StorageConfiguration06

Prior to the 0.7 release, Cassandra storage configuration is described by the *conf/storage-conf.xml* file. As of 0.7, it is described by the *conf/cassandra.yaml* file. As the syntax evolves with releases, this wiki page tries to document those changes using *[New in X.Y: ....]* lines.

-Dcassandra.config

[New in 0.7:

You can specify any configuration to load by passing a VM parameter to java when Cassandra starts up. This is done by supplying a value to *-Dcassandra.conf=<your value here>*. Values can include files located in the classpath or local and remote URLs. Here are a few valid examples:

Value	Implication
-Dcassandra.config=alternate-cassandra.yaml	loads *alternate-cassandra.yaml* if it can be found in the cassandra classpath.
-Dcassandra.config=http://www.example.com/remote-cassandra.yaml	loads a configuration file from a remote host.
-Dcassandra.config=file:///home/me/external-local-cassandra.yaml	loads a local configuration file that is not located in the cassandra classpath.

]

AutoBootstrap

[New in 0.5:

Turn on to make new [non-seed] nodes automatically migrate the right data to themselves. (If no InitialToken is specified, they will pick one such that they will get half the range of the most-loaded node.) If a node starts up without bootstrapping, it will mark itself bootstrapped so that you can't subsequently accidently bootstrap a node with data on it. (You can reset this by wiping your data and commitlog directories.)

Off by default so that new clusters and upgraders from 0.4 don't bootstrap immediately. You should turn this on when you start adding new nodes to a cluster that already has data on it. (If you are upgrading from 0.4, start your cluster with it off once before changing it to true. Otherwise, no data will be lost but you will incur a lot of unnecessary I/O before your cluster starts up.)

```
  <AutoBootstrap>false</AutoBootstrap>
```

]

Cluster Name

The name of this cluster. This is mainly used to prevent machines in one logical cluster from joining another.

Example:

```
<ClusterName>Test Cluster</ClusterName>
```

Authenticator

[New in 0.6:

Allows for pluggable authentication of users, which defines whether it is necessary to call the Thrift 'login' method, and which parameters are required to login. The default '!AllowAllAuthenticator' does not require users to call 'login': any user can perform any operation. The other built in option is '! SimpleAuthenticator', which requires users and passwords to be defined in property files, and for users to call login with a valid combo.

Example:

```
<Authenticator>org.apache.cassandra.auth.SimpleAuthenticator</Authenticator>
```

]

Keyspaces and ColumnFamilies

Keyspaces and `ColumnFamilies`: A `ColumnFamily` is the Cassandra concept closest to a relational table. `Keyspaces` are separate groups of `ColumnFamilies`. Except in very unusual circumstances you will have one Keyspace per application.

There is an implicit keyspace named 'system' for Cassandra internals.

```
<Keyspaces>
 <Keyspace Name="Keyspace1">
```

[New in 0.5:

The fraction of keys per sstable whose locations we keep in memory in "mostly LRU" order. (JUST the key locations, NOT any column values.) The amount of memory used by the default setting of 0.01 is comparable to the amount used by the internal per-sstable key index. Consider increasing this if you have fewer, wider rows. Set to 0 to disable entirely.

```
      <KeysCachedFraction>0.01</KeysCachedFraction>
```

]

[New in 0.6: !EndPointSnitch, !ReplicaPlacementStrategy and !ReplicationFactor became configurable per keyspace. Prior to that they were global settings.]

EndPointSnitch

EndPointSnitch: Setting this to the class that implements `IEndPointSnitch` which will see if two endpoints are in the same data center or on the same rack. Out of the box, Cassandra provides `org.apache.cassandra.locator.EndPointSnitch`

```
<EndPointSnitch>org.apache.cassandra.locator.EndPointSnitch</EndPointSnitch>
```

Note: this class will work on hosts' IPs only. There is no configuration parameter to tell Cassandra that a node is in rack *R* and in datacenter *D*. The current rules are based on the two methods: (see EndPointSnitch.java):

- isOnSameRack: Look at the IP Address of the two hosts. Compare the 3rd octet. If they are the same then the hosts are in the same rack else different racks.
- isInSameDataCenter: Look at the IP Address of the two hosts. Compare the 2nd octet. If they are the same then the hosts are in the same datacenter else different datacenter.

