Cloudera Runtime 7.2.0

Apache Kudu Administration

Date published: 2020-02-28
Date modified: 2020-06-16

https://docs.cloudera.com/

Legal Notice

Contents

Apache Kudu administration

You can perform the following common administrative tasks and workflows with Apache Kudu:

Starting and stopping Kudu processes

You can start, stop, and configure Kudu services to start automatilcally by using the CLI commands.

Start Kudu services using the following commands:

```
sudo service kudu-master start
sudo service kudu-tserver start
```

To stop Kudu services, use the following commands:

```
sudo service kudu-master stop
sudo service kudu-tserver stop
```

Configure the Kudu services to start automatically when the server starts, by adding them to the default runlevel.

```
sudo chkconfig kudu-master on            # RHEL / CentOS
sudo chkconfig kudu-tserver on           # RHEL / CentOS
sudo update-rc.d kudu-master defaults    # Ubuntu
sudo update-rc.d kudu-tserver defaults   # Ubuntu
```

Kudu web interfaces

Kudu tablet servers and masters expose useful operational information on a built-in web interface.

Kudu master web interface

Kudu master processes serve their web interface on port 8051. The interface exposes several pages with information about the state of the cluster.

- A list of tablet servers, their host names, and the time of their last heartbeat.
- A list of tables, including schema and tablet location information for each.
- SQL code which you can paste into Impala Shell to add an existing table to Impala's list of known data sources.

Kudu tablet server web interface

Each tablet server serves a web interface on port 8050. The interface exposes information about each tablet hosted on the server, its current state, and debugging information about maintenance background operations.

Common web interface pages

Both Kudu masters and tablet servers expose the following information via their web interfaces:

- HTTP access to server logs.
- An /rpcz endpoint which lists currently running RPCs via JSON.
- Details about the memory usage of different components of the process.
- The current set of configuration flags.
- Currently running threads and their resource consumption.
- A JSON endpoint exposing metrics about the server.
- The version number of the daemon deployed on the cluster.

These interfaces are linked from the landing page of each daemon's web UI.

Kudu metrics

Kudu daemons expose a large number of metrics. Some metrics are associated with an entire server process, whereas others are associated with a particular tablet replica.

Listing available metrics

The full set of available metrics for a Kudu server can be dumped using a special command line flag:

```
$ kudu-tserver --dump_metrics_json
$ kudu-master --dump_metrics_json
```

This will output a large JSON document. Each metric indicates its name, label, description, units, and type. Because the output is JSON-formatted, this information can easily be parsed and fed into other tooling which collects metrics from Kudu servers.

For the complete list of metrics collected by Cloudera Manager for a Kudu service, look for the Kudu metrics listed under *Cloudera Manager Metrics*.

Related Information

Cloudera Manager metrics

Collecting metrics via HTTP

Metrics can be collected from a server process via its HTTP interface by visiting /metrics. The output of this page is JSON for easy parsing by monitoring services. This endpoint accepts several GET parameters in its query string:

- /metrics?metrics=<substring1>,<substring2>,… - Limits the returned metrics to those which contain at least one of the provided substrings. The substrings also match entity names, so this may be used to collect metrics for a specific tablet.
- /metrics?include_schema=1 - Includes metrics schema information such as unit, description, and label in the JSON output. This information is typically omitted to save space.
- /metrics?compact=1 - Eliminates unnecessary whitespace from the resulting JSON, which can decrease bandwidth when fetching this page from a remote host.
- /metrics?include_raw_histograms=1 - Include the raw buckets and values for histogram metrics, enabling accurate aggregation of percentile metrics over time and across hosts.
- /metrics?level=info - Limits the returned metrics based on their severity level. The levels are ordered and lower levels include the levels above them. If no level is specified, debug is used to include all metrics. The valid values are:
  - debug - Metrics that are diagnostically helpful but generally not monitored during normal operation.
  - info - Generally useful metrics that operators always want to have available but may not be monitored under normal circumstances.
  - warn - Metrics which can often indicate operational oddities, which may need more investigation.

For example:

```
$ curl -s 'http://example-ts:8050/metrics?include_schema=1&metrics=connectio
ns_accepted'
```

```
[
    {
        "type": "server",
        "id": "kudu.tabletserver",
        "attributes": {},
        "metrics": [
            {
```

```
                "name": "rpc_connections_accepted",
                "label": "RPC Connections Accepted",
                "type": "counter",
                "unit": "connections",
                "description": "Number of incoming TCP connections made to
the RPC server",
                "value": 92
            }
        ]
    }
]
```

```
$ curl -s 'http://example-ts:8050/metrics?metrics=log_append_latency'
```

```
[
    {
        "type": "tablet",
        "id": "c0ebf9fef1b847e2a83c7bd35c2056b1",
        "attributes": {
            "table_name": "lineitem",
            "partition": "hash buckets: (55), range: [(<start>), (<end>))",
            "table_id": ""
        },
        "metrics": [
            {
                "name": "log_append_latency",
                "total_count": 7498,
                "min": 4,
                "mean": 69.3649,
                "percentile_75": 29,
                "percentile_95": 38,
                "percentile_99": 45,
                "percentile_99_9": 95,
                "percentile_99_99": 167,
                "max": 367244,
                "total_sum": 520098
            }
        ]
    }
]
```

Diagnostics logging

Kudu may be configured to periodically dump all of its metrics to a local log file using the --metrics_log_interv al_msflag. Set this flag to the interval at which metrics should be written to a diagnostics log file.

The diagnostics log will be written to the same directory as the other Kudu log files, with a similar naming format, substituting diagnostics instead of a log level like INFO. After any diagnostics log file reaches 64MB uncompressed, the log will be rolled and the previous file will be gzip-compressed.

The log file generated has three space-separated fields. The first field is the word metrics. The second field is the current timestamp in microseconds since the Unix epoch. The third is the current value of all metrics on the server, using a compact JSON encoding. The encoding is the same as the metrics fetched via HTTP described above.

Rack awareness (Location awareness)

Kudu supports a rack awareness feature. Kudu's ordinary re-replication methods ensure the availability of the cluster in the event of a single node failure. However, clusters can be vulnerable to correlated failures of multiple nodes. For example, all of the physical hosts on the same rack in a datacenter may become unavailable simultaneously if the top-

of-rack switch fails. Kudu's rack awareness feature provides protection from certain kinds of correlated failures, such as the failure of a single rack in a datacenter.

The first element of Kudu's rack awareness feature is location assignment. When a tablet server registers with a master, the master assigns it a location. A location is a /-separated string that begins with a / and where each /-separated component consists of characters from the set [a-zA-Z0-9_-.]. For example, /dc-0/rack-09 is a valid location, while rack-04 and /rack=1 are not valid locations. Thus location strings resemble absolute UNIX file paths where characters in directory and file names are restricted to the set [a-zA-Z0-9_-.]. Presently, Kudu does not use the hierarchical structure of locations, but it may in the future. Location assignment is done by a user-provided command, whose path should be specified using the --location_mapping_cmd master flag. The command should take a single argument, the IP address or hostname of a tablet server, and return the location for the tablet server. Make sure that all Kudu masters are using the same location mapping command.

The second element of Kudu's rack awareness feature is the placement policy: Do not place a majority of replicas of a tablet on tablet servers in the same location.

The leader master, when placing newly created replicas on tablet servers and when re-replicating existing tablets, will attempt to place the replicas in a way that complies with the placement policy. For example, in a cluster with five tablet servers A, B, C, D, and E, with respective locations /L0, /L0, /L1, /L1, /L2, to comply with the placement policy a new 3x replicated tablet could have its replicas placed on A, C, and E, but not on A, B, and C, because then the tablet would have 2/3 replicas in location /L0. As another example, if a tablet has replicas on tablet servers A, C, and E, and then C fails, the replacement replica must be placed on D in order to comply with the placement policy.

In the case where it is impossible to place replicas in a way that complies with the placement policy, Kudu will violate the policy and place a replica anyway. For example, using the setup described in the previous paragraph, if a tablet has replicas on tablet servers A, C, and E, and then E fails, Kudu will re-replicate the tablet onto one of B or D, violating the placement policy, rather than leaving the tablet under-replicated indefinitely. The kudu cluster rebalan ce tool can reestablish the placement policy if it is possible to do so. The kudu cluster rebalance tool can also be used to reimpose the placement policy on a cluster if the cluster has just been configured to use the rack awareness feature and existing replicas need to be moved to comply with the placement policy. See Running a tablet rebalancing tool on a rack-aware cluster on page 32 for more information.

Backup and restore

Kudu supports both full and incremental table backups via a job implemented using Apache Spark. Additionally, it supports restoring tables from full and incremental backups via 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

Apache Spark Documentation

Submitting Spark applications

Backing up tables

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

- --rootPath: The root path is used to output backup data. It accepts any Spark-compatible path.
- --kuduMasterAddresses: This is used to specify a comma-separated addresses of Kudu masters. The default value is localhost.
- <*table*>...: Is used to indicate a list of tables that you want to back up.

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

Following is an example of a KuduBackup job execution which backs up the tables foo and bar to the HDFS directory kudu-backups:

```
spark-submit --class org.apache.kudu.backup.KuduBackup kudu-backup2_2.11-1.1
2.0.jar \
  --kuduMasterAddresses master1-host,master-2-host,master-3-host \
  --rootPath hdfs:///kudu-backups \
  foo bar
```

Restoring tables from backups

You can use the KuduRestore Spark job to restore one or more Kudu tables. For each backed up table, the KuduRest ore 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 --tableS uffix 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 on page 10 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
```

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
```

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.

Common Kudu workflows

Some common Kudu administrative tasks include migrating to multiple Kudu masters, recovering from a dead Kudu master, removing unwanted masters from a multi-master deployment, adding or updating hostnames of the masters within the clusters without aliases, monitoring the health of the cluster using ksck, changing directory configuration, recovering from disk failures, bringing a tablet that has lost a majority of replicas back online, rebuilding a Kudu filesystem layout, taking physical backups of an entire node, and scale the storage for the Kudu master and the tablet severs in the cloud.

Migrating to multiple Kudu masters

To provide high availability and to avoid a single point of failure, Kudu clusters should be created with multiple masters. Many Kudu clusters were created with just a single master, either for simplicity or because Kudu multi-master support was still experimental at the time. This workflow demonstrates how to migrate to a multi-master configuration. It can also be used to migrate from two masters to three with straightforward modifications.

Important:

- This workflow is unsafe for adding new masters to an existing multi-master configuration that already has three or more masters. Do not use it for that purpose.
- An even number of masters doesn't provide any benefit over having one fewer masters. This guide should always be used for migrating to three masters.
- This workflow presumes you are familiar with Kudu configuration management, with or without Cloudera Manager.
- All of the command line steps below should be executed as the Kudu UNIX user. The example commands assume the Kudu Unix user is kudu, which is typical.

