

# PostgreSQL

- [Overview](#)
- [How to Connect](#)
  - [Connecting with Node.js](#)
- [How-To Guides](#)
  - [Creating a Database](#)
- [Database Migration](#)
  - [Database Migration Service for PostgreSQL](#)
- [Cluster Management](#)
  - [Overview](#)

# Overview

PostgreSQL is an open-source relational database management system. It supports SQL language and offers features like transactions, referential integrity, and user-defined types and functions. PostgreSQL can handle complex queries, supports various data types, and is extensible, allowing users to add custom functions and data types. It runs on multiple operating systems, including Windows, Linux, and macOS.

## Key Features of PostgreSQL:

- **Extensibility:** Users can define custom data types, operators, and index methods, tailoring the database to specific application needs.
- **Standards Compliance:** PostgreSQL conforms to at least 170 of the 177 mandatory features for SQL:2023 Core conformance, ensuring compatibility with SQL standards.
- **Advanced Data Types:** Supports a wide array of data types, including JSON for unstructured data, arrays, hstore (key-value pairs), and geometric types, enhancing its versatility.
- **Concurrency and Performance:** Utilizes Multi-Version Concurrency Control (MVCC) to handle multiple transactions simultaneously without conflicts, ensuring data integrity and performance.
- **Replication and High Availability:** Offers asynchronous replication, allowing data to be copied to standby servers for load balancing and failover support, enhancing reliability.
- **ACID Compliance:** Ensures transactions are processed reliably through Atomicity, Consistency, Isolation, and Durability properties, which are crucial for applications requiring data integrity.
- **Security Features:** Provides robust security mechanisms, including role-based access control, data encryption, and connection security, safeguarding data against unauthorized access.
- **Cross-Platform Support:** Runs on all major operating systems, including Windows, Linux, macOS, FreeBSD, and OpenBSD, offering flexibility in deployment environments.

These features make PostgreSQL a preferred choice for developers and enterprises seeking a reliable, feature-rich, and scalable database solution.

# How to Connect

# Connecting with Node.js

This guide explains how to establish a connection between a Node.js application and a PostgreSQL database using the `pg` package. It walks through the necessary setup, configuration, and execution of a simple SQL query.

## Variables

Certain parameters must be provided to establish a successful connection to a PostgreSQL database. Below is a breakdown of each required variable, its purpose, and where to find it. Here’s what each variable represents:

Variable	Description	Purpose
<code>USER</code>	PostgreSQL username, from the Elestio service overview page	Identifies the database user who has permission to access the PostgreSQL database.
<code>PASSWORD</code>	PostgreSQL password, from the Elestio service overview page	The authentication key required for the specified <code>USER</code> to access the database
<code>HOST</code>	Hostname for PostgreSQL connection, from the Elestio service overview page	The address of the server hosting the PostgreSQL database.
<code>PORT</code>	Port for PostgreSQL connection, from the Elestio service overview page	The network port is used to connect to PostgreSQL. The default port is <code>5432</code> .
<code>DATABASE</code>	Database Name for PostgreSQL connection, from the Elestio service overview page	The name of the database being accessed. A PostgreSQL instance can contain multiple databases.

These values can usually be found in the Elestio service overview details as shown in the image below, make sure to take a copy of these details and add it to the code moving ahead.



postgresql-2p7j1

PostgreSQL

Running

Open terminal

Delete service

Clone this service

Overview

Tools

Backups

Metrics

Monitoring

Logs

Audit

Security

Alerts

Termination protection

Disabled. VM can be powered off and terminated.

