



BDA DATA LAKE IMPLEMENTATION

ABSTRACT

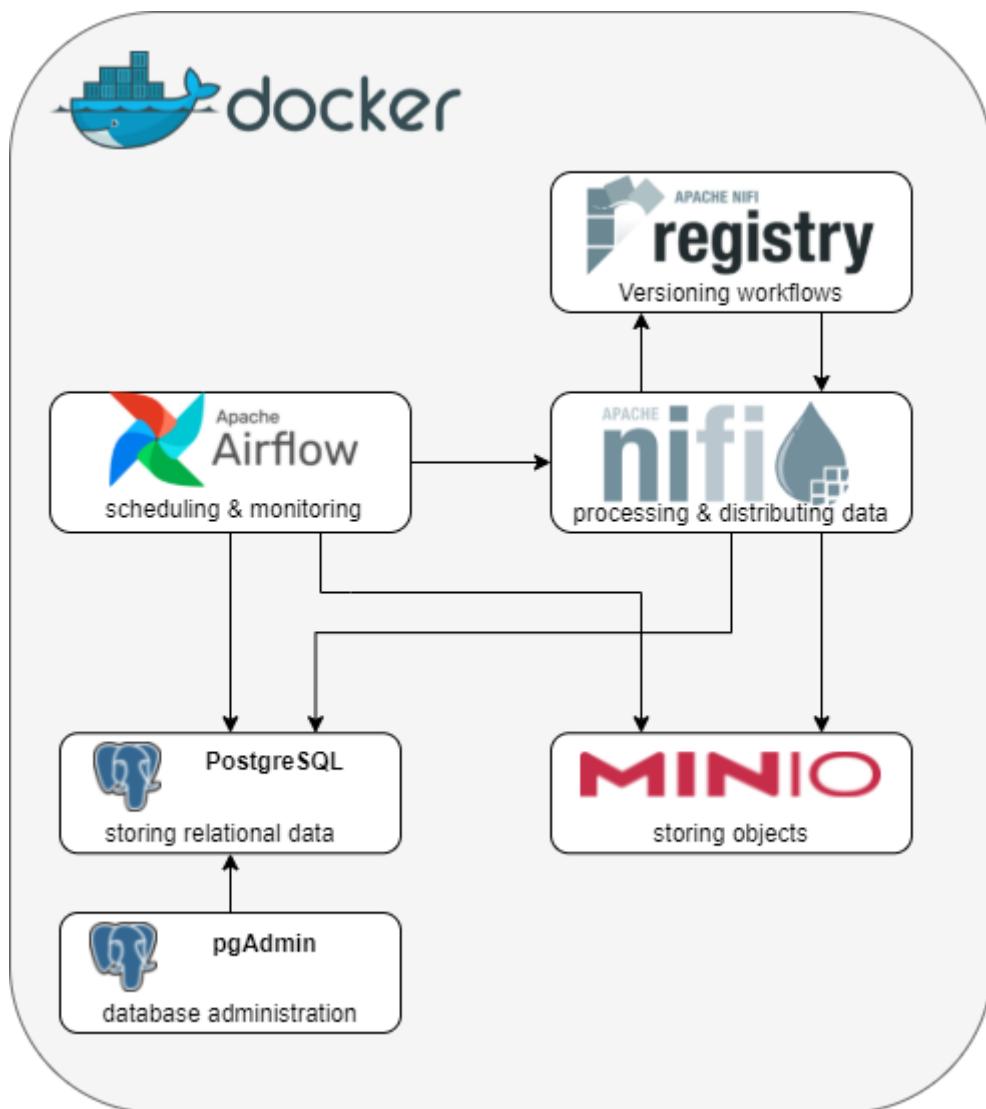
It's the implementation of Data Lake on docker with the following components: Apache Registry, Apache Airflow, Apache nifi, PostgreSQL, Minio, PgAdmin

Usman Khan
Big Data Analytics

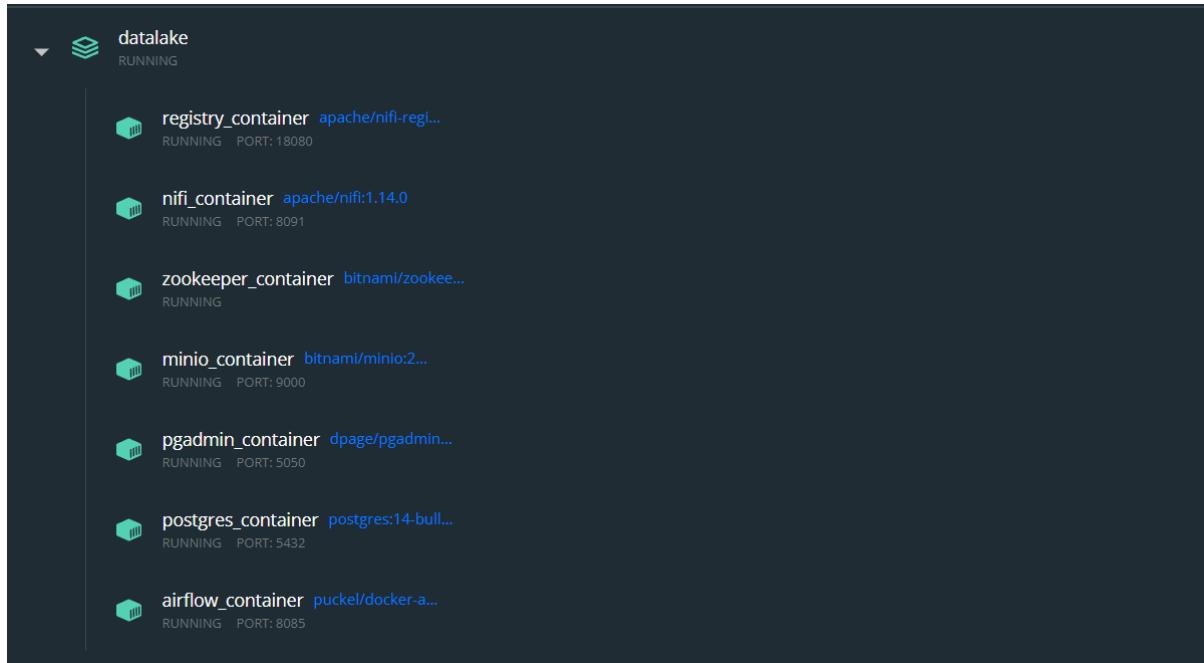
Contents

Setting up the pgAdmin	4
Setting up Apache NiFi.....	6
Apache NiFi: Hello postgres database.....	11
Processors.....	17
Apache NiFi: Hello MinIO	21
Configuring NiFi.....	24
Airflow	25
Apache Airflow: Hello postgres database.....	26
Apache Airflow: Hello Minio/S3	29

Architecture



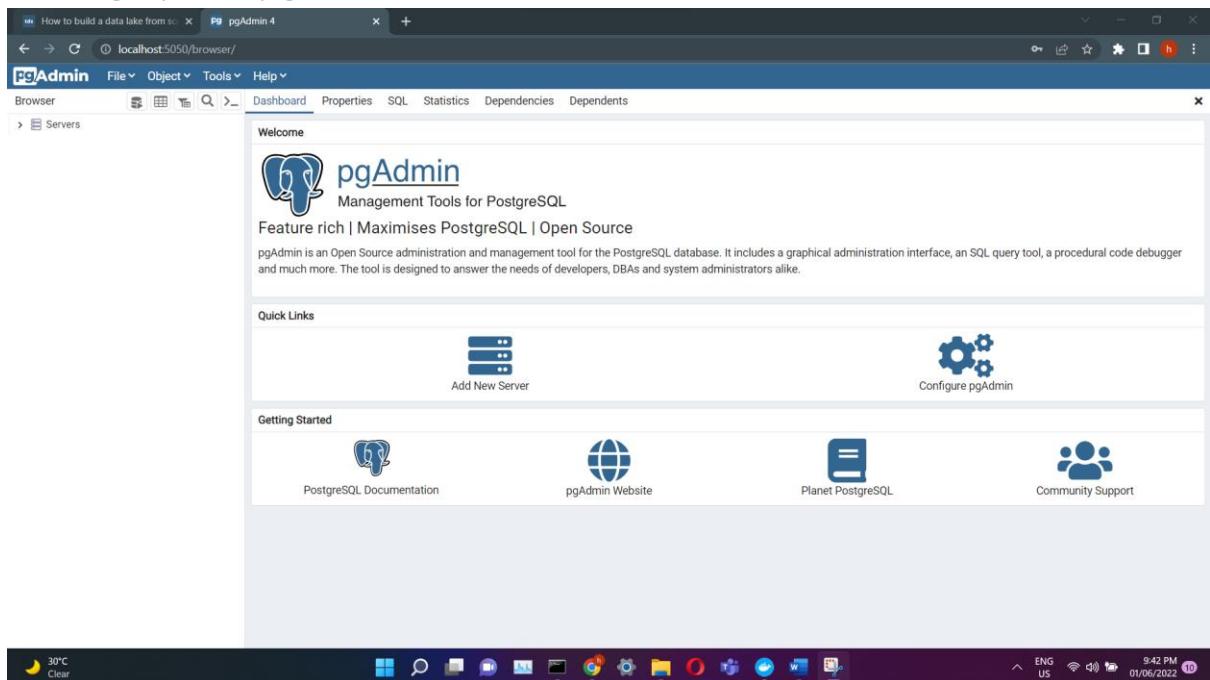
Docker desktop showing all containers are running after docker-compose up command:



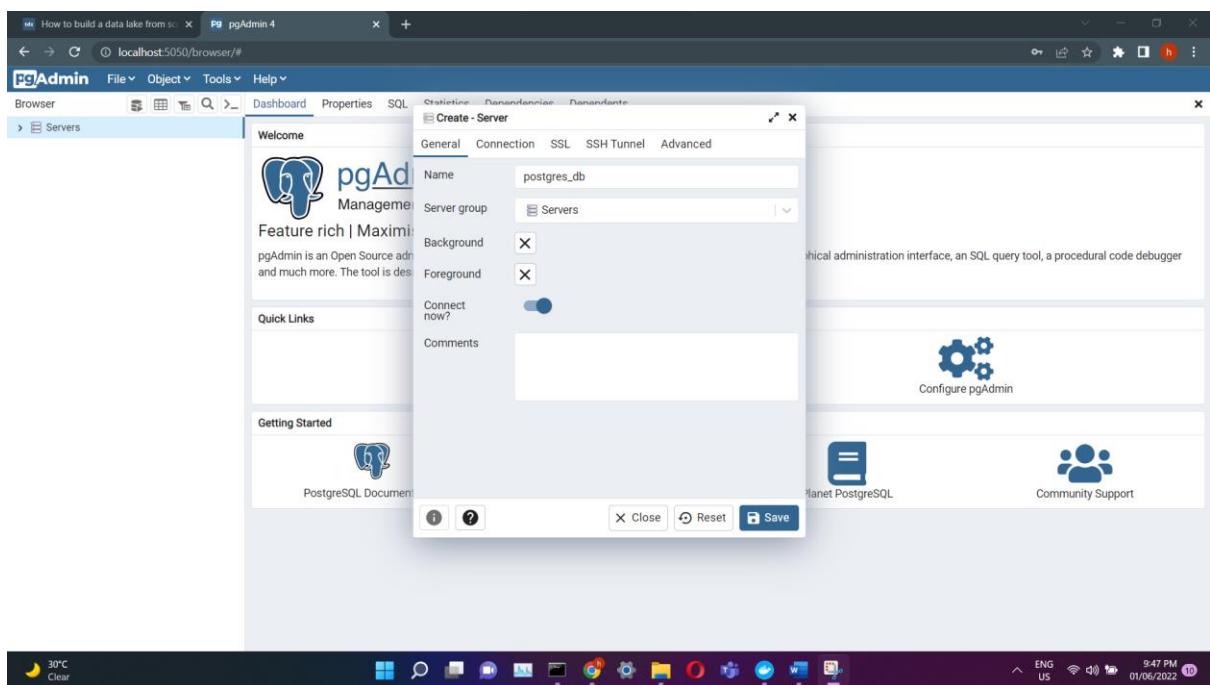
Can be used through these urls:

1. Apache NiFi — <http://localhost:8091/nifi/>
2. Apache NiFi Registry — <http://localhost:18080/nifi-registry/>
3. Apache Airflow — <http://localhost:8085/admin/>
4. pgAdmin — <http://localhost:5050/browser/>
5. minIO — <http://localhost:9000/>

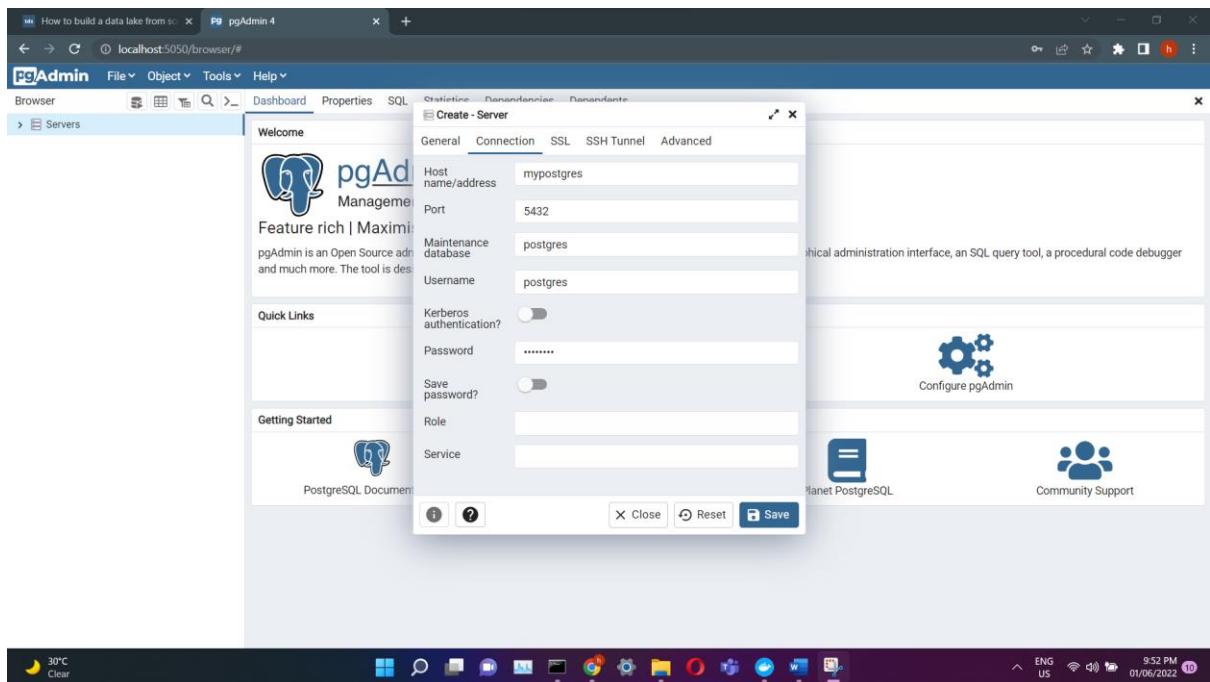
Setting up the pgAdmin



To create a server click on the top left corner on server and right click on server and then click on create to add new server.



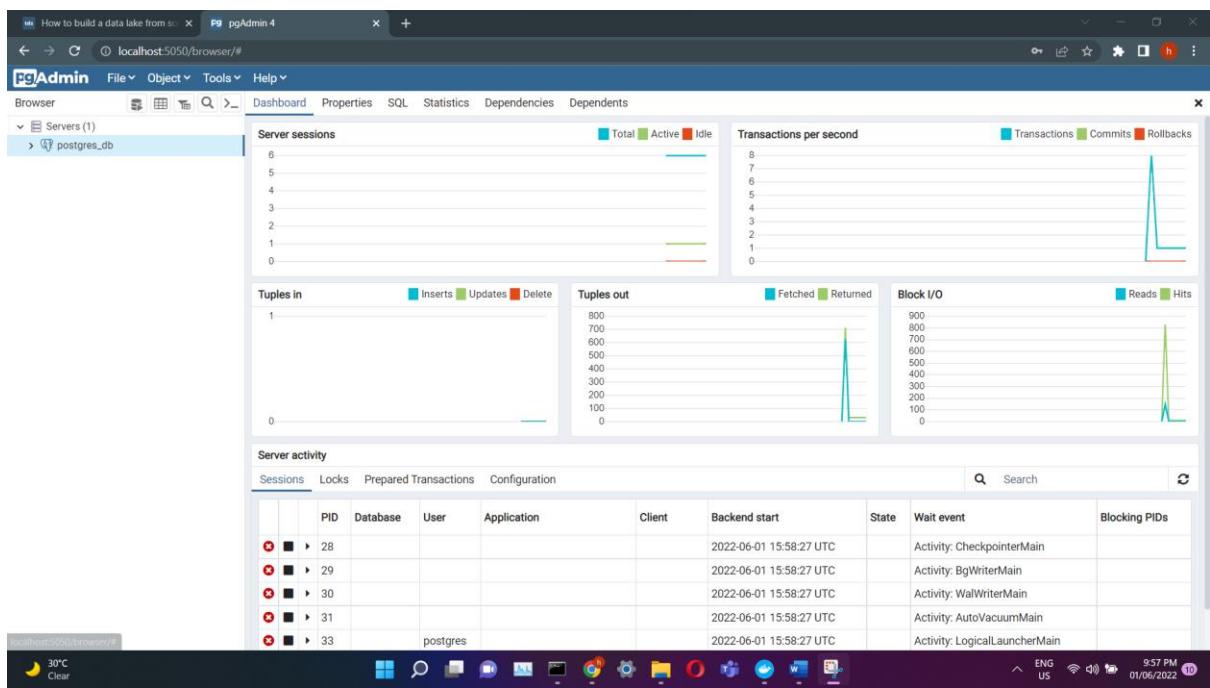
In general tab, set the name to “postgres_db”.



