Troubleshooting in PostgreSQL

Estimated time needed: 30 minutes

In this lab, you will obtain hands-on experience in troubleshooting common issues you may encounter as a database administrator. The most common problems encountered with databases are caused by poor performance, improper configuration, or poor connectivity. You will use a PostgreSQL server instance to explore some of these possible problems and rectify them.

Objectives

After completing this lab, you will be able to:

- Enable error logging for your PostgreSQL instance.
- Access server logs for troubleshooting.
- Diagnose commonly encountered issues caused by poor performance, improper configuration, or poor connectivity.
- Resolve common issues you may encounter as a database administrator.

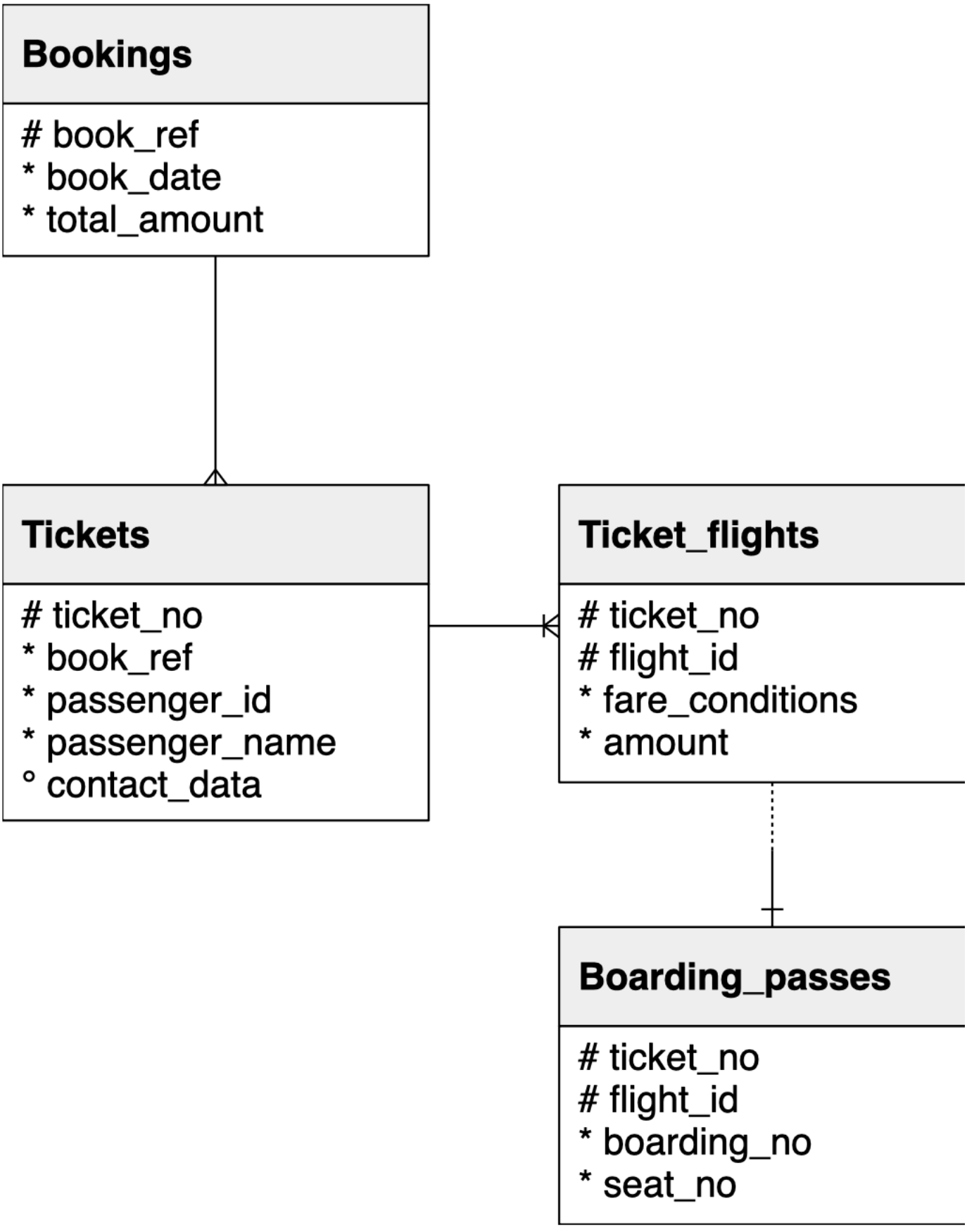
Software Used in This Lab

In this lab, you will be using PostgreSQL. It is a popular open source object relational database management system (RDBMS) capable of performing a wealth of database administration tasks, such as storing, manipulating, retrieving, and archiving data.

To complete this lab, you will be accessing the PostgreSQL service through the IBM Skills Network (SN) Cloud IDE, which is a virtual development environnement you will use throughout this course.

Database Used in This Lab

In this lab, you will use a database from https://postgrespro.com/education/demodb distributed under the PostgreSQL license. It stores a month of data about airline flights in Russia and is organized according to the following schema:



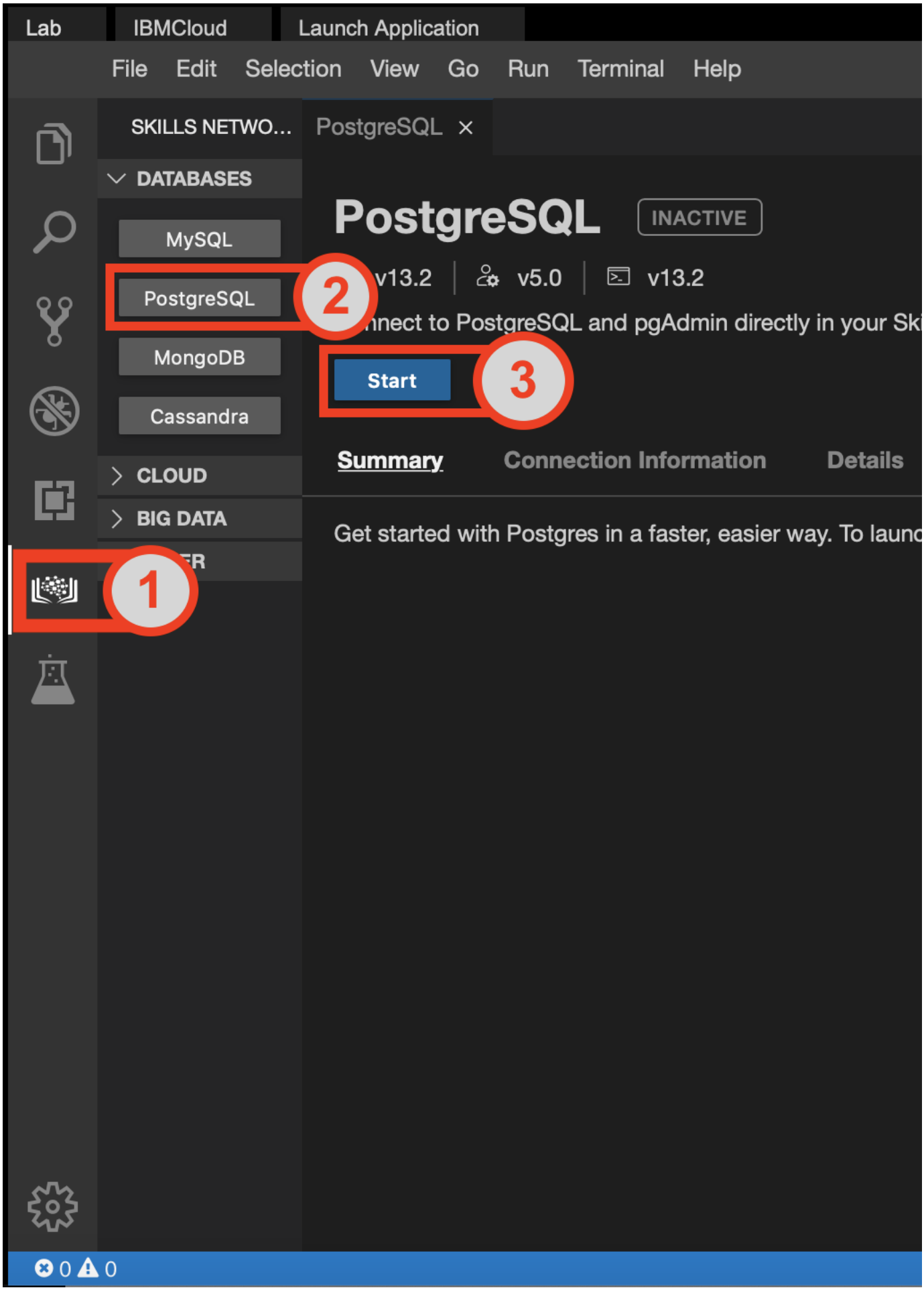
Exercise 1: Set Up Your Database in PostgreSQL

Task A: Launch PostgreSQL in Cloud IDE

To get started with this lab, launch PostgreSQL using the Cloud IDE. You can do this by following these steps:

1. Select the Skills Network extension button in the left pane.
2. Open the “DATABASES” dropdown menu and select “PostgreSQL.”

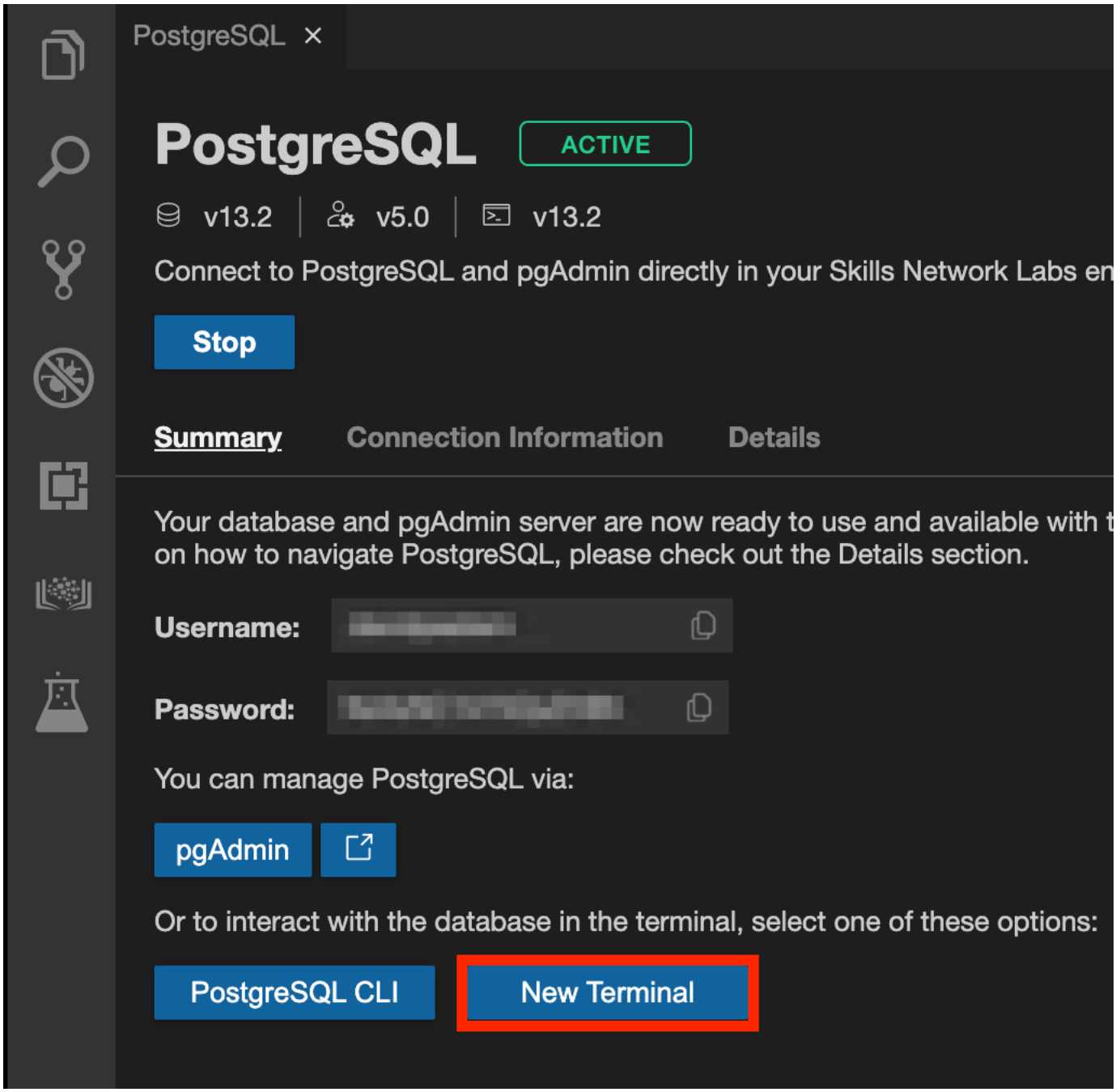
3. Select the “Start” button. PostgreSQL may take a few moments to start.



Task B: Download and Create the Database

First, you will need to download the database.

