CONTENTS INCLUDE:

Apache Cassandra

A Fault-Tolerant, Massively Scalable NoSQL Database

By Brian O'Neill

APACHE CASSANDRA

Apache Cassandra is a high-performance, extremely scalable, fault tolerant (i.e., no single point of failure), distributed non-relational database solution. Cassandra combines all the benefits of Google Bigtable and Amazon Dynamo to handle the types of database management needs that traditional RDBMS vendors cannot support. DataStax is the leading worldwide commercial provider of Cassandra products, services, support, and training.

Who is using Cassandra?

Cassandra is in use at Netflix, Twitter, Urban Airship, Constant Contact, Reddit, Cisco, OpenX, Rackspace, Ooyala, and more companies that have large active data sets. The largest known Cassandra cluster has over 300 TB of data in over 400 machines.

(From: http://cassandra.apache.org/)

	Cassandra	RDBMS
Atomicity	Success or failure on a row-by-row basis.	Enforced at every scope, at the cost of performance and scalability.
Sharding	Native share-nothing architecture, inherently partitioned by a configurable strategy.	Often forced when scaling, partitioned by key or function
Consistency	No consistency in the ACID sense. Can be tuned to provide consistency in the CAP sense--data is consistent across all the nodes in a distributed database cluster ,guaranteeing read-after-write or eventual readability.	Favors consistency over availability tunable via isolation levels.
Durability	Writes are durable to a replica node, being recorded in memory and the commit log before acknowledged. In the event of a crash, the commit log replays on restart to recover any lost writes before data is flushed to disk.	Typically, data is written to a single master node, sometimes configured with synchronous replication at the cost of performance and cumbersome data restoration.
Multi-Datacenter Replication	Native capabilities for data replication over lower bandwidth, higher latency, less reliable connections.	Typically only limited long-distance replication to read-only slaves receiving asynchronous updates.
Security	Coarse-grained and primitive.	Fine-grained access control to objects.

DATA MODEL OVERVIEW

Cassandra has a simple schema comprising keyspaces, column families, rows, and columns.

	Definition	RDBMS Analogy	Object Equivalent
Schema/ Keyspace	A collection of column families.	Schema/Database	Set<ColumnFamily>
Table/ Column Family	A set of rows.	Table	Map<rowKey, Row>
Row	An ordered set of columns.	Row	OrderedMap <columnKey, Column>
Column	A key/value pair and timestamp.	Column (Name, Value)	(key, value, timestamp)

Schema

Also known as a keyspace, the schema is akin to a database or schema in RDBMS and contains a set of tables. A schema is also the unit for Cassandra's access control mechanism. When enabled, users must authenticate to access and manipulate data in a schema or table.

Tables

A table, also known as a column family, is a map of rows. A table defines the column names and data types. The client application provides rows that conform to the schema. Each row has the same fixed set of columns. As values for these properties, Cassandra provides the following CQL data types for columns.

As Values for these properties, Cassandra provides the following CQL data types for columns.

Type	Purpose	Storage
ascii	Efficient storage for simple ASCII strings.	Arbitrary number of ASCII bytes (i.e., values are 0-127).
boolean	True or False.	Single byte.
blob	Arbitrary byte content.	Arbitrary number of byes.
Composite Type	A single type comprising sub-components each with their own types.	An arbitrary number of bytes comprising concatenated values of the subtypes.
counter	Used for counters, which are cluster-wide incrementing values.	8 bytes.
timestamp	Stores time in milliseconds.	8 bytes.
decimal	Stores BigDecimals.	4 bytes to store the scale, plus an arbitrary number of bytes to store the value.
double	Stores Doubles.	8 bytes.
float	Stores Floats.	4 bytes.
int	Stores 4-byte integer.	4 bytes.

varint	Stores variable precision integer.	An arbitraty number of bytes used to store the value.
bigint	Stores Longs.	8 bytes.
text, varchar	Stores text as UTF8.	UTF8.
uuid	Suitable for UUID storage.	16 bytes.

