Hypertheasu Data lake basic tutorial

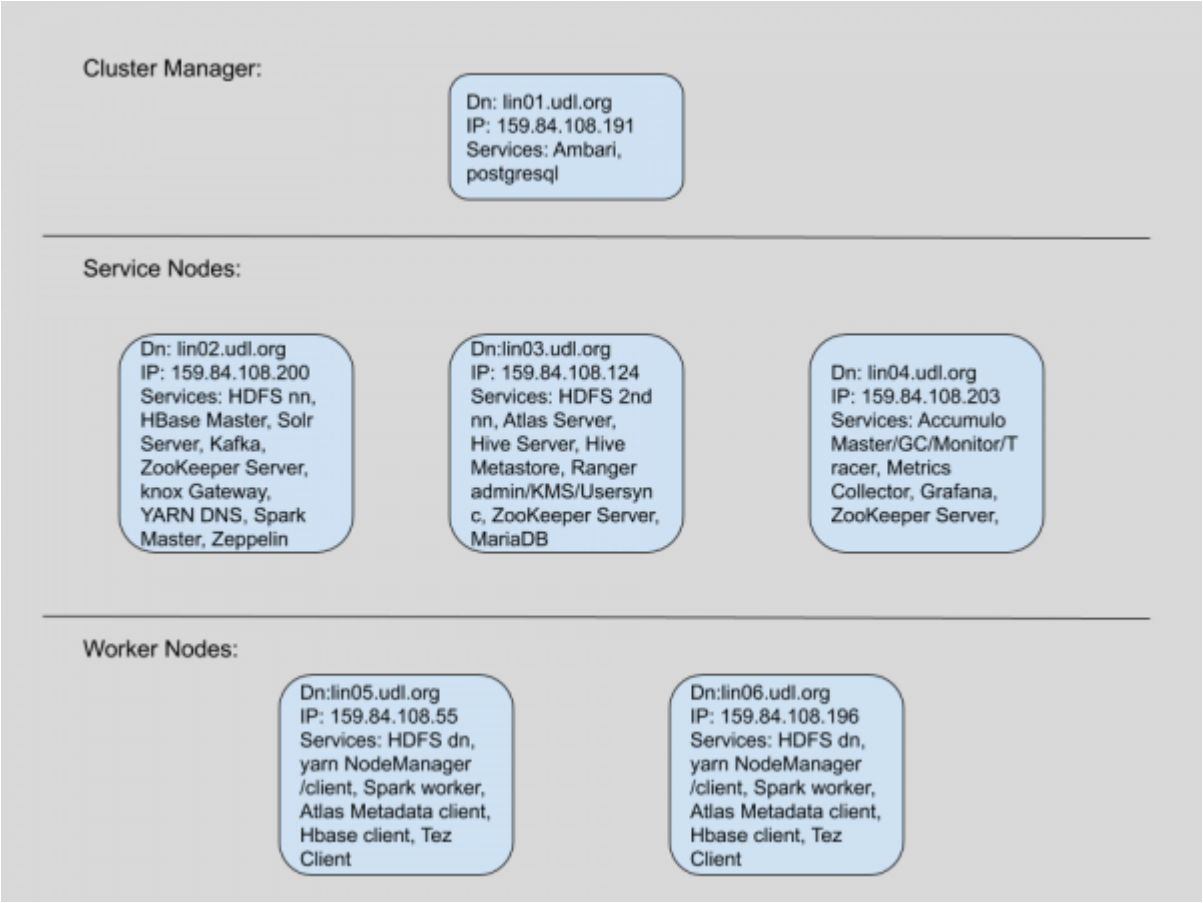
This tutorial is for new comer who wants to understand the basic features of **Hypertheasu Data lake**.

It has the following contents:

- 1. Architecture and key components
- 2. Ingest data
  - 2.1 Ingest data via ambari-webGui
  - 2.2 Ingest data via hdfs-cli
  - 2.3 Ingest data via sqoop
- 3. Data storage
  - 3.1 Name node
  - 3.2 Data node
  - 3.3 Hypertheasu Data lake hdfs implementation
- 4. Data processing and analytics
  - 4.1 Hive cli
  - 4.2 Spark cli
  - 4.3 Hive/Spark via Zeppelin
- 5. Data management
  - 5.1 Create metadata entities for data
  - 5.2 Search data by using metadata
  - 5.3 metadata properities and lineage
  - 5.4 Thesaurus

1. Architecture and key components

Below shows the HyperTheasu data lake deployment architecture in Eric Vsphere Cloud



Below are the key service in the data lake:

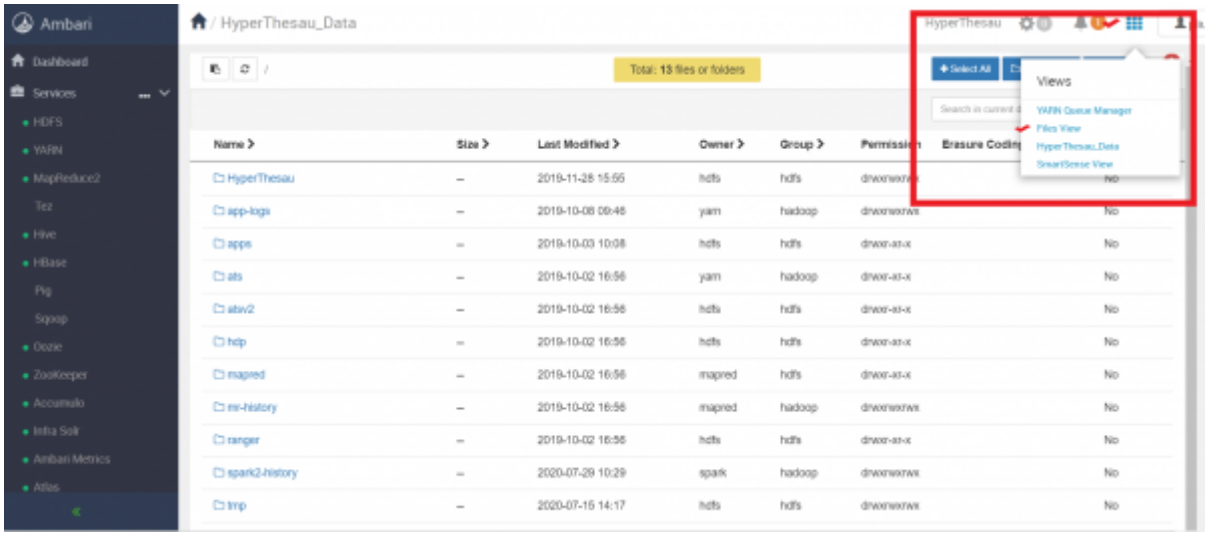
- Ingestion: Sqoop, ambari-webGui, hdfs-cli
- Storage: HDFS(namenode, datanode)
- data process and analytics: Hive, spark (via zeppelin or cli)
- data management: Atlas(via HBase, Solr, Kafka)