1. Open a new terminal by selecting the “New Terminal” button near the bottom of the interface.



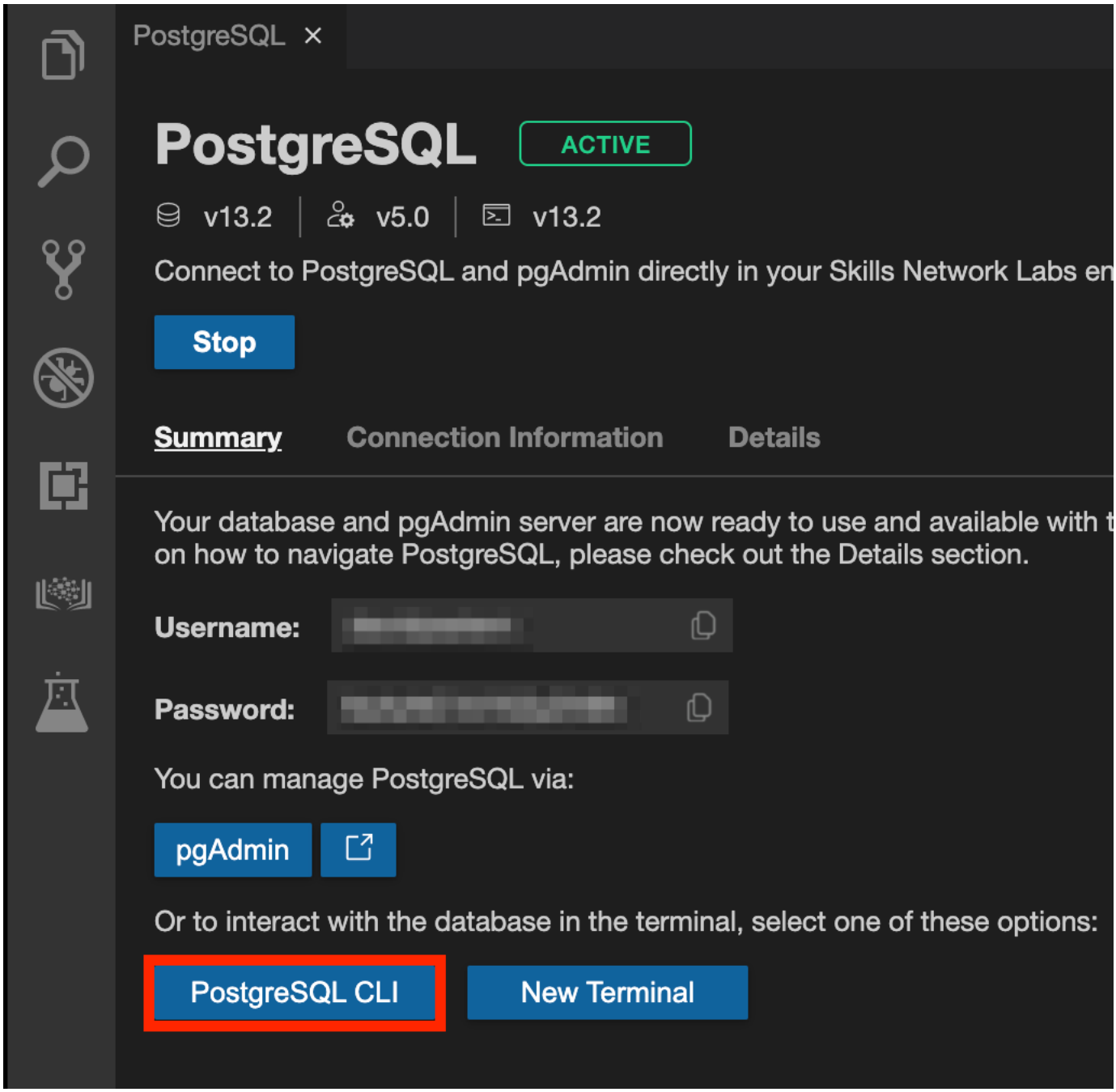
2. Run the following command in the terminal:

```
1. 1
1. wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/example-guided-project/flights_RUSSIA_small.sql
```

Copied!

The file that you downloaded is a full database backup of a month of flight data in Russia. Now, you can perform a full restoration of the data set by first opening the PostgreSQL CLI.

3. Near the bottom of the window, select the “PostgreSQL CLI” button to launch the command line interface (CLI).



4. In the PostgreSQL CLI, enter the command to restore the data you downloaded into a new database called **demo**.

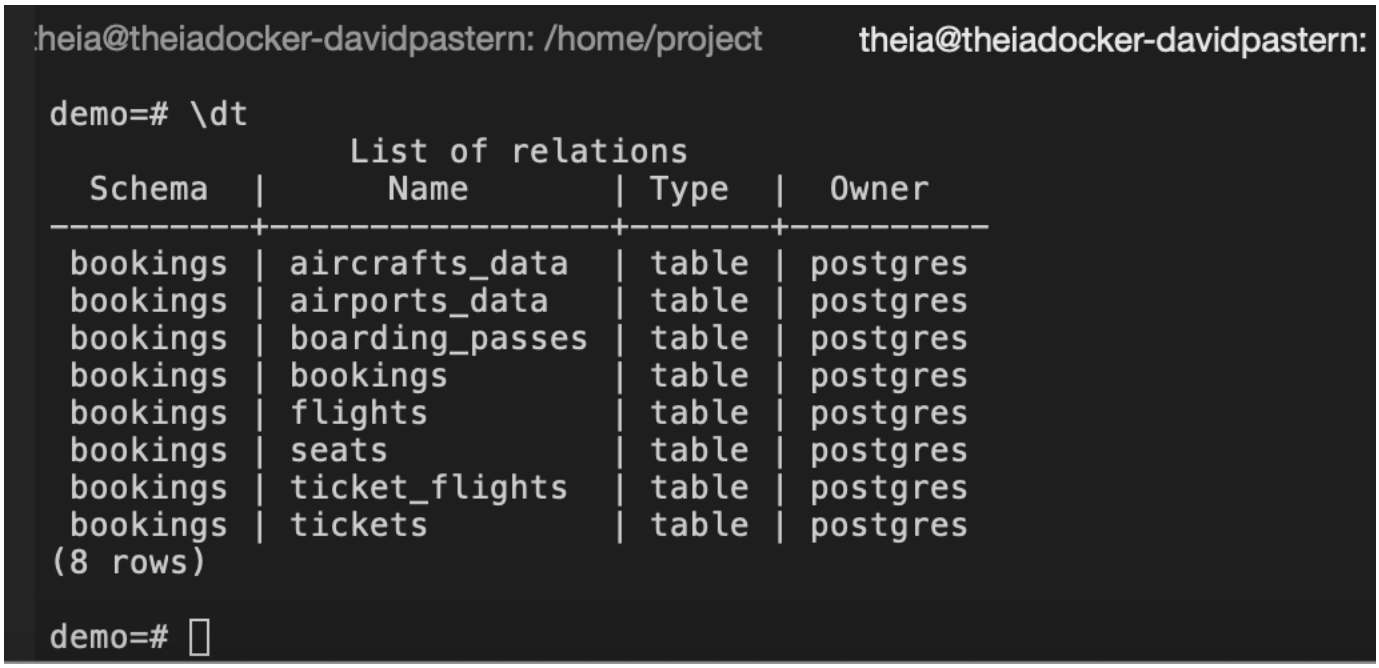
▶ Hint (click here)
▶ Solution (click here)

5. Verify that the database was properly created by entering the following command:

```
1. 1
1. \dt
```

Copied!

You should see the following output showing all the tables that are part of the `bookings` schema in the `demo` database.

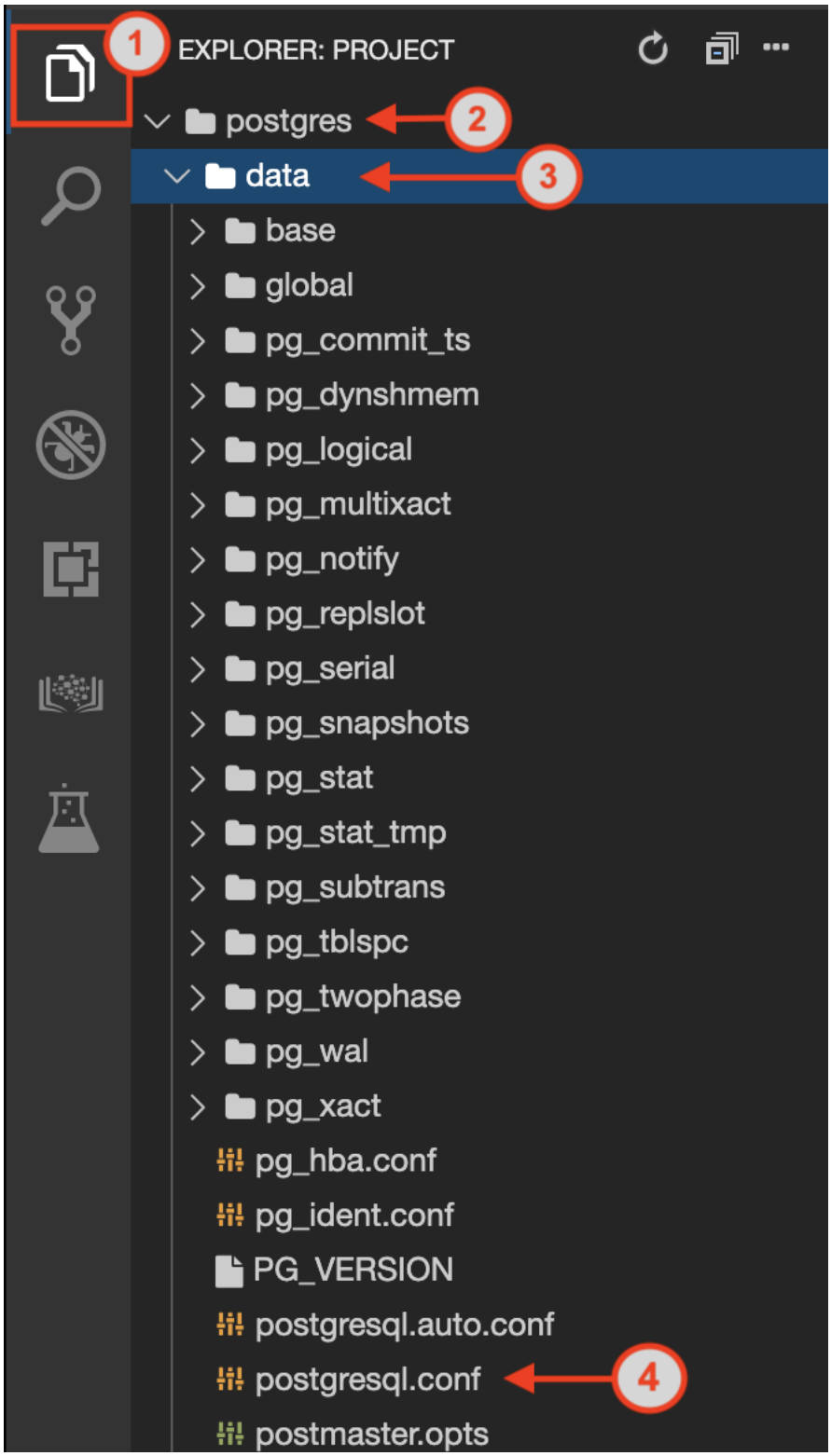


Exercise 2: Enable Error Logging and Observe Logs

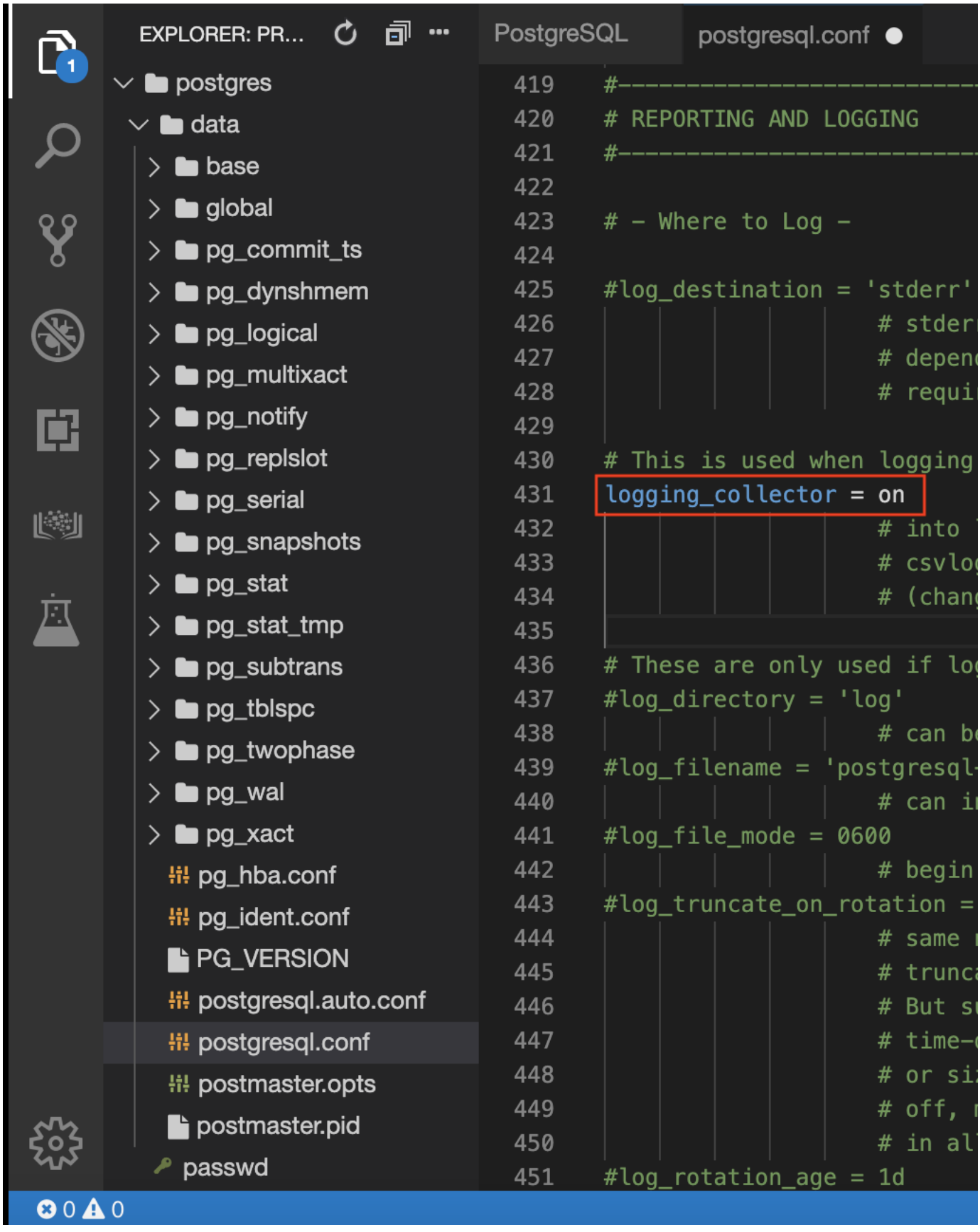
Task A: Enable Server Logging

First, to enable error logging on your PostgreSQL server instance, you will need to configure your server to support it. You can do so by using the Cloud IDE file explorer to open `postgresql.conf`, which stores the configuration parameters that are read upon server startup. Let's go ahead and do it.

