# Hortonworks Data Platform

## Data Movement and Integration

(Nov 30, 2016)

docs.hortonworks.com

# Hortonworks Data Platform: Data Movement and Integration

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.

# Table of Contents

# List of Figures

# List of Tables

# List of Examples

# 1. HDP Data Movement and Integration

Enterprises that adopt a modern data architecture 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, data movement and lineage must be managed centrally and comprehensively to provide security, data governance, and administration teams the necessary oversight to ensure compliance with corporate standards for data management. Hortonworks offers the HDP *Data Movement and Integration Suite (DMI Suite)* to provide that comprehensive management for data movement in to and out of Hadoop.

Use cases for data movement and integration (DMI) include the following:

- Definition and scheduling of data manipulation jobs, including:
  - Data transfer
  - Data replication
  - Mirroring
  - Snapshots
  - Disaster recovery
  - Data processing
- Monitoring and administration of data manipulation jobs
- Root cause analysis of failed jobs
- Job restart, rerun, suspend and termination
- Workflow design and management
- Ad-hoc bulk data transfer and transformation
- Collection, aggregation, and movement of large amounts of streaming data

## 1.1. Intended Audience

Administrators, operators, and DevOps team members who are responsible for the overall health and performance of the HDP ecosystem use DMI for management, monitoring, and administration. Management, monitoring, and administration are performed using the Falcon Dashboard.

| | |
|---|---|
| Database Administrators | Responsible for establishing recurring transfers of data between RDBMS systems and Hadoop. |
| Business Analysts or other business users | Need the ability to perform ad-hoc ETL and analytics with a combination of Hadoop-based and RDBMS-based data. |

| | |
|---|---|
| DevOps | Responsible for:<br>• Maximizing the predictability, efficiency, security, and maintainability of operational processes.<br>Use the DMI Suite to create an abstraction of sources, data sets and target systems along with jobs and processes for importing, exporting, disaster recovery and processing.<br>• Designing workflows of various types of actions including Java, Apache Hive, Apache Pig, Apache Spark, , Hadoop distributed file system (HDFS) operations, along with SSH, shell, and email.<br>• The collection, aggregation, and movement of streaming data, such as log events. . |

## 1.2. Data Movement Components

The HDP Data Movement and Integration Suite (DMI Suite) leverages the following Apache projects:

| | |
|---|---|
| **Apache Falcon** | Management and abstraction layer to simplify and manage data movement in Hadoop |
| **Apache Oozie** | Enterprise workflow operations |
| **Apache Sqoop** | Bulk data transfers between Hadoop and RDBMS systems |
| **Apache Flume** | Distributed, reliable service for collecting, aggregating, and moving large amounts of streaming data |

In addition, the DMI Suite integrates other Apache APIs to simplify creation of complex processes, validate user input, and provide integrated management and monitoring.

Beyond the underlying components, the DMI Suite provides powerful user interfaces that simplify and streamline creation and management of complex processes.

# 2. Data Management and Falcon Overview

In the Hortonworks Data Platform (HDP), the overall management of the data lifecycle in the platform is achieved through Apache Falcon. Falcon provides powerful, end-to-end data management for the Hadoop platform. It is also a core part of data governance.

Core capabilities include the following:

- Data lifecycle management
- Management and monitoring of all data movement on HDP
- Data transfer in to and out of HDP
- Support for complex data processing topologies using data pipelines
- Integration with Apache Atlas for data lineage and provenance management

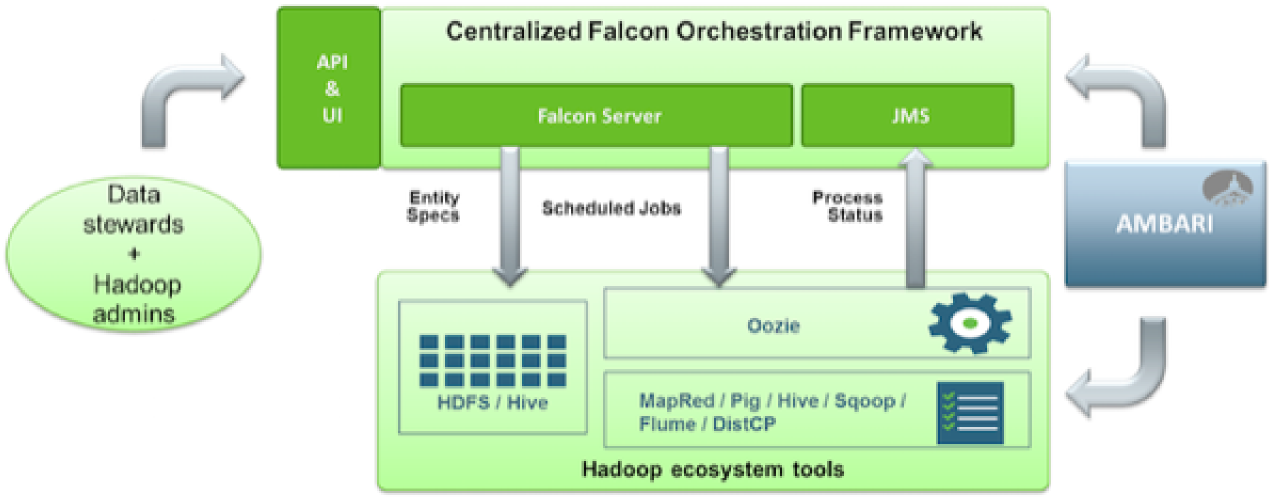
  For more information about data governance and Apache Atlas, see the Data Governance Guide.

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 leverages other HDP components, such as Apache Pig, Apache Hadoop Distributed File System (HDFS), Apache Sqoop, Apache Hive, Apache Spark, and Apache Oozie.

Falcon enables simplified management by providing a framework to define and manage backup, replication, and data transfer. Falcon addresses the following data management and movement requirements:

- 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 databases and tables on clusters using server-side extensions 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 2.1. Falcon Architecture


Centralized Falcon Orchestration Framework
API & UI
Falcon Server
JMS
Entity Specs
Scheduled Jobs
Process Status
Data stewards + Hadoop admins
AMBARI
HDFS / Hive
Oozie
MapRed / Pig / Hive / Sqoop / Flume / DistCP
Hadoop ecosystem tools

## 2.1. Falcon Access

There are three methods to access Falcon:

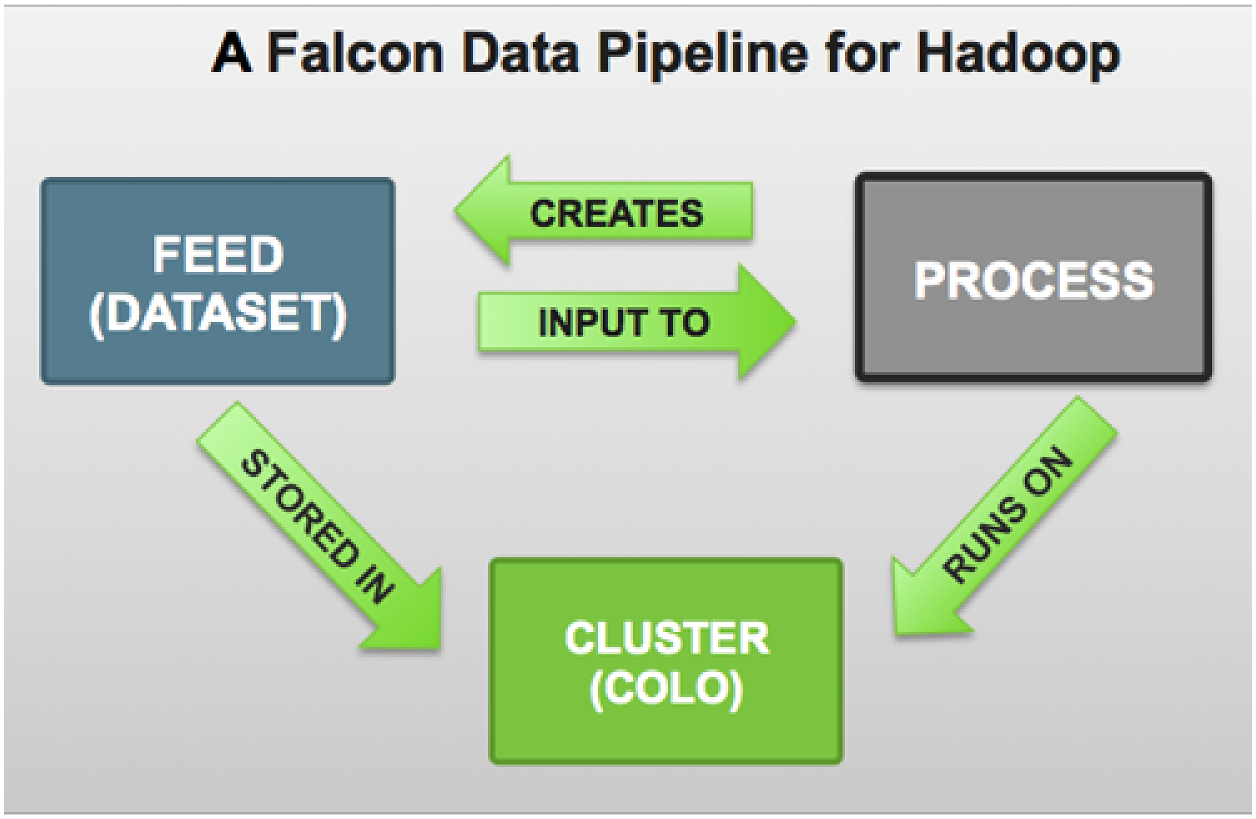
- Using the Falcon graphical user interface, accessible from Apachi Ambari either stand-alone or as an Ambari View
- Using the Falcon CLI
- Using the Falcon APIs, as a service

## 2.2. Understanding Entity Relationships

Falcon entities are configured to form a data pipeline. A pipeline consists of a dataset and the processing that acts on the dataset across your HDFS cluster. You can configure data pipelines for data replication and mirroring.

When creating a data pipeline, you must define cluster storage locations and interfaces, dataset feeds, and the processing logic to be applied to the datasets.

Figure 2.2. 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.

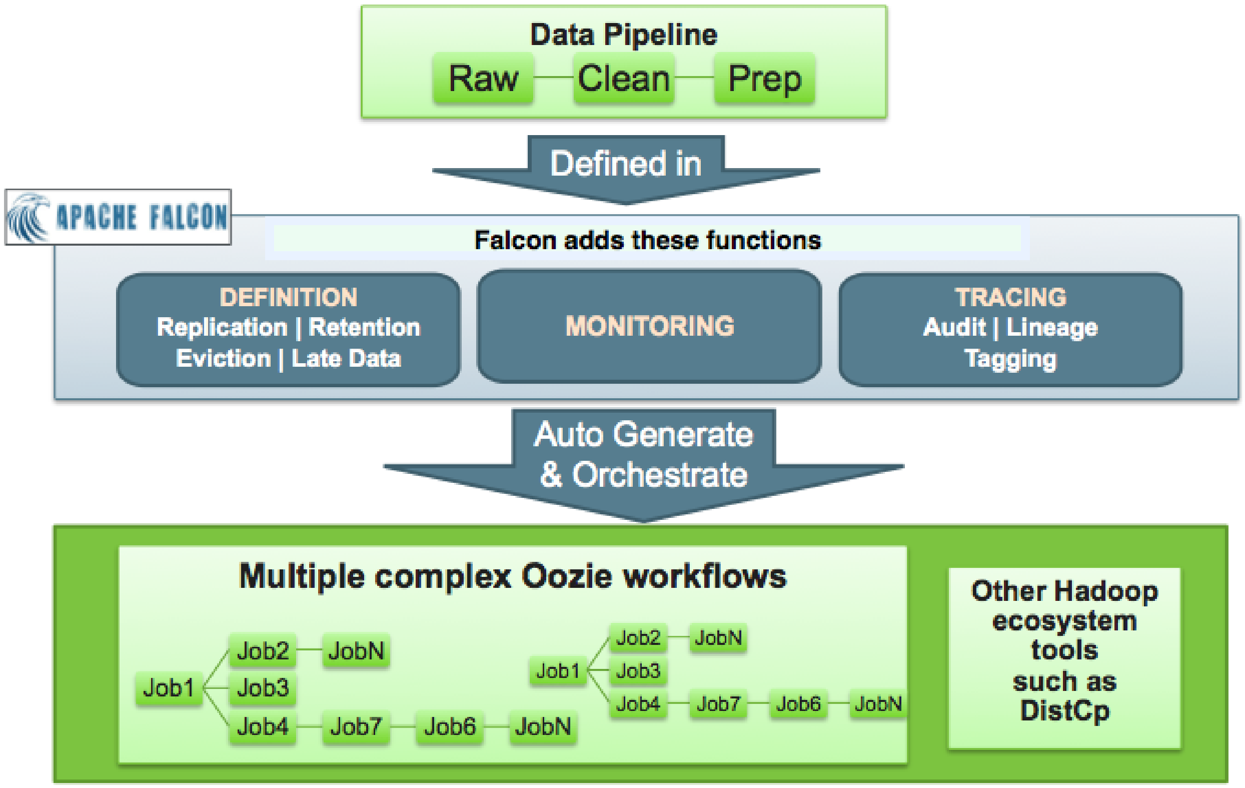
There are three types of entities:

| | |
|---|---|
| Cluster | Defines the cluster, including its interfaces, where data and processes are stored. |
| Feed | Defines the datasets to be cleaned and processed. |
| Process | Defines how the process (such as a Pig or Hive jobs) works with the dataset on top of a cluster. A process consumes feeds, invokes processing logic (including late data handling), and produces further feeds. It also defines the configuration of the Oozie workflow and defines when and how often the workflow should run. |

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, retry, 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, Spark jobs, Pig scripts, and Hive queries. For example, Falcon data management policies become Oozie coordinator jobs:

**Figure 2.3. Data Pipeline Flow**



## Note

The auditing and lineage functionality is part of data governance and is not addressed in this documentation. See the Data Governance Guide for information about auditing and lineage.

# 3. Configuring for High Availability

To enable high availability you must manually configure the Falcon server. When the primary Falcon server is down, the system administrator must manually start the backup Falcon server. Then the backup Falcon server continues where the primary server stopped.

**Note**

Due to changes in Hive, for the Oozie HCAT URI (which is used for Hive table feeds) Falcon supports URIs with only one metastore. This applies even if you have multiple metastores configured.

## 3.1. Configuring Properties and Setting Up Directory Structure for High Availability

The Falcon server stores its data in the `startup.properties` file that is located in the `<falcon_home>/conf` directory. You should configure the startup properties as follows to achieve 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 has permissions set to 755 (rwx/r-x/r-x).

***.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 value must point to an NFS-mounted directory to ensure that the changes made on the primary Falcon server are populated to the backup server.

**To set up an NFS-mounted directory, follow these steps**:

The example input uses 240.0.0.10 for the NFS server, 240.0.0.12 for the primary Falcon server, and 240.0.0.13 for the backup Falcon server.

1. On the server that hosts the NFS mount directory, log in as *root* and perform the following steps.

   a. Install and start NFS:

   ```
   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. Create the NFS mount directory, and then mount the data directories:

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

## 3.2. Preparing the Falcon Servers

**To prepare the Falcon servers for high availability:**

1. Logged in as *root* on each of the Falcon servers, ensure that the properties ***.retry.recorder.path** and ***.falcon.graph.storage.directory** point to a directory within the NFS-mounted directory: for example, the */hadoop/falcon/data* directory shown in the previous example.

2. Logged in as the *falcon* user, start the primary Falcon server, *not* the backup server:

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

## 3.3. Manually Failing Over the Falcon Servers

When the primary Falcon server fails, you must manually fail over to the backup server:

1. Logged in as the *falcon* user, ensure that the Falcon process is not running on the backup server:

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

2. Logged in as *root*, update the `client.properties` files on all of the Falcon client nodes and set the property **falcon.url** to the fully qualified domain name of the backup 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 backup Falcon server:

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

# 4. Creating Falcon Entity Definitions

Before you can use Falcon to process, manage, or replicate data or configure for disaster recovery, you must create directories in HDFS for Falcon to use, then create the necessary Falcon entity definitions. You can create the entity definitions from the web UI or from the CLI.

There are three types of entities.

| | |
|---|---|
| Cluster | Defines the cluster, including its interfaces, where data and processes are stored.<br><br>Cluster entities are required for all jobs. |
| Feed | Defines the datasets to be cleaned and processed.<br><br>Feed entities are required for replication and retention. |
| Process | Defines how the process (such as a Pig or Hive job) works with the dataset on top of a cluster. A process consumes feeds, invokes processing logic (including late data handling), and produces further feeds. It also defines the configuration of the Oozie workflow and defines when and how often the workflow should run.<br><br>Process entities are required for all jobs. However, you can run a process without a feed entity if data replication or retention are not required. |

## 4.1. Replication Between HDP Versions

Before you define the resources to be used in replication jobs, ensure that the versions of HDP you are running are compatible for replication.

### Table 4.1. Supported HDP Versions for Replication

| HDP Release | Can Replicate To | Cannot Replicate To |
|---|---|---|
| 2.0 | 2.0, 2.1 | 2.2 or later releases |
| 2.1 | 2.1 | 2.2 or later releases |
| 2.2 or later | 2.2 and any subsequent release | 2.1 or earlier releases |

## 4.2. Running Falcon in a Secure Environment

Before accessing Falcon in a secure environment, your cluster must be properly configured following the instructions in the *HDP Security* guide.

After completing an Ambari secure installation with Falcon, Kerberos is enabled. You can access the Falcon web UI from the Ambari Views menu without any additional configuration.

When creating Falcon entities, most of the required Kerberos information is automatically populated in the web UI.

If secure HDP is installed from the CLI, you need to ensure that Falcon was also configured for Kerberos.

If you need to access the standalone Falcon web UI, you might need to add the VM name and domain to your browser, following the browser instructions for Kerberos single sign-on (SSO) configuration.

## 4.3. Creating HDFS Directories for Falcon

You must create the following directories in HDFS prior to submitting a cluster entity to Falcon:

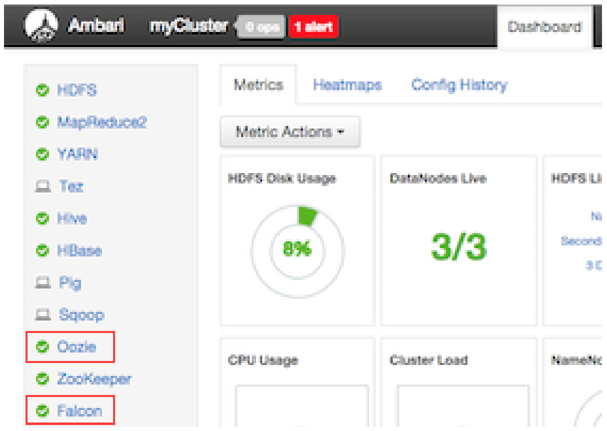
| | |
|---|---|
| `/staging` | Falcon stores artifacts of processes and feeds, such as the feed and process definitions and job logs, in `/staging`. When you schedule an entity, the workflow and library you specify in the entity are copied to the staging path. |
| `/working` | Falcon copies the `.jar` files needed to run processes and feeds to the working directory. |
| `/temp` | Falcon uses the Temp directory to do intermediate processing of entities in HDFS. |

**Steps**

1. In the Ambari UI, ensure that the Falcon and Oozie services are running.

   Confirm that the Falcon and Oozie services have green check marks adjacent to them on the Ambari dashboard:

   **Figure 4.1. Ambari Dashboard Falcon and Oozie Service Indicators**



2. Log in to the HDFS server as the *falcon* user.

