

# Apache Atlas

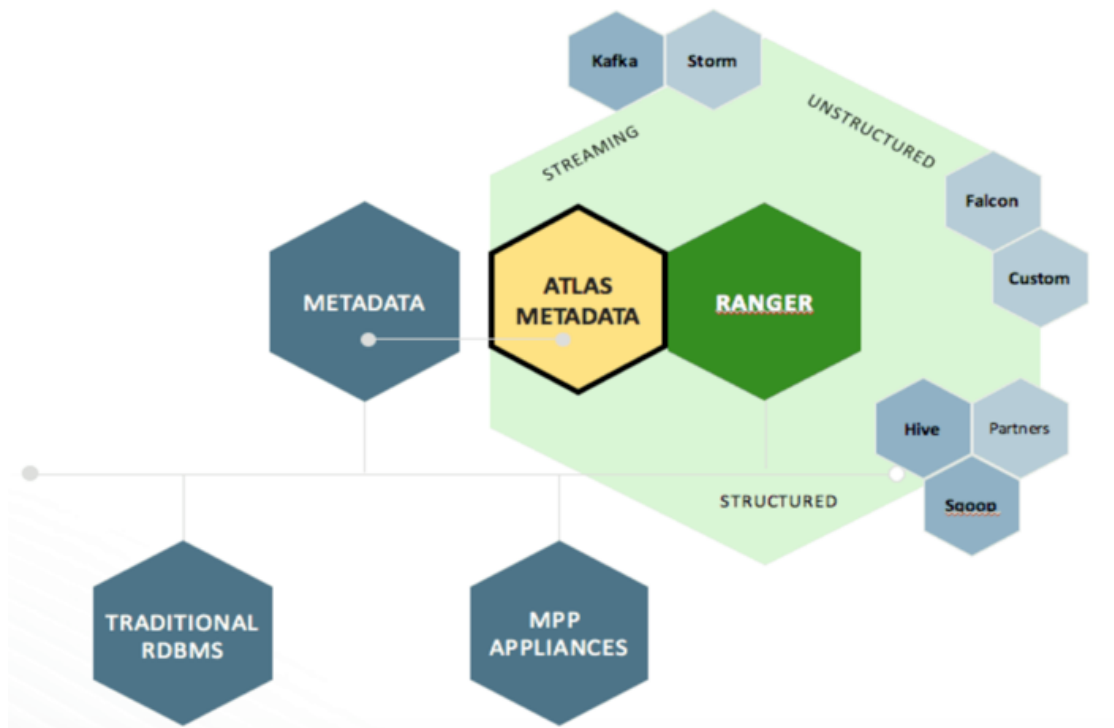
Apache Atlas 是 Hadoop 社区为解决 Hadoop 生态系统的元数据治理问题而产生的开源项目，它为 Hadoop 集群提供了包括数据分类、集中策略引擎、数据血缘、安全和生命周期管理在内的元数据治理核心能力。

## 概述

面对海量且持续增加的各式各样的数据对象，你是否有信心知道哪些数据从哪里来以及它如何随时间而变化？采用 Hadoop 必须考虑数据管理的实际情况，元数据与数据治理成为企业级数据湖的重要部分。

为寻求数据治理的开源解决方案，Hortonworks 公司联合其他厂商与用户于 2015 年发起数据治理倡议，包括数据分类、集中策略引擎、数据血缘、安全和生命周期管理等方面。Apache Atlas 项目就是这个倡议的结果，社区伙伴持续的为该项目提供新的功能和特性。该项目用于管理共享元数据、数据分级、审计、安全性以及数据保护等方面，努力与 Apache Ranger 整合，用于数据权限控制策略。

Apache Atlas 是 hadoop 的数据治理和元数据框架，它提供了一个可伸缩和可扩展的核心基础数据治理服务集，使得企业可以有效的和高效的满足 Hadoop 中的合规性要求，并允许与整个企业的数据生态系统集成：



