Hive Transactions

ACID and Transactions in Hive

Hive 3 Warning

Any transactional tables created by a Hive version prior to Hive 3 require Major Compaction to be run on every partition before upgrading to 3.0. More precisely, any partition which has had any update/delete/merge statements executed on it since the last Major Compaction, has to undergo another Major Compaction. No more update/delete/merge may happen on this partition until after Hive is upgraded to Hive 3.

What is ACID and why should you use it?

ACID stands for four traits of database transactions: Atomicity (an operation either succeeds completely or fails, it does not leave partial data), Consistency (once an application performs an operation the results of that operation are visible to it in every subsequent operation), Isolation (an incomplete operation by one user does not cause unexpected side effects for other users), and Durability (once an operation is complete it will be preserved even in the face of machine or system failure). These traits have long been expected of database systems as part of their transaction functionality.

Up until Hive 0.13, atomicity, consistency, and durability were provided at the partition level. Isolation could be provided by turning on one of the available locking mechanisms (ZooKeeper or in memory). With the addition of transactions in Hive 0.13 it is now possible to provide full ACID semantics at the row level, so that one application can add rows while another reads from the same partition without interfering with each other.

Transactions with ACID semantics have been added to Hive to address the following use cases:

1. Streaming ingest of data. Many users have tools such as Apache Flume, Apache Storm, or Apache Kafka that they use to stream data into their Hadoop cluster. While these tools can write data at rates of hundreds or more rows per second, Hive can only add partitions every fifteen minutes to an hour. Adding partitions more often leads quickly to an overwhelming number of partitions in the table. These tools could stream data into existing partitions, but this would cause readers to get dirty reads (that is, they would see data written after they had started their queries) and leave many small files in their directories that would put pressure on the NameNode. With this new functionality this use case will be supported while allowing readers to get a consistent view of the data and avoiding too many files.
2. Slow changing dimensions. In a typical star schema data warehouse, dimensions tables change slowly over time. For example, a retailer will open new stores, which need to be added to the stores table, or an existing store may change its square footage or some other tracked characteristic. These changes lead to inserts of individual records or updates of records (depending on the strategy chosen). Starting with 0.14, Hive is able to support this.
3. Data restatement. Sometimes collected data is found to be incorrect and needs correction. Or the first instance of the data may be an approximation (90% of servers reporting) with the full data provided later. Or business rules may require that certain transactions be restated due to subsequent transactions (e.g., after making a purchase a customer may purchase a membership and thus be entitled to discount prices, including on the previous purchase). Or a user may be contractually required to remove their customer's data upon termination of their relationship. Starting with Hive 0.14 these use cases can be supported via *INSERT*, *UPDATE*, and *DELETE*.
4. Bulk updates using SQL MERGE statement.

Limitations

- *BEGIN*, *COMMIT*, and *ROLLBACK* are not yet supported. All language operations are auto-commit. The plan is to support these in a future release.
- Only ORC file format is supported in this first release. The feature has been built such that transactions can be used by any storage format that can determine how updates or deletes apply to base records (basically, that has an explicit or implicit row id), but so far the integration work has only been done for ORC.

- By default transactions are configured to be off. See the Configuration section below for a discussion of which values need to be set to configure it.
- Tables must be bucketed to make use of these features. Tables in the same system not using transactions and ACID do not need to be bucketed. External tables cannot be made ACID tables since the changes on external tables are beyond the control of the compactor (HIVE-13175).
- Reading/writing to an ACID table from a non-ACID session is not allowed. In other words, the Hive transaction manager must be set to org.apache.hadoop.hive.ql.lockmgr.DbTxnManager in order to work with ACID tables.
- At this time only snapshot level isolation is supported. When a given query starts it will be provided with a consistent snapshot of the data. There is no support for dirty read, read committed, repeatable read, or serializable. With the introduction of BEGIN the intention is to support snapshot isolation for the duration of transaction rather than just a single query. Other isolation levels may be added depending on user requests.
- The existing ZooKeeper and in-memory lock managers are not compatible with transactions. There is no intention to address this issue. See Basic Design below for a discussion of how locks are stored for transactions.
- ~~Schema changes using ALTER TABLE is NOT supported for ACID tables. HIVE-11421 is tracking it.~~ Fixed in 1.3.0/2.0.0.
- Using Oracle as the Metastore DB and "datanucleus.connectionPoolingType=BONECP" may generate intermittent "No such lock.." and "No such transaction..." errors. Setting "datanucleus.connectionPoolingType=DBCP" is recommended in this case.
- LOAD DATA... statement is not supported with transactional tables. (This was not properly enforced until HIVE-16732)

Streaming APIs

Hive offers APIs for streaming data ingest and streaming mutation:

- Hive HCatalog Streaming API
- Hive Streaming API (Since Hive 3)
- HCatalog Streaming Mutation API (available in Hive 2.0.0 and later)

A comparison of these two APIs is available in the Background section of the Streaming Mutation document.

