Introducing Apache Kudu

Kudu is a columnar storage manager developed for the Apache Hadoop platform. Kudu shares the common technical properties of Hadoop ecosystem applications: it runs on commodity hardware, is horizontally scalable, and supports highly available operation.

Kudu’s design sets it apart. Some of Kudu’s benefits include:

* Fast processing of OLAP workloads.
* Integration with MapReduce, Spark and other Hadoop ecosystem components.
* Tight integration with Apache Impala (incubating), making it a good, mutable alternative to using HDFS with Apache Parquet.
* Strong but flexible consistency model, allowing you to choose consistency requirements on a per-request basis, including the option for strict-serializable consistency.
* Strong performance for running sequential and random workloads simultaneously.
* Easy to administer and manage with Cloudera Manager.
* High availability. Tablet Servers and Masters use the [Raft Consensus Algorithm](http://kudu.apache.org/docs/index.html#raft), which ensures that as long as more than half the total number of replicas is available, the tablet is available for reads and writes. For instance, if 2 out of 3 replicas or 3 out of 5 replicas are available, the tablet is available.

Reads can be serviced by read-only follower tablets, even in the event of a leader tablet failure.

* Structured data model.

By combining all of these properties, Kudu targets support for families of applications that are difficult or impossible to implement on current generation Hadoop storage technologies. A few examples of applications for which Kudu is a great solution are:

* Reporting applications where newly-arrived data needs to be immediately available for end users
* Time-series applications that must simultaneously support:
  + queries across large amounts of historic data
  + granular queries about an individual entity that must return very quickly
* Applications that use predictive models to make real-time decisions with periodic refreshes of the predictive model based on all historic data

For more information about these and other scenarios, see [Example Use Cases](http://kudu.apache.org/docs/index.html#kudu_use_cases).

[Kudu-Impala Integration Features](http://kudu.apache.org/docs/index.html#_kudu_impala_integration_features)

**CREATE/ALTER/DROP TABLE**

Impala supports creating, altering, and dropping tables using Kudu as the persistence layer. The tables follow the same internal / external approach as other tables in Impala, allowing for flexible data ingestion and querying.

**INSERT**

Data can be inserted into Kudu tables in Impala using the same syntax as any other Impala table like those using HDFS or HBase for persistence.

**UPDATE / DELETE**

Impala supports the UPDATE and DELETE SQL commands to modify existing data in a Kudu table row-by-row or as a batch. The syntax of the SQL commands is chosen to be as compatible as possible with existing standards. In addition to simple DELETE or UPDATE commands, you can specify complex joins with a FROM clause in a subquery.

**Flexible Partitioning**

Similar to partitioning of tables in Hive, Kudu allows you to dynamically pre-split tables by hash or range into a predefined number of tablets, in order to distribute writes and queries evenly across your cluster. You can partition by any number of primary key columns, by any number of hashes, and an optional list of split rows. See [Schema Design](http://kudu.apache.org/docs/schema_design.html).

**Parallel Scan**

To achieve the highest possible performance on modern hardware, the Kudu client used by Impala parallelizes scans across multiple tablets.

**High-efficiency queries**

Where possible, Impala pushes down predicate evaluation to Kudu, so that predicates are evaluated as close as possible to the data. Query performance is comparable to Parquet in many workloads.

For more details regarding querying data stored in Kudu using Impala, please refer to the Impala documentation.

[Concepts and Terms](http://kudu.apache.org/docs/index.html#_concepts_and_terms)

**Columnar Data Store**

Kudu is a *columnar data store*. A columnar data store stores data in strongly-typed columns. With a proper design, it is superior for analytical or data warehousing workloads for several reasons.

**Read Efficiency**

For analytical queries, you can read a single column, or a portion of that column, while ignoring other columns. This means you can fulfill your query while reading a minimal number of blocks on disk. With a row-based store, you need to read the entire row, even if you only return values from a few columns.

**Data Compression**

Because a given column contains only one type of data, pattern-based compression can be orders of magnitude more efficient than compressing mixed data types, which are used in row-based solutions. Combined with the efficiencies of reading data from columns, compression allows you to fulfill your query while reading even fewer blocks from disk. See [Data Compression](http://kudu.apache.org/docs/schema_design.html#encoding)

**Table**

A *table* is where your data is stored in Kudu. A table has a schema and a totally ordered primary key. A table is split into segments called tablets.

**Tablet**

A *tablet* is a contiguous segment of a table, similar to a *partition* in other data storage engines or relational databases. A given tablet is replicated on multiple tablet servers, and at any given point in time, one of these replicas is considered the leader tablet. Any replica can service reads, and writes require consensus among the set of tablet servers serving the tablet.

**Tablet Server**

A *tablet server* stores and serves tablets to clients. For a given tablet, one tablet server acts as a leader, and the others act as follower replicas of that tablet. Only leaders service write requests, while leaders or followers each service read requests. Leaders are elected using [Raft Consensus Algorithm](http://kudu.apache.org/docs/index.html#raft). One tablet server can serve multiple tablets, and one tablet can be served by multiple tablet servers.

**Master**

The *master* keeps track of all the tablets, tablet servers, the [Catalog Table](http://kudu.apache.org/docs/index.html#catalog_table), and other metadata related to the cluster. At a given point in time, there can only be one acting master (the leader). If the current leader disappears, a new master is elected using [Raft Consensus Algorithm](http://kudu.apache.org/docs/index.html#raft).

The master also coordinates metadata operations for clients. For example, when creating a new table, the client internally sends the request to the master. The master writes the metadata for the new table into the catalog table, and coordinates the process of creating tablets on the tablet servers.

All the master’s data is stored in a tablet, which can be replicated to all the other candidate masters.

Tablet servers heartbeat to the master at a set interval (the default is once per second).

**Raft Consensus Algorithm**

Kudu uses the [Raft consensus algorithm](https://raft.github.io/) as a means to guarantee fault-tolerance and consistency, both for regular tablets and for master data. Through Raft, multiple replicas of a tablet elect a *leader*, which is responsible for accepting and replicating writes to *follower* replicas. Once a write is persisted in a majority of replicas it is acknowledged to the client. A given group of N replicas (usually 3 or 5) is able to accept writes with at most (N - 1)/2 faulty replicas.

**Catalog Table**

The *catalog table* is the central location for metadata of Kudu. It stores information about tables and tablets. The catalog table may not be read or written directly. Instead, it is accessible only via metadata operations exposed in the client API.

The catalog table stores two categories of metadata:

**Tables**

table schemas, locations, and states

**Tablets**

the list of existing tablets, which tablet servers have replicas of each tablet, the tablet’s current state, and start and end keys.

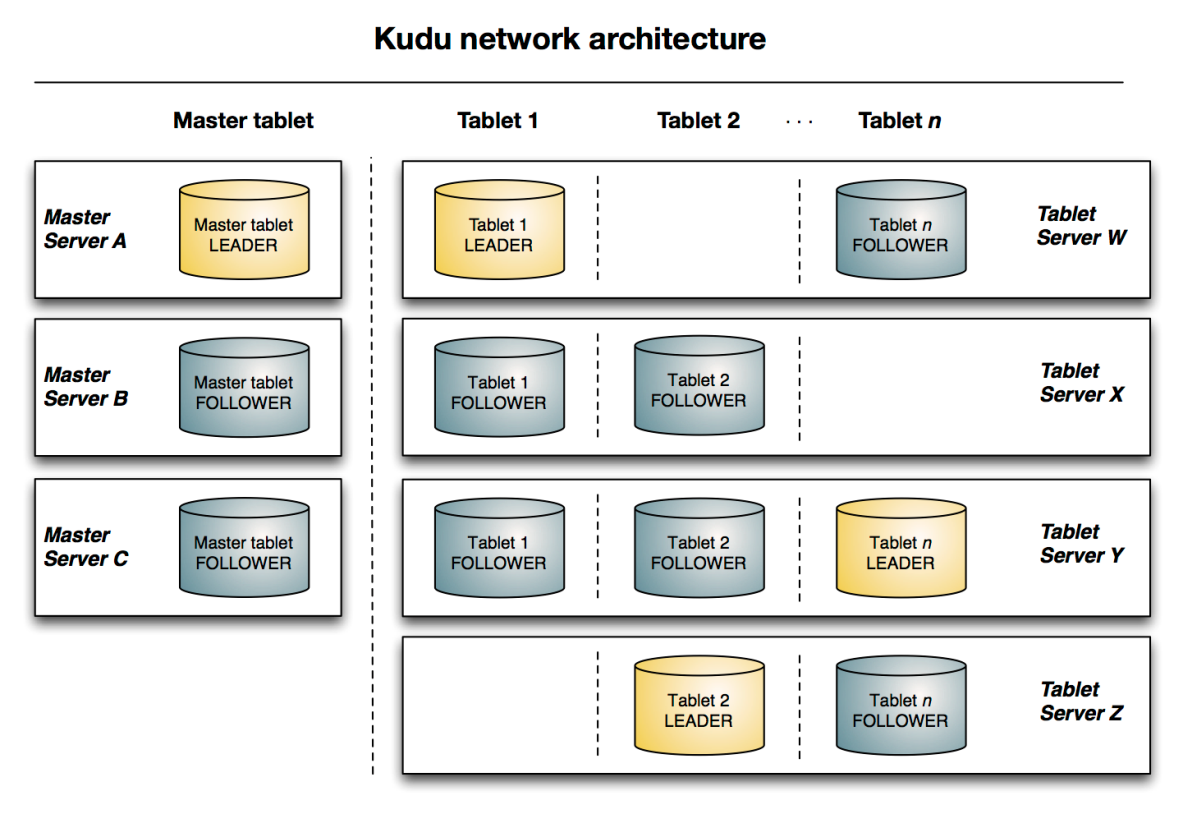
**Logical Replication**

Kudu replicates operations, not on-disk data. This is referred to as *logical replication*, as opposed to *physical replication*. This has several advantages:

* Although inserts and updates do transmit data over the network, deletes do not need to move any data. The delete operation is sent to each tablet server, which performs the delete locally.
* Physical operations, such as compaction, do not need to transmit the data over the network in Kudu. This is different from storage systems that use HDFS, where the blocks need to be transmitted over the network to fulfill the required number of replicas.
* Tablets do not need to perform compactions at the same time or on the same schedule, or otherwise remain in sync on the physical storage layer. This decreases the chances of all tablet servers experiencing high latency at the same time, due to compactions or heavy write loads.

[Architectural Overview](http://kudu.apache.org/docs/index.html#_architectural_overview)

The following diagram shows a Kudu cluster with three masters and multiple tablet servers, each serving multiple tablets. It illustrates how Raft consensus is used to allow for both leaders and followers for both the masters and tablet servers. In addition, a tablet server can be a leader for some tablets, and a follower for others. Leaders are shown in gold, while followers are shown in blue.



[Example Use Cases](http://kudu.apache.org/docs/index.html#kudu_use_cases)

**Streaming Input with Near Real Time Availability**

A common challenge in data analysis is one where new data arrives rapidly and constantly, and the same data needs to be available in near real time for reads, scans, and updates. Kudu offers the powerful combination of fast inserts and updates with efficient columnar scans to enable real-time analytics use cases on a single storage layer.

**Time-series application with widely varying access patterns**

A time-series schema is one in which data points are organized and keyed according to the time at which they occurred. This can be useful for investigating the performance of metrics over time or attempting to predict future behavior based on past data. For instance, time-series customer data might be used both to store purchase click-stream history and to predict future purchases, or for use by a customer support representative. While these different types of analysis are occurring, inserts and mutations may also be occurring individually and in bulk, and become available immediately to read workloads. Kudu can handle all of these access patterns simultaneously in a scalable and efficient manner.

Kudu is a good fit for time-series workloads for several reasons. With Kudu’s support for hash-based partitioning, combined with its native support for compound row keys, it is simple to set up a table spread across many servers without the risk of "hotspotting" that is commonly observed when range partitioning is used. Kudu’s columnar storage engine is also beneficial in this context, because many time-series workloads read only a few columns, as opposed to the whole row.

In the past, you might have needed to use multiple data stores to handle different data access patterns. This practice adds complexity to your application and operations, and duplicates your data, doubling (or worse) the amount of storage required. Kudu can handle all of these access patterns natively and efficiently, without the need to off-load work to other data stores.

**Predictive Modeling**

Data scientists often develop predictive learning models from large sets of data. The model and the data may need to be updated or modified often as the learning takes place or as the situation being modeled changes. In addition, the scientist may want to change one or more factors in the model to see what happens over time. Updating a large set of data stored in files in HDFS is resource-intensive, as each file needs to be completely rewritten. In Kudu, updates happen in near real time. The scientist can tweak the value, re-run the query, and refresh the graph in seconds or minutes, rather than hours or days. In addition, batch or incremental algorithms can be run across the data at any time, with near-real-time results.

**Combining Data In Kudu With Legacy Systems**

Companies generate data from multiple sources and store it in a variety of systems and formats. For instance, some of your data may be stored in Kudu, some in a traditional RDBMS, and some in files in HDFS. You can access and query all of these sources and formats using Impala, without the need to change your legacy systems.

# Apache Kudu 1.4.0 Release Notes

## [New features](http://kudu.apache.org/docs/release_notes.html#rn_1.4.0_new_features)

* The C++ and Java client libraries now support the ability to alter the storage attributes (e.g. encoding and compression) and default value of existing columns. Additionally, it is now possible to rename a column which is part of a table’s primary key.
* The C++ client library now includes an experimental KuduPartitioner API which may be used to efficiently map rows to their associated partitions and hosts. This may be used to achieve better locality or distribution of writes in client applications.
* The Java client library now supports enabling fault tolerance on scanners. Fault tolerant scanners are able to transparently recover from concurrent server crashes at the cost of some performance overhead. See the Java API documentation for more details on usage.
* The kudu command line tool now includes a new advanced administrative command kudu remote\_replica unsafe\_change\_config. This command may be used to force a tablet to perform an unsafe change of its Raft replication configuration. This can be used to recover from scenarios such as a loss of a majority of replicas, at the risk of losing edits.
* The kudu command line tool now includes the kudu fs check command which performs various offline consistency checks on the local on-disk storage of a Kudu Tablet Server or Master. In addition to detecting various inconsistencies or corruptions, it can also detect and remove data blocks that are no longer referenced by any tablet but were not fully removed from disk due to a crash or a bug in prior versions of Kudu.
* The kudu command line tool can now be used to list the addresses and identifiers of the servers in the cluster using either kudu master list or kudu tserver list.
* Kudu 1.4 now includes the optional ability to compute, store, and verify checksums on all pieces of data stored on a server. Prior versions only performed checksums on certain portions of the stored data. This feature is not enabled by default since it makes a backward-incompatible change to the on-disk formats and thus prevent downgrades. Kudu 1.5 will enable the feature by default.

## [Optimizations and improvements](http://kudu.apache.org/docs/release_notes.html#_optimizations_and_improvements)

* kudu cluster ksck now detects and reports new classes of inconsistencies and issues. In particular, it is better able to detect cases where a configuration change such as a replica eviction or addition is pending but is unable to be committed. It also now properly detects and reports cases where a tablet has no elected leader.
* The default size for Write Ahead Log (WAL) segments has been reduced from 64MB to 8MB. Additionally, in the case that all replicas of a tablet are fully up to date and data has been flushed from memory, servers will now retain only a single WAL segment rather than two. These changes are expected to reduce the average consumption of disk space on the configured WAL disk by 16x, as well as improve the startup speed of tablet servers by reducing the number and size of WAL segments that need to be re-read.
* The default on-disk storage system used by Kudu servers (Log Block Manager) has been improved to compact its metadata and remove dead containers. This compaction and garbage collection occurs only at startup. Thus, the first startup after upgrade is expected to be longer than usual, and subsequent restarts should be shorter.
* The usability of the Kudu web interfaces has been improved, particularly for the case where a server hosts many tablets or a table has many partitions. Pages that list tablets now include a top-level summary of tablet status and show the complete list under a toggleable section.
* The Maintenance Manager has been improved to improve 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. This can improve the throughput of write-heavy workloads.
* The Maintenance Manager will now aggressively schedule flushes of in-memory data when memory consumption crosses 60% of the configured process-wide memory limit. The backpressure mechanism which begins to throttle client writes has been accordingly adjusted to not begin throttling until reaching 80% of the configured limit. These two changes together result in improved write throughput, more consistent latency, and fewer timeouts due to memory exhaustion.
* Many performance improvements were made to write performance. Applications which send large batches of writes to Kudu should see substantially improved throughput in Kudu 1.4.
* Several improvements were made to reduce the memory consumption of Kudu Tablet Servers which hold large volumes of data. The specific amount of memory saved varies depending on workload, but the expectation is that approximately 350MB of excess peak memory usage has been eliminated per TB of data stored.
* The number of threads used by the Kudu Tablet Server has been reduced. Previously, each tablet used a dedicated thread to append to its WAL. Those threads now automatically stop running if there is no activity on a given tablet for a short period of time.

## [Fixed Issues](http://kudu.apache.org/docs/release_notes.html#rn_1.4.0_fixed_issues)

* [KUDU-2020](https://issues.apache.org/jira/browse/KUDU-2020) Fixed an issue where re-replication after a failure would proceed significantly slower than expected. This bug caused many tablets to be unnecessarily copied multiple times before successfully being considered re-replicated, resulting in significantly more network and IO bandwidth usage than expected. Mean time to recovery on clusters with large amounts of data is improved by up to 10x by this fix.
* [KUDU-1982](https://issues.apache.org/jira/browse/KUDU-1982) Fixed an issue where the Java client would call NetworkInterface.getByInetAddress very often, causing performance problems particularly on Windows where this function can be quite slow.
* [KUDU-1755](https://issues.apache.org/jira/browse/KUDU-1755) Improved the accuracy of the on\_disk\_size replica metrics to include the size consumed by bloom filters, primary key indexes, and superblock metadata, and delta files. Note that, because the size metric is now more accurate, the reported values are expected to increase after upgrading to Kudu 1.4. This does not indicate that replicas are using more space after the upgrade; rather, it is now accurately reporting the amount of space that has always been used.
* [KUDU-1192](https://issues.apache.org/jira/browse/KUDU-1192) Kudu servers will now periodically flush their log messages to disk even if no WARNING-level messages have been logged. This makes it easier to tail the logs to see progress output during normal startup.
* [KUDU-1999](https://issues.apache.org/jira/browse/KUDU-1999) Fixed the ability to run Spark jobs in "cluster" mode against Kudu clusters secured by Kerberos.

## [Wire Protocol compatibility](http://kudu.apache.org/docs/release_notes.html#rn_1.4.0_wire_compatibility)

Kudu 1.4.0 is wire-compatible with previous versions of Kudu:

* Kudu 1.4 clients may connect to servers running Kudu 1.0 or later. If the client uses features that are not available on the target server, an error will be returned.
* Kudu 1.0 clients may connect to servers running Kudu 1.4 with the exception of the below-mentioned restrictions regarding secure clusters.
* Rolling upgrade between Kudu 1.3 and Kudu 1.4 servers is believed to be possible though has not been sufficiently tested. Users are encouraged to shut down all nodes in the cluster, upgrade the software, and then restart the daemons on the new version.

The authentication features introduced in Kudu 1.3 place the following limitations on wire compatibility between Kudu 1.4 and versions earlier than 1.3:

* If a Kudu 1.4 cluster is configured with authentication or encryption set to "required", clients older than Kudu 1.3 will be unable to connect.
* If a Kudu 1.4 cluster is configured with authentication and encryption set to "optional" or "disabled", older clients will still be able to connect.

## [Incompatible Changes in Kudu 1.4.0](http://kudu.apache.org/docs/release_notes.html#rn_1.4.0_incompatible_changes)

* Kudu servers, by default, will now only allow unencrypted or unauthenticated connections from trusted subnets, which are private networks (127.0.0.0/8,10.0.0.0/8,172.16.0.0/12, 192.168.0.0/16,169.254.0.0/16) and local subnets of all local network interfaces. Unencrypted or unauthenticated connections from publicly routable IPs will be rejected, even if encryption and authentication are not configured.

The trusted subnets can be configured using the --trusted\_subnets flag, which can be set to IP blocks represented in CIDR notation separated by comma. Set it to '0.0.0.0/0' to allow unauthenticated connections from all remote IP addresses. However, if network access is not otherwise restricted by a firewall, malicious users may be able to gain unauthorized access. This can be mitigated if authentication and encryption are configured to be required.

### [Client Library Compatibility](http://kudu.apache.org/docs/release_notes.html#rn_1.4.0_client_compatibility)

* The Kudu 1.4 Java client library is API- and ABI-compatible with Kudu 1.3. Applications written against Kudu 1.3 will compile and run against the Kudu 1.4 client library and vice-versa, unless one of the following newly added APIs is used:
  + [Async]KuduScannerBuilder.setFaultTolerant(…​)
  + New methods in AlterTableOptions: removeDefault, changeDefault, changeDesiredBlockSize, changeEncoding, changeCompressionAlgorithm
  + KuduClient.updateLastPropagatedTimestamp
  + KuduClient.getLastPropagatedTimestamp
  + New getters in PartialRow: getBoolean, getByte, getShort, getInt, getLong, getFloat, getDouble, getString, getBinaryCopy, getBinary, isNull, isSet.
* The Kudu 1.4 C++ client is API- and ABI-forward-compatible with Kudu 1.3. Applications written and compiled against the Kudu 1.3 client library will run without modification against the Kudu 1.4 client library. Applications written and compiled against the Kudu 1.4 client library will run without modification against the Kudu 1.3 client library unless they use one of the following new APIs:
  + KuduPartitionerBuilder
  + `KuduPartitioner
  + KuduScanner::SetRowFormatFlags (unstable API)
  + KuduScanBatch::direct\_data, KuduScanBatch::indirect\_data (unstable API)
* The Kudu 1.4 Python client is API-compatible with Kudu 1.3. Applications written against Kudu 1.3 will continue to run against the Kudu 1.4 client and vice-versa.

# Apache Kudu Quickstart

Follow these instructions to set up and run the Kudu VM, and start with Kudu, Kudu\_Impala, and CDH in minutes.

## [Get The Kudu Quickstart VM](http://kudu.apache.org/docs/quickstart.html#quickstart_vm)

### [Prerequisites](http://kudu.apache.org/docs/quickstart.html#_prerequisites)

1. Install [Oracle Virtualbox](https://www.virtualbox.org/). The VM has been tested to work with VirtualBox version 4.3 on Ubuntu 14.04 and VirtualBox version 5 on OSX 10.9. VirtualBox is also included in most package managers: apt-get, brew, etc.
2. After the installation, make sure that VBoxManage is in your PATH by using the which VBoxManage command.

### [Installation](http://kudu.apache.org/docs/quickstart.html#_installation)

To download and start the VM, execute the following command in a terminal window.

$ curl -s https://raw.githubusercontent.com/cloudera/kudu-examples/master/demo-vm-setup/bootstrap.sh | bash

This command downloads a shell script which clones the kudu-examples Git repository and then downloads a VM image of about 1.2GB size into the current working directory.[[1](http://kudu.apache.org/docs/quickstart.html#_footnote_1)] You can examine the script after downloading it by removing the | bash component of the command above. Once the setup is complete, you can verify that everything works by connecting to the guest via SSH:

$ ssh demo@quickstart.cloudera

The username and password for the demo account are both demo. In addition, the demo user has password-less sudo privileges so that you can install additional software or manage the guest OS. You can also access the kudu-examples as a shared folder in/home/demo/kudu-examples/ on the guest or from your VirtualBox shared folder location on the host. This is a quick way to make scripts or data visible to the guest.

You can quickly verify if Kudu and Impala are running by executing the following commands:

$ ps aux | grep kudu

$ ps aux | grep impalad

If you have issues connecting to the VM or one of the processes is not running, make sure to consult the [Troubleshooting](http://kudu.apache.org/docs/quickstart.html#trouble) section.

## [Load Data](http://kudu.apache.org/docs/quickstart.html#_load_data)

To practice some typical operations with Kudu and Impala, we’ll use the [San Francisco MTA GPS dataset](https://data.sfgov.org/Transportation/Raw-AVL-GPS-data/5fk7-ivit/data). This dataset contains raw location data transmitted periodically from sensors installed on the buses in the SF MTA’s fleet.