In the connection tab: we can just use the hostname “mypostgres” instead of a fixed IP-address and let docker take care of DNS resolving in the background.

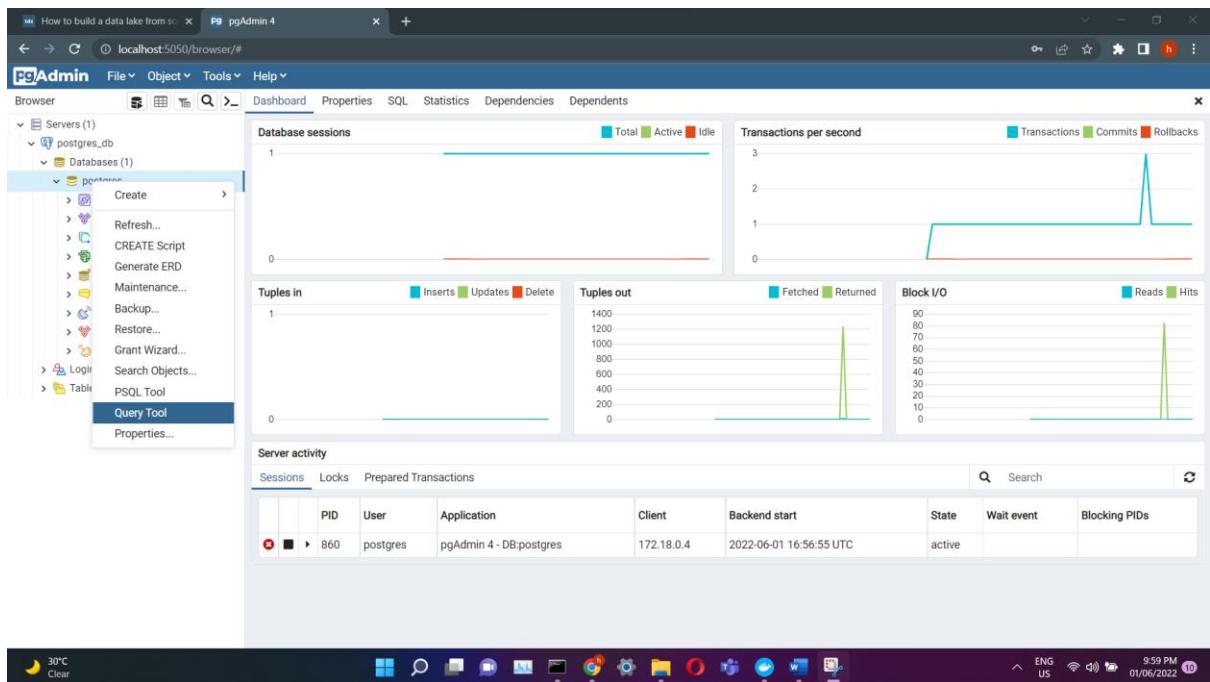
In the Connection tab: The port is the standard application port of the postgres database — 5432.

Set username and password to “postgres”

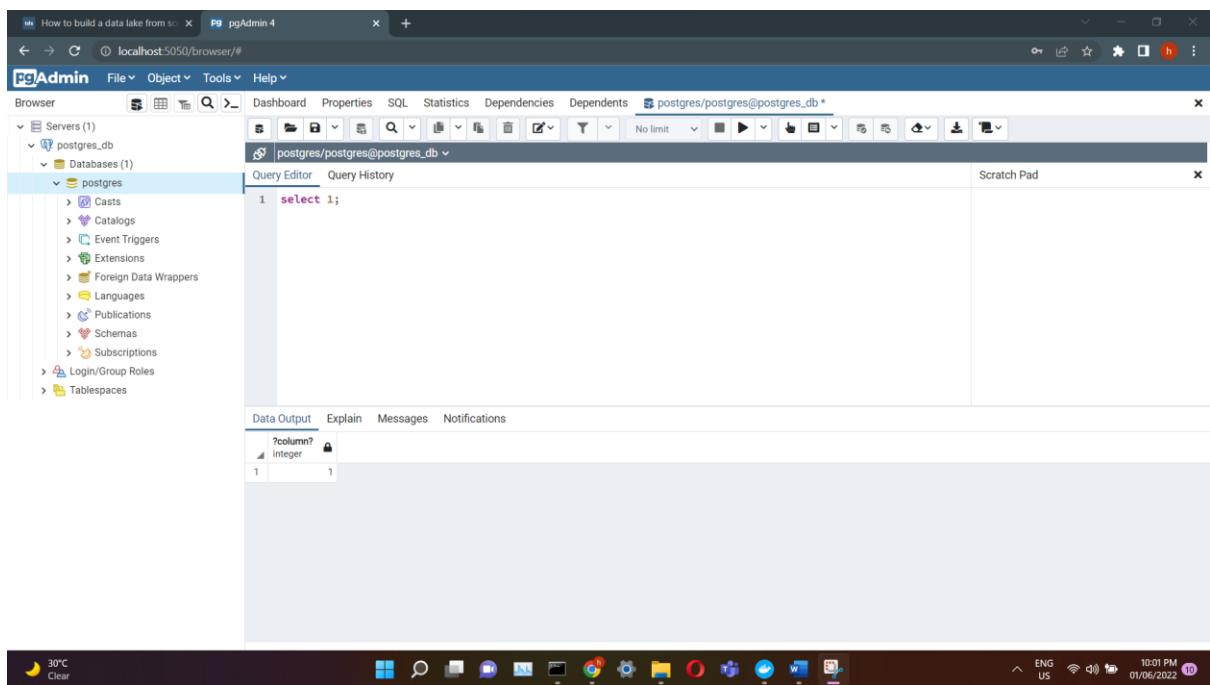


After saving at the previous step you will get this above screen.

After connecting to the database, we can now run queries. Right-click on the database and select Query Tool.



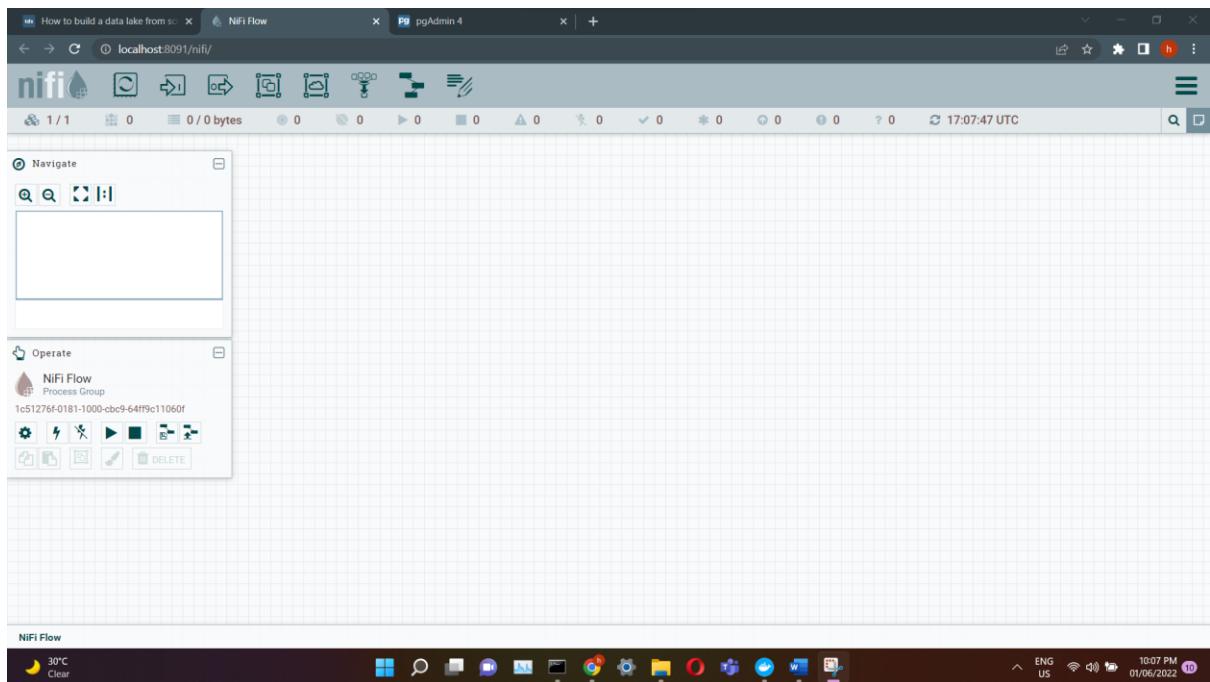
Run select 1



PgAdmin is now connected to the postgres database!

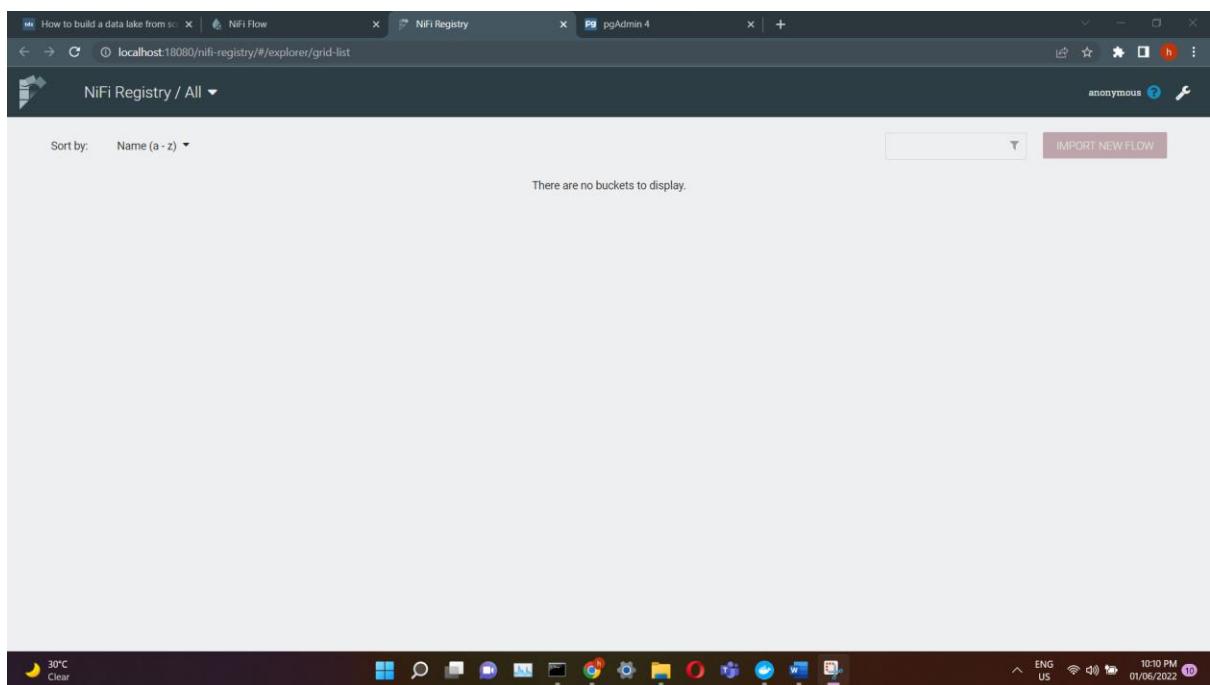
Setting up Apache NiFi.

Access Apache Nifi using the above-mentioned link: Home page of Apache Nifi.



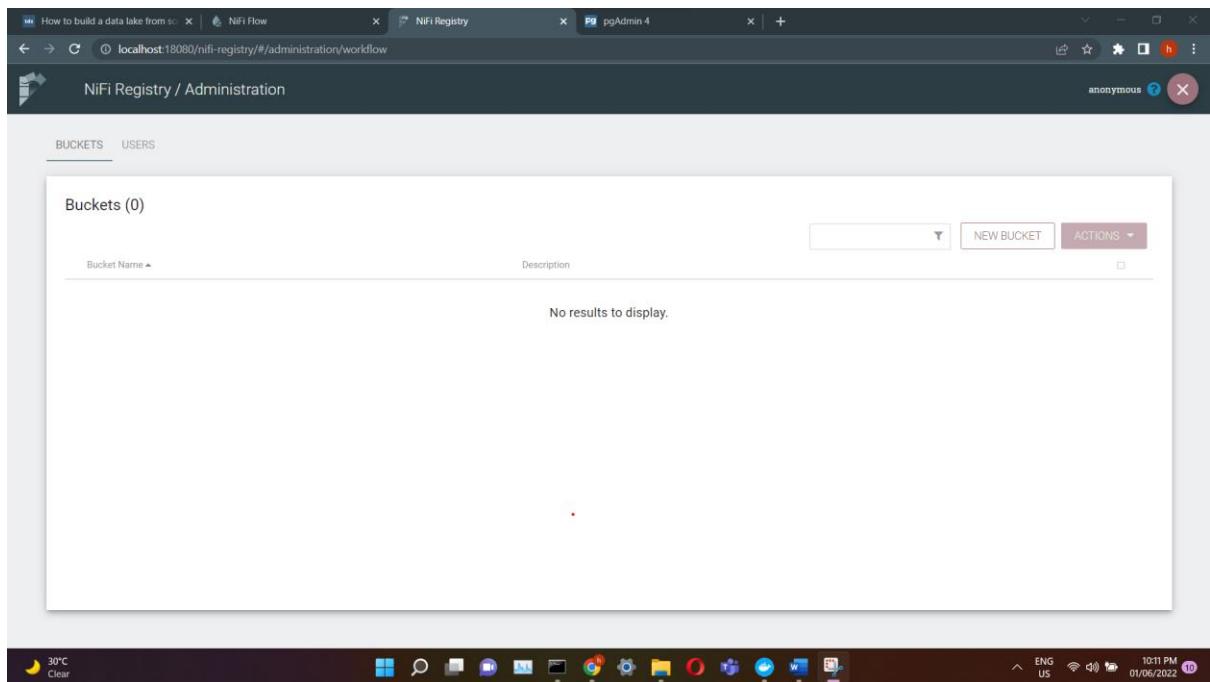
We want to connect to:

- the NiFi registry
- the postgres database
- MinIO

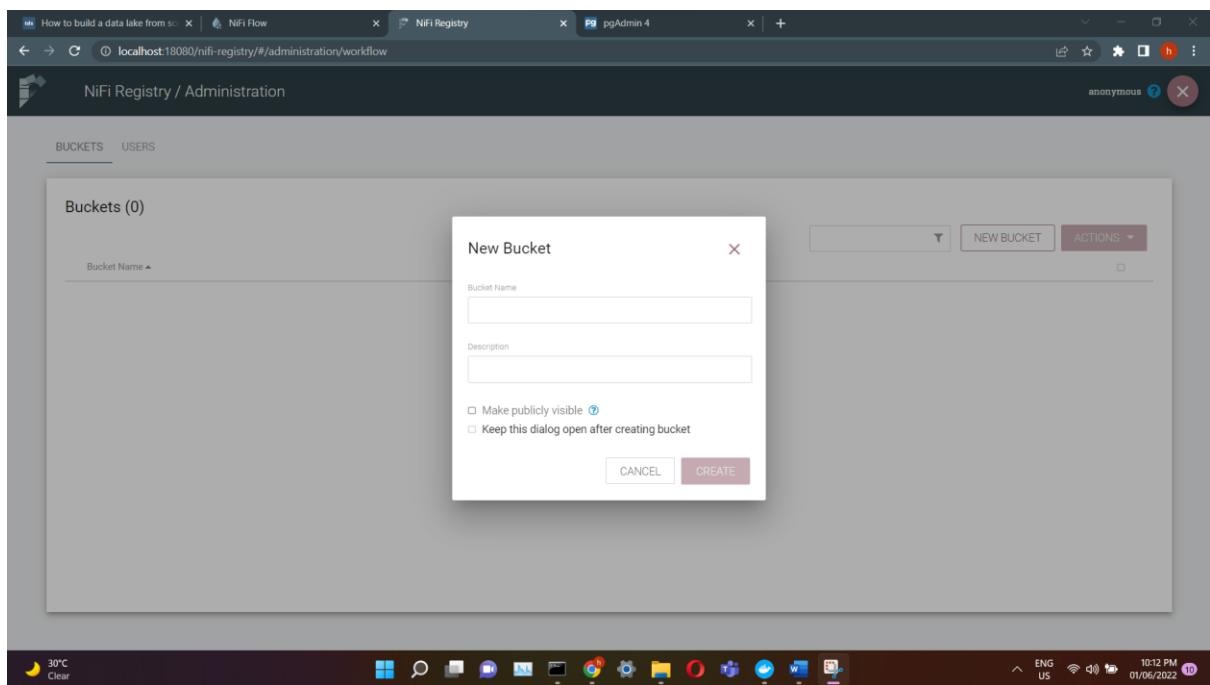


To connect NiFi to the registry, we first need to create a “bucket” in registry to store and organize our data flows.

