

Hortonworks Data Platform

Data Governance

(July 21, 2015)

Hortonworks Data Platform: Data Governance

Copyright © 2012-2015 Hortonworks, Inc. Some rights reserved.

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, ZooKeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, [training](#) and partner-enablement services. All of our technology is, and will remain, free and open source.

Please visit the [Hortonworks Data Platform](#) page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the [Support](#) or [Training](#) page. Feel free to [contact us](#) directly to discuss your specific needs.



Except where otherwise noted, this document is licensed under
Creative Commons Attribution ShareAlike 4.0 License.
<http://creativecommons.org/licenses/by-sa/4.0/legalcode>

Table of Contents

| | |
|--|----|
| 1. HDP Data Governance | 1 |
| 1.1. Falcon Overview | 1 |
| 1.2. Atlas Overview | 2 |
| 2. Data Pipelines (Falcon) | 4 |
| 2.1. Understanding Data Pipelines | 4 |
| 2.2. Using the Falcon Web UI to Define Data Pipelines | 5 |
| 2.2.1. Creating a Cluster Entity | 7 |
| 2.2.2. Creating a Feed Entity | 9 |
| 2.2.3. Creating a Process Entity | 11 |
| 2.3. Search For and Manage Data Pipeline Entities | 13 |
| 2.4. Mirroring Data (Falcon) | 14 |
| 2.5. Using the Falcon CLI to Define Data Pipelines | 16 |
| 2.5.1. Deploying Data Pipelines | 19 |
| 2.5.2. Replicating Data (Falcon) | 19 |
| 2.5.3. Viewing Alerts in Falcon | 27 |
| 2.5.4. Late Data Handling | 28 |
| 2.5.5. Setting a Retention Policy | 29 |
| 2.5.6. Setting a Retry Policy | 29 |
| 2.6. Understanding Dependencies in Falcon | 30 |
| 2.7. Viewing Dependencies | 30 |
| 3. Metadata Services Framework (Atlas) | 32 |
| 3.1. Understanding the HDP Metadata Services Framework | 32 |
| 3.2. Using the Atlas Web UI to Search Metadata | 33 |
| 4. Reference (Falcon) | 37 |
| 4.1. Cluster | 37 |
| 4.1.1. Valid Cluster Tag Attributes | 37 |
| 4.1.2. Cluster Interfaces | 37 |
| 4.1.3. Cluster XSD Specification | 37 |
| 4.2. Feed Entity | 37 |
| 4.3. Process Entity | 38 |
| 4.4. Using the CLI to Manage Entities and Instances | 38 |
| 4.4.1. Managing Entities with the CLI | 38 |
| 4.4.2. Managing Instances with the CLI | 39 |
| 5. Troubleshooting (Falcon) | 40 |
| 5.1. Falcon logs | 40 |
| 5.2. Falcon Server Failure | 40 |
| 5.3. Delegation Token Renewal Issues | 40 |
| 5.4. Invalid Entity Schema | 40 |
| 5.5. Incorrect Entity | 40 |
| 5.6. Bad Config Store Error | 40 |
| 5.7. Unable to set DataSet Entity | 41 |
| 5.8. Oozie Jobs | 41 |
| 6. Configuring High Availability (Falcon Server) | 42 |
| 6.1. Configuring Properties and Setting Up Directory Structure for High Availability | 42 |
| 6.2. Preparing the Falcon Servers | 43 |
| 6.3. Manually Failing Over the Falcon Servers | 43 |
| 7. Metadata Store REST API Reference (Atlas) | 44 |

| | |
|--------------------------------------|----|
| 7.1. Data Model | 44 |
| 7.2. AdminResource | 44 |
| 7.3. EntityResource | 45 |
| 7.4. HiveLineageResource | 47 |
| 7.5. MetadataDiscoveryResource | 47 |
| 7.6. RexsterGraphResource | 49 |
| 7.7. TypesResource | 50 |

List of Figures

| | |
|---|----|
| 1.1. Falcon Architecture | 2 |
| 1.2. Atlas Overview | 3 |
| 2.1. Data Pipeline | 4 |
| 2.2. Data Pipeline Flow | 5 |
| 2.3. Ambari Dashboard Falcon and Oozie Service Indicators | 6 |
| 2.4. New Cluster Configuration Dialog | 8 |
| 2.5. New Feed Configuration Dialog | 10 |
| 2.6. New Process Configuration Dialog | 12 |
| 2.7. Falcon Search UI | 13 |
| 2.8. New Mirror Configuration Dialog | 15 |
| 2.9. Graph_view.png | 31 |
| 3.1. Atlas Architecture | 32 |
| 3.2. Enter Tag to Search in Atlas Dashboard | 34 |
| 3.3. Click the Tag Link to View Details | 34 |
| 3.4. Details Tab | 35 |
| 3.5. Schema Tab | 35 |
| 3.6. Output Tab | 36 |

List of Tables

| | |
|---|----|
| 2.1. Cluster Entity Configuration Values | 8 |
| 2.2. General Feed Configuration Values | 10 |
| 2.3. General Process Configuration Values | 12 |
| 2.4. Mirror Configuration Values | 16 |
| 2.5. Available Falcon Event Alerts | 27 |
| 4.1. Cluster tag elements | 37 |
| 4.2. Cluster Interfaces | 37 |
| 4.3. Entity Actions | 38 |
| 4.4. Instance Actions | 39 |

1. HDP Data Governance

Enterprises that adopt modern data architectures with Hadoop must reconcile data management realities when they bring existing and new data from disparate platforms under management. As Hadoop is deployed in corporate data and processing environments, metadata and data governance must be vital parts of any enterprise-ready [data lake](#) to realize true value.

In HDP, the overall management of the data life cycle in the platform is achieved by using *data pipelines*, which ingest, move, tag, process, and expire data, and an underlying flexible *metadata store* that manages all data for all components of HDP. This underlying metadata store simplifies data governance for Hadoop because you no longer must create interfaces to each HDP component. Instead, you can program your third-party governance applications to access one HDP metadata store that gives you access to all metadata for the platform.

Data governance in HDP is managed by the following components:

- **Apache Falcon:** solves enterprise challenges related to Hadoop data replication, business continuity, and lineage tracing by deploying a framework for data management and processing. The Falcon framework can also leverage other HDP components, such as Pig, HDFS, and Oozie. Falcon enables this simplified management by providing a framework to define, deploy, and manage data pipelines. Data pipelines contain:
 - A definition of the dataset to be processed.
 - Interfaces to the Hadoop cluster where the data resides.
 - A process definition that defines how the data is consumed and invokes processing logic.
- **Apache Atlas:** extends Falcon's governance capabilities by adding business taxonomical and operational metadata. Atlas is a scalable and extensible set of core governance services that enable enterprises to meet their compliance requirements within the Hadoop stack and to integrate with their data ecosystem outside HDP. Atlas provides:
 - Data classification.
 - Centralized auditing.
 - Search and lineage history.
 - Security and policy engines.

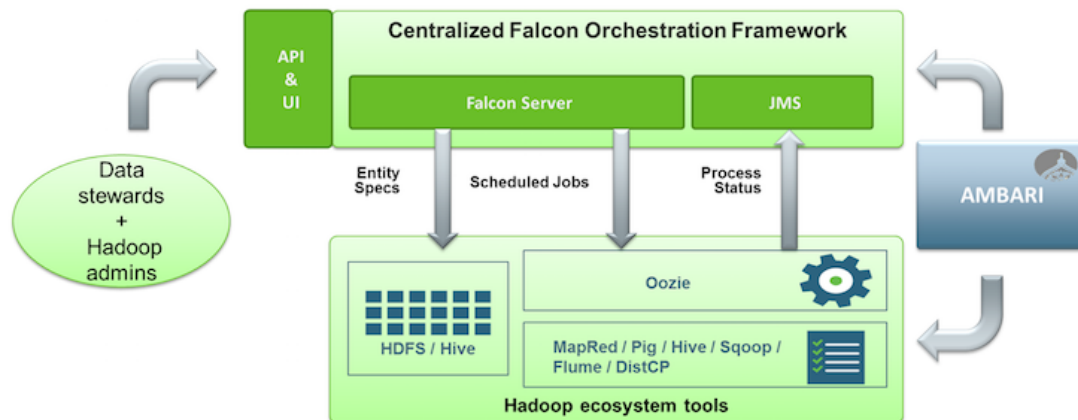
1.1. Falcon Overview

Apache Falcon addresses the following data governance requirements and provides a wizard-like GUI that eliminates hand coding of complex data sets and offers:

- **Centrally manage the data lifecycle:** Falcon enables you to manage the data lifecycle in one common place where you can define and manage policies and pipelines for data ingest, processing, and export.

- **Business continuity and disaster recovery:** Falcon can replicate HDFS and Hive datasets, trigger processes for retry, and handle late data arrival logic. In addition, Falcon can mirror file systems or Hive HCatalog on clusters using recipes that enable you to re-use complex workflows.
- **Address audit and compliance requirements:** Falcon provides audit and compliance features that enable you to visualize data pipeline lineage, track data pipeline audit logs, and tag data with business metadata.

Figure 1.1. Falcon Architecture



Falcon can be installed and managed by Apache Ambari, and jobs can be traced through the native Falcon UI. Falcon can process data from:

- Oozie jobs
- Pig scripts
- Hive scripts

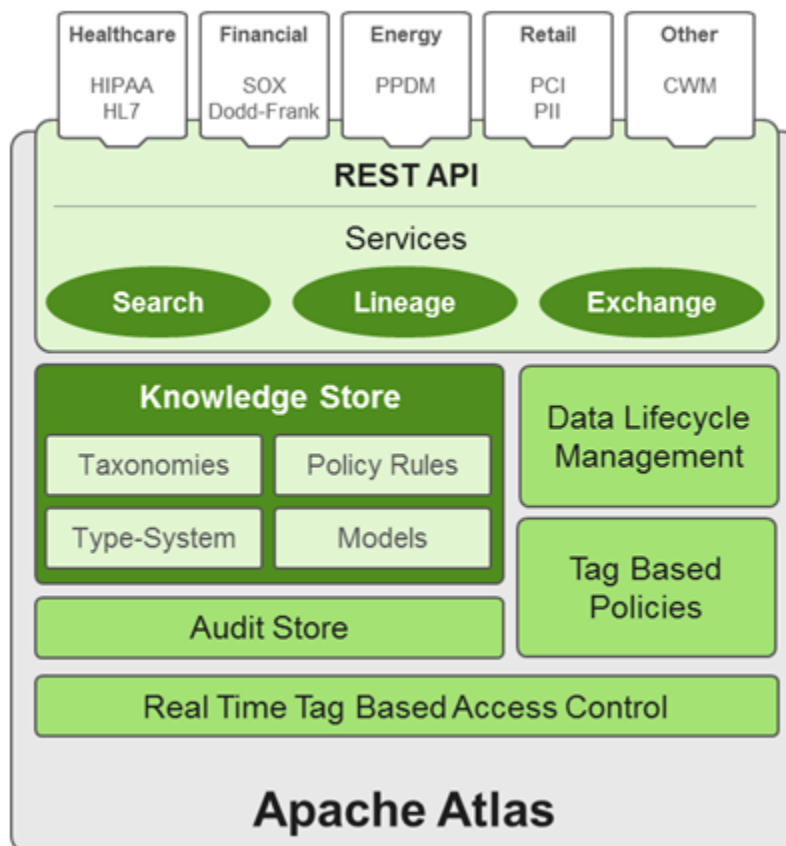
1.2. Atlas Overview

Apache Atlas is a low-level service in the Hadoop stack that provides core metadata services. Initially, Atlas provides metadata services for Hive, but in subsequent releases all components of HDP will be brought under Atlas metadata management. Atlas provides:

- **Knowledge store that leverages existing Hadoop metastores:** Categorized into a business-oriented taxonomy of data sets, objects, tables, and columns. Supports the exchange of metadata between HDP foundation components and third-party applications or governance tools.
- **Data lifecycle management:** Leverages existing investment in Apache Falcon with a focus on provenance, multi-cluster replication, data set retention and eviction, late data handling, and automation.
- **Audit store:** Historical repository for all governance events, including security events (access, grant, deny), operational events related to data provenance and metrics. The Atlas audit store is indexed and searchable for access to governance events.

- **Security:** Integration with HDP security that enables you to establish global security policies based on data classifications and that leverages Apache Ranger plug-in architecture for security policy enforcement.
- **Policy engine:** Fully extensible policy engine that supports metadata-based, geo-based, and time-based rules that rationalize at runtime.
- **RESTful interface:** Supports extensibility by way of REST APIs to third-party applications so you can use your existing tools to view and manipulate metadata in the HDP foundation components.

