



Table of Contents

5.2. Create a new cluster

Enterprise Edition

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
- Remove a Core from a cluster
- Elections and leadership
- Initial discovery of cluster members
 - Explicitly listing discovery members
 - Initial discovery of cluster members with DNS
 - Initial discovery of cluster members with Kubernetes
- Store copy

5.2.1. Introduction

In this section we describe how to set up a new Causal Cluster consisting of three Core instances. We then proceed to add more Core Servers as well as Read Replicas. Using this basic pattern, we will be able to create any sized cluster.



Three Cores is the minimum number of servers needed in order to form a fault-tolerant Causal Cluster. See [Section 5.1.2.1, “Core Servers”](#) ([./../reference/introduction/#causal-clustering-core-servers](#)) for a discussion on the number of servers required in various scenarios.

Refer to Section B.1, “Set up a local Causal Cluster” ([./../tutorial/local-causal-cluster/](#)) for a tutorial on how to set up a Causal Cluster on a local machine.

5.2.2. Configure a Core-only cluster

The following configuration settings are important to consider for basic cluster operation of a new Causal Cluster. See also Section 5.5, “Settings reference” ([./../settings/](#)) for more detailed descriptions and examples.

Table 5.1. Important settings for a new Causal Cluster

Option name	Description
<code>dbms.connectors.default_listen_address</code> (./../reference/configuration-settings/#config_dbms.connectors.default_listen_address)	The address or network interface this machine uses to listen for incoming messages. Setting this value to <code>0.0.0.0</code> allows Neo4j to bind to any available network interface.
<code>dbms.connectors.default_advertised_address</code> (./../reference/configuration-settings/#config_dbms.connectors.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.
<code>dbms.mode</code> (./../reference/configuration-settings/#config_dbms.mode)	The operating mode of a single database instance. For Causal Clustering, there are two possible modes: <code>CORE</code> or <code>READ_REPLICA</code> .
<code>causal_clustering.minimum_core_cluster_size_at_formation</code> (./../reference/configuration-settings/#config_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.
<code>causal_clustering.minimum_core_cluster_size_at_runtime</code> (./../reference/configuration-settings/#config_causal_clustering.minimum_core_cluster_size_at_runtime)	The minimum number of Core instances which will exist in the consensus group.

Option name neo4j (.../...) Search Neo4j docs...	Description
<code>causal_clustering.initial_discovery_members</code> (.../.../reference/configuration-settings/#config_causal_clustering.initial_discovery_members)	<p>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 <code>:5000</code>. It is good practice to set this parameter to the same value on all Core Servers.</p> <p>The behavior of this setting can be modified by configuring the setting <code>causal_clustering.discovery_type</code>. This is described in detail in Section 5.2.7.2, “Initial discovery of cluster members with DNS”.</p>

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

Example 5.1. 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 (.../.../configuration/file-locations/)` on each server. Note that they are all identical, except for the configuration of `dbms.connectors.default_advertised_address`:

***neo4j.conf* on core01.example.com:**

```
dbms.connectors.default_listen_address=0.0.0.0
dbms.connectors.default_advertised_address=core01.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.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.example.com:5000
```

***neo4j.conf* on core02.example.com:**

```
dbms.connectors.default_listen_address=0.0.0.0
dbms.connectors.default_advertised_address=core02.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.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.example.com:5000
```

***neo4j.conf* on core03.example.com:**



```
dbms.connectors.default_listen_address=0.0.0.0
dbms.connectors.default_advertised_address=core03.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.initial_discovery_members=core01.example.com:5000,core02.example.com:5000,core03.example.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* (`././configuration/file-locations/`).

5.2.3. Add a Core Server to an existing cluster

Core Servers are added to an existing cluster by starting a new database server with the appropriate configuration. The new server will integrate itself with the existing cluster, and will 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.

If the new server is intended to be a permanent member of the cluster, it is good practice to update `causal_clustering.discovery_members` on all the servers in the cluster to include the new server.

Example 5.2. 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 Example 5.1, “Configure a Core-only cluster”.



We configure the following entries in *neo4j.conf* (*../configuration/file-locations/*):

neo4j.conf on **core04.example.com**:

```
dbms.connectors.default_listen_address=0.0.0.0
dbms.connectors.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.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.

5.2.4. 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 5.3. 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 Example 5.1, “Configure a Core-only cluster”.

We configure the following entries in *neo4j.conf* (*../configuration/file-locations/*):

neo4j.conf on **replica01.example.com**:

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

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



5.2.5. Remove a Core from a cluster

Search Neo4j docs...

A Core Server can be downgraded to a standalone instance, using the `neo4j-admin unbind (../../tools/unbind/)` command.

Once a server has been unbound from a cluster, the store files are equivalent to a Neo4j standalone instance. From this point those files could be used to run a standalone instance by restarting it in `SINGLE` mode.



The on-disk state of Core Server instances is different to that of standalone server instances. It is important to understand that once an instance unbinds from a cluster, it cannot be re-integrated with that cluster. This is because both the cluster and the single instance are now separate databases with different and irreconcilable writes having been applied to them. Technically the cluster will have written entries to its Raft log, whilst the standalone instance will have written directly to the transaction log (since there is no Raft log in a standalone instance).

If we try to reinsert the standalone instance 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.

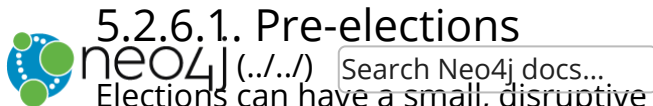
5.2.6. Elections and leadership

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* that 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 had no contact from a leader, then it can initiate an election. A new leader will be elected from the Cores, and the old leader will stand down.

