



Developer Guides Getting Started Getting Started What is a Graph Database? Intro to Graph DBs Video Series Concepts: RDBMS to Graph Concepts: NoSQL to Graph Getting Started Resources Neo4j Graph Platform Graph Platform Overview Neo4j Desktop Intro Neo4j Browser Intro... [Read more →](#)

## Developer Guides

### Getting Started

- [Getting Started](#)
- [What is a Graph Database?](#)
- [Intro to Graph DBs Video Series](#)
- [Concepts: RDBMS to Graph](#)
- [Concepts: NoSQL to Graph](#)
- [Getting Started Resources](#)

### Neo4j Graph Platform

- [Graph Platform Overview](#)
- [Neo4j Desktop Intro](#)
- [Neo4j Browser Intro](#)
- [Neo4j Bloom Intro](#)
- [How-To: Neo4j ETL Tool](#)
- [Neo4j APOC Library](#)
- [Neo4j Graph Algorithms](#)
- [Neo4j & GraphQL](#)

### Cypher Query Language

- [Cypher Overview](#)
- [Cypher Basics I](#)
- [Cypher Basics II](#)
- [Filtering Query Results](#)
- [Sub Queries](#)
- [Aggregation, Returns, & Functions](#)
- [Cypher Style Guide](#)
- [From SQL to Cypher](#)
- [User Defined Procedures & Functions](#)
- [Tutorial: Dates, Datetimes, and Durations](#)
- [Tutorial: Build a Recommendation Engine](#)
- [Cypher Resources](#)

### Graph Data Modeling

- [Graph Modeling Overview](#)
- [Graph Modeling Guidelines](#)
- [Modeling: RDBMS to Graph](#)
- [Modeling Designs](#)
- [Graph Modeling Tips](#)
- [Tutorial: Refactoring a graph model](#)
- [Interactive Graph Models](#)

### Data Import

- [Import Overview](#)
- [Importing CSV](#)
- [Importing API Data](#)
- [Import: RDBMS to Graph](#)
- [Example: Northwind Dataset](#)
- [How-To: Desktop CSV Import](#)

- [Example Datasets](#)

Graph Visualization

- [Visualization Overview](#)
- [Visualization Tools](#)
- [Other Visualizations](#)

Drivers & Language Guides

- [Drivers Overview](#)
- [Java](#)
  - [Java Driver Spring Boot Starter](#)
  - [Neo4j Object Graph Mapper](#)
  - [Spring Data Neo4j](#)
  - [Spring Data Neo4j RX](#)
  - [Procedures and Functions](#)
  - [Third-party libraries](#)
- [.NET](#)
- [JavaScript](#)
- [Python](#)
- [Go](#)
- [Ruby](#)
- [PHP](#)
- [Erlang & Elixir](#)
- [Perl](#)

Neo4j Tools & Integrations

- [Integrations Overview](#)
- [Apache Spark](#)
- [Elastic-Search](#)
- [MongoDB](#)
- [Cassandra](#)

Neo4j Aura DBaaS

- [Aura DBaaS Overview](#)
- [Connect from Neo4j Desktop](#)
- [Connect from Cypher Shell](#)
- [Connect from your application](#)
- [Data Import with Neo4j Aura](#)
- [Deploying a GRANDstack application to Aura](#)
- [Bloom Visualization with Aura](#)
- [Monitoring](#)

Neo4j Administration

- [Administration Overview](#)
- [Tutorial: Managing Multiple Databases](#)
- [Tutorial: Multi Tenancy Worked Example](#)
- [Sharding Graphs with Fabric](#)
- [Clustering Neo4j](#)
- [Performance Tuning](#)
- [Docker & Neo4j](#)
- [How-To: Run Neo4j in Docker](#)
- [Startups: Free Neo4j Enterprise](#)
- [Online Course: Neo4j Administration](#)

Neo4j in the Cloud

- [Cloud Overview](#)
- [Deploying to Amazon EC2](#)
- [Deploying to Google Compute Engine](#)