## 核心特性

Apache Atlas 为 Hadoop 的元数据治理提供了以下特性：

## 数据分类

- 为元数据导入或定义业务导向的分类注释
- 定义，注释，以及自动捕获数据集和底层元素之间的关系
- 导出元数据到第三方系统

## 集中审计

- 捕获与所有应用，过程以及与数据交互的安全访问信息
- 捕获执行，步骤，活动等操作的信息

## 搜索与血缘

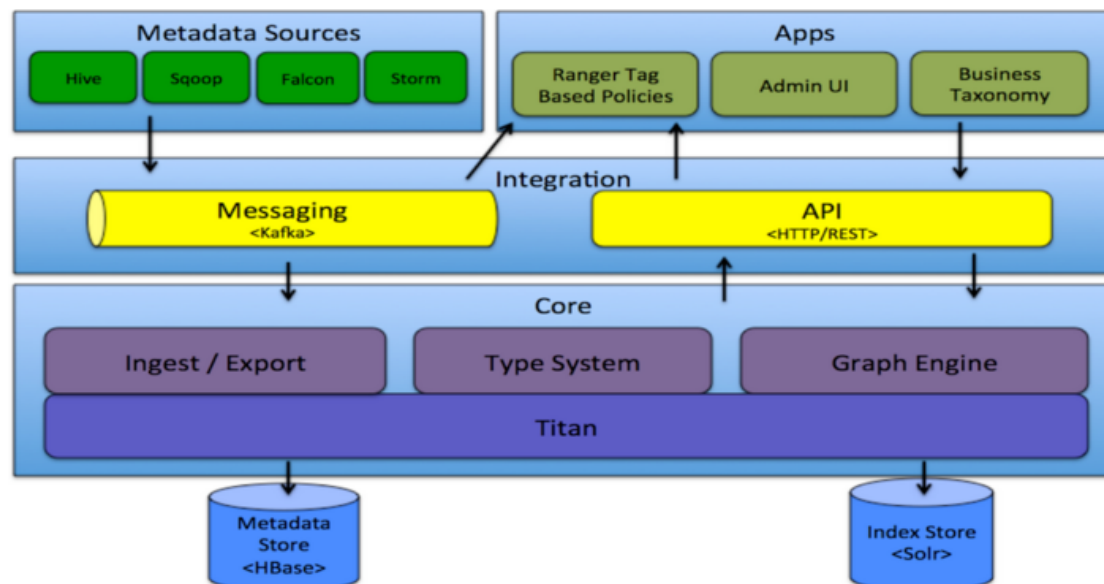
- 预定义的导航路径用来探索数据分类以及审计信息

- 基于文本的搜索特性来快速和准确的定位相关联的数据和审计事件
- 对数据集血缘关系的可视化浏览使用户可以下钻到操作，安全以及数据起源相关的信息

## 安全与策略引擎

- 基于数据分类模式，属性以及角色的运行时合理合规策略
- 基于分类-预测的高级策略定义以防止数据推导
- 基于 cell 的属性和值的行/列级别的 masking

Apache Atlas 的架构如下图所示：



Atlas 的组件可以分为以下几个部分：

### Core

此类别包含实现 Atlas 功能核心的组件，包括：

- Type System: Atlas 允许用户为他们想要管理的元数据对象定义一个模型。该模型由称为 "类型" 的定义组成。"类型" 的实例被称为 "实体" 表示被管理的实际元数据对象。类型系统是一个组件，允许用户定义和管理类型和实体。由 Atlas 管理的所有元数据对象（例如 Hive 表）都使用类型进行建模，并表示为实体。要在 Atlas 中存储新类型的元数据，需要了解类型系统组件的概念。
- Ingest/Export : Ingest 组件允许将元数据添加到 Atlas。类似地，Export 组件

暴露由 Atlas 检测到的元数据更改，以作为事件引发，消费者可以使用这些更改事件来实时响应元数据更改。

- Graph Engine：在内部，Atlas 通过使用图形模型管理元数据对象。以实现元数据对象之间的巨大灵活性和丰富的关系。图形引擎是负责在类型系统的类型和实体之间进行转换的组件，以及基础图形模型。除了管理图形对象之外，图形引擎还为元数据对象创建适当的索引，以便有效地搜索它们。

- Titan：目前，Atlas 使用 Titan 图数据库来存储元数据对象。Titan 使用两个存储：默认情况下元数据存储配置为 HBase，索引存储配置为 Solr。也可以通过构建相应的配置文件使用 BerkeleyDB 存储元数据存储 和使用 Elasticsearch 存储 Index。元数据存储用于存储元数据对象本身，索引存储用于存储元数据属性的索引，其允许高效搜索。

## Integration

用户可以使用两种方法管理 Atlas 中的元数据：

- API：Atlas 的所有功能都可以通过 REST API 提供给最终用户，允许创建，更新和删除类型和实体。它也是查询和发现通过 Atlas 管理的类型和实体的主要方法。

