The Neo4j Operations Manual v4.0

Table of Contents

1. Introduction . . . 2
  1.1. Editions . . . 2
    1.1.1. Community Edition . . . 2
    1.1.2. Enterprise Edition . . . 2
    1.1.3. Feature details . . . 2
2. Installation . . . 4
  2.1. System requirements . . . 4
    2.1.1. Supported platforms . . . 4
    2.1.2. Hardware requirements . . . 4
    2.1.3. Software requirements . . . 6
    2.1.4. Filesystem . . . 6
    2.1.5. Java . . . 6
  2.2. Neo4j Desktop . . . 6
  2.3. Linux installation . . . 7
    2.3.1. Debian . . . 7
      Installation . . . 8
      Upgrade . . . 10
      File locations . . . 10
      Operation . . . 10
    2.3.2. Deploy Neo4j using the Neo4j RPM package . . . 11
      Install on Red Hat, CentOS, Fedora or Amazon Linux . . . 11
      Install on SUSE . . . 13
    2.3.3. Linux tarball installation . . . 13
      Unix console application. . . . 14
      Linux service . . . 14
      Setting the number of open files. . . . 14
    2.3.4. Neo4j system service . . . 15
      Configuration . . . 15
      Starting the service automatically on system start. . . . 15
      Controlling the service . . . 16
      Log . . . 16
  2.4. macOS installation . . . 16
    2.4.1. Unix console application . . . 16
    2.4.2. macOS service . . . 17
    2.4.3. macOS file descriptor limits . . . 17
  2.5. Windows installation . . . 17
    2.5.1. Windows console application . . . 17
    2.5.2. Windows service . . . 18
      Java options . . . 18
    2.5.3. Windows PowerShell module . . . 19
      System requirements . . . 19
      Managing Neo4j on Windows . . . 19
      How do I import the module? . . . 19
      How do I get help about the module? . . . 19
      Example usage . . . 20
      Common PowerShell parameters . . . 20

3. Docker . . . 21
3.1. Introduction . . . 21
3.1.1. Ports . . . 21
3.1.2. Volumes . . . 21
3.1.3. Running Neo4j as a non-root user . . . 22
3.1.4. Neo4j editions . . . 22
Neo4j Enterprise Edition license . . . 22
3.2. Configuration . . . 23
3.2.1. Environment variables . . . 23
Neo4j Enterprise Edition . . . 24
3.2.2. */conf* volume . . . 24
3.2.3. Building a new image . . . 24
3.3. Clustering . . . 25
3.3.1. Setting up a Causal Cluster . . . 25
3.4. Docker specific operations . . . 26
3.4.1. Using */plugins* . . . 26
3.4.2. Using Cypher Shell . . . 26
3.4.3. Upgrading Neo4j on Docker . . . 27
3.5. Security . . . 27
3.5.1. Encryption . . . 27
4. Configuration . . . 28
4.1. The neo4j.conf file . . . 28
4.1.1. Introduction . . . 28
4.1.2. Syntax . . . 28
4.1.3. JVM-specific configuration settings . . . 29
4.1.4. List currently active settings . . . 29
4.2. File locations . . . 29
4.2.1. Where to find important files . . . 30
4.2.2. Log files . . . 30
4.2.3. File permissions . . . 31
4.3. Ports . . . 31
4.4. Set an initial password . . . 32
4.5. Password and user recovery . . . 33
4.6. Configure connectors . . . 34
4.6.1. Available connectors . . . 34
4.6.2. Configuration options . . . 34
4.6.3. Options for Bolt thread pooling . . . 35
4.6.4. Defaults for addresses . . . 35
4.7. Dynamic settings . . . 36
4.7.1. Introduction . . . 37
4.7.2. Discover dynamic settings . . . 37
4.7.3. Update dynamic settings . . . 37
4.7.4. Dynamic settings reference . . . 38
4.8. Transaction logs . . . 39
5. Manage databases . . . 42
5.1. Introduction . . . 42
5.1.1. Concepts . . . 42
5.1.2. The `system` database . . . 43
5.1.3. The default database . . . 44

5.2. Administration and configuration . . . 44
5.2.1. Administrative commands . . . 45
5.2.2. Configuration parameters . . . 45
5.3. Queries . . . 46
5.3.1. Show the status of a specific database . . . 46
5.3.2. Show the status of all databases. . . . 47
5.3.3. Show the status of the default database . . . 47
5.3.4. Create a database . . . 47
5.3.5. Stop a database . . . 48
5.3.6. Start a database. . . . 49
5.3.7. Drop, or remove a database . . . 50
5.4. Databases in a Causal Cluster . . . 50
5.4.1. Running Cypher administrative commands from Cypher Shell on a Causal Cluster. . . . 51
6. Clustering. . . . 54
6.1. Introduction . . . 54
6.1.1. Overview . . . 54
6.1.2. Operational view . . . 55
Core Servers . . . 56
Read Replicas . . . 56
6.1.3. Causal consistency . . . 56
6.1.4. Summary. . . . 57
6.2. Deploy a cluster. . . . 58
6.2.1. Introduction . . . 58
6.2.2. Configure a Core-only cluster . . . 58
6.2.3. Add a Core Server to an existing cluster . . . 59
6.2.4. Add a Read Replica to an existing cluster . . . 60
6.3. Seed a cluster . . . 60
6.3.1. Introduction . . . 61
6.3.2. Seed from backups . . . 61
6.3.3. Seed using the import tool. . . . 62
6.4. Discovery . . . 63
6.4.1. Overview . . . 63
Discovery using a list of server addresses . . . 63
Discovery using DNS with multiple records. . . . 63
Discovery in Kubernetes. . . . 64
6.5. Intra-cluster encryption . . . 64
6.5.1. Introduction . . . 65
6.5.2. Example deployment . . . 65
Generate and install cryptographic objects . . . 65
Configure the cluster SSL policy . . . 66
Validate the secure operation of the cluster . . . 67
6.6. Internals of clustering. . . . 67
6.6.1. Elections and leadership . . . 68
6.6.2. Multi-database and the reconciler . . . 68
6.6.3. Store copy. . . . 68
6.6.4. On-disk state . . . 69
6.7. Settings reference . . . 69
6.7.1. Multi-data center settings . . . 72
7. Fabric . . . 73

7.1. Introduction . . . 73
7.1.1. Overview . . . 73
7.1.2. Fabric concepts . . . 73
The fabric database . . . 73
Fabric graphs . . . 74
7.1.3. Deployment examples . . . 74
Development deployment . . . 74
Cluster deployment with no single point of failure . . . 75
Multi-cluster deployment . . . 75
7.2. Configuration . . . 76
7.2.1. Fabric database setup . . . 76
Development setup example . . . 76
Naming graphs . . . 77
Cluster setup with no single point of failure example . . . 77
Cluster routing context . . . 78
7.2.2. Authentication and authorization . . . 79
Credentials . . . 79
User and role administration . . . 79
Privileges on the Fabric database . . . 79
7.2.3. Important settings . . . 79
System settings . . . 79
Graph settings . . . 80
Drivers settings . . . 80
7.3. Queries . . . 81
7.3.1. Query a single graph . . . 81
7.3.2. Query multiple graphs . . . 82
7.3.3. Query all graphs . . . 82
7.3.4. Query result aggregation . . . 82
7.3.5. Correlated subquery . . . 82
7.3.6. Updating query . . . 83
7.3.7. Mapping functions . . . 83
7.3.8. Fabric built-in functions . . . 84
7.4. Further considerations . . . 84
8. Upgrade . . . 86
8.1. Upgrade planning . . . 86
8.1.1. Supported upgrade paths . . . 86
8.1.2. Limitations . . . 86
8.1.3. Prepare to upgrade . . . 87
8.2. Upgrade a single instance . . . 87
8.3. Upgrade a Causal Cluster . . . 88
9. Backup . . . 91
9.1. Backup planning . . . 91
9.1.1. Introduction . . . 91
9.1.2. Online and offline backups . . . 92
9.1.3. Server configuration . . . 92
9.1.4. Databases to backup . . . 92
9.1.5. Storage considerations . . . 93
9.1.6. Cluster considerations . . . 93
9.1.7. Using SSL/TLS for backups . . . 93

9.1.8. Additional files to back up 94
9.2. Perform a backup 94
9.2.1. Backup command 94
9.2.2. Backup process 95
9.2.3. Memory configuration 96
9.3. Restore a backup 97
9.3.1. Restore command 97
9.3.2. Restore a standalone server 97
9.3.3. Restore a cluster 98
10. Authentication and authorization 99
10.1. Introduction 99
10.2. Built-in roles 100
10.3. Fine-grained access control 101
10.3.1. The data model 102
10.3.2. Security 103
10.3.3. Access control using built-in roles 104
10.3.4. Sub-graph access control using privileges 105
Privileges of `itadmin` 105
Privileges of `researcher` 106
Privileges of `doctor` 109
Privileges of `receptionist` 110
Privileges of nurses 111
10.4. Integration with LDAP 112
10.4.1. Introduction 113
10.4.2. Configure the LDAP auth provider 113
Configuration for Active Directory 113
Configuration for openLDAP 114
10.4.3. Use `ldapsearch` to verify the configuration 115
10.4.4. The auth cache 116
10.4.5. Available methods of encryption 117
Use LDAP with encryption via StartTLS 117
Use LDAP with encrypted LDAPS 117
10.4.6. Use a self-signed certificate in a test environment 117
10.5. Manage procedure permissions 118
10.5.1. Introduction 118
10.5.2. Configuration steps 118
Create a custom role 118
Manage procedure permissions 119
10.6. Terminology 120
11. Security 122
11.1. Securing extensions 122
11.1.1. Sandboxing 122
11.1.2. White listing 123
11.2. SSL framework 123
11.2.1. Introduction 124
11.2.2. Certificates 124
11.2.3. Configuration 124
Settings 124
11.2.4. Choosing an SSL provider 126

11.2.5. Terminology . . . 126
11.3. Browser credentials handling . . . 128
11.4. Security checklist . . . 128
12. Monitoring . . . 130
12.1. Metrics . . . 130
12.1.1. Types of metrics . . . 131
Global metrics . . . 131
Database metrics . . . 132
12.1.2. Expose metrics . . . 132
Enable metrics logging . . . 132
Graphite . . . 133
Prometheus . . . 133
CSV files . . . 133
JMX Beans . . . 133
12.1.3. Metrics reference . . . 134
General-purpose metrics . . . 134
Metrics specific to Causal Clustering . . . 136
Java Virtual Machine Metrics . . . 138
12.2. Logging . . . 138
12.2.1. Query logging . . . 139
Log configuration . . . 139
Attach metadata to a query . . . 140
12.2.2. Security events logging . . . 141
Log configuration . . . 141
12.3. Query management . . . 142
12.3.1. List all running queries . . . 142
12.3.2. List all active locks for a query . . . 144
12.3.3. Terminate multiple queries . . . 145
12.3.4. Terminate a single query . . . 146
12.4. Transaction management . . . 147
12.4.1. Configure transaction timeout . . . 147
12.4.2. Configure lock acquisition timeout . . . 148
12.4.3. List all running transactions . . . 148
12.5. Connection management . . . 150
12.5.1. List all network connections . . . 150
12.5.2. Terminate multiple network connections . . . 151
12.5.3. Terminate a single network connection . . . 152
12.6. Monitoring a Causal Cluster . . . 153
12.6.1. Procedures for monitoring a Causal Cluster . . . 153
Find out the role of a cluster member . . . 154
Gain an overview over the instances in the cluster . . . 154
Get routing recommendations . . . 155
12.6.2. Endpoints for status information . . . 156
Adjusting security settings for Causal Clustering endpoints . . . 156
Unified endpoints . . . 157
12.7. Monitoring individual database states . . . 160
12.7.1. Listing Databases . . . 160
12.7.2. Listing a single database . . . 164
13. Performance . . . 166

13.1. Memory configuration . . . 166
13.1.1. Overview . . . 166
13.1.2. Considerations . . . 167
13.1.3. Capacity planning . . . 168
13.1.4. Configure query heap usage . . . 169
13.2. Index configuration . . . 170
13.2.1. Introduction . . . 170
13.2.2. B-tree indexes . . . 171
Limitations . . . 171
Index migration . . . 173
Procedures to create index and index backed constraint . . . 173
13.2.3. Full-text indexes . . . 174
Configuration . . . 174
13.3. Tuning of the garbage collector . . . 174
13.4. Bolt thread pool configuration . . . 176
13.4.1. How thread pooling works . . . 176
13.4.2. Configuration options . . . 176
13.4.3. How to size your Bolt thread pool . . . 176
13.5. Linux file system tuning . . . 177
13.6. Disks, RAM and other tips . . . 177
13.6.1. Storage . . . 178
13.6.2. Page cache . . . 178
Active page cache warmup . . . 178
Checkpoint IOPS limit . . . 178
13.7. Statistics and execution plans . . . 179
13.7.1. Statistics . . . 179
13.7.2. Execution plans . . . 180
14. Tools . . . 182
14.1. Neo4j Admin . . . 182
14.1.1. Introduction . . . 182
14.1.2. Syntax and commands . . . 182
14.1.3. Environment variables . . . 183
14.1.4. Exit codes . . . 184
14.2. Consistency checker . . . 184
14.2.1. Check consistency of a database or a backup . . . 184
14.3. Report tool . . . 186
14.4. Display store information . . . 187
14.5. Memory recommendations . . . 188
14.6. Import . . . 189
14.6.1. Syntax . . . 190
14.6.2. CSV file header format . . . 190
Header files . . . 191
Properties . . . 191
Node files . . . 193
Relationship files . . . 194
Using ID spaces . . . 195
Skipping columns . . . 196
Import compressed files . . . 197
14.6.3. Options . . . 197

Output . . . 201
14.7. Dump and load databases . . . 201
14.8. Unbind a Core Server . . . 202
14.9. Copy a database . . . 203
14.10. Cypher Shell . . . 205
14.10.1. About Cypher Shell . . . 206
14.10.2. Syntax . . . 206
14.10.3. Query parameters . . . 208
14.10.4. Transactions . . . 208
14.10.5. Procedures . . . 209
Appendix A: Reference . . . 211
A.1. Configuration settings . . . 211
A.2. Built-in procedures . . . 275
A.2.1. Procedures, editions and modes . . . 276
A.2.2. Enterprise Edition procedures . . . 276
A.2.3. Community Edition procedures . . . 289
Appendix B: Tutorials . . . 296
B.1. Set up a local Causal Cluster . . . 296
B.2. Use the Import tool . . . 302
B.2.1. Basic example . . . 303
B.2.2. Customizing configuration options . . . 304
B.2.3. Using separate header files . . . 305
B.2.4. Multiple input files . . . 306
Using regular expressions for specifying multiple input files . . . 307
B.2.5. Types and labels . . . 307
Using the same label for every node . . . 307
Using the same relationship type for every relationship . . . 308
B.2.6. Property types . . . 309
B.2.7. ID handling . . . 309
Working with sequential or auto incrementing identifiers . . . 310
B.2.8. Bad input data . . . 310
Relationships referring to missing nodes . . . 310
Multiple nodes with same ID within same ID space . . . 311
Appendix C: Advanced Causal Clustering . . . 313
C.1. Causal Clustering lifecycle . . . 313
C.1.1. Introduction . . . 313
C.1.2. Discovery protocol . . . 314
C.1.3. Core membership . . . 314
C.1.4. Read Replica membership . . . 315
C.1.5. Transacting via the Raft protocol . . . 316
C.1.6. Catchup protocol . . . 316
C.1.7. Read Replica shutdown . . . 317
C.1.8. Core shutdown . . . 318
C.2. Multi-data center . . . 318
C.2.1. Licensing for multi-data center operations . . . 319
C.2.2. Multi-data center design . . . 319
Introduction . . . 319
Core Server deployment scenarios . . . 319
Allowing Read Replicas to catch up from other Read Replicas . . . 322

C.2.3. Multi-data center operations . . . 326
Enable multi-data center operations . . . 327
Server groups . . . 327
Strategy plugins . . . 328
C.2.4. Multi-data center load balancing . . . 332
Introduction . . . 332
Prerequisite configuration . . . 333
The load balancing framework . . . 333
Load balancing examples . . . 335
C.2.5. Data center disaster recovery . . . 336
Data center loss scenario . . . 337
Procedure for recovering from data center loss . . . 338
C.3. Advanced settings reference . . . 339
Appendix D: Deprecated security procedures . . . 340
D.1. Enterprise Edition . . . 340
D.1.1. Activate a suspended user . . . 341
D.1.2. Assign a role to the user . . . 342
D.1.3. Change the current user’s password . . . 342
D.1.4. Change the given user’s password . . . 342
D.1.5. Create a new role . . . 343
D.1.6. Create a new user . . . 344
D.1.7. Delete the specified role . . . 345
D.1.8. Delete the specified user . . . 345
D.1.9. List all available roles . . . 346
D.1.10. List all roles assigned to the specified user . . . 347
D.1.11. List all local users . . . 347
D.1.12. List all users currently assigned the specified role . . . 348
D.1.13. Unassign a role from the user . . . 349
D.1.14. Suspend the specified user . . . 350
D.1.15. Show the current user . . . 350
D.1.16. List roles per procedure . . . 351
D.2. Community Edition . . . 352
D.2.1. Change the current user’s password . . . 353
D.2.2. Add a user . . . 353
D.2.3. Delete a user . . . 353
D.2.4. List all native users . . . 354
D.2.5. Show details for the current user . . . 355

© 2019 Neo4j, Inc.

License: Creative Commons 4.0

This is the operations manual for Neo4j version 4.0, authored by the Neo4j Team.

This manual covers the following areas:

- Introduction — Introduction of Neo4j Community and Enterprise Editions.
- Installation — Instructions on how to install Neo4j in different deployment contexts.
- Docker — Instructions on how to use Neo4j on Docker.
- Configuration — Instructions on how to configure certain parts of Neo4j.
- Manage databases — Instructions on how to manage multiple active databases with Neo4j.
- Clustering — Comprehensive descriptions of Neo4j Causal Clustering.
- Fabric — Instructions on how to configure and use Neo4j Fabric.
- Upgrade — Instructions on upgrading Neo4j.
- Backup — Instructions on setting up Neo4j backups.
- Authentication and authorization — Instructions on user management and role-based access control.
- Security — Instructions on server security.
- Monitoring — Instructions on setting up Neo4j monitoring.
- Performance — Instructions on how to go about performance tuning for Neo4j.
- Tools — Description of Neo4j tools.
- Reference — Listings of all Neo4j configuration parameters.
- Tutorials — Step-by-step instructions on various scenarios for setting up Neo4j.
- Advanced Causal Clustering — Advanced concepts and actions for Neo4j Causal Clustering.
- Deprecated security procedures — Deprecated security procedures.

Who should read this?

This manual is written for:

- the engineer performing the Neo4j production deployment.
- the operations engineer supporting and maintaining the Neo4j production database.
- the enterprise architect investigating database options.
- the infrastructure architect planning the Neo4j production deployment.

Chapter 1. Introduction

This chapter introduces Neo4j.

Neo4j is the world’s leading graph database. Its architecture is designed for optimal management, storage and traversal of nodes and relationships. The database takes a property graph approach which is beneficial for both traversal performance and operations runtime. Neo4j offers dedicated memory management as well as memory efficient operations.

It is scalable and can be deployed as a standalone server or scaled out across multiple machines in a fault-tolerant cluster for production environments. Other features for production applications include hot backups and extensive monitoring.

Cypher is a declarative query language for graphs. Learn the details in the Cypher Manual. The recommended way of interacting with the database programmatically is either through the official drivers, or using the Java API.

1.1. Editions

There are two editions of Neo4j to choose from: *Community Edition* and *Enterprise Edition*:

1.1.1. Community Edition

The Community Edition is a fully functional edition of Neo4j, suitable for single instance deployments. It has full support for key Neo4j features, such as ACID compliance, Cypher, and programming APIs. It is ideal for learning Neo4j, for do-it-yourself projects, and for applications in small workgroups.

1.1.2. Enterprise Edition

The Enterprise Edition extends the functionality of Community Edition to include key features for performance and scalability, such as a clustering architecture and online backup functionality. Additional security features include role-based access control and LDAP support; for example, Active Directory. It is the choice for production systems with requirements for scale and availability, such as commercial solutions and critical internal solutions.

1.1.3. Feature details

Table 1. Features

Edition	Community	Enterprise
Property graph model	✓	✓
Native graph processing & storage	✓	✓
ACID transactions	✓	✓
Cypher graph query language	✓	✓
Neo4j Browser with syntax highlighting	✓	✓
Bolt binary protocol	✓	✓
Language drivers for C#, Java, JavaScript & Python [1]	✓	✓
High-performance native API	✓	✓
High-performance caching	✓	✓

Edition	Community	Enterprise
Cost-based query optimizer	▯	▯
Graph algorithms library to support AI initiatives [1]	▯	▯
Fast writes via native label indexes	▯	▯
Composite indexes	▯	▯
Full-text node & relationship indexes	▯	▯
Slotted and *Pipelined* Cypher runtimes	-	▯
Property-existence constraints	-	▯
Node Key constraints	-	▯
Listing and terminating running queries	-	▯
Auto-reuse of space	-	▯
Role-based access control	-	▯
Subgraph access control	-	▯
LDAP and Active Directory integration	-	▯
Kerberos security option	-	▯

[1]Note that these need to be downloaded and installed separately.

Table 2. Performance & Scalability

Edition	Community	Enterprise
Causal Clustering for global scale applications	-	▯
Enterprise lock manager accesses all cores on server	-	▯
Intra-cluster encryption	-	▯
Offline backups	▯	▯
Online backups	-	▯
Encrypted backups	-	▯
Rolling upgrades	-	▯
Automatic cache warming	-	▯
Routing and load balancing with Neo4j Drivers	-	▯
Advanced monitoring	-	▯
Graph size limitations	34B nodes, 34B relationships, 68B properties	No limit
Bulk import tool	▯	▯
Bulk import tool, resumable	-	▯

Chapter 2. Installation

This chapter describes installation of Neo4j in different deployment contexts, such as Linux, macOS, and Windows.

The topics described are:

- System requirements — The system requirements for a production deployment of Neo4j.
- Neo4j Desktop — About Neo4j Desktop
- Linux — Installation instructions for Linux.
- macOS — Installation instructions for macOS.
- Windows — Installation instructions for Windows.

As an alternative to installation, you can also run Neo4j in a Docker container. For information on running Neo4j on Docker, see Docker.

2.1. System requirements

This section provides an overview of the system requirements for running Neo4j in a production environment.

Neo4j can be installed in many environments and for different scopes, therefore system requirements largely depends on the use of the software. This section distinguishes between a personal/development installation, and a server-based installation.

This section contains the following:

- Supported platforms
- Hardware requirements
- Software requirements
- Filesystem
- Java

2.1.1. Supported platforms

Neo4j is supported on systems with x86_64 architectures, whether they are a physical, virtual, or containerized environments.

2.1.2. Hardware requirements

In terms of hardware requirements, follow these guidelines:

Table 3. Hardware requirement guidelines.

CPU	Performance is generally memory or I/O bound for large graphs, and compute bound for graphs that fit in memory.

Memory	More memory allows for larger graphs, but it needs to be configured properly to avoid disruptive garbage collection operations.
Storage	Aside from capacity, the performance characteristics of the disk are the most important when selecting storage: • Neo4j workloads tend significantly toward random reads. • Select media with low average seek time: SSD over spinning disks. • Consult Disks, RAM and other tips for more details.

For personal use and software development:

Table 4. Hardware requirement guidelines for personal use and software development.

CPU	Intel Core i3 minimum, Intel Core i7 recommended.
Memory	2GB minimum, 16GB or more recommended.
Storage	10GB SATA Minimum, SSD with SATA Express or NVMe recommended.

For cloud environments:

Table 5. Hardware requirement guidelines for cloud environments.

CPU	2vCPU minimum, 16+ recommended, possibly Xeon processors.
Memory	2GB minimum, size depends on workloads: in some cases, it is recommended to use instances with memory that fits the size of the graph in use.
Storage	10GB minimum block storage, attached NVMe SSD recommended. Storage size depends on the size of the databases.

For server-based, on-premise environments:

Table 6. Hardware requirement guidelines for server-based, on-premise environments.

CPU	Intel Xeon processors.
Memory	8GB minimum, size depends on workloads; in some cases, it is recommended to use instances with memory that fits the size of the graph in use.
Storage	SATA i7.2K RPM 6Gbps Hard Drive minimum, NVMe SSD recommended. Storage size depends on the size of the databases.

2.1.3. Software requirements

For personal use and software development:

Table 7. Software requirements for personal use and software development.

Operating System	Supported JDK
MacOS 10.14+	ZuluJDK 11 or OpenJDK 11
Ubuntu Desktop 16.04+	ZuluJDK 11, OpenJDK 11 or OracleJDK 11
Fedora 29+	OpenJDK 11 or ZuluJDK 11
Debian 9+	OpenJDK 11 or OracleJDK 11
Windows 10	ZuluJDK 11 or OracleJDK 11

For cloud environments, and server-based, on-premise environments:

Table 8. Software requirements for cloud environments, and server-based, on-premise environments.

Operating System	Supported JDK
Ubuntu Server 16.04+	OpenJDK 11 or OracleJDK 11
Red Hat Enterprise Linux Server 7.5+	OpenJDK 11 or Red Hat OpenJDK 11
CentOS Server 7.5+	OpenJDK 11
SUSE Linux Enterprise Server 15+	OracleJDK 11
Amazon Linux AMI 2018.03+	Amazon Corretto 11, OpenJDK 11 or OracleJDK 11
Windows Server 2012+	OracleJDK 11

2.1.4. Filesystem

For proper ACID behavior, the filesystem must support flush (*fsync*, *fdatasync*). See Linux file system tuning for a discussion on how to configure the filesystem in Linux for optimal performance.

2.1.5. Java

Neo4j Desktop is available for developers and personal users.

Neo4j Desktop includes a Java Virtual Machine, *JVM*, for convenience. All other versions of Neo4j require Java to be pre-installed.

2.2. Neo4j Desktop

This section introduces Neo4j Desktop.

Neo4j Desktop is a convenient way for developers to work with local Neo4j databases.

To install Neo4j Desktop, go to Neo4j Download Center and follow the instructions.

While most functionality is the same, the instructions in this manual are not written for Neo4j Desktop. For example, file locations for a database installed via Neo4j Desktop will be different from those described here.

Neo4j Desktop is not suited for production environments.

2.3. Linux installation

This section describes how to install Neo4j on Linux using Debian or RPM packages, or from a Tar archive.

This section describes the following:

* Install Neo4j on Debian and Debian-based distributions
  * Installation
  * Upgrade
  * File locations
  * Operation
* Deploy Neo4j using the Neo4j RPM package
  * Install on Red Hat, CentOS, Fedora or Amazon Linux
    * Standard installation
    * Non-interactive installation of Neo4j Enterprise Edition
    * Offline installation
  * Install on SUSE
* Install Neo4j on Linux from a tarball
  * Unix console application
  * Linux service
  * Setting the number of open files
* Install Neo4j as a system service
  * Configuration
  * Controlling the service
  * Log

2.3.1. Debian

This section describes how to install Neo4j on Debian, and Debian-based distributions like Ubuntu, using the Neo4j Debian package.

This section describes the following:

* Installation
  * Prerequisites (Oracle Java, Debian 9 and Ubuntu 16.04 only)
  * Add the repository
  * Install Neo4j
* Upgrade
* File locations
* Operation

Installation

To install Neo4j on Debian you need to make sure of the following:

- An OpenJDK Java 11 runtime is installed or available through your package manager.
- The repository containing the Neo4j Debian package is known to the package manager.

Java Prerequisites (Oracle Java, Debian 9+ and Ubuntu 16.04+ only)

Neo4j 4.0 requires the Java 11 runtime. Java 11 is not included in Ubuntu 16.04 LTS or Debian 9 (stretch) and will have to be set up manually prior to installing or upgrading to Neo4j 4.0, as described below. Debian 9 users can find OpenJDK 11 in backports. Debian 10 and Ubuntu 18.04 onwards already have the Openjdk Java 11 package available through `apt`.

Oracle Java and Debian

Neo4j is compatible with Oracle Java on Debian/Ubuntu Linux, but should be installed via tarball. The Debian installer may still be used, but it will install OpenJDK Java 11 in addition to any existing Java installations.

This is due to changes in Oracle's Debian package manifest between Java versions 8 and 11.

Java 11 on Debian 9

Add the line `deb http://httpredir.debian.org/debian stretch-backports main` to a file with the ".list" extension in */etc/apt/sources.list.d/*. Then run `apt-get update`:

```
echo "deb http://httpredir.debian.org/debian stretch-backports main" | sudo tee -a
/etc/apt/sources.list.d/stretch-backports.list
sudo apt-get update
```

You are now ready to install Neo4j, which will install Java 11 automatically if it is not already installed. See Dealing with multiple installed Java versions to make sure you can start Neo4j after install.

Java 11 on Ubuntu 16.04

Add the official OpenJDK package repository to `apt`:

```
sudo add-apt-repository -y ppa:openjdk-r/ppa
sudo apt-get update
```

You are now ready to install Neo4j, which will install Java 11 automatically if it is not already installed. See Dealing with multiple installed Java versions to make sure you can start Neo4j after install.

Dealing with multiple installed Java versions

It is important that you configure your default Java version to point to Java 11, or Neo4j 4.0.0 will be unable to start. Do so with the `update-java-alternatives` command.

- First list all your installed version of Java with `update-java-alternatives --list`

  Your results may vary, but this is an example of the output:

  ```
  java-1.11.0-openjdk-amd64 1071 /usr/lib/jvm/java-1.11.0-openjdk-amd64
  java-1.8.0-openjdk-amd64 1069 /usr/lib/jvm/java-1.8.0-openjdk-amd64
  ```

- Identify your Java 11 version, in this case it is `java-1.11.0-openjdk-amd64`. Then set it as the default with (replacing `<java11name>` with the appropriate name from above)

  ```
  sudo update-java-alternatives --jre --set <java11name>
  ```

Add the repository

The Debian package is available from https://debian.neo4j.com.

- To use the repository for generally available versions of Neo4j, run:

  ```
  wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
  echo 'deb https://debian.neo4j.com stable latest' | sudo tee -a /etc/apt/sources.list.d/neo4j.list
  sudo apt-get update
  ```

To avoid the risk of the `apt` package manager accidentally forcing a database upgrade, different major and minor releases of Neo4j are also available separately inside the repository. To install Neo4j this way, specify the major and minor version required, in place of `latest`. We recommend the following method for production or business critical installations:

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
echo 'deb https://debian.neo4j.com stable {neo4j-version}' | sudo tee -a
/etc/apt/sources.list.d/neo4j.list
sudo apt-get update
```

- To use the repository for milestone releases, alpha, beta, release candidate and preview versions of Neo4j, run:

  ```
  wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
  echo 'deb https://debian.neo4j.com testing latest' | sudo tee -a /etc/apt/sources.list.d/neo4j.list
  sudo apt-get update
  ```

- Once the repository has been added into `apt`, you can verify which Neo4j versions are available by running:

  ```
  apt list -a neo4j
  ```

Install Neo4j

To install Neo4j Community Edition:

```
sudo apt-get install neo4j=1:4.0.0
```

To install Neo4j Enterprise Edition:

```
sudo apt-get install neo4j-enterprise=1:4.0.0
```

Note that the version includes an epoch version component (`1:`), in accordance with the Debian policy on versioning.

Versions of Neo4j that are not yet generally available may differ in naming.

The naming structure of packages are normally composed as `neo4j-enterprise=1:<version>~<release>`. For example, Neo4j Enterprise Edition Milestone Release 3 would be: `neo4j-enterprise=1:4.0.0~beta03mr03`.

Refer to the download page for more information regarding the name of packages.

When installing Neo4j Enterprise Edition, you will be prompted to accept the license agreement. Once the license agreement is accepted installation begins. Your answer to the license agreement prompt will be remembered for future installations on the same system.

To forget the stored answer, and trigger the license agreement prompt on subsequent installation, use `debconf-communicate` to purge the stored answer:

```
echo purge | sudo debconf-communicate neo4j-enterprise
```

Non-interactive installation of Neo4j Enterprise Edition

For Neo4j Enterprise Edition, the license agreement is presented in an interactive prompt. If you require non-interactive installation of Neo4j Enterprise Edition, you can indicate that you have read and accepted the license agreement using `debconf-set-selections`:

```
echo "neo4j-enterprise neo4j/question select I ACCEPT" | sudo debconf-set-selections
echo "neo4j-enterprise neo4j/license note" | sudo debconf-set-selections
```

Upgrade

For upgrade of any 3.x version of Neo4j to 4.0.0, follow instructions in Upgrade.

File locations

File locations for all Neo4j packages are documented *here*.

Operation

Most Neo4j configuration goes into *neo4j.conf*.

For operating systems using `systemd`, some package-specific options are set in *neo4j.service* and can be edited using `systemctl edit neo4j.service`.

For operating systems that are not using `systemd`, some package-specific options are set in */etc/default/neo4j*.

Environment variable	Default value	Details
NEO4J_SHUTDOWN_TIMEOUT	*120*	Timeout in seconds when waiting for Neo4j to stop. If it takes longer than this then the shutdown is considered to have failed. This may need to be increased if the system serves long-running transactions.
NEO4J_ULIMIT_NOFILE	*60000*	Maximum number of file handles that can be opened by the Neo4j process.

2.3.2. Deploy Neo4j using the Neo4j RPM package

This section describes how to deploy Neo4j using the Neo4j RPM package on Red Hat, CentOS, Fedora, or Amazon Linux distributions.

This section describes the following:

- Install on Red Hat, CentOS, Fedora or Amazon Linux
  - Standard installation
  - Non-interactive installation of Neo4j Enterprise Edition
  - Offline installation
- Install on SUSE

Install on Red Hat, CentOS, Fedora or Amazon Linux

Standard installation

1. Add the repository.

   Use the following as `root` to add the repository:

```
rpm --import https://debian.neo4j.com/neotechnology.gpg.key
cat <<EOF>  /etc/yum.repos.d/neo4j.repo
[neo4j]
name=Neo4j RPM Repository
baseurl=https://yum.neo4j.com/stable
enabled=1
gpgcheck=1
EOF
```

   For milestone, alpha, beta, and release candidate releases, replace the `baseurl` line with `baseurl=https://yum.neo4j.org/testing`

2. Ensure the correct Java version.

   Neo4j 4.0 requires the Java 11 runtime. Most of our supported RPM Linux distributions have Java 11 available by default. There is some minor setup required for Amazon Linux, and for compatibility with Oracle Java 11.

   - Java 11 on Amazon Linux.

     To enable OpenJDK 11 on Amazon Linux run the shell command:

```
amazon-linux-extras enable java-openjdk11
```

     You are now ready to install Neo4j 4.0.0, which will install Java 11 automatically if it is not already installed.

   - Oracle Java 11.

     Oracle and OpenJDK provide incompatible RPM packages for Java 11. We provide an adapter for Oracle Java 11 which must be installed before Neo4j. The adapter contains no code, but will stop the package manger from installing OpenJDK 11 as a dependency despite an existing Java 11 installation.

This step assumes that you have performed the previous step to set up the yum repository.

a. Download and install the Oracle java 11 JDK from the Oracle website.
b. Install the adapter:

```
sudo yum install https://dist.neo4j.org/neo4j-java11-adapter.noarch.rpm
```

The SHA-256 of the adapter package can be verified against https://dist.neo4j.org/neo4j-java11-adapter.noarch.rpm.sha256.

You are now ready to install Neo4j 4.0.0.

3. Install Neo4j.
   - To install Neo4j Community Edition as `root`:

```
yum install neo4j-4.0.0
```

   - To install Neo4j Enterprise Edition as `root`:

```
yum install neo4j-enterprise-4.0.0
```

If you are installing a milestone, alpha, beta, or release candidate release, the name of the package is `neo4j-enterprise-<version>-0.<release>.1`. For example, Neo4j Enterprise Edition Milestone Release 3 would be: `neo4j-enterprise-4.0.0-0.beta03mr03.1`

4. Run the following to return the version and edition of Neo4j that has been installed:

```
rpm -qa | grep neo
```

Non-interactive installation of Neo4j Enterprise Edition

When installing Neo4j Enterprise Edition, you will be required to accept the license agreement before installation is allowed to complete. This is an interactive prompt. If you require non-interactive installation of Neo4j Enterprise Edition, you can indicate that you have read and accepted the license agreement by setting the environment variable `NEO4J_ACCEPT_LICENSE_AGREEMENT` to `yes`:

```
NEO4J_ACCEPT_LICENSE_AGREEMENT=yes yum install neo4j-enterprise-4.0.0
```

Offline installation

If you cannot reach `https://yum.neo4j.org/stable`, perhaps due to a firewall, you will need to obtain Neo4j via an alternative machine which has the relevant access, and then move the RPM package manually.

It is important to note that using this method will mean that the offline machine will not receive the dependencies that are that are normally downloaded and installed automatically when using RPM for installing Neo4j; Cypher Shell and Java (if not installed already):

- The Cypher Shell RPM package can be downloaded from Neo4j Download Center.
- For information on supported versions of Java, see System requirements.

1. Run the following to obtain the required RPM package:
   - Neo4j Enterprise Edition:

     ```
     curl -O http://yum.neo4j.org/stable/neo4j-enterprise-4.0.0-1.noarch.rpm
     ```

   - Neo4j Community Edition:

     ```
     curl -O http://yum.neo4j.org/stable/neo4j-4.0.0-1.noarch.rpm
     ```

2. Manually move the downloaded RPM package to the offline machine.
3. Run the following on the offline machine to install Neo4j:

   ```
   rpm -i <rpm file name>
   ```

Install on SUSE

SUSE is not certified for production use. These instructions are provided for convenience for those wishing to use Neo4j in non-production environments.

For SUSE-based distributions the steps are as follows:

1. Use the following as `root` to add the repository:

   ```
   zypper addrepo --refresh https://yum.neo4j.org/stable neo4j-repository
   ```

2. Install Neo4j.
   - To install Neo4j Community Edition as `root`:

     ```
     zypper install neo4j-4.0.0
     ```

   - To install Neo4j Enterprise Edition as `root`:

     ```
     zypper install neo4j-enterprise-4.0.0
     ```

2.3.3. Linux tarball installation

This section describes how to install Neo4j on Linux from a tarball, and run it as a console application or service.

Unix console application

1. Download the latest release from Neo4j Download Center.

   Select the appropriate tar.gz distribution for your platform.

2. Check that the SHA hash of the downloaded file is correct:

   a. To find the correct SHA hash, go to Neo4j Download Center and click on `SHA-256` which will be located below your downloaded file.

   b. Using the appropriate commands for your platform, display the `SHA-256` hash for the file that you downloaded.

   c. Ensure that the two are identical.

3. Extract the contents of the archive, using `tar -xf <filename>`

   Refer to the top-level extracted directory as: NEO4J_HOME

4. Change directory to: `$NEO4J_HOME`

   Run `./bin/neo4j console`

5. Stop the server by typing `Ctrl-C` in the console.

Linux service

If you want to run Neo4j as a system service, you can install either the Debian or RPM package.

For more information on configuring and operating the Neo4j system service, see Neo4j system service.

Setting the number of open files

Linux platforms impose an upper limit on the number of concurrent files a user may have open. This number is reported for the current user and session with the `ulimit -n` command:

```
user@localhost:~$ ulimit -n
1024
```

It is possible that the default value of 1024 may not be not enough. This is especially true when many indexes are used or a server installation sees too many connections. Network sockets count against the limit as well. Users are therefore encouraged to increase the limit to a healthy value of 40000 or more, depending on usage patterns.

It is possible to set the limit with the `ulimit` command, but only for the root user, and it only affects the current session. To set the value system wide, follow the instructions for your platform.

The following steps will set the open file descriptor limit to 60000 for user *neo4j* under Ubuntu 16.04 LTS, Debian 8, Centos 7 or later versions of those operating systems:

1. If you run Neo4j as a service, you must complete the following:

   a. Run the following command, which will enable you to edit the *neo4j.service* file:

   ```
   user@localhost:~$ sudo systemctl edit neo4j.service
   ```

b. Append the `[Service]` section in the *neo4j.service* file:

```
[Service]
LimitNOFILE=60000
```

2. If you run Neo4j as an interactive user (which can be the case for testing purposes), you must complete the following:

   a. Run the following command, which will enable you to edit the *user.conf* file:

   ```
   user@localhost:~$ sudo vi /etc/systemd/user.conf
   ```

   b. Uncomment and define the value for `DefaultLimitNOFILE`, found in the `[MANAGER]` section:

   ```
   [Manager]
   ...
   DefaultLimitNOFILE=60000
   ```

   c. Run the following command which will append the */etc/security/limits.conf* file:

   ```
   user@localhost:~$ sudo vi /etc/security/limits.conf
   neo4j   soft    nofile  60000
   neo4j   hard    nofile  60000
   ```

3. Reload `systemd` settings with the command:

   ```
   user@localhost:~$ sudo systemctl daemon-reload
   ```

4. Reboot your machine.

After completing the above procedure, the Neo4j user will have a limit of 60000 simultaneous open files. If you continue experiencing exceptions on `Too many open files` or `Could not stat() directory`, you may have to raise the limit further.

2.3.4. Neo4j system service

This article covers configuring and operating the Neo4j system service. It assumes that your system has `systemd`, which is the case for most Linux distributions.

Setting the number of open files.

For instructions on how to set the number of concurrent files that a user can have open, see Setting the number of open files.

Configuration

Configuration is stored in */etc/neo4j/neo4j.conf*. See File locations for a complete catalog of where files are found for the various packages.

Starting the service automatically on system start

If you installed the RPM package and want Neo4j to start automatically on system boot then you need to enable the service. On Debian-based distributions this is done for you at installation time.

```
systemctl enable neo4j
```

Controlling the service

System services are controlled with the `systemctl` command. It accepts a number of commands:

```
systemctl {start|stop|restart} neo4j
```

Service customizations can be placed in a service override file. To edit your specific options, do the following command which will open up an editor of the appropriate file:

```
systemctl edit neo4j
```

Then place any customizations under a `[Service]` section. The following example lists default values which may be interesting to change for some users:

```
[Service]
# The user and group which the service runs as.
User=neo4j
Group=neo4j
# If it takes longer than this then the shutdown is considered to have failed.
# This may need to be increased if the system serves long-running transactions.
TimeoutSec=120
```

You can print the effective service, including possible overrides, with:

```
systemctl cat neo4j
```

Remember to restart neo4j if you change any settings.

```
systemctl restart neo4j
```

Log

The neo4j log is written to `journald` which can be viewed using the `journalctl` command:

```
journalctl -e -u neo4j
```

`journald` automatically rotates the log after a certain time and by default it commonly does not persist across reboots. Please see `man journald.conf` for further details.

2.4. macOS installation

This section describes how to install Neo4j on macOS.

2.4.1. Unix console application

1. Download the latest release from Neo4j Download Center.

   Select the appropriate tar.gz distribution for your platform.

2. Check that the SHA hash of the downloaded file is correct:
   a. To find the correct SHA hash, go to Neo4j Download Center and click on `SHA-256` which will be located below your downloaded file.
   b. Using the appropriate commands for your platform, display the `SHA-256` hash for the file that you downloaded.
   c. Ensure that the two are identical.
3. Extract the contents of the archive, using `tar -xf <filename>`

   Refer to the top-level extracted directory as: NEO4J_HOME
4. Change directory to: `$NEO4J_HOME`

   Run `./bin/neo4j console`
5. Stop the server by typing `Ctrl-C` in the console.

When Neo4j runs in console mode, logs are printed to the terminal.

2.4.2. macOS service

Use the standard macOS system tools to create a service based on the `neo4j` command.

2.4.3. macOS file descriptor limits

The limit of *open file descriptors* may have to be increased if a database has many indexes or if there are many connections to the database. The currently configured open file descriptor limitation on your macOS system can be inspected with the `launchctl limit maxfiles` command. The method for changing the limit may differ depending on the version of macOS. Consult the documentation for your operating system in order to find out the appropriate command.

If you raise the limit above 10240, then you must also add the following setting to your *neo4j.conf* file:

```
dbms.jvm.additional=-XX:-MaxFDLimit
```

Without this setting, the file descriptor limit for the JVM will not be increased beyond 10240. Note, however, that this only applies to macOS. On all other operating systems, you should always leave the `MaxFDLimit` JVM setting enabled.

2.5. Windows installation

This section describes how to install Neo4j on Windows.

2.5.1. Windows console application

1. Download the latest release from Neo4j Download Center.

   Select the appropriate ZIP distribution.
2. Check that the SHA hash of the downloaded file is correct:
   a. To find the correct SHA hash, go to Neo4j Download Center and click on `SHA-256` which will be located below your downloaded file.

   b. Using the appropriate commands for your platform, display the `SHA-256` hash for the file that you downloaded.
   c. Ensure that the two are identical.
3. Right-click the downloaded file, click Extract All.
4. Change directory to the top-level extracted directory.

   Run `bin\neo4j console`
5. Stop the server by typing `Ctrl-C` in the console.

2.5.2. Windows service

Neo4j can also be run as a Windows service. Install the service with `bin\neo4j install-service`, and start it with `bin\neo4j start`.

The available commands for `bin\neo4j` are: `help`, `start`, `stop`, `restart`, `status`, `install-service`, `uninstall-service`, and `update-service`.

Java options

When Neo4j is installed as a service, Java options are stored in the service configuration. Changes to these options after the service is installed will not take effect until the service configuration is updated. For example, changing the setting `dbms.memory.heap.max_size` in *neo4j.conf* will not take effect until the service is updated and restarted. To update the service, run `bin\neo4j update-service`. Then restart the service to run it with the new configuration.

The same applies to the path to where Java is installed on the system. If the path changes, for example when upgrading to a new version of Java, it is necessary to run the `update-service` command and restart the service. Then the new Java location will be used by the service.

Example 1. Update service example

1. Install service

```
bin\neo4j install-service
```

2. Change memory configuration

```
echo dbms.memory.heap.initial_size=8g >> conf\neo4j.conf
echo dbms.memory.heap.max_size=16g >> conf\neo4j.conf
```

3. Update service

```
bin\neo4j update-service
```

4. Restart service

```
bin\neo4j restart
```

2.5.3. Windows PowerShell module

The Neo4j PowerShell module allows administrators to:

- Install, start and stop Neo4j Windows® Services.
- Start tools, such as `Neo4j Admin` and `Cypher Shell`.

The PowerShell module is installed as part of the ZIP file distributions of Neo4j.

System requirements

- Requires PowerShell v2.0 or above.
- Supported on either 32 or 64 bit operating systems.

Managing Neo4j on Windows

On Windows, it is sometimes necessary to *Unblock* a downloaded ZIP file before you can import its contents as a module. If you right-click on the ZIP file and choose "Properties" you will get a dialog which includes an "Unblock" button, which will enable you to import the module.

Running scripts has to be enabled on the system. This can, for example, be achieved by executing the following from an elevated PowerShell prompt:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
```

For more information, see About execution policies.

The PowerShell module will display a warning if it detects that you do not have administrative rights.

How do I import the module?

The module file is located in the *bin* directory of your Neo4j installation, i.e. where you unzipped the downloaded file. For example, if Neo4j was installed in *C:\Neo4j* then the module would be imported like this:

```
Import-Module C:\Neo4j\bin\Neo4j-Management.psd1
```

This will add the module to the current session.

Once the module has been imported you can start an interactive console version of a Neo4j Server like this:

```
Invoke-Neo4j console
```

To stop the server, issue `Ctrl-C` in the console window that was created by the command.

How do I get help about the module?

Once the module is imported you can query the available commands like this:

```
Get-Command -Module Neo4j-Management
```

The output should be similar to the following:

```
CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Function        Invoke-Neo4j                        4.0.0      Neo4j-Management
Function        Invoke-Neo4jAdmin                   4.0.0      Neo4j-Management
Function        Invoke-Neo4jBackup                  4.0.0      Neo4j-Management
Function        Invoke-Neo4jImport                  4.0.0      Neo4j-Management
Function        Invoke-Neo4jShell                   4.0.0      Neo4j-Management
```

The module also supports the standard PowerShell help commands.

```
Get-Help Invoke-Neo4j
```

Run the following to see examples of help commands:

```
Get-Help Invoke-Neo4j -examples
```

Example usage

- List of available commands:

  ```
  Invoke-Neo4j
  ```

- Current status of the Neo4j service:

  ```
  Invoke-Neo4j status
  ```

- Install the service with verbose output:

  ```
  Invoke-Neo4j install-service -Verbose
  ```

- Available commands for administrative tasks:

  ```
  Invoke-Neo4jAdmin
  ```

Common PowerShell parameters

The module commands support the common PowerShell parameter of `Verbose`.

Chapter 3. Docker

This chapter describes how run Neo4j in a Docker container.

This chapter describes the following:

- Introduction — Introduction to running Neo4j in a Docker container.
- Configuration — How to configure Neo4j to run in a Docker container.
- Clustering — How to set up Causal Clustering when using Docker.
- Docker specific operations - Descriptions of various operations that are specific to using Docker.
- Security - Information about using encryption with the Docker image.

Docker does not run natively on macOS or Windows. For running Docker on macOS and Windows, please consult the documentation provided by Docker.

3.1. Introduction

An introduction to how Neo4j runs in a Docker container.

The Neo4j Docker image, and instructions on how to start using it, can be found here: https://hub.docker.com/_/neo4j/.

3.1.1. Ports

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

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

Note that when Docker is used, Neo4j is configured automatically to allow remote access to the HTTP, HTTPS, and Bolt services. This is different than the default Neo4j configuration, where the HTTP, HTTPS, and Bolt services do not allow remote connections. For more information on configuring connections, see Configure connectors.

3.1.2. Volumes

The Docker image also exposes the following volumes. Directories on the host can be mounted using the `--volume` option. See File locations for details about the directories used by default in different Neo4j distributions.

- */conf*
- */data*
- */import*
- */logs*
- */metrics*
- */plugins*
- */ssl*

It is often desirable to keep database and logs outside of the container. The following command will start a container with ports for Bolt and HTTP published, and with the */data* and */logs* volumes mapped to directories on the host.

```
docker run \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    neo4j:4.0
```

Point your browser at `http://localhost:7474` on Linux or `http://$(docker-machine ip default):7474` on macOS.

All the volumes in this documentation are stored under `$HOME` in order to work on macOS where `$HOME` is automatically mounted into the machine VM. On Linux the volumes can be stored anywhere.

By default Neo4j requires authentication and requires you to login with `neo4j/neo4j` at the first connection and set a new password. You can set the password for the Docker container directly by specifying `--env NEO4J_AUTH=neo4j/<password>` in your run directive. Alternatively, you can disable authentication by specifying `--env NEO4J_AUTH=none` instead.

3.1.3. Running Neo4j as a non-root user

For security reasons Neo4j will run as the `neo4j` user inside the container. You can specify which user to run as by invoking docker with the `--user` argument. For example, the following would run Neo4j as your current user:

```
docker run \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --user="$(id -u):$(id -g)" \
    neo4j:4.0
```

3.1.4. Neo4j editions

Tags are available for both Community Edition and Enterprise Edition. Version-specific Enterprise Edition tags have an `-enterprise` suffix, for example: `neo4j:4.0.0-enterprise`. Community Edition tags have no suffix, for example `neo4j:4.0.0`. The latest Neo4j Enterprise Edition release is available as `neo4j:enterprise`.

Neo4j Enterprise Edition license

In order to use Neo4j Enterprise Edition you must accept the license agreement.

> © Network Engine for Objects in Lund AB. 2018. All Rights Reserved. Use of this Software without a proper commercial license with Neo4j, Inc. or its affiliates is prohibited.
>
> Email inquiries can be directed to: licensing@neo4j.com
>
> More information is also available at: https://neo4j.com/licensing/

To accept the license agreement set the environment variable `NEO4J_ACCEPT_LICENSE_AGREEMENT=yes`.

To do this you can use the following docker argument:

```
--env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes
```

3.2. Configuration

This chapter describes how configure Neo4j to run in a Docker container.

The default configuration provided by this image is intended for learning about Neo4j, but must be modified to make it suitable for production use. In particular, the default memory assignments to Neo4j are very limited (`NEO4J_dbms_memory_pagecache_size=512M` and `NEO4J_dbms_memory_heap_max__size=512M`), to allow multiple containers to be run on the same server. You can read more about configuring Neo4j in the Configuration settings.

There are three ways to modify the configuration:

- Set environment variables.
- Mount a */conf* volume.
- Build a new image.

Which one to choose depends on how much you need to customize the image.

3.2.1. Environment variables

Pass environment variables to the container when you run it.

```
docker run \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --env NEO4J_dbms_memory_pagecache_size=4G \
    neo4j:4.0
```

Any configuration value (see Configuration settings) can be passed using the following naming scheme:

- Prefix with `NEO4J_`.
- Underscores must be written twice: `_` is written as `__`.
- Periods are converted to underscores: `.` is written as `_`.

As an example, `dbms.tx_log.rotation.size` could be set by specifying the following argument to Docker:

```
--env NEO4J_dbms_tx__log_rotation_size
```

Variables which can take multiple options, such as `dbms_jvm_additional`, must be defined just once, and include a concatenation of the multiple values. For example:

```
--env NEO4J_dbms_jvm_additional="-Dcom.sun.management.jmxremote.authenticate=true
-Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.password.file=
$HOME/conf/jmx.password -Dcom.sun.management.jmxremote.access.file=$HOME/conf/jmx.access
-Dcom.sun.management.jmxremote.port=3637"
```

Neo4j Enterprise Edition

The following environment variables are specific to Causal Clustering, and are available in the Neo4j Enterprise Edition:

- `NEO4J_dbms_mode`: the database mode, defaults to `SINGLE`, set to `CORE` or `READ_REPLICA` for Causal Clustering.
- `NEO4J_causal__clustering_expected__core__cluster__size`: the initial cluster size (number of Core instances) at startup.
- `NEO4J_causal__clustering_initial__discovery__members`: the network addresses of an initial set of Core cluster members.
- `NEO4J_causal__clustering_discovery__advertised__address`: hostname/ip address and port to advertise for member discovery management communication.
- `NEO4J_causal__clustering_transaction__advertised__address`: hostname/ip address and port to advertise for transaction handling.
- `NEO4J_causal__clustering_raft__advertised__address`: hostname/ip address and port to advertise for cluster communication.

See Clustering for examples of how to configure Causal Clustering on Docker.

3.2.2. */conf* volume

To make arbitrary modifications to the Neo4j configuration, provide the container with a */conf* volume.

```
docker run \
    --detach \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=$HOME/neo4j/data:/data \
    --volume=$HOME/neo4j/logs:/logs \
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:4.0
```

Any configuration files in the */conf* volume will override files provided by the image. So if you want to change one value in a file you must ensure that the rest of the file is complete and correct. Environment variables passed to the container by Docker will still override the values in configuration files in */conf* volume.

If you use a configuration volume you must make sure to listen on all network interfaces. This can be done by setting `dbms.default_listen_address=0.0.0.0`.

To dump an initial set of configuration files, run the image with the `dump-config` command.

```
docker run --rm \
    --volume=$HOME/neo4j/conf:/conf \
    neo4j:4.0 dump-config
```

3.2.3. Building a new image

For more complex customization of the image you can create a new image based on this one.

```
FROM neo4j:4.0
```

If you need to make your own configuration changes, we provide a hook so you can do that in a script:

```
COPY extra_conf.sh /extra_conf.sh
```

Then you can pass in the `EXTENSION_SCRIPT` environment variable at runtime to source the script:

```
docker run -e "EXTENSION_SCRIPT=/extra_conf.sh" cafe12345678
```

When the extension script is sourced, the current working directory will be the root of the Neo4j installation.

3.3. Clustering

This chapter describes how to set up Causal Clustering when running Neo4j in a Docker container.

3.3.1. Setting up a Causal Cluster

In order to run Neo4j in CC mode under Docker you need to wire up the containers in the cluster so that they can talk to each other. Each container must have a network route to each of the others and the `NEO4J_causal__clustering_expected__core__cluster__size` and `NEO4J_causal__clustering_initial__discovery__members` environment variables must be set for cores. Read Replicas only need to define `NEO4J_causal__clustering_initial__discovery__members`.

Within a single Docker host, this can be achieved as follows. Note that the default ports for HTTP, HTTPS and Bolt are used. For each container, these ports are mapped to a different set of ports on the Docker host.

```
docker network create --driver=bridge cluster

docker run --name=core1 --detach --network=cluster \
    --publish=7474:7474 --publish=7473:7473 --publish=7687:7687 \
    --env NEO4J_dbms_mode=CORE \
    --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
    --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_dbms_connector_bolt_advertised__address=localhost:7687 \
    --env NEO4J_dbms_connector_http_advertised__address=localhost:7474 \
    neo4j:4.0-enterprise

docker run --name=core2 --detach --network=cluster \
    --publish=8474:7474 --publish=8473:7473 --publish=8687:7687 \
    --env NEO4J_dbms_mode=CORE \
    --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
    --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_dbms_connector_bolt_advertised__address=localhost:8687 \
    --env NEO4J_dbms_connector_http_advertised__address=localhost:8474 \
    neo4j:4.0-enterprise

docker run --name=core3 --detach --network=cluster \
    --publish=9474:7474 --publish=9473:7473 --publish=9687:7687 \
    --env NEO4J_dbms_mode=CORE \
    --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
    --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
    --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
    --env NEO4J_dbms_connector_bolt_advertised__address=localhost:9687 \
    --env NEO4J_dbms_connector_http_advertised__address=localhost:9474 \
    neo4j:4.0-enterprise
```

Additional instances can be added to the cluster in an ad-hoc fashion. A Read Replica can for example be added with:

```
docker run --name=read_replica1 --detach --network=cluster \
         --publish=10474:7474 --publish=10473:7473 --publish=10687:7687 \
         --env NEO4J_dbms_mode=READ_REPLICA \
         --env NEO4J_causal__clustering_initial__discovery__members=core1:5000,core2:5000,core3:5000 \
         --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
         --env NEO4J_dbms_connector_bolt_advertised__address=localhost:10687 \
         --env NEO4J_dbms_connector_http_advertised__address=localhost:10474 \
         neo4j:4.0-enterprise
```

When each container is running on its own physical machine and Docker network is not used, it is necessary to define the advertised addresses to enable communication between the physical machines. Each container should also bind to the host machine's network.

Each instance would then be invoked similar to:

```
docker run --name=neo4j-core --detach \
         --network=host \
         --publish=7474:7474 --publish=7687:7687 \
         --publish=5000:5000 --publish=6000:6000 --publish=7000:7000 \
         --env NEO4J_dbms_mode=CORE \
         --env NEO4J_causal__clustering_expected__core__cluster__size=3 \
         --env NEO4J_causal__clustering_initial__discovery__members=core1-public-address:5000,core2-
public-address:5000,core3-public-address:5000 \
         --env NEO4J_causal__clustering_discovery__advertised__address=public-address:5000 \
         --env NEO4J_causal__clustering_transaction__advertised__address=public-address:6000 \
         --env NEO4J_causal__clustering_raft__advertised__address=public-address:7000 \
         --env NEO4J_dbms_connectors_default__advertised__address=public-address \
         --env NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
         --env NEO4J_dbms_connector_bolt_advertised__address=public-address:7687 \
         --env NEO4J_dbms_connector_http_advertised__address=public-address:7474 \
         neo4j:4.0-enterprise
```

Where `public-address` is the public hostname or ip-address of the machine.

See Deploy a cluster for more details of Neo4j Causal Clustering.

3.4. Docker specific operations

This chapter describes various operations that are specific to running Neo4j in a Docker container.

3.4.1. Using */plugins*

To install user-defined procedures, provide a */plugins* volume containing the jars.

```
docker run --publish=7474:7474 --publish=7687:7687 --volume=$HOME/neo4j/plugins:/plugins neo4j:4.0
```

See Java Reference Procedures for more details on procedures.

3.4.2. Using Cypher Shell

The Neo4j Shell can be run locally within a container using a command like this:

```
docker exec --interactive --tty <container> bin/cypher-shell
```

To determine the <*container*>, you can run `docker ps` to list the currently running Docker containers. From the results, use the `CONTAINER ID` in place of <*container*>.

To pass a local file into Cypher Shell, you can run the following, again using the `CONTAINER ID` identified above to replace *<container>*:

```
cat <local-file> | sudo docker exec --interactive <container> bin/cypher-shell -u <username> -p <password>
```

When you run the above, the contents of the *<local-file>* will be passed into the Docker container.

It will then run `bin/cypher-shell`, and authenticate with the provided *<username>* and *<password>*.

For more information on using Cypher Shell, see Cypher Shell.

3.4.3. Upgrading Neo4j on Docker

To enable upgrades, set `NEO4J_dbms_allow__upgrade` to `true`. For more details on upgrading, see:

- Single-instance upgrade
- Upgrade a Neo4j Causal Cluster

3.5. Security

This chapter describes security in Neo4j when running in a Docker container.

3.5.1. Encryption

The Docker image can expose Neo4j's native TLS support. To use your own key and certificate, provide an */ssl* volume with the key and certificate inside. The files must be called *neo4j.key* and *neo4j.cert*. You must also publish port `7473` to access the HTTPS endpoint.

```
docker run --publish=7473:7473 --publish=7687:7687 --volume=$HOME/neo4j/ssl:/ssl neo4j:4.0
```

Chapter 4. Configuration

This chapter describes the configuration of Neo4j components.

The topics described are:

- The *neo4j.conf* file — An introduction to the primary configuration file in Neo4j.
- File locations — An overview of where files are stored in the different Neo4j distributions and the necessary file permissions for running Neo4j.
- Ports — An overview of the ports relevant to a Neo4j installation.
- Set initial password — How to set an initial password.
- Password and user recovery — How to recover after a lost admin password.
- Configure Neo4j connectors — How to configure Neo4j connectors.
- Configure dynamic settings — How to configure certain Neo4j parameters while Neo4j is running.
- Transaction logs — How to configure transaction logs.

For a complete reference of Neo4j configuration settings, see Configuration settings.

4.1. The neo4j.conf file

This section introduces the neo4j.conf file, and its syntax.

This section contains the following:

- Introduction
- Syntax
- JVM-specific configuration settings
- List currently active settings

For a complete reference of Neo4j configuration settings, see Configuration settings.

4.1.1. Introduction

The *neo4j.conf* file is the main source of configuration settings in Neo4j, and includes the mappings of configuration setting keys to values. The location of the *neo4j.conf* file in the different configurations of Neo4j is listed in The locations of important files.

Most of the configuration settings in the *neo4j.conf* file apply directly to Neo4j itself, but there are also some settings which apply to the Java Runtime (the JVM) on which Neo4j runs. For more information, see the JVM specific configuration settings below. Many of the configuration settings are also used by the `neo4j` launcher scripts.

4.1.2. Syntax

- The equals sign (`=`) maps configuration setting keys to configuration values.
- Lines that start with the number sign (`#`) are handled as comments.
- Empty lines are ignored.
- Configuring a setting in *neo4j.conf* will overwrite any default values. In case a setting can define a

list of values, and you wish to amend the default values with custom values, you will have to explicitly list the default values along with the new values.

- There is no order for configuration settings, and each setting in the *neo4j.conf* file must be uniquely specified. If you have multiple configuration settings with the same key, but different values, this can lead to unpredictable behavior.

  The only exception to this is `dbms.jvm.additional`. If you set more than one value for `dbms.jvm.additional`, then each setting value will add another custom JVM argument to the `java` launcher.

4.1.3. JVM-specific configuration settings

- `dbms.memory.heap.initial_size`
- `dbms.memory.heap.max_size`
- `dbms.jvm.additional`

4.1.4. List currently active settings

You can use the procedure `dbms.listConfig()` to list the currently active configuration settings and their values.

Example 2. List currently active configuration settings

```
CALL dbms.listConfig()
YIELD name, value
WHERE name STARTS WITH 'dbms.default'
RETURN name, value
ORDER BY name
LIMIT 3;
```

```
+---------------------------------------------------+
| name                               | value        |
+---------------------------------------------------+
| "dbms.default_advertised_address" | "localhost" |
| "dbms.default_database"           | "neo4j"     |
| "dbms.default_listen_address"     | "localhost" |
+---------------------------------------------------+
```

See also Dynamic settings for information about dynamic settings.

4.2. File locations

This section provides an overview of where files are stored in the different Neo4j distributions, and the necessary file permissions for running Neo4j.

This section describes the following:

- Where to find important files
- Log files
- File permissions

4.2.1. Where to find important files

The table below lists the location of important files, and whether the location is customizable:

Table 9. The locations of important files

Package	Configuration [1]	Data [2]	Logs	Metrics	Import	Bin	Lib	Plugins
Linux or macOS tarball	*<neo4j-home>/conf/neo4j.conf*	*<neo4j-home>/data*	*<neo4j-home>/logs*	*<neo4j-home>/metrics*	*<neo4j-home>/import*	*<neo4j-home>/bin*	*<neo4j-home>/lib*	*<neo4j-home>/plugins*
Windows zip	*<neo4j-home>\conf\neo4j.conf*	*<neo4j-home>\data*	*<neo4j-home>\logs*	*<neo4j-home>\metrics*	*<neo4j-home>\import*	*<neo4j-home>\bin*	*<neo4j-home>\lib*	*<neo4j-home>\plugins*
Debian	*/etc/neo4j/neo4j.conf*	*/var/lib/neo4j/data*	*/var/log/neo4j/* [3]	*/var/lib/neo4j/metrics*	*/var/lib/neo4j/import*	*/usr/bin*	*/usr/share/neo4j/lib*	*/var/lib/neo4j/plugins*
RPM	*/etc/neo4j/neo4j.conf*	*/var/lib/neo4j/data*	*/var/log/neo4j/* [3]	*/var/lib/neo4j/metrics*	*/var/lib/neo4j/import*	*/usr/bin*	*/usr/share/neo4j/lib*	*/var/lib/neo4j/plugins*
Neo4j Desktop [4]	Use the *Open Folder* button in the application to locate the relevant directories.							
Customizable by option	See Configuration of *<neo4j-home>* and *conf*	`dbms.directories.data`	`dbms.directories.logs`	`dbms.directories.metrics`	`dbms.directories.import`	Not applicable	`dbms.directories.lib`	`dbms.directories.plugins`

[1] For details about *neo4j.conf*, see: The neo4j.conf file.

[2] Please note that the data directory is internal to Neo4j and its structure is subject to change between versions without notice.

[3] To view the *neo4j.log* for Debian and RPM, use `journalctl --unit=neo4j`.

[4] Applicable to all operating systems where Neo4j Desktop is supported.

The locations of *<neo4j-home>* and *conf* can be configured using environment variables, as described below:

Table 10. Configuration of <neo4j-home> and conf

Location	Default	Environment variable	Notes
<neo4j-home>	Root directory of the installation	`NEO4J_HOME`	Must be an absolute path.
conf	*<neo4j-home>/conf*	`NEO4J_CONF`	Must be set explicitly if it is not a subdirectory of *<neo4j-home>*.

4.2.2. Log files

Filename	Description
neo4j.log	The standard log, where general information about Neo4j is written. Not written for Debian and RPM packages. See relevant sections.
debug.log	Information useful when debugging problems with Neo4j.
http.log	Request log for the HTTP API.
gc.log	Garbage Collection logging provided by the JVM.

Filename	Description
query.log	Log of executed queries (Enterprise Edition only).
security.log	Log of security events (Enterprise Edition only).
service-error.log	Log of errors encountered when installing or running the Windows service. (Windows only.)

4.2.3. File permissions

The user that Neo4j runs as must have the following permissions:

Read only

- *conf*
- *import*
- *bin*
- *lib*
- *plugins*

Read and write

- *data*
- *logs*
- *metrics*

Execute

- all files in *bin*

4.3. Ports

This section lists ports relevant to a Neo4j installation.

This section provides an overview for determining which Neo4j-specific ports should be opened up in your firewalls. Note that these ports are in addition to those necessary for ordinary network operation. Specific recommendations on port openings cannot be made, as the firewall configuration must be performed taking your particular conditions into consideration.

Name	Default port number	Related settings	Comments
Backups	`6362-6372`	`dbms.backup.enabled` `dbms.backup.listen_address`	Backups are enabled by default. In production environments, external access to the backup port(s) should be blocked by a firewall. See also Backup.
HTTP	`7474`	See Configure connectors.	It is recommended to not open up this port for external access in production environments, since traffic is unencrypted. Used by Neo4j Browser.
HTTPS	`7473`	See Configure connectors.	
Bolt	`7687`	See Configure connectors.	Used by Cypher Shell and by Neo4j Browser.

Name	Default port number	Related settings	Comments
Causal Cluster	`5000`, `6000`, `7000`	`causal_clustering.discovery_listen_address` `causal_clustering.transaction_listen_address` `causal_clustering.raft_listen_address`	The listed ports are the default ports in *neo4j.conf*. The ports are likely be different in a production installation; therefore the potential opening of ports must be modified accordingly. See also Settings reference.
Graphite monitoring	`2003`	`metrics.graphite.server`	This is an outbound connection in order for the Neo4j database to communicate with the Graphite server. See also Metrics.
Prometheus monitoring	`2004`	`metrics.prometheus.enabled` and `metrics.prometheus.endpoint`	See also Metrics.
JMX monitoring	`3637`	`dbms.jvm.additional=-Dcom.sun.management.jmxremote.port=3637`	This setting is for exposing the JMX. This is not the recommended way of inspecting a Neo4j database. It is not enabled by default.

4.4. Set an initial password

This section describes how to set an initial password for Neo4j.

Use the `set-initial-password` command of `neo4j-admin` to define the password for the native user `neo4j`. This must be performed before starting up the database for the first time.

Syntax:

`neo4j-admin set-initial-password <password> [--require-password-change]`

Example 3. Use the `set-initial-password` command of neo4j-admin

Set the password for the native `neo4j` user to 'h6u4%kr' before starting the database for the first time.

```
$neo4j-home> bin/neo4j-admin set-initial-password h6u4%kr
```

Example 4. Use the `set-initial-password` command of neo4j-admin with the optional `--require-password-change` flag

Set the password for the native `neo4j` user to 'secret' before starting the database for the first time. You will be prompted to change this password to one of your own choice at first login.

```
$neo4j-home> bin/neo4j-admin set-initial-password secret --require-password-change
```

If the password is not set explicitly using this method, it will be set to the default password `neo4j`. In that case, you will be prompted to change the default password at first login.

4.5. Password and user recovery

This section describes how to recover from a lost password, specifically for an admin user.

It is recommended to block network connections during the recovery phase, so users should connect to Neo4j only via localhost. This can be achieved by temporarily commenting/setting this parameter in neo4j.conf:

```
#dbms.connectors.default_listen_address=your_configuration
```

or

```
dbms.connectors.default_listen_address=127.0.0.1
```

Use the following steps to set a new password (assuming your admin user is named neo4j):

1. Stop Neo4j

```
$ bin/neo4j stop
```

2. Modify neo4j.conf:

```
dbms.security.auth_enabled=false
```

3. Start Neo4j

```
$ bin/neo4j start
```

4. Connect via Cypher Shell to modify the admin user password:

```
$ bin/cypher-shell -d system

neo4j@system> ALTER USER neo4j SET PASSWORD 'mynewpass';

neo4j@system> :exit
```

5. Stop Neo4j

```
$ bin/neo4j stop
```

6. Modify neo4j.conf by commenting out `dbms.security.auth_enabled` *like this:*

```
# dbms.security.auth_enabled=false
```

or setting `dbms.security.auth_enabled` *to true like this:*

```
dbms.security.auth_enabled=true
```

7. Start Neo4j

```
$ bin/neo4j start
```

4.6. Configure connectors

This section describes how to configure connectors for Neo4j.

This section describes the following:

- Available connectors
- Configuration options
- Options for Bolt thread pooling
- Defaults for addresses

4.6.1. Available connectors

The table below lists the available Neo4j connectors:

Table 11. Default connectors and their ports

Connector name	Protocol	Default port number
`dbms.connector.bolt`	Bolt	`7687`
`dbms.connector.http`	HTTP	`7474`
`dbms.connector.https`	HTTPS	`7473`

When configuring the HTTPS or Bolt connector, see also SSL framework for details on how to work with SSL certificates.

4.6.2. Configuration options

The connectors are configured by settings on the format `dbms.connector.<connector-name>.<setting-suffix>>`. The available suffixes are described in the table below:

Table 12. Configuration option suffixes for connectors

Option name	Default	Setting(s)	Description
`enabled`	`true` [1]	`dbms.connector.bolt.enabled`, `dbms.connector.http.enabled`, `dbms.connector.https.enabled` [2]	This setting allows the client connector to be enabled or disabled. When disabled, Neo4j does not listen for incoming connections on the relevant port. [1] When Neo4j is used in embedded mode, the default value is `false`. [2] The default value for `dbms.connector.https.enabled` is `false`.
`listen_address`	`127.0.0.1:<connector-default-port>`	`dbms.connector.bolt.listen_address`, `dbms.connector.https.listen_address`, `dbms.connector.http.listen_address`	This setting specifies how Neo4j listens for incoming connections. It consists of two parts; an IP address (e.g. 127.0.0.1 or 0.0.0.0) and a port number (e.g. 7687), and is expressed in the format `<ip-address>:<port-number>`. See below for an example of usage.

Option name	Default	Setting(s)	Description
advertised_address	localhost:<connector-default-port>	dbms.connector.bolt.advertised_address, dbms.connector.https.advertised_address, dbms.connector.http.advertised_address	This setting specifies the address that clients should use for this connector. This is useful in a Causal Cluster as it allows each server to correctly advertise addresses of the other servers in the cluster. The advertised address consists of two parts; an address (fully qualified domain name, hostname, or IP address) and a port number (e.g. 7687), and is expressed in the format `<address>:<port-number>`. See below for an example of usage.
tls_level	DISABLED	dbms.connector.bolt.tls_level	This setting is only applicable to the Bolt connector. It allows the connector to accept encrypted and/or unencrypted connections. The default value is `DISABLED`, where only unencrypted client connections are to be accepted by this connector, and all encrypted connections will be rejected. Other values are `REQUIRED` and `OPTIONAL`. Use `REQUIRED` when only encrypted client connections are to be accepted by this connector, and all unencrypted connections will be rejected. Use `OPTIONAL` where either encrypted or unencrypted client connections are accepted by this connector.

Example 5. Specify `listen_address` for the Bolt connector

To listen for Bolt connections on all network interfaces (0.0.0.0) and on port 7000, set the `listen_address` for the Bolt connector:

```
dbms.connector.bolt.listen_address=0.0.0.0:7000
```

Example 6. Specify `advertised_address` for the Bolt connector

If routing traffic via a proxy, or if port mappings are in use, it is possible to specify `advertised_address` for each connector individually. For example, if port 7687 on the Neo4j Server is mapped from port 9000 on the external network, specify the `advertised_address` for the Bolt connector:

```
dbms.connector.bolt.advertised_address=<server-name>:9000
```

4.6.3. Options for Bolt thread pooling

See Bolt thread pool configuration to learn more about Bolt thread pooling and how to configure it on the connector level.

4.6.4. Defaults for addresses

It is possible to specify defaults for the configuration options with `listen_address` and `advertised_address` suffixes, as described below. Setting a default value will apply to all the connectors, unless specifically configured for a certain connector.

`dbms.default_listen_address`

This configuration option defines a default IP address of the settings with the `listen_address` suffix for all connectors. If the IP address part of the `listen_address` is not specified, it is inherited from the shared setting `dbms.default_listen_address`.

Example 7. Specify `listen_address` for the Bolt connector

To listen for Bolt connections on all network interfaces (0.0.0.0) and on port 7000, set the `listen_address` for the Bolt connector:

```
dbms.connector.bolt.listen_address=0.0.0.0:7000
```

This is equivalent to specifying the IP address by using the `dbms.default_listen_address` setting, and then specifying the port number for the Bolt connector.

```
dbms.default_listen_address=0.0.0.0

dbms.connector.bolt.listen_address=:7000
```

`dbms.default_advertised_address`

This configuration option defines a default address of the settings with the `advertised_address` suffix for all connectors. If the address part of the `advertised_address` is not specified, it is inherited from the shared setting `dbms.default_advertised_address`.

Example 8. Specify `advertised_address` for the Bolt connector

Specify the address that clients should use for the Bolt connector:

```
dbms.connector.bolt.advertised_address=server1:9000
```

This is equivalent to specifying the address by using the `dbms.default_advertised_address` setting, and then specifying the port number for the Bolt connector.

```
dbms.default_advertised_address=server1

dbms.connector.bolt.advertised_address=:9000
```

The default address settings can only accept the hostname or IP address portion of the full socket address. Port numbers are protocol-specific, and can only be added by the protocol-specific connector configuration.

For example, if you configure the default address value to be `example.com:9999`, Neo4j will fail to start and you will get an error in *neo4j.log*.

4.7. Dynamic settings

This section describes how to change your Neo4j configuration while Neo4j is running, and which settings can be changed.

This section contains the following:

- Introduction
- Discover dynamic settings
- Update dynamic settings
- Dynamic settings reference

4.7.1. Introduction

Neo4j Enterprise Edition supports changing some configuration settings at runtime, without restarting the service.

Changes to the configuration at runtime are not persisted. To avoid losing changes when restarting Neo4j make sure to update *neo4j.conf* as well.

4.7.2. Discover dynamic settings

Use the procedure `dbms.listConfig()` to discover which configuration values can be dynamically updated, or consult Dynamic settings reference.

Example 9. Discover dynamic settings

```
CALL dbms.listConfig()
YIELD name, dynamic
WHERE dynamic
RETURN name
ORDER BY name
LIMIT 4;
```

```
+----------------------------------------------+
| name                                         |
+----------------------------------------------+
| "dbms.checkpoint.iops.limit"                 |
| "dbms.logs.query.allocation_logging_enabled" |
| "dbms.logs.query.enabled"                    |
| "dbms.logs.query.page_logging_enabled"       |
+----------------------------------------------+
4 rows
```

4.7.3. Update dynamic settings

An administrator is able to change some configuration settings at runtime, without restarting the service.

Syntax:

`CALL dbms.setConfigValue(setting, value)`

Returns:

Nothing on success.

Exceptions:

Unknown or invalid setting name.
The setting is not dynamic and can not be changed at runtime.

Invalid setting value.

The following example shows how to dynamically enable query logging.

Example 10. Set a config value

```
CALL dbms.setConfigValue('dbms.logs.query.enabled', 'info')
```

If an invalid value is passed, the procedure will show a message to that effect.

Example 11. Try to set invalid config value

```
CALL dbms.setConfigValue('dbms.logs.query.enabled', 'yes')
```

```
Failed to invoke procedure `dbms.setConfigValue`: Caused by:
org.neo4j.graphdb.config.InvalidSettingException: Bad value 'yes' for setting
'dbms.logs.query.enabled': 'yes' not one of [OFF, INFO, VERBOSE]
```

To reset a config value to its default, pass an empty string as the *value* argument.

Example 12. Reset a config value to default

```
CALL dbms.setConfigValue('dbms.logs.query.enabled', '')
```

4.7.4. Dynamic settings reference

Dynamic settings reference

- cypher.query_max_allocations: The maximum amount of heap memory allocations to for cypher to perform on a single query, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g').
- dbms.checkpoint.iops.limit: Limit the number of IOs the background checkpoint process will consume per second.
- dbms.logs.debug.level: Debug log level threshold.
- dbms.logs.query.allocation_logging_enabled: Log allocated bytes for the executed queries being logged.
- dbms.logs.query.enabled: Log executed queries.
- dbms.logs.query.page_logging_enabled: Log page hits and page faults for the executed queries being logged.
- dbms.logs.query.parameter_logging_enabled: Log parameters for the executed queries being logged.
- dbms.logs.query.rotation.keep_number: Maximum number of history files for the query log.
- dbms.logs.query.rotation.size: The file size in bytes at which the query log will auto-rotate.
- dbms.logs.query.runtime_logging_enabled: Logs which runtime that was used to run the query.
- dbms.logs.query.threshold: If the execution of query takes more time than this threshold, the query is logged once completed - provided query logging is set to INFO.

- dbms.logs.query.time_logging_enabled: Log detailed time information for the executed queries being logged.
- dbms.track_query_allocation: Enables or disables tracking of how many bytes are allocated by the execution of a query.
- dbms.track_query_cpu_time: Enables or disables tracking of how much time a query spends actively executing on the CPU.
- dbms.transaction.concurrent.maximum: The maximum number of concurrently running transactions.
- dbms.transaction.sampling.percentage: Transaction sampling percentage.
- dbms.transaction.timeout: The maximum time interval of a transaction within which it should be completed.
- dbms.transaction.tracing.level: Transaction creation tracing level.
- dbms.tx_log.preallocate: Specify if Neo4j should try to preallocate logical log file in advance.
- dbms.tx_log.rotation.retention_policy: Make Neo4j keep the logical transaction logs for being able to backup the database.
- dbms.tx_log.rotation.size: Specifies at which file size the logical log will auto-rotate.

4.8. Transaction logs

This section explains the retention and rotation policies for the Neo4j transaction logs, and how to configure them.

The transaction logs record all write operations in the database. This includes additions or modifications to data, as well as the addition or modification of any indexes or constraints. The transaction logs are the "source of truth" in scenarios where the database needs to be recovered. They are used to provide for incremental backups, as well as for cluster operations. For any given configuration, at least the latest non-empty transaction log will be kept.

Log location

By default, transaction logs for a database are located at *<neo4j-home>/data/transactions/<database-name>*. Each database keeps its own directory with transaction logs. The root directory where those folders are located is configured by `dbms.directories.transaction.logs.root`. For maximum performance, it is recommended to configure transaction logs to be stored on a dedicated device.

Log rotation

Log rotation is configured using the parameter `dbms.tx_log.rotation.size`. By default, log switches happen when log sizes surpass 250 MB.

Log retention

There are several different means of controlling the amount of transaction logs that are kept, using the parameter `dbms.tx_log.rotation.retention_policy`. This parameter can be configured in two different ways:

- `dbms.tx_log.rotation.retention_policy=<true/false>`

  If this parameter is set to `true`, transaction logs will be kept indefinitely. This option is not recommended due to the effectively unbounded storage usage. Old transaction logs cannot be safely archived or removed by external jobs, since safe log pruning requires knowledge about the most recent successful checkpoint.

  If this parameter is set to `false`, only the most recent non-empty log will be kept. This option is not recommended in production Enterprise Edition environments, as incremental backups rely on the presence of the transaction logs since the last backup.

- `dbms.tx_log.rotation.retention_policy=<amount> <type>`

Log pruning

Transaction log pruning refers to the safe and automatic removal of old, unnecessary transaction log files. The transaction log can be pruned when one or more files fall outside of the configured retention policy. Two things are necessary for a file to be removed:

- The file must have been rotated.
- At least one checkpoint must have happened in a more recent log file.

