AFOUR TECHNOLOGIES

WHITEPAPER

Things to know for building a Data Governance Model

Some why’s and how’s to know to start modeling for data governance.

Shailesh Gaikwad
Email : shailesh.gaikwad@afourtech.com

Why do you need Data Governance?

In era of digital world, a huge chunk of data is generated and processed across wide varieties of sources from On-Premises, Clouds like Amazon or Azure Big data services and with that it has added multiple challenges in front of Data Orchestrators, Data Stewards and End Consumers.

Challenges ranges from

- **Searching time** to discover target data for analysis purpose
- Multiple copies of data residing at multiple location create **ambiguity** in usage of data
- Different types of data locations like On Prem RDBMS Servers or Cloud based locations limit data discovery & in turn resulting in duplication of data sets
- No way to track the usage for BI and Analytics
- Reverse track the sources of target datasets
- **Limitations in identifying data schema**
- Mechanism to track ETL jobs producing and **processing** the data
- Limitations in compliance with GDPR policies as there is no easy way to differentiate between PII and Non-PII data
- Lack of classifiers on data adds up to limited functionality to build **secured model** of accessing the data

Real Time – Big Data Platform Use Case

A Product company tracks the subscriptions and usage of its enterprise application through client telemetry & server telemetry. And then uses the generated data to deep dive into usage patterns to penetrate more.

To address these primary requirement multiple teams comes into picture –

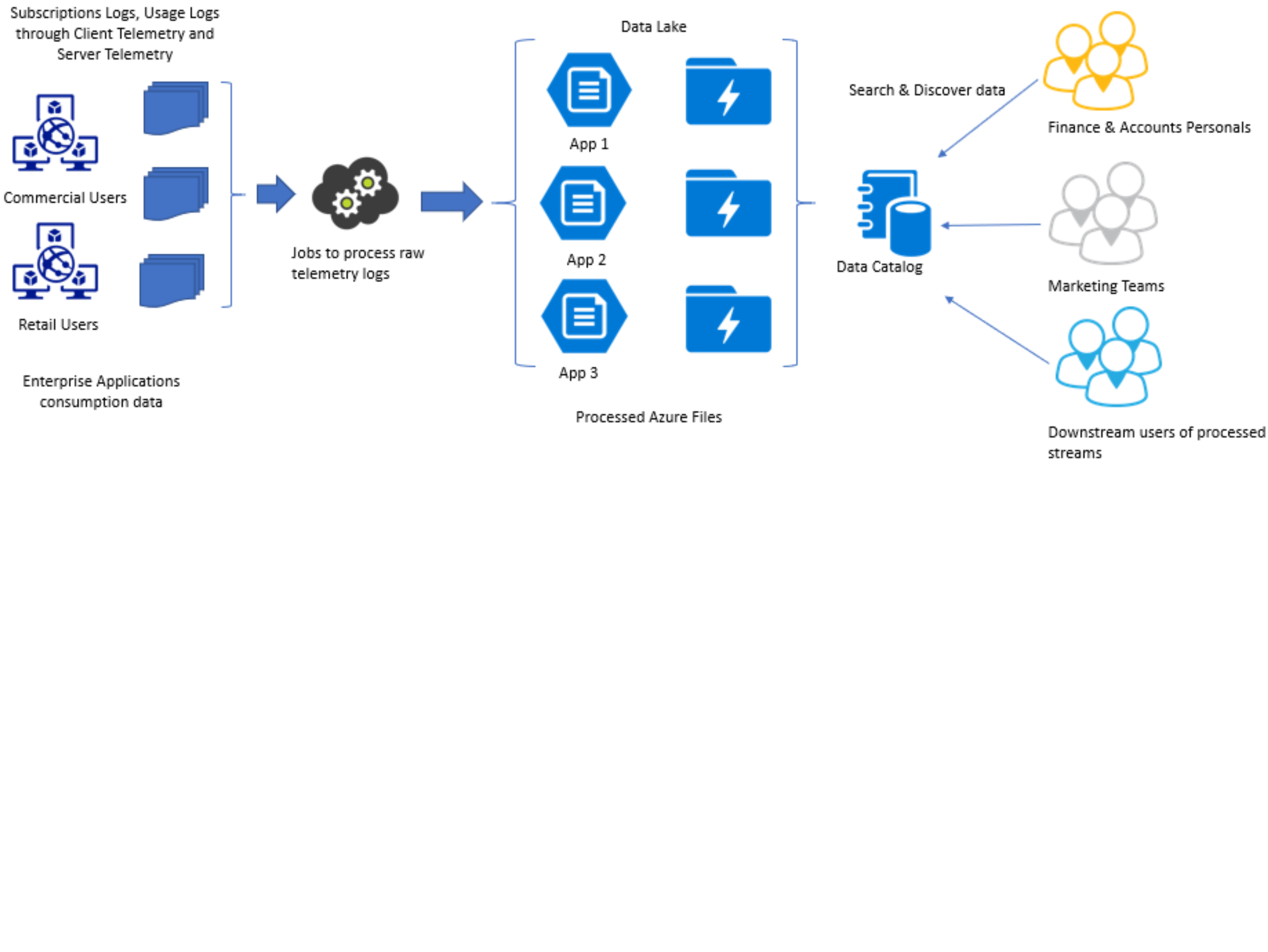
- Data engineering team to segregate the huge volumes of daily generated data into multiple sets of processed logs per application for their downstream users.

- End consumers like the Marketing team, use the data for targeted campaigns.

- Product engineering team builds solution

  - To comply with GDPR requirements or
  - Add appropriate layers to add security layer to access PII or sensitive data.
  - Add new features or enhancements to application

And for all these use cases a Data Governance team comes into the picture which helps in reducing the efforts required to easily discover and consume data by building a Data Catalog using tools like Apache Atlas or Azure Data Catalog.



