# InfluxData | Documentation | Cluster Setup

https://docs.influxdata.com/influxdb/v0.10/clustering/cluster\_setup/

**Warning!** This page documents an old version of InfluxDB, which is no longer actively developed. [**InfluxDB v1.2**](https://docs.influxdata.com/influxdb/v1.2/clustering/cluster_setup/) is the most recent stable version of InfluxDB.

***Note:****Clustering is now a commerial product called InfluxEnterprise. More information can be found*[***here***](https://portal.influxdata.com/)*.*

This guide briefly introduces the InfluxDB cluster model and provides step-by-step instructions for setting up a cluster.

[InfluxDB cluster model](https://docs.influxdata.com/influxdb/v0.10/clustering/cluster_setup/#influxdb-cluster-model)

InfluxDB supports arbitrarily sized clusters and any [**replication factor**](https://docs.influxdata.com/influxdb/v0.10/concepts/glossary/#replication-factor) from 1 to the number of nodes in the cluster.

There are three types of nodes in an InfluxDB cluster: [**consensus nodes**](https://docs.influxdata.com/influxdb/v0.10/concepts/glossary/#consensus-node), [**data nodes**](https://docs.influxdata.com/influxdb/v0.10/concepts/glossary/#data-node), and [**hybrid nodes**](https://docs.influxdata.com/influxdb/v0.10/concepts/glossary/#hybrid-node). A cluster must have an odd number of nodes running the [**consensus service**](https://docs.influxdata.com/influxdb/v0.10/concepts/glossary/#consensus-service) to form a Raft consensus group and remain in a healthy state.

Hardware requirements vary for the different node types. See [**Hardware Sizing**](https://docs.influxdata.com/influxdb/v0.10/guides/hardware_sizing/#general-hardware-guidelines-for-clusters) for cluster hardware requirements.

[Cluster setup](https://docs.influxdata.com/influxdb/v0.10/clustering/cluster_setup/#cluster-setup)

The following steps configure and start up an InfluxDB cluster with three [**hybrid nodes**](https://docs.influxdata.com/influxdb/v0.10/concepts/glossary/#hybrid-node). If you’re interested in having any of the different node types, see [**Cluster Node Configuration**](https://docs.influxdata.com/influxdb/v0.10/clustering/cluster_node_config/) for their configuration details. Note that your first three nodes must be either hybrid nodes or consensus nodes.

We assume that you are running some version of Linux, and, while it is possible to build a cluster on a single server, it is not recommended.

***Note:****Always use the*[***most recent release***](https://influxdata.com/downloads/#influxdb)*for clustering as there are significant improvements with each release.*

**1**   [**Install**](https://docs.influxdata.com/influxdb/v0.10/introduction/installation/) InfluxDB on three machines. Do not start the daemon on any of the machines.

**2**  Configure the three nodes.

Where IP is the node’s IP address or hostname, each node’s /etc/influxdb/influxdb.conf file should have the following settings:

[meta]

enabled = true

...

bind-address = "<IP>:8088"

http-bind-address = "<IP>:8091"

...

[data]

enabled = true

[http]

...

bind-address = "<IP>:8086"

* Setting [meta] enabled = true and [data] enabled = true makes the node a hybrid node.
* The [meta] bind-address is the address for cluster wide communication.
* The [meta] http-bind-address is the address for meta node communication.
* The [http] bind-address is the address for the HTTP API.

***NOTE:****The hostnames for each machine must be resolvable by all members of the cluster.*

**3**  Start InfluxDB on the first node:

sudo service influxdb start

**4**  Point the second and third nodes to the first node.

On the second and third nodes, set INFLUXD\_OPTS in /etc/default/influxdb:

INFLUXD\_OPTS="-join <IP1>:8091"

where IP1 is the first node’s IP address or hostname.

If the /etc/default/influxdb file does not exist, create it.

**5**  Start InfluxDB on the second and third nodes:

sudo service influxdb start

**6**  Verify that the cluster is healthy.

Issue a SHOW SERVERS query to each node in your cluster using the [**influx CLI**](https://docs.influxdata.com/influxdb/v0.10/tools/shell/). The output should show that your cluster is made up of three hybrid nodes (hybrid nodes appear as both data\_nodes and meta\_nodes in theSHOW SERVERS query results):

> SHOW SERVERS

name: data\_nodes

----------------

id http\_addr tcp\_addr

1 IP1:8086 IP1:8088

3 IP2:8086 IP2:8088

5 IP3:8086 IP3:8088

name: meta\_nodes

----------------

id http\_addr tcp\_addr

1 IP1:8091 IP1:8088

2 IP2:8091 IP2:8088

4 IP3:8091 IP3:8088

***Notes:***

* *Currently, the SHOW SERVERS query groups results into data\_nodes and meta\_nodes. The term meta\_nodes is outdated and refers to a node that runs the consensus service.*
* *The irregular node id numbers in the SHOW SERVERS results is a known issue and a fix is underway. For now, it may be easier to identify data nodes and consensus nodes by the IP addresses reported in the SHOW SERVERS results.*

And that’s your three node cluster!