Rows

Cassandra 1.1 supports tables defined with composite primary keys. The first column in a composite key definition is used as the partition key. Remaining columns are automatically clustered. Rows that share a partition key are sorted by the remaining components of the primary key.

Columns

A column is a triplet: key, value, and timestamp. The validation and comparator on the column family define how Cassandra sorts and stores the bytes in column keys.

The timestamp portion of the column is used to sequence mutations. The timestamp is defined and specified by the client and can be anything the client wishes to use. By convention, the timestamp is typically microseconds since epoch. If time-based, clients must be careful to synchronize clocks.

Columns may optionally have a time-to-live (TTL), after which Cassandra asynchronously deletes them.

Hot Tip Originally SuperColumns were one of Cassandra's data model primitives. Although they are still supported in the API, we recommend you use CompositeTypes instead.

CASSANDRA ARCHITECTURE

Cassandra uses a ring architecture. The ring represents a cyclic range of token values (i.e., the token space). Each node is assigned a position on the ring based on its token. A node is responsible for all tokens between its initial token and the initial token of the closest previous node along the ring.

PARTITIONING

Keys are mapped into the token space by a partitioner. The important distinction between the partitioners is order preservation (OP). Users can define their own partitioners by implementing IPartitioner, or they can use one of the native partitioners:

	Map Function	Token Space	OP
RandomPartitioner	MD5	BigInteger	No
BytesOrder Partitioner	Identity	Bytes	Yes

The following examples illustrate this point.

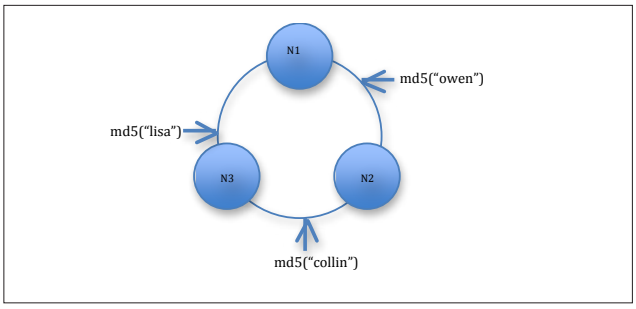
Random Partitioner

Since the Random Partitioner uses an MD5 hash function to map keys into tokens, on average those keys will evenly distribute across the cluster. For this reason, RandomPartitioner is the default partitioner.

The row key determines the node placement:

Row Key			
Lisa	state: CA	graduated: 2008	gender: F
Owen	state: TX	gender: M	
Collin	state: UT	gender: M	

This may result in following ring formation, where "collin", "owen", and "lisa" are rowkeys.



The MD5 hash operation results in a 128-bit number for keys, shown as md5("name"):

Row Key	MD5 Hash	Node
collin	CC982736AD62AB	3
owen	9567238FF72635	2
lisa	001AB62DE123FF	1

Notice that the keys are not in order. With RandomPartitioner, the keys are evenly distributed across the ring using hashes, but you sacrifice order, which means any range query needs to query all nodes in the ring.

Order Preserving Partitioners (OPP)

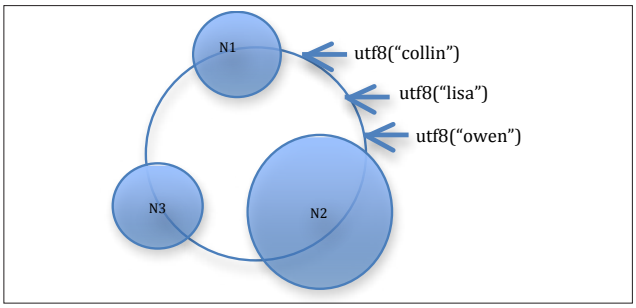
The Order Preserving Partitioners preserve the order of the row keys as they are mapped into the token space.