1. You can open the file by first opening the file explorer on Cloud IDE then selecting `postgres > data > postgresql.conf`.

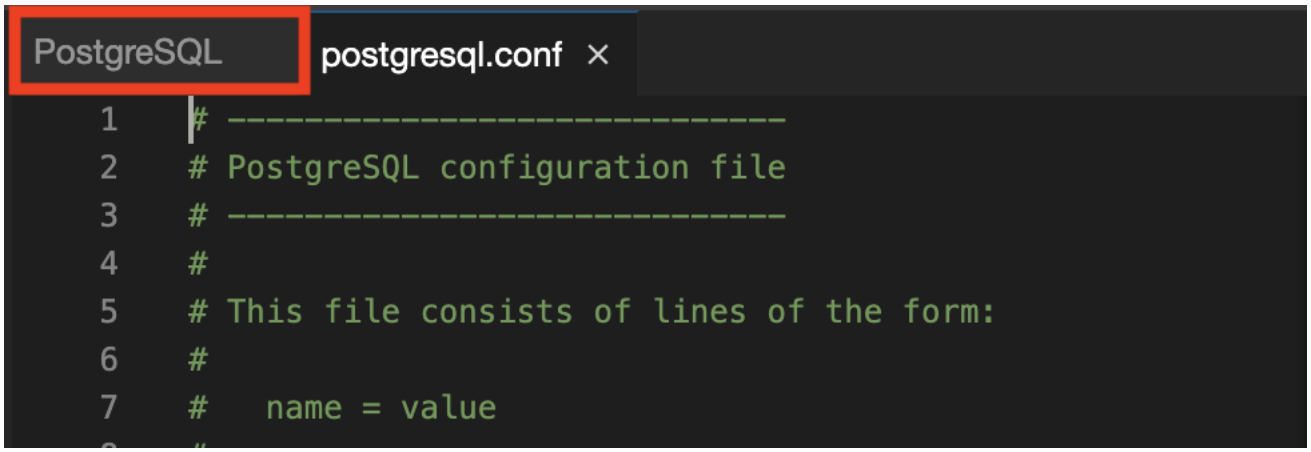


2. With the configuration file open, scroll down to line 431. Replace `logging_collector = off` with `logging_collector = on` and uncomment the parameter by removing the # before the line.

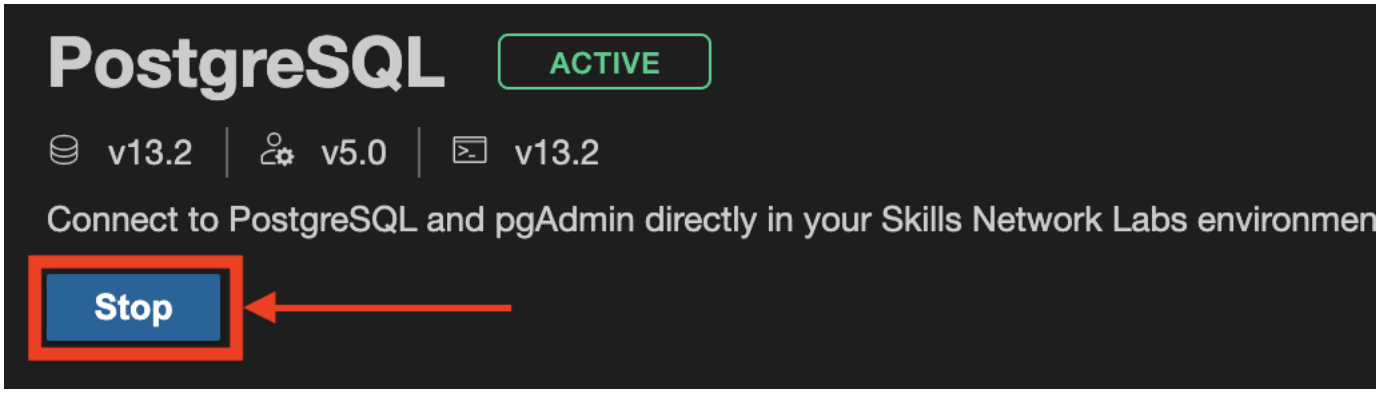


3. Save the changes to `postgresql.conf` by either navigating to **File > Save** at the top toolbar or by pressing **Ctrl + S** (Mac: ⌘ + S).

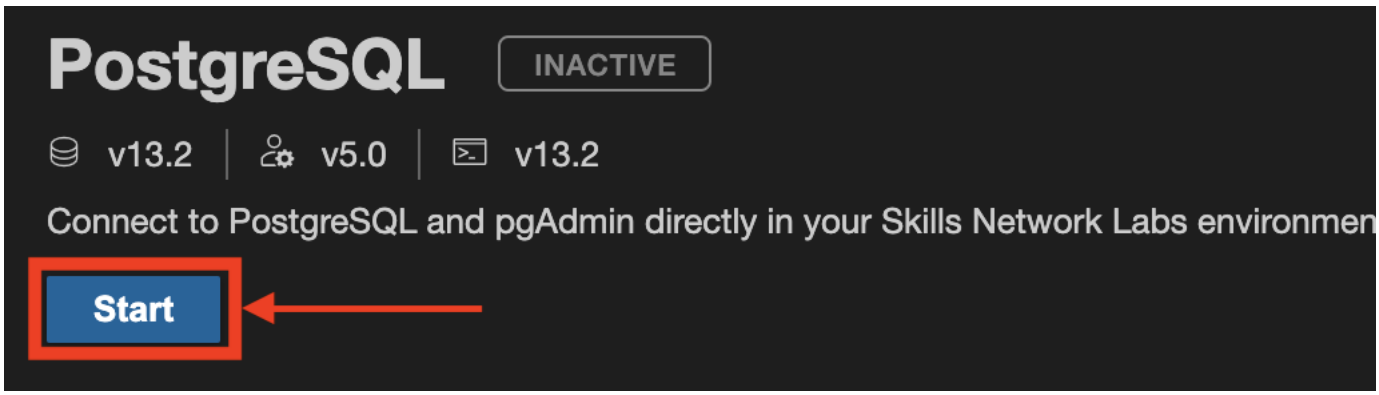
4. Changing this parameter requires a server restart in order to take effect. Select the PostgreSQL tab in Cloud IDE.



5. Stop the PostgreSQL server by selecting the “Stop” button and close all CLI and terminal tabs.



6. Now restart the PostgreSQL server by selecting the “Start” button. It may take a few moments to start up again. When it does so, reopen the PostgreSQL CLI.

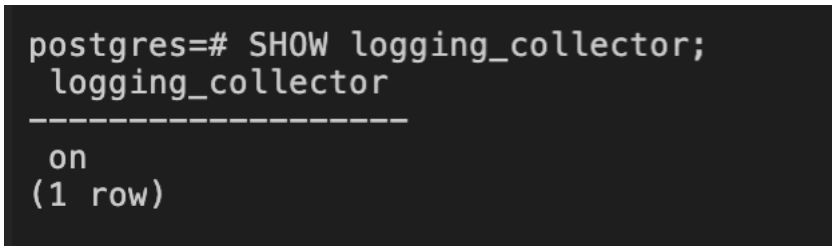


7. Confirm that the configuration parameter was successfully changed and loaded into the PostgreSQL instance by entering the following command into the CLI:

```
1. 1
1. SHOW logging_collector;
```

Copied!

You should see that the command returns **on**.



Task B: View the Server Logs

In this task, you will navigate the Cloud IDE file explorer to open up and inspect the server logs created after you enabled the logging in the previous task. The logs can be a valuable tool when troubleshooting issues as a database administrator. For now, let’s look at the logs created during normal operation, with nothing broken yet.

1. To find where the system logs are stored, enter the following command into the CLI:

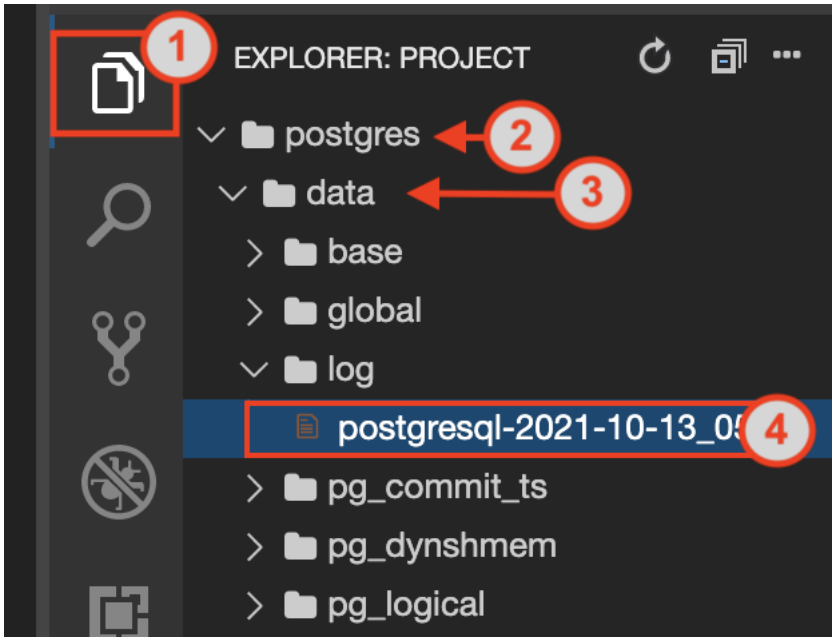
```
1. 1
1. SHOW log_directory;
```

Copied!

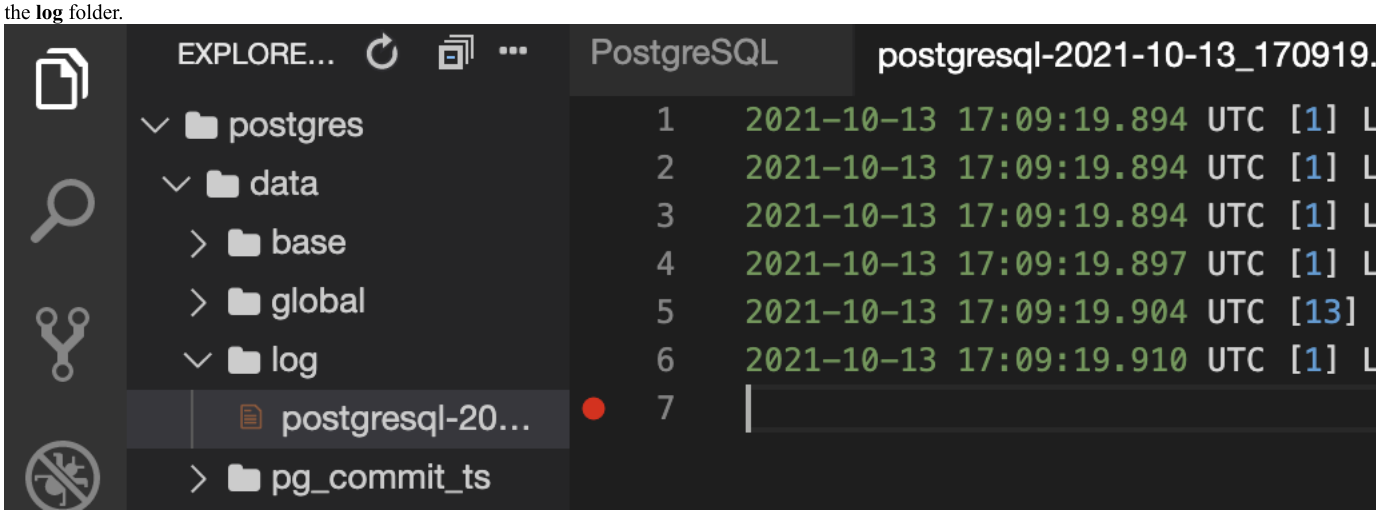
```
postgres=# SHOW log_directory;
 log_directory
---------------
 log
(1 row)
```

2. Open up the file explorer on Cloud IDE and navigate through **postgres > data > log**.

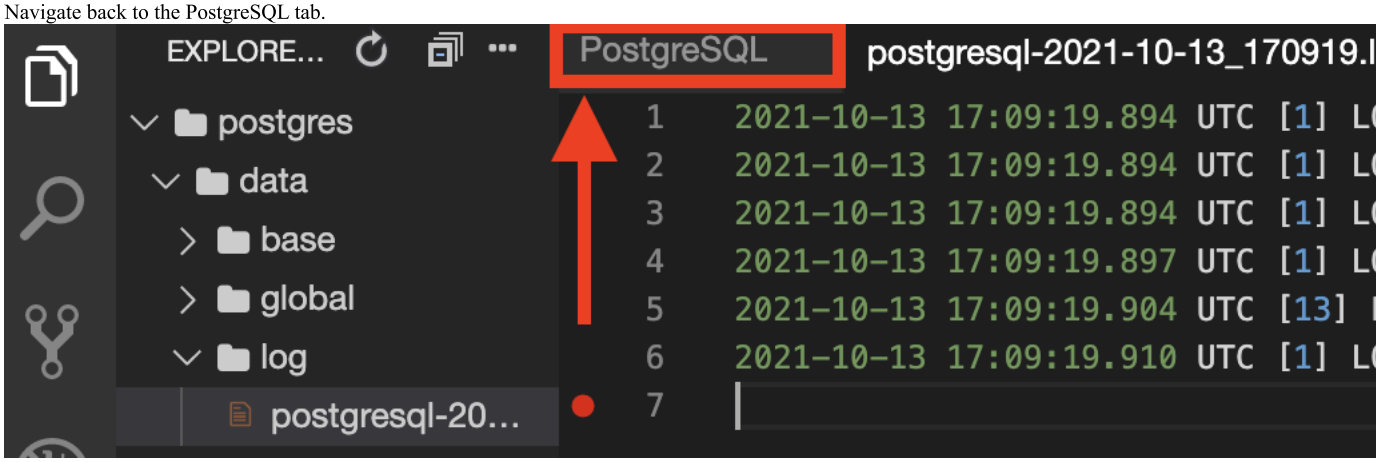
3. You will see a file with a name of the form `postgresql-YYYY-MM-DD-<numbers>.log`. Go ahead and open it.