2. Ingest data

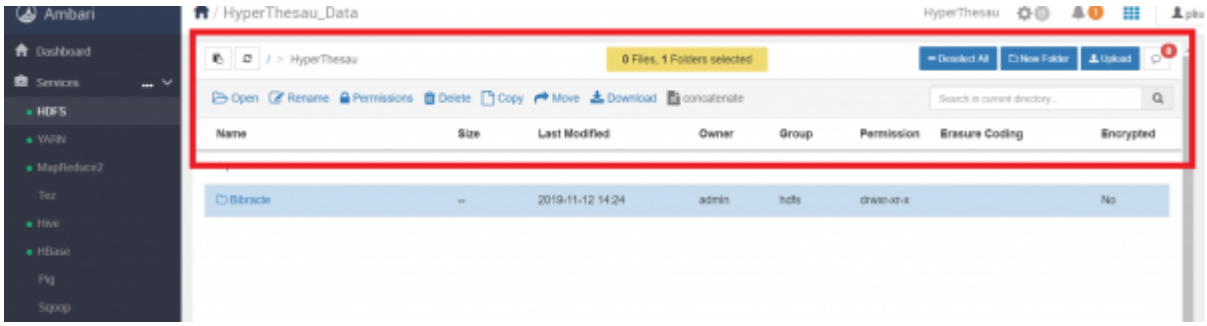
2.1 Ingest data via ambari-webGui

Open the ambari web interface http://lin01.udl.org:8080/

Click on the button on the top right, then click on the Files View



In this interface, you can upload, download data, or change data acl.



2.2 Ingest data via hdfs-cli

To use hdfs-cli, you need to install it first on your pc. This tool has been installed on server lin02.udl.org Use ssh to connect to lin02.udl.org, then you can use hdfs-cli directly

```
# list all data in the data lake
[pliu@lin02 ~]$ hdfs dfs -ls /
Found 13 items
drwxrwxrwx   - hdfs   hdfs            0 2019-11-28 15:55 /HyperThesau
drwxrwxrwt   - yarn   hadoop          0 2019-10-08 09:46 /app-logs
drwxr-xr-x   - hdfs   hdfs            0 2019-10-03 10:08 /apps
drwxr-xr-x   - yarn   hadoop          0 2019-10-02 16:56 /ats
drwxr-xr-x   - hdfs   hdfs            0 2019-10-02 16:56 /atsv2
drwxr-xr-x   - hdfs   hdfs            0 2019-10-02 16:56 /hdp
drwxr-xr-x   - mapred hdfs            0 2019-10-02 16:56 /mapred
drwxrwxrwx   - mapred hadoop          0 2019-10-02 16:56 /mr-history
drwxr-xr-x   - hdfs   hdfs            0 2019-10-02 16:56 /ranger
drwxrwxrwx   - spark  hadoop          0 2020-07-29 10:44 /spark2-history
drwxrwxrwx   - hdfs   hdfs            0 2020-07-15 14:17 /tmp
drwxr-xr-x   - hdfs   hdfs            0 2019-10-07 10:31 /user
drwxr-xr-x   - hdfs   hdfs            0 2019-10-02 16:58 /warehouse

# upload data
[pliu@lin02 tmp]$ hdfs dfs -put test.txt /tmp/
```

```
# download data
[pliu@lin02 tmp]$ hdfs dfs -get /tmp/test.txt .

# change data owner
[pliu@lin02 tmp]$ hdfs dfs -chown pliu:pliu /tmp/test.txt

# change data acl
[pliu@lin02 tmp]$ hdfs dfs -chmod 0777 /tmp/test.txt

# delete data
[pliu@lin02 tmp]$ hdfs dfs -rm -skipTrash /tmp/test.txt
Deleted /tmp/test.txt
```

2.3 Ingest data via sqoop

Sqoop reads the database and write to hdfs/hive. It must have the right to do so. So first we need to check if Sqoop can read the database.

Note: you need to have the appropriate JDBC drivers to run the commands.

```
#for mysql server
$ sqoop list-tables --connect jdbc:mysql://localhost:3306/retail_db --
username root --password hadoop

#for postgresql server
sqoop list-tables --connect jdbc:postgresql://127.0.0.1:5432/northwind --
username pliu -P
```

2.3.1 Import a table into hdfs

Sqoop supports four formats:

- Text file format - Using command argument **as-textfile** (default)
- Sequence file format - Using command argument **as-sequencefile** (mapreduce default)
- Avro file format - Using command argument **as-avrofile** (row oriented)
- Parquet file format Using command argument **as-parquetfile** (column oriented)

To import a mysql table into hdfs [database name is retail_db, table name is categories] with default file format

```
$ sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username
root --password hadoop --table categories --target-dir
/tmp/sqoop_test/categories

# check the imported data
$ hdfs dfs -cat /tmp/sqoop_test/categories/part-m-*
```

```
# The default file format of sqoop import is textfile(csv), that's why we
can use cat to show it.

# If we don't specify --target-dir, sqoop will create a dir with the name of
table (e.g. categories in the root dir of hdfs)
```