1. Download the sample data and load it into HDFS

First we’ll download the sample dataset, prepare it, and upload it into the HDFS cluster.

The SF MTA’s site is often a bit slow, so we’ve mirrored a sample CSV file from the dataset at <http://kudu-sample-data.s3.amazonaws.com/sfmtaAVLRawData01012013.csv.gz>

The original dataset uses DOS-style line endings, so we’ll convert it to UNIX-style during the upload process using tr.

$ wget http://kudu-sample-data.s3.amazonaws.com/sfmtaAVLRawData01012013.csv.gz

$ hdfs dfs -mkdir /sfmta

$ zcat sfmtaAVLRawData01012013.csv.gz | tr -d '\r' | hadoop fs -put - /sfmta/data.csv

1. Create a new external Impala table to access the plain text data. To connect to Impala in the virtual machine issue the following command:

ssh demo@quickstart.cloudera -t impala-shell

Now, you can execute the following commands:

CREATE EXTERNAL TABLE sfmta\_raw (

revision int,

report\_time string,

vehicle\_tag int,

longitude float,

latitude float,

speed float,

heading float

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LOCATION '/sfmta/'

TBLPROPERTIES ('skip.header.line.count'='1');

1. Validate if the data was actually loaded run the following command:

SELECT count(\*) FROM sfmta\_raw;

+----------+

| count(\*) |

+----------+

| 859086 |

+----------+

1. Next we’ll create a Kudu table and load the data. Note that we convert the string report\_time field into a unix-style timestamp for more efficient storage.

CREATE TABLE sfmta

PRIMARY KEY (report\_time, vehicle\_tag)

PARTITION BY HASH(report\_time) PARTITIONS 8

STORED AS KUDU

AS SELECT

UNIX\_TIMESTAMP(report\_time, 'MM/dd/yyyy HH:mm:ss') AS report\_time,

vehicle\_tag,

longitude,

latitude,

speed,

heading

FROM sfmta\_raw;

+------------------------+

| summary |

+------------------------+

| Inserted 859086 row(s) |

+------------------------+

Fetched 1 row(s) in 5.75s

The created table uses a composite primary key. See [Kudu Impala Integration](http://kudu.apache.org/docs/kudu_impala_integration.html#kudu_impala) for a more detailed introduction to the extended SQL syntax for Impala.

## [Read and Modify Data](http://kudu.apache.org/docs/quickstart.html#_read_and_modify_data)

Now that the data is stored in Kudu, you can run queries against it. The following query finds the data point containing the highest recorded vehicle speed.

SELECT \* FROM sfmta ORDER BY speed DESC LIMIT 1;

+-------------+-------------+--------------------+-------------------+-------------------+---------+

| report\_time | vehicle\_tag | longitude | latitude | speed | heading |

+-------------+-------------+--------------------+-------------------+-------------------+---------+

| 1357022342 | 5411 | -122.3968811035156 | 37.76665878295898 | 68.33300018310547 | 82 |

+-------------+-------------+--------------------+-------------------+-------------------+---------+

With a quick [Google search](https://www.google.com/search?q=122.3968811035156W+37.76665878295898N) we can see that this bus was traveling east on 16th street at 68MPH. At first glance, this seems unlikely to be true. Perhaps we do some research and find that this bus’s sensor equipment was broken and we decide to remove the data. With Kudu this is very easy to correct using standard SQL:

DELETE FROM sfmta WHERE vehicle\_tag = '5411';

-- Modified 1169 row(s), 0 row error(s) in 0.25s

## [Next steps](http://kudu.apache.org/docs/quickstart.html#_next_steps)

The above example showed how to load, query, and mutate a static dataset with Impala and Kudu. The real power of Kudu, however, is the ability to ingest and mutate data in a streaming fashion.

As an exercise to learn the Kudu programmatic APIs, try implementing a program that uses the [SFMTA XML data feed](http://www.nextbus.com/xmlFeedDocs/NextBusXMLFeed.pdf) to ingest this same dataset in real time into the Kudu table.

### [Troubleshooting](http://kudu.apache.org/docs/quickstart.html#trouble)

#### [Problems accessing the VM via SSH](http://kudu.apache.org/docs/quickstart.html#_problems_accessing_the_vm_via_ssh)

* Make sure the host has a SSH client installed.
* Make sure the VM is running, by running the following command and checking for a VM called kudu-demo:

$ VBoxManage list runningvms

* Verify that the VM’s IP address is included in the host’s /etc/hosts file. You should see a line that includes an IP address followed by the hostname quickstart.cloudera. To check the running VM’s IP address, use the VBoxManage command below.
* $ VBoxManage guestproperty get kudu-demo /VirtualBox/GuestInfo/Net/0/V4/IP
* Value: 192.168.56.100
* If you’ve used a Cloudera Quickstart VM before, your .ssh/known\_hosts file may contain references to the previous VM’s SSH credentials. Remove any references to quickstart.cloudera from this file.

#### [Failing with lack of SSE4.2 support when running inside VirtualBox](http://kudu.apache.org/docs/quickstart.html#_failing_with_lack_of_sse4_2_support_when_running_inside_virtualbox)

* Running Kudu currently requires a CPU that supports SSE4.2 (Nehalem or later for Intel). To pass through SSE4.2 support into the guest VM, refer to the [VirtualBox documentation](https://www.virtualbox.org/manual/ch09.html#sse412passthrough)

# Installing Apache Kudu

You can deploy Kudu on a cluster using packages or you can build Kudu from source. To run Kudu without installing anything, use the [Kudu Quickstart VM](http://kudu.apache.org/docs/quickstart.html#quickstart_vm).

|  |  |
| --- | --- |
|  | Kudu is currently easier to install and manage with [Cloudera Manager](http://www.cloudera.com/content/www/en-us/products/cloudera-manager.html), version 5.4.7 or newer. If you use Cloudera Manager, see also Cloudera’s [Kudu documentation](http://www.cloudera.com/documentation/kudu/latest/topics/kudu_installation.html). |

**Upgrading Kudu**

To upgrade Kudu from a previous version, see [Upgrade from a Previous Version of Kudu](http://kudu.apache.org/docs/installation.html#upgrade).

## [Prerequisites and Requirements](http://kudu.apache.org/docs/installation.html#_prerequisites_and_requirements)

**Hardware**

* One or more hosts to run Kudu masters. It is recommended to have either one master (no fault tolerance), three masters (can tolerate one failure), or five masters (two failures).
* One or more hosts to run Kudu tablet servers. When using replication, a minimum of three tablet servers is necessary.

**Operating System Requirements**

**Linux**

* RHEL 6, RHEL 7, CentOS 6, CentOS 7, Ubuntu 14.04 (Trusty), Ubuntu 16.04 (Xenial), Debian 8 (Jessie), or SLES 12.
* A kernel and filesystem that support hole punching. Hole punching is the use of the fallocate(2) system call with the FALLOC\_FL\_PUNCH\_HOLE option set. See [troubleshooting hole punching](http://kudu.apache.org/docs/troubleshooting.html#req_hole_punching) for more information.
* ntp.
* xfs or ext4 formatted drives.

**macOS**

* OS X 10.10 Yosemite, OS X 10.11 El Capitan, or macOS Sierra.
* Prebuilt macOS packages are not provided.

**Windows**

* Microsoft Windows is unsupported.

**Storage**

* If solid state storage is available, storing Kudu WALs on such high-performance media may significantly improve latency when Kudu is configured for its highest durability levels.

**Management**

* If you use Cloudera Manager and CDH, Cloudera Manager 5.4.3 or newer is required. Cloudera Manager 5.4.7 and newer provide better monitoring and administration options.

## [Install Using Packages](http://kudu.apache.org/docs/installation.html#install_packages)

You can install Kudu using packages managed by the operating system.

| **Table 1. Kudu Package Locations** | | |
| --- | --- | --- |
| **OS** | **Repository** | **Individual Packages** |
| **RHEL or CentOS** | [RHEL 6 or CentOS 6](http://archive.cloudera.com/kudu/redhat/6/x86_64/kudu/cloudera-kudu.repo), [RHEL 7 or CentOS 7](http://archive.cloudera.com/kudu/redhat/7/x86_64/kudu/cloudera-kudu.repo) | [RHEL 6 or CentOS 6](http://archive.cloudera.com/kudu/redhat/6/x86_64/kudu/5/RPMS/x86_64/), [RHEL 7 or CentOS 7](http://archive.cloudera.com/kudu/redhat/7/x86_64/kudu/5/RPMS/x86_64/) |
| **SLES** | [SLES 12](http://archive.cloudera.com/kudu/sles/12/x86_64/kudu/cloudera-kudu.repo) | [SLES 12](http://archive.cloudera.com/kudu/sles/12/x86_64/kudu/5/RPMS/x86_64/) |
| **Ubuntu** | [Trusty](http://archive.cloudera.com/kudu/ubuntu/trusty/amd64/kudu/cloudera.list), [Xenial](http://archive.cloudera.com/kudu/ubuntu/xenial/amd64/kudu/cloudera.list) | [Trusty](http://archive.cloudera.com/kudu/ubuntu/trusty/amd64/kudu/pool/contrib/k/kudu/), [Xenial](http://archive.cloudera.com/kudu/ubuntu/xenial/amd64/kudu/pool/contrib/k/kudu/) |
| **Debian** | [Jessie](http://archive.cloudera.com/kudu/debian/jessie/amd64/kudu/cloudera.list) | [Jessie](http://archive.cloudera.com/kudu/debian/jessie/amd64/kudu/pool/contrib/k/kudu/) |

### [Install on RHEL or CentOS Hosts](http://kudu.apache.org/docs/installation.html#_install_on_rhel_or_centos_hosts)

1. Download and configure the Kudu repositories for your operating system, or manually download individual RPMs, using the appropriate link from [Kudu Package Locations](http://kudu.apache.org/docs/installation.html#kudu_package_locations).
2. If using a Yum repository, use the following commands to install Kudu packages on each host, after saving the cloudera-kudu.repo file to /etc/yum.repos.d/.
3. sudo yum install kudu # Base Kudu files
4. sudo yum install kudu-master # Kudu master init.d service script and default configuration
5. sudo yum install kudu-tserver # Kudu tablet server init.d service script and default configuration
6. sudo yum install kudu-client0 # Kudu C++ client shared library
7. sudo yum install kudu-client-devel # Kudu C++ client SDK
8. To manually install the Kudu RPMs, first download them, then use the command sudo rpm -ivh <RPM to install> to install them.
9. Note: the kudu-master and kudu-tserver packages are only necessary on hosts where there is a master or tserver respectively (and completely unnecessary if using Cloudera Manager). Each provides configuration files and an init.d script to manage the corresponding Kudu process. Once installed, the Kudu process is started and stopped automatically when the host starts up and shuts down.

### [Install on SLES Hosts](http://kudu.apache.org/docs/installation.html#_install_on_sles_hosts)

1. Download and configure the Kudu repositories for your operating system, or manually download individual RPMs, using the appropriate link from [Kudu Package Locations](http://kudu.apache.org/docs/installation.html#kudu_package_locations).
2. If using a Zypper repository, use the following commands to install Kudu packages on each host, after saving the cloudera-kudu.repo file to /etc/zypp/repos.d.
3. sudo zypper install kudu # Base Kudu files
4. sudo zypper install kudu-master # Kudu master init.d service script and default configuration
5. sudo zypper install kudu-tserver # Kudu tablet server init.d service script and default configuration
6. sudo zypper install kudu-client0 # Kudu C++ client shared library
7. sudo zypper install kudu-client-devel # Kudu C++ client SDK
8. To manually install the Kudu RPMs, first download them, then use the command sudo rpm -ivh <RPM to install> to install them.
9. Note: the kudu-master and kudu-tserver packages are only necessary on hosts where there is a master or tserver respectively (and completely unnecessary if using Cloudera Manager). Each provides configuration files and an init.d script to manage the corresponding Kudu process. Once installed, the Kudu process is started and stopped automatically when the host starts up and shuts down.

### [Install on Ubuntu or Debian Hosts](http://kudu.apache.org/docs/installation.html#_install_on_ubuntu_or_debian_hosts)

1. If using an Ubuntu or Debian repository, use the following commands to install Kudu packages on each host after saving the cloudera.list file to

 /etc/apt/sources.list.d/.

1. sudo apt-get install kudu # Base Kudu files
2. sudo apt-get install kudu-master # Service scripts for managing kudu-master
3. sudo apt-get install kudu-tserver # Service scripts for managing kudu-tserver
4. sudo apt-get install libkuduclient0 # Kudu C++ client shared library
5. sudo apt-get install libkuduclient-dev # Kudu C++ client SDK
6. To manually install the Kudu RPMs, first download them, then use the command sudo rpm -ivh <RPM to install> to install them.
7. Note: the kudu-master and kudu-tserver packages are only necessary on hosts where there is a master or tserver respectively (and completely unnecessary if using Cloudera Manager). Each provides configuration files and an init.d script to manage the corresponding Kudu process. Once installed, the Kudu process is started and stopped automatically when the host starts up and shuts down.

### [Verify the Installation](http://kudu.apache.org/docs/installation.html#_verify_the_installation)

1. Verify that services are running using one of the following methods:
   * Examine the output of the ps command on servers to verify one or both of kudu-master or kudu-tserver processes is running.
   * Access the Master or Tablet Server web UI by opening http://<\_host\_name\_>:8051/ for masters or http://<\_host\_name\_>:8050/ for tablet servers.
2. If Kudu isn’t running, have a look at the log files in '/var/log/kudu', and if there’s a file ending with '.FATAL' then it means Kudu wasn’t able to start.
   * If the error is 'Error during hole punch test', it might be a problem [with your OS](http://kudu.apache.org/docs/troubleshooting.html#req_hole_punching).
   * If the error is 'Couldn’t get the current time', it’s a [problem with ntp](http://kudu.apache.org/docs/troubleshooting.html#ntp).
   * If it’s something else that doesn’t seem obvious or if you’ve tried the above solutions without luck, you can ask for help on the [user mailing list](http://kudu.apache.org/community.html).

### [Required Configuration](http://kudu.apache.org/docs/installation.html#required_config_without_cm)

Additional configuration steps are required on each host before you can start Kudu services.

The packages create a kudu-conf entry in the operating system’s alternatives database, and they ship the built-in conf.distalternative. To adjust your configuration, you can either edit the files in /etc/kudu/conf/ directly, or create a new alternative using the operating system utilities, make sure it is the link pointed to by /etc/kudu/conf/, and create custom configuration files there. Some parts of the configuration are configured in  /etc/default/kudu-master and /etc/default/kudu-tserver files as well. You should include or duplicate these configuration options if you create custom configuration files.

Review the configuration, including the default WAL and data directory locations, and adjust them according to your requirements.

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 / SLES

$ sudo chkconfig kudu-tserver on # RHEL / CentOS / SLES

$ sudo update-rc.d kudu-master defaults # Debian / Ubuntu

$ sudo update-rc.d kudu-tserver defaults # Debian / Ubuntu

For additional configuration of Kudu services, see [Configuring Kudu](http://kudu.apache.org/docs/configuration.html).

## [Build From Source](http://kudu.apache.org/docs/installation.html#build_from_source)

If installing Kudu using parcels or packages does not provide the flexibility you need, you can build Kudu from source. You can build from source on any supported operating system.

|  |  |
| --- | --- |
|  | **Known Build Issues**   * It is not possible to build Kudu on Microsoft Windows. * A C+11 capable compiler (GCC 4.8) is required. |

### [RHEL or CentOS](http://kudu.apache.org/docs/installation.html#rhel_from_source)

RHEL or CentOS 6.6 or later is required to build Kudu from source. To build on a version older than 7.0, the Red Hat Developer Toolset must be installed (in order to have access to a C++11 capable compiler).

Install the prerequisite libraries, if they are not installed.

$ sudo yum install autoconf automake cyrus-sasl-devel cyrus-sasl-gssapi \

cyrus-sasl-plain gcc gcc-c++ gdb git krb5-server krb5-workstation libtool \

make openssl-devel patch pkgconfig redhat-lsb-core rsync unzip vim-common which

If building on RHEL or CentOS older than 7.0, install the Red Hat Developer Toolset.

$ DTLS\_RPM=rhscl-devtoolset-3-epel-6-x86\_64-1-2.noarch.rpm

$ DTLS\_RPM\_URL=https://www.softwarecollections.org/repos/rhscl/devtoolset-3/epel-6-x86\_64/noarch/${DTLS\_RPM}

$ wget ${DTLS\_RPM\_URL} -O ${DTLS\_RPM}

$ sudo yum install -y scl-utils ${DTLS\_RPM}

$ sudo yum install -y devtoolset-3-toolchain

Optional: Install some additional packages, including ruby, if you plan to build documentation.

$ sudo yum install doxygen gem graphviz ruby-devel zlib-devel

|  |  |
| --- | --- |
|  | If building on RHEL or CentOS older than 7.0, the gem package may need to be replaced with rubygems |

Clone the Git repository and change to the new kudu directory.

$ git clone https://github.com/apache/kudu

$ cd kudu

Build any missing third-party requirements using the build-if-necessary.sh script. Not using the devtoolset will result in Host compiler appears to require libatomic, but cannot find it.

$ build-support/enable\_devtoolset.sh thirdparty/build-if-necessary.sh

1. Build Kudu, using the utilities installed in the previous step. Choose a build directory for the intermediate output, which can be anywhere in your filesystem except for the kudu directory itself. Notice that the devtoolset must still be specified, else you’ll get cc1plus: error: unrecognized command line option "-std=c++11".

mkdir -p build/release

cd build/release

../../build-support/enable\_devtoolset.sh \

../../thirdparty/installed/common/bin/cmake \

-DCMAKE\_BUILD\_TYPE=release \

../..

make -j4

1. Optional: Install Kudu binaries, libraries, and headers. If you do not specify an installation directory through the DESTDIRenvironment variable, /usr/local/ is the default.

sudo make DESTDIR=/opt/kudu install

1. Optional: Build the documentation. NOTE: This command builds local documentation that is not appropriate for uploading to the Kudu website.

$ make docs

**Example 1. RHEL / CentOS Build Script**

This script provides an overview of the procedure to build Kudu on a newly-installed RHEL or CentOS host, and can be used as the basis for an automated deployment scenario. It skips the steps marked **Optional** above.

#!/bin/bash

sudo yum -y install autoconf automake cyrus-sasl-devel cyrus-sasl-gssapi \

cyrus-sasl-plain gcc gcc-c++ gdb git krb5-server krb5-workstation libtool \

make openssl-devel patch pkgconfig redhat-lsb-core rsync unzip vim-common which

DTLS\_RPM=rhscl-devtoolset-3-epel-6-x86\_64-1-2.noarch.rpm

DTLS\_RPM\_URL=https://www.softwarecollections.org/repos/rhscl/devtoolset-3/epel-6-x86\_64/noarch/${DTLS\_RPM}

wget ${DTLS\_RPM\_URL} -O ${DTLS\_RPM}

sudo yum install -y scl-utils ${DTLS\_RPM}

sudo yum install -y devtoolset-3-toolchain

git clone https://github.com/apache/kudu

cd kudu

build-support/enable\_devtoolset.sh thirdparty/build-if-necessary.sh

mkdir -p build/release

cd build/release

../../build-support/enable\_devtoolset.sh \

../../thirdparty/installed/common/bin/cmake \

-DCMAKE\_BUILD\_TYPE=release \

../..

make -j4

### [Ubuntu or Debian](http://kudu.apache.org/docs/installation.html#ubuntu_from_source)

1. Install the prerequisite libraries, if they are not installed.
2. $ sudo apt-get install autoconf automake curl g++ gcc gdb git \
3. krb5-admin-server krb5-kdc krb5-user libkrb5-dev libsasl2-dev libsasl2-modules \
4. libsasl2-modules-gssapi-mit libssl-dev libtool lsb-release make ntp openssl \

patch pkg-config python rsync unzip vim-common

1. Optional: Install some additional packages, including ruby, if you plan to build documentation.

$ sudo apt-get install doxygen gem graphviz ruby-dev xsltproc zlib1g-dev

1. Clone the Git repository and change to the new kudu directory.
2. $ git clone https://github.com/apache/kudu

$ cd kudu

1. Build any missing third-party requirements using the build-if-necessary.sh script.

$ thirdparty/build-if-necessary.sh

1. Build Kudu, using the utilities installed in the previous step. Choose a build directory for the intermediate output, which can be anywhere in your filesystem except for the kudu directory itself.
2. mkdir -p build/release
3. cd build/release
4. ../../thirdparty/installed/common/bin/cmake -DCMAKE\_BUILD\_TYPE=release ../..

make -j4

1. Optional: Install Kudu binaries, libraries, and headers. If you do not specify an installation directory through the DESTDIRenvironment variable, /usr/local/ is the default.

sudo make DESTDIR=/opt/kudu install

1. Optional: Build the documentation. NOTE: This command builds local documentation that is not appropriate for uploading to the Kudu website.

$ make docs

**Example 2. Ubuntu / Debian Build Script**

This script provides an overview of the procedure to build Kudu on Ubuntu, and can be used as the basis for an automated deployment scenario. It skips the steps marked **Optional** above.

#!/bin/bash

sudo apt-get -y install autoconf automake curl g++ gcc gdb git \

krb5-admin-server krb5-kdc krb5-user libkrb5-dev libsasl2-dev libsasl2-modules \

libsasl2-modules-gssapi-mit libssl-dev libtool lsb-release make ntp openssl \

patch pkg-config python rsync unzip vim-common

git clone https://github.com/apache/kudu

cd kudu

thirdparty/build-if-necessary.sh

mkdir -p build/release

cd build/release

../../thirdparty/installed/common/bin/cmake \

-DCMAKE\_BUILD\_TYPE=release \

../..

make -j4

### [SUSE Linux Enterprise Server](http://kudu.apache.org/docs/installation.html#sles_from_source)

1. Install the prerequisite libraries, if they are not installed.

$ sudo zypper install autoconf automake curl cyrus-sasl-devel \

cyrus-sasl-gssapi gcc gcc-c++ gdb git krb5-devel libtool lsb-release make ntp \

openssl-devel patch pkg-config python rsync unzip vim

1. Clone the Git repository and change to the new kudu directory.

$ git clone https://github.com/apache/kudu

$ cd kudu

1. Build any missing third-party requirements using the build-if-necessary.sh script.

$ thirdparty/build-if-necessary.sh

1. Build Kudu, using the utilities installed in the previous step. Choose a build directory for the intermediate output, which can be anywhere in your filesystem except for the kudu directory itself.

mkdir -p build/release

cd build/release

../../thirdparty/installed/common/bin/cmake \

-DCMAKE\_BUILD\_TYPE=release \

../..

make -j4

1. Optional: Install Kudu binaries, libraries, and headers. If you do not specify an installation directory through the DESTDIRenvironment variable, /usr/local/ is the default.

sudo make DESTDIR=/opt/kudu install

**Example 3. SLES Build Script**

This script provides an overview of the procedure to build Kudu on SLES, and can be used as the basis for an automated deployment scenario. It skips the steps marked **Optional** above.