```
sudo su falcon
```

3. Create the directory structure on HDFS for the staging, temporary, and working folders in which the cluster entity stores the dataset.

   These folders must be owned by the *falcon* user.

```
hadoop fs -mkdir -p /apps/falcon/primary_Cluster
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
```

**Tip**

You can also create the directories from the Ambari UI File View. You must be logged in as user *falcon*.

4. Set permissions on the cluster staging directory to 777 (read/write/execute for owner/group/others) to allow multiple user access.

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

Only Oozie job definitions are written to the staging directory, so setting permissions to 777 does not create any vulnerability. Keep permissions on the working directory set to 755 (rwx/r-w/r-w).

**Tip**

You can also change permissions on directories from the Ambari UI Administration menu. You must be logged in as user *falcon* with administrator access.

5. Launch the Falcon web UI from 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**.

**Important**

To access the Falcon UI on a secured cluster, use `kinit` before accessing the Kerberos-protected web server and configure your browsers for SPNEGO access. For more information, see Configure Browser for SPNEGO.

## 4.4. Defining Entities Using the Falcon Web UI

The Apache Falcon web UI simplifies the process of defining and deploying 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. You can access the Falcon web UI from Apache Ambari.

See Ambari Automated Install Guide or Non-Ambari Cluster Installation Guide for Falcon installation instructions.

**Prerequisites**

Before you can begin working with Falcon, ensure that you have the following components installed on your cluster:

- Hortonworks Data Platform (HDP)
- Apache Falcon
- Apache Oozie

**Steps**

To create a data pipeline, you must perform the following actions:

1. Section 4.3, “Creating HDFS Directories for Falcon” [10]
2. Section 4.4.1, “Creating a Cluster Entity Definition Using the Web UI” [12]
3. Section 4.4.2, “Creating a Feed Entity Definition Using the Web UI” [14]
4. .Section 4.4.3, “Creating a Process Entity Definition Using the Web UI” [17]

### 4.4.1. Creating a Cluster Entity Definition Using the Web UI

You must 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 Apache cluster entity XSD.

**Steps**

1. At the top of the Falcon web UI page, click **Create** > **Cluster**.
2. On the New Cluster page, specify the values for the following properties:

**Table 4.2. Cluster Entity General Properties**

| Property | Description |
|---|---|
| **Cluster Name** | Name of the cluster resource entity. Not necessarily the actual cluster name. Cannot be changed after the entity is created. The naming criteria are as follows:<br>• Must be unique to the data center<br>• Must start with a letter<br>• Is case sensitive<br>• Can contain a maximum of 40 characters<br>• Can include numbers<br>• Can use a dash (-) but no other special characters<br>• Cannot contain spaces<br>• Must match the name of the directory containing the staging, tmp, and working subdirectories. |
| **Data Center or Colo Name** and **Description** | The data center to which the cluster belongs. Used to identify the correct cluster for the entity. |
| **Tags** | Metadata tagging to classify the cluster. |

### Table 4.3. Cluster Entity Interface Properties

| Property | Description |
|---|---|
| **File System Read Endpoint Address** | A read-only interface that specifies the endpoint and protocol to connect to a data source. Required for DistCp (distributed copy). This would be used for replication, such as importing from a data source into HDFS. Located in Ambari at `HDFS>Configs>Advanced>NameNode>NameNode host`. The URI might be hftp://<hostname>:50070 or hdfs://<hostname>:8020. |
| **File System Default Address** | A write interface that specifies the endpoint and protocol to write to the datasource. Falcon uses this interface to export data from HDFS to the data source. Located in Ambari at `HDFS>Config>Advanced>Advanced core-site> fs.defaultFS`. |
| **YARN Resource Manager Address** | Required to write jobs to MapReduce. Located in Ambari at `Yarn>Config>Advanced>Resource Manager>ResourceManager host`. |
| **Workflow Address** | Required to submit workflow jobs. Located in Ambari at `Oozie>Config>Advanced>Oozie Server>Oozie Server host`. |
| **Message Broker Address** | Required to send alerts. Located in Ambari at `Falcon>Config>Advanced>broker.url`. |
| **Metadata Catalog Registry** | Optional. Use to register or deregister partitions in the Hive Metastore and to fetch events about partition availability. Located in Ambari at `Hive>Config>Advanced>General>hive.metastore.uris`.<br><br>In a secure environment, you must also add a property name and value for `hive.metastore.kerberos.principal` found under `Advanced hive-site` in Ambari. Example: key=hive.metastore.kerberos.principal, value=hive/_HOST@EXAMPLE.COM |
| **Spark** | Optional. Select if you want to run a Spark job. You can deploy Spark on a YARN cluster or client, locally, or on a custom site. You only need to add an endpoint for the Custom option. |

### Table 4.4. Cluster Entity Properties & Location Properties

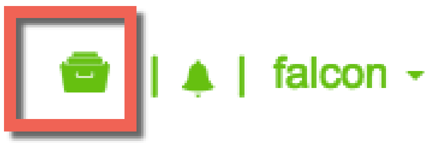
| Property | Description |
|---|---|
| **Properties** | Specify a name and value for each property. In a secure environment, the Name and Value fields are automatically populated with content from the `Custom falcon-startup.properties` section in Ambari. Example: key name=`dfs.namenode.kerberos.principal`, value=`nn/_HOST@EXAMPLE.COM` |
| **Location** | Specify the HDFS locations for the staging, temp, and working directories. The directory that contains the staging and working subdirectories must have the same name as the entity's Cluster Name. For more information, see Creating HDFS Directories for Falcon. |

### Table 4.5. Cluster Entity Advanced Properties

| Property | Description |
|---|---|
| **Access Control List** | Specify the owner, group, and access permissions for the cluster. Default permissions are 755 (rwx/r-x/r-x). |

3. Click **Next** to view a summary of your cluster entity definition.
4. (Optional) Click **Preview XML** to review or edit the entity definition in XML.
5. After verifying the cluster entity definition, click **Save**.

   The entity is automatically submitted for verification, but it is not scheduled to run.
6. Verify that you successfully created the cluster entity by clicking the Clusters icon at the top right of the window.



If the entity name appears in the cluster list, the entity was created.

7. Schedule the entity.

   a. In the cluster list, click the checkbox next to an entity name with status of `Submitted`.

   b. Click Schedule.

      After a few seconds a success message displays.

**Tip**

You can modify an existing cluster entity from the clusters list by clicking Edit. However, the Falcon service must be in safemode before you can edit the cluster entity.

## 4.4.2. Creating a Feed Entity Definition Using the Web UI

The feed entity defines the datasets that are cleaned and processed in your data pipeline. For more information, see the Apache feed entity XSD.

The Feed entities have policies attached to them that need to be explicitly scheduled by Falcon. Falcon takes the retention, replication, feed frequency, and delays and creates Oozie Coordinator jobs to automate all of these actions for you. To process data, you must define two feed entities: One for data input and one for data output.

**Steps**

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

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

**Table 4.6. General Feed Properties**

| Property | Description |
|---|---|
| **Feed Name** and **Description** | The dataset feed name must be unique on the cluster. This name is referenced by processes as input or output feed. Cannot be changed after the entity is created. |
| **Tags** | Metadata tagging to classify the dataset. For example, you can set the **key** to "externalTarget" and the corresponding **value** to "Marketing", tagging this feed for marketing. |
| **Feed Groups** | A group is a logical association of feeds. Feeds can belong to multiple groups. A group is said to be available if all the feeds belonging to a group are available. The frequency value must be the same for all feeds in a group. |
| **Type** | Options are Hive, HDFS, RDBMS Import, and RDBMS Export. You can also choose whether to enable replication for the feed type. |

## Table 4.7. Hive Source and Target Feed Properties

| Property | Description |
|---|---|
| **Cluster** | The target cluster entity. |
| **Table URI** | Metadata catalog specification for the Hive table, in the metadata format `catalog:`*`$database_name`*`:`*`$table#(partition-key=partition-value);`*+. Example: catalog:logs-db:clicks#ds=${YEAR}-${MONTH}-${DAY} |
| **Start and End Times** | Date is entered in mm/dd/yyyy format. Time is set to a 12-hour scale. After the feed job starts, it continues at the given Frequency setting until reaching the end time. |
| **Retention** | Period to retain instances and number of instances to retain. |
| **Frequency** | How often the feed is generated. Valid frequency types are minutes, hours, days, and months. |

## Table 4.8. HDFS Source and Target Feed Properties

| Property | Description |
|---|---|
| **Cluster** | The target cluster entity. |
| **Data Path** | Path of the HDFS data to be exported. Example: `/weblogs/${YEAR}-${MONTH}-${DAY}-${HOUR}` |
| **Statistics Path** | (Optional) Path to directory in which to store statistics. |
| **Start and End Times** | Date is entered in mm/dd/yyyy format. Time is set to a 12-hour scale. After the feed job starts, it continues at the given Frequency setting until reaching the end time. |
| **Retention** | Period to retain instances and number of instances to retain. |
| **Frequency** | How often the feed is generated. Valid frequency types are minutes, hours, days, and months. |

## Table 4.9. RDBMS Import Source and Target Feed Properties

| Property | Description |
|---|---|
| **Datasource** | Connection information for a remote data source, such as an RDBMS. |
| **Table** | Name of the RDBMS table to be imported or exported. |
| **Extract Type** | Options are Full or Incremental |
| **Merge Type** | Options are Snapshot or Append |
| **Columns** | Select to have all columns replicated, or enter the specific columns to include or exclude. |
| **Location** | Options are HDFS or Hive |
| **Cluster** | The target cluster entity. |
| **Data Path** | (HDFS only) Path of the HDFS data to be exported. Example: `/weblogs/${YEAR}-${MONTH}-${DAY}-${HOUR}` |
| **Statistics Path** | (HDFS only–optional) Path to directory in which to store statistics. |
| **Table URI** | (Hive only) Metadata catalog specification for the Hive table, in the metadata format `catalog:`*`$database_name`*`:`*`$table#(partition-key=partition-value);`*+. Example: catalog:logs-db:clicks#ds=${YEAR}-${MONTH}-${DAY} |
| **Start and End Times** | Date is entered in mm/dd/yyyy format. Time is set to a 12-hour scale. After the feed job starts, it continues at the given Frequency setting until reaching the end time. |
| **Retention** | Period to retain instances and number of instances to retain. |
| **Frequency** | How often the feed is generated. Valid frequency types are minutes, hours, days, and months. |

### Table 4.10. RDBMS Export Source and Target Feed Properties

| Property | Description |
|---|---|
| **Location** | Options are HDFS or Hive |
| **Cluster** | The target cluster entity. |
| **Data Path** | (HDFS only) Path of the HDFS data to be exported. Example: `/weblogs/${YEAR}-${MONTH}-${DAY}-${HOUR}` |
| **Statistics Path** | (HDFS only–optional) Path to directory in which to store statistics. |
| **Table URI** | (Hive only) Metadata catalog specification for the Hive table, in the metadata format `catalog:`*`$database_name`*`:`*`$table#(partition-key=partition-value);`*`+`. Example: catalog:logs-db:clicks#ds=${YEAR}-${MONTH}-${DAY} |
| **Start and End Times** | Date is entered in mm/dd/yyyy format. Time is set to a 12-hour scale. After the feed job starts, it continues at the given Frequency setting until reaching the end time. |
| **Retention** | Period to retain instances and number of instances to retain. |
| **Datasource** | Connection information for a remote data source, such as an RDBMS. |
| **Table** | Name of the RDBMS table to be imported or exported. |
| **Load Method** | Options are Update Only or Allow Insert. Indicates whether rows can be inserted into the RDBMS or only updates to existing rows are allowed. |
| **Columns** | Select to have all columns replicated, or enter the specific columns to include or exclude. |
| **Frequency** | How often the feed is generated. Valid frequency types are minutes, hours, days, and months. |

### Table 4.11. Advanced Feed Properties

| Property | Description |
|---|---|
| **Queue Name** | Hadoop job queue. |
| **Job Priority** | Hadoop job priority. |
| **Late Arrival Cutoff** | The timeframe within which a dataset feed can arrive late and still be processed. The feed instance is rerun within the time specified to capture any late arriving data. |
| **Availability Flag** | Specifies the name of a file that when present in a feed's data directory, determines that the feed is available. |
| **Max Map Jobs** | The maximum number of maps used during replication. This setting impacts performance and throttling. |
| **Max Bandwidth** | The bandwidth in MB/s used by each mapper during replication. This setting impacts performance and throttling. |
| **Parallel** | The concurrent replication instances that can run at any given time. This setting impacts performance and throttling. |
| **Access Control List** | Specify the HDFS access permissions. The owner must be the owner of the Falcon `staging` and `working` directories. |
| **Feed Schema** | Specify the schema location and provider. This is required for HDFS. If none, enter **/none**. |

3. Click **Next** to view a summary of your feed entity definition.

4. (Optional) Click **Preview XML** to review or edit the entity definition in XML.

5. After verifying the feed entity definition, click **Save**.

   The entity is automatically submitted for verification, but it is not scheduled to run.

6. Verify that you successfully created the feed entity.

   a. Type the feed entity name in the Falcon web UI **Search** field and press **Enter**.

b. If the feed entity name appears in the search results, it was successfully created.

   For more information about the search function, see "Locating and Managing Entities" in Using Advanced Falcon Features.

7. Schedule the entity.

   a. In the search results, click the checkbox next to an entity name with status of `Submitted`.

   b. Click Schedule.

      After a few seconds a success message displays.

## 4.4.3. Creating a Process Entity Definition Using the Web UI

The process entity consumes the feeds, invokes processing logic, and can produce additional feeds. For more information, see the Apache process entity XSD.

You can use one script to work on many different Falcon-defined processes. This helps reduce code complexity and makes it easier to manage multiple workflows. Although you can call complicated Oozie workflows with one Falcon job, we recommend that you split up the complex Oozie workflows into modular steps. This helps Falcon better manage retention of any intermediary datasets. It also allows for reuse of Processes and Feeds.

When submitting a `workflow.xml` file as part of a process entity definition, ensure the following:

- The `workflow.xml` path is absolute.
- The file is in a location that is readable by the user submitting the entity and by service user "falcon".

**Steps:**

1. At the top of the Falcon web UI page, click **Create** > **Process**.
2. On the New Process page, specify the values for the following properties:

### Table 4.12. General Process Properties

| **Property** | **Description** |
|---|---|
| **Process Name** | Name of the process entity. Must be unique for the cluster. Cannot be changed after the entity is created. |
| **Tags** | Optional business labels, such as "Finance." There is no input validation on this field, so there can be duplicates. |

### Table 4.13. Process Detail and Engine Properties

| **Property** | **Description** |
|---|---|
| **Engine** | Specify which workflow engine to use. Choices are Spark, Oozie, Pig, and Hive. Some properties for the entity might change, depending on the engine selected. |
| **Workflow Name** | The name of the workflow to be used. The naming criteria are as follows: • Must be unique to the data center |

| Property | Description |
|---|---|
| | • Must start with a letter<br>• Is case sensitive<br>• Can contain 2 to 100 characters<br>• Can include numbers<br>• Can use a dash (-) but no other special characters<br>• Cannot contain spaces |
| **Workflow Path** | The path to the workflow engine on HDFS. The workflow path must be absolute, and the workflow file must be in a location that is readable by the user submitting the entity and by user "Falcon".<br><br>For example, if you are using a Pig script to define the workflow, you can set the path to `/apps/clickstream/clean-script.pig`. The workflow definition on HDFS contains the actual job that should run and it should conform to the workflow specification of the engine specified. The libraries required by the workflow should be in the `/lib` folder inside the workflow path. |
| **Cluster** | Select an existing cluster entity to specify where the workflow runs. |
| **Inputs** | The input data for the workflow. Each input maps to a feed. The path and frequency of input data is picked up from the feed entity definition. Feeds with Hive table storage can be used as inputs to a process. The workflow job starts only if all the inputs are available. |
| **Outputs** | The output data that is generated by the workflow. Each output is mapped to a feed and the output path is picked up from the feed definition. |
| **Name** | Name of the Spark application. The naming criteria are as follows:<br>• Must start with a letter<br>• Can contain 2-40 characters<br>• Can include numbers<br>• Can use a dash (-) but no other special characters<br>• Cannot contain spaces |
| **Application** (Spark only) | Specifies the `.jar` or Python file to be executed by Spark. Enter a valid HDFS path, including the file name. |
| **Main Class** (Spark only) | The main class for the `.jar` file specified. If the entry does not conform to a valid Java class name, the process cannot be submitted and a message is displayed: *Please enter your application's main class name*. |
| **Runs On** (Spark only) | Determines whether the application runs through YARN or locally on the Spark Master daemon. If the process is submitted with a master unreachable, the process is flagged as non-executable. |
| **Mode** (Spark only) | Only available if Spark is running on YARN. If run in cluster mode, it runs remotely on an ApplicationMaster. In client mode, the driver process runs locally. |
| **Spark Options** (Spark only) | Enter any Spark options you want to implement with this process. |
| **Spark Arguments** (Spark only) | Enter the arguments for the Spark options. |
| **Run Duration Start and End** | Combined with the frequency value to determine the window of time in which a Falcon job can execute. The times at which the process begins and finishes. The workflow job starts executing after the schedule time and when all the inputs are available.The workflow ends *before* the specified end time, so there is not a workflow instance at end time. Also known as *validity* time. |
| **Frequency** | How often the process is generated. Valid frequency types are minutes, hours, days, and months. |
| **Timezone** | The timezone is associated with the duration start and end times. Default timezone is UTC. |

### Table 4.14. Advanced Process Properties

| Property | Description |
|---|---|
| **Retry Policy Type** | Defines how the workflow failures should be handled. Options are Periodic, Exponential Backup, and None. |
| **Delay Up to** | The time period after which a retry attempt is made. For example, an Attempt value of 3 and Delay value of 10 minutes would cause the workflow retry to occur after 10 minutes, 20 minutes, and 30 minutes after the start time of the workflow. |
| **Attempts** | How many times the retry policy should be implemented before the job fails. |
| **Max Parallel Instances** | How many instances of the workflow can run concurrently. Must be greater than 0. For example, a value of 2 allows two instances of the workflow to run at the same time; the next instance starts only after the running instance completes. |
| **Order** | The order in which the ready instances are picked up. Options are First-in-first-out (FIFO), Last-in-first-out (LIFO), and Last_Only. |
| **Properties** | Enter the name and value for any properties you want to assign to this process. |
| **Access Control List** | Specify the HDFS owner, group, and access permissions for the cluster. Default permissions are 755 (rwx/r-x/r-x). The owner must be the owner of the Falcon `staging` and `working` directories. |

3. Click **Next** to view a summary of your process entity definition.

4. (Optional) Click **Preview XML** to review or edit the entity definition in XML.

5. After verifying the entity definition, click **Save**.

   The entity is automatically submitted for verification, but it is not scheduled to run.

6. Verify that you successfully created the process entity.

   a. Type the entity name in the Falcon web UI **Search** field and press **Enter**.

   b. If the entity name appears in the search results, it was successfully created.

      For more information about the search function, see "Locating and Managing Entities" in Using Advanced Falcon Features.

7. Schedule the entity.

   a. In the search results, click the checkbox next to an entity name with status of `Submitted`.

   b. Click Schedule.

      After a few seconds a success message displays.

## 4.4.4. Scheduling or Pausing an Entity Using the Web UI

When you create a Falcon entity in the web UI, the entity is automatically submitted to the cluster servers you specified when creating the entity. When submitted, the entity is added to the configuration store in Falcon and several validations are run against the entity definition. However, the entity must be manually scheduled before a job can run.

When you schedule an entity, the workflow and library you specified are copied to the staging path you created, and the scheduler references the workflow and library from that staging path.

**Note**

You must have completed Section 4.3, “Creating HDFS Directories for Falcon” [10] prior to submitting the Falcon entities.

1. Use the Search field to locate the entity you want schedule.
2. In the search results list, click the checkbox next to the names of the feeds or processes with status *Submitted* that you want to schedule and then click **Schedule**.

   After a few seconds, you should receive a success message for each entity.
3. To suspend a scheduled entity, click the checkbox next to the names of the feeds or processes with status *Running* and then click **Pause**.

   You should receive a success message for each entity.

## 4.5. Defining Entities Using the CLI

You can manually create your Falcon entity definition files and workflow files, then submit and execute them from the CLI.

**Prerequisites**

Before you can begin working with Falcon, ensure that you have the following components installed on your cluster:

- Hortonworks Data Platform (HDP)
- Apache Falcon

  See Non-Ambari Cluster Installation Guide or Ambari Automated Install Guide for Falcon installation instructions.
- Apache Oozie client and server

**Steps**

To create a data pipeline, you must perform the following actions:

1. Section 4.3, “Creating HDFS Directories for Falcon” [10].
2. Section 4.5.1, “Creating a Cluster Entity Definition Using the CLI” [21].
3. Section 4.5.2, “Creating a Feed Entity Definition Using the CLI” [22].
4. Section 4.5.3, “Creating a Process Entity Definition Using the CLI” [23].
5. Section 4.5.4, “Submitting and Scheduling an Entity Using the CLI” [24]

### 4.5.1. Creating a Cluster Entity Definition Using the CLI

Create the cluster specification XML file, also known as a cluster entity. There are several items to define in a cluster entity.

In the example cluster entity provided below, the following properties are defined:

- **Colo:** Name of the Data Center
- **Name:** File name of the Data Center
- **<interface>:** Specify the interface type

#### Important

Ensure that the required HDFS directories have been created. 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. Keep the working directory permissions set to 755 (rwx/r-w/r-w).