If you believe that you did the above steps correctly, but are still experiencing problems, try restarting each node in your cluster.

[**Adding nodes to your cluster**](https://docs.influxdata.com/influxdb/v0.10/clustering/cluster_setup/#adding-nodes-to-your-cluster)

Once your initial cluster is healthy and running appropriately, you can start adding nodes to the cluster. Additional nodes can be consensus nodes, data nodes, or hybrid nodes. See [**Cluster Node Configuration**](https://docs.influxdata.com/influxdb/v0.10/clustering/cluster_node_config/) for how to configure the different node types.

Adding a node to your cluster follows the same procedure that we outlined above. Note that in step 4, when you point your new node to the cluster, you must set INFLUXD\_OPTS to the hostname:port pair of a pre-existing cluster member that is running the [**consensus service**](https://docs.influxdata.com/influxdb/v0.10/concepts/glossary/#consensus-service). If you specify more than one hostname:port pair in a comma delimited list, Influx will try to connect with the additional pairs if it cannot connect with the first.

[**Removing nodes from your cluster**](https://docs.influxdata.com/influxdb/v0.10/clustering/cluster_setup/#removing-nodes-from-your-cluster)

Please see the [**reference documentation on DROP SERVER**](https://docs.influxdata.com/influxdb/v0.10/query_language/spec/#drop-server).

# pygmalios/influxdb-cluster:

Docker definiton for configurable InfluxDB node building

https://github.com/pygmalios/influxdb-cluster

InfluxDB Cluster Setup

A simplistic approach to configuring and starting InfluxDB cluster nodes.

The configuration of InfluxDB on startup is determined by two key environmental variables, INFLUXD\_CONFIG & INFLUXD\_OPTS, and the CMD passed into the Docker invocation.

The variable INFLUXD\_CONFIG represents the path to the configuration file that influxd uses to bring up the node.

Additional startup options can be stored in the INFLUXD\_OPTS variable (this is optional), or by passing them into the Docker CMD invocation.

Influxd Configuration

The default behavior of the node is to create a new configuration file by executing the influxd config command at startup and piping the contents to /etc/influxdb/influxdb.conf. Altering the value of INFLUXD\_CONFIG will change the location of this generated file.

Values in the generated file can be patched/overridden through ENV variables or by mounting your own configuration.

Patching/Overriding defaults with a partial config

As of InfluxDB 0.10.0 the influxd config command accepts a -config option to submit a partial config that will overwrite the default generation. Mounting a custom partial config can be used to patch defaults without writing an entire config file.

Consider the following partial custom config:

[meta]

dir = "/mnt/db/meta"

[data]

dir = "/mnt/db/data"

wal-dir = "/mnt/influx/wal"

[hinted-handoff]

dir = "/mnt/db/hh"

Mounting this file to /root/influxdb.conf.patch when creating/starting the container will patch the default config with the values provided. Partial custom configurations can be mounted elsewhere but the value of the ENV variable INFLUXD\_PATCH must be changed in addition to reflect the non-standard location of the custom partial file.

Patching/Overriding Defaults with ENV

If it is the case that most of the default configuration is acceptable, values can be patched piecemeal by defining ENV variables using the naming convention INFLUX\_\_\_<section>\_\_\_<option>=<value>. In many cases, passing ENV variables is easier than mounting custom configs as well. Passing ENV variables in this manner overrides custom partial files as described above.

The variable must start with the string "INFLUX", followed by three underscores (\_\_\_), the name of the configuration section, three more underscores (\_\_\_), and the name of the option.

If the section or option name contains an underscore (\_), replace it in the ENV name with two underscores (\_\_). Replace dashes (-) with a single underscore (\_).

