Hortonworks Data Platform

Data Governance with Apache Falcon

(May 26, 2015)

docs.hortonworks.com

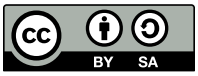
Hortonworks Data Platform : Data Governance with Apache Falcon

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 3.0 License.
http://creativecommons.org/licenses/by-sa/3.0/legalcode

Table of Contents

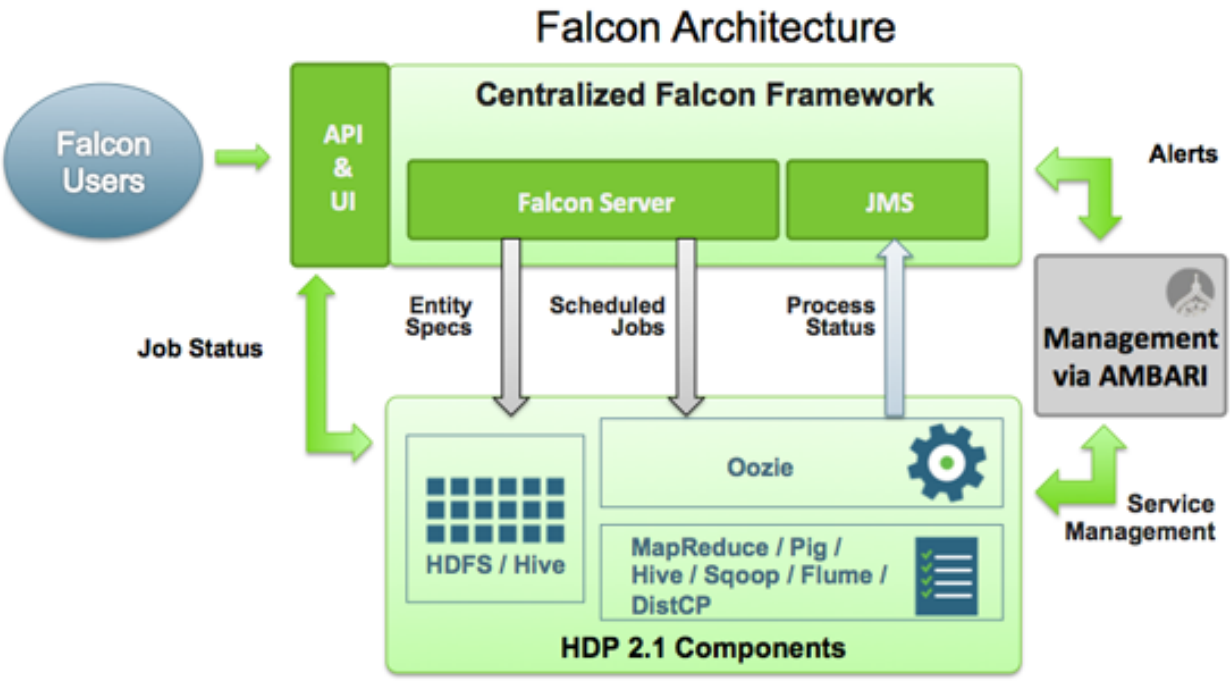
List of Figures

List of Tables

1. Data Governance with Apache Falcon

Apache Falcon provides a framework for automating data governance by defining data pipelines and providing dynamic changes to that pipeline through the Falcon interface. Falcon eliminates hard coding complex data sets and offers:

- **Data Replication:** Falcon can replicate HDFS and Hive datasets, trigger processes for retry, and handle late data arrival logic.
- **Data Lifecycle Management:** Falcon schedules eviction based on data retention policies you set.
- **Dataset Traceability:** Falcon exposes coarse-grained dependencies between clusters, datasets, and processes.