```
<?xml version="1.0"?>
<!--
   Cluster Example
 -->
<cluster colo="$MyDataCenter" description="description" name="$MyDataCenter">

  <interfaces>
    <interface type="readonly" endpoint="hftp://nn:50070" version="2.4.2" />
 <!-- Required for distcp for replications. -->
    <interface type="write" endpoint="hdfs://nn:8020" version="2.4.2" /> <!--
 Needed for writing to HDFS-->
    <interface type="execute" endpoint="rm:8050" version="2.4.2" /> <!--
 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 Non-Ambari Cluster Installation guide.

For more information, see the Apache cluster entity XSD.

### 4.5.2. Creating a Feed Entity Definition Using the CLI

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.
- Optional. Specify an Email Notification. The email notification tag must be placed before the <ACL> tag.
- **<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>
  <notification type="email" to="falcon@xyz.com"/>
  <ACL owner="hdfs" group="users" permission="0755"/>  <!-- Required for HDFS.
 -->
  <schema location="/none" provider="none"/> <!-- Required for HDFS. -->
</feed>
```

For more information, see the Apache feed entity XSD.

### 4.5.3. Creating a Process Entity Definition Using the CLI

When submitting a `workflow.xml` file as part of a process entity definition, ensure the following:

- The `workflow.xml` path is absolute.
- The file is in a location that is readable by the user submitting the entity and by service user "falcon".

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.
- Optional. Specify an Email Notification.

```
<?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>
    <notification type="email" to="falcon@xyz.com, falcon_2@xyz.com"/>
</process>
```

**Note**

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

For more information, see the Apache process entity XSD.

## 4.5.4. Submitting and Scheduling an Entity Using the CLI

After creating the Falcon entities, you must submit and schedule the entities with the cluster servers you specified when creating the entities.

When you submit an entity, the entity is added to the configuration store in Falcon and several validations are run against the entity definition.

When you schedule an entity, the workflow and library you specified are copied to the staging path you created, and the scheduler references the workflow and library from that staging path.

### Note

You must have completed Section 4.3, “Creating HDFS Directories for Falcon” [10] prior to submitting the Falcon entities.

You should receive a success message after submitting and scheduling each entity.

1. Submit your entities to Falcon, ensuring you specify the correct entity type for each file.

   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, Spark jobs, Pig scripts, and Hive queries. You can now explore other sections such as Late Data Handling or Retry Policy.

# 5. Mirroring Data with Falcon

You can mirror data between on-premise clusters or between an on-premises HDFS cluster and a cluster in the cloud using Microsoft Azure or Amazon S3.

## 5.1. Preparing to Mirror Data

Mirroring data produces an exact copy of the data and keeps both copies synchronized. You can use Falcon to mirror HDFS directories, Hive tables, and snapshots.

**Before creating a mirror, complete the following actions:**

1. Set permissions to allow read, write, and execute access to the source and target directories.

   You must be logged in as the owner of the directories.

   Example: If the source directory were in `/user/ambari-qa/falcon`, type the following:

   ```
   [bash ~]$ su - root
   [root@bash ~]$ su - ambari-qa
   [ambari-qa@bash ~]$ hadoop fs -chmod 755 /user/ambari-qa/falcon/
   ```

2. Create the source and target cluster entity definitions, if they do not exist.

   See "Creating a Cluster Entity Definition" in Creating Falcon Entity Definitions for more information.

3. For snapshot mirroring, you must also enable the snapshot capability on the source and target directories.

   You must be logged in as the HDFS Service user and the source and target directories must be owned by the user submitting the job.

   For example:

   ```
   [ambari-qa@bash ~]$ su - hdfs
   ## Run the following command on the target cluster
   [hdfs@bash ~]$ hdfs dfsadmin -allowSnapshot /apps/falcon/snapshots/target
   ## Run the following command on the source cluster
   [hdfs@bash ~]$ hdfs dfsadmin -allowSnapshot /apps/falcon/snapshots/source
   ```

## 5.2. Mirroring File System Data Using the Web UI

You can use the Falcon web UI to quickly define a mirror job and start a mirror job on HDFS.

**Prerequisites**

Your environment must meet the HDP versioning requirements described in "Replication Between HDP Versions" in Creating Falcon Entity Definitions.

### Steps

1. Ensure that you have set permissions correctly and defined required entities as described in Preparing to Mirror Data.
2. At the top of the Falcon web UI page, click **Create** > **Mirror** > **File System**.
3. On the New HDFS Mirror page, specify the values for the following properties:

### Table 5.1. General HDFS Mirror Properties

| **Property** | **Description** |
|---|---|
| **Mirror Name** | Name of the mirror job. The naming criteria are as follows:<br>• Must be unique<br>• Must start with a letter<br>• Is case sensitive<br>• Can contain a maximum of 40 characters<br>• Can include numbers<br>• Can use a dash (-) but no other special characters<br>• Cannot contain spaces |
| **Tags** | Enter the key/value pair for metadata tagging to assist in entity search in Falcon. The criteria are as follows:<br>• Can contain 1 to 100 characters<br>• Can include numbers<br>• Can use a dash (-) but no other special characters<br>• Cannot contain spaces |

### Table 5.2. Source and Target Mirror Properties

| **Property** | **Description** |
|---|---|
| **Source Location** | Specify whether the source data is local on HDFS or on Microsoft Azure or Amazon S3 in the cloud. If your target is Azure or S3, you can only use HDFS for the source. |
| **Source Cluster** | Select an existing cluster entity. |
| **Source Path** | Enter the path to the source data. |
| **Target Location** | Specify whether the mirror target is local on HDFS or on Microsoft Azure or Amazon S3 in the cloud. If your target is Azure or S3, you can only use HDFS for the source. |
| **Target Cluster** | Select an existing cluster entity to serve as target for the mirrored data. |
| **Target Path** | Enter the path to the directory that will contain the mirrored data. |
| **Run job here** | Choose whether to execute the job on the source or on the target cluster. |
| **Validity Start**and **End** | Combined with the frequency value to determine the window of time in which a Falcon mirror job can execute. The workflow job starts executing after the schedule time and when all the inputs are available. The workflow ends *before* the specified end time, so there is not a workflow instance at end time. Also known as *run duration*. |
| **Frequency** | How often the process is generated. Valid frequency types are minutes, hours, days, and months. |

| Property | Description |
|---|---|
| **Timezone** | The timezone is associated with the start and end times. Default timezone is UTC. |
| **Send alerts to** | A comma-separated list of email addresses to which alerts are sent, in the format *name@company.com*. |

### Table 5.3. Advanced HDFS Mirror Properties

| Property | Description |
|---|---|
| **Max Maps for DistCp** | The maximum number of maps used during replication. This setting impacts performance and throttling. Default is 5. |
| **Max Bandwidth (MB)** | The bandwidth in MB/s used by each mapper during replication. This setting impacts performance and throttling. Default is 100 MB. |
| **Retry Policy** | Defines how the workflow failures should be handled. Options are Periodic, Exponential Backoff, and Final. |
| **Delay** | The time period after which a retry attempt is made. For example, an Attempt value of 3 and Delay value of 10 minutes would cause the workflow retry to occur after 10 minutes, 20 minutes, and 30 minutes after the start time of the workflow. Default is 30 minutes. |
| **Attempts** | How many times the retry policy should be implemented before the job fails. Default is 3. |
| **Access Control List** | Specify the HDFS owner, group, and access permissions for the cluster. Default permissions are 755 (rwx/r-x/r-x). |

4. Click **Next** to view a summary of your entity definition.
5. (Optional) Click **Preview XML** to review or edit the entity definition in XML.
6. After verifying the entity definition, click **Save**.

   The entity is automatically submitted for verification, but it is not scheduled to run.
7. Verify that you successfully created the entity.
   a. Type the entity name in the Falcon web UI **Search** field and press **Enter**.
   b. If the entity name appears in the search results, it was successfully created.

      For more information about the search function, see "Locating and Managing Entities" in Using Advanced Falcon Features.
8. Schedule the entity.
   a. In the search results, click the checkbox next to an entity name with status of `Submitted`.
   b. Click Schedule.

      After a few seconds a success message displays.

## 5.3. Mirroring Hive Data Using the Web UI

You can quickly mirror Apache Hive databases or tables between source and target clusters with HiveServer2 endpoints. You can also enable TDE encryption for your mirror.

1. Ensure that you have set permissions correctly and defined required entities as described in Preparing to Mirror Data.

2. At the top of the Falcon web UI page, click **Create** > **Mirror** > **Hive**.

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

### Table 5.4. General Hive Mirror Properties

| Property | Description |
|---|---|
| **Mirror Name** | Name of the mirror job. The naming criteria are as follows:<br>• Must be unique<br>• Must start with a letter<br>• Is case sensitive<br>• Can contain 2 to 40 characters<br>• Can include numbers<br>• Can use a dash (-) but no other special characters<br>• Cannot contain spaces |
| **Tags** | Enter the key/value pair for metadata tagging to assist in entity search in Falcon. The criteria are as follows:<br>• Can contain 1 to 100 characters<br>• Can include numbers<br>• Can use a dash (-) but no other special characters<br>• Cannot contain spaces |

### Table 5.5. Source and Target Hive Mirror Properties

| Property | Description |
|---|---|
| **Source, Cluster** | Select an existing cluster entity to serve as source for the mirrored data. At least one cluster entity must be available in Falcon. |
| **Source, HiveServer2 Endpoint** | The Hive2 server endpoint, in the format hive2://*localhost*:1000. This is the location of data to be mirrored. |
| **Source, Hive2 Kerberos Principal** | The service principal for the metastore Thrift server. The field is automatically populated with the value displayed in Ambari at `Hive>Config>Advanced>Advanced hive-site>hive.metastore.kerberos.principal`. |
| **Source, Meta Store URI** | Used by the metastore client to connect to the remote metastore. Found in Ambari at `Hive>Config>Advanced>General>hive.metastore.uris`. |
| **Source, Kerberos Principal** | The field is automatically populated with the value. |
| **Target, Cluster** | Select an existing cluster entity to serve as target for the mirrored data. At least one cluster entity must be available in Falcon. |
| **Target, HiveServer2 Endpoint** | The Hive2 server endpoint, in the format hive2://*localhost*:1000. This is the location of the mirrored data. |
| **Target, Hive2 Kerberos Principal** | The service principal for the metastore Thrift server. The field is automatically populated with the value displayed in |

| Property | Description |
|---|---|
| | Ambari at `Hive>Config>Advanced>Advanced hive-site>hive.metastore.kerberos.principal`. |
| **Target, Meta Store URI** | Used by the metastore client to connect to the remote metastore. Found in Ambari at `Hive>Config>Advanced>General>hive.metastore.uris`. |
| **Target, Kerberos Principal** | The field is automatically populated with the value. |
| **Run job here** | Choose whether to execute the job on the source or on the target cluster. |
| **I want to copy** | Select to copy one or more Hive databases or copy one or more tables from a single database. You must identify the specific databases and tables to be copied. |
| **Validity Start**and **End** | Combined with the frequency value to determine the window of time in which a Falcon mirror job can execute. The workflow job starts executing after the schedule time and when all the inputs are available. The workflow ends *before* the specified end time, so there is not a workflow instance at end time. Also known as *run duration*. |
| **Frequency** | How often the process is generated. Valid frequency types are minutes, hours, days, and months. |
| **Timezone** | The timezone is associated with the validity start and end times. Default timezone is UTC. |
| **Send alerts to** | A comma-separated list of email addresses to which alerts are sent, in the format *name@xyz.com*. |

## Table 5.6. Advanced Hive Mirror Properties

| Property | Description |
|---|---|
| **TDE Encryption** | Enable to encrypt data at rest. See "Enabling Transparent Data Encryption" in Using Advanced Features for more information. |
| **Max Maps for DistCp** | The maximum number of maps used during replication. This setting impacts performance and throttling. Default is 5. |
| **Max Bandwidth (MB)** | The bandwidth in MB/s used by each mapper during replication. This setting impacts performance and throttling. Default is 100 MB. |
| **Retry Policy** | Defines how the workflow failures should be handled. Options are Periodic, Exponential Backoff, and Final. |
| **Delay** | The time period after which a retry attempt is made. For example, an Attempt value of 3 and Delay value of 10 minutes would cause the workflow retry to occur after 10 minutes, 20 minutes, and 30 minutes after the start time of the workflow. Default is 30 minutes. |
| **Attempts** | How many times the retry policy should be implemented before the job fails. Default is 3. |
| **Access Control List** | Specify the HDFS owner, group, and access permissions for the cluster. Default permissions are 755 (rwx/r-x/r-x). |

4. Click **Next** to view a summary of your entity definition.

5. (Optional) Click **Preview XML** to review or edit the entity definition in XML.

6. After verifying the entity definition, click **Save**.

   The entity is automatically submitted for verification, but it is not scheduled to run.

7. Verify that you successfully created the entity.

   a. Type the entity name in the Falcon web UI **Search** field and press **Enter**.

   b. If the entity name appears in the search results, it was successfully created.

      For more information about the search function, see "Locating and Managing Entities" in Using Advanced Falcon Features.

8. Schedule the entity.

   a. In the search results, click the checkbox next to an entity name with status of `Submitted`.

   b. Click Schedule.

      After a few seconds a success message displays.

## 5.4. Mirroring Data Using Snapshots

Snapshot-based mirroring is an efficient data backup method because only updated content is actually transferred during the mirror job. You can mirror snapshots from a single source directory to a single target directory. The destination directory is the target for the backup job.

**Prerequisites**

- Source and target clusters must run Hadoop 2.7.0 or higher.

  Falcon does not validate versions.

- Source and target clusters should both be either secure or unsecure.

  This is a recommendation, not a requirement.

- Source and target clusters must have snapshot capability enabled (the default is "enabled").

- The user submitting the mirror job must have access permissions on both the source and target clusters.

**To mirror snapshot data with the Falcon web UI:**

1. Ensure that you have set permissions correctly, enabled snapshot mirroring, and defined required entities as described in Preparing to Mirror Data.

2. At the top of the Falcon web UI page, click **Create** > **Mirror** > **Snapshot**.

3. On the New Snapshot Based Mirror page, specify the values for the following properties:

Table 5.7. Source and Target Snapshot Mirror Properties

| Property | Description |
|---|---|
| **Source, Cluster** | Select an existing source cluster entity. At least one cluster entity must be available in Falcon. |
| **Target, Cluster** | Select an existing target cluster entity. At least one cluster entity must be available in Falcon. |

| Property | Description |
|---|---|
| **Source, Directory** | Enter the path to the source data. |
| **Source, Delete Snapshot After** | Specify the time period after which the mirrored snapshots are deleted from the source cluster. Snapshots are retained past this date if the number of snapshots is less than the Keep Last setting. |
| **Source, Keep Last** | Specify the number of snapshots to retain on the source cluster, even if the delete time has been reached. Upon reaching the number specified, the oldest snapshot is deleted when the next job is run. |
| **Target, Directory** | Enter the path to the location on the target cluster in which the snapshot is stored. |
| **Target, Delete Snapshot After** | Specify the time period after which the mirrored snapshots are deleted from the target cluster. Snapshots are retained past this date if the number of snapshots is less than the Keep Last setting. |
| **Target, Keep Last** | Specify the number of snapshots to retain on the target cluster, even if the delete time has been reached. Upon reaching the number specified, the oldest snapshot is deleted when the next job is run. |
| **Run job here** | Choose whether to execute the job on the source or on the target cluster. |
| **Run Duration Start**and **End** | Combined with the frequency value to determine the window of time in which a Falcon mirror job can execute. The workflow job starts executing after the schedule time and when all the inputs are available. The workflow ends *before* the specified end time, so there is not a workflow instance at end time. Also known as *validity* time. |
| **Frequency** | How often the process is generated. Valid frequency types are minutes, hours, days, and months. |
| **Timezone** | Default timezone is UTC. |

## Table 5.8. Advanced Snapshot Mirror Properties

| Property | Description |
|---|---|
| **TDE Encryption** | Enable to encrypt data at rest. See "Enabling Transparent Data Encryption" in Using Advanced Features for more information. |
| **Retry Policy** | Defines how the workflow failures should be handled. Options are Periodic, Exponential Backup, and Final. |
| **Delay** | The time period after which a retry attempt is made. For example, an Attempt value of 3 and Delay value of 10 minutes would cause the workflow retry to occur after 10 minutes, 20 minutes, and 30 minutes after the start time of the workflow. Default is 30 minutes. |
| **Attempts** | How many times the retry policy should be implemented before the job fails. Default is 3. |
| **Max Maps** | The maximum number of maps used during DistCp replication. This setting impacts performance and throttling. Default is 5. |
| **Max Bandwidth (MB)** | The bandwidth in MB/s used by each mapper during replication. This setting impacts performance and throttling. Default is 100 MB. |
| **Send alerts to** | A comma-separated list of email addresses to which alerts are sent, in the format *name@xyz.com*. |
| **Access Control List** | Specify the HDFS owner, group, and access permissions for the cluster. Default permissions are 755 (rwx/r-x/r-x). |

4. Click **Next** to view a summary of your entity definition.

5. (Optional) Click **Preview XML** to review or edit the entity definition in XML.

6. After verifying the entity definition, click **Save**.

The entity is automatically submitted for verification, but it is not scheduled to run.

7. Verify that you successfully created the entity.

   a. Type the entity name in the Falcon web UI **Search** field and press **Enter**.

   b. If the entity name appears in the search results, it was successfully created.

      For more information about the search function, see "Locating and Managing Entities" in Using Advanced Falcon Features.

8. Schedule the entity.

   a. In the search results, click the checkbox next to an entity name with status of `Submitted`.

   b. Click Schedule.

      After a few seconds a success message displays.

## 5.5. Mirroring File System Data Using the CLI

See the Apache Falcon website for information about using the CLI to mirror data.

# 6. Replicating Data with Falcon

Falcon can replicate data across multiple clusters using DistCp A replication feed allows you to set a retention policy 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 that pulls the data using DistCp from the source cluster.

### Note

Due to changes in Hive, for the Oozie HCAT URI (which is used for Hive table feeds) Falcon supports URIs with only one metastore. This applies even if you have multiple metastores configured.

## 6.1. Prerequisites

- Your environment must meet the HDP versioning requirements described in "Replication Between HDP Versions" in Creating Falcon Entity Definitions.
- You must have the following components installed on your cluster:
  - **HDP**: Installed on your cluster (using Ambari or a manual installation)
  - **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

## 6.2. Replicating Data Using the CLI

### 6.2.1. 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>
   ```

   `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.
4.2" />

          <!-- Needed for writing to HDFS-->
          <interface type="write" endpoint="hdfs://nn:8020" version="2.4.
2" />

          <!-- Required. An execute interface specifies the interface for
 job tracker.-->
          <interface type="execute" endpoint="rm:8050" version="2.4.2" />

          <!-- 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.
4.2" />

          <!-- Needed for writing to HDFS-->
          <interface type="write" endpoint="hdfs://nn:8020" version="2.4.
2" />

          <!-- Needed to write to jobs as MapReduce-->
          <interface type="execute" endpoint="rm:8050" version="2.4.2" />

          <!-- 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.
4.2" />

          <!-- Needed for writing to HDFS-->
          <interface type="write" endpoint="hdfs://nn:8020" version="2.4.
2" />

          <!-- Needed to write to jobs as MapReduce-->
          <interface type="execute" endpoint="rm:8050" version="2.4.2" />

          <!-- 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.

