSQL installation steps are included in the above videos while launching the EC2 instance. If you prefer, you can watch the video below for installing PostgreSQL 17.

Freedium

🚪 Step 2.4 — Change PostgreSQL 17 Port to Avoid Conflict

Both PostgreSQL 16 and 17 default to **port 5432**. If you plan to run them simultaneously on the same server or within a testing environment, you must change the port for PostgreSQL 17 to **5433**.

Step-by-step:

1. Open the PostgreSQL 17 config file:

```
Copy

sudo vi /var/lib/pgsql/17/data/postgresql.conf
```

2. Find the line:

```
Copy

#port = 5432
```

Uncomment and modify it:

```
Copy

port = 5432
```

Freedium

```
Copy

listen_addresses = '*'
```

4. Save and exit the file.

Restart PostgreSQL 17 to apply changes:

```
Copy

sudo systemctl restart postgresql-17
```

🔎 Confirm It's Listening on Port 5433

Run:

```
Copy

sudo ss -ltnp | grep 5432
```

You should see a `postgres` process listening on **port 5433.**

✅ Final Validation

To confirm everything is ready:

On the PostgreSQL 16 (Publisher):

```
Copy

sudo systemctl status postgresql-16
```

Freedium

On the PostgreSQL 17 (Subscriber):

```
Copy

sudo systemctl status postgresql-17
```

Both should report "**active (running)**" and listening on their respective ports (5432 for PG16, 5433 for PG17).

⬅️ Summary

By the end of this step, you have:

- Installed PostgreSQL 16 and PostgreSQL 17 on two separate EC2 instances
- Initialized both database clusters
- Configured PostgreSQL 17 to run on a non-default port (5433)
- Verified both services are running and listening correctly

🌐 Step 3: Configure PostgreSQL for Remote Access Between EC2 Instances (RHEL 10)

Freedium

on the other (Subscriber). Before we can begin replication between these servers, PostgreSQL must be configured to **allow remote access**.

By default, PostgreSQL is locked down to local connections (i.e., `localhost`). To enable communication between these two nodes — which is essential for **logical replication** — we need to edit two configuration files on both instances:

- `postgresql.conf`
- `pg_hba.conf`

📁 Why These Files Matter

- **`postgresql.conf`** — Controls how PostgreSQL behaves, including which IPs it listens to and whether logical replication is allowed.
- **`pg_hba.conf`** — Controls which clients can connect to the database, and what authentication methods they must use.

Without proper changes here, your replication setup will fail — even if the database services themselves are running fine.

🔧 1. Edit `postgresql.conf` to Allow Remote Connections

📍 Location of File

- PostgreSQL 16: `/var/lib/pgsql/16/data/postgresql.conf`
- PostgreSQL 17: `/var/lib/pgsql/17/data/postgresql.conf`

> 🔑 *You must edit this file on **both** the publisher and subscriber*

Freedium

🛠️ What to Change

Open the file using `vi` or any preferred editor:

```
Copy

vi /var/lib/pgsql/16/data/postgresql.conf
```

Find and **update or add** the following lines:

```
Copy

listen_addresses = '*'
wal_level = logical
max_wal_senders = 10
max_replication_slots = 10
 grep -E "wal_level|listen_addresses|max_wal_senders|max_replication_slo
 /var/lib/pgsql/16/data/postgresql.conf
```

Output:

```
Copy

[postgres@ip-172-31-25-61 ~]$ grep -E "wal_level|listen_addresses|max_wa
 /var/lib/pgsql/16/data/postgresql.conf

listen_addresses = '*'                  # what IP address(es) to listen
wal_level = logical                     # minimal, replica, or logical
max_wal_senders = 10            # max number of walsender processes
max_replication_slots = 10      # max number of replication slots
[postgres@ip-172-31-25-61 ~]$
```

- PostgreSQL 17:

```
Copy
```

Freedium

Find and **update or add** the following lines:

```
listen_addresses = '*'
wal_level = logical
max_wal_senders = 10
max_replication_slots = 10
```

output:

```
[postgres@ip-172-31-25-251 ~]$ grep -E "wal_level|listen_addresses|max_wa
 /var/lib/pgsql/17/data/postgresql.conf
listen_addresses = '*'          # what IP address(es) to listen on;
wal_level = logical                     # minimal, replica, or logical
max_wal_senders = 10            # max number of walsender processes
max_replication_slots = 10      # max number of replication slots
[postgres@ip-172-31-25-251 ~]$
```

🧠 Explanation of Each Parameter

- `listen_addresses = '*'` Tells PostgreSQL to accept connections from **any IP address**, not just `localhost`. This is **essential** for remote access.
- `wal_level = logical` Required for **logical replication**. It ensures that PostgreSQL writes enough information to its Write-Ahead Logs (WAL) to replicate individual table changes.
- `max_wal_senders = 10` Sets how many WAL sender processes PostgreSQL can run. Each replication connection uses one. `10` is generally safe for most use cases.