Observing that you have more transaction log files than you expected is likely due to checkpoints either not happening frequently enough, or taking too long. This is a temporary condition and the gap between expected and observed number of log files will be closed on the next successful checkpoint. The interval between checkpoints can be configured using `dbms.checkpoint.interval.time` and `dbms.checkpoint.interval.tx`. If your goal is to have the least amount of transaction log data, it can also help to speed up the checkpoint process itself. The configuration parameter `dbms.checkpoint.iops.limit` controls the number of IOs per second the checkpoint process is allowed to use. Setting the value of this parameter to `-1` allows unlimited IOPS, which can speed up checkpointing. Note that disabling the IOPS limit can cause transaction processing to slow down a bit.

Table 13. Types that can be used to control log retention

Type	Description	Example
files	Number of most recent logical log files to keep	"10 files"
size	Max disk size to allow log files to occupy	"300M size" or "1G size"
txs	Number of transactions to keep	"250k txs" or "5M txs"
hours	Keep logs which contains any transaction committed within N hours from current time	"10 hours"
days	Keep logs which contains any transaction committed within N days from current time	"50 days"

Example 13. Configure log retention policy

This example shows some different ways to configure the log retention policy.

- Keep transaction logs indefinitely:

```
dbms.tx_log.rotation.retention_policy=true
```

- Keep only the most recent non-empty log:

```
dbms.tx_log.rotation.retention_policy=false
```

- Keep logical logs which contain any transaction committed within 30 days:

```
dbms.tx_log.rotation.retention_policy=30 days
```

- Keep logical logs which contain any of the most recent 500 000 transactions:

```
dbms.tx_log.rotation.retention_policy=500k txs
```

Chapter 5. Manage databases

This chapter describes how to create and manage multiple active databases.

This chapter describes the following:

- Introduction
- Administration and configuration
- Queries
- Databases in a Causal Cluster

5.1. Introduction

This section gives an introduction to managing multiple active databases with Neo4j.

This section describes the following:

- Concepts
- The `system` database
- The default database

5.1.1. Concepts

With Neo4j 4.0 you can create and use more than one active database at the same time.

DBMS
: Neo4j is a Database Management System, or *DBMS*, capable of managing multiple databases. The DBMS can manage a standalone server, or a group of servers in a Causal Cluster.

Instance
: A Neo4j instance is a Java process that is running the Neo4j server code.

Transaction domain
: A transaction domain is a collection of graphs that can be updated within the context of a single transaction.

Execution context
: An execution context is a runtime environment for the execution of a request. In practical terms, a request may be a query, a transaction, or an internal function or procedure.

Database
: A database is an administrative partition of a DBMS. In practical terms, it is a physical structure of files organized within a directory or folder, that has the same name of the database. In logical terms, a database is a container for one or more graphs.

  A database defines a *transaction domain* and an *execution context*. This means that a transaction cannot span across multiple databases. Similarly, a procedure is called within a database, although its logic may access data that is stored in other databases.

  A default installation of Neo4j 4.0 contains two databases:

* `system` - the system database, containing metadata on the DBMS and security configuration.
* `neo4j` - the default database, a single database for user data. This has a default name of `neo4j`. A different name can be configured before starting Neo4j for the first time.

The following image illustrates a default installation, including the `system` database and a single database named `neo4j` for user data:

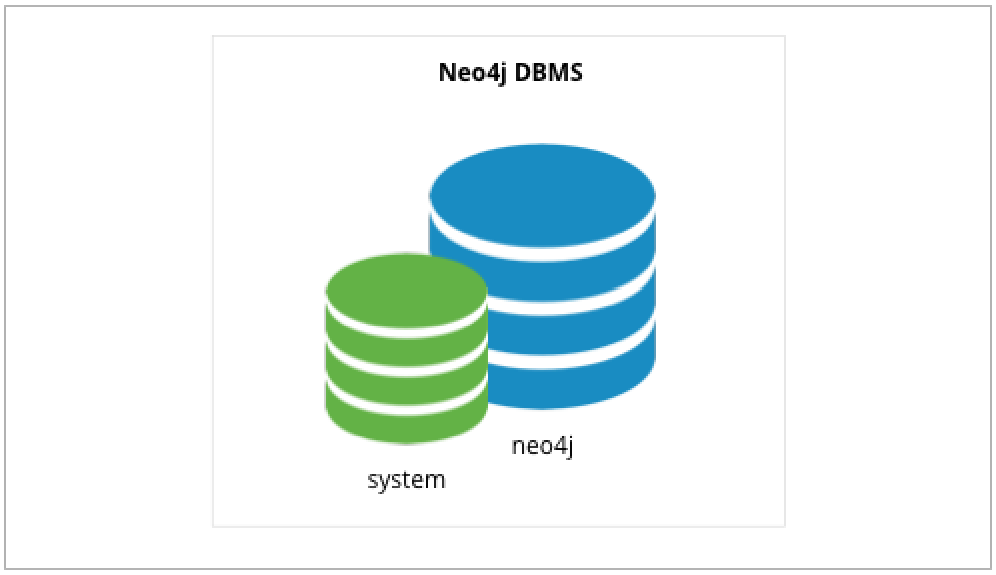

Figure 1. A default Neo4j installation.

Editions

The edition of Neo4j determines the number of possible databases:

* Installations of Community Edition can have exactly **one** user database.
* Installations of Enterprise Edition can have any number of user databases.

All installations include the `system` database.

5.1.2. The `system` database

All installations include a built-in database named `system`, which contains meta-data and security configuration.

The `system` database behaves differently than all other databases. In particular, when connected to this database you can only perform a specific set of administrative functions, as described in detail in Cypher Manual Administration.

Most of the available administrative commands are restricted to users with specific administrative privileges. An example of configuring security privileges is described in Fine-grained access control. Security administration is described in detail in Cypher Manual Security of administration .

The following image illustrates an installation of Neo4j with multiple active databases, named `marketing`, `sales`, and `hr`:

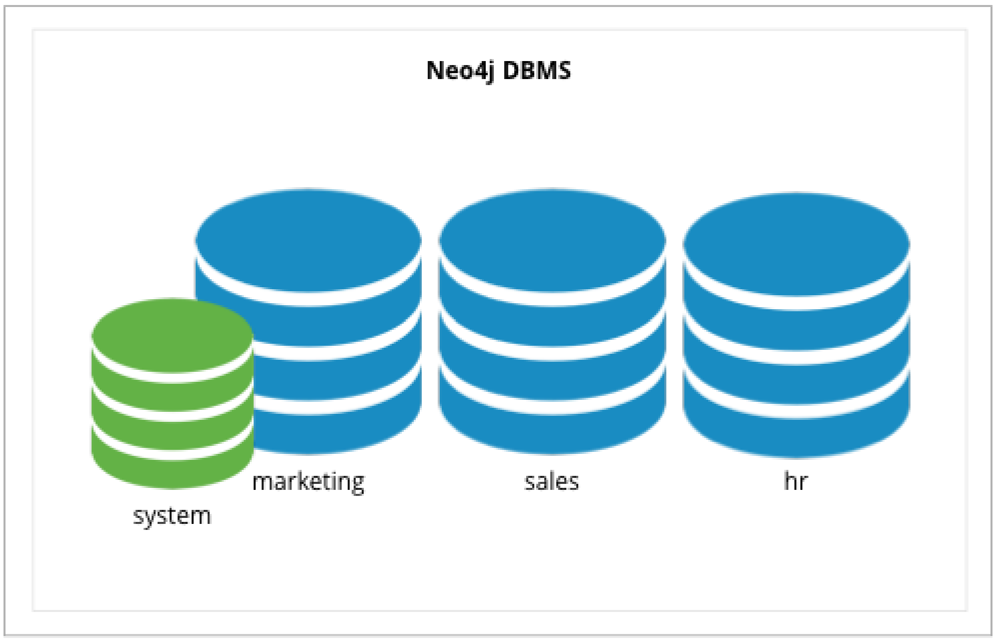

Figure 2. A multiple database Neo4j installation.

5.1.3. The default database

Each Neo4j instance has a default database. If a user connects to Neo4j without specifying a database, it will connect to the default database.

The default database is configurable. See configuration parameters for details.

The following image illustrates an installation of Neo4j containing the three databases for user data, named `marketing`, `sales` and `hr`, and the `system` database. The default database is `sales`:

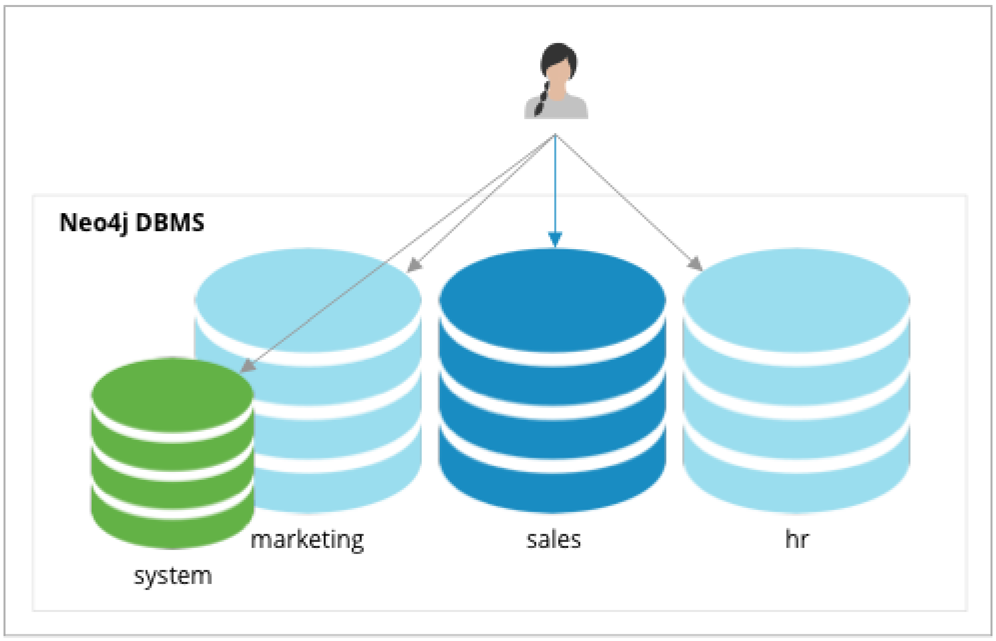

Figure 3. A multiple database Neo4j installation, with a default database.

5.2. Administration and configuration

This section describes how to manage multiple active databases.

This section describes the following:

- Administrative commands
- Configuration parameters

5.2.1. Administrative commands

The following Cypher commands are used on the `system` database to manage multiple databases:

Command	Description
`CREATE DATABASE name`	Create and start a new database.
`DROP DATABASE name`	Drop (remove) an existing database.
`START DATABASE name`	Start a database that has been stopped.
`STOP DATABASE name`	Shut down a database.
`SHOW DATABASE name`	Show the status of a specific database.
`SHOW DATABASES`	Show the name and status of all the databases.
`SHOW DEFAULT DATABASE`	Show the name and status of the default database.

Naming rules for databases are as follows:

- Length must be between 3 and 63 characters.
- The first character of a name must be an ASCII alphabetic character.
- Subsequent characters must be ASCII alphabetic or numeric characters, dots or dashes; `[a..z][0..9].-`
- Names are case-insensitive, and normalized to lowercase.
- Names that begin with an underscore, and with the prefix `system` are reserved for internal use.
- All of the above commands are executed as Cypher commands, and the database name is subject to the standard Cypher restrictions on valid identifiers. In particular, the `-` (dash) character is not legal in Cypher variables, and therefore names with dashes need to be escaped by enclosing with back-ticks. For example, ``CREATE DATABASE `main-db` ``.

For detailed information on Cypher administrative commands, see Cypher Manual Administration.

For examples of using the Cypher administrative commands to manage multiple active databases, see Queries.

5.2.2. Configuration parameters

Configuration parameters are defined in the neo4j.conf file.

The following configuration parameters are applicable for managing databases:

Parameter name	Description	Default value
`dbms.default_database`	Name of the default database for the Neo4j instance. The database is created if it does not exist when the instance starts.	`neo4j`
`dbms.max_databases`	Maximum number of databases that can be used in a Neo4j single instance or Causal Cluster. The number includes all the online and offline databases. The value is an integer with a minimum value of 2. Note that once the limit has been reached, it is not possible to create any additional databases. Similarly, if the limit is changed to a number lower than the total number of existing databases, no additional databases can be created.	`100`

5.3. Queries

This section provides examples of queries and Cypher commands that can be used to create and manage multiple active databases.

This section describes the following:

- Show the status of a specific database
- Show the status of all databases
- Show the status of the default database
- Create a database
- Stop a database
- Start a database
- Drop, or remove a database

For detailed information on Cypher administrative commands, see Cypher Manual → Administration.

5.3.1. Show the status of a specific database

Example 14. `SHOW DATABASE`

```
neo4j@system> SHOW DATABASE neo4j;
```

```
+------------------------------+
| name    | status   | default |
+------------------------------+
| "neo4j" | "online" | TRUE    |
+------------------------------+
```

5.3.2. Show the status of all databases

Example 15. `SHOW DATABASES`

```
neo4j@system> SHOW DATABASES;
```

```
+-------------------------------+
| name     | status   | default |
+-------------------------------+
| "neo4j"  | "online" | TRUE    |
| "system" | "online" | FALSE   |
+-------------------------------+
```

Switching between `online` and `offline` states is achieved using the `START DATABASE` and `STOP DATABASE` commands.

5.3.3. Show the status of the default database

The config setting `dbms.default_database` defines which database is created and started by default when Neo4j starts. The default value of this setting is `neo4j`.

Example 16. `SHOW DEFAULT DATABASE`

```
neo4j@system> SHOW DEFAULT DATABASE;
```

```
+--------------------+
| name    | status   |
+--------------------+
| "neo4j" | "online" |
+--------------------+
```

You can change the default database by using `dbms.default_database`, and restarting the server.

In Community Edition, the default database is the only database available, other than the `system` database.

5.3.4. Create a database

Example 17. `CREATE DATABASE`

```
neo4j@system> CREATE DATABASE sales;
```

```
+---------------------+
| name    | status    |
+---------------------+
| "sales" | "online" |
+---------------------+

1 row available after 58 ms, consumed after another 0 ms
Added 1 nodes, Set 4 properties, Added 1 labels
```

```
neo4j@system> SHOW DATABASES;
```

```
+---------------------------------+
| name     | status   | default |
+---------------------------------+
| "neo4j"  | "online" | TRUE    |
| "system" | "online" | FALSE   |
| "sales"  | "online" | FALSE   |
+---------------------------------+

3 rows available after 6 ms, consumed after another 0 ms
```

5.3.5. Stop a database

Example 18. `STOP DATABASE`

```
neo4j@system> STOP DATABASE sales;
```

```
0 rows available after 18 ms, consumed after another 6 ms
```

```
neo4j@system> SHOW DATABASES;
```

```
+----------------------------------+
| name     | status    | default |
+----------------------------------+
| "neo4j"  | "online"  | TRUE    |
| "system" | "online"  | FALSE   |
| "sales"  | "offline" | FALSE   |
+----------------------------------+

3 rows available after 5 ms, consumed after another 1 ms
```

```
neo4j@system> :use sales
```

```
The database is not currently available to serve your request, refer to the database logs for more
details. Retrying your request at a later time may succeed.
```

```
neo4j@sales[UNAVAILABLE]>
```

5.3.6. Start a database

Example 19. `START DATABASE`

```
neo4j@sales[UNAVAILABLE]> :use system
neo4j@system>
neo4j@system> START DATABASE sales;
```

```
0 rows available after 21 ms, consumed after another 1 ms
```

```
SHOW DATABASES;
```

```
+--------------------------------+
| name     | status   | default |
+--------------------------------+
| "neo4j"  | "online" | TRUE    |
| "system" | "online" | FALSE   |
| "sales"  | "online" | FALSE   |
+--------------------------------+

3 rows available after 5 ms, consumed after another 0 ms
```

```
neo4j@system> :use sales
neo4j@sales>
```

5.3.7. Drop, or remove a database

Example 20. `DROP DATABASE`

```
neo4j@sales> :use system
neo4j@system> DROP DATABASE sales;
```

```
0 rows available after 82 ms, consumed after another 1 ms
```

```
neo4j@system> SHOW DATABASES;
```

```
+--------------------------------+
| name     | status   | default |
+--------------------------------+
| "neo4j"  | "online" | TRUE    |
| "system" | "online" | FALSE   |
+--------------------------------+

2 rows available after 5 ms, consumed after another 1 ms
```

5.4. Databases in a Causal Cluster

This section describes how to manage multiple active databases in a Causal Cluster.

Multiple databases in a Causal Cluster are managed the same way as a single instance. Administrators can use the same Cypher commands described in Administrative commands to manage databases.

This is based on two main principles:

- All databases are available on all members of a cluster - this applies to Core servers and Read Replicas.
- Administrative commands must be executed on the `system` database, on the Leader member of the cluster.

5.4.1. Running Cypher administrative commands from Cypher Shell on a Causal Cluster.

For the following examples, consider a Causal Cluster environment formed by 5 members, 3 Core servers, and 2 Read Replicas:

Example 21. View the members of a Causal Cluster

```
neo4j@neo4j> CALL dbms.cluster.overview();
```

```
+----------------------------------------------------------------------------------------------------
--------------------------------------------------------+
| id        | addresses                                                                         |
databases                    | groups |
+----------------------------------------------------------------------------------------------------
--------------------------------------------------------+
| "8c...3d" | ["bolt://localhost:7683", "http://localhost:7473", "https://localhost:7483"] | {neo4j:
"FOLLOWER", system: "FOLLOWER"}         | []     |
| "8f...28" | ["bolt://localhost:7681", "http://localhost:7471", "https://localhost:7481"] | {neo4j:
"LEADER", system: "LEADER"}                   | []     |
| "e0...4d" | ["bolt://localhost:7684", "http://localhost:7474", "https://localhost:7484"] | {neo4j:
"READ_REPLICA", system: "READ_REPLICA"}     | []     |
| "1a...64" | ["bolt://localhost:7682", "http://localhost:7472", "https://localhost:7482"] | {neo4j:
"FOLLOWER", system: "FOLLOWER"}         | []     |
| "59...87" | ["bolt://localhost:7685", "http://localhost:7475", "https://localhost:7485"] | {neo4j:
"READ_REPLICA", system: "READ_REPLICA"}     | []     |
+----------------------------------------------------------------------------------------------------
--------------------------------------------------------+

5 rows available after 5 ms, consumed after another 0 ms
```

The leader is currently the instance exposing port `7681` for the `bolt` protocol, and `7471/7481` for the `http/https` protocol.

Administrators can connect and execute Cypher commands in the following ways:

Example 22. Using the `bolt://` scheme to connect to the Leader:

```
$ bin/cypher-shell -a bolt://localhost:7681 -d system -u neo4j -p neo4j1
```

```
Connected to Neo4j 4.0.0 at bolt://localhost:7681 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@system> SHOW DATABASES;
```

```
+-------------------------------+
| name     | status   | default |
+-------------------------------+
| "neo4j"  | "online" | TRUE    |
| "system" | "online" | FALSE   |
+-------------------------------+

2 rows available after 34 ms, consumed after another 0 ms
```

```
neo4j@system> CREATE DATABASE data001;
```

```
0 rows available after 378 ms, consumed after another 12 ms
Added 1 nodes, Set 4 properties, Added 1 labels
neo4j@system> SHOW DATABASES;
+--------------------------------+
| name      | status   | default |
+--------------------------------+
| "neo4j"   | "online" | TRUE    |
| "system"  | "online" | FALSE   |
| "data001" | "online" | FALSE   |
+--------------------------------+

3 rows available after 2 ms, consumed after another 1 ms
```

Example 23. Using the `neo4j://` scheme to connect to any Core member:

```
$ bin/cypher-shell -a neo4j://localhost:7683 -d system -u neo4j -p neo4j1
```

```
Connected to Neo4j 4.0.0 at neo4j://localhost:7683 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@system> SHOW DATABASES;
```

```
+--------------------------------+
| name      | status   | default |
+--------------------------------+
| "neo4j"   | "online" | TRUE    |
| "system"  | "online" | FALSE   |
| "data001" | "online" | FALSE   |
+--------------------------------+

3 rows available after 0 ms, consumed after another 0 ms
```

```
neo4j@system> CREATE DATABASE data002;
```

```
0 rows available after 8 ms, consumed after another 1 ms
Added 1 nodes, Set 4 properties, Added 1 labels
```

```
neo4j@system> SHOW DATABASES;
```

```
+--------------------------------+
| name      | status   | default |
+--------------------------------+
| "neo4j"   | "online" | TRUE    |
| "system"  | "online" | FALSE   |
| "data001" | "online" | FALSE   |
| "data002" | "online" | FALSE   |
+--------------------------------+

4 rows available after 33 ms, consumed after another 0 ms
```

The `neo4j://` scheme is the equivalent to the `bolt+routing:` scheme available in earlier versions of Neo4j, but it can be used seamlessly with a standalone and clustered DBMS.

Chapter 6. Clustering

This chapter describes the configuration and operation of a Neo4j Causal Cluster.

This chapter describes the following:

- Introduction — An overview of the Causal Clustering architecture.
- Deploy a cluster — The basics of configuring and deploying a new cluster.
- Seed a cluster — How to deploy a Causal Cluster with pre-existing data.
- Discovery — How members of a cluster discover each other.
- Intra-cluster encryption — How to secure the cluster communication.
- Internals — A few internals regarding the operation of the cluster.
- Settings reference — A summary of the most important Causal Cluster settings.

Further information:

- For instructions on setting up Causal Clustering when running Neo4j in a Docker container, see Causal Clustering on Docker.
- For an example of managing multiple databases in a Causal Cluster, see Multiple databases in a Causal Cluster.
- For instructions on how you can upgrade your Neo4j Causal Cluster, see Upgrading a Causal Cluster.
- For a summary of the facilities that are available for monitoring a Neo4j Causal Cluster, see Monitoring (and specifically, Monitoring a Causal Cluster).
- For a tutorial on setting up a test cluster locally on a single machine, see Set up a local Causal Cluster.
- For advanced concepts, including the implementation of the Raft Protocol, see Advanced Causal Clustering

6.1. Introduction

This section gives an introduction to the Neo4j Causal Clustering architecture.

This section includes:

- Overview
- Operational view
    - Core Servers
    - Read Replicas
- Causal consistency
- Summary

6.1.1. Overview

Neo4j's Causal Clustering provides three main features:

1. **Safety:** Core Servers provide a fault tolerant platform for transaction processing which will remain

available while a simple majority of those Core Servers are functioning.

2. **Scale:** Read Replicas provide a massively scalable platform for graph queries that enables very large graph workloads to be executed in a widely distributed topology.
3. **Causal consistency:** when invoked, a client application is guaranteed to read at least its own writes.

Together, this allows the end-user system to be fully functional and both read and write to the database in the event of multiple hardware and network failures and makes reasoning about database interactions straightforward.

In the remainder of this section we will provide an overview of how causal clustering works in production, including both operational and application aspects.

6.1.2. Operational view

From an operational point of view, it is useful to view the cluster as being composed of servers with two different roles: Cores and Read Replicas.

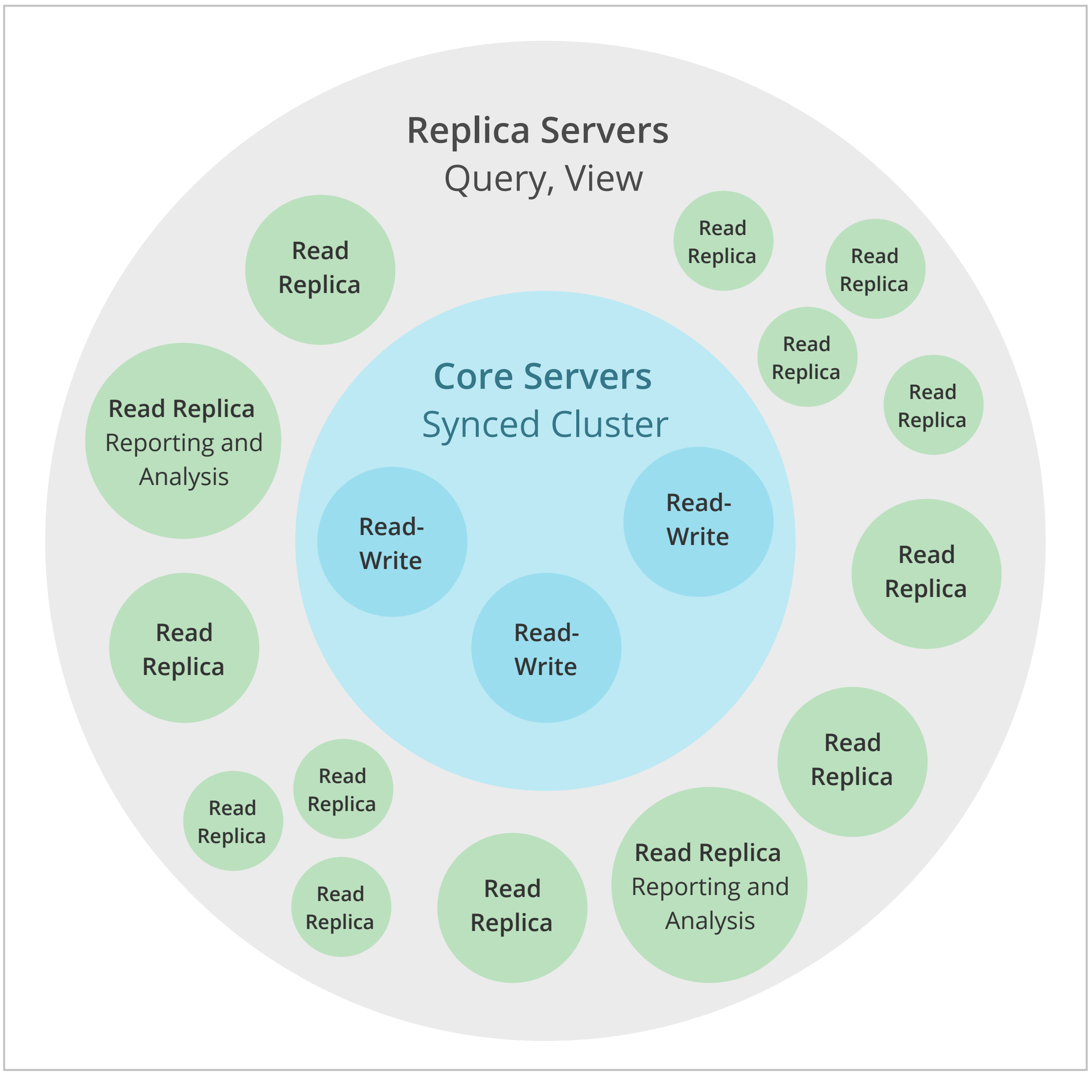

Figure 4. Causal Cluster Architecture

The two roles are foundational in any production deployment but are managed at different scales from one another and undertake different roles in managing the fault tolerance and scalability of the overall cluster.

Core Servers

The main responsibility of Core Servers is to safeguard data. Core Servers do so by replicating all transactions using the Raft protocol. Raft ensures that the data is safely durable before confirming transaction commit to the end user application. In practice this means once a majority of Core Servers in a cluster (`N/2+1`) have accepted the transaction, it is safe to acknowledge the commit to the end user application.

The safety requirement has an impact on write latency. Implicitly writes will be acknowledged by the fastest majority, but as the number of Core Servers in the cluster grows so do the size of the majority needed to acknowledge a write.

In practice this means that there are relatively few machines in a typical Core Server cluster, enough to provide sufficient fault tolerance for the specific deployment. This is calculated with the formula `M = 2F + 1` where `M` is the number of Core Servers required to tolerate `F` faults. For example:

- In order to tolerate two failed Core Servers we would need to deploy a cluster of five Cores.
- The smallest *fault tolerant* cluster, a cluster that can tolerate one fault, must have three Cores.
- It is also possible to create a Causal Cluster consisting of only two Cores. However, that cluster will not be fault-tolerant. If one of the two servers fails, the remaining server will become read-only.

Should the cluster suffer enough Core failures then it can no longer process writes and it will become read-only to preserve safety.

Read Replicas

The main responsibility of Read Replicas is to scale out graph workloads. Read Replicas act like caches for the graph data that the Core Servers safeguard and are fully capable of executing arbitrary (read-only) queries and procedures.

Read Replicas are asynchronously replicated from Core Servers via transaction log shipping. They will periodically poll an upstream server for new transactions and have these shipped over. Many Read Replicas can be fed data from a relatively small number of Core Servers, allowing for a large fan out of the query workload for scale.

Read Replicas should typically be run in relatively large numbers and treated as disposable. Losing a Read Replica does not impact the cluster's availability, aside from the loss of its fraction of graph query throughput. It does not affect the fault tolerance capabilities of the cluster.

6.1.3. Causal consistency

While the operational mechanics of the cluster are interesting from an application point of view, it is also helpful to think about how applications will use the database to get their work done. In an application we typically want to read from the graph and write to the graph. Depending on the nature of the workload we usually want reads from the graph to take into account previous writes to ensure causal consistency.

Causal consistency is one of numerous consistency models used in distributed computing. It ensures that causally related operations are seen by every instance in the system in the same order. Consequently, client applications are guaranteed to read their own writes, regardless of which instance they communicate with. This simplifies interaction with large clusters, allowing clients to treat them as a single (logical) server.

Causal consistency makes it possible to write to Core Servers (where data is safe) and read those writes from a Read Replica (where graph operations are scaled out). For example, causal consistency guarantees that the write which created a user account will be present when that same user subsequently attempts to log in.

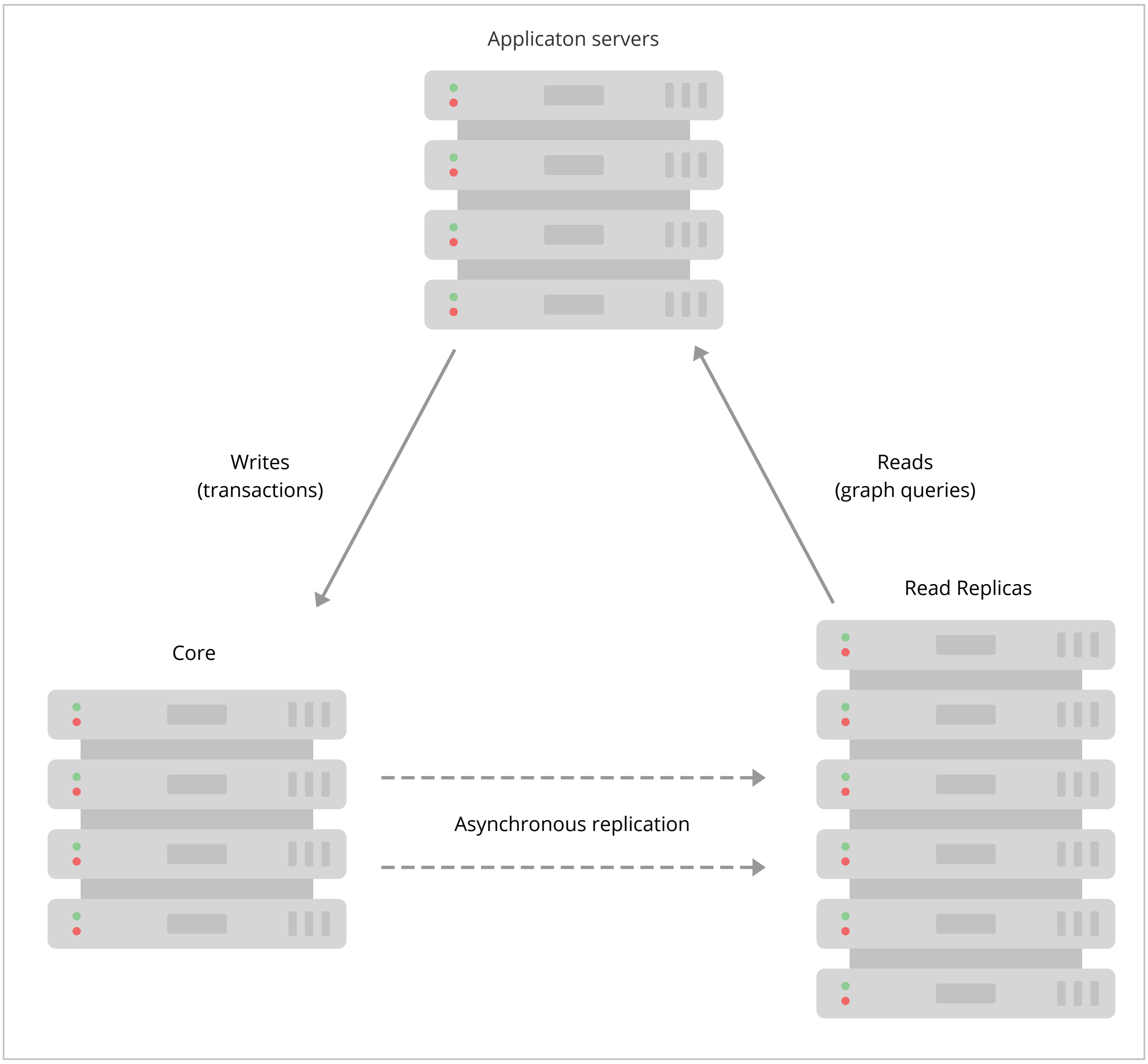

Figure 5. Causal Cluster setup with causal consistency via Neo4j drivers

On executing a transaction, the client can ask for a bookmark which it then presents as a parameter to subsequent transactions. Using that bookmark the cluster can ensure that only servers which have processed the client's bookmarked transaction will run its next transaction. This provides a *causal chain* which ensures correct read-after-write semantics from the client's point of view.

Aside from the bookmark everything else is handled by the cluster. The database drivers work with the cluster topology manager to choose the most appropriate Core Servers and Read Replicas to provide high quality of service.

6.1.4. Summary

In this section we have examined Causal Clustering at a high level from an operational and an application development point of view. We now understand that the Core Servers are responsible for the long-term safekeeping of data while the more numerous Read Replicas are responsible for scaling out graph query workloads. Reasoning about this powerful architecture is greatly simplified by the Neo4j drivers which abstract the cluster topology to easily provide read levels like causal consistency.

6.2. Deploy a cluster

This section describes how to deploy a new Neo4j Causal Cluster.

This section includes:

- Introduction
- Configure a Core-only cluster
- Add a Core Server to an existing cluster
- Add a Read Replica to an existing cluster

6.2.1. Introduction

In this section we describe how to set up a new Causal Cluster consisting of three Core instances. We then proceed to show how more Core Servers as well as Read Replicas can be added to a running cluster.

Three Cores is the minimum number of servers needed in order to form a fault-tolerant Causal Cluster. See Core Servers for a discussion on the number of servers required in various scenarios.

Refer to Set up a local Causal Cluster for a tutorial on how to set up a Causal Cluster on a local machine.

6.2.2. Configure a Core-only cluster

The following configuration settings are important to consider when deploying a new Causal Cluster. See also Settings reference for more detailed descriptions and examples.

Table 14. Important settings for a new Causal Cluster

Option name	Description
`dbms.default_listen_address`	The address or network interface this machine uses to listen for incoming messages. Setting this value to `0.0.0.0` makes Neo4j bind to all available network interfaces.
`dbms.default_advertised_address`	The address that other machines are told to connect to. In the typical case, this should be set to the fully qualified domain name or the IP address of this server.
`dbms.mode`	The operating mode of a single server instance. For Causal Clustering, there are two possible modes: `CORE` or `READ_REPLICA`.
`causal_clustering.minimum_core_cluster_size_at_formation`	The minimum number of Core machines in the cluster at formation. A cluster will not form without the number of Cores defined by this setting, and this should in general be configured to the full and fixed amount.
`causal_clustering.minimum_core_cluster_size_at_runtime`	The minimum number of Core instances which will exist in the consensus group.
`causal_clustering.initial_discovery_members`	The network addresses of an initial set of Core cluster members that are available to bootstrap this Core or Read Replica instance. In the default case, the initial discovery members are given as a comma-separated list of address/port pairs, and the default port for the discovery service is `:5000`. It is good practice to set this parameter to the same value on all Core Servers. The behavior of this setting can be modified by configuring the setting `causal_clustering.discovery_type`. This is described in detail in Discovery.

Listen configuration

Listening on 0.0.0.0 makes the ports publicly available. Make sure you understand the security implications and strongly consider setting up encryption.

The following example shows how to set up a simple cluster with three Core servers:

Example 24. Configure a Core-only cluster

In this example, we will configure three Core Servers named `core01.example.com`, `core02.example.com` and `core03.example.com`. We have already installed Neo4j Enterprise Edition on all three servers. We configure them by preparing *neo4j.conf* on each server. Note that they are all identical, except for the configuration of `dbms.default_advertised_address`:

neo4j.conf on core01.example.com:

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core01.example.com
dbms.mode=CORE
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.ex
ample.com:5000
```

neo4j.conf on core02.example.com:

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core02.example.com
dbms.mode=CORE
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.ex
ample.com:5000
```

neo4j.conf on core03.example.com:

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core03.example.com
dbms.mode=CORE
causal_clustering.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.ex
ample.com:5000
```

Now we are ready to start the Neo4j servers. The startup order does not matter.

After the cluster has started, we can connect to any of the instances and run `CALL dbms.cluster.overview()` to check the status of the cluster. This will show information about each member of the cluster.

We now have a Neo4j Causal Cluster of three instances running.

Startup time

The instance may appear unavailable while it is joining the cluster. If you want to follow along with the startup, you can follow the messages in *neo4j.log*.

6.2.3. Add a Core Server to an existing cluster

Core Servers are added to an existing cluster by starting a new Neo4j instance with the appropriate configuration. The new server will join the existing cluster and become available once it has copied the data from its peers. It may take some time for the new instance to perform the copy if the existing cluster contains large amounts of data.

The setting `causal_clustering.initial_discovery_members` shall be updated on all the servers in the cluster to include the new server.

Example 25. Add a Core Server to an existing cluster

In this example, we will add a Core Server, `core04.example.com`, to the cluster that we created in Configure a Core-only cluster.

We configure the following entries in *neo4j.conf*:

neo4j.conf on core04.example.com:

```
dbms.default_listen_address=0.0.0.0
dbms.default_advertised_address=core04.example.com
dbms.mode=CORE
causal_clustering.minimum_core_cluster_size_at_formation=3
causal_clustering.minimum_core_cluster_size_at_runtime=3
causal_clustering.discovery_members=core01.example.com:5000,core02.example.com:5000,core03.example.co
m:5000,core04.example.com:5000
```

Note that the configuration is very similar to that of the previous servers. In this example, the new server is not intended to be a permanent member of the cluster, thus it is not included in `causal_clustering.discovery_members`.

Now we can start the new Core Server and let it add itself to the existing cluster.

6.2.4. Add a Read Replica to an existing cluster

Initial Read Replica configuration is provided similarly to Core Servers via *neo4j.conf*. Since Read Replicas do not participate in cluster quorum decisions, their configuration is shorter; they only need to know the addresses of some of the Core Servers which they can bind to in order to discover the cluster. They can then choose an appropriate Core Server from which to copy data.

Example 26. Add a Read Replica to an existing cluster

In this example, we will add a Read Replica, `replica01.example.com`, to the cluster that we created in Configure a Core-only cluster.

We configure the following entries in *neo4j.conf*:

neo4j.conf on replica01.example.com:

```
dbms.mode=READ_REPLICA
causal_clustering.discovery_members=core01.example.com:5000,core02.example.com:5000,core03.example.co
m:5000
```

Now we can start the new Read Replica and let it add itself to the existing cluster.

6.3. Seed a cluster

This section describes how to seed a new Neo4j Causal Cluster with existing data.

This section includes:

- Introduction
- Seed from backups
- Seed using the import tool

6.3.1. Introduction

In Deploy a cluster we learned how to create a cluster with empty databases. However, regardless of whether we are just playing around with Neo4j or setting up a production environment, it is likely that we have some existing data that we wish to transfer into our cluster.

This section outlines how to create a Causal Cluster containing data either seeded from an existing online or offline Neo4j database, or imported from some other data source using the import tool. The general steps to seed a cluster will follow the same pattern, regardless of which format our data is in:

1. Create a new Neo4j Core-only cluster.
2. Seed the cluster.
3. Start the cluster.

> The databases which you are using to seed the cluster must be of the same version of Neo4j as the cluster itself.

6.3.2. Seed from backups

For this section, it is assumed that we already have healthy backups of an existing Neo4j deployment. This could be online or offline backups from a standalone Neo4j instance or a Neo4j Causal Cluster. For details on performing online backups, please refer to Backup.

> Moving files and directories manually in or out of a Neo4j installation is not recommended and considered unsupported usage. If you have an existing Neo4j database which you wish to use for a new cluster, then use `neo4j-admin dump` to create an offline backup.

The process described here can also be used to seed a new Causal Cluster from an existing Read Replica. This can be useful, for example, in disaster recovery where some servers have retained their data during a catastrophic event.

1. Create a new Neo4j Core-only cluster.

   Follow the instructions in Configure a Core-only cluster to create a new Neo4j Core-only cluster.

   > You could start the cluster now in order to test that everything is correctly configured, but this will create default databases as part of cluster formation. Since you are trying to seed with a set of databases you have to subsequently stop every instance, unbind them from the cluster using `neo4j-admin unbind` and remove those databases so that the correct seeds can be used instead.

2. Seed the cluster.

   Use `neo4j-admin restore` or `neo4j-admin load` to seed all the Core instances in the cluster.

   The examples assume that we are restoring one user database with the default name of `neo4j` in addition to the `system` database which contains replicated configuration state. Modify the command line arguments to match your exact setup.

Example 27. Seed using neo4j-admin restore.

```
neo4j-01$ ./bin/neo4j-admin restore --from=/path/to/system-backup-dir --database=system
neo4j-01$ ./bin/neo4j-admin restore --from=/path/to/neo4j-backup-dir --database=neo4j
```

```
neo4j-02$ ./bin/neo4j-admin restore --from=/path/to/system-backup-dir --database=system
neo4j-02$ ./bin/neo4j-admin restore --from=/path/to/neo4j-backup-dir --database=neo4j
```

```
neo4j-03$ ./bin/neo4j-admin restore --from=/path/to/system-backup-dir --database=system
neo4j-03$ ./bin/neo4j-admin restore --from=/path/to/neo4j-backup-dir --database=neo4j
```

Example 28. Seed using neo4j-admin load.

```
neo4j-01$ ./bin/neo4j-admin load --from=/path/to/system.dump --database=system
neo4j-01$ ./bin/neo4j-admin load --from=/path/to/neo4j.dump --database=neo4j
```

```
neo4j-02$ ./bin/neo4j-admin load --from=/path/to/system.dump --database=system
neo4j-02$ ./bin/neo4j-admin load --from=/path/to/neo4j.dump --database=neo4j
```

```
neo4j-03$ ./bin/neo4j-admin load --from=/path/to/system.dump --database=system
neo4j-03$ ./bin/neo4j-admin load --from=/path/to/neo4j.dump --database=neo4j
```

3. Start the cluster.

   At this point, all of the instances in the Core cluster have been seeded. Between them, the Core Servers have everything necessary to form a cluster. We are ready to start all instances. The cluster will form and the replicated Neo4j DBMS deployment will come online.

   Example 29. Start each of the Core instances.

```
neo4j-01$ ./bin/neo4j start
```

```
neo4j-02$ ./bin/neo4j start
```

```
neo4j-03$ ./bin/neo4j start
```

6.3.3. Seed using the import tool

In order to create a cluster based on imported data, it is recommended to first import the data into a standalone Neo4j DBMS and then use an offline backup to seed the cluster.

1. Import the data.
   a. Deploy a standalone Neo4j DBMS.
   b. Import the data using the import tool.
2. Use `neo4j-admin dump` to create an offline backup of the `neo4j` database.

3. Seed a new cluster using the instructions in Seed from backups.

   Skip the `system` database in this scenario since it is not needed.

6.4. Discovery

This section explains how members of a cluster discover each other.

This section includes:

- Overview
- Discovery using a list of server addresses
- Discovery using DNS with multiple records
- Discovery in Kubernetes

6.4.1. Overview

In order to form or connect to a running cluster, a Core Server or a Read Replica needs to know the addresses of some of the Core Servers. This information is used to bind to the Core Servers in order to run the discovery protocol and get the full information about the cluster. The way in which this is best done depends on the configuration in each specific case.

If the addresses of the other cluster members are known upfront, they can be listed explicitly. This is convenient, but has limitations:

- If Core members are replaced and the new members have different addresses, the list will become outdated. An outdated list can be avoided by ensuring that the new members can be reached via the same address as the old members, but this is not always practical.
- Under some circumstances the addresses are unknown when configuring the cluster. This can be the case, for example, when using container orchestration to deploy a Causal Cluster.

Additional mechanisms for using DNS are provided for the cases where it is not practical or possible to explicitly list the addresses of cluster members to discover.

The discovery configuration is just used for initial discovery and a running cluster will continuously exchange information about changes to the topology. The behavior of the initial discovery is determined by the parameters `causal_clustering.discovery_type` and `causal_clustering.initial_discovery_members`, and is described in the following sections.

Discovery using a list of server addresses

If the addresses of the other cluster members are known upfront, they can be listed explicitly. We use the default `causal_clustering.discovery_type=LIST` and hard code the addresses in the configuration of each machine. This alternative is illustrated by Configure a Core-only cluster.

Discovery using DNS with multiple records

When using initial discovery with DNS, a DNS record lookup is performed when an instance starts up. Once an instance has joined a cluster, further membership changes are communicated amongst Core members as part of the discovery service.

The following DNS-based mechanisms can be used to get the addresses of Core Cluster members for discovery:

`causal_clustering.discovery_type=DNS`
With this configuration, the initial discovery members will be resolved from *DNS A* records to find the IP addresses to contact. The value of `causal_clustering.initial_discovery_members` should be set to a single domain name and the port of the discovery service. For example: `causal_clustering.initial_discovery_members=cluster01.example.com:5000`. The domain name should return an A record for every Core member when a DNS lookup is performed. Each A record returned by DNS should contain the IP address of the Core Server. The configured Core Server will use all the IP addresses from the A records to join or form a cluster.

The discovery port must be the same on all Cores when using this configuration. If this is not possible, consider using the discovery type `SRV` instead.

`causal_clustering.discovery_type=SRV`
With this configuration, the initial discovery members will be resolved from *DNS SRV* records to find the IP addresses/hostnames and discovery service ports to contact. The value of `causal_clustering.initial_discovery_members` should be set to a single domain name and the port set to `0`. For example: `causal_clustering.initial_discovery_members=cluster01.example.com:0`. The domain name should return a single SRV record when a DNS lookup is performed. The SRV record returned by DNS should contain the IP address or hostname, and the discovery port, for the Core Servers to be discovered. The configured Core Server will use all the addresses from the SRV record to join or form a cluster.

Discovery in Kubernetes

A special case is when a Causal Cluster is running in Kubernetes and each Core Server is running as a Kubernetes service. Then the addresses of Core Cluster members can be obtained using the List Service API.

The following settings are used to configure for this scenario:

- Set `causal_clustering.discovery_type=K8S`.
- Set `causal_clustering.kubernetes.label_selector` to a label selector for the Causal Cluster services.
- Set `causal_clustering.kubernetes.service_port_name` to the name of the service port used in the Kubernetes service definition for the Core's discovery port.

With this configuration, `causal_clustering.initial_discovery_members` is not used and any value assigned to it will be ignored.

- The pod running Neo4j must use a service account which has permission to list services. For further information, see the Kubernetes documentation on RBAC authorization or ABAC authorization.
- The configured `causal_clustering.discovery_advertised_address` must exactly match the Kubernetes-internal DNS name, which will be of the form `<service-name>.<namespace>.svc.cluster.local`.

As with DNS-based methods, the Kubernetes record lookup is only performed at start up.

6.5. Intra-cluster encryption

This chapter describes how to secure the cluster communication between server instances.

Securing client to server communication is not covered in this chapter (e.g. Bolt, HTTPS, Backup).

This section includes:

- Introduction
- Example deployment
  - Generate and install cryptographic objects
  - Configure the cluster SSL policy
  - Validate the secure operation of the cluster

6.5.1. Introduction

The security solution for cluster communication is based on standard SSL/TLS technology (referred to jointly as SSL). Encryption is in fact just one aspect of security, with the other cornerstones being authentication and integrity. A secure solution will be based on a key infrastructure which is deployed together with a requirement of authentication.

The SSL support in the platform is documented in detail in SSL framework. This section will cover the specifics as they relate to securing a cluster.

Under SSL, an endpoint can authenticate itself using certificates managed by a Public Key Infrastructure (*PKI*).

It should be noted that the deployment of a secure key management infrastructure is beyond the scope of this manual, and should be entrusted to experienced security professionals. The example deployment illustrated below is for reference purposes only.

6.5.2. Example deployment

The following steps will create an example deployment, and each step is expanded in further detail below.

- Generate and install cryptographic objects
- Configure Causal Clustering with the SSL policy
- Validate the secure operation of the cluster

Generate and install cryptographic objects

The generation of cryptographic objects is for the most part outside the scope of this manual. It will generally require having a PKI with a Certificate Authority (*CA*) within the organization and they should be able to advise here. Please note that the information in this manual relating to the PKI is mainly for illustrative purposes.

When the certificates and private keys have been obtained they can be installed on each of the servers. Each server will have a certificate of its own, signed by a CA, and the corresponding private key. The certificate of the CA is installed into the `trusted` directory, and any certificate signed by the CA will thus be trusted. This means that the server now has the capability of establishing trust with other servers.

> Please be sure to exercise caution when using CA certificates in the `trusted` directory, as any certificates signed by that CA will then be trusted to join the cluster. For this reason, never use a public CA to sign certificates for your cluster. Instead, use an intermediate certificate or a CA certificate which originates from and is controlled by your organization.

In this example we will deploy a mutual authentication setup, which means that both ends of a channel have to authenticate. To enable mutual authentication the SSL policy must have `client_auth` set to REQUIRE (which is the default). Servers are by default required to authenticate themselves, so there is no corresponding server setting.

If the certificate for a particular server is compromised it is possible to revoke it by installing a Certificate Revocation List (*CRL*) in the revoked directory. It is also possible to redeploy using a new CA. For contingency purposes, it is advised that you have a separate intermediate CA specifically for the cluster which can be substituted in its entirety should it ever become necessary. This approach would be much easier than having to handle revocations and ensuring their propagation.

Example 30. Generate and install cryptographic objects

In this example we assume that the private key and certificate file are named *private.key* and *public.crt*, respectively. If you want to use different names you may override the policy configuration for the key and certificate names/locations. We want to use the default configuration for this server so we create the appropriate directory structure and install the certificate:

```
$neo4j-home> mkdir certificates/cluster
$neo4j-home> mkdir certificates/cluster/trusted
$neo4j-home> mkdir certificates/cluster/revoked

$neo4j-home> cp $some-dir/private.key certificates/cluster
$neo4j-home> cp $some-dir/public.crt certificates/cluster
```

Configure the cluster SSL policy

By default, cluster communication is unencrypted. To configure a Causal Cluster to encrypt its intra-cluster communication, set dbms.ssl.policy.cluster.enabled to true.

An SSL policy utilizes the installed cryptographic objects and additionally allows parameters to be configured. We will use the following parameters in our configuration:

Table 15. Example settings

Setting suffix	Value	Comment
client_auth	REQUIRE	Setting this to REQUIRE effectively enables mutual authentication for servers.
ciphers	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	We can enforce a particular single strong cipher and remove any doubt about which cipher gets negotiated and chosen. The cipher chosen above offers Perfect Forward Secrecy (PFS) which is generally desirable. It also uses Advanced Encryption Standard (*AES*) for symmetric encryption which has great support for acceleration in hardware and thus allows performance to generally be negligibly affected.
tls_versions	TLSv1.2	Since we control the entire cluster we can enforce the latest TLS standard without any concern for backwards compatibility. It has no known security vulnerabilities and uses the most modern algorithms for key exchanges, etc.

In the following example we will create and configure an SSL policy that we will use in our cluster.

Example 31. Configure the cluster SSL policy

In this example we assume that the directory structure has been created, and certificate files have been installed, as per the previous example.

We add the following content to our *neo4j.conf* file:

```
dbms.ssl.policy.cluster.enabled=true
dbms.ssl.policy.cluster.tls_versions=TLSv1.2
dbms.ssl.policy.cluster.ciphers=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
dbms.ssl.policy.cluster.client_auth=REQUIRE
```

Any user data communicated between instances will now be secured. Please note that an instance which is not correctly setup would not be able to communicate with the others.

Note that the policy must be configured on every server with the same settings. The actual cryptographic objects installed will be mostly different since they do not share the same private keys and corresponding certificates. The trusted CA certificate will be shared however.

Validate the secure operation of the cluster

To make sure that everything is secured as intended it makes sense to validate using external tooling such as, for example, the open source assessment tools `nmap` or `OpenSSL`.

Example 32. Validate the secure operation of the cluster

In this example we will use the `nmap` tool to validate the secure operation of our cluster. A simple test to perform is a cipher enumeration using the following command:

```
nmap --script ssl-enum-ciphers -p <port> <hostname>
```

The hostname and port have to be adjusted according to our configuration. This can prove that TLS is in fact enabled and that the only the intended cipher suites are enabled. All servers and all applicable ports should be tested.

For testing purposes we could also attempt to utilize a separate testing instance of Neo4j which, for example, has an untrusted certificate in place. The expected result of this test is that the test server is not able to participate in replication of user data. The debug logs will generally indicate an issue by printing an SSL or certificate-related exception.

6.6. Internals of clustering

This section details a few selected internals of a Neo4j Causal Cluster. Understanding the internals is not vital but can be helpful in diagnosing and resolving operational issues.

This section includes:

- Elections and leadership
- Multi-database and the reconciler
- Store copy
- On-disk state

6.6.1. Elections and leadership

The Core Servers in a Causal Cluster use the Raft protocol to ensure consistency and safety. An implementation detail of Raft is that it uses a *Leader* role to impose an ordering on an underlying log with other instances acting as *Followers* which replicate the leader's state. In Neo4j terms this means writes to the database are ordered by the Core instance currently playing the leader role.

If a follower has not heard from the leader for a while, then it can initiate an election and attempt to become the new leader. The follower makes itself a *Candidate* and asks other Cores to vote for it. If it can get a majority of the votes, then it assumes the leader role. Cores will not vote for a candidate which is less up-to-date than itself. There can only be one leader at any time, and that leader is guaranteed to have the most up-to-date log.

It is expected for elections to occur during the normal running of a cluster and they do not pose an issue in and of itself. If you are experiencing frequent re-elections and they are disturbing the operation of the cluster then you should try to figure out what is causing them. Some common causes are environmental issues (e.g. a flaky networking) and work overload conditions (e.g. more concurrent queries and transactions than the hardware can handle).

6.6.2. Multi-database and the reconciler

Databases operate as independent entities in a Neo4j DBMS, both in standalone and in a cluster. Since a cluster consists of multiple independent server instances, the effects of administrative operations like creating a new database happen asynchronously and independently for each server. However, the immediate effect of an administrative operation is to safely commit the desired state in the system database.

The desired state committed in the system database gets replicated and is picked up by an internal component called the reconciler. It runs on every instance and takes the appropriate actions required locally on that instance for reaching the desired state; creating, starting, stopping and dropping databases.

Every database runs in an independent Raft group and since there are two databases in a fresh cluster, `system` and `neo4j`, this means that it also has two Raft groups. Every Raft group also has an independent leader and thus a particular Core server could be the leader for one database and a follower for another.

6.6.3. Store copy

Store copies are initiated when an instance does not have an up-to-date copy of the database. For example, this will be the case when a new instance is joining a cluster (without a seed). It can also happen as a consequence of falling behind the rest of the cluster, for reasons such as connectivity issues or having been shutdown. Upon re-establishing connection with the cluster, an instance will recognize that it is too far behind and fetch a new copy from the rest of the cluster.

A store copy is a major operation which may disrupt the availability of instances in the cluster. Store copies should not be a frequent occurrence in a well-functioning cluster, but rather be an exceptional operation that happens due to specific causes, e.g. network outages or planned maintenance outages. If store copies happen during regular operation, then the configuration of the cluster, or the workload directed at it, might have to be reviewed so that all instances can keep up, and that there is enough of a buffer of Raft logs and transaction logs to handle smaller transient issues.

The protocol used for store copies is robust and configurable. The network requests will be directed at an upstream member according to configuration and they will be retried despite transient failures. The maximum amount of time to retry every request can be modified by setting `causal_clustering.store_copy_max_retry_time_per_request`. If a request fails and the maximum retry time has elapsed then it will stop retrying and the store copy will fail.

Use `causal_clustering.catch_up_client_inactivity_timeout` to configure the inactivity timeout for any particular request.

> This setting is for all requests from the catchup client, including the pulling of transactions.

The default upstream strategy differs for Cores and Read Replicas. Cores will always send the initial request to the leader to get the most up-to-date information about the store. The strategy for the file and index requests for Cores is to vary every other request to a random Read Replica and every other to a random Core member.

Read Replicas use the same strategy for store copies as it uses for pulling transactions. The default is to pull from a random Core member.

If you are running a multi-data center cluster, then upstream strategies for both Cores and Read Replicas can be configured. Remember that for Read Replicas this also affects from where transactions are pulled. See more in Configure for multi-data center operations.

6.6.4. On-disk state

The on-disk state of cluster instances is different to that of standalone instances. The biggest difference being the existence of additional cluster state. Most of the files there are relatively small, but the Raft logs can become quite large depending on the configuration and workload.

It is important to understand that once a database has been extracted from a cluster and used in a standalone deployment, it must not be put back into an operational cluster. This is because the cluster and the standalone deployment now have separate databases, with different and irreconcilable writes applied to them.

> If you try to reinsert the modified database back into the cluster, then the logs and stores will mismatch. Operators should not try to merge standalone databases into the cluster in the optimistic hope that their data will become replicated. That will not happen and will likely lead to unpredictable cluster behavior.

6.7. Settings reference

This section lists the important settings related to running a Neo4j Causal Cluster.

Parameter	Explanation
`dbms.mode`	This setting configures the operating mode of the database. For Causal Clustering, there are two possible modes: `CORE` or `READ_REPLICA`. **Example:** `dbms.mode=READ_REPLICA` will define this server as a Read Replica.
`causal_clustering.minimum_core_cluster_size_at_formation`	Minimum number of Core machines required to form a cluster. **Example:** `causal_clustering.minimum_core_cluster_size_at_formation=3` will specify that the cluster will form when at least three Core members have discovered each other.

Parameter	Explanation
causal_clustering.minimum_core_cluster_size_at_runtime	The minimum size of the dynamically adjusted voting set (which only Core members may be a part of). Adjustments to the voting set happen automatically as the availability of Core members changes, due to explicit operations such as starting or stopping a member, or unintended issues such as network partitions. Please note that this dynamic scaling of the voting set is generally desirable, as under some circumstances it can increase the number of instance failures which may be tolerated. A majority of the voting set must be available before members are voted in or out. **Example:** `causal_clustering.minimum_core_cluster_size_at_runtime=3` will specify that the cluster should not try to dynamically adjust below three Core members in the voting set.

causal_clustering.discovery_type	This setting specifies the strategy that the instance will use to determine the addresses for other instances in the cluster to contact for *bootstrapping*. Possible values are: `LIST`, `DNS`, `SRV`, and `K8S`.
	
	`LIST`
	Treat `causal_clustering.initial_discovery_members` as a list of addresses of Core Servers to contact for discovery.
	
	`DNS`
	Treat `causal_clustering.initial_discovery_members` as a domain name to resolve via DNS. Expect DNS resolution to provide A records with hostnames or IP addresses of Cores to contact for discovery, on the port specified by `causal_clustering.initial_discovery_members`.
	
	`SRV`
	Treat `causal_clustering.initial_discovery_members` as a domain name to resolve via DNS. Expect DNS resolution to provide SRV records with hostnames or IP addresses, and ports, of Cores to contact for discovery.
	
	`K8S`
	Access the Kubernetes list service API to derive addresses of Cores to contact for discovery. Requires `causal_clustering.kubernetes.label_selector` to be a Kubernetes label selector for Kubernetes services running a Core each and `causal_clustering.kubernetes.service_port_name` to be a service port name identifying the discovery port of Core services. The value of `causal_clustering.initial_discovery_members` is ignored for this option.
	
	The value of this setting determines how `causal_clustering.initial_discovery_members` is interpreted. Detailed information about discovery and discovery configuration options is given in Discovery using DNS with multiple records.
	
	Example: `causal_clustering.discovery_type=DNS` combined with `causal_clustering.initial_discovery_members=cluster01.example.com:5000` will fetch all DNS A records for *cluster01.example.com* and attempt to reach Neo4j instances listening on port 5000 for each A record's IP address.

Parameter	Explanation
causal_clustering.initial_discovery_members	The network addresses of an initial set of Core cluster members that are available to bootstrap this Core or Read Replica instance. In the default case, the initial discovery members are given as a comma-separated list of address/port pairs, and the default port for the discovery service is `:5000`. It is good practice to set this parameter to the same value on all Core Servers. It is good practice to set this parameter to the same value on all Core Servers. The behavior of this setting can be modified by configuring the setting `causal_clustering.discovery_type`. This is described in detail in Discovery using DNS with multiple records. **Example:** `causal_clustering.discovery_type=LIST` combined with `core01.example.com:5000,core02.example.com:5000,core03.example.com:5000` will attempt to reach Neo4j instances listening on *core01.example.com*, core01.example.com and core01.example.com; all on port 5000.
causal_clustering.raft_advertised_address	The address/port setting that specifies where the Neo4j instance advertises to other members of the cluster that it will listen for Raft messages within the Core cluster. **Example:** `causal_clustering.raft_advertised_address=192.168.33.20:7000` will listen for for cluster communication in the network interface bound to 192.168.33.20 on port 7000.
causal_clustering.transaction_advertised_address	The address/port setting that specifies where the instance advertises where it will listen for requests for transactions in the transaction-shipping catchup protocol. **Example:** `causal_clustering.transaction_advertised_address=192.168.33.20:6001` will listen for transactions from cluster members on the network interface bound to 192.168.33.20 on port 6001.
causal_clustering.discovery_listen_address	The address/port setting for use by the discovery protocol. This is the setting which will be included in the setting `causal_clustering.initial_discovery_members` which are set in the configuration of the other members of the cluster. **Example:** `causal_clustering.discovery_listen_address=0.0.0.0:5001` will listen for cluster membership communication on any network interface at port 5001.
causal_clustering.raft_listen_address	The address/port setting that specifies which network interface and port the Neo4j instance will bind to for cluster communication. This setting must be set in coordination with the address this instance advertises it will listen at in the setting `causal_clustering.raft_advertised_address`. **Example:** `causal_clustering.raft_listen_address=0.0.0.0:7000` will listen for cluster communication on any network interface at port 7000.

Parameter	Explanation
`causal_clustering.transaction_listen_address`	The address/port setting that specifies which network interface and port the Neo4j instance will bind to for cluster communication. This setting must be set in coordination with the address this instance advertises it will listen at in the setting `causal_clustering.transaction_advertised_address`. **Example:** `causal_clustering.transaction_listen_address=0.0.0.0:6001` will listen for cluster communication on any network interface at port 7000.
`causal_clustering.store_copy_max_retry_time_per_request`	Condition for when store copy should eventually fail. A request is allowed to retry for any amount of attempts as long as the configured time has not been met. For very large stores or other reason that might make transferring of files slow this could be increased. **Example:** `causal_clustering.store_copy_max_retry_time_per_request=60min`

6.7.1. Multi-data center settings

Parameter	Explanation
`causal_clustering.multi_dc_license`	Enables multi-data center features. Requires appropriate licensing. **Example:** `causal_clustering.multi_dc_license=true` will enable the multi-data center features.
`causal_clustering.server_groups`	A list of group names for the server used when configuring load balancing and replication policies. **Example:** `causal_clustering.server_groups=us,us-east` will add the current instance to the groups `us` and `us-east`.
`causal_clustering.upstream_selection_strategy`	An ordered list in descending preference of the strategy which Read Replicas use to choose upstream database server from which to pull transactional updates. **Example:** `causal_clustering.upstream_selection_strategy=connect-randomly-within-server-group,typically-connect-to-random-read-replica` will configure the behavior so that the Read Replica will first try to connect to any other instance in the group(s) specified in `causal_clustering.server_groups`. Should we fail to find any live instances in those groups, then we will connect to a random Read Replica. A value of `user-defined` will enable custom strategy definitions using the setting `causal_clustering.user_defined_upstream_strategy`.
`causal_clustering.user_defined_upstream_strategy`	Defines the configuration of upstream dependencies. Can only be used if `causal_clustering.upstream_selection_strategy` is set to `user-defined`. **Example:** `causal_clustering.user_defined_upstream_strategy=groups(north2); groups(north); halt()` will look for servers in the `north2`. If none are available it will look in the `north` server group. Finally, if we cannot resolve any servers in any of the previous groups, then rule chain will be stopped via `halt()`.
`causal_clustering.load_balancing.plugin`	The load balancing plugin to use. One pre-defined plugin named `server_policies` is available by default. **Example:** `causal_clustering.load_balancing.plugin=server_policies` will enable custom policy definitions.
`causal_clustering.load_balancing.config.server_policies.<policy-name>`	Defines a custom policy under the name `<policy-name>`. Note that load balancing policies are cluster-global configurations and should be defined the exact same way on all core machines. **Example:** `causal_clustering.load_balancing.config.server_policies.north1_only=groups(north1)→min(2); halt();` will define a load balancing policy named `north1_only`. Queries are only sent to servers in the `north1` server group, provided there are two of them available. If there are less than two servers in `north1` then the chain is halted.

Chapter 7. Fabric

This chapter describes the configuration and operation of Neo4j Fabric.

This chapter describes the following:

- Introduction
- Configuration
- Queries
- Further Considerations

7.1. Introduction

This section gives an introduction of Neo4j Fabric.

This section describes the following:

- Overview
- Fabric concepts
- Deployment examples

7.1.1. Overview

Fabric, introduced in Neo4j 4.0, is a way to store and retrieve data in multiple databases, whether they are on the same Neo4j DBMS or in multiple DBMSs, using a single Cypher query. Fabric achieves a number of desirable objectives:

- a unified view of local and distributed data, accessible via a single client connection and user session
- increased scalability for read/write operations, data volume and concurrency
- predictable response time for queries executed during normal operations, a failover or other infrastructure changes
- High Availability and No Single Point of Failure for large data volume.

In practical terms, Fabric provides the infrastructure and tooling for:

- **Data Federation**: the ability to access data available in distributed sources in the form of **disjointed graphs**.
- **Data Sharding**: the ability to access data available in distributed sources in the form of a **common graph partitioned on multiple databases**.

With Fabric, a Cypher query can store and retrieve data in multiple federated and sharded graphs.

7.1.2. Fabric concepts

The fabric database

A Fabric setup includes a Fabric database, that acts as the entry point to a federated or sharded graph infrastructure. This database is also referred in Fabric as *the virtual database*. Drivers and client applications access and use the Fabric database like any other Neo4j database. The exception is that it cannot store any data, and queries against it don't return any data. The Fabric database can be configured only on a standalone Neo4j DBMS, i.e. on a Neo4j DBMS where the configuration setting `dbms.mode` must be set to `SINGLE`.

Fabric graphs

In a Fabric database, data is organized in the form of graphs. Graphs are seen by client applications as local logical structures, where physically data is stored in one or more databases. Databases configured as Fabric graphs may be local, i.e. in the same Neo4j DBMS, or they may be located in external Neo4j DBMSes. The databases are accessible by client applications also from regular local connections in their respective Neo4j DBMSs.

7.1.3. Deployment examples

Fabric constitutes an extremely versatile environment that provides scalability and availability with no single point of failure in various topologies. Users and developers may use applications that can work on a standalone DBMS as well on a very complex and largely distributed infrastructure without the need to apply any change to the queries accessing the Fabric graphs.

Development deployment

In its simplest deployment, Fabric can be used on a single instance, where Fabric graphs are associated to local databases. This approach is commonly used by software developers to create applications that will be deployed on multiple Neo4j DBMSs, or by power users who intend to execute Cypher queries against local disjoint graphs.

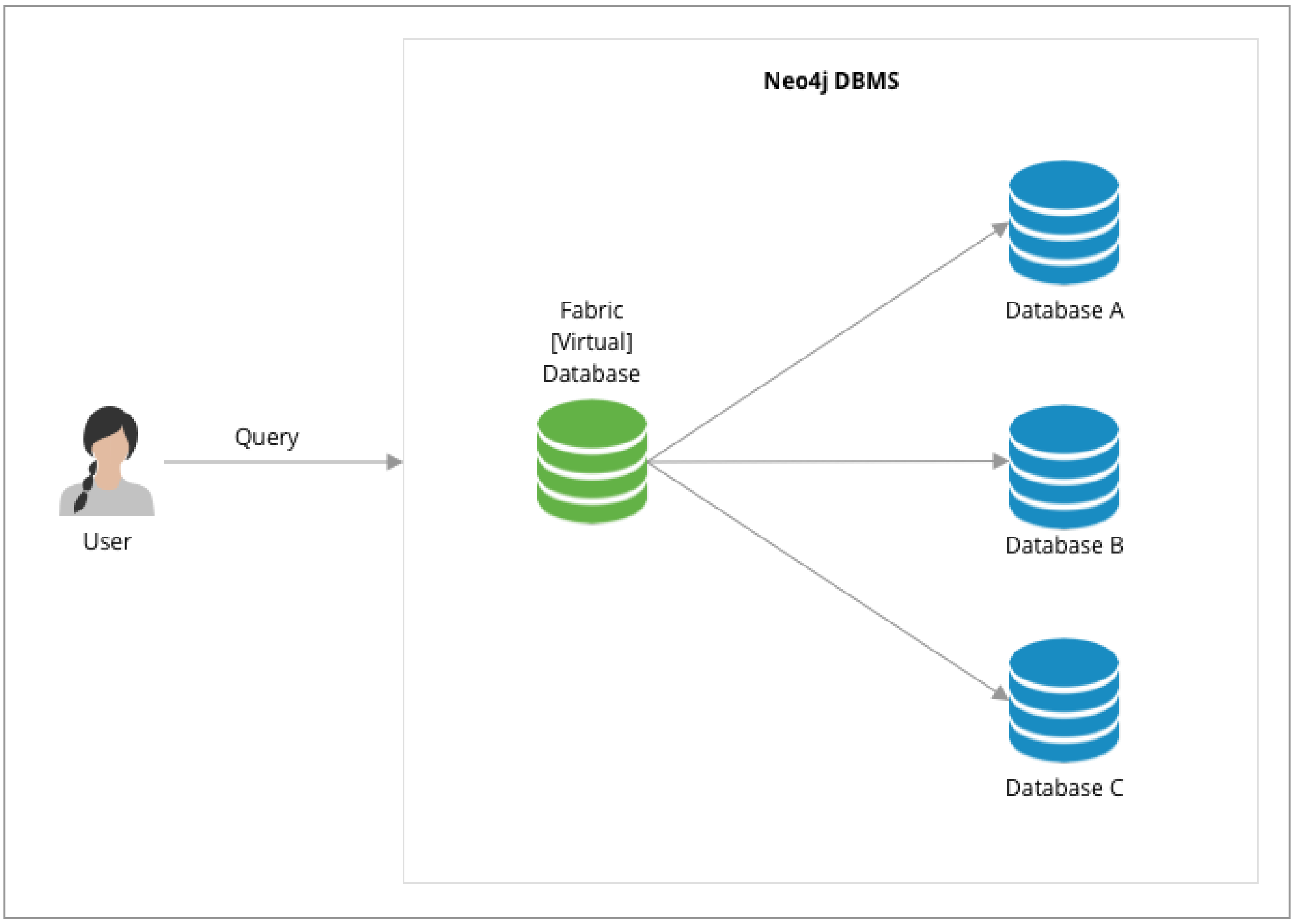

Figure 6. Fabric deployment in a single instance

Cluster deployment with no single point of failure

In this deployment Fabric guarantees access to disjoint graphs in high availability with no single point of failure. Availability if reached by creating redundant entry points for the Fabric Database (i.e. two standalone Neo4j DBMSs with the same Fabric configuration) and a minimum Causal Cluster of three members for data storage and retrieval. This approach is suitable for production environments and it can be used by power users who intend to execute Cypher queries against disjoint graphs.

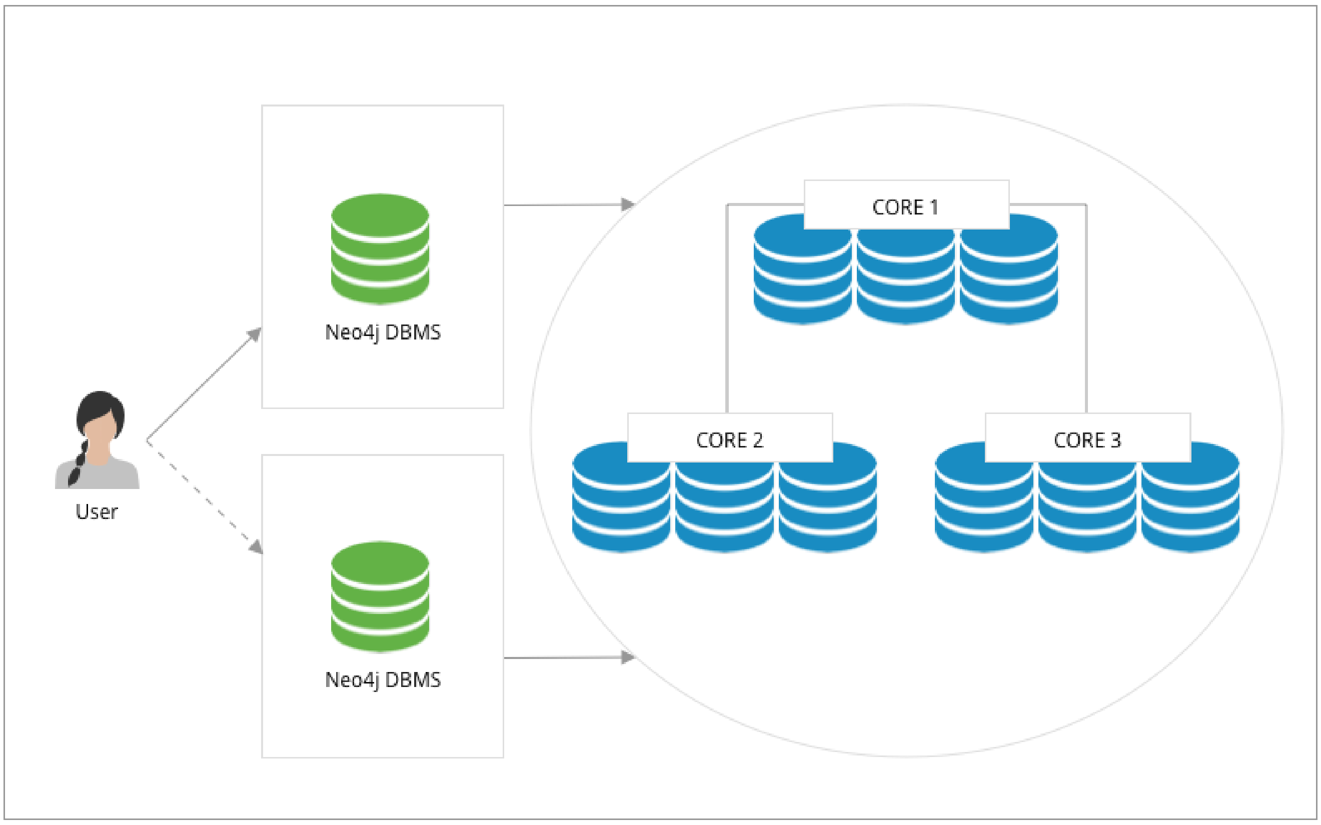

Figure 7. Fabric deployment with no single point of failure

Multi-cluster deployment

In this deployment Fabric provides high scalability and availability with no single point of failure. Disjoint clusters can be sized according to the expected workload and Databases may be colocated in the same cluster or they can be hosted in their own cluster to provide higher throughput. This approach is suitable for production environments where database can be sharded, federated or a combination of the two.

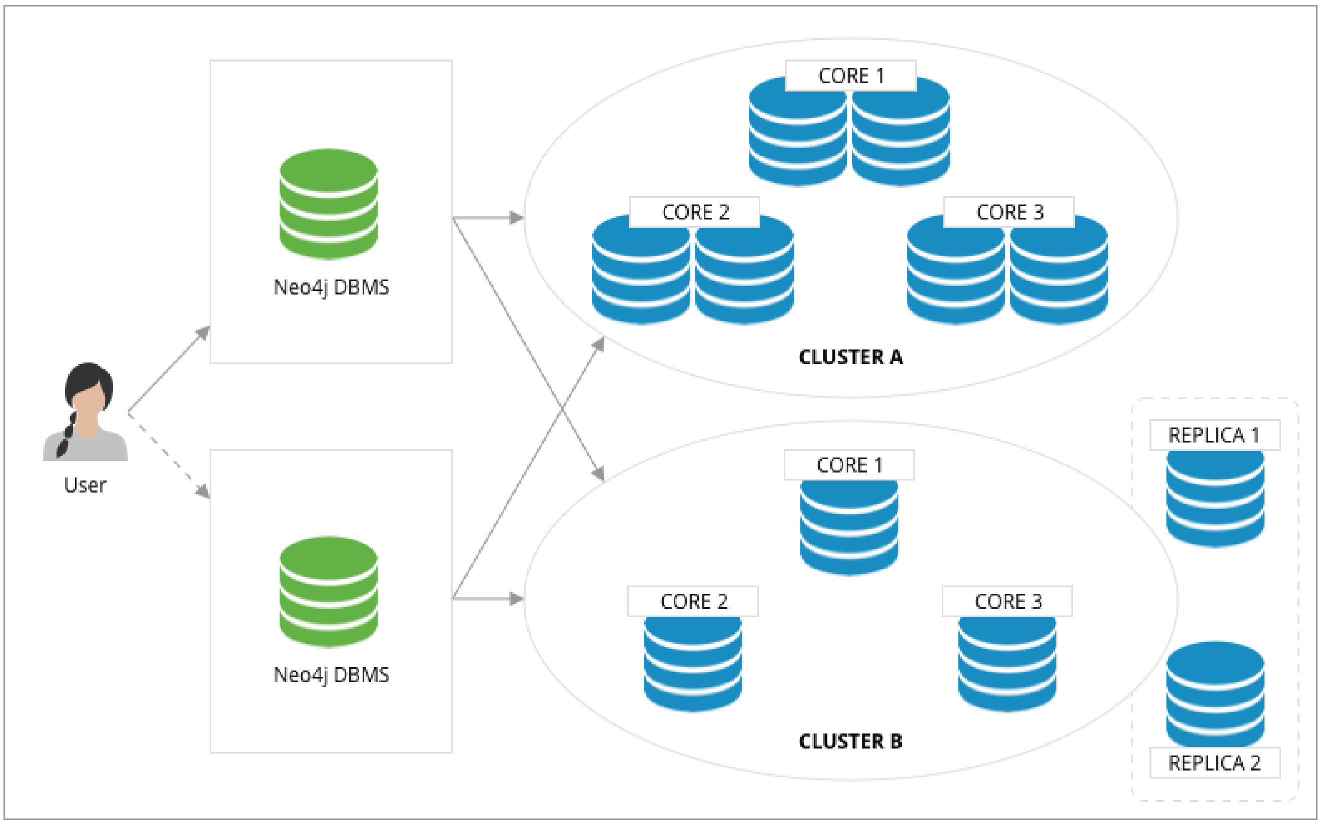

Figure 8. Fabric deployment for scalability with no single point of failure

7.2. Configuration

This section describes how to configure Neo4j Fabric.

This section describes the following:

- Fabric database setup
- Authentication and authorization
- Important settings

7.2.1. Fabric database setup

Fabric must be set on a standalone Neo4j DBMS: the settings in *neo4j.conf* are identified by the `fabric` namespace. The minimal requirements to setup Fabric are:

- A **virtual database name**: this is the entry point used by the client applications to access the Fabric environment.
- One or more **Fabric graph URI and database**: this a reference of a URI and a database for each graph set in the Fabric environment.

Development setup example

Consider a standalone Neo4j DBMS which has two databases, `db1` and `db2`. Note that all databases except for the default and `system` need to be created using the `CREATE DATABASE`.

The simplest configuration of Fabric is:

```
fabric.database.name=example

fabric.graph.0.uri=neo4j://localhost:7687
fabric.graph.0.database=db1

fabric.graph.1.uri=neo4j://localhost:7687
fabric.graph.1.database=db2
```

The configuration associates the Fabric database to an existing database named `example`, and it is accessible using the default URI, i.e. `neo4j://localhost:7687`. The Fabric graphs are uniquely identified with an ID `0` and `1`.

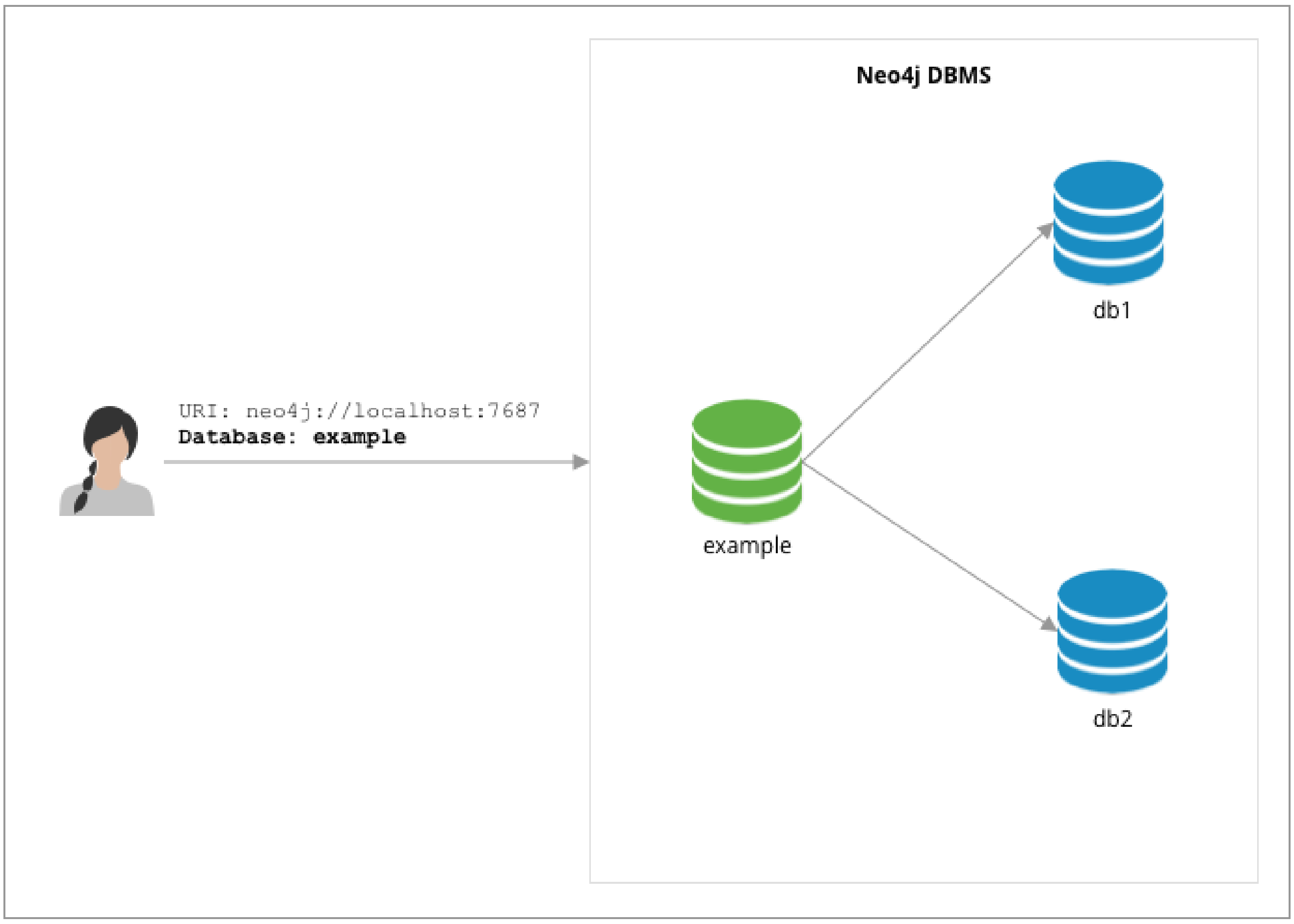

Figure 9. Minimal Fabric setting in a development setup

Naming graphs

Graphs may be identified by their ID or by a name. A graph can be named by adding an extra configuration setting, `fabric.graph.<ID>.name`. In the previous example, assuming that the given names are `graphA` (associated to `db1`) and `graphB` (associated to `db2`), the two additional settings would be:

```
fabric.graph.0.name=graphA
fabric.graph.1.name=graphB
```

Cluster setup with no single point of failure example

In this example, all components are redundant and data is stored in a Causal Cluster. In addition to the settings described in the previous example, a setting with no single point of failure requires the use of the *routing servers* parameter, which specifies a list of standalone Neo4j DBMSs that expose the same Fabric database and configuration. This parameter is required in order to simulate the same connectivity that client applications use with Causal Cluster, i.e. in case of fault of one instance the client application may revert to another existing instance.

Assume that in this example, the data is stored in three databases: db1, db2 and db3. The configuration of Fabric would be:

```
dbms.mode=SINGLE

fabric.database.name=example
fabric.routing.servers=server1:7687,server2:7687

fabric.graph.0.name=graphA
fabric.graph.0.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.0.database=db1

fabric.graph.1.name=graphB
fabric.graph.1.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.1.database=db2

fabric.graph.1.name=graphC
fabric.graph.1.uri=neo4j://core1:7687,neo4j://core2:7687,neo4j://core3:7687
fabric.graph.1.database=db3
```

The configuration above must be added to the *neo4j.conf* file of the Neo4j DBMSs `server1` and `server2`. The parameter `fabric.routing.servers` contains the list of available standalone Neo4j DBMSs hosting the Fabric database. The parameter `fabric.graph.<ID>.uri` can contain a list of URIs, so in case the first server does not respond to the request, the connection can be established to another server that is part of the cluster. The URIs refer to the `neo4j://` schema so that Fabric can retrieve a routing table and can use one of the members of the cluster to connect.

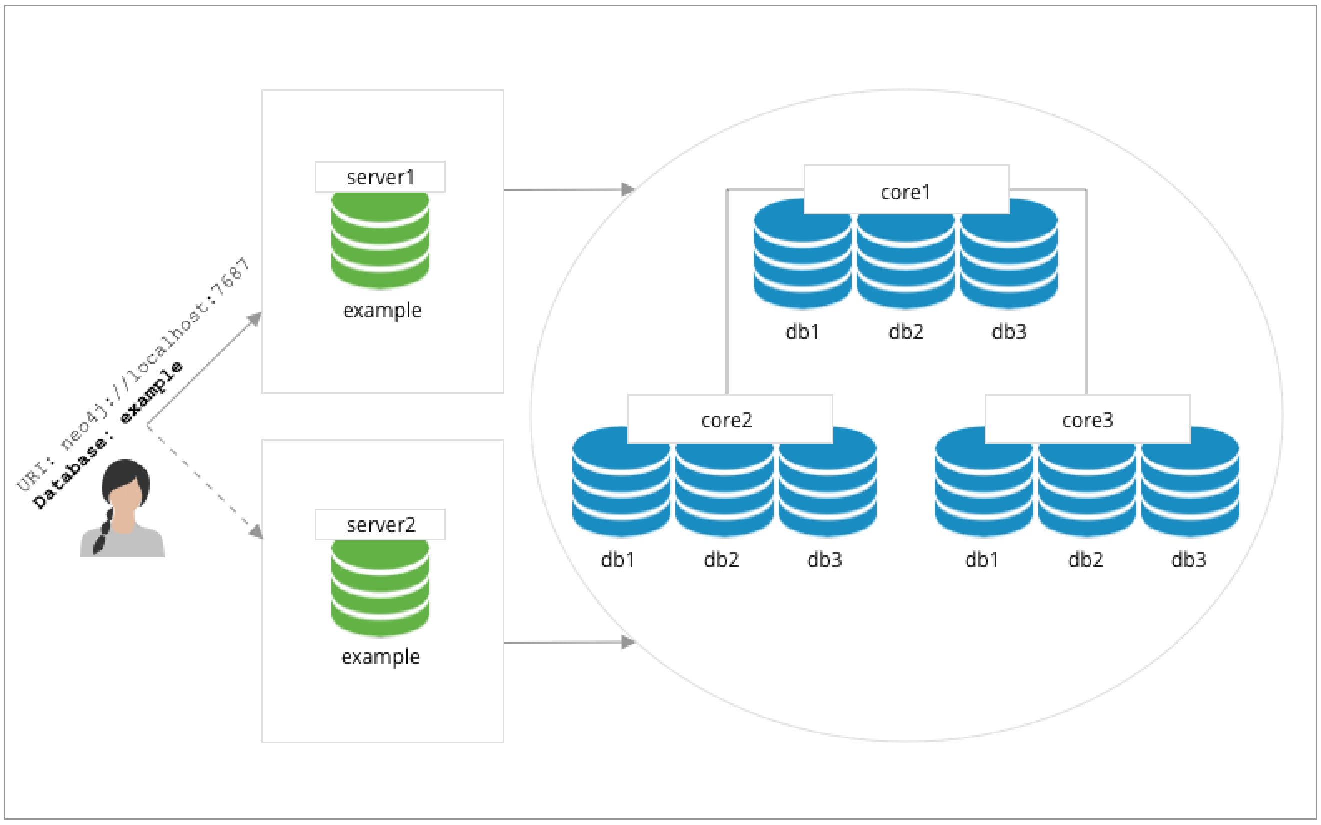

Figure 10. Fabric setting with Causal Cluster and no single point of failure

Cluster routing context

The URIs in the graph settings may include routing contexts. This can be used to associate a Fabric graph with a filtered subset of Causal Cluster members, by selecting a routing policy.

As an example, assuming we have a server policy called `read_replicas` defined in the configuration of the cluster we are targeting, we might set up a Fabric graph that accesses only the read replicas of the cluster.