Take the following configuration section:

[continuous\_queries]

...

compute-no-more-than = "2m0s"

Override compute-no-more-than by setting the ENV variable:

INFLUX\_\_\_CONTINUOUS\_\_QUERIES\_\_\_COMPUTE\_NO\_MORE\_THAN="5m0s"

Which yields:

[continuous\_queries]

...

compute-no-more-than = "5m0s"

**Suggestion:** Store your patched options in an Env file to make container invocation simpler.

Mounting A Custom Configuration

Instead of patching individual options, an entire configuration can be mounted into the container. Ensure that the location of the mounted config is reflected in the INFLUXD\_CONFIG variable:

docker run --rm --interactive --tty \

--env INFLUXD\_CONFIG=/influxdb/influxdb.conf \

--volume $(pwd)/example:/influxdb \

amancevice/influxdb-cluster

Clustering

It would be a good idea to review the instructions on InfluxDB's documentation on [clustering](https://docs.influxdata.com/influxdb/v0.10/guides/clustering/#configuration) before continuing.

Example Cluster Setup

Assume we have set up three EC2 instances on AWS using [InfluxDB's installation guide](https://docs.influxdata.com/influxdb/v0.10/introduction/installation/#hosting-on-aws). Having followed the instructions, assume two EBS volumes have been mounted at /mnt/influx and /mnt/db. These volumes are to be mounted to the container at the same location as the host.

Assume that the addressable hostnames for each of the three nodes are as follows:

* ix0.mycluster
* ix1.mycluster
* ix2.mycluster

Patch the configuration

Create an Envfile for each node in the cluster that makes the [recommended patches](https://docs.influxdata.com/influxdb/v0.10/introduction/installation/#configuring-the-instance). See the example at [./example/Envfile](https://github.com/pygmalios/influxdb-cluster/blob/0.10.0/example/Envfile):

INFLUX\_\_\_META\_\_\_DIR="/mnt/db/meta"

INFLUX\_\_\_DATA\_\_\_DIR="/mnt/db/data"

INFLUX\_\_\_DATA\_\_\_WAL\_DIR="/mnt/influx/wal"

INFLUX\_\_\_HINTED\_HANDOFF\_\_\_DIR="/mnt/db/hh"

As of InfluxDB 0.10.0, you must patch the [meta] section's values for bind-address and http-bind-address as well as the[http] bind-address option. We will use the values <hostname>:8088, <hostname>:8091, and <hostname>:8086, respectively.

**NOTE** the hostnames used must be accessible from the other nodes in the cluster. Additionally, we **must** assign the true hostname to the container using the --hostname option of docker run/docker create.

Bring up the first node

Update the Envfile with the patched bind addresses or pass them in directly:

docker run --detach --name ix0 \

--env INFLUX\_\_\_META\_\_\_BIND\_ADDRESS='"ix0.mycluster:8088"' \

--env INFLUX\_\_\_META\_\_\_HTTP\_BIND\_ADDRESS='"ix0.mycluster:8091"' \

--env INFLUX\_\_\_HTTP\_\_\_BIND\_ADDRESS='"ix0.mycluster:8086"' \

--env-file ./Envfile \

--hostname ix0.mycluster \

--publish 8083:8083 \

--publish 8086:8086 \

--publish 8088:8088 \

--publish 8091:8091 \

--volume /mnt/db:/mnt/db \

--volume /mnt/influx:/mnt/influx \

amancevice/influxdb-cluster

Bring up the second node

The second follower node is started almost identically to the first node, altering the CMD to join to the leader on port 8091:

docker run --detach --name ix1 \

--env INFLUX\_\_\_META\_\_\_BIND\_ADDRESS='"ix1.mycluster:8088"' \

--env INFLUX\_\_\_META\_\_\_HTTP\_BIND\_ADDRESS='"ix1.mycluster:8091"' \

--env INFLUX\_\_\_HTTP\_\_\_BIND\_ADDRESS='"ix1.mycluster:8086"' \

--env-file ./Envfile \

--hostname ix1.mycluster \

--publish 8083:8083 \

--publish 8086:8086 \

--publish 8088:8088 \

--publish 8091:8091 \

--volume /mnt/db:/mnt/db \

--volume /mnt/influx:/mnt/influx \

amancevice/influxdb-cluster -join ix0.mycluster:8091

Bring up the third node

Bring up the third follower node following this pattern:

docker run --detach --name ix2 \

--env INFLUX\_\_\_META\_\_\_BIND\_ADDRESS='"ix2.mycluster:8088"' \

--env INFLUX\_\_\_META\_\_\_HTTP\_BIND\_ADDRESS='"ix2.mycluster:8091"' \

--env INFLUX\_\_\_HTTP\_\_\_BIND\_ADDRESS='"ix2.mycluster:8086"' \

--env-file ./Envfile \

--hostname ix2.mycluster \

--publish 8083:8083 \

--publish 8086:8086 \

--publish 8088:8088 \

--publish 8091:8091 \

--volume /mnt/db:/mnt/db \

--volume /mnt/influx:/mnt/influx \

amancevice/influxdb-cluster -join ix0.mycluster:8091

And so on...

