





# 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**

**Choose a Cloud SQL edition**  
A Cloud SQL edition determines foundational characteristics of your instance. Choose the best option for your price and performance needs. [Learn more](#)

☐ **Enterprise Plus**

- 99.99% availability SLA
- Sub-second planned maintenance downtime
- Near-zero downtime instance scale-up
- Performance optimized machines
- Up to 35 days point-in-time recovery window
- Up to 3x higher read throughput with data cache
- Advanced disaster recovery with easy switchback
- Support for managed connection pool

☒ **Enterprise**

- 99.95% availability SLA
- Less than 60 seconds planned maintenance downtime
- General purpose machines
- Up to 7 days point-in-time recovery window

Choose a preset for this edition. Presets can be customized later as needed.

Edition preset: Sandbox

[Compare edition presets](#)

### Summary

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

### Pricing estimate (without discounts)

These items represent Cloud SQL compute, memory and storage resources only, and reflect how you configured your instance so far. Discounts not

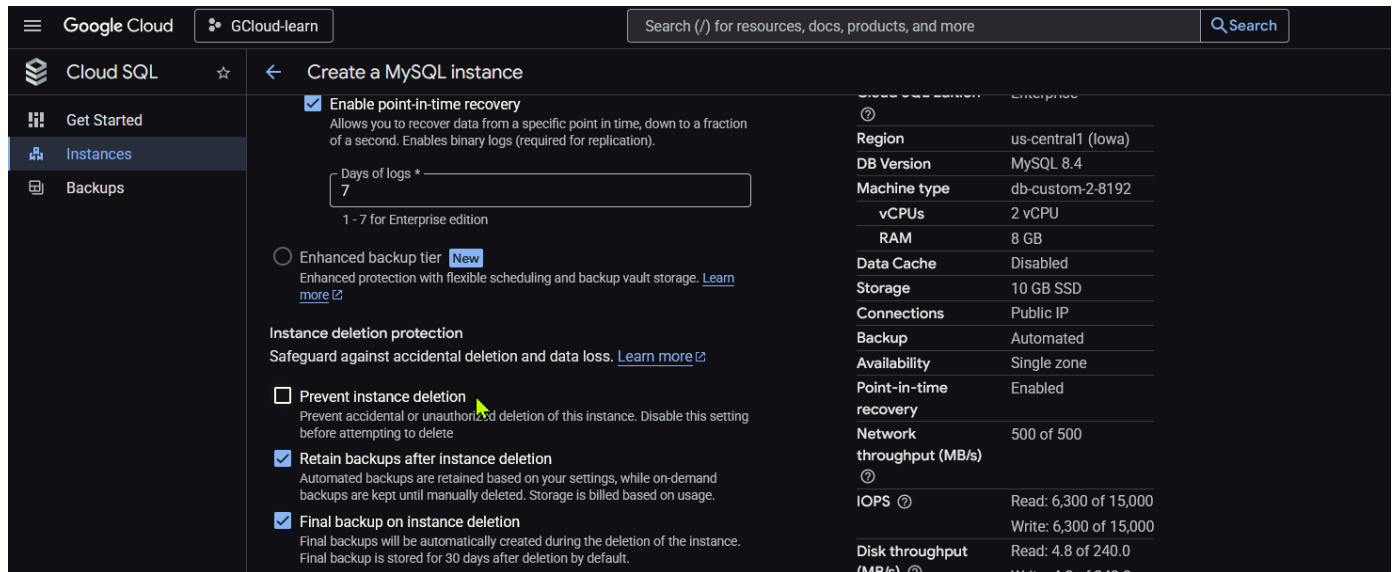
## 4. Recommended Settings:

- **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



**Google Cloud** GCloud-learn Search (/) for resources, docs, products, and more Q Search

**Cloud SQL** ☆

**Create a MySQL instance**

☒ **Enable point-in-time recovery**  
Allows you to recover data from a specific point in time, down to a fraction of a second. Enables binary logs (required for replication).

Days of logs \*   
1 - 7 for Enterprise edition

☐ **Enhanced backup tier** **New**  
Enhanced protection with flexible scheduling and backup vault storage. [Learn more](#)

**Instance deletion protection**  
Safeguard against accidental deletion and data loss. [Learn more](#)

☐ **Prevent instance deletion**  
Prevent accidental or unauthorized deletion of this instance. Disable this setting before attempting to delete

☒ **Retain backups after instance deletion**  
Automated backups are retained based on your settings, while on-demand backups are kept until manually deleted. Storage is billed based on usage.