Figure 1.2. Atlas Overview



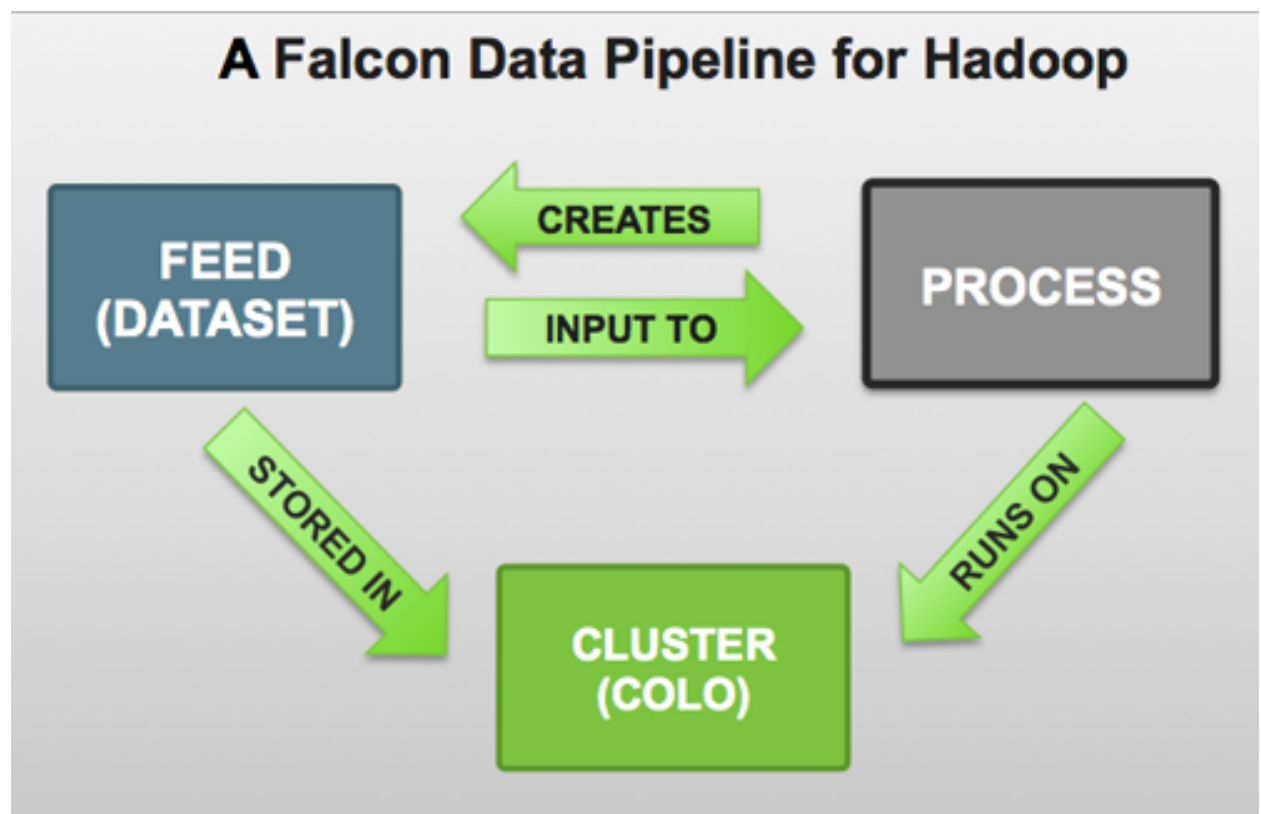
2. Data Pipelines (Falcon)

Data pipelines, which consist of cluster storage location definitions, dataset feeds, and processing logic can be configured using either the Falcon command-line interface (CLI) or the web UI. This topic explains what data pipelines are and how to configure them for data replication and mirroring. Information for using both the web UI and the CLI is included. The CLI commands support automating data pipeline creation.

2.1. Understanding Data Pipelines

A data pipeline consists of a dataset and processing that acts on the dataset across your HDFS cluster.

Figure 2.1. Data Pipeline



Each pipeline consists of XML pipeline specifications, called entities. These entities act together to provide a dynamic flow of information to load, clean, and process data.

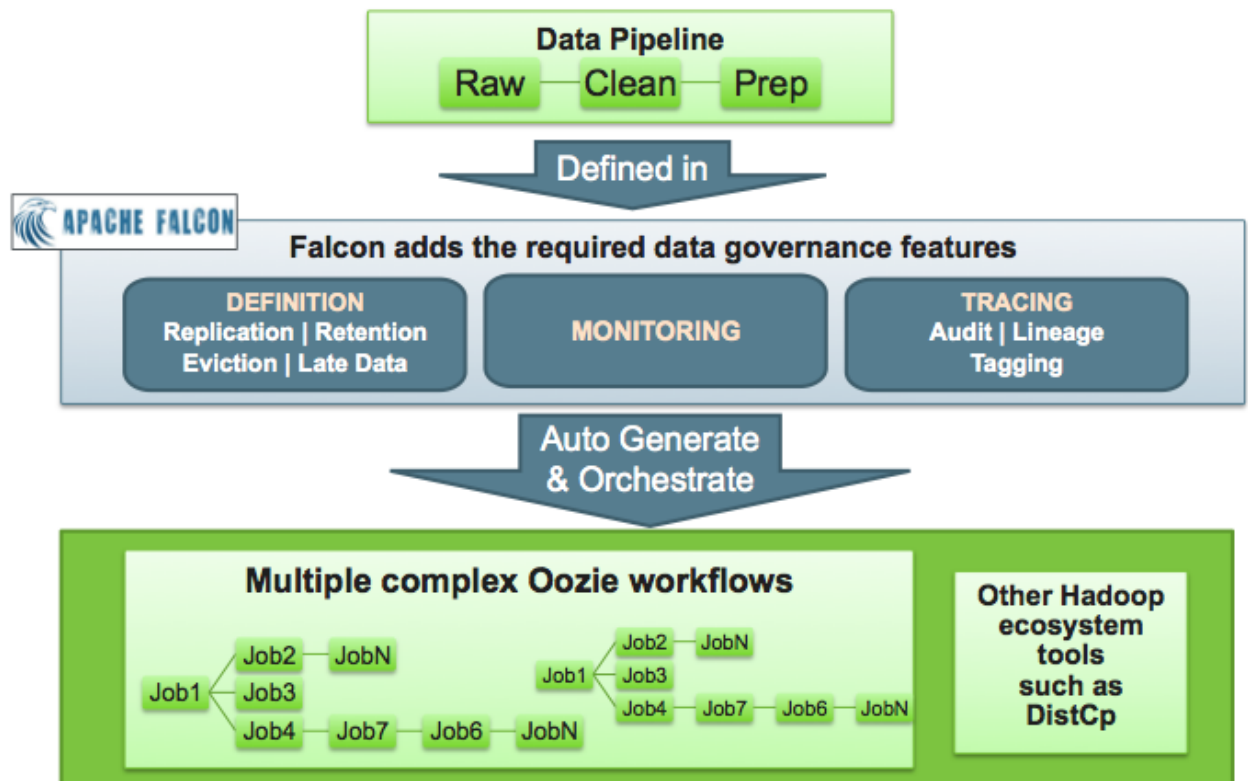
There are three types of entities:

- **Cluster:** Defines where data and processes are stored.
- **Feed:** Defines the datasets to be cleaned and processed.
- **Process:** Consumes feeds, invokes processing logic, and produces further feeds. A process defines the configuration of the Oozie workflow and defines when and how often the workflow should run. Also allows for late data handling.

Each entity is defined separately and then linked together to form a data pipeline. Falcon provides predefined policies for data replication, retention, late data handling, and replication. These sample policies are easily customized to suit your needs.

These entities can be reused many times to define data management policies for Oozie jobs, Pig scripts, and Hive queries. For example, Falcon data management policies become Oozie coordinator jobs:

Figure 2.2. Data Pipeline Flow



2.2. Using the Falcon Web UI to Define Data Pipelines

The Falcon web UI enables you to define and deploy data pipelines. Using the web UI ensures that the XML definition file that you use to deploy the data pipeline to the Falcon server is well-formed.



Important

This section is intended as a quick start for the new Falcon web UI. The content will be updated on a regular cadence over the new few months.

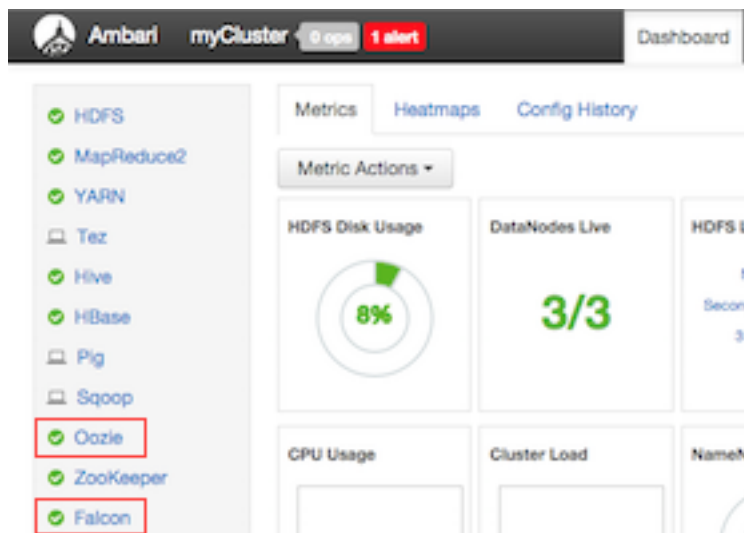
Prerequisite Setup Steps:

Before you define a data pipeline, a system administrator must:

- Make sure that you have the following components installed on your cluster:

- HDP
- Falcon
- Oozie client and server
- Make sure that the Falcon and Oozie services are running. For example, if you are using Ambari, confirm that the Falcon and Oozie services have green check marks adjacent to them on the Ambari dashboard:

Figure 2.3. Ambari Dashboard Falcon and Oozie Service Indicators



- Create the directory structure on HDFS for the staging, temp, and working folders where the cluster entity stores the dataset. These folders must be owned by the **falcon** user.

For example:

```
sudo su falcon
hadoop fs mkdir -p /apps/falcon/primary_Cluster/staging
hadoop fs mkdir -p /apps/falcon/primary_Cluster/working
hadoop fs mkdir -p /apps/falcon/tmp
```

These commands create the following directories that are owned by the **falcon** user:

/apps/falcon/primary_Cluster/staging

/apps/falcon/primary_Cluster/working

/apps/falcon/tmp



Important

Permissions on the cluster staging directory must be set to 777 (read/write/execute for owner/group/others). Only Oozie job definitions are written to the staging directory so setting permissions to 777 does not create any vulnerability.

Run:

```
hadoop fs -chmod -R 777 <your_staging_directory_path>
```

- Launch the Falcon web UI. If you are using Ambari:
 1. On the Services tab, select **Falcon** in the services list.
 2. At the top of the Falcon service page, click **Quick Links**, and then click **Falcon Web UI**.

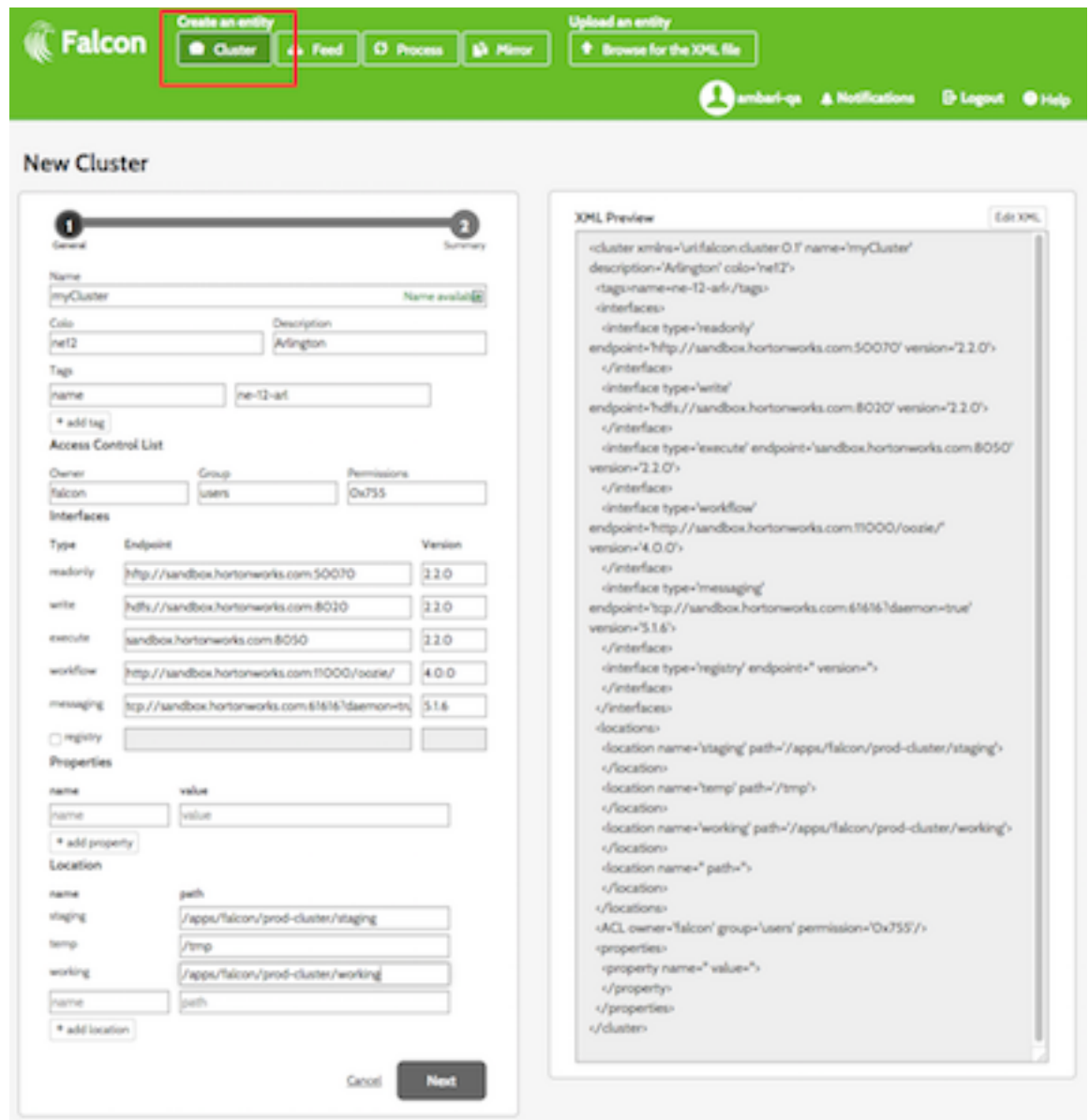
2.2.1. Creating a Cluster Entity

Always specify a cluster entity before defining other elements in your data pipeline. The cluster entity defines where the data and the processes for your data pipeline are stored. For more information, see the cluster entity XSD [here](#).

