
Kylo Documentation

Release 0.8.0

Think Big, a Teradata Company

Jun 29, 2017

1	Features	3
2	FAQ	5
3	Terminology	15
4	Release Notes	19
5	Deployment Guides	39
6	Authentication	99
7	Access Control	105
8	Enable Ranger Authorization	111
9	Enable Sentry Authorization	115
10	Kerberos	119
11	Kylo Kerberos SPNEGO	127
12	Kylo UI and SSL	131
13	Setup A NiFi Cluster in a Kylo Sandbox	135
14	NiFi and SSL	137
15	Configuring NiFi for HDFS Encryption	143
16	NiFi & Kylo Reporting Task	149
17	NiFi Processor Guide	157
18	Accessing S3 from the Data Wrangler	163
19	Feed Lineage Configuration	165
20	Sentry Installation Guide	171

21 SUSE Configuration Changes	175
22 Configuration Properties	177
23 Validator Tuning	181
24 Developer Getting Started Guide	183
25 Contributing to Kylo	187
26 Kylo REST API	191
27 Operations Guide	193
28 Troubleshooting & Tips	221
29 Best Practices	235



Kylo website:

The documentation for the site is organized into a few sections:

- *About*
- *Installation*
- *Security*
- *How to guides*
- *Developer guides*
- *User guides*
- *Tips and tricks*

CHAPTER 1

Features

Kylo is a full-featured Data Lake platform built on Apache Hadoop and Spark. Kylo provides a turn-key, business-friendly Data Lake solution enabling data ingest, data preparation, and data discovery.

Features	Description
License	Apache 2.0
Major Features	
Data Ingest	Users can easily configure feeds in guided UI
Data Preparation	Visual sql builder and data wrangling
Operations dashboard	Feed health and service monitoring
Global search	Lucene search against data and metadata
Data Processing	
Data Ingest	Guided UI for data ingest into Hive (extensible)
Data Export	Export data to RDBMS or other targets
Data Wrangling	Visually wrangle data and build/schedule recipes
PySpark, Spark Jobs	Execute Spark jobs
Custom Pipelines	Build and templatize new pipelines
Feed Chaining	Trigger feeds based on dependencies and rules
Ingest Features	
Batch	Batch processing
Streaming	Streaming processing
Snapshot/Incremental Loads	Track highwater using date field or replace target
Schema Discovery	Infer schema from source file samples
Data Validation	Configure field validation in UI
Data Profile	Automatically profile statistics
Data Cleanse/Standardization	Easily configure field standardization rules
Custom Partitioning	Configure Hive partitioning
Ingest Sources	

Continued on next page

Table 1.1 – continued from previous page

FTP, SFTP	Source from FTP, SFTP
Filesystem	Poll files from a filesystem
HDFS, S3	Extract files from HDFS and S3
RDBMS	Efficiently extract RDBMS data
JMS, KAFKA	Source events from queues
REST, HTTP	Source data from messages
Ingest Targets	
HDFS	Store data in HDFS
HIVE	Store data in Hive tables
HBase	Store data in HBase
Ingest Formats	
ORC, Parquet, Avro, RCFile, Text	Store data in popular table formats
Format Compression	Specify compression for ORC and Parquet types
Extensible source formats	Ability to define custom schema plug-in Serdes
Metadata	
Tag/Glossary	Add tags to feeds for searchability
Business Metadata (extended properties)	Add business-defined fields to feeds
REST API	Powerful REST APIs for automation and integration
Visual Lineage	Explore process lineage
Profile History	View history of profile statistics
Search/Discover	Lucene syntax search against data and metadata
Operational Metadata	Extensive metadata capture
Security	
Kerberos Support	Supports Kerberized clusters
Obfuscation	Configure field-level data protection
Encryption at Rest	Compatible with HDFS encryption features
Access Control (LDAP, KDC, AD, SSO)	Flexible security options
Data Protection	UI configurable data protection policies
Application Groups, Roles	Admin configured roles
Operations	
Dashboard	KPIs, alerts, performance, troubleshooting
Scheduler	Timer, Cron-style based on Quartz engine
SLA Monitoring	Service level agreements tied to feed performance
Alerts	Alerts with integration options to enterprise
Health Monitoring	Quickly identify feed and service health issues
Performance Reporting	Pivot on performance statistics
Scalability	
Edge Clustering	Scale edge resources

About Kylo

What is Kylo?

Kylo is a feature-rich data lake platform built on Apache Hadoop and Spark. Kylo provides a turn-key, business-friendly, data lake solution enabling self-service data ingest, data preparation, and data discovery.

Kylo's web application layer offers features oriented to business users, including data analysts, data stewards, data scientists, and IT operations personnel. Kylo integrates best practices around metadata capture, security, and data quality. Furthermore, Kylo provides a flexible data processing framework (leveraging Apache NiFi) for building batch or streaming pipeline templates, and for enabling self-service features without compromising governance requirements.

What are Kylo's origins?

Kylo was developed by (a Teradata company) and it is in use at a dozen major corporations globally. Think Big provides big data and analytics consulting to the world's largest organizations, working across every industry in performing 150 successful big data projects over the last seven years. Think Big has been a major beneficiary of the open-source Hadoop ecosystem and elected to open-source Kylo in order to contribute back to the community and improve value.

What does Kylo mean?

Kylo is a play on the Greek word meaning "flow".

What software license is Kylo provided under?

(a Teradata company) has released Kylo under the Apache 2.0 license.

Who uses Kylo?

Kylo is being used in beta and production at a dozen major multi-national companies worldwide across industries such as manufacturing, banking/financial, retail, and insurance. Teradata is working with legal departments of these companies to release names in upcoming press releases.

What skills are required for a Kylo-based Data Lake implementation?

Many organizations have found implementing big data solutions on the Hadoop stack to be a complex endeavor. Big data technologies are heavily oriented to software engineering and system administrators, and even organizations with deep engineering capabilities struggle to staff teams with big data implementation experience. This leads to multi-year implementation efforts that unfortunately can lead to data swamps and fail to produce business value. Furthermore, the business-user is often overlooked in features available for in-house data lake solutions.

Kylo attempts to change all this by providing out-of-the-box features and patterns critical to an enterprise-class data lake. Kylo provides an IT framework for delivering powerful pipelines as templates and enabling user self-service to create feeds from these data processing patterns. Kylo provides essential Operations capabilities around monitoring feeds, troubleshooting, and measuring service levels. Designed for extensibility, software engineers will find Kylo's APIs and plug-in architecture flexible and easy to use.

Enterprise Support

Is enterprise support available for Kylo?

Yes, (a Teradata company) offers support subscription at the standard and enterprise level. Please visit the website for more information.

Are professional services and consulting available for Kylo?

(a Teradata company) provides global consulting services with expertise in implementing Kylo-based solutions. It is certainly possible to install and learn Kylo using internal resources. Think Big's Data Lake Foundation provides a quick start to installing and delivering on your first set of data lake use cases. Think Big's service includes hands-on training to ensure that your business is prepared to assume operations.

Is enterprise training available for Kylo?

Yes, (a Teradata company) offers training on Kylo, Hadoop, and Spark.

Are commercial managed services available for Kylo?

Yes, (a Teradata company) can provide managed operations for your Hadoop cluster, including Kylo, whether it is hosted on-premise or in the cloud. The managed services team is trained specifically on Kylo and they have operations experience with major Hadoop distributions.

Architecture

What is the deployment architecture?

Kylo is a modern web application installed on a Linux “edge node” of a Spark & Hadoop cluster. Kylo contains a number of special purposed routines for data lake operations leveraging Spark and Apache Hive.

Kylo utilizes Apache NiFi as its scheduler and orchestration engine, providing an integrated framework for designing new types of pipelines with 200 processors (data connectors and transforms). Kylo has an integrated metadata server currently compatible with databases such as MySQL and Postgres.

Kylo can integrate with Apache Ranger or Sentry and CDH Navigator or Ambari for cluster monitoring.

Kylo can optionally be deployed in the cloud.

What are the individual component/technologies involved in a Kylo deployment?

- Kylo UI. AngularJS browser app with Google Material Design running in a Tomcat container
- Kylo Services. Services, REST APIs, and plug-ins perform the backbone of Kylo. All features and integrations with other technologies are managed through the services layer.
- Kylo Spark Shell. Manages Spark sessions for data wrangling.
- Kylo Metadata Server. Combination of JBoss ModeShape and MySQL (or Postgres) store all metadata generated by Kylo.
- Apache NiFi. Pipeline orchestration engine and scheduler.
- ActiveMQ. JMS queue for inter-process communication.
- Apache Spark. Executes Kylo jobs for data profiling, data validation, and data cleansing. Also supports data wrangling and schema detection.
- Elasticsearch. Provides the index for search features in Kylo such as free-form data and metadata
- Apache Hadoop. All Hadoop technologies are available but most notably YARN, HDFS, Hive

Is Kylo compatible with Cloudera, Hortonworks, Map R, EMR, and vanilla Hadoop distributions?

Yes. Kylo generally relies on standard Hadoop APIs and common Hadoop technologies like HDFS, Hive, and Spark. NiFi operates on the “edge” so isn’t bound to any particular Hadoop distribution. It is therefore compatible with most Hadoop distributions, although we currently only provide install instructions for Cloudera and Hortonworks.

Does Kylo support either Apache NiFi or Hortonworks DataFlow (HDF)? What is the difference?

Yes, Kylo supports vanilla Apache NiFi or NiFi bundled with Hortonworks DataFlow. HDF bundles Apache NiFi, Storm, and Kafka within a distribution. Apache NiFi within HDF contains the same codebase as the open-source project. NiFi is a critical component of the Kylo solution. Kylo is an HDF-certified technology. Kylo’s commercial support subscription bundles 16 cores of Apache NiFi support.

Can Kylo be used in the cloud?

Absolutely. Kylo is used in production on AWS utilizing EC2, S3, SQS, and other AWS features for at least one major Fortune 100 company. Kylo has also been used with Azure.

Does Kylo support high-availability (HA) features?

Yes, Kylo clustering is possible via a load-balancer. In addition, current data processing running under NiFi will not be impacted if Kylo becomes unavailable or during upgrades.

Metadata

What type of metadata does Kylo capture?

Kylo captures extensive business and technical (for example, schema) metadata defined during the creation of feeds and categories. Kylo processes lineage as relationships between feeds, sources, and sinks. Kylo automatically captures all operational metadata generated by feeds. In addition, Kylo stores job and feed performance metadata and SLA metrics. We also generate data profile statistics and samples.

How does Kylo support metadata exchange with 3rd party metadata servers

Kylo's metadata server has REST APIs that could be used for metadata exchange and documented directly in the application through Swagger.

What is Kylo's metadata server?

A key part of Kylo's metadata architecture relies on the open-source JBoss ModeShape framework. ModeShape is a JCR compliant store. Modeshape supports dynamic schemas providing the ability to easily extend Kylo's own data model.

Some core features:

- Dynamic schemas - provide extensible features for extending schema towards custom business metadata in the field
- Versioning - ability to track changes to metadata over time
- Text Search - flexible searching metastore
- Portability - can run on sql and nosql databases

See:

How extensible is Kylo metadata model?

Very extensible due our use of ModeShape (see above).

In addition, the Kylo application allows an administrator to define standard business metadata fields that users will be prompted to enter when creating feeds and categories.

Are there any business-related data captured, or are they all operational metadata?

Business metadata fields can be defined by the user and will appear in the UI during the feed setup process.

What does the REST API look like?

Please access the REST documentation through a running Kylo instance: <http://kylo-host:8400/api-docs/index.html>

Does the Kylo application provide a visual lineage?

Yes, Kylo provides a visual process lineage feature for exploring relationships between feeds and shared sources and sinks. Job instance level lineage is stored as “steps” visible in the feed job history.

What type of process metadata do we capture?

Kylo captures job and step level information on the status of the process, with some information on the number of records loaded, how long it took, when it was started and finished, and what errors or warnings may have been generated. We capture operational metadata at each step, which can include record counts, dependent upon the type of step.

Development Lifecycle

What’s the pipeline development process using Kylo?

Pipeline templates developed with Apache NiFi and registered with Kylo can be developed and tested in a sandbox environment, exported from Kylo, and then imported into Kylo in a UAT and production environment after testing. Once the NiFi template is registered with Kylo, a business user can configure new feeds through Kylo’s step-guided user interface.

Existing Kylo feeds can be exported from one environment into a zip file that contains a combination of the underlying template and metadata. The package can then be imported to the production NiFi environment by an administrator.

Does deployment of new templates or feeds require restart?

No restart is required to deploy new pipeline templates or feeds.

Can new feeds be created in automated fashion instead of manually through the UI?

Yes, via Kylo’s REST API. See the section on Swagger documentation (above).

Tool Comparisons

Is Kylo similar to any commercial products?

Kylo has similar capabilities to Podium and Zaloni Bedrock. Kylo is an open-source option. One differentiator is Kylo's extensibility. Kylo provides a plug-in architecture with a variety of extensions available to developers, and the use of NiFi templates provides incredible flexibility for batch and streaming use cases.

Is Kylo's operations dashboard similar to Cloudera Manager and Apache Ambari?

Kylo's dashboard is feed-health centric. Health of a feed is determined by job completion status, service level agreement violations, and rules that measure data quality. Kylo provides the ability to monitor feed performance and troubleshoot issues with feed job failures.

Kylo monitors services in the cluster and external dependencies to provide a holistic view of services your data lake depends on. Kylo provides a simple plugin for adding enterprise services to monitor. Kylo includes plugins for pulling service status from Ambari and Cloudera Navigator. This is useful for correlating service issues with feed health problems.

Is Kylo's metadata server similar to Cloudera Navigator, Apache Atlas?

In some ways. Kylo is not trying to compete with these and could certainly imagine integration with these tools. Kylo includes its own extensible metadata server. Navigator is a governance tool that comes as part of the Cloudera Enterprise license. Among other features, it provides data lineage of your Hive SQL queries. We think this is useful but only provides part of the picture. Kylo's metadata framework is really the foundation of an entire data lake solution. It captures both business and operational metadata. It tracks lineage at the feed-level. Kylo provides IT Operations with a useful dashboard, providing the ability to track/enforce Service Level Agreements, and performance metrics. Kylo's REST APIs can be used to do metadata exchange with tools like Atlas and Navigator.

How does Kylo compare to traditional ETL tools like Talend, Informatica, Data Stage?

Kylo uses Apache NiFi to orchestrate pipelines. NiFi can connect to many different sources and perform lightweight transformations on the edge using 180+ built-in processors. Generally workload is delegated to the cluster where the bulk of processing power is available. Kylo's NiFi processor extensions can effectively invoke Spark, Sqoop, Hive, and even invoke traditional ETL tools (for example: wrap 3rd party ETL jobs).

Many ETL (extract-transform-load) tools are focused on SQL transformations using their own proprietary technology. Data warehouse style transformations tend to be focused on issues such as loading normalized relational schemas such as a star or snowflake. Hadoop data patterns tend to follow ELT (extract and load raw data, then transform). In Hadoop, source data is often stored in raw form, or flat denormalized structures. Powerful transformation techniques are available via Hadoop technologies, including Kylo's leveraging of Spark. We don't often see the need for expensive and complicated ETL technologies for Hadoop.

Kylo provides a user interface for an end-user to configure new data feeds including schema, security, validation, and cleansing. Kylo provides the ability to wrangle and prepare visual data transformations using Spark as an engine.

What is Kylo's value-add over plain Apache NiFi?

NiFi acts as Kylo's pipeline orchestration engine, but NiFi itself does not provide all of the tooling required for a data lake solution. Some of Kylo's distinct benefits over vanilla NiFi and Hadoop:

- Write-once, use many times. NiFi is a powerful IT tool for designing pipelines, but most data lake feeds utilize just a small number of unique flows or “patterns”. Kylo allows IT the flexibility to design and register a NiFi template as a data processing model for feeds. This enables non-technical business users to configure dozens, or even hundreds of new feeds through Kylo’s simple, guided stepper-UI. In other words, our UI allows users to setup feeds without having to code them in NiFi. As long as the basic ingestion pattern is the same, there is no need for new coding. Business users will be able to bring in new data sources, perform standard transformations, and publish to target systems.
- Operations Dashboard UI can be used for monitoring data feeds. It provides centralized health monitoring of feeds and related infrastructure services, Service Level Agreements, data quality metrics reporting, and alerts.
- Web modules offer key data lake features such as metadata search, data discovery, data wrangling, data browse, and event-based feed execution (to chain together flows).
- Rich metadata model with integrated governance and best practices.
- Kylo adds a set of data lake specific NiFi extensions around Data Profile, Data Cleanse, Data Validate, Merge/Dedupe, High-water. In addition, general Spark and Hive processors not yet available with vanilla NiFi.
- Pre-built templates that implement data lake best practices: Data Ingest, ILM, and Data Processing.

Scheduler

How does Kylo manage job priority?

Kylo exposes the ability to control which yarn queue a task executes on. Typically scheduling this is done through the scheduler. There are some advanced techniques in NiFi that allow further prioritization for shared resources.

Can Kylo support complicated ETL scheduling?

Kylo supports cron-based scheduling, but also timer-based, or event-based using JMS and an internal Kylo ruleset. NiFi embeds the Quartz.

What’s the difference between “timer” and “cron” schedule strategies?

Timer is fixed interval, “every 5 minutes or 10 seconds”. Cron can be configured to do that as well, but can handle more complex cases like “every tues at 8AM and 4PM”.

Does Kylo support 3rd party schedulers

Yes, feeds can be triggered via JMS or REST.

Does Kylo support chaining feeds? One data feed consumed by another data feed?

Kylo supports event-based triggering of feeds based on preconditions or rules. One can define rules in the UI that determine when to run a feed, such as “run when data has been processed by feed a and feed b and wait up to an hour before running anyway”. We support simple rules up to very complicated rules requiring use of our API.

Security

Does Kylo support roles?

Kylo supports the definition of roles (or groups), and the specific permissions a user with that role can perform, down to the function level.

What authentication methods are available?

Kylo uses Spring Security. Using pluggable login-modules, it can integrate with Active Directory, Kerberos, LDAP, or most any authentication provider. See *Developer Getting Started Guide*.

What security features does Kylo support?

Kylo provides plugins that integrate with Apache Ranger or Apache Sentry, depending on the distribution that you are running. These can be used to configure feed-based security and impersonating users properly to enforce user permissions. Kylo fully supports Kerberized clusters and built-in features, such as HDFS encryption.

Is Kylo PCI compliant?

Kylo can be configured to use TLSv1.2 for all network communication it uses internally or externally. We are testing running NiFi repositories on encrypted disk with a client. v0.8 will include some improvements required for full PCI compliance.

Data Ingest

What is Kylo's standard batch ingest workflow?

Kylo includes a sample pipeline template that implements many best practices around data ingest, mostly utilizing Spark. Kylo makes it very simple for a business user to configure ingest of new source files and RDMBS tables into Hive. Data can be read from a filesystem attached to the edge node, or directly using Kylo's sqoop processor into Hadoop. Original data is archived into a distinct location. Small files are optionally merged and headers stripped, if needed. Data is cleansed, standardized, and validated based on user-defined policies. Invalid records are binned into a separate table for later inspection. Valid records are inserted into a final Hive table with options such as (append, snapshot, merge with dedupe, upsert, etc). Target format can differ from the raw source, contain custom partitions, and group-based security. Finally each batch of valid data is automatically profiled.

Does Kylo support batch and streaming?

Yes, either types of pipelines can configured with Kylo. Kylo tracks performance statistics of streaming-style feeds in activity over units of time. Kylo tracks performance of batch feeds in jobs and steps.

Which raw formats does Kylo support?

Kylo has a pluggable architecture for adding support for new types. Currently Kylo supports delimited-text formats (for example: csv, tab, pipe) and all Hadoop formats, such as ORC, Parquet, RCFile, AVRO, and JSON.

Which target formats for Hive does Kylo support?

Kylo supports text-file, Parquet and ORC (default) with optional block compression, AVRO, text, and RCFile.

How does “incremental” loading strategy of a data feed work?

Kylo supports a simple incremental extract component. We maintain a high-water mark for each load using a date field in the source record.

Can Kylo ingest from relational databases?

Yes, Kylo allows a user to select tables from RDBMS sources and easily configure ingest feeds choosing the target table structure, cleansing and validation rules, and target format. Kylo invokes Sqoop via NiFi to avoid IO through the edge node.

Kylo’s RDBMS ingest support requires configuring a type-specific JDBC driver. It has been tested with data sources such as Teradata, SQL Server, Oracle, Postgres, and MySQL.

There are a lot of new terms with Kylo and NiFi, and trying to learn them all, including distinctions between Kylo and NiFi usage, can be overwhelming. The goal of this document is to detail the semantics of these terms within the context of Kylo and NiFi. This document does not aim to write a definition for every term you will encounter in Kylo and Apache NiFi.

Additional Resources:

- NiFi has documentation on its on their website. However, some of the terms will be outlined here in the context of Kylo.

Apache NiFi Terminology

Processor

Refer to the NiFi document for NiFi-specific terminology.

- A processor has properties that are configured. The values for these properties can be hard-coded, or they can be made dynamic by using the NiFi expression language, which will allow you to access the attributes of a FlowFile as they go through the processor. They can also be set or overridden through Kylo.

FlowFile

Immutable NiFi object that encapsulates the data that moves through a NiFi flow. It consists of the data (content) and some additional properties (attributes)

- NiFi wraps data in FlowFiles. FlowFiles can contain a piece of data, an entire dataset, and batches of data, depending upon which processors are used, and their configurations. A NiFi flow can have multiple FlowFiles running through it at one time, and the FlowFiles can move from processor to processor independently of one another. It is important to note that FlowFiles only conceptually “contain” the data. For scalability reasons, FlowFiles actually have a pointer to the data in the NiFi Content Repository.

Connection

A connection between two processors, between input/output ports, or between both

- FlowFiles move from processor to processor through connections. A connection houses a queue. If a processor on the receiving end of a connection is stopped or disabled, the FlowFiles will sit in that queue/connection until the receiving processor is able to receive FlowFiles again.

Relationship

Closely tied to NiFi connections, see definition in NiFi terminology document

- When a processor is done with a FlowFile, it will route it to one or more relationships. These relationships can either be set to auto-terminate (this would mark the end of the journey for FlowFiles that get routed to auto-terminating relationships), or they can be attached to NiFi connections. The most common example is the success and failure relationships. Processors, when finished with a FlowFile, determine which relationship(s) to route the FlowFile to. This can create diverging paths in a flow, and can be used to represent conditional business logic. For example: a flow can be designed so that when processor A routes to the success relationship it goes to processor B, and when processor A routes to the failure relationship it routes to processor C.

Flow/Dataflow

A logically grouped sequence of connected processors and NiFi components

- You could also think of a flow as a program or a pipeline.

Controller Service

Refer to the NiFi document for NiFi-specific terminology.

- An example is the Hive Thrift Service of type ThriftConnectionPool, which is a controller service that lets the ExecuteHQL and ExecuteHQLStatement processor types connect to a HiveServer2 instance.

NAR files

Similar to an uber JAR, a NiFi archive which may contain custom NiFi processors, controllers and all library dependencies

- NAR files are bundles of code that you use to extend NiFi. If you write a custom processor or other custom extension for NiFi, you must package it up in a NAR file and deploy it to NiFi.

Template

Refer to the NiFi document for NiFi-specific terminology.

- A template is a flow that has been saved for reuse. You can use a template to model a common pattern, and then create useful flows out of that by configuring the processors to your specific use case. They can be exported and imported as XML. The term “template” becomes overloaded with the introduction of Kylo, so it is important when thinking and talking about Kylo to specify which kind of “template” you are referring to.

Kylo Terminology

Registered Template

The blueprint from which Kylo feeds are created.

- In Kylo, a template typically refers to a registered template. A registered template is a NiFi template that has been registered through Kylo. When trying to register a NiFi template, there are multiple courses of action. The first option is to upload a NiFi template that has been previously exported from NiFi as XML. This option does not actually add the NiFi template to the list of registered templates in Kylo. Instead, this will upload the NiFi template to the running instance of NiFi, which is futile if you already have that template available in the running instance of NiFi. The second option is to register a NiFi template directly through NiFi. This will allow you to choose from the NiFi templates that are available in the running instance of NiFi and register it. This does add it to the list of registered templates. The third option is to upload a template that has been exported from Kylo as a zip. Registered templates can be exported from one running instance of Kylo and registered in other instances of Kylo by uploading the archive file (zip). An archive of a registered template will also have the NiFi template in it. It is easiest to think of Kylo templates (a.k.a., registered templates) as being a layer on top of NiFi templates.

Category

A container for grouping feeds

- Each feed must belong to a category. A feed cannot belong to multiple categories, but a category can contain multiple feeds. A category is used as metadata in Kylo, and also manifests itself as a process group in the running instance of NiFi

Input Processor or Source

The processor in a feed's underlying flow that is at the beginning of the flow and generates FlowFiles rather than transforming incoming ones

- There are processors that do not take incoming connections, and instead generate FlowFiles from external sources. An example is the GetFile processor, which runs at a configured interval to check a specified directory for data. While these processors don't necessarily "kick off" a flow, as a flow is always running (unless the components are stopped or disabled), these processors are the origin for a flow and are considered the source or input processors of a feed.

Feed

Typically will represent the key movement of data between a source (flat file) and sink (e.g. Hive)

- An instantiation of a Kylo template
- Feeds are created from templates. The idea is that NiFi templates are created to be reusable and generic. Then, the NiFi templates are registered in Kylo, and the technical configurations of the NiFi template are hidden and default values are set so that it is prepared for the end user. Then, the end user, equipped with their domain knowledge, creates feeds from the Kylo templates.

Job

A single run of a feed

- When an input processor generates a FlowFile, a new job for that feed starts. The job follows the FlowFile through its feed's underlying flow, capturing metadata along the way. Jobs can be of two types, FEED or CHECK. By default, all jobs are of type FEED. They can be set to type CHECK by configuring one of the processors to set the `tb.jobType` attribute to CHECK.

Step

A stage in a job

- Steps are specific to jobs in Kylo, and correlate directly to the processors that the FlowFile goes through for that job. Flows can have conditional logic and multiple relationships, so each FlowFile that goes through a flow may not follow the same path every time. A job follows a FlowFile, and has a step for each processor that the FlowFile goes through.

Service

A service that Kylo has been configured to monitor

- Services in Kylo are not NiFi controller services. They are simply services, such as HDFS and Kafka, that Kylo will monitor using either Ambari's API or Cloudera's REST client.

Latest

Release 0.8.1 (May 24, 2017)

Highlights

- 140+ issues resolved
- You can now assign users and groups access to feeds, categories, and templates giving you fine grain control of what users can see and do. Refer to the [Access Control](#) for more information.
- Kylo can now be clustered for high availability. Refer to `../installation/KyloClusterConfiguration` for more information.
- You now mix and match the order of standardizers and validators giving you more control over the processing of your data.
- The wrangler has been improved with a faster transformation grid and now shows column level profile statistics as you transform your data.

Download Links

- RPM : <http://bit.ly/2r4P47A>
- Debian : <http://bit.ly/2rYObgz>

Upgrade Instructions from v0.7.1

- If upgrading directly from v0.7.1, run the database update to enable Liquibase.

```
/opt/kylo/setup/sql/mysql/kylo/0.8.0/update.sh <db-hostname> <db-user> <db-password>
```

Upgrade Instructions from v0.8.0

Build or [download the RPM](#)

1. Uninstall Kylo:

```
/opt/kylo/remove-kylo.sh
```

3. Install the new RPM:

```
rpm -ivh <RPM_FILE>
```

4. Copy the application.properties file from the 0.8.0.1 install. If you have customized the application.properties file you will want to copy the 0.8.0.1 version and add the new properties that were added for this release.

4.1 Find the /bkup-config/TIMESTAMP/kylo-services/application.properties file

- Kylo will backup the application.properties file to the following location, `/opt/kylo/bkup-config/YYYY_MM_DD_HH_MM_millis/kylo-services/application.properties`, replacing the “YYYY_MM_DD_HH_MM_millis” with a valid time:

4.2 Copy the backup file over to the /opt/kylo/kylo-services/conf folder

```
### move the application.properties shipped with the .rpm to a_
↳ backup file
mv /opt/kylo/kylo-services/application.properties application.
↳ properties.0_8_1_template
### copy the backup properties (Replace the YYYY_MM_DD_HH_MM_
↳ millis with the valid timestamp)
cp /opt/kylo/bkup-config/YYYY_MM_DD_HH_MM_millis/kylo-services/
↳ application.properties /opt/kylo/kylo-services/conf
```

4.3 Two new properties were added. Add in the 2 new properties to the /opt/kylo/kylo-services/conf/application.properties file

```
# Entity-level access control. To enable, uncomment below line and_
↳ set value as true
#security.entity.access.controlled=false

## optional. If added you can control the timeout when you delete_
↳ a feed
kylo.feed.mgr.cleanup.timeout=60000
```

Refer to the [Access Control](#) document for more information about entity level access control. To enable entity access control ensure the property above is set to true.

4.4 Ensure the property `security.jwt.key` in both `kylo-services` and `kylo-ui` application.properties file match. They property below needs to match in both of these files:

- `/opt/kylo/kylo-ui/conf/application.properties`
- `/opt/kylo/kylo-services/conf/application.properties`.

```
security.jwt.key=
```

5. Backup the Kylo database. Run the following code against your kyp database to export the ‘kylo’ schema to a file. Replace the PASSWORD with the correct login to your kylo database.

```
mysqldump -u root -pPASSWORD --databases kylo >kylo-0_8_0_1_backup.sql
```

6. Database updates. Kylo uses liquibase to perform database updates. Two modes are supported.

- Automatic updates

By default Kylo is set up to automatically upgrade its database on Kylo services startup. As such, there isn't anything specific an end user has to do. When Kylo services startup the kylo database will be automatically upgraded to latest version if required. This is configured via an application.properties setting

```
liquibase.enabled=true
```

- Manual updates

Sometimes, however you may choose to disable liquibase and manually apply the upgrade scripts. By disabling liquibase you are in control of how the scripts are applied. This is needed if the kylo database user doesn't have privileges to make schema changes to the kylo database. Please follow this [../how-to-guides/DatabaseUpgrades](#) on how to manually apply the additional database updates.

