Cloudera Runtime 7.3.1

Apache Atlas Reference

Date published: 2020-07-28
Date modified: 2025-03-18

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2025. 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....5

Apache Atlas Statistics reference....8

Apache Atlas metadata attributes....11

Dynamic handling of failure in updating index....14
Configurations used for index recovery....14
List of REST API for dynamic index recovery....15

Defining Apache Atlas enumerations....16

Purging deleted entities....17
Auditing purged entities....18
PUT /admin/purge/ API....19
POST /admin/audits/ API....22

Apache Atlas technical metadata migration reference....24
System metadata migration....24
HDFS entity metadata migration....25
Hive entity metadata migration....26
Impala entity metadata migration....30
Spark entity metadata migration....32
AWS S3 entity metadata migration....33

NiFi metadata collection....35
How Lineage strategy works....36
Understanding the data that flow into Atlas....38
NiFi lineage....38
Atlas NiFi relationships....39
Atlas NiFi audit entries....41
How the reporting task runs in a NiFi cluster....41
Analysing event analysis....42
Limitations of Atlas-NiFi integration....43

HiveServer metadata collection....44
HiveServer actions that produce Atlas entities....44
HiveServer entities created in Atlas....45
HiveServer relationships....50
HiveServer lineage....51
HiveServer audit entries....52

HBase metadata collection....53
HBase actions that produce Atlas entities....54
HBase entities created in Atlas....54
Changing the column family compression type....58
Hbase lineage....58
HBase audit entries....59

Schema Registry metadata collection....59
Configuring Atlas and Schema Registry....61
Schema Registry actions that produce Atlas entities....62
Schema replationships....62
Schema Registry audit entries....64
Troubleshooting Schema Registry....64

Impala metadata collection....64
Impala actions that produce Atlas entities....65
Impala entities created in Atlas....66
Impala lineage....68
Impala audit entries....69

Kafka metadata collection....69
Kafka actions that produce Atlas entities....70
Kafka relationships....70
Kafka lineage....71
Kafka audit entries....71

Spark metadata collection....72
Spark actions that produce Atlas entities....73
Spark entities created in Apache Atlas....73
Spark lineage....76
Spark relationships....76
Spark audit entries....77
Spark Connector configuration in Apache Atlas....77
Spark troubleshooting....78

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 that 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, qualifie
dName
```

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 FullNa
me
```

Searches with system attributes

In the attribute filter lists, system attributes appear with normal text names. When you use them in advanced searches, use the corresponding field name, which is prefixed with two underscores.

System Attribute	Description	Identifier in Advanced Search
Type	The Atlas entity type.	__typeName
Status	The entity status in Atlas: this field indicates if a data asset has been deleted; Atlas maintains the entity information after the asset no longer exists on the cluster.	__state
Created By User	The Atlas user who created this entity. Typically this is the Atlas system user. If an entity was created by an API call or created manually by users, the active user account would be included in this attribute.	__createdBy
Last Modified User	The Atlas user who last updated the entity, whether through Atlas metadata collection from a cluster service, an Atlas API, or a change through the Atlas UI.	__modifiedBy
Created timestamp	The date Atlas created the entity. Note that this field is different from the technical attribute for the creation date of the original data asset or operation.	__timestamp
Last Modified timestamp	The date when an entity was last updated in Atlas. Note that this field is different from the technical attribute for the last modification date of the actual data asset or operation on the cluster.	__modificationTimestamp
GUID	A unique identifier generated by Atlas. This is the 32-digit code found in the browser URL for an entity.	__guid
Labels	Label metadata added to an entity.	__labels
User-Defined Properties	Key-value pair metadata added to an entity.	__customAttributes
Classifications	Classifications added to an entity.	__classificationNames
Propagated Classifications	Classifications added to entities downstream from an entity where the classification was added by a user.	__propagatedClassificationNames
—	A concatenated string of classification names and attributes for an entity. This attribute is not available through the Atlas UI.	__classificationsText
IsIncomplete	A system indicator that entities were created because they were referenced in the metadata collected by a service other than the source type associated with the entity type. An entity is typically marked "isIncomplete" when Atlas receives metadata out of order from when the events occurred. If IsIncomplete entities remain “incomplete” for a long time, it may indicate that the original messages for entity metadata have not arrived.	__isIncomplete

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 the following query:

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

Related Information

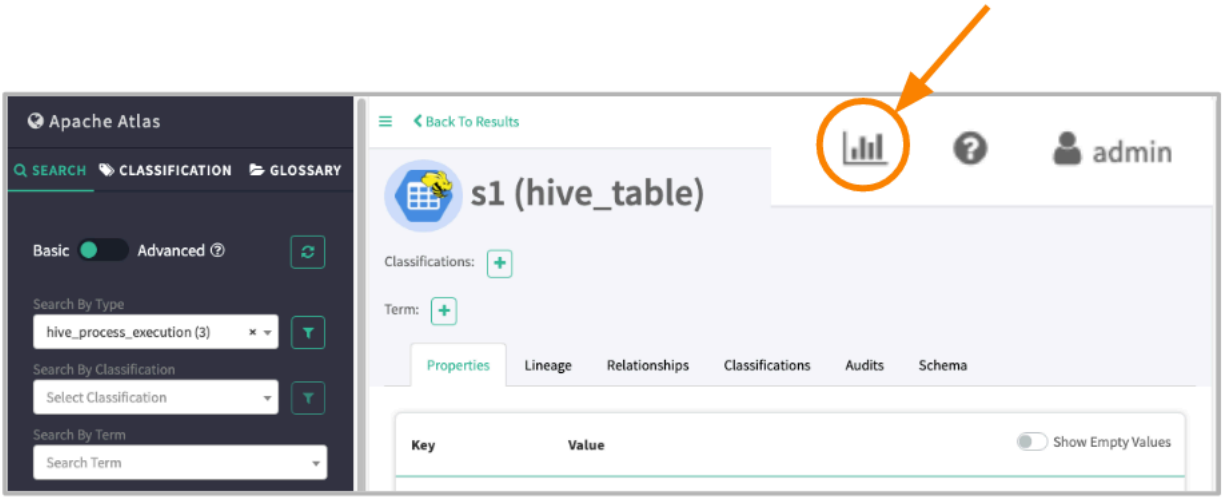
Apache Atlas metadata attributes

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.

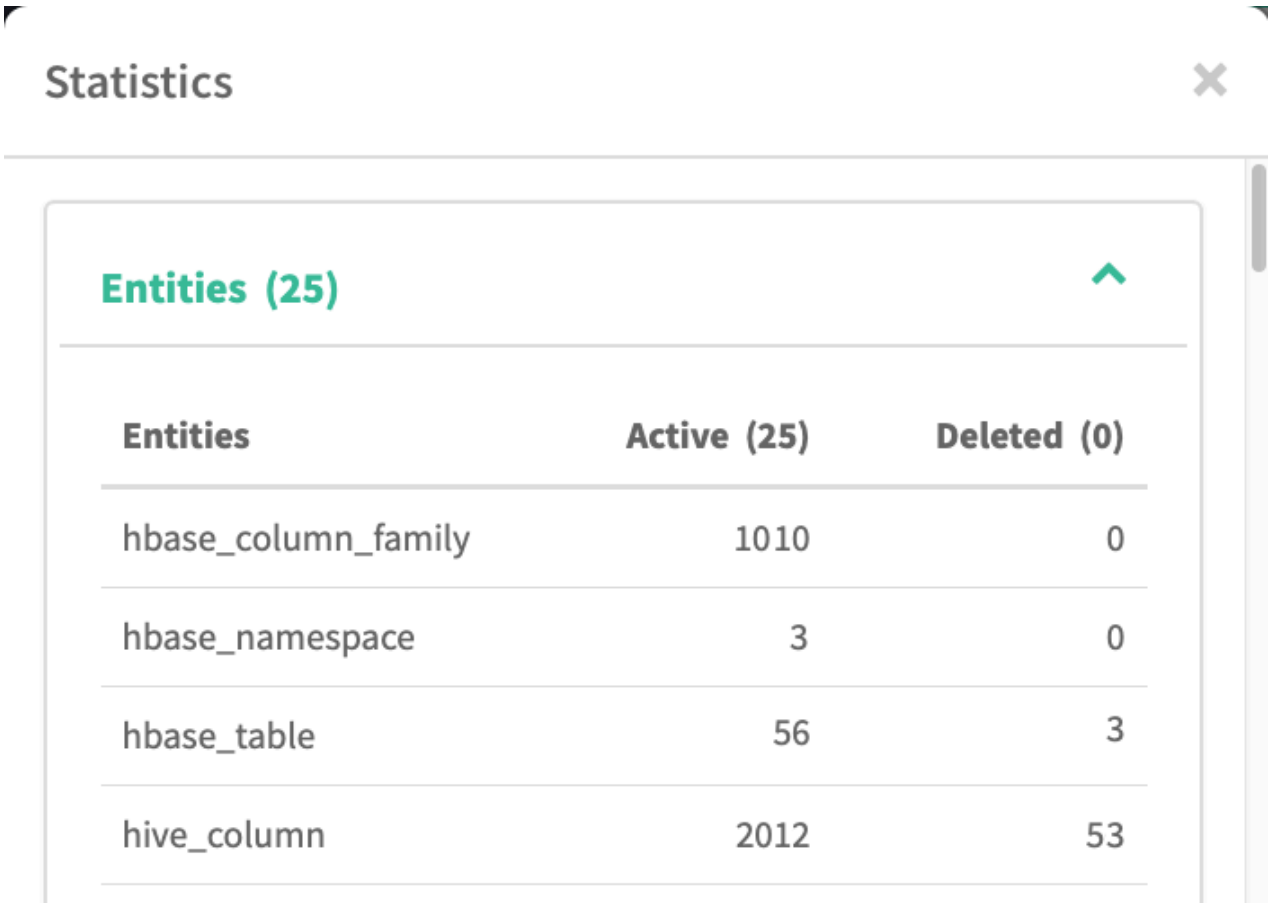
To view statistics, click the graph button in the top right corner:



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.

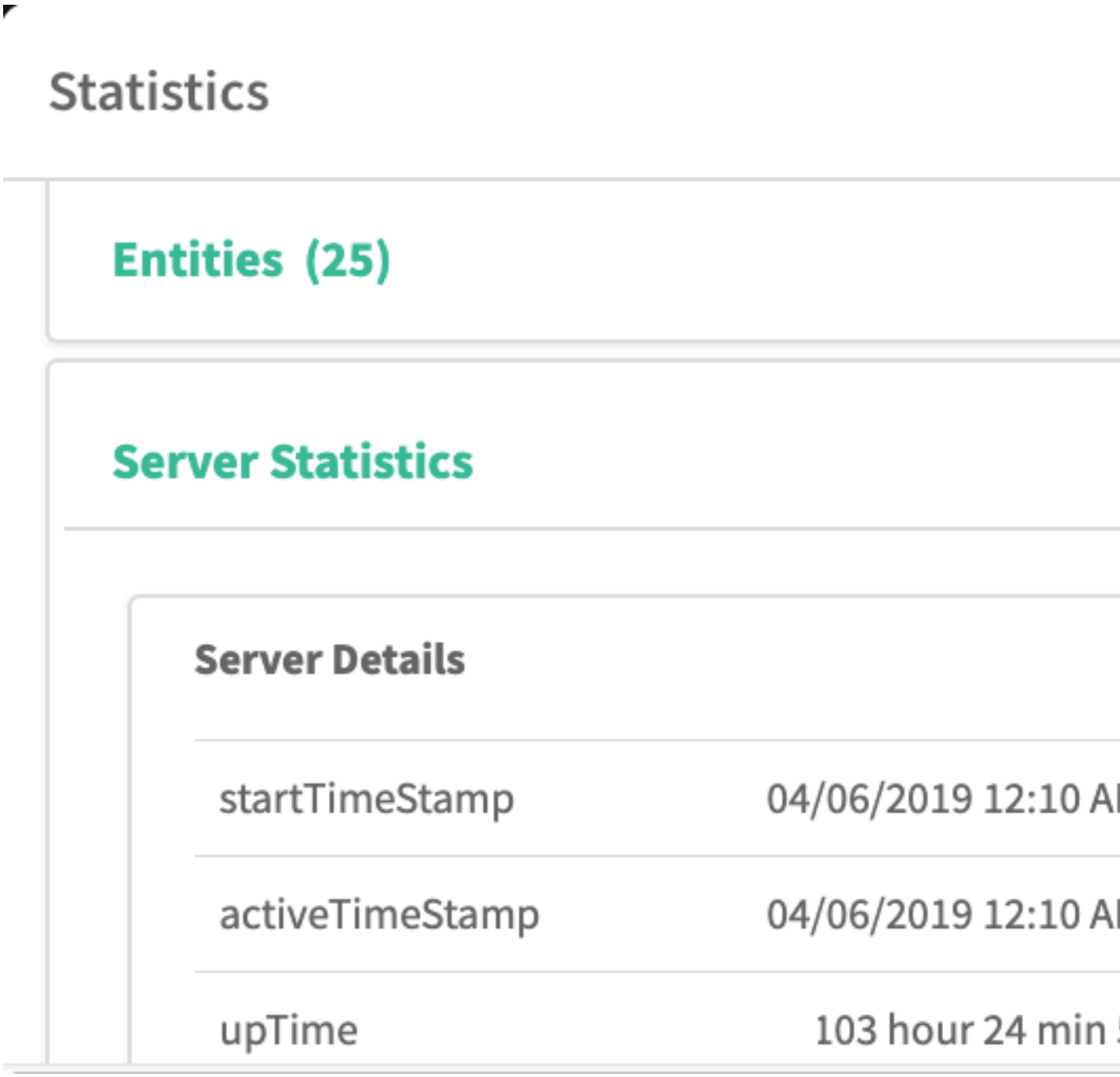


Classification Statistics

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.



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 where 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.

Apache Atlas metadata attributes

Attributes are the key-value pairs that hold metadata details for entities and classifications.

Different types of attributes are populated with values differently.
Technical Attributes

These attributes are the entity fields that contain technical metadata defined in entity models. For the built-in entity types, Atlas collects this information from services on the cluster. These attributes are read-only in the UI but can be updated using the Atlas API. All entity types share basic metadata such as names and qualified names; however, the rest of the technical metadata is specific to the entity type.

System Attributes

These attributes are populated by Atlas when it creates an entity instance. They include:

System Attribute	Description	Identifier in Advanced Search
Type	The Atlas entity type.	__typeName
Status	The entity status in Atlas: this field indicates if a data asset has been deleted; Atlas maintains the entity information after the asset no longer exists on the cluster.	__state
Created By User	The Atlas user who created this entity. Typically this is the Atlas system user. If an entity was created by an API call or created manually by users, the active user account would be included in this attribute.	__createdBy
Last Modified User	The Atlas user who last updated the entity, whether through Atlas metadata collection from a cluster service, an Atlas API, or a change through the Atlas UI.	__modifiedBy
Created timestamp	The date Atlas created the entity. Note that this field is different from the technical attribute for the creation date of the original data asset or operation.	__timestamp
Last Modified timestamp	The date when an entity was last updated in Atlas. Note that this field is different from the technical attribute for the last modification date of the actual data asset or operation on the cluster.	__modificationTimestamp
GUID	A unique identifier generated by Atlas. This is the 32-digit code found in the browser URL for an entity.	__guid
Labels	Label metadata added to an entity.	__labels
User-Defined Properties	Key-value pair metadata added to an entity.	__customAttributes
Classifications	Classifications added to an entity.	__classificationNames
Propagated Classifications	Classifications added to entities downstream from an entity where the classification was added by a user.	__propagatedClassificationNames
—	A concatenated string of classification names and attributes for an entity. This attribute is not available through the Atlas UI.	__classificationsText
IsIncomplete	A system indicator that entities were created because they were referenced in the metadata collected by a service other than the source type associated with the entity type. An entity is typically marked "isIncomplete" when Atlas receives metadata out of order from when the events occurred. If IsIncomplete entities remain “incomplete” for a long time, it may indicate that the original messages for entity metadata have not arrived.	__isIncomplete

When defining new models, you can take advantage of the isAppendOnPartialUpdate option in attribute definitions. This option allows array or map type attribute values to be updated by appending rather than replacing the entire set. For example, to represent a list of key-value pairs that can be augmented over time, you might define an attribute metadata as a map with the option isAppendOnPartialUpdate set to true:

```
{
   "name": "metadata",
   "typeName": "map<string,string>",
   "isOptional": true,
   "cardinality": "SINGLE",
   "valuesMinCount": 0,
   "valuesMaxCount": 1,
   "isUnique": false,
   "isIndexable": false,
   "includeInNotification": false,
```

```
   "description": "Contains key-value pairs that provide metadata
",
   "searchWeight": -1,
   "options": {
   "isAppendOnPartialUpdate": "true"
}
```

Classifications, labels, and user-defined properties are included as system attributes in the context of search. They are modeled as entity attributes so that when you access an entity (through the UI or API), you get all these entity-specific metadata.

Business Metadata Attributes

These attributes are populated in the Atlas UI or through API calls. They provide a way to extend the metadata stored for entity instances. You can define Business Metadata attributes to apply to a specific entity type or to many entity types. Administrators can control the users or groups who can set values for these attributes by creating a Ranger policy against the Business Metadata collection that contains the attribute.

Classification Attributes

These attributes are populated in the Atlas UI or through API calls. They provide a way to enrich the worth of a classification for searching, for access policies in Ranger, and for organizing cluster data assets.

Classifications can also be assigned to entities through lineage: if the classification is defined to allow lineage propagation, a classification assigned to an entity is also assigned to all entities that have output relationships to the classified entity. The propagation applies to all further generations of the lineage. Note that Atlas distinguishes between classifications that were specifically assigned to an entity and classifications that were assigned through lineage propagation.

User-defined Properties

These attributes are populated in the Atlas UI or through API calls. They allow users to add metadata in the form of key-value pairs to any entity instance. Values are limited to strings. Both key and value are included in searches. They are not centrally managed like classifications or Business Metadata attributes. They are not accessible through Ranger for specifying access policies.

Defining attributes

Attribute names can include letters, numbers, underscores, and hyphens; they must start with a letter or number. All attributes can have values one of the following Java data types:

- string
- Boolean
- byte
- short
- int
- float
- double
- long
- date
- enumeration

Where enumeration type values are strings from pre-defined enumeration defined using the Atlas API.

Note: The Atlas Free-text search only works with attributes with string values.

When you define an attribute, you an indicate that the value can include more than one entry. Atlas records multiple values in a comma-separated list. Thus, when searching on attributes with multiple values, users should use the logical operator "Contains" rather than "=" so the search matches on a single value rather than the whole list.

Related Information

Atlas Business Metadata overview
Working with Atlas Classifications and Labels
Configuring Atlas Authorization using Ranger
Defining Apache Atlas enumerations

Dynamic handling of failure in updating index

The JanusGraph database transaction might fail in certain scenarios and this failure can be handled dynamically using a specific configuration.

There are certain scenarios in which JanusGraph transactions might fail, including, while indexing into Solr. When you create HIVE entities, the first level of storage includes persisting into HBase and the corresponding indexes are logged in Solr.

The index updation can be partially successful when data gets stored in HBase but Solr indexing fails. There could be inconsistencies with logged indexes for the failed transactions in Solr. These scenarios might lead to a mismatch between the basic and advanced search results in Atlas. Re-indexing the data is an option which can reindex the entire data; however, it is time consuming.

To have a better system resilience and evolution of data tracking, a transaction log processor option named write-ahead is implemented. When enabled, JanusGraph maintains all the transaction log information which can be used to recover indices in the event of failures as described earlier. The log data information consumes more storage space but having a robust system coupled with maintaining data movement overrides the need to have additional storage systems.

The write-ahead configuration inspects the Solr health and registers the time in the event of any issues with Solr. Once Solr is up and running, the indices can be recovered from the previously noted time.

About JanusGraph

Learn what JanusGraph is and how Apache Atlas uses the JanusGraph database to support search index capabilities.

JanusGraph is a scalable graph database optimized for storing and querying graphs containing hundreds of billions of vertices and edges distributed across a multi-machine cluster.

Apache Atlas uses the JanusGraph database at the heart of its metadata repository. This graph is used to show the interconnected relationships between data sources; the data sets they host; the business meaning of the data elements within each data set; the classification of these elements in terms of quality, confidentiality, retention; who (people and processes) are using them and for which purposes.

JanusGraph uses a pluggable persistence store to save the metadata content and a search index for its search API. Apache Atlas takes advantage of this configurability to support a range of size, scalability, and performance requirements.

Configurations used for index recovery

To enable index recovery, you must configure some of the underlying entities.

Before you begin

Configuration properties that must be set:

- atlas.index.recovery.enable: To enable or disable index recovery.

Example: atlas.index.recovery.enable=true.

By default index recovery is enabled on Atlas startup. To disable index recovery, set the value to false. When no index recovery data is available by default, Atlas considers recovery start time based on TTL - (current time minus tx.log TTL). By default SOLR tx.log default TTL is configured as 10 days.

- atlas.graph.index.status.check.frequency: To configure the frequency of SOLR health check. It takes input as time in milliseconds. By default SOLR health checkup takes effect every 30 seconds.

Example: atlas.graph.index.status.check.frequency=5000.

- atlas.index.recovery.start.time: To start index recovery by custom recovery time as provided by the user. This property must be set to a timestamp in the yyyy-MM-dd'T'HH:mm:ssformat

If there are frequent SOLR restart operations or if SOLR is down, setting the start time configuration provides additional clarity.

Example: atlas.index.recovery.start.time=2023-03-24T09:17:23.656Z.

- write.ahead.log.ttl.in.hours: To configure time duration for which the write-ahead transaction log is available. This property value must be set in hours.

Note: The write-ahead logs live up to 10 days as per JanusGraph default settings.

Example: write.ahead.log.ttl.in.hours=36.

List of REST API for dynamic index recovery

You can use Atlas REST APIs that provide flexibility to start index recovery on demand, for index recovery. POST API and GET API are the supported APIs to handle index recovery.

POST API - Start on-demand index recovery

With this API, you can commence index recovery on-demand operation providing the start time for index recovery.

API: POST api/atlas/v2/indexrecovery/start?startTime=2023-04-10T06:17:23.656Z

This API recovers indices starting from earliest of 2023-04-10T06:17:23.656Z or from the transaction log available time.

Note: You can initiate the on-demand index recovery feature only as a user with administrator privileges.

GET API - Get index recovery timing details

Without REST API, if youwanted to retrieve the details about previous or upcoming index recovery, the only option available is to search through Atlas logs. But with this API, you can directly request for index recovery details.

API: GET api/atlas/v2/indexrecovery/

Response to GET API provides recovery time of previous and recent/upcoming index recovery and also provides the customTime which indicates on-demand index recovery started by you through the REST API.

```
"customTime": "2023-04-10T06:17:23.656Z", // Start time of index recovery requested through REST API
"startTime": "2023-04-18T04:04:24.411Z", // Start time of recent/upcoming Index recovery
"prevTime": "2023-04-17T03:54:54.084Z" // Start time of previous index recovery
```

Index recovery time

Learn about the Index recovery vertex properties and how Index recovery time persists in a single graph vertex.