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

- Oozie jobs
- Pig scripts
- Hive scripts

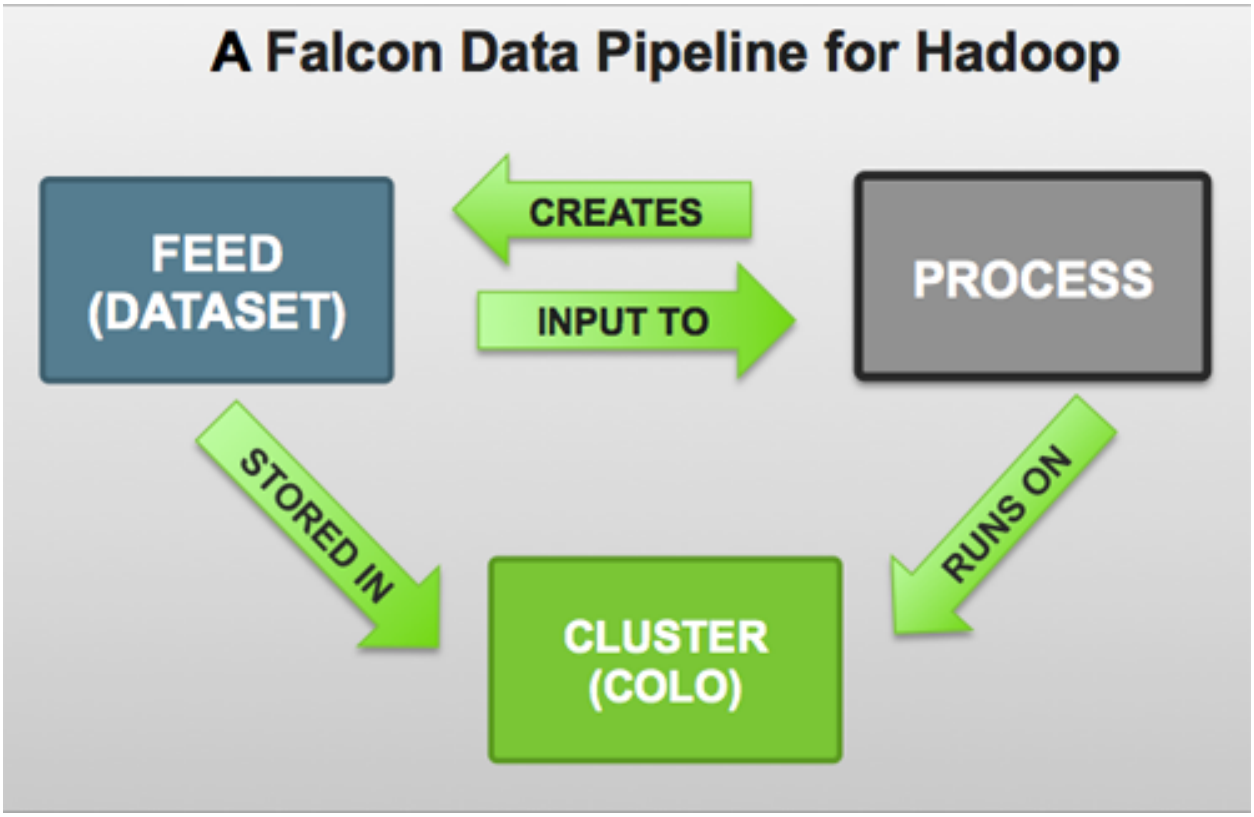
These jobs can then trigger alerts back to Falcon to give you the latest status on your data pipeline activities.

To learn more about Falcon, choose any of the following topics:

- Understanding Falcon

1.1. Understanding Falcon

Falcon manages dynamic data processing through the concept of pipelines. A pipeline combines data and processes across your cluster.



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.