#!/bin/bash

sudo zypper install -y autoconf automake curl cyrus-sasl-devel \

cyrus-sasl-gssapi gcc gcc-c++ gdb git krb5-devel libtool lsb-release make ntp \

openssl-devel patch pkg-config python rsync unzip vim

git clone https://github.com/apache/kudu

cd kudu

thirdparty/build-if-necessary.sh

mkdir -p build/release

cd build/release

../../thirdparty/installed/common/bin/cmake \

-DCMAKE\_BUILD\_TYPE=release \

../..

make -j4

### [macOS](http://kudu.apache.org/docs/installation.html#osx_from_source)

The [Xcode](https://developer.apple.com/xcode/) package is necessary for compiling Kudu. Some of the instructions below use [Homebrew](http://brew.sh/) to install dependencies, but manual dependency installation is possible.

|  |  |
| --- | --- |
|  | **macOS Known Issues**  Kudu support for macOS is experimental, and should only be used for development. See [macOS Limitations & Known Issues](https://issues.cloudera.org/browse/KUDU-1219) for more information. |

1. Install the prerequisite libraries, if they are not installed.
2. $ brew tap homebrew/dupes

$ brew install autoconf automake cmake git krb5 libtool openssl pkg-config pstree

1. Optional: Install some additional packages, including ruby, if you plan to build documentation.
2. $ brew install doxygen graphviz ruby

$ brew install gnu-sed --with-default-names #The macOS default sed handles the -i parameter differently

1. Clone the Git repository and change to the new kudu directory.
2. $ git clone https://github.com/apache/kudu

$ cd kudu

1. Build any missing third-party requirements using the build-if-necessary.sh script.

$ thirdparty/build-if-necessary.sh

* + If different versions of the dependencies are installed and used when calling thirdparty/build-if-necessary.sh, you may get stuck with output similar to the following:
  + ./configure: line 16299: error near unexpected token `newline'

./configure: line 16299: ` PKG\_CHECK\_MODULES('

The thirdparty builds may be cached and may reflect the incorrect versions of the dependencies. Ensure that you have the correct dependencies listed in Step 1, clean the workspace, and then try to re-build.

$ git clean -fdx

$ thirdparty/build-if-necessary.sh

* + Some combinations of Homebrew installations and system upgrades can result with a different kind of error:
  + libtool: Version mismatch error.  This is libtool 2.4.6, but the
  + libtool: definition of this LT\_INIT comes from libtool 2.4.2.
  + libtool: You should recreate aclocal.m4 with macros from libtool 2.4.6

libtool: and run autoconf again.

As described in this [thread](https://github.com/Homebrew/legacy-homebrew/issues/43874), a possible fix is to uninstall and reinstall libtool:

$ brew uninstall libtool && brew install libtool

1. Build Kudu. Choose a build directory for the intermediate output, which can be anywhere in your filesystem except for the kududirectory itself.
2. mkdir -p build/release
3. cd build/release
4. ../../thirdparty/installed/common/bin/cmake \
5. -DCMAKE\_BUILD\_TYPE=release \
6. -DOPENSSL\_ROOT\_DIR=/usr/local/opt/openssl \
7. ../..

make -j4

**Example 4. macOS Build Script**

This script provides an overview of the procedure to build Kudu on macOS, and can be used as the basis for an automated deployment scenario. It assumes Xcode and Homebrew are installed.

#!/bin/bash

brew tap homebrew/dupes

brew install autoconf automake cmake git krb5 libtool openssl pkg-config pstree

git clone https://github.com/apache/kudu

cd kudu

thirdparty/build-if-necessary.sh

mkdir -p build/release

cd build/release

../../thirdparty/installed/common/bin/cmake \

-DCMAKE\_BUILD\_TYPE=release \

-DOPENSSL\_ROOT\_DIR=/usr/local/opt/openssl \

../..

make -j4

## [Installing the C++ Client Libraries](http://kudu.apache.org/docs/installation.html#build_cpp_client)

If you need access to the Kudu client libraries for development, install the kudu-client and kudu-client-devel package for your platform. See [Install Using Packages](http://kudu.apache.org/docs/installation.html#install_packages).

|  |  |
| --- | --- |
|  | Only build against the client libraries and headers (kudu\_client.so and client.h). Other libraries and headers are internal to Kudu and have no stability guarantees. |

## [Build the Java Client](http://kudu.apache.org/docs/installation.html#build_java_client)

**Requirements**

* JDK 7
* Apache Maven 3.x

To build the Java client, clone the Kudu Git repository, change to the java directory, and issue the following command:

$ mvn install -DskipTests

For more information about building the Java API, as well as Eclipse integration, see java/README.md.

## [View API Documentation](http://kudu.apache.org/docs/installation.html#view_api)

**C++ API Documentation**

You can view the [C++ client API documentation](http://kudu.apache.org/cpp-client-api/index.html) online. Alternatively, after [building Kudu from source](http://kudu.apache.org/docs/installation.html#build_from_source), you can additionally build the doxygen target (e.g., run make doxygen if using make) and use the locally generated API documentation by openingdocs/doxygen/client\_api/html/index.html file in your favorite Web browser.

|  |  |
| --- | --- |
|  | In order to build the doxygen target, it’s necessary to have doxygen with Dot (graphviz) support installed at your build machine. If you installed doxygen after building Kudu from source, you will need to run cmake again to pick up the doxygen location and generate appropriate targets. |

**Java API Documentation**

You can view the [Java API documentation](http://kudu.apache.org/apidocs/index.html) online. Alternatively, after [building the Java client](http://kudu.apache.org/docs/installation.html#build_java_client), Java API documentation is available in java/kudu-client/target/apidocs/index.html.

## [Upgrade from a Previous Version of Kudu](http://kudu.apache.org/docs/installation.html#upgrade)

Before upgrading, you should read the [Release Notes](http://kudu.apache.org/docs/release_notes.html) for the version of Kudu that you are about to install. Pay close attention to the incompatibilities, upgrade, and downgrade notes that are documented there.

|  |  |
| --- | --- |
|  | Currently rolling upgrades are not supported. Please shut down all Kudu services before upgrading the software. |

### [Upgrade Procedure](http://kudu.apache.org/docs/installation.html#upgrade_procedure)

1. Stop the Kudu master and tablet server services:
2. $ sudo service kudu-master stop

$ sudo service kudu-tserver stop

1. Upgrade the packages.
   * On RHEL or CentOS hosts:
   * sudo yum clean all

sudo yum upgrade kudu

* + On SLES hosts:
  + sudo zypper clean --all

sudo zypper update kudu

* + On Ubuntu or Debian hosts:
  + sudo apt-get update

sudo apt-get install kudu

1. Start the Kudu master and tablet server services:
2. $ sudo service kudu-master start

$ sudo service kudu-tserver start

# Configuring Apache Kudu

|  |  |
| --- | --- |
|  | Kudu is easier to configure with [Cloudera Manager](http://www.cloudera.com/content/www/en-us/products/cloudera-manager.html) than in a standalone installation. See Cloudera’s [Kudu documentation](http://www.cloudera.com/documentation/kudu/latest/topics/kudu_installation.html)for more details about using Kudu with Cloudera Manager. |

## [Configure Kudu](http://kudu.apache.org/docs/configuration.html#_configure_kudu)

### [Configuration Basics](http://kudu.apache.org/docs/configuration.html#_configuration_basics)

To configure the behavior of each Kudu process, you can pass command-line flags when you start it, or read those options from configuration files by passing them using one or more --flagfile=<file> options. You can even include the --flagfile option within your configuration file to include other files. Learn more about gflags by reading [its documentation](https://gflags.github.io/gflags/).

You can place options for masters and tablet servers into the same configuration file, and each will ignore options that do not apply.

Flags can be prefixed with either one or two - characters. This documentation standardizes on two: --example\_flag.

### [Discovering Configuration Options](http://kudu.apache.org/docs/configuration.html#_discovering_configuration_options)

Only the most common configuration options are documented here. For a more exhaustive list of configuration options, see the [Configuration Reference](http://kudu.apache.org/docs/configuration_reference.html).

To see all configuration flags for a given executable, run it with the --help option. Take care when configuring undocumented flags, as not every possible configuration has been tested, and undocumented options are not guaranteed to be maintained in future releases.

### [Configuring the Kudu Master](http://kudu.apache.org/docs/configuration.html#_configuring_the_kudu_master)

To see all available configuration options for the kudu-master executable, run it with the --help option:

$ kudu-master --help

| **Table 1. Supported Configuration Flags for Kudu Masters** | | | |
| --- | --- | --- | --- |
| **Flag** | **Valid Options** | **Default** | **Description** |
| --master\_addresses | string | localhost | Comma-separated list of all the RPC addresses for Master consensus-configuration. If not specified, assumes a standalone Master. |
| --fs\_data\_dirs | string |  | Comma-separated list of directories where the Master will place its data blocks. |
| --fs\_wal\_dir | string |  | The directory where the Master will place its write-ahead logs. May be the same as one of the directories listed in --fs\_data\_dirs, but not a sub-directory of a data directory. |
| --log\_dir | string | /tmp | The directory to store Master log files. |

For the full list of flags for masters, see the [Kudu Master Configuration Reference](http://kudu.apache.org/docs/configuration_reference.html#master_configuration_reference).

### [Configuring Tablet Servers](http://kudu.apache.org/docs/configuration.html#_configuring_tablet_servers)

To see all available configuration options for the kudu-tserver executable, run it with the --help option:

$ kudu-tserver --help

| **Table 2. Supported Configuration Flags for Kudu Tablet Servers** | | | |
| --- | --- | --- | --- |
| **Flag** | **Valid Options** | **Default** | **Description** |
| --fs\_data\_dirs | string |  | Comma-separated list of directories where the Tablet Server will place its data blocks. |
| --fs\_wal\_dir | string |  | The directory where the Tablet Server will place its write-ahead logs. May be the same as one of the directories listed in --fs\_data\_dirs, but not a sub-directory of a data directory. |
| --log\_dir | string | /tmp | The directory to store Tablet Server log files |
| --tserver\_master\_addrs | string | 127.0.0.1:7051 | Comma separated addresses of the masters which the tablet server should connect to. The masters do not read this flag. |
| --block\_cache\_capacity\_mb | integer | 512 | Maximum amount of memory allocated to the Kudu Tablet Server’s block cache. |
| --memory\_limit\_hard\_bytes | integer | 4294967296 | Maximum amount of memory a Tablet Server can consume before it starts rejecting all incoming writes. |

For the full list of flags for tablet servers, see the [Kudu Tablet Server Configuration Reference](http://kudu.apache.org/docs/configuration_reference.html#tablet_server_configuration_reference).

# Using Apache Kudu with Apache Impala (incubating)

Kudu has tight integration with Apache Impala (incubating), allowing you to use Impala to insert, query, update, and delete data from Kudu tablets using Impala’s SQL syntax, as an alternative to using the [Kudu APIs](http://kudu.apache.org/docs/installation.html#view_api) to build a custom Kudu application. In addition, you can use JDBC or ODBC to connect existing or new applications written in any language, framework, or business intelligence tool to your Kudu data, using Impala as the broker.

## [Requirements](http://kudu.apache.org/docs/kudu_impala_integration.html#_requirements)

* This documentation is specific to the certain versions of Impala. The syntax described will work only in the following releases:
  + The version of Impala 2.7.0 that ships with CDH 5.10. SELECT VERSION() will report impalad version 2.7.0-cdh5.10.0.
  + Apache Impala 2.8.0 releases compiled from source. SELECT VERSION() will report impalad version 2.8.0.

Older versions of Impala 2.7 (including the special IMPALA\_KUDU releases previously available) have incompatible syntax. Future versions are likely to be compatible with this syntax, but we recommend checking that this is the latest available documentation corresponding to the appropriate version you have installed.

* This documentation does not describe Impala installation procedures. Please refer to the Impala documentation and be sure that you are able to run simple queries against Impala tables on HDFS before proceeding.

## [Configuration](http://kudu.apache.org/docs/kudu_impala_integration.html#_configuration)

No configuration changes are required within Kudu to enable access from Impala.

Although not strictly necessary, it is recommended to configure Impala with the locations of the Kudu Master servers:

* Set the --kudu\_master\_hosts=<master1>[:port],<master2>[:port],<master3>[:port] flag in the Impala service configuration. If you are using Cloudera Manager, please refer to the appropriate Cloudera Manager documentation to do so.

If this flag is not set within the Impala service, it will be necessary to manually provide this configuration each time you create a table by specifying the kudu\_master\_addresses property inside a TBLPROPERTIES clause.

The rest of this guide assumes that the configuration has been set.

## [Using the Impala Shell](http://kudu.apache.org/docs/kudu_impala_integration.html#_using_the_impala_shell)

|  |  |
| --- | --- |
|  | This is only a small sub-set of Impala Shell functionality. For more details, see the [Impala Shell](http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/impala_impala_shell.html) documentation. |

* Start Impala Shell using the impala-shell command. By default, impala-shell attempts to connect to the Impala daemon on localhost on port 21000. To connect to a different host,, use the -i <host:port> option. To automatically connect to a specific Impala database, use the -d <database> option. For instance, if all your Kudu tables are in Impala in the database impala\_kudu, use -d impala\_kudu to use this database.
* To quit the Impala Shell, use the following command: quit;

### [Internal and External Impala Tables](http://kudu.apache.org/docs/kudu_impala_integration.html#_internal_and_external_impala_tables)

When creating a new Kudu table using Impala, you can create the table as an internal table or an external table.

**Internal**

An internal table is managed by Impala, and when you drop it from Impala, the data and the table truly are dropped. When you create a new table using Impala, it is generally a internal table.

**External**

An external table (created by CREATE EXTERNAL TABLE) is not managed by Impala, and dropping such a table does not drop the table from its source location (here, Kudu). Instead, it only removes the mapping between Impala and Kudu. This is the mode used in the syntax provided by Kudu for mapping an existing table to Impala.

See the [Impala documentation](http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/impala_tables.html) for more information about internal and external tables.

### [Querying an Existing Kudu Table In Impala](http://kudu.apache.org/docs/kudu_impala_integration.html#_querying_an_existing_kudu_table_in_impala)

Tables created through the Kudu API or other integrations such as Apache Spark are not automatically visible in Impala. To query them, you must first create an external table within Impala to map the Kudu table into an Impala database:

CREATE EXTERNAL TABLE my\_mapping\_table

STORED AS KUDU

TBLPROPERTIES (

'kudu.table\_name' = 'my\_kudu\_table'

);

### [Creating a New Kudu Table From Impala](http://kudu.apache.org/docs/kudu_impala_integration.html#kudu_impala_create_table)

Creating a new table in Kudu from Impala is similar to mapping an existing Kudu table to an Impala table, except that you need to specify the schema and partitioning information yourself.

Use the following example as a guideline. Impala first creates the table, then creates the mapping.

CREATE TABLE my\_first\_table

(

id BIGINT,

name STRING,

PRIMARY KEY(id)

)

PARTITION BY HASH PARTITIONS 16

STORED AS KUDU;

In the CREATE TABLE statement, the columns that comprise the primary key must be listed first. Additionally, primary key columns are implicitly marked NOT NULL.

When creating a new Kudu table, you are required to specify a distribution scheme. See [Partitioning Tables](http://kudu.apache.org/docs/kudu_impala_integration.html#partitioning_tables). The table creation example above is distributed into 16 partitions by hashing the id column, for simplicity. See [Partitioning Rules of Thumb](http://kudu.apache.org/docs/kudu_impala_integration.html#partitioning_rules_of_thumb) for guidelines on partitioning.

#### [CREATE TABLE AS SELECT](http://kudu.apache.org/docs/kudu_impala_integration.html#__code_create_table_as_select_code)

You can create a table by querying any other table or tables in Impala, using a CREATE TABLE …​ AS SELECT statement. The following example imports all rows from an existing table old\_table into a Kudu table new\_table. The names and types of columns in new\_table will determined from the columns in the result set of the SELECT statement. Note that you must additionally specify the primary key and partitioning.

CREATE TABLE new\_table

PRIMARY KEY (ts, name)

PARTITION BY HASH(name) PARTITIONS 8

STORED AS KUDU

AS SELECT ts, name, value FROM old\_table;

#### [Specifying Tablet Partitioning](http://kudu.apache.org/docs/kudu_impala_integration.html#_specifying_tablet_partitioning)

Tables are divided into tablets which are each served by one or more tablet servers. Ideally, tablets should split a table’s data relatively equally. Kudu currently has no mechanism for automatically (or manually) splitting a pre-existing tablet. Until this feature has been implemented, **you must specify your partitioning when creating a table**. When designing your table schema, consider primary keys that will allow you to split your table into partitions which grow at similar rates. You can designate partitions using a PARTITION BY clause when creating a table using Impala:

|  |  |
| --- | --- |
|  | Impala keywords, such as group, are enclosed by back-tick characters when they are not used in their keyword sense. |

CREATE TABLE cust\_behavior (

\_id BIGINT PRIMARY KEY,

salary STRING,

edu\_level INT,

usergender STRING,

`group` STRING,

city STRING,

postcode STRING,

last\_purchase\_price FLOAT,

last\_purchase\_date BIGINT,

category STRING,

sku STRING,

rating INT,

fulfilled\_date BIGINT

)

PARTITION BY RANGE (\_id)

(

PARTITION VALUES < 1439560049342,

PARTITION 1439560049342 <= VALUES < 1439566253755,

PARTITION 1439566253755 <= VALUES < 1439572458168,

PARTITION 1439572458168 <= VALUES < 1439578662581,

PARTITION 1439578662581 <= VALUES < 1439584866994,

PARTITION 1439584866994 <= VALUES < 1439591071407,

PARTITION 1439591071407 <= VALUES

)

STORED AS KUDU;

If you have multiple primary key columns, you can specify partition bounds using tuple syntax: ('va',1), ('ab',2). The expression must be valid JSON.

#### [Impala Databases and Kudu](http://kudu.apache.org/docs/kudu_impala_integration.html#_impala_databases_and_kudu)

Every Impala table is contained within a namespace called a database. The default database is called default, and users may create and drop additional databases as desired.

When a managed Kudu table is created from within Impala, the corresponding Kudu table will be named my\_database::table\_name.

#### [Impala Keywords Not Supported for Kudu Tables](http://kudu.apache.org/docs/kudu_impala_integration.html#_impala_keywords_not_supported_for_kudu_tables)

The following Impala keywords are not supported when creating Kudu tables: - PARTITIONED - LOCATION - ROWFORMAT

### [Optimizing Performance for Evaluating SQL Predicates](http://kudu.apache.org/docs/kudu_impala_integration.html#_optimizing_performance_for_evaluating_sql_predicates)

If the WHERE clause of your query includes comparisons with the operators =, <=, '\<', '\>', >=, BETWEEN, or IN, Kudu evaluates the condition directly and only returns the relevant results. This provides optimum performance, because Kudu only returns the relevant results to Impala. For predicates !=, LIKE, or any other predicate type supported by Impala, Kudu does not evaluate the predicates directly, but returns all results to Impala and relies on Impala to evaluate the remaining predicates and filter the results accordingly. This may cause differences in performance, depending on the delta of the result set before and after evaluating the WHERE clause.

### [Partitioning Tables](http://kudu.apache.org/docs/kudu_impala_integration.html#partitioning_tables)

Tables are partitioned into tablets according to a partition schema on the primary key columns. Each tablet is served by at least one tablet server. Ideally, a table should be split into tablets that are distributed across a number of tablet servers to maximize parallel operations. The details of the partitioning schema you use will depend entirely on the type of data you store and how you access it. For a full discussion of schema design in Kudu, see [Schema Design](http://kudu.apache.org/docs/schema_design.html).

Kudu currently has no mechanism for splitting or merging tablets after the table has been created. You must provide a partition schema for your table when you create it. When designing your tables, consider using primary keys that will allow you to partition your table into tablets which grow at similar rates.

You can partition your table using Impala’s PARTITION BY keyword, which supports distribution by RANGE or HASH. The partition scheme can contain zero or more HASH definitions, followed by an optional RANGE definition. The RANGE definition can refer to one or more primary key columns. Examples of [basic](http://kudu.apache.org/docs/kudu_impala_integration.html#basic_partitioning) and [advanced](http://kudu.apache.org/docs/kudu_impala_integration.html#advanced_partitioning) partitioning are shown below.

#### [Basic Partitioning](http://kudu.apache.org/docs/kudu_impala_integration.html#basic_partitioning)

**PARTITION BY RANGE**

You can specify range partitions for one or more primary key columns. Range partitioning in Kudu allows splitting a table based based on specific values or ranges of values of the chosen partition keys. This allows you to balance parallelism in writes with scan efficiency.

Suppose you have a table that has columns state, name, and purchase\_count. The following example creates 50 tablets, one per US state.

|  |  |
| --- | --- |
|  | **Monotonically Increasing Values**  If you partition by range on a column whose values are monotonically increasing, the last tablet will grow much larger than the others. Additionally, all data being inserted will be written to a single tablet at a time, limiting the scalability of data ingest. In that case, consider distributing by HASH instead of, or in addition to, RANGE. |

CREATE TABLE customers (

state STRING,

name STRING,

purchase\_count int,

PRIMARY KEY (state, name)

)

PARTITION BY RANGE (state)

(

PARTITION VALUE = 'al',

PARTITION VALUE = 'ak',

PARTITION VALUE = 'ar',

-- ... etc ...

PARTITION VALUE = 'wv',

PARTITION VALUE = 'wy'

)

STORED AS KUDU;

**PARTITION BY HASH**

Instead of distributing by an explicit range, or in combination with range distribution, you can distribute into a specific number of 'buckets' by hash. You specify the primary key columns you want to partition by, and the number of buckets you want to use. Rows are distributed by hashing the specified key columns. Assuming that the values being hashed do not themselves exhibit significant skew, this will serve to distribute the data evenly across buckets.

You can specify multiple definitions, and you can specify definitions which use compound primary keys. However, one column cannot be mentioned in multiple hash definitions. Consider two columns, a and b: \*  HASH(a), HASH(b) \*  HASH(a,b) \* HASH(a), HASH(a,b)

|  |  |
| --- | --- |
|  | PARTITION BY HASH with no column specified is a shortcut to create the desired number of buckets by hashing all primary key columns. |

Hash partitioning is a reasonable approach if primary key values are evenly distributed in their domain and no data skew is apparent, such as timestamps or serial IDs.

The following example creates 16 tablets by hashing the id and sku columns. This spreads writes across all 16 tablets. In this example, a query for a range of sku values is likely to need to read all 16 tablets, so this may not be the optimum schema for this table. See [Advanced Partitioning](http://kudu.apache.org/docs/kudu_impala_integration.html#advanced_partitioning) for an extended example.

CREATE TABLE cust\_behavior (

id BIGINT,

sku STRING,

salary STRING,

edu\_level INT,

usergender STRING,

`group` STRING,

city STRING,

postcode STRING,

last\_purchase\_price FLOAT,

last\_purchase\_date BIGINT,

category STRING,

rating INT,

fulfilled\_date BIGINT,

PRIMARY KEY (id, sku)

)

PARTITION BY HASH PARTITIONS 16

STORED AS KUDU;

#### [Advanced Partitioning](http://kudu.apache.org/docs/kudu_impala_integration.html#advanced_partitioning)

You can combine HASH and RANGE partitioning to create more complex partition schemas. You can specify zero or more HASHdefinitions, followed by zero or one RANGE definitions. Each definition can encompass one or more columns. While enumerating every possible distribution schema is out of the scope of this document, a few examples illustrate some of the possibilities.

#### [PARTITION BY HASH and RANGE](http://kudu.apache.org/docs/kudu_impala_integration.html#__code_partition_by_hash_code_and_code_range_code)

Consider the [simple hashing](http://kudu.apache.org/docs/kudu_impala_integration.html#distribute_by_hash) example above, If you often query for a range of sku values, you can optimize the example by combining hash partitioning with range partitioning.

The following example still creates 16 tablets, by first hashing the id column into 4 buckets, and then applying range partitioning to split each bucket into four tablets, based upon the value of the sku string. Writes are spread across at least four tablets (and possibly up to 16). When you query for a contiguous range of sku values, you have a good chance of only needing to read from a quarter of the tablets to fulfill the query.

|  |  |
| --- | --- |
|  | By default, the entire primary key is hashed when you use PARTITION BY HASH. To hash on only part of the primary key, specify it by using syntax like PARTITION BY HASH (id, sku). |

CREATE TABLE cust\_behavior (

id BIGINT,

sku STRING,

salary STRING,

edu\_level INT,

usergender STRING,

`group` STRING,

city STRING,

postcode STRING,

last\_purchase\_price FLOAT,

last\_purchase\_date BIGINT,

category STRING,

rating INT,

fulfilled\_date BIGINT,

PRIMARY KEY (id, sku)

)

PARTITION BY HASH (id) PARTITIONS 4,

RANGE (sku)

(

PARTITION VALUES < 'g',

PARTITION 'g' <= VALUES < 'o',

PARTITION 'o' <= VALUES < 'u',

PARTITION 'u' <= VALUES

)

STORED AS KUDU;

**Multiple PARTITION BY HASH Definitions**

Again expanding the example above, suppose that the query pattern will be unpredictable, but you want to ensure that writes are spread across a large number of tablets You can achieve maximum distribution across the entire primary key by hashing on both primary key columns.

