

Installing Neo4j 2025 on Linux

Note: For Linux, the assumption is that you are installing via a tarball (.tgz) vs. rpm installation. These instructions are current as of 28-Mar-2025 and Neo4j Enterprise 2025.03.0. Later versions may change some of these instructions. Email jeff.tallman@neo4j.com for later copies if more than 6 months past the above date.

Also, these instructions may vary from the Neo4j Operations guide as they provide a lot more detail for some operations. If you'd rather follow those instructions, you can find them at <https://neo4j.com/docs/operations-manual/current/>

Preparation

Before installation can happen, the following preparatory steps need to be done.

Ports to be opened (single instance):

The following ports need to be available/open:

Port	Protocol	Used for
7474	http	Neo4j browser/bloom applet downloads
7473	https	same as above but using SSL/TLS
7687	bolt/neo4j	All query interactions with the server
6362	transaction log format	Remote backup (this port is optional)
8491	Apache Arrows	GDS Flight Server

(Cluster) Additional ports to be opened

In addition to the above basic ports, if you are installing a Neo4j cluster, you may need to also have the following ports open:

Port	Protocol	Used for
5000	discovery	Used for cluster members to find each other
6000	transaction log format	Used to forward transaction log records for transactions pending commit
7000	raft	Cluster synchronization/commit coordination
7688	bolt	Server-side routing

Testing ports

If you try to start Neo4j and it starts but you can't access it remotely, you can test whether the ports are the problem using the netcat tool.

1. Shutdown neo4j
2. run "nc -l -p 7687" (or 7474 or whatever port) on neo4j host
3. run "nc -v <hostname> 7687" from a desktop client
4. type any phrase on the desktop client - it should echo on the neo4j host

If testing from a customer's windows laptop, they may have to install netcat - there is a Gnu version called NMap.

Virtual Machine Sizing

What machine sizes are available will often depend on your cloud provider and type of machine (general purpose vs. high memory vs. computational) category selected. Generally speaking, a "high memory" machine is more desirable as it supports a larger database size in-memory vs. general purpose (typically high memory VM's have twice the memory for the same number of cores as general purpose). For high user concurrency transactional workloads (or large numbers of parallel query executions), you may find it better to select computation or general purpose VM instances.

Note that Neo4j is typically licensed by cores and memory size – and most cloud providers use a 2 vCPU per core sizing.

Memory Depends on the Use Case

The amount of memory depends on the size of the data and the use case. For GenAI/GraphRAG or applications using vector indexes, choose the next higher machine size and plan for large heap memory size allocations.

Graph Analytics (non-GDS)

Some extremely rough sizing estimates are as follows:

- ~2.5GB of memory for page cache per 1 million nodes
- ~1-2GB of memory for heap for each concurrent complex analytical query
- ~8-16GB of memory for heap for each large bulk load operation

Note that this is simply a rough estimate based on anecdotal averages. Some typical machine sizes we often see are:

Cores	vCPU	Memory (GB)	Page Cache	Heap Size*	DB Size (GB)	Installation
2	4	16	4-8	2-4	8GB	laptop dev/test
4	8	32	12-16	4-8	24GB	dev/test
8	16	64	24-32	8-16	48GB	small
16	32	128	64-80	16-24	96GB	typical
32	64	256	128-192	24-32	200GB	medium
48	96	384	192-256	32-64	250-400GB	large
64	128	512	304-368	32-64	400-600GB	very large

* Heap Size - If using the larger number for heap, you likely will need to set page cache to the smaller value. Typically, page cache + heap size should total no more than ~75% of RAM

Graph Data Science (GDS) Installations

Some extremely rough sizing estimates are as follows when using GDS are:

- Page Cache: 25-50% of database size
 - If using predominantly native projections and minimal non-GDS queries, the lower bound may be sufficient
 - If using cypher projections and/or non-GDS analytical queries, the higher bound may be necessary for desired performance
- Heap Size: $\frac{1}{2}$ to $\frac{3}{4}$ GB of memory per million nodes per concurrent GDS user for heap.

Note that this is simply a rough estimate based on anecdotal averages. Some typical machine sizes we often see for GDS implementations are:

Cores	vCPU	Memory (GB)	Page Cache*	Heap Size	DB Size (GB)	GDS Users**
4	8	64	12-24	24-32	<50	dev/test (1-2)
8	16	128	12-24	64-84	<50	3-8
16	32	256	24-32	128-192	<100	7-15
24	48	384	48-64	256-296	<200	10-20
32	64	512	96-128	256-384	<300	15-25

* Page Cache - assumes GDS source data and GDS algorithms run in same instance - if using remote projections and only running algorithms in instance, most of page cache can be eliminated and added to Heap Size

**GDS Users - concurrent executions of GDS algorithms - actually number of graph projections and non-GDS sessions may be higher

A note on concurrent executions. The amount of memory depends on the algorithm and the amount of data projected. For example, some similarity algorithms on a graph projection of ~100 million nodes can use ~24GB of heap for a single algorithm. If 3 users are running the same algorithm simultaneously on similar sized projections, you would need ~24GB * 3= ~72GB total memory in heap.

Note that the above memory estimations are strictly anecdotal and may or may not be representative of your requirements. You can use GDS projection and algorithm estimation

modes to get more accurate estimates. It is a good practice when first developing your pipelines to run the algorithm with the estimation methods to get a good idea of the memory requirements for your pipeline.