4. Inspect and familiarize yourself with the logs given for a PostgreSQL server startup. Every time you start the server again, a new `.log` file will be created in the **log** folder.



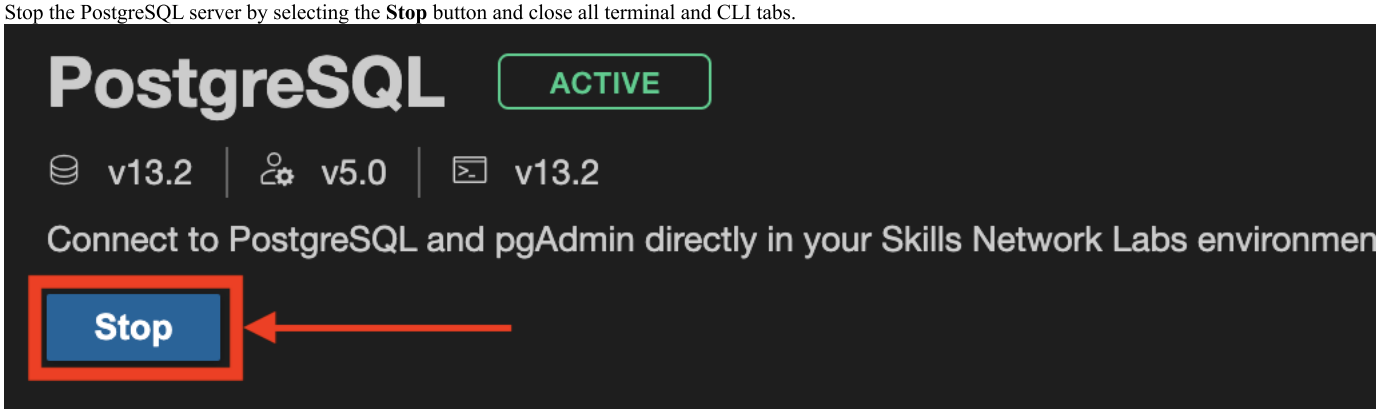
5. Navigate back to the PostgreSQL tab.



6. **Try it yourself:** Stop the PostgreSQL server and close all terminal tabs.

▼ Hint (click here)
Recall how you stopped the PostgreSQL server in the previous task.
▼ Solution (click here)

Stop the PostgreSQL server by selecting the **Stop** button and close all terminal and CLI tabs.



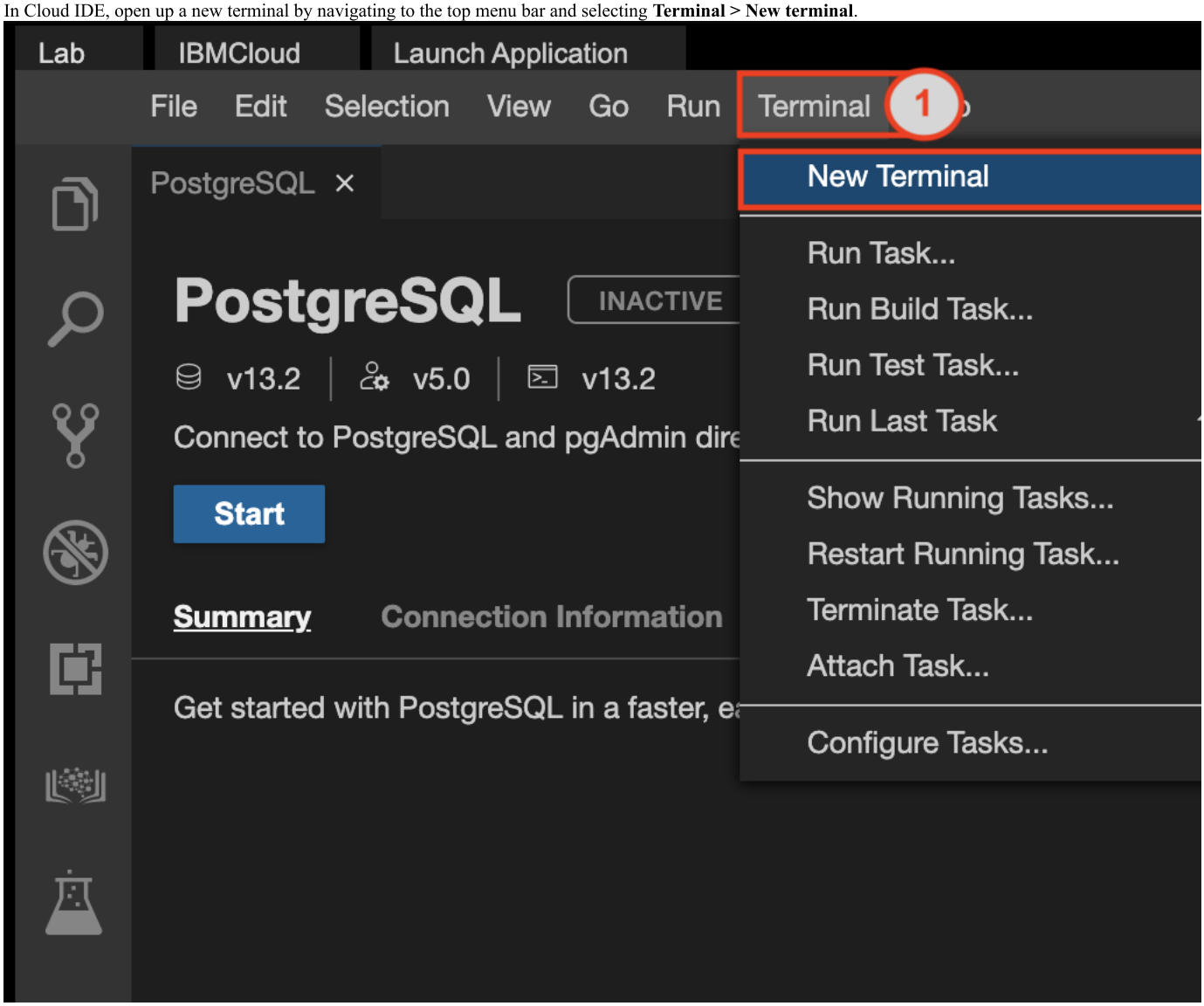
Exercise 3: Test the Performance of the PostgreSQL Server

The most common problems encountered with databases are caused by poor performance, improper configuration, or poor connectivity. Server configuration issues, such as inadequate hardware resources or misconfigured settings, can significantly impact performance. In this exercise, you will gain some hands-on experience in studying the performance of the PostgreSQL server and inspecting the logs to identify and resolve slow performance and connection disruptions.

Task A: Preparation for the Exercise

Before you get started, you'll have to set up a few things so that you can begin troubleshooting. In this task, you will first delete the **postgresql.conf** file and replace it with a new configuration file that has some parameters changed. This task is entirely setup and will allow you to complete the remainder of the tasks where you will test the performance of the server.

1. In Cloud IDE, open up a new terminal by navigating to the top menu bar and selecting **Terminal > New terminal**.



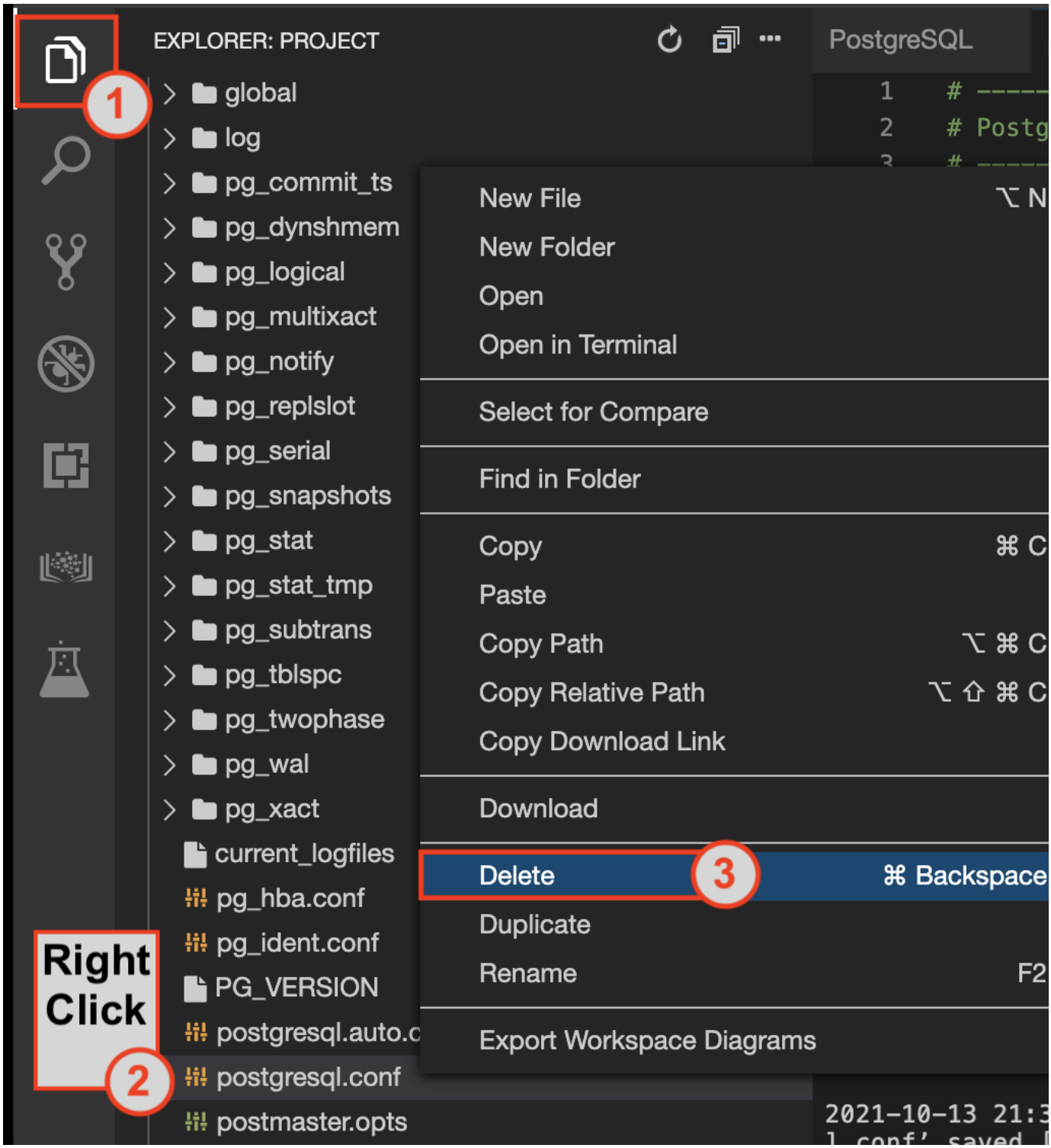
2. In the terminal, enter the following command to download a new `postgresql.conf` configuration file:

```
1. 1
1. wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0231EN-SkillsNetwork/labs/PostgreSQL/Lab%20-%20Troubleshootir
```

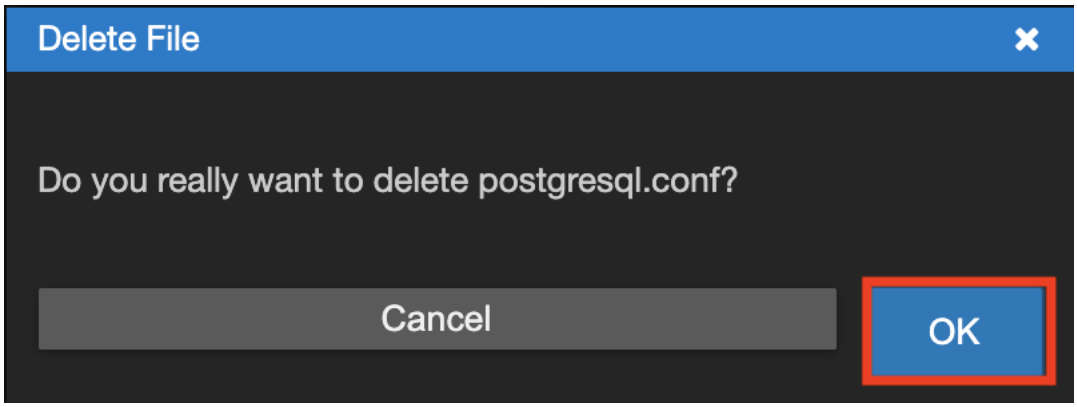
Copied!