- Messaging：除了 API 之外，用户还可以选择使用基于 Kafka 的消息接口与 Atlas 集成。这对于将元数据对象传输到 Atlas 以及从 Atlas 使用可以构建应用程序的元数据更改事件都非常有用。如果希望使用与 Atlas 更松散耦合的集成，这可以允许更好的可扩展性，可靠性等，消息传递接口是特别有用的。Atlas 使用 Apache Kafka 作为通知服务器用于钩子和元数据通知事件的下游消费者之间的通信。事件由钩子(hook)和 Atlas 写到不同的 Kafka 主题：

- ATLAS\_HOOK: 来自 各个组件的 Hook 的元数据通知事件通过写入到名为 ATLAS\_HOOK 的 Kafka topic 发送到 Atlas

- ATLAS\_ENTITIES：从 Atlas 到其他集成组件（如 Ranger）的事件写入到名为 ATLAS\_ENTITIES 的 Kafka topic

## Metadata source

Atlas 支持与许多元数据源的集成，将来还会添加更多集成。目前，Atlas 支持从以下数据源获取和管理元数据：

- Hive：通过 hive bridge， atlas 可以接入 Hive 的元数据，包括 hive\_db/hive\_table/hive\_column/hive\_process
  - Sqoop：通过 sqoop bridge， atlas 可以接入关系型数据库的元数据，包括 sqoop\_operation\_type/sqoop\_dbstore\_usage/sqoop\_process/sqoop\_dbdatastore
  - Falcon：通过 falcon bridge， atlas 可以接入 Falcon 的元数据，包括 falcon\_cluster/falcon\_feed/falcon\_feed\_creation/falcon\_feed\_replication/falcon\_process
  - Storm：通过 storm bridge， atlas 可以接入流式处理的元数据，包括 storm\_topology/storm\_spout/storm\_bolt
- Atlas 集成大数据组件的元数据源需要实现以下两点：
- 首先，需要基于 atlas 的类型系统定义能够表达大数据组件元数据对象的数据模型（例如 Hive 的元数据模型实现在 org.apache.atlas.hive.model.HiveDataModelGenerator）；
  - 然后，需要提供 hook 组件去从大数据组件的元数据源中提取元数据对象，实时侦听元数据的变更并反馈给 atlas；

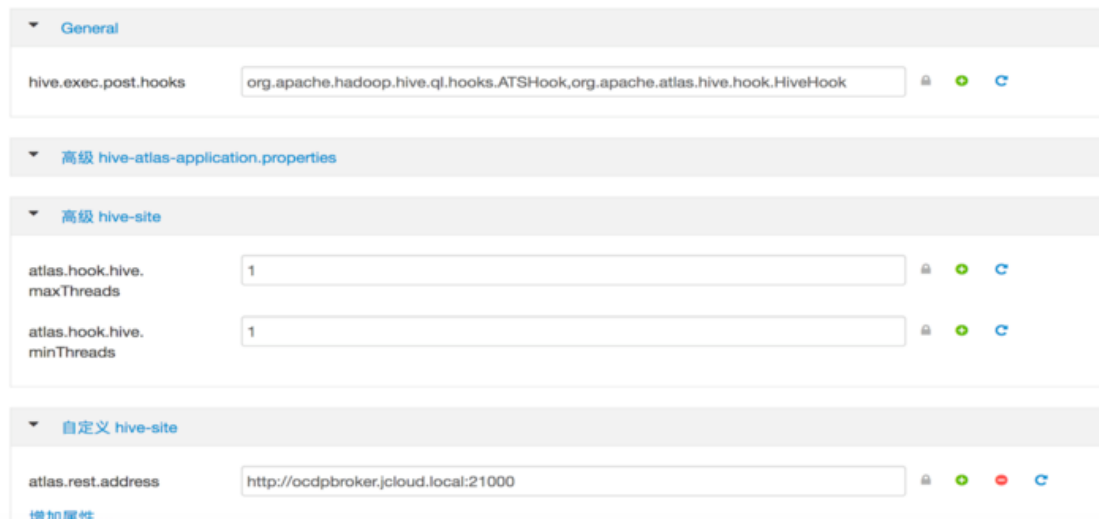
## Applications

- Atlas Admin UI: 该组件是一个基于 Web 的应用程序，允许数据管理员和科学家发现和注释元数据。Admin UI 提供了搜索界面和 类 SQL 的查询语言，可以用来查询由 Atlas 管理的元数据类型和对象。Admin UI 使用 Atlas 的 REST API 来构建其功能。
- Tag Based Policies: Apache Ranger 是针对 Hadoop 生态系统的高级安全管理解决方案，与各种 Hadoop 组件具有广泛的集成。通过与 Atlas 集成，Ranger 允许安全管理员定义元数据驱动的安全策略，以实现有效的治理。Ranger 是由 Atlas 通知的元数据更改事件的消费者。
- Business Taxonomy: 从元数据源获取到 Atlas 的元数据对象主要是一种技术形式的元数据。为了增强可发现性和治理能力，Atlas 提供了一个业务分类界面，允许用户首先定义一组代表其业务域的业务术语，并将其与 Atlas 管理的元数据实体相关联。业务分类法是一种 Web 应用程序，目前是 Atlas Admin UI 的一部分，并且使用 REST API 与 Atlas 集成。
- 在 HDP2.5 中，Business Taxonomy 是提供了 Technical Preview 版本，需要在

