

Cloudera Runtime 7.0.3

Apache Atlas Reference

Date published: 2019-11-22

Date modified:

CLOUDERA

<https://docs.cloudera.com/>

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

| | |
|---|-----------|
| Apache Atlas Advanced Search language reference..... | 4 |
| Apache Atlas Statistics reference..... | 6 |
| HiveServer metadata collection..... | 9 |
| HiveServer actions that produce Atlas entities..... | 9 |
| HiveServer entities created in Atlas..... | 10 |
| HiveServer relationships..... | 15 |
| HiveServer lineage..... | 16 |
| HiveServer audit entries..... | 17 |
| HBase metadata collection..... | 18 |
| HBase actions that produce Atlas entities..... | 19 |
| HBase entities created in Atlas..... | 19 |
| Hbase lineage..... | 22 |
| HBase audit entries..... | 23 |
| Impala metadata collection..... | 23 |
| Impala actions that produce Atlas entities..... | 23 |
| Impala entities created in Atlas..... | 24 |
| Impala lineage..... | 26 |
| Impala audit entries..... | 27 |
| Spark metadata collection..... | 27 |
| Spark actions that produce Atlas entities..... | 27 |
| Spark entities created in Apache Atlas..... | 28 |
| Spark lineage..... | 29 |
| Spark relationships..... | 30 |
| Spark audit entries..... | 31 |
| Spark troubleshooting..... | 31 |

Apache Atlas Advanced Search language reference

Atlas lets you search for metadata using a domain-specific language with a SQL-like format.

If you find that the Basic Search or Free-text Search doesn't allow you to search as precisely as you would like, you can create a query in the Advanced Search interface to return exactly the results you are looking for. Advanced Search queries use a domain-specific language that is intentionally SQL-like.

Each Advanced Search query is in the form of three clauses:

```
FROM WHERE SELECT
```

Additional keywords such as GROUPBY, ORDERBY, and LIMIT can be used to affect the output.

FROM clause

The value specified in the FROM clause acts as the scope of the query. You can specify any entity type in the FROM clause. The possible entity types are the same list as in the Type search; the names are case-sensitive.

The FROM clause is required and also assumed: the first item included in the query (if not literally the word "from") is assumed to be the object of the FROM clause.

Examples

With or without FROM: To retrieve all entities of type "hive_db" use one of the following queries:

```
hive_db
from hive_db
```

If you only specify a FROM clause, Atlas returns all entities of that type.



Note: To avoid unintentional load on the server because of an overly broad search, Atlas returns a maximum of 100 results when no limit is set.

Where Clause

The WHERE clause allows for filtering over the result set identified in the FROM clause by specifying a condition of the form:

```
identifier operator 'literal'
```

The identifier is the name of a property of the entity type specified in the FROM clause. The properties for a given entity type are those shown in the Properties tab of an entity detail page. The names are case-sensitive.

Operators vary by the data type of the literal and include the following:

String: = LIKE

Numeric, Date: = <>

Boolean: =

The LIKE operator allows you to use wildcards in the literal. Asterisk (*) replaces zero to multiple values; question mark (?) replaces a single value.

The literal must be enclosed in single or double quotes. Matches are case-sensitive. Literals can be lists of values. If you specify comma-separated values in square brackets, they act as an OR operation.

Dates used in literals need to be specified using the ISO 8601 format and in single or double quotes.

Boolean values used in literals are lower case "true" and "false" without quotation marks.

You can specify multiple conditions using AND or OR operators. Note that making a list of values is more efficient than using the same identifier in multiple conditions.

Examples:

Exact string: To retrieve all entities of type `hive_table` with a specific name "time_dim", use:

```
from hive_table where name = 'time_dim'
```

Multiple conditions: To retrieve entity of type `hive_table` with name that can be either "time_dim" or "customer_dim":

```
from hive_table where name = 'time_dim' or name = 'customer_dim'
```

List of values: The query in the example above can be written using a value array:

```
from hive_table where name = ["customer_dim", "time_dim"]
```

Wildcard filtering: To retrieve entity of type `hive_table` whose name ends with '_dim':

```
from hive_table where name LIKE '*_dim'
```

To retrieve a `hive_db` whose name starts with R followed by any 3 characters, followed by rt followed by at least 1 character, followed by none or any number of characters:

```
DB where name like "R???rt?*"
```

Date Literal: To retrieve entity of type `hive_table` created within 2019 and 2020, use the date portion of the time value and specify a range using two phrases connected by AND:

```
from hive_table where createTime > '2019-01-01' and
  createTime < '2019-01-03'
```

Boolean Literal: To retrieve entity of type `hdfs_path` whose attribute `isFile` is set to true and whose name is Invoice:

```
from hdfs_path where isFile = true and name = "Invoice"
```

Select Clause

The select clause allows you to specify the properties you want returned in the search results. Properties with simple values can be returned; properties that contain other entities are not available. The property names are case sensitive.

To display column headers that are more meaningful than the system property names, you can use aliases using 'as.'

Examples

Select clause only: To retrieve entities of type "hive_table" with some of its properties:

```
from hive_table select owner, name, qualifiedName
```

WHERE and SELECT clauses: To retrieve entity of type `hive_table` for a specific table with some properties:

```
from hive_table where name = 'customer_dim' select owner, name, qualifiedName
```

Change output names using AS: To display column headers as 'Owner', 'Name' and 'FullName'.

```
from hive_table select owner as Owner, name as Name, qualifiedName as FullName
```

Advanced Searches using Classifications

You can search for entities that are tagged with a specific classification using "is" or "isa" keywords in either the From or Where clauses. Is and Isa are interchangeable.

Examples

FROM or WHERE clause: To retrieve all entities of type "hive_table" that are tagged with the "Dimension" classification, you could use any of the following queries:

```
hive_table is Dimension
from hive_table where hive_table isa Dimension
```

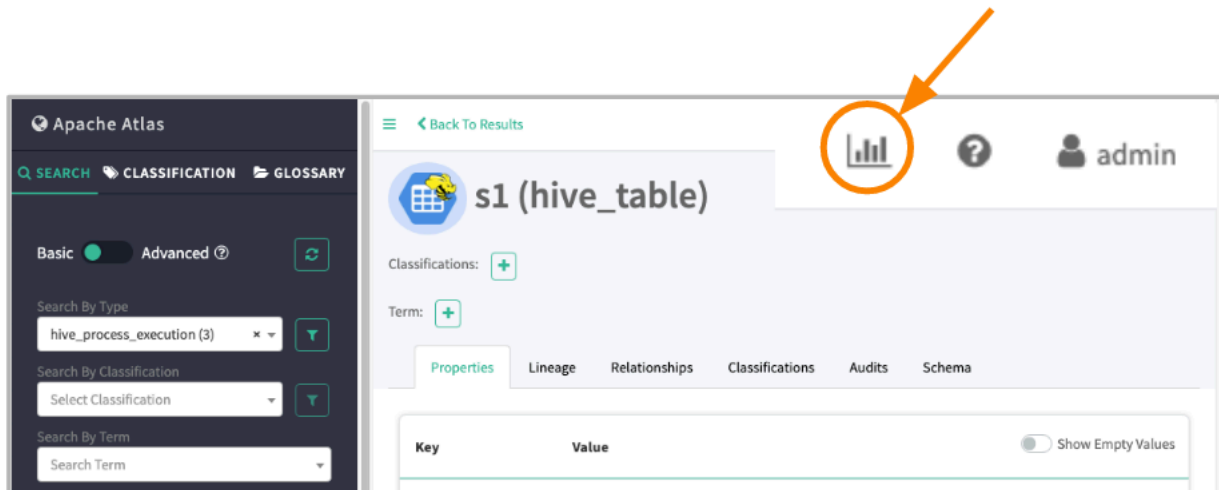
Related Information

[Apache Atlas Advanced Search](#)

Apache Atlas Statistics reference

Atlas collects statistics on the metadata it processes. Use this information to help troubleshoot problems and to gauge performance.