Protection deactivated



Database Admin

Display your database credentials

Hide DB Credentials

Host postgresql-2p7j1-u7774.vm.elestio.app



Port 25432



User postgres



Password \*\*\*\*\*

Show password



CLI PGPASSWORD=\*\*\*\*\* psql --host=postgresql-2p7j1-u7774.vm.elestio.app --port=25432 --username=postgres

Show password



# Prerequisites

## • Install Node.js and NPM

- Check if Node.js is installed by running:

```
node -v
```

- If not installed, download it from [nodejs.org](https://nodejs.org) and install.
- Verify npm installation:

```
npm -v
```

## • Install the `pg` Package

The `pg` package enables Node.js applications to interact with PostgreSQL. Install it using:

```
npm install pg --save
```

# Code

Once all prerequisites are set up, create a new file named `pg.js` and add the following code:

```
const pg = require("pg");
```

```
// Database connection configuration
const config = {
  user: "USER",
  password: "PASSWORD",
  host: "HOST",
  port: "PORT",
  database: "DATABASE",
};

// Create a new PostgreSQL client
const client = new pg.Client(config);

// Connect to the database
client.connect((err) => {
  if (err) {
    console.error("Connection failed:", err);
    return;
  }
  console.log("Connected to PostgreSQL");

  // Run a test query to check the PostgreSQL version
  client.query("SELECT VERSION()", [], (err, result) => {
    if (err) {
      console.error("Query execution failed:", err);
      client.end();
      return;
    }

    console.log("PostgreSQL Version:", result.rows[0]);

    // Close the database connection
    client.end((err) => {
      if (err) console.error("Error closing connection:", err);
    });
  });
});
```

To execute the script, open the terminal or command prompt and navigate to the directory where `pg.js`. Once in the correct directory, run the script with the command

```
node pg.js
```

If the connection is successful, the terminal will display output similar to:

```
Connected to PostgreSQL
```

```
PostgreSQL Version: {
```

```
  version: 'PostgreSQL 16.8 (Debian 16.8-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian  
12.2.0-14) 12.2.0, 64-bit'
```

```
}
```

# How-To Guides



# Creating a Database

PostgreSQL allows you to create databases using different methods, including the PostgreSQL interactive shell (`psql`), Docker (assuming PostgreSQL is running inside a container), and the command-line interface (`createdb`). This guide explains each method step-by-step, covering required permissions, best practices, and troubleshooting common issues.

## Creating Using psql CLI

PostgreSQL is a database system that stores and manages structured data efficiently. The `psql` tool is an interactive command-line interface (CLI) that allows users to execute SQL commands directly on a PostgreSQL database. Follow these steps to create a database:

### Connect to PostgreSQL

Open terminal on your local system, and if PostgreSQL is installed locally, connect using the following command. If not installed, install from [official website](#):

```
psql -U postgres
```

For a remote database, use:

```
psql -h HOST -U USER -d DATABASE
```

Replace `HOST` with the database server address, `USER` with the PostgreSQL username, and `DATABASE` with an existing database name.

### Create a New Database

Inside the `psql` shell, run:

```
CREATE DATABASE mydatabase;
```

The default settings will apply unless specified otherwise. To customize the encoding and collation, use:

```
CREATE DATABASE mydatabase ENCODING 'UTF8' LC_COLLATE 'en_US.UTF-8' LC_CTYPE 'en_US.UTF-8'  
TEMPLATE template0;
```

# Creating Database in Docker

Docker is a tool that helps run applications in isolated environments called containers. A PostgreSQL container provides a self-contained database instance that can be quickly deployed and managed. If you are running PostgreSQL inside a Docker container, follow these steps:

## Pull the PostgreSQL Docker Image

As you start, make sure your docker is running correctly. Next, if you haven't already downloaded the PostgreSQL image, run the following command and it will take a while to pull the image successfully:

```
docker pull postgres
```

## Start a PostgreSQL Container

Run a new PostgreSQL instance using Docker. This will run the docker image as a container using the given credentials, ports etc:

```
docker run --name my-postgres -e POSTGRES_USER=admin -e POSTGRES_PASSWORD=secret -p 5432:5432 -d  
postgres
```

## Access the PostgreSQL Shell Inside the Container

Connect to the running container, this will execute the given command inside the docker container:

```
docker exec -it my-postgres psql -U admin
```

## Create a Database

Inside the PostgreSQL shell, run following command to create the specific database:

```
CREATE DATABASE mydatabase;
```

# Creating Using createdb CLI

The `createdb` command simplifies database creation from the terminal without using `psql`.