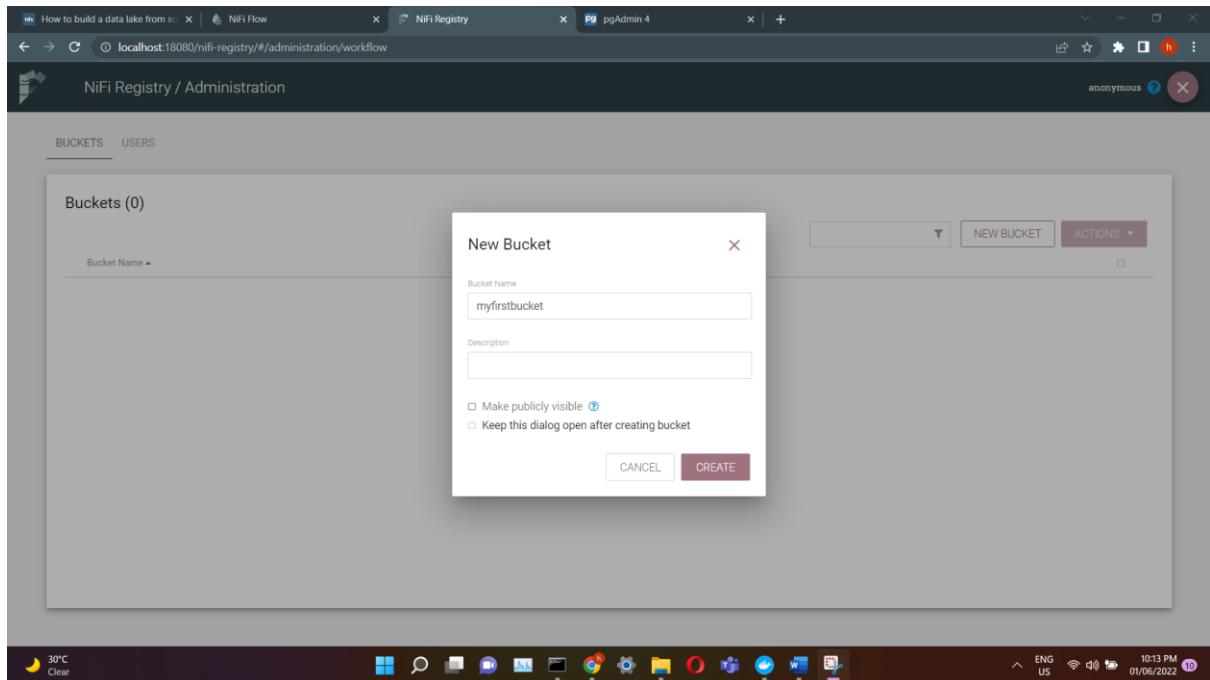
Click on the wrench-symbol in the top right corner of the window.



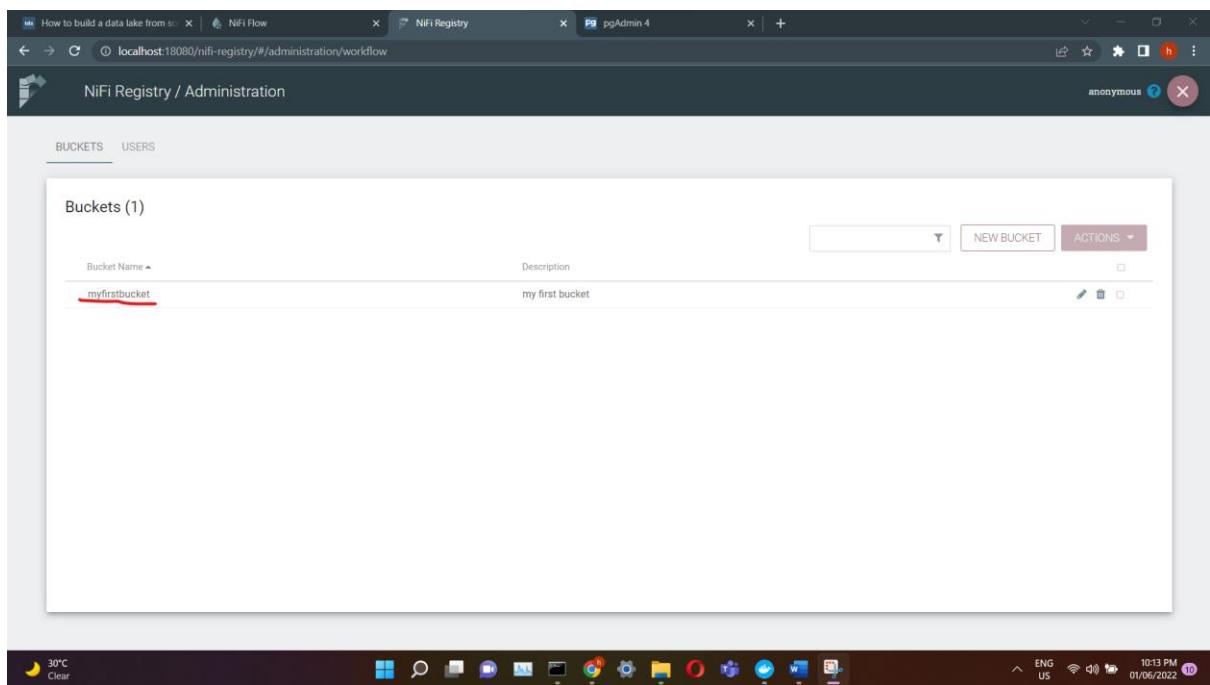
Click on “NEW BUCKET” on the right side.



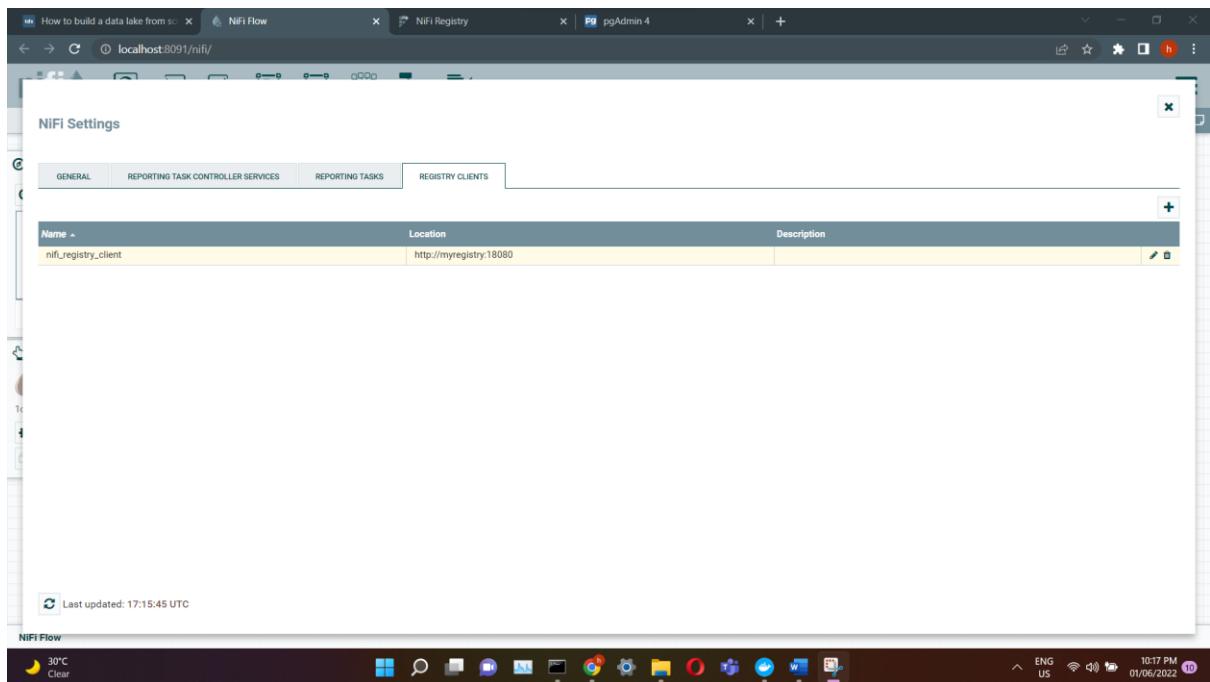
Enter a name for you bucket, for instance myfirstbucket.



The image after the creating the bucket.



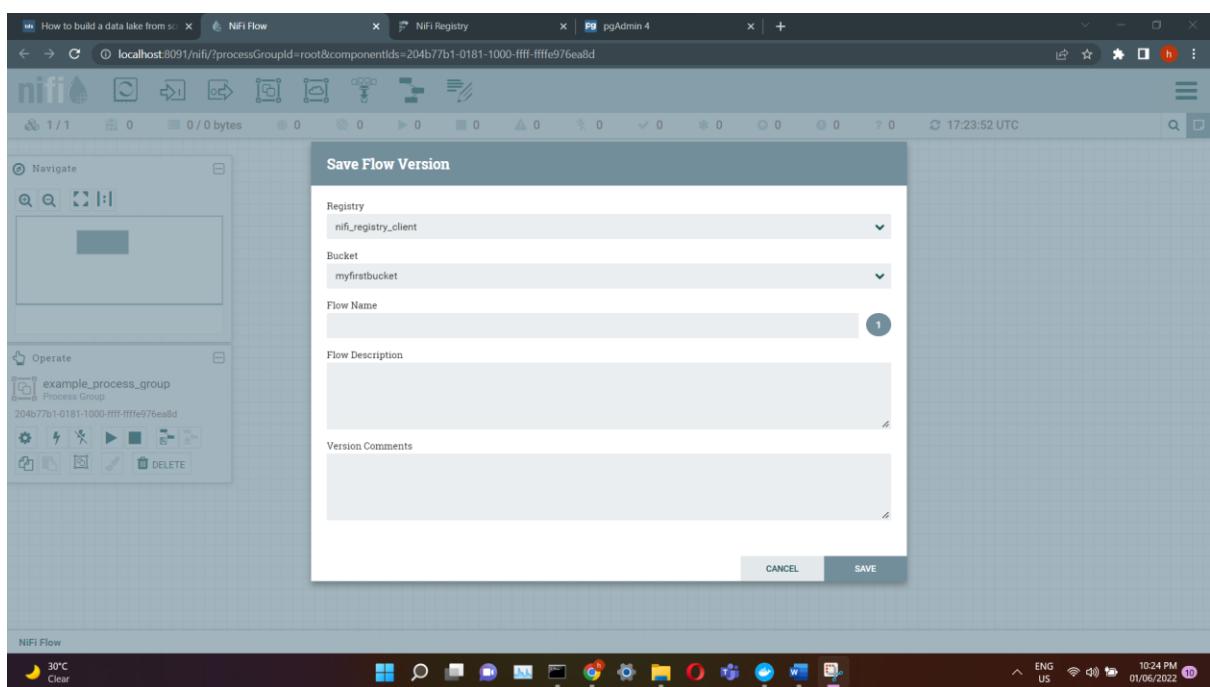
Head over to <http://localhost:8091/nifi/> and click on the three bars in the top right corner to access the controller setting. Then click on the tab registry client and on the plus-symbol on the right-hand side.



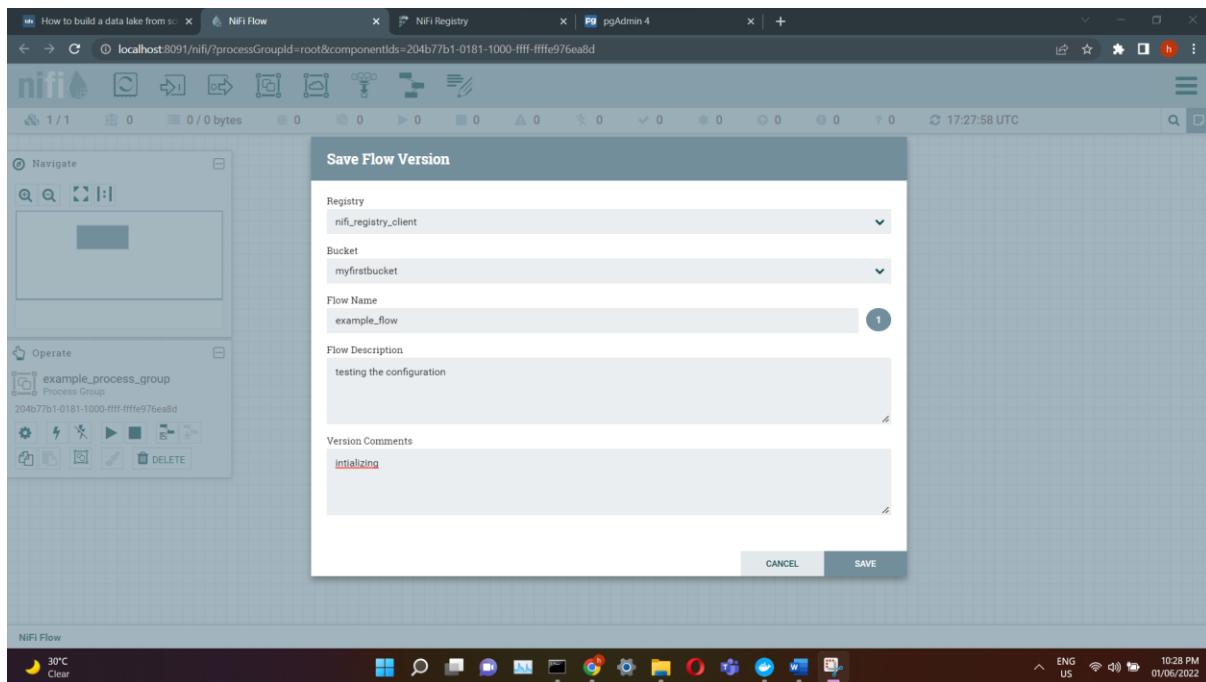
Add the following in name and location:

Name: nifi_registry_client

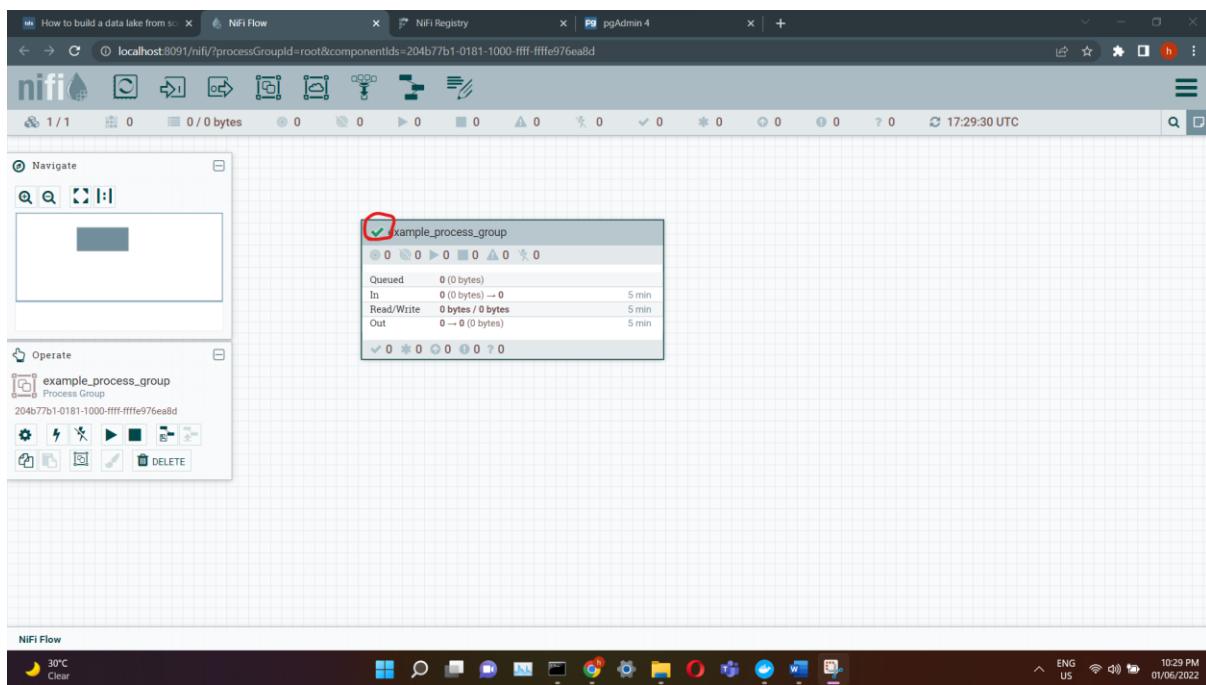
Location: <http://myregistry:18080>



Right-click on the process group and choose Version -> Start version control:



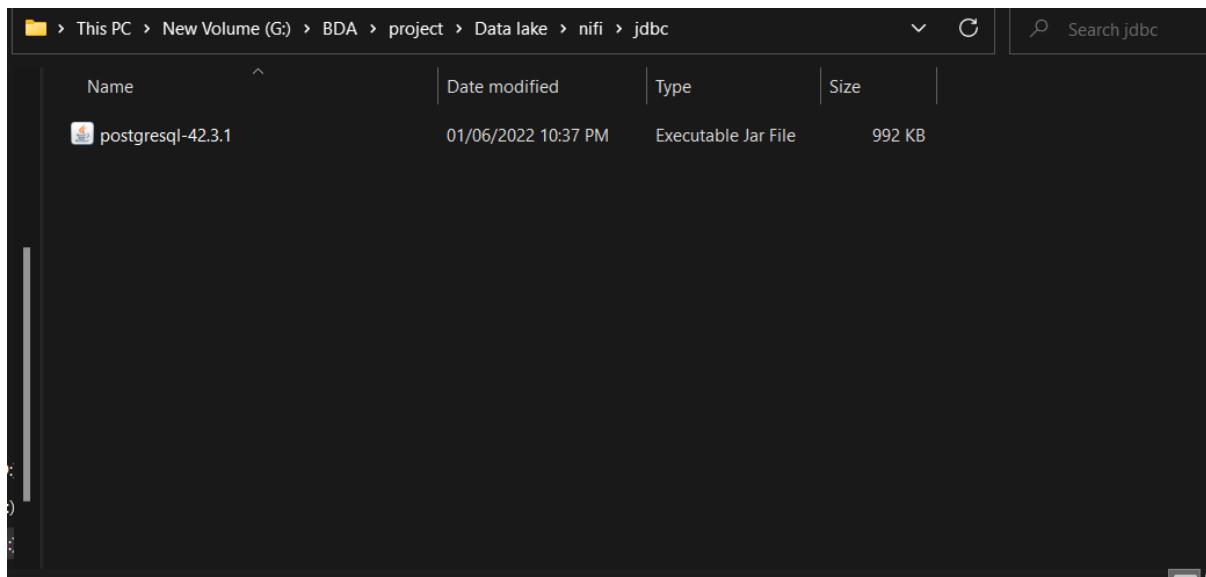
Enter a name and optionally also a description and version comment.



Once you clicked on “Save” you will notice a green checkmark in the top left corner of the process group. This means that the current state of the group is saved to the bucket! We now have a full version control system for the Apache NiFi data flows and ETL-pipelines in place!

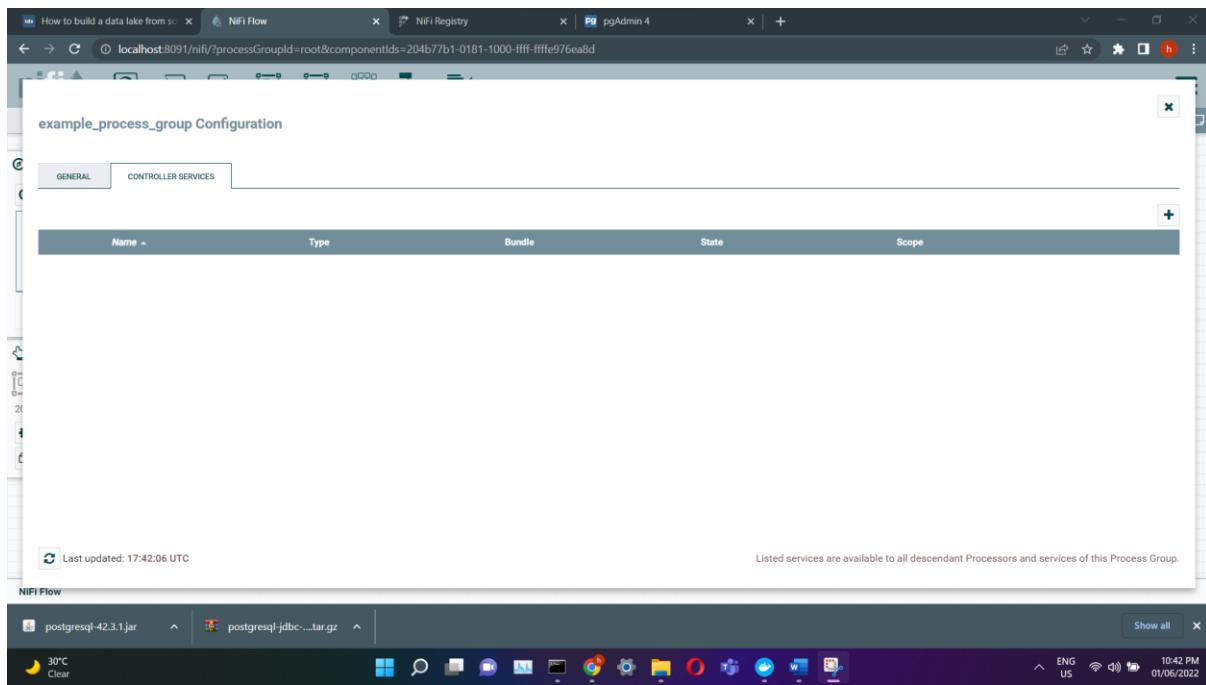
Apache NiFi: Hello postgres database

To be able to set up a controller service connecting to a postgres database, we need to download the current JDBC driver for postgres.

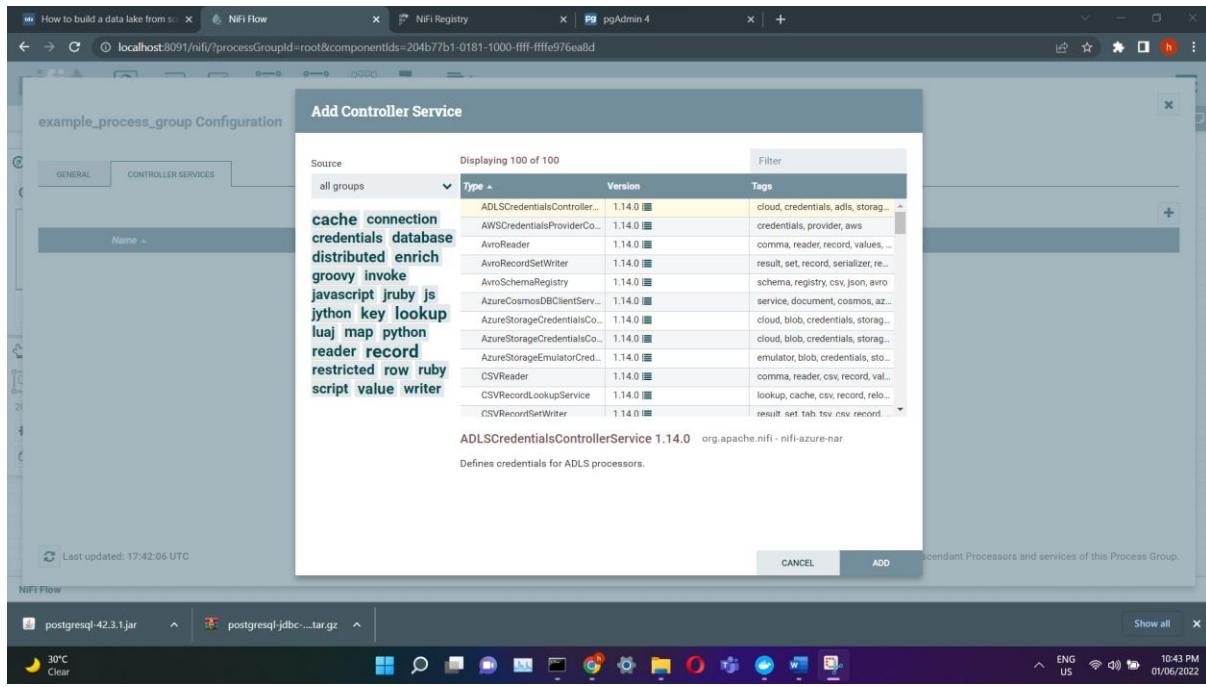


Copy the postgresql-42.3.1.jar file (or whichever version you use) to the docker-mounted directory /nifi/jdbc/.

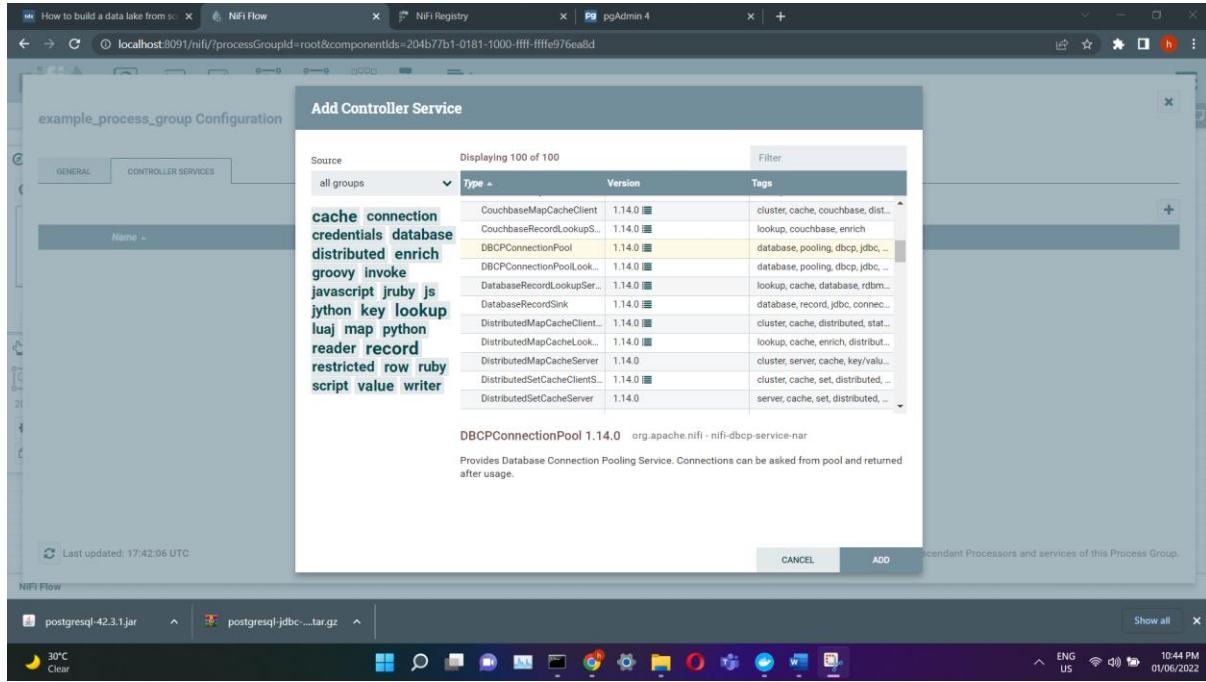
Access the controller configurations in NiFi via the gear-symbol button of the current process group on the left-hand side.



Add a new controller by clicking the plus-symbol on the right-hand side.

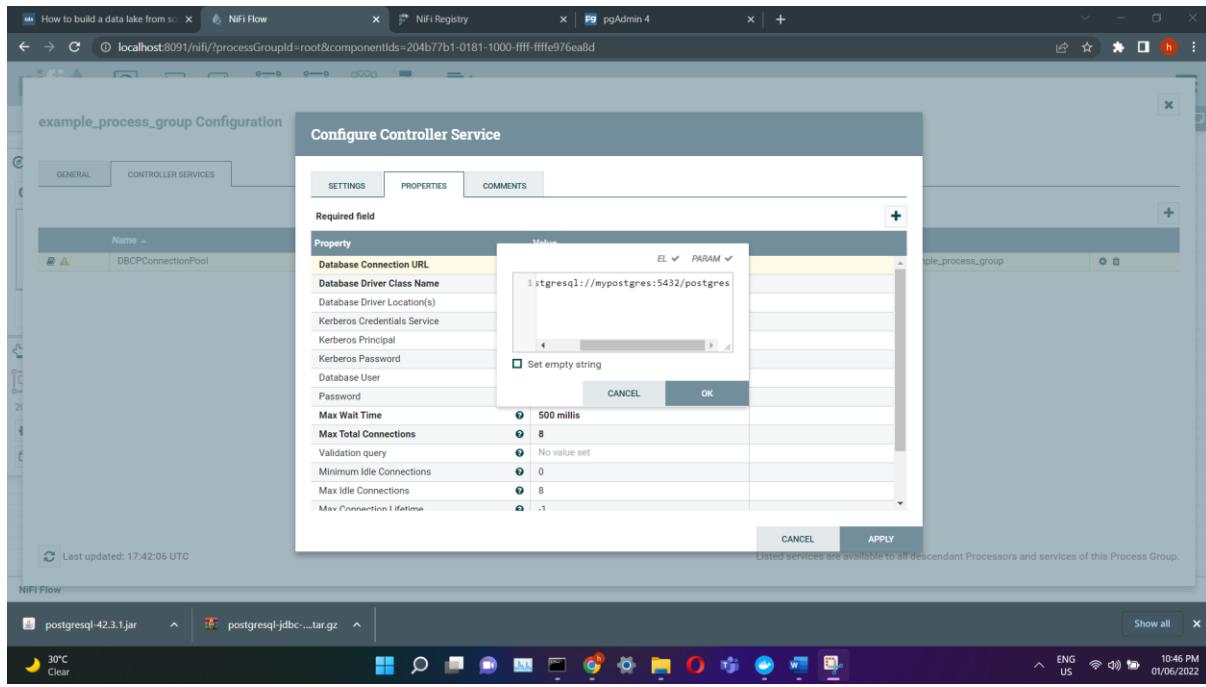


Choose DBCPConnectionPool from the long list of available controllers and click Add.

