MySQL

Overview

MySQL is an open-source relational database management system. It supports SQL language and offers features like transactions, indexing, and replication. MySQL is widely used for web applications and enterprise solutions due to its performance, reliability, and ease of use. It runs on multiple operating systems, including Windows, Linux, and macOS.

Key Features of MySQL:

- **Performance and Scalability:** Designed to handle high-volume environments with fast read and write operations. It supports partitioning, indexing, and query optimization for better performance.
- **Replication and High Availability:** Offers master-slave and group replication setups, enabling load balancing, redundancy, and failover support to maintain uptime and data consistency.
- **Storage Engines:** Provides support for multiple storage engines, including InnoDB for ACID-compliant transactions and MyISAM for faster read-heavy workloads, giving flexibility in storage design.
- **Security Features:** Includes features like SSL support, role-based access control, user privilege management, and data encryption to secure database access and data integrity.
- **ACID Compliance:** With the InnoDB storage engine, MySQL ensures Atomicity, Consistency, Isolation, and Durability in transactions, which is essential for reliable data management.
- **Cross-Platform Support:** Compatible with all major operating systems, such as Windows, Linux, and macOS, allowing flexible deployment options in different environments.
- **JSON Support:** Provides native JSON data type and functions, enabling semi-structured data handling within relational structures.
- **Ease of Use and Tooling:** Offers tools like MySQL Workbench and integration with phpMyAdmin, making it accessible for both developers and administrators to manage schemas, run queries, and monitor performance.

These features make MySQL a preferred choice for developers and organizations seeking a stable, efficient, and well-supported database system.

How to Connect

Connecting with Node.js

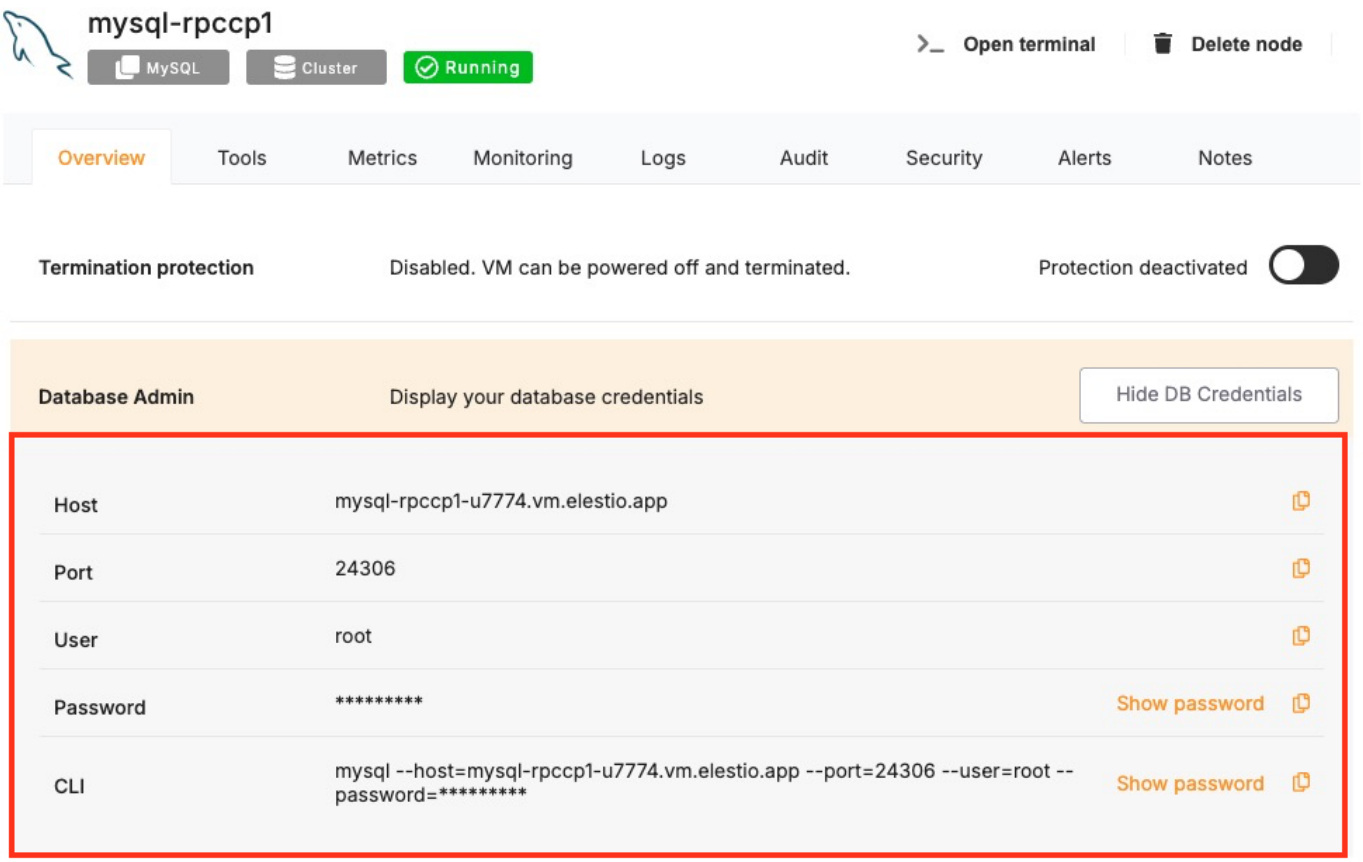
This guide explains how to establish a connection between a Node.js application and a MySQL database using the `mysql2` package. It walks through the necessary setup, configuration, and execution of a simple SQL query.

Variables

Certain parameters must be provided to establish a successful connection to a MySQL 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
`USER`	MySQL username, from the Elestio service overview page	Identifies the database user who has permission to access the MySQL database.
`PASSWORD`	MySQL password, from the Elestio service overview page	The authentication key is required for the specified USER to access the database.
`HOST`	Hostname for MySQL connection, from the Elestio service overview page	The address of the server hosting the MySQL database.
`PORT`	Port for MySQL connection, from the Elestio service overview page	The network port used to connect to MySQL. The default port is 3306.
`DATABASE`	Database Name for MySQL connection, from the Elestio service overview page	The name of the database being accessed. A MySQL 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.



Prerequisites

- **Install Node.js and NPM**
  - Check if Node.js is installed by running: `node -v`
  - If not installed, download it from nodejs.org and install. Additionally, verify npm installation: `npm -v`
- **Install the mysql2 Package**
  - The mysql2 package enables Node.js applications to interact with MySQL. Install it using: `npm install mysql2 --save`

Code

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

```
const mysql = require("mysql2");


// Database connection configuration
```

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

// Create a MySQL connection
const connection = mysql.createConnection(config);

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

  // Run a test query to check the MySQL version
  connection.query("SELECT VERSION() AS version", (err, results) => {
    if (err) {
      console.error("Query execution failed:", err);
      connection.end();
      return;
    }

    console.log("MySQL Version:", results[0]);

    // Close the database connection
    connection.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 mysql.js is located. Once in the correct directory, run the script with the command:

```
node mysql.js
```

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

```
Connected to MySQL
MySQL Version: { version: '8.0.41' }
```

Connecting with Python

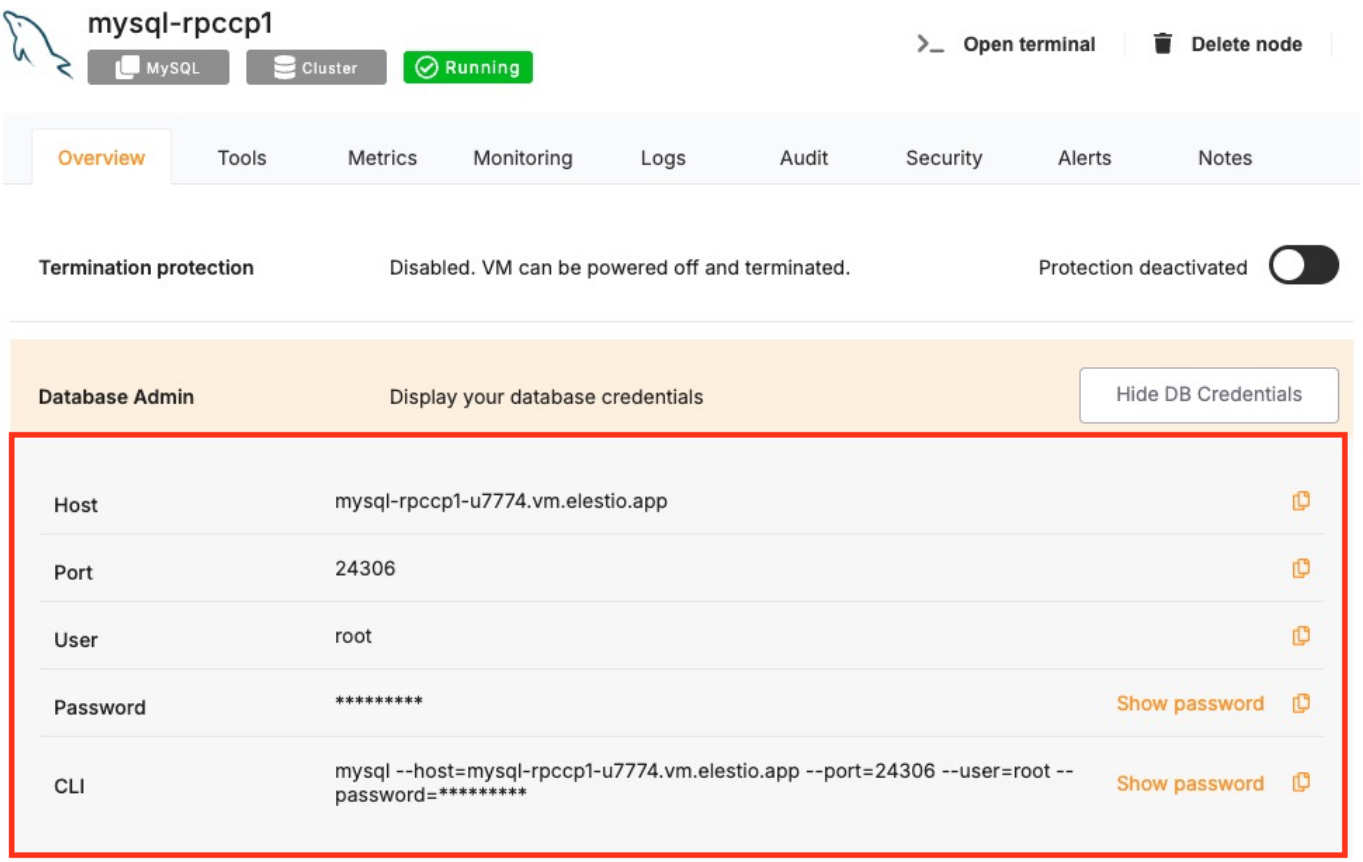
This guide explains how to establish a connection between a Python application and a MySQL database using the `mysql-connector-python` package. It walks through the necessary setup, configuration, and execution of a simple SQL query.

Variables

Certain parameters must be provided to establish a successful connection to a MySQL 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
`USER`	MySQL username, from the Elestio service overview page	Identifies the database user who has permission to access the MySQL database.
`PASSWORD`	MySQL password, from the Elestio service overview page	The authentication key is required for the specified USER to access the database.
`HOST`	Hostname for MySQL connection, from the Elestio service overview page	The address of the server hosting the MySQL database.
`PORT`	Port for MySQL connection, from the Elestio service overview page	The network port used to connect to MySQL. The default port is 3306.
`DATABASE`	Database Name for MySQL connection, from the Elestio service overview page	The name of the database being accessed. A MySQL 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.



Prerequisites

- **Install Python**
  - Check if Python is installed by running: `python --version`
  - If not installed, download it from python.org and install it.

- **Install the `mysql-connector-python` Package**
  - The mysql-connector-python package enables Python applications to interact with MySQL. Install it using: `pip install mysql-connector-python`

Code

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

```
import mysql.connector


# Database connection configuration
config = {
```

```
    "host": "HOST",
    "user": "USER",
    "password": "PASSWORD",
    "database": "DATABASE",
    "port": PORT
}

try:
    # Establish the connection
    connection = mysql.connector.connect(**config)
    print("Connected to MySQL")

    # Create a cursor and execute a test query
    cursor = connection.cursor()
    cursor.execute("SELECT VERSION()")

    # Fetch and print the result
    version = cursor.fetchone()
    print("MySQL Version:", version[0])

except mysql.connector.Error as err:
    print("Connection failed:", err)

finally:
    if 'cursor' in locals():
        cursor.close()
    if 'connection' in locals() and connection.is_connected():
        connection.close()
        print("Connection closed")
```

To execute the script, open the terminal or command prompt and navigate to the directory where `mysql_connect.py` is located. Once in the correct directory, run the script with the command:

```
python mysql_connect.py
```

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

```
Connected to MySQL
MySQL Version: 8.0.41
Connection closed
```

Connecting with PHP

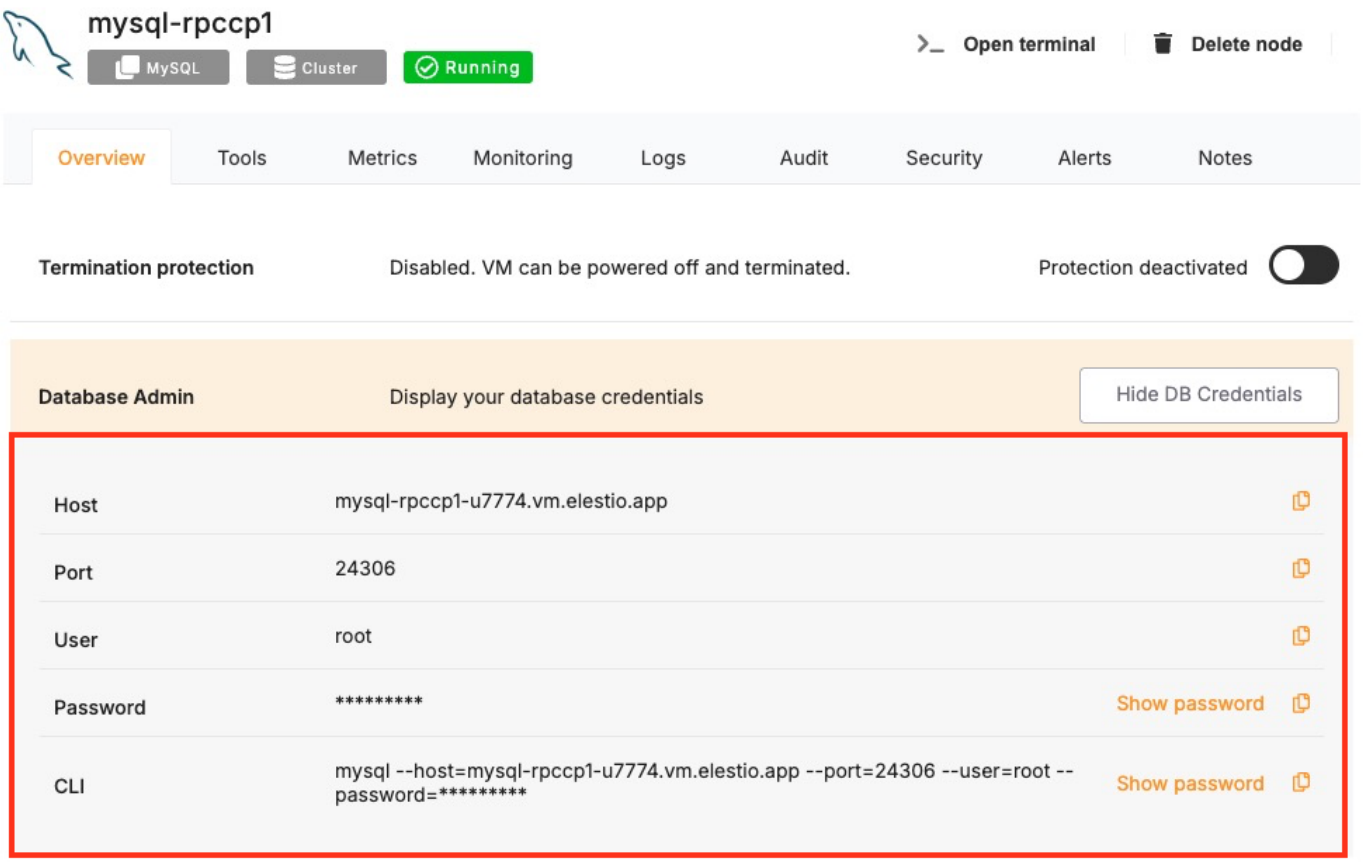
This guide explains how to establish a connection between a PHP application and a MySQL database using the mysqli extension. It walks through the necessary setup, configuration, and execution of a simple SQL query.

Variables

Certain parameters must be provided to establish a successful connection to a MySQL 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
USER	MySQL username, from the Elestio service overview page	Identifies the database user who has permission to access the MySQL database.
PASSWORD	MySQL password, from the Elestio service overview page	The authentication key is required for the specified USER to access the database.
HOST	Hostname for MySQL connection, from the Elestio service overview page	The address of the server hosting the MySQL database.
PORT	Port for MySQL connection, from the Elestio service overview page	The network port used to connect to MySQL. The default port is 3306.
DATABASE	Database Name for MySQL connection, from the Elestio service overview page	The name of the database being accessed. A MySQL 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.



Prerequisites

- **Install PHP**
  - Check if PHP is installed by running: `php -v`
  - If not installed, download it from php.net and install.
  - Make sure the mysqli extension is enabled in your php.ini configuration.

Code

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

```
<?php
$host = "HOST";
$user = "USER";
$password = "PASSWORD";
$database = "DATABASE";
$port = PORT;
```

```
// Create connection
$conn = new mysqli($host, $user, $password, $database, $port);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected to MySQL<br>";

// Run a test query to check the MySQL version
$result = $conn->query("SELECT VERSION()");

if ($result) {
    $row = $result->fetch_assoc();
    echo "MySQL Version: " . $row["VERSION()"];
    $result->free();
} else {
    echo "Query execution failed: " . $conn->error;
}

// Close connection
$conn->close();
?>
```

To execute the script, run the PHP server in the directory where `mysql_connect.php` is located using:

```
php -S localhost:8000
```

Then, open a browser and go to:

```
http://localhost:8000/mysql_connect.php
```

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

```
Connected to MySQL
MySQL Version: 8.0.36
```

Connecting with Go

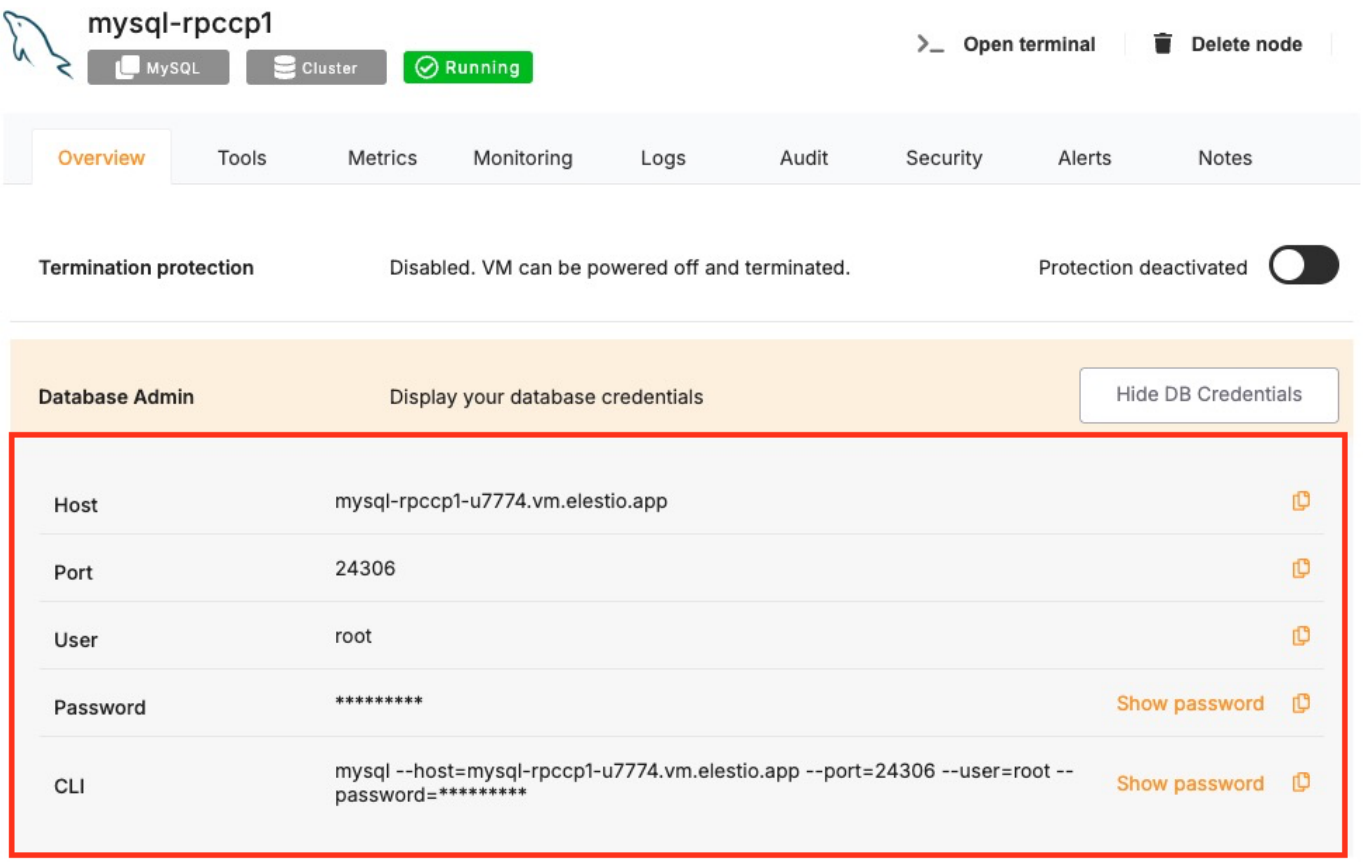
This guide explains how to establish a connection between a Go application and a MySQL database using the `go-sql-driver/mysql` package. It walks through the necessary setup, configuration, and execution of a simple SQL query.

Variables

Certain parameters must be provided to establish a successful connection to a MySQL 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
`USER`	MySQL username, from the Elestio service overview page	Identifies the database user who has permission to access the MySQL database.
`PASSWORD`	MySQL password, from the Elestio service overview page	The authentication key is required for the specified USER to access the database.
`HOST`	Hostname for MySQL connection, from the Elestio service overview page	The address of the server hosting the MySQL database.
`PORT`	Port for MySQL connection, from the Elestio service overview page	The network port used to connect to MySQL. The default port is 3306.
`DATABASE`	Database Name for MySQL connection, from the Elestio service overview page	The name of the database being accessed. A MySQL 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 them to the code moving ahead.



Prerequisites

- **Install Go**
  - Check if Go is installed by running: `go version`
  - If not installed, download it from golang.org and install.

- **Install the MySQL Driver**
  - Use the following command to install the go-sql-driver/mysql driver: `go get -u github.com/go-sql-driver/mysql`

Code

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

```
package main

import (
"database/sql"
```

```
	"fmt"
	"log"

	_ "github.com/go-sql-driver/mysql"
)

func main() {
	user := "USER"
	password := "PASSWORD"
	host := "HOST"
	port := "PORT"
	database := "DATABASE"

	// Construct DSN (Data Source Name)
	dsn := fmt.Sprintf("%s:%s@tcp(%s:%s)/%s", user, password, host, port, database)

	// Open a connection
	db, err := sql.Open("mysql", dsn)
	if err != nil {
		log.Fatalf("Connection failed: %v", err)
	}
	defer db.Close()

	// Ping to verify connection
	if err := db.Ping(); err != nil {
		log.Fatalf("Ping failed: %v", err)
	}
	fmt.Println("Connected to MySQL")

	// Run a test query to check the MySQL version
	var version string
	err = db.QueryRow("SELECT VERSION()").Scan(&version)
	if err != nil {
		log.Fatalf("Query execution failed: %v", err)
	}
	fmt.Printf("MySQL Version: %s\n", version)
}
```

To execute the script, open the terminal and navigate to the directory where `mysql_connect.go` is located. Once in the correct directory, run the script with the commands:

```
go mod init example.com/mysqlconnect
go run mysql_connect.go
```

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

```
Connected to MySQL
MySQL Version: 8.0.36
```

Connecting with Java

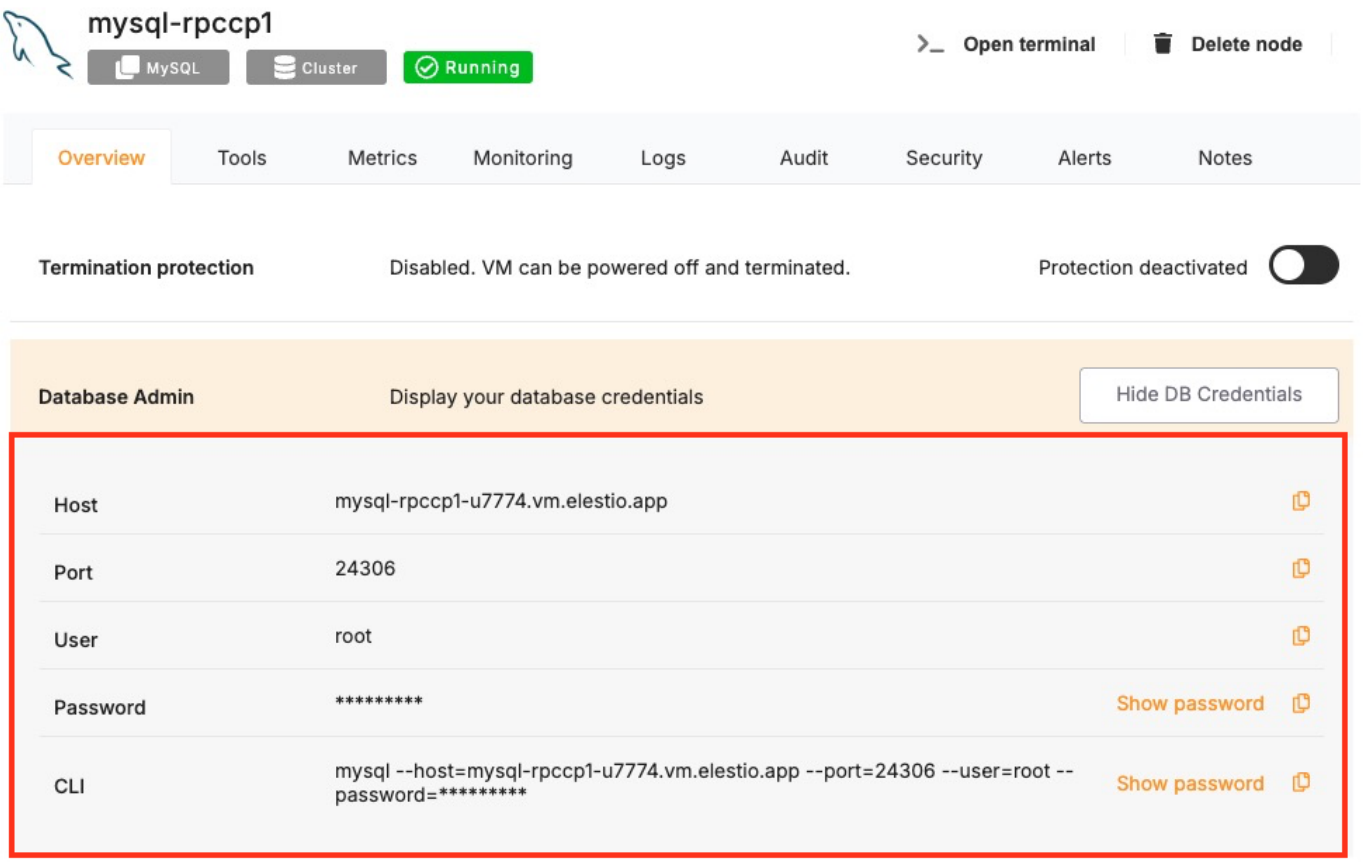
This guide explains how to establish a connection between a Java application and a MySQL database using the `mysql-connector-j` JDBC driver. It walks through the necessary setup, configuration, and execution of a simple SQL query.