CREATE TABLE cust\_behavior (

id BIGINT,

sku STRING,

salary STRING,

edu\_level INT,

usergender STRING,

`group` STRING,

city STRING,

postcode STRING,

last\_purchase\_price FLOAT,

last\_purchase\_date BIGINT,

category STRING,

rating INT,

fulfilled\_date BIGINT,

PRIMARY KEY (id, sku)

)

PARTITION BY HASH (id) PARTITIONS 4,

HASH (sku) PARTITIONS 4

STORED AS KUDU;

The example creates 16 partitions. You could also use HASH (id, sku) PARTITIONS 16. However, a scan for sku values would almost always impact all 16 partitions, rather than possibly being limited to 4.

**Non-Covering Range Partitions**

Kudu 1.0 and higher supports the use of non-covering range partitions, which address scenarios like the following:

* Without non-covering range partitions, in the case of time-series data or other schemas which need to account for constantly-increasing primary keys, tablets serving old data will be relatively fixed in size, while tablets receiving new data will grow without bounds.
* In cases where you want to partition data based on its category, such as sales region or product type, without non-covering range partitions you must know all of the partitions ahead of time or manually recreate your table if partitions need to be added or removed, such as the introduction or elimination of a product type.

Non-covering range partitions have some caveats. Be sure to read the link:/docs/schema\_design.html [Schema Design guide].

This example creates a tablet per year (5 tablets total), for storing log data. The table only accepts data from 2012 to 2016. Keys outside of these ranges will be rejected.

CREATE TABLE sales\_by\_year (

year INT, sale\_id INT, amount INT,

PRIMARY KEY (sale\_id, year)

)

PARTITION BY RANGE (year) (

PARTITION VALUE = 2012,

PARTITION VALUE = 2013,

PARTITION VALUE = 2014,

PARTITION VALUE = 2015,

PARTITION VALUE = 2016

)

STORED AS KUDU;

When records start coming in for 2017, they will be rejected. At that point, the 2017 range should be added as follows:

ALTER TABLE sales\_by\_year ADD RANGE PARTITION VALUE = 2017;

In use cases where a rolling window of data retention is required, range partitions may also be dropped. For example, if data from 2012 should no longer be retained, it may be deleted in bulk:

ALTER TABLE sales\_by\_year DROP RANGE PARTITION VALUE = 2012;

Note that, just like dropping a table, this irrecoverably deletes all data stored in the dropped partition.

#### [Partitioning Rules of Thumb](http://kudu.apache.org/docs/kudu_impala_integration.html#partitioning_rules_of_thumb)

* For large tables, such as fact tables, aim for as many tablets as you have cores in the cluster.
* For small tables, such as dimension tables, aim for a large enough number of tablets that each tablet is at least 1 GB in size.

In general, be mindful the number of tablets limits the parallelism of reads, in the current implementation. Increasing the number of tablets significantly beyond the number of cores is likely to have diminishing returns.

### [Inserting Data Into Kudu Tables](http://kudu.apache.org/docs/kudu_impala_integration.html#_inserting_data_into_kudu_tables)

Impala allows you to use standard SQL syntax to insert data into Kudu.

#### [Inserting Single Values](http://kudu.apache.org/docs/kudu_impala_integration.html#_inserting_single_values)

This example inserts a single row.

INSERT INTO my\_first\_table VALUES (99, "sarah");

This example inserts three rows using a single statement.

INSERT INTO my\_first\_table VALUES (1, "john"), (2, "jane"), (3, "jim");

#### [Inserting In Bulk](http://kudu.apache.org/docs/kudu_impala_integration.html#kudu_impala_insert_bulk)

When inserting in bulk, there are at least three common choices. Each may have advantages and disadvantages, depending on your data and circumstances.

**Multiple single INSERT statements**

This approach has the advantage of being easy to understand and implement. This approach is likely to be inefficient because Impala has a high query start-up cost compared to Kudu’s insertion performance. This will lead to relatively high latency and poor throughput.

**Single INSERT statement with multiple VALUES**

If you include more than 1024 VALUES statements, Impala batches them into groups of 1024 (or the value of batch\_size) before sending the requests to Kudu. This approach may perform slightly better than multiple sequential INSERT statements by amortizing the query start-up penalties on the Impala side. To set the batch size for the current Impala Shell session, use the following syntax: set batch\_size=10000;

|  |  |
| --- | --- |
|  | Increasing the Impala batch size causes Impala to use more memory. You should verify the impact on your cluster and tune accordingly. |

**Batch Insert**

The approach that usually performs best, from the standpoint of both Impala and Kudu, is usually to import the data using a SELECT FROM statement in Impala.

1. If your data is not already in Impala, one strategy is to [import it from a text file](http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/impala_txtfile.html), such as a TSV or CSV file.
2. [Create the Kudu table](http://kudu.apache.org/docs/kudu_impala_integration.html#kudu_impala_create_table), being mindful that the columns designated as primary keys cannot have null values.
3. Insert values into the Kudu table by querying the table containing the original data, as in the following example:

INSERT INTO my\_kudu\_table

SELECT \* FROM legacy\_data\_import\_table;

**Ingest using the C++ or Java API**

In many cases, the appropriate ingest path is to use the C++ or Java API to insert directly into Kudu tables. Unlike other Impala tables, data inserted into Kudu tables via the API becomes available for query in Impala without the need for any INVALIDATE METADATA statements or other statements needed for other Impala storage types.

#### [INSERT and Primary Key Uniqueness Violations](http://kudu.apache.org/docs/kudu_impala_integration.html#insert_ignore)

In most relational databases, if you try to insert a row that has already been inserted, the insertion will fail because the primary key would be duplicated. See [Failures During INSERT, UPDATE, and DELETE Operations](http://kudu.apache.org/docs/kudu_impala_integration.html#impala_insertion_caveat). Impala, however, will not fail the query. Instead, it will generate a warning, but continue to execute the remainder of the insert statement.

If the inserted rows are meant to replace existing rows, UPSERT may be used instead of INSERT.

INSERT INTO my\_first\_table VALUES (99, "sarah");

UPSERT INTO my\_first\_table VALUES (99, "zoe");

-- the current value of the row is 'zoe'

### [Updating a Row](http://kudu.apache.org/docs/kudu_impala_integration.html#_updating_a_row)

UPDATE my\_first\_table SET name="bob" where id = 3;

|  |  |
| --- | --- |
|  | The UPDATE statement only works in Impala when the target table is in Kudu. |

#### [Updating In Bulk](http://kudu.apache.org/docs/kudu_impala_integration.html#_updating_in_bulk)

You can update in bulk using the same approaches outlined in [Inserting In Bulk](http://kudu.apache.org/docs/kudu_impala_integration.html#kudu_impala_insert_bulk).

UPDATE my\_first\_table SET name="bob" where age > 10;

### [Deleting a Row](http://kudu.apache.org/docs/kudu_impala_integration.html#_deleting_a_row)

DELETE FROM my\_first\_table WHERE id < 3;

You can also delete using more complex syntax. A comma in the FROM sub-clause is one way that Impala specifies a join query. For more information about Impala joins, see <http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/impala_joins.html>.

DELETE c FROM my\_second\_table c, stock\_symbols s WHERE c.name = s.symbol;

|  |  |
| --- | --- |
|  | The DELETE statement only works in Impala when the target table is in Kudu. |

#### [Deleting In Bulk](http://kudu.apache.org/docs/kudu_impala_integration.html#_deleting_in_bulk)

You can delete in bulk using the same approaches outlined in [Inserting In Bulk](http://kudu.apache.org/docs/kudu_impala_integration.html#kudu_impala_insert_bulk).

DELETE FROM my\_first\_table WHERE id < 3;

### [Failures During INSERT, UPDATE, and DELETE Operations](http://kudu.apache.org/docs/kudu_impala_integration.html#impala_insertion_caveat)

INSERT, UPDATE, and DELETE statements cannot be considered transactional as a whole. If one of these operations fails part of the way through, the keys may have already been created (in the case of INSERT) or the records may have already been modified or removed by another process (in the case of UPDATE or DELETE). You should design your application with this in mind.

### [Altering Table Properties](http://kudu.apache.org/docs/kudu_impala_integration.html#_altering_table_properties)

You can change Impala’s metadata relating to a given Kudu table by altering the table’s properties. These properties include the table name, the list of Kudu master addresses, and whether the table is managed by Impala (internal) or externally.

**Rename an Impala Mapping Table**

ALTER TABLE my\_table RENAME TO my\_new\_table;

|  |  |
| --- | --- |
|  | Renaming a table using the ALTER TABLE …​ RENAME statement only renames the Impala mapping table, regardless of whether the table is an internal or external table. This avoids disruption to other applications that may be accessing the underlying Kudu table. |

**Rename the underlying Kudu table for an internal table**

If a table is an internal table, the underlying Kudu table may be renamed by changing the kudu.table\_name property:

ALTER TABLE my\_internal\_table

SET TBLPROPERTIES('kudu.table\_name' = 'new\_name')

**Remapping an external table to a different Kudu table**

If another application has renamed a Kudu table under Impala, it is possible to re-map an external table to point to a different Kudu table name.

ALTER TABLE my\_external\_table\_

SET TBLPROPERTIES('kudu.table\_name' = 'some\_other\_kudu\_table')

**Change the Kudu Master Address**

ALTER TABLE my\_table

SET TBLPROPERTIES('kudu.master\_addresses' = 'kudu-new-master.example.com:7051');

**Change an Internally-Managed Table to External**

ALTER TABLE my\_table SET TBLPROPERTIES('EXTERNAL' = 'TRUE');

### [Dropping a Kudu Table Using Impala](http://kudu.apache.org/docs/kudu_impala_integration.html#_dropping_a_kudu_table_using_impala)

If the table was created as an internal table in Impala, using CREATE TABLE, the standard DROP TABLE syntax drops the underlying Kudu table and all its data. If the table was created as an external table, using CREATE EXTERNAL TABLE, the mapping between Impala and Kudu is dropped, but the Kudu table is left intact, with all its data.

DROP TABLE my\_first\_table;

# Apache Kudu Administration

|  |  |
| --- | --- |
|  | Kudu is easier to manage with [Cloudera Manager](http://www.cloudera.com/content/www/en-us/products/cloudera-manager.html) than in a standalone installation. See Cloudera’s [Kudu documentation](http://www.cloudera.com/documentation/kudu/latest/topics/kudu_installation.html)for more details about using Kudu with Cloudera Manager. |

## [Starting and Stopping Kudu Processes](http://kudu.apache.org/docs/administration.html#_starting_and_stopping_kudu_processes)

|  |  |
| --- | --- |
|  | These instructions are relevant only when Kudu is installed using operating system packages (e.g. rpm or deb). |

1. Start Kudu services using the following commands:

$ sudo service kudu-master start

$ sudo service kudu-tserver start

1. To stop Kudu services, use the following commands:

$ sudo service kudu-master stop

$ sudo service kudu-tserver stop

## [Kudu Web Interfaces](http://kudu.apache.org/docs/administration.html#_kudu_web_interfaces)

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

### [Kudu Master Web Interface](http://kudu.apache.org/docs/administration.html#_kudu_master_web_interface)

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

* 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](http://kudu.apache.org/docs/administration.html#_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](http://kudu.apache.org/docs/administration.html#_common_web_interface_pages)

Both Kudu masters and tablet servers expose a common set of information via their web interfaces:

* HTTP access to server logs.
* an /rpcz endpoint which lists currently running RPCs via JSON.
* pages giving an overview and detailed information on the memory usage of different components of the process.
* information on the current set of configuration flags.
* information on the currently running threads and their resource consumption.
* a JSON endpoint exposing metrics about the server.
* information on the deployed version number of the daemon.

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

## [Kudu Metrics](http://kudu.apache.org/docs/administration.html#_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](http://kudu.apache.org/docs/administration.html#_listing_available_metrics)

The full set of available metrics for a Kudu server can be dumped via 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.

### [Collecting metrics via HTTP](http://kudu.apache.org/docs/administration.html#_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 elided 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.

For example:

$ curl -s 'http://example-ts:8050/metrics?include\_schema=1&metrics=connections\_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

}

]

}

]

|  |  |
| --- | --- |
|  | All histograms and counters are measured since the server start time, and are not reset upon collection. |

### [Collecting metrics to a log](http://kudu.apache.org/docs/administration.html#_collecting_metrics_to_a_log)

Kudu may be configured to periodically dump all of its metrics to a local log file using the --metrics\_log\_interval\_ms flag. Set this flag to the interval at which metrics should be written to a log file.

The metrics log will be written to the same directory as the other Kudu log files, with the same naming format. After any metrics 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.

|  |  |
| --- | --- |
|  | Although metrics logging automatically rolls and compresses previous log files, it does not remove old ones. Since metrics logging can use significant amounts of disk space, consider setting up a system utility to monitor space in the log directory and archive or delete old segments. |

## [Common Kudu workflows](http://kudu.apache.org/docs/administration.html#_common_kudu_workflows)

### [Migrating to Multiple Kudu Masters](http://kudu.apache.org/docs/administration.html#migrate_to_multi_master)

For 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.

|  |  |
| --- | --- |
|  | The workflow is unsafe for adding new masters to an existing multi-master configuration. Do not use it for that purpose. |
|  | The workflow presupposes at least basic familiarity with Kudu configuration management. If using Cloudera Manager (CM), 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 migration](http://kudu.apache.org/docs/administration.html#_prepare_for_the_migration)

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 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 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:

$ kudu fs dump uuid --fs\_wal\_dir=<master\_data\_dir> 2>/dev/null

**master\_data\_dir**

existing master’s previously recorded data directory

**Example**

$ kudu fs dump uuid --fs\_wal\_dir=/var/lib/kudu/master 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).

|  |  |
| --- | --- |
|  | Without DNS aliases it is not possible to recover from permanent master failures, and as such it is highly recommended. |

1. 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 colocated 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 via system packages (in which case the kudu and kudu-masterpackages should be installed), or via 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](http://kudu.apache.org/docs/administration.html#_perform_the_migration)

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 command sequence:

$ kudu fs format --fs\_wal\_dir=<master\_data\_dir>

$ kudu fs dump uuid --fs\_wal\_dir=<master\_data\_dir> 2>/dev/null

**master\_data\_dir**

new master’s previously recorded data directory

**Example**

$ kudu fs format --fs\_wal\_dir=/var/lib/kudu/master

$ kudu fs dump uuid --fs\_wal\_dir=/var/lib/kudu/master 2>/dev/null

f5624e05f40649b79a757629a69d061e

1. If using CM, 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.
2. Rewrite the master’s Raft configuration with the following command, executed on the existing master machine:

$ kudu local\_replica cmeta rewrite\_raft\_config --fs\_wal\_dir=<master\_data\_dir> <tablet\_id> <all\_masters>

**master\_data\_dir**

existing master’s previously recorded data directory

**tablet\_id**

must be the string 00000000000000000000000000000000

**all\_masters**

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**

master’s previously recorded UUID

**hostname**

master’s previously recorded hostname or alias

**port**

master’s previously recorded RPC port number

**Example**

$ kudu local\_replica cmeta rewrite\_raft\_config --fs\_wal\_dir=/var/lib/kudu/master 00000000000000000000000000000000 4aab798a69e94fab8d77069edff28ce0:master-1:7051 f5624e05f40649b79a757629a69d061e:master-2:7051 988d8ac6530f426cbe180be5ba52033d:master-3:7051

1. 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**

master’s previously recorded hostname or alias

**port**

master’s previously recorded RPC port number

1. Start the existing master.
2. Copy the master data to each new master with the following command, executed on each new master machine:

$ kudu local\_replica copy\_from\_remote --fs\_wal\_dir=<master\_data\_dir> <tablet\_id> <existing\_master>

**master\_data\_dir**

new master’s previously recorded data directory

**tablet\_id**

must be the string 00000000000000000000000000000000

**existing\_master**

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

**hostname**

existing master’s previously recorded hostname or alias

**port**

existing master’s previously recorded RPC port number

**Example**

$ kudu local\_replica copy\_from\_remote --fs\_wal\_dir=/var/lib/kudu/master 00000000000000000000000000000000 master-1:7051

1. Start all of the new masters.

|  |  |
| --- | --- |
|  | Skip the next step if using CM. |

1. 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**

master’s previously recorded hostname or alias

**port**

master’s previously recorded RPC port number

1. Start all of the tablet servers.

Congratulations, the cluster has now been migrated to multiple masters! To verify that all masters are working properly, consider performing the following sanity checks:

* Using a browser, visit each master’s web UI. Look at the /masters page. All of the masters should 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. See [Checking Cluster Health with ksck](http://kudu.apache.org/docs/administration.html#ksck) for more details.

### [Recovering from a dead Kudu Master in a Multi-Master Deployment](http://kudu.apache.org/docs/administration.html#_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](https://issues.apache.org/jira/browse/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 brief one as masters generally restart quickly.

|  |  |
| --- | --- |
|  | Kudu does not yet support Raft configuration changes for masters. As such, it is only possible to replace a master if the deployment was created with DNS aliases. See the [multi-master migration workflow](http://kudu.apache.org/docs/administration.html#migrate_to_multi_master) for more details. |
|  | The workflow presupposes at least basic familiarity with Kudu configuration management. If using Cloudera Manager (CM), 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](http://kudu.apache.org/docs/administration.html#_prepare_for_the_recovery)

1. 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.
2. 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.
3. Choose an unused machine in the cluster where the new master will live. The master generates very little load so it can be colocated 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.
4. 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-masterpackages should be installed), or via some other means.
   * Choose and record the directory where the master’s data will live.
5. 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:

$ kudu fs dump uuid --fs\_wal\_dir=<master\_data\_dir> 2>/dev/null

**master\_data\_dir**

live master’s previously recorded data directory

**Example**

$ kudu fs dump uuid --fs\_wal\_dir=/var/lib/kudu/master 2>/dev/null

80a82c4b8a9f4c819bab744927ad765c

1. 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 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 UUIDs of every master in the cluster, using the following command:

$ kudu local\_replica cmeta print\_replica\_uuids --fs\_wal\_dir=<master\_data\_dir> <tablet\_id> 2>/dev/null

**master\_data\_dir**

reference master’s previously recorded data directory

**tablet\_id**

must be the string 00000000000000000000000000000000

**Example**

$ kudu local\_replica cmeta print\_replica\_uuids --fs\_wal\_dir=/var/lib/kudu/master 00000000000000000000000000000000 2>/dev/null

80a82c4b8a9f4c819bab744927ad765c 2a73eeee5d47413981d9a1c637cce170 1c3f3094256347528d02ec107466aef3

1. 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](http://kudu.apache.org/docs/administration.html#_perform_the_recovery)

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

$ kudu fs format --fs\_wal\_dir=<master\_data\_dir> --uuid=<uuid>

**master\_data\_dir**

replacement master’s previously recorded data directory

**uuid**

dead master’s previously recorded UUID

**Example**

$ kudu fs format --fs\_wal\_dir=/var/lib/kudu/master --uuid=80a82c4b8a9f4c819bab744927ad765c

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

$ kudu local\_replica copy\_from\_remote --fs\_wal\_dir=<master\_data\_dir> <tablet\_id> <reference\_master>

**master\_data\_dir**

replacement master’s previously recorded data directory

**tablet\_id**

must be the string 00000000000000000000000000000000

**reference\_master**

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

**hostname**

reference master’s previously recorded hostname or alias

**port**

reference master’s previously recorded RPC port number

**Example**

$ kudu local\_replica copy\_from\_remote --fs\_wal\_dir=/var/lib/kudu/master 00000000000000000000000000000000 master-2:7051

1. If using CM, 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.
   * Add the port number (separated by a colon) if using a non-default RPC port value.
2. Reconfigure the DNS alias for the dead master to point at the replacement master.
3. Start the replacement master.
4. Restart the existing live masters. This results in a brief availability outage, but it should last only as long as it takes for the masters to come back up.

Congratulations, the dead master has been replaced! To verify that all masters are working properly, consider performing the following sanity checks:

* Using a browser, visit each master’s web UI. Look at the /masters page. All of the masters should 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. See [Checking Cluster Health with ksck](http://kudu.apache.org/docs/administration.html#ksck) for more details.

### [Checking Cluster Health with ksck](http://kudu.apache.org/docs/administration.html#ksck)

The kudu CLI includes a tool named ksck which can be used for checking cluster health and data integrity. 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, and requires the full list of master addresses to be specified:

$ 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 a success message, and return a zero (success) exit status.

Connected to the Master

Fetched info from all 1 Tablet Servers

Table IntegrationTestBigLinkedList is HEALTHY (1 tablet(s) checked)

The metadata for 1 table(s) is HEALTHY

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:

Connected to the Master

WARNING: Unable to connect to Tablet Server 8a0b66a756014def82760a09946d1fce

(tserver-01.example.com:7050): Network error: could not send Ping RPC to server: Client connection negotiation failed: client connection to 192.168.0.2:7050: connect: Connection refused (error 61)

WARNING: Fetched info from 0 Tablet Servers, 1 weren't reachable

Tablet ce3c2d27010d4253949a989b9d9bf43c of table 'IntegrationTestBigLinkedList'

is unavailable: 1 replica(s) not RUNNING

8a0b66a756014def82760a09946d1fce (tserver-01.example.com:7050): TS unavailable [LEADER]

Table IntegrationTestBigLinkedList has 1 unavailable tablet(s)

WARNING: 1 out of 1 table(s) are not in a healthy state

==================

Errors:

==================

error fetching info from tablet servers: Network error: Not all Tablet Servers are reachable

table consistency check error: Corruption: 1 table(s) are bad

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 IntegrationTestBigLinkedList table can be done with the following command:

$ kudu cluster ksck --checksum\_scan --tables IntegrationTestBigLinkedList master-01.example.com,master-02.example.com,master-03.example.com

### [Recovering from Disk Failure](http://kudu.apache.org/docs/administration.html#disk_failure_recovery)

Kudu tablet servers are not resilient to disk failure. When a disk containing a data directory or the write-ahead log (WAL) dies, the entire tablet server must be rebuilt. Kudu will automatically re-replicate tablets on other servers after a tablet server fails, but manual intervention is needed in order to restore the failed tablet server to a running state.

The first step to restoring a tablet server after a disk failure is to replace the failed disk, or remove the failed disk from the data-directory and/or WAL configuration. Next, the contents of the data directories and WAL directory must be removed. For example, if the tablet server is configured with --fs\_wal\_dir=/data/0/kudu-tserver-wal and --fs\_data\_dirs=/data/1/kudu-tserver,/data/2/kudu-tserver, the following commands will remove the data directories and WAL directory contents:

$ rm -rf /data/0/kudu-tserver-wal/\* /data/1/kudu-tserver/\* /data/2/kudu-tserver/\*

After the WAL and data directories are emptied, the tablet server process can be started. When Kudu is installed using system packages, service is typically used:

$ sudo service kudu-tserver start

Once the tablet server is running again, new tablet replicas will be created on it as necessary.

# Apache Kudu Troubleshooting

## [Startup Errors](http://kudu.apache.org/docs/troubleshooting.html#_startup_errors)

### [Errors During Hole Punching Test](http://kudu.apache.org/docs/troubleshooting.html#req_hole_punching)

Kudu requires hole punching capabilities in order to be efficient. Hole punching support depends upon your operation system kernel version and local filesystem implementation.

* RHEL or CentOS 6.4 or later, patched to kernel version of 2.6.32-358 or later. Unpatched RHEL or CentOS 6.4 does not include a kernel with support for hole punching.
* Ubuntu 14.04 includes version 3.13 of the Linux kernel, which supports hole punching.
* Newer versions of the EXT4 or XFS file systems support hole punching, but EXT3 does not. Older versions of XFS that do not support hole punching return a EOPNOTSUPP (operation not supported) error. Older versions of either EXT4 or XFS that do not support hole punching cause Kudu to emit an error message such as the following and to fail to start:
* Error during hole punch test. The log block manager requires a
* filesystem with hole punching support such as ext4 or xfs. On el6,
* kernel version 2.6.32-358 or newer is required. To run without hole
* punching (at the cost of some efficiency and scalability), reconfigure
* Kudu with --block\_manager=file. Refer to the Kudu documentation for more details. Raw error message follows.

