COHESITY

Cassandra Data Protection Best Practices and Recommendations

Cohesity Solution for Cassandra Database Backup and Restore

ABSTRACT

This guide provides an overview of Cohesity's backup and recovery solution to protect Cassandra databases. The guide also offers prerequisites, best practices, and recommendations to help implement the Cohesity Cassandra DataProtect solution.

Table of Contents

Figures

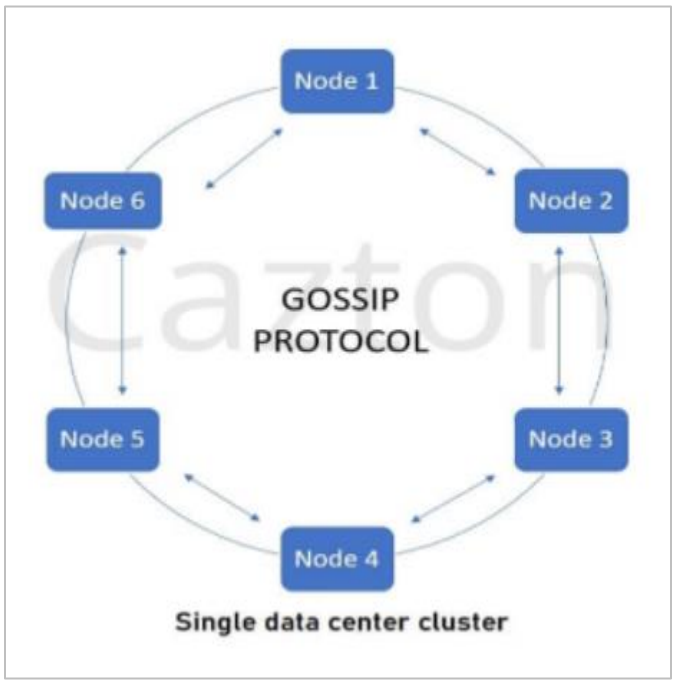
Tables

Introduction

The Cohesity Cassandra DataProtect solution natively integrates with Apache Cassandra and DataStax Enterprise to provide consistent backup and recovery for clustered Cassandra deployments.

Cassandra Overview and Architecture

Figure 1: Cassandra Architecture



- Cassandra is a distributed NoSQL column store database with multiple nodes creating a cluster. A cluster can span multiple DCs.
- Cassandra is designed to have no master or slave nodes.
- It has a ring-type architecture, that is, its nodes are logically distributed like a ring.
- Data is automatically distributed across all the nodes.
- A client application can connect to any node and read or write data.

Cassandra Data Model

- Data in Cassandra is stored in keyspaces and tables.
- Logical Construct: Each cluster can host multiple keyspaces and each keyspace can host multiple tables.
- Physical Construct: On disk, data is stored as **SSTables** (.db files). SSTable is immutable.
- When a write occurs, Cassandra stores the data in a memory structure called **memtable**.
- The memtable stores and writes in sorted order until reaching a configurable limit, then flushed to SSTable**.**
- Since every SSTable is immutable, it will always make a new SSTable each time it flushes.
- Compaction is a process that will automatically merge the tables into a single new table.
- The default Cassandra data directory structure: /cassandra/data///la-1-big-Data.db.

Cassandra Snapshots

- A snapshot of Cassandra when taken, creates hard links of all live SSTables (.db files). The tool provided by Cassandra to perform snapshots is known as “nodetool”.
- The nodetool can take snapshots at keyspace and table level.
- The nodetool is designed to take snapshots on one node at a time.
- For a multi-node Cassandra cluster, nodetool needs to be run in parallel to snapshot the entire cluster.
- Snapshots (Hard links) are stored in the following directory structure: /cassandra/data///snapshots/.

Plan and Prepare

To back up Cassandra databases, you need to register it as a source on the Cohesity Platform. Before registration, ensure you meet the following prerequisites, considerations, and minimum permissions to successfully deploy the Cassandra DataProtect solution.

Supported Versions

Cassandra DataProtect Solution supports the following enterprise versions:

Table 1: Supported Versions

APPLICATION	SUPPORTED VERSIONS	
	APACHE	DATASTAX ENTERPRISE
Cassandra	4.0	N/A
	3.11	6.8.1
	3.10	6.7
	3.0	5.1.10
	2.1.11	5.1.9
	2.1.8	5.1.8
	2.1	5.1.7
	N/A	5.1.2
	N/A	5.1.0

NOTE: Users whose existing Cassandra setups match any of the following criteria must edit their Cassandra source and go through the registration process.