Atlas > Configs > Advanced > Custom application-properties 中添加 atlas.feature.taxonomy.enable=true 并重启 atlas 服务来开启

## 部署与配置

在 HDP 集群中，可以通过 Ambari 快速部署 Apache atlas 服务：



The image shows the Ambari configuration interface for Apache Atlas. It is divided into three main sections: General, 高级 hive-atlas-application.properties, and 自定义 hive-site. The General section shows the hive.exec.post.hooks configuration with the value org.apache.hadoop.hive.ql.hooks.ATSHook,org.apache.atlas.hive.hook.HiveHook. The 高级 hive-atlas-application.properties section shows the atlas.hook.hive.maxThreads and atlas.hook.hive.minThreads configurations, both set to 1. The 自定义 hive-site section shows the atlas.rest.address configuration with the value http://ocdpbroker.jcloud.local:21000.

Apache Atlas 需要依赖以下 HDP 组件：

- HBase：Titan 默认使用 HBase 存储元数据
- Ambari infra/Solr：Titan 默认使用 Solr 存储元数据索引
- Kafka：Apache Atlas 使用 Kafka 作为消息队列，实现 hook 和元数据通知事件的消费者之间的通信

部署完 Apache Atlas 之后，可以通过脚本导入 Atlas 自带的示例数据：

```
[root@ocdpbroker ~]# /usr/hdp/2.5.0.0-1245/atlas/bin/quick_start.py http://ocdpbroker.jcloud.local:21000
Enter username for atlas :-
admin
Enter password for atlas :-
admin
```

Apache Atlas 支持配置多个 Atlas Web 服务实例来实现自动故障转移(failover)，多个服务实例之间采取主动/被动模式。其中一个实例将被自动选择为“活动”实例来为用户请求提供服务。其他人将自动被视为“被动”。如果“活动”实例由于故意停止或由于意外故障而变得不可用，则其他实例之一将自动选为“活动”实例，并开始为用户请求提供服务。

“活动”实例是能够正确响应用户请求的唯一实例，其可以创建，删除，修改或响应元数据对象上的查询。“被动”实例将接受用户请求，但会使用 HTTP 重定向将其重定向到当前已知的“活动”实例。

多个 Apache Atlas Web 服务实例之间使用 Zookeeper 进行协调。

当配置为高可用性模式时，具有以下优势：

- 在维护间隔期间不间断服务：如果需要停用 Atlas Web 服务的活动实例进行维护，则另一个实例将自动变为活动状态并可以为请求提供服务。
- 在意外故障事件中的不间断服务：如果由于软件或硬件错误，Atlas Web 服务的活动实例失败，另一个实例将自动变为活动状态并可以为请求提供服务。

Apache Atlas 的高可用性目前只支持手动配置，Ambari 尚不支持添加新的 Atlas Metadata Server。

## 类型系统

Atlas 允许用户为他们想要管理的元数据对象定义一个模型。该模型由称为“类型” (type) 的定义组成。被称为“实体” (entities) 的“类型”实例表示被管理的实际元数据对象。由 Atlas 管理的所有元数据对象（例如 Hive 表）都使用类型进行建模，并表示为实体。

**Type**：Atlas 中的“类型”定义了如何存储和访问特定类型的元数据对象。

类型表示了所定义元数据对象的一个或多个属性集合。具有开发背景的用户可以将“类型”理解成面向对象的编程语言的“类”定义的或关系数据库的“表模式”。类型具有元类型，元类型表示 Atlas 中此模型的类型：

- 基本元类型：Int, String, Boolean 等
- 枚举元类型
- 集合元类型：例如 Array, Map
- Class, Struct, Trait