7. Re-import Data Ingest template (data_ingest.zip).

- Kylo now allows converting data ingested from a database into AVRO format, and then running it further through the flow.
- To enable this, re-import the data_ingest.zip file (Templates -> + icon -> Import from a file -> Choose file -> Check yes to 'overwrite' feed template -> Check yes to 'Replace the reusable template' -> Import template)

Previous Releases

Release 0.8.0 (Apr 19, 2017)

Highlights

- 90+ issues resolved
- Automatic and manual database upgrades. See [../how-to-guides/DatabaseUpgrades](#)
- Support for PostgreSQL as Kylo metadata store
- Join Hive and any JDBC tables in Data Transformation feeds by creating a new Data Source.
- Wrangler can now use standardization and validation functions, and be merged, profiled, and indexed.
- The Feed/Template import provides visual feedback and progress
- Kylo will now encrypt 'sensitive' properties and enforce 'required' properties.

Upgrade Instructions from v0.7.1

Build or [download the RPM](#)

1. Shut down NiFi:

```
service nifi stop
```

2. Uninstall Kylo:

```
/opt/kylo/remove-kylo.sh
```

3. Install the new RPM:

```
rpm -ivh <RPM_FILE>
```

4. Copy the application.properties file from the 0.7.1 install. If you have customized the application.properties file you will want to copy the 0.7.1 version and add the new properties that were added for this release.

4.1 Find the /bkup-config/TIMESTAMP/kylo-services/application.properties file

- Kylo will backup the application.properties file to the following location, `/opt/kylo/bkup-config/YYYY_MM_DD_HH_MM_millis/kylo-services/application.properties`, replacing the “YYYY_MM_DD_HH_MM_millis” with a valid time:

4.2 Copy the backup file over to the /opt/kylo/kylo-services/conf folder

```
### move the application.properties shipped with the .rpm to a_
↳ backup file
mv /opt/kylo/kylo-services/application.properties application.
↳ properties.0_8_0_template
### copy the backup properties (Replace the YYYY_MM_DD_HH_MM_
↳ millis with the valid timestamp)
cp /opt/kylo/bkup-config/YYYY_MM_DD_HH_MM_millis/kylo-services/
↳ application.properties /opt/kylo/kylo-services/conf
```

4.3 Add in the 2 new properties to the /opt/kylo/kylo-services/conf/application.properties file

```
liquibase.enabled=true
liquibase.change-log=classpath:com/thinkbiganalytics/db/master.xml
```

4.4 Ensure the property `security.jwt.key` in both `kylo-services` and `kylo-ui` application.properties file match. They property below needs to match in both of these files:

- `/opt/kylo/kylo-ui/conf/application.properties`
- `/opt/kylo/kylo-services/conf/application.properties`.

```
security.jwt.key=
```

4.5 If using Spark 2 then add the following property to the `/opt/kylo/kylo-services/conf/application.properties` file

```
config.spark.version=2
```

5. Backup the Kylo database. Run the following code against your kylo database to export the ‘kylo’ schema to a file. Replace the PASSWORD with the correct login to your kylo database.

```
mysqldump -u root -pPASSWORD --databases kylo >kylo-0_7_1_backup.sql
```

6. Upgrade Kylo database:

```
/opt/kylo/setup/sql/mysql/kylo/0.8.0/update.sh localhost root <password or_
↳ blank>
```

7. Additional Database updates. Kylo now uses liquibase to perform database updates. Two modes are supported.

- Automatic updates

By default Kylo is set up to automatically upgrade its database on Kylo services startup. As such, there isn't anything specific an end user has to do. When Kylo services startup the kylo database will be automatically upgraded to latest version if required.

- Manual updates

Sometimes, however you may choose to disable liquibase and manually apply the upgrade scripts. By disabling liquibase you are in control of how the scripts are applied. This is needed if the kylo database user doesn't have privileges to make schema changes to the kylo database. Please follow this [../how-to-guides/DatabaseUpgrades](#) on how to manually apply the additional database updates.

8. Update the NiFi nars. Run the following shell script to copy over the new NiFi nars/jars to get new changes to NiFi processors and services.

```
/opt/kylo/setup/nifi/update-nars-jars.sh
```

9. Update the NiFi Templates.

- The Data Transformation template now allows you to apply standardization and validation rules to the feed. To take advantage of this you will need to import the new template. The new data transformation template can be found:

If you import the new Data Transformation template, be sure to re-initialize your existing Data Transformation feeds if you update them.

Data Transformation Enhancement Changes

New to this release is the ability for the data wrangler to connect to various JDBC data sources, allowing you to join Hive tables with, for example, MySQL or Teradata. The JDBC drivers are automatically read from `/opt/nifi/mysql/` when Kylo is starting up. When Kylo Spark Shell is run in `yarn-client` mode then these jars need to be added manually to the `run-kylo-spark-shell.sh` script:

- Edit `/opt/kylo/kylo-services/bin/run-kylo-spark-shell.sh` and append `-jars` to the `spark-submit` command-line:

```
spark-submit --jars /opt/nifi/mysql/mariadb-java-client-1.5.7.jar ...
```

Additional driver locations can be added separating each location with a comma

```
spark-submit --jars /opt/nifi/mysql/mariadb-java-client-1.5.7.jar,/opt/nifi/
↳teradata/terajdbc4.jar ...
```

Ambari Service Monitor Changes

The Ambari Service Monitor is now a Kylo plugin jar. In order for Kylo to report status on Ambari services you will need to do the following:

1. Modify/Ensure the connection properties are setup. The ambari connection parameters need to be moved out of the main `kylo-services` application.properties to a new file called `ambari.properties`
 - Create a new file `/opt/kylo/kylo-services/conf/ambari.properties`. Ensure the owner of the file is `kylo`
 - Add and configure the following properties in that file:

```
ambariRestClientConfig.host=127.0.0.1
ambariRestClientConfig.port=8080
ambariRestClientConfig.username=admin
ambariRestClientConfig.password=admin
ambari.services.status=HDFS,HIVE,MAPREDUCE2,SQOOP
```

2. Copy the `/opt/kylo/setup/plugins/kylo-service-monitor-ambari-0.8.0.jar` to `/opt/kylo/kylo-services/plugin`

```
cp /opt/kylo/setup/plugins/kylo-service-monitor-ambari-0.8.0.jar /opt/kylo/kylo-
↪services/plugin/
```

Release 0.7.1 (Mar 13, 2017)

Highlights

- 64 issues resolved
- UI performance. Modules combined in a single page application and many other optimizations.
- Lineage auto-alignment. Correctly aligns feeds, sources, and destinations.
- Wrangle and machine learning. Added over 50 machine learning functions to the data wrangler. The data wrangler now supports over 600 functions!
- Test framework. Initial groundwork for automated integration testing.
- **Notable issues resolved:**
 - Multiple Spark validation and profiler issues resolved
 - Login issues when using https
 - Race condition on startup of Kylo and Modeshape service
- For a complete list of resolved issues see here: [ReleaseNotes7.1.resolvedIssues](#)

RPM

Upgrade Instructions from v0.7.0

Build or download the RPM:

1. Uninstall the RPM, run:

```
/opt/kylo/remove-kylo.sh
```

2. Install the new RPM:

```
rpm -ivh <RPM_FILE>
```

3. Update the Database:

```
/opt/kylo/setup/sql/mysql/kylo/0.7.1/update.sh localhost root <password or blank>
```

4. Start kylo apps:


```
/opt/kylo/start-kylo-apps.sh
```

Release 0.7.0 (Feb. 13, 2017)

Highlights

- Renamed thinkbig artifacts to kylo
- Our REST API documentation has been updated and reorganized for easier reading. If you have Kylo running you can open <http://localhost:8400/api-docs/index.html> to view the docs.
- Kylo is now available under the Apache 2 open-source license. Visit our new [GitHub](#) page!
- Login to Kylo with our new form-based authentication. A logout option has been added to the upper-right menu in both the Feed Manager and the Operations Manager.

RPM

<http://bit.ly/2l5p1tK>

Upgrade Instructions from v0.6.0

Build or download the rpm.

1. Shut down NiFi:

```
service nifi stop
```

2. Run:

```
useradd -r -m -s /bin/bash kylo
```

3. Run:

```
usermod -a -G hdfs kylo
```

4. Run:

```
/opt/thinkbig/remove-kylo-datalake-accelerator.sh to uninstall  
the RPM
```

5. Install the new RPM:

```
rpm -ivh <RPM_FILE>
```

6. Migrate the “thinkbig” database schema to “kylo”.

Kylo versions 0.6 and below use the thinkbig schema in MySQL. Starting from version 0.7, Kylo uses the kylo schema. This guide is intended for installations that are already on Kylo 0.6, and want to upgrade to Kylo 0.7. It outlines the procedure for migrating the current thinkbig schema to kylo schema, so that Kylo 0.7 can work.

Migration Procedure

- 6a. Uninstall Kylo 0.6 (Refer to deployment guide and release notes for details).
- 6b. Install Kylo 0.7 (Refer to deployment guide and release notes for details).

Do not yet start Kylo services.

6c. Log into MySQL instance used by Kylo, and list the schemas:

```
mysql> show databases
```

6d. Verify that:

- thinkbig schema exists
- kylo schema does not exist

6e. Navigate to Kylo's setup directory for MySQL.

```
cd /opt/kylo/setup/sql/mysql
```

6f. Execute the migration script. It takes 3 parameters. For no password, provide the 3rd parameter as `../migrate-schema-thinkbig-to-kylo-mysql.sh <host> <user> <password>`

- Step 1 of migration: kylo schema is set up.
- Step 2 of migration: thinkbig schema is migrated to kylo schema.

6g. Start Kylo services. Verify that Kylo starts and runs successfully. At this point, there are two schemas in MySQL: kylo and thinkbig.

Once Kylo is running normally and migration is verified, the thinkbig schema can be dropped.

6h. Navigate to Kylo's setup directory for MySQL.

```
cd /opt/kylo/setup/sql/mysql
```

6i. Execute the script to drop thinkbig schema. It takes 3 parameters. For no password, provide the 3rd parameter as:

```
../drop-schema-thinkbig-mysql.sh <host> <user> <password>
```

6j. Verify that only kylo schema now exists in MySQL.

```
mysql> show databases
```

This completes the migration procedure.

7. Update the database:

```
/opt/kylo/setup/sql/mysql/kylo/0.7.0/update.sh localhost root <password or ↵  
↵blank>
```

8. Run:

```
/opt/kylo/setup/nifi/update-nars-jars.sh
```

9. Edit:

```
/opt/nifi/current/conf/bootstrap.conf
```

Change `"java.arg.15=Dthinkbig.nifi.configPath=/opt/nifi/ext-config"` to `"java.arg.15=Dkylo.nifi.configPath=/opt/nifi/ext-config"`.

10. Run:

```
mv /opt/thinkbig/bkup-config /opt/kylo
chown -R kylo:kylo bkup-config
```

11. Run:

```
mv /opt/thinkbig/encrypt.key /opt/kylo
```

If prompted for overwrite, answer ‘yes’.

12. Run:

```
chown kylo:kylo /opt/kylo/encrypt.key
```

13. Copy the mariadb driver to access MySQL database.

14. Run:

```
> cp /opt/kylo/kylo-services/lib/mariadb-java-client-*.jar /opt/nifi/mysql
> chown nifi:users /opt/nifi/mysql/mariadb-java-client-*.jar
```

15. Start NiFi (wait to start):

```
service nifi start
```

16. In the standard-ingest template, update the “Validate and Split Records” processor and change the Application-JAR value to:

```
/opt/nifi/current/lib/app/kylo-spark-validate-cleanse-jar-with-dependencies.
↪ jar
```

17. In the standard-ingest template update the “Profile Data” processor and change the ApplicationJAR value to:

```
/opt/nifi/current/lib/app/kylo-spark-job-profiler-jar-with-dependencies.jar
```

18. For the MySQL controller service (type: DBCPConnectionPool), update the properties to use the mariadb driver:

- **Database Driver Class Name:** org.mariadb.jdbc.Driver
- **Database Driver Location(s):** file:///opt/nifi/mysql/mariadb-java-client-1.5.7.jar

19. For the JMSConnectionFactoryProvider controller service, set the *MQ Client Libraries path* property value to:

```
/opt/kylo/kylo-services/lib
```

20. For the StandardSqoopConnectionService, copy the value of *Source Driver* to *Source Driver (Avoid providing value)* then delete the *Source Driver* property.

21. Update, with your custom configuration, the configuration files at:

```
/opt/kylo/kylo-ui/conf/, /opt/kylo/kylo-services/conf/
/opt/kylo/kylo-spark shell/conf/
```

A backup of the previous version’s configuration is available from /opt/kylo/bkup-config/.

22. Modify both of the metadata controller services in NiFi with the new REST endpoint.

- The first one should be under the root process group and is used by our processors. The REST Client URL property should be changed to <http://localhost:8400/proxy/v1/metadata>.

- The second is under the right-hand menu and is used by our reporting task. The REST Client URL property should be changed to <http://localhost:8400/proxy/v1/metadata>.

23. If using NiFi v0.7 or earlier, modify:

```
/opt/kylo/kylo-services/conf/application.properties
```

Change `spring.profiles.active` from **nifi-v1** to **nifi-v0**.

24. Modify permissions to allow existing NiFi flows to use `/tmp/kylo` directory.

Note:

After re-importing `data_ingest.zip` in a later step, any new feeds created will use the `/tmp/kylo-nifi` folder. The below command will allow existing flows to continue using the `/tmp/kylo` folder.

```
> chmod 777 /tmp/kylo
```

25. Start kylo apps:

```
/opt/kylo/start-kylo-apps.sh
```

26. Re-import the `data_ingest.zip` template. (New feeds will use the temp location `/tmp/kylo-nifi`.)

27. (Optional) If unused, the mysql driver in `/opt/nifi/mysql` can be deleted.

28. Run:

```
> rm /opt/nifi/mysql/mysql-connector-java-*.jar
```

Release 0.6.2 (Feb. 7, 2017)

Highlights

- Support for triggering multiple dependent feeds
- Added a flag to allow operations manager to query and display NiFi bulletins on feed failure to help show any logs NiFi generated during that execution back in operations manager
- Fixed NiFi provenance reporting to support manual emptying of flow files which will now fail the job in ops manager
- The Audit Log table in Kylo will now track feed updates

Upgrade Instructions from v0.6.0

Build or download the RPM.

1. Shut down NiFi:

```
service nifi stop
```

2. To uninstall the RPM, run:

```
/opt/kylo/remove-kylo-datalake-accelerator.sh
```

3. Install the new RPM:

```
rpm -ivh <RPM_FILE>
```

4. Run:

```
/opt/thinkbig/setup/nifi/update-nars-jars.sh
```

5. Start NiFi: (wait to start)

```
service nifi start
```

6. Update, using your custom configuration, the configuration files at:

```
/opt/thinkbig/thinkbig-ui/conf/
/opt/thinkbig/thinkbig-services/conf/
/opt/thinkbig/thinkbig-spark-shell/conf/
```

A backup of the previous version's configuration is available from `/opt/thinkbig/bkup-config/`.

7. If using NiFi v0.7 or earlier, modify `/opt/thinkbig/thinkbig-services/conf/application.properties` by changing `spring.profiles.active` from `nifi-v1` to `nifi-v0`.

8. Start thinkbig apps:

```
/opt/thinkbig/start-thinkbig-apps.sh
```

9. Ensure the reporting task is configured A ReportingTask is now used for communication between NiFi and Operations Manager. In order to see Jobs and Steps in Ops Manager, you will need to configure this following instructions found here:

NiFi & Kylo Reporting Task

Whats coming in 0.7.0?

The next release will be oriented to public open-source release and select issues identified by the field for client projects.

The approximate release date is February 13, 2017.

Release 0.6.1 (Jan. 26, 2017)

Highlights

- Improved NiFi provenance reporting performance
- Added timeout option to the NiFi ExecuteSparkJob processor
- Fixed missing Cloudera dependency
 - To build for Cloudera, substitute “thinkbig-service-monitor-ambari” with “thinkbig-service-monitor-cloudera-service” in `services/service-app/pom.xml`

Potential Impacts

Upon upgrading the ExecuteSparkJob processors will be marked as invalid saying: “Max wait time is invalid property”. You will need to stop these processors and delete the “Max wait time” property.

Upgrade Instructions from v0.6.0

Build or download the RPM:

1. Shut down NiFi:

```
service nifi stop
```

2. To uninstall the RPM, run:

```
/opt/thinkbig/remove-thinkbig.sh
```

3. Install the new RPM:

```
rpm -ivh <RPM_FILE>
```

4. Run:

```
/opt/thinkbig/setup/nifi/update-nars-jars.sh
```

5. Start NiFi: (wait to start)

```
service nifi start
```

6. Update, using your custom configuration, the configuration files at:

```
/opt/thinkbig/thinkbig-ui/conf/  
/opt/thinkbig/thinkbig-services/conf/  
/opt/thinkbig/thinkbig-spark-shell/conf/
```

A backup of the previous version's configuration is available from `/opt/thinkbig/bkup-config/`.

7. If using NiFi v0.7 or earlier, modify `/opt/thinkbig/thinkbig-services/conf/application.properties` by changing `spring.profiles.active` from `nifi-v1` to `nifi-v0`.

8. Start thinkbig apps: -

```
/opt/thinkbig/start-thinkbig-apps.sh
```

9. Update the ExecuteSparkJob processors (Validate and Profile processors) fixing the error: "Max wait time is invalid property" by removing that property.
10. Ensure the reporting task is configured A ReportingTask is now used for communication between NiFi and Operations Manager. In order to see Jobs and Steps in Ops Manager you will need to configure this following the instructions found here:

NiFi & Kylo Reporting Task

Release 0.6.0 (Jan. 19, 2017)

Highlights

- 90+ issues resolved
- NiFi clustering support. Ability to cluster NiFi with Kylo.
- Streaming enhancements. New streaming UI plots and higher throughput performance between Kylo and NiFi. Ability to specify a feed as a streaming type to optimize display.

- Improved schema manipulation. Ability for feeds and target Hive tables to diverge (for example: drop fields, rename fields, and change data types for fields that exist in raw files regardless of raw type).
- Alert persistence. Ability for alert panel to store alerts (and clear) including and APIs for plugging in custom alert responder and incorporate SLA alerts.
- Configurable data profiling. Profiled columns can be toggled to avoid excessive Spark processing.
- Tags in search. Ability to search tags in global search.
- Legacy NiFi version cleanup. Deletes retired version of NiFi feed flows.
- Avro format option for database fetch. GetTableData processor has been updated to allow writing rows in Avro format and to allow setting a custom column delimiter when the output type is a delimited text file.
- Feed file upload. Ability to upload a file directly to a feed and have it trigger immediately (for feeds using filesystem).
- Tutorials. New admin tutorial videos.

Potential Impacts

- JMS topics switch to queues in order to support NiFi clustering. Check your ActiveMQ Topics page (<http://localhost:8161/admin/topics.jsp>) to ensure that there are no pending messages before shutting down NiFi. The number of enqueued and dequeued messages should be the same.
- Apache NiFi versions 0.6 and 0.7 are no longer supported. Some features may continue to function normally but haven't been properly tested. These will stop working in future releases. Upgrading to the latest version of Apache NiFi is recommended.
- (for VirtualBox sandbox upgrades) The default password for MySQL has changed. If you are using default config files deployed via RPM, modify your MySQL password to match or alter the configuration files.

Upgrade Instructions from v0.5.0

Build or download the RPM:

1. Shut down NiFi:

```
service nifi stop
```

2. Run the following to uninstall the RPM:

```
/opt/thinkbig/remove-thinkbig.sh
```

3. Install the new RPM:

```
rpm -ivh <RPM_FILE>
```

4. Run:

```
/opt/thinkbig/setup/nifi/update-nars-jars.sh
```

5. Update /opt/nifi/current/conf/nifi.properties file and change it to use the default PersistentProvenanceRepository:

```
nifi.provenance.repository.implementation=org.apache.nifi.provenance.  
↪PersistentProvenanceRepository
```

6. Execute the database upgrade script:

```
/opt/thinkbig/setup/sql/mysql/thinkbig/0.6.0/update.sh localhost root <password or ↵  
↵blank>
```

7. Create the “/opt/nifi/activemq” folder and copy the jars:

```
$ mkdir /opt/nifi/activemq  
  
$ cp /opt/thinkbig/setup/nifi/activemq/*.jar  
/opt/nifi/activemq  
  
$ chown -R nifi /opt/nifi/activemq/
```

8. Add a service account for thinkbig application to nifi group. (This will allow Kylo to upload files to the dropzone location defined in NiFi). This step will differ per operating system. Note also that these may differ depending on how the service accounts were created.

```
$ sudo usermod -a -G nifi thinkbig
```

Note: All dropzone locations must allow the thinkbig service account to write.

9. Start NiFi: (wait to start)

```
service nifi start
```

Note: If errors occur, try removing the transient provenance data: `rm -fR /PATH/TO/NIFI/provenance_repository/`.

10. Update, using your custom configuration, the configuration files at:

```
/opt/thinkbig/thinkbig-ui/conf/  
/opt/thinkbig/thinkbig-services/conf/  
/opt/thinkbig/thinkbig-spark-shell/conf/
```

A backup of the previous version’s configuration is available from `/opt/thinkbig/bkup-config/`.

11. If using NiFi v0.7 or earlier, modify `/opt/thinkbig/thinkbig-services/conf/application.properties` by changing `spring.profiles.active` from `nifi-v1` to `nifi-v0`.

12. Start thinkbig apps:

```
/opt/thinkbig/start-thinkbig-apps.sh
```

13. Update the re-usable standard-ingest template, `index_schema_service`, and the `index_text_service`.

1. The standard-ingest template can be updated through the templates page. (`/opt/thinkbig/setup/data/templates/nifi-1.0/`) The upgrade will:
 - (a) Add “json field policy file” path as one of the parameters to Profiler processor to support selective column profiling. See “Configurable data profiling” in highlights.
 - (b) Add feed field specification to support UI ability to modify feeds. See “Improved schema manipulation” in highlights above.
 - (c) Adds shared library path to activemq libraries required going forward.
2. The `index_schema_service` and `index_text_service` templates are feed templates and should be updated through the feeds page. (`/opt/thinkbig/setup/data/feeds/nifi-1.0/`).

- (a) Go to the feeds page.
 - (b) Click the Plus icon.
 - (c) Click on the “import from file” link.
 - (d) Choose one of the Elasticsearch templates and check the overwrite box.
14. A ReportingTask is now used for communication between NiFi and Operations Manager. In order to see Jobs and Steps in Ops Manager you will need to configure this following these instructions:

NiFi & Kylo Reporting Task

Release 0.5.0 (Dec. 14, 2016)

Highlights

- 65 issues resolved
- Audit tracking. All changes in Kylo are tracked for audit logging.
- Spark 2.0 support!
- PySparkExec support. New NiFi processor for executing Spark Python scripts
- Plug-in API for adding raw formats. Ability to plug-in support for new raw file formats and introspect schema
- New raw formats: Parquet, ORC, Avro, JSON
- Customize partition functions. Ability to add custom UDF functions to dropdown for deriving partition keys
- Feed import enhancements. Allow users to change target category on feed import
- Sqoop improvements. Improved compatibility with Kylo UI and behavior
- JPA conversion. Major conversion away from legacy Spring Batch persistence to JPA for Ops Mgr
- Date/time standardization. Storage of all dates and times will be epoch time to preserve the ability to apply timezones
- New installation document showing an example on how to install Kylo on AWS in an HDP 2.5 cluster. Refer to *HDP 2.5 Kerberos/Ranger Cluster Deployment Guide*
- Ranger enabled
- Kerberos enabled
- Minimal admin privileges
- NiFi and Kylo on separate edge nodes

Known Issues

Modeshape versioning temporarily disabled for feeds due to rapid storage growth. We will have a fix for this issue and re-introduce it in 0.5.1.

Potential Impacts

- JPA conversion requires one-time script (see install instructions)
- Spark Shell moved into Think Big services /opt directory

- Date/time modification Timestamp fields converted to Java time for portability and timezone consistency. Any custom reports will need to be modified

Release 0.4.3 (Nov. 18, 2016)

Highlights

- 67 issues resolved
- Hive user impersonation. Ability to restrict Hive table access throughout Kylo based on permissions of logged-in user.
- Visual data lineage. Visualization of relationship between feeds, data sources, and sinks. Refer to [Feed Lineage Configuration](#)
- Auto-layout NiFi feeds. Beautified display of Kylo-generated feeds in NiFi.
- Sqoop export. Sqoop export and other Sqoop improvements from last release.
- Hive table formats. Final Hive table format extended to: RCFILE, CSV, AVRO (in addition to ORC, PARQUET).
- Hive change tracking. Batch timestamp (processing_dttm partition value) carried into final table for change tracking.
- Delete, disable, reorder templates. Ability to disable and/or remove templates as well as change their order in Kylo.
- Spark yarn-cluster support. ExecuteSparkJob processor now supports yarn-cluster mode (thanks Prav!).
- Kylo logo replaces Teradata Thinkbig logo (note: this is not our final approved logo).

Known Issues

Hive impersonation is not supported with CDH if using Sentry.

Wrangler does not yet support user impersonation.

Potential Impacts

- Existing wrangler feed tables will need to ALTER TABLE to add a processing_dttm field to table in order to work.
- Processing_dttm field is now java epoch time instead of formatted date to be timezone independent. Older feeds will now have partition keys in two different formats.
- All non-feed tables will now be created as managed tables.

Release 0.4.2 (Nov. 4, 2016)

Highlights

- 70-plus issues resolved
- NiFi version 1.0 and HDF version 2.0 support

- Encrypted properties and passwords in configuration files. Refer to “Encrypting Configuration Property Values” in the *Deployment Guide*
- SSL support. SSL between services. Refer to *NiFi and SSL*
- Feed-chaining context. Context can be passed from dependent feeds. Refer to the Trigger Feed section in *NiFi Processor Guide*
- Lineage tracking. Schema, feed, and preconditions.
- UI/UX improvements
- CSVSerde support and improved schema discovery for text files
- NiFi Template upgrades
- Procedure for relocating install locations of Kylo and dependencies. Refer to *TAR File Installation*

Release 0.4.1 (Oct. 20, 2016)

Highlights

- Resolved approximately 65 issues
- Ranger and Sentry integration (ability to assign groups to feed tables)
- NEW Sqoop import processor for efficient database ingest (tested with Sqoop version 1.4.6, Databases-Teradata, Oracle, and MySQL)
- Watermark service provides support for incremental loads
- Hive merge on Primary Key option
- Skip header support
- Configurable root paths for Hive and HDFS folders (multi-tenancy phase I)
- New and simplified standard ingest and re-usable wrangler flows
- Support for Hive decimal type
- Support for choosing existing field as partition
- Documentation updates
- UI usability improvements (validation, etc)

Known Issues

Creating a feed using standard data ingest with Database as the input may fail on initial run. There are 2 workarounds you can do to resolve this:

1. Go to the “Feed Details” screen for the feed and disable and then enable the feed; or,
2. During creation of the feed on the last “Schedule” step you can uncheck the “Enable Feed Immediately”. This will save the feed in a “disabled state”. Once the feed has been created on the Success screen click “View Details” then enable the feed.

Release 0.4.0 (Oct. 4, 2016)

Highlights

- Support Streaming/Rapid Fire feeds from NiFi

Note: Operations Manager User Interfaces for viewing Streaming feeds will come in a near future release.

- Security enhancements including integration with LDAP and administration of users and groups through the web UI
- Business metadata fields can be added to categories and feeds
- Category and feed metadata can be indexed with Elasticsearch, Lucene, or Solr for easier searching
- Fixed bug with kylo init.d service scripts not support the startup command
- Fixed issues preventing preconditions or cleanup feeds from triggering
- Fixed usability issues with the visual query
- Better error notification and bug fixes when importing templates
- Service level agreement assessments are now stored in our relational metadata store
- Spark Validator and Profiler NiFi processors can now handle additional Spark arguments
- Redesign of job details page in operations manager to view steps/details in vertical layout
- Allow injection of properties for any processor or controller service in the application.properties file. The feed properties will be overridden when importing a template. This includes support to auto fill all kerberos properties.

Known Issues

- The Data Ingest and Data Transformation templates may fail to import on a new install. You will need to manually start the *SpringContextLoaderService* and the *Kylo Cleanup Service* in NiFi, then re-import the template in the Feed Manager.
- When deleting a Data Transformation feed, a few Hive tables are not deleted as part of the cleanup flow and must be deleted manually.

Running in the IDE

- If you are running things via your IDE (Eclipse or IntelliJ) you will need to run the following command under the **core/operational-metadata/operational-metadata-jpa** module
- `mvn generate-sources`

This is because it is now using JPA along with QueryDSL(<http://www.querydsl.com/>), which generates helper Query classes for the JPA entities. Once this runs you will notice it generates a series of Java classes prefixed with “Q” (i.e. `QNifiJobExecution`) in the **core/operational-metadata/operational-metadata-jpa/target/generated-sources/java/**

Optionally you could just run a `mvn install` on this module which will also trigger the `generate-sources`.

- Additionally, if you haven't done so, you need to ensure the latest `nifi-provenance-repo.nar` file is in the `/opt/nifi/data/lib` folder.

Release 0.3.2 (Sept. 19, 2016)

Highlights

- Fixes a few issues found in version 0.3.1.
- Removed thinkbig, nifi, and activemq user creation from RPM install and installation scripts. Creating those users are now a manual process to support clients who use their own user management tools.
- Kerberos support for the UI features (data wrangling, hive tables, feed profiling page). Data wrangling uses the thinkbig user keytab and the rest uses the hive user keytab.
- Fixed bug introduced in 0.3.1 where the nifi symbolic link creation is broken during a new installation.
- Added support for installation Elasticsearch on SUSE.

Note: The activemq download URL was changed. To manually update the installation script edit: `/opt/thinkbig/setup/activemq/install-activemq.sh` and change the URL on line 25 to be <https://archive.apache.org/dist/activemq/5.13.3/apache-activemq-5.13.3-bin.tar.gz>

Release 0.3.1 (Aug. 17, 2016)

Highlights

- Fixes a few issues found in version 0.3.0.
- Fixes the download link to NiFi for generating an offline tar file.
- Compatibility with MySQL 5.7.
- Installs a stored procedure required for deleting feeds.
- PC-393 Automatically reconnects to the Hive metastore.
- PC-396 Script to update NiFi plugins and required JARs.