Variables

Certain parameters must be provided to establish a successful connection to a MySQL 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
`USER`	MySQL username, from the Elestio service overview page	Identifies the database user who has permission to access the MySQL database.
`PASSWORD`	MySQL password, from the Elestio service overview page	The authentication key is required for the specified USER to access the database.
`HOST`	Hostname for MySQL connection, from the Elestio service overview page	The address of the server hosting the MySQL database.
`PORT`	Port for MySQL connection, from the Elestio service overview page	The network port used to connect to MySQL. The default port is 3306.
`DATABASE`	Database Name for MySQL connection, from the Elestio service overview page	The name of the database being accessed. A MySQL 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.



Prerequisites

- **Install Java**
  - Check if Java is installed by running: `java -version`.
  - If not installed, download it from oracle.com or install OpenJDK.
- **Install MySQL Connector/J**
  - Download the latest version `mysql-connector-j` from the official MySQL site.

Code

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

```
import java.sql.*;
import java.util.*;


public class MySQLConnect {
```

```
    public static void main(String[] args) {
        Map<String, String> config = new HashMap<>();
        for (int i = 0; i < args.length - 1; i += 2)
            config.put(args[i], args[i + 1]);

        String url = String.format("jdbc:mysql://%s:%s/%s?useSSL=true",
                config.get("-host"), config.get("-port"), config.get("-database"));

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            try (Connection conn = DriverManager.getConnection(url, config.get("-username"), config.get("-
password"));
                 Statement stmt = conn.createStatement();
                 ResultSet rs = stmt.executeQuery("SELECT VERSION()")) {
                System.out.println("Connected to MySQL");
                if (rs.next()) System.out.println("MySQL Version: " + rs.getString(1));
            }
        } catch (Exception e) {
            System.err.println("Connection error: " + e.getMessage());
        }
    }
}
```

To compile and run the Java program, use the following commands in your terminal:

```
javac MySQLConnect.java && java -cp mysql-connector-j-9.3.0.jar:. MySQLConnect -host HOST -port PORT -
database DATABASE -username avnadmin -password PASSWORD
```

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

```
Connected to MySQL
MySQL Version: 8.0.41
```

Connecting with phpMyAdmin

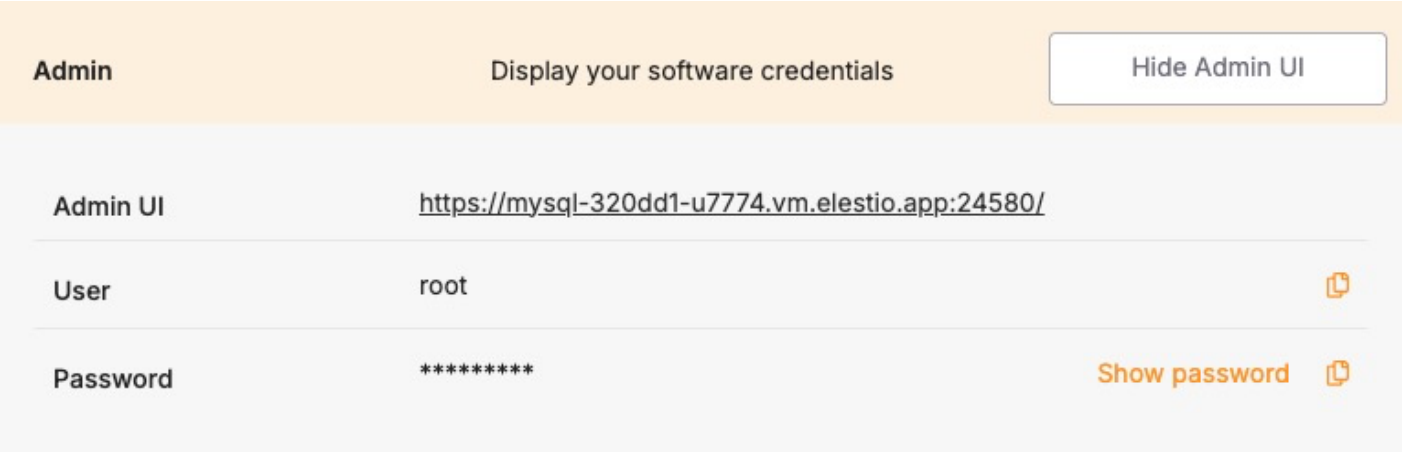
phpMyAdmin is a widely used web-based interface for MySQL that allows you to manage databases, run SQL queries, and administer users through a graphical interface.

Variables

To connect using phpMyAdmin, you'll need the following connection parameters. When you deploy a MySQL service on Elestio, you also get a phpMyAdmin dashboard configured for you to use with these variables. These details are available in the Elestio service overview page:

Variable	Description	Purpose
USER	phpMyAdmin username	Identifies the database user.
PASSWORD	phpMyAdmin password	Authentication key for the `USER`.

You can find these values in your Elestio project dashboard under the Admin section.



Prerequisites

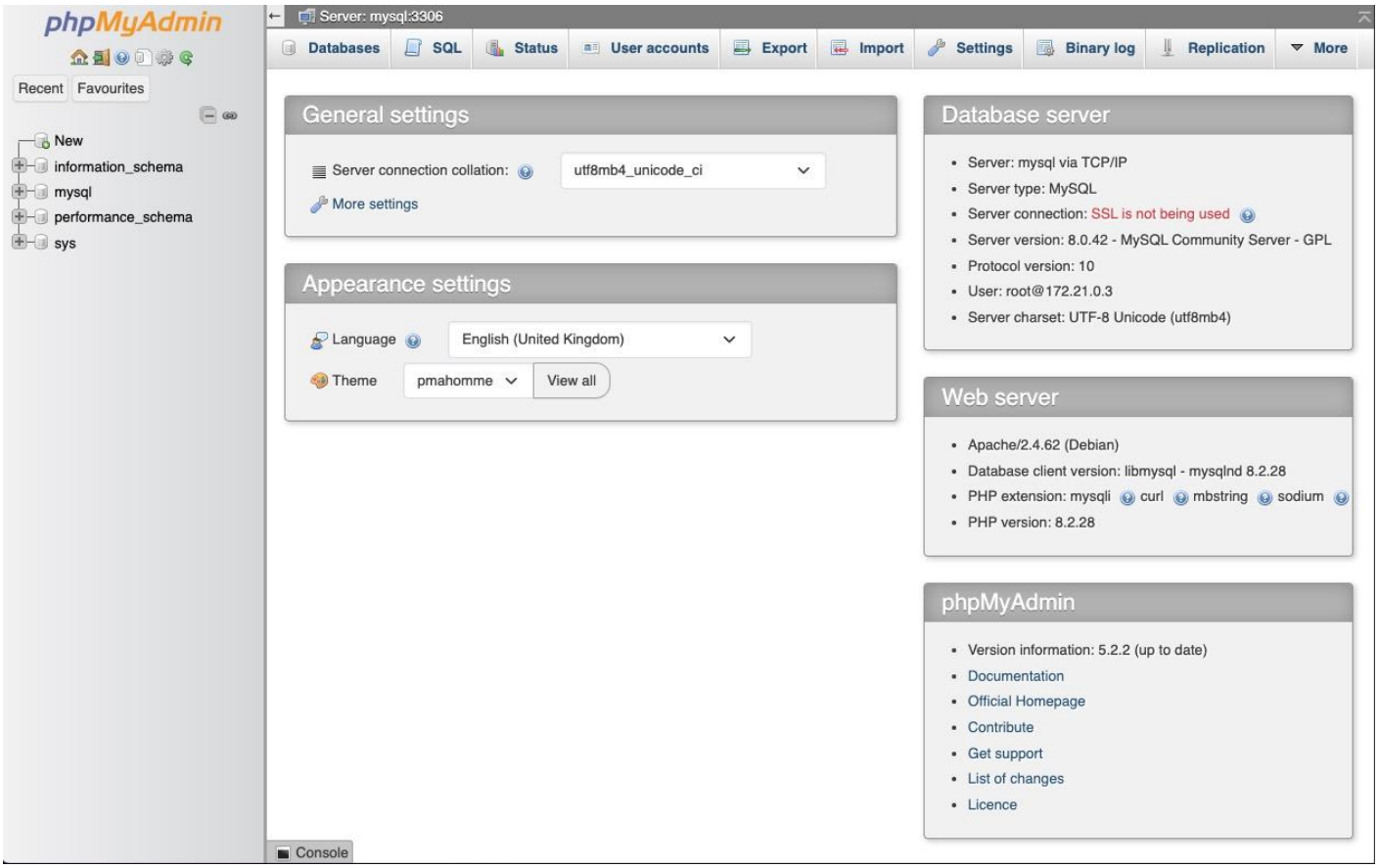
Make sure the MySQL service is correctly deployed on Elestio and you are able to access the Admin section where phpMyAdmin is listed, similar to the example shown in the image above.

Setting Up the Connection

Launch phpMyAdmin using the Admin UI URL and log in with the credentials acquired from the Elestio service dashboard. Once the login screen is loaded, enter the following:

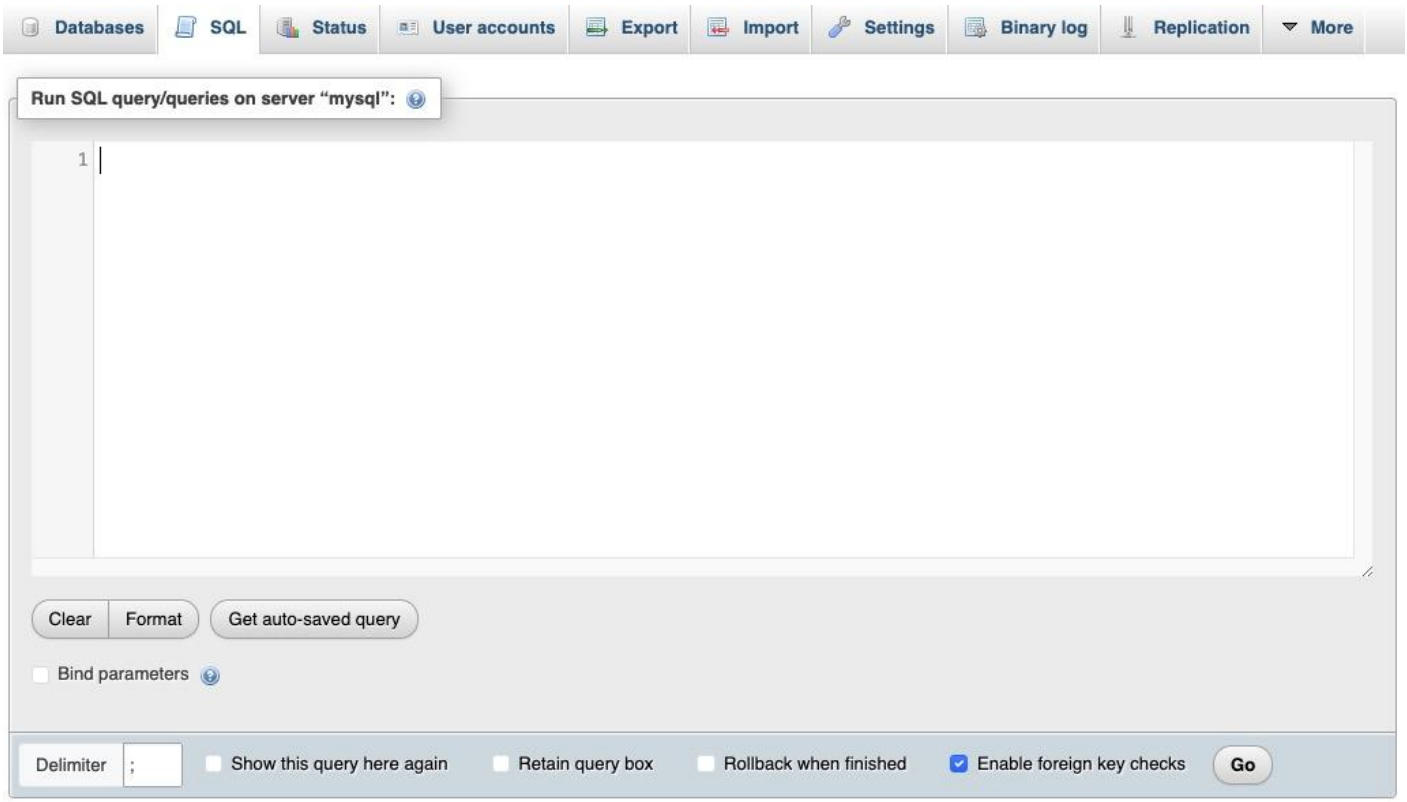
- **Username**: USER
- **Password**: PASSWORD

Click on **Go** to access the phpMyAdmin interface.

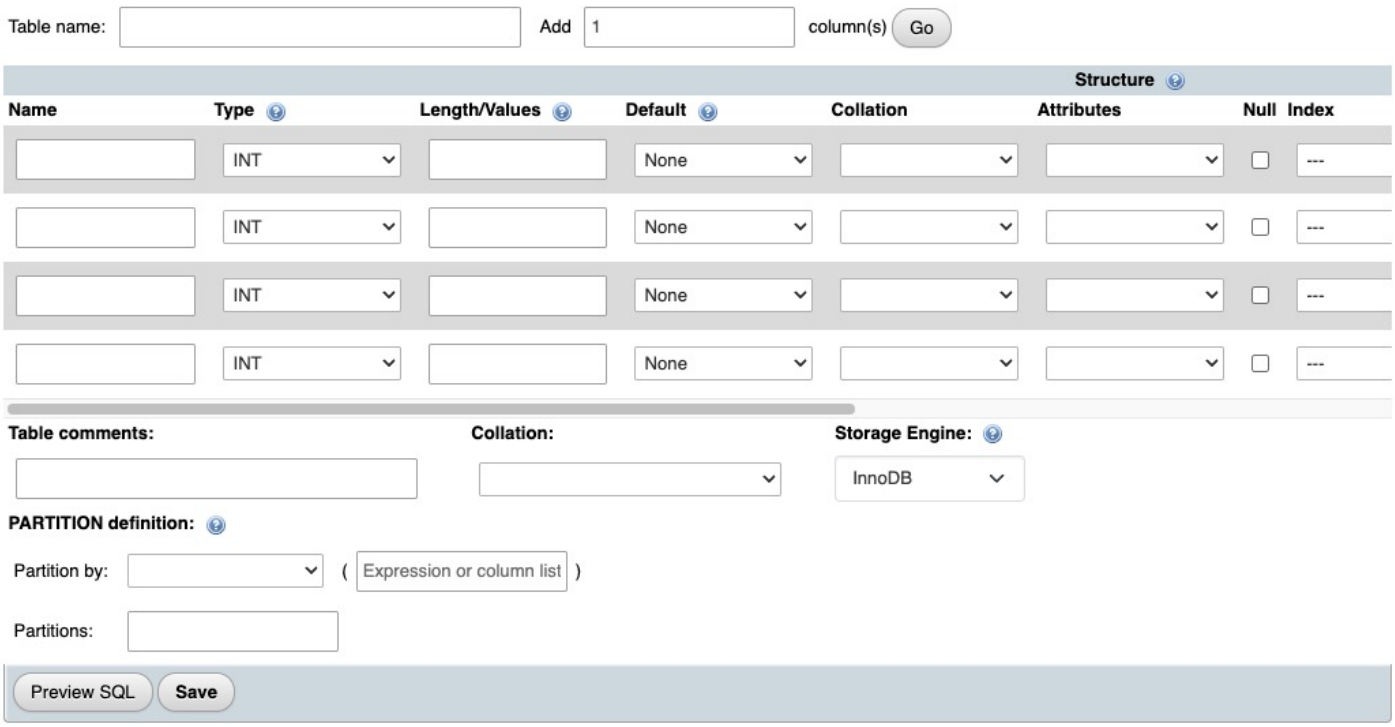


Once logged in, you can see your available databases listed in the left panel. From here, you can:

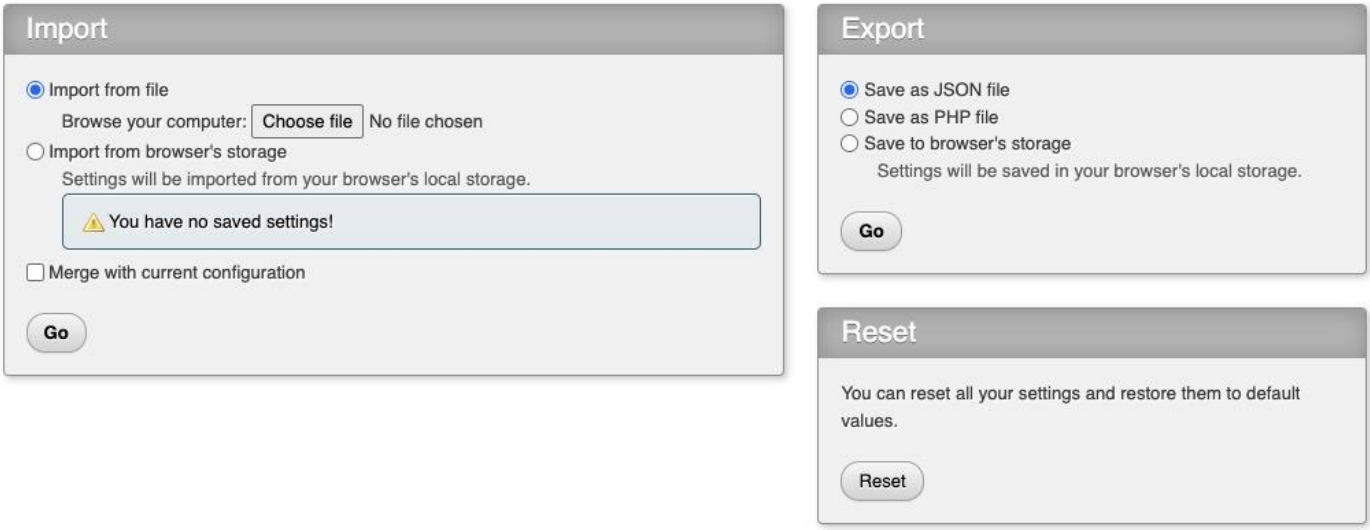
- Run SQL queries through the **SQL** tab



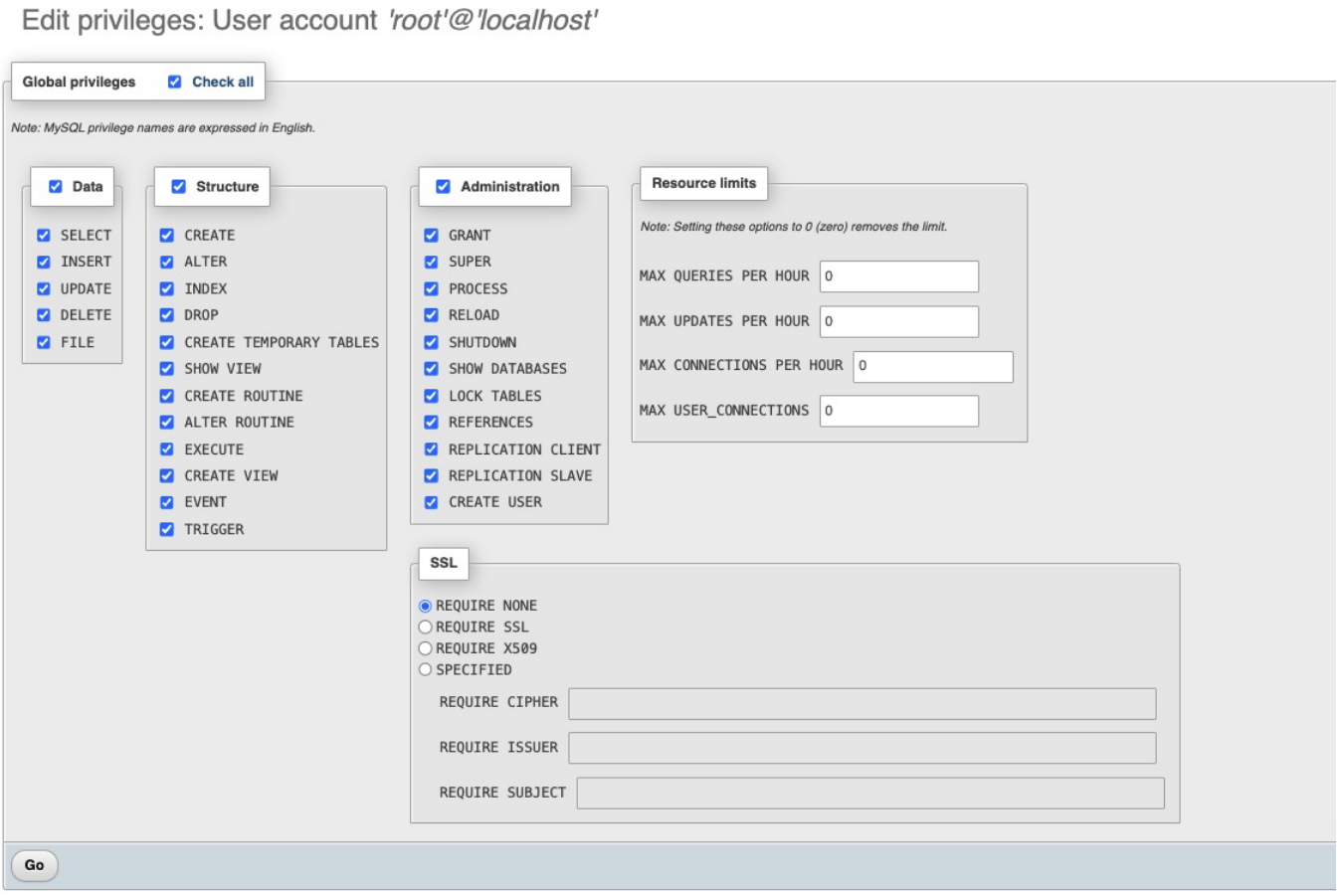
- View or modify table structures



- Export or import database backups



- Manage users and privileges if applicable



Connecting with mysql

This guide explains how to connect to a MySQL database using the `mysql` command-line tool. It walks through the necessary setup, connection process, and execution of a simple SQL query.

Variables

To connect to a MySQL database, you will need the following individual connection parameters. These are available on the Elestio service overview page:

Variable	Description	Purpose
`USER`	MySQL username	Identifies the database user.
`PASSWORD`	MySQL password	Authenticates the user.
`HOST`	MySQL host address	Endpoint to connect to the database service.
`PORT`	MySQL port number	Default is usually 3306, unless otherwise configured.
`DATABASE`	Database name	The specific database you want to connect to.

You can find all of these values in your Elestio project dashboard under the **Admin** or **Database Info** section.

Prerequisites

Make sure the MySQL client is installed on your local system. If not, download and install it from:

https://dev.mysql.com/downloads/

Connecting to MySQL

Open your terminal and run the following command to connect to the MySQL database using the values you copied from your Elestio service:

```
mysql -h HOST -P PORT -u USER -p DATABASE
```

- Replace `HOST`, `PORT`, `USER`, and `DATABASE` with the actual values.
- After running the command, you will be prompted to enter the `PASSWORD`.

If the connection is successful, you will see output similar to this:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 8.0.34 MySQL Community Server - GPL

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Verifying the Connection

To ensure you’re connected correctly, run the following command in the MySQL prompt:

```
SELECT VERSION();
```

You should see output like this:

```
+-----------+
| version() |
+-----------+
| 8.0.34    |
+-----------+
1 row in set (0.00 sec)
```

This confirms that your connection to the Elestio-hosted MySQL service is working correctly.

How-To Guides

Creating a Database

MySQL is a leading open-source relational database management system (RDBMS) known for its reliability, scalability, and ease of use. Setting up a database properly in MySQL is crucial for ensuring long-term maintainability, performance, and security of applications. This guide walks through different ways to create a MySQL database: using the MySQL CLI, using Docker containers, and using the mysqladmin tool. It also emphasises best practices that should be followed at each step.

Creating a Database Using MySQL CLI

The most common and straightforward way to create a database is by using the MySQL command-line interface (mysql client). First, a connection must be established to the MySQL server using an account with appropriate privileges, typically the root account or a designated administrative user.

Connect to MySQL:

Connect to MySQL:

```
mysql -u root -p
```

To connect to a remote MySQL database using the MySQL CLI, you need to specify the host's IP address or domain name using the `-h` flag along with the username and password:

```
mysql -h <remote_host> -P <port> -u <username> -p
```

You will be prompted to enter the password for the root user. Upon successful login, the MySQL shell opens where SQL queries can be executed.

Create a New Database

To create a database with default settings:

```
CREATE DATABASE mydatabase;
```

However, it is a best practice to explicitly define the character set and collation. This prevents potential problems with encoding and sorting, especially when dealing with multilingual data or special characters. The recommended character set for modern applications is utf8mb4, which fully supports Unicode.

