Cloudera Runtime 7.3.1

Atlas Audits

Date published: 2020-07-28
Date modified: 2024-12-10

https://docs.cloudera.com/

Legal Notice

Contents

Audit Operations....4
Atlas Type Definitions....6
Atlas Export and Import operations....7
Exporting data using Connected type....8
Atlas Server Operations....9
Audit enhancements....10
Examples of Audit Operations....12

Storage reduction for Atlas....16
Using audit aging....17
Enabling audit aging....17
Using default audit aging....17
Using Sweep out configurations....18
Using custom audit aging....19
Aging patterns....20
Audit aging reference configurations....20
Audit aging using REST API....21
Using custom audit filters....22
Supported operators....22
Rule configurations....23
Use cases and sample payloads....24

Audit Operations

As an Atlas administrator you can view the audit operations listed in the Audits tab in the Atlas UI. The audit data is captured implicitly by Atlas for various operations.

The audit operations include:

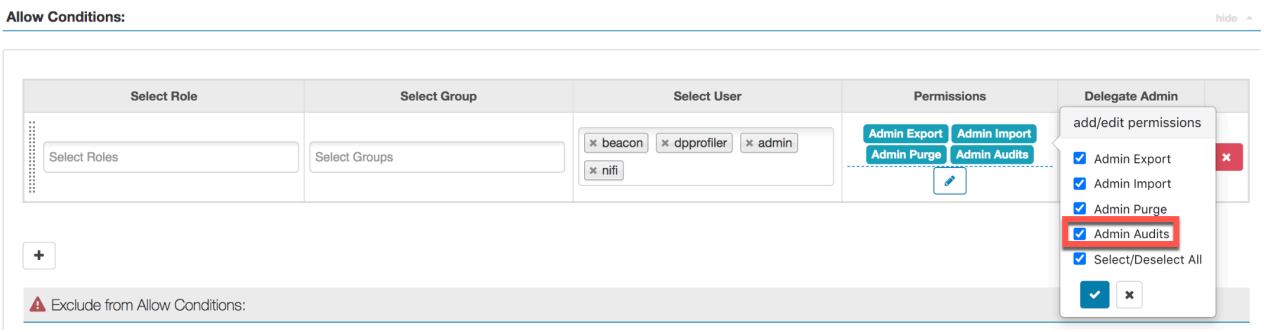
- Import
- Export
- Server start
- Server state active (In case of HA environment)
- Type Definition create
- Type Definition update
- Type Definition delete

The Type Definition can be of any type category listed:

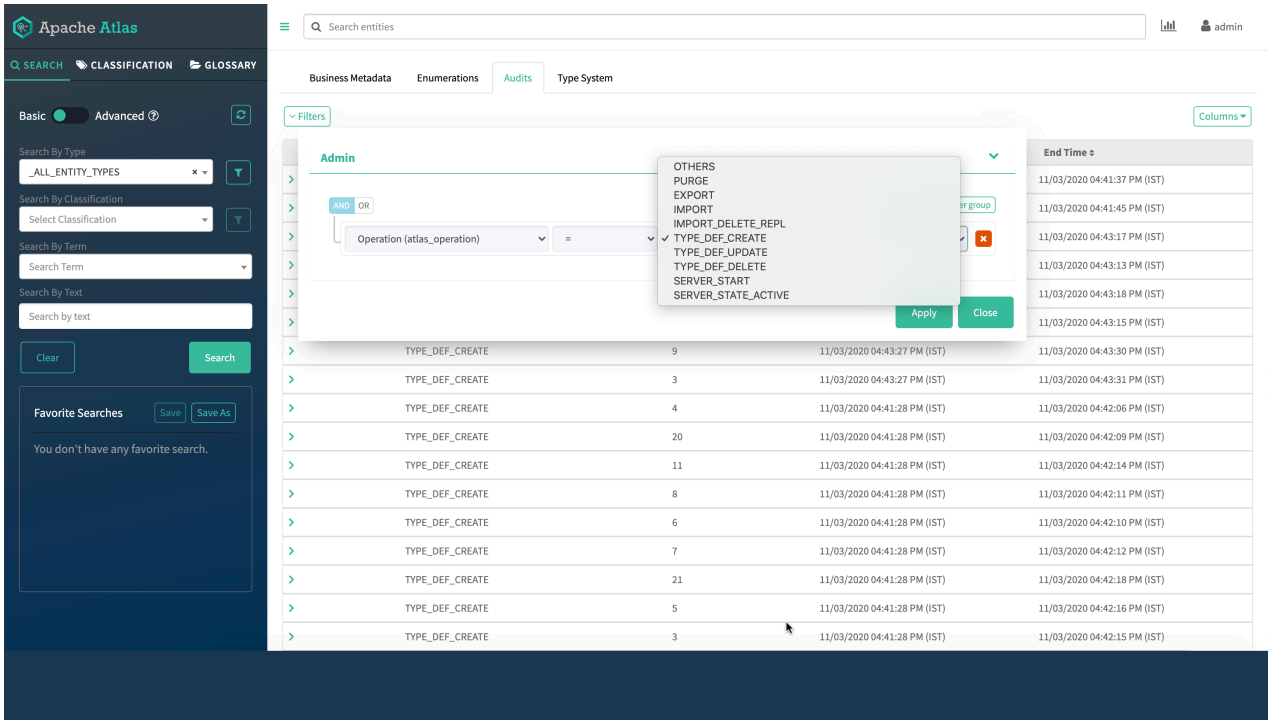
- Business Metadata
- Classification
- Enum
- Entity
- Relationship
- Struct

Audit is collected for every create, update and delete operations.

Note: A user or a group must have the Admin Audits Ranger permission to access audits and related information.



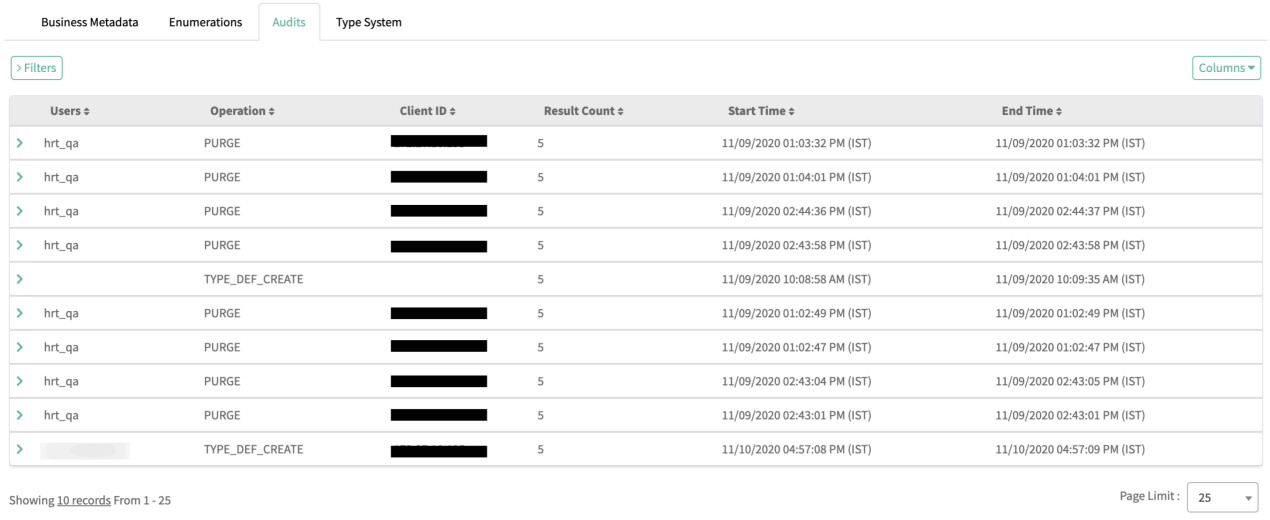
The JSON data is the payload which contains the actual data. You can submit the payload using the appropriate REST API tool.



An audit entry logs the total number of Type Definitions that are created for create, update, and delete operations. Type Definitions are categorized according to entity types, struct types, Enum types, relationships, classification, and Business Metadata. For every Type Definition, the JSON data is stored in the audit entry.