Index recovery vertex has the following properties:

- Property-1: Key as “__idxRecovery_name” and with constant value as ”__solrIndexRecoveryInfo”.

- Property-2: Key as “__idxrecovery_startTime” and it holds the current recovery time.
- Property-3: Key as “__idxrecovery_prevTime” and it holds the previous recovery time.
- Property-4 Key as “__idxRecovery_customTime” and it holds the previous recovery time started through REST API.

  Whenever Apache Solr moves to an unhealthy state the upcoming recovery time is logged in Atlas application log as Index Recovery: Stopped! Recovery time: <timestamp>

  Every time Solr moves to an Active state from an unhealthy state it is logged in Atlas application log as Index Recovery: Started! Recovery time: <timestamp> and current recovery time is reset to null by assigning that time to previous recovery time.

Defining Apache Atlas enumerations

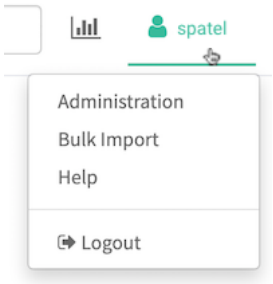
Atlas lets you define enumerations to use as attribute values.

Enumerations are a top-level objects in the Atlas data model. They can be used to standardize values available for users to select when assigning Business Metadata attributes to entities.

Users need administrator privileges to create or update enumerations.

To define enumerations:

1. Log in to Atlas.
2. To access the Atlas Administration features, choose Administration from the user menu in the top right of the Atlas window.



   Users need administrator privileges to access the Administration panel features.
3. Go to the Enumerations tab.
4. Enter the name of the new enumeration and select it in the list.

   Enumeration names must start with a letter and can include letters, numbers, spaces, and underscores.
5. Enter the enumeration values, separating each value by pressing Enter.

   Enumeration values are stored as strings and can include UTF-8 characters including spaces.
6. Click Update.

The enumeration is now available to be used as the type for Business Metadata attributes.

If the enumeration name or values don't meet the requirements, you'll see an error in the top right corner of the Atlas UI.

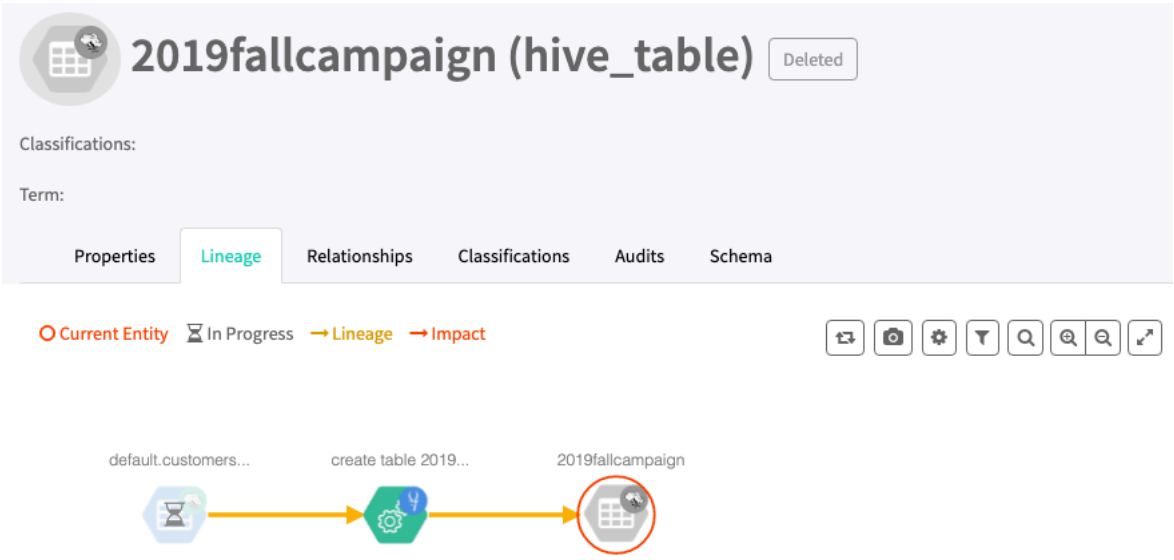
Related Information

Adding attributes to Business Metadata

Purging deleted entities

You can use Atlas REST API calls to remove entities from Atlas. Only entities that have been deleted in the source system and marked as deleted in Atlas can be purged.

When a data asset is deleted, such as after a DROP TABLE command in Hive, Atlas continues to retain the asset's entity, including metadata, lineage, and audit record. The status of the entity is set to "deleted"; deleted entities show up in search results when the checkbox to Show historical entities is checked. Deleted entities appear in lineage graph dimmed-out.



In some cases, it may be appropriate to remove entities for deleted assets from Atlas. For example, in a development or test environment, you may choose to clean out specific entities rather than clearing the entire Atlas database. Be careful not to purge entities in a production environment without understanding the impact of removing entities on compliance processes in your organization.

Deleted entities can be removed completely from Atlas by using the REST API call PUT /admin/purge.

When you purge a deleted entity:

- The entity is removed from Atlas.
- Related, dependent entities are also removed. For example, when purging a deleted Hive table, the deleted entities for the table columns, DLL, and storage description are also purged.
- The entity is no longer available in search results, even with Show historical entities enabled.
- Lineage relationships that include the purged entities are removed, which breaks lineages that depend upon a purged entity to show connections between ancestors and descendents.
- Classifications propagated across the purged entities are removed in all descendent entities.
- Classifications assigned to the purged entities and set to propagate are removed from all descendent entities.

Note that classifications can propagate to an entity from more than one source; if one source is purged, the classification will remain on the entity as propagated from the other source.

Purged entities cannot be restored.

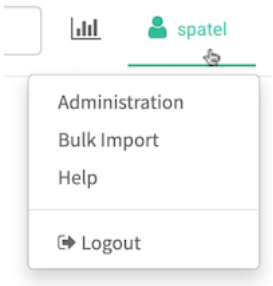
Atlas retains an audit record of the purge operations, which is available through the REST API call POST /admin/audit. This call allows you to retrieve a list of entities purged in a given time interval. In addition, the Administration Audit tab in the Atlas UI records entities that were successfully purged.

Auditing purged entities

The results of a successful entity purge appear in the Audits tab of the Administration page in Atlas.

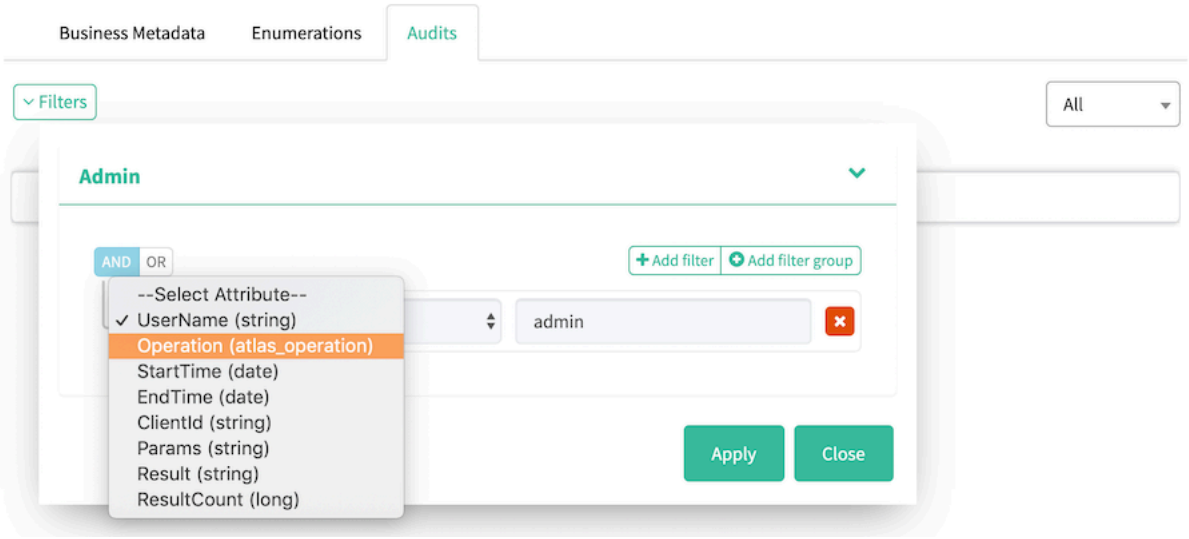
To see an audit of successful purges:

1. Log in to Atlas.
2. To access the Atlas Administration features, choose Administration from the user menu in the top right of the Atlas window.



Users need administrator privileges to access the Administration panel features.

3. Go to the Audits tab.
4. Open the Filters to set one or more filters to reduce the volume of entries.



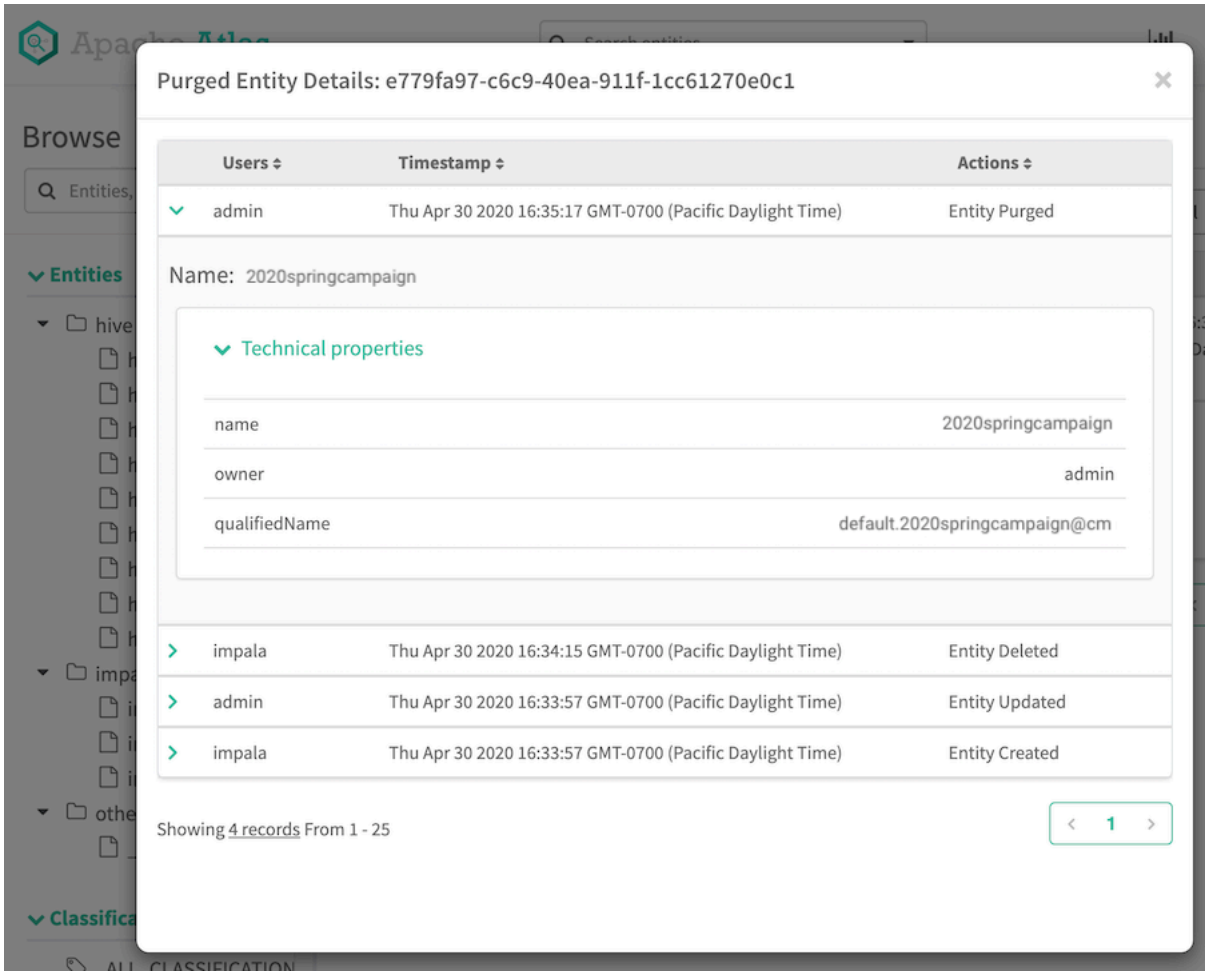
For example:

- Set Operation = PURGE to show only purge audits.
- Set Start Time and End Time to reduce the range of audits.

5. Open a purge audit entry to show a list of GUIDs that were purged.

Use the arrow on the left end of the row to show the details of the audit entry.

6. Click a GUID to show details about that purged entity.



PUT /admin/purge/ API

The PUT /admin/purge/ API endpoint allows you to remove a list of deleted entities from Atlas.

To purge deleted entities, use the PUT method on the /admin/purge/ endpoint with a payload containing a JSON list of Atlas GUIDs:

```
PUT /api/atlas/admin/purge/
```

This call takes a list of GUIDs for Atlas entities; each entity in the list is purged from Atlas if the entity is already marked as deleted. This call requires a user account with Atlas administrator privileges. The successfully purged entities are listed in the audit log, referenced by their GUIDs.

The parameters include:

Parameter	Format / Value	Description
body	JSON list: ["GUID", "GUID", ...]	One or more Atlas entity GUIDs for deleted entities that you want to purge from Atlas. If the list includes an entity that is not already marked as deleted, the entity will be ignored and Atlas will process the remaining entities in the list. Entity GUIDs appear on the URL for the entity detail page and in the metadata returned from an API search call such as GET /search/dsl or POST /search/basic.
header: content type	application/json	Atlas expects the payload of the call to be in JSON format.

Parameter	Format / Value	Description
header: authorization	Per your environment	Atlas requires authentication, which you can provide with username and password as a parameter in a cURL call (-u username:password) or using cookies or other generated methods to pass appropriate credentials. The user must have Atlas administrator privileges.

The response from a PUT /admin/purge call is a JSON-formatted list of the entities that were successfully purged. The content includes the GUID and qualified name of each entity, its status ("DELETED"), and lists of classifications, terms, labels, and related entities that were associated with the purged entity. If no entities are purged, the response is an empty list {}.

Note that the number of entities provided in the request may not match the number of entities included in the response. When Atlas purges an entity, it also purges all additional entities dependent on the indicated entity. Thus if you request to purge a single Hive table, the response includes an entry for the Hive table and entries for each of the table's columns, its DDL, and its storage description.

For example, a cURL command to purge two Hive tables might look like the following, where the authorization is passed in the call as an encrypted string:

```
curl -X PUT 'http://host3.acme.com:31000/api/atlas/admin/purge/' \
-H 'Content-Type: application/json' \
-H 'Authorization: Basic YWRtaW46YWRtaW4=' \
-d '["b9355eab-bbf5-4cd6-b711-12f85a3e9d01", "9fed31f5-0a27-40dc-ba97-96d1
53fc297b"]'
```

The response would include each table, its DDL, its storage description, and all its columns:

```
{
    "mutatedEntities": {
        "PURGE": [
            {
                "typeName": "hive_table_ddl",
                "attributes": {
                    "qualifiedName": "default.2020springcampaign@cm:157896
8155000"
                },
                "guid": "6cfb43ba-d6ec-4628-b98c-da13a7fe35a0",
                "status": "DELETED",
                "displayText": "default.2020springcampaign@cm:1578968155000
",
                "classificationNames": [],
                "meaningNames": [],
                "meanings": [],
                "isIncomplete": false,
                "labels": []
            },
            {
                "typeName": "hive_table",
                "attributes": {
                    "owner": "admin",
                    "createTime": 1578968155000,
                    "qualifiedName": "default.2020springcampaign@cm",
                    "name": "2020springcampaign"
                },
                "guid": "9fed31f5-0a27-40dc-ba97-96d153fc297b",
                "status": "DELETED",
                "displayText": "2020springcampaign",
                "classificationNames": [
                    "Fact"
                ],
                "meaningNames": [],
                "meanings": [],
```

```
                "isIncomplete": false,
                "labels": [
                    "ReviewComplete"
                ]
            },
            {
                "typeName": "hive_storagedesc",
                "attributes": {
                    "qualifiedName": "default.2020springcampaign@cm_storage"
                },
                "guid": "ed59c502-64d9-485a-b5b7-fd2f3d41e2b8",
                "status": "DELETED",
                "displayText": "default.2020springcampaign@cm_storage",
                "classificationNames": [],
                "meaningNames": [],
                "meanings": [],
                "isIncomplete": false,
                "labels": []
            },
            {
                "typeName": "hive_column",
                "attributes": {
                    "owner": "admin",
                    "qualifiedName": "default.2020springcampaign.id@cm",
                    "name": "id"
                },
                "guid": "1f8c8d86-f9d8-4810-889b-0dbfee2c73ff",
                "status": "DELETED",
                "displayText": "id",
                "classificationNames": [],
                "meaningNames": [],
                "meanings": [],
                "isIncomplete": false,
                "labels": []
            },
            {
                "typeName": "hive_column",
                "attributes": {
                    "owner": "admin",
                    "qualifiedName": "default.2020springcampaign.name@cm",
                    "name": "name"
                },
                "guid": "72d689d2-6fae-4de3-bb75-27ab764e1083",
                "status": "DELETED",
                "displayText": "name",
                "classificationNames": [],
                "meaningNames": [],
                "meanings": [],
                "isIncomplete": false,
                "labels": []
            },
            {
                "typeName": "hive_table_ddl",
                "attributes": {
                    "qualifiedName": "default.2019wintercampaign@cm:14889
68156001"
                },
                "guid": "6cfb43ba-d6ec-4628-c98c-bc13a7fe3982",
                "status": "DELETED",
                "displayText": "default.2019wintercampaign@cm:148896815600
1",
                "classificationNames": [],
                "meaningNames": [],
                "meanings": [],
```

```
                "isIncomplete": false,
                "labels": []
            },
            {
                "typeName": "hive_table",
                "attributes": {
                    "owner": "admin",
                    "createTime": 1488968156001,
                    "qualifiedName": "default.2019wintercampaign@cm",
                    "name": "2019wintercampaign"
                },
                "guid": "9fed31f5-0a27-40dc-ba98-bc13a7fe3983",
                "status": "DELETED",
                "displayText": "2019wintercampaign",
                "classificationNames": [
                    "Fact"
                ],
                "meaningNames": [],
                "meanings": [],
                "isIncomplete": false,
                "labels": [
                    "ReviewComplete"
                ]
            }, <additional entries for Hive table columns, ddl, storage de
scription>
        ]
    }
}
```

POST /admin/audits/ API

The POST /admin/audits/ API endpoint can be used to retrieve the entity purge operations that have occurred in a given time period.

To report what entities were purged, use the POST method on the /admin/audits/ endpoint with a payload containing a JSON-formated query for purged entities:

```
POST /api/atlas/admin/audits/
```

where the parameters include:

Parameter	Format / Value	Description
body: auditFilters	JSON-formatted query including: • username (required) • purge operation (required) • start time • end time The results can further be controlled with: • limit • offset • sort by (required) • sort order	The payload describes a query that returns PURGE operations performed by a specific user. The query can include a filter for a specific time range, where the start and end times are specified in UNIX epoch time. The query results can also be paged using a result count (limit) and offset so multiple calls can be made to retrieve unique result sets.
header: content type	application/json	Atlas expects the payload of the call to be in JSON format.
header: authorization	Per your environment	Atlas requires authentication, which you can provide with a username and password as a parameter in a cURL call (-u username:password) or using cookies or other generated methods to pass appropriate credentials. The user must have Atlas administrator privileges.

The audit filter in this call uses the same syntax as Atlas' search filters. Here are some guidelines that are useful for using this interface for purge auditing:

- Valid operators for the time criteria include less than (lt), greater than (gt), less than or equal to (lte), greater than or equal to (gte), equal to (eq), and not equal to (neq).
- Valid operators for the string criteria include like, startsWith, endsWith, contains, isNull, notNull, and equals (eq).
- The sortOrder can be ASCENDING or DESCENDING.

The response from a POST /admin/audits call is a JSON-formatted list of the purge operations that occurred in the specified time range. The content includes the parameters passed in the purge call and the list of GUIDs for the entities that were successfully purged. If no entities are purged, the response is an empty list [].

For example, a cURL command to return the most recent 10 purge operations might look like the following where the authorization is passed in the call as a username and clear-text password:

```
curl -X POST 'http://host3.acme.com:31000/api/atlas/admin/audits/' \
-H 'Content-Type: application/json' \
-d '{
    "auditFilters": {
        "condition": "AND",
        "criterion": [
            {
                "attributeName": "userName",
                "operator": "like",
                "attributeValue": "admin"
            },
            {
                "attributeName": "operation",
                "operator": "like",
                "attributeValue": "PURGE"
            }
        ]
    },
    "limit": 10,
    "offset": 0,
    "sortBy": "endTime",
    "sortOrder": "DESCENDING"
}
' -u username:password
```

The response includes a list of purge operations (this example is shortened to only two entries):

```
[
    {
        "guid": "d93c7664-6e41-4aa9-aed8-b740b985c9a0",
        "userName": "admin",
        "operation": "PURGE",
        "params": "[ac2772e8-984d-4ab6-9e99-323f1be2d3c0, 90231026-6581-4
168-8828-f010aa9b097c]",
        "startTime": 1576261685009,
        "endTime": 1576261685197,
        "clientId": "10.16.1.255",
        "result": "ac2772e8-984d-4ab6-9e99-323f1be2d3c0,ae143e74-48d4-4f4b
-8164-192bc842ed3b,dd742369-2a2c-44ee-902c-4d78a55b7100,90231026-6581-4168-8
828-f010aa9b097c"
    },
    {
        "guid": "b964079d-1f55-43f3-af7b-3a3701378826",
        "userName": "admin",
        "operation": "PURGE",
        "params": "[35b7aaad-2aaf-4af8-a043-e7b524e1314e]",
        "startTime": 1576028426951,
        "endTime": 1576028427165,
```

```
        "clientId": "10.16.1.255",
        "result": "35b7aaad-2aaf-4af8-a043-e7b524e1314e,add45b8a-4bef-4ebb-
a8d0-0b8b920f068d"
    },...
]
```

Related Information

Auditing purged entities

Apache Atlas technical metadata migration reference

This documentation includes an exhaustive reference of how Cloudera Navigator technical metadata is migrated into Atlas entities.