Add 3-4x the memory as disk space

For example, if you anticipate using 128GB of memory, allocate between 384-512GB of disk space. The exact amount of space will depend on the frequency of backups and the transaction log retention you specify. However, the minimum amount recommended is 3x the memory configuration.

(Optional) Add separate data, transaction log and backup devices

For high IOPS (>5000 IOPS) or systems in which you are expecting the total size of all databases in the instance to exceed 300GB, consider adding additional storage units for separating data and transaction log IO's as well as backup operations to reduce IO contention – especially during large/write operations such as data loads. For cloud based VM's, the storage can be added during the VM creation or afterwards.

This is typically accomplished via the cloud provider's web interface for the specific VM. For example, here is the GCP display for a demo VM that I use:

The screenshot displays the GCP console interface for a VM instance. The left sidebar shows the navigation menu with categories like Virtual machines, Storage, Instance groups, and VM Manager. The main panel shows the 'DETAILS' tab for the VM instance 'tallman-demo-2'. Under the 'Storage' section, the 'Boot disk' is listed as 'tallman-demo-2' with a size of 200 GB. Below this, the 'Local disks' section shows 'None'. The 'Additional disks' section lists three disks: 'backups' (750 GB, SCSI, Balanced persistent disk), 'data-disk' (750 GB, SCSI, SSD persistent disk), and 'txn-logs' (250 GB, SCSI, SSD persistent disk). All additional disks are managed by Google and are in 'Read/write' mode. At the bottom, the 'Backup plan' is shown as 'Managed by Backup and DR Service'.

Name	Image	Interface type	Size (GB)	Device name	Type	Architecture	Encryption	Mode	Wh
tallman-demo-2	centos-stream-9-v20230203	SCSI	200	tallman-demo-2	Balanced persistent disk	x86_64	Google-managed	Boot, read/write	Del

Name	Image	Interface type	Size (GB)	Device name	Type	Architecture	Encryption	Mode	Wh
backups	—	SCSI	750	backups	Balanced persistent disk	—	Google-managed	Read/write	Del
data-disk	—	SCSI	750	data-disk	SSD persistent disk	—	Google-managed	Read/write	Del
txn-logs	—	SCSI	250	txn-logs	SSD persistent disk	—	Google-managed	Read/write	Del

For traditional RDBMS systems, most DBA's know that optimal performance and recoverability has long depended on having multiple separate devices for data and transaction log - and often recommended for indexes, etc. While the Neo4j Operations Manual doesn't call this out, for

large mission critical systems, this common optimization recommendation applies to Neo4j as well.

As a bit of a background, most cloud providers do IO provisioning based on IOPS and throughput limit based on device size for general users. So, for example, on AWS a device may have an IOPS limit of 5000 and a throughput of 125MB/sec. The most IO's you will see will depend on which of those limits are hit first. If you are doing a backup or large import, it is likely that you will hit the 125MB/sec throughput limit long before you see 5000 IOPS. However, if you are doing a ton of small/fast write transactions, you will likely hit the 5000 IOPS limit far below the 125MB/sec throughput limit.

Additionally, the IOPS and throughput may be tied directly to the device size. For example, on Azure, a 512GB device has a 2300 IOPS limit. A 600 GB device will have a 5000 IOPS limit as the next device size cap is at 1024GB. As a result, you may want to size your devices to some small amount above the power of 2 sizing you were considering - e.g. 600 vs. 512 or 1200GB vs. 1024, etc.

Increase process open file limits

For Linux systems, you will also need to increase the number of open files a process can have via editing `/etc/security/limits.conf` and adding the lines below. You may need sudo permissions to do this (substitute the user you are using to install Neo4j for `<username>`):

```
<username> soft    nproc    65536
<username> hard    nproc    65536
#setting the file limit to unlimited may prevent logging in
#so use a setting greater than 400000 but less than unlimited
<username> soft    nofile   500000
<username> hard    nofile   500000
```

On clusters with large numbers of databases (e.g. low 100's+), you may have to increase this limit significantly due to the number of databases and corresponding number of indexes. However, this setting does preallocate memory from the OS and setting it too high can block the user from logging in. As a result, be careful when setting above 1 million unless the system has a lot of memory available. The minimum recommended value for clusters would be 400000. The above setting of 500000 seems to do quite well at supporting 20-30 databases of large schema complexity.

Install Java 21 (JDK or JRE)

While a JRE is sufficient, a JDK is preferable if planning on writing any stored procedures or User Defined Functions (UDF) using the Neo4j Graph Java API. Note that Neo4j versions 5.14.0 and higher support JDK 21 while earlier versions require JDK 17. JDK 21 is preferred - especially for GraphRAG/GenAI applications or any use case using Neo4j vector indices as you can enable the JDK SIMD instructions for performing vector math in parallel.