2.3.2 Import a table into hive

To import a mysql table into hive

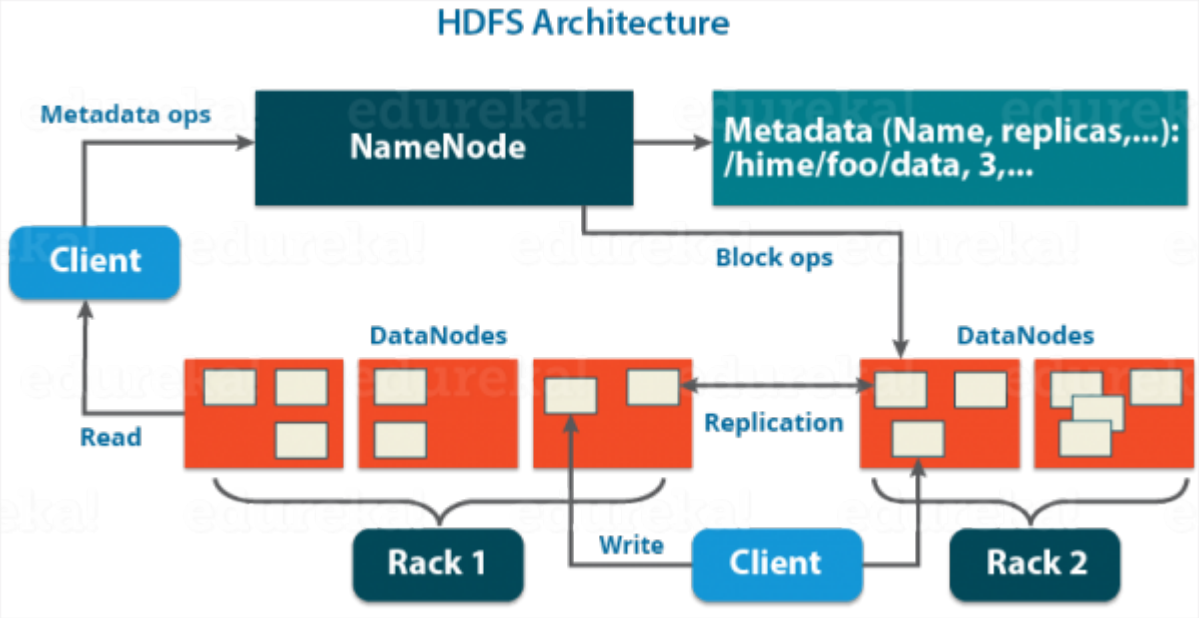
```
# By default, if nothing is specified, sqoop will use the mysql table name
as the hive table name, and the
# table will be stored in the hive default database(db with name default)
$ sqoop import --connect jdbc:mysql://localhost:3306/retail_db --username
root --password hadoop --table categories --hive-import

# If you want to change the hive table-name, you can use option --hive-
table. If you want to also give a
# database name, you can use <db_name>.<table_name> (But this works only for
hive 2)
$ sqoop import --connect jdbc:mysql://lin03.udl.org:3306/retail_db --
username hive -P --table categories --hive-import --hive-table
retail_db.categories


# In hive 3.*, we can't use the expression database.table anymore. We have
to use --hive-database to express database name , and --hive-table to
express table name. As a result, the above query should be like this:
$ sqoop import --connect jdbc:mysql://lin03.udl.org:3306/retail_db --
username hive -P --table categories --hive-import --hive-table categories --
hive-database retail_db
```

3. Data storage

Hypertheasu Data lake uses hdfs as data storage service. Hdfs is a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster. It has two types of node:

- name node
- data node

Hdfs cuts data into blocks, these blocks are stored across a cluster of one or several machines. HDFS Architecture follows a Master/Slave Architecture, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes).



3.1 Name node

NameNode is the master node in the Apache Hadoop HDFS Architecture that maintains and manages the blocks present on the DataNodes (slave nodes). NameNode is a very highly available server that manages the File System Namespace and controls access to files by clients. The HDFS architecture is built in such a way that the user data never resides on the NameNode. The data resides on DataNodes only.

Functions of NameNode:

- It is the master daemon that maintains and manages the DataNodes (slave nodes)
- It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata:
    - FsImage: It contains the complete state of the file system namespace since the start of the NameNode.
    - EditLogs: It contains all the recent modifications made to the file system with respect to the most recent FsImage.
- It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
- It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
- It keeps a record of all the blocks in HDFS and in which nodes these blocks are located.
- The NameNode is also responsible to take care of the replication factor of all the blocks which we will discuss in detail later in this HDFS tutorial blog.
- In case of the DataNode failure, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes.

3.2 Data node

DataNodes are the slave nodes in HDFS. The DataNode is a block server that stores the data in the

local file ext3 or ext4.

Functions of DataNode:

- These are slave daemons or process which runs on each slave machine.
- The actual data is stored on DataNodes.
- The DataNodes perform the low-level read and write requests from the file system's clients.
- They send heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds.

3.3 Hypertheasu Data lake hdfs implementation

In hypertheasu Data lake, we implement the hdfs as followed:

- lin02.udl.org: primary name node
- lin03.udl.org: secondary name node (for HA)
- lin05.udl.org: data node
- lin06.udl.org: data node

4. Data processing and analytics

Hypertheasu Data lake uses two tools(i.e. Hive and Spark) to process and analyze data.

We have a Command-line interface and web interface to use them.