Overall summary and deliverables expected from Data Governance

Business Problem

Data Discovery

Secure Data

Identify source of Data

Deliverable

Meta data management framework for Data Governance

End users to discover the data

End users to raise access requests

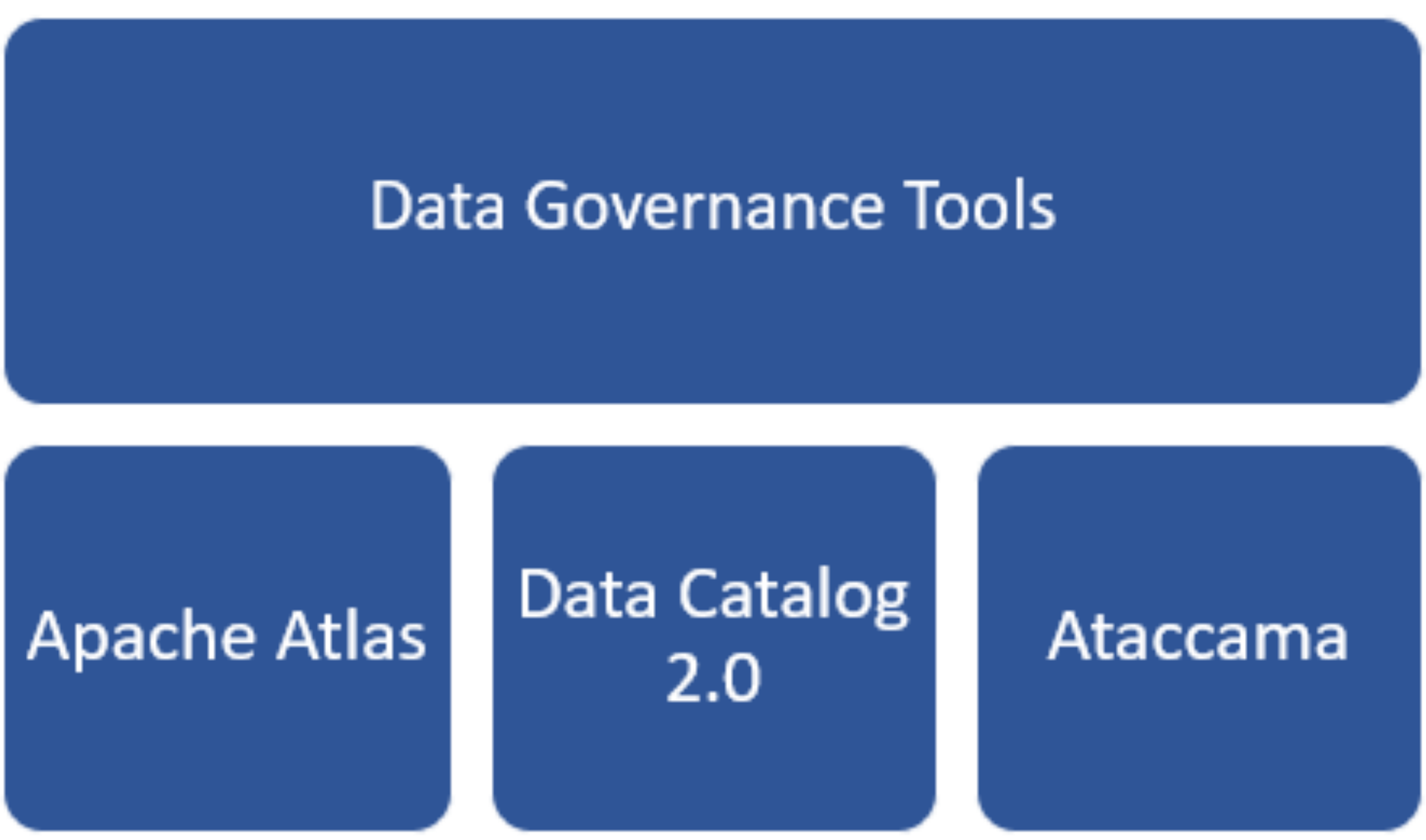
Identify owners of Data

Lineage & Relationships between data sets

AFour Technologies over the period acquired the expertise to automate Data Governance needs using tools like Apache Atlas and Azure Data Catalog.

What are the tools at AFour explored and implemented?

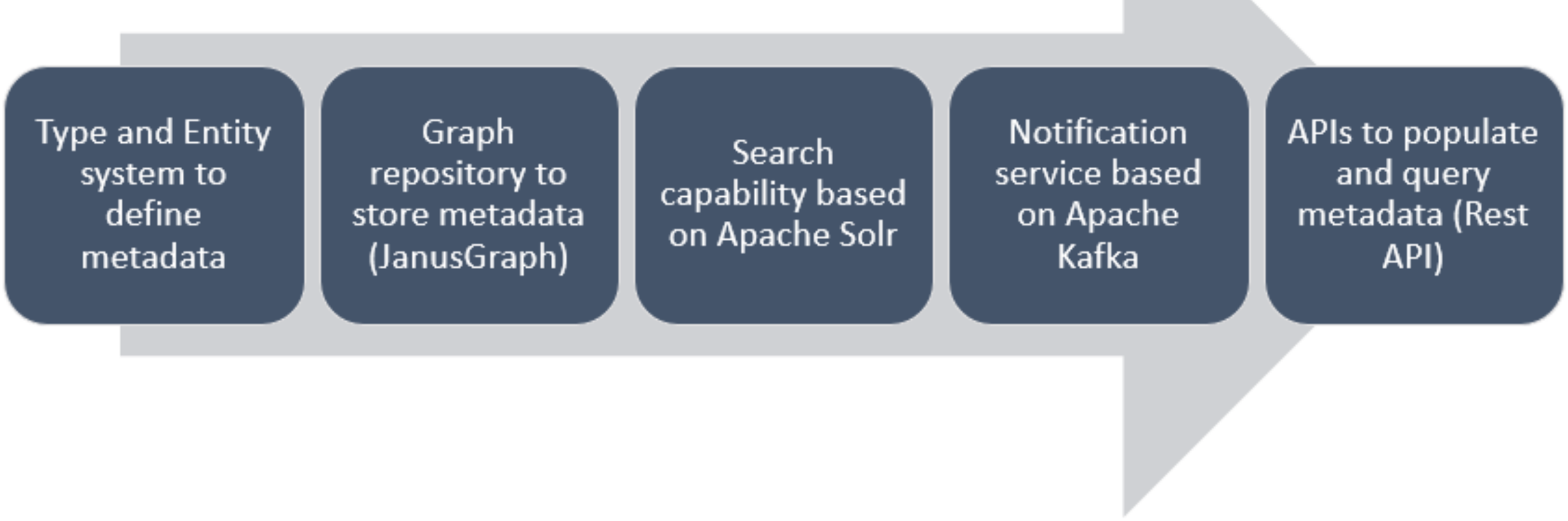
Azure Data Catalog with inbuilt support to extract metadata from Azure Big Data Services helps organizations who use Azure Ecosystem as their storage backends. Apache Atlas with inbuilt support for Hadoop based systems like Hive, helps to extract the metadata from RDBMS like Oracle via SQOOP.



Why Apache Atlas and Data Catalog 2.0

Data Catalog 2.0 is built on top of Apache Atlas, few basic terminologies and principals if learnt then one can work with either tool based on requirements.

What are the main components in Apache Atlas?