Install Open JDK 21

Install OpenJDK (or Oracle JDK if licensed) via the following commands (you will likely need to be sudo):

```
#sudo yum list | grep java-21
sudo yum install java-21-openjdk.x86_64 -y
sudo yum install java-21-openjdk-devel.x86_64 -y

#sudo yum list | grep zip
sudo yum install zip.x86_64 -y

sudo yum list | grep sysstat
sudo yum install sysstat.x86_64 -y
```

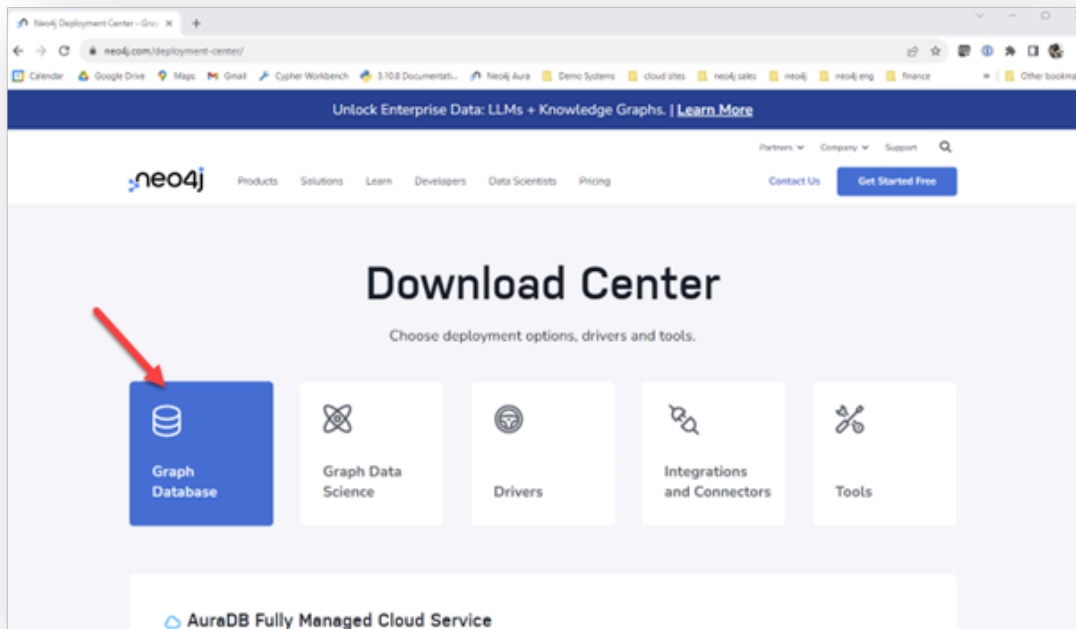
Note that the above commands are RedHat compatible Linux distributions such as CentOS. Debian/Ubuntu distributions will use the “apt get” and similar commands instead.

Download the Neo4j software to a local machine

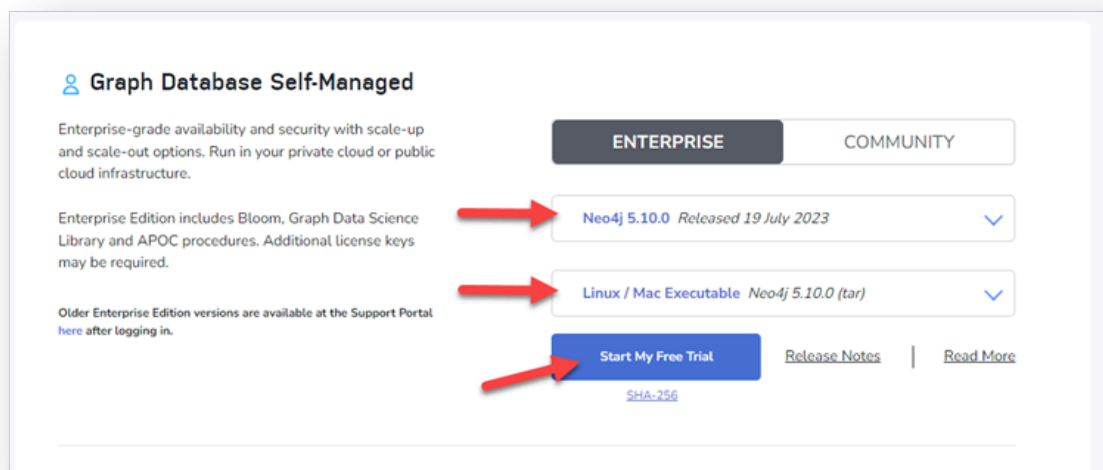
Doing this step in advance makes it faster than waiting for a slow connection during the walk-through of the installation.

Proof Of Concept/Trial Installation

The software can be found at: <https://neo4j.com/deployment-center/> (aka Neo4j Download Center). Select Graph Database:




And then scroll down to the self-managed installation section, pick the OS and distribution desired:




Note that the release defaults to the latest. If you want an earlier version for some reason you can do so. Also be sure to select “ENTERPRISE”. If you select “COMMUNITY”, you will get the OpenSource version which has functional limitations and is not supported by field engineering.

Start a Free 30-Day Trial


Businesses, enterprises and government organizations rely on Neo4j to power their applications.




1,000 TIMES FASTER THAN SQL
Neo4j uses native graph storage and processing engine for efficient results.



OVERCOME SQL PAIN
Simplify difficult-to-write operations in SQL with only a few lines of code.



SCALE WITH AGILITY
Neo4j Enterprise closes the gap between business and IT, allowing for on-the-fly changes as business needs evolve.



GARTNER RATED
Only graph database vendor named to Gartner's Magic Quadrant for Operational Database Management Systems.

Download your free 30-day trial of Neo4j Enterprise Edition.

*

*

*

*

*

Job Function

*

*

Country/Territory

*

Industry

By clicking 'Start My Free Trial' you agree to the [Neo4j Evaluation Agreement for Neo4j Software](#).