☒ **Final backup on instance deletion**  
Final backups will be automatically created during the deletion of the instance. Final backup is stored for 30 days after deletion by default.

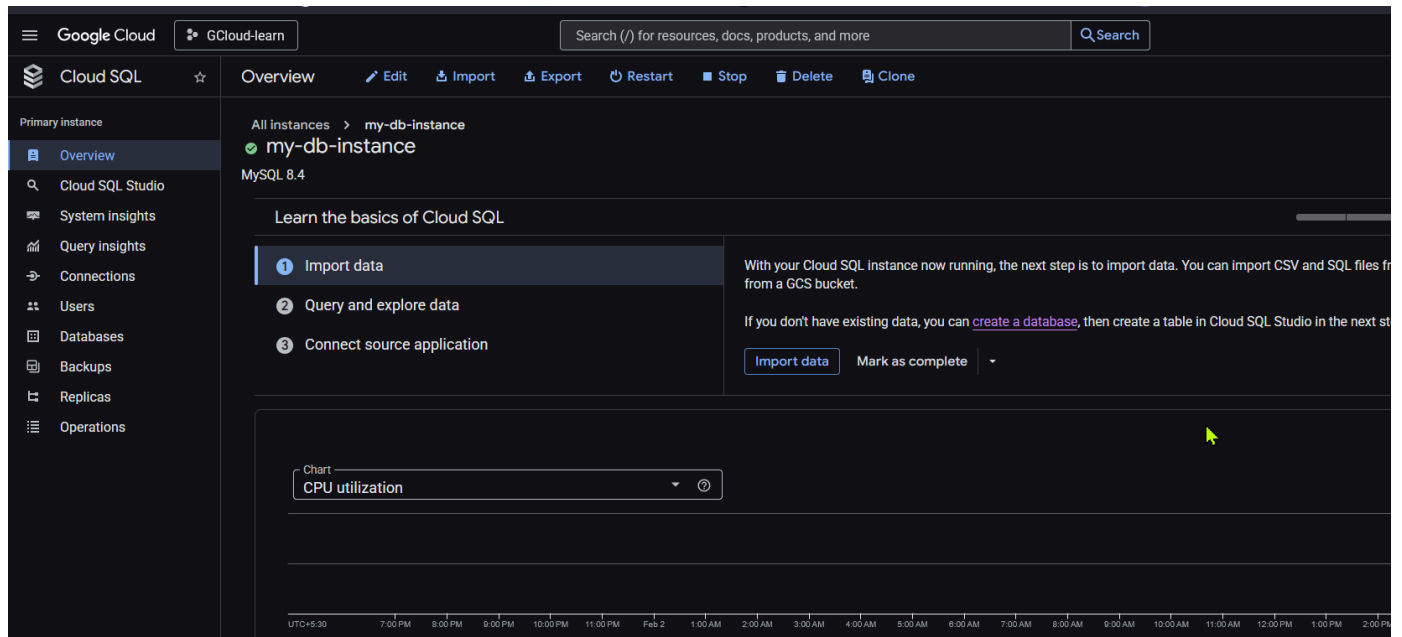
<b>Region</b>	us-central1 (Iowa)
<b>DB Version</b>	MySQL 8.4
<b>Machine type</b>	db-custom-2-8192
<b>vCPUs</b>	2 vCPU
<b>RAM</b>	8 GB
<b>Data Cache</b>	Disabled
<b>Storage</b>	10 GB SSD
<b>Connections</b>	Public IP
<b>Backup</b>	Automated
<b>Availability</b>	Single zone
<b>Point-in-time recovery</b>	Enabled
<b>Network throughput (MB/s)</b>	500 of 500
<b>IOPS</b>	Read: 6,300 of 15,000 Write: 6,300 of 15,000
<b>Disk throughput (MB/s)</b>	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 won't charge anything for Creating Database Instance and Running for 2 Hours, Still If you see charges on billing on Cloud Console, It's 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 :



**Google Cloud** GCloud-learn Search (/) for resources, docs, products, and more Q Search

**Cloud SQL** ☆

**Overview** Edit Import Export Restart Stop Delete Clone

Primary instance

All instances > my-db-instance

my-db-instance

MySQL 8.4

Learn the basics of Cloud SQL

1 Import data

2 Query and explore data

3 Connect source application

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.

If you don't have existing data, you can [create a database](#), then create a table in Cloud SQL Studio in the next step.