Grammar Changes

INSERT...VALUES, UPDATE, and *DELETE* have been added to the SQL grammar, starting in Hive 0.14. See LanguageManual DML for details.

Several new commands have been added to Hive's DDL in support of ACID and transactions, plus some existing DDL has been modified.

A new command *SHOW TRANSACTIONS* has been added, see Show Transactions for details.

A new command *SHOW COMPACTIONS* has been added, see Show Compactions for details.

The *SHOW LOCKS* command has been altered to provide information about the new locks associated with transactions. If you are using the ZooKeeper or in-memory lock managers you will notice no difference in the output of this command. See Show Locks for details.

A new option has been added to *ALTER TABLE* to request a compaction of a table or partition. In general users do not need to request compactions, as the system will detect the need for them and initiate the compaction. However, if compaction is turned off for a table or a user wants to compact the table at a time the system would not choose to, *ALTER TABLE* can be used to initiate the compaction. See Alter Table/Partition Compact for details. This will enqueue a request for compaction and return. To watch the progress of the compaction the user can use *SHOW COMPACTIONS*.

A new command *ABORT TRANSACTIONS* has been added, see Abort Transactions for details.

Basic Design

HDFS does not support in-place changes to files. It also does not offer read consistency in the face of writers appending to files being read by a user. In order to provide these features on top of HDFS we have followed the standard approach used in other data warehousing tools. Data for the table or partition is stored in a set of base files. New records, updates, and deletes are stored in delta files. A new set of delta files is created for each transaction (or in the case of streaming agents such as Flume or Storm, each batch of transactions) that alters a table or partition. At read time the reader merges the base and delta files, applying any updates and deletes as it reads.

Base and Delta Directories

Previously all files for a partition (or a table if the table is not partitioned) lived in a single directory. With these changes, any partitions (or tables) written with an ACID aware writer will have a directory for the base files and a directory for each set of delta files. Here is what this may look like for an unpartitioned table "t":

Filesystem Layout for Table "t"

```
hive> dfs -ls -R /user/hive/warehouse/t;
drwxr-xr-x   - ekoifman staff          0 2016-06-09 17:03 /user/hive/warehouse/t/base_0000022
-rw-r--r--   1 ekoifman staff        602 2016-06-09 17:03 /user/hive/warehouse/t/base_0000022/bucket_00000
drwxr-xr-x   - ekoifman staff          0 2016-06-09 17:06 /user/hive/warehouse/t/delta_0000023_0000023_0000
-rw-r--r--   1 ekoifman staff        611 2016-06-09 17:06 /user/hive/warehouse/t/delta_0000023_0000023_0000
/bucket_00000
```

```
drwxr-xr-x   - ekoifman staff          0 2016-06-09 17:07 /user/hive/warehouse/t/delta_0000024_0000024_0000
-rw-r--r--   1 ekoifman staff        610 2016-06-09 17:07 /user/hive/warehouse/t/delta_0000024_0000024_0000
/bucket_00000
```

Compactor

Compactor is a set of background processes running inside the Metastore to support ACID system. It consists of Initiator, Worker, Cleaner, AcidHouseKeeperService and a few others.

Delta File Compaction

As operations modify the table more and more delta files are created and need to be compacted to maintain adequate performance. There are three types of compactions, minor, major and rebalance.

- Minor compaction takes a set of existing delta files and rewrites them to a single delta file per bucket.
- Major compaction takes one or more delta files and the base file for the bucket and rewrites them into a new base file per bucket. Major compaction is more expensive but is more effective.
- More information about rebalance compaction can be found here: Rebalance compaction

All compactions are done in the background. Minor and major compactions do not prevent concurrent reads and writes of the data. Rebalance compaction uses exclusive write lock, therefore it prevents concurrent writes. After a compaction the system waits until all readers of the old files have finished and then removes the old files.

