Tools for Working with Big Data

06016414 and 06026207: NoSQL Database Systems

Instructor: Asst. Prof. Dr. Praphan Pavarangkoon

Outline

- Designing for Column Family Databases
    - Tools for Working with Big Data
    - Case Study: Customer Data Analysis
- Lab Session

Tools for Working with Big Data

- NoSQL database options, such as key-value, document, and graph databases, are used with a wide range of applications with varying data sizes.
- Column family databases certainly could be used with small data sets, but other database types are probably better options.

Tools for Working with Big Data (cont.)

- The term *Big Data* does not have a precise definition. Informally, data sets that are too large to efficiently store and analyze in relational databases are considered Big Data.
- A more formal and commonly used definition is due to the Gartner research group.
  - High velocity
  - High volume
  - High variety

Tools for Working with Big Data (cont.)

- Databases are designed to store and retrieve data, and they perform these operations well.
- There are, however, a number of supporting and related tasks that are usually required to get the most out of your database.
  - Extracting, transforming, and loading data (ETL)
  - Analyzing data
  - Monitoring database performance

Extracting, Transforming, and Loading Big Data

- Moving large amounts of data is challenging for several reasons, including
  - Insufficient network throughput for the volume of data
  - The time required to copy large volumes of data
  - The potential for corrupting data during transmission
  - Storing large amounts of data at the source and target
- There are many ETL tools available for data warehouse developers.
- Scaling ETL to Big Data volumes and variety requires attention to factors that are not common to smaller data warehouse implementations.

Extracting, Transforming, and Loading Big Data (cont.)

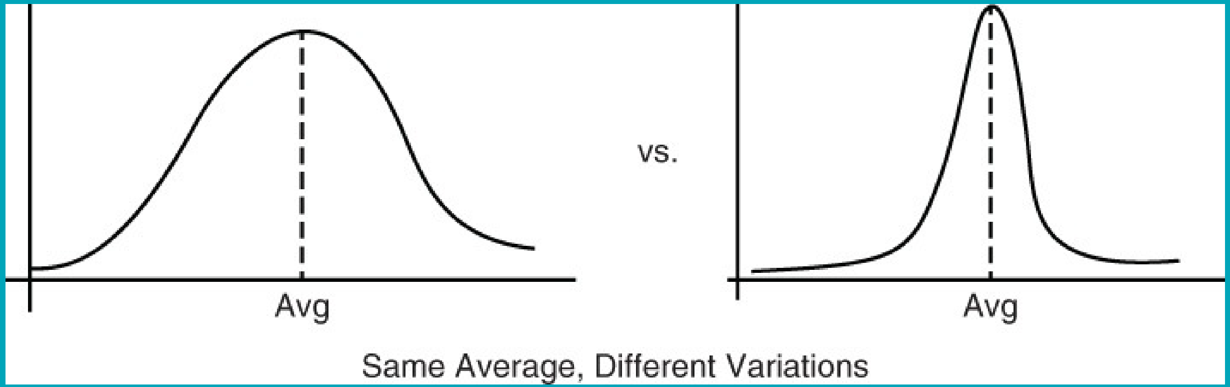
- Examples of ETL tools for Big Data include
  - Apache Flume
    - Apache Flume is designed to move large amounts of log data, but it can be used for other types of data as well.
  - Apache Sqoop
    - Apache Sqoop works with relational databases to move data to and from Big Data sources, such as the Hadoop file system and to the HBase column family database.
    - Sqoop also allows developers to run massively parallel computations in the form of MapReduce jobs.
  - Apache Pig
    - Apache Pig is a data flow language that provides a succinct way to transform data.

Analyzing Big Data

- One of the reasons companies and other organizations collect Big Data is that such data holds potentially valuable insights.
- That is, someone can glean those insights from all the data.
- There are many ways to analyze data, look for patterns, and otherwise extract useful information.
- Two broad disciplines are useful here: statistics and machine learning.

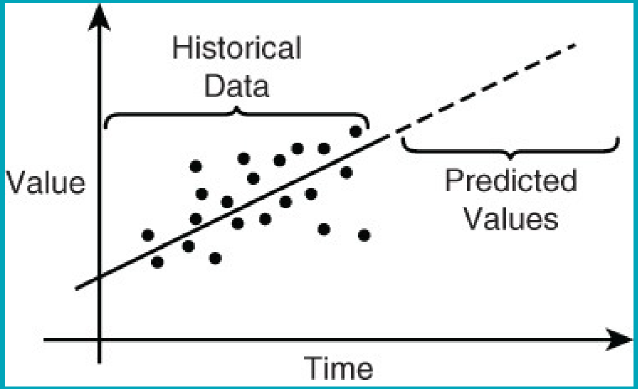
Describing and Predicting with Statistics

- Statistics is the branch of mathematics that studies how to describe large data sets, also known as populations in statistics parlance, and how to make inference from data.
- Descriptive statistics are particularly useful for understanding the characteristics of your data.



Describing and Predicting with Statistics (cont.)

- Predictive, or inferential, statistics is the study of methods for making predictions based on data



Finding Patterns with Machine Learning

- Machine learning is another discipline proving useful for Big Data analysis.
- Machine learning incorporates methods from several disciplines, including computer science, artificial intelligence, statistics, linear algebra, and others.
- Many services might be taken for granted, such as getting personal recommendations based on past purchases, analyzing the sentiment in social media posts, fraud detection, and machine translation, but all depend on machine-learning techniques.