### 6.2.2. 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.

### 6.2.3. 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">
   </feed>
   ```

3. Specify the frequency of the feed.

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

```
<feed description="$rawInputFeed" name=”testFeed” xmlns="uri:falcon:feed:0.
1">

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

</feed>
```

4. Set how frequently the feed should run.

   For example:

```
<?xml version="1.0"?>
<feed description="$rawInputFeed" name=”testFeed” xmlns="uri:falcon:feed:0.
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 and set a retention policy for the data.

   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}-
${DAY}-${HOUR} "/>
       <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}-
${DAY}-${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>
</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}-
${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"/>

</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>
```

## 6.2.4. Submit and Schedule the Entities

You must submit and schedule all of your entities. The entity definitions are validated as part of the submission process.

1. Submit all of your entities using the following syntax:

   ```
   falcon entity -type <EntityType> -submit -file <YourEntity.xml>
   ```

2. Schedule your feed and process entities using the following syntax:

   ```
   falcon entity -type <EntityType> -name <YourEntity -schedule
   ```

See "Submitting and Scheduling an Entity Using the CLI" in Creating Falcon Entity Definitions for more specific instructions.

## 6.2.5. Confirm Results

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

# 7. Mirroring Data with HiveDR in a Secure Environment

You can use Falcon to mirror data in Hive databases, tables, or partitions. Before setting up a HiveDR job in Falcon, you should configure a secure environment.

### Important

Before implementing HiveDR with Falcon, you should discuss your plans with Hortonworks Professional Services and Product Management to ensure proper component configuration and adequate protection of your data.

**Prerequisites**

- The Kerberized source and target clusters must exist.
- You must be running the following Apache components:
  - Hive 1.2.0 or later
  - Oozie 4.2.0 or later
  - Sqoop 1.4.6 or later

    Falcon uses Apache Sqoop for import and export operations. Sqoop requires a database driver to connect to the relational database. Refer to the Sqoop documentation if you need more information.
- Ensure that the database driver `.jar` file is copied into the Oozie share library for Sqoop.

**About this task**

Database replication definitions can be set up only on preexisting databases. Therefore, before you can replicate a database the first time, you must create a target database. To prevent unintentional replication of private data, Hive does not allow the replication of a database without a pre-existing target.

After the database target is created, you must export all tables in the source database and import them in the target database.

**Steps**

1. Create a Hive database on the target cluster

   You only need to perform this task if you are replicating an entire database. See the Data Access guide or the Apache Hive documentation.
2. Create a Hive table/partition on the source cluster

   You only need to perform this task if the table or partition you want to replicate does not already exist. See the Data Access guide or the Apache Hive documentation
3. Configuring for High Availability

4. Preparing to Mirror Data
5. Section 7.1, “Configure Properties for HiveDR” [42]
6. Section 7.2, “Initialize Data for Hive Replication” [43]
7. Section 7.3, “Mirror Hive Data Using the Web UI” [44]

## 7.1. Configure Properties for HiveDR

You must configure several Hadoop and Hive properties to set up a secure environment before you can implement HiveDR jobs.

1. In Ambari, on the *source* cluster, navigate to **HDFS** > **Configs** > **Advanced**.
2. Scroll to the **Custom core-site** section and specify the host names for each of the following attributes:

   **Property:**hadoop.proxyuser.hdfs.hosts, **Value:** The host name is located at **HDFS** > **Configs** > **Advanced** > **NameNode**.
   **Property:**hadoop.proxyuser.yarn.hosts, **Value:** The host name is located at **YARN** > **Configs** > **Advanced** > **Resource Manager**.
   **Property:**hadoop.proxyuser.hive.hosts, **Value:** The host name is located at **Hive** > **Configs** > **Advanced** > **Hive Metastore**.
   **Property:**hadoop.proxyuser.oozie.hosts, **Value:** The host name is located at **Oozie** > **Configs** > **Oozie Server**.
   **Property:**hadoop.proxyuser.hcat.hosts, **Value:** The host name is located at **Hive** > **Configs** > **Advanced** > **Hive Metastore**
   **Property:**hadoop.proxyuser.http.hosts, **Value:** The host name is located at **Hive** > **Configs** > **Advanced** > **Hive Metastore**
   **Property:**hadoop.proxyuser.falcon.hosts, **Value:** The host name is located at **Falcon** > **Configs** > **Falcon Server**.

   You can use an asterisk (*) in place of a host name to indicate that any host can connect.
3. Navigate to **Hive** > **Configs** > **Advanced**.
4. Scroll to the **Custom hive-site** section and click **Add Property**.
5. Add the indicated values for the following properties:

   **Property:** hive.metastore.event.listeners, **Value:** org.apache.hive.hcatalog.listener.DbNotificationListener
   **Property:** hive.metastore.dml.events, **Value:** true
   **Property:** hive.metastore.event.db.listener.timetolive, **Value:** 432000s
   **Property:** hive.exim.strict.repl.tables, **Value:** true
   **Property:** hive.server2.transport.mode, **Value:** binary
6. Restart the impacted services on the cluster.
7. Repeat steps 1 through 6 on the *target* cluster.

**What's Next?**

You need to create both the source and the destination cluster entities from the Falcon UI.

"Creating a Cluster Entity Definition Using the Web UI"

## 7.2. Initialize Data for Hive Replication

Before running a HiveDR job, you must create an initial copy of any table or database that you intend to replicate on an ongoing basis. This initialization process is sometimes referred to as *bootstrapping*. Falcon uses the initialized data on the target cluster as a baseline to do incremental replications.

You perform an initialization by using the `export` and `import` commands. When you export the source data, you assign a base event ID. The event ID is incremented each time a new replication job runs. When the exported data is imported, a check is done to determine if a table or database already exists that has the same base event ID. If the incremented event ID of the imported data indicates that the imported data is newer, then the existing data with the same base ID is replaced. If the imported data is older, no update is performed and no error is generated.

**Prerequisites**

- For database replication, the target database must exist on the target cluster.
- The Hive tables or partitions must exist on the source cluster.
- The Falcon cluster entities must exist on both the source cluster and the destination cluster.
- You need the full path to any databases and tables you want to replicate.

In the following steps, replaceable text in code examples is shown in italics.

1. Export the table or partition data to an output location on the source, using the for replication(eventid) tag.

   This ID tag must not contain numbers.

   For example, to export an SQL table named `global_sales`:

   ```
   hive > export table global_sales to '/user/tmp/global_sales' for
   replication(july_global_sales);
   ```

2. Run the **distcp** command to copy the exported data to the target Hive data warehouse on the destination cluster.

   ```
   $ hadoop distcp hdfs://machine-1-1.openstacklocal:8020/user/
   tmp/global_sales hdfs://machine-2-1.openstacklocal:8020/user/
   hive_imports/global_sales
   ```

3. On the destination cluster, perform an import on the exported data.

   ```
   hive > import table global_sales from '/user/hive_imports/
   global_sales' location '/user/sales/global_sales';
   ```

For more information, see the Apache Hive LanguageManual ImportExport wiki page.

## 7.3. Mirror Hive Data Using the Web UI

You can quickly mirror Apache Hive databases or tables between source and target clusters with HiveServer2 endpoints. You can also enable TDE encryption for your mirror.

1. Ensure that you have set permissions correctly and defined required entities as described in Preparing to Mirror Data.
2. At the top of the Falcon web UI page, click **Create** > **Mirror** > **Hive**.
3. On the New Hive Mirror page, specify the values for the following properties:

### Table 7.1. General Hive Mirror Properties

| Property | Description | Details |
|---|---|---|
| **Mirror Name** | Name of the mirror job. | The naming criteria are as follows:<br>• Must be unique<br>• Must start with a letter<br>• Is case sensitive<br>• Can contain 2 to 40 characters<br>• Can include numbers<br>• Can use a dash (-) but no other special characters<br>• Cannot contain spaces |
| **Tags** | Enter the key/value pair for metadata tagging to assist in entity search in Falcon. | The criteria are as follows:<br>• Can contain 1 to 100 characters<br>• Can include numbers<br>• Can use a dash (-) but no other special characters<br>• Cannot contain spaces |

### Table 7.2. Source and Target Hive Mirror Properties

| Property | Description | Details |
|---|---|---|
| **Source, Cluster** | Select an existing cluster entity to serve as source for the mirrored data. | At least one cluster entity must be available in Falcon. |
| **Source, HiveServer2 Endpoint** | The Hive2 server endpoint. This is the location of data to be mirrored. | The format is hive2://*localhost*:1000. |
| **Source, Hive2 Kerberos Principal** | The service principal for the metastore Thrift server. The field is automatically populated with the value. | The value is displayed in Ambari at `Hive>Config>Advanced>Advanced hive-site>hive.metastore.kerberos.principal`. |
| **Source, Meta Store URI** | Used by the metastore client to connect to the remote metastore. | The value is displayed in Ambari at `Hive>Config>Advanced>General>hive.metastore.uris`. |
| **Source, Kerberos Principal** | The field is automatically populated. | Property=dfs.namenode.kerberos.principal and Value=nn/_HOST@EXAMPLE.COM. |

| Property | Description | Details |
|---|---|---|
| **Target, Cluster** | Select an existing cluster entity to serve as target for the mirrored data. | At least one cluster entity must be available in Falcon. |
| **Target, HiveServer2 Endpoint** | The Hive2 server endpoint. This is the location of the mirrored data. | The format is hive2://*localhost*:1000. |
| **Target, Hive2 Kerberos Principal** | The service principal for the metastore Thrift server. The field is automatically populated. | The value is displayed in Ambari at `Hive>Config>Advanced>Advanced hive-site>hive.metastore.kerberos.principal`. |
| **Target, Meta Store URI** | Used by the metastore client to connect to the remote metastore. | The value is displayed in Ambari at `Hive>Config>Advanced>General>hive.metastore.uris`. |
| **Target, Kerberos Principal** | The field is automatically populated. | Property=dfs.namenode.kerberos.principal and Value=nn/_HOST@EXAMPLE.COM. |
| **Run job here** | Choose whether to execute the job on the source cluster or on the target cluster. | None |
| **I want to copy** | Select to copy one or more Hive databases or copy one or more tables from a single database. You must identify the specific databases and tables to be copied. | None |
| **Validity Startand End** | Combined with the frequency value to determine the window of time in which a Falcon mirror job can execute. | The workflow job starts executing after the schedule time and when all the inputs are available. The workflow ends *before* the specified end time, so there is not a workflow instance at end time. Also known as *run duration*. |
| **Frequency** | How often the process is generated. | Valid frequency types are minutes, hours, days, and months. |
| **Timezone** | The timezone is associated with the validity start and end times. | Default timezone is UTC. |
| **Send alerts to** | A comma-separated list of email addresses to which alerts are sent. | The format is *name@xyz.com*. |

## Table 7.3. Advanced Hive Mirror Properties

| Property | Description | Details |
|---|---|---|
| **TDE Encryption** | Enables encryption of data at rest. | See "Enabling Transparent Data Encryption" in Using Advanced Features for more information. |
| **Max Maps for DistCp** | The maximum number of maps used during replication. | This setting impacts performance and throttling. Default is 5. |
| **Max Bandwidth (MB)** | The bandwidth in MB/s used by each mapper during replication. | This setting impacts performance and throttling. Default is 100 MB. |
| **Retry Policy** | Defines how the workflow failures should be handled. | Options are Periodic, Exponential Backoff, and Final. |
| **Delay** | The time period after which a retry attempt is made. | For example, an Attempt value of 3 and Delay value of 10 minutes would cause the workflow retry to occur after 10 minutes, 20 minutes, and 30 minutes after the start time of the workflow. Default is 30 minutes. |
| **Attempts** | How many times the retry policy should be implemented before the job fails. | Default is 3. |
| **Access Control List** | Specify the HDFS owner, group, and access permissions for the cluster. | Default permissions are 755 (rwx/r-x/r-x). |

4. Click **Next** to view a summary of your entity definition.

5. (Optional) Click **Preview XML** to review or edit the entity definition in XML.
6. After verifying the entity definition, click **Save**.

   The entity is automatically submitted for verification, but it is not scheduled to run.
7. Verify that you successfully created the entity.
   a. Type the entity name in the Falcon web UI **Search** field and press **Enter**.
   b. If the entity name appears in the search results, it was successfully created.

      For more information about the search function, see "Locating and Managing Entities" in Using Advanced Falcon Features.
8. Schedule the entity.
   a. In the search results, click the checkbox next to an entity name with status of `Submitted`.
   b. Click Schedule.

      After a few seconds a success message displays.

# 8. Enabling Mirroring and Replication with Azure Cloud Services

You can enable secure connectivity between your on-premises Apache Hadoop cluster and a Microsoft Azure Data Factory service in the cloud. You might want to create a hybrid data pipeline for uses such as maintaining sensitive data on premises while leveraging the cloud for nonsensitive data, providing disaster recovery or replication services, and using the cloud for development or test environments.

**Prerequisites**

- You must have created a properly configured data factory in the cloud.
- Your environment must meet the HDP versioning requirements described in "Replication Between HDP Versions" in Creating Falcon Entity Definitions.

### Note

Due to changes in Hive, for the Oozie HCAT URI (which is used for Hive table feeds) Falcon supports URIs with only one metastore. This applies even if you have multiple metastores configured.

## 8.1. Connect the Azure Data Factory to Your On-premises Hadoop Cluster

1. In the Data Factory editor, create a new Hadoop cluster.

   As part of the creation process, you must identify the following linked services. You need the names of these services when configuring the Falcon instance.

   - Hadoop cluster Linked Service

     Represents the on-premises Hadoop cluster as a data factory compute resource. You can use this resource as compute target for your data factory Hive and Pig jobs. This linked service references the service bus (transport) linked service. The service is identified as OnPremisesHadoopCluster linked service.

   - Service Bus Namespace

     Contains information about a unique Azure service bus namespace that is used for communicating job requests and status information between the data factory and your on-premises Hadoop cluster. The service is identified as TransportLinkedService.

   The JSON object for the linked services can be found in the `./Data Factory JSONs/LinkedServices` folder.

2. Configure an on-premises Falcon instance to connect with the data factory Hadoop cluster.

Add the following information to your Falcon `conf/startup.properties` file, with the following changes:

- Replace <**`your azure service bus namespace`**> with the name you assigned to the service bus in step 1.
- Replace <**`your Azure service bus SAS key`**> with the credentials for the Azure bus namespace taken from the Azure web portal.

```
######### ADF Configurations start #########

# A String object that represents the namespace

*.microsoft.windowsazure.services.servicebus.namespace=<your azure service
 bus namespace>

# Request and status queues on the namespace

*.microsoft.windowsazure.services.servicebus.requestqueuename=adfrequest
*.microsoft.windowsazure.services.servicebus.statusqueuename=adfstatus

# A String object that contains the SAS key name
*.microsoft.windowsazure.services.servicebus.sasKeyName=
RootManageSharedAccessKey

# A String object that contains the SAS key
*.microsoft.windowsazure.services.servicebus.sasKey=<your Azure service bus
 SAS key>

# A String object containing the base URI that is added to your Service Bus
 namespace to form the URI to connect
# to the Service Bus service. To access the default public Azure service,
 pass ".servicebus.windows.net"
*.microsoft.windowsazure.services.servicebus.serviceBusRootUri=.servicebus.
windows.net

# Service bus polling frequency (in seconds)
*.microsoft.windowsazure.services.servicebus.polling.frequency=60
```

3. Restart Falcon from the Ambari UI.

   a. Click the **Services** tab and select the **Falcon** service.

   b. On the **Summary** page, verify that the Falcon service status is **Stopped**.

   c. Click **Service Actions** > **Start**.

      A dialog box displays.

   d. (Optional) Turn off Falcon's maintenance mode by clicking the checkbox.

      Maintenance Mode suppresses alerts.

   e. Click **Confirm Start** > **OK**.

      On the Summary page, the Falcon status displays as Started when the service is available.

After restarting Falcon, the Azure data factory and the on-premises Falcon instance should successfully connect.

4. Create datasets and data pipelines to run Hive and Pig jobs that target the on-premises Hadoop cluster.

   **Example**

   Following is an example of a script to test running a Hive script on an on-premises HDP cluster.

   JSONs for all the objects can be found in ./Data factory JSONs/ folder.