## Ensure PostgreSQL is Running

Check the PostgreSQL service status, this ensure that the PostgreSQL instance is running on your local instance:

```
sudo systemctl status postgresql
```

If not running, start it:

```
sudo systemctl start postgresql
```

## Create a Database

Now, you can create a simple database using the following command:

```
createdb -U postgres mydatabase
```

To specify encoding and collation:

```
createdb -U postgres --encoding=UTF8 --lc-collate=en_US.UTF-8 --lc-ctype=en_US.UTF-8 mydatabase
```

## Verify Database Creation

List all databases using following commands as it will list all the databases available under your postgres:

```
psql -U postgres -l
```

## Connect to the New Database

Next you can easily connect with the database using `psql` command and starting working on it.

```
psql -U postgres -d mydatabase
```

# Required Permissions for Database Creation

Creating a database requires the `CREATEDB` privilege. By default, the `postgres` user has this privilege. To grant it to another user:

```
ALTER USER username CREATEDB;
```

For restricted access, assign specific permissions:

```
CREATE ROLE newuser WITH LOGIN PASSWORD 'securepassword';  
GRANT CONNECT ON DATABASE mydatabase TO newuser;  
GRANT USAGE ON SCHEMA public TO newuser;  
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO newuser;
```

# Best Practices for Creating Databases

- **Use Meaningful Names:** Choosing clear and descriptive names for databases helps in organization and maintenance. Avoid generic names like `testdb` or `database1`, as they do not indicate the database's purpose. Instead, use names that reflect the type of data stored, such as `customer_data` or `sales_records`. Meaningful names make it easier for developers and administrators to understand the database's function without extra documentation.
- **Follow Naming Conventions:** A standardized naming convention ensures consistency across projects and simplifies database management. PostgreSQL is case-sensitive, so using lowercase letters and underscores (e.g., `order_details`) is recommended to avoid unnecessary complexities. Avoid spaces and special characters in names, as they require additional quoting in SQL queries.
- **Restrict User Permissions:** Granting only the necessary permissions improves database security and reduces risks. By default, users should have the least privilege required for their tasks, such as read-only access for reporting tools. Superuser or administrative privileges should be limited to trusted users to prevent accidental or malicious changes. Using roles and groups simplifies permission management and ensures consistent access control.
- **Enable Backups:** Regular backups ensure data recovery in case of accidental deletions, hardware failures, or security breaches. PostgreSQL provides built-in tools like `pg_dump` for single-database backups and `pg_basebackup` for full-instance backups. Automating backups using cron jobs or scheduling them through a database management tool reduces the risk of data loss.
- **Monitor Performance:** Monitoring database performance helps identify bottlenecks, optimize queries, and ensure efficient resource utilization. PostgreSQL provides system views like `pg_stat_activity` and `pg_stat_database` to track query execution and database

usage. Analyzing slow queries using `EXPLAIN ANALYZE` helps in indexing and optimization.

```
SELECT datname, numbackends, xact_commit, blks_read FROM pg_stat_database;
```

# Common Issues and Troubleshooting

Issue	Possible Cause	Solution
<code>ERROR: permission denied to create database</code>	User lacks <code>CREATEDB</code> privileges	Grant permission using <code>ALTER USER username CREATEDB;</code>
<code>ERROR: database "mydatabase" already exists</code>	Database name already taken	Use a different name or drop the existing one with <code>DROP DATABASE mydatabase;</code>
<code>FATAL: database "mydatabase" does not exist</code>	Attempting to connect to a non-existent database	Verify creation using <code>\l</code>
<code>psql: could not connect to server</code>	PostgreSQL is not running	Start PostgreSQL with <code>sudo systemctl start postgresql</code>
<code>ERROR: role "username" does not exist</code>	The specified user does not exist	Create the user with <code>CREATE ROLE username WITH LOGIN PASSWORD 'password';</code>

# Database Migration

# Database Migration Service for PostgreSQL

Elestio provides a structured approach for migrating PostgreSQL databases from various environments, such as on-premises systems or other cloud platforms, to its managed services. This process ensures data integrity and minimizes downtime, facilitating a smooth transition to a managed environment.