- [Deploying to Azure](#)
- [Neo4j Cloud VMs](#)
- [Orchestration Tools](#)

#### Documentation & Resources

- [Resource Overview](#)
- [Learn through GraphAcademy](#)
- [Tutorial: Create Custom Browser Guide](#)
- [How-To: Build with Ruby & Neo4j](#)
- [Available Neo4j Browser Guides](#)
- [Neo4j Documentation](#)
  - [Java Developer Reference](#)
  - [Operations Manual](#)
  - [Cypher Refcard](#)

#### Contributing to Neo4j

- [Contributing Overview](#)
- [Help on Community Forums](#)
- [Speaker Program: Share your Story](#)



Want to Speak? [Get \\$ back.](#)

## How-To: Run Neo4j in Docker

### Goals

You will learn how to create and run a Neo4j graph database in a Docker container. This tutorial is designed for you to follow along and step through the process.

### Prerequisites

This guide builds upon the basic concepts discussed in earlier guides and some knowledge of Docker. To better understand and utilize Neo4j with Docker, it helps to know the following:

- [What is a Graph Database?](#)
- [Docker: What is a Container?](#)

You should also have downloaded [Docker](#) for your appropriate operating system and be familiar with navigating it from the command line.

### Beginner

## Neo4j Docker Image

There is an official [Neo4j image on DockerHub](#) we can use to give us a standard, ready-to-run package of Neo4j. From the DockerHub repo, we can run Community Edition or Enterprise Edition with a variety of versions of the database. The list from Neo4j's options in dockerhub is shown below.



## Supported tags and respective Dockerfile links

- `3.5.3` , `3.5` , `latest` ([3.5.3/community/Dockerfile](#))
- `3.5.3-enterprise` , `3.5-enterprise` , `enterprise` ([3.5.3/enterprise/Dockerfile](#))
- `3.5.2` ([3.5.2/community/Dockerfile](#))
- `3.5.2-enterprise` ([3.5.2/enterprise/Dockerfile](#))
- `3.5.1` ([3.5.1/community/Dockerfile](#))
- `3.5.1-enterprise` ([3.5.1/enterprise/Dockerfile](#))
- `3.5.0` ([3.5.0/community/Dockerfile](#))
- `3.5.0-enterprise` ([3.5.0/enterprise/Dockerfile](#))
- `3.4.12` , `3.4` ([3.4.12/community/Dockerfile](#))
- `3.4.12-enterprise` , `3.4-enterprise` ([3.4.12/enterprise/Dockerfile](#))
- `3.4.11` ([3.4.11/community/Dockerfile](#))
- `3.4.11-enterprise` ([3.4.11/enterprise/Dockerfile](#))

To determine which image we want, we need to piece together a few options. First, the `neo4j` tag starts each image name. After that, the version is preceded with a colon like the `neo4j:3.5.0` image. We will choose to pull the latest version of the image because we want to get all the latest features. For community, the latest version is specified with `neo4j:latest`.

The Enterprise Edition has a `-enterprise` ending to the name after the version. The latest version of enterprise is tagged with `neo4j:enterprise`.

You need to have a valid commercial license in order to use the Enterprise Edition. Using an enterprise Docker image will require you to accept the official Enterprise license agreement.

## Neo4j Configuration

The Neo4j Docker image includes some basic configuration defaults that should not need adjustment for most cases. However, if interested, the full list of default configurations for Neo4j in Docker can be found on [the GitHub repository](#).

By default, the Docker image exposes three ports for remote access:

- 7474 for HTTP
- 7473 for HTTPS
- 7687 for Bolt

We will use these ports to connect to Neo4j inside the container, accessing it from Neo4j Browser, an application, or other methods.

It is also possible to create a custom Docker image with Neo4j included, but we will not cover that here.

## Run Docker with Neo4j

Retrieving and running Neo4j within a Docker container is very simple using one of the provided images. We will need to execute the basic `docker run` command with the `neo4j` image and specify any options or versions we want along with that. Let us take a look at a few options available with the `docker run` command.