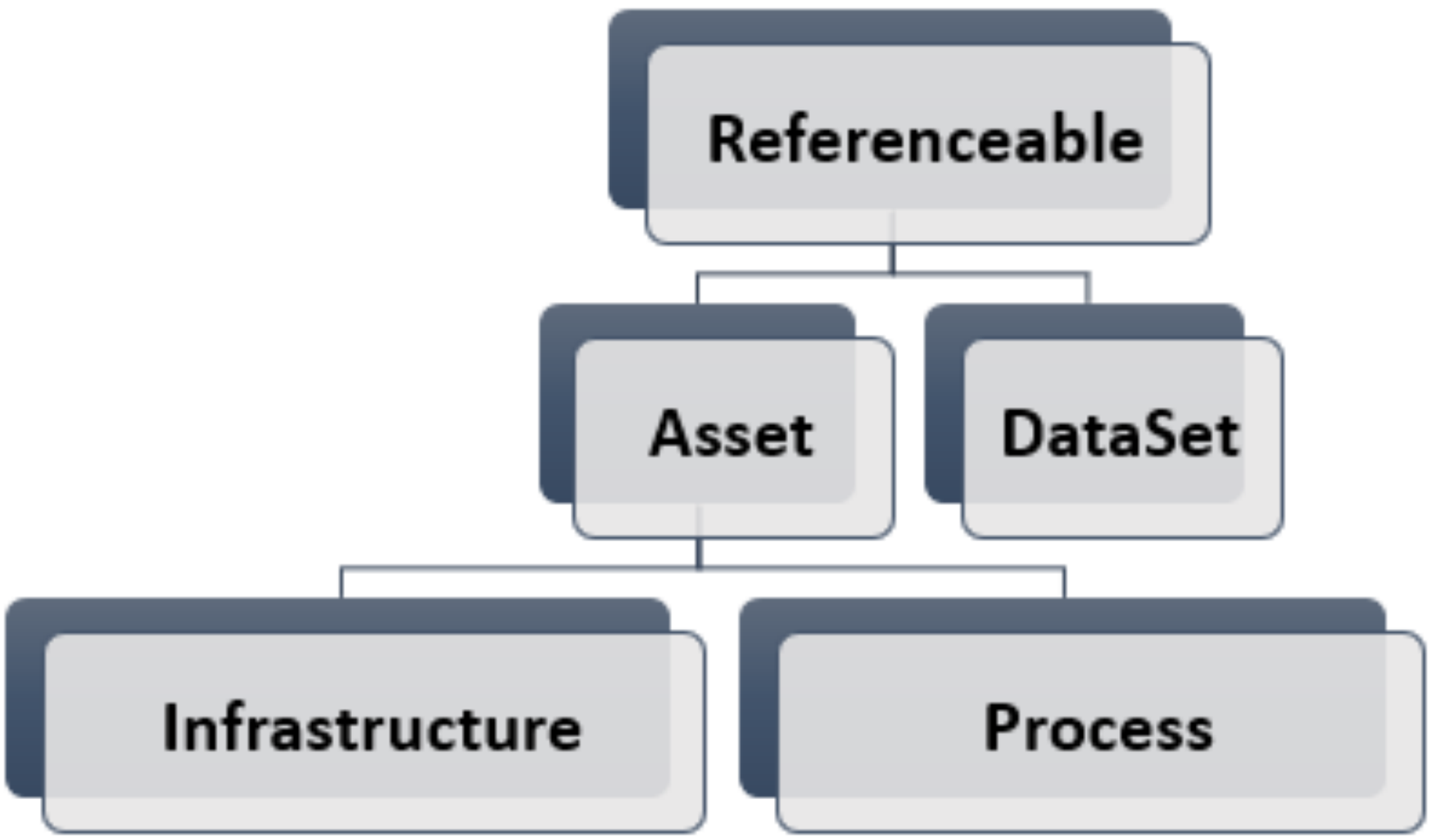


What are Types and Entities?

Atlas allows users to define a model for the metadata objects they want to manage.

- The model is composed of definitions called ‘types’.
- Instances of ‘types’ are called ‘entities’ which represent the actual metadata objects that are managed.
- The ‘Type System’ is a component that allows users to define and manage the types and entities.
- All metadata objects managed by Atlas out of the box (e.g. Hive tables) are modelled using types and represented as entities.
- To store new types of metadata in Atlas, one needs to understand the concepts of

In order to define Types and Entities, a mandatory step in Apache Atlas is to understand the Types and Entities.



Referenceable: This type represents all entities that can be searched for using a unique attribute called **qualifiedName**.

Asset: This type extends Referenceable and adds attributes like name, description and owner. **Name** is a **required attribute** (isOptional=false), the others are optional.

The purpose of Referenceable and Asset is to provide modellers with a way to enforce consistency when defining and querying entities of their own types. Having these fixed sets of attributes allows applications and user interfaces to make convention-based assumptions about what attributes they can expect of types by default.

Infrastructure: This type extends Asset and typically can be used to be a common supertype for infrastructural metadata objects like **clusters, hosts etc**.

DataSet: This type extends Referenceable. Conceptually, it can be used to **represent a type that stores data**.

In Atlas, hive tables, hbase_tables etc are all types that extend from DataSet.

Types that extend DataSet can be expected to have a Schema in the sense that they would have an attribute that defines attributes of that dataset. For e.g. the columns attribute in a hive_table. Also entities of types that extend DataSet participate in data transformation and this transformation can be captured by Atlas via lineage (or provenance) graphs.

Process: This type extends Asset. Conceptually, it can be used to **represent any data transformation operation**.