Freedium

> 🔁 *Repeat the same steps and edits for the **PostgreSQL 17** instance.*

🔐 2. Edit `pg_hba.conf` to Allow the Remote Instance to Connect

📍 Location of File

- PostgreSQL 16: `/var/lib/pgsql/16/data/pg_hba.conf`
- PostgreSQL 17: `/var/lib/pgsql/17/data/pg_hba.conf`

➕ Add the Remote IP for Access

Open the file:

```
Copy

vi /var/lib/pgsql/16/data/pg_hba.conf
```

Then add a new line to allow access from the **subscriber's IP address**:

```
Copy

host    all             all             <peer-IP>/32            scram-sha
```

> 🔁 *Repeat this for PostgreSQL 17 and add the **publisher's IP address** instead.*

```
Copy
```

Freedium

```
# host name, or it is made up of an IP address and a CIDR mask that is
# specifies the number of significant bits in the mask.  A host name
# that starts with a dot (.) matches a suffix of the actual host name.
host    all             all             127.0.0.1/32            scram-sha
host    all             all             ::1/128                 scram-sha
host    replication     all             127.0.0.1/32            scram-sha
host    replication     all             172.31.25.251/32        scram-sha
host    replication     all             ::1/128                 scram-sha
[postgres@ip-172-31-25-61 ~]$
```

Example:

If the Publisher (PostgreSQL 16) has a private IP of `172.31.25.251` and the Subscriber (PostgreSQL 17) has `172.31.25.61`, then:

- On PostgreSQL 16 server:

Copy

```
host    all             all             172.31.25.251/32        scram-sha
```

- On PostgreSQL 17 server:

Copy

```
vi /var/lib/pgsql/17/data/pg_hba.conf
host    all             all             172.31.25.61/32         scram-sha
```

> ✅ *This tells PostgreSQL to allow any user (`all`) to connect to any database (`all`) from the specified IP address, using* **scram-sha-256** *password authentication.*

🔁 3. Restart PostgreSQL Services to Apply Changes

Freedium

On the PostgreSQL 16 instance:

```
Copy

sudo systemctl restart postgresql-16
```

output:

```
Copy

[ec2-user@ip-172-31-25-61 ~]$ sudo systemctl restart postgresql-16
[ec2-user@ip-172-31-25-61 ~]$
```

On the PostgreSQL 17 instance:

```
Copy

sudo systemctl restart postgresql-17
```

output:

```
Copy

[ec2-user@ip-172-31-25-251 ~]$ sudo systemctl restart postgresql-17
[ec2-user@ip-172-31-25-251 ~]$
```

You can confirm the services restarted successfully by checking their status:

```
Copy
```

Freedium

output for PostgreSQL 16:

Copy

```
[ec2-user@ip-172-31-25-61 ~]$ sudo systemctl status postgresql-16
● postgresql-16.service - PostgreSQL 16 database server
     Loaded: loaded (/usr/lib/systemd/system/postgresql-16.service; enabl
     Active: active (running) since Fri 2025-08-22 00:20:12 UTC; 55s ago
 Invocation: cb3a94b7c9314bcc87cb23612030d6d4
       Docs: https://www.postgresql.org/docs/16/static/
    Process: 25925 ExecStartPre=/usr/pgsql-16/bin/postgresql-16-check-db
   Main PID: 25932 (postgres)
      Tasks: 7 (limit: 5687)
     Memory: 17.3M (peak: 18.6M)
        CPU: 78ms
     CGroup: /system.slice/postgresql-16.service
             ├─25932 /usr/pgsql-16/bin/postgres -D /var/lib/pgsql/16/data
             ├─25933 "postgres: logger "
             ├─25934 "postgres: checkpointer "
             ├─25935 "postgres: background writer "
             ├─25937 "postgres: walwriter "
             ├─25938 "postgres: autovacuum launcher "
             └─25939 "postgres: logical replication launcher "

Aug 22 00:20:12 ip-172-31-25-61.ec2.internal systemd[1]: Starting postgre
Aug 22 00:20:12 ip-172-31-25-61.ec2.internal postgres[25932]: 2025-08-22
Aug 22 00:20:12 ip-172-31-25-61.ec2.internal postgres[25932]: 2025-08-22
Aug 22 00:20:12 ip-172-31-25-61.ec2.internal systemd[1]: Started postgres
[ec2-user@ip-172-31-25-61 ~]$
```

output for PostgreSQL 17:

Copy

```
[ec2-user@ip-172-31-25-251 ~]$ sudo systemctl status postgresql-17
● postgresql-17.service - PostgreSQL 17 database server
     Loaded: loaded (/usr/lib/systemd/system/postgresql-17.service; enabl
     Active: active (running) since Fri 2025-08-22 00:20:43 UTC; 1min 7s
```

Freedium

```
   Main PID: 10853 (postgres)
      Tasks: 5 (limit: 5687)
     Memory: 18.1M (peak: 19.1M)
        CPU: 99ms
     CGroup: /system.slice/postgresql-17.service
             ├─10853 /usr/pgsql-17/bin/postgres -D /var/lib/pgsql/17/data
             ├─10854 "postgres: logger "
             ├─10855 "postgres: checkpointer "
             ├─10856 "postgres: background writer "
             └─10857 "postgres: startup recovering 000000010000000000000

Aug 22 00:20:43 ip-172-31-25-251.ec2.internal systemd[1]: Starting postgr
Aug 22 00:20:43 ip-172-31-25-251.ec2.internal postgres[10853]: 2025-08-22
Aug 22 00:20:43 ip-172-31-25-251.ec2.internal postgres[10853]: 2025-08-22
Aug 22 00:20:43 ip-172-31-25-251.ec2.internal systemd[1]: Started postgre
[ec2-user@ip-172-31-25-251 ~]$
```

If you see "active (running)" — you're good to go!

🔍 (Optional) Verify Listening Ports and Connections

Use this command to confirm PostgreSQL is actively listening on the correct port and IP address:

Copy

```
sudo ss -ltnp | grep postgres
```

Look for lines like:

Copy

```
[ec2-user@ip-172-31-25-251 ~]$ sudo ss -ltnp | grep postgres
LISTEN 0      200          0.0.0.0:5432      0.0.0.0:*    users:(("postgr
LISTEN 0      200             [::]:5432         [::]:*    users:(("postgr
[ec2-user@ip-172-31-25-251 ~]$
```

Freedium

```
LISTEN 0      200             [::]:5432         [::]:*    users:(("postgr
[ec2-user@ip-172-31-25-61 ~]$
```

That means your PostgreSQL instance is open to external traffic on port 5432 (or 5433, depending on your setup).

✅ Final Check

Try to connect from one instance to the other using `psql` :

Copy

```
psql -U postgres -h <peer-IP> -d postgres
```

If successful, it confirms that both PostgreSQL instances can communicate with each other securely — and you're now ready to set up logical replication in the next step.

Copy

```
[postgres@ip-172-31-25-251 ~]$ psql -U postgres -h 172.31.25.61 -d postgr
Password for user postgres:
psql (17.6, server 16.10)
Type "help" for help.

postgres=#

[postgres@ip-172-31-25-61 ~]$ psql -U postgres -h 172.31.25.251 -d postgr
Password for user postgres:
psql (17.6)
Type "help" for help.

postgres=#
```

Freedium

📌 Summary

In this step, we:

- Enabled PostgreSQL to **listen for remote connections**
- Set critical parameters required for **logical replication**
- Modified `pg_hba.conf` to **allow trusted access**
- Restarted the PostgreSQL services to apply changes

These changes create the foundation for setting up a **logical replication stream** from PostgreSQL 16 to PostgreSQL 17 — allowing for seamless, low-downtime upgrades.

🔐 Step 4 & 🔄 Step 5: Setting Up Logical Replication Between PostgreSQL 16 and 17 on RHEL 10

As part of your PostgreSQL upgrade strategy — especially when zero downtime or cross-version replication is needed — **logical replication** is a powerful tool. In this part of the journey, we'll move beyond installation and configuration, and set up the actual database, user roles, and replication streams between two EC2 instances.

