Cloudera Runtime 7.1.7

Backing up and recovering Apache Kudu

Date published: 2020-11-04
Date modified: 2021-08-05

https://docs.cloudera.com/

Legal Notice

Contents

Kudu backup

Kudu supports both full and incremental table backups through a job implemented using Apache Spark.

Kudu backup and restore jobs use Apache Spark. Therefore, ensure that you install Apache Spark in your environment. To download Apache Spark, see the *Apache Spark documentation*. You can also review the *Submitting Spark applications* topics.

Related Information

Submitting Spark applications

Back up tables

You can use the KuduBackup Spark job to backup one or more Kudu tables.g

When you first run the job for a table, a full backup is run. Additional runs will perform incremental backups which will only contain the rows that have changed since the initial full backup. A new set of full backups can be forced at anytime by passing the --forceFull flag to the backup job.

Following are some of the common flags that you can use while taking a backup:

- --kuduMasterAddresses: This is used to specify a comma-separated addresses of Kudu masters. The default value is localhost.
- --rootPath: The root path is used to output backup data. It accepts any Spark-compatible path.

Note: You can see the full list of the job options by passing the --help flag.

The following code run a KuduBackup job which backs up the tables foo and bar to an HDFS directory:

```
spark-submit --class org.apache.kudu.backup.KuduBackup [***FULL PATH TO kudu
-backup2_2.11-1.12.0.jar***] \
  --kuduMasterAddresses [***KUDU MASTER HOSTNAME 1***]:7051,[***KUDU MASTER
 HOSTNAME 2***]:7051 \
  --rootPath hdfs:///[***DIRECTORY TO USE FOR BACKUP***] \
   impala::[***DATABASE NAME***].foo impala::[***DATABASE NAME***].bar
```

Backup tools

An additional kudu-backup-tools JAR is available to provide some backup exploration and garbage collection capabilities. This jar does not use Spark directly, but instead only requires the Hadoop classpath to run.

Commands:

- list: Lists the backups in the rootPath
- clean: Cleans up old backed up data in the rootPath

Note: You can see the full list of the job options by passing the --help flag.

Following is an example execution which prints the command options:

```
java -cp $(hadoop classpath):kudu-backup-tools-1.12.0.jar org.apache.kudu.ba
ckup.KuduBackupCLI --help
```

Generate a table list

To generate a list of tables to backup using the kudu table list tool along with grep can be useful.

Following is an example that generates a list of all tables that start with my_db.:

```
kudu table list <master_addresses> | grep "^my_db\.*" | tr '\n' ' '
```

Note: This list could be saved as a part of your backup process so that it can be used while restoring.

Backup directory structure

The backup directory structure in the rootPath is considered an internal detail and could change in future versions of Kudu. Additionally, the format and content of the data and metadata files is meant for the backup and restore process only and could change in future versions of Kudu. That said, understanding the structure of the backup rootPath and how it is used can be useful when working with Kudu backups.

The backup directory structure in the rootPath is as follows:

```
/<rootPath>/<tableId>-<tableName>/<backup-id>/
   .kudu-metadata.json
   part-*.<format>
```

- rootPath: Can be used to distinguish separate backup groups, jobs, or concerns
- tableId: The unique internal ID of the table being backed up
- tableName: The name of the table being backed up
  - Note: Table names are URL encoded to prevent pathing issues
- backup-id: A way to uniquely identify/group the data for a single backup run
- .kudu-metadata.json: Contains all of the metadata to support recreating the table, linking backups by time, and handling data format changes

  Written last so that failed backups will not have a metadata file and will not be considered at restore time or backup linking time.
- part-*.<format>: The data files containing the tables data.
  - Currently 1 part file per Kudu partition
  - Incremental backups contain an additional “RowAction” byte column at the end
  - Currently the only supported format/suffix is parquet

Physical backups of an entire node

Kudu does not provide a built-in physical backup and restore functionality yet. However, it is possible to create a physical backup of a Kudu node (either tablet server or master) and restore it later.