In our example, since:

```
"collin" < "lisa" < "owen"
then,
token("collin") < token("lisa") < token("owen")
```

With OPP, range queries are simplified and a query may not need to consult each node in the ring. This seems like an advantage, but it comes at a price. Since the partitioner is preserving order, the ring may become unbalance unless the rowkeys are naturally distributed across the token space.

This is illustrated below.



To manually balance the cluster, you can set the initial token for each node in the Cassandra configuration.

Hot Tip If possible, it is best to design your data model to use RandomPartitioner to take advantage of the automatic load balancing and decreased administrative overhead of manually managing token assignment.

REPLICATION

Cassandra provides high availability and fault tolerance through data replication. The replication uses the ring to determine nodes used for replication. Each keyspace has an independent replication factor, n. When writing information, the data is written to the target node as determined by the partitioner and n-1 subsequent nodes along the ring.

There are two replication strategies: SimpleStrategy and NetworkTopologyStrategy.

SimpleStrategy

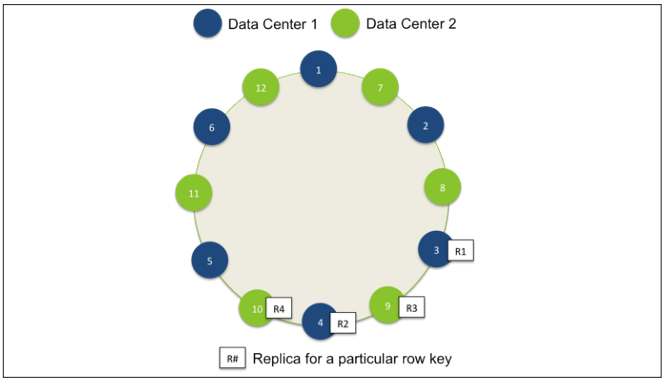
The SimpleStrategy is the default strategy and blindly writes the data to subsequent nodes along the ring. In the previous example with a replication factor of 2, this would result in the following storage allocation.

Row Key	Replica 1 (as determined by partitioner)	Replica 2 (found by traversing the ring)
collin	3	1
owen	2	3
N3	1	2

NetworkTopologyStrategy

The NetworkTopologyStrategy is useful when deploying to multiple data centers. It ensures data is replicated across data centers.

Effectively, the NetworkTopologyStrategy executes the SimpleStrategy independently for each data center, spreading replicas across distant racks. Cassandra writes a copy in each data center as determined by the partitioner. Data is written simultaneously along the ring to subsequent nodes within that data center with preference for nodes in different racks to offer resilience to hardware failure. All nodes are peers and data files can be loaded through any node in the cluster, eliminating the single point of failure inherent in master-slave architecture and making Cassandra fully fault-tolerant and highly available.



Given the following ring and deployment topology:

With blue nodes (N1-N3) deployed to one data center (DC1), red nodes (N4-N6) deployed to another data center (DC2), and a replication factor of 4, Cassandra would write a row with key "lisa" as follows.

NOTE: Cassandra attempts to write data simultaneously to all target nodes then waits for confirmation from the relevant number of nodes needed to satisfy the specified consistency level.

Consistency Levels

One of the unique characteristics of Cassandra that sets it apart from other databases is its approach to consistency. Clients can specify the consistency level on both read and write operations trading off between high availability, consistency, and performance.

Write

Level	Expectation
ANY	The write was logged, but the data may not be available for reads immediately. This is useful where you need high availability for writes but only eventual consistency on reads.
ONE	Data is committed to at least one replica and is available for reads.
TWO	Data is committed to at least two replicas and is available for reads.
THREE	Data is committed to at least three replicas and is available for reads.
QUORUM	Data is committed to at least n/2+1 replicas and is available for reads, where n is the replication factor.
LOCAL_QUORUM	Data is committed to at least n/2+1 replicas within the local data center.
EACH_QUORUM	Data is committed to at least n/2+1 replicas within each data center.
ALL	Data is committed to and available from all n replicas. This is useful when absolute read consistency and/or fault tolerance are necessary (e.g., online disaster recovery).