Import data Mark as complete

Chart

CPU utilization

UTC+5:30 7:00 PM 8:00 PM 9:00 PM 10:00 PM 11:00 PM Feb 2 1:00 AM 2:00 AM 3:00 AM 4:00 AM 5:00 AM 6:00 AM 7:00 AM 8:00 AM 9:00 AM 10:00 AM 11:00 AM 12:00 PM 1:00 PM 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

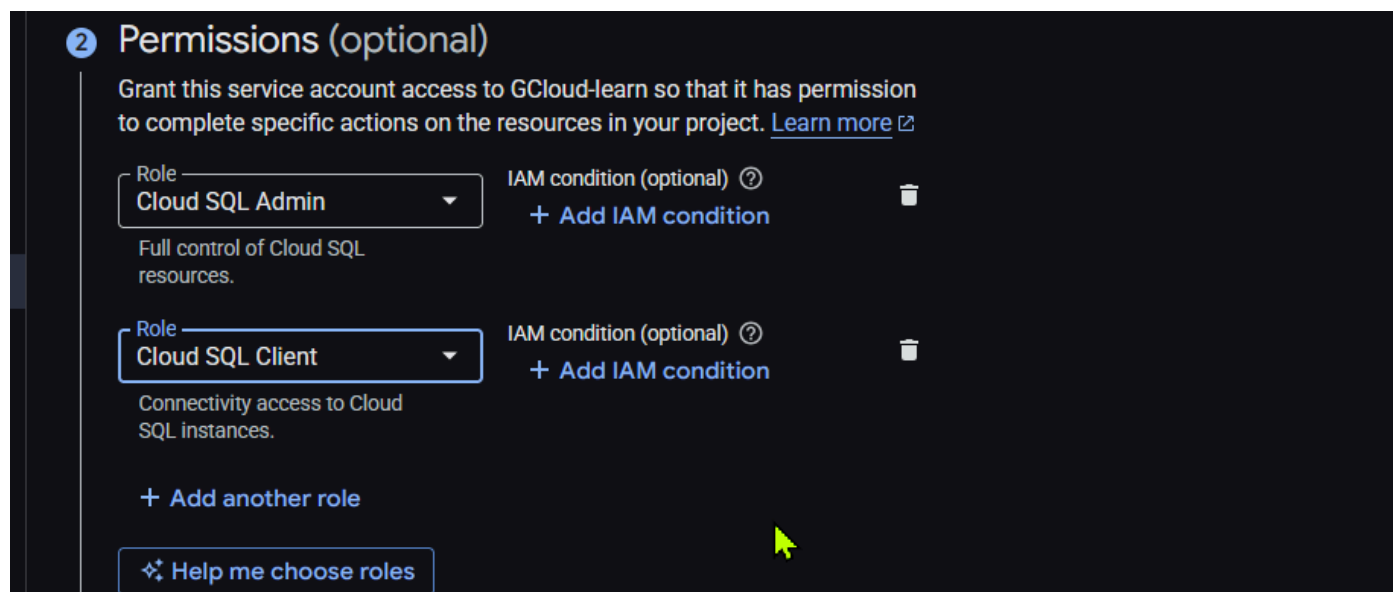
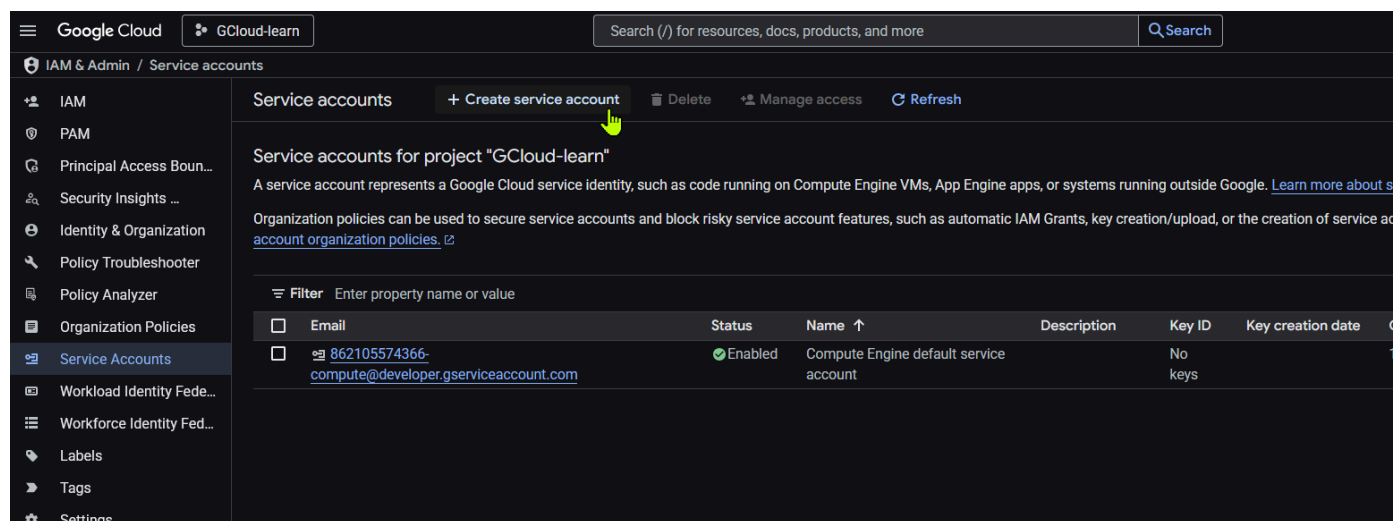
1. Navigate to **IAM & Admin** → **Service Accounts**
2. Click **Create Service Account**
3. Enter name (example: `nodejs-app-sa`)
4. Click **Create and Continue**