Start My Free Trial

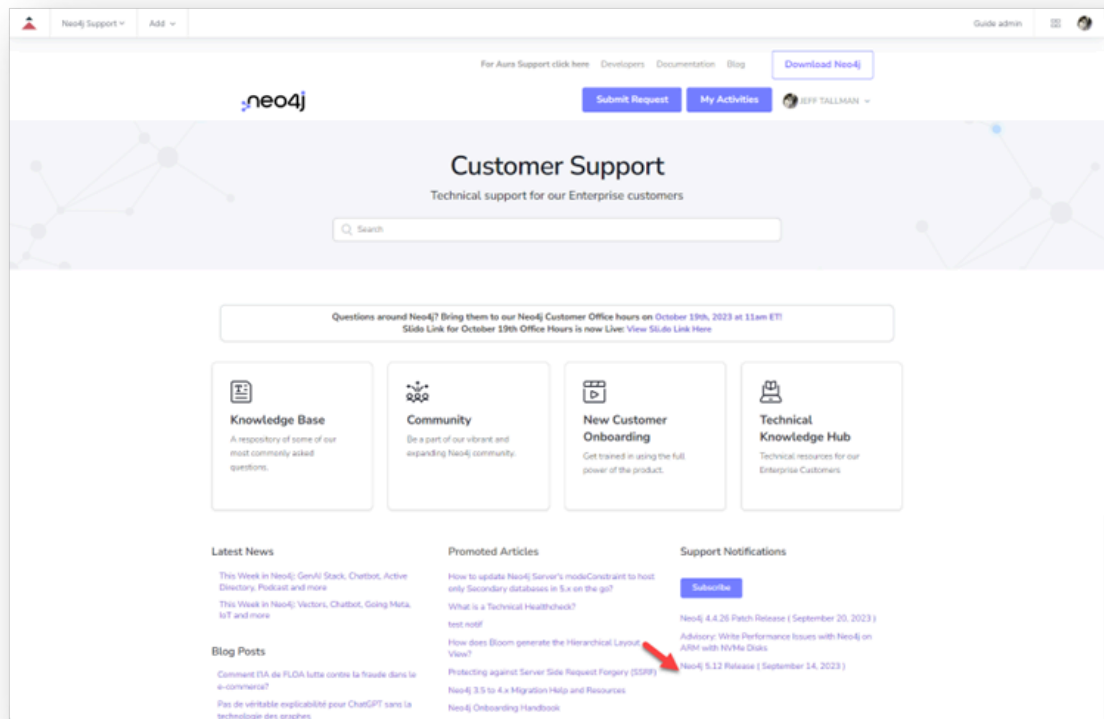
The information you provide will be used in accordance with the terms of our [privacy policy](#).

After selecting the desired options, click “Start My Free Trial”, which will open up a page similar to the below where it will ask for trial information:

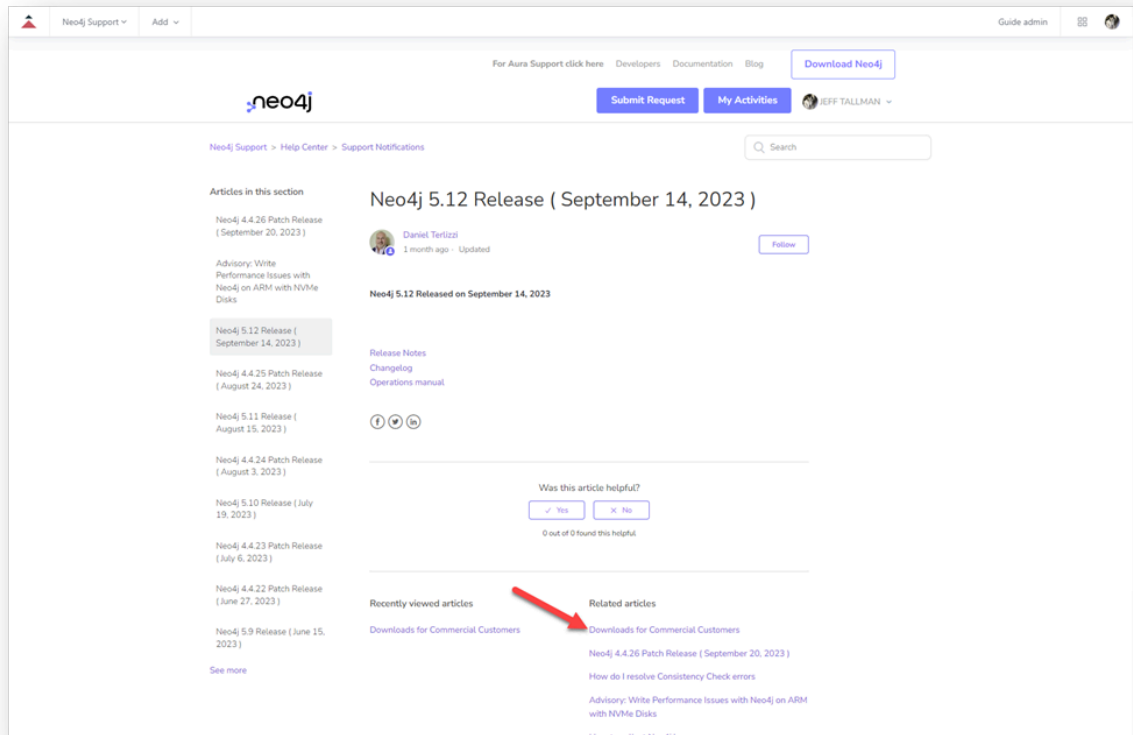
The software should then download.