3. Open up the file explorer on Cloud IDE and navigate to **postgres** > **data**.

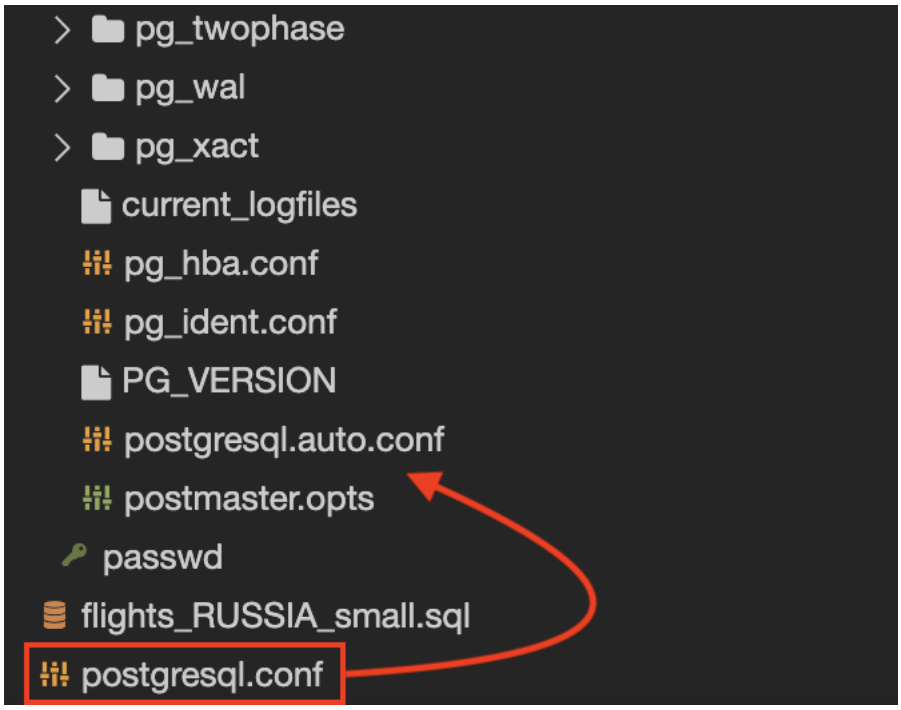
4. Right-click `postgresql.conf` in this directory and select **Delete**.



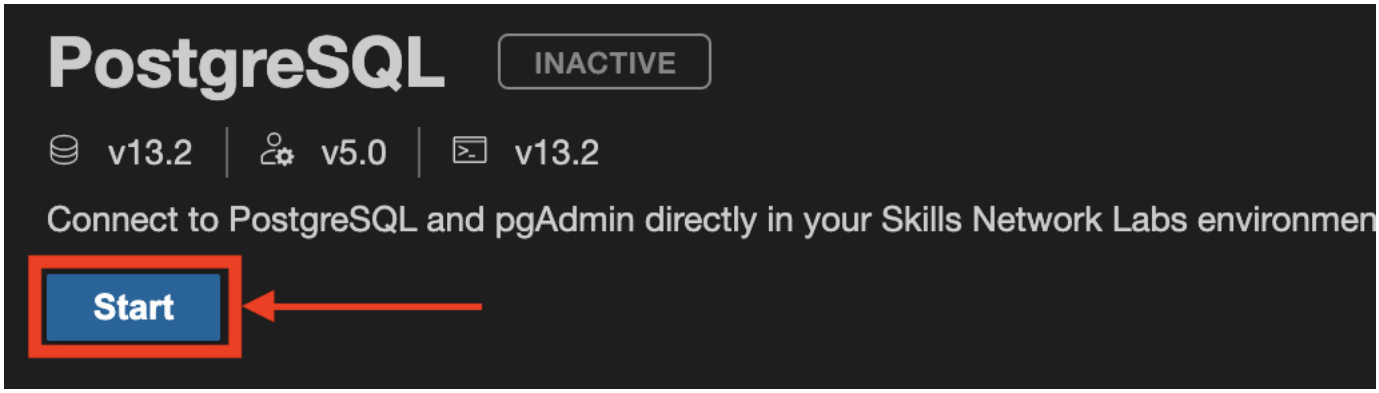
5. You will be prompted to confirm that you wish to delete this file. Select **OK** to confirm.



6. In the file explorer, you will see the `postgresql.conf` file you downloaded in Step 1 sitting in the root directory. Drag it into the **postgres > data** directory, as shown below.



7. Now go ahead and start up the PostgreSQL server again by selecting the **Start** button.

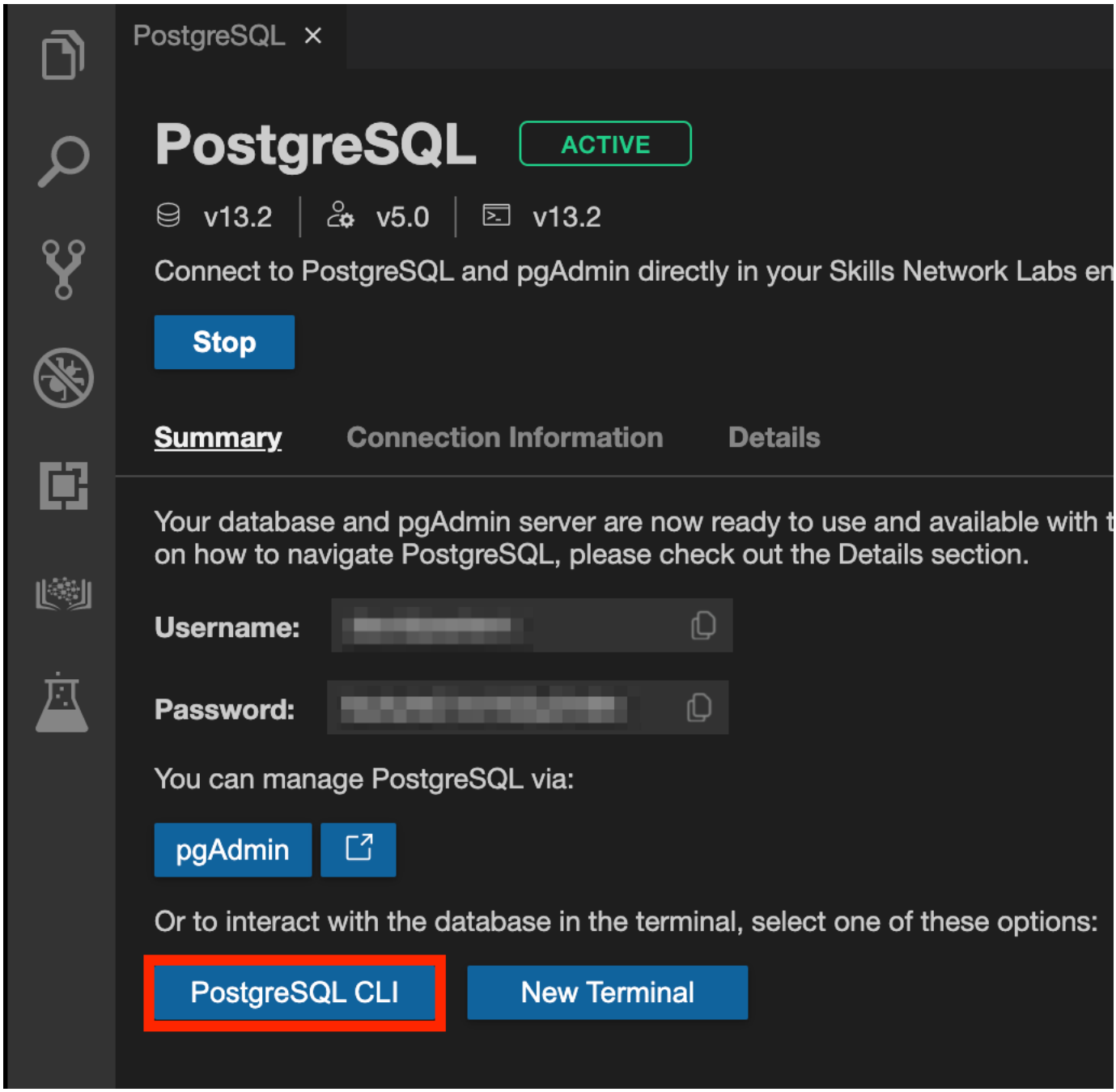


Task B: Test the Performance of the Server

In this part of the exercise, you will run a few SQL commands and analyze the server's performance, inspect the error logs, then finally, identify and resolve issues that could be hindering the performance of the database.

Let's try running some queries on the database and analyze its performance.

1. First, open up the PostgreSQL command line interface (CLI) by selecting the **PostgreSQL CLI** button.



2. **Try it yourself:** Use the CLI to connect to the **demo** database.

   ▶ Solution (click here)

3. To inspect how long each query or command takes, enable the timer with the following command in the CLI:

```
1. 1
1. \timing
```
Copied!

   This will tell you how long each query takes (in milliseconds).

4. Let's start off with a very simple query on the **aircrafts_data** table. Enter the following into the CLI:

```
1. 1
1. SELECT * FROM aircrafts_data;
```
Copied!

```
demo=# SELECT * FROM aircrafts_data;
 aircraft_code |            model             | range
---------------+------------------------------+-------
 773           | {"en": "Boeing 777-300"}     | 11100
 763           | {"en": "Boeing 767-300"}     |  7900
 SU9           | {"en": "Sukhoi Superjet-100"} |  3000
 320           | {"en": "Airbus A320-200"}    |  5700
 321           | {"en": "Airbus A321-200"}    |  5600
 319           | {"en": "Airbus A319-100"}    |  6700
 733           | {"en": "Boeing 737-300"}     |  4200
 CN1           | {"en": "Cessna 208 Caravan"} |  1200
 CR2           | {"en": "Bombardier CRJ-200"} |  2700
(9 rows)

Time: 1.048 ms
demo=#
```

As you can see, this query was on a small table and was quick--only about 1 millisecond. No problems here.

5. Let's try something a little more computationally heavy and see how the server handles it. The following command goes through each element in the **boarding_passes** table and reassigns each value to itself. In other words, it does not change the table but allows you to see how the server handles this task. Enter the following into the CLI:

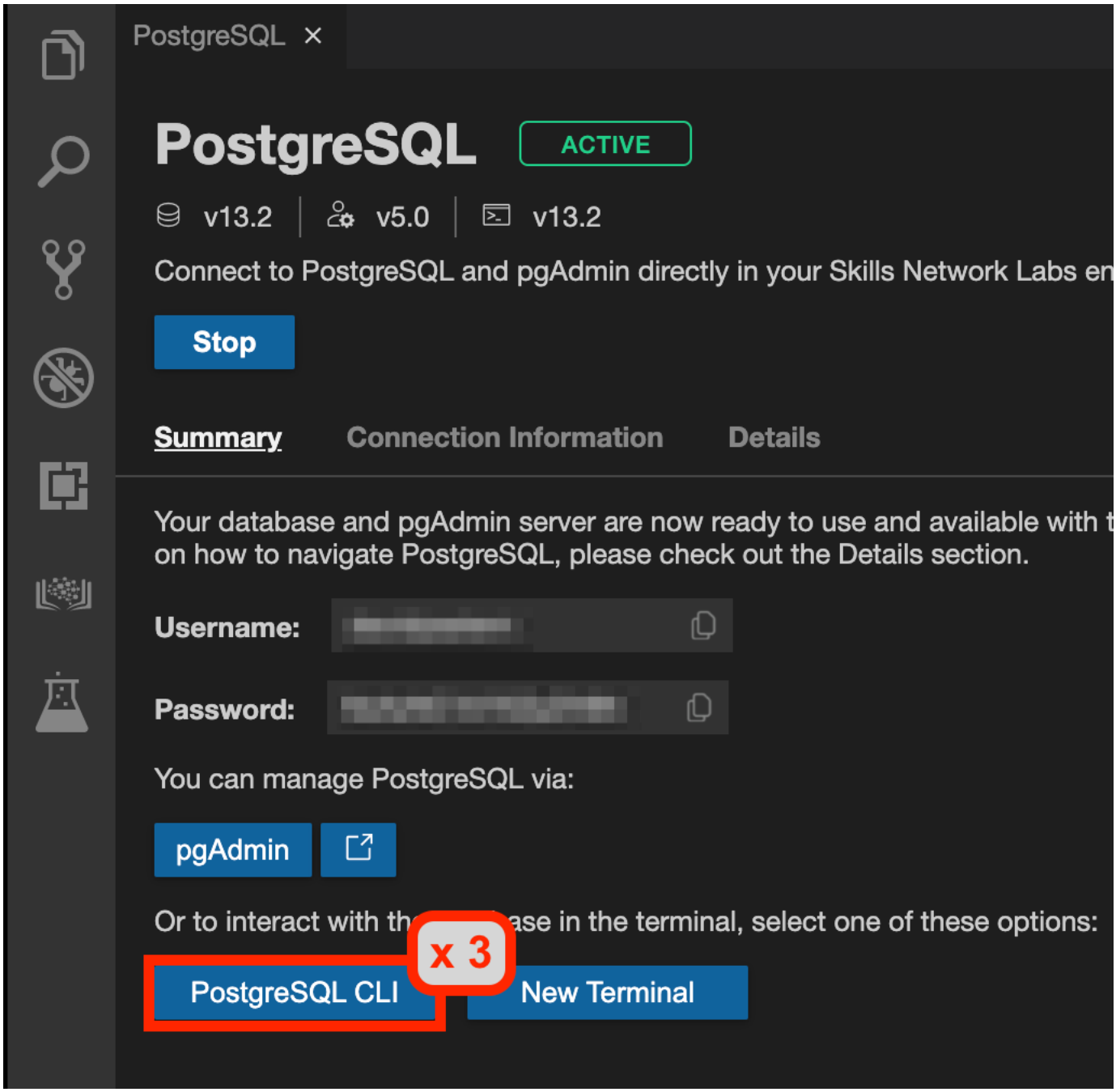
```
1. 1
1. UPDATE boarding_passes SET ticket_no = ticket_no, flight_id = flight_id, boarding_no = boarding_no, seat_no = seat_no;
```

Copied!