Read

Level	Expectation
ONE	The client receives data from the first replica to respond.
TWO	The client receives the most current data between two replicas based on the timestamps.
THREE	The client receives the most current data between three replicas based on the timestamps.
QUORUM	The client receives the most current data once n/2+1 replicas have responded.
LOCAL_QUORUM	The client receives the most current data once n/2+1 replicas have responded within the local data center.
EACH_QUORUM	The client receives the most current data once n/2+1 replicas have responded within each data center.
ALL	The client receives the most current data once all replicas have responded.

NETWORK TOPOLOGY

As input into the replication strategy and to efficiently route communication, Cassandra uses a snitch to determine the data center and rack of the nodes in the cluster. A snitch is a component that detects and informs Cassandra about the network topology of the deployment.

The snitch dictates what is used in the strategy options to identify replication groups when configuring replication for a keyspace.

Nodes	Rack	DC	Reason
N4	3	2	As determined by partitioner in DC1.
N2	1	1	As determined by partitioner in DC2.
N6	4	2	Preference shown for Rack 4 (over Rack 3).
N3	1	1	Written to same rack hosting N2 since no other rack was available.

The following table shows the four snitches provided by Cassandra and what you should use in your keyspace configuration for each snitch.

Snitch	Specify
SimpleSnitch	Specify only the replication factor in your strategy options.
PropertyFileSnitch	Specify the data center names from your properties file in the keyspace strategy options.
RackInferringSnitch	Specify the second octet of the IPv4 address in your keyspace strategy options.
EC2Snitch	Specify the region name in the keyspace strategy options.

SimpleSnitch

The SimpleSnitch provides Cassandra no information regarding racks or data centers. It is the default setting and is useful for simple deployments where all servers are collocated.

PropertyFileSnitch

The PropertyFileSnitch allows users to be explicit about their network topology. The user specifies the topology in a properties file, cassandra-topology.properties. The file specifies which nodes belong to which racks and data centers. Below is an example property file for our sample cluster.

```
# DC1
192.168.0.1=DC1:RAC1
192.168.0.2=DC1:RAC1
192.168.0.3=DC1:RAC2

# DC2
192.168.1.4=DC2:RAC3
192.168.1.5=DC2:RAC3
192.168.1.6=DC2:RAC4

# Default for nodes
default=DC3:RAC5
```

RackInferringSnitch

The RackInferringSnitch infers network topology by convention. From the IPv4 address (e.g., 9.100.47.75), the snitch uses the following convention to identify the data center and rack:

Octet	Example	Indicates
1	9	Nothing
2	100	Data Center
3	47	Rack
4	75	Node

EC2Snitch

The EC2Snitch is useful for deployments to Amazon's EC2. It uses Amazon's API to examine the regions to which nodes are deployed. It then treats each region as a separate data center.

EC2MultiRegionSnitch

Use this snitch for deployments on Amazon EC2 where the cluster spans multiple regions. This snitch treats data centers and availability zones as racks within a data center and uses public IPs as broadcast_address to allow cross-region connectivity. Cassandra nodes in one EC2 region can bind to nodes in another region, thus enabling multi-data center support.

QUERYING/INDEXING

Cassandra provides simple primitives. Its simplicity allows it to scale linearly with high availability and very little performance degradation. That simplicity allows for extremely fast read and write operations for specific keys, but servicing more sophisticated queries that span keys requires pre-planning.

Using the primitives that Cassandra provides, you can construct indexes that support exactly the query patterns of your application. Note, however, that queries may not perform well without properly designing your schema.