- Cassandra cluster set up with different public and private IP addresses (rpc_address/broadcast_rpc_address and different listen_address)
- Cassandra setup with different listen_address and broadcast_rpc_address(listen_address = address1, rpc_address = 0.0.0.0, broadcast_rpc_address = address2)

Prerequisites

Ensure the following prerequisites are met before backing up Cassandra databases:

- The NoSQL and Hadoop Service on the Cohesity cluster is in Running status. See NoSQL and Hadoop Service for details.
- Refer to the NoSQL and Hadoop Service - Post Installation Procedures to get a general idea about the following operations:
  - Manage SSL-enabled NoSQL primary clusters
  - Work with Kerberos Security enabled primaries for data protection
- Linux User on Cassandra Primary
- SSL Enabled Cassandra Clusters
  - Steps Required to Support SSL-Enabled Cassandra Clusters
  - Obtain SSL Certificates in Cassandra Nodes and Import them on the Cohesity cluster that are Running the NoSQL and Hadoop Services
  - Import NoSQL and Hadoop Services Certificates
- Setting the Number of Open Files Limit for Cassandra
- Enabling JMX

Considerations

- For users who have their Cassandra cluster set up with **public address = rpc_address/broadcast_rpc_address** and **private address = listen_address**, for DSE below 6.x (5.x and below), a bulk restore will not work if the private IPs for the Cassandra nodes are not accessible by the Cohesity cluster (SCP restore will work as expected).
- Users must refresh a data source when the configuration settings of the data source are modified. For example, if changes are made to cassandra.yaml, cassandra-env.sh, or dse.yaml file of a data source, then the data source must be re-discovered and re-verified.

- Backup is not supported for the following system keyspaces:
  - system
  - system_auth
  - system_distributed
  - system_schema
  - system_traces
  - dse_leases
  - dse_perf
  - dse_security
  - dse_system
    - solr_admin
    - dse_insights_local
    - dse_insights
    - dse_system_local
    - system_virtual_schema
    - system_views
    - system_backups
- Restore of system keyspaces such as OpsCenter is only supported for the same cluster from which it was backed up.
- Backup and recovery of tables containing semicolon (';') in the column name is not supported.
- Data recovery from Cassandra 2.x to 3.x is unsupported if the backed-up tables contain dropped columns.
- For Cassandra 2.x or DSE 4.x, backup recovery is not supported for tables where all columns are part of the Primary Key.
- The appending suffix is not supported while restoring graph objects. This is because individual graph keyspaces cannot be renamed. This issue occurs during the jobtag recovery process when the global suffix is applied to all keyspaces thus wrongly renaming the '_system' keyspace for a graph. For example, we need to back up two types of keyspaces for a graph. If the name of the graph is ‘exampleGraph’, then we should back up ‘exampleGraph’ and ‘exampleGraph_system’. However, during jobtag recovery, if the global suffix is applied to all keyspaces it wrongly renames the keyspaces 'exampleGraph_suffix' and 'exampleGraph_system_suffix' for a graph.
- Support for non-default store_type, algorithm, and protocol in cassandra.yaml is not available in this release.
- **For DataStax Enterprise 6.x, the following considerations are applicable:
  - Some of the new features of DSE 6.7 such as NodeSync are not supported.
  - To restore DSE 5.x backups to DSE 6.x cluster, the minimum version must be DSE 6.0.6.
  - If a job gets killed or the primary Cassandra node goes down during a restore, the temporary directories on the primary nodes are not deleted. In such cases, the user has to manually delete the temporary directories on the primary Cassandra nodes.
    The directories are created in each of the storage directories on the primary nodes and are named by the restore job's UUID.

- In DSE 6.x, restoring a Cassandra table where the primary key contains UDT would fail if the corresponding keyspace object had been renamed.
- Apache Cassandra 4.0 does not support the experimental features.
- 3 Data restore from Cassandra 3.x to 4.0 is not supported.

Minimum Permissions

Authentication Enabled Cassandra Cluster

In the case of an authentication-enabled Cassandra cluster, the specified Cassandra user must be a super user.

Discovery:

- The ssh user should have read/execute access to the Cassandra config directory.

Backup:

- The ssh user should have read/execute access to Cassandra data directories.

Restore:

For optimized recovery:

- The ssh user must have read/write/execute access to Cassandra data directories.
- The ssh user must have privileges to change the file ownership to the Cassandra user.

Additionally, for DSE 6.x and restore with staging dir option selection, the ssh user should have permission to execute sstableloader cmd on the Cassandra nodes. This means that sstableloader cmd should be in the PATH of the ssh user, and the ssh user must have the write access to the following directories:

- commitlog directory
- cdc_raw directory (if this dir is missing, an empty dir must be created.)
- data directory
- saved_caches directory

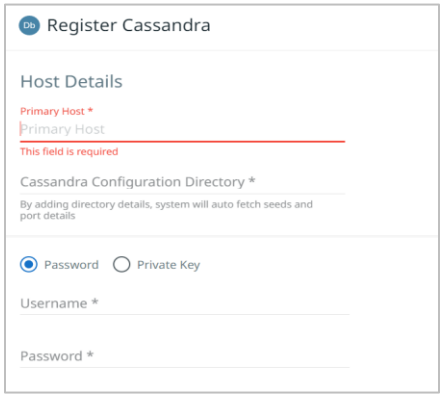
Write access to these directories is a DSE requirement since the sstableloader tool checks for the same.