🧩 Overview of This Step

Logical replication in PostgreSQL operates using a **publisher–subscriber** model:

- **Publisher (PostgreSQL 16)**: The source of truth; changes made here are sent out.
- **Subscriber (PostgreSQL 17)**: Receives changes and applies them, keeping data in sync.

Freedium

- Create a common database and table structure.
- Configure user permissions.
- Create a publication on the Publisher.
- Set up a subscription on the Subscriber.

🔐 Step 4: Create Users and Tables for Replication

Let's begin by creating a replication-ready environment on both the Publisher and Subscriber.

🔸 On the Publisher (PostgreSQL 16)

1. Switch to the PostgreSQL user:

```
Copy

sudo su - postgres
psql
```

2. Create a new database for replication:

```
Copy

CREATE DATABASE repdb;
```

This database will host the tables we want to replicate.

```
Copy

[ec2-user@ip-172-31-25-61 ~]$ sudo su - postgres
Last login: Fri Aug 22 01:29:11 UTC 2025 on pts/1
[postgres@ip-172-31-25-61 ~]$

[postgres@ip-172-31-25-61 ~]$ psql
```

Freedium

```
postgres=#
postgres=# CREATE DATABASE repdb;
CREATE DATABASE
postgres=#
postgres=#
```

3. Connect to the new database:

Copy

```
\c repdb
postgres=# \c repdb
You are now connected to database "repdb" as user "postgres".
repdb=#
repdb=#
```

4. Create a sample table:

Copy

```
CREATE TABLE reptable (
    id SERIAL PRIMARY KEY,
    name TEXT,
    price NUMERIC
);
```

This table represents the data that will be replicated to the subscriber.

Copy

```
repdb=# CREATE TABLE reptable (
    id SERIAL PRIMARY KEY,
    name TEXT,
    price NUMERIC
);
```

Freedium

5. Create a replication user:

```
Copy

CREATE ROLE repuser WITH REPLICATION LOGIN PASSWORD 'postgres';
repdb=# CREATE ROLE repuser WITH REPLICATION LOGIN PASSWORD 'postgres';
CREATE ROLE
repdb=#
```

This user will be used by the subscriber to connect and pull data. The role must:

- Be able to log in
- Have the `REPLICATION` attribute

6. Grant privileges:

```
Copy

GRANT ALL PRIVILEGES ON DATABASE repdb TO repuser;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO repuser;
```

This ensures `repuser` can access both the database and its tables.

```
Copy

repdb=# GRANT ALL PRIVILEGES ON DATABASE repdb TO repuser;
GRANT
repdb=#
repdb=# GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO repuser;
GRANT
repdb=#
```

Freedium

> *to be part of the replication stream, remember to grant privileges accordingly or use* `ALTER DEFAULT PRIVILEGES` *.*

🔹 On the Subscriber (PostgreSQL 17)

The subscriber only needs to mirror the schema structure — **data will be replicated automatically.**

1. **Switch to the PostgreSQL user:**

```
Copy

sudo su - postgres
psql
```

2. Create the same database:

```
Copy

CREATE DATABASE repdb;
[ec2-user@ip-172-31-25-251 ~]$ sudo su - postgres
Last login: Fri Aug 22 00:25:27 UTC 2025 on pts/1
[postgres@ip-172-31-25-251 ~]$

[postgres@ip-172-31-25-251 ~]$ psql
psql (17.6)
Type "help" for help.

postgres=#
postgres=# CREATE DATABASE repdb;
CREATE DATABASE
postgres=#
postgres=#
```

3. Connect to the database:

```
Copy
```

Freedium

4. Create the same table structure:

Copy

```
CREATE TABLE reptable (
    id SERIAL PRIMARY KEY,
    name TEXT,
    price NUMERIC
);
```

> ⚠️ *It's important that the schema — column names, data types, and keys — match exactly between the publisher and subscriber.*

Copy

```
repdb=# CREATE TABLE reptable (
    id SERIAL PRIMARY KEY,
    name TEXT,
    price NUMERIC
);
CREATE TABLE
repdb=#
```

🔄 Step 5: Set Up Publication and Subscription

With your environment prepped, let's create the replication streams.

