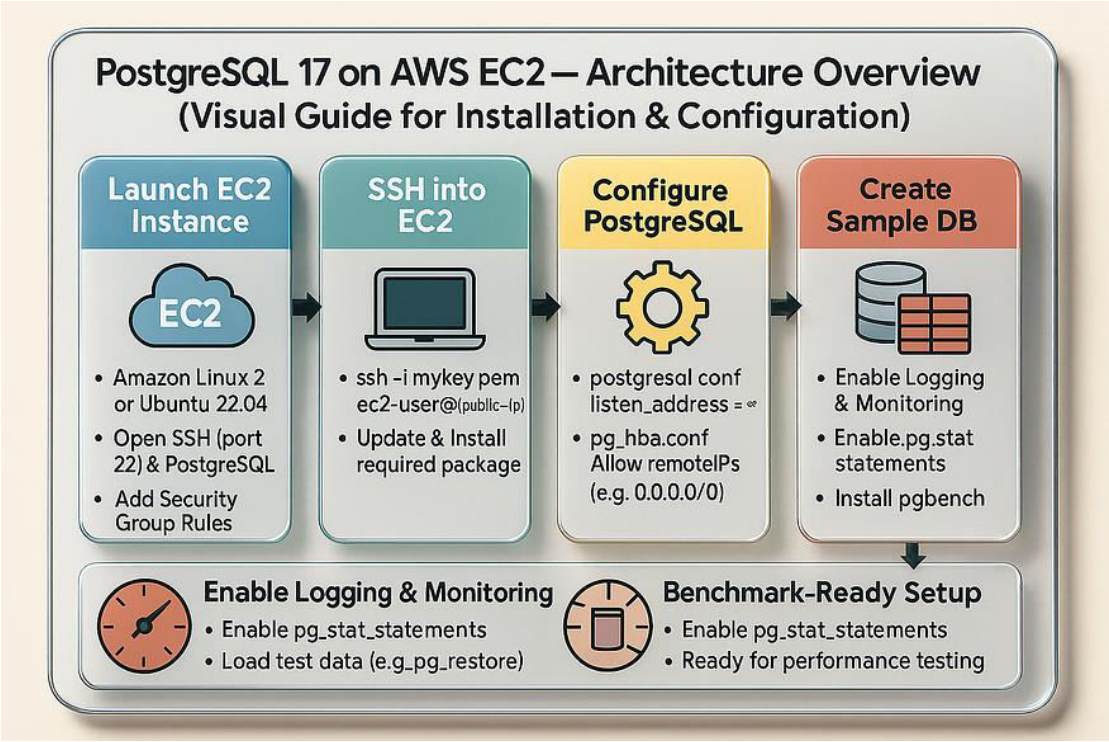
PostgreSQL 17 on AWS EC2 — Full Installation & Configuration Walkthrough



Setting up **PostgreSQL 17 on AWS EC2** might seem complex at first — but once you break down each component, it becomes an efficient and repeatable process.

This guide is designed for **PostgreSQL DBAs** and **Cloud Engineers** who are looking to set up a clean, reliable, and production-grade PostgreSQL environment on AWS. Whether you're building a lab, preparing for performance benchmarking, or supporting a real-world application, this post gives you a clear roadmap.

✅ What You'll Learn — Step by Step

This walkthrough covers the **six core tasks** required to set up PostgreSQL 17 on an EC2 instance in AWS. Each step represents a

foundational block in building a stable, accessible, and observable database system.

1 Connecting to Your EC2 Instance

Your first step is establishing a secure connection to your AWS EC2 virtual machine. This includes launching the instance, configuring key access (SSH key pairs), and ensuring your security groups allow appropriate access — especially for PostgreSQL's default port `5432`. This connection is the gateway to everything else.

2 Installing PostgreSQL 17

Once connected, you'll proceed to install **PostgreSQL 17**. This involves setting up the PostgreSQL Global Development Group (PGDG) repository, installing the server and client packages, and initializing the PostgreSQL cluster. This ensures that your EC2 instance is properly equipped to run the latest PostgreSQL version in a secure and efficient manner.