Without hole punching support, the log block manager is unsafe to use. It won’t ever delete blocks, and will consume ever more space on disk.

If you can’t use hole punching in your environment, you can still try Kudu. Enable the file block manager instead of the log block manager by adding the --block\_manager=file flag to the commands you use to start the master and tablet servers. The file block manager does not scale as well as the log block manager.

|  |  |
| --- | --- |
|  | The file block manager is known to scale and perform poorly, and should only be used for small-scale evaluation and development. |

### [NTP Clock Synchronization](http://kudu.apache.org/docs/troubleshooting.html#ntp)

For the master and tablet server daemons, the server’s clock must be synchronized using NTP. In addition, the **maximum clock error**(not to be mistaken with the estimated error)  be below a configurable threshold. The default value is 10 seconds, but it can be set with the flag --max\_clock\_sync\_error\_usec.

If NTP is not installed, or if the clock is reported as unsynchronized, Kudu will not start, and will emit a message such as:

F0924 20:24:36.336809 14550 hybrid\_clock.cc:191 Couldn't get the current time: Clock unsynchronized. Status: Service unavailable: Error reading clock. Clock considered unsynchronized.

If NTP is installed and synchronized, but the maximum clock error is too high, the user will see a message such as:

Sep 17, 8:13:09.873 PM FATAL hybrid\_clock.cc:196 Couldn't get the current time: Clock synchronized, but error: 11130000, is past the maximum allowable error: 10000000

or

Sep 17, 8:32:31.135 PM FATAL tablet\_server\_main.cc:38 Check failed: \_s.ok() Bad status: Service unavailable: Cannot initialize clock: Cannot initialize HybridClock. Clock synchronized but error was too high (11711000 us).

|  |  |
| --- | --- |
|  | If NTP is installed the user can monitor the synchronization status by running ntptime. The relevant value is what is reported for maximum error. |

To install NTP, use the appropriate command for your operating system:

| **OS** | **Command** |
| --- | --- |
| Debian/Ubuntu | sudo apt-get install ntp |
| RHEL/CentOS | sudo yum install ntp |

If NTP is installed but not running, start it using one of these commands:

| **OS** | **Command** |
| --- | --- |
| Debian/Ubuntu | sudo service ntp restart |
| RHEL/CentOS | sudo /etc/init.d/ntpd restart |

|  |  |
| --- | --- |
|  | NTP requires a network connection and may take a few minutes to synchronize the clock. In some cases a spotty network connection may make NTP report the clock as unsynchronized. A common, though temporary, workaround for this is to restart NTP with one of the commands above. |

If the clock is being reported as synchronized by NTP, but the maximum error is too high, the user can increase the threshold to a higher value by setting the above mentioned flag. For example to increase the possible maximum error to 20 seconds the flag should be set like: --max\_clock\_sync\_error\_usec=20000000

## [Reporting Kudu Crashes](http://kudu.apache.org/docs/troubleshooting.html#crash_reporting)

Kudu uses the [Google Breakpad](https://chromium.googlesource.com/breakpad/breakpad/) library to generate a minidump whenever Kudu experiences a crash. These minidumps are typically only a few MB in size and are generated even if core dump generation is disabled. At this time, generating minidumps is only possible in Kudu on Linux builds.

A minidump file contains important debugging information about the process that crashed, including shared libraries loaded and their versions, a list of threads running at the time of the crash, the state of the processor registers and a copy of the stack memory for each thread, and CPU and operating system version information.

It is also possible to force Kudu to create a minidump without killing the process by sending a USR1 signal to the kudu-tserver or kudu-master process. For example:

sudo pkill -USR1 kudu-tserver

By default, Kudu stores its minidumps in a subdirectory of its configured glog directory called minidumps. This location can be customized by setting the --minidump\_path flag. Kudu will retain only a certain number of minidumps before deleting the oldest ones, in an effort to avoid filling up the disk with minidump files. The maximum number of minidumps that will be retained can be controlled by setting the --max\_minidumps gflag.

Minidumps contain information specific to the binary that created them and so are not usable without access to the exact binary that crashed, or a very similar binary.

|  |  |
| --- | --- |
|  | A minidump can be emailed to a Kudu developer or attached to a JIRA in order to help a Kudu developer debug a crash. In order for it to be useful, the developer will need to know the exact version of Kudu and the operating system where the crash was observed. Note that while a minidump does not contain a heap memory dump, it does contain stack memory and therefore it is possible for application data to appear in a minidump. If confidential or personal information is stored on the cluster, do not share minidump files. |

## [Performance Troubleshooting](http://kudu.apache.org/docs/troubleshooting.html#_performance_troubleshooting)

### [Kudu Tracing](http://kudu.apache.org/docs/troubleshooting.html#kudu_tracing)

The kudu-master and kudu-tserver daemons include built-in tracing support based on the open source [Chromium Tracing](https://www.chromium.org/developers/how-tos/trace-event-profiling-tool)framework. You can use tracing to help diagnose latency issues or other problems on Kudu servers.

#### [Accessing the tracing interface](http://kudu.apache.org/docs/troubleshooting.html#_accessing_the_tracing_interface)

The tracing interface is accessed via a web browser as part of the embedded web server in each of the Kudu daemons.

| **Table 1. Tracing Interface URLs** | |
| --- | --- |
| **Daemon** | **URL** |
| Tablet Server | <http://tablet-server-1.example.com:8050/tracing.html> |
| Master | <http://master-1.example.com:8051/tracing.html> |

|  |  |
| --- | --- |
|  | The tracing interface is known to work in recent versions of Google Chrome. Other browsers may not work as expected. |

#### [Collecting a trace](http://kudu.apache.org/docs/troubleshooting.html#_collecting_a_trace)

After navigating to the tracing interface, click the **Record** button on the top left corner of the screen. When beginning to diagnose a problem, start by selecting all categories. Click **Record** to begin recording a trace.

During the trace collection, events are collected into an in-memory ring buffer. This ring buffer is fixed in size, so it will eventually fill up to 100%. However, new events are still being collected while older events are being removed. While recording the trace, trigger the behavior or workload you are interested in exploring.

After collecting for several seconds, click **Stop**. The collected trace will be downloaded and displayed. Use the **?** key to display help text about using the tracing interface to explore the trace.

#### [Saving a trace](http://kudu.apache.org/docs/troubleshooting.html#_saving_a_trace)

You can save collected traces as JSON files for later analysis by clicking **Save** after collecting the trace. To load and analyze a saved JSON file, click **Load** and choose the file.

### [RPC Timeout Traces](http://kudu.apache.org/docs/troubleshooting.html#_rpc_timeout_traces)

If client applications are experiencing RPC timeouts, the Kudu tablet server WARNING level logs should contain a log entry which includes an RPC-level trace. For example:

W0922 00:56:52.313848 10858 inbound\_call.cc:193] Call kudu.consensus.ConsensusService.UpdateConsensus

from 192.168.1.102:43499 (request call id 3555909) took 1464ms (client timeout 1000).

W0922 00:56:52.314888 10858 inbound\_call.cc:197] Trace:

0922 00:56:50.849505 (+ 0us) service\_pool.cc:97] Inserting onto call queue

0922 00:56:50.849527 (+ 22us) service\_pool.cc:158] Handling call

0922 00:56:50.849574 (+ 47us) raft\_consensus.cc:1008] Updating replica for 2 ops

0922 00:56:50.849628 (+ 54us) raft\_consensus.cc:1050] Early marking committed up to term: 8 index: 880241

0922 00:56:50.849968 (+ 340us) raft\_consensus.cc:1056] Triggering prepare for 2 ops

0922 00:56:50.850119 (+ 151us) log.cc:420] Serialized 1555 byte log entry

0922 00:56:50.850213 (+ 94us) raft\_consensus.cc:1131] Marking committed up to term: 8 index: 880241

0922 00:56:50.850218 (+ 5us) raft\_consensus.cc:1148] Updating last received op as term: 8 index: 880243

0922 00:56:50.850219 (+ 1us) raft\_consensus.cc:1195] Filling consensus response to leader.

0922 00:56:50.850221 (+ 2us) raft\_consensus.cc:1169] Waiting on the replicates to finish logging

0922 00:56:52.313763 (+1463542us) raft\_consensus.cc:1182] finished

0922 00:56:52.313764 (+ 1us) raft\_consensus.cc:1190] UpdateReplicas() finished

0922 00:56:52.313788 (+ 24us) inbound\_call.cc:114] Queueing success response

These traces can give an indication of which part of the request was slow. Please include them in bug reports related to RPC latency outliers.

### [Kernel Stack Watchdog Traces](http://kudu.apache.org/docs/troubleshooting.html#_kernel_stack_watchdog_traces)

Each Kudu server process has a background thread called the Stack Watchdog, which monitors the other threads in the server in case they have blocked for longer-than-expected periods of time. These traces can indicate operating system issues or bottlenecked storage.

When the watchdog thread identifies a case of thread blockage, it logs an entry in the WARNING log like the following:

W0921 23:51:54.306350 10912 kernel\_stack\_watchdog.cc:111] Thread 10937 stuck at /data/kudu/consensus/log.cc:505 for 537ms:

Kernel stack:

[<ffffffffa00b209d>] do\_get\_write\_access+0x29d/0x520 [jbd2]

[<ffffffffa00b2471>] jbd2\_journal\_get\_write\_access+0x31/0x50 [jbd2]

[<ffffffffa00fe6d8>] \_\_ext4\_journal\_get\_write\_access+0x38/0x80 [ext4]

[<ffffffffa00d9b23>] ext4\_reserve\_inode\_write+0x73/0xa0 [ext4]

[<ffffffffa00d9b9c>] ext4\_mark\_inode\_dirty+0x4c/0x1d0 [ext4]

[<ffffffffa00d9e90>] ext4\_dirty\_inode+0x40/0x60 [ext4]

[<ffffffff811ac48b>] \_\_mark\_inode\_dirty+0x3b/0x160

[<ffffffff8119c742>] file\_update\_time+0xf2/0x170

[<ffffffff8111c1e0>] \_\_generic\_file\_aio\_write+0x230/0x490

[<ffffffff8111c4c8>] generic\_file\_aio\_write+0x88/0x100

[<ffffffffa00d3fb1>] ext4\_file\_write+0x61/0x1e0 [ext4]

[<ffffffff81180f5b>] do\_sync\_readv\_writev+0xfb/0x140

[<ffffffff81181ee6>] do\_readv\_writev+0xd6/0x1f0

[<ffffffff81182046>] vfs\_writev+0x46/0x60

[<ffffffff81182102>] sys\_pwritev+0xa2/0xc0

[<ffffffff8100b072>] system\_call\_fastpath+0x16/0x1b

[<ffffffffffffffff>] 0xffffffffffffffff

User stack:

@ 0x3a1ace10c4 (unknown)

@ 0x1262103 (unknown)

@ 0x12622d4 (unknown)

@ 0x12603df (unknown)

@ 0x8e7bfb (unknown)

@ 0x8f478b (unknown)

@ 0x8f55db (unknown)

@ 0x12a7b6f (unknown)

@ 0x3a1b007851 (unknown)

@ 0x3a1ace894d (unknown)

@ (nil) (unknown)

These traces can be useful for diagnosing root-cause latency issues when they are caused by systems below Kudu, such as disk controllers or file systems.

## [Issues using Kudu](http://kudu.apache.org/docs/troubleshooting.html#_issues_using_kudu)

### [ClassNotFoundException: com.cloudera.kudu.hive.KuduStorageHandler](http://kudu.apache.org/docs/troubleshooting.html#hive_handler)

Users will encounter this exception when trying to use a Kudu table via Hive. This is not a case of a missing jar, but simply that Impala stores Kudu metadata in Hive in a format that’s unreadable to other tools, including Hive itself and Spark. There is no workaround for Hive users. Spark users need to create temporary tables.

# Developing Applications With Apache Kudu

Kudu provides C++, Java and Python client APIs, as well as reference examples to illustrate their use.

|  |  |
| --- | --- |
|  | Use of server-side or private interfaces is not supported, and interfaces which are not part of public APIs have no stability guarantees. |

## [Viewing the API Documentation](http://kudu.apache.org/docs/developing.html#_viewing_the_api_documentation)

**C++ API Documentation**

You can view the [C++ client API documentation](http://kudu.apache.org/cpp-client-api/index.html) online. Alternatively, after [building Kudu from source](http://kudu.apache.org/docs/developing.html#build_from_source), you can additionally build the doxygen target (e.g., run make doxygen if using make) and use the locally generated API documentation by openingdocs/doxygen/client\_api/html/index.html file in your favorite Web browser.

|  |  |
| --- | --- |
|  | In order to build the doxygen target, it’s necessary to have doxygen with Dot (graphviz) support installed at your build machine. If you installed doxygen after building Kudu from source, you will need to run cmake again to pick up the doxygen location and generate appropriate targets. |

**Java API Documentation**

You can view the [Java API documentation](http://kudu.apache.org/apidocs/index.html) online. Alternatively, after [building the Java client](http://kudu.apache.org/docs/developing.html#build_java_client), Java API documentation is available in java/kudu-client/target/apidocs/index.html.

## [Working Examples](http://kudu.apache.org/docs/developing.html#_working_examples)

Several example applications are provided in the [kudu-examples](https://github.com/cloudera/kudu-examples) Github repository. Each example includes a README that shows how to compile and run it. These examples illustrate correct usage of the Kudu APIs, as well as how to set up a virtual machine to run Kudu. The following list includes some of the examples that are available today. Check the repository itself in case this list goes out of date.

**java/java-example**

A simple Java application which connects to a Kudu instance, creates a table, writes data to it, then drops the table.

**java/collectl**

A small Java application which listens on a TCP socket for time series data corresponding to the Collectl wire protocol. The commonly-available collectl tool can be used to send example data to the server.

**java/insert-loadgen**

A Java application that generates random insert load.

**python/dstat-kudu**

An example program that shows how to use the Kudu Python API to load data into a new / existing Kudu table generated by an external program, dstat in this case.

**python/graphite-kudu**

An experimental plugin for using graphite-web with Kudu as a backend.

**demo-vm-setup**

Scripts to download and run a VirtualBox virtual machine with Kudu already installed. See [Quickstart](http://kudu.apache.org/docs/quickstart.html) for more information.

These examples should serve as helpful starting points for your own Kudu applications and integrations.

### [Maven Artifacts](http://kudu.apache.org/docs/developing.html#_maven_artifacts)

The following Maven <dependency> element is valid for the Apache Kudu public release (since 1.0.0):

<dependency>

<groupId>org.apache.kudu</groupId>

<artifactId>kudu-client</artifactId>

<version>1.1.0</version>

</dependency>

Convenience binary artifacts for the Java client and various Java integrations (e.g. Spark, Flume) are also now available via the [ASF Maven repository](http://repository.apache.org/) and [Maven Central repository](https://mvnrepository.com/artifact/org.apache.kudu).

## [Example Impala Commands With Kudu](http://kudu.apache.org/docs/developing.html#_example_impala_commands_with_kudu)

See [Using Impala With Kudu](http://kudu.apache.org/docs/kudu_impala_integration.html) for guidance on installing and using Impala with Kudu, including several impala-shell examples.

## [Kudu Integration with Spark](http://kudu.apache.org/docs/developing.html#_kudu_integration_with_spark)

Kudu integrates with Spark through the Data Source API as of version 1.0.0. Include the kudu-spark dependency using the --packages option:

Use the kudu-spark\_2.10 artifact if using Spark with Scala 2.10

spark-shell --packages org.apache.kudu:kudu-spark\_2.10:1.1.0

Use kudu-spark2\_2.11 artifact if using Spark 2 with Scala 2.11

spark-shell --packages org.apache.kudu:kudu-spark2\_2.11:1.1.0

then import kudu-spark and create a dataframe:

import org.apache.kudu.spark.kudu.\_

import org.apache.kudu.client.\_

import collection.JavaConverters.\_

// Read a table from Kudu

val df = sqlContext.read.options(Map("kudu.master" -> "kudu.master:7051","kudu.table" -> "kudu\_table")).kudu

// Query using the Spark API...

df.select("id").filter("id" >= 5).show()

// ...or register a temporary table and use SQL

df.registerTempTable("kudu\_table")

val filteredDF = sqlContext.sql("select id from kudu\_table where id >= 5").show()

// Use KuduContext to create, delete, or write to Kudu tables

val kuduContext = new KuduContext("kudu.master:7051", sqlContext.sparkContext)

// Create a new Kudu table from a dataframe schema

// NB: No rows from the dataframe are inserted into the table

kuduContext.createTable(

"test\_table", df.schema, Seq("key"),

new CreateTableOptions()

.setNumReplicas(1)

.addHashPartitions(List("key").asJava, 3))

// Insert data

kuduContext.insertRows(df, "test\_table")

// Delete data

kuduContext.deleteRows(filteredDF, "test\_table")

// Upsert data

kuduContext.upsertRows(df, "test\_table")

// Update data

val alteredDF = df.select("id", $"count" + 1)

kuduContext.updateRows(filteredRows, "test\_table"

// Data can also be inserted into the Kudu table using the data source, though the methods on KuduContext are preferred

// NB: The default is to upsert rows; to perform standard inserts instead, set operation = insert in the options map

// NB: Only mode Append is supported

df.write.options(Map("kudu.master"-> "kudu.master:7051", "kudu.table"-> "test\_table")).mode("append").kudu

// Check for the existence of a Kudu table

kuduContext.tableExists("another\_table")

// Delete a Kudu table

kuduContext.deleteTable("unwanted\_table")

### [Spark Integration Known Issues and Limitations](http://kudu.apache.org/docs/developing.html#_spark_integration_known_issues_and_limitations)

* Kudu tables with a name containing upper case or non-ascii characters must be assigned an alternate name when registered as a temporary table.
* Kudu tables with a column name containing upper case or non-ascii characters may not be used with SparkSQL. Columns may be renamed in Kudu to work around this issue.
* <> and OR predicates are not pushed to Kudu, and instead will be evaluated by the Spark task. Only LIKE predicates with a suffix wildcard are pushed to Kudu, meaning that LIKE "FOO%" is pushed down but LIKE "FOO%BAR" isn’t.
* Kudu does not support all types supported by Spark SQL, such as Date, Decimal and complex types.
* Kudu tables may only be registered as temporary tables in SparkSQL. Kudu tables may not be queried using HiveContext.

## [Kudu Python Client](http://kudu.apache.org/docs/developing.html#_kudu_python_client)

The Kudu Python client provides a Python friendly interface to the C++ client API. The sample below demonstrates the use of part of the Python client.

import kudu

from kudu.client import Partitioning

from datetime import datetime

# Connect to Kudu master server

client = kudu.connect(host='kudu.master', port=7051)

# Define a schema for a new table

builder = kudu.schema\_builder()

builder.add\_column('key').type(kudu.int64).nullable(False).primary\_key()

builder.add\_column('ts\_val', type\_=kudu.unixtime\_micros, nullable=False, compression='lz4')

schema = builder.build()

# Define partitioning schema

partitioning = Partitioning().add\_hash\_partitions(column\_names=['key'], num\_buckets=3)

# Create new table

client.create\_table('python-example', schema, partitioning)

# Open a table

table = client.table('python-example')

# Create a new session so that we can apply write operations

session = client.new\_session()

# Insert a row

op = table.new\_insert({'key': 1, 'ts\_val': datetime.utcnow()})

session.apply(op)

# Upsert a row

op = table.new\_upsert({'key': 2, 'ts\_val': "2016-01-01T00:00:00.000000"})

session.apply(op)

# Updating a row

op = table.new\_update({'key': 1, 'ts\_val': ("2017-01-01", "%Y-%m-%d")})

session.apply(op)

# Delete a row

op = table.new\_delete({'key': 2})

session.apply(op)

# Flush write operations, if failures occur, capture print them.

try:

session.flush()

except kudu.KuduBadStatus as e:

print(session.get\_pending\_errors())

# Create a scanner and add a predicate

scanner = table.scanner()

scanner.add\_predicate(table['ts\_val'] == datetime(2017, 1, 1))

# Open Scanner and read all tuples

# Note: This doesn't scale for large scans

result = scanner.open().read\_all\_tuples()

## [Integration with MapReduce, YARN, and Other Frameworks](http://kudu.apache.org/docs/developing.html#_integration_with_mapreduce_yarn_and_other_frameworks)

Kudu was designed to integrate with MapReduce, YARN, Spark, and other frameworks in the Hadoop ecosystem. See[RowCounter.java](https://github.com/apache/kudu/blob/master/java/kudu-client-tools/src/main/java/org/apache/kudu/mapreduce/tools/RowCounter.java) and [ImportCsv.java](https://github.com/apache/kudu/blob/master/java/kudu-client-tools/src/main/java/org/apache/kudu/mapreduce/tools/ImportCsv.java) for examples which you can model your own integrations on. Stay tuned for more examples using YARN and Spark in the future.

# Apache Kudu Schema Design

Kudu tables have a structured data model similar to tables in a traditional RDBMS. Schema design is critical for achieving the best performance and operational stability from Kudu. Every workload is unique, and there is no single schema design that is best for every table. This document outlines effective schema design philosophies for Kudu, paying particular attention to where they differ from approaches used for traditional RDBMS schemas.

At a high level, there are three concerns when creating Kudu tables: [column design](http://kudu.apache.org/docs/schema_design.html#column-design), [primary key design](http://kudu.apache.org/docs/schema_design.html#primary-key), and [partitioning design](http://kudu.apache.org/docs/schema_design.html#partitioning). Of these, only partitioning will be a new concept for those familiar with traditional non-distributed relational databases. The final sections discuss [altering the schema](http://kudu.apache.org/docs/schema_design.html#alter-schema) of an existing table, and [known limitations](http://kudu.apache.org/docs/schema_design.html#known-limitations) with regard to schema design.

## [The Perfect Schema](http://kudu.apache.org/docs/schema_design.html#_the_perfect_schema)

The perfect schema would accomplish the following:

* Data would be distributed in such a way that reads and writes are spread evenly across tablet servers. This is impacted by partitioning.
* Tablets would grow at an even, predictable rate and load across tablets would remain steady over time. This is most impacted by partitioning.
* Scans would read the minimum amount of data necessary to fulfill a query. This is impacted mostly by primary key design, but partitioning also plays a role via partition pruning.

The perfect schema depends on the characteristics of your data, what you need to do with it, and the topology of your cluster. Schema design is the single most important thing within your control to maximize the performance of your Kudu cluster.

## [Column Design](http://kudu.apache.org/docs/schema_design.html#column-design)

A Kudu Table consists of one or more columns, each with a defined type. Columns that are not part of the primary key may be nullable. Supported column types include:

* boolean
* 8-bit signed integer
* 16-bit signed integer
* 32-bit signed integer
* 64-bit signed integer
* unixtime\_micros (64-bit microseconds since the Unix epoch)
* single-precision (32-bit) IEEE-754 floating-point number
* double-precision (64-bit) IEEE-754 floating-point number
* UTF-8 encoded string (up to 64KB uncompressed)
* binary (up to 64KB uncompressed)

Kudu takes advantage of strongly-typed columns and a columnar on-disk storage format to provide efficient encoding and serialization. To make the most of these features, columns should be specified as the appropriate type, rather than simulating a 'schemaless' table using string or binary columns for data which may otherwise be structured. In addition to encoding, Kudu allows compression to be specified on a per-column basis.

### [Column Encoding](http://kudu.apache.org/docs/schema_design.html#encoding)

Each column in a Kudu table can be created with an encoding, based on the type of the column.

| **Table 1. Encoding Types** | | |
| --- | --- | --- |
| **Column Type** | **Encoding** | **Default** |
| int8, int16, int32 | plain, bitshuffle, run length | bitshuffle |
| int64, unixtime\_micros | plain, bitshuffle, run length | bitshuffle |
| float, double | plain, bitshuffle | bitshuffle |
| bool | plain, run length | run length |
| string, binary | plain, prefix, dictionary | dictionary |

**Plain Encoding**

Data is stored in its natural format. For example, int32 values are stored as fixed-size 32-bit little-endian integers.