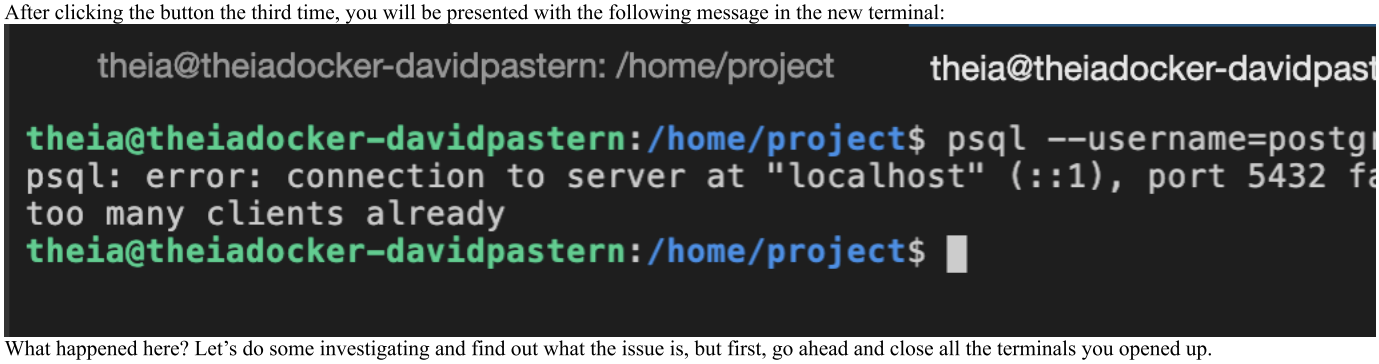
```
demo=# UPDATE boarding_passes SET ticket_no = ticket_no, flight_id =
 boarding_no = boarding_no, seat_no = seat_no;
UPDATE 579686
Time: 57946.030 ms (00:57.946)
demo=#
```

This heavier command took almost a minute to execute–a fairly long time, but the server was nonetheless able to complete the command. Still, you may want to improve this performance.

6. Now, as the database administrator, you will likely not be the *only* one who needs to access the database you are working with. Other users will likely need to connect to the database for a wide variety of reasons, including retrieving and inputting data. Let's simulate additional users connecting to the database. You can do this by opening additional **PostgreSQL CLI** terminals in Cloud IDE, as each one establishes a new connection to the server. Click **PostgreSQL CLI** three times, opening three new CLI terminals:



After clicking the button the third time, you will be presented with the following message in the new terminal:



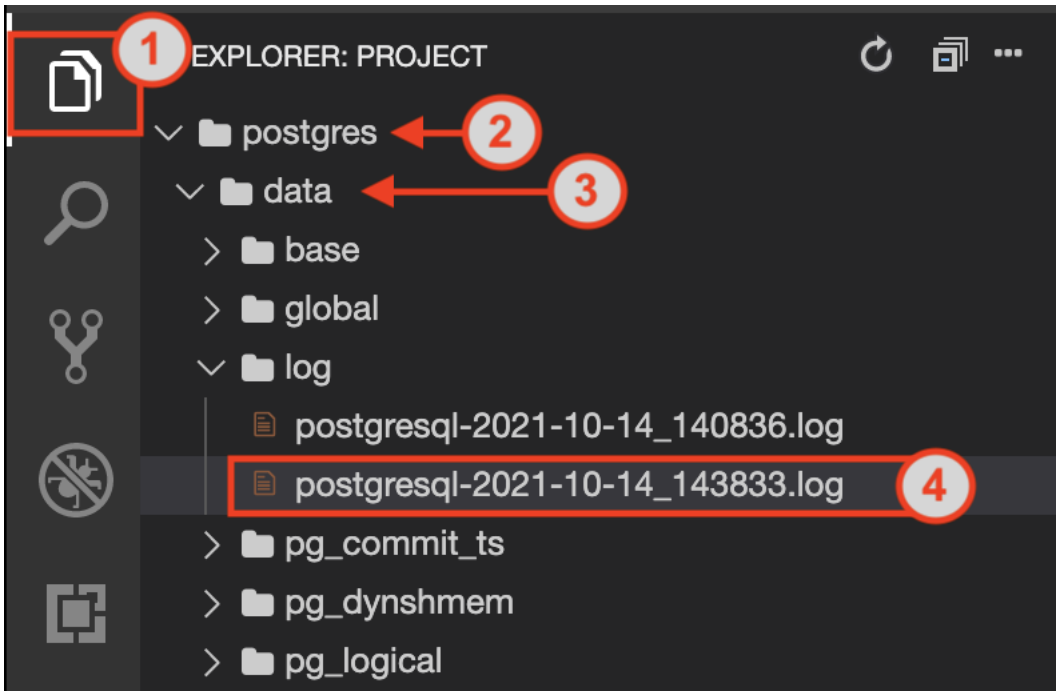
What happened here? Let's do some investigating and find out what the issue is, but first, go ahead and close all the terminals you opened up.

Exercise 4: Troubleshoot

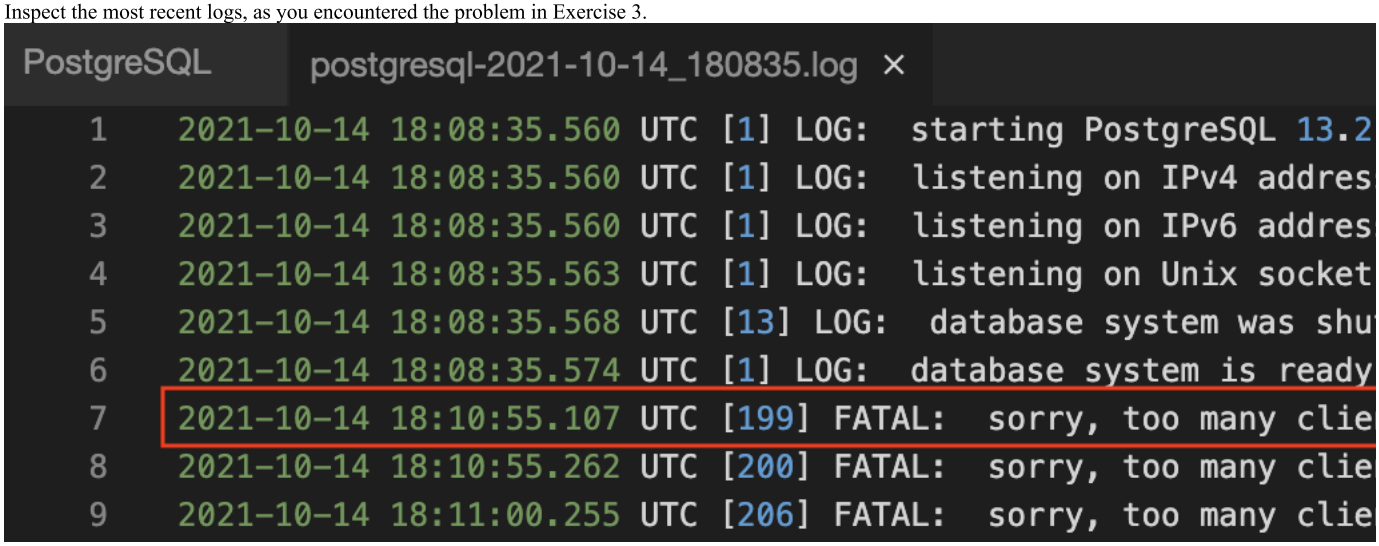
In the previous exercise, you encountered a problem and the server shut down. Now it's time to figure out what happened, why it happened, and how to fix it so that it does not happen again.

Task A: Diagnose the Issue

1. First, let's check the server logs to see what happened. Open up the Cloud IDE file explorer and navigate to **postgres > data > log**.
2. Since you restarted the server in the previous exercise, a new log file will have been created for this new session. Open up the most recent one.



3. Inspect the most recent logs, as you encountered the problem in Exercise 3.

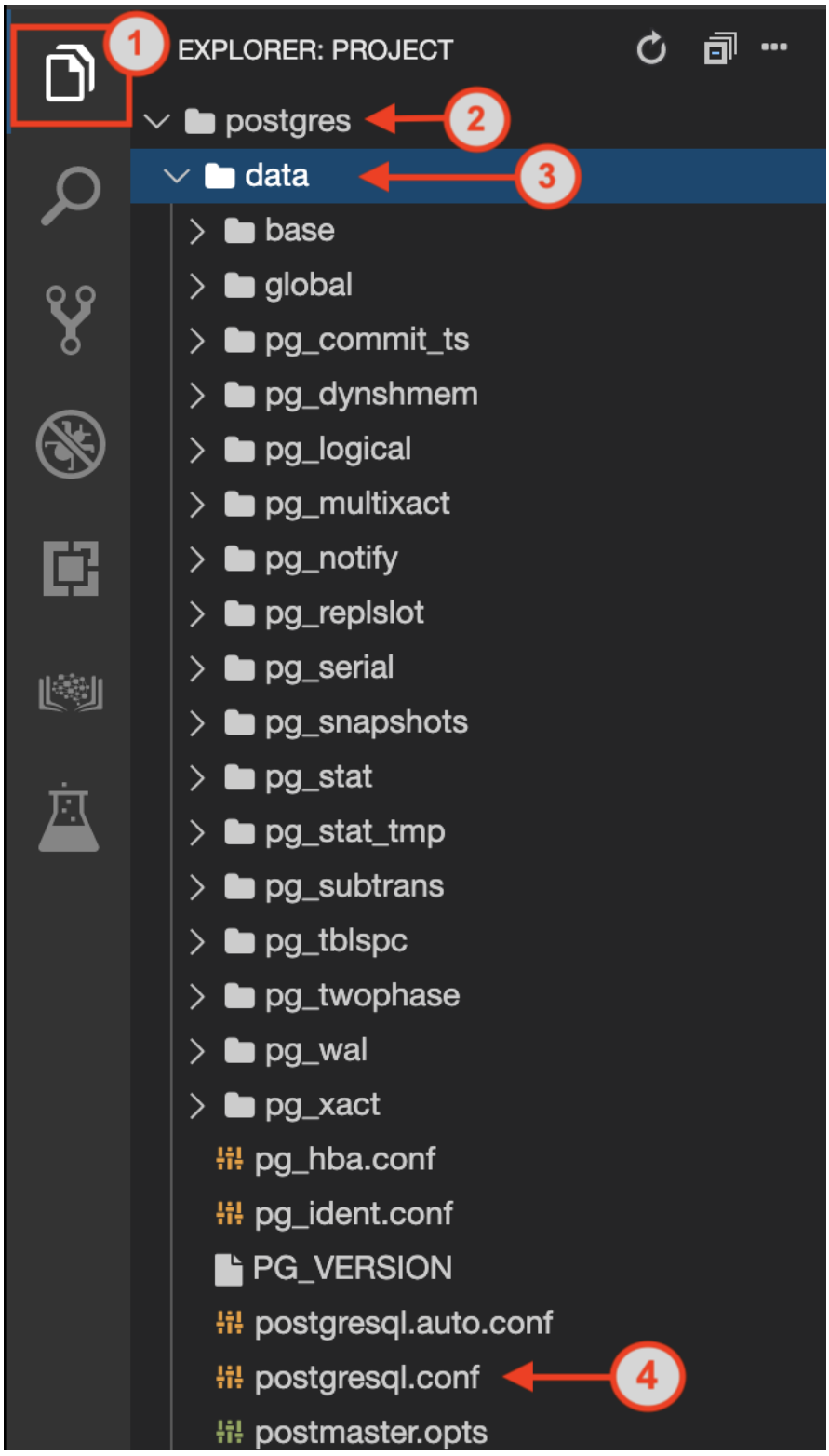


As you can see, some error logs were created from opening that last CLI terminal, with the message `FATAL: sorry, too many clients already`. This message is repeated several times as the connection is repeatedly attempting to re-establish.