**Entities**：Atlas 中的“实体”是类“类型”的特定值或实例，因此表示真实世界中的特定元数据对象。回顾我们的面向对象编程语言的类比，“实例”是某个“类”的“对象”。

**Attributes**：Atlas 中的属性还有一些属性，其定义了与类型系统相关的更多

概念，包括：

- isComposite - 是否复合
- isIndexable - 是否索引

- isUnique - 是否唯一

- multiplicity - 指示此属性是（必需的 / 可选的 / 还是可以是多值）的

Atlas 提供了一些预定义的系统类型：

**Referenceable**：此类型表示可使用名为 qualifiedName 的唯一属性搜索的所有实体

**Asset**：此类型包含名称，说明和所有者等属性

**Infrastructure**：此类型扩展了 Referenceable 和 Asset，通常可用于基础设施元数据对象（如群集，主机等）的常用超类型

**DataSet**：此类型扩展了 Referenceable 和 Asset。在概念上，它可以用于表示存储数据的类型。在 Atlas 中，hive 表，Sqoop RDBMS 表等都是从 DataSet 扩展的类型。扩展 DataSet 的类型可以期望具有模式，它们将具有定义该数据集的属性的属性。例如，hive\_table 中的 columns 属性。另外，扩展 DataSet 的实体类型的实体参与数据转换，这种转换可以由 Atlas 通过 lineage（或 provenance）生成图形。

**Process**：此类型扩展了 Referenceable 和 Asset。在概念上，它可以用于表示任何数据变换操作。例如，将原始数据的 hive 表转换为存储某个聚合的另一个 hive 表的 ETL 过程可以是扩展过程类型的特定类型。流程类型有两个特定的属性，输入和输出。

## 元数据搜索与血缘

Atlas 支持使用以下 2 种方式搜索元数据：



## Search using DSL

### DSL Examples

For the model, Asset - attributes name, owner, description DB - supertype Asset - attributes clusterName, parameters, comment Column - extends Asset - attributes type, comment Table - supertype Asset - db, columns, parameters, comment Traits - PII, Log Data

DSL queries: \* from DB

- DB where name="Reporting" select name, owner
- DB where name="Reporting" select name, owner orderby name
- DB where name="Reporting" select name limit 10
- DB where name="Reporting" select name, owner limit 10 offset 0
- DB where name="Reporting" select name, owner orderby name limit 10 offset 5
- DB where name="Reporting" select name, owner orderby name desc limit 10 offset 5
- DB has name
- DB is JdbcAccess
- Column where Column isa PII
- Table where name="sales\_fact", columns
- Table where name="sales\_fact", columns as column select column.name, column.dataType, column.comment
- DB groupby(owner) select owner, count()
- DB groupby(owner) select owner, max(name)
- DB groupby(owner) select owner, min(name)
- from Person select count() as 'count', max(Person.age) as 'max', min(Person.age)
- `Log Data`

## Full-text search

- Atlas 也支持 lucene 风格的全文检索

Apache atlas 的其中一个核心特性就是可以追溯数据湖(Data Lake)中数据的血缘关系并以可视化的方式呈现, 使用户能够快速了解数据的生命周期, 并能够知晓自己的数据时从哪里来以及和数据湖中的那些数据具有关联关系。

参考下面的例子, 创建两张 hive 表, 然后通过表的 join 创建一张新表:

- 创建一张 hive 表存储以数组形式存储 tweet 文本中的单词

```
CREATE TABLE words_array AS SELECT tweet_id AS id, split(text, ' ') AS words
FROM tweets;
```

- 创建一张 hive 表将文本中的数组切分成单独的单词

```
CREATE TABLE tweet_word AS SELECT id AS id, word FROM words_array
LATERAL VIEW explode(words) w as word;
```

- 通过对上述表的 Join 操作创建新的表

```
CREATE TABLE word_join AS SELECT tweet_word.id, tweet_word.word,
sentiment_dictionary.rating FROM tweet_word LEFT OUTER JOIN
sentiment_dictionary ON (tweet_word.word=sentiment_dictionary.word);
```

在 Atlas 中, 上述操作生成的 word\_join 表的血缘关系图如下所示:



## 基于标签的安全策略

Atlas 通过与 Ranger 集成，为 Hadoop 提供了基于标签(Tag)的动态访问权限控制，通过控制与资源关联的标签而非资源本身可以为权限控制模型提供诸多便利：

- 将资源分类从权限控制中分离出来，不同 Hadoop 组件的资源(比如 HDFS 目录，Hive 表，HBase 表)中的同一类数据(比如社保账号/信用卡帐号)可以被打上同一个标签，以统一的权限来控制访问
- 如果 Hadoop 资源被打上标签，那么与标签相关的权限将被自动赋予该资源
- 单独的访问控制策略可以应用于不同的 Hadoop 组件的资源，而不再需要为每一个组件的资源都创建单独的策略