Secondary Indexes

To satisfy simple query patterns, Cassandra provides a native indexing capability called Secondary Indexes. A column family may have multiple secondary indexes. A secondary index is hash-based and uses specific columns to provide a reverse lookup mechanism from a specific column value to the relevant row keys. Under the hood, Cassandra maintains hidden column families that store the index. The strength of Secondary Indexes is allowing queries by value. Secondary indexes are built in the background automatically without blocking reads or writes. To create a Secondary Index using CQL is straight-forward. For example, define a table of data about movie fans, and then create a secondary index of states where they live:

```
CREATE TABLE fans ( watcherID uuid, favorite_actor text,  address text, zip int,  state text PRIMARY KEY (watcherID)  );
CREATE INDEX watcher_state ON fans (state);
```

Range Queries

It is important to consider partitioning when designing your schema to support range queries.

Range Queries with Order Preservation

Since order is preserved, order preserving partitioners better supports range queries across a range of rows. Cassandra only needs to retrieve data from the subset of nodes responsible for that range. For example, if we are querying against a column family keyed by phone number and we want to find all phone numbers between that begin with 215-555, we could create a range query with start key 215-555-0000 and end key 215-555-9999.To service this request with OrderPreservingPartitioning, it's possible for Cassandra to compute the two relevant tokens: token(215-555-0000) and token(215-555-9999).Then satisfying that querying simply means consulting nodes responsible for that token range and retrieving the rows/tokens in that range.

Range Queries with Random Partitioning

The RandomPartitioner provides no guarantees of any kind between keys and tokens. In fact, ideally row keys are distributed around the token ring evenly. Thus, the corresponding tokens for a start key and end key are not useful when trying to retrieve the relevant rows from tokens in the ring with the RandomPartitioner. Consequently, Cassandra must consult all nodes to retrieve the result. Fortunately, there are well known design patterns to accommodate range queries. These are described below.

Index Patterns

There are a few design patterns to implement indexes. Each services different query patterns. The patterns leverage the fact that Cassandra columns are always stored in sorted order and all columns for a single row reside on a single host.

Inverted Indexes

First, let's consider the inverted index pattern. In an inverted index, columns in one row become row keys in another. Consider the following data set, where users IDs are row keys.

Column Family: Users			
RowKey	Columns		
BONE42	{ name : "Brian"}	{ zip: 15283}	{dob : 09/19/1982}
LKEL76	{ name : "Lisa"}	{ zip: 98612}	{dob : 07/23/1993}
COW89	{ name : "Dennis"}	{ zip: 98612}	{dob : 12/25/2004}

Without indexes, searching for users in a specific Zip Code would mean scanning our Users column family row-by-row to find the users in the relevant Zip Code. Obviously, this does not perform well.

To remedy the situation, we can create a column family that represents the query we want to perform, inverting rows and columns. This would result in the following column family.

Column Family: Users_by_ZipCode			
RowKey	Columns		
98612	{ user_id : LKEL76 }	{ user_id : COW89 }	
15283	{ user_id : BONE42 }		

Since each row is stored on a single machine, Cassandra can quickly return all user IDs within a single Zip Code by returning all columns within a single row. Cassandra simply goes to a single host based on token(zipcode) and returns the contents of that single row.

Wide-row Indexes

When working with time series data, consider storing the complete set of data for each event in the timeline itself by serializing the entire event into a single column value or by using composite column names of the form <timestamp>:<event_field>. Unless the data for each event is very large, this approach scales well with large data sets and provides efficient reads. Fetch a time slice of events by reading a contiguous portion of a row on one set of replicas. When you track the same event in multiple timelines, denormalizing and storing all of the event data in each of the timelines works well.

Materialized View Table		
lsmith: 1332960000	C4e1ee6f-e053-41f5-9890-674636d51095: {“user”: “lsmith”, “body”: “There are . . . “}	39f71a85-7af0 . . . {“user”: “lsmith”, “body”: “Yes, . . .
cbrown: 1332960000	e572bad1-f98d-4346-80a0-13e7d37d38d0: {“user”:”cbrown”, “body”: “My dog is . . .”}	aa33bgbfd-8f16 . . . {“user”:”cbrown”, “body”:”No, . . .

