StorageConfiguration

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.

Loading the Config

config-converter

To convert a working storage-conf.xml into yaml, bin/config-converter will complete a best-effort conversion. It assumes that your xml file is at conf/storage-conf.xml, and it will output to conf/cassandra.yaml. Note though that the scope of many properties have changed (endpoint_snitch is global instead of per-KS, gc_grace_seconds is per-KS instead of global, etc.), so the generated yaml will undoubtedly need to be looked over.

-Dcassandra.config

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.

"Where are my keyspaces?"

LiveSchemaUpdates. You can load the schema once by using:

```
bin/schematool HOST PORT import
```

Config Overview

Not going to cover every value, just the interesting ones. When in doubt, check out the comments on the default cassandra.yaml as they're well documented there.

per-Cluster (Global) Settings

- **authenticator**

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.

Default is: 'org.apache.cassandra.auth.AllowAllAuthenticator', a no-op.

- **auto_bootstrap**

Set to 'true' 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.)

Default is: 'false', so that new clusters don't bootstrap immediately. You should turn this on when you start adding new nodes to a cluster that already has data on it.

- **cluster_name**

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

- **commitlog_directory** and **data_file_directories** and **saved_caches_directory**

Be sure to seperate your commitlog and data disks, as commitlog performance is reliant on its append-only nature, and seeking to random data at the same time will damage write speed. The saved caches directory holds the saved caches of the column families. See **key /row_cache_save_period_in_seconds** for elucidation.

Defaults are: '/var/lib/cassandra/commitlog', '/var/lib/cassandra/data', and '/var/lib/cassandra/saved_caches' respectively.

- **concurrent_reads** and **concurrent_writes**

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 4 concurrent_reads per processor core. It's unwise to adjust the `concurrent_writes` until you have a a performance problem to address. In general, though, for a dedicated cluster it should exceed somewhat the number of cpu-cores on the ring

Defaults are: '8' c. reads, and '32' c. writes.

- **commitlog_rotation_threshold_in_mb**, **commitlog_sync**, **commitlog_sync_period_in_ms**, and **commitlog_sync_batch_window_in_ms**

The rotation threshold determines how often a new commitlog segment is created. This number is generally never changed, but can be reduced if small amount of memory need to be squeezed from the system.

`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 "periodic", where writes may be acked immediately and the `CommitLog` is simply synced every `CommitLogSyncPeriodInMS` milliseconds. 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.

Defaults are: '128'mb commitlog size, 'periodic' sync, '10000' ms between syncs.

- **disk_access_mode**

In 0.7, the default 'auto' is recommended.

- **dynamic_snitch** and **endpoint_snitch**

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 4 such Snitches, found in `org.apache.cassandra.locator` :

1. `org.apache.cassandra.locator.SimpleSnitch` which will do nothing,
2. `org.apache.cassandra.locator.RackInferringSnitch` which will assume that the 2nd and 3rd octets of the IP contain the datacenter and rack,
3. `org.apache.cassandra.locator.PropertyFileSnitch` which will read from cassandra-topology.properties to give explicit proximities, and
4. `org.apache.cassandra.locator.Ec2Snitch` which will read the region and zone from the ec2 node and use them as datacenter and rack. Don't use this if you're not running on EC2.

Dynamic Snitch is a boolean that controls if the above snitch is wrapped with a dynamic snitch, which will monitor read latencies and avoid reading from hosts that have slowed.

Defaults are: 'org.apache.cassandra.locator.SimpleSnitch' and 'false'.

- **listen_address**

Commenting out this property 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: on cloud services you should ensure the private interface is used).

Default is: 'localhost'. This must be changed for other nodes to contact this node.

- **memtable_flush_after_mins**, **memtable_operations_in_millions**, and **memtable_throughput_in_mb**

memtable_flush_after_mins: 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.

memtable_operations_in_millions: 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.

memtable_throughput_in_mb: 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.

Both mem_ops and mem_size are defaulted based on the heap allocation during boot.

Defaults are: '60' minutes, 'HeapSize/512 * 0.3' millions (i.e. 300k operations per 512 mb of heap, or 64 mb of throughput), and 'HeapSize/8' mb respectively, where HeapSize is measured in mb.

- **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`, `org.apache.cassandra.dht.ByteOrderedPartitioner`, 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.) The only difference between BOP and OPP is that OPP requires keys to be UTF-8 encoded. 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.