Create a database with specific character set and collation:

```
CREATE DATABASE mydatabase CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

You can verify that the database was created by listing all databases:

```
SHOW DATABASES;
```

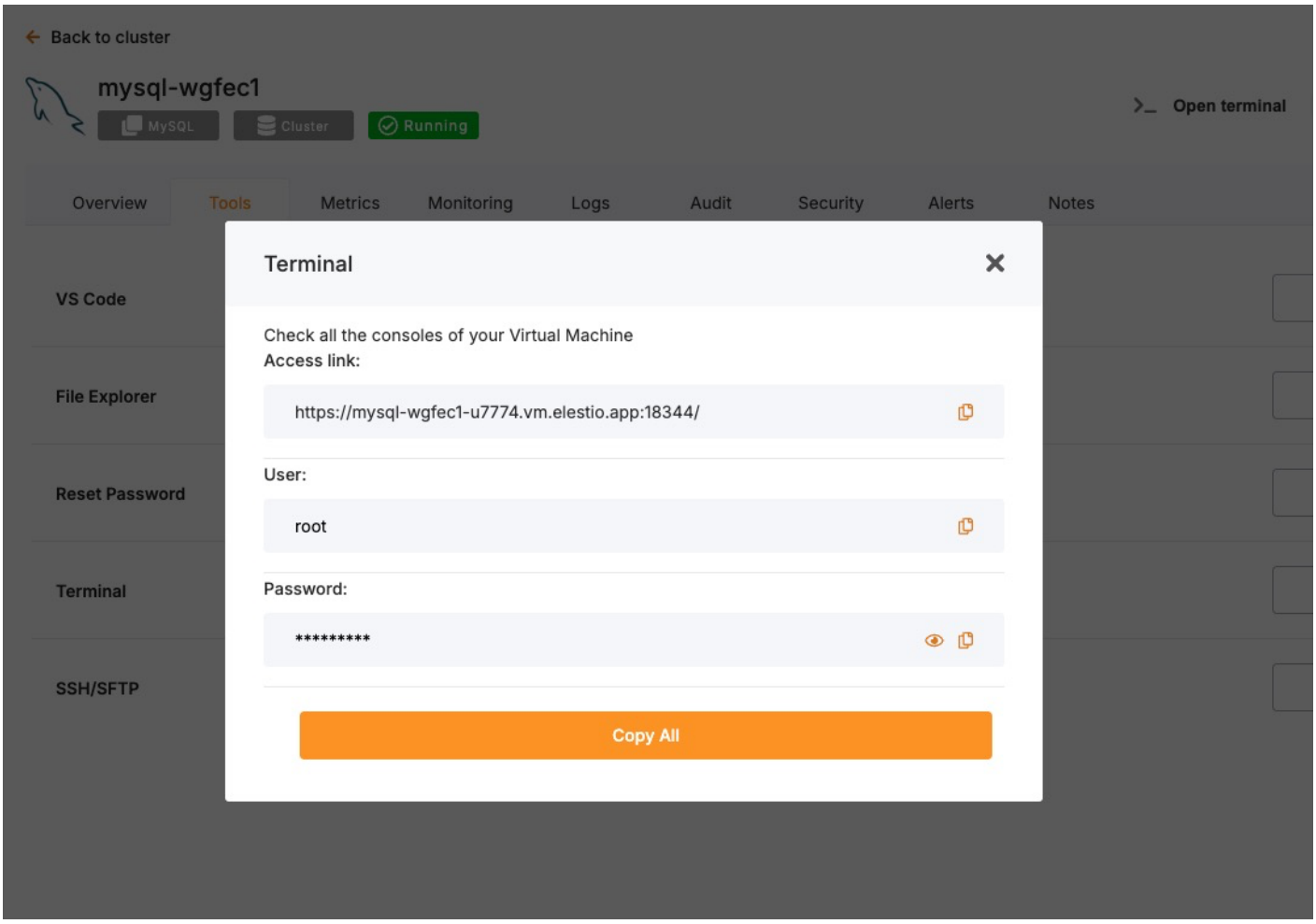
Explicitly setting the character set ensures data consistency and minimizes future migration issues. Additionally, defining these settings at creation time avoids relying on server defaults, which can vary across different environments.

Creating a Database in Docker

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

Access Elestio Terminal

Head over to your deployed MySQL service dashboard and head over to **Tools > Terminal**. Use the credentials provided there to log in to your terminal.



Once you are in your terminal, run the following command to head over to the correct directory to perform the next steps

```
cd /opt/app/
```

Access the MySQL Container Shell

Instead of pulling an image or running the container manually, use Docker Compose to interact with your running container. As you are using Elestio, it will already be a Docker compose:

```
docker-compose exec mysql bash
```

Inside the container, access the MySQL shell:

```
mysql -u root -p
```

Create Database

Now, to create a database, use the following command. This command tells MySQL to create a new logical database called `mydatabase`. By default, it inherits settings like encoding and collation from the template database (template1), unless specified otherwise.

```
CREATE DATABASE mydatabase;
```

Creating a Database Using `mysqladmin`

The `mysqladmin` utility provides a non-interactive way to create databases. It is particularly useful for automation scripts and quick administrative tasks.

To create a database:

```
mysqladmin -h <host> -P <port> -u root -p create mydatabase
```

One limitation of `mysqladmin` is that it does not allow specifying character set and collation at creation time. Therefore, if these settings need to be controlled explicitly (which is generally recommended), it is better to use the mysql CLI instead.

Before using mysqladmin, ensure the MySQL server is running. On traditional installations, check the status:

```
sudo systemctl status mysql
```

If the service is not active, it can be started with:

```
sudo systemctl start mysql
```

Best Practices for Creating Databases

Use Meaningful Names

Choosing clear and descriptive names for databases helps in organisation and long-term maintenance. Avoid generic names like testdb or database1, as they do not convey the database’s role or content. Instead, choose names that reflect the kind of data stored or the application it supports, such as customer_data, sales_records, or analytics_db. Meaningful names improve clarity for developers, DBAs, and future maintainers who need to quickly understand the purpose of each database without relying heavily on documentation.

Follow Naming Conventions

A standardized naming convention across all environments and teams simplifies database management and reduces confusion. MySQL database names are case-sensitive on Unix-based systems, so consistent use of lowercase letters is recommended. Use underscores to separate words (e.g., order_details) rather than camelCase or spaces. This avoids the need for extra quoting in SQL queries and prevents platform-specific bugs. Additionally, avoid using reserved MySQL keywords or special characters in database names, as these can lead to parsing errors and unexpected behaviour.

Restrict User Permissions

Granting only the minimum required permissions significantly strengthens database security and reduces the likelihood of accidental damage or data leaks. Following the Principle of Least Privilege, reporting users should only be given SELECT access, while application users may require `SELECT`, `INSERT`, `UPDATE`, and `DELETE` rights. Only a few trusted administrative users should have powerful privileges like `ALTER`, `DROP`, or `GRANT`. Avoid assigning superuser access unless absolutely necessary. Creating user roles or groups with defined scopes can help standardise permission levels across teams and services.

Enable Backups

Regular backups are critical to ensure business continuity and safeguard against data loss from unexpected events such as accidental deletions, server crashes, or software bugs. MySQL provides tools like mysqldump for logical backups of individual databases, and mysqlpump or xtrabackup for more advanced use cases. It’s good practice to schedule automated backups using cron jobs or database orchestration tools. Backup files should be stored securely and regularly tested for restoration to verify that the process works as expected during emergencies.

Monitor Performance

Ongoing performance monitoring is essential to maintain the responsiveness and stability of MySQL databases. Monitoring tools like `performance_schema`, `information_schema`, or external platforms like Percona Monitoring and Management (PMM) help identify slow queries, locked transactions, and system resource bottlenecks. Use `EXPLAIN` and `ANALYZE` to understand query plans and optimize indexes. Keeping an eye on connection stats, query latency, and buffer pool usage allows for timely tuning and ensures efficient database operations at scale.

Common Issues and Their Solutions

Here’s a table summarizing common problems faced during database creation and how to resolve them:

Issue	Cause	Solution
ERROR 1044 (42000): Access denied for user	The connected user does not have the CREATE privilege.	Connect as a user with administrative privileges or grant necessary permissions.
ERROR 1007 (HY000): Can't create database; database exists	Attempting to create a database that already exists.	Choose a different name or drop the existing database if appropriate using DROP DATABASE.
Can't connect to MySQL server on 'localhost'	MySQL server is not running, or incorrect connection parameters are used.	Start the MySQL service and verify network and authentication parameters.
Collation or character set issues later in application	Database created without explicitly specifying character set or collation.	Always specify utf8mb4 and a collation like utf8mb4_unicode_ci during database creation.
Docker MySQL container refuses connections	MySQL container not ready or port mappings not correctly set.	Check container logs with docker-compose logs mysql and verify port exposure settings in docker-compose.yml.

Upgrading to a Major Version

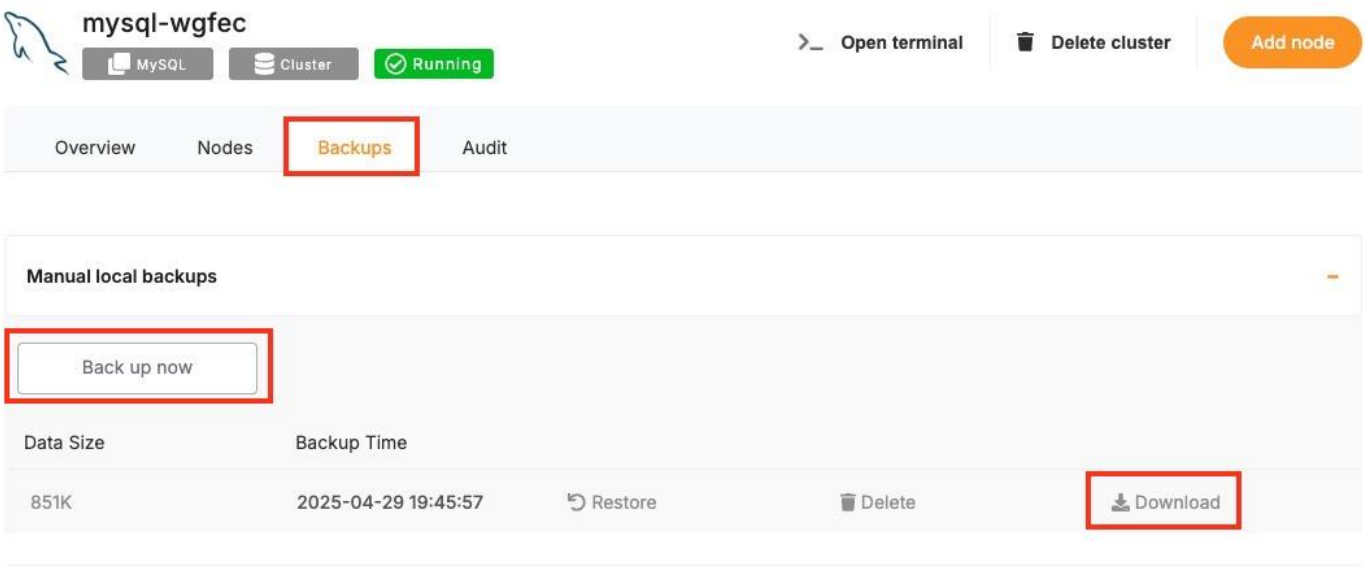
Upgrading a database service on Elestio can be done without creating a new instance or performing a full manual migration. Elestio provides a built-in option to change the database version directly from the dashboard. This is useful for cases where the upgrade does not involve breaking changes or when minimal manual involvement is preferred. The version upgrade process is handled by Elestio internally, including restarting the database service if required. This method reduces the number of steps involved and provides a way to keep services up to date with minimal configuration changes.

Log In and Locate Your Service

To begin the upgrade process, log in to your Elestio dashboard and navigate to the specific database service you want to upgrade. It is important to verify that the correct instance is selected, especially in environments where multiple databases are used for different purposes such as staging, testing, or production. The dashboard interface provides detailed information for each service, including version details, usage metrics, and current configuration. Ensure that you have access rights to perform upgrades on the selected service. Identifying the right instance helps avoid accidental changes to unrelated environments.

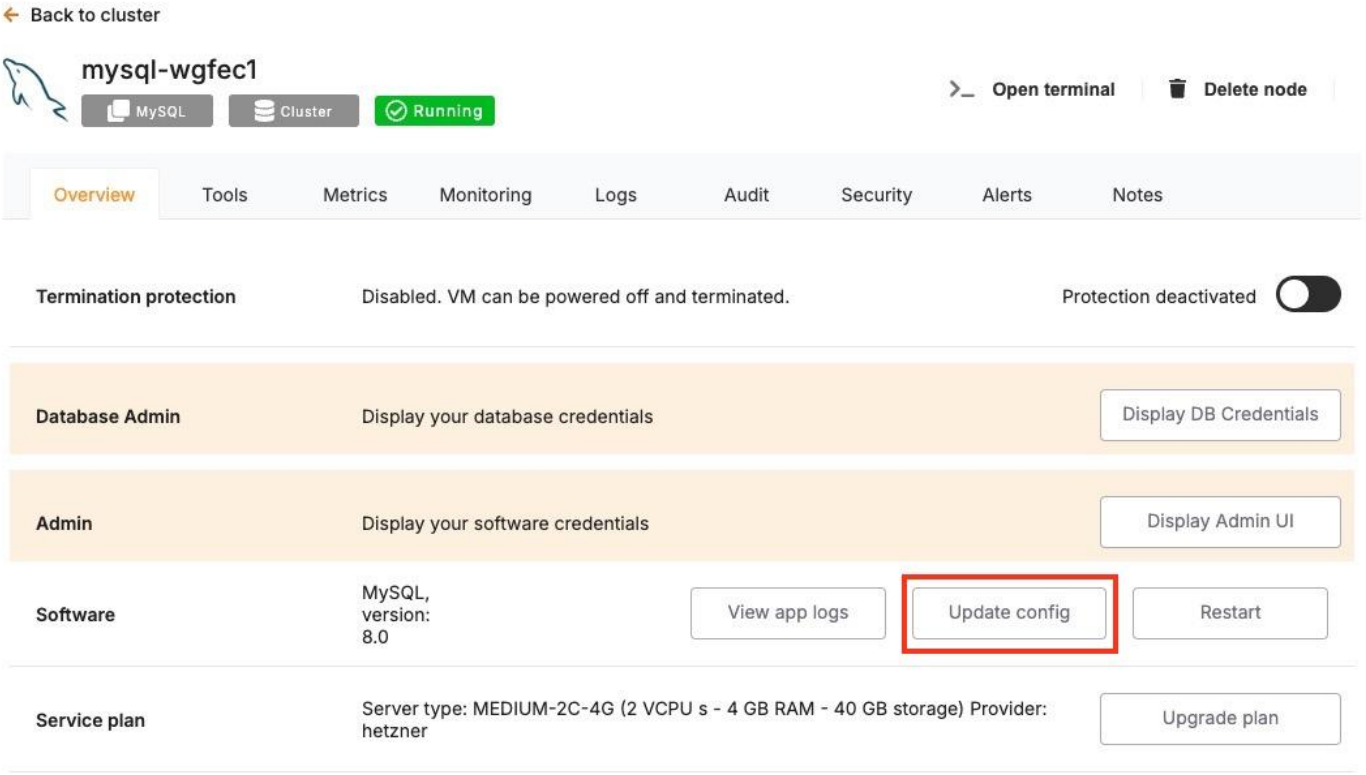
Back Up Your Data

Before starting the upgrade, create a backup of your database. A backup stores the current state of your data, schema, indexes, and configuration, which can be restored if something goes wrong during the upgrade. In Elestio, this can be done through the **Backups** tab by selecting **Back up now** under Manual local backups and **Download** the backup file. Scheduled backups may also be used, but it is recommended to create a manual one just before the upgrade. Keeping a recent backup allows quick recovery in case of errors or rollback needs. This is especially important in production environments where data consistency is critical.

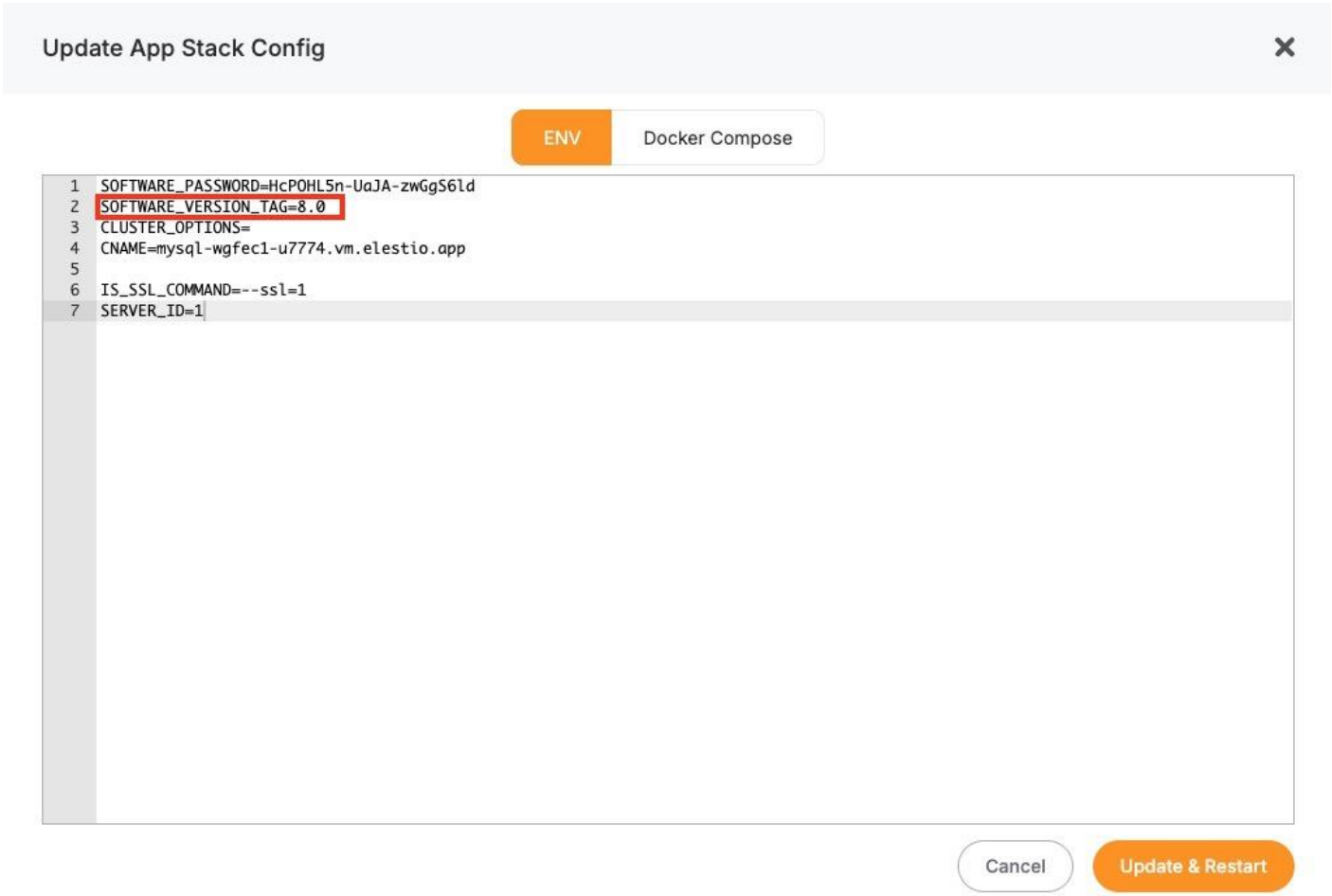


Select the New Version

Once your backup is secure, proceed to the **Overview** and then **Software > Update config** tab within your database service page.



Here, you'll find an option labeled **ENV**. In the **ENV** menu, change the desired database version to `SOFTWARE_VERSION`. After confirming the version, Elestio will begin the upgrade process automatically. During this time, the platform takes care of the version change and restarts the database if needed. No manual commands are required, and the system handles most of the operational aspects in the background.



Monitor the Upgrade Process

The upgrade process may include a short downtime while the database restarts. Once it is completed, it is important to verify that the upgrade was successful and the service is operating as expected. Start by checking the logs available in the Elestio dashboard for any warnings or errors during the process. Then, review performance metrics to ensure the database is running normally and responding to queries. Finally, test the connection from your client applications to confirm that they can interact with the upgraded database without issues.

Installing and Updating an Extension

MySQL supports a variety of **plugins** that extend the functionality of the database engine. These plugins add features like authentication methods, full-text search improvements, audit logging, and more. Popular examples include `auth_socket`, `validate_password`, and `audit_log`. In Elestio-hosted MySQL instances, many common plugins are already available and can be enabled or disabled as needed. This guide explains how to install, manage, and troubleshoot MySQL plugins and verify compatibility with different MySQL versions.

Installing and Enabling Plugins

In MySQL, plugins are usually installed globally at the server level, not per-database. If the plugin binaries are available, they can be loaded dynamically at runtime without restarting the server.

Start by connecting to your MySQL database using a client like mysql:

```
mysql -u root -p -h your-elestio-hostname
```

To enable a plugin, use the INSTALL PLUGIN command. For example, to enable the `validate_password` plugin:

```
INSTALL PLUGIN validate_password SONAME 'validate_password.so';
```

You can verify that the plugin is installed by checking the plugin list:

```
SHOW PLUGINS;
```

To enable a plugin automatically at server startup, add its configuration in the my.cnf file. However, for managed Elestio instances, this may require support team intervention unless custom configuration access is provided.

Checking Plugin Availability & Compatibility

Plugins must be compiled for the specific MySQL version and platform. Before upgrading MySQL or installing a new plugin, verify that a compatible version is available for your target setup. You can find plugin binaries under the plugin_dir, which you can locate with:

```
SHOW VARIABLES LIKE 'plugin_dir';
```

To check if a specific plugin is installed and active:

```
SELECT * FROM INFORMATION_SCHEMA.PLUGINS WHERE PLUGIN_NAME = 'validate_password';
```

If a plugin is incompatible or missing from the plugin_dir, the server will return an error when you attempt to install it. In this case, contact Elestio support to request installation or confirm version compatibility.

Updating or Uninstalling Plugins

After a MySQL version upgrade, some plugins may need to be reinstalled or updated. If a plugin is malfunctioning after an upgrade, it is good practice to uninstall and reinstall it:

```
UNINSTALL PLUGIN validate_password;
INSTALL PLUGIN validate_password SONAME 'validate_password.so';
```

Not all plugins support automatic upgrades. You should consult plugin-specific documentation or Elestio’s compatibility matrix before proceeding.

Troubleshooting Common Plugin Issues

Issue	Cause	Resolution
Can't open shared library	Plugin binary not found in plugin_dir	Check if the `.so` file exists and has correct permissions; contact Elestio if needed
Plugin already installed	Attempting to install a plugin that is already active	Use `SHOW PLUGINS` to verify and avoid duplicate installation
Permission denied	Current user lacks `SUPER` privilege	Log in as a user with `SUPER` or administrative rights
Plugin is not loaded at startup	Plugin not defined in configuration file	Contact Elestio to add it to the MySQL startup config (`my.cnf`)

Security Considerations

Plugins can have significant control over database behavior. Only enable trusted plugins from verified sources. Avoid enabling plugins you do not need, as they can introduce security or performance risks. Always test plugin behavior in a staging environment before deploying to production.

Creating Manual Backups

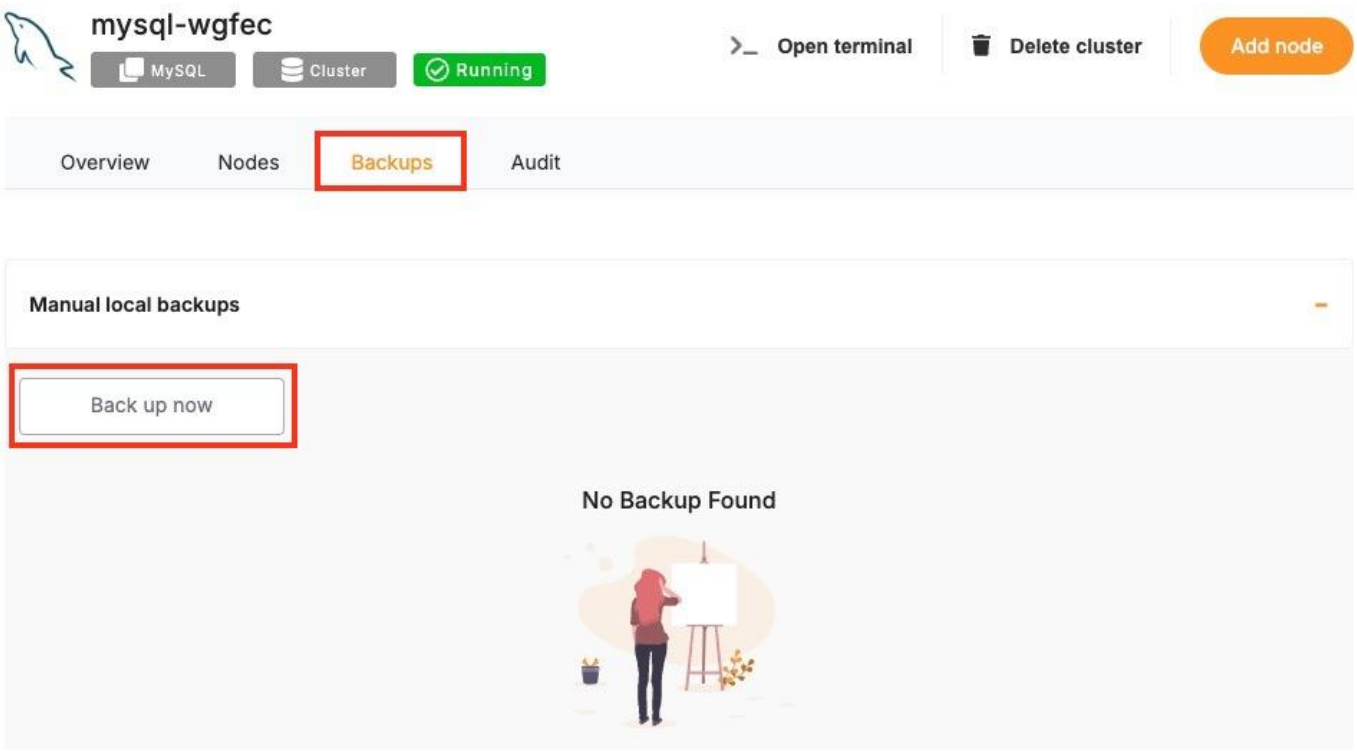
Regular backups are a key part of managing a MySQL deployment. While Elestio provides automated backups by default, you may want to perform manual backups for specific reasons, such as preparing for a major change, keeping a local copy, or testing backup automation. This guide walks through how to create MySQL backups on Elestio using multiple approaches. It covers manual backups through the Elestio dashboard, using MySQL CLI tools, and Docker Compose-based setups. It also includes advice for backup storage, retention policies, and automation using scheduled jobs.

Manual Service Backups on Elestio

If you're using Elestio's managed MySQL service, the easiest way to create a manual backup is through the dashboard. This built-in method creates a full snapshot of your current database state and stores it within Elestio's infrastructure. These backups are tied to your service and can be restored through the same interface. This option is recommended when you need a quick, consistent backup without using any terminal commands.

To trigger a manual backup from the Elestio dashboard:

- Log in to the Elestio dashboard and navigate to your MySQL service/cluster.
- Click the **Backups** tab in the service menu.
- Select **Back up now** to generate a snapshot.

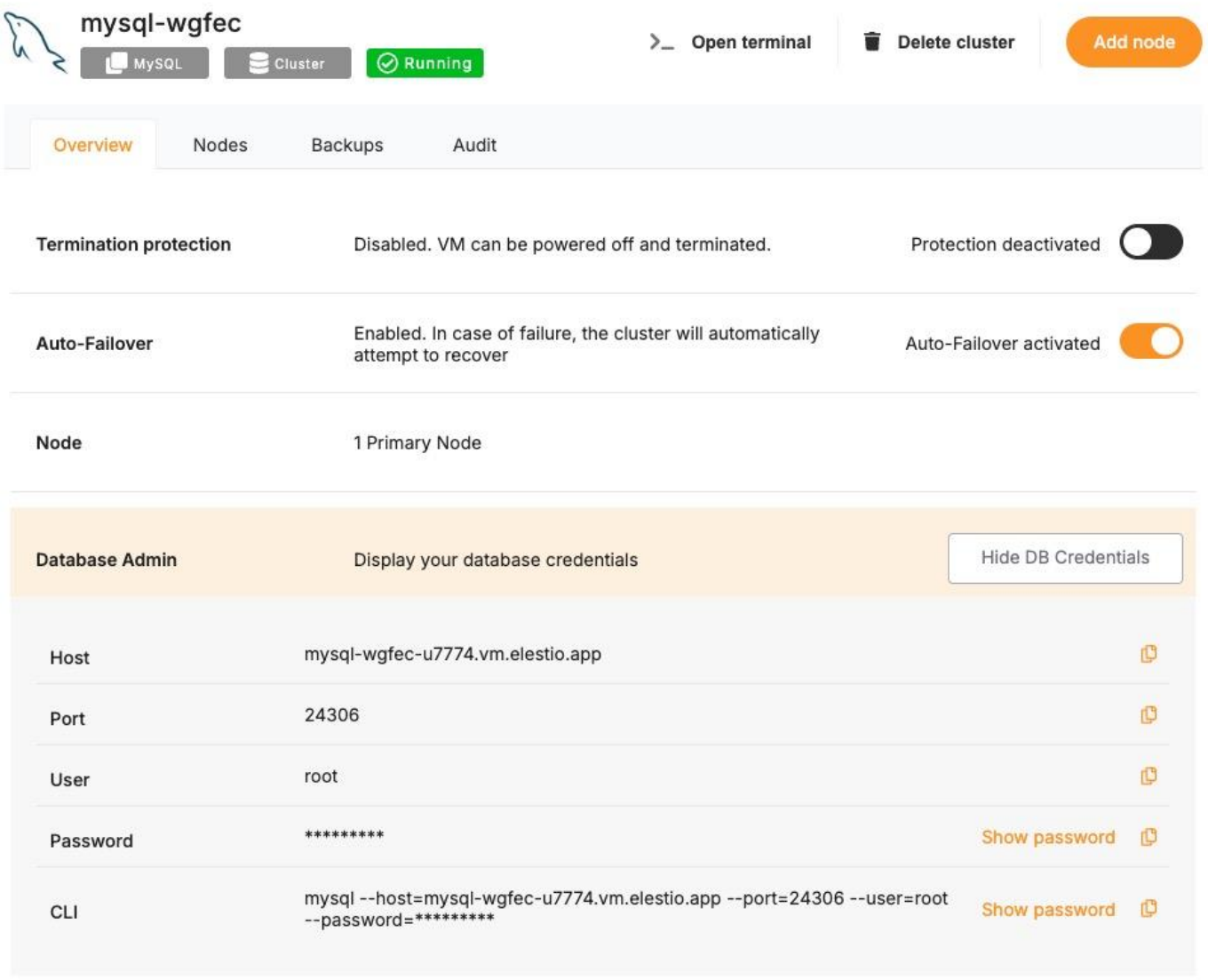


Manual Backups Using MySQL CLI

MySQL provides a set of command-line tools that are useful when you want to create backups from your terminal. These include `mysqldump` for exporting databases and mysql for connectivity and basic queries. This approach is useful when you need to store backups locally or use them with custom automation workflows. The CLI method gives you full control over the backup format and destination.

Collect Database Connection Info

To use the CLI tools, you'll need the database host, port, name, username, and password. These details can be found in the **Overview** section of your MySQL service in the Elestio dashboard.



Back Up with mysqldump

Use mysqldump to export the database to a .sql file. This file can later be used to recreate the database or specific tables.