How Cohesity Protects Cassandra

Cohesity integrates with Cassandra without any agents to discover databases, and tables, and perform backups and recovery. The first step is to register and discover Cassandra topology, which is covered under the registration workflow.

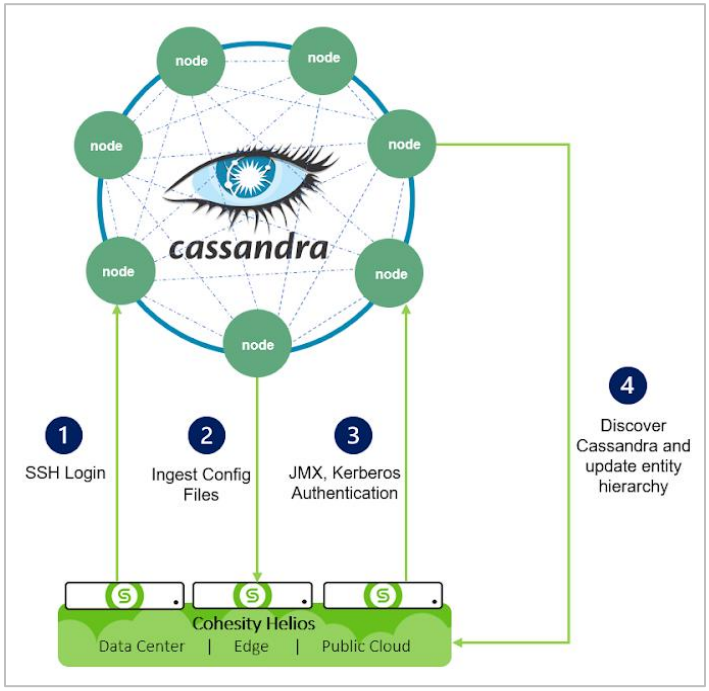
Registration Workflow

1. During the registration workflow, the user must provide Cassandra **Primary Host** to connect to, along with configuration directories and SSH credentials of this **Primary Host** (Password or Private Key) on the registration form as shown below:



2. Once the user enters these details and clicks **Register**, it triggers the following workflow:

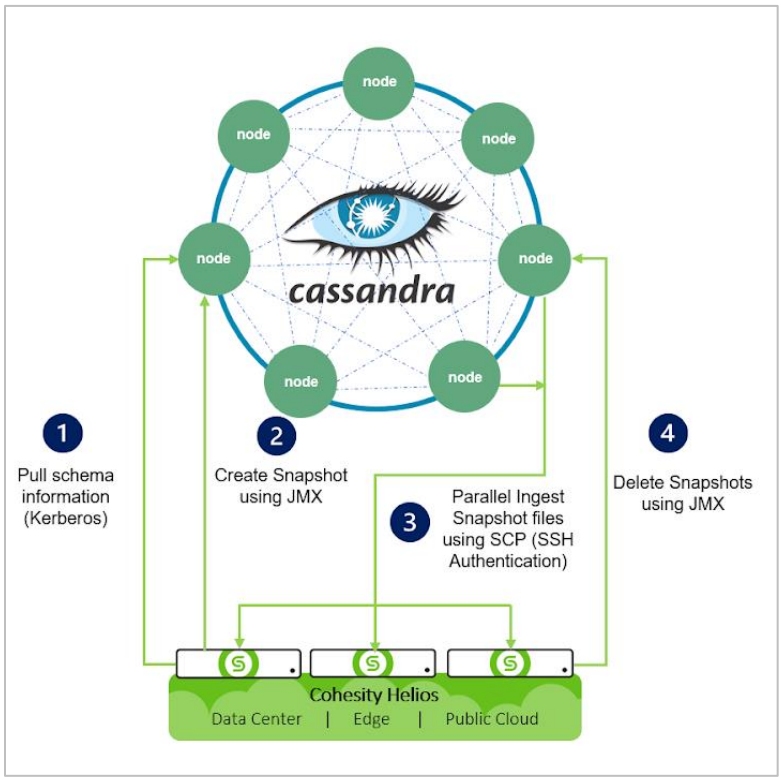
Figure 2: Registration Workflow



3. SSH connection is made to Cassandra primary node using the name/ip and credentials provided. The primary node can be any node in the Cassandra cluster.
4. Config files cassandra.yaml and cassandra-env.sh files are ingested into the Cohesity cluster and examined for authentication requirements (JMX, Kerberos).
5. The user has to provide JMX credential and Kerberos principal name to be used as required by Cassandra on the registration form and they are used to make a successful connection to browse databases and tables on Cassandra source.
6. These databases and tables are updated in Cohesity's entity hierarchy which is then used for protection workflow.