See the example at [./example/cluster.sh](https://github.com/pygmalios/influxdb-cluster/blob/0.10.0/example/cluster.sh) to see how to bring up a simple cluster on your machine.

# amancevice/influxdb-cluster - Docker Hub

https://hub.docker.com/r/amancevice/influxdb-cluster/

**InfluxDB Cluster Setup**

InfluxDB version:*0.11.1*

**NOTE** InfluxDB no longer supports clustering as of version 0.12.0. As such this repository is effectively deprecated. You can still use versions 0.9.6 through 0.11.1, but I'm no longer maintaining this codebase. See [amancevice/influxdb](https://hub.docker.com/r/amancevice/influxdb-cluster/) to use versions of InfluxDB past 0.12.0 the fork of this repository.

A simplistic approach to configuring and starting InfluxDB cluster nodes.

The configuration of InfluxDB on startup is determined by two key environmental variables, INFLUXD\_CONFIG &INFLUXD\_OPTS, and the CMD passed into the Docker invocation.

The variable INFLUXD\_CONFIG represents the path to the configuration file that influxd uses to bring up the node.

Additional startup options can be stored in the INFLUXD\_OPTS variable (this is optional), or by passing them into the Docker CMD invocation.

**Influxd Configuration**

The default behavior of the node is to create a new configuration file by executing the influxd configcommand at startup and piping the contents to /etc/influxdb/influxdb.conf. Altering the value ofINFLUXD\_CONFIG will change the location of this generated file.

Values in the generated file can be patched/overridden through ENV variables or by mounting your own configuration.

**Patching/Overriding defaults with a partial config**

As of InfluxDB 0.10.0 the influxd config command accepts a -config option to submit a partial config that will overwrite the default generation. Mounting a custom partial config can be used to patch defaults without writing an entire config file.

Consider the following partial custom config:

[meta]

dir = "/mnt/db/meta"

[data]

dir = "/mnt/db/data"

wal-dir = "/mnt/influx/wal"

[hinted-handoff]

dir = "/mnt/db/hh"

Mounting this file to /root/influxdb.conf.patch when creating/starting the container will patch the default config with the values provided. Partial custom configurations can be mounted elsewhere but the value of theENV variable INFLUXD\_PATCH must be changed in addition to reflect the non-standard location of the custom partial file.

**Patching/Overriding Defaults with ENV**

If it is the case that most of the default configuration is acceptable, values can be patched piecemeal by definingENV variables using the naming convention INFLUX\_\_\_<section>\_\_\_<option>=<value>. In many cases, passing ENV variables is easier than mounting custom configs as well. Passing ENV variables in this manner overrides custom partial files as described above.

The variable must start with the string "INFLUX", followed by three underscores (\_\_\_), the name of the configuration section, three more underscores (\_\_\_), and the name of the option.

If the section or option name contains an underscore (\_), replace it in the ENV name with two underscores (\_\_). Replace dashes (-) with a single underscore (\_).

Take the following configuration section:

[continuous\_queries]

...

compute-no-more-than = "2m0s"

Override compute-no-more-than by setting the ENV variable:

INFLUX\_\_\_CONTINUOUS\_\_QUERIES\_\_\_COMPUTE\_NO\_MORE\_THAN="5m0s"

Which yields:

[continuous\_queries]

...

compute-no-more-than = "5m0s"

**Suggestion:** Store your patched options in an Envfile to make container invocation simpler.

**Mounting A Custom Configuration**

Instead of patching individual options, an entire configuration can be mounted into the container. Ensure that the location of the mounted config is reflected in the INFLUXD\_CONFIG variable:

docker run --rm --interactive --tty \

--env INFLUXD\_CONFIG=/influxdb/influxdb.conf \

--volume $(pwd)/example:/influxdb \

amancevice/influxdb-cluster

**Clustering**

It would be a good idea to review the instructions on InfluxDB's documentation on [clustering](https://docs.influxdata.com/influxdb/v0.10/guides/clustering/#configuration) before continuing.