Ranger 0.6 版本引入了一个新的服务(ranger-tagsync)用来同步 Atlas 的 tag，ranger-tagsync 通过监听 Kafka topic(ATLAS\_ENTITIES)的消息变更来接受来自 Atlas 的 tag(以及 Taxonomy Terms)详细信息；当 Atlas 的 tag(以及 Taxonomy Terms)在 atlas 里被增加，修改或删除时，ranger-tagsync 会接受来自 Kafka 的通知并更新 Ranger 的数据库(ranger.x\_tag\*)。

[Ranger Admin](#) ✔ Started 没有告警

[Ranger Usersync](#) ✔ Started 1 alert

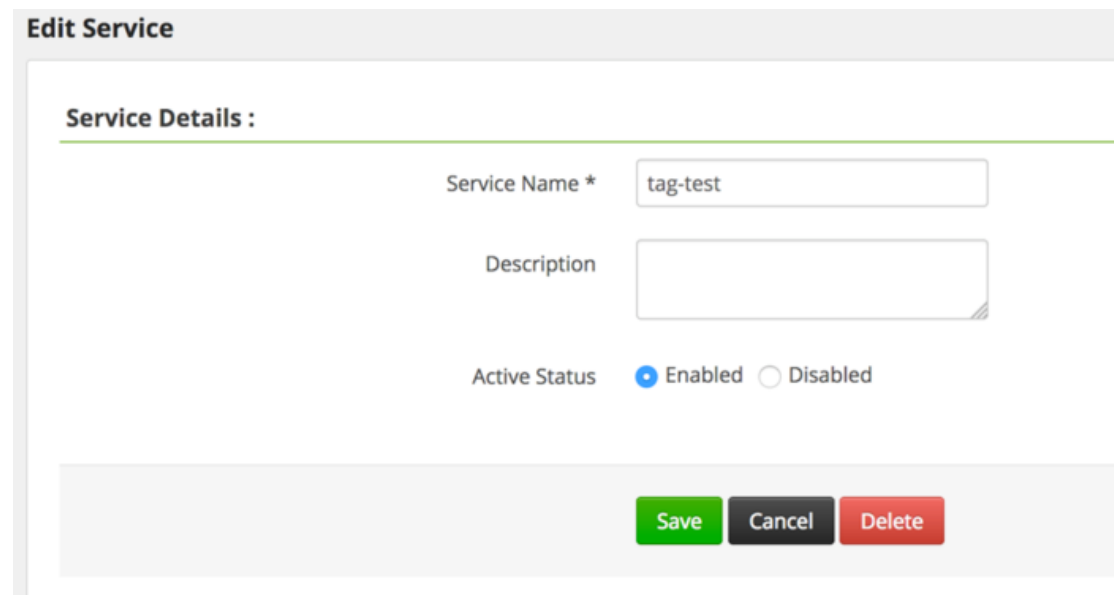
[Ranger Tagsyncs](#) 1/1 已启动

通用 Ranger Admin UI 创建基于标签的策略需要如下步骤：

- 创建 tag 服务实例(tag 服务实例可以创建多个以归类不同集群的基于标签策

略)

- Login to Ranger Admin
- Select menu: Access Manager è Tag Based Policies
- Click the + icon next to TAG
- In 'Service Name' field, enter tagdev and click 'Add'