Option	Description	Example
<code>--name</code>	Name your container (avoids generic id)	<code>docker run --name myneo4j neo4j</code>
<code>-p</code>	Specify container ports to expose	<code>docker run -p7687:7687 neo4j</code>
<code>-d</code>	Detach container to run in background	<code>docker run -d neo4j</code>
<code>-v</code>	Bind mount a volume	<code>docker run -v \$HOME/neo4j/data:/data neo4j</code>
<code>--env</code>	Set config as environment variables for Neo4j database	<code>docker run --env NEO4J_AUTH=neo4j/test</code>
<code>--help</code>	Output full list of <i>docker run</i> options	<code>docker run --help</code>

By default, Neo4j requires authentication and requires us to first login with `neo4j/neo4j` and set a new password. We will skip this password reset by initializing the password when we create the Docker container using the `--env NEO4J_AUTH=neo4j/<password>` option.

Let us go ahead and create our Neo4j container by running the command below. An explanation of each option is in the following paragraphs.

```
docker run \
  --name testneo4j \
  -p7474:7474 -p7687:7687 \
  -d \
  -v $HOME/neo4j/data:/data \
  -v $HOME/neo4j/logs:/logs \
  -v $HOME/neo4j/import:/var/lib/neo4j/import \
  -v $HOME/neo4j/plugins:/plugins \
  --env NEO4J_AUTH=neo4j/test \
  neo4j:latest
```

The `docker run` simply creates and starts a container. On the next line, `--name testneo4j` defines the name we want to use for the container as `testneo4j`. This avoids us having to reference the container by its generic

id, making our container easier to reference and to remember.

Using the `-p` option with ports 7474 and 7687 allows us to expose and listen for traffic on both the HTTP and Bolt ports. Having the HTTP port means we can connect to our database with Neo4j Browser, and the Bolt port means efficient and type-safe communication requests between other layers and the database.

Next, we have `-d`. This detaches the container to run in the background, meaning we can access the container separately and see into all of its processes.

The next several lines start with the `-v` option. These lines define volumes we want to bind in our local directory structure so we can access certain files locally.

- The first one is for our `/data` directory, which stores the authentication and roles for each database, as well as the actual data contents of each database instance (in `graph.db` folder).
- The second `-v` option is for the `/logs` directory. Outputting the Neo4j logs to a place outside the container ensures we can troubleshoot any errors in Neo4j, even if the container crashes.
- The third line with the `-v` option binds the import directory, so we can copy CSV or other flat files into that directory for importing into Neo4j. Load scripts for importing that data can also be placed in this folder for us to execute.
- The next `-v` option line sets up our plugins directory. If we want to include any custom extensions or add the Neo4j APOC or graph algorithms library, exposing this directory simplifies the process of copying the jars for Neo4j to access.

On the next line with the `--env` parameter, we initiate our Neo4j instance with a username and password. Neo4j automatically sets up basic authentication with the `neo4j` username as a foundation for security. Since it will initiate authentication and require a password change when first connecting, we can handle all of that in this parameter.

Finally, the last line of the command above references the Docker image we want to pull from DockerHub (`neo4j`), as well as any specified version (in this case, just the `latest` edition).

When we run this command, it will create and start the container. We can see this because it generates a container id like in the image below. Even though it creates a container id, you can reference the container using the name we set up in the command – `testneo4j`.