4.1 Hive cli

The recommended hive cli is called beeline. It's already installed on lin02.udl.org. Follow the below example to run beeline.

```
[pliu@lin02 ~]$ beeline
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/hdp/3.1.4.0-315/hive/lib/log4j-slf4j-
impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/hdp/3.1.4.0-315/hadoop/lib/slf4j-
log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
explanation.
SLF4J: Actual binding is of type
[org.apache.logging.slf4j.Log4jLoggerFactory]
Connecting to
jdbc:hive2://lin02.udl.org:2181,lin03.udl.org:2181,lin04.udl.org:2181/defaul
t;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2
Enter username for
jdbc:hive2://lin02.udl.org:2181,lin03.udl.org:2181,lin04.udl.org:2181/defaul
t: hive
Enter password for
```

```
jdbc:hive2://lin02.udl.org:2181,lin03.udl.org:2181,lin04.udl.org:2181/defaul
t: ****
20/07/29 12:34:21 [main]: INFO jdbc.HiveConnection: Connected to lin03:10000
Connected to: Apache Hive (version 3.1.0.3.1.4.0-315)
Driver: Hive JDBC (version 3.1.0.3.1.4.0-315)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 3.1.0.3.1.4.0-315 by Apache Hive
0: jdbc:hive2://lin02.udl.org:2181,lin03.udl.> show tables;
INFO  : Compiling command(queryId=hive_20200729123426_7929676b-4c3a-49ee-
be5f-b111e61ccf01): show tables
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema:
Schema(fieldSchemas:[FieldSchema(name:tab_name, type:string, comment:from
deserializer)], properties:null)
INFO  : Completed compiling
command(queryId=hive_20200729123426_7929676b-4c3a-49ee-be5f-b111e61ccf01);
Time taken: 0.018 seconds
INFO  : Executing command(queryId=hive_20200729123426_7929676b-4c3a-49ee-
be5f-b111e61ccf01): show tables
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing
command(queryId=hive_20200729123426_7929676b-4c3a-49ee-be5f-b111e61ccf01);
Time taken: 0.016 seconds
INFO  : OK
+-------------+
|  tab_name   |
+-------------+
| categories  |
+-------------+
1 row selected (0.203 seconds)
0: jdbc:hive2://lin02.udl.org:2181,lin03.udl.> select * from catalog
catalog        catalog_name
0: jdbc:hive2://lin02.udl.org:2181,lin03.udl.> select * from categories;
INFO  : Compiling
command(queryId=hive_20200729123455_86238637-9e79-4a2f-8567-fcdf79871c01):
select * from categories
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema:
Schema(fieldSchemas:[FieldSchema(name:categories.category_id, type:int,
comment:null), FieldSchema(name:categories.category_department_id, type:int,
comment:null), FieldSchema(name:categories.category_name, type:string,
comment:null)], properties:null)
INFO  : Completed compiling
command(queryId=hive_20200729123455_86238637-9e79-4a2f-8567-fcdf79871c01);
Time taken: 0.1 seconds
INFO  : Executing
command(queryId=hive_20200729123455_86238637-9e79-4a2f-8567-fcdf79871c01):
select * from categories
INFO  : Completed executing
command(queryId=hive_20200729123455_86238637-9e79-4a2f-8567-fcdf79871c01);
```

```
Time taken: 0.01 seconds
INFO  : OK
+--------------------------+-------------------------------------+------------
---------------+
| categories.category_id  | categories.category_department_id  |
categories.category_name  |
+--------------------------+-------------------------------------+------------
---------------+
| 1                        | 2                                   | Football
|
| 2                        | 2                                   | Soccer
|
| 3                        | 2                                   | Baseball &
Softball       |
| 4                        | 2                                   | Basketball
|
| 5                        | 2                                   | Lacrosse
|
```