## Key Steps in Migrating to Elestio

### Pre-Migration Preparation

Before initiating the migration process, it's essential to undertake thorough preparation to ensure a smooth transition:

- **Create an Elestio Account:** Register on the Elestio platform to access their suite of managed services. This account will serve as the central hub for managing your PostgreSQL instance and related resources.
- **Deploy the Target PostgreSQL Service:** Set up a new PostgreSQL instance on Elestio to serve as the destination for your data. It's crucial to match the software version of your current PostgreSQL database to avoid compatibility issues during data transfer. Detailed prerequisites and guidance can be found in Elestio's [migration documentation](#).

### Initiating the Migration Process

With the preparatory steps completed, you can proceed to migrate your PostgreSQL database to Elestio:

1. **Access the Migration Tool:** Navigate to the overview of your PostgreSQL service on the Elestio dashboard. Click on the "Migrate Database" button to initiate the migration process. This tool is designed to facilitate a smooth transition by guiding you through each step.

2. **Configure Migration Settings:** A modal window will open, prompting you to ensure that your target service has sufficient disk space to accommodate your database. Adequate storage is vital to prevent interruptions during data transfer. Once confirmed, click on the "Get started" button to proceed.
3. **Validate Source Database Connection:** Provide the connection details for your existing PostgreSQL database, including:
  - **Hostname:** The address of your current database server.
  - **Port:** The port number on which your PostgreSQL service is running (default is 5432).
  - **Database Name:** The name of the database you intend to migrate.
  - **Username:** The username with access privileges to the database.
  - **Password:** The corresponding password for the user.

After entering these details, click on "Run Check" to validate the connection. This step ensures that Elestio can securely and accurately access your existing data. You can find these details under Database admin section under your deployed PostgreSQL service.

Database Admin		Display your database credentials	Hide DB Credentials
Host	postgresql-3c5xl-u7774.vm.elestio.app		
Port	25432		
User	postgres		
Password	*****		Show password
CLI	PGPASSWORD=***** psql --host=postgresql-3c5xl-u7774.vm.elestio.app --port=25432 --username=postgres		Show password

4. **Execute the Migration:** If all checks pass without errors, initiate the migration by selecting "Start migration." Monitor the progress through the real-time migration logs displayed on the dashboard. This transparency allows for immediate detection and resolution of any issues, ensuring data integrity throughout the process.

## Post-Migration Validation and Optimization

After completing the migration, it's crucial to perform validation and optimization tasks to ensure the integrity and performance of your database in the new environment:

- **Verify Data Integrity:** Conduct thorough checks to ensure all data has been accurately transferred. This includes comparing row counts, checksums, and sample data between the source and target databases. Such verification maintains the reliability of your database and ensures that no data was lost or altered during migration.
- **Test Application Functionality:** Ensure that applications interacting with the database function correctly in the new environment. Update connection strings and configurations



as necessary to reflect the new database location. This step prevents potential disruptions and ensures seamless operation of dependent systems.

- **Optimize Performance:** Utilize Elestio's managed service features to fine-tune database performance. Set up automated backups to safeguard your data, monitor resource utilization to identify and address bottlenecks, and configure scaling options to accommodate future growth. These actions contribute to improved application responsiveness and overall system efficiency.
- **Implement Security Measures:** Review and configure security settings to protect your data within the Elestio environment. Set up firewalls to control access, manage user access controls to ensure only authorized personnel can interact with the database, and enable encryption where applicable to protect data at rest and in transit. Implementing these security measures safeguards your data against unauthorized access and potential threats.