Initiator

This module is responsible for discovering which tables or partitions are due for compaction. This should be enabled in a Metastore using hive.compactor.initiator.on. There are several properties of the form *.threshold in "New Configuration Parameters for Transactions" table below that control when a compaction task is created and which type of compaction is performed. Each compaction task handles 1 partition (or whole table if the table is unpartitioned). If the number of consecutive compaction failures for a given partition exceeds hive.compactor.initiator.failed.compacts.threshold, automatic compaction scheduling will stop for this partition. See Configuration Parameters table for more info.

Worker

Each Worker handles a single compaction task. A compaction is a MapReduce job with name in the following form: <hostname>-compactor-<db>.<table>.<partition>. Each worker submits the job to the cluster (via hive.compactor.job.queue if defined) and waits for the job to finish. hive.compactor.worker.threads determines the number of Workers in each Metastore. The total number of Workers in the Hive Warehouse determines the maximum number of concurrent compactions.

Cleaner

This process is a process that deletes delta files after compaction and after it determines that they are no longer needed.

AcidHouseKeeperService

This process looks for transactions that have not heartbeated in hive.txn.timeout time and aborts them. The system assumes that a client that initiated a transaction stopped heartbeating crashed and the resources it locked should be released.

SHOW COMPACTIONS

This commands displays information about currently running compaction and recent history (configurable retention period) of compactions. This history display is available since HIVE-12353.

Also see LanguageManual DDL#ShowCompactions for more information on the output of this command and NewConfigurationParametersforTransactions/Compaction History for configuration properties affecting the output of this command. The system retains the last N entries of each type: failed, succeeded, attempted (where N is configurable for each type).

Transaction/Lock Manager

A new logical entity called "transaction manager" was added which incorporated previous notion of "database/table/partition lock manager" (hive.lock.manager with default of org.apache.hadoop.hive.ql.lockmgr.zookeeper.ZooKeeperHiveLockManager). The transaction manager is now additionally responsible for managing of transactions locks. The default DummyTxnManager emulates behavior of old Hive versions: has no transactions and uses hive.lock.manager property to create lock manager for tables, partitions and databases. A newly added DbTxnManager manages all locks /transactions in Hive metastore with DbLockManager (transactions and locks are durable in the face of server failure). This means that previous behavior of locking in ZooKeeper is not present anymore when transactions are enabled. To avoid clients dying and leaving transaction or locks dangling, a heartbeat is sent from lock holders and transaction initiators to the metastore on a regular basis. If a heartbeat is not received in the configured amount of time, the lock or transaction will be aborted.

As of Hive 1.3.0, the length of time that the DbLockManger will continue to try to acquire locks can be controlled via hive.lock.numretires and hive.lock.sleep.between.retries. When the DbLockManager cannot acquire a lock (due to existence of a competing lock), it will back off and try again after a certain time period. In order to support short running queries and not overwhelm the metastore at the same time, the DbLockManager will double the wait time

after each retry. The initial back off time is 100ms and is capped by hive.lock.sleep.between.retries. hive.lock.numretries is the total number of times it will retry a given lock request. Thus the total time that the call to acquire locks will block (given values of 100 retries and 60s sleep time) is (100ms + 200ms + 400ms + ... + 51200ms + 60s + 60s + ... + 60s) = 91m:42s:300ms.

More details on locks used by this Lock Manager.

Note that the lock manager used by DbTxnManager will acquire locks on all tables, even those without "transactional=true" property. By default, Insert operation into a non-transactional table will acquire an exclusive lock and thus block other inserts and reads. While technically correct, this is a departure from how Hive traditionally worked (i.e. w/o a lock manger). For backwards compatibility, hive.txn.strict.locking.mode (see table below) is provided which will make this lock manager acquire shared locks on insert operations on non-transactional tables. This restores previous semantics while still providing the benefit of a lock manager such as preventing table drop while it is being read. Note that for transactional tables, insert always acquires share locks since these tables implement MVCC architecture at the storage layer and are able to provide strong read consistency (Snapshot Isolation) even in presence of concurrent modification operations.