Protection Workflow

Figure 3: Protection Workflow



As shown in the figure above, the following happens under the hood when a backup is triggered:

1. Cohesity pulls schema information from Cassandra primary using Kerberos authentication.
2. Depending on the granularity of protection configured, a snapshot is triggered on selected tables and DB's using JMX. This uses JMX authentication. Since Cohesity is aware of the complete topology as part of the discovery process, it allows Cohesity to map keyspaces and tables to Cassandra nodes that host them. Cohesity executes parallel snapshots on each node that hosts the keyspaces and tables to be protected.

3. Snapshot files (hard links) are batched into **streams** based on the size and number of files to be backed up. Stream count can be configured at a protection job level. The default value is 16. These streams are then ingested parallelly into Cohesity SpanFS across multiple Cohesity nodes using SCP utility. SCP uses SSH authentication.
4. Once the backup is completed, Cohesity connects back using JMX authentication and deletes the snapshot that was created in Step 2.

After all the steps are completed, the backup process is complete.

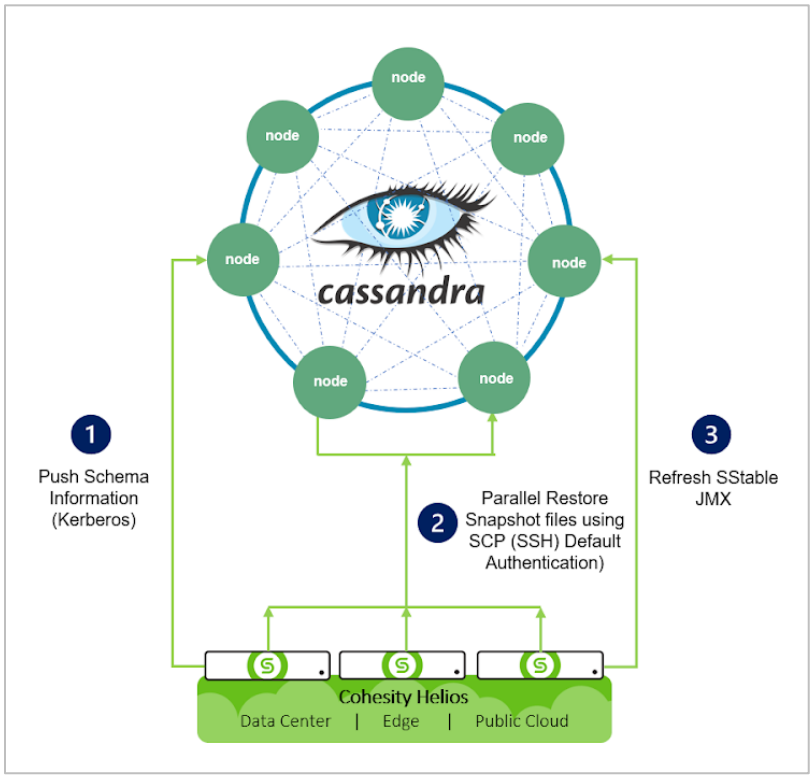
Recovery Workflow

Recovery workflow is dependent on certain factors. Each scenario is detailed below and handled transparently by the code.

Default Approach

The default approach uses SCP-based restore and is detailed as follows. This approach requires that Cassandra version is DSE 6.x or above.

Figure 4: Default Approach



As shown in the figure above, if recovery is being performed to DSE 6.x or above Cassandra with no limitations on SCP (SSH authentication has **the** appropriate privilege to perform write operations).

Once the recovery is initiated, the following are the steps involved:

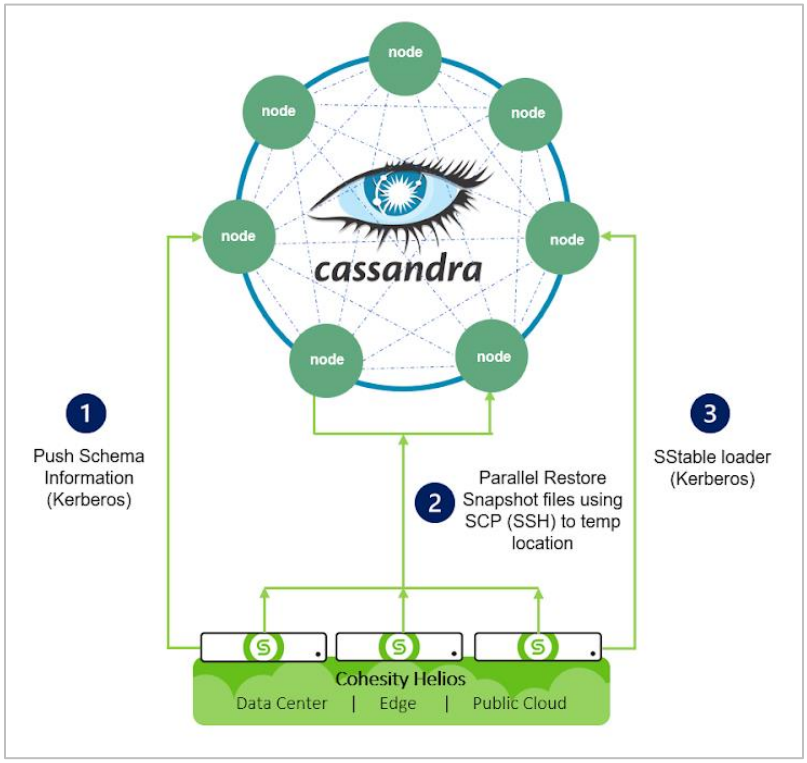
1. Schema information is pushed using Kerberos authentication.
2. Required files are copied in parallel using SCP - uses SSH Authentication.
3. Once done, Cohesity issues a refresh SStable call using JMX authentication.

Once these steps are completed successfully, restore is marked complete.

Failback - Approach 1

The failback approach described here is applicable when the primary being restored to is running DSE 6.x or above and has limitations with SSH authentication (No write privilege to Cassandra filesystem).

Figure 5: Failback-Approach 1



Once the recovery is initiated, the following are the steps involved:

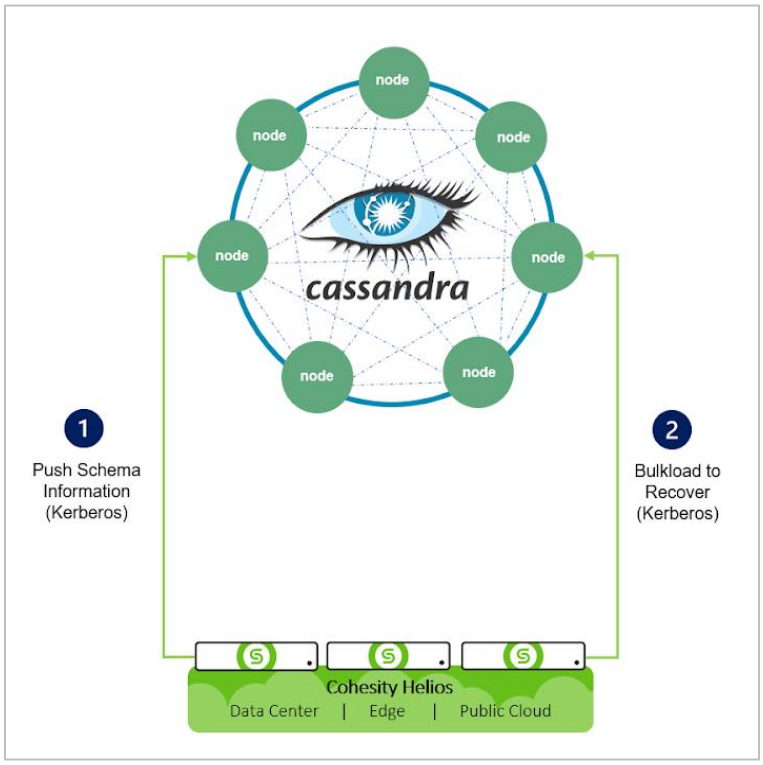
1. Schema information is pushed using Kerberos authentication.
2. Required files are copied in parallel using SCP to a Temporary location. This temporary location can be provided in the Cohesity recovery workflow form - which uses SSH Authentication.
3. Once done, Cohesity uses Kerberos authentication to run SStableloader function to complete the restore.

Once these steps are completed successfully, restore is marked complete.

Failback - Approach 2

The failback approach described here is applicable when the primary being restored is running Open-source Apache Cassandra or Pre DSE 6.x as the SCP-based recovery method is not feasible for these versions of Cassandra.