**Edit Service**

**Service Details :**

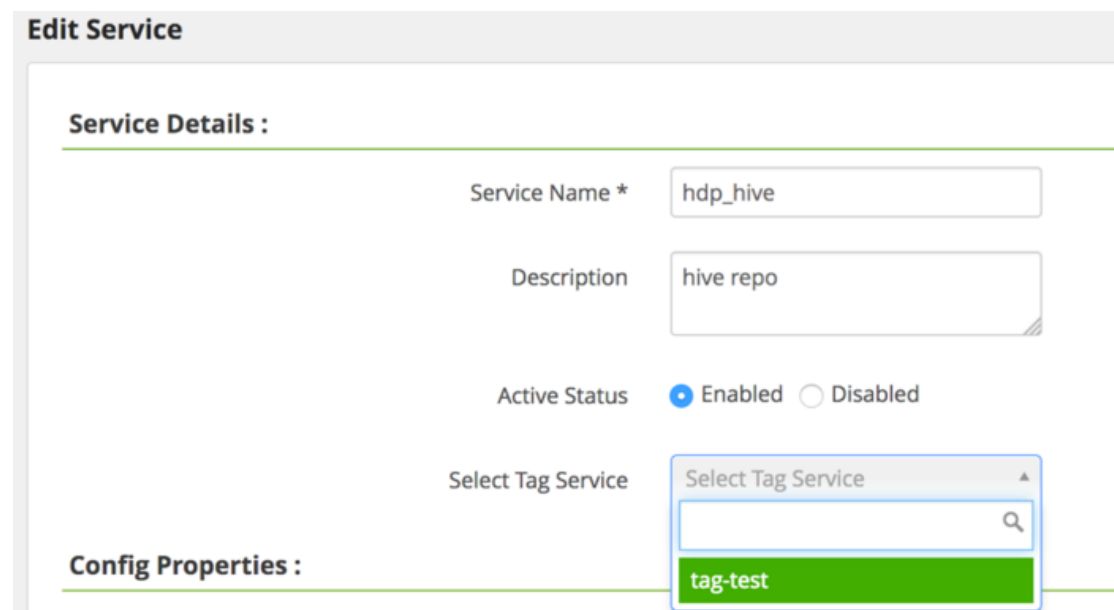
Service Name \*

Description

Active Status ☒ Enabled ☐ Disabled

- Ranger 的各个组件的服务实例需要被更新以实施指定的 tag 服务实例所提供的基于标签的访问控制策略，以 hive 为例：

- Login to Ranger Admin
- Select menu: Access Manager è Resource Based Policies
- Click on the edit icon next to your hive service instance, like hdp\_hive,
- In 'Select Tag Service' field, select tag-test and click 'Save'