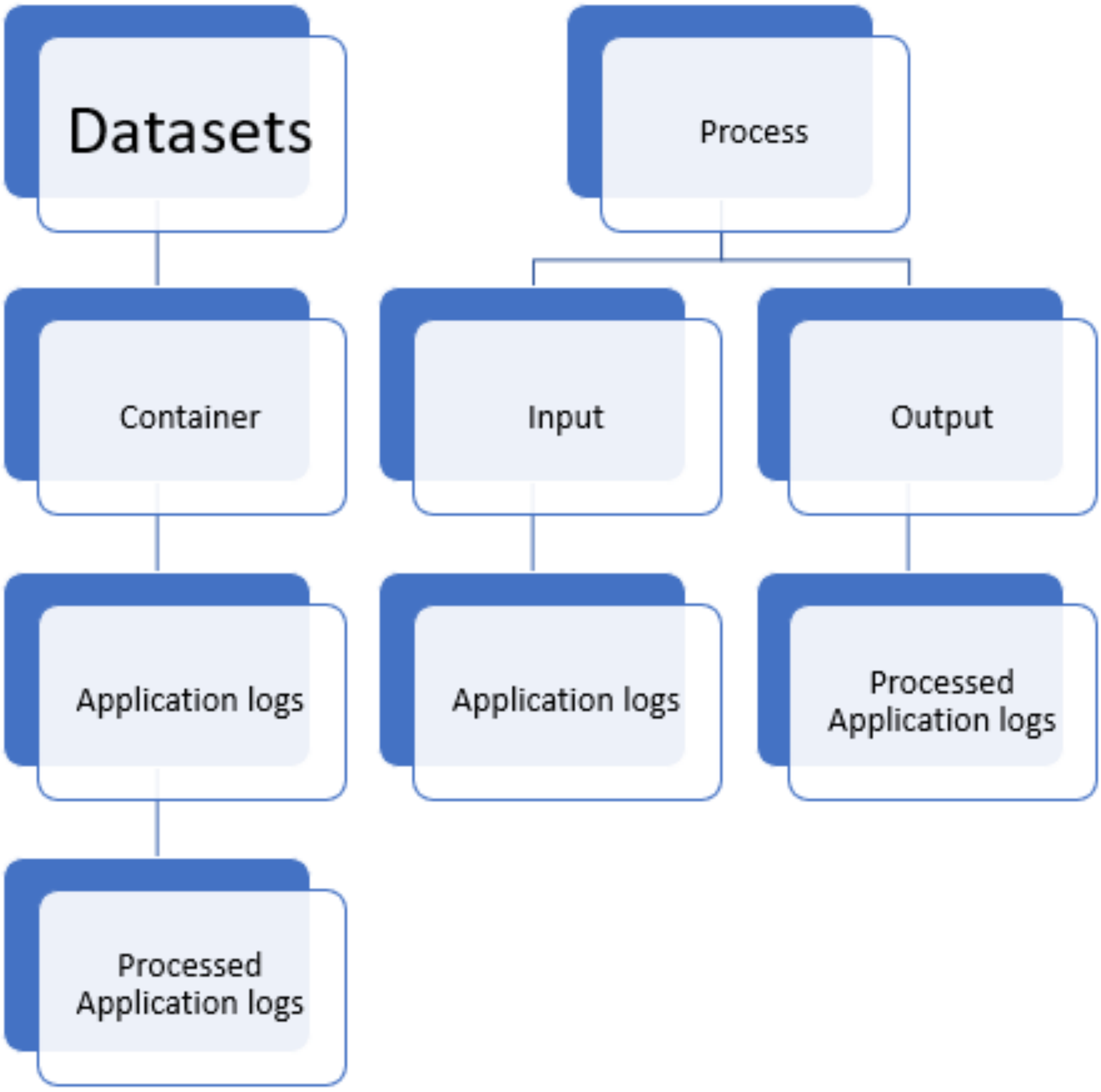
For example, an ETL process that transforms a hive table with raw data to another hive table that stores some aggregate can be a specific type that extends the Process type.

A Process type has two specific attributes, inputs and outputs. Both inputs and outputs are arrays of DataSet entities. Thus, an instance of a Process type can use these inputs and outputs to capture how the lineage of a DataSet evolves.

In Above example there are multiple entities

- Raw Logs generated across the globe from consumption of applications
- Jobs to process the Raw Logs
- Processed outputs
- Data lake paths
- Data Catalog – to store metadata from every file processed and generated.
- Classification – to classify logs into multiple categories like – Applications, AssetsType (Laptop, Mobile, PC), OS type (Windows, Mac), User Role in Organization (Engineer, Finance, Business or Leadership team)
- Relationships – Processed file relation with Data Lake path, source streams
- Lineage – Processed file relation source streams via Processes

Model in Atlas to uniquely identify all these entities



RestAPIs to import metadata into Apache Atlas

Apache Atlas has exposed set of RestAPIs to create Types and entities into Apache Atlas, user guide & Swagger details can be referred here : https://atlas.apache.org/api/v2/ui/index.html
In order to create Models for our problem statement, below are the definitions for Types & Entities.

Container Type – a type definition for grouping logs


```
{
  "entityDefs": [
    {
      "superTypes": [
        "DataSet"
      ],
      "category": "ENTITY",
      "name": "Container",
      "description": "a type definition for grouping logs",
      "typeVersion": "1.0",
      "attributeDefs": [
      ],
       "subTypes": [
                "applicationrawlogs"
            ]
    }
  ]
}
```

applicationrawlogs


```
{
  "entityDefs": [
    {
        "category": "ENTITY",
        "name": "applicationrawlogs",
        "description": "Applicationrawlogs are the raw logs generated by devices and
        from server side for consumptions & usages of applications.",
      "typeVersion": "2.0",
      "attributeDefs": [
          {
              "name": "filename",
              "typeName": "string",
              "isOptional": false,
              "cardinality": "SINGLE",
              "valuesMinCount": 1,
              "valuesMaxCount": 1,
              "isUnique": false,
              "isIndexable": true,
              "includeInNotification": false
          },
          {
              "name": "storagepath",
              "typeName": "string",
              "isOptional": false,
              "cardinality": "SINGLE",
              "valuesMinCount": 1,
              "valuesMaxCount": 1,
              "isUnique": false,
              "isIndexable": true,
              "includeInNotification": false
          }
      ],
      "superTypes": [
          "Container"
      ],
      "subTypes": [
          "applicationprocessedlogs"
      ]
    }
  ]
}
```

applicationprocessedlogs


```
{
  "entityDefs": [
    {
        "category": "ENTITY",
        "name": "applicationprocessedlogs",
        "description": "applicationprocessedlogs are the processed logs
        generated by ETL jobs",
        "typeVersion": "2.0",
        "options": {
            "schemaElementsAttribute": "columns"
        }
        "attributeDefs": [
            {
                "name": "filename",
                "typeName": "string",
                "isOptional": false,
                "cardinality": "SINGLE",
                "valuesMinCount": 1,
                "valuesMaxCount": 1,
                "isUnique": false,
                "isIndexable": true,
                "includeInNotification": false
            },
            {
                "name": "storagepath",
                "typeName": "string",
                "isOptional": false,
                "cardinality": "SINGLE",
                "valuesMinCount": 1,
                "valuesMaxCount": 1,
                "isUnique": false,
                "isIndexable": true,
                "includeInNotification": false
            }
        ],
        "superTypes": [
            "applicationrawlogs"
        ],
        "subTypes": [
            "applicationprocessedlogscolumns"
        ]
    }
    ]
}
```

applicationprocessedlogscolumns


```
{
    "name": "applicationprocessedlogscolumns",
    "superTypes": [
        "applicationprocessedlogs"
    ],
    "typeVersion": "2.0",
    "attributeDefs": [
        {
            "name": "type",
            "typeName": "string",
            "cardinality": "SINGLE",
            "isIndexable": true,
            "isOptional": false,
            "isUnique": false
        },
        {
            "name": "description",
            "typeName": "string",
            "cardinality": "SINGLE",
            "isIndexable": false,
            "isOptional": true,
            "isUnique": false
        },
        {
            "name": "shortName",
            "typeName": "string",
            "cardinality": "SINGLE",
            "isIndexable": true,
            "isOptional": true,
            "isUnique": false
        }
    ]
}
```

What are Classifications?

Classifications help in to organize data, locate and retrieve data.
Some of the Classifications fall in below categories

- PII,
- SENSITIVE,
- EXPIRES_ON,
- DATA_QUALITY

Note : Entities can be associated with multiple classifications, enabling easier discovery and security enforcement

How to organize data into categories that make it easy to retrieve, sort and store for future use.