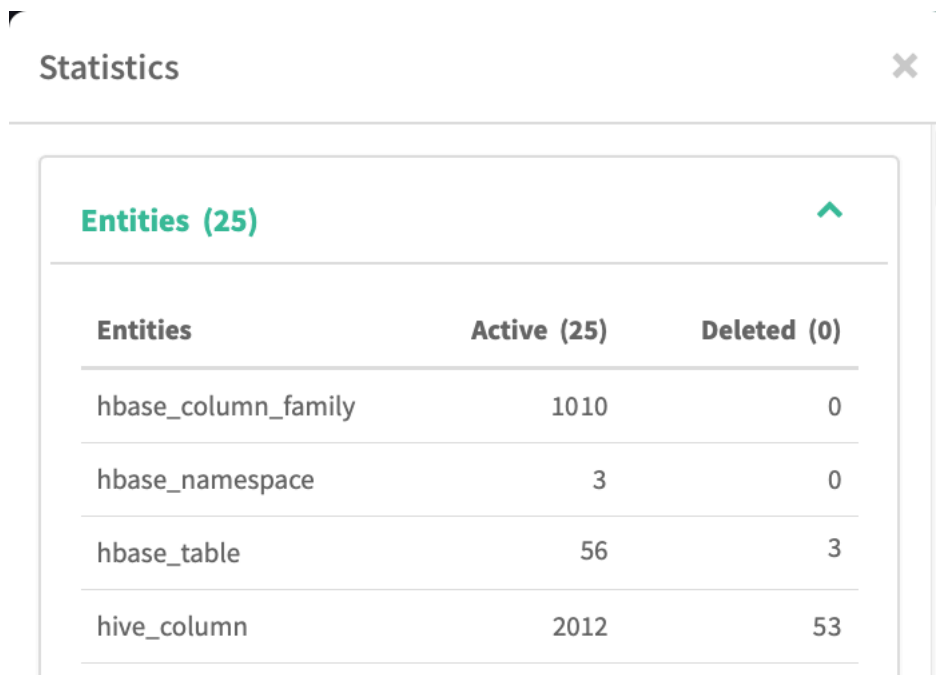
To view statistics, click the graph button in the top right corner:



The statistics available are categorized into Entity Statistics and Server Statistics:

Entity Statistics

The distribution of entity across their types. A second column gives the number of these entities that have been marked as deleted.



Statistics [Close]

Entities (25) [Up Arrow]

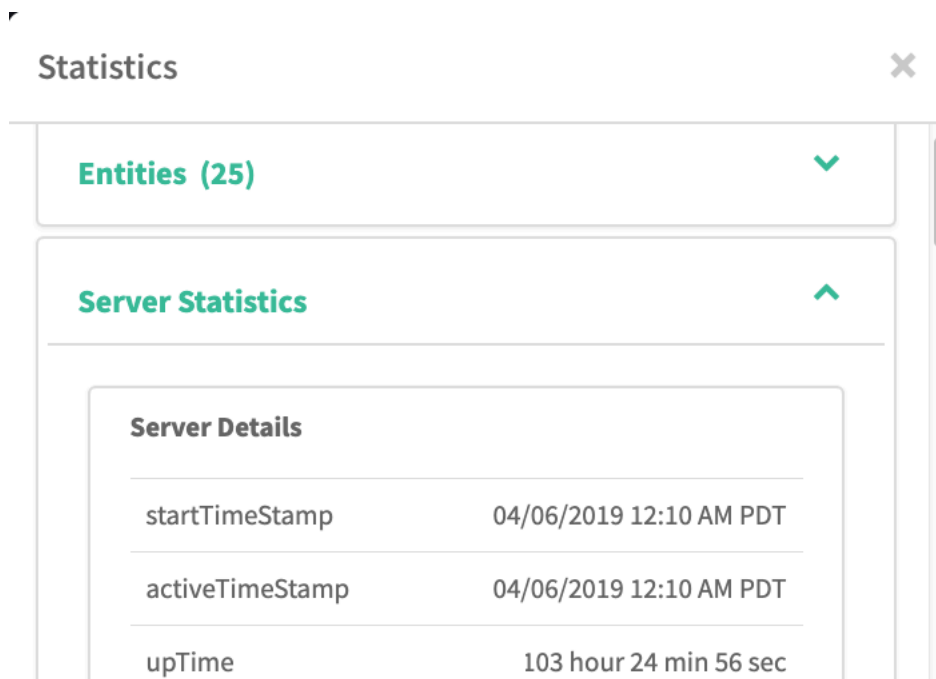
| Entities | Active (25) | Deleted (0) |
|-------------------------------------|-------------|-------------|
| hbase_column_family | 1010 | 0 |
| hbase_namespace | 3 | 0 |
| hbase_table | 56 | 3 |
| hive_column | 2012 | 53 |

Classification Statistics

A list of classifications assigned to entities and the count of entities marked with that classification. The count for each classification is a hyperlink that runs a search for entities marked with the classification.

Server Statistics

Server statistics reflect the current server session and the metadata collection messages that Atlas reads from a dedicated Kafka topic.



Statistics [Close]

Entities (25) [Down Arrow]

Server Statistics [Up Arrow]

Server Details

| | |
|-----------------|-------------------------|
| startTimeStamp | 04/06/2019 12:10 AM PDT |
| activeTimeStamp | 04/06/2019 12:10 AM PDT |
| upTime | 103 hour 24 min 56 sec |

Server Details
startTimeStampThe

The timestamp of the most recent start of the Atlas server.

activeTimeStamp

Same as the startTimeStamp unless Atlas was disabled.

upTime

The amount of time between startTimeStamp and the current time when the server was running.

statusBackendStore

The status of the Atlas server connection to the HBase namespace where entity metadata is stored.

statusIndexStore

The status of the Atlas server connection to the Solr collection where entity metadata is indexed.

collectionTime

The last time metrics were calculated.

lastMessageProcessedTime

The timestamp of the last message Atlas recorded from the Kafka topic where services publish metadata.

offsetCurrent

The index in the Kafka partition that was most recently read.

offsetStart

The index in the Kafka partition that was first read.

Notification Details: Kafka Topic-Partition**Atlas Hook**

The primary topic through which services send metadata to Atlas and Atlas sends metadata to Ranger.

Spark-Atlas Hook Topic

A supplementary topic provided for Spark communication to Atlas.

Notification Details: Message Statistics**Period**

The interval that the statistic applies to, including the total lifetime of Atlas. Each period indicated includes a timestamp for when the period started.

Count

The number of messages processed by Atlas during the period.

Avg Time (ms)

The average duration between the time that a hook published a message to the Kafka topic to the time entities were successfully created or updated.

Creates

The number of entities produced from the messages processed during the period.

Updates

The number of entities updated based on the messages processed during the period.

Deletes

The number of entities updated based on the messages processed during the period.

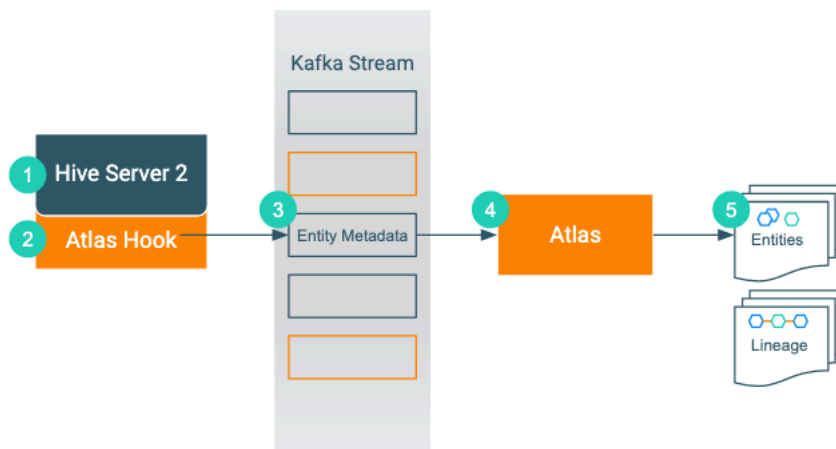
Failed

The number of messages that were received but not processed. For more information on what might have prevented these messages from being processed.

HiveServer metadata collection

Atlas can collect metadata from HiveServer, including queries and the data assets the queries affect.

An Atlas hook runs in each HiveServer instance. This hook sends metadata to Atlas for both Hive operations and Hive data assets. Operations are represented by process and process execution entities in Atlas. Hive databases, tables, views, and columns are represented by entities in Atlas. When a Hive operation involves files, the metadata for the file system and files are represented in Atlas as file system paths.



1. When an action occurs in the HiveServer instance...
2. The corresponding Atlas hook collects information for the action into metadata entities.
3. The hook publishes the metadata on a Kafka topic.
4. Atlas reads the message from the topic and determines what information will create new entities and what information updates existing entities.
5. Atlas creates and updates the appropriate entities and determines lineage from existing entities to the new entities.

The Atlas bridge for HBase pulls the same metadata as the hook, but instead of sending the metadata through Kafka, it passes message in bulk in an API call. The bridge creates entities in Atlas for all of the existing HBase namespaces, tables, columns, and column families.

HiveServer actions that produce Atlas entities

Operations that create, update, or delete Hive metadata will affect Atlas entities; operations that only affect data do not show up in Atlas.