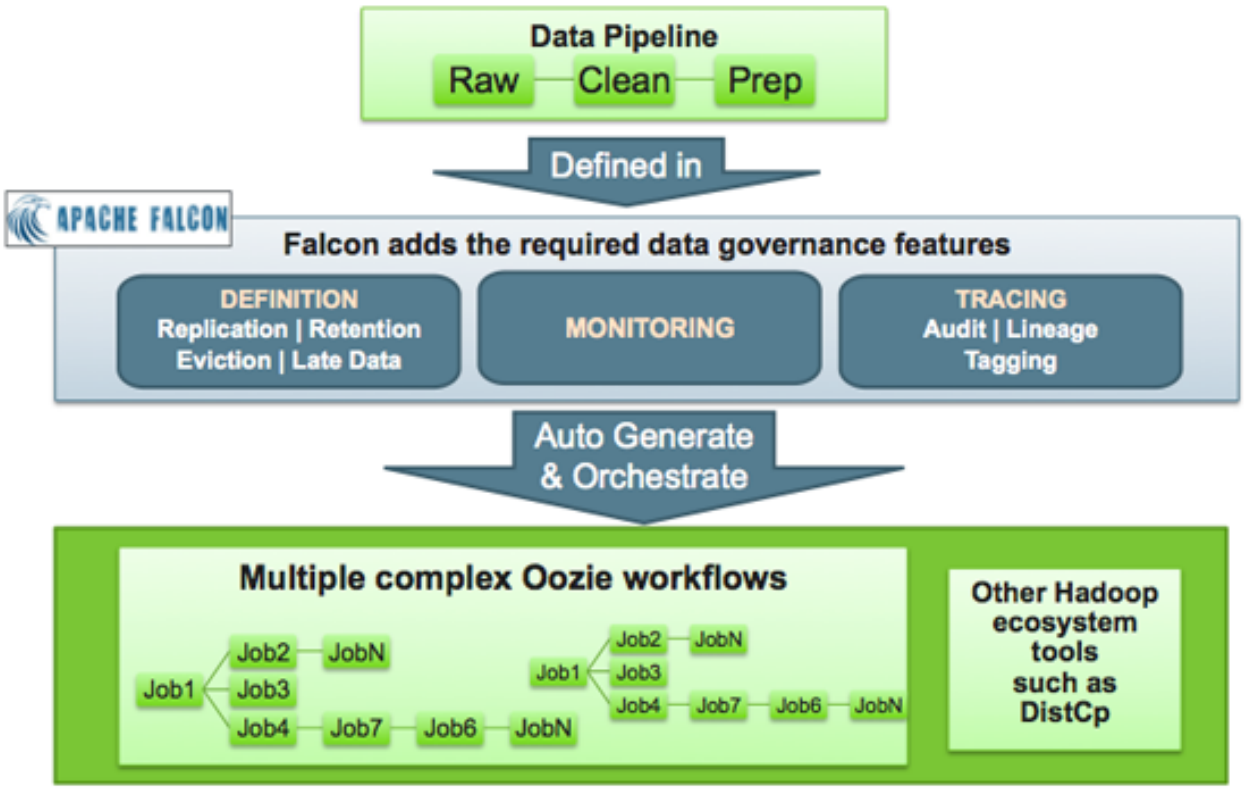
There are three types of Falcon entities:

- **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 in these areas.

These entities are defined and 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 1.1. falc2flow.png



1.1.1. Additonal Reading

For additional sources on the Falcon architecture and control flow:

- Falcon Community Documentation on Falcon Architecture
- Falcon Community Documentation on Falcon Control Flow

1.2. Defining Data Pipelines

To create a data pipeline you must:

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

```
<?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.

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}-
${HOUR} "/>
  <retention limit="days(7)" action="delete"/>
    </cluster>
  </clusters>
  <locations> <!-- Global location across clusters - HDFS paths or Hive
 tables -->
    <location type="data" path="/weblogs/${YEAR}-${MONTH}-${DAY}-${HOUR} "/>
  </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. Specifiy 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 Piplelines.

1.3. Deploying Data Pipelines

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

To deploy the data pipeline:

1. Submit your entities to Falcon. Be sure to specifiy 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.

1.4. Data Replication

Falcon can replicate data across multiple clusters using distcp, and do it according to the fequency 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. And, for every instance that a feed is replicated Falcon sends a JMS message on the success or failure of the replication instance.

For example, in this feed two clusters are replicating data to a backup cluster:

```
<clusters>
   <cluster name=Cluster1" type="source" partition="${cluster.name}" delay=
"days(2)">
       <validity start="2011-11-01T00:00Z" end="2021-11-30T00:00Z"/>
   </cluster>
   <cluster name="Cluster2" type="source" partition="COUNTRY/${cluster.
name}">
       <validity start="2011-11-01T00:00Z" end="2021-11-30T00:00Z"/>
   </cluster>
   <cluster name="Backup" type="target">
       <validity start="2011-11-01T00:00Z" end="2011-11-31T00:00Z"/>
   </cluster>
</clusters>
```

Note

We recommend that the data path be as granular as the frequency of the feed. For example, if you are specifying the feed frequency in hours, provide a data path that is/${YEAR}/${MONTH}/${DAY}/${HOUR}.

In this example, two coordinators are scheduled to pull data in to the target, Backup, one coordinator pulls the data from a partition in Cluster1 and the other coordinator pulls from a partition in Cluster2. A replication delay of 2 days has been set for Cluster1, which means that it will run every 30 days with an offset of 2 days. This means that the feed instance that is scheduled for replication November 30 is elligible December 2nd.

If you are using Falcon for Data Replication, explore the following topics:

- Falcon Community Documentation on Language Expression
- Section 1.4.1, “distCP Throttle” [8]
- Replacing JMS with ActiveMQ

1.4.1. distCP Throttle

Falcon uses distcp (distributed copy) for data replication. If you need to optimize bandwidth between data centers, you can throttle bandwidth during Falcon data replication as needed and limit the number of maps used during replication.

To throttle distcp:

1. If you already have Falcon running on your clusters, suspend your current active feeds and processes:

```
$FALCON_HOME/bin/falcon entity -type $feedName -name $name -suspend

$FALCON_HOME/bin/falcon entity -type $processName -name $name -suspend
```

2. Edit your feed entity or entities. Add the following lines:

```
<properties>
   <property name="maxMaps" value="$integerValue" />
   <property name="maxBandwidth" value="$MB/svalue" />
</properties>
```

Note

Specify the maximum number of mappers for Falcon to use in `maxMaps`. Specify the bandwidth in MB for each mapper in `mapBandwidth`.

3. Sumbit your updated feed entity.

```
$FALCON_HOME/bin/falcon entity -submit -type feed -file ~$feedFileName
```

4. Resume your processes.

```
$FALCON_HOME/bin/falcon entity -type $processName -name $name -resume

$FALCON_HOME/bin/falcon entity -type $feedName -name $name -resume
```

1.4.2. Replacing JMS with ActiveMQ

Falcon embeds ActiveMQ in its distribution.

To use ActiveMQ to broker JMS messaging:

1. If you already have Falcon running on your clusters, suspend your current active feeds and processes:

```
$FALCON_HOME/bin/falcon entity -type $feedName -name $name -suspend

$FALCON_HOME/bin/falcon entity -type $processName -name $name -suspend
```

2. Edit your cluster entity or entities. Add the following line:

```
<properties>
   <property name="brokerImplClass" value="org.apache.activemq.
ActiveMQConnectionFactory" />
</properties>
```

3. Submit your updated cluster entity.

```
$FALCON_HOME/bin/falcon entity -submit -type cluster -file ~$clusterFileName
```

4. Resume your processes.

```
$FALCON_HOME/bin/falcon entity -type $processName -name $name -resume
```

```
$FALCON_HOME/bin/falcon entity -type $feedName -name $name -resume
```

In ActiveMQ, you should now see Falcon publishing messages to:

- **FALCON.my-process topic:** For each execution of the process.
- **FALCON.ENTITY.TOPIC topic:** For each change on the feeds.

1.5. 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 1.1. Available Falcon Event Alerts

Entity Type	Action	Returns Success/Falilure
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
Feed	Feed remove events	Yes
Feed	Feed instance deletion event	No
Feed	Feed instance deletion failure event (no retries)	No

Entity Type	Action	Returns Success/Falilure
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

Note

For more information on alerting, see the Falcon Communit Documentation on Alerting.

1.6. 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)”/>
```

2. 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>
```

1.7. 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/state of the cluster.

To set a retention policy, add the following lines to your cluster entity as part of the <cluster> definition:

```
<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.

Note

Delete is the only supported action for HDP 2.1.