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.



5.2.6.1. Pre-elections

Elections can have a small, disruptive effect. To minimise unnecessary elections, Neo4j supports pre-elections.

Before initiating an election, a Core will ask its peers whether it could potentially receive votes from them. If it does not receive a positive response from a quorum of the cluster, then it does not proceed. A positive response is only given if the Core would make a suitable leader, and the responder is also asking whether it could receive votes.

This mechanism can prevent unnecessary elections in the following ways:

- A Core whose log is behind will not initiate an election. This can happen when a network partition heals; during the partition, log entries do not reach a Core. When it heals, the Core will attempt to start an election as it has not heard from the leader, but it will lose because its log is not up to date.
- If a quorum of Cores can communicate with the leader, then there will be no election. In this case there would be no quorum to start an election for any Cores that cannot.

The setting `causal_clustering.enable_pre_voting (../../reference/configuration-settings/#config_causal_clustering.enable_pre_voting)` controls whether pre-elections are enabled. Changing this requires a complete restart of the cluster.

5.2.6.2. Bias cluster leadership with follower-only instances

In some situations, operators might want to actively prevent some instances from taking on the leader role. For example in a multi-data center scenario, we might want to ensure that the leader remains in a favored geographic location for performance or governance reasons. In Neo4j Causal Clustering we can configure instances to *refuse to become leader*, which is equivalent to always remaining a follower. This is done by configuring the setting `causal_clustering.refuse_to_be_leader (../../reference/configuration-settings/#config_causal_clustering.refuse_to_be_leader)`. It is not generally advisable to use this option, since the first priority in the cluster is to maximize availability. The highest availability stems from having any healthy instance take leadership of the cluster on pathological failure.



Despite superficial similarities, a non-leader capable Core instance is not the same as a Read Replica. Read Replicas do not participate in transaction processing, nor are they permitted to be involved in cluster topology management.

Conversely, a follower-only Core instance will still process transactions and cluster membership requests as per Raft to ensure consistency and safety.

5.2.7. Initial discovery of cluster members

In order to connect to a cluster, a Core Server or a Read Replica needs to know the addresses of some of the Core Servers which it can bind to, in order to run the discovery protocol and get the full information about the Core 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 behavior of the initial discovery is determined by the parameters `causal_clustering.discovery_type` (`../../reference/configuration-settings/#config_causal_clustering.discovery_type`) and `causal_clustering.initial_discovery_members` (`../../reference/configuration-settings/#config_causal_clustering.initial_discovery_members`), and is described in the following sections.

5.2.7.1. Explicitly listing discovery members



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 Example 5.1, “Configure a Core-only cluster”.

5.2.7.2. Initial discovery of cluster members with DNS

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.

5.2.7.3. Initial discovery of cluster members for Kubernetes

 A special case is when a Causal Cluster is running in Kubernetes (<https://kubernetes.io/>) and each Core Server is running as a Kubernetes service. Then the addresses of Core Cluster members can be obtained using the List Service (<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#list-service-v1-core>) API.

The following settings are used to configure for this scenario:

- Set `causal_clustering.discovery_type=K8S`.
- Set `causal_clustering.kubernetes.label_selector` (`../../reference/configuration-settings/#config_causal_clustering.kubernetes.label_selector`) to a label selector (<https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors>) for the Causal Cluster services.
- Set `causal_clustering.kubernetes.service_port_name` (`../../reference/configuration-settings/#config_causal_clustering.kubernetes.service_port_name`) to the name of the service port (<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#serviceport-v1-core>) 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.


Please note that:



- 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 (<https://kubernetes.io/docs/reference/access-authn-authz/rbac/>) or ABAC authorization (<https://kubernetes.io/docs/reference/access-authn-authz/abac/>).
- 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.

5.2.8. Store copy

 Causal Clustering uses a robust and configurable store copy protocol. When a store copy is started it will first send a prepare request to the specified instance. If the prepare request is successful the client will send file and index requests, one request per file or index, to provided upstream members in the cluster. The retry logic per request can be modified through `causal_clustering.store_copy_max_retry_time_per_request (../../reference/configuration-settings/#config_causal_clustering.store_copy_max_retry_time_per_request)`. If a request fails and that maximum retry time is met it will stop retrying and the store copy will fail.

Use `causal_clustering.catch_up_client_inactivity_timeout (../../reference/configuration-settings/#config_causal_clustering.catch_up_client_inactivity_timeout)` to configure the inactivity timeout for any catchup request. Bear in mind that this setting is for all requests from the catchup client, including pulling of transactions.

There are three scenarios that will start a store copy. The upstream selection strategy is different for each scenario.

Backup

Upstream strategy is set to a fixed member by the `neo4j-admin backup` command. All requests will go to the specified member.

Seeding a new member with empty store

Will use configured upstream strategy for that instance.

Instance falling too far behind

Will use configured upstream strategy for that instance.

The upstream strategy differs for Cores and Read Replicas. Cores will always send the prepare request to the leader to get the most up-to-date information of the store. The strategy for the file and index requests for Cores is to vary every other request to random Read Replica and every other to random Core member. Read Replicas' strategy is the same as for pulling transactions. The default is to pull from a random Core member.

If you are running a multi-data-center cluster, both Cores and Read Replicas upstream strategy can be configured. Remember that for Read Replicas this also effect from whom transactions are pulled. See more in Configure for multi-data center operations ([../../clustering-advanced/multi-data-center/configuration/](https://neo4j.com/docs/operations-manual/current/clustering-advanced/multi-data-center/configuration/))