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.

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.

Default is: 'org.apache.cassandra.dht.RandomPartitioner'. Manually assigning tokens is highly recommended to guarantee even load distribution.

- **seeds**

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

- **thrift_framed_transport_size_in_mb**

Setting this to '0' is how to denote using unframed (Buffered) transport, and a nonzero value for framed transport.

Default is: '15' mb.

per-Keyspace Settings

- **name**

Required field. Will not allow you to use dashes.

- **replica_placement_strategy** and **replication_factor**

Pre-0.8.1

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.)

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.

Defaults are: 'org.apache.cassandra.locator.RackUnawareStrategy' and '1'. RF of at least 2 is highly recommended, keeping in mind that your effective number of nodes is (N total nodes / RF).

0.8.1

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.SimpleStrategy`, `org.apache.cassandra.locator.LocalStrategy` and `org.apache.cassandra.locator.NetworkTopologyStrategy` (place one replica in a different datacenter, and the others on different racks in the same one.)

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.

Defaults are: 'org.apache.cassandra.locator.NetworkTopologyStrategy' and '1'. RF of at least 2 is highly recommended, keeping in mind that your effective number of nodes is (N total nodes / RF).

per-ColumnFamily Settings

- **comment** and **name**

You can describe a ColumnFamily in plain text by setting these properties.

- **compare_with**

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`.

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

These are currently the same types used for validators.

- **default_validation_class**

Used in conjunction with the validation_class property in the per-column settings to guarantee the type of a column value.

Default is: 'BytesType', a no-op.

- **gc_grace_seconds**

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.

Default is: '864000' seconds, or 10 days.

- **keys_cached** and **rows_cached**

Determines how many keys and rows to cache. The values can either be an absolute value or a double between 0 and 1 (inclusive on both ends) denoting what fraction should be cached.

Each key cache hit saves 1 seek and each row cache hit saves 2 seeks at the minimum, sometimes more. The key cache is fairly tiny for the amount of time it saves, so it's worthwhile to use it at large numbers all the way up to 1.0 (all keys cached). The row cache saves even more time, but must store the whole values of its rows, so it is extremely space-intensive. It's best to only use the row cache if you have hot rows or static rows.

Defaults are: '200000' keys cached, and '0', disabled row cache.

- **key_cache_save_period_in_seconds** and **row_cache_save_period_in_seconds**

Determines how often Cassandra saves the cache to the **saved_caches_directory**. Saved caches greatly improve cold-start speeds, and is relatively cheap in terms of I/O for the key cache. Row cache saving is much more expensive and has limited use.

Defaults are: '3600' seconds (1 hour) between saves of the key cache, and '0' (disabled) row cache saving.

- **min_compaction_threshold** and **max_compaction_threshold**

Previously in the CompactionManager, these values tune the size and frequency of minor compactions. The min and max boundaries are the number of tables to attempt to merge together at once. Raising the minimum will make minor compactions take more memory and run less often, lowering the maximum will have the opposite effect.

Note: Setting minimum and maximum to 0 will disable minor compactions. USE AT YOUR OWN PERIL!

Defaults are: '4' minimum tables to compact at once, and '32' maximum.

- **preload_row_cache**

Attempts to populate the row cache on start up with sequential reads. Despite the improvement over random seeks during runtime, this can still take a fairly long time if the row cache to fill is massive.

Default is: 'false'.

- **read_repair_chance**

Before 0.7, read_repair was either invoked on every read request or on none of them. This is now tunable as a double between 0 and 1 (inclusive on both ends) for the chance of invoking the repair. Note that this property is ignored if the ConsistencyLevel is greater than ONE and read repair always occurs. For details see ReadRepair.

Default is: '1.0', read repair on every read request.

per-Column Settings

- **index_name** and **index_type**

These settings control the secondary (automatic) indexes. Both need to be set at the same time to be used. The name is something user-friendly and unique on the CF, the type is currently only KEYS. See SecondaryIndexes.

Default is: None, no secondary index support.

- **name**

This binds the validator (and optionally the automatic indexer) to every column with this **name** in every row of the enclosing CF. Required.

- **validation_class**

Used with the default_validation_class property in the per-columnfamily settings. Whenever the column with this name is populated, the value is validated with the validation classes' validate() method. Required.