**Example Cluster Setup**

Assume we have set up three EC2 instances on AWS using [InfluxDB's installation guide](https://docs.influxdata.com/influxdb/v0.10/introduction/installation/#hosting-on-aws). Having followed the instructions, assume two EBS volumes have been mounted at /mnt/influx and /mnt/db. These volumes are to be mounted to the container at the same location as the host.

Assume that the addressable hostnames for each of the three nodes are as follows:

* ix0.mycluster
* ix1.mycluster
* ix2.mycluster

**Patch the configuration**

Create an Envfile for each node in the cluster that makes the [recommended patches](https://docs.influxdata.com/influxdb/v0.10/introduction/installation/#configuring-the-instance). See the example at[./example/Envfile](https://hub.docker.com/r/amancevice/influxdb-cluster/example/Envfile):

INFLUX\_\_\_META\_\_\_DIR="/mnt/db/meta"

INFLUX\_\_\_DATA\_\_\_DIR="/mnt/db/data"

INFLUX\_\_\_DATA\_\_\_WAL\_DIR="/mnt/influx/wal"

INFLUX\_\_\_HINTED\_HANDOFF\_\_\_DIR="/mnt/db/hh"

As of InfluxDB 0.10.0, you must patch the [meta] section's values for bind-address and http-bind-address as well as the [http] bind-address option. We will use the values <hostname>:8088,<hostname>:8091, and <hostname>:8086, respectively.

**NOTE** the hostnames used must be accessible from the other nodes in the cluster. Additionally, we **must** assign the true hostname to the container using the --hostname option of docker run/docker create.

**Bring up the first node**

Update the Envfile with the patched bind addresses or pass them in directly:

docker run --detach --name ix0 \

--env INFLUX\_\_\_META\_\_\_BIND\_ADDRESS='"ix0.mycluster:8088"' \

--env INFLUX\_\_\_META\_\_\_HTTP\_BIND\_ADDRESS='"ix0.mycluster:8091"' \

--env INFLUX\_\_\_HTTP\_\_\_BIND\_ADDRESS='"ix0.mycluster:8086"' \

--env-file ./Envfile \

--hostname ix0.mycluster \

--publish 8083:8083 \

--publish 8086:8086 \

--publish 8088:8088 \

--publish 8091:8091 \

--volume /mnt/db:/mnt/db \

--volume /mnt/influx:/mnt/influx \

amancevice/influxdb-cluster

**Bring up the second node**

The second follower node is started almost identically to the first node, altering the CMD to join to the leader on port 8091:

docker run --detach --name ix1 \

--env INFLUX\_\_\_META\_\_\_BIND\_ADDRESS='"ix1.mycluster:8088"' \

--env INFLUX\_\_\_META\_\_\_HTTP\_BIND\_ADDRESS='"ix1.mycluster:8091"' \

--env INFLUX\_\_\_HTTP\_\_\_BIND\_ADDRESS='"ix1.mycluster:8086"' \

--env-file ./Envfile \

--hostname ix1.mycluster \

--publish 8083:8083 \

--publish 8086:8086 \

--publish 8088:8088 \

--publish 8091:8091 \

--volume /mnt/db:/mnt/db \

--volume /mnt/influx:/mnt/influx \

amancevice/influxdb-cluster -join ix0.mycluster:8091

**Bring up the third node**

Bring up the third follower node following this pattern:

docker run --detach --name ix2 \

--env INFLUX\_\_\_META\_\_\_BIND\_ADDRESS='"ix2.mycluster:8088"' \

--env INFLUX\_\_\_META\_\_\_HTTP\_BIND\_ADDRESS='"ix2.mycluster:8091"' \

--env INFLUX\_\_\_HTTP\_\_\_BIND\_ADDRESS='"ix2.mycluster:8086"' \

--env-file ./Envfile \

--hostname ix2.mycluster \

--publish 8083:8083 \

--publish 8086:8086 \

--publish 8088:8088 \

--publish 8091:8091 \

--volume /mnt/db:/mnt/db \

--volume /mnt/influx:/mnt/influx \

amancevice/influxdb-cluster -join ix0.mycluster:8091

And so on...

See the example at [./example/cluster.sh](https://hub.docker.com/r/amancevice/influxdb-cluster/example/cluster.sh) to see how to bring up a simple cluster on your machine.