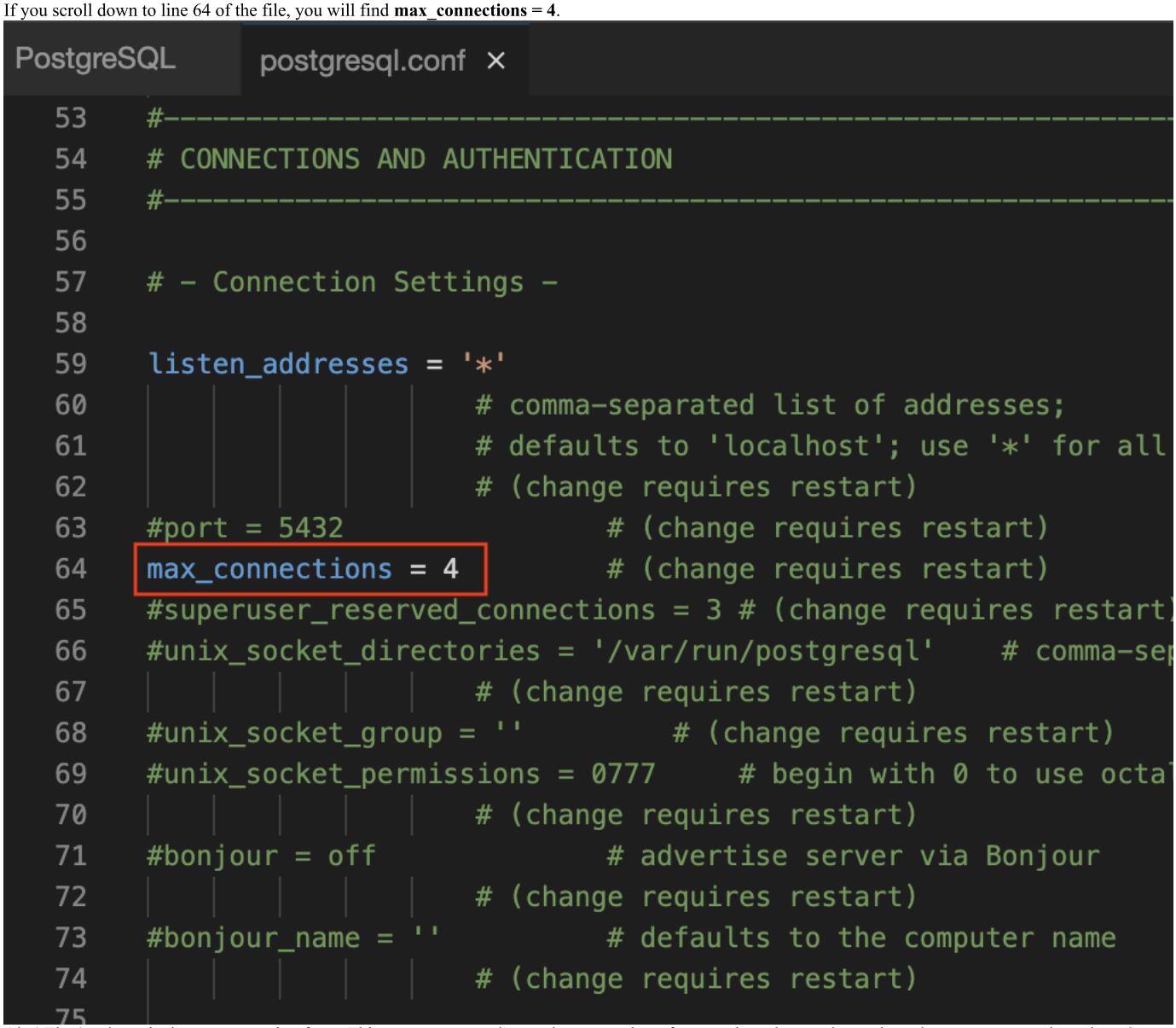
Some of the most common connectivity problems are not being able to connect to the database server, the database server or instance not running properly, and client login credentials being incorrect. You can likely rule out the last two, since the login credentials are automatically inputted for us on Cloud IDE and you know that the server instance is running properly, since you are already connected to it on 3 other terminals. This likely means you could be experiencing some problems connecting to the database server when you open the fourth connection. But why is this?

Server configuration issues, such as inadequate hardware resources or misconfigured settings, can significantly impact performance. Perhaps this could explain the connection problem as well as the slow performance you saw on the database query in Exercise 3. Let's take a look at the server configuration and see if you can spot anything.

4. Using the Cloud IDE file explorer, navigate to **postgres > data** and open the **postgresql.conf** configuration file.


1
EXPLORER: PROJECT
postgres
2
data
3
base
global
pg_commit_ts
pg_dynshmem
pg_logical
pg_multixact
pg_notify
pg_replslot
pg_serial
pg_snapshots
pg_stat
pg_stat_tmp
pg_subtrans
pg_tblspc
pg_twophase
pg_wal
pg_xact
pg_hba.conf
pg_ident.conf
PG_VERSION
postgresql.auto.conf
postgresql.conf
4
postmaster.opts

5. If you scroll down to line 64 of the file, you will find **max_connections = 4**.



Aha! That's where the issue was coming from. This parameter sets the maximum number of connections that can be made to the server at any given time. So when you tried to open that fourth CLI terminal, the max number of connections was reached, giving that FATAL error in the logs. Therefore, the problem you encountered comes from improper server configuration, since it's reasonable to expect more than four users to be connected to the database. Let's go ahead and fix the issue.

Task B: Resolve the Issue

In Task A, you discovered that the issues you encountered in Exercise 3 were caused by improper server configuration. Now let's modify the configuration parameters to resolve the issue.

1. With the **postgresql.conf** file open, change the **max_connections** parameter from 4 to 100. A maximum connections of 100 is a standard value that will support more than enough connections for most applications.

PostgreSQL postgresql.conf

```
53  #------------------------------------------------------------------
54  # CONNECTIONS AND AUTHENTICATION
55  #------------------------------------------------------------------
56
57  # - Connection Settings -
58
59  listen_addresses = '*'
60                      # comma-separated list of addresses;
61                      # defaults to 'localhost'; use '*' for all
62                      # (change requires restart)
63  #port = 5432                # (change requires restart)
64  max_connections = 100       # (change requires restart)
65  #superuser_reserved_connections = 3 # (change requires restart)
66  #unix_socket_directories = '/var/run/postgresql'    # comma-sep
67                      # (change requires restart)
68  #unix_socket_group = ''         # (change requires restart)
69  #unix_socket_permissions = 0777     # begin with 0 to use octal
70                      # (change requires restart)
71  #bonjour = off              # advertise server via Bonjour
72                      # (change requires restart)
73  #bonjour_name = ''          # defaults to the computer name
74                      # (change requires restart)
75
```

That should fix the issue you encountered when opening those additional CLI terminals.

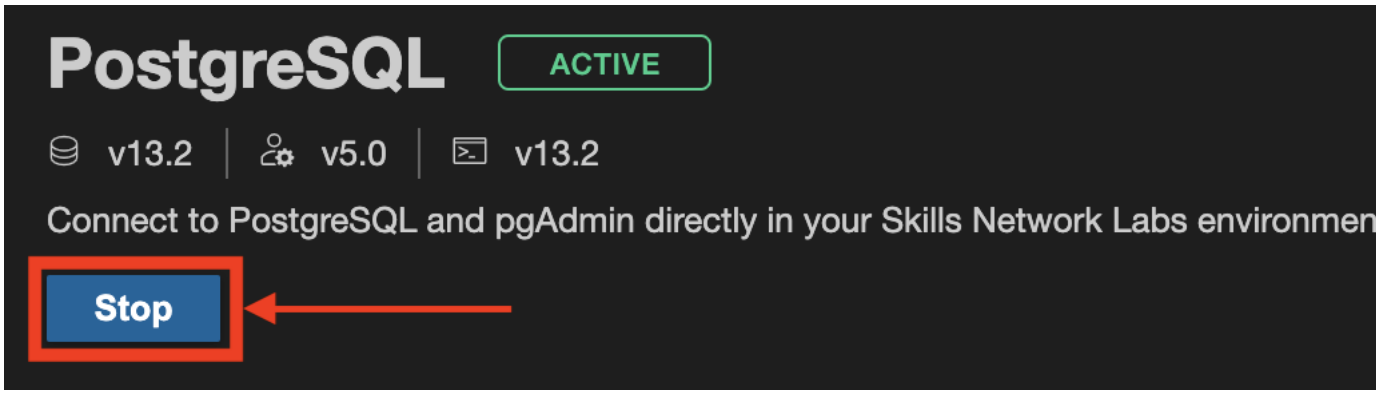
2. Since the server can now support far more connections than before, it will also need more available memory to support these connections. The **shared_buffers** configuration parameter sets the amount of memory the database server has at its disposal for shared memory buffers. Scroll down to line 121 to find the **shared_buffers** parameter.

PostgreSQL postgresql.conf

```
115  #------------------------------------------------------------------
116  # RESOURCE USAGE (except WAL)
117  #------------------------------------------------------------------
118
119  # - Memory -
120
121  shared_buffers = 128kB          # min 128kB
122                                  # (change requires restart)
123  #huge_pages = try               # on, off, or try
124                                  # (change requires restart)
125  #temp_buffers = 8MB             # min 800kB
126  #max_prepared_transactions = 0      # zero disables the featur
127                                  # (change requires restart)
128  # Caution: it is not advisable to set max_prepared_transaction
129  # you actively intend to use prepared transactions.
130  work_mem = 64kB                 # min 64kB
131  #hash_mem_multiplier = 1.0      # 1-1000.0 multiplier on hash
132  maintenance_work_mem = 1MB      # min 1MB
133  #autovacuum_work_mem = -1       # min 1MB, or -1 to use mainte
134  #logical_decoding_work_mem = 64MB   # min 64kB
135  #max_stack_depth = 2MB          # min 100kB
136  #shared_memory_type = mmap      # the default is the first opt
137                                  # supported by the operating system:
138                                  #   mmap
139                                  #   sysv
140                                  #   windows
141                                  # (change requires restart)
```

Notice that the parameter is set to 128kB, which is the minimum value.

3. Increase the available memory by changing the **shared_buffers** parameter from **128kB** to **128MB**.
4. While you're at it, you can also increase the server performance so that the slow query you executed in Exercise 3 will run more quickly. Increase the **work_mem** parameter from the minimum **64kB** to **4MB**.
5. Change the **maintenance_work_mem** from the minimum **1MB** to a more standard **64MB**.
6. Save the changes to `postgresql.conf` by either navigating to **File > Save** at the top toolbar or by pressing **Ctrl + S** (Mac: ⌘ + S).
7. Close all open terminal tabs and stop the PostgreSQL server by selecting the **Stop** button.

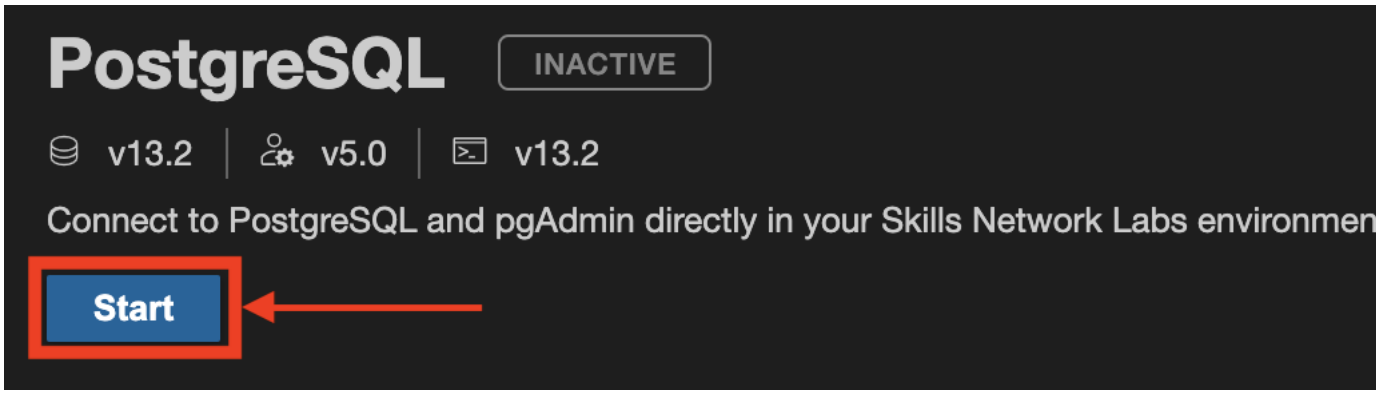


Exercise 5: Try it Yourself!

The changes you made to the PostgreSQL server configuration parameters should fix the problems you encountered in Exercise 3. However, it's certainly good practice to test this out and confirm that your fix was successful. In this practice exercise, you will run through much of the same process you did in Exercise 3 to confirm that the issues you encountered are resolved and will not arise again.

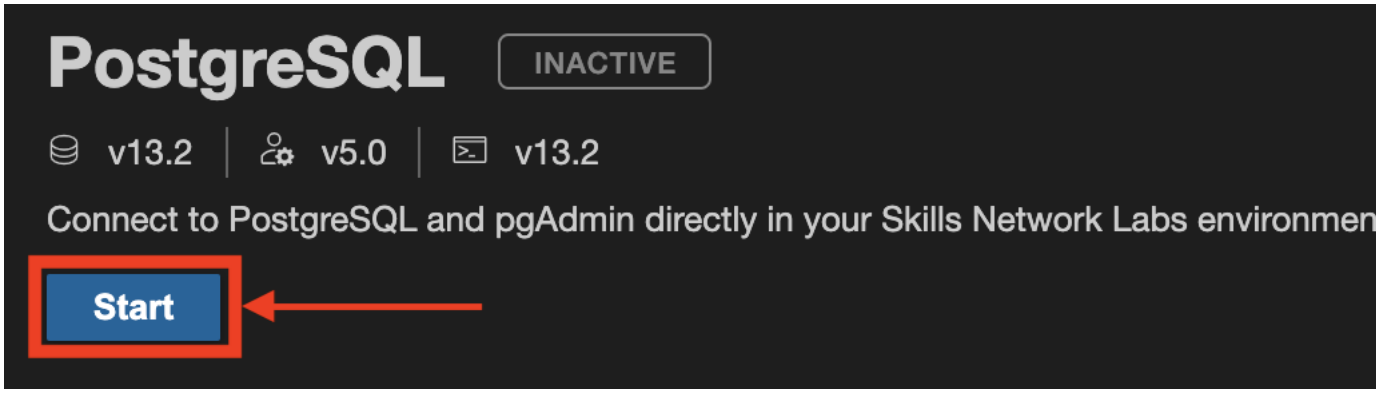
1. **Try it yourself:** Restart the PostgreSQL server.