Figure 6: Failback - Approach 2



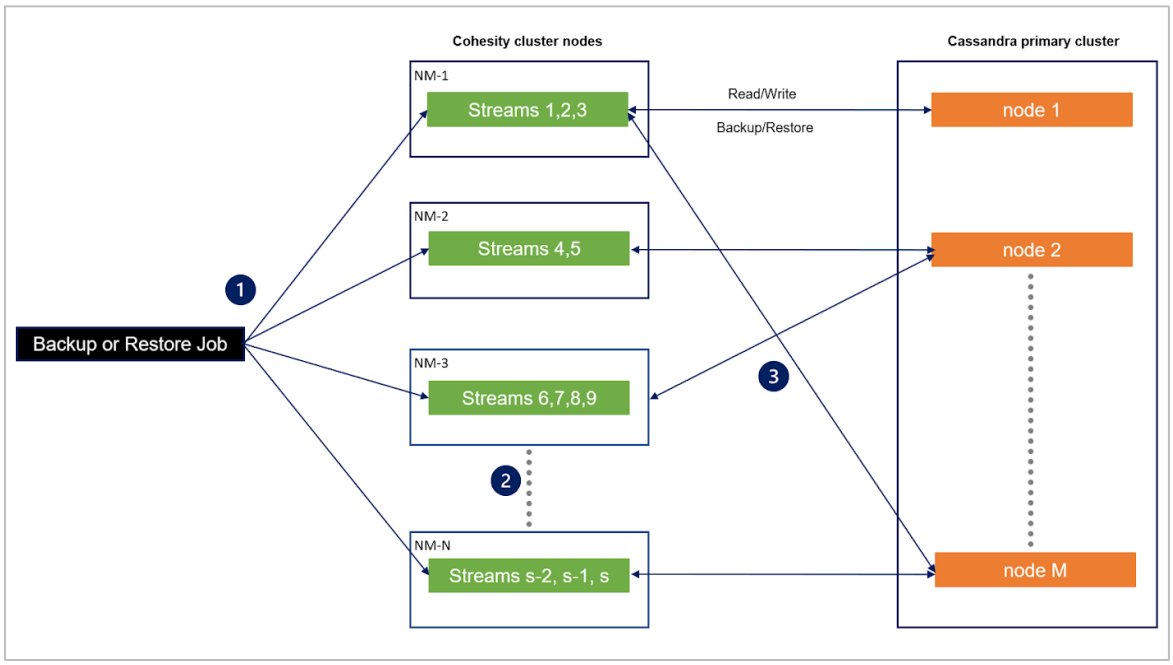
Once the recovery is initiated, follow the steps below:

1. Schema information is pushed using Kerberos authentication.
2. Cohesity uses Kerberos authentication and runs bulkload protocol through Cassandra code running in Cohesity to perform recovery.

Once these steps are completed successfully, restore is marked as complete.

Multi-Node Multi-Stream Backup and Recovery

Figure 7: Multi-Node Multi-Stream Backup and Recovery



As shown in the figure above,

1. A backup/restore job is triggered.
2. A backup/restore job is split into ‘s’ number of splits based on the size and number of files to be backed up.
3. Each NM (Node Manager) could be reading/writing data to multiple primary Cassandra nodes at any point. Multiple NM’s could be connected to the same primary node as well in some cases.

- A minimum of 3 node managers are required, however, it can scale based on the sizing of the cluster.
- The node managers can be small, medium, or large based on the sizing of the cluster (Available CPU and Memory). This is derived through sizing exercise.
- A small node manager can handle 10 streams.
- The nodemanager size and count define no. of streams Cohesity can handle along with parallelism across nodes. It does not have any relation to the performance of a single stream.

Point-in-time Recovery (PITR)

Point In Time Recovery (PITR) enables you to have aggressive Recovery Point Objectives (RPO) for Cassandra source. PITR provides more granular recovery to scenarios of accidental data error or corruption. For PITR, commit logs on the Cassandra cluster are backed up which helps you to select a recovery point on a time scale.

NOTE: You must apply the **6.8.1-p1** patch on the Cohesity cluster to be able to use the Cassandra PITR functionality for a replication job on the Cohesity cluster.

Prerequisites

- On the Cassandra cluster:
- You must enable commit log backup for successful point-in-time recovery.
- You must ensure that the location to which the commit logs are restored must have sufficient disk space for new logs.
- Cassandra service must be restarted on all nodes after enabling Commit Log Archival for backup to copy the commit logs.
- You must run one full backup after enabling point in time on the Cassandra cluster.
- The SSH user must have read and write access to the /backup location.
- The backup location must be the same for all nodes on the Cassandra cluster where PITR is enabled.
- For PITR, the Cassandra cluster must be registered on one Cohesity cluster only.

Configure Commit Log Archival

For PITR, the archival of commit logs must be enabled on the Cassandra cluster.

You need to configure the commit log archival on the Cassandra cluster in the commitlog_archiving.properties configuration file. The commit log is put to the archive location at node startup and when a commit log is written onto disk, or at a specific point in time.

The command archive_command expects only a single command with arguments. The parameters must be entered verbatim. STDOUT and STDIN or multiple commands cannot be executed. To execute multiple commands, you need to create a script with multiple commands and then call the script from the configuration file.