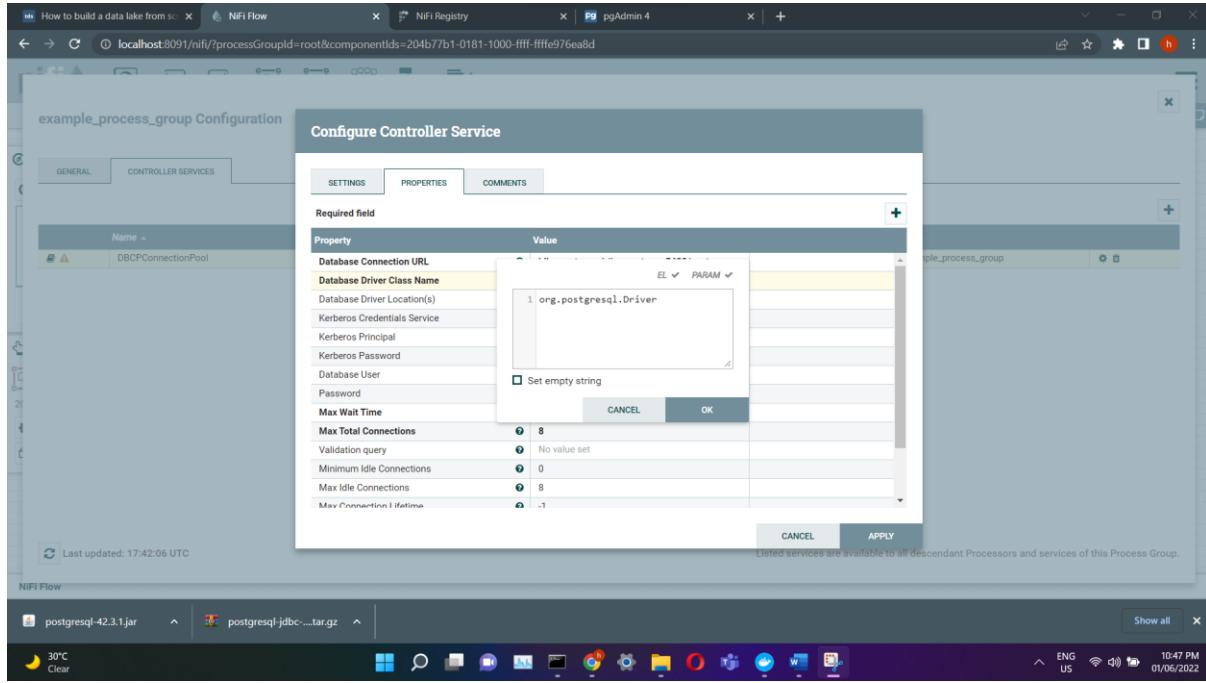


Edit the newly created controller's properties as follows:

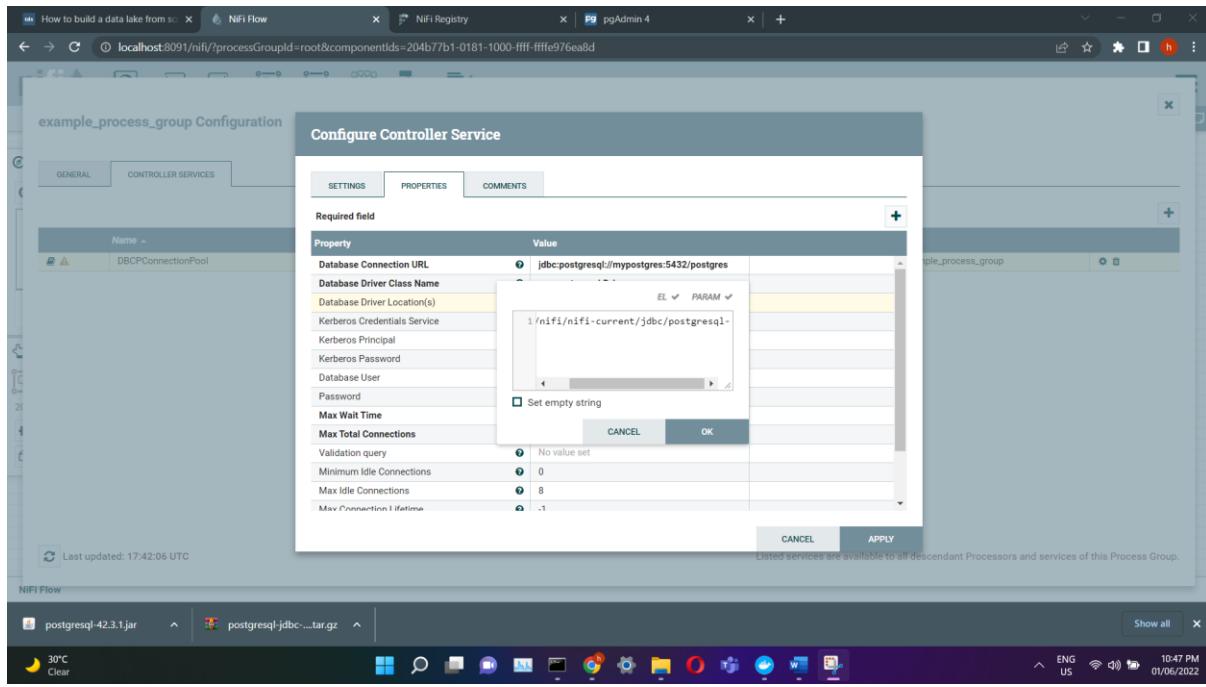
- Database Connection URL: `jdbc:postgresql://mypostgres:5432/postgres`



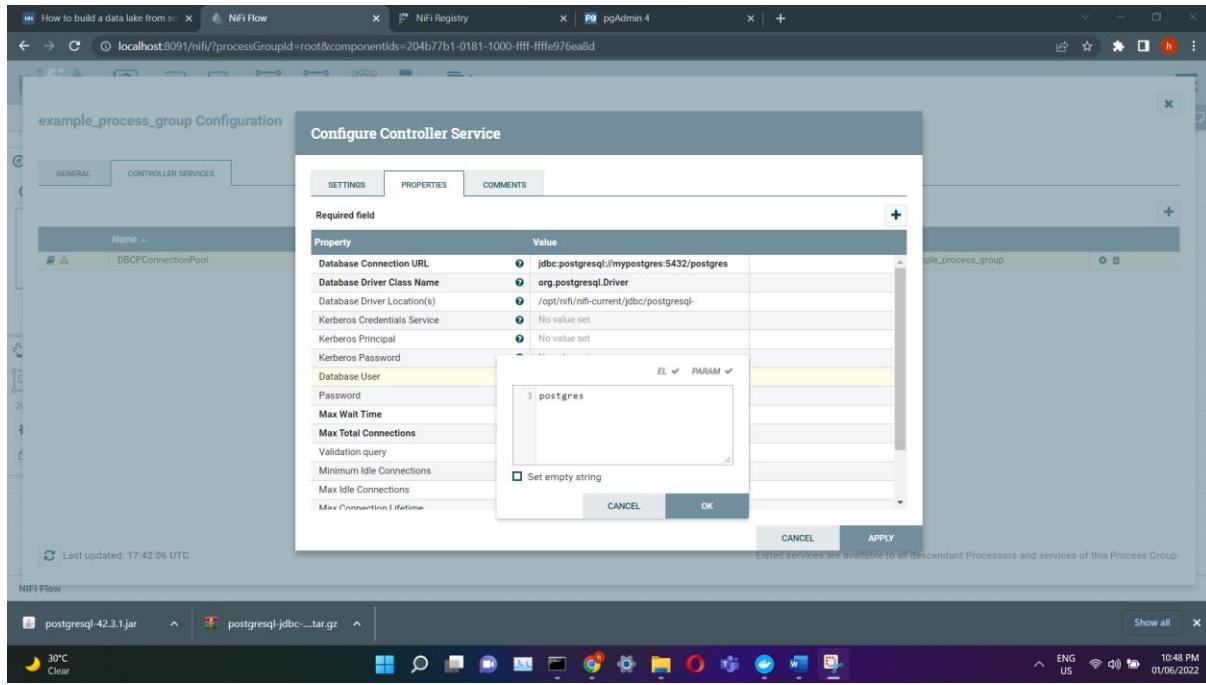
- Database Driver Class Name: org.postgresql.Driver



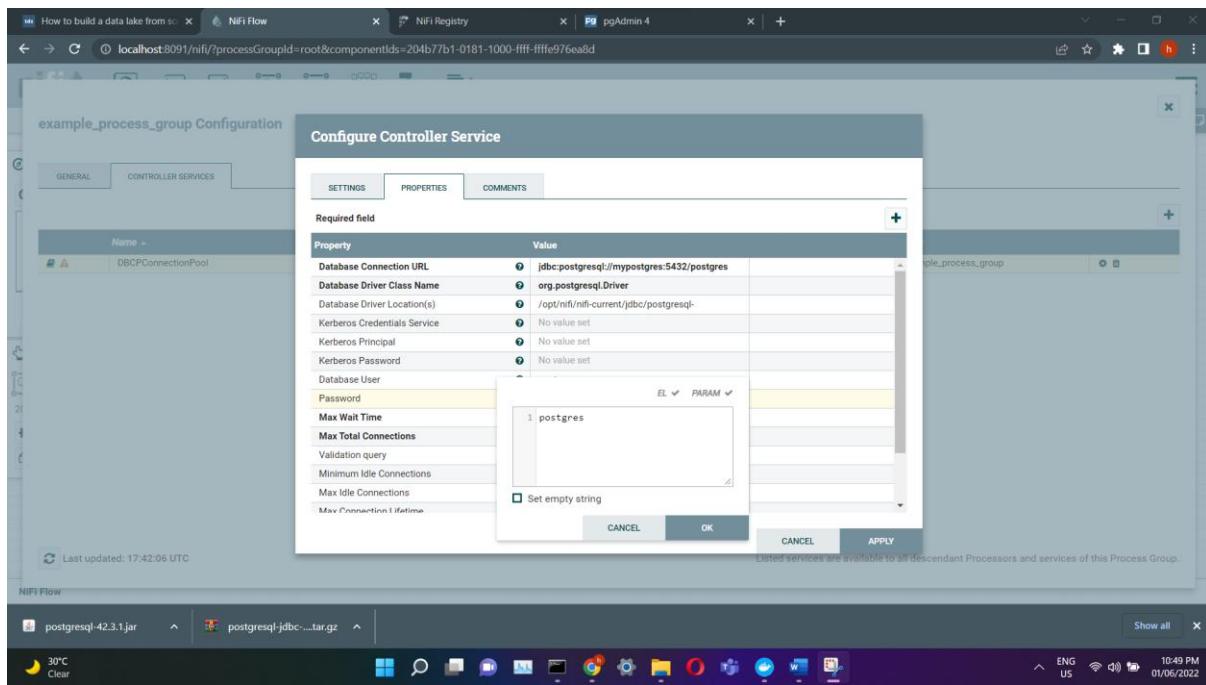
- Database Driver Location(s): /opt/nifi/nifi-current/jdbc/postgresql-42.3.1.jar (Note that we configure the path inside the docker container, not the path on our local machine!)



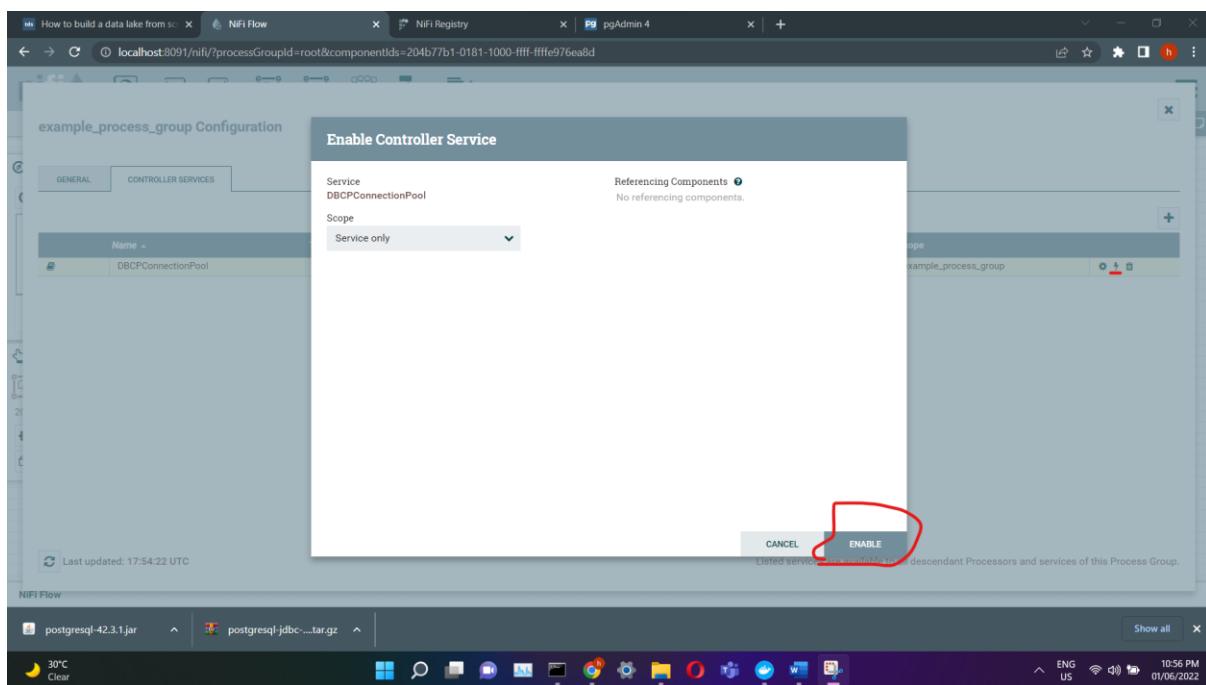
- Database User: postgres



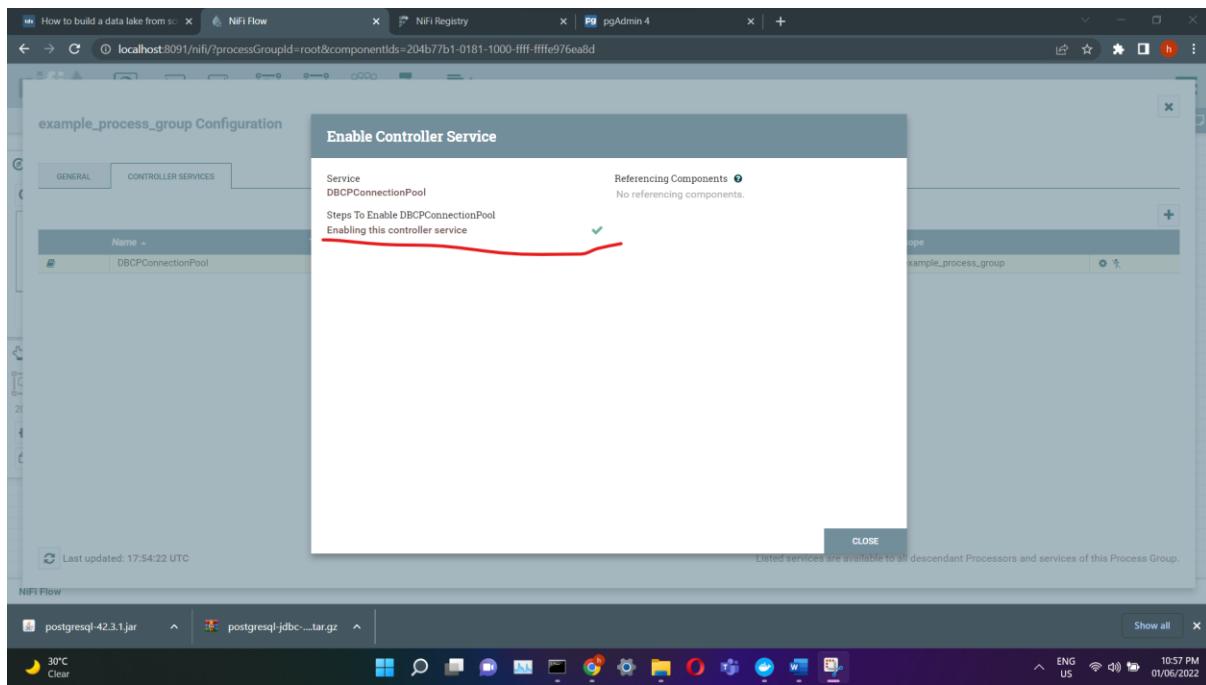
- Password: postgres



Click Apply and enable the service by selecting the lightning-symbol of the line depicting the newly created controller and choosing Enable in the window which opens.