4.2 Spark cli

The spark-shell client is already installed on lin02.udl.org. To use it, follow the below example

```
[pliu@lin02 tmp]$ sudo -u hdfs spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
Spark context Web UI available at http://lin02:4040
Spark context available as 'sc' (master = yarn, app id =
application_1594025881974_0010).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.2.3.1.4.0-315
      /_/
Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java
1.8.0_112)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

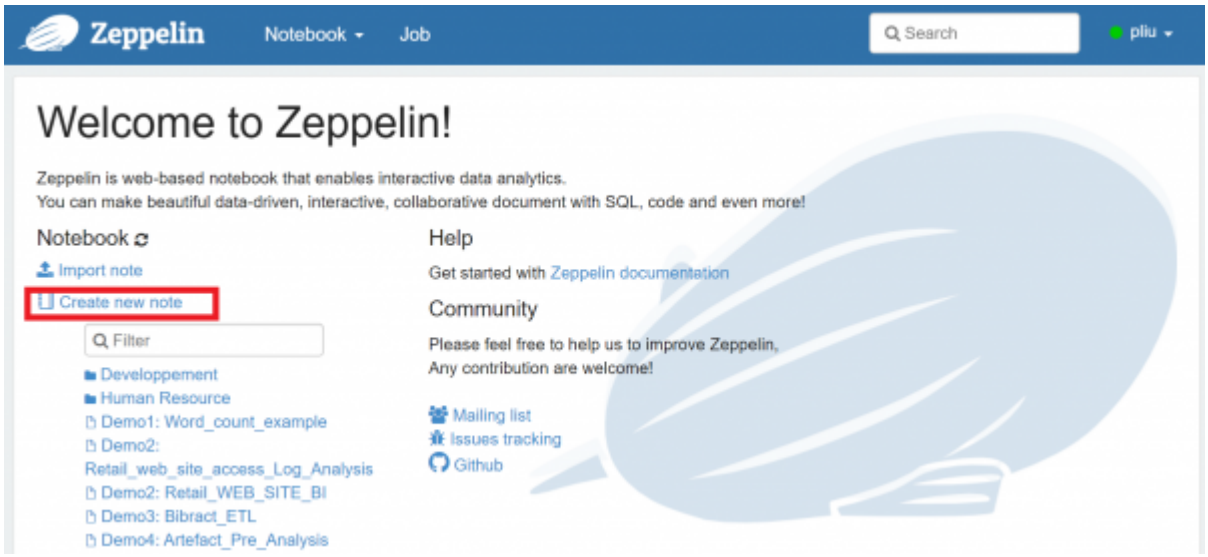
4.3 Hive/Spark via Zeppelin

Zeppelin is a note which can connect to a Hive server or Spark server. It's already installed on lin02.udl.org. You can access it by using the following url:

http://lin02.udl.org:9995/#/

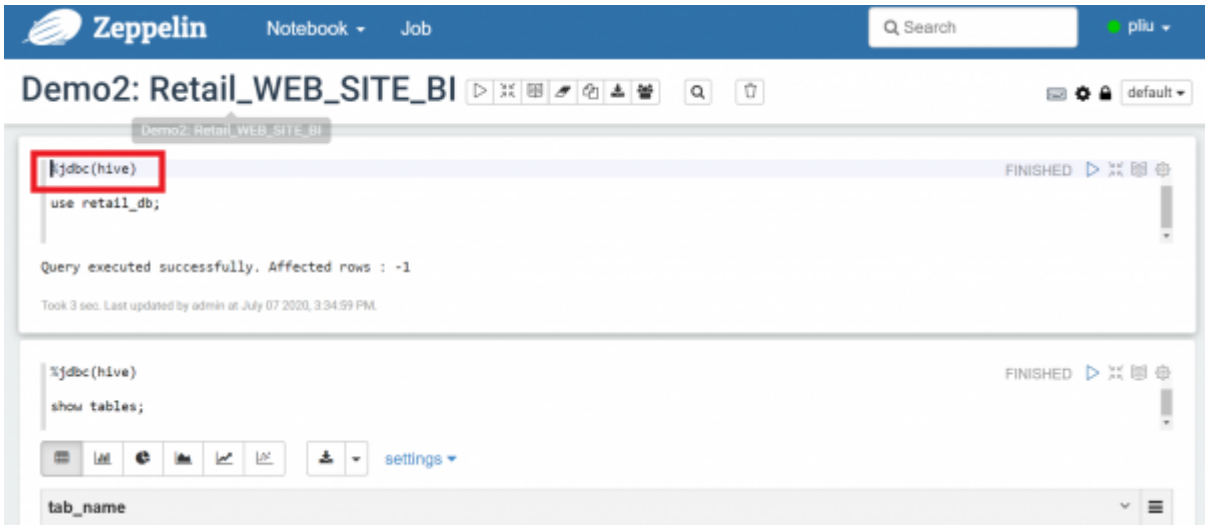
To add a new user or change the user password. See my other documentations on zeppelin security.

Inside zeppelin, you can add a new note, see the below example



4.3.1 Hive via Zeppelin

To connect to a Hive server, you need to specify that you want to use hive interpreter in your notebook. Todo so, you need to add %jdbc(hive)



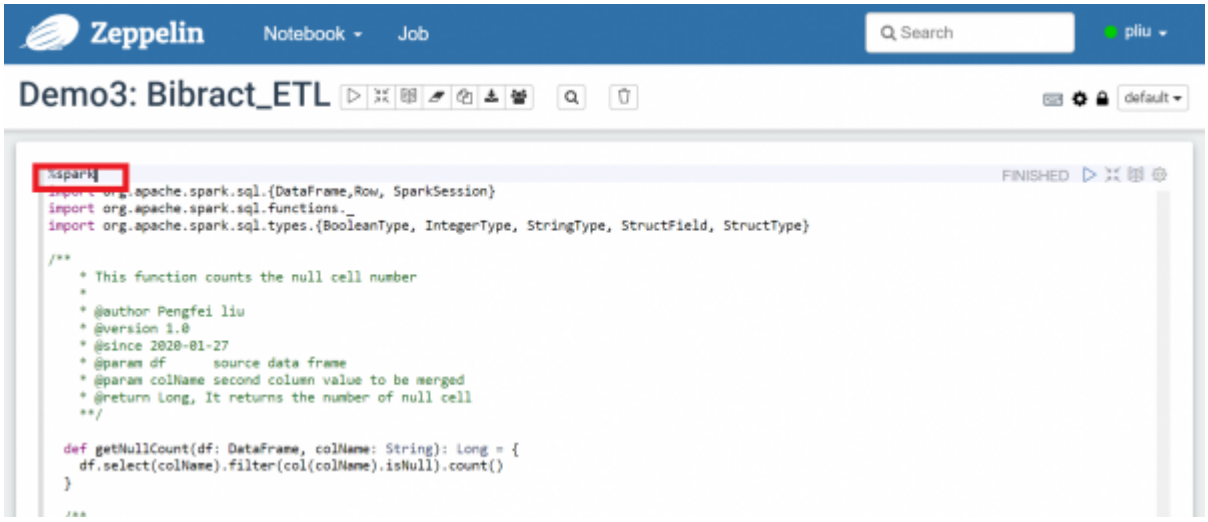
4.3.1 Spark via Zeppelin

As spark provides three api, so zeppelin provides three interpreters for spark:

- Spark Scala
- PySpark
- sparkR

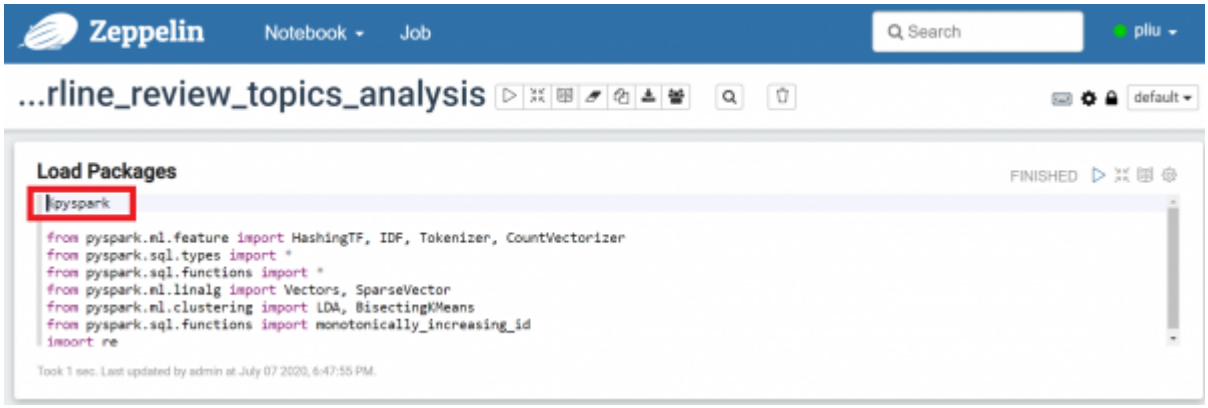
Spark scala

To use spark scala, just add %spark



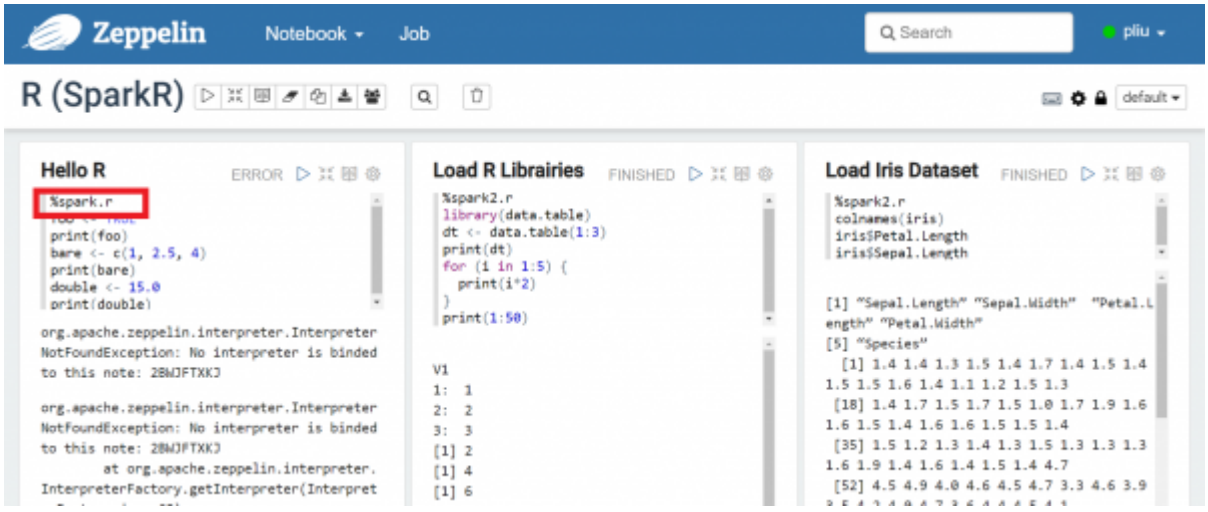
PySpark

To use pyspark, just add %pyspark



SparkR

To use sparkR, just add %spark2.r(%spark.r for spark version 1, not default setup)



5. Data management

Hypertheasu Data lake uses atlas for data management. It uses Hbase to store the metadata, and solr to index and search the metadata

It's installed on lin03.udl.org. To access it, use http://lin03.udl.org:21000/

To add a new user or change the user password. See my other documentations on Atlas security.

5.1 Create metadata entities for data

Before adding a metadata entity, we need to first create its type. Because all metadata entities in Atlas must have a type which defines the list of attributes of metadata

5.1.1 Create metadata types

To illustrate how to create a new type in atlas, we use the following example. Suppose we store all our data on a network storage which called gpfs.

We write the following data type definiton

typeDef_datafile.json