```
mysqldump -h <host> -P <port> -u <username> -p<password> <database_name> > <output_file>.sql
```

- Replace the placeholders with actual values from your Elestio dashboard.
- The -p<password> flag must not have a space between -p and the password.

Example:

```
mysqldump -h mysql-example.elestio.app -P 24306 -u elestio -pelestioPass mydb > mydb_backup.sql
```

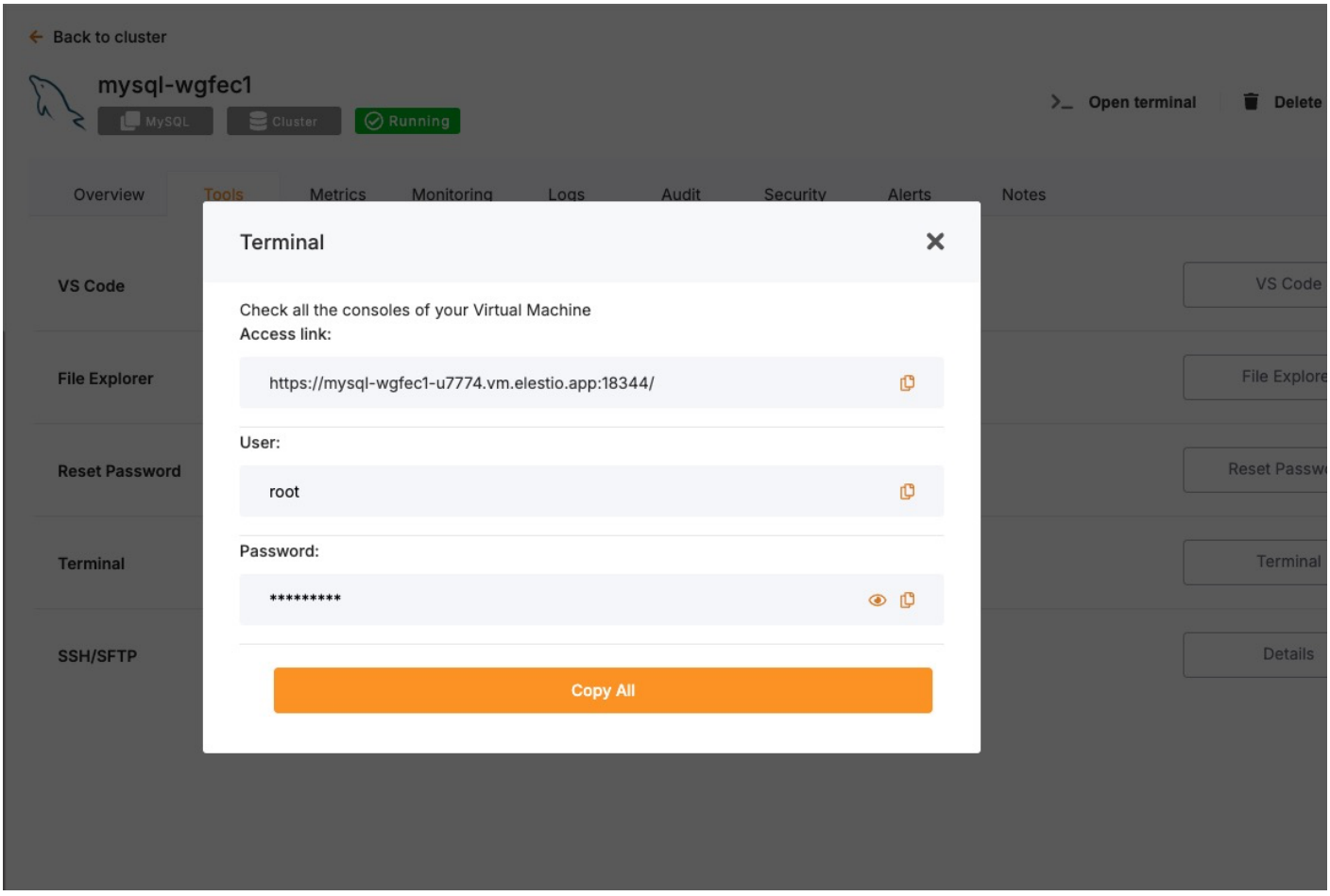
You can add the `--single-transaction` flag for InnoDB tables to ensure consistency during the dump.

Manual Backups Using Docker Compose

If your MySQL database is deployed through a Docker Compose setup on Elestio, you can run the mysqldump command from within the running container. This is useful when the tools are installed inside the container environment and you want to keep everything self-contained. The backup can be created inside the container and then copied to your host system for long-term storage or transfer.

Access Elestio Terminal

Head over to your deployed MySQL service dashboard and go to **Tools > Terminal**. Use the credentials provided there to log in to your terminal.



Once you are in your terminal, navigate to the correct directory:

```
cd /opt/app/
```

Run mysqldump Inside the Container

Use this command to run the backup from within the MySQL container. Ensure environment variables like `MYSQL_USER`, `MYSQL_PASSWORD`, and `MYSQL_DATABASE` are defined, or replace them with actual values.

```
docker-compose exec mysql \
  bash -c "mysqldump -u \$MYSQL_USER -p\$MYSQL_PASSWORD \$MYSQL_DATABASE > /tmp/manual_backup.sql"
```

This command saves the backup to /tmp/manual_backup.sql inside the container.

Copy Backup to Host

Once the backup is created inside the container, use the following command to copy it to your host system:

```
docker cp $(docker-compose ps -q mysql):/tmp/manual_backup.sql ./manual_backup.sql
```

This creates a local copy of the backup file, which you can then upload to external storage or keep for versioned snapshots.

Backup Storage & Retention Best Practices

Once backups are created, they should be stored securely and managed with a clear retention policy. Proper naming, encryption, and rotation reduce the risk of data loss and help during recovery. Use timestamped filenames to identify when the backup was created. External storage services such as AWS S3, Backblaze B2, or an encrypted server volume are recommended for long-term storage.

Guidelines to follow:

- **Name backups clearly:** `mydb_backup_2025_04_29.sql`
- **Store in secure, off-site storage** if possible.
- **Retain 7 daily backups, 4 weekly backups, and 3–6 monthly backups.**
- **Remove old backups automatically** to save space using automation or scripts.

By combining storage hygiene with regular scheduling, you can maintain a reliable backup history and reduce manual effort.

Automating Manual Backups (cron)

Manual backup commands can be scheduled using tools like cron on Linux-based systems. This allows you to regularly back up your database without needing to run commands manually. Automating the process also reduces the risk of forgetting backups and ensures more consistent retention.

Example: Daily Backup at 2 AM

Open your crontab file for editing:

```
crontab -e
```

Then add a job like the following:

```
0 2 * * * mysqldump -h db.vm.elestio.app -P 24306 -u elestio -pelestioPass mydatabase > /backups/backup_$(date +\%F).sql
```

- This will create a timestamped .sql file every day at 2 AM.
- Make sure the /backups/ directory exists and is writable by the user running the cron job.

You can also compress the backup or upload it to cloud storage in the same script using tools like gzip, rclone, or aws-cli.

Restoring a Backup

Restoring backups is essential for recovery, environment duplication, or rollback scenarios. Elestio supports restoring backups both through its built-in dashboard and via command-line tools like mysql and mysqldump. You can also restore from inside Docker Compose environments. This guide provides detailed steps for full and partial restores using each method and explains how to address common errors that occur during restoration.

Restoring from a Backup via Terminal

This method is used when you've created a `.sql` dump file using mysqldump. You can restore it using the mysql command-line client. This approach is useful for restoring backups to new environments, during version upgrades, or testing data locally.

Create the target database if it does not exist

If the database you're restoring into doesn't already exist, you must create it first:

```
mysql -u <username> -p -h <host> -P <port> -e "CREATE DATABASE <database_name>;"
```

You'll be prompted to enter the password after running the command.

Run MySQL to import the backup

This command restores the full contents of the .sql file into the specified database:

```
mysql -u <username> -p -h <host> -P <port> <database_name> < <backup_file>.sql
```

You'll again be prompted for the password. This command restores everything from the dump file, including schema and data.

Restoring via Docker Compose

If your MySQL service is deployed using Docker Compose, you can restore the database inside the container environment. This is useful when MySQL runs in an isolated Docker setup, and you want to handle all backup and restore processes inside that environment.

Copy the backup into the container

Use docker cp to move the .sql file from your host machine to the MySQL container:

```
docker cp ./manual_backup.sql $(docker-compose ps -q mysql):/tmp/manual_backup.sql
```

Run the restore inside the container

Use the mysql CLI tool from within the container to restore the file:

```
docker-compose exec mysql \
  bash -c "mysql -u \$MYSQL_USER -p\"\$MYSQL_PASSWORD\" \$MYSQL_DATABASE < /tmp/manual_backup.sql"
```

Make sure your environment variables in the Docker Compose file (MYSQL_USER, MYSQL_PASSWORD, MYSQL_DATABASE) match the values used here.

Partial Restores

MySQL supports partial restores when the dump file is created with selective options in mysqldump . For example, you can restore just a specific table or only schema definitions.

Restore a specific table

If you created a dump for a specific table using mysqldump -t, you can restore it independently:

```
mysql -u <username> -p -h <host> -P <port> <database_name> < <table_dump_file>.sql
```

Restore schema only (no data)

To restore only the schema (no table contents), ensure that your dump file was created using:

```
mysqldump -u <username> -p -h <host> -P <port> --no-data <database_name> > schema_only.sql
```

Then restore it like this:

```
mysql -u <username> -p -h <host> -P <port> <database_name> < schema_only.sql
```

Partial restores work best when the original backup was generated with the appropriate level of granularity.

Common Errors & How to Fix Them

Errors during restore are often caused by permission issues, incorrect formats, or existing conflicting objects. Understanding the error messages and their causes will help you recover faster and avoid data loss.

1. Access denied for user

```
ERROR 1045 (28000): Access denied for user 'user'@'host'
```

Ensure you are using the correct username/password and that the user has privileges to access the target database.

2. Table already exists

```
ERROR 1050 (42S01): Table 'my_table' already exists
```

Either drop the target database before restoring:

```
mysql -u <username> -p -h <host> -P <port> -e "DROP DATABASE <database_name>;"
mysql -u <username> -p -h <host> -P <port> -e "CREATE DATABASE <database_name>;"
```

Or manually drop the conflicting tables before restore.

3. ERROR 1064 (Syntax Error)

```
ERROR 1064 (42000): You have an error in your SQL syntax...
```

Check if you’re trying to import a binary or incorrectly formatted file. Ensure you’re using .sql text dump files with the mysql command and not raw .ibd or .frm files.

4. ERROR 1049 (Unknown Database)

```
ERROR 1049 (42000): Unknown database 'mydatabase'
```

The specified database doesn’t exist. Create it manually before restoring.

```
mysql -u <username> -p -h <host> -P <port> -e "CREATE DATABASE mydatabase;"
```

Identifying Slow Queries

Slow queries can degrade the performance of your MySQL-based application, leading to lag, timeouts, or higher resource consumption. On Elestio, whether you’re accessing MySQL via terminal, inside a Docker Compose container, or using MySQL CLI tools, there are structured ways to inspect and optimize query performance. This guide covers how to analyze slow queries, interpret execution plans, and apply performance improvements using techniques like EXPLAIN, slow query logs, and schema analysis.

Analyzing Slow Queries from the Terminal

When connected to a MySQL server from a terminal, you can use native SQL statements and built-in features to analyze the performance of specific queries. This is ideal for diagnosing issues in staging or production without needing container access.

Connect to your MySQL instance via terminal

To begin, log in to your MySQL server using the MySQL client:

```
mysql -u <username> -h <host> -p
```

You’ll be prompted for the password. Once inside, you can start analyzing queries.

Use EXPLAIN to view the execution plan

The EXPLAIN keyword shows how MySQL plans to execute a query. It breaks down how tables are accessed and joined, whether indexes are used, and how many rows are expected to be scanned.

```
EXPLAIN SELECT * FROM orders WHERE customer_id = 42;
```

Review the type, key, rows, and Extra columns in the output. Look out for full table scans (type = ALL), which often signal that an index may be missing.

Check current running queries

To view which queries are actively running and their duration, use:

```
SHOW FULL PROCESSLIST;
```

This can help identify long-running or stuck queries in real time.

Analyzing Inside Docker Compose

If your MySQL service is running inside a Docker Compose setup (as Elestio uses), it may not be directly exposed on your host. In this case, analysis must be done from within the container.

Access the MySQL container

Open a shell inside your MySQL container using Docker Compose:

```
docker-compose exec mysql bash
```

This gives you a command-line shell inside the container.

Connect to MySQL from inside the container

Once inside the container, use the environment-defined credentials to access the database:

```
mysql -u $MYSQL_USER -p$MYSQL_PASSWORD $MYSQL_DATABASE
```

This gives you the same SQL interface as from the host terminal, enabling you to use EXPLAIN, SHOW PROCESSLIST, and performance schema tools.

Enable and view the slow query log

MySQL can log slow queries to a file. This must be enabled in your container's `my.cnf` configuration file:

```
[mysqld]
slow_query_log = 1
slow_query_log_file = /var/log/mysql/slow.log
long_query_time = 1
```

After applying these settings, restart the container. Slow queries taking longer than long_query_time (in seconds) will be logged.

You can then inspect the log file:

```
cat /var/log/mysql/slow.log
```

Using Performance Schema

MySQL’s performance schema and built-in commands help track query statistics over time. This is useful when diagnosing repeat offenders or inefficient patterns.

Enable the performance schema (if not already)

Ensure performance_schema is enabled in your MySQL configuration:

```
[mysqld]
performance_schema=ON
```

Restart the container after updating the config.

Identify top queries using statement summaries

This SQL query shows which SQL statements have the longest average execution times:

```
SELECT digest_text, count_star, avg_timer_wait/1000000000000 AS avg_time_sec
FROM performance_schema.events_statements_summary_by_digest
ORDER BY avg_timer_wait DESC
LIMIT 10;
```

This helps you find the most resource-intensive queries over time.

Understanding the MySQL Execution Plan

Reading the output of EXPLAIN is essential to understand how MySQL processes your query and whether it is using indexes efficiently.

Key output fields to interpret:

- **type**: The join type. Prefer ref, range, or const over ALL (which indicates a full table scan).
- **key**: The index used for the query. A NULL value may indicate a missing index.
- **rows**: Estimated number of rows MySQL will scan. Lower is better.
- **Extra**: Look for warnings like Using temporary or Using filesort, which may suggest suboptimal queries.

Use `EXPLAIN ANALYZE` (available in MySQL 8.0+) to see actual vs. estimated performance:

```
EXPLAIN ANALYZE SELECT * FROM orders WHERE customer_id = 42;
```

Optimizing Queries for Better Performance

Once you've identified inefficient queries, optimization involves rewriting queries, adding indexes, or adjusting the schema.

Common techniques:

- **Add indexes** to columns frequently used in WHERE, JOIN, and ORDER BY.
- **Avoid SELECT*** : Only fetch columns you need to reduce I/O.
- **Use LIMIT** when fetching preview data or paginated results.
- **Re-write joins or subqueries** to reduce temporary tables and filesort operations.
- **Update statistics** with ANALYZE TABLE:

```
ANALYZE TABLE orders;
```

Detect and terminate long-running queries

Long-running queries in MySQL can degrade database performance by consuming system resources like CPU, memory, and disk I/O for extended periods. In production environments such as Elestio, it’s essential to monitor and manage these queries effectively to maintain responsiveness and avoid service disruptions. This guide explains how to detect, analyze, and safely terminate long-running queries in MySQL using terminal tools, Docker Compose setups, and built-in logging features. It also includes preventive strategies for avoiding such queries in the future.

Monitoring Long-Running Queries

When connected to your MySQL instance via the terminal using the MySQL CLI, you can inspect active sessions and identify queries that have been running for an excessive duration. This is useful for spotting inefficient or blocked operations.

To check all current sessions and running queries, execute:

```
SHOW FULL PROCESSLIST;
```

This will return all client connections along with their process ID (Id), command type (Command), execution time in seconds (Time), and the actual SQL query (Info). The Time column indicates how long each query has been executing, allowing you to prioritize the longest-running ones.

If you want to isolate queries that have been running for more than a certain duration, such as 60 seconds, you can filter using the `information_schema.processlist` view:

```
SELECT * FROM information_schema.processlist
WHERE COMMAND != 'Sleep' AND TIME > 60;
```

This command excludes idle connections and focuses only on active queries that may need attention.

Terminating Long-Running Queries Safely

Once you've identified a query that's taking too long, MySQL allows you to stop it using its process ID (Id). This can be done either by cancelling just the query or by killing the entire connection.

To stop the query and leave the connection active, run:

```
KILL QUERY <Id>;
```

This interrupts the execution of the current query but keeps the client connection open.

If the connection is completely stuck or no longer needed, you can terminate it entirely:

```
KILL CONNECTION <Id>;
```

Use this approach with caution, especially in shared environments, as it may interrupt ongoing operations or cause errors for connected applications.

Managing Long-Running Queries

If MySQL is running in a Docker Compose setup on Elestio, you'll first need to access the container to inspect queries. You can do so by opening a shell inside the MySQL container:

```
docker-compose exec mysql bash
```

Once inside the container, connect to the MySQL service using the credentials defined in your environment:

```
mysql -u $MYSQL_USER -p$MYSQL_PASSWORD $MYSQL_DATABASE
```

After connecting, you can run the same SHOW FULL PROCESSLIST and KILL commands to identify and handle long-running queries directly from inside the container environment. The logic and process are identical; the only difference is that you're executing these operations within the container shell.

Using Slow Query Logs

MySQL supports slow query logging, which records statements that exceed a specified execution time. This is useful for long-term analysis and identifying recurring performance issues.

To enable this feature, update your MySQL configuration file (e.g., my.cnf or mysqld.cnf) with the following lines:

```
[mysqld]
slow_query_log = 1
slow_query_log_file = /var/log/mysql/slow.log
long_query_time = 1
```

This setup logs any query taking longer than one second. Once configured, restart the MySQL service to apply the changes.

You can then inspect the log with:

```
cat /var/log/mysql/slow.log
```

To summarize patterns in slow queries, use the mysqldumpslow tool:

```
mysqldumpslow /var/log/mysql/slow.log
```

This helps you identify repetitive or particularly expensive SQL statements based on execution time and frequency.

Analyzing Expensive Queries Over Time

To gain visibility into queries that are consistently slow over time, enable MySQL’s performance_schema. This built-in feature aggregates statistics about SQL statement execution, allowing you to pinpoint inefficiencies.

Make sure performance schema is enabled in your config:

```
[mysqld]
performance_schema = ON
```

Once it’s active, use this query to analyze the most time-consuming query patterns:

```
SELECT digest_text, count_star, avg_timer_wait/1000000000000 AS avg_time_sec
FROM performance_schema.events_statements_summary_by_digest
ORDER BY avg_timer_wait DESC
LIMIT 10;
```

This highlights SQL statements that are not just slow once, but frequently expensive, helping you focus on queries with the biggest overall impact.

Best Practices to Prevent Long-Running Queries

It’s better to prevent long-running queries than to reactively terminate them. A few strategic adjustments in your query design and database configuration can significantly improve performance.

- **Index critical columns** used in WHERE, JOIN, and ORDER BY clauses to speed up lookups and sorting.
- **Avoid SELECT *** in queries — fetch only the necessary columns to reduce result size and memory usage.
- **Use EXPLAIN** to analyze how a query will be executed and whether indexes are being used

```
EXPLAIN SELECT * FROM orders WHERE customer_id = 42;
```

- **Limit result sets** in user-facing tools or admin dashboards using LIMIT clauses to avoid returning large datasets unnecessarily.
- **Set execution time limits** at the session level

```
SET SESSION MAX_EXECUTION_TIME = 2000; -- in milliseconds
```

- **Implement timeouts in applications** and ORMs to prevent client-side hanging when the database becomes slow.
- **Monitor actively** using slow query logs, processlist views, and the performance schema. Consider integrating this into your monitoring stack to set up alerts for unusually long or frequent queries.

Preventing Full Disk Issues

Running out of disk space in a MySQL environment can result in failed writes, temporary unavailability, and even data corruption. MySQL requires space not only for storing table data and indexes, but also for binary logs, temporary tables, transaction logs, and background operations. On platforms like Elestio, while the infrastructure is managed, users are responsible for monitoring data growth, managing logs, and planning for scale. This guide covers how to monitor disk usage, configure alerts, clean up unused data, and follow best practices to prevent full disk scenarios in a MySQL setup.

Monitoring Disk Usage

Effective disk usage monitoring allows you to detect unexpected growth before it becomes critical. A combination of operating system-level checks and MySQL-specific queries gives a complete view of space consumption.

To inspect overall system storage from the terminal or container shell, use:

```
df -h
```

This command shows available and used space for each mount point. Identify the mount that hosts your MySQL data directory—usually `/var/lib/mysql` on Linux systems.

To check database-level usage inside MySQL, connect using the MySQL CLI and run:

```
SELECT table_schema AS db_name,
    ROUND(SUM(data_length + index_length) / 1024 / 1024, 2) AS size_mb
FROM information_schema.tables
GROUP BY table_schema
ORDER BY size_mb DESC;
```

This reveals the size of each database schema in megabytes, including both data and indexes. For insights at the table level, run:

```
SELECT table_name,
    ROUND((data_length + index_length) / 1024 / 1024, 2) AS size_mb
FROM information_schema.tables
```

```
WHERE table_schema = 'your_database_name'
ORDER BY size_mb DESC
LIMIT 10;
```

Replace 'your_database_name' with your actual schema name. This helps pinpoint which tables are growing fastest or consuming disproportionate space.

Configuring Alerts and Cleaning Up Storage

Monitoring alone isn’t enough—automatic alerting and cleanup strategies ensure you’re notified in time and can act without downtime. In Docker Compose setups, container disk usage can be reviewed using:

```
docker system df
```

This shows disk consumption across images, containers, and volumes. To list and inspect unused volumes:

```
docker volume ls
```

And to remove a specific unused volume:

```
docker volume rm <volume-name>
```

Do not remove any volume actively used by MySQL. Before any cleanup, confirm that your database volumes are backed up and not mounted by a running service. Within MySQL, temporary tables, binary logs, and undo logs can consume space rapidly. You can check the binary log directory and purge old logs manually:

```
SHOW BINARY LOGS;
```

To delete older binary logs and reclaim space:

```
PURGE BINARY LOGS BEFORE NOW() - INTERVAL 7 DAY;
```

This deletes logs older than 7 days. Adjust the interval based on your backup retention policy. You can also automate this behavior using the configuration option:

```
[mysqld]
expire_logs_days = 7
```

Managing & Optimizing Temporary Files

MySQL uses temporary files for complex queries, especially those involving large sorts or joins without indexes. These files are stored in the tmpdir directory and can fill up if not managed. Monitor the temp directory using OS tools:

```
du -sh /tmp
```

If temp file usage is consistently high, consider tuning the tmp_table_size and max_heap_table_size variables to reduce reliance on disk-based temporary tables.

To identify tables with excessive unused space, use:

```
SHOW TABLE STATUS WHERE Data_free > 0;
```

These tables may benefit from optimization. Reclaim the unused space by running:

```
OPTIMIZE TABLE your_table_name;
```

This rewrites the table and defragments it, reclaiming disk space. For InnoDB tables, this can also compact the clustered index.

Best Practices for Disk Space Management

Long-term disk health in MySQL requires more than just cleanup—it demands strategic design and active space governance.

- **Avoid storing large files in the database.** Use external object storage for PDFs, images, or videos and store references (e.g., URLs) in the database.
- **Implement data retention policies.** Archive old transactional data to another schema, flat files, or cold storage if it’s no longer queried frequently.

- **Partition large tables** using range or list partitioning to separate older data. Partitioning improves manageability and enables easier purging or archiving.
- **Rotate logs regularly.** Besides binary logs, general logs and error logs should be rotated using tools like logrotate, especially in containerized environments.
- **Monitor InnoDB transaction logs** (the ib_logfile* files). These are critical for crash recovery but should not grow indefinitely. If they become too large, you may need to reconfigure their size safely and restart the service.
- **Store backups offsite.** Backups stored on the same volume as your live database can fill your disk. Use Elestio's backup tools to export backups to cloud storage or another disk.

Checking Database Size and Related Issues

As your MySQL database grows, it’s crucial to track how space is being used across schemas, tables, and indexes. Uncontrolled growth can slow down queries, consume disk space, and complicate backups. While Elestio provides managed infrastructure, database storage optimization is still your responsibility. This guide explains how to inspect database size, find the largest tables and indexes, detect unused or bloated space, and optimize your data layout in MySQL.

Checking Database and Table Sizes

MySQL’s information_schema tables provide insights into how storage is distributed across your databases. This data helps prioritize cleanup, tuning, or archiving strategies. To calculate the total size used by each database:

```
SELECT table_schema AS db_name,
    ROUND(SUM(data_length + index_length) / 1024 / 1024, 2) AS size_mb
FROM information_schema.tables
GROUP BY table_schema
ORDER BY size_mb DESC;
```

This output includes both table data and indexes, giving you an overview of which databases are consuming the most space. To identify the largest tables across all schemas:

```
SELECT table_schema, table_name,
    ROUND((data_length + index_length) / 1024 / 1024, 2) AS total_size_mb
FROM information_schema.tables
ORDER BY total_size_mb DESC
LIMIT 10;
```

This helps pinpoint space-heavy tables so you can review their contents, indexes, or retention policy. To break down data size versus index size for the top tables in a specific schema:

```
SELECT table_name,
    ROUND(data_length / 1024 / 1024, 2) AS table_mb,
    ROUND(index_length / 1024 / 1024, 2) AS index_mb
FROM information_schema.tables
WHERE table_schema = 'your_database'
ORDER BY table_mb DESC
LIMIT 10;
```

Replace '`your_database`' with your actual schema name. A high index-to-data ratio could indicate overly aggressive indexing or opportunities for consolidation.

Detecting Bloat and Unused Space

MySQL tables especially those using the InnoDB storage engine can accumulate unused space over time due to updates, deletes, or internal fragmentation. This can inflate table size and degrade performance.

To list tables with free (unused) space that could be reclaimed:

```
SELECT table_name,
    ROUND(data_free / 1024 / 1024, 2) AS free_space_mb
FROM information_schema.tables
WHERE table_schema = 'your_database'
  AND data_free > 0
ORDER BY free_space_mb DESC
LIMIT 10;
```

Large data_free values may indicate internal fragmentation or deleted rows that haven’t been reclaimed yet. You can recover this space using table optimization. To view the number of rows deleted but not yet reclaimed (estimated):

```
SHOW TABLE STATUS FROM your_database;
```

Check the Rows and Data_free columns for each table. If many rows have been deleted but space hasn’t shrunk, the table may need to be optimized.

Optimizing and Reclaiming Storage

Once bloated or inefficient tables have been identified, MySQL provides several tools for optimization:

Reclaim free space and defragment tables

```
OPTIMIZE TABLE your_table;
```

This command rewrites the table and indexes, reclaiming space and improving performance. It's safe for InnoDB tables and especially useful after large `DELETE` or `UPDATE` operations.

Rebuild fragmented or oversized indexes

If indexes have grown large due to repeated updates or inserts, rebuilding them can reduce size and improve query speed. Use:

```
ALTER TABLE your_table ENGINE=InnoDB;
```

This effectively recreates the table and all associated indexes, helping reclaim space and improve internal ordering.

> Note: Both OPTIMIZE and ALTER ENGINE operations lock the table for a short period. Run these during maintenance windows if the table is actively queried.

Remove or archive old rows

For time-series data or logs, consider deleting or archiving old records:

```
DELETE FROM your_table
WHERE created_at < NOW() - INTERVAL 90 DAY;
```

Use EXPLAIN before executing large deletes to ensure they use indexes efficiently. You may also consider archiving to flat files or cold-storage tables.

Partition large tables for better control

If tables grow continuously (e.g., transaction logs or audit trails), use MySQL's **range partitioning** or **list partitioning**:

```
CREATE TABLE logs (
  id BIGINT,
  created_at DATE,
  ...
)
PARTITION BY RANGE (YEAR(created_at)) (
  PARTITION p2022 VALUES LESS THAN (2023),
  PARTITION p2023 VALUES LESS THAN (2024),
  PARTITION pmax VALUES LESS THAN MAXVALUE
);
```

Partitioning allows you to drop old data in chunks without full table scans or long DELETE operations.

Best Practices for Storage Management

- **Avoid storing large binary data in MySQL.** Store files like images and videos in external object storage and reference them by URL or metadata.
- **Monitor binary logs and purge them periodically.** If replication or point-in-time recovery isn’t needed beyond a certain timeframe, add to your config:

```
expire_logs_days = 7
```

- Or purge manually:

```
PURGE BINARY LOGS BEFORE NOW() - INTERVAL 7 DAY;
```

- **Track backup file size and location.** Ensure backups are stored on a separate volume or offsite to avoid filling the same disk as your live database.
- **Enable slow query logging** to detect inefficient queries that cause unnecessary data scans and table growth.
- **Use monitoring tools** (like Netdata, Prometheus exporters, or custom alert scripts) to track disk consumption trends over time.

Database Migration

Cloning a Service to Another Provider or Region

Migrating or cloning services across cloud providers or geographic regions is a critical part of modern infrastructure management. Whether you're optimizing for latency, preparing for disaster recovery, meeting regulatory requirements, or simply switching providers, a well-planned migration ensures continuity, performance, and data integrity. This guide outlines a structured methodology for service migration, applicable to most cloud-native environments.

Pre-Migration Preparation

Before initiating a migration, thorough planning and preparation are essential. This helps avoid unplanned downtime, data loss, or misconfiguration during the move:

- **Evaluate the Current Setup**: Begin by documenting the existing service's configuration. This includes runtime environments (container images, platform versions), persistent data (databases, object storage), network rules (ports, firewalls), and application dependencies (APIs, credentials, linked services).
- **Define the Migration Target**: Choose the new cloud provider or region you plan to migrate to. Confirm service compatibility, resource limits, and geographic latency requirements. If you're replicating an existing environment, make sure the target region supports the same compute/storage features and versions.
- **Provision the Target Environment**: Set up the target infrastructure where the service will be cloned. This could involve creating new Kubernetes clusters, VM groups, container registries, databases, or file storage volumes—depending on your stack.
- **Backup the Current Service**: Always create a full backup or snapshot of the current service and its associated data before proceeding. This acts as a rollback point in case of migration issues and ensures recovery in the event of failure.

Cloning Execution

The first step in executing a clone is to replicate the configuration of the original service in the target environment. This involves deploying the same container image or service binary using the same runtime settings. If you're using Kubernetes or container orchestrators, this can be done via

Helm charts or declarative manifests. Pay close attention to environment variables, secrets, mounted paths, storage class definitions, and health check configurations to ensure a consistent runtime environment.

Next, you'll need to migrate any persistent data tied to the service. For file-based storage, tools like rsync or rclone are effective for copying volume contents over SSH or cloud storage backends. It's crucial to verify compatibility across disk formats, database versions, and encoding standards to avoid corruption or mismatched behavior.

After replicating the environment and data, it's important to validate the new service in isolation. This means confirming that all application endpoints respond as expected, background tasks or cron jobs are functioning, and third-party integrations (e.g., payment gateways, S3 buckets) are accessible. You should test authentication flows, data read/write operations, and retry logic to ensure the new service is functionally identical. Use observability tools to monitor resource consumption and application logs during this stage.

Once validation is complete, configure DNS and route traffic to the new environment. This might involve updating DNS A or CNAME records, changing cloud load balancer configurations, or applying new firewall rules. For high-availability setups, consider using health-based routing or weighted DNS to gradually transition traffic from the old instance to the new one.

Post-Migration Validation and Optimization

Once the new environment is live and receiving traffic, focus on optimizing and securing the setup:

- **Validate Application Functionality**: Test all integrations, user workflows, and background jobs to confirm proper behavior. Review logs for silent errors or timeouts. Ensure all applications pointing to the service are updated with the new URL or connection string.
- **Monitor Performance**: Analyze load, CPU, memory, and storage utilization. Scale resources as needed, or optimize runtime settings for the new provider/region. Enable autoscaling where applicable.
- **Secure the Environment**: Implement firewall rules, IP restrictions, and access controls. Rotate secrets and validate that no hardcoded credentials or endpoints point to the old service.
- **Cleanup and Documentation**: Once validated, decommission the old setup safely. Update internal documentation with new deployment details, endpoint addresses, and any configuration changes.

Benefits of Cloning

Cloning a database service, particularly for engines like MySQL offers several operational and strategic advantages. It allows teams to test schema migrations, version upgrades, or major application features in an isolated environment without affecting production. By maintaining a cloned copy, developers and QA teams can work against realistic data without introducing risk.

Cloning also simplifies cross-region redundancy setups. A replica in another region can be promoted quickly if the primary region experiences an outage. For compliance or analytics purposes, cloned databases allow for read-only access to production datasets, enabling safe reporting or data processing without interrupting live traffic.

Additionally, rather than building a new environment from scratch, you can clone the database into another provider, validate it, and cut over with minimal disruption. This helps maintain operational continuity and reduces the effort needed for complex migrations.

Database Migration Service for MySQL

Elestio provides a structured approach for migrating MySQL 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 MySQL instance and related resources.
- **Deploy the Target MySQL Service:** Set up a new MySQL instance on Elestio to serve as the destination for your data. It's crucial to match the software version of your current MySQL database to avoid compatibility issues during data transfer. Detailed prerequisites and guidance can be found in Elestio's migration documentation.

Initiating the Migration Process

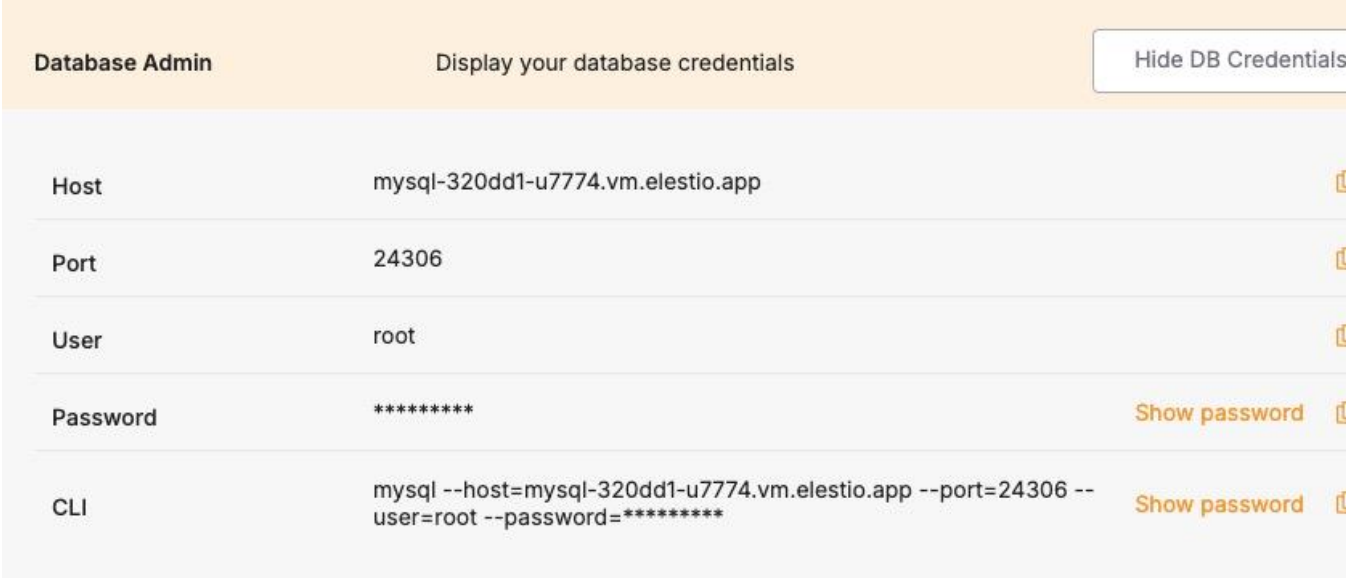
With the preparatory steps completed, you can proceed to migrate your MySQL database to Elestio:

1. **Access the Migration Tool:** Navigate to the overview of your MySQL 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 MySQL database, including:
   - **Hostname:** The address of your current database server.
   - **Port:** The port number on which your MySQL service is running (default is 3306).
   - **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 MySQL service.



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 MySQL

Migrating your MySQL 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 MySQL 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 MySQL instances for performance based on workload requirements. The platform supports the latest MySQL versions, incorporating updates that improve database operations. Its infrastructure handles different workloads and maintains performance during high usage periods.

- **Scalability:** Elestio's MySQL 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.

Manual Migration Using mysqldump and mysql

Manual migrations using MySQL's built-in tools `mysqldump` and `mysql`, are ideal for users who require full control over data export and import, particularly during transitions between providers, database version upgrades, or importing existing self-managed MySQL datasets into Elestio's managed environment. This guide walks through the process of performing a manual migration to and from Elestio MySQL services using command-line tools, ensuring data portability, consistency, and transparency at every step.

When to Use Manual Migration

Manual migration using mysqldump is well-suited for scenarios that demand complete control over the migration process. It is especially useful when transferring databases from a self-hosted MySQL instance, an on-premises server, or another cloud provider into Elestio's managed MySQL service. This method supports one-time imports without requiring ongoing connections between source and destination systems.

It also provides a reliable approach for performing version upgrades. Because mysqldump creates logical backups, the resulting SQL files can be restored into newer MySQL versions with minimal compatibility issues. When Elestio's built-in tools are not applicable such as in migrations from isolated environments or in selective schema transfers manual migration becomes the preferred option. It also enables offline backup archiving, providing users with transportable and restorable datasets independent of platform-specific backup formats.

Performing the Migration

Prepare the Environments

Before starting the migration, ensure that MySQL is properly installed on both the source system and your Elestio service. The source MySQL server must allow network connections (if remote) and have a user with sufficient privileges to export the database, including read access to all necessary tables, views, stored procedures, and triggers.

On the Elestio side, provision a MySQL service through the dashboard. Once it's active, retrieve the connection credentials from the **Database Info** section this includes host, port, database name,

username, and password. Verify that your public IP is allowed under **Cluster Overview > Security > Limit access per IP**, or the MySQL port will not be reachable.

Create a Dump Using mysqldump

Use mysqldump to export the entire source database into a SQL file. This utility serializes the schema, data, indexes, and routines into a plain-text SQL script.

```
mysqldump -h <source_host> -P <source_port> -u <source_user> -p <source_database> > backup.sql
```

You'll be prompted to enter the source password. Once complete, this produces a portable SQL file named backup.sql. You can also include flags such as `--routines` or `--triggers` if your database uses stored procedures or custom triggers.

To avoid restore-time permission issues, consider adding:

```
--skip-add-drop-table --skip-add-locks --set-gtid-purged=OFF
```

This is especially helpful when importing into a different environment or user setup than the source.

Transfer the Dump File to the Target

If your local system differs from the Elestio service access point, transfer the dump file using a secure file transfer tool:

```
scp backup.sql your_user@your_workstation:/path/to/local/
```

Ensure that the file is accessible on the system you plan to use to run the restore. The dump does not need to be uploaded to the Elestio server — restores are executed via a remote connection using MySQL's native protocol.

Create the Target Database

Elestio provisions a default database during setup. If the dump file references a different database name, you may need to create it manually before restore.

Connect to Elestio's MySQL service using the CLI:

```
mysql -h <elestio_host> -P <elestio_port> -u <elestio_user> -p
```

Once inside the MySQL shell, create the target database:

```
CREATE DATABASE target_database CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

This ensures proper character encoding and collation settings for compatibility with modern applications and multilingual content.

Restore Using MySQL Client

With the dump file in place and the target database created, restore the data:

```
mysql -h <elestio_host> -P <elestio_port> -u <elestio_user> -p target_database < /path/to/backup.sql
```

This command connects to the Elestio-managed MySQL instance and executes every statement from the dump file, recreating tables, inserting data, and applying schema-level objects.

Ensure the target user has privileges to create tables, insert data, and apply any stored routines or triggers included in the dump file.

Validate the Migration

After completing the import, verify that the migration was successful by connecting to the Elestio MySQL instance and running checks against key tables and schema components.

Start by listing the tables and checking row counts:

```
SHOW TABLES;
SELECT COUNT(*) FROM your_important_table;
```

Also review any stored procedures, views, or functions if your application depends on them. Confirm the database schema matches the original setup and that your application is able to connect and operate as expected.

If you've updated environment variables or connection strings, make sure these changes are reflected in your deployment or application config. Consider enabling automated backups via Elestio to protect your imported database going forward.

Benefits of Manual Migration

Manual MySQL migration using mysqldump and mysql offers several important advantages:

- **Portability and Compatibility**: Logical SQL dumps can be restored into any MySQL-compatible instance, whether hosted locally, on another cloud, or in containers.
- **Version Flexibility**: Migrate across MySQL versions without relying on binary replication or platform-specific formats.
- **Offline Storage**: SQL files serve as portable backups that can be stored offline, versioned, or archived for disaster recovery.

- **Platform Independence**: Elestio does not enforce proprietary formats standard MySQL tools give you complete control over the migration and restore process.

This method complements Elestio’s automated backup and migration features by enabling custom workflows and one-off imports with full visibility into each stage.

Cluster Management

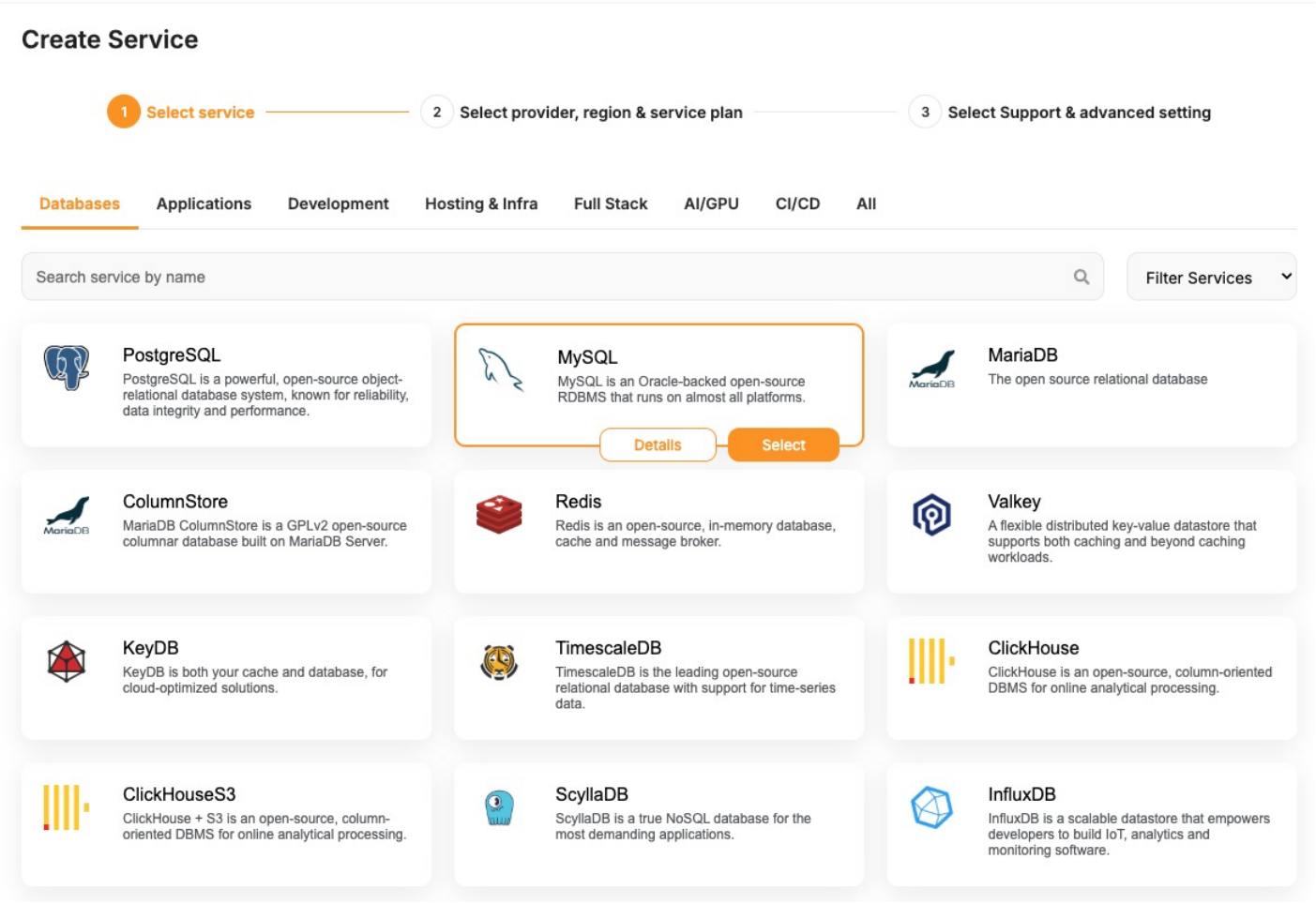
Overview

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.

Deploying a New Cluster

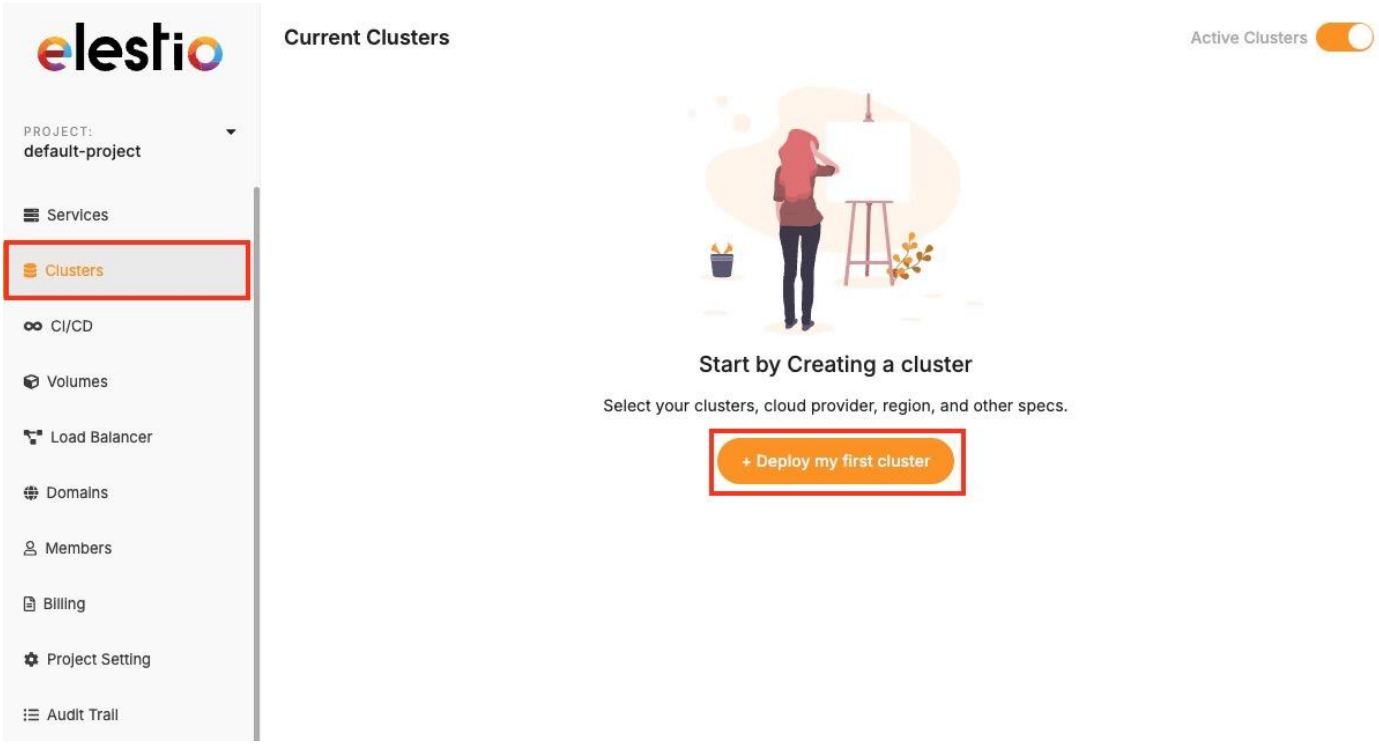
Creating a cluster is a foundational step when deploying services in Elestio. Clusters provide isolated environments where you can run containerized workloads, databases, and applications. Elestio’s web dashboard helps the process, allowing you to configure compute resources, choose cloud providers, and define deployment regions without writing infrastructure code. This guide walks through the steps required to create a new cluster using the Elestio dashboard.