Licensed Customer Download

Customers have the advantage of downloading without the “spam-me” questionnaire. When you login to the support portal you see a screen similar to:



This should take you to a page similar to:



This will take you to a page with the most recent supported versions. However, you can always just download directly using the following links and altering the versions:

<https://neo4j.com/customer/download/neo4j-enterprise-2025.03.0-windows.zip>
<https://neo4j.com/customer/download/neo4j-enterprise-2025.03.0-unix.tar.gz>

Software Installation

Create installation directories

Note that when installing Neo4j, there is nothing special about a “neo4j” user at the host OS level. You can install Neo4j using any host user that you wish. In these instructions, we will use placeholders such as <username> and <groupname> to indicate you should use the host OS user and group you are using for installation.

Create software directory

It is a security risk to install software that is owned by root, so the process is to not only create the installation directory, but also to change the ownership to be the “neo4j” (or whichever host login will be the owner) as follows:

```
cd /opt
sudo mkdir neo4j
sudo chown <username>:<groupname> neo4j
```

Create data, transaction log directories

This takes a bit more effort if you have created separate devices. Either way, you start with creating the desired data, transaction log and backup directories under the neo4j installation directory by using the following:

```
# assumes you are root (su) to start with
su - <username>
cd /opt/neo4j
mkdir data
mkdir txnlogs
mkdir backup
mkdir licenses
mkdir ssl_certificates
```

(Optional) Mount all devices

If you have created separate devices, you will first need to see what block device each of the storage devices has been mounted as. This can be done by using:

```
sudo lsblk -f
```

Be sure to record the block device (/dev/sd*) and the device UUID for each device. The output may resemble something like the following:

```
$ sudo lsblk -f
NAME        FSTYPE FSVER LABEL UUID                                 FSAVAIL FSUSE% MOUNTPOINTS
sda
??sda1     vfat   FAT16          325B-1692                192.3M    4% /boot/efi
??sda2     xfs                                ce709dde-29ac-41fb-8501-dfe32dae5e9c 195.1G    2% /
sdb         xfs                                f91dcf24-346a-46b9-868f-0603d820253b
sdc         xfs                                fdfd52ae-615f-46e8-867a-b75b489f02a8
sdd         xfs                                f1e9eb97-9845-430b-936e-ef6617050760
```

As you can see, /dev/sda is initialized and has file systems created. You will need to create a file system on all the devices except the boot device (likely /dev/sda) using commands such as:

```
sudo mkfs -t xfs /dev/sdb
sudo mkfs -t xfs /dev/sdc
sudo mkfs -t xfs /dev/sdd
```

Then you will need to add each device to the mount table using the command:

```
sudo vi /etc/fstab
```

And add the newly created devices and appropriate mount points (the directories created earlier):

```
#
# /etc/fstab
# Created by anaconda on Tue Nov 29 07:22:17 2022
#
# Accessible filesystems, by reference, are maintained under '/dev/disk/'.
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info.
#
# After editing this file, run 'systemctl daemon-reload' to update systemd
# units generated from this file.
#
UUID=ce709dde-29ac-41fb-8501-dfe32dae5e9c /                xfs      defaults
0 0
UUID=325B-1692 /boot/efi                vfat
defaults,uid=0,gid=0,umask=077,shortname=winnt 0 2
UUID=f91dcf24-346a-46b9-868f-0603d820253b /opt/neo4j/data xfs      noatime
UUID=fdfd52ae-615f-46e8-867a-b75b489f02a8 /opt/neo4j/txnlogs xfs      noatime
UUID=f1e9eb97-9845-430b-936e-ef6617050760 /opt/neo4j/backup xfs      noatime
```

Then mount and verify the file systems using the commands:

```
sudo mount -a
sudo lsblk -f
```

Sometimes after mounting filesystems the first time, it defaults the ownership back to root. Go to the neo4j directory and check the ownership via `ls -l`. If set back to root, change the ownership to neo4j using the commands:

```
cd /opt/neo4j
sudo chown <username>:<groupname> backup
sudo chown <username>:<groupname> data
sudo chown <username>:<groupname> txnlogs
```

Untar the tgz/tar file

Simply copy/ftp the tarball to the /opt/neo4j (or what ever installation directory you chose) and extract using the normal tar commands such as:

```
tar -xzvf neo4j-enterprise-2025.03.0-unix.tar.gz
```

Configure the instance.

Set the NEO4J_HOME environment variable

For windows users, open the system control panel and add the environment variable NEO4J_HOME and set the value to the software installation directory such as c:\neo4j\neo4j-enterprise-2025.03.0. For Linux users, modify the \$HOME\.bashrc file and add a line

```
export NEO4J_HOME=/opt/neo4j/neo4j-enterprise-2025.03.0
export JAVA_HOME=/etc/alternatives/java_sdk_17
```

And then re-source the .bashrc file to have it take effect using (notice the leading “.”):

```
. $HOME\.bashrc
```

Set the initial password/license the server

For Linux users, cd to ./neo4j-enterprise-2025.03.0/bin directory. Then issue the following commands (replace the initial password with any 8 character or larger password):

```
cd $NEO4J_HOME/bin
./neo4j-admin dbms set-initial-password <any8+charword>
./neo4j-admin server license --accept-evaluation
```

Note that this is just the initial password to get the instance booted. After booting, you can change the admin user’s password as well as add other logins. If this is not a trial, and a full installation, the last command changes to:

```
./neo4j-admin server license --accept-commercial
```

For trials/POC’s, the --accept-evaluation will run for 30 days. If you need longer, at the end of 30 days, simply run the same command but with --extend-evaluation option - it will extend for another 90 days. If that is not enough, you can do only one more extension using --extend-evaluation for another 90 days (7 months total).