1.8. Setting a Retry Policy

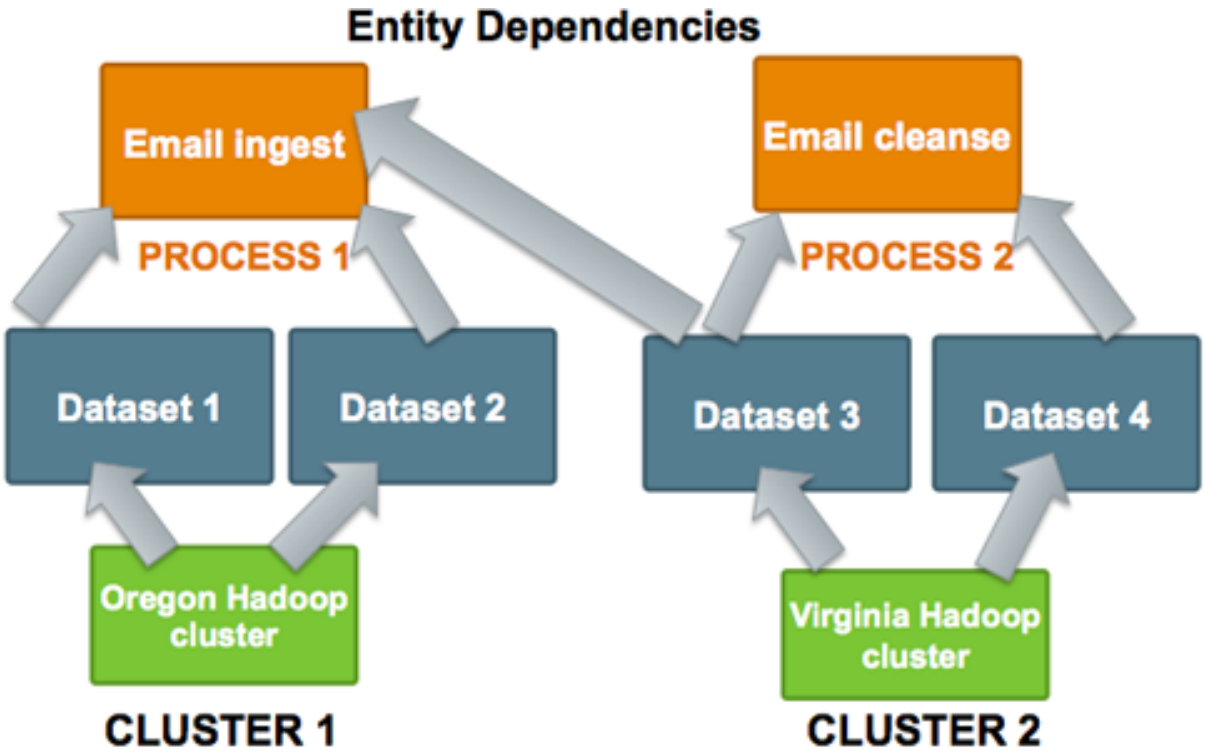
You can set retry policies on a per-process basis.