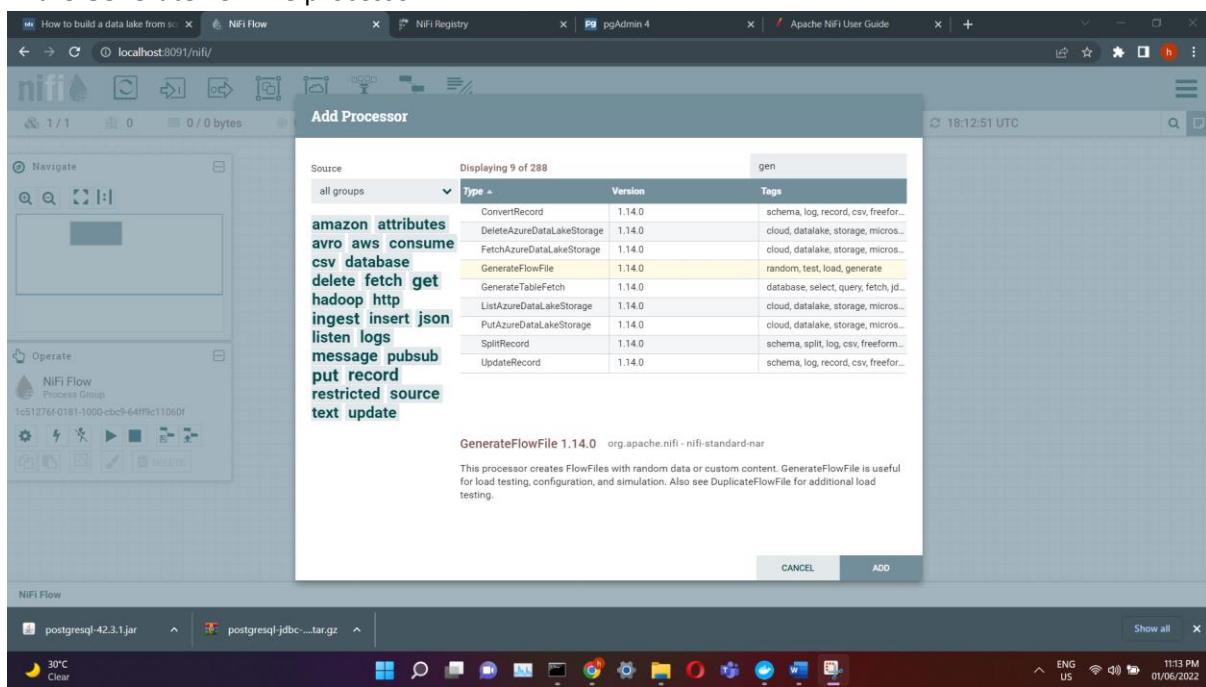


Enabled

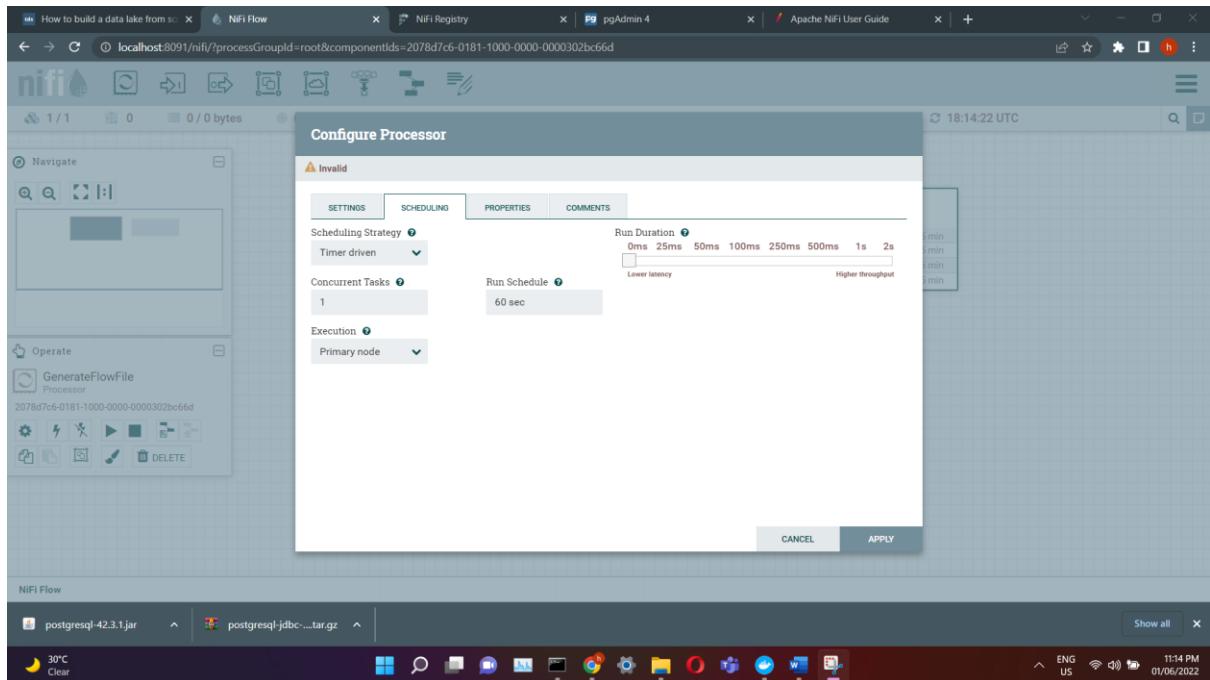


Processors

In the **GenerateFlowFile** processor

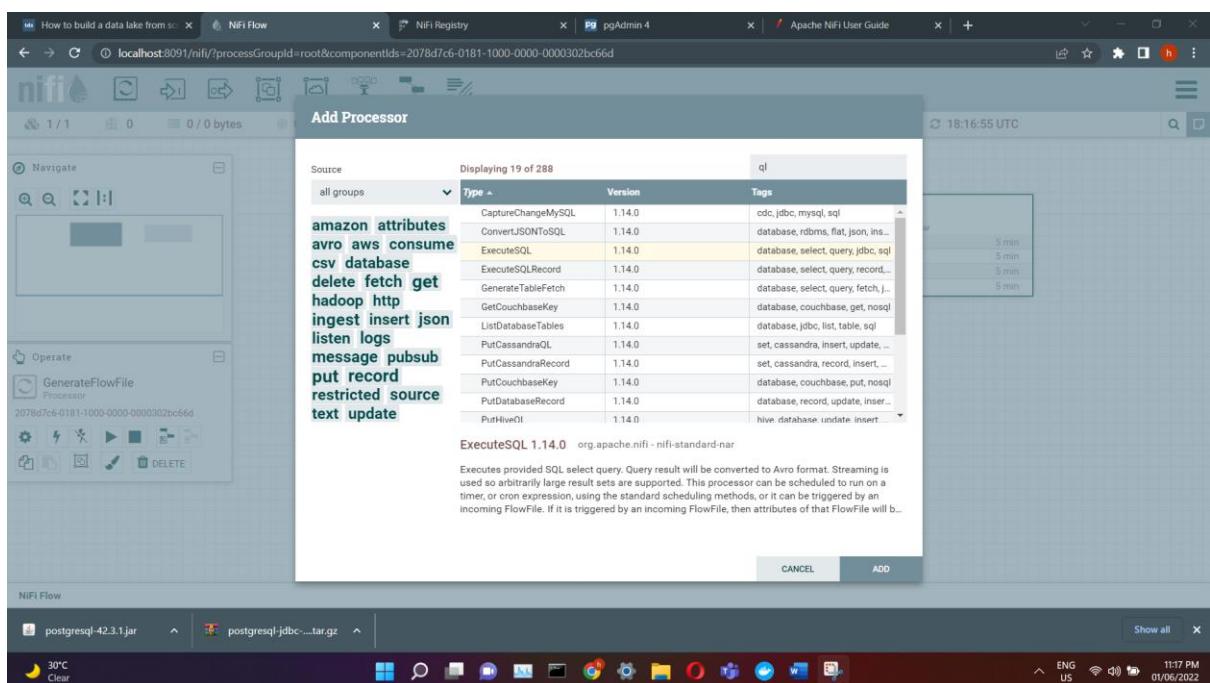


Change the scheduling option Run Schedule to 60 sec as well as the scheduling option Execution to Primary node.

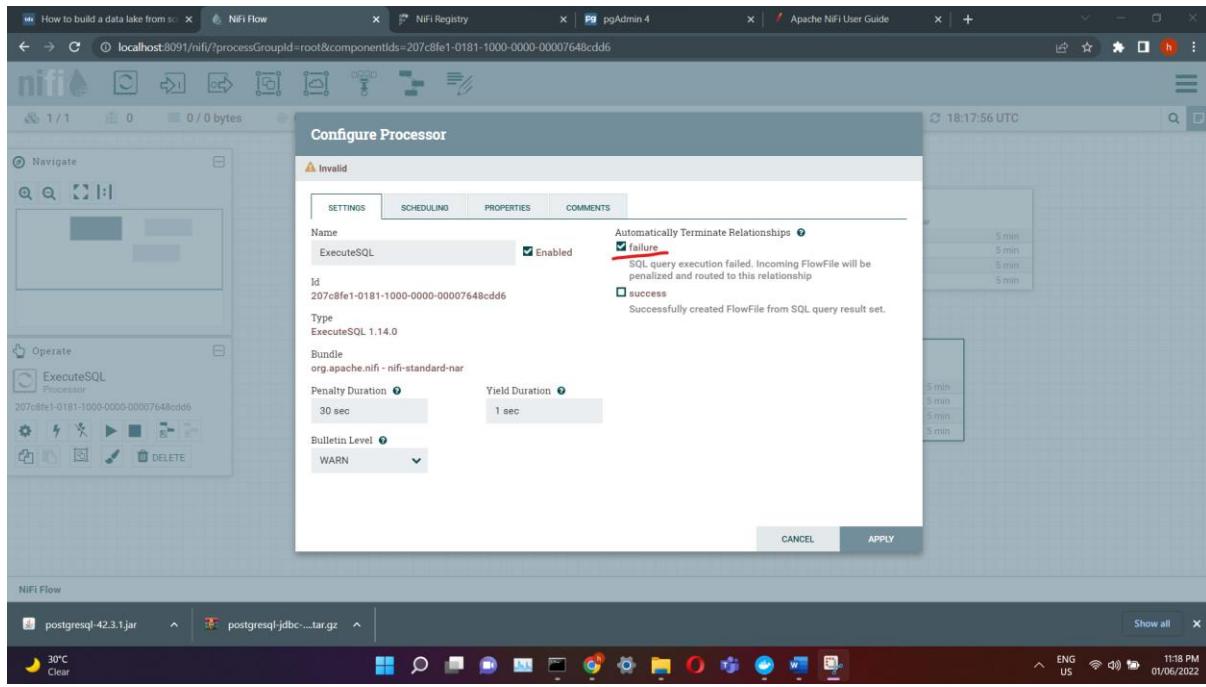


ExecuteSQL

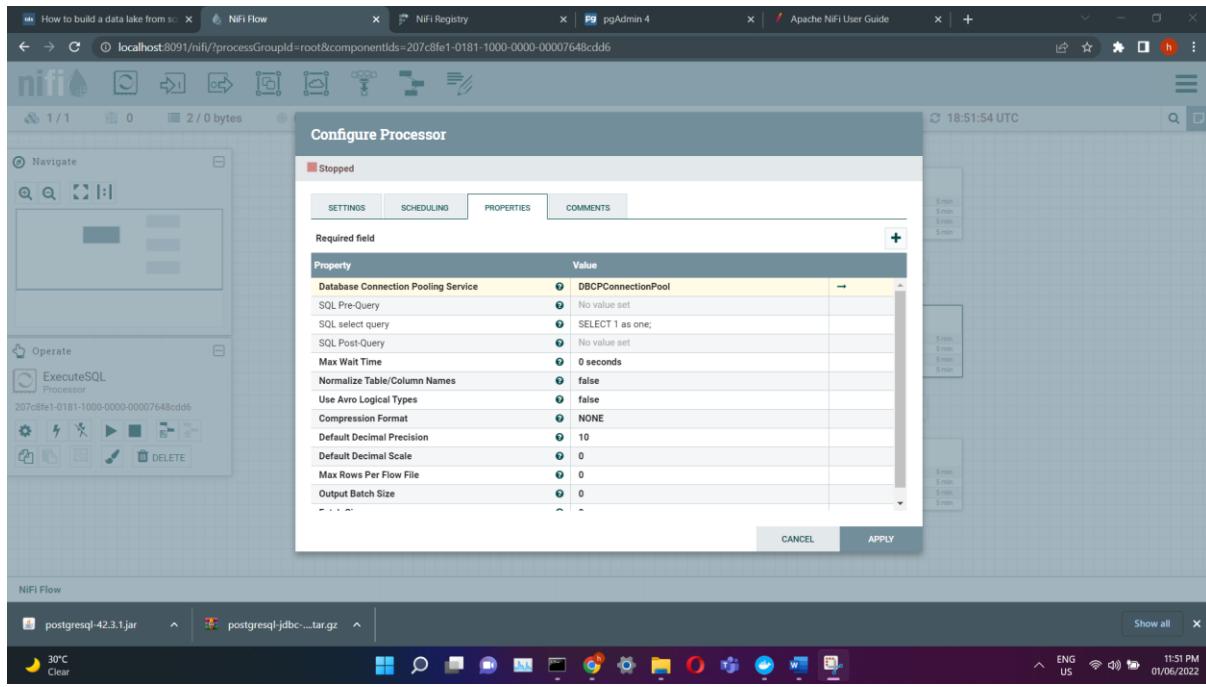
In the **ExecuteSQL** processor,



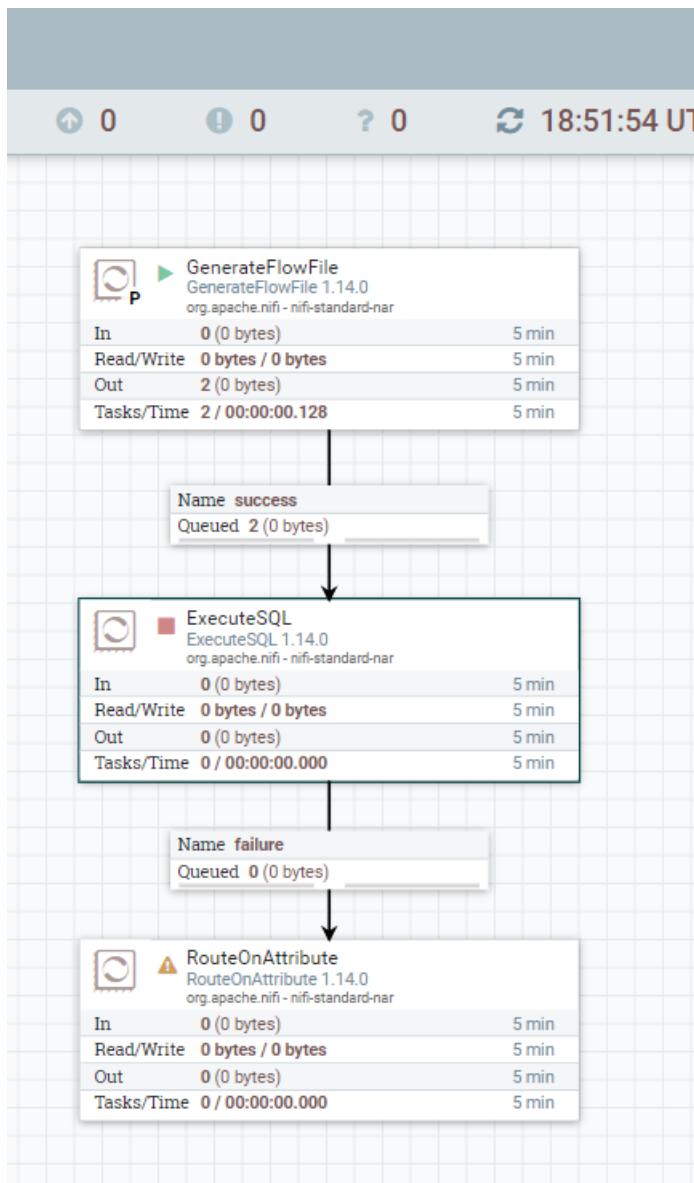
check the box in the main configuration next to failure to automatically terminate the process in case the query fails.



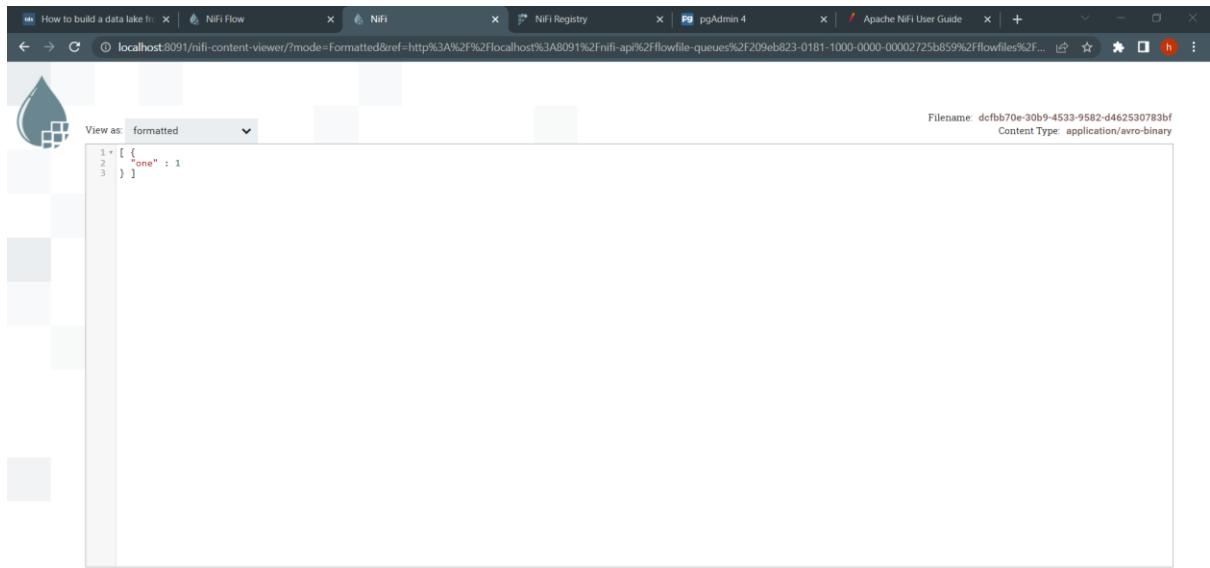
In the properties tab, change the Database Connection Pooling Service property to the newly created postgresql_connectionpool and edit the field SQL select query to a test-query for our database connection.