Finding Patterns with Machine Learning (cont.)

- One area of machine learning, called unsupervised learning, is useful for exploring large data sets.
  - A common unsupervised learning method is clustering.
  - Clustering algorithms are helpful when you want to find nonapparent structures or common patterns in data.
- Supervised learning techniques provide the means to learn from examples.

Tools for Analyzing Big Data

- NoSQL database users have the option of using freely available distributed platforms for building their own tools or using available statistics and machine-learning tools.
- Four widely used tools are
  - MapReduce
  - Spark
  - R
  - Mahout
- MapReduce and Spark are distributed platforms.
- R is a widely used statistics package, and Mahout is a machine-learning system designed for Big Data.

MapReduce

- MapReduce is a programming model used for distributed, parallel processing.
- MapReduce programs consist of two primary components: a mapping function and a reducing function.
  - The MapReduce engine applies the mapping function to a set of input data to generate a stream of output values.
  - Those values are then transformed by a reducing function, which often performs aggregating operations, like counting, summing, or averaging the input data.
- The MapReduce model is a core part of the Apache Hadoop project and is widely used for Big Data analysis.

Spark

- Spark is another distributed computational platform.
- Spark was designed by researchers at the University of California, Berkley, as an alternative to MapReduce.
- Both are designed to solve similar types of problems, but they take different approaches.
  - MapReduce writes much data to disk, whereas Spark makes more use of memory.
  - MapReduce employs a fairly rigid computational model (map operation followed by reduce operation), whereas Spark allows for more general computational models.

R

- R is an open source statistics platform.
- The core platform contains modules for many common statistical functions.
- Libraries with additional capabilities are added to the R environment as needed by users.
  - Libraries are available to support machine learning and data mining, specialized disciplines (for example, aquatic sciences), visualization, and specialized statistical methods.
- R did not start out as a tool for Big Data, but at least two libraries are available that support Big Data analysis.

Mahout

- Mahout is an Apache project developing machine-learning tools for Big Data.
- Mahout machine-learning packages were originally written as MapReduce programs, but newer implementations are using Spark.
- Mahout is especially useful for recommendations, classification, and clustering.

Tools for Monitoring Big Data

- One of the primary responsibilities of system administrators is ensuring applications and servers are running as expected.
- When an application runs on a cluster of servers instead of a single server, the system administrator’s job is even more difficult.
- General cluster-monitoring tools and database-specific tools can help with distributed systems management.
- Examples of these tools are
  - Ganglia
  - Hannibal for HBase
  - OpsCenter for Cassandra

Ganglia

- Ganglia is monitoring tool designed for high-performance clusters.
- It is not specific to any one type of database.
- It uses a hierarchical model to represent nodes in a cluster and manage communication between nodes.
- Ganglia is a freely available open source tool.

hannibal

Hannibal for HBase

- Hannibal is an open source monitoring tool for HBase.
- It is especially useful for monitoring and managing regions, which are high-level data structures used in distributing data.
- Hannibal includes visualization tools that allow administrators to quickly assess the current and historical state of data distribution in the cluster.

OpsCenter for Cassandra

- OpsCenter is another open source tool, but it is designed for the Cassandra database.
- OpsCenter gives system administrators a single point of access to information about the state of the cluster and jobs running on the cluster.

Case Study: Customer Data Analysis

- “Different Databases for Different Requirements” introduced TransGlobal Transport and Shipping (TGTS), a fictitious transportation company that coordinates the movement of goods around the globe for businesses of all sizes.
- The following case study applies concepts you learned to show how TGTS can use column family databases to store and analyze large volumes of data about its customers and their shipping practices.

Understanding User Needs

- Analysts at TGTS would like to understand how customer shipping patterns are changing.
- The analysts have several hypotheses about why some customers are shipping more and others are shipping less.
- They would like to have a large data store with a wide range of data, including
  - All shipping orders for all customers since the start of the company
  - All details kept in customer records
  - News articles, industry newsletters, and other text on their customers' industries and markets
  - Historical data about the shipping industry, especially financial databases

Understanding User Needs (cont.)

- The variety and volume of data put this project into the Big Data category, so the development team decides to use a column family database.
- Next, they turn their attention to specific query requirements.

Understanding User Needs (cont.)

- In the first phase of the project, analysts want to apply statistical and machine-learning techniques to get a better sense of the data.
- Questions include:
  - Are there clusters of similar customers or shipping orders?
  - How does the average value of order vary by customer and by time of year the shipment is made?
- They also want to run reports on specific customers and shipping routes.

Understanding User Needs (cont.)

- The queries for these reports are
  - For a particular customer, what orders have been placed?
  - For a particular order, what items were shipped?
  - For a particular route, how many ships used that route in a given time period?
  - For customers in a particular industry, how many shipments were made during a particular period of time?

Understanding User Needs (cont.)

- The database designers have a good sense of the entities that need to be modeled in the first phase of the project.
- The column store database will need tables for
  - Customers
  - Orders
  - Ships
  - Routes

Understanding User Needs (cont.)