ReplicaPlacementStrategy and ReplicationFactor

Strategy: Setting this to the class that implements `IReplicaPlacementStrategy` will change the way the node picker works. Out of the box, Cassandra provides `org.apache.cassandra.locator.RackUnawareStrategy` and `org.apache.cassandra.locator.RackAwareStrategy` (place one replica in a different datacenter, and the others on different racks in the same one.)

```
<ReplicaPlacementStrategy>org.apache.cassandra.locator.RackUnawareStrategy</ReplicaPlacementStrategy>
```

Number of replicas of the data

```
<ReplicationFactor>1</ReplicationFactor>
```

Note that the replication factor (RF) is the *total* number of nodes onto which the data will be placed. So, a replication factor of 1 means that only 1 node will have the data. It does **not** mean that one *other* node will have the data.

ColumnFamilies

The `CompareWith` attribute tells Cassandra how to sort the columns for slicing operations. The default is `BytesType`, which is a straightforward lexical comparison of the bytes in each column. Other options are `AsciiType`, `UTF8Type`, `LexicalUUIDType`, `TimeUUIDType`, and `LongType`. You can also specify the fully-qualified class name to a class of your choice extending `org.apache.cassandra.db.marshal.AbstractType`.

- `SuperColumns` have a similar `CompareSubcolumnsWith` attribute.
- `BytesType`: Simple sort by byte value. No validation is performed.
- `AsciiType`: Like `BytesType`, but validates that the input can be parsed as US-ASCII.
- `UTF8Type`: A string encoded as UTF8
- `LongType`: A 64bit long
- `LexicalUUIDType`: A 128bit UUID, compared lexically (by byte value)
- `TimeUUIDType`: a 128bit version 1 UUID, compared by timestamp

(To get the closest approximation to 0.3-style `supercolumns`, you would use `CompareWith=UTF8Type CompareSubcolumnsWith=LongType`.)

If `FlushPeriodInMinutes` is configured and positive, it will be flushed to disk with that period whether it is dirty or not. This is intended for lightly-used `columnfamilies` so that they do not prevent commitlog segments from being purged.

[New in 0.5: An optional `Comment` attribute may be used to attach additional human-readable information about the column family to its definition. *]*

```
<ColumnFamily CompareWith="BytesType"
       Name="Standard1"
       FlushPeriodInMinutes="60"/>
<ColumnFamily CompareWith="UTF8Type"
       Name="Standard2"/>
<ColumnFamily CompareWith="TimeUUIDType"
       Name="StandardByUUID1"/>
<ColumnFamily ColumnType="Super"
       CompareWith="UTF8Type"
       CompareSubcolumnsWith="UTF8Type"
       Name="Super1"
       Comment="A column family with supercolumns, whose column and subcolumn names are UTF8 strings"/>
```

Partitioner

Partitioner: any `IPartitioner` may be used, including your own as long as it is on the classpath. Out of the box, Cassandra provides `org.apache.cassandra.dht.RandomPartitioner`, `org.apache.cassandra.dht.OrderPreservingPartitioner`, and `org.apache.cassandra.dht.CollatingOrderPreservingPartitioner`. (CollatingOPP colates according to EN,US rules, not naive byte ordering. Use this as an example if you need locale-aware collation.) Range queries require using an order-preserving partitioner.

Achtung! Changing this parameter requires wiping your data directories, since the partitioner can modify the !sstable on-disk format.

Example:

```
<Partitioner>org.apache.cassandra.dht.RandomPartitioner</Partitioner>
```

If you are using an order-preserving partitioner and you know your key distribution, you can specify the token for this node to use. (Keys are sent to the node with the "closest" token, so distributing your tokens equally along the key distribution space will spread keys evenly across your cluster.) This setting is only checked the first time a node is started.

This can also be useful with `RandomPartitioner` to force equal spacing of tokens around the hash space, especially for clusters with a small number of nodes.

```
<InitialToken></InitialToken>
```

Cassandra uses MD5 hash internally to hash the keys to place on the ring in a `RandomPartitioner`. So it makes sense to divide the hash space equally by the number of machines available using `InitialToken` ie, If there are 10 machines, each will handle 1/10th of maximum hash value) and expect that the machines will get a reasonably equal load.

With `OrderPreservingPartitioner` the keys themselves are used to place on the ring. One of the potential drawback of this approach is that if rows are inserted with sequential keys, all the write load will go to the same node.