Note:

The node to be backed up must be offline during the procedure, or else the backed up (or restored) data will be inconsistent.

Certain aspects of the Kudu node (such as its hostname) are embedded in the on-disk data. As such, it’s not yet possible to restore a physical backup of a node onto another machine.

Procedure

1. Stop all Kudu processes in the cluster. This prevents the tablets on the backed up node from being rereplicated elsewhere unnecessarily.
2. If creating a backup, make a copy of the WAL, metadata, and data directories on each node to be backed up. It is important that this copy preserve all file attributes as well as sparseness.
3. If restoring from a backup, delete the existing WAL, metadata, and data directories, then restore the backup via move or copy. As with creating a backup, it is important that the restore preserve all file attributes and sparseness.
4. Start all Kudu processes in the cluster.

Kudu recovery

Kudu supports restoring tables from full and incremental backups through a restore job implemented using Apache Spark.

Kudu backup and restore jobs use Apache Spark. Therefore, ensure that you install Apache Spark in your environment. To download Apache Spark, see the *Apache Spark documentation*. You can also review the *Submitting Spark applications* topics.

Related Information

Submitting Spark applications

Restore tables from backups

You can use the KuduRestore Spark job to restore one or more Kudu tables. For each backed up table, the KuduRestore job restores the full backup and each associated incremental backup until the full table state is restored.

Restoring the full series of full and incremental backups is possible because the backups are linked via the from_ms and to_ms fields in the backup metadata. By default the restore job will create tables with the same name as the table that was backed up. If you want to side-load the tables without affecting the existing tables, you can pass the --tableSuffix flag to append a suffix to each restored table.

Following are the common flags that are used when restoring the tables:

- --rootPath: The root path to the backup data. Accepts any Spark-compatible path.

  See *Backup directory structure* for the directory structure used in the rootPath.
- --kuduMasterAddresses: Comma-separated addresses of Kudu masters. The default value is localhost.
- --createTables: If set to true, the restore process creates the tables. Set it to false if the target tables already exist. The default value is true.
- --tableSuffix: If set, it adds a suffix to the restored table names. Only used when createTables is true.
- --timestampMs: A UNIX timestamp in milliseconds that defines the latest time to use when selecting restore candidates. The default is System.currentTimeMillis().
- <table>…: A list of tables to restore.

Note: You can see the full list of the job options by passing the --help flag.

Following is an example of a KuduRestore job execution which restores the tables foo and bar from the HDFS directory kudu-backups:

```
spark-submit --class org.apache.kudu.backup.KuduRestore kudu-backup2_2.11-1.
12.0.jar \
  --kuduMasterAddresses master1-host,master-2-host,master-3-host \
  --rootPath hdfs:///kudu-backups \
  foo bar
```

Related Information

Backup directory structure

Recover from disk failure

Kudu nodes can only survive failures of disks on which certain Kudu directories are mounted. For more information about the different Kudu directory types, see the *Directory configuration* topic.

About this task

The following table summarizes the resilience to disk failure in different releases of Apache Kudu.

Table 1: Kudu disk failure behavior

Node Type	Kudu directory type	Kudu releases that crash on disk failure
Master	All	All
Tablet Server	Directory containing WALs	All
Tablet Server	Directory containing tablet metadata	All
Tablet Server	Directory containing data blocks only	Pre-1.6.0

When a disk failure occurs that does not lead to a crash, Kudu will stop using the affected directory, shut down tablets with blocks on the affected directories, and automatically re-replicate the affected tablets to other tablet servers. The affected server will remain alive and print messages to the log indicating the disk failure, for example:

```
E1205 19:06:24.163748 27115 data_dirs.cc:1011] Directory /data/8/kudu/data m
arked as failed
E1205 19:06:30.324795 27064 log_block_manager.cc:1822] Not using report from
 /data/8/kudu/data: IO error: Could not open container 0a6283cab82d4e75848f4
9772d2638fe: /data/8/kudu/data/0a6283cab82d4e75848f49772d2638fe.metadata: Re
ad-only file system (error 30)
E1205 19:06:33.564638 27220 ts_tablet_manager.cc:946] T 4957808439314e0d9
7795c1394348d80 P 70f7ee61ead54b1885d819f354eb3405: aborting tablet bootstra
p: tablet has data in a failed directory
```