Note: A bug was introduced with installation of NiFi from the setup wizard (Fixed in the 0.4.0-SNAPSHOT). If installing a new copy of PCNG, make the following change:

Edit `/opt/kylo/setup/nifi/install-kylo-components.sh` and change `"/create-symbolic-links.sh"` to `"$NIFI_SETUP_DIR/create-symbolic-links.sh"`

Release 0.3.0 (Aug. 10, 2016)

Highlights

- 65 issues resolved by the team
- **Feed migration.** Import/export feeds across environments
- **Feed delete.** Delete a feed (including tables and data)
- **Business metadata.** Ability to add user-defined business metadata to categories and feeds
- **Feed chaining.** Chain feeds using UI-driven precondition rules

- **SLA support.** Create Service Level Agreements in UI
- **Alerting.** Alert framework and built-in support for JIRA and email
- **Profiling UI.** New graphical UI for viewing profile statistics
- **Wrangler XML support.** Wrangler enhancements including improved XML support
- **Authentication.** Pluggable authentication support

Release 0.2.0 (June 22, 2016)

Whats New

Release data: June 22, 2016

R&D is pleased to announce the release of version 0.2.0 of the framework, which represents the last three weeks of sprint development.

- Over 60 issues were resolved by the team working in collaboration with our International teams using the framework for client projects.
- Dependency on Java 8
- Updated metadata server to ModeShape framework, which supports many of our underlying architectural requirements:
 - Dynamic schemas - provides extensible features for extending schema towards custom business metadata in the field
 - Versioning - ability to track changes to metadata over time
 - Text Search - flexible searching metastore
 - Portability - can run on sql and nosql databases
 - See: <http://modeshape.jboss.org/>

Deployment Guides

Master

Deployment Guide

About

This document provides procedures for installing the Kylo framework, as well as Elasticsearch, NiFi, and ActiveMQ. Installation options allow new users to choose the method best suited to their interests and installation environments:

- **Setup Wizard** - For local development and single node development boxes, the [Setup Wizard Deployment Guide](#) can be used to quickly bootstrap your environment to get you up and running.
- **Manual** - In a test and production environment, you will likely be installing on multiple nodes. The [Manual Deployment Guide](#) provides detailed instructions on how to install each individual component.

For advanced users, there are additional installation options:

- **Cloudera EC2 Docker Sandbox** – The [Cloudera Docker Sandbox Deployment Guide](#) details options for those who want to deploy Kylo to a single node Cloudera sandbox in AWS. This is useful when you need to get a quick Cloudera instance running to test Kylo but don't have the resources to install a Cloudera cluster.
- **TAR File** – The previous install options are Red-Hat Package Manager (RPM) installations, but TAR File installation is available for those who want to install Kylo in a folder other than /opt/kylo, or want to run Kylo as a different user. See the [TAR File Installation](#).
- **HDP 2.5 Cluster Ranger/Kerberos with 2 Edge Nodes** - Kylo may also be installed, with minimal admin privileges, on an HDP 2.5 cluster. A procedure is provided for configuring an installation with NiFi on a separate edge node. See the [HDP 2.5 Kerberos/Ranger Cluster Deployment Guide](#).

Whichever installation option you choose, refer to the System Requirements and Prerequisites sections of this Kylo Deployment Guide to verify that your system is prepared for a Kylo Installation.

System Requirements

Dependencies

Kylo services should be installed on an edge node. The following should be available prior to the installation.

See the Dependencies section in the deployment checklist: *Deployment Checklist*

Platform	URL	Version
Hortonworks Sandbox	https://hortonworks.com/products/sandbox/	HDP 2.3,2.4,2.5
Cloudera Sandbox	https://www.cloudera.com/downloads/quickstart_vms/5-8.html	5.8

Prerequisites

Hortonworks Sandbox

If you are installing in a new Hortonworks sandbox, make sure to do the following first before running through the installation steps below.

Hortonworks Sandbox Configuration

Java Requirements

Kylo requires Java 8 for NiFi, kylo-ui, and kylo-services. If you already have Java 8 installed as the system level Java you have the option to leverage that.

In some cases, such as with an HDP install, Java 7 is the system version and you likely will not want to change it to Java 8. In this case you can leverage the mentioned scripts below to download and configure Java 8 in the /opt/java directory. The scripts will also modify the startup scripts for NiFi, kylo-ui and kylo-services to reference the /opt/java JAVA_HOME.

If you already have Java 8 installed in a different location you will have the option to use that as well.

Note: When installing the RPM the applications are defaulted to use the /opt/java/current location. This default saves a step for developers so that they can uninstall and re-install the RPM without having to run any other scripts.

Linux Users

If you will be creating Linux users as part of the install, the commands will be documented. If using an external user management system for adding users to a linux box, you must have those users created before installing the Kylo stack. You will need a user/group including:

- activemq
- elasticsearch
- kylo
- nifi

Note: Those exact names are required (note the lowercase).

Configuration

Configuration for Kylo services are located under the following files:

```
/opt/kylo/kylo-ui/conf/application.properties  
/opt/kylo/kylo-services/conf/application.properties
```

Ranger / Sentry

If you've changed the default Ranger or Sentry permissions, then you will need to add permissions for Kylo and NiFi.

Enable Ranger Authorization

Enable Sentry Authorization

Kerberos

If you are installing Kylo on a Kerberos cluster, you will need to configure the applications before certain features will work

Optional: Configure Kerberos For Your Local HDP Sandbox

This guide will help you enabled Kerberos for your local development sandbox for development and testing:

Kerberos Installation Example - Cloudera

Step 1: Configure Kerberos for NiFi

Some additional configuration is required for allowing the NiFi components to work with a Kerberos cluster.

NiFi Configuration for a Kerberos Cluster

Step 2: Configure Kerberos for Kylo Applications

Additional configuration is required for allowing some features in the Kylo applications to work with a Kerberos cluster.

Configuration for a Kerberos Cluster

SUSE Configuration

If you are installing Kylo on SUSE, please read the following document to work around ActiveMQ and Elasticsearch issues.

SUSE Configuration Changes

Encrypting Configuration Property Values

By default, a new Kylo installation does not have any of its configuration properties encrypted. Once you have started Kylo for the first time, the easiest way to derive encrypted versions of property values is to post values to the Kylo services/encrypt endpoint to have it generate an encrypted form for you. You could then paste the encrypted value back into your properties file and mark it as encrypted by prepending the values with {cipher}. For instance, if you wanted to encrypt the Hive datasource password specified in application.properties (assuming the password is “mypassword”), you can get its encrypted form using the curl command like this:

```
$ curl -u dladmin:thinkbig -H "Content-Type: text/plain; charset=UTF-8" \
→localhost:8400/proxy/v1/feedmgr/util/encrypt -d mypassword
29fcf1534a84700c68f5c79520ecf8911379c8b5ef4427a696d845cc809b4af0
```

You then copy that value and replace the clear text password string in the properties file with the encrypted value:

```
hive.datasource.password={cipher}
→29fcf1534a84700c68f5c79520ecf8911379c8b5ef4427a696d845cc809b4af0
```

The benefit of this approach is that you will be getting a value that is guaranteed to work with the encryption settings of the server where that configuration value is being used. Once you have replaced all properties you wish to have encrypted in the properties files, you can restart the Kylo services to use them.

Optimizing Performance

You can adjust the memory setting for each services using the below environment variables:

```
/opt/kylo/kylo-ui/bin/run-kylo-ui.sh
export KYLO_UI_OPTS= -Xmx4g

/opt/kylo/kylo-services/bin/run-kylo-services.sh
export KYLO_SERVICES_OPTS= -Xmx4g
```

The setting above would set the Java maximum heap size to 4 GB.

Change the Java Home

By default, the kylo-services and kylo-ui application set the JAVA_HOME location to /opt/java/current. This can easily be changed by editing the JAVA_HOME environment variable in the following two files:

```
/opt/kylo/kylo-ui/bin/run-kylo-ui.sh
/opt/kylo/kylo-services/bin/run-kylo-services.sh
```

In addition, if you run the script to modify the NiFi JAVA_HOME variable you will need to edit:

```
/opt/nifi/current/bin/nifi.sh
```

S3 Support For Data Transformations

Spark requires additional configuration in order to read Hive tables located in S3. Please see the [Accessing S3 from the Data Wrangler](#) how-to article.

Starting and Stopping the Services Manually

If you follow the instructions for the installations steps above, all of the below applications will be set to startup automatically if you restart the server. In the Hortonworks sandbox, the services for Kylo and NiFi are set to start after all of the services managed by Ambari have started.

To start and stop the three Kylo services, run the following scripts:

```
/opt/kylo/start-kylo-apps.sh
/opt/kylo/stop-kylo-apps.sh
```

1. To Start Individual Services:

```
$ service activemq start
$ service elasticsearch start
$ service nifi start
$ service kylo-spark-shell start
$ service kylo-services start
$ service kylo-ui start
```

2. To Stop individual services:

```
$ service activemq stop
$ service elasticsearch stop
$ service nifi stop
$ service kylo-spark-shell stop
$ service kylo-services stop
$ service kylo-ui stop
```

3. To get the status of individual services \$ service activemq status:

```
$ service elasticsearch status
$ service nifi status
$ service kylo-spark-shell status
$ service kylo-services status
$ service kylo-ui status
```

Log Output

Configuring Log Output

Log output for the services mentioned above are configured at:

```
/opt/kylo/kylo-ui/conf/log4j.properties
/opt/kylo/kylo-services/conf/log4j.properties
```

You may place logs where desired according to the 'log4j.appender.file.File' property. Note the configuration line:

```
log4j.appender.file.File=/var/log/<app>/<app>.log
```

Viewing Log Output

The default log locations for the various applications are located at:

```
/var/log/<service_name>
```

Web and REST Access

Below are the default URL's and ports for the services:

```
Feed Manager and Operations UI
http://127.0.0.1:8400
username: dladmin
password: thinkbig
```

```
NiFi UI
http://127.0.0.1:8079/nifi
```

```
Elasticsearch REST API
http://127.0.0.1:9200
```

```
ActiveMQ Admin
http://127.0.0.1:8161/admin
```

Appendix: Cleanup scripts

For development and sandbox environments you can leverage the cleanup script to remove all of the Kylo services as well as Elasticsearch, ActiveMQ, and NiFi.

```
$ /opt/kylo/setup/dev/cleanup-env.sh
```

Important: Only run this in a DEV environment. This will delete all application and the MySQL schema.

In addition there is a script for cleaning up the Hive schema and HDFS folders that are related to a specific “category” that is defined in the UI.

```
$ /opt/kylo/setup/dev/cleanupCategory.sh [categoryName]

Example: /opt/kylo/setup/dev/cleanupCategory.sh customers
```

Appendix: Postgres Integration

Postgres Metastore Configuration

Deployment Checklist

This document provides a sample checklist to help prepare for a Kylo deployment.

Edge Node Resource Requirements

- Kylo and Apache NiFi can be installed on a single edge node, however it is recommended that they run on separate edge nodes.

- Minimum production recommendation is 4 cores CPU, 16 GB RAM.
- Preferred production recommendation is 8 cores CPU, 32 GB RAM.

Dependencies

Red Hat Enterprise Linux or other GNU/Linux Distributions (CentOS, Fedora).
RPM (for install)
Java 1.8+
Hadoop 2.4+ cluster
Spark 1.5.2+
Apache NiFi 1.x+ (or HDF 2.x)
Hive 1.2.x+
MySQL 5.x+

Linux User/Group Creation

There are three Linux user accounts that need to be created before installing the Kylo stack. If an external user management tool is used, these user accounts need to be created ahead of time. If not, there are instructions in the *Manual Deployment Guide* (see Step 2: Create Linux Users and Groups) on how to create the users and groups.

- kylo
- nifi
- activemq

Please create the above users and groups with the same names.

Edge Node Ports

The following ports are required to be open for browser access unless using a web proxy server:

- Kylo UI – 8400
- NiFi – 8079
- ActiveMQ JMS – 61616 (only if on a different edge node than NiFi or Kylo)

The following is optional:

- ActiveMQ Admin – 8161

Cluster Host Names, User Names, and Ports

Collect the following information to speed up configuration of Kylo:

- Hive Hostname/IP Address:
- Ambari IP Hostname/IP Address (if used):
- Ambari “kylo” user username/password (if used):
- KDC Hostname/IP Address (if used):
- MySQL Metastore Hostname/IP Address:
- Kylo Edge Hostname/IP Address:

- NiFi Edge Hostname/IP Address:
- Kylo MySQL Installation User username/password (Create Schema Required):
- Kylo MySQL application username/password (For the kylo-services application and Hive metadata access):

Kerberos Principals (if using Kerberos)

Note the following Kerberos principals after the step of creating the Keytabs:

- Kerberos Principal for “kylo”:
- Kerberos Principal for “nifi”:
- Kerberos Principal for “hive” on the Hive Server2 Host:

Dependencies

Component Versions

Below is a list of some of the major components Kylo uses along with the version that Kylo currently supports:

Cat-e-gory	Item	Version	Description
Pers-istence	MySQL	5.x (tested with 5.1.73)	Used to store both the Modeshape (JCR 2.0) metadata and the Operational Relational (Kylo Ops Manager) metadata
Pers-istence	Post-gres	9.x	Not fully supported yet. Kylo Operations Manager piece should work in Postgres; however we haven't fully tested it with Feed Manager. (See below on the Persistence Note)
Pers-istence	Mode-shape	5.0	Jboss Modeshape Java Content Repository (JCR 2.0)
JMS	Ac-tiveMq	5.x (tested with 5.13.3)	Used to send messages between different modules and to send Provenance from NiFi to Kylo
NiFi	NiFi	1.0,(HDF 2.0)	Either HDF or open source NiFi work.
Spark	Spark	1.5.x, 1.6.x, 2.x	NiFi and Kylo have routines that leverage Spark.
UI	Tom-cat	8.0.32	Tomcat is the default engine for Spring Boot. If needed Spring Boot allows you to change to a different server (i.e. Jetty) but this hasn't been tested.
Java	Java	Java 8_92+	The Kylo install will setup its own Java Home so it doesn't affect any other Java versions running on the machine.
Search	Elas-tic-search	2.3.x	For index and search of Hive metadata and indexing feed data when selected as part of creating a feed
OS	Linux	Various	Tested with RHEL and CentOS 6.x, 7.x, SUSE v11

Service Accounts

Required new linux service accounts are listed below. Within enterprises there are often approvals required and long lead times to obtain service accounts. Kerberos principals are required where the service interacts with a Kerberized

Hadoop cluster. These services are not typically deployed to control and data nodes. The Nifi, activemq, Elastic services and Kylo metastore databases (mysql or postgres) are IO intensive.

Service	Purpose	Local Linux Users	Local Linux Groups	Keytab file	upn	spn
kylo-services	Kylo Coordinator	kylo	kylo, hdfs or supergroup	/etc/security/keytabs/kylo-headless.keytab		
kylo-ui	Provides Kylo feed and operations user interface	kylo	kylo, hdfs or supergroup			
nifi	Orchestrate data flows	nifi	nifi, hdfs or supergroup	/etc/security/keytabs/nifi-headless.keytab		
activemq	Broker messages between components	activemq	activemq			
elastic	Manages searchable index	elastic	elastic			
mysql or postgres	Metastore for Kylo feed manager and operational metadata	mysql or postgres	mysql or postgres			

Persistence Usage

Kylo captures and stores three types of metadata:

1. Setup/Configuration metadata. This data is captured in the Kylo Feed Manager which describes how Feeds, Categories, Templates, etc are structured.
 - (a) This is stored using Java Content Repository (JCR 2.) using Modeshape as the JCR implementation and persisted to MySQL.
2. Operational/Transactional metadata. This data is captured by the system when it processes data, feeds, slas, etc.
 - (a) This is stored in MySQL.
3. Searchable data index. This data is captured by the data flows where 'Index' field check box is marked.
 - (a) This is stored in Elasticsearch.

Install Guides

Setup Wizard Deployment Guide

About

Follow the steps below to install Kylo using the installation wizard script. This is convenient for local sandboxes (HDP/Cloudera) and single node development boxes. The WGET command is used to download binaries so internet access is required.

Note: The setup wizard is designed for easy installation of all components on one node.

Installation Locations

Installing Kylo installs the following software at these Linux file system locations:

- Kylo Applications - /opt/kylo
- Java 8 - /opt/java/current
- NiFi - /opt/nifi/current
- ActiveMQ - /opt/activemq
- Elasticsearch - RPM installation default location

Installation

The steps below require root access.

Step 1: Download the RPM

Download the RPM and place it on the host Linux machine that you want to install Kylo services on.

Note: To use wget instead, right-click the download link and copy the url.

Download the Latest RPM

```
http://bit.ly/2r4P47A
```

Step 2: Create the Linux Users/Groups

Creation of users and groups is done manually because many organizations have their own user and group management system. Therefore, it cannot be scripted as part of the RPM install. Here is an example of how to create the users and groups:

```
$ useradd -r -m -s /bin/bash nifi
$ useradd -r -m -s /bin/bash kylo
$ useradd -r -m -s /bin/bash activemq
```

Validate that the above commands created a group by looking at /etc/group. Some operating systems may not create them by default.

```
$ cat /etc/group
```

If the groups are missing then run the following:

```
$ groupadd kylo
$ groupadd nifi
$ groupadd activemq
```


Step 3: Run the Kylo RPM Install

```
$ rpm -ivh kylo-<version>.noarch.rpm
```

Note: The RPM is hard coded at this time to install to /opt/kylo.

Step 4: Optional - Generate TAR file for Offline Mode

To run the wizard on an edge node with no internet access, generate a TAR file that contains everything in the /opt/kylo/setup folder including the downloaded application binaries.

1. Install the Kylo RPM on a node that has internet access.
2. Run the offline install:

```
$ /opt/kylo/setup/generate-offline-install.sh
```

Note If installing the Debian packages make sure to change the Elasticsearch download from RPM to DEB

3. Copy the /opt/kylo/setup/kylo-install.tar file to the node you install the RPM on. This can be copied to a temp directory. It doesn't have to be put in the /opt/kylo/setup folder
4. Run the Kylo TAR install:

```
tar -xvf kylo-install.tar
```

The script downloads all application binaries and puts them in their respective directory in the setup folder. Last it will TAR up the setup folder.

Step 5: Run the Setup Wizard

Note: If installing in an HDP or Cloudera sandbox, choose option #2 on the Java step to download and install Java in the /opt/java/current directory.

1. From the /opt/kylo/setup directory

```
$ /opt/kylo/setup/setup-wizard.sh
```

2. Offline mode from another directory (using TAR file)

```
$ <PathToSetupFolder>/setup/setup-wizard.sh -o
```

Note: Both -o and -O work.

Follow the directions to install the following:

- MySQL or Postgres scripts into the local database
- Elasticsearch
- ActiveMQ

- Java 8 (If the system Java is 7 or below)
- NiFi and the Kylo dependencies

The Elasticsearch, NiFi, and ActiveMQ services start when the wizard is finished.

Step 6: Add “nifi” and “kylo” Users

In this step, add “nifi” and “kylo” users to the HDFS supergroup, or to the group defined in `hdfs-site.xml`. For example:

Hortonworks

```
$ usermod -a -G hdfs nifi
$ usermod -a -G hdfs kylo
```

Cloudera

```
$ groupadd supergroup
# Add nifi and hdfs to that group:
$ usermod -a -G supergroup nifi
$ usermod -a -G supergroup hdfs
```

Optional: If you want to perform actions as a root user in a development environment run the below command:

```
$ usermod -a -G supergroup root
```

Step 7: Additional Cluster Configuration

In addition to adding the nifi/kylo user to the supergroup on the edge node, add the users/groups to the name nodes on a cluster.

Hortonworks

```
$ useradd kylo
$ useradd nifi
$ usermod -G hdfs nifi
$ usermod -G hdfs kylo
```

Cloudera

```
TBD (need to test this out)
```

Step 8: Create a Dropzone Folder

For example:

```
$ mkdir -p /var/dropzone
$ chown nifi /var/dropzone
```

Note: Files should be copied into the dropzone such that user nifi can read and remove.

Step 9: Cloudera Configuration (Cloudera Only)

See the appendix section below “Cloudera Configuration File Changes”.

Step 10: Edit the Properties Files

Step 11: Start the Three Kylo Services

```
$ /opt/kylo/start-kylo-apps.sh
```

At this point, all services should be running. Note that services are started automatically on boot.

Appendix: Cloudera Configuration File Changes

The configuration is setup to work out of the box with the Hortonworks sandbox. There are a few differences that require configuration changes for Cloudera.

1. /opt/kylo/kylo-services/conf/application.properties
 - (a) Update the 3 MySQL password values to “cloudera”:

```
spring.datasource.password=cloudera
metadata.datasource.password=cloudera
hive.metastore.datasource.password=cloudera
modeshape.datasource.password=cloudera
```

2. Update the Hive username:

```
hive.datasource.username=hive
```

3. Update the Hive Metastore URL:

```
hive.metastore.datasource.url=jdbc:mysql://localhost:3306/metastore
```

4. Update the following parameters:

```
config.hive.schema=metastore
nifi.executesparkjob.sparkhome=/usr/lib/spark
```

Manual Deployment Guide

Preface

This document explains how to install each component of the Kylo framework manually. This is useful when you are installing across multiple edge nodes. Use this link to the install wizard ([Setup Wizard Deployment Guide](#)) if you would prefer not to do the installation manually.

Note: Many of the steps below are similar to running the wizard-based install. If you want to take advantage of the same scripts as the wizard, you can tar up the /opt/kylo/setup folder and untar it to a temp directory on each node.

Audience

This guide provides step-by-step instruction for installing Kylo. The reader is assumed to be a systems administrator, with knowledge of Linux.

Refer to the Kylo Operations Guide (*Operations Guide*) for detailed instructions on how to effectively manage:

- Production processing
- Ongoing maintenance
- Performance monitoring

For enterprise support options, please visit:

<http://kylo.io/>

Installation Components

Installing Kylo installs the following software:

- **Kylo Applications:** Kylo provides services to produce Hive tables, generate a schema based on data brought into Hadoop, perform Spark-based transformations, track metadata, monitor feeds and SLA policies, and publish to target systems.
- **Java 8:** Kylo uses the Java 8 development platform.
- **NiFi:** Kylo uses Apache NiFi for orchestrating data pipelines.
- **ActiveMQ:** Kylo uses Apache ActiveMQ to manage communications with clients and servers.
- **Elasticsearch:** Kylo uses Elasticsearch, a distributed, multi-tenant capable full-text search engine.

Installation Locations

Installing Kylo installs the following software at these Linux file system locations:

- Kylo Applications - /opt/kylo
- Java 8 - /opt/java/current
- NiFi - /opt/nifi/current
- ActiveMQ - /opt/activemq
- Elasticsearch - RPM installation default location

Installation

For each step below, you will need to login to the target machine with root access permissions. Installation commands will be executed from the command-line interface (CLI).

Step 1: Setup Directory

Kylo is most often installed on one edge node. If you are deploying everything to one node, the setup directory would typically be:

```
SETUP_DIR=/opt/kylo/setup
```

Sometimes administrators install NiFi on a second edge node to communicate with a Hortonworks or Cloudera cluster. In this case, copy the setup folder to nodes that do not have the Kylo applications installed. In that case, use this `SETUP_DIR` command:

```
SETUP_DIR=/tmp/kylo-install
```

Optional - Offline Mode

If an edge node has no internet access, you can generate a TAR file that contains everything in the `/opt/kylo/setup` folder, including the downloaded application binaries.

1. Install the Kylo RPM on a node that has internet access.

```
$ rpm -ivh kylo-<version>.noarch.rpm.rpm
```

2. Run the script, which will download all application binaries and put them in their respective directory in the setup folder.

```
$ /opt/kylo/setup/generate-offline-install.sh
```

Note If installing the Debian packages make sure to change the Elasticsearch download from RPM to DEB

3. Copy the `/opt/kylo/setup/kylo-install.tar` file to the node you install the RPM on. This can be copied to a temp directory. It doesn't have to be put in the `/opt/kylo/setup` folder.
4. Run the command to tar up the setup folder.

```
tar -xvf kylo-install.tar
```

5. Note the directory name where you untar'd the files. This will be referred to in the rest of the doc by `OFFLINE_SETUP_DIR`.

Step 2: Create Linux Users and Groups

Creation of users and groups is done manually because many organizations have their own user and group management system. Therefore we cannot script it as part of the RPM install.

Note: Each of these should be run on the node on which the software will be installed. If a machine will run nifi, kylo and activemq, all users/groups should be created. If running individual services, only the appropriate user/group for that service should be created, not all of them.

To create all the users and groups on a single machine, run the following command:

To create individual users, run the following commands on the appropriate machines:

The following command can be used to confirm if the user and group creation was successful:

This command should give two results per user, one for the user in `/etc/passwd` and one in `/etc/group`. For example, if you added all the users to an individual machine, there should be six lines of output. If you just added an individual user, there will be two lines of output.

If the groups are missing, they can be added individually:

If all groups are missing, they can be all added with the following command:

Step 3: Install Kylo Services

1. Download the RPM and place it on the host Linux machine that you want to install Kylo services on.

Note: To use `wget` instead, right-click the download link and copy the url.

[Download the latest Kylo RPM](#)

2. Run the Kylo RPM install.

```
$ rpm -ivh kylo-<version>.noarch.rpm
```

Note: The RPM is hard coded at this time to install to `/opt/kylo`.

Step 4: Run the database scripts

The database scripts will create one schema called “kylo” and install to that schema. Run the following script:

```
$ <SETUP_DIR>/sql/mysql/setup-mysql.sh [db_host_or_ip] [db_user] [db_password]
```

Note: If `db_user` does not have password, the `db_password` can be provided as “”. (For example, if using HDP 2.4 sandbox)

Step 5: Install and Configure Elasticsearch

To get Kylo installed and up and running quickly, a script is provided to stand up a single node Elasticsearch instance. You can also leverage an existing Elasticsearch instance. For example, if you stand up an ELK stack you will likely want to leverage the same instance.

Option 1: Install Elasticsearch from our script.

Note: The included Elasticsearch script was meant to speed up installation in a sandbox or DEV environment.

1. Online Mode

```
$ <SETUP_DIR>/elasticsearch/install-elasticsearch.sh
```

2. Offline Mode

```
$ <OFFLINE_SETUP_DIR>/elasticsearch/install-elasticsearch.sh -o <OFFLINE_SETUP_DIR>
```

```
Example: /tmp/kylo-install/setup/elasticsearch/install-elasticsearch.sh -o /tmp/kylo-install/setup
```

Option 2: Use an existing Elasticsearch. To leverage an existing Elasticsearch instance, you must update all feed templates that you created with the correct Elasticsearch URL. You can do this by going to the “Additional Properties” tab for that feed. If you added any reusable flow templates you will need to modify the Elasticsearch processors in NiFi.

Note: Tip: To test that Elasticsearch is running type “curl localhost:9200”. You should see a JSON response.

Step 6: Install ActiveMQ

Another script has been provided to stand up a single node ActiveMQ instance. You can also leverage an existing ActiveMQ instance.

Option 1: Install ActiveMQ from the script

Note: The included ActiveMQ script was meant to speed up installation in a sandbox or DEV environment. It is not a production ready configuration.

1. Online Mode

```
$ <SETUP_DIR>/activemq/install-activemq.sh
```

2. Offline Mode

```
$ <OFFLINE_SETUP_DIR>/activemq/install-activemq.sh -o <OFFLINE_SETUP_DIR>
```

Example: /tmp/kylo-install/setup/activemq/install-activemq.sh -o /tmp/kylo-install/
↪ setup

Note: If installing on a different node than NiFi and kylo-services you will need to update the following properties

```
1. /opt/nifi/ext-config/config.properties

    spring.activemq.broker-url
    (Perform this configuration update after installing NiFi, which is step 9 in_
    ↪ this guide)

2. /opt/kylo/kylo-services/conf/application.properties

   .jms.activemq.broker.url
    (By default, its value is tcp://localhost:61616)
```

Option 2: Leverage an existing ActiveMQ instance

Update the below properties so that NiFi and kylo-services can communicate with the existing server.

```
1. /opt/nifi/ext-config/config.properties

    spring.activemq.broker-url

2. /opt/kylo/kylo-services/conf/application.properties

   .jms.activemq.broker.url
```

Installing on SUSE

The deployment guide currently addresses installation in a Red Hat based environment. There are a couple of issues installing Elasticsearch and ActiveMQ on SUSE. Below are some instructions on how to install these two on SUSE.

- **ActiveMQ**

When installing ActiveMQ, you might see the following error:

Warning: ERROR: Configuration variable JAVA_HOME or JAVACMD is not defined correctly.
(JAVA_HOME='', JAVACMD='java')

This indicates that ActiveMQ isn't properly using Java as it is set in the system. To fix this issue, use the following steps to set the JAVA_HOME directly:

1. Edit /etc/default/activemq and set JAVA_HOME at the bottom.

```
JAVA_HOME=<location_of_java_home>
```

2. Restart ActiveMQ

```
$ service activemq restart
```

- **Elasticsearch**

RPM installation isn't supported on SUSE. To work around this issue, we created a custom init.d service script and wrote up a manual procedure to install Elasticsearch on a single node.

We have created a service script to make it easy to start and stop Elasticsearch, as well as leverage chkconfig to automatically start Elasticsearch when booting up the machine. Below are the instructions on how we installed Elasticsearch on a SUSE box.

```
1. Make sure Elasticsearch service user/group exists
2. mkdir /opt/elasticsearch
3. cd /opt/elasticsearch
4. mv /tmp/elasticsearch-2.3.5.tar.gz
5. tar -xvf elasticsearch-2.3.5.tar.gz
6. rm elasticsearch-2.3.5.tar.gz
7. ln -s elasticsearch-2.3.5 current
8. cp elasticsearch.yml elasticsearch.yml.orig
9. Modify elasticsearch.yml if you want to change the cluster name. The standard Kylo_
  ↳ installation scripts have this file in directory: /opt/kylo/setup/elasticsearch
10. chown -R elasticsearch:elasticsearch /opt/elasticsearch/
11. Uncomment and set the JAVA_HOME on line 44 of the file: /opt/kylo/setup/
  ↳ elasticsearch/init.d/sles/elasticsearch
12. vi /etc/init.d/elasticsearch - paste in the values from /opt/kylo/setup/
  ↳ elasticsearch/init.d/sles/elasticsearch
13. chmod 755 /etc/init.d/elasticsearch
```



```
14. chkconfig elasticsearch on
15. service elasticsearch start
```

Step 7: Install Java 8

Note: If you are installing NiFi and the kylo services on two separate nodes, you may need to perform this step on each node.

There are 3 scenarios for configuring the applications with Java 8.

Scenario 1: Java 8 is installed on the system and is already in the classpath.