```
{
    "name": "OnpremisesInputHDFSForHadoopHive",
    "properties": {
        "published": false,
        "type": "CustomDataset",
        "linkedServiceName": "OnpremisesHadoopCluster",
        "typeProperties": {
            "tableName": "partitionedcdrs",
            "partitionedBy": "yearno=${YEAR};monthno=${MONTH}"
        },
        "availability": {
            "frequency": "Day"
            "interval": 1
        },
        "external": true,
        "policy": {
            "executionPriorityOrder": "OldestFirst"
        }
    }
    }

    {
        "name": "OnpremisesOutputHDFSForHadoopHive",
        "properties": {
        "published": false,
        "type": "CustomDataset",
        "linkedServiceName": "OnpremisesHadoopCluster",
        "typeProperties": {
            "tableName": "callsummarybymonth",
            "partitionedBy": "yearno=${YEAR};monthno=${MONTH}"
        },
        "availability": {
            "frequency": "Day",
            "interval": 1
        }
    }
    }

    {
    "name": "TestHiveRunningOnHDPHiveInHDFS",
    "properties": {
        "description": "Test pipeline to run Hive script on an on-premises
 HDP cluster (Hive is in HDFS location)",
        "activities": [
            {
```

```
            "type": "HadoopHive",
            "typeProperties": {
                "runOnBehalf": "ambari-qa",
                "scriptPath": "/apps/falcon/adf-demo/demo.hql",
                "Year": "$$Text.Format('{0:yyyy}',SliceStart)",
                "Month": "$$Text.Format('{0:%M}',SliceStart)",
                "Day": "$$Text.Format('{0:%d}',SliceStart)"
            },
            "inputs": [
                {
                    "name": "OnpremisesInputHDFSForHadoopHive"
                }
            ],
            "outputs": [
                {
                    "name": "OnpremisesOutputHDFSForHadoopHive"
                }
            ],
            "policy": {
                "executionPriorityOrder": "OldestFirst",
                "timeout": "00:05:00",
                "concurrency": 1,
                "retry": 1
            },
            "scheduler": {
                "frequency": "Day",
                "interval": 1
            },
            "name": "HiveScriptOnHDPCluster",
            "linkedServiceName": "OnpremisesHadoopCluster"
        }
    ],
        "start": "2014-11-01T00:00:00Z",
        "end": "2014-11-02T00:00:00Z",
        "isPaused": false,
        "hubName": "hwkadftest1026_hub",
        "pipelineMode": "Scheduled"
  }
}
```

## 8.2. Configuring to Copy Files From an On-premises HDFS Store to Azure Blob Storage

You must configure access in your HDFS environment to be able to move data to and from the blob storage.

**Prerequisite Setup:**

Before you copy files, you must ensure the following:

- You created a Microsoft Storage Blob Container.
- You have the blob credentials available.

  For Apache Falcon replication jobs, HDFS requires Azure blob credentials to move data to and from the Azure blob storage.

1. Log in to Ambari at http://[`cluster ip`]:8080.
2. Select **HDFS** and click **Configs > Advanced**.
3. Expand the **Custom core-site** section and click **Add Property**.
4. Add the Azure credential as a key/value property.
   - Use the following format for the key, replacing *account_name* with the name of the Azure blob: **`fs.azure.account.key.`*`account_name`*`.blob.core.windows.net`**
   - Use the Azure blob account key for the value.
5. [(Optional) Take an HDFS checkpoint.]

   If a recent checkpoint does not exist, the NameNode(s) can take a very long time to start up.

   a. Login to the NameNode host.

   b. Put the NameNode in Safe Mode (read-only mode).

   ```
   sudo su hdfs -l -c 'hdfs dfsadmin -safemode enter'
   ```

   c. Once in Safe Mode, create a checkpoint.

   ```
   sudo su hdfs -l -c 'hdfs dfsadmin -saveNamespace'
   ```
6. Restart the relevant components from the Ambari UI.

   a. Click the **Services** tab and select **HDFS**.

   b. Click **Service Actions** > **Restart All**.

   c. Click **Confirm Restart All** > **OK**.

   d. Verify that the status is **Started** for HDFS, YARN, MapReduce2, Oozie, and Falcon.

   **Note**

   If you restart the components at the command line, restart them in the following order: HDFS, YARN, MapReduce2, Oozie, Falcon.
7. Test if you can access the Azure blob storage through HDFS.

   Example: hdfs dfs -ls -R wasb://`[container-name@azure-user]`.blob.core.windows.net/
8. You can now set up the on-premises input dataset, output dataset in Azure blob and a pipeline to copy the data with the replication activity. The sample JSON objects are shown below.

```
{
    "name": "OnpremisesInputHDFSForHadoopMirror",
    "properties": {
```

```
        "published": false,
        "type": "CustomDataset",
        "linkedServiceName": "OnpremisesHadoopCluster",
        "typeProperties": {
            "folderPath": "/apps/hive/warehouse/callsummarybymonth/yearno=
${YEAR}/monthno=${MONTH}"
        },
        "availability": {
            "frequency": "Day",
            "interval": 1
        },
        "external": true,
        "policy": {
            "executionPriorityOrder": "OldestFirst"
        }
    }
}

{
    "name": "AzureBlobDatasetForHadoopMirror",
    "properties": {
        "published": false,
        "type": "AzureBlob",
        "linkedServiceName": "AzureBlobStorage",
        "typeProperties": {
            "folderPath": "results/${YEAR}/${MONTH}",
            "format": {
                "type": "TextFormat"
            }
        },
        "availability": {
            "frequency": "Day",
            "interval": 1
        }
    }
}

{
    "name": "TestReplicate2Azure",
    "properties": {
        "description": "Test pipeline to mirror data on onpremises HDFS to
 azure",
        "activities": [
            {
                "type": "HadoopMirror",
                "typeProperties": {},
                "inputs": [
                    {
                        "name": "OnpremisesInputHDFSForHadoopMirror"
                    }
                ],
                "outputs": [
                    {
                        "name": "AzureBlobDatasetForHadoopMirror"
                    }
                ],
                "policy": {
                    "executionPriorityOrder": "OldestFirst",
                    "timeout": "00:05:00",
                    "concurrency": 1,
                    "retry": 1
                },
                "scheduler": {
                    "frequency": "Day",
                    "interval": 1
                },
                "name": "MirrorData 2Azure",
                "linkedServiceName": "OnpremisesHadoopCluster"
            }
        ],
        "start": "2014-11-01T00:00:00Z",
        "end": "2014-11-02T00:00:00Z",
        "isPaused": false,
        "hubName": "hwkadftest1026_hub",
        "pipelineMode": "Scheduled"
    }
}
```

# 9. Configuring for Disaster Recovery

From the Apache Falcon UI, you can configure your cluster for disaster recovery using HDFS replication or Hive table replication.

## 9.1. Prerequisites

Before creating the required entity definitions in Falcon, you must modify several configuration files. These files can be modified from Ambari or from the command line. To access the service properties in Ambari, click the Services menu and select the service you want, then click the Configs tab. For some services, you must also click the Advanced tab.

### 9.1.1. auth-to-local Setting

For Kerberos principals to be mapped correctly across clusters, changes must be made to the **hadoop.security.auth_to_local** setting in the `core-site.xml` file on both clusters. These entries should be made for the ambariqa, hbase, and hdfs principals.

For example, to add the principal mapping for *`CLUSTER02`* into *`CLUSTER01`* add the following lines into the **hadoop.security.auth_to_local** field in *`CLUSTER01`*. Note, these lines must be inserted before the wildcard line `RULE:[1:$1@$0](.*@HADOOP)s/@.*//`:

```
RULE:[1:$1@$0](ambari-qa-CLUSTER02@HADOOP)s/.*/ambari-qa/
RULE:[1:$1@$0](hbase-CLUSTER02@HADOOP)s/.*/hbase/
RULE:[1:$1@$0](hdfs-CLUSTER02@HADOOP)s/.*/hdfs/
```

#### Important

Ensure the **hadoop.security.auth_to_local** setting in the HDFS `core-site.xml` file is consistent across all clusters. Inconsistencies in rules for **hadoop.security.auth_to_local** can lead to issues with delegation token renewals.

## 9.2. Proxy User

To allow the Falcon Oozie workflows to query the primary status of both clusters, the Oozie server hosts must be whitelisted as proxy hosts to allow cross-site Oozie access. On both clusters, the **hadoop.proxyuser.oozie.hosts** setting in the HDFS `core-site.xml` must contain the hostnames of all the Oozie servers in both clusters. The hostnames of all the Oozie servers must also be added to the **hadoop.proxyuser.hive.hosts** because the Oozie server proxies as the Hive user during replication.

Also, to allow cross-site Apache Hive access for Hive replication, Hive and HCatalog hosts for both clusters must be whitelisted as proxy hosts. On both clusters, the **hadoop.proxyuser.hive.hosts** and **hadoop.proxyuser.hcat.hosts** settings in the HDFS `core-site.xml` must contain the hostnames of all the Hive Metastore and Hive Server2 hosts for Hive and WebHCat hosts for HCatalog.

You can change these settings through Ambari.

## 9.3. Nameservices

For cross-site DistCp to function, both clusters must be able to resolve the nameservice of the other cluster. In Ambari, modify the following properties in the **hdfs-site** section to add the appropriate cluster names.

```
dfs.internal.nameservices:CLUSTER01
dfs.client.failover.proxy.provider.CLUSTER02=org.apache.hadoop.hdfs.server.
namenode.ha.ConfiguredFailover c ,! ProxyProvider
dfs.ha.namenodes.CLUSTER02=nn1,nn2
dfs.namenode.http-address.CLUSTER02.nn1=namenode01.hostname:50070
dfs.namenode.http-address.CLUSTER02.nn2=namenode02.hostname:50070
dfs.namenode.https-address.CLUSTER02.nn1=namenode01.hostname:50470
dfs.namenode.https-address.CLUSTER02.nn2=namenode02.hostname:50470
dfs.namenode.rpc-address.CLUSTER02.nn1=namenode01.hostname:8020
dfs.namenode.rpc-address.CLUSTER02.nn2=namenode02.hostname:8020
```

## 9.4. HDFS and Hive Replication Using a Trigger File

Data replication for both Apache HDFS and Apache Hive can be managed through Apache Falcon. The workflows must be defined in both the primary and secondary clusters and failover is controlled by a trigger file in HDFS.

The workflows begin by looking for the file you identify as the trigger. The cluster that contains the trigger file is identified as the primary cluster. The data replication direction automatically reverses when the primary status trigger file is switched to the alternate cluster. The DistCp jobs used for the replication run in the secondary cluster and pull replicated data from the primary. If both or neither of the clusters are marked as primary, or the target file cannot be accessed, the workflows fail. You can configure alerts to be sent for the workflow failures. If the workflow fails, examine the Oozie logs for possible causes.

### 9.4.1. Hive Replication Properties

To allow Hive replication, the following properties must be set in the `hive-site.xml` file in both clusters. This can be done through Ambari by adding them to **Custom hive-site** properties:

```
hive.metastore.event.listeners=org.apache.hive.hcatalog.listener.
DbNotificationListener
hive.metastore.dml.events=true
hive.metastore.event.db.listener.timetolive=432000s
hive.exim.strict.repl.tables=true
```

**Note**

Currently Falcon Hive replication only supports Hiveserver2 on binary transport mode.

### 9.4.2. Example of a Workflow for HDFS DR Using a Trigger File

The Oozie workflow is used by the Falcon process to schedule DistCp replication. The workflow definition file should be placed in both clusters at the location referenced in the process definitions.

The following example is a modified version of the stock Falcon HDFS Mirror workflow but includes the primary site trigger file checks. The Falcon process references this trigger file, which must be placed in HDFS in both clusters, for example, in `/apps/data-mirroring/workflows/hdfs-replication-workflow-ha.xml`.

```
<workflow-app xmlns='uri:oozie:workflow:0.3' name='falcon-
dr-fs-workflow'> <start to='check_primary_this'/> <!
Replication action --> <decision name="check_primary_this">
<switch> <case to="check_not_primary_that">
${fs:exists(concat(primaryClusterFS,hdfsTriggerFile))}</
case> <case to="check_primary_that">${!
fs:exists(concat(primaryClusterFS,hdfsTriggerFile))}</
case> <default to="failed_to_check_primary"/> </switch>
</decision> <decision name="check_not_primary_that">
<switch> <case to="dr-replication">${!
fs:exists(concat(secondaryClusterFS,hdfsTriggerFile))}</
case> <case to="failed_two_primaries_defined">
${fs:exists(concat(secondaryClusterFS,hdfsTriggerFile) c ,! )}</
case> <default to="failed_to_check_secondary"/> </switch> </
decision> <decision name="check_primary_that"> <switch> <case
to="end">
${fs:exists(concat(secondaryClusterFS,hdfsTriggerFile))}</
case> <case to="failed_no_primaries_defined">${!
fs:exists(concat(secondaryClusterFS,hdfsTriggerFile) c ,! )}</
case> <default to="failed_to_check_secondary"/> </switch> </
decision> <action name="dr-replication"> <java> <job-tracker>
${jobTracker}</job-tracker> <name-node>${nameNode}</name-
node> <configuration> <property> <!-- hadoop 2 parameter --
> <name>oozie.launcher.mapreduce.job.user.classpath.first</
name> <value>true</value> </property> <property>
<name>mapred.job.queue.name</name> <value>${queueName}</value> </
property> <property> <name>oozie.launcher.mapred.job.priority</
name> <value>${jobPriority}</value> </property> <property>
<name>oozie.use.system.libpath</name> <value>true</value> </
property> <property> <name>oozie.action.sharelib.for.java</
name> <value>distcp</value> </property> <property>
<name>oozie.launcher.oozie.libpath</name> <value>
${wf:conf("falcon.libpath")}</value> </property> <property>
<name>oozie.launcher.mapreduce.job.hdfs-servers</name>
<value>${primaryClusterFS},${secondaryClusterFS}</
value> </property> </configuration> <main-
class>org.apache.falcon.replication.FeedReplicator</main-
class> <arg>-Dmapred.job.queue.name=${queueName}</arg> <arg>-
Dmapred.job.priority=${jobPriority}</arg> <arg>-maxMaps</arg>
```

```
<arg>${distcpMaxMaps}</arg> <arg>-mapBandwidth</arg> <arg>
${distcpMapBandwidth}</arg> <arg>-sourcePaths</arg> <arg>
${primaryClusterFS}${replicatedDirectory}</arg> <arg>-targetPath</
arg> <arg>${secondaryClusterFS}${replicatedDirectory}</
arg> <arg>-falconFeedStorageType</arg> <arg>FILESYSTEM</
arg> <arg>-availabilityFlag</arg> <arg>${availabilityFlag ==
'NA' ? "NA" : availabilityFlag}</arg> <arg>-counterLogDir</
arg> <arg>${logDir}/job-${nominalTime}/${srcClusterName
== 'NA' ? '' : srcClusterName}</arg> </java> <ok
to="end"/> <error to="copy_failed"/> </action> <kill
name="failed_to_check_primary"> <message>Failed to check Trigger
file at: ${primaryClusterFS}${hdfsTriggerFile}</message> </kill>
<kill name="failed_to_check_secondary"> <message>Failed to check
Trigger file at: ${secondaryClusterFS}${hdfsTriggerFile}</message>
</kill> <kill name="failed_two_primaries_defined"> <message>Two
primaries are defined at: ${primaryClusterFS}${hdfsTriggerFile}
and ,! ${secondaryClusterFS}${hdfsTriggerFile}</message> </kill>
<kill name="failed_no_primaries_defined"> <message>No primaries
are defined at either: ${primaryClusterFS}${hdfsTriggerFile}
and ,! ${secondaryClusterFS}${hdfsTriggerFile}</message> </kill>
<kill name="copy_failed"> <message>Workflow action failed, error
message[${wf:errorMessage(wf:lastErrorNode())}]</message> </kill>
<end name="end"/> </workflow-app>
```

## 9.4.3. Example of a Workflow for Hive DR Using a Trigger File

This Falcon Hive replication recipe is a modified version of the stock Falcon Hive Mirror workflow that includes the primary site trigger file checks. The Oozie workflow definition file should be placed in both clusters at the location referenced in the Hive process definitions, for example at `/apps/data-mirroring/workflows/hive-disaster-recovery-ha-secure-workflow.xml`.

```
<workflow-app xmlns='uri:oozie:workflow:0.3' name='falcon-dr-hive-workflow'>
<credentials>
<credential name='hive_src_credentials' type='hcat'>
<property>
<name>hcat.metastore.uri</name>
<value>${primaryHiveMetastore}</value>
</property>
<property>
<name>hcat.metastore.principal</name>
<value>hive/_HOST@${kerberosRealm}</value>
</property>
</credential>
<credential name='hive_tgt_credentials' type='hcat'>
<property>
<name>hcat.metastore.uri</name>
<value>${secondaryHiveMetastore}</value>
</property>
<property>
<name>hcat.metastore.principal</name>
<value>hive/_HOST@${kerberosRealm}</value>
```

```
</property>
</credential>
<credential name="hive2_src_credentials" type="hive2">
<property>
<name>hive2.server.principal</name>
<value>hive/_HOST@${kerberosRealm}</value>
</property>
<property>
<name>hive2.jdbc.url</name>
<value>jdbc:${primaryHiveServer2}/${replicationDatabase}</value>
</property>
</credential>
<credential name="hive2_tgt_credentials" type="hive2">
<property>
<name>hive2.server.principal</name>
<value>hive/_HOST@${kerberosRealm}</value>
</property>
<property>
<name>hive2.jdbc.url</name>
<value>jdbc:${secondaryHiveServer2}/${replicationDatabase}</value>
</property>
</credential>
</credentials>
<start to='check_primary_this'/>
<!-- Replication action -->
<decision name="check_primary_this">
<switch>
<case to="check_not_primary_that">${fs:exists(concat(primaryClusterFS,
hiveTriggerFile))}</case>
<case to="check_primary_that">${!fs:exists(concat(primaryClusterFS,
hiveTriggerFile))}</case>
<default to="failed_to_check_primary"/>
</switch>
</decision>
<decision name="check_not_primary_that">
<switch>
<case to="hivedr-replication">${!fs:exists(concat(secondaryClusterFS,
hiveTriggerFile))}</case>
<case to="failed_two_primaries_defined">${fs:exists(concat(secondaryClusterFS,
hiveTriggerFile) c
,! )}</case>
<default to="failed_to_check_secondary"/>
</switch>
</decision>
<decision name="check_primary_that">
<switch>
<case to="end">${fs:exists(concat(secondaryClusterFS,hiveTriggerFile))}</case>
<case to="failed_no_primaries_defined">${!fs:exists(concat(secondaryClusterFS,
hiveTriggerFile) c
,! )}</case>
<default to="failed_to_check_secondary"/>
</switch>
</decision>
<action name="hivedr-replication" cred="hive_tgt_credentials">
<java>
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
<configuration>
<property> <!-- hadoop 2 parameter -->
<name>oozie.launcher.mapreduce.job.user.classpath.first</name>
```

```
<value>true</value>
</property>
<property>
<name>mapred.job.queue.name</name>
<value>${queueName}</value>
</property>
<property>
<name>oozie.launcher.mapred.job.priority</name>
<value>${jobPriority}</value>
</property>
<property>
<name>oozie.use.system.libpath</name>
<value>true</value>
</property>
<property>
<name>oozie.action.sharelib.for.java</name>
<value>distcp,hive,hive2,hcatalog</value>
</property>
<property>
<name>oozie.launcher.mapreduce.job.hdfs-servers</name>
<value>${primaryClusterFS},${secondaryClusterFS}</value>
</property>
<property>
<name>mapreduce.job.hdfs-servers</name>
<value>${primaryClusterFS},${secondaryClusterFS}</value>
</property>
</configuration>
<main-class>org.apache.falcon.hive.HiveDRTool</main-class>
<arg>-Dmapred.job.queue.name=${queueName}</arg>
<arg>-Dmapred.job.priority=${jobPriority}</arg>
<arg>-falconLibPath</arg>
<arg>${wf:conf("falcon.libpath")}</arg>
<arg>-sourceCluster</arg>
<arg>${primaryClusterFS}</arg>
<arg>-sourceMetastoreUri</arg>
<arg>${primaryHiveMetastore}</arg>
<arg>-sourceHiveServer2Uri</arg>
<arg>${primaryHiveServer2}</arg>
<arg>-sourceDatabase</arg>
<arg>${replicationDatabase}</arg>
<arg>-sourceTable</arg>
<arg>${replicationTables}</arg>
<arg>-sourceStagingPath</arg>
<arg>${primaryStagingPath}</arg>
<arg>-sourceNN</arg>
<arg>${primaryClusterFS}</arg>
<arg>-sourceNNKerberosPrincipal</arg>
<arg>nn/_HOST@${kerberosRealm}</arg>
<arg>-sourceHiveMetastoreKerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-sourceHive2KerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-targetCluster</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-targetMetastoreUri</arg>
<arg>${secondaryHiveMetastore}</arg>
<arg>-targetHiveServer2Uri</arg>
<arg>${secondaryHiveServer2}</arg>
<arg>-targetStagingPath</arg>
<arg>${secondaryStagingPath}</arg>
```