Each audit entry logs the following details:

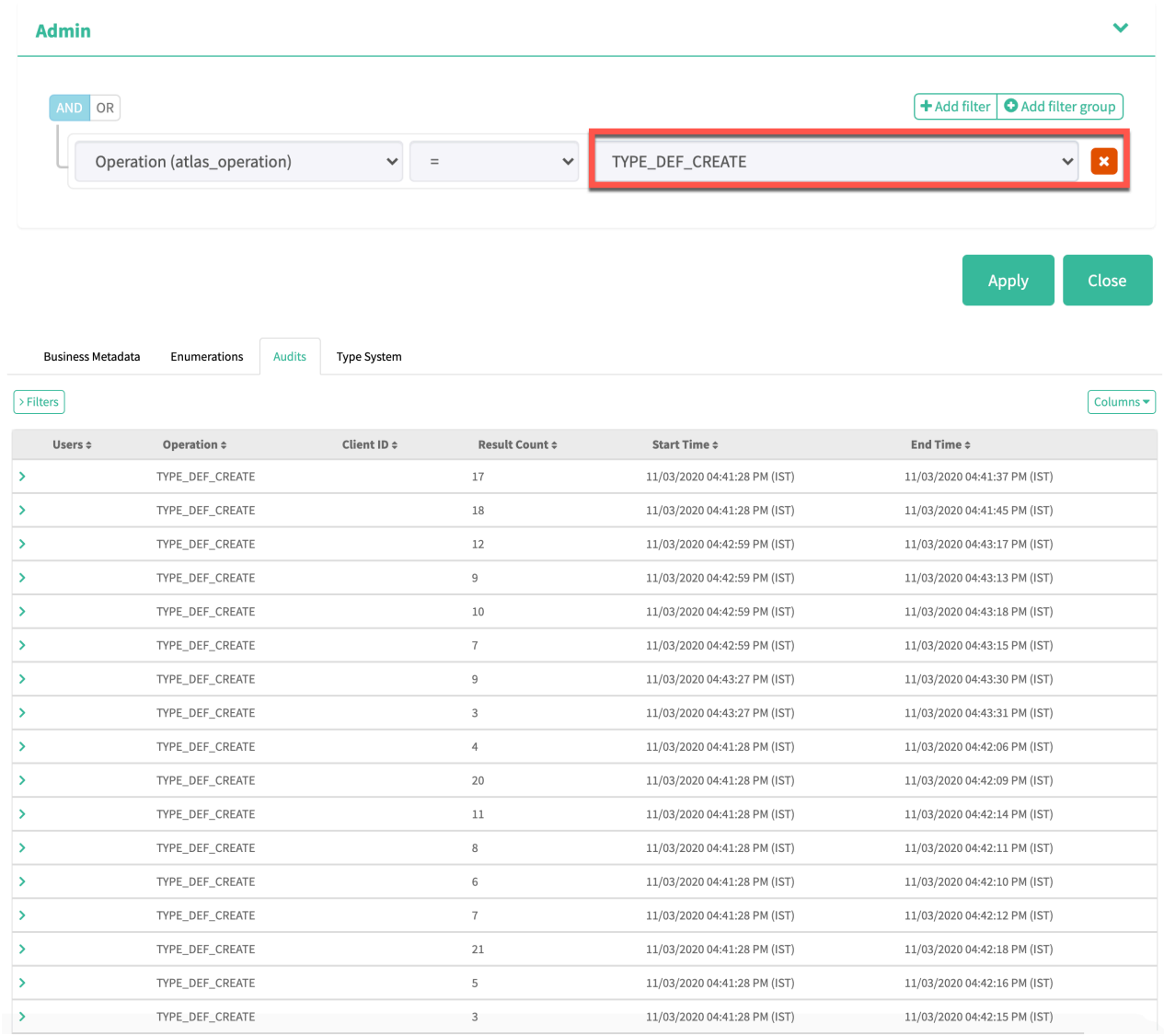
- Users - Indicates the user name of the user who performed the audit operation..
- Operation - Indicates an operation enum; can be used for searching audit entities as well
- Client ID - Indicates the IP address of the machine from which the request was generated.
- Result Count - Provides the total number of artifacts on which the operation was performed.
- Start Time - Indicates the actual time when the request was generated.
- End Time - Indicates the actual time when the requested operation was completed.
- Duration - Indicates the time taken by a request to complete the intended operation.



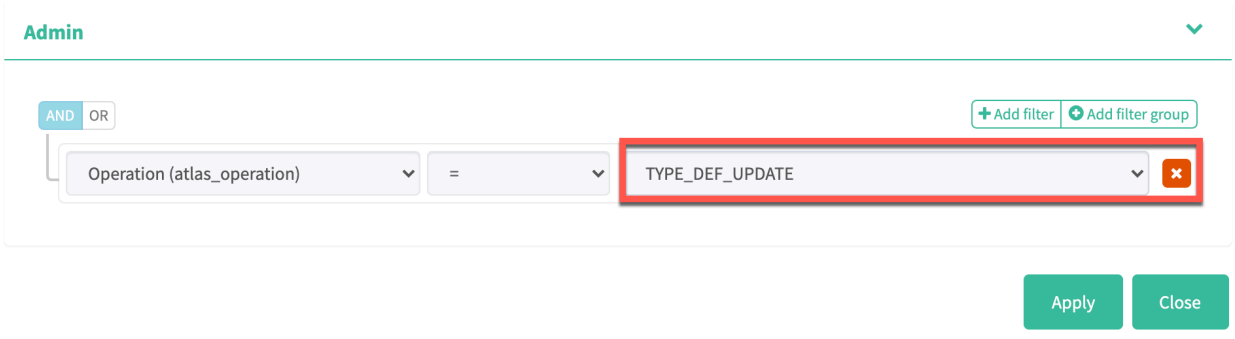
Atlas Type Definitions

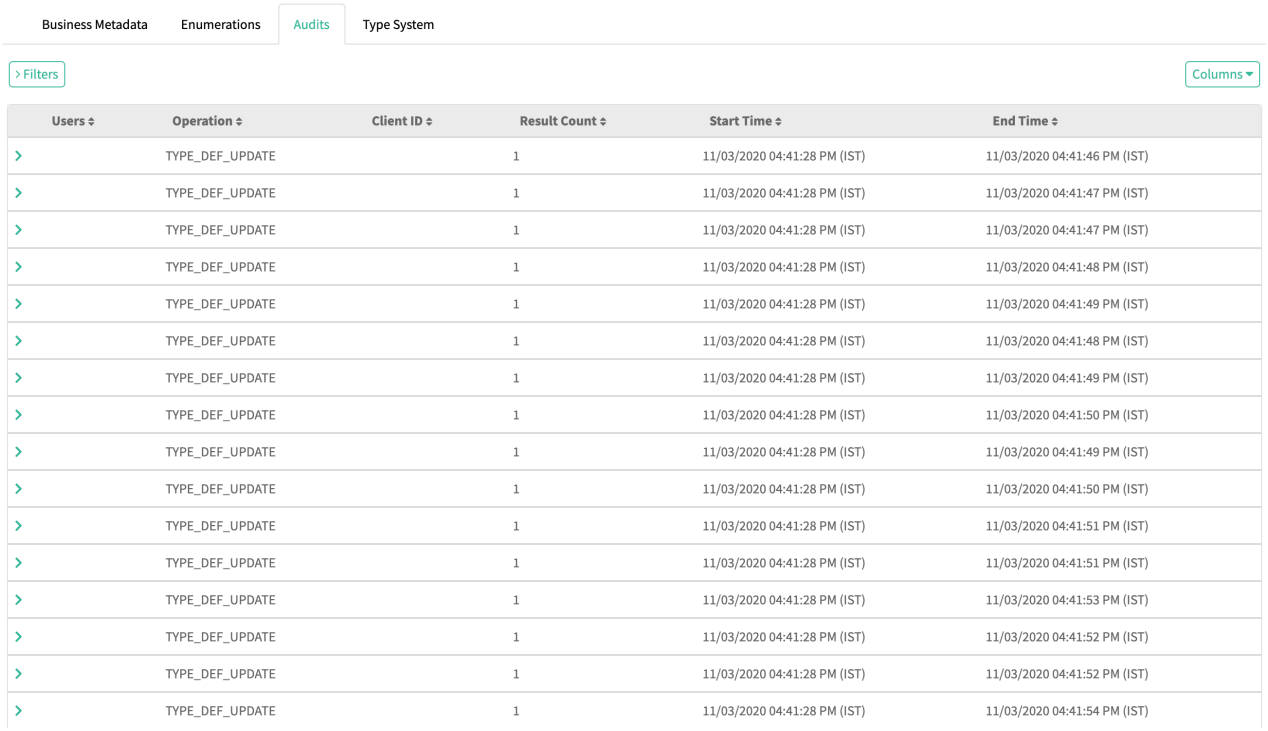
Using Type Definitions, you can create, update, and delete entities of various types.