In this case you need to remove the default JAVA_HOME used as part of the install. Run the following script:

```
For kylo-ui and kylo-services
$ <SETUP_DIR>/java/remove-default-kylo-java-home.sh
```

To test this you can look at each file referenced in the scripts for kylo-ui and kylo-services to validate the 2 lines setting and exporting the JAVA_HOME are gone.

Scenario 2: Install Java in the default /opt/java/current location.

Note: You can modify and use the following script to uninstall Java 8:

Online Mode

```
$ <SETUP_DIR>/java/install-java8.sh
```

Offline Mode

```
$ <OFFLINE_SETUP_DIR>/java/install-java8.sh -o <OFFLINE_SETUP_DIR>
```

```
Example: /tmp/kylo-install/setup/java/install-java8.sh -o /tmp/kylo-install/setup
```

Scenario 3: Java 8 is installed on the node, but it's not in the default JAVA_HOME path.

If you already have Java 8 installed, and want to reference that installation, there is a script to remove the existing path and another script to set the new path for the kylo apps.

```
For kylo-ui and kylo-services
$ /opt/kylo/setup/java/remove-default-kylo-java-home.sh
$ /opt/kylo/setup/java/change-kylo-java-home.sh <PATH_TO_JAVA_HOME>
```

Step 8: Install Java Cryptographic Extension

The Java 8 install script above will automatically download and install the [Java Cryptographic Extension](#). This extension is required to allow encrypted property values in the Kylo configuration files. If you already have a Java 8 installed on the system, you can install the Java Cryptographic Extension by running the following script:

```
$ <SETUP_DIR>/java/install-java-crypt-ext.sh <PATH_TO_JAVA_HOME>
```

This script downloads the extension zip file and extracts the replacement jar files into the JRE security directory (\$JAVA_HOME/jre/lib/security). It will first make backup copies of the original jars it is replacing.

Step 9: Install NiFi

You can leverage an existing NiFi installation or follow the steps in the setup directory that are used by the wizard.

Note: Note that Java 8 is required to run NiFi with our customizations. Make sure Java 8 is installed on the node.

Option 1: Install NiFi from our scripts.

This method downloads and installs NiFi, and also installs and configures the Kylo-specific libraries. This instance of NiFi is configured to store persistent data outside of the NiFi installation folder in /opt/nifi/data. This makes it easy to upgrade since you can change the version of NiFi without migrating data out of the old version.

1. Install NiFi in either online or offline mode:

Online Mode

```
$ <SETUP_DIR>/nifi/install-nifi.sh
```

Offline Mode

```
$ <OFFLINE_SETUP_DIR>/nifi/install-nifi.sh -o <OFFLINE_SETUP_DIR>
```

2. Update JAVA_HOME (default is /opt/java/current).

```
$ <SETUP_DIR>/java/change-nifi-java-home.sh <path to JAVA_HOME>
```

3. Install Kylo specific components.

```
$ <SETUP_DIR>/nifi/install-kylo-components.sh
```

Option 2: Leverage an existing NiFi instance

In some cases you may want to leverage separate instances of NiFi or Hortonworks Data Flow. Follow the steps below to include the Kylo resources.

Note: If Java 8 isn't being used for the existing instance, then you will be required to change it.

1. Copy the <SETUP_DIR>/nifi/kylo-*.nar and kylo-spark-.jar files to the node NiFi is running on. If it's on the same node you can skip this step.
2. Shutdown the NiFi instance.
3. Create folders for the jar files. You may choose to store the jars in another location if you want.

```
$ mkdir -p <NIFI_HOME>/kylo/lib
```

4. Copy the kylo-*.nar files to the <NIFI_HOME>/kylo/lib directory.
5. Create a directory called "app" in the <NIFI_HOME>/kylo/lib directory.

```
$ mkdir <NIFI_HOME>/kylo/lib/app
```

6. Copy the kylo-spark-*.jar files to the <NIFI_HOME>/kylo/lib/app directory.

7. Create symbolic links for all of the .NARs and .JARs. Below is an example of how to create it for one NAR file and one JAR file. At the time of this writing there are eight NAR files and three spark JAR files.

```
$ ln -s <NIFI_HOME>/kylo/lib/kylo-nifi-spark-nar-*.nar <NIFI_HOME>/lib/kylo-nifi-
↳spark-nar.nar

$ ln -s <NIFI_HOME>/kylo/lib/app/kylo-spark-interpreter-*.jar-with-dependencies.jar
  <NIFI_HOME>/lib/app/kylo-spark-interpreter-jar-with-dependencies.jar
```

8. Modify <NIFI_HOME>/conf/nifi.properties and update the port NiFi runs on.

```
nifi.web.http.port=8079
```

Note: If you decide to leave the port number set to the current value, you must update the “nifi.rest.port” property in the kylo-services application.properties file.

9. There is a controller service that requires a MySQL database connection. You will need to copy the driver jar to a location on the NiFi node. The pre-defined templates have the default location set to /opt/nifi/mysql.
 - (a) Create a folder to store the driver jar in.
 - (b) Copy the /opt/kylo/kylo-services/lib/mariadb-java-client-<version>.jar to the folder in step #1.
 - (c) If you created a folder name other than the /opt/nifi/mysql default folder you will need to update the “MySQL” controller service and set the new location. You can do this by logging into NiFi and going to the Controller Services section at root process group level.
10. Create an ext-config folder to provide JMS information and location of cache to store running feed flowfile data if NiFi goes down.

Note: Right now the plugin is hard coded to use the /opt/nifi/ext-config directory to load the properties file.

Configure the ext-config folder

1. Create the folder.

```
$ mkdir /opt/nifi/ext-config
```

2. Copy the /opt/kylo/setup/nifi/config.properties file to the /opt/nifi/ext-config folder.
3. Change the ownership of the above folder to the same owner that nifi runs under. For example, if nifi runs as the “nifi” user:

```
$ chown -R nifi:users /opt/nifi
```

11. Create an activemq folder to provide JARs required for the JMS processors.

Configure the activemq folder

1. Create the folder.

```
$ mkdir /opt/nifi/activemq
```

2. Copy the `/opt/kylo/setup/nifi/activemq/*.jar` files to the `/opt/nifi/activemq` folder.

```
$ cp /opt/kylo/setup/nifi/activemq/*.jar /opt/nifi/activemq
```

3. Change the ownership of the folder to the same owner that nifi runs under. For example, if nifi runs as the “nifi” user:

```
$ chown -R nifi:users /opt/nifi/activemq
```

OPTIONAL: The `/opt/kylo/setup/nifi/install-kylo-components.sh` contains steps to install NiFi as a service so that NiFi can startup automatically if you restart the node. This might be useful to add if it doesn’t already exist for the NiFi instance.

Step 10: Set Permissions for HDFS

This step is required on the node that NiFi is installed on to set the correct permissions for the “nifi” user to access HDFS.

1. NiFi Node - Add nifi user to the HDFS supergroup or the group defined in `hdfs-site.xml`, for example:

Hortonworks (HDP)

```
$ usermod -a -G hdfs nifi
```

Cloudera (CDH)

```
$ groupadd supergroup
# Add nifi and hdfs to that group:
$ usermod -a -G supergroup nifi
$ usermod -a -G supergroup hdfs
```

Note: If you want to perform actions as a root user in a development environment, run the below command.

```
$ usermod -a -G supergroup root
```

2. kylo-services node - Add kylo user to the HDFS supergroup or the group defined in `hdfs-site.xml`, for example:

Hortonworks (HDP)

```
$ usermod -a -G hdfs kylo
```

Cloudera (CDH)

```
$ groupadd supergroup
# Add nifi and hdfs to that group:
$ usermod -a -G supergroup hdfs
```

Note: If you want to perform actions as a root user in a development environment run the below command.

```
$ usermod -a -G supergroup root
```

3. For Clusters:

In addition to adding the nifi and kylo users to the supergroup on the edge node you also need to add the users/groups to the **NameNodes** on a cluster.

Hortonworks (HDP)

```
$ useradd kylo
$ useradd nifi
$ usermod -G hdfs nifi
$ usermod -G hdfs kylo
```

Cloudera (CDH) - <Fill me in after testing >

Step 11: Create a dropzone folder on the edge node for file ingest

Perform the following step on the node on which NiFi is installed:

```
$ mkdir -p /var/dropzone
$ chown nifi /var/dropzone
```

Note: Files should be copied into the dropzone such that user nifi can read and remove. Do not copy files with permissions set as root.

Complete this step for Cloudera installations ONLY

<Fill me in after testing Cloudera-specific configuration file changes>

Step 12: (Optional) Edit the Properties Files

If required for any specific customization, edit the properties files for Kylo services:

```
$ vi /opt/kylo/kylo-services/conf/application.properties
$ vi /opt/kylo/kylo-ui/conf/application.properties
```

Step 13: Final Step: Start the 3 Kylo Services

```
$ /opt/kylo/start-kylo-apps.sh
```

At this point all services should be running. Verify by running:

```
$ /opt/kylo/status-kylo-apps.sh
```

Cloudera Docker Sandbox Deployment Guide

About

In some cases, you may want to deploy a Cloudera sandbox in AWS for a team to perform a simple proof-of-concept, or to avoid system resource usage on the local computer. Cloudera offers a Docker image, similar to the Cloudera sandbox, that you download and install to your computer.

Warning: Once you create the docker container called “cloudera” do not remove the container unless you intend to delete all of your work and start cleanly. There are instructions below on how to start and stop an existing container to retain your data.

Prerequisites

You need access to an AWS instance and permission to create an EC2 instance.

Installation

Step 1: Create an EC2 instance

For this document, we will configure a CoreOS AMI which is optimized for running Docker images.

1. Choose an AMI for the region in which you will configure the EC2 instance.

Note: For detailed procedures for instance, visit [Running CoreOS Container Linux on EC2](#) on the CoreOS website.

2. Create the EC2 instance. You might want to add more disk space than the default 8GB.
3. Configure the EC2 security group.
4. After starting up the instance, Login to the EC2 instance:

```
$ ssh -i <private_key> core@<IP_ADDRESS>
```

Step 2: Create Script to Start Docker Container

Create a shell script to startup the Docker container. This makes it easier to create a new container if you decided to delete it at some point and start clean.

1. Start Cloudera:

```
$ vi startCloudera.sh
```

2. Add the following:

```
#!/bin/bash
docker run --name cloudera =
  --hostname=quickstart.cloudera \
  --privileged=true -t -d \
  -p 8888:8888 \
  -p 7180:7180 \
```

```
-p 80:80 \
-p 7187:7187 \
-p 8079:8079 \
-p 8400:8400 \
-p 8161:8161 \
cloudera/quickstart:5.7.0-0-beta /usr/bin/docker-quickstart
```

3. Change permissions:

```
$ chmod 744 startCloudera.sh
```

4. Start the Container:

```
$ /startCloudera.sh
```

It will have to first download the Docker image, which is about 4GB, so give it some time.

Step 3: Login to the Cloudera Container and Start Cloudera Manager

1. Login to the Docker container:

```
$ docker exec -it cloudera bash
```

2. Start Cloudera Manager:

```
$ /home/cloudera/cloudera-manager --express
```

3. Login to Cloudera Manager:

```
<EC2_HOST>:7180 (username/password is cloudera/cloudera)
```

4. Start all services in Cloudera Manager.

5. After it's started exit the container to go back to the CoreOS host.

Step 4: Build a Cloudera Distribution of Kylo and Copy it to the Docker Container

1. Modify the pom.xml file for the kylo-services-app module. Change:

```
<dependency>
<groupId>com.thinkbiganalytics.datalake</groupId>
<artifactId>kylo-service-monitor-ambari</artifactId>
<version>0.3.0-SNAPSHOT</version>
</dependency>

To

<dependency>
<groupId>com.thinkbiganalytics.datalake</groupId>
<artifactId>kylo-service-monitor-cloudera</artifactId>
<version>0.3.0-SNAPSHOT</version>
</dependency>
```

2. From the kylo root folder, run:

```
$ mvn clean install -o -DskipTests
```

3. Copy the new RPM file to the CoreOS box.

```
$ scp -i ~/.ssh/<EC2_PRIVATE_KEY>  
<DLA_HOME>/install/target/rpm/tkylo/RPMS/noarch/kylo  
core@<EC2_IP_ADDRESS>:/home/core
```

4. From the CoreOS host, copy the RPM file to the Docker container.

```
$ docker cp  
/home/core/kylo-<VERSION>.noarch.rpm  
cloudera:/tmp
```

Step 5: Install Kylo in the Docker Container

1. Login to the Cloudera Docker container.

```
$ docker exec -it cloudera bash  
  
$ cd /tmp
```

2. Create Linux Users and Groups.

Creation of users and groups is done manually because many organizations have their own user and group management system. Therefore we cannot script it as part of the RPM install.

```
$ useradd -r -m -s /bin/bash nifi  
$ useradd -r -m -s /bin/bash kylo  
$ useradd -r -m -s /bin/bash activemq
```

Validate the above commands created a group as well by looking at /etc/group. Some operating systems may not create them by default.

```
$ cat /etc/group
```

If the groups are missing then run the following:

```
$ groupadd kylo  
$ groupadd nifi  
$ groupadd activemq
```

3. Follow the instructions in the Deployment Wizard guide to install the RPM and other components.

Note: There is an issue installing the database script so say No to the wizard step asking to install the database script. We will do that manually. I will update this section when it's fixed.

4. Follow these steps, that are not in the wizard deployment guide but are required to run Kylo in this environment:
 - (a) Run the database scripts:

```
$ /opt/kylo/setup/sql/mysql/setup-mysql.sh root cloudera
```

2. Edit /opt/kylo/kylo-services/conf/application.properties:

Make the following changes in addition to the Cloudera specific changes, described in the Appendix section of the wizard deployment guide for Cloudera:

```
###Ambari Services Check
#ambariRestClientConfig.username=admin
#ambariRestClientConfig.password=admin
#ambariRestClientConfig.serverUrl=http://127.0.0.1:8080/api/v1
#ambari.services.status=HDFS,HIVE,MAPREDUCE2,SQOOP
###Cloudera Services Check
clouderaRestClientConfig.username=cloudera
clouderaRestClientConfig.password=cloudera
clouderaRestClientConfig.serverUrl=127.0.0.1
cloudera.services.status=HDFS/[DATANODE,NAMENODE],HIVE/[HIVEMETASTORE,HIVESERVER2],
↪YARN
##HDFS/[DATANODE,NAMENODE,SECONDARYNAMENODE],HIVE/[HIVEMETASTORE,HIVESERVER2],YARN,
↪SQOOP
```

3. Add the “kylo” user to the supergroup:

```
$ usermod -a -G supergroup kylo
```

4. Run the following commands to address an issue with the Cloudera Sandbox and fix permissions.

```
$ su - hdfs
$ hdfs dfs -chmod 775 /
```

5. Start up the Kylo Apps:

```
$ /opt/kylo/start-kylo-apps.sh
```

6. Try logging into <EC2_HOST>:8400 and <EC2_HOST>:8079.

Shutting down the container when not in use

EC2 instance can get expensive to run. If you don't plan to use the sandbox for a period of time, we recommend shutting down the EC2 instance. Here are instructions on how to safely shut down the Cloudera sandbox and CoreOS host.

1. Login to Cloudera Manager and tell it to stop all services.
2. On the CoreOS host, type “docker stop cloudera”.
3. Shutdown the EC2 Instance.

Starting up an Existing EC2 instance and Cloudera Docker Container

1. Start the EC2 instance.
2. Login to the CoreOS host.
3. Type “docker start cloudera” to start the container.
4. SSH into the docker container.

```
$ docker exec -it cloudera bash
```

5. Start Cloudera Manager.

```
$ /home/cloudera/cloudera-manager --express
```

6. Login to Cloudera Manager and start all services.

TAR File Installation

Introduction

At this time, an RPM file is the only artifact built into Kylo. An RPM installation is meant to be an opinionated way of installing an application, and it reduces the number of steps required to complete the installation. That said, some clients have strict requirements as to where they need to install Kylo, and the user that Kylo must be run under. These instructions will guide you through an alternative way to install Kylo, if that is required in your case.

Determine Service Account and Kylo Install Location

Let's assume, for this example, that Kylo will run under an account name "kylo_user", and it will be installed in /opt/apps/.

Step 1: Install the RPM and copy the /opt/kylo folder to a temporary location.

```
cp -R /opt/kylo /opt/tb-test
```

Step 2: Copy init.d scripts to the same temporary location.

```
cp /etc/init.d/kylo-services /opt/tb-test
cp /etc/init.d/kylo-spark-shell /opt/tb-test
cp /etc/init.d/kylo-ui /opt/tb-test
```

Your temp location should look like this:

```
[root@sandbox tb-test]# ls -l /opt/tb-test
drwxr-xr-x 8 root root 4096 2016-10-27 20:13 kylo
-rwxr-xr-x 1 root root 1561 2016-10-27 20:20 kylo-services
-rwxr-xr-x 1 root root 1281 2016-10-27 20:21 kylo-spark-shell
-rwxr-xr-x 1 root root 1447 2016-10-27 20:21 kylo-ui
```

Step 3 (Optional): Tar up the folder and copy it to the edge node if you aren't already on it.

Step 4: Install the files.

1. Copy the Kylo folder to /opt/apps.
2. Copy the 3 init.d scripts to /etc/init.d.

Step 5: Create Log Folders

```
[root@sandbox tb-test]# mkdir /var/log/kylo-services
[root@sandbox tb-test]# chown kylo_user:kylo_user
/var/log/kylo-services
[root@sandbox tb-test]# mkdir /var/log/kylo-ui
[root@sandbox tb-test]# chown kylo_user:kylo_user
/var/log/kylo-ui
[root@sandbox tb-test]# mkdir /var/log/kylo-spark-shell
[root@sandbox tb-test]# chown kylo_user:kylo_user
/var/log/kylo-spark-shell/
```

Step 6: Modify the user in the init.d scripts.

Set this line to be the correct user:

```
RUN_AS_USER=kylo_user
```

Also change any reference of /opt/kylo to /opt/apps/kylo.

Step 7: Modify the bin scripts for the 3 kylo apps.

Modify the following files and change /opt/kylo references to /opt/kylo/apps:

```
/opt/apps/kylo/kylo-ui/bin/run-kylo-ui.sh
/opt/apps/kylo/kylo-services/bin/run-kylo-services.sh
/opt/apps/kylo/kylo-spark-shell/bin/run-kylo-spark-shell.sh
```

Step 8: Start up Kylo and test.

HDP 2.5 Kerberos/Ranger Cluster Deployment Guide

About

This guide will help you understand the steps involved with deploying Kylo to a Kerberos cluster with minimal admin privileges. No super user privileges will be provided to the “nifi” or “kylo” user. The only usage for an administrative account will be for kylo-services to access the Ranger REST API.

There are two ways you can configure Hive to manage users on the cluster.

1. You can configure it to run Hive jobs as the end user, but all HDFS access is done as the Hive user.
2. Run Hive jobs and HDFS commands as the end user.

Note: For detailed information on refer to Best Practices for Hive Authorization Using Apache Ranger in HDP 2.2 on the Hortonworks website.

This document will configure option #2 to show how you can configure Kylo to grant appropriate access to both Hive and HDFS for the end user.

Cluster Topography

The cluster used in this example contains the following:

- 3 master nodes
- 3 data nodes
- 1 Kylo edge node
- 1 NiFi edge node

There are a couple of things to notes about the cluster:

- The cluster is leveraging the MIT KDC for Kerberos.
- The cluster uses Linux file system-based authorization (not LDAP or AD).

Known Issues

Kylo does not support Hive HA Thrift URL connections yet. If the cluster is configured for HA and zookeeper, you will need to connect directly to the thrift server.

You may see an error similar to the following:

Error: Requested user nifi is not whitelisted and has id 496, which is below the minimum allowed 500”.

If you do, do the following to change the user ID or lower the minimum ID:

1. Login to Ambari and edit the yarn “Advanced yarn-env”.
2. Set the “Minimum user ID for submitting job” = 450.

Prepare a Checklist

Leverage the deployment checklist to take note of information you will need to speed up configuration.

Deployment Checklist

Prepare the HDP Cluster

Before installing the Kylo stack, prepare the cluster by doing the following:

1. Ensure that Kerberos is enabled.
2. Enable Ranger, including the HDFS and Hive plugin.
3. Change Hive to run both Hive and HDFS as the end user.
 - (a) Login to Ambari.
 - (b) Go to Hive → Config.
 - (c) Change “Run as end user instead of Hive user” to true.
 - (d) Restart required applications.
4. Create an Ambari user for Kylo to query the status REST API’s.
 - (a) Login to Ambari.

- (b) Got to “Manage Ambari” → Users.
 - (c) Click on “Create Local User”.
 - (d) Create a user called “kylo” and save the password for later.
 - (e) Go to the “Roles” screen.
 - (f) Assign the “kylo” user to the “Cluster User” role.
5. If your Spark job fails when running in HDP 2.4.2 while interacting with an empty ORC table, you will get this error message:

Error: “ExecuteSparkJob[id=1fb1b9a0-e7b5-4d85-87d2-90d7103557f6] java.util.NoSuchElementException: next on empty iterator “

This is due to a change Hortonworks added to change how it loads the schema for the table. To fix the issue you can modify the following properties:

1. On the edge node edit /usr/hdp/current/spark-client/conf/spark-defaults.conf.
2. Add this line to the file “spark.sql.hive.convertMetastoreOrc false”

Optionally, rather than editing the configuration file you can add this property in Ambari:

1. Login to Ambari.
 2. Go to the Spark config section.
 3. Go to “custom Spark defaults”.
 4. Add the property “spark.sql.hive.convertMetastoreOrc” and set to “false”.
6. Create the “nifi” and “kylo” user on the master and data nodes.

Note: If the operations team uses a user management tool then create the users that way.

If you are using linux /etc/group based authorization in your cluster you are required to create any users that will have access to HDFS or Hive on the following:

Master Nodes:

```
$ useradd -r -m -s /bin/bash nifi
$ useradd -r -m -s /bin/bash kylo
```

Data Nodes: In some cases this is not required on data nodes.

```
$ useradd -r -m -s /bin/bash nifi
$ useradd -r -m -s /bin/bash kylo
```

Prepare the Kylo Edge Node

1. Install the MySQL client on the edge node, if not already there:

```
$ yum install mysql
```

2. Create a MySQL admin user or use root user to grant “create schema” access from the Kylo edge node.

This is required to install the “kylo” schema during Kylo installation.

Example:

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'KYLO_EDGE_NODE_HOSTNAME' IDENTIFIED BY 'abc123'  
↪ ' WITH GRANT OPTION; FLUSH PRIVILEGES;
```

3. Create the “kylo” MySQL user.

```
CREATE USER 'kylo'@'<KYLO_EDGE_NODE>' IDENTIFIED BY 'abc123';  
grant create, select, insert, update, delete, execute ON kylo.* to kylo'@'KYLO_EDGE_  
↪NODE_HOSTNAME';  
FLUSH PRIVILEGES;
```

4. Grant kylo user access to the hive MySQL metadata.

```
GRANT select ON hive.SDS TO 'kylo'@'KYLO_EDGE_NODE_HOSTNAME';  
GRANT select ON hive.TBLS TO 'kylo'@'KYLO_EDGE_NODE_HOSTNAME';  
GRANT select ON hive.DBS TO 'kylo'@'KYLO_EDGE_NODE_HOSTNAME';  
GRANT select ON hive.COLUMNS_V2 TO 'kylo'@'KYLO_EDGE_NODE_HOSTNAME';
```

Note: If the Hive database is installed in a separate MySQL instance, you will need to create the “kylo” non privileged user in that database before running the grants.

5. Make sure the Spark client and Hive client is installed.**6. Create the “kylo” user on edge node.**

```
Kylo Edge Node:  
$ useradd -r -m -s /bin/bash kylo  
$ useradd -r -m -s /bin/bash activemq
```

7. Optional - Create offline TAR file for an offline Kylo installation.

```
[root]# cd /opt/kylo/setup/  
[root setup]# ./generate-offline-install.sh
```

Copy the TAR file to both the Kylo edge node as well as the NiFi edge node.

8. Prepare a list of feed categories you wish to create.

This is required due to the fact that we are installing Kylo without privileged access. We will create Ranger policies ahead of time to all Kylo access to the Hive Schema and HDFS folders.

9. Create “kylo” home folder in HDFS. This is required for Hive queries to work in HDP.

```
[root]$ su - hdfs  
[hdfs]$ kinit -kt /etc/security/keytabs/hdfs.headless.keytab <hdfs_principal_name>  
[hdfs]$ hdfs dfs -mkdir /user/kylo  
[hdfs]$ hdfs dfs -chown kylo:kylo /user/kylo  
[hdfs]$ hdfs dfs -ls /user
```

Tip: If you do not know the HDFS Kerberos principal name run “klist -kt/etc/security/keytabs/hdfs.headless.keytab”.

Prepare the NiFi Edge Node**1. Install the MySQL client on the edge node, if not already there.**

```
$ yum install mysql
```

2. Grant MySQL access from the NiFi edge node.

Example:

```
GRANT ALL PRIVILEGES ON *.* TO 'kylo'@'nifi_edge_node' IDENTIFIED BY 'abc123';
FLUSH PRIVILEGES;
```

3. Make sure the Spark client and Hive client is installed.
4. Create the “nifi” user on edge node, master nodes, and data nodes.

Edge Nodes:

```
$ useradd -r -m -s /bin/bash nifi
```

5. Optional - Copy the offline TAR file created above to this edge node, if necessary.
6. Create the “nifi” home folders in HDFS.

This is required for Hive queries to work in HDP.

```
[root]$ su - hdfs
[hdfs]$ kinit -kt /etc/security/keytabs/hdfs.headless.keytab <hdfs_principal_name>
[hdfs]$ hdfs dfs -mkdir /user/nifi
[hdfs]$ hdfs dfs -chown nifi:nifi /user/nifi
[hdfs]$ hdfs dfs -ls /user
```

Tip: If you don’t know the HDFS Kerberos principal name, run:

```
“klist -kt /etc/security/keytabs/hdfs.headless.keytab”
```

Create the Keytabs for “nifi” and “kylo” Users

1. Login to the host that is running the KDC and create the keytabs.

```
[root]# kadmin.local
kadmin.local: addprinc -randkey "kylo/<KYLO_EDGE_HOSTNAME>@US-WEST-2.COMPUTE.INTERNAL"
kadmin.local: addprinc -randkey "nifi/<NIFI_EDGE_HOSTNAME>@US-WEST-2.COMPUTE.INTERNAL"
kadmin.local: xst -k /tmp/kylo.service.keytab kylo/<KYLO_EDGE_HOSTNAME>@US-WEST-2.
↪COMPUTE.INTERNAL
kadmin.local: xst -k /tmp/nifi.service.keytab nifi/<NIFI_EDGE_HOSTNAME>@US-WEST-2.
↪COMPUTE.INTERNAL
```

2. Note the Hive principal name for the thrift connection later.

```
# Write down the principal name for Hive for the KDC node
kadmin.local: listprincs

kadmin.local: exit
```

3. Move the keytabs to the correct edge nodes.
4. Configure the Kylo edge node. This step assumes that, to configure the keytab, you SCP’d the files to:

```
/tmp
```

Configure the edge node:

```
[root opt]# mv /tmp/kylo.service.keytab /etc/security/keytabs/  
[root keytabs]# chown kylo:kylo /etc/security/keytabs/kylo.service.keytab  
[root opt]# chmod 400 /etc/security/keytabs/kylo.service.keytab
```

5. Test the keytab on the Kylo edge node.

```
[root keytabs]# su - kylo  
[kylo ~]$ kinit -kt /etc/security/keytabs/kylo.service.keytab kylo/<KYLO_EDGE_  
↪HOSTNAME>@US-WEST-2.COMPUTE.INTERNAL  
[kylo ~]$ klist  
[kylo ~]$ klist  
Ticket cache: FILE:/tmp/krb5cc_496  
Default principal: kylo/ip-172-31-42-133.us-west-2.compute.internal@US-WEST-2.COMPUTE.  
↪INTERNAL  
Valid starting Expires Service principal  
11/29/2016 22:37:57 11/30/2016 22:37:57 krbtgt/US-WEST-2.COMPUTE.INTERNAL@US-WEST-2.  
↪COMPUTE.INTERNAL  
  
[kylo ~]$ hdfs dfs -ls /  
Found 10 items ....  
  
# Now try hive  
[kylo ~]$ hive
```

6. Configure the NiFi edge node.

```
root opt]# mv /tmp/nifi.service.keytab /etc/security/keytabs/  
[root keytabs]# chown nifi:nifi /etc/security/keytabs/nifi.service.keytab  
[root opt]# chmod 400 /etc/security/keytabs/nifi.service.keytab
```

7. Test the keytab on the NiFi edge node.

```
[root keytabs]# su - nifi  
[nifi ~]$ kinit -kt /etc/security/keytabs/nifi.service.keytab nifi/i<NIFI_EDGE_  
↪HOSTNAME>@US-WEST-2.COMPUTE.INTERNAL  
[nifi ~]$ klist  
Ticket cache: FILE:/tmp/krb5cc_497  
Default principal: nifi/<NIFI_EDGE_HOSTNAME>@US-WEST-2.COMPUTE.INTERNAL  
Valid starting Expires Service principal  
11/29/2016 22:40:08 11/30/2016 22:40:08 krbtgt/US-WEST-2.COMPUTE.INTERNAL@US-WEST-2.  
↪COMPUTE.INTERNAL  
  
[nifi ~]$ hdfs dfs -ls /  
Found 10 items
```



```
[nifi ~]$ hive
```

8. Test with Kerberos test client.

Kylo provides a kerberos test client to ensure the keytabs work in the JVM. There have been cases where kinit works on the command line but getting a kerberos ticket breaks in the JVM.

```
https://github.com/kyloio/kylo/tree/master/core/kerberos/kerberos-test-client
```

9. Optional - Test Beeline connection.

Install NiFi on the NiFi Edge Node

1. SCP the kylo-install.tar tar file to /tmp (if running in offline mode).
2. Run the setup wizard (example uses offline mode) [root tmp]# cd /tmp

```
[root tmp]# mkdir tba-install
[root tmp]# mv kylo-install.tar tba-install/
[root tmp]# cd tba-install/
[root tba-install]# tar -xvf kylo-install.tar

[root tba-install]# /tmp/tba-install/setup-wizard.sh -o
```

3. Install the following using the wizard.

- NiFi
- Java (Option #2 most likely)

4. Stop NiFi.