Once you start the first two processors, a flow file will be created and the ExecuteSQL processor will be triggered. The flow file after the ExecuteSQL processor will now contain the data which we queried from the database.



The flow file content in the browser inspection (formatted view) looks like this:



The screenshot shows a web browser window with multiple tabs open. The active tab is titled "localhost:8091/nifi-content-viewer?mode=Formatted&ref=http%3A%2F%2Fnifi-api%2Fflowfile-queues%2F209eb823-0181-1000-0000-00002725b859%2Fflowfiles%2F...". The content area displays an Avro schema with one record:

```
1 < [ { "one" : 1
2   }
3 } ]
```

Header information at the top right of the content area includes "Filename: dcfbb70e-30b9-4533-9582-d462530783bf" and "Content Type: application/avro-binary".



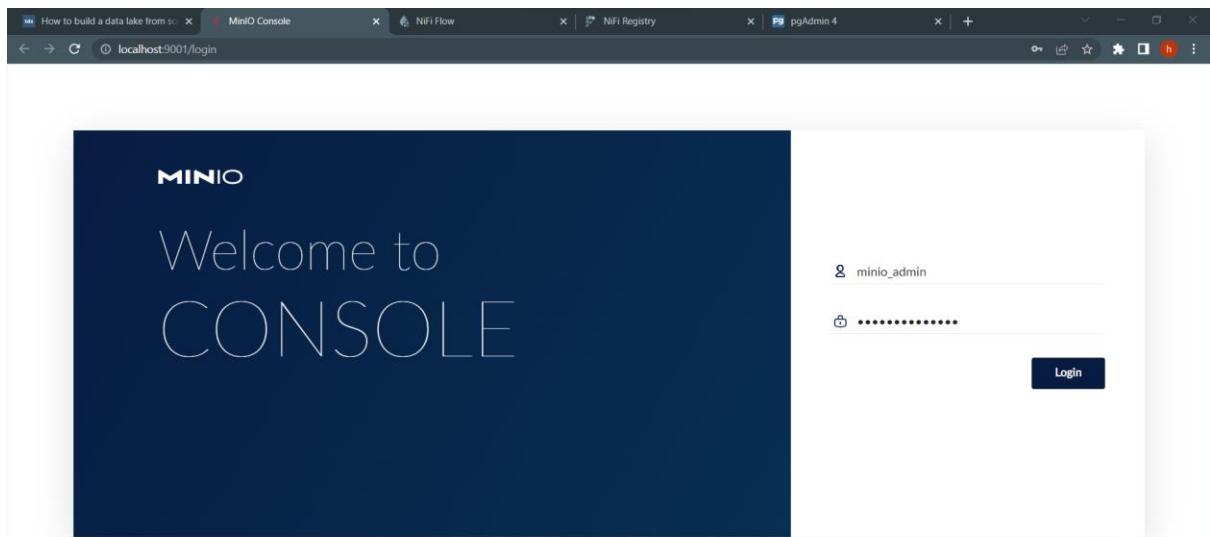
This means that we have successfully queried the database from NiFi!.

Apache NiFi: Hello MinIO

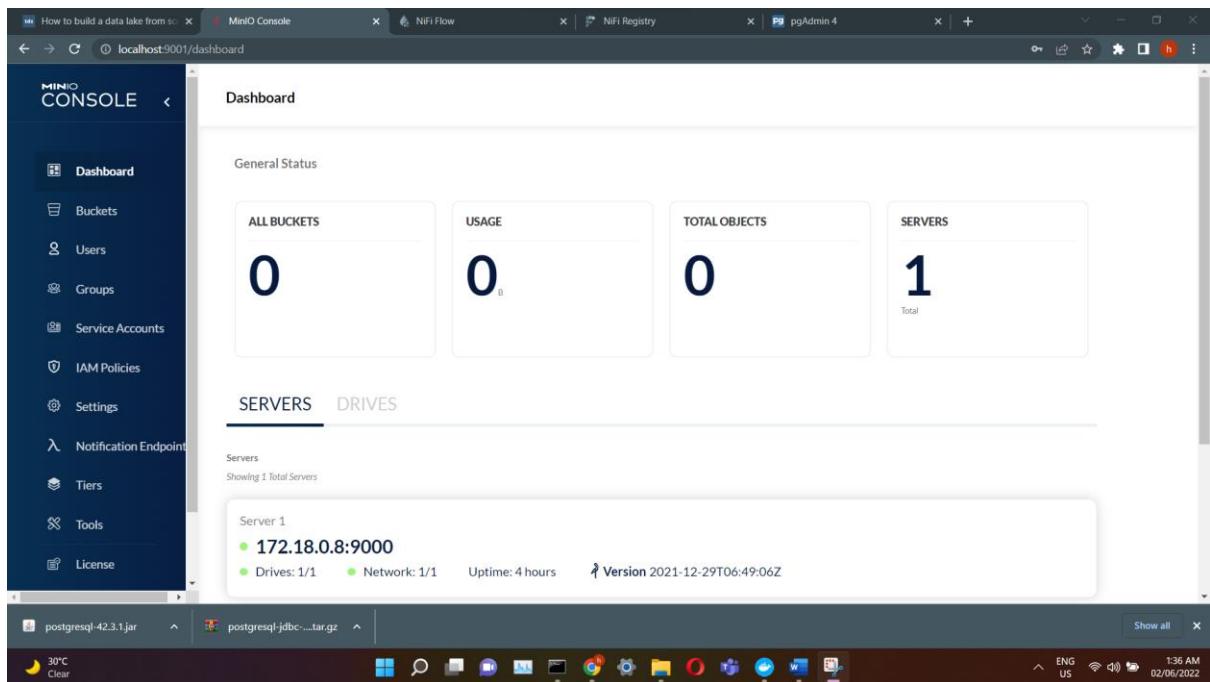
The username and password: for the MinIO:

MINIO_ACCESS_KEY: minio_admin

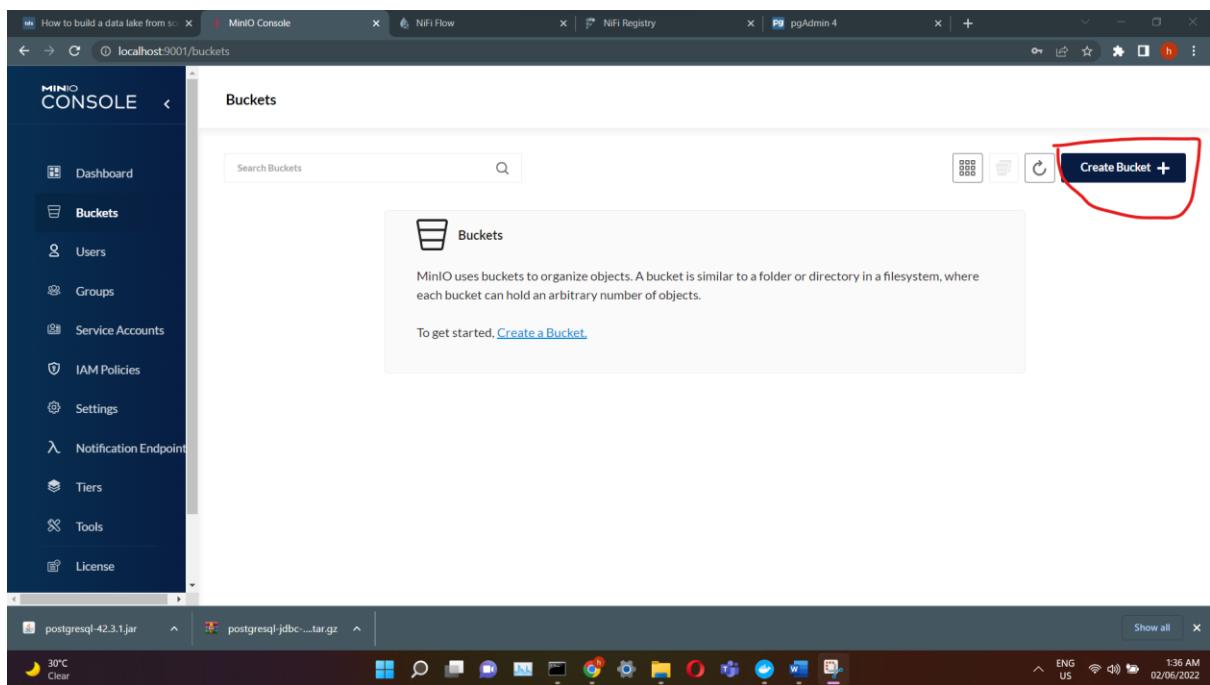
MINIO_SECRET_KEY: minio_password

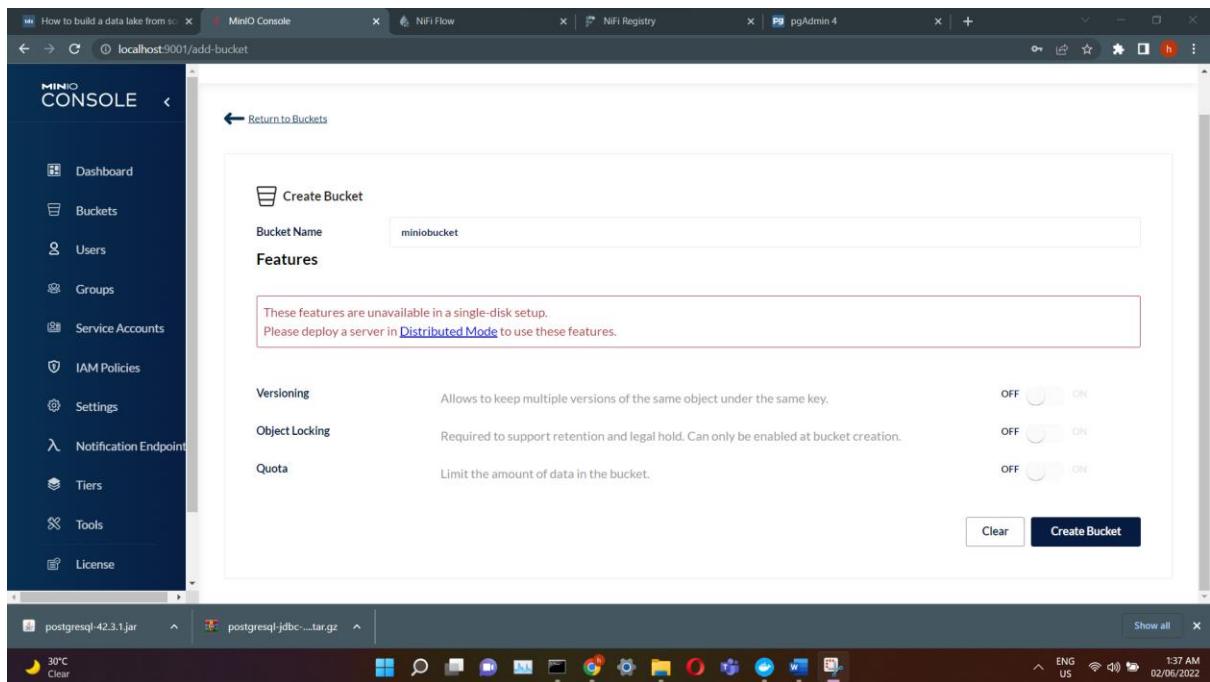


MinIO home page

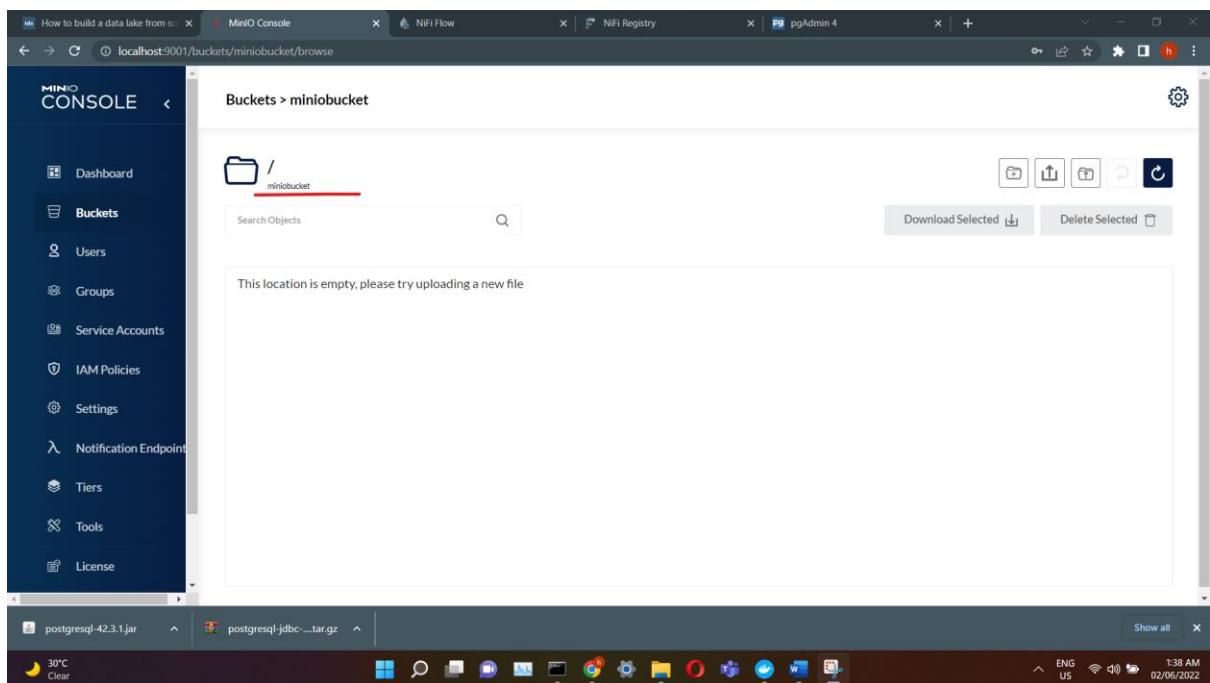


Click on Buckets in the navigation bar on the left and click Create Bucket in the top right corner.