An example of Type Definition - Create

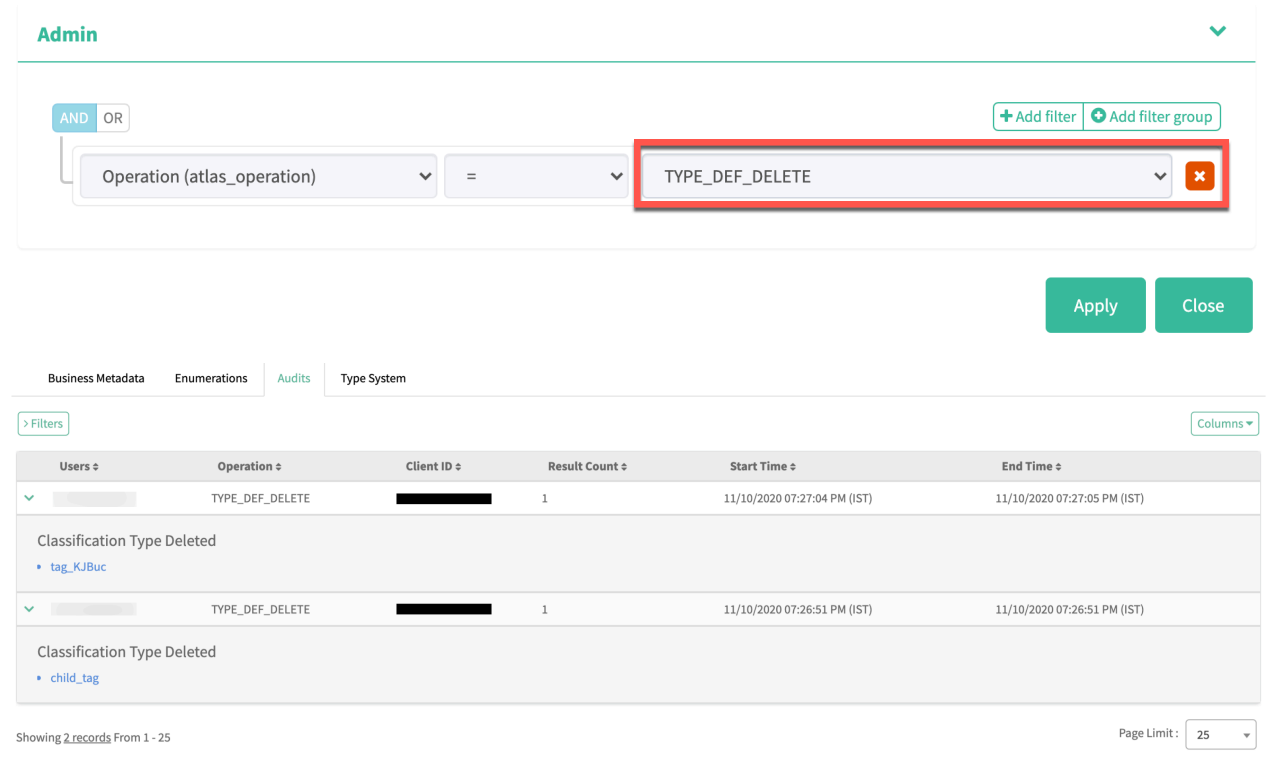


An example of Type Definition - Update





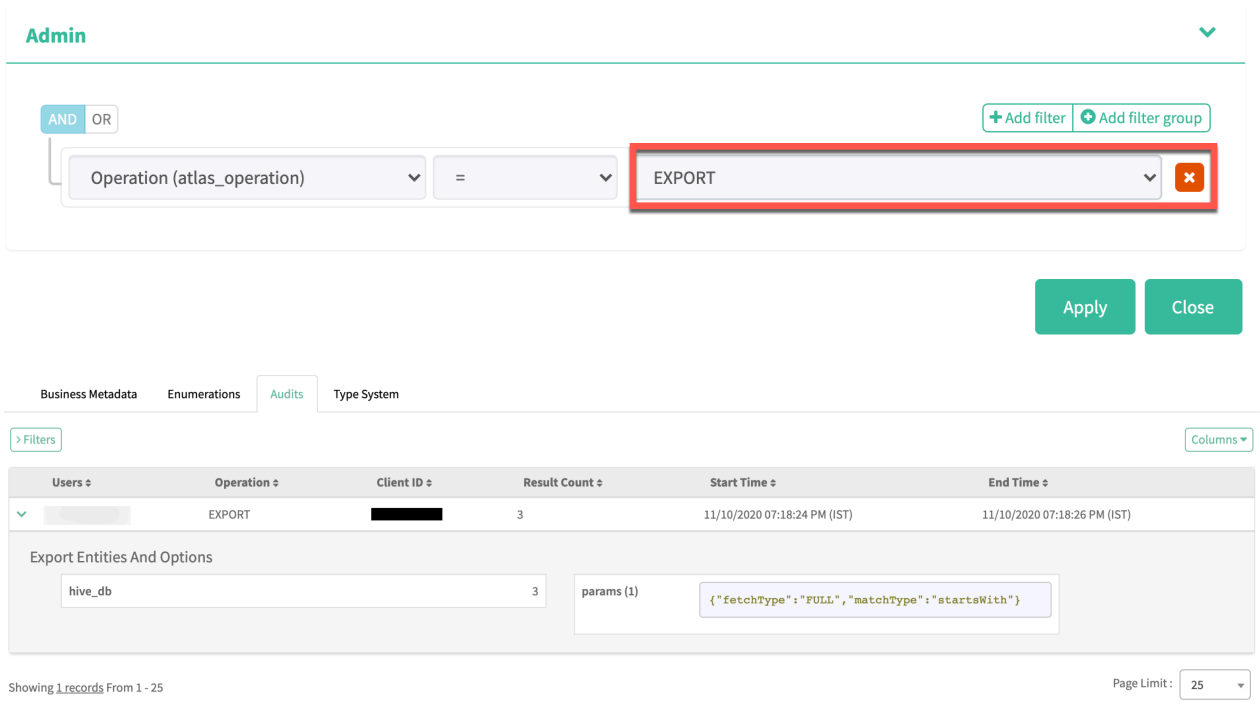
An example of Type Definition - Delete



Atlas Export and Import operations

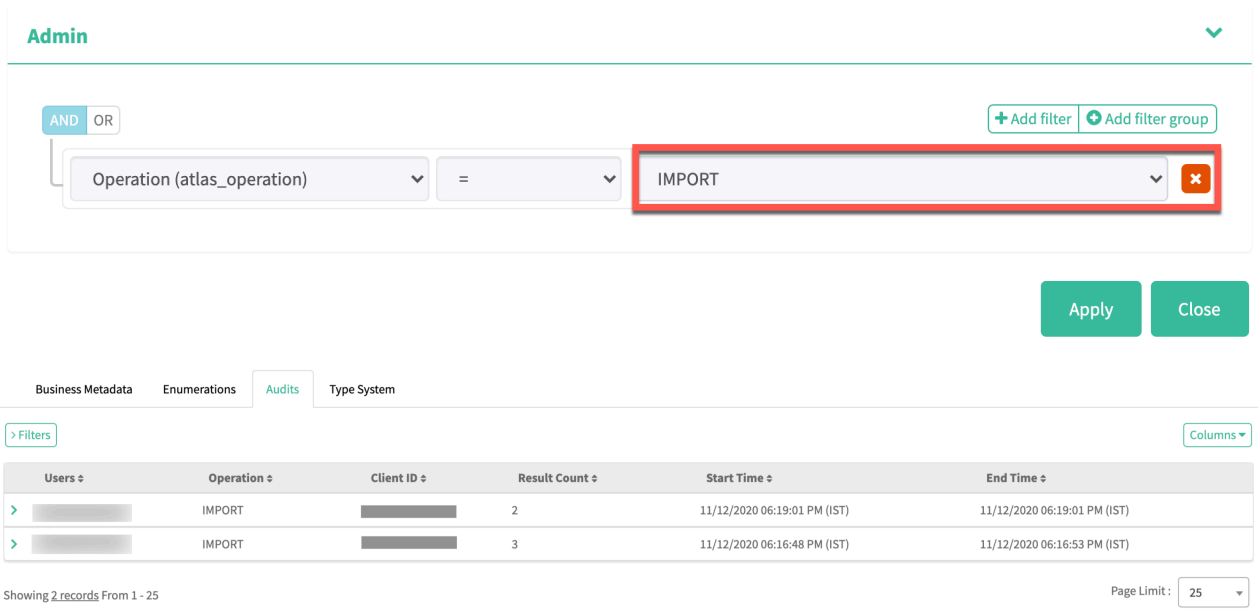
An audit entry is created for every export and import operation.