Click to zoom

```
Jennifer-Reif-MBP:docker jenniferreif$ docker run \
> --name testneo4j \
> -p7474:7474 -p7687:7687 \
> -d \
> -v $HOME/neo4j/data:/data \
> -v $HOME/neo4j/logs:/logs \
> -v $HOME/neo4j/import:/var/lib/neo4j/import \
> -v $HOME/neo4j/plugins:/plugins \
> --env NEO4J_AUTH=neo4j/test \
> neo4j:latest
de64cfc4af5d80e780086cd12549f21ddcf7947699fbfb0347cfb625a332710
Jennifer-Reif-MBP:docker jenniferreif$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
de64cfc4af5d	neo4j:latest	"/sbin/tini -g -- /d..."	4 seconds ago	Up 2 seconds	0.0.0.0:7474->7474/tcp, 7473/tcp, 0.0.0.0:7687->7687/tcp	testneo4j

## Verifying Execution and Stopping the Container 🔗

Once we execute the command above, Neo4j should be running in our Docker container! You can verify this by running `docker ps`.

If you do not see your container in the list when you run `docker ps`, you can run `docker ps -a` instead to see if the container crashed and any associated exit codes.

Click to zoom

```
Jennifer-Reif-MBP:docker jenniferreif$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
fa0be6afb6d9	neo4j:latest	"/sbin/tini -g -- /d..."	10 seconds ago	Up 8 seconds	0.0.0.0:7474->7474/tcp, 7473/tcp, 0.0.0.0:7687->7687/tcp	testneo4j



The above image shows the results of the `docker ps` command, showing the container id, image:version, command, created duration, current status, exposed ports, and the container name.

Since the container is currently running, we can stop the container (without destroying it) using the `docker stop testneo4j` command. To start it again, we can simply execute `docker start testneo4j`. Output of both those commands is shown in the image below. We have added `docker ps` commands in between the start and stop, so we can see the status of the container before and after each command.

[Click to zoom](#)

```
Jennifer-Reif-MBP:docker jenniferreif$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
fa0be6af6bd9       neo4j:latest       "/sbin/tini -g -- /d..." 9 minutes ago       Up 2 seconds       0.0.0.0:7474->7474/tcp, 7473/tcp, 0.0.0.0:7687->7687/tcp  testneo4j
Jennifer-Reif-MBP:docker jenniferreif$ docker stop testneo4j
testneo4j
Jennifer-Reif-MBP:docker jenniferreif$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
Jennifer-Reif-MBP:docker jenniferreif$ docker start testneo4j
testneo4j
Jennifer-Reif-MBP:docker jenniferreif$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
fa0be6af6bd9       neo4j:latest       "/sbin/tini -g -- /d..." 10 minutes ago      Up 2 seconds       0.0.0.0:7474->7474/tcp, 7473/tcp, 0.0.0.0:7687->7687/tcp  testneo4j
```

If we did not create the container properly, and we want to start over, we will need to destroy the container before executing the `docker run` again with the same container name. Running the same run command that we did above will notify us that we cannot create another container with the same name as an existing container. This is shown in the output below.

[Click to zoom](#)

```
Jennifer-Reif-MBP:docker jenniferreif$ docker run \
> --name testneo4j \
> -p7474:7474 -p7687:7687 \
> -d \
> -v $HOME/neo4j/data:/data \
> -v $HOME/neo4j/logs:/logs \
> -v $HOME/neo4j/import:/var/lib/neo4j/import \
> -v $HOME/neo4j/plugins:/plugins \
> --env NEO4J_AUTH=neo4j/test \
> neo4j:latest
docker: Error response from daemon: Conflict. The container name "/testneo4j" is already in use by container "de64cfc4af5d80e780086cd12549f21ddcf7947699fbfbc0347cfb625a332710". You have to remove (or rename) that container to be able to reuse that name. See 'docker run --help'.
```

In order to avoid this, we can destroy the old container first using the `docker rm testneo4j` command. Once we run this, we can use the same `docker run` command from earlier to create our container again.

