

Gemini CLI with MCP Servers

(With an example of a connection to a PostgreSQL Database)

Quick Guide (CLI)

1. Installation of the Gemini CLI

To install the Gemini CLI on your device you need:

- a stable internet connection
- to install [node.js](#)

HERE IS HOW TO INSTALL NODE USING ONLY THE TERMINAL:

-> ON WINDOWS: > *winget install -e --id OpenJS.NodeJS.LTS*

-> ON LINUX: > *curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -*
 > *sudo apt-get install -y nodejs*

-> ON MAC: > *brew install node*

When you have it installed:

1. Check if **node** installed correctly:

```
> node -v  
> npx -v  
> npm -v
```

If installed correctly go on, if not try again or use docs ([Node.js API documentation](#))

2. Installation of Gemini CLI

To install the Gemini CLI globally in the terminal write:

```
> npm install -g @google/gemini-cli
```

If no errors occur go on, if something shows up clean cache (npm cache clean OR npm cache clean -- force) and try again or use docs ([Gemini CLI documentation](#))

3. Starting up and logging in

To startup the Gemini CLI interface use:

```
> gemini
```

Then if everything went according to plan you should see this screen:



From here you have 3 options:

- Login with Google ([Simple login with your google account](#))
- Google API Key ([Auth using your Gemini API Key](#))
- Vertex AI ([Auth using Google Cloud's Vertex AI platform](#))

After a successful authentication you will see this screen:



And from here on out you can use the Gemini CLI every time you type in “`gemini`” into the terminal!

2. Using a MCP Server in Gemini CLI

To find a MCP Server you want to use visit [Gemini CLI Extensions](#). When you install your desired MCP Server don't forget to check out its documentation which you can find by going to your selected extensions GitHub page.

3. PostgreSQL MCP Server (*EXAMPLE*)

If you want to connect your Gemini CLI with your PostgreSQL database you need to use one of the following extensions:

- [PostgreSQL Cloud MCP Server](#)
- [PostgreSQL Local MCP Server](#)

You can install them easily using the following commands:

```
> gemini extensions install
```

<https://github.com/gemini-cli-extensions/cloud-sql-postgresql> (Cloud)

> gemini extensions install <https://github.com/gemini-cli-extensions/postgres> (Local)

According to the documentation of both extensions, after installing your chosen MCP Server you first need to set environment variables that allow the server to connect to the database.

For PostgreSQL Cloud MCP Server:

ON LINUX AND MAC:

```
> export CLOUD_SQL_POSTGRES_PROJECT=<your-gcp-project-id>
> export CLOUD_SQL_POSTGRES_REGION=<your-cloud-sql-region>
> export CLOUD_SQL_POSTGRES_INSTANCE=<your-cloud-sql-instance-id>
> export CLOUD_SQL_POSTGRES_DATABASE=<your-database-name>
> export CLOUD_SQL_POSTGRES_USER=<your-database-user> # Optional
> export CLOUD_SQL_POSTGRES_PASSWORD=<your-database-password> # Optional
> export CLOUD_SQL_POSTGRES_IP_TYPE="PUBLIC" # Optional
Options: `PUBLIC`, `PRIVATE`, `PSC`.
(Defaults to `PUBLIC`).
```

ON WINDOWS:

```
> $env:CLOUD_SQL_POSTGRES_PROJECT = "<your-gcp-project-id>"
> $env:CLOUD_SQL_POSTGRES_REGION = "<your-cloud-sql-region>"
> $env:CLOUD_SQL_POSTGRES_INSTANCE = "<your-cloud-sql-instance-id>"
> $env:CLOUD_SQL_POSTGRES_DATABASE = "<your-database-name>"
> $env:CLOUD_SQL_POSTGRES_USER = "<your-database-user>" # Optional
> $env:CLOUD_SQL_POSTGRES_PASSWORD = "<your-database-password>" # Optional
> $env:CLOUD_SQL_POSTGRES_IP_TYPE = "PUBLIC" # Optional
Options: `PUBLIC`, `PRIVATE`, `PSC`.
(Defaults to `PUBLIC`).
```

For PostgreSQL Local MCP Server:

ON LINUX AND MAC:

```
> export POSTGRES_HOST=<your-postgres-host>
> export POSTGRES_PORT=<your-postgres-port>
> export POSTGRES_DATABASE=<your-database-name>
> export POSTGRES_USER=<your-database-user>
> export POSTGRES_PASSWORD=<your-database-password>
```

ON WINDOWS:

```
> $env:POSTGRES_HOST = "<your-postgres-host>"
> $env:POSTGRES_PORT = "<your-postgres-port>"
> $env:POSTGRES_DATABASE = "<your-database-name>"
> $env:POSTGRES_USER = "<your-database-user>"
> $env:POSTGRES_PASSWORD = "<your-database-password>"
```

To load the environmental variables you can also use a **.env** file!