An example of - Export operation



Each import audit entry provides information about the total number of entities imported, along with the number of entities imported with each Type Definition.

An example of - Import operation



Exporting data using Connected type

As per Apache Atlas Software Foundation notes, only directly connected entities must be exported and when the data is exported with the starting entity as Hive table and the fetch type is "CONNECTED", the exported entities must not include the external and managed locations.

But the expected behavior is that all the entities which are directly connected entities get exported. Additionally, other dependent entities will be updated like the managed location of a database or associated database of a table which also gets exported.

For example:

```
db1.table1 --> p1 ---> db2.table2 ---> p2 ---> db3.table3 ---> p3 ---> db4.t
able4 --> p4 ---> db5.table5

Export db3.table3 with options
{
"itemsToExport": [{
"typeName": "hive_table", "uniqueAttributes":
{ "qualifiedName": "db3.table3@cluster0" }
}],
"options":
{ "fetchType": "connected" }
}
```

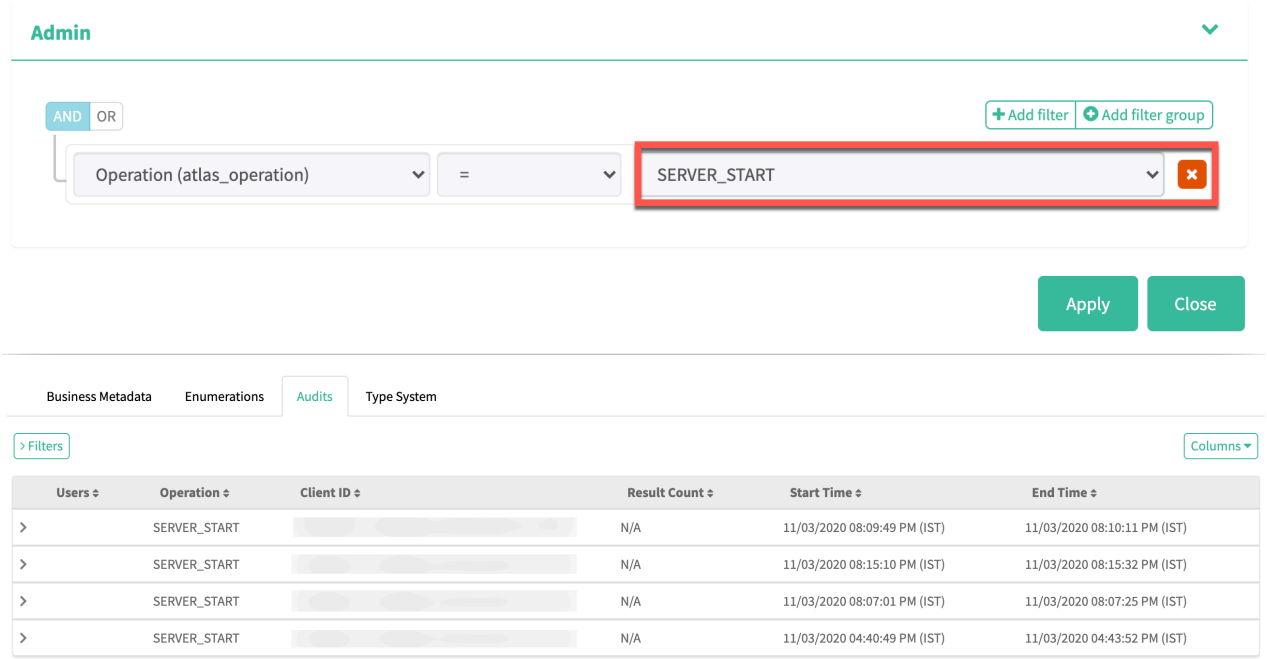
Result: The exported ZIP file must contain entities: db2, db3, db4, table2, table3, table4, p2, and p3.

Atlas Server Operations

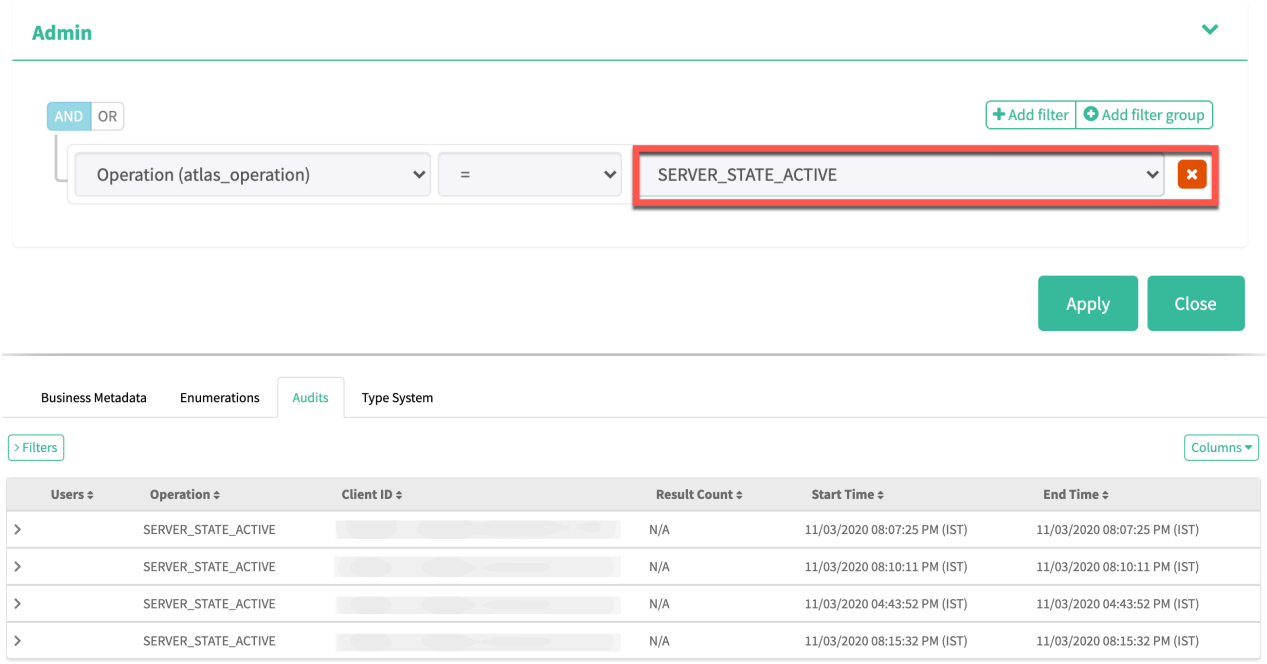
When you perform server related tasks, audit entries are logged.

When the Apache Atlas server is started, an audit entry is logged. Also, when the server is started in the Active mode using High Availability (HA), an audit entry is logged. For more information, see About Atlas High Availability.

An example of - Server Start operation



An example of - Server State Active operation



Related Information

About Atlas High Availability

Audit enhancements

When any entity is created in Apache Atlas, the created entity is stored in Apache HBase tables.

Currently, when the created entity is modified or updated, for example, entity core attributes, relationship attributes, custom attributes, or associated classifications, the changed entity is captured by Atlas either as a full entity object or partial object (with only updated attributes or relations) and stored in HBase tables.

While processing update requests, Atlas generates entity audit events and stores them in the HBase table. These audit events have complete entity information in the JSON format. Multiple updates on a single entity results in generating multiple audit events, each of which has a complete entity instance duplicated with minimal changes.

For example, if entity A1 is updated or modified for about five times, for every update, along with the changes (minimal), the entire entity is stored in the HBase tables. This process consumes additional storage space in HBase tables. In simple terms, A1 + the number of times the changes made is the resultant output that is saved in the HBase tables. Even if the changes are minimal (updating an attribute or relations or something that is not significant), Atlas captures the complete entity.