The migration process moves technical metadata from Navigator to Atlas in one of these ways:

- One-to-one mapping. There are no field mapping notes.
- Type conversion. The field mapping notes indicate the new type applied.
- Data conversion. The field mapping notes indicate how the data was converted, such as from a string to a Boolean value (for example, type=FILE to isFile=True).
- No reason to migrate. Navigator stored a value that has no use in Atlas, such as the system ID. The field mapping notes indicate that the value is not used in Atlas.
- No value in Navigator. In most cases, if an Atlas field does not have an equivalent in Navigator, the Atlas field is left as null. The migration notes indicate if a value is filled in by default.
- Not migrated. There is one case where metadata in Navigator is not migrated and potentially the information is lost: the Spark operation (spark_process) metadata for principal is not migrated to Atlas. The principal is migrated at the operation execution (spark_process_execution) entity level. The field mapping notes indicate this case.

All Atlas entities share "system" attributes. The mapping for these attributes is described once but apply to all entities.

Related Information

Mapping Navigator business metadata to Atlas

System metadata migration

All migrated entities in Atlas include the same top-level metadata attributes, such as name, description, and creation time.

The following sections describe how Navigator "common" entity metadata is mapped to Atlas "system" metadata. If Atlas requires metadata that wasn't available in Navigator, the migration notes describe how the Atlas metadata values are generated.

Navigator Metadata	Atlas Metadata	Migration Notes
created	createTime	Converted to date type.
deleted	status	If True in Navigator, Atlas status is set to "DELETED"; otherwise status is set to "ACTIVE".
description	attributes.userDescription	
extractorRunId		No equivalent in Atlas.
identity	guid	Converted to Atlas value
internalType	typeName	Converted to Atlas values.
lastModified	updateTime	Converted to date type.
lastModifiedBy	updatedBy	

Navigator Metadata	Atlas Metadata	Migration Notes
name	attributes.displayName	
originalName	attributes.name	
originalDescription	attributes.description	
owner	attributes.owner	
packageName		No equivalent in Atlas.
properties	customAttribues	Added to the Atlas entity custom attributes as a key value pair with the Navigator name as the key.
sourceId		Inferred rather than migrated.
technicalProperties	customAttribues	Added to the Atlas entity custom attributes as a key value pair with the Navigator name as the key.
tags	labels	
	attributes.qualifiedName	Atlas uniquely identifies entity instances with the qualified name. See the entity-specific migration reference to see how these values are generated.
	clusterName	Supplied in nav2atlas migration command.
	homeId	Not used currently in Atlas.
	isProxy	Not used currently in Atlas.
	provenanceType	Not used currently in Atlas.
	version	Not used currently in Atlas.

HDFS entity metadata migration

HDFS metadata entities are migrated from Navigator to Atlas when they appear in a lineage relationship from Hive, Impala, or Spark processes.

The following sections describe how metadata is mapped from Navigator to Atlas; if Atlas requires metadata that wasn't available in Navigator, the migration notes describe how the Atlas metadata values are generated.

Migrated entities include:

- HDFS Directory on page 25
- HDFS File on page 26

For entity metadata that is common to all entities, see System metadata migration on page 24.

HDFS Directory

Navigator fselement entities of type=DIRECTORY are migrated to Atlas hdfs_path entities with the isFile attribute set to false.

Navigator Metadata	Atlas Metadata	Migration Notes
blockSize		Null for directories.
created	attributes.createTime	Converted to date type.
ezKeyName		Null for directories.
fileSystemPath	attributes.path	
group	attributes.group	
lastAccessed	attributes.modifiedTime	Converted to date type.
mimeType		Null for directories.

Navigator Metadata	Atlas Metadata	Migration Notes
owner	attributes.owner	
permissions	attributes.posixPermissions	Converted to Atlas values.
replication	attributes.numberOfReplicas	Null for directories.
size	attributes.fileSize	Null for directories.
type	attributes.isFile	The Navigator type=DIRECTORY property is converted to isFile=FALSE.
	attributes.isSymLink	Defaults to FALSE.
	attributes.nameServiceId	Optional in Atlas.
	attributes.qualifiedName	Generated as a string in the format <path>@clustername.

HDFS File

Navigator fselement entities of type=FILE are migrated to Atlas hdfs_path entities with the isFile attribute set to true.

Navigator Metadata	Atlas Metadata	Migration Notes
blockSize		Added to the Atlas entity custom attributes as a key value pair with the Navigator name as the key.
created	attributes.createTime	Converted to date type.
ezKeyName		Added to the Atlas entity custom attributes as a key value pair with the Navigator name as the key.
fileSystemPath	attributes.path	
group	attributes.group	
lastAccessed	attributes.modifiedTime	Converted to date type.
mimeType		Added to the Atlas entity custom attributes as a key value pair with the Navigator name as the key.
owner	attributes.owner	
parentPath	attributes.extendedAttributes	
permissions	attributes.posixPermissions	Converted to Atlas values.
replication	attributes.numberOfReplicas	
size	attributes.fileSize	
type	attributes.isFile	The Navigator type=FILE property is converted to isFile=TRUE.
	attributes.isSymLink	Defaults to FALSE.
	attributes.nameServiceId	Optional in Atlas.
	attributes.qualifiedName	Generated as a string in the format <path>@clustername.

Hive entity metadata migration

Hive metadata entities are fully migrated from Navigator to Atlas.

The following sections describe how metadata is mapped from Navigator to Atlas; if Atlas requires metadata that wasn't available in Navigator, the migration notes describe how the Atlas metadata values are generated.

Migrated entities include:

- Hive Database on page 27
- Hive Table on page 27
- Hive View on page 28

- Hive Storage Description on page 28
- Hive Column on page 28
- Hive Process on page 28
- Hive Column Lineage on page 29
- Hive Process Execution on page 29

Hive Database

Navigator hv_database entities are migrated to Atlas hive_db entities.

Navigator Metadata	Atlas Metadata	Migration Notes
fileSystemPath	attributes.location	
firstClassParentId	Not needed in Atlas	
params	attributes.parameters	
parentPath		Not needed in Atlas.
technicalProperties	customAttributes	
type		Inferred rather than migrated.
	attributes.ownerType	
	attributes.parameters	
	attributes.qualifiedName	Generated as a string in the format dbname@clustername.

Hive Table

Navigator hv_table entities are migrated to Atlas hive_table entities.

Navigator Metadata	Atlas Metadata	Migration Notes
clusterByColNames	bucketCols	Not needed in Atlas.
group	attributes.parameters	Added to Atlas entity as a key value pair with the Navigator name as the key.
params	attributes.parameters	Added to Atlas entity as a key value pair with the Navigator name as the key.
partColNames	relationshipAttributes.partitionKeys	
sortByColName	attributes.sortCols	Converted from string to array type.
technicalProperties	attributes.parameters	Added to the Atlas entity attributes as a key value pair with the Navigator name as the key.
	attributes.aliases	Defaults to null.
	attributes.comment	Defaults to null.
	attributes.lastAccessTime	Defaults to null.
	attributes.qualifiedName	Generated as a string in the format <parent_db>.<tablename>@<clustername>.
	attributes.retention	Defaults to null.
	attributes.tableType	Defaults to null.
	attributes.temporary	Defaults to null.
	attributes.viewOriginalText	Defaults to null.
	attributes.viewExpandedText	Defaults to null.

Hive View

Navigator hv_view entities are migrated to Atlas hive_table entities. Atlas does not distinguish between Hive tables and Hive views.

Hive Storage Description

Atlas includes a separate entity that represents how Hive table data is stored. Navigator included this metadata as part of its hv_table entity and the logical-physical lineage relationship. The migration creates the Atlas hive_storagedesc entity using metadata from the HMS table information.

Navigator Metadata	Atlas Metadata	Migration Notes
compressed	attributes.compressed	
fileSystemPath	attributes.location	
inputFormat	attributes.inputFormat	
outputFormat	attributes.outputFormat	
partColNames	attributes.bucketColNames	
serdeLibName	attributes.serdeInfo.serializationLib	
serdeProps	attributes.serdeInfo	
sortByColNames	attributes.sortCols	Converted from string to array type.
	attributes.numBuckets	
	attributes.parameters	
	attributes.qualifiedName	Generated as a string in the format <parent_db>.<tablename>@<clustername>_storage.
	attributes.sortedAsSubDirectories	

Hive Column

Navigator hv_column entities are migrated to Atlas hive_column entities. Note that the Atlas owner value is not available from Navigator and remains blank.

Navigator Metadata	Atlas Metadata	Migration Notes
dataType	attributes.type	
firstClassParentId		Not used in Atlas.
fieldIndex	attributes.position	
parentPath		Not used in Atlas.
	attributes.comment	Defaults to null.
	attributes.owner	Defaults to null.
	attributes.qualifiedName	Generated as a string in the format <parent_db>.<tablename>.<columnname>@<clustername>.

Hive Process

Navigator hv_query entities are migrated to Atlas hive_process entities.

Navigator Metadata	Atlas Metadata	Migration Notes
inputs	inputs	Points to the input entities as relationship attributes.
outputs	outputs	Points to the output entities as relationship attributes.

Navigator Metadata	Atlas Metadata	Migration Notes
queryHash		Not used in Atlas.
queryText	attributes.queryText	Not used currently in Atlas.
sourceId		Not used in Atlas.
unparsed		Not used in Atlas.
wfIds		Not used in Atlas.
	attributes.startTime	Not used currently in Atlas.
	attributes.endTime	Not used currently in Atlas.
	attributes.userName	Not used currently in Atlas.
	attributes.operationType	Defaults to null.
	attributes.qualifiedName	Generated as a string with the operation, input entities, and output entities, where each entity is noted by <asset_qualifiedName>:<createTime> and entries are separated by colons, and an arrow shows the break between input and output entities. For example: `dbblue.table_aqua@clustercolor:1589373411000->dbblue.table_teal@clustercolor:1589394039000`
	attributes.queryId	Defaults to null.
	attributes.queryGraph	Defaults to null.
	attributes.recentQueries	Defaults to null.

Hive Column Lineage

Navigator hv_query_part entities are migrated to Atlas hive_column_lineage entities.

Navigator Metadata	Atlas Metadata	Migration Notes
inputs	attributes.inputs	Points to the input column entities as relationship attributes.
outputs	attributes.outputs	Points to the output column entities as relationship attributes.
firstClassParentId	attributes.query	Points to the parent hive_process entity as a relationship attribute.
originalName	attributes.qualifiedName	Generated as a string with the operation, input entities, output entities, and target column name, where each entity is noted by <column_qualifiedName>:<createTime> and entries are separated by colons, and an arrow shows the break between input and output entities. For example: `dbblue.table_aqua@clustercolor:1589373411000->dbblue.table_teal@cm:1589390050000:column_beach`
	attributes.dependencyType	Set to "SIMPLE".
	attributes.expression	Defaults to null.

Hive Process Execution

Navigator hv_query_execution entities are migrated to Atlas hive_process_execution entities.

Navigator Metadata	Atlas Metadata	Migration Notes
inputs	inputs	Points to the input entities as relationship attributes.
outputs	outputs	Points to the output entities as relationship attributes.

Navigator Metadata	Atlas Metadata	Migration Notes
ended	attributes.endTime	
operation	attributes.process	Points to the parent hive_process entity as a relationship attribute.
originalName	attributes.queryText	
principal	attributes.userName	
started	attributes.startTime	
	attributes.hostName	Defaults to null.
	attributes.qualifiedName	Generated as a string with the operation, input entities, output entities, and execution start and end timestamps, where each entity is noted by <asset_qualifiedName>:<creat eTime> and entries are separated by colons, and an arrow shows the break between input and output entities. For example: `dbblue.table_aqua@clustercolor:1589373411000 ->dbblue.table_teal@clustercolor:15893900500 00:1589394047386:1589394064097`
	attributes.queryGraph	Defaults to null.
	attributes.queryId	Defaults to null.
	attributes.queryPlan	Set to "Not Supported".

Impala entity metadata migration

Impala metadata entities are fully migrated from Navigator to Atlas.

The following sections describe how metadata is mapped from Navigator to Atlas; if Atlas requires metadata that wasn't available in Navigator, the migration notes describe how the Atlas metadata values are generated.

Migrated entities include:

- Impala Process on page 30
- Impala Column Lineage on page 31
- Impala Process Execution on page 31

For entity metadata that is common to all entities, see System metadata migration on page 24.

Impala Process

Navigator impala_operation entities are migrated to Atlas impala_process entities.

Navigator Metadata	Atlas Metadata	Migration Notes
inputs	inputs	Points to the input entities as relationship attributes.
outputs	outputs	Points to the output entities as relationship attributes.
queryHash		Not used in Atlas.
queryText	attributes.queryText	Not used currently in Atlas (see Impala Process Execution on page 31).
sourceId		Not used in Atlas.
unparsed		Not used in Atlas.
wfIds		Not used in Atlas.
	attributes.endTime	Not used currently in Atlas.
	attributes.operationType	Defaults to null.

Navigator Metadata	Atlas Metadata	Migration Notes
	attributes.qualifiedName	Generated as a string with the input entities and output entities, where each entity is noted by <asset_qualifiedName>:<createTime> and entries are separated by colons, and an arrow shows the break between input and output entities. For example: `dbblue.table_aqua@clustercolor:1589373411000->dbblue.table_teal@clustercolor:1589390050000`
	attributes.queryGraph	Defaults to null.
	attributes.queryId	Defaults to null.
	attributes.recentQueries	Defaults to null.
	attributes.startTime	Not used currently in Atlas.
	attributes.userName	Not used currently in Atlas.

Impala Column Lineage

Navigator impala_sub_operation entities are migrated to Atlas impala_column_lineage entities.

Navigator Metadata	Atlas Metadata	Migration Notes
inputs	attributes.inputs	Points to the input column entities as relationship attributes.
outputs	attributes.outputs	Points to the output column entities as relationship attributes.
firstClassParentId	attributes.query	Points to the parent impala_process entity as a relationship attribute.
originalName	attributes.qualifiedName	Generated as a string with the input entities, output entities, and target column name, where each entity is noted by <column_qualifiedName>:<createTime> and entries are separated by colons, and an arrow shows the break between input and output entities. For example: `dbblue.table_aqua@clustercolor:1589373411000->dbblue.table_teal@cm:1589390050000:column_beach`
	attributes.dependencyType	Set to "SIMPLE".
	attributes.expression	Defaults to null.

Impala Process Execution

Navigator impala_operation_execution entities are migrated to Atlas impala_process_execution entities.

Navigator Metadata	Atlas Metadata	Migration Notes
inputs	inputs	Points to the input entities as relationship attributes.
outputs	outputs	Points to the output entities as relationship attributes.
ended	attributes.endTime	
operation	attributes.process	Points to the parent impala_process entity as a relationship attribute.
originalName	attributes.queryText	
principal	attributes.userName	
started	attributes.startTime	
	attributes.hostName	Defaults to null.

Navigator Metadata	Atlas Metadata	Migration Notes
	attributes.qualifiedName	Generated as a string with the input entities, output entities, and execution start and end timestamps, where each entity is noted by <asset_qualifiedName>:<createTime> and entries are separated by colons, and an arrow shows the break between input and output entities. For example: `dbblue.table_aqua@clustercolor:1589373411000 ->dbblue.table_teal@clustercolor:1589390050000:1589390056000:1589390058000`
	attributes.queryGraph	Defaults to null.
	attributes.queryId	Defaults to null.
	attributes.queryPlan	Set to "Not Supported".

Spark entity metadata migration

Spark metadata entities are fully migrated from Navigator to Atlas.

The following sections describe how metadata is mapped from Navigator to Atlas; if Atlas requires metadata that wasn't available in Navigator, the migration notes describe how the Atlas metadata values are generated.

Migrated entities include:

- Spark Process on page 32
- Spark Process Execution on page 32

For entity metadata that is common to all entities, see System metadata migration on page 24.

Spark Process

Navigator spark_operation entities are migrated to Atlas spark_process entities.

Notice: The metadata for the principal from Navigator is not migrated to Atlas.

Navigator Metadata	Atlas Metadata	Migration Notes
inputs	relationshipAttributes.inputs	Points to the input directories as relationship attributes.
outputs	relationshipAttributes.outputs	Points to the output directory as a relationship attribute.
principal		Not mapped.
	attributes.currUser	Defaults to null.
	attributes.details	Defaults to null.
	attributes.executionId	Defaults to null.
	attributes.remoteUser	Defaults to null.
	attributes.sparkPlanDescription	
	attributes.qualifiedName	Generated as a string in the format process_id@clustername. For example, application_1589303388872_0001@clustercolor.20324.

Spark Process Execution

Navigator spark_operation_execution entities are migrated to Atlas spark_process_execution entities.

Navigator Metadata	Atlas Metadata	Migration Notes
inputs	relationshipAttributes.inputs	Points to the input files as relationship attributes.

Navigator Metadata	Atlas Metadata	Migration Notes
outputs	relationshipAttributes.outputs	Points to the output files as relationship attributes.
ended	attributes.endTime	
operation	attributes.process	Points to the parent spark_process entity as a relationship attribute.
originalName	attributes.queryText	
principal	attributes.userName	
started	attributes.startTime	
	attributes.hostName	Defaults to null.
	attributes.qualifiedName	Generated as a string in the format process_id-exec.timestamp@clustername. For example, application_1589303388872_0001-exec.1589331800580@clustercolor.

AWS S3 entity metadata migration

S3 metadata entities are migrated from Navigator to Atlas.

The following sections describe how metadata is mapped from Navigator to Atlas; if Atlas requires metadata that wasn't available in Navigator, the migration notes describe how the Atlas metadata values are generated.

Migrated entities include:

- S3 Bucket on page 33
- S3 Object: Directory on page 33
- S3 Object: File on page 34

For entity metadata that is common to all entities, see System metadata migration on page 24.

S3 Bucket

Navigator s3_bucket entities are migrated to Atlas aws_s3_v2_bucket entities.

Navigator Metadata	Atlas Metadata	Migration Notes
encryption	attributes.encryption	
eTag	attributes.eTag	
owner	attributes.owner	
ownerId	attributes.ownerId	
region	attributes.region	
	attributes.qualifiedName	Generated as a string in the format <bucket_name>@<cluster_name>. For example yellow_bucket@clustercolor.

S3 Object: Directory

Navigator s3_object entities of type=DIRECTORY is converted to aws_s3_v2_directory entities.

Navigator Metadata	Atlas Metadata	Migration Notes
bucketName	attributes.bucketName	
depth	customAttributes	Added to the Atlas entity custom attributes as a key value pair with the Navigator name as the key.
encryption	attributes.encryption	
eTag	attributes.eTag	

Navigator Metadata	Atlas Metadata	Migration Notes
fileSystemPath	attributes.storageLocation	
firstClassParentId	relationshipAttributes.container	Points to the parent directory or bucket as a relationship attribute.
implicit	attributes.implicit	
newObject	customAttributes	Added to the Atlas entity custom attributes as a key value pair with the Navigator name as the key.
owner	attributes.owner	
ownerId	attributes.ownerId	
parentPath	attributes.objectPrefix	The Atlas value is derived from the Navigator value (no one-to-one migration).
region	attributes.region	
sequencer	customAttributes	Added to the Atlas entity custom attributes as a key value pair with the Navigator name as the key.
size	attributes.size	
storageClass	attributes.storageClass	.
type		Used to determine the Atlas entity type.
	attributes.qualifiedName	Generated as a string in the format <object_prefix>://<bucket_name>/<directory_name>@<cluster_name>. For example s3a://yellow_bucket/hive_storage_color_table_dir@clustercolor.

S3 Object: File

Navigator s3_object entities of type=file is converted to aws_s3_v2_object entities.

Navigator Metadata	Atlas Metadata	Migration Notes
bucketName	attributes.bucketName	
depth	customAttributes	Added to the Atlas entity custom attributes as a key value pair with the Navigator name as the key.
encryption	attributes.encryption	
eTag	attributes.eTag	
fileSystemPath	customAttributes	Added to the Atlas entity custom attributes as a key value pair with the Navigator name as the key.
firstClassParentId	relationshipAttributes.container	Points to the parent directory or bucket as a relationship attribute.
implicit	attributes.implicit	
newObject	customAttributes	Added to the Atlas entity custom attributes as a key value pair with the Navigator name as the key.
owner	attributes.owner	
ownerId	attributes.ownerId	
parentPath	attributes.objectPrefix	The Atlas value is derived from the Navigator value (no one-to-one migration).
region	attributes.region	
size	attributes.size	
storageClass	attributes.storageClass	
type		Used to determine the Atlas entity type.

Navigator Metadata	Atlas Metadata	Migration Notes
	attributes.qualifiedName	Generated as a string in the format <object_prefix>://<bucket_name>/<directory_name>/<file_name>@<cluster_name>. For example s3a://yellow_bucket/hive_storage_color_table_dir/.hive-staging_hive_2020-03-20_01-43-14_862_995309273321986325-1@clustercolor.

NiFi metadata collection

Atlas collects metadata of events from NiFi clusters to illustrate the high-level relationships among processes and data sets and potentially detailed NiFi event-level lineage.

The metadata connector between NiFi and Atlas is implemented as the NiFi reporting task ReportLineageToAtlas.

This reporting task stores two types of NiFi flow information:

- NiFi flow structure describes the components running within a NiFi flow and how they are connected.
- NiFi data lineage describes how NiFi flows interact with different data assets such as HDFS files or Hive tables. Lineage is determined by analyzing NiFi provenance events.

When the Atlas reporting task runs in a NiFi cluster, the primary node performs the one-time task of inserting the NiFi-specific entity types in Atlas. The primary node sends metadata to Atlas as and when an event occurs in the NiFi cluster. Every node (including primary node) analyzes NiFi provenance events stored in a provenance event repository to create lineage between NiFi and other data assets such as Hive tables and HDFS paths.

The NiFi reporting tasks provide a number of properties to control what metadata is collected and at what level of NiFi events are described in lineage. The implementation details to be aware of are:

1. Mapping the NiFi cluster to a cluster name identifier in the Atlas environment.

   You must specifically note the following:

   - Which cluster name should be used to represent the current NiFi cluster?
   - When NiFi components perform operations on data assets, which cluster does the data asset reside in?

   The mapping can be defined by Dynamic Properties with a name in the 'hostnamePattern.ClusterName' format, having its value as a set of Regular Expression Patterns to match IP addresses or host names to a particular cluster name.

   This mapping is described in detail in Cluster Hostname Resolution in the Apache NiFi documentation.
2. Level of lineage graph detail.

   NiFi Lineage Strategy

Choose a strategy for showing lineage information in Atlas by using one of the following:

- Simple Path

A high-level view of NiFi processes that summarizes how each data asset and NiFi process are related when the data flows among services. Specifically, this strategy maps data I/O provenance events such as SEND/RECEIVE to 'nifi_flow_path' created by NiFi flow structure analysis. The result is that if different data assets go through the same 'nifi_flow_path', the lineage picture gives the impression that there is a relationship among the data assets because they are processed in the same flow. No information is lost; however, as you can use the NiFi provenance events to investigate the details.

This strategy generates the least amount of data in Atlas.

- Complete Path

Lineage is created by traversing provenance events backwards from a DROP event, reporting the entire lineage for a given FlowFile including where it is created, and where it goes. This strategy generates more detail and shows separate lineage pictures for data sets that are not related but happen to be processed through the same flow path.