To use the Falcon web UI to define a cluster entity:

1. At the top of the Falcon web UI page, click **Cluster**.

Figure 2.4. New Cluster Configuration Dialog



2. On the New Cluster page, specify the following values:

Table 2.1. Cluster Entity Configuration Values

| Value | Description |
|-----------------------------|--|
| Name | Name of the cluster entity. Not necessarily the actual cluster name. |
| Colo and Description | Name and description of the data center. |
| Tags | Metadata tagging. |
| Access Control List | Specify the HDFS access permissions. |
| Interfaces | Specify the interface types: |

| Value | Description |
|-------------------|---|
| | <ul style="list-style-type: none"> • readonly – Required for distcp (distributed copy) used in replication. • write –Required to write to HDFS. • execute –Required to write jobs to MapReduce. • workflow –Required. This interface submits Oozie jobs. • messaging –Required to send alerts. • registry –Required to register or deregister partitions in the Hive Metastore and to fetch events on partition availability. |
| Properties | Specify a name and value for each property. |
| Location | Specify HDFS locations for the staging, temp, and working directories. For more information, see Prerequisite Setup Steps [5] . |

3. Click **Next** to view a summary of your cluster entity definition. The XML file is displayed to the right of the summary. Click **Edit XML** to edit the XML directly.
4. If you are satisfied with the cluster entity definition, click **Save**.
5. To verify that you successfully created the cluster entity, enter the cluster entity name in the Falcon web UI Search well and press Enter. If the cluster entity name appears in the search results, it was successfully created. See [Search For and Manage Data Pipeline Entities](#).

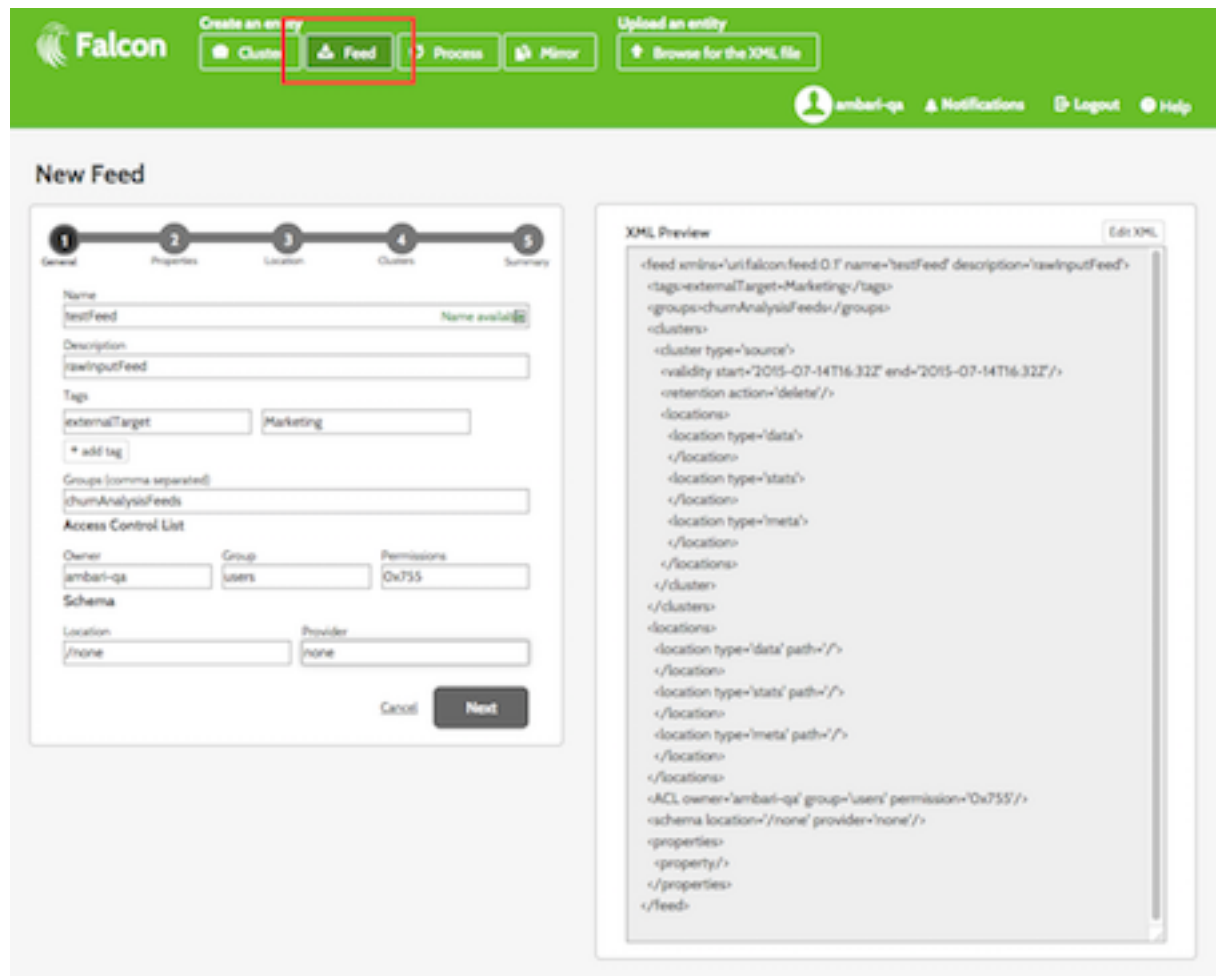
2.2.2. Creating a Feed Entity

The feed entity defines the datasets that are cleaned and processed in your data pipeline. For more information, see the feed entity XSD [here](#).

To use the Falcon web UI to define a feed entity:

1. At the top of the Falcon web UI page, click **Feed**.

Figure 2.5. New Feed Configuration Dialog



2. On the New Feed page, specify the following values:

Table 2.2. General Feed Configuration Values

| Value | Description |
|-----------------------------|--|
| Name and Description | Name and description of the feed entity. |
| Tags | Metadata tagging. For example, you can set the key to "externalTarget" and the corresponding value to "Marketing" tagging this feed for marketing. |
| Groups | Specify the feed group. Feeds can belong to multiple groups. |
| Access Control List | Specify the HDFS access permissions. Required for HDFS. |
| Schema | Specify the schema location and provider. This is required for HDFS. |

3. Click **Next** to advance to the **Properties** configuration where you can configure the timing and other feed properties.

4. Click **Next** to advance to the **Location** configuration where you can specify the global location across clusters. For HDFS paths, choose **File System** and for Hive tables, choose

Catalog Storage. For example, to specify a data path for a File System location, in the **Data path** text box, enter `/weblogs/${YEAR}-${MONTH}-${DAY}-${HOUR}` to point to the web logs.

5. Click **Next** to advance to the **Clusters** configuration where you can:
 - Select the target cluster entity that you defined in [Creating a Cluster Entity](#) for retention or replication.
 - Specify the **Storage type** and **Location**. If you do not specify a location, the location that you specified in the Properties configuration is used.
 - Select the **Validity** interval.
6. Click **Next** to view a summary of your feed entity definition. The XML file is displayed to the right of the summary. Click **Edit XML** to edit the XML directly.
7. If you are satisfied with the feed entity definition, click **Save**.
8. To verify that you successfully created the feed entity, enter the feed entity name in the Falcon web UI Search well and press Enter. If the feed entity name appears in the search results, it was successfully created. See [Search For and Manage Data Pipeline Entities](#).

2.2.3. Creating a Process Entity

The process entity consumes the feeds, invokes processing logic, and can produce additional feeds. For more information, see the process entity XSD [here](#)

To use the Falcon web UI to define a process entity:

1. At the top of the Falcon web UI page, click **Process**.

Figure 2.6. New Process Configuration Dialog

The screenshot shows the 'New Process' configuration dialog in the Falcon interface. The 'Process' button in the top navigation bar is highlighted with a red box. The dialog is divided into five steps: General, Properties, Clusters, Inputs & Outputs, and Summary. The 'General' step is currently active. The form contains the following fields and values:

- Name:** process-test (Name available)
- Tags:** Key: key, Value: value
- Workflow Name:** testWorkflow
- Engine:** Pig (selected), Version: pig-0.10.0
- Path:** /apps/clickstream/clean-script.pig
- Access Control List:** Owner: ambari-qa, Group: users, Permissions: 0x755

An XML Preview pane on the right shows the following XML configuration:

```
<process xmlns="uri:falcon:process:0.7" name="process-test">
  <clusters>
    <cluster>
      <validity start="2015-07-14T16:48Z" end="2015-07-14T16:48Z"/>
    </cluster>
  </clusters>
  <parallel>1</parallel>
  <workflow name="testWorkflow" version="pig-0.10.0" engine="pig"
    path="/apps/clickstream/clean-script.pig"/>
  <ACL owner="ambari-qa" group="users" permission="0x755"/>
</process>
```

- On the New Process page, specify the following values:

Table 2.3. General Process Configuration Values

| Value | Description |
|----------------------------|--|
| Name | Name of the process entity. |
| Tags | Business labels, such as "Finance." There is no input validation on this field, so there can be duplicates, which is resolved in environments with Apache Atlas integration. See Configuring, Using, and Managing the Metadata Store (Atlas) . |
| Workflow | Specify a Name for the workflow, which Engine it uses, and the Path to the workflow engine. For example, if you are using a Pig script to define the workflow, you can set the Path to /apps/clickstream/clean-script.pig |
| Access Control List | Specify the HDFS access permissions. Required for HDFS. |

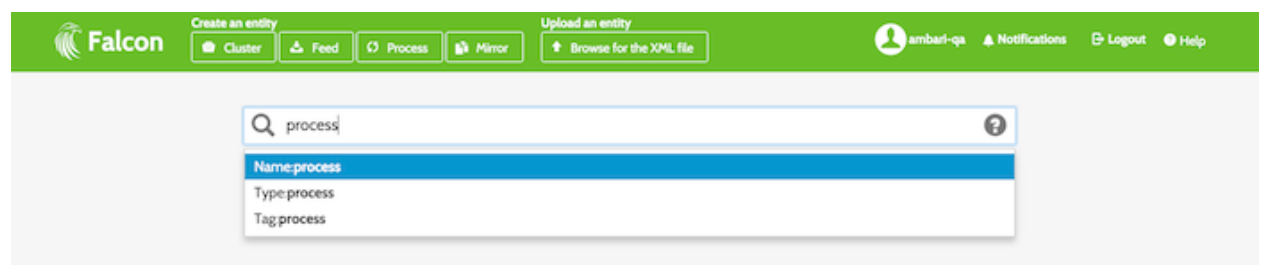
- Click **Next** to advance to the **Properties** configuration where you can configure the time zone, timing, and retry policy.
- Click **Next** to advance to the **Clusters** configuration where you can:
 - Select the target cluster entity that you defined in [Creating a Cluster Entity](#) to specify where the process runs.
 - Select the **Validity** interval.

5. Click **Next** to advance to the **Inputs & Outputs** configuration where you can configure:
 - **Inputs:** Feeds that are consumed by the process.
 - **Outputs:** Feeds that are generated and output by the process.
6. Click **Next** to view a summary of your process entity definition. The XML file is displayed to the right of the summary. Click **Edit XML** to edit the XML directly.
7. If you are satisfied with the process entity definition, click **Save**.
8. To verify that you successfully created the process entity, enter the process entity name in the Falcon web UI Search well and press Enter. If the process entity name appears in the search results, it was successfully created. See [Search For and Manage Data Pipeline Entities](#).

2.3. Search For and Manage Data Pipeline Entities

The best way to search for and manage data pipeline entities is by using the Falcon web UI.

Figure 2.7. Falcon Search UI



To search for and manage data pipeline entities with the Falcon web UI:

1. Launch the Falcon web UI. If you are using Ambari:
 - a. On the Services tab, select **Falcon** in the services list.
 - b. At the top of the Falcon service page, click **Quick Links**, and then click **Falcon Web UI**.
2. Enter your query in the **Search** well, and press **Enter**.

You can filter entities based on names, types, or tags. By default, the first argument in your query is the Name filter. Wildcards are supported, such as asterisk (*). The search is interactive so you can refine your search by adding and removing tags to tune your result set.

| Filter | Description |
|--------|--|
| Name | Subsequence of the entity name (cluster, feed, or process name). Not case sensitive. The entity name must contain all of the characters in the subsequence in the same order as the original sequence from which it derives. For example: |

| Filter | Description |
|--------|--|
| | <ul style="list-style-type: none"> "sample1" matches the entity named "SampleFeed1-2" "mhs" matches the entity named "New-My-Hourly-Summary" |
| Tag | Keywords in metadata tags. Not case sensitive. Entities that are returned in search results have tags that match all of the tag keywords. |
| Type | Specifies the type of entity. Valid entity types are cluster, feed, or process. The Falcon search UI infers the type filter automatically. For example, to add a "process" filter type, enter process in the search well, and then choose type:process from the hints offered in the UI as shown in the previous screen capture. |

- Select entities in the search results and then select the action you want to perform. Depending on the type of entity you select, you can schedule, resume, pause, edit, copy, delete, or download the XML. In addition, when you click on an entity in the search results, you can view its instances and property details.
- Click the Falcon icon in the upper left corner of the window to exit the search results and start a new search.



Note

Click tags in the search results to add them to the search well so you can search on the tag definitions.

2.4. Mirroring Data (Falcon)

Mirroring data produces an exact copy of the data and keeps both copies synchronized. You can use Falcon to mirror HDFS directories or Hive tables and you can mirror between HDFS and Amazon S3 or Microsoft Azure. A whole database replication can be performed with Hive.

To mirror data with the Falcon web UI:

- Launch the Falcon web UI. If you are using Ambari:
 - On the Services tab, select **Falcon** in the services list.
 - At the top of the Falcon service page, click **Quick Links**, and then click **Falcon Web UI**.
- At the top of the Falcon web UI page, click **Mirror**.