Configuration

Minimally, these configuration parameters must be set appropriately to turn on transaction support in Hive:

Client Side

- hive.support.concurrency – true
- hive.enforce.bucketing – true (Not required as of Hive 2.0)
- hive.exec.dynamic.partition.mode – nonstrict
- hive.txn.manager – org.apache.hadoop.hive.ql.lockmgr.DbTxnManager

Server Side (Metastore)

- hive.compactor.initiator.on – true (See table below for more details)
- hive.compactor.cleaner.on – true (See table below for more details)
- hive.compactor.worker.threads – a positive number on at least one instance of the Thrift metastore service

The following sections list all of the configuration parameters that affect Hive transactions and compaction. Also see Limitations above and Table Properties below.

New Configuration Parameters for Transactions

A number of new configuration parameters have been added to the system to support transactions.

Configuration key	Values	Location	Notes
hive.txn.manager	*Default:* org.apache.hadoop.hive.ql.lockmgr.DummyTxnManager *Value required for transactions:* org.apache.hadoop.hive.ql.lockmgr.DbTxnManager	Client/ HiveServer2	DummyTxnManager replicates pre Hive-0.13 behavior and provides no transactions.
hive.txn.strict.locking.mode	*Default:* true	Client/ HiveServer2	In strict mode non-ACID resources use standard R/W lock semantics, e.g. INSERT will acquire exclusive lock. In non-strict mode, for non-ACID resources, INSERT will only acquire shared lock, which allows two concurrent writes to the same partition but still lets lock manager prevent DROP TABLE etc. when the table is being written to (as of Hive 2.2.0).
hive.txn.timeout	*Default:* 300	Client/ HiveServer2/ Metastore	Time after which transactions are declared aborted if the client has not sent a heartbeat, in seconds. It's critical that this property has the same value for all components/services.[5]
hive.txn.heartbeat.threadpool.size	*Default:* 5	Client/ HiveServer2	The number of threads to use for heartbeating (as of Hive 1.3.0 and 2.0.0).
hive.timedout.txn.reaper.start	*Default:* 100s	Metastore	Time delay of first reaper (the process which aborts timed-out transactions) run after the metastore starts (as of Hive 1.3.0). Controls AcidHouseKeeperServcie above.
	Default: 180s	Metasto	Time interval describing how often the reaper (the process which aborts timed-out transactions) runs (as of Hive 1.3.0). Controls AcidHouseKeeperServcie above.

hive.timedout.txn.reaper.interval		re	
hive.txn.max.open.batch	*Default:* 1000	Client	Maximum number of transactions that can be fetched in one call to open_txns().[1]
hive.max.open.txns	*Default:* 100000	HiveServer2/ Metastore	Maximum number of open transactions. If current open transactions reach this limit, future open transaction requests will be rejected, until the number goes below the limit. (As of Hive 1.3.0 and 2.1.0.)
hive.count.open.txns.interval	*Default:* 1s	HiveServer2/ Metastore	Time in seconds between checks to count open transactions (as of Hive 1.3.0 and 2.1.0).
hive.txn.retryable.sqlex.regex	*Default:* "" (empty string)	HiveServer2/ Metastore	Comma separated list of regular expression patterns for SQL state, error code, and error message of retryable SQLExceptions, that's suitable for the Hive metastore database (as of Hive 1.3.0 and 2.1.0). For an example, see Configuration Properties.
hive.compactor.initiator.on	*Default:* false *Value required for transactions:* true (for exactly one instance of the Thrift metastore service)	Metastore	Whether to run the initiator thread on this metastore instance. Prior to Hive 1.3.0 it's critical that this is enabled on exactly one standalone metastore service instance (not enforced yet). As of Hive 1.3.0 this property may be enabled on any number of standalone metastore instances.
hive.compactor.cleaner.on	*Default:* false *Value required for transactions:* true (for exactly one instance of the Thrift metastore service)	Metastore	Whether to run the cleaner thread on this metastore instance. Before **Hive 4.0.0** Cleaner thread can be started/stopped with config hive.compactor.initiator.on. This config helps to enable/disable initiator/cleaner threads independently
hive.compactor.worker.threads	*Default:* 0 *Value required for transactions:* > 0 on at least one instance of the Thrift metastore service	Metastore	How many compactor worker threads to run on this metastore instance.[2]
hive.compactor.worker.timeout	*Default:* 86400	Metastore	Time in seconds after which a compaction job will be declared failed and the compaction re-queued.
hive.compactor.cleaner.run.interval	*Default*: 5000	Metastore	Time in milliseconds between runs of the cleaner thread. (Hive 0.14.0 and later.)
hive.compactor.check.interval	*Default:* 300	Metastore	Time in seconds between checks to see if any tables or partitions need to be compacted.[3]
hive.compactor.delta.num.threshold	*Default:* 10	Metastore	Number of delta directories in a table or partition that will trigger a minor compaction.
hive.compactor.delta.pct.threshold	*Default:* 0.1	Metastore	Percentage (fractional) size of the delta files relative to the base that will trigger a major compaction. 1 = 100%, so the default 0.1 = 10%.
hive.compactor.abortedtxn.threshold	*Default:* 1000	Metastore	Number of aborted transactions involving a given table or partition that will trigger a major compaction.
hive.compactor.aborted.txn.time.threshold	*Default*: 12h	Metastore	Age of table/partition's oldest aborted transaction when compaction will be triggered. Default time unit is: hours. Set to a negative number to disable.
hive.compactor.max.num.delta	Default: 500	Metastore	Maximum number of delta files that the compactor will attempt to handle in a single job (as of Hive 1.3.0).[4]