SAMPLE .env FILE:

For PostgreSQL Cloud MCP Server:

```
CLOUD_SQL_POSTGRES_PROJECT=<your-gcp-project-id>
CLOUD_SQL_POSTGRES_REGION=<your-cloud-sql-region>
CLOUD_SQL_POSTGRES_INSTANCE=<your-cloud-sql-instance-id>
CLOUD_SQL_POSTGRES_DATABASE=<your-database-name>
CLOUD_SQL_POSTGRES_USER=<your-database-user> #Optional
CLOUD_SQL_POSTGRES_PASSWORD=<your-database-password> #Optional
CLOUD_SQL_POSTGRES_IP_TYPE=PUBLIC
#Optional (Options: PUBLIC, PRIVATE, PSC)
```

For PostgreSQL Local MCP Server:

```
POSTGRES_HOST = "<your-postgres-host>"
POSTGRES_PORT = "<your-postgres-port>"
POSTGRES_DATABASE = "<your-database-name>"
POSTGRES_USER = "<your-database-user>"
POSTGRES_PASSWORD = "<your-database-password>"
```

TO USE THIS METHOD YOU JUST NEED TO:

1. Install dotenv (using node):

```
> npm install dotenv
```

2. Load it in your code (usually at the top of index.js or app.js):

```
import dotenv from "dotenv";
dotenv.config(); // Loads variables from .env
```

```
console.log(process.env.POSTGRES_HOST);
```

OR

1. Find your settings.json file

Usually found here **C:\Users\<username>\.gemini\settings.json**

2. Insert your variables straight into the connection string

```
"mcpServers": {
  "postgres": {
    "command": [
      "npx",
      "-y",
      "@modelcontextprotocol/server-postgres",
      "postgres://postgres@localhost:5432/TestDB"
    ]
  }
},
```

SAMPLE CONNECTION STRING:

> *postgres://username:password@hostname:port/database*

Remember to set the user and password to an account with the correct permissions for what you want the LLM to be able to do!

Using natural language prompts and a user with sufficient privileges, you can select, insert, delete, and perform virtually any database operation faster and more easily.

⚠️ Be careful: the LLM won't care about preserving your data, so make sure you use a restricted account or test environment!

After setting the environmental variables you can use the *gemini* command to run the Gemini CLI and check if the MCP Server is working. To check if the MCP Server is running use the following command:

> */mcp refresh*

After running the command you should see this in your CLI:

```
● postgresql (from postgres) - Ready (11 tools)
Tools:
- execute_sql
- get_query_plan
- list_active_queries
- list_autovacuum_configurations
- list_available_extensions
- list_installed_extensions
- list_invalid_indexes
- list_memory_configurations
- list_replication_slots
- list_tables
- list_top_bloated_tables
```

This indicates that the MCP Server is up and running, connected to the database using the environmental variables.

If you see this after running the command:

```
● postgres - Disconnected
```

This means that your MCP Server didn't connect properly to the database.

If the MCP Server is running you're going to be able to prompt the LLM from Gemini CLI to use the MCP Server for getting information about, out and into the database using a natural language and explaining what you mean instead of complex commands.

SAMPLE PROMPT: “*Use the postgresql MCP Server to analyze, query and document my database*”

4. Use the Gemini CLI with the VS Code IDE

- **Run Gemini CLI in the VS Code Terminal**
- **Accept the VS Code Integration Prompt:** Accept the prompt about adding it to VS Code, and the extension should install itself automatically.
- **Manual Installation (If the Prompt Doesn't Appear):** If the prompt doesn't appear, there's no need to worry. Simply enter the

following command yourself in the Gemini CLI: `/ide install` The extension should then install.

- **Open Your Project Directory:** Next, open the project directory you want to work in.
- **Enable the Agent in the Project:** Relaunch Gemini and use the command: `/ide enable`

CONGRATULATIONS! YOU CAN NOW PROMPT THE AI AGENT FROM GEMINI CLI TO WRITE CODE!

CONGRATULATIONS YOU NOW KNOW HOW TO USE Gemini CLI AND HOW TO INSTALL AND USE MCP SERVERS WITH IT!