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"/>
```

1.9. Understanding Dependencies in Falcon

Cross-entity depedencies 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 dependenices becomes more crucial. For information on viewing dependencies in Falcon, see Viewing Dependencies. For more information on Cross-Enity validations, see Falcon Community Documentation on Cross-Entity Validations.

1.10. Viewing Dependencies

The Falcon native UI provides dependency viewing for datasets and processes as the following possible views:

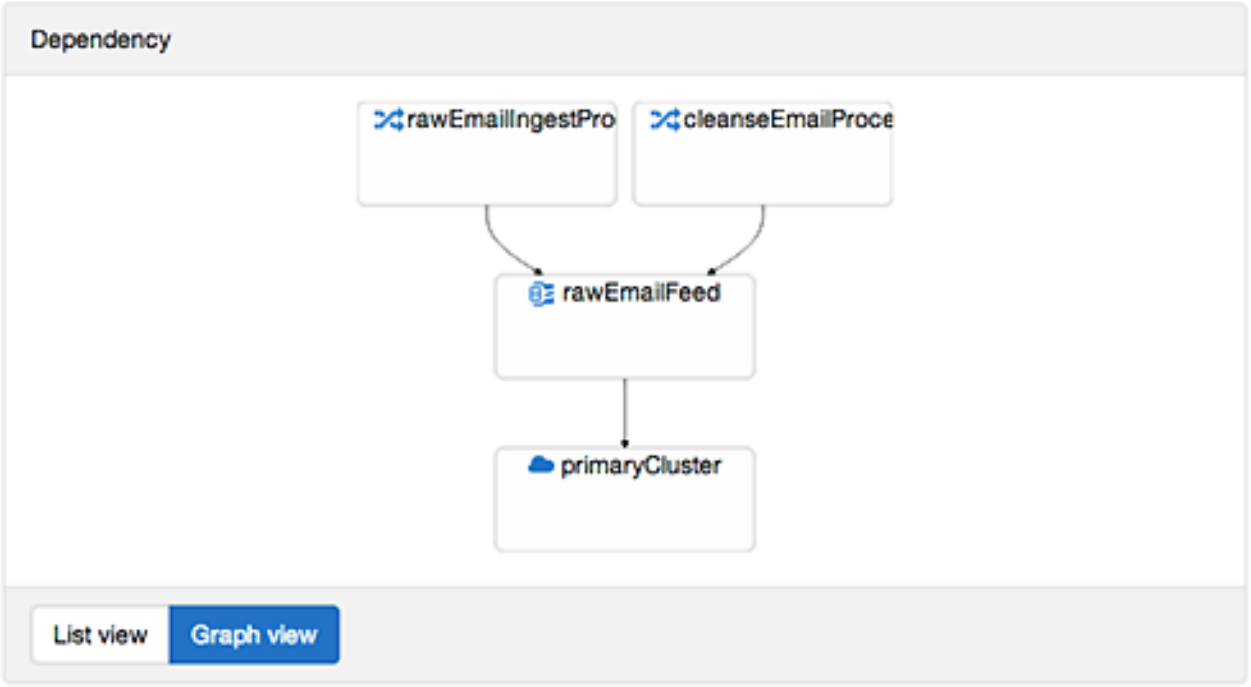
- **List:** View the various dependencies and their types in a linear format.

- **Graph:** View the relationships between dependencies as a graph to determine requirements for removal.

Note

Dependencies are view-only in HDP 2.1. You must remove processes and datasets using the Falcon CLI.

Figure 1.2. Graph_view.png



2. Troubleshooting Falcon

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

2.1. 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
 "yyyy-MM-dd'T'HH:mm'Z'" [-end "yyyy-MM-dd'T'HH:mm'Z'"] [-runid $runid]
```

2.2. 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.

2.3. 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.

2.4. Invalid Entity Schema

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

Review Falcon schemas.

2.5. 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 sumbit -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'.]
```

2.6. Bad Config Store Error

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

2.7. 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’
```

2.8. 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.

3. Appendix A: Falcon Reference

Valid entity schemas are required for a successful data pipline.

3.1. Cluster

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

3.1.1. Valid Cluster Tag Attributes

The Cluster tag contains the following attributes to set:

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

Table 3.1. Cluster tag elements

Example	Definition	Required?
`colo="$unique_name"`	Unique name of the cluster, such as New Jersey Data Center.	Yes
`description="$your_text"`	Description of the cluster, if desired.	No
`name="$filename"`	Description of the cluster readiness.	Yes

3.1.2. Cluster Interfaces

You can define the following interfaces in your cluster entity:

Table 3.2. Cluster Interfaces

Type	Required	Interface Example Code
readonly	Yes	`<interface type="readonly" endpoint="hftp://nn: 50070" version="2.4.0" />`
write	Yes	`<interface type="write" endpoint="hdfs://nn:8020" version="2.4.0" />`
execute	Yes	`<interface type="execute" endpoint ="rm:8050"  version="0.20.2" />`
workflow	Yes	`<interface type="workflow" endpoint="http://localhost:11000/oozie/" version="3.1" />`
registry	No, unless your feeds are Hive tables.	`<interface type="registry" endpoint="thrift://localhost:9083" version="0.11.0" />`
messaging	Yes	`<interface type="messaging" endpoint="tcp://localhost:61616?daemon=true" version="5.4`

3.1.3. Cluster XSD Specification

The Cluster XSD specification is defined here.

3.2. Feed Entity

The Feed XSD specification is defined here.

3.3. Process Entity

The Process XSD specification is defined here.

3.4. Managing Entities Using the CLI

Falcon supports the following options for Entity Management:

Table 3.3. Entity Actions

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

3.5. Managing Falcon using the CLI

Table 3.4. Entity Actions

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