Figure 2.8. New Mirror Configuration Dialog

The screenshot shows the 'New Mirror' configuration dialog in the Falcon interface. The top navigation bar includes 'Cluster', 'Feed', 'Process', and 'Mirror' (highlighted with a red box), along with an 'Upload an entity' section for XML files. The dialog is divided into two steps: 'General' (step 1) and 'Summary' (step 2). The 'General' step contains the following fields and options:

- Mirror Name:** A text input field.
- Tags:** A field with 'key' and 'value' inputs and an '+ add tag' button.
- _falcon_mirroring_type:** A dropdown menu currently set to 'HDFS'.
- Minor type:** Radio buttons for 'File System' (selected) and 'HIVE(catalog Storage)'.
- Source:** A section with a 'Run job here' radio button (unselected). It includes a 'Location' dropdown (HDFS selected), a '-Select cluster-' dropdown, and a 'Path' input field.
- Target:** A section with a 'Run job here' radio button (selected). It includes a 'Location' dropdown (HDFS selected), a '-Select cluster-' dropdown, and a 'Path' input field.
- Validity:** Fields for 'Start' (07/14/2015 01:16 PM) and 'End' (mm/dd/yyyy 01:16 PM), with a '(GMT) West' time zone selector.
- Send alerts to:** An 'Email' input field and an '+add alert' button.
- Advanced options:** A collapsed section indicated by a downward arrow.

At the bottom right of the dialog, there are 'Cancel' and 'Next' buttons.

3. On the New Mirror page, specify the following values:

Table 2.4. Mirror Configuration Values

| Value | Description |
|-------------------------|--|
| Mirror Name | Name of the mirror entity. |
| Tags | Metadata tagging. An example is provided in the UI. |
| Mirror Type | Select whether this is a File System or Hive catalog mirror type. |
| Source | Specify the location, name, and path of the cluster or Hive table that is to be mirrored, and specify if the mirroring job runs on the source cluster. |
| Target | Specify the location, name, and path where the mirrored cluster is stored, and specify if the mirroring job runs on the target cluster. |
| Validity | Specify the validity interval. |
| Advanced Options | Expand the Advanced Options section of the page to configure how often the target cluster is updated, throttle distcp operations, set a retry policy, and specify the ACL for the mirror entity. |

4. Click **Next** to view a summary of your mirror entity definition.
5. If you are satisfied with the mirror entity definition, click **Save**.
6. To verify that you successfully created the mirror entity, enter the mirror entity name in the Falcon web UI Search well and press Enter. If the mirror entity name appears in the search results, it was successfully created. See [Search For and Manage Data Pipeline Entities](#).

2.5. Using the Falcon CLI to Define Data Pipelines

To use the Falcon CLI to define a data pipeline:

1. Create the cluster specification XML file, also known as a cluster entity. There are several interfaces to define in a cluster entity. For example, here is a cluster entity with all cluster interfaces defined:
 - **Colo:** Name of the Data Center
 - **Name:** Filename of the Data Center
 - **<interface>:** Specify the interface type



Important

Permissions on the cluster staging directory must be set to 777 (read/write/execute for owner/group/others). Only Oozie job definitions are written to the staging directory so setting permissions to 777 does not create any vulnerability.

```
<?xml version="1.0"?>
<!--
  Cluster Example
-->
```

```

<cluster colo="$MyDataCenter" description="description" name=
"$MyDataCenter">
  <interfaces>
    <interface type="readonly" endpoint="hftp://nn:50070" version="2.4.0" />
    <!-- Required for distcp for replications. -->
    <interface type="write" endpoint="hdfs://nn:8020" version="2.4.0" />
    <!-- Needed for writing to HDFS-->
    <interface type="execute" endpoint="rm:8050" version="2.4.0" /> <!--
    Needed to write to jobs as MapReduce-->
    <interface type="workflow" endpoint="http://os:11000/oozie/" version="4.
    0.0" /> <!-- Required. Submits Oozie jobs.-->
    <interface type="registry" endpoint="thrift://hms:9083" version="0.
    13.0" /> <!--Register/deregister partitions in the Hive Metastore and get
    events on partition availability
    -->
    <interface type="messaging" endpoint="tcp://mq:61616?daemon=true"
    version="5.1.6" /> <!--Needed for alerts-->
  </interfaces>
  <locations>
    <location name="staging" path="/apps/falcon/prod-cluster/staging" />
    <!--HDFS directories used by the Falcon server-->
    <location name="temp" path="/tmp" />
    <location name="working" path="/apps/falcon/prod-cluster/working" />
  </locations>
</cluster>

```



Note

Additional properties must be set if you are configuring for a secure cluster. For more information, see "Configuring for Secure Clusters" in the [Installing HDP Manually](#) guide.

2. Next, create a dataset specification XML file, or feed entity:

- Reference the cluster entity to determine which clusters the feed uses.
- **<frequency>**: Specify the frequency of the feed.
- **<retention limit>**: Choose a retention policy for the data to remain on the cluster.
- **<location>**: Provide the HDFS path to the files.
- **<ACL owner>**: Specify the HDFS access permissions.
- Optional. Specify a [Late Data Handling](#) cut-off.

```

<?xml version="1.0"?>
<!--
  Feed Example
-->
<feed description="$rawInputFeed" name="testFeed" xmlns="uri:falcon:feed:0.
1">
  <frequency>hours(1)</frequency> <!--Feed run frequency-->
  <late-arrival cut-off="hours(6)"/> <!-- Late arrival cut-off -->
  <groups>churnAnalysisFeeds</groups> <!--Feed group, feeds can belong to
  multiple groups -->
  <tags externalSource=$MyEDW, externalTarget=Marketing> <!-- Metadata
  tagging -->

```

```

<clusters> <!-- Target clusters for retention and replication. -->
  <cluster name="$MyDataCenter" type="source">
    <validity start="$date" end="$date"/>
    <retention limit="days($n)" action="delete"> <!--Currently delete is
the only action available -->
  </cluster>
  <cluster name="$MyDataCenter-secondary" type="target">
    <validity start="2012-01-01T00:00Z" end="2099-12-31T00:00Z"/>
    <location type="data" path="/churn/weblogs/${YEAR}-${MONTH}-${DAY}-
${HOURL}"/>
    <retention limit="days(7)" action="delete"/>
  </cluster>
</clusters>
<locations> <!-- Global location across clusters - HDFS paths or Hive
tables -->
  <location type="data" path="/weblogs/${YEAR}-${MONTH}-${DAY}-${HOURL}"/>
</locations>
<ACL owner="hdfs" group="users" permission="0755"/> <!-- Required for
HDFS. -->
<schema location="/none" provider="none"/> <!-- Required for HDFS. -->
</feed>

```

3. Create the process specification XML file:

- **<cluster name>**: Reference the cluster entity to define where the process runs.
- **<feed>**: Reference the feed entity to define the datasets that the process uses.
- Optional. Specify [Late Data Handling](#) policies or a [Retry Policy](#).

```

<?xml version="1.0"?>
<!--
  Process Example
-->
<process name="process-test" xmlns="uri:falcon:process:0.1">
  <clusters>
    <cluster name="$MyDataCenter">
      <validity start="2011-11-02T00:00Z" end="2011-12-30T00:00Z"
    </cluster>
  </clusters>
  <parallel>1</parallel>
  <order>FIFO</order> <!--You can also use LIFO and LASTONLY but FIFO is
recommended in most cases-->
  <frequency>days(1)</frequency>
  <inputs>
    <input end="today(0,0)" start="today(0,0)" feed="feed-clicks-raw"
name="input" />
  </inputs>
  <outputs>
    <output instance="now(0,2)" feed="feed-clicks-clean" name="output" /
>
  </outputs>
  <workflow engine="pig" path="/apps/clickstream/clean-script.pig" />
  <retry policy="periodic" delay="minutes(10)" attempts="3"/>
  <late-process policy="exp-backoff" delay="hours(1)">
  <late-input input="input" workflow-path="/apps/clickstream/late" />
  </late-process>
</process>

```




Note

LIFO and LASTONLY are also supported schedule changes for <order>.

You can now move on to [Deploying Data Pipelines](#).

2.5.1. Deploying Data Pipelines

After you create your data pipeline with Falcon, you can deploy it with the Falcon CLI.

To deploy the data pipeline:

1. Submit your entities to Falcon. Be sure to specify the correct entity type.

a. Submit your cluster entity.

For example, to submit \$sampleClusterFile.xml:

```
falcon entity -type cluster -submit -file $sampleClusterFile.xml
```

b. Submit your dataset or feed entity.

For example, to submit \$sampleFeedFile.xml:

```
falcon entity -type feed -submit -file $sampleFeedFile.xml
```

c. Submit your process entity.

For example, to submit \$sampleProcessFile.xml:

```
falcon entity -type process -submit -file $sampleProcessFile.xml
```

2. Schedule your feed and process entities.

a. Schedule your feed.

For example, to schedule \$feedName:

```
falcon entity -type feed -schedule -name $feedName
```

b. Schedule your process.

For example, to schedule \$processName:

```
falcon entity -type process -schedule -name $processName
```

Your data pipeline is now deployed with basic necessary information to run Oozie jobs, Pig scripts, and Hive queries. You can now explore other sections such as [Late Data Handling](#) or [Retry Policy](#).

2.5.2. Replicating Data (Falcon)

Falcon can replicate data across multiple clusters using distcp, and do it according to the frequency you specify in the feed entity. Falcon uses a pull-based replication mechanism,

meaning in every target cluster, for a given source cluster, a coordinator is scheduled which pulls the data using distcp from source cluster.

2.5.2.1. Prerequisites

Before you begin setting up Data Replication, that you have the following components installed on your cluster:

- **HDP.** Installed on your cluster (using Ambari or a Manual Install)
- **Falcon.** Installed on your cluster and the Falcon Service is running.
- **Oozie Client and Server.** Installed on your cluster and the Oozie Service is running on your cluster.

2.5.2.2. Define the Data Source: Set Up a Source Cluster Entity

Define where data and processes are stored in the cluster entity.

1. Create an XML file for the Cluster entity. This file contains all properties for the cluster. Include the XML version:

```
<?xml version="1.0"?>
```

2. Define the `colo` and `name` attributes for the cluster.

```
<?xml version="1.0"?>
<cluster colo="<MyDataCenter>" description="description"
        name="<MyDataCenterFilename>">
</cluster>
```



Note

`colo` specifies the data center to which this cluster belongs.

`name` is the name of the cluster, which must be unique.

3. Define the interfaces for the cluster. For each interface specify type of interface, endpoint, and Apache version.

For example:

```
<cluster colo="<MyDataCenter>" description="description"
        name="<MyDataCenterFilename>">
  <interfaces>

    <!-- Required for distcp for replications. -->
    <interface type="readonly" endpoint="hftp://nn:50070" version="2.
4.0" />

    <!-- Needed for writing to HDFS-->
    <interface type="write" endpoint="hdfs://nn:8020" version="2.4.
0" />

    <!-- Required. An execute interface specifies the interface for
job tracker.-->
```

```

    <interface type="execute" endpoint="rm:8050" version="2.4.0" />

    <!-- Required. A workflow interface specifies the interface for
    workflow engines, such as Oozie.-->
    <interface type="workflow" endpoint="http://os:11000/oozie/"
    version="4.0.0" />

    <!--A registry interface specifies the interface for the metadata
    catalog, such as Hive Metastore or HCatalog.-->
    <interface type="registry" endpoint="thrift://hms:9083" version=
    "0.13.0" />

    <!--Messaging interface specifies the interface for sending
    alerts.-->
    <interface type="messaging" endpoint="tcp://mq:61616?daemon=true"
    version="5.1.6" />
    </interfaces>
</cluster>

```

4. Provide the locations for the HDFS paths to files.

For example:

```

<cluster colo="<MyDataCenter>" description="description"
    name="<MyDataCenter>">
  <interfaces>

    <!-- Required for distcp for replications. -->
    <interface type="readonly" endpoint="hftp://nn:50070" version="2.
4.0" />

    <!-- Needed for writing to HDFS-->
    <interface type="write" endpoint="hdfs://nn:8020" version="2.4.
0" />

    <!-- Needed to write to jobs as MapReduce-->
    <interface type="execute" endpoint="rm:8050" version="2.4.0" />

    <!-- Required. Submits Oozie jobs.-->
    <interface type="workflow" endpoint="http://os:11000/oozie/"
    version="4.0.0" />

    <!--Register/deregister partitions in the Hive Metastore and get
    events on partition availability-->
    <interface type="registry" endpoint="thrift://hms:9083" version=
    "0.13.0" />

    <!--Needed for alerts-->
    <interface type="messaging" endpoint="tcp://mq:61616?daemon=true"
    version="5.1.6" />
    </interfaces>

    <locations>

    <!--HDFS directories used by the Falcon server-->
    <location name="staging" path="/apps/falcon/prod-cluster/
staging" />
    <location name="temp" path="/tmp" />
    <location name="working" path="/apps/falcon/prod-cluster/
working" />

```

```

    </locations>
</cluster>

```

The cluster entity is complete if you are using a non-secure environment. If you are using an environment that is secured with Kerberos, continue on with the next step.

5. For secure clusters, define the following properties in all your cluster entities as shown below:

```

<cluster colo="<MyDataCenter>" description="description"
    name="<MyDataCenter>">

    <interfaces>

        <!-- Required for distcp for replications. -->
        <interface type="readonly" endpoint="hftp://nn:50070" version="2.
4.0" />

        <!-- Needed for writing to HDFS-->
        <interface type="write" endpoint="hdfs://nn:8020" version="2.4.
0" />

        <!-- Needed to write to jobs as MapReduce-->
        <interface type="execute" endpoint="rm:8050" version="2.4.0" />

        <!-- Required. Submits Oozie jobs.-->
        <interface type="workflow" endpoint="http://os:11000/oozie/"
version="4.0.0" />

        <!--Register/deregister partitions in the Hive Metastore and get
events on partition availability-->
        <interface type="registry" endpoint="thrift://hms:9083" version=
"0.13.0" />

        <!--Needed for alerts-->
        <interface type="messaging" endpoint="tcp://mq:61616?daemon=true"
version="5.1.6" />
    </interfaces>

    <locations>

        <!--HDFS directories used by the Falcon server-->
        <location name="staging" path="/apps/falcon/prod-cluster/
staging" />
        <location name="temp" path="/tmp" />
        <location name="working" path="/apps/falcon/prod-cluster/
working" />
    </locations>

    <properties>
        <property name="dfs.namenode.kerberos.principal" value="nn/$my.
internal@EXAMPLE.COM"/>
        <property name="hive.metastore.kerberos.principal" value="hive/
$my.internal@EXAMPLE.COM"/>
        <property name="hive.metastore.uris" value="thrift://$my.
internal:9083"/>
        <property name="hive.metastore.sasl.enabled" value="true"/>
    </properties>
</cluster>

```

Replace `$my.internal@EXAMPLE.COM` and `$my.internal` with your own values.