```
fabric.graph.0.name=graphA
fabric.graph.0.uri=neo4j://core1:7687?policy=read_replicas
fabric.graph.0.database=db1
```

This enables scenarios where queries executed through Fabric are explicitly offloaded to specific instances in clusters.

7.2.2. Authentication and authorization

Credentials

Connections between the Fabric database and the Neo4j DBMSs hosting the data are created using the same credentials that are supplied in the client connection to the Fabric database. It is recommended to maintain a set of user credentials on all the Neo4j DBMSs; if required, a subset of credentials may be set for local access on the remote DBMSs.

User and role administration

User and role administration actions are not automatically propagated to the Fabric environment, therefore security settings must be executed on any DBMS that is part of Fabric.

Privileges on the Fabric database

In order to use all Fabric features, users of Fabric databases need `ACCESS` and `READ` privileges.

7.2.3. Important settings

This section provides general information about Fabric settings and describes the ones important for creating a fabric set-up. Please refer to `Configuration settings` fort the full list of Fabric configuration options.

Fabric settings are divided in the following categories:

- **System Settings**: DBMS-level settings.
- **Graph Settings**: definition and configuration of Fabric graphs.
- **Drivers Settings**: configuration of drivers used to access Neo4j DBMSs and databases associated to Fabric graphs.

System settings

Table 16. Fabric system settings

Parameter	Description
`fabric.database.name`	Name of the Fabric database. Neo4j Fabric currently supports one Fabric database in a standalone Neo4j DBMS.
`fabric.routing.servers`	A comma-separated list of Neo4j DBMSs that share the same Fabric configuration. These DBMSs form a routing group. A client application will route transactions through a Neo4j driver or connector to one of the members of the routing group. A Neo4j DBMS is represented by its Bolt connector address. Example: `fabric.routing.servers=server1:7687,server2:7687`.

Graph settings

The `<ID>` in the following settings is the integer associated to each Fabric graph.

Table 17. Fabric graph settings

Parameter	Description
`fabric.graph.<ID>.uri`	URI of the Neo4j DBMS hosting the database associated to the Fabric graph. Example: `neo4j://somewhere:7687`
`fabric.graph.<ID>.database`	Name of the database associated to the Fabric graph.
`fabric.graph.<ID>.name`	Name assigned to the Fabric graph. The name can be used in Fabric queries.
`fabric.graph.<ID>.driver.*`	Any specific driver setting, i.e. any setting related to a connection to a specific Neo4j DBMS and database.This setting overrides a global driver setting.

When configuring access to a remote DBMS, please make sure that the remote is configured to advertise its address correcly. This is done through either `dbms.default_advertised_address` or `dbms.connector.bolt.advertised_address`. Fabric reads the routing table from the remote DBMS and then connects back using an appropriate entry in that table.

Drivers settings

Fabric uses the Neo4j Java driver to connect to and access the data stored in Neo4j databases associated to Fabric graphs. This section presents the most important parameters available to configure the driver.

Drivers settings are configured with parameters with names of the format:

`fabric.driver.<suffix>`

A setting can be global, i.e. be valid for all the drivers used in Fabric, or it can be specific for a given connection to a Neo4j database associated to a graph. The graph-specific setting overrides the global configuration for that graph.

Example 33. Global drivers setting versus graph-specific drivers setting

A drivers setting for Fabric as the following is valid for all the connections established with the Neo4j DBMSs set in Fabric:

```
fabric.driver.api=RX
```

A graph-specific connection for the database with `ID=6` will override the `fabric.driver.api` setting for that database:

```
fabric.graph.6.driver.api=ASYNC
```

Table 18. Fabric drivers setting suffixes

Parameter suffix	Explanation
`ssl_enabled`	SSL for Fabric drivers is configured using the `fabric` SSL policy. This setting can be used to instruct the driver not to use SSL even though the `fabric` SSL policy is configured. The driver will use SSL if the `fabric` SSL policy is configured, and this setting is set to `true`. This parameter can only be used in `fabric.graph.<graph ID>.driver.ssl_enabled` and not `fabric.driver.ssl_enabled`.
`api`	Determines which driver API will be used. Supported values are `RX` and `ASYNC`. `ASYNC` must be used when the remote instance is 3.5.

Most driver options described in Driver Manual Configuration have an equivalent in Fabric configuration.

7.3. Queries

This section provides examples of queries and Cypher commands that can be used with Neo4j Fabric.

This section describes the following:

- Query a single graph
- Query multiple graphs
- Query all graphs
- Query result aggregation
- Correlated subquery
- Updating query
- Mapping functions
- Fabric built-in functions

In this section we will look at a few example queries that show how to perform a range of different tasks.

The examples in this section make use of the two Cypher clauses: `USE` and `CALL {}`. The syntax is explained in detail in the Cypher Manual:

- See Cypher Manual CALL {} for details about the `CALL {}` clause.
- See Cypher Manual USE for details about the `USE` clause.

7.3.1. Query a single graph

Example 34. Reading and returning data from a single graph.

```
USE example.graphA
MATCH (movie:Movie)
RETURN movie.title AS title
```

The USE clause at the beginning of the query causes the rest of the query to execute against the example.graphA graph.

7.3.2. Query multiple graphs

Example 35. Reading and returning data from two named graphs

```
USE example.graphA
MATCH (movie:Movie)
RETURN movie.title AS title
  UNION
USE example.graphB
MATCH (movie:Movie)
RETURN movie.title AS title
```

The first part of the UNION query executes against the example.graphA graph and the second part executes against the example.graphB graph.

7.3.3. Query all graphs

Example 36. Reading and returning data from all graphs

```
UNWIND example.graphIds() AS graphId
CALL {
  USE example.graph(graphId)
  MATCH (movie:Movie)
  RETURN movie.title AS title
}
RETURN title
```

We call the built-in function example.graphIds() to get the graph IDs for all remote graphs in our Fabric setup. We UNWIND the result of that function to get one record per graph ID. The CALL {} subquery is executed once per incoming record. We use a USE clause in the subquery with a dynamic graph lookup, causing the subquery to execute once against each remote graph. At the end of the main query we simply RETURN the title variable.

7.3.4. Query result aggregation

Example 37. Getting the earliest release year of all movies in all graphs

```
UNWIND example.graphIds() AS graphId
CALL {
  USE example.graph(graphId)
  MATCH (movie:Movie)
  RETURN movie.released AS released
}
RETURN min(released) AS earliest
```

From each remote graph we return the released property of each movie. At the end of the main query we aggregate across the full result to calculate the global minimum.

7.3.5. Correlated subquery

Example 38. Correlated subquery

Assume that graphA contains American movies and graphB contains European movies. Find all European movies released in the same year as the latest released American movie:

```
CALL {
  USE example.graphA
  MATCH (movie:Movie)
  RETURN max(movie.released) AS usLatest
}
CALL {
  USE example.graphB
  WITH usLatest
  MATCH (movie:Movie)
  WHERE movie.released = usLatest
  RETURN movie
}
RETURN movie
```

We query the example.graphA and return the release year of the latest release. We then query the example.graphB. WITH usLatest is an import clause which lets us refer to the usLatest variable inside the subquery. We find all the movies in this graph that fulfill our condition and return them.

7.3.6. Updating query

Example 39. Create a new movie node

```
USE example.graphB
CREATE (m:Movie)
SET m.title = ‘Léon: The Professional’
SET m.tagline = ‘If you want the job done right, hire a professional.’
SET m.released = 1994
```

7.3.7. Mapping functions

Mapping functions are a common Fabric usage pattern. In the previous examples, graphs were identified by providing static graph names in the query. Fabric may be used in scenarios where graphs are identified by a mapping mechanism that can, for example, identify a key of an object contained in a graph. This can be achieved by using user defined functions or other functions that may be already available. These functions ultimately return the ID of a graph in Fabric.

Mapping functions are commonly used in sharding scenarios. In Fabric, shards are associated to graphs, hence mapping functions are used to identify a graph, i.e. a shard.

Refer to Java Reference User-defined functions for details on how to create user-defined functions.

Let’s assume that Fabric is setup in order to store and retrieve data associated to nodes with the label user. User nodes are partitioned in several graphs (shards) in Fabric. Each user has a numerical userId, which is unique in all Fabric. We decide on a simple scheme where each user is located on a graph determined by taking the userId modulo the number of graphs. We create a user-defined function which implements the following pseudo code:

```
sharding.userIdToGraphId(userId) = userId % NUM_SHARDS
```

Assuming we have supplied a query parameter $userId with the specific userId that we are interested in, we use our function in this way:

```
USE example.graph( sharding.userIdToGraphId($userId) )
MATCH (u:User) WHERE u.userId = $userId
RETURN u
```

7.3.8. Fabric built-in functions

Fabric functions are located in a namespace corresponding to a Fabric database in which they are used. The following table provides a description of Fabric built-in functions:

Table 19. Fabric built-in functions

Function	Explanation
`<fabric database name>.graphIds()`	Provides a list of IDs of all remote graph configured for the given Fabric database. This function is supported only in `USE` clauses
`<fabric database name>.graph(graphId)`	Maps a graph ID to a Graph. It accepts a graph ID as a parameter and returns a graph representation accepted by USE clause. This function is supported only in `USE` clauses

7.4. Further considerations

This section presents considerations about Fabric that developers and administrators must be aware of.

DBMS mode

The DBMS hosting the Fabric virtual database cannot be part of a Causal Cluster: it can only be a DBMS with `dbms.mode=SINGLE`.

Database compatibility

Fabric is part of Neo4j DBMS and does not require any special installation or plugin. Fabric graphs can be associated to databases available on Neo4j DBMS version 3.5 or 4.0.

Fabric configuration

The Neo4j DBMSs that host the same Fabric virtual database must have the same configuration settings. The configuration must be kept in-sync and applied by the Database Administrator.

Security credentials

The Neo4j DBMSs that host the same Fabric virtual database must have the same user credentials. Any change of password on a machine that is part of Fabric, must be kept in-sync and applied to all the Neo4j DBMSs that are part of Fabric.

Transactions in Fabric

In Fabric, ACID compliance is guaranteed only within a single graph. This means, that the current version of Fabric does not support transactions that span across multiple graphs. To avoid common mistakes that may lead to data corruption, Fabric does not allow write operations on more than one graph within the same transaction. Transactions with queries that read from multiple graphs, or read from multiple graphs and write to a single graph, are allowed.

Administation commands

Fabric does not support issuing Cypher administration commands, on, or through the Fabric virtual database. Any database management commands, index and constraint management commands or user and security management commands must be issued directly to the DBMSs and databases that are part of the Fabric setup.

Neo4j embedded

Fabric is not available when Neo4j is used as an embedded database in Java applications. Fabric can be used only in a typical client/server mode, when users connect to a Neo4j DBMS from their client application or tool, via Bolt or HTTP protocol.

Chapter 8. Upgrade

This chapter describes how to upgrade Neo4j from an earlier version.

It is recommended that your installation of Neo4j is kept up to date. Upgrading your installation to Neo4j 4.0.0 will ensure that you are provided with improvements in performance and security, as well as any latest bug fixes.

This chapter describes the following:

- Upgrade planning
  - Supported upgrade paths
  - Limitations
  - Prepare to upgrade
- Single-instance upgrade
- Upgrade a Causal Cluster

8.1. Upgrade planning

This section describes how to plan for an upgrade of Neo4j.

This guide describes the following:

- Supported upgrade paths
- Limitations
- Prepare to upgrade

8.1.1. Supported upgrade paths

The following upgrade path is supported:

3.5.any 4.0.0

The following steps are required if you need to upgrade from a version earlier than 3.5:

1. Upgrade to version 3.5.latest by following the instructions in the Neo4j Operations Manual for 3.5.
2. Upgrade to version 4.0.0 as per instructions in this manual.

8.1.2. Limitations

- Neo4j does not support downgrades. If the upgrade is not successful, you have to do a full rollback, including restoring a pre-upgrade backup.
- A Neo4j upgrade must be performed as an isolated operation. If you are planning to upgrade from a single-instance installation to a Causal Cluster, this must be performed separately from the migration to 4.0.0.
- In order to further minimize risk, it is recommended that while migrating, you do not switch from Community Edition to Enterprise Edition, change configuration, perform architectural restructuring, or similar tasks.

8.1.3. Prepare to upgrade

Refer to the 4.0 Migration Guide for details on how to prepare for this upgrade.

8.2. Upgrade a single instance

This section describes how to upgrade a single Neo4j instance.

For instructions on upgrading a Neo4j Causal Cluster, see Upgrade a Causal Cluster.

Pre-upgrade steps

- Read Supported upgrade paths and Limitations regarding supported upgrade paths before you start planning your upgrade.
- Read the 4.0 Migration Guide thoroughly and perform all the steps listed there.

Shutdown and backup

1. If the database is running, shut it down cleanly.
2. Perform and verify backups:
   - Back up *neo4j.conf*.
   - Back up all the files used for encryption, i.e. private key, public certificate, and the contents of the *trusted* and *revoked* directories. The locations of these are described in SSL framework.
   - Verify that you have a full backup that is stored in a safe location, either using the online backup tool or offline backups.

Upgrade

1. Install Neo4j 4.0.0 using one of the following methods, specific to your technology:
   a. If using a tarball or zipfile for installation:
      i. Untar or unzip Neo4j 4.0.0.
      ii. Transfer the new *neo4j.conf* that you prepared in the *Apply configuration changes* step in Upgrade planning.
      iii. Set `dbms.allow_upgrade=true` in *neo4j.conf* of the 4.0.0 installation. Neo4j will fail to start without this configuration.
      iv. Copy the files used for encryption from the old installation to the new one.
      v. Copy the *data* directory from the old installation to the new one. This step is not applicable if you have `dbms.directories.data` pointing to a directory outside of *NEO4J_HOME*.
      vi. If using custom plugins, place the plugins that are adjusted for the new version in the */plugins* directory.
   b. If using a Debian or RPM distribution:
      i. Set `dbms.allow_upgrade=true` in *neo4j.conf*.
      ii. Install Neo4j 4.0.0.
      iii. When prompted, review the differences between the *neo4j.conf* files of the previous version and Neo4j 4.0.0. Transfer any custom settings to the 4.0.0 installation, as noted under the *Apply configuration changes* step in 4.0 Migration Guide → Prepare to upgrade. Make sure to preserve `dbms.allow_upgrade=true` as set in the instruction above. Neo4j will fail to start without this configuration.
      iv. If using custom plugins, place the plugins that are adjusted for the new version in the

/plugins directory.

2. Start up Neo4j 4.0.0. The database upgrade will take place during startup.

   The *neo4j.log* file contains valuable information on how many steps the upgrade will involve and how far it has progressed. For large upgrades, it is a good idea to monitor this log continuously.

Post-upgrade steps

1. When the upgrade has finished, `dbms.allow_upgrade` should be set to `false` or be removed.
2. Restart the database.
3. It is good practice to make a full backup immediately after the upgrade.

8.3. Upgrade a Causal Cluster

This section describes how to upgrade a Neo4j Causal Cluster.

Pre-upgrade steps

1. Read Supported upgrade paths and Limitations regarding supported upgrade paths before you start planning your upgrade.
2. Read the 4.0 Migration Guide thoroughly and perform all the steps listed there.
3. Perform and verify backups:
   - Back up *neo4j.conf*.
   - Back up all the files used for encryption, i.e. private key, public certificate, and the contents of the *trusted* and *revoked* directories. The locations of these are described in SSL framework. You should back up these files on each server in the cluster.
   - Verify that you have a full backup that is stored in a safe location.
4. Prepare a new *neo4j.conf* file for each of the servers in the cluster, following the instructions under the *Apply configuration changes* step in 4.0 Migration Guide Prepare to upgrade.
5. If using custom plugins, make sure that you have the plugins that are adjusted for the new version in an accessible location.

Limitations

- Neo4j does not support downgrades. If the upgrade is not successful, you have to do a full rollback, including restoring a pre-upgrade backup.
- In order to minimize risk, it is recommended that while upgrading, you do not change configuration, perform architectural restructuring, or similar tasks.

Downtime

The migration from 3.5 to 4.x will involve downtime. A test upgrade on a production-like equipment provides information on the duration of the downtime.

Upgrade steps

1. Shut down all the servers in the cluster
2. On one of the Core Servers:
   a. Install Neo4j using one of the following methods, specific to your technology:
      - If using a tarball or zipfile for installation:
        i. Untar or unzip the version of Neo4j that you want to upgrade to.
        ii. Transfer the new *neo4j.conf* that you prepared in the pre-upgrade steps.

            iii. Set `dbms.mode=SINGLE` in *neo4j.conf*.

            iv. Set `dbms.allow_upgrade=true` in *neo4j.conf*. Neo4j will fail to start without this configuration.

            v. Copy the files used for encryption from the old installation to the new one.

            vi. Copy the *data* directory from the old installation to the new one. This step is not applicable if you have `dbms.directories.data` pointing to a directory outside of *NEO4J_HOME*.

            vii. If using custom plugins, place the plugins that are adjusted for the new version in the */plugins* directory.

        - If using a Debian or RPM distribution:

            i. Set `dbms.mode=SINGLE` in *neo4j.conf*.

            ii. Set `dbms.allow_upgrade=true` in *neo4j.conf*.

            iii. Install the version of Neo4j that you want to upgrade to.

            iv. When prompted, review the differences between the *neo4j.conf* files of the previous version and Neo4j 4.0.0. Transfer any custom settings to the new installation, as prepared in the pre-upgrade step. Make sure to preserve `dbms.mode=SINGLE` and `dbms.allow_upgrade=true` as set in the instruction above.

            v. If using custom plugins, place the plugins that are adjusted for the new version in the */plugins* directory.

    b. Start up Neo4j. The database upgrade will take place during startup.

        The *neo4j.log* file contains valuable information on how many steps the upgrade will involve and how far it has progressed. For large upgrades, it is a good idea to monitor this log continuously.

    c. Stop your Neo4j database once again.

    d. Set `dbms.allow_upgrade=false`, or remove it.

    e. Set `dbms.mode=CORE` in *neo4j.conf* to re-enable Causal Clustering in the configuration.

    f. Use `neo4j-admin dump` to make a copy of the database.

    g. Do not yet restart the database.

3. On each of the other Core Servers:

    a. Delete the database directory (in a default configuration, this is the directory *databases/neo4j* which is located in the *data* directory).

    b. Install the version of Neo4j that you want to upgrade to.

    c. Transfer any custom settings to the new installation, as prepared in the pre-upgrade step.

    d. If using a tarball or zipfile for installation: Copy the files used for encryption from the old installation to the new one.

    e. If using custom plugins, place the plugins that are adjusted for the new version in the */plugins* directory.

    f. Perform `neo4j-admin unbind` on the instance.

    g. Using `neo4j-admin load`, restore the upgraded database onto this server.

4. Startup all the Core Servers and see the cluster form.

5. On each of the Read Replica servers:

    a. Stop Neo4j.

    b. Delete the database directory (in a default configuration, this is the directory *databases/neo4j* which is located in the *data* directory).

c. Install the version of Neo4j that you want to upgrade to.
d. Transfer any custom settings to the new installation, as prepared in the pre-upgrade step.
e. If using a tarball or zipfile for installation: Copy the files used for encryption from the old installation to the new one.
f. Perform `neo4j-admin unbind` on the instance.
g. If using custom plugins, place the plugins that are adjusted for the new version in the */plugins* directory.
h. Using `neo4j-admin dump/load`, restore the upgraded database onto this server. Alternatively, you can omit this step and let the Read Replica do a complete store copy.
i. Start the Read Replica and see it join the cluster.

Chapter 9. Backup

This chapter covers how to perform and restore backups of a Neo4j DBMS deployed as a Causal Cluster or a single instance.

This chapter describes the following:

- Backup planning
  - Introduction
  - Online and offline backups
  - Server configuration
  - Databases to backup
  - Storage considerations
  - Cluster considerations
  - Using SSL/TLS for backups
  - Additional files to back up
- Perform a backup
  - Backup command
  - Backup process
  - Memory configuration
- Restore a backup
  - Restore commands
  - Restore a single database
  - Restore a cluster

9.1. Backup planning

This section explains how to prepare for backing up a Neo4j deployment.

This section includes:

- Introduction
- Online and offline backups
- Server configuration
- Databases to backup
- Storage considerations
- Cluster considerations
- Using SSL/TLS for backups
- Additional files to back up

9.1.1. Introduction

Designing an appropriate backup strategy for your Neo4j DBMS is a fundamental part of operations. The backup strategy should take into account elements such as:

- Demands on performance during backup actions.
- Tolerance for data loss in case of failure.
- Tolerance for downtime in case of failure.
- Data volumes.

The backup strategy will answer question such as:

- What type of backup method used; online or offline backups?
- What physical setup meets our demands?
- What backup media — offline or remote storage, cloud storage etc. — should we use?
- How long do we archive backups for?
- With what frequency should we perform backups;

  If using online backups:

  - How often should we perform full backups?
  - How often should we perform incremental backups?
- How do we test recovery routines, and how often?

9.1.2. Online and offline backups

Online backups are typically required for production environments, but it is also possible to perform offline backups.

Offline backups are a more limited method for backing up a database. For example:

- Online backups run against a live Neo4j instance, while offline backups require that the database is shut down.
- Online backups can be full or incremental, but there is no support for backing up incrementally with offline backups.

For more details about offline backups, see Dump and load databases.

The remainder of this chapter is dedicated to describing *online* backups.

9.1.3. Server configuration

The table below lists the basic server parameters relevant to backups. Note that by default the backup service is enabled but only listens on localhost (127.0.0.1) and this needs to be changed if backups are to be taken from another machine.

Table 20. Server parameters for backups

Parameter name	Default value	Description
`dbms.backup.enabled`	`true`	Enable support for running online backups.
`dbms.backup.listen_address`	`127.0.0.1:6362`	Listening server for online backups.

9.1.4. Databases to backup

Since a Neo4j DBMS can host multiple databases and they are backed up independently of one another, it is important to plan a backup strategy for every database and to not forget any databases. In a new deployment there are two databases by default, `neo4j` and `system`. The system database contains configuration, e.g. operational states of databases, security configuration, etc.

9.1.5. Storage considerations

For any backup it is important that you store your data separately from the production system, where there are no common dependencies, and preferably off-site. If you are running Neo4j in the cloud, you could for example use a different availability zone or even a separate cloud provider.

Since backups are kept for a long time, the longevity of archival storage should be considered as part of backup planning.

You may also want to override the settings used for pruning and rotation of transaction log files. The transaction log files are files that keep track of recent changes. Please note that removing transaction logs manually can result in a broken backup.

Recovered servers do not need all of the transaction log files that have already been applied, so it is possible to reduce storage size even further by reducing the size of the files to the bare minimum.

This can be done by setting `dbms.tx_log.rotation.size=1M` and `dbms.tx_log.rotation.retention_policy=3 files`. Alternatively you can use the `--additional-config` override.

9.1.6. Cluster considerations

In a cluster it is possible to take a backup from any server, and each server has two configurable ports capable of serving a backup. These ports are configured by `dbms.backup.listen.address` and `causal_clustering.transaction_listen_address` respectively. Functionally they are equivalent for backups, but separating them can allow some operational flexibility, while using just a single port can simplify the configuration.

It is generally recommended to select Read Replicas to act as backup servers, since they are more numerous than Core Servers in typical cluster deployments. Furthermore, the possibility of performance issues on a Read Replica, caused by a large backup, will not affect the performance or redundancy of the Core Cluster. If a Read Replica is not available, then a Core can be picked based on factors such as its physical proximity, bandwidth, performance, and liveness.

Note that both Read Replicas and Cores can fall behind the leader and be out-of-date. We can look at transaction IDs in order to avoid taking a backup from a server that has lagged too far behind. The latest transaction ID can be found by exposing Neo4j metrics or via Neo4j Browser. To view the latest processed transaction ID (and other metrics) in Neo4j Browser, type `:sysinfo` at the prompt.

9.1.7. Using SSL/TLS for backups

The backup server can be configured to require SSL/TLS. If that is the case then the backup client must also be configured to use it with a compatible policy. Refer to SSL framework to learn how SSL is configured in general. See below table for more details about how configured SSL policies map to the configured ports.

Table 21. Mapping backup configuraton to SSL policies

Backup target address on database server	SSL policy setting on database server	SSL policy setting on backup client	Default port
`dbms.backup.listen_address`	`dbms.ssl.policy.backup`	`dbms.ssl.policy.backup`	`6362`
`causal_clustering.transaction_listen_address`	`dbms.ssl.policy.cluster`	`dbms.ssl.policy.backup`	`6000`

9.1.8. Additional files to back up

The files listed below are not included in online nor offline backups. Make sure to back them up separately.

- If you have a cluster, it may be relevant to back up *neo4j.conf* on each server.
- Back up all the files used for SSL/TLS, i.e. private keys, public certificates, and the contents of the *trusted* and *revoked* directories. The locations of these are described in SSL framework. If you have a cluster, you should back up these files on each server in the cluster.

9.2. Perform a backup

This section describes how to perform an online backup of a Neo4j database.

Remember to back up all of your created databases, including the `system` database.

This section includes:

- Backup command
- Backup process
- Memory configuration

9.2.1. Backup command

A Neo4j database can be backed up in online mode using the `backup` command of `neo4j-admin`. The machine that runs the backup command must have Neo4j installed, but does not need to run a Neo4j server.

Syntax

```
neo4j-admin backup --backup-dir=<path>
                   [--verbose]
                   [--from=<host:port>]
                   [--database=<database>]
                   [--fallback-to-full]
                   [--pagecache=<size>]
                   [--check-consistency]
                   [--check-graph=<true/false>]
                   [--check-indexes=<true/false>]
                   [--check-label-scan-store=<true/false>]
                   [--check-property-owners=<true/false>]
                   [--report-dir=<path>]
                   [--additional-config=<path>]
```

Options

Option	Default	Description
--backup-dir		Directory to place backup in.
--verbose	false	Enable verbose output.
--from	localhost:6362	Host and port of Neo4j.
--database	neo4j	Name of the database to back up. If a backup of the specified database exists in the target directory, then an incremental backup will be attempted.

Option	Default	Description
--fallback-to-full	true	If an incremental backup fails backup will move the old backup to <name>.err.<N> and fallback to a full backup instead.
--pagecache	8M	The size of the page cache to use for the backup process.
--check-consistency	true	If a consistency check should be made.
--check-graph	true	Perform checks between nodes, relationships, properties, types and tokens.
--check-indexes	true	Perform checks on indexes.
--check-label-scan-store	true	Perform checks on the label scan store.
--check-property-owners	false	Perform additional checks on property ownership. This check is **very** expensive in time and memory.
--report-dir	.	Directory where consistency report will be written.
--additional-config		Configuration file to supply additional configuration in.

Exit codes

`neo4j-admin backup` will exit with different codes depending on success or error. In the case of error, this includes details of what error was encountered.

Table 22. Neo4j Admin backup exit codes

Code	Description
`0`	Success.
`1`	Backup failed.
`2`	Backup succeeded but consistency check failed.
`3`	Backup succeeded but consistency check found inconsistencies.

9.2.2. Backup process

The backup client can operate in two slightly different modes referred to as performing a *full backup* or an *incremental backup*. A full backup is always required initially for the very first backup into a target location. Subsequent backups will attempt to use the incremental mode where just the delta of the transcation logs since the last backup are transferred and applied onto the target location. If the required transaction logs aren't available on the backup server then the backup client will fallback to performing a full backup instead, unless `--fallback-to-full` is disabled.

After the backup has been successfully performed the *consistency checker* will be invoked by default. Checking the consistency of the backup is a major operation which can consume significant computational resources, e.g. memory, CPU, I/O.

It is strongly discouraged to run the backup client on a live Neo4j server, especially together with a consistency check. Doing so can adversely affect the server.

To avoid adversely affecting a running server with the resource demands of the backup client it is recommended to take the backup and perform the consistency check on a dedicated machine which has sufficient free resources to perform the consistency check. Another alternative is to decouple the backup operation from the consistency checking and schedule that part of the workflow to happen at a later point in time on a dedicated machine. The value of consistency checking a backup should not be underestimated as it is vital for safe guarding and ensuring the quality of your data.

The transaction log files in the backup are rotated and pruned based on the provided configuration. For example, setting `dbms.tx_log.rotation.retention_policy=3 files` will keep 3 transaction log files in the backup. You can use the `--additional-config` parameter to override this configuration.

Example 40. Full backup

In this example, set environment variables in order to control memory usage.

The page cache is defined by using the command line option `--pagecache`. Further, the `HEAP_SIZE` environment variable will specify the maximum heap size allocated to the backup process.

Now you can perform a full backup:

```
$neo4j-home> export HEAP_SIZE=2G
$neo4j-home> mkdir /mnt/backups
$neo4j-home> bin/neo4j-admin backup --from=192.168.1.34 --backup-dir=/mnt/backups/neo4j --database
=neo4j --pagecache=4G
Doing full backup...
2017-02-01 14:09:09.510+0000 INFO  [o.n.c.s.StoreCopyClient] Copying neostore.nodestore.db.labels
2017-02-01 14:09:09.537+0000 INFO  [o.n.c.s.StoreCopyClient] Copied neostore.nodestore.db.labels 8.00
kB
2017-02-01 14:09:09.538+0000 INFO  [o.n.c.s.StoreCopyClient] Copying neostore.nodestore.db
2017-02-01 14:09:09.540+0000 INFO  [o.n.c.s.StoreCopyClient] Copied neostore.nodestore.db 16.00 kB
...
...
...
```

If you do a directory listing of */mnt/backups* you will now see that you have a backup in a directory called `neo4j`.

Example 41. Incremental backup

This example assumes that you have performed a full backup as per the previous example. In the same way as before, make sure to control the memory usage.

To perform an incremental backup you need to specify the location of your previous backup:

```
$neo4j-home> export HEAP_SIZE=2G
$neo4j-home> bin/neo4j-admin backup --from=192.168.1.34 --backup-dir=/mnt/backups/neo4j --database
=neo4j --pagecache=4G
Destination is not empty, doing incremental backup...
Backup complete.
```

9.2.3. Memory configuration

The following options are available for configuring the memory allocated to the backup client:

Configure heap size for the backup

This is done by setting the environment variable `HEAP_SIZE` before starting the backup program. If not specified by `HEAP_SIZE`, the Java Virtual Machine will choose a value based on server resources. `HEAP_SIZE` configures the maximum heap size allocated for the backup process.

Configure page cache for the backup

The page cache size can be determined for the backup program by using the `--pagecache` option to the `neo4j-admin backup` command. If not explicitly defined, the page cache will default to 8MB.

9.3. Restore a backup

This section describes how to restore from backups of a Neo4j deployment.

This section includes:

- Restore command
- Restore a standalone server
- Restore a cluster

9.3.1. Restore command

A Neo4j database can be restored using the `restore` command of `neo4j-admin`.

Note that restoring a database does not configure it as being created in the Neo4j DBMS. That configuration lives in the *system* database and if a backup of the system database is restored then any configuration for other databases in it will also be restored. For the case where the restored database is not created yet, use the `CREATE DATABASE` command after restoring to create it.

Syntax

```
neo4j-admin restore --from=<path> [--verbose] [--database=<database>] [--force]
```

Options

Option	Default	Description
--from		Path to backup to restore from.
--database	neo4j	Name of database.
--force		If an existing database should be replaced.

9.3.2. Restore a standalone server

To restore from backups, follow these steps:

1. If the server is running, shut it down.
2. Run `neo4j-admin restore` for every database.
3. Start up the server.

Example 42. Restore a standalone server

Restore the databases `system` and `neo4j` from the backups located in */mnt/backups*. Note that the server must be shut down.

```
neo4j-home> bin/neo4j stop
neo4j-home> bin/neo4j-admin restore --from=/mnt/backups/system --database=system --force
neo4j-home> bin/neo4j-admin restore --from=/mnt/backups/neo4j --database=neo4j --force
neo4j-home> bin/neo4j start
```

9.3.3. Restore a cluster

To restore a Causal Cluster from backups, follow these steps:

1. Shut down all server instances in the cluster.
2. Run the `neo4j-admin unbind` command on each of the Core Servers.
3. Restore the backups on each instance, using `neo4j-admin restore`.
4. If you are restoring onto new hardware, please review the *Causal Clustering* settings in *neo4j.conf*.

   In particular, check the settings `causal_clustering.initial_discovery_members`, `causal_clustering.minimum_core_cluster_size_at_formation`, and `causal_clustering.minimum_core_cluster_size_at_runtime`, and ensure that they correctly reflect the new setup.
5. Start the server instances.

Refer to Seed a cluster for more detailed information.

Chapter 10. Authentication and authorization

This chapter describes authentication and authorization in Neo4j.

Ensure that your Neo4j deployment adheres to your company’s information security guidelines by setting up the appropriate authentication and authorization rules.

Ths section describes the following:

- Introduction
- Built-in roles
- Fine-grained access control
- Integration with LDAP
- Manage procedure permissions
- Terminology

The functionality described in this section is applicable to Enterprise Edition. A limited set of user management functions are also available in Community Edition. Native roles overview gives a quick overview of these.

10.1. Introduction

This section provides an overview of authentication and authorization in Neo4j.

Authentication is the process of ensuring that a user is who the user claims to be, while authorization pertains to checking whether the authenticated user is allowed to perform a certain action. Authorization is managed using role-based access control (*RBAC*). Permissions that define access control are assigned to roles, which are in turn assigned to users.

Neo4j has the following auth providers, that can perform user authentication and authorization:

Native auth provider

Neo4j provides a native auth provider that stores user and role information in the `system` database. This option is controlled by the parameter `dbms.security.auth_enabled`, which is set to `true` by default. The Cypher commands to manage users, roles and permissions are described in detail in Cypher Manual → Administration. Various scenarios that describe the use of the native auth provider are available in Fine-grained access control.

LDAP auth provider

Another way of controlling authentication and authorization is through external security software such as Active Directory or OpenLDAP, which is accessed via the built-in LDAP connector. A description of the LDAP plugin using Active Directory is available in Integration with LDAP.

Custom-built plugin auth providers

For clients with specific requirements not satisfied with either native or LDAP, Neo4j provides a plugin option for building custom integrations. It is recommended that this option is used as part of a custom delivery as negotiated with Neo4j Professional Services. The plugin is described in Java Reference → Authentication and authorization plugins.

Kerberos authentication and single sign-on

In addition to LDAP, Native and custom providers, Neo4j supports Kerberos for authentication and single sign-on. Kerberos support is provided via the Neo4j Kerberos Add-On.

10.2. Built-in roles

This section describes the roles that come pre-defined with Neo4j.

Neo4j provides the following native roles:

`reader`
- Read-only access to the data graph (all nodes, relationships, properties).

`editor`
- Read/write access to the data graph.
- Write access limited to creating and changing existing properties key, node labels, and relationship types of the graph.

`publisher`
- Read/write access to the data graph.

`architect`
- Read/write access to the data graph.
- Set/delete access to indexes along with any other future schema constructs.

`admin`
- Read/write access to the data graph.
- Set/delete access to indexes along with any other future schema constructs.
- View/terminate queries.

A user who has no assigned roles will not have any rights or capabilities regarding the data, not even read privileges. A user may have more than one assigned role, and the union of these determine what action(s) on the data may be undertaken by the user.

When an administrator suspends or deletes another user, the following rules apply:

- Administrators can suspend or delete any other user (including other administrators), but not themselves.
- Deleting a user terminates all of the user's running queries and sessions.
- All queries currently running for the deleted user are rolled back.
- The user will no longer be able to log back in (until re-activated by an administrator if suspended).
- There is no need to remove assigned roles from a user prior to deleting the user.

The set of actions on the data and database prescribed by each role are described below. The subset of the functionality which is available with Community Edition is also included:

Table 23. Native roles overview

Action	`reader`	`editor`	`publisher`	`architect`	`admin`	(no role)	Available in Community Edition
Change own password	X	X	X	X	X	X	X
View own details	X	X	X	X	X	X	X
Read data	X	X	X	X	X		X

Action	`reader`	`editor`	`publisher`	`architect`	`admin`	(no role)	Available in Community Edition
View own queries	X	X	X	X	X		
Terminate own queries	X	X	X	X	X		
Write/update/delete data		X	X	X	X		X
Create new types of properties key			X	X	X		X
Create new types of nodes labels			X	X	X		X
Create new types of relationship types			X	X	X		X
Create/drop index/constraint				X	X		X
Create/delete user					X		X
Change another user’s password					X		
Assign/remove role to/from user					X		
Suspend/activate user					X		
View all users					X		X
View all roles					X		
View all roles for a user					X		
View all users for a role					X		
View all queries					X		
Terminate all queries					X		
Dynamically change configuration (see Dynamic settings)					X		

10.3. Fine-grained access control

This section contains a worked example that illustrates various aspects of security and fine-grained access control.

The topics described in this section are:

- The data model
- Security
- Access control using built-in roles
- Sub-graph access control using privileges

10.3.1. The data model

Consider a *healthcare* database, as could be relevant in a medical clinic or hospital. A simple version of this might contain only three labels, representing three entity types:

`(:Patient)`
: Nodes of this type represent patients that visit the clinic because they have some symptoms. Information specific to the patient can be captured in properties:

  - Name
  - SSN
  - Address
  - Date of birth

`(:Symptom)`
: A medical database contains a catalog of known illnesses and associated symptoms, which can be described using properties:

  - Name
  - Description

`(:Disease)`
: A medical database contains a catalog of known illnesses and associated symptoms, which can be described using properties:

  - Name
  - Description

These entities will be modelled as nodes, and connected using relationships of the following types:

`(:Patient)-[:HAS]→(:Symptom)`
: When a patient reports to the clinic, they will describe their symptoms to the nurse or the doctor. The nurse or doctor will then enter this information into the database in the form of connections between the patient node and a graph of known symptoms. Possible properties of interest on this relationship could be:

  - Date - date when symptom was reported

`(:Symptom)-[:OF]→(:Disease)`
: The graph of known symptoms is part of a graph of diseases and their symptoms. The relationship between a symptom and a disease can include a probability factor for how likely or common it is for people with that disease to express that symptom. This will make it easier for the doctor to make a diagnosis using statistical queries.

  - Probability - probability of symptom matching disease

`(:Patient)-[:DIAGNOSIS]→(:Disease)`
: The doctor can use the graph of diseases and their symptoms to perform an initial investigation into the most likely diseases to match the patient. Based on this, and their own assessment of the

patient, they may make a diagnosis which they would persist to the graph through the addition of this relationship with appropriate properties:

- By: doctor’s name
- Date: date of diagnosis
- Description: additional doctors' notes

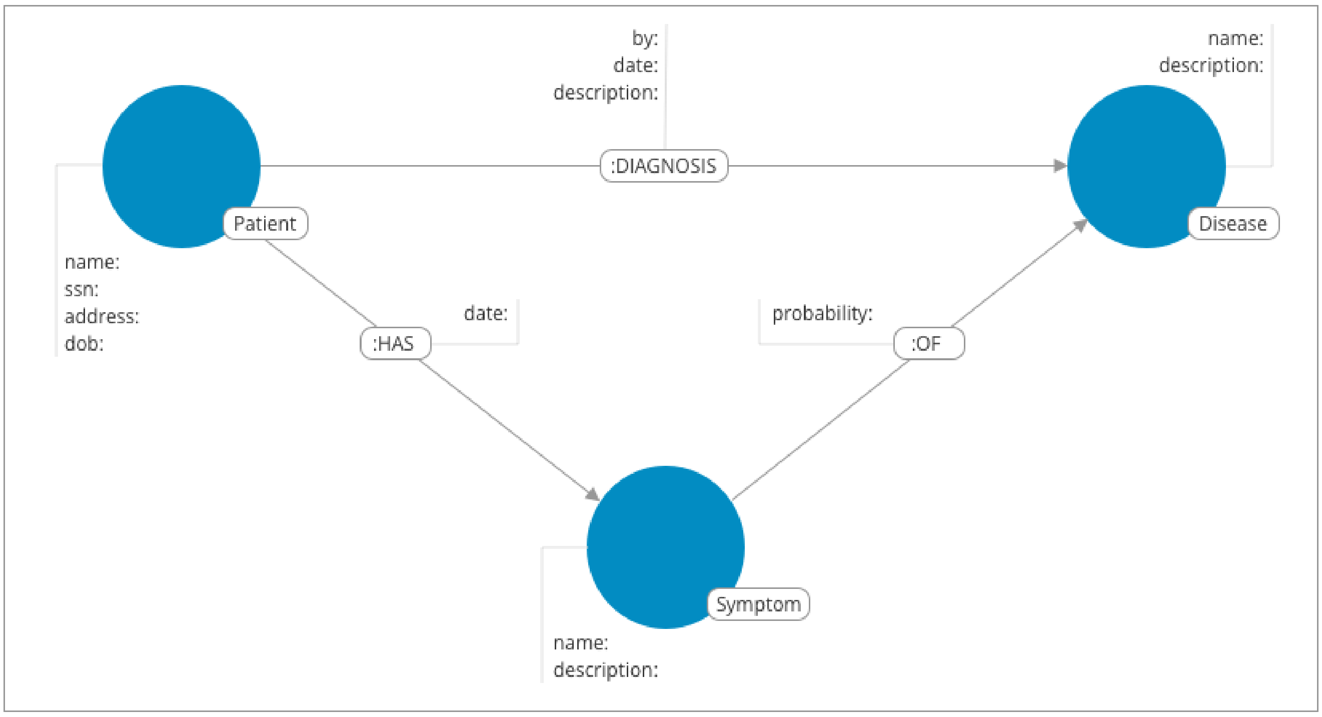

Figure 11. Healthcare use case

The database would be used by a number of different user types, with different needs for access.

- **Doctors** who need to perform diagnosis on patients.
- **Nurses** who need to treat patients.
- **Receptionists** who need to identify and record patient information.
- **Researchers** who need to perform statistical analysis of medical data.
- **IT administrators** who need to administer the database, creating and assigning users.

10.3.2. Security

When building an application for a specific domain, it usual to model the different users within the application itself. However, when working with a database that provides rich user management with roles and privileges, it is possible to model these entirely within the database security model. This results in separation of concerns for the access control to the data and the data itself. We will show two approaches to using Neo4j security features to support the *healthcare* database application. First, a simple approach using built-in roles, and then a more advanced approach using fine-grained privileges for sub-graph access control.

Our *healthcare* example will involve five users of the database:

- Alice the doctor
- Daniel the nurse
- Bob the receptionist
- Charlie the researcher

- Tina the IT administrator

These users can be created using the `CREATE USER` command:

Example 43. Creating users

```
CREATE USER charlie SET PASSWORD $secret1 CHANGE NOT REQUIRED;
CREATE USER alice SET PASSWORD $secret2 CHANGE NOT REQUIRED;
CREATE USER daniel SET PASSWORD $secret3 CHANGE NOT REQUIRED;
CREATE USER bob SET PASSWORD $secret4 CHANGE NOT REQUIRED;
CREATE USER tina SET PASSWORD $secret5 CHANGE NOT REQUIRED;
```

At this point the users have no ability to interact with the database, so we need to grant those capabilities using roles. There are two different ways of doing this, either by using the built-in roles or a more fine-grained access control using privileges and custom roles.

10.3.3. Access control using built-in roles

Neo4j 4.0 comes with a number of built-in roles that cover a number of common needs:

- `reader` - Can only read data from the database.
- `editor` - Can read and update the database, but not expand the schema with new labels, relationship types or property names.
- `publisher` - Can read and edit, as well as add new labels, relationship types and property names.
- `architect` - Has all the capabilities of the publisher as well as the ability to manage indexes and constraints.
- `admin` - Can perform architect actions as well as manage database, users, roles and privileges.

Charlie is a researcher and will not need write access to the database, and so he is assigned the `reader` role. Alice the doctor, Daniel the nurse and Bob the receptionist all need to update the database with new patient information, but do not need to expand the schema with new labels, relationship types, property names or index. We assign them all the `editor` role. Tina is the IT administrator that installs and manages the database. In order to create all other users, Tina is assigned the `admin` role.

Example 44. Granting roles

```
GRANT ROLE reader TO charlie;
GRANT ROLE editor TO alice;
GRANT ROLE editor TO daniel;
GRANT ROLE editor TO bob;
GRANT ROLE admin TO tina;
```

A limitation of this approach is that it does allow all users to see all data in the database, and in many real-world scenarios we would prefer to restrict the users' access. For example, we would want to restrict the `researcher` from being able to read any personal information about the patients, and the `receptionist` should only be able to see the patient records and no more.

These, and more restrictions, could be coded into the application layer. However, it is possible and more secure to enforce these kinds of fine-grained restrictions directly within the Neo4j security model, by creating custom roles and assigning specific privileges to those roles. 2 Since we will be creating new custom roles, the first thing to do is revoke the current roles from the users:

Example 45. Revoking roles

```
REVOKE ROLE reader FROM charlie;
REVOKE ROLE editor FROM alice;
REVOKE ROLE editor FROM daniel;
REVOKE ROLE editor FROM bob;
REVOKE ROLE admin FROM tina;
```

Now the users are unable to do anything, and so we can start building the set of new privileges based on a complete understanding of what we want each user to be able to do.

10.3.4. Sub-graph access control using privileges

With privileges, we can take much more control over what each user is capable of doing. We start by identifying each type of user:

Doctor
: Should be able to read and write most of the graph. We would, however, like to prevent the doctor from reading the patient's address.

Receptionist
: Should be able to read and write all patient data, but not be able to see the symptoms, diseases or diagnoses.

Researcher
: Should be able to perform statistical analysis on all data, except patients' personal information, and as such should not be able to read most patient properties.

Nurse
: The nurse should be able to perform all tasks that both the doctor and the receptionist can do. For this reason, we do not need to create a dedicated role, but can assign nurses to both `doctor` and `receptionist` roles.

IT administrator
: This role is very similar to the built-in `admin` role, except that we wish to restrict access to the patients `SSN`. To achieve this, we can create this role by copying the built-in `admin` role and modifying the privileges of the copy.

Example 46. Creating custom roles

```
CREATE ROLE doctor;
CREATE ROLE receptionist;
CREATE ROLE researcher;
CREATE ROLE itadmin AS COPY OF admin;
```

Before we assign the new roles to Alice, Bob, Daniel, Charlie and Tina, we should define the privileges of each role.

Privileges of `itadmin`

This role was created as a copy of the built-in `admin` role, and so all we need to do is restrict access to the patient's `SSN`:

```
DENY READ {ssn} ON GRAPH healthcare NODES Patient TO itadmin;
```

The complete set of privileges available to users assigned the `itadmin` role can be viewed using the following command:

```
SHOW ROLE itadmin PRIVILEGES
```

```
+-----------------------------------------------------------------------------------------------+
| access    | action     | resource         | graph        | segment            | role      |
+-----------------------------------------------------------------------------------------------+
| "GRANTED" | "read"     | "all_properties" | "*"          | "NODE(*)"          | "itadmin" |
| "GRANTED" | "write"    | "all_properties" | "*"          | "NODE(*)"          | "itadmin" |
| "GRANTED" | "traverse" | "graph"          | "*"          | "NODE(*)"          | "itadmin" |
| "GRANTED" | "read"     | "all_properties" | "*"          | "RELATIONSHIP(*)"  | "itadmin" |
| "GRANTED" | "write"    | "all_properties" | "*"          | "RELATIONSHIP(*)"  | "itadmin" |
| "GRANTED" | "traverse" | "graph"          | "*"          | "RELATIONSHIP(*)"  | "itadmin" |
| "GRANTED" | "access"   | "database"       | "*"          | "database"         | "itadmin" |
| "GRANTED" | "admin"    | "database"       | "*"          | "database"         | "itadmin" |
| "GRANTED" | "schema"   | "database"       | "*"          | "database"         | "itadmin" |
| "GRANTED" | "token"    | "database"       | "*"          | "database"         | "itadmin" |
| "DENIED"  | "read"     | "property(ssn)"  | "healthcare" | "NODE(Patient)"    | "itadmin" |
+-----------------------------------------------------------------------------------------------+
```

In order for the IT-Admin `tina` to be provided these privileges, she must be assigned the new role `itadmin`.

```
GRANT ROLE itadmin TO tina;
```

To demonstrate that Tina is not able to see the patients `SSN`, we can login to `healthcare` as `tina` and run the query:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
"n.name"        "n.ssn" "n.address"              "n.dateOfBirth"

"Mark Smith"    null    "1 secret way, downtown" "1970-01-21"

"Mary Smith"    null    "1 secret way, downtown" "1970-12-30"

"Sally Stone"   null    "1 secret way, downtown" "1971-06-17"

"Jane Anderson" null    "1 secret way, downtown" "1971-10-22"
```

Privileges of `researcher`

Charlie the researcher was previously our only read-only user. We could do something similar to what we did with the `itadmin` role, by copying and modifying the `reader` role. However, we would like to explicitly illustrate how to build a role from scratch. There are various possibilities for building this role using the concepts of *whitelisting* and *blacklisting*:

- **Blacklisting**:

  We could grant the role the ability to find all nodes and read all properties (much like the `reader` role) and then deny read access to the `Patient` properties we want to restrict the researcher from seeing, such as `name`, `SSN` and `address`. This approach is simple but suffers from one problem. If `Patient` nodes are assigned additional properties, *after* we have restricted access, these new properties will automatically be visible to the researcher, which may not be desirable.

  Example 47. Blacklisting

  ```
  GRANT ACCESS ON DATABASE healthcare TO researcherB;
  // First grant access to everything
  GRANT MATCH {*}
      ON GRAPH healthcare
      TO researcherB;
  // Then deny read on specific node properties
  DENY READ {name, address, ssn}
      ON GRAPH healthcare
      NODES Patient
      TO researcherB;
  // And deny traversal of the doctors diagnosis
  DENY TRAVERSE
      ON GRAPH healthcare
      RELATIONSHIPS DIAGNOSIS
      TO researcherB;
  ```

- **Whitelisting**:

  An alternative is to only provide specific access to the properties we wish the researcher to see. Then, the addition of new properties will not automatically make them visible to the researcher. In this case, adding new properties to a `Patient` will not mean that the researcher can see them by default. If we wish to have them visible, we need to explicitly grant read access.

  Example 48. Whitelisting

  ```
  GRANT ACCESS ON DATABASE healthcare TO researcherW;
  // We allow the researcher to find all nodes
  GRANT TRAVERSE
      ON GRAPH healthcare
      NODES *
      TO researcherW;
  // Now only allow the researcher to traverse specific relationships
  GRANT TRAVERSE
      ON GRAPH healthcare
      RELATIONSHIPS HAS, OF
      TO researcherW;
  // Allow reading of all properties of medical metadata
  GRANT READ {*}
      ON GRAPH healthcare
      NODES Symptom, Disease
      TO researcherW;
  // Only allow reading dateOfBirth for research purposes
  GRANT READ {dateofbirth}
      ON GRAPH healthcare
      NODES Patient
      TO researcherW;
  ```

In order to test that Charlie now has the privileges we have specified, we assign him to the `researcherB` role with blacklisting:

```
GRANT ROLE researcherB TO charlie;
```

We can use a version of the `SHOW PRIVILEGES` command to see Charlies access rights:

```
neo4j@system> SHOW USER charlie PRIVILEGES;
```

```
+----------------------------------------------------------------------------------------------------
-----------------+
| access    | action     | resource             | graph        | segment                    | role
| user      |
+----------------------------------------------------------------------------------------------------
-----------------+
| "GRANTED" | "read"     | "all_properties"     | "healthcare" | "NODE(*)"                  |
"researcherB" | "charlie" |
| "GRANTED" | "traverse" | "graph"              | "healthcare" | "NODE(*)"                  |
"researcherB" | "charlie" |
| "DENIED"  | "read"     | "property(address)" | "healthcare" | "NODE(Patient)"            |
"researcherB" | "charlie" |
| "DENIED"  | "read"     | "property(name)"     | "healthcare" | "NODE(Patient)"            |
"researcherB" | "charlie" |
| "DENIED"  | "read"     | "property(ssn)"      | "healthcare" | "NODE(Patient)"            |
"researcherB" | "charlie" |
| "GRANTED" | "read"     | "all_properties"     | "healthcare" | "RELATIONSHIP(*)"          |
"researcherB" | "charlie" |
| "GRANTED" | "traverse" | "graph"              | "healthcare" | "RELATIONSHIP(*)"          |
"researcherB" | "charlie" |
| "DENIED"  | "traverse" | "graph"              | "healthcare" | "RELATIONSHIP(DIAGNOSIS)" |
"researcherB" | "charlie" |
| "GRANTED" | "access"   | "database"           | "healthcare" | "database"                 |
"researcherB" | "charlie" |
+----------------------------------------------------------------------------------------------------
-----------------+
```

Now when Charlie logs into the `healthcare` database and tries to run a command similar to the one used by the `itadmin` above, we will see different results:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
"n.name" "n.ssn" "n.address" "n.dateOfBirth"

null     null    null        "1970-01-21"

null     null    null        "1970-12-30"

null     null    null        "1971-06-17"

null     null    null        "1971-10-22"
```

Only the date of birth is available, so Charlie the researcher may perform statistical analysis, for example. Another query Charlie could try is to find the ten diseases a patient younger than 25 is most likely to be diagnosed with, listed by probability:

```
WITH datetime() - duration({years:25}) AS timeLimit
MATCH (n:Patient)
WHERE n.dateOfBirth > date(timeLimit)
MATCH (n)-[h:HAS]->(s:Symptom)-[o:OF]->(d:Disease)
WITH d.name AS disease, o.probability AS prob
RETURN disease, sum(prob) AS score ORDER BY score DESC LIMIT 10;
```

```
"disease"            "score"

"Chronic Whatitis"    111.30050876448621

"Chronic Someitis"    110.56964390091147

"Acute Yellowitis"    98.82266316401365

"Chronic Otheritis"   80.41346486864003

"Acute Otheritis"     79.82679831362869

"Acute Placeboitis"   78.86090865510758

"Chronic Yellowitis"  77.53519713418886

"Chronic Argitis"     70.04150610048167

"Acute Someitis"      69.45011166554933

"Chronic Placeboitis" 64.36353437805441
```

Privileges of `doctor`

Doctors should be given the ability to read and write almost everything. We would, however, like to remove the ability to read the patients' `address` property. This role can be built from scratch by assigning full read and write access, and then specifically denying access to the `address` property:

```
GRANT ACCESS ON DATABASE healthcare TO doctor;
GRANT TRAVERSE ON GRAPH healthcare TO doctor;
GRANT READ {*} ON GRAPH healthcare TO doctor;
GRANT WRITE ON GRAPH healthcare TO doctor;
DENY READ {address} ON GRAPH healthcare NODES Patient TO doctor;
```

To allow Alice to have these privileges, we grant her this new role:

```
GRANT ROLE doctor TO alice;
```

To demonstrate that Alice is not able to see patient addresses, we can run the query:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
"n.name"        "n.ssn" "n.address" "n.dateOfBirth"

"Mark Smith"    1234610 null        "1970-01-21"

"Mary Smith"    1234640 null        "1970-12-30"

"Sally Stone"   1234641 null        "1971-06-17"

"Jane Anderson" 1234652 null        "1971-10-22"
```

As we can see, the doctor has the expected privileges, including being able to see the SSN, but not the address of each patient.

Privileges of `receptionist`

Receptionists should only be able to manage patient information. They are not allowed to find or read any other parts of the graph:

```
GRANT ACCESS ON DATABASE healthcare TO receptionist;
GRANT MATCH {*} ON GRAPH healthcare NODES Patient TO receptionist;
GRANT WRITE {*} ON GRAPH healthcare TO receptionist;
```

It is currently not possible to be specific on WRITE access, and therefore a user that is granted write access is able to write to all nodes and relationships. For example, the receptionist could create a new Symptom node, even if they are then not able to find that in the database due to the restricted read access.

```
GRANT ROLE receptionist TO bob;
```

With these privileges, if Bob tries to read the entire database, he will still only see the patients:

```
MATCH (n) WITH labels(n) AS labels
RETURN labels, count(*);
```

```
"labels"    "count(*)"

["Patient"] 101
```

However, Bob is able to see all fields of the Patient records:

```
MATCH (n:Patient)
 WHERE n.dateOfBirth < date('1972-06-12')
RETURN n.name, n.ssn, n.address, n.dateOfBirth;
```

```
"n.name"        "n.ssn" "n.address"             "n.dateOfBirth"

"Mark Smith"    1234610 "1 secret way, downtown" "1970-01-21"

"Mary Smith"    1234640 "1 secret way, downtown" "1970-12-30"

"Sally Stone"   1234641 "1 secret way, downtown" "1971-06-17"

"Jane Anderson" 1234652 "1 secret way, downtown" "1971-10-22"
```

Let say that Bob the receptionist wants to remove a patient from the database:

```
MATCH (n:Patient)
 WHERE n.SSN = 1234610
DETACH DELETE n;
```

```
org.neo4j.graphdb.ConstraintViolationException: Cannot delete node<42>, because it still has
relationships. To delete this node, you must first delete its relationships.
```

The reason this fails is that Bob can find the `(:Patient)` node, but does not have sufficient traverse rights to find the outgoing relationships from it. Either he needs to ask Tina the `itadmin` for help for this task, or we can add more privileges to the `receptionist` role:

```
GRANT TRAVERSE ON GRAPH healhcare NODES Symptom, Disease TO receptionist;
GRANT TRAVERSE ON GRAPH healthcare RELATIONSHIPS HAS, DIAGNOSIS TO receptionist;
```

Privileges of nurses

The nurse is not defined as a separate role because, as we have established, nurses have the capabilities of both doctors *and* receptionists. Therefore we can assign both those roles to Daniel the nurse and achieve desired behaviour for a nurse.

```
GRANT ROLE doctor, receptionist TO daniel;
```

Now we can see that the user 'Daniel' has a combined set of privileges:

```
neo4j@system> SHOW USER daniel PRIVILEGES;
```

```
+-----------------------------------------------------------------------------------------------------
---------+
| access    | action     | resource             | graph        | segment            | role           |
user      |
+-----------------------------------------------------------------------------------------------------
---------+
| "GRANTED" | "read"     | "all_properties"     | "healthcare" | "NODE(*)"          | "doctor"       |
"daniel"  |
| "GRANTED" | "write"    | "all_properties"     | "healthcare" | "NODE(*)"          | "doctor"       |
"daniel"  |
| "GRANTED" | "traverse" | "graph"              | "healthcare" | "NODE(*)"          | "doctor"       |
"daniel"  |
| "DENIED"  | "read"     | "property(address)"  | "healthcare" | "NODE(Patient)"    | "doctor"       |
"daniel"  |
| "GRANTED" | "read"     | "all_properties"     | "healthcare" | "RELATIONSHIP(*)"  | "doctor"       |
"daniel"  |
| "GRANTED" | "write"    | "all_properties"     | "healthcare" | "RELATIONSHIP(*)"  | "doctor"       |
"daniel"  |
| "GRANTED" | "traverse" | "graph"              | "healthcare" | "RELATIONSHIP(*)"  | "doctor"       |
"daniel"  |
| "GRANTED" | "access"   | "database"           | "healthcare" | "database"         | "doctor"       |
"daniel"  |
| "GRANTED" | "write"    | "all_properties"     | "healthcare" | "NODE(*)"          | "receptionist" |
"daniel"  |
| "GRANTED" | "read"     | "all_properties"     | "healthcare" | "NODE(Patient)"    | "receptionist" |
"daniel"  |
| "GRANTED" | "traverse" | "graph"              | "healthcare" | "NODE(Patient)"    | "receptionist" |
"daniel"  |
| "GRANTED" | "write"    | "all_properties"     | "healthcare" | "RELATIONSHIP(*)"  | "receptionist" |
"daniel"  |
| "GRANTED" | "access"   | "database"           | "healthcare" | "database"         | "receptionist" |
"daniel"  |
+-----------------------------------------------------------------------------------------------------
---------+
```

10.4. Integration with LDAP

This section describes Neo4j support for integrating with LDAP systems.

This section describes the following:

- Introduction
- Configure the LDAP auth provider
  - Configuration for Active Directory
  - Configuration for openLDAP
- Use 'ldapsearch' to verify the configuration
- The auth cache
- Available methods of encryption
  - Use LDAP with encryption via StartTLS
  - Use LDAP with encrypted LDAPS
- Use a self-signed certificate in a test environment

10.4.1. Introduction

Neo4j supports LDAP which allows for integration with Active Directory, OpenLDAP or other LDAP-compatible authentication services. We will show example configurations where management of federated users is deferred to the LDAP service, using that service's facilities for administration. This means that we completely turn off native Neo4j user and role administration and map LDAP groups to the Neo4j roles.

10.4.2. Configure the LDAP auth provider

All settings need to be defined at server startup time in the default configuration file *neo4j.conf*.

First, configure Neo4j to use LDAP as authentication and authorization provider:

```
# Turn on security:
dbms.security.auth_enabled=true

# Choose LDAP connector as security provider for both authentication and authorization:
dbms.security.authentication_providers=ldap
dbms.security.authorization_providers=ldap
```

Configuration for Active Directory

See below for an example configuration for Active Directory:

```
# Configure LDAP to point to the AD server:
dbms.security.ldap.host=ldap://myactivedirectory.example.com

# Provide details on user structure within the LDAP system:
dbms.security.ldap.authentication.user_dn_template=cn={0},cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_base=cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(cn={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf

# Configure the actual mapping between groups in the LDAP system and roles in Neo4j:
dbms.security.ldap.authorization.group_to_role_mapping=\
  "cn=Neo4j Read Only,cn=Users,dc=neo4j,dc=com"      = reader       ;\
  "cn=Neo4j Read-Write,cn=Users,dc=neo4j,dc=com"     = publisher    ;\
  "cn=Neo4j Schema Manager,cn=Users,dc=neo4j,dc=com" = architect    ;\
  "cn=Neo4j Administrator,cn=Users,dc=neo4j,dc=com"  = admin        ;\
  "cn=Neo4j Procedures,cn=Users,dc=neo4j,dc=com"     = allowed_role

# In case defined users are not allowed to search for themselves, we can specify credentials for a user
with read access to all users and groups.
# Note that this account only needs read-only access to the relevant parts of the LDAP directory and does
not need to have access rights to Neo4j or any other systems.
# dbms.security.ldap.authorization.use_system_account=true
# dbms.security.ldap.authorization.system_username=cn=search-account,cn=Users,dc=example,dc=com
# dbms.security.ldap.authorization.system_password=secret
```

Below is an alternative configuration for Active Directory that allows for logging in with `sAMAccountName`:

```
# Configure LDAP to point to the AD server:
dbms.security.ldap.host=ldap://myactivedirectory.example.com

# Provide details on user structure within the LDAP system:
dbms.security.ldap.authorization.user_search_base=cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(samaccountname={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf

# Configure the actual mapping between groups in the LDAP system and roles in Neo4j:
dbms.security.ldap.authorization.group_to_role_mapping=\
  "cn=Neo4j Read Only,cn=Users,dc=neo4j,dc=com"      = reader       ;\
  "cn=Neo4j Read-Write,cn=Users,dc=neo4j,dc=com"     = publisher    ;\
  "cn=Neo4j Schema Manager,cn=Users,dc=neo4j,dc=com" = architect    ;\
  "cn=Neo4j Administrator,cn=Users,dc=neo4j,dc=com"  = admin        ;\
  "cn=Neo4j Procedures,cn=Users,dc=neo4j,dc=com"     = allowed_role

# In case defined users are not allowed to search for themselves, we can specify credentials for a user
with read access to all users and groups.
# Note that this account only needs read-only access to the relevant parts of the LDAP directory and does
not need to have access rights to Neo4j or any other systems.
dbms.security.ldap.authorization.use_system_account=true
dbms.security.ldap.authorization.system_username=cn=search-account,cn=Users,dc=example,dc=com
dbms.security.ldap.authorization.system_password=secret

# Perform authentication with sAMAccountName instead of DN.
# Using this setting requires dbms.security.ldap.authorization.system_username and
dbms.security.ldap.authorization.system_password to be used, since there is no way to log in through LDAP
directly with the sAMAccountName.
# Instead, the login name will be resolved to a DN that will be used to log in with.
dbms.security.ldap.authentication.use_samaccountname=true
```

Below is an alternative configuration for Active Directory that allows for authenticating users from different organizational units by using the Active Directory attribute `sAMAccountName`:

```
# Configure LDAP to point to the AD server:
dbms.security.ldap.host=ldap://myactivedirectory.example.com

dbms.security.ldap.authentication.user_dn_template={0}@example.com
dbms.security.ldap.authorization.user_search_base=dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=user)(sAMAccountName={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf

# Configure the actual mapping between groups in the LDAP system and roles in Neo4j:
dbms.security.ldap.authorization.group_to_role_mapping=\
"cn=Neo4j Read Only,cn=Users,dc=example,dc=com" = reader ;\
"cn=Neo4j Read-Write,cn=Users,dc=example,dc=com" = publisher ;\
"cn=Neo4j Schema Manager,cn=Users,dc=example,dc=com" = architect ;\
"cn=Neo4j Administrator,cn=Users,dc=example,dc=com" = admin ;\
"cn=Neo4j Procedures,cn=Users,dc=example,dc=com" = allowed_role
```

Specifying the `{0}@example.com` pattern in the `user_dn_template` enables the authentication to start at the root domain. The whole tree is checked to find the user, regardless of where it is located within the tree.

Note that the setting `dbms.security.ldap.authentication.use_samaccountname` is not configured in this example.

Configuration for openLDAP

See below for an example configuration for openLDAP:

```
# Configure LDAP to point to the OpenLDAP server:
dbms.security.ldap.host=myopenldap.example.com

# Provide details on user structure within the LDAP system:
dbms.security.ldap.authentication.user_dn_template=cn={0},ou=users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_base=ou=users,dc=example,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(uid={0}))
dbms.security.ldap.authorization.group_membership_attributes=gidnumber

# Configure the actual mapping between groups in the OpenLDAP system and roles in Neo4j:
dbms.security.ldap.authorization.group_to_role_mapping=\
  101 = reader       ;\
  102 = publisher    ;\
  103 = architect    ;\
  104 = admin        ;\
  105 = allowed_role

# In case defined users are not allowed to search for themselves, we can specify credentials for a user
with read access to all users and groups.
# Note that this account only needs read-only access to the relevant parts of the LDAP directory and does
not need to have access rights to Neo4j or any other systems.
# dbms.security.ldap.authorization.use_system_account=true
# dbms.security.ldap.authorization.system_username=cn=search-account,ou=users,dc=example,dc=com
# dbms.security.ldap.authorization.system_password=search-account-password
```

We would like to draw attention to some details in the configuration examples. A comprehensive overview of LDAP configuration options is available in Configuration settings.

Parameter name	Default value	Description
dbms.security.ldap.authentication.user_dn_template	`uid={0},ou=users,dc=example,dc=com`	Converts usernames into LDAP-specific fully qualified names required for logging in.
dbms.security.ldap.authorization.user_search_base	`ou=users,dc=example,dc=com`	Sets the base object or named context to search for user objects.
dbms.security.ldap.authorization.user_search_filter	`(&(objectClass=*)(uid={0}))`	Sets up an LDAP search filter to search for a user principal.
dbms.security.ldap.authorization.group_membership_attributes	`[memberOf]`	Lists attribute names on a user object that contains groups to be used for mapping to roles.
dbms.security.ldap.authorization.group_to_role_mapping		Lists an authorization mapping from groups to the pre-defined built-in roles `admin`, `architect`, `publisher` and `reader`, or to any other custom-defined roles.

10.4.3. Use `ldapsearch` to verify the configuration

We can use the LDAP command-line tool `ldapsearch` to verify that the configuration is correct, and that the LDAP server is actually responding. We do this by issuing a search command that includes LDAP configuration setting values.

These example searches verify both the authentication (using the `simple` mechanism) and authorization of user 'john'. See the `ldapsearch` documentation for more advanced usage and how to use SASL authentication mechanisms.

With `dbms.security.ldap.authorization.use_system_account=false` (default):

```
#ldapsearch -v -H ldap://<dbms.security.ldap.host> -x -D
<dbms.security.ldap.authentication.user_dn_template : replace {0}> -W -b
<dbms.security.ldap.authorization.user_search_base> "<dbms.security.ldap.authorization.user_search_filter
: replace {0}>" <dbms.security.ldap.authorization.group_membership_attributes>
ldapsearch -v -H ldap://myactivedirectory.example.com:389 -x -D cn=john,cn=Users,dc=example,dc=com -W -b
cn=Users,dc=example,dc=com "(&(objectClass=*)(cn=john))" memberOf
```

With dbms.security.ldap.authorization.use_system_account=true:

```
#ldapsearch -v -H ldap://<dbms.security.ldap.host> -x -D
<dbms.security.ldap.authorization.system_username> -w <dbms.security.ldap.authorization.system_password>
-b <dbms.security.ldap.authorization.user_search_base>
"<dbms.security.ldap.authorization.user_search_filter>"
<dbms.security.ldap.authorization.group_membership_attributes>
ldapsearch -v -H ldap://myactivedirectory.example.com:389 -x -D cn=search-
account,cn=Users,dc=example,dc=com -w secret -b cn=Users,dc=example,dc=com "(&(objectClass=*)(cn=john))"
memberOf
```

Then verify that we get a successful response, and that the value of the returned membership attribute is a group that is mapped to a role in dbms.security.ldap.authorization.group_to_role_mapping.

```
# extended LDIF
#
# LDAPv3
# base <cn=Users,dc=example,dc=com> with scope subtree
# filter: (cn=john)
# requesting: memberOf
#

# john, Users, example.com
dn: CN=john,CN=Users,DC=example,DC=com
memberOf: CN=Neo4j Read Only,CN=Users,DC=example,DC=com

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

10.4.4. The auth cache

The *auth cache* is the mechanism by which Neo4j caches the result of authentication via the LDAP server in order to aid performance. It is configured with the dbms.security.ldap.authentication.cache_enabled and dbms.security.auth_cache_ttl parameters.

```
# Turn on authentication caching to ensure performance
dbms.security.ldap.authentication.cache_enabled=true
dbms.security.auth_cache_ttl=10m
```

Parameter name	Default value	Description
dbms.security.ldap.authentication.cache_enabled	true	Determines whether or not to cache the result of authentication via the LDAP server. Whether authentication caching should be enabled or not must be considered in view of your company's security guidelines.
dbms.security.auth_cache_ttl	600 seconds	Is the time to live (TTL) for cached authentication and authorization info. Setting the TTL to 0 will disable all auth caching. A short TTL will require more frequent re-authentication and re-authorization, which can impact performance. A very long TTL will also mean that changes to the users settings on an LDAP server may not be reflected in the Neo4j authorization behaviour in a timely manner. Valid units are ms, s, m; default unit is s.

An administrator can clear the auth cache to force the re-querying of authentication and authorization information from the federated auth provider system.

Example 49. Clear the auth cache

Use Neo4j Browser or Neo4j Cypher Shell to execute this statement.

```
CALL dbms.security.clearAuthCache()
```

10.4.5. Available methods of encryption

All the following ways of specifying the `dbms.security.ldap.host` parameter are valid. Doing so will configure using LDAP without encryption. Not specifying the protocol or port will result in `ldap` being used over the default port `389`.

```
dbms.security.ldap.host=myactivedirectory.example.com
dbms.security.ldap.host=myactivedirectory.example.com:389
dbms.security.ldap.host=ldap://myactivedirectory.example.com
dbms.security.ldap.host=ldap://myactivedirectory.example.com:389
```

Use LDAP with encryption via StartTLS

To configure Active Directory with encryption via StartTLS, set the following parameters:

```
dbms.security.ldap.use_starttls=true
dbms.security.ldap.host=ldap://myactivedirectory.example.com
```

Use LDAP with encrypted LDAPS

To configure Active Directory with encrypted LDAPS, set `dbms.security.ldap.host` to one of the following. Not specifying the port will result in `ldaps` being used over the default port `636`.

```
dbms.security.ldaps.host=ldaps://myactivedirectory.example.com
dbms.security.ldap.host=ldaps://myactivedirectory.example.com:636
```

This method of securing Active Directory is being deprecated and is therefore not recommended. Instead, use Active Directory with encryption via StartTLS.

10.4.6. Use a self-signed certificate in a test environment

Production environments should always use an SSL certificate issued by a Certificate Authority for secure access to the LDAP server. However, there are scenarios, for example in test environments, where you may wish to use a self-signed certificate on the LDAP server. In these scenarios you will have to tell Neo4j about the local certificate. This is done by entering the details of the certificate using `dbms.jvm.additional` in *neo4j.conf*.

Example 50. Specify details for self-signed certificate on LDAP server

This example shows how to specify details for a self-signed certificate on an LDAP server. The path to the certificate file `MyCert.jks` is an absolute path on the Neo4j server.

```
dbms.jvm.additional=-Djavax.net.ssl.keyStore=/path/to/MyCert.jks
dbms.jvm.additional=-Djavax.net.ssl.keyStorePassword=secret
dbms.jvm.additional=-Djavax.net.ssl.trustStore=/path/to/MyCert.jks
dbms.jvm.additional=-Djavax.net.ssl.trustStorePassword=secret
```

10.5. Manage procedure permissions

This section describes how to configure subgraph access control in Neo4j.

This section describes the following:

- Introduction
- Configuration steps
  - Create a custom role
  - Manage procedure permissions

10.5.1. Introduction

It is possible to restrict a user's access to, and subsequent actions on, specified portions of the graph. For example, a user can be allowed to read, but not write, nodes with specific labels and relationships with certain types.

To implement subgraph access control, you must complete the following steps:

1. Put a procedure or function in place that performs the reads from, and/or writes to, the portion of the graph that you wish to control. This can either be developed in-house, or be made available as a third-party library. Please refer to Java Reference Extending Neo4j for a description on creating and using user-defined procedures and functions.
2. Create one, or several, custom roles, with which to run the procedure described above. These roles can subsequently be assigned the relevant privileges.
3. Configure the procedure so that it can be executed by users with the custom roles.

The steps below assume that the procedure or function is already developed and installed.

10.5.2. Configuration steps

Create a custom role

Create a custom role and manage it either through native user management or through federated user management with LDAP.

Native users scenario

In the native users scenario, a custom role is created and assigned to the relevant user(s).

Example 51. Native users scenario

In this example, we will use Cypher to create a custom `accounting` role and assign it to a pre-existing user, `billsmith`.

```
CALL dbms.security.createRole('accounting')

CALL dbms.security.addRoleToUser('accounting', 'billsmith')
```

Federated users scenario (LDAP)

In the LDAP scenario, the LDAP user group is mapped to a custom role in Neo4j.

Example 52. Federated users scenario (LDAP)

In this example, we will use Cypher to create a custom `accounting` role.

```
CALL dbms.security.createRole('accounting')
```

We will then map the `accounting` role to the LDAP group with groupID `101`.