The following table lists the HiveServer actions that produce or update metadata in Atlas.

| This Action in HiveServer... | ...Produces metadata for these Atlas entities |
|--|--|
| ALTER DATABASE, CREATE DATABASE, DROP DATABASE | hive_db, hive_db_ddl |
| ALTER TABLE, CREATE TABLE, CREATE TABLE AS SELECT, DROP TABLE | hive_process, hive_process_execution, hive_table, hive_table_ddl, hive_column, hive_column_lineage, hive_storagedesc, hdfs_path |

| This Action in HiveServer... | ...Produces metadata for these Atlas entities |
|--|---|
| ALTER VIEW, ALTERVIEW_AS_SELECT, CREATE VIEW, CREATE VIEW AS SELECT, DROP VIEW | hive_process, hive_process_execution, hive_table, hive_column, hive_column_lineage, hive_table_ddl |
| INSERT INTO (SELECT), INSERT OVERWRITE | hive_process, hive_process_execution |

Notable actions in HiveServer that do NOT produce process or process execution entities in Atlas, meaning that no lineage is produced for these operations:

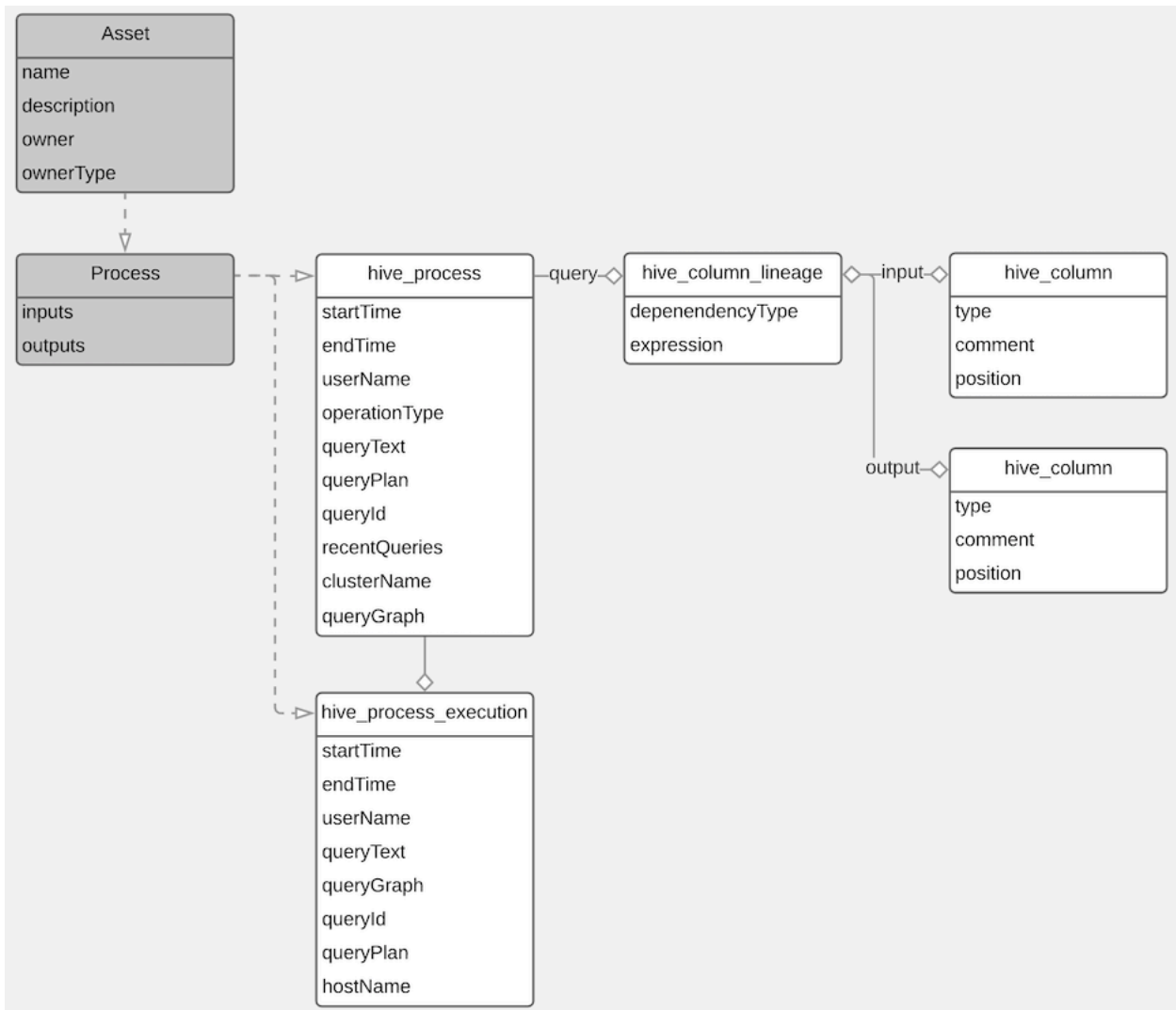
- SELECT

HiveServer entities created in Atlas

Each HiveServer entity in Atlas includes detailed metadata collected from Hive.

The following diagrams show a summary of the entities created in Atlas for Hive operations and assets. The supertypes that contribute attributes to the entity types are shaded.

Figure 1: Atlas Entity Types for HiveServer Data Sets



The metadata collected for each entity type is as follows:

Hive Process

| Identifier | Example content |
|---------------|---|
| typeName | hive_process |
| guid | System generated ID. This value is used to identify the entity in the Atlas Dashboard URL. |
| qualifiedName | <database>.<target table>@<clustername>:<generated ID> The generated ID is distinct from the GUID. |
| name | Text of the query. |
| inputs | List of the input tables or views, including each entity's type name and the qualified name. |
| outputs | List of the output objects, including each entity's type name and the qualified name. |
| recentQueries | Last query executed (duplicated in process_execution). |
| operationType | One of the operations that triggers metadata collection. |
| queryPlan | Reserved for future use. |

Hive Process Execution

| Identifier | Example Content |
|-----------------------|---|
| typeName | hive_process_execution |
| guid | System generated ID. This value is used to identify the entity in the Atlas Dashboard URL. |
| qualifiedName | <database>.<target table>@<clustername>:<ID from process qualified name>:<ID from the process execution name>:<generated ID for this process execution> |
| name | Text of the query with a system-generated ID added to the end. |
| queryText | Text of the query. |
| queryPlan | Reserved for future use. |
| queryId | impala_<date as yyyyymmddhhmmss>_<generated id> |
| startTime | Query start time. |
| endTime | Query end time. |
| userName | The user who ran the query. |
| Relationship: Process | One process to one or more process executions. hive_process_process_execution |

Hive Database

| Identifier | Example Content |
|----------------------------|--|
| typeName | hive_db |
| guid | System generated ID. This value is used to identify the entity in the Atlas Dashboard URL. |
| qualifiedName | <database>@<clustername> |
| name | Database name as reported from Hive. |
| clusterName | Cluster name. |
| location | The file system path where the backing files for the database are stored. This could be an HDFS path, an AWS S3 object, or an Azure data storage location. |
| owner | The user who initially created the database. |
| ownerType | The principal type of the database owner. Could be USER, ROLE, or GROUP. |
| parameters | Additional key-value pair metadata that comes from Hive such as table size, number of rows, and number of storage files. |
| Relationship: Table | One database to many tables. hive_table_db |
| Relationship: Database DDL | One database to many database DDL entities. hive_db_ddl_queries |

Hive Table

| Identifier | Example Content |
|---------------|--|
| typeName | hive_table |
| guid | System generated ID. This value is used to identify the entity in the Atlas Dashboard URL. |
| qualifiedName | <database>.<tablename>@<clustername> |
| name | Table name. |
| columns | List of the columns defined in the table. The Atlas Dashboard shows these as links to the column entity details. |
| owner | The user who created the table. |

| Identifier | Example Content |
|------------------------------------|--|
| parameters | Table details from HiveServer such as: <ul style="list-style-type: none"> totalSize External numFiles transient_lastDdlTime bucketing_version |
| retention | Provided by HS2. Integer value |
| sd | The location of the table data, the storage description. <database>.<table>@<clustername>_storage |
| tableType | How the table was created: one of EXTERNAL_TABLE, VIRTUAL_VIEW, or MANAGED_TABLE. |
| Relationship: Database | One database to many tables. hive_table_db |
| Relationship: Columns | One table to one or more columns. hive_table_columns |
| Relationship: Partition Key Column | One table to one or more columns that are partition keys. hive_table_partitionkeys |
| Relationship: Storage Description | One table to one storage description. hive_table_storagedesc |
| Relationship: DDL | One table to many DDL entities. hive_table_ddl_queries |