While in this state, the affected node will avoid using the failed disk, leading to lower storage volume and reduced read parallelism. The administrator can remove the failed directory from the --fs_data_dirs gflag to avoid seeing these errors.

When the disk is repaired, remounted, and ready to be reused by Kudu, take the following steps:

Procedure

1. Make sure that the Kudu portion of the disk is completely empty.
2. Stop the tablet server.
3. Update the --fs_data_dirs gflag to add /data/3, on the server with the disk failure. For example,

   ```
   $ sudo -u kudu kudu fs update_dirs --force --fs_wal_dir=/wals --fs_data_
   dirs=/data/1,/data/2,/data/3
   ```

4. Start the tablet server.
5. Run ksck to verify cluster health, on any kudu server in the cluster. For example:

   ```
   $ sudo -u kudu kudu cluster ksck master-01.example.com
   ```

What to do next

Note: Note that existing tablets will not stripe to the restored disk, but any new tablets will stripe to the restored disk.

Related Information

Directory configurations

Changing directory configuration

Recover from full disks

By default, Kudu reserves a small amount of space, 1% by capacity, in its directories. Kudu considers a disk full if there is less free space available than the reservation. Kudu nodes can only tolerate running out of space on disks on which certain Kudu directories are mounted.

The following table describes this behavior for each type of directory. The behavior is uniform across masters and tablet servers.

Kudu Directory Type	Crash on Full Disk?
Directory containing WALs	Yes
Directory containing tablet metadata	Yes
Directory containing data blocks only	No (see below)

Prior to Kudu 1.7.0, Kudu stripes tablet data across all directories, and will avoid writing data to full directories. Kudu will crash if all data directories are full.

In 1.7.0 and later, new tablets are assigned a disk group consisting of data directories. The number of data directories are as specified by the -fs_target_data_dirs_per_tablet flag with the default being 3. If Kudu is not configured with enough data directories for a full disk group, all data directories are used. When a data directory is full, Kudu will stop writing new data to it and each tablet that uses that data directory will write new data to other data directories within its group. If all data directories for a tablet are full, Kudu will crash. Periodically, Kudu will check if full data directories are still full, and will resume writing to those data directories if space has become available.

If Kudu does crash because its data directories are full, freeing space on the full directories will allow the affected daemon to restart and resume writing. Note that it may be possible for Kudu to free some space by running:

```
$ sudo -u kudu kudu fs check --repair
```

However, the above command may also fail if there is too little space left.

It is also possible to allocate additional data directories to Kudu in order to increase the overall amount of storage available. Note that existing tablets will not use new data directories, so adding a new data directory does not resolve issues with full disks.

Related Information

Directory configurations

Changing directory configuration

Bring a tablet that has lost a majority of replicas back online

If a tablet has permanently lost a majority of its replicas, it cannot recover automatically and operator intervention is required. If the tablet servers hosting a majority of the replicas are down (i.e. ones reported as "TS unavailable" by ksck), they should be recovered instead if possible.

Attention: The steps below may cause recent edits to the tablet to be lost, potentially resulting in permanent data loss. Only attempt the procedure below if it is impossible to bring a majority back online.