- Customers will have a single column family with data about company name, addresses, contacts, industry, and market categories.
- Orders will have details about items shipped, such as names, descriptions, and weights.
- Ships will have details about the capacity, age, maintenance history, and other features of the vessels.
- Routes will store descriptive information about routes as well as geographic details of the route.

Understanding User Needs (cont.)

- In addition to the tables for the four primary entities, the designers will implement tables as indexes for the following:
    - Orders by customer
    - Shipped items by order
    - Ships by route
- The tables as indexes allow for rapid retrieval of data as needed by the queries.
- In addition, because some of the queries make a reference to a period of time, the designers will implement database-managed indexes on data columns.

Lab Session

- Using Advanced Data Types in Apache Cassandra™
  - Learn about universally unique identifiers (UUIDs), collections, tuples, user-defined types (UDTs) and counters

Using Advanced Data Types in Apache Cassandra™

- Universally unique identifier (UUID) data types
- Collection data types
- Tuple data type
- User-defined types (UDTs)
- Counter data type

Using Advanced Data Types in Apache Cassandra™ (cont.)

- **CQL data types**
  - You may have already used many CQL data types in your table definitions.
  - Most of them, including TEXT, VARCHAR, ASCII, TINYINT, SMALLINT, INT, BIGINT, VARINT, FLOAT, DOUBLE, DECIMAL, TIME, TIMESTAMP, DATE, DURATION, BOOLEAN, BLOB, and INET, are easy to understand and use.
  - There are also several "more advanced" CQL data types:
    - Universally unique identifier (UUID) data types: UUID and TIMEUUID
    - Collection data types: SET, LIST and MAP
    - Tuple data type: TUPLE
    - User-defined types (UDTs): CREATE TYPE, ALTER TYPE, DROP TYPE and DESCRIBE TYPE
    - Counter data type: COUNTER

Using Advanced Data Types in Apache Cassandra™ (cont.)

- **Let's get started ...**
  - Start the CQL shell:

```
cqlsh
```

  - Create the keyspace:

```
CREATE KEYSPACE killr_video
WITH REPLICATION = {
  'class' : 'SimpleStrategy',
  'replication_factor' : 1 };
```

  - Set the current working keyspace:

```
USE killr_video;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- **UUIDs**
  - A *universally unique identifier (UUID)* is a 128-bit number that can be automatically generated and used to identify an entity or relationship in a Cassandra database.
  - UUIDs provide an efficient way to assign unique identifiers and help to prevent accidental upserts or eliminate race conditions.

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Cassandra Query Language supports the following two UUID data types:
  - UUID is a *Version 4 UUID* that is randomly generated. To generate a value of type UUID, you can use function uuid()
  - TIMEUUID is a *Version 1 UUID* that is generated based on a MAC address and a timestamp. To generate a value of type TIMEUUID, you can use function now(). When needed, the timestamp component of a TIMEUUID value can be extracted using functions unixTimestampOf() or dateOf(). Moreover, TIMEUUID values in clustering columns are automatically ordered based on their underlying timestamps and can also be retrieved based on the timestamps using functions minTimeuuid(timestamp) and maxTimeuuid(timestamp).

Using Advanced Data Types in Apache Cassandra™ (cont.)

- As an example, let's create table users with partition key id of type UUID and insert two rows:

```
CREATE TABLE users (
  id UUID,
  name TEXT,
  age INT,
  PRIMARY KEY ((id))
);

INSERT INTO users (id, name, age)
VALUES (7902a572-e7dc-4428-b056-0571af415df3, 'Joe', 25);
INSERT INTO users (id, name, age)
VALUES (uuid(), 'Jen', 27);

SELECT * FROM users;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Next, create table movies with partition key id of type UUID and insert the following two rows:

id	title	year	duration
5069cc15-4300-4595-ae77-381c3af5dc5e	Alice in Wonderland	2010	108
uuid()	Alice in Wonderland	1951	75

Using Advanced Data Types in Apache Cassandra™ (cont.)

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Finally, study this more advanced example, where TIMEUUID is used to both guarantee uniqueness and provide a timestamp for each row in table comments_by_user:

```
CREATE TABLE comments_by_user (
  user_id UUID,
  comment_id TIMEUUID,
  movie_id UUID,
  comment TEXT,
  PRIMARY KEY ((user_id), comment_id, movie_id)
);
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

```
INSERT INTO comments_by_user (user_id, comment_id, movie_id,
comment)
VALUES (7902a572-e7dc-4428-b056-0571af415df3,
        63b00000-bfde-11d3-8080-808080808080,
        5069cc15-4300-4595-ae77-381c3af5dc5e,
        'First watched in 2000');
INSERT INTO comments_by_user (user_id, comment_id, movie_id,
comment)
VALUES (7902a572-e7dc-4428-b056-0571af415df3,
        9ab0c000-f668-11de-8080-808080808080,
        5069cc15-4300-4595-ae77-381c3af5dc5e,
        'Watched again in 2010');
INSERT INTO comments_by_user (user_id, comment_id, movie_id,
comment)
VALUES (7902a572-e7dc-4428-b056-0571af415df3,
        now(),
        5069cc15-4300-4595-ae77-381c3af5dc5e,
        'Watched again today');
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

```
SELECT comment_id, dateOf(comment_id) AS date, comment
FROM comments_by_user
WHERE user_id = 7902a572-e7dc-4428-b056-0571af415df3;

SELECT comment_id, dateOf(comment_id) AS date, comment
FROM comments_by_user
WHERE user_id = 7902a572-e7dc-4428-b056-0571af415df3
  AND comment_id > maxTimeuuid('1999-01-01 00:00+0000')
  AND comment_id < minTimeuuid('2019-01-01 00:00+0000')
ORDER BY comment_id DESC;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Sets
  - A *set* is an unordered collection of distinct values. In Cassandra, sets are intended for storing a small number of values of the same type.
  - To define a set data type, Cassandra Query Language provides construct SET<type>, where type can refer to a CQL data type like INT, DATE, UUID and so forth.
  - As an example, alter table movies to add column production of type SET<TEXT>:

```
ALTER TABLE movies ADD production SET<TEXT>;
SELECT title, year, production FROM movies;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Add three production companies for one of the movies:

```
UPDATE movies
SET production = { 'Walt Disney Pictures',
                   'Roth Films' }
WHERE id = 5069cc15-4300-4595-ae77-381c3af5dc5e;
UPDATE movies
SET production = production + { 'Team Todd' }
WHERE id = 5069cc15-4300-4595-ae77-381c3af5dc5e;
SELECT title, year, production FROM movies;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Next, alter table movies to add column genres and add values *Adventure*, *Family* and *Fantasy* for one of the movies:

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Lists
  - A *list* is an ordered collection of values, where the same value may occur more than once. In Cassandra, lists are intended for storing a small number of values of the same type.
  - To define a list data type, Cassandra Query Language provides construct LIST<type>, where type can refer to a CQL data type like INT, DATE, UUID and so forth.
  - As an example, alter table users to add column searches of type LIST<TEXT>:

```
ALTER TABLE users ADD searches LIST<TEXT>;
SELECT id, name, searches FROM users;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Add three latest search leterals for one of the users:

```
UPDATE users
SET searches = [ 'Alice in Wonderland' ]
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;
UPDATE users
SET searches = searches + [ 'Comedy movies' ]
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;
UPDATE users
SET searches = searches + [ 'Alice in Wonderland' ]
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;
SELECT id, name, searches FROM users;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Delete the oldest search literal and add a new one:

```
DELETE searches[0] FROM users
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;
UPDATE users
SET searches = searches + [ 'New releases' ]
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;
SELECT id, name, searches FROM users;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Next, alter table users to add column emails and add two email addresses for one of the users:

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Maps
  - A *map* is a collection of key-value pairs, where each pair has a unique key. In Cassandra, maps are intended for storing a small number of key-value pairs of the same type.
  - To define a map data type, Cassandra Query Language provides construct MAP<type1,type2>, where type1 and type2 can refer to same or different CQL data types, including INT, DATE, UUID and so forth.
  - As an example, alter table users to add column sessions of type MAP<TIMEUUID,INT>:

```
ALTER TABLE users ADD sessions MAP<TIMEUUID,INT>;
SELECT name, sessions FROM users;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Add two sessions with TIMEUUID keys and INT duration values for one of the users:

```
UPDATE users
SET sessions = { now(): 32,
    e22deb70-b65f-11ea-9aac-99396fc4f757: 7 }
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;
SELECT name, sessions FROM users;
```

- Update a duration value for one of the sessions:

```
UPDATE users
SET sessions[e22deb70-b65f-11ea-9aac-99396fc4f757] = 9
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;
SELECT name, sessions FROM users;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Next, alter table users to add column preferences and add key-value pairs (*color-scheme*: *dark*) and (*quality*: *auto*) for one of the users:

Using Advanced Data Types in Apache Cassandra™ (cont.)

- **Nested collections**
  - It is also possible to define collection data types that contain nested collections, such as *list of maps* or *set of sets of sets*.
  - A nested collection definition has to be designated as FROZEN, which means that a nested collection is stored as a single blob value and manipulated as a whole.
  - In other words, when an individual element of a frozen collection needs to be updated, the entire collection must be overwritten.
  - As a result, nested collections are generally less efficient unless they hold immutable or rarely changing data.

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Let's alter table movies again to be able to store movie casts and crews in column crew of type MAP<TEXT,FROZEN<LIST<TEXT>>>:

```
ALTER TABLE movies
ADD crew MAP<TEXT,FROZEN<LIST<TEXT>>>;
SELECT title, year, crew FROM movies;
```

- Add a movie crew:

```
UPDATE movies
SET crew = {
  'cast': ['Johnny Depp', 'Mia Wasikowska'],
  'directed by': ['Tim Burton']
 }
WHERE id = 5069cc15-4300-4595-ae77-381c3af5dc5e;
SELECT title, year, crew FROM movies;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- **Tuples**
  - A *tuple* is a fixed-length list, where values can be of different types.
  - To define a tuple data type, Cassandra Query Language provides construct TUPLE<type1,type2,..., typeN>, where type1 , type2 , ... , typeN can refer to same or different CQL data types, including INT, DATE, UUID and so forth.
  - Here is an example of a TUPLE column:

```
ALTER TABLE users ADD full_name TUPLE<TEXT,TEXT,TEXT>;

UPDATE users
SET full_name = ('Joe', 'The', 'Great')
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;