```
dbms.security.realms.ldap.authorization.group_to_role_mapping=101=accounting
```

Manage procedure permissions

In standard use, procedures and functions are executed according to the same security rules as regular Cypher statements. For example, users assigned any one of the native roles `publisher`, `architect` and `admin` will be able to execute a procedure with `mode=WRITE`, whereas a user assigned only the `reader` role will not be allowed to execute the procedure.

For the purpose of subgraph access control, we allow specific roles to execute procedures that they would otherwise be prevented from accessing through their assigned native roles. The user is given the privilege that comes with the mode of the procedure, during the execution of the procedure only. The following two parameters are used to configure the desired behavior:

`dbms.security.procedures.default_allowed`

The setting `dbms.security.procedures.default_allowed` defines a single role that is allowed to execute any procedure or function that is not matched by the `dbms.security.procedures.roles` configuration.

Example 53. Configure a default role that can execute procedures and functions

Assume that we have the following configuration:

```
dbms.security.procedures.default_allowed=superAdmin
```

This will have the following effects:

- If the setting `dbms.security.procedures.roles` is left unconfigured, the role `superAdmin` will be able to execute all custom procedures and functions.
- If the setting `dbms.security.procedures.roles` has some roles and functions defined, the role `superAdmin` will be able to execute all custom procedures and functions that are *not* configured by `dbms.security.procedures.roles`.

`dbms.security.procedures.roles`

The `dbms.security.procedures.roles` setting provides fine-grained control over procedures.

Example 54. Configure roles for the execution of specific procedures

Assume that we have the following configuration:

```
dbms.security.procedures.roles=apoc.convert.*:Converter;apoc.load.json.*:Converter,DataSource;apoc
.trigger.add:TriggerHappy
```

This will have the following effects:

- All users with the role `Converter` will be able to execute all procedures in the `apoc.convert` namespace.
- All users with the roles `Converter` and `DataSource` will be able to execute procedures in the `apoc.load.json` namespace.
- All users with the role `TriggerHappy` will be able to execute the specific procedure `apoc.trigger.add`.

A procedure will fail if it attempts to execute database operations that violates its mode. For example, a procedure assigned the mode of `READ` will fail if it is programmed to do write actions. This will happen regardless of user or role configuration.

10.6. Terminology

This section lists the relevant terminology related to authentication and authorization in Neo4j.

The following terms are relevant to role-based access control within Neo4j:

active user
A user who is active within the system and can perform actions prescribed by any assigned roles on the data. This is in contrast to a suspended user.

administrator
: This is a user who has been assigned the admin role.

current user
: This is the currently logged-in user invoking the commands described in this chapter.

password policy
: The password policy is a set of rules of what makes up a valid password. For Neo4j, the following rules apply:

  * The password cannot be the empty string.
  * When changing passwords, the new password cannot be the same as the previous password.

role
: This is a collection of actions — such as read and write — permitted on the data.

suspended user
: A user who has been suspended is not able to access the database in any capacity, regardless of any assigned roles.

user

  * A user is composed of a username and credentials, where the latter is a unit of information, such as a password, verifying the identity of a user.
  * A user may represent a human, an application etc.

Chapter 11. Security

This chapter covers important security aspects in Neo4j.

Ensure your physical data security by following industry best practices with regard to server and network security.

This chapter includes the following:

- Securing extensions
- SSL framework
- Credentials handling in Neo4j Browser
- Security checklist

Additionally, logs can be useful for continuous analysis, or for specific investigations. Facilities are available for producing security event logs as well as query logs as described in Monitoring.

Refer to Authentication and authorization for information on how to manage users and their authentication and authorization.

11.1. Securing extensions

This section describes how to ensure the security of custom-written additions in Neo4j.

Neo4j can be extended by writing custom code which can be invoked directly from Cypher, as described in Java Reference Procedures and functions. This section describes how to ensure the security of these additions.

11.1.1. Sandboxing

Neo4j provides sandboxing to ensure that procedures do not inadvertently use insecure APIs. For example, when writing custom code it is possible to access Neo4j APIs that are not publicly supported, and these internal APIs are subject to change, without notice. Additionally, their use comes with the risk of performing insecure actions. The sandboxing functionality limits the use of extensions to publicly supported APIs, which exclusively contain safe operations, or contain security checks.

The configuration setting `dbms.security.procedures.unrestricted` provides the possibility to circumvent the sandboxing functionality by defining a comma-separated list of procedures and/or user-defined functions that are allowed full access to the database. The list may contain fully-qualified procedure names, and partial names with the wildcard `*`.

Any attempt to load an extension which contains an unsupported API, which has not been marked as allowed in `dbms.security.procedures.unrestricted`, will result in a warning in the security log. The warning will point out that the extension does not have access to the components it is trying to load. Additionally, a mocked procedure will be loaded with the procedure's name. Calling the mocked procedure will result in an error, saying that the procedure failed to load due to needing more permissions.

Example 55. Sandboxing

In this example we assume that the procedure `my.extensions.example`, as well as some procedures in the `my.procedures` library, make use of unsupported APIs. In order to allow the running of these procedures we configure the setting as shown below.

```
# Example sandboxing
dbms.security.procedures.unrestricted=my.extensions.example,my.procedures.*
```

11.1.2. White listing

White listing can be used to allow loading only a few extensions from a larger library.

The configuration setting `dbms.security.procedures.whitelist` is used to name certain procedures that should be available from a library. It defines a comma-separated list of procedures that are to be loaded. The list may contain both fully-qualified procedure names, and partial names with the wildcard `*`.

Example 56. White listing

In this example we wish to allow the use of the method `apoc.load.json` as well as all the methods under `apoc.coll`. We do not want to make available any additional extensions from the `apoc` library, other than the ones matching these criteria.

```
# Example white listing
dbms.security.procedures.whitelist=apoc.coll.*,apoc.load.*
```

There are a few things that should be noted about `dbms.security.procedures.whitelist`:

- If using this setting, no extensions other than those listed will be loaded. In particular, if it is set to the empty string, no extensions will be loaded.
- The default of the setting is `*`. This means that if you do not explicitly give it a value (or no value), all libraries in the *plugins* directory will be loaded.
- If the extensions pointed out by this parameter are programmed to access internal APIs, they also have to be explicitly allowed, as described in Sandboxing.

11.2. SSL framework

This section describes SSL/TLS integration for securing communication channels in Neo4j.

Neo4j supports the securing of communication channels using standard SSL/TLS technology.

This section describes the following:

- Introduction
- Certificates
- Configuration
    - Settings
- Choosing an SSL provider

* Terminology

11.2.1. Introduction

The SSL support is enabled per communication channel and requires SSL certificates encoded in the `PEM` format. The process is described in the following sections.

11.2.2. Certificates

The instructions in this section assume that you have already acquired the required certificates.

All certificates must be in the `PEM` format, and they can be combined into one file. The private key is also required to be in the `PEM` format. Multi-host and wildcard certificates are supported.

11.2.3. Configuration

The SSL policies are configured by assigning values to parameters of the following format:

```
dbms.ssl.policy.<scope>.<setting-suffix>
```

The `scope` is the name of the communication channel, and must be one of `bolt`, `https`, `cluster`, `backup` or `fabric`.

Each policy needs to be explicitly enabled by setting:

```
dbms.ssl.policy.<scope>.enabled=true
```

Settings

The valid values for `setting-suffix` are described below.

Setting suffix	Description	Default value
	Basic	
`enabled`	Setting this to `true` will enable this policy.	`false`
`base_directory`	The base directory under which cryptographic objects are searched for by default.	`certificates/<scope>`
`private_key`	The private key used for authenticating and securing this instance.	`private.key`
`private_key_password`	The passphrase to decode the private key. Only applicable for encrypted private keys.	
`public_certificate`	A public certificate matching the private key signed by a Certificate Authority (CA).	`public.crt`
`trusted_dir`	A directory populated with certificates of trusted parties.	`trusted/`
`revoked_dir`	A directory populated with certificate revocation lists (CRLs).	`revoked/`
	Advanced	

Setting suffix	Description	Default value
verify_hostname	Enabling this setting will turn on client-side hostname verification. After the client has received the servers public certificate, it will compare the address it used against the certificate Common Name (CN) and Subject Alternative Names (SAN) fields. If the address used doesn't match those fields, the client will disconnect.	false
ciphers	A comma-separated list of ciphers suits that will be allowed during cipher negotiation. Valid values depend on the current JRE and SSL provider, see note below for examples.	Java platform default allowed cipher suites
tls_versions	A comma-separated list of allowed TLS versions.	TLSv1.2
client_auth	Whether or not clients must be authenticated. Setting this to REQUIRE effectively enables mutual authentication for servers. Available values given to this setting are NONE, OPTIONAL, or REQUIRE.	OPTIONAL for bolt and https and REQUIRE for cluster and backup.
trust_all	Setting this to true will result in all clients and servers being trusted. The content of the trusted_dir directory will be ignored. Use of this is discouraged, since it will not offer security. It is provided as a mean of debugging.	false

Ciphers supported by the Oracle JRE can be found here: https://docs.oracle.com/en/java/javase/11/docs/specs/security/standard-names.html#jsse-cipher-suite-names

For security reasons, Neo4j will not attempt to automatically create any of these directories. The creation of an SSL policy therefore requires the appropriate file system structure to be set up manually. Note that the existence of the directories is mandatory, as well as the presence of the certificate file and the private key. Ensure correct permissions are set on the private key, such that only the Neo4j user can read it.

Example 57. Enable Bolt SSL

In this example we will configure Bolt to use SSL/TLS. As the simplest configuration possible, we will just enable it in *neo4j.conf* and rely on the default values:

```
dbms.ssl.policy.bolt.enabled=true
```

Then create the mandatory directories:

```
$neo4j-home> mkdir certificates/bolt
$neo4j-home> mkdir certificates/bolt/trusted
$neo4j-home> mkdir certificates/bolt/revoked
```

Finally, place the files *private.key* and *public.crt* into the base directory:

```
$neo4j-home> cp /path/to/certs/private.key certificates/bolt
$neo4j-home> cp /path/to/certs/public.crt certificates/bolt
```

The base directory should now show the following listings:

```
$neo4j-home> ls certificates/bolt
-r-------- ... private.key
-rw-r--r-- ... public.crt
drwxr-xr-x ... revoked
drwxr-xr-x ... trusted
```

11.2.4. Choosing an SSL provider

The secure networking in Neo4j is provided through the Netty library, which supports both the native JDK SSL provider as well as Netty-supported OpenSSL derivatives.

Follow these steps to utilize OpenSSL:

1. Install a suitable dependency into the `plugins/` folder of Neo4j. Dependencies can be downloaded from https://netty.io/wiki/forked-tomcat-native.html.
2. Set `dbms.netty.ssl.provider=OPENSSL`.

Using OpenSSL can significantly improve performance, especially for AES-GCM-cryptos, e.g. TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.

11.2.5. Terminology

The following terms are relevant to SSL support within Neo4j:

Certificate Authority (CA)
: A trusted entity that issues electronic documents that can verify the identity of a digital entity. The term commonly refers to globally recognized CAs, but can also include internal CAs that are trusted inside of an organization. The electronic documents are digital certificates. They are an essential part of secure communication, and play an important part in the Public Key Infrastructure.

Certificate Revocation List (CRL)
: In the event of a certificate being compromised, that certificate can be revoked. This is done by means of a list (located in one or several files) spelling out which certificates are revoked. The CRL

is always issued by the CA which issues the corresponding certificates.

cipher
: An algorithm for performing encryption or decryption. In the most general implementation of encryption of Neo4j communications, we make implicit use of ciphers that are included as part of the Java platform. The configuration of the SSL framework also allows for the explicit declaration of allowed ciphers.

communication channel
: A means for communicating with the Neo4j database. Available channels are:

- Bolt client traffic
- HTTPS client traffic
- intra-cluster communication
- backup traffic

cryptographic objects
: A term denoting the artifacts private keys, certificates and CRLs.

configuration parameters
: These are the parameters defined for a certain ssl policy in *neo4j.conf*.

certificate
: SSL certificates are issued by a trusted certificate authority (*CA*). The public key can be obtained and used by anyone to encrypt messages intended for a particular recipient. The certificate is commonly stored in a file named *<file name>.crt*. This is also referred to as the public key.

SAN
: SAN is an acronym for *Subject Alternative Names*. It is an extension to certificates that one can include optionally. When presented with a certificate that includes SAN entries, it is recommended that the address of the host is checked against this field. Verifying that the hostname matches the certificate SAN helps prevent attacks where a rogue machine has access to a valid key pair.

SSL
: SSL is an acronym for *Secure Sockets Layer*, and is the predecessor of TLS. It is common to refer to SSL/TLS as just SSL. However, the modern and secure version is TLS, and this is also the default in Neo4j.

SSL policy
: An SSL policy in Neo4j consists of a a digital certificate and a set of configuration parameters defined in *neo4j.conf*.

private key
: The private key ensures that encrypted messages can be deciphered only by the intended recipient. The private key is commonly stored in a file named *<file name>.key*. It is important to protect the private key to ensure the integrity of encrypted communication.

Public Key Infrastructure (PKI)
: A set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption.

public key
: The public key can be obtained and used by anyone to encrypt messages intended for a particular recipient. This is also referred to as the certificate.

TLS protocol
The cryptographic protocol that provides communications security over a computer network. The Transport Layer Security (TLS) protocol and its predecessor, the Secure Sockets Layer (SSL) protocol are both frequently referred to as "SSL".

TLS version
A version of the TLS protocol.

11.3. Browser credentials handling

This section explains how to control how credentials are handled in Neo4j Browser.

Neo4j Browser has two mechanisms for avoiding users having to repeatedly enter their Neo4j credentials.

First, while the Browser is open in a web browser tab, it ensures that the existing database session is kept alive. This is subject to a timeout. The timeout is configured in the setting `browser.credential_timeout`. The timeout is reset whenever there is user interaction with the Browser.

Second, the Browser can also cache the user's Neo4j credentials locally. When credentials are cached, they are stored unencrypted in the web browser's local storage. If the web browser tab is closed and then re-opened, the session is automatically re-established using the cached credentials. This local storage is also subject to the timeout configured in the setting `browser.credential_timeout`. In addition, caching credentials in browser local storage can be disabled altogether. To disable credentials caching, set `browser.retain_connection_credentials=false` in the server configuration.

If the user issues a `:server disconnect` command then any existing session is terminated and the credentials are cleared from local storage.

11.4. Security checklist

This section provides a summary of recommendations regarding security in Neo4j.

Below is a simple checklist highlighting the specific areas within Neo4j that may need some extra attention in order to ensure the appropriate level of security for your application.

1. Deploy Neo4j on safe servers in safe networks:
   a. Use subnets and firewalls.
   b. Only open up for the necessary ports. For a list of relevant ports see Ports.

      In particular, ensure that there is no external access to the port specified by the setting `dbms.backup.listen_address`. Failing to protect this port may leave a security hole open by which an unauthorized user can make a copy of the database onto a different machine.
2. Protect data-at-rest:
   a. Use volume encryption (e.g. Bitlocker).
   b. Manage access to database dumps (refer to Dump and load databases) and backups (refer to Perform a backup).
   c. Manage access to data files and transaction logs by ensuring the correct file permissions on the Neo4j files. Refer to File permissions for instructions on permission levels.
3. Protect data-in-transit:
   a. For remote access to the Neo4j database, only open up for encrypted Bolt or HTTPS.

b. Use SSL certificates issued from a trusted Certificate Authority.

   i. For configuring your Neo4j installation to use encrypted communication, refer to SSL framework.

   ii. If using Causal Clustering, configure and use encryption for intra-cluster communication. For details, see Intra-cluster encryption.

   iii. If using Causal Clustering, configure and use encryption for backups. This ensures that only servers with the specified SSL policy and SSL certificates will be able to access the server and perform the backup.

   iv. For configuring your Bolt and/or HTTPS connectors, refer to Configure connectors.

   v. If using LDAP, configure your LDAP system with encryption via StartTLS; see Use LDAP with encryption via StartTLS.

4. Be on top of the security for custom extensions:

   a. Validate any custom code that you deploy (procedures and unmanaged extensions) and ensure that they do not expose any parts of the product or data unintentionally.

   b. Survey the settings `dbms.security.procedures.unrestricted` and `dbms.security.procedures.whitelist` to ensure that they exclusively contain intentionally exposed extensions.

5. Ensure the correct file permissions on the Neo4j files.

6. Protect against the execution of unauthorized extensions by restricting access to the *bin*, *lib*, and *plugins* directories. Only the operating system user that Neo4j runs as should have permissions to those files. Refer to File permissions for instructions on permission levels.

7. If `LOAD CSV` is enabled, ensure that it does not allow unauthorized users to import data. How to configure `LOAD CSV` is described in Cypher Manual → `LOAD CSV`.

8. Use Neo4j authentication. The setting `dbms.security.auth_enabled` controls native authentication. The default value is `true`, which enables the native auth provider.

9. Survey your *neo4j.conf* file for ports relating to deprecated functions such as remote JMX (controlled by the parameter setting `dbms.jvm.additional=-Dcom.sun.management.jmxremote.port=3637`).

10. Review Browser credentials handling to determine whether the default credentials handling in Neo4j Browser complies with your security regulations. Follow the instructions to configure it if necessary.

11. Use the latest patch version of Neo4j.

Chapter 12. Monitoring

This chapter describes the tools that are available for monitoring Neo4j.

Neo4j provides mechanisms for continuous analysis through the output of metrics as well as the inspection and management of currently-executing queries.

Logs can be harvested for continuous analysis, or for specific investigations. Facilities are available for producing security event logs as well as query logs. The query management functionality is provided for specific investigations into query performance. Monitoring features are also provided for ad-hoc analysis of a Causal Cluster.

This chapter describes the following:

- Metrics
  - Expose metrics
  - Metrics reference
- Logging
  - Security events logging
  - Query logging
- Query management
  - List all running queries
  - List all active locks for a query
  - Terminate multiple queries
  - Terminate a single query
- Transaction management
  - Configure transaction timeout
  - Configure lock acquisition timeout
  - List all running transactions
- Connection management
  - List all network connections
  - Terminate multiple network connections
  - Terminate a single network connection
- Monitoring a Causal Cluster
  - Procedures for monitoring a Causal Cluster
  - Endpoints for status information
- Monitoring the state of individual databases

12.1. Metrics

This section describes the Neo4j metrics output facilities.

This section describes the following:

- Types of metrics

    - Global metrics
    - Database metrics
- Expose metrics
    - Enable metrics logging
    - Graphite
    - Prometheus
    - CSV files
    - JMX Beans
- Metrics reference
    - General-purpose metrics
    - Metrics specific to Causal Clustering
    - Java Virtual Machine metrics

12.1.1. Types of metrics

This section describes the types of metrics available in Neo4j.

Neo4j provides a built-in metrics subsystem. Reported metrics can be queried via JMX, retrieved from CSV files, or consumed by third-party monitoring tools.

Neo4j has the following types of metrics:

- Global
- Per database - covers individual database

Global metrics

Global metrics cover the whole database management system, and represents the status of the system as a whole.

Global metrics have following name format: `<user-configured-prefix>.metric.name`

Metrics of this type are reported as soon as the database management system is available. For example, all JVM related metrics are global. In particular, the `neo4j.vm.thread.count` metric has a default user-configured-prefix `neo4j` and the metric name is `vm.thread.count`.

By default, global metrics include:

- Page cache metrics
- GC metrics
- Thread metrics
- Memory pool metrics
- Memory buffers metrics
- File descriptor metrics
- Bolt metrics
- Web Server metrics

Database metrics

Each database metric is reported for a particular database only. Database metrics are only available during the lifetime of the database. When a database becomes unavailable, all of its metrics become unavailable also.

Database metrics have following name format: `<user-configured-prefix>.<database-name>.metric.name`

For example, any transaction metric is a database metric. In particular, the `neo4j.mydb.transaction.started` metric has a default user-configured-prefix `neo4j` and it is a metric for the `mydb` database.

By default, database metrics include:

- Transaction metrics
- Checkpoint metrics
- Log rotation metrics
- Database data metrics
- Cypher metrics
- Causal clustering metrics

12.1.2. Expose metrics

This section describes how to log and display various metrics by using the Neo4j metrics output facilities.

This section describes the following:

- Enable metrics logging
- Graphite
- Prometheus
- CSV files
- JMX Beans

Enable metrics logging

By default, metrics logging into CSV files and exposure via JMX beans are enabled. All metrics are enabled once `metrics.enabled=true` is set and you can use the individual settings to disable specific metrics.

```
# Setting for enabling all supported metrics.
metrics.enabled=true

# Setting for exposing metrics about transactions; number of transactions started, committed, etc.
metrics.neo4j.tx.enabled=false

# Setting for exposing metrics about the Neo4j page cache; page faults, evictions, flushes and exceptions,
etc.
metrics.neo4j.pagecache.enabled=false

# Setting for exposing metrics about approximately entities are in the database; nodes, relationships,
properties, etc.
metrics.neo4j.counts.enabled=false
```

Graphite

Send metrics to Graphite or any monitoring tool based on the Graphite protocol.

Add the following settings to *neo4j.conf* in order to enable integration with Graphite:

```
# Enable the Graphite integration. Default is 'false'.
metrics.graphite.enabled=true
# The IP and port of the Graphite server on the format <hostname or IP address>:<port number>.
# The default port number for Graphite is 2003.
metrics.graphite.server=localhost:2003
# How often to send data. Default is 3 seconds.
metrics.graphite.interval=3s
# Prefix for Neo4j metrics on Graphite server.
metrics.prefix=Neo4j_1
```

Start Neo4j and connect to Graphite via a web browser in order to monitor your Neo4j metrics.

If you configure the Graphite server to be a hostname or DNS entry you should be aware that the JVM resolves hostnames to IP addresses and by default caches the result indefinitely for security reasons. This is controlled by the value of `networkaddress.cache.ttl` in the JVM Security properties. See https://docs.oracle.com/javase/8/docs/technotes/guides/net/properties.html for more information.

Prometheus

Publish metrics for polling as Prometheus endpoint.

Add the following settings to *neo4j.conf* in order to enable the Prometheus endpoint.

```
# Enable the Prometheus endpoint. Default is 'false'.
metrics.prometheus.enabled=true
# The IP and port the endpoint will bind to in the format <hostname or IP address>:<port number>.
# The default is localhost:2004.
metrics.prometheus.endpoint=localhost:2004
```

When Neo4j is fully started a Prometheus endpoint will be available at the configured address.

CSV files

Export metrics to CSV files.

Add the following settings to *neo4j.conf* in order to enable export of metrics into local .CSV files:

```
# Enable the CSV exporter. Default is 'true'.
metrics.csv.enabled=true
# Directory path for output files.
# Default is a "metrics" directory under NEO4J_HOME.
#dbms.directories.metrics='/local/file/system/path'
# How often to store data. Default is 3 seconds.
metrics.csv.interval=3s
# The maximum number of CSV files that will be saved. Default is 7.
metrics.csv.rotation.keep_number
# The file size at which the csv files will auto-rotate. Default is 10M.
metrics.csv.rotation.size=10M
```

JMX Beans

Expose metrics over JMX beans.

In order to enable metrics exposure via JMX, add the following setting to *neo4j.conf*:

```
# Enable settings export via JMX. Default is 'true'.
metrics.jmx.enabled=true
```

12.1.3. Metrics reference

This section provides a listing of available metrics.

This section describes the following:

- General-purpose metrics
- Metrics specific to Causal Clustering
- Java Virtual Machine metrics

General-purpose metrics

Table 24. Database checkpointing metrics

Name	Description
`<prefix>.check_point.events`	The total number of check point events executed so far.
`<prefix>.check_point.total_time`	The total time spent in check pointing so far.
`<prefix>.check_point.duration`	The duration of the last check point event.

Table 25. Database data metrics

Name	Description
`<prefix>.ids_in_use.relationship_type`	The total number of different relationship types stored in the database.
`<prefix>.ids_in_use.property`	The total number of different property names used in the database.
`<prefix>.ids_in_use.relationship`	The total number of relationships stored in the database.
`<prefix>.ids_in_use.node`	The total number of nodes stored in the database.

Table 26. Database page cache metrics

Name	Description
`<prefix>.page_cache.eviction_exceptions`	The total number of exceptions seen during the eviction process in the page cache.
`<prefix>.page_cache.flushes`	The total number of flushes executed by the page cache.
`<prefix>.page_cache.unpins`	The total number of page unpins executed by the page cache.
`<prefix>.page_cache.pins`	The total number of page pins executed by the page cache.
`<prefix>.page_cache.evictions`	The total number of page evictions executed by the page cache.
`<prefix>.page_cache.page_faults`	The total number of page faults happened in the page cache.

Name	Description
`<prefix>.page_cache.hits`	The total number of page hits happened in the page cache.
`<prefix>.page_cache.hit_ratio`	The ratio of hits to the total number of lookups in the page cache.
`<prefix>.page_cache.usage_ratio`	The ratio of number of used pages to total number of available pages.

Table 27. Database transaction metrics

Name	Description
`<prefix>.transaction.started`	The total number of started transactions.
`<prefix>.transaction.peak_concurrent`	The highest peak of concurrent transactions.
`<prefix>.transaction.active`	The number of currently active transactions.
`<prefix>.transaction.active_read`	The number of currently active read transactions.
`<prefix>.transaction.active_write`	The number of currently active write transactions.
`<prefix>.transaction.committed`	The total number of committed transactions.
`<prefix>.transaction.committed_read`	The total number of committed read transactions.
`<prefix>.transaction.committed_write`	The total number of committed write transactions.
`<prefix>.transaction.rollbacks`	The total number of rolled back transactions.
`<prefix>.transaction.rollbacks_read`	The total number of rolled back read transactions.
`<prefix>.transaction.rollbacks_write`	The total number of rolled back write transactions.
`<prefix>.transaction.terminated`	The total number of terminated transactions.
`<prefix>.transaction.terminated_read`	The total number of terminated read transactions.
`<prefix>.transaction.terminated_write`	The total number of terminated write transactions.
`<prefix>.transaction.last_committed_tx_id`	The ID of the last committed transaction.
`<prefix>.transaction.last_closed_tx_id`	The ID of the last closed transaction.

Table 28. Cypher metrics

Name	Description
`<prefix>.cypher.replan_events`	The total number of times Cypher has decided to re-plan a query.
`<prefix>.cypher.replan_wait_time`	The total number of seconds waited between query replans.

Table 29. Database transaction log metrics

Name	Description
`<prefix>.log.rotation_events`	The total number of transaction log rotations executed so far.
`<prefix>.log.rotation_total_time`	The total time spent in rotating transaction logs so far.
`<prefix>.log.rotation_duration`	The duration of the last log rotation event.
`<prefix>.log.appended_bytes`	The total number of bytes appended to transaction log.

Table 30. Bolt metrics

Name	Description
`<prefix>.bolt.sessions_started`	The total number of Bolt sessions started since this instance started. This includes both succeeded and failed sessions (deprecated, use connections_opened instead).
`<prefix>.bolt.connections_opened`	The total number of Bolt connections opened since this instance started. This includes both succeeded and failed connections.
`<prefix>.bolt.connections_closed`	The total number of Bolt connections closed since this instance started. This includes both properly and abnormally ended connections.
`<prefix>.bolt.connections_running`	The total number of Bolt connections currently being executed.
`<prefix>.bolt.connections_idle`	The total number of Bolt connections sitting idle.
`<prefix>.bolt.messages_received`	The total number of messages received via Bolt since this instance started.
`<prefix>.bolt.messages_started`	The total number of messages that began processing since this instance started. This is different from messages received in that this counter tracks how many of the received messages havebeen taken on by a worker thread.
`<prefix>.bolt.messages_done`	The total number of messages that completed processing since this instance started. This includes successful, failed and ignored Bolt messages.
`<prefix>.bolt.messages_failed`	The total number of messages that failed processing since this instance started.
`<prefix>.bolt.accumulated_queue_time`	The accumulated time messages have spent waiting for a worker thread.
`<prefix>.bolt.accumulated_processing_time`	The accumulated time worker threads have spent processing messages.

Table 31. Server metrics

Name	Description
`<prefix>.server.threads.jetty.idle`	The total number of idle threads in the jetty pool.
`<prefix>.server.threads.jetty.all`	The total number of threads (both idle and busy) in the jetty pool.

Metrics specific to Causal Clustering

Table 32. Core metrics

Name	Description
`<prefix>.causal_clustering.core.append_index`	Append index of the RAFT log.
`<prefix>.causal_clustering.core.commit_index`	Commit index of the RAFT log.

Name	Description
`<prefix>.causal_clustering.core.term`	RAFT Term of this server.
`<prefix>.causal_clustering.core.tx_retries`	Transaction retries.
`<prefix>.causal_clustering.core.is_leader`	Is this server the leader?
`<prefix>.causal_clustering.core.in_flight_cache.total_bytes`	In-flight cache total bytes.
`<prefix>.causal_clustering.core.in_flight_cache.max_bytes`	In-flight cache max bytes.
`<prefix>.causal_clustering.core.in_flight_cache.element_count`	In-flight cache element count.
`<prefix>.causal_clustering.core.in_flight_cache.max_elements`	In-flight cache maximum elements.
`<prefix>.causal_clustering.core.in_flight_cache.hits`	In-flight cache hits.
`<prefix>.causal_clustering.core.in_flight_cache.misses`	In-flight cache misses.
`<prefix>.causal_clustering.core.message_processing_delay`	Delay between RAFT message receive and process.
`<prefix>.causal_clustering.core.message_processing_timer`	Timer for RAFT message processing.
`<prefix>.causal_clustering.core.discovery.replicated_data`	Size of replicated data structures.
`<prefix>.causal_clustering.core.replication_new`	Raft replication new request count.
`<prefix>.causal_clustering.core.replication_attempt`	Raft replication attempt count.
`<prefix>.causal_clustering.core.replication_fail`	Raft Replication fail count.
`<prefix>.causal_clustering.core.replication_maybe`	Raft Replication maybe count.
`<prefix>.causal_clustering.core.replication_success`	Raft Replication success count.
`<prefix>.causal_clustering.core.discovery.cluster.members`	Discovery cluster member size.
`<prefix>.causal_clustering.core.discovery.cluster.unreachable`	Discovery cluster unreachable size.
`<prefix>.causal_clustering.core.discovery.cluster.converged`	Discovery cluster convergence.

Table 33. Read Replica Metrics

Name	Description
`<prefix>.causal_clustering.read_replica.pull_updates`	The total number of pull requests made by this instance.
`<prefix>.causal_clustering.read_replica.pull_update_highest_tx_id_requested`	The highest transaction id requested in a pull update by this instance.
`<prefix>.causal_clustering.read_replica.pull_update_highest_tx_id_received`	The highest transaction id that has been pulled in the last pull updates by this instance.

Java Virtual Machine Metrics

These metrics are environment dependent and they may vary on different hardware and with JVM configurations. Typically these metrics will show information about garbage collections (for example the number of events and time spent collecting), memory pools and buffers, and finally the number of active threads running.

Table 34. GC metrics.

Name	Description
`<prefix>.vm.gc.time.%s`	Accumulated garbage collection time in milliseconds.
`<prefix>.vm.gc.count.%s`	Total number of garbage collections.

Table 35. JVM memory buffers metrics.

Name	Description
`<prefix>.vm.memory.buffer.%s.count`	Estimated number of buffers in the pool.
`<prefix>.vm.memory.buffer.%s.used`	Estimated amount of memory used by the pool.
`<prefix>.vm.memory.buffer.%s.capacity`	Estimated total capacity of buffers in the pool.

Table 36. JVM memory pools metrics.

Name	Description
`<prefix>.vm.memory.pool.%s`	Estimated number of buffers in the pool.

Table 37. JVM threads metrics.

Name	Description
`<prefix>.vm.thread.count`	Estimated number of active threads in the current thread group.
`<prefix>.vm.thread.total`	The total number of live threads including daemon and non-daemon threads.

12.2. Logging

This section describes security and query logging in Neo4j.

Neo4j provides two types of logs for inspection of queries that are run in the database, and of security events that have occurred.

The section describes the following:

- Query logging
- Security events logging

12.2.1. Query logging

This section describes Neo4j support for query logging.

Neo4j can be configured to log queries executed in the database.

Query logging is enabled by default and is controlled by the setting `dbms.logs.query.enabled`.

Configuration options are:

Option	Default	Description
`off`		Will completely disable logging.
`info`		Will log at the end of queries that have either succeeded or failed. The `dbms.logs.query.threshold` parameter is used to determine the threshold for logging a query. If the execution of a query takes a longer time than this threshold, it will be logged. Setting the threshold to `0` will result in all queries being logged.
`verbose`	Yes	Will log all queries at both start and finish, regardless of `dbms.logs.query.threshold`.

Log configuration

The name of the log file is `query.log` and it resides in the *logs* directory (see File locations).

Rotation of the query log can be configured in the *neo4j.conf* configuration file. The following parameters are available:

Parameter name	Default value	Description
`dbms.logs.query.allocation_logging_enabled`	`false`	Log allocated bytes for the executed queries being logged.
`dbms.logs.query.enabled`	`verbose`	Log executed queries.
`dbms.logs.query.page_logging_enabled`	`false`	Log page hits and page faults for the executed queries being logged.
`dbms.logs.query.parameter_logging_enabled`	`true`	Log parameters for executed queries that take longer than the configured threshold.
`dbms.logs.query.rotation.keep_number`	`7`	Sets number of historical log files kept.
`dbms.logs.query.rotation.size`	`20M`	Sets the file size at which the query log will auto-rotate.
`dbms.logs.query.threshold`	`0`	If the execution of query takes a longer time than this threshold, the query is logged (provided query logging is set to `info`).

Parameter name	Default value	Description
dbms.logs.query.time_logging_enabled	false	Log detailed time information for the executed queries being logged.

Example 58. Configure for simple query logging

In this example we set query logging to `info`, but leave all other query log parameters at their defaults.

```
dbms.logs.query.enabled=info
```

Below is an example of the query log with this basic configuration:

```
2017-11-22 14:31 ... INFO  9 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:31 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  3 ms: server-session http    127.0.0.1   /db/data/cypher neo4j - CALL
dbms.procedures() - {}
2017-11-22 14:32 ... INFO  1 ms: server-session http    127.0.0.1   /db/data/cypher neo4j - CALL
dbms.showCurrentUs...
2017-11-22 14:32 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  0 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59167  ...
2017-11-22 14:32 ... INFO  2 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
client/127.0.0.1:59261  ...
```

Example 59. Configure for query logging with more details

In this example we turn query logging on, and also enable some additional logging.

```
dbms.logs.query.parameter_logging_enabled=true
dbms.logs.query.time_logging_enabled=true
dbms.logs.query.allocation_logging_enabled=true
dbms.logs.query.page_logging_enabled=true
```

Below is an example of the query log with these configuration parameters enabled:

```
2017-11-22 12:38 ... INFO  3 ms: bolt-session   bolt    johndoe neo4j-javascript/1.4.1
...
2017-11-22 22:38 ... INFO  61 ms: (planning: 0, cpu: 58, waiting: 0) - 6164496 B - 0 page hits, 1
page faults  ...
2017-11-22 12:38 ... INFO  78 ms: (planning: 40, cpu: 74, waiting: 0) - 6347592 B - 0 page hits, 0
page faults ...
2017-11-22 12:38 ... INFO  44 ms: (planning: 9, cpu: 25, waiting: 0) - 1311384 B - 0 page hits, 0
page faults  ...
2017-11-22 12:38 ... INFO  6 ms: (planning: 2, cpu: 6, waiting: 0) - 420872 B - 0 page hits, 0 page
faults -   ...
```

Attach metadata to a query

It is possible to attach metadata to a query and have it printed in the query log, using the built-in procedure `tx.setMetaData`. This is typically done programmatically, but can be illustrated as follows, using `cypher-shell`.

Example 60. Attach metadata to a query

Start a transaction and call `tx.setMetaData` with a list of meta data.

```
neo4j> :begin
neo4j# CALL tx.setMetaData({ User: 'jsmith', AppServer: 'app03.dc01.company.com'});
neo4j# CALL dbms.procedures() YIELD name RETURN COUNT(name);
COUNT(name)
39
neo4j# :commit
```

Below are the corresponding results in the query log:

```
... CALL tx.setMetaData({ User: 'jsmith', AppServer: 'app03.dc01.company.com'}); - {} - {}
... CALL dbms.procedures() YIELD name RETURN COUNT(name); - {} - {User: 'jsmith', AppServer:
'app03.dc01.company.com'}
```

12.2.2. Security events logging

This section describes Neo4j support for security events logging.

Neo4j provides security event logging that records all security events.

For native user management, the following actions are recorded:

- Login attempts - per default both successful and unsuccessful logins are recorded.
- All administration commands run towards the system database.
- All security procedures run towards the system database.

Log configuration

The name of the log file is `security.log` and it resides in the *logs* directory (see File locations).

Rotation of the security events log can be configured in the *neo4j.conf* configuration file. The following parameters are available:

Parameter name	Default value	Description
`dbms.logs.security.rotation.size`	20M	Sets the file size at which the security event log will auto-rotate.
`dbms.logs.security.rotation.delay`	300s	Sets the minimum time interval after the last log rotation occurred, before the log may be rotated again.
`dbms.logs.security.rotation.keep_number`	7	Sets number of historical log files kept.

If using LDAP as the authentication method, some cases of LDAP misconfiguration will also be logged, as well as LDAP server communication events and failures.

If many programmatic interactions are expected, it is advised to disable the logging of successful logins. Logging of successful logins is disabled by setting the `dbms.security.log_successful_authentication` parameter in the `neo4j.conf` file:

```
dbms.security.log_successful_authentication=false
```

Below is an example of the security log:

```
2019-12-09 13:45:00.796+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: logged in
2019-12-09 13:47:53.443+0000 ERROR [AsyncLog @ 2019-12-09 ...]  [johndoe]: failed to log in: invalid
principal or credentials
2019-12-09 13:48:28.566+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: CREATE USER janedoe SET
PASSWORD '******' CHANGE REQUIRED
2019-12-09 13:48:32.753+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: CREATE ROLE custom
2019-12-09 13:49:11.880+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: GRANT ROLE custom TO janedoe
2019-12-09 13:49:34.979+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: GRANT TRAVERSE ON GRAPH *
NODES A, B (*) TO custom
2019-12-09 13:49:37.053+0000 INFO  [AsyncLog @ 2019-12-09 ...]  [johnsmith]: DROP USER janedoe
```

12.3. Query management

This section describes facilities for query management.

This section describes the following:

- List all running queries
- List all active locks for a query
- Terminate multiple queries
- Terminate a single query

12.3.1. List all running queries

An administrator is able to view all queries that are currently executing within the instance. Alternatively, the current user may view all of their own currently-executing queries.

Syntax:

`CALL dbms.listQueries()`

Returns:

Name	Type	Description
`queryId`	String	This is the ID of the query.
`username`	String	This is the username of the user who is executing the query.
`metaData`	Map	This is any metadata associated with the transaction.
`query`	String	This is the query itself.
`parameters`	Map	This is a map containing all the parameters used by the query.
`planner`	String	Planner used by the query
`runtime`	String	Runtime used by the query
`indexes`	List	Indexes used by the query
`startTime`	String	This is the time at which the query was started.
`elapsedTime`	String	Deprecated: Use `elapsedTimeMillis` instead. This is the time that has elapsed since the query was started.

Name	Type	Description
`connectionDetails`	String	Deprecated: Use `requestScheme`, `clientAddress`,`requestUri` These are the connection details pertaining to the query.
`protocol`	String	The protocol used by connection issuing the query.
`connectionId`	String	The ID of the connection issuing the query. This field will be null if the query was issued using embedded API.
`clientAddress`	String	The client address of the connection issuing the query.
`requestUri`	String	The request URI used by the client connection issuing the query.
`status`	String	Status of the executing query.
`resourceInformation`	Map	Status of the executing query.
`activeLockCount`	Integer	Count of active locks held by transaction executing the query.
`elapsedTimeMillis`	Integer	This is the time in milliseconds that has elapsed since the query was started.
`cpuTimeMillis`	Integer	CPU time in milliseconds that has been actively spent executing the query. This field will be null unless the config parameter `dbms.track_query_cpu_time` is set to `true`.
`waitTimeMillis`	Integer	Wait time in milliseconds that has been spent waiting to acquire locks.
`idleTimeMillis`	Integer	Idle time in milliseconds. This field will be null unless the config parameter `dbms.track_query_cpu_time` is set to `true`.
`allocatedBytes`	Integer	Bytes allocated for the executing query. This number is cumulative over the duration of the query. For memory-intense or long-running queries the value may be larger than the current memory allocation. This field will be null unless the config parameter `dbms.track_query_allocation` is set to `true`.
`pageHits`	Integer	Page hits occurred during the execution.
`pageFaults`	Integer	Page faults occurred during the execution.
`database`	String	This is the name of the database the query is executing against.

Example 61. Viewing queries that are currently executing

The following example shows that the user `alwood` is currently running `dbms.listQueries()` yielding specific variables, namely `queryId`, `username`, `query`, `elapsedTimeMillis`, `requestUri`, and `status`. It is executed against the database `myDb`, which is indicated by the prefix in the `queryId`.

```
CALL dbms.listQueries() YIELD queryId, username, query, elapsedTimeMillis, requestUri, status
```

```
 "queryId"       "username"  "query"                        "elapsedTimeMillis"  "requestUri"
 "status"


 "myDb-query-33" "alwood"    "CALL dbms.listQueries() YIELD "1"
 "127.0.0.1:7687" {"state":"RUNNING"}
                              queryId, username, query, ela

                             psedTime, requestUri, status"



1 row
```

12.3.2. List all active locks for a query

An administrator is able to view all active locks held by the transaction executing the query with the `queryId`.

Syntax:

`CALL dbms.listActiveLocks(queryId)`

Returns:

Name	Type	Description
`mode`	String	Lock mode corresponding to the transaction.
`resourceType`	String	Resource type of the locked resource
`resourceId`	Integer	Resource id of the locked resource .

Example 62. Viewing active locks for a query

The following example shows the active locks held by transaction executing query with id `myDb-query-614`

```
CALL dbms.listActiveLocks( "myDb-query-614" )
```

```
 "mode"   "resourceType" "resourceId"

 "SHARED" "SCHEMA"       "0"

1 row
```

The following example shows the active locks for all currently executing queries by yielding the `queryId` from `dbms.listQueries` procedure

```
CALL dbms.listQueries() YIELD queryId, query
CALL dbms.listActiveLocks( queryId ) YIELD resourceType, resourceId, mode
RETURN queryId, query, resourceType, resourceId, mode
```

```
 "queryId"               "query"                        "resourceType" "resourceId" "mode"

 "myDb-query-614"        "match (n), (m), (o), (p), (q) "SCHEMA"       "0"          "SHARED"
                          return count(*)"

 "myOtherDb-query-684"   "CALL dbms.listQueries() YIELD "SCHEMA"       "0"          "SHARED"
                          .."

2 rows
```

12.3.3. Terminate multiple queries

An administrator is able to terminate within the instance all transactions executing a query with any of the given query IDs. Alternatively, the current user may terminate all of their own transactions executing a query with any of the given query IDs.

Syntax:

`CALL dbms.killQueries(queryIds)`

Arguments:

Name	Type	Description
`ids`	List<String>	This is a list of the IDs of all the queries to be terminated.

Returns:

Name	Type	Description
`queryId`	String	This is the ID of the terminated query.
`username`	String	This is the username of the user who was executing the (now terminated) query.

Example 63. Terminating multiple queries

The following example shows that the administrator has terminated the queries with IDs `joeDb-query-378` and `anneDb-query-765`, started by the users `joesmith` and `annebrown`, respectively.

This command can target queries from multiple databases at the same time, as noted by the prefixes `joeDb-` and `anneDb-`:

```
CALL dbms.killQueries(['joeDb-query-378','anneDb-query-765'])
```

```
+-----------------------------------+
| queryId            | username     |
+-----------------------------------+
| "joeDb-query-378"  | "joesmith"   |
| "anneDb-query-765" | "annebrown"  |
+-----------------------------------+
2 rows
```

12.3.4. Terminate a single query

An administrator is able to terminate within the instance any transaction executing the query whose ID is provided. Alternatively, the current user may terminate their own transaction executing the query whose ID is provided.

Syntax:

`CALL dbms.killQuery(queryId)`

Arguments:

Name	Type	Description
`id`	String	This is the ID of the query to be terminated.

Returns:

Name	Type	Description
`queryId`	String	This is the ID of the terminated query.
`username`	String	This is the username of the user who was executing the (now terminated) query.
`message`	String	A message stating whether the query was successfully found.

Example 64. Terminating a single query

The following example shows that the user `joesmith` has terminated his query with the ID `joeDb-query-502`.

```
CALL dbms.killQuery('joeDb-query-502')
```

```
+-------------------------------------------------+
| queryId           | username    | message     |
+-------------------------------------------------+
| "joeDb-query-502" | "joesmith"  | Query found |
+-------------------------------------------------+
1 row
```

The following example shows the output when trying to kill a query with an ID that does not exist.

```
CALL dbms.killQuery('myDb-query-502')
```

```
+-----------------------------------------------------------------+
| queryId          | username    | message                        |
+-----------------------------------------------------------------+
| "myDb-query-502" | "n/a"       | No Query found with this id |
+-----------------------------------------------------------------+
1 row
```

12.4. Transaction management

This section describes facilities for transaction management.

This section describes the following:

- Configure transaction timeout
- Configure lock acquisition timeout
- List all running transactions

12.4.1. Configure transaction timeout

It is possible to configure Neo4j to terminate transactions whose execution time has exceeded the configured timeout. To enable this feature, set `dbms.transaction.timeout` to some positive time interval value denoting the default transaction timeout. Setting `dbms.transaction.timeout` to `0` — which is the default value — disables the feature.

Example 65. Configure transaction timeout

Set the timeout to ten seconds.

```
dbms.transaction.timeout=10s
```

Configuring transaction timeout will have no effect on transactions executed with custom timeouts (via the Java API), as a custom timeout will override the value set for `dbms.transaction.timeout`.

The *transaction timeout* feature is also known as the *transaction guard*.

12.4.2. Configure lock acquisition timeout

An executing transaction may get stuck while waiting for some lock to be released by another transaction. A transaction in such state is not desirable, and in some cases it is better for the transaction to instead give up and fail.

To enable this feature, set `dbms.lock.acquisition.timeout` to some positive time interval value denoting the maximum time interval within which any particular lock should be acquired, before failing the transaction. Setting `dbms.lock.acquisition.timeout` to `0` — which is the default value — disables the lock acquisition timeout.

Example 66. Configure lock acquisition timeout

Set the timeout to ten seconds.

```
dbms.lock.acquisition.timeout=10s
```

12.4.3. List all running transactions

An administrator is able to view all transactions that are currently executing within the instance. Alternatively, the current user may view all of their own currently-executing transactions.

Syntax:

`CALL dbms.listTransactions()`

Returns:

Name	Type	Description
`transactionId`	String	This is the ID of the transaction.
`username`	String	This is the username of the user who is executing the transaction.
`metaData`	Map	This is any metadata associated with the transaction.
`startTime`	String	This is the time at which the transaction was started.
`protocol`	String	The protocol used by connection issuing the transaction.
`connectionId`	String	The ID of the connection issuing the transaction. This field will be null if the transaction was issued using embedded API.
`clientAddress`	String	The client address of the connection issuing the transaction.
`requestUri`	String	The request URI used by the client connection issuing the transaction.
`currentQueryId`	String	This is the ID of the current query executed by transaction.
`currentQuery`	String	This is the current query executed by transaction.

Name	Type	Description
`activeLockCount`	Integer	Count of active locks held by transaction.
`status`	String	Status of the executing transaction.
`resourceInformation`	Map	Information about what transaction is waiting for when it is blocked.
`elapsedTimeMillis`	Integer	This is the time in milliseconds that has elapsed since the transaction was started.
`cpuTimeMillis`	Integer	CPU time in milliseconds that has been actively spent executing the transaction.
`waitTimeMillis`	Integer	Wait time in milliseconds that has been spent waiting to acquire locks.
`idleTimeMillis`	Integer	Idle time in milliseconds.
`allocatedBytes`	Integer	Bytes allocated for the executing transaction. This number is cumulative over the duration of the transaction. For memory-intense or long-running transactions the value may be larger than the current memory allocation.
`allocatedDirectBytes`	Integer	Direct bytes used by the executing transaction.
`pageHits`	Integer	Page hits occurred during the execution.
`pageFaults`	Integer	Page faults occurred during the execution.
`database`	String	This is the name of the database the transaction is executing against.

Example 67. Viewing transactions that are currently executing

The following example shows that the user '**alwood**' is currently running `dbms.listTransactions()`. The procedure call yields specific information about the running transaction, namely `transactionId`, `username`, `currentQuery`, `elapsedTimeMillis`, `requestUri`, and `status`.

```
CALL dbms.listTransactions() YIELD transactionId, username, currentQuery, elapsedTimeMillis,
requestUri, status
```

```


 "transactionId"       "username"  "currentQuery"                           "elapsedTimeMillis"
 "requestUri"     "status"


 "myDb-transaction-22" "alwood"    "CALL dbms.listTransactions() YIELD    "1"
 "127.0.0.1:7687" "Running"
                                    transactionId, username, currentQuery

                                   elapsedTime, requestUri, status"



1 row
```

12.5. Connection management

This section describes facilities for connection management.

This section describes the following:

- List all network connections
- Terminate multiple network connections
- Terminate a single network connection

12.5.1. List all network connections

An administrator is able to view all network connections within the database instance. Alternatively, the current user may view all of their own network connections.

The procedure `dbms.listConnections` lists all accepted network connections for all configured connectors, including Bolt, HTTP, and HTTPS. Some listed connections might never perform authentication. For example, HTTP GET requests to the Neo4j Browser endpoint fetches static resources and does not need to authenticate. However, connections made using Neo4j Browser require the user to provide credentials and perform authentication.

Syntax:

```
CALL dbms.listConnections()
```

Returns:

Name	Type	Description
`connectionId`	String	This is the ID of the network connection.
`connectTime`	String	This is the time at which the connection was started.
`connector`	String	Name of the connector that accepted the connection.
`username`	String	This is the username of the user who initiated the connection. This field will be null if the transaction was issued using embedded API. It can also be null if connection did not perform authentication.
`userAgent`	String	Name of the software that is connected. This information is extracted from the `User-Agent` request header for HTTP and HTTPS connections. It is available natively for Bolt connections which supply the agent string in an initialization message.
`serverAddress`	String	The server address this connection is connected to.
`clientAddress`	String	The client address of the connection.

Example 68. List all network connections

The following example shows that the user '**alwood**' is connected using Java driver and a Firefox web browser. The procedure call yields specific information about the connection, namely `connectionId`, `connectTime`, `connector`, `username`, `userAgent`, and `clientAddress`.

```
CALL dbms.listConnections() YIELD connectionId, connectTime, connector, username, userAgent,
clientAddress
```

```
 "connectionId" "connectTime"              "connector" "username" "userAgent"
 "clientAddress"   "status"


 "bolt-21"      "2018-10-10T12:11:42.276Z" "bolt"      "alwood"   "neo4j-java/1.6.3"
 "127.0.0.1:53929" "Running"


 "http-11"      "2018-10-10T12:37:19.014Z" "http"      null       "Mozilla/5.0 (Macintosh; Intel
macOS 10.13; rv:62.0) Gecko/20100101 Firefox/62.0" "127.0.0.1:54118" "Running"


2 rows
```

12.5.2. Terminate multiple network connections

An administrator is able to terminate within the instance all network connections with any of the given IDs. Alternatively, the current user may terminate all of their own network connections with any of the given IDs.

Syntax:

`CALL dbms.killConnections(connectionIds)`

Arguments:

Name	Type	Description
`ids`	List<String>	This is a list of the IDs of all the connections to be terminated.

Returns:

Name	Type	Description
`connectionId`	String	This is the ID of the terminated connection.
`username`	String	This is the username of the user who initiated the (now terminated) connection.
`message`	String	A message stating whether the connection was successfully found.

Considerations:

Bolt connections are stateful. Termination of a Bolt connection results in termination of the ongoing query/transaction.
Termination of an HTTP/HTTPS connection can terminate the ongoing HTTP/HTTPS request.

Example 69. Terminate multiple network connections

The following example shows that the administrator has terminated the connections with IDs '**bolt-37**' and '**https-11**', started by the users '**joesmith**' and '**annebrown**', respectively. The administrator also attempted to terminate the connection with ID '**http-42**' which did not exist.

```
CALL dbms.killConnections(['bolt-37', 'https-11', 'http-42'])
```

```
 "connectionId" "username"  "message"
 "bolt-37"      "joesmith"  "Connection found"
 "https-11"     "annebrown" "Connection found"
 "http-42"      "n/a"       "No connection found with this id"
3 rows
```

12.5.3. Terminate a single network connection

An administrator is able to terminate within the instance any network connection with the given ID. Alternatively, the current user may terminate their own network connection with the given ID.

Syntax:

`CALL dbms.killConnection(connectionId)`

Arguments:

Name	Type	Description
`id`	String	This is the ID of the connection to be terminated.

Returns:

Name	Type	Description
`connectionId`	String	This is the ID of the terminated connection.
`username`	String	This is the username of the user who initiated the (now terminated) connection.
`message`	String	A message stating whether the connection was successfully found.

Considerations:

Bolt connections are stateful. Termination of a Bolt connection results in termination of the ongoing query/transaction.
Termination of an HTTP/HTTPS connection can terminate the ongoing HTTP/HTTPS request.

Example 70. Terminate a single network connection

The following example shows that the user '**joesmith**' has terminated his connection with the ID '**bolt-4321**'.

```
CALL dbms.killConnection('bolt-4321')
```

```
 "connectionId" "username"  "message"
 "bolt-4321"    "joesmith"  "Connection found"
1 row
```

The following example shows the output when trying to kill a connection with an ID that does not exist.

```
CALL dbms.killConnection('bolt-987')
```

```
 "connectionId" "username"  "message"
 "bolt-987"     "n/a"       "No connection found with this id"
1 row
```

12.6. Monitoring a Causal Cluster

This section covers additional facilities available for monitoring a Neo4j Causal Cluster.

In addition to specific metrics as described in previous sections, Neo4j Causal Clusters provide an infrastructure that operators will wish to monitor. The procedures can be used to inspect the cluster state and to understand its current condition and topology. Additionally, there are HTTP endpoints for checking health and status.

This section describes the following:

- Procedures for monitoring a Causal Cluster
  - Find out the role of a cluster member
  - Gain an overview over the instances in the cluster
  - Get routing recommendations
- Endpoints for status information
  - Adjusting security settings for Causal Clustering endpoints
  - Unified endpoints

12.6.1. Procedures for monitoring a Causal Cluster

This section covers procedures for monitoring a Neo4j Causal Cluster.

A number of procedures are available that provide information about a cluster. This section describes the following:

- Find out the role of a cluster member
- Gain an overview over the instances in the cluster
- Get routing recommendations

Find out the role of a cluster member

The procedure `dbms.cluster.role(databaseName)` can be called on every instance in a Causal Cluster to return the role of the instance. Each instance holds multiple databases and participates in multiple independent Raft groups. The role returned by the procedure is for the database denoted by the `databaseName` parameter.

Syntax:

```
CALL dbms.cluster.role(databaseName)
```

Arguments:

Name	Type	Description
`databaseName`	String	The name of the database to get the cluster role for.

Returns:

Name	Type	Description
`role`	String	This is the role of the current instance, which can be `LEADER`, `FOLLOWER`, or `READ_REPLICA`.

Considerations:

- While this procedure is useful in and of itself, it serves as basis for more powerful monitoring procedures.

Example 71. Check the role of this instance

The following example shows how to find out the role of the current instance for database `neo4j`, which in this case is `FOLLOWER`.

```
CALL dbms.cluster.role("neo4j")
```

role
FOLLOWER

Gain an overview over the instances in the cluster

The procedure `dbms.cluster.overview()` provides an overview of cluster topology by returning details on all the instances in the cluster.

Syntax:

```
CALL dbms.cluster.overview()
```

Returns:

Name	Type	Description
`id`	String	This is id of the instance.
`addresses`	List	This is a list of all the addresses for the instance.
`groups`	List	This is a list of all the server groups which an instance is part of.
`databases`	Map	This is a map of all databases with corresponding roles which the instance is hosting. The keys in the map are database names. The values are roles of this instance in the corresponding Raft groups, which can be `LEADER`, `FOLLOWER`, or `READ_REPLICA`.

Example 72. Get an overview of the cluster

The following example shows how to explore the cluster topology.

```
CALL dbms.cluster.overview()
```

id	addresses	groups	databases
08eb9305-53b9-4394-9237-0f0d63bb05d5	[bolt://neo20:7687, http://neo20:7474, https://neo20:7473]	[]	{system: LEADER, neo4j: FOLLOWER}
cb0c729d-233c-452f-8f06-f2553e08f149	[bolt://neo21:7687, http://neo21:7474, https://neo21:7473]	[]	{system: FOLLOWER, neo4j: FOLLOWER}
ded9eed2-dd3a-4574-bc08-6a569f91ec5c	[bolt://neo22:7687, http://neo22:7474, https://neo22:7473]	[]	{system: FOLLOWER, neo4j: LEADER}
00000000-0000-0000-0000-000000000000	[bolt://neo34:7687, http://neo34:7474, https://neo34:7473]	[]	{system: READ_REPLICA, neo4j: READ_REPLICA}
00000000-0000-0000-0000-000000000000	[bolt://neo28:7687, http://neo28:7474, https://neo28:7473]	[]	{system: READ_REPLICA, neo4j: READ_REPLICA}
00000000-0000-0000-0000-000000000000	[bolt://neo31:7687, http://neo31:7474, https://neo31:7473]	[]	{system: READ_REPLICA, neo4j: READ_REPLICA}

Get routing recommendations

From the application point of view it is not interesting to know about the role a member plays in the cluster. Instead, the application needs to know which instance can provide the wanted service. The procedure `dbms.routing.getRoutingTable(routingContext, databaseName)` provides this information.

Syntax:

`CALL dbms.routing.getRoutingTable(routingContext, databaseName)`

Arguments:

Name	Type	Description
`routingContext`	Map	The routing context used for multi-data center deployments. It should be used in combination with multi-data center load balancing.
`databaseName`	String	The name of the database to get the routing table for.

Example 73. Get routing recommendations

The following example shows how discover which instances in the cluster can provide which services for database `neo4j`.

```
CALL dbms.routing.getRoutingTable({}, "neo4j")
```

The procedure returns a map between a particular service, `READ`, `WRITE` and `ROUTE`, and the addresses of instances that provide this service. It also returns a Time To Live (TTL) in seconds as a suggestion on how long the client could cache the response.

The result is not primarily intended for human consumption. Expanding it this is what it looks like.

```
{
    "ttl": 300,
    "servers": [
        {
            "addresses": ["neo20:7687"],
            "role": "WRITE"
        },
        {
            "addresses": ["neo21:7687", "neo22:7687", "neo34:7687", "neo28:7687", "neo31:7687"],
            "role": "READ"
        },
        {
            "addresses": ["neo20:7687", "neo21:7687", "neo22:7687"],
            "role": "ROUTE"
        }
    ]
}
```

12.6.2. Endpoints for status information

This section describes HTTP endpoints for monitoring the health of a Neo4j Causal Cluster.

A Causal Cluster exposes some HTTP endpoints which can be used to monitor the health of the cluster. In this section we will describe these endpoints and explain their semantics.

The section includes:

- Adjusting security settings for Causal Clustering endpoints
- Unified endpoints

Adjusting security settings for Causal Clustering endpoints

If authentication and authorization is enabled in Neo4j, the Causal Clustering status endpoints will also require authentication credentials. The setting `dbms.security.auth_enabled` controls whether the native auth provider is enabled. For some load balancers and proxy servers, providing authentication credentials with the request is not an option. For those situations, consider disabling authentication of the Causal Clustering status endpoints by setting `dbms.security.causal_clustering_status_auth_enabled=false` in *neo4j.conf*.

Unified endpoints

A unified set of endpoints exist, both on Core Servers and on Read Replicas, with the following behavior:

- `/db/<databasename>/cluster/writable` — Used to direct `write` traffic to specific instances.
- `/db/<databasename>/cluster/read-only` — Used to direct `read` traffic to specific instances.
- `/db/<databasename>/cluster/available` — Available for the general case of directing arbitrary request types to instances that are available for processing read transactions.
- `/db/<databasename>/cluster/status` — Gives a detailed description of this instance's view of its own status within the cluster. See Status endpoint for further details.

Every endpoint targets a specific database with its own Raft-group. The `databaseName` path parameter represents the name of the database. By default, a fresh Neo4j installation will have endpoints for two databases `system` and `neo4j`:

```
http://localhost:7474/db/system/cluster/writable
http://localhost:7474/db/system/cluster/read-only
http://localhost:7474/db/system/cluster/available
http://localhost:7474/db/system/cluster/status

http://localhost:7474/db/neo4j/cluster/writable
http://localhost:7474/db/neo4j/cluster/read-only
http://localhost:7474/db/neo4j/cluster/available
http://localhost:7474/db/neo4j/cluster/status
```

Table 38. Unified HTTP endpoint responses

Endpoint	Instance state	Returned code	Body text
`/db/<databasename>/cluster/writable`	Leader	`200 OK`	`true`
	Follower	`404 Not Found`	`false`
	Read Replica	`404 Not Found`	`false`
`/db/<databasename>/cluster/read-only`	Leader	`404 Not Found`	`false`
	Follower	`200 OK`	`true`
	Read Replica	`200 OK`	`true`

Endpoint	Instance state	Returned code	Body text
/db/<databasename>/cluster/available	Leader	200 OK	true
	Follower	200 OK	true
	Read Replica	200 OK	true
/db/<databasename>/cluster/status	Leader	200 OK	JSON - See Status endpoint for details.
	Follower	200 OK	JSON - See Status endpoint for details.
	Read Replica	200 OK	JSON - See Status endpoint for details.

Example 74. Use a Causal Clustering monitoring endpoint

From the command line, a common way to ask those endpoints is to use `curl`. With no arguments, `curl` will do an HTTP `GET` on the URI provided and will output the body text, if any. If you also want to get the response code, just add the `-v` flag for verbose output. Here are some examples:

- Requesting `writable` endpoint on a Core Server that is currently elected leader with verbose output:

```
#> curl -v localhost:7474/db/neo4j/cluster/writable
* About to connect() to localhost port 7474 (#0)
*   Trying ::1...
* connected
* Connected to localhost (::1) port 7474 (#0)
> GET /db/neo4j/clustering/writable HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
> Host: localhost:7474
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Access-Control-Allow-Origin: *
< Transfer-Encoding: chunked
< Server: Jetty(9.4.17)
<
* Connection #0 to host localhost left intact
true* Closing connection #0
```

Status endpoint

Typically, you will want to have some guarantee that a Core is safe to shutdown before removing it from a cluster. The status endpoint provides the following information in order to help resolve such issues:

Example 75. Example status response

```
{
  "lastAppliedRaftIndex":0,
  "votingMembers":["30edc1c4-519c-4030-8348-7cb7af44f591","80a7fb7b-c966-4ee7-88a9-35db8b4d68fe"
,"f9301218-1fd4-4938-b9bb-a03453e1f779"],
  "memberId":"80a7fb7b-c966-4ee7-88a9-35db8b4d68fe",
  "leader":"30edc1c4-519c-4030-8348-7cb7af44f591",
  "millisSinceLastLeaderMessage":84545,
  "participatingInRaftGroup":true,
  "core":true,
  "healthy":true,
  "raftCommandsPerSecond":124
}
```

Table 39. Status endpoint descriptions

Field	Type	Optional	Example	Description
`core`	boolean	no	`true`	Used to distinguish between Core Servers and Read Replicas.
`lastAppliedRaftIndex`	number	no	`4321`	Every transaction in a cluster is associated with a raft index. Gives an indication of what the latest applied raft log index is.
`participatingInRaftGroup`	boolean	no	`false`	A participating member is able to vote. A Core is considered participating when it is part of the voter membership and has kept track of the leader.
`votingMembers`	string[]	no	`[]`	A member is considered a voting member when the leader has been receiving communication with it. List of member's `memberId` that are considered part of the voting set by this Core.
`healthy`	boolean	no	`true`	Reflects that the local database of this member has not encountered a critical error preventing it from writing locally.
`memberId`	string	no	`30edc1c4-519c-4030-8348-7cb7af44f591`	Every member in a cluster has it's own unique member id to identify it. Use `memberId` to distinguish between Core and Read Replica.
`leader`	string	yes	`80a7fb7b-c966-4ee7-88a9-35db8b4d68fe`	Follows the same format as `memberId`, but if it is null or missing, then the leader is unknown.
`millisSinceLastLeaderMessage`	number	yes	`1234`	The number of milliseconds since the last heartbeat-like leader message. Not relevant to Read Replicas, and hence is not included.
`raftCommandsPerSecond`	number	yes	`124`	An estimate of the average Raft state machine throughput over a sampling windown configurable via `causal_clustering.status_throughput_window` setting.

In general, you will want to follow the pattern of first adding a new, updated instance, and then removing an old instance. After an instance has been switched on, you can access the status endpoint in order to make sure all the guarantees listed in the table below are met. This process can then be repeated until all old Cores have been removed.

Table 40. Measured values, accessed via the status endpoint

Name of check	Method of calculation	Description
`allServersAreHealthy`	Every Core's status endpoint indicates `dbHealth==true`.	We want to make sure the data across the entire cluster is healthy. Whenever any Cores are false that indicates a larger problem.
`allVotingSetsAreEqual`	For any 2 Cores (A and B), status endpoint A's `votingMembers`== status endpoint B's `votingMembers`.	When the voting begins, all the Cores are equal to each other, and you know all members agree on membership.
`allVotingSetsContainAtLeastTargetCluster`	For all Cores (**S**), excluding Core Z (to be switched off), every member in **S** contains **S** in their voting set. Membership is determined by using the `memberId` and `votingMembers` from the status endpoint.	Sometimes network conditions will not be perfect and it may make sense to switch off a different Core to the one we originally wanted to switch off. If you run this check for all Cores, the ones that match this condition can be switched off (providing other conditions are also met).
`hasOneLeader`	For any 2 Cores (A and B), `A.leader == B.leader && leader!=null`.	If the leader is different then there may be a partition (alternatively, this could also occur due to bad timing). If the leader is unknown, that means the leader messages have actually timed out.
`noMembersLagging`	For Core A with `lastAppliedRaftIndex` = `min`, and Core B with `lastAppliedRaftIndex` = `max`, `B.lastAppliedRaftIndex-A.lastAppliedRaftIndex<raftIndexLagThreshold`.	If there is a large difference in the applied indexes between Cores, then it could be dangerous to switch off a Core.

12.7. Monitoring individual database states

This section covers the use of `SHOW DATABASES`, and other related Cypher commands.

In addition to the system-wide metrics and logs described in previous sections, operators may wish to monitor the state of individual databases being hosted within a Neo4j instance. The `SHOW DATABASES` command may be used for this purpose.

12.7.1. Listing Databases

First ensure that you are executing queries against the `system` database, either by running the command `:use system` (if using the Cypher shell or Neo4j Browser) or by creating a session against the `system` database using a Neo4j driver. Subsequently, run the `SHOW DATABASES` command.

Syntax:

```
SHOW DATABASES
```

Returns:

Name	Type	Description
`name`	String	The human-readable name of the database.
`address`	String	The bolt address of the Neo4j instance hosting the database.
`role`	String	The cluster role which the Neo4j instance fulfils for this database.

Name	Type	Description
`requestedStatus`	String	The state that an operator has requested the database to be in.
`currentStatus`	String	The state the database is actually in on this Neo4j instance.
`error`	String	Error encountered by the Neo4j instance when transitioning the database to `requestedStatus`, if any.
`default`	String	Whether this database is the default for this DBMS.

Example 76. Listing databases in standalone Neo4j

When executing `SHOW DATABASES` against a standalone instance of Neo4j, you should see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:7687"	"standalone"	"online"	"online"	""	true
"system"	"localhost:7687"	"standalone"	"online"	"online"	""	false

Note that the `role` and `address` columns are primarily intended to distinguish between the states of a given database, across multiple Neo4j instances deployed in a Causal Cluster. In a standalone deployment where you have a single Neo4j instance, your `address` field should be the same for every database, and your `role` field should always be "standalone".

If an error occurred whilst creating (or stopping, dropping etc.) a database, you should see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:7687"	"standalone"	"online"	"online"	""	true
"system"	"localhost:7687"	"standalone"	"online"	"online"	""	false
"foo"	"localhost:7687"	"standalone"	"online"	"Reached maximum number of active databases ..."	""	false

Note that for failed databases, the `currentStatus` and `requestedStatus` are different. This can imply an error. For example:

- a database may take a while to transition from "offline" to "online", due to performing recovery.
- during normal operation, the `currentStatus` of a database may be transiently different from its `requestedStatus`, due to a necessary automatic process, such as one Neo4j instance copying store files from another.

The possible statuses are "initial", "online", "offline", "store copying", and "unknown".

Example 77. Listing databases in a Neo4j Causal Cluster

When running `SHOW DATABASES` against a Neo4j Causal Cluster you might see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:20031"	"follower"	"online"	"online"	""	true
"neo4j"	"localhost:20010"	"follower"	"online"	"online"	""	true
"neo4j"	"localhost:20005"	"leader"	"online"	"online"	""	true

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:20034"	"read_replica"	"online"	"online"	""	true
"system"	"localhost:20031"	"follower"	"online"	"online"	""	false
"system"	"localhost:20010"	"follower"	"online"	"online"	""	false
"system"	"localhost:20005"	"leader"	"online"	"online"	""	false
"system"	"localhost:20034"	"read_replica"	"online"	"online"	""	false
"foo"	"localhost:20031"	"leader"	"online"	"online"	""	false
"foo"	"localhost:20010"	"follower"	"online"	"online"	""	false
"foo"	"localhost:20005"	"follower"	"online"	"online"	""	false
"foo"	"localhost:20034"	"read_replica"	"online"	"online"	""	false

Note that `SHOW DATABASES` does **not** return 1 row per database. Instead, it returns 1 row per database, per Neo4j instance in the cluster. Therefore, if you have a 4-instance cluster, hosting 3 databases, you will have 12 rows.

If an error occurred whilst creating (or stopping, dropping etc.) a database, you should see output like the following:

name	address	role	requestedStatus	currentStatus	error	default
"neo4j"	"localhost:20031"	"follower"	"online"	"online"	""	true
"neo4j"	"localhost:20010"	"follower"	"online"	"online"	""	true
"neo4j"	"localhost:20005"	"leader"	"online"	"online"	""	true
"neo4j"	"localhost:20034"	"read_replica"	"online"	"online"	""	true
"system"	"localhost:20031"	"follower"	"online"	"online"	""	false
"system"	"localhost:20010"	"follower"	"online"	"online"	""	false
"system"	"localhost:20005"	"leader"	"online"	"online"	""	false
"system"	"localhost:20034"	"read_replica"	"online"	"online"	""	false
"foo"	"localhost:20031"	"unknown"	"online"	"initial"	""	false
"foo"	"localhost:20010"	"leader"	"online"	"online"	""	false
"foo"	"localhost:20005"	"follower"	"online"	"online"	""	false
"foo"	"localhost:20034"	"unknown"	"online"	"initial"	""	false

Note that different instances may have different roles for each database.

If a database is offline on a particular Neo4j instance, either because it was stopped by an operator or an error has occurred, its cluster `role` is "unknown". This is because the cluster role of a given instance/database combination cannot be assumed in advance. This differs from standalone Neo4j instances, where the role of that instance for each database can always be assumed to be "standalone".

The possible roles are "standalone", "leader", "follower", "read_replica", and "unknown".

12.7.2. Listing a single database

The number of rows returned by `SHOW DATABASES` can be quite large, especially when run in a cluster. You can filter the rows returned by database name (e.g. "foo") by using the command `SHOW DATABASE foo`.

Syntax:

```
SHOW DATABASE databaseName
```

Arguments:

Name	Type	Description
`databaseName`	String	The name of the database whose status to report

Returns:

Name	Type	Description
`name`	String	The human-readable name of the database.
`address`	String	The bolt address of the Neo4j instance hosting the database.
`role`	String	The cluster role which the Neo4j instance fulfils for this database.
`requestedStatus`	String	The state that an operator has requested the database to be in.
`currentStatus`	String	The state the database is actually in on this Neo4j instance.
`error`	String	Error encountered by Neo4j instance when transitioning the database to `requestedStatus`, if any.

Example 78. Listing statuses for database foo

When running `SHOW DATABASE foo` in a Neo4j Causal Cluster, you should see output like the following:

`name`	`address`	`role`	`requestedStatus`	`currentStatus`	`error`	`default`
"foo"	"localhost:20031"	"unknown"	"online"	"initial"	""	false
"foo"	"localhost:20010"	"leader"	"online"	"online"	""	false
"foo"	"localhost:20005"	"follower"	"online"	"online"	""	false
"foo"	"localhost:20034"	"unknown"	"online"	"initial"	""	false

Chapter 13. Performance

This chapter describes factors that affect operational performance, and how to tune Neo4j for optimal throughput.

The topics described in this chapter are:

- Memory configuration — How to configure memory settings for efficient operations.
- Index configuration — How to configure indexes.
- Garbage collector — How to configure the Java Virtual Machine's garbage collector.
- Bolt thread pool configuration — How to configure the Bolt thread pool.
- Linux file system tuning — How to configure the Linux file system.
- Disks, RAM and other tips — Disks, RAM and other tips.
- Statistics and execution plans — How schema statistics and execution plans affect Cypher query performance.

13.1. Memory configuration

This section describes the different aspects of Neo4j memory configuration and use.

13.1.1. Overview

Consider the image below. As you can see, the RAM of the Neo4j server has a number of usage areas, with some sub-areas:

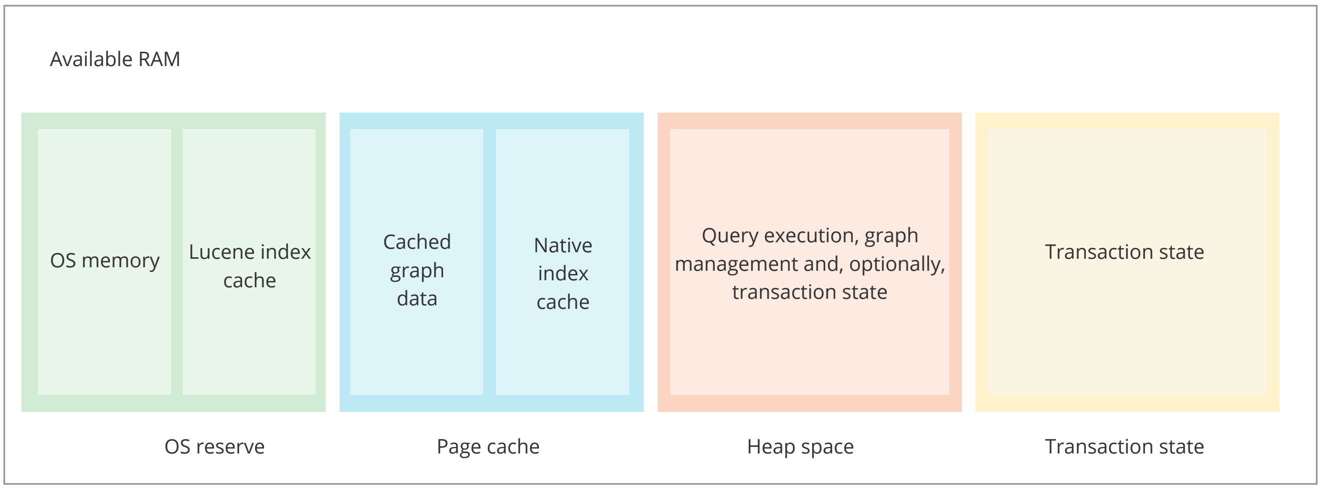

Figure 12. Neo4j memory management

OS memory

Some memory must be reserved for running the processes of the operating system itself. It is not possible to explicitly configure the amount of RAM that should be reserved for the operating system, as this is what RAM remains available after configuring page cache and heap space. However, if we do not leave enough space for the OS it will start swapping to disk, which will heavily affect performance.

1GB is a good starting point for a server that is dedicated to running Neo4j. However, there are cases where the amount reserved for the OS is significantly larger than 1GB, such as servers with exceptionally large RAM.

Lucene index cache

Neo4j uses Apache Lucene for some of its indexing functionality. Index lookup performance is optimized by ensuring that as much as possible of the indexes are cached into memory. Similar to the OS memory, the Lucene index cache can not be explicitly configured. Instead we estimate the memory needed and make sure that there is enough headroom left for Lucene indexes after the page cache and heap cache have been assigned.

Page cache

The page cache is used to cache the Neo4j data and native indexes. The caching of graph data and indexes into memory will help avoid costly disk access and result in optimal performance.

The parameter for specifying how much memory Neo4j is allowed to use for the page cache is: `dbms.memory.pagecache.size`.

Heap size

The heap space is used for query execution, transaction state, management of the graph etc. The size needed for the heap is very dependent on the nature of the usage of Neo4j. For example, long-running queries, or very complicated queries, are likely to require a larger heap than simpler queries.

Generally speaking, in order to aid performance, we want to configure a large enough heap to sustain concurrent operations.

In case of performance issues we may have to tune our queries, and monitor their memory usage, in order to determine whether the heap needs to be increased.

The heap memory size is determined by the parameters `dbms.memory.heap.initial_size` and `dbms.memory.heap.max_size`. It is recommended to set these two parameters to the same value. This will help avoid unwanted full garbage collection pauses.

Transaction state

Transaction state is the memory that is needed to hold data and intermediate results in transactions that update records in the database. Queries that only read data do not require transaction state memory allocation. By default, transaction state is allocated off-heap. When the transaction state is allocated off-heap, the maximum size of the transaction state can be defined using the parameter `dbms.tx_state.max_off_heap_memory`. Note that the transaction state memory is not pre-allocated; it will grow and shrink as needed by the activity in the database. Keeping transaction state off-heap is particularly beneficial to applications characterized by large, write-intensive transactions.

Transaction state can also be configured to be allocated on-heap, by using the parameter `dbms.tx_state.memory_allocation`. Note that when the transaction state is configured on-heap, its maximum size cannot be specified.

13.1.2. Considerations

Always use explicit configuration

In order to have good control of a system’s behavior, it is recommended that you always define the page cache and heap size parameters explicitly in *neo4j.conf*. If these parameters are not explicitly defined, some heuristic values will be computed at startup based on available system resources.

Initial memory recommendation

Use the `neo4j-admin memrec` command to get an initial recommendation for how to distribute a certain amount of memory. The values may need to be adjusted to cater for each specific use case.

Inspect the memory settings of a database

The `neo4j-admin memrec --database` command is useful for inspecting the current distribution of data and indexes in an existing database.

Example 79. Use `neo4j-admin memrec` to inspect the memory settings of a database

We wish to estimate the total size of the database files.

```
$neo4j-home> bin/neo4j-admin memrec --database=neo4j
...
...
...
# Lucene indexes: 6690m
# Data volume and native indexes: 17050m
```

We can see that the Lucene indexes take up approximately 6.7GB of data, and that the data volume and native indexes combined take up approximately 17GB.

Using this information, we can do a sanity check of our memory configuration:

- Compare the value for data volume and native indexes to the value of `dbms.memory.pagecache.size`.
- For cases when *off-heap* transaction state is used, estimate transactional workload and how much memory is left to the value of `dbms.tx_state.max_off_heap_memory`.
- Compare the value for Lucene indexes to how much memory is left after assigning `dbms.memory.pagecache.size` and `dbms.memory.heap.initial_size`.

Note that even though we strive for caching as much of our data and indexes as possible, in some production systems the access to memory is limited and must be negotiated between different areas. Then there will will be a certain amount of testing and tuning to figure out the optimal division of the available memory.

13.1.3. Capacity planning

In many use cases, it is advantageous to try to cache as much of the data and indexes as possible. The following examples illustrate methods for estimating the page cache size, depending on whether we are already running in production or planning for a future deployment:

Example 80. Estimate page cache for an existing Neo4j database

First estimate the total size of data and indexes, and then multiply with some factor, for example 20%, to allow for growth.

```
$neo4j-home> bin/neo4j-admin memrec --database=neo4j
...
...
...
# Lucene indexes: 6690m
# Data volume and native indexes: 35050m
```

We can see that the data volume and native indexes combined take up approximately 35GB. In our specific use case we estimate that 20% will provide sufficient head room for growth.

`dbms.memory.pagecache.size` = 1.2 * (35GB) = 42GB

We configure the page cache by adding the following to *neo4j.conf*:

```
dbms.memory.pagecache.size=42GB
```

Example 81. Estimate page cache for a new Neo4j database

When planning for a future database, it is useful to run an import with a fraction of the data, and then multiply the resulting store size by that fraction plus some percentage for growth. For example, import 1/100th of the data and measure its data volume and native indexes. Then multiply that number by 120 to size up the result, and allow for 20% growth.

Assume that we have imported 1/100th of the data into a test database.