```
{
  "structDefs": [
    {
      "category": "STRUCT",
      "name": "schema_col",
      "description": "column definition for schema",
      "typeVersion": "1.0",
      "attributeDefs": [
        {
          "name": "col",
          "typeName": "string",
          "isOptional": false,
          "cardinality": "SINGLE",
          "valuesMinCount": 1,
          "valuesMaxCount": 1,
          "isUnique": false,
          "isIndexable": false
        },
        {
          "name": "data_type",
          "typeName": "string",
          "isOptional": false,
          "cardinality": "SINGLE",
```

```
          "valuesMinCount": 1,
          "valuesMaxCount": 1,
          "isUnique": false,
          "isIndexable": false
        },
        {
          "name": "required",
          "typeName": "boolean",
          "isOptional": false,
          "cardinality": "SINGLE",
          "valuesMinCount": 1,
          "valuesMaxCount": 1,
          "isUnique": false,
          "isIndexable": false
        }
      ]
    }
  ],
  "entityDefs": [
    {
      "superTypes": [
        "DataSet"
      ],
      "category": "ENTITY",
      "name": "GPFSDataFile",
      "description": "a type definition for a file in gpfs which
contains data, this could a file that needs to be processed or it can
be an output (ex: extracts)",
      "typeVersion": "1.0",
      "attributeDefs": [
        {
          "name": "file_name",
          "typeName": "string",
          "isOptional": false,
          "cardinality": "SINGLE",
          "valuesMinCount": 1,
          "valuesMaxCount": 1,
          "isUnique": true,
          "isIndexable": true
        },
        {
          "name": "parent_dir",
          "typeName": "string",
          "isOptional": false,
          "cardinality": "SINGLE",
          "valuesMinCount": 1,
          "valuesMaxCount": 1,
          "isUnique": true,
          "isIndexable": true
        },
        {
```

```
          "name": "user",
          "typeName": "string",
          "isOptional": false,
          "cardinality": "SINGLE",
          "valuesMinCount": 1,
          "valuesMaxCount": 1,
          "isUnique": false,
          "isIndexable": false
        },
        {
          "name": "group",
          "typeName": "string",
          "isOptional": false,
          "cardinality": "SINGLE",
          "valuesMinCount": 1,
          "valuesMaxCount": 1,
          "isUnique": false,
          "isIndexable": false
        },
        {
          "name": "permission",
          "typeName": "string",
          "isOptional": true,
          "cardinality": "SINGLE",
          "valuesMinCount": 1,
          "valuesMaxCount": 1,
          "isUnique": false,
          "isIndexable": false
        },
        {
          "name": "creation_time",
          "typeName": "date",
          "isOptional": false,
          "cardinality": "SINGLE",
          "valuesMinCount": 1,
          "valuesMaxCount": 1,
          "isUnique": false,
          "isIndexable": false
        },
        {
          "name": "format",
          "typeName": "string",
          "isOptional": false,
          "cardinality": "SINGLE",
          "valuesMinCount": 1,
          "valuesMaxCount": 1,
          "isUnique": false,
          "isIndexable": false
        },
        {
```

```
          "name": "schema",
          "typeName": "array<schema_col>",
          "isOptional": true,
          "cardinality": "SINGLE",
          "valuesMinCount": 1,
          "valuesMaxCount": 1,
          "isUnique": false,
          "isIndexable": false
        }
      ]
    }
  ]
}
```

We could notice that the above type definition has two parts. The second part is the main type defnition, It starts with key word “entityDefs” and contains the following information:

- SuperTypes: Every customer types in Atlas must have one
- Category: A meta data type definition has the Catefory “ENTITY”,
- name: Name of the meta data type
- description:
- typeVersion:
- attributeDefs: list of the attributes definition, every attributes must have a type. The predefined types almost cover every thing.
    - Primitive metatypes: boolean, byte, short, int, long, float, double, biginteger, bigdecimal, string, date
    - Enum metatypes
    - Collection metatypes: array, map
    - Composite metatypes: Entity, Struct, Classification, Relationship

If you want something special, you can also add new types for attributes. In the above example, I add a new struct type with name schema_col.

For more informaiton on how to define metadata types in Atlas, please visit https://atlas.apache.org/TypeSystem.html

Now we can use the bulk add typedef api to add the above type definition into Atlas

```
# add new type definition (entity)
curl -u admin:kdHVNXuo1zMjnM32QqAk -X POST -H 'Content-Type:
application/json' -H 'Accept: application/json'
"http://localhost:21000/api/atlas/v2/types/typedefs" -d
"@./typeDef_datafile.json"