```
<arg>-targetNN</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-targetNNKerberosPrincipal</arg>
<arg>nn/_HOST@${kerberosRealm}</arg>
<arg>-targetHiveMetastoreKerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-targetHive2KerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-maxEvents</arg>
<arg>${maxEvents}</arg>
<arg>-clusterForJobRun</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-clusterForJobRunWriteEP</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-clusterForJobNNKerberosPrincipal</arg>
<arg>nn/_HOST@${kerberosRealm}</arg>
<arg>-drJobName</arg>
<arg>hivereplicationdr-${nominalTime}</arg>
<arg>-executionStage</arg>
<arg>lastevents</arg>
</java>
<ok to="export-dr-replication"/>
<error to="fail"/>
</action>
<!-- Export Replication action -->
<action name="export-dr-replication" cred="hive_src_credentials,
hive2_src_credentials">
<java>
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
<configuration>
<property> <!-- hadoop 2 parameter -->
<name>oozie.launcher.mapreduce.job.user.classpath.first</name>
<value>true</value>
</property>
<property>
<name>mapred.job.queue.name</name>
<value>${queueName}</value>
</property>
<property>
<name>oozie.launcher.mapred.job.priority</name>
<value>${jobPriority}</value>
</property>
<property>
<name>oozie.use.system.libpath</name>
<value>true</value>
</property>
<property>
<name>oozie.action.sharelib.for.java</name>
<value>distcp,hive,hive2,hcatalog</value>
</property>
<property>
<name>oozie.launcher.mapreduce.job.hdfs-servers</name>
<value>${primaryClusterFS},${secondaryClusterFS}</value>
</property>
<property>
<name>mapreduce.job.hdfs-servers</name>
<value>${primaryClusterFS},${secondaryClusterFS}</value>
</property>
</configuration>
```

```
<main-class>org.apache.falcon.hive.HiveDRTool</main-class>
<java-opts>-Dlog4j.configuration=file:///tmp/log4j.properties</java-opts>
<arg>-Dlog4j.configuration=file:///tmp/log4j.properties</arg>
<arg>-Dmapred.job.queue.name=${queueName}</arg>
<arg>-Dmapred.job.queue.name=${queueName}</arg>
<arg>-Dmapred.job.priority=${jobPriority}</arg>
<arg>-falconLibPath</arg>
<arg>${wf:conf("falcon.libpath")}</arg>
<arg>-replicationMaxMaps</arg>
<arg>${replicationMaxMaps}</arg>
<arg>-distcpMaxMaps</arg>
<arg>${distcpMaxMaps}</arg>
<arg>-sourceCluster</arg>
<arg>${primaryClusterFS}</arg>
<arg>-sourceMetastoreUri</arg>
<arg>${primaryHiveMetastore}</arg>
<arg>-sourceHiveServer2Uri</arg>
<arg>${primaryHiveServer2}</arg>
<arg>-sourceDatabase</arg>
<arg>${replicationDatabase}</arg>
<arg>-sourceTable</arg>
<arg>${replicationTables}</arg>
<arg>-sourceStagingPath</arg>
<arg>${primaryStagingPath}</arg>
<arg>-sourceNN</arg>
<arg>${primaryClusterFS}</arg>
<arg>-sourceNNKerberosPrincipal</arg>
<arg>nn/_HOST@${kerberosRealm}</arg>
<arg>-sourceHiveMetastoreKerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-sourceHive2KerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-targetCluster</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-targetMetastoreUri</arg>
<arg>${secondaryHiveMetastore}</arg>
<arg>-targetHiveServer2Uri</arg>
<arg>${secondaryHiveServer2}</arg>
<arg>-targetStagingPath</arg>
<arg>${secondaryStagingPath}</arg>
<arg>-targetNN</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-targetNNKerberosPrincipal</arg>
<arg>nn/_HOST@${kerberosRealm}</arg>
<arg>-targetHiveMetastoreKerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-targetHive2KerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-maxEvents</arg>
<arg>${maxEvents}</arg>
<arg>-distcpMapBandwidth</arg>
<arg>${distcpMapBandwidth}</arg>
<arg>-clusterForJobRun</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-clusterForJobRunWriteEP</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-clusterForJobNNKerberosPrincipal</arg>
<arg>nn/_HOST@${kerberosRealm}</arg>
<arg>-drJobName</arg>
<arg>hivereplicationdr-${nominalTime}</arg>
```

```
<arg>-executionStage</arg>
<arg>export</arg>
</java>
<ok to="import-dr-replication"/>
<error to="fail"/>
</action>
<!-- Import Replication action -->
<action name="import-dr-replication" cred="hive_tgt_credentials,
hive2_tgt_credentials">
<java>
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
<configuration>
<property> <!-- hadoop 2 parameter -->
<name>oozie.launcher.mapreduce.job.user.classpath.first</name>
<value>true</value>
</property>
<property>
<name>mapred.job.queue.name</name>
<value>${queueName}</value>
</property>
<property>
<name>oozie.launcher.mapred.job.priority</name>
<value>${jobPriority}</value>
</property>
<property>
<name>oozie.use.system.libpath</name>
<value>true</value>
</property>
<property>
<name>oozie.action.sharelib.for.java</name>
<value>distcp,hive,hive2,hcatalog</value>
</property>
<property>
<name>oozie.launcher.mapreduce.job.hdfs-servers</name>
<value>${primaryClusterFS},${secondaryClusterFS}</value>
</property>
<property>
<name>mapreduce.job.hdfs-servers</name>
<value>${primaryClusterFS},${secondaryClusterFS}</value>
</property>
</configuration>
<main-class>org.apache.falcon.hive.HiveDRTool</main-class>
<arg>-Dmapred.job.queue.name=${queueName}</arg>
<arg>-Dmapred.job.priority=${jobPriority}</arg>
<arg>-falconLibPath</arg>
<arg>${wf:conf("falcon.libpath")}</arg>
<arg>-replicationMaxMaps</arg>
<arg>${replicationMaxMaps}</arg>
<arg>-distcpMaxMaps</arg>
<arg>${distcpMaxMaps}</arg>
<arg>-sourceCluster</arg>
<arg>${primaryClusterFS}</arg>
<arg>-sourceMetastoreUri</arg>
<arg>${primaryHiveMetastore}</arg>
<arg>-sourceHiveServer2Uri</arg>
<arg>${primaryHiveServer2}</arg>
<arg>-sourceDatabase</arg>
<arg>${replicationDatabase}</arg>
<arg>-sourceTable</arg>
```

```
<arg>${replicationTables}</arg>
<arg>-sourceStagingPath</arg>
<arg>${primaryStagingPath}</arg>
<arg>-sourceNN</arg>
<arg>${primaryClusterFS}</arg>
<arg>-sourceNNKerberosPrincipal</arg>
<arg>nn/_HOST@${kerberosRealm}</arg>
<arg>-sourceHiveMetastoreKerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-sourceHive2KerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-targetCluster</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-targetMetastoreUri</arg>
<arg>${secondaryHiveMetastore}</arg>
<arg>-targetHiveServer2Uri</arg>
<arg>${secondaryHiveServer2}</arg>
<arg>-targetStagingPath</arg>
<arg>${secondaryStagingPath}</arg>
<arg>-targetNN</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-targetNNKerberosPrincipal</arg>
<arg>nn/_HOST@${kerberosRealm}</arg>
<arg>-targetHiveMetastoreKerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-targetHive2KerberosPrincipal</arg>
<arg>hive/_HOST@${kerberosRealm}</arg>
<arg>-maxEvents</arg>
<arg>${maxEvents}</arg>
<arg>-distcpMapBandwidth</arg>
<arg>${distcpMapBandwidth}</arg>
<arg>-clusterForJobRun</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-clusterForJobRunWriteEP</arg>
<arg>${secondaryClusterFS}</arg>
<arg>-clusterForJobNNKerberosPrincipal</arg>
<arg>nn/_HOST@${kerberosRealm}</arg>
<arg>-drJobName</arg>
<arg>hivereplicationdr-${nominalTime}</arg>
<arg>-executionStage</arg>
<arg>import</arg>
</java>
<ok to="end"/>
<error to="fail"/>
</action>
<kill name="fail">
<message>
Workflow action failed, error message[${wf:errorMessage(wf:lastErrorNode())}]
</message>
</kill>
<kill name="failed_to_check_primary">
<message>Failed to check Trigger file at:
 ${primaryClusterFS}${hiveTriggerFile}</message>
</kill>
<kill name="failed_to_check_secondary">
<message>Failed to check Trigger file at:
 ${secondaryClusterFS}${hiveTriggerFile}</message>
</kill>
<kill name="failed_two_primaries_defined">
```

```
<message>Two primaries are defined at: ${primaryClusterFS}${hiveTriggerFile}
 and
,! ${secondaryClusterFS}${hiveTriggerFile}</message>
</kill>
<kill name="failed_no_primaries_defined">
<message>No primaries are defined at either:
 ${primaryClusterFS}${hiveTriggerFile} and
,! ${secondaryClusterFS}${hiveTriggerFile}</message>
</kill>
<kill name="copy_failed">
<message>Workflow action failed, error
 message[${wf:errorMessage(wf:lastErrorNode())}]</message>
</kill>
<end name="end"/>
</workflow-app>
```

## 9.4.4. Creating the Falcon Cluster Definition

The Falcon definitions for both clusters must be present in both Falcon servers. The definitions can be submitted either through the Falcon UI or from the CLI by submitting an XML definition. All end-points can support a highly-available configuration. Properties for a DR cluster definition include the following:

| | |
|---|---|
| read/write | Give the HDFS nameservice to utilise Namenode HA. |
| execute | When using ResourceManager (RM) HA, giving a single RM host allows RM HA for Falcon as the alternate RM host is picked up by Oozie from the `yarn-site.xml` file on the Oozie server. |
| workflow | When using Oozie HA with a load-balancer, the load-balancer address can be used for Oozie HA in Falcon. |
| hive metastore | A list of metastore addresses can be given in the Falcon cluster definition. |
| hiveserver2 | A Zookeeper connect string can be provided for the HiveServer2 JDBC connect string in the cluster definition to enable HS2 HA. |

## 9.4.5. Creating the Falcon Process Definition for HDFS DR Using a Trigger File

A Falcon process should be defined for each HDFS folder that is to be replicated in each cluster. It can be defined either through the Falcon UI by creating a new process or submitting the definition through the Falcon CLI.

You must submit two process definitions: One for `cluster01`, to be used when cluster one is secondary, and one for `cluster02`, when cluster two is secondary. You should schedule the process for each cluster and have them both ready to run as secondary, regardless of the current cluster status. If a failure occurs, you manually change cluster status and the replication jobs run based on the status of the cluster as dictated by the trigger file. See Section 9.4.8, “Reversing Primary Cluster Status” [67] for instructions.

Note that this is a pull replication, so the **<cluster name>** attribute should be the *secondary* cluster.

Following is an example of a process definition for the `/HADOOP/drtest` folder.

### Example 9.1. Example of a process definition for the /HADOOP/drtest folder

The values of primaryTriggerFile and secondaryTriggerFile are the trigger files used to control primary and secondary status of the clusters.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<process name="falcon-dr-hdfs-drtest" xmlns="uri:falcon:process:0.1">
<tags>_falcon_mirroring_type=HDFS</tags>
<clusters>
<cluster name="CLUSTER01">
<validity start="2016-08-17T01:12Z" end="2017-08-17T01:12Z"/>
</cluster>
</clusters>
<parallel>1</parallel>
<order>LAST_ONLY</order>
<frequency>minutes(5)</frequency>
<timezone>GMT+00:00</timezone>
<properties>
<property name="oozie.wf.subworkflow.classpath.inheritance" value="true"/>
<property name="distcpMaxMaps" value="5"/>
<property name="distcpMapBandwidth" value="100"/>
<property name="replicatedDirectory" value="/HADOOP/drtest"/>
<property name="hdfsTriggerFile" value="/cluster_status/hdfs_primary"/>
<property name="queueName" value="default"/>
<property name="jobPriority" value="HIGH"/>
</properties>
<workflow name="falcon-dr-fs-workflow" engine="oozie"
,! path="/apps/data-mirroring/workflows/hdfs-replication-workflow-ha.xml" lib=
""/>
<retry policy="periodic" delay="minutes(30)" attempts="3"/>
<ACL owner="hdfs" group="hadoop_admin" permission="0775"/>
</process>
```

## 9.4.6. Creating the Falcon Process Definition for Hive DR Using a Trigger File

A Falcon process should be defined for each group of tables in a database to be replicated in each cluster. Typically, there is one Falcon process per project. The process can be defined either through the Falcon UI or by submitting the definition from the Falcon CLI.

You must submit two process definitions: One for `cluster01`, to be used when cluster one is secondary, and one for `cluster02`, when cluster two is secondary. You should schedule the process for each cluster and have them both ready to run as secondary, regardless of the current cluster status. If a failure occurs, you manually change cluster status and the replication jobs run based on the status of the cluster as dictated by the trigger file. See Section 9.4.8, “Reversing Primary Cluster Status” [67] for instructions.

Because this is a pull replication, the **<cluster name>** attribute should be the secondary cluster.

Hive replication can be defined at table level in a given database, however a comma-separated list of tables can be given or a wildcard can be used to replicate all tables in a database. For example, to replicate all tables in the `reptest` database, you might use the following configuration: **<property name="replicationTables" value="*"/> <property name="replicationDatabase" value="reptest"/>**

### Example 9.2. Example of a process definition for the a and b tables in the reptest database

The values of primaryTriggerFile and secondaryTriggerFile are the trigger files used to control primary and secondary status of the clusters.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<process name="falcon-dr-hive-reptest" xmlns="uri:falcon:process:0.1">
<tags>_falcon_mirroring_type=HIVE</tags>
<clusters>
<cluster name="CLUSTER01">
<validity start="2016-08-25T18:23Z" end="2018-08-01T18:23Z"/>
</cluster>
</clusters>
<parallel>1</parallel>
<order>LAST_ONLY</order>
<frequency>minutes(5)</frequency>
<timezone>GMT+00:00</timezone>
<properties>
<property name="oozie.wf.subworkflow.classpath.inheritance" value="true"/>
<property name="hiveTriggerFile" value="/cluster_status/hive_primary"/>
<property name="queueName" value="default"/>
<property name="jobPriority" value="HIGH"/>
<property name="distcpMaxMaps" value="1"/>
<property name="distcpMapBandwidth" value="100"/>
<property name="primaryStagingPath" value="/apps/falcon/cluster-primary/
staging"/>
<property name="secondaryStagingPath" value="/apps/falcon/cluster-backup/
staging"/>
<property name="replicationTables" value="a,b"/>
<property name="replicationDatabase" value="reptest"/>
<property name="maxEvents" value="-1"/>
<property name="replicationMaxMaps" value="5"/>
</properties>
<workflow name="falcon-dr-hive-workflow" engine="oozie"
,! path="/apps/data-mirroring/workflows/hive-disaster-recovery-ha-secure-
workflow.xml" lib=""/>
<retry policy="periodic" delay="minutes(30)" attempts="3"/>
<ACL owner="hdfs" group="hadoop_admin" permission="0775"/>
</process>
```

## 9.4.7. Configuring Alerts

You can monitor HDFS replication and create and modify alert notifications through Ambari. HDFS checks the status of the Falcon processes responsible for the replication. See "Monitoring and Alerts" in the *Ambari Users Guide* for information and instructions about managing and customizing alerts.

### Note

Ensure that your version of Falcon is able to renew Kerberos tickets properly, as the alert mechanism relies of Falcon’s ability to query Oozie.

## 9.4.8. Reversing Primary Cluster Status

To perform a replication failover, you must reverse the HDFS primary trigger file. Guards are used in the workflow to protect against multiple clusters marked as primary.

If the trigger file is not located in the directory that you identified in the process definitions, or the file is not accessible for any reason, failover does not complete.

**Context**

The following commands can be executed in any order.

1. Remove the trigger file from the primary cluster.

   ```
   hdfs dfs -rm hdfs://CLUSTER01HA/cluster_status/hdfs_primary
   ```

2. Create an empty trigger file on the cluster you want to identify as the new primary.

   ```
   hdfs dfs -touchz hdfs://CLUSTER02HA/cluster_status/hdfs_primary
   ```

# 10. Using Advanced Falcon Features

## 10.1. Locating and Managing Entities

There are two methods you can use to locate entities in Falcon. You can use the Falcon web UI search functionality to find feed, process, or mirror entities that are defined for the cluster. To display a list of cluster entities, you use the Cluster icon. After locating an entity, you can perform actions on them or view information about them.

Available data sources can only be viewed from the Datasource field on the Create New Feed page.

**Steps**

1. Type your query in the **Search** field, and press **Enter**.

   **Note**

   You can use Search to find feed, process, and mirror entities, but not cluster or data source entities. See Step 3 to locate cluster entities. Available data sources can only be viewed from the Datasource field on the Create New Feed page.

   You can filter the entity search 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.

   **Tip**

   To see all feed, process, and mirror entities, search using an asterisk (*) for the entity name.

| Filter | Description |
|---|---|
| Name | Can be entered as a subsequence of the entity name (feed, process, or mirror name). Not case sensitive. The characters entered must be in the same order as the original sequence in the entity name.<br><br>For example:<br><br>• "sample1" matches the entity named "SampleFeed1-2"<br><br>• "mhs" matches the entity named "New-My-Hourly-Summary"<br><br>Searching on the asterisk (*) character displays all feed, process, and mirror entities. |
| 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 feed, process, and mirror. The Falcon search UI infers the type filter automatically. For example, to add a "process" filter type, enter **process** in the search field, and then choose **type:process** from the hints offered in the UI as shown in the previous screen capture. |

2. In the search results, you can do the following:

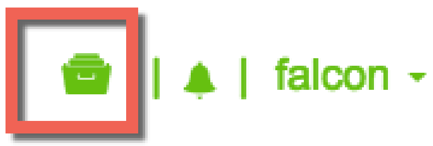
- Display the actions you can perform on the entity by selecting the checkbox next to the entity name.

  You can schedule, resume, pause, edit, copy, delete, or download the entity definition XML for a feed, process, or mirror.

- Display the instances and property details of the entity by clicking the entity name.

3. Click the Clusters icon to view a list of cluster entities.

   **Figure 10.1. Clusters icon**

4. In the search results, you can do the following:

   - Display the actions you can perform on the entity by selecting the check-box next to the cluster name.

     You can delete a cluster entity or download the entity definition XML.

     You can edit the cluster entity if your Falcon service is in safemode.

   - Display the properties of the cluster entity by clicking the cluster name.

## 10.2. Accessing File Properties from Ambari

You can set properties for files such as `core-site.xml`, `oozie-site.xml`, `hive-site.xml`, and so forth, from the Ambari UI.

You can access the properties by performing the following steps:

1. Log in to Ambari at http://[**`cluster ip`**]:8080.
2. Click **Service** and select the service that you want to modifiy.

   A page with details about the service displays.

3. On the service details page, click **Configs**, then click **Advanced**, if available.

   An Advanced page is not available for all services.

4. Click the section name for the **Custom** or **Advanced** properties to expand the section.
5. To add a property to the list, scroll to the bottom of properties and click **Add Property**.

   The **Add Property** option is available only for some configuration files.

## 10.3. Enabling Transparent Data Encryption

Falcon provides transparent data encryption (TDE) for data at rest, such as databases. TDE encryption is supported in Falcon on HDFS, Hive, and snapshot mirroring jobs.

You can enable the TDE encryption property in the following ways:

- Select the TDE Encryption option when creating a mirror in the Falcon web UI.
- Add the property `tdeEncryptionEnabled=true` directly to the properties file extensions in a `<workflow>.xml` file.
- Add `<arg>skipChecksum</arg>` and `<arg>true</arg>` to a `<properties>.json` file.

TDE encryption performs a checksum to determine if changes have been made to the data. If the checksum value is different, HDP performs a replication. The encryptions key can be different in the source and target clusters, which would result in a modified checksum value, so the checksum might indicate changes even if the data has not changed. If you skip the checksum, HDP compares individual blocks to see if replication is needed, which can increase the time to complete a mirror job.

## 10.4. Putting Falcon in Safe Mode

Safe mode (read-only mode) is useful when you want to prevent Falcon users from scheduling entities in the workflow engine, such as when upgrading Hadoop clusters or updating Falcon cluster entities. When in safe mode, users can only perform limited operations. For example, you cannot perform submit, schedule, edit, delete, or resume operations on entities. For further detail, see the Falcon Community Documentation.

You must be superuser or administrative (admin) user to set safe mode. A superuser is the user who owns the Falcon process or belongs to a group specified in the Falcon startup property `falcon.security.authorization.superusergroup`. An admin user is the user listed under the Falcon startup property `falcon.security.authorization.admin.users`, or they belong to a group listed under the startup property `falcon.security.authorization.admin.groups`.

If you want to modify a cluster entity, you must be in safe mode.

1. Put the Falcon server in safe mode:

   ```
   /bin/falcon admin -setsafemode true
   ```

2. Take the Falcon server out of safe mode:

   ```
   /bin/falcon admin -setsafemode false
   ```

## 10.5. 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 10.1. 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 |
| 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 |

| Entity Type | Action | Returns Success/Failure |
|---|---|---|
| Process | Process re-run due to late feed event | No |
| N/A | Transaction rollback failed event | No |

## 10.6. 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 and 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 a late data check is done. The 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>
   ```