📢 On the Publisher (PostgreSQL 16)

5. Switch to the replication database:

Copy

```
\c repdb
postgres=# \c repdb
```

Freedium

```
repdb=#
```

6. Create a publication:

```
Copy

CREATE PUBLICATION my_publication FOR ALL TABLES;
```

This command tells PostgreSQL to monitor **all tables** in `repdb` and make their changes available for replication.

```
Copy

repdb=# CREATE PUBLICATION my_publication FOR ALL TABLES;
CREATE PUBLICATION
repdb=#
```

> ✅ *If you only want to replicate specific tables:*

```
Copy

CREATE PUBLICATION my_publication FOR TABLE reptable;
```

📥 On the Subscriber (PostgreSQL 17)

1. Switch to the replication database:

```
Copy

\c repdb
```

2. Create the subscription:

```
Copy
```

Freedium

```
  PUBLICATION my_publication;
```

> *Replace `<PUBLISHER_IP>` (172.31.25.61) with the IP address of your PostgreSQL 16 server.*

Copy

```
repdb=# CREATE SUBSCRIPTION my_subscription
  CONNECTION 'host=172.31.25.61 port=5432 user=repuser password=postgres
  PUBLICATION my_publication;
NOTICE:  created replication slot "my_subscription" on publisher
CREATE SUBSCRIPTION
repdb=#
```

Once this command runs:

- PostgreSQL will establish a connection to the publisher.
- It will copy the initial snapshot of the tables.
- Future changes (INSERT, UPDATE, DELETE) will be streamed and applied live.

🧪 Verification

After setting up the subscription, test it:

1. **Insert a few rows on the Publisher:** (PostgreSQL 16)

Copy

```
INSERT INTO reptable (name, price) VALUES ('Coffee', 2.99), ('Tea', 1.99
repdb=# INSERT INTO reptable (name, price) VALUES ('Coffee', 2.99), ('Tea
INSERT 0 3
repdb=#
repdb=# select * from reptable;
```

Freedium

```
  2 | Tea    |  1.99
  3 | Cake   |  3.49
(3 rows)

repdb=#
```

2. Query the same table on the Subscriber:

Copy

```
SELECT * FROM reptable;
```

> ✅ *If all went well, you should see the same rows replicated almost instantly!*

Copy

```
repdb=# select * from reptable;
 id |  name  | price
----+--------+-------
  1 | Coffee |  2.99
  2 | Tea    |  1.99
  3 | Cake   |  3.49
(3 rows)

repdb=#
```

📝 Summary

At this point, you have:

- Created a **replication-ready database and user** on PostgreSQL 16.
- Mirrored the table schema on PostgreSQL 17.
- Created a **publication** to broadcast changes.

Freedium

This forms the backbone of logical replication, which is an excellent upgrade path between major PostgreSQL versions with minimal downtime and maximum flexibility.

✅ Step 6: Verifying Logical Replication Between PostgreSQL 16 and 17 on RHEL 10

Upgrading PostgreSQL with zero downtime is a big deal — and logical replication is one of the most effective methods to achieve it. But all the configuration and setup won't mean much unless **replication actually works.**

In this step, we'll **verify** that our logical replication setup from PostgreSQL 16 (the **Publisher**) to PostgreSQL 17 (the **Subscriber**) is working correctly by inserting test data and checking for immediate synchronization.

🧪 Why This Step Matters

After completing configuration, user creation, publication and subscription, it's time to validate the results. Logical replication should mirror data changes from the Publisher to the Subscriber in near real time. This test ensures your replication is:

- Fully active
- Properly configured
- Free of permissions or connectivity issues
- Syncing the intended tables correctly

Let's walk through this final check.

Freedium

🔸 Step 1: Insert Sample Data into the Publisher (PostgreSQL 16)

We'll simulate a typical change by inserting some data into the table on the **Publisher**.

🔐 Connect to PostgreSQL 16

SSH into your EC2 instance running PostgreSQL 16 and switch to the `postgres` user:

Copy

```
ssh -i my-key.pem ec2-user@<publisher-public-ip>
sudo su - postgres
psql
```

Then connect to your replication database:

Copy

```
\c repdb
```

🍰 Insert Sample Rows into `reptable`

Use the following SQL to insert two records:

Copy

```
INSERT INTO reptable (name, price) VALUES ('Apple Pie', 5.99);
INSERT INTO reptable (name, price) VALUES ('Muffin', 3.25);
repdb=# INSERT INTO reptable (name, price) VALUES ('Apple Pie', 5.99);
INSERT 0 1
repdb=#
repdb=# INSERT INTO reptable (name, price) VALUES ('Muffin', 3.25);
INSERT 0 1
repdb=#
```

Freedium

🔹 Step 2: Query the Subscriber (PostgreSQL 17)

Next, let's check if those new records made it to the **Subscriber** instance.