Directories

Directories: Specify where Cassandra should store different data on disk. Keep the data disks and the `CommitLog` disks separate for best performance. See also what kind of hardware should I use?

```
<CommitLogDirectory>/var/lib/cassandra/commitlog</CommitLogDirectory>
<DataFileDirectories>
      <DataFileDirectory>/var/lib/cassandra/data</DataFileDirectory>
</DataFileDirectories>
```

Seeds

Addresses of hosts that are deemed contact points. Cassandra nodes use this list of hosts to find each other and learn the topology of the ring. You must change this if you are running multiple nodes!

```
<Seeds>
 <Seed>127.0.0.1</Seed>
</Seeds>
```

Never use a node's own address as a seed if you are bootstrapping it by setting AutoBootstrap to true.

Miscellaneous

Time to wait for a reply from other nodes before failing the command

```
<RpcTimeoutInMillis>5000</RpcTimeoutInMillis>
```

Size to allow commitlog to grow to before creating a new segment

```
<CommitLogRotationThresholdInMB>128</CommitLogRotationThresholdInMB>
```

Local hosts and ports

Address to bind to and tell other nodes to connect to. You *must* change this if you want multiple nodes to be able to communicate!

Leaving it blank leaves it up to `InetAddress.getLocalHost()`. This will always do the Right Thing *if* the node is properly configured (hostname, name resolution, etc), and the Right Thing is to use the address associated with the hostname (it might not be). The ControlPort setting is deprecated in 0.6 and can be safely removed from configuration.

```
<ListenAddress>localhost</ListenAddress>
<!-- TCP port, for commands and data -->
<StoragePort>7000</StoragePort>
<!-- UDP port, for membership communications (gossip) -->
<ControlPort>7001</ControlPort>
```

The address to bind the Thrift RPC service to. Unlike `ListenAddress` above, you *can* specify `0.0.0.0` here if you want Thrift to listen on all interfaces.

Leaving this blank has the same effect it does for `ListenAddress`, (i.e. it will be based on the configured hostname of the node).

```
<ThriftAddress>localhost</ThriftAddress>
<!-- Thrift RPC port (the port clients connect to). -->
<ThriftPort>9160</ThriftPort>
```

Whether or not to use a framed transport for Thrift. If this option is set to true then you must also use a framed transport on the client-side, (framed and non-framed transports are not compatible).

```
<ThriftFramedTransport>false</ThriftFramedTransport>
```

Memory, Disk, and Performance

Access mode. mmapped i/o is substantially faster, but only practical on a 64bit machine (which notably does not include EC2 "small" instances) or relatively small datasets. "auto", the safe choice, will enable mmapping on a 64bit JVM. Other values are "mmap", "mmap_index_only" (which may allow you to get part of the benefits of mmap on a 32bit machine by mmapping only index files) and "standard". (The buffer size settings that follow only apply to standard, non-mmapped i/o.)

```
<DiskAccessMode>auto</DiskAccessMode>
```

Buffer size to use when performing contiguous column slices. Increase this to the size of the column slices you typically perform. (Name-based queries are performed with a buffer size of ColumnIndexSizeInKB.)

```
<SlicedBufferSizeInKB>64</SlicedBufferSizeInKB>
```

Buffer size to use when flushing !memtables to disk. (Only one !memtable is ever flushed at a time.) Increase (decrease) the index buffer size relative to the data buffer if you have few (many) columns per key. Bigger is only better *if* your !memtables get large enough to use the space. (Check in your data directory after your app has been running long enough.)

```
<FlushDataBufferSizeInMB>32</FlushDataBufferSizeInMB>
<FlushIndexBufferSizeInMB>8</FlushIndexBufferSizeInMB>
```

Add column indexes to a row after its contents reach this size. Increase if your column values are large, or if you have a very large number of columns. The competing causes are, Cassandra has to deserialize this much of the row to read a single column, so you want it to be small - at least if you do many partial-row reads - but all the index data is read for each access, so you don't want to generate that wastefully either.

```
<ColumnIndexSizeInKB>64</ColumnIndexSizeInKB>
```

The maximum amount of data to store in memory per ColumnFamily before flushing to disk. Note: There is one memtable per column family, and this threshold is based solely on the amount of data stored, not actual heap memory usage (there is some overhead in indexing the columns). See also MemtableThresholds.

```
<MemtableSizeInMB>64</MemtableSizeInMB>
```

