Chapter 48.

December 2020

Welcome to the December 2020 edition of DataStax Developer's Notebook (DDN). This month we answer the following question(s);

> Enjoyed the past article on Apache Cassandra and virtualization (VMs). I didn't see you detail how to recover from a failed node (VM) though. Can you help ?
>
> *Excellent question ! Good catch; an oversight on our part. Is this edition of DDN we detail how to implement and test node recovery from failure, when using virtual machines.*

Software versions

The primary DataStax software component used in this edition of DDN is DataStax Enterprise (DSE), currently release 6.9.1, or DataStax Astra (Apache Cassandra version 4.0.0.682), as required. All of the steps outlined below can be run on one laptop with 16 GB of RAM, or if you prefer, run these steps on Amazon Web Services (AWS), Microsoft Azure, or similar, to allow yourself a bit more resource.

For isolation and (simplicity), we develop and test all systems inside virtual machines using a hypervisor (Oracle Virtual Box, VMWare Fusion version 8.5, or similar). The guest operating system we use is Ubuntu Desktop version 18.04, 64 bit.

48.1 Terms and core concepts

As stated above, ultimately the end goal is implement Apache Cassandra node recovery, when using virtual machines.

In this last edition of this article series we,

- Produced a primer to operating virtual machines (VMs) on GCP; how to get started, how to program and manage, other.
- And we detailed a number of scripts we use when benchmarking or similar in this area using Apache Cassandra.
- But, we did not detail recovery from node failure, when using VMs.

Of course, if the Apache Cassandra node fails, it may come up. If the node will never come up, there are options:

- The recommended best practice to storage within Apache Cassandra is to use a replication factor equal to 3 (RF=3), on any keyspaces.

  If you have to bootstrap a net new Cassandra node (a node that starts with zero data), that is built into the Apache Cassandra server. The net new node will copy whatever data it needs from any existing nodes; such is the purpose of having 3 copies of all data.

  In more recent releases, Cassandra uses 'zero copy streaming', which is largely an optimization over functionality that Cassandra has always had. In effect, serializing and deserializing copies of data from one node to a recovering node is now much more highly optimized.

  How long (time) does it take to recover a node in this manner ? Sorry; your mileage may vary.
- But, there are other options too.

  Of course there are backup and recovery topics, and even Cassandra SSTable restore options.

  This document moves forward discussing what can happen, when you are using virtual machines and using network attached (versus locally attached) storage.

  When using network attached storage, the data is still available, even if the Cassandra node proper is not.

Note: Apache Cassandra version 2.x and 3.x documentation pages list using network attached storage (NAS) as an anti-pattern, we expect, because of the lesser performance NAS likely provides.

Okay, that’s likely fair.

However, what you might give up in performance can be offset in greater operator and programmer productivity.

How would you know where the data is recovered from

As stated above, a net new Apache Cassandra node will automatically recover data from the other, surviving nodes. This is an automatic and built in feature to Cassandra. If you have network attached storage to any failed node, you need not perform this (restoration); the data already exists.

But how would you prove where the data was (restored) from ?

Easy; for testing purposes, don’t give Apache Cassandra 3 copies of data, from which to copy.

Example 48-1 below details how to make a no-copy Apache Cassandra keyspace. (E.g., replication factor equal to one.) A code review follows.

Example 48-1   RF=1 keyspace and data

```
DROP KEYSPACE IF EXISTS ks_losenode1;
   //
   //  Drop keyspace with a node down yields,
   //  OperationTimedOut: errors={'Connection defunct by
heartbeat': 'Client request timeout. See
Session.execute[_async](timeout)'}, last_host=10.128.15.205:9042

   CREATE KEYSPACE ks_losenode1
      WITH replication = {'class': 'SimpleStrategy',
'replication_factor': '1'};
   USE ks_losenode1;
   CREATE TABLE t1
      (
      col1 TEXT PRIMARY KEY,
      col2 TEXT,
      col3 TEXT,
      col4 TEXT
      );
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('111', '111', '111', '111');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('222', '222', '222', '222');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('333', '333', '333', '333');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('444', '444', '444', '444');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('555', '555', '555', '555');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('666', '666', '666', '666');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('777', '777', '777', '777');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('888', '888', '888', '888');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('999', '999', '999', '999');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('000', '000', '000', '000');
   SELECT COUNT(*) FROM t1;
```