[Click to zoom](#)

```
Jennifer-Reif-MBP:docker jenniferreif$ docker rm testneo4j
testneo4j
Jennifer-Reif-MBP:docker jenniferreif$ docker run \
> --name testneo4j \
> -p7474:7474 -p7687:7687 \
> -d \
> -v $HOME/neo4j/data:/data \
> -v $HOME/neo4j/logs:/logs \
> -v $HOME/neo4j/import:/var/lib/neo4j/import \
> -v $HOME/neo4j/plugins:/plugins \
> --env NEO4J_AUTH=neo4j/test \
> neo4j:latest
a4554eea288420b8ac944f2b2fea0ad1f9c710c78f942cd542588e8c9f5db5eb
Jennifer-Reif-MBP:docker jenniferreif$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
a4554eea2884       neo4j:latest       "/sbin/tini -g -- /d..." 2 seconds ago       Up 1 second        0.0.0.0:7474->7474/tcp, 7473/tcp, 0.0.0.0:7687->7687/tcp  testneo4j
```

## Executing Other Functionality in Neo4j Containers 🔗

Once you are comfortable with creating, starting, and stopping the Docker container, you can start exploring other Neo4j functionality. Much of the other typical Neo4j processes for importing data, adding plugins, and interacting via Neo4j Browser work the same way as with any other Neo4j installation with the proper directory volumes mounted.

### Cypher and Cypher Shell 🔗

To run any Cypher against our database within the container, we can use either Neo4j Browser or the Cypher shell tool.

#### Neo4j Browser

Neo4j Browser works the same as it does with any other Neo4j instance. Simply ensure the database is running, then open a browser window and enter the url `localhost:7474`.

## Cypher Shell

If we want to run Cypher directly in our container, we need to first access our container. We will need to use the command below in order to run any commands in a running container. In this case, we are telling docker to run bash within our container, allowing us to interact with our container using Linux bash commands. For a full list of options, check out [Docker's info](#) on the exec command.