Know importance of Tags and their Propagation

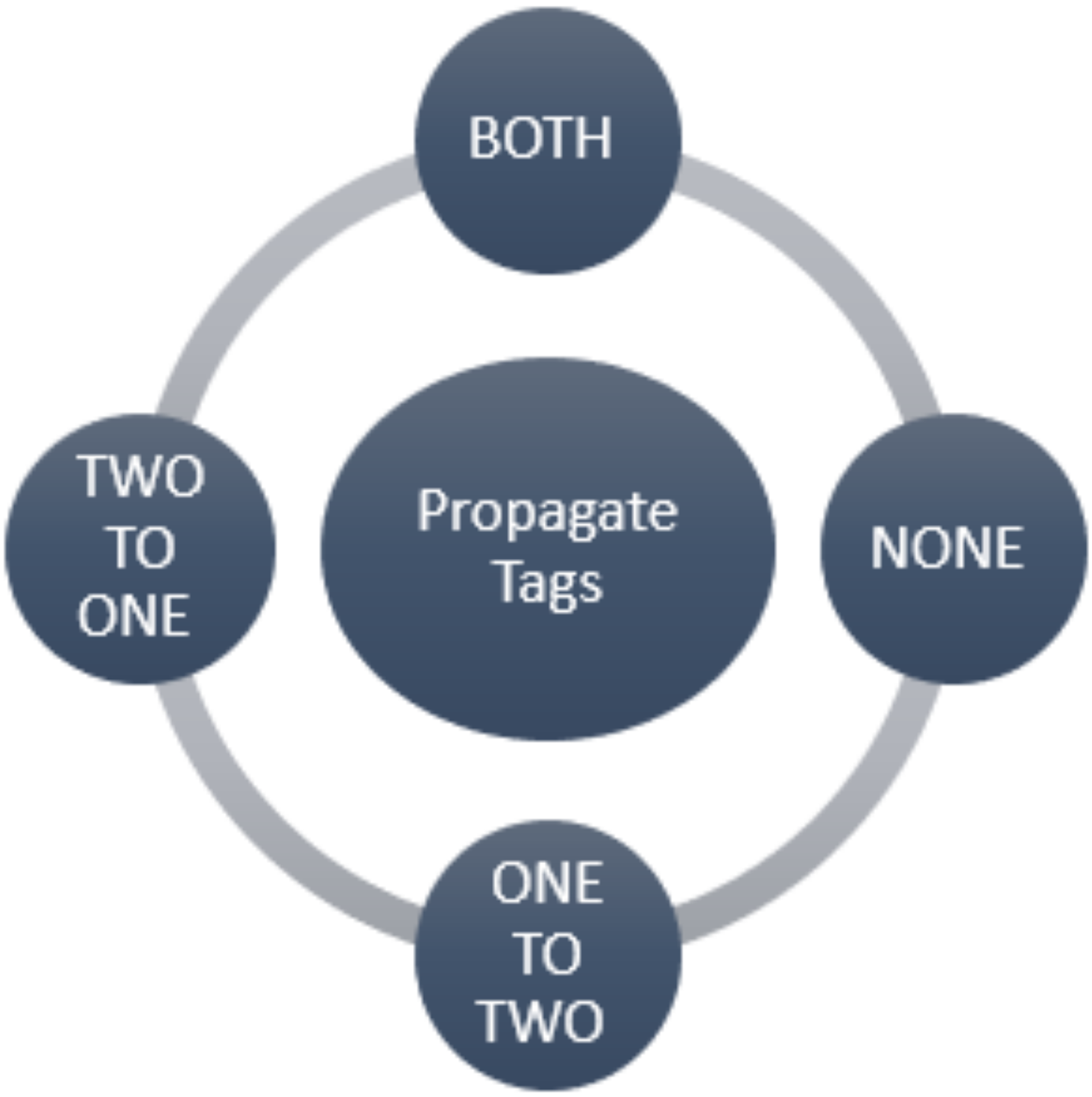
Data classification involves tagging data to make it easily searchable and trackable.
It also eliminates multiple duplications of data, which can reduce storage and backup costs while speeding up the search process.

Classification Propagation : Classification propagation enables classifications associated to an entity to be automatically associated with other related entities of the entity.

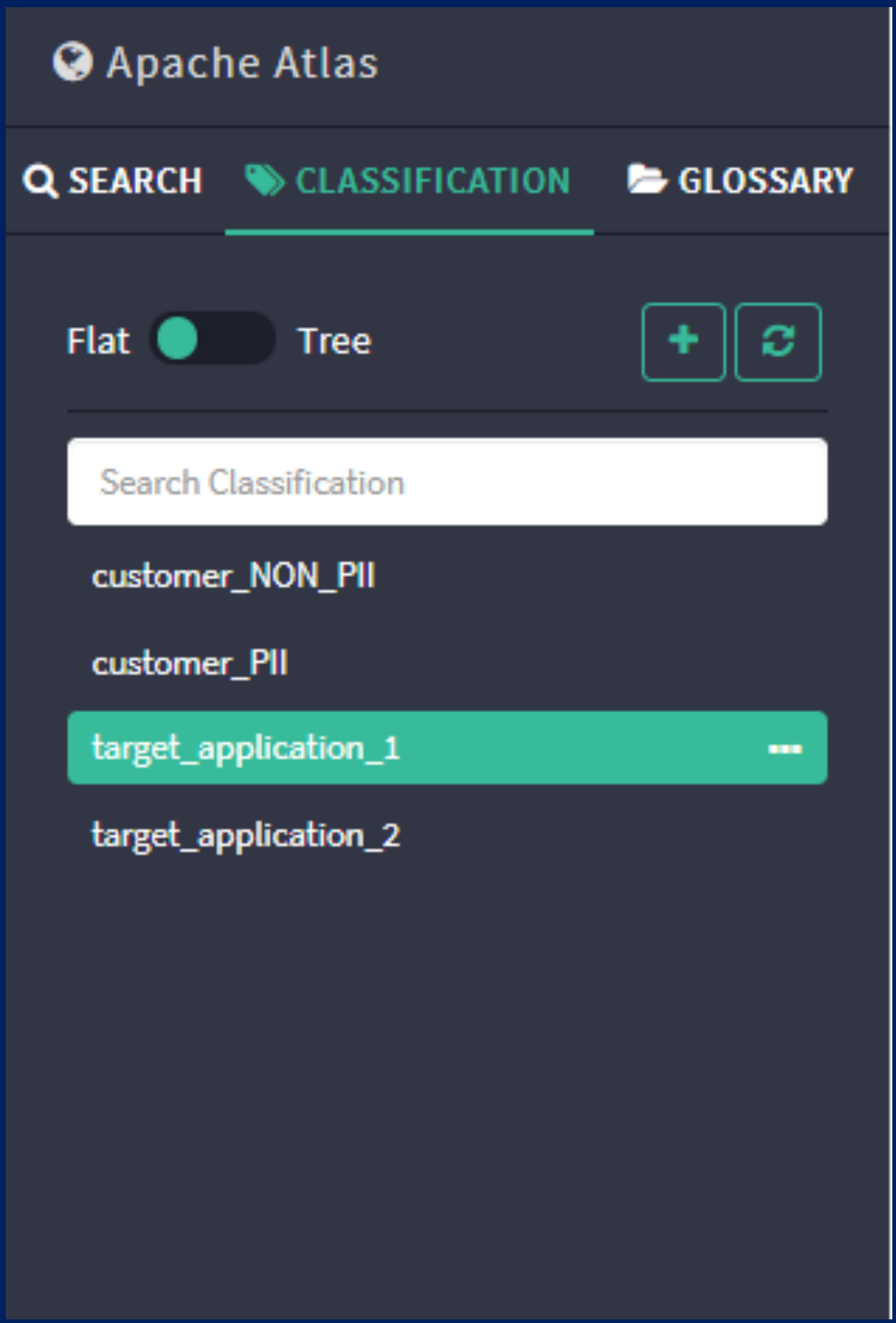
Propagate Tags : PropagateTags indicates whether tags should propagate across the relationship instance.