Prepare for the migration

To prepare for the migration, record the port, UUID, and the location of the write-ahead log on the existing master. Decide the number of masters that you want to use. Then select an unused machine from the cluster and configure it as the new master.

Procedure

1. Establish a maintenance window (one hour should be sufficient). During this time the Kudu cluster will be unavailable.
2. Decide how many masters to use. The number of masters should be odd. Three or five node master configurations are recommended; they can tolerate one or two failures respectively.
3. Perform the following preparatory steps for the existing master:
   - Identify and record the directories where the master’s write-ahead log (WAL) and data live. If using Kudu system packages, their default locations are /var/lib/kudu/master, but they may be customized using the fs_wal_dir and fs_data_dirs configuration parameters. The command below assume that fs_wal_dir is /data/kudu/test/wal and fs_data_dirs is /data/kudu/test/data. Your configuration may differ.
   - Identify and record the port the master is using for RPCs. The default port value is 7051, but it may have been customized using the rpc_bind_addresses configuration parameter.

- Identify the master's UUID. It can be fetched using the following command:

```
$ sudo -u kudu kudu fs dump uuid --fs_wal_dir=<master_wal_dir> [--fs_dat
a_dirs=<master_data_dir>] 2>/dev/null
```

master_data_dir

The location of the existing master's previously recorded data directory.

For example:

```
$ sudo -u kudu kudu fs dump uuid --fs_wal_dir=/var/lib/kudu/mast
er 2>/dev/null
4aab798a69e94fab8d77069edff28ce0
```

- (Optional) Configure a DNS alias for the master. The alias could be a DNS cname (if the machine already has an A record in DNS), an A record (if the machine is only known by its IP address), or an alias in /etc/hosts. The alias should be an abstract representation of the master (e.g. master-1).

Important: Without DNS aliases, it is not possible to recover from permanent master failures without bringing the cluster down for maintenance. It is highly recommended that you use DNS aliases.

4. If you have Kudu tables that are accessed from Impala, you must update the master addresses in the Apache Hive Metastore (HMS) database.
   - If you set up the DNS aliases, run the following statement in impala-shell, replacing master-1, master-2, and master-3 with your actual aliases.

```
ALTER TABLE table_name
SET TBLPROPERTIES
('kudu.master_addresses' = 'master-1,master-2,master-3');
```

   - If you do not have DNS aliases set up, see Step #11 in the Performing the migration section for updating HMS.
5. Perform the following preparatory steps for each new master:
   - Choose an unused machine in the cluster. The master generates very little load so it can be collocated with other data services or load-generating processes, though not with another Kudu master from the same configuration.
   - Ensure Kudu is installed on the machine, either using system packages (in which case the kudu and kudu-master packages should be installed), or some other means.
   - Choose and record the directory where the master's data will live.
   - Choose and record the port the master should use for RPCs.
   - (Optional) Configure a DNS alias for the master (e.g. master-2, master-3, etc).

Perform the migration

For migrating to multiple Kudu masters, you need to bring the Kudu clusters down. Therefore, identify at least a one-hour maintenance window for this task.

Procedure

1. Stop all the Kudu processes in the entire cluster.
2. Format the data directory on each new master machine, and record the generated UUID. Use the following commands:

```
$ sudo -u kudu kudu fs format --fs_wal_dir=<master_wal_dir> [--fs_data_d
irs=<master_data_dir>]
```

```
$ sudo -u kudu kudu fs dump uuid --fs_wal_dir=<master_wal_dir> [--fs_data
_dirs=<master_data_dir>] 2>/dev/null
```

master_data_dir

The new master's previously recorded data directory.

For example:

```
$ sudo -u kudu kudu fs format --fs_wal_dir=/data/kudu/test/wal --
fs_data_dirs=/data/kudu/test/data
sudo -u kudu kudu fs dump uuid --fs_wal_dir=/data/kudu/test/wal -
-fs_data_dirs=/data/kudu/test/data 2>/dev/null
f5624e05f40649b79a757629a69d061e
```

3. If you are using Cloudera Manager, add the new Kudu master roles now, but do not start them.
   - If using DNS aliases, override the empty value of the Master Address parameter for each role (including the existing master role) with that master's alias.
   - Add the port number (separated by a colon) if using a non-default RPC port value.
4. Rewrite the master's Raft configuration with the following command, executed on the existing master:

```
$ sudo -u kudu kudu local_replica cmeta rewrite_raft_config --fs_wal_dir
=<master_wal_dir> [--fs_data_dirs=<master_data_dir>] <tablet_id> <all_ma
sters>
```

master_data_dir

The existing master's previously recorded data directory

tablet_id

This must be set to the string, 00000000000000000000000000000000.

all_masters

A space-separated list of masters, both new and existing. Each entry in the list must be a string of the form <uuid>:<hostname>:<port>.

uuid

The master's previously recorded UUID.

hostname

The master's previously recorded hostname or alias.

port

The master's previously recorded RPC port number.

For example:

```
$ sudo -u kudu kudu local_replica cmeta rewrite_raft_config --fs
_wal_dir=/data/kudu/test/wal --fs_data_dirs=/data/kudu/test/data
 00000000000000000000000000000000 4aab798a69e94fab8d77069edff28c
e0:master-1:7051 f5624e05f40649b79a757629a69d061e:master-2:7051
988d8ac6530f426cbe180be5ba52033d:master-3:7051
```

Important: If you are using Cloudera Manager, skip the next step.

5. Modify the value of the master_addresses configuration parameter for both existing master and new masters. The new value must be a comma-separated list of all of the masters. Each entry is a string of the form, <hostname>:<port>.

   hostname

   The master's previously recorded hostname or alias.

   port

   The master's previously recorded RPC port number.

6. Start the existing master.
7. Copy the master data to each new master with the following command, executed on each new master machine.

   

   Important: If your Kudu cluster is secure, in addition to running as the Kudu UNIX user, you must authenticate as the Kudu service user prior to running this command.

   ```
   $ sudo -u kudu kudu local_replica copy_from_remote --fs_wal_dir=<master_
   data_dir> <tablet_id> <existing_master>
   ```

   master_data_dir

   The new master's previously recorded data directory.

   tablet_id

   Must be set to the string, 00000000000000000000000000000000.

   existing_master

   RPC address of the existing master. It must be a string of the form <hostname>:<port>.

   hostname

   The existing master's previously recorded hostname or alias.

   port

   The existing master's previously recorded RPC port number.

   Example

   ```
   $ sudo -u kudu kudu local_replica copy_from_remote --fs_wal_dir=
   /data/kudu/test/wal --fs_data_dirs=/data/kudu/test/data  0000000
   00000000000000000000000000 master-1:7051
   ```

8. Start all the new masters.

   

   Important: If you are using Cloudera Manager, skip the next step.

9. Modify the value of the tserver_master_addrs configuration parameter for each tablet server. The new value must be a comma-separated list of masters where each entry is a string of the form <hostname>:<port>

   hostname

   The master's previously recorded hostname or alias

   port

   The master's previously recorded RPC port number

10. Start all the tablet servers.

11. If you have Kudu tables that are accessed from Impala and you didn't set up DNS aliases, update the HMS database manually in the underlying database that provides the storage for HMS.
    - The following is an example SQL statement you would run in the HMS database:

      ```
      UPDATE TABLE_PARAMS
      SET PARAM_VALUE =
        'master-1.example.com,master-2.example.com,master-3.example.com'
      WHERE PARAM_KEY = 'kudu.master_addresses' AND PARAM_VALUE = 'old-master
      ';
      ```
    - Invalidate the metadata by running the command in impala-shell:

      ```
      INVALIDATE METADATA;
      ```

What to do next

To verify that all masters are working properly, consider performing the following sanity checks:

- Using a browser, visit each master's web UI and navigate to the /masters page. All the masters should now be listed there with one master in the LEADER role and the others in the FOLLOWER role. The contents of /masters on each master should be the same.
- Run a Kudu system check (ksck) on the cluster using the kudu command line tool.

Related Information

Monitoring cluster health with ksck

Recovering from a dead Kudu master in a multi-master deployment

Kudu multi-master deployments function normally in the event of a master loss. However, it is important to replace the dead master. Otherwise a second failure may lead to a loss of availability, depending on the number of available masters. This workflow describes how to replace the dead master.

Due to KUDU-1620, it is not possible to perform this workflow without also restarting the live masters. As such, the workflow requires a maintenance window, albeit a potentially brief one if the cluster was set up with DNS aliases.

Important:

- Kudu does not yet support live Raft configuration changes for masters. As such, it is only possible to replace a master if the deployment was created with DNS aliases or if every node in the cluster is first shut down. See the previous multi-master migration workflow for more details on deploying with DNS aliases.
- The workflow presupposes at least basic familiarity with Kudu configuration management. If using Cloudera Manager, the workflow also presupposes familiarity with it.
- All of the command line steps below should be executed as the Kudu UNIX user, typically kudu.

Prepare for the recovery

It is crucial to make sure that the master node is truly dead and does not accidentally restart while you are preparing for the recovery.

Procedure

1. If the cluster was configured without DNS aliases perform the following steps. Otherwise move on to step 2:
   a) Establish a maintenance window (one hour should be sufficient). During this time the Kudu cluster will be unavailable.
   b) Shut down all Kudu tablet server processes in the cluster.
2. Ensure that the dead master is well and truly dead. Take whatever steps needed to prevent it from accidentally restarting; this can be quite dangerous for the cluster post-recovery.
3. Choose one of the remaining live masters to serve as a basis for recovery. The rest of this workflow will refer to this master as the "reference" master.