# Architeture for influxdb cluster | Cafeteria

http://www.zhexuany.com/2017/03/20/influxd-cluster-architeture/

Architeture for influxdb cluster

 Posted on 2017-03-20 |

Influxdb-cluster

In order to improve the availability of influxDB, we create this project. The original plan is to replicate influxdata’s cluster implementation, but this is not good  
because the old plan is highly coupled with influxDB. If influxdata changes a lot interface and method signautre, the whole project will be broken. Hence, we need change our architeture and decide not to intrude influxDB too much. Please think this as a plugin, we will extract some code from influxDB for our own benefit. Two things we decide to follow is influxQL and line protocol. These two are basic feature of influxDB, hence it is much more stable than other module in influxDB. All write and query will be done in http with the endpoints provided byinfluxd.

Architecture

┌───────┐ ┌───────┐

│ │ │ │

│ node1 │◀───▶│ node2 │

│ │ │ │

└───────┘ └───────┘

▲ ▲

│ │

│ ┌───────┐ │

│ │ │ │

└─▶│ node3 │◀─┘

│ │

└───────┘

In a cluster, we will have some nodes. The metadata are synchorized via Raft algorithm. The metadata includes:

1. Meta nodes hold all of the following meta data
2. all nodes in the cluster and their role
3. all databases and retention policies that exist in the cluster
4. all shards and shard groups, and on what nodes they exist
5. all continuous queries

The data node part will store the following data:

1. measurements
2. tag keys and values
3. field keys and values

Compared with influxdata’s plan, we want meta and data node to reside in one machine. In futrue, we may separate them for better performance. But for now, it is the best start  
point. A large projet should evolve on the way of growing.

How write work in this cluster?

In a cluster, write is not easy to answer. It is a hard question. The consistency level and replication factor will affect this. For now, our primary job  
is releasing a prototype that can solve our cluster need.

Before write points into influxDB, we first know how many nodes that these points should write into and where these nodes are. These two operations should be done in the meta node part.

All data stored in cluster as a unit called Shard. Shard will be distributed and replicated across different nodes. The basic rule for know this write belong to which node  
is

|  |  |
| --- | --- |
| 1 | points ----> any node in cluster -----> find nodes -> buffer write in hinted handoff |

The key is about how to determine the destination of these point. It is related with Shards Groups.

Shards Groups

All data stored in cluster as a form of Shards. Consering the replication factor is x and there is n node available in cluster. Then n/x shards will be created in  
each group, discarding any fractions.

This means that a new shard group will be created for each dat of data that gets written in. Within shard group 2 shards will be created. Because of the replication factor of 2,  
each of those two shards will be copied on 2 servers.  
When a write comes in with values that have a timestamp, we first determine which ShardGroup that this write goes to. After this, we take the concatatention of measurement and tagset as out key and hash such key for bucketing into the correct shard. In Go, it will be the following.

|  |  |
| --- | --- |
| 1  2  3  4 | // key is measurement + tagset  // shardGroup is the group for the values based on timestamp  // hash with fnv and then bucket  shard := shardGroup.shards[fnv.New64a(key) % len(shardGroup.Shards)] |

There are multiple implications to this scheme for determining where data lives in a cluster. First, for any given metaseries all data on any given day will exist in a single shard, and thus only on those servers hosting a copy of that shard. Second, once a shard group is created, adding new servers to the cluster won’t scale out write capacity for that shard group. But we will figure a way to expand the writes in cluster in future.

When a batch points arrives, we apply the rule we just described above to find the correct shard and write data into disk.

When points arrives influxDB either in tcp or udp, it can handle this on its own. We do not need worry about this.

How query work in this cluster?

|  |  |
| --- | --- |
| 1 | query ----> any node in cluster -----> analyse query --------> find nodes -> wait until all distributed query come back -> send all data back to client |

Query in a cluster are distributed based on time range being queried and the replication factor of the data. For example if the retention policy has a replication factor of 4, the coordinating data node receiving the query randomly picks any of the 4 data nodes that store a replica of the shard(s) to receive the query. If we assume that  
the system has shard durations of one day, then for each day of time covered by a query the coordinating node will select one data node to receive the query for that day. The coordinating node will execute and fulfill the query locally whenever possible. If a query must scan multiple shard groups (multiple days in our example above), the node will will forward queries to other nodes for shard(s) it does not have locally. The queries are forwarded in parallel to scanning its own local data. The queries are distributed to as many nodes as required to query each shard group once. As the results come back from each data node, the node combines them into the final result that gets returned to the user.

Conclusion

This may not be a perfect architeture for implementing cluster for influxDB, but this is the best solution we can come up with because we want to control the intrusion the less the better.

Your support will encourage me create more！