Important

Make sure `hadoop.security.auth_to_local` in `core-site.xml` is consistent across all clusters. Inconsistencies in rules for `hadoop.security.auth_to_local` can lead to issues with delegation token renewals.

2.5.2.3. Create the Replication Target: Define a Cluster Entity

Replication targets must also be defined as cluster entities. These entities include:

- `colo` and `name` attributes for the cluster.
- Interfaces for the cluster.
- Locations for the HDFS paths to files.
- (For secure clusters only) security properties.

2.5.2.4. Create the Feed Entity

The feed entity defines the data set that Falcon replicates. Reference your cluster entities to determine which clusters the feed uses.

1. Create an XML file for the Feed entity.

```
<?xml version="1.0"?>
```

2. Describe the feed.

```
<?xml version="1.0"?>
<feed description="$rawInputFeed" name="testFeed" xmlns="uri:falcon:feed:0.1">
  1">
</feed>
```

3. Specify the frequency of the feed.

```
<?xml version="1.0"?>
<feed description="$rawInputFeed" name="testFeed" xmlns="uri:falcon:feed:0.1">
  1">

<!--Feed run frequency-->
<frequency>hours(1)</frequency>

</feed>
```

4. Choose a retention policy for the data to remain on the cluster.

For example:

```
<?xml version="1.0"?>
<feed description="$rawInputFeed" name="testFeed" xmlns="uri:falcon:feed:0.1">
  1">

<!--Feed run frequency-->
```

```
<frequency>hours(1)</frequency>

</feed>
```

5. (Optional) Set a late-arrival cut-off policy. The supported policies for late data handling are backoff, exp-backoff (default), and final.

For example, to set the policy to a late cutoff of 6 hours:

```
<?xml version="1.0"?>
<feed description="$rawInputFeed" name="testFeed" xmlns="uri:falcon:feed:0.1">

<!--Feed run frequency-->
<frequency>hours(1)</frequency>

<!-- Late arrival cut-off -->
<late-arrival cut-off="hours(6)"/>

</feed>
```

6. Define your source and target clusters for the feed.

For example, for two clusters, MyDataCenter and MyDataCenter-secondary cluster:

```
<?xml version="1.0"?>
<feed description="$rawInputFeed" name="testFeed" xmlns="uri:falcon:feed:0.1">

<!--Feed run frequency-->
<frequency>hours(1)</frequency>

<!-- Late arrival cut-off -->
<late-arrival cut-off="hours(6)"/>

<!-- Target clusters for retention and replication. -->
<clusters>
  <cluster name="MyDataCenter" type="source">
    <validity start="$date" end="$date"/>

    <!--Currently delete is the only action available -->
    <retention limit="days($n)" action="delete">
  </cluster>

  <cluster name="$MyDataCenter-secondary" type="target">
    <validity start="2012-01-01T00:00Z" end="2099-12-31T00:00Z"/>
    <location type="data" path="/churn/weblogs/${YEAR}-${MONTH}-${DAY}-${HOURL}" />
    <retention limit="days(7)" action="delete"/>
  </cluster>
</clusters>
</feed>
```

7. Specify the HDFS weblogs path locations or Hive table locations. For example to specify the HDFS weblogs location:

```
<?xml version="1.0"?>

<feed description="$rawInputFeed" name="testFeed" xmlns="uri:falcon:feed:0.1">
```

```

<!--Feed run frequency-->
<frequency>hours(1)</frequency>

<!-- Late arrival cut-off -->
<late-arrival cut-off="hours(6)"/>

<!-- Target clusters for retention and replication. -->
<clusters>
  <cluster name="<MyDataCenter>" type="source">
    <validity start="$date" end="$date"/>

    <!--Currently delete is the only action available -->
    <retention limit="days($n)" action="delete">
  </cluster>

  <cluster name="$MyDataCenter-secondary" type="target">
    <validity start="2012-01-01T00:00Z" end="2099-12-31T00:00Z"/>
    <location type="data" path="/churn/weblogs/${YEAR}-${MONTH}-
${DAY}-${HOURL}" />
    <retention limit="days(7)" action="delete"/>
  </cluster>
</clusters> <locations>

<!-- Global location across clusters - HDFS paths or Hive tables -->
<location type="data" path="/weblogs/${YEAR}-${MONTH}-${DAY}-${HOURL}" />
</locations>
</feed>

```

8. Specify HDFS ACLs. Set the owner, group, and level of permissions for HDFS. For example:

```

<?xml version="1.0"?>
<feed description="$rawInputFeed" name="testFeed" xmlns="uri:falcon:feed:0.1">

<!--Feed run frequency-->
<frequency>hours(1)</frequency>

<!-- Late arrival cut-off -->
<late-arrival cut-off="hours(6)"/>

<!-- Target clusters for retention and replication. -->
<clusters>
  <cluster name="<MyDataCenter>" type="source">
    <validity start="$date" end="$date"/>

    <!--Currently delete is the only action available -->
    <retention limit="days($n)" action="delete">
  </cluster>

  <cluster name="$MyDataCenter-secondary" type="target">
    <validity start="2012-01-01T00:00Z" end="2099-12-31T00:00Z"/>
    <location type="data" path="/churn/weblogs/${YEAR}-${MONTH}-${DAY}-
${HOURL}" />
    <retention limit="days(7)" action="delete"/>
  </cluster>
</clusters>

<!-- Global location across clusters - HDFS paths or Hive tables -->

```

```

<locations>
  <location type="data" path="/weblogs/${YEAR}-${MONTH}-${DAY}-${HOUR} " /
>
</locations>

<!-- Required for HDFS. -->
<ACL owner="hdfs" group="users" permission="0755"/>

</feed>

```

9. Specify the location of the schema file for the feed as well as the provider of the schema like protobuf, thrift etc. For example:

```

<?xml version="1.0"?>
<feed description="$rawInputFeed" name="testFeed" xmlns="uri:falcon:feed:0.1">

<!--Feed run frequency-->
<frequency>hours(1)</frequency>

<!-- Late arrival cut-off -->
<late-arrival cut-off="hours(6)"/>

<!-- Target clusters for retention and replication. -->
<clusters>
  <cluster name="<MyDataCenter>" type="source">
    <validity start="$date" end="$date"/>

    <!--Currently delete is the only action available -->
    <retention limit="days($n)" action="delete">
    </cluster>

    <cluster name="$MyDataCenter-secondary" type="target">
    <validity start="2012-01-01T00:00Z" end="2099-12-31T00:00Z"/>
    <location type="data" path="/churn/weblogs/${YEAR}-${MONTH}-${DAY}-${HOUR} " />
    <retention limit="days(7)" action="delete"/>
    </cluster>
  </clusters>

<!-- Global location across clusters - HDFS paths or Hive tables -->
<locations>
  <location type="data" path="/weblogs/${YEAR}-${MONTH}-${DAY}-${HOUR} " /
>
</locations>

<!-- Required for HDFS. -->
<ACL owner="hdfs" group="users" permission="0755"/>
<schema location="/schema" provider="protobuf"/>

</feed>

```

2.5.2.5. Submit and Validate the Entities

- Submit your cluster entities. For example:

```
falcon entity -type cluster -submit -file <YourCluster>.xml
```

For each entity, you should see the following success message for submit:

```
falcon/default/Submit successful ($entity type) $yourEntityFile
```


- Submit your feed entity. For example:

```
falcon entity -type feed -submit -file <YourFeed>.xml
```

For each feed entity, you should see the following success message for submit:

```
falcon/default/Submit successful (feed) <YourFeed>
```

- Schedule your feed entity. For example:

```
falcon entity -type feed -name <YourFeed> -schedule
```

For each feed entity, you should see the following success message for schedule:

```
falcon/default/Schedule successful (feed) <YourFeed>
```

2.5.2.6. Confirm Results

To confirm your results, check your target cluster and review your Oozie jobs.

2.5.3. Viewing Alerts in Falcon

Falcon provides alerting for a variety of events to let you monitor the health of your data pipelines. All events are logged to the `metric.log` file, which is installed by default in your `$user/logs/` directory. You can view the events from the log or capture them using a custom interface.

Each event logged provides the following information:

- **Date:** UTC date of action.
- **Action:** Event name.
- **Dimensions:** List of name/value pairs of various attributes for a given action.
- **Status:** Result of the action. Can be FAILED or SUCCEEDED (when applicable).
- **Time-taken:** Time in nanoseconds for a given action to complete.

For example, a new process-definition alert would log the following information:

```
2012-05-04 12:23:34,026 {Action:submit, Dimensions:{entityType=process},
  Status: SUCCEEDED, Time-taken:97087000 ns}
```

Table 2.5. Available Falcon Event Alerts

| Entity Type | Action | Returns Success/Failure |
|-------------|---|-------------------------|
| Cluster | New cluster definitions submitted to Falcon | Yes |
| Cluster | Cluster update events | Yes |
| Cluster | Cluster remove events | Yes |
| Feed | New feed definition submitted to Falcon | Yes |
| Feed | Feed update events | Yes |
| Feed | Feed suspend events | Yes |
| Feed | Feed resume events | Yes |

| Entity Type | Action | Returns Success/Failure |
|-------------|---|-------------------------|
| Feed | Feed remove events | Yes |
| Feed | Feed instance deletion event | No |
| Feed | Feed instance deletion failure event (no retries) | No |
| Feed | Feed instance replication event | No |
| Feed | Feed instance replication failure event | No |
| Feed | Feed instance replication auto-retry event | No |
| Feed | Feed instance replication retry exhaust event | No |
| Feed | Feed instance late arrival event | No |
| Feed | Feed instance post cut-off arrival event | No |
| Process | New process definition posted to Falcon | Yes |
| Process | Process update events | Yes |
| Process | Process suspend events | Yes |
| Process | Process resume events | Yes |
| Process | Process remove events | Yes |
| Process | Process instance kill events | Yes |
| Process | Process instance re-run events | Yes |
| Process | Process instance generation events | No |
| Process | Process instance failure events | No |
| Process | Process instance auto-retry events | No |
| Process | Process instance retry exhaust events | No |
| Process | Process re-run due to late feed event | No |
| N/A | Transaction rollback failed event | No |

2.5.4. Late Data Handling

Late data handling in Falcon defines how long data can be delayed and how that late data is handled. For example, a late arrival cut-off of `hours(6)` in the feed entity means that data for the specified hour can delay as much as 6 hours later. The late data specification in the process entity defines how this late data is handled. The late data policy in the process entity defines how frequently Falcon checks for late data.

The supported policies for late data handling are:

- **backoff**: Take the maximum late cut-off and check every specified time.
- **exp-backoff (default)**: Recommended. Take the maximum cut-off date and check on an exponentially determined time.
- **final**: Take the maximum late cut-off and check once.

The policy, along with delay, defines the interval at which late data check is done. Late input specification for each input defines the workflow that should run when late data is detected for that input.

To handle late data, you need to modify the feed and process entities.

1. Specify the cut-off time in your feed entity.

For example, to set a cut-off of 4 hours:

```
<late-arrival cut-off="hours(4)"/>
```

- Specify a check for late data in all your process entities that reference that feed entity.

For example, to check each hour until the cut-off time with a specified policy of `backoff` and a delay of 1 hour:

```
<late-process policy="exp-backoff" delay="hours(1)">
  <late-input input="input" workflow-path="/apps/clickstream/late" />
</late-process>
```

2.5.5. Setting a Retention Policy

You can set retention policies on a per-cluster basis. You must specify the amount of time to retain data before deletion.

Falcon kicks off the retention policy on the basis of the time value you specify:

- **Less than 24 hours:** Falcon kicks off the retention policy every 6 hours.
- **More than 24 hours:** Falcon kicks off the retention policy every 24 hours.
- **When a feed is scheduled:** Falcon kicks off the retention policy immediately.



Note

When a feed is successfully scheduled, Falcon triggers the retention policy immediately regardless of the current timestamp or state of the cluster.

To set a retention policy, add the following lines to your feed entity for each cluster that the feed belongs to:

```
<clusters>
  <cluster name="corp" type="source">
    <validity start="2012-01-30T00:00Z" end="2013-03-31T23:59Z"
      timezone="UTC" />
    <retention limit="$unitOfTime($n)" action="delete" /> <!--
Retention policy. -->
  </cluster>
</clusters>
```

Where `limit` can be minutes, hours, days, or months and then a specified numeric value. Falcon then retains data spanning from the current moment back to the time specified in the attribute. Any data beyond the limit (past or future) is erased.

2.5.6. Setting a Retry Policy

You can set retry policies on a per-process basis. The policies determine how workflow failures are handled. Depending on the delay and number of attempts, the workflow is retried after specified intervals.