**Bitshuffle Encoding**

A block of values is rearranged to store the most significant bit of every value, followed by the second most significant bit of every value, and so on. Finally, the result is LZ4 compressed. Bitshuffle encoding is a good choice for columns that have many repeated values, or values that change by small amounts when sorted by primary key. The [bitshuffle](https://github.com/kiyo-masui/bitshuffle) project has a good overview of performance and use cases.

**Run Length Encoding**

Runs (consecutive repeated values) are compressed in a column by storing only the value and the count. Run length encoding is effective for columns with many consecutive repeated values when sorted by primary key.

**Dictionary Encoding**

A dictionary of unique values is built, and each column value is encoded as its corresponding index in the dictionary. Dictionary encoding is effective for columns with low cardinality. If the column values of a given row set are unable to be compressed because the number of unique values is too high, Kudu will transparently fall back to plain encoding for that row set. This is evaluated during flush.

**Prefix Encoding**

Common prefixes are compressed in consecutive column values. Prefix encoding can be effective for values that share common prefixes, or the first column of the primary key, since rows are sorted by primary key within tablets.

### [Column Compression](http://kudu.apache.org/docs/schema_design.html#compression)

Kudu allows per-column compression using the LZ4, Snappy, or zlib compression codecs. By default, columns are stored uncompressed. Consider using compression if reducing storage space is more important than raw scan performance.

Every data set will compress differently, but in general LZ4 is the most performant codec, while zlib will compress to the smallest data sizes. Bitshuffle-encoded columns are automatically compressed using LZ4, so it is not recommended to apply additional compression on top of this encoding.

## [Primary Key Design](http://kudu.apache.org/docs/schema_design.html#primary-keys)

Every Kudu table must declare a primary key index comprised of one or more columns. Primary key columns must be non-nullable, and may not be a boolean or floating-point type. Once set during table creation, the set of columns in the primary key may not be altered. Like an RDBMS primary key, the Kudu primary key enforces a uniqueness constraint; attempting to insert a row with the same primary key values as an existing row will result in a duplicate key error.

Unlike an RDBMS, Kudu does not provide an auto-incrementing column feature, so the application must always provide the full primary key during insert. Row delete and update operations must also specify the full primary key of the row to be changed; Kudu does not natively support range deletes or updates. The primary key values of a column may not be updated after the row is inserted; however, the row may be deleted and re-inserted with the updated value.

### [Primary Key Index](http://kudu.apache.org/docs/schema_design.html#indexing)

As with many traditional relational databases, Kudu’s primary key is a clustered index. All rows within a tablet are kept in primary key sorted order. Kudu scans which specify equality or range constraints on the primary key will automatically skip rows which can not satisfy the predicate. This allows individual rows to be efficiently found by specifying equality constraints on the primary key columns.

|  |  |
| --- | --- |
|  | Primary key indexing optimizations apply to scans on individual tablets. See the [Partition Pruning](http://kudu.apache.org/docs/schema_design.html#partition-pruning) section for details on how scans can use predicates to skip entire tablets. |

## [Partitioning](http://kudu.apache.org/docs/schema_design.html#partitioning)

In order to provide scalability, Kudu tables are partitioned into units called tablets, and distributed across many tablet servers. A row always belongs to a single tablet. The method of assigning rows to tablets is determined by the partitioning of the table, which is set during table creation.

Choosing a partitioning strategy requires understanding the data model and the expected workload of a table. For write-heavy workloads, it is important to design the partitioning such that writes are spread across tablets in order to avoid overloading a single tablet. For workloads involving many short scans, where the overhead of contacting remote servers dominates, performance can be improved if all of the data for the scan is located in the same tablet. Understanding these fundamental trade-offs is central to designing an effective partition schema.

|  |  |
| --- | --- |
|  | **No Default Partitioning**  Kudu does not provide a default partitioning strategy when creating tables. It is recommended that new tables which are expected to have heavy read and write workloads have at least as many tablets as tablet servers. |

Kudu provides two types of partitioning: [range partitioning](http://kudu.apache.org/docs/schema_design.html#range-partitioning) and [hash partitioning](http://kudu.apache.org/docs/schema_design.html#hash-partitioning). Tables may also have [multilevel partitioning](http://kudu.apache.org/docs/schema_design.html#multilevel-partitioning), which combines range and hash partitioning, or multiple instances of hash partitioning.

### [Range Partitioning](http://kudu.apache.org/docs/schema_design.html#range-partitioning)

Range partitioning distributes rows using a totally-ordered range partition key. Each partition is assigned a contiguous segment of the range partition keyspace. The key must be comprised of a subset of the primary key columns. If the range partition columns match the primary key columns, then the range partition key of a row will equal its primary key. In range partitioned tables without hash partitioning, each range partition will correspond to exactly one tablet.

The initial set of range partitions is specified during table creation as a set of partition bounds and split rows. For each bound, a range partition will be created in the table. Each split will divide a range partition in two. If no partition bounds are specified, then the table will default to a single partition covering the entire key space (unbounded below and above). Range partitions must always be non-overlapping, and split rows must fall within a range partition.

|  |  |
| --- | --- |
|  | see the [Range Partitioning Example](http://kudu.apache.org/docs/schema_design.html#range-partitioning-example) for further discussion of range partitioning. |

#### [Range Partition Management](http://kudu.apache.org/docs/schema_design.html#range-partition-management)

Kudu allows range partitions to be dynamically added and removed from a table at runtime, without affecting the availability of other partitions. Removing a partition will delete the tablets belonging to the partition, as well as the data contained in them. Subsequent inserts into the dropped partition will fail. New partitions can be added, but they must not overlap with any existing range partitions. Kudu allows dropping and adding any number of range partitions in a single transactional alter table operation.

Dynamically adding and dropping range partitions is particularly useful for time series use cases. As time goes on, range partitions can be added to cover upcoming time ranges. For example, a table storing an event log could add a month-wide partition just before the start of each month in order to hold the upcoming events. Old range partitions can be dropped in order to efficiently remove historical data, as necessary.

### [Hash Partitioning](http://kudu.apache.org/docs/schema_design.html#hash-partitioning)

Hash partitioning distributes rows by hash value into one of many buckets. In single-level hash partitioned tables, each bucket will correspond to exactly one tablet. The number of buckets is set during table creation. Typically the primary key columns are used as the columns to hash, but as with range partitioning, any subset of the primary key columns can be used.

Hash partitioning is an effective strategy when ordered access to the table is not needed. Hash partitioning is effective for spreading writes randomly among tablets, which helps mitigate hot-spotting and uneven tablet sizes.

|  |  |
| --- | --- |
|  | see the [Hash Partitioning Example](http://kudu.apache.org/docs/schema_design.html#hash-partitioning-example) for further discussion of hash partitioning. |

### [Multilevel Partitioning](http://kudu.apache.org/docs/schema_design.html#multilevel-partitioning)

Kudu allows a table to combine multiple levels of partitioning on a single table. Zero or more hash partition levels can be combined with an optional range partition level. The only additional constraint on multilevel partitioning beyond the constraints of the individual partition types, is that multiple levels of hash partitions must not hash the same columns.

When used correctly, multilevel partitioning can retain the benefits of the individual partitioning types, while reducing the downsides of each. The total number of tablets in a multilevel partitioned table is the product of the number of partitions in each level.

|  |  |
| --- | --- |
|  | see the [Hash and Range Partitioning Example](http://kudu.apache.org/docs/schema_design.html#hash-range-partitioning-example) and the [Hash and Hash Partitioning Example](http://kudu.apache.org/docs/schema_design.html#hash-hash-partitioning-example) for further discussion of multilevel partitioning. |

### [Partition Pruning](http://kudu.apache.org/docs/schema_design.html#partition-pruning)

Kudu scans will automatically skip scanning entire partitions when it can be determined that the partition can be entirely filtered by the scan predicates. To prune hash partitions, the scan must include equality predicates on every hashed column. To prune range partitions, the scan must include equality or range predicates on the range partitioned columns. Scans on multilevel partitioned tables can take advantage of partition pruning on any of the levels independently.

### [Partitioning Examples](http://kudu.apache.org/docs/schema_design.html#partitioning-examples)

To illustrate the factors and trade-offs associated with designing a partitioning strategy for a table, we will walk through some different partitioning scenarios. Consider the following table schema for storing machine metrics data (using SQL syntax and date-formatted timestamps for clarity):

CREATE TABLE metrics (

host STRING NOT NULL,

metric STRING NOT NULL,

time INT64 NOT NULL,

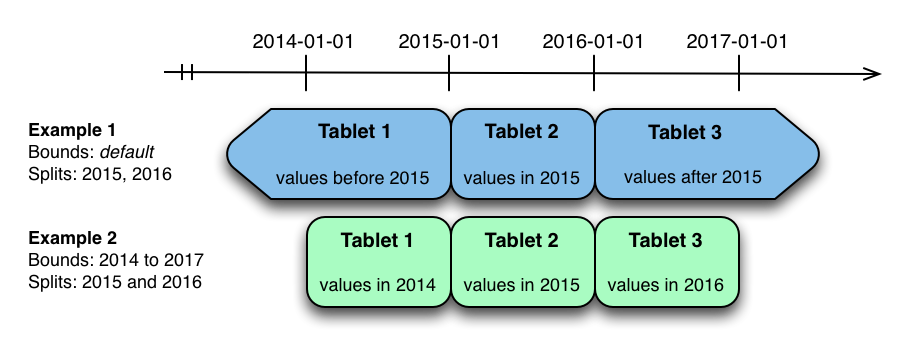
value DOUBLE NOT NULL,

PRIMARY KEY (host, metric, time),

);

#### [Range Partitioning Example](http://kudu.apache.org/docs/schema_design.html#range-partitioning-example)

A natural way to partition the metrics table is to range partition on the time column. Let’s assume that we want to have a partition per year, and the table will hold data for 2014, 2015, and 2016. There are at least two ways that the table could be partitioned: with unbounded range partitions, or with bounded range partitions.



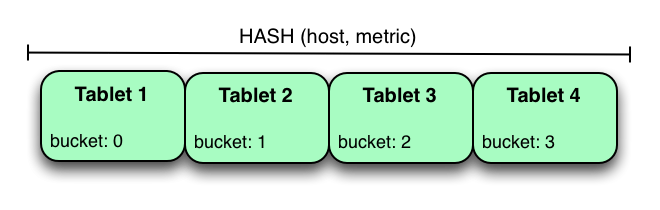
The image above shows the two ways the metrics table can be range partitioned on the time column. In the first example (in blue), the default range partition bounds are used, with splits at 2015-01-01 and 2016-01-01. This results in three tablets: the first containing values before 2015, the second containing values in the year 2015, and the third containing values after 2016. The second example (in green) uses a range partition bound of [(2014-01-01), (2017-01-01)], and splits at 2015-01-01 and 2016-01-01. The second example could have equivalently been expressed through range partition bounds of [(2014-01-01), (2015-01-01)], [(2015-01-01), (2016-01-01)], and [(2016-01-01), (2017-01-01)], with no splits. The first example has unbounded lower and upper range partitions, while the second example includes bounds.

Each of the range partition examples above allows time-bounded scans to prune partitions falling outside of the scan’s time bound. This can greatly improve performance when there are many partitions. When writing, both examples suffer from potential hot-spotting issues. Because metrics tend to always be written at the current time, most writes will go into a single range partition.

The second example is more flexible than the first, because it allows range partitions for future years to be added to the table. In the first example, all writes for times after 2016-01-01 will fall into the last partition, so the partition may eventually become too large for a single tablet server to handle.

#### [Hash Partitioning Example](http://kudu.apache.org/docs/schema_design.html#hash-partitioning-example)

Another way of partitioning the metrics table is to hash partition on the host and metric columns.



In the example above, the metrics table is hash partitioned on the  host and metric columns into four buckets. Unlike the range partitioning example earlier, this partitioning strategy will spread writes over all tablets in the table evenly, which helps overall write throughput. Scans over a specific host and metric can take advantage of partition pruning by specifying equality predicates, reducing the number of scanned tablets to one. One issue to be careful of with a pure hash partitioning strategy, is that tablets could grow indefinitely as more and more data is inserted into the table. Eventually tablets will become too big for an individual tablet server to hold.

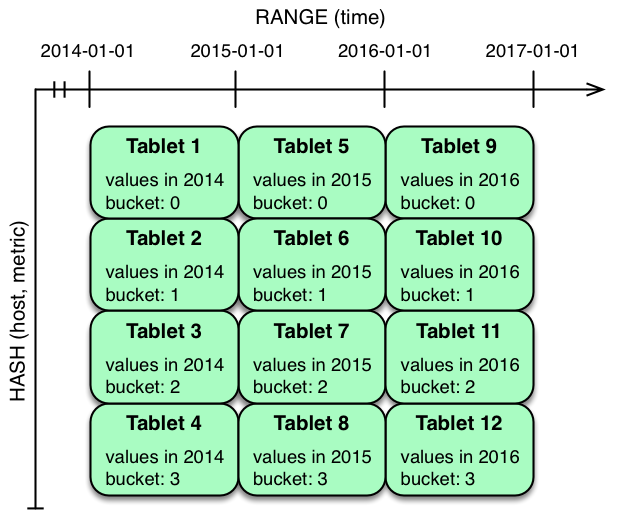
|  |  |
| --- | --- |
|  | Although these examples number the tablets, in reality tablets are only given UUID identifiers. There is no natural ordering among the tablets in a hash partitioned table. |

#### [Hash and Range Partitioning Example](http://kudu.apache.org/docs/schema_design.html#hash-range-partitioning-example)

The previous examples showed how the metrics table could be range partitioned on the time column, or hash partitioned on the host and metric columns. These strategies have associated strength and weaknesses:

| **Table 2. Partitioning Strategies** | | | |
| --- | --- | --- | --- |
| **Strategy** | **Writes** | **Reads** | **Tablet Growth** |
| range(time) | ✗ - all writes go to latest partition | ✓ - time-bounded scans can be pruned | ✓ - new tablets can be added for future time periods |
| hash(host, metric) | ✓ - writes are spread evenly among tablets | ✓ - scans on specific hosts and metrics can be pruned | ✗ - tablets could grow too large |

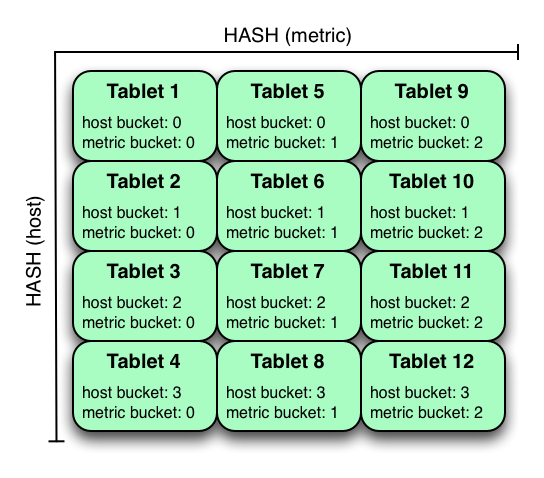
Hash partitioning is good at maximizing write throughput, while range partitioning avoids issues of unbounded tablet growth. Both strategies can take advantage of partition pruning to optimize scans in different scenarios. Using multilevel partitioning, it is possible to combine the two strategies in order to gain the benefits of both, while minimizing the drawbacks of each.



In the example above, range partitioning on the time column is combined with hash partitioning on the host and metric columns. This strategy can be thought of as having two dimensions of partitioning: one for the hash level and one for the range level. Writes into this table at the current time will be parallelized up to the number of hash buckets, in this case 4. Reads can take advantage of time bound **and** specific host and metric predicates to prune partitions. New range partitions can be added, which results in creating 4 additional tablets (as if a new column were added to the diagram).

#### [Hash and Hash Partitioning Example](http://kudu.apache.org/docs/schema_design.html#hash-hash-partitioning-example)

Kudu can support any number of hash partitioning levels in the same table, as long as the levels have no hashed columns in common.



In the example above, the table is hash partitioned on host into 4 buckets, and hash partitioned on metric into 3 buckets, resulting in 12 tablets. Although writes will tend to be spread among all tablets when using this strategy, it is slightly more prone to hot-spotting than when hash partitioning over multiple independent columns, since all values for an individual host or metric will always belong to a single tablet. Scans can take advantage of equality predicates on the host and metric columns separately to prune partitions.

Multiple levels of hash partitioning can also be combined with range partitioning, which logically adds another dimension of partitioning.

## [Schema Alterations](http://kudu.apache.org/docs/schema_design.html#alter-schema)

You can alter a table’s schema in the following ways:

* Rename the table
* Rename primary key columns
* Rename, add, or drop non-primary key columns
* Add and drop range partitions

Multiple alteration steps can be combined in a single transactional operation.

|  |  |
| --- | --- |
|  | **Renaming Primary Key Columns**  [KUDU-1626](https://issues.apache.org/jira/browse/KUDU-1626): Kudu does not yet support renaming primary key columns. |

## [Known Limitations](http://kudu.apache.org/docs/schema_design.html#known-limitations)

Kudu currently has some known limitations that may factor into schema design.

**Number of Columns**

By default, Kudu will not permit the creation of tables with more than 300 columns. We recommend schema designs that use fewer columns for best performance.

**Size of Cells**

No individual cell may be larger than 64KB before encoding or compression. The cells making up a composite key are limited to a total of 16KB after the internal composite-key encoding done by Kudu. Inserting rows not conforming to these limitations will result in errors being returned to the client.

**Size of Rows**

Although individual cells may be up to 64KB, and Kudu supports up to 300 columns, it is recommended that no single row be larger than a few hundred KB.

**Valid Identifiers**

Identifiers such as table and column names must be valid UTF-8 sequences and no longer than 256 bytes.

**Immutable Primary Keys**

Kudu does not allow you to update the primary key columns of a row.

**Non-alterable Primary Key**

Kudu does not allow you to alter the primary key columns after table creation.

**Non-alterable Partitioning**

Kudu does not allow you to change how a table is partitioned after creation, with the exception of adding or dropping range partitions.

**Non-alterable Column Types**

Kudu does not allow the type of a column to be altered.

**Partition Splitting**

Partitions cannot be split or merged after table creation.

# Apache Kudu Security

Kudu includes security features which allow Kudu clusters to be hardened against access from unauthorized users. This guide describes the security features provided by Kudu. [Configuring a Secure Kudu Cluster](http://kudu.apache.org/docs/security.html#configuration) lists essential configuration options when deploying a secure Kudu cluster. [Known Limitations](http://kudu.apache.org/docs/security.html#known-limitations) contains a list of known deficiencies in Kudu’s security capabilities.

## [Authentication](http://kudu.apache.org/docs/security.html#_authentication)

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 the connecting user or services for authorization checks. Authentication in Kudu is designed to interoperate with other secure Hadoop components by utilizing Kerberos.

Authentication can be configured on Kudu servers using the --rpc-authentication flag, which can be set to required, optional, or disabled. By default, the flag is set to optional. When required, Kudu will reject connections from clients and servers who lack authentication credentials. When optional, Kudu will attempt to use strong authentication. When disabled or strong authentication fails for 'optional', by default Kudu will only allow unauthenticated connections from trusted subnets, which are private networks (127.0.0.0/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16, 169.254.0.0/16) and local subnets of all local network interfaces. Unauthenticated connections from publicly routable IPs will be rejected.

The trusted subnets can be configured using the --trusted\_subnets flag, which can be set to IP blocks in CIDR notation separated by comma. Set it to '0.0.0.0/0' to allow unauthenticated connections from all remote IP addresses. However, if network access is not otherwise restricted by a firewall, malicious users may be able to gain unauthorized access. This can be mitigated if authentication is configured to be required.

|  |  |
| --- | --- |
|  | When the --rpc-authentication flag is set to optional, the cluster does not prevent access from unauthenticated users. To secure a cluster, use --rpc-authentication=required. |

### [Internal PKI](http://kudu.apache.org/docs/security.html#_internal_pki)

Kudu uses an internal PKI system to issue X.509 certificates to servers in the cluster. Connections between peers who have both obtained certificates will use TLS for authentication, which doesn’t require contacting the Kerberos KDC. These certificates are onlyused for internal communication among Kudu servers, and between Kudu clients and servers. The 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](http://kudu.apache.org/docs/security.html#_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 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 could not be used indefinitely. For the most part, authentication tokens should be completely transparent to users. By using authentication tokens, Kudu takes 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 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 secured Kudu cluster where only the planner node has Kerberos credentials.

## [Scalability](http://kudu.apache.org/docs/security.html#_scalability)

Kudu authentication is designed to scale to thousands of nodes, which requires avoiding unnecessary coordination with a central authentication authority (such as the Kerberos KDC). Instead, Kudu servers and clients will use Kerberos to establish initial trust with the Kudu master, and then use alternate credentials for subsequent connections. In particular, the master will issue internal X.509 certificates to servers, and temporary authentication tokens to clients.

## [Encryption](http://kudu.apache.org/docs/security.html#_encryption)

Kudu allows all communications among servers and between clients and servers to be encrypted with TLS.

Encryption can be configured on Kudu servers using the --rpc-encryption flag, which can be set to required, optional, or disabled. By default, the flag is set to optional. When required, Kudu will reject unencrypted connections. When optional, Kudu will attempt to use encryption. Same as authentication, when disabled or encryption fails for optional, Kudu will only allow unencrypted connections from trusted subnets and reject any unencrypted connections from publicly routable IPs. To secure a cluster, use --rpc-encryption=required.

|  |  |
| --- | --- |
|  | 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 like Spark and Impala to avoid encryption overhead, while still ensuring data confidentiality. |

## [Coarse-Grained Authorization](http://kudu.apache.org/docs/security.html#_coarse_grained_authorization)

Kudu supports coarse-grained authorization of client requests based on the authenticated client Kerberos principal (i.e. user or service). The two levels of access which can be configured are:

* **Superuser** - principals authorized as a superuser are able to perform certain administrative functionality such as using the kuducommand line tool to diagnose or 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.

|  |  |
| --- | --- |
|  | Internally, Kudu has a third access level for the daemons themselves. This ensures that users cannot connect to the cluster and pose as tablet servers. |

Access levels are granted using whitelist-style Access Control Lists (ACLs), one for each of the two levels. Each access control list either specifies a comma-separated list of users, or may be set to \* to indicate that all authenticated users are able to gain access at the specified level. See [Configuring a Secure Kudu Cluster](http://kudu.apache.org/docs/security.html#configuration) below for examples.

|  |  |
| --- | --- |
|  | The default value for the User ACL is \*, which allows all users access to the cluster. However, if authentication is enabled, this still restricts access to only those users who are able to successfully authenticate via Kerberos. Unauthenticated users on the same network as the Kudu servers will be unable to access the cluster. |

## [Web UI Encryption](http://kudu.apache.org/docs/security.html#web-ui)

The Kudu web UI can be configured to use secure HTTPS encryption by providing each server with TLS certificates. See [Configuring a Secure Kudu Cluster](http://kudu.apache.org/docs/security.html#configuration) for more information on web UI HTTPS configuration.

## [Web UI Redaction](http://kudu.apache.org/docs/security.html#_web_ui_redaction)

To prevent sensitive data from being exposed in the web UI, all row data is redacted. Table metadata, such as table names, column names, and partitioning information is not redacted. The web UI can be completely disabled by setting the --webserver-enabled=false flag on Kudu servers.

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

## [Log Security](http://kudu.apache.org/docs/security.html#logs)

To prevent sensitive data from being included in Kudu server logs, all row data is redacted by default. This feature can be turned off configuring the --redact flag.

## [Configuring a Secure Kudu Cluster](http://kudu.apache.org/docs/security.html#configuration)

The following configuration parameters should be set on all servers (master and tablet server) in order to ensure that a Kudu cluster is secure:

# 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

# 'nightly\_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,nightly\_etl\_service\_account

--superuser-acl=hadoopadmin

Further information about these flags can be found in the configuration flag reference.

## [Known Limitations](http://kudu.apache.org/docs/security.html#known-limitations)

Kudu has a few known security limitations:

**Long-lived Tokens**

The Java Kudu client does not automatically request fresh authn tokens after initial token expiration, so long-lived Java clients in secure clusters are not supported. However, the C Kudu client does automatically request fresh authn tokens, so long-lived C clients (i.e. beyond authn token lifetime) in secure clusters are supported.

