

# Connect Node.js with Google Cloud SQL (MySQL) using Password Authentication

---

This blog explains **step-by-step** how to connect a **Node.js application** with **Google Cloud SQL (MySQL)** using the **password-based approach**.

This approach is:

- Easy to understand
- Beginner friendly
- Good for learning & quick integration
- ! Less secure than IAM (we'll cover IAM later)

## What You'll Learn

- What is Cloud SQL
- Why Service Account is still required (even with password)
- How to create Cloud SQL MySQL instance
- How to create database and users
- How to connect Node.js with Cloud SQL using **password authentication**
- How to test the connection locally

## Understanding the Authentication Flow

When using **password-based authentication**, there are **two layers**:

### 1 GCP Authentication (Infrastructure Level)

- Uses **Service Account + key.json**
- Allows your app to reach Cloud SQL securely

### 2 MySQL Authentication (Database Level)

- Uses **MySQL username + password** (example: `root`)
- Controls access inside the database

Even if you use a password, **service account is still required** to access Cloud SQL.

# Step 1: Create Cloud SQL MySQL Instance

1. Open **Google Cloud Console**
2. Navigate to **☰ → Cloud SQL → Instances**
3. Click **Create Instance** → Choose **MySQL**

The screenshot shows the 'Create a MySQL instance' page in the Google Cloud Console. On the left, there's a sidebar with 'Cloud SQL' selected. The main area has a heading 'Choose a Cloud SQL edition' with a note about determining foundational characteristics. Two options are shown: 'Enterprise Plus' and 'Enterprise'. 'Enterprise Plus' is described with features like 99.99% availability SLA, sub-second planned maintenance downtime, and up to 35 days point-in-time recovery window. 'Enterprise' is described with 99.95% availability SLA, less than 60 seconds planned maintenance downtime, and general purpose machines. Below these, a note says 'Choose a preset for this edition. Presets can be customized later as needed.' A dropdown for 'Edition preset' is set to 'Sandbox'. To the right is a 'Summary' table with the following data:

Cloud SQL Edition	Enterprise
Region	us-central1 (Iowa)
DB Version	MySQL 8.4
Machine type	db-custom-2-8192
vCPUs	2 vCPU
RAM	8 GB
Data Cache	Disabled
Storage	10 GB SSD
Connections	Public IP
Backup	Automated
Availability	Single zone
Point-in-time recovery	Enabled
Network throughput (MB/s)	500 of 500
IOPS	Read: 6,300 of 15,000 Write: 6,300 of 15,000
Disk throughput (MB/s)	Read: 4.8 of 240.0 Write: 4.8 of 240.0

Below the summary is a section for 'Pricing estimate (without discounts)' with a note about compute, memory, and storage resources.

## 4. Recommended Settings:

- **Edition:** Enterprise
- **Edition Preset:** Sandbox
- **Instance ID:** `my-db-instance` (or any name)
- **Root Password:** Click *Generate* and save it safely
- **Region:** `us-central1`
- **Zonal Availability:** Single zone

## 5. Uncheck **Prevent instance deletion** (useful for testing)

Go to **Data Protection** Section and Uncheck the **Prevent Instance Deletion**. So that later we can

## delete the Instance

The screenshot shows the 'Create a MySQL instance' page in the Google Cloud Platform. The left sidebar has 'Cloud SQL' selected under 'Instances'. The main form includes fields for 'Days of logs' (set to 7), 'Enhanced backup tier' (radio button selected), 'Instance deletion protection' (checkbox selected), 'Prevent instance deletion' (checkbox unselected), 'Retain backups after instance deletion' (checkbox selected), and 'Final backup on instance deletion' (checkbox selected). To the right, detailed instance metadata is listed:

Cloud SQL	Enterprise
Region	us-central1 (Iowa)
DB Version	MySQL 8.4
Machine type	db-custom-2-8192
vCPUs	2 vCPU
RAM	8 GB
Data Cache	Disabled
Storage	10 GB SSD
Connections	Public IP
Backup	Automated
Availability	Single zone
Point-in-time recovery	Enabled
Network throughput (MB/s)	500 of 500
IOPS	Read: 6,300 of 15,000 Write: 6,300 of 15,000
Disk throughput (MB/s)	Read: 4.8 of 240.0 Write: 4.8 of 240.0