3 Configuring for Remote Connections

PostgreSQL is conservative out of the box — it only allows local access. To make your database available externally (e.g., for developers, apps, BI tools), you'll modify the configuration files (`postgresql.conf` and `pg_hba.conf`) to allow connections from external hosts. This is critical for team collaboration and production deployments.

4 Enabling Logs and Monitoring

To maintain a healthy PostgreSQL instance, logging is non-negotiable. You'll enable **query logs**, **slow query capture**, and **error reporting**, setting up the groundwork for future analysis and debugging. This section ensures you're not flying blind when issues arise or when performance tuning is needed.

5 Creating Sample Databases with Test Data

No database setup is complete without data to validate functionality. You'll create a test database (like `dvdrental`) and populate it with

representative data. This is useful for practicing query tuning, load testing, or demonstrating features to teams or stakeholders.

6️⃣ Preparing the Instance for Benchmarking

Lastly, you'll ready your PostgreSQL instance for benchmarking. Whether you're evaluating IOPS, testing query latency under load, or comparing schema designs, this section sets the foundation. Tools like `pgbench` and logging extensions help you simulate real-world usage and measure performance.

🔁 Repeatable, Real-World Installation Flow

Whether your goal is **development, benchmarking, or production readiness**, this guide offers a structured and practical path for deploying PostgreSQL 17 on AWS EC2.

🚀 Launching Your First EC2 Instance on AWS

Before diving into PostgreSQL setup on AWS, the very first step is launching a virtual machine — also known as an **EC2 instance**. This section walks you through how to launch your first instance using the **AWS Management Console**. While this is a beginner-friendly guide, the steps also form the foundation for more advanced cloud infrastructure deployments.

✅ Step-by-Step Guide to Launch an EC2 Instance

This tutorial is optimized for simplicity and speed — it avoids advanced networking or provisioning complexities. Perfect for first-time users or for setting up test environments.

1️⃣ Open the EC2 Console

Navigate to the Amazon EC2 Console in your web browser. Once you're logged in:

- Look at the **navigation bar** at the top.

- Choose your preferred **AWS Region** (e.g., Ohio, N. Virginia). It's best to pick a region close to your physical location to reduce latency.

2 Launch the Instance Wizard

From the EC2 **Dashboard**:

- Locate the **Launch instance** pane.
- Click on **Launch instance** to begin configuration.

3 Set a Name

Under the **Name and tags** section:

- Assign a **descriptive name** to your instance (e.g., `pg17-ec2-dev`).
- This helps you identify it later, especially if you manage multiple instances.

4 Choose an OS Image (AMI)

Under **Application and OS Images (Amazon Machine Image)**:

- Select **Quick Start**.
- Choose an operating system. For beginners, **Amazon Linux** is recommended.
- Make sure the AMI you choose is labeled **Free Tier eligible** (ideal for experimentation or small test workloads).

5 Choose an Instance Type

Under **Instance type**:

- Pick a type like `t2.micro` or `t3.micro`, which are often **Free Tier eligible**.

- These types provide sufficient resources for basic PostgreSQL installations.

6 Configure SSH Key Pair

Under **Key pair (login)**:

- Choose an existing **key pair** if you already have one.
- Or click **Create new key pair** to generate a new `.ppk` file.

> ⚠ **Warning**: If you select "Proceed without a key pair," you **won't** be able to securely connect to your instance via SSH. This is **not recommended**.

7 Set Network and Firewall Settings

The console will auto-select your **default VPC** and a default subnet:

- A **security group rule** is created automatically to allow **SSH (port 22)** from anywhere: `0.0.0.0/0`.

> ⚠ **Security Note**: Allowing global access (`0.0.0.0/0`) is acceptable for short-term **testing**, but not safe for production environments. Always restrict access by IP or CIDR range in real deployments.

You can optionally:

- Choose a specific **subnet** (Edit → Subnet).
- Use a different **VPC** (Edit → VPC).
- Restrict access to your **organization's IP range** instead of "Anywhere".
- Select an **existing security group** with custom rules.

8️⃣ Configure Storage

Under **Configure storage**:

- A **root volume** is automatically provisioned.
- No additional data volumes are needed for basic PostgreSQL setup.

For test environments, this default setup is sufficient.

9️⃣ Review and Launch

Before launching:

- Review all configurations in the **Summary panel**.
- When ready, click **Launch instance**.