The Atlas audit enhancement optimizes storage space by not capturing the entire entity instance in each audit event. You can configure Atlas to store only differential information in audit events.

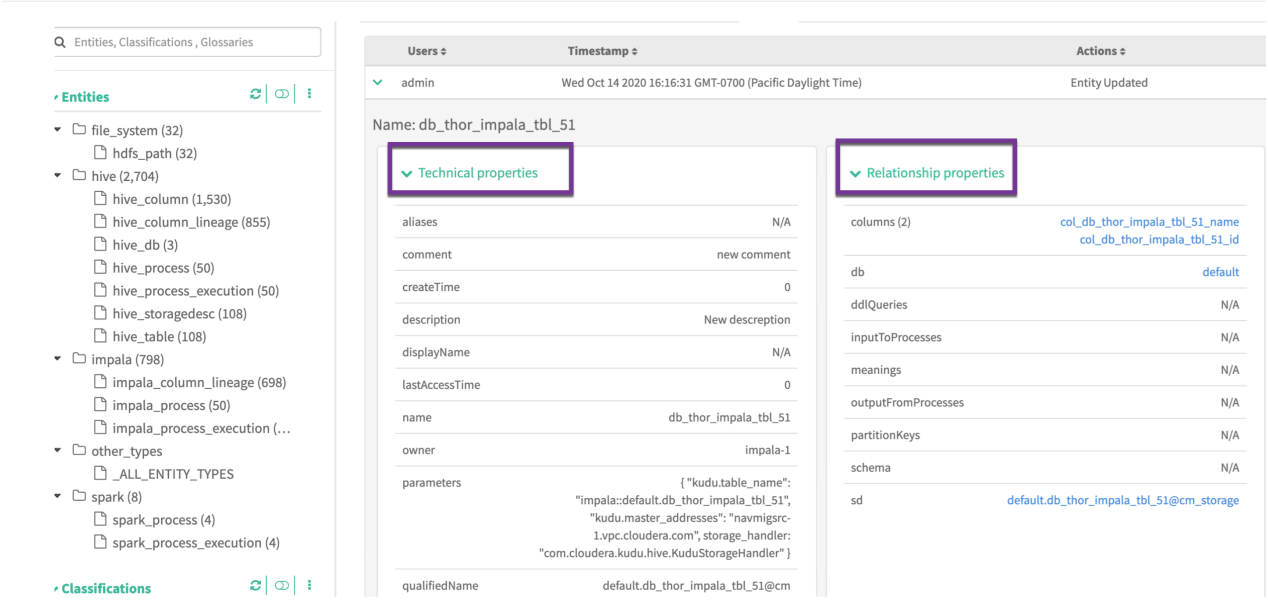
You must enable the application flag by setting the parameter using:

Cloudera Manager UI > Atlas > Configuration > Click Advanced (Under Category) > Enter the following parameter and the value under Atlas Server Advanced Configuration Snippet (Safety Valve) for confi/atlas-application.properties text field.

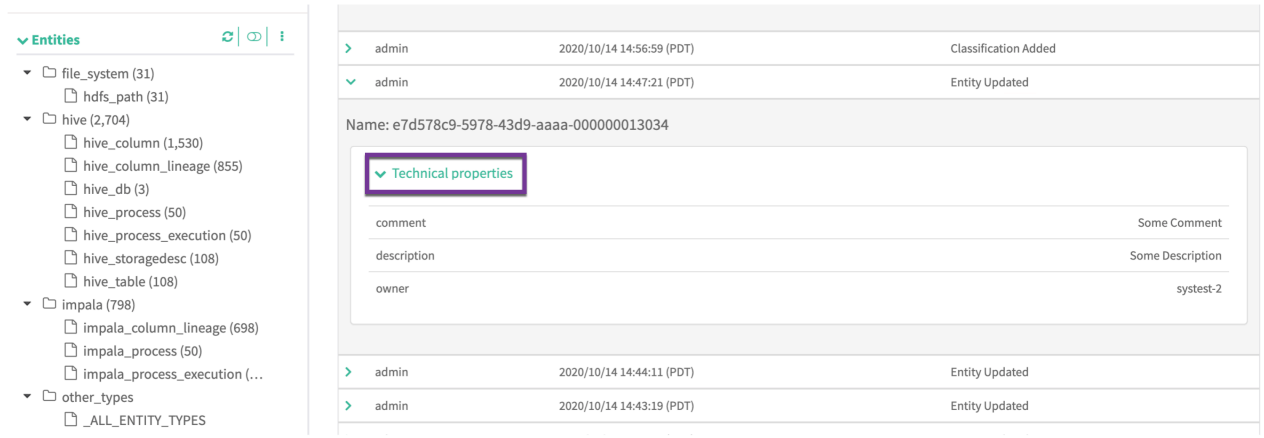
atlas.entity.audit.differential=true

To have significant savings in the HBase table memory footprint, only the difference between the original and updated entity state is captured in Atlas.

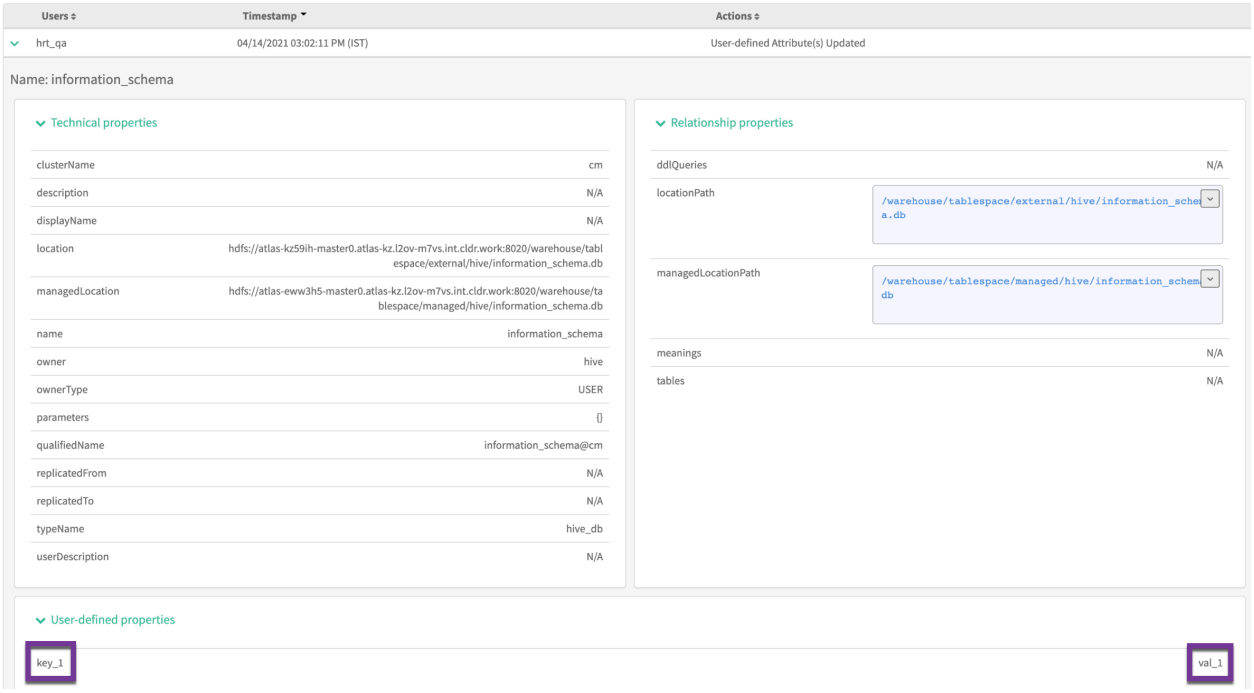
Previously, Atlas exhibited full entity as shown in the image:



Currently, Atlas displays only the differences between the original and updated entity state as shown in the image



As a use case, previously when you add data under user-defined-properties, for example, key_1 and val_1, Atlas displayed the same as seen in the image.



Currently Atlas displays only the changed or updated entity values.

Examples of Audit Operations

Some examples of payload submission for audit operations.