SELECT name, full_name FROM users;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- **UDTs**
  - A *user-defined type (UDT)* is a custom data type composed of one or more named and typed fields.
  - UDT fields can be of simple types, collection types or even other UDTs.
  - Cassandra Query Language provides statements CREATE TYPE, ALTER TYPE, DROP TYPE and DESCRIBE TYPE to work with UDTs.
  - Let's define UDT ADDRESS with four fields:

```
CREATE TYPE ADDRESS (
    street TEXT,
    city TEXT,
    state TEXT,
    postal_code TEXT
);
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Alter table users to add column address of type ADDRESS:

```
ALTER TABLE users ADD address ADDRESS;
SELECT name, address FROM users;
```

- Add an address for one of the users:

```
UPDATE users
SET address = { street: '1100 Congress Ave',
                city: 'Austin',
                state: 'Texas',
                postal_code: '78701' }
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;
SELECT name, address FROM users
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

```
UPDATE users
SET address.state = 'TX'
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;
SELECT name,
       address.street      AS street,
       address.city        AS city,
       address.state       AS state,
       address.postal_code AS zip
FROM users
WHERE id = 7902a572-e7dc-4428-b056-0571af415df3;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Next, alter table users to add column previous_addresses and add at least two previous addresses for one of the users:

Using Advanced Data Types in Apache Cassandra™ (cont.)

Using Advanced Data Types in Apache Cassandra™ (cont.)

- **Counters**
  - A *distributed counter* is a 64-bit signed integer, whose value can be efficiently modified by concurrent transactions without causing *race conditions*.
  - Counters are useful for keeping track of various statistics by counting events or adding up integer values. To define a counter column, Cassandra Query Language provides data type COUNTER.

Using Advanced Data Types in Apache Cassandra™ (cont.)

- There are a number of restrictions on how counters can be used in Cassandra:
  - A counter cannot be set or reset. It can only be incremented or decremented.
  - A counter value does not exist until the first increment or decrement operation is performed. The first operation assumes the initial counter value of 0.
  - A table with one or more counter columns cannot have non-counter columns other than primary key columns. Counter columns cannot be primary key columns.

Using Advanced Data Types in Apache Cassandra™ (cont.)

- As an example, let's create a new table to keep track of movie rating statistics:

```
CREATE TABLE movie_stats (
  id UUID,
  num_ratings COUNTER,
  sum_ratings COUNTER,
  PRIMARY KEY ((id))
);
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Update the counters to account for two ratings of 7 and 9 for the same movie:

```
UPDATE movie_stats
SET num_ratings = num_ratings + 1,
    sum_ratings = sum_ratings + 7
WHERE id = 5069cc15-4300-4595-ae77-381c3af5dc5e;
UPDATE movie_stats
SET num_ratings = num_ratings + 1,
    sum_ratings = sum_ratings + 9
WHERE id = 5069cc15-4300-4595-ae77-381c3af5dc5e;
SELECT * FROM movie_stats;
```

Using Advanced Data Types in Apache Cassandra™ (cont.)

- Next, alter table movie_stats to add another COUNTER column to keep track of how many times each movie was watched and increment the counter value three times for one of the movies:

```
ALTER TABLE movie_stats ADD num_views COUNTER;

UPDATE movie_stats
SET num_views = num_views + 1
WHERE id = 5069cc15-4300-4595-ae77-381c3af5dc5e;
UPDATE movie_stats
SET num_views = num_views + 1
WHERE id = 5069cc15-4300-4595-ae77-381c3af5dc5e;
UPDATE movie_stats
SET num_views = num_views + 1
WHERE id = 5069cc15-4300-4595-ae77-381c3af5dc5e;

SELECT * FROM movie_stats;
```

Q & A

Additional Slides

- SSTable-Attached Indexing (SASI) allow you create one or multiple secondary indexes on the same database table, with each SASI index based on any column.
- Exception: there is no need to define an SASI index based on a single-column partition key.