However, reporting complete flow paths for every single FlowFile produces too many entities in Atlas and needs more computing resources to collect the relevant metadata.

Choose the strategy before implementing lineage collection in a production environment because switching among strategies can leave you with residual entities shown in lineage pictures and the lineage pictures may show different entities for otherwise equivalent processes.

About Namespaces

An entity in Atlas can be identified either by its GUID for any existing objects, or type name and unique attribute if GUID is not known. A qualified name is commonly used as the unique attribute.

One Atlas instance can be used to manage multiple environments, and objects in different environments may have the same name. For example, a Hive table 'request_logs' in two different clusters, 'cluster-A' and 'cluster-B'. For this reason the qualified names contain a so-called metadata namespace.

It is common practice to provide the cluster name as the namespace, but the cluster name it can be any arbitrary string.

With this, a qualified name has a 'componentId@namespace' format. For example, a Hive table qualified name is dbName.tableName@namespace (default.request_logs@cluster-A).

From this NiFi reporting task standpoint, a namespace is needed to be resolved in the following situations:

- To register NiFi component entities. Which namespace should be used to represent the current NiFi environment?
- To create lineages from NiFi components to other datasets. Which environment does the dataset reside in?

To answer such questions, ReportLineageToAtlas reporting task provides a way to define mappings from IP address or hostname to a namespace. The mapping can be defined by Dynamic Properties with a name in the 'hostnamePattern.namespace' format, having its value as a set of Regular Expression Patterns to match IP addresses or host names to a particular namespace.

As an example, following mapping definition resolves namespace 'namespace-A' for IP address such as '192.168.30.123' or hostname 'namenode1.a.example.com', and 'namespace-B' for '192.168.40.223' or 'nifi3.b.example.com'.

Dynamic Property Name for namespace-A

hostnamePattern.namespace-A

Value can have multiple Regular Expression patterns separated by new line

192\.168\.30\.\d+

[^\.]+\.a\.example\.com

Dynamic Property Name for namespace-B

hostnamePattern.namespace-B

Values

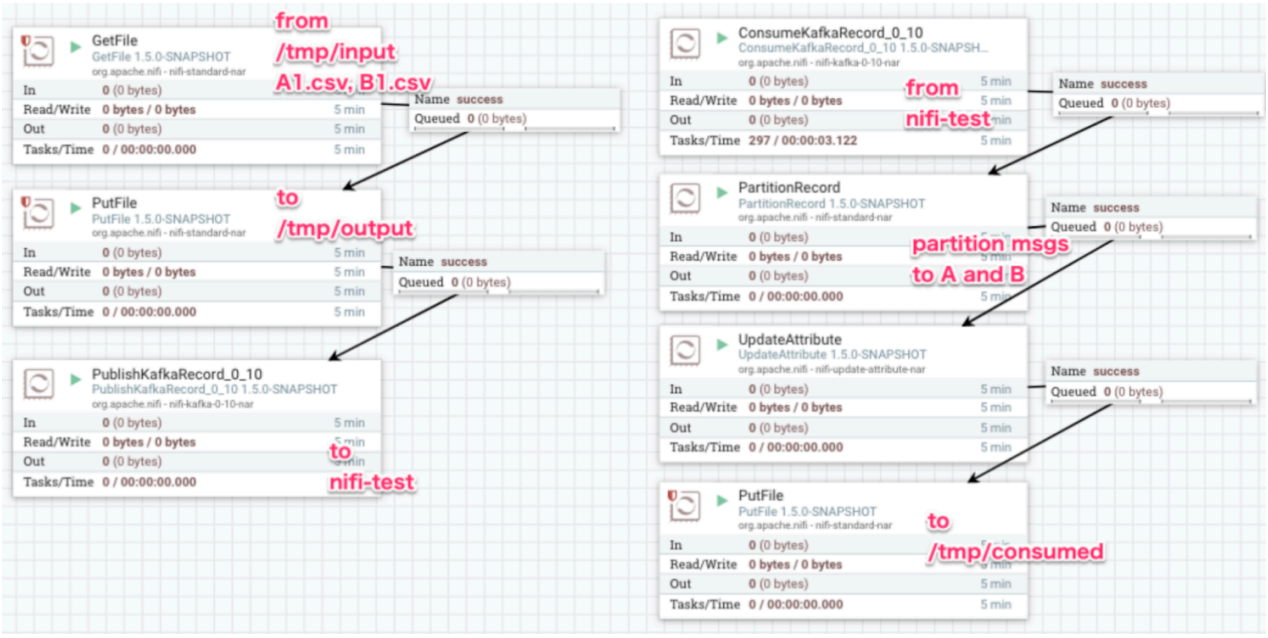
192\.168\.40\.\d+

[^\.]+\.b\.example\.com

If no namespace mapping matches, then a name defined at 'Atlas Default Metadata Namespace' is used.

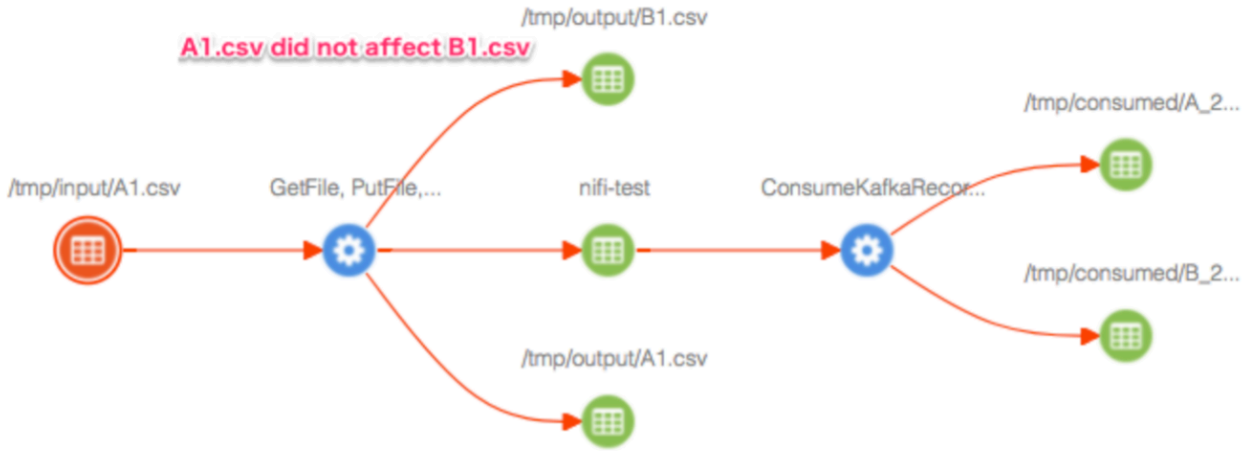
How Lineage strategy works

To illustrate the difference between lineage strategies, consider a sample NiFi flow.



With 'Simple Path', Atlas lineage is reported as seen in the following image, when '/tmp/input/A1.csv' is selected.

Because 'Simple Path' maps I/O events to a 'nifi_flow_path', '/tmp/output/B1.csv' is shown in the lineage graph as the file that is written by the 'GetFile, PutFile...' process.

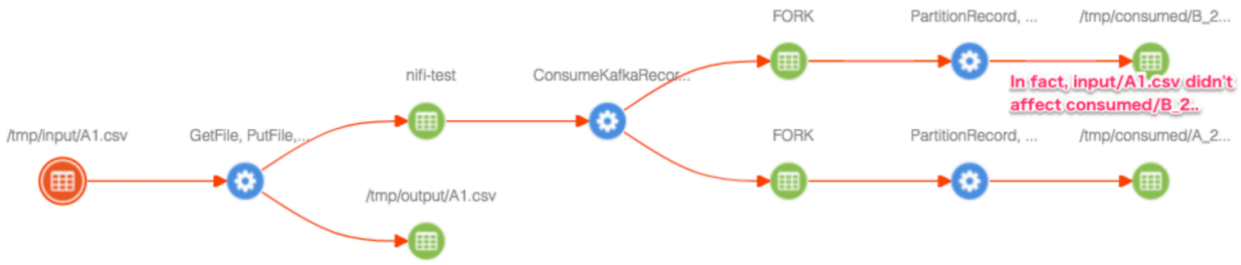


With 'Complete Path', Atlas lineage is reported as seen in the following image. In this use case, 'GetFile, PutFile...' process is not linked to '/tmp/output/B1.csv'.

The 'Complete Path' strategy creates two different 'nifi_flow_path' entities. One for '/tmp/input/A1.csv -> /tmp/output/A1.csv' and another for '/tmp/input/B1.csv -> /tmp/output/B1.csv'.

However, after the data records ingest from A.csv and B.csv into a bigger dataset, 'nifi-test' Kafka topic in the following example (or whatever dataset such as a database table or a concatenated file ... and so on), record level lineage tracking about where it came from can no longer be tracked.

The resulting '/tmp/consumed/B_2..' is shown in the same lineage graph, although the file does not contain any data that came from '/tmp/input/A1.csv'.



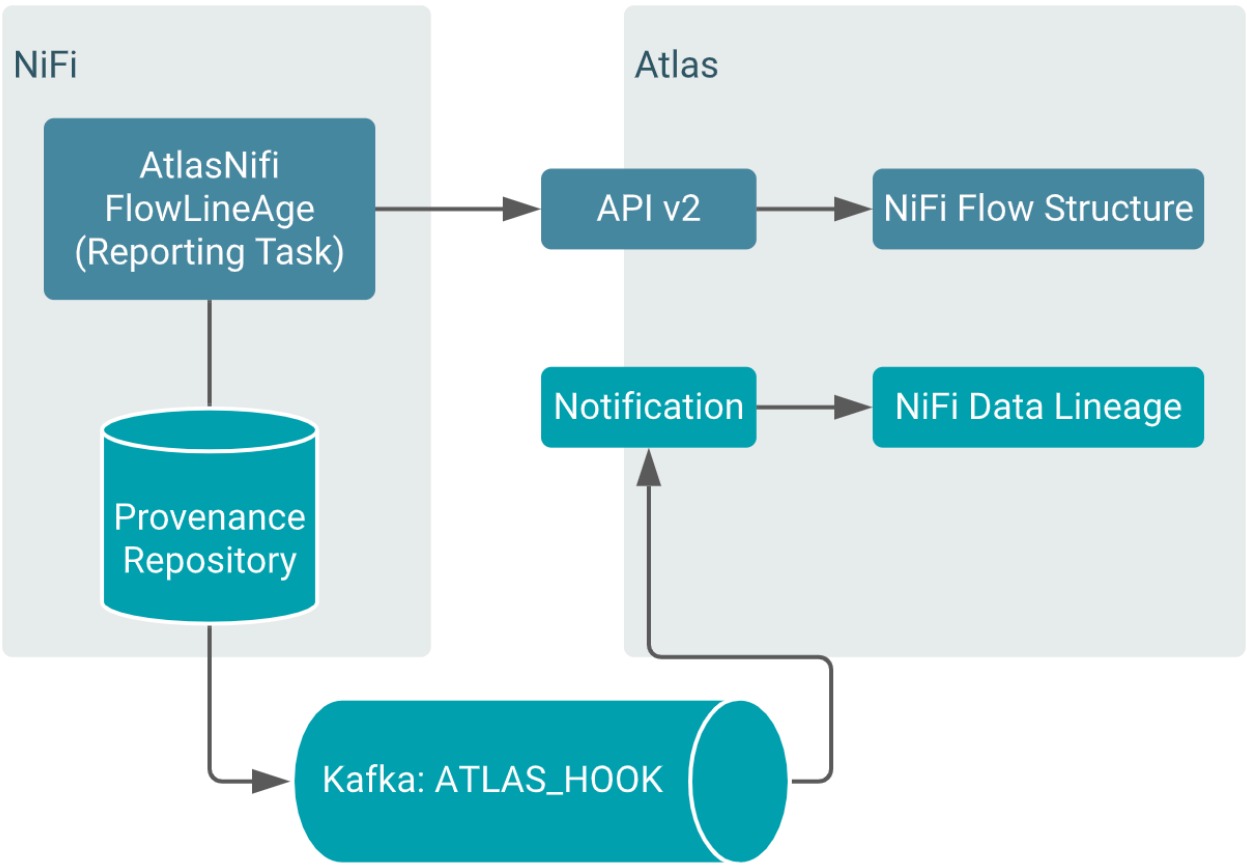
Understanding the data that flow into Atlas

The reporting task stores two types of NiFi flow information.

- NiFi flow structure
- NiFi data lineage

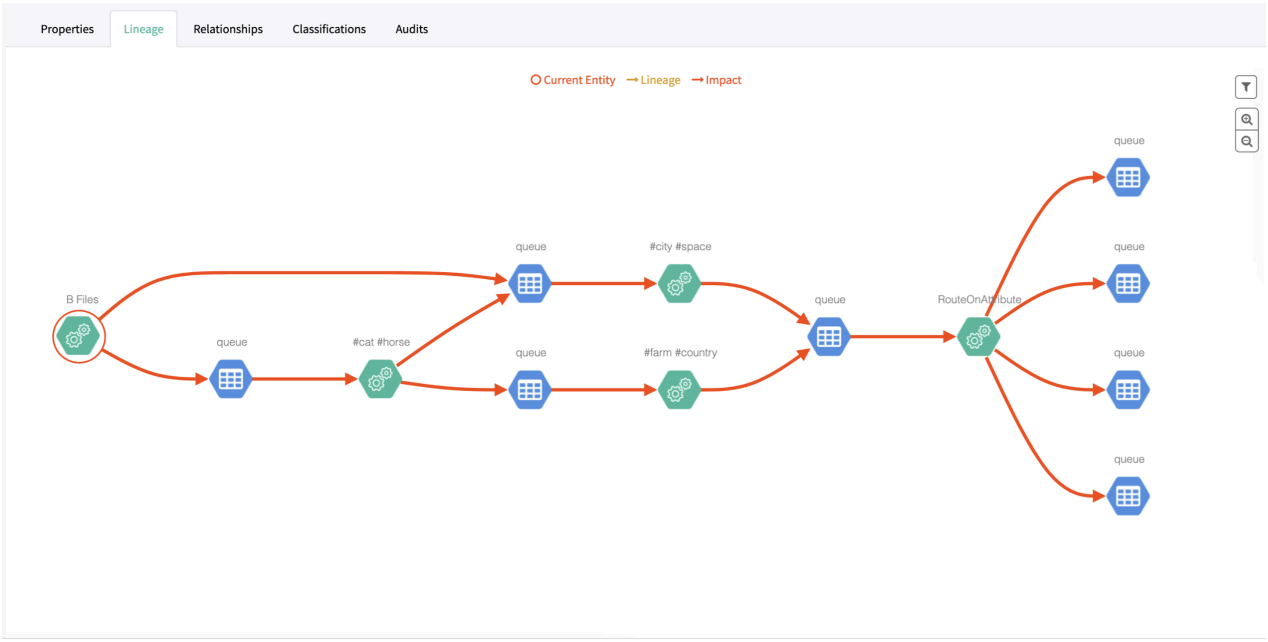
'NiFi flow structure' indicates what components are running within a NiFi flow and how these are connected. It is reported by analyzing current NiFi flow structure, specifically NiFi component relationships.

'NiFi data lineage' indicates what part of NiFi flow interacts with different datasets such as HDFS files or Hive tables. It is reported by analyzing NiFi provenance events.



NiFi lineage

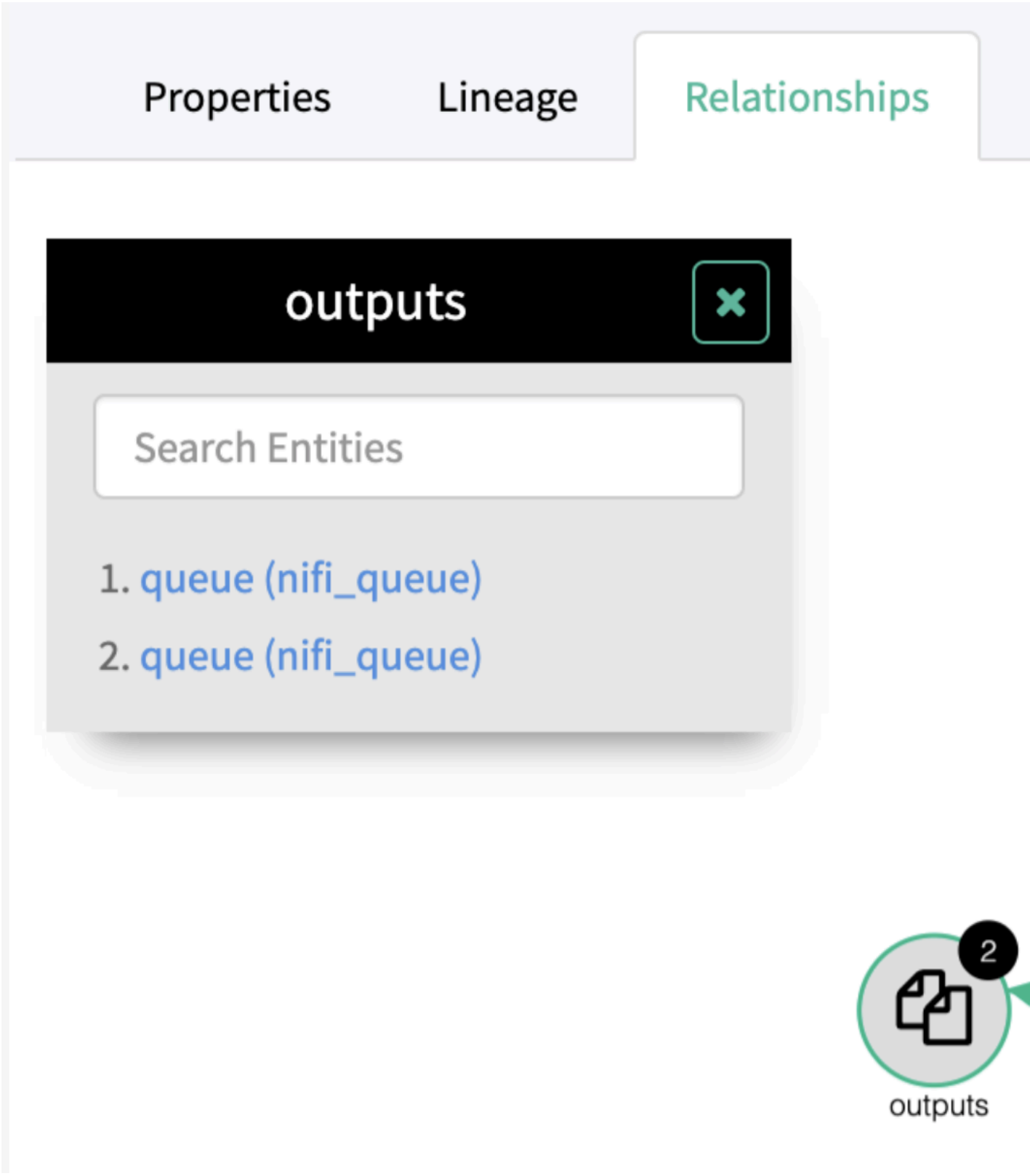
Atlas collects metadata from NiFi to represent the lineage among data assets in the Lineage tab.



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 are provided as an input to processes that lead to the current entity or they are output from processes for which the current entity is an input. NiFi processes follow this pattern.

Atlas NiFi relationships

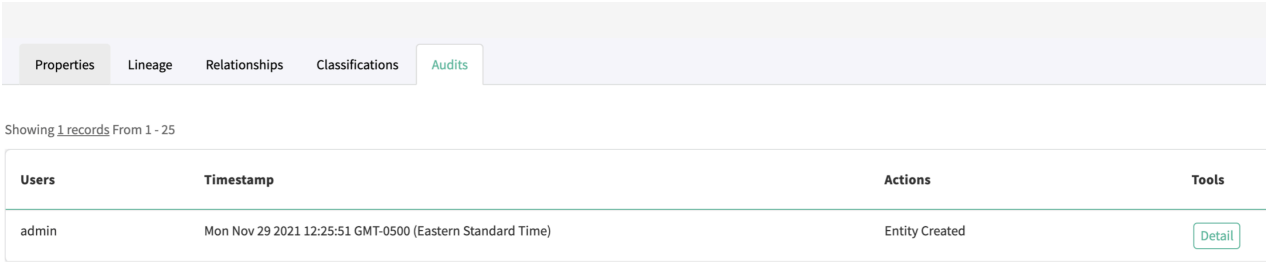
Atlas shows the related entities in the Relationships tab in the Dashboard.



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

Atlas NiFi audit entries

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



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

Note: If the data is active while the entity is created, it displays some additional information. You must be aware that after the entity is created, the flow files do not create an event, unless the processor is listed.

How the reporting task runs in a NiFi cluster

When the reporting task runs in a NiFi cluster, the following tasks are executed only by the primary node:

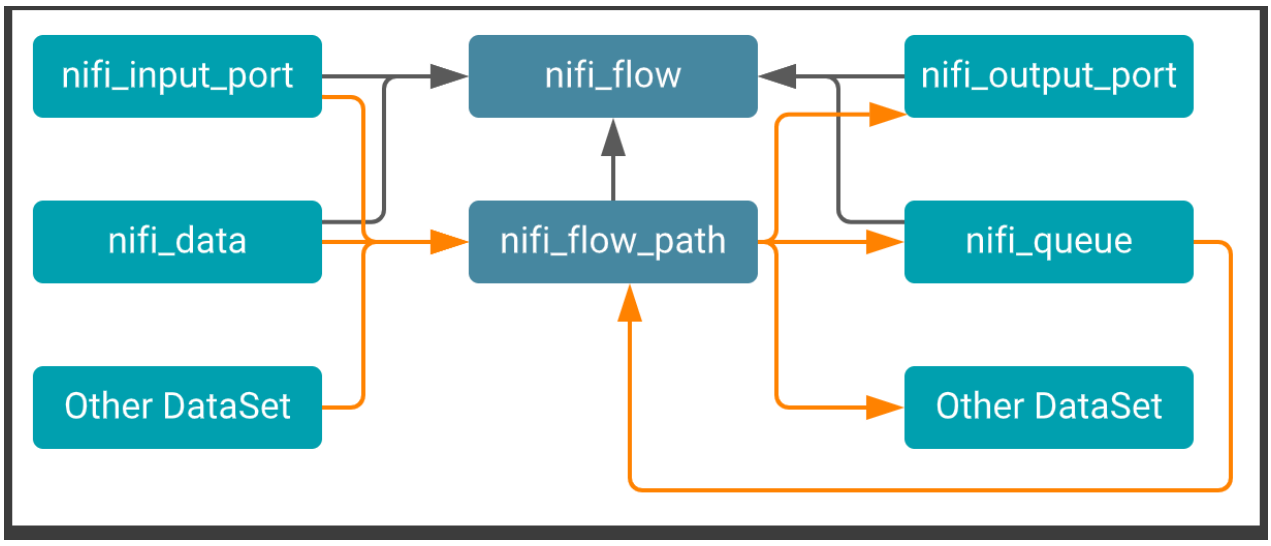
- Create NiFi Atlas types in the Atlas type system.
- Maintain NiFi flow structure and metadata in Atlas both which consists of NiFi component entities such as 'nifi_flow', 'nifi_flow_path', and 'nifi_input(output)_port'.