You will see a **Success** message and a link to your new instance's ID.

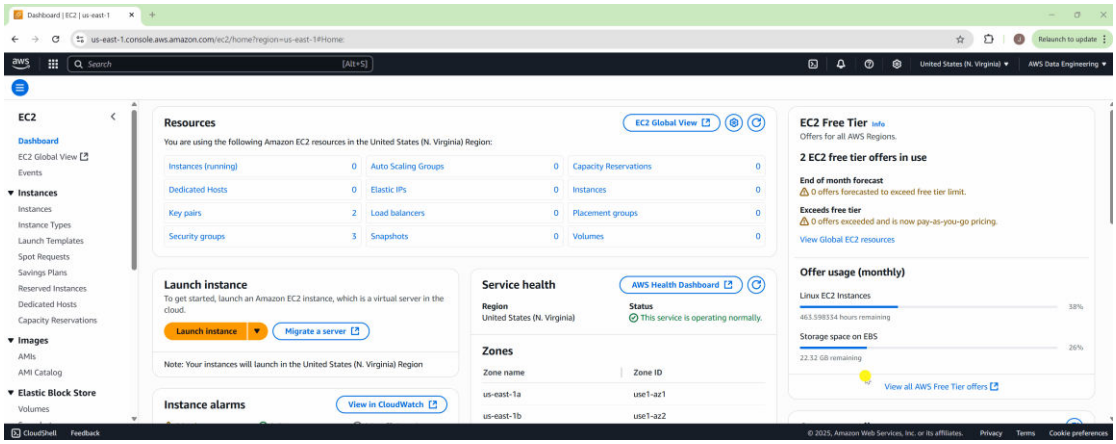
🔍 Monitor Instance Status

On the **Instances page**:

- Find your newly created instance and **select its checkbox**.
- Its status will show **Pending**.
- Once fully initialized, it will switch to **Running**.
- Under the **Status and alarms** tab, wait for all **status checks** to pass.

Your EC2 instance is now fully operational and ready for SSH access or application deployment.

Press enter or click to view image in full size



🛠️ Step 2: Get Your EC2 Public IP Address

1. Go to the **EC2 Dashboard → Instances**.
2. Copy the **Public IPv4 address** or **Public DNS** of your EC2 instance.

💻 Step 3: Connect via PuTTY

1. Open **PuTTY**.
2. In the **Host Name (or IP address)** field, type:

```
ec2-user@<your-ec2-public-ip>
```

1. Example (for Amazon Linux):

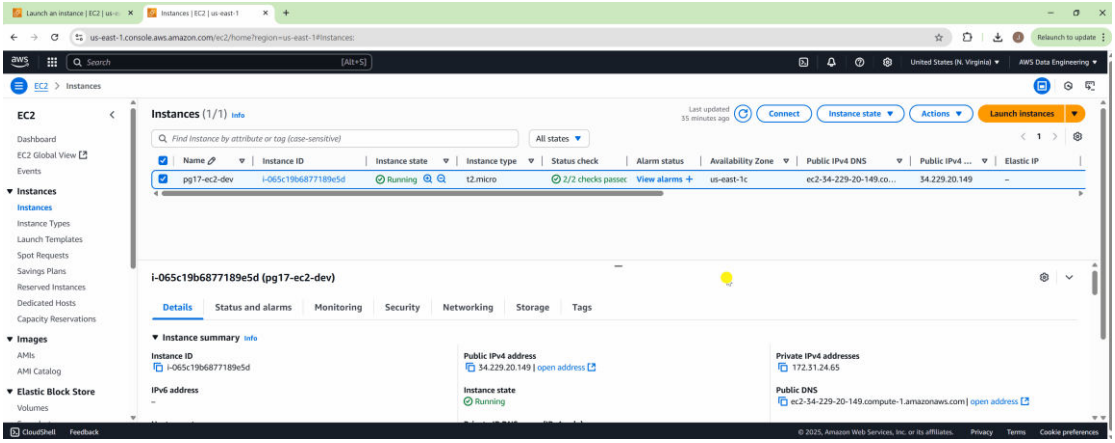
```
ec2-user@18.219.101.54
```

1. For Ubuntu, use:

```
ubuntu@<your-ec2-public-ip>
```

1. In the **Category** panel on the left, in the **Conection** and expand **SSH** → select **Auth**.
2. Click **Browse** and select your `.ppk` file (the private key you saved earlier).
3. Go back to **Session** (top of left panel).
4. (Optional) Save your session settings with a name in the **Saved Sessions** box and click **Save**.
5. Click **Open** to initiate the connection.