Prerequisites

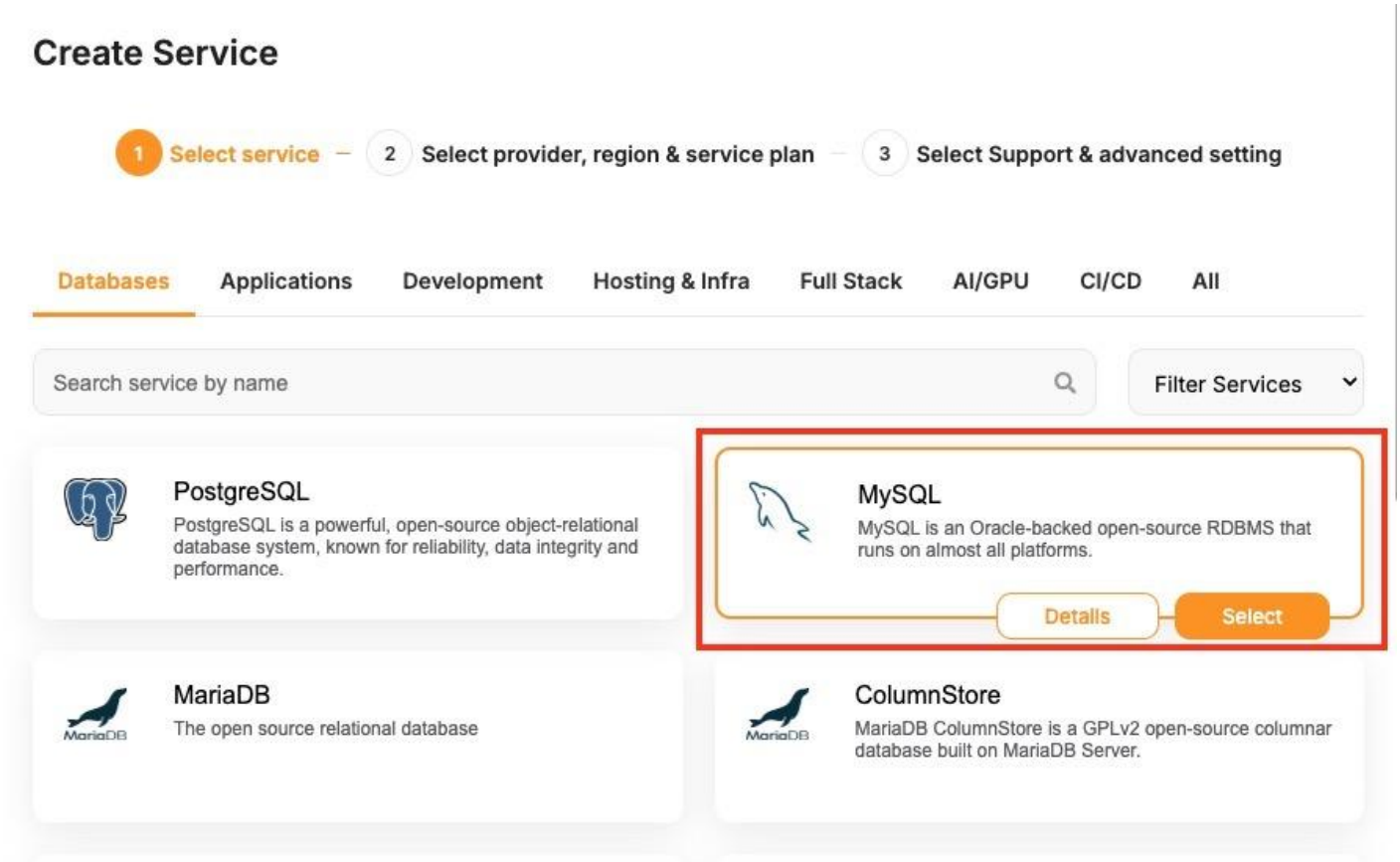
To get started, you’ll need an active Elestio account. If you’re planning to use your own infrastructure, make sure you have valid credentials for your preferred cloud provider (like AWS, GCP, Azure, etc.). Alternatively, you can choose to deploy clusters using Elestio-managed infrastructure, which requires no external configuration.

Creating a Cluster

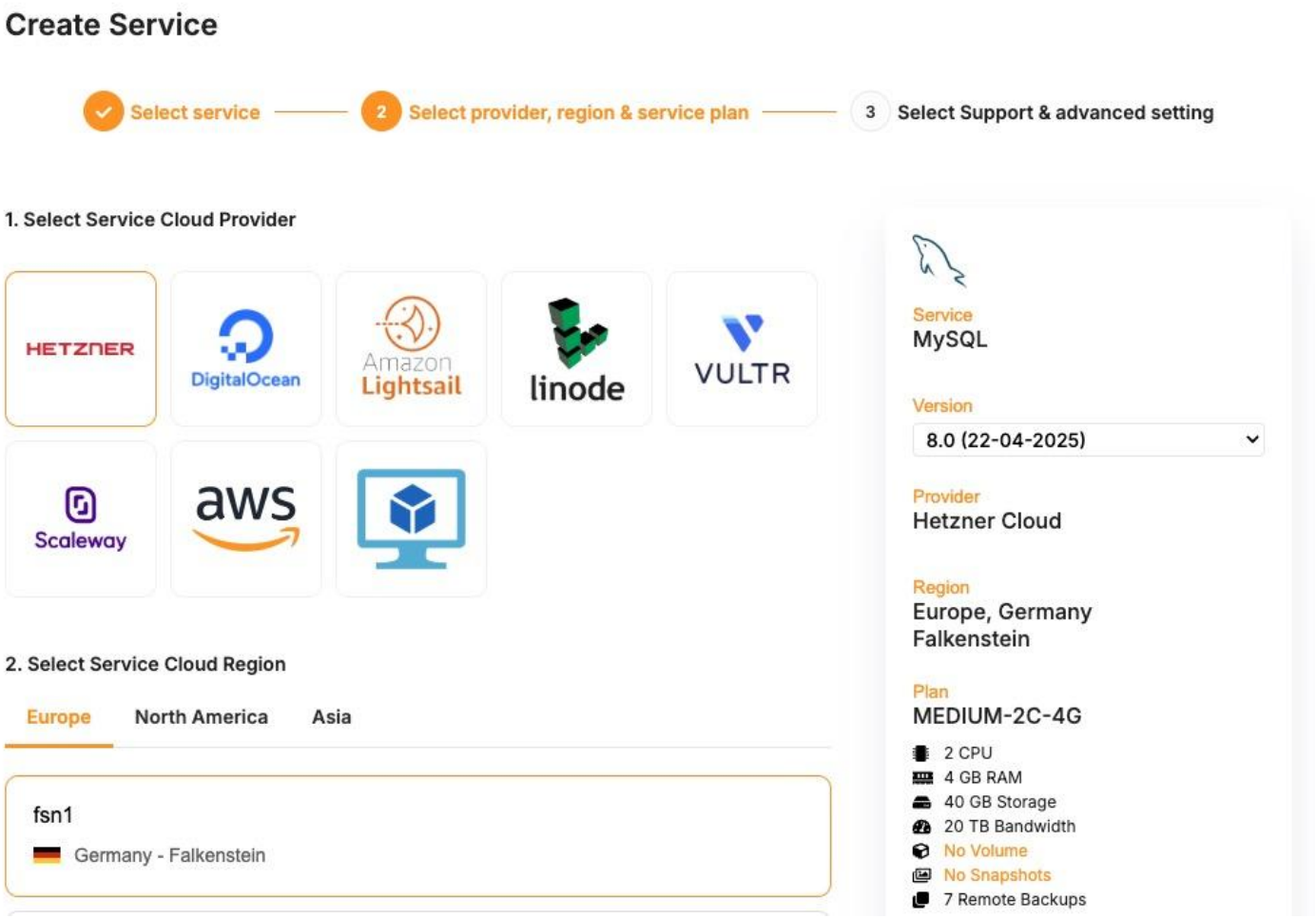
Once you’re logged into the Elestio dashboard, navigate to the **Clusters** section from the sidebar. You’ll see an option to **Create a new cluster**—clicking this will start the configuration process. The cluster creation flow is flexible but simple for defining essential details like provider, region, and resources in one place.



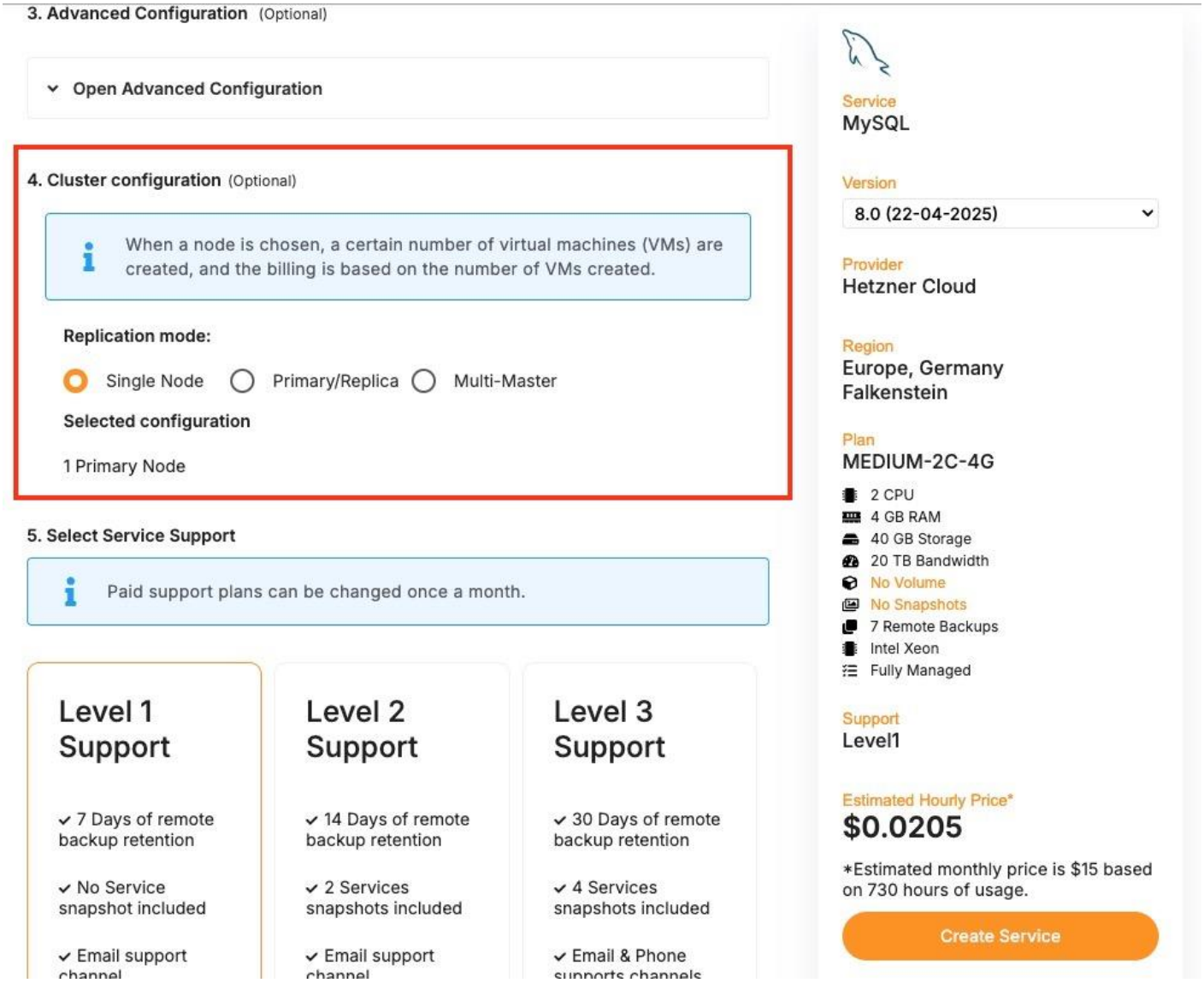
Now, select the database service of your choice that you need to create in a cluster environment. Click on **Select** button as you choose one.



During setup, you'll be asked to choose a hosting provider. Elestio supports both managed and BYOC (Bring Your Own Cloud) deployments, including AWS, DigitalOcean, Hetzner, and custom configurations. You can then select a region based on latency or compliance needs, and specify the number of nodes along with CPU, RAM, and disk sizes per node.



If you're setting up a high-availability cluster, the dashboard also allows you to configure cluster-related details under **Cluster configuration,** where you get to select things like replication modes, number of replicas, etc. After you've configured the cluster, review the summary to ensure all settings are correct. Click the **Create Cluster** button to begin provisioning.



Elestio will start the deployment process, and within a few minutes, the cluster will appear in your dashboard. Once your cluster is live, it can be used to deploy new nodes and additional configurations. Each cluster supports real-time monitoring, log access, and scaling operations through the dashboard. You can also set up automated backups and access control through built-in features available in the cluster settings.

Node Management

Node management plays a critical role in operating reliable and scalable infrastructure on Elestio. Whether you're deploying stateless applications or stateful services like databases, managing the underlying compute units nodes is essential for maintaining stability and performance.

Understanding Nodes

In Elestio, a **node** is a virtual machine that contributes compute, memory, and storage resources to a cluster. Clusters can be composed of a single node or span multiple nodes, depending on workload demands and availability requirements. Each node runs essential services and containers as defined by your deployed applications or databases.

Nodes in Elestio are provider-agnostic, meaning the same concepts apply whether you're using Elestio-managed infrastructure or connecting your own cloud provider (AWS, Azure, GCP, etc.). Each node is isolated at the VM level but participates fully in the cluster's orchestration and networking. This abstraction allows you to manage infrastructure without diving into the complexity of underlying platforms.

Node Operations

The Elestio dashboard allows you to manage the lifecycle of nodes through clearly defined operations. These include:

- **Creating a node**, which adds capacity to your cluster and helps with horizontal scaling of services. This is commonly used when load increases or when preparing a high-availability deployment.
- **Deleting a node**, which removes underutilized or problematic nodes. Safe deletion includes draining workloads to ensure service continuity.
- **Promoting a node**, which changes the role of a node within the cluster—typically used in clusters with redundancy, where certain nodes may need to take on primary or leader responsibilities.

Each of these operations is designed to be safely executed through the dashboard and is validated against the current cluster state to avoid unintended service disruption. These actions are supported by Elestio's backend orchestration, which handles tasks like container rescheduling and

load balancing when topology changes.

Monitoring and Maintenance

Monitoring is a key part of effective node management. Elestio provides per-node visibility through the dashboard, allowing you to inspect **CPU**, **memory**, and **disk utilization** in real time. Each node also exposes **logs**, **status indicators**, and **health checks** to help detect anomalies or degradation early.

In addition to passive monitoring, the dashboard supports active maintenance tasks. You can **reboot a node** when applying system-level changes or troubleshooting, or **drain a node** to safely migrate workloads away from it before performing disruptive actions. Draining ensures that running containers are rescheduled on other nodes in the cluster, minimizing service impact.

For production setups, combining resource monitoring with automation like scheduled reboots, log collection, and alerting can help catch issues before they affect users. While Elestio handles many aspects of orchestration automatically, having visibility at the node level helps teams make informed decisions about scaling, updates, and incident response.

Cluster-wide resource graphs and node-level metrics are also useful for capacity planning. Identifying trends such as memory saturation or disk pressure allows you to preemptively scale or rebalance workloads, reducing the risk of downtime.

Adding a Node

As your application usage grows or your infrastructure requirements change, scaling your cluster becomes essential. In Elestio, you can scale horizontally by adding new nodes to an existing cluster. This operation allows you to expand your compute capacity, improve availability, and distribute workloads more effectively.

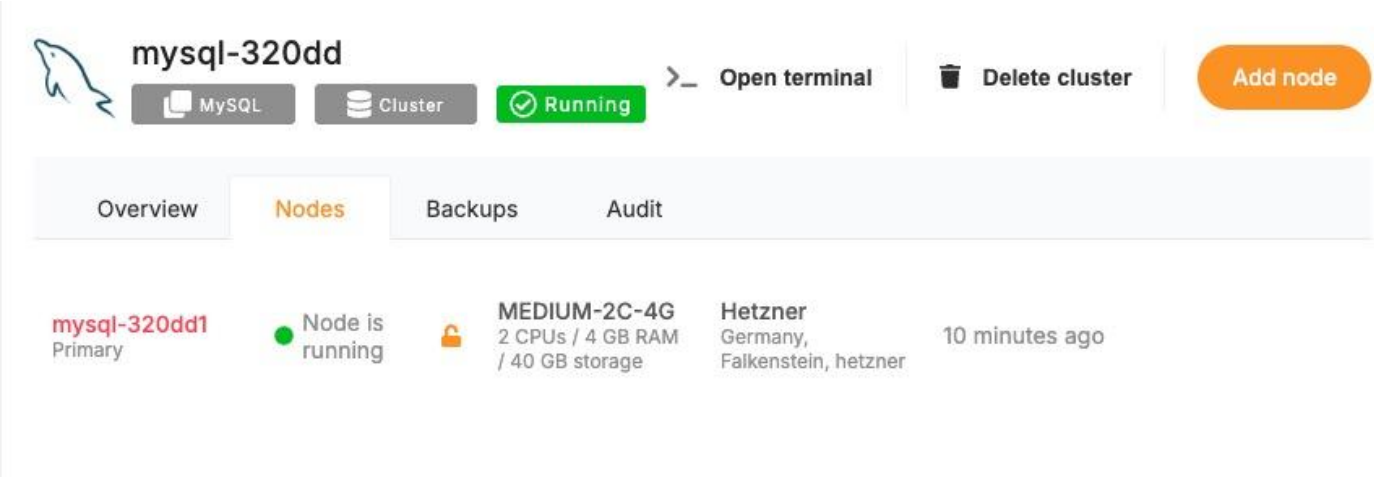
Need to Add a Node

There are several scenarios where adding a node becomes necessary. One of the most common cases is **resource saturation** when existing nodes are fully utilized in terms of CPU, memory, or disk. Adding another node helps distribute the workload and maintain performance under load.

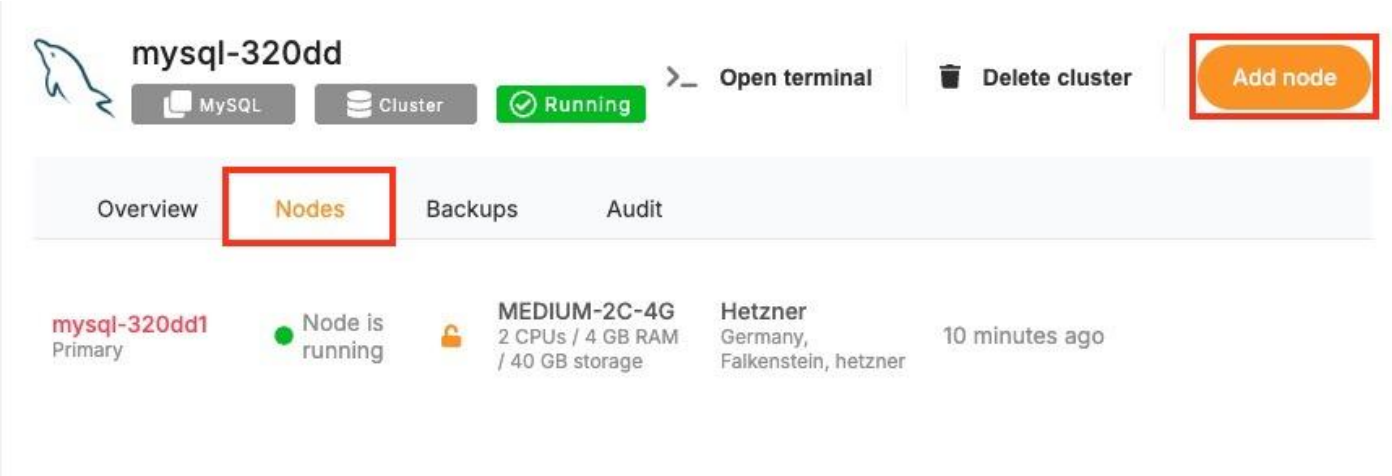
In clusters that run **stateful services** or require **high availability**, having additional nodes ensures that workloads can fail over without downtime. Even in development environments, nodes can be added to isolate environments or test services under production-like load conditions. Scaling out also gives you flexibility when deploying services with different resource profiles or placement requirements.

Add a Node to Cluster

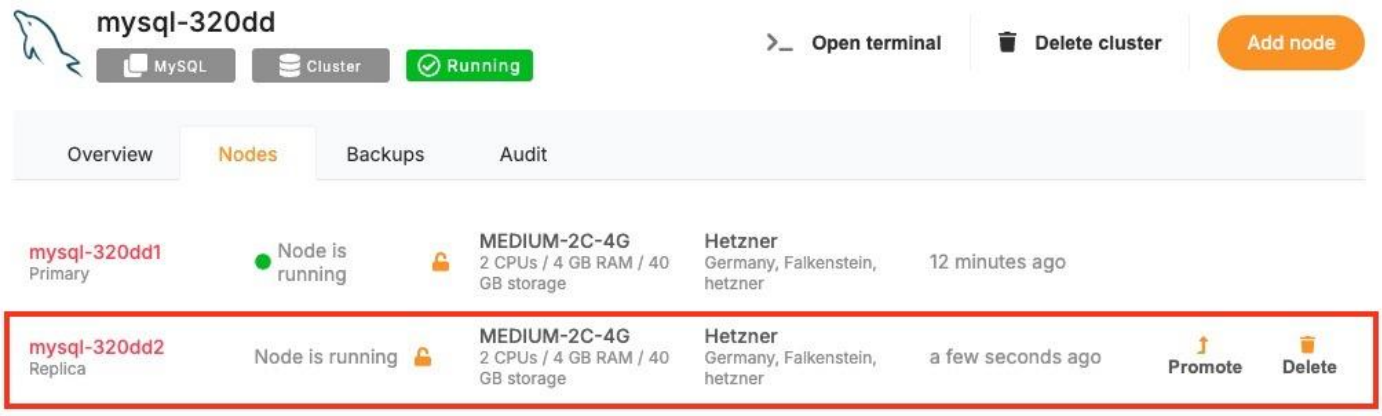
To begin, log in to the Elestio dashboard and navigate to the **Clusters** section from the sidebar. Select the cluster you want to scale. Once inside the cluster view, switch to the **Nodes** tab. This section provides an overview of all current nodes along with their health status and real-time resource usage.



To add a new node, click the **“Add Node”** button. This opens a configuration panel where you can define the specifications for the new node. You’ll be asked to specify the amount of **CPU**, **memory**, and **disk** you want to allocate. If you’re using a bring-your-own-cloud setup, you may also need to confirm or choose the cloud provider and deployment region.



After configuring the node, review the settings to ensure they meet your performance and cost requirements. Click **“Create”** to initiate provisioning. Elestio will begin setting up the new node, and once it’s ready, it will automatically join your cluster.



Once provisioned, the new node will appear in the node list with its own metrics and status indicators. You can monitor its activity, verify that workloads are being scheduled to it, and access its logs directly from the dashboard. From this point onward, the node behaves like any other in the cluster and can be managed using the same lifecycle actions such as rebooting or draining.

Post-Provisioning Considerations

After the node has been added, it becomes part of the active cluster and is available for scheduling workloads. Elestio’s orchestration layer will begin using it automatically, but you can further customize service placement through resource constraints or affinity rules if needed.

For performance monitoring, the dashboard provides per-node metrics, including CPU load, memory usage, and disk I/O. This visibility helps you confirm that the new node is functioning correctly and contributing to workload distribution as expected.

Maintenance actions such as draining or rebooting the node are also available from the same interface, making it easy to manage the node lifecycle after provisioning.

Promoting a Node

Clusters can be designed for high availability or role-based workloads, where certain nodes may take on leadership or coordination responsibilities. In these scenarios, promoting a node is a key administrative task. It allows you to change the role of a node. While not always needed in basic setups, node promotion becomes essential in distributed systems, replicated databases, or services requiring failover control.

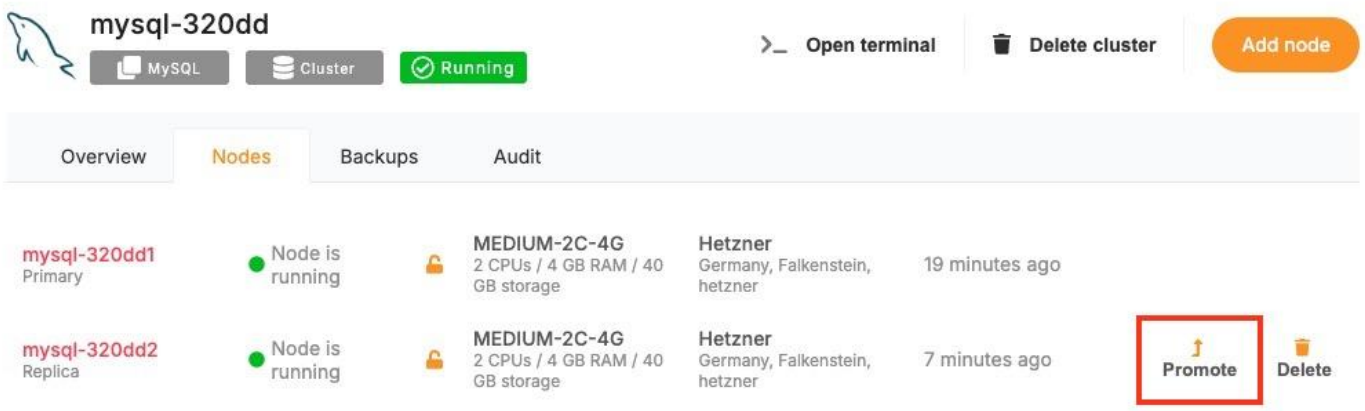
When to Promote a Node?

Promoting a node is typically performed in clusters where role-based architecture is used. In high-availability setups, some nodes may act as leaders while others serve as followers or replicas. If a leader node becomes unavailable or needs to be replaced, you can promote another node to take over its responsibilities and maintain continuity of service.

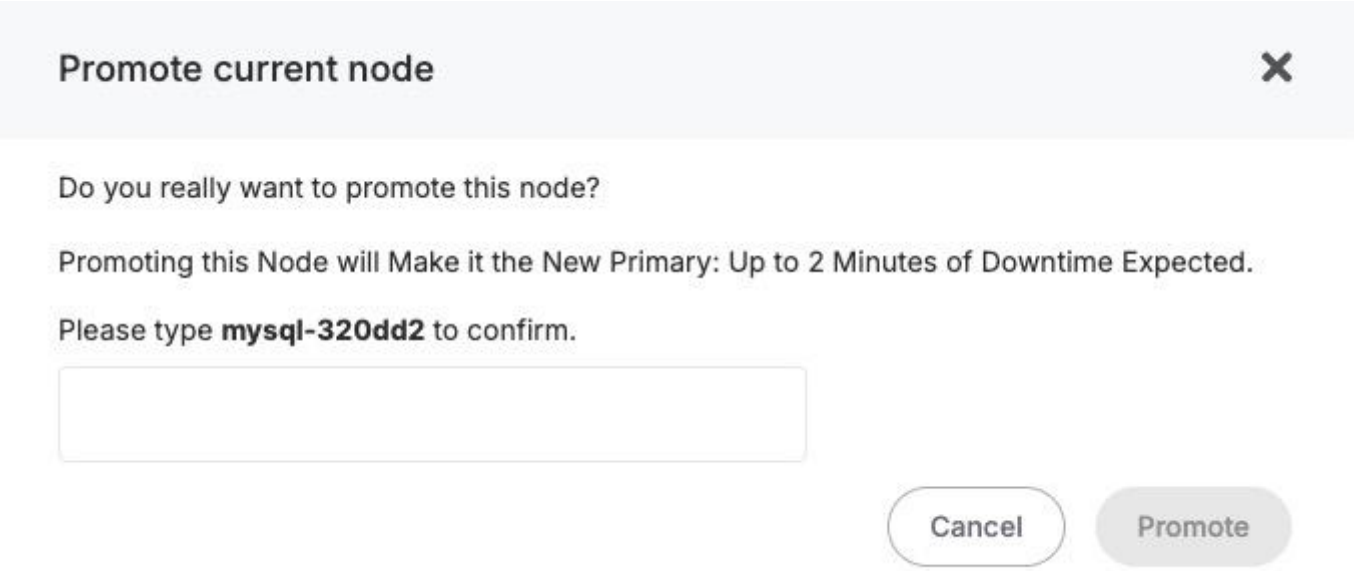
Node promotion is also useful when scaling out and rebalancing responsibilities across a larger cluster. For example, promoting a node to handle scheduling, state tracking, or replication leadership can reduce bottlenecks and improve responsiveness. In cases involving database clusters or consensus-driven systems, promotion ensures a clear and controlled transition of leadership without relying solely on automatic failover mechanisms.