```
$neo4j-home> bin/neo4j-admin memrec --database=neo4j
...
...
...
# Lucene indexes: 425.0
# Data volume and native indexes: 251100k
```

We can see that the data volume and native indexes combined take up approximately 250MB. We size up the result and additionally reserve 20% for growth:

`dbms.memory.pagecache.size` = 120 * (250MB) = 30GB

We configure the page cache by adding the following to *neo4j.conf*:

```
dbms.memory.pagecache.size=30G
```

13.1.4. Configure query heap usage

When running a Cypher query, Neo4j will utilize the heap internally for storing results. It may be difficult to predict how much memory a query needs and if a query ends up using too much memory it could severely hamper the overall performance of the database.

There are two settings that can be enabled in *neo4j.conf* that can help improve the heap utilization of Neo4j:

```
dbms.track_query_allocation=true
cypher.query_max_allocations.size=1G
```

If the former setting is enabled, Neo4j will be tracking the heap utilization of all Cypher queries. You can view the utilization of running queries by calling:

```
CALL dbms.listQueries()
```

Or alternatively, you can enable `dbms.logs.query.allocation_logging_enabled` and the memory usage of each query will be logged in the query log.

By setting the latter configuration, `cypher.query_max_allocations.size`, you can limit the amount of memory each query can use. Whenever that limit is reached, the query will be gracefully terminated without affecting the overall health of the database.

The heap-usage of query is only an estimate and the actual heap utilization might be slightly bigger or slightly smaller than the estimated value.

13.2. Index configuration

This section describes how to configure indexes to enhance performance in search, and to enable full-text search.

This section contains the following:

- Introduction
- B-tree indexes
  - Limitations
    - Limitations for queries using `CONTAINS` and `ENDS WITH`
    - Limitations on key size
    - Workarounds to address limitations
  - Index migration
  - Procedures to create index and index backed constraint
- Full-text indexes
  - Configuration

13.2.1. Introduction

In Neo4j there are two different index types: b-tree and full-text.

B-tree indexes can be created and dropped using Cypher. Users typically do not have to know about the index in order to use it, since Cypher's query planner decides which index to use in which situation. B-tree indexes are good at exact look-ups on all types of values, and range scans, full scans, and prefix searches.

For details on the configuration aspects of b-tree indexes, see B-tree indexes.

Full-text indexes differ from b-tree indexes, in that they are optimised for indexing and searching text. They are used for queries that demand an understanding of language, and they only index string data. They must also be queried explicitly via procedures, as Cypher will not make plans that rely on them.

An example of a use case for full-text indexes is parsing a book for a certain term, and taking advantage of the knowledge that the book is written in a certain language. The use of an *analyzer* for that language will, among other things, enable you to exclude words that are not relevant for the search (for example *"if"* and *"and"*), and include conjugations of words that are.

Another use case example is indexing the various address fields and text data in a corpus of emails. Indexing this data using the `email` analyzer would enable someone to find all emails that are sent from, or to, or mentions, an email account.

In contrast to b-tree indexes, full-text indexes are created, queried, and dropped using built-in procedures. The use of full-text indexes do require a familiarity with how the indexes operate.

For details on the configuration aspects of full-text indexes, see Full-text indexes.

For details on creating, querying and dropping full-text indexes, see Cypher Manual Indexes to support full-text search.

The type of an index can be identified according to the table below:

Index type	Procedure	Core API
B-tree index	`db.indexes#BTREE`	`org.neo4j.graphdb.schema.IndexType#BTREE`
Full-text index	`db.indexes#FULLTEXT`	`org.neo4j.graphdb.schema.IndexType#FULLTEXT`

13.2.2. B-tree indexes

B-tree indexes can be backed by two different index providers, `native-btree-1.0` and the deprecated `lucene+native-3.0`. If not explicitly set, `native-btree-1.0` will be used.

For more information on the different index types, refer to Cypher Manual Indexes.

Deprecated index providers

Index provider `lucene+native-3.0` has been deprecated, and will be removed in a future release.

The recommended index provider to use is `native-btree-1.0`.

The only reason to use a deprecated provider should be due to the limitations, as described in Limitations for queries using `CONTAINS` and `ENDS WITH` or Limitations on key size.

Limitations

In this section a few limitations for b-tree indexes are described, together with suggested workarounds.

Limitations for queries using `CONTAINS` and `ENDS WITH`

The index provider `native-btree-1.0` has limited support for `ENDS WITH` and `CONTAINS` queries. These queries will not be able to do an optimized search as per queries that use `STARTS WITH`, `=`, and `<>`. Instead, the index result will be a stream of an index scan with filtering.

In the future, `ENDS WITH` and `CONTAINS` queries will be supported with full-text indexes, but for now the deprecated index provider `lucene+native-3.0` can be used instead. Please note that `lucene+native-3.0` only has support for `ENDS WITH` and `CONTAINS` for single property strings.

- For details about execution plans, refer to Cypher Manual → Execution plans.
- For details about string operators, refer to Cypher Manual → Operators.

Limitations on key size

The index provider `native-btree-1.0` has a key size limit of around 8kB.

If a transaction reaches the key size limit for one or more of its changes, that transaction will fail before committing any changes. If the limit is reached during index population, the resulting index will be in a failed state, and as such will not be usable for any queries.

If this is an issue, you can use the deprecated index provider `lucene+native-3.0` instead. This provider has a key size limit for single property strings of around 32kB.

Please note that `lucene+native-3.0` has been deprecated and will be removed in the future, whereby 8kB will be the key size limit for b-tree indexes. The recommended option for such large values is to use full-text index.

Workarounds to address limitations

To workaround problems with key size, or performance issues related to `ENDS WITH` or `CONTAINS`, you can use the deprecated index provider `lucene+native-3.0`. This only works for single-property string indexes.

This can be done using either of the following methods:

Option 1. Use a built-in procedure (recommended)

There are built-in procedures that can be used to specify index provider on index creation, unique property constraint creation, and node key creation (for details on constraints, see Cypher manual → Constraints.

For more information, see Built-in procedures.

Option 2. Change the config

1. Configure the setting `dbms.index.default_schema_provider` to the one required.
2. Restart Neo4j.
3. Drop and recreate the relevant index.
4. Change `dbms.index.default_schema_provider` back to the original value.
5. Restart Neo4j.

Please note that the index setting `dbms.index.default_schema_provider` has been deprecated, and will be removed in a future release. It will be a fully internal concern which index provider an index is using.

The recommended way to set index provider for an index is to use the built in procedures for index creation, unique property constraint creation, and node key creation.

For more information, see Built-in procedures

Index migration

When upgrading a 3.5 store to 4.0.0, all indexes will be upgraded to the latest index version, and rebuilt automatically, with the exception for the indexes that were previously using Lucene for single-property strings. They will be upgraded to a fallback version which still use Lucene for those properties. Please note that they will still need to be rebuilt.

The table below shows the migration mapping:

Index provider in 3.5	Index provider in 4.0
`native-btree-1.0`	`native-btree-1.0`
`lucene+native-2.0`	`native-btree-1.0`
`lucene+native-1.0`	`lucene+native-3.0`
`lucene-1.0`	`lucene+native-3.0`

The caching of indexes takes place in different memory areas for different index providers. See Memory configuration. It can be useful to run `neo4j-admin memrec --database` before and after the rebuilding of indexes, and adjust memory settings in accordance with the findings.

Procedures to create index and index backed constraint

Indexes and constraints are best created through Cypher, but when these indexes or constraints need to be more specifically configured than what is possible through Cypher, then you can use the procedures described in the example below.

Example 82. Example of procedures to create index and index backed constraint

The following procedures provide the option to specify both index provider and index settings (optional). Note that settings keys need to be escaped with back-ticks if they contain dots.

Use `db.createIndex` procedure to create an index:

```
CALL db.createIndex("MyIndex", ["Person"], ["name"], "native-btree-1.0", {+`spatial.cartesian.max+`:
[100.0,100.0], +`spatial.cartesian.min+`: [-100.0,-100.0]})
```

If a settings map is not provided, the settings will be picked up from the Neo4j config file, the same way as when creating an index or constraint through Cypher.

```
CALL db.createIndex("MyIndex", ["Person"], ["name"], "native-btree-1.0")
```

Use `db.createUniquePropertyConstraint` to create a node property uniqueness constraint (the example is without settings map, left out for abbreviation):

```
CALL db.createUniquePropertyConstraint("MyIndex", ["Person"], ["name"], "native-btree-1.0",
{+`spatial.cartesian.max+`: [100.0,100.0], +`spatial.cartesian.min+`: [-100.0,-100.0]})
```

Use `db.createNodeKey` to create node key constraint (the example is without settings map, left out for abbreviation):

```
CALL db.createNodeKey("MyIndex", ["Person"], ["name"], "native-btree-1.0",
{+`spatial.cartesian.max+`: [100.0,100.0], +`spatial.cartesian.min+`: [-100.0,-100.0]})
```

13.2.3. Full-text indexes

Full-text indexes are powered by the Apache Lucene indexing and search library. A full-text index enables you to write queries that matches within the *contents* of indexed string properties. A full description on how to create and use full-text indexes is provided in the Cypher Manual Indexes to support full-text search.

Configuration

The following options are available for configuring full-text indexes:

`dbms.index.fulltext.default_analyzer`

The name of the analyzer that the full-text indexes should use by default. This setting only has effect when a full-text index is created, and will be remembered as an index-specific setting from then on.

The list of possible analyzers is available through the `db.index.fulltext.listAvailableAnalyzers()` Cypher procedure.

Unless otherwise specified, the default analyzer is `standard-no-stop-words`, which is the same as the `StandardAnalyzer` from Lucene, except no stop-words are filtered out.

`dbms.index.fulltext.eventually_consistent`

Used to declare whether full-text indexes should be eventually consistent, or not. This setting only has effect when a full-text index is created, and will be remembered as an index-specific setting from then on.

Indexes are normally fully consistent, and the committing of a transaction does not return until both the store and the indexes have been updated. Eventually consistent full-text indexes, on the other hand, are not updated as part of commit, but instead have their updates queued up and applied in a background thread. This means that there can be a short delay between committing a change, and that change becoming visible via any eventually consistent full-text indexes. This delay is just an artifact of the queueing, and will usually be quite small since eventually consistent indexes are updated "as soon as possible".

By default, this is turned off, and full-text indexes are fully consistent.

`dbms.index.fulltext.eventually_consistent_index_update_queue_max_length`

Eventually consistent full-text indexes have their updates queued up and applied in a background thread, and this setting determines the maximum size of that update queue. If the maximum queue size is reached, then committing transactions will block and wait until there is more room in the queue, before adding more updates to it.

This setting applies to all eventually consistent full-text indexes, and they all use the same queue. The maximum queue length must be at least 1 index update, and must be no more than 50 million due to heap space usage considerations.

The default maximum queue length is 10.000 index updates.

13.3. Tuning of the garbage collector

This section discusses the effect of the Java Virtual Machine's garbage collector with regards to Neo4j performance.

The heap is separated into an *old generation* and a *young generation*. New objects are allocated in the young generation, and then later moved to the old generation, if they stay live (in use) for long enough. When a generation fills up, the garbage collector performs a collection, during which all other threads in the process are paused. The young generation is quick to collect since the pause time correlates with the *live set* of objects. In the old generation, pause times roughly correlates with the size of the heap. For this reason, the heap should ideally be sized and tuned such that transaction and query state never makes it to the old generation.

The heap size is configured with the `dbms.memory.heap.max_size` (in MBs) setting in the *neo4j.conf* file. The initial size of the heap is specified by the `dbms.memory.heap.initial_size` setting, or with the `-Xms???m` flag, or chosen heuristically by the JVM itself if left unspecified. The JVM will automatically grow the heap as needed, up to the maximum size. The growing of the heap requires a full garbage collection cycle. It is recommended to set the initial heap size and the maximum heap size to the same value. This way the pause that happens when the garbage collector grows the heap can be avoided.

If the new generation is too small, short-lived objects may be moved to the old generation too soon. This is called premature promotion and will slow the database down by increasing the frequency of old generation garbage collection cycles. If the new generation is too big, the garbage collector may decide that the old generation does not have enough space to fit all the objects it expects to promote from the new to the old generation. This turns new generation garbage collection cycles into old generation garbage collection cycles, again slowing the database down. Running more concurrent threads means that more allocations can take place in a given span of time, in turn increasing the pressure on the new generation in particular.

The *Compressed OOPs* feature in the JVM allows object references to be compressed to use only 32 bits. The feature saves a lot of memory but is only available for heaps up to 32 GB. The maximum applicable size varies from platform and JVM version. The `-XX:+UseCompressedOops` option can be used to verify whether the system can use the *Compressed OOPs* feature. If it cannot, this will will be logged in the default process output stream.

How to tune the specific garbage collection algorithm depends on both the JVM version and the workload. It is recommended to test the garbage collection settings under realistic load for days or weeks. Problems like heap fragmentation can take a long time to surface.

To gain good performance, these are the things to look into first:

- Make sure the JVM is not spending too much time performing garbage collection. The goal is to have a large enough heap to make sure that heavy/peak load will not result in so called GC-trashing. Performance can drop as much as two orders of magnitude when GC-trashing happens. Having too large heap may also hurt performance so you may have to try some different heap sizes.
- Neo4j needs enough heap memory for the transaction state and query processing, plus some head-room for the garbage collector. As heap memory requirements are so workload-dependent, it is common to see heap memory configurations from 1 GB, up to 32 GB.

Edit the following properties:

Table 41. neo4j.conf JVM tuning properties

Property Name	Meaning
`dbms.memory.heap.initial_size`	initial heap size (in MB)
`dbms.memory.heap.max_size`	maximum heap size (in MB)
`dbms.jvm.additional`	additional literal JVM parameter

13.4. Bolt thread pool configuration

This section discusses the thread pool infrastructure built into Bolt connectors, and how it can be configured.

The Bolt connector is backed by a thread pool on the server side. The thread pool is constructed as part of the server startup process.

13.4.1. How thread pooling works

The Bolt thread pool has a minimum and a maximum capacity. It starts with a minimum number of threads available, and grows up to the maximum count depending on the workload. Threads that sit idle for longer than a specified time period are stopped and removed from the pool in order to free up resources. However, the size of the pool will never go below the minimum.

Each connection being established is assigned to the connector's thread pool. Idle connections do not consume any resources on the server side, and they are monitored against messages arriving from the client. Each message arriving on a connection triggers the scheduling of a connection on an available thread in the thread pool. If all the available threads are busy, and there is still space to grow, a new thread is created and the connection is handed over to it for processing. If the pool capacity is filled up, and no threads are available to process, the job submission is rejected and a failure message is generated to notify the client of the problem.

The default values assigned to the Bolt thread pool will fit most workloads, so it is generally not necessary to configure the connection pool explicitly. If the maximum pool size is set too low, an exception will be thrown with an error message indicating that there are no available threads to serve. The message will also be written to *neo4j.log*.

Any connection with an active explicit, or implicit, transaction will stick to the thread that starts the transaction, and will not return that thread to the pool until the transaction is closed. Therefore, in applications that are making use of explicit transactions, it is important to close the transactions appropriately. To learn more about transactions, refer to the Neo4j Driver Manual.

13.4.2. Configuration options

The following configuration options are available for configuring the Bolt connector:

Table 42. Thread pool options

Option name	Default	Description
`dbms.connector.bolt.thread_pool_min_size`	`5`	The minimum number of threads that will always be up even if they are idle.
`dbms.connector.bolt.thread_pool_max_size`	`400`	The maximum number of threads that will be created by the thread pool.
`dbms.connector.bolt.thread_pool_keep_alive`	`5m`	The duration that the thread pool will wait before killing an idle thread from the pool. However, the number of threads will never go below `dbms.connector.bolt.thread_pool_min_size`.

13.4.3. How to size your Bolt thread pool

Select values for thread pool sizing based on your workload. Since each active transaction will borrow a thread from the pool until the transaction is closed, it is basically the minimum and maximum active transaction at any given time that determine the values for pool configuration options. You can use the monitoring capabilities (see Monitoring) of the database to discover more about your workload.

Configure `dbms.connector.bolt.thread_pool_min_size` based on your minimum or average workload. Since there will always be this many amount of threads in the thread pool, sticking with lower values may be more resource-friendly than having too many idle threads waiting for job submissions.

Configure `dbms.connector.bolt.thread_pool_max_size` based on your maximum workload. This should basically be set after the maximum number of active transactions that is expected on the server. You should also account for non-transaction operations that will take place on the thread pool, such as connection and disconnection of clients.

Example 83. Configure the thread pool for a Bolt connector

In this example we configure the Bolt thread pool to be of minimum size `5`, maximum size `100`, and have a keep-alive time of `10 minutes`.

```
dbms.connector.bolt.thread_pool_min_size=10
dbms.connector.bolt.thread_pool_max_size=100
dbms.connector.bolt.thread_pool_keep_alive=10m
```

13.5. Linux file system tuning

This section covers Neo4j I/O behavior, and how to optimize for operations on disk.

Databases often produce many small and random reads when querying data, and few sequential writes when committing changes. For maximum performance, it is recommended to store database and transaction logs on separate physical devices.

Often, recommended practice is to disable file and directory access time updates. This way, the file system won't have to issue writes that update this meta-data, thus improving write performance.

Since databases can put a high and consistent load on a storage system for a long time, it is recommended to use a file system that has good aging characteristics. The EXT4 and XFS file systems are recommended as a first choice.

A high read and write I/O load can also degrade SSD performance over time. The first line of defense against SSD wear is to ensure that the working dataset fits in RAM. A database with a high write workload will, however, still cause wear on SSDs. The simplest way to combat this is to over-provision; use SSDs that are at least 20% larger than you strictly need them to be.

To be able to achieve optimum performance, we do not recommend the use of NFS or NAS as database storage.

For more detailed and precise configuration for your operating system and hardware, please consult corresponding manuals.

13.6. Disks, RAM and other tips

This section provides an overview of performance considerations for disk and RAM when running Neo4j.

As with any persistence solution, performance depends a lot on the persistence media used. In general, the faster storage you have, and the more of your data you can fit in RAM, the better performance you will get.

13.6.1. Storage

If you have multiple disks or persistence media available it may be a good idea to divide the store files and transaction logs across those disks. Keeping the store files on disks with low seek time can do wonders for read operations.

To achieve maximum performance, it is recommended to provide Neo4j with as much RAM as possible in order to avoid hitting the disk.

Use tools like `dstat` or `vmstat` to gather information when your application is running. If the swap or paging numbers are high, that is a sign that the database don't quite fit in memory. In this case, database access can have high latencies.

13.6.2. Page cache

When Neo4j starts up, its page cache is empty and needs to warm up. The pages, and their graph data contents, are loaded into memory on demand as queries need them. This can take a while, especially for large stores. It is not uncommon to see a long period with many blocks being read from the drive, and high IO wait times. This will show up in the page cache metrics as an initial spike in page faults. The page fault spike is then followed by a gradual decline of page fault activity, as the probability of queries needing a page that is not yet in memory drops.

Active page cache warmup

The Neo4j Enterprise Edition has a feature called *active page cache warmup*, which shortens the page fault spike and makes the page cache warm up faster. This is done by periodically recording *cache profiles* of the store files, as the database is running. These profiles contain information about what data is and is not in memory, and are stored in a "profiles" sub-directory of the store directory. When Neo4j is later restarted, it will look for these cache profiles and eagerly load in the same data that was in memory when the profile was created. The profiles are also copied as part of online backup and cluster store-copy, and helps warm up new database instances that join a cluster.

It is also possible to configure *page cache warmup* to prefetch database data. To enable it, you can use the `dbms.memory.pagecache.warmup.preload` setting. Data prefetching will disregard the cache profile files, and will prefetch data into a page cache as part of the startup. This can be quite useful in scenarios where the database size is smaller than the page cache, and you want to make sure that data is loaded before it will be required.

Checkpoint IOPS limit

Neo4j flushes its page cache in the background as part of its checkpoint process. This will show up as a period of elevated write IO activity. If the database is serving a write-heavy workload, the checkpoint can slow the database down by reducing the IO bandwidth that is available to query processing. Running the database on a fast SSD, which can service a lot of random IOs, significantly reduces this problem. If a fast SSD is not available in your environment, or if it is insufficient, then an artificial IOPS limit can be placed on the checkpoint process. The `dbms.checkpoint.iops.limit` restricts the IO bandwidth that the checkpoint process is allowed to use. Each IO is, in the case of the checkpoint process, an 8 KiB write. An IOPS limit of 300, for instance, would thus only allow the checkpoint process to write at a rate of roughly 2.5 MiB per second. This will, on the other hand, make checkpoints take longer to complete. A longer time between checkpoints can cause more transaction log data to accumulate, and can lengthen recovery times. See the transaction logs section for more details on the relationship between checkpoints and log pruning. The IOPS limit can be changed at runtime, making it possible to tune it until you have the right balance between IO usage and checkpoint time.

13.7. Statistics and execution plans

This section describes the configuration options that affect the gathering of statistics, and the replanning of query plans in the Cypher query engine.

When a Cypher query is issued, it gets compiled to an execution plan that can run and answer the query. The Cypher query engine uses available information about the database, such as schema information about which indexes and constraints exist in the database. Neo4j also uses statistical information about the database to optimize the execution plan.

For further details, please see Cypher Manual Query tuning and Cypher Manual Execution plans.

The frequency of statistics gathering and the replanning of execution plans are described in the sections below.

13.7.1. Statistics

The statistical information that Neo4j keeps is:

1. The number of nodes having a certain label.
2. The number of relationships by type.
3. The number of relationships by type, ending or starting from a node with a specific label.
4. Selectivity per index.

Neo4j keeps the statistics up to date in two different ways. For label and relationship counts, the number is updated whenever you set or remove a label from a node. For indexes, however, Neo4j needs to scan the full index to produce the selectivity number. Since this is potentially a very time-consuming operation, these numbers are collected in the background when enough data on the index has been changed.

The following settings allow you to control whether statistics are collected automatically, and at which rate:

Parameter name	Default value	Description
`dbms.index_sampling.background_enabled`	`true`	Controls whether indexes will automatically be re-sampled when they have been updated enough. The Cypher query planner depends on accurate statistics to create efficient plans, so it is important it is kept up to date as the database evolves.
`dbms.index_sampling.update_percentage`	`5`	Controls the percentage of the index that has to have been updated before a new sampling run is triggered.

It is possible to manually trigger index sampling, using the following built-in procedures:

`db.resampleIndex()`
: Trigger resampling of an index.

`db.resampleOutdatedIndexes()`
: Trigger resampling of all outdated indexes.

Example 84. Manually trigger index resampling

The following example illustrates how to trigger a resampling of the index on the label `Person` and property `name`, by calling `db.resampleIndex()`:

```
CALL db.resampleIndex(":Person(name)");
```

The following example illustrates how to call `db.resampleOutdatedIndexes()` in order to trigger a resampling of all outdated indexes:

```
CALL db.resampleOutdatedIndexes();
```

13.7.2. Execution plans

Execution plans are cached and will not be replanned until the statistical information used to produce the plan has changed. The following setting enables you to control how sensitive replanning should be to updates of the database:

Parameter name	Default value	Description
`cypher.statistics_divergence_threshold`	`0.75`	The threshold when a plan is considered stale. If any of the underlying statistics used to create the plan have changed more than this value, the plan will be considered stale and will be replanned. Change is calculated as `abs(a-b)/max(a,b)`. This means that a value of `0.75` requires the database to approximately quadruple in size before replanning occurs. A value of 0 means replan as soon as possible, with the soonest being defined by the parameter `cypher.min_replan_interval`, which defaults to 10s. After this interval the divergence threshold will slowly start to decline, reaching 10% after about 7h. This will ensure that long-running databases will still get query replanning on even modest changes, while not replanning frequently unless the changes are very large.

It is possible to manually force the database to replan execution plans that are already in the cache, using the following built-in procedures:

`db.clearQueryCaches()`

Clears out all query caches, but does not change the database statistics.

```
CALL db.clearQueryCaches();
```

`db.prepareForReplanning()`

Completely recalculates all database statistics, so that they can be used for any subsequent query planning.

It triggers an index resampling and waits for it to complete, and after that it also clears all query caches. After this procedure has finished queries will be planned on empty caches using the latest database statistics.

```
CALL db.prepareForReplanning();
```

For more information, see Built-in procedures.

Chapter 14. Tools

This chapter describes the Neo4j tools Neo4j Admin and Cypher Shell, and explains how to use commands to import data into a new database, dump and load an existing database, and how to check consistency.

This chapter comprises the following topics:

- A description of the Neo4j Admin tool
- How to check the consistency of a Neo4j database using Neo4j Admin
- How to collect the most common information needed for remote assessments
- How to display information about a database store
- How to get an initial recommendation for Neo4j memory settings
- How to import data into Neo4j using the import tool
    - Syntax
    - CSV header format
    - Options
- How to dump and load Neo4j databases using Neo4j Admin
- How to remove cluster state data from a Neo4j server.
- How to copy data from an existing database to a new database.
- How to use the Cypher Shell

14.1. Neo4j Admin

This section describes the Neo4j Admin tool.

This section describes the following:

- Introduction
- Syntax and commands
- Environment variables
- Exit codes

14.1.1. Introduction

Neo4j Admin is the primary tool for managing your Neo4j instance. It is a command-line tool that is installed as part of the product and can be executed with a number of commands. Some of the commands are described in more detail in separate sections.

14.1.2. Syntax and commands

Syntax

Neo4j Admin is located in the *bin* directory and is invoked as:

```
neo4j-admin <command>
```

Commands

Functionality area	Command	Description

General	`help`
	`help <command>`
		Display help text.
		
		Display help text for `<command>`

	`check-consistency`	Check the consistency of a database. For details, see Consistency checker.
	`report`	Collect the most common information needed for remote assessments. For details, see Report tool.
	`store-info`	Print information about a Neo4j database store. For details, see Display store information.
	`memrec`	Print Neo4j heap and pagecache memory settings recommendations. For details, see Memory recommendations.
	`import`	Import from a collection of CSV files or a pre-3.0 database. For details, see Import.
Authentication	`set-default-admin`	Sets the default admin user when no roles are present.
	`set-initial-password`	Sets the initial password of the initial admin user (`neo4j`). For details, see Set an initial password.
Offline backup For details see Dump and load databases.	`dump`	Dump a database into a single-file archive.
	`load`	Load a database from an archive created with the dump command.
Online backup For details see Backup.	`backup`	Perform an online backup from a running Neo4j server.
	`restore`	Restore a backed-up database.
Clustering	`unbind`	Removes cluster state data from a stopped Neo4j server. For details, see Unbind a Core Server.

Limitations

`neo4j-admin` must be invoked as the `neo4j` user in order to ensure the appropriate file permissions.

14.1.3. Environment variables

Neo4j Admin can utilize the following environment variables:

NEO4J_DEBUG
: Set to anything to enable debug output.

NEO4J_HOME
: Neo4j home directory.

NEO4J_CONF
: Path to directory which contains *neo4j.conf*.

HEAP_SIZE

Set JVM maximum heap size during command execution. Takes a number and a unit, for example 512m.

JAVA_OPTS
Additional JVM arguments.

14.1.4. Exit codes

If `neo4j-admin` finishes as expected it will return an exit code of `0`. A non-zero exit code means something undesired happened during command execution. The non-zero exit code can contain further information about the error, such as the `backup` command's exit codes.

14.2. Consistency checker

This section describes the Neo4j consistency checker.

The consistency of a database or a backup can be checked using the `check-consistency` argument to the `neo4j-admin` tool.

14.2.1. Check consistency of a database or a backup

The `neo4j-admin` tool is located in the `bin` directory. Run it with the `check-consistency` argument in order to check the consistency of a database.

Syntax

```
neo4j-admin check-consistency ([--database=<database>] | [--backup=<path>]) [--verbose] [--additional-config=<path>] [--check-graph=<true/false>] [--check-indexes=<true/false>] [--check-index-structure=<true/false>] [--check-label-scan-store=<true/false>] [--check-property-owners=<true/false>] [--report-dir=<path>]
```

Options

Option	Default	Description
--database	neo4j	Name of database.
--backup		Path to backup to check consistency of. Cannot be used together with --database.
--additional-config		Configuration file to supply additional configuration in.
--verbose	false	Enable verbose output.
--report-dir	.	Directory to write report file in.
--check-graph	true	Perform checks between nodes, relationships, properties, types and tokens.
--check-indexes	true	Perform checks on indexes by comparing content with the store.
--check-index-structure	true	Perform physical structure check on indexes. No comparison with the store takes place.
--check-label-scan-store	true	Perform checks on the label scan store.

Option	Default	Description
--check-property-owners	false	Perform additional checks on property ownership. This check is **very** expensive in time and memory.

Limitations

The consistency checker cannot be used with a database which is currently in use. If used with a running database, it will stop and print an error.

Output

If the consistency checker does not find errors, it will exit cleanly and not produce a report. If the consistency checker finds errors, it will exit with an exit code of `1` and write a report file with a name on the format `inconsistencies-YYYY-MM-DD.HH24.MI.SS.report`. The location of the report file is the current working directory, or as specified by the parameter `report-dir`.

Example 85. Run the consistency checker

Run with the `--database` option to check the consistency of a database. Note that the database must be stopped first.

```
$neo4j-home> bin/neo4j stop
$neo4j-home> bin/neo4j-admin check-consistency --database=neo4j

2019-11-13 12:42:14.479+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Selected
RecordFormat:StandardV4_0[SF4.0.b] record format from store /data/databases/neo4j
2019-11-13 12:42:14.481+0000 INFO [o.n.k.i.s.f.RecordFormatSelector] Format not configured for store
/data/databases/neo4j. Selected format from the store files: RecordFormat:StandardV4_0[SF4.0.b]
Index structure consistency check
....................  10%
....................  20%
....................  30%
....................  40%
....................  50%
....................  60%
....................  70%
....................  80%
....................  90%
.................... 100%
Full Consistency Check
....................  10%
....................  20%
....................  30%
....................  40%
....................  50%
....................  60%
....................  70%
....................  80%
....................  90%
.Checking node and relationship counts
....................  10%
....................  20%
....................  30%
....................  40%
....................  50%
....................  60%
....................  70%
....................  80%
....................  90%
.................... 100%
```

Run with the `--backup` option to check the consistency of a backup.

```
bin/neo4j-admin check-consistency --backup backup/neo4j-backup
```

14.3. Report tool

This chapter describes the `report` command of Neo4j Admin.

Use the `report` command of `neo4j-admin` to gather information about a Neo4j installation and save it to an archive.

```
neo4j-admin report [--force] [--list] [--verbose] [--pid=<pid>] [--to=<path>] [<classifier>…]
```

The intended usage of the report tool is to simplify the support process by collecting the relevant information in a standard way. This tool does not send any information automatically. To share this information with the Neo4j Support organization, you will have to send it manually.

Options

Option	Default	Description
--to	reports/	Specifies to target directory where the report should be written to.
--list		Will list available classifiers.
--verbose		Will instruct the tool to print more verbose output.
--force		Will disable the available disk space check.
--pid		Specify process id of running Neo4j instance. Only applicable when used together with classifiers indicated as *Online* in the table below.

By default, the tool tries to estimate the final size of the report and use that to assert that there is enough disk space available for it. If there is not enough available space the tool will abort. However, this estimation is pessimistic and does not take the compression into consideration, so if you are confident that you do have enough disk space, you can disable this check with the option `--force`.

Classifiers

Classifier	Online	Description
`all`		Include all of the available classifiers listed below.
`activetxs`	X	Include the output of `dbms.listTransactions()`.
`ccstate`		Include the current cluster state.
`config`		Include the `neo4j.conf` file.
`env`	X	Include a list of all environment variables.
`heap`	X	Include a heap dump.
`logs`		Include log files, e.g `debug.log`, `neo4j.log` etc.
`metrics`		Include the collected metrics.
`plugins`		Include a text view of the plugin directory (no files are collected).
`ps`		include a list of running processes.
`raft`		Include the raft log.

Classifier	Online	Description
`sysprop`	X	Include a list of Java system properties.
`threads`	X	Include a thread dump of the running instance.
`tree`		Include a text view of the folder structure of the data directory (no files are collected).
`tx`		Include transaction logs.

The classifiers indicated as *Online* only work when you have a running Neo4j instance that the tool can find.

If no classifiers are specified, the following classifiers will be used: `logs`, `config`, `plugins`, `tree`, `metrics`, `threads`, `env`, `sysprop` and `ps`.

The report tool does not read any data from your database. However, the heap, the raft logs, and the transaction logs may contain data. Additionally while the standard `neo4j.conf` file does not contain password information, for certain configurations it may have this type of information. Therefore, be aware of your organization’s data security rules before using the classifiers `heap`, `tx`, `raft` and `config`.

This tool uses the Java Attach API to gather data from a running Neo4j instance. Therefore, it requires the Java JDK in order to execute properly.

14.4. Display store information

This chapter describes the `store-info` command of Neo4j Admin.

The version of a Neo4j database can be displayed using the `store-info` command of `neo4j-admin`:

```
neo4j-admin store-info <path-to-store>
```

The `store` argument takes a path to a Neo4j database or backup. We will know if the store will need to go through a format migration as part of restoring a backup, by the presence of the message `Store format superseded in:` in the output.

The Neo4j Admin `store-info` command can not be used on a running database. The store is locked for protection while the database is running, and using the `store-info` command on it will fail with an error message indicating this.

Example 86. Example usage with older backup

You have an old backup of Neo4j a folder called */backups/graph-db.2016-11-11/* which you would like to restore. It is unclear what the version of Neo4j was at the time of backup. To find out, you would run:

```
$neo4j-home> bin/neo4j-admin store-info /backups/graph-db.2016-11-11
Store format version:         vE.H.0
Introduced in version:        3.0.0
Store format superseded in:   3.0.6
```

The output reveals that the database was configured to use the high-limit format, which is Enterprise Edition only. This means the backup can only be used with Neo4j Enterprise Edition.

The output also explains that the format was introduced in Neo4j 3.0.0, but a newer format was shipped in Neo4j 3.0.6. This means that a format migration must be performed if the backup is restored (see Upgrade for details).

Example 87. Example usage with newer backup

As with the previous example, let us assume that another backup of Neo4j is present in */backups/graph-db.2016-11-11/*. In this case the output is:

```
$neo4j-home> bin/neo4j-admin store-info /backups/graph-db.2016-11-11
Store format version:    v0.A.7
Introduced in version:   3.0.0
```

The output tells us that the backup is made with the standard store format, which all editions of Neo4j support. It is also the newest version of the format known to Neo4j. This means that no format migration is necessary for the backup to be restored.

14.5. Memory recommendations

This chapter describes the `memrec` command of Neo4j Admin.

Use the `memrec` command of `neo4j-admin` to get an initial recommendation on how to configure memory parameters for Neo4j:

```
neo4j-admin memrec [--memory=<memory dedicated to Neo4j>]
```

The recommendations will be given in a format such that it can be copied and pasted straight into *neo4j.conf*.

Options

Option	Default	Description
--memory	The memory capacity of the machine	The amount of memory to allocate to Neo4j. Valid units are: `k`, `K`, `m`, `M`, `g`, `G`.

Considerations

The `neo4j-admin memrec` command calculates a valid starting point for Neo4j memory settings, based on the provided memory. The specific conditions for your use case may warrant adjustment of these values. See Memory configuration for a description of the memory settings in Neo4j.

Example 88. Use the `memrec` command of `neo4j-admin`

The following example illustrates how `neo4j-admin memrec` provides a recommendation on how to use 16g of memory:

```
$neo4j-home> bin/neo4j-admin memrec --memory=16g

...
...
...
# Based on the above, the following memory settings are recommended:
dbms.memory.heap.initial_size=5g
dbms.memory.heap.max_size=5g
dbms.memory.pagecache.size=7g
```

14.6. Import

This section describes how to perform batch imports of data into Neo4j.

You can do batch imports of large amounts of data into a Neo4j database from CSV files, using the `import` command of `neo4j-admin`. This tool can only be used to load data into a previously unused database. By default, this database is set to `neo4j` but can be configured to other names and locations. If you wish to import small to medium-sized CSV files into an existing database, use `LOAD CSV` (see Cypher Manual `LOAD CSV`).

These are some things you will need to keep in mind when creating your input files:

- Fields are comma-separated by default but a different delimiter can be specified.
- All files must use the same delimiter.
- Multiple data sources can be used for both nodes and relationships.
- A data source can optionally be provided using multiple files.
- A separate file with a header that provides information on the data fields, must be the first specified file of each data source.
- Fields without corresponding information in the header will not be read.
- UTF-8 encoding is used.
- By default, the importer will trim extra whitespace at the beginning and end of strings. Quote your data to preserve leading and trailing whitespaces.

Indexes and constraints

Indexes and constraints are not created during the import. Instead, you will need to add these afterwards (see Cypher Manual Indexes).

This chapter explains how to format the input data and use the import tool. If you wish to see in-depth examples of using the import tool, refer to the Import tutorial.

The following sections describe how to use the import tool:

- Syntax - The syntax of the `neo4j-admin import` command.
- CSV header format - How to construct the header row of each CSV file.
- Options - The options available for use with `neo4j-admin import`.

14.6.1. Syntax

The syntax for importing a set of CSV files is:

```
neo4j-admin import [--verbose]
                   [--high-io[=<true/false>]]
                   [--cache-on-heap]
                   [--ignore-empty-strings[=<true/false>]]
                   [--ignore-extra-columns[=<true/false>]]
                   [--legacy-style-quoting[=<true/false>]]
                   [--multiline-fields[=<true/false>]]
                   [--normalize-types[=<true/false>]]
                   [--skip-bad-entries-logging[=<true/false>]]
                   [--skip-bad-relationships[=<true/false>]]
                   [--skip-duplicate-nodes[=<true/false>]]
                   [--trim-strings[=<true/false>]]
                   [--additional-config=<path>]
                   [--array-delimiter=<char>]
                   [--bad-tolerance=<num>]
                   [--database=<database>]
                   [--delimiter=<char>]
                   [--id-type=<idType>]
                   [--input-encoding=<character-set>]
                   [--max-memory=<size>]
                   [--processors=<num>]
                   [--quote=<char>]
                   [--read-buffer-size=<size>]
                   [--report-file=<path>]
                   --nodes=[<label>[:<label>]...=]<files>...
                   [--nodes=[<label>[:<label>]...=]<files>...]...
                   [--relationships=[<type>=]<files>...]...
```

Example 89. Import data from CSV files

Assume that we have formatted our data as per CSV file header format so that we have it in six different files: *movies_header.csv*, *movies.csv*, *actors_header.csv*, *actors.csv*, *roles_header.csv*, and *roles.csv*. The following command will import the three datasets:

```
neo4j_home$ bin/neo4j-admin import --nodes import/movies_header.csv,import/movies.csv \
--nodes import/actors_header.csv,import/actors.csv \
--relationships import/roles_header.csv,import/roles.csv
```

14.6.2. CSV file header format

This section explains the header format of CSV files when using the Neo4j import tool.

This section describes the following:

- Header files
- Properties
- Node files
- Relationship files
- Using ID spaces
- Skipping columns
- Compressed files

Header files

The header file of each data source specifies how the data fields should be interpreted. You must use the same delimiter for the header file and for the data files.

The header contains information for each field, with the format `<name>:<field_type>`. The `<name>` is used for properties and node IDs. In all other cases, the `<name>` part of the field is ignored.

Properties

For properties, the `<name>` part of the field designates the property key, while the `<field_type>` part assigns a data type (see below). You can have properties in both node data files and relationship data files.

Data types

Use one of `int`, `long`, `float`, `double`, `boolean`, `byte`, `short`, `char`, `string`, `point`, `date`, `localtime`, `time`, `localdatetime`, `datetime`, and `duration` to designate the data type for properties. If no data type is given, this defaults to `string`. To define an array type, append `[]` to the type. By default, array values are separated by `;`. A different delimiter can be specified with `--array-delimiter`. Boolean values are *true* if they match exactly the text `true`. All other values are *false*. Values that contain the delimiter character need to be escaped by enclosing in double quotation marks, or by using a different delimiter character with the `--delimiter` option.

Example 90. Header format with data types

This example illustrates several different data types specified in the CSV header.

```
:ID,name,joined:date,active:boolean,points:int
user01,Joe Soap,2017-05-05,true,10
user02,Jane Doe,2017-08-21,true,15
user03,Moe Know,2018-02-17,false,7
```

Special considerations for the `point` data type

A point is specified using the Cypher syntax for maps. The map allows the same keys as the input to the Cypher Manual → Point function. The point data type in the header can be amended with a map of default values used for all values of that column, e.g. `point{crs: 'WGS-84'}`. Specifying the header this way allows you to have an incomplete map in the value position in the data file. Optionally, a value in a data file may override default values from the header.

Example 91. Property format for `point` data type

This example illustrates various ways of using the `point` data type in the import header and the data files.

We are going to import the name and location coordinates for cities. First, we define the header as:

```
:ID,name,location:point{crs:WGS-84}
```

We then define cities in the data file.

- The first city's location is defined using `latitude` and `longitude`, as expected when using the coordinate system defined in the header.
- The second city uses `x` and `y` instead. This would normally lead to a point using the coordinate reference system `cartesian`. Since the header defines `crs:WGS-84`, that coordinate reference system will be used.
- The third city overrides the coordinate reference system defined in the header, and sets it explicitly to `WGS-84-3D`.

```
:ID,name,location:point{crs:WGS-84}
city01,"Malmö","{latitude:55.6121514, longitude:12.9950357}"
city02,"London","{y:51.507222, x:-0.1275}"
city03,"San Mateo","{latitude:37.554167, longitude:-122.313056, height: 100, crs:'WGS-84-3D'}"
```

Note that all point maps are within double quotation marks `"` in order to prevent the enclosed `,` character from being interpreted as a column separator. An alternative approach would be to use `--delimiter='\t'` and reformat the file with tab separators, in which case the `"` characters are not required.

```
:ID name    location:point{crs:WGS-84}
city01  Malmö   {latitude:55.6121514, longitude:12.9950357}
city02  London  {y:51.507222, x:-0.1275}
city03  San Mateo   {latitude:37.554167, longitude:-122.313056, height: 100, crs:'WGS-84-3D'}
```

Special considerations for temporal data types

The format for all temporal data types must be defined as described in Cypher Manual → Temporal instants syntax and Cypher Manual → Durations syntax. Two of the temporal types, *Time* and *DateTime*, take a time zone parameter which might be common between all or many of the values in the data file. It is therefor possible to specify a default time zone for *Time* and *DateTime* values in the header, for example: `time{timezone:+02:00}` and: `datetime{timezone:Europe/Stockholm}`. If no default time zone is specified, the default timezone is determined by the `db.temporal.timezone` configuration setting. The default time zone can be explicitly overridden in the values in the data file.

Example 92. Property format for temporal data types

This example illustrates various ways of using the `datetime` data type in the import header and the data files.

First, we define the header with two *DateTime* columns. The first one defines a time zone, but the second one does not:

```
:ID,date1:datetime{timezone:Europe/Stockholm},date2:datetime
```

We then define dates in the data file.

- The first row has two values that do not specify an explicit timezone. The value for `date1` will use the `Europe/Stockholm` time zone that was specified for that field in the header. The value for `date2` will use the configured default time zone of the database.
- In the second row, both `date1` and `date2` set the time zone explicitly to be `Europe/Berlin`. This overrides the header definition for `date1`, as well as the configured default time zone of the database.

```
1,2018-05-10T10:30,2018-05-10T12:30
2,2018-05-10T10:30[Europe/Berlin],2018-05-10T12:30[Europe/Berlin]
```

Node files

Files containing node data can have an `ID` field, a `LABEL` field as well as properties.

ID

Each node must have a unique ID if it is to be connected by any relationships created in the import. The IDs are used to find the correct nodes when creating relationships. Note that the ID has to be unique across all nodes in the import; even for nodes with different labels. The unique ID can be persisted in a property whose name is defined by the `<name>` part of the field definition `<name>:ID`. If no such property name is defined, the unique ID will be used for the purpose of the import but not be available for reference later. If no ID is specified, the node will be imported but it will not be able to be connected by any relationships during the import.

LABEL

Read one or more labels from this field. Like array values, multiple labels are separated by `;`, or by the character specified with `--array-delimiter`.

Example 93. Define nodes files

We define the headers for movies in the *movies_header.csv* file. Movies have the properties `movieId`, `year` and `title`. We also specify a field for labels.

```
movieId:ID,title,year:int,:LABEL
```

We define three movies in the *movies.csv* file. They contain all the properties defined in the header file. All the movies are given the label `Movie`. Two of them are also given the label `Sequel`.

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

Similarly, we also define three actors in the *actors_header.csv* and *actors.csv* files. They all have the properties `personId` and `name`, and the label `Actor`.

```
personId:ID,name,:LABEL
```

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

Relationship files

Files containing relationship data have three mandatory fields and can also have properties. The mandatory fields are:

TYPE
: The relationship type to use for this relationship.

START_ID
: The ID of the start node for this relationship.

END_ID
: The ID of the end node for this relationship.

The `START_ID` and `END_ID` refer to the unique node ID defined in one of the node data sources, as explained in the previous section. None of these takes a name, e.g. if `<name>:START_ID` or `<name>:END_ID` is defined, the `<name>` part will be ignored.

Example 94. Define relationships files

In this example we assume that the two nodes files from the previous example are used together with the following relationships file.

We define relationships between actors and movies in the files *roles_header.csv* and *roles.csv*. Each row connects a start node and an end node with a relationship of relationship type `ACTED_IN`. Notice how we use the unique identifiers `personId` and `movieId` from the nodes files above. The name of character that the actor is playing in this movie is stored as a `role` property on the relationship.

```
:START_ID,role,:END_ID,:TYPE
```

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

Using ID spaces

By default, the import tool assumes that node identifiers are unique across node files. In many cases the ID is only unique across each entity file, for example when our CSV files contain data extracted from a relational database and the ID field is pulled from the primary key column in the corresponding table. To handle this situation we define *ID spaces*. ID spaces are defined in the `ID` field of node files using the syntax `ID(<ID space identifier>)`. To reference an ID of an ID space in a relationship file, we use the syntax `START_ID(<ID space identifier>)` and `END_ID(<ID space identifier>)`.

Example 95. Define and use ID spaces

Define a `Movie-ID` ID space in the *movies_header.csv* file.

```
movieId:ID(Movie-ID),title,year:int,:LABEL
```

```
1,"The Matrix",1999,Movie
2,"The Matrix Reloaded",2003,Movie;Sequel
3,"The Matrix Revolutions",2003,Movie;Sequel
```

Define an `Actor-ID` ID space in the header of the *actors_header.csv* file.

```
personId:ID(Actor-ID),name,:LABEL
```

```
1,"Keanu Reeves",Actor
2,"Laurence Fishburne",Actor
3,"Carrie-Anne Moss",Actor
```

Now use the previously defined ID spaces when connecting the actors to movies.

```
:START_ID(Actor-ID),role,:END_ID(Movie-ID),:TYPE
```

```
1,"Neo",1,ACTED_IN
1,"Neo",2,ACTED_IN
1,"Neo",3,ACTED_IN
2,"Morpheus",1,ACTED_IN
2,"Morpheus",2,ACTED_IN
2,"Morpheus",3,ACTED_IN
3,"Trinity",1,ACTED_IN
3,"Trinity",2,ACTED_IN
3,"Trinity",3,ACTED_IN
```

Skipping columns

IGNORE

If there are fields in the data that we wish to ignore completely, this can be done using the `IGNORE` keyword in the header file. `IGNORE` must be prepended with a `:`.

Example 96. Skip a column

In this example, we are not interested in the data in the third column of the nodes file and wish to skip over it. Note that the `IGNORE` keyword is prepended by a `:`.

```
personId:ID,name,:IGNORE,:LABEL
```

```
keanu,"Keanu Reeves","male",Actor
laurence,"Laurence Fishburne","male",Actor
carrieanne,"Carrie-Anne Moss","female",Actor
```

If all your superfluous data is placed in columns located to the right of all the columns that you wish to import, you can instead use the command line option `--ignore-extra-columns`.

Import compressed files

The import tool can handle files compressed with `zip` or `gzip`. Each compressed file must contain a single file.

Example 97. Perform an import using compressed files

```
neo4j_home$ ls import
actors-header.csv  actors.csv.zip  movies-header.csv  movies.csv.gz  roles-header.csv  roles.csv.gz
neo4j_home$ bin/neo4j-admin import --nodes import/movies-header.csv,import/movies.csv.gz --nodes
import/actors-header.csv,import/actors.csv.zip --relationships import/roles-
header.csv,import/roles.csv.gz
```

14.6.3. Options

This section describes in details the options available when using the Neo4j import tool to import data from CSV files.

Some of the options below are marked as **Advanced**. These options should not be used for experimentation.

For more information, please contact Neo4j Professional Services.

`--verbose`
Enable verbose output.

`--database=<name>`
Name of database.

Default: `neo4j`

`--additional-config=<config-file-path>`
Configuration file with which to supply additional configuration.

`--report-file=<filename>`
File in which to store the report of the csv-import.

Default: `import.report`

`--id-type=<STRING|INTEGER|ACTUAL>`
Each node must provide a unique ID in order to be used for creating relationships during the import.

Possible values are:

- `STRING`: arbitrary strings for identifying nodes,
- `INTEGER`: arbitrary integer values for identifying nodes,
- `ACTUAL`: (Advanced) actual node IDs.

Default: `STRING`

`--input-encoding=<character-set>`
Character set that input data is encoded in.

Default: `UTF-8`

`--ignore-extra-columns[=<true/false>]`

If unspecified columns should be ignored during the import.

Default: `false`

`--multiline-fields[=<true/false>]`

Determines whether or not fields from input source can span multiple lines, i.e. contain newline characters.

Setting `--multiline-fields=true` can severely degrade performance of the importer. Therefore, use it with care, especially with large imports.

Default: `false`

`--ignore-empty-strings[=<true/false>]`

Determines whether or not empty string fields, such as "", from input source are ignored (treated as null).

Default: `false`

`--trim-strings[=<true/false>]`

Determines whether or not strings should be trimmed for whitespaces.

Default: `false`

`--legacy-style-quoting[=<true/false>]`

Determines whether or not backslash-escaped quote e.g. \" is interpreted as inner quote.

Default: `false`

`--delimiter=<char>`

Delimiter character between values in CSV data.

Unicode character encoding can be used if prepended by `\`. For example, `\44` is equivalent to `,`.

Default: `,`

`--array-delimiter=<char>`

Delimiter character between array elements within a value in CSV data.

Unicode character encoding can be used if prepended by `\`. For example, `\59` is equivalent to `;`.

Default: `;`

`--quote=<char>`

Character to treat as quotation character for values in CSV data.

Quotes can be escaped as per RFC 4180 by doubling them, for example `""` would be interpreted as a literal `"`.

You cannot escape using `\`.

`--read-buffer-size=<size>`

Size of each buffer for reading input data.

It has to at least be large enough to hold the biggest single value in the input data. Value can be a plain number or byte units string, e.g. `128k`, `1m`.

Default: `4m`

`--max-memory=<size>`
: Maximum memory that `neo4j-admin` can use for various data structures and caching to improve performance.

  Values can be plain numbers such as `10000000`, or `20G` for 20 gigabyte. It can also be specified as a percentage of the available memory, for example `70%`.

  Default: `90%`

`--high-io[=<true/false>]`
: Ignore environment-based heuristics, and specify whether the target storage subsystem can support parallel IO with high throughput.

  Typically this is `true` for SSDs, large raid arrays and network-attached storage.

`--cache-on-heap[=<true/false>]` **Advanced**
: Determines whether or not to allow allocating memory for the cache on heap.

  If `false`, then caches will still be allocated off-heap, but the additional free memory inside the JVM will not be allocated for the caches.

  Use this to have better control over the heap memory.

  Default: `false`

`--processors=<num>` **Advanced**
: Max number of processors used by the importer.

  Defaults to the number of available processors reported by the JVM. There is a certain amount of minimum threads needed, so for that reason there is no lower bound for this value.

  For optimal performance, this value shouldn't be greater than the number of available processors.

`--bad-tolerance=<num>`
: Number of bad entries before the import is considered failed.

  This tolerance threshold is about relationships referring to missing nodes. Format errors in input data are still treated as errors.

  Default: `1000`

`--skip-bad-entries-logging[=<true/false>]`
: Determines whether or not to skip logging bad entries detected during import.

  Default: `false`

`--skip-bad-relationships[=<true/false>]`
: Determines whether or not to skip importing relationships that refer to missing node IDs, i.e. either start or end node ID/group referring to node that wasn't specified by the node input data.

  Skipped nodes will be logged, containing at most the number of entities specified by bad-tolerance, unless otherwise specified by the skip-bad-entries-logging option.

Default: `false`

`--skip-duplicate-nodes[=<true/false>]`

Determines whether or not to skip importing nodes that have the same ID/group.

In the event of multiple nodes within the same group having the same ID, the first encountered will be imported, whereas consecutive such nodes will be skipped.

Skipped nodes will be logged, containing at most the number of entities specified by bad-tolerance, unless otherwise specified by the skip-bad-entries-logging option.

Default: `false`

`--normalize-types[=<true/false>]`

Determines whether or not to normalize property types to Cypher types, e.g. `int` becomes `long` and `float` becomes `double`.

Default: `true`

`--nodes=[<label>[:<label>]…=]<files>…`

Node CSV header and data.

- Multiple files will be logically seen as one big file from the perspective of the importer.
- The first line must contain the header.
- Multiple data sources like these can be specified in one import, where each data source has its own header.
- Files can also be specified using regular expressions.

For an example, see Using regular expressions for specifying multiple input files.

`--relationships=[<type>=]<files>…`

Relationship CSV header and data.

- Multiple files will be logically seen as one big file from the perspective of the importer.
- The first line must contain the header.
- Multiple data sources like these can be specified in one import, where each data source has its own header.
- Files can also be specified using regular expressions.

For an example, see Using regular expressions for specifying multiple input files.

`@<arguments-file-path>`

File containing all arguments, used as an alternative to supplying all arguments on the command line directly.

Each argument can be on a separate line, or multiple arguments per line and separated by space.

Arguments containing spaces must be quoted.

Heap size for the import

You want to set the maximum heap size to a relevant value for the import. This is done by defining the `HEAP_SIZE` environment parameter before starting the import. For example, 2G is an appropriate value for smaller imports.

If doing imports in the order of magnitude of 100 billion entities, 20G will be an appropriate value.

Record format

If your import data will result in a graph that is larger than 34 billion nodes, 34 billion relationships, or 68 billion properties you will need to configure the importer to use the high limit record format. This is achieved by setting the parameter `dbms.record_format=high_limit` in a configuration file, and supplying that file to the importer with `--additional-config`.

The `high_limit` format is available for Enterprise Edition only.

Output

The location of the import log file can be controlled using the --report-file option. If you run large imports of CSV files that have low data quality, the import log file can grow very large. For example, CSV files that contain duplicate node IDs, or that attempt to create relationships between non-existent nodes, could be classed as having low data quality. In these cases, you may wish to direct the output to a location that can handle the large log file.

If you are running on a UNIX-like system and you are not interested in the output, you can get rid of it altogether by directing the report file to `/dev/null`.

If you need to debug the import, it might be useful to collect the stack trace. This is done by using `--verbose` option.

14.7. Dump and load databases

This section describes the dump and load commands of Neo4j Admin.

A Neo4j database can be dumped and loaded using the following commands:

`neo4j-admin dump --database=<database> --to=<destination-path>`

`neo4j-admin load --from=<archive-path> --database=<database> [--force]`

Limitations

- The database should be shutdown before running the `dump` and `load` commands.
- `neo4j-admin` must be invoked as the `neo4j` user in order to ensure the appropriate file permissions.

Usage

Using the `dump` and `load` commands are the recommended, and safe, way of transferring databases between environments. They understand which files need to be exported and imported and which should not.

+ By contrast, file system copy-and-paste of databases is not supported.

Examples

Example 98. Use the `dump` command of neo4j-admin

Dump the database called `neo4j` into a file called `/backups/neo4j/2016-10-02.dump`. The destination directory for the dump file — in this case */backups/neo4j* — must exist before calling the command.

```
$neo4j-home> bin/neo4j-admin dump --database=neo4j --to=/backups/neo4j/2016-10-02.dump

$neo4j-home> ls /backups/neo4j

$neo4j-home> 2016-10-02.dump
```

Example 99. Use the `load` command of neo4j-admin

Load the backed-up database contained in the file `/backups/neo4j/2016-10-02.dump` into database `neo4j`. If you have a database running, it needs to be shutdown first. When we use the `--force` option, any existing database gets overwritten.

```
$neo4j-home> bin/neo4j-admin load --from=/backups/neo4j/2016-10-02.dump --database=neo4j --force
```

If using the `load` command to seed a Causal Cluster, you must first perform `neo4j-admin unbind` on each of the cluster instances. The procedure is described in Seed from backups.

14.8. Unbind a Core Server

This section describes how to remove cluster state data for a Neo4j server.

The cluster state of a Causal Cluster member can be removed by using the following command:

`neo4j-admin unbind`

Limitations

The Neo4j server process should be shutdown before running the `unbind` command.

Examples of usage

- When transforming a Causal Cluster member to a standalone server:

  The `unbind` command can be used to turn a Causal Cluster server into a standalone server. To start the database in single (standalone) mode after unbinding it from the cluster, first set `dbms.mode=SINGLE` in *neo4j.conf*.

- When seeding a Causal Cluster with existing store files:

  If you wish to seed a new Causal Cluster using the store files of a previous cluster, you must first run `neo4j-admin unbind` on each server. For more information about seeding Causal Clusters, see Seed a cluster.

- When recovering a Causal Cluster:

  In the event of serious failures you may need to recover an entire Causal Cluster from backups.

Before restoring those backups, you must first run `neo4j-admin unbind` on each server. For more information about recovering databases from online backups, see Restore a backup.

Unlike versions of Neo4j prior to v4.0.0, you **must** run the `unbind` command on both Read Replicas and Core members.

14.9. Copy a database

This chapter describes the `copy` command of Neo4j Admin.

The `copy` command of `neo4j-admin` is used to copy data from an existing database to a new database.

The syntax follows:

```
neo4j-admin copy --to-database=<database>
                (--from-database=<database> | --from-path=<path>)
                [--from-path-tx=<path>]
                [--to-format=<format>]
                [--delete-nodes-with-labels=<label>[,<label>...]]
                [--skip-labels=<label>[,<label>...]]
                [--skip-properties=<property>[,<property>...]]
                [--skip-relationships=<relationship>[,<relationship>...]]`
                [--force]
                [--verbose]
```

The existing database must be stopped before copying from it, and the destination database must not yet exist.

The `copy` command can process an optional set of filters. This can be used to remove data that is unwanted in the destination database.

The schema definitions, i.e. index and constraint, are not automatically transferred. However, they will be extracted and presented as Cypher statements so you can recreate the ones you want.

Options

Option	Description
`--from-database`	The database name to copy from. Will assume the database is in the configured location.
`--from-path`	The path to the database to copy from. It can be used to target databases outside of the installation, e.g. backups.
`--from-path-tx`	The path to the transaction log files. You only need to use this if the command is unable to determine where they are located.
`--to-database`	The destination database name.

Option	Description
`--to-format`	The store format of the destination database. Valid values are `same`, `standard`, `high_limit`. The default value for this option is the format of the source database.
`--delete-nodes-with-labels`	A list of labels. Any node matching any of the labels will be ignored during copy.
`--skip-properties`	A list of property keys to ignore during the copy.
`--skip-labels`	A list of labels to ignore during the copy.
`--skip-relationships`	A list of relationship types to ignore during the copy.
`--verbose`	Will instruct the tool to print more verbose output.
`--force`	Will force the command to proceed even if the integrity of the database can not be verified.

Due to the way filters are processed, the `id` of the node might change. This is true even if no filters are specified.

Examples

Example 100. Use the `copy` command to take a copy of the database `neo4j`.

To begin, you must stop the database named `neo4j`; this can be done by issuing the following Cypher statement:

```
STOP DATABASE neo4j
```

You can now copy the data from `neo4j`, to a new database called `copy`:

```
$neo4j-home> bin/neo4j-admin copy --from-database=neo4j --to-database=copy
```

A new database with the name `copy` now exists on the server, but it is not automatically picked up by Neo4j. To start the new database you have to insert it into Neo4j with the following Cypher query:

```
CREATE DATABASE copy
```

Neo4j will then detect the copied database and begin to use that.

Remember to start the database you copied from, if you still want it, using:

```
START DATABASE neo4j
```

The console output is saved to *logs/neo4j-admin-copy-<date>.log*.

Example 101. Use the `copy` command with filters.

The command can perform some basic forms of processing. You can remove nodes, labels, properties, and/or relationships.

The difference with `--skip-labels` and `--delete-nodes-with-labels` is that `--skip-labels` will just remove the labels, potentially leaving nodes without any labels.

```
$neo4j-home> bin/neo4j-admin copy --from-database=neo4j --to-database=copy --delete-nodes-with-
labels="Cat,Dog"
```

After this command, you will have a copy of the database `neo4j`, without nodes with the labels `:Cat` and `:Dogs`.

Labels are processed independently, i.e. the filter described above will delete any node with either `:Cat` or `:Dogs`, and not only nodes that have both of the labels.

14.10. Cypher Shell

This section describes Neo4j Cypher Shell.

This section describes the following:

* About Cypher Shell
* Syntax
* Query parameters
* Transactions
* Procedures

14.10.1. About Cypher Shell

Cypher Shell is a command-line tool that comes with the Neo4j installation. It can also be downloaded from Neo4j Download Center and installed separately.

Cypher Shell is used to run queries and perform administrative tasks. It communicates via the encrypted binary protocol Bolt.

14.10.2. Syntax

Cypher Shell is located in the `bin` directory if installed as part of the product. The syntax is:

```
cypher-shell [-h] [-a ADDRESS] [-u USERNAME] [-p PASSWORD] [--encryption {true,false}] [{-d
DATABASE,--database DATABASE}] [--fail-fast | --fail-at-end] [--format {verbose,plain}] [{--P
PARAM,--param PARAM}] [--debug] [--non-interactive] [--sample-rows SAMPLE-ROWS] [--wrap
{true,false}] [-v] [--driver-version] [{-f FILE,--file FILE}] [cypher]
```

Arguments

Positional arguments:	
cypher	An optional string of Cypher to execute and then exit.
Optional arguments:	
-h, --help	Show help message and exit.
--fail-fast	Exit and report failure on first error when reading from file (this is the default behavior).
--fail-at-end	Exit and report failures at end of input when reading from file.
--format {auto,verbose,plain}	Desired output format. `auto` displays results in tabular format if you use the shell interactively and with minimal formatting if you use it for scripting. `verbose` displays results in tabular format and prints statistics and `plain` displays data with minimal formatting (default: auto).
--P PARAM, --param PARAM	Add a parameter to this session. Example: `-P "number => 3"`. This argument can be specified multiple times.
--debug	Print additional debug information (default: false).
--non-interactive	Force non-interactive mode; only useful if auto-detection fails (default: false).
--sample-rows SAMPLE-ROWS	Number of rows sampled to compute table widths (only for format=VERBOSE) (default: 1000).
--wrap {true,false}	Wrap table column values if column is too narrow (only for format=VERBOSE) (default: true).
-v, --version	Print version of cypher-shell and exit (default: false).
--driver-version	Print version of the Neo4j Driver used and exit (default: false).
-f FILE, --file FILE	Pass a file with Cypher statements to be executed. After the statements have been executed cypher-shell will be shut down.

Connection arguments:	
-a ADDRESS, --address ADDRESS	Address and port to connect to (default: bolt://localhost:7687).
-u USERNAME, --username USERNAME	Username to connect as. Can also be specified using environment variable NEO4J_USERNAME (default:).
-p PASSWORD, --password PASSWORD	Password to connect with. Can also be specified using environment variable NEO4J_PASSWORD (default:).
--encryption {true,false}	Whether the connection to Neo4j should be encrypted; must be consistent with Neo4j's configuration (default: true).
-d DATABASE, --database DATABASE	Database to connect to. Can also be specified using environment variable NEO4J_DATABASE (default:).

Example 102. Invoke Cypher Shell with username and password

```
$neo4j-home> bin/cypher-shell -u johndoe -p secret
```

```
Connected to Neo4j 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>
```

Example 103. Invoke help from within Cypher Shell

```
neo4j> :help
```

```
Available commands:
  :begin    Open a transaction
  :commit   Commit the currently open transaction
  :exit     Exit the logger
  :help     Show this help message
  :history  Print a list of the last commands executed
  :param    Set the value of a query parameter
  :params   Prints all currently set query parameters and their values
  :rollback Rollback the currently open transaction
  :source   Interactively executes Cypher statements from a file
  :use      Set the active database

For help on a specific command type:
    :help command
```

Example 104. Execute a query from within Cypher Shell

```
neo4j> MATCH (n) RETURN n;
```

```
+-----------------------------------------------------------------+
| n                                                               |
+-----------------------------------------------------------------+
| (:Person {name: "Bruce Wayne", alias: "Batman"})                |
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]}) |
+-----------------------------------------------------------------+
```

Example 105. Invoke Cypher Shell with a Cypher script from the command line

Below is the contents of a file called examples.cypher:

```
MATCH (n) RETURN n;

MATCH (batman:Person {name: 'Bruce Wayne'}) RETURN batman;
```

Invoke the 'examples.cypher' script from the command-line. All the examples in the remainder of this section will use the `--format plain` flag for a simple output.

```
$neo4j-home> cat examples.cypher | bin/cypher-shell -u neo4j -p secret --format plain
```

```
n
(:Person {name: "Bruce Wayne", alias: "Batman"})
(:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]})
batman
(:Person {name: "Bruce Wayne", alias: "Batman"})
```

14.10.3. Query parameters

Cypher Shell supports querying based on parameters. This is often used while scripting.

Example 106. Use parameters within Cypher Shell

Set the parameter 'thisAlias' to 'Robin' using the ':param' keyword. Check the parameter using the ':params' keyword.

```
neo4j> :param thisAlias => 'Robin'
neo4j> :params
:param thisAlias => 'Robin'
```

Now use the parameter 'thisAlias' in a Cypher query. Verify the result.

```
neo4j> CREATE (:Person {name : 'Dick Grayson', alias : {thisAlias} });
Added 1 nodes, Set 2 properties, Added 1 labels
neo4j> MATCH (n) RETURN n;
+-----------------------------------------------------------------+
| n                                                               |
+-----------------------------------------------------------------+
| (:Person {name: "Bruce Wayne", alias: "Batman"})                |
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]}) |
| (:Person {name: "Dick Grayson", alias: "Robin"})                |
+-----------------------------------------------------------------+
3 rows available after 2 ms, consumed after another 2 ms
```

14.10.4. Transactions

Cypher Shell supports explicit transactions. Transaction states are controlled using the keywords `:begin`, `:commit`, and `:rollback`:

Example 107. Use fine-grained transaction control

Start a transaction in your first Cypher Shell session:

```
neo4j> MATCH (n) RETURN n;
+------------------------------------------------------------------+
| n                                                                |
+------------------------------------------------------------------+
| (:Person {name: "Bruce Wayne", alias: "Batman"})                 |
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]}) |
| (:Person {name: "Dick Grayson", alias: "Robin"})                 |
+------------------------------------------------------------------+
3 rows available after 2 ms, consumed after another 2 ms
neo4j> :begin
neo4j# CREATE (:Person {name : 'Edward Mygma', alias : 'The Riddler' });
```

If you now open up a second Cypher Shell session, you will notice no changes from the latest `CREATE` statement:

```
neo4j> MATCH (n) RETURN n;
+------------------------------------------------------------------+
| n                                                                |
+------------------------------------------------------------------+
| (:Person {name: "Bruce Wayne", alias: "Batman"})                 |
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]}) |
| (:Person {name: "Dick Grayson", alias: "Robin"})                 |
+------------------------------------------------------------------+
3 rows available after 2 ms, consumed after another 2 ms
```

Go back to the first session and commit the transaction:

```
neo4j# :commit
0 rows available after 1 ms, consumed after another 0 ms
Added 1 nodes, Set 2 properties, Added 1 labels
neo4j> MATCH (n) RETURN n;
+------------------------------------------------------------------+
| n                                                                |
+------------------------------------------------------------------+
| (:Person {name: "Bruce Wayne", alias: "Batman"})                 |
| (:Person {name: "Selina Kyle", alias: ["Catwoman", "The Cat"]}) |
| (:Person {name: "Dick Grayson", alias: "Robin"})                 |
| (:Person {name: "Edward Mygma", alias: "The Riddler"})           |
+------------------------------------------------------------------+
4 rows available after 1 ms, consumed after another 1 ms

neo4j>
```

14.10.5. Procedures

Cypher Shell supports running any procedures for which the current user is authorized. Here, we are using the natively built-in procedure `dbms.showCurrentUser()`.

Example 108. Call a procedure from within Cypher Shell

```
neo4j> CALL dbms.showCurrentUser();
+-------------------------------+
| username | roles     | flags |
+-------------------------------+
| "neo4j"  | ["admin"] | []    |
+-------------------------------+