An example of creating a Type Definition:

```
enumDefs:
1.days_of_week
entityDefs:
1.Country
2.State
3.Vehicle

relationshipDefs:
```

```
1.country_state_rel
curl --location --request POST -u admin:admin 'http://car123.test1234.root
.hwx.site:23400/api/atlas/v2/types/typedefs' \
--header 'Content-Type: application/json' \
--data-raw '{
"enumDefs": [
{
"name": "days_of_week",
"typeVersion": "1.0",
"elementDefs": [
{
"ordinal": 1,
"value": "MONDAY"
},
{
"ordinal": 2,
"value": "TUESDAY"
},
{
"ordinal": 3,
"value": "WEDNESDAY"
},
{
"ordinal": 4,
"value": "THURSDAY"
},
{
"ordinal": 5,
"value": "FRIDAY"
},
{
"ordinal": 6,
"value": "SATURDAY"
},
{
"ordinal": 7,
"value": "SUNDAY"
}
]
}
],
"entityDefs": [
{
"category": "ENTITY",
"createdBy": "admin",
"updatedBy": "admin",
"createTime": 1537261952180,
"updateTime": 1537262097732,
"version": 1,
"name": "Vehicle",
"description": "desc Vehicle",
"typeVersion": "1.1",
"attributeDefs": [
{
"name": "no_of_wheels",
"typeName": "int",
"isOptional": true,
"cardinality": "SINGLE",
"valuesMinCount": 1,
"valuesMaxCount": 1,
"isUnique": false,
"isIndexable": false,
"includeInNotification": false
}
```

```
]
},
{
"category": "ENTITY",
"createdBy": "admin",
"updatedBy": "admin",
"createTime": 1537261952180,
"updateTime": 1537262097732,
"version": 1,
"name": "Country",
"description": "desc Country",
"typeVersion": "1.1",
"attributeDefs": [
{
"name": "ISD_CODE",
"typeName": "string",
"isOptional": false,
"cardinality": "SINGLE",
"valuesMinCount": 1,
"valuesMaxCount": 1,
"isUnique": false,
"isIndexable": false,
"includeInNotification": false
}
]
},
{
"category": "ENTITY",
"createdBy": "admin",
"updatedBy": "admin",
"createTime": 1537261952180,
"updateTime": 1537262097732,
"version": 1,
"name": "State",
"description": "desc State",
"typeVersion": "1.1",
"attributeDefs": [
{
"name": "STD_CODE",
"typeName": "string",
"isOptional": false,
"cardinality": "SINGLE",
"valuesMinCount": 1,
"valuesMaxCount": 1,
"isUnique": false,
"isIndexable": false,
"includeInNotification": false
}
]
}
],
"relationshipDefs": [
{
"name": "country_state_rel",
"typeVersion": "1.1",
"relationshipCategory": "AGGREGATION",
"endDef1": {
"type": "Country",
"name": "state_st",
"isContainer": false,
"cardinality": "SINGLE",
"isLegacyAttribute": true
},
"endDef2": {
```

```
"type": "State",
"name": "country_ct",
"isContainer": true,
"cardinality": "SET"
},
"propagateTags": "NONE"
}
]
}'
```

An example of updating a Type Definition:

```
enumDefs:days_of_week
entityDefs:Country
curl --location --request PUT -u admin:admin 'http://car123.car123-1.root.
hwx.site:31000/api/atlas/v2/types/typedefs' \
--header 'Content-Type: application/json' \
--data-raw '{
"enumDefs": [
{
"name": "days_of_week",
"typeVersion": "1.0",
"elementDefs": [
{
"ordinal": 1,
"value": "MONDAY"
},
{
"ordinal": 2,
"value": "TUESDAY"
},
{
"ordinal": 3,
"value": "WEDNESDAY"
},
{
"ordinal": 4,
"value": "THURSDAY"
},
{
"ordinal": 5,
"value": "FRIDAY"
},
{
"ordinal": 6,
"value": "SATURDAY"
},
{
"ordinal": 7,
"value": "SUNDAY"
},
{
"ordinal": 8,
"value": "HOLIDAY"
}
]
}
],
"entityDefs": [
{
"category": "ENTITY",
"createdBy": "admin",
```

```
"updatedBy": "admin",
"createTime": 1537261952180,
"updateTime": 1537262097732,
"version": 1,
"name": "Country",
"description": "desc Country Updated",
"typeVersion": "1.1",
"attributeDefs": [
{
"name": "ISD_CODE",
"typeName": "string",
"isOptional": false,
"cardinality": "SINGLE",
"valuesMinCount": 1,
"valuesMaxCount": 1,
"isUnique": false,
"isIndexable": false,
"includeInNotification": false
}
]
}
]
}'
```

An example of deleting a Type Definition:

```
curl --location --request DELETE -u admin:admin
          'http://car123.car123-1.root.hwx.site:31000/api/atlas/v2/types/typ
edef/name/Vehicle'
```

An example of exporting an Atlas entity:

```
'{
    "itemsToExport": [
            { "typeName": "DB", "uniqueAttributes": { "name": "Sales" }},
            { "typeName": "DB", "uniqueAttributes": { "name": "Reporting" }}
,
            { "typeName": "DB", "uniqueAttributes": { "name": "Logging" }}
    ],
        "options": { "fetchType": "full" }
    }'
```

For additional reference related to exporting entities, see https://atlas.apache.org/#/ExportAPI.

An example of importing an Atlas entity:

Performing an import operation should create an entry in the audits tab.

For additional reference related to exporting entities, see https://atlas.apache.org/#/ImportAPI.

Storage reduction for Atlas

Audit aging reduces the existing audit data in the Atlas system which is based on the end user criteria and configuration changes that users can manage. With this configuration, users can control the type of DDL events pertaining to schema changes that will be retained for a period of time determined by their compliance requirements.

Atlas creates audit events for creation, update, and deletion (CUD) of entities in Cloudera, including TAG/ Classification and DML events. Notification processing with less significant metadata, exponential growth of audit events with no constraints, and retaining of audit data for long periods of time causes overhead on the application processing and consumes additional storage space, thereby causing operational inefficiencies.

Atlas supports various audit reduction strategies, pertaining to the type of audit data the users can store and the period of time that they can retain the data in the Atlas systems.

Some of the audit reduction options include:

- DML Audit Filter - A hook based filter that skips processing of DML events.
- Custom Audit Filters - Users can decide which audit events will be ignored.
- Audit Aging - Users can purge existing audit data based on the criteria they set.

Note: Audit reduction is targeted only for the entity audit data which persists in the ATLAS_ENTITY_AUDIT_EVENTS table. Filtering and aging data is exclusive to Admin Audit data which typically persists in the atlas_janus table.

Attention: Audit aging is employed for optimizing the existing audit events and Audit filters are used to prevent unnecessary audit events.

Using audit aging

You can employ various options to manage the volume of audit data stored and reduce the storage limits on your Atlas instance.

Audit aging overview

Using the audit aging feature helps you to manage the Atlas storage in an efficient manner. Based on certain conditions the Atlas audit data can be deleted or aged out.

Attention:

- Audit aging is allowed only for users with **admin-audits** policy.
- Solr should be up and healthy to perform Audit aging as it depends on Basic search (Solr based) to fetch the entity details.
- By default Audit Aging is disabled through REST API. Default aging is disabled by default in REST API, as the sole purpose of REST API is to trigger custom scenarios than default aging and also default aging option can be time consuming in case of larger or huge data.

Enabling audit aging

You can initiate and configure Audit aging feature in Atlas either though application-properties in Cloudera Manager and the REST API route.

For all the audit aging operations, the configuration can be enabled through application properties.

From Cloudera Manager > Configuration tab > Atlas Server Advanced Configuration Snippet (Safety Valve) for conf/atlas-application.properties

Once the configuration is updated in Cloudera Manager, restart Atlas to enable audit aging and also follow the audit aging process based on the scheduled frequency.

You can specify the audit aging cases based on the configuration property named atlas.audit.aging.enabled=true. By default, this property is disabled. If needed, you must explicitly enable the property.

The following types of audit aging are supported:

- Default Audit Aging
- Sweep out Audits
- Custom Audit Aging

Using default audit aging

You can use the default audit aging mechanism to configure Time-to-Live (TTL), audit count limit, and disable default audit aging processes.

Using Time-to-Live (TTL) configuration

(TTL) is a value for the period of time that a packet, or data, should exist on a computer or network before being discarded.

Configuration to set TTL: atlas.audit.default.ageout.ttl.in.days

This configuration makes a final decision on how long the audit data can be retained in the database. This configuration is used by the Audit aging scheduler to delete all audit data when the audit data lifetime crosses the configured TTL. This configuration is applicable for all entity and audit action types.

By default only TTL will be configured for 90 days and no audit count will be considered.