Care needs to be taken when specifying. The use cases we are aware of where this flag is useful:

- propagating confidentiality classifications from a table to columns - ONE_TO_TWO could be used here
- propagating classifications around Glossary synonyms - BOTH could be used here.



```
{
 "classificationDefs": [
 {
     "category": "CLASSIFICATION",
     "name": "target_application_1",
     "description": "Used for classifying a application type feed",
     "typeVersion": "1.0",
     "attributeDefs": [],
     "superTypes": []
 },
 {
     "category": "CLASSIFICATION",
     "name": "target_application_2",
     "description": "Used for classifying a application type feed",
     "typeVersion": "1.0",
     "attributeDefs": [],
     "superTypes": []
},
{
  "category": "CLASSIFICATION",
  "name": "customer_PII",
  "description": "Used for classifying a data which contains customer personal
  information, hence indicating confidential private",
  "typeVersion": "1.0",
  "attributeDefs": [],
  "superTypes": []
},
  {
    "category": "CLASSIFICATION",
    "name": "customer_NON_PII",
    "description": "Used for classifying a data which contains customer non personal
    information, hence indicating confidential lesser than private",
    "typeVersion": "1.0",
    "attributeDefs": [],
    "superTypes": []
  }
],
```


Apache Atlas
SEARCH
CLASSIFICATION
GLOSSARY
Flat
Tree
Search Classification
customer_NON_PII
customer_PII
target_application_1
target_application_2

Apache Atlas

SEARCH | CLASSIFICATION | GLOSSARY

Basic Advanced

Search By Type

Select Type

Search By Classification

Select Classification

Search By Term

Search Term

Search By Text

Search by text

Clear | Search

Favorite Searches | Save | Save As

You don't have any favorite search.

Back To Results | admin

enterpriseapplicationsrawlogs_03122020.tsv (applicationrawlogs)

Classifications: customer_NON_PII

Term:

Properties | Lineage | Relationships | Classifications | Audits

Key	Value
filename	enterpriseapplicationsrawlogs_03122020.tsv
name	enterpriseapplicationsrawlogs_03122020.tsv
qualifiedName	enterpriseapplicationsrawlogs_03122020.tsv
storagepath	adl:enterpriseapplicationslogs
telemetryType	clienttelemetry

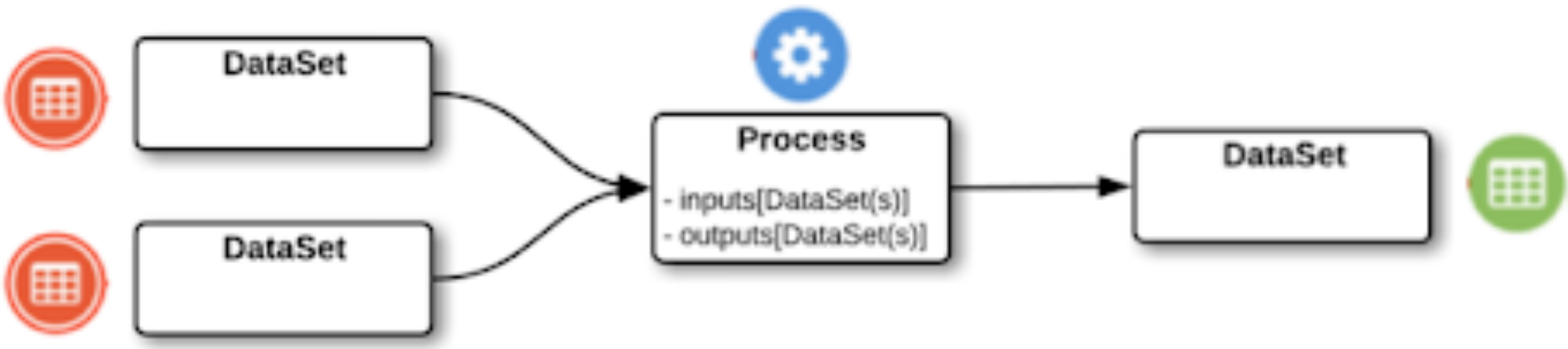
Show Empty Values

What is Lineage in Apache Atlas

Data Lineage states where data is coming from, where it is going, and what transformations are applied to it as it flows through multiple processes

Lineage

- Intuitive UI to view lineage of data as it moves through various processes
- REST APIs to access and update lineage



Type definition for ETL Job


```
{
 "entityDefs":[
    {
        "superTypes":[
            "Process"
        ],
        "category":"ENTITY",
        "name":"etl_azureanalytics_job",
        "description":"a type definition for ETL job",
        "typeVersion":"1.0",
        "attributeDefs":[
            {
                "name":"etl_job_name",
                "typeName":"string",
                "isOptional":true,
                "cardinality":"SINGLE",
                "valuesMinCount":1,
                "valuesMaxCount":1,
                "isUnique":false,
                "isIndexable":false
            },
            {
                "name":"etl_job_owner",
                "typeName":"string",
                "isOptional":false,
                "cardinality":"SINGLE",
                "valuesMinCount":1,
                "valuesMaxCount":1,
                "isUnique":true,
                "isIndexable":true
            },
            {
                "name":"run_date",
                "typeName":"string",
                "isOptional":true,
                "cardinality":"SINGLE",
                "valuesMinCount":1,
                "valuesMaxCount":1,
                "isUnique":false,
                "isIndexable":false
            }
        ]
    }
]
```

Type for Relationship Definitions:


```
],
"relationshipDefs": [
    {
        "name": "applicationprocessedlogsvscolumn",
        "typeVersion": "1.2",
        "relationshipCategory": "COMPOSITION",
        "relationshipLabel": "_application_processed_logs_.columns",
        "endDef1": {
            "type": "applicationprocessedlogs",
            "name": "columns",
            "isContainer": true,
            "cardinality": "SET",
            "isLegacyAttribute": true
        },
        "endDef2": {
            "type": "applicationprocessedlogscolumns",
            "name": "application_processed_logs",
            "isContainer": false,
            "cardinality": "SINGLE",
            "isLegacyAttribute": true
        },
        "propagateTags": "NONE"
    }
]
```