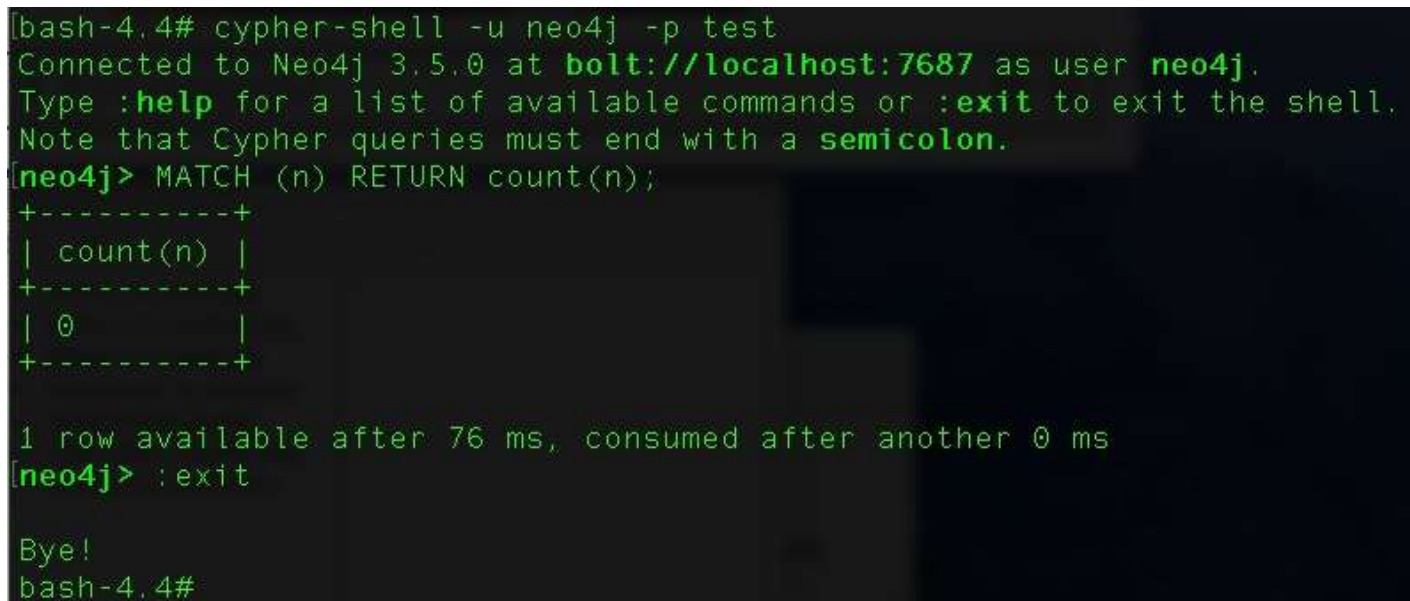
```
docker exec -it testneo4j bash
```

After the above command is run, we can now access Cypher shell by running the `cypher-shell` command, which is shown below. Notice that we also need to specify the username (`-u neo4j`) and password (`-p test`) in order to access the database, using the authentication values we set up when we created the container.

```
cypher-shell -u neo4j -p test
```

We can use the returning prompt to write and run various Cypher statements against our data. The image below shows the command and prompt to access Cypher shell, as well as a query to see how many nodes are in the database (at this point, 0). The final command exits Cypher shell using `:exit` and returns to our bash prompt.

Click to zoom

A terminal window showing the execution of the 'cypher-shell' command. The prompt is '[bash-4.4# cypher-shell -u neo4j -p test]'. The output shows a connection to Neo4j 3.5.0 at bolt://localhost:7687 as user neo4j. It lists available commands and notes that queries must end with a semicolon. The user enters 'MATCH (n) RETURN count(n);' and the output shows a table with one row containing the value 0. The user then enters ':exit' and the prompt returns to '[bash-4.4#]'.

```
[bash-4.4# cypher-shell -u neo4j -p test
Connected to Neo4j 3.5.0 at bolt://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
[neo4j> MATCH (n) RETURN count(n);
+-----+
| count(n) |
+-----+
| 0        |
+-----+

1 row available after 76 ms, consumed after another 0 ms
[neo4j> :exit
Bye!
bash-4.4#
```

## Overriding Default Config

If you do need to modify any of the preset configuration values, you can do so in 3 different ways. We will review each in the next paragraphs.

### 1. Set environment variables

Defaults are set for pagecache and memory (512M each default). To change these, we can use the `--env` parameter in our `docker run` command to set different values for these.

```
docker run \
... \
```



```
--env NEO4J_dbms_memory_pagecache_size=1G \  
neo4j:latest
```

## 2. Mount a /conf volume

We can mount the /conf directory to a local filesystem (like our other directories), so we can modify the neo4j.conf configuration file. To do this, we only need to add another -v option to our docker run command.

```
docker run \  
... \  
-v $HOME/neo4j/conf:/conf \  
neo4j:latest
```

## 3. Build a custom image

To create a custom image, we will need to create our own Dockerfile that includes anything we want to have in our container. While we will not go into detail on this approach, there is more information in our [documentation](#).

## Authentication

As we have discussed and shown above, Neo4j (by default) requires authentication and requires us to login with neo4j/neo4j at the first connection and set a new password.

Just as we did above, we can set the password for the Docker container directly by specifying --env NEO4J\_AUTH=neo4j/<password> in your run directive. We could also disable authentication entirely by specifying --env NEO4J\_AUTH=none instead.

Another way is to run Neo4j as a non-root user by altering the docker run command with a different option. Instead of the --env, we can use the --user option and pass in the user's id and group for access. We can see an example of this below, where it passes in the current user and group as the authentication.

```
docker run \  
... \  
--user="$(id -u):$(id -g)" \  
neo4j:latest
```

## Wrapping Up

Congratulations! You have successfully created and started a Neo4j graph database in a Docker container!

If you have any questions or need assistance using Neo4j with Docker, reach out to us on the [Community Site](#)!

To learn more about running Neo4j with Docker, check out our [documentation](#).

## Contents

- [Neo4j Docker Image](#)
- [Neo4j Configuration](#)
- [Run Docker with Neo4j](#)
- [Verifying Execution and Stopping the Container](#)
- [Executing Other Functionality in Neo4j Containers](#)
- [Wrapping Up](#)