4. Choose an unused machine in the cluster where the new master will live. The master generates very little load so it can be co-located with other data services or load-generating processes, though not with another Kudu master from the same configuration. The rest of this workflow will refer to this master as the "replacement" master.
5. Perform the following preparatory steps for the replacement master:
   - Ensure Kudu is installed on the machine, either via system packages (in which case the kudu and kudu-master packages should be installed), or via some other means.
   - Choose and record the directory where the master's data will live.
6. Perform the following preparatory steps for each live master:
   - Identify and record the directory where the master's data lives. If using Kudu system packages, the default value is /var/lib/kudu/master, but it may be customized via the fs_wal_dir and fs_data_dirs configuration parameter. Please note if you've set fs_data_dirs to some directories other than the value of fs_wal_dir, it should be explicitly included in every command below where fs_wal_dir is also included.
   - Identify and record the master's UUID. It can be fetched using the following command:

     ```
     $ sudo -u kudu kudu fs dump uuid --fs_wal_dir=<master_wal_dir> [--fs_dat
     a_dirs=<master_data_dir>] 2>/dev/null
     ```

     master_data_dir

     live master's previously recorded data directory

     Example

     ```
     $ sudo -u kudu kudu fs dump uuid --fs_wal_dir=/data/kudu/test/wal
      --fs_data_dirs=/data/kudu/test/data 2>/dev/null
     80a82c4b8a9f4c819bab744927ad765c
     ```
7. Perform the following preparatory steps for the reference master:
   - Identify and record the directory where the master's data lives. If using Kudu system packages, the default value is /var/lib/kudu/master, but it may be customized using the fs_wal_dir and fs_data_dirs configuration parameter. If you have set fs_data_dirs to some directories other than the value of fs_wal_dir, it should be explicitly included in every command below where fs_wal_dir is also included.
   - Identify and record the UUIDs of every master in the cluster, using the following command:

     ```
     $ sudo -u kudu kudu local_replica cmeta print_replica_uuids --fs_wal_dir
     =<master_data_dir> <tablet_id> 2>/dev/null
     ```

     master_data_dir

     The reference master's previously recorded data directory.

     tablet_id

     Must be set to the string, 00000000000000000000000000000000.

     For example

     ```
     $ sudo -u kudu kudu local_replica cmeta print_replica_uuids --fs
     _wal_dir=/data/kudu/test/wal --fs_data_dirs=/data/kudu/test/data
      00000000000000000000000000000000 2>/dev/null
     80a82c4b8a9f4c819bab744927ad765c 2a73eeee5d47413981d9a1c637cce170
      1c3f3094256347528d02ec107466aef3
     ```
8. Using the two previously-recorded lists of UUIDs (one for all live masters and one for all masters), determine and record (by process of elimination) the UUID of the dead master.

Perform the recovery

After you have identified a reference master, you need to copy the master data to the replacement master node. You need to bring the Kudu clusters down. Therefore, identify at least a one-hour maintenance window for this task.

Procedure

1. Format the data directory on the replacement master machine using the previously recorded UUID of the dead master. Use the following command sequence:

```
$ sudo -u kudu kudu fs format --fs_wal_dir=<master_wal_dir> [--fs_data_d
irs=<master_data_dir>] --uuid=<uuid>
```

master_data_dir

The replacement master's previously recorded data directory.

uuid

The dead master's previously recorded UUID.

For example:

```
$ sudo -u kudu kudu fs format --fs_wal_dir=/data/kudu/test/wal -
-fs_data_dirs=/data/kudu/test/data --uuid=80a82c4b8a9f4c819bab74
4927ad765c
```

2. Copy the master data to the replacement master with the following command:

Important: If your Kudu cluster is secure, in addition to running as the Kudu UNIX user, you must authenticate as the Kudu service user prior to running this command.

```
$ sudo -u kudu kudu local_replica copy_from_remote --fs_wal_dir=<master_
wal_dir> [--fs_data_dirs=<master_data_dir>] <tablet_id> <reference_master>
```

master_data_dir

The replacement master's previously recorded data directory.

tablet_id

Must be set to the string, 00000000000000000000000000000000.

reference_master

The RPC address of the reference master. It must be a string of the form <hostname>:<port>.

hostname

The reference master's previously recorded hostname or alias.

port

The reference master's previously recorded RPC port number.

For example:

```
$ sudo -u kudu kudu local_replica copy_from_remote --fs_wal_dir=
/data/kudu/test/wal --fs_data_dirs=/data/kudu/test/data 00000000
00000000000000000000000 master-2:7051
```

3. If you are using Cloudera Manager, add the replacement Kudu master role now, but do not start it.
   - Override the empty value of the Master Address parameter for the new role with the replacement master's alias.
   - If you are using a non-default RPC port, add the port number (separated by a colon) as well.
4. If the cluster was set up with DNS aliases, reconfigure the DNS alias for the dead master to point at the replacement master.

5. If the cluster was set up without DNS aliases, perform the following steps:
   a) Stop the remaining live masters.
   b) Rewrite the Raft configurations on these masters to include the replacement master. See Step 4 in the *Perform the migration* topic for more details.
6. Start the replacement master.
7. Restart the remaining masters in the new multi-master deployment. While the masters are shut down, there will be an availability outage, but it should last only as long as it takes for the masters to come back up.

What to do next

To verify that all masters are working properly, consider performing the following sanity checks:

- Using a browser, visit each master's web UI and navigate to the /masters page. All the masters should now be listed there with one master in the LEADER role and the others in the FOLLOWER role. The contents of /masters on each master should be the same.
- Run a Kudu system check (ksck) on the cluster using the kudu command line tool.

Removing Kudu masters from a multi-master deployment

In the event that a multi-master deployment has been overallocated nodes, the following steps should be taken to remove the unwanted masters.

Important:

- In planning the new multi-master configuration, keep in mind that the number of masters should be odd and that three or five node master configurations are recommended.
- Dropping the number of masters below the number of masters currently needed for a Raft majority can incur data loss. To mitigate this, ensure that the leader master is not removed during this process.

Prepare for removal

In order to remove the unwanted masters from a multi-master deployment, you need to identify them and note their UUID and RPC addresses.

Procedure

1. Establish a maintenance window (one hour should be sufficient). During this time the Kudu cluster will be unavailable.
2. Identify the UUID and RPC address current leader of the multi-master deployment by visiting the /masters page of any master's web UI. This master must not be removed during this process; its removal may result in severe data loss.
3. Stop all the Kudu processes in the entire cluster.
4. If you are using Cloudera Manager, remove the unwanted Kudu master from your cluster's Kudu service.

Perform the removal

When you remove any Kudu masters from a multi-master deployment, you need to rewrite the Raft configuration on the remaining masters, remove data and WAL directories from the unwanted masters, and finaly modify the value of the tserver_master_addrs configuration parameter for the tablet servers to remove the unwanted masters. You need to bring the Kudu clusters down. Therefore, identify at least a one-hour maintenance window for this task.

Procedure

1. Rewrite the Raft configuration on the remaining masters to include only the remaining masters.

```
$ sudo -u kudu kudu local_replica cmeta rewrite_raft_config --fs_wal_dir
=<master_wal_dir> [--fs_data_dirs=<master_data_dir>] <tablet_id> <all_ma
sters>
```

master_data_dir

The existing master's previously recorded data directory

tablet_id

This must be set to the string, 00000000000000000000000000000000.

all_masters

A space-separated list of masters, both new and existing. Each entry in the list must be a string of the form <uuid>:<hostname>:<port>.

uuid

The master's previously recorded UUID.

hostname

The master's previously recorded hostname or alias.

port

The master's previously recorded RPC port number.

For example:

```
$ sudo -u kudu kudu local_replica cmeta rewrite_raft_config --fs
_wal_dir=/data/kudu/test/wal --fs_data_dirs=/data/kudu/test/data
 00000000000000000000000000000000 4aab798a69e94fab8d77069edff28c
e0:master-1:7051 f5624e05f40649b79a757629a69d061e:master-2:7051
988d8ac6530f426cbe180be5ba52033d:master-3:7051
```

Important: If you are using Cloudera Manager, skip the next step.

2. Remove the data directories and WAL directory on the unwanted masters. This is a precaution to ensure that they cannot start up again and interfere with the new multi-master deployment.
3. Modify the value of the master_addresses configuration parameter for the masters of the new multi-master deployment. If migrating to a single-master deployment, the master_addresses flag should be omitted entirely.
4. Start all of the masters that were not removed.

Important: If you are using Cloudera Manager, skip the next step.

5. Modify the value of the tserver_master_addrs configuration parameter for the tablet servers to remove any unwanted masters.
6. Start all of the tablet servers.

What to do next

To verify that all masters are working properly, consider performing the following sanity checks:

- Using a browser, visit each master's web UI and navigate to the /masters page. All the masters should now be listed there with one master in the LEADER role and the others in the FOLLOWER role. The contents of /masters on each master should be the same.

- Run a Kudu system check (ksck) on the cluster using the kudu command line tool.

Related Information

Monitoring cluster health with ksck

Changing master hostnames

When replacing dead masters, use DNS aliases to prevent long maintenance windows. If the cluster was set up without aliases, change the host names as described in this section.

Prepare for hostname changes

In this step, you need to identify a down-time window, and note the UUID and the RPC address of each master.

Procedure

1. Establish a maintenance window during which the Kudu cluster will be unavailable. One hour should be sufficient.
2. On the **Masters** page in Kudu Web UI, note the UUID and RPC address of each master.
3. Stop all the Kudu processes in the cluster.
4. Set up the new hostnames to point to the masters and verify all servers and clients properly resolve them.

Perform hostname changes

You need to bring the Kudu clusters down to update the hostnames. Therefore, identify at least a one-hour maintenance window for this task.

Procedure

1. Rewrite each master's Raft configuration with the following command, executed on each master host:

```
$ sudo -u kudu kudu local_replica cmeta rewrite_raft_config --fs_wal_dir
=<master_wal_dir> [--fs_data_dirs=<master_data_dir>] 00000000000000000000
0000000000000 <all_masters>
```

   For example:

```
$ sudo -u kudu kudu local_replica cmeta rewrite_raft_config --fs_wal_dir=/
data/kudu/test/wal --fs_data_dirs=/data/kudu/test/data 00000000000000000000
0000000000000 4aab798a69e94fab8d77069edff28ce0:new-master-name-1:7051 f5
624e05f40649b79a757629a69d061e:new-master-name-2:7051 988d8ac6530f426cbe
180be5ba52033d:new-master-name-3:7051
```

2. Update the master address:
   - In an environment not managed by Cloudera Manager, change the gflag file of the masters so the master_addresses parameter reflects the new hostnames.
   - In an environment managed by Cloudera Manager, specify the new hostname in the Master Address (server.address) field on each Kudu role.
3. Change the gflag file of the tablet servers to update the tserver_master_addrs parameter with the new hostnames. In an environment managed by Cloudera Manager, this step is not needeed.
4. Start the masters.

5. To verify that all masters are working properly, perform the following sanity checks:
   a) In each master's Web UI, click Masters on the Status Pages. All of the masters should be listed there with one master in the LEADER role field and the others in the FOLLOWER role field. The contents of Masters on all master should be the same.
   b) Run the below command to verify all masters are up and listening. The UUIDs are the same and belong to the same master as before the hostname change:

```
$ sudo -u kudu kudu master list new-master-name-1:7051,new-master-name-2
:7051,new-master-name-3:7051
```

6. Start all of the tablet servers.
7. Run a Kudu system check (ksck) on the cluster using the kudu command line tool. After startup, some tablets may be unavailable as it takes some time to initialize all of them.
8. If you have Kudu tables that are accessed from Impala, update the HMS database manually in the underlying database that provides the storage for HMS.
   a) The following is an example SQL statement you run in the HMS database:

```
UPDATE TABLE_PARAMSSET PARAM_VALUE =
'new-master-name-1:7051,new-master-name-2:7051,new-master-name-3:7051'
WHERE PARAM_KEY = 'kudu.master_addresses'
AND PARAM_VALUE = 'master-1:7051,master-2:7051,master-3:7051';
```

   b) In impala-shell, run:

```
INVALIDATE METADATA;
```

   c) Verify updating the metadata worked by running a simple SELECT query on a Kudu-backed Impala table.