Every node (including primary node) analyzes NiFi provenance events stored in a provenance event repository, to create lineage between 'nifi_flow_path' and other dataset (for example: Hive tables and HDFS path).

NiFi Atlas Types

The reporting task creates the following NiFi specific types in Atlas Type system if these type definitions are not found.

Green boxes represent sub-types of dataset and blue ones are sub-types of Process. Gray lines represent entity ownership. Red lines represent lineage.



Nifi_flow: Represents a NiFi data flow. As shown in the diagram, nifi_flow owns other nifi_component types. This owning relationship is defined by Atlas 'owned' constraint so that when a 'nifi_flow' entity is removed, all owned NiFi component entities are removed in a cascading manner.

When the reporting task runs, it analyzes and traverses the entire flow structure, and creates NiFi component entities in Atlas. During later runs, it compares the current flow structure with the one stored in Atlas to figure out if any changes have been made since the last time the flow was reported. The reporting task updates NiFi component entities in Atlas if needed.

NiFi components that are removed from a NiFi flow also get deleted from Atlas. However those entities can be seen in Atlas search results or lineage graphs because Atlas uses 'Soft Delete' by default.

- Attributes:
  - qualifiedName: Root ProcessGroup ID@namespace (For example, 86420a14-2fab-3e1e-4331-fb6ab42f58e0@ns1)
  - name: Name of the Root ProcessGroup.
  - url: URL of the NiFi instance. This can be specified using the reporting task 'NiFi URL for Atlas' property.

Nifi_flow_path: Part of a NiFi data flow containing one or more processing NiFi components such as Processors and RemoteGroupPorts. The reporting task divides a NiFi flow into multiple flow paths.

- Attributes:
  - qualifiedName: The first NiFi component Id in a path@namespace (for example: 529e6722-9b49-3b66-9c94-00da9863ca2d@ns1)
  - name: NiFi component names within a path are concatenated (for example: GenerateFlowFile, PutFile, LogAttribute)
  - url: A deep link to the first NiFi component in corresponding NiFi UI

nifi_input/output_port: Represents a RootGroupPort which can be accessed by RemoteProcessGroup through Site-to-Site protocol.

- Attributes:
  - qualifiedName: Port ID@namespace (for example: 3f6d405e-6e3d-38c9-c5af-ce158f8e593d@ns1)
  - name: Name of the Port.

Nifi_data: Represents unknown datasets created by CREATE/SEND/RECEIVE NiFi provenance events that do not have a particular provenance event analyzer.

- Attributes:
  - qualifiedName: ID of a Processor which generated the provenance event@namespace (for example: db8bb12c-5cd3-3011-c971-579f460ebedf@ns1)
  - name: Name of the Processor.

Nifi_queue: An internal dataset of NiFi flows which connects nifi_flow_paths. Atlas lineage graph requires a dataset in between Process entities.

- Attributes:
  - qualifiedName: ID of the first Processor in the destination nifi_flow_path.
  - name: Name of the Processor.

Analysing event analysis

To create lineage describing which NiFi component interacts with what datasets, dataset entity, and Process entity need to be created in Atlas. Specifically, at least three entities are required to draw a lineage graph on Atlas UI.

A Process entity, and a dataset which is referred by a Process 'inputs' attribute, and a dataset referred from an 'outputs' attribute. For example:

```
# With following entities
            guid: 1
            typeName: fs_path (extends dataset)
            qualifiedName: /data/A1.csv@BranchOffice1
            guid: 2
            typeName: nifi_flow_path (extends Process)
            name: GetFile, PutHDFS
            qualifiedName: 529e6722-9b49-3b66-9c94-00da9863ca2d@BranchOff
ice1
            inputs: refer guid(1)
            outputs: refer guid(3)

            guid: 3
            typeName: hdfs_path (extends dataset)
            qualifiedName: /data/input/A1.csv@Analytics
            # Atlas draws lineage graph
            /data/A1.csv -> GetFile, PutHDFS -> /data/input/A1.csv
```

To identify such Process and dataset Atlas entities, this reporting task uses NiFi Provenance Events. At the minimum, the reporting task needs to derive the following details from a NiFi Provenance event record:

- typeName (for example: fs_path, hive_table)
- qualifiedName in uniqueId@namespace (for example: /data/A1.csv@ns1)

'namespace' in 'qualifiedName' attribute is resolved by mapping an IP address or hostname available at the NiFi Provenance event 'transitUri' to a namespace.

For 'typeName' and 'qualifiedName', different analysis rules are needed for different datasets. ReportLineageToAtlas provides an extension point called 'NiFiProvenanceEventAnalyzer' to implement such analysis logic for particular datasets.

When a Provenance event is analyzed, registered NiFiProvenanceEventAnalyzer implementations are searched in the following order to find a best matching analyzer implementation:

1. By component type (for example: KafkaTopic)
2. By transit URI protocol (for example: HDFSPath)
3. By event type, if none of above analyzers matches (for example: Create)

Limitations of Atlas-NiFi integration

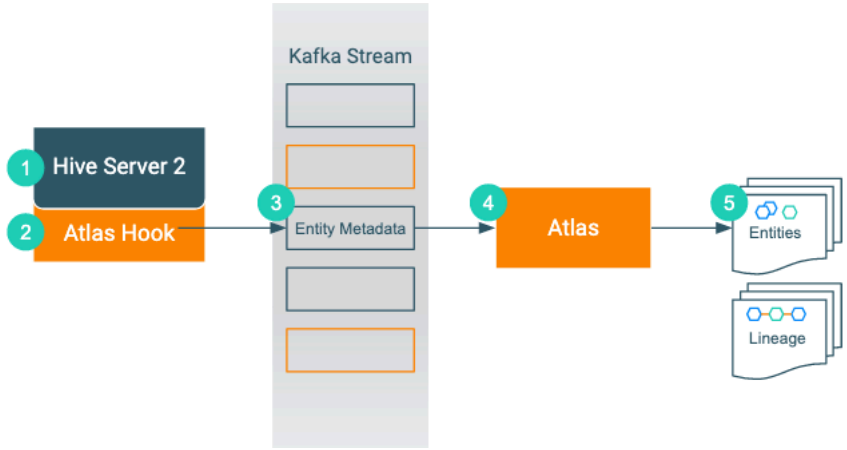
Some limitations of this integration.

- Requires Atlas 0.8-incubating or later. The reporting task requires Atlas REST API version 2, which is introduced in Atlas 0.8-incubating. Older versions of Atlas are not supported.
- Supports limited datasets and Processors. To report lineage to Atlas, the reporting task should be familiar with what a given processor does with a certain dataset. Later, create an 'Atlas Object Id' for a dataset which uniquely identifies an entity in Atlas. Atlas Object ID has a unique properties map, and mostly 'qualifiedName' is set in the unique properties map to identify an entity. The format of a qualifiedName depends on each dataset. To create this Atlas Object ID, you must implement Processor-specific code that analyzes configured properties.
- Requires restart of NiFi to update some ReportingTask properties. As the underlying Atlas client library caches configurations when it runs the first time, some properties of this reporting task can not be updated by stopping, configuring, and restarting the reporting task. The NiFi process needs to be restarted in such cases.

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

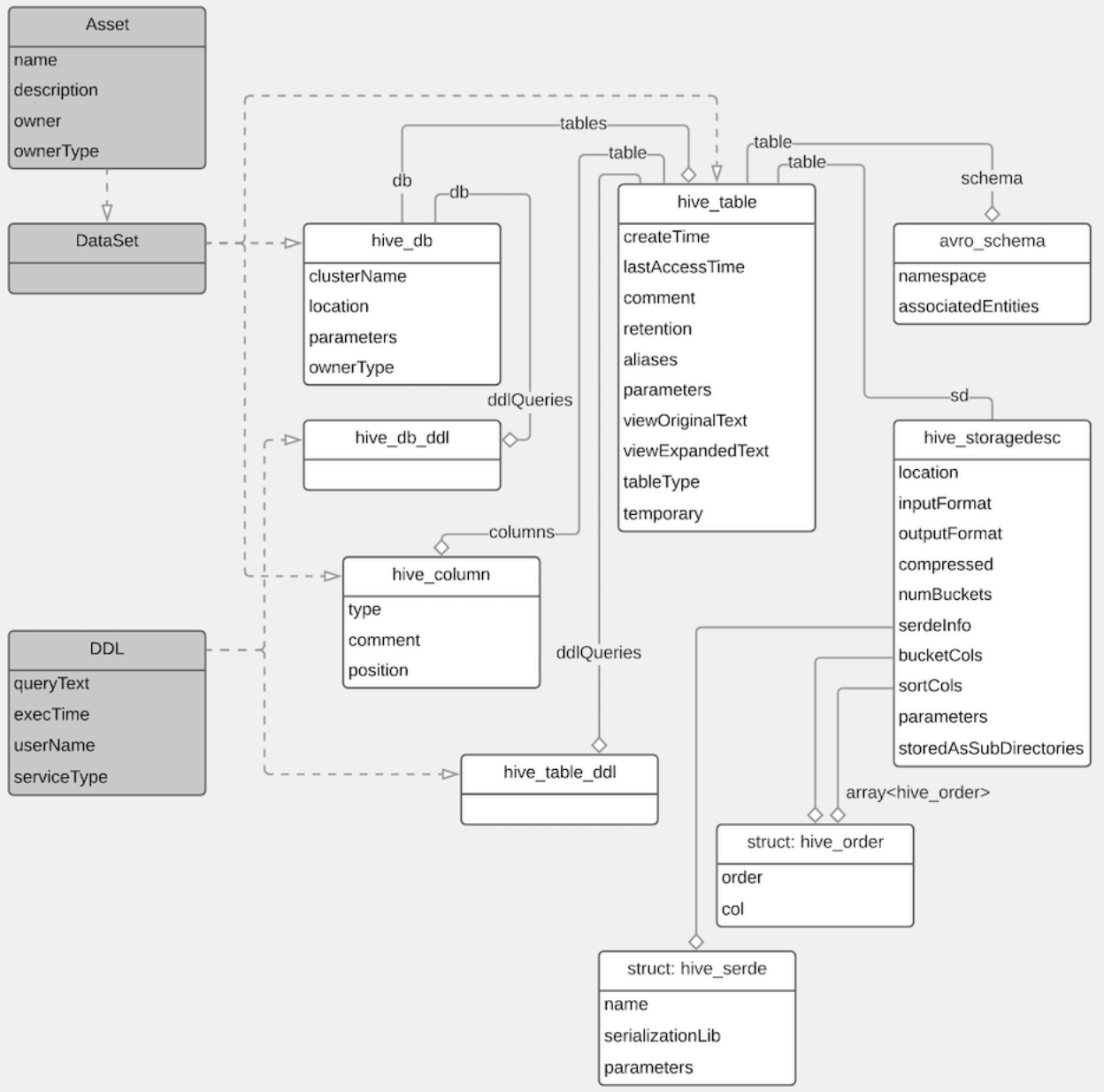
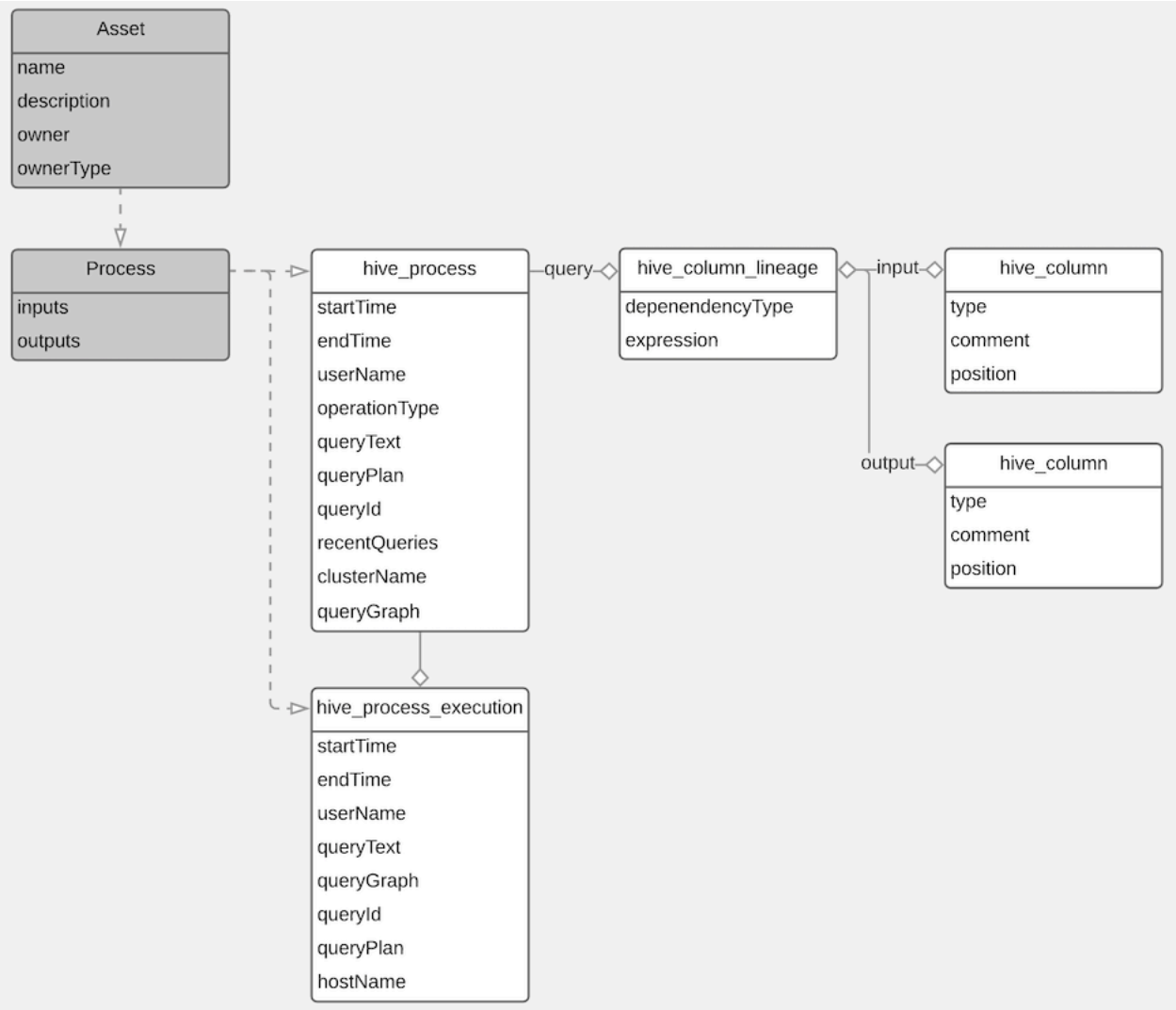


Figure 2: Atlas Entity Types for HiveServer Processes



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 yyyymmddhhmmss>_<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: • 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.

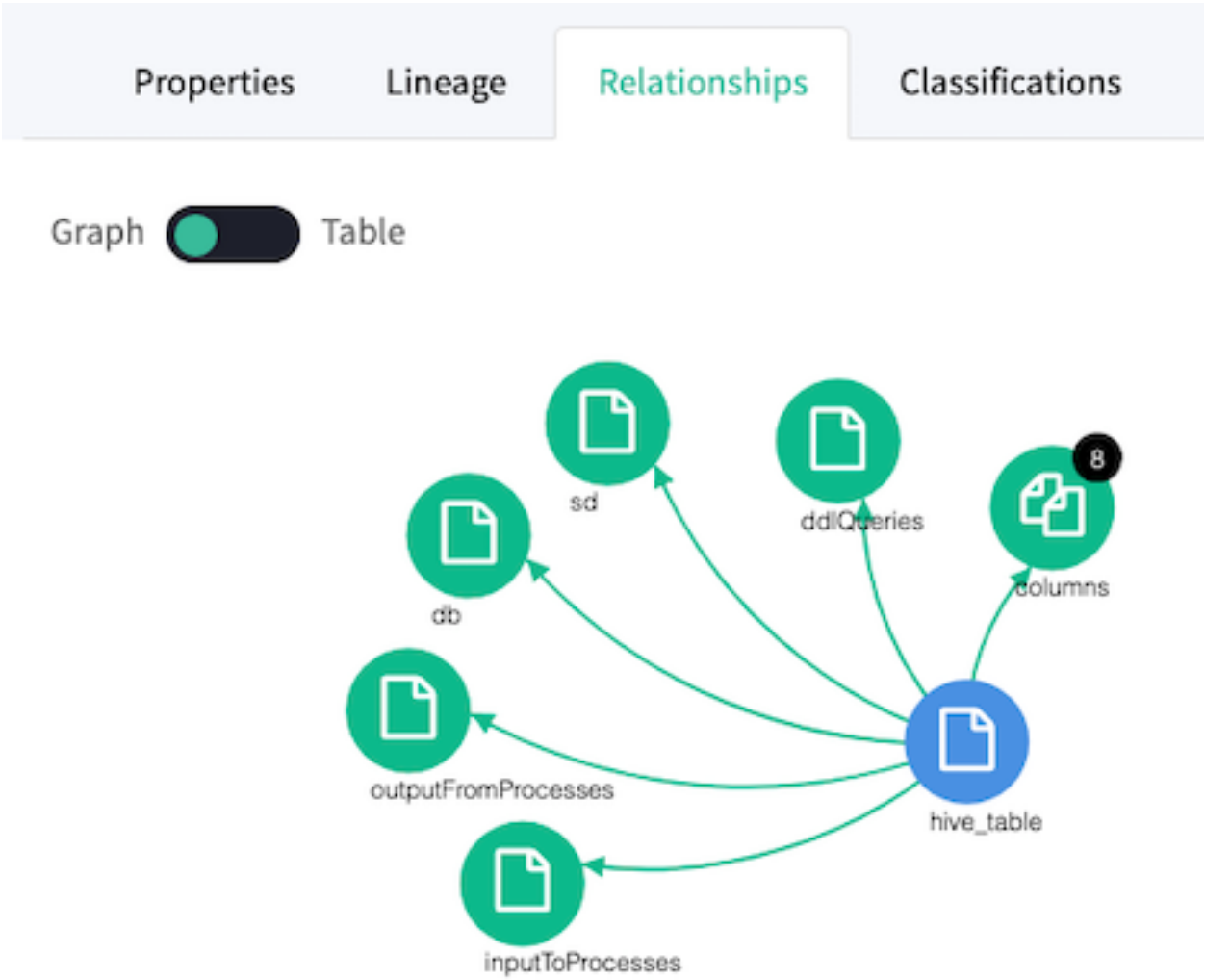
Hive Storage Description

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.

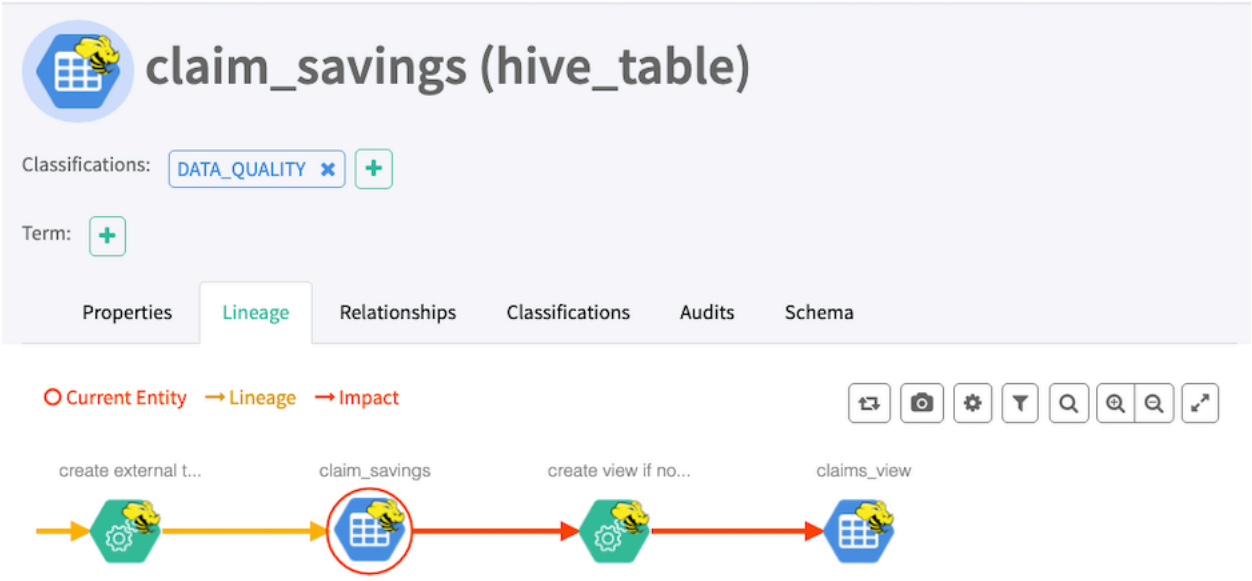
The Relationship tab shows the relationships that exist for an entity. Use this view to navigate among related entities.



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.