The maximum number of columns in millions to store in memory per ColumnFamily before flushing to disk. This is also a per-memtable setting. Use with `MemtableSizeInMB` to tune memory usage.

```
<MemtableObjectCountInMillions>0.1</MemtableObjectCountInMillions>
```

[New in 0.5

The maximum time to leave a dirty memtable unflushed. (While any affected columnfamilies have unflushed data from a commit log segment, that segment cannot be deleted.) This needs to be large enough that it won't cause a flush storm of all your memtables flushing at once because none has hit the size or count thresholds yet. For production, a larger value such as 1440 is recommended.

```
  <MemtableFlushAfterMinutes>60</MemtableFlushAfterMinutes>
```

]

Unlike most systems, in Cassandra writes are faster than reads, so you can afford more of those in parallel. A good rule of thumb is 2 concurrent reads per processor core. Increase `ConcurrentWrites` to the number of clients writing at once if you enable `CommitLogSync + CommitLogSyncDelay`.

```
<ConcurrentReads>8</ConcurrentReads>
<ConcurrentWrites>32</ConcurrentWrites>
```

`CommitLogSync` may be either "periodic" or "batch." When in batch mode, Cassandra won't ack writes until the commit log has been fsynced to disk. It will wait up to `CommitLogSyncBatchWindowInMS` milliseconds for other writes, before performing the sync.

This is less necessary in Cassandra than in traditional databases since replication reduces the odds of losing data from a failure after writing the log entry but before it actually reaches the disk. So the other option is "timed," where writes may be acked immediately and the `CommitLog` is simply synced every `CommitLogSyncPeriodInMS` milliseconds.

```
<CommitLogSync>periodic</CommitLogSync>
```

Interval at which to perform syncs of the `CommitLog` in periodic mode. Usually the default of 1000ms is fine; increase it only if the CommitLog PendingTasks backlog in jmx shows that you are frequently scheduling a second sync while the first has not yet been processed.

```
<CommitLogSyncPeriodInMS>1000</CommitLogSyncPeriodInMS>
```

Delay (in milliseconds) during which additional commit log entries may be written before fsync in batch mode. This will increase latency slightly, but can vastly improve throughput where there are many writers. Set to zero to disable (each entry will be synced individually). Reasonable values range from a minimal 0.1 to 10 or even more if throughput matters more than latency.

```
<!-- <CommitLogSyncBatchWindowInMS>1</CommitLogSyncBatchWindowInMS> -->
```

Time to wait before garbage-collection deletion markers. Set this to a large enough value that you are confident that the deletion marker will be propagated to all replicas by the time this many seconds has elapsed, even in the face of hardware failures. The default value is ten days.

```
<GCGraceSeconds>864000</GCGraceSeconds>
```

Number of threads to run when flushing memtables to disk. Set this to the number of disks you physically have in your machine allocated for `DataDirectory * 2`. If you are planning to use the Binary Memtable, its recommended to increase the max threads to maintain a higher quality of service while under load when normal memtables are flushing to disk.

```
<FlushMinThreads>1</FlushMinThreads>
<FlushMaxThreads>1</FlushMaxThreads>
```

The threshold size in megabytes the binary memtable must grow to, before it's submitted for flushing to disk.

```
<BinaryMemtableSizeInMB>256</BinaryMemtableSizeInMB>
```

Including Configuration Fragments

It's common that a Cassandra configuration will be shared among many machines but needs to be slightly tuned on each one (directories are different, memory available is less, etc.). You can include a XML fragment with this syntax.

```
...
<!DOCTYPE storage [
<!-- include all these entities from external files-->
<!ENTITY seeds       SYSTEM "seeds.xml">
<!ENTITY directories SYSTEM "directories.xml">
<!ENTITY network     SYSTEM "network.xml">
<!ENTITY tuning      SYSTEM "tuning.xml">
]>
...
<Storage>
...
&seeds;
&directories;
&network;
&tuning;
...
</Storage>
```

And then the external files are simply what you'd specify inline, for example directories.xml. Note these fragments are not valid XML alone.

```
  <CommitLogDirectory>/var/lib/cassandra/commitlog</CommitLogDirectory>
  <DataFileDirectories>
      <DataFileDirectory>/var/lib/cassandra/data</DataFileDirectory>
  </DataFileDirectories>
  <StagingFileDirectory>/var/lib/cassandra/staging</StagingFileDirectory>
```