## 10.7. 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. You set a retention policy in the feed entity.

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.

## 10.8. 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. The retry policy is set on the process entity.

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.

## 10.9. Enabling Email Notifications

You can enable email notifications in feed entities and process entities. When email notifications are enabled, an email is sent to the specified email address when the scheduled feed or process instance completes. Email notifications can be specified in feed or process entities.

You can set email notifications in feed and process entities from the web UI. This can be done when initially creating an entity, or you can edit an existing entity to add or modify email notifications.

### Setting Email Notifications from the CLI

To set email notifications, add the following to the feed or process entity:

```
<process name="<process_name>
     ...
     <notification type="email" to="jdoe@example.com, sjones@company.com"/>
     ...
</process>
```

Where `type` specifies the type of notification. Currently, only the `email` notification type is supported. The `to` attribute specifies where the notification is to be sent. In the case of email notifications, specify the email address where you want notifications sent for the `to` attribute. Multiple recipients can be specified as a comma-separated list of email addresses as shown in the previous example. The <`notification`> tag must be placed before the <ACL> tag.

If you are using email notification in a Hive or HDFS disaster recovery configuration, you must add the following properties to the Falcon server-side extensions file:

- jobNotificationType=email
- jobNotificationReceivers=<address-of-email-receiver>

  The email addresses are entered as a comma-separated list.

Falcon email notifications require SMTP server configurations to be defined in the Falcon `startup.properties` file that is located in the `FALCON_HOME/conf` directory. After setting the values, restart the Falcon service.

Falcon does not check for multiple versions of properties in the properties file. If multiple versions of a property exist, Falcon uses the last one in the file.

Configure the following startup properties for email notifications:

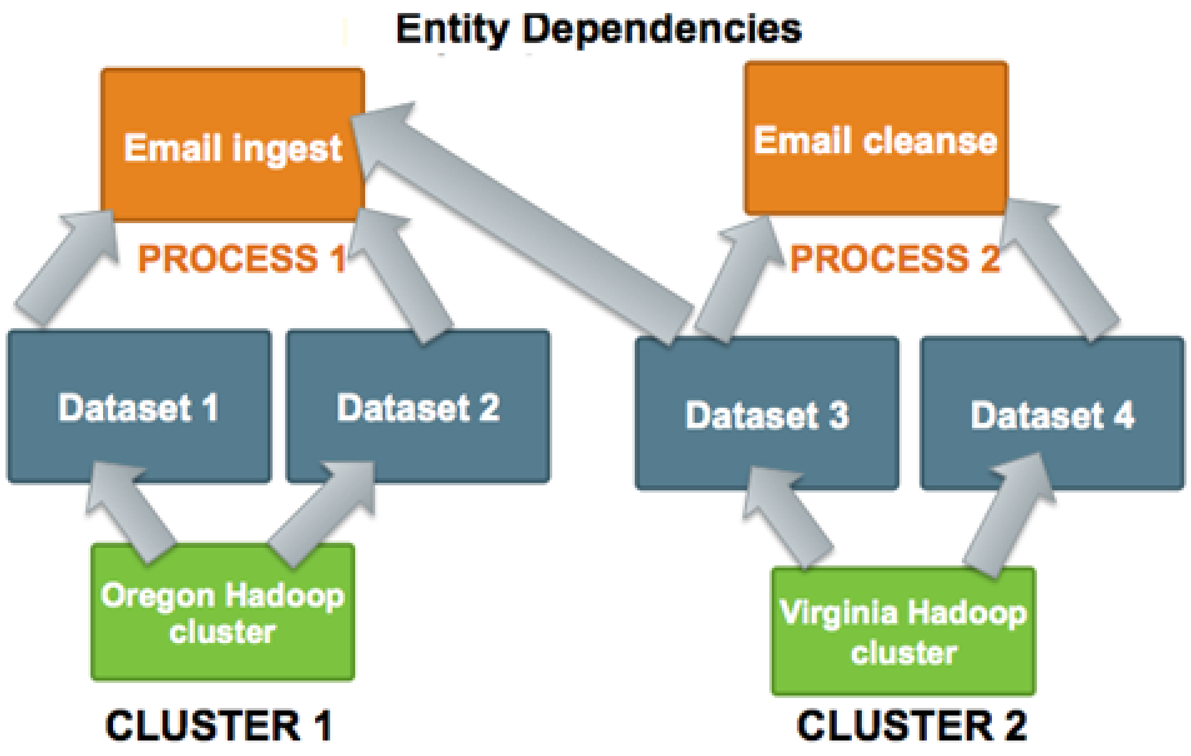
### Table 10.2. Email Notifications Startup Properties

| Property | Description | Default Values |
|---|---|---|
| **falcon.email.smtp.host** | Name of the host where the SMTP server can be found. | `localhost` |
| **falcon.email.smtp.port** | The SMTP server host port to connect to. | 25 |
| **falcon.email.from.address** | The "From:" address used for all notification emails. | `falcon@localhost` |
| **falcon.email.smtp.auth** | Indicates whether user sending the email is authenticated. Boolean value (true \| false) | `false` |
| **falcon.email.smtp.user** | If authentication is enabled, this property specifies the username that is used to log in. | none |
| **falcon.email.smtp.password** | If authentication is enabled, the username's password that is used to authenticate the user. | none |
| **monitoring.plugins** | Ensure that the email notification plugin is listed for this property to enable email notifications.<br><br>For example:<br><br>**`org.apache.falcon.plugin.`**<br>**`EmailNotificationPlugin`**`,`<br>`org.apache.falcon.plugin.`<br>`DefaultMonitoringPlugin` | none |

## 10.10. 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).

**Figure 10.2. Entity dependencies in Falcon**



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. For further information about dependencies and cross-entity validations, see the Falcon Community Documentation

## 10.11. Viewing Dependencies

The Falcon web UI provides dependency viewing for clusters, datasets, and processes that shows lineage in a graphical format. You can view the relationships between dependencies as a graph to determine requirements for removal.

You can access the dependencies graph from the entity lists.

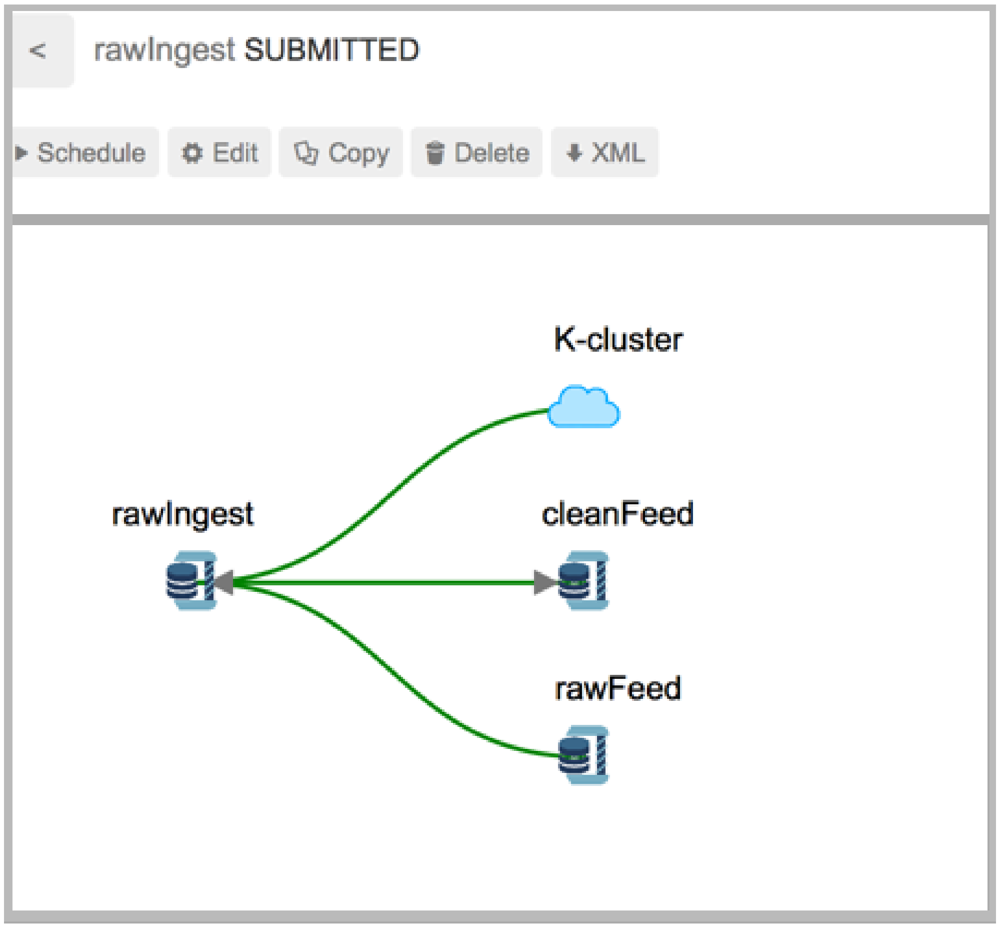
To view cluster dependencies:

1. Click the Clusters icon.
2. In the list that displays, click on the name of a cluster.

To view feed or process entities, or mirror jobs:

1. Enter a term in the Search field.
2. In the search results, click on the name of an entity or mirror job.

**Figure 10.3. Graph view of dependencies**



# 11. Using Apache Sqoop to Transfer Bulk Data

Hortonworks Data Platform deploys Apache Sqoop for your Hadoop cluster. Sqoop is a tool designed to transfer data between Hadoop and relational databases. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle into the Hadoop Distributed File System (HDFS), transform the data in Hadoop MapReduce, and then export the data back into an RDBMS. Sqoop automates most of this process, relying on the database to describe the schema for the data to be imported. Sqoop uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance.

For additional information see the Apache Sqoop documentation, including these sections in the Sqoop User Guide:

- Basic Usage
- Sqoop Tools
- Troubleshooting

## 11.1. Apache Sqoop Connectors

Sqoop uses a connector-based architecture which supports plugins that provide connectivity to external systems. Using specialized connectors that work with JDBC drivers, Sqoop connects with external systems that have optimized import and export facilities. Connectors also enable Sqoop to connect with external systems that do not support native JDBC. Connectors are plugin components based on the Sqoop extension framework and can be added to any existing Sqoop installation.

It might be necessary to download and install JDBC drivers for some connectors to function correctly. You can obtain JDBC drivers from the client distribution for your external system's operating system or from the manufacturer of your external system. To install, copy the JDBC driver to the `$SQOOP_HOME/lib` directory of your Sqoop installation.

Hortonworks provides the following connectors for Sqoop in the HDP 2 distribution:

- **MySQL connector:** Instructions for using this connector are available in the Apache Sqoop documentation for MySQL JDBC Connector.
- **Netezza connector:** See Section 11.3, "Netezza Connector" [78] and the Apache Sqoop documentation for Netezza Connector for more information.
- **Oracle JDBC connector:** Instructions for using this connector are available in the Apache Sqoop documentation for Data Connector for Oracle and Hadoop.
- **PostgreSQL connector:** Instructions for using this connector are in the Apache Sqoop documentation for PostgreSQL Connector.
- **Microsoft SQL Server connector:** Instructions for using this connector are in the Apache Sqoop documentation for Microsoft SQL Connector.

A Sqoop connector for Teradata is available from the Hortonworks Add-ons page:

- **Teradata connector:** The connector and its documentation can be downloaded from the Hortonworks Downloads page. Search the page for "Teradata".

## 11.2. Sqoop Import Table Commands

When connecting to an Oracle database, the Sqoop import command requires case-sensitive table names and usernames (typically uppercase). Otherwise the import fails with error message "Attempted to generate class with no columns!"

Prior to the resolution of the issue SQOOP-741, import-all-tables would fail for an Oracle database. See the JIRA for more information.

The import-all-tables command has additional restrictions. See Chapter 8 in the Sqoop User Guide.

## 11.3. Netezza Connector

Netezza connector for Sqoop is an implementation of the Sqoop connector interfaces for accessing a Netezza data warehouse appliance, so that data can be exported and imported to a Hadoop environment from Netezza data warehousing environments.

The HDP 2 Sqoop distribution includes Netezza connector software. To deploy it, the only requirement is that you acquire the JDBC jar file (named `nzjdbc.jar`) from IBM and copy it to the `/usr/local/nz/lib` directory.

**Extra Arguments**

This table describes extra arguments supported by the Netezza connector:

| Argument | Description |
|---|---|
| `--partitioned-access` | Whether each mapper acts on a subset of data slices of a table or all. |
| `-max-errors` | Applicable only in direct mode. This option specifies the error threshold per mapper while transferring data. If the number of errors encountered exceeds this threshold, the job fails. |
| `--log-dir` | Applicable only in direct mode. Specifies the directory where Netezza external table operation logs are stored |

**Direct Mode**

Netezza connector supports an optimized data transfer facility using the Netezza external tables feature. Each map task of Netezza connector's import job works on a subset of the Netezza partitions and transparently creates and uses an external table to transport data.

Similarly, export jobs use the external table to push data quickly onto the NZ system. Direct mode does not support staging tables and upsert options.

Direct mode is specified by the `--direct` Sqoop option.

Here is an example of a complete command line for import using the Netezza external table feature:

```
$ sqoop import \
 --direct \
 --connect jdbc:netezza://nzhost:5480/sqoop \
 --table nztable \
 --username nzuser \
 --password nzpass \
 --target-dir hdfsdir
```

Here is an example of a complete command line for export with tab (\t) as the field terminator character:

```
$ sqoop export \
 --direct \
 --connect jdbc:netezza://nzhost:5480/sqoop \
 --table nztable \
 --username nzuser \
 --password nzpass \
 --export-dir hdfsdir \
 --input-fields-terminated-by "\t"
```

**Null String Handling**

In direct mode the Netezza connector supports the null-string features of Sqoop. Null string values are converted to appropriate external table options during export and import operations.

| Argument | Description |
|---|---|
| `--input-null-non-string <null-string>` | The string to be interpreted as null for non-string columns. |
| `--input-null-non-string <null-string>` | The string to be interpreted as null for non-string columns. |

In direct mode, both the arguments must either be left to the default values or explicitly set to the same value. The null string value is restricted to 0-4 UTF-8 characters.

On export, for non-string columns, if the chosen null value is a valid representation in the column domain, then the column might not be loaded as null. For example, if the null string value is specified as "1", then on export, any occurrence of "1" in the input file will be loaded as value 1 instead of NULL for int columns.

For performance and consistency, specify the null value as an empty string.

**Supported Import Control Arguments**

| Argument | Description |
|---|---|
| `--null-string <null-string>` | The string to be interpreted as null for string columns |
| `--null-non-string <null-string>` | The string to be interpreted as null for non-string columns. |

In direct mode, both the arguments must either be left to the default values or explicitly set to the same value. The null string value is restricted to 0-4 UTF-8 characters.

On import, for non-string columns in the current implementation, the chosen null value representation is ignored for non-character columns. For example, if the null string value is specified as "\N", then on import, any occurrence of NULL for non-char columns in the table will be imported as an empty string instead of \N, the chosen null string representation.

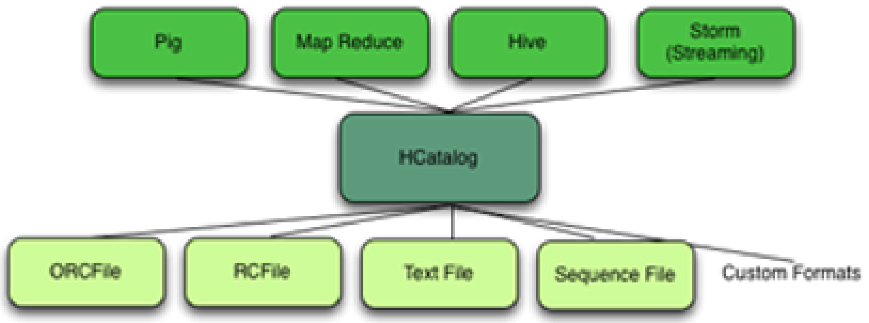
For performance and consistency, specify the null value as an empty string.

## 11.4. Sqoop-HCatalog Integration

This section describes the interaction between HCatalog with Sqoop.

HCatalog is a table and storage management service for Hadoop that enables users with different data processing tools – Pig, MapReduce, and Hive – to more easily read and write data on the grid. HCatalog's table abstraction presents users with a relational view of data in the Hadoop distributed file system (HDFS) and ensures that users need not worry about where or in what format their data is stored: RCFile format, text files, or SequenceFiles.