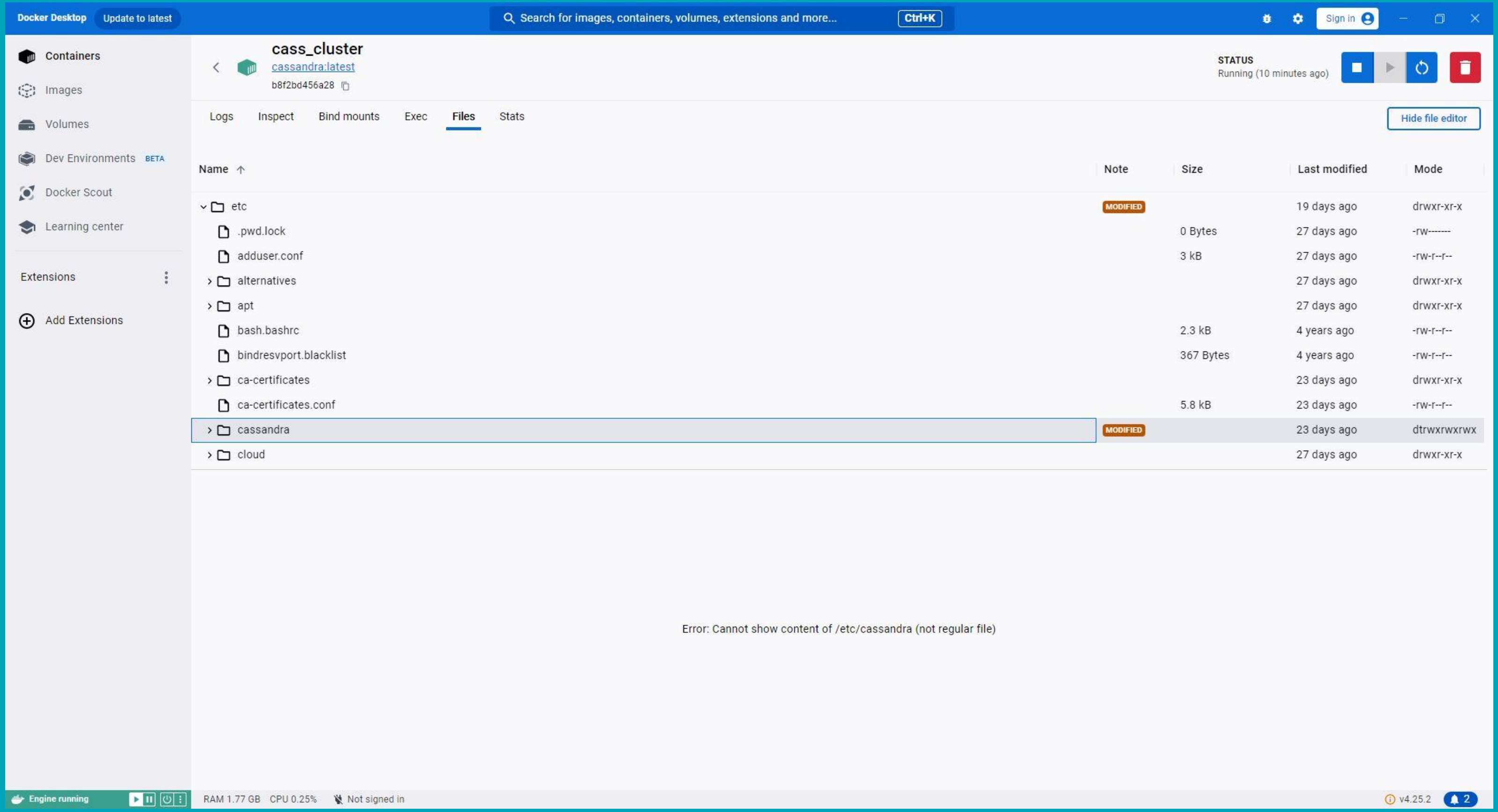
Configure SASI indexes

- SASI is an experimental indexing feature that is not recommended for production use.
- SASI is disabled by default in the cassandra.yaml file.
- To enable SASI, set the sasi_indexes_enabled parameter to true:

```
sasi_indexes_enabled: true
```