1 row available after 66 ms, consumed after another 2 ms
neo4j> :exit
```

Appendix A: Reference

This appendix contains the Neo4j configuration settings reference, the list of built-in procedures bundled with Neo4j, and a description of user management for Community Edition.

This appendix contains the following:

- Configuration settings
- Built-in procedures

A.1. Configuration settings

This section contains a complete reference of Neo4j configuration settings. They can be set in neo4j.conf. Refer to The neo4j.conf file for details on how to use configuration settings.

All settings

- browser.allow_outgoing_connections: Configure the policy for outgoing Neo4j Browser connections.
- browser.credential_timeout: Configure the Neo4j Browser to time out logged in users after this idle period.
- browser.post_connect_cmd: Commands to be run when Neo4j Browser successfully connects to this server.
- browser.remote_content_hostname_whitelist: Whitelist of hosts for the Neo4j Browser to be allowed to fetch content from.
- browser.retain_connection_credentials: Configure the Neo4j Browser to store or not store user credentials.
- causal_clustering.catch_up_client_inactivity_timeout: The catch up protocol times out if the given duration elapses with no network activity.
- causal_clustering.catchup_batch_size: The maximum batch size when catching up (in unit of entries).
- causal_clustering.cluster_allow_reads_on_followers: Configure if the `dbms.routing.getRoutingTable()` procedure should include followers as read endpoints or return only read replicas.
- causal_clustering.cluster_topology_refresh: Time between scanning the cluster to refresh current server's view of topology.
- causal_clustering.connect-randomly-to-server-group: Comma separated list of groups to be used by the connect-randomly-to-server-group selection strategy.
- causal_clustering.discovery_advertised_address: Advertised cluster member discovery management communication.
- causal_clustering.discovery_listen_address: Host and port to bind the cluster member discovery management communication.
- causal_clustering.discovery_type: Configure the discovery type used for cluster name resolution.
- causal_clustering.enable_pre_voting: Enable pre-voting extension to the Raft protocol (this is breaking and must match between the core cluster members).
- causal_clustering.global_session_tracker_state_size: The maximum file size before the global session tracker state file is rotated (in unit of entries).

- causal_clustering.handshake_timeout: Time out for protocol negotiation handshake.
- causal_clustering.in_flight_cache.max_bytes: The maximum number of bytes in the in-flight cache.
- causal_clustering.in_flight_cache.max_entries: The maximum number of entries in the in-flight cache.
- causal_clustering.in_flight_cache.type: Type of in-flight cache.
- causal_clustering.initial_discovery_members: A comma-separated list of other members of the cluster to join.
- causal_clustering.join_catch_up_timeout: Time out for a new member to catch up.
- causal_clustering.kubernetes.address: Address for Kubernetes API.
- causal_clustering.kubernetes.ca_crt: File location of CA certificate for Kubernetes API.
- causal_clustering.kubernetes.label_selector: LabelSelector for Kubernetes API.
- causal_clustering.kubernetes.namespace: File location of namespace for Kubernetes API.
- causal_clustering.kubernetes.service_port_name: Service port name for discovery for Kubernetes API.
- causal_clustering.kubernetes.token: File location of token for Kubernetes API.
- causal_clustering.last_applied_state_size: The maximum file size before the storage file is rotated (in unit of entries).
- causal_clustering.leader_election_timeout: The time limit within which a new leader election will occur if no messages are received.
- causal_clustering.load_balancing.plugin: The load balancing plugin to use.
- causal_clustering.load_balancing.shuffle: Enables shuffling of the returned load balancing result.
- causal_clustering.log_shipping_max_lag: The maximum lag allowed before log shipping pauses (in unit of entries).
- causal_clustering.middleware.logging.level: The level of middleware logging.
- causal_clustering.minimum_core_cluster_size_at_formation: Minimum number of Core machines initially required to form a cluster.
- causal_clustering.minimum_core_cluster_size_at_runtime: The minimum size of the dynamically adjusted voting set (which only core members may be a part of).
- causal_clustering.multi_dc_license: Enable multi-data center features.
- causal_clustering.protocol_implementations.catchup: Catchup protocol implementation versions that this instance will allow in negotiation as a comma-separated list.
- causal_clustering.protocol_implementations.compression: Network compression algorithms that this instance will allow in negotiation as a comma-separated list.
- causal_clustering.protocol_implementations.raft: Raft protocol implementation versions that this instance will allow in negotiation as a comma-separated list.
- causal_clustering.pull_interval: Interval of pulling updates from cores.
- causal_clustering.raft_advertised_address: Advertised hostname/IP address and port for the RAFT server.
- causal_clustering.raft_in_queue_max_batch_bytes: Largest batch processed by RAFT in bytes.
- causal_clustering.raft_in_queue_max_bytes: Maximum number of bytes in the RAFT in-queue.
- causal_clustering.raft_listen_address: Network interface and port for the RAFT server to listen on.
- causal_clustering.raft_log_implementation: RAFT log implementation.
- causal_clustering.raft_log_prune_strategy: RAFT log pruning strategy.
- causal_clustering.raft_log_pruning_frequency: RAFT log pruning frequency.

- causal_clustering.raft_log_reader_pool_size: RAFT log reader pool size.
- causal_clustering.raft_log_rotation_size: RAFT log rotation size.
- causal_clustering.raft_membership_state_size: The maximum file size before the membership state file is rotated (in unit of entries).
- causal_clustering.raft_term_state_size: The maximum file size before the term state file is rotated (in unit of entries).
- causal_clustering.raft_vote_state_size: The maximum file size before the vote state file is rotated (in unit of entries).
- causal_clustering.refuse_to_be_leader: Prevents the current instance from volunteering to become Raft leader.
- causal_clustering.replicated_lease_state_size: The maximum file size before the replicated lease state file is rotated (in unit of entries).
- causal_clustering.replication_leader_await_timeout: The duration for which the replicator will await a new leader.
- causal_clustering.replication_retry_timeout_base: The initial timeout until replication is retried.
- causal_clustering.replication_retry_timeout_limit: The upper limit for the exponentially incremented retry timeout.
- causal_clustering.server_groups: A list of group names for the server used when configuring load balancing and replication policies.
- causal_clustering.state_machine_apply_max_batch_size: The maximum number of operations to be batched during applications of operations in the state machines.
- causal_clustering.state_machine_flush_window_size: The number of operations to be processed before the state machines flush to disk.
- causal_clustering.status_throughput_window: Sampling window for throughput estimate reported in the status endpoint.
- causal_clustering.store_copy_chunk_size: Store copy chunk size.
- causal_clustering.store_copy_max_retry_time_per_request: Maximum retry time per request during store copy.
- causal_clustering.transaction_advertised_address: Advertised hostname/IP address and port for the transaction shipping server.
- causal_clustering.transaction_listen_address: Network interface and port for the transaction shipping server to listen on.
- causal_clustering.unknown_address_logging_throttle: Throttle limit for logging unknown cluster member address.
- causal_clustering.upstream_selection_strategy: An ordered list in descending preference of the strategy which read replicas use to choose the upstream server from which to pull transactional updates.
- causal_clustering.user_defined_upstream_strategy: Configuration of a user-defined upstream selection strategy.
- cypher.default_language_version: Set this to specify the default parser (language version).
- cypher.forbid_exhaustive_shortestpath: This setting is associated with performance optimization.
- cypher.forbid_shortestpath_common_nodes: This setting is associated with performance optimization.
- cypher.hints_error: Set this to specify the behavior when Cypher planner or runtime hints cannot be fulfilled.
- cypher.lenient_create_relationship: Set this to change the behavior for Cypher create relationship when the start or end node is missing.

- cypher.min_replan_interval: The minimum time between possible cypher query replanning events.
- cypher.planner: Set this to specify the default planner for the default language version.
- cypher.query_max_allocations: The maximum amount of heap memory allocations to for cypher to perform on a single query, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g').
- cypher.statistics_divergence_threshold: The threshold when a plan is considered stale.
- db.temporal.timezone: Database timezone for temporal functions.
- dbms.allow_upgrade: Whether to allow an upgrade in case the current version of the database starts against an older version.
- dbms.backup.enabled: Enable support for running online backups.
- dbms.backup.listen_address: Network interface and port for the backup server to listen on.
- dbms.checkpoint: Configures the general policy for when check-points should occur.
- dbms.checkpoint.interval.time: Configures the time interval between check-points.
- dbms.checkpoint.interval.tx: Configures the transaction interval between check-points.
- dbms.checkpoint.iops.limit: Limit the number of IOs the background checkpoint process will consume per second.
- dbms.config.strict_validation: A strict configuration validation will prevent the database from starting up if unknown configuration options are specified in the neo4j settings namespace (such as dbms., cypher., etc).
- dbms.connector.bolt.advertised_address: Advertised address for this connector.
- dbms.connector.bolt.enabled: Enable the bolt connector.
- dbms.connector.bolt.listen_address: Address the connector should bind to.
- dbms.connector.bolt.thread_pool_keep_alive: The maximum time an idle thread in the thread pool bound to this connector will wait for new tasks.
- dbms.connector.bolt.thread_pool_max_size: The maximum number of threads allowed in the thread pool bound to this connector.
- dbms.connector.bolt.thread_pool_min_size: The number of threads to keep in the thread pool bound to this connector, even if they are idle.
- dbms.connector.bolt.tls_level: Encryption level to require this connector to use.
- dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time: The maximum time to wait for the thread pool to finish processing its pending jobs and shutdown.
- dbms.connector.http.advertised_address: Advertised address for this connector.
- dbms.connector.http.enabled: Enable the http connector.
- dbms.connector.http.listen_address: Address the connector should bind to.
- dbms.connector.https.advertised_address: Advertised address for this connector.
- dbms.connector.https.enabled: Enable the https connector.
- dbms.connector.https.listen_address: Address the connector should bind to.
- dbms.db.timezone: Database timezone.
- dbms.default_advertised_address: Default hostname or IP address the server uses to advertise itself.
- dbms.default_database: Name of the default database.
- dbms.default_listen_address: Default network interface to listen for incoming connections.
- dbms.directories.data: Path of the data directory.
- dbms.directories.import: Sets the root directory for file URLs used with the Cypher `LOAD CSV`

clause.

- dbms.directories.lib: Path of the lib directory.
- dbms.directories.logs: Path of the logs directory.
- dbms.directories.metrics: The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.
- dbms.directories.neo4j_home: Root relative to which directory settings are resolved.
- dbms.directories.plugins: Location of the database plugin directory.
- dbms.directories.run: Path of the run directory.
- dbms.directories.transaction.logs.root: Root location where Neo4j will store transaction logs for configured databases.
- dbms.dynamic.setting.whitelist: A list of setting name patterns (comma separated) that are allowed to be dynamically changed.
- dbms.filewatcher.enabled: Allows the enabling or disabling of the file watcher service.
- dbms.import.csv.buffer_size: The size of the internal buffer in bytes used by `LOAD CSV`.
- dbms.import.csv.legacy_quote_escaping: Selects whether to conform to the standard https://tools.ietf.org/html/rfc4180 for interpreting escaped quotation characters in CSV files loaded using `LOAD CSV`.
- dbms.index.default_schema_provider: Index provider to use for newly created schema indexes.
- dbms.index.fulltext.default_analyzer: The name of the analyzer that the fulltext indexes should use by default.
- dbms.index.fulltext.eventually_consistent: Whether or not fulltext indexes should be eventually consistent by default or not.
- dbms.index.fulltext.eventually_consistent_index_update_queue_max_length: The eventually_consistent mode of the fulltext indexes works by queueing up index updates to be applied later in a background thread.
- dbms.index_sampling.background_enabled: Enable or disable background index sampling.
- dbms.index_sampling.sample_size_limit: Index sampling chunk size limit.
- dbms.index_sampling.update_percentage: Percentage of index updates of total index size required before sampling of a given index is triggered.
- dbms.index_searcher_cache_size: The maximum number of open Lucene index searchers.
- dbms.jvm.additional: Additional JVM arguments.
- dbms.lock.acquisition.timeout: The maximum time interval within which lock should be acquired.
- dbms.logs.debug.level: Debug log level threshold.
- dbms.logs.debug.path: Path to the debug log file.
- dbms.logs.debug.rotation.delay: Minimum time interval after last rotation of the debug log before it may be rotated again.
- dbms.logs.debug.rotation.keep_number: Maximum number of history files for the debug log.
- dbms.logs.debug.rotation.size: Threshold for rotation of the debug log.
- dbms.logs.gc.enabled: Enable GC Logging.
- dbms.logs.gc.options: GC Logging Options.
- dbms.logs.gc.rotation.keep_number: Number of GC logs to keep.
- dbms.logs.gc.rotation.size: Size of each GC log that is kept.
- dbms.logs.http.enabled: Enable HTTP request logging.
- dbms.logs.http.path: Path to HTTP request log.

- dbms.logs.http.rotation.keep_number: Number of HTTP logs to keep.
- dbms.logs.http.rotation.size: Size of each HTTP log that is kept.
- dbms.logs.query.allocation_logging_enabled: Log allocated bytes for the executed queries being logged.
- dbms.logs.query.enabled: Log executed queries.
- dbms.logs.query.page_logging_enabled: Log page hits and page faults for the executed queries being logged.
- dbms.logs.query.parameter_logging_enabled: Log parameters for the executed queries being logged.
- dbms.logs.query.path: Path to the query log file.
- dbms.logs.query.rotation.keep_number: Maximum number of history files for the query log.
- dbms.logs.query.rotation.size: The file size in bytes at which the query log will auto-rotate.
- dbms.logs.query.runtime_logging_enabled: Logs which runtime that was used to run the query.
- dbms.logs.query.threshold: If the execution of query takes more time than this threshold, the query is logged once completed - provided query logging is set to INFO.
- dbms.logs.query.time_logging_enabled: Log detailed time information for the executed queries being logged.
- dbms.logs.security.level: Security log level threshold.
- dbms.logs.security.path: Path to the security log file.
- dbms.logs.security.rotation.delay: Minimum time interval after last rotation of the security log before it may be rotated again.
- dbms.logs.security.rotation.keep_number: Maximum number of history files for the security log.
- dbms.logs.security.rotation.size: Threshold for rotation of the security log.
- dbms.logs.user.path: Path to the user log file.
- dbms.logs.user.rotation.delay: Minimum time interval after last rotation of the user log before it may be rotated again.
- dbms.logs.user.rotation.keep_number: Maximum number of history files for the user log.
- dbms.logs.user.rotation.size: Threshold for rotation of the user log.
- dbms.logs.user.stdout_enabled: Send user logs to the process stdout.
- dbms.max_databases: The maximum number of databases.
- dbms.memory.heap.initial_size: Initial heap size.
- dbms.memory.heap.max_size: Maximum heap size.
- dbms.memory.pagecache.size: The amount of memory to use for mapping the store files, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g').
- dbms.memory.pagecache.swapper: Specify which page swapper to use for doing paged IO.
- dbms.memory.pagecache.warmup.enable: Page cache can be configured to perform usage sampling of loaded pages that can be used to construct active load profile.
- dbms.memory.pagecache.warmup.preload: Page cache warmup can be configured to prefetch files, preferably when cache size is bigger than store size.
- dbms.memory.pagecache.warmup.preload.whitelist: Page cache warmup prefetch file whitelist regex.
- dbms.memory.pagecache.warmup.profile.interval: The profiling frequency for the page cache.
- dbms.mode: Configure the operating mode of the database — 'SINGLE' for stand-alone operation, 'CORE' for operating as a core member of a Causal Cluster, or 'READ_REPLICA' for operating as a

read replica member of a Causal Cluster.

- dbms.netty.ssl.provider: Netty SSL provider.
- dbms.query_cache_size: The number of Cypher query execution plans that are cached.
- dbms.read_only: Only allow read operations from this Neo4j instance.
- dbms.reconciler.max_backoff: Defines the maximum amount of time to wait before retrying after the dbms fails to reconcile a database to its desired state.
- dbms.reconciler.may_retry: Defines whether the dbms may retry reconciling a database to its desired state.
- dbms.reconciler.min_backoff: Defines the minimum amount of time to wait before retrying after the dbms fails to reconcile a database to its desired state.
- dbms.record_format: Database record format.
- dbms.recovery.fail_on_missing_files: If `true`, Neo4j will abort recovery if logical log files are missing.
- dbms.relationship_grouping_threshold: Relationship count threshold for considering a node to be dense.
- dbms.rest.transaction.idle_timeout: Timeout for idle transactions in the REST endpoint.
- dbms.routing_ttl: How long callers should cache the response of the routing procedure `dbms.routing.getRoutingTable()`.
- dbms.security.allow_csv_import_from_file_urls: Determines if Cypher will allow using file URLs when loading data using `LOAD CSV`.
- dbms.security.auth_cache_max_capacity: The maximum capacity for authentication and authorization caches (respectively).
- dbms.security.auth_cache_ttl: The time to live (TTL) for cached authentication and authorization info when using external auth providers (LDAP or plugin).
- dbms.security.auth_cache_use_ttl: Enable time-based eviction of the authentication and authorization info cache for external auth providers (LDAP or plugin).
- dbms.security.auth_enabled: Enable auth requirement to access Neo4j.
- dbms.security.auth_lock_time: The amount of time user account should be locked after a configured number of unsuccessful authentication attempts.
- dbms.security.auth_max_failed_attempts: The maximum number of unsuccessful authentication attempts before imposing a user lock for the configured amount of time.The locked out user will not be able to log in until the lock period expires, even if correct credentials are provided.
- dbms.security.authentication_providers: A list of security authentication providers containing the users and roles.
- dbms.security.authorization_providers: A list of security authorization providers containing the users and roles.
- dbms.security.causal_clustering_status_auth_enabled: Require authorization for access to the Causal Clustering status endpoints.
- dbms.security.http_access_control_allow_origin: Value of the Access-Control-Allow-Origin header sent over any HTTP or HTTPS connector.
- dbms.security.http_auth_whitelist: Defines a whitelist of http paths where Neo4j authentication is not required.
- dbms.security.http_strict_transport_security: Value of the HTTP Strict-Transport-Security (HSTS) response header.
- dbms.security.ldap.authentication.cache_enabled: Determines if the result of authentication via the LDAP server should be cached or not.
- dbms.security.ldap.authentication.mechanism: LDAP authentication mechanism.

- dbms.security.ldap.authentication.use_samaccountname: Perform authentication with sAMAccountName instead of DN. Using this setting requires `dbms.security.ldap.authorization.system_username` and dbms.security.ldap.authorization.system_password to be used since there is no way to log in through ldap directly with the sAMAccountName, instead the login name will be resolved to a DN that will be used to log in with.
- dbms.security.ldap.authentication.user_dn_template: LDAP user DN template.
- dbms.security.ldap.authorization.group_membership_attributes: A list of attribute names on a user object that contains groups to be used for mapping to roles when LDAP authorization is enabled.
- dbms.security.ldap.authorization.group_to_role_mapping: An authorization mapping from LDAP group names to Neo4j role names.
- dbms.security.ldap.authorization.system_password: An LDAP system account password to use for authorization searches when `dbms.security.ldap.authorization.use_system_account` is `true`.
- dbms.security.ldap.authorization.system_username: An LDAP system account username to use for authorization searches when `dbms.security.ldap.authorization.use_system_account` is `true`.
- dbms.security.ldap.authorization.use_system_account: Perform LDAP search for authorization info using a system account instead of the user's own account. If this is set to `false` (default), the search for group membership will be performed directly after authentication using the LDAP context bound with the user's own account.
- dbms.security.ldap.authorization.user_search_base: The name of the base object or named context to search for user objects when LDAP authorization is enabled.
- dbms.security.ldap.authorization.user_search_filter: The LDAP search filter to search for a user principal when LDAP authorization is enabled.
- dbms.security.ldap.connection_timeout: The timeout for establishing an LDAP connection.
- dbms.security.ldap.host: URL of LDAP server to use for authentication and authorization.
- dbms.security.ldap.read_timeout: The timeout for an LDAP read request (i.e.
- dbms.security.ldap.referral: The LDAP referral behavior when creating a connection.
- dbms.security.ldap.use_starttls: Use secure communication with the LDAP server using opportunistic TLS.
- dbms.security.log_successful_authentication: Set to log successful authentication events to the security log.
- dbms.security.procedures.default_allowed: The default role that can execute all procedures and user-defined functions that are not covered by the `dbms.security.procedures.roles` setting.
- dbms.security.procedures.roles: This provides a finer level of control over which roles can execute procedures than the `dbms.security.procedures.default_allowed` setting.
- dbms.security.procedures.unrestricted: A list of procedures and user defined functions (comma separated) that are allowed full access to the database.
- dbms.security.procedures.whitelist: A list of procedures (comma separated) that are to be loaded.
- dbms.security.property_level.enabled: This has been replaced by privilege management on roles.
- dbms.shutdown_transaction_end_timeout: The maximum amount of time to wait for running transactions to complete before allowing initiated database shutdown to continue.
- dbms.threads.worker_count: Number of Neo4j worker threads.
- dbms.track_query_allocation: Enables or disables tracking of how many bytes are allocated by the execution of a query.
- dbms.track_query_cpu_time: Enables or disables tracking of how much time a query spends actively executing on the CPU.
- dbms.transaction.bookmark_ready_timeout: The maximum amount of time to wait for the

database state represented by the bookmark.

- dbms.transaction.concurrent.maximum: The maximum number of concurrently running transactions.
- dbms.transaction.monitor.check.interval: Configures the time interval between transaction monitor checks.
- dbms.transaction.sampling.percentage: Transaction sampling percentage.
- dbms.transaction.timeout: The maximum time interval of a transaction within which it should be completed.
- dbms.transaction.tracing.level: Transaction creation tracing level.
- dbms.tx_log.preallocate: Specify if Neo4j should try to preallocate logical log file in advance.
- dbms.tx_log.rotation.retention_policy: Make Neo4j keep the logical transaction logs for being able to backup the database.
- dbms.tx_log.rotation.size: Specifies at which file size the logical log will auto-rotate.
- dbms.tx_state.max_off_heap_memory: The maximum amount of off-heap memory that can be used to store transaction state data; it's a total amount of memory shared across all active transactions.
- dbms.tx_state.memory_allocation: Defines whether memory for transaction state should be allocated on- or off-heap.
- dbms.tx_state.off_heap.block_cache_size: Defines the size of the off-heap memory blocks cache.
- dbms.tx_state.off_heap.max_cacheable_block_size: Defines the maximum size of an off-heap memory block that can be cached to speed up allocations for transaction state data.
- dbms.unmanaged_extension_classes: Comma-separated list of <classname>=<mount point> for unmanaged extensions.
- dbms.windows_service_name: Name of the Windows Service.
- fabric.database.name: Name of the Fabric database.
- fabric.driver.api: Determines which driver API will be used.
- fabric.driver.connection.connect_timeout: Socket connection timeout. A timeout of zero is treated as an infinite timeout and will be bound by the timeout configured on the operating system level.
- fabric.driver.connection.max_lifetime: Pooled connections older than this threshold will be closed and removed from the pool. Setting this option to a low value will cause a high connection churn and might result in a performance hit. It is recommended to set maximum lifetime to a slightly smaller value than the one configured in network equipment (load balancer, proxy, firewall, etc.
- fabric.driver.connection.pool.acquisition_timeout: Maximum amount of time spent attempting to acquire a connection from the connection pool. This timeout only kicks in when all existing connections are being used and no new connections can be created because maximum connection pool size has been reached. Error is raised when connection can't be acquired within configured time. Negative values are allowed and result in unlimited acquisition timeout.
- fabric.driver.connection.pool.idle_test: Pooled connections that have been idle in the pool for longer than this timeout will be tested before they are used again, to ensure they are still alive. If this option is set too low, an additional network call will be incurred when acquiring a connection, which causes a performance hit. If this is set high, no longer live connections might be used which might lead to errors. Hence, this parameter tunes a balance between the likelihood of experiencing connection problems and performance Normally, this parameter should not need tuning. Value 0 means connections will always be tested for validity.
- fabric.driver.connection.pool.max_size: Maximum total number of connections to be managed by a connection pool. The limit is enforced for a combination of a host and user.
- fabric.driver.logging.level: Sets level for driver internal logging.
- fabric.routing.servers: A comma-separated list of Fabric instances that form a routing group.

- fabric.routing.ttl: The time to live (TTL) of a routing table for fabric routing group.
- fabric.stream.buffer.low_watermark: Number of records in prefetching buffer that will trigger prefetching again.
- fabric.stream.buffer.size: Maximal size of a buffer used for pre-fetching result records of remote queries. To compensate for latency to remote databases, the Fabric execution engine pre-fetches records needed for local executions. This limit is enforced per fabric query.
- fabric.stream.concurrency: Maximal concurrency within Fabric queries. Limits the number of iterations of each subquery that are executed concurrently.
- metrics.bolt.messages.enabled: Enable reporting metrics about Bolt Protocol message processing.
- metrics.csv.enabled: Set to `true` to enable exporting metrics to CSV files.
- metrics.csv.interval: The reporting interval for the CSV files.
- metrics.csv.rotation.keep_number: Maximum number of history files for the csv files.
- metrics.csv.rotation.size: The file size in bytes at which the csv files will auto-rotate.
- metrics.cypher.replanning.enabled: Enable reporting metrics about number of occurred replanning events.
- metrics.enabled: Enable metrics.
- metrics.graphite.enabled: Set to `true` to enable exporting metrics to Graphite.
- metrics.graphite.interval: The reporting interval for Graphite.
- metrics.graphite.server: The hostname or IP address of the Graphite server.
- metrics.jmx.enabled: Set to `true` to enable the JMX metrics endpoint.
- metrics.jvm.buffers.enabled: Enable reporting metrics about the buffer pools.
- metrics.jvm.file.descriptors.enabled: Enable reporting metrics about the number of open file descriptors.
- metrics.jvm.gc.enabled: Enable reporting metrics about the duration of garbage collections.
- metrics.jvm.heap.enabled: Enable reporting metrics about the heap memory usage.
- metrics.jvm.memory.enabled: Enable reporting metrics about the memory usage.
- metrics.jvm.threads.enabled: Enable reporting metrics about the current number of threads running.
- metrics.neo4j.causal_clustering.enabled: Enable reporting metrics about Causal Clustering mode.
- metrics.neo4j.checkpointing.enabled: Enable reporting metrics about Neo4j check pointing.
- metrics.neo4j.counts.enabled: Enable reporting metrics about approximately how many entities are in the database.
- metrics.neo4j.data.counts.enabled: Enable reporting metrics about number of entities in the database.
- metrics.neo4j.logs.enabled: Enable reporting metrics about the Neo4j transaction logs.
- metrics.neo4j.pagecache.enabled: Enable reporting metrics about the Neo4j page cache.
- metrics.neo4j.server.enabled: Enable reporting metrics about Server threading info.
- metrics.neo4j.size.enabled: Enable reporting metrics about the store size of each database.
- metrics.neo4j.tx.enabled: Enable reporting metrics about transactions.
- metrics.prefix: A common prefix for the reported metrics field names.
- metrics.prometheus.enabled: Set to `true` to enable the Prometheus endpoint.
- metrics.prometheus.endpoint: The hostname and port to use as Prometheus endpoint.

Table 43. browser.allow_outgoing_connections

Description	Configure the policy for outgoing Neo4j Browser connections.
Valid values	browser.allow_outgoing_connections, a boolean
Default value	`true`

Table 44. browser.credential_timeout

Description	Configure the Neo4j Browser to time out logged in users after this idle period. Setting this to 0 indicates no limit.
Valid values	browser.credential_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`0ns`

Table 45. browser.post_connect_cmd

Description	Commands to be run when Neo4j Browser successfully connects to this server. Separate multiple commands with semi-colon.
Valid values	browser.post_connect_cmd, a string
Default value	

Table 46. browser.remote_content_hostname_whitelist

Description	Whitelist of hosts for the Neo4j Browser to be allowed to fetch content from.
Valid values	browser.remote_content_hostname_whitelist, a string
Default value	`guides.neo4j.com,localhost`

Table 47. browser.retain_connection_credentials

Description	Configure the Neo4j Browser to store or not store user credentials.
Valid values	browser.retain_connection_credentials, a boolean
Default value	`true`

Table 48. causal_clustering.catch_up_client_inactivity_timeout

Description	The catch up protocol times out if the given duration elapses with no network activity. Every message received by the client from the server extends the time out duration.
Valid values	causal_clustering.catch_up_client_inactivity_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`10m`

Table 49. causal_clustering.catchup_batch_size

Description	The maximum batch size when catching up (in unit of entries)
Valid values	causal_clustering.catchup_batch_size, an integer
Default value	`64`

Table 50. causal_clustering.cluster_allow_reads_on_followers

Description	Configure if the `` `dbms.routing.getRoutingTable()` `` procedure should include followers as read endpoints or return only read replicas. Note: if there are no read replicas in the cluster, followers are returned as read end points regardless the value of this setting. Defaults to true so that followers are available for read-only queries in a typical heterogeneous setup.
Valid values	causal_clustering.cluster_allow_reads_on_followers, a boolean
Default value	`true`

Table 51. causal_clustering.cluster_topology_refresh

Description	Time between scanning the cluster to refresh current server's view of topology.
Valid values	causal_clustering.cluster_topology_refresh, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's') which is minimum `PT1S`
Default value	`5s`

Table 52. causal_clustering.connect-randomly-to-server-group

Description	+Comma separated list of groups to be used by the connect-randomly-to-server-group selection strategy. The connect-randomly-to-server-group strategy is used if the list of strategies (`causal_clustering.upstream_selection_strategy`) includes the value `connect-randomly-to-server-group`.
Valid values	causal_clustering.connect-randomly-to-server-group, a ',' separated list with elements of type 'a string'.
Default value	

Table 53. causal_clustering.discovery_advertised_address

Description	Advertised cluster member discovery management communication.
Valid values	causal_clustering.discovery_advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:5000`

Table 54. causal_clustering.discovery_listen_address

Description	Host and port to bind the cluster member discovery management communication.
Valid values	causal_clustering.discovery_listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:5000`

Table 55. causal_clustering.discovery_type

Description	Configure the discovery type used for cluster name resolution.
Valid values	causal_clustering.discovery_type, one of [DNS, LIST, SRV, K8S]
Default value	`LIST`

Table 56. causal_clustering.enable_pre_voting

Description	Enable pre-voting extension to the Raft protocol (this is breaking and must match between the core cluster members)
Valid values	causal_clustering.enable_pre_voting, a boolean
Default value	`true`

Table 57. causal_clustering.global_session_tracker_state_size

Description	The maximum file size before the global session tracker state file is rotated (in unit of entries)
Valid values	causal_clustering.global_session_tracker_state_size, an integer
Default value	`1000`

Table 58. causal_clustering.handshake_timeout

Description	Time out for protocol negotiation handshake.
Valid values	causal_clustering.handshake_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`20s`

Table 59. causal_clustering.in_flight_cache.max_bytes

Description	The maximum number of bytes in the in-flight cache.
Valid values	causal_clustering.in_flight_cache.max_bytes, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`)
Default value	`2147483648`

Table 60. causal_clustering.in_flight_cache.max_entries

Description	The maximum number of entries in the in-flight cache.
Valid values	causal_clustering.in_flight_cache.max_entries, an integer
Default value	`1024`

Table 61. causal_clustering.in_flight_cache.type

Description	Type of in-flight cache.
Valid values	causal_clustering.in_flight_cache.type, one of [NONE, CONSECUTIVE, UNBOUNDED]
Default value	`CONSECUTIVE`

Table 62. causal_clustering.initial_discovery_members

Description	A comma-separated list of other members of the cluster to join.
Valid values	causal_clustering.initial_discovery_members, a ',' separated list with elements of type 'a socket address'.

Table 63. causal_clustering.join_catch_up_timeout

Description	Time out for a new member to catch up.
Valid values	causal_clustering.join_catch_up_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`10m`

Table 64. causal_clustering.kubernetes.address

Description	Address for Kubernetes API.
Valid values	causal_clustering.kubernetes.address, a socket address
Default value	`kubernetes.default.svc:443`

Table 65. causal_clustering.kubernetes.ca_crt

Description	File location of CA certificate for Kubernetes API.
Valid values	causal_clustering.kubernetes.ca_crt, a path
Default value	`/var/run/secrets/kubernetes.io/serviceaccount/ca.crt`

Table 66. causal_clustering.kubernetes.label_selector

Description	LabelSelector for Kubernetes API.

Valid values	causal_clustering.kubernetes.label_selector, a string

Table 67. causal_clustering.kubernetes.namespace

Description	File location of namespace for Kubernetes API.
Valid values	causal_clustering.kubernetes.namespace, a path
Default value	`/var/run/secrets/kubernetes.io/serviceaccount/namespace`

Table 68. causal_clustering.kubernetes.service_port_name

Description	Service port name for discovery for Kubernetes API.
Valid values	causal_clustering.kubernetes.service_port_name, a string

Table 69. causal_clustering.kubernetes.token

Description	File location of token for Kubernetes API.
Valid values	causal_clustering.kubernetes.token, a path
Default value	`/var/run/secrets/kubernetes.io/serviceaccount/token`

Table 70. causal_clustering.last_applied_state_size

Description	The maximum file size before the storage file is rotated (in unit of entries)
Valid values	causal_clustering.last_applied_state_size, an integer
Default value	`1000`

Table 71. causal_clustering.leader_election_timeout

Description	The time limit within which a new leader election will occur if no messages are received.
Valid values	causal_clustering.leader_election_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`7s`

Table 72. causal_clustering.load_balancing.plugin

Description	The load balancing plugin to use.
Valid values	causal_clustering.load_balancing.plugin, a string
Default value	`server_policies`

Table 73. causal_clustering.load_balancing.shuffle

Description	Enables shuffling of the returned load balancing result.
Valid values	causal_clustering.load_balancing.shuffle, a boolean
Default value	`true`

Table 74. causal_clustering.log_shipping_max_lag

Description	The maximum lag allowed before log shipping pauses (in unit of entries)
Valid values	causal_clustering.log_shipping_max_lag, an integer
Default value	`256`

Table 75. causal_clustering.middleware.logging.level

Description	The level of middleware logging.
Valid values	causal_clustering.middleware.logging.level, one of [DEBUG, INFO, WARN, ERROR, NONE]
Default value	`WARN`

Table 76. causal_clustering.minimum_core_cluster_size_at_formation

Description	Minimum number of Core machines initially required to form a cluster. The cluster will form when at least this many Core members have discovered each other.
Valid values	causal_clustering.minimum_core_cluster_size_at_formation, an integer which is minimum `2`
Default value	`3`

Table 77. causal_clustering.minimum_core_cluster_size_at_runtime

Description	The minimum size of the dynamically adjusted voting set (which only core members may be a part of). Adjustments to the voting set happen automatically as the availability of core members changes, due to explicit operations such as starting or stopping a member, or unintended issues such as network partitions. Note that this dynamic scaling of the voting set is generally desirable as under some circumstances it can increase the number of instance failures which may be tolerated. A majority of the voting set must be available before voting in or out members.
Valid values	causal_clustering.minimum_core_cluster_size_at_runtime, an integer which is minimum `2`
Default value	`3`

Table 78. causal_clustering.multi_dc_license

Description	Enable multi-data center features. Requires appropriate licensing.
Valid values	causal_clustering.multi_dc_license, a boolean
Default value	`false`

Table 79. causal_clustering.protocol_implementations.catchup

Description	Catchup protocol implementation versions that this instance will allow in negotiation as a comma-separated list. Order is not relevant: the greatest value will be preferred. An empty list will allow all supported versions. Example value: "1.1, 1.2, 2.1, 2.2"
Valid values	causal_clustering.protocol_implementations.catchup, a ',' separated list with elements of type 'an application protocol version'.
Default value	

Table 80. causal_clustering.protocol_implementations.compression

Description	Network compression algorithms that this instance will allow in negotiation as a comma-separated list. Listed in descending order of preference for incoming connections. An empty list implies no compression. For outgoing connections this merely specifies the allowed set of algorithms and the preference of the remote peer will be used for making the decision. Allowable values: [Gzip, Snappy, Snappy_validating, LZ4, LZ4_high_compression, LZ_validating, LZ4_high_compression_validating]
Valid values	causal_clustering.protocol_implementations.compression, a ',' separated list with elements of type 'a string'.
Default value	

Table 81. causal_clustering.protocol_implementations.raft

Description	Raft protocol implementation versions that this instance will allow in negotiation as a comma-separated list. Order is not relevant: the greatest value will be preferred. An empty list will allow all supported versions. Example value: "1.0, 1.3, 2.0, 2.1"
Valid values	causal_clustering.protocol_implementations.raft, a ',' separated list with elements of type 'an application protocol version'.
Default value	

Table 82. causal_clustering.pull_interval

Description	Interval of pulling updates from cores.
Valid values	causal_clustering.pull_interval, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`1s`

Table 83. causal_clustering.raft_advertised_address

Description	Advertised hostname/IP address and port for the RAFT server.
Valid values	causal_clustering.raft_advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:7000`

Table 84. causal_clustering.raft_in_queue_max_batch_bytes

Description	Largest batch processed by RAFT in bytes.
Valid values	causal_clustering.raft_in_queue_max_batch_bytes, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`)
Default value	`8388608`

Table 85. causal_clustering.raft_in_queue_max_bytes

Description	Maximum number of bytes in the RAFT in-queue.
Valid values	causal_clustering.raft_in_queue_max_bytes, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`)
Default value	`2147483648`

Table 86. causal_clustering.raft_listen_address

Description	Network interface and port for the RAFT server to listen on.
Valid values	causal_clustering.raft_listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:7000`

Table 87. causal_clustering.raft_log_implementation

Description	RAFT log implementation.
Valid values	causal_clustering.raft_log_implementation, a string
Default value	`SEGMENTED`

Table 88. causal_clustering.raft_log_prune_strategy

Description	RAFT log pruning strategy.
Valid values	causal_clustering.raft_log_prune_strategy, a string
Default value	`1g size`

Table 89. causal_clustering.raft_log_pruning_frequency

Description	RAFT log pruning frequency.
Valid values	causal_clustering.raft_log_pruning_frequency, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`10m`

Table 90. causal_clustering.raft_log_reader_pool_size

Description	RAFT log reader pool size.
Valid values	causal_clustering.raft_log_reader_pool_size, an integer
Default value	`8`

Table 91. causal_clustering.raft_log_rotation_size

Description	RAFT log rotation size.
Valid values	causal_clustering.raft_log_rotation_size, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `1024`
Default value	`262144000`

Table 92. causal_clustering.raft_membership_state_size

Description	The maximum file size before the membership state file is rotated (in unit of entries)
Valid values	causal_clustering.raft_membership_state_size, an integer
Default value	`1000`

Table 93. causal_clustering.raft_term_state_size

Description	The maximum file size before the term state file is rotated (in unit of entries)
Valid values	causal_clustering.raft_term_state_size, an integer
Default value	`1000`

Table 94. causal_clustering.raft_vote_state_size

Description	The maximum file size before the vote state file is rotated (in unit of entries)
Valid values	causal_clustering.raft_vote_state_size, an integer
Default value	`1000`

Table 95. causal_clustering.refuse_to_be_leader

Description	Prevents the current instance from volunteering to become Raft leader. Defaults to false, and should only be used in exceptional circumstances by expert users. Using this can result in reduced availability for the cluster.
Valid values	causal_clustering.refuse_to_be_leader, a boolean
Default value	`false`

Table 96. causal_clustering.replicated_lease_state_size

Description	The maximum file size before the replicated lease state file is rotated (in unit of entries)
Valid values	causal_clustering.replicated_lease_state_size, an integer
Default value	`1000`

Table 97. causal_clustering.replication_leader_await_timeout

Description	The duration for which the replicator will await a new leader.
Valid values	causal_clustering.replication_leader_await_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`10s`

Table 98. causal_clustering.replication_retry_timeout_base

Description	The initial timeout until replication is retried. The timeout will increase exponentially.
Valid values	causal_clustering.replication_retry_timeout_base, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`10s`

Table 99. causal_clustering.replication_retry_timeout_limit

Description	The upper limit for the exponentially incremented retry timeout.
Valid values	causal_clustering.replication_retry_timeout_limit, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`1m`

Table 100. causal_clustering.server_groups

Description	A list of group names for the server used when configuring load balancing and replication policies.
Valid values	causal_clustering.server_groups, a ',' separated list with elements of type 'a string'.

Default value	

Table 101. causal_clustering.state_machine_apply_max_batch_size

Description	The maximum number of operations to be batched during applications of operations in the state machines.
Valid values	causal_clustering.state_machine_apply_max_batch_size, an integer
Default value	`16`

Table 102. causal_clustering.state_machine_flush_window_size

Description	The number of operations to be processed before the state machines flush to disk.
Valid values	causal_clustering.state_machine_flush_window_size, an integer
Default value	`4096`

Table 103. causal_clustering.status_throughput_window

Description	Sampling window for throughput estimate reported in the status endpoint.
Valid values	causal_clustering.status_throughput_window, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's') which is in the range `PT1S` to `PT5M`
Default value	`5s`

Table 104. causal_clustering.store_copy_chunk_size

Description	Store copy chunk size.
Valid values	causal_clustering.store_copy_chunk_size, an integer which is in the range `4096` to `1048576`
Default value	`32768`

Table 105. causal_clustering.store_copy_max_retry_time_per_request

Description	+Maximum retry time per request during store copy. Regular store files and indexes are downloaded in separate requests during store copy. This configures the maximum time failed requests are allowed to resend.
Valid values	causal_clustering.store_copy_max_retry_time_per_request, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`20m`

Table 106. causal_clustering.transaction_advertised_address

Description	Advertised hostname/IP address and port for the transaction shipping server.

Valid values	causal_clustering.transaction_advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:6000`

Table 107. causal_clustering.transaction_listen_address

Description	Network interface and port for the transaction shipping server to listen on. Please note that it is also possible to run the backup client against this port so always limit access to it via the firewall and configure an ssl policy.
Valid values	causal_clustering.transaction_listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:6000`

Table 108. causal_clustering.unknown_address_logging_throttle

Description	Throttle limit for logging unknown cluster member address.
Valid values	causal_clustering.unknown_address_logging_throttle, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`10s`

Table 109. causal_clustering.upstream_selection_strategy

Description	An ordered list in descending preference of the strategy which read replicas use to choose the upstream server from which to pull transactional updates.
Valid values	causal_clustering.upstream_selection_strategy, a ',' separated list with elements of type 'a string'.
Default value	`default`

Table 110. causal_clustering.user_defined_upstream_strategy

Description	+Configuration of a user-defined upstream selection strategy. The user-defined strategy is used if the list of strategies (`causal_clustering.upstream_selection_strategy`) includes the value `user_defined`.
Valid values	causal_clustering.user_defined_upstream_strategy, a string
Default value	

Table 111. cypher.default_language_version

Description	Set this to specify the default parser (language version).
Valid values	cypher.default_language_version, one of [default, 3.5, 4.0]
Default value	`default`

Table 112. cypher.forbid_exhaustive_shortestpath

Description	This setting is associated with performance optimization. Set this to `true` in situations where it is preferable to have any queries using the 'shortestPath' function terminate as soon as possible with no answer, rather than potentially running for a long time attempting to find an answer (even if there is no path to be found). For most queries, the 'shortestPath' algorithm will return the correct answer very quickly. However there are some cases where it is possible that the fast bidirectional breadth-first search algorithm will find no results even if they exist. This can happen when the predicates in the `WHERE` clause applied to 'shortestPath' cannot be applied to each step of the traversal, and can only be applied to the entire path. When the query planner detects these special cases, it will plan to perform an exhaustive depth-first search if the fast algorithm finds no paths. However, the exhaustive search may be orders of magnitude slower than the fast algorithm. If it is critical that queries terminate as soon as possible, it is recommended that this option be set to `true`, which means that Neo4j will never consider using the exhaustive search for shortestPath queries. However, please note that if no paths are found, an error will be thrown at run time, which will need to be handled by the application.
Valid values	cypher.forbid_exhaustive_shortestpath, a boolean
Default value	`false`

Table 113. cypher.forbid_shortestpath_common_nodes

Description	This setting is associated with performance optimization. The shortest path algorithm does not work when the start and end nodes are the same. With this setting set to `false` no path will be returned when that happens. The default value of `true` will instead throw an exception. This can happen if you perform a shortestPath search after a cartesian product that might have the same start and end nodes for some of the rows passed to shortestPath. If it is preferable to not experience this exception, and acceptable for results to be missing for those rows, then set this to `false`. If you cannot accept missing results, and really want the shortestPath between two common nodes, then re-write the query using a standard Cypher variable length pattern expression followed by ordering by path length and limiting to one result.
Valid values	cypher.forbid_shortestpath_common_nodes, a boolean
Default value	`true`

Table 114. cypher.hints_error

Description	Set this to specify the behavior when Cypher planner or runtime hints cannot be fulfilled. If true, then non-conformance will result in an error, otherwise only a warning is generated.
Valid values	cypher.hints_error, a boolean
Default value	`false`

Table 115. cypher.lenient_create_relationship

Description	Set this to change the behavior for Cypher create relationship when the start or end node is missing. By default this fails the query and stops execution, but by setting this flag the create operation is simply not performed and execution continues.
Valid values	cypher.lenient_create_relationship, a boolean
Default value	`false`

Table 116. cypher.min_replan_interval

Description	The minimum time between possible cypher query replanning events. After this time, the graph statistics will be evaluated, and if they have changed by more than the value set by <<config_cypher.statistics_divergence_threshold,cypher.statistics_divergence_threshold>>, the query will be replanned. If the statistics have not changed sufficiently, the same interval will need to pass before the statistics will be evaluated again. Each time they are evaluated, the divergence threshold will be reduced slightly until it reaches 10% after 7h, so that even moderately changing databases will see query replanning after a sufficiently long time interval.
Valid values	cypher.min_replan_interval, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`10s`

Table 117. cypher.planner

Description	Set this to specify the default planner for the default language version.
Valid values	cypher.planner, one of [DEFAULT, COST]
Default value	`DEFAULT`

Table 118. cypher.query_max_allocations

Description	The maximum amount of heap memory allocations to for cypher to perform on a single query, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). Zero means 'unlimited'. If a query exceeds this limit, it will be terminated. Determining the heap memory allocations done by a query is a rough estimate and not an exact measurement. If no memory limit is configured, queries will be allowed to allocate as much heap memory as needed. This could potentially lead to queries consuming more heap memory than available, which will kill the Neo4j server.
Valid values	cypher.query_max_allocations, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `0`
Dynamic	true
Default value	`0`

Table 119. cypher.statistics_divergence_threshold

Description	The threshold when a plan is considered stale. If any of the underlying statistics used to create the plan have changed more than this value, the plan will be considered stale and will be replanned. Change is calculated as abs(a-b)/max(a,b). This means that a value of 0.75 requires the database to approximately quadruple in size. A value of 0 means replan as soon as possible, with the soonest being defined by the <<config_cypher.min_replan_interval,cypher.min_replan_interval>> which defaults to 10s. After this interval the divergence threshold will slowly start to decline, reaching 10% after about 7h. This will ensure that long running databases will still get query replanning on even modest changes, while not replanning frequently unless the changes are very large.
Valid values	cypher.statistics_divergence_threshold, a double which is in the range `0.0` to `1.0`
Default value	`0.75`

Table 120. db.temporal.timezone

Description	Database timezone for temporal functions. All Time and DateTime values that are created without an explicit timezone will use this configured default timezone.
Valid values	db.temporal.timezone, a string describing a timezone, either described by offset (e.g. '+02:00') or by name (e.g. 'Europe/Stockholm')
Default value	`Z`

Table 121. dbms.allow_upgrade

Description	Whether to allow an upgrade in case the current version of the database starts against an older version.
Valid values	dbms.allow_upgrade, a boolean
Default value	`false`

Table 122. dbms.backup.enabled

Description	Enable support for running online backups.
Valid values	dbms.backup.enabled, a boolean
Default value	`true`

Table 123. dbms.backup.listen_address

Description	Network interface and port for the backup server to listen on.
Valid values	dbms.backup.listen_address, a socket address
Default value	`127.0.0.1:6362`

Table 124. dbms.checkpoint

Description	Configures the general policy for when check-points should occur. The default policy is the 'periodic' check-point policy, as specified by the '<<config_dbms.checkpoint.interval.tx,dbms.checkpoint.interval.tx>>' and '<<config_dbms.checkpoint.interval.time,dbms.checkpoint.interval.time>>' settings. The Neo4j Enterprise Edition provides two alternative policies: The first is the 'continuous' check-point policy, which will ignore those settings and run the check-point process all the time. The second is the 'volumetric' check-point policy, which makes a best-effort at check-pointing often enough so that the database doesn't get too far behind on deleting old transaction logs in accordance with the '<<config_dbms.tx_log.rotation.retention_policy,dbms.tx_log.rotation.retention_policy>>' setting.
Valid values	dbms.checkpoint, one of [PERIODIC, CONTINUOUS, VOLUMETRIC]
Default value	`PERIODIC`

Table 125. dbms.checkpoint.interval.time

Description	Configures the time interval between check-points. The database will not check-point more often than this (unless check pointing is triggered by a different event), but might check-point less often than this interval, if performing a check-point takes longer time than the configured interval. A check-point is a point in the transaction logs, from which recovery would start from. Longer check-point intervals typically means that recovery will take longer to complete in case of a crash. On the other hand, a longer check-point interval can also reduce the I/O load that the database places on the system, as each check-point implies a flushing and forcing of all the store files.
Valid values	dbms.checkpoint.interval.time, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`15m`

Table 126. dbms.checkpoint.interval.tx

Description	Configures the transaction interval between check-points. The database will not check-point more often than this (unless check pointing is triggered by a different event), but might check-point less often than this interval, if performing a check-point takes longer time than the configured interval. A check-point is a point in the transaction logs, from which recovery would start from. Longer check-point intervals typically means that recovery will take longer to complete in case of a crash. On the other hand, a longer check-point interval can also reduce the I/O load that the database places on the system, as each check-point implies a flushing and forcing of all the store files. The default is '100000' for a check-point every 100000 transactions.
Valid values	dbms.checkpoint.interval.tx, an integer which is minimum `1`
Default value	`100000`

Table 127. dbms.checkpoint.iops.limit

Description	Limit the number of IOs the background checkpoint process will consume per second. This setting is advisory, is ignored in Neo4j Community Edition, and is followed to best effort in Enterprise Edition. An IO is in this case a 8 KiB (mostly sequential) write. Limiting the write IO in this way will leave more bandwidth in the IO subsystem to service random-read IOs, which is important for the response time of queries when the database cannot fit entirely in memory. The only drawback of this setting is that longer checkpoint times may lead to slightly longer recovery times in case of a database or system crash. A lower number means lower IO pressure, and consequently longer checkpoint times. Set this to -1 to disable the IOPS limit and remove the limitation entirely; this will let the checkpointer flush data as fast as the hardware will go. Removing the setting, or commenting it out, will set the default value of 300.
Valid values	dbms.checkpoint.iops.limit, an integer
Dynamic	true
Default value	`300`

Table 128. dbms.config.strict_validation

Description	A strict configuration validation will prevent the database from starting up if unknown configuration options are specified in the neo4j settings namespace (such as dbms., cypher., etc).
Valid values	dbms.config.strict_validation, a boolean
Default value	`false`

Table 129. dbms.connector.bolt.advertised_address

Description	Advertised address for this connector.
Valid values	dbms.connector.bolt.advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:7687`

Table 130. dbms.connector.bolt.enabled

Description	Enable the bolt connector.
Valid values	dbms.connector.bolt.enabled, a boolean
Default value	`true`

Table 131. dbms.connector.bolt.listen_address

Description	Address the connector should bind to.
Valid values	dbms.connector.bolt.listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address

Default value	`:7687`

Table 132. dbms.connector.bolt.thread_pool_keep_alive

Description	The maximum time an idle thread in the thread pool bound to this connector will wait for new tasks.
Valid values	dbms.connector.bolt.thread_pool_keep_alive, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`5m`

Table 133. dbms.connector.bolt.thread_pool_max_size

Description	The maximum number of threads allowed in the thread pool bound to this connector.
Valid values	dbms.connector.bolt.thread_pool_max_size, an integer
Default value	`400`

Table 134. dbms.connector.bolt.thread_pool_min_size

Description	The number of threads to keep in the thread pool bound to this connector, even if they are idle.
Valid values	dbms.connector.bolt.thread_pool_min_size, an integer
Default value	`5`

Table 135. dbms.connector.bolt.tls_level

Description	Encryption level to require this connector to use.
Valid values	dbms.connector.bolt.tls_level, one of [REQUIRED, OPTIONAL, DISABLED]
Default value	`DISABLED`

Table 136. dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time

Description	The maximum time to wait for the thread pool to finish processing its pending jobs and shutdown.
Valid values	dbms.connector.bolt.unsupported_thread_pool_shutdown_wait_time, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`5s`

Table 137. dbms.connector.http.advertised_address

Description	Advertised address for this connector.

Valid values	dbms.connector.http.advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:7474`

Table 138. dbms.connector.http.enabled

Description	Enable the http connector.
Valid values	dbms.connector.http.enabled, a boolean
Default value	`true`

Table 139. dbms.connector.http.listen_address

Description	Address the connector should bind to.
Valid values	dbms.connector.http.listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:7474`

Table 140. dbms.connector.https.advertised_address

Description	Advertised address for this connector.
Valid values	dbms.connector.https.advertised_address, a socket address. If missing port or hostname it is acquired from dbms.default_advertised_address
Default value	`:7473`

Table 141. dbms.connector.https.enabled

Description	Enable the https connector.
Valid values	dbms.connector.https.enabled, a boolean
Default value	`false`

Table 142. dbms.connector.https.listen_address

Description	Address the connector should bind to.
Valid values	dbms.connector.https.listen_address, a socket address. If missing port or hostname it is acquired from dbms.default_listen_address
Default value	`:7473`

Table 143. dbms.db.timezone

Description	Database timezone. Among other things, this setting influences which timezone the logs and monitoring procedures use.

Valid values	dbms.db.timezone, one of [UTC, SYSTEM]
Default value	`UTC`

Table 144. dbms.default_advertised_address

Description	Default hostname or IP address the server uses to advertise itself.
Valid values	dbms.default_advertised_address, a socket address which has no specified port
Default value	`localhost`

Table 145. dbms.default_database

Description	Name of the default database.
Valid values	dbms.default_database, A valid database name. Containing only alphabetic characters, numbers, dots and dashes, with a length between 3 and 63 characters. It should be starting with an alphabetic character but not with the name 'system'.
Default value	`neo4j`

Table 146. dbms.default_listen_address

Description	+Default network interface to listen for incoming connections. To listen for connections on all interfaces, use "0.0.0.0".
Valid values	dbms.default_listen_address, a socket address which has no specified port
Default value	`localhost`

Table 147. dbms.directories.data

Description	Path of the data directory. You must not configure more than one Neo4j installation to use the same data directory.
Valid values	dbms.directories.data, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`data`

Table 148. dbms.directories.import

Description	Sets the root directory for file URLs used with the Cypher \`LOAD CSV\` clause. This should be set to a directory relative to the Neo4j installation path, restricting access to only those files within that directory and its subdirectories. For example the value "import" will only enable access to files within the 'import' folder. Removing this setting will disable the security feature, allowing all files in the local system to be imported. Setting this to an empty field will allow access to all files within the Neo4j installation folder.

Valid values	dbms.directories.import, a path. If relative it is resolved from dbms.directories.neo4j_home

Table 149. dbms.directories.lib

Description	Path of the lib directory.
Valid values	dbms.directories.lib, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`lib`

Table 150. dbms.directories.logs

Description	Path of the logs directory.
Valid values	dbms.directories.logs, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`logs`

Table 151. dbms.directories.metrics

Description	The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.
Valid values	dbms.directories.metrics, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`metrics`

Table 152. dbms.directories.neo4j_home

Description	Root relative to which directory settings are resolved.
Valid values	dbms.directories.neo4j_home, a path which is absolute
Default value	`Defaults to current working directory`

Table 153. dbms.directories.plugins

Description	Location of the database plugin directory. Compiled Java JAR files that contain database procedures will be loaded if they are placed in this directory.
Valid values	dbms.directories.plugins, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`plugins`

Table 154. dbms.directories.run

Description	Path of the run directory. This directory holds Neo4j's runtime state, such as a pidfile when it is running in the background. The pidfile is created when starting neo4j and removed when stopping it. It may be placed on an in-memory filesystem such as tmpfs.
Valid values	dbms.directories.run, a path. If relative it is resolved from dbms.directories.neo4j_home
Default value	`run`

Table 155. dbms.directories.transaction.logs.root

Description	Root location where Neo4j will store transaction logs for configured databases.
Valid values	dbms.directories.transaction.logs.root, a path. If relative it is resolved from dbms.directories.data
Default value	`transactions`

Table 156. dbms.dynamic.setting.whitelist

Description	A list of setting name patterns (comma separated) that are allowed to be dynamically changed. The list may contain both full setting names, and partial names with the wildcard '*'. If this setting is left empty all dynamic settings updates will be blocked.
Valid values	dbms.dynamic.setting.whitelist, a ',' separated list with elements of type 'a string'.
Default value	`*`

Table 157. dbms.filewatcher.enabled

Description	Allows the enabling or disabling of the file watcher service. This is an auxiliary service but should be left enabled in almost all cases.
Valid values	dbms.filewatcher.enabled, a boolean
Default value	`true`

Table 158. dbms.import.csv.buffer_size

Description	The size of the internal buffer in bytes used by `LOAD CSV`. If the csv file contains huge fields this value may have to be increased.
Valid values	dbms.import.csv.buffer_size, a long which is minimum `1`
Default value	`2097152`

Table 159. dbms.import.csv.legacy_quote_escaping

Description	Selects whether to conform to the standard https://tools.ietf.org/html/rfc4180 for interpreting escaped quotation characters in CSV files loaded using `LOAD CSV`. Setting this to `false` will use the standard, interpreting repeated quotes '""' as a single in-lined quote, while `true` will use the legacy convention originally supported in Neo4j 3.0 and 3.1, allowing a backslash to include quotes in-lined in fields.
Valid values	dbms.import.csv.legacy_quote_escaping, a boolean
Default value	`true`

Table 160. dbms.index.default_schema_provider

Description	Index provider to use for newly created schema indexes. An index provider may store different value types in separate physical indexes. native-btree-1.0: All value types and arrays of all value types, even composite keys, are stored in one native index. lucene+native-3.0: Like native-btree-1.0 but single property strings are stored in Lucene. A native index has faster updates, less heap and CPU usage compared to a Lucene index. A native index has some limitations around key size and slower execution of CONTAINS and ENDS WITH string index queries, compared to a Lucene index. Deprecated: Which index provider to use will be a fully internal concern.
Valid values	dbms.index.default_schema_provider, a string
Default value	`native-btree-1.0`
Deprecated	The `dbms.index.default_schema_provider` configuration setting has been deprecated.

Table 161. dbms.index.fulltext.default_analyzer

Description	The name of the analyzer that the fulltext indexes should use by default.
Valid values	dbms.index.fulltext.default_analyzer, a string
Default value	`standard-no-stop-words`

Table 162. dbms.index.fulltext.eventually_consistent

Description	Whether or not fulltext indexes should be eventually consistent by default or not.
Valid values	dbms.index.fulltext.eventually_consistent, a boolean
Default value	`false`

Table 163. dbms.index.fulltext.eventually_consistent_index_update_queue_max_length

Description	The eventually_consistent mode of the fulltext indexes works by queueing up index updates to be applied later in a background thread. This newBuilder sets an upper bound on how many index updates are allowed to be in this queue at any one point in time. When it is reached, the commit process will slow down and wait for the index update applier thread to make some more room in the queue.
Valid values	dbms.index.fulltext.eventually_consistent_index_update_queue_max_length, an integer which is in the range `1` to `50000000`
Default value	`10000`

Table 164. dbms.index_sampling.background_enabled

Description	Enable or disable background index sampling.
Valid values	dbms.index_sampling.background_enabled, a boolean
Default value	`true`

Table 165. dbms.index_sampling.sample_size_limit

Description	Index sampling chunk size limit.
Valid values	dbms.index_sampling.sample_size_limit, an integer which is in the range `1048576` to `2147483647`
Default value	`8388608`

Table 166. dbms.index_sampling.update_percentage

Description	Percentage of index updates of total index size required before sampling of a given index is triggered.
Valid values	dbms.index_sampling.update_percentage, an integer which is minimum `0`
Default value	`5`

Table 167. dbms.index_searcher_cache_size

Description	The maximum number of open Lucene index searchers.
Valid values	dbms.index_searcher_cache_size, an integer which is minimum `1`
Default value	`2147483647`
Deprecated	The `dbms.index_searcher_cache_size` configuration setting has been deprecated.

Table 168. dbms.jvm.additional

Description	Additional JVM arguments. Argument order can be significant. To use a Java commercial feature, the argument to unlock commercial features must precede the argument to enable the specific feature in the config value string. For example, to use Flight Recorder, `-XX:+UnlockCommercialFeatures` must come before `-XX:+FlightRecorder`.
Valid values	dbms.jvm.additional, a string
Default value	

Table 169. dbms.lock.acquisition.timeout

Description	The maximum time interval within which lock should be acquired.
Valid values	dbms.lock.acquisition.timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`0ns`

Table 170. dbms.logs.debug.level

Description	Debug log level threshold.
Valid values	dbms.logs.debug.level, one of [DEBUG, INFO, WARN, ERROR, NONE]
Dynamic	true
Default value	`INFO`

Table 171. dbms.logs.debug.path

Description	Path to the debug log file.
Valid values	dbms.logs.debug.path, a path. If relative it is resolved from dbms.directories.logs
Default value	`debug.log`

Table 172. dbms.logs.debug.rotation.delay

Description	Minimum time interval after last rotation of the debug log before it may be rotated again.
Valid values	dbms.logs.debug.rotation.delay, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`5m`

Table 173. dbms.logs.debug.rotation.keep_number

Description	Maximum number of history files for the debug log.

Valid values	dbms.logs.debug.rotation.keep_number, an integer which is minimum `1`
Default value	`7`

Table 174. dbms.logs.debug.rotation.size

Description	Threshold for rotation of the debug log.
Valid values	dbms.logs.debug.rotation.size, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is in the range `0` to `9223372036854775807`
Default value	`20971520`

Table 175. dbms.logs.gc.enabled

Description	Enable GC Logging.
Valid values	dbms.logs.gc.enabled, a boolean
Default value	`false`

Table 176. dbms.logs.gc.options

Description	GC Logging Options.
Valid values	dbms.logs.gc.options, a string

Table 177. dbms.logs.gc.rotation.keep_number

Description	Number of GC logs to keep.
Valid values	dbms.logs.gc.rotation.keep_number, an integer
Default value	`0`

Table 178. dbms.logs.gc.rotation.size

Description	Size of each GC log that is kept.
Valid values	dbms.logs.gc.rotation.size, a long

Table 179. dbms.logs.http.enabled

Description	Enable HTTP request logging.
Valid values	dbms.logs.http.enabled, a boolean
Default value	`false`

Table 180. dbms.logs.http.path

Description	Path to HTTP request log.
Valid values	dbms.logs.http.path, a path. If relative it is resolved from dbms.directories.logs
Default value	`http.log`

Table 181. dbms.logs.http.rotation.keep_number

Description	Number of HTTP logs to keep.
Valid values	dbms.logs.http.rotation.keep_number, an integer
Default value	`5`

Table 182. dbms.logs.http.rotation.size

Description	Size of each HTTP log that is kept.
Valid values	dbms.logs.http.rotation.size, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is in the range `0` to `9223372036854775807`
Default value	`20971520`

Table 183. dbms.logs.query.allocation_logging_enabled

Description	Log allocated bytes for the executed queries being logged. The logged number is cumulative over the duration of the query, i.e. for memory intense or long-running queries the value may be larger than the current memory allocation. Requires `<<config_dbms.track_query_allocation,dbms.track_query_allocation>>=true`
Valid values	dbms.logs.query.allocation_logging_enabled, a boolean
Dynamic	true
Default value	`false`

Table 184. dbms.logs.query.enabled

Description	Log executed queries. Valid values are 'OFF', 'INFO' & 'VERBOSE'. OFF: no logging. INFO: log queries at the end of execution, that take longer than the configured threshold, <<config_dbms.logs.query.threshold,dbms.logs.query.threshold>>. VERBOSE: log queries at the start and end of execution, regardless of <<config_dbms.logs.query.threshold,dbms.logs.query.threshold>>. Log entries are by default written to the file __query.log__ located in the Logs directory. For location of the Logs directory, see <<file-locations>>. This feature is available in the Neo4j Enterprise Edition.
Valid values	dbms.logs.query.enabled, one of [OFF, INFO, VERBOSE]

Dynamic	true
Default value	`VERBOSE`

Table 185. dbms.logs.query.page_logging_enabled

Description	Log page hits and page faults for the executed queries being logged.
Valid values	dbms.logs.query.page_logging_enabled, a boolean
Dynamic	true
Default value	`false`

Table 186. dbms.logs.query.parameter_logging_enabled

Description	Log parameters for the executed queries being logged.
Valid values	dbms.logs.query.parameter_logging_enabled, a boolean
Dynamic	true
Default value	`true`

Table 187. dbms.logs.query.path

Description	Path to the query log file.
Valid values	dbms.logs.query.path, a path. If relative it is resolved from dbms.directories.logs
Default value	`query.log`

Table 188. dbms.logs.query.rotation.keep_number

Description	Maximum number of history files for the query log.
Valid values	dbms.logs.query.rotation.keep_number, an integer which is minimum `1`
Dynamic	true
Default value	`7`

Table 189. dbms.logs.query.rotation.size

Description	The file size in bytes at which the query log will auto-rotate. If set to zero then no rotation will occur. Accepts a binary suffix \`k\`, \`m\` or \`g\`.
Valid values	dbms.logs.query.rotation.size, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is in the range `0` to `9223372036854775807`

Dynamic	true
Default value	`20971520`

Table 190. dbms.logs.query.runtime_logging_enabled

Description	Logs which runtime that was used to run the query.
Valid values	dbms.logs.query.runtime_logging_enabled, a boolean
Dynamic	true
Default value	`false`

Table 191. dbms.logs.query.threshold

Description	If the execution of query takes more time than this threshold, the query is logged once completed - provided query logging is set to INFO. Defaults to 0 seconds, that is all queries are logged.
Valid values	dbms.logs.query.threshold, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Dynamic	true
Default value	`0ns`

Table 192. dbms.logs.query.time_logging_enabled

Description	Log detailed time information for the executed queries being logged. Requires \`<<config_dbms.track_query_cpu_time,dbms.track_query_cpu_time>>=true\`
Valid values	dbms.logs.query.time_logging_enabled, a boolean
Dynamic	true
Default value	`false`

Table 193. dbms.logs.security.level

Description	Security log level threshold.
Valid values	dbms.logs.security.level, one of [DEBUG, INFO, WARN, ERROR, NONE]
Default value	`INFO`

Table 194. dbms.logs.security.path

Description	Path to the security log file.

Valid values	dbms.logs.security.path, a path. If relative it is resolved from dbms.directories.logs
Default value	`security.log`

Table 195. dbms.logs.security.rotation.delay

Description	Minimum time interval after last rotation of the security log before it may be rotated again.
Valid values	dbms.logs.security.rotation.delay, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`5m`

Table 196. dbms.logs.security.rotation.keep_number

Description	Maximum number of history files for the security log.
Valid values	dbms.logs.security.rotation.keep_number, an integer which is minimum `1`
Default value	`7`

Table 197. dbms.logs.security.rotation.size

Description	Threshold for rotation of the security log.
Valid values	dbms.logs.security.rotation.size, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is in the range `0` to `9223372036854775807`
Default value	`20971520`

Table 198. dbms.logs.user.path

Description	Path to the user log file.
Valid values	dbms.logs.user.path, a path. If relative it is resolved from dbms.directories.logs
Default value	`neo4j.log`

Table 199. dbms.logs.user.rotation.delay

Description	Minimum time interval after last rotation of the user log before it may be rotated again.
Valid values	dbms.logs.user.rotation.delay, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`5m`

Table 200. dbms.logs.user.rotation.keep_number

Description	Maximum number of history files for the user log.
Valid values	dbms.logs.user.rotation.keep_number, an integer which is minimum `1`
Default value	`7`

Table 201. dbms.logs.user.rotation.size

Description	Threshold for rotation of the user log. If set to 0 log rotation is disabled.
Valid values	dbms.logs.user.rotation.size, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is in the range `0` to `9223372036854775807`
Default value	`0`

Table 202. dbms.logs.user.stdout_enabled

Description	Send user logs to the process stdout. If this is disabled then logs will instead be sent to the file __neo4j.log__ located in the logs directory. For location of the Logs directory, see <<file-locations>>.
Valid values	dbms.logs.user.stdout_enabled, a boolean
Default value	`true`

Table 203. dbms.max_databases

Description	The maximum number of databases.
Valid values	dbms.max_databases, a long which is minimum `2`
Default value	`100`

Table 204. dbms.memory.heap.initial_size

Description	Initial heap size. By default it is calculated based on available system resources. (valid units are \`k\`, \`K\`, \`m\`, \`M\`, \`g\`, \`G\`).
Valid values	dbms.memory.heap.initial_size, a string
Default value	

Table 205. dbms.memory.heap.max_size

Description	Maximum heap size. By default it is calculated based on available system resources. (valid units are \`k\`, \`K\`, \`m\`, \`M\`, \`g\`, \`G\`).
Valid values	dbms.memory.heap.max_size, a string
Default value	

Table 206. dbms.memory.pagecache.size

Description	The amount of memory to use for mapping the store files, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). If Neo4j is running on a dedicated server, then it is generally recommended to leave about 2-4 gigabytes for the operating system, give the JVM enough heap to hold all your transaction state and query context, and then leave the rest for the page cache. If no page cache memory is configured, then a heuristic setting is computed based on available system resources.
Valid values	dbms.memory.pagecache.size, a string

Table 207. dbms.memory.pagecache.swapper

Description	Specify which page swapper to use for doing paged IO. This is only used when integrating with proprietary storage technology.
Valid values	dbms.memory.pagecache.swapper, a string

Table 208. dbms.memory.pagecache.warmup.enable

Description	Page cache can be configured to perform usage sampling of loaded pages that can be used to construct active load profile. According to that profile pages can be reloaded on the restart, replication, etc. This setting allows disabling that behavior. This feature available in Neo4j Enterprise Edition.
Valid values	dbms.memory.pagecache.warmup.enable, a boolean
Default value	`true`

Table 209. dbms.memory.pagecache.warmup.preload

Description	+Page cache warmup can be configured to prefetch files, preferably when cache size is bigger than store size. Files to be prefetched can be filtered by 'dbms.memory.pagecache.warmup.preload.whitelist'. Enabling this disables warmup by profile
Valid values	dbms.memory.pagecache.warmup.preload, a boolean
Default value	`false`

Table 210. dbms.memory.pagecache.warmup.preload.whitelist

Description	Page cache warmup prefetch file whitelist regex. By default matches all files.
Valid values	dbms.memory.pagecache.warmup.preload.whitelist, a string
Default value	`.*`

Table 211. dbms.memory.pagecache.warmup.profile.interval

Description	The profiling frequency for the page cache. Accurate profiles allow the page cache to do active warmup after a restart, reducing the mean time to performance. This feature available in Neo4j Enterprise Edition.
Valid values	dbms.memory.pagecache.warmup.profile.interval, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`1m`

Table 212. dbms.mode

Description	Configure the operating mode of the database -- 'SINGLE' for stand-alone operation, 'CORE' for operating as a core member of a Causal Cluster, or 'READ_REPLICA' for operating as a read replica member of a Causal Cluster.
Valid values	dbms.mode, one of [SINGLE, CORE, READ_REPLICA]
Default value	`SINGLE`

Table 213. dbms.netty.ssl.provider

Description	Netty SSL provider.
Valid values	dbms.netty.ssl.provider, one of [JDK, OPENSSL, OPENSSL_REFCNT]
Default value	`JDK`

Table 214. dbms.query_cache_size

Description	The number of Cypher query execution plans that are cached.
Valid values	dbms.query_cache_size, an integer which is minimum `0`
Default value	`1000`

Table 215. dbms.read_only

Description	Only allow read operations from this Neo4j instance. This mode still requires write access to the directory for lock purposes.
Valid values	dbms.read_only, a boolean
Default value	`false`

Table 216. dbms.reconciler.max_backoff

Description	Defines the maximum amount of time to wait before retrying after the dbms fails to reconcile a database to its desired state.
Valid values	dbms.reconciler.max_backoff, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's') which is minimum `PT1M`

Default value	`1h`

Table 217. dbms.reconciler.may_retry

Description	Defines whether the dbms may retry reconciling a database to its desired state.
Valid values	dbms.reconciler.may_retry, a boolean
Default value	`false`

Table 218. dbms.reconciler.min_backoff

Description	Defines the minimum amount of time to wait before retrying after the dbms fails to reconcile a database to its desired state.
Valid values	dbms.reconciler.min_backoff, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's') which is minimum `PT1S`
Default value	`2s`

Table 219. dbms.record_format

Description	Database record format. Valid values: \`standard\`, \`high_limit\`. The \`high_limit\` format is available for Enterprise Edition only. It is required if you have a graph that is larger than 34 billion nodes, 34 billion relationships, or 68 billion properties. A change of the record format is irreversible. Certain operations may suffer from a performance penalty of up to 10%, which is why this format is not switched on by default.
Valid values	dbms.record_format, a string
Default value	

Table 220. dbms.recovery.fail_on_missing_files

Description	If \`true\`, Neo4j will abort recovery if logical log files are missing. Setting this to \`false\` will allow Neo4j to create new empty missing files for already existing database, but, the integrity of the database might be compromised.
Valid values	dbms.recovery.fail_on_missing_files, a boolean
Default value	`true`

Table 221. dbms.relationship_grouping_threshold

Description	Relationship count threshold for considering a node to be dense.
Valid values	dbms.relationship_grouping_threshold, an integer which is minimum `1`
Default value	`50`

Table 222. dbms.rest.transaction.idle_timeout

Description	Timeout for idle transactions in the REST endpoint.
Valid values	dbms.rest.transaction.idle_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`1m`

Table 223. dbms.routing_ttl

Description	How long callers should cache the response of the routing procedure `dbms.routing.getRoutingTable()`
Valid values	dbms.routing_ttl, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's') which is minimum `PT1S`
Default value	`5m`

Table 224. dbms.security.allow_csv_import_from_file_urls

Description	Determines if Cypher will allow using file URLs when loading data using `LOAD CSV`. Setting this value to `false` will cause Neo4j to fail `LOAD CSV` clauses that load data from the file system.
Valid values	dbms.security.allow_csv_import_from_file_urls, a boolean
Default value	`true`

Table 225. dbms.security.auth_cache_max_capacity

Description	The maximum capacity for authentication and authorization caches (respectively).
Valid values	dbms.security.auth_cache_max_capacity, an integer
Default value	`10000`

Table 226. dbms.security.auth_cache_ttl

Description	The time to live (TTL) for cached authentication and authorization info when using external auth providers (LDAP or plugin). Setting the TTL to 0 will disable auth caching. Disabling caching while using the LDAP auth provider requires the use of an LDAP system account for resolving authorization information.
Valid values	dbms.security.auth_cache_ttl, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`10m`

Table 227. dbms.security.auth_cache_use_ttl

Description	Enable time-based eviction of the authentication and authorization info cache for external auth providers (LDAP or plugin). Disabling this setting will make the cache live forever and only be evicted when `<<config_dbms.security.auth_cache_max_capacity,dbms.security.auth_cache_max_capacity>>` is exceeded.
Valid values	dbms.security.auth_cache_use_ttl, a boolean
Default value	`true`

Table 228. dbms.security.auth_enabled

Description	Enable auth requirement to access Neo4j.
Valid values	dbms.security.auth_enabled, a boolean
Default value	`true`

Table 229. dbms.security.auth_lock_time

Description	The amount of time user account should be locked after a configured number of unsuccessful authentication attempts. The locked out user will not be able to log in until the lock period expires, even if correct credentials are provided. Setting this configuration option to a low value is not recommended because it might make it easier for an attacker to brute force the password.
Valid values	dbms.security.auth_lock_time, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's') which is minimum `PT0S`
Default value	`5s`

Table 230. dbms.security.auth_max_failed_attempts

Description	The maximum number of unsuccessful authentication attempts before imposing a user lock for the configured amount of time.The locked out user will not be able to log in until the lock period expires, even if correct credentials are provided. Setting this configuration option to values less than 3 is not recommended because it might make it easier for an attacker to brute force the password.
Valid values	dbms.security.auth_max_failed_attempts, an integer which is minimum `0`
Default value	`3`

Table 231. dbms.security.authentication_providers

Description	A list of security authentication providers containing the users and roles. This can be any of the built-in `native` or `ldap` providers, or it can be an externally provided plugin, with a custom name prefixed by `plugin-`, i.e. `plugin-<AUTH_PROVIDER_NAME>`. They will be queried in the given order when login is attempted.

Valid values	dbms.security.authentication_providers, a ',' separated list with elements of type 'a string'.
Default value	`native`

Table 232. dbms.security.authorization_providers

Description	A list of security authorization providers containing the users and roles. This can be any of the built-in `native` or `ldap` providers, or it can be an externally provided plugin, with a custom name prefixed by `plugin-`, i.e. `plugin-<AUTH_PROVIDER_NAME>`. They will be queried in the given order when login is attempted.
Valid values	dbms.security.authorization_providers, a ',' separated list with elements of type 'a string'.
Default value	`native`

Table 233. dbms.security.causal_clustering_status_auth_enabled

Description	Require authorization for access to the Causal Clustering status endpoints.
Valid values	dbms.security.causal_clustering_status_auth_enabled, a boolean
Default value	`true`

Table 234. dbms.security.http_access_control_allow_origin

Description	Value of the Access-Control-Allow-Origin header sent over any HTTP or HTTPS connector. This defaults to '*', which allows broadest compatibility. Note that any URI provided here limits HTTP/HTTPS access to that URI only.
Valid values	dbms.security.http_access_control_allow_origin, a string
Default value	`*`

Table 235. dbms.security.http_auth_whitelist

Description	Defines a whitelist of http paths where Neo4j authentication is not required.
Valid values	dbms.security.http_auth_whitelist, a ',' separated list with elements of type 'a string'.
Default value	`/,/browser.*`

Table 236. dbms.security.http_strict_transport_security

Description	Value of the HTTP Strict-Transport-Security (HSTS) response header. This header tells browsers that a webpage should only be accessed using HTTPS instead of HTTP. It is attached to every HTTPS response. Setting is not set by default so 'Strict-Transport-Security' header is not sent. Value is expected to contain directives like 'max-age', 'includeSubDomains' and 'preload'.

Valid values	dbms.security.http_strict_transport_security, a string

Table 237. dbms.security.ldap.authentication.cache_enabled

Description	Determines if the result of authentication via the LDAP server should be cached or not. Caching is used to limit the number of LDAP requests that have to be made over the network for users that have already been authenticated successfully. A user can be authenticated against an existing cache entry (instead of via an LDAP server) as long as it is alive (see \`<<config_dbms.security.auth_cache_ttl,dbms.security.auth_cache_ttl>>\`). An important consequence of setting this to \`true\` is that Neo4j then needs to cache a hashed version of the credentials in order to perform credentials matching. This hashing is done using a cryptographic hash function together with a random salt. Preferably a conscious decision should be made if this method is considered acceptable by the security standards of the organization in which this Neo4j instance is deployed.
Valid values	dbms.security.ldap.authentication.cache_enabled, a boolean
Default value	`true`

Table 238. dbms.security.ldap.authentication.mechanism

Description	LDAP authentication mechanism. This is one of \`simple\` or a SASL mechanism supported by JNDI, for example \`DIGEST-MD5\`. \`simple\` is basic username and password authentication and SASL is used for more advanced mechanisms. See RFC 2251 LDAPv3 documentation for more details.
Valid values	dbms.security.ldap.authentication.mechanism, a string
Default value	`simple`

Table 239. dbms.security.ldap.authentication.use_samaccountname

Description	Perform authentication with sAMAccountName instead of DN. Using this setting requires \`<<config_dbms.security.ldap.authorization.system_username,dbms.security.ldap.authorization.system_username>>\` and <<config_dbms.security.ldap.authorization.system_password,dbms.security.ldap.authorization.system_password>> to be used since there is no way to log in through ldap directly with the sAMAccountName, instead the login name will be resolved to a DN that will be used to log in with.
Valid values	dbms.security.ldap.authentication.use_samaccountname, a boolean
Default value	`false`

Table 240. dbms.security.ldap.authentication.user_dn_template

Description	LDAP user DN template. An LDAP object is referenced by its distinguished name (DN), and a user DN is an LDAP fully-qualified unique user identifier. This setting is used to generate an LDAP DN that conforms with the LDAP directory's schema from the user principal that is submitted with the authentication token when logging in. The special token {0} is a placeholder where the user principal will be substituted into the DN string.
Valid values	dbms.security.ldap.authentication.user_dn_template, a string
Default value	`uid={0},ou=users,dc=example,dc=com`

Table 241. dbms.security.ldap.authorization.group_membership_attributes

Description	A list of attribute names on a user object that contains groups to be used for mapping to roles when LDAP authorization is enabled.
Valid values	dbms.security.ldap.authorization.group_membership_attributes, a ',' separated list with elements of type 'a string'.
Default value	`memberOf`

Table 242. dbms.security.ldap.authorization.group_to_role_mapping

Description	+An authorization mapping from LDAP group names to Neo4j role names. The map should be formatted as a semicolon separated list of key-value pairs, where the key is the LDAP group name and the value is a comma separated list of corresponding role names. For example: group1=role1;group2=role2;group3=role3,role4,role5 You could also use whitespaces and quotes around group names to make this mapping more readable, for example:
	`dbms.security.ldap.authorization.group_to_role_mapping=\`
	`         "cn=Neo4j Read Only,cn=users,dc=example,dc=com"      = reader;    \`
	`         "cn=Neo4j Read-Write,cn=users,dc=example,dc=com"     = publisher; \`
	`         "cn=Neo4j Schema Manager,cn=users,dc=example,dc=com" = architect; \`
	`         "cn=Neo4j Administrator,cn=users,dc=example,dc=com"  = admin`
	Deprecated: This will be replaced by dynamic configuration in the system graph in 4.0, including a migration step for the existing setting value.
	+

Valid values	dbms.security.ldap.authorization.group_to_role_mapping, a string
Deprecated	The `dbms.security.ldap.authorization.group_to_role_mapping` configuration setting has been deprecated.

Table 243. dbms.security.ldap.authorization.system_password

Description	An LDAP system account password to use for authorization searches when `<<config_dbms.security.ldap.authorization.use_system_account,dbms.security.ldap.authorization.use_system_account>>` is `true`.

Valid values	dbms.security.ldap.authorization.system_password, a secure string

Table 244. dbms.security.ldap.authorization.system_username

Description	An LDAP system account username to use for authorization searches when `<<config_dbms.security.ldap.authorization.use_system_account,dbms.security.ldap.authorization.use_system_account>>` is `true`. Note that the `<<config_dbms.security.ldap.authentication.user_dn_template,dbms.security.ldap.authentication.user_dn_template>>` will not be applied to this username, so you may have to specify a full DN.
Valid values	dbms.security.ldap.authorization.system_username, a string

Table 245. dbms.security.ldap.authorization.use_system_account

Description	Perform LDAP search for authorization info using a system account instead of the user's own account. If this is set to `false` (default), the search for group membership will be performed directly after authentication using the LDAP context bound with the user's own account. The mapped roles will be cached for the duration of `<<config_dbms.security.auth_cache_ttl,dbms.security.auth_cache_ttl>>`, and then expire, requiring re-authentication. To avoid frequently having to re-authenticate sessions you may want to set a relatively long auth cache expiration time together with this option. NOTE: This option will only work if the users are permitted to search for their own group membership attributes in the directory. If this is set to `true`, the search will be performed using a special system account user with read access to all the users in the directory. You need to specify the username and password using the settings `<<config_dbms.security.ldap.authorization.system_username,dbms.security.ldap.authorization.system_username>>` and `<<config_dbms.security.ldap.authorization.system_password,dbms.security.ldap.authorization.system_password>>` with this option. Note that this account only needs read access to the relevant parts of the LDAP directory and does not need to have access rights to Neo4j, or any other systems.
Valid values	dbms.security.ldap.authorization.use_system_account, a boolean
Default value	`false`

Table 246. dbms.security.ldap.authorization.user_search_base

Description	The name of the base object or named context to search for user objects when LDAP authorization is enabled. A common case is that this matches the last part of `<<config_dbms.security.ldap.authentication.user_dn_template,dbms.security.ldap.authentication.user_dn_template>>`.
Valid values	dbms.security.ldap.authorization.user_search_base, a string
Default value	`ou=users,dc=example,dc=com`

Table 247. dbms.security.ldap.authorization.user_search_filter

Description	The LDAP search filter to search for a user principal when LDAP authorization is enabled. The filter should contain the placeholder token {0} which will be substituted for the user principal.
Valid values	dbms.security.ldap.authorization.user_search_filter, a string
Default value	`(&(objectClass=*)(uid={0}))`

Table 248. dbms.security.ldap.connection_timeout

Description	The timeout for establishing an LDAP connection. If a connection with the LDAP server cannot be established within the given time the attempt is aborted. A value of 0 means to use the network protocol's (i.e., TCP's) timeout value.
Valid values	dbms.security.ldap.connection_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`30s`

Table 249. dbms.security.ldap.host

Description	URL of LDAP server to use for authentication and authorization. The format of the setting is `<protocol>://<hostname>:<port>`, where hostname is the only required field. The supported values for protocol are `ldap` (default) and `ldaps`. The default port for `ldap` is 389 and for `ldaps` 636. For example: `ldaps://ldap.example.com:10389`. You may want to consider using STARTTLS (`<<config_dbms.security.ldap.use_starttls,dbms.security.ldap.use_starttls>>`) instead of LDAPS for secure connections, in which case the correct protocol is `ldap`.
Valid values	dbms.security.ldap.host, a string
Default value	`localhost`

Table 250. dbms.security.ldap.read_timeout

Description	The timeout for an LDAP read request (i.e. search). If the LDAP server does not respond within the given time the request will be aborted. A value of 0 means wait for a response indefinitely.
Valid values	dbms.security.ldap.read_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`30s`

Table 251. dbms.security.ldap.referral

Description	The LDAP referral behavior when creating a connection. This is one of `follow`, `ignore` or `throw`. * `follow` automatically follows any referrals * `ignore` ignores any referrals * `throw` throws an exception, which will lead to authentication failure.
Valid values	dbms.security.ldap.referral, a string

Default value	`follow`

Table 252. dbms.security.ldap.use_starttls

Description	Use secure communication with the LDAP server using opportunistic TLS. First an initial insecure connection will be made with the LDAP server, and a STARTTLS command will be issued to negotiate an upgrade of the connection to TLS before initiating authentication.
Valid values	dbms.security.ldap.use_starttls, a boolean
Default value	`false`

Table 253. dbms.security.log_successful_authentication

Description	Set to log successful authentication events to the security log. If this is set to \`false\` only failed authentication events will be logged, which could be useful if you find that the successful events spam the logs too much, and you do not require full auditing capability.
Valid values	dbms.security.log_successful_authentication, a boolean
Default value	`true`

Table 254. dbms.security.procedures.default_allowed

Description	The default role that can execute all procedures and user-defined functions that are not covered by the \`<<config_dbms.security.procedures.roles,dbms.security.procedures.roles>>\` setting. If the \`\`dbms.security.procedures.default_allowed\`\` setting is the empty string (default), procedures will be executed according to the same security rules as normal Cypher statements. Deprecated: This will be replaced by dynamic configuration in the system graph in 4.0, including a migration step for the existing setting value.
Valid values	dbms.security.procedures.default_allowed, a string
Default value	
Deprecated	The `dbms.security.procedures.default_allowed` configuration setting has been deprecated.

Table 255. dbms.security.procedures.roles

Description	This provides a finer level of control over which roles can execute procedures than the `<<config_dbms.security.procedures.default_allowed,dbms.security.procedures.default_allowed>>` setting. For example: `+dbms.security.procedures.roles=apoc.convert.*:reader;apoc.load.json*:writer;apoc.trigger.add:TriggerHappy` will allow the role `reader` to execute all procedures in the `apoc.convert` namespace, the role `writer` to execute all procedures in the `apoc.load` namespace that starts with `json` and the role `TriggerHappy` to execute the specific procedure `apoc.trigger.add`. Procedures not matching any of these patterns will be subject to the `dbms.security.procedures.default_allowed` setting. Deprecated: This will be replaced by dynamic configuration in the system graph in 4.0, including a migration step for the existing setting value.+
Valid values	dbms.security.procedures.roles, a string
Default value	
Deprecated	The `dbms.security.procedures.roles` configuration setting has been deprecated.

Table 256. dbms.security.procedures.unrestricted

Description	A list of procedures and user defined functions (comma separated) that are allowed full access to the database. The list may contain both fully-qualified procedure names, and partial names with the wildcard '*'. Note that this enables these procedures to bypass security. Use with caution.
Valid values	dbms.security.procedures.unrestricted, a ',' separated list with elements of type 'a string'.
Default value	

Table 257. dbms.security.procedures.whitelist

Description	A list of procedures (comma separated) that are to be loaded. The list may contain both fully-qualified procedure names, and partial names with the wildcard '*'. If this setting is left empty no procedures will be loaded.
Valid values	dbms.security.procedures.whitelist, a ',' separated list with elements of type 'a string'.
Default value	`*`

Table 258. dbms.security.property_level.enabled

Description	This has been replaced by privilege management on roles. Setting it to true will prevent the server from starting.
Valid values	dbms.security.property_level.enabled, a boolean
Default value	`false`

Deprecated	The `dbms.security.property_level.enabled` configuration setting has been deprecated.

Table 259. dbms.shutdown_transaction_end_timeout

Description	The maximum amount of time to wait for running transactions to complete before allowing initiated database shutdown to continue.
Valid values	dbms.shutdown_transaction_end_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`10s`

Table 260. dbms.threads.worker_count

Description	Number of Neo4j worker threads. This setting is only valid for REST, and does not influence bolt-server. It sets the amount of worker threads for the Jetty server used by neo4j-server. This option can be tuned when you plan to execute multiple, concurrent REST requests, with the aim of getting more throughput from the database. Your OS might enforce a lower limit than the maximum value specified here.
Valid values	dbms.threads.worker_count, an integer which is in the range `1` to `44738`
Default value	`Number of available processors, or 500 for machines which have more than 500 processors.`

Table 261. dbms.track_query_allocation

Description	Enables or disables tracking of how many bytes are allocated by the execution of a query. If enabled, calling \`dbms.listQueries\` will display the allocated bytes. If enabled, the maximum allocated bytes of a query can be limited using \`<<config_cypher.query_max_allocations,cypher.query_max_allocations>>\`. This can also be logged in the query log by using \`<<config_dbms.logs.query.allocation_logging_enabled,dbms.logs.query.allocation_logging_enabled>>\`.
Valid values	dbms.track_query_allocation, a boolean
Dynamic	true
Default value	`false`

Table 262. dbms.track_query_cpu_time

Description	Enables or disables tracking of how much time a query spends actively executing on the CPU. Calling \`dbms.listQueries\` will display the time. This can also be logged in the query log by using \`log_queries_detailed_time_logging_enabled\`.
Valid values	dbms.track_query_cpu_time, a boolean

Dynamic	true
Default value	`false`

Table 263. dbms.transaction.bookmark_ready_timeout

Description	The maximum amount of time to wait for the database state represented by the bookmark.
Valid values	dbms.transaction.bookmark_ready_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's') which is minimum `PT1S`
Default value	`30s`

Table 264. dbms.transaction.concurrent.maximum

Description	The maximum number of concurrently running transactions. If set to 0, limit is disabled.
Valid values	dbms.transaction.concurrent.maximum, an integer
Dynamic	true
Default value	`1000`

Table 265. dbms.transaction.monitor.check.interval

Description	Configures the time interval between transaction monitor checks. Determines how often monitor thread will check transaction for timeout.
Valid values	dbms.transaction.monitor.check.interval, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`2s`

Table 266. dbms.transaction.sampling.percentage

Description	Transaction sampling percentage.
Valid values	dbms.transaction.sampling.percentage, an integer which is in the range `1` to `100`
Dynamic	true
Default value	`5`

Table 267. dbms.transaction.timeout

Description	The maximum time interval of a transaction within which it should be completed.

Valid values	dbms.transaction.timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Dynamic	true
Default value	`0ns`

Table 268. dbms.transaction.tracing.level

Description	Transaction creation tracing level.
Valid values	dbms.transaction.tracing.level, one of [DISABLED, SAMPLE, ALL]
Dynamic	true
Default value	`DISABLED`

Table 269. dbms.tx_log.preallocate

Description	Specify if Neo4j should try to preallocate logical log file in advance.
Valid values	dbms.tx_log.preallocate, a boolean
Dynamic	true
Default value	`true`

Table 270. dbms.tx_log.rotation.retention_policy

Description	Make Neo4j keep the logical transaction logs for being able to backup the database. Can be used for specifying the threshold to prune logical logs after. For example "10 days" will prune logical logs that only contains transactions older than 10 days from the current time, or "100k txs" will keep the 100k latest transactions and prune any older transactions.
Valid values	dbms.tx_log.rotation.retention_policy, a string which matches the pattern `^(true|keep_all|false|keep_none|(\d+[KkMmGg]?((files|size|txs|entries|hours|days))))$` (must be `true`, `false` or of format `<number><optional unit> <type>`. Valid units are `k`, `M` and `G`. Valid types are `files`, `size`, `txs`, `entries`, `hours` and `days`. For example, `100M size` will limiting logical log space on disk to 100Mb, or `200k txs` will limiting the number of transactions to keep to 200 000)
Dynamic	true
Default value	`7 days`

Table 271. dbms.tx_log.rotation.size

Description	+Specifies at which file size the logical log will auto-rotate. Minimum accepted value is 128 KiB.

Valid values	dbms.tx_log.rotation.size, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `131072`
Dynamic	true
Default value	`262144000`

Table 272. dbms.tx_state.max_off_heap_memory

Description	The maximum amount of off-heap memory that can be used to store transaction state data; it's a total amount of memory shared across all active transactions. Zero means 'unlimited'. Used when <<config_dbms.tx_state.memory_allocation,dbms.tx_state.memory_allocation>> is set to 'OFF_HEAP'.
Valid values	dbms.tx_state.max_off_heap_memory, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `0`
Default value	`2147483648`

Table 273. dbms.tx_state.memory_allocation

Description	Defines whether memory for transaction state should be allocated on- or off-heap.
Valid values	dbms.tx_state.memory_allocation, one of [ON_HEAP, OFF_HEAP]
Default value	`OFF_HEAP`

Table 274. dbms.tx_state.off_heap.block_cache_size

Description	Defines the size of the off-heap memory blocks cache. The cache will contain this number of blocks for each block size that is power of two. Thus, maximum amount of memory used by blocks cache can be calculated as 2 * <<config_dbms.tx_state.off_heap.max_cacheable_block_size,dbms.tx_state.off_heap.max_cacheable_block_size>> * `dbms.tx_state.off_heap.block_cache_size`
Valid values	dbms.tx_state.off_heap.block_cache_size, an integer which is minimum `16`
Default value	`128`

Table 275. dbms.tx_state.off_heap.max_cacheable_block_size

Description	Defines the maximum size of an off-heap memory block that can be cached to speed up allocations for transaction state data. The value must be a power of 2.
Valid values	dbms.tx_state.off_heap.max_cacheable_block_size, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is minimum `4096` and is power of 2
Default value	`524288`

Table 276. dbms.unmanaged_extension_classes

Description	Comma-separated list of <classname>=<mount point> for unmanaged extensions.
Valid values	dbms.unmanaged_extension_classes, a ',' separated list with elements of type '<classname>=<mount point> string'.
Default value	

Table 277. dbms.windows_service_name

Description	Name of the Windows Service.
Valid values	dbms.windows_service_name, a string
Default value	`neo4j`

Table 278. fabric.database.name

Description	Name of the Fabric database. Only one Fabric database is currently supported per Neo4j instance.
Valid values	fabric.database.name, A valid database name. Containing only alphabetic characters, numbers, dots and dashes, with a length between 3 and 63 characters. It should be starting with an alphabetic character but not with the name 'system'.

Table 279. fabric.driver.api

Description	Determines which driver API will be used. ASYNC must be used when the remote instance is 3.5.
Valid values	fabric.driver.api, one of [RX, ASYNC]
Default value	`RX`

Table 280. fabric.driver.connection.connect_timeout

Description	Socket connection timeout. A timeout of zero is treated as an infinite timeout and will be bound by the timeout configured on the operating system level.
Valid values	fabric.driver.connection.connect_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`5s`

Table 281. fabric.driver.connection.max_lifetime

Description	Pooled connections older than this threshold will be closed and removed from the pool. Setting this option to a low value will cause a high connection churn and might result in a performance hit. It is recommended to set maximum lifetime to a slightly smaller value than the one configured in network equipment (load balancer, proxy, firewall, etc. can also limit maximum connection lifetime). Zero and negative values result in lifetime not being checked.
Valid values	fabric.driver.connection.max_lifetime, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`1h`

Table 282. fabric.driver.connection.pool.acquisition_timeout

Description	Maximum amount of time spent attempting to acquire a connection from the connection pool. This timeout only kicks in when all existing connections are being used and no new connections can be created because maximum connection pool size has been reached. Error is raised when connection can't be acquired within configured time. Negative values are allowed and result in unlimited acquisition timeout. Value of 0 is allowed and results in no timeout and immediate failure when connection is unavailable.
Valid values	fabric.driver.connection.pool.acquisition_timeout, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`1m`

Table 283. fabric.driver.connection.pool.idle_test

Description	Pooled connections that have been idle in the pool for longer than this timeout will be tested before they are used again, to ensure they are still alive. If this option is set too low, an additional network call will be incurred when acquiring a connection, which causes a performance hit. If this is set high, no longer live connections might be used which might lead to errors. Hence, this parameter tunes a balance between the likelihood of experiencing connection problems and performance Normally, this parameter should not need tuning. Value 0 means connections will always be tested for validity.
Valid values	fabric.driver.connection.pool.idle_test, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`No connection liveliness check is done by default.`

Table 284. fabric.driver.connection.pool.max_size

Description	Maximum total number of connections to be managed by a connection pool. The limit is enforced for a combination of a host and user. Negative values are allowed and result in unlimited pool. Value of 0is not allowed.
Valid values	fabric.driver.connection.pool.max_size, an integer
Default value	`Unlimited`

Table 285. fabric.driver.logging.level

Description	Sets level for driver internal logging.
Valid values	fabric.driver.logging.level, one of [DEBUG, INFO, WARN, ERROR, NONE]
Default value	`Value of dbms.logs.debug.level`

Table 286. fabric.routing.servers

Description	A comma-separated list of Fabric instances that form a routing group. A driver will route transactions to available routing group members. A Fabric instance is represented by its Bolt connector address.
Valid values	fabric.routing.servers, a ',' separated list with elements of type 'a socket address'.
Default value	`localhost:7687`

Table 287. fabric.routing.ttl

Description	The time to live (TTL) of a routing table for fabric routing group.
Valid values	fabric.routing.ttl, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`1m`

Table 288. fabric.stream.buffer.low_watermark

Description	Number of records in prefetching buffer that will trigger prefetching again. This is strongly related to <<config_fabric.stream.buffer.size,fabric.stream.buffer.size>>
Valid values	fabric.stream.buffer.low_watermark, an integer
Default value	`300`

Table 289. fabric.stream.buffer.size

Description	Maximal size of a buffer used for pre-fetching result records of remote queries. To compensate for latency to remote databases, the Fabric execution engine pre-fetches records needed for local executions. This limit is enforced per fabric query. If a fabric query uses multiple remote stream at the same time, this setting represents the maximal number of pre-fetched records counted together for all such remote streams.
Valid values	fabric.stream.buffer.size, an integer
Default value	`1000`

Table 290. fabric.stream.concurrency

Description	Maximal concurrency within Fabric queries. Limits the number of iterations of each subquery that are executed concurrently. Higher concurrency may consume more memory and network resources simultaneously, while lower concurrency may force sequential execution, requiring more time.
Valid values	fabric.stream.concurrency, an integer
Default value	`The number of remote graphs`

Table 291. metrics.bolt.messages.enabled

Description	Enable reporting metrics about Bolt Protocol message processing.
Valid values	metrics.bolt.messages.enabled, a boolean
Default value	`true`

Table 292. metrics.csv.enabled

Description	Set to \`true\` to enable exporting metrics to CSV files.
Valid values	metrics.csv.enabled, a boolean
Default value	`true`

Table 293. metrics.csv.interval

Description	The reporting interval for the CSV files. That is, how often new rows with numbers are appended to the CSV files.
Valid values	metrics.csv.interval, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`3s`

Table 294. metrics.csv.rotation.keep_number

Description	Maximum number of history files for the csv files.
Valid values	metrics.csv.rotation.keep_number, an integer which is minimum `1`
Default value	`7`

Table 295. metrics.csv.rotation.size

Description	The file size in bytes at which the csv files will auto-rotate. If set to zero then no rotation will occur. Accepts a binary suffix \`k\`, \`m\` or \`g\`.
Valid values	metrics.csv.rotation.size, a byte size (valid multipliers are `k`, `m`, `g`, `K`, `M`, `G`) which is in the range `0` to `9223372036854775807`
Default value	`10485760`

Table 296. metrics.cypher.replanning.enabled

Description	Enable reporting metrics about number of occurred replanning events.
Valid values	metrics.cypher.replanning.enabled, a boolean
Default value	`true`

Table 297. metrics.enabled

Description	Enable metrics. Setting this to \`false\` will to turn off all metrics.
Valid values	metrics.enabled, a boolean
Default value	`true`

Table 298. metrics.graphite.enabled

Description	Set to \`true\` to enable exporting metrics to Graphite.
Valid values	metrics.graphite.enabled, a boolean
Default value	`false`

Table 299. metrics.graphite.interval

Description	The reporting interval for Graphite. That is, how often to send updated metrics to Graphite.
Valid values	metrics.graphite.interval, a duration (Valid units are: 'ns', 'ms', 's', 'm' and 'h'; default unit is 's')
Default value	`3s`

Table 300. metrics.graphite.server

Description	The hostname or IP address of the Graphite server.
Valid values	metrics.graphite.server, a hostname and port
Default value	`:2003`

Table 301. metrics.jmx.enabled

Description	Set to \`true\` to enable the JMX metrics endpoint.
Valid values	metrics.jmx.enabled, a boolean
Default value	`true`

Table 302. metrics.jvm.buffers.enabled

Description	Enable reporting metrics about the buffer pools.
Valid values	metrics.jvm.buffers.enabled, a boolean
Default value	`true`

Table 303. metrics.jvm.file.descriptors.enabled

Description	Enable reporting metrics about the number of open file descriptors.
Valid values	metrics.jvm.file.descriptors.enabled, a boolean
Default value	`true`

Table 304. metrics.jvm.gc.enabled

Description	Enable reporting metrics about the duration of garbage collections.
Valid values	metrics.jvm.gc.enabled, a boolean
Default value	`true`

Table 305. metrics.jvm.heap.enabled

Description	Enable reporting metrics about the heap memory usage.
Valid values	metrics.jvm.heap.enabled, a boolean
Default value	`true`

Table 306. metrics.jvm.memory.enabled

Description	Enable reporting metrics about the memory usage.
Valid values	metrics.jvm.memory.enabled, a boolean
Default value	`true`

Table 307. metrics.jvm.threads.enabled

Description	Enable reporting metrics about the current number of threads running.
Valid values	metrics.jvm.threads.enabled, a boolean
Default value	`true`

Table 308. metrics.neo4j.causal_clustering.enabled

Description	Enable reporting metrics about Causal Clustering mode.
Valid values	metrics.neo4j.causal_clustering.enabled, a boolean

Default value	`true`

Table 309. metrics.neo4j.checkpointing.enabled

Description	Enable reporting metrics about Neo4j check pointing; when it occurs and how much time it takes to complete.
Valid values	metrics.neo4j.checkpointing.enabled, a boolean
Default value	`true`

Table 310. metrics.neo4j.counts.enabled

Description	Enable reporting metrics about approximately how many entities are in the database; nodes, relationships, properties, etc.
Valid values	metrics.neo4j.counts.enabled, a boolean
Default value	`true`

Table 311. metrics.neo4j.data.counts.enabled

Description	Enable reporting metrics about number of entities in the database.
Valid values	metrics.neo4j.data.counts.enabled, a boolean
Default value	`true`

Table 312. metrics.neo4j.logs.enabled

Description	Enable reporting metrics about the Neo4j transaction logs.
Valid values	metrics.neo4j.logs.enabled, a boolean
Default value	`true`

Table 313. metrics.neo4j.pagecache.enabled

Description	Enable reporting metrics about the Neo4j page cache; page faults, evictions, flushes, exceptions, etc.
Valid values	metrics.neo4j.pagecache.enabled, a boolean
Default value	`true`

Table 314. metrics.neo4j.server.enabled

Description	Enable reporting metrics about Server threading info.
Valid values	metrics.neo4j.server.enabled, a boolean
Default value	`true`

Table 315. metrics.neo4j.size.enabled

Description	Enable reporting metrics about the store size of each database.
Valid values	metrics.neo4j.size.enabled, a boolean
Default value	`true`

Table 316. metrics.neo4j.tx.enabled

Description	Enable reporting metrics about transactions; number of transactions started, committed, etc.
Valid values	metrics.neo4j.tx.enabled, a boolean
Default value	`true`

Table 317. metrics.prefix

Description	A common prefix for the reported metrics field names.
Valid values	metrics.prefix, a string
Default value	`neo4j`

Table 318. metrics.prometheus.enabled

Description	Set to \`true\` to enable the Prometheus endpoint.
Valid values	metrics.prometheus.enabled, a boolean
Default value	`false`

Table 319. metrics.prometheus.endpoint

Description	The hostname and port to use as Prometheus endpoint.
Valid values	metrics.prometheus.endpoint, a hostname and port
Default value	`localhost:2004`

A.2. Built-in procedures

This section contains a reference of Neo4j built-in procedures.

This section includes:

- Procedures, editions and modes
- Procedures available in standalone Neo4j Enterprise Edition
- Procedures available in Neo4j Community Edition

A.2.1. Procedures, editions and modes

The procedures available depends on the type of installation. Enterprise Edition provides a fuller set of procedures than Community Edition. Cluster members have procedures that are not available in standalone mode.

The cluster-specific procedures are not included in this reference, instead see Procedures for monitoring a Causal Cluster. To check which procedures are available in your Neo4j instance, use the `dbms.procedures()` procedure.

Example 109. List available procedures

To list the procedures available on your particular installation, run the following command in Neo4j Browser or in Cypher Shell:

```
CALL dbms.procedures()
```

A.2.2. Enterprise Edition procedures

Table 320. Enterprise Edition procedures

Name	Description	Signature	Mode	Roles
db.awaitIndex()	Wait for an index to come online (for example: CALL db.awaitIndex("My Index", 300)).	`db.awaitIndex(indexName :: STRING?, timeOutSeconds = 300 :: INTEGER?) :: VOID`	`READ`	N/A
db.awaitIndexes()	Wait for all indexes to come online (for example: CALL db.awaitIndexes(300)).	`db.awaitIndexes(timeOutSeconds = 300 :: INTEGER?) :: VOID`	`READ`	N/A

Name	Description	Signature	Mode	Roles
db.checkpoint()	Initiate and wait for a new check point, or wait any already on-going check point to complete. Note that this temporarily disables the `dbms.checkpoint.iops.limit` setting in order to make the check point complete faster. This might cause transaction throughput to degrade slightly, due to increased IO load.	`db.checkpoint() :: (success :: BOOLEAN?, message :: STRING?)`	`DBMS`	N/A
db.clearQueryCaches()	Clears all query caches.	`db.clearQueryCaches() :: (value :: STRING?)`	`DBMS`	N/A
db.constraints()	List all constraints in the database.	`db.constraints() :: (name :: STRING?, description :: STRING?)`	`READ`	N/A
db.createIndex()	Create a named schema index with specified index provider and configuration (optional). Yield: name, labels, properties, providerName, status	`db.createIndex(indexName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)`	`SCHEMA`	N/A
db.createLabel()	Create a label	`db.createLabel(newLabel :: STRING?) :: VOID`	`WRITE`	N/A
db.createNodeKey()	Create a named node key constraint. Backing index will use specified index provider and configuration (optional). Yield: name, labels, properties, providerName, status	`db.createNodeKey(constraintName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)`	`SCHEMA`	N/A

Name	Description	Signature	Mode	Roles
db.createProperty()	Create a Property	`db.createProperty(newProperty :: STRING?) :: VOID`	`WRITE`	N/A
db.createRelationshipType()	Create a RelationshipType	`db.createRelationshipType(newRelationshipType :: STRING?) :: VOID`	`WRITE`	N/A
db.createUniquePropertyConstraint()	Create a named unique property constraint. Backing index will use specified index provider and configuration (optional). Yield: name, labels, properties, providerName, status	`db.createUniquePropertyConstraint(constraintName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)`	`SCHEMA`	N/A
db.index.fulltext.awaitEventuallyConsistentIndexRefresh()	Wait for the updates from recently committed transactions to be applied to any eventually-consistent full-text indexes.	`db.index.fulltext.awaitEventuallyConsistentIndexRefresh() :: VOID`	`READ`	N/A

Name	Description	Signature	Mode	Roles
db.index.fulltext.createNodeIndex()	Create a node full-text index for the given labels and properties. The optional 'config' map parameter can be used to supply settings to the index. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the `db.index.fulltext.listAvailableAnalyzers` procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread.	`db.index.fulltext.createNodeIndex(indexName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, config = {} :: MAP?) :: VOID`	`SCHEMA`	N/A

Name	Description	Signature	Mode	Roles
db.index.fulltext.createRelationshipIndex()	Create a relationship full-text index for the given relationship types and properties. The optional 'config' map parameter can be used to supply settings to the index. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the `db.index.fulltext.listAvailableAnalyzers` procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread.	`db.index.fulltext.createRelationshipIndex(indexName :: STRING?, relationshipTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, config = {} :: MAP?) :: VOID`	`SCHEMA`	N/A
db.index.fulltext.drop()	Drop the specified index.	`db.index.fulltext.drop(indexName :: STRING?) :: VOID`	`SCHEMA`	N/A
db.index.fulltext.listAvailableAnalyzers()	List the available analyzers that the full-text indexes can be configured with.	`db.index.fulltext.listAvailableAnalyzers() :: (analyzer :: STRING?, description :: STRING?, stopwords :: LIST? OF STRING?)`	`READ`	N/A
db.index.fulltext.queryNodes()	Query the given full-text index. Returns the matching nodes and their Lucene query score, ordered by score.	`db.index.fulltext.queryNodes(indexName :: STRING?, queryString :: STRING?) :: (node :: NODE?, score :: FLOAT?)`	`READ`	N/A

Name	Description	Signature	Mode	Roles
db.index.fulltext.queryRelationships()	Query the given full-text index. Returns the matching relationships and their Lucene query score, ordered by score.	`db.index.fulltext.queryRelationships(indexName :: STRING?, queryString :: STRING?) :: (relationship :: RELATIONSHIP?, score :: FLOAT?)`	`READ`	N/A
db.indexDetails()	Detailed description of specific index.	`db.indexDetails(indexName :: STRING?) :: (id :: INTEGER?, name :: STRING?, state :: STRING?, populationPercent :: FLOAT?, uniqueness :: STRING?, type :: STRING?, entityType :: STRING?, labelsOrTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, provider :: STRING?, indexConfig :: MAP?, failureMessage :: STRING?)`	`READ`	N/A
db.indexes()	List all indexes in the database.	`db.indexes() :: (id :: INTEGER?, name :: STRING?, state :: STRING?, populationPercent :: FLOAT?, uniqueness :: STRING?, type :: STRING?, entityType :: STRING?, labelsOrTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, provider :: STRING?)`	`READ`	N/A
db.labels()	List all available labels in the database.	`db.labels() :: (label :: STRING?)`	`READ`	N/A
db.prepareForReplanning()	Triggers an index resample and waits for it to complete, and after that clears query caches. After this procedure has finished queries will be planned using the latest database statistics.	`db.prepareForReplanning(timeOutSeconds = 300 :: INTEGER?) :: VOID`	`READ`	N/A
db.propertyKeys()	List all property keys in the database.	`db.propertyKeys() :: (propertyKey :: STRING?)`	`READ`	N/A

Name	Description	Signature	Mode	Roles
db.relationshipTypes()	List all available relationship types in the database.	`db.relationshipTypes() :: (relationshipType :: STRING?)`	READ	N/A
db.resampleIndex()	Schedule resampling of an index (for example: CALL db.resampleIndex("MyIndex")).	`db.resampleIndex(indexName :: STRING?) :: VOID`	READ	N/A
db.resampleOutdatedIndexes()	Schedule resampling of all outdated indexes.	`db.resampleOutdatedIndexes() :: VOID`	READ	N/A
db.schema.nodeTypeProperties()	Show the derived property schema of the nodes in tabular form.	`db.schema.nodeTypeProperties() :: (nodeType :: STRING?, nodeLabels :: LIST? OF STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)`	READ	N/A
db.schema.relTypeProperties()	Show the derived property schema of the relationships in tabular form.	`db.schema.relTypeProperties() :: (relType :: STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)`	READ	N/A
db.schema.visualization()	Visualize the schema of the data.	`db.schema.visualization() :: (nodes :: LIST? OF NODE?, relationships :: LIST? OF RELATIONSHIP?)`	READ	N/A
db.schemaStatements()	List all statements for creating and dropping existing indexes and constraints.	`db.schemaStatements() :: (name :: STRING?, type :: STRING?, createStatement :: STRING?, dropStatement :: STRING?)`	READ	N/A
db.stats.clear()	Clear collected data of a given data section. Valid sections are 'QUERIES'	`db.stats.clear(section :: STRING?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)`	READ	N/A

Name	Description	Signature	Mode	Roles
db.stats.collect()	Start data collection of a given data section. Valid sections are 'QUERIES'	`db.stats.collect(section :: STRING?, config = {} :: MAP?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)`	`READ`	N/A
db.stats.retrieve()	Retrieve statistical data about the current database. Valid sections are 'GRAPH COUNTS', 'TOKENS', 'QUERIES', 'META'	`db.stats.retrieve(section :: STRING?, config = {} :: MAP?) :: (section :: STRING?, data :: MAP?)`	`READ`	N/A
db.stats.retrieveAllAnonymized()	Retrieve all available statistical data about the current database, in an anonymized form.	`db.stats.retrieveAllAnonymized(graphToken :: STRING?, config = {} :: MAP?) :: (section :: STRING?, data :: MAP?)`	`READ`	N/A
db.stats.status()	Retrieve the status of all available collector daemons, for this database.	`db.stats.status() :: (section :: STRING?, status :: STRING?, data :: MAP?)`	`READ`	N/A
db.stats.stop()	Stop data collection of a given data section. Valid sections are 'QUERIES'	`db.stats.stop(section :: STRING?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)`	`READ`	N/A
dbms.cluster.routing.getRoutingTable()	Returns endpoints of this instance.	`dbms.cluster.routing.getRoutingTable(context :: MAP?, database = null :: STRING?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)`	`DBMS`	N/A
dbms.components()	List DBMS components and their versions.	`dbms.components() :: (name :: STRING?, versions :: LIST? OF STRING?, edition :: STRING?)`	`DBMS`	N/A
dbms.database.state()	The actual status of the database with the provided name on this neo4j instance.	`dbms.database.state(databaseName :: STRING?) :: (role :: STRING?, address :: STRING?, status :: STRING?, error :: STRING?)`	`DBMS`	N/A

Name	Description	Signature	Mode	Roles
dbms.functions()	List all functions in the DBMS.	`dbms.functions() :: (name :: STRING?, signature :: STRING?, description :: STRING?, aggregating :: BOOLEAN?, defaultBuiltInRoles :: LIST? OF STRING?)`	`DBMS`	N/A
dbms.killConnection()	Kill network connection with the given connection id.	`dbms.killConnection(id :: STRING?) :: (connectionId :: STRING?, username :: STRING?, message :: STRING?)`	`DBMS`	N/A
dbms.killConnections()	Kill all network connections with the given connection ids.	`dbms.killConnections(ids :: LIST? OF STRING?) :: (connectionId :: STRING?, username :: STRING?, message :: STRING?)`	`DBMS`	N/A
dbms.killQueries()	Kill all transactions executing a query with any of the given query ids.	`dbms.killQueries(ids :: LIST? OF STRING?) :: (queryId :: STRING?, username :: STRING?, message :: STRING?)`	`DBMS`	N/A
dbms.killQuery()	Kill all transactions executing the query with the given query id.	`dbms.killQuery(id :: STRING?) :: (queryId :: STRING?, username :: STRING?, message :: STRING?)`	`DBMS`	N/A
dbms.killTransaction()	Kill transaction with provided id.	`dbms.killTransaction(id :: STRING?) :: (transactionId :: STRING?, username :: STRING?, message :: STRING?)`	`DBMS`	N/A
dbms.killTransactions()	Kill transactions with provided ids.	`dbms.killTransactions(ids :: LIST? OF STRING?) :: (transactionId :: STRING?, username :: STRING?, message :: STRING?)`	`DBMS`	N/A
dbms.listActiveLocks()	List the active lock requests granted for the transaction executing the query with the given query id.	`dbms.listActiveLocks(queryId :: STRING?) :: (mode :: STRING?, resourceType :: STRING?, resourceId :: INTEGER?)`	`DBMS`	N/A
dbms.listConfig()	List the currently active config of Neo4j.	`dbms.listConfig(searchString = :: STRING?) :: (name :: STRING?, description :: STRING?, value :: STRING?, dynamic :: BOOLEAN?)`	`DBMS`	N/A

Name	Description	Signature	Mode	Roles
dbms.listConnections()	List all accepted network connections at this instance that are visible to the user.	`dbms.listConnections() :: (connectionId :: STRING?, connectTime :: STRING?, connector :: STRING?, username :: STRING?, userAgent :: STRING?, serverAddress :: STRING?, clientAddress :: STRING?)`	`DBMS`	N/A
dbms.listQueries()	List all queries currently executing at this instance that are visible to the user.	`dbms.listQueries() :: (queryId :: STRING?, username :: STRING?, metaData :: MAP?, query :: STRING?, parameters :: MAP?, planner :: STRING?, runtime :: STRING?, indexes :: LIST? OF MAP?, startTime :: STRING?, protocol :: STRING?, clientAddress :: STRING?, requestUri :: STRING?, status :: STRING?, resourceInformation :: MAP?, activeLockCount :: INTEGER?, elapsedTimeMillis :: INTEGER?, cpuTimeMillis :: INTEGER?, waitTimeMillis :: INTEGER?, idleTimeMillis :: INTEGER?, allocatedBytes :: INTEGER?, pageHits :: INTEGER?, pageFaults :: INTEGER?, connectionId :: STRING?, database :: STRING?)`	`DBMS`	N/A

Name	Description	Signature	Mode	Roles
dbms.listTransactions()	List all transactions currently executing at this instance that are visible to the user.	`dbms.listTransactions() :: (transactionId :: STRING?, username :: STRING?, metaData :: MAP?, startTime :: STRING?, protocol :: STRING?, clientAddress :: STRING?, requestUri :: STRING?, currentQueryId :: STRING?, currentQuery :: STRING?, activeLockCount :: INTEGER?, status :: STRING?, resourceInformation :: MAP?, elapsedTimeMillis :: INTEGER?, cpuTimeMillis :: INTEGER?, waitTimeMillis :: INTEGER?, idleTimeMillis :: INTEGER?, allocatedBytes :: INTEGER?, allocatedDirectBytes :: INTEGER?, pageHits :: INTEGER?, pageFaults :: INTEGER?, connectionId :: STRING?, initializationStackTrace :: STRING?, database :: STRING?)`	`DBMS`	N/A
dbms.procedures()	List all procedures in the DBMS.	`dbms.procedures() :: (name :: STRING?, signature :: STRING?, description :: STRING?, mode :: STRING?, defaultBuiltInRoles :: LIST? OF STRING?, worksOnSystem :: BOOLEAN?)`	`DBMS`	N/A
dbms.queryJmx()	Query JMX management data by domain and name. For instance, "org.neo4j:*"	`dbms.queryJmx(query :: STRING?) :: (name :: STRING?, description :: STRING?, attributes :: MAP?)`	`DBMS`	N/A
dbms.routing.getRoutingTable()	Returns endpoints of this instance.	`dbms.routing.getRoutingTable(context :: MAP?, database = null :: STRING?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)`	`DBMS`	N/A
dbms.scheduler.groups()	List the job groups that are active in the database internal job scheduler.	`dbms.scheduler.groups() :: (group :: STRING?, threads :: INTEGER?)`	`DBMS`	N/A

Name	Description	Signature	Mode	Roles
dbms.scheduler.profile()	Begin profiling all threads within the given job group, for the specified duration. Note that profiling incurs overhead to a system, and will slow it down.	`dbms.scheduler.profile(method :: STRING?, group :: STRING?, duration :: STRING?) :: (profile :: STRING?)`	DBMS	N/A
dbms.security.activateUser()	Activate a suspended user.	`dbms.security.activateUser(username :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID`	DBMS	N/A
dbms.security.addRoleToUser()	Assign a role to the user.	`dbms.security.addRoleToUser(roleName :: STRING?, username :: STRING?) :: VOID`	DBMS	N/A
dbms.security.changePassword()	Change the current user's password.	`dbms.security.changePassword(password :: STRING?, requirePasswordChange = false :: BOOLEAN?) :: VOID`	DBMS	N/A
dbms.security.changeUserPassword()	Change the given user's password.	`dbms.security.changeUserPassword(username :: STRING?, newPassword :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID`	DBMS	N/A
dbms.security.clearAuthCache()	Clears authentication and authorization cache.	`dbms.security.clearAuthCache() :: VOID`	DBMS	N/A
dbms.security.createRole()	Create a new role.	`dbms.security.createRole(roleName :: STRING?) :: VOID`	DBMS	N/A
dbms.security.createUser()	Create a new user.	`dbms.security.createUser(username :: STRING?, password :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID`	DBMS	N/A
dbms.security.deleteRole()	Delete the specified role. Any role assignments will be removed.	`dbms.security.deleteRole(roleName :: STRING?) :: VOID`	DBMS	N/A
dbms.security.deleteUser()	Delete the specified user.	`dbms.security.deleteUser(username :: STRING?) :: VOID`	DBMS	N/A

Name	Description	Signature	Mode	Roles
dbms.security.listRoles()	List all available roles.	`dbms.security.listRoles() :: (role :: STRING?, users :: LIST? OF STRING?)`	DBMS	N/A
dbms.security.listRolesForUser()	List all roles assigned to the specified user.	`dbms.security.listRolesForUser(username :: STRING?) :: (value :: STRING?)`	DBMS	N/A
dbms.security.listUsers()	List all native users.	`dbms.security.listUsers() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?)`	DBMS	N/A
dbms.security.listUsersForRole()	List all users currently assigned the specified role.	`dbms.security.listUsersForRole(roleName :: STRING?) :: (value :: STRING?)`	DBMS	N/A
dbms.security.removeRoleFromUser()	Unassign a role from the user.	`dbms.security.removeRoleFromUser(roleName :: STRING?, username :: STRING?) :: VOID`	DBMS	N/A
dbms.security.suspendUser()	Suspend the specified user.	`dbms.security.suspendUser(username :: STRING?) :: VOID`	DBMS	N/A
dbms.setConfigValue()	Updates a given setting value. Passing an empty value will result in removing the configured value and falling back to the default value. Changes will not persist and will be lost if the server is restarted.	`dbms.setConfigValue(setting :: STRING?, value :: STRING?) :: VOID`	DBMS	N/A
dbms.showCurrentUser()	Show the current user.	`dbms.showCurrentUser() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?)`	DBMS	N/A
tx.getMetaData()	Provides attached transaction metadata.	`tx.getMetaData() :: (metadata :: MAP?)`	DBMS	N/A

Name	Description	Signature	Mode	Roles
tx.setMetaData()	Attaches a map of data to the transaction. The data will be printed when listing queries, and inserted into the query log.	`tx.setMetaData(data :: MAP?) :: VOID`	`DBMS`	N/A

A.2.3. Community Edition procedures

Table 321. Community Edition procedures

Name	Description	Signature	Mode
db.awaitIndex()	Wait for an index to come online (for example: CALL db.awaitIndex("MyIndex", 300)).	`db.awaitIndex(indexName :: STRING?, timeOutSeconds = 300 :: INTEGER?) :: VOID`	`READ`
db.awaitIndexes()	Wait for all indexes to come online (for example: CALL db.awaitIndexes(300)).	`db.awaitIndexes(timeOutSeconds = 300 :: INTEGER?) :: VOID`	`READ`
db.clearQueryCaches()	Clears all query caches.	`db.clearQueryCaches() :: (value :: STRING?)`	`DBMS`
db.constraints()	List all constraints in the database.	`db.constraints() :: (name :: STRING?, description :: STRING?)`	`READ`
db.createIndex()	Create a named schema index with specified index provider and configuration (optional). Yield: name, labels, properties, providerName, status	`db.createIndex(indexName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)`	`SCHEMA`
db.createLabel()	Create a label	`db.createLabel(newLabel :: STRING?) :: VOID`	`WRITE`
db.createProperty()	Create a Property	`db.createProperty(newProperty :: STRING?) :: VOID`	`WRITE`
db.createRelationshipType()	Create a RelationshipType	`db.createRelationshipType(newRelationshipType :: STRING?) :: VOID`	`WRITE`

Name	Description	Signature	Mode
db.createUniquePropertyConstraint()	Create a named unique property constraint. Backing index will use specified index provider and configuration (optional). Yield: name, labels, properties, providerName, status	`db.createUniquePropertyConstraint(constraintName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, config = {} :: MAP?) :: (name :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, providerName :: STRING?, status :: STRING?)`	`SCHEMA`
db.index.fulltext.awaitEventuallyConsistentIndexRefresh()	Wait for the updates from recently committed transactions to be applied to any eventually-consistent full-text indexes.	`db.index.fulltext.awaitEventuallyConsistentIndexRefresh() :: VOID`	`READ`
db.index.fulltext.createNodeIndex()	Create a node full-text index for the given labels and properties. The optional 'config' map parameter can be used to supply settings to the index. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the `db.index.fulltext.listAvailableAnalyzers` procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread.	`db.index.fulltext.createNodeIndex(indexName :: STRING?, labels :: LIST? OF STRING?, properties :: LIST? OF STRING?, config = {} :: MAP?) :: VOID`	`SCHEMA`

Name	Description	Signature	Mode
db.index.fulltext.createRelationshipIndex()	Create a relationship full-text index for the given relationship types and properties. The optional 'config' map parameter can be used to supply settings to the index. Supported settings are 'analyzer', for specifying what analyzer to use when indexing and querying. Use the `db.index.fulltext.listAvailableAnalyzers` procedure to see what options are available. And 'eventually_consistent' which can be set to 'true' to make this index eventually consistent, such that updates from committing transactions are applied in a background thread.	`db.index.fulltext.createRelationshipIndex(indexName :: STRING?, relationshipTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, config = {} :: MAP?) :: VOID`	`SCHEMA`
db.index.fulltext.drop()	Drop the specified index.	`db.index.fulltext.drop(indexName :: STRING?) :: VOID`	`SCHEMA`
db.index.fulltext.listAvailableAnalyzers()	List the available analyzers that the full-text indexes can be configured with.	`db.index.fulltext.listAvailableAnalyzers() :: (analyzer :: STRING?, description :: STRING?, stopwords :: LIST? OF STRING?)`	`READ`
db.index.fulltext.queryNodes()	Query the given full-text index. Returns the matching nodes and their Lucene query score, ordered by score.	`db.index.fulltext.queryNodes(indexName :: STRING?, queryString :: STRING?) :: (node :: NODE?, score :: FLOAT?)`	`READ`
db.index.fulltext.queryRelationships()	Query the given full-text index. Returns the matching relationships and their Lucene query score, ordered by score.	`db.index.fulltext.queryRelationships(indexName :: STRING?, queryString :: STRING?) :: (relationship :: RELATIONSHIP?, score :: FLOAT?)`	`READ`

Name	Description	Signature	Mode
db.indexDetails()	Detailed description of specific index.	`db.indexDetails(indexName :: STRING?) :: (id :: INTEGER?, name :: STRING?, state :: STRING?, populationPercent :: FLOAT?, uniqueness :: STRING?, type :: STRING?, entityType :: STRING?, labelsOrTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, provider :: STRING?, indexConfig :: MAP?, failureMessage :: STRING?)`	`READ`
db.indexes()	List all indexes in the database.	`db.indexes() :: (id :: INTEGER?, name :: STRING?, state :: STRING?, populationPercent :: FLOAT?, uniqueness :: STRING?, type :: STRING?, entityType :: STRING?, labelsOrTypes :: LIST? OF STRING?, properties :: LIST? OF STRING?, provider :: STRING?)`	`READ`
db.labels()	List all available labels in the database.	`db.labels() :: (label :: STRING?)`	`READ`
db.prepareForReplanning()	Triggers an index resample and waits for it to complete, and after that clears query caches. After this procedure has finished queries will be planned using the latest database statistics.	`db.prepareForReplanning(timeOutSeconds = 300 :: INTEGER?) :: VOID`	`READ`
db.propertyKeys()	List all property keys in the database.	`db.propertyKeys() :: (propertyKey :: STRING?)`	`READ`
db.relationshipTypes()	List all available relationship types in the database.	`db.relationshipTypes() :: (relationshipType :: STRING?)`	`READ`
db.resampleIndex()	Schedule resampling of an index (for example: CALL db.resampleIndex("MyIndex")).	`db.resampleIndex(indexName :: STRING?) :: VOID`	`READ`
db.resampleOutdatedIndexes()	Schedule resampling of all outdated indexes.	`db.resampleOutdatedIndexes() :: VOID`	`READ`

Name	Description	Signature	Mode
db.schema.nodeTypeProperties()	Show the derived property schema of the nodes in tabular form.	`db.schema.nodeTypeProperties() :: (nodeType :: STRING?, nodeLabels :: LIST? OF STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)`	`READ`
db.schema.relTypeProperties()	Show the derived property schema of the relationships in tabular form.	`db.schema.relTypeProperties() :: (relType :: STRING?, propertyName :: STRING?, propertyTypes :: LIST? OF STRING?, mandatory :: BOOLEAN?)`	`READ`
db.schema.visualization()	Visualize the schema of the data.	`db.schema.visualization() :: (nodes :: LIST? OF NODE?, relationships :: LIST? OF RELATIONSHIP?)`	`READ`
db.schemaStatements()	List all statements for creating and dropping existing indexes and constraints.	`db.schemaStatements() :: (name :: STRING?, type :: STRING?, createStatement :: STRING?, dropStatement :: STRING?)`	`READ`
db.stats.clear()	Clear collected data of a given data section. Valid sections are 'QUERIES'	`db.stats.clear(section :: STRING?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)`	`READ`
db.stats.collect()	Start data collection of a given data section. Valid sections are 'QUERIES'	`db.stats.collect(section :: STRING?, config = {} :: MAP?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)`	`READ`
db.stats.retrieve()	Retrieve statistical data about the current database. Valid sections are 'GRAPH COUNTS', 'TOKENS', 'QUERIES', 'META'	`db.stats.retrieve(section :: STRING?, config = {} :: MAP?) :: (section :: STRING?, data :: MAP?)`	`READ`
db.stats.retrieveAllAnonymized()	Retrieve all available statistical data about the current database, in an anonymized form.	`db.stats.retrieveAllAnonymized(graphToken :: STRING?, config = {} :: MAP?) :: (section :: STRING?, data :: MAP?)`	`READ`
db.stats.status()	Retrieve the status of all available collector daemons, for this database.	`db.stats.status() :: (section :: STRING?, status :: STRING?, data :: MAP?)`	`READ`
db.stats.stop()	Stop data collection of a given data section. Valid sections are 'QUERIES'	`db.stats.stop(section :: STRING?) :: (section :: STRING?, success :: BOOLEAN?, message :: STRING?)`	`READ`

Name	Description	Signature	Mode
dbms.cluster.routing.getRoutingTable()	Returns endpoints of this instance.	`dbms.cluster.routing.getRoutingTable(context :: MAP?, database = null :: STRING?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)`	`DBMS`
dbms.components()	List DBMS components and their versions.	`dbms.components() :: (name :: STRING?, versions :: LIST? OF STRING?, edition :: STRING?)`	`DBMS`
dbms.database.state()	The actual status of the database with the provided name on this neo4j instance.	`dbms.database.state(databaseName :: STRING?) :: (role :: STRING?, address :: STRING?, status :: STRING?, error :: STRING?)`	`DBMS`
dbms.functions()	List all functions in the DBMS.	`dbms.functions() :: (name :: STRING?, signature :: STRING?, description :: STRING?, aggregating :: BOOLEAN?, defaultBuiltInRoles :: LIST? OF STRING?)`	`DBMS`
dbms.listConfig()	List the currently active config of Neo4j.	`dbms.listConfig(searchString = :: STRING?) :: (name :: STRING?, description :: STRING?, value :: STRING?, dynamic :: BOOLEAN?)`	`DBMS`
dbms.procedures()	List all procedures in the DBMS.	`dbms.procedures() :: (name :: STRING?, signature :: STRING?, description :: STRING?, mode :: STRING?, defaultBuiltInRoles :: LIST? OF STRING?, worksOnSystem :: BOOLEAN?)`	`DBMS`
dbms.queryJmx()	Query JMX management data by domain and name. For instance, "org.neo4j:*"	`dbms.queryJmx(query :: STRING?) :: (name :: STRING?, description :: STRING?, attributes :: MAP?)`	`DBMS`
dbms.routing.getRoutingTable()	Returns endpoints of this instance.	`dbms.routing.getRoutingTable(context :: MAP?, database = null :: STRING?) :: (ttl :: INTEGER?, servers :: LIST? OF MAP?)`	`DBMS`
dbms.security.changePassword()	Change the current user's password.	`dbms.security.changePassword(password :: STRING?) :: VOID`	`DBMS`
dbms.security.createUser()	Create a new user.	`dbms.security.createUser(username :: STRING?, password :: STRING?, requirePasswordChange = true :: BOOLEAN?) :: VOID`	`DBMS`
dbms.security.deleteUser()	Delete the specified user.	`dbms.security.deleteUser(username :: STRING?) :: VOID`	`DBMS`
dbms.security.listUsers()	List all native users.	`dbms.security.listUsers() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?)`	`DBMS`

Name	Description	Signature	Mode
dbms.showCurrentUser()	Show the current user.	`dbms.showCurrentUser() :: (username :: STRING?, roles :: LIST? OF STRING?, flags :: LIST? OF STRING?)`	`DBMS`
tx.getMetaData()	Provides attached transaction metadata.	`tx.getMetaData() :: (metadata :: MAP?)`	`DBMS`
tx.setMetaData()	Attaches a map of data to the transaction. The data will be printed when listing queries, and inserted into the query log.	`tx.setMetaData(data :: MAP?) :: VOID`	`DBMS`

Appendix B: Tutorials

This appendix contains examples and tutorials that further describe usages of Neo4j.

The following step-by-step tutorials cover common operational tasks or otherwise exemplify working with Neo4j.

- Set up a local Causal Cluster
- Use the Import tool to import data into Neo4j

B.1. Set up a local Causal Cluster

This tutorial walks through the basics of setting up a Neo4j Causal Cluster. The result is a local cluster of six instances: three Cores and three Read Replicas.

This tutorial describes the following:

- Introduction
- Download Neo4j and configure the Core instances
  - Configure the first Core instance
  - Configure the second Core instance
  - Configure the third Core instance
- Start the Core Servers
- Check the status of the cluster
- Configure the Read Replicas
  - Configure the first Read Replica
  - Configure the second Read Replica
  - Configure the third Read Replica
- Start the Read Replicas
- Test the cluster with Read Replicas

Introduction

In this tutorial we will learn how to deploy a Causal Cluster locally, on a single machine. This is a useful learning exercise, and will get you started quickly with developing an application against a Neo4j Causal Cluster. Please note that in practice, a cluster on a single machine has no fault tolerance and is therefore not suitable for production use.

We will begin by configuring and starting a cluster of three Core instances. This is the minimal deployment for a fault-tolerant Causal Cluster (for a discussion on the number of servers required for a Causal Cluster, see Core Servers). The Core instances are responsible for keeping the data safe.

After the Core of the cluster is operational we will add three Read Replicas. The Read Replicas are responsible for scaling the capacity of the cluster.

The Core of the Causal Cluster is intended to remain stable over time. The roles within the Core will change as needed, but the Core itself is long-lived and stable. Read Replicas live at the edge of the cluster and can be brought up and taken down without affecting the Core. They can be added as needed to increase the operational capacity of the cluster as a whole.

In this tutorial we will be running all instances in the cluster on a single machine. Many of the default configuration settings work well out of the box in a production deployment, with multiple machines. Some of these we have to change when deploying multiple instances on a single machine, so that instances do not try to use the same network ports. We call out the settings that are specific to this scenario as we go along.

Download Neo4j and configure the Core instances

1. Create a local working directory.
2. Download a copy of Neo4j Enterprise Edition from the Neo4j download site.
3. Unpack Neo4j in the working directory.
4. Make a copy of the *neo4j-enterprise-4.0.0* directory, and name it *core-01*. We will need to keep the original directory to use when setting up the Read Replicas later. The *core-01* directory will contain the first Core instance.

Configure the first Core instance

All configuration that we will do takes place in the Neo4j configuration file, *conf/neo4j.conf*. If you used a different package than in the download instructions above, see File locations to locate the configuration file.

The first settings we will change represent the minimum configuration for a Core instance:

1. Locate and uncomment the setting `dbms.mode=CORE`.
2. Locate and uncomment the setting `causal_clustering.minimum_core_cluster_size_at_formation=3`.
3. Locate and uncomment the setting `causal_clustering.minimum_core_cluster_size_at_runtime=3`.
4. Locate and uncomment the setting `causal_clustering.initial_discovery_members=localhost:5000,localhost:5001,localhost:5002`.

Since we are setting up the Causal Cluster to run on a single machine, we must do some additional configuration. Please note that these steps would not be necessary if the instances are running on different servers.

1. Locate and uncomment the setting `causal_clustering.discovery_listen_address=:5000`.
2. Locate and uncomment the setting `causal_clustering.transaction_listen_address=:6000`.
3. Locate and uncomment the setting `causal_clustering.raft_listen_address=:7000`.
4. Locate and uncomment the setting `dbms.connector.bolt.listen_address=:7687`.
5. Locate and uncomment the setting `dbms.connector.http.listen_address=:7474`.
6. Locate the `dbms.connector.https.listen_address` setting and change the value to `:6474`.
7. Locate the `dbms.backup.listen_address` setting and change the value to `:6362`.

Configure the second Core instance

We can now create the second Core instance.

As mentioned above, we also need to amend some of the additional values in the *neo4j.conf* file so that our cluster can run on a single machine:

1. Make a copy of the *core-01* directory and rename it *core-02*.

2. Open the *neo4j.conf* file in the new *core-02* directory.
   a. Locate the `causal_clustering.discovery_listen_address` setting and change the value to `:5001`.
   b. Locate the `causal_clustering.transaction_listen_address` setting and change the value to `:6001`.
   c. Locate the `causal_clustering.raft_listen_address` setting and change the value to `:7001`.
   d. Locate the `dbms.connector.bolt.listen_address` setting and change the value to `:7688`.
   e. Locate the `dbms.connector.http.listen_address` setting and change the value to `:7475`.
   f. Locate the `dbms.connector.https.listen_address` setting and change the value to `:6475`.
   g. Locate the `dbms.backup.listen_address` setting and change the value to `:6363`.

Configure the third Core instance

We can now create the third, and final Core instance.

Again, we also need to amend some of the additional values in the *neo4j.conf* file so that our cluster can run on a single machine:

1. Make a copy of the *core-02* directory and rename it *core-03*.
2. Open the *neo4j.conf* file in the new *core-03* directory.
   a. Locate the `causal_clustering.discovery_listen_address` setting and change the value to `:5002`.
   b. Locate the `causal_clustering.transaction_listen_address` setting and change the value to `:6002`.
   c. Locate the `causal_clustering.raft_listen_address` setting and change the value to `:7002`.
   d. Locate the `dbms.connector.bolt.listen_address` setting and change the value to `:7689`.
   e. Locate the `dbms.connector.http.listen_address` setting and change the value to `:7476`.
   f. Locate the `dbms.connector.https.listen_address` setting and change the value to `:6476`.
   g. Locate the `dbms.backup.listen_address` setting and change the value to `:6364`.

Start the Core servers

In any order, we can now start each of the Neo4j instances:

```
core-01$ ./bin/neo4j start
```

```
core-02$ ./bin/neo4j start
```

```
core-03$ ./bin/neo4j start
```

Startup Time

If you want to follow along with the startup of a server you can follow the messages in *logs/neo4j.log*:

- On a Unix system issue the command `tail -f logs/neo4j.log`.
- On Windows Server run `Get-Content .\logs\neo4j.log -Tail 10 -Wait`.

While an instance is joining the cluster, the server may appear unavailable. In the case where an instance is joining a cluster with lots of data, it may take a number of minutes for the new instance to download the data from the cluster and become available.

Check the status of the cluster

Now the minimal cluster of three Core Servers is operational and is ready to serve requests.

1. Connect to any of the three Core instances to check the cluster status. For example, for *core-01* point your web browser to http://localhost:7474.
2. Authenticate with the default `neo4j/neo4j` credentials, and set a new password when prompted. These credentials are not shared between cluster members. A new password must be set on each instance when connecting for the first time.
3. Check the status of the cluster by running the following in Neo4j Browser:

```
:sysinfo
```

Example 110. Example of a Cluster overview, achieved by running `:sysinfo`

The **Causal Cluster Members** table shows the status of instances in the cluster. The table below is an example of a test cluster:

Roles	Addresses	Actions
LEADER	bolt://localhost:7687, http://localhost:7474, https://localhost:6474	`Open`
FOLLOWER	bolt://localhost:7688, http://localhost:7475, https://localhost:6475	`Open`
FOLLOWER	bolt://localhost:7689, http://localhost:7476, https://localhost:6476	`Open`

The three Core instances in this cluster are now operational.

Now you can run queries to create nodes and relationships, and see that the data gets replicated in the cluster.

4. Click the `Open` action on the instance that has the *LEADER* role. This will open a new Neo4j Browser session against the Leader of the cluster.
5. Authenticate and set a new password, as before.
6. Run the following query to create nodes and relationships:

```
UNWIND range(0, 100) AS value
MERGE (person1:Person {id: value})
MERGE (person2:Person {id: toInteger(100.0 * rand())})
MERGE (person1)-[:FRIENDS]->(person2)
```

7. When the query has executed, choose an instance with the *FOLLOWER* role from the sysinfo view. Click the `Open` action to connect.
8. Run the following query to see that the data has been replicated:

```
MATCH path = (person:Person)-[:FRIENDS]-(friend)
RETURN path
LIMIT 10
```

Configure the Read Replicas

Read Replicas instances do not participate in quorum decisions, so their configuration is simpler than the configuration of Core Servers as there are fewer settings to amend.

All that a Read Replica needs to know is the addresses of Core Servers which they can bind to in order to discover the cluster. See Discovery protocol for the details of how this works. Once it has completed the initial discovery, the Read Replica becomes aware of the currently available Core Servers and can choose an appropriate one from which to catch up. See Catchup protocol for the details of how this works.

Configure the first Read Replica

1. In your working directory, make a copy of the *neo4j-enterprise-4.0.0* directory and name it *replica-01*.
2. Open the *neo4j.conf* file in the new *replica-01* directory. The first settings we will change represent the minimum configuration for a Read Replica:
   a. Locate and uncomment the `dbms.mode` setting and change the value to `READ_REPLICA`.
   b. Locate and uncomment the setting `causal_clustering.initial_discovery_members=localhost:5000,localhost:5001,localhost:5002` .
3. Since we are setting up the Causal Cluster to run on a single machine, we must do some additional configuration. Please note that the following steps would not be necessary if the instances are running on different servers:
   a. Locate and uncomment the `causal_clustering.transaction_listen_address` setting and change the value to `:6003`.
   b. Locate and uncomment the `dbms.connector.bolt.listen_address` setting and change the value to `:7690`.
   c. Locate and uncomment the `dbms.connector.http.listen_address` setting and change the value to `:7477`.
   d. Locate and uncomment the `dbms.connector.https.listen_address` setting and change the value to `:6477`.
   e. Locate and uncomment the `dbms.backup.listen_address` setting and change the value to `:6365`.

Configure the second Read Replica

We can now create the second Read Replica.

As mentioned above, we also need to amend some of the additional values in the *neo4j.conf* file so that our cluster can run on a single machine:

1. Make a copy of the *replica-01* directory and rename it *replica-02*.
2. Open the *neo4j.conf* file in the new *replica-02* directory.
   a. Locate the `causal_clustering.transaction_listen_address` setting and change the value to `:6004`.
   b. Locate the `dbms.connector.bolt.listen_address` setting and change the value to `:7691`.
   c. Locate the `dbms.connector.http.listen_address` setting and change the value to `:7478`.
   d. Locate the `dbms.connector.https.listen_address` setting and change the value to `:6478`.
   e. Locate the `dbms.backup.listen_address` setting and change the value to `:6366`.

Configure the third Read Replica

We can now create the third, and final Read Replica.

Again, we also need to amend some of the additional values in the *neo4j.conf* file so that our cluster can run on a single machine:

1. Make a copy of the *replica-02* directory and rename it *replica-03*.
2. Open the *neo4j.conf* file in the new *replica-03* directory.
   a. Locate the `causal_clustering.transaction_listen_address` setting and change the value to `:6005`.
   b. Locate the `dbms.connector.bolt.listen_address` setting and change the value to `:7692`.
   c. Locate the `dbms.connector.http.listen_address` setting and change the value to `:7479`.
   d. Locate the `dbms.connector.https.listen_address` setting and change the value to `:6479`.
   e. Locate the `dbms.backup.listen_address` setting and change the value to `:6367`.

Start the Read Replicas

In any order, we can now start the Read Replica instances:

```
replica-01$ ./bin/neo4j start
```

```
replica-02$ ./bin/neo4j start
```

```
replica-03$ ./bin/neo4j start
```

Test the cluster with Read Replicas

To test the status of the cluster now that the Read Replicas are running, we will repeat the steps from earlier, but via a Read Replica:

1. Connect to any of the three Read Replica instances. For example, for *replica-01* point your web browser to http://localhost:7477.
2. Authenticate with the default `neo4j/neo4j` credentials. Once again, you will need to set a new password when prompted.

3. Check the status of the cluster by running the following in Neo4j Browser:

```
:sysinfo
```

Example 111. Example of a cluster with both Core instances and Read Replicas, achieved by running `:sysinfo`

The following table shows the status of a test cluster which now includes Read Replicas:

Roles	Addresses	Actions
LEADER	bolt://localhost:7687, http://localhost:7474, https://localhost:6474	`Open`
FOLLOWER	bolt://localhost:7688, http://localhost:7475, https://localhost:6475	`Open`
FOLLOWER	bolt://localhost:7689, http://localhost:7476, https://localhost:6476	`Open`
READ_REPLICA	bolt://localhost:7690, http://localhost:7477, https://localhost:6477	`Open`
READ_REPLICA	bolt://localhost:7691, http://localhost:7478, https://localhost:6478	`Open`
READ_REPLICA	bolt://localhost:7692, http://localhost:7479, https://localhost:6479	`Open`

4. Click the `Open` action to connect to any of the Read Replicas.
5. Run the same query as before:

```
MATCH path = (person:Person)-[:FRIENDS]-(friend)
RETURN path
LIMIT 10
```

B.2. Use the Import tool

This tutorial provides detailed examples of using the Neo4j import tool.

This tutorial walks us through a series of examples to illustrate the capabilities of the Import tool.

When using CSV files for loading a database, each node must have a unique ID to be able to be referenced when creating relationships between nodes in the same import. In cases where the node ID is only unique within files, using *ID spaces* is a way to ensure uniqueness across all nodes files. Please see below and Using ID spaces for more information.

Relationships are created by connecting the node IDs. In the examples below, the node IDs are stored as properties on the nodes. Node IDs may be of interest later for cross-reference to other systems, traceability etc., but they are not mandatory. If you do not want the IDs to persist after a completed import, then do not specify a property name in the `:ID` field.

It is possible to import only nodes using the import tool by omitting a relationships file when calling `neo4j-admin import`. Any relationships between the imported nodes will have to be created later by another method, since the import tool works for initial graph population only.

For this tutorial we will use a data set containing movies, actors and roles. If not stated otherwise, the examples assume that the name of the database is `neo4j` (the default name) and that all CSV files are located in the *import* directory. Note that if you wish to run one example after another you have to remove the database in between.

Header files

For the basic examples we will use the first row of the data file for the header. This is fine when experimenting, but when working with anything but the smallest datasets we recommend keeping the header in a separate file.

B.2.1. Basic example

First we will look at the movies. Each movie has an ID, which is used for referring to it from other data sources. Moreover, each movie also has a title and a year. Along with these properties we also add the node labels `Movie` and `Sequel`:

movies.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

Next up are the actors. They have an ID - in this case a shorthand of their name - and a name. All the actors have the node label `Actor`:

actors.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

The node ID is needed to create relationships, but the node labels are optional.

Finally we have the roles that an actor plays in a movie, which will be represented by relationships in the database. There are three mandatory fields for relationships; `START_ID`, `END_ID` and `:TYPE`.

In order to create a relationship between nodes we use the IDs defined in `actors.csv` and `movies.csv` for the `START_ID` and `END_ID` fields. We also need to provide a relationship type (in this case `ACTED_IN`) for the `:TYPE` field:

roles.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies.csv --nodes=import/actors.csv
--relationships=import/roles.csv
```

In Community Edition, the database `neo4j` must be empty before the import, or else the import will fail. If `neo4j` is not empty, the entire directory may be deleted as it is created on import if non-existent. However, in Enterprise Edition, if there is an unfinished import in the `neo4j` folder, the tool will attempt to complete that import but fail if it cannot.

Now start up a database from the target directory:

```
neo4j_home$ ./bin/neo4j start
```

To see your imported data in the graph, try a simple command:

```
MATCH (n)
RETURN count(n) as count
```

B.2.2. Customizing configuration options

We can customize the configuration options that the import tool uses (see Options) if our data does not fit the default format. The following CSV files are delimited by `;`, use `|` as the array delimiter and use `'` for quotes:

movies2.csv

```
movieId:ID;title;year:int;:LABEL
tt0133093;'The Matrix';1999;Movie
tt0234215;'The Matrix Reloaded';2003;Movie|Sequel
tt0242653;'The Matrix Revolutions';2003;Movie|Sequel
```

actors2.csv

```
personId:ID;name;:LABEL
keanu;'Keanu Reeves';Actor
laurence;'Laurence Fishburne';Actor
carrieanne;'Carrie-Anne Moss';Actor
```

roles2.csv

```
:START_ID;role;:END_ID;:TYPE
keanu;'Neo';tt0133093;ACTED_IN
keanu;'Neo';tt0234215;ACTED_IN
keanu;'Neo';tt0242653;ACTED_IN
laurence;'Morpheus';tt0133093;ACTED_IN
laurence;'Morpheus';tt0234215;ACTED_IN
laurence;'Morpheus';tt0242653;ACTED_IN
carrieanne;'Trinity';tt0133093;ACTED_IN
carrieanne;'Trinity';tt0234215;ACTED_IN
carrieanne;'Trinity';tt0242653;ACTED_IN
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies2.csv --nodes=import/actors2.csv
--relationships=import/roles2.csv --delimiter=";" --array-delimiter="|" --quote="'"
```

B.2.3. Using separate header files

When dealing with very large CSV files it is more convenient to have the header in a separate file. This makes it easier to edit the header as you avoid having to open a huge data file just to change it. The header file must be specified before the rest of the files in each file group.

The import tool can also process single file compressed archives, for example: . `--nodes nodes.csv.gz` . `--relationships rels.zip`

We will use the same data as in the previous example but put the headers in separate files:

movies3-header.csv

```
movieId:ID,title,year:int,:LABEL
```

movies3.csv

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors3-header.csv

```
personId:ID,name,:LABEL
```

actors3.csv

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles3-header.csv

```
:START_ID,role,:END_ID,:TYPE
```

roles3.csv

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

The call to `neo4j-admin import` would look as follows, note how the file groups are enclosed in quotation marks in the command:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies3-header.csv,import/movies3.csv
--nodes=import/actors3-header.csv,import/actors3.csv --relationships=import/roles3
-header.csv,import/roles3.csv
```

The header line for a file group, whether it is the first line of a file in the group or a dedicated header file, must be the *first* line in the file group.

B.2.4. Multiple input files

In addition to using a separate header file you can also provide multiple nodes or relationships files. This may be useful for example for processing the output from a Hadoop pipeline. Files within such an input group can be specified with multiple match strings, delimited by `,`, where each match string can be either the exact file name or a regular expression matching one or more files. Multiple matching files will be sorted according to their characters and their natural number sort order for file names containing numbers:

movies4-header.csv

```
movieId:ID,title,year:int,:LABEL
```

movies4-part1.csv

```
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
```

movies4-part2.csv

```
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors4-header.csv

```
personId:ID,name,:LABEL
```

actors4-part1.csv

```
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
```

actors4-part2.csv

```
carrieanne,"Carrie-Anne Moss",Actor
```

roles4-header.csv

```
:START_ID,role,:END_ID,:TYPE
```

roles4-part1.csv

```
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
```

roles4-part2.csv

```
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies4-header.csv,import/movies4
-part1.csv,import/movies4-part2.csv --nodes=import/actors4-header.csv,import/actors4
-part1.csv,import/actors4-part2.csv --relationships=import/roles4-header.csv,import/roles4
-part1.csv,import/roles4-part2.csv
```

See also Using regular expressions for specifying multiple input files.

Using regular expressions for specifying multiple input files

File names can be specified using regular expressions in order to simplify using the command line when there are many data source files. Each file name that matches the regular expression will be included.

As mentioned in a previous section, for the import to work correctly, the header file must be first in the file group. When using regular expressions to specify the input files, the list of files will be sorted according to the names of the files that match the expression. The matching is aware of numbers inside the file names and will sort them accordingly, without the need for padding with zeros.

For example, let's assume that we have the following files:

- *movies4-header.csv*
- *movies4-data1.csv*
- *movies4-data2.csv*
- *movies4-data12.csv*

If we use the regular expression `movies4.*`, the sorting will place the header file last and the import will fail. A better alternative would be to name the header file explicitly and use a regular expression that only matches the names of the data files. For example: `--nodes "import/movies4-header.csv,movies-data.*"` will accomplish this.

Using the same data files as in the previous example, the call to `neo4j-admin import` can be simplified to:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies4-header.csv,import/movies4-part.*
--nodes=import/actors4-header.csv,import/actors4-part.* --relationships=import/roles4
-header.csv,import/roles4-part.*
```

The use of regular expressions should not be confused with file globbing.

The expression `.*` means: "zero or more occurrences of any character except line break". Therefore, the regular expression `movies4.*` will list all files starting with `movies4`. Conversely, with file globbing, `ls movies4.*` will list all files starting with `movies4.`.

Another important difference to pay attention to is the sorting order. The result of a regular expression matching will place the file `movies4-part2.csv` before the file `movies4-part12.csv`. If doing `ls movies4-part*` in a directory containing the above listed files, the file `movies4-part12.csv` will be listed before the file `movies4-part2.csv`.

B.2.5. Types and labels

Using the same label for every node

If you want to use the same node label(s) for every node in your nodes file you can do this by specifying the appropriate value as an option to `neo4j-admin import`. There is then no need to specify the `:LABEL` field in the node file if you pass it as a command line option. If you do then both the label provided in the file and the one provided on the command line will be added to the node.

In this example we put the label `Movie` on every node specified in `movies5a.csv`, and we put the labels `Movie` and `Sequel` on the nodes specified in `sequels5a.csv`:

movies5a.csv

```
movieId:ID,title,year:int
tt0133093,"The Matrix",1999
```

sequels5a.csv

```
movieId:ID,title,year:int
tt0234215,"The Matrix Reloaded",2003
tt0242653,"The Matrix Revolutions",2003
```

actors5a.csv

```
personId:ID,name
keanu,"Keanu Reeves"
laurence,"Laurence Fishburne"
carrieanne,"Carrie-Anne Moss"
```

roles5a.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=Movie=import/movies5a.csv
--nodes=Movie:Sequel=import/sequels5a.csv --nodes=Actor=import/actors5a.csv
--relationships=import/roles5a.csv
```

Using the same relationship type for every relationship

If you want to use the same relationship type for every relationship in your relationships file this can be done by specifying the appropriate value as an option to `neo4j-admin import`. If you provide a relationship type both on the command line and in the relationships file, the one in the file will be applied. In this example we put the relationship type `ACTED_IN` on every relationship specified in `roles5b.csv`:

movies5b.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors5b.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles5b.csv

```
:START_ID,role,:END_ID
keanu,"Neo",tt0133093
keanu,"Neo",tt0234215
keanu,"Neo",tt0242653
laurence,"Morpheus",tt0133093
laurence,"Morpheus",tt0234215
laurence,"Morpheus",tt0242653
carrieanne,"Trinity",tt0133093
carrieanne,"Trinity",tt0234215
carrieanne,"Trinity",tt0242653
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies5b.csv --nodes=import/actors5b.csv
--relationships=ACTED_IN=import/roles5b.csv
```

B.2.6. Property types

The type for properties specified in nodes and relationships files is defined in the header row. (see CSV file header format)

The following example creates a small graph containing one actor and one movie connected by an `ACTED_IN` relationship. There is a `roles` property on the relationship which contains an array of the characters played by the actor in a movie:

movies6.csv

```
movieId:ID,title,year:int,:LABEL
tt0099892,"Joe Versus the Volcano",1990,Movie
```

actors6.csv

```
personId:ID,name,:LABEL
meg,"Meg Ryan",Actor
```

roles6.csv

```
:START_ID,roles:string[],:END_ID,:TYPE
meg,"DeDe;Angelica Graynamore;Patricia Graynamore",tt0099892,ACTED_IN
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies6.csv --nodes=import/actors6.csv
--relationships=import/roles6.csv
```

B.2.7. ID handling

A node ID is used to find the correct nodes when creating relationships. Each node processed by `neo4j-admin import` must provide an ID if it is to be connected in any relationships.

Working with sequential or auto incrementing identifiers

The import tool makes the assumption that identifiers are unique across node files. This may not be the case for data sets which use sequential, auto incremented or otherwise colliding identifiers. Those data sets can define ID spaces where identifiers are unique within their respective ID space.

For example, if movies and people both use sequential identifiers then we would define `Movie` and `Actor` ID spaces:

movies7.csv

```
movieId:ID(Movie-ID),title,year:int,:LABEL
1,"The Matrix",1999,Movie
2,"The Matrix Reloaded",2003,Movie;Sequel
3,"The Matrix Revolutions",2003,Movie;Sequel
```

actors7.csv

```
personId:ID(Actor-ID),name,:LABEL
1,"Keanu Reeves",Actor
2,"Laurence Fishburne",Actor
3,"Carrie-Anne Moss",Actor
```

We also need to reference the appropriate ID space in our relationships file so it knows which nodes to connect together:

roles7.csv

```
:START_ID(Actor-ID),role,:END_ID(Movie-ID)
1,"Neo",1
1,"Neo",2
1,"Neo",3
2,"Morpheus",1
2,"Morpheus",2
2,"Morpheus",3
3,"Trinity",1
3,"Trinity",2
3,"Trinity",3
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies7.csv --nodes=import/actors7.csv
--relationships=ACTED_IN=import/roles7.csv
```

B.2.8. Bad input data

The import tool has no tolerance for bad entities (relationships or nodes) and will fail the import on the first bad entity. You can specify explicitly that you want it to ignore rows that contain bad entities.

There are two different types of bad input: bad relationships and bad nodes. We will have a closer look at these in the following examples.

Relationships referring to missing nodes

Relationships that refer to missing node IDs, either for `:START_ID` or `:END_ID` are considered bad relationships. Whether or not such relationships are skipped is controlled with `--skip-bad-relationships` flag which can have the values `true` or `false` or no value, which means `true`. The default is `false`, which means that any bad relationship is considered an error and will fail the import. For more information, see the `--skip-bad-relationships` option.

In the following example there is a missing `emil` node referenced in the roles file:

movies8a.csv

```
movieId:ID,title,year:int,:LABEL
tt0133093,"The Matrix",1999,Movie
tt0234215,"The Matrix Reloaded",2003,Movie;Sequel
tt0242653,"The Matrix Revolutions",2003,Movie;Sequel
```

actors8a.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
```

roles8a.csv

```
:START_ID,role,:END_ID,:TYPE
keanu,"Neo",tt0133093,ACTED_IN
keanu,"Neo",tt0234215,ACTED_IN
keanu,"Neo",tt0242653,ACTED_IN
laurence,"Morpheus",tt0133093,ACTED_IN
laurence,"Morpheus",tt0234215,ACTED_IN
laurence,"Morpheus",tt0242653,ACTED_IN
carrieanne,"Trinity",tt0133093,ACTED_IN
carrieanne,"Trinity",tt0234215,ACTED_IN
carrieanne,"Trinity",tt0242653,ACTED_IN
emil,"Emil",tt0133093,ACTED_IN
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies8a.csv --nodes=import/actors8a.csv
--relationships=import/roles8a.csv
```

Since there was a bad relationship in the input data, the import process will fail completely.

Let's see what happens if we append the `--skip-bad-relationships` flag:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/movies8a.csv --nodes=import/actors8a.csv
--relationships=import/roles8a.csv --skip-bad-relationships
```

The data files are successfully imported and the bad relationship is ignored. An entry is written to the *import.report* file.

ignore bad relationships

```
InputRelationship:
   source: roles8a.csv:11
   properties: [role, Emil]
   startNode: emil (global id space)
   endNode: tt0133093 (global id space)
   type: ACTED_IN
 referring to missing node emil
```

Multiple nodes with same ID within same ID space

Nodes that specify `:ID` which has already been specified within the ID space are considered bad nodes. Whether or not such nodes are skipped is controlled with `--skip-duplicate-nodes` flag which can have the values `true` or `false` or no value, which means `true`. The default is `false`, which means that any duplicate node is considered an error and will fail the import. For more information, see the

`--skip-duplicate-nodes` option.

In the following example there is a node ID that is specified twice within the same ID space:

actors8b.csv

```
personId:ID,name,:LABEL
keanu,"Keanu Reeves",Actor
laurence,"Laurence Fishburne",Actor
carrieanne,"Carrie-Anne Moss",Actor
laurence,"Laurence Harvey",Actor
```

The call to `neo4j-admin import` would look like this:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/actors8b.csv
```

Since there was a bad node in the input data, the import process will fail completely.

Let’s see what happens if we append the `--skip-duplicate-nodes` flag:

```
neo4j_home$ bin/neo4j-admin import --nodes=import/actors8b.csv --skip-duplicate-nodes
```

The data files are successfully imported and the bad node is ignored. An entry is written to the *import.report* file.

ignore bad nodes

```
ID 'laurence' is defined more than once in global ID space, at least at actors8b.csv:3 and actors8b.csv:5
```

Appendix C: Advanced Causal Clustering

This appendix describes advanced features of Neo4j Causal Clustering.

This section includes information about advanced deployments and configuration options for multi-data center operations.

- Causal Clustering lifecycle — A walk-through of the lifecycle of a cluster.
- Multi-data center — Overview of the multi-data center section.
    - Licensing for multi-data center operations — Information about licensing for multi-data center operations.
    - Multi-data center design — Patterns for multi-data center deployments.
    - Multi-data center operations — Configuration options for multi-data center deployments.
    - Multi-data center load balancing — Configuration options for making client applications aware of multi-data center topologies.
    - Data center disaster recovery — How to recover a cluster to full working capability after data center loss.
- Advanced settings reference — Advanced settings related to running a Neo4j Causal Cluster.

C.1. Causal Clustering lifecycle

This section describes the lifecycle of a Neo4j Causal Cluster.

This section includes:

- Introduction
- Discovery protocol
- Core membership
- Read Replica membership
- Transacting via the Raft protocol
- Catchup protocol
- Read Replica shutdown
- Core shutdown

C.1.1. Introduction

In this section we will develop some deeper knowledge of how the cluster operates. By developing our understanding of how the cluster works we will be better equipped to design, deploy, and troubleshoot our production systems.

Our in-depth tour will follow the lifecycle of a cluster. We will boot a Core cluster and pick up key architectural foundations as the cluster forms and transacts. We will then add in Read Replicas and show how they bootstrap join the cluster and then catchup and remain caught up with the Core Servers. We will then see how backup is used in live cluster environments before shutting down Read Replicas and Core Servers.

C.1.2. Discovery protocol

The discovery protocol is the first step in forming a Causal Cluster. It takes in some information about existing *Core* cluster servers, and uses this to initiate a network join protocol.

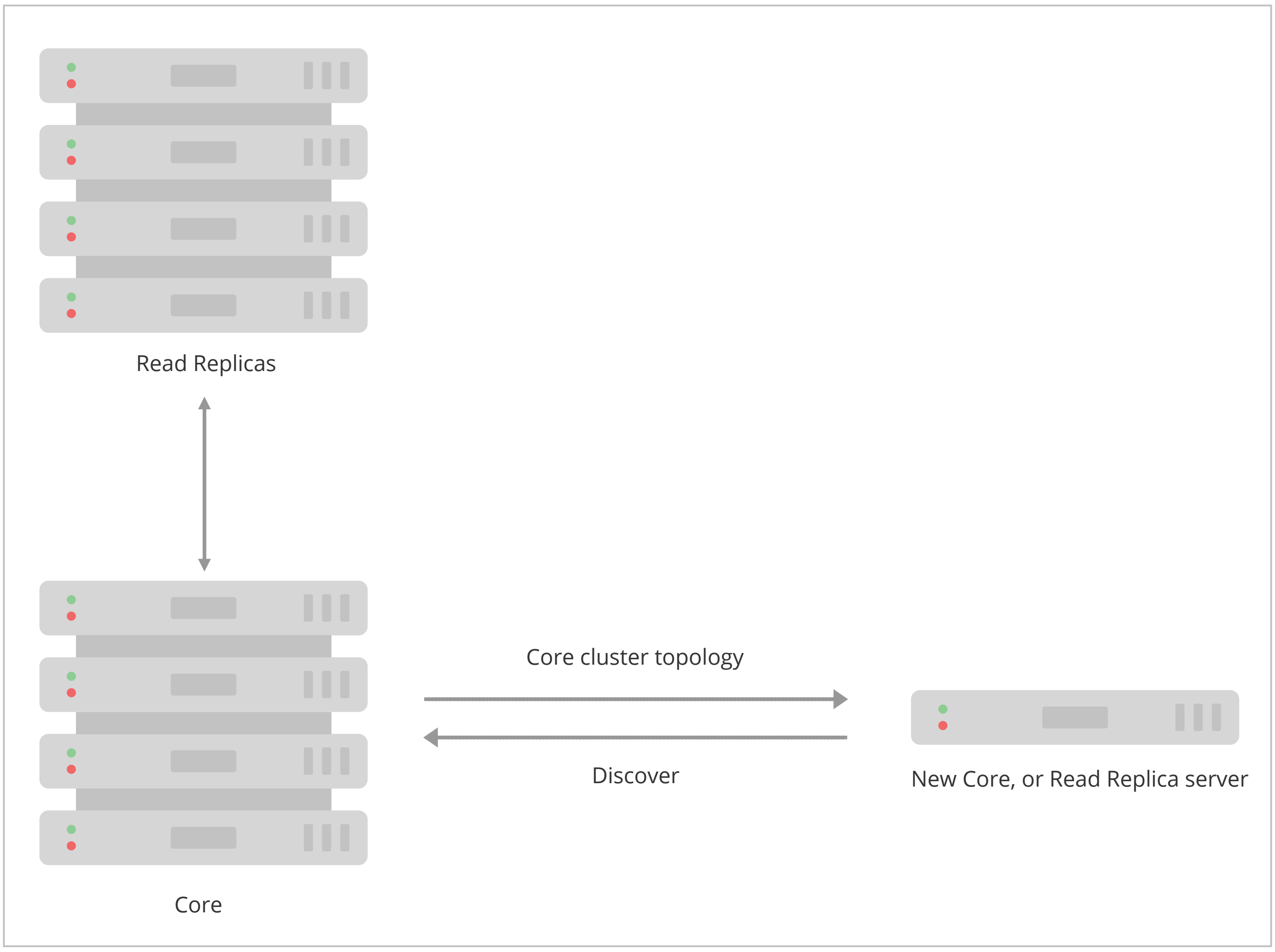

Figure 13. Causal Cluster discovery protocol: Core-to-Core or Read Replica-to-Core only.

Using this information, the server will either join an existing cluster or form one of its own.

The discovery protocol targets Core Servers only regardless of whether it is a Core Server or Read Replica performing discovery. It is because we expect Read Replicas to be both numerous and, relatively speaking, transient whereas Core Servers will likely be fewer in number and relatively stable over time.

The discovery protocol takes information from `causal_clustering.initial_discovery_members` in *neo4j.conf*, which lists which IP addresses and ports that form the cluster on startup. Detailed information about discovery and discovery configuration options is given in the Initial discovery of cluster members section. When consuming this information, the server will try to handshake with the other listed servers. On successful handshake with another server (or servers), the current server will discover the whole current topology.

The discovery protocol continues to run throughout the lifetime of the Causal Cluster and is used to maintain the current state of available servers and to help clients route queries to an appropriate server via the client-side drivers.

C.1.3. Core membership

If it is a Core Server that is performing discovery, once it has made a connection to the one of the existing Core Servers, it then joins the Raft protocol.

Raft is a distributed algorithm for maintaining a consistent log across multiple shared-nothing servers designed by Diego Ongaro for his 2014 Ph.D. thesis. See the Raft thesis for details.

Raft handles cluster membership by making it a normal part of keeping a distributed log in sync. Joining a cluster involves the insertion of a cluster membership entry into the Raft log which is then reliably replicated around the existing cluster. Once that entry is applied to enough members of the Raft consensus group (those machines running the specific instance of the algorithm), they update their view of the cluster to include the new server. Thus membership changes benefit from the same safety properties as other data transacted via Raft (see Transacting via the Raft protocol for more information).

The new Core Server must also catch up its own Raft log with respect to the other Core Servers as it initializes its internal Raft instance. This is the normal case when a cluster is first booted and has performed few operations. There will be a delay before the new Core Server becomes available if it also needs to catch up (as per Catchup protocol) graph data from other servers. This is the normal case for a long lived cluster where the servers holds a great deal of graph data.

When an instance establishes a connection to any other instance, it determines the current state of the cluster and ensures that it is eligible to join. To be eligible the Neo4j instance must host the same database store as other members of the cluster (although it is allowed to be in an older, outdated, state), or be a new deployment without a database store.

C.1.4. Read Replica membership

When a Read Replica performs discovery, once it has made a connection to any of the available Core clusters it proceeds to add itself into a shared whiteboard.

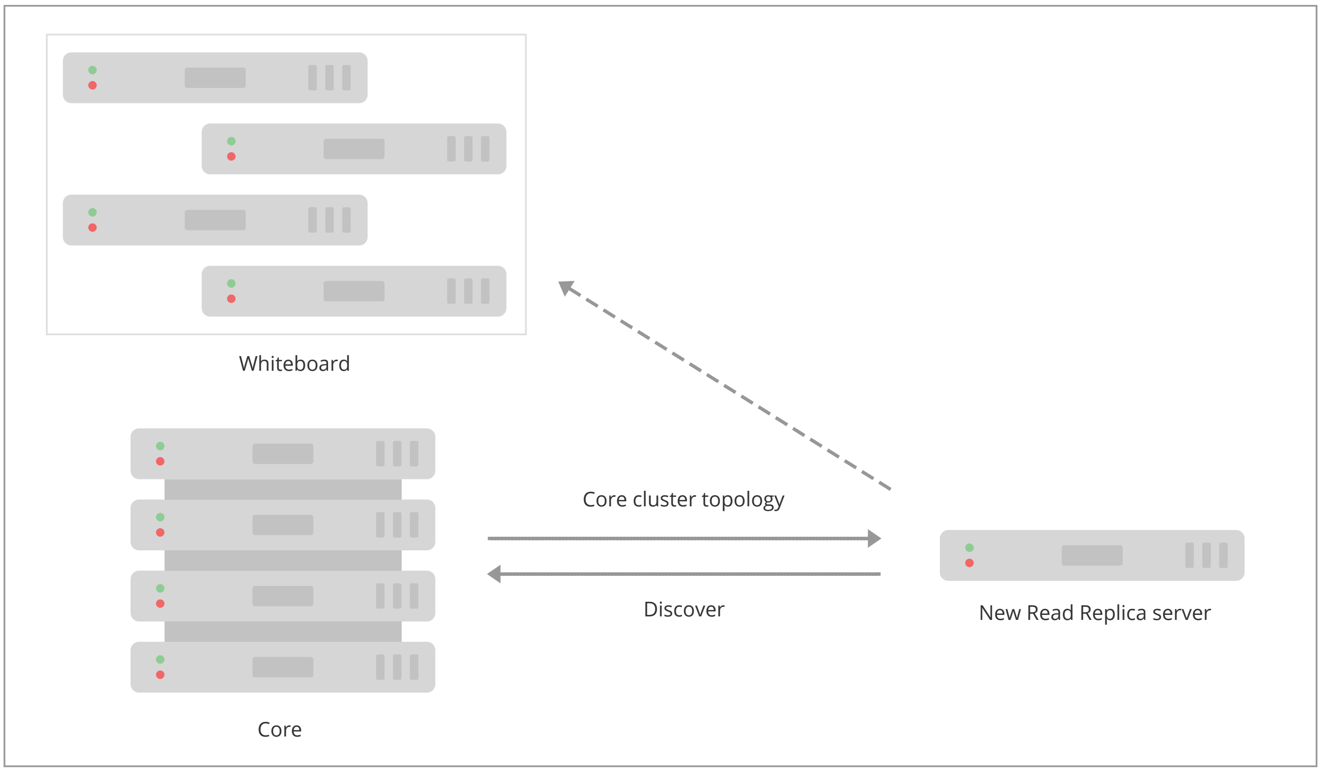

Figure 14. All Read Replicas registered with shared whiteboard.

This whiteboard provides a view of all live Read Replicas and is used both for routing requests from database drivers that support end-user applications and for monitoring the state of the cluster.

The Read Replicas are not involved in the Raft protocol, nor are they able to influence cluster topology. Hence a shared whiteboard outside of Raft comfortably scales to very large numbers of Read Replicas.

The whiteboard is kept up to date as Read Replicas join and leave the cluster, even if they fail abruptly rather than leaving gracefully.

C.1.5. Transacting via the Raft protocol

Once bootstrapped, each Core Server spends its time processing database transactions. Updates are reliably replicated around Core Servers via the Raft protocol. Updates appear in the form of a (committed) Raft log entry containing transaction commands which is subsequently applied to update the database.

> One of Raft's primary design goals is to be easily understandable so that there are fewer places for tricky bugs to hide in implementations. As a side-effect, it is also possible for database operators to reason about their Core Servers in their Causal Clusters.

The Raft Leader for the current term (a logical clock) appends the transaction (an 'entry' in Raft terminology) to the head of its local log and asks the other instances to do the same. When the Leader can see that a majority instances have appended the entry, it can be considered committed into the Raft log. The client application can now be informed that the transaction has safely committed since there is sufficient redundancy in the system to tolerate any (non-pathological) faults.

> The Raft protocol describes three roles that an instance can be playing: *Leader*, *Follower*, and *Candidate*. These are transient roles and any Core Server can expect to play them throughout the lifetime of a cluster. While it is interesting from a computing science point of view to understand those states, operators should not be overly concerned: they are an implementation detail.

For safety, within any Raft protocol instance there is only one Leader able to make forward progress in any given term. The Leader bears the responsibility for imposing order on Raft log entries and driving the log forward with respect to the *Followers*.

Followers maintain their logs with respect to the current Leader's log. Should any participant in the cluster suspect that the Leader has failed, then they can instigate a leadership election by entering the *Candidate* state. In Neo4j Core Servers this is happens at ms timescale, around 500ms by default.

Whichever instance is in the best state (including the existing Leader, if it remains available) can emerge from the election as Leader. The "best state" for a Leader is decided by highest term, then by longest log, then by highest committed entry.

The ability to fail over roles without losing data allows forward progress even in the event of faults. Even where Raft instances fail, the protocol can rapidly piece together which of the remaining instances is best placed to take over from the failed instance (or instances) **without data loss**. This is the essence of a *non-blocking* consensus protocol which allows Neo4j Causal Clustering to provide continuous availability to applications.

C.1.6. Catchup protocol

Read Replicas spend their time concurrently processing graph queries and applying a stream of transactions from the Core Servers to update their local graph store.

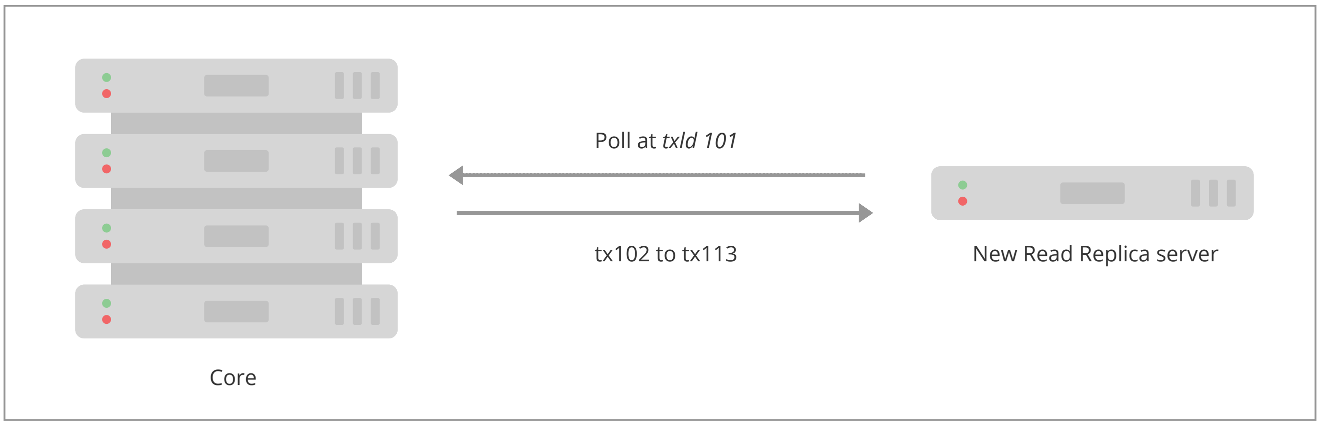

Figure 15. Transactions shipped from Core to Read Replica.

Updates from Core Servers to Read Replicas are propagated by transaction shipping. Transaction shipping is instigated by Read Replicas frequently *polling* any of the Core Servers specifying the ID of the last transaction they received and processed. The frequency of polling is an operational choice.

Neo4j transaction IDs are strictly monotonic integer values (they always increase). This makes it possible to determine whether or not a transaction has been applied to a Read Replica by comparing its last processed transaction ID with that of a Core Server.

If there is a large difference between an Read Replica's transaction history and that of a Core Server, polling may not result in any transactions being shipped. This is quite expected, for example when a new Read Replica is introduced to a long-running cluster or where a Read Replica has been down for some significant period of time. In such cases the catchup protocol will realize the gap between the Core Servers and Read Replica is too large to fill via transaction shipping and will fall back to copying the database store directly from Core Server to Read Replica. Since we are working with a live system, at the end of the database store copy the Core Server's database is likely to have changed. The Read Replica completes the catchup by asking for any transactions missed during the copy operation before becoming available.

A very slow database store copy could conceivably leave the Read Replica too far behind to catch up via transaction log shipping as the Core Server has substantially moved on. In such cases the Read Replica server repeats the catchup protocol. In pathological cases the operator can intervene to snapshot, restore, or file copy recent store files from a fast backup.

C.1.7. Read Replica shutdown

On clean shutdown, a Read Replica will invoke the discovery protocol to remove itself from the shared whiteboard overview of the cluster. It will also ensure that the database is cleanly shutdown and consistent, immediately ready for future use.

On an unclean shutdown such as a power outage, the Core Servers maintaining the overview of the cluster will notice that the Read Replica's connection has abruptly been cut. The discovery machinery will initially hide the Read Replica's whiteboard entry, and if the Read Replica does not reappear quickly its modest memory use in the shared whiteboard will be reclaimed.

On unclean shutdown it is possible the Read Replica will not have entirely consistent store files or transaction logs. On subsequent reboot the Read Replica will rollback any partially applied transactions such that the database is in a consistent state.

C.1.8. Core shutdown

A clean Core Server shutdown, like Core Server booting, is handled via the Raft protocol. When a Core Server is shut down, it appends a membership entry to the Raft log which is then replicated around the Core Servers. Once a majority of Core Servers have committed that membership entry the leaver has logically left the cluster and can safely shut down. All remaining instances accept that the cluster has grown smaller, and is therefore less fault tolerant. If the leaver happened to be playing the Leader role at the point of leaving, it will be transitioned to another Core Server after a brief election.

An unclean shutdown does not directly inform the cluster that a Core Server has left. Instead the Core cluster size remains the same for purposes of computing majorities for commits. Thus an unclean shutdown in a cluster of 5 Core Servers now requires 3/4 members to agree to commit which is a tighter margin than 3/5 before the unclean shutdown.

Of course when Core Servers fail, operators or monitoring scripts can be alerted so that they can intervene in the cluster if necessary.

If the leaver was playing the Leader role, there will be a brief election to produce a new Leader. Once the new Leader is established, the Core cluster continues albeit with less redundancy. However even with this failure, a Core cluster of 5 servers reduced to 4 can still tolerate one more fault before becoming read-only.

C.2. Multi-data center

This section introduces the multi-data center functionality in Neo4j.

Some use cases present high needs for availability, redundancy, locality of client applications, or simply scale. In these cases it is important that the cluster is aware of its physical topology so that it can optimize for workload. This makes configuring a single cluster to span multiple data centers a necessary proposition.

The following sections are dedicated to describing the different aspects of multi-data center operations of a Causal Cluster.

- Licensing for multi-data center operations
- Multi-data center design
  - Introduction
  - Core Server deployment scenarios
  - Allowing Read Replicas to catch up from other Read Replicas
- Multi-data center operations
  - Enable multi-data center operations
  - Server groups
  - Strategy plugins
- Multi-data center load balancing
  - Introduction
  - Prerequisite configuration
  - The load balancing framework
  - Load balancing examples
- Data center disaster recovery

    - Data center loss scenario
    - Procedure for recovering from data center loss

C.2.1. Licensing for multi-data center operations

Multi-data center functionality is intended for very demanding users of Neo4j who typically operate under a commercial database license. As a result, multi-data center functionality is licensed separately from the single-data center Causal Clustering features.

In order to confirm that you are operating under a suitable license, you must explicitly set the following in *neo4j.conf*:

```
causal_clustering.multi_dc_license=true
```

Without this configuration, all of the multi-data center features will remain disabled.

C.2.2. Multi-data center design

This section describes common patterns for multi-data center deployments that can act as building blocks for your own multi-data center production environments.

This section describes the following:

- Introduction
- Core Server deployment scenarios
- Allowing Read Replicas to catch up from other Read Replicas
    - Hierarchical Read Replica deployment
    - Catch up (mostly) from peer Read Replicas
    - Maintaining causal consistency in scale-out topologies

Introduction

This section is based on a series of examples to illustrate the different considerations we should take into account when designing our Causal Cluster for a multi-data center environment. We'll come to understand the weaknesses and benefits of common multi-data center deployment scenarios. Each scenario is presented at a high architectural level for clarity. In subsequent sections we will go into more detail on how such deployments are configured.

Core Server deployment scenarios

We will start with the conceptually simplest multi-data center scenario where we deploy the same number and kind of instances into each DC. This is a *homogeneous* deployment because each data center is identical to the other.

Example 112. Homogeneous three data center deployment

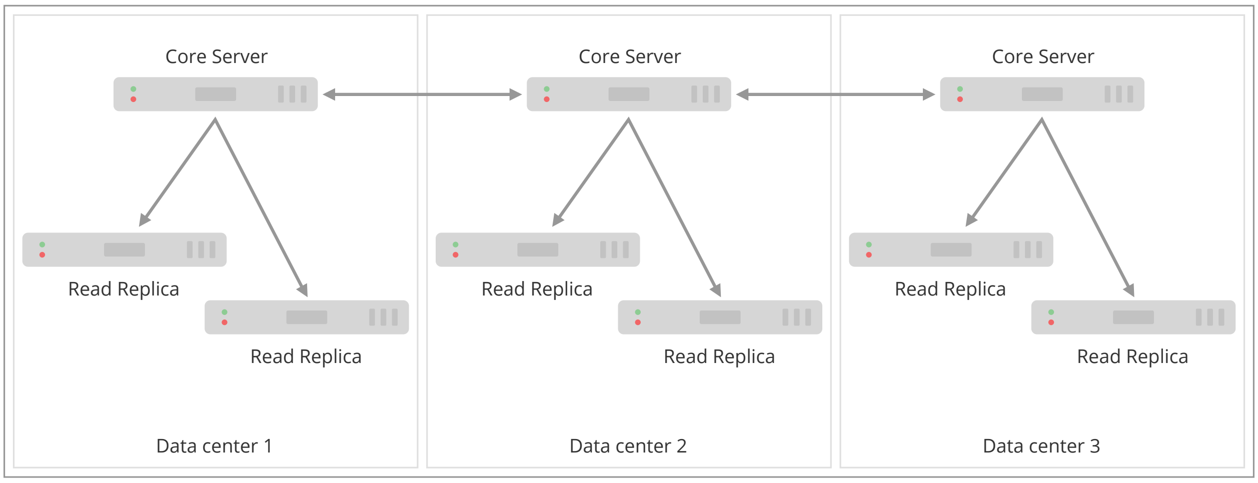

Figure 16. Homogeneous deployment across three data centers with one Core instance in each

In diagram above we have three data centers, each identically equipped with a single Core Server and a small number of Read Replicas.

Since Raft only requires a majority of the instances to acknowledge a write before it is safely committed, the latency of the commit path for this pattern involves only the two fastest data centers. As such the cost of committing to this setup is two WAN messages: one to send the transaction and one ACK message. In a non-failure case the other data center will not be far behind and will apply the transaction as well.

Within each of the data centers we can increase machine-level redundancy by adding more Core instances. For example we could add two more machines in each data center so that we can tolerate the spontaneous loss of up to four machines anywhere in the cluster or a single data center as a whole.

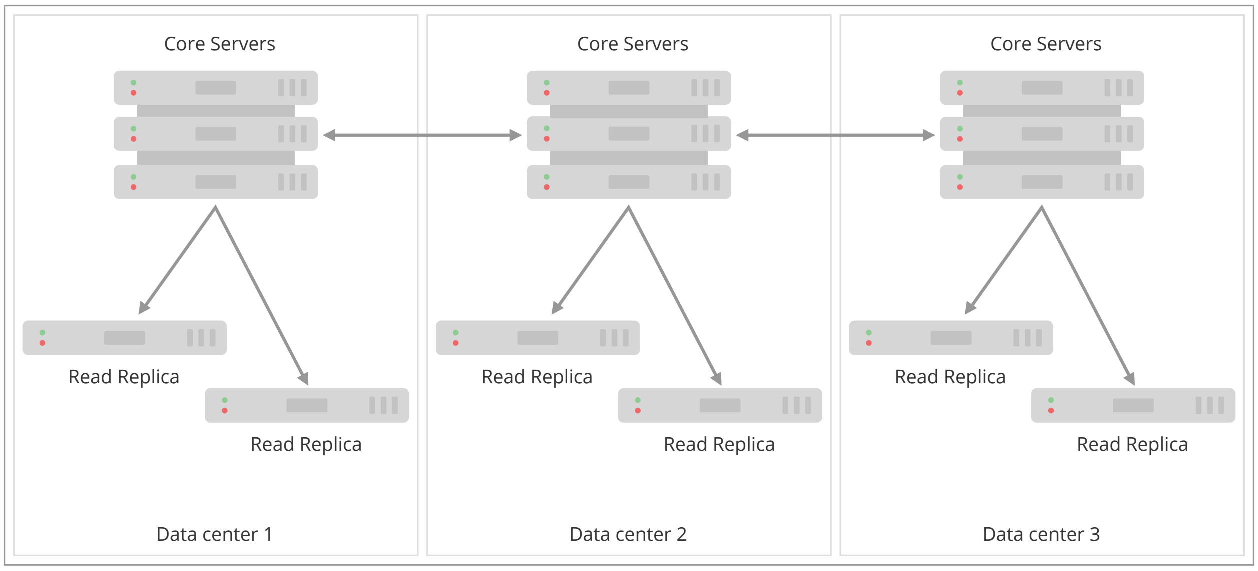

Figure 17. Homogeneous deployment across three data centers with three Core instances in each

To recap the strengths and weaknesses of this deployment pattern:

- We can lose an entire data center without losing availability and, depending on the number of machines in each data center, we may still be able to tolerate the loss of individual servers regardless of which data center they are in.
- The commit path for transactions is short, just two WAN messages exchanged.
- While the loss of majority data centers will need to be recovered, the operational procedure is identical irrespective of which of the data centers are lost.

As will be shown in the section on multi-data center configuration the Read Replicas can be biased to catchup from their data center-local Core Servers to minimize catchup latency. Data center-local client applications would also likely be routed to those same Read Replicas both for topological locality and scaling. More details are available in the section on multi-data center load balancing.

In the two data center case, our first instinct is to balance the available servers for operational consistency. An example of a homogeneous deployment across two data centers with two Core instances in each is illustrated in the diagram below:

Example 113. Homogeneous two data center deployment

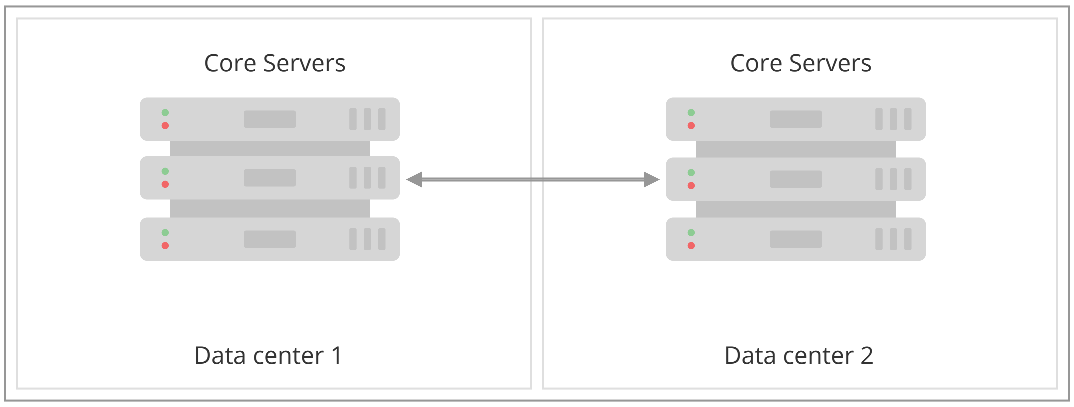

Figure 18. Homogeneous deployment across two data centers

The problem with this configuration is that while architecturally simple, it does not play to the strengths of the Raft protocol which is based on majority consensus. In the non-failure case, we incur two WAN messages to commit any transaction because a majority commit implies at least one response from the non-local data center instances. Worse, if we lose either data center the cluster will become read-only because it is impossible to achieve a majority.

As seen in the example above, the homogeneous deployment over two data centers does not take full advantage of the strengths of Causal Clustering. However it guarantees that the full Raft log will be present in either data center in the case of total data center loss.

The opposite of spreading Core Servers around our data centers, is to have them all hosted in a single one. This may be for technical or governance reasons, but either way has the advantage of LAN commit latencies for writes.

While our Core Servers are colocated, we spread out our Read Replicas close to the client applications to enable fan-out scaling.

Example 114. Core Servers and Read Replicas segregated by data center

The diagram below shows an example of a heterogeneous deployment directing writes to one data center, and reads to all. This pattern provides high survivability for data because of geo-replication. It also provides locality for client applications. However, if the Core Server data center is lost, we must immediately instigate recovery and turn one of the remaining Read Replica data centers into a new Core cluster.

It is possible that none of the Read Replicas have received all of the confirmed transactions prior to losing Data Center 1. While this is a convenient pattern for geo-replication, its semantics are best-effort. Cluster designers must take this aspect under consideration when deciding on recovery strategy.

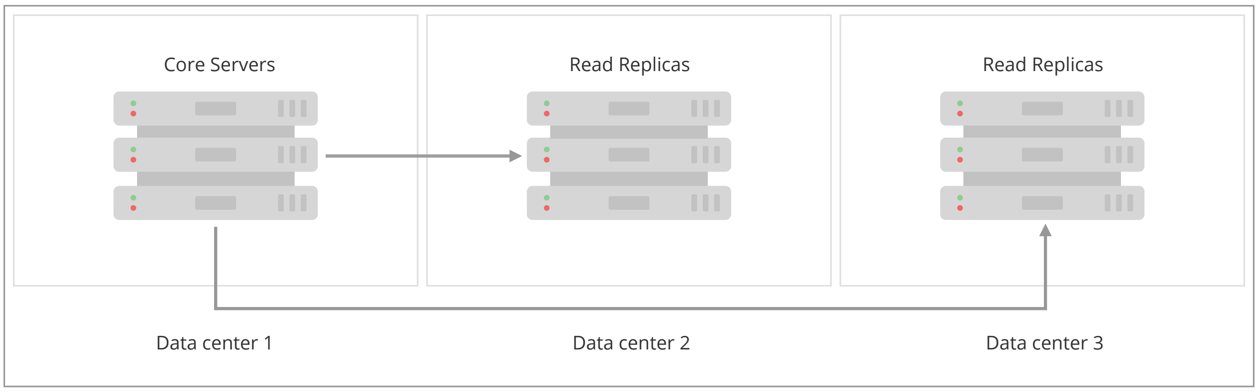

Figure 19. Heterogeneous deployment separating Read Replicas from the Core cluster

An operational tweak to this approach would be to host a Core Server in Data Center 2 and 3 as the starting point for recovery. During normal operations, these extra Core Servers should be configured with `causal_clustering.refuse_to_be_leader=true`. Should we lose Data Center 1, then we can use one of these Core Servers to quickly bootstrap a new Core cluster and return to full service rapidly.

To recap the strengths of this deployment pattern:

- Core Servers commit at LAN latencies if using the setup with Core Servers exclusively in one data center.
- Read Replicas provide scale and locality for client applications.
- Geo-replication provides high survivability for data.

Allowing Read Replicas to catch up from other Read Replicas

With an understanding of the basic multi-data center patterns at our disposal, we can refine our deployment models to embrace local catchup within data centers. This means that any server, including Read Replicas, can act as a source of transactions for Read Replica server. When catching up from data center-local instances we aim to amortize the cost of WAN traffic catchup across many local replications.

Allowing Read Replicas to choose a data center-local Core Server or even another Read Replica gives us a great deal of design freedom, and importantly allows us to scale to truly huge numbers of Read Replicas. Using this feature we might choose to fan-out Read Replicas so that the catchup load on the Core Servers grows (approximately) logarithmically rather than linearly.

Hierarchical Read Replica deployment

The primary motivation for Read Replicas catching up from other Read Replicas is to allow for fan-out scale. To achieve a fan-out we arrange the Read Replicas in a hierarchy, with each layer of the hierarchy being broader than the one above.

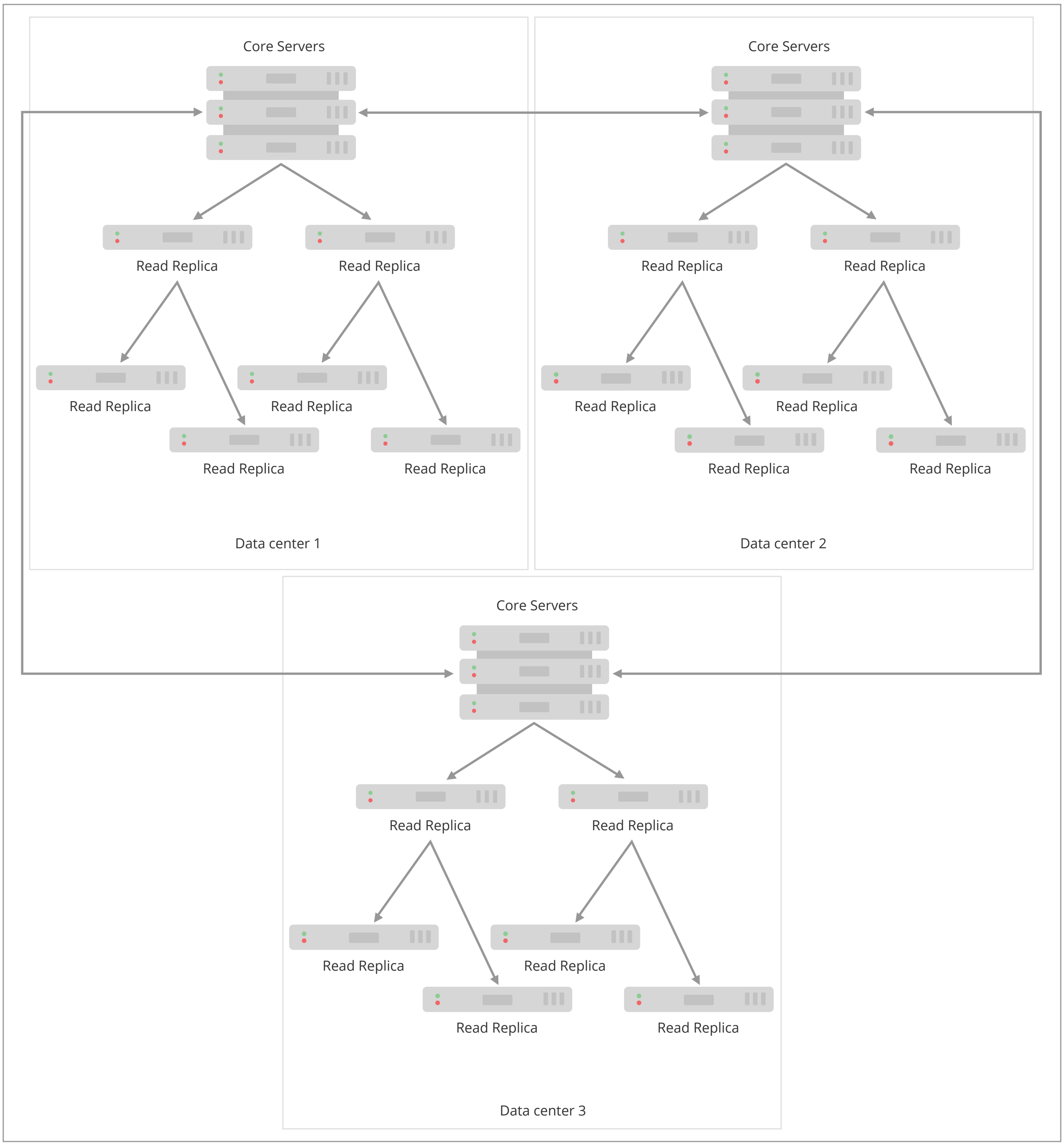

Figure 20. Fan out from Core Servers for scale at log cost

An illustrative hierarchy is presented in the diagram above. The Core Servers supply transactions to a relatively small number of Read Replicas at the first tier. This results in a relatively modest load on the Core Servers, freeing up resources to focus on the commit path. Those Read Replicas in the first tier in turn feed a larger number of Read Replicas in the second tier. This pattern can be reasonably extended to several tiers to provide enormous fan-out.

At each tier we expand the scalability of the Read Replicas, but we add another level of catchup latency. By careful measurement we can ascertain the appropriate depth and breadth of the hierarchy to match the application requirements.

We should also take care that each tier in the hierarchy has sufficient redundancy so that failures do not compromise transmission of data from the Core Servers. A strategy for keeping Read Replicas current in the presence of failures is to occasionally have them subvert the hierarchy. That is, if a given Read Replica occasionally goes to its grandparents or even directly to the Core Servers then we can avoid pathologically high replication latencies under fault conditions.

Catch up (mostly) from peer Read Replicas

Another strategy for Read Replica catchup is to treat them all as peers and have peer-to-peer catchup. This avoids the need to manage tiers of replicas to maintain availability since the Read Replicas catch up from one another in a mesh.

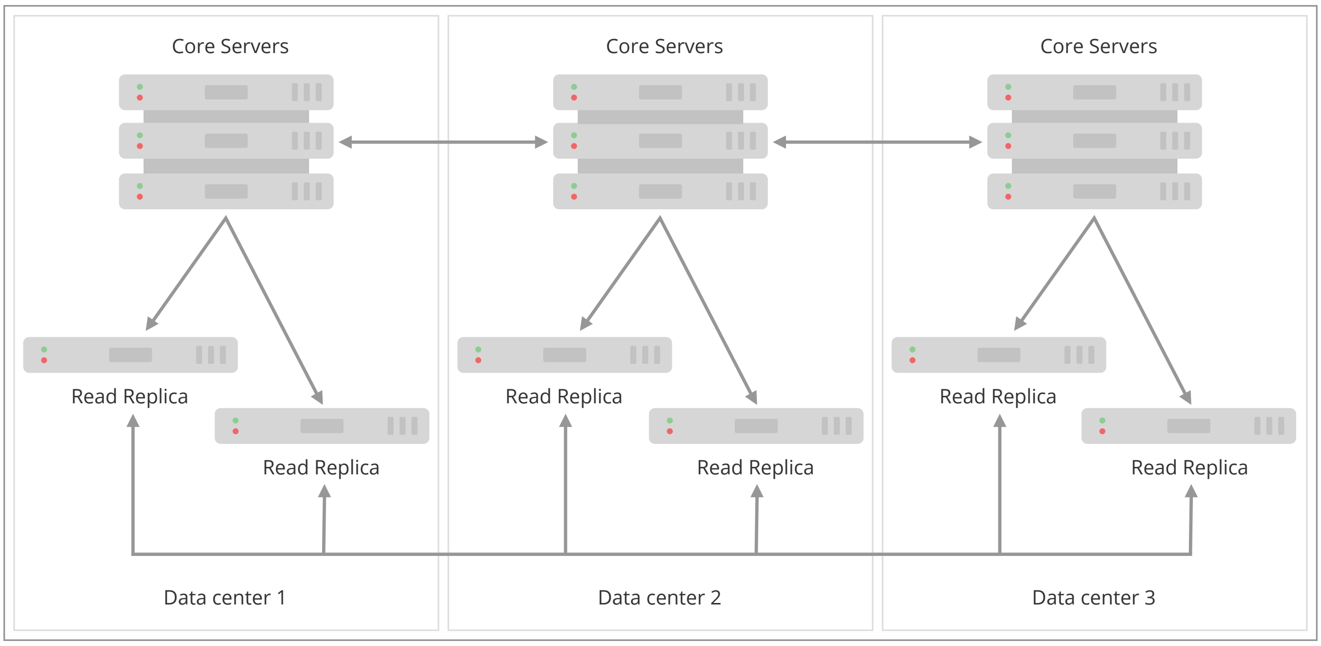

Figure 21. Peer-to-peer Read Replica catchup

Having a reduced load on the Core Servers allows us to scale out. For example if only one in ten catchup requests goes to the Core Servers, we could expand the number of Read Replicas by approximately a factor of 10.

To avoid groups of orphans in the mesh, Read Replicas will occasionally catch up directly from Core Servers. Having Read Replicas catch up with Core Servers ensures that no Read Replica is left behind indefinitely, placing an upper bound on replication latency. While this places some load on the Core Servers, it is far less than if all catch up attempts from Read Replicas were directed to a Core Server.

The upper bound on replication latency for this mode of operation is the number of catchup attempts served by Read Replicas before trying core. The average replication latency will be half the number of attempts to replicate. This is because on average half the Read Replicas will be ahead and half behind any given Read Replica.

Connecting to a random Core Server on failure to retrieve updates from other sources is the default behavior of Read Replicas.

Maintaining causal consistency in scale-out topologies

Causal consistency is always maintained, even in extreme situations with chains of Read Replicas catching up from other upstream Read Replicas. The key trade-off to understand, as so often in distributed systems, is that of latency for scale.

In Fan out from Core Servers for scale at log cost we see that number of hops required for a transaction to propagate to the lowest tier is 2: the highest latency in this topology. Equally we see how the bottommost tier has far more members than any other tier giving it scale advantages.

Correspondingly, in the middle tier we have better latency (one hop) but less scale. At the top most tier (Core Servers) we have very little latency (just the Raft commit path) but the fewest available servers. This means we should target queries at the most appropriate tier based on latency, scale, and locality.

Summary on latency versus scalability:

- Issuing read queries to a Core Server generally has the lowest latency in principle but may have the highest contention.
- Issuing read queries to a Read Replica topologically closest to Core Servers typically has higher latency but also higher scalability.
- Issuing read queries to a Read Replica topologically further from Core Servers typically has the highest latency but also the highest scalability.

In large systems like the scale-out hierarchy above, we are conventionally used to having relaxed or *eventual* consistency semantics. With Neo4j multi-data center setups, that is also possible. Where we don't care about causality we can read from any Read Replica and accept that we might see older values. However the causal consistency semantics are maintained.

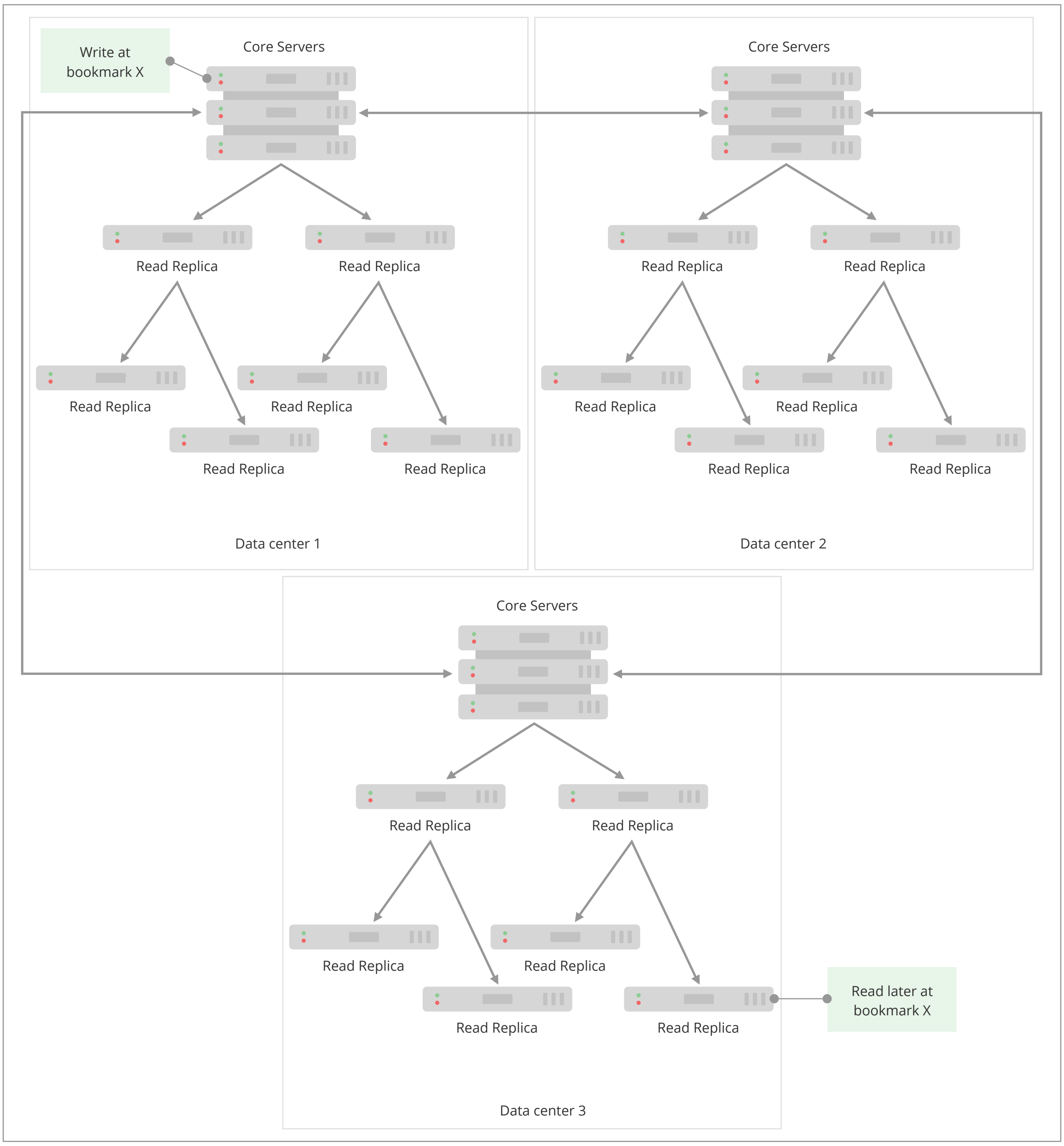

Figure 22. Each tier in the Read Replicas is further behind the source of truth, but offers greater scale-out

As we can see in diagram above, even if the client binds to a Read Replica that is multiple hops/data centers away from the source of truth, causal consistency is maintained. While the query may be suspended while the necessary transaction propagates to the Read Replica, the benefit is that there will be more Read Replicas available and so overall client throughput is higher than with a single-tier configuration.

C.2.3. Multi-data center operations

This section shows how to configure Neo4j servers so that they are topology/data center-aware. It describes the precise configuration needed to achieve a scalable multi-data center deployment.

This section describes the following:

- Enable multi-data center operations
- Server groups
- Strategy plugins
  - Configuring upstream selection strategy using pre-defined strategies
  - Configuring user-defined strategies
  - Building upstream strategy plugins using Java
  - Favoring data centers

Enable multi-data center operations

Before doing anything else, we must enable the multi-data center functionality. This is described in Licensing for multi-data center operations.

Licensing for multi-data center
The multi-data center functionality is separately licensed and must be specifically enabled.

Server groups

In order to optimize the use of our Causal Cluster servers according to our specific requirements, we sort them into *Server Groups*. Server Group membership can map to data centers, availability zones, or any other significant topological elements from the operator's domain. Server Groups can also overlap.

Server Groups are defined as a key that maps onto a set of servers in a Causal Cluster. Server Group membership is defined on each server using the `causal_clustering.server_groups` parameter in *neo4j.conf*. Each server in a Causal Cluster can belong to zero or more server groups.

Example 115. Definition of Server Group membership

The membership of a server group or groups can be set in *neo4j.conf* as in the following examples:

```
# Add the current instance to the groups `us` and `us-east`
causal_clustering.server_groups=us,us-east
```

```
# Add the current instance into the group `london`
causal_clustering.server_groups=london
```

```
# Add the current instance into the group `eu`
causal_clustering.server_groups=eu
```

We must be aware that membership of each server group is explicit. For example, a server in the `gb-london` group is not automatically part of some `gb` or `eu` group unless that server is explicitly added to those groups. That is, any (implied) relationship between groups is reified only when those groups are used as the basis for requesting data from upstream systems.

Server Groups are not mandatory, but unless they are present, we cannot set up specific upstream transaction dependencies for servers. In the absence of any specified server groups, the cluster defaults to its most pessimistic fall-back behavior: each Read Replica will catch up from a random Core Server.

Strategy plugins

Strategy plugins are sets of rules that define how Read Replicas contact servers in the cluster in order to synchronize transaction logs. Neo4j comes with a set of pre-defined strategies, and also provides a Design Specific Language, *DSL*, to flexibly create user-defined strategies. Finally, Neo4j supports an API which advanced users may use to enhance upstream recommendations.

Once a strategy plugin resolves a satisfactory upstream server, it is used for pulling transactions to update the local Read Replica for a single synchronization. For subsequent updates, the procedure is repeated so that the most preferred available upstream server is always resolved.

Configuring upstream selection strategy using pre-defined strategies

Neo4j ships with the following pre-defined strategy plugins. These provide coarse-grained algorithms for choosing an upstream instance:

Plugin name	Resulting behavior
`connect-to-random-core-server`	Connect to any **Core Server** selecting at random from those currently available.
`typically-connect-to-random-read-replica`	Connect to any available **Read Replica**, but around 10% of the time connect to any random Core Server.
`connect-randomly-to-server-group`	Connect at random to any available **Read Replica** in any of the server groups specified in the comma-separated list `causal_clustering.connect-randomly-to-server-group`.
`leader-only`	Connect only to the current Raft leader of the Core Servers.
`connect-randomly-within-server-group`	Connect at random to any available **Read Replica** in any of the server groups to which this server belongs. Deprecated, please use `connect-randomly-to-server-group`.

Pre-defined strategies are used by configuring the `causal_clustering.upstream_selection_strategy` option. Doing so allows us to specify an ordered preference of strategies to resolve an upstream provider of transaction data. We provide a comma-separated list of strategy plugin names with preferred strategies earlier in that list. The upstream strategy is chosen by asking each of the strategies in list-order whether they can provide an upstream server from which transactions can be pulled.

Example 116. Define an upstream selection strategy

Consider the following configuration example:

```
causal_clustering.upstream_selection_strategy=connect-randomly-to-server-group,typically-connect-to-random-read-replica
```

With this configuration the instance will first try to connect to any other instance in the group(s) specified in `config_causal_clustering.connect-randomly-to-server-group`. Should we fail to find any live instances in those groups, then we will connect to a random Read Replica.

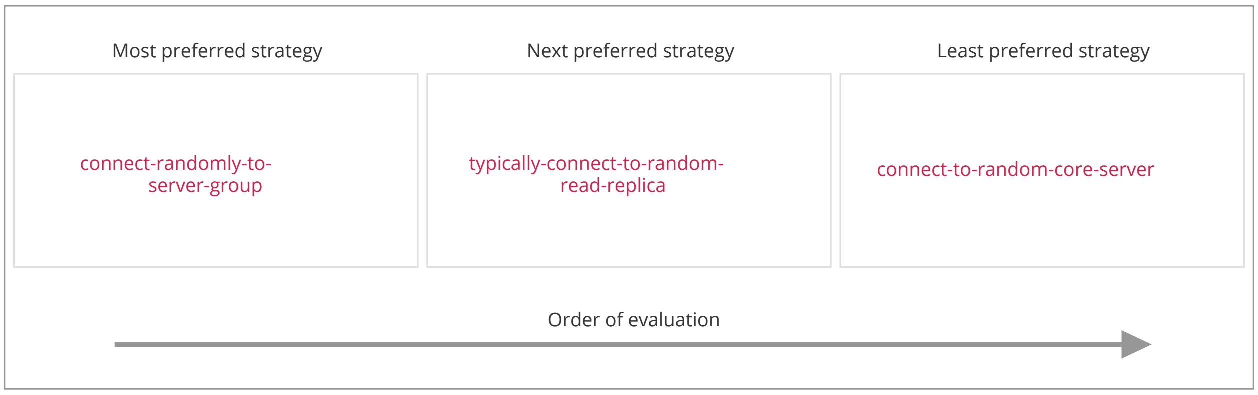
Most preferred strategy
Next preferred strategy
Least preferred strategy
connect-randomly-to-server-group
typically-connect-to-random-read-replica
connect-to-random-core-server
Order of evaluation

Figure 23. The first satisfactory response from a strategy will be used.

To ensure that downstream servers can still access live data in the event of upstream failures, the last resort of any instance is always to contact a random Core Server. This is equivalent to ending the `causal_clustering.upstream_selection_strategy` configuration with `connect-to-random-core-server`.

Configuring user-defined strategies

Neo4j Causal Clusters support a small DSL for the configuration of client-cluster load balancing. This is described in detail in Policy definitions and Filters. The same DSL is used to describe preferences for how an instance binds to another instance to request transaction updates.

The DSL is made available by selecting the `user-defined` strategy as follows:

```
causal_clustering.upstream_selection_strategy=user-defined
```

Once the user-defined strategy has been specified, we can add configuration to the `causal_clustering.user_defined_upstream_strategy` setting based on the server groups that have been set for the cluster.

We will describe this functionality with two examples:

Example 117. Defining a user-defined strategy

For illustrative purposes we propose four regions: `north`, `south`, `east`, and `west` and within each region we have a number of data centers such as `north1` or `west2`. We configure our server groups so that each data center maps to its own server group. Additionally we will assume that each data center fails independently from the others and that a region can act as a supergroup of its constituent data centers. So an instance in the `north` region might have configuration like `causal_clustering.server_groups=north2,north` which puts it in two groups that match to our physical topology as shown in the diagram below.

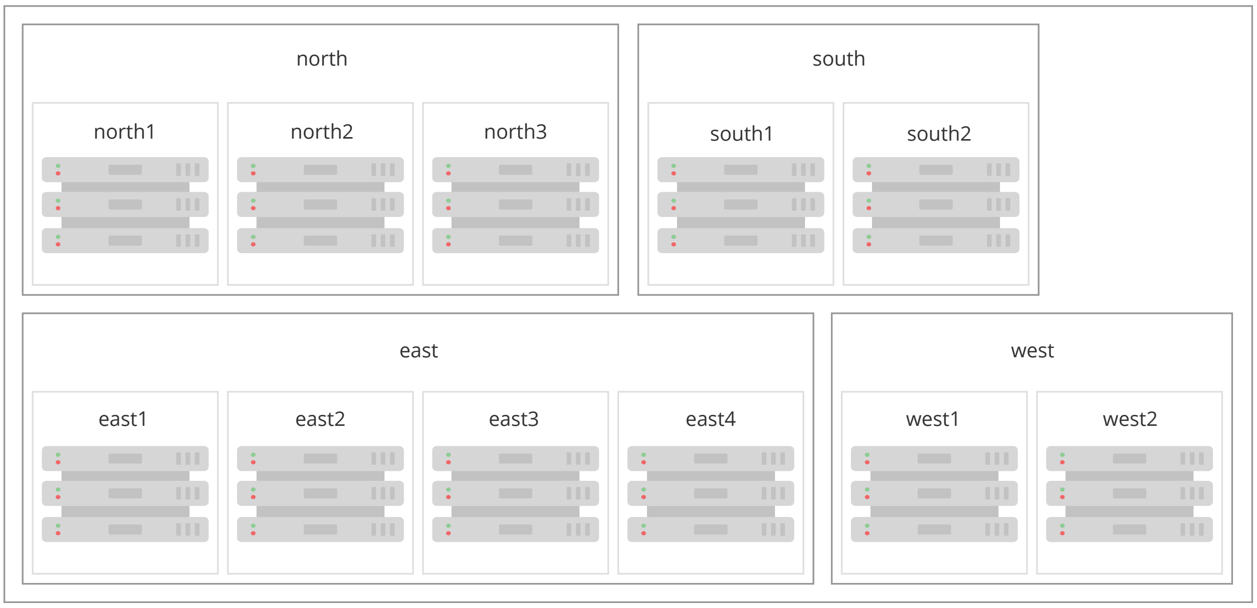

Figure 24. Mapping regions and data centers onto server groups

Once we have our server groups, our next task is to define some upstream selection rules based on them. For our design purposes, let's say that any instance in one of the `north` region data centers prefers to catchup within the data center if it can, but will resort to any northern instance otherwise. To configure that behavior we add:

```
causal_clustering.user_defined_upstream_strategy=groups(north2); groups(north); halt()
```

The configuration is in precedence order from left to right. The `groups()` operator yields a server group from which to catch up. In this case only if there are no servers in the `north2` server group will we proceed to the `groups(north)` rule which yields any server in the `north` server group. Finally, if we cannot resolve any servers in any of the previous groups, then we will stop the rule chain via `halt()`.

Note that the use of `halt()` will end the rule chain explicitly. If we don't use `halt()` at the end of the rule chain, then the `all()` rule is implicitly added. `all()` is expansive: it offers up all servers and so increases the likelihood of finding an available upstream server. However `all()` is indiscriminate and the servers it offers are not guaranteed to be topologically or geographically local, potentially increasing the latency of synchronization.

The example above shows a simple hierarchy of preferences. But we can be more sophisticated if we so choose. For example we can place conditions on the server groups from which we catch up.

Example 118. User-defined strategy with conditions

In this example we wish to roughly qualify cluster health before choosing from where to catch up. For this we use the `min()` filter as follows:

```
causal_clustering.user_defined_upstream_strategy=groups(north2)->min(3), groups(north)->min(3);
all();
```

`groups(north2)->min(3)` states that we want to catch up from the `north2` server group if it has three available machines, which we here take as an indicator of good health. If `north2` can't meet that requirement (is not healthy enough) then we try to catch up from any server across the `north` region provided there are at least three of them available as per `groups(north)->min(3)`. Finally, if we cannot catch up from a sufficiently healthy `north` region, then we'll (explicitly) fall back to the whole cluster with `all()`.

The `min()` filter is a simple but reasonable indicator of server group health.

Building upstream strategy plugins using Java

Neo4j supports an API which advanced users may use to enhance upstream recommendations in arbitrary ways: load, subnet, machine size, or anything else accessible from the JVM. In such cases we are invited to build our own implementations of `org.neo4j.causalclustering.readreplica.UpstreamDatabaseSelectionStrategy` to suit our own needs, and register them with the strategy selection pipeline just like the pre-packaged plugins.

We have to override the `org.neo4j.causalclustering.readreplica.UpstreamDatabaseSelectionStrategy#upstreamDatabase()` method in our code. Overriding that class gives us access to the following items:

Resource	Description
`org.neo4j.causalclustering.discovery.TopologyService`	This is a directory service which provides access to the addresses of all servers and server groups in the cluster.
`org.neo4j.kernel.configuration.Config`	This provides the configuration from *neo4j.conf* for the local instance. Configuration for our own plugin can reside here.
`org.neo4j.causalclustering.identity.MemberId`	This provides the unique cluster `MemberId` of the current instance.

Once our code is written and tested, we have to prepare it for deployment. `UpstreamDatabaseSelectionStrategy` plugins are loaded via the Java Service Loader. This means when we package our code into a jar file, we'll have to create a file *META-INF.services/org.neo4j.causalclustering.readreplica.UpstreamDatabaseSelectionStrategy* in which we write the fully qualified class name(s) of the plugins, e.g. `org.example.myplugins.PreferServersWithHighIOPS`.

To deploy this jar into the Neo4j server we copy it into the *plugins* directory and restart the instance.

Favoring data centers

In a multi-DC scenario, while it remains a rare occurrence, it is possible to bias which data centers are used to host Raft leaders (and thus where writes are directed). To do so, we can apply `causal_clustering.refuse_to_be_leader=true` on the leaders in the data centers where we do not want leaders to materialize. In doing so we implicitly prefer the instances where we have **not** applied that setting.

This may be useful when planning for highly distributed multi-data center deployments. However this must be very carefully considered because in failure scenarios it limits the availability of the cluster. It is advisable to engage Neo4j Professional Services to help design a suitably resilient topology.

C.2.4. Multi-data center load balancing

This section describes the topology-aware load balancing options available for client applications in a multi-data center Neo4j deployment. It describes how to configure the load balancing for the cluster so that client applications can direct its workload at the most appropriate cluster members, such as those nearby.

This section describes the following:

- Introduction
- Prerequisite configuration
  - Enable multi-data center operations
  - Server groups
  - Cores for reading
- The load balancing framework
  - Policy definitions
  - Policy names
  - Filters
- Load balancing examples

> *Enabling load balancing*
>
> The load balancing functionality is part of the separately licensed multi-data center package and must be specifically enabled. See Licensing for multi-data center operations for details.

Introduction

When deploying a multi-data center cluster we often wish to take advantage of locality to reduce latency and improve performance. For example, we would like our graph-intensive workloads to be executed in the local data center at LAN latencies rather than in a faraway data center at WAN latencies. Neo4j's enhanced load balancing for multi-data center scenarios facilitates precisely this and can also be used to define fall-back behaviors. This means that failures can be planned for upfront and persistent overload conditions be avoided.

The load balancing system is a cooperative system where the driver asks the cluster on a recurring basis where it should direct the different classes of its workload (e.g. writes and reads). This allows the driver to work independently for long stretches of time, yet check back from time to time to adapt to changes like for example a new server having been added for increased capacity. There are also failure situations where the driver will ask again immediately, for example when it cannot use any of its allocated servers.

This is mostly transparent from the perspective of a client. On the server side we configure the load balancing behaviors and expose them under a named *load balancing policy* which the driver can bind to. All server-side configuration is performed on the Core Servers.

Use load balancing from Neo4j drivers

This chapter describes how to configure a Causal Cluster to use custom load balancing policies Once enabled and configured, the custom load balancing feature is used by drivers to route traffic as intended. See the Driver Manual for instructions on how to configure drivers to use custom load balancing.

Prerequisite configuration

Enable multi-data center operations

In order to configure a cluster for load balancing we must enable the multi-data center functionality. This is described in Licensing for multi-data center operations.

Server groups

In common with server-to-server catchup, load balancing across multiple data centers is predicated on the *server group* concept. Servers can belong to one or more potentially overlapping server groups, and decisions about where to route requests from client to cluster member are parameterized based on that configuration. For details on server group configuration, refer to Server groups.

Cores for reading

Depending on the deployment and the available number of servers in the cluster different strategies make sense for whether or not the reading workload should be routed to the Core Servers. The following configuration will allow the routing of read workload to Core Servers. Valid values are `true` and `false`.

```
causal_clustering.cluster_allow_reads_on_followers=true
```

The load balancing framework

The load balancing system is based on a plugin architecture for future extensibility and for allowing user customizations. The current version ships with exactly one such canned plugin called the *server policies* plugin.

The server policies plugin is selected by setting the following property:

```
causal_clustering.load_balancing.plugin=server_policies
```

Under the server policies plugin, a number of load balancing policies can be configured server-side and be exposed to drivers under unique names. The drivers, in turn, must on instantiation select an appropriate policy by specifying its name. Common patterns for naming policies are after geographical regions or intended application groups.

It is of crucial importance to define the exact same policies on all core machines since this is to be regarded as cluster-wide configuration and failure to do so will lead to surprising behavior. Similarly, policies which are in active use should not be removed or renamed since it will break applications trying to use these policies. It is perfectly acceptable and expected however that policies be modified under the same name.

If a driver asks for a policy name which is not available, then it will not be able to use the cluster. A driver which does not specify any name at all will get the behavior of the default policy as configured. The default policy, if left unchanged, distributes the load across all servers. It is possible to change the default policy to any behavior that a named policy can have.

A misconfigured driver or load balancing policy will result in suboptimal routing choices or even prevent successful interactions with the cluster entirely.

The details of how to write a custom plugin is not documented here. Please get in contact with Neo4j Professional Services if you think that you need a custom plugin.

Policy definitions

The configuration of load balancing policies is transparent to client applications and expressed via a simple DSL. The syntax consists of a set of rules which are considered in order. The first rule to produce a non-empty result will be the final result.

```
rule1; rule2; rule3
```

Each rule in turn consists of a set of filters which limit the considered servers, starting with the complete set. Note that the evaluation of each rule starts fresh with the complete set of available servers.

There is a fixed set of filters which compose a rule and they are chained together using arrows

```
filter1 -> filter2 -> filter3
```

If there are any servers still left after the last filter then the rule evaluation has produced a result and this will be returned to the driver. However, if there are no servers left then the next rule will be considered. If no rule is able to produce a usable result then the driver will be signalled a failure.

Policy names

The policies are configured under the namespace of the *server policies* plugin and named as desired. Policy names can contain alphanumeric characters and underscores, and they are case sensitive. Below is the property key for a policy with the name `mypolicy`.

`causal_clustering.load_balancing.config.server_policies.mypolicy=`

The actual policy is defined in the value part using the DSL.

The `default` policy name is reserved for the default policy. It is possible to configure this policy like any other and it will be used by driver clients which do not specify a policy.

Additionally, any number of policies can be created using unique policy names. The policy name can suggest a particular region or an application for which it is intended to be used.

Filters

There are four filters available for specifying rules, detailed below. The syntax is similar to a method call with parameters.

- `groups(name1, name2, …)`
  - Only servers which are part of any of the specified groups will pass the filter.
  - The defined names must match those of the *server groups*.
- `min(count)`
  - Only the minimum amount of servers will be allowed to pass (or none).
  - Allows overload conditions to be managed.

* `all()`
  * No need to specify since it is implicit at the beginning of each rule.
  * Implicitly the last rule (override this behavior using halt).
* `halt()`
  * Only makes sense as the last filter in the last rule.
  * Will stop the processing of any more rules.

The groups filter is essentially an OR-filter, e.g. `groups(A,B)` which will pass any server in either A, B or both (the union of the server groups). An AND-filter can also be created by chaining two filters as in `groups(A) -> groups(B)`, which will only pass servers in both groups (the intersect of the server groups).

Load balancing examples

In our discussion on multi-data center clusters we introduced a four region, multi-data center setup. We used the cardinal compass points for regions and numbered data centers within those regions. We'll use the same hypothetical setup here too.

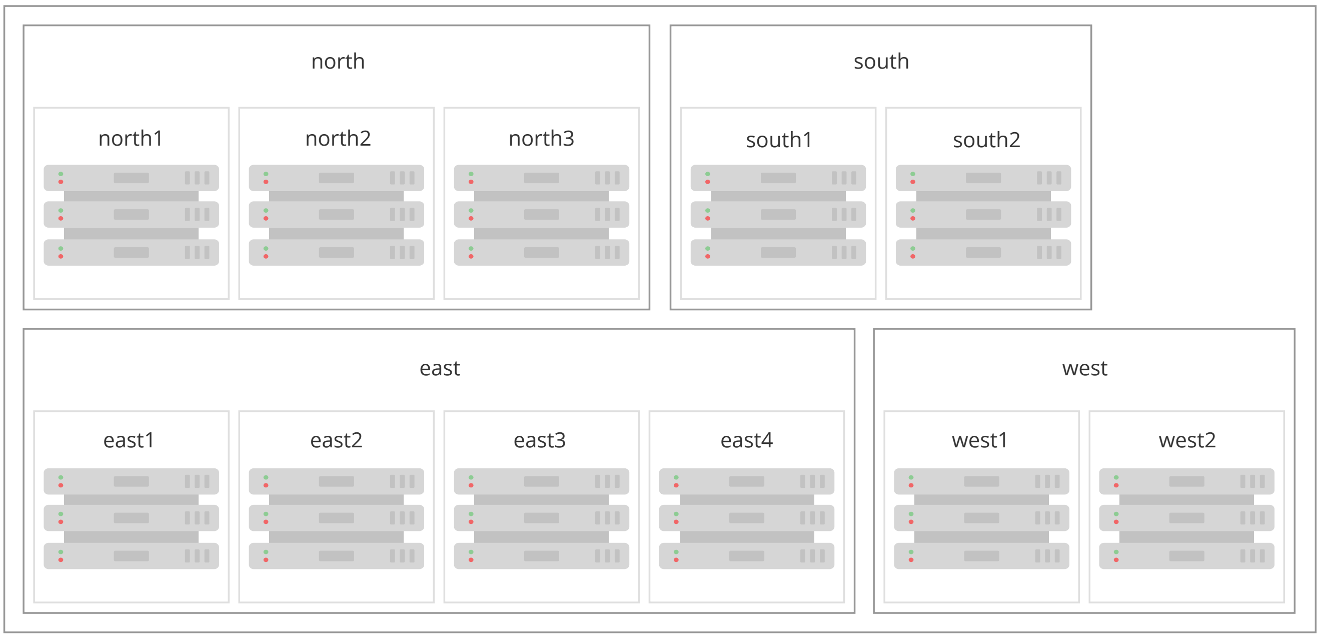

Figure 25. Mapping regions and data centers onto server groups

We configure the behavior of the load balancer in the property `causal_clustering.load_balancing.config.server_policies.<policy-name>`. The rules we specify will allow us to fine tune how the cluster routes requests under load.

In the examples we will make use of the line continuation character `\` for better readability. It is valid syntax in *neo4j.conf* as well and it is recommended to break up complicated rule definitions using this and a new rule on every line.

The most restrictive strategy would be to insist on a particular data center to the exclusion of all others:

Example 119. Specific data center only

```
causal_clustering.load_balancing.config.server_policies.north1_only=\
groups(north1)->min(2); halt();
```

In this case we're stating that we are only interested in sending queries to servers in the `north1` server group, which maps onto a specific physical data center, provided there are two of them available. If we cannot provide at least two servers in `north1` then we should `halt()`, i.e. not try any other data center.

While the previous example demonstrates the basic form of our load balancing rules, we can be a little more expansive:

Example 120. Specific data center preferably

```
causal_clustering.load_balancing.config.server_policies.north1=\
groups(north1)->min(2);
```

In this case if at least two servers are available in the `north1` data center then we will load balance across them. Otherwise we will use any server in the whole cluster, falling back to the implicit, final `all()` rule.

The previous example considered only a single data center before resorting to the whole cluster. If we have a hierarchy or region concept exposed through our server groups we can make the fall back more graceful:

Example 121. Gracefully falling back to neighbors

```
causal_clustering.load_balancing.config.server_policies.north_app1=\
groups(north1,north2)->min(2);\
groups(north);\
all();
```

In this case we're saying that the cluster should load balance across the `north1` and `north2` data centers provided there are at least two machines available across them. Failing that, we'll resort to any instance in the `north` region, and if the whole of the north is offline we'll resort to any instances in the cluster.

C.2.5. Data center disaster recovery

This section describes how to recover your Neo4j Causal Cluster following a data center failure. Specifically it covers safely turning a small number of surviving instances from a read-only state back into a fully operational cluster of read/write instances.

This section describes the following:

- Data center loss scenario
- Procedure for recovering from data center loss

Data center loss scenario

This section describes how to recover a multi-data center deployment which owing to external circumstances has reduced the cluster below half of its members. It is most easily typified by a 2x2 deployment with 2 data centers each containing two instances. This deployment topology can either arise because of other data center failures, or be a deliberate choice to ensure the geographic survival of data for catastrophe planning. However, by distributing an instance over three data centers instead, you could avoid having the cluster lose quorum through a single data center failure. For example, in a *1x1x1* deployment.

Under normal operation this provides a stable majority quorum where the fastest three out of four machines will execute users' transactions, as we see highlighted in Two Data Center Deployment with Four Core Instances.

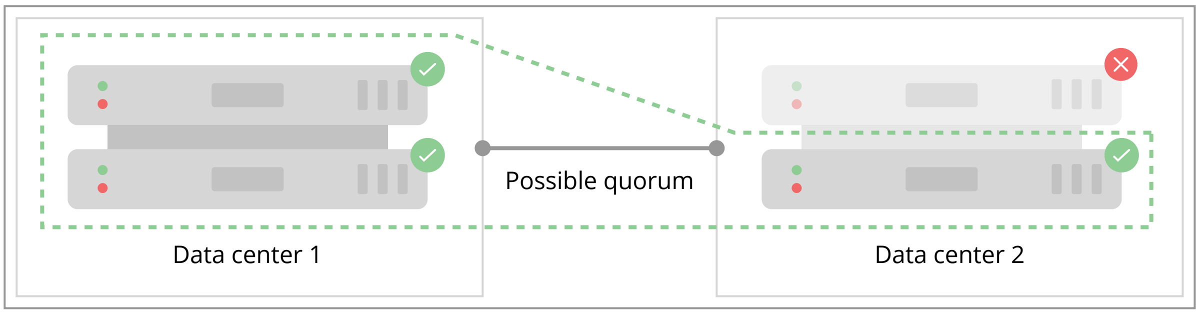

Figure 26. Two Data Center Deployment with Four Core Instances

However if an entire data center becomes offline because of some disaster, then a *majority quorum* cannot be formed in this case.

Neo4j Core clusters are based on the Raft consensus protocol for processing transactions. The Raft protocol requires a majority of cluster members to agree in order to ensure the safety of the cluster and data. As such, the loss of a majority quorum results in a read-only situation for the remaining cluster members.

When data center is lost abruptly in a disaster rather than having the instances cleanly shut down, the surviving members still believe that they are part of a larger cluster. This is different from even the case of rapid failures of individual instances in a live data center which can often be detected by the underlying cluster middleware, allowing the cluster to automatically reconfigure.

Conversely if we lose a data center, there is no opportunity for the cluster to automatically reconfigure. The loss appears instantaneous to other cluster members. However, because each remaining machine has only a partial view of the state of the cluster (its own), it is not safe to allow any individual machine to make an arbitrary decision to reform the cluster.

In this case we are left with two surviving machines which cannot form a quorum and thus make progress.

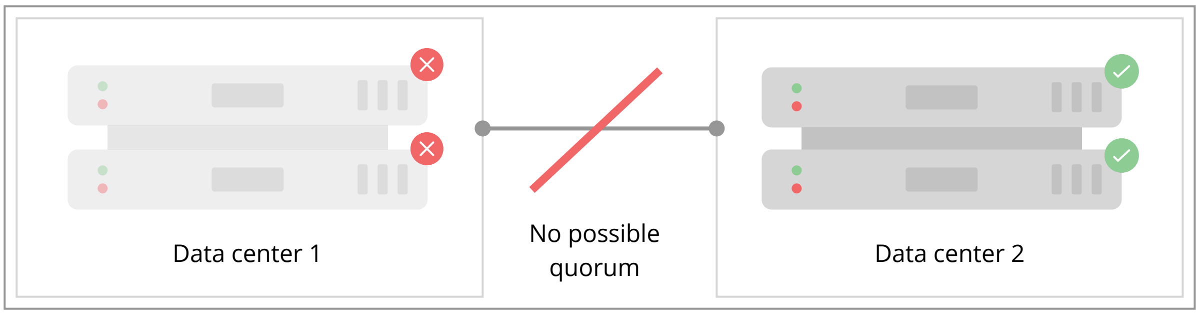

Figure 27. Data Center Loss Requires Guided Recovery

But, from a birds’s eye view, it’s clear we have surviving machines which are sufficient to allow a non-fault tolerant cluster to form under operator supervision.

> Groups of individual cluster members (e.g. those in a single data center) may become isolated from the cluster during network partition for example. If they arbitrarily reformed a new, smaller cluster there is a risk of *split-brain*. That is from the clients' point of view there may be *two* or more smaller clusters that are available for reads and writes depending on the nature of the partition. Such situations lead to divergence that is tricky and laborious to reconcile and so best avoided.

To be safe, an operator or other out-of-band agent (e.g. scripts triggered by well-understood, trustworthy alerts) that has a trusted view on the whole of the system estate must make that decision. In the surviving data center, the cluster can be rebooted into a smaller configuration whilst retaining all data committed to that point. While end users may experience unavailability during the switch over, no committed data will be lost.

Procedure for recovering from data center loss

The following procedure for performing recovery of a data center should not be done lightly. It assumes that we are completely confident that a disaster has occurred and our previously data center-spanning cluster has been reduced to a read-only cluster in a single data center, where there is no possible way to repair a connection to the lost instances. Further it assumes that the remaining cluster members are fit to provide a seed from which a new cluster can be created from a data quality point of view.

Having acknowledged the above, the procedure for returning the cluster to full availability following catastrophic loss of all but one data centers can be done using one of the following options, depending on your infrastructure.

Please note that the main difference between the options is that Option 2 will allow read-availability during recovery.

Option 1.

If you are unable to add instances to the current data-center, and can only use the current read-only cluster, the following steps are recommended:

1. Verify that a catastrophe has occurred, and that access to the surviving members of the cluster in the surviving data center is possible. Then for each instance:
    a. Stop the instance with `bin/neo4j stop` or shut down the service.
    b. Change the configuration in *neo4j.conf* such that the `causal_clustering.initial_discovery_members` property contains the DNS names or IP addresses of the other surviving instances.
    c. Optional: you may need to update `causal_clustering.minimum_core_cluster_size_at_formation`, depending on the current size of the cluster (in the current example, two cores).
    d. Unbind the instance using `neo4j-admin unbind`.
    e. Start the instance with `bin/neo4j start` or start the `neo4j` service.

Option 2.

If it is possible to create a new cluster while the previous read-only cluster is still running, then the following steps will enable you to keep read-availability during recovery:

1. Verify that a catastrophe has occurred, and that access to the surviving members of the cluster in the surviving data center is possible.
2. Perform an online backup of the currently running, read-only, cluster.

3. Seed a new cluster (in the current example, two new cores) using the backup from the read-only cluster, as described in Seed a cluster.
4. When the new cluster is up, load balance your workload over to the new cluster.
5. Shutdown the old, read-only, cluster.

Once your chosen recovery procedure is completed for each instance, they will form a cluster that is available for reads and writes. It recommended at this point that other cluster members are incorporated into the cluster to improve its load handling and fault tolerance. See Deploy a cluster for details of how to configure instances to join the cluster from scratch.

C.3. Advanced settings reference

This section lists the advanced settings related to running a Neo4j Causal Cluster.

Parameter	Explanation
`causal_clustering.multi_dc_license`	Enables multi-data center features. Requires appropriate licensing. **Example:** `causal_clustering.multi_dc_license=true` will enable the multi-data center features.
`causal_clustering.server_groups`	A list of group names for the server used when configuring load balancing and replication policies. **Example:** `causal_clustering.server_groups=us,us-east` will add the current instance to the groups `us` and `us-east`.
`causal_clustering.upstream_selection_strategy`	An ordered list in descending preference of the strategy which Read Replicas use to choose upstream database server from which to pull transactional updates. **Example:** `causal_clustering.upstream_selection_strategy=connect-randomly-within-server-group,typically-connect-to-random-read-replica` will configure the behavior so that the Read Replica will first try to connect to any other instance in the group(s) specified in `causal_clustering.server_groups`. Should we fail to find any live instances in those groups, then we will connect to a random Read Replica. A value of `user_defined` will enable custom strategy definitions using the setting `causal_clustering.user_defined_upstream_strategy`.
`causal_clustering.user_defined_upstream_strategy`	Defines the configuration of upstream dependencies. Can only be used if `causal_clustering.upstream_selection_strategy` is set to `user_defined`. **Example:** `causal_clustering.user_defined_upstream_strategy=groups(north2); groups(north); halt()` will look for servers in the `north2`. If none are available it will look in the `north` server group. Finally, if we cannot resolve any servers in any of the previous groups, then rule chain will be stopped via `halt()`.
`causal_clustering.load_balancing.plugin`	The load balancing plugin to use. One pre-defined plugin named `server_policies` is available by default. **Example:** `causal_clustering.load_balancing.plugin=server_policies` will enable custom policy definitions.
`causal_clustering.load_balancing.config.server_policies.<policy-name>`	Defines a custom policy under the name `<policy-name>`. Note that load balancing policies are cluster-global configurations and should be defined the exact same way on all core machines. **Example:** `causal_clustering.load_balancing.config.server_policies.north1_only=groups(north1)→min(2); halt();` will define a load balancing policy named `north1_only`. Queries are only sent to servers in the `north1` server group, provided there are two of them available. If there are less than two servers in `north1` then the chain is halted.

Appendix D: Deprecated security procedures

This appendix describes deprecated procedures for security management.

This appendix describes deprecated procedures for security management:

- Enterprise Edition
- Community Edition

The procedures described in this appendix have been deprecated and will be removed in a future release.

It is strongly recommended to migrate to the security features as described in Cypher Manual → Administration → Security

See also a worked example in Fine-grained access control.

D.1. Enterprise Edition

This section describes deprecated procedures for native user and role management for Neo4j Enterprise Edition.

A subset of this functionality is also available in Community Edition. The table below includes an indication of which functions this is valid for. Refer to Community Edition for a complete description.

In Neo4j, native user and role management are managed by using built-in procedures through Cypher. This section gives a list of all the security procedures for user management along with some simple examples. Use Neo4j Browser or Neo4j Cypher Shell to run the examples provided.

The following table lists the available procedures:

Procedure name	Description	Executable by role(s)	Available in Community Edition
`dbms.security.activateUser`	Activate a suspended user	[admin]	
`dbms.security.addRoleToUser`	Assign a role to the user	[admin]	
`dbms.security.changePassword`	Change the current user's password	[reader,editor,publisher,architect,admin]	✓
`dbms.security.changeUserPassword`	Change the given user's password	[admin]	
`dbms.security.createRole`	Create a new role	[admin]	
`dbms.security.createUser`	Create a new user	[admin]	✓
`dbms.security.deleteRole`	Delete the specified role. Any role assignments will be removed	[admin]	
`dbms.security.deleteUser`	Delete the specified user	[admin]	✓
`dbms.security.listRoles`	List all available roles	[admin]	
`dbms.security.listRolesForUser`	List all roles assigned to the specified user	[admin]	
`dbms.security.listUsers`	List all local users	[admin]	✓

Procedure name	Description	Executable by role(s)	Available in Community Edition
`dbms.security.listUsersForRole`	List all users currently assigned the specified role	[admin]	
`dbms.security.removeRoleFromUser`	Unassign a role from the user	[admin]	
`dbms.security.suspendUser`	Suspend the specified user	[admin]	
`dbms.showCurrentUser`	Show the current user	[reader,editor,publisher,architect,admin]	□
`dbms.procedures`	List roles per procedure	[reader,editor,publisher,architect,admin]	Not applicable

D.1.1. Activate a suspended user

An administrator is able to activate a suspended user so that the user is once again able to access the data in their original capacity.

Syntax:

```
CALL dbms.security.activateUser(username, requirePasswordChange)
```

Arguments:

Name	Type	Description
`username`	String	This is the username of the user to be activated.
`requirePasswordChange`	Boolean	This is optional, with a default of `true`. If this is `true`, (i) the user will be forced to change their password when they next log in, and (ii) until the user has changed their password, they will be forbidden from performing any other operation.

Exceptions:

The current user is not an administrator.
The username does not exist in the system.
The username matches that of the current user (i.e. activating the current user is not permitted).

Considerations:

This is an idempotent procedure.

Example 122. Activate a suspended user

The following example activates a user with the username '**jackgreen**'. When the user '**jackgreen**' next logs in, he will be required to change his password.

```
CALL dbms.security.activateUser('jackgreen')
```

D.1.2. Assign a role to the user

An administrator is able to assign a role to any user in the system, thus allowing the user to perform a series of actions upon the data.

Syntax:

`CALL dbms.security.addRoleToUser(roleName, username)`

Arguments:

Name	Type	Description
`roleName`	String	This is the name of the role to be assigned to the user.
`username`	String	This is the username of the user who is to be assigned the role.

Exceptions:

The current user is not an administrator.
The username does not exist in the system.
The username contains characters other than alphanumeric characters and the '_' character.
The role name does not exist in the system.
The role name contains characters other than alphanumeric characters and the '_' character.

Considerations:

This is an idempotent procedure.

Example 123. Assign a role to the user

The following example assigns the role `publisher` to the user with username '**johnsmith**'.

```
CALL dbms.security.addRoleToUser('publisher', 'johnsmith')
```

D.1.3. Change the current user's password

The procedure `dbms.security.changePassword(newPassword, requirePasswordChange)` has been entirely removed since the corresponding Cypher administration command also requires the old password, and thus is more secure. Please use `ALTER CURRENT USER SET PASSWORD FROM 'oldPassword' TO 'newPassword'`, documented in the Cypher Manual, instead.

D.1.4. Change the given user's password

An administrator is able to change the password of any user within the system. Alternatively, the current user may change their own password.

Syntax:

`CALL dbms.security.changeUserPassword(username, newPassword, requirePasswordChange)`

Arguments:

Name	Type	Description
`username`	String	This is the username of the user whose password is to be changed.
`newPassword`	String	This is the new password for the user.
`requirePasswordChange`	Boolean	This is optional, with a default of `true`. If this is `true`, (i) the user will be forced to change their password when they next log in, and (ii) until the user has changed their password, they will be forbidden from performing any other operation.

Exceptions:

The current user is not an administrator and the username does not match that of the current user.
The username does not exist in the system.
The password is the empty string.
The password is the same as the user's previous password.

Considerations:

This procedure may be invoked by the current user to change their own password, irrespective of whether or not the current user is an administrator.
This procedure may be invoked by an administrator to change another user's password.
In addition to changing the user's password, this will terminate with immediate effect all of the user's sessions and roll back any running transactions.

Example 124. Change a given user's password

The following example changes the password of the user with the username '**joebloggs**' to '**h6u4%kr**'. When the user '**joebloggs**' next logs in, he will be required to change his password.

```
CALL dbms.security.changeUserPassword('joebloggs', 'h6u4%kr')
```

D.1.5. Create a new role

An administrator is able to create custom roles in the system.

Syntax:

`CALL dbms.security.createRole(roleName)`

Arguments:

Name	Type	Description
`roleName`	String	This is the name of the role to be created.

Exceptions:

The current user is not an administrator.

The role name already exists in the system.
The role name is empty.
The role name contains characters other than alphanumeric characters and the '_' character.
The role name matches one of the native roles: `reader`, `publisher`, `architect`, and `admin`.

Example 125. Create a new role

The following example creates a new custom role.

```
CALL dbms.security.createRole('operator')
```

D.1.6. Create a new user

An administrator is able to create a new user. This action ought to be followed by assigning a role to the user, which is described here.

Syntax:

`CALL dbms.security.createUser(username, password, requirePasswordChange)`

Arguments:

Name	Type	Description
`username`	String	This is the user's username.
`password`	String	This is the user's password.
`requirePasswordChange`	Boolean	This is optional, with a default of `true`. If this is `true`, (i) the user will be forced to change their password when they log in for the first time, and (ii) until the user has changed their password, they will be forbidden from performing any other operation.

Exceptions:

The current user is not an administrator.
The username either contains characters other than the ASCII characters between `!` and `~`, or contains `:` and `,`.
The username is already in use within the system.
The password is the empty string.

Example 126. Create a new user

The following example creates a user with the username '**johnsmith**' and password '**h6u4%kr**'. When the user '**johnsmith**' logs in for the first time, he will be required to change his password.

```
CALL dbms.security.createUser('johnsmith', 'h6u4%kr')
```

D.1.7. Delete the specified role

An administrator is able to delete custom roles from the system. The native roles `reader`, `publisher`, `architect`, and `admin` cannot be deleted.

Syntax:

`CALL dbms.security.deleteRole(roleName)`

Arguments:

Name	Type	Description
`roleName`	String	This is the name of the role to be deleted.

Exceptions:

The current user is not an administrator.
The role name does not exist in the system.
The role name matches one of the native roles: `reader`, `publisher`, `architect`, and `admin`.

Considerations:

Any role assignments will be removed.

Example 127. Delete the specified role

The following example deletes the custom role '**operator**' from the system.

```
CALL dbms.security.deleteRole('operator')
```

D.1.8. Delete the specified user

An administrator is able to delete permanently a user from the system. It is not possible to undo this action, so, if in any doubt, consider suspending the user instead.

Syntax:

`CALL dbms.security.deleteUser(username)`

Arguments:

Name	Type	Description
`username`	String	This is the username of the user to be deleted.

Exceptions:

The current user is not an administrator.
The username does not exist in the system.
The username matches that of the current user (i.e. deleting the current user is not permitted).

Considerations:

It is not necessary to remove any assigned roles from the user prior to deleting the user.
Deleting a user will terminate with immediate effect all of the user’s sessions and roll back any running transactions.
As it is not possible for the current user to delete themselves, there will always be at least one administrator in the system.

Example 128. Delete the specified user

The following example deletes a user with the username '**janebrown**'.

```
CALL dbms.security.deleteUser('janebrown')
```

D.1.9. List all available roles

An administrator is able to view all assigned users for each role in the system.

Syntax:

`CALL dbms.security.listRoles()`

Returns:

Name	Type	Description
`role`	String	This is the name of the role.
`users`	List<String>	This is a list of the usernames of all users who have been assigned the role.

Exceptions:

The current user is not an administrator.

Example 129. List all available roles

The following example shows, for each role in the system, the name of the role and the usernames of all assigned users.

```
CALL dbms.security.listRoles()
```

```
+------------------------------+
| role        | users          |
+------------------------------+
| "reader"    | ["bill"]       |
| "architect" | []             |
| "admin"     | ["neo4j"]      |
| "publisher" | ["john","bob"] |
+------------------------------+
4 rows
```

D.1.10. List all roles assigned to the specified user

Any active user is able to view all of their assigned roles. An administrator is able to view all assigned roles for any user in the system.

Syntax:

`CALL dbms.security.listRolesForUser(username)`

Arguments:

Name	Type	Description
`username`	String	This is the username of the user.

Returns:

Name	Type	Description
`value`	String	This returns all roles assigned to the requested user.

Exceptions:

The current user is not an administrator and the username does not match that of the current user.
The username does not exist in the system.

Considerations:

This procedure may be invoked by the current user to view their roles, irrespective of whether or not the current user is an administrator.
This procedure may be invoked by an administrator to view the roles for another user.

Example 130. List all roles assigned to the specified user

The following example lists all the roles for the user with username '**johnsmith**', who has the roles `reader` and `publisher`.

```
CALL dbms.security.listRolesForUser('johnsmith')
```

```
+-------------+
| value       |
+-------------+
| "reader"    |
| "publisher" |
+-------------+
2 rows
```

D.1.11. List all local users

An administrator is able to view the details of every user in the system.

Syntax:

`CALL dbms.security.listUsers()`

Returns:

Name	Type	Description
`username`	String	This is the user’s username.
`roles`	List<String>	This is a list of roles assigned to the user.
`flags`	List<String>	This is a series of flags indicating whether the user is suspended or needs to change their password.

Exceptions:

The current user is not an administrator.

Example 131. List all local users

The following example shows, for each user in the system, the username, the roles assigned to the user, and whether the user is suspended or needs to change their password.

```
CALL dbms.security.listUsers()
```

```
+----------------------------------------------------------------------+
| username | roles                      | flags                        |
+----------------------------------------------------------------------+
| "neo4j"  | ["admin"]                  | []                           |
| "anne"   | []                         | ["password_change_required"] |
| "bill"   | ["reader"]                 | ["is_suspended"]             |
| "john"   | ["architect","publisher"] | []                           |
+----------------------------------------------------------------------+
4 rows
```

D.1.12. List all users currently assigned the specified role

An administrator is able to view all assigned users for a role.

Syntax:

`CALL dbms.security.listUsersForRole(roleName)`

Arguments:

Name	Type	Description
`roleName`	String	This is the name of the role.

Returns:

Name	Type	Description
`value`	String	This returns all assigned users for the requested role.

Exceptions:

The current user is not an administrator.
The role name does not exist in the system.

Example 132. List all users currently assigned the specified role

The following example lists all the assigned users - '**bill**' and '**anne**' - for the role `publisher`.

```
CALL dbms.security.listUsersForRole('publisher')
```

```
+--------+
| value  |
+--------+
| "bill" |
| "anne" |
+--------+
2 rows
```

D.1.13. Unassign a role from the user

An administrator is able to remove a role from any user in the system, thus preventing the user from performing upon the data any actions prescribed by the role.

Syntax:

`CALL dbms.security.removeRoleFromUser(roleName, username)`

Arguments:

Name	Type	Description
`roleName`	String	This is the name of the role which is to be removed from the user.
`username`	String	This is the username of the user from which the role is to be removed.

Exceptions:

The current user is not an administrator.
The username does not exist in the system.
The role name does not exist in the system.
The username is that of the current user and the role is `admin`.

Considerations:

If the username is that of the current user and the role name provided is `admin`, an error will be thrown; i.e. the current user may not be demoted from being an administrator.
As it is not possible for the current user to remove the `admin` role from themselves, there will always be at least one administrator in the system.
This is an idempotent procedure.

Example 133. Unassign a role from the user

The following example removes the role `publisher` from the user with username '**johnsmith**'.

```
CALL dbms.security.removeRoleFromUser('publisher', 'johnsmith')
```

D.1.14. Suspend the specified user

An administrator is able to suspend a user from the system. The suspended user may be activated at a later stage.

Syntax:

```
CALL dbms.security.suspendUser(username)
```

Arguments:

Name	Type	Description
`username`	String	This is the username of the user to be suspended.

Exceptions:

The current user is not an administrator.
The username does not exist in the system.
The username matches that of the current user (i.e. suspending the current user is not permitted).

Considerations:

Suspending a user will terminate with immediate effect all of the user’s sessions and roll back any running transactions.
All of the suspended user’s attributes — assigned roles and password — will remain intact.
A suspended user will not be able to log on to the system.
As it is not possible for the current user to suspend themselves, there will always be at least one active administrator in the system.
This is an idempotent procedure.

Example 134. Suspend the specified user

The following example suspends a user with the username '**billjones**'.

```
CALL dbms.security.suspendUser('billjones')
```

D.1.15. Show the current user

The current user is able to view whether or not they need to change their password.

Syntax:

```
CALL dbms.showCurrentUser()
```

Returns:

Name	Type	Description
`username`	String	This is the user’s username.
`flags`	List<String>	This is a flag indicating whether the user needs change their password.

Example 135. Show the current user

The following example shows that the current user — with the username '**johnsmith**' — does not need to change his password.

```
CALL dbms.showCurrentUser()
```

```
+--------------------------------------------------+
| username    | roles                      | flags |
+--------------------------------------------------+
| "johnsmith" | ["architect","publisher"] | []    |
+--------------------------------------------------+
1 row
```

D.1.16. List roles per procedure

Any active user is able to view all procedures in the system, including which role(s) have the privilege to execute them. Note that the `defaultBuiltInRoles` column does not include roles created by administration commands, unless they are included in the procedure security settings.

Syntax:

`CALL dbms.procedures()`

Returns:

Name	Type	Description
`name`	String	This is the name of the procedure.
`signature`	String	This is the signature of the procedure.
`description`	String	This is a description of the procedure.
`defaultBuiltInRoles`	List<String>	This is a list of roles having the privilege to execute the procedure.

Example 136. List roles per procedure

The following example shows, for four of the security procedures, the procedure name, the description, and which roles have the privilege to execute the procedure.

```
CALL dbms.procedures()
YIELD name, signature, description, roles
WITH name, description, roles
WHERE name contains 'security'
RETURN name, description, roles
ORDER BY name
LIMIT 4
```

```
+-----------------------------------------------------------------------------------------------------
----------+
|name                              |description                          |roles
|
+-----------------------------------------------------------------------------------------------------
----------+
|"dbms.security.activateUser"      |"Activate a suspended user."         | ["admin"]
|
|"dbms.security.addRoleToUser"     |"Assign a role to the user."         | ["admin"]
|
|"dbms.security.changePassword"    |"Change the current user's password."|
["reader","editor","publisher", ... |
|"dbms.security.changeUserPassword"|"Change the given user's password."  | ["admin"]
|
+-----------------------------------------------------------------------------------------------------
----------+
4 rows
```

D.2. Community Edition

This section describes deprecated procedures for user and password management for Neo4j Community Edition.

User and password management for Community Edition is a subset of the functionality available in Enterprise Edition. The following is true for user management in Community Edition:

- It is possible to create multiple users.
- All users assume the privileges of an `admin` for the available functionality.

Users are managed by using built-in procedures through Cypher. This section gives a list of all the security procedures for user management along with some simple examples. Use Neo4j Browser or Neo4j Cypher Shell to run the examples provided. Unless stated otherwise, all arguments to the procedures described in this section must be supplied.

Name	Description
dbms.security.changePassword	Change the current user’s password
dbms.security.createUser	Add a user
dbms.security.deleteUser	Delete a user
dbms.security.listUsers	List all users
dbms.showCurrentUser	Show details for the current user

D.2.1. Change the current user's password

The procedure `dbms.security.changePassword(newPassword, requirePasswordChange)` has been entirely removed since the corresponding Cypher administration command also requires the old password, and thus is more secure. Please use `ALTER CURRENT USER SET PASSWORD FROM 'oldPassword' TO 'newPassword'`, documented in the Cypher Manual, instead.

D.2.2. Add a user

The current user is able to add a user to the system.

Syntax:

`CALL dbms.security.createUser(username, password, requirePasswordChange)`

Arguments:

Name	Type	Description
`username`	String	This is the user's username.
`password`	String	This is the user's password.
`requirePasswordChange`	Boolean	This is optional, with a default of `true`. If this is `true`, (i) the user will be forced to change their password when they log in for the first time, and (ii) until the user has changed their password, they will be forbidden from performing any other operation.

Exceptions:

The username either contains characters other than the ASCII characters between `!` and `~`, or contains `:` and `,`.
The username is already in use within the system.
The password is the empty string.

Example 137. Add a user

The following example creates a user with the username '**johnsmith**' and password '**h6u4%kr**'. When the user '**johnsmith**' logs in for the first time, he will be required to change his password.

```
CALL dbms.security.createUser('johnsmith', 'h6u4%kr', true)
```

D.2.3. Delete a user

The current user is able to delete permanently a user from the system.

Syntax:

`CALL dbms.security.deleteUser(username)`

Arguments:

Name	Type	Description
`username`	String	This is the username of the user to be deleted.

Exceptions:

The username does not exist in the system.
The username matches that of the current user (i.e. deleting the current user is not permitted).

Considerations:

Deleting a user will terminate with immediate effect all of the user's sessions and roll back any running transactions.
As it is not possible for the current user to delete themselves, there will always be at least one user in the system.

Example 138. Delete a user

The following example deletes a user with the username '**janebrown**'.

```
CALL dbms.security.deleteUser('janebrown')
```

D.2.4. List all native users

The current user is able to view the details of every user in the system.

Syntax:

`CALL dbms.security.listUsers()`

Returns:

Name	Type	Description
`username`	String	This is the user's username.
`flags`	List<String>	This is a flag indicating whether the user needs to change their password.

Example 139. List all users

The following example shows the username for each user in the system, and whether the user needs to change their password.

```
CALL dbms.security.listUsers()
```

```
+-------------------------------------------+
| username | flags                          |
+-------------------------------------------+
| "neo4j"  | []                             |
| "anne"   | ["password_change_required"]   |
| "bill"   | []                             |
+-------------------------------------------+
3 rows
```

D.2.5. Show details for the current user

The current user is able to view whether or not they need to change their password.

Syntax:

```
CALL dbms.showCurrentUser()
```

Returns:

Name	Type	Description
`username`	String	This is the user’s username.
`flags`	List<String>	This is a flag indicating whether the user needs change their password.

Example 140. Show details for the current user

The following example shows that the current user — with the username '**johnsmith**' — does not need to change his password.

```
CALL dbms.showCurrentUser()
```

```
+---------------------+
| username    | flags |
+---------------------+
| "johnsmith" | []    |
+---------------------+
1 row
```

License

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)

You are free to

Share

copy and redistribute the material in any medium or format

Adapt

remix, transform, and build upon the material

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms

Attribution

You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial

You may not use the material for commercial purposes.

ShareAlike

If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions

You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

See https://creativecommons.org/licenses/by-nc-sa/4.0/ for further details. The full license text is available at https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.