License option	License time	Total
initial --accept-evaluation	30 days	30 days
first extension --extend-evaluation	90 days	120 days
second extension --extend-evaluation	90 days	210 days
--accept-commercial	(no limit)	(perpetual)

Edit the configuration file

Note: Most of the time, Neo4j is being installed in a customer's cloud environment. In those situations, the customer may have a dedicated access point to the cloud provider in which cloud systems are added to the corporation DNS. In such cases, cloud external IP addresses are not available and even cloud internal IP addresses are assigned via the DNS to avoid ephemeral issues. As a result, during configuration, all the values will need to use the DNS fully qualified domain name (aka fqdn) instead of IP address. For installations that use IP addresses instead, there is a difference between the external IP address that is visible outside the cloud provider and cloud internal IP addresses that can only be used within the cloud provider's environment. As a result, the configurations below will specify <external-ip-or-fqdn> or <internal-ip-or-fqdn>.

In the \$NEO4J_HOME/conf directory, edit the neo4j.conf file and add the following line at the top:

```
server.config.strict_validation.enabled=false
```

Then scroll to the bottom of the neo4j.conf file and add the following lines. Replace the values with those appropriate.

```
server.directories.data=/opt/neo4j/data
server.directories.metrics=/opt/neo4j/backup/metrics
server.directories.transaction.logs.root=/opt/neo4j/txnlogs
server.directories.dumps.root=/opt/neo4j/backup
server.directories.licenses=/opt/neo4j/licenses
server.directories.import=/opt/neo4j/import
```

```
server.memory.heap.initial_size=16g
server.memory.heap.max_size=16g
server.memory.pagecache.size=80g
```

```
# if the server default is set, you don't need to specify the ip address
# or fqdn for the protocol entries unless you want to restrict a
# protocol to only using a single IP/network interface of multiple
# ones available. Otherwise, just enter the port for protocol addresses
server.default_listen_address=0.0.0.0
server.default_advertised_address=<external-ip-or-fqdn>
```

```

server.bolt.enabled=true
server.bolt.listen_address=:7687
server.bolt.advertised_address=:7687
server.http.enabled=true
server.http.listen_address=:7474
server.http.advertised_address=:7474

db.logs.query.enabled=INFO
db.logs.query.threshold=250ms

#the following setting is for dev/test only
#production systems should use a setting based on 2x backup frequency
db.tx_log.rotation.retention_policy=1 hours

```

Enabling Parallel Query runtime

If planning on supporting parallel query runtime, add the following line:

```

# specify a positive integer smaller than number of vcpu's
# a good starting point is 50% of the vcpu's on machine
server.cypher.parallel.worker_limit=8

```

Vector index additional configuration settings

If planning on using Neo4j's vector index (common for GraphRAG/GenAI applications) you may wish to make the following configuration changes:

```

# increase heap memory for vector support at least +16-32GB above normal
server.memory.heap.initial_size=32g
server.memory.heap.max_size=32g
server.memory.pagecache.size=64g

# enable SIMD vector math (will be removed after incubator phase)
server.jvm.additional=--add-modules jdk.incubator.vector

```

In Neo4j, lists and arrays are serialized in storage and in page cache. When queried, the lists/arrays are unserialized in heap memory. For Neo4j vectors, the amount of heap memory you will need may be dependent on the number of vectors, vector byte size and the vector dimension. For example, with a common 32-bit (4 byte) vector dimension of 1536, 1 million vectors will need $1536 \times 4 \times 1000000 = 6144000000 = \sim 5.5\text{GB}$ to fully instantiate in heap. The vector index will also need about the same, however, it is likely that a query will mainly access the vector index and not both. Since heap memory in Neo4j is per user process, this will allocate and deallocate as vectors are accessed, but concurrent users could increase heap usage much more than without vector indexes. This can be monitored via the database metrics file `neo4j.dbms.vm.heap.used.csv`.

GDS additional configuration settings

If planning on using GDS enterprise edition, add the following lines:

```
gds.enterprise.license_file=/opt/neo4j/licenses/<filename>.lic
gds.model.store_location=/opt/neo4j/GDS/models
gds.export.location=/opt/neo4j/GDS/export
gds.progress_tracking_enabled=true
gds.arrow.enabled=true
gds.arrow.listen_address=0.0.0.0:8491
gds.arrow.advertised_listen_address=<external-ip-or-fqdn>:8491
```

Note that the default for most licenses would be in the folder /opt/neo4j/neo4j-enterprise-2025.03.0/licenses and generally you can simply put <filename>.license in the neo4j.conf without specifying the path and the server will look in that folder by default.

While this works, with each upgrade/patch, you have to remember to copy the license files - easily forgotten. By creating a ./licenses folder in the parent /opt/neo4j folder and specifying the full path, you avoid this step and the likely “forgetting” of copying the license files.