hive.compactor.job.queue	*Default*: "" (empty string)	Metastore	Used to specify name of Hadoop queue to which Compaction jobs will be submitted. Set to empty string to let Hadoop choose the queue (as of Hive 1.3.0).
Compaction History			
hive.compactor.history.retention.succeeded	*Default: 3*	Metastore	Number of successful compaction entries to retain in history (per partition).
hive.compactor.history.retention.failed	*Default: 3*	Metastore	Number of failed compaction entries to retain in history (per partition).
hive.compactor.history.retention.attempted	*Default: 2*	Metastore	Number of attempted compaction entries to retain in history (per partition).
hive.compactor.initiator.failed.compacts.threshold	*Default: 2*	Metastore	Number of of consecutive failed compactions for a given partition after which the Initiator will stop attempting to schedule compactions automatically. It is still possible to use ALTER TABLE to initiate compaction. Once a manually initiated compaction succeeds auto initiated compactions will resume. Note that this must be less than hive.compactor.history.retention.failed.
hive.compactor.history.reaper.interval	*Default: 2m*	Metastore	Controls how often the process to purge historical record of compactions runs.

[1]hive.txn.max.open.batch controls how many transactions streaming agents such as Flume or Storm open simultaneously. The streaming agent then writes that number of entries into a single file (per Flume agent or Storm bolt). Thus increasing this value decreases the number of delta files created by streaming agents. But it also increases the number of open transactions that Hive has to track at any given time, which may negatively affect read performance.

[2]Worker threads spawn MapReduce jobs to do compactions. They do not do the compactions themselves. Increasing the number of worker threads will decrease the time it takes tables or partitions to be compacted once they are determined to need compaction. It will also increase the background load on the Hadoop cluster as more MapReduce jobs will be running in the background. Each compaction can handle one partition at a time (or whole table if it's unpartitioned).

[3]Decreasing this value will reduce the time it takes for compaction to be started for a table or partition that requires compaction. However, checking if compaction is needed requires several calls to the NameNode for each table or partition that has had a transaction done on it since the last major compaction. So decreasing this value will increase the load on the NameNode.

[4]If the compactor detects a very high number of delta files, it will first run several partial minor compactions (currently sequentially) and then perform the compaction actually requested.

[5]If the value is not the same active transactions may be determined to be "timed out" and consequently Aborted. This will result in errors like "No such transaction...", "No such lock ..."

Configuration Values to Set for *INSERT, UPDATE, DELETE*

In addition to the new parameters listed above, some existing parameters need to be set to support *INSERT ... VALUES, UPDATE,* and *DELETE*.