   ▼ Solution (Click Here)
   As before, select the "Start" button to start the PostgreSQL server.

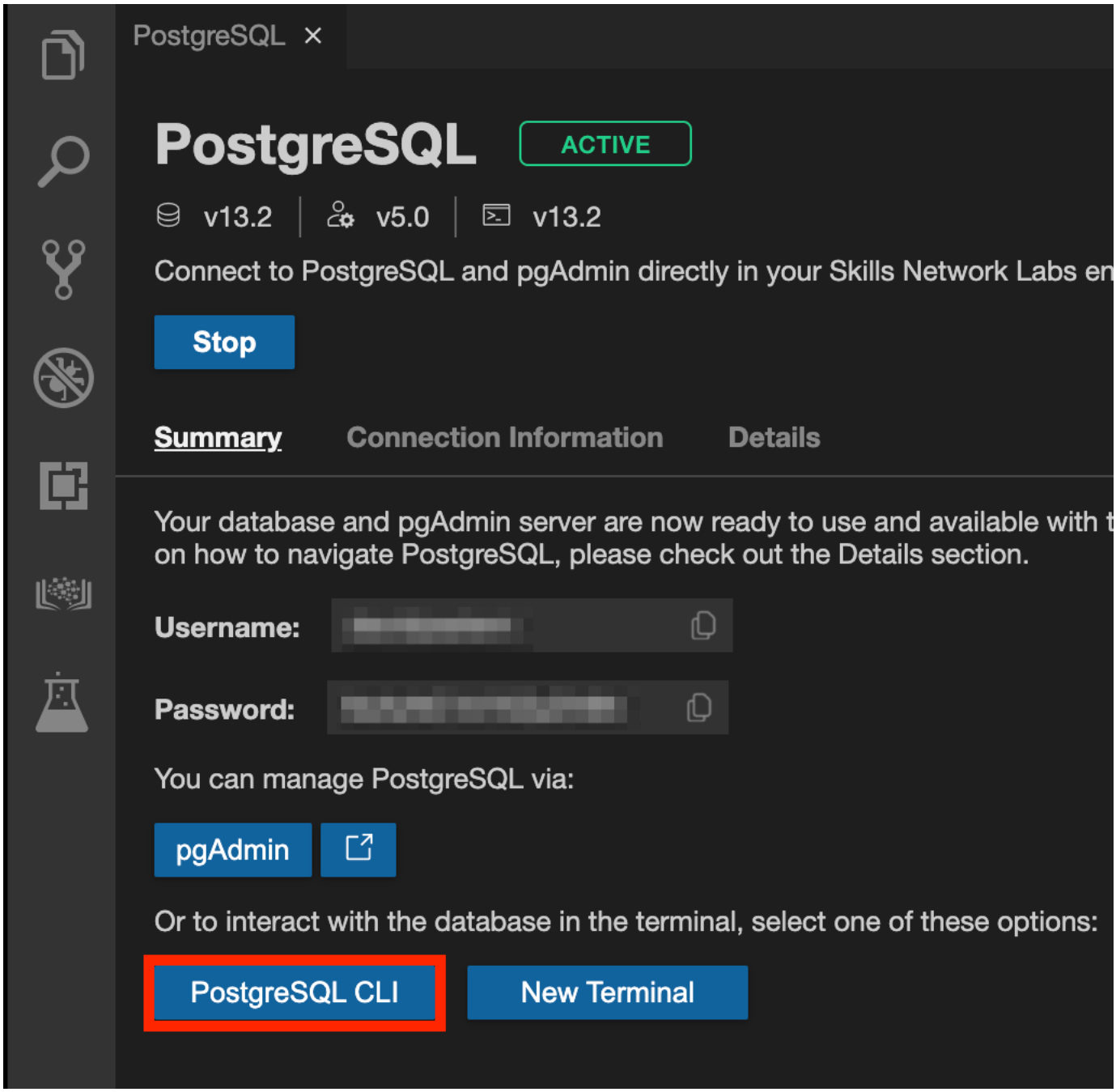


2. **Try it yourself:** Compare the performance of querying the **aircrafts_data** table now compared to before changing the configuration parameters.

   ▼ Hint (Click Here)
   You'll first need to open up the PostgreSQL CLI, connect to the **demo** database, and enable timing.



   ▼ Solution (Click Here)
   1. Open up a PostgreSQL CLI terminal.



2. Connect to the **demo** database by entering the following into the CLI:

```
1. 1
1. \connect demo
```

Copied!

3. Enable timing with the following command in the CLI:

```
1. 1
1. \timing
```

Copied!

4. Enter the following query into the CLI:

```
1. 1
1. SELECT * FROM aircrafts_data;
```

Copied!

```
demo=# SELECT * FROM aircrafts_data;
 aircraft_code |              model               | range
---------------+----------------------------------+-------
 773           | {"en": "Boeing 777-300"}         | 11100
 763           | {"en": "Boeing 767-300"}         |  7900
 SU9           | {"en": "Sukhoi Superjet-100"}    |  3000
 320           | {"en": "Airbus A320-200"}        |  5700
 321           | {"en": "Airbus A321-200"}        |  5600
 319           | {"en": "Airbus A319-100"}        |  6700
 733           | {"en": "Boeing 737-300"}         |  4200
 CN1           | {"en": "Cessna 208 Caravan"}     |  1200
 CR2           | {"en": "Bombardier CRJ-200"}     |  2700
(9 rows)

Time: 0.917 ms
```

As you can see, the query took less than 1 millisecond. Extremely quick, but fairly similar to the results before you changed the configuration parameters. This is because this query is on such a small table that the server didn't get close to the memory limits when executing it, so there was no issue with that query to begin with.

3. Run the same command in the CLI that you did in Step 5 of Exercise 3 and compare the performance before and after changing the configuration parameters. To save you the scrolling and losing your place, the command you entered earlier is given below:

```
1. 1
1. UPDATE boarding_passes SET ticket_no = ticket_no, flight_id = flight_id, boarding_no = boarding_no, seat_no = seat_no;
```

Copied!

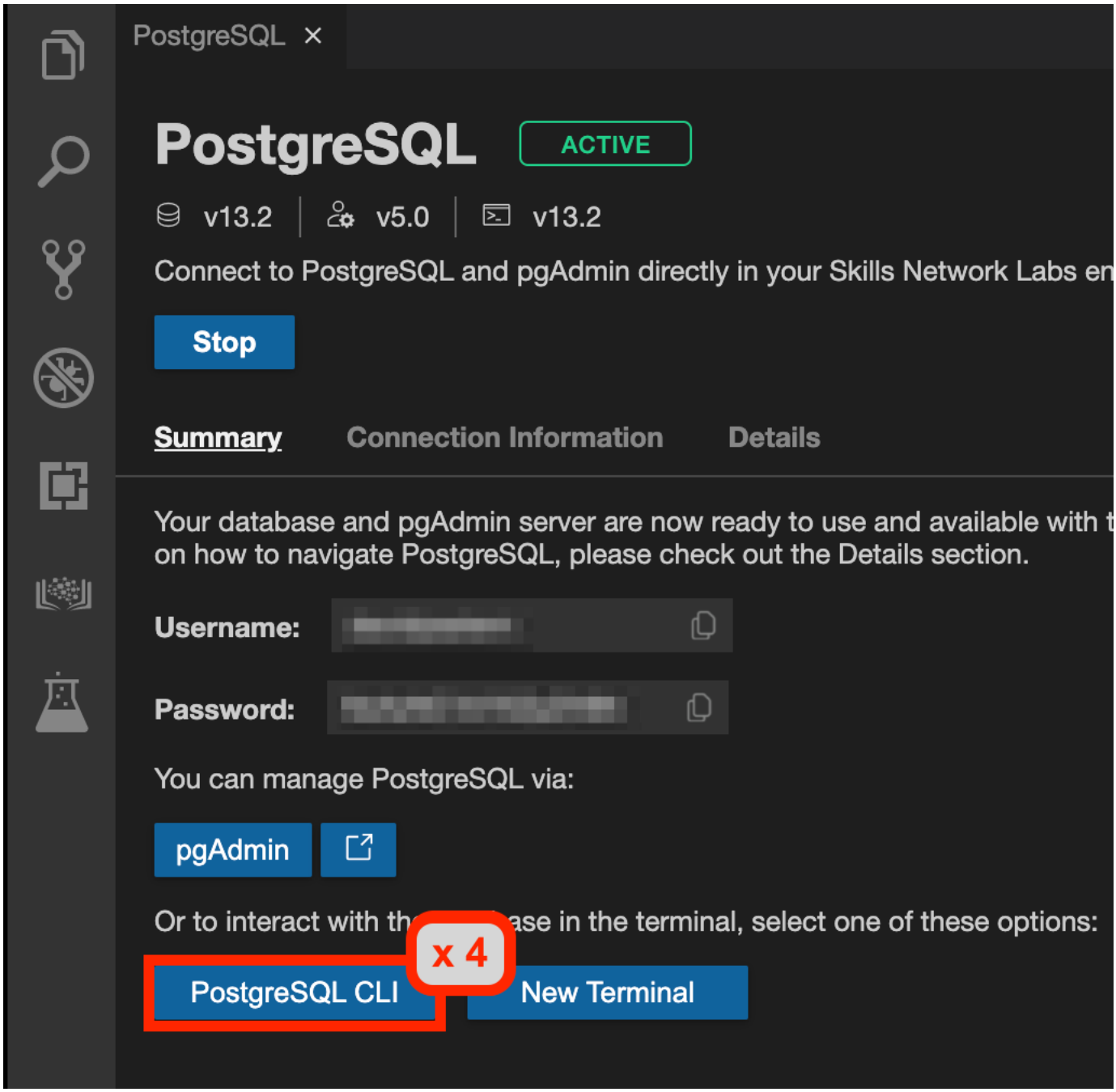
▼ Results (Click Here)

```
demo=# UPDATE boarding_passes SET ticket_no = ticket_no, flight_id =
boarding_no = boarding_no, seat_no = seat_no;
UPDATE 579686
Time: 11126.111 ms (00:11.126)
demo=#
```

After increasing the **shared_buffers** and **work_mem** parameters, you saw a significant improvement in performance! This command took only about 10 seconds to execute as opposed to close to a minute as before. By reconfiguring the server, you greatly improved server performance. Well done!

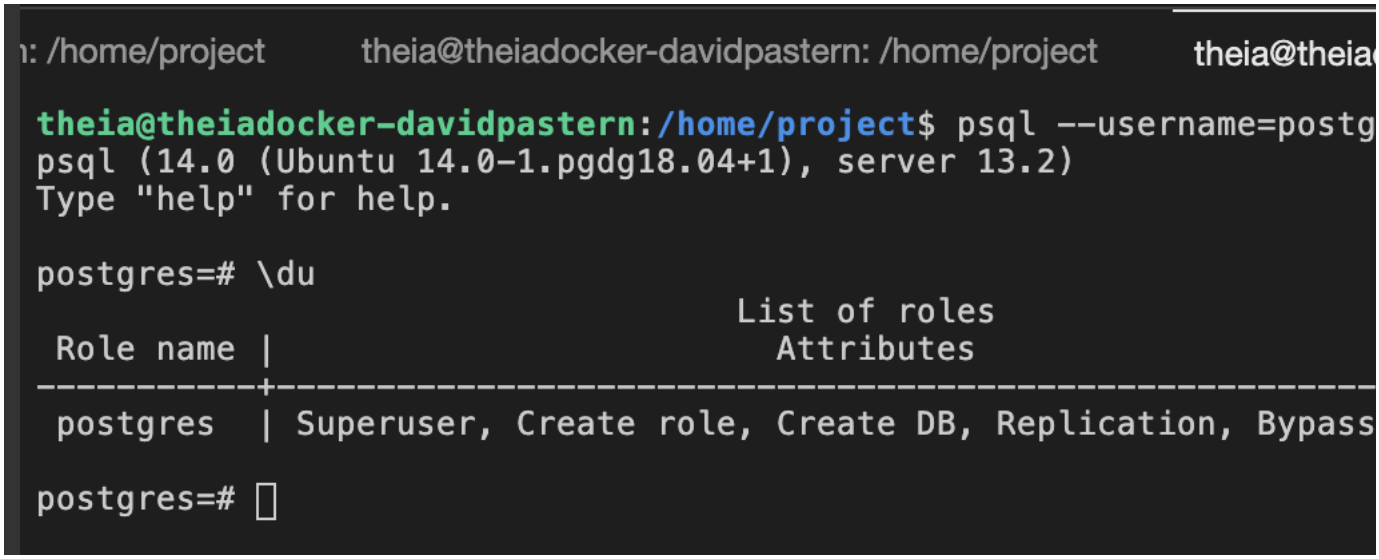
4. **Try it yourself:** Finally, test to confirm that the server can now handle *at least* 5 connections.

▼ Hint (Click Here)
Recall that opening additional PostgreSQL terminals constitute additional connections to the server.
▼ Solution (Click Here)
1. Click the "PostgreSQL CLI" button four times.



2. Notice that no error was raised on the fifth terminal and the CLI is still open.

3. You could even enter a test command, such as `\du` to confirm the CLI is working.



4. Furthermore, you could check the server logs to confirm that no error was raised and everything is running as intended.

Conclusion

Congratulations on completing this lab on troubleshooting a relational database management system. You now have some foundational knowledge on how to identify and resolve some common issues you may face in PostgreSQL as a database administrator.

Author

David Pasternak

Other Contributors

Rav Ahuja

Changelog

Date	Version	Changed by	Change Description
2021-10-12	0.1	David Pasternak	Initial version created
2022-07-12	0.2	Lakshmi Holla	Updated html tags
2023-05-08	0.3	Jaskomal Natt	Updated copyright date