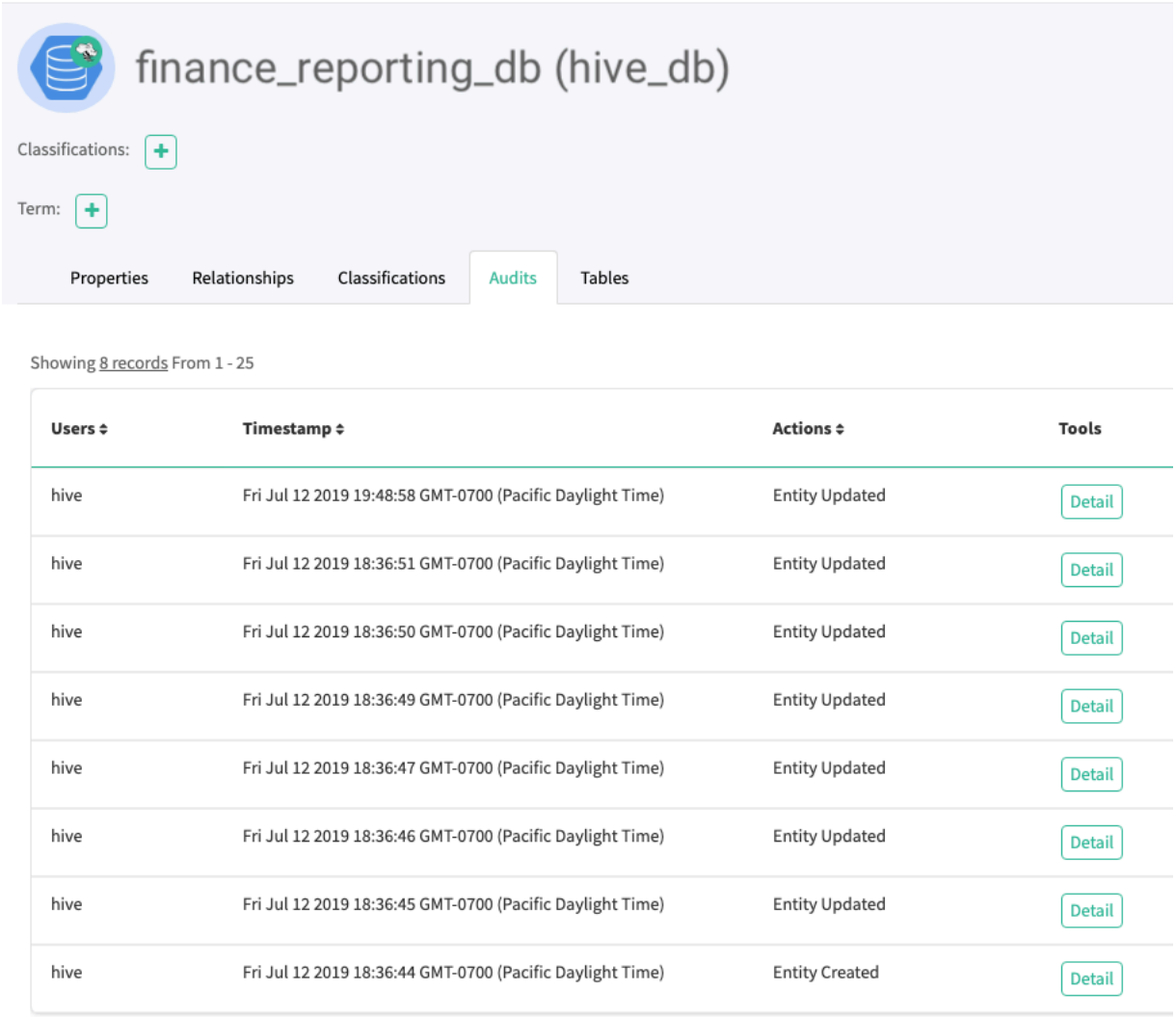
Related Information

Viewing lineage

HiveServer audit entries

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

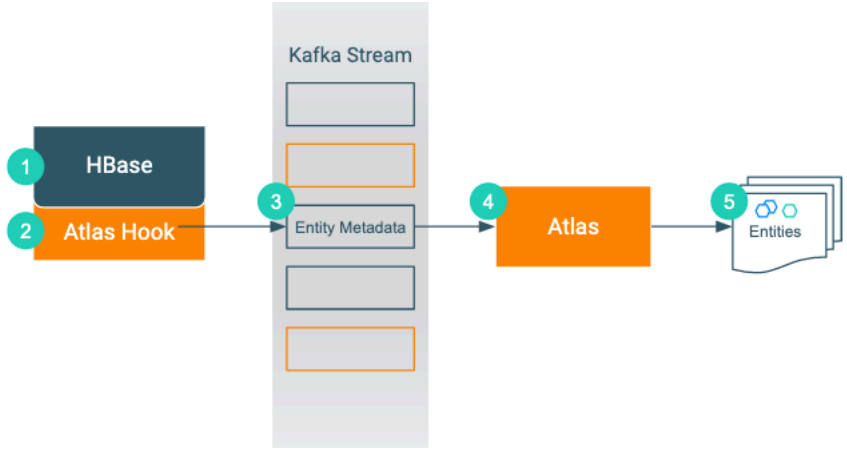
Atlas tracks the lifecycle of each Hive entity, including its creation, update, and deletion. User access and actions that affect the data content of the source asset are not included in the audit.



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:

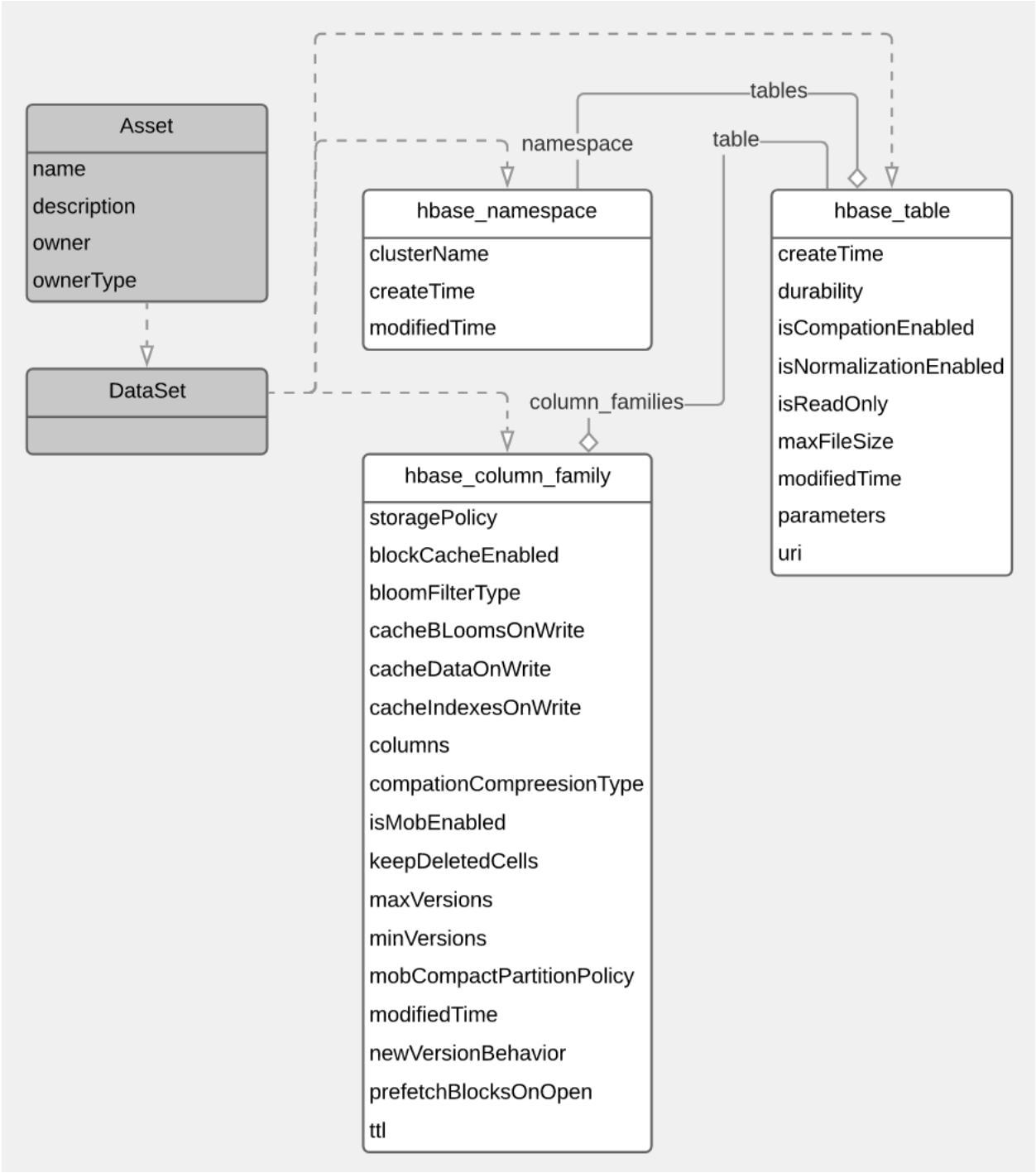
- Truncate table
- Put (cell value)]
- Disable/enable table

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, LZO, LZ4, GZ. The default value is SNAPPY. **Note:** Atlas is only certified to use Snappy and Gzip for HBase table compaction.
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

Changing the HBase column family compression type

Learn how to configure compression types. Changing the default Snappy algorithm to Gzip might be required depending on your use case and operational requirements.

About this task

Note: Atlas is only certified to use Snappy and Gzip for HBase table compaction.

Procedure

1. Add the atlas.graph.storage.hbase.compression-algorithm property in Atlas Server Advanced Configuration Snippet for conf/atlas-application.properties.
2. Enter the desired compression type in the property.
   atlas.graph.storage.hbase.compression-algorithm=*GZ*
3. Select the Master-(Active) Role Type for the HBase Instance from Cloudera Manager. Then, log in by using SSH in the terminal.
4. Run kinit -kt ****PATH TO HBASE.KEYTAB**** hbase to authenticate into HBase.
5. Enter the HBase shell.
   a) Run list to see the available HBase tables.
6. Run 'desc 'atlas_janus'' to see all column descriptions.
7. Change the compression type for each required table:

```
alter 'atlas_janus', {NAME => 'e' , COMPRESSION => ''GZ''}
alter 'atlas_janus', {NAME => 'f' , COMPRESSION => 'GZ'}
alter 'atlas_janus', {NAME => 'g' , COMPRESSION => 'GZ'}
alter 'atlas_janus', {NAME => 'h' , COMPRESSION => 'GZ'}
alter 'atlas_janus', {NAME => 'i' , COMPRESSION => 'GZ'}
alter 'atlas_janus', {NAME => 'l' , COMPRESSION => 'GZ'}
alter 'atlas_janus', {NAME => 'm' , COMPRESSION => 'GZ'}
alter 'atlas_janus', {NAME => 's' , COMPRESSION => 'GZ'}
alter 'atlas_janus', {NAME => 't' , COMPRESSION => 'GZ'}
```

8. Run major_compact 'atlas_janus' only in HBase Shell to start compacting your tables.

Results

Your HBase column family uses the newly set compression type.

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.

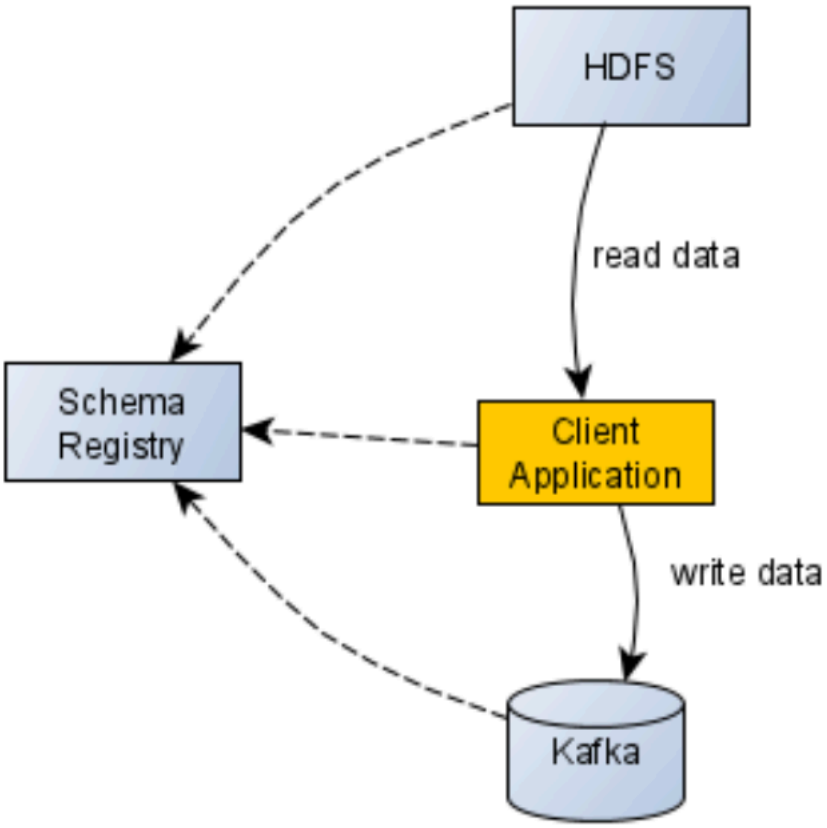
Atlas tracks the lifecycle of each HBase entity, including its creation, update, and deletion. User access and actions that affect the data content of the source asset are not included in the audit.

Schema Registry metadata collection

Integrating Schema Registry with Atlas will enable you to persist and view the schemas in Atlas.

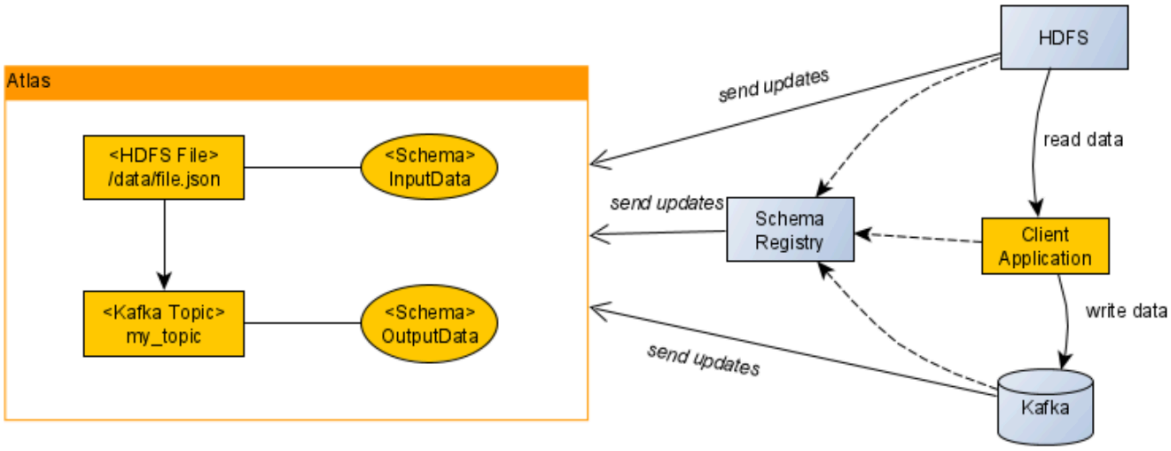
Atlas serves as a metadata catalogue that shows relationships between entities. Additionally, when Kafka is also integrated with Atlas you can connect Kafka topics and schemas.

As an example, an ETL job which reads encoded data from Hadoop, performs some operations on it and writes it into Kafka. In order to read the encoded data, the ETL job needs access to Schema Registry to obtain the schema. To write the data into Kafka, it needs to be encoded again, perhaps with a different schema. A relationship graph for this example is detailed in the following diagram.



In a real-world environment, however, we can have many components communicating with each other and it might become hard to track the relations and data flows.

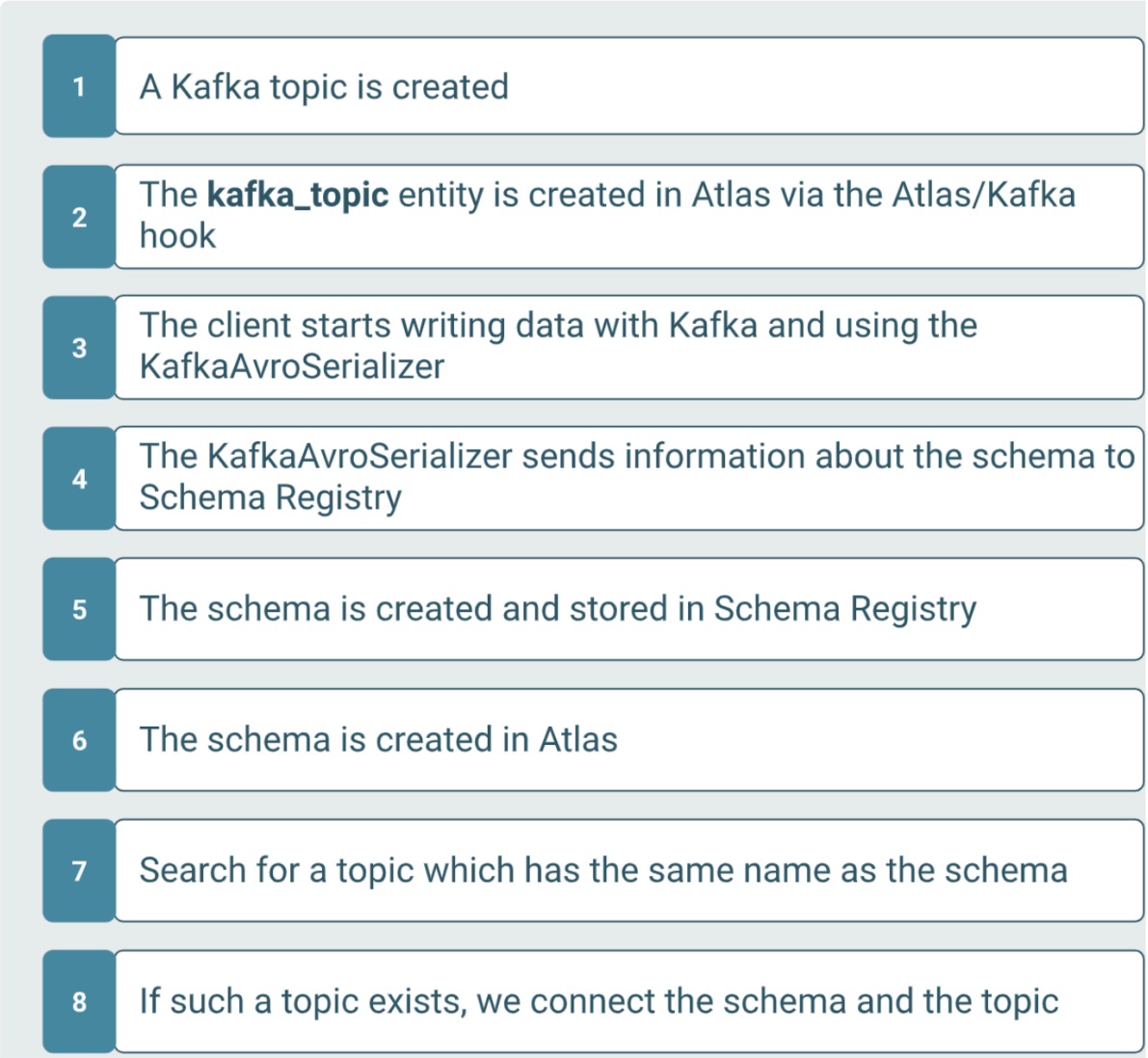
Atlas in its core is a metadata catalog which uses a graph database to store entities and its relationships, which lets users track the data flows as discussed earlier in this section. The graph would be built automatically for the end-user. The corresponding services (Hadoop, Schema Registry, Kafka, Hive (HiveServer2 and HMS), Impala, Spark, HBase, Sqoop, Storm, NiFi) send updates to Atlas and build the graph on the go.



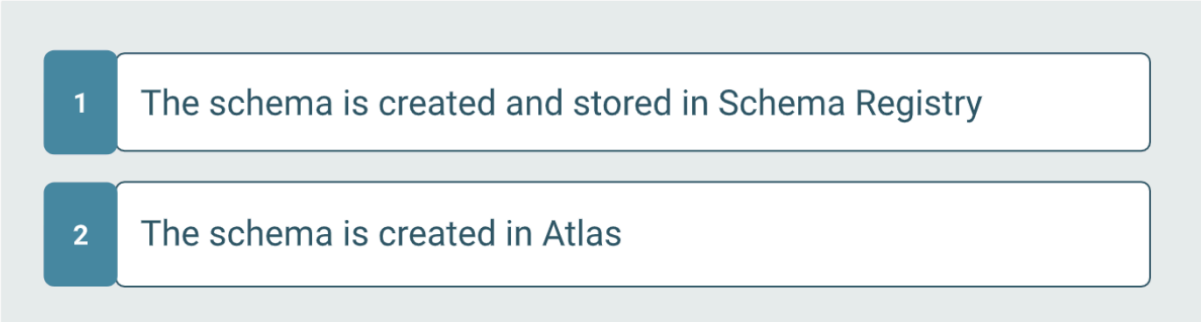
Note: The implementation of how those updates are sent varies between services.

There are two scenarios with which this integration works:

- When Kafka topic is created:



- When Schema data is created in Atlas:



Configuring Atlas and Schema Registry

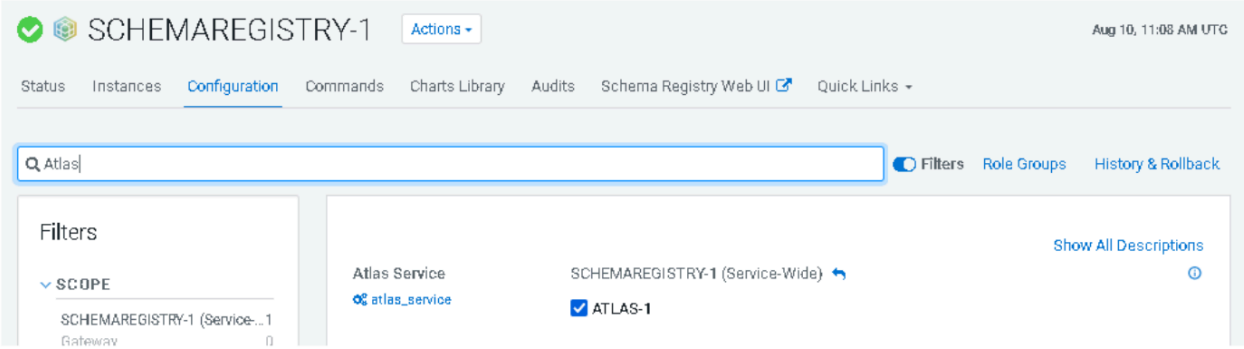
You must configure the settings in your Cloudera Manager instance

Procedure

1. Navigate to the Configuration tab of the Schema Registry in your Cloudera Manager instance and enter “Atlas”.
2. Check the Atlas Service checkbox.
3. Click Save Changes and restart Schema Registry.

What to do next

Once Apache Atlas integration is enabled, you can create new schemas which show up in Atlas. Note that the process which mirrors schemas from Schema Registry to Atlas is asynchronous and runs once every minute. A schema which is created does not show up immediately, but only after one minute.



Note: If a Kafka topic exists with the same name as the schema, then a relationship is created between them in Atlas. If no such topic exists, only the schema is created.

Schema Registry actions that produce Atlas entities

Once the Schema Registry application is integrated with Atlas, entities are created in Atlas to represent the metadata.