To set a retry policy, add the following lines to your process entity:

```
<retry policy=[retry policy] delay=[retry delay]attempts=[attempts]/>
<retry policy="$policy" delay="minutes($n)" attempts="$n"/>
```

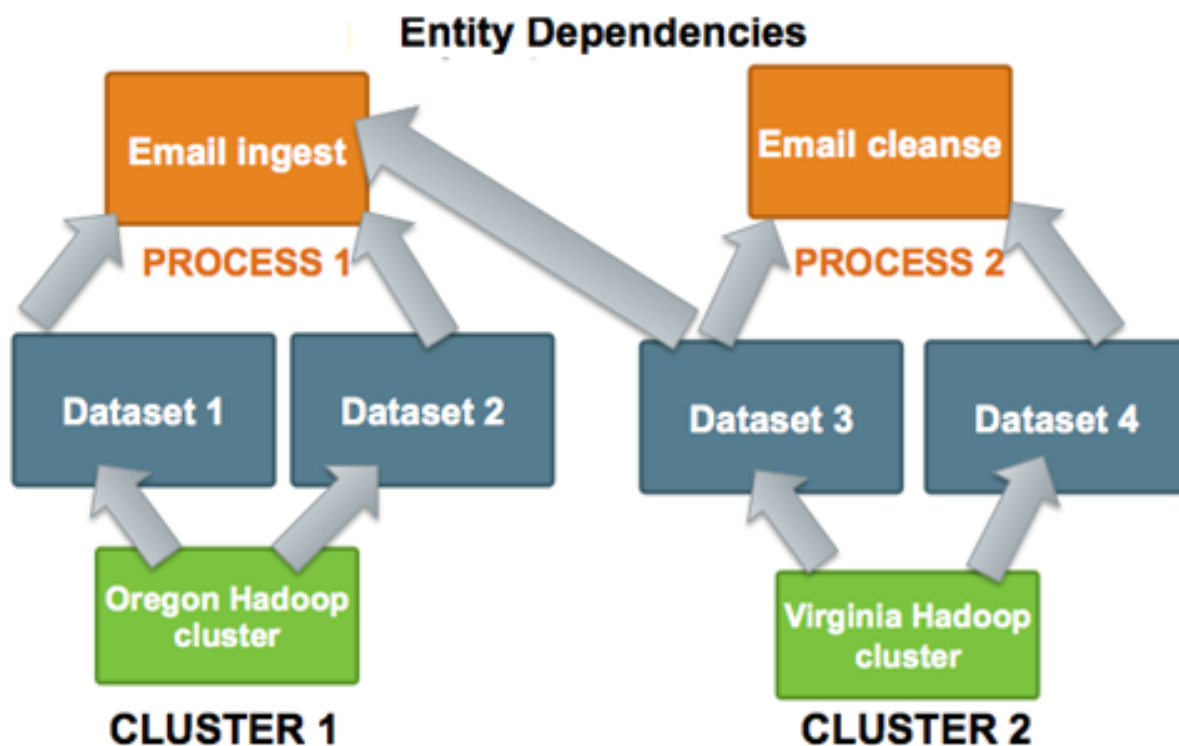
For example:

```
<process name = "[sample-process]">
...
  <retry policy="periodic" delay="minutes(10)" attempts="3"/>
...
</process>
```

In this example, the workflow is retried after 10 minutes, 20 minutes, and 30 minutes.

2.6. Understanding Dependencies in Falcon

Cross-entity dependencies in Falcon are important because a dependency cannot be removed until all the dependents are first removed. For example, if Falcon manages two clusters, one in Oregon and one in Virginia, and the Oregon cluster is going to be taken down, you must first resolve the Virginia cluster dependencies as one Dataset (Dataset 3) has a cross-entity dependency and depends on Email Ingest (Process 1).



To remove the Oregon cluster, you must resolve this dependency. Before you can remove the Oregon Hadoop cluster, you must remove not only Process 1, Datasets 1 and 2 but also modify the Dataset 3 entity to remove its dependence on Process 1.

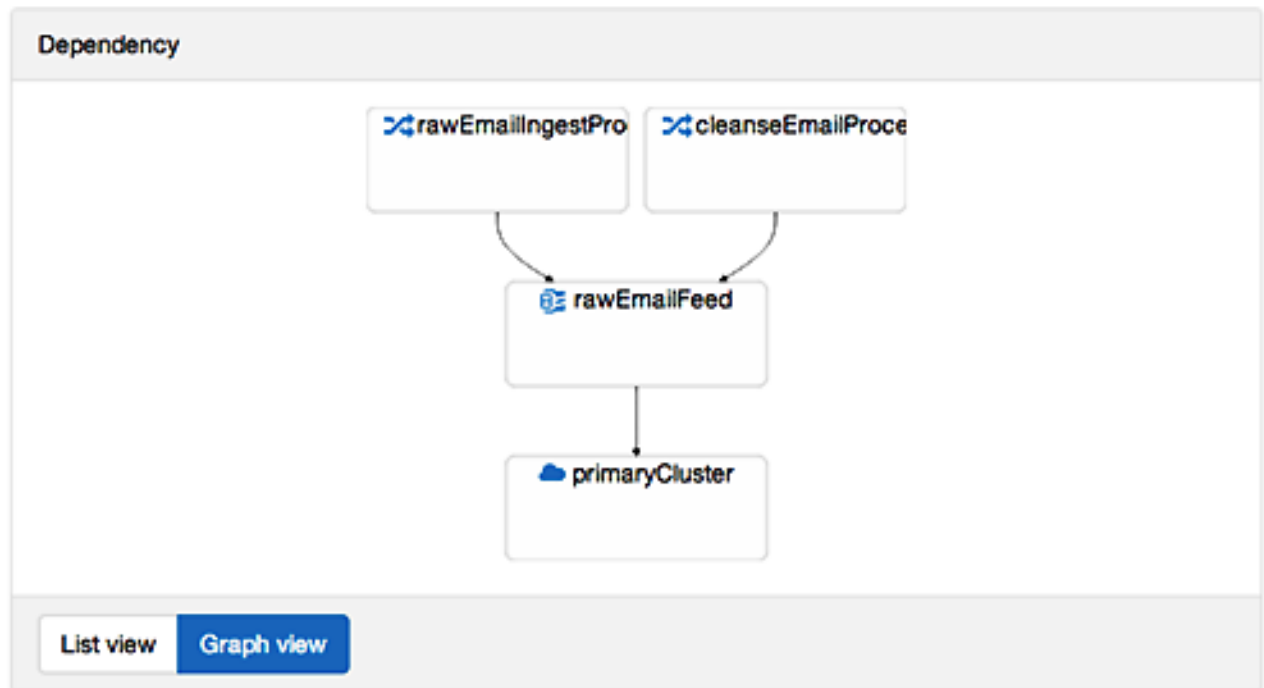
As Falcon manages more clusters, viewing these dependencies becomes more crucial.

2.7. Viewing Dependencies

The Falcon native UI provides dependency viewing for clusters, datasets, and processes that shows lineage in a list or graphical format:

- **List:** View the various dependencies and their types in a linear format.
- **Graph:** View the relationships between dependencies as a graph to determine requirements for removal.

Figure 2.9. Graph_view.png



3. Metadata Services Framework (Atlas)

Atlas is a low-level service that provides metadata services to the HDP platform. Veracity of the metadata is maintained by leveraging Apache Ranger to prevent unauthorized access at runtime, using both role-based (RBAC) and attribute-based (ABAC) access control.



Important

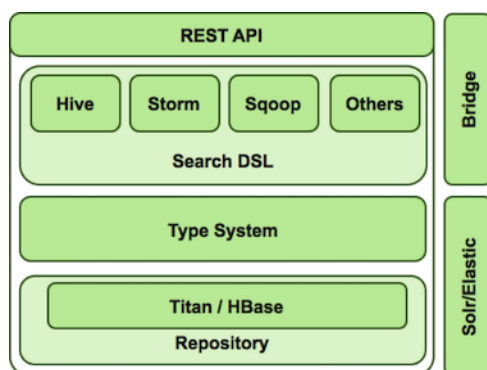
This chapter is intended as a quick start for the Atlas web UI. The content will be updated on a regular cadence over the next few months.

3.1. Understanding the HDP Metadata Services Framework

Hadoop presents data governance challenges because it is a platform comprised of autonomous projects that define their own future and share no common framework. For example, disparate tools, such as HCatalog, Ranger, and Falcon provide pieces of an overall data governance solution, but there is no comprehensive governance within the Hadoop stack. In addition, there is no means to integrate the Hadoop stack with external governance frameworks.

Atlas provides the means to centrally manage the data lifecycle in HDP, providing a repository that collects metadata for the platform that can be searched, tagged, and managed. A REST API is also available that can be used to integrate third-party governance tools with HDP. For information about the REST API, see [Appendix D](#) in this guide.

Figure 3.1. Atlas Architecture



- REST API handles all interaction with the metadata services.
- Existing HDP stack plug-in model leveraged by metadata services.
- Metadata search provided in two ways:
 - DSL (domain-specific language) search. A SQL-like query language.
 - Lucene-style full text search.

- Type system provides flexible modeling capability to model any business, data asset, or process, including inheritance.
- Titan/HBase Graph database that runs the type system.
- Bridge, a native connector to automatically fetch lineage and metadata. The Hive bridge connector ships with HDP 2.3. Additional components to follow.
- Solr/Elastic provide additional pluggable search capability that can be used without affecting the REST API or Atlas capabilities.

3.2. Using the Atlas Web UI to Search Metadata

Using the Atlas web UI is an efficient way to interact with HDP metadata services. Use Ambari to deploy your cluster, choose Atlas as one of your services, and the Atlas web UI is automatically installed. See the [Automated Install with Ambari guide](#).

Try the Atlas web UI by installing sample metadata with the `quick_start.py` Python script. This script installs metadata in your repository so you can test the search capabilities of the Atlas web UI:

To install sample metadata in your Atlas repository:

1. At a command prompt, log in to the host on your cluster where Atlas is installed.
2. Run the following command as the Atlas user:

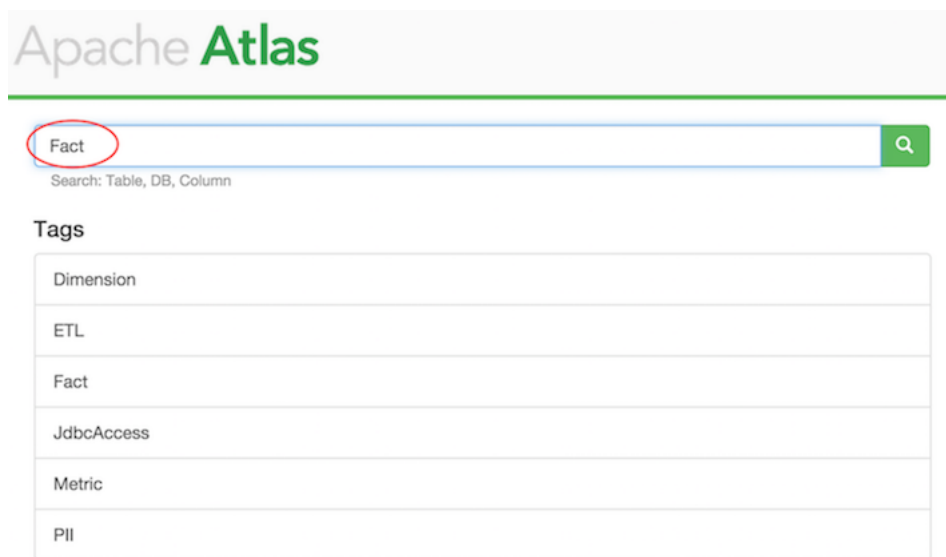
```
su atlas -c '/usr/hdp/current/atlas-server/bin/quick_start.py'
```

After you have installed the sample metadata, you can explore the Atlas web UI.

To search metadata with the Atlas web UI:

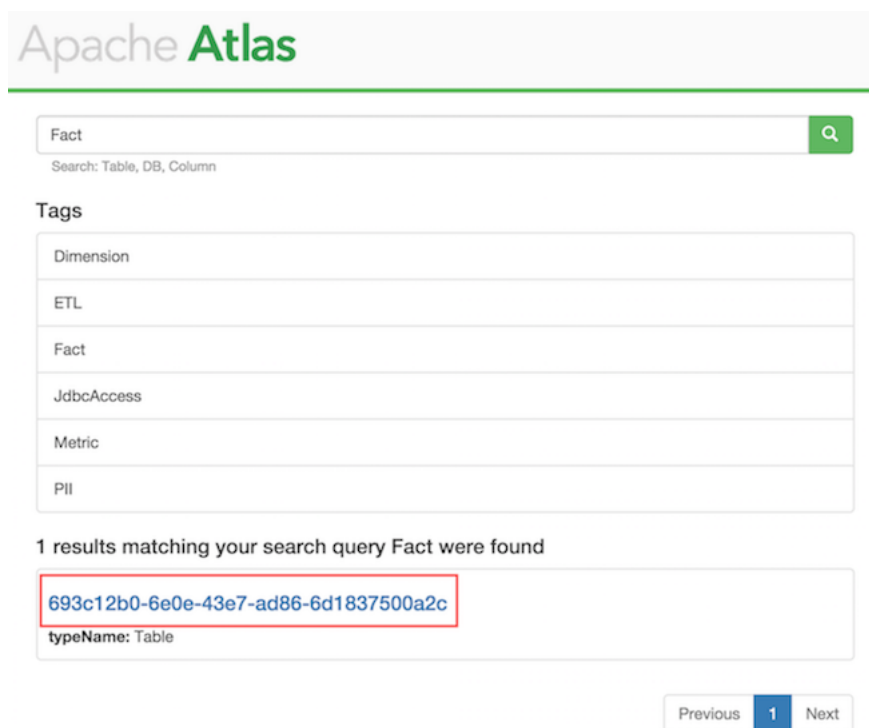
1. From the Ambari dashboard, click **Services > Atlas**. On the Summary tab, make sure that the Atlas Metadata Server is started.
2. Click **Quick Links > Atlas Dashboard** to launch the Atlas web UI.
3. Type a tag name in the search well and press enter:

Figure 3.2. Enter Tag to Search in Atlas Dashboard



4. The search returns all metadata types that are associated with the tag. Click the identifier link to view details about the metadata object:

Figure 3.3. Click the Tag Link to View Details



5. You can view four types of information for each metadata object by clicking each tab: **Details**, **Schema**, **Output**, and **Input**.