Entities & Relationships


```
{
    "entities": [
        {
            "typeName": "applicationprocessedlogscolumns",
            "attributes": {
                "name": "userid",
                "qualifiedName": "userid",
                "position": 1,
                "type": "string",
                "shortName": "UId",
                 "application_processed_logs": {
                    "guid" : "-100",
                    "typeName": "applicationprocessedlogs"
                }
            }
        },
        {
            "typeName": "applicationprocessedlogscolumns",
            "attributes": {
                "name": "featureused",
                "qualifiedName": "featureused",

                "position": 2,
                "type": "string",
                "shortName": "featureused",
                 "application_processed_logs": {
                    "guid" : "-100",
                    "typeName": "applicationprocessedlogs"
                }
            }
        }
    ],
    "referredEntities": {
    "-100": {
      "guid": "-100",
      "typeName": "applicationprocessedlogs",
      "attributes": {
                "name": "application_log_targetapplication_1_20200313.tsv",
                "description": "target_application_1 log",
                "qualifiedName": "application_log_targetapplication_1_20200313",
      }
    }
  }
}
```

application_log_targetapplication_1_20200313.tsv (applicationprocessedlogs)

Classifications: target_application_1 ✖ +

Term: +

Propagated Classifications: customer_NON_PII

Properties | Lineage | Relationships | Classifications | Audits | Schema

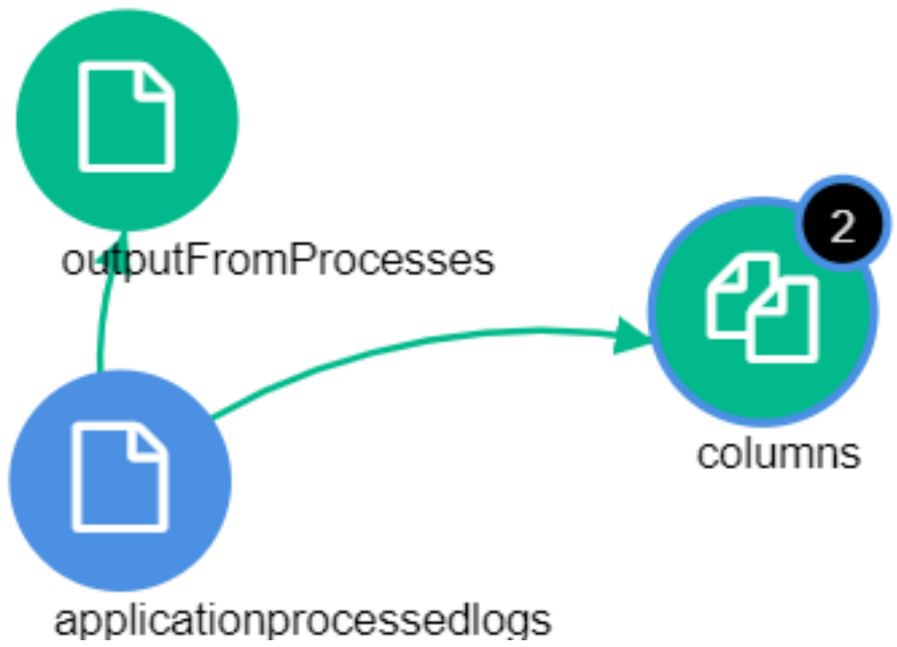
Graph Table

→Active →Deleted

columns ✖

Search Entities

1. featureused (applicationprocessedlogscolumns)
2. userid (applicationprocessedlogscolumns)



Setting up Lineage


```
{
    "entities":[
        {
            "typeName":"etl_azureanalytics_job",
            "createdBy":"etl team",
            "attributes":{
                "qualifiedName":"etl_azureanalytics_job_001",
                "name":"raw logs etl job",
                "etl_job_owner":"engineeringteam",
                "run_date":"2020-03-13",
                "inputs":[
                    {
                        "qualifiedName":"application_raw_log_allapplications_20200313",
                        "typeName":"applicationrawlogs"
                    }
                ],
                "outputs":[
                    {
                        "qualifiedName" : "application_log_targetapplication_1_20200313"
                        "typeName":"applicationprocessedlogs"
                    },
                    {
                        "qualifiedName" : "application_log_targetapplication_2_20200313"
                        "typeName":"applicationprocessedlogs"
                    }
                ]
            }
        }
    ]
}
```

Back To Results

logprocessingjob (etl_azureanalytics_job)

Classifications:

Term:

Propagated Classifications: customer_NON_PII

Properties | Lineage | Relationships | Classifications | Audits

Current Entity → Lineage → Impact

logprocessingjob (etl_azureanalytics_job)

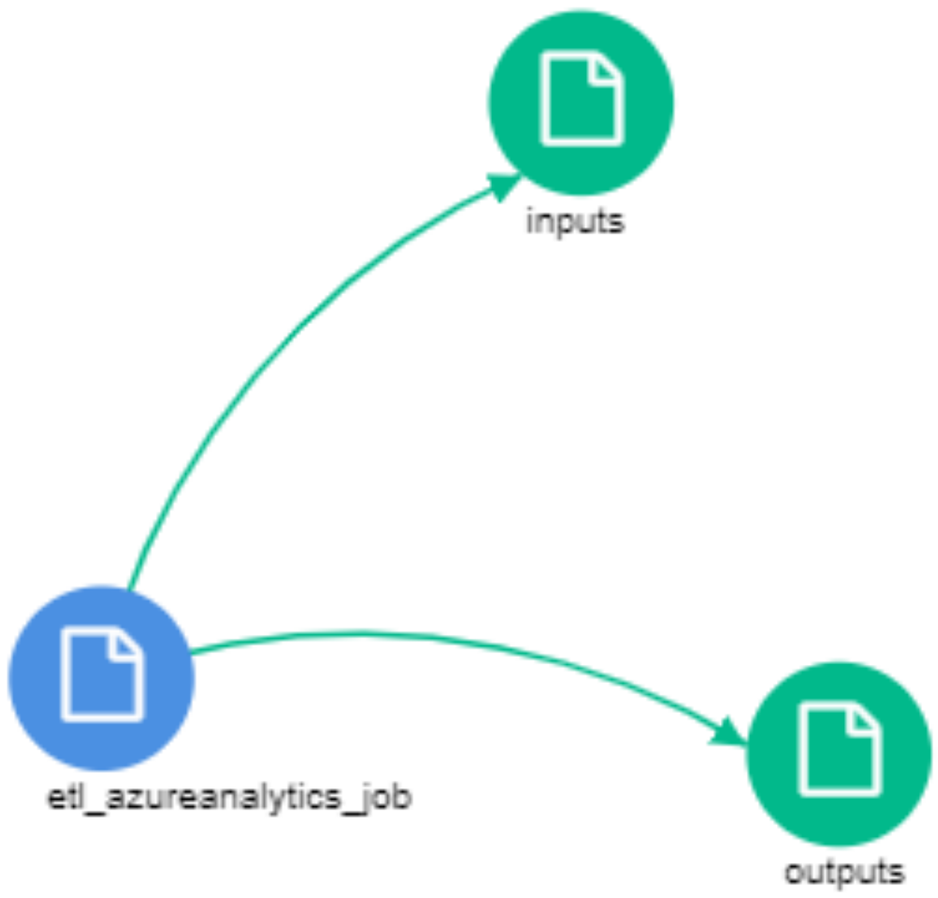
Classifications:

Term:

Propagated Classifications: customer_NON_PII

Properties | Lineage | Relationships | Classifications | Audits

Graph Table



How the relationships are stored in Apache Atlas – JanusGraph?