```
$ service nifi stop
```

5. Edit nifi.properties to set Kerberos setting.

```
[root]# vi /opt/nifi/current/conf/nifi.properties

nifi.kerberos.krb5.file=/etc/krb5.conf
```

6. Edit the config.properties file.

```
[root]# vi /opt/nifi/ext-config/config.properties

jms.activemq.broker.url=tcp://<KYLO_EDGE_HOST>:61616
```

7. Start NiFi.

```
[root]# service nifi start
```

8. Tail the logs to look for errors.

```
tail -f /var/log/nifi/nifi-app.log
```

Install the Kylo Application on the Kylo Edge Node

1. Install the RPM.

```
$ rpm -ivh /tmp/kylo-<VERSION>.noarch.rpm
```

2. SCP the kylo-install.tar tar file to /tmp (if running in offline mode).
3. Run the setup wizard (example uses offline mode)

```
[root tmp]# cd /tmp
[root tmp]# mkdir tba-install
[root tmp]# mv kylo-install.tar tba-install/
[root tmp]# cd tba-install/
[root tba-install]# tar -xvf kylo-install.tar
```

```
[root tba-install]# /tmp/tba-install/setup-wizard.sh -o
```

4. Install the following using the wizard (everything but NiFi).

- MySQL database scripts
- Elasticsearch
- ActiveMQ
- Java (Option #2 most likely)

5. Update Elasticsearch configuration.

In order for Elasticsearch to allow access from an external server you need to specify the hostname in addition to localhost.

```
$ vi /etc/elasticsearch/elasticsearch.yml
network.host: localhost,<KYLO_EDGE_HOST>
```

```
$ service elasticsearch restart
```

6. Edit the thinbig-spark-shell configuration file.

```
[root kylo]# vi /opt/kylo/kylo-services/conf/spark.properties
```

```
kerberos.kylo.kerberosEnabled=true
kerberos.kylo.hadoopConfigurationResources=/etc/hadoop/conf/core-site.xml,/etc/hadoop/
↪ conf/hdfs-site.xml
```

```
kerberos.kylo.kerberosPrincipal=<kylo_principal_name>
kerberos.kylo.keytabLocation=/etc/security/keytabs/kylo.service.keytab
```

7. Edit the kylo-services configuration file.

```
[root /]# vi /opt/kylo/kylo-services/conf/application.properties
```

```
spring.datasource.url=jdbc:mysql://<MYSQL_HOSTNAME>:3306/kylo?
↳noAccessToProcedureBodies=true
spring.datasource.username=kylo
spring.datasource.password=password

ambariRestClientConfig.host=<AMBARI_SERVER_HOSTNAME>
ambariRestClientConfig.username=kylo
ambariRestClientConfig.password=password

metadata.datasource.url=jdbc:mysql://<MYSQL_HOSTNAME>:3306/kylo?
↳noAccessToProcedureBodies=true
metadata.datasource.username=kylo
metadata.datasource.password=password

hive.datasource.url=jdbc:hive2://<HIVE_SERVER2_HOSTNAME>:10000/default;principal=
↳<HIVE_PRINCIPAL_NAME>

hive.metastore.datasource.url=jdbc:mysql://<MYSQL_HOSTNAME>:3306/hive
hive.metastore.datasource.username=kylo
hive.metastore.datasource.password=password
```

```
modeshape.datasource.url=jdbc:mysql://<MYSQL_HOSTNAME>:3306/kylo?
↳noAccessToProcedureBodies=true
modeshape.datasource.username=kylo
modeshape.datasource.password=password

nifi.rest.host=<NIFI_EDGE_HOST>

kerberos.hive.kerberosEnabled=true
kerberos.hive.hadoopConfigurationResources=/etc/hadoop/conf/core-site.xml,/etc/hadoop/
↳conf/hdfs-site.xml
kerberos.hive.kerberosPrincipal=<KYLO_PRINCIPAL_NAME>
```

```

kerberos.hive.keytabLocation=/etc/security/keytabs/kylo.service.keytab

nifi.service.mysql.database_user=kylo
nifi.service.mysql.password=password
nifi.service.mysql.database_connection_url=jdbc:mysql://<MYSQL_HOSTNAME>

nifi.service.hive_thrift_service.database_connection_url=jdbc:hive2://<HIVE_SERVER2_
↪HOSTNAME>:10000/default;principal=<HIVE_PRINCIPAL_NAME>
nifi.service.hive_thrift_service.kerberos_principal=<NIFI_PRINCIPAL_NAME>
nifi.service.hive_thrift_service.kerberos_keytab=/etc/security/keytabs/nifi.service.
↪keytab
nifi.service.hive_thrift_service.hadoop_configuration_resources=/etc/hadoop/conf/core-
↪site.xml,/etc/hadoop/conf/hdfs-site.xml

nifi.service.think_big_metadata_service.rest_client_url=http://<KYLO_EDGE_HOSTNAME>
↪:8400/proxy/metadata

nifi.executesparkjob.sparkmaster=yarn-cluster
nifi.executesparkjob.extra_jars=/usr/hdp/current/spark-client/lib/datanucleus-api-jdo-
↪3.2.6.jar,/usr/hdp/current/spark-client/lib/datanucleus-core-3.2.10.jar,/usr/hdp/
↪current/spark-client/lib/datanucleus-rdbms-3.2.9.jar
nifi.executesparkjob.extra_files=/usr/hdp/current/spark-client/conf/hive-site.xml

nifi.all_processors.kerberos_principal=<NIFI_PRINCIPAL_NAME>
nifi.all_processors.kerberos_keytab=/etc/security/keytabs/nifi.service.keytab
nifi.all_processors.hadoop_configuration_resources=/etc/hadoop/conf/core-site.xml,/
↪etc/hadoop/conf/hdfs-site.xml

```

Set the JMS server hostname for the Kylo hosted JMS server:

```
config.elasticsearch.jms.url=tcp://<KYLO_EDGE_HOST>:61616
```

8. Install the Ranger Plugin.

- (a) SCP Ranger plugin to /tmp.
- (b) Install the Ranger plugin.

```

[root plugin]# mv /tmp/kylo-hadoop-authorization-ranger-<VERSION>.jar /opt/kylo/kylo-
↪services/plugin
[root plugin]# chown kylo:kylo /opt/kylo/kylo-services/plugin/kylo-hadoop-
↪authorization-ranger-<VERSION>.jar
[root plugin]# touch /opt/kylo/kylo-services/conf/authorization.ranger.properties
[root plugin]# chown kylo:kylo /opt/kylo/kylo-services/conf/authorization.ranger.
↪properties

```

3. Edit the properties file.

```
vi /opt/kylo/kylo-services/conf/authorization.ranger.properties
```

```
ranger.hostName=<RANGER_HOST_NAME>
ranger.port=6080
ranger.userName=admin
ranger.password=admin
hdfs.repository.name=Sandbox_hadoop
hive.repository.name=Sandbox_hive
```

9. Start the Kylo applications.

```
[root]# /opt/kylo/start-kylo-apps.sh
```

10. Check the logs for errors.

```
/var/log/kylo-services.log
/var/log/kylo-ui/kylo-ui.log
/var/log/kylo-services/kylo-spark-shell.err
```

11. Login to the Kylo UI.

```
http://<KYLO_EDGE_HOSTNAME>:8400
```

Create Folders for NiFi standard-ingest Feed

1. Create the dropzone directory on the NiFi edge node.

```
$ mkdir -p /var/dropzone
$ chown nifi /var/dropzone
```

2. Create the HDFS root folders.

This will be required since we are running under non-privileged users.

```
[root]# su - hdfs
[hdfs ~]$ kinit -kt /etc/security/keytabs/hdfs.service.keytab
<HDFS_PRINCIPAL_NAME>
[hdfs ~]$ hdfs dfs -mkdir /etl
[hdfs ~]$ hdfs dfs -chown nifi:nifi /etl
[hdfs ~]$ hdfs dfs -mkdir /model.db
[hdfs ~]$ hdfs dfs -chown nifi:nifi /model.db
[hdfs ~]$ hdfs dfs -mkdir /archive
[hdfs ~]$ hdfs dfs -chown nifi:nifi /archive
[hdfs ~]$ hdfs dfs -mkdir -p /app/warehouse
[hdfs ~]$ hdfs dfs -chown nifi:nifi /app/warehouse
[hdfs ~]$ hdfs dfs -ls /
```

Create Ranger Policies

1. Add the “kylo” and “nifi user to Ranger if they don’t exist.
2. Create the HDFS NiFi policy.

- (a) Click into the HDFS repository
- (b) Click on “Add New Policy”

```
name: kylo-nifi-access
Resource Path:
  /model.db/*
  /archive/*
  /etl/*
  /app/warehouse/*
user: nifi
permissions: all
```

3. Create the Hive NiFi policy.

- (a) Click into the Hive repository.
- (b) Click on “Add New Policy”.

```
Policy Name: kylo-nifi-access
Hive Database: userdata, default (required for access for some reason)
table: *
column: *
user: nifi
permissions: all
```

4. Create the Hive Kylo policy.

Grant Hive access to “kylo” user for Hive tables, profile, and wrangler.

Note: Kylo supports user impersonation (add doc and reference it).

1. Click into the Hive repository.
 2. Click on “Add New Policy”.
-

```
Policy Name: kylo-kylo-access
Hive Database: userdata
table: *
column: *
user: kylo
permissions: select
```

Import Kylo Templates

1. Import Index Schema Template (For Elasticsearch).

- (a) Locate the `index_schema_service.zip` file. You will need the file locally to upload it. You can find it in one of two places:
 - i. `<kylo_project>/samples/feeds/nifi-1.0/`
 - ii. `/opt/kylo/setup/data/feeds/nifi-1.0`
- (b) Go to the the Feeds page in Kylo.
- (c) Click on the plus icon to add a feed.
- (d) Select “Import from a file”.

- (e) Choose the index_schema_service.zip file.
 - (f) Click “Import Feed”.
2. Update the Index Schema processors.
 - (a) Login to NiFi.
 - (b) Go to the system → index_schema_service process group
 - i. Edit the “Receive Schema Index Request” processor and set the URL value to <KYLO_EDGE_HOSTNAME>.
 - ii. In addition to the URL field you might have to edit the.jms-subscription property file as instructed above.
 - iii. Edit the “Index Metadata Elasticsearch” processor and set the HostName value to <KYLO_EDGE_HOSTNAME>.
 3. Import Index Text Template (For Elasticsearch).
 - (a) Locate the index_text_service.zip file. You will need the file locally to upload it. You can find it in one of two places:
 - <kylo_project>/samples/feeds/nifi-1.0/
 - /opt/kylo/setup/data/feeds/nifi-1.0
 - (b) Go to the the Feeds page in Kylo.
 - (c) Click on the plus icon to add a feed.
 - (d) Select “Import from a file”.
 - (e) Choose the index_text_service.zip file.
 - (f) Click “Import Feed”.
 4. Update the Index Text processors.
 - (a) Login to NiFi.
 - (b) Go to the system → index_text_service process group.
 - i. Edit the “Receive Index Request” processor and set the URL value to <KYLO_EDGE_HOSTNAME>.
 - ii. In addition to the URL field you might have to edit the.jms-subscription property file as instructed above.
 - iii. Edit the “Update Elasticsearch” processor and set the HostName value to <KYLO_EDGE_HOSTNAME>.

Note: An issue was found with the getJmsTopic processor URL. If you import the template using localhost and need to change it there is a bug that won’t allow the URL to be changed. The value is persisted to a file.

```
[root@ip-10-0-178-60 conf]# pwd
/opt/nifi/current/conf
[root@ip-10-0-178-60 conf]# ls -l
total 48
-rw-rw-r-- 1 nifi users 3132 Dec 6 22:05 bootstrap.conf
-rw-rw-r-- 1 nifi users 2119 Aug 26 13:51 bootstrap-notification-services.xml
-rw-rw-r-- 1 nifi nifi 142 Dec 7 00:36.jms-subscription-2bd64d8a-2b1f-1ef0-e961-
↪e50680e34686
```

```
-rw-rw-r-- 1 nifi nifi 142 Dec 7 00:54 jms-subscription-2bd64d97-2b1f-1ef0-7fc9-  
↪279eacf076dd  
-rw-rw-r-- 1 nifi users 8243 Aug 26 13:51 logback.xml  
-rw-rw-r-- 1 nifi users 8701 Dec 7 00:52 nifi.properties  
-rw-rw-r-- 1 nifi users 3637 Aug 26 13:51 state-management.xml  
-rw-rw-r-- 1 nifi users 1437 Aug 26 13:51 zookeeper.properties
```

1. Edit the file named named “jms-subscription-<processor_id>”.
2. Change the hostname.
3. Restart NiFi.
5. Import the data ingest template.
 - (a) Locate the data_ingest.zip file. You will need the file locally to upload it. You can find it in one of two places:
 - i. <kylo_project>/samples/templates/nifi-1.0/
 - ii. /opt/kylo/setup/data/templates/nifi-1.0
 - (b) Go to the templates page and import the data ingest template.
 - (c) Manually update the Spark validate processor.

Add this variable to the \${table_field_policy_json_file}. It should look like this:

```
$ {table_field_policy_json_file}, /usr/hdp/current/spark-client/conf/hive-site.xml
```

4. Edit the “Upload to HDFS” and remove “Remote Owner” and “Remote Group” (since we aren’t using supe-ruser).
6. Update NiFi processors for Kylo template versions prior to 0.5.0.

We need to update a few settings in the elasticsearch and standard ingest template. This is not required with 0.5.0 or greater since they will be set during import.

 - (a) Login to NiFi.
 - (b) Go to the reusable_templates → standard-ingest process group.
 - i. Edit the “Register Index” processor and set the URL to the <KYLO_EDGE_HOSTNAME>.
 - ii. Edit the “Update Index” processor and set teh URL to the <KYLO_EDGE_HOSTNAME>.
8. Import the transform feed (Optional).

Create Data Ingest Feed Test

1. Create a userdata feed to test.
2. Test the feed.

```
cp -p <PATH_TO_FILE>/userdata1.csv /var/dropzone/
```

Configuration Guides

Hortonworks Sandbox Configuration

Introduction

This guide will help you install the Hortonworks sandbox for development and RPM testing.

Install and Configure the Hortonworks Sandbox

Download the latest HDP sandbox and import it into Virtual Box. We want to change the CPU and and RAM settings:

- CPU - 4
- RAM - 10GB

Add Virtual Box Shared Folder

Adding a shared folder to Virtual Box will allow you to access the Kylo project folder outside of the VM so you can copy project artifacts to the sandbox for testing.

Note: This should be done before starting the VM to that you can auto mount the folder.

```
VBx GUI > Settings > Shared Folders > Add
```

```
Folder Path = <pathToProjectFolder>
Folder Name = kylo
```

Choose Auto-mount so that it remembers next time you start the VM.

Open VM Ports

The following ports needs to be forwarded to the VM:

```
(On Virtual Box > Settings > Network > Port Forwarding
```

This table shows the ports to add.

Application Name	Host Port	Guest Port	Comment
Kylo UI	8401	8400	Use 8401 on the HostIP side so that you can run it in your IDE under 8400 and still test in the VM
Kylo Spark Shell	8450	8450	
NiFi	8079	8079	
ActiveMQ Admin	8161	8161	
ActiveMQ JMS	61616	61616	
MySQL	3306	3306	

Note: HDP 2.5+ sandbox for VirtualBox now uses Docker container, which means configuring port-forwarding in the VirtualBox UI is not enough anymore. You should do some extra steps described in:

Startup the Sandbox

1. Start the sandbox.
2. SSH into the sandbox.

```
$ ssh root@localhost -p 2222 (password is "hadoop")
```

Note: You will be prompted to change your password.

3. Add the Ambari admin password.

```
$ ambari-admin-password-reset
```

After setting the password the Ambari server will be started.

Configuration Guide

TBD or consolidated with other docs

Walk through the properties

memory settings

database settings, schema location, tables etc

Kerberos Installation Example - Cloudera

Important: This document should only be used for DEV/Sandbox purposes. It is useful to help quickly Kerberize your Cloudera sandbox so that you can test Kerberos features.

Prerequisite

Java

All client node should have java installed on it.

```
$ java version "1.7.0_80"
$ Java(TM) SE Runtime Environment (build 1.7.0_80-b15)
$ Java HotSpot(TM) 64-Bit Server VM (build 24.80-b11, mixed mode)

$ echo $JAVA_HOME
$ /usr/java/jdk1.7.0_80
```

Install Java Cryptography Extensions (JCE)

```

sudo wget -nv --no-check-certificate --no-cookies --header "Cookie:
oraclelicense=accept-securebackup-cookie"
http://download.oracle.com/otn-pub/java/jce/7/UnlimitedJCEPolicyJDK7.zip
-O
/usr/java/jdk1.7.0_80/jre/lib/security/UnlimitedJCEPolicyJDK7.zip

cd /usr/java/jdk1.7.0_80/jre/lib/security

sudo unzip UnlimitedJCEPolicyJDK7.zip

sudo cp UnlimitedJCEPolicy/* .

#sudo rm -r UnlimitedJCEPolicy*

ls -l

```

Test Java Cryptography Extension

Create a java Test.java and paste below mentioned code in it.

```

$ vi Test.java

import javax.crypto.Cipher;
class Test {
public static void main(String[] args) {
try {
    System.out.println("Hello World!");
    int maxKeyLen = Cipher.getMaxAllowedKeyLength("AES");
    System.out.println(maxKeyLen);
} catch (Exception e){
    System.out.println("Sad world :(");
}
}
}

```

Compile:

```
$ javac Test.java
```

Run test, the expected number is: 2147483647

```

$ java Test
Hello World!
2147483647

```

Install Kerberos

On a cluster, go to the master node for installation of Kerberos utilities.

1. Install a new version of the KDC server:

```
yum install krb5-server krb5-libs krb5-workstation
```

2. Using a text editor, open the KDC server configuration file, located by default here:

```
vi /etc/krb5.conf
```

3. Change the [realms] as below to “quickstart.cloudera”. Update KDC and Admin Server Information.

[logging]

```
default = FILE:/var/log/krb5libs.log
kdc = FILE:/var/log/krb5kdc.log
admin_server = FILE:/var/log/kadmind.log
```

```
default_realm = quickstart.cloudera
dns_lookup_realm = false
dns_lookup_kdc = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true
```

[realms]

```
quickstart.cloudera = {
  kdc = quickstart.cloudera
  admin_server = quickstart.cloudera
}
```

4. Update /var/kerberos/krb5kdc/kdc.conf. Change the [realms] as “quickstart.cloudera”.

[kdcdefaults]

```
kdc_ports = 88
kdc_tcp_ports = 88
```

[realms]

```
quickstart.cloudera = {
  #master_key_type = aes256-cts
  acl_file = /var/kerberos/krb5kdc/kadm5.acl
  dict_file = /usr/share/dict/words
  admin_keytab = /var/kerberos/krb5kdc/kadm5.keytab
  supported_enctypes = aes256-cts:normal aes128-cts:normal
  des3-hmac-sha1:normal arcfour-hmac:normal des-hmac-sha1:normal
  des-cbc-md5:normal des-cbc-crc:normal
}
```

5. Update /var/kerberos/krb5kdc/kadm5.acl and replace EXAMPLE.COM with “quickstart.cloudera”.

```
*/admin@quickstart.cloudera*
```

6. Create the Kerberos Database. Use the utility kdb5_util to create the Kerberos database. While asking for password, enter password as thinkbig.

```
kdb5_util create -s
```

7. Start the KDC. Start the KDC server and the KDC admin server.

```
/etc/rc.d/init.d/krb5kdc start
/etc/rc.d/init.d/kadmin start
```

Note: When installing and managing your own MIT KDC, it is very important to set up the KDC server to auto start on boot.

```
chkconfig krb5kdc on
chkconfig kadmin on
```

8. Create a KDC admin by creating an admin principal. While asking for password , enter password as thinkbig.

```
kadmin.local -q "addprinc admin/admin"
```

9. Confirm that this admin principal has permissions in the KDC ACL. Using a text editor, open the KDC ACL file:

```
vi /var/kerberos/krb5kdc/kadm5.acl
```

10. Ensure that the KDC ACL file includes an entry so to allow the admin principal to administer the KDC for your specific realm. The file should have an entry:

```
*/quickstart.cloudera*
```

11. After editing and saving the kadm5.acl file, you must restart the kadmin process.

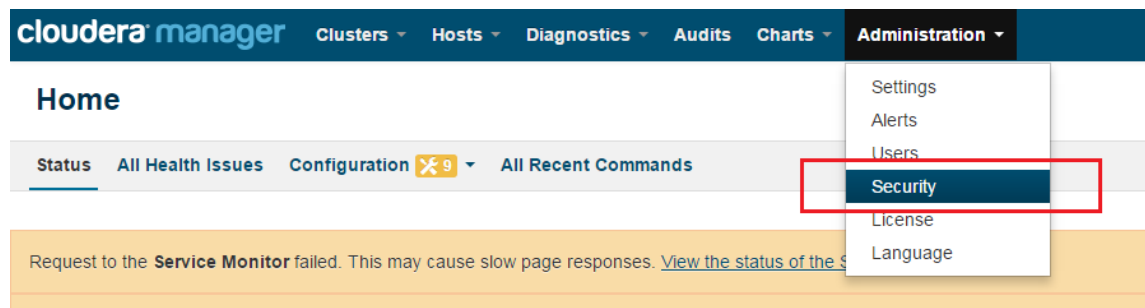
```
/etc/rc.d/init.d/kadmin restart
```

12. Create a user in the linux by typing below. We will use this user to test whether the Kerberos authentication is working or not. We will first run the command `hadoop fs ls /` but switching to this user. And we will run the same command again when we enable Kerberos.

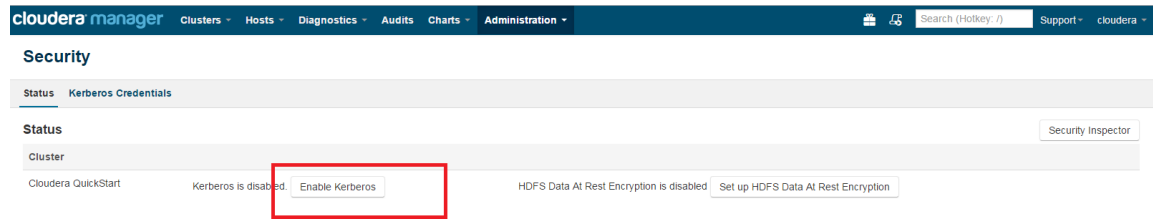
```
adduser testUser
su testUser
hadoop fs ls /
```

Install Kerberos on Cloudera Cluster

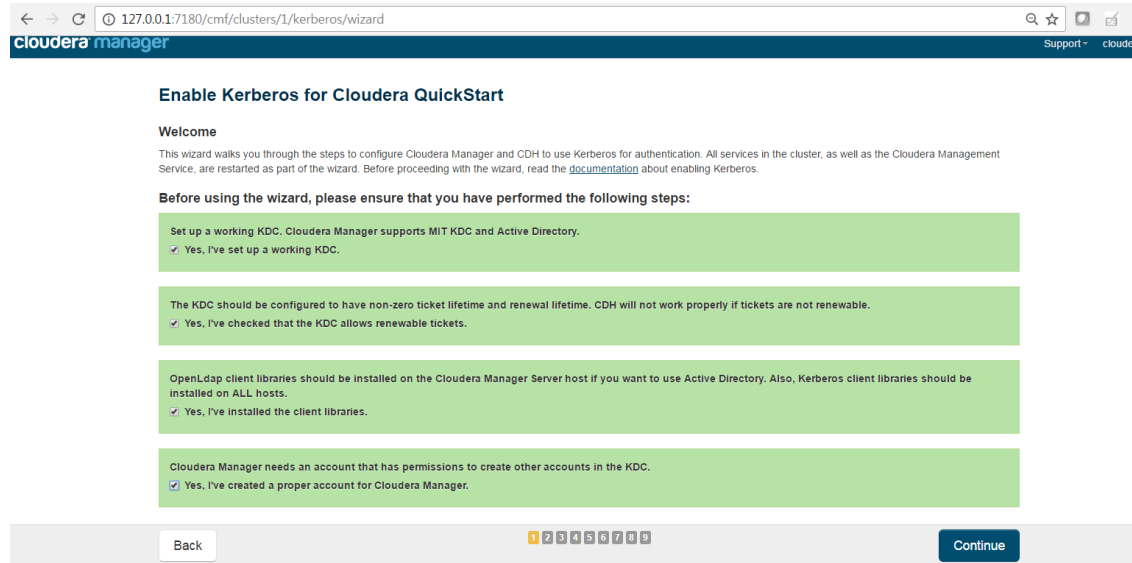
1. Login to Cloudera Manager and Select Security option from Administration tab.



2. Click on Enable Kerberos.



3. Select each item and click on continue.



4. The Kerberos Wizard needs to know the details of what the script configured. Fill in the entries as follows and click continue.

```
KDC Server Host: quickstart.cloudera
Kerberos Security Realm: quickstart.cloudera
Kerberos Encryption Types: aes256-cts-hmac-sha1-96
```

← → ↻ 127.0.0.1:7180/cm/clusters/1/kerberos/wizard#step=kerberosKRB5ConfStep cloudera manager Support

Enable Kerberos for Cloudera QuickStart

KDC Information

Specify information about the KDC. The properties below are used by Cloudera Manager to generate principals for CDH daemons running on the cluster.

KDC Type ☒ MIT KDC ☐ Active Directory ?

KDC Server Host C ?

Kerberos Security Realm C ?

Kerberos Encryption Types + - C ?
 Encryption types supported by KDC. **Note:** To use AES encryption, make sure you have deployed JCE Unlimited Strength Policy File by following the instructions [here](#). X

Maximum Renewable Life for Principals day(s) ?

Back 1 2 3 4 5 6 7 8 9 Continue

5. Select checkbox Manage krb5.conf through cloudera manager.

← → ↻ 127.0.0.1:7180/cm/clusters/1/kerberos/wizard#step=kerberosKRB5Conf2Step cloudera manager Support

Enable Kerberos for Cloudera QuickStart

KRB5 Configuration

Specify the properties needed for generating krb5.conf for the cluster. You can use the safety valve fields to specify configuration of an advanced KDC setup, for example, with cross-realm authentication.

Manage krb5.conf through Cloudera Manager ☒ C ?

Kerberos Ticket Lifetime day(s) ?

Kerberos Renewable Lifetime day(s) ?

DNS Lookup KDC ☐ ?

Forwardable Tickets ☒ ?

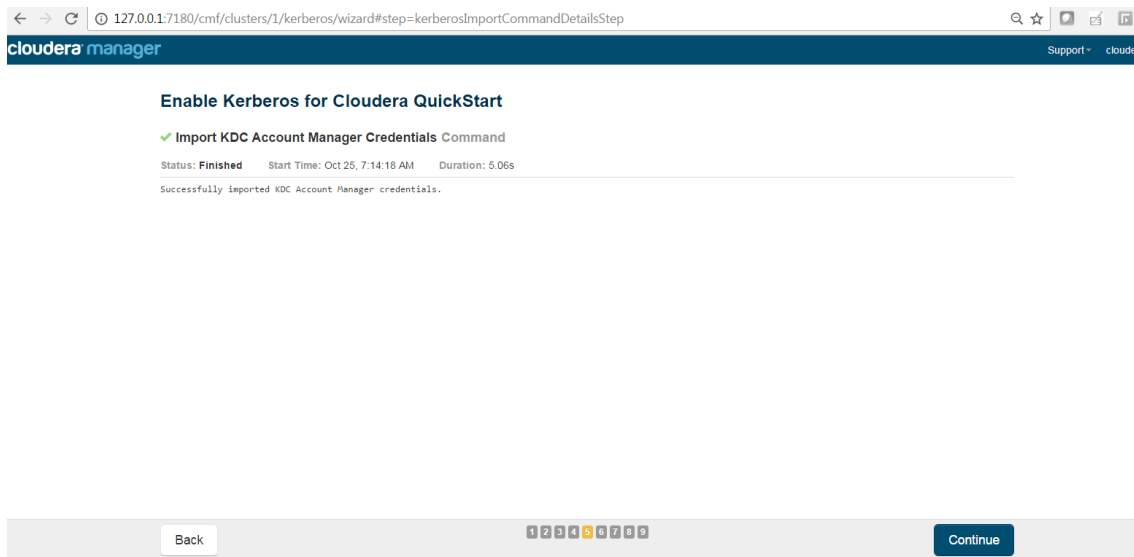
KDC Timeout second(s) ?

Back 1 2 3 4 5 6 7 8 9 Continue

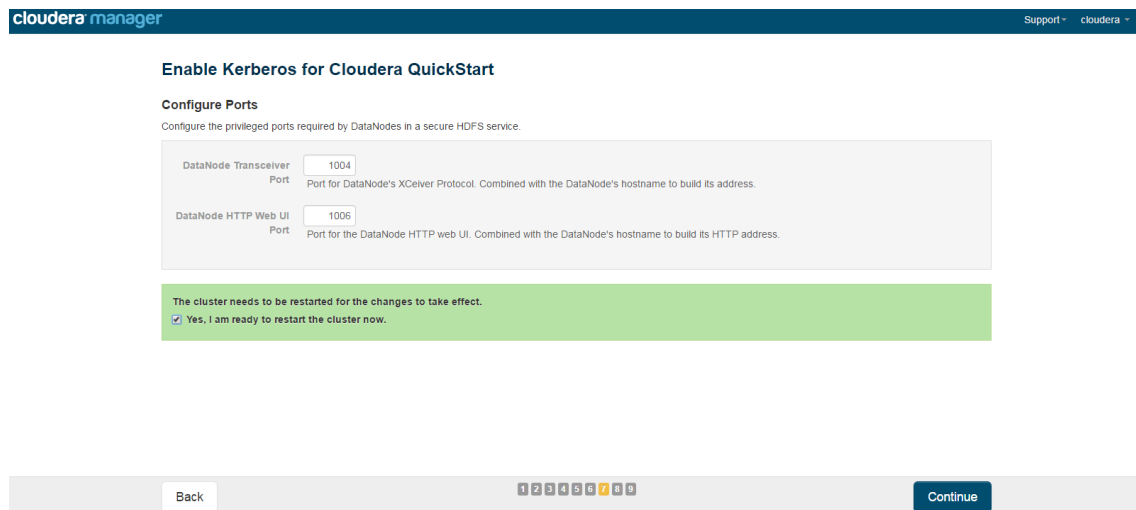
6. Enter username and password for of KDC admin user.