Hive Column

| Identifier | Example Content |
|------------------------------------|--|
| typeName | hive_column |
| comment | Metadata from Hive from the column description. |
| name | Column name as reported by HMS. |
| owner | Table owner name as reported by HMS. |
| position | This column's position in the list of columns in a zero-based index. |
| qualifiedName | <database>.<table>.<column>@<clustername> |
| table | Table name. Also modeled as relationship. |
| type | Column data type as reported by HMS. |
| Relationship: table | One table to one or more columns. hive_table_columns |
| Relationship: inputToProcesses | The hive_column_lineage entities that include this column in the input to a transformation. The relationship type is dataset_process_inputs. |
| Relationship: outputFromProcesses | The hive_column_lineage entities that include this column in the output to a transformation. The relationship type is process_dataset_outputs. |
| Relationship: Table | One table to one or more columns. hive_table_columns |
| Relationship: Partition Key Column | One table to one or more columns that are partition keys. hive_table_partitionkeys |

Hive Column Lineage

| Identifier | Example Content |
|----------------|--|
| typeName | hive_column_lineage |
| dependencyType | The type of relationship between the input and output columns; one of SIMPLE, EXPRESSION, or SCRIPT. |
| name | <database>.<table>@<clustername>:<generated ID>:<output_column> |

| Identifier | Example Content |
|-----------------------------------|--|
| inputs | List of 0 or more hive_column entities that contributed to the output columns. This is a legacy model component: the more current model uses a relationship attribute. |
| outputs | This is a legacy model component: the more current model uses a relationship attribute. |
| qualifiedName | Same as name. |
| query | Name of the hive_process entity that produced this lineage. This is a legacy model component: the more current model uses a relationship attribute. |
| Relationship: Process | Name of the hive_process entity that produced this lineage. hive_process_column_lineage |
| Relationship: inputToProcesses | List of 0 or more hive_column entities that contributed to the output columns. |
| Relationship: outputFromProcesses | List of 0 or more hive_column entities that were produced in the process. |

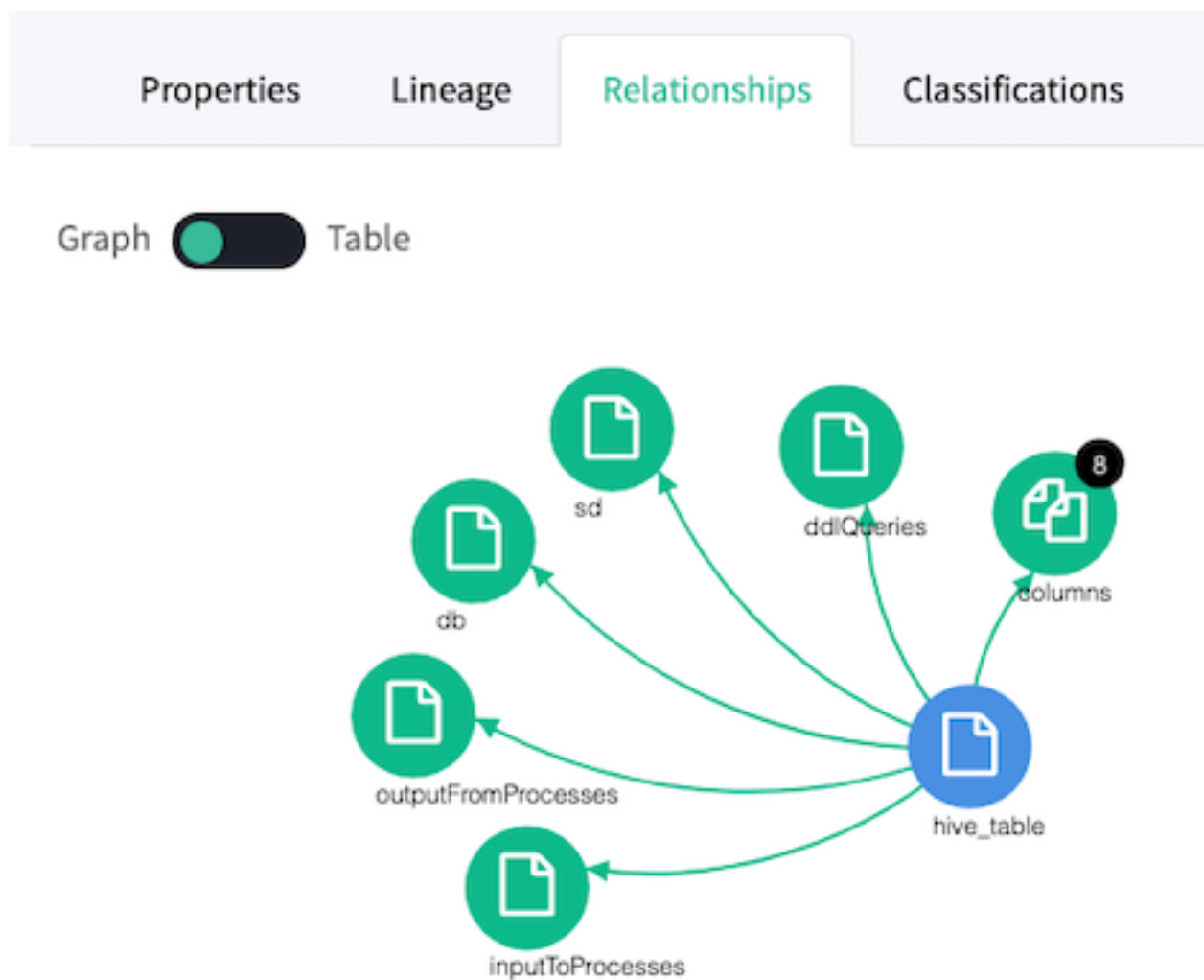
Hive Storage Description

| Identifier | Example Content |
|------------------------|---|
| typeName | hive_storagedesc |
| compressed | Metadata from Hive indicating whether the table is stored compressed. |
| inputFormat | Metadata from Hive indicating the storage input format. |
| outputFormat | Metadata from Hive indicating the storage output format. |
| parameters | Additional metadata from Hive in the form of key-value pairs. |
| qualifiedName | <database>.<table>@<clustername>_storage |
| serdeInfo | Metadata from Hive indicating the serialization/deserialization implementation used to write/read table data. |
| sortCols | Metadata from Hive listing the column or columns used to sort the table data. |
| storedAsSubDirectories | Metadata from Hive indicating whether a skewed table uses the list bucketing feature, which creates subdirectories for skewed values. |
| numBuckets | Metadata from Hive indicating the number of buckets for bucketed tables. Non-bucketed tables are indicated by -1. |
| table | The table that this storage description holds data for. Also represented as a relationship. |
| Relationship: table | The table that this storage description holds data for. |

HiveServer relationships

Atlas shows the entities related to this entity in the Relationships tab in the Dashboard.

The Relationship tab shows the relationships that exist for an entity. Use this view to navigate among related entities.



HiveServer lineage

Atlas collects metadata from HiveServer to represent the lineage among data assets.

The Atlas lineage graph shows the input and output processes that the current entity participated in, specifically those relationships modeled as “inputToProcesses” and “outputFromProcesses.” Entities are included if they were inputs to processes that lead to the current entity or they are output from processes for which the current entity was an input. HiveServer processes follow this pattern.









claim_savings (hive_table)

Classifications: DATA_QUALITY +


Term: +

Properties Lineage Relationships Classifications Audits Schema

○ Current Entity → → Lineage → → Impact

create external t... claim_savings create view if no... claims_view



Related Information
[Viewing lineage](#)

HiveServer audit entries

Atlas lists changes to metadata entities in the Audit tab in the Dashboard.

Atlas tracks the lifecycle of each Hive entity, including its creation, update, and deletion. User access and actions that affect the data content of the source asset are not included in the audit.