Promote a Node in Elestio

To promote a node, start by accessing the **Clusters** section in the Elestio dashboard. Choose the cluster containing the node you want to promote. Inside the cluster view, navigate to the **Nodes** tab to see the full list of nodes, including their current roles, health status, and resource usage. Locate the node that you want to promote and open its action menu. From here, select the **“Promote Node”** option.



You may be prompted to confirm the action, depending on the configuration and current role of the node. This confirmation helps prevent unintended role changes that could affect cluster behavior.



Once confirmed, Elestio will initiate the promotion process. This involves reconfiguring the cluster's internal coordination state to acknowledge the new role of the promoted node. Depending on the service architecture and the software running on the cluster, this may involve reassigning leadership, updating replication targets, or shifting service orchestration responsibilities.

After promotion is complete, the node's updated role will be reflected in the dashboard. At this point, it will begin operating with the responsibilities assigned to its new status. You can monitor its activity, inspect logs, and validate that workloads are being handled as expected.

Considerations for Promotion

Before promoting a node, ensure that it meets the necessary resource requirements and is in a stable, healthy state. Promoting a node that is under high load or experiencing performance issues

can lead to service degradation. It’s also important to consider replication and data synchronization, especially in clusters where stateful components like databases are in use.

Promotion is a safe and reversible operation, but it should be done with awareness of your workload architecture. If your system relies on specific leader election mechanisms, promoting a node should follow the design patterns supported by those systems.

Removing a Node

Over time, infrastructure needs change. You may scale down a cluster after peak load, decommission outdated resources, or remove a node that is no longer needed for cost, isolation, or maintenance reasons. Removing a node from a cluster is a safe and structured process designed to avoid disruption. The dashboard provides an accessible interface for performing this task while preserving workload stability.

Why Remove a Node?

Node removal is typically part of resource optimization or cluster reconfiguration. You might remove a node when reducing costs in a staging environment, when redistributing workloads across fewer or more efficient machines, or when phasing out a node for maintenance or retirement.

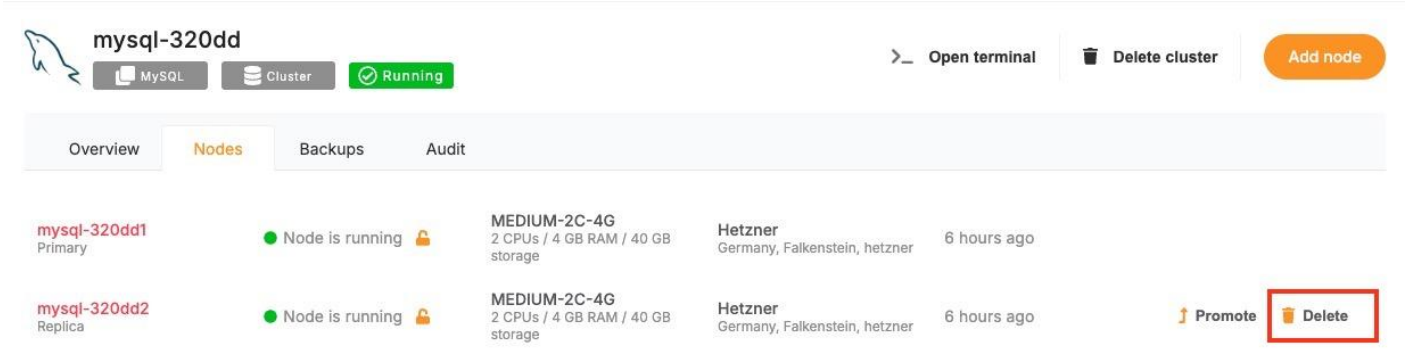
Another common scenario is infrastructure rebalancing, where workloads are shifted to newer nodes with better specs or different regions. Removing an idle or underutilized node can simplify management and reduce noise in your monitoring stack. It also improves scheduling efficiency by removing unneeded targets from the orchestration engine.

In high-availability clusters, node removal may be preceded by data migration or role reassignment (such as promoting a replica). Proper planning helps maintain system health while reducing reliance on unnecessary compute resources.

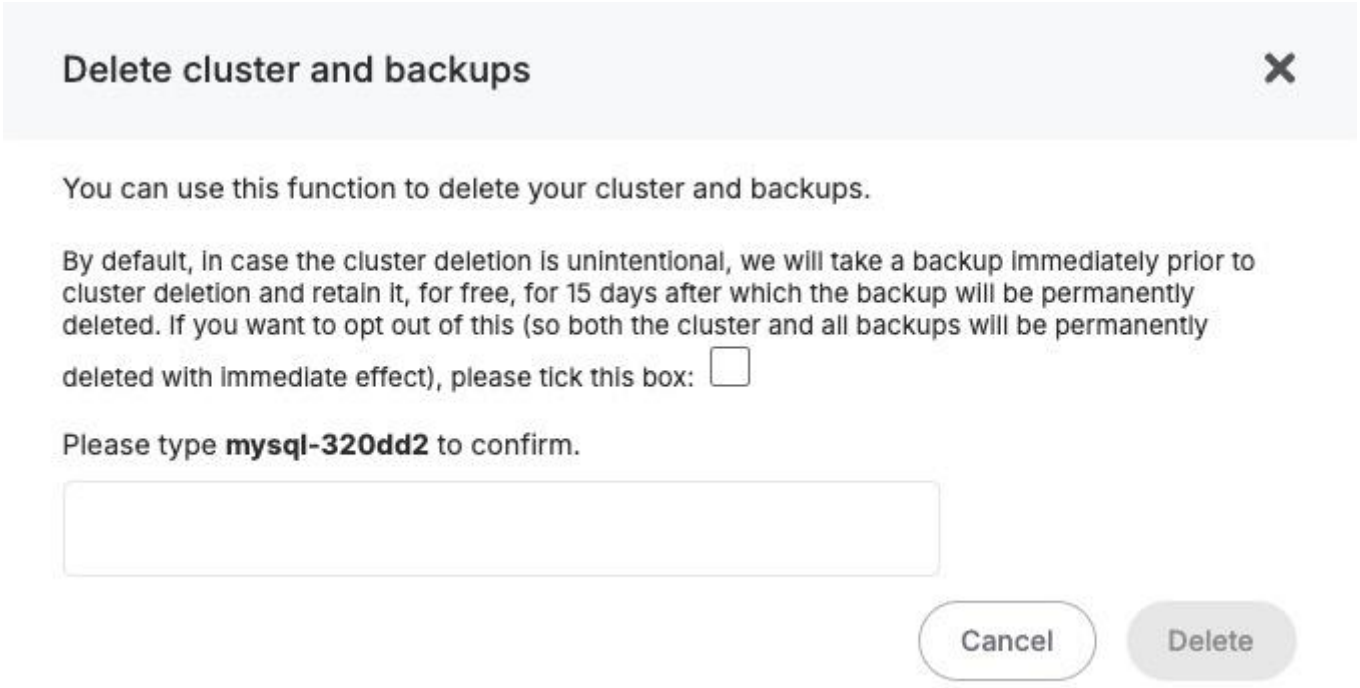
Remove a Node

To begin the removal process, open the Elestio dashboard and navigate to the **Clusters** section. Select the cluster that contains the node you want to remove. From within the cluster view, open the **Nodes** tab to access the list of active nodes and their statuses.

Find the node you want to delete from the list. If the node is currently running services, ensure that those workloads can be safely rescheduled to other nodes or are no longer needed. Since Elestio does not have a built-in drain option, any workload redistribution needs to be handled manually, either by adjusting deployments or verifying that redundant nodes are available. Once the node is drained and idle, open the action menu for that node and select **“Delete Node”**.



The dashboard may prompt you to confirm the operation. After confirmation, Elestio will begin the decommissioning process. This includes detaching the node from the cluster, cleaning up any residual state, and terminating the associated virtual machine.



Once the operation completes, the node will no longer appear in the cluster’s node list, and its resources will be released.

Considerations for Safe Node Removal

Before removing a node in Elestio, it’s important to review the services and workloads currently running on that node. Since Elestio does not automatically redistribute or migrate workloads during

node removal, you should ensure that critical services are either no longer in use or can be manually rescheduled to other nodes in the cluster. This is particularly important in multi-node environments running stateful applications, databases, or services with specific affinity rules.

You should also verify that your cluster will have sufficient capacity after the node is removed. If the deleted node was handling a significant portion of traffic or compute load, removing it without replacement may lead to performance degradation or service interruption. In high-availability clusters, ensure that quorum-based components or replicas are not depending on the node targeted for deletion. Additionally, confirm that the node is not playing a special role such as holding primary data or acting as a manually promoted leader before removal. If necessary, reconfigure or promote another node prior to deletion to maintain cluster integrity.

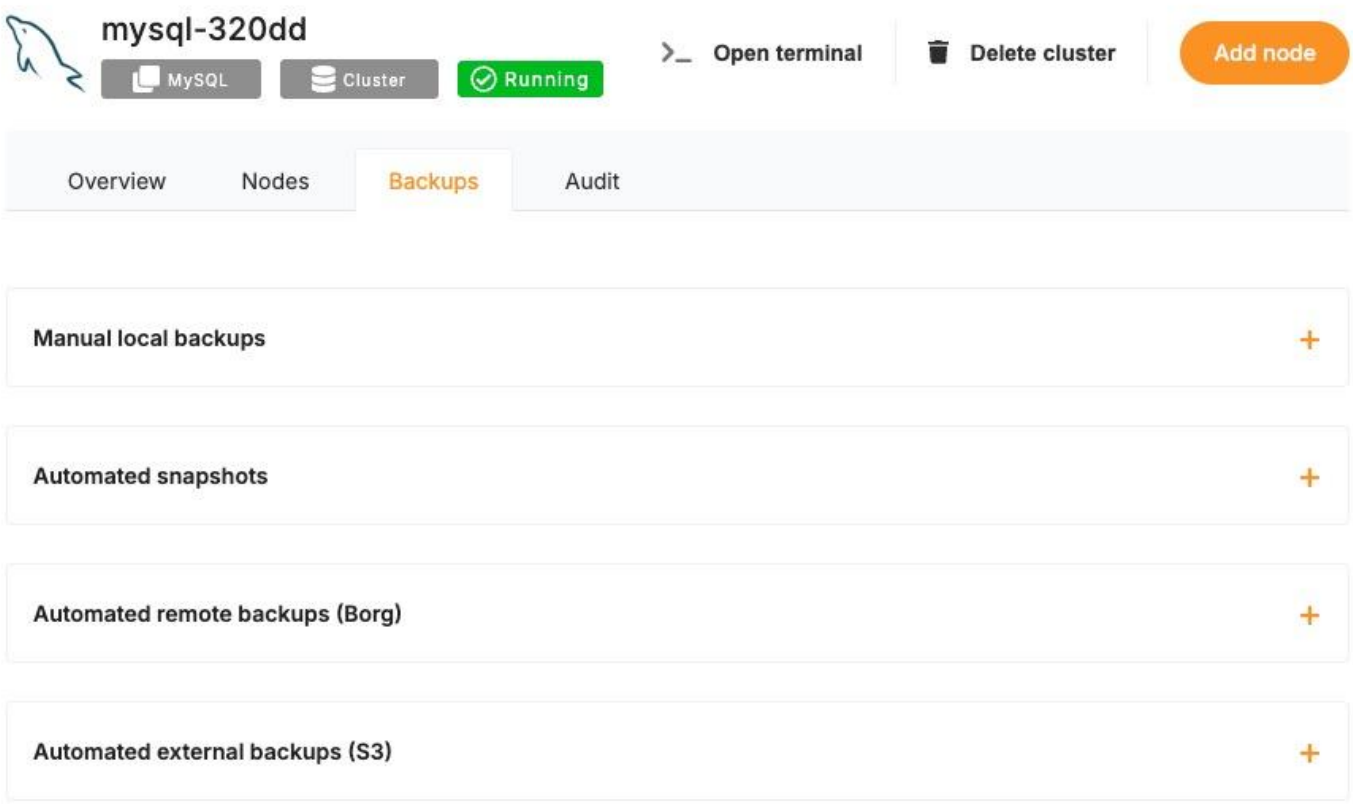
Backups and Restores

Reliable backups are essential for data resilience, recovery, and business continuity. Elestio provides built-in support for managing backups across all supported services, ensuring that your data is protected against accidental loss, corruption, or infrastructure failure. The platform includes an automated backup system with configurable retention policies and a straightforward restore process, all accessible from the dashboard. Whether you're operating a production database or a test environment, understanding how backups and restores work in Elestio is critical for maintaining service reliability.

Cluster Backups

Elestio provides multiple backup mechanisms designed to support various recovery and compliance needs. Backups are created automatically for most supported services, with consistent intervals and secure storage in managed infrastructure. These backups are performed in the background to ensure minimal performance impact and no downtime during the snapshot process. Each backup is timestamped, versioned, and stored securely with encryption. You can access your full backup history for any given service through the dashboard and select any version for restoration.

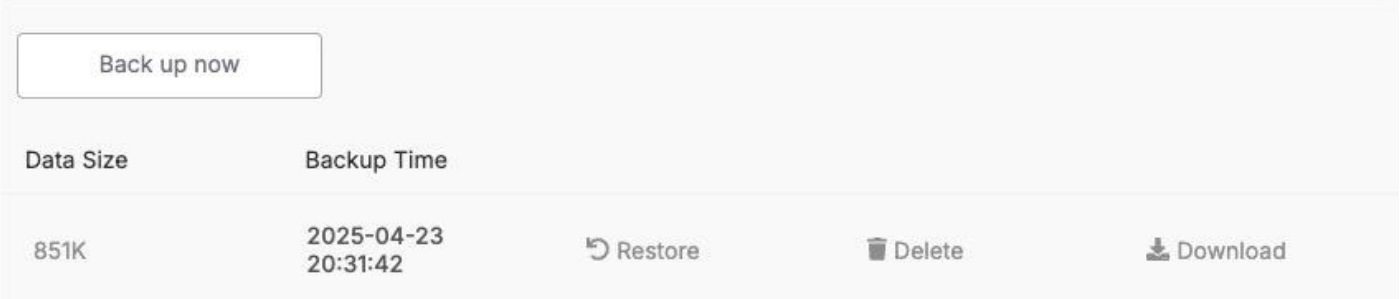
You can utilize different backup options depending on your preferences and operational requirements. Elestio supports **manual local backups** for on-demand recovery points, **automated snapshots** that capture the state of the service at fixed intervals, and **automated remote backups using Borg**, which securely stores backups on external storage volumes managed by Elestio. In addition, you can configure **automated external backups to S3-compatible storage**, allowing you to maintain full control over long-term retention and geographic storage preferences.



Restoring from a Backup

Restoring a backup in Elestio is a user-initiated operation, available directly from the service dashboard. Once you're in the dashboard, select the service you'd like to restore. Navigate to the **Backups** section, where you'll find a list of all available backups along with their creation timestamps.

To initiate a restore, choose the desired backup version and click on the **"Restore"** option. You will be prompted to confirm the operation. Depending on the type of service, the restore can either overwrite the current state or recreate the service as a new instance from the selected backup.



The restore process takes a few minutes, depending on the size of the backup and the service type. Once completed, the restored service is immediately accessible. In the case of databases, you can validate the restore by connecting to the database and inspecting the restored data.

Considerations for Backup & Restore

- Before restoring a backup, it’s important to understand the impact on your current data. Restores may **overwrite existing service state**, so if you need to preserve the current environment, consider creating a manual backup before initiating the restore. In critical environments, restoring to a new instance and validating the data before replacing the original is a safer approach.
- Keep in mind that restore operations are not instantaneous and may temporarily affect service availability. It’s best to plan restores during maintenance windows or periods of low traffic, especially in production environments.
- For services with high-frequency data changes, be aware of the backup schedule and retention policy. Elestio’s default intervals may not capture every change, so for high-volume databases, consider exporting incremental backups manually or using continuous replication where supported.

Monitoring Backup Health

Elestio provides visibility into your backup history directly through the dashboard. You can monitor the **status**, **timestamps**, and **success/failure** of backup jobs. In case of errors or failed backups, the dashboard will display alerts, allowing you to take corrective actions or contact support if necessary.

It’s good practice to periodically verify that backups are being generated and that restore points are recent and complete. This ensures you’re prepared for unexpected failures and that recovery options remain reliable.

Cluster Resynchronization

In distributed systems, consistency and synchronization between nodes are critical to ensure that services behave reliably and that data remains accurate across the cluster. Elestio provides built-in mechanisms to detect and resolve inconsistencies across nodes using a feature called **Cluster Resynchronization**. This functionality ensures that node-level configurations, data replication, and service states are properly aligned, especially after issues like node recovery, temporary network splits, or service restarts.

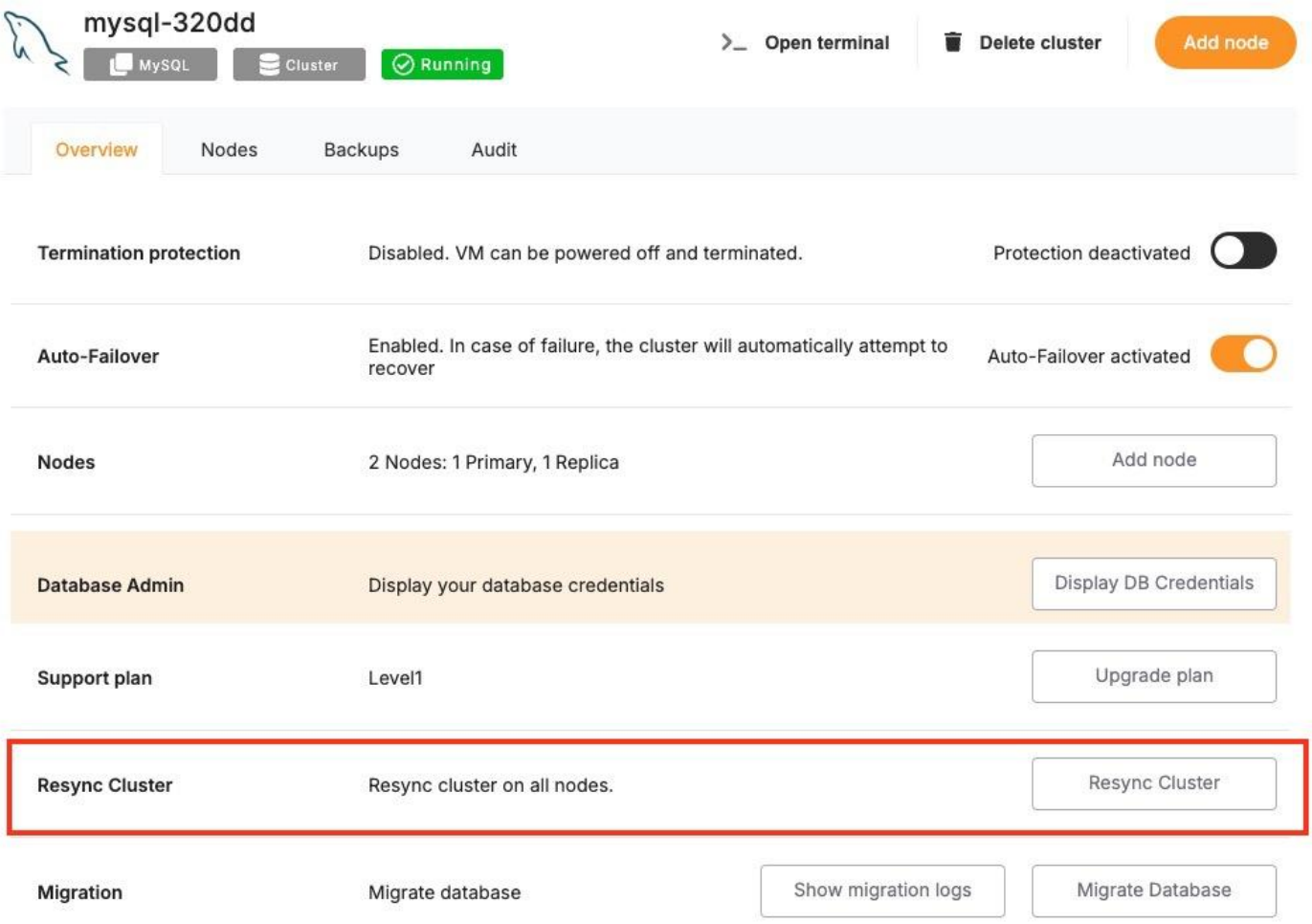
Need for Cluster Resynchronization

Resynchronization is typically required when secondary nodes in a cluster are no longer consistent with the primary node. This can happen due to temporary network failures, node restarts, replication lag, or partial service interruptions. In such cases, secondary nodes may fall behind or store incomplete datasets, which could lead to incorrect behavior if a failover occurs or if read operations are directed to those nodes. Unresolved inconsistencies can result in data divergence, serving outdated content, or failing health checks in load-balanced environments. Performing a resynchronization ensures that all secondary nodes are forcibly aligned with the current state of the primary node, restoring a clean and unified cluster state.

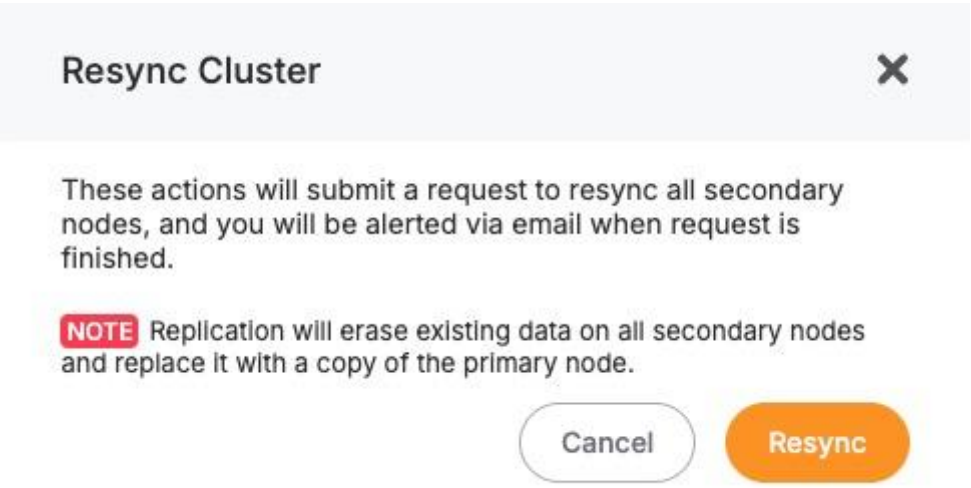
It may also be necessary to perform a resync after restoring a service from backup, during infrastructure migrations, or after recovering a previously offline node. In each of these cases, resynchronization acts as a corrective mechanism to ensure that every node is operating with the same configuration and dataset, reducing the risk of drift and maintaining data integrity across the cluster.

Cluster Resynchronization

To perform a resynchronization, start by accessing the Elestio dashboard and navigating to the **Clusters** section. Select the cluster where synchronization is needed. On the **Cluster Overview** page, scroll down slightly until you find the **“Resync Cluster”** option. This option is visible as part of the cluster controls and is available only in clusters with multiple nodes and a defined primary node.



Clicking the **Resync** button opens a confirmation dialog. The message clearly explains that this action will initiate a request to resynchronize **all secondary nodes**. During the resync process, **existing data on all secondary nodes will be erased and replaced with a copy of the data from the primary node**. This operation ensures full consistency across the cluster but should be executed with caution, especially if recent changes exist on any of the secondaries that haven't yet been replicated.



You will receive an email notification once the resynchronization is complete. During this process, Elestio manages the replication safely, but depending on the size of the data, the operation may take a few minutes. It’s advised to avoid making further changes to the cluster while the resync is in progress.

Considerations Before Resynchronizing

- Before triggering a resync, it’s important to verify that the primary node holds the desired state and that the secondary nodes do not contain any critical unsynced data. Since the resync **overwrites** the secondary nodes completely, any local changes on those nodes will be lost.
- This action is best used when you’re confident that the primary node is healthy, current, and stable. Avoid initiating a resync if the primary has recently experienced errors or data issues. Additionally, consider performing this operation during a low-traffic period, as synchronization may temporarily impact performance depending on the data volume.
- If your application requires high consistency guarantees, it’s recommended to monitor your cluster closely during and after the resync to confirm that services are functioning correctly and that the replication process completed successfully.

Database Migrations

When managing production-grade services, the ability to perform reliable and repeatable database migrations is critical. Whether you're applying schema changes, updating seed data, or managing version-controlled transitions, Elestio provides a built-in mechanism to execute migrations safely from the dashboard. This functionality is especially relevant when running containerized database services like PostgreSQL, MySQL, or similar within a managed cluster.