Enable Commit Log Archival

You must specify the parameters in the archive_command to enable commit log archival.

The following is a sample command:

archive_command=/bin/ln %path /backup/%name where

- %path is the fully qualified path of the segment to archive.
- /backup is the location where you want to archive the commit log.
- %name is the name of the commit log.

NOTE:

- The SSH user must have read and write access to the /backup location.
- The backup location must be the same for all nodes on the Cassandra cluster where PITR is enabled.

Disable Commit Log Archival

To disable commit log archival, you must reset the archive_command as archive_command=(must be set to empty).

Points to Note:

- The modification on the commitlog_archiving.properties configuration file must be done on all the Cassandra nodes.
- The Cassandra process must be restarted after editing the configuration file.
- Commit logs are archived frequently because all writes to Cassandra are part of commit logs and this may consume more disk space. The disk on which the archive location is mounted must have adequate disk space.
- For improved performance, Cohesity recommends having the Cassandra directory and commit log directory on separate disks.

Configure Log Replay

You need to configure the restore_command, restore_directories, and restore_point_in_time commands on the Cassandra cluster.

The command restore_command expects only a single command with arguments. The parameters must be entered verbatim. STDOUT and STDIN or multiple commands cannot be executed. To execute multiple commands, you need to create a script with multiple commands and then call the script from the configuration file.

Enable Restore of Archived Commit Log

You must specify the parameters in the restore_command to enable the restoration of the archived commit log.

The following is a sample command: restore_command=/bin/cp -f %from %to where

- %from is the fully qualified path of the archived commit log segment from the restore_directories.
- %to is the name of the live commit log directory.

You must also specify the parameters in the restore_directories command.

The following is a sample command:

```
restore_directories=/restore_directory
```

You must also specify the parameters in the restore_point_in_time command to restore to a point-in-time. Cassandra replays all the logs in the restore directory till the specified time.

The command accepts the time in the following format represented in GMT:

```
YYYY::MM::DD HH:MM:SS
```

The following is a sample command:

```
restore_point_in_time=2013:12:11 17:00:00
```

Disable Restore of Archived Commit Log

To disable the restore of commit log archival, the restore_command, restore_directories, and restore_point_in_time commands must be set to empty.

Backup

A point-in-time backup is achieved by backing up the data and the commit logs of the Cassandra cluster. Data can be backed up at a lesser frequency than the logs. In Cassandra, commit logs are generated for the entire node and cannot be filtered based on keyspaces or tables. Therefore, the commit log backup and data backups cannot be tied together based on keyspace or table. You need to create two Protection Groups, one for data backup (keyspace and table) and the other for commit log backup. For more information, see Backup.

Perform the following steps to create a Cassandra commit log backup:

- Configure the commit log archival in the Cassandra cluster. For more information, see Configure Commit Log Archival.

  NOTE: The SSH user must have read and write permissions to the archive directory.

- Specify the archive log location while registering the Cassandra cluster as a source. For more information, see Register.

- Create a protection policy that has the commit log backup details. You can also use an existing policy that has the commit log backup details. For more information, see Create or Edit a Standard Policy.
- Create a Protection Group to back up the commit log. For more information, see Backup.

  NOTE: Cohesity removes the backed-up commit logs from the /backup location.

Recover

Point-in-time recovery is achieved by copying both the data and commit logs on the Cassandra primary. Data restore is a copy of the ss tables from Cohesity. Commit log restore includes both copying and replaying the logs.

For more information, see Recover.

Considerations

- Commit logs can be replayed only to the original Cassandra cluster and with the same schema.
- Archival of Log Backups is not supported.
- For PITR, object renaming is not supported.
- DDL changes are not restored during log replay.
- You cannot back up specific keyspace or table commit logs for point-in-time recovery. The commit log backup is for the entire Cassandra cluster.
- You must have performed at least one protection run after the first commit log backup for successful point-in-time recovery.
- For recovery from a remote cluster (archival or replication), you must specify the target cluster in the protection run policy, data backup, and commit log backup.
- Cohesity allows only one backup of the commit logs per source.
- You cannot recover objects using a Log Backup Protection Group. For the recovery of objects, you must create a regular backup Protection Group.
- For PITR:
  - Do not register Cassandra on multiple Cohesity clusters.
  - Do not register the same Cassandra cluster with different names or IPs on a Cohesity cluster.