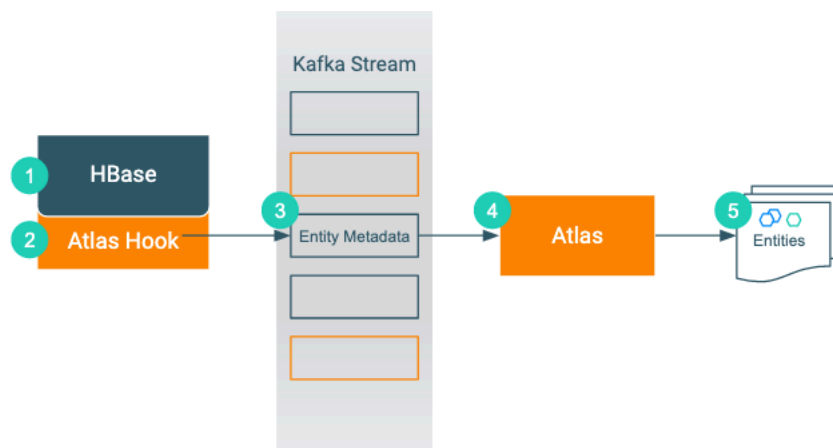
Showing 8 records From 1 - 25

| Users ↕ | Timestamp ↕ | Actions ↕ | Tools |
|---------|---|----------------|------------------------|
| hive | Fri Jul 12 2019 19:48:58 GMT-0700 (Pacific Daylight Time) | Entity Updated | Detail |
| hive | Fri Jul 12 2019 18:36:51 GMT-0700 (Pacific Daylight Time) | Entity Updated | Detail |
| hive | Fri Jul 12 2019 18:36:50 GMT-0700 (Pacific Daylight Time) | Entity Updated | Detail |
| hive | Fri Jul 12 2019 18:36:49 GMT-0700 (Pacific Daylight Time) | Entity Updated | Detail |
| hive | Fri Jul 12 2019 18:36:47 GMT-0700 (Pacific Daylight Time) | Entity Updated | Detail |
| hive | Fri Jul 12 2019 18:36:46 GMT-0700 (Pacific Daylight Time) | Entity Updated | Detail |
| hive | Fri Jul 12 2019 18:36:45 GMT-0700 (Pacific Daylight Time) | Entity Updated | Detail |
| hive | Fri Jul 12 2019 18:36:44 GMT-0700 (Pacific Daylight Time) | Entity Created | Detail |

HBase metadata collection

Atlas can collect metadata from HBase that describes the data assets HBase manages.

An Atlas hook runs in each HBase instance. This hook sends metadata to Atlas for HBase data assets. HBase namespaces, tables, columns, and column families are represented by entities in Atlas.



1. When an action occurs in the HBase instance...
2. The corresponding Atlas hook collects information for the action into metadata entities.
3. The hook publishes the metadata on a Kafka topic.
4. Atlas reads the message from the topic and determines what information will create new entities and what information updates existing entities.
5. Atlas creates and updates the appropriate entities.

The Atlas bridge for HBase pulls the same metadata as the hook, but instead of sending the metadata through Kafka, it passes message in bulk in an API call. The bridge creates entities in Atlas for all of the existing HBase namespaces, tables, columns, and column families.

HBase actions that produce Atlas entities

As data assets are created in HBase, Atlas generates entities to represent those assets. Atlas does not create processes to represent HBase operations.

The following table lists the HBase actions that produce or update metadata in Atlas.

| This Action in HBase... | ...Produces metadata for these Atlas entities |
|---|---|
| alter_async | hbase_namespace, hbase_table, hbase_column_family |
| create_namespace, alter_namespace, drop_namespace | hbase_namespace |
| create table, alter table, drop table, drop_all tables | alter table (create column family), alter table (alter column family), alter table (delete column family) |
| alter table (create column family), alter table (alter column family), alter table (delete column family) | hive_process, hive_process_execution |

Notable actions in HBase that do NOT produce metadata entities include any actions that affect only data and not metadata. In addition, Atlas does not collect metadata for HBase columns. Actions that do not create Atlas entities include:

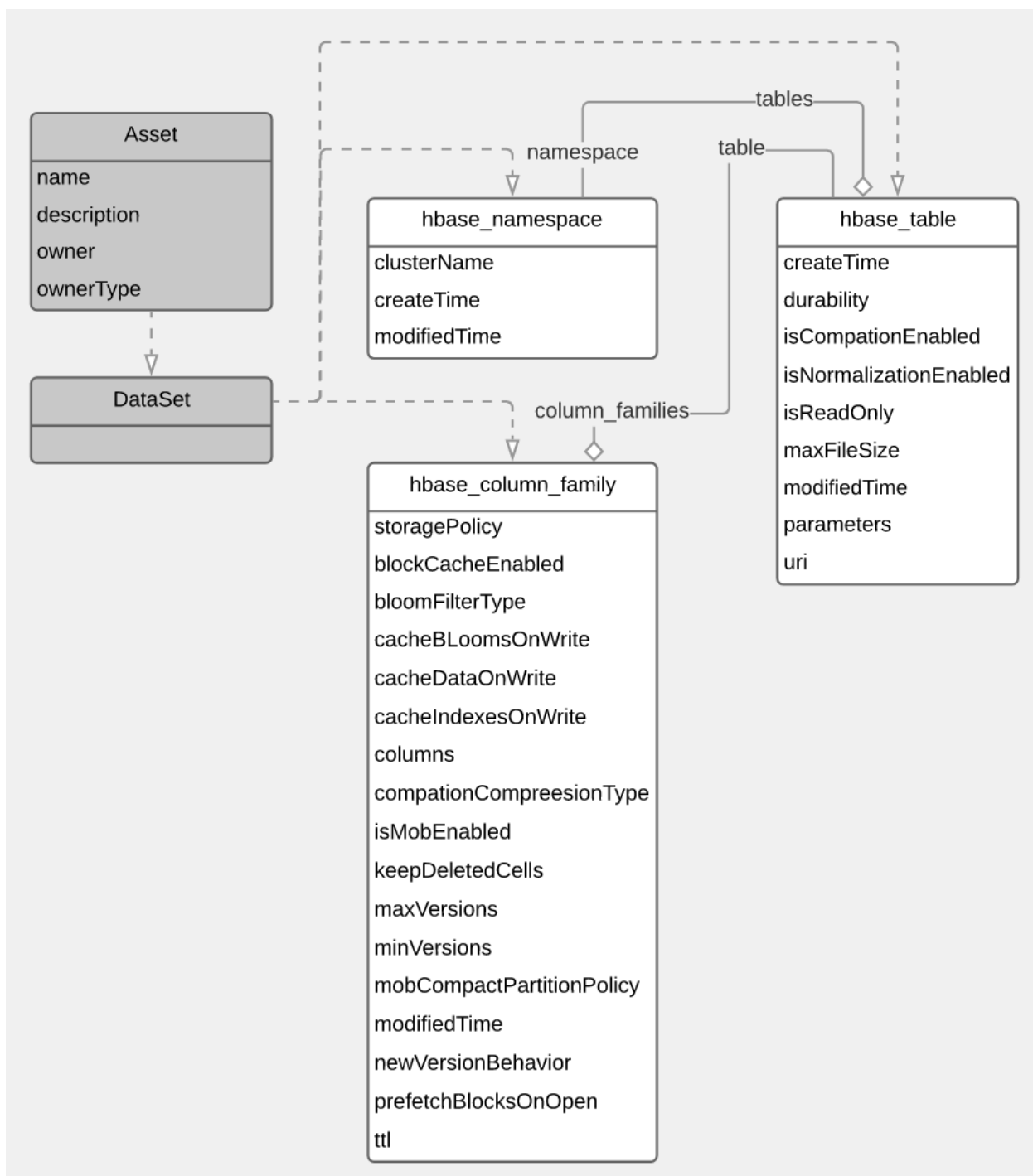
- Truncate table
- Put (cell value]
- Disable/enable table

HBase entities created in Atlas

Each HBase data set entity in Atlas includes detailed metadata collected from HBase.

The following diagrams show a summary of the entities created in Atlas for Hive operations and assets. The supertypes that contribute attributes to the entity types are shaded.

Figure 3: Atlas Entity Types for HBase Data Sets



The metadata collected for each entity type is as follows:

HBase Namespace

| Identifier | Example content |
|---------------|--|
| typeName | hbase_namespace |
| guid | System generated ID. This value is used to identify the entity in the Atlas Dashboard URL. |
| qualifiedName | <name>@<clustername> |
| name | Namespace name as reported from HBase. |

| Identifier | Example content |
|----------------------|---|
| description | String description metadata from HBase. |
| modifiedTime | Time from HBase indicating a change to the namespace. Formatted as in this example: "Wed Apr 17 2019 18:32:14 GMT-0700 (Pacific Daylight Time)" |
| owner | Owner as reported from HBase. |
| parameters | Reserved for future use. |
| replicatedFrom | Reserved for future use. |
| replicatedTo | Reserved for future use. |
| Relationship: tables | One namespace to many tables. hbase_table_namespace |

HBase Table

| Identifier | Example content |
|-------------------------------|---|
| typeName | hbase_table |
| guid | System generated ID. This value is used to identify the entity in the Atlas Dashboard URL. |
| qualifiedName | <namespace>:<tablename>@<clustername> |
| name | Table name as reported from HBase. |
| description | String description metadata from HBase. |
| modifiedTime | Time from HBase indicating a change to the table. Formatted as in this example: "Wed Apr 17 2019 18:32:14 GMT-0700 (Pacific Daylight Time)" |
| owner | Owner as reported from HBase. |
| parameters | Reserved for future use. |
| replicatedFrom | Reserved for future use. |
| replicatedTo | Reserved for future use. |
| durability | Storage property as reported from HBase. Values include true or false. |
| isCompactionEnabled | Storage property as reported from HBase. Values include true or false. |
| isNormalizationEnabled | Storage property as reported from HBase. Values include true or false. |
| isReadOnly | |
| maxFileSize | Storage property as reported from HBase. -1 indicates that no maximum was set. |
| uri | Table name. |
| Relationship: namespace | One namespace to many tables. hbase_table_namespace |
| Relationship: column families | Column families associated with this table. hbase_table_column_families |