Suppose a tablet has lost a majority of its replicas. The first step in diagnosing and fixing the problem is to examine the tablet's state using ksck:

```
$ sudo -u kudu kudu cluster ksck --tablets=e822cab6c0584bc0858219d1539a17e6
master-00,master-01,master-02
Connected to the Master
Fetched info from all 5 Tablet Servers
Tablet e822cab6c0584bc0858219d1539a17e6 of table 'my_table' is unavailable:
 2 replica(s) not RUNNING
  638a20403e3e4ae3b55d4d07d920e6de (tserver-00:7150): RUNNING
  9a56fa85a38a4edc99c6229cba68aeaa (tserver-01:7150): bad state
    State:       FAILED
    Data state:  TABLET_DATA_READY
    Last status: <failure message>
  c311fef7708a4cf9bb11a3e4cbcaab8c (tserver-02:7150): bad state
    State:       FAILED
    Data state:  TABLET_DATA_READY
    Last status: <failure message>
```

This output shows that, for tablet e822cab6c0584bc0858219d1539a17e6, the two tablet replicas on tserver-01 and tser ver-02 failed. The remaining replica is not the leader, so the leader replica failed as well. This means the chance of data loss is higher since the remaining replica on tserver-00 may have been lagging. In general, to accept the potential data loss and restore the tablet from the remaining replicas, divide the tablet replicas into two groups:

1. Healthy replicas: Those in RUNNING state as reported by ksck
2. Unhealthy replicas

For example, in the above ksck output, the replica on tablet server tserver-00 is healthy while the replicas on tserver-01 and tserver-02 are unhealthy. On each tablet server with a healthy replica, alter the consensus configuration to remove unhealthy replicas. In the typical case of 1 out of 3 surviving replicas, there will be only one healthy replica, so the consensus configuration will be rewritten to include only the healthy replica.

```
$ sudo -u kudu kudu remote_replica unsafe_change_config tserver-00:7150 <tab
let-id> <tserver-00-uuid>
```

where <tablet-id> is e822cab6c0584bc0858219d1539a17e6 and <tserver-00-uuid> is the uuid of tserver-00, 638a 20403e3e4ae3b55d4d07d920e6de.

Once the healthy replicas' consensus configurations have been forced to exclude the unhealthy replicas, the healthy replicas will be able to elect a leader. The tablet will become available for writes though it will still be under-replicated. Shortly after the tablet becomes available, the leader master will notice that it is under-replicated, and will cause the tablet to re-replicate until the proper replication factor is restored. The unhealthy replicas will be tombstoned by the master, causing their remaining data to be deleted.

Rebuild a Kudu filesystem layout

In the event that critical files are lost, i.e. WALs or tablet-specific metadata, all Kudu directories on the server must be deleted and rebuilt to ensure correctness. Doing so will destroy the copy of the data for each tablet replica hosted on the local server. Kudu will automatically re-replicate tablet replicas removed in this way, provided the replication factor is at least three and all other servers are online and healthy.

About this task

Note: These steps use a tablet server as an example, but the steps are the same for Kudu master servers.

Warning: If multiple nodes need their FS layouts rebuilt, wait until all replicas previously hosted on each node have finished automatically re-replicating elsewhere before continuing. Failure to do so can result in permanent data loss.

Attention: Before proceeding, ensure the contents of the directories are backed up, either as a copy or in the form of other tablet replicas.

Procedure

1. The first step to rebuilding a server with a new directory configuration is emptying all of the server's existing directories. For example, if a tablet server is configured with --fs_wal_dir=/data/0/kudu-tserver-wal, --fs_metadata_dir=/data/0/kudu-tserver-meta, and --fs_data_dirs=/data/1/kudu-tserver,/data/2/kudu-tserver, the following commands will remove the WAL directory's and data directories' contents:

```
# Note: this will delete all of the data from the local tablet server.
$ rm -rf /data/0/kudu-tserver-wal/* /data/0/kudu-tserver-meta/* /data/1/kudu-tserver/* /data/2/kudu-tserver/*
```

2. If using Cloudera Manager, update the configurations for the rebuilt server to include only the desired directories. Make sure to only update the configurations of servers to which changes were applied, rather than of the entire Kudu service.
3. After directories are deleted, the server process can be started with the new directory configuration. The appropriate sub-directories will be created by Kudu upon starting up.