# Benefits of Using Elestio for PostgreSQL

Migrating your PostgreSQL database to Elestio offers several advantages:

- **Simplified Management:** Elestio automates database maintenance tasks, including software updates, backups, and system monitoring, reducing manual work. The platform provides a dashboard with real-time insights into database performance and resource usage. It allows for adjusting service plans, scaling CPU and RAM as needed. Users can modify environment variables and access software information to manage configurations.
- **Security:** Elestio keeps PostgreSQL instances updated with security patches to protect against vulnerabilities. The platform automates backups to ensure data integrity and availability. It provides secure access mechanisms, including randomly generated passwords for database instances, which can be managed through the dashboard.
- **Performance:** Elestio configures PostgreSQL instances for performance based on workload requirements. The platform supports the latest PostgreSQL versions, incorporating updates that improve database operations. Its infrastructure handles different workloads and maintains performance during high usage periods.
- **Scalability:** Elestio's PostgreSQL service allows for scaling database resources to handle growth and changing workloads without major downtime. Users can upgrade or downgrade service plans, adjusting CPU and RAM as needed. The platform supports adding network volumes to increase storage capacity.

# Cluster Management

# Overview

Elestio provides a complete solution for setting up and managing software clusters. This helps users deploy, scale, and maintain applications more reliably. Clustering improves performance and ensures that services remain available, even if one part of the system fails. Elestio supports different cluster setups to handle various technical needs like load balancing, failover, and data replication.

## Supported Software for Clustering:

Elestio supports clustering for a wide range of open-source software. Each is designed to support different use cases like databases, caching, and analytics:

- **MySQL:**

Supports Single Node, Primary/Replica, and Multi-Master cluster types. These allow users to create simple setups or more advanced ones where reads and writes are distributed across nodes. In a Primary/Replica setup, replicas are updated continuously through replication. These configurations are useful for high-traffic applications that need fast and reliable access to data.

- **PostgreSQL:**

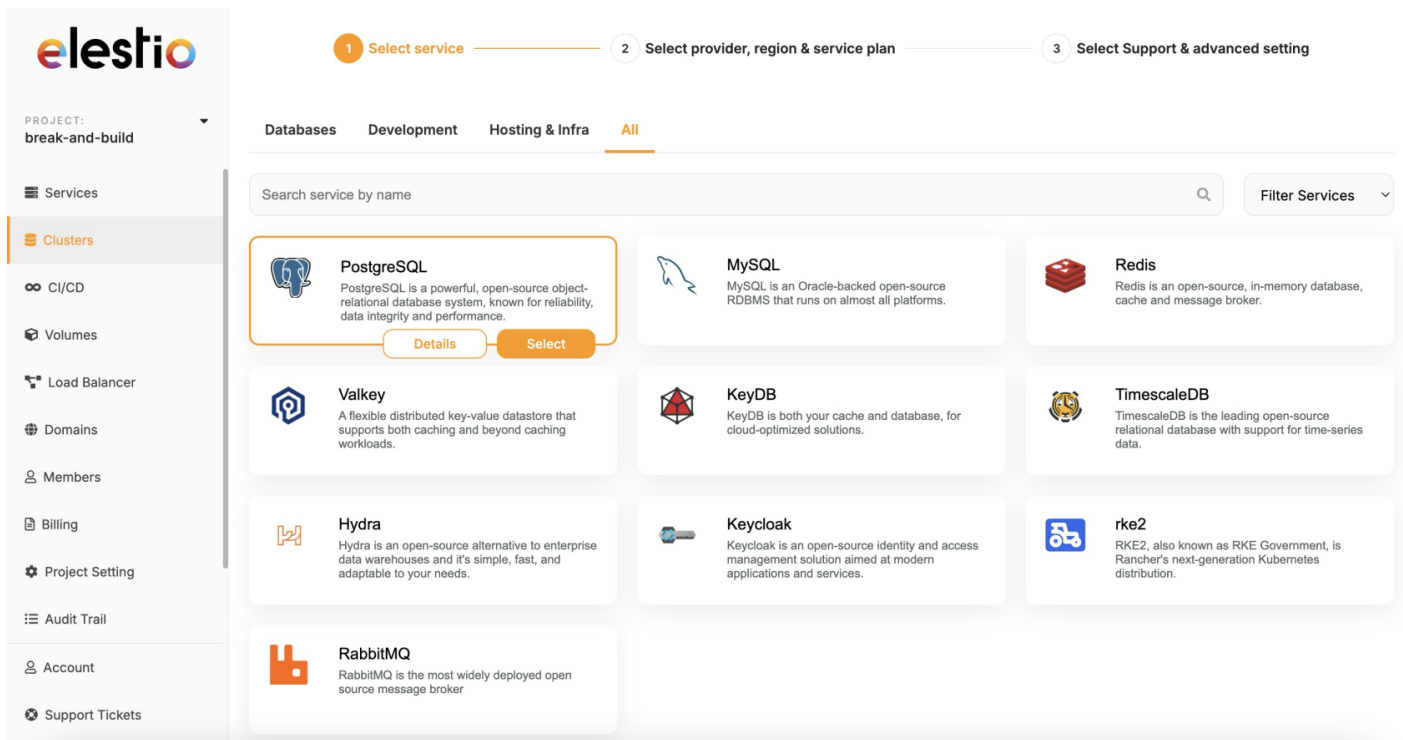
PostgreSQL clusters can be configured for read scalability and failover protection. Replication ensures that data written to the primary node is copied to replicas. Clustering PostgreSQL also improves query throughput by offloading read queries to replicas. Elestio handles replication setup and node failover automatically.