**Custom Kerberos Principal**

Kudu does not support setting a custom service principal for Kudu processes. The principal must be 'kudu'.

**External PKI**

Kudu does not support externally-issued certificates for internal wire encryption (server to server and client to server).

**Fine-grained Authorization**

Kudu does not have the ability to restrict access based on operation type or target (table, column, etc). ACLs currently do not support authorization based on membership in a group.

**On-disk Encryption**

Kudu does not have built-in on-disk encryption. However, Kudu can be used with whole-disk encryption tools such as dm-crypt.

**Web UI Authentication**

The Kudu web UI lacks Kerberos-based authentication (SPNEGO), so access cannot be restricted based on Kerberos principals.

**Flume Integration**

Flume integration is not supported with secure Kudu clusters which require authentication or encryption.

# Transaction Semantics in Apache Kudu

This is a brief introduction to Kudu’s transaction and consistency semantics. For an in-depth technical exposition of most of what is mentioned here, and why it is correct, see the technical report [[1]](http://kudu.apache.org/docs/transaction_semantics.html#1).

Kudu’s transactional semantics and architecture are inspired by state-of-the-art systems such as Spanner [[2]](http://kudu.apache.org/docs/transaction_semantics.html#2) and Calvin [[3]](http://kudu.apache.org/docs/transaction_semantics.html#3). Kudu builds upon decades of database research. The core philosophy is to make the lives of developers easier by providing transactions with simple, strong semantics, without sacrificing performance or the ability to tune to different requirements.

Kudu is designed to eventually be fully ACID, however, multi-tablet transactions are not yet implemented. As such, this discussion focuses on single-tablet write operations, and only briefly touches multi-tablet reads. Eventually Kudu will support fully strict-serializable semantics. In fact it already does in a limited context, but not all corner cases are covered as this is still a work in progress.

Kudu currently allows the following operations:

* **Write operations** are sets of rows to be inserted, updated, or deleted in the storage engine, in a single tablet with multiple replicas. Write operations do not have separate "read sets" i.e. they do not scan existing data before performing the write. Each write is only concerned with previous state of the rows that are about to change. Writes are not "committed" explicitly by the user. Instead, they are committed automatically by the system, after completion.
* **Scans** are read operations that can traverse multiple tablets and read information with some consistency or correctness guarantees. Scans can perform time-travel reads, i.e. the user is able to set a scan timestamp in the past and get back results that reflect the state of the storage engine at that point in time.

|  |  |
| --- | --- |
|  | **Before We Begin**   * The term timestamp is mentioned several times to illustrate the functionality, but timestamp is an internal concept mostly invisible to users, except when setting timestamp on a KuduScanner. * We generally refer to methods and classes of the C++ client. While the Java client mostly has analogous methods and classes, the exact names of the APIs may differ. |

## [Single tablet write operations](http://kudu.apache.org/docs/transaction_semantics.html#_single_tablet_write_operations)

Kudu employs Multiversion Concurrency Control (MVCC) and the Raft consensus algorithm [[4]](http://kudu.apache.org/docs/transaction_semantics.html#4). Each write operation in Kudu must go through the tablet’s leader.

1. The leader acquires all locks for the rows that it will change.
2. The leader assigns the write a timestamp before the write is submitted for replication. This timestamp will be the write’s "tag" in MVCC.
3. After a majority of replicas acknowledges the change, the actual rows are changed.
4. After the changes are complete, they are made visible to concurrent writes and reads, atomically.

All replicas of a tablet observe the same order of operations, and if a write operation is assigned timestamp n and changes row x, a second write operation at timestamp m > n is guaranteed to see the new value of x.

This strict ordering of lock acquisition and timestamp assignment is enforced to be consistent across all replicas of a tablet through consensus. Therefore, write operations are totally ordered with regard to clock-assigned timestamps, relative to other writes in the same tablet. In other words, writes have strict-serializable semantics, though in an admittedly limited context. See this [blog post](http://www.bailis.org/blog/linearizability-versus-serializability) for a little more context regarding what these semantics mean.

While Isolated and Durable in an ACID sense, multi-row write operations are not yet fully Atomic. The failure of a single write in a batch operation does not roll back the operation, but produces per-row errors.

## [Writing to multiple tablets](http://kudu.apache.org/docs/transaction_semantics.html#_writing_to_multiple_tablets)

Kudu does not yet support transactions that span multiple tablets. However, consistent snapshot reads are possible (with caveats in the current implementation) as explained below.

Writes from a Kudu client are optionally buffered in memory until they are flushed and sent to the server. When client’s session flushes, the rows for each tablet are batched together, and sent to the tablet server which hosts the leader replica of the tablet. Since there are no inter-tablet transactions, each of these batches represents a single, independent write operation with its own timestamp. However, the client API provides the option to impose some constraints on the assigned timestamps and on how writes to different tablets can be observed by clients.

Kudu, like Spanner, was designed to be externally consistent [[5]](http://kudu.apache.org/docs/transaction_semantics.html#5), preserving consistency even when operations span multiple tablets and even multiple data centers. In practice this means that, if a write operation changes item x at tablet A, and a following write operation changes item y at tablet B, you might want to enforce that if the change to y is observed, the change to x must also be observed. There are many examples where this can be important. For example, if Kudu is storing clickstreams for further analysis, and two clicks follow each other but are stored in different tablets, subsequent clicks should be assigned subsequent timestamps so that the causal relationship between them is captured.

**CLIENT\_PROPAGATED Consistency**

Kudu’s default external consistency mode is called CLIENT\_PROPAGATED. See [[1]](http://kudu.apache.org/docs/transaction_semantics.html#1) for an extensive explanation on how it works. In brief, this mode causes writes from a single client to be automatically externally consistent. In the clickstream scenario above, if the two clicks are submitted by different client instances, the application must manually propagate timestamps from one client to the other for the causal relationship to be captured.

Timestamps between clients a and b can be propagated as follows:

**Java Client**

Call AsyncKuduClient#getLastPropagatedTimestamp() on client a, propagate the timestamp to client b, and callAsyncKuduClient#setLastPropagatedTimestamp() on client b.

**C++ Client**

Call KuduClient::GetLatestObservedTimestamp() on client a, propagate the timestamp to client b, and callKuduClient::SetLatestObservedTimestamp() on client b.

**COMMIT\_WAIT Consistency**

Kudu also has an experimental implementation of an external consistency model used in Google’s Spanner , called COMMIT\_WAIT. COMMIT\_WAIT works by tightly synchronizing the clocks on all machines in the cluster. Then, when a write occurs, timestamps are assigned and the results of the write are not made visible until enough time has passed so that no other machine in the cluster could possibly assign a lower timestamp to a following write.

When using this mode, the latency of writes is tightly tied to the accuracy of clocks on all the cluster hosts, and using this mode with loose clock synchronization causes writes to take a long time to complete or even time out. See [Known Issues and Limitations](http://kudu.apache.org/docs/transaction_semantics.html#known_issues).

The COMMIT\_WAIT consistency mode may be selected as follows:

**Java Client**

Call KuduSession#setExternalConsistencyMode(ExternalConsistencyMode.COMMIT\_WAIT)

**C++ Client**

Call KuduSession::SetExternalConsistencyMode(COMMIT\_WAIT)

|  |  |
| --- | --- |
|  | COMMIT\_WAIT consistency is considered an experimental feature. It may return incorrect results, exhibit performance issues, or negatively impact cluster stability. Use in production environments is discouraged. |

## [Read Operations (Scans)](http://kudu.apache.org/docs/transaction_semantics.html#_read_operations_scans)

Scans are read operations performed by clients that may span one or more rows across one or more tablets. When a server receives a scan request, it takes a snapshot of the MVCC state and then proceeds in one of two ways depending on the read mode selected by the user. The mode may be selected as follows:

**Java Client**

Call KuduScannerBuilder#setReadMode(…​)

**C++ Client**

Call KuduScanner::SetReadMode()

The following modes are available in both clients:

**READ\_LATEST**

This is the default read mode. The server takes a snapshot of the MVCC state and proceeds with the read immediately. Reads in this mode only yield 'Read Committed' isolation.

**READ\_AT\_SNAPSHOT**

In this read mode, scans are consistent and repeatable. A timestamp for the snapshot is selected either by the server, or set explicitly by the user through KuduScanner::SetSnapshotMicros(). Explicitly setting the timestamp is recommended; see [Recommendations](http://kudu.apache.org/docs/transaction_semantics.html#recommendations). The server waits until this timestamp is 'safe' (until all write operations that have a lower timestamp have completed and are visible). This delay, coupled with an external consistency method, will eventually allow Kudu to have full strict-serializable semantics for reads and writes. This is still a work in progress and some anomalies are still possible (see [Known Issues and Limitations](http://kudu.apache.org/docs/transaction_semantics.html#known_issues)). Only scans in this mode can be fault-tolerant.

Selecting between read modes requires balancing the trade-offs and making a choice that fits your workload. For instance, a reporting application that needs to scan the entire database might need to perform careful accounting operations, so that scan may need to be fault-tolerant, but probably doesn’t require a to-the-microsecond up-to-date view of the database. In that case, you might choose READ\_AT\_SNAPSHOT and select a timestamp that is a few seconds in the past when the scan starts. On the other hand, a machine learning workload that is not ingesting the whole data set and is already statistical in nature might not require the scan to be repeatable, so you might choose READ\_LATEST instead.

## [Known Issues and Limitations](http://kudu.apache.org/docs/transaction_semantics.html#known_issues)

There are several gaps and corner cases that prevent Kudu from being fully strictly-serializable in some situations, at the moment. Below are the details and next, some recommendations.

### [Reads (Scans)](http://kudu.apache.org/docs/transaction_semantics.html#known_issues_scans)

* Support for COMMIT\_WAIT is experimental and requires careful tuning of the time-synchronization protocol, such as NTP (Network Time Protocol). Its use is discouraged in production environments.

### [Writes](http://kudu.apache.org/docs/transaction_semantics.html#_writes)

* On a leader change, READ\_AT\_SNAPSHOT scans at a snapshot whose timestamp is beyond the last write may also yield non-repeatable reads (see [KUDU-1188](https://issues.apache.org/jira/browse/KUDU-1188)). See [Recommendations](http://kudu.apache.org/docs/transaction_semantics.html#recommendations) for a workaround.
* Impala scans are currently performed as READ\_LATEST and have no consistency guarantees.
* In AUTO\_BACKGROUND\_FLUSH mode, or when using "async" flushing mechanisms, writes applied to a single client session may become reordered due to the concurrency of flushing the data to the server. This may be particularly noticeable if a single row is quickly updated with different values in succession. This phenomenon affects all client API implementations. Workarounds are described in the API documentation for the respective implementations in the docs for FlushMode or AsyncKuduSession. See [KUDU-1767](https://issues.apache.org/jira/browse/KUDU-1767).

### [Recommendations](http://kudu.apache.org/docs/transaction_semantics.html#recommendations)

* If repeatable snapshot reads are a requirement, use READ\_AT\_SNAPSHOT with a timestamp that is slightly in the past (between 2-5 seconds, ideally). This will circumvent the anomaly described in [Reads (Scans)](http://kudu.apache.org/docs/transaction_semantics.html#known_issues_scans). Even when the anomaly has been addressed, back-dating the timestamp will always make scans faster, since they are unlikely to block.
* If external consistency is a requirement and you decide to use COMMIT\_WAIT, the time-synchronization protocol needs to be tuned carefully. Each transaction will wait 2x the maximum clock error at the time of execution, which is usually in the 100 msec. to 1 sec. range with the default settings, maybe more. Thus, transactions would take at least 200 msec. to 2 sec. to complete when using the default settings and may even time out.
  + A local server should be used as a time server. We’ve performed experiments using the default NTP time source available in a Google Compute Engine data center and were able to obtain a reasonable tight max error bound, usually varying between 12-17 milliseconds.
  + The following parameters should be adjusted in /etc/ntp.conf to tighten the maximum error:
    - server my\_server.org iburst minpoll 1 maxpoll 8
    - tinker dispersion 500
    - tinker allan 0

|  |  |
| --- | --- |
|  | The above parameters minimize maximum error at the expense of estimated error, the latter might be orders of magnitude above it’s "normal" value. These parameters also may place a greater load on the time server, since they make the servers poll much more frequently. |

## [References](http://kudu.apache.org/docs/transaction_semantics.html#_references)

* [1] David Alves, Todd Lipcon and Vijay Garg. Technical Report: HybridTime - Accessible Global Consistency with High Clock Uncertainty. April, 2014. <http://users.ece.utexas.edu/~garg/pdslab/david/hybrid-time-tech-report-01.pdf>
* [2] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2012. Spanner: Google’s globally-distributed database. In Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation (OSDI'12). USENIX Association, Berkeley, CA, USA, 251-264.
* [3] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. 2012. Calvin: fast distributed transactions for partitioned database systems. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12). ACM, New York, NY, USA, 1-12. DOI=10.1145/2213836.2213838 <http://doi.acm.org/10.1145/2213836.2213838>
* [4] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference (USENIX ATC'14), Garth Gibson and Nickolai Zeldovich (Eds.). USENIX Association, Berkeley, CA, USA, 305-320.
* [5] Kwei-Jay Lin, "Consistency issues in real-time database systems," in System Sciences, 1989. Vol.II: Software Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on , vol.2, no., pp.654-661 vol.2, 3-6 Jan 1989 doi: 10.1109/HICSS.1989.48069

# Apache Kudu Background Maintenance Tasks

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

## [Maintenance manager](http://kudu.apache.org/docs/background_tasks.html#_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 the improvement 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.

## [Flushing data to disk](http://kudu.apache.org/docs/background_tasks.html#_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 ops have the prefix flush\_mrs and flush\_dms, respectively.

## [Compacting on-disk data](http://kudu.apache.org/docs/background_tasks.html#_compacting_on_disk_data)

Kudu constantly performs several types of 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. There are two types of delta store compaction operations that may be run as well: MinorDeltaCompactionOp and MajorDeltaCompactionOp.

For more information on what these different types of compaction operations do, please see the [Kudu Tablet design document](https://github.com/apache/kudu/blob/master/docs/design-docs/tablet.md).

The metrics associated with these tasks have the prefix compact\_rs, delta\_minor\_compact\_rs, and delta\_major\_compact\_rs, respectively.

## [Write-ahead log GC](http://kudu.apache.org/docs/background_tasks.html#_write_ahead_log_gc)

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 configured size (controlled by --log\_segment\_size\_mb). 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 GC and the ancient history mark](http://kudu.apache.org/docs/background_tasks.html#_tablet_history_gc_and_the_ancient_history_mark)

Because Kudu uses a multiversion concurrency control (MVCC) mechanism to ensure that snapshot scans can proceeed isolated from new changes to a table, 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 point 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 --tablet\_history\_max\_age\_sec.

There are two background tasks that GC historical MVCC data older than the AHM: the one that runs the merging compaction, called CompactRowSetsOp (see above), and a separate background task that 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.

# Apache Kudu Command Line Tools Reference

## [Command Hierarchy](http://kudu.apache.org/docs/command_line_tools_reference.html#_command_hierarchy)

* [cluster](http://kudu.apache.org/docs/command_line_tools_reference.html#cluster)
  + [ksck](http://kudu.apache.org/docs/command_line_tools_reference.html#cluster-ksck)
* [fs](http://kudu.apache.org/docs/command_line_tools_reference.html#fs)
  + [check](http://kudu.apache.org/docs/command_line_tools_reference.html#fs-check)
  + [format](http://kudu.apache.org/docs/command_line_tools_reference.html#fs-format)
  + [dump](http://kudu.apache.org/docs/command_line_tools_reference.html#fs-dump)
    - [cfile](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-cfile)
    - [tree](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-tree)
    - [uuid](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-uuid)
* [local\_replica](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica)
  + [copy\_from\_remote](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica-copy_from_remote)
  + [delete](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica-delete)
  + [list](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica-list)
  + [cmeta](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica-cmeta)
    - [print\_replica\_uuids](http://kudu.apache.org/docs/command_line_tools_reference.html#cmeta-print_replica_uuids)
    - [rewrite\_raft\_config](http://kudu.apache.org/docs/command_line_tools_reference.html#cmeta-rewrite_raft_config)
  + [dump](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica-dump)
    - [block\_ids](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-block_ids)
    - [meta](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-meta)
    - [rowset](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-rowset)
    - [wals](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-wals)
* [master](http://kudu.apache.org/docs/command_line_tools_reference.html#master)
  + [set\_flag](http://kudu.apache.org/docs/command_line_tools_reference.html#master-set_flag)
  + [status](http://kudu.apache.org/docs/command_line_tools_reference.html#master-status)
  + [timestamp](http://kudu.apache.org/docs/command_line_tools_reference.html#master-timestamp)
  + [list](http://kudu.apache.org/docs/command_line_tools_reference.html#master-list)
* [pbc](http://kudu.apache.org/docs/command_line_tools_reference.html#pbc)
  + [dump](http://kudu.apache.org/docs/command_line_tools_reference.html#pbc-dump)
* [remote\_replica](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica)
  + [check](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-check)
  + [copy](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-copy)
  + [delete](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-delete)
  + [dump](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-dump)
  + [list](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-list)
  + [unsafe\_change\_config](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-unsafe_change_config)
* [table](http://kudu.apache.org/docs/command_line_tools_reference.html#table)
  + [delete](http://kudu.apache.org/docs/command_line_tools_reference.html#table-delete)
  + [list](http://kudu.apache.org/docs/command_line_tools_reference.html#table-list)
* [tablet](http://kudu.apache.org/docs/command_line_tools_reference.html#tablet)
  + [leader\_step\_down](http://kudu.apache.org/docs/command_line_tools_reference.html#tablet-leader_step_down)
  + [change\_config](http://kudu.apache.org/docs/command_line_tools_reference.html#tablet-change_config)
    - [add\_replica](http://kudu.apache.org/docs/command_line_tools_reference.html#change_config-add_replica)
    - [change\_replica\_type](http://kudu.apache.org/docs/command_line_tools_reference.html#change_config-change_replica_type)
    - [remove\_replica](http://kudu.apache.org/docs/command_line_tools_reference.html#change_config-remove_replica)
* [test](http://kudu.apache.org/docs/command_line_tools_reference.html#test)
  + [loadgen](http://kudu.apache.org/docs/command_line_tools_reference.html#test-loadgen)
* [tserver](http://kudu.apache.org/docs/command_line_tools_reference.html#tserver)
  + [set\_flag](http://kudu.apache.org/docs/command_line_tools_reference.html#tserver-set_flag)
  + [status](http://kudu.apache.org/docs/command_line_tools_reference.html#tserver-status)
  + [timestamp](http://kudu.apache.org/docs/command_line_tools_reference.html#tserver-timestamp)
  + [list](http://kudu.apache.org/docs/command_line_tools_reference.html#tserver-list)
* [wal](http://kudu.apache.org/docs/command_line_tools_reference.html#wal)
  + [dump](http://kudu.apache.org/docs/command_line_tools_reference.html#wal-dump)

## [Command Details](http://kudu.apache.org/docs/command_line_tools_reference.html#_command_details)

### [cluster: Operate on a Kudu cluster](http://kudu.apache.org/docs/command_line_tools_reference.html#cluster)

#### [ksck: Check the health of a Kudu cluster](http://kudu.apache.org/docs/command_line_tools_reference.html#cluster-ksck)

By default, ksck checks that master and tablet server processes are running, and that table metadata is consistent. Use the 'checksum' flag to check that tablet data is consistent (also see the 'tables' and 'tablets' flags). Use the 'checksum\_snapshot' along with 'checksum' if the table or tablets are actively receiving inserts or updates.  
**Usage:**  
kudu cluster ksck <master\_addresses> [-checksum\_cache\_blocks] [-checksum\_scan] [-checksum\_scan\_concurrency=<concurrency>] [-nochecksum\_snapshot] [-color=<color>] [-noconsensus] [-tables=<tables>] [-tablets=<tablets>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_addresses | Comma-separated list of Kudu Master addresses where each address is of form 'hostname:port' | string | none |
| checksum\_cache\_blocks (optional) | Should the checksum scanners cache the read blocks | bool | false |
| checksum\_scan (optional) | Perform a checksum scan on data in the cluster. | bool | false |
| checksum\_scan\_concurrency (optional) | Number of concurrent checksum scans to execute per tablet server. | int32 | 4 |
| checksum\_snapshot (optional) | Should the checksum scanner use a snapshot scan | bool | true |
| color (optional) | Specifies whether output should be colorized. The default value 'auto' colorizes output if the output is a terminal. The other valid values are 'always' or 'never'. | string | auto |
| consensus (optional) | Whether to check the consensus state from each tablet against the master. | bool | true |
| tables (optional) | Tables to check (comma-separated list of names). If not specified, checks all tables. | string | none |
| tablets (optional) | Tablets to check (comma-separated list of IDs) If not specified, checks all tablets. | string | none |

### [fs: Operate on a local Kudu filesystem](http://kudu.apache.org/docs/command_line_tools_reference.html#fs)

#### [check: Check a Kudu filesystem for inconsistencies](http://kudu.apache.org/docs/command_line_tools_reference.html#fs-check)

**Usage:**  
kudu fs check [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>] [-repair]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |
| repair (optional) | Repair any inconsistencies in the filesystem. | bool | false |

#### [format: Format a new Kudu filesystem](http://kudu.apache.org/docs/command_line_tools_reference.html#fs-format)

**Usage:**  
kudu fs format [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>] [-uuid=<uuid>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |
| uuid (optional) | The uuid to use in the filesystem. If not provided, one is generated | string | none |

#### [dump: Dump a Kudu filesystem](http://kudu.apache.org/docs/command_line_tools_reference.html#fs-dump)

##### [cfile: Dump the contents of a CFile (column file)](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-cfile)

**Usage:**  
kudu fs dump cfile <block\_id> [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>] [-noprint\_meta] [-noprint\_rows]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| block\_id | block identifier | string | none |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |
| print\_meta (optional) | Include metadata in output | bool | true |
| print\_rows (optional) | Print each row in the CFile | bool | true |

##### [tree: Dump the tree of a Kudu filesystem](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-tree)

**Usage:**  
kudu fs dump tree [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |

##### [uuid: Dump the UUID of a Kudu filesystem](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-uuid)

**Usage:**  
kudu fs dump uuid [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |

### [local\_replica: Operate on local tablet replicas via the local filesystem](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica)

#### [copy\_from\_remote: Copy a tablet replica from a remote server](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica-copy_from_remote)

**Usage:**  
kudu local\_replica copy\_from\_remote <tablet\_id> <source> [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tablet\_id | Tablet Identifier | string | none |
| source | Source RPC address of form hostname:port | string | none |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |

#### [delete: Delete a tablet replica from the local filesystem. By default, leaves a tombstone record.](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica-delete)

**Usage:**  
kudu local\_replica delete <tablet\_id> [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>] [-clean\_unsafe]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tablet\_id | Tablet Identifier | string | none |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |
| clean\_unsafe (optional) | Delete the local replica completely, not leaving a tombstone. This is not guaranteed to be safe because it also removes the consensus metadata (including Raft voting record) for the specified tablet, which violates the Raft vote durability requirements. | bool | false |

#### [list: Show list of tablet replicas in the local filesystem](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica-list)

**Usage:**  
kudu local\_replica list [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>] [-list\_detail]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |
| list\_detail (optional) | Print partition info for the local replicas | bool | false |

#### [cmeta: Operate on a local tablet replica’s consensus metadata file](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica-cmeta)

##### [print\_replica\_uuids: Print all tablet replica peer UUIDs found in a tablet’s Raft configuration](http://kudu.apache.org/docs/command_line_tools_reference.html#cmeta-print_replica_uuids)

**Usage:**  
kudu local\_replica cmeta print\_replica\_uuids <tablet\_id> [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tablet\_id | Tablet Identifier | string | none |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |

##### [rewrite\_raft\_config: Rewrite a tablet replica’s Raft configuration](http://kudu.apache.org/docs/command_line_tools_reference.html#cmeta-rewrite_raft_config)