6. Click **Create** and wait 4–5 minutes.

**💡 NOTE :** Don't worry you wont charge anything for Creating Database Instance and Running for 2 Hours, Still If you see charges on billing on Cloud Console, Its Fine you will get 100% discount. Your Instance Will be Created in 4-5 Minutes

## Step 2: Create Database in Cloud SQL Instance

After Instance Created You Will get a Dashboard like this :

The screenshot shows the 'Overview' page for the 'my-db-instance' in the Cloud SQL Instances section. The left sidebar has 'Overview' selected under 'Primary instance'. The main area shows the instance name 'my-db-instance' and MySQL 8.4. Below it, a 'Learn the basics of Cloud SQL' section lists steps: 'Import data', 'Query and explore data', and 'Connect source application'. A note says: 'With your Cloud SQL instance now running, the next step is to import data. You can import CSV and SQL files from a GCS bucket.' At the bottom, there's a chart titled 'CPU utilization' with a timeline from UTC+5:30 to 2:00 PM.

2. 1. Open your Cloud SQL instance

2.2 Go to Databases → Create Database

2.3 Enter database name (example: `my-node-db`)

2.4 Click **Create**

Your DB will Created within 2-3 Minutes

## Step 3: Create Service Account

1. Navigate to **IAM & Admin → Service Accounts**
2. Click **Create Service Account**
3. Enter name (example: `nodejs-app-sa`)
4. Click **Create and Continue**

### Assign Roles

Add these roles:

- **Cloud SQL Admin** (for learning / testing)
- **Cloud SQL Client** (required to connect)

Click **Done**.

Email	Status	Name	Description	Key ID	Key creation date
862105574366-compute@developer.gserviceaccount.com	Enabled	Compute Engine default service account		No keys	

② Permissions (optional)

Grant this service account access to GCloud-learn so that it has permission to complete specific actions on the resources in your project. [Learn more](#)

Role	IAM condition (optional)	Action
Cloud SQL Admin	+ Add IAM condition	
Cloud SQL Client	+ Add IAM condition	

[+ Add another role](#)

[Help me choose roles](#)

**Cloud SQL Admin Role** have Full control on Cloud SQL , While **Cloud SQL Client Role** allow you to connect Cloud SQL with our Backend

## Step 4: Download Service Account Key

1. Open your service account
2. Go to **Keys** tab
3. Click **Add Key** → **Create New Key**
4. Select **JSON**
5. Download and rename it to `key.json`

📌 Place `key.json` in your Node.js project root.

## Step 5: Configure MySQL User

### Option 1: Use Existing `root` User

1. Cloud SQL → Instance → **Users**
2. Click : next to `root`
3. Choose **Change Password**
4. Generate password and save it

Click on Users, You will get a `root` user , Click on (:) dots and Choose **Change password**

The screenshot shows the Cloud SQL Instances page. On the left, there's a sidebar with options like Overview, Cloud SQL Studio, System insights, Query insights, Connections, and the current selection, **Users**. The main area shows 'my-db-instance' with a 'Users' tab selected. It displays a table with one row for 'root'. The table columns are User name, Host name, Authentication, and Password status. The 'root' row has a context menu with options: Change password (highlighted with a yellow arrow), Edit password policy, and Remove.

User name	Host name	Authentication	Password status
root	% (any host)	Built-in	N/A

### Option 2: Create New User (Optional)

You can also create a custom MySQL user if needed.

All instances > my-db-instance  
my-db-instance

MySQL 8.4

User accounts enable users and applications to connect to your instance. [Learn more](#)

**Add user account**

Added users    Authenticated IAM group members

These are accounts that you have granted instance access to, using either built-in or IAM authentication.