- **Redis/KeyDB/Valkey:**

These in-memory data stores support clustering to improve speed and fault tolerance. Clustering divides data across multiple nodes (sharding), allowing horizontal scaling. These tools are commonly used for caching and real-time applications, so fast failover and data availability are critical.

- **Hydra and TimescaleDB:**

These support distributed and time-series workloads, respectively. Clustering helps manage large datasets spread across many nodes. TimescaleDB, built on PostgreSQL, benefits from clustering by distributing time-based data for fast querying. Hydra uses clustering to process identity and access management workloads more efficiently in high-load environments.



**Note:** Elestio is frequently adding support for more clustered software like OpenSearch, Kafka, and ClickHouse. Always check the Elestio catalogue for the latest supported services.

## Cluster Configurations:

Elestio offers several clustering modes, each designed for a different balance between simplicity, speed, and reliability:

- **Single Node:**

This setup has only one node and is easy to manage. It acts as a standalone Primary node. It's good for testing, development, or low-traffic applications. Later, you can scale to more nodes without rebuilding the entire setup. Elestio lets you expand this node into a full cluster with just a few clicks.

- **Primary/Replica:**

One node (Primary) handles all write operations, and one or more Replicas handle read queries. Replication is usually asynchronous and ensures data is copied to all replicas. This improves read performance and provides redundancy if the primary node fails. Elestio manages automatic data syncing and failover setup.

## Cluster Management Features:

Elestio's cluster dashboard includes tools for managing, monitoring, and securing your clusters. These help ensure stability and ease of use:

- **Node Management:**

You can scale your cluster by adding or removing nodes as your app grows. Adding a node increases capacity; removing one helps reduce costs. Elestio handles provisioning and configuring nodes automatically, including replication setup. This makes it easier to scale

horizontally without downtime.

- **Backups and Restores:**

Elestio provides scheduled and on-demand backups for all nodes. Backups are stored securely and can be restored if something goes wrong. You can also create a snapshot before major changes to your system. This helps protect against data loss due to failures, bugs, or human error.

- **Access Control:**

You can limit access to your cluster using IP allowlists, ensuring only trusted sources can connect. Role-based access control (RBAC) can be applied for managing different user permissions. SSH and database passwords are generated securely and can be rotated easily from the dashboard. These access tools help reduce the risk of unauthorized access.

- **Monitoring and Alerts:**

Real-time metrics like CPU, memory, disk usage, and network traffic are available through the dashboard. You can also check logs for troubleshooting and set alerts for high resource usage or failure events. Elestio uses built-in observability tools to monitor the health of your cluster and notify you if something needs attention. This allows you to catch problems early and take action.