Related Information

Monitoring cluster health with ksck

Best practices when adding new tablet servers

A common workflow when administering a Kudu cluster is adding additional tablet server instances, in an effort to increase storage capacity, decrease load or utilization on individual hosts, increase compute power, and more.

By default, any newly added tablet servers will not be utilized immediately after their addition to the cluster. Instead, newly added tablet servers will only be utilized when new tablets are created or when existing tablets need to be replicated, which can lead to imbalanced nodes. It's recommended to run the rebalancer CLI tool just after adding a new tablet server into the cluster.

Avoid placing multiple tablet servers on a single node. Doing so nullifies the point of increasing the overall storage capacity of a Kudu cluster and increases the likelihood of tablet unavailability when a single node fails (the latter drawback is not applicable if the cluster is properly configured to use the rack awareness (location awareness) feature.

To add additional tablet servers to an existing cluster, the following steps can be taken to ensure tablet replicas are uniformly distributed across the cluster:

1. Ensure that Kudu is installed on the new machines being added to the cluster, and that the new instances have been correctly configured to point to the pre-existing cluster. Then, start the new tablet server instances.
2. Verify that the new instances check in with the Kudu Master(s) successfully. A quick method for verifying whether they have successfully checked in with the existing Master instances is to view the Kudu Master WebUI, specifically the /tablet-servers section, and validate that the newly added instances are registered, and have a heartbeat.
3. Once the tablet server(s) are successfully online and healthy, follow the steps to run the rebalancing tool which spreads the existing tablet replicas to the newly added tablet servers.
4. After the rebalancer tool has completed, or even during its execution, you can check the health of the cluster using the ksck command-line utility.

Monitoring cluster health with ksck

The kudu CLI includes a tool called ksck that can be used for gathering information about the state of a Kudu cluster, including checking its health. ksck will identify issues such as under-replicated tablets, unreachable tablet servers, or tablets without a leader.

ksck should be run from the command line as the Kudu admin user, and requires the full list of master addresses to be specified:

```
$ sudo -u kudu kudu cluster ksck master-01.example.com,master-02.example.com
,master-03.example.com
```

To see a full list of the options available with ksck, use the --help flag. If the cluster is healthy, ksck will print information about the cluster, a success message, and return a zero (success) exit status.

```
Master Summary
              UUID               |       Address          | Status
---------------------------------+------------------------+---------
 a811c07b99394df799e6650e7310f282 | master-01.example.com | HEALTHY
 b579355eeeea446e998606bcb7e87844 | master-02.example.com | HEALTHY
 cfdcc8592711485fad32ec4eea4fbfcd | master-02.example.com | HEALTHY

Tablet Server Summary
              UUID               |        Address         | Status
---------------------------------+------------------------+---------
 a598f75345834133a39c6e51163245db | tserver-01.example.com | HEALTHY
 e05ca6b6573b4e1f9a518157c0c0c637 | tserver-02.example.com | HEALTHY
 e7e53a91fe704296b3a59ad304e7444a | tserver-03.example.com | HEALTHY

Version Summary
 Version |      Servers
---------+-------------------------
  1.7.1  | all 6 server(s) checked

Summary by table
   Name   | RF | Status  | Total Tablets | Healthy | Recovering | Under-rep
licated | Unavailable
----------+----+---------+---------------+---------+------------+---------
---------+-------------
 my_table | 3  | HEALTHY | 8             | 8       | 0          | 0
       | 0

                | Total Count
----------------+-------------
 Masters        | 3
 Tablet Servers | 3
 Tables         | 1
 Tablets        | 8
 Replicas       | 24
OK
```

If the cluster is unhealthy, for instance if a tablet server process has stopped, ksck will report the issue(s) and return a non-zero exit status, as shown in the abbreviated snippet of ksck output below:

```
Tablet Server Summary
              UUID               |        Address         |   Status
---------------------------------+------------------------+-------------
 a598f75345834133a39c6e51163245db | tserver-01.example.com | HEALTHY
 e05ca6b6573b4e1f9a518157c0c0c637 | tserver-02.example.com | HEALTHY
 e7e53a91fe704296b3a59ad304e7444a | tserver-03.example.com | UNAVAILABLE
Error from 127.0.0.1:7150: Network error: could not get status from server:
 Client connection negotiation failed: client connection to 127.0.0.1:7150:
connect: Connection refused (error 61) (UNAVAILABLE)
```

```
... (full output elided)

------------------
Errors:
------------------
Network error: error fetching info from tablet servers: failed to gather i
nfo for all tablet servers: 1 of 3 had errors
Corruption: table consistency check error: 1 out of 1 table(s) are not hea
lthy

FAILED
Runtime error: ksck discovered errors
```

To verify data integrity, the optional --checksum_scan flag can be set, which will ensure the cluster has consistent data by scanning each tablet replica and comparing results. The --tables or --tablets flags can be used to limit the scope of the checksum scan to specific tables or tablets, respectively.

For example, checking data integrity on the my_table table can be done with the following command:

```
$ sudo -u kudu kudu cluster ksck --checksum_scan --tables my_table master-01
.example.com,master-02.example.com,master-03.example.com
```

By default, ksck will attempt to use a snapshot scan of the table, so the checksum scan can be done while writes continue.

Finally, ksck also supports output in JSON format using the --ksck_format flag. JSON output contains the same information as the plain text output, but in a format that can be used by other tools. See kudu cluster ksck --help for more information.

Orchestrating a rolling restart with no downtime

Kudu 1.12 provides tooling to restart a cluster with no downtime. This topic provides the steps to perform rolling restart.

About this task

Note: If any tables in the cluster have a replication factor of 1, some quiescing tablet servers will never become fully quiesced, as single-replica tablets do not naturally relinquish leadership. If such tables exist, use the kudu cluster rebalance tool to move replicas of these tables away from the quiescing tablet server by specifying the --ignored_tservers, --move_replicas_from_ignored_tservers, and --tables options.

Note: If running with rack awareness, the following steps can be performed by restarting multiple tablet servers within a single rack at the same time. Use ksck to ensure that the location assignment policy is enforced while going through these steps, and that no more than a single location is restarted at the same time. At least three locations should be defined in the cluster to safely restart multiple tablet service within one location.

Cloudera Manager can automate this process, by using the “Rolling Restart” command on the Kudu service.

Note: Cloudera Manager does not support automatic moving of the single-replica tablets.

Cloudera Manager will prompt you to specify how many tablet servers to restart concurrently. If running with rack awareness with and at least three racks specified across all hosts that contain Kudu roles, it is safe to specify the restart batch with up to one rack at a time, provided the rack assignment policy is being enforced.

The following service configurations can be set to tune the parameters the rolling restart will run with:

- Rolling Restart Health Check Interval: the interval in seconds that Cloudera Manager will run ksck after restarting a batch of tablet servers, waiting for the cluster to become healthy.

- Maximum Allowed Runtime to Rolling Restart a Batch of Servers: the total amount of time in seconds Cloudera Manager will wait for the cluster to become healthy after restarting a batch of tablet servers, before exiting with an error.

Procedure

1. Restart the master(s) one-by-one. If there is only a single master, this may cause brief interference with on-going workloads.
2. Starting with a single tablet server, put the tablet server into maintenance mode by using the kudu tserver state e nter_maintenance tool.
3. Start quiescing the tablet server using the kudu tserver quiesce start tool. This signals Kudu to stop hosting leaders on the specified tablet server and to redirect new scan requests to other tablet servers.
4. Periodically run kudu tserver quiesce start with the --error_if_not_fully_quiesced option, until it returns success, indicating that all leaders have been moved away from the tablet server and that all on-going scans have completed.
5. Restart the tablet server.
6. Periodically run ksck until the cluster ireports a healthy status.
7. Exit maintenance mode on the tablet server by running kudu tserver state exit_maintenance. This allows new tablet replicas to be placed on the tablet server.
8. Repeat these steps for all tablet servers in the cluster.

Changing directory configuration

For higher read parallelism and larger volumes of storage per server, you may want to configure servers to store data in multiple directories on different devices. You can add or remove data directories to an existing master or tablet server by updating the --fs_data_dirs Gflag configuration and restarting the server. Data is striped across data directories, and when a new data directory is added, new data will be striped across the union of the old and new directories.

About this task

Note:

Removing a data directory from --fs_data_dirs may result in failed tablet replicas in cases where there were data blocks in the directory that was removed. Use ksck to ensure the cluster can fully recover from the directory removal before moving onto another server.

In versions of Kudu below 1.12, Kudu requires that the kudu fs update_dirs tool be run before restarting with a different set of data directories. Such versions will fail to start if not run.

If on a Kudu version below 1.12, once a server is started, users must go through the below steps to change the directory configuration:

Note: Unless the --force flag is specified, Kudu will not allow for the removal of a directory across which tablets are configured to spread data. If --force is specified, all tablets configured to use that directory will fail upon starting up and be replicated elsewhere.

Note: If the metadata directory overlaps with a data directory, as was the default prior to Kudu 1.7, or if a non-default metadata directory is configured, the --fs_metadata_dir configuration must be specified when running the kudu fs update_dirs tool.

Note: Only new tablet replicas, i.e. brand new tablets' replicas and replicas that are copied to the server for high availability, will use the new directory. Existing tablet replicas on the server will not be rebalanced across the new directory.

Attention: All of the command line steps below should be executed as the Kudu UNIX user, typically kudu.

Procedure

1. Use ksck to ensure the cluster is healthy, and establish a maintenance window to bring the tablet server offline.
2. The tool can only run while the server is offline, so establish a maintenance window to update the server. The tool itself runs quickly, so this offline window should be brief, and as such, only the server to update needs to be offline.