HBase Column Family

| Identifier | Example content |
|---------------|---|
| typeName | hbase_column_family |
| guid | System generated ID. This value is used to identify the entity in the Atlas Dashboard URL. |
| qualifiedName | <namespace>:<tablename>.<columnfamily>@<clustername> |
| name | Column family name as reported from HBase. |
| description | String description metadata from HBase. |
| modifiedTime | Time from HBase indicating a change to the column family. Formatted as in this example: "Wed Apr 17 2019 18:32:14 GMT-0700 (Pacific Daylight Time)" |

| Identifier | Example content |
|---------------------------|--|
| owner | Owner as reported from HBase. |
| StoragePolicy | Value for the storagePolicy property for the column family. Values include N/A, ALL_SSD, ONE_SSD, HOT, WARM, COLD. |
| blockCacheEnabled | Storage property as reported from HBase. Values include true or false. |
| bloomFilterType | Value for the BLOOM_FILTER_TYPE property for the column family. Values include NONE, ROW, or ROWCOL. |
| cacheBloomsOnWrite | Boolean value for the CACHE_BLOOMS_ON_WRITE property for the column family. |
| cacheDataOnWrite | Boolean value for the CACHE_DATA_ON_WRITE property for the column family. |
| cacheIndexesOnWrite | Boolean value for the CACHE_INDEX_ON_WRITE property for the column family. |
| columns | List of columns included in the column family. |
| compactionCompressionType | Storage property as reported from HBase. |
| compressionType | Value for the COMPRESSION property for the column family. Values include NONE, SNAPPY, LZ0, LZ4, GZ. |
| createTime | Time from HBase indicating when the column family was created. Formatted as in this example: “Wed Apr 17 2019 18:32:14 GMT-0700 (Pacific Daylight Time)” |
| dataBlockEncoding | The DATA_BLOCK_ENCODING property for the column family. Values include NONE, PREFIX, DIFF, FAST_DIFF, ROW_INDEX_V1. |
| encryptionType | Column family encryption property. Values include “N/A”, and AES. |
| evictBlocksOnClose | Boolean value for the EVICT_BLOCKS_ON_CLOSE property for the column family. |
| inMemoryCompactionPolicy | In memory compaction behavior (IN_MEMORY_COMPACTION) set for the column family. Values include NONE, BASIC, EAGER, ADAPTIVE, or “N/A”. |
| isMobEnabled | Boolean value for Medium Object (MOB) properties for the column family (IS_MOB). |
| keepDeletedCells | Boolean value for the KEEP_DELETED_CELLS property of the column family. |
| maxVersions | The maximum number of row versions this column family is configured to store. |
| minVersions | The minimum number of row versions this column family is configured to store. |
| mobCompactPartitionPolicy | The MOB_COMPACT_PARTITION_POLICY for this column family. Values include DAILY, WEEKLY, MONTHLY. |
| modifiedTime | Time from HBase indicating a change to the column family. Formatted as in this example: “Wed Apr 17 2019 18:32:14 GMT-0700 (Pacific Daylight Time)” |
| newVersionBehavior | Boolean value for the NEW_VERSION_BEHAVIOR property for this column family. |
| prefetchBlocksOnOpen | Boolean value for PREFETCH_BLOCKS_ON_OPEN property of the column family. |
| replicatedFrom | Not used. |
| replicatedTo | Not used. |
| table | The table that the column family corresponds to. Also modeled as a relationship. |
| ttl | Time to live (TTL) length in seconds. The TTL time encoded in the HBase for the row is specified in UTC. |
| Relationship: columns | Not used. |
| Relationship: table | One table to many column families. hbase_table_column_families |

Hbase lineage

Atlas collects lineage information for HBase data assets when HBase tables are referenced in HiveServer or Impala operations.

The Atlas lineage graph shows the input and output processes that the current entity participated in, specifically those relationships modeled as “inputToProcesses” and “outputFromProcesses.” Entities are included if they were inputs to processes that lead to the current entity or they are output from processes for which the current entity was an input.

No lineage metadata is collected directly from HBase.

Related Information

[Viewing lineage](#)

HBase audit entries

Atlas lists changes to metadata entities in the Audit tab in the Dashboard.

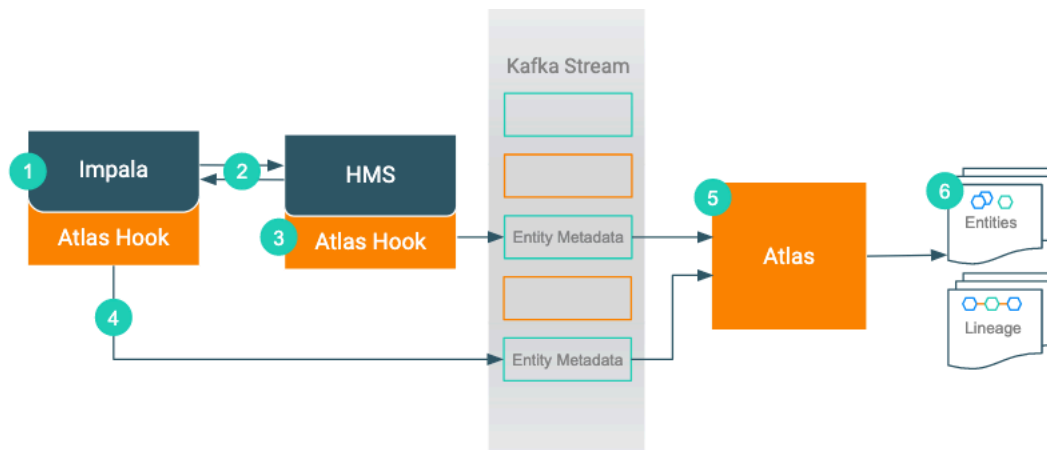
Atlas tracks the lifecycle of each HBase entity, including its creation, update, and deletion. User access and actions that affect the data content of the source asset are not included in the audit.

Impala metadata collection

Atlas can collect metadata for queries from Impala. It collects metadata for affected data assets from Hive Metastore (HMS).

An Atlas hook runs in each Impalad instance. This hook sends metadata to Atlas for Impala operations, which are represented by process and process execution entities in Atlas.

In addition, an Atlas hook runs in Hive Metastore (HMS). Before sending metadata to Atlas, Impala synchronizes its metadata with HMS. This synchronization makes sure that Impala uses the same names and IDs as HMS.



1. When an action occurs in the Impala instance...
2. Impala updates HMS with information about the assets affected by the action.
3. The Atlas hook for HMS collects information for the changed and new assets and forms it into metadata entities. It publishes the metadata to a Kafka topic.
4. The Atlas hook for the Impala instance collects information for the action and forms it into metadata entities. It publishes the metadata to a Kafka topic.
5. Atlas reads the messages from the topic and determines what information will create new entities and what information updates existing entities. Atlas is able to determine the correct entities regardless of the order in which Atlas receives messages from the Kafka topic.
6. Atlas creates the appropriate entities and determines lineage from existing entities to the new entities.

Impala actions that produce Atlas entities

Impala operations that create, update, or delete Hive metadata will affect Atlas entities; operations that only affect data do not show up in Atlas.