**Usage:**  
kudu local\_replica cmeta rewrite\_raft\_config <tablet\_id> <peers>…​ [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tablet\_id | Tablet Identifier | string | none |
| peers…​ | List of peers where each peer is of form 'uuid:hostname:port' | string | none |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |

#### [dump: Dump a Kudu filesystem](http://kudu.apache.org/docs/command_line_tools_reference.html#local_replica-dump)

##### [block\_ids: Dump the IDs of all blocks belonging to a local replica](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-block_ids)

**Usage:**  
kudu local\_replica dump block\_ids <tablet\_id> [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tablet\_id | Tablet Identifier | string | none |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |

##### [meta: Dump the metadata of a local replica](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-meta)

**Usage:**  
kudu local\_replica dump meta <tablet\_id> [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tablet\_id | Tablet Identifier | string | none |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |

##### [rowset: Dump the rowset contents of a local replica](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-rowset)

**Usage:**  
kudu local\_replica dump rowset <tablet\_id> [-dump\_data] [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>] [-metadata\_only] [-nrows=<nrows>] [-rowset\_index=<index>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tablet\_id | Tablet Identifier | string | none |
| dump\_data (optional) | Dump the data for each column in the rowset. | bool | false |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |
| metadata\_only (optional) | Only dump the block metadata when printing blocks. | bool | false |
| nrows (optional) | Number of rows to dump | int64 | 0 |
| rowset\_index (optional) | Index of the rowset in local replica, default value(-1) will dump all the rowsets of the local replica | int64 | -1 |

##### [wals: Dump all WAL (write-ahead log) segments of a local replica](http://kudu.apache.org/docs/command_line_tools_reference.html#dump-wals)

**Usage:**  
kudu local\_replica dump wals <tablet\_id> [-fs\_wal\_dir=<dir>] [-fs\_data\_dirs=<dirs>] [-print\_entries=<entries>] [-noprint\_meta] [-truncate\_data=<data>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tablet\_id | Tablet Identifier | string | none |
| fs\_wal\_dir (optional) | Directory with write-ahead logs. If this is not specified, the program will not start. May be the same as fs\_data\_dirs | string | none |
| fs\_data\_dirs (optional) | Comma-separated list of directories with data blocks. If this is not specified, fs\_wal\_dir will be used as the sole data block directory. | string | none |
| print\_entries (optional) | How to print entries: false|0|no = don’t print true|1|yes|decoded = print them decoded pb = print the raw protobuf id = print only their ids | string | decoded |
| print\_meta (optional) | Include metadata in output | bool | true |
| truncate\_data (optional) | Truncate the data fields to the given number of bytes before printing. Set to 0 to disable | int32 | 100 |

### [master: Operate on a Kudu Master](http://kudu.apache.org/docs/command_line_tools_reference.html#master)

#### [set\_flag: Change a gflag value on a Kudu Master](http://kudu.apache.org/docs/command_line_tools_reference.html#master-set_flag)

**Usage:**  
kudu master set\_flag <master\_address> <flag> <value> [-force]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_address | Address of a Kudu Master of form 'hostname:port'. Port may be omitted if the Master is bound to the default port. | string | none |
| flag | Name of the gflag | string | none |
| value | New value for the gflag | string | none |
| force (optional) | If true, allows the set\_flag command to set a flag which is not explicitly marked as runtime-settable. Such flag changes may be simply ignored on the server, or may cause the server to crash. | bool | false |

#### [status: Get the status of a Kudu Master](http://kudu.apache.org/docs/command_line_tools_reference.html#master-status)

**Usage:**  
kudu master status <master\_address>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_address | Address of a Kudu Master of form 'hostname:port'. Port may be omitted if the Master is bound to the default port. | string | none |

#### [timestamp: Get the current timestamp of a Kudu Master](http://kudu.apache.org/docs/command_line_tools_reference.html#master-timestamp)

**Usage:**  
kudu master timestamp <master\_address>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_address | Address of a Kudu Master of form 'hostname:port'. Port may be omitted if the Master is bound to the default port. | string | none |

#### [list: List masters in a Kudu cluster](http://kudu.apache.org/docs/command_line_tools_reference.html#master-list)

**Usage:**  
kudu master list <master\_addresses> [-columns=<columns>] [-format=<format>] [-timeout\_ms=<ms>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_addresses | Comma-separated list of Kudu Master addresses where each address is of form 'hostname:port' | string | none |
| columns (optional) | Comma-separated list of master info fields to include in output. Possible values: uuid, rpc-addresses, http-addresses, version, and seqno | string | uuid,rpc-addresses |
| format (optional) | Format to use for printing list output tables. Possible values: pretty, space, tsv, csv, and json | string | pretty |
| timeout\_ms (optional) | RPC timeout in milliseconds | int64 | 60000 |

### [pbc: Operate on PBC (protobuf container) files](http://kudu.apache.org/docs/command_line_tools_reference.html#pbc)

#### [dump: Dump a PBC (protobuf container) file](http://kudu.apache.org/docs/command_line_tools_reference.html#pbc-dump)

**Usage:**  
kudu pbc dump <path> [-oneline]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| path | path to PBC file | string | none |
| oneline (optional) | print each protobuf on a single line | bool | false |

### [remote\_replica: Operate on remote tablet replicas on a Kudu Tablet Server](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica)

#### [check: Check if all tablet replicas on a Kudu Tablet Server are running](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-check)

**Usage:**  
kudu remote\_replica check <tserver\_address>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tserver\_address | Address of a Kudu Tablet Server of form 'hostname:port'. Port may be omitted if the Tablet Server is bound to the default port. | string | none |

#### [copy: Copy a tablet replica from one Kudu Tablet Server to another](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-copy)

**Usage:**  
kudu remote\_replica copy <tablet\_id> <src\_address> <dst\_address> [-force\_copy]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tablet\_id | Tablet Identifier | string | none |
| src\_address | Address of a Kudu Tablet Server of form 'hostname:port'. Port may be omitted if the Tablet Server is bound to the default port. | string | none |
| dst\_address | Address of a Kudu Tablet Server of form 'hostname:port'. Port may be omitted if the Tablet Server is bound to the default port. | string | none |
| force\_copy (optional) | Force the copy when the destination tablet server has this replica | bool | false |

#### [delete: Delete a tablet replica from a Kudu Tablet Server](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-delete)

**Usage:**  
kudu remote\_replica delete <tserver\_address> <tablet\_id> <reason>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tserver\_address | Address of a Kudu Tablet Server of form 'hostname:port'. Port may be omitted if the Tablet Server is bound to the default port. | string | none |
| tablet\_id | Tablet Identifier | string | none |
| reason | Reason for deleting the replica | string | none |

#### [dump: Dump the data of a tablet replica on a Kudu Tablet Server](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-dump)

**Usage:**  
kudu remote\_replica dump <tserver\_address> <tablet\_id>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tserver\_address | Address of a Kudu Tablet Server of form 'hostname:port'. Port may be omitted if the Tablet Server is bound to the default port. | string | none |
| tablet\_id | Tablet Identifier | string | none |

#### [list: List all tablet replicas on a Kudu Tablet Server](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-list)

**Usage:**  
kudu remote\_replica list <tserver\_address>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tserver\_address | Address of a Kudu Tablet Server of form 'hostname:port'. Port may be omitted if the Tablet Server is bound to the default port. | string | none |

#### [unsafe\_change\_config: Force the specified replica to adopt a new Raft config](http://kudu.apache.org/docs/command_line_tools_reference.html#remote_replica-unsafe_change_config)

The members of the new Raft config must be a subset of (or the same as) the members of the existing committed Raft config on that replica.  
**Usage:**  
kudu remote\_replica unsafe\_change\_config <tserver\_address> <tablet\_id> <peer uuids>…​   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tserver\_address | Address of a Kudu Tablet Server of form 'hostname:port'. Port may be omitted if the Tablet Server is bound to the default port. | string | none |
| tablet\_id | Tablet Identifier | string | none |
| peer uuids…​ | List of peer uuids to be part of new config | string | none |

### [table: Operate on Kudu tables](http://kudu.apache.org/docs/command_line_tools_reference.html#table)

#### [delete: Delete a table](http://kudu.apache.org/docs/command_line_tools_reference.html#table-delete)

**Usage:**  
kudu table delete <master\_addresses> <table\_name>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_addresses | Comma-separated list of Kudu Master addresses where each address is of form 'hostname:port' | string | none |
| table\_name | Name of the table to delete | string | none |

#### [list: List all tables](http://kudu.apache.org/docs/command_line_tools_reference.html#table-list)

**Usage:**  
kudu table list <master\_addresses> [-list\_tablets]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_addresses | Comma-separated list of Kudu Master addresses where each address is of form 'hostname:port' | string | none |
| list\_tablets (optional) | Include tablet and replica UUIDs in the output | bool | false |

### [tablet: Operate on remote Kudu tablets](http://kudu.apache.org/docs/command_line_tools_reference.html#tablet)

#### [leader\_step\_down: Force the tablet’s leader replica to step down](http://kudu.apache.org/docs/command_line_tools_reference.html#tablet-leader_step_down)

**Usage:**  
kudu tablet leader\_step\_down <master\_addresses> <tablet\_id>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_addresses | Comma-separated list of Kudu Master addresses where each address is of form 'hostname:port' | string | none |
| tablet\_id | Tablet Identifier | string | none |

#### [change\_config: Change a tablet’s Raft configuration](http://kudu.apache.org/docs/command_line_tools_reference.html#tablet-change_config)

##### [add\_replica: Add a new replica to a tablet’s Raft configuration](http://kudu.apache.org/docs/command_line_tools_reference.html#change_config-add_replica)

**Usage:**  
kudu tablet change\_config add\_replica <master\_addresses> <tablet\_id> <replica\_uuid> <replica\_type>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_addresses | Comma-separated list of Kudu Master addresses where each address is of form 'hostname:port' | string | none |
| tablet\_id | Tablet Identifier | string | none |
| replica\_uuid | New replica’s UUID | string | none |
| replica\_type | New replica’s type. Must be VOTER or NON-VOTER. | string | none |

##### [change\_replica\_type: Change the type of an existing replica in a tablet’s Raft configuration](http://kudu.apache.org/docs/command_line_tools_reference.html#change_config-change_replica_type)

**Usage:**  
kudu tablet change\_config change\_replica\_type <master\_addresses> <tablet\_id> <replica\_uuid> <replica\_type>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_addresses | Comma-separated list of Kudu Master addresses where each address is of form 'hostname:port' | string | none |
| tablet\_id | Tablet Identifier | string | none |
| replica\_uuid | Existing replica’s UUID | string | none |
| replica\_type | Existing replica’s new type. Must be VOTER or NON-VOTER. | string | none |

##### [remove\_replica: Remove an existing replica from a tablet’s Raft configuration](http://kudu.apache.org/docs/command_line_tools_reference.html#change_config-remove_replica)

**Usage:**  
kudu tablet change\_config remove\_replica <master\_addresses> <tablet\_id> <replica\_uuid>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_addresses | Comma-separated list of Kudu Master addresses where each address is of form 'hostname:port' | string | none |
| tablet\_id | Tablet Identifier | string | none |
| replica\_uuid | Existing replica’s UUID | string | none |

### [test: Run various tests against a Kudu cluster](http://kudu.apache.org/docs/command_line_tools_reference.html#test)

#### [loadgen: Run load generation test with optional scan afterwards](http://kudu.apache.org/docs/command_line_tools_reference.html#test-loadgen)

Run load generation tool which inserts auto-generated data into already existing or auto-created table as fast as possible. If requested, also run scan over the inserted rows to check whether the actual count or inserted rows matches the expected one.  
**Usage:**  
kudu test loadgen <master\_addresses> [-buffer\_flush\_watermark\_pct=<pct>] [-buffer\_size\_bytes=<bytes>] [-buffers\_num=<num>] [-flush\_per\_n\_rows=<rows>] [-keep\_auto\_table] [-num\_rows\_per\_thread=<thread>] [-num\_threads=<threads>] [-run\_scan] [-seq\_start=<start>] [-show\_first\_n\_errors=<errors>] [-string\_fixed=<fixed>] [-string\_len=<len>] [-table\_name=<name>] [-table\_num\_buckets=<buckets>] [-table\_num\_replicas=<replicas>] [-use\_random]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_addresses | Comma-separated list of master addresses to run against. Addresses are in 'hostname:port' form where port may be omitted if a master server listens at the default port. | string | none |
| buffer\_flush\_watermark\_pct (optional) | Mutation buffer flush watermark, in percentage of total size. | double | 0.5 |
| buffer\_size\_bytes (optional) | Size of the mutation buffer, per session (bytes). | int32 | 4194304 |
| buffers\_num (optional) | Number of mutation buffers per session. | int32 | 2 |
| flush\_per\_n\_rows (optional) | Perform async flush per given number of rows added. Setting to non-zero implicitly turns on manual flush mode. | int32 | 0 |
| keep\_auto\_table (optional) | If using the auto-generated table, enabling this option retains the table populated with the data after the test finishes. By default, the auto-generated table is dropped after sucessfully finishing the test. NOTE: this parameter has no effect if using already existing table (see the '--table\_name' flag): the existing tables nor their data are never dropped/deleted. | bool | false |
| num\_rows\_per\_thread (optional) | Number of rows each thread generates and inserts; 0 means unlimited. All rows generated by a thread are inserted in the context of the same session. | uint64 | 1000 |
| num\_threads (optional) | Number of generator threads to run. Each thread runs its own KuduSession. | int32 | 2 |
| run\_scan (optional) | Whether to run post-insertion scan to verify that the count of the inserted rows matches the expected number. If enabled, the scan is run only if no errors were encountered while inserting the generated rows. | bool | false |
| seq\_start (optional) | Initial value for the generator in sequential mode. This is useful when running multiple times against already existing table: for every next run, set this flag to (num\_threads \* num\_rows\_per\_thread \* column\_num + seq\_start). | uint64 | 0 |
| show\_first\_n\_errors (optional) | Output detailed information on the specified number of first n errors (if any). | int32 | 0 |
| string\_fixed (optional) | Pre-defined string to write into binary and string columns. Client generates more data per second using pre-defined string compared with auto-generated strings of the same length if run with the same CPU/memory configuration. If left empty, then auto-generated strings of length specified by the '--string\_len' parameter are used instead. | string | none |
| string\_len (optional) | Length of strings to put into string and binary columns. This parameter is not in effect if '--string\_fixed' is specified. | int32 | 32 |
| table\_name (optional) | Name of an existing table to use for the test. The test will determine the structure of the table schema and populate it with data accordingly. If left empty, the test automatically creates a table of pre-defined columnar structure with unique name and uses it to insert auto-generated data. The auto-created table is dropped upon successful completion of the test if not overridden by the '--keep\_auto\_table' flag. If running the test against an already existing table, it’s highly recommended to use a dedicated table created just for testing purposes: the existing table nor its data is never dropped/deleted. | string | none |
| table\_num\_buckets (optional) | The number of buckets to create when this tool creates a new table. | int32 | 8 |
| table\_num\_replicas (optional) | The number of replicas for the auto-created table; 0 means 'use server-side default'. | int32 | 1 |
| use\_random (optional) | Whether to use random numbers instead of sequential ones. In case of using random numbers collisions are possible over the data for columns with unique constraint (e.g. primary key). | bool | false |

### [tserver: Operate on a Kudu Tablet Server](http://kudu.apache.org/docs/command_line_tools_reference.html#tserver)

#### [set\_flag: Change a gflag value on a Kudu Tablet Server](http://kudu.apache.org/docs/command_line_tools_reference.html#tserver-set_flag)

**Usage:**  
kudu tserver set\_flag <tserver\_address> <flag> <value> [-force]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tserver\_address | Address of a Kudu Tablet Server of form 'hostname:port'. Port may be omitted if the Tablet Server is bound to the default port. | string | none |
| flag | Name of the gflag | string | none |
| value | New value for the gflag | string | none |
| force (optional) | If true, allows the set\_flag command to set a flag which is not explicitly marked as runtime-settable. Such flag changes may be simply ignored on the server, or may cause the server to crash. | bool | false |

#### [status: Get the status of a Kudu Tablet Server](http://kudu.apache.org/docs/command_line_tools_reference.html#tserver-status)

**Usage:**  
kudu tserver status <tserver\_address>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tserver\_address | Address of a Kudu Tablet Server of form 'hostname:port'. Port may be omitted if the Tablet Server is bound to the default port. | string | none |

#### [timestamp: Get the current timestamp of a Kudu Tablet Server](http://kudu.apache.org/docs/command_line_tools_reference.html#tserver-timestamp)

**Usage:**  
kudu tserver timestamp <tserver\_address>   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| tserver\_address | Address of a Kudu Tablet Server of form 'hostname:port'. Port may be omitted if the Tablet Server is bound to the default port. | string | none |

#### [list: List tablet servers in a Kudu cluster](http://kudu.apache.org/docs/command_line_tools_reference.html#tserver-list)

**Usage:**  
kudu tserver list <master\_addresses> [-columns=<columns>] [-format=<format>] [-timeout\_ms=<ms>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| master\_addresses | Comma-separated list of Kudu Master addresses where each address is of form 'hostname:port' | string | none |
| columns (optional) | Comma-separated list of tserver info fields to include in output. Possible values: uuid, rpc-addresses, http-addresses, version, seqno, and heartbeat | string | uuid,rpc-addresses |
| format (optional) | Format to use for printing list output tables. Possible values: pretty, space, tsv, csv, and json | string | pretty |
| timeout\_ms (optional) | RPC timeout in milliseconds | int64 | 60000 |

### [wal: Operate on WAL (write-ahead log) files](http://kudu.apache.org/docs/command_line_tools_reference.html#wal)

#### [dump: Dump a WAL (write-ahead log) file](http://kudu.apache.org/docs/command_line_tools_reference.html#wal-dump)

**Usage:**  
kudu wal dump <path> [-print\_entries=<entries>] [-noprint\_meta] [-truncate\_data=<data>]   
**Arguments:**

| **Name** | **Description** | **Type** | **Default** |
| --- | --- | --- | --- |
| path | path to WAL file | string | none |
| print\_entries (optional) | How to print entries: false|0|no = don’t print true|1|yes|decoded = print them decoded pb = print the raw protobuf id = print only their ids | string | decoded |
| print\_meta (optional) | Include metadata in output | bool | true |
| truncate\_data (optional) | Truncate the data fields to the given number of bytes before printing. Set to 0 to disable | int32 | 100 |

# Known Issues and Limitations

## [Schema](http://kudu.apache.org/docs/known_issues.html#_schema)

### [Primary keys](http://kudu.apache.org/docs/known_issues.html#_primary_keys)

* The primary key may not be changed after the table is created. You must drop and recreate a table to select a new primary key.
* The columns which make up the primary key must be listed first in the schema.
* The primary key of a row may not be modified using the UPDATE functionality. To modify a row’s primary key, the row must be deleted and re-inserted with the modified key. Such a modification is non-atomic.
* Columns with DOUBLE, FLOAT, or BOOL types are not allowed as part of a primary key definition. Additionally, all columns that are part of a primary key definition must be NOT NULL.
* Auto-generated primary keys are not supported.
* Cells making up a composite primary key are limited to a total of 16KB after the internal composite-key encoding done by Kudu.

### [Columns](http://kudu.apache.org/docs/known_issues.html#_columns)

* DECIMAL, CHAR, VARCHAR, DATE, and complex types like ARRAY are not supported.
* Type and nullability of existing columns cannot be changed by altering the table.
* Tables can have a maximum of 300 columns.

### [Tables](http://kudu.apache.org/docs/known_issues.html#_tables)

* Tables must have an odd number of replicas, with a maximum of 7.
* Replication factor (set at table creation time) cannot be changed.

### [Cells (individual values)](http://kudu.apache.org/docs/known_issues.html#_cells_individual_values)

* Cells cannot be larger than 64KB before encoding or compression.

### [Other usage limitations](http://kudu.apache.org/docs/known_issues.html#_other_usage_limitations)

* Kudu is primarily designed for analytic use cases. You are likely to encounter issues if a single row contains multiple kilobytes of data.
* Secondary indexes are not supported.
* Multi-row transactions are not supported.
* Relational features, like foreign keys, are not supported.
* Identifiers such as column and table names are restricted to be valid UTF-8 strings. Additionally, a maximum length of 256 characters is enforced.
* Dropping a column does not immediately reclaim space. Compaction must run first.
* There is no way to run compaction manually, but dropping the table will reclaim the space immediately.

## [Partitioning Limitations](http://kudu.apache.org/docs/known_issues.html#_partitioning_limitations)

* Tables must be manually pre-split into tablets using simple or compound primary keys. Automatic splitting is not yet possible. Range partitions may be added or dropped after a table has been created. See [Schema Design](http://kudu.apache.org/docs/schema_design.html) for more information.
* Data in existing tables cannot currently be automatically repartitioned. As a workaround, create a new table with the new partitioning and insert the contents of the old table.
* Tablets that lose a majority of replicas (such as 1 left out of 3) require manual intervention to be repaired.

## [Cluster management](http://kudu.apache.org/docs/known_issues.html#_cluster_management)

* Rack awareness is not supported.
* Multi-datacenter is not supported.
* Rolling restart is not supported.

## [Server management](http://kudu.apache.org/docs/known_issues.html#_server_management)

* Production deployments should configure a least 4GB of memory for tablet servers, and ideally more than 10GB.
* Write ahead logs (WAL) can only be stored on one disk.
* Disk failures are not tolerated and tablets servers will crash as soon as one is detected.
* Failed disks with unrecoverable data require the formatting of all the Kudu data for that tablet server before it can be started again.
* Data directories cannot be added/removed; all must be reformatted to change the set of directories.
* Tablet servers cannot be gracefully decommissioned.
* Tablet servers can’t change address/port.
* Kudu has a hard requirement on having up-to-date NTP. Kudu masters and tablet servers will crash when out of sync.
* Kudu releases are only tested with NTP. Other time synchronization providers like Chrony may or may not work.

## [Scale](http://kudu.apache.org/docs/known_issues.html#_scale)

* Recommended maximum number of tablet servers is 100.
* Recommended maximum number of masters is 3.
* Recommended maximum amount of stored data, post-replication and post-compression, per tablet server is 4TB.
* Recommended maximum number of tablets per tablet server is 1000, post-replication.
* Maximum number of tablets per table for each tablet server is 60, post-replication, at table-creation time.

## [Replication and Backup Limitations](http://kudu.apache.org/docs/known_issues.html#_replication_and_backup_limitations)

* Kudu does not currently include any built-in features for backup and restore. Users are encouraged to use tools such as Spark or Impala to export or import tables as necessary.

## [Security Limitations](http://kudu.apache.org/docs/known_issues.html#_security_limitations)

* Authorization is only available at a system-wide, coarse-grained level. Table-level, column-level, and row-level authorization features are not available.
* Data encryption at rest is not built in. Kudu has been reported to run correctly on systems using local block device encryption (e.g. dmcrypt).
* Kudu server Kerberos principals must follow the pattern kudu/<HOST>@DEFAULT.REALM. Configuring an alternate Kerberos principal is not supported.
* Kudu’s integration with Apache Flume does not support writing to Kudu clusters that require Kerberos authentication.
* Kudu client instances retrieve authentication tokens upon first contact with the cluster. These tokens expire after one week. Use of a single Kudu client instance for more than one week is only supported by the C++ client but not by the Java client.

## [Other Known Issues](http://kudu.apache.org/docs/known_issues.html#_other_known_issues)

The following are known bugs and issues with the current release of Kudu. They will be addressed in later releases. Note that this list is not exhaustive, and is meant to communicate only the most important known issues.

* If the Kudu master is configured with the -log\_force\_fsync\_all option, tablet servers and clients will experience frequent timeouts, and the cluster may become unusable.
* If a tablet server has a very large number of tablets, it may take several minutes to start up. It is recommended to limit the number of tablets per server to 100 or fewer. Consider this limitation when pre-splitting your tables. If you notice slow start-up times, you can monitor the number of tablets per server in the web UI.
* Kerberos authentication does not function correctly on hosts which contain capital letters in their hostname.
* Kerberos authentication does not function correctly if rdns = false is configured in krb5.conf.