# check the newly added type
curl -u admin:kdHVNXuo1zMjnM32QqAk
http://localhost:21000/api/atlas/v2/types/typedef/name/GPFSDataFile
```

5.1.2 Add a new metadata entity

Now we can add a new metadata entity of type GPFSDataFile

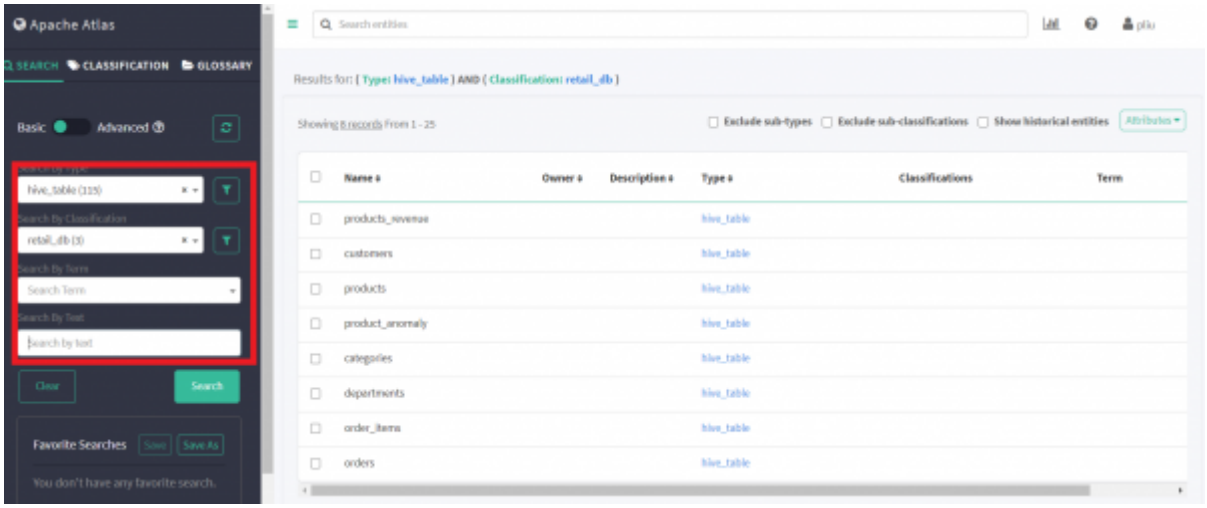
```
{
  "entities": [
    {
      "typeName": "GPFSDataFile",
      "createdBy": "pliu",
      "attributes": {
        "qualifiedName": "it's only a simple attribute",
        "uri": "pengfei.org",
        "name": "human_blood_sample.csv",
        "file_name": "human_blood_sample.csv",
        "parent_dir": "/sps/bioater/pt6/pliu/mosaic/data/",
        "user":"pliu",
        "group":"bioaster",
        "permission":"650",
        "creation_time":"2019-08-18T18:49:44.000Z",
        "format":"csv",
        "description":"test rest api",
        "owner":"pliu"
      },
      "classifications": [
        { "typeName": "Mosaic" }
      ]
    },
{
      "typeName": "GPFS_Path",
      "createdBy": "pliu",
      "attributes": {
        "qualifiedName": "rabbit tissu image",
        "uri": "pengfei.org",
        "name": "rabbit_124_tissu_infection.jpg",
        "file_name": "rabbit_124_tissu_infection.jpg",
        "parent_dir": "/sps/bioater/pt6/pliu/mosaic/data/rabbit",
        "user":"pliu",
        "group":"bioaster",
        "permission":"650",
        "creation_time":"2019-08-19T18:49:44.000Z",
        "format":"jpg",
        "description":"test rest api",
        "owner":"pliu"
      },
      "classifications": [
        { "typeName": "Mosaic" }
      ]
    }
```

```
]
}
```

```
# The Below command load the above json file into the atlas
curl -u admin:kdHVNXuo1zMjnM32QqAk -X POST -H 'Content-Type:
application/json' -H 'Accept: application/json'
"http://localhost:21000/api/atlas/v2/entity/bulk" -d "@./load_data.json"
```

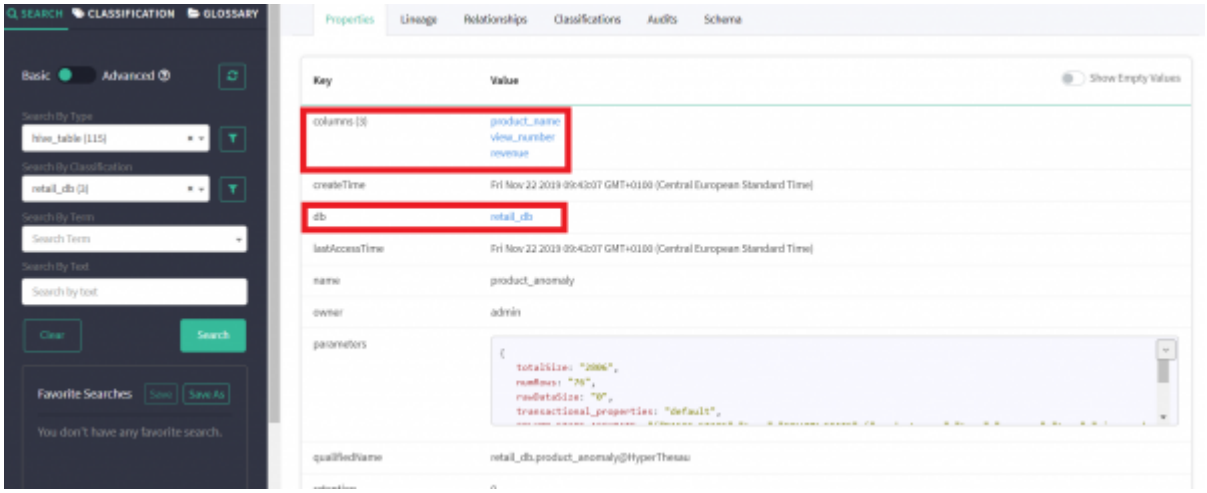
5.2 Search data by using metadata

Atlas allows us to search for data via different properties. For example, we can search data by using its attributes, classifications, etc..



5.3 metadata properities and lineage

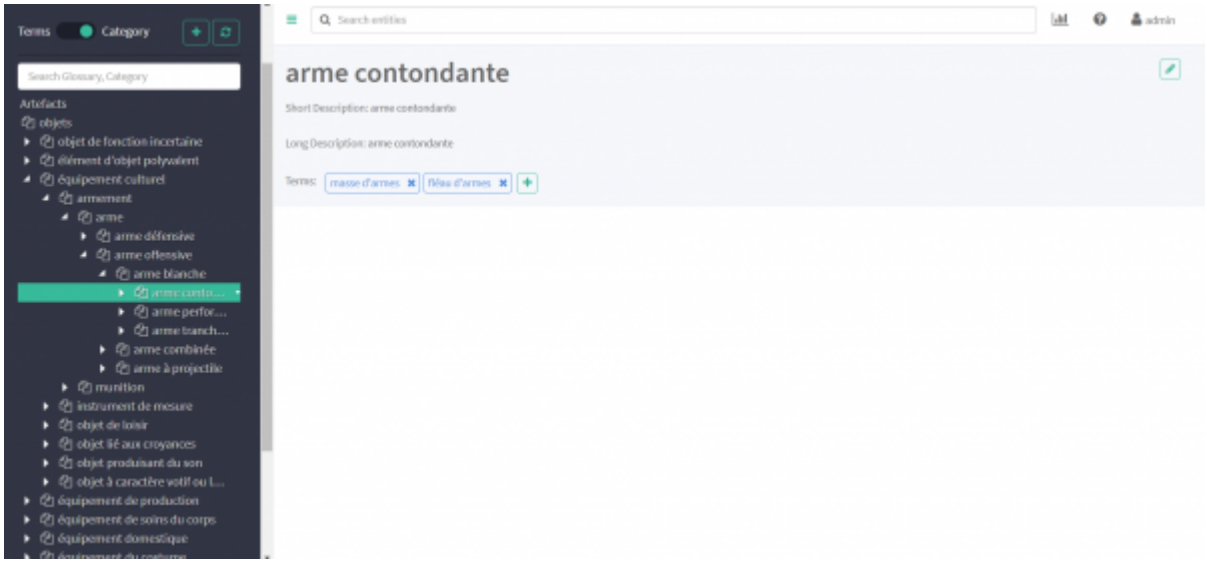
Once we find the metadata entity, we can view the list of the properties which describe the actual data. For example, the below image shows a table in a database. With the help of these properties, we know that this table is located at the database “retail_db” and has three columns.



The data lineage shows where these data came from. For example, the below image shows that this table is built by using three tables and one file.

5.4 Thesaurus

Atlas also allows us to use a thesaurus to index metadata. The below image shows a thesaurus imported from the artefact.



Load a theasurus into Atlas

For example, we can load a simple term by using the following command.