Relative to Example 48-1, the following is offered:

- To allow this script to re-run over iterative tests, we start with a drop keyspace. Per the commented error message, you can not drop a keyspace that is experiencing a down node.
- The create keyspaces is created with a simple strategy, and a replication factor equal to one. This means we will only have one copy of data stored in this keyspace; bad for production, good for this test.
- We add 10 rows of data, and we select count to verify.

- – By default, Apache Cassandra will flush contents from memory every 10 seconds. If we were automating these tests, we might execute a nodetool flush after these inserts.

Example 48-2 details examples when using an RF=1 keyspace, when nodes are down. A code review follows.

Example 48-2 Using an RF=1 keyspace, when nodes are down

```
USE ks_losenode1;
   SELECT COUNT(*) FROM t1;
   //  NoHostAvailable:

   CONSISTENCY ANY;
   SELECT COUNT(*) FROM t1;
   //  InvalidRequest: Error from server: code=2200 [Invalid
query] message="ANY ConsistencyLevel is only supported for
writes"

   CONSISTENCY ONE;
   SELECT COUNT(*) FROM t1;
   //  NoHostAvailable:

   CONSISTENCY ANY;
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('aaa', 'aaa', 'aaa', 'aaa');
   //  works

   CONSISTENCY ONE;
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('bbb', 'bbb', 'bbb', 'bbb');
   //  works

   CONSISTENCY TWO;
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('ccc', 'ccc', 'ccc', 'ccc');
   //  NoHostAvailable:
```

```
   //
   //  This makes sense

   CONSISTENCY THREE;
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('ddd', 'ddd', 'ddd', 'ddd');
   //  NoHostAvailable:
   //
   //  This makes sense


   Information only; Repeat the above with a higher level
replication factor
   #####################################################

   //  One node still down

   /opt/dse_68/node1/bin/cqlsh `head -1
/opt/92_IntIP_addresses.txt`

   DROP KEYSPACE IF EXISTS ks_losenode2;
   CREATE KEYSPACE ks_losenode2
      WITH replication = {'class': 'SimpleStrategy',
'replication_factor': '2'};
   //  OperationTimedOut: errors={'Connection defunct by
heartbeat': 'Client request timeout. See
Session.execute[_async](timeout)'}, last_host=10.128.15.205:9042


   Information only; Repeat the above with a higher level
replication factor
   #####################################################

   //  Restore the down node (boot DSE, we had killed it)
   /opt/dse_68/node1/bin/cqlsh `head -1
/opt/92_IntIP_addresses.txt`

   USE ks_losenode1;
   SELECT COUNT(*) FROM t1;
   //  accurate

   DROP KEYSPACE IF EXISTS ks_losenode2;
   CREATE KEYSPACE ks_losenode2
      WITH replication = {'class': 'SimpleStrategy',
'replication_factor': '2'};
   USE ks_losenode2;
   CREATE TABLE t1
      (
      col1 TEXT PRIMARY KEY,
      col2 TEXT,
      col3 TEXT,
      col4 TEXT
      );
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('111', '111', '111', '111');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('222', '222', '222', '222');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('333', '333', '333', '333');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('444', '444', '444', '444');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('555', '555', '555', '555');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('666', '666', '666', '666');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('777', '777', '777', '777');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('888', '888', '888', '888');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('999', '999', '999', '999');
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('000', '000', '000', '000');
   SELECT COUNT(*) FROM t1;

   //  Then kill the node again

   SELECT COUNT(*) FROM t1;
   // works

   CONSISTENCY ANY;
   SELECT COUNT(*) FROM t1;
   //  InvalidRequest: Error from server: code=2200 [Invalid
query] message="ANY ConsistencyLevel is only supported for
writes"

   CONSISTENCY ONE;
   SELECT COUNT(*) FROM t1;
   //  works

   CONSISTENCY TWO;
   SELECT COUNT(*) FROM t1;
   //  NoHostAvailable:

   CONSISTENCY ANY;
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('aaa', 'aaa', 'aaa', 'aaa');
   //  works

   CONSISTENCY ONE;
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('bbb', 'bbb', 'bbb', 'bbb');
   //  works

   CONSISTENCY TWO;
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('ccc', 'ccc', 'ccc', 'ccc');
   //  NoHostAvailable:

   CONSISTENCY THREE;
   INSERT INTO t1 (col1, col2, col3, col4)
      VALUES ('ddd', 'ddd', 'ddd', 'ddd');
   //  NoHostAvailable:
```