   However, if the server is offline for too long (see the follower_unavailable_considered_failed_sec flag), the tablet replicas on it may be evicted from their Raft groups. To avoid this, it may be desirable to bring the entire cluster offline while performing the update.
3. Run the tool with the desired directory configuration flags. For example, if a cluster was set up with --fs_wal_dir =/wals, ##fs_metadata_dir=/meta, and ##fs_data_dirs=/data/1,/data/2,/data/3, and /data/3 is to be removed (e.g. due to a disk error), run the command:

   ```
   $ sudo -u kudu kudu fs update_dirs --force --fs_wal_dir=/wals --fs_metad
   ata_dir=/meta --fs_data_dirs=/data/1,/data/2
   ```
4. Modify the value of the --fs_data_dirs flag for the updated sever. If using Cloudera Manager, make sure to only update the configurations of the updated server, rather than of the entire Kudu service.
5. Once complete, the server process can be started. When Kudu is installed using system packages, service is typically used:

   ```
   $ sudo service kudu-tserver start
   ```
6. Use ksck to ensure Kudu returns to a healthy state before resuming normal operation.

Recovering 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.

Note: In versions of Kudu below 1.12, in order to start Kudu with a different set of directories, the administrator should schedule a brief window to update the node's directory configuration. Kudu will fail to start otherwise.

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, potentially using the update_dirs tool if on a version of Kudu that is below 1.12:

```
$ 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. 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

Changing directory configuration

Recovering 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

Changing directory configuration

Bringing 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.

Rebuilding 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.

Physical backups of an entire node

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

Before you begin

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

Attention: 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.

Scaling storage on Kudu master and tablet servers in the cloud

If you find that the size of your Kudu cloud deployment has exceeded previous expectations, or you simply wish to allocate more storage to Kudu, use the following set of high-level steps as a guide to increasing storage on your Kudu master or tablet server hosts. You must work with your cluster's Hadoop administrators and the system administrators to complete this process. Replace the file paths in the following steps to those relevant to your setup.

Procedure

1. Run a consistency check on the cluster hosts using ksck.
2. On all Kudu hosts, create a new file system with the storage capacity you require. For example, /new/data/dir.
3. Shutdown the cluster services. For a cluster managed by Cloudera Manager, see *Stopping a cluster*.
4. Copy the contents of your existing data directory, /current/data/dir, to the new filesystem at /new/data/dir.
5. Move your existing data directory, /current/data/dir, to a separate temporary location such as /tmp/data/dir.
6. Create a new /current/data/dir directory.

```
mkdir /current/data/dir
```

7. Mount /new/data/dir as /current/data/dir. Make changes to fstab as needed.
8. Perform steps 4-7 on all Kudu hosts.
9. Startup cluster services.
10. Run a consistency check on the cluster hosts using ksck.
11. After 10 days, if everything is in working order on all the hosts, get approval from the Hadoop administrators to remove the /backup/data/dir directory.

Related Information

Monitoring cluster health with ksck

Migrating Kudu data from one directory to another on the same host

Take the following steps to move the entire Kudu data from one directory to another.

About this task

Note: The steps were verified on an environment where the master and the server instances were configured to write the WAL/Data to the same directory.

Procedure

1. Stop the Kudu service.
2. Modify the directory configurations for the Master/Server instances.
3. Move the existing data from the old directory, to the new one.
4. Make sure the file/directory ownership is set to the kudu user.

5. Restart the Kudu service.
6. Run ksck and verify for the healthy status.

Related Information

Changing directory configuration

Minimizing cluster disruption during temporary planned downtime of a single tablet server

If a single tablet server is brought down temporarily in a healthy cluster, all tablets will remain available and clients will function as normal, after potential short delays due to leader elections. However, if the downtime lasts for more than --follower_unavailable_considered_failed_sec (default 300) seconds, the tablet replicas on the down tablet server will be replaced by new replicas on available tablet servers. This will cause stress on the cluster as tablets re-replicate and, if the downtime lasts long enough, significant reduction in the number of replicas on the down tablet server, which may require the rebalancer to fix.

To work around this, in Kudu versions 1.11 onward, the kudu CLI contains a tool to put tablet servers into maintenance mode. While in this state, the tablet server's replicas are not re-replicated due to its downtime alone, though re-replication may still occur in the event that the server in maintenance suffers from a disk failure or if a follower replica on the tablet server falls too far behind its leader replica. Upon exiting maintenance, re-replication is triggered for any remaining under-replicated tablets.

The kudu tserver state enter_maintenance and kudu tserver state exit_maintenance tools are added to orchestrate tablet server maintenance. The following can be run from a tablet server to put it into maintenance:

```
$ TS_UUID=$(sudo -u kudu kudu fs dump uuid --fs_wal_dir=<wal_dir> --fs_data_
dirs=<data_dirs>)
$ sudo -u kudu kudu tserver state enter_maintenance <master_addresses> "$T
S_UUID"
```

The tablet server maintenance mode is shown in the "Tablet Servers" page of the Kudu leader master's web UI, and in the output of kudu cluster ksck. To exit maintenance mode, run the following command:

```
sudo -u kudu kudu tserver state exit_maintenance <master_addresses> "$TS_UUI
D"
```

Running tablet rebalancing tool

The kudu CLI contains a rebalancing tool that can be used to rebalance tablet replicas among tablet servers. For each table, the tool attempts to balance the number of replicas per tablet server. It will also, without unbalancing any table, attempt to even out the number of replicas per tablet server across the cluster as a whole.

The rebalancing tool should be run as the Kudu admin user, specifying all master addresses:

```
sudo -u kudu kudu cluster rebalance master-01.example.com,master-02.example.
com,master-03.example.com
```

When run, the rebalancer will report on the initial tablet replica distribution in the cluster, log the replicas it moves, and print a final summary of the distribution when it terminates:

```
Per-server replica distribution summary:
       Statistic       |   Value
-----------------------+-----------
 Minimum Replica Count | 0
 Maximum Replica Count | 24
 Average Replica Count | 14.400000
Per-table replica distribution summary:
```

```
 Replica Skew |  Value
--------------+----------
 Minimum      | 8
 Maximum      | 8
 Average      | 8.000000

I0613 14:18:49.905897 3002065792 rebalancer.cc:779] tablet e7ee9ade95b342a7a
94649b7862b345d: 206a51de1486402bbb214b5ce97a633c -> 3b4d9266ac8c45ff9a5d4d7
c3e1cb326 move scheduled
I0613 14:18:49.917578 3002065792 rebalancer.cc:779] tablet 5f03944529f44626
a0d6ec8b1edc566e: 6e64c4165b864cbab0e67ccd82091d60 -> ba8c22ab030346b4baa289
d6d11d0809 move scheduled
I0613 14:18:49.928683 3002065792 rebalancer.cc:779] tablet 9373fee3bfe74ce
c9054737371a3b15d: fab382adf72c480984c6cc868fdd5f0e -> 3b4d9266ac8c45ff9a5d4
d7c3e1cb326 move scheduled

... (full output elided)

I0613 14:19:01.162802 3002065792 rebalancer.cc:842] tablet f4c046f18b174cc
2974c65ac0bf52767: 206a51de1486402bbb214b5ce97a633c -> 3b4d9266ac8c45ff9a5d4
d7c3e1cb326 move completed: OK

rebalancing is complete: cluster is balanced (moved 28 replicas)
Per-server replica distribution summary:
       Statistic       |   Value
-----------------------+-----------
 Minimum Replica Count | 14
 Maximum Replica Count | 15
 Average Replica Count | 14.400000

Per-table replica distribution summary:
 Replica Skew |  Value
--------------+----------
 Minimum      | 1
 Maximum      | 1
 Average      | 1.000000
```

If more details are needed in addition to the replica distribution summary, use the --output_replica_distribution_details flag. If added, the flag makes the tool print per-table and per-tablet server replica distribution statistics as well.

Use the --report_only flag to get a report on table-wide and cluster-wide replica distribution statistics without starting any rebalancing activity.

The rebalancer can also be restricted to run on a subset of the tables by supplying the --tables flag. Note that, when running on a subset of tables, the tool will not attempt to balance the cluster as a whole.

The length of time rebalancing is run for can be controlled with the flag --max_run_time_sec. By default, the rebalancer will run until the cluster is balanced. To control the amount of resources devoted to rebalancing, modify the flag --max_moves_per_server. See kudu cluster rebalance --help for more.

It's safe to stop the rebalancer tool at any time. When restarted, the rebalancer will continue rebalancing the cluster.

The rebalancer tool requires all registered tablet servers to be up and running to proceed with the rebalancing process in order to avoid possible conflicts and races with the automatic re-replication and to keep replica placement optimal for current configuration of the cluster. If a tablet server becomes unavailable during the rebalancing session, the rebalancer will exit. As noted above, it's safe to restart the rebalancer after resolving the issue with unavailable tablet servers.

The rebalancing tool can rebalance Kudu clusters running older versions as well, with some restrictions. Consult the following table for more information. In the table, "RF" stands for "replication factor".

Version Range	Rebalances RF = 1 Tables?	Rebalances RF > 1 Tables?
v < 1.4.0	No	No

Version Range	Rebalances RF = 1 Tables?	Rebalances RF > 1 Tables?
1.4.0 <= v < 1.7.1	No	Yes
v >= 1.7.1	Yes	Yes

If the rebalancer is running against a cluster where rebalancing replication factor one tables is not supported, it will rebalance all the other tables and the cluster as if those singly-replicated tables did not exist.

Running a tablet rebalancing tool on a rack-aware cluster

It is possible to use the kudu cluster rebalance tool to establish the placement policy on a cluster. This might be necessary when the rack awareness feature is first configured or when re-replication violated the placement policy.

About this task

The rebalancing tool breaks its work into three phases:

Procedure

1. The rack-aware rebalancer tries to establish the placement policy. Use the ##disable_policy_fixer flag to skip this phase.
2. The rebalancer tries to balance load by location, moving tablet replicas between locations in an attempt to spread tablet replicas among locations evenly. The load of a location is measured as the total number of replicas in the location divided by the number of tablet servers in the location. Use the ##disable_cross_location_rebalancing flag to skip this phase.
3. The rebalancer tries to balance the tablet replica distribution within each location, as if the location were a cluster on its own. Use the ##disable_intra_location_rebalancing flag to skip this phase.

What to do next

By using the ##report_only flag, it’s also possible to check if all tablets in the cluster conform to the placement policy without attempting any replica movement.

Related Information

Rack awareness (Location awareness)

Running a tablet rebalancing tool in Cloudera Manager

You access and run the tablet rebalancing tool from Cloudera Manager.

Procedure

1. Browse to Clusters Kudu .
2. Click Actions and select Run Kudu Rebalancer Tool.

What to do next

In Cloudera Manager, the rebalancer runs with the default flags.

Decommissioning or permanently removing a tablet server from a cluster

Starting with Kudu 1.12, the Kudu rebalancer tool can be used to decommission a tablet server by supplying the --ignored_tservers and --move_replicas_from_ignored_tservers arguments.

About this task

Note: Do not decommission multiple tablet servers at once. To remove multiple tablet servers from the cluster, follow the below instructions for each tablet server, ensuring that the previous tablet server is removed from the cluster and ksck is healthy before shutting down the next.

Procedure

1. Ensure the cluster is in good health using ksck.
2. Put the tablet server into a maintenance mode by using the kudu tserver state enter_maintenance tool.
3. Run the kudu cluster rebalance tool, supplying the --ignored_tservers argument with the UUIDs of the tablet servers to be decommissioned, and the --move_replicas_from_ignored_tservers flag.
4. Wait for the moves to complete and for ksck to show the cluster in a healthy state.
5. The decommissioned tablet server can be brought offline.
6. To completely remove it from the cluster so ksck shows the cluster as completely healthy, restart the masters.

   If you have only one master in your deployment, this may cause cluster downtime. In a multi-master deployment, restart the masters in sequence to avoid cluster downtime.

Related Information

Monitoring cluster health with ksck

Using cluster names in the kudu command line tool

When using the kudu command line tool, it can be difficult to remember the precise list of Kudu master RPC addresses needed to communicate with a cluster, especially when managing multiple clusters. As an alternative, you can use the command line tool to identify clusters by name.

Procedure

1. Create a new directory to store the Kudu configuration file.
2. Export the path to this newly created directory in the KUDU_CONFIG environment variable.
3. Create a file called kudurc in the new directory.
4. Populate kudurc as follows, substituting your own cluster names and RPC addresses:

```
clusters_info:
  cluster_name1:
    master_addresses: ip1:port1,ip2:port2,ip3:port3
  cluster_name2:
    master_addresses: ip4:port4
```

5. When using the kudu command line tool, replace the list of Kudu master RPC addresses with the cluster name, prepended with the character @. For example:

```
$ sudo -u kudu kudu ksck @cluster_name1
```

Note: Cluster names may be used as input in any invocation of the kudu command line tool that expects a list of Kudu master RPC addresses.

Managing Kudu with Cloudera Manager

This topic describes the tasks you can perform to manage the Kudu service using Cloudera Manager. You can use Cloudera Manager to start and stop the Kudu service, monitor operations, and configure the Kudu master and tablet servers, among other tasks. Depending on your deployment, there are several different configuration settings you may need to modify.

Enabling core dump for the Kudu service

If Kudu crashes, you can use Cloudera Manager to generate a core dump to get more information about the crash.

Procedure

1. Go to the Kudu service.
2. Click the Configuration tab.
3. Search for core dump.
4. Check the checkbox for the Enable Core Dump property.
5. (Optional) Unless otherwise configured, the dump file is generated in the default core dump directory, /var/log/kudu, for both the Kudu master and the tablet servers.
   - To configure a different dump directory for the Kudu master, modify the value of the Kudu Master Core Dump Directory property.
   - To configure a different dump directory for the Kudu tablet servers, modify the value of the Kudu Tablet Server Core Dump Directory property.
6. Click Save Changes.

Verifying the Impala dependency on Kudu

In a Cloudera Manager deployment, once the Kudu service is installed, Impala will automatically identify the Kudu Master. However, if your Impala queries don't work as expected, use the following steps to make sure that the Impala service is set to be dependent on Kudu.

Procedure

1. Go to the Impala service.
2. Click the Configuration tab.
3. Search for kudu.
4. Make sure the Kudu Service property is set to the right Kudu service.
5. Click Save Changes.

Using the Charts Library with the Kudu service

By default, the Status tab for the Kudu service displays a dashboard containing a limited set of charts.

About this task

For details on the terminology used in these charts, and instructions on how to query for time-series data, display chart details, and edit charts, see Charting Time-Series Data.

The Kudu service's Charts Library tab also displays a dashboard containing a much larger set of charts, organized by categories such as process charts, host charts, CPU charts, and so on, depending on the entity (service, role, or host) that you are viewing. You can use these charts to keep track of disk space usage, the rate at which data is being inserted/modified in Kudu across all tables, or any critical cluster events. You can also use them to keep track of individual tables. For example, to find out how much space a Kudu table is using on disk:

Procedure

1. Go to the Kudu service and navigate to the Charts Library tab.
2. On the left-hand side menu, click Tables to display the list of tables currently stored in Kudu.
3. Click on a table name to view the default dashboard for that table. The Total Tablet Size On Disk Across Kudu Replicas chart displays the total size of the table on disk using a time-series chart.

   Hovering with your mouse over the line on the chart opens a small pop-up window that displays information about that data point. Click the data stream within the chart to display a larger pop-up window that includes additional information for the table at the point in time where the mouse was clicked.

Kudu security

Kudu includes security features that allow Kudu clusters to be hardened against access from unauthorized users. Kudu uses strong authentication with Kerberos, while communication between Kudu clients and servers can now be encrypted with TLS. Kudu also allows you to use HTTPS encryption to connect to the web UI.

The rest of this topic describes the security capabilities of Apache Kudu and how to configure a secure Kudu cluster. Currently, there are a few known limitations in Kudu security that might impact your cluster. See Security limitations.

Kudu authentication with Kerberos

Kudu can be configured to enforce secure authentication among servers, and between clients and servers. Authentication prevents untrusted actors from gaining access to Kudu, and securely identifies connecting users or services for authorization checks. Authentication in Kudu is designed to interoperate with other secure Hadoop components by utilizing Kerberos.

Configure authentication on Kudu servers using the --rpc_authentication flag, which can be set to one of the following options:

- required - Kudu will reject connections from clients and servers who lack authentication credentials.
- optional - Kudu will attempt to use strong authentication, but will allow unauthenticated connections.
- disabled - Kudu will only allow unauthenticated connections.

By default, the flag is set to optional. To secure your cluster, set --rpc_authentication to required.

Internal private key infrastructure (PKI)

Kudu uses an internal PKI to issue X.509 certificates to servers in the cluster. Connections between peers who have both obtained certificates will use TLS for authentication. In such cases, neither peer needs to contact the Kerberos KDC.

X.509 certificates are only used for internal communication among Kudu servers, and between Kudu clients and servers. These certificates are never presented in a public facing protocol. By using internally-issued certificates, Kudu offers strong authentication which scales to huge clusters, and allows TLS encryption to be used without requiring you to manually deploy certificates on every node.

Authentication tokens

After authenticating to a secure cluster, the Kudu client will automatically request an authentication token from the Kudu master. An authentication token encapsulates the identity of the authenticated user and carries the Kudu master's RSA signature so that its authenticity can be verified. This token will be used to authenticate subsequent connections.

By default, authentication tokens are only valid for seven days, so that even if a token were compromised, it cannot be used indefinitely. For the most part, authentication tokens should be completely transparent to users. By using authentication tokens, Kudu is able to take advantage of strong authentication, without paying the scalability cost of communicating with a central authority for every connection.

When used with distributed compute frameworks such as Apache Spark, authentication tokens can simplify configuration and improve security. For example, the Kudu Spark connector will automatically retrieve an authentication token during the planning stage, and distribute the token to tasks. This allows Spark to work against a secure Kudu cluster where only the planner node has Kerberos credentials.

Client authentication to secure Kudu clusters

Users running client Kudu applications must first run the kinit command to obtain a Kerberos ticket-granting ticket.

For example:

```
kinit admin@EXAMPLE-REALM.COM
```

Once authenticated, you use the same client code to read from and write to Kudu servers with and without the Kerberos configuration.

Scalability

Kudu authentication is designed to scale to thousands of nodes, which means it must avoid unnecessary coordination with a central authentication authority (such as the Kerberos KDC) for each connection. Instead, Kudu servers and clients use Kerberos to establish initial trust with the Kudu master, and then use alternate credentials for subsequent connections. As described previously, the Kudu master issues internal X.509 certificates to tablet servers on startup, and temporary authentication tokens to clients on first contact.

Coarse-grained authorization

Kudu supports coarse-grained authorization checks for client requests based on the client's authenticated Kerberos principal (user or service). Access levels are granted based on whitelist-style Access Control Lists (ACLs), one for each level. Each ACL specifies a comma-separated list of users, or may be set to '*' to indicate that all authenticated users have access rights at the specified level.

The two levels of access which can be configured are:

- Superuser - Principals authorized as a superuser can perform certain administrative functions such as using the kudu command line tool to diagnose and repair cluster issues.
- User - Principals authorized as a user are able to access and modify all data in the Kudu cluster. This includes the ability to create, drop, and alter tables, as well as read, insert, update, and delete data. The default value for the User ACL is '*', which allows all users access to the cluster. However, if authentication is enabled, this will restrict access to only those users who are able to successfully authenticate using Kerberos. Unauthenticated users on the same network as the Kudu servers will be unable to access the cluster.

Note: Internally, Kudu has a third access level for the daemons themselves called Service. This is used to ensure that users cannot connect to the cluster and pose as tablet servers.

Fine-grained authorization

Kudu can be configured to enforce fine-grained authorization across servers. This ensures that users can see only the data they are explicitly authorized to see. Kudu supports this by leveraging policies defined in Apache Sentry 2.2 and later. Starting with Kudu 1.12.0, Kudu now supports fine-grained authorization by leveraging policies defined in Apache Ranger 2.1 and later.

Note: Since support for Apache Sentry authorization has been deprecated since Kudu 1.12.0 and may be completely removed in the future, fine-grained authorization via Apache Ranger is preferred going forward.

Note: Fine-grained authorization policies are not enforced when accessing the web UI. User data may appear on various pages of the web UI (e.g. in logs, metrics, scans, etc.). As such, it is recommended to either limit access to the web UI ports, or redact or disable the web UI entirely, as desired.

Apache Ranger

Apache Ranger models tabular objects are stored in a Kudu cluster in the following hierarchy: Database, Table, Column.

Note: Ranger allows you to add separate service repositories to manage privileges for different Kudu clusters. Depending on the value of the ranger.plugin.kudu.service.name configuration in the Ranger client, Kudu knows which service repository to connect to. For more details about Ranger service repository, see the Apache Ranger documentation.

Database: Kudu does not have the concept of a database. Therefore, a database is indicated as a prefix of table names with the format <database>.<table>. Since Kudu's only restriction on table names is that they be valid UTF-8 encoded strings, Kudu considers special characters to be valid parts of database or table names. For example, if a managed Kudu table created from Impala is named impala::bar.foo, its database will be impala::bar.

Table: Is a single Kudu table.

Column: Is a column within a Kudu table.

In Ranger, privileges are also associated with specific actions. Access to Kudu tables may rely on privileges on the following actions:

- ALTER
- CREATE
- DELETE
- DROP
- INSERT
- UPDATE
- SELECT
- ALL
- METADATA

If a user has the ALL privileges on a given table specifically, then that user has all of the above privileges on the table. METADATA privilege is modeled as any privilege. If a user has any privilege on a given table, that user has METADATA privileges on the table, i.e. a privilege granted on any action on a table implies that the user has the METADATA privilege on that table.

In term of privilege evaluation Ranger doesn't have the concept of hierarchical implication. To be more specific, if a user has SELECT privilege on a database, it does not imply that user has SELECT privileges on every table belonging to that database. On the other hand, Ranger supports privilege wildcard matching. For example, db=a->table=* matches all the tables that belong to database a. Therefore, in Ranger users actually need the SELECT privilege on db=a->table=*->column=* to match the semantics of the SELECT privilege on db=a in Sentry.

Nevertheless, with Ranger integration, when a Kudu master receives a request, it consults Ranger to determine what privileges a user has. And the required policies documented in the <<security.adoc#policy-for-kudu-masters, policy section>> are enforced to determine whether the user is authorized to perform the requested action or not.

Note: Even though Kudu table names remain case sensitive with Ranger integration, policies authorization is considered case-insensitive.

Authorization tokens

Rather than having every tablet server communicate directly with the underlying authorization service (Ranger), privileges are propagated and checked via authorization tokens. These tokens encapsulate what privileges a user has on a given table. Tokens are generated by the master and returned to Kudu clients upon opening a Kudu table. Kudu clients automatically attach authorization tokens when sending requests to tablet servers.

Authorization tokens are a means to limiting the number of nodes directly accessing the authorization service to retrieve privileges. As such, since the expected number of tablet servers in a cluster is much higher than the number of Kudu masters, they are only used to authorize requests sent to tablet servers. Kudu masters fetch privileges directly from the authorization service or cache.

Similar to the validity interval for authentication tokens, to limit the window of potential unwanted access if a token becomes compromised, authorization tokens are valid for five minutes by default. The acquisition and renewal of a token is hidden from the user, as Kudu clients automatically retrieve new tokens when existing tokens expire.

When a tablet server that has been configured to enforce fine-grained access control receives a request, it checks the privileges in the attached token, rejecting it if the privileges are not sufficient to perform the requested operation, or if it is invalid (e.g. expired).

Trusted users

It may be desirable to allow certain users to view and modify any data stored in Kudu. Such users can be specified via the --trusted_user_acl master configuration. Trusted users can perform any operation that would otherwise require fine-grained privileges, without Kudu consulting the authorization service.

Additionally, some services that interact with Kudu may authorize requests on behalf of their end users. For example, Apache Impala authorizes queries on behalf of its users, and sends requests to Kudu as the Impala service user, commonly "impala". Since Impala authorizes requests on its own, to avoid extraneous communication between the authorization service and Kudu, the Impala service user should be listed as a trusted user.

Note: When accessing Kudu through Impala, Impala enforces its own fine-grained authorization policy. This policy is similar to Kudu's and can be found in the Impala authorization documentation.

Configuring Kudu's integration with Apache Ranger

Apache Ranger has wider adoption and provides a more comprehensive security features (such as attribute based access control, audit, etc) than Sentry. This topic provides information to configure Kudu with Apache Ranger.

About this task

Note:

- Ranger is often configured with Kerberos authentication.
- Sentry integration can not be enabled at the same time with Ranger integration.

Procedure

1. After building Kudu from source, find the kudu-subprocess.jar under the build directory, for example build/release/bin.

   Note its path, as it is the one to the JAR file containing the Ranger subprocess, which houses the Ranger client that Kudu will use to communicate with the Ranger server.
2. Use the kudu table list tool to find any table names in the cluster that are not Ranger-compatible, which are names that begin or end with a period (.). Also check that there are no two table names that only differ by case, since authorization is case-insensitive.

   For those tables that do not comply with the requirements, use the kudu table rename_table tool to rename the tables.
3. Create a Ranger client ranger-kudu-security.xml configuration file, and note down the directory containing this file.

```
<property>
  <name>ranger.plugin.kudu.policy.cache.dir</name>
  <value>policycache</value>
  <description>Directory where Ranger policies are cached after successful
 retrieval from the Ranger service</description>
</property>
<property>
  <name>ranger.plugin.kudu.service.name</name>
  <value>kudu</value>
  <description>Name of the Ranger service repository storing policies for
this Kudu cluster</description>
</property>
<property>
  <name>ranger.plugin.kudu.policy.rest.url</name>
  <value>http://host:port</value>
  <description>Ranger Admin URL</description>
</property>
<property>
  <name>ranger.plugin.kudu.policy.source.impl</name>
  <value>org.apache.ranger.admin.client.RangerAdminRESTClient</value>
  <description>Ranger client implementation to retrieve policies from the
 Ranger service</description>
</property>
<property>
  <name>ranger.plugin.kudu.policy.rest.ssl.config.file</name>
  <value>ranger-kudu-policymgr-ssl.xml</value>
  <description>Path to the file containing SSL details to connect Ranger A
dmin</description>
</property>
<property>
  <name>ranger.plugin.kudu.policy.pollIntervalMs</name>
  <value>30000</value>
  <description>Ranger client policy polling interval</description>
</property>
```

4. When Secure Socket Layer (SSL) is enabled for Ranger Admin, add the ranger-kudu-policymgr-ssl.xml file to the Ranger client configuration directory with the following configurations:

```
<property>
  <name>xasecure.policymgr.clientssl.keystore</name>
  <value>[/path/to/keystore].jks</value>
  <description>Java keystore files</description>
</property>
<property>
  <name>xasecure.policymgr.clientssl.keystore.credential.file</name>
  <value>jceks://file/[path/to/credentials].jceks</value>
  <description>Java keystore credential file</description>
</property>
<property>
  <name>xasecure.policymgr.clientssl.truststore</name>
  <value>[/path/to/truststore].jks</value>
  <description>Java truststore file</description>
</property>
<property>
  <name>xasecure.policymgr.clientssl.truststore.credential.file</name>
  <value>jceks://file/[path/to/credentials].jceks</value>
  <description>Java truststore credential file</description>
</property>
```

5. Set the following configurations on the Kudu master:

```
# The path to directory containing Ranger client configuration. This exa
mple
# assumes the path is '/kudu/ranger-config'.
--ranger_config_path=/kudu/ranger-config

# The path where the Java binary was installed. This example assumes
# '$JAVA_HOME=/usr/local'
--ranger_java_path=/usr/local/bin/java

# The path to the JAR file containing the Ranger subprocess. This example
# assumes '$KUDU_HOME=/kudu'
--ranger_jar_path=/kudu/build/release/bin/kudu-subprocess.jar
# This example ACL setup allows the 'impala' user to access all data st
ored in
# Kudu, assuming Impala will authorize requests on its own. The 'kudu' u
ser is
# also granted access to all Kudu data, which may facilitate testing and
# debugging (such as running the 'kudu cluster ksck' tool).
--trusted_user_acl=impala,kudu
```

6. Set the following configurations on the tablet servers:

```
--tserver_enforce_access_control=true
```

7. Add a Kudu service repository with the following configurations via the Ranger Admin web UI:

```
# This example setup configures the Kudu service user as a privileged user
 to be
# able to retrieve authorization policies stored in Ranger.

<property>
  <name>policy.download.auth.users</name>
  <value>kudu</value>
</property>
```

Ranger client caching

Ranger provides client side cache that use privileges and can periodically poll the privilege store for any changes. When a change is detected, the cache is automatically updated.

Update the ranger.plugin.kudu.policy.pollIntervalMs property specified in ranger-kudu-security.xml to set how often the Ranger client cache refreshes the privileges from the Ranger service.

Policy for Kudu masters

The following authorization policy is enforced by Kudu masters:

Table 2: Authorization Policy for Masters

Operation	Required Privilege
CreateTable	CREATE ON DATABASE
CreateTable with a different owner specified than the requesting user	ALL ON DATABASE with the Sentry GRANT OPTION.
DeleteTable	DROP ON TABLE
AlterTable (with no rename)	ALTER ON TABLE
AlterTable (with rename)	ALL ON TABLE <old-table> and CREATE ON DATABASE <new-database>
IsCreateTableDone	METADATA ON TABLE
IsAlterTableDone	METADATA ON TABLE
ListTables	METADATA ON TABLE
GetTableLocations	METADATA ON TABLE
GetTableSchema	METADATA ON TABLE
GetTabletLocations	METADATA ON TABLE

Policy for Kudu tablet servers

The following authorization policy is enforced by Kudu tablet servers:

Table 3: Authorization Policy for Tablet Servers

Operation	Required Privilege
Scan	SELECT ON TABLE, or METADATA ON TABLE and SELECT ON COLUMN for each projected column and each predicate column
Scan (no projected columns, equivalent to COUNT(*))	SELECT ON TABLE, or SELECT ON COLUMN for each column in the table
Scan (with virtual columns)	SELECT ON TABLE, or SELECT ON COLUMN for each column in the table
Scan (in ORDERED mode)	<privileges required for a Scan> and SELECT ON COLUMN for each primary key column
Insert	INSERT ON TABLE
Update	UPDATE ON TABLE
Upsert	INSERT ON TABLE and UPDATE ON TABLE
Delete	DELETE ON TABLE
SplitKeyRange	SELECT ON COLUMN for each primary key column and SELECT ON COLUMNfor each projected column

Operation	Required Privilege
Checksum	User must be configured in --superuser_acl
ListTablets	User must be configured in --superuser_acl

Note: Unlike Impala, Kudu only supports all-or-nothing access to a table's schema, rather than showing only authorized columns.

Encryption

Kudu allows you to use TLS to encrypt all communications among servers, and between clients and servers.

Configure TLS encryption on Kudu servers using the --rpc_encryption flag, which can be set to one of the following options:

- required - Kudu will reject unencrypted connections.
- optional - Kudu will attempt to use encryption, but will allow unencrypted connections.
- disabled - Kudu will not use encryption.

By default, the flag is set to optional. To secure your cluster, set --rpc_encryption to required.

Note: Kudu will automatically turn off encryption on local loopback connections, since traffic from these connections is never exposed externally. This allows locality-aware compute frameworks, such as Spark and Impala, to avoid encryption overhead, while still ensuring data confidentiality.

Web UI encryption

The Kudu web UI can be configured to use secure HTTPS encryption by providing each server with TLS certificates. Use the --webserver_certificate_file and --webserver_private_key_file properties to specify the certificate and private key to be used for communication.

Alternatively, you can choose to completely disable the web UI by setting --webserver_enabled flag to false on the Kudu servers.

Web UI redaction

To prevent sensitive data from being included in the web UI, all row data is redacted. Table metadata, such as table names, column names, and partitioning information is not redacted. Alternatively, you can choose to completely disable the web UI by setting the --webserver_enabled flag to false on the Kudu servers.

Note: Disabling the web UI will also disable REST endpoints such as /metrics. Monitoring systems rely on these endpoints to gather metrics data.

Log redaction

To prevent sensitive data from being included in Kudu server logs, all row data will be redacted. You can turn off log redaction using the --redact flag.

Configuring a secure Kudu cluster using Cloudera Manager

First you need to enable Kerberos authentication and RPC encryption. Next, configure coarse-grained authorization with ALCs. Lastly, configure HTTPS encryption for both the Kudu master and tablet server web UIs.

Enabling Kerberos authentication and RPC encryption

You must aleady have a secure Cloudera Manager cluster with Kerberos authentication enabled.

Procedure

1. In Cloudera Manager, navigate to Kudu Configuration .
2. In the Search field, type Kerberos to show the relevant properties.

3. Find and edit the following properties according to your cluster configuration:

Field	Usage Notes
Kerberos Principal	Set to the default principal, kudu. Currently, Kudu does not support configuring a custom service principal for Kudu processes.
Enable Secure Authentication And Encryption	Select this checkbox to enable authentication and RPC encryption between all Kudu clients and servers, as well as between individual servers. Only enable this property after you have configured Kerberos.

4. Click Save Changes.
5. You will see an error message that tells you the Kudu keytab is missing. To generate the keytab, go to the top navigation bar and click Administration Security .
6. Go to the Kerberos Credentials tab. On this page you will see a list of the existing Kerberos principals for services running on the cluster.
7. Click Generate Missing Credentials. Once the Generate Missing Credentials command has finished running, you will see the Kudu principal added to the list.

Configuring coarse-grained authorization with ACLs

The coarse-grained authorization can be configured with the following two ACLs: the Superuser Access Control List and the User Access Control List. The Superuser ACL is the list of all the superusers that can access the cluster. User-level access can be controlled by using the User ACL. By default, all the users can access the clusters. But when you enable authentication using Kerberos, only the users who are able to authenticate successfully can access the cluster.

Procedure

1. Go to the Kudu service.
2. Click the Configuration tab.
3. Select Category Security .
4. In the Search field, type ACL to show the relevant properties.
5. Edit the following properties according to your cluster configuration:

Field	Usage Notes
Superuser Access Control List	Add a comma-separated list of superusers who can access the cluster. By default, this property is left blank. '*' indicates that all authenticated users will be given superuser access.
User Access Control List	Add a comma-separated list of users who can access the cluster. By default, this property is set to '*'. The default value of '*' allows all users access to the cluster. However, if authentication is enabled, this will restrict access to only those users who are able to successfully authenticate using Kerberos. Unauthenticated users on the same network as the Kudu servers will be unable to access the cluster. Add the impala user to this list to allow Impala to query data in Kudu. You might choose to add any other relevant usernames if you want to give access to Spark Streaming jobs.

6. Click Save Changes.

Enabling Ranger authorization

You can configure fine-grained authorization using Apache Ranger. This topic provides the steps to enable Kudu's integration with Ranger from Cloudera Manager.

Procedure

1. From Cloudera Manager, go to Clusters Kudu Configurations .
2. Select the Ranger Service with which Kudu should authorize requests.

3. If Ranger high-availability is enabled for the cluster, add a Kudu service repository with the following configurations through the Ranger Admin web UI is required:

```
# This example setup configures the Kudu service user as a privileged user
 to be
# able to retrieve authorization policies stored in Ranger.

<property>
  <name>policy.download.auth.users</name>
  <value>kudu</value>
</property>
```

The name of the added Kudu service repository needs to match the one specified in ranger.plugin.kudu.service.name of the Ranger client ranger-kudu-security.xml configuration file.

Note: When a Kudu client opens a table, the Kudu Master will authorize all possible actions the user may want to perform on the given table (ALL, and if it's not allowed, then INSERT, SELECT, UDPATE, DELETE). This results in auditing these requests when a client opens a table, even if they'll never do any of these operations.

Configuring HTTPS encryption for the Kudu master and tablet server web UIs

Lastly, you enable TLS/SSL encryption (over HTTPS) for browser-based connections to both the Kudu master and tablet server web UIs.

Procedure

1. Go to the Kudu service.
2. Click the Configuration tab.
3. Select Category Security .
4. In the Search field, type TLS/SSL to show the relevant properties.
5. Edit the following properties according to your cluster configuration:

Field	Usage Notes
Master TLS/SSL Server Private Key File (PEM Format)	Set to the path containing the Kudu master host's private key (PEM-format). This is used to enable TLS/SSL encryption (over HTTPS) for browser-based connections to the Kudu master web UI.
Tablet Server TLS/SSL Server Private Key File (PEM Format)	Set to the path containing the Kudu tablet server host's private key (PEM-format). This is used to enable TLS/SSL encryption (over HTTPS) for browser-based connections to Kudu tablet server web UIs.
Master TLS/SSL Server Certificate File (PEM Format)	Set to the path containing the signed certificate (PEM-format) for the Kudu master host's private key (set in Master TLS/SSL Server Private Key File). The certificate file can be created by concatenating all the appropriate root and intermediate certificates required to verify trust.
Tablet Server TLS/SSL Server Certificate File (PEM Format)	Set to the path containing the signed certificate (PEM-format) for the Kudu tablet server host's private key (set in Tablet Server TLS/SSL Server Private Key File). The certificate file can be created by concatenating all the appropriate root and intermediate certificates required to verify trust.
Enable TLS/SSL for Master Server	Enables HTTPS encryption on the Kudu master web UI.
Enable TLS/SSL for Tablet Server	Enables HTTPS encryption on the Kudu tablet server Web UIs.

6. Click Save Changes.

Configuring a secure Kudu cluster using the command line

You should set the configuration parameters on all the servers (master and tablet servers) to ensure that a Kudu cluster is secure.

Important: Follow these command-line instructions on systems that do not use Cloudera Manager. If you are using Cloudera Manager, see Configuring a secure Kudu cluster using Cloudera Manager on page 41.

```
# Connection Security
#--------------------
--rpc_authentication=required
--rpc_encryption=required
--keytab_file=<path-to-kerberos-keytab>
# Web UI Security
#--------------------
--webserver_certificate_file=<path-to-cert-pem>
--webserver_private_key_file=<path-to-key-pem>
# optional
--webserver_private_key_password_cmd=<password-cmd>

# If you prefer to disable the web UI entirely:
--webserver_enabled=false

# Coarse-grained authorization
#-----------------------------
# This example ACL setup allows the 'impala' user as well as the
# 'etl_service_account' principal access to all data in the
# Kudu cluster. The 'hadoopadmin' user is allowed to use administrative
# tooling. Note that by granting access to 'impala', other users
# may access data in Kudu via the Impala service subject to its own
# authorization rules.
--user_acl=impala,etl_service_account
--admin_acl=hadoopadmin
```

More information about these flags can be found in the configuration reference documentation.

Related Information

Apache Kudu Configuration Reference

Apache Kudu background maintenance tasks

Kudu relies on running background tasks for many important maintenance activities. These tasks include flushing data from memory to disk, compacting data to improve performance, freeing up disk space, and more.

Maintenance manager

The maintenance manager schedules and runs background tasks. At any given point in time, the maintenance manager is prioritizing the next task based on improvements needed at that moment, such as relieving memory pressure, improving read performance, or freeing up disk space.

The number of worker threads dedicated to running background tasks can be controlled by setting --maintenance_manager_num_threads.

With Kudu 1.4, the maintenance manager features improved utilization of the configured maintenance threads. Previously, maintenance work would only be scheduled a maximum of 4 times per second, but now maintenance work will be scheduled immediately whenever any configured thread is available. Make sure that the --maintenance_manager_num_threads property is set to at most a 1:3 ratio for Maintenance Manager threads to the number of data directories (for spinning disks). This will improve the throughput of write-heavy workloads.

Flushing data to disk

Flushing data from memory to disk relieves memory pressure and can improve read performance by switching from a write-optimized, row-oriented in-memory format in the MemRowSet, to a read-optimized, column-oriented format on disk.

Background tasks that flush data include FlushMRSOp and FlushDeltaMemStoresOp. The metrics associated with these operations have the prefix flush_mrs and flush_dms, respectively.

With Kudu 1.4, the maintenance manager aggressively schedules flushes of in-memory data when memory consumption crosses 60 percent of the configured process-wide memory limit. The backpressure mechanism which begins to throttle client writes was also adjusted to not begin throttling until memory consumption reaches 80 percent of the configured limit. These two changes together result in improved write throughput, more consistent latency, and fewer timeouts due to memory exhaustion.

Compacting on-disk data

Kudu constantly performs several compaction tasks in order to maintain consistent read and write performance over time.

- A merging compaction, which combines multiple DiskRowSets together into a single DiskRowSet, is run by CompactRowSetsOp.
- Kudu also runs two types of delta store compaction operations: MinorDeltaCompactionOp and MajorDeltaCompactionOp.

  For more information on what these compaction operations do, see the Kudu Tablet design document.

The metrics associated with these tasks have the prefix compact_rs, delta_minor_compact_rs, and delta_major_compact_rs, respectively.

Related Information

Kudu Tablet design document

Write-ahead log garbage collection

Kudu maintains a write-ahead log (WAL) per tablet that is split into discrete fixed-size segments. A tablet periodically rolls the WAL to a new log segment when the active segment reaches a size threshold (configured by the --log_segment_size_mb property).

In order to save disk space and decrease startup time, a background task called LogGCOp attempts to garbage-collect (GC) old WAL segments by deleting them from disk once it is determined that they are no longer needed by the local node for durability.

The metrics associated with this background task have the prefix log_gc.

Tablet history garbage collection and the ancient history mark

Kudu uses a multiversion concurrency control (MVCC) mechanism to ensure that snapshot scans can proceed isolated from new changes to a table. Therefore, periodically, old historical data should be garbage-collected (removed) to free up disk space. While Kudu never removes rows or data that are visible in the latest version of the data, Kudu does remove records of old changes that are no longer visible.

The specific threshold in time (in the past) beyond which historical MVCC data becomes inaccessible and is free to be deleted is called the ancient history mark (AHM). The AHM can be configured by setting the --tablet_history_max_age_sec property.

There are two background tasks that remove historical MVCC data older than the AHM:

- The one that runs the merging compaction, called CompactRowSetsOp (see above).
- A separate background task deletes old undo delta blocks, called UndoDeltaBlockGCOp. Running UndoDeltaBlockGCOp reduces disk space usage in all workloads, but particularly in those with a higher volume of updates or upserts. The metrics associated with this background task have the prefix undo_delta_block.