**Edit Service**

**Service Details :**

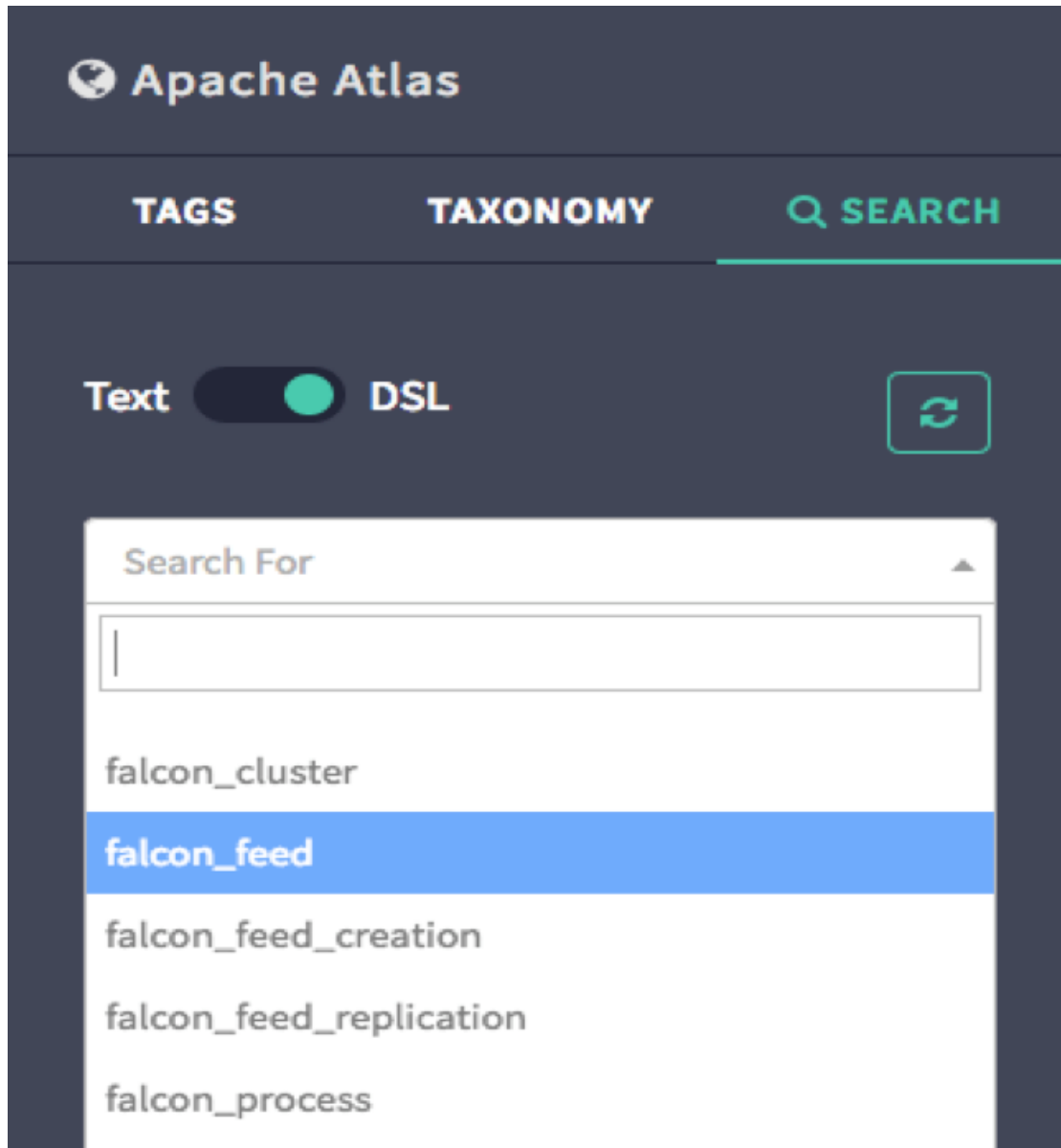
Service Name \*

Description

Active Status ☒ Enabled ☐ Disabled

Select Tag Service

**Config Properties :**



- Atlas 可以对 Falcon 元数据进行分类(tag/taxonomy)

| <input type="checkbox"/> | Name       | Description                | Owner | Tags   | Terms  |
|--------------------------|------------|----------------------------|-------|--|--|
| <input type="checkbox"/> | feedTest01 |                            |       | <input data-bbox="933 1547 949 1570" type="button" value="+"/> | <input data-bbox="1045 1547 1117 1570" type="button" value="Assign Term"/> |
| <input type="checkbox"/> | feedTest01 | Feed creation - feedTest01 |       | <input data-bbox="933 1597 949 1619" type="button" value="+"/> | <input data-bbox="1045 1597 1117 1619" type="button" value="Assign Term"/> |
| <input type="checkbox"/> | feedTest01 |                            | admin | <input data-bbox="933 1646 949 1668" type="button" value="+"/> | <input data-bbox="1045 1646 1117 1668" type="button" value="Assign Term"/> |

- Atlas 可以展示 Falcon 元数据的血缘关系

feedTest01

Tags: 

LINEAGE



## 总结

Apache Atlas 为 Hadoop 集群提供了包括数据分类、集中策略引擎、数据血缘、安全和生命周期管理在内的元数据治理核心能力，其与 Apache Falcon，Apache Ranger 相互整合可以形成完整的数据治理解决方案。但是 Atlas 目前还是 Apache 孵化项目，尚未成熟，有待发展。

Atlas 目前还存在以下一些需要改进之处：

- 缺乏对元数据的全局视图，对元数据的血缘追溯只能够展示具体某张表或某个 SQL 的生命周期(其前提是用户必须对 Hadoop 的元数据结构十分清楚，才能够通过 Atlas 的查询语句去定位自己需要了解的表)
- 0.8 以前的版本，对元数据只能进行只读操作，例如只能展示 Hive 的表但是不能创建新表
- 与 Hadoop 各组件的集成尚待完善，例如 Atlas 对 Hive 的元数据变更操作的捕获只支持 hive CLI，不支持 beeline/JDBC

## Reference

- <http://atlas.apache.org/>
- <https://zh.hortonworks.com/apache/atlas/>

- 在” Tag Based Policies” 页面创建基于策略的访问控制策略，例如：

**Create Policy**

**Policy Details :**

Policy Name \* PII enabled

TAG \* PII ← Tag Name

Description Restrict access to resources tagged with PII

Audit Logging YES

**Allow Conditions :** Allow Conditions

Select Group audit Select User Select User Policy Conditions Add Conditions Component Permissions HIVE

Exceptions : show

**Deny Conditions :** Deny Conditions

Select Group public Select User Select User Policy Conditions Add Conditions Component Permissions HIVE

Exceptions : show

**Deny Exceptions :** Deny Exceptions

Select Group audit Select User Select User Policy Conditions Add Conditions Component Permissions HIVE

Add Cancel

## 集成 Apache Falcon

- Atlas 可以提供对 Falcon 元数据(cluster/feed/process)的存储和搜索
  - Atlas 的 Falcon Hook 监听 Falcon 的元数据变更(比如 falcon feed/process 的创建，更改，删除)并将这些元数据变更详细信息写入 Kafka 的 topic 中，Atlas metadata server 从 Kafka 接收到 Falcon 的元数据然后作为 entity 更新到 Titan 数据库中
  - 用户通过 Atlas Admin UI 可以通过 DSL 或全文方式去搜索 Falcon 的元数据