GDS & Bloom server plugin additional configuration settings

If planning on using the Bloom plugin or are using GDS (which includes some number of Bloom licenses currently as part of the GDS license), add the following lines:

```
dbms.bloom.license_file=/opt/neo4j/licenses/bloom-server-plugin.license
server.unmanaged_extension_classes=com.neo4j.bloom.server=/browser/bloom
dbms.security.http_auth_allowlist=/,/browser.*,/bloom.*
dbms.bloom.authorization_role=admin,reader
dbms.security.procedures.unrestricted=apoc.*,bloom.*,gds.*
```

Similar to the comments above for GDS, the default for most licenses would be in the folder /opt/neo4j/neo4j-enterprise-2025.03.0/licenses and generally you can simply put <filename>.license in the neo4j.conf without specifying the path and the server will look in that folder by default.

While this works, with each upgrade/patch, you have to remember to copy the license files - easily forgotten. By creating a ./licenses folder in the parent /opt/neo4j folder and specifying the full path, you avoid this step and the likely “forgetting” of copying the license files.

Cluster additional configuration settings

If creating a cluster, add the following lines. On each instance, replace the advertised_address configuration with the corresponding hostname/IP address for that instance:

```
#5.23.0 and later versions should use V2 discovery service
dbms.cluster.discovery.resolver_type=LIST
dbms.cluster.discovery.version=V2_ONLY
dbms.cluster.discovery.v2.endpoints=<internal-ip-or-fqdn>:6000,<internal-ip-or-fqdn>:6000,10.<internal-ip-or-fqdn>:6000
```



```

server.cluster.listen_address=0.0.0.0:6000
server.cluster.advertised_address=<internal-ip-or-fqdn>:6000
server.cluster.raft.listen_address=0.0.0.0:7000
server.cluster.raft.advertised_address=<internal-ip-or-fqdn>:7000
server.routing.listen_address=0.0.0.0:7688
server.routing.advertised_address=<internal-ip-or-fqdn>:7688

# NONE, PRIMARY or SECONDARY. For nodes in a cluster using GDS, you will need
# to set this to SECONDARY. Otherwise, the constraint should be either NONE or
# PRIMARY. Note that if you set it to PRIMARY and are running a large cluster,
# it will prevent the node from hosting secondary copies of databases -
#
# e.g. create database xyzdb topology 3 primaries 2 secondaries
#
# in the above situation, if the mode_constraint is set to PRIMARY, the
# secondary copies could not utilize this node (essentially, NONE = either).
initial.server.mode_constraint=NONE

# The default number of system databases is 3. If setting up a cluster with
# fewer than 3 nodes, you will need to set the below to 2
# dbms.cluster.minimum_initial_system_primaries_count=3
initial.dbms.default_primaries_count=3

```

Save and exit the file.

Install the basic plugins

There are several very common plugins that can optionally be installed. They are shipped with the product. Strongly recommended is to first add the core APOC procedures/functions by copying the APOC core jar file from the labs subdirectory ala command similar to:

```
cp $NEO4J_HOME/labs/apoc*core.jar $NEO4J_HOME/plugins
```

It is also highly recommended to also copy the Graph Data Science (GDS) plugin. Without a license, it will operate in Community Edition mode with some restrictions, but at least you will be able to exploit the GDS algorithms for some analysis.

```
cp $NEO4J_HOME/products/neo4j-graph-data-science*.jar $NEO4J_HOME/plugins
```

If planning on using Bloom, you can also install the bloom plugin via:

```
cp $NEO4J_HOME/products/bloom*.jar $NEO4J_HOME/plugins
```

If planning on using any GenAI features such as integration with cloud AI tools for creating vectors, you should install the GenAI plugin:

```
cp $NEO4J_HOME/products/neo4j-genai*.jar $NEO4J_HOME/plugins
```

Other plugins are available via the Neo4j Labs github repositories and can be optionally installed, such as NeoSemantics (for parsing RDF/ontology files), extended APOC procedures, Natural Language Processing, etc.

(Recommended) Change query and security log formats to JSON

The query log (query.log) and security log (security.log) are often searched or parsed for analysis reasons. This is made much easier if the logs are in json format vs. the default free text format. To change the log format, you edit the \$NEO4J_HOME/conf/server-logs.xml file (same directory as neo4j.conf) and change the PatternLayout item lines ~44 and ~54. The author prefers to comment out the originals using XML commenting so that you can revert back if desired. A sample snippet of the changes is below:

```
<RollingRandomAccessFile name="QueryLog" fileName="${config:server.directories.logs}/query.log"
    filePattern="${config:server.directories.logs}/query.log.%02i">
    <!--<PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/>-->
    <JsonTemplateLayout eventTemplateUri="classpath:org/neo4j/logging/StructuredJsonLayout.json"/>
    <Policies>
        <SizeBasedTriggeringPolicy size="100 MB"/>
    </Policies>
    <DefaultRolloverStrategy fileIndex="min" max="10"/>
</RollingRandomAccessFile>

<RollingRandomAccessFile name="SecurityLog" fileName="${config:server.directories.logs}/security.log"
    filePattern="${config:server.directories.logs}/security.log.%02i">
    <!--<PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSSZ}{GMT+0} %-5p %m%n"/> -->
    <JsonTemplateLayout eventTemplateUri="classpath:org/neo4j/logging/StructuredLayoutWithMessage.json"/>
    <Policies>
        <SizeBasedTriggeringPolicy size="100 MB"/>
    </Policies>
    <DefaultRolloverStrategy fileIndex="min" max="3"/>
</RollingRandomAccessFile>
```

The original formatting PatternLayout elements are commented out as shown in light gray above using the XML comment characters <!-- (start comment) and --> (end comment). The JsonTemplateLayout element is then provided (dark red). These default JSON layouts can be found in the documentation - but for ease of implementation, the above ones work with Neo4j 5.x up through 5.23 as of this document.