```
{
"anchor":{
"displayText" : "Artefacts",
"glossaryGuid":"ee3f05ba-2fe1-4806-a81e-660c777a3403",
"relationGuid":"3be4d5d8-9a98-421c-a3f4-f8279604e50c"},
"longDescription":"Pièce ornementale ou de renfort fixée sur une autre
(fr)",
"name":"applique",
```

```
"qualifiedName":"applique@Artefacts",
"shortDescription":"applique"
}
```

```
curl -u admin:pwd -X POST -H 'Content-Type: application/json' -H 'Accept:
application/json' "http://localhost:21000/api/atlas/v2/glossary/term" -d
"@./simple_term.json"
```

The full version of the json template for loading term can be found here http://atlas.apache.org/api/v2/resource_GlossaryREST.html#resource_GlossaryREST_createGlossaryTerm_POST.

The full_HyperThesaurus.json is the full template for hyperThesaurus project. He only needs “see also” and “synonyms” as term relation.

full_HyperThesaurus.json

```
{
  "abbreviation" : "...",
  "anchor" : {
    "displayText" : "...",
    "glossaryGuid" : "...",
    "relationGuid" : "..."
  },
  "assignedEntities" : [ {
    "displayText" : "...",
    "entityStatus" : "ACTIVE",
    "relationshipAttributes" : {
      "attributes" : {
        "property1" : { },
        "property2" : { }
      },
      "typeName" : "..."
    },
    "relationshipGuid" : "...",
    "relationshipStatus" : "DELETED",
    "relationshipType" : "...",
    "guid" : "...",
    "typeName" : "...",
    "uniqueAttributes" : {
      "property1" : { },
      "property2" : { }
    }
  }, {
    "displayText" : "...",
    "entityStatus" : "DELETED",
    "relationshipAttributes" : {
      "attributes" : {
        "property1" : { },
        "property2" : { }
      },
```

```
      "typeName" : "..."
    },
    "relationshipGuid" : "...",
    "relationshipStatus" : "DELETED",
    "relationshipType" : "...",
    "guid" : "...",
    "typeName" : "...",
    "uniqueAttributes" : {
      "property1" : { },
      "property2" : { }
    }
  } ],
  "categories" : [ {
    "categoryGuid" : "...",
    "description" : "...",
    "displayText" : "...",
    "relationGuid" : "...",
    "status" : "DEPRECATED"
  }, {
    "categoryGuid" : "...",
    "description" : "...",
    "displayText" : "...",
    "relationGuid" : "...",
    "status" : "DEPRECATED"
  } ],

  "seeAlso" : [ {
    "description" : "...",
    "displayText" : "...",
    "expression" : "...",
    "relationGuid" : "...",
    "source" : "...",
    "status" : "ACTIVE",
    "steward" : "...",
    "termGuid" : "..."
  }, {
    "description" : "...",
    "displayText" : "...",
    "expression" : "...",
    "relationGuid" : "...",
    "source" : "...",
    "status" : "DRAFT",
    "steward" : "...",
    "termGuid" : "..."
  } ],
  "synonyms" : [ {
    "description" : "...",
    "displayText" : "...",
    "expression" : "...",
    "relationGuid" : "...",
```

```
    "source" : "...",
    "status" : "OTHER",
    "steward" : "...",
    "termGuid" : "..."
  }, {
    "description" : "...",
    "displayText" : "...",
    "expression" : "...",
    "relationGuid" : "...",
    "source" : "...",
    "status" : "ACTIVE",
    "steward" : "...",
    "termGuid" : "..."
  } ],



  "classifications" : [ {
    "entityGuid" : "...",
    "entityStatus" : "DELETED",
    "propagate" : true,
    "removePropagationsOnEntityDelete" : true,
    "validityPeriods" : [ {
      "endTime" : "...",
      "startTime" : "...",
      "timeZone" : "..."
    }, {
      "endTime" : "...",
      "startTime" : "...",
      "timeZone" : "..."
    } ],
    "attributes" : {
      "property1" : { },
      "property2" : { }
    },
    "typeName" : "..."
  }, {
    "entityGuid" : "...",
    "entityStatus" : "ACTIVE",
    "propagate" : true,
    "removePropagationsOnEntityDelete" : true,
    "validityPeriods" : [ {
      "endTime" : "...",
      "startTime" : "...",
      "timeZone" : "..."
    }, {
      "endTime" : "...",
      "startTime" : "...",
      "timeZone" : "..."
    } ],
    "attributes" : {
```

```
      "property1" : { },
      "property2" : { }
    },
    "typeName" : "..."
  } ],
  "longDescription" : "...",
  "name" : "...",
  "qualifiedName" : "...",
  "shortDescription" : "...",
  "guid" : "..."
}
```