🔐 Connect to PostgreSQL 17

SSH into your **PostgreSQL 17** EC2 instance (Subscriber) and log in as `postgres`:

Copy

```
ssh -i my-key.pem ec2-user@<subscriber-public-ip>
sudo su - postgres
psql
```

Connect to the replication database:

Copy

```
\c repdb
postgres=# \c repdb
You are now connected to database "repdb" as user "postgres".
repdb=#
```

📊 View the Contents of `reptable`

Run the following SQL command:

Copy

```
SELECT * FROM reptable;
```

Freedium

✅ Expected Output

You should see something like:

Copy

```
repdb=# SELECT * FROM reptable;
 id |   name    | price
----+-----------+-------
  1 | Coffee    |  2.99
  2 | Tea       |  1.99
  3 | Cake      |  3.49
  4 | Apple Pie |  5.99
  5 | Muffin    |  3.25
(5 rows)

repdb=#
```

If the rows inserted on the Publisher are now visible on the Subscriber, **congratulations!** 🎉 Your logical replication setup is fully functional.

🛠️ What to Do If It Didn't Work

If you don't see any rows, or only see a partial result:

✅ Basic Troubleshooting Checklist

- Check the subscription status:

Copy

```
SELECT * FROM pg_stat_subscription;
repdb=# SELECT * FROM pg_stat_subscription;
 subid |     subname     | worker_type |  pid  | leader_pid | relid | re
-------+-----------------+-------------+-------+------------+-------+---
 16960 | my_subscription | apply       | 11600 |            |       | 0/
(1 row)
```

Freedium

- **Ensure schema compatibility:** Column names, types, and primary keys must match exactly on both sides.
- **Verify** `wal_level = logical` on the Publisher.
- **Check** `pg_hba.conf` to make sure the Subscriber's IP is allowed to connect.
- **Confirm port accessibility (usually 5432)** through Security Groups or firewalls.
- **Inspect logs** on both servers for errors:

```
Copy

journalctl -u postgresql-16 journalctl -u postgresql-17
```

- **Restart PostgreSQL services** if config changes were made but not applied:

```
Copy

sudo systemctl restart postgresql-16
sudo systemctl restart postgresql-17
```

🔁 Why Logical Replication is Powerful

This test isn't just about verifying setup. It proves that logical replication can:

- **Handle live data changes**
- **Work across different PostgreSQL versions**
- **Enable in-place upgrades with zero downtime**
- **Allow selective table-level replication**

Freedium

incredibly valuable feature.

✅ Final Thoughts

By verifying logical replication with actual data, you've completed the most crucial step in your zero-downtime PostgreSQL upgrade journey. At this point:

- Your PostgreSQL 16 Publisher is feeding real-time changes to your PostgreSQL 17 Subscriber.
- You've preserved business continuity with no downtime or user interruption.
- You're ready to begin migration, switchover, or scaling strategies.

🧹 Step 7: Cleanup and Switchover — Finalizing Your PostgreSQL Logical Replication Upgrade

You've come a long way in your PostgreSQL upgrade journey — from setting up EC2 instances on RHEL 10, configuring logical replication between PostgreSQL 16 and 17, and verifying data sync. Now it's time for the final and most important part: the **cutover and cleanup**.

This step ensures a **graceful transition** from the old version (PostgreSQL 16) to the new one (PostgreSQL 17), minimizing risks and maintaining high availability.

🚀 What Does "Switchover" Mean?

The **switchover** is the point where your applications stop reading/writing to the old PostgreSQL 16 database and start working

Freedium

🔄 Step-by-Step Cleanup and Switchover Process

✅ 1. Cut Over Application Traffic to PostgreSQL 17

Update your application's **connection configuration** (environment variables, connection strings, service definitions, etc.) so that it now points to the PostgreSQL 17 server.

For example, change:

```
Copy

DB_HOST=old-postgres16-host
```

to:

```
Copy

DB_HOST=new-postgres17-host
```

Things to check before switching:

- Is PostgreSQL 17 using the correct `port` (usually 5432)?
- Are all database users, roles, and permissions configured properly?
- Is the schema identical to PostgreSQL 16?
- Have recent inserts, updates, or deletes replicated correctly?