HCatalog supports reading and writing files in any format for which a Hive SerDe (serializer-deserializer) has been written. By default, HCatalog supports RCFile, CSV, JSON, and SequenceFile formats. To use a custom format, you must provide the InputFormat and OutputFormat as well as the SerDe.



The ability of HCatalog to abstract various storage formats is used in providing RCFile (and future file types) support to Sqoop.

**Exposing HCatalog Tables to Sqoop**

HCatalog interaction with Sqoop is patterned on an existing feature set that supports Avro and Hive tables. This section introduces five command line options. Some command line options defined for Hive are reused.

**Relevant Command-Line Options**

| Command-line Option | Description |
|---|---|
| `--hcatalog-database` | Specifies the database name for the HCatalog table. If not specified, the default database name 'default' is used. Providing the –hcatalog-database option without –hcatalog-table is an error. This is not a required option. |
| `-hcatalog-table` | The argument value for this option is the HCatalog tablename. The presence of the –hcatalog-table option signifies that the import or export job is done using HCatalog tables, and it is a required option for HCatalog jobs. |
| `--hcatalog-home` | The home directory for the HCatalog installation. The directory is expected to have a lib subdirectory and a share/hcatalog subdirectory with necessary HCatalog libraries. If not specified, the system environment variable HCAT_HOME will be checked and failing that, a system property hcatalog.home will be checked. If none of these are set, the default value will be used and currently the |

| Command-line Option | Description |
|---|---|
| | default is set to /usr/lib/hcatalog. This is not a required option. |
| `--create-hcatalog-table` | This option specifies whether an HCatalog table should be created automatically when importing data. By default, HCatalog tables are assumed to exist. The table name will be the same as the database table name translated to lower case. Further described in Section 11.6, "Automatic Table Creation" [82]. |
| `--hcatalog-storage- stanza` | This option specifies the storage stanza to be appended to the table. Further described in Section 11.6, "Automatic Table Creation" [82]. |

### Supported Sqoop Hive Options

The following Sqoop options are also used along with the –hcatalog-table option to provide additional input to the HCatalog jobs. Some of the existing Hive import job options are reused with HCatalog jobs instead of creating HCatalog-specific options for the same purpose.

| Command-line Option | Description |
|---|---|
| `--map-column-hive` | This option maps a database column to HCatalog with a specific HCatalog type. |
| `--hive-home` | The Hive home location. |
| `--hive-partition-key` | Used for static partitioning filter. The partitioning key should be of type STRING. There can be only one static partitioning key. |
| `--hive-partition-value` | The value associated with the partition. |

### Direct Mode Support

HCatalog integration in Sqoop has been enhanced to support direct mode connectors. Direct mode connectors are high performance connectors specific to a database. The Netezza direct mode connector is enhanced to use this feature for HCatalog jobs.

**Important**

Only the Netezza direct mode connector is currently enabled to work with HCatalog.

### Unsupported Sqoop Hive Import Options

Sqoop Hive options that are not supported with HCatalog jobs:

- `--hive-import`
- `--hive-overwrite`

In addition, the following Sqoop export and import options are not supported with HCatalog jobs:

- `--direct`
- `--export-dir`
- `--target-dir`

- `--warehouse-dir`
- `--append`
- `--as-sequencefile`
- `--as-avrofile`

**Ignored Sqoop Options**

All input delimiter options are ignored.

Output delimiters are generally ignored unless either `--hive-drop-import-delims` or `--hive-delims-replacement` is used. When the `--hive-drop-import-delims` or `--hive-delims-replacement` option is specified, all database columns of type CHAR are post-processed to either remove or replace the delimiters, respectively. (See Section 11.7, “Delimited Text Formats and Field and Line Delimiter Characters” [83].) This is only needed if the HCatalog table uses text format.

## 11.5. Controlling Transaction Isolation

Sqoop uses read-committed transaction isolation in its mappers to import data. However, this may not be ideal for all ETL workflows, and you might want to reduce the isolation guarantees. Use the `--relaxed-isolation` option to instruct Sqoop to use read-uncommitted isolation level.

The read-uncommitted transaction isolation level is not supported on all databases, such as Oracle. Specifying the `--relaxed-isolation` may also not be supported on all databases.

**Note**

There is no guarantee that two identical and subsequent uncommitted reads will return the same data.

## 11.6. Automatic Table Creation

One of the key features of Sqoop is to manage and create the table metadata when importing into Hadoop. HCatalog import jobs also provide for this feature with the option `--create-hcatalog-table`. Furthermore, one of the important benefits of the HCatalog integration is to provide storage agnosticism to Sqoop data movement jobs. To provide for that feature, HCatalog import jobs provide an option that lets a user specifiy the storage format for the created table.

The option `--create-hcatalog-table` is used as an indicator that a table has to be created as part of the HCatalog import job.

The option `--hcatalog-storage-stanza` can be used to specify the storage format of the newly created table. The default value for this option is "stored as rcfile". The value specified for this option is assumed to be a valid Hive storage format expression. It will be appended to the CREATE TABLE command generated by the HCatalog import job as part of automatic table creation. Any error in the storage stanza will cause the table creation to fail and the import job will be aborted.

Any additional resources needed to support the storage format referenced in the option `--hcatalog-storage-stanza` should be provided to the job either by placing them in `$HIVE_HOME/lib` or by providing them in `HADOOP_CLASSPATH` and `LIBJAR` files.

If the option `--hive-partition-key` is specified, then the value of this option is used as the partitioning key for the newly created table. Only one partitioning key can be specified with this option.

Object names are mapped to the lowercase equivalents as specified below when mapped to an HCatalog table. This includes the table name (which is the same as the external store table name converted to lower case) and field names.

## 11.7. Delimited Text Formats and Field and Line Delimiter Characters

HCatalog supports delimited text format as one of the table storage formats. But when delimited text is used and the imported data has fields that contain those delimiters, then the data may be parsed into a different number of fields and records by Hive, thereby losing data fidelity.

For this case, one of these existing Sqoop import options can be used:

- `--hive-delims-replacement`
- `--hive-drop-import-delims`

If either of these options is provided on input, then any column of type `STRING` will be formatted with the Hive delimiter processing and then written to the HCatalog table.

## 11.8. HCatalog Table Requirements

The HCatalog table should be created before using it as part of a Sqoop job if the default table creation options (with optional storage stanza) are not sufficient. All storage formats supported by HCatalog can be used with the creation of the HCatalog tables. This makes this feature readily adopt new storage formats that come into the Hive project, such as ORC files.

## 11.9. Support for Partitioning

The Sqoop HCatalog feature supports the following table types:

- Unpartitioned tables
- Partitioned tables with a static partitioning key specified
- Partitioned tables with dynamic partition keys from the database result set
- Partitioned tables with a combination of a static key and additional dynamic partitioning keys

## 11.10. Schema Mapping

Sqoop currently does not support column name mapping. However, the user is allowed to override the type mapping. Type mapping loosely follows the Hive type mapping already present in Sqoop except that the SQL types FLOAT and REAL are mapped to the HCatalog type “float.” In the Sqoop type mapping for Hive, these two SQL types are mapped to “double.” Type mapping is primarily used for checking the column definition correctness only and can be overridden with the `--map-column-hive` option.

All types except binary are assignable to a string type.

Any field of number type (int, shortint, tinyint, bigint and bigdecimal, float and double) is assignable to another field of any number type during exports and imports. Depending on the precision and scale of the target type of assignment, truncations can occur.

Furthermore, date/time/timestamps are mapped to string (the full date/time/timestamp representation) or bigint (the number of milliseconds since epoch) during imports and exports.

BLOBs and CLOBs are only supported for imports. The BLOB/CLOB objects when imported are stored in a Sqoop-specific format and knowledge of this format is needed for processing these objects in a Pig/Hive job or another Map Reduce job.

Database column names are mapped to their lowercase equivalents when mapped to the HCatalog fields. Currently, case-sensitive database object names are not supported.

Projection of a set of columns from a table to an HCatalog table or loading to a column projection is allowed (subject to table constraints). The dynamic partitioning columns, if any, must be part of the projection when importing data into HCatalog tables.

Dynamic partitioning fields should be mapped to database columns that are defined with the NOT NULL attribute (although this is not validated). A null value during import for a dynamic partitioning column will abort the Sqoop job.

## 11.11. Support for HCatalog Data Types

All the primitive HCatalog types are supported. Currently all the complex HCatalog types are unsupported.

BLOB/CLOB database types are only supported for imports.

## 11.12. Providing Hive and HCatalog Libraries for the Sqoop Job

With the support for HCatalog added to Sqoop, any HCatalog job depends on a set of jar files being available both on the Sqoop client host and where the Map/Reduce tasks run. To run HCatalog jobs, the environment variable HADOOP_CLASSPATH must be set up as shown below before launching the Sqoop HCatalog jobs:

```
HADOOP_CLASSPATH=$(hcat -classpath)
export HADOOP_CLASSPATH
```

The necessary HCatalog dependencies will be copied to the distributed cache automatically by the Sqoop job.

## 11.13. Examples

Create an HCatalog table, such as:

```
hcat -e "create table txn(txn_date string, cust_id string, amount
float, store_id int) partitioned by (cust_id string) stored as
rcfile;"
```

Then use Sqoop to import and export the "txn" HCatalog table as follows:

**Import**

```
$SQOOP_HOME/bin/sqoop import --connect <jdbc-url> -table <table-
name> --hcatalog-table txn
```

**Export**

```
$SQOOP_HOME/bin/sqoop export --connect <jdbc-url> -table <table-
name> --hcatalog-table txn
```

## 11.14. Configuring a Sqoop Action to Use Tez to Load Data into a Hive Table

You can use the Tez execution engine to load data into a Hive table using the `--hive-import` option,

In the code example in each step, replace the sample text in [square brackets] with the appropriate information for your configuration.

1. Create a workflow directory.

   ```
   hdfs dfs -mkdir -p [/user/dummy/app]
   ```

2. Create a `lib` directory in the workflow directory.

   ```
   hdfs dfs -mkdir -p [/user/dummy/app/lib]
   ```

3. Copy the database JDBC driver jar file to the `lib` directory.

   ```
   hadoop fs -copyFromLocal [/usr/share/java/mysql-connector-java.jar]
           [/user/dummy/app/lib]
   ```

4. Copy the `hive-site.xml` and `tez-site.xml` files to a location accessible by the workflow. For example:

   ```
   hadoop fs -copyFromLocal [/etc/oozie/conf/action-conf/hive/hive-site.xml /
   user/dummy/app]
   hadoop fs -copyFromLocal [/etc/oozie/conf/action-conf/hive/tez-site.xml /
   user/dummy/app]
   ```

5. In the Sqoop action of the workflow, do the following:

- Add `hive-site` and `tez-site` resources in the `<file>` element of the Sqoop action in the workflow.

  ```
  <file>/user/dummy/app/hive-site.xml#hive-site.xml</file>
  <file>/user/dummy/app/tez-site.xml#tez-site.xml</file>
  ```

- Include the `--hive-import` option in the `<command>` element.

  ```
  <command>import --connect [jdbc:mysql://db_host:port/database] --
  username [user]
   --password [pwd] --driver c[om.mysql.jdbc.Driver] --table [table_name]
  --hive-import -m 1 </command>
  ```

6. Add the following into the `job.properties` file.

   ```
   oozie.use.system.libpath=true
   oozie.action.sharelib.for.sqoop=sqoop,hive
   ```

More information regarding the Sqoop parameters can be found in the Apache documentation at https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html#_importing_data_into_hive

**Example Workflow Action**

Replace all sample text in [square brackets] in the example below with the appropriate workflow name, URI, paths, file names, etc. for your configuration.

```
<action name="sqoop-node">
      <sqoop xmlns="uri:oozie:sqoop-action:0.2">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <configuration>
                 <property>
                      <name>mapred.job.queue.name</name>
                      <value>${queueName}</value>
                 </property>
            </configuration>
            <command>import --connect [jdbc:mysql://db_host:port/database] --
username [user]
--password [pwd] --driver [com.mysql.jdbc.Driver] --table [table_name] --hive-
import -m 1</command>
            <file>[/user/dummy/app/hive-site.xml#hive-site.xml]</file>
            <file>[/user/dummy/app/tez-site.xml#tez-site.xml]</file>
      </sqoop>
      <ok to="end"/>
      <error to="killJob"/>
</action>
```

## 11.15. Troubleshooting Sqoop

### 11.15.1. "Can't find the file" Error when Using a Password File

If you get an error that states "Can't find the file" when using the **–password-file** option with Sqoop, you might need to remove nonvisible content from the password file. Some text editors add bits to the file that cause the data export or import job to fail.

For example, by default VI and VIM add a single bit at the end of a line. To prevent the extra bit from being added, do the following:

1. ç

   ```
   vi -b [filename]
   ```

2. Type the password into the file.

3. Set the option to eliminate the end-of-line bit.

   ```
   :set noeol
   ```

4. Close the file.

   ```
   :wq
   ```

# 12. Using HDP for Workflow and Scheduling With Oozie

Hortonworks Data Platform deploys Apache Oozie for your Hadoop cluster.

Oozie is a server-based workflow engine specialized in running workflow jobs with actions that execute Hadoop jobs, such as MapReduce, Pig, Hive, Sqoop, HDFS operations, and sub-workflows. Oozie supports coordinator jobs, which are sequences of workflow jobs that are created at a given frequency and start when all of the required input data is available.

A command-line client and a browser interface allow you to manage and administer Oozie jobs locally or remotely.

After installing an HDP 2.x cluster by using Ambari 1.5.x, access the Oozie web UI at the following URL:

```
http://{your.oozie.server.hostname}:11000/oozie
```

**Setting the Oozie Client Environment**

The Oozie client requires JDK 1.6 or higher and must be available on all systems where Oozie command line will be run. Java must be included on the path or `$JAVA_HOME` must be set to point to a Java 6+ JDK/JRE.

This is a behavior change for the Oozie client from previous releases.

**Additional Oozie Resources**

For additional Oozie documentation, use the following resources:

- Quick Start Guide
- Developer Documentation
  - Oozie Workflow Overview
  - Running the Examples
  - Workflow Functional Specification
  - Coordinator Functional Specification
  - Bundle Functional Specification
  - EL Expression Language Quick Reference
  - Command Line Tool
  - Workflow Rerun
  - Email Action

  - Writing a Custom Action Executor
  - Oozie Client Javadocs
  - Oozie Core Javadocs
  - Oozie Web Services API
- Administrator Documentation
  - Oozie Installation and Configuration
  - Oozie Monitoring
  - Command Line Tool

## 12.1. ActiveMQ With Oozie and Falcon

You must configure an ActiveMQ URL in Apache Oozie and Apache Falcon components, if those components communicate using an ActiveMQ server that is running on a different host.

If either of the following circumtances apply to your environment, perform the indicated action.

- If Falcon starts ActiveMQ server by default, but Oozie is running on a different host: Set the ActiveMQ server URL in Oozie.
- If Falcon and Oozie are communicating with a standalone ActiveMQ server: Set the ActiveMQ server URL in both Oozie and Falcon.

**To configure ActiveMQ URL in Oozie, add the following property via Ambari and restart Oozie:**

1. In Ambari, navigate to **Services** > **Oozie** > **Configs**.
2. Add the following key/value pair as a property in the **Custom oozie-site** section.

   Key=

   ```
   oozie.jms.producer.connection.properties
   ```

   Value=

   ```
   java.naming.factory.initial#org.apache.activemq.jndi.
   ActiveMQInitialContextFactory;java.naming.provider.url#tcp://{ActiveMQ-
   server-host}:61616;connectionFactoryNames#ConnectionFactory
   ```
3. Navigate to **Services** > **Falcon** > **Configs**.
4. Add the following value for **broker.url** in the **Falcon startup.properties** section.

   ```
   *.broker.url=tcp://{ActiveMQ-server-host}:61616
   ```
5. Click **Service Actions** > **Restart All** to restart the Falcon service.

## 12.2. Configuring Pig Scripts to Use HCatalog in Oozie Workflows

To access HCatalog with a Pig action in an Oozie workflow, you need to modify configuration information to point to the Hive metastore URIs.

There are two methods for providing this configuration information. Which method you use depends upon how often your Pig scripts access the HCatalog.

### 12.2.1. Configuring Individual Pig Actions to Access HCatalog

If only a few individual Pig actions access HCatalog, do the following:

1. Identify the URI (host and port) for the Thrift metastore server.

   a. In Ambari, click **Hive > Configs > Advanced**.

   b. Make note of the URI in the **hive.metastore.uris** field in the General section.

      This information is also stored in the `hive.default.xml` file.

2. Add the following two properties to the <configuration> elements in each Pig action.

   **Note**

   Replace *`[host:port(default:9083)]`* in the example below with the host and port for the Thrift metastore server.

```
<configuration>
    <property>
        <name>hive.metastore.uris</name>
        <value>thrift://[host:port(default:9083)]</value>
        <description>A comma separated list of metastore uris the client can
 use to contact the
        metastore server.</description>
    </property>
    <property>
        <name>oozie.action.sharelib.for.pig</name>
        <value>pig,hive,hcatalog</value>
        <description>A comma separated list of libraries to be used by the
 Pig action.</description>
    </property>
</configuration>
```

### 12.2.2. Configuring All Pig Actions to Access HCatalog

If all of your Pig actions access HCatalog, do the following:

1. Add the following line to the `job.properties` files, located in your working directory:

```
oozie.action.sharelib.for.pig=pig,hive,hcatalog
```

```
<!-- A comma separated list of libraries to be used by the Pig action.-->
```

2. Identify the URI (host and port) for the Thrift metastore server.

   a. In Ambari, click **Hive > Configs > Advanced**.

   b. Make note of the URI in the **hive.metastore.uris** field in the General section.

      This information is also stored in the `hive.default.xml` file.

3. Add the following property to the <configuration> elements in each Pig action.

   **Note**

   Replace *`[host:port(default:9083)]`* in the example below with the host and port for the Thrift metastore server.

```
<configuration>
    <property>
        <name>hive.metastore.uris</name>
        <value>thrift://[host:port(default:9083)]</value>
        <description>A comma separated list of metastore uris the client can
 use to contact the
        metastore server.</description>
    </property>
    </configuration>
```

# 13. Using Apache Flume for Streaming

See the Apache Flume 1.5.2 documentation on the Hortonworks technical documentation website for information about using Flume.

# 14. Troubleshooting

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

## 14.1. 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]
```

## 14.2. 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.

## 14.3. 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.

## 14.4. Invalid Entity Schema

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

Review Falcon entity specifications.

## 14.5. 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'.]
```

## 14.6. Bad Config Store Error

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

## 14.7. 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’
  ```

## 14.8. 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.

# 15. Appendix

## 15.1. Configuring for Disaster Recovery in Releases Prior to HDP 2.5

**MapReduce for Hive Replication**

A temporary fix is needed in MapReduce for Hive replication. The string `/etc/hadoop/conf` must be added to the `mapreduce.application.classpath` property in `mapred-site` through Ambari, before `/etc/hadoop/conf/secure`. For example, the end of the parameter will look like: `mapreduce.application.classpath=...:/etc/hadoop/conf:/etc/hadoop/conf/secure`.

This is a temporary workaround for `https://hortonworks.jira.com/browse/BUG-56803` fixed in versions of HDP 2.4.2 and later. This workaround may cause MapReduce jobs to fail during rolling upgrades due to the underlying configuration changing during job execution.