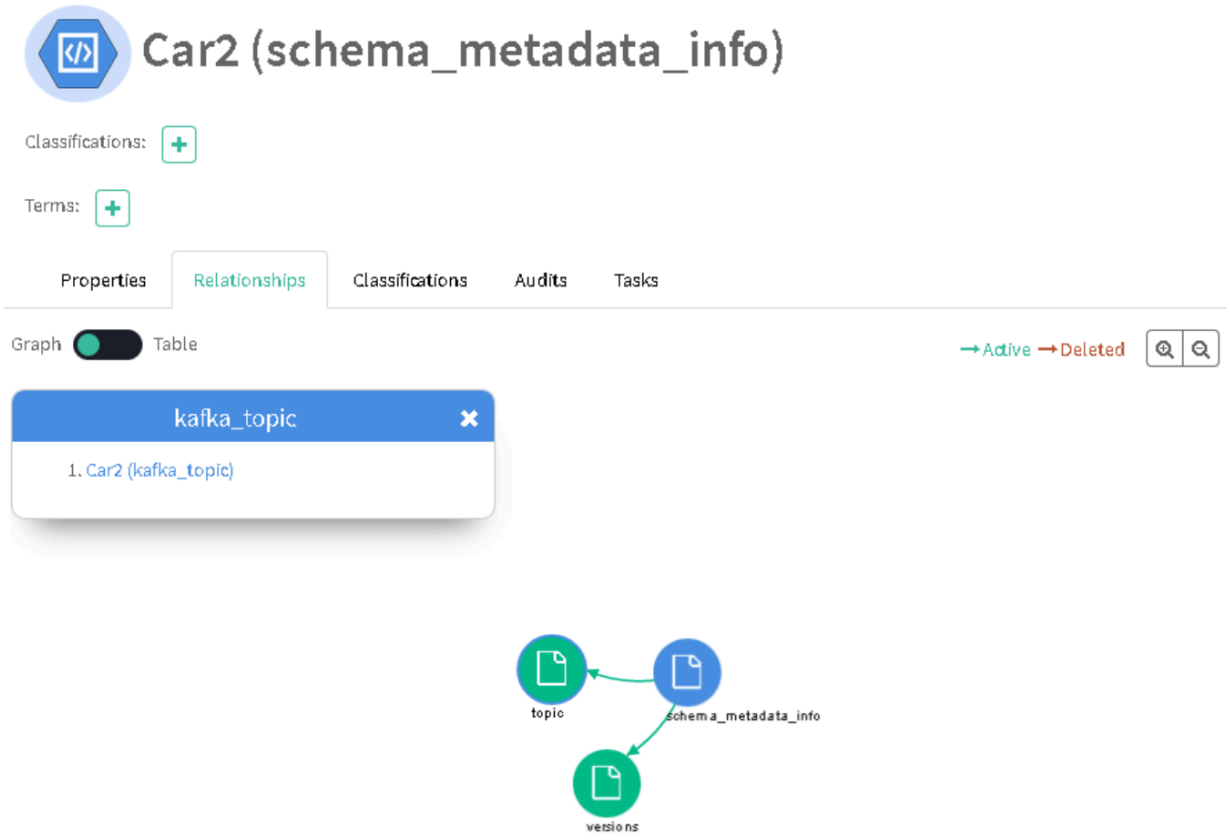
The following table shows a summary of the entities created in Atlas to represent the metadata.

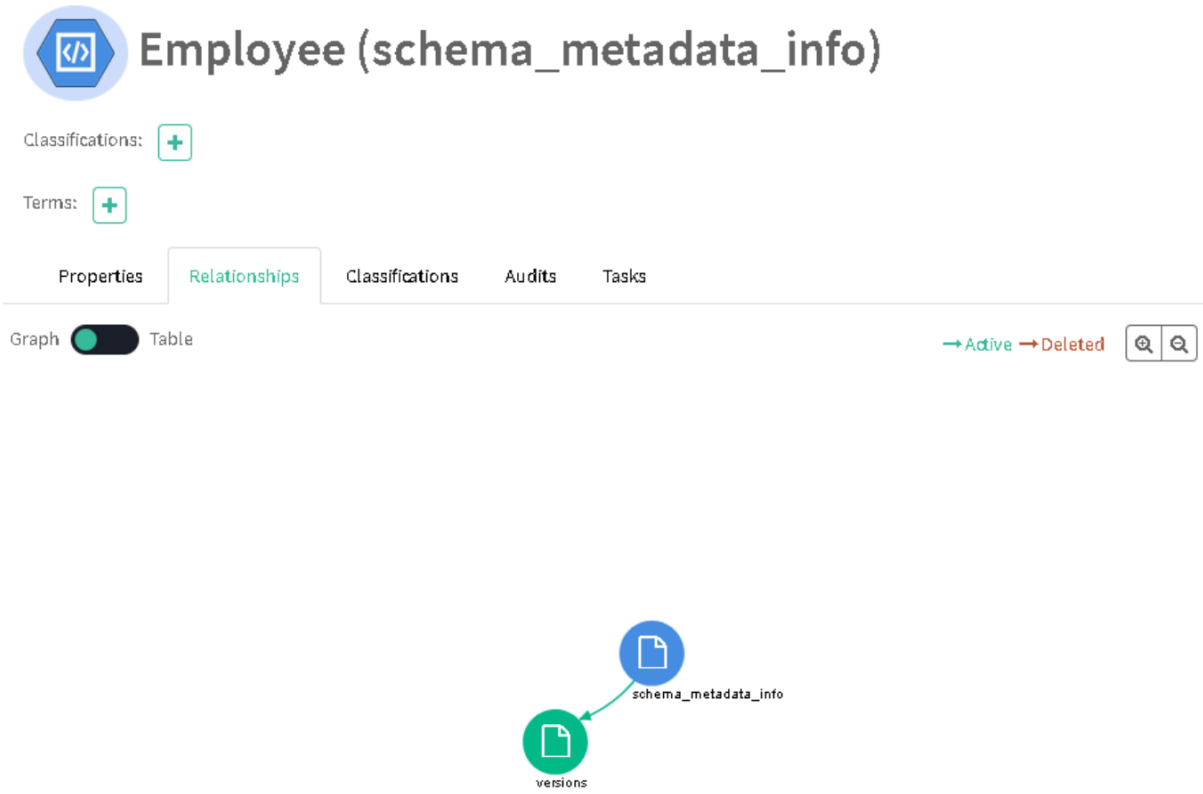
Atlas Entity	Description
Schema metadata	Contains the properties of a schema.
Schema version	A schema can have one or more versions. They are all connected to one metadata.
Kafka topic	If a Kafka topic exists and the Atlas-Kafka integration is enabled, then a topic entity is created. When a schema is created with the same name as the topic then a relationship is created between them.

Schema replationships

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.

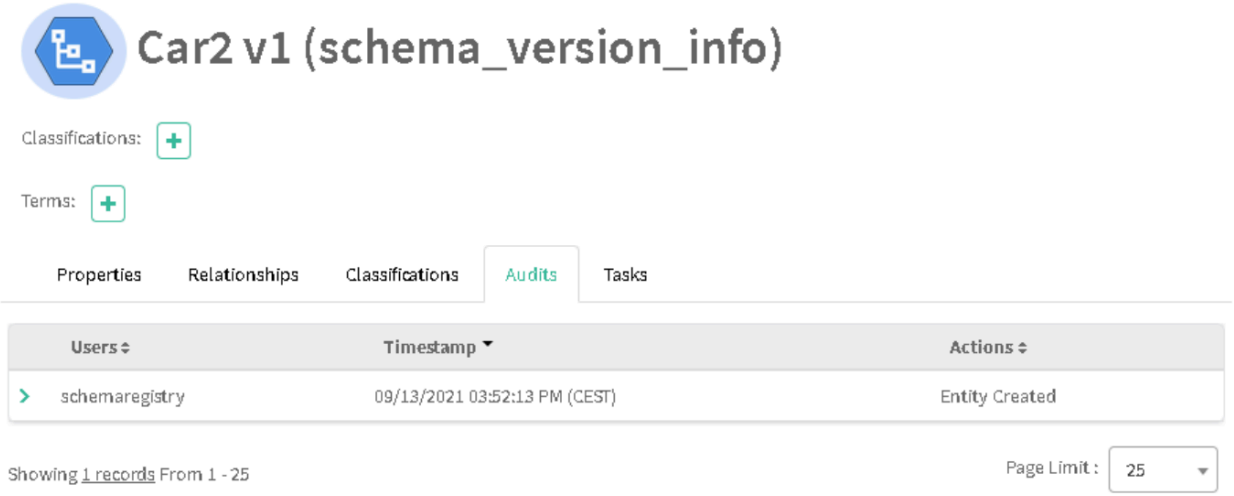

Car2 (schema_metadata_info)
Classifications:
Terms:
Properties
Relationships
Classifications
Audits
Tasks
Graph
Table
Active
Deleted
kafka_topic
1. Car2 (kafka_topic)
topic
schema_metadata_info
versions


Employee (schema_metadata_info)
Classifications:
Terms:
Properties
Relationships
Classifications
Audits
Tasks
Graph
Table
Active
Deleted
schema_metadata_info
versions

Schema Registry audit entries

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

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



Troubleshooting Schema Registry

What do you do if you are not able to view Atlas metadata from Schema Registry?

Mirroring schemas from Schema Registry to Atlas is done by an asynchronous process. When a schema or schema version is created, the information about this event is generally stored in the database table atlas_events. When an event has been processed, its status will be updated to either success or failure.

Mirroring a schema to Atlas can fail for multiple reasons but not limited to the following:

- Atlas may be down
- Network connection failure

If you have analyzed the problem and can fix it by “replaying” the event, you must update the table and set the failure flag to false. In this case, the event will be picked up again and processed once again.

The SQL format to perform the above:

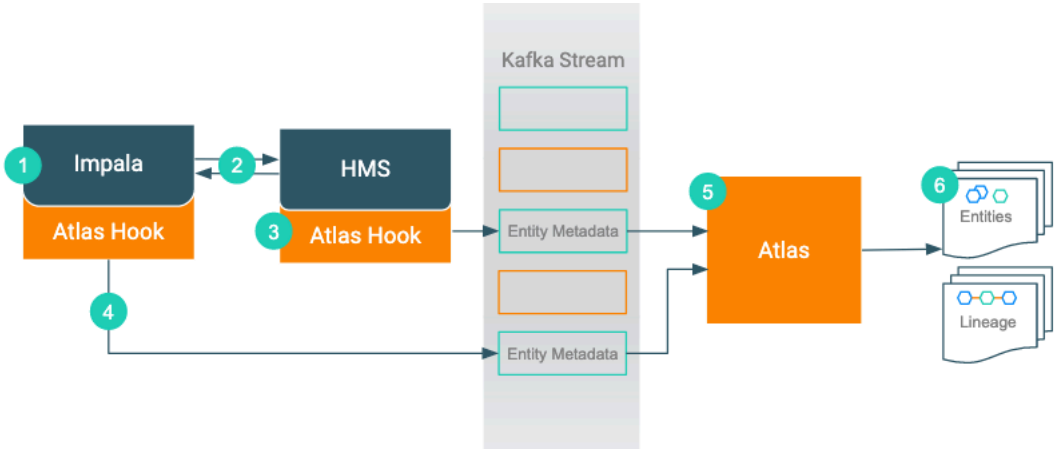
update atlas_events set processed=0, failed=0 where id = <event id>

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_storagedesc, hive_db hive_table_ddl	hive_table_db, hive_table_columns, hive_table_partitionkeys, hive_table_storagedesc, 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

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
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_storagedesc, 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:

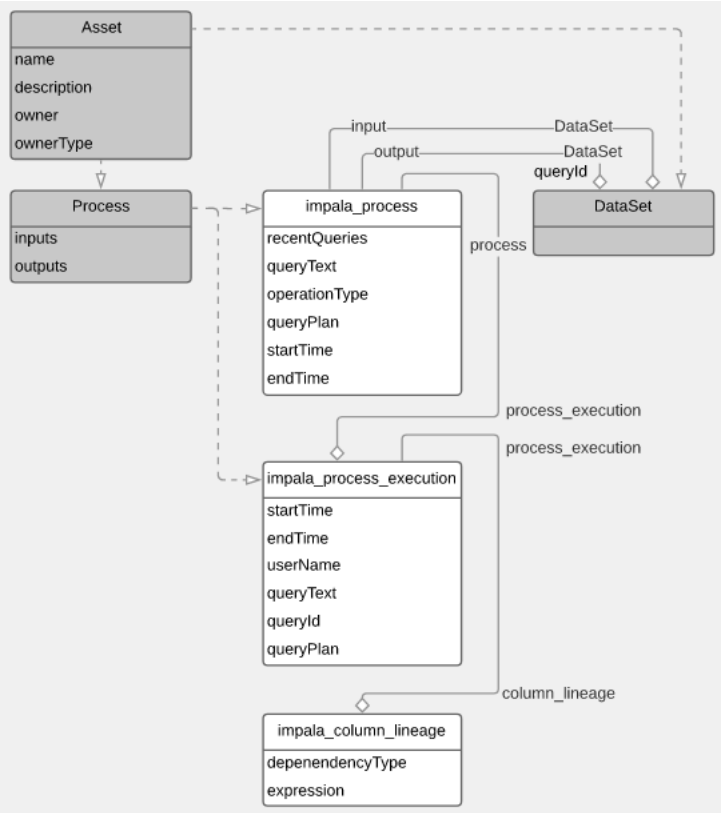
- LOAD DATA INPATH
- CREATE TABLE (table metadata produced by HMS)
- ALTER VIEW (table metadata produced by HMS)
- SELECT or other queries that don’t produce output

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 yyyymmddhhmmss>_<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.

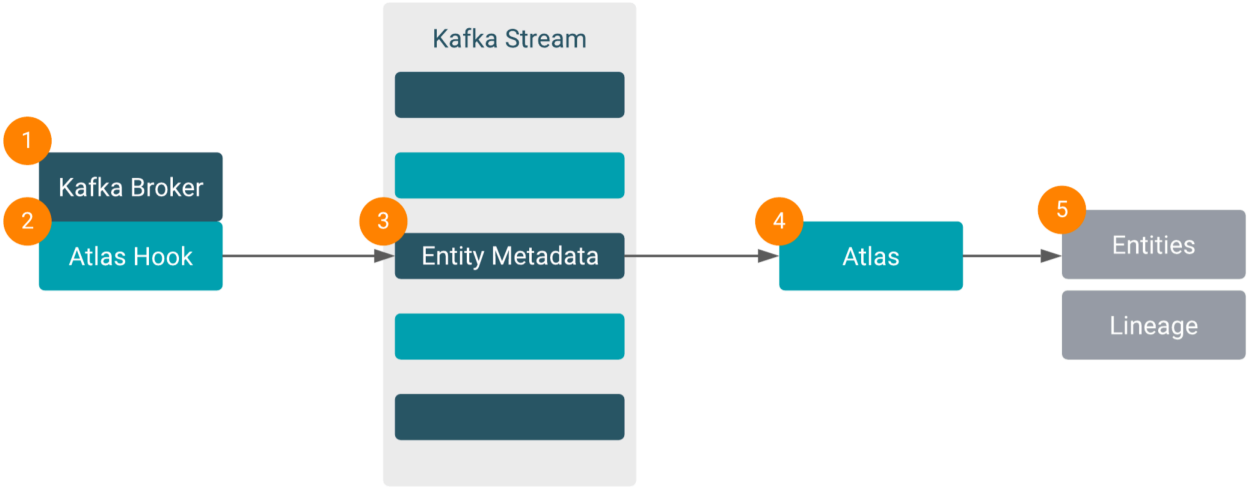
Atlas tracks the lifecycle of each Impala entity, including its creation, update, and deletion.

Kafka metadata collection

Atlas can collect metadata from Kafka using the concept called metadata namespace.

When the Kafka cluster is configured to audit using Atlas, the Kafka brokers start notifying Atlas about metadata changes in the Kafka cluster. Clients do not have to be integrated with Atlas.

Note: Atlas auditing of Kafka topics is not supported for Kafka consumers using the version lower than 2.5.



Atlas is used to track the metadata of multiple clusters. Metadata namespace in Atlas is used to categorise entities into groups. To avoid entity collision between clusters, entities belonging to different clusters can be grouped together, into separate metadata namespaces.

An Atlas hook runs in each Kafka instance. This hook transfers the metadata of Kafka assets to Atlas. The Kafka-Atlas auditing involves two metadata namespaces, one each for topics and clients.

The topic entities are created in the topic metadata namespace. Producer, Consumer and ConsumerGroup entities are created in the client metadata namespace.

For a simple use case, as a default configuration, the auditor can be configured to use the same namespace for both types.

Client metadata namespace can be used to insert entities representing an application spanning multiple Kafka clusters in the same namespace.

For example, if an application is connecting to a couple of Kafka clusters, and they have separate namespaces, the client entities are created twice, because each Kafka cluster will create them in its own client namespace. If Kafka clusters use the same client namespace, they will create and update the same client entity, so the application will be represented by a single producer and/or consumer entity.

Important: If Atlas auditing is enabled for an existing Kafka cluster with multiple topics that are not pushed into Atlas, the topics can be imported using the import-kafka tool manually or using the Import Kafka Topics Into Atlas action available in Cloudera Manager. Next, the auditor within the Kafka brokers retains the metadata of the Kafka cluster in sync with Atlas.

Related Information
Configuring the Atlas hook in Kafka
Importing Kafka entities into Atlas

Kafka actions that produce Atlas entities

After the Kafka application is integrated with Atlas, entities are created in Atlas to represent the metadata.

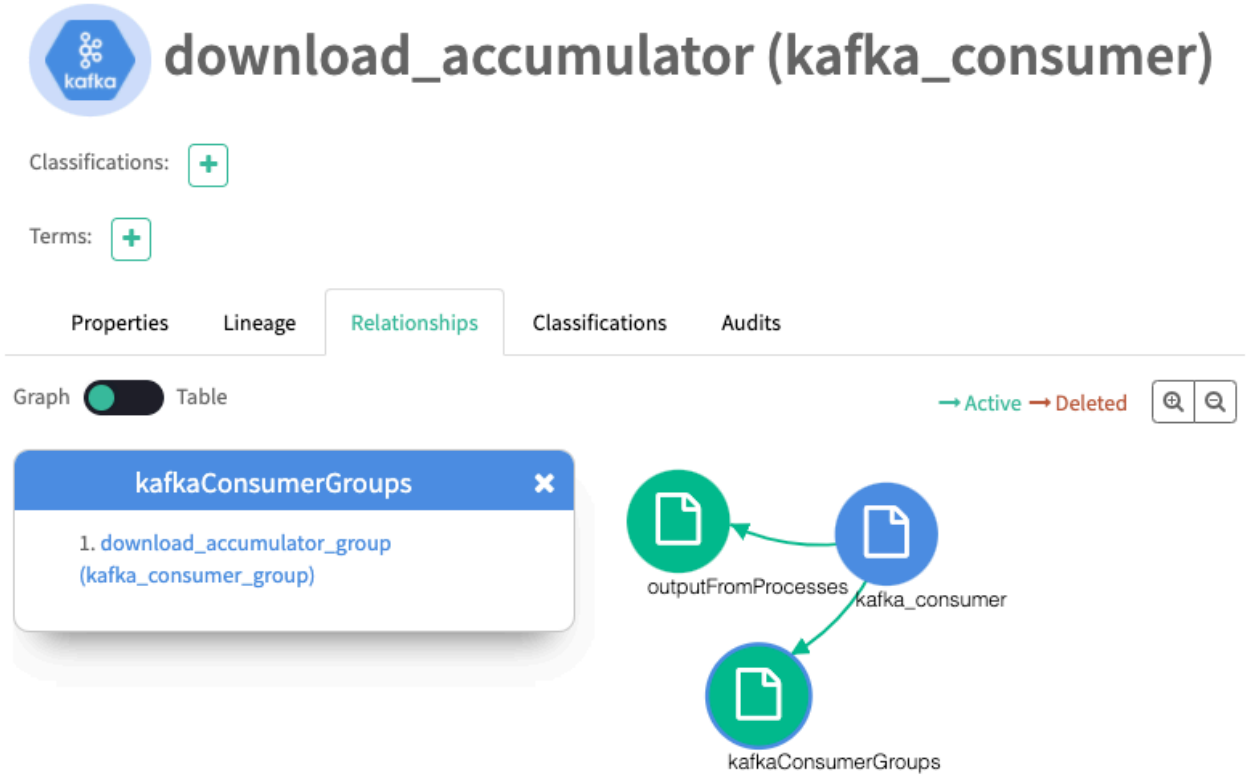
The following table show a summary of the entities created in Atlas to represent the metadata.

Atlas Entity	Description
Topic entities	Represents a Kafka topic
Producer entity	Represents a Kafka producer client. Producers are identified by their client.id.
Consumer entity	Represents a Kafka consumer client. Consumers are identified by their client.id.
ConsumerGroup entity	Represents a consumer group. Consumer groups are identified by their consumer group ID.
Producer lineage entity	Artificial process type entity to support the lineage of producers.
Consumer lineage entity	Artificial process type entity to support the lineage of consumers

Kafka relationships

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

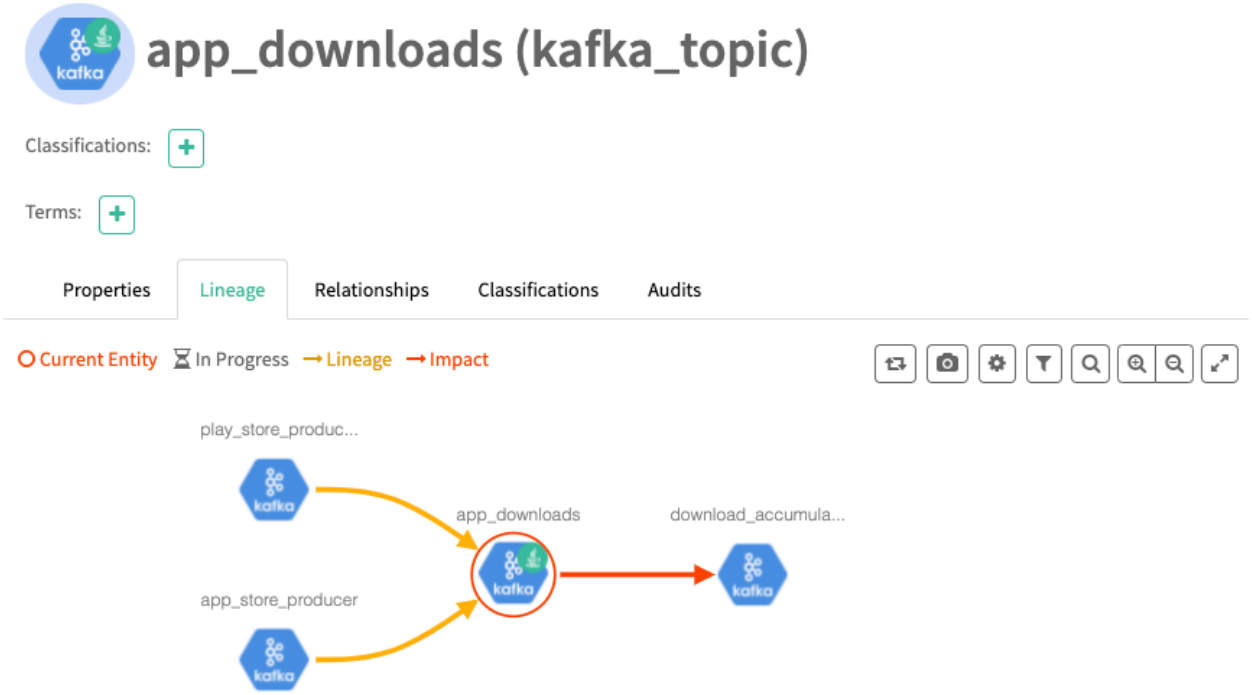
The Relationship tab shows the relationships that exist for an entity. Use this view to navigate among related entities.