When you use composite keys in CQL, Cassandra supports wide Cassandra rows using composite column names. In CQL 3, a primary key can have any number (1 or more) of component columns, but there must be at least one column in the column family that is not part of the primary key. The new wide row technique consumes more storage because for every piece of data stored, the column name is stored along with it.

```
CREATE TABLE History.events (
  event uuid PRIMARY KEY,
  author varchar,
  body varchar);

CREATE TABLE timeline (
  user varchar,
  event uuid,
  author varchar,
  body varchar,
```

Hot Tip Wide-Row indexes can cause hotspots in the cluster. Since the index is a single row, it is stored on a single node (plus replicas). If that is a heavily used index, those nodes may be overwhelmed.

Composite-Types in Indexes

Using composite keys in indexes, we can create queries along multiple dimensions. If we combine the previous examples, we could create a single wide-row capable of serving a compound query such as, “How many users within the 18964 Zip Code are older than 21?”

Simply create a composite type containing the Zip Code and the date of birth and use that as the column name in the index.

Denormalization

Finally, it is worth noting that each of the indexing strategies as presented would require two steps to service a query if the request requires the actual column data (e.g., user name). The first step would retrieve the keys out of the index. The second step would fetch each relevant column by row key.

We can skip the second step if we denormalize the data. In Cassandra, denormalization is the norm. If we duplicate the data, the index becomes a true materialized view that is custom tailored to the exact query we need to support.

INSERTING/UPDATING/DELETING

Everything in Cassandra is a write, typically referred to as a mutation. Since Cassandra is effectively a key-value store, operations are simply mutations of a key/value pairs. The column is atomic, but the fundamental unit is a row in the ACID sense. If you have multiple updates to the same key, group the changes into a single update.

Hinted Handoff

Similar to ReadRepair, Hinted Handoff is a background process that ensures data integrity and eventual consistency. If a replica is down in the cluster and the client requests a consistency level of ANY, a write may still succeed by writing a “hint” to a coordinator node, which will disseminate that data to replicas when they become available.

OPERATIONS AND MAINTENANCE

Cassandra provides tools for operations and maintenance. Some of the maintenance is mandatory because of Cassandra’s eventually consistent architecture. Other facilities are useful to support alerting and statistics gathering. Use nodetool to manage Cassandra. Datastax provides a reference card on nodetool available here:http://www.datastax.com/wp-content/uploads/2012/01/DS_nodetool_web.pdf

Nodetool Repair

Cassandra keeps record of deleted values for some time to support the eventual consistency of distributed deletes. These values are called tombstones. Tombstones are purged after some time (GCGraceSeconds, which defaults to 10 days). Since tombstones prevent improper data propagation in the cluster, you will want to ensure that you have consistency before they get purged.

To ensure consistency, run:

```
>$CASSANDRA_HOME/bin/nodetool repair
```

The repair command replicates any updates missed due to downtime or loss of connectivity. This command ensures consistency across the cluster and obviates the tombstones. You will want to do this periodically on each node in the cluster (within the window before tombstone purge).

Monitoring

Cassandra has support for monitoring via JMX, but the simplest way to monitor the Cassandra node is by using OpsCenter, which is designed to manage and monitor Cassandra database clusters. There is a free community edition as well as an enterprise edition that provides management of Apache SOLR and Hadoop.

Simply download mx4j and execute the following:

```
> cp $MX4J_HOME/lib/mx4j-tools.jar $CASSANDRA_HOME/lib
```

The following are key attributes to track per column family.

Attribute	Provides
Read Count	Frequency of reads against the column family.
Read Latency	Latency of reads against the column family.
Write Count	Frequency of writes against the column family.
Write Latency	Latency of writes against the column family.
Pending Tasks	Queue of pending tasks, informative to know if tasks are queuing.

Backup

OpsCenter facilitates backing up data by providing snapshots of the data. A snapshot creates a new hardlink to every live SSTable. Cassandra also provides online backup facilities using nodetool. To take a snapshot of the data on the cluster, invoke:

```
>$CASSANDRA_HOME/bin/nodetool snapshot
```

This will create a snapshot directory in each keyspace data directory. Restoring the snapshot is then a matter of shutting down the node, deleting the commitlogs and the data files in the keyspace, and copying the snapshot files back into the keyspace directory.