The Details tab shows information about the object, such as when it was created, the owner, when it was last accessed, and so on:

Figure 3.4. Details Tab

The screenshot shows the Apache Atlas interface. At the top, there is a header with the Apache Atlas logo and a "Back To Result" link. Below the header, the object name "Name: sales_fact" and description "Description: sales fact table" are displayed. There are four tabs: "Details" (selected), "Schema", "Output", and "Input". A table with two columns, "Key" and "Value", lists the following details:

| Key | Value |
|----------------|---|
| createTime | |
| db | id : d1fdd238-49ac-4ecf-be9f-d3f81451cb35 jsonClass : org.apache.atlas.typesystem.json.InstanceSerialization\$_Id typeName : DB |
| lastAccessTime | |
| owner | Joe |
| retention | |
| sd | jsonClass : org.apache.atlas.typesystem.json.InstanceSerialization\$_Reference typeName : StorageDesc |
| tableType | Managed |

6. Click the **Schema** tab to view the metadata object schema:

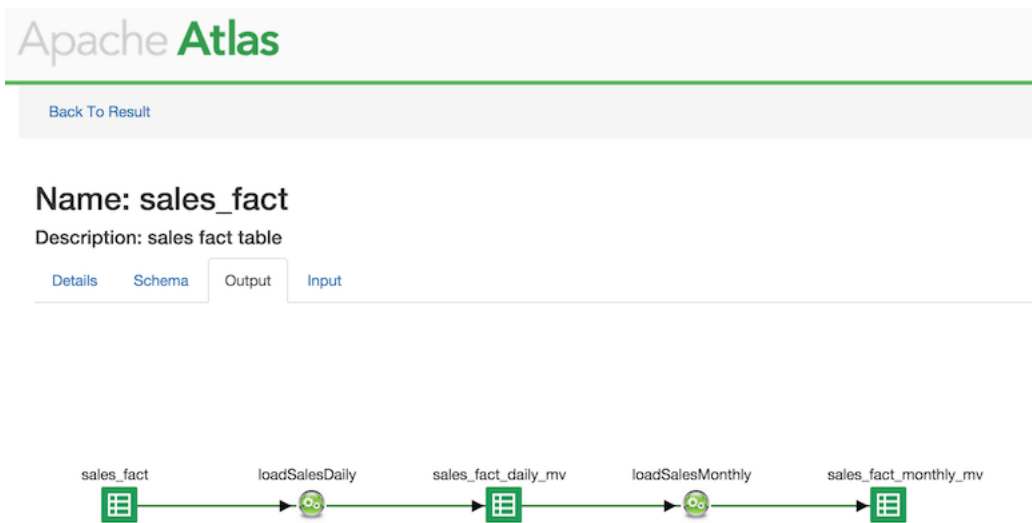
Figure 3.5. Schema Tab

The screenshot shows the Apache Atlas interface with the "Schema" tab selected. The object name "Name: sales_fact" and description "Description: sales fact table" are displayed. There are four tabs: "Details", "Schema" (selected), "Output", and "Input". A table with three columns, "Name", "Comment", and "DataType", lists the following schema details:

| Name | Comment | DataType |
|-------------|-------------|----------|
| time_id | time id | int |
| product_id | product id | int |
| customer_id | customer id | int |
| sales | product id | double |

7. Click the **Output** or the **Input** tabs to view lineage of the metadata:

Figure 3.6. Output Tab



In the above image, the lineage, or where the data comes from and where it goes when it is output is shown.

4. Reference (Falcon)

Valid entity schemas are required for a successful data pipeline.

To use the Falcon REST API, see [RESTful Resources](#) on the Apache web site.

4.1. Cluster

Always specify a cluster entity before determining the other elements in your data pipeline.

4.1.1. Valid Cluster Tag Attributes

The Cluster tag contains the following attributes to set:

```
<cluster colo="NJ-datacenter" description="test_cluster" name="prod-cluster">
```

Table 4.1. Cluster tag elements

| Example | Definition | Required? |
|--|---|-----------|
| <code>colo="\$unique_name"</code> | Unique name of the cluster, such as New Jersey Data Center. | Yes |
| <code>description="\$your_text"</code> | Description of the cluster, if desired. | No |
| <code>name="\$filename"</code> | Description of the cluster readiness. | Yes |

4.1.2. Cluster Interfaces

You can define the following interfaces in your cluster entity:

Table 4.2. Cluster Interfaces

| Type | Required | Interface Example Code |
|-----------|--|--|
| readonly | Yes | <code><interface type="readonly" endpoint="hftp://nn: 50070" version="2.4.0" /></code> |
| write | Yes | <code><interface type="write" endpoint="hdfs://nn:8020" version="2.4.0" /></code> |
| execute | Yes | <code><interface type="execute" endpoint ="rm:8050" version="0.20.2" /></code> |
| workflow | Yes | <code><interface type="workflow" endpoint="http://localhost:11000/oozie/" version="3.1" /></code> |
| registry | No, unless your feeds are Hive tables. | <code><interface type="registry" endpoint="thrift://localhost:9083" version="0.11.0" /></code> |
| messaging | Yes | <code><interface type="messaging" endpoint="tcp://localhost:61616?daemon=true" version="5.4.6" /></code> |

4.1.3. Cluster XSD Specification

The Cluster XSD specification is defined [here](#).

4.2. Feed Entity

The Feed XSD specification is defined [here](#).

4.3. Process Entity

The Process XSD specification is defined [here](#).

4.4. Using the CLI to Manage Entities and Instances

Falcon supports managing entities and instances with the CLI. Entities include all data pipeline components, such as clusters, feeds, and processes. Instances include only feeds and processes.

4.4.1. Managing Entities with the CLI

The following table provides information about CLI options you can use to manage entities:

Table 4.3. Entity Actions

| Option | Entities | Definition | CLI Usage |
|------------|------------------|---|--|
| definition | All | Current entity definition. Any documentation you have made within the entity will NOT be retained. | <code>\$FALCON_HOME/bin/falcon entity -type [cluster feed process] -name \$name -definition</code> |
| delete | All | Removes the entity from any scheduled activity and the Falcon configuration store. | <code>\$FALCON_HOME/bin/falcon entity -type [cluster feed process] -name \$name -delete</code> |
| dependency | Feeds, Processes | CLI dependency tracking. Returns all dependencies of the specified entity. | <code>\$FALCON_HOME/bin/falcon entity -type [cluster feed process] -name \$name -dependency</code> |
| list | All | Lists all scheduled and submitted entities in Falcon for a specified entity. | <code>\$FALCON_HOME/bin/falcon entity -type [cluster feed process] -list</code> |
| resume | Feeds, Processes | Restores a feed or process back to the active state, resuming the related Oozie bundle. | <code>\$FALCON_HOME/bin/falcon entity -type [feed process] -name \$name -resume</code> |
| schedule | Feeds, Processes | Schedules submitted feeds or processes. | <code>\$FALCON_HOME/bin/falcon entity -type [process feed] -name \$name -schedule</code> |
| status | All | Current status of the entity. | <code>\$FALCON_HOME/bin/falcon entity -type [cluster feed process] -name \$name -status</code> |
| submit | All | Creates a new cluster, feed, or process entity and validate it against the appropriate XSD. Check for dependent entities. | <code>\$FALCON_HOME/bin/falcon entity -submit -type cluster -file /cluster/definition.xml</code> |
| suspend | Feeds, Processes | Suspends any scheduled entity by triggering suspend on the Oozie bundle. | <code>\$FALCON_HOME/bin/falcon entity -type [feed process] -name \$name -suspend</code> |
| update | Feeds, Processes | Allows an already submitted or scheduled entity to be | <code>\$FALCON_HOME/bin/falcon entity -type [feed process]</code> |

| Option | Entities | Definition | CLI Usage |
|--------|----------|--|--|
| | | updated. Not allowed for cluster entities. | -name \$name -update [-effective \$effective time] |

4.4.2. Managing Instances with the CLI

The following table provides information about CLI options you can use to manage feed or process instances:

Table 4.4. Instance Actions

| Option | Definition | CLI Usage |
|----------|---|---|
| continue | Continue a process instance in a terminal state such as SUCCEEDED, KILLED, or FAILED. | <code>\$FALCON_HOME/bin/falcon instance -type \$feed/process -name \$name -continue -start "yyyy-MM-dd'T'HH:mm'Z" -end "yyyy-MM-dd'T'HH:mm'Z"</code> |
| help | Returns help on Falcon commands. | <code>\$FALCON_HOME/bin/falcon admin -help</code> |
| kill | Kills all the instances of the specified process whose nominal time is between the given start time and end time. | <code>\$FALCON_HOME/bin/falcon instance -type \$feed/process -name \$name -kill -start "yyyy-MM-dd'T'HH:mm'Z" -end "yyyy-MM-dd'T'HH:mm'Z"</code> |
| logs | Gets logs for instance actions. | <code>\$FALCON_HOME/bin/falcon instance -type \$feed/process -name \$name -logs -start "yyyy-MM-dd'T'HH:mm'Z" [-end "yyyy-MM-dd'T'HH:mm'Z"] [-runid \$runid]</code> |
| rerun | Rerun a process instance in a terminal state such as SUCCEEDED, KILLED, or FAILED. | <code>\$FALCON_HOME/bin/falcon instance -type \$feed/process -name \$name -rerun -start "yyyy-MM-dd'T'HH:mm'Z" -end "yyyy-MM-dd'T'HH:mm'Z" [-file \$properties file]</code> |
| resume | Resumes any instance in a suspended state. | <code>\$FALCON_HOME/bin/falcon instance -type \$feed/process -name \$name -resume -start "yyyy-MM-dd'T'HH:mm'Z" -end "yyyy-MM-dd'T'HH:mm'Z"</code> |
| running | Provides all running instances of the specified process. | <code>\$FALCON_HOME/bin/falcon instance -type \$feed/process -name \$name -running</code> |
| status | Gets the status of one or multiple instances of a process. | <code>\$FALCON_HOME/bin/falcon instance -type \$feed/process -name \$name -status -start "yyyy-MM-dd'T'HH:mm'Z" -end "yyyy-MM-dd'T'HH:mm'Z"</code> |
| summary | Summary of the status of feeds or processes within the time periods specified. | <code>\$FALCON_HOME/bin/falcon instance -type \$feed/process -name \$name -summary -start "yyyy-MM-dd'T'HH:mm'Z" -end "yyyy-MM-dd'T'HH:mm'Z"</code> |
| suspend | Suspends one or more instances for the given process. Pauses the parent workflow at the state. | <code>\$FALCON_HOME/bin/falcon instance -type \$feed/process -name \$name -suspend -start "yyyy-MM-dd'T'HH:mm'Z" -end "yyyy-MM-dd'T'HH:mm'Z"</code> |
| version | Returns current version of Falcon. | <code>\$FALCON_HOME/bin/falcon admin -version</code> |

5. Troubleshooting (Falcon)

The following information can help you troubleshoot issues with your Falcon server installation.

5.1. Falcon logs

The Falcon server logs are available in the logs directory under `$FALCON_HOME`.

To get logs for an instance of a feed or process:

```
$FALCON_HOME/bin/falcon instance -type $feed/process -name $name -logs -start "yyy-MM-dd'T'HH:mm'Z'" [-end "yyy-MM-dd'T'HH:mm'Z'"] [-runid $runid]
```

5.2. Falcon Server Failure

The Falcon server is stateless. All you need to do is restart Falcon for recovery, because a Falcon server failure does not affect currently scheduled feeds and processes.

5.3. Delegation Token Renewal Issues

Inconsistencies in rules for `hadoop.security.auth_to_local` can lead to issues with delegation token renewals.

If you are using secure clusters, verify that `hadoop.security.auth_to_local` in `core-site.xml` is consistent across all clusters.

5.4. Invalid Entity Schema

Invalid values in cluster, feeds (datasets), or processing schema can occur.

Review [Falcon entity specifications](#).

5.5. Incorrect Entity

Failure to specify the correct entity type to Falcon for any action results in a validation error.

For example, if you specify `-type feed` to `submit` `-type process`, you will see the following error:

```
[org.xml.sax.SAXParseException; lineNumber: 5; columnNumber: 68; cvc-elt.1.a: Cannot find the declaration of element 'process'.]
```

5.6. Bad Config Store Error

The configuration store directory must be owned by your "falcon" user.

5.7. Unable to set DataSet Entity

Ensure 'validity times' make sense.

- They must align between clusters, processes, and feeds.
- In a given pipeline Dates need to be ISO8601 format:

```
yyyy-MM-dd 'T' HH:mm 'Z'
```

5.8. Oozie Jobs

Always start with the Oozie bundle job, one bundle job per feed and process. Feeds have one coordinator job to set the retention policy and one coordinator for the replication policy.

6. Configuring High Availability (Falcon Server)

Currently, configuring high availability for the Falcon server is a manual process. When the primary Falcon server is down, the backup Falcon server must be manually started by the system administrator. Then the backup Falcon server picks up where the primary server stopped.

6.1. Configuring Properties and Setting Up Directory Structure for High Availability

Required Properties for Falcon Server High Availability:

The Falcon server stores its data in the **startup.properties** file that is located in the `<falcon_home>/conf` directory. Configure the start-up properties as follows for high availability:

- ***.config.store.uri**: This location should be a directory on HDFS.
- ***.retry.recorder.path**: This location should be an NFS-mounted directory that is owned by Falcon, and with permissions set to 755.
- ***.falcon.graph.storage.directory**: This location should also be an NFS-mounted directory that is owned by Falcon, and with permissions set to 755.
- **Falcon conf directory**: The default location of this directory is `<falcon_home>/conf`, which is symbolically linked to `/etc/falcon/conf`. This directory must point to an NFS-mounted directory to ensure that the changes made on the primary Falcon server are populated to the back-up server.

To set up an NFS-mounted directory:

The following instructions use 240.0.0.10 for the NFS server, 240.0.0.12 for the primary Falcon server, and 240.0.0.13 for the back-up Falcon server.