Kafka lineage

Using the Lineage tab, Atlas collects metadata from Kafka to represent the lineage among data assets.

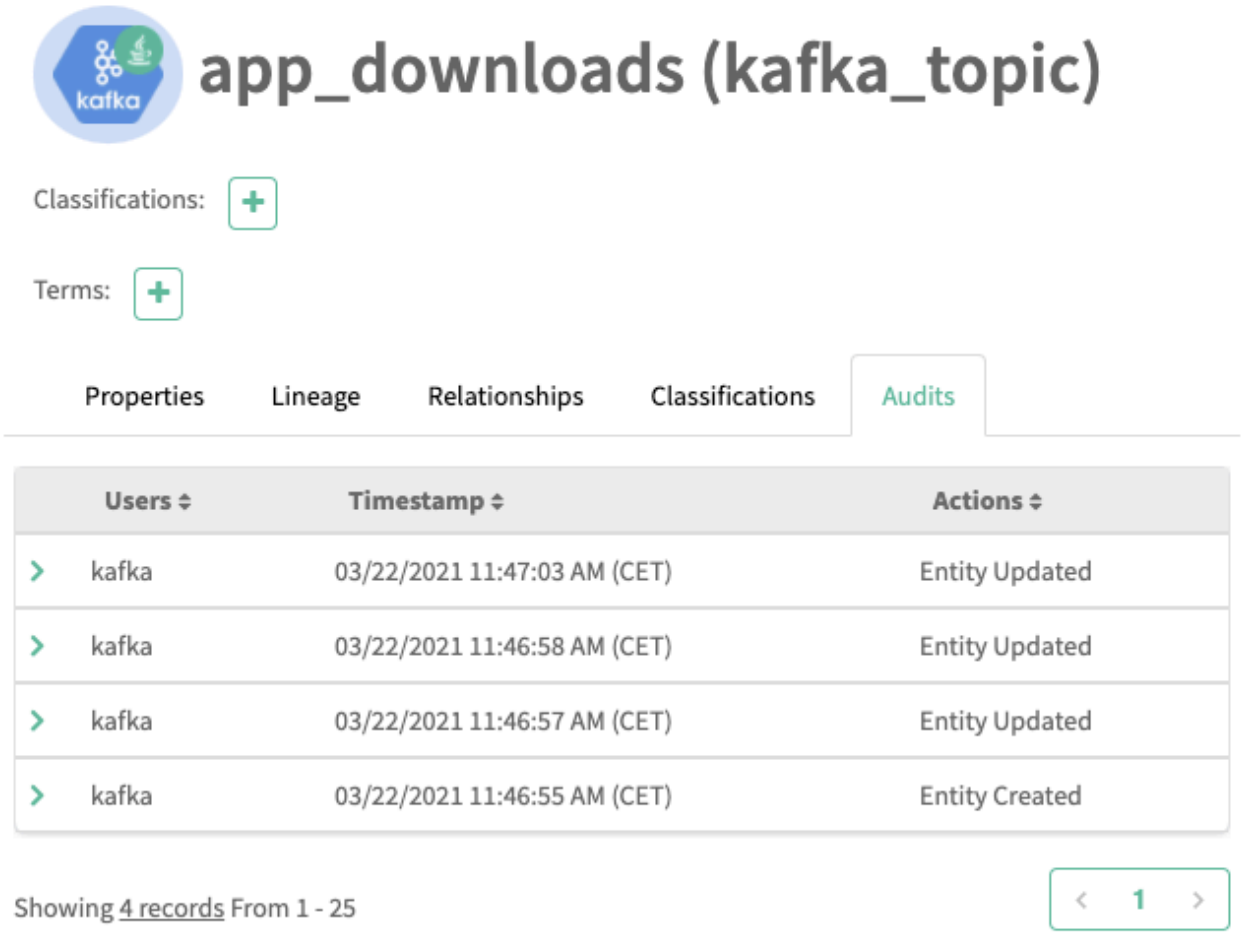
The Atlas lineage graph shows the input and output processes that the current entity participated in, specifically those relationships modeled as “inputToProcesses” and “outputFromProcesses.” Entities are included if they were inputs to processes that lead to the current entity or they are output from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



Kafka audit entries

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

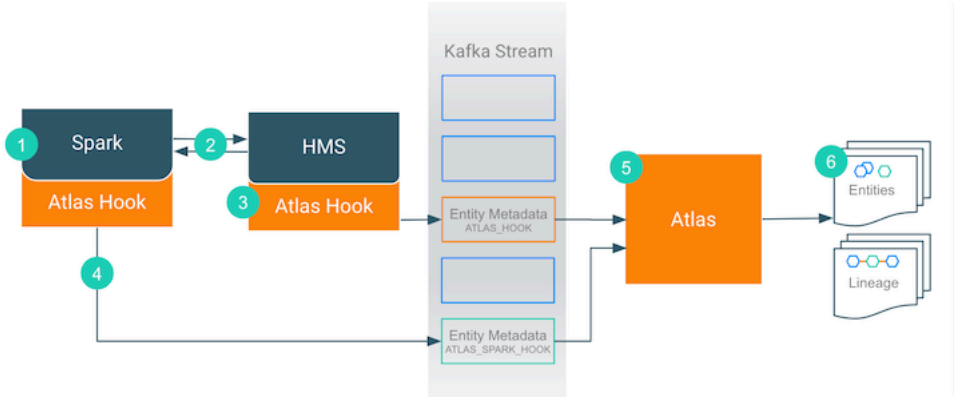
Atlas tracks the lifecycle of each Kafka entity, including its creation, update, and deletion. User access and actions that affect the data content of the source asset are not included in the audit.



Spark 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 and Spark 3.

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

Spark jobs create Spark application and 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_application, spark_column_lineage, spark_process, hive_table, hive_column, hive_storagedesc
CREATE VIEW AS SELECT,	spark_application, spark_process, hive_table, hive_column, hive_storagedesc
INSERT INTO (SELECT), LOAD DATA [LOCAL] INPATH	spark_application, spark_process

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

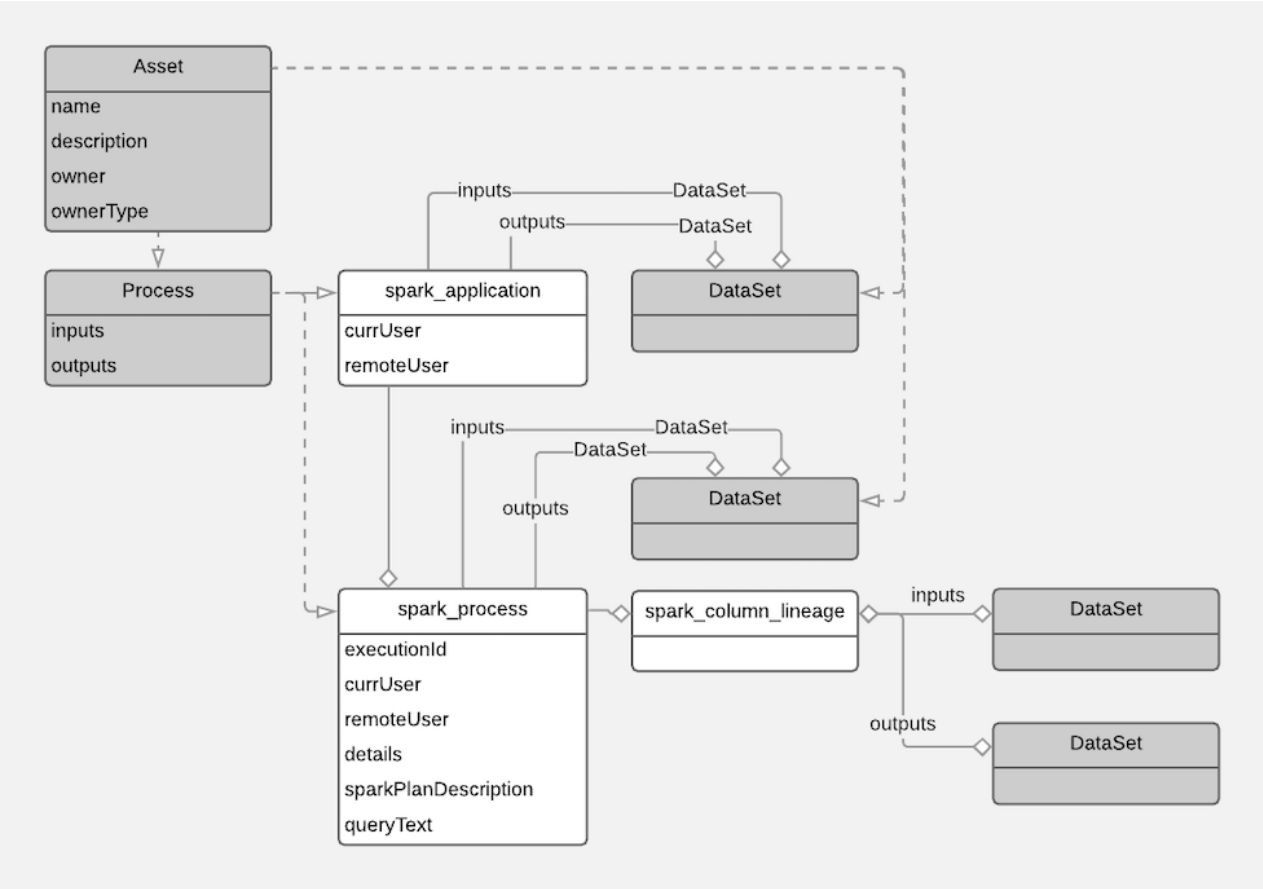
- LOAD DATA INPATH (when not coming from a local file source)
- CREATE TABLE (hive_table metadata produced by HMS)
- ALTER VIEW (hive_table metadata produced by HMS)
- SELECT or other queries that don’t change table metadata

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 Application

Identifier	Example content
typeName	spark_application
guid	System generated ID. This value is used to identify the entity in the Atlas Dashboard URL.
qualifiedName	<Spark application ID>
name	Spark Job + <Spark application ID>
description	Metadata from Spark. Reserved for future use.
displayName	Reserved for future use.
owner	Metadata from Spark. Reserved for future use.
currentUser	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.
userDescription	Metadata from Spark. Reserved for future use.
replicatedFrom	Reserved for future use.
replicatedTo	Reserved for future use.
Relationship: inputs	Reserved for future use.
Relationship: outputs	Reserved for future use.
Relationship: processes	List of Spark process entities created as part of the processing accomplished in this Spark job.

Spark Process

Identifier	Example content
typeName	spark_process
guid	System generated ID. This value is used to identify the entity in the Atlas Dashboard URL.
qualifiedName	*APPLICATION-ID*-execution-*N* where *N* is a sequential integer assigned by the Spark engine and the application ID is for the parent Spark job.
name	execution-*N* where *N* is a sequential integer assigned by the Spark engine. The number is unique only for the job, so it is possible to have Spark processes with duplicate names in Atlas.
description	Metadata from Spark. Reserved for future use.
owner	Metadata from Spark. Reserved for future use.
details	Metadata from Spark describing the logical plan.
displayName	Reserved for future use.
executionId	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.
queryText	Metadata from Spark. Reserved for future use.
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.
replicatedFrom	Reserved for future use.
replicatedTo	Reserved for future use.
userDescription	Metadata from Spark. Reserved for future use.
Relationship: inputs	List of the input tables or views, including each entity's type name and the qualified name.
Relationship: outputs	List of the output objects, including each entity's type name and the qualified name.
Relationship: application	The Spark application entity that describes the Spark job in which this process was created.

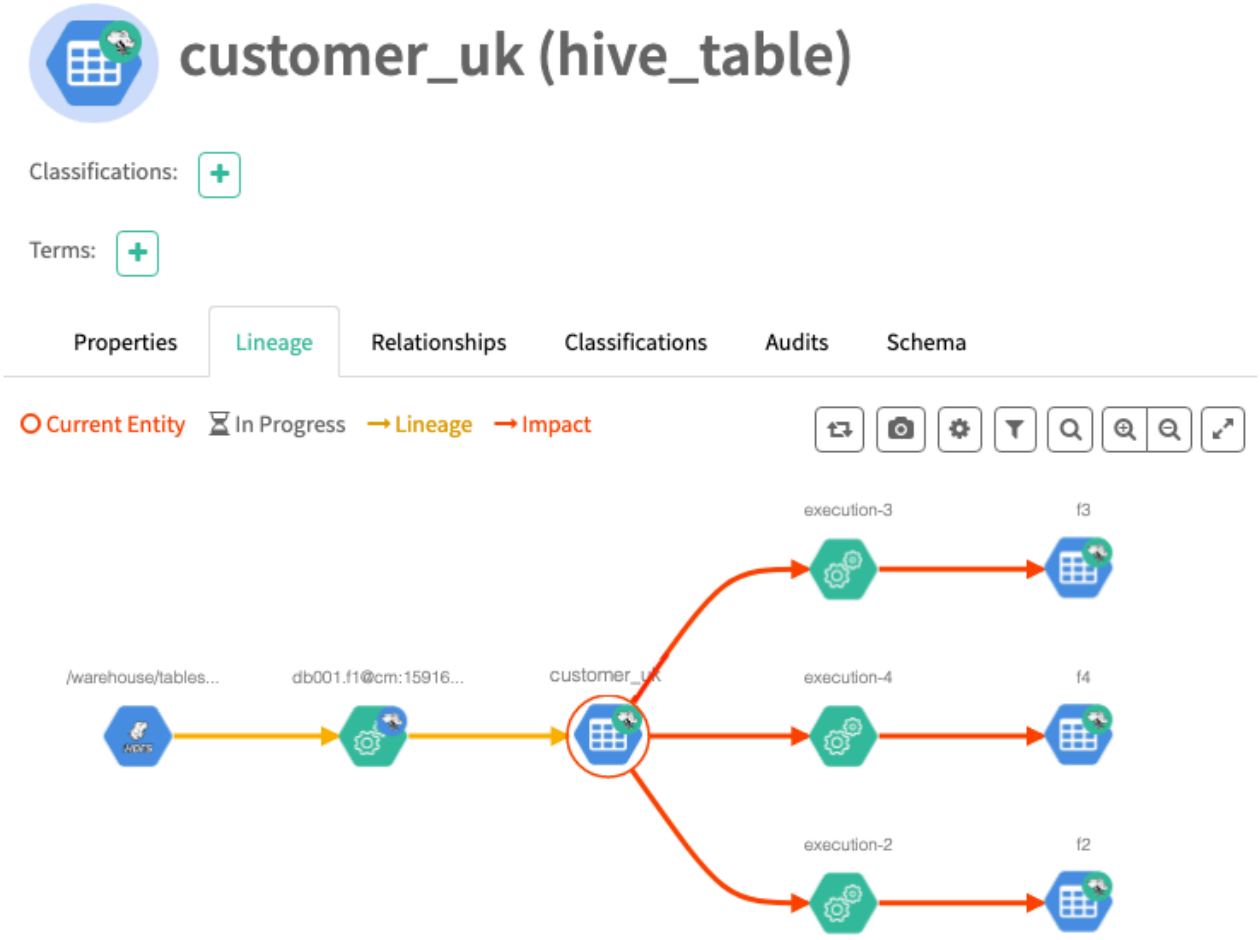
Spark Column Lineage

Identifier	Example Content
typeName	spark_column_lineage
guid	System generated ID. This value is used to identify the entity in the Atlas Dashboard URL.
qualifiedName	*<OUTPUT_TABLE_NAME>-<TIMESTAMP>-<OUTPUT_COLUMN_NAME>*
name	Same as qualifiedName.
Relationship: Process	Name of the spark_process entity that produced this lineage. spark_process_column_lineages
Relationship: inputs	List of the input columns, including each entity's type name and the qualified name.
Relationship: outputs	Output column, including each entity type name and the qualified name.

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. In the context of Spark, a Spark job is modeled as a spark_application entity. Each application entity includes relationships to one or more processes that were executed in the job. The spark_process entities are automatically named "execution-*N*" where *N* is an integer incremented sequentially.



It is possible to have two spark process entities with the same name in a lineage graph; be sure to check the qualified name to make sure you are looking at the appropriate process.

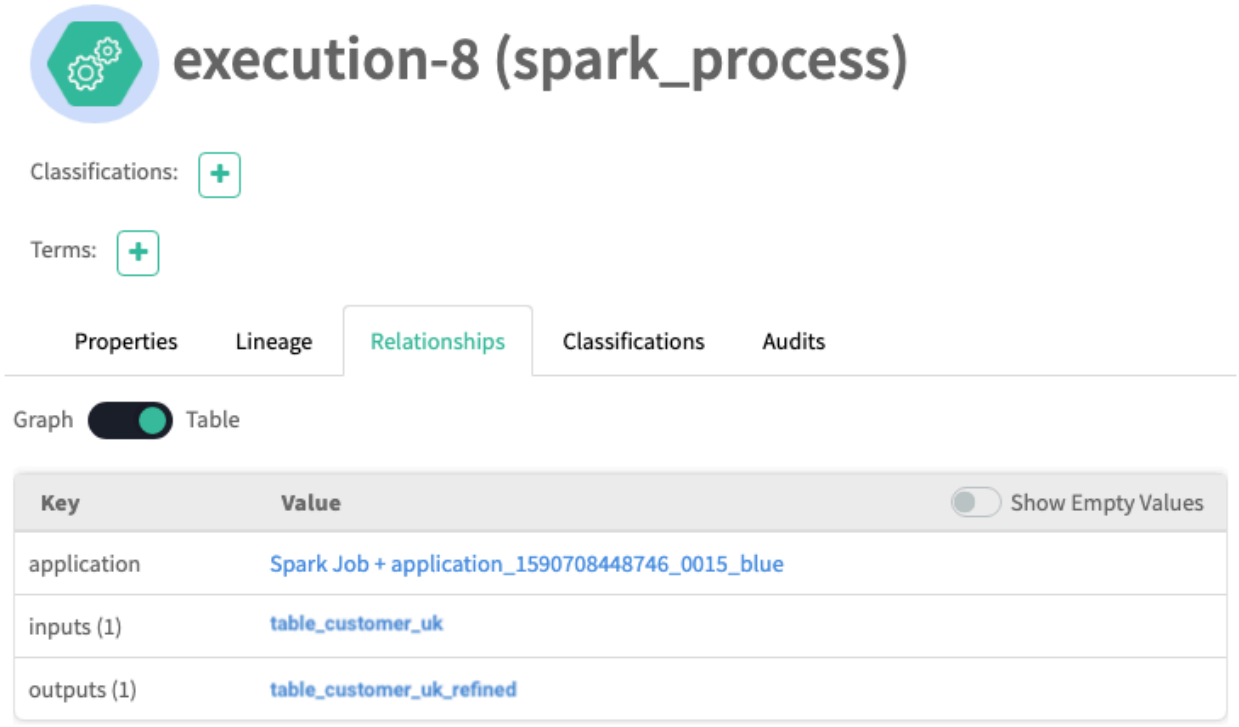
Related Information

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.



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. Note that if you change the name of an application in Atlas, the change will cause each related Spark process entity to be updated. User access and actions that affect the data content of the source asset are not included in the audit.

Spark Connector configuration in Apache Atlas

Learn to configure the Spark Atlas Connector so that Spark jobs can run even when Kafka brokers are down. This ensures that your job submissions do not fail.

Note: The Spark Atlas Connector is available in Cloudera Public Cloud from version 7.2.18 and in CDS Powered by Apache Spark from version 3.4.1 from 7.2.18.

Before you begin the configuration, you must enable the following properties in Cloudera Manager:

Atlas Service:

1. Log in to Cloudera Manager.
2. Select the Spark service.
3. Select the Configurations tab.
4. Search for and select the Atlas Service (atlas_service) parameter.
5. Click Save Changes.

Spark Lineage:

1. Log in to Cloudera Manager.
2. Select the Spark service.
3. Select the Configurations tab.

4. Search for and select the Spark Lineage (spark.lineage.enabled) parameter.
5. Click Save Changes.

Using the spark.lineage.kafka.fault-tolerant.timeout.ms parameter

If all Kafka brokers are down, the deploy mode is cluster mode without the keytab/principal. Spark Atlas lineage is also enabled which does not allow a delegation token from the Kafka broker. Therefore, the job submission fails. As a result, --deploy-mode is set but --principal PRINCIPAL and --keytab KEYTAB are not set. Use spark.lineage.kafka.fault-tolerant.timeout.ms parameter to resolve this case.

The default timeout value for the Kafka delegation token creation is 0. This means that the fault tolerant mode is disabled. If the value is greater than 0, the fault tolerant mode is enabled with the configured timeout value.

See the example PySpark code below:

```
from pyspark.sql import SparkSession
import time
import sys
id_string = str(time.time()).split(".")[0]
spark = SparkSession.builder.appName("Python CTAS example").getOr
Create()
spark.sql("CREATE TABLE parquet_spark1_" + id_string +" ( id bi
gint, data string, category string) using parquet")
spark.sql("CREATE TABLE parquet_spark2_" + id_string +" USING pa
rquet AS SELECT * from parquet_spark1_" + id_string)
```

Example commands:

```
spark-submit --conf spark.lineage.kafka.fault-tolerant.timeout.m
s=8000 --deploy-mode cluster test.py
```

```
spark3-submit --conf spark.lineage.kafka.fault-tolerant.timeout.
ms=8000 --deploy-mode cluster test.py
```

Limitation:

This timeout will permit the job submission in cluster mode if all Kafka Brokers are down. There will be no Spark Atlas Connector lineage because we do not have Kafka delegation token which is required for the lineage creation.

Spark troubleshooting

What do you do if you do not see Apache Atlas metadata from Spark or face any related issues.

Spark runs an Atlas "hook" or plugin called Spark Atlas Connector (SAC) on every host where Spark 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 Cloudera, 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 On YARN service configuration page in Cloudera Manager to ensure that Spark is configured to send metadata to Atlas (Atlas Service property). 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.
- When customer is running a standard Spark structured streaming application in DataHub Data Engineering cluster, the Spark job might throw exceptions in the YARN App logs.

  For example:

  WARN clients.NetworkClient: [Producer clientId=producer-1] Error while fetching metadata with correlation id 1 : {ATLAS_HOOK=TOPIC_AUTHORIZATION_FAILED} ERROR hook.AtlasHook: Giving up after 3 failed attempts to send notification to Atlas

  The Spark application tries to publish the metadata to the Kafka topic named ATLAS_HOOK. At that stage it was failing because of the permission issue for the spark application user. For example: ‘csso_jonathan’.

  To overcome this permission issue, you must provide the Ranger permission for the application user 'csso_jonathan’ using following steps:

  Go to Ranger Admin> Kafka policies> Provide "publish" permission to "csso_jonathan” user for the following:
- ATLAS_HOOK
- ATLAS_ENTITIES
- ATLAS_SPARK_HOOK policies.

Later, re-run the Spark job and verify if the Atlas related errors are still visible in the application logs.