Relative to Example 48-2, the following is offered:

- Given a down node, on a replication factor equal to one keyspace, a select count will fail; the accuracy of the results can not be guaranteed.
- Changing the consistency to any, and a subsequent read will fail; any is a setting for writes.
- Setting the consistency to one and a read will fail; again, can not guarantee the accuracy of data that needs to be returned.

Note: Why doesn't Apache Cassandra support partial, or possibly incorrect results when a node is down ?

Well, consider that would be an extreme edge case for Cassandra. Cassandra easily, seamlessly offer multiple writable copies of all data.

- With a consistency ANY, or one, writes are generally accepted. When you ask for the higher level consistency guarantees, then the write may not be achieved, because there are not enough nodes (by count) available.
- Calling to drop a keyspace still fails, when nodes are down.
- And then we move to the selects and similar to confirm that data is present as expected, once any down node has been restored.

How to replace the data to any downed node

So again, this (behavior, ability) is built in and automatic to Apache Cassandra. However, is you are using network attached storage, you can perform a manual set of steps that might allow the downed node to come up even more quickly.

Why we detail this at all-

- For quality-assurance, and development, folks use these procedures for 'cluster cloning' and 'differentiated data'.
  - An (Apache Cassandra) is what it sounds like; you create and load a Cassandra cluster with gobs of complex data (this takes time), and you wish to create copies quickly, to best support application unit and system tests.
  - Or, nearly the same, you wish to (reset) data to its original state after unit or system tests.
  - Both of the use cases above can be achieved quickly using these same basic techniques.
- And, with network attached storage, you may wish to consider a manual procedure to bring a node up in minutes, versus many-many minutes.

These topics are well covered in a number of articles, listed here for reference,

```
https://thelastpickle.com/blog/2018/02/21/replace-node-without
-bootstrapping.html
```

```
https://docs.datastax.com/en/dse/6.8/dse-admin/datastax_enterp
rise/operations/opsChangeIp.html
```

Generally, to test, other;

- Make 4 (count) VMs. Copy the Apache Cassandra binaries to each.
- Boot Cassandra using only 3 nodes.
- Load data. To be certain where your data is being (restored) from, use a replication factor equal to one keyspace.
- Flush the node you are about to kill.

To simplify our test below, we can just copy the data out of the node we are about to fail/kill. Comments;

- You only need the 'data_file_directories' files from the Cassandra cassandra.yaml configuration file; nothing else.
- For this, a simple test, SSH, SCP, and similar are fine for copying these files.

– Ps, nodetool flush|drain currently behave differently on Apache Cassandra, versus DataStax Enterprise. Regardless, you want to flush, then kill the node.

Procedure to (restore) the down node

Consider that on some level, the node 'is' its data; move the data, and you moved the node. Only the IP address of the node will change.

Why do we mention this ?

You will not be decommissioning or similar the failed node. We're just moving the node.

After making data on the node to be failed, copying that data to the net new node, and killing the node to be failed, complete the following:

– Correct any 'seed nodes' IP addresses on every node's cassandra.yaml.

– Correct the cassandra-topology.properties file entries; again, IP addresses are changing for the failed, now net new node.

– Add this line to the bottom of the cassandra-env.sh file, on the net new node,

```
JVM_OPTS=$JVM_OPTS:
-Dcassandra.replace_address_first_boot=10.128.15.218
```

Where 10.* is the address of the net new Cassandra node.