The following table lists the Impala actions that produce or update metadata in Atlas.

| This Action in Impala... | ...Produces metadata for these Atlas entities | ...Triggers HMS to produce metadata for these Atlas entities | ...Produces metadata for these Atlas relationships |
|--------------------------------------|---|---|---|
| CREATETABLE_AS_SELECT | impala_process, impala_process_execution, impala_column_lineage, hive_db hive_table_ddl | hive_table, hive_column(s), hive_storedesc, hive_db hive_table_ddl | hive_table_db, hive_table_columns, hive_table_partitionkeys, hive_table_storedesc, hive_process_process_execution, hive_process_columnlineage, hive_table_ddl_queries, hive_db_ddl_queries |
| CREATEVIEW | impala_process, impala_process_execution, impala_column_lineage, hive_table_ddl | hive_table, hive_column(s), hive_db | hive_table_db, hive_table_columns, hive_table_partitionkeys, hive_process_process_execution, hive_process_columnlineage, hive_table_ddl_queries |
| ALTERVIEW_AS_SELECT | impala_process, impala_process_execution, impala_column_lineage, hive_table_ddl | Updates to: hive_table, hive_column(s) | hive_process_process_execution, hive_process_columnlineage, hive_table_ddl_queries |
| INSERT INTO, INSERT, OVERWRITE | impala_process, impala_process_execution | If not already in Atlas, HMS sends metadata for data assets indicated in the query: hive_table, hive_column(s), hive_storedesc, hive_db | hive_process_process_execution |

Notable actions in Impala that do NOT produce process or process execution entities in Atlas, meaning that no lineage is produced for these operations:

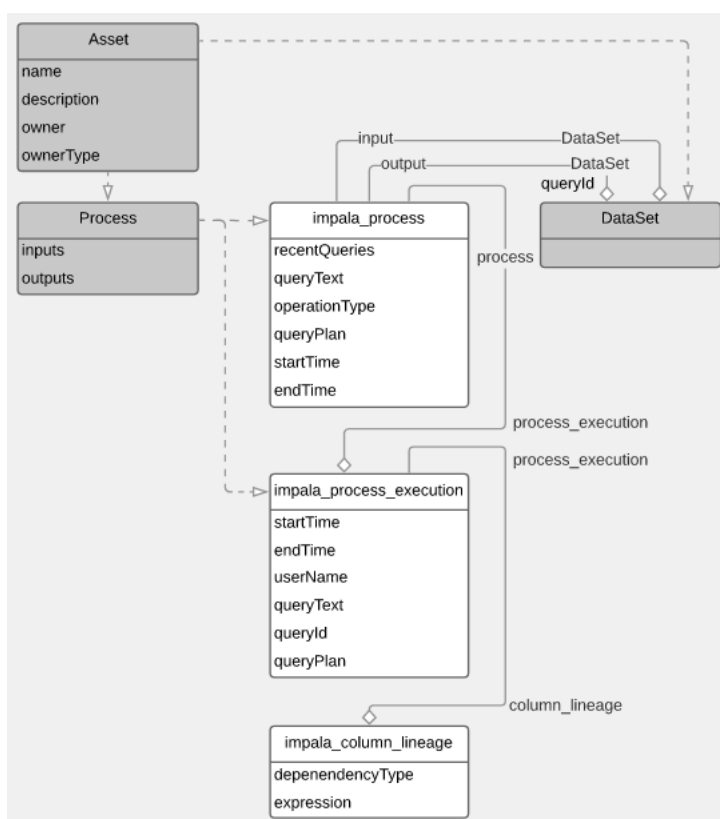
- LOAD DATA INPATH
- CREATE TABLE (table metadata produced by HMS)
- ALTER VIEW (table metadata produced by HMS)
- SELECT or other queries that don't produce output

Impala entities created in Atlas

Each Impala entity in Atlas includes detailed metadata for Impala queries.

The following diagrams show a summary of the entities created in Atlas for Impala operations. The supertypes that contribute attributes to the entity types are shaded.

Figure 4: Atlas Entity Types for Impala Operations



The metadata collected for each entity type is as follows:

Impala Process

| Identifier | Example content |
|---------------------------------|---|
| typeName | impala_process |
| guid | System generated ID. This value is used to identify the entity in the Atlas Dashboard URL. |
| qualifiedName | <database>.<target table>@<clustername>:<generated ID> The generated ID is distinct from the GUID. |
| name | Text of the query. |
| inputs | List of the input tables or views, including each entity's type name and the qualified name. |
| outputs | List of the output objects, including each entity's type name and the qualified name. |
| recentQueries | Last query executed (duplicated in process_execution). |
| operationType | One of the operations that triggers metadata collection. |
| queryPlan | Reserved for future use. |
| startTime | Most recent query start time. |
| endTime | Most recent query end time. |
| Relationship: Process Execution | One process to one or more process executions. impala_process_process_execution |
| Relationship: Column Lineage | One process to one or more column lineages. impala_process_column_lineage |

Impala Process Execution

| Identifier | Example Content |
|-----------------------|---|
| typeName | impala_process_execution |
| guid | System generated ID. This value is used to identify the entity in the Atlas Dashboard URL. |
| qualifiedName | <database>.<target table>@<clustername>:<ID from process qualified name>:<ID from the process execution name>:<generated ID for this process execution> |
| name | Text of the query with a system-generated ID added to the end. |
| queryText | Text of the query. |
| queryPlan | Reserved for future use. |
| queryId | impala_<date as yyyyymmddhhmmss>_<generated id> |
| startTime | Query start time. |
| endTime | Query end time. |
| userName | The user who ran the query. |
| Relationship: Process | One process to one or more process executions. impala_process_process_execution |

Impala Column Lineage

| Identifier | Example Content |
|-----------------------------------|--|
| typeName | impala_column_lineage |
| dependencyType | The type of relationship between the input and output columns; one of SIMPLE, EXPRESSION, or SCRIPT. |
| name | <database>.<table>@<clustername>:<generated ID>:<output_column> |
| inputs | List of 0 or more hive_column entities that contributed to the output columns. This is a legacy model component: the more current model uses a relationship attribute. |
| outputs | This is a legacy model component: the more current model uses a relationship attribute. |
| qualifiedName | Same as name. |
| query | Name of the impala_process entity that produced this lineage. This is a legacy model component: the more current model uses a relationship attribute. |
| Relationship: Process | Name of the impala_process entity that produced this lineage. impala_process_column_lineage |
| Relationship: inputToProcesses | List of 0 or more hive_column entities that contributed to the output columns. |
| Relationship: outputFromProcesses | List of 0 or more hive_column entities that were produced in the process. |

Impala lineage

You can use the Atlas lineage graph to understand the source and impact of data and changes to data over time and across all your data.

Atlas collects metadata from Impala to represent the lineage among data assets. The Atlas lineage graph shows the input and output processes that the current entity participated in. Entities are included if they were inputs to processes that lead to the current entity or they are output from processes for which the current entity was an input. Impala processes follow this pattern.

Note that lineage is not updated between a table and views that the table is a part of when an Impala ALTER TABLE operation runs on the table.

Related Information

[Viewing lineage](#)

Impala audit entries

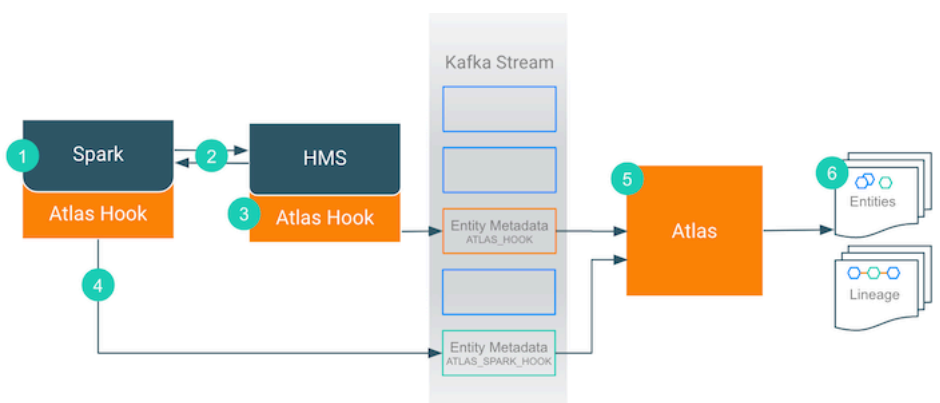
Atlas lists changes to metadata entities in the Audit tab in the Dashboard.

Atlas tracks the lifecycle of each Impala entity, including its creation, update, and deletion.

Spark metadata collection

Atlas can collect metadata from Spark, including queries on Hive tables. The Spark Atlas Connector (SAC) is available as of Spark 2.4 and Atlas 2.1.

An Atlas hook runs in each Spark instance. This hook sends metadata to Atlas for Spark operations. Operations are represented by process entities in Atlas. Hive databases, tables, views, and columns that are referenced in the Spark operations are also represented in Atlas, but the metadata for these entities is collected from HMS. When a Spark operation involves files, the metadata for the file system and files are represented in Atlas as file system paths.



1. When an action occurs in the Spark instance...
2. It updates HMS with information about the assets affected by the action.
3. The Atlas hook corresponding to HMS collects information for the changed and new assets and forms it into metadata entities. It publishes the metadata to the Kafka topic named ATLAS_HOOK.
4. The Atlas hook corresponding to the Spark instance collects information for the action and forms it into metadata entities. It publishes the metadata to a different Kafka topic named ATLAS_SPARK_HOOK.
5. Atlas reads the messages from the topics and determines what information will create new entities and what information updates existing entities. Atlas is able to determine the correct entities regardless of the order in which Atlas receives messages from the Kafka topics.
6. Atlas creates the appropriate entities and the relationships among them and determines lineage from existing entities to the new entities.