CLIENT LIBRARIES

Cassandra has a very active community developing libraries in different languages.

Java

Client	Description
Astyanax	Inspired by Hector, Astyanax is a client library developed by the folks at Netflix. https://github.com/Netflix/astyanax
Hector	Hector is one of the first APIs to wrap the underlying Thrift API. Hector is one of the most commonly used client libraries. https://github.com/rantav/hector

CQL

Client	Description
CQL	Cassandra provides an SQL-like query language called the Cassandra Query Language (CQL). The CQL shell allows you to interact with Cassandra as if it were a SQL database. Start the shell with: >$CASSANDRA_HOME/bin/cqlsh Datastax provides a reference card for CQL available here: http://www.datastax.com/wp-content/uploads/2012/01/DS_CQL_web.pdf

PHP CQL

Client	Description
Cassandra-PDO	A CQL (Cassandra Query Language) driver for PHP. http://code.google.com/a/apache-extras.org/p/cassandra-pdo/

Python

Client	Description
Pycassa	Pycassa is the most well known Python library for Cassandra. https://github.com/pycassa/pycassa

Ruby

Client	Description
Ruby Gem	Ruby has support for Cassandra via a gem. http://rubygems.org/gems/cassandra

REST

Client	Description
Virgil	Virgil is a java-based REST client for Cassandra. https://github.com/hmsonline/virgil

.NET

Client	Description
Cassandra-Sharp	.NET client for Cassandra

Node.js

Client	Description
Cassandra-Node	Cassandra/ CQL driver for Node.js

There are also community supported client libraries for .NET [http://code.google.com/p/cassandra-sharp/] and Node.js [https://github.com/racker/node-cassandra-client]

Command Line Interface (CLI)

Cassandra also provides a Command Line Interface (CLI) through which you can perform all schema related changes. It also allows you to manipulate data. Datastax provides a reference card on the CLI available here:

http://www.datastax.com/wp-content/uploads/2012/01/DS_CLI_web.pdf

Hadoop Support

DataStax Enterprise provides Cassandra with an enhanced Hadoop distribution that is compatible with existing HDFS, Hadoop, and Hive tools and utilities. Cassandra also provides out-of-the-box support for Hadoop. To see the canonical word count example, take a look at:

https://github.com/apache/cassandra/tree/trunk/examples/hadoop_word_count

DataStax Community Edition

DataStax Community Edition provides the latest release from the Apache Cassandra community.

- Binary tarballs for Linux and Mac installation
- Packaged installations for Red Hat Enterprise Linux, CentOS, Debian, and Ubuntu
- A GUI installer for Windows

RHEL and Debian packages are supported through yum and apt package management tools. The DataStax Community Edition also includes the DataStax OpsCenter.

ABOUT THE AUTHOR

Brian O'Neill is Lead Architect at Health Market Science, where he heads design and development of their Master Data Management (MDM) solution and Big Data platform, which is powered by Cassandra. Brian has experience as a technology leader and architect in a wide variety of settings from early startups to Fortune 500 companies. He has participated in a number of Java Community Process expert groups and has patents in artificial intelligence and dynamic application data routing. On github, Brian leads the Virgil project, which is a services layer built on Cassandra that provides REST, map/reduce, search and distributed processing capabilities. He also leads projects delivering trigger functionality and server-side wide-row indexing.

RECOMMENDED BOOK

Cassandra: High Performance Cookbook

This book provides detailed recipes that describe how to use the features of Cassandra and improve its performance. Recipes cover topics ranging from setting up Cassandra for the first time to complex multiple data center installations. The recipe format presents the information in a concise actionable form

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
150 Preston Executive Dr.
Suite 200
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-45-5
ISBN-10: 1-936502-45-3
50795
9 781936 502455
$7.95

Version 1.0