Press enter or click to view image in full size



🚀 Installing PostgreSQL 17 on AWS EC2: A Complete Step-by-Step Guide

PostgreSQL 17 brings performance enhancements, modern features, and improved compatibility. If you're running workloads in AWS, spinning up a PostgreSQL 17 instance on **Amazon EC2** gives you full control over configuration, optimization, and monitoring.

This section covers **everything you need to install PostgreSQL 17** on a Red Hat-based EC2 instance (e.g., Amazon Linux 2023 or RHEL 9/10), including initialization and starting the database service.

🔗 Prerequisites

Ensure you:

- Have an EC2 instance up and running.
- Logged in via SSH (e.g., using PuTTY or terminal).
- Have `sudo` privileges on the instance.
- Are connected to the internet (for downloading packages).

🛠️ Step 1: Add the Official PostgreSQL Repository

PostgreSQL packages are not included in Amazon Linux by default or may be outdated. So, we begin by adding the **official PostgreSQL YUM repository** for PostgreSQL 17:

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-10-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

🔷 **Why this matters:** This gives you access to the latest version (v17) and official packages maintained by the PostgreSQL Global Development Group.

🚫 Step 2: Disable the Default PostgreSQL Module

Amazon Linux and RHEL systems come with older PostgreSQL modules pre-enabled. To prevent conflicts, we disable them:

```
sudo dnf -qy module disable postgresql
```

🔷 **Why this matters:** If you don't do this, your system might install an older version (e.g., PostgreSQL 13 or 14) instead of 17.

📦 Step 3: Install PostgreSQL 17 Server

Install the actual PostgreSQL server packages:

```
sudo dnf install -y postgresql17-server
```

◈ **What's included?** This command installs:

- PostgreSQL binaries
- Server daemon (`postgres`)
- Client tools (`psql`, `createdb`, etc.)

⚙ Step 4: Initialize the PostgreSQL Data Directory

Before PostgreSQL can start, it needs to set up its data directory and system tables:

```
sudo /usr/pgsql-17/bin/postgresql-17-setup initdb
```

◈ **This does the following:**

- Creates the `postgres` data directory under `/var/lib/pgsql/17/data/`
- Initializes configuration files like `postgresql.conf` and `pg_hba.conf`
- Prepares system catalogs and metadata

🔄 Step 5: Enable PostgreSQL to Start on Boot

You'll want PostgreSQL to restart automatically after system reboots:

```
sudo systemctl enable postgresql-17
```

🔷 **Why this matters:** This adds PostgreSQL to the systemd startup list so it runs on every boot.

▶️ Step 6: Start the PostgreSQL Service

Now start the service:

```
sudo systemctl start postgresql-17
```

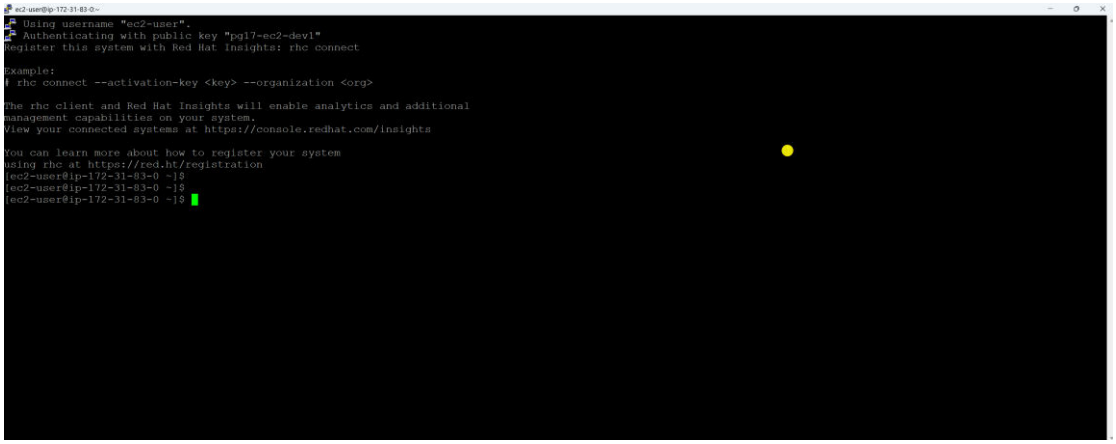
✅ PostgreSQL is now **running and ready** to accept local connections. To check the service status:

```
sudo systemctl status postgresql-17
```