	User name ↑	Host name	Authentication	Password status	⋮
●	root	% (any host)	Built-in	N/A	⋮

## Step 6: Create Node.js Project

```
mkdir nodejs-app
cd nodejs-app
npm init -y
```

### Install dependencies:

```
npm install express mysql2 @google-cloud/cloud-sql-connector
```

Update `package.json`:

```
{
  "type": "module"
}
```

## Step 7: Node.js Application Code

Create `index.js`:

```
import express from "express";
import mysql from "mysql2/promise";
import { Connector } from "@google-cloud/cloud-sql-connector";

const app = express();
const PORT = process.env.PORT || 3000;
app.use(express.json());
```

```

const connector = new Connector();

async function getDbConfig() {
  const clientOpts = await connector.getOptions({
    instanceConnectionName: process.env.INSTANCE_CONNECTION_NAME,
    authType: 'PASSWORD',
  });

  return {
    ...clientOpts,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME,
  };
}

app.get('/health/db', async (req, res) => {
  try {
    const dbConfig = await getDbConfig();
    const conn = await mysql.createConnection(dbConfig);
    const [rows] = await conn.execute('SELECT NOW()');
    await conn.end();

    res.json({ status: 'success', data: rows[0] });
  } catch (err) {
    res.status(500).json({ status: 'error', error: err.message });
  }
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});

```

## Step 8: Environment Variables

Create `.env` file:

```
# Cloud SQL Configuration
INSTANCE_CONNECTION_NAME=project-id:region:instance-name
DB_USER=root
DB_PASSWORD=your-root-password
DB_NAME=my-node-db

# Application
PORT=3000
NODE_ENV=development
```

 **NOTE:** Ensure there is not spaces

Set credentials:

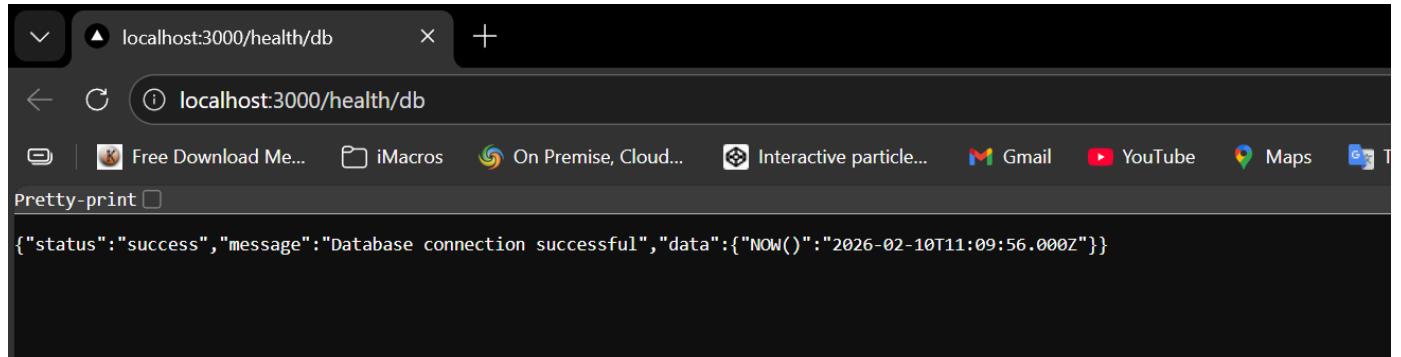
```
set GOOGLE_APPLICATION_CREDENTIALS=../key.json
```

## Step 9: Run the Application

```
node --env-file=.env index.js
```

Open browser: <http://localhost:3000/health/db>

- If successful, you'll see current database timestamp.



## Common Errors & Fixes

### Access denied for user

- Check username & password
- Ensure DB name is correct

### Cannot find credentials

- Verify `GOOGLE_APPLICATION_CREDENTIALS`
- Ensure `key.json` exists

### Connection timeout

- Ensure Cloud SQL instance is running
  - Check instance connection name
- 

## Security Notes

-  Do NOT commit `key.json`
  -  Do NOT commit DB password
  -  Use `.gitignore`
  -  Password-based auth is **not recommended for production**
- 

## What's Next?

In the next blog, we'll upgrade this setup to:

### IAM Database Authentication (No Passwords)

- More secure
  - Production ready
  - Best for GKE / Cloud Run
- 

## Conclusion

You've successfully connected **Node.js with Google Cloud SQL using password authentication**.

This method is perfect for:

- Learning Cloud SQL
- Local testing
- Understanding Cloud SQL internals

 Next step: **IAM-based authentication** 