## Assign Roles

Add these roles:

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

Click **Done**.



**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.

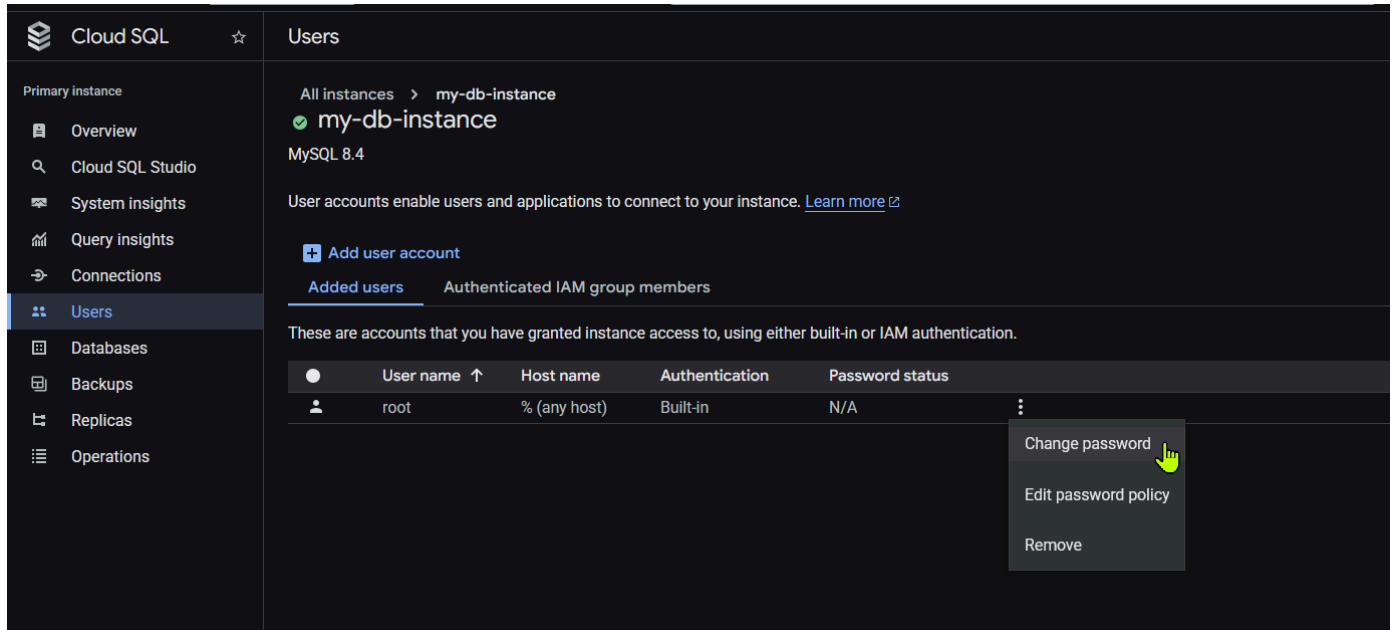
---

## 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**

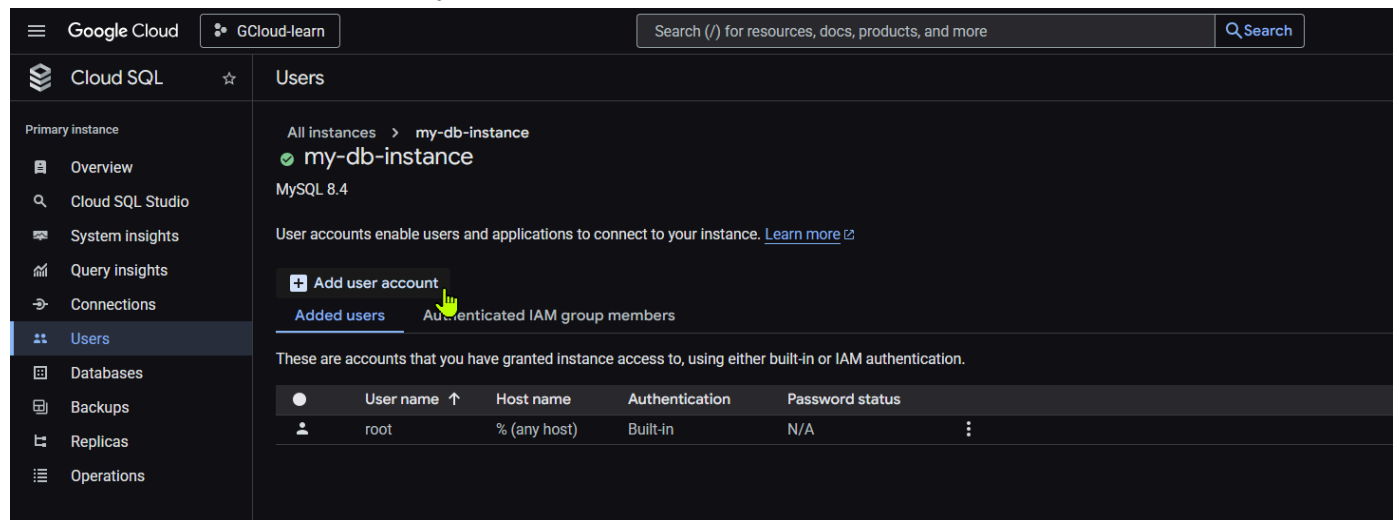


The screenshot shows the Google Cloud SQL Admin console. On the left is a sidebar with navigation options: Overview, Cloud SQL Studio, System insights, Query insights, Connections, Users (selected), Databases, Backups, Replicas, and Operations. The main panel shows the 'Users' page for a MySQL instance named 'my-db-instance'. It includes a table of users with columns: User name, Host name, Authentication, and Password status. The 'root' user is listed with authentication 'Built-in' and password status 'N/A'. A context menu is open for the 'root' user, showing options: 'Change password' (highlighted with a yellow cursor), '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.