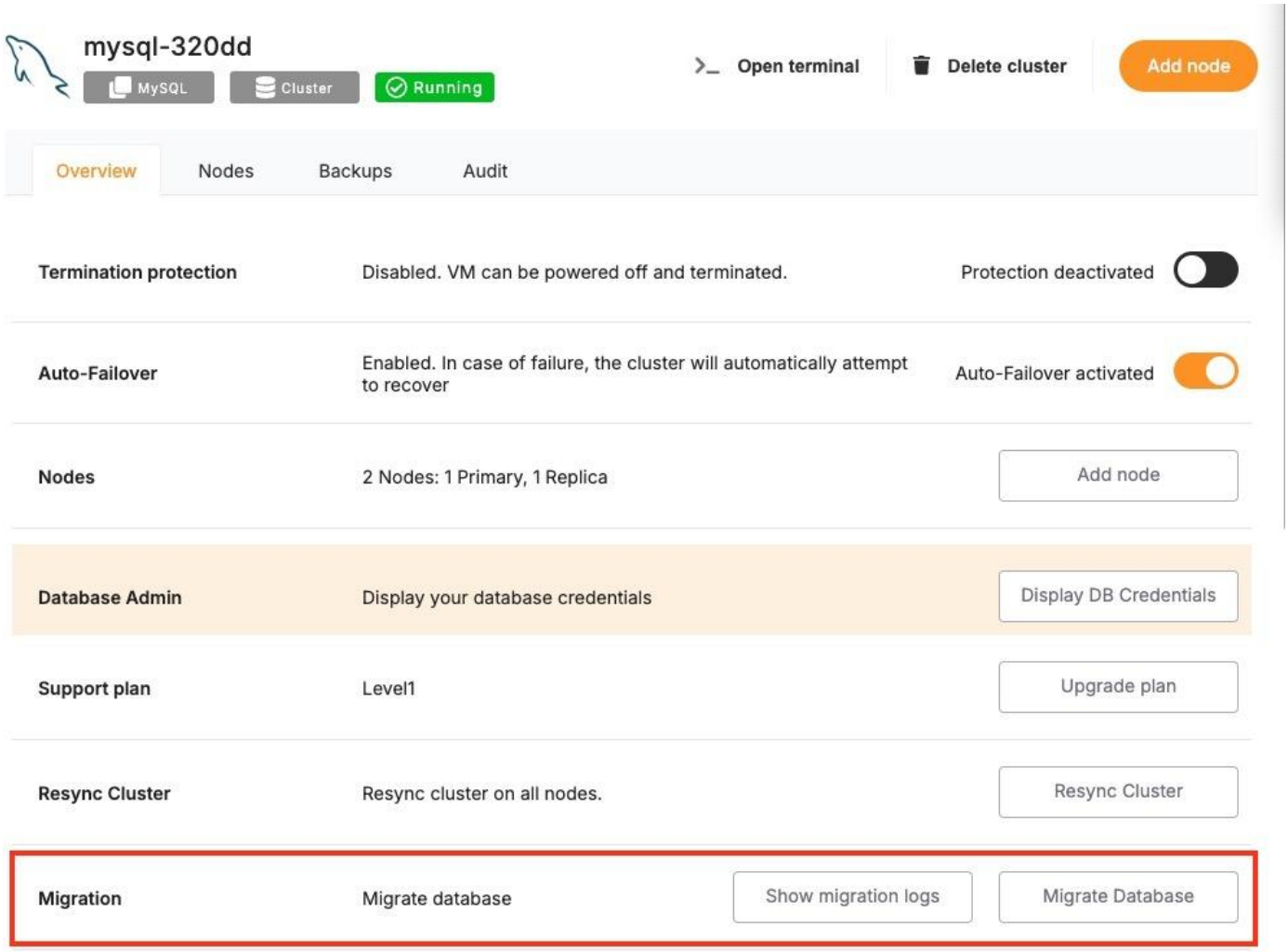
Need for Migrations

Database migrations are commonly required when updating your application's data model or deploying new features. Schema updates such as adding columns, modifying data types, creating indexes, or introducing new tables need to be synchronized with the deployment lifecycle of your application code.

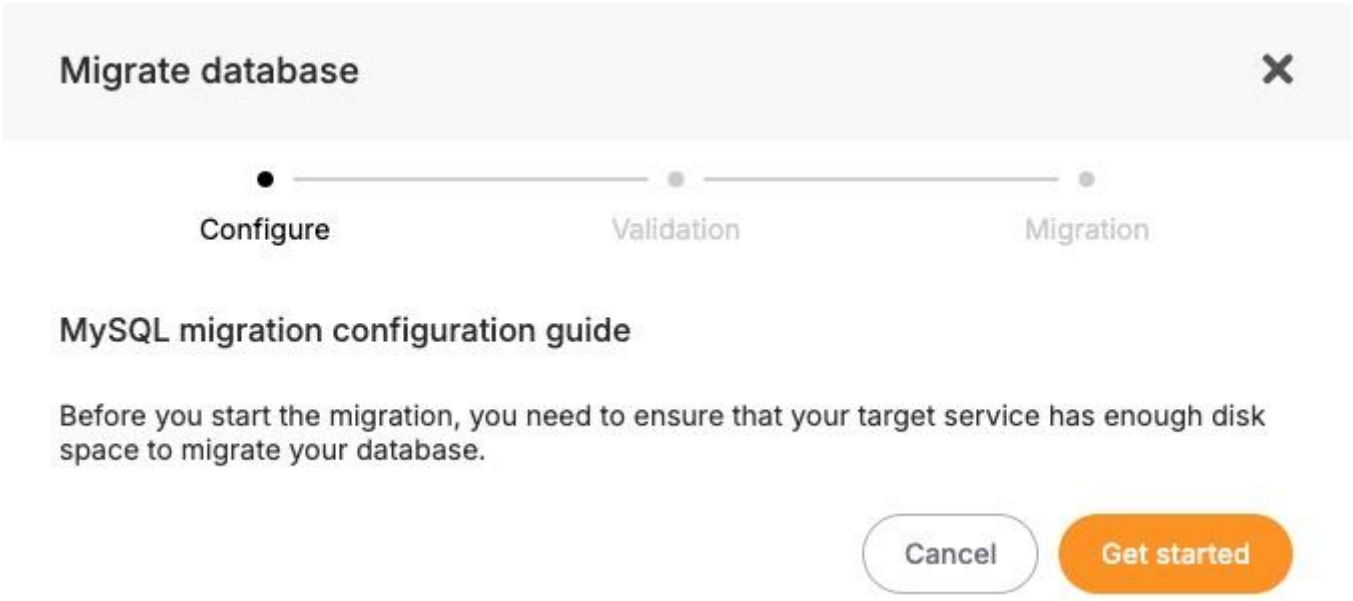
Migrations may also be needed during version upgrades to introduce structural or configuration changes required by newer database engine versions. In some cases, teams use migrations to apply baseline datasets, adjust permissions, or clean up legacy objects. Running these changes through a controlled migration system ensures consistency across environments and helps avoid untracked manual changes.

Running Database Migration

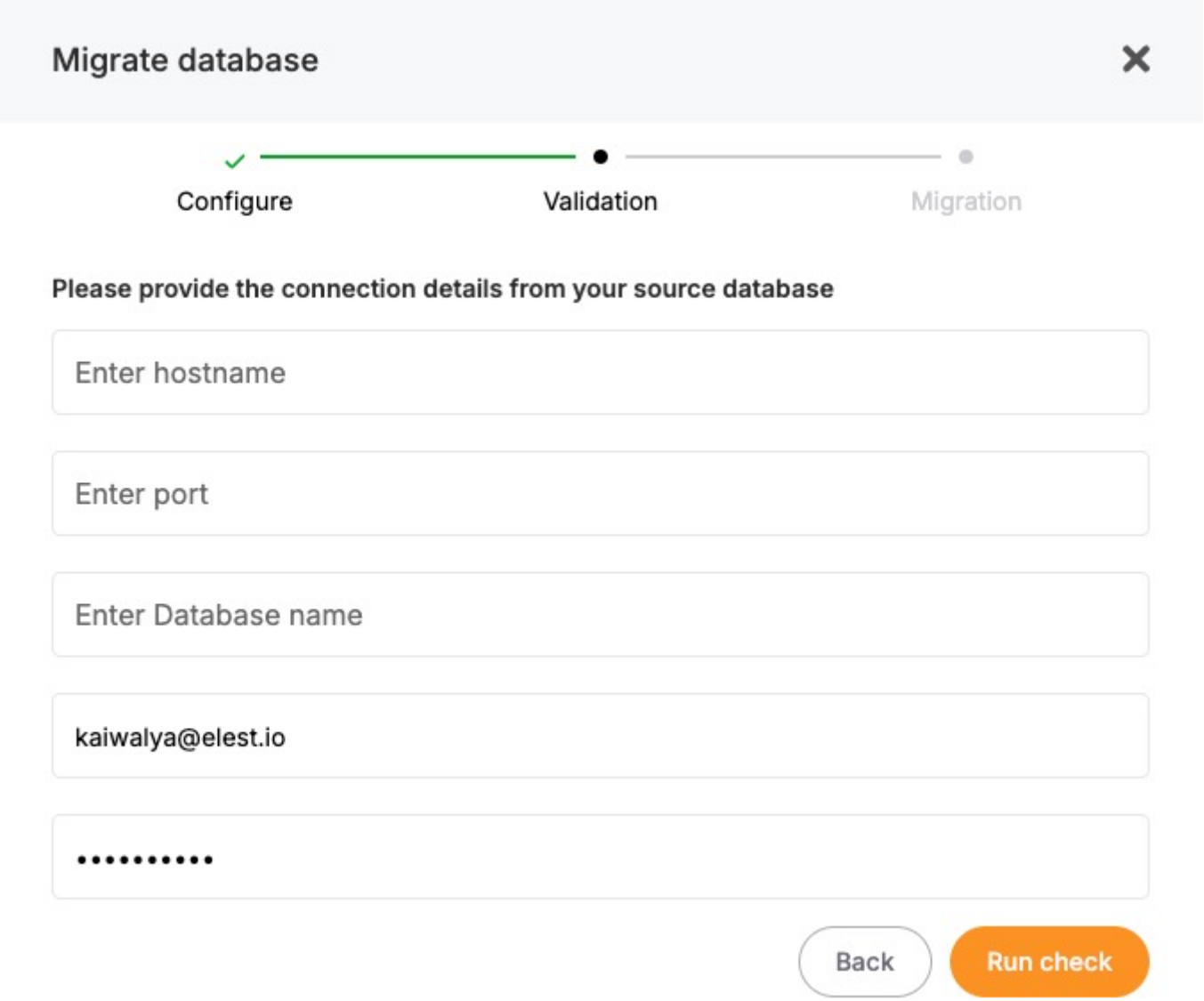
To run a database migration in Elestio, start by logging into the Elestio dashboard and navigating to the **Clusters** section. Select the cluster that contains the target database service. From the **Cluster Overview** page, scroll down until you find the **"Migration"** option.



Clicking this option will open the migration workflow, which follows a **three-step process**: **Configure**, **Validation**, and **Migration**. In the **Configure** step, Elestio provides a migration configuration guide specific to the database type, such as MySQL. At this point, you must ensure that your target service has sufficient **disk space** to complete the migration. If there is not enough storage available, the migration may fail midway, so it’s strongly recommended to review storage utilization beforehand.



Once configuration prerequisites are met, you can proceed to the **Validation** step. Elestio will check the secondary database details you have provided for the migration.



If the validation passes, the final **Migration** step will become active. You can then initiate the migration process. Elestio will handle the actual data transfer, schema replication, and state synchronization internally. The progress is tracked, and once completed, the migrated database will be fully operational on the target service.

Considerations Before Running Migrations

- Before running any migration, it’s important to validate the script or changes in a staging environment. Since migrations may involve irreversible changes—such as dropping columns, altering constraints, or modifying data—careful review and version control are

essential.

- In production environments, plan migrations during maintenance windows or low-traffic periods to minimize the impact of any schema locks or temporary unavailability. If you’re using replication or high-availability setups, confirm that the migration is compatible with your architecture and will not disrupt synchronization between primary and secondary nodes.
- You should also ensure that proper backups are in place before applying structural changes. In Elestio, the backup feature can be used to create a restore point that allows rollback in case the migration introduces issues.

Deleting a Cluster

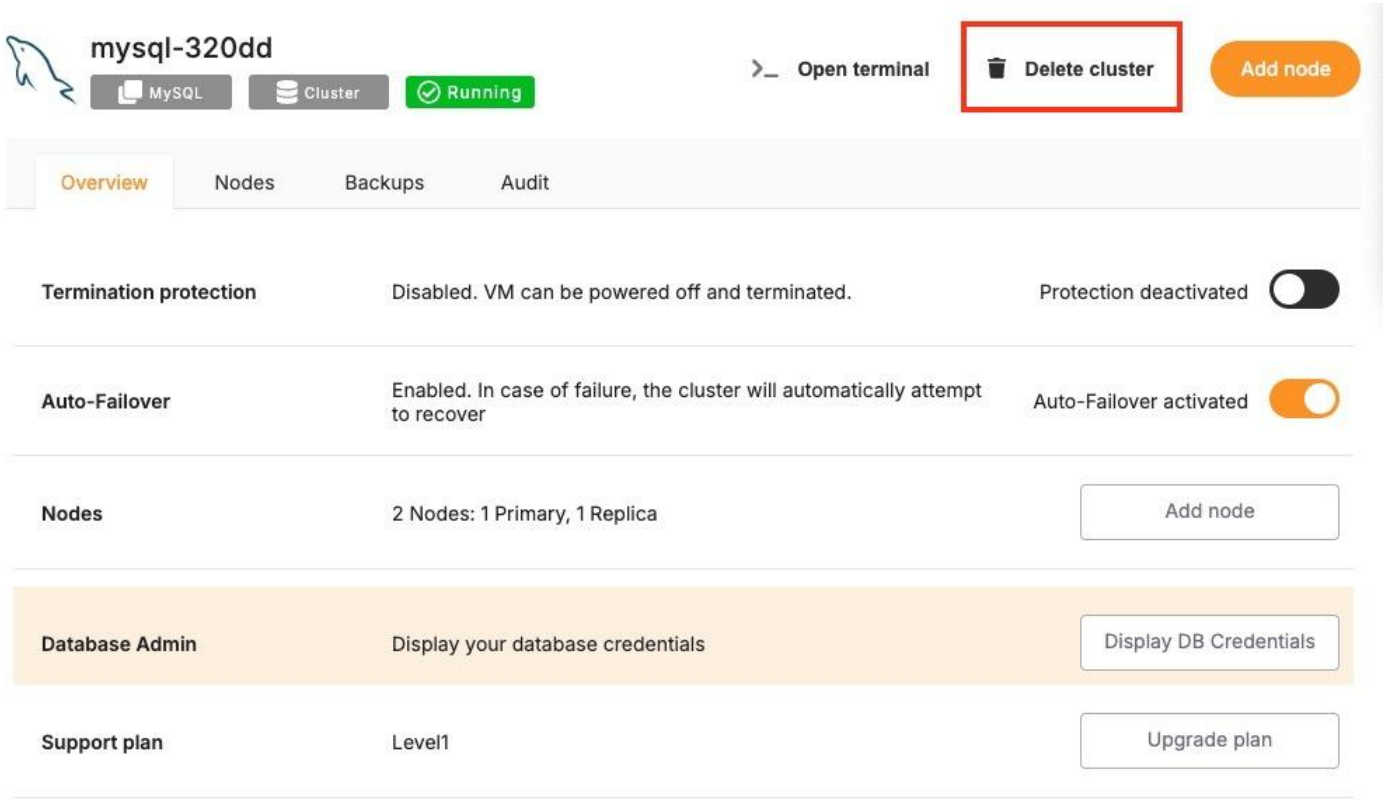
When a cluster is no longer needed—whether it was created for testing, staging, or an obsolete workload—deleting it helps free up resources and maintain a clean infrastructure footprint. Elestio provides a straightforward and secure way to delete entire clusters directly from the dashboard. This action permanently removes the associated services, data, and compute resources tied to the cluster.

When to Delete a Cluster

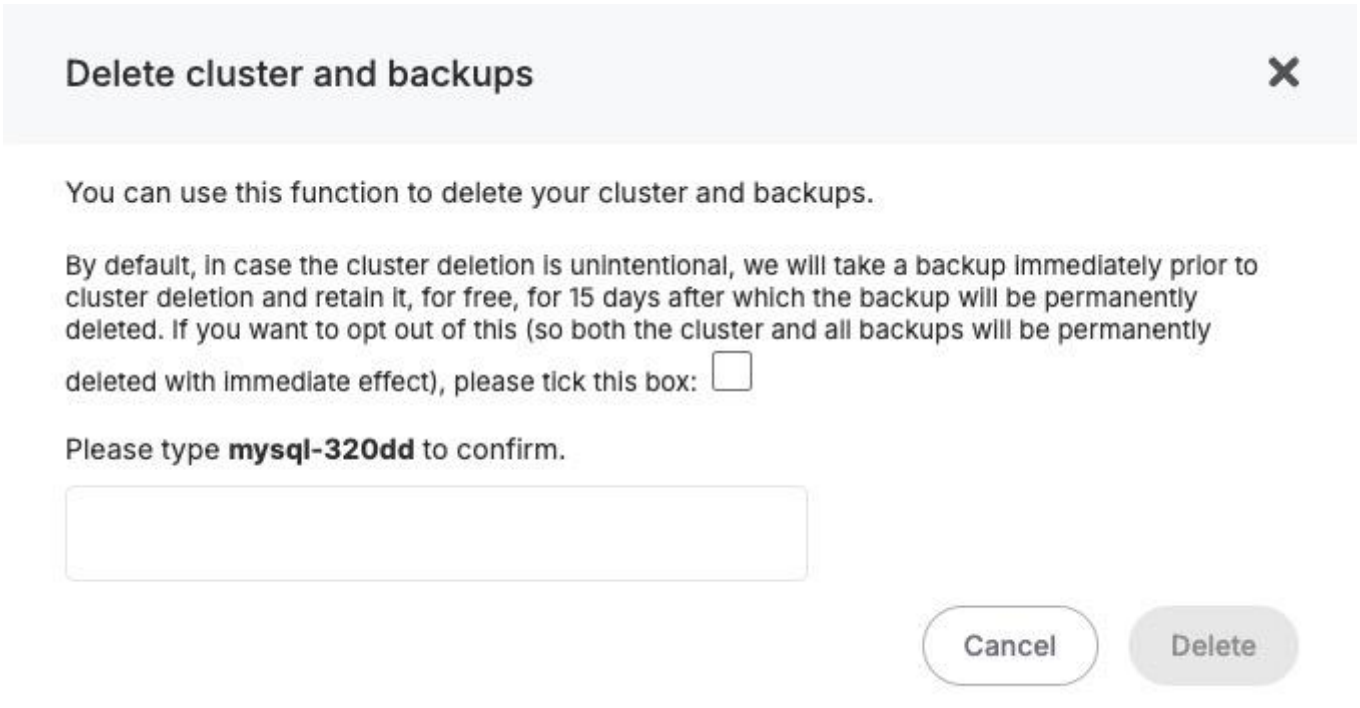
Deleting a cluster is a final step often performed when decommissioning an environment. This could include shutting down a test setup, replacing infrastructure during migration, or retiring an unused production instance. In some cases, users also delete and recreate clusters as part of major version upgrades or architectural changes. It is essential to confirm that all data and services tied to the cluster are no longer required or have been backed up or migrated before proceeding. Since cluster deletion is irreversible, any services, volumes, and backups associated with the cluster will be permanently removed.

Delete a Cluster

To delete a cluster, log in to the Elestio dashboard and navigate to the **Clusters** section. From the list of clusters, select the one you want to remove. Inside the selected cluster, you'll find a **navigation bar** at the top of the page. One of the available options in this navigation bar is **"Delete Cluster."**



Clicking this opens a confirmation dialog that outlines the impact of deletion. It will clearly state that deleting the cluster will **permanently remove** all associated services, storage, and configurations. By acknowledging a warning or typing in the cluster name, depending on the service type. Once confirmed, Elestio will initiate the deletion process, which includes tearing down all resources associated with the cluster. This typically completes within a few minutes, after which the cluster will no longer appear in your dashboard.



Considerations Before Deleting

Deleting a cluster also terminates any linked domains, volumes, monitoring configurations, and scheduled backups. These cannot be recovered once deletion is complete, so plan accordingly before confirming the action. If the cluster was used for production workloads, consider archiving data to external storage (e.g., S3) or exporting final snapshots for compliance and recovery purposes.

Before deleting a cluster, verify that:

- All required data has been backed up externally (e.g., downloaded dumps or exports).
- Any active services or dependencies tied to the cluster have been reconfigured or shut down.
- Access credentials, logs, or stored configuration settings have been retrieved if needed for auditing or migration.

Restricting Access by IP

Securing access to services is a fundamental part of managing cloud infrastructure. One of the most effective ways to reduce unauthorized access is by restricting connectivity to a defined set of IP addresses. Elestio supports IP-based access control through its dashboard, allowing you to explicitly define which IPs or IP ranges are allowed to interact with your services. This is particularly useful when exposing databases, APIs, or web services over public endpoints.

Need to Restrict Access by IP

Restricting access by IP provides a first layer of network-level protection. Instead of relying solely on application-layer authentication, you can control who is allowed to even initiate a connection to your service. This approach reduces the surface area for attacks such as brute-force login attempts, automated scanning, or unauthorized probing.

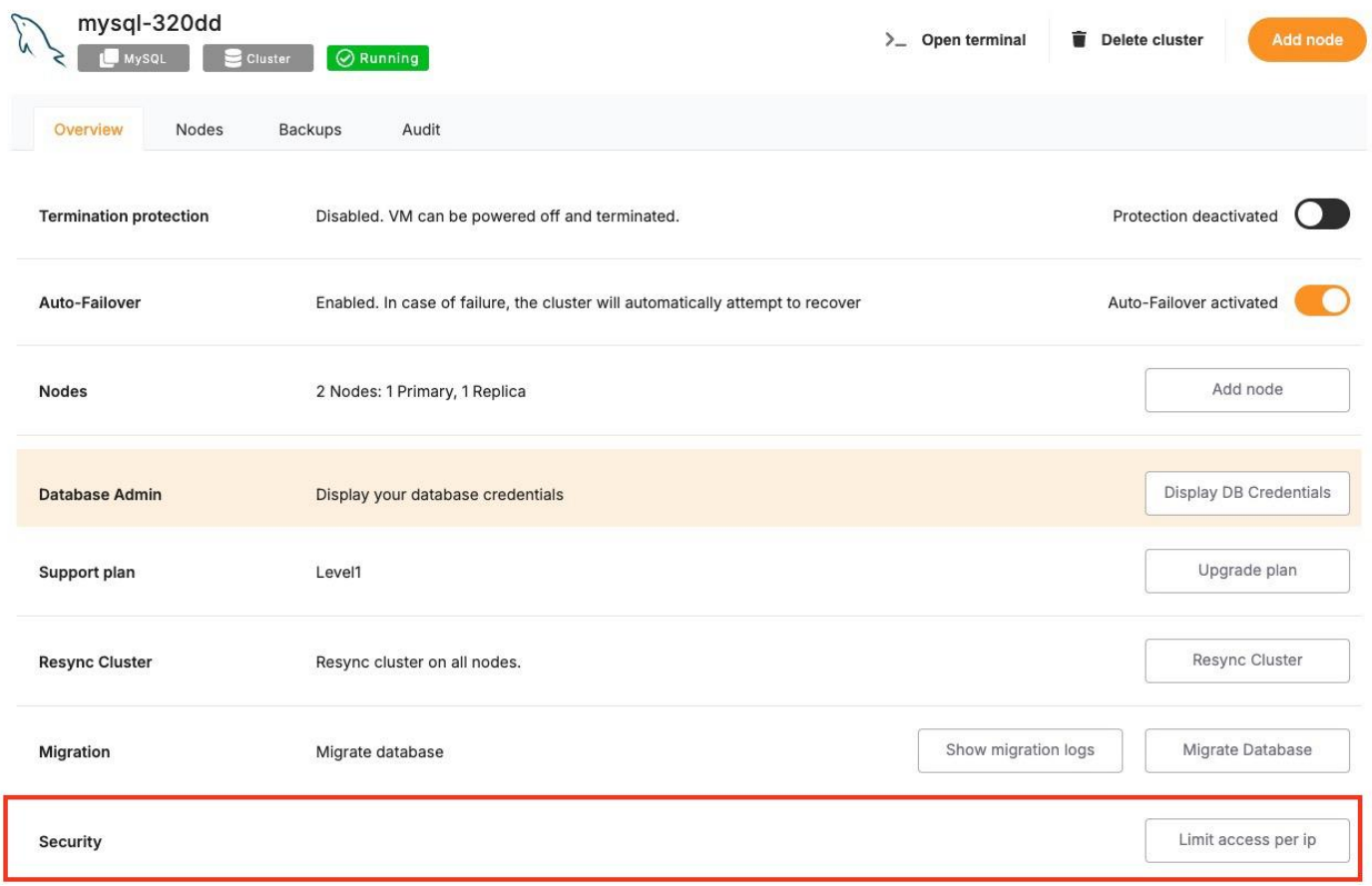
Common use cases include:

- Limiting access to production databases from known office networks or VPNs.
- Allowing only CI/CD pipelines or monitoring tools with static IPs to connect.
- Restricting admin dashboards or internal tools to internal teams.

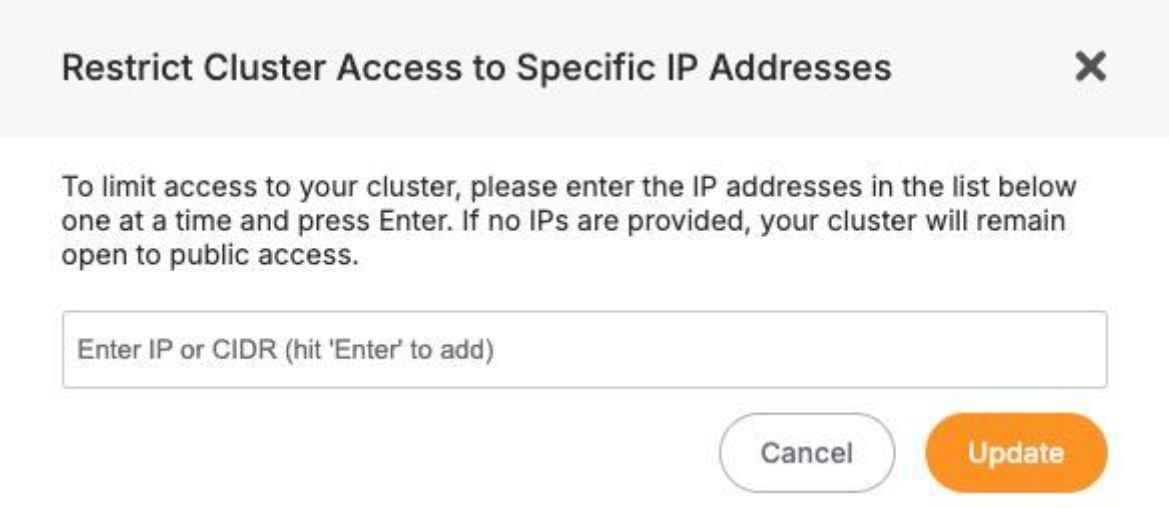
By defining access rules at the infrastructure level, you gain more control over who can reach your services, regardless of their authentication or API access status.

Restrict Access by IP

To restrict access by IP in Elestio, start by logging into the Elestio dashboard and navigating to the **Clusters** section. Select the cluster that hosts the service you want to protect. Once inside the **Cluster Overview** page, locate the **Security** section.



Within this section, you'll find a setting labeled **"Limit access per IP"**. This is where you can define which IP addresses or CIDR ranges are permitted to access the services running in the cluster. You can add a specific IPv4 or IPv6 address (e.g., 203.0.113.5) or a subnet in CIDR notation (e.g., 203.0.113.0/24) to allow access from a range of IPs.



After entering the necessary IP addresses, save the configuration. The changes will apply to all services running inside the cluster, and only the defined IPs will be allowed to establish network connections. All other incoming requests from unlisted IPs will be blocked at the infrastructure level.

Considerations When Using IP Restrictions

- When applying IP restrictions, it’s important to avoid locking yourself out. Always double-check that your own IP address is included in the allowlist before applying rules, especially when working on remote infrastructure.
- For users on dynamic IPs (e.g., home broadband connections), consider using a VPN or a static jump host that you can reliably allowlist. Similarly, if your services are accessed through cloud-based tools, make sure to verify their IP ranges and update your rules accordingly when those IPs change.
- In multi-team environments, document and review IP access policies regularly to avoid stale rules or overly permissive configurations. Combine IP restrictions with secure authentication and encrypted connections (such as HTTPS or SSL for databases) for layered security.