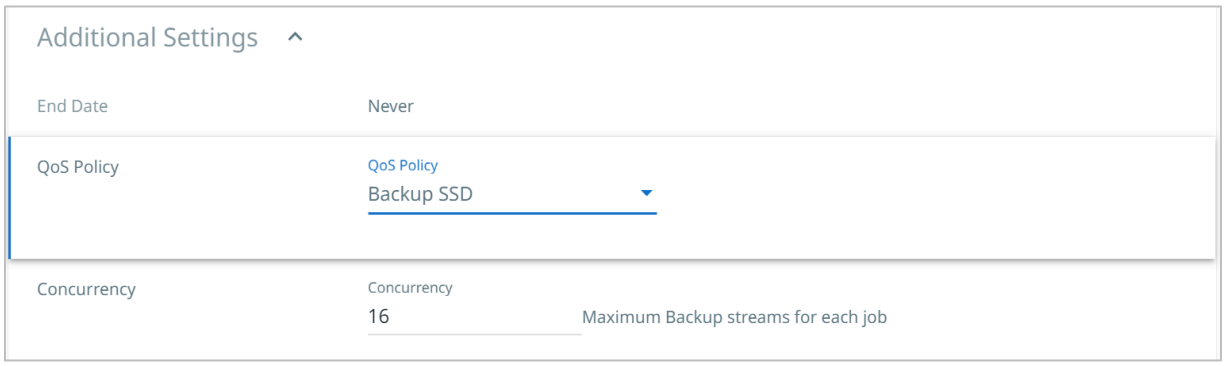
Best Practices and Tunables

Sizing

- Ensure the Cohesity cluster is sized properly and has gone through a sizing exercise.
- Sizing ensures there are enough resources available for Cohesity to run required nodemanager PODS to support a given change rate and meet required SLAs.

Cohesity Tunable

- QoS policy on protection jobs can be configured to use SSD for faster ingestion.
- No. of backup streams can be tuned on the protection job (Default is 16). Screenshot below. Note that tuning this to a higher value would result in more resource utilization on the Cassandra cluster.



- Recommendation for protection job to objects (Keyspace, Tables) mapping:

  Cohesity's Cassandra backup architecture allows backing up all keyspaces and tables on a particular source to be configured in one or multiple groups. The process transparently distributes load to available node managers to stream data into Cohesity. However, one should be careful as a large dataset on a single group could lead to the next schedule missing.

  A general recommendation from Cohesity would be to create a new Protection Group every 100 keyspaces or 20 TB of protected keyspaces.

- Recommendation for job schedule staggering:

  Staggering should be considered when backing up the same source through multiple jobs. For example, there are 200 keyspaces to be backed up from the same source. As recommended one can create 2 jobs to back up 100 keyspaces each. Since these are going to be backed up from the same source, care should be taken to stagger the schedule so that they do not start at the same time. This helps avoid load on the source.

  NOTE: Megafile-based Cassandra backups and restores.

Upgrade Your Disaster Recovery Preparedness

Disaster recovery (DR) and business continuity are closely related plans designed to protect a business's infrastructure and data proactively. Taking Cassandra backups is only one part of protecting your business; you must also protect the data from corruption and catastrophic disaster. You can achieve this by keeping a series of backups, replicating those backups off-site, and archiving them under a long-term retention plan.

Once your backups are stored within a Cohesity cluster, Cohesity gives you the foundation to build a DR plan to protect your business:

- **Capture and Store**. Protect your Cassandra databases from loss and corruption with regularly scheduled backups.
- **Geo-Redundancy**. Replicate your Cassandra backups to an off-site location to protect them from catastrophic loss and disaster.
- **Cost-Effective Archival**. Archive your Cassandra backups to the cloud and store them on lower-cost storage tiers for long-term retention.

Use a Protection Group to schedule regular backups of your database and assign a Protection Policy to include archiving and replicating those backups for long-term retention and disaster recovery.

Your Feedback

Was this document helpful? Send us your feedback!

About the Authors

Scott Lorenz is a Staff Solutions Engineer at Cohesity. Scott focuses on business-critical databases, applications, cloud storage, and enterprise data protection in his role. Scott has over 26 years of experience as an enterprise DBA.

Other essential contributors include:

- John Tidd is a Senior Product Solutions Engineer.

Document Version History

VERSION	DATE	DOCUMENT HISTORY
1.2	Sep 2024	Content updates
1.0	Jan 2023	First release

ABOUT COHESITY

Cohesity is a leader in AI-powered data security and management. Aided by an extensive ecosystem of partners, Cohesity makes it easier to protect, manage, and get value from data – across the data center, edge, and cloud. Cohesity helps organizations defend against cybersecurity threats with comprehensive data security and management capabilities, including immutable backup snapshots, AI-based threat detection, monitoring for malicious behavior, and rapid recovery at scale. Cohesity solutions are delivered as a service, self-managed, or provided by a Cohesity-powered partner. Cohesity is headquartered in San Jose, CA, and is trusted by the world's largest enterprises, including six of the Fortune 10 and 44 of the Fortune 100.

Visit our website and blog, follow us on Twitter and LinkedIn and like us on Facebook.