The screenshot shows the Google Cloud console interface. On the left is a navigation menu with options like Overview, Cloud SQL Studio, System insights, Query insights, Connections, Users (selected), Databases, Backups, Replicas, and Operations. The main area displays the 'Users' page for the instance 'my-db-instance'. It shows the instance name, version (MySQL 8.4), and a link to 'Add user account'. Below this, there are tabs for 'Added users' and 'Authenticated IAM group members'. A table lists the current users, showing a single user named 'root' with built-in authentication and no password status.

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
```

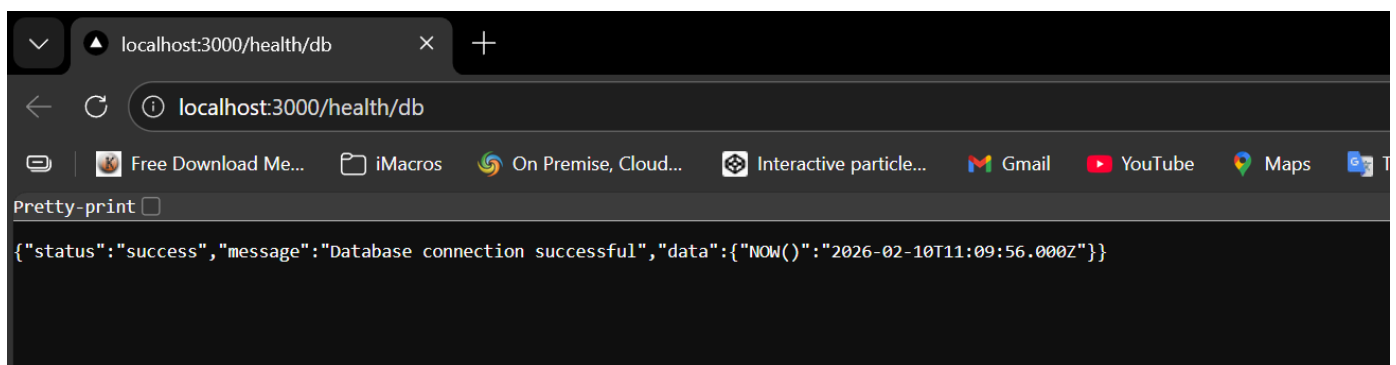
---

## 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** 