Configuration key	Must be set to
hive.support.concurrency	true (default is false)
hive.enforce.bucketing	true (default is false) (Not required as of Hive 2.0)
hive.exec.dynamic.partition.mode	nonstrict (default is strict)

Configuration Values to Set for Compaction

If the data in your system is not owned by the Hive user (i.e., the user that the Hive metastore runs as), then Hive will need permission to run as the user who owns the data in order to perform compactions. If you have already set up HiveServer2 to impersonate users, then the only additional work to do is assure that Hive has the right to impersonate users from the host running the Hive metastore. This is done by adding the hostname to `hadoop.proxyuser.hive.hosts` in Hadoop's `core-site.xml` file. If you have not already done this, then you will need to configure Hive to act as a proxy user. This requires you to set up keytabs for the user running the Hive metastore and add `hadoop.proxyuser.hive.hosts` and `hadoop.proxyuser.hive.groups` to Hadoop's `core-site.xml` file. See the Hadoop documentation on secure mode for your version of Hadoop (e.g., for Hadoop 2.5.1 it is at Hadoop in Secure Mode).

Compaction pooling

More in formation on compaction pooling can be found here: Compaction pooling

Table Properties

If a table is to be used in ACID writes (insert, update, delete) then the table property "`transactional=true`" must be set on that table, starting with Hive 0.14.0. Note, once a table has been defined as an ACID table via TBLPROPERTIES ("transactional"="true"), it cannot be converted back to a non-ACID table, i.e., changing TBLPROPERTIES ("transactional"="false") is not allowed. Also, hive.txn.manager must be set to org.apache.hadoop.hive.ql.lockmgr. DbTxnManager either in hive-site.xml or in the beginning of the session before any query is run. Without those, inserts will be done in the old style; updates and deletes will be prohibited prior to HIVE-11716. Since HIVE-11716 operations on ACID tables without DbTxnManager are not allowed. Howev er, this does not apply to Hive 0.13.0.

If a table owner does not wish the system to automatically determine when to compact, then the table property "`NO_AUTO_COMPACTION`" can be set. This will prevent all automatic compactions. Manual compactions can still be done with Alter Table/Partition Compact statements.

Table properties are set with the TBLPROPERTIES clause when a table is created or altered, as described in the Create Table and Alter Table Properties sections of Hive Data Definition Language. The "`transactional`" and "`NO_AUTO_COMPACTION`" table properties are case-sensitive in Hive releases 0.x and 1.0, but they are case-insensitive starting with release 1.1.0 (HIVE-8308).

More compaction related options can be set via TBLPROPERTIES as of Hive 1.3.0 and 2.1.0. They can be set at both table-level via CREATE TABLE, and on request-level via ALTER TABLE/PARTITION COMPACT. These are used to override the Warehouse/table wide settings. For example, to override an MR property to affect a compaction job, one can add "compactor.<mr property name>=<value>" in either CREATE TABLE statement or when launching a compaction explicitly via ALTER TABLE. The "<mr property name>=<value>" will be set on JobConf of the compaction MR job. Similarly, "tblprops. <prop name>=<value>" can be used to set/override any table property which is interpreted by the code running on the cluster. Finally, "compactorthreshold.<prop name>=<value>" can be used to override properties from the "New Configuration Parameters for Transactions" table above that end with ".threshold" and control when compactions are triggered by the system. Examples:

Example: Set compaction options in TBLPROPERTIES at table level

```
CREATE TABLE table_name (
  id                int,
  name              string
)
CLUSTERED BY (id) INTO 2 BUCKETS STORED AS ORC
TBLPROPERTIES ("transactional"="true",
  "compactor.mapreduce.map.memory.mb"="2048",     -- specify compaction map job properties
  "compactorthreshold.hive.compactor.delta.num.threshold"="4",  -- trigger minor compaction if there are more
than 4 delta directories
  "compactorthreshold.hive.compactor.delta.pct.threshold"="0.5" -- trigger major compaction if the ratio
of size of delta files to
                                                                -- size of base files is greater than 50%
);
```

Example: Set compaction options in TBLPROPERTIES at request level

```
ALTER TABLE table_name COMPACT 'minor'
   WITH OVERWRITE TBLPROPERTIES ("compactor.mapreduce.map.memory.mb"="3072");  -- specify compaction map job
properties
ALTER TABLE table_name COMPACT 'major'
   WITH OVERWRITE TBLPROPERTIES ("tblprops.orc.compress.size"="8192");         -- change any other Hive table
properties
```

Talks and Presentations

Transactional Operations In Hive by Eugene Koifman at Dataworks Summit 2017, San Jose, CA, USA

- Slides
- Video

DataWorks Summit 2018, San Jose, CA, USA - Covers Hive 3 and ACID V2 features

- Slides
- Video