Enabling SASI Index

- Modify the casssandra.yaml

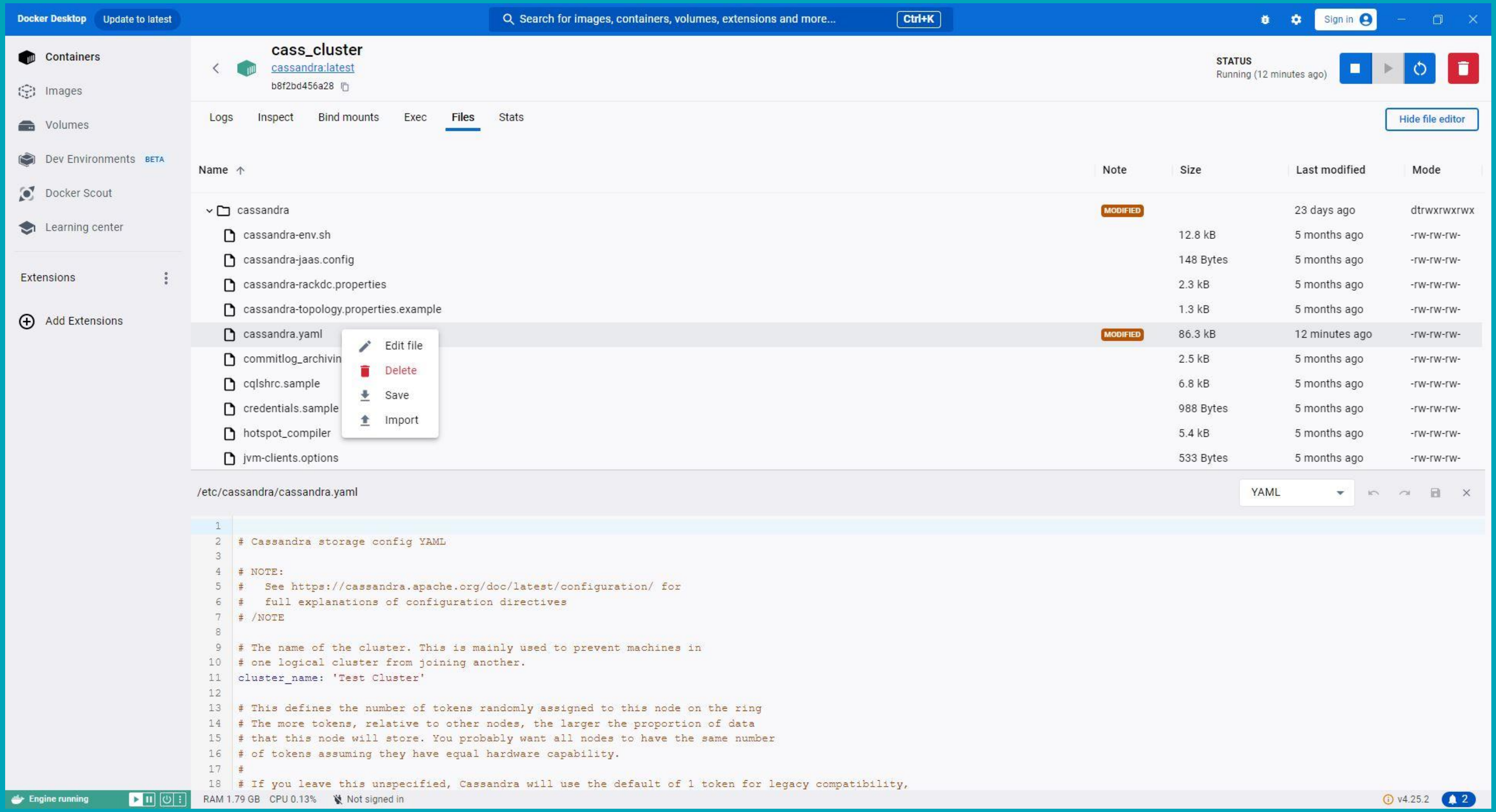


Name	Note	Size	Last modified	Mode
etc	MODIFIED		19 days ago	drwxr-xr-x
.pwd.lock		0 Bytes	27 days ago	-rw-------
adduser.conf		3 kB	27 days ago	-rw-r--r--
alternatives			27 days ago	drwxr-xr-x
apt			27 days ago	drwxr-xr-x
bash.bashrc		2.3 kB	4 years ago	-rw-r--r--
bindresvport.blacklist		367 Bytes	4 years ago	-rw-r--r--
ca-certificates			23 days ago	drwxr-xr-x
ca-certificates.conf		5.8 kB	23 days ago	-rw-r--r--
cassandra	MODIFIED		23 days ago	dtrwxrwxrwx
cloud			27 days ago	drwxr-xr-x

```
Error: Cannot show content of /etc/cassandra (not regular file)
```

Enabling SASI Index (cont.)

- /etc/cassandra/cassandra.yaml



Name	Note	Size	Last modified	Mode
cassandra	MODIFIED		23 days ago	dtrwxrwxrwx
cassandra-env.sh		12.8 kB	5 months ago	-rw-rw-rw-
cassandra-jaas.config		148 Bytes	5 months ago	-rw-rw-rw-
cassandra-rackdc.properties		2.3 kB	5 months ago	-rw-rw-rw-
cassandra-topology.properties.example		1.3 kB	5 months ago	-rw-rw-rw-
cassandra.yaml	MODIFIED	86.3 kB	12 minutes ago	-rw-rw-rw-
commitlog_archivin		2.5 kB	5 months ago	-rw-rw-rw-
cqlshrc.sample		6.8 kB	5 months ago	-rw-rw-rw-
credentials.sample		988 Bytes	5 months ago	-rw-rw-rw-
hotspot_compiler		5.4 kB	5 months ago	-rw-rw-rw-
jvm-clients.options		533 Bytes	5 months ago	-rw-rw-rw-