Only proceed when you're 100% confident the new instance is ready.

Freedium

📊 2. Monitor Replication Using `pg_stat_subscription`

Even after switching traffic, it's wise to monitor your logical replication status. On the **PostgreSQL 17 (subscriber)** server, run:

Copy

```
SELECT * FROM pg_stat_subscription;
repdb=# SELECT * FROM pg_stat_subscription;
 subid |     subname     | worker_type |  pid  | leader_pid | relid | re
-------+-----------------+-------------+-------+------------+-------+---
 16960 | my_subscription | apply       | 11600 |            |       | 0/
(1 row)

repdb=#
repdb=# SELECT * FROM pg_stat_subscription;
 subid |     subname     | worker_type |  pid  | leader_pid | relid | re
-------+-----------------+-------------+-------+------------+-------+---
 16960 | my_subscription | apply       | 11600 |            |       | 0/
(1 row)

repdb=#
```

This will show:

- **`relid`** : Table ID
- **`received_lsn`** : Last received log sequence number
- **`write_lag`** , **`flush_lag`** , **`replay_lag`** : Replication delay in milliseconds
- **`status`** : Should be `streaming`

A `NULL` or `0` value for lag fields means replication is caught up — exactly what we want to see before finalizing the upgrade.

Freedium

🧼 3. Drop Subscription and Publication (Optional but Recommended)

Once you're satisfied that everything is working on PostgreSQL 17 and you no longer need ongoing replication, you can safely remove the replication objects.

On the Subscriber (PostgreSQL 17):

```
Copy

DROP SUBSCRIPTION my_subscription;
```

This stops PostgreSQL 17 from trying to receive further changes from the old server.

On the Publisher (PostgreSQL 16):

```
Copy

DROP PUBLICATION my_publication;
```

This removes the publication created earlier for logical replication.

Why do this? It reduces system overhead and prevents unwanted background processes from running.

🧯 4. Decommission the PostgreSQL 16 Server

If PostgreSQL 17 is now your primary production database, you may want to **decommission the PostgreSQL 16 instance** to reclaim resources and avoid confusion.

Freedium

1. Backup the old server one final time:

```
Copy

pg_dumpall > /tmp/postgres16-final-backup.sql
```

2. Stop the PostgreSQL 16 service:

```
Copy

sudo systemctl stop postgresql-16
```

3. Optionally uninstall it:

```
Copy

sudo dnf remove postgresql16*
```

4. Archive or move old data directory:

```
Copy

sudo mv /var/lib/pgsql/16 /var/lib/pgsql/16_old
```

5. (Optional) Terminate or repurpose the EC2 instance.

> ⚠️ ***Important:*** *Never delete or uninstall until you're fully confident that PostgreSQL 17 is operating smoothly and all data has been verified.*

📝 Conclusion: A Smooth PostgreSQL Upgrade via Logical Replication

Freedium

RHEL 10 EC2 instances. Unlike in-place upgrades or dump/restore strategies, this method offered you:

🎯 Key Benefits

✅ **Cross-version replication:** Works between different major versions without compatibility issues.

✅ **Minimal downtime:** Your applications stayed connected to the old system until the new one was fully ready.

✅ **Controlled rollout:** You decided when and how to switch over, minimizing risks.

✅ **Safe rollback (before cutover):** If something went wrong, you could keep the old server running without disruption.

💡 Final Thoughts

Logical replication isn't just for upgrades — it's also great for:

- High availability and redundancy
- Migrating workloads to new hardware
- Real-time analytics
- Cloud-to-cloud or on-prem-to-cloud transitions

By mastering this approach, you've future-proofed your PostgreSQL upgrade process and embraced a best-practice method used by enterprise teams globally.

📢 Stay Updated with Daily PostgreSQL & Cloud Tips!

Performance Tuning, and DBA Best Practices — I invite you to subscribe to my Medium account.

🔔 Subscribe here 👉 https://medium.com/@jramcloud1/subscribe

Your support means a lot — and you'll never miss a practical guide again!

🔗 Let's Connect!

If you enjoyed this post or would like to connect professionally, feel free to reach out to me on LinkedIn: 👉 Jeyaram Ayyalusamy

I regularly share content on **PostgreSQL, database administration, cloud technologies, and data engineering.** Always happy to connect, collaborate, and discuss ideas!

#postgresql #oracle #mysql #mongodb #aws