(Optional) Create the Systemd/systemctl Service

Normally, you can start/stop the neo4j process manually by using:

```
$NEO4J_HOME/bin/neo4j [start|stop]
```

While this method does background (nohup actually) the process so it keeps running after you have disconnected, if the OS is rebooted, Neo4j will not automatically restart. To enable this functionality, you need to create a service for Neo4j.

For Linux systems, you can create a service for systemctl, but at this point, this document doesn't contain the steps. Note that if you opt to install via rpm vs. tar file, the package manager will automatically set up a service for you.

A bit of background information. The systemctl utility normally requires sudo access to create as well as start/stop services. In some environments, the neo4j administrators may not have sudo access except during the initial installation. In addition, starting the service using sudo

often results in neo4j running as root which exposes a number of potential security issues and typically is forbidden in sites that audit their security infrastructure.

There are two options:

1. Create the service as normal, but specify user & group settings in the service file so that when systemctl runs even as sudo that it starts the process as the neo4j user
2. Use the systemctl --user mode which allows any user to create services

This document describes the latter. The Neo4j Operations Guide 5.0 has an example of the former at

<https://neo4j.com/docs/operations-manual/current/installation/linux/tarball/#linux-tarball-start-automatically> :

```
[Unit]
Description=Neo4j Graph Database
After=network-online.target
Wants=network-online.target

[Service]
ExecStart=/opt/neo4j/bin/neo4j console
Restart=on-abnormal
User=<username>
Group=<groupname>
Environment="NEO4J_CONF=/opt/neo4j/conf" "NEO4J_HOME=/opt/neo4j"
LimitNOFILE=60000
TimeoutSec=120

[Install]
WantedBy=multi-user.target
```

While the documentation doesn't mention this, the implication of the above is that it is run as root (assumption based on the LimitNOFILE setting which is unnecessary if running as the user and the /etc/security/limits.conf file edited, the User/Group entries....and subsequent commands in the documentation merely call systemctl directly vs. sudo systemctl.), so this document describes an alternative approach that addresses some of the (author's) perceived issues with the above:

- Likely requires running as root
- Has to be updated with each upgrade of neo4j or use symbolic links
- Runs in console mode (not sure why)
- No stop (ExecStop) method

Create start/stop script files

This step simplifies upgrading/patching neo4j in that you won't need to edit the service itself. The assumption is that each patch/update to neo4j will be installed in a new directory to

facilitate an immediate rollback if there are any issues. Simply create start/stop shell scripts in /opt/neo4j base directory similar to:

```
#!/bin/sh
#export JAVA_HOME=/etc/alternatives/java_sdk_21
export NEO4J_HOME=/opt/neo4j/neo4j-enterprise-2025.03.0
$NEO4J_HOME/bin/neo4j start --verbose

#!/bin/sh
#export JAVA_HOME=/etc/alternatives/java_sdk_21
export NEO4J_HOME=/opt/neo4j/neo4j-enterprise-2025.03.0
$NEO4J_HOME/bin/neo4j stop --verbose
```

Setting JAVA_HOME may not be necessary if you only have a single Java JDK environment on the machine. If you are running multiple java environments, set the JAVA_HOME to the desired Java JDK environment you want to use for Neo4j. After creating each file, make sure to set the execute bit as normal for shell scripts using:

```
chmod 755 <start script.sh>
chmod 755 <stop script.sh>
```

Create the service unit file

When using systemctl --user mode, the service file has to be located in the user's home directory in a subdirectory:

```
/home/<username>/.config/systemd/user
```

The .config subdirectory likely exists, but you may have to create the rest of the path ala:

```
cd ~/.config
mkdir systemd
cd systemd
mkdir user
cd user
```

In the user subdirectory, create a file with a .service extension (e.g. neo4j_svc.service) similar to the following contents:

```
[Unit]
Description=Restarts the neo4j database server after reboots
After=network.target

[Service]
Type=forking
ExecStart=/opt/neo4j/start_neo4j.sh
ExecStop=/opt/neo4j/stop_neo4j.sh
```

```
Restart=no
```

```
[Install]
```

```
WantedBy=default.target
```

Note the highlighted entries should point to the earlier stop/start shell scripts you created. One comment on `Restart=no` - if this is set to the default, the systemd daemon will automatically try to restart the database if it detects that it isn't running. This can lead to some fun during upgrades or get into a bit of a ping-pong cycling if there is an issue. Best practice is to not automatically restart and have manual intervention from the DBA to investigate the cause of any failures.

Then you enable the service via the command:

```
systemctl --user enable neo4j_svc.service
```

You can immediately test the syntax of your UNIT file via a status check ala:

```
systemctl --user status neo4j_svc.service
```

If there are errors, it will report some form of error message. A common cause is not having `Type=forking` which is required when systemd is invoking a shell script

Starting/Stopping the service

The service can simply be started or stopped using the following commands:

```
systemctl --user start neo4j_svc.service  
systemctl --user stop neo4j_svc.service
```

Note that if you start it as a service, it is best to stop the service vs. using `./bin/neo4j stop`. If the Neo4j instance is already running, attempting to start it via `systemctl` will fail - just like attempting to start it manually when already running. A fast way to check if it is running is to use the command:

```
ps -ef | grep neo4j-enterprise
```

...and look for a java process with a long pathname behind it.