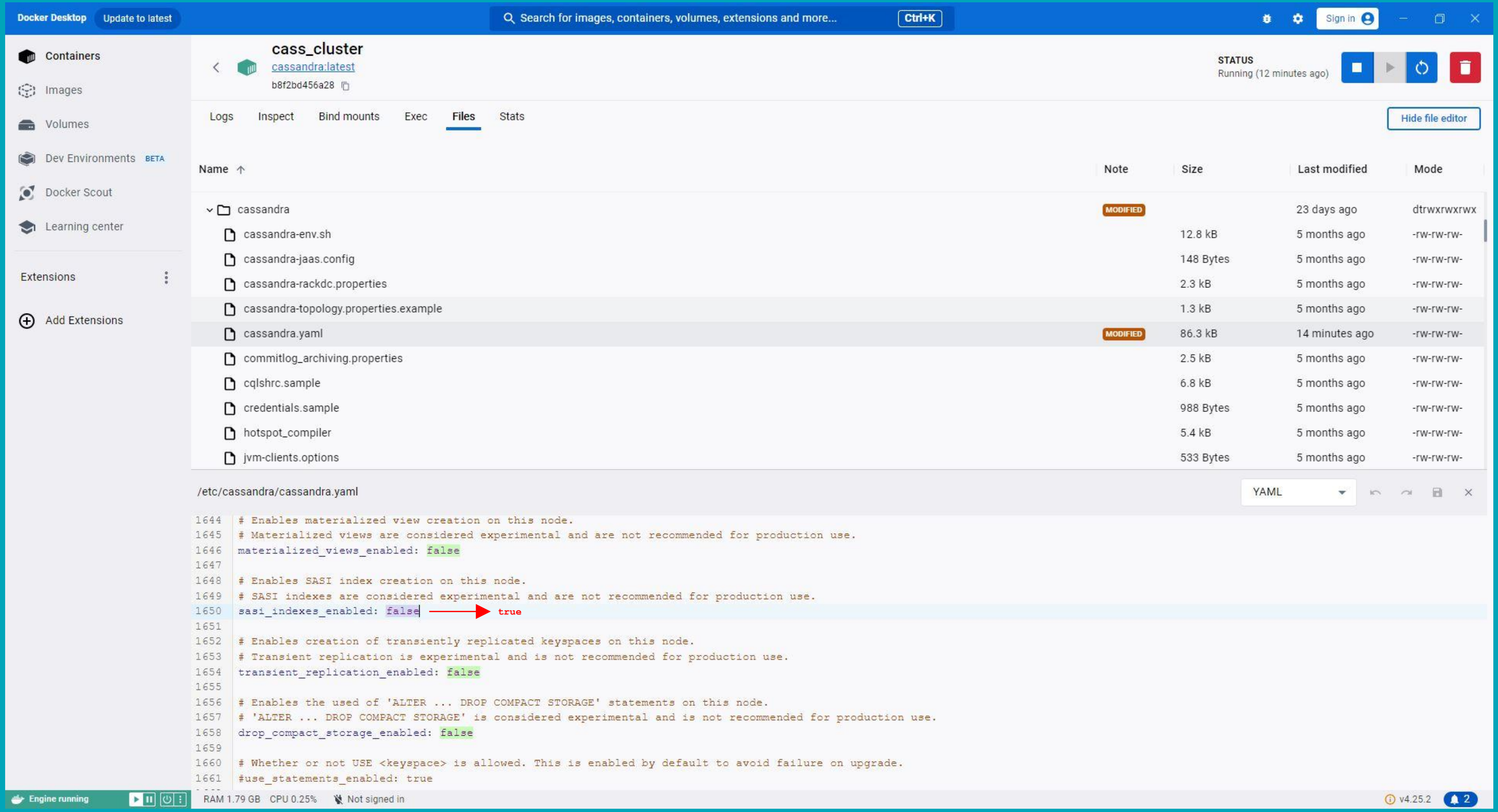
/etc/cassandra/cassandra.yaml

```yaml

# Cassandra storage config YAML

# NOTE:
#   See https://cassandra.apache.org/doc/latest/configuration/ for
#   full explanations of configuration directives
# /NOTE

# The name of the cluster. This is mainly used to prevent machines in
# one logical cluster from joining another.
cluster_name: 'Test Cluster'

# This defines the number of tokens randomly assigned to this node on the ring
# The more tokens, relative to other nodes, the larger the proportion of data
# that this node will store. You probably want all nodes to have the same number
# of tokens assuming they have equal hardware capability.
#
# If you leave this unspecified, Cassandra will use the default of 1 token for legacy compatibility,
```

RAM 1.79 GB CPU 0.13% Not signed in

Enabling SASI Index (cont.)

- Scroll down to the end of the file and enable the SASI index



Docker Desktop | Update to latest | Search for images, containers, volumes, extensions and more... Ctrl+K | Sign in

Containers | Images | Volumes | Dev Environments BETA | Docker Scout | Learning center | Extensions | Add Extensions

cass_cluster
cassandra:latest
b8f2bd456a28

STATUS
Running (12 minutes ago)

Logs | Inspect | Bind mounts | Exec | Files | Stats | Hide file editor

Name	Note	Size	Last modified	Mode
cassandra	MODIFIED		23 days ago	dtrwxrwxrwx
cassandra-env.sh		12.8 kB	5 months ago	-rw-rw-rw-
cassandra-jaas.config		148 Bytes	5 months ago	-rw-rw-rw-
cassandra-rackdc.properties		2.3 kB	5 months ago	-rw-rw-rw-
cassandra-topology.properties.example		1.3 kB	5 months ago	-rw-rw-rw-
cassandra.yaml	MODIFIED	86.3 kB	14 minutes ago	-rw-rw-rw-
commitlog_archiving.properties		2.5 kB	5 months ago	-rw-rw-rw-
cqlshrc.sample		6.8 kB	5 months ago	-rw-rw-rw-
credentials.sample		988 Bytes	5 months ago	-rw-rw-rw-
hotspot_compiler		5.4 kB	5 months ago	-rw-rw-rw-
jvm-clients.options		533 Bytes	5 months ago	-rw-rw-rw-

/etc/cassandra/cassandra.yaml | YAML

```yaml
# Enables materialized view creation on this node.
# Materialized views are considered experimental and are not recommended for production use.
materialized_views_enabled: false

# Enables SASI index creation on this node.
# SASI indexes are considered experimental and are not recommended for production use.
sasi_indexes_enabled: false  → true

# Enables creation of transiently replicated keyspaces on this node.
# Transient replication is experimental and is not recommended for production use.
transient_replication_enabled: false

# Enables the used of 'ALTER ... DROP COMPACT STORAGE' statements on this node.
# 'ALTER ... DROP COMPACT STORAGE' is considered experimental and is not recommended for production use.
drop_compact_storage_enabled: false

# Whether or not USE <keyspace> is allowed. This is enabled by default to avoid failure on upgrade.
#use_statements_enabled: true
```

Engine running | RAM 1.79 GB CPU 0.25% | Not signed in | v4.25.2