1. Logged in as **root** on the server that hosts the NFS mount directory:

- a. Install and start NFS with the following command:

```
yum install nfs-utils nfs-utils-lib
chkconfig nfs on
service rpcbind start
service nfs start
```

- b. Create a directory that holds the Falcon data:

```
mkdir -p /hadoop/falcon/data
```

- c. Add the following lines to the file `/etc/exports` to share the data directories:

```
/hadoop/falcon/data 240.0.0.12(rw,sync,no_root_squash,no_subtree_check)
/hadoop/falcon/data 240.0.0.13(rw,sync,no_root_squash,no_subtree_check)
```


- d. Export the shared data directories:

```
exportfs -a
```

2. Logged in as **root**, install the `nfs-utils` package and its library on each of the Falcon servers.

```
yum install nfs-utils nfs-utils-lib
```

3. After installing the NFS utilities packages, still logged in as **root**, create the NFS mount directory, and then mount the directories with the following commands:

```
mkdir -p /hadoop/falcon/data  
mount 240.0.0.10:/hadoop/falcon/data/hadoop/falcon/data
```

6.2. Preparing the Falcon Servers

To prepare the Falcon servers for high availability:

1. Logged in as **root** on each of the Falcon servers, make sure that the properties `*.retry.recorder.path` and `*.falcon.graph.storage.directory` point to a directory under the NFS-mounted directory. For example, the `/hadoop/falcon/data` directory as shown in the above example.
2. Logged in as the **falcon** user, start the primary Falcon server. Do not start the back-up Falcon server.

```
<falcon_home>/bin/falcon-start
```

6.3. Manually Failing Over the Falcon Servers

When the primary Falcon server fails, the failover to the back-up server is a manual process:

1. Logged in as the **falcon** user, make sure that the Falcon process is not running on the back-up server:

```
<falcon-home>/bin/falcon-stop
```

2. Logged in as **root**, update the `client.properties` files on all of the Falcon client nodes. Set the property `falcon.url` to the fully qualified domain name of the back-up server.

If Transport Layer Security (TLS) is disabled, use port 15000:

```
falcon.url=http://<back-up-server>:15000/ ### if TLS is disabled
```

If TLS is enabled, use port 15443:

```
falcon.url=https://<back-up-server>:15443/ ### if TLS is enabled
```

3. Logged in as the **falcon** user, start the back-up Falcon server:

```
<falcon-home>/bin/falcon-start
```

7. Metadata Store REST API Reference (Atlas)

This API supports a [Representational State Transfer \(REST\)](#) model for accessing a set of resources through a fixed set of operations. The following resources are accessible through the RESTful model:

- [AdminResource \[44\]](#)
- [EntityResource \[45\]](#)
- [HiveLineageResource \[47\]](#)
- [MetadataDiscoveryResource \[47\]](#)
- [RexsterGraphResource \[49\]](#)
- [TypesResource \[50\]](#)



Important

This appendix is intended as a quick start for the Atlas REST API. The content will be updated on a regular cadence over the next few months.

7.1. Data Model

All endpoints act on a common set of data. The data can be represented with difference media (i.e. "MIME") types, depending on the endpoint that consumes and/or produces the data. The data can be described by an [XML Schema](#), which definitively describes the XML representation of the data, but is also useful for describing the other formats of the data, such as [JSON](#).

This document describes the data using terms based on an XML Schema. Data can be grouped by namespace with a schema document describing the elements and types of the namespace. Types define the structure of the data and elements are instances of a type. For example, elements are usually produced by (or consumed by) a REST endpoint, and the structure of each element is described by its type.

7.2. AdminResource

Jersey Resource for administrative operations. The following resources are applicable:

- [???TITLE??? \[44\]](#)
- [???TITLE??? \[45\]](#)

/admin/stack

| | | |
|---------------|---|----------|
| GET | Fetches the thread stack dump for this application. | |
| Response Body | element: | (custom) |

| | |
|--|------------|
| media types: | text/plain |
| JSON represents the thread stack dump. | |

/admin/version

| | | |
|---------------|---|------------------|
| GET | Fetches the version for this application. | |
| Response Body | element: | (custom) |
| | media types: | application/json |
| | JSON represents the version. | |

7.3. EntityResource

Entity management operations. An entity is an instance of a type. Entities conform to the definition of the type that they they correspond to. The following resources are applicable:

- [???TITLE??? \[45\]](#)
- [???TITLE??? \[45\]](#)
- [???TITLE??? \[46\]](#)
- [???TITLE??? \[46\]](#)

/entities

| | | |
|---------------|--|------------------|
| POST | Submits an entity definition (instance) corresponding to a given type. | |
| Response Body | element: | (custom) |
| | media types: | application/json |
| | | |

| | | | | |
|---------------|--|----------------------------|-------------|----------------|
| GET | Fetches the list of entities for an entity type. | | | |
| Parameters | name | description | type | default |
| | type | The name of a unique type. | query | |
| Response Body | element: | (custom) | | |
| | media types: | application/json | | |
| | | | | |

/entities/{guid}

| | | | | |
|---------------|---|---|-------------|----------------|
| GET | Fetches the complete definition of the entity identified by the GUID. | | | |
| Parameters | name | description | type | default |
| | GUID | The globally unique identifier of the entity. | path | |
| Response Body | element: | (custom) | | |
| | media types: | application/json | | |
| | | | | |

| | | | | |
|---------------|-----------------------------------|---|------------------|----------------|
| PUT | Adds a property to the entity ID. | | | |
| Parameters | name | description | type | default |
| | GUID | The globally unique identifier of the entity. | path | |
| | property | The property that must be added. | query | |
| | value | The value of the property. | query | |
| Response Body | element: | | (custom) | |
| | media types: | | application/json | |
| | Response payload as JSON. | | | |

/entities/{guid}/traits

| | | | | |
|---------------|--|---|------------------|----------------|
| GET | Gets the list of trait names for the entity that is represented by the GUID. | | | |
| Parameters | name | description | type | default |
| | GUID | The globally unique identifier of the entity. | path | |
| | | | | |
| Response Body | element: | | (custom) | |
| | media types: | | application/json | |
| | A list of trait names for the entity that is identified by the GUID. | | | |

| | | | | |
|---------------|--|---|------------------|----------------|
| POST | Submits a new trait to an existing entity that is represented by the GUID. | | | |
| Parameters | name | description | type | default |
| | GUID | The globally unique identifier of the entity. | path | |
| | | | | |
| Response Body | element: | | (custom) | |
| | media types: | | application/json | |
| | | | | |

/entities/{guid}/traits/{traitName}

| | | | | |
|---------------|--|---|-------------|----------------|
| DELETE | Deletes a trait from the entity that is represented by the GUID. | | | |
| Parameters | name | description | type | default |
| | GUID | The globally unique identifier of the entity. | path | |
| | traitName | The name of the trait. | path | |
| Response Body | element: | | (custom) | |
| | | | | |

| | |
|--------------|------------------|
| media types: | application/json |
|--------------|------------------|

7.4. HiveLineageResource

Jersey Resource for the Hive table lineage. The following resources are applicable:

- [???TITLE??? \[47\]](#)
- [???TITLE??? \[47\]](#)
- [???TITLE??? \[47\]](#)

/lineage/hive/table/{tableName}/inputs/graph

| | | | | |
|---------------|---|------------------------|-------------|----------------|
| GET | Fetches the inputs graph for an entity. | | | |
| Parameters | name | description | type | default |
| | tableName | The name of the table. | path | |
| Response Body | element: | (custom) | | |
| | media types: | application/json | | |

/lineage/hive/table/{tableName}/outputs/graph

| | | | | |
|---------------|--|------------------------|-------------|----------------|
| GET | Fetches the outputs graph for an entity. | | | |
| Parameters | name | description | type | default |
| | tableName | The name of the table. | path | |
| Response Body | element: | (custom) | | |
| | media types: | application/json | | |

/lineage/hive/table/{tableName}/schema

| | | | | |
|---------------|-----------------------------------|------------------------|-------------|----------------|
| GET | Fetches the schema for the table. | | | |
| Parameters | name | description | type | default |
| | tableName | The name of the table. | path | |
| Response Body | element: | (custom) | | |
| | media types: | application/json | | |

7.5. MetadataDiscoveryResource

Jersey Resource for metadata operations. The following resources are applicable:

- [???TITLE??? \[48\]](#)
- [???TITLE??? \[48\]](#)
- [???TITLE??? \[48\]](#)

- [???TITLE??? \[48\]](#)

/discovery/search

| | | | | |
|---------------|---------------------------------------|---|------------------|----------------|
| GET | Search by using a query. | | | |
| Parameters | name | description | type | default |
| | query | The search query in raw Gremlin or DSL format that falls back to full text. | query | |
| Response Body | element: | | (custom) | |
| | media types: | | application/json | |
| | JSON represents the type and results. | | | |

/discovery/search/dsl

| | | | | |
|---------------|---------------------------------------|---------------------------------|------------------|----------------|
| GET | Search by using the query DSL format. | | | |
| Parameters | name | description | type | default |
| | query | The search query in DSL format. | query | |
| Response Body | element: | | (custom) | |
| | media types: | | application/json | |
| | JSON represents the type and results. | | | |

/discovery/search/fulltext

| | | | | |
|---------------|---------------------------------------|-----------------------------|------------------|----------------|
| GET | Search by using full text search. | | | |
| Parameters | name | description | type | default |
| | query | The full text search query. | query | |
| Response Body | element: | | (custom) | |
| | media types: | | application/json | |
| | JSON represents the type and results. | | | |

/discovery/search/gremlin

| | | | | |
|---------------|---|---|------------------|----------------|
| GET | Search by using the raw gremlin query format. | | | |
| Parameters | name | description | type | default |
| | query | The search query in raw gremlin format. | query | |
| Response Body | element: | | (custom) | |
| | media types: | | application/json | |
| | JSON represents the type and results. | | | |

7.6. RexsterGraphResource

Jersey Resource for lineage metadata operations. Implements most of the GET operations of the Rexster API without the indexes. This is a subset of the Rexster REST API, designed to provide read-only methods for accessing the back-end graph. See <https://github.com/tinkerpop/rexster/wiki/Basic-REST-API>.

The following resources are applicable:

- [???TITLE??? \[49\]](#)
- [???TITLE??? \[49\]](#)
- [???TITLE??? \[49\]](#)
- [???TITLE??? \[50\]](#)
- [???TITLE??? \[50\]](#)

/graph/edges/{id}

| | | | | |
|---------------|---|--------------------|------------------|----------------|
| GET | Fetches a single edge with a unique ID. For example, GET <code>http://host/metadata/lineage/edges/id graph.getEdge(id);</code> | | | |
| Parameters | name | description | type | default |
| | id | | path | |
| Response Body | element: | | (custom) | |
| | media types: | | application/json | |

/graph/vertices

| | | | | |
|---------------|--|--------------------|------------------|----------------|
| GET | Fetches a list of vertices that match a property key and value. For example, GET <code>http://host/metadata/lineage/vertices?key=&value= graph.getVertices(key,value);</code> | | | |
| Parameters | name | description | type | default |
| | key | | query | |
| | value | | query | |
| Response Body | element: | | (custom) | |
| | media types: | | application/json | |

/graph/vertices/{id}


| | | | | |
|------------|--|--------------------|-------------|----------------|
| GET | Fetches a single vertex with a unique ID. For example, GET <code>http://host/metadata/lineage/vertices/id graph.getVertex(id);</code> | | | |
| Parameters | name | description | type | default |
| | id | | path | |

| | | |
|---------------|--------------|------------------|
| Response Body | element: | (custom) |
| | media types: | application/json |

/graph/vertices/{id}/{direction}

| | | | | |
|---------------|--|--------------------|-------------|----------------|
| GET | Fetches a list of adjacent edges with a direction. For example, GET <code>http://host/metadata/lineage/vertices/id / directiongraph.getVertex(id).get{Direction}Edges(); direction: {(?!outE)(?!bothE)(?!inE)(?!out)(?!both)(?!in)(?!query).+}</code> | | | |
| Parameters | name | description | type | default |
| | id | | path | |
| | direction | | path | |
| Response Body | element: | (custom) | | |
| | media types: | application/json | | |

/graph/vertices/properties/{id}

| | | | | |
|---------------|---|--------------------|---|----------------|
| GET | Fetches properties for a single vertex with a unique ID. For example, GET <code>http://host/metadata/lineage/vertices/properties/id</code> | | | |
| |  | Note | This method is not part of the Rexster API. | |
| Parameters | name | description | type | default |
| | id | | path | |
| | relationships | | query | false |
| Response Body | element: | (custom) | | |
| | media types: | application/json | | |

7.7. TypesResource

This class provides a RESTful API for types. A type is the description of any representable item, for example, a Hive table. You can represent any meta model of any domain using these types. The following resources are applicable:

- [???TITLE??? \[50\]](#)
- [???TITLE??? \[51\]](#)

/types

| | | |
|---------------|--|----------|
| POST | Submits a type definition that corresponds to a type that represents a domain meta model. This method can represent objects like a Hive database, Hive table, and so on. | |
| Response Body | element: | (custom) |

| | | | |
|--|--|---|-------------|
| | media types: | application/json | |
| GET | Fetches a list of trait type names that are registered in the type system. | | |
| Parameters | name | description | type |
| | type | The name of the enumerator org.apache.atlas.typesystem.types.DataTypes.TypeCategory. Typically, this can be one of: all, TRAIT, CLASS, ENUM, STRUCT. | query |
| Response Body | default | all | |
| | element: | (custom) | |
| | media types: | application/json | |
| The entity names response payload represented as JSON. | | | |

/types/{typeName}

| | | | |
|---------------|--|------------------------------|-------------|
| GET | Fetches the complete definition of a unique type name. | | |
| Parameters | name | description | type |
| | typename | The unique name of the type. | path |
| Response Body | default | | |
| | element: | (custom) | |
| | media types: | application/json | |