You should see something like:

```
Active: active (running) since ...
Main PID: xxxx (postgres)
```

Press enter or click to view image in full size



🛡️ PostgreSQL 17 on AWS EC2 — Remote Access, Security, Roles, and Performance

Now that PostgreSQL 17 is successfully installed on your AWS EC2 instance, it's time to **prepare it for real-world use**. This section covers five key steps:

- Allowing remote access
- Creating roles and databases
- Enabling detailed logging
- Securing the instance via firewall and security groups
- Benchmarking performance

Let's walk through each in detail.

🌐 1. Allow Remote Access

By default, PostgreSQL only listens to `localhost` for security. To allow access from your local machine or an app server, you need to configure:

🔧 **`postgresql.conf`**

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

Find this line:

```
#listen_addresses = 'localhost'
#port = 5432
```

Change it to:

```
listen_addresses = '*'
port = 5432
```

💡 This tells PostgreSQL to listen on all available network interfaces. You can also specify IPs like `'192.168.1.100'` for tighter control.

🔐 **`pg_hba.conf`**

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

Add a line at the bottom (adjust IP range as needed):

```
host    all             all             0.0.0.0/0               md5
```

This allows **password-based connections** from any IP address. In production, replace `0.0.0.0/0` with your office or app server IP range for better security.

🔄 Restart PostgreSQL after changes:

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

🧑‍🚀 2. Create Roles and Databases

Once PostgreSQL is running and accessible, switch to the `postgres` user:

```
sudo su - postgres
psql
```

◆ Create a new user (role):

```
CREATE ROLE myuser WITH LOGIN PASSWORD 'mypassword';
ALTER ROLE myuser CREATEDB;
```

- `LOGIN` allows the user to connect.
- `CREATEDB` gives them permission to create databases.

◆ Create a new database:

```
CREATE DATABASE myappdb OWNER myuser;
```

Exit `psql` by typing `\q`.

📋 3. Enable Detailed Logging

Logging is critical for monitoring query performance, identifying slow queries, and auditing.

Edit the config:

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

Enable these parameters:

```
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%a.log'
log_statement = 'all'
log_min_duration_statement = 500    # log queries longer than 500 ms
```

🔄 Restart PostgreSQL:

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

🗂 Logs will be stored under `/var/lib/pgsql/17/data/log/`

Connect to the database

```
[postgres@ip-172-31-83-0 ~]$ psql -U myuser -d myappdb -h localhost -p 5432
Password for user myuser:
psql (17.5)
Type "help" for help.

myappdb=>
myappdb=>
```

Press enter or click to view image in full size

🔒 4. Set Up Firewall and Security Groups

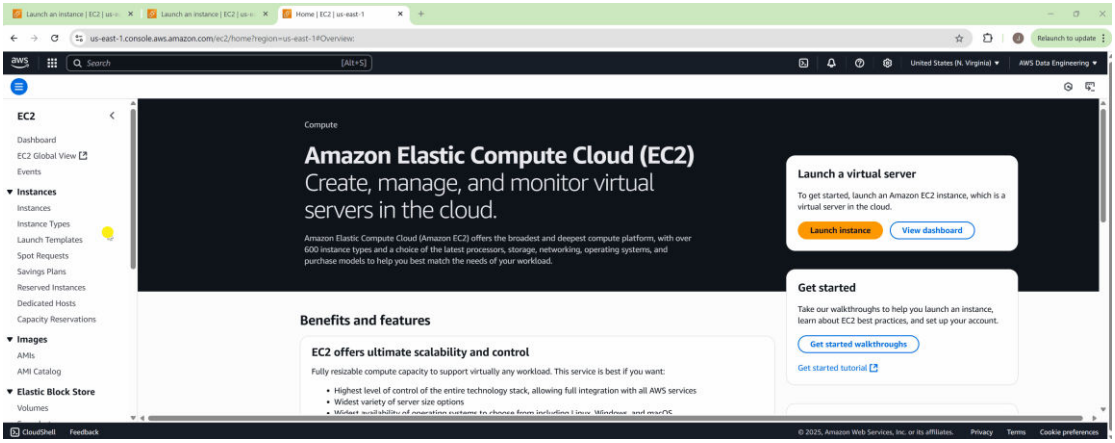
PostgreSQL uses **port 5432**. You must allow inbound traffic through this port:

AWS Security Group:

1. Go to **EC2 Dashboard > Instances > choose Instance > Security**
2. Click on the security group attached to your instance.
3. Under **Inbound rules**, click **Edit inbound rules**.
4. Add:
5. Type Protocol Port Source PostgreSQL TCP 5432 Your IP (e.g., `203.0.113.0/32`)

✅ Never use `0.0.0.0/0` unless it's a test instance.

Press enter or click to view image in full size



🚀 5. Benchmark Performance (Optional)

To test the performance of your PostgreSQL setup, you can use the built-in `pgbench` tool:

Install `postgresql-contrib`:

```
sudo dnf install -y postgresql17-contrib
```

Initialize benchmark:

```
pgbench -i -s 10 myappdb
```

This populates the database with sample data.

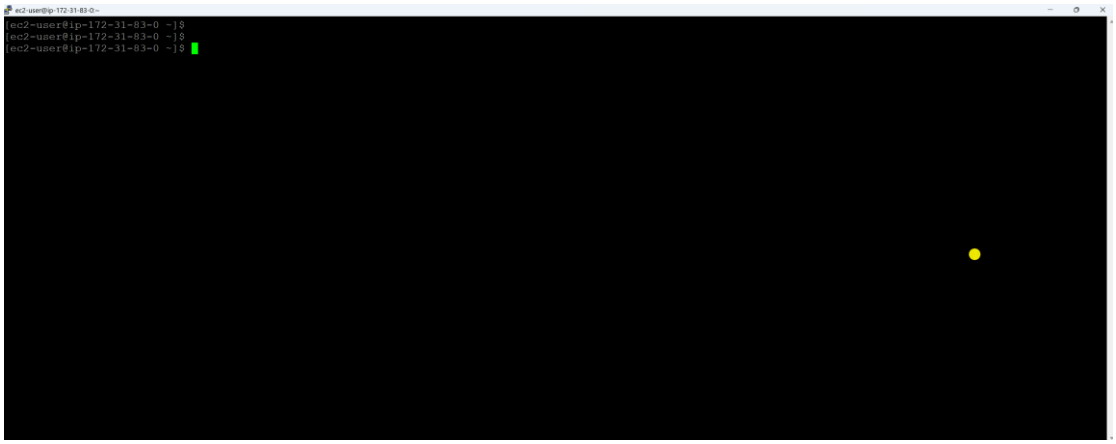
Run test:

```
pgbench -c 10 -j 2 -T 60 myappdb
```

- `-c`: number of clients
- `-j`: number of threads
- `-T`: duration (in seconds)

📊 You'll get a performance summary including TPS (transactions per second).

Press enter or click to view image in full size



✅ Recap

By completing these steps, you've:

- Enabled **secure remote access** to PostgreSQL
- Created **database roles and schemas**
- Turned on **detailed logging**
- Hardened your server using **security groups**
- Validated performance using **pgbench**

This setup is now ready for development or production use 🚀.

Congratulations — you now have a fully operational **PostgreSQL 16 environment** on AWS EC2:

- ✅ Remote-access ready
- ✅ Logging and monitoring enabled
- ✅ Benchmark data loaded
- ✅ WAL files configured for safety

🚀 Why This Setup Works in Real World:

- ✅ **Cloud friendly**: Optimized for EC2
- ✅ **DBA friendly**: Logging + monitoring enabled by default
- ✅ **Performance ready**: Immediate benchmarking support
- ✅ **Safe for production**: Minimal downtime config changes