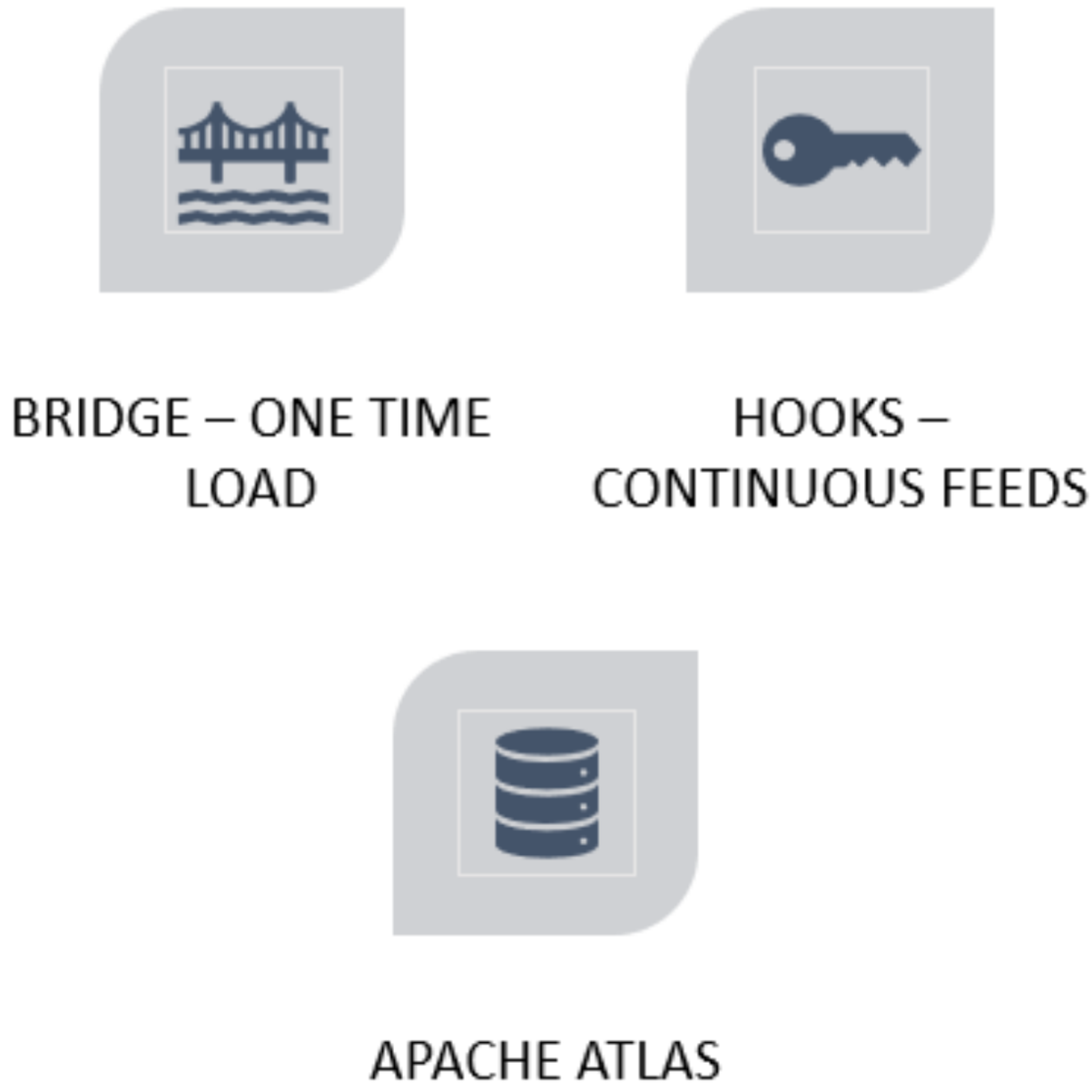
Graph repository to store metadata

Information in JanusGraph is stored as Property Graph model (vertexes, edges, properties), where each type is represented as a vertex and the relationships among the types are represented as edges in the graph

How to import metadata into Apache Atlas

- A bridge is a one time load of metadata from a data platform/service/engine. This is used to perform the initial load of metadata into Atlas for the current state of the data resources

- A hook is an on-going trickle feed of updates to Atlas as the state of the data resources changes in the data platform/ service/engine.



Conclusion:

In end data governance team needs to focus on

- Select tool to opt for Data Governance
- Automation needs to capture metadata from big data streams
- Identify ETL jobs or processes involved in building the streams
- Plan and build classifiers & business glossaries to uniquely identify data
- Tying it up with the security requirements to allow only eligible users to access the data.

This white paper covers the basics of Data Governance and the knowledge gained by AFour from implementations in projects executed till date. If you have queries and requirements do reach out to us through our website: https://afourtech.com/

About the Author

Shailesh Gaikwad

Technical Project Manager with over 17+ years of experience in the areas of analytics, big data, data governance, data quality, master data management, and business intelligence using Azure & AWS Big Data Services like Azure Data Lake Analytics, Azure Databricks, AWS EMR and Apache Spark.

About AFour Technologies

AFour Technologies – Agility. Automation. Attitude. Aptitude.

AFour Technologies (a.k.a. AFour) focuses on Software Product Engineering Services comprising UI/UX Design, Architecture Design and Consulting, Software Development, DevOps, QA and Monitoring, Big Data and Data Analytics technologies / frameworks.

With an excellent track record of over 10 years and a focus on software product engineering, AFour is a true example of self-belief, consistency, and transparency.

AFour is an ideation and technology house. Product companies associate with AFour for its product conceptualization and technology skills in a wide range of technologies like Java, Python, .NET, JavaScript (MEAN), LAMP (Perl and PHP), Angular, React, C++.

The company is a hub of every cutting-edge technology in software engineering — Hyper-convergence, SDN, Virtualization, Next Generation Data Center Technologies, Networking, Enterprise Mobility.