Spark actions that produce Atlas entities

Operations that create Spark process entities and create, update, or delete the data assets affected by those operations will affect Atlas entities; operations that only affect data do not show up in Atlas.

The following table lists the Spark actions that produce or update metadata in Atlas.

| This Action in Spark... | ...Produces metadata for these Atlas entities |
|---|--|
| CREATE TABLE USING CREATE TABLE AS SELECT, CREATE TABLE USING ... AS SELECT | spark_process, hive_table, hive_column, hive_storagedesc |

| This Action in Spark... | ...Produces metadata for these Atlas entities |
|---|--|
| CREATE VIEW AS SELECT, | spark_process, hive_table, hive_column, hive_storagedesc |
| INSERT INTO (SELECT), LOAD DATA [LOCAL] INPATH | spark_process |

Notable actions in Spark that do NOT produce process entities in Atlas, meaning that no lineage is produced for these operations:

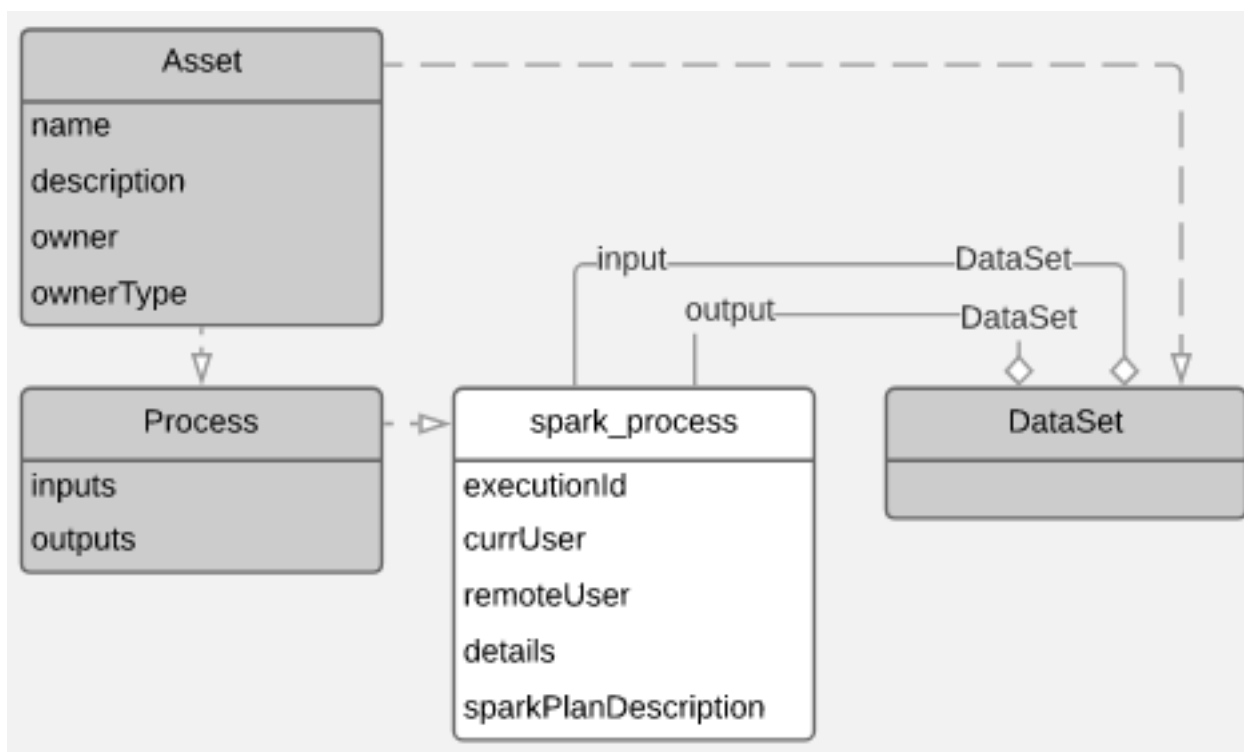
- LOAD DATA INPATH (when not coming from a local file source)
- CREATE TABLE (hive_table metadata produced by HMS)
- ALTER VIEW (hive_table metadata produced by HMS)
- SELECT or other queries that don't change table metadata

Spark entities created in Apache Atlas

Each Spark entity in Atlas includes detailed metadata collected from Spark.

The following diagrams show a summary of the entities created in Atlas for Spark operations. The data assets that Spark operations act upon are collected through HMS. The supertypes that contribute attributes to the entity types are shaded.

Figure 5: Atlas Entity Types for Spark Data Sets



The metadata collected for each entity type is as follows:

Spark Process

| Identifier | Example content |
|------------|-----------------|
| typeName | spark_process |

| Identifier | Example content |
|----------------------|---|
| guid | System generated ID. This value is used to identify the entity in the Atlas Dashboard URL. |
| qualifiedName | <generated ID> The generated ID is distinct from the GUID. |
| name | process_<generated ID> |
| description | Metadata from Spark. |
| owner | Metadata from Spark. |
| ownerType | Metadata from Spark. |
| inputs | List of the input tables or views, including each entity's type name and the qualified name. |
| outputs | List of the output objects, including each entity's type name and the qualified name. |
| executionId | Metadata from Spark. |
| currUser | Metadata from Spark. In a Kerberized environment, this value contains the principal name. |
| remoteUser | Metadata from Spark. In a Kerberized environment, this value contains the principal name. |
| executionTime | Metadata from Spark. |
| details | Query plan text, including parsed logical plan, analyzed logical plan, optimized logical plan, and physical plan. |
| sparkPlanDescription | Physical plan text. |

Spark lineage

Atlas collects metadata from Spark to represent the lineage among data assets.

The Atlas lineage graph shows the input and output processes that the current entity participated in, specifically those relationships modeled as “inputToProcesses” and “outputFromProcesses.” Entities are included if they were inputs to processes that lead to the current entity or they are output from processes for which the current entity was an input. Spark processes follow this pattern.

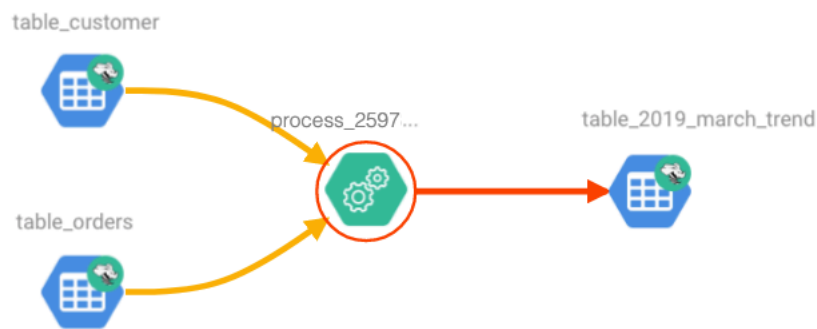
process_2597 application_1568 (spark_process)

Classifications: +

Term: +

Properties Lineage Relationships Classifications Audits

○ Current Entity → Lineage → Impact



Related Information

[Viewing lineage](#)

Spark relationships

Atlas shows the entities related to this entity in the Relationships tab in the Dashboard.

The Relationship tab shows the relationships that exist for an entity. Use this view to navigate among related entities.

process_2597 application_1568

Classifications: +

Term: +

Properties Lineage Relationships Classifications Audits

Graph Table

| Key | Value | <input type="checkbox"/> Show Empty Values |
|-------------|---|--|
| inputs (2) | table_customers table_orders | |
| outputs (1) | table_2019_march_trends | |

Spark audit entries

Atlas lists changes to metadata entities in the Audit tab in the Dashboard.

Atlas tracks the lifecycle of each Spark entity, including its creation, update, and deletion. User access and actions that affect the data content of the source asset are not included in the audit.

Spark troubleshooting

What do you do if you don't see Atlas metadata from Spark?

Spark runs an Atlas "hook" or plugin on every host where it runs. To troubleshoot problems, consider the following methods for narrowing down where the problem is:

- Are you missing all metadata?

Make sure that all the services supporting Atlas are configured and running. For CDP, the configuration is done for you; look in Cloudera Manager to see that Kafka, Solr, and Atlas services are running in the Data Lake.

- Are you missing all Spark process metadata?

By default, Spark operations are configured to send metadata to Atlas. To check that these settings have not been rolled back, look at the Spark configuration page in Cloudera Manager to ensure that Spark is configured to send metadata to Atlas. Assuming this configuration is enabled, you can next check the Kafka topic queue to make sure that metadata messages are being produced in Spark and making it to the Kafka topic.

- Missing only some Spark metadata?

Because each instance of Spark collects metadata independently of other instances, it is possible that one instance failed to send metadata to Atlas. To determine if this is the problem, check the Kafka topic queue to see if one of the Spark hosts is not sending metadata.