Note: Using the REST API route, when default aging is enabled without any ttl configuration, default aging will be triggered with default ttl value which is 90 days.

As an example for TTL configuration usage, consider the following scenario::

You must maintain the entire audit data only for 40 days. The following configuration deletes audit data older than 40 days.

atlas.audit.default.ageout.ttl.in.day=40

Note: The parameter atlas.audit.default.ageout.ignore.ttl to be configured in case ttl is ignored and only audit count is to be considered for default aging.

Using Audit count limit parameter

Configuration to set allowed: atlas.audit.default.ageout.count

Using this configuration limits audit data for each entity. Atlas deletes all old audit data exceeding the configured audit count for all entities. This configuration is applicable for all entity and audit action types.

As an example for Audit limit count configuration usage, consider the following scenario:

You must maintain only the latest 20 audits for any entity.

atlas.audit.default.ageout.count=20

Using Disable Default Audit Aging parameter

Configuration to disable default audit aging: atlas.audit.default.ageout.enabled

For all entities, the process is default aging.This process consumes more time and resources. Under certain circumstances, if you want to execute only custom aging or sweep out features for minimal data, using this property default aging can be disabled.

By default, default audit aging is enabled.

Related Information

Using Sweep out configurations

Using custom audit aging

Using Sweep out configurations

You can employ the sweep out option for the entire audit data without any limitations or restrictions of TTL or minimum data count.

Caution: Use the sweep out feature carefully as any wrong usage can wipe out the entire audit data.

Sweep out option supports the following configurations to manage your audit data:

- The default Sweep out configuration is atlas.audit.sweep.out.enabled=true. The default value is false (disabled).

  To delete the entire audit data for specific entity and audit action types, you must configure at least one of the following properties:

  - atlas.audit.sweep.out.entity.types=<List of entity types> With sweep out option enabled, by default it is NOT applicable for any entity or action types. You can override the configuration with specific entity or action types.
  - atlas.audit.sweep.out.action.types=<List of audit action types> With sweep out option enabled, by default it is NOT applicable for any entity or action types. You can override the configuration with specific entity or action types.

Use cases detailing the usage of the Sweep out option using specific configurations.

Sweep out action	Applicable configuration
Delete entire audit data for hive_column and hive_storagedesc	atlas.audit.sweep.out.enabled=trueatlas.audit.sweep.out.entity.types
Delete entity update events for all entity types	atlas.audit.sweep.out.enabled=true atlas.audit.sweep.out.action.types=ENTITY_UPDATE
Delete all entity update events for hive_table	atlas.audit.sweep.out.enabled=true atlas.audit.sweep.out.entity.types=hive_table atlas.audit.sweep.out.action.types=ENTITY_UPDATE

Related Information

Using default audit aging

Using custom audit aging

Using custom audit aging

For additional flexibility to manage audit data, the custom audit aging mechanism is used to contrive the configuration of TTL and limit audit event count which is based on entity or audit action type.

Attention: Default configuration for custom aging is not supported. For custom aging to be in effect, users must specifically configure ttl/audit count and entityTypes/actionTypes.

Supported custom aging configurations:

- atlas.audit.custom.ageout.count=20
- atlas.audit.custom.ageout.ttl.in.days=30
- atlas.audit.custom.ageout.entity.types=<List of entity types>
- atlas.audit.custom.ageout.action.types=<List of audit action types>

Using these configurations, Atlas limits (with audit count) or age-out (with TTL) audit data for the configured entity and audit action types.

Note: Custom audit count and custom TTL configurations are applicable only when at least one of the custom entity or action types is configured.

Custom audit aging configurations are categorized using the following use cases:

Actions by	Description
Entity Type	Example: Limit to five latest audits for hive_column. atlas.audit.custom.ageout.count=5 atlas.audit.custom.ageout.entity.types=hive_column

By Action Type	Example: Delete all ENTITY_UPDATE audit events older than ten days. atlas.audit.custom.ageout.ttl.in.days=10 atlas.audit.custom.ageout.action.types=ENTITY_UPDATE
Limited audit by Action type for specific entity type	Example: Limit to five latest ENTITY_UPDATE audits for hive_storagedesc entities. • atlas.audit.custom.ageout.count=5 • atlas.audit.custom.ageout.entity.types=hive_storagedesc • atlas.audit.custom.ageout.action.types=ENTITY_UPDATE Example: Limit to five latest audits created in the last 1 week for hive_db entities • atlas.audit.custom.ageout.count=5 • atlas.audit.custom.ageout.ttl.in.days=7 • atlas.audit.custom.ageout.entity.types=hive_db

Attention: Regular expressions patterns are allowed patterns for action types and entity types.

The supported regex for entity types is entity type suffixed with '*'

For Example: hive_* considers all the entity types that start with hive_

The supported regex for action types is action type prefixed/suffixed with '*'.

For Example: CLASSIFICATION* or *CLASSIFICATION => Consider all the action types that contain CLASSIFICATION

Related Information

Using default audit aging

Using Sweep out configurations

Aging patterns

When you simultaneously perform multiple aging operations, certain features override the order in which the aging process takes place.

You must note about the following aging options and how they take effect.

- If the sweep out option is configured, sweep out entity types action types (if any) gets skipped from Custom aging and Default aging operations.
- If Custom aging is configured, custom entity types and action types (if any) get skipped from Default aging operations.
- Default aging skips the aging process for the entity and action types configured under custom and sweep out options and shall continue to process for all the remaining entity and action types.

Finally, if all the three aging options are simultaneously configured during the audit reduction cycle, Atlas follows the process in the following order of occurrence:

Sweep out -> Custom aging -> Default aging

Related Information

Audit aging reference configurations

Audit aging reference configurations

You must be aware about other related audit aging configurations that are required while working with optimizing the storage requirements and retaining the required audit data in your Atlas application.

The following configuration options provide you flexibility to manage your audit aging operations.

- By default all the aging options are applicable to only non **ENTITY_CREATE** events. Atlas ignores deleting all **ENTITY_CREATE** / **ENTITY_CREATED_BY_IMPORT** events by default. If users need

to delete audit data, including entity creation events, they can configure this property to true. By default it is configured as false. If you set the value of atlas.audit.create.events.ageout.allowed=true , **ENTITY_CREATE / ENTITY_IMPORT_CREATE** events will be considered as any other non-create events and will be deleted.

- The frequency for performing the audit aging process is moderated by using the atlas.audit.aging.scheduler.frequency.in.days=10 parameter. By default the audit aging process is scheduled for every 30 days.
- You cannot directly configure the custom or default TTL value less than the minimum TTL configured for this property - min.ttl.to.maintain. If the TTL value is configured for less than the minimum TTL, Atlas considers TTL as the value configured for the parameter. The default value is 7.
- You cannot directly configure custom or default audit count less than the minimum audit count configured for this property - min.audit.count.to.maintain. If you configure a lesser value, Atlas considers audit count as the value configured for the parameter. The default value is 50.
- By default audit aging will be executed for all the subtypes of the entity types configured using the property atlas.audit.aging.subtypes.included. By default it is configured as true. To limit audit aging operation for configured entity types but not sub types, this configuration can be used to exclude the subtypes. For example, the iceberg_ table is a subtype of hive_table and if the user wants to purge audit data for only hive_table, in that case this configuration can be set to false.

Related Information

Aging patterns

Audit aging using REST API

Optionally, Atlas triggers Audit aging through REST API configuration. Using REST API to enable Audit aging features can be less time consuming and beneficial. By default Audit Aging is disabled through REST API.

Examples for employing REST APIs in specific scenarios:

- For a single instance, when you can perform an audit aging process for any specific scenario without changing any scheduled audit aging configuration, you can trigger audit aging using the REST API by passing the specified audit criteria as a payload.
- When regular configuration based Audit aging is scheduled for every 30 days, and user wants to trigger it before the next scheduled time, it can be done through REST API by passing the query parameter

```
useAuditConfig
```

Note: Default aging option is disabled by default while using the REST API option.