– Boot the net new node.

You are successful when all data, including that in a replication factor equal to one keyspace, is present.

Success as viewed from the Cassandra message log file is below,

```
tail  /opt2/system.log    #  relevant lines kept


            ...
         INFO  [main] 2020-11-20 22:04:53,760
StorageService.java:622 - Populating token metadata from system
tables
         INFO  [main] 2020-11-20 22:04:53,770
StorageService.java:629 - Token metadata: Normal Tokens:
         /10.128.0.38:7000:[6531134697375635165]
         /10.128.0.39:7000:[1428815533260344076]
         /10.128.0.41:7000:[93997132151849619]
            ...
         INFO  [main] 2020-11-20 22:04:54,352
InboundConnectionInitiator.java:128 - Listening on address:
(/10.128.0.41:7000), nic: ens4, encryption: disabled
         INFO  [Messaging-EventLoop-3-6] 2020-11-20 22:04:54,480
OutboundConnection.java:1151 -

/10.128.0.41:7000(/10.128.0.41:56534)->/10.128.0.39:7000-URGENT_
MESSAGES-22dd8b5b
            successfully connected, version = 12, framing = CRC,
encryption = disabled
         INFO  [Messaging-EventLoop-3-3] 2020-11-20 22:04:54,480
OutboundConnection.java:1151 -

/10.128.0.41:7000(/10.128.0.41:38630)->/10.128.0.38:7000-URGENT_
MESSAGES-ec6f287f
            successfully connected, version = 12, framing = CRC,
encryption = disabled
               ...
         INFO  [main] 2020-11-20 22:04:55,547
StorageService.java:2581 - Node /10.128.0.41:7000 state jump to
NORMAL
            ...
         INFO  [GossipStage:1] 2020-11-20 22:04:56,544
Gossiper.java:1206 - InetAddress /10.128.0.39:7000 is now UP
         INFO  [GossipStage:1] 2020-11-20 22:04:56,544
Gossiper.java:1206 - InetAddress /10.128.0.38:7000 is now UP
         INFO  [GossipStage:1] 2020-11-20 22:04:56,545
Gossiper.java:1206 - InetAddress /10.128.0.38:7000 is now UP
         INFO  [GossipStage:1] 2020-11-20 22:04:56,599
Gossiper.java:1206 - InetAddress /10.128.0.39:7000 is now UP

         # *****************
         WARN  [GossipStage:1] 2020-11-20 22:04:56,774
StorageService.java:2620 - Not updating host ID
b099ed9e-bb67-4fb8-9db2-ec87a9c05764 for /10.128.0.40:7000
because it's mine
         INFO  [GossipStage:1] 2020-11-20 22:04:56,774
StorageService.java:2552 - Nodes () and /10.128.0.40:7000 have
the same token /10.128.0.41:7000.  Ignoring 93997132151849619
         # *****************

            INFO  [GossipStage:1] 2020-11-20 22:04:56,855
  Gossiper.java:1224 - InetAddress /10.128.0.40:7000 is now DOWN
```

Above, note that the net new Cassandra node correctly takes ownership of its assigned partition key tokens.

48.2 Complete the following

At this point in this document we have covered the steps to replace a node, using accessible network attached storage data files. We've detailed the necessary Cassandra configuration file settings, and procedures.

You know you are successful when you can recover data for which there was only the one, original copy.

48.3 In this document, we reviewed or created:

This month and in this document we detailed the following:

- How to recover a Cassandra node that uses network attached storage, more quickly.

Persons who help this month.

Kiyu Gabriel, Dave Bechberger, and Jim Hatcher.

Additional resources:

Free DataStax Enterprise training courses,

```
https://academy.datastax.com/courses/
```

Take any class, any time, for free. If you complete every class on DataStax Academy, you will actually have achieved a pretty good mastery of DataStax Enterprise, Apache Spark, Apache Solr, Apache TinkerPop, and even some programming.

This document is located here,

```
https://github.com/farrell0/DataStax-Developers-Notebook
https://tinyurl.com/ddn3000
```