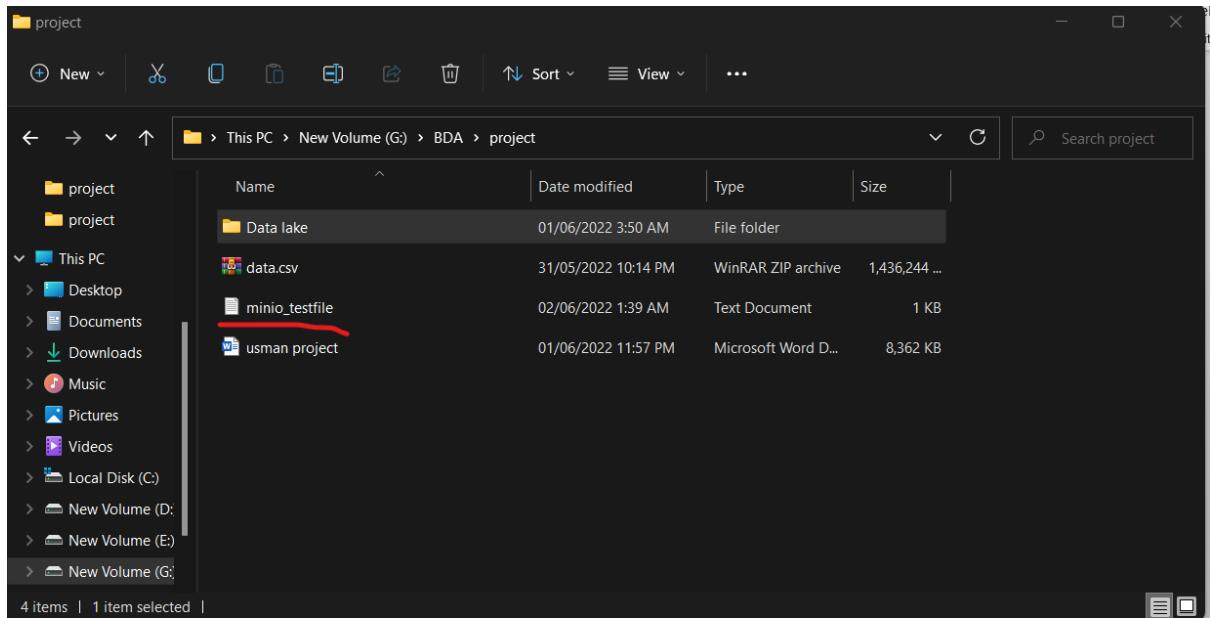




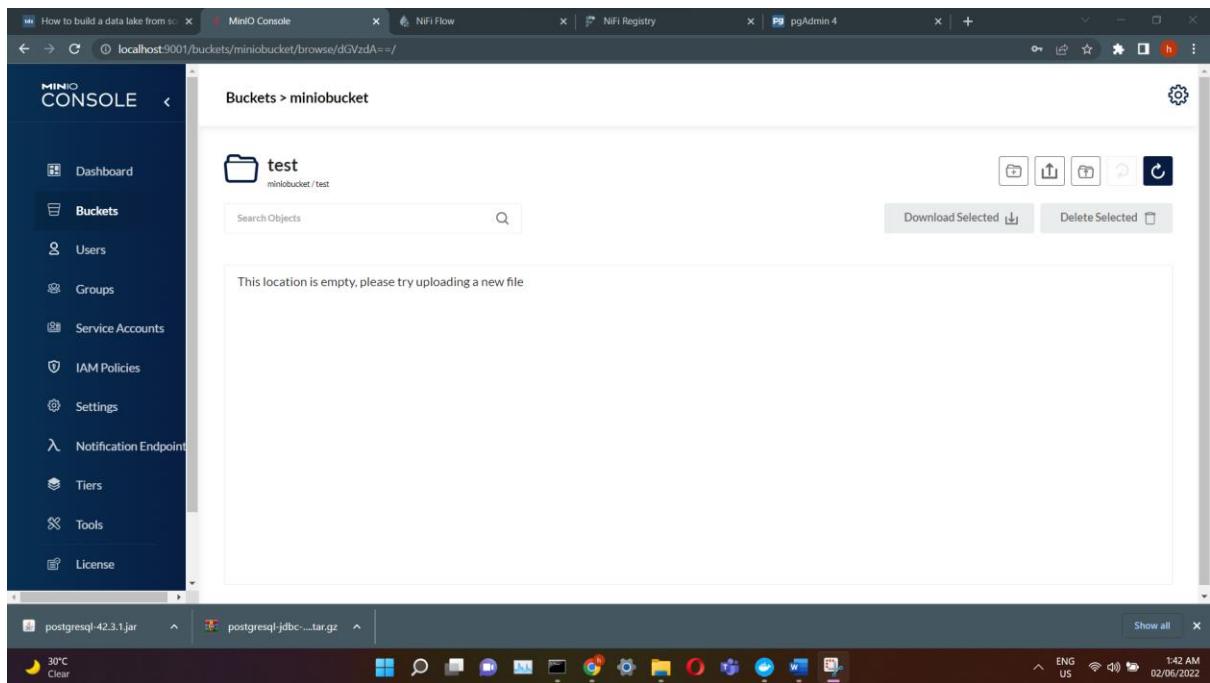
Choose a name for your bucket, e.g. minibucket, keep the default configurations and click on Save.



After creating the bucket, locally create a new file named minio_testfile.txt with the content This is the content of a testfile from MinIO.



Select your bucket and click on Browse. Create a new directory called test and within it.



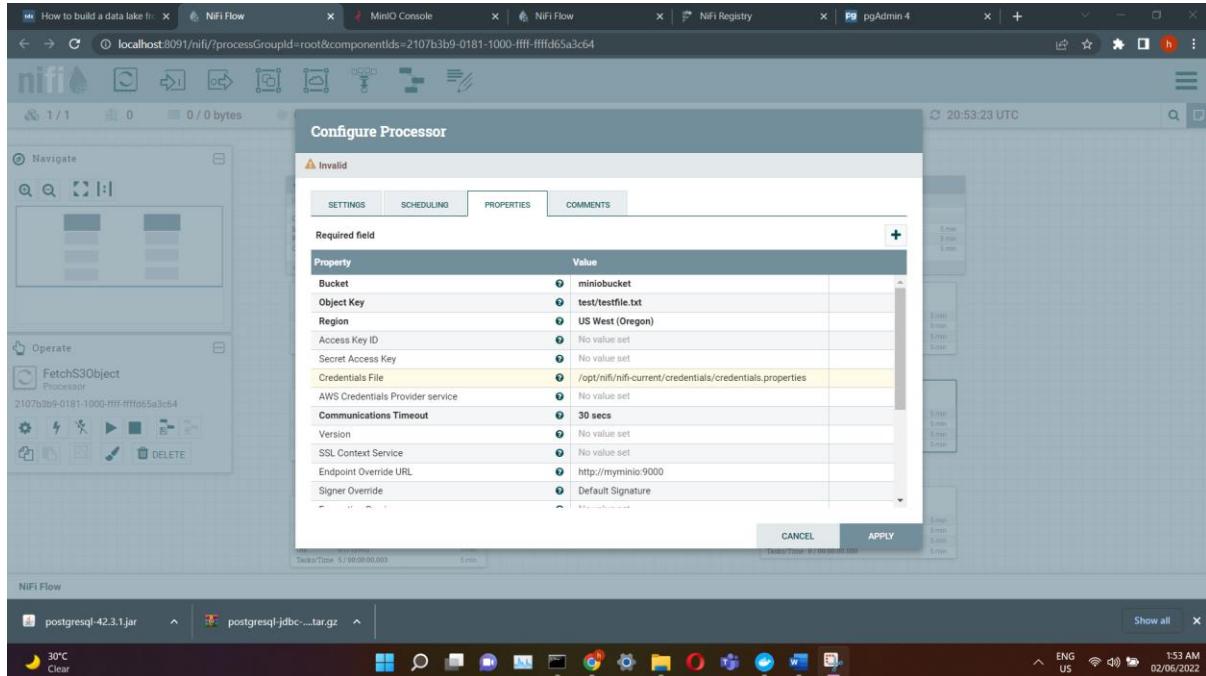
click the Upload file-symbol to upload minio_testfile.txt.

Configuring NiFi

In the FetchS3Object processor, check the box to automatically terminate the relationship on failure and configure the following properties:

- Object Key: test/testfile.txt (we need to configure the full path within the bucket including the directory)
- Endpoint Override URL: <http://myminio:9000> (overriding the endpoint for AWS S3 with the one of our private MinIO instance)

- Credentials File: /opt/nifi/nifi-current/credentials/credentials.properties (the path as specified inside the container)



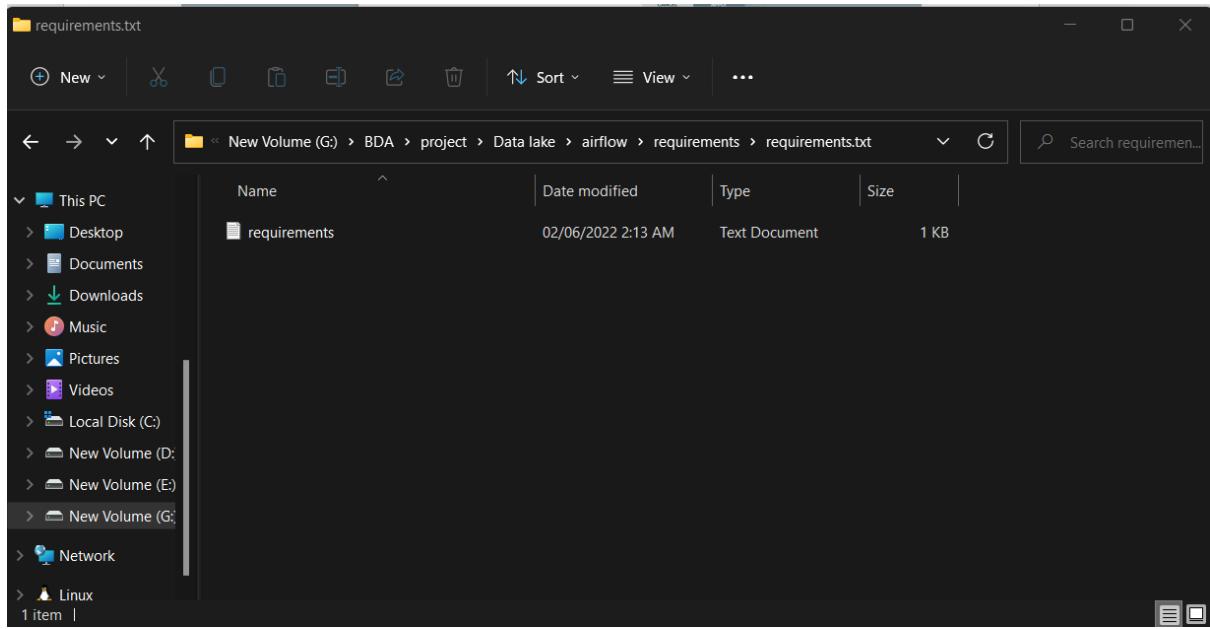
"run exactly once". By right-clicking on the queue leading up to the RouteOnAttribute processor, we can inspect the flowfile.

```
View as: original
1 This is the content of a testfile from MinIO.|
```

NiFi — Flowfile contents — Image created by the author.

Airflow

we need to install a python package on the Airflow service. We can do this by mounting a requirements.txt file. Create the following file in your locally mounted directory (./airflow/requirements/requirements.txt) with the content:



After creating the file, we will need to restart our docker services. On startup, Airflow will automatically install all packages which are mentioned in the file.

```
[2022-06-01 21:18:52 +0000] [134] [INFO] Booting worker with pid: 134
[2022-06-01 21:18:53 +0000] [56] [INFO] Handling signal: ttou
[2022-06-01 21:18:53 +0000] [60] [INFO] Worker exiting (pid: 60)
[2022-06-01 21:19:24 +0000] [56] [INFO] Handling signal: ttin
[2022-06-01 21:19:24 +0000] [153] [INFO] Booting worker with pid: 153
[2022-06-01 21:19:25 +0000] [56] [INFO] Handling signal: ttou
[2022-06-01 21:19:25 +0000] [61] [INFO] Worker exiting (pid: 61)
[2022-06-01 21:19:55 +0000] [56] [INFO] Handling signal: ttin
[2022-06-01 21:19:55 +0000] [173] [INFO] Booting worker with pid: 173
[2022-06-01 21:19:56 +0000] [56] [INFO] Handling signal: ttou
[2022-06-01 21:19:56 +0000] [62] [INFO] Worker exiting (pid: 62)
[2022-06-01 21:17:51,035] {{dagbag.py:403}} INFO - Filling up the DagBag from /usr/local/airflow/dags
[2022-06-01 21:17:51,041] {{__init__.py:51}} INFO - Using executor SequentialExecutor
[2022-06-01 21:17:51,042] {{dagbag.py:403}} INFO - Filling up the DagBag from /usr/local/airflow/dags
[2022-06-01 21:17:51,105] {{__init__.py:51}} INFO - Using executor SequentialExecutor
[2022-06-01 21:17:51,106] {{dagbag.py:403}} INFO - Filling up the DagBag from /usr/local/airflow/dags
172.18.0.5 - - [01/Jun/2022:21:18:15 +0000] "GET /admin/ HTTP/1.1" 200 21944 "-" "curl/7.64.0"
[2022-06-01 21:18:21,888] {{__init__.py:51}} INFO - Using executor SequentialExecutor
[2022-06-01 21:18:21,889] {{dagbag.py:403}} INFO - Filling up the DagBag from /usr/local/airflow/dags
172.18.0.5 - - [01/Jun/2022:21:18:46 +0000] "GET /admin/ HTTP/1.1" 200 21944 "-" "curl/7.64.0"
[2022-06-01 21:18:53,218] {{__init__.py:51}} INFO - Using executor SequentialExecutor
[2022-06-01 21:18:53,218] {{dagbag.py:403}} INFO - Filling up the DagBag from /usr/local/airflow/dags
172.18.0.5 - - [01/Jun/2022:21:19:16 +0000] "GET /admin/ HTTP/1.1" 200 21944 "-" "curl/7.64.0"
[2022-06-01 21:19:24,552] {{__init__.py:51}} INFO - Using executor SequentialExecutor
[2022-06-01 21:19:24,552] {{dagbag.py:403}} INFO - Filling up the DagBag from /usr/local/airflow/dags
172.18.0.5 - - [01/Jun/2022:21:19:46 +0000] "GET /admin/ HTTP/1.1" 200 21944 "-" "curl/7.64.0"
[2022-06-01 21:19:55,884] {{__init__.py:51}} INFO - Using executor SequentialExecutor
[2022-06-01 21:19:55,885] {{dagbag.py:403}} INFO - Filling up the DagBag from /usr/local/airflow/dags
172.18.0.5 - - [01/Jun/2022:21:20:16 +0000] "GET /admin/ HTTP/1.1" 200 21944 "-" "curl/7.64.0"
```

Apache Airflow: Hello postgres database

Open the service in your browser at <http://localhost:8085/admin/> and click on Admin -> Connections in the top bar.

The screenshot shows the Airflow Admin - Connections page. The table lists 38 connections, each with a delete icon. The columns are Conn Id, Conn Type, Host, Port, Is Encrypted, and Is Extra Encrypted. The connections listed include airflow_db, aws_default, azure_container_instances_default, azure_cosmos_default, azure_data_lake_default, beeline_default, bigquery_default, cassandra_default, databricks_default, dingding_default, druid_broker_default, druid_ingest_default, and emr_default.

	Conn Id	Conn Type	Host	Port	Is Encrypted	Is Extra Encrypted
airflow_db	mysql	mysql			●	●
aws_default	aws				●	●
azure_container_instances_default	azure_container_instances				●	○
azure_cosmos_default	azure_cosmos				●	○
azure_data_lake_default	azure_data_lake				●	○
beeline_default	beeline	localhost	10000	●	○	
bigquery_default	google_cloud_platform				●	●
cassandra_default	cassandra	cassandra	9042	●	●	
databricks_default	databricks	localhost			●	●
dingding_default	http				●	●
druid_broker_default	druid	druid-broker	8082	●	○	
druid_ingest_default	druid	druid-overlord	8081	●	○	
emr_default	emr				●	○

Click on Create and fill in the necessary details:

- Conn Id: mypostgres_connection - the ID with which we can retrieve the connection details later on.
- Conn Type: Postgres - Select it from the dropdown menu.
- Host: mypostgres - Docker will resolve the hostname.
- Schema: postgres - the database name (the label is misleading)
- Login: postgres - or whichever username you set in your docker-compose.yml file.
- Password: postgres - or whichever password you set in your docker-compose.yml file.
- Port: 5432 - the standard port for the database within the docker network.

Create a file in your locally mounted directory: airflow/dags/hello_postgres.py with the following content:

Name	Date modified	Type	Size
__pycache__	02/06/2022 2:26 AM	File folder	
hello_postgres	02/06/2022 2:26 AM	Python File	2 KB

The python file appeared in airflow

DAGs

DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
hello_postgres_postgres_operator	None	airflow	○○○○○○○○○○		○○○	Trigger DAG Edit Delete Logs Metrics Table File Help

Showing 1 to 1 of 1 entries

Hide Paused DAGs

Save the file and wait for it to show up in your DAGS view in your web browser. Depending on your DAG, this may take some time. Once it gets listed, activate it on the left and click on "Trigger DAG" on the right-hand side.

Triggered hello_postgres_postgres_operator, it should start any moment now.

DAGs

DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
hello_postgres_postgres_operator	None	airflow	○○○○○○○○○○	2022-06-01 21:30	○○○○○○○○○○	Trigger DAG Edit Delete Logs Metrics Table File Help

Showing 1 to 1 of 1 entries

Hide Paused DAGs

30°C Clear 2:30 AM ENG US 02/06/2022

We can now verify the created table in the interface of pgadmin:

Tables (1)

- testing_connection

NIFI connection with airflow;

let's create a connection in Airflow for our NiFi service as follows:

- Conn ID: mynifi_connection
- Conn Type: Postgres Since there is no NiFi connection type, thus we use postgres as a stand-in.
- Host: http://mynifi
- Port: 8080 This is the port inside the network, not the one we mapped externally!

Connection [create]

List **Create**

Conn Id * mynifi_connection

Conn Type Postgres

Host http://mynifi

Schema (empty)

Login (empty)

Password (empty)

Port 8080

Extra (empty)

Save **Save and Add Another** **Save and Continue Editing** **Cancel**



The output

DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
On hello_nifi	0 3 * * ?	airflow		2022-06-01 21:57		

Apache Airflow: Hello Minio/S3

Admin -> Connections GUI:

- Conn ID: myminio_connection
- Conn Type: S3 Since there is no NiFi connection type, thus we use postgres as a stand-in.

In the Airflow log of the task, you should be able to see the following line File contents: 'This is the content of a testfile from MinIO.'

minio_testfile	02/06/2022 1:39 AM	Text Document	1 KB
----------------	--------------------	---------------	------