An example usage of the REST API feature:

```
POST /api/atlas/admin/audits/ageout?useAuditConfig=false
Payload options:
{
    “auditAgingEnabled”:false,
    “defaultAgeoutEnabled”:false,
    “createEventsAgeoutAllowed”:false,
    “subTypesIncluded”:true,
    “defaultAgeoutAuditCount”:10,
    “defaultAgeoutTTLInDays”:30,
    “customAgeoutAuditCount”:10,
    “customAgeoutTTLInDays”:30,
    “customAgeoutEntityTypes”:”hive_table”,
    “customAgeoutActionTypes”:”ENTITY_UPDATE”,
    “auditSweepoutEnabled”:true,
    “sweepoutEntityTypes”:”hive_column”,
    sweepoutActionTypes:”CLASSIFICATION_ADD”
}
```

Using custom audit filters

Customizing Atlas audit filters using the users' criteria provides you with the flexibility to finalize which audit events can be persisted with and which of the events can be ignored.

Custom audit filters are configured in the Atlas server which improves the performance of audit trail and an enhancement towards audit reduction. Using this feature, you can set some predefined rules or conditions that enable you to filter certain events that queues up the storage space in Atlas.

Common example use-cases:

- User might want to skip audit for all temporary tables.
- User might want to skip all entities name starts with "test".
- User might want to skip all update events for all entities.
- All types of operation can be allowed for auditing.
- All types of operation can be ignored from auditing.
- You can accept/discard audits by one specific attribute or combination of multiple attributes. Example attributes include, typeName, isIncomplete, temporary, and attributes (like owner, qualifiedName, name).

The audit filters operate based on a logic that transforms into a rule that results in a single action. The logic executes rules stored in the database to perform the following actions:

- Retrieve appropriate rules stored in the database
- Evaluate one or more rules.
- Match the rule conditions.
- Perform one or more actions that match the conditions.
- Result orientation.
- Repeat the process for the remaining rules.

Custom audit filters are supported using the REST API method.

The following REST API calls are used to perform various custom audit filter operations.

- POST: /api/atlas/admin/audits/rules
- GET: /api/atlas/admin/audits/rules
- PUT: /api/atlas/admin/audits/rules/{guid}
- DELETE: /api/atlas/admin/audits/rules/guid/{guid}
- DELETE: /api/atlas/admin/audits/rules
- DELETE: /api/atlas/admin/audits/rules/all

Supported operators

Operators currently supported for custom audit filters are numeric, boolean, and string.

Table 1: Supported operations types

Numeric operations	Boolean operations	String operations
“<”: less than “>”: greater than “<= : less than or equal to “>=: greater than or equal to	“==”: equals to “!=”: not equals to	“startsWith”: starts with “endsWith”: ends with “contains”: contains (case-sensitive) “notContains”: does not contain (case-sensitive) “isNull”: is null “notNull”: is not null “containsIgnoreCase”: contains (case-insensitive) “notContainsIgnoreCase”: does not contain (case-insensitive)

A typical rule is defined in the following manner:

```
{
  "desc": "Discard all hive_table audits with name containing test",
  "action": "DISCARD",
  "ruleName": "test_rule_1",
  "ruleExpr": {
      "ruleExprObjList": [
          {
              "typeName": "hive_table",
              "attributeName": "name",
              "operator": "contains",
              "attributeValue": "test"
          }
      ]
  }
 }
```

Related Information

Rule configurations

Rule configurations

You must use the following configurations to set up the custom audit filters.

Use the following configurations to control the usage:

From Cloudera Manager > Configuration tab > Atlas Server Advanced Configuration Snippet (Safety Valve) for conf/atlas-application.properties

Enabling custom filters:

atlas.entity.audit.filter.enabled=false

By default this feature is disabled.

When no rules are created or defined, the default action is to accept the rule. The value can be changed to discard using the following parameter.

atlas.entity.audit.filter.default.action=DISCARD

Note: Permitted values are ACCEPT and DISCARD

Related Information

Supported operators

Use cases and sample payloads

Use cases and sample payloads

Assuming the default action is to ACCEPT an audit and the user wants to discard the audits conditionally, you must understand the rules payload for some of the common use case scenarios.

Discard temporary and test hive_table audits (Nested rules example)

```
"action":"DISCARD",
"ruleName":"test_rule_1",
"ruleExpr":{
   "ruleExprObjList":[
      {
         "typeName":"hive_table",
         "condition":"AND",
         "criterion":[
            {
               "operator":"==",
               "attributeName":"temporary",
               "attributeValue":"false"
            },
            {
               "condition":"OR",
               "criterion":[
                  {
                     "operator":"==",
                     "attributeName":"name",
                     "attributeValue":"tmp"
                  },
                  {
                     "operator":"==",
                     "attributeName":"qualifiedName",
                     "attributeValue":"tmp"
                  }
               ]
            }
         ]
      }
   ]
}
```

Discard all audits of a type

```
"action":"DISCARD",
"ruleName":"test_rule_1",
"ruleExpr":{
   "ruleExprObjList":[
      {
         "typeName":"hive_table"
      }
   ]
}
```

Discard all update audits for all entities

```
"action":"DISCARD",
"ruleName":"test_rule_1",
"ruleExpr":{
   "ruleExprObjList":[
      {
         "typeName":"”_ALL_ENTITY_TYPES”",
         "operator":"==",
```

```
         "attributeName":"operationType",
         "attributeValue":"ENTITY_UPDATE"
      }
   ]
}
```

Discard all CLASSIFICATION_ADD audits for all entities

```
"action":"DISCARD",
"ruleName":"test_rule_1",
"ruleExpr":{
   "ruleExprObjList":[
      {
         "typeName":"_ALL_ENTITY_TYPES",
         "operator":"==",
         "attributeName":"operationType",
         "attributeValue":"CLASSIFICATION_ADD"
      }
   ]
}
```

Discard audits for entity DELETE operation types such as ENTITY_DELETE, ENTITY_IMPORT_DELETE, CLASSIFICATION_DELETE, PROPAGATED_CLASSIFICATION_DELETE, and LABEL_DELETE.

```
{
  "desc":"test3",
  "action":"DISCARD",
  "ruleName":"rule123",
  "ruleExpr":{
     "ruleExprObjList":[
        {
           "typeName":"hive_table",
           "condition":"AND",
           "criterion":[
              {
                 "operator":"contains",
                 "attributeName":"operationType",
                 "attributeValue":"DELETE"
              }
           ]
        }
     ]
  }
```

Usage of DELETE API for multiple rules: specify array of guids using comma separator in payload

```
URL: api/atlas/admin/audits/rules
Payload : ["477d8fcd-3d89-4c4c-bb91-9586c49fbd19","8b2b37a8-30eb-4510-b19a
-589816e45b80"]
```

Usage of DELETE API to delete all rules

```
URL: api/atlas/admin/audits/rules/all
Payload : not required
```

Discard audits for hive tables where description is null

```
Payload:
{
   "action": "DISCARD",
```

```
    "ruleName": "hiverule3",
    "ruleExpr": {
        "ruleExprObjList": [
            {
                "typeName": "hive_table",
                "includeSubTypes": "false",
                "attributeName": "description",
                "operator": "isNull"
            }
        ]
    }
```

Discard audits of event for a specific type based on attribute value

```
"action":"DISCARD",
"ruleName":"test_rule_1",
"ruleExpr":{
   "ruleExprObjList":[
      {
         "typeName":"hive_table",
         "condition":"AND",
         "criterion":[
            {
               "operator":"==",
               "attributeName":"operationType",
               "attributeValue":"ENTITY_UPDATE"
            },
            {
               "operator":"==",
               "attributeName":"name",
               "attributeValue":"employee"
            }
         ]
      }
   ]
}
```

Discard audits for all types under a hook type (Regex supported with wildcard character *)

```
"action":"DISCARD",
"ruleName":"test_rule_1",
"ruleExpr":{
   "ruleExprObjList":[
      {
         "typeName":"hive*",
         "operator":"==",
         "attributeName":"operationType",
         "attributeValue":"CLASSIFICATION_ADD"
      }
   ]
}
```

Note: All audits of Hive table, Hive column, DB, storage desc, and process are discarded for CLASSIFICATION_ADD event.

Discard all audits of a type and its sub types

```
"action":"DISCARD",
"ruleName":"test_rule_1",
"ruleExpr":{
   "ruleExprObjList":[
```

```
      {
         "typeName":"Asset",
         "includeSubTypes":true
      }
   ]
}
```

Note: All asset type audits are discarded.

CSV of type-names is supported

```
"action":"DISCARD",
"ruleName":"test_rule_1",
"ruleExpr":{
   "ruleExprObjList":[
      {
         "typeName":"hive_table,hbase_table",
         "attributeName":"name",
         "operator":"contains",
         "attributeValue":"test1"
      }
   ]
}
```

Related Information

Rule configurations