```
Username : admin/admin@quickstart.cloudera
Password : thinkbig
```

The next screen provides good news. It lets you know that the wizard was able to successfully authenticate.



7. Select “I’m ready to restart the cluster now” and click on continue.



8. Make sure all services started properly. Kerberos is successfully installed on cluster.

KeyTab Generation

1. Create a keytab file for Nifi user.

```
kadmin.local
addprinc -randkey nifi@quickstart.cloudera
xst -norandkey -k /etc/security/nifi.headless.keytab nifi@quickstart.cloudera
exit

chown nifi:hadoop /etc/security/keytabs/nifi.headless.keytab
chmod 440 /etc/security/keytabs/nifi.headless.keytab

[Optional] You can initialize your keytab file using below command.

kinit -kt /etc/security/keytabs/nifi.headless.keytab nifi
```


Yarn Cluster Mode Configuration

Overview

In order for the yarn cluster mode to work to validate the Spark processor, the JSON policy file has to be passed to the cluster. In addition the hive-site.xml file needs to be passed. This should work for both HDP and Cloudera clusters.

Requirements

You must have Kylo installed.

Step 1: Add the Data Nucleus Jars

Note: This step is required only for HDP and is not required on Cloudera.

If using Hive in your Spark processors, provide Hive jar dependencies and hive-site.xml so that Spark can connect to the right Hive metastore. To do this, add the following jars into the “Extra Jars” parameter:

```
/usr/hdp/current/spark-client/lib (/usr/hdp/current/spark-client/lib/datanucleus-api-
→jdo-x.x.x.jar, /usr/hdp/current/spark-client/lib/datanucleus-core-x.x.x.jar, /usr/hdp/
→current/spark-client/lib/datanucleus-rdbms-x.x.x.jar)
```

Step 2: Add the hive-site.xml File

Specify “hive-site.xml”. It should be located in the following location:

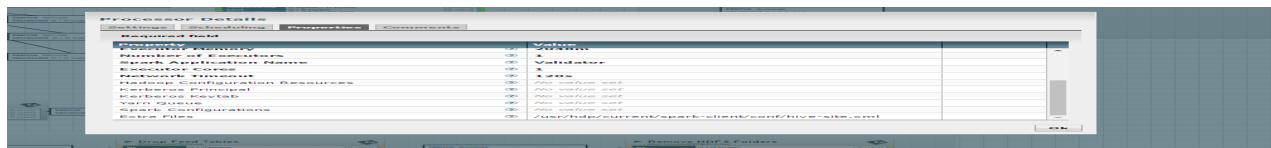
Hortonworks

```
/usr/hdp/current/spark-client/conf/hive-site.xml
```

Cloudera

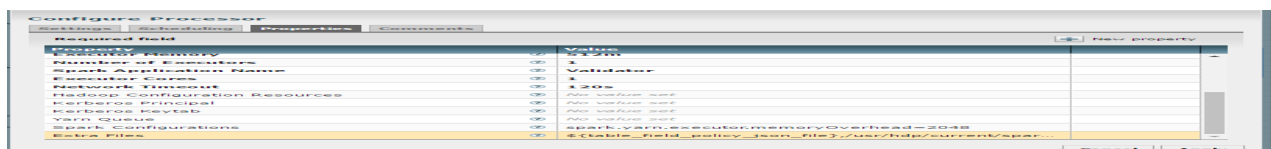
```
/etc/hive/conf.cloudera.hive/hive-site.xml
```

Add this file location to the “Extra Files” parameter. To add multiple files, separate them with a comma.



Step 3: Validate and Split Records Processor

If using the “Validate and Split Records” processor in the standard-ingest template, pass the JSON policy file as well.



Configuration for a Kerberos Cluster

The Kylo applications contain features that leverage the thrift server connection to communicate with the cluster. In order for them to work in a Kerberos cluster, some configuration is required. Some examples are:

- Profiling statistics
- Tables page
- Wrangler

Prerequisites

Below are the list of prerequisites for enabling Kerberos for the Kylo data lake platform.

1. Running Hadoop cluster
2. Kerberos should be enabled
3. Running Kylo 0.4.0 or higher

Configuration Steps

1. Create a Headless Keytab File for the Hive and Kylo User.

Note: Perform the following as root. Replace “sandbox.hortonworks.com” with your domain.

```
[root]# kadmin.local

kadmin.local: addprinc -randkey "kylo@sandbox.hortonworks.com"

kadmin.local: xst -norandkey -k
/etc/security/keytabs/kylo.headless.keytab
kylo@sandbox.hortonworks.com

kadmin.local: xst -norandkey -k
/etc/security/keytabs/hive-kylo.headless.keytab
hive/sandbox.hortonworks.com@sandbox.hortonworks.com

kadmin.local: exit

[root]# chown kylo:hadoop
/etc/security/keytabs/kylo.headless.keytab

[root]# chmod 440 /etc/security/keytabs/kylo.headless.keytab

[root]# chown kylo:hadoop
/etc/security/keytabs/hive-kylo.headless.keytab

[root]# chmod 440
/etc/security/keytabs/hive-kylo.headless.keytab
```

1. Validate that the Keytabs Work.

```
[root]# su - kylo
```

```
[root]# kinit -kt /etc/security/keytabs/kylo.headless.keytab kylo
```

```
[root]# klist
[root]# su - hive
[root]# kinit -kt /etc/security/keytabs/hive-kylo.headless.keytab hive/sandbox.hortonworks.com
[root]# klist
```

2. Modify the kylo-spark-shell configuration.

```
[root]# vi /opt/kylo/kylo-services/conf/spark.properties
kerberos.kylo.kerberosEnabled=true

[root]# vi /opt/kylo/kylo-services/bin/run-kylo-spark-shell.sh
#!/bin/bash

spark-submit -principal 'kylo@sandbox.hortonworks.com' -keytab
/etc/security/keytabs/kylo.headless.keytab ...
```

3. Modify the kylo-services configuration.

Tip: Replace “sandbox.hortonworks.com” with your domain.

To add Kerberos support to kylo-services, you must enable the feature and update the Hive connection URL to support Kerberos.

```
[root]# vi
/opt/kylo/kylo-services/conf/application.properties
```

```
# This property is for the hive thrift connection used by
kylo-services

hive.datasource.url=jdbc:hive2://localhost:10000/default;principal=hive/sandbox.
↳hortonworks.com@sandbox.hortonworks.com

# This property will default the URL when importing a template using
the thrift connection

nifi.service.hive_thrift_service.database_connection_url=jdbc:hive2://localhost:10000/
↳default;principal=hive/sandbox.hortonworks.com@sandbox.hortonworks.com

# Set Kerberos to true for the kylo-services application and set
the 3 required properties

kerberos.hive.kerberosEnabled=true

kerberos.hive.hadoopConfigurationResources=/etc/hadoop/conf/core-site.xml,/etc/hadoop/
↳conf/hdfs-site.xml

kerberos.hive.kerberosPrincipal=hive/sandbox.hortonworks.com

kerberos.hive.keytabLocation=/etc/security/keytabs/hive-kylo.headless.keytab

# uncomment these 3 properties to default all NiFi processors that
have these fields. Saves time when importing a template

nifi.all_processors.kerberos_principal=nifi
```

```
nifi.all_processors.kerberos_keytab=/etc/security/keytabs/nifi.headless.keytab  
nifi.all_processors.hadoop_configuration_resources=/etc/hadoop/conf/core-site.xml,/  
→etc/hadoop/conf/hdfs-site.xml
```

4. Restart the kylo-services and kylo-spark-shell.

```
[root]# service kylo-services restart
```

```
[root]# service kylo-spark-shell restart
```

Kylo is now configured for a Kerberos cluster. You can test that it is configured correctly by looking at profile statistics (if applicable): go to the Tables page and drill down into a Hive table, and go to the Wrangler feature and test that it works.

NiFi Configuration for a Kerberos Cluster

Prerequisites

Below are the list of prerequisites to enable Kerberos for the NiFi data lake platform:

- A Hadoop cluster must be running.
- NiFi should be running with latest changes.
- Kerberos should be enabled.
- Keytabs should be created and accessible.

Types of Processors to be Configured

HDFS

- IngestHDFS
- CreateHDFSFolder
- PutHDFS

Hive

- TableRegister
- ExecuteHQLStatement
- TableMerge

Spark

- ExecuteSparkJob

Configuration Steps

1. Create a Kerberos keytab file for Nifi user.

```
kadmin.local
addprinc -randkey nifi@sandbox.hortonworks.com
xst -norandkey -k /etc/security/keytabs/nifi.headless.keytab nifi@sandbox.hortonworks.com
exit
chown nifi:hadoop /etc/security/keytabs/nifi.headless.keytab
chmod 440 /etc/security/keytabs/nifi.headless.keytab
Test that the keytab works. You can initialize your keytab file using below command.
su - nifi
kinit -kt /etc/security/keytabs/nifi.headless.keytab nifi
klist
```

2. Make sure nifi.properties file is available in conf directory of NiFi installed location.

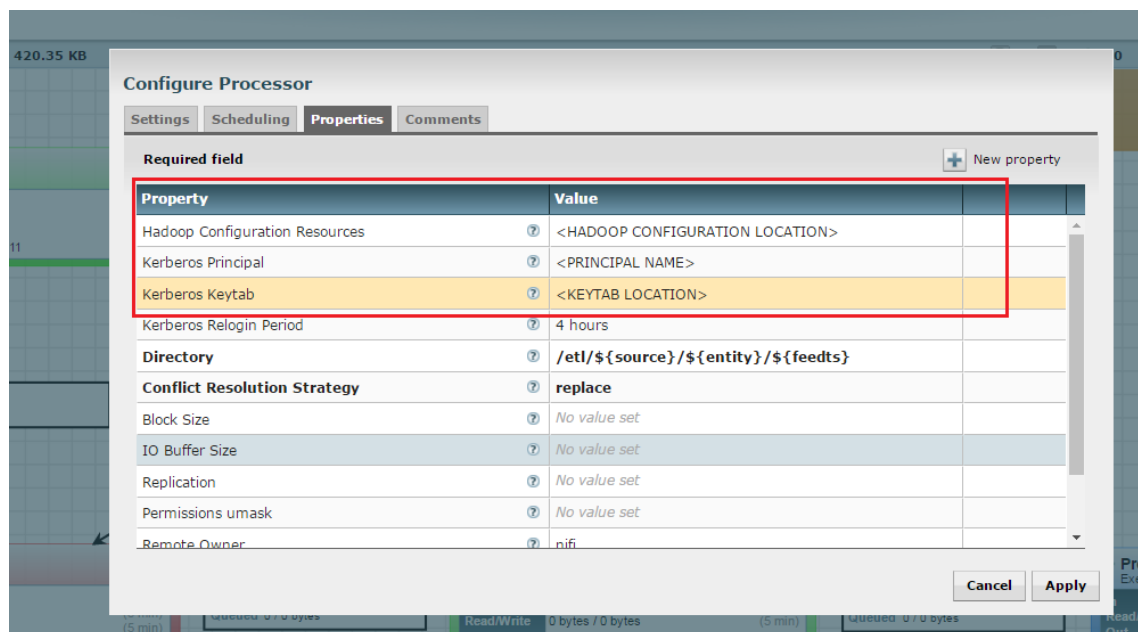
```
[ec2-user@ip-172-31-28-19 conf]$ pwd
/opt/nifi/nifi-0.5.1/conf
[ec2-user@ip-172-31-28-19 conf]$ vi nifi.properties
```

3. Open nifi.properties file and set location of krb5.conf file to property nifi.kerberos.krb5.file.

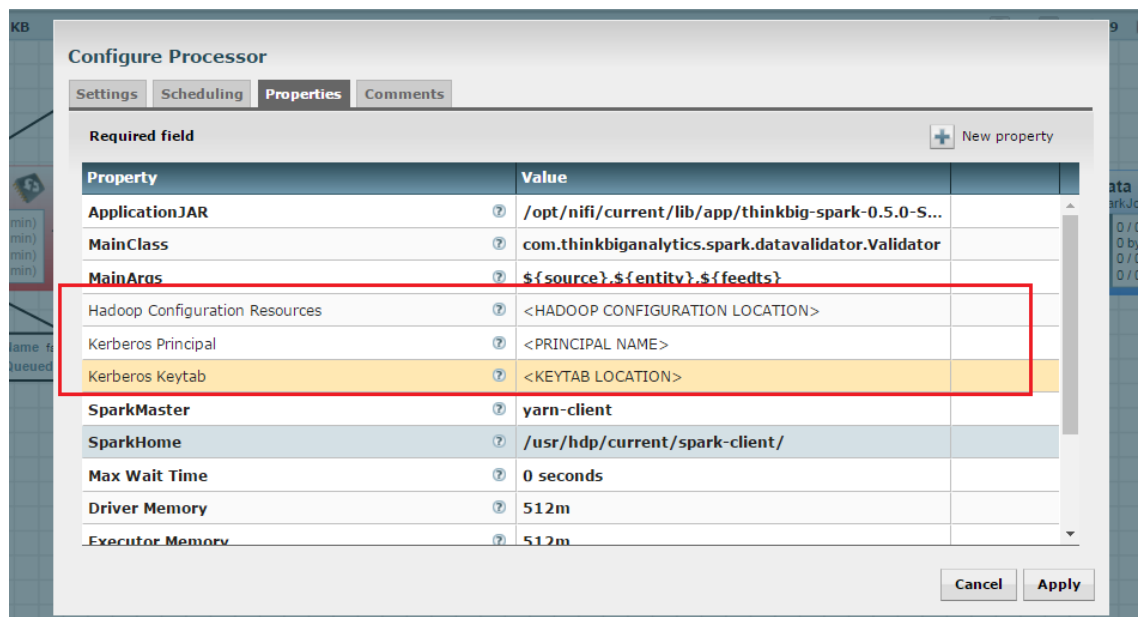
```
vi nifi.properties
nifi.kerberos.krb5.file=/etc/krb5.conf
```

4. HDFS Processor Configuration : Log in to NiFi UI and select HDFS processor and set properties which is highlighted in red box.

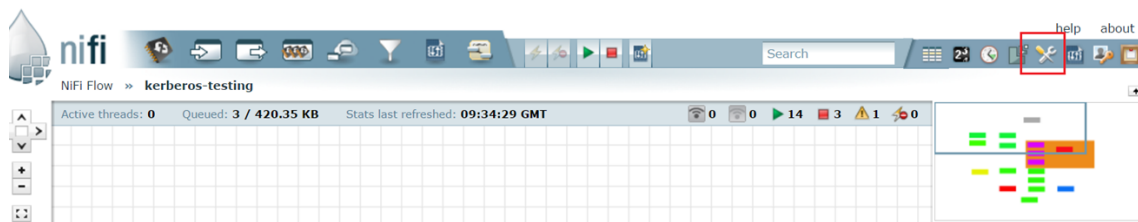
```
Hadoop Configuration Resource : /etc/hadoop/conf/core-site.xml,/etc/hadoop/conf/hdfs-site.xml
Kerberos Principal: nifi
Kerberos Keytab : /etc/security/keytabs/nifi.headless.keytab
```



5. SPARK Processor Configuration : Log in to NiFi UI and select HDFS processor and set properties which is highlighted in red box.



6. Hive Processor Configuration : Log in to NiFi UI and go to toolbar.



7. Go to Controller Service Tab and disable Thrift Controller Services if already running which highlighted in red box.

NiFi Flow Settings

General Controller Services Reporting Tasks Last updated: 09:35:36 GMT			
Name	Type	State	
MetadataConnectionService	MetadataConnectionService	Enabled	
ThriftConnectionPool	ThriftConnectionPool	Enabled	

8. Make sure everything has stopped properly like below.

ing Task

Service

ThriftConnectionPool

Steps to disable ThriftConnectionPool

Stopping referencing processors and reporting tasks

Disabling referencing controller services

Disabling this controller service

Referencing Components

Processors (4)

Register Tables TableRegister

Create Feed Partition ExecuteHQLStatement

TableMerge TableMerge

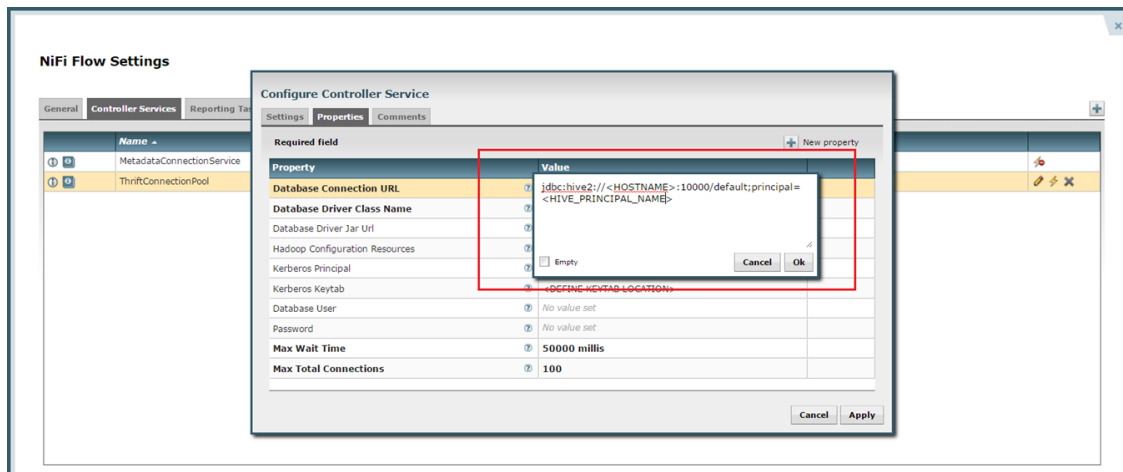
TableRegister TableRegister

Close

9. Update HiveServer2 hostname and Hive principal name.

```
Database Connection URL:
'jdbc:hive2://:<HOSTNAME>:10000/default;principal=hive/<HOSTNAME>@HOSTNAME'

ex.
'jdbc:hive2://localhost:10000/default;principal=hive/sandbox.hortonworks.com@sandbox.
↪hortonworks.com'
```



10. Update Kerberos user information and Hadoop Configuration. Apply Changes and start controller services.
You have successfully configured NiFi DataLake Platform with Kerberos.

Postgres Metastore Configuration

Introduction

Kylo currently requires MySQL for the kylo schema. However, you can configure Kylo to work with a cluster that uses Postgres. We need to make some modifications to support Hive.

Kylo Services Configuration

For Kylo to connect to a Postgres databases for the Hive metadata you need to change the following section of the kylo-services application.properties file.

```
hive.metastore.datasource.driverClassName=org.postgresql.Driver
hive.metastore.datasource.url=jdbc:postgresql://<hostname>:5432/hive
hive.metastore.datasource.username=hive
hive.metastore.datasource.password=
hive.metastore.datasource.validationQuery=SELECT 1
hive.metastore.datasource.testOnBorrow=true
```

Elasticsearch NiFi Template Changes

The index_schema_service template is used to query out feed metadata from the Hive tables, which is then stored in elasticsearch so it can be searched for in Kylo. The following steps need to be taken to the template to support Postgres:

Step 1: Copy the Postgres JAR file to NiFi

```
mkdir /opt/nifi/postgres
cp /opt/kylo/kylo-services/lib/postgresql-9.1-901-1.jdbc4.jar
/opt/nifi/postgres
chown -R nifi:users /opt/nifi/postgres
```


Step 2: Create a Controller Service for Postgres Connection

You will need to create an additional database controller services to connect to the second database.

Controller Service Properties:

```
Controller Service Type: DBCPConnectionPool
Database Connection URL: jdbc:postgresql://<host>:5432/hive
Database Driver Class Name: org.postgresql.Driver
Database Driver Jar URL:
file:///opt/nifi/postgres/postgresql-9.1-901-1.jdbc4.jar Database
User: hive
Password: <password>
```

Enable the Controller Service.

Step 3: Update “Query Hive Table Metadata” Processor

Edit the “Query Hive Table Schema” processor and make two changes:

1. Disable the “Query Hive Table Metadata” processor.
2. Change the Database Connection Pooling Service to the Postgres Hive controller service created above.
3. Update the “SQL select Query” to be a Postgres query.

```
SELECT d."NAME", d."OWNER_NAME", t."CREATE_TIME", t."TBL_NAME",
t."TBL_TYPE",
c."COLUMN_NAME", c."TYPE_NAME"
FROM "COLUMNS_V2" c
JOIN "SDS" s on s."CD_ID" = c."CD_ID"
JOIN "TBLS" t ON s."SD_ID" =t."SD_ID"
JOIN "DBS" d on d."DB_ID" = t."DB_ID"
where d."NAME" = '${category}' and t."TBL_NAME" like '${feed}';
```

4. Enable the “Query Hive Table Metadata” processor.
5. Test a feed to make sure the data is getting indexed.

Overview

Kylo supports a pluggable authentication architecture that allows customers to integrate their existing infrastructure when authenticating a user. The pluggability is built around , which delegates authentication to one or more configured that all collaborate in an authentication attempt.

Kylo supplies LoginModule implementations for the most common authentication scenarios, though customers will be able to provide their own modules to replace or augment the modules provided by Kylo.

In addition to performing authentication, LoginModules may, upon successful login, associate the logged-in user with a set of principals (user ID and groups/roles) that can be used to make authorization checks. For instance, a LoginModule that authenticates a user's credentials using LDAP may also load any groups defined in the LDAP store for that user, and these groups can have permissions granted to them in Kylo.

Built-In Pluggable Authentication Profiles

Kylo comes with some pre-built authentication configurations that may be activated by adding the appropriate Spring profiles to the UI and server configuration *application.properties* files. By default, whenever any of these profiles are added to the configuration it is equivalent to adding their associated LoginModules to the overall JAAS configuration using the “required” control flag.

Note: More than one profile may be activated at one time. If multiple profiles are used, authentication in Kylo will only occur if all of the login requirements of each of the profiles are satisfied.

The table below lists all of the profiles currently supported by Kylo out-of-the-box. When any of these profiles are activated certain properties are expected to be present in the *application.properties* files.

Login Method	Spring Profile	Description
Kylo User	<i>auth-kylo</i>	Authenticates users against the Kylo user/group store
LDAP	<i>auth-ldap</i>	Authenticates users stored in LDAP
Active Directory	<i>auth-ad</i>	Authenticates users stored in Active Directory
Users file	<i>auth-file</i>	Authenticates users in a file <i>users.properties</i> (typically used in development only)
Simple	<i>auth-simple</i>	Allows only one admin user defined in the configuration properties (development only)

auth-kylo

When this profile is active, a LoginModule will be added to the configuration that validates whether the authenticating user is present in the Kylo user store.

Note: This profile is typically used in conjunction with other profiles (such as *auth-ldap*) as this configuration does not perform any password validation.

Properties	Re-quired	Ex-ample	Description
<code>security.auth.kylo.login.services</code>	No	<code>required</code>	Corresponds to the control flag for LoginModule configurations: <i>required</i> , <i>requisite</i> , <i>sufficient</i> , and <i>optional</i> . Possible values are <i>required</i> , <i>requisite</i> , <i>sufficient</i> , and <i>optional</i>

auth-file

When this profile is active, a LoginModule will be added to the configuration that authenticates a username/password using user information within specific files on the file system. For validating the credentials it looks by default, unless configured otherwise, for a file called *users.properties* on the classpath containing a mapping of usernames to passwords in the form:

```
user1=pw1
user2=pw2
```

If authentication is successful it will then look for a *groups.properties* file on the classpath to load the groups that have been assigned to the authenticated user. The format of this file is:

```
user1=groupA,groupB
user2=groupA,groupC
```

Note that use of the *groups.properties* file is optional when used in conjunction with other authentication profiles. For instance, it would be redundant (but not invalid) to have a groups file when *auth-file* is used with *auth-kylo*, as the latter profile will load any user assigned groups from the Kylo store as well as those defined in the group file. It would likely be confusing to have to manage groups from two different sources.

Note: The *auth-file* profile should generally not be used in a production environment because it currently stores user passwords in the clear. It is primarily used only in development and testing.

Properties	Re-quired	Example	Description
security.auth.file.users	No	users.properties	The value is either a name of a resource found on the classpath or, if prepended by <i>file:///</i> , a direct file path
security.auth.file.groups	No	groups.properties	The same as security.auth.file.users but for the groups file

If *auth-file* is active and no users file property is specified in the configuration then these implicit username/password properties will be assumed:

```
dladmin=thinkbig
analyst=analyst
designer=designer
operator=operator
```

auth-ldap

This profile configures a LoginModule that authenticates the username and password against an LDAP server.

Property	Re-quired	Example	Description
security.auth.ldap.server.uri	Yes	ldap://localhost:52389/ dc=example, dc=com	The URI to the LDAP server and root context
security.auth.ldap.authenticator.userDnPatterns	Yes	uid={0}, ou=people	The DN filter patterns, minus the root context portion, that identifies the entry for the user. The username is substituted for the {0} tag. If more than one pattern is supplied they should be separated by vertical bars
security.auth.ldap.user.enableGroups	No	true	Activates user group loading; default: false
security.auth.ldap.user.groupsBase	No	ou=groups	The filter pattern that identifies group entries
security.auth.ldap.user.groupNameAttr	No	ou	The attribute of the group entry containing the group name
security.auth.ldap.server.authDn	No	uid=admin, ou=people, dc=example, dc=com	The LDAP account with the privileges necessary to access user or group entries; usually only needed (if at all) when group loading is activated
security.auth.ldap.server.password	No		The password for the account with the privileges necessary to access user or group entries

auth-ad

This profile configures a LoginModule that authenticates the username and password against an Active Directory server.

Property	Re-quired	Example Value	Description
security.auth.ad.server.uri	Yes	ldap://example.com/	The URI to the AD server
security.auth.ad.server.domain	Yes	test.example.com	The AD domain of the users to authenticate
security.auth.ad.user.enableGroups	No	true	Activates user group loading; default: false

auth-simple

This profile configures a LoginModule that authenticates a single user as an administrator using username and password properties specified in *application.properties*. The specified user will be the only one able to login to Kylo. Obviously, this profile should only be used in development.

Property	Required	Example Value	Description
authenticationService.username	Yes	dladmin	The username of the administrator
authenticationService.password	Yes	thinkbig	The password of the administrator

User Group Handling

Kylo access control is governed by permissions assigned to user groups, so upon successful authentication any groups to which the user belongs must be loaded and associated with the current authenticated request being processed. JAAS LoginModules have two responsibilities:

1. Authenticate a login attempt
2. Optionally, associate principals (user and group identifiers) with the security context of the request

A number of authentication profiles described above support loading of user groups at login time. For *auth-kylo* this is done automatically, for others (*auth-ldap*, ‘auth-file’, etc.) this must be configured. If more than one group-loading profile is configured, the result is additive. For example, if your configuration activates the profiles *auth-kylo* and *auth-LDAP*, and the LDAP properties enable groups, then any groups associated with the user in both LDAP and the Kylo user store will be combined and associated with the user’s security context.

JAAS Application Configuration

Currently, there are two applications (from a JAAS perspective) for which LoginModules may be configured for authentication: the Kylo UI and Services REST API. Kylo provides an API that allows plugins to easily integrate custom login modules into the authentication process.

Creating a Custom Authentication Plugin

The first step is to create Kylo plugin containing a that performs whatever authentication is required and then adds any username/group principals upon successful authentication. This module will be added to whatever other LoginModules may be associated with the target application (Kylo UI and/or Services.)

The service-auth framework provides an API to make it easy to integrate a new LoginModule into the authentication of the Kylo UI or services REST API. The easiest way to integrate your custom LoginModule is to create a Spring configuration class, which will be bundled into your plugin jar along with your custom LoginModule. That then uses the framework-provided LoginConfigurationBuilder to incorporate your LoginModule into the authentication

sequence. The following is an example of a configuration class that adds a new module to the authentication sequence of both the Kylo UI and Services; each with different configuration options:

```
@Configuration
public class MyCustomAuthConfig {
    @Bean
    public LoginConfiguration myLoginConfiguration(LoginConfigurationBuilder builder)
    ↪{
        return builder
            .loginModule(JaasAuthConfig.JAAS_UI)
            .moduleClass(MyCustomLoginModule.class)
            .controlFlag("required")
            .option("customOption", "customValue1")
            .add()
            .loginModule(JaasAuthConfig.JAAS_SERVICES)
            .moduleClass(MyCustomLoginModule.class)
            .controlFlag("required")
            .option("customOption", "customValue2")
            .option("anotherOption", "anotherValue")
            .add()
            .build();
    }
}
```

As with any Kylo plugin, to deploy this configuration you would create a jar file containing the above configuration class, your custom login module class, and a `plugin/plugin-context.xml` file to bootstrap your plugin configuration. Dropping this jar into the plugin directories of the UI and Services would allow your custom `LoginModule` to participate in their login process.

Overview

A goal is to support authentication and authorization seamlessly between the Kylo applications and the Hadoop cluster.

Authorization

Authorization within Kylo will use access control lists (ACL) to control what actions users may perform and what data they may see. The actions that a user or group may perform, whether to invoke a function or access data, are organized into a hierarchy, and privileges may be granted at any level in that hierarchy.

Authorization in Kylo is divided into two layers: service-level (Kylo-wide) permissions and entity-level permissions. A permission in Kylo is the granting to a user or group the right to perform some action, such as see the description of a template, create and edit a category, enable/disable a feed, etc. Access to these functions can often be controlled at both the service-level and entity-level.

Users and Groups can be updated using the Users and Groups pages under the Admin section in Kylo.

Note: If groups are enabled for an authentication plugin module, then user groups will be provided by the external provider and may not be updatable from the Users page.

Default Users and Groups

When Kylo is newly installed, it will be pre-configured with a few default users and groups defined; with varying permissions assigned to each group. The default groups are:

- Administrators
- Operations
- Designers

- Analysts
- Users

The default users and their assigned groups are:

- Data Lake Administrator - Administrators, Users
- Analyst - Analysts, Users
- Designer - Designers, Users
- Operator - Operations, Users

The initial installation will also have the *auth-kylo* and *auth-file* included in the active profiles configured in the `conf/application.properties` file of both the UI and Services. With these profiles active the authentication process will use both the built-in Kylo user store and a username/password file to authenticate requests. In this configuration, all activated login modules will have to successfully authenticate a request before access will be granted.

Service-Level Authorization

Service-level access controls what functions are permitted kylo-wide. Access is controlled by granting permissions to groups to perform a set of actions. A logged in user would then be authorized to perform any actions permitted to the groups to which the user is a member.

At the service-level, the hierarchical actions available for granting to groups are organized as follows:

- **Access Kylo Metadata** - Allows the ability to view and query directly the data in the Kylo metadata store, including extensible types
 - **Administer Kylo Metadata** - Allows the ability to directly manage the data in the Kylo metadata store (edit raw metadata, create/update/delete extensible types, update feed status events)
- **Access Feed Support** - Allows access to feeds and feed-related functions
 - **Access Feeds** - Allows access to feeds and their metadata
 - * **Edit Feeds** - Allows creating, updating, enabling and disabling feeds
 - * **Import Feeds** - Allows importing of previously exported feeds (.zip files)
 - * **Export Feeds** - Allows exporting feeds definitions (.zip files)
 - * **Administer Feeds** - Allows deleting feeds and editing feed metadata
 - **Access Tables** - Allows listing and querying Hive tables
 - **Access Visual Query** - Allows access to visual query data wrangler
 - **Access Categories** - Allows access to categories and their metadata
 - * **Edit Categories** - Allows creating, updating and deleting categories
 - * **Administer Categories** - Allows updating category metadata
 - **Access Templates** - Allows access to feed templates
 - * **Edit Templates** - Allows creating, updating, deleting and sequencing feed templates
 - * **Import Templates** - Allows importing of previously exported templates (.xml and .zip files)
 - * **Export Templates** - Allows exporting template definitions (.zip files)
 - * **Administer Templates** - Allows enabling and disabling feed templates

- **Access Data Sources** - Allows (a) access to data sources (b) viewing tables and schemas from a data source (c) using a data source in transformation feed
 - * **Edit Data Sources** - Allows creating and editing data sources
 - * **Administer Data Sources** - Allows getting data source details with sensitive info
- **Access Service Level Agreements** - Allows access to service level agreements
 - * **Edit Service Level Agreements** - Allows creating and editing service level agreements
- **Access Global Search** - Allows access to search all indexed columns
- **Access Users and Groups Support** - Allows access to user and group-related functions
 - **Access Users** - Allows the ability to view existing users
 - * **Administer Users** - Allows the ability to create, edit and delete users
 - **Access Groups** - Allows the ability to view existing groups
 - * **Administer Groups** - Allows the ability to create, edit and delete groups
- **Access Operational Information** - Allows access to operational information like active feeds, execution history, job and feed stats, health status, etc.
 - **Administer Operations** - Allows administration of operations, such as creating/updating alerts, restart/stop/abandon/fail jobs, start/pause scheduler, etc.
- **Access Encryption Services** - Allows the ability to encrypt and decrypt values

The above actions are hierarchical, in that being permitted a lower level action (such as Edit Feeds) implies being granted the higher-level actions (Access Feeds & Access Feed Support).

Note: Although permissions to perform the above actions are currently granted to groups, a future Kylo version may switch to a role-based mechanism similar to the entity-level access control (see below).

Entity-Level Authorization

Entity-level authorization is an additional, optional form of access control that applies to individual entities: templates, feeds, categories, etc. Entity-level access control is similar to service-level in that it involves granting permissions to perform a hierarchical set of actions. These actions, though, would apply only to an individual entity.

Entity-level access control is turned off by default. To activate this feature you must set this property to true in `kylo-services/conf/application.properties` and then restart Kylo:

```
security.entity.access.controlled=true
```

Warning: Turning on entity-level access control is a one-way operation; you cannot reset the above property back to false to deactivate this feature

Roles

Entity-level access control differs from service-level access control in that permissions are not granted to individual groups, rather they are granted to one or more **roles**. A role is a named, pre-configured set of granted permissions that may be applied to a group or individual user for a particular entity instance. Roles are defined and associated

with each kind of entity and may be granted permission to perform any of the actions defined for that entity type. The actual members (users or groups) of a role are associated at the entity-level, though, and grant permissions to perform actions on that entity only.

For instance, there might be the roles *Editor*, *Admin*, and *Read-Only* defined that grant varying sets of permissions for feeds. Adding a user, or any group that user belongs to, as a member of the *Editors* role of a specific feed will permit that user to make changes to it. A particular user might be a member of the *Editor* role for one feed, an *Admin* member of another feed, but only a *Read-Only* member of a third feed.

Default Roles

Kylo comes with a set of default roles for each kind of entity as described below.

Note: As of Kylo version 0.8.1, entity roles and their granted permissions are fixed. Future versions of Kylo will allow for creation and management of custom roles and assigned permissions.

Template Roles	
Editor	Allows a user to edit and export a template
Admin	All capabilities defined in the ‘Editor’ role along with the ability to change the permissions
Read-Only	Allows a user to view, but not modify, the template

Category Roles	
Editor	Allows a user to edit and delete feeds using this category
Admin	All capabilities defined in the ‘Editor’ role along with the ability to change the permissions
Read-Only	Allows a user to view the category
Feed Creator	Allows a user to create a new feed using this category

Feed Roles	
Editor	Allows a user to edit, enable/disable, delete, export, and access job operations of the feed
Admin	All capabilities defined in the ‘Editor’ role along with the ability to change the permissions
Read-Only	Allows a user to view the feed and access job operations

Data Source Roles	
Editor	Allows a user to edit and delete the datasource
Admin	All capabilities defined in the ‘Editor’ role along with the ability to change the permissions
Read-Only	Allows a user to view the datasource

Why Two Levels of Access Control?

Kylo support two levels access control because not all installations require the fine-grained control of entity-level authorization. Service-level authorization is generally easier to manage if your security requirements are not very selective or stringent. If you only need the ability to restrict some Kylo actions to certain select groups of users then service-level might be sufficient.

If your installation deals with sensitive information, and you need to be very selective of what data certain users and groups can see and manipulate, then you should use entity-level authorization to provide tight controls over that data.

Having two security schemes can pose management challenges as there is a bit of an overlap between the service-level and entity-level permissions, and both levels of access control must be satisfied for a user’s action to be successful. If you choose to use entity-level control then it may be helpful to loosen up the service-level access a bit more where the entity and service permissions are redundant. To help determine what permissions are needed to perform common Kylo activities, the next section describes both kinds of access requirements depending upon what actions are attempted in Kylo.

Roles and Permissions Required for Common Activities

To help understand and manage permissions required by users when using Kylo, the following tables show:

1. Common actions in Kylo
2. The default entity-level roles that permit those actions
3. Additional service-level permissions required to perform those actions

Template Actions

Action	Roles Permitted	Service-level Permissions
View template and its summary	Editor, Admin, Read-Only	Access Templates
Edit template and its details	Editor, Admin	Edit Templates
Delete template	Editor, Admin	Edit Templates
Export template	Editor, Admin	Export Templates
Grant permissions on template to users/groups	Admin	Edit Templates
Import template (new)	N/A	Import Templates
Import template (existing)	Editor, Admin	Import Templates, Edit Templates
Enable template	N/A	Admin Templates
Disable template	N/A	Admin Templates

Category Actions

Action	Roles Permitted	Service-level Permissions
View category and its summary	Editor, Admin, Feed Creator, Read-Only	Access Categories
Edit category summary	Editor, Admin	Edit Categories
View category and its details	Editor, Admin, Feed Creator	Access Categories
Edit category details	Editor, Admin	Edit Categories
Edit set user fields	Editor, Admin	Admin Categories
Delete category	Editor, Admin	Edit Categories
Create feeds under category	Feed Creator	Edit Categories
Grant permissions on category to users/groups	Admin	Edit Categories

Feed Actions

Action	Roles Permitted	Service-level Permissions
View feed and its details	Editor, Admin, Read-Only	Access Feeds
Edit feed summary	Editor, Admin	Edit Feeds
Edit feed details	Editor, Admin	Edit Feeds
Edit feed user fields	Editor, Admin	Admin Feeds
Delete feed	Editor, Admin	Admin Feeds
Enable feed	Editor, Admin	Edit Feeds
Disable feed	Editor, Admin	Edit Feeds
Export feed	Editor, Admin	Export Feeds
Import feed (new)	N/A	Import Feeds
Import feed (existing)	Editor, Admin	Import Feeds
View operational history of feed	Editor, Admin, Read-Only	Access Feeds
Grant permissions on feed to users/groups	Admin	Edit Feeds

Data Source Actions

Action	Roles Permitted	Service-level Permissions
View data source summary and use in data transformations	Editor, Admin, Read-Only	Access Data Sources
Edit data source summary	Editor, Admin	Edit Data Sources
View data source and its details	Editor, Admin	Access Data Sources
View data source details, including sensitive information	Editor, Admin	Admin Data Sources
Edit data source details	Editor, Admin	Edit Data Sources
Delete data source	Editor, Admin	Edit Data Sources
Grant permissions on data source to users/groups	Admin	Edit Data Sources

Enable Ranger Authorization

Prerequisite

Java

Java must be installed on all client nodes.

```
$ java -version
$ java version "1.8.0_92"
$ OpenJDK Runtime Environment (rhel-2.6.4.0.el6_7-x86_64 u95-b00)
$ OpenJDK 64-Bit Server VM (build 24.95-b01, mixed mode)

$ echo $JAVA_HOME
$ /opt/java/jdk1.8.0_92/
```

Kylo

This documentation assumes that you have Kylo installed and running on a cluster.

Optional: Delete/Disable HDFS/HIVE Global Policy

If you are using HDP sandbox, remove all HDFS/HIVE global policy.

Disable the HDFS Policy.

Ranger Access Manager Audit Settings

Service Manager / Sandbox_hadoop Policies

List of Policies : Sandbox_hadoop

Search for your policy...

Add New Policy

Policy ID	Policy Name	Status	Audit Logging	Groups	Users	Action
5	Sandbox_hadoop-1-20160229184056	Enabled	Enabled		ambari-qa	
7	HDFS Global Allow	Disabled	Enabled	public		

Success
Policy updated successfully

Disable the HIVE policy.

Ranger Access Manager Audit Settings

Service Manager / Sandbox_hive Policies

List of Policies : Sandbox_hive

Search for your policy...

Add New Policy

Policy ID	Policy Name	Status	Audit Logging	Groups	Users	Action
1	Sandbox_hive-1-20160229183752	Enabled	Enabled		ambari-qa	
2	Sandbox_hive-2-20160229183752	Enabled	Enabled		ambari-qa	
3	Hive Global Tables Allow	Disabled	Enabled	public		
9	Hive Global UDF Allow	Disabled	Enabled	public		
14	Call_Details_Table	Disabled	Enabled	IT Network		
15	Customer_Details_Table	Disabled	Enabled	Marketing		
16	Hive Demo Table Loader	Disabled	Enabled		hive	
17	Hive Demo UDF Loader	Disabled	Enabled		hive	

Create a NiFi Super User Policy in Hive

1. Login to Ranger UI.
2. Select Hive Repository.
3. Click on Add Policy.
4. Create a policy as shown in image below.

Policy Name : ranger_superuser_policy Select user : nifi Permission : All

Ranger Access Manager Audit Settings

Policy Name * ranger_superuser_policy ☒ enabled

Hive Database * ☒ include

table * ☒ include

Hive Column * ☒ include

Description

Audit Logging ☒ YES

User and Group Permissions :

Permissions	Select Group	Select User	Permissions	Delegate Admin
	<input type="text"/>	<input type="text"/>	<input checked="" type="checkbox"/> All <input type="checkbox"/> Alter <input type="checkbox"/> Create <input type="checkbox"/> Drop <input type="checkbox"/> Index <input type="checkbox"/> Lock <input type="checkbox"/> select <input type="checkbox"/> update	<input type="checkbox"/>

+

Add Cancel

Create a Hive User Policy in the HDFS Repository

1. Login to Ranger UI.
2. Select HDFS Repository.
3. Click on Add Policy.
4. Create a policy as shown in the image below.

```
Policy Name : hive_user_policy_kylo
Resource Path : /model.db/
                /app/warehouse/
                /etl/
```

Ranger Access Manager Audit Settings

Service Manager > Sandbox_hadoop Policies > Edit Policy

Edit Policy

Policy Details :

Policy ID: 19

Policy Name *: hive_user_policy_kylo **enabled**

Resource Path *: /model.db/ /app/warehouse/ /etl/ **recursive**

Description: Grant access to hive user access to kylo folders

Audit Logging: **YES**

User and Group Permissions :

Permissions	Select Group	Select User	Permissions	Delegate Admin
	Select Group	x: hive	Read Write Execute	
+				

Ranger authorization is configured successfully. Now create a feed from the Kylo UI and create feed for testing.

Enable Sentry Authorization

Prerequisite

Java

Java must be installed on all client nodes.

```
$ java -version
$ java version "1.8.0_92"
$ OpenJDK Runtime Environment (rhel-2.6.4.0.el6_7-x86_64 u95-b00)
$ OpenJDK 64-Bit Server VM (build 24.95-b01, mixed mode)

$ echo $JAVA_HOME
$ /opt/java/jdk1.8.0_92/
```

Cluster Requirements

- This documentation assumes that you have Kylo installed and running on a cluster.
- Kerberos is mandatory. For testing purposes, set `sentry.hive.testing.mode` to `true`.
- You must be running Hive Server2.
- In order to define policy for a role, you should have the user-group created on all nodes of a cluster, and you must then map each role to user-group.
- Only Sentry Admin can grant all access (create role, grant, revoke) to a user. You can add a normal user to Sentry admin group via Cloudera Manager.

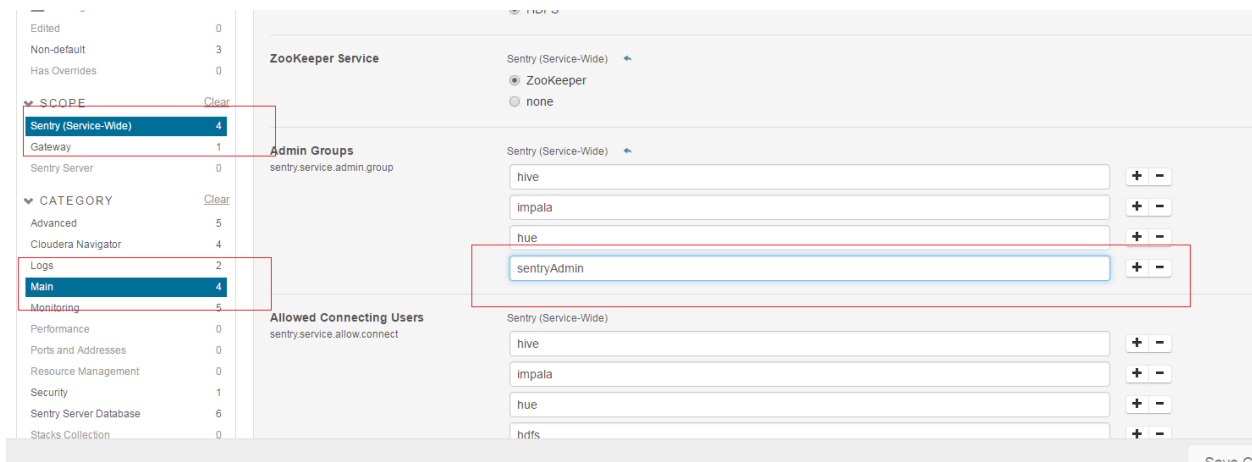
Grant Sentry Admin Access to NiFi User

1. Create a `sentryAdmin` group and assign a NiFi user to it.

```
groupadd sentryAdmin usermod -a -G sentryAdmin nifi
```

2. Add sentryAdmin group to Sentry Admin List.

- (a) Log in to Cloudera Manager.
- (b) Select Sentry Service.
- (c) Go to Configuration tab.
- (d) Select Sentry(Service-Wide) from Scope.
- (e) Select Main from Category.
- (f) Look for sentry.service.admin.group property.
- (g) Add sentryAdmin to list.
- (h) Click **Save** and **Restart Service**.



Enabling Sentry for Hive

Change Hive Warehouse Ownership

The Hive warehouse directory (/user/hive/warehouse or any path you specify as hive.metastore.warehouse.dir in your hive-site.xml) must be owned by the Hive user and group.

```
$ sudo -u hdfs hdfs dfs -chmod -R 771 /user/hive/warehouse
$ sudo -u hdfs hdfs dfs -chown -R hive:hive /user/hive/warehouse
```

If you have a Kerberos-enabled cluster:

```
$ sudo -u hdfs kinit -kt <hdfs.keytab> hdfs
$ sudo -u hdfs hdfs dfs -chmod -R 771 /user/hive/warehouse
$ sudo -u hdfs hdfs dfs -chown -R hive:hive /user/hive/warehouse
```

Disable Impersonation for HiveServer2

1. Go to the Hive service.
2. Click the Configuration tab.

3. Select Scope > HiveServer2.
4. Select Category > Main.
5. Uncheck the HiveServer2 Enable Impersonation checkbox.
6. Click **Save Changes** to commit the changes.

Yarn Setting For Hive User

1. Open the Cloudera Manager Admin Console and go to the YARN service.
2. Click the Configuration tab.
3. Select Scope > NodeManager.
4. Select Category > Security.
5. Ensure the Allowed System Users property includes the Hive user. If not, add Hive.
6. Click **Save Changes** to commit the changes.
7. Repeat steps 1-6 for every NodeManager role group for the YARN service that is associated with Hive.
8. Restart the YARN service.

Enabled Sentry

1. Go to the Hive service.
2. Click the Configuration tab.
3. Select Scope > Hive (Service-Wide).
4. Select Category > Main.
5. Locate the Sentry Service property and select Sentry.
6. Click **Save Changes** to commit the changes.
7. Restart the Hive service.

Configuration Switch to the classic layout Role Groups

Filters Clear

▼ STATUS

- Error 0
- Warning 0
- Edited 1
- Non-default 5
- Has Overrides 0

▼ SCOPE Clear

- Hive (Service-Wide) 11
- Gateway 1
- HiveServer2 2
- Hive Metastore Server 0
- WebHCat Server 0

▼ CATEGORY Clear

- Advanced 14
- Cloudera Navigator 5
- Hive Metastore Database 11
- Logs 5
- Main 11
- Monitoring 8
- Performance 0
- Policy File Based Sentry 4

Search

Show All Descriptions

Hive Warehouse Directory	Hive (Service-Wide)	?
hive.metastore.warehouse.dir	/user/hive/warehouse	
ZooKeeper Service	Hive (Service-Wide) <ul style="list-style-type: none"> <input checked="" type="radio"/> ZooKeeper <input type="radio"/> none 	?
MapReduce Service	Hive (Service-Wide) <ul style="list-style-type: none"> <input checked="" type="radio"/> YARN (MR2 Included) 	?
Sentry Service	Hive (Service-Wide) <ul style="list-style-type: none"> <input checked="" type="radio"/> Sentry <input type="radio"/> none 	?
Spark On YARN Service	Hive (Service-Wide) <ul style="list-style-type: none"> <input checked="" type="radio"/> Spark <input type="radio"/> none 	?
HBase Service	Hive (Service-Wide)	?

1 Edited Value **Save Changes**

Administrative Privilege

Once the sentryAdmin group is part of Sentry Admin list, it will be able to create policies in Sentry but sentryAdmin will not be allowed to read/write any tables. To do that, privileges must be granted to the sentryAdmin group.

```
CREATE ROLE admin_role GRANT ALL ON SERVER server1 TO ROLE admin_role; GRANT ROLE
admin_role TO GROUP sentryAdmin;
```

Enabled HDFS ACL

1. Go to the Cloudera Manager Admin Console and navigate to the HDFS service.
2. Click the Configuration tab.
3. Select Scope > HDFS-1 (Service-Wide).
4. Select Category > Security.
5. Locate the Enable Access Control Lists property and select its checkbox to enable HDFS ACLs.
6. Click **Save Changes** to commit the changes.

Property	Value	Scope	Category
Hadoop Secure Authorization	hadoop.security.authorization	HDFS (Service-Wide)	?
Authorized Users	HDFS (Service-Wide)	HDFS (Service-Wide)	?
Authorized Groups	HDFS (Service-Wide)	HDFS (Service-Wide)	?
Authorized Admin Users	HDFS (Service-Wide)	HDFS (Service-Wide)	?
Authorized Admin Groups	HDFS (Service-Wide)	HDFS (Service-Wide)	?
Enable Access Control Lists	dfs.namenode.acls.enabled	HDFS (Service-Wide)	?
Enable Sentry Synchronization		HDFS (Service-Wide)	?

Display 25 Per Page << < 1 2 > >>

1 Edited Value Reason for change... **Save Changes**

Sentry authorization is configured successfully. Now create a feed from the Kylo UI and test it.

Configurations and Install Examples

Below are instructions for how to configure both Kylo and NiFi to support integration with a Kerberos cluster. Provided are also some examples on how to enable Kerberos for your HDP or Cloudera sandbox if needed for development.

Kerberos Installation Example - HDP 2.4

Important: This document should only be used for DEV/Sandbox installation purposes. It is useful to help quickly Kerberize your Hortonworks sandbox so that you can test Kerberos features.

Prerequisite

Java

Java must be installed on all client nodes.

```
$ java version "1.7.0_80"  
$ Java(TM) SE Runtime Environment (build 1.7.0_80-b15)  
$ Java HotSpot(TM) 64-Bit Server VM (build 24.80-b11, mixed mode)  
  
$ echo $JAVA_HOME  
$ /usr/java/jdk1.7.0_80
```

Install Java Cryptography Extensions (JCE)

```
sudo wget -nv --no-check-certificate --no-cookies --header "Cookie:↵
↵oraclelicense=accept-securebackup-cookie"
http://download.oracle.com/otn-pub/java/jce/7/UnlimitedJCEPolicyJDK7.zip
-O /usr/java/jdk1.7.0_80/jre/lib/security/UnlimitedJCEPolicyJDK7.zip
cd /usr/java/jdk1.7.0_80/jre/lib/security

sudo unzip UnlimitedJCEPolicyJDK7.zip
sudo cp UnlimitedJCEPolicy/* .
#sudo rm -r UnlimitedJCEPolicy*

ls -l
```

Test Java Cryptography Extension

Create a java Test.java and paste below mentioned code in it.

```
$ vi Test.java

import javax.crypto.Cipher;
class Test {
public static void main(String[] args) {
try {
    System.out.println("Hello World!");
    int maxKeyLen = Cipher.getMaxAllowedKeyLength("AES");
    System.out.println(maxKeyLen);
} catch (Exception e){
    System.out.println("Sad world :(");
}
}
}
```

Compile:

```
$ javac Test.java
```

Run test. The expected number is: 2147483647.

```
$ java Test

Hello World!

2147483647
```

Install Kerberos

On a cluster, go to the master node for installation of Kerberos utilities.

1. Install a new version of the KDC server:

```
yum install krb5-server krb5-libs krb5-workstation
```

2. Using a text editor, open the KDC server configuration file, located by default here:
3. Change the [realms], as below, to sandbox.hortonworks.com. Update KDC and Admin Server Information.

4. Update `/var/kerberos/krb5kdc/kdc.conf`. Change the `[realms]` as `sandbox.hortonworks.com`.

[kdcdefaults]

```
kdc_ports = 88
kdc_tcp_ports = 88
```

[realms]

```
sandbox.hortonworks.com = {
#master_key_type = aes256-cts
acl_file = /var/kerberos/krb5kdc/kadm5.acl
dict_file = /usr/share/dict/words
admin_keytab = /var/kerberos/krb5kdc/kadm5.keytab
supported_enctypes = aes256-cts:normal aes128-cts:normal
des3-hmac-sha1:normal arcfour-hmac:normal des-hmac-sha1:normal
des-cbc-md5:normal des-cbc-crc:normal
}
```

5. Update `/var/kerberos/krb5kdc/kadm5.acl` and replace `EXAMPLE.COM` with `sandbox.hortonworks.com`.

```
*/admin@sandbox.hortonworks.com *
```

6. Create the Kerberos Database. Use the utility `kdb5_util` to create the Kerberos database. Enter the password: `thinkbig`.

```
kdb5_util create -s
```

7. Start the KDC. Start the KDC server and the KDC admin server.

```
/etc/rc.d/init.d/krb5kdc start
/etc/rc.d/init.d/kadmin start
```

8. When installing and managing your own MIT KDC, it is important to set up the KDC server to auto-start on boot.

```
chkconfig krb5kdc on
chkconfig kadmin on
```

9. Create a KDC admin by creating an admin principal. Enter the password: `thinkbig`.

```
kadmin.local -q "addprinc admin/admin"
```

10. Confirm that this admin principal has permissions in the KDC ACL. Using a text editor, open the KDC ACL file:

```
vi /var/kerberos/krb5kdc/kadm5.acl
```

11. Ensure that the KDC ACL file includes an entry that allows the admin principal to administer the KDC for your specific realm. The file should have an entry:

```
*/sandbox.hortonworks.com *
```

12. After editing and saving the `kadm5.acl` file, restart the `kadmin` process.

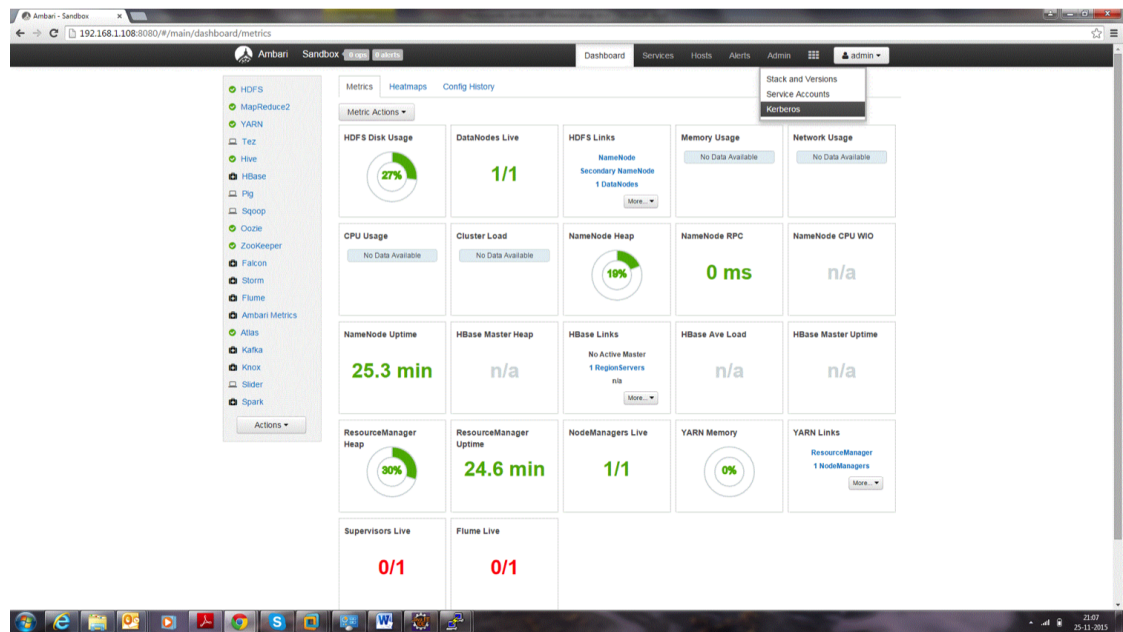
```
/etc/rc.d/init.d/kadmin restart
```

13. Create a user in Linux by typing the adduser command as shown below. We will use this user to test whether the Kerberos authentication is working or not. We will first run the command `hadoop fs -ls /` but switching to this user. And we will run the same command again when we enable Kerberos.

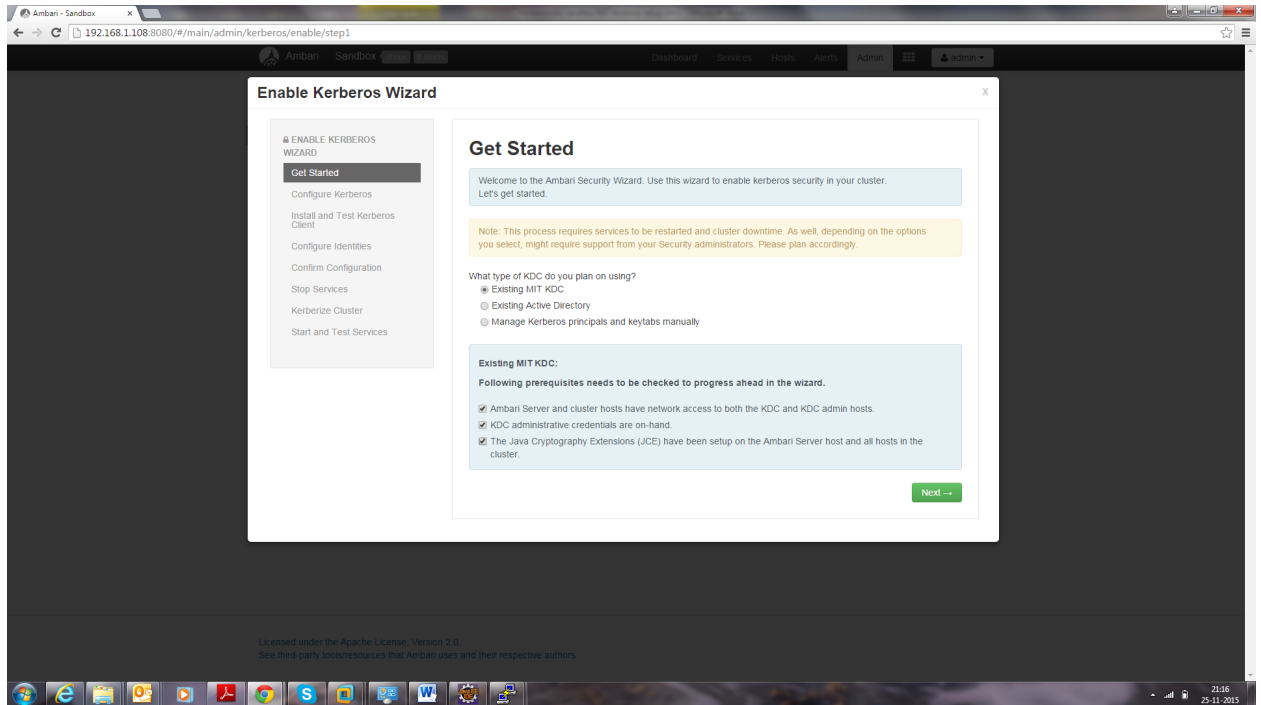
```
adduser testUser
su testUser
hadoop fs -ls /
```

Install Kerberos on an HDP Cluster

1. Open Ambari and then go to admin tab and select Kerberos.



2. Click on enable Kerberos. Then following screen will display. Tick the checkboxes as shown in this screenshot, then click Next.



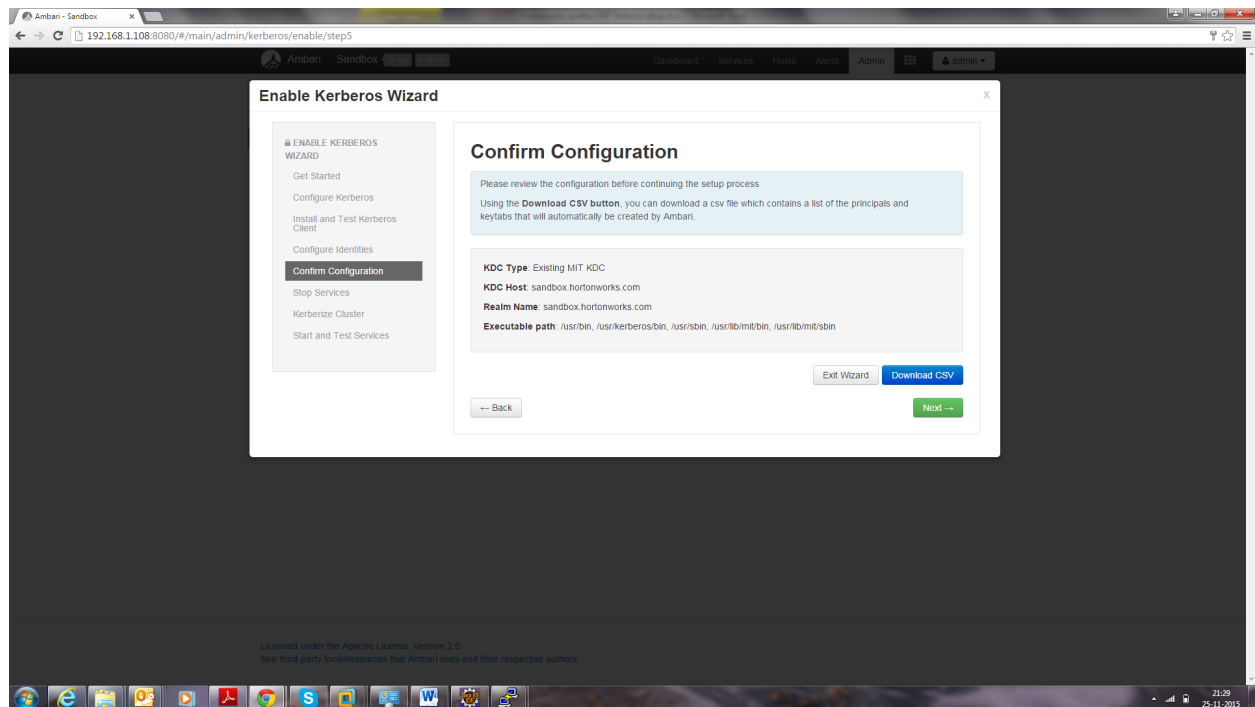
- Put `sandbox.hortonworks.com` in the KDC tab and click to test the KDC connection. Then, in Kadmin, put `sandbox.hortonworks.com` as host and admin principal as `*admin/admin@sandbox.hortonworks.com*`, and enter the password created in step 7.

Leave the advanced Kerberos-env and advanced krb5-conf as it is. And click **Next**.

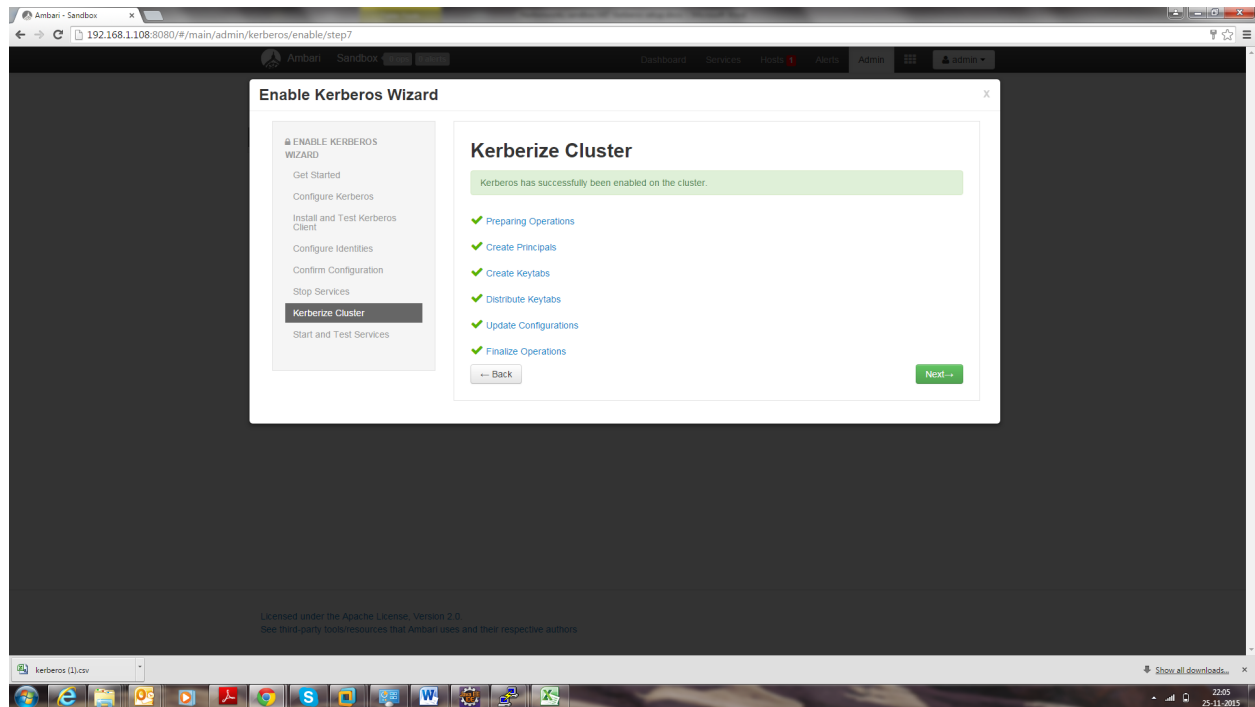
Global	
Keytab Dir	/etc/security/keytabs
Realm	HDP-TBRND-DEV
Spnego Principal	HTTP/_HOST@\${realm}
Spnego Keytab	\${keytab_dir}/spnego.service.keytab

Ambari Principals	
Smoke user principal	\${cluster-env/smokeuser}-\${cluster_name}@\${realm}
Smoke user keytab	\${keytab_dir}/smokeuser.headless.keytab
HDFS user principal	\${hadoop-env/hdfs_user}-\${cluster_name}@\${realm}
HDFS user keytab	\${keytab_dir}/hdfs.headless.keytab
HBase user principal	\${hbase-env/hbase_user}-\${cluster_name}@\${realm}
HBase user keytab	\${keytab_dir}/hbase.headless.keytab
Spark user principal	\${spark-env/spark_user}-\${cluster_name}@\${realm}
Spark user keytab	\${keytab_dir}/spark.headless.keytab

4. Download the .csv file and save it.



5. Click Next through the end of the process, until finally you can click **Complete**.



Kerberos Installation Example - Cloudera

Make sure all services started properly. Kerberos is successfully installed on the cluster.

KeyTab Generation

Create a keytab file for NiFi user.

```
kadmin.local
addprinc -randkey nifi@sandbox.hortonworks.com
xst -norandkey -k /etc/security/keytabs/nifi.headless.keytab
nifi@sandbox.hortonworks.co
exit

chown nifi:hadoop /etc/security/keytabs/nifi.headless.keytab
chmod 440 /etc/security/keytabs/nifi.headless.keytab
```

[Optional] You can initialize your keytab file using this command:

```
kinit -kt /etc/security/keytabs/nifi.headless.keytab nifi

>>>>>> Update
KerberosInstallation.adoc:docs/latest/security/KerberosInstallation.adoc
```

Make sure all services started properly. Kerberos is successfully installed on the cluster.

Kylo Kerberos SPNEGO

Configuration

Kylo

Kerberos SPNEGO is activated in Kylo by adding the profile `auth-krb-spnego` to the list of active profiles in the UI and services properties files.

Currently, if SPNEGO is activated, then the `auth-kylo` profile must be used as well, and all Kerberos-authenticated users must be present in the Kylo user store. Once activated, the following properties are required to configure Kerberos SPNEGO:

Property	Description
<code>security.auth.krb.service-principal</code>	Names the service principal used to access Kylo
<code>security.auth.krb.keytab</code>	Specifies path to the keytab file containing the service principal
<code>security.auth.kylo.login.username</code>	<i>(kylo-ui/application.properties only)</i> Specifies a username with the rights to retrieve all of the Kylo user store
<code>security.auth.kylo.login.password</code>	<i>(kylo-ui/application.properties only)</i> Specifies the password of the above username retrieving the user store

Note: Other authentication profile(s) must be activated in the *kylo-services/application.properties*, such as `auth-simple`, to allow the user/password specified in the `security.auth.kylo.login.*` properties above to be authenticated.

Kerberos

In addition to having a principal for every user present in your Kerberos KDC, you will also need to have a service principal of the form `HTTP/<Kylo host domain name>/@<YOUR REALM>` registered. This service principal should be exported into a keytab file and placed on file system of the host running Kylo (typically `/opt/kylo/kylo.keytab`). These values would then be used in the Kylo configuration as specified above.

Verifying Access

Once Kylo is configured for Kerberos SPNEGO, you can use `curl` to verify access. See the `curl --negotiate` option documentation (<https://curl.haxx.se/docs/manual.html>) to see the library requirements to support SPNEGO. Use the `-V` option to verify whether these requirements are met.

In these examples we will be accessing Kylo using URLs in the form: `http://localhost:8420/`. Therefore, `curl` will be requesting tickets from Kerberos for access to the service principle: `HTTP/localhost.localdomain@YOUR_REALM`.

If you use a different URL, say `http://host.example.com:8400/`, then the requested service principal will look like: `HTTP/host.example.com@YOUR_REALM`. In either case these service principals must be present in your KDC, exported into the keytab file, and the service principal name added to Kylo's configuration property `security.auth.krb.service-principal`.

First, log into Kerberos with your username ("myname" here) using `kinit`. The `@YOUR_REALM` part is optional if your KDC configuration has a default realm:

```
$ kinit myname@YOUR_REALM
```

Attempt to access the feeds API of kylo-services directly:

```
$ curl -v --negotiate -u : http://localhost:8420/api/v1/metadata/feed/
```

Attempt to access the same feeds API through the kylo-ui proxy:

```
$ curl -v --negotiate -u : http://localhost:8400/proxy/v1/metadata/feed/
```

Attempt to access the feeds HTML page on the kylo-ui:

```
$ curl -v --negotiate -u : http://localhost:8400/feed-mgr/index.html
```

Using the `-v` option causes `curl` to output the headers and status info exchanged with Kylo during the processing of the request before writing out the response. If Kerberos SPNEGO authentication was successful for each `curl` command, the output should include lines such as these:

```
> GET /proxy/v1/metadata/feed/ HTTP/1.1
< HTTP/1.1 401 Unauthorized
< WWW-Authenticate: Negotiate

> GET /proxy/v1/metadata/feed/ HTTP/1.1
> Authorization: Negotiate YII...
< HTTP/1.1 200 OK
```

This shows `curl`:

1. Attempt to get the feed resource
2. Receive an unauthorized response (401) and a challenge to negotiate authentication
3. Retry the request, but this time supplying the Kerberos ticket in an authorization header
4. Finally receiving a successful response (200)

Test Environment

The following links provide useful information on setting up your own KDC in a test environment:

- [Appendices of the Spring Kerberos Reference Documentation](#)
- [MIT Kerberos Admin Guide](#)

Overview

This guide provides details on what configuration changes are required to enable Kylo UI to use SSL. Broadly, the changes will be two-fold:

1. Changes to Kylo UI
2. Changes to Nifi

1. Changes to Kylo UI

1.1 Create Self-Signed Certificate in a Keystore

Lets assume you are in a development mode and you want to try out Kylo UI on SSL. You will need a self-signed certificate which is stored in a keystore. Make note of the kylo-ui.jks path, which we will refer to in the following section when updating Kylo UI properties.

If you are in production, you would have your certificate issued by a trusted certificate authority. You can then import it to your keystore.

```
mkdir /opt/kylo/ssl

# Generate keys and keystore
keytool -genkeypair -alias kylo-ui -dname cn=kylo-ui -validity 10000 -keyalg RSA -
↳keysize 2048 -keystore kylo-ui.jks -keypass changeit -storepass changeit

# Create certificate sign request
keytool -certreq -alias kylo-ui -file localhost.csr -keystore kylo-ui.jks -keypass_
↳changeit -storepass changeit

# Create certificate
keytool -gencert -alias kylo-ui -infile localhost.csr -outfile localhost.crt -ext_
↳SubjectAlternativeName=dns:localhost -keystore kylo-ui.jks -keypass changeit -
↳storepass changeit
```

```
# Import certificate into keystore
keytool -importcert -alias kylo-ui -file localhost.crt -keystore kylo-ui.jks -keypass_
↪changeit -storepass changeit

chown -R kylo /opt/kylo/ssl
```

1.2 Kylo UI Application Properties

Add following properties to `/opt/kylo/kylo-ui/conf/application.properties`. Change the port to your liking and update path to keystore 'kylo-ui.jks' we generated in previous section.

```
server.ssl.enabled=true
server.port=8444
server.ssl.key-store=/opt/kylo/ssl/kylo-ui.jks
server.ssl.key-store-password=changeit
server.ssl.key-store-type=jks
server.ssl.key-alias=kylo-ui
```

1.3 Restart Kylo UI

You can now restart Kylo UI and browse to <https://localhost:8444/ops-mgr/index.html>. The note protocol and port number have changed from default configuration and now are HTTPS and 8444 respectively. Since we are using a self-signed certificate, expect browsers to complain about inadequate security. That is okay for development purposes.

```
service kylo-ui restart
```

2. Changes to Nifi

2.1 Import Kylo UI's Certificate into a Truststore

You can either import Kylo UI's certificate 'localhost.crt', generated in step *1.1 Create Self-Signed Certificate in a Keystore*, into a new truststore; or, if you are in a hurry, simply re-use Kylo UI's keystore as Nifi's truststore.

Create a new truststore and import the cert to keep things clean. Make sure 'nifi' user has access to this truststore, e.g. keep the truststore in `/opt/nifi/data/ssl` directory, which belongs to 'nifi' user.

```
mkdir /opt/nifi/data/ssl

# Import certificate into keystore
keytool -importcert -alias kylo-ui -file localhost.crt -keystore kylo-ui-truststore.
↪jks -keypass changeit -storepass changeit

chown -R nifi /opt/nifi/data/ssl
```

2.2 Setup StandardSSLContextService in Nifi

There are two places where you need to add `StandardSSLContextService` in Nifi. One is on the root level next to all other controller services, and the other is in controller services next to Kylo Reporting Task. See *NiFi & Kylo Reporting Task* on what Reporting Task is.

Set following properties on SSL Context Service:

Truststore Filename /opt/nifi/data/ssl/kylo-ui-truststore.jks

Truststore Password changeit

Truststore Type JKS

Configure Controller Service

SETTINGS PROPERTIES COMMENTS

Required field +

Property	Value
Keystore Filename	No value set
Keystore Password	No value set
Key Password	No value set
Keystore Type	No value set
Truststore Filename	/opt/nifi/truststores/kylo-ui-truststore.jks
Truststore Password	Sensitive value set
Truststore Type	JKS
SSL Protocol	TLS

CANCEL APPLY

2.3 Update MetadataProviderSelectorService

Just like StandardSSLContextService, you will need to update two instances of MetadataProviderSelectorService: one at root level and one next to Kylo Reporting Task.

Set the following properties on MetadataProviderSelectorService, making sure host and port correspond to where Kylo UI is running:

REST Client URL <https://localhost:8444/proxy/metadata>

SSL Context Service StandardSSLContextService

Configure Controller Service

SETTINGS PROPERTIES COMMENTS

Required field +

Property	Value
Implementation	REST API
REST Client URL	https://localhost:8444/proxy/metadata
REST Client User Name	dladmin
REST Client Password	Sensitive value set
SSL Context Service	StandardSSLContextService →

CANCEL APPLY

Setup A NiFi Cluster in a Kylo Sandbox

Purpose

This document is intended for advanced NiFi users who wish to run a NiFi cluster in their Kylo sandbox. The NiFi cluster is intended for testing of failover scenarios only.

Prerequisite

You will need to have set up a Kylo sandbox according to the *Setup Wizard Deployment Guide*.

Install a Second NiFi Node

Each new node in a NiFi cluster should be a fresh install to ensure that the new node starts with an empty repository. You will then configure the new node and enable NiFi clustering.

1. Rename the existing NiFi directory to make room for the new install:

```
service nifi stop
mv /opt/nifi /opt/nifi-temp
```

2. Reinstall NiFi using the Kylo install wizard:

```
/opt/kylo/setup/nifi/install-nifi.sh
/opt/kylo/setup/java/change-nifi-java-home.sh /opt/java/current
/opt/kylo/setup/nifi/install-kylo-components.sh
```

3. Rename the new NiFi directory and restore the old NiFi directory:

```
service nifi stop
mv /opt/nifi /opt/nifi-2
mv /opt/nifi-temp /opt/nifi
```

4. Create a new init.d script for nifi-2 by changing the NiFi path:

```
sed 's#/opt/nifi#/opt/nifi-2#' /etc/init.d/nifi > /etc/init.d/nifi-2
chmod 744 /etc/init.d/nifi-2
```

5. Create a log directory for nifi-2:

```
mkdir /var/log/nifi-2
chown nifi:nifi /var/log/nifi-2
sed -i 's#NIFI_LOG_DIR=".*"#NIFI_LOG_DIR="/var/log/nifi-2"#' /opt/nifi-2/current/bin/
↪nifi-env.sh
```

6. Edit /opt/nifi-2/current/conf/nifi.properties and replace all references to /opt/nifi with /opt/nifi-2:

```
sed -i 's#/opt/nifi#/opt/nifi-2#' /opt/nifi-2/current/conf/nifi.properties
```

Enable NiFi Clustering

Each node in the NiFi cluster will need to be configured to connect to the cluster.

1. Edit the /opt/nifi/current/conf/nifi.properties file:

```
nifi.cluster.is.node=true
nifi.cluster.node.address=localhost
nifi.cluster.node.protocol.port=8078
nifi.zookeeper.connect.string=localhost:2181
```

2. Edit the /opt/nifi-2/current/conf/nifi.properties file:

```
nifi.web.http.port=8077
nifi.cluster.is.node=true
nifi.cluster.node.address=localhost
nifi.cluster.node.protocol.port=8076
nifi.zookeeper.connect.string=localhost:2181
```

Start Each Node

Now that your cluster is created and configured, start the services:

```
service nifi start
service nifi-2 start
```

Don't forget to open up the nifi.web.http.port property's port number in your VM.

You should be able to open the NiFi UI under either <http://localhost:8079> or <http://localhost:8077> and see in the upper left a cluster icon and 2/2.

This link provides additional instruction for enabling SSL for NiFi:

Creating a Self-signed Cert

1. Download the NiFi toolkit from <https://nifi.apache.org/download.html>
2. Unzip it to a directory.

```
/opt/nifi/nifi-toolkit-1.0.0
```

3. Go into that directory.

```
cd /opt/nifi/nifi-toolkit-1.0.0/bin
```

4. Update the “tls-toolkit.sh” file and add the current version of JAVA_HOME.

(a) Add this line to the start of the script:

```
export JAVA_HOME=/opt/java/current
```

5. Make an SSL directory under /opt/nifi/data as the nifi owner:

```
mkdir /opt/nifi/data/ssl  
chown nifi /opt/nifi/data/ssl
```

6. Change to that directory and generate certs using the tls-toolkit.

```
cd /opt/nifi/data/ssl  
/opt/nifi/nifi-toolkit-1.0.0/bin/tls-toolkit.sh standalone -n 'localhost' -C  
→ 'CN=kylo, OU=NIFI' -o .  
  
.. note:: Use the hostname of the NiFi node if running Kylo and NiFi on different_  
→ servers to prevent certificate issues
```

This will generate one client cert and password file along with a server keystore and trust store:

```
-rwxr-xr-x 1 nifi root 1675 Apr 26 21:28 nifi-key.key
-rwxr-xr-x 1 nifi root 1200 Apr 26 21:28 nifi-cert.pem
-rwxr-xr-x 1 nifi root  43 Apr 26 21:28 CN=kylo_OU=NIFI.password
-rwxr-xr-x 1 nifi root 3434 Apr 26 21:28 CN=kylo_OU=NIFI.p12
drwxr-xr-x 2 nifi root 4096 Apr 26 21:46 localhost
```

Note: The client cert is the p.12 (PKCS12) file along with its respective password. This will be needed later when you add the client cert to the browser/computer.

The directory 'localhost' is for the server side keystore and truststore .jks files.

```
-rwxr-xr-x 1 nifi root 3053 Apr 26 21:28 keystore.jks
-rwxr-xr-x 1 nifi root  911 Apr 26 21:28 truststore.jks
-rwxr-xr-x 1 nifi root 8921 Apr 26 21:28 nifi.properties
```

7. Change permissions on files.

```
chown nifi -R /opt/nifi/data/ssl/*
chmod 755 -R /opt/nifi/data/ssl/*
```

8. Merge the generated properties (/opt/nifi/data/ssl/localhost) with the the NiFi configuration properties (/opt/nifi/current/conf/nifi.properties).

- (a) Open the /opt/nifi/data/ssl/localhost/nifi.properties file.
- (b) Compare and update the below properties

Note: Below is an example. Do not copy this text directly, as your keystore/truststore passwords will be different!

```
# Site to Site properties
nifi.remote.input.host=localhost
nifi.remote.input.secure=true
nifi.remote.input.socket.port=10443
nifi.remote.input.http.enabled=true
nifi.remote.input.http.transaction.ttl=30 sec

# web properties #
nifi.web.war.directory=./lib
nifi.web.http.host=
nifi.web.http.port=
nifi.web.https.host=0.0.0.0
nifi.web.https.port=9443
nifi.web.jetty.working.directory=./work/jetty
nifi.web.jetty.threads=200

# security properties #
nifi.sensitive.props.key=
nifi.sensitive.props.key.protected=
nifi.sensitive.props.algorithm=PBEWITHMD5AND256BITAES-CBC-OPENSSL
nifi.sensitive.props.provider=BC
nifi.sensitive.props.additional.keys=
```

```
nifi.security.keystore=/opt/nifi/data/ssl/localhost/keystore.jks
nifi.security.keystoreType=jks
nifi.security.keystorePasswd=fCrusEdGOKdik7P5UORRegQOILOZTBQ+9kyhf8D+PUU
nifi.security.keyPasswd=fCrusEdGOKdik7P5UORRegQOILOZTBQ+9kyhf8D+PUU
nifi.security.truststore=/opt/nifi/data/ssl/localhost/truststore.jks
nifi.security.truststoreType=jks
nifi.security.truststorePasswd=DHJS0+HIaUMRkhrbqlK/ys5j7iL/ef9mnGJIDRlFokA
nifi.security.needClientAuth=
nifi.security.user.authorizer=file-provider
nifi.security.user.login.identity.provider=
nifi.security.ocsp.responder.url=
nifi.security.ocsp.responder.certificate=
```

9. Edit the `/opt/nifi/data/conf/authorizers.xml` file to add the initial admin identity. This entry needs to match the phrase you used to generate the certificates in step 6.

```
<property name="Initial Admin Identity">CN=kylo,OU=NIFI</property>
```

Here is an example:

```
<authorizer>
  <identifier>file-provider</identifier>
  <class>org.apache.nifi.authorization.FileAuthorizer</class>
  <property name="Authorizations File">./conf/authorizations.xml</property>
  <property name="Users File">./conf/users.xml</property>
  <property name="Initial Admin Identity">CN=kylo, OU=NIFI</property>
  <property name="Legacy Authorized Users File"></property>

  <!-- Provide the identity (typically a DN) of each node when clustered, see above,
  ↳description of Node Identity.
  <property name="Node Identity 1"></property>
  <property name="Node Identity 2"></property>
  -->
</authorizer>
```

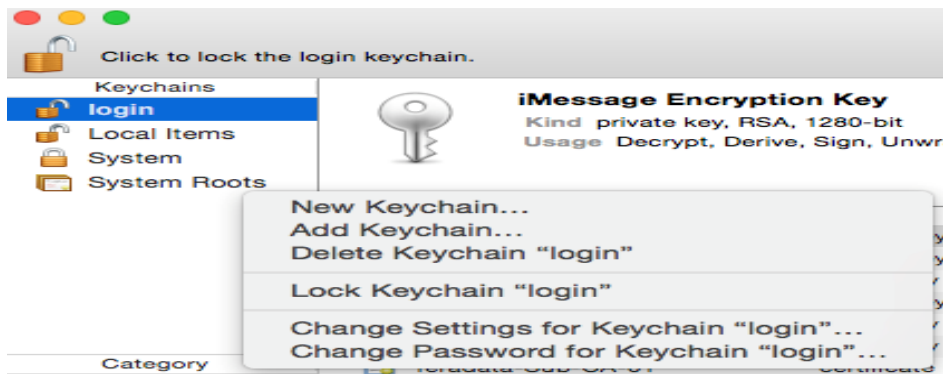
For reference: This will create a record in the `/opt/nifi/current/conf/users.xml`. Should you need to regenerate your SSL file with a different CN, you will need to modify the `users.xml` file for that entry.

10. Set the following parameters in the `kylo-services "application.properties"` file for the NiFi connection.

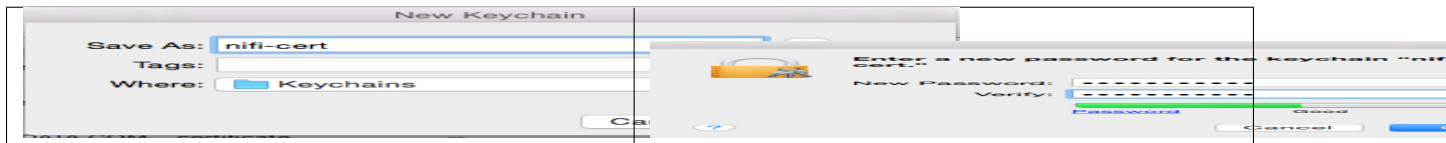
```
nifi.rest.host=localhost
nifi.rest.https=true
### The port should match the port found in the /opt/nifi/current/conf/nifi.
↳properties (nifi.web.https.port)
nifi.rest.port=9443
nifi.rest.useConnectionPooling=false
nifi.rest.truststorePath=/opt/nifi/data/ssl/localhost/truststore.jks
##the truststore password below needs to match that found in the nifi.properties file,
↳(nifi.security.truststorePasswd)
nifi.rest.truststorePassword=UsqLPVksIe/taZbfpVIsYElF8qFLhXbeVGRgB0pLjKE
nifi.rest.truststoreType=JKS
nifi.rest.keystorePath=/opt/nifi/data/ssl/CN=kylo_OU=NIFI.p12
###value found in the .password file /opt/nifi/data/ssl/CN=kylo_OU=NIFI.password
nifi.rest.keystorePassword=mw5ePri
nifi.rest.keystoreType=PKCS12
```

Importing the Client Cert on the Mac

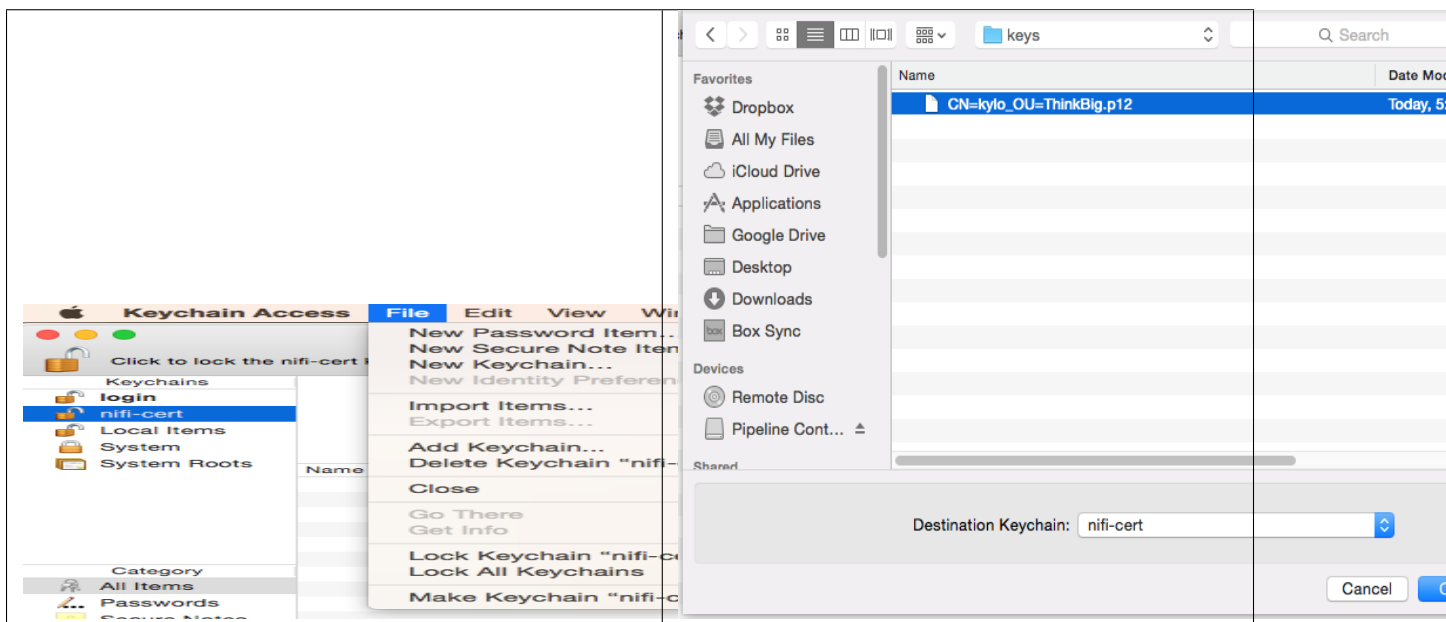
1. Copy the .p12 file that you created above (/opt/nifi/data/ssl/CN=kylo_OU=NIFI.p12) in step 6 to your Mac.
2. Open Keychain Access.
3. Create a new keychain with a name. The client cert is copied into this new keychain, which in the example here is named “nifi-cert”. If you add it directly to the System, the browser will ask you for the login/pass every time NiFi does a request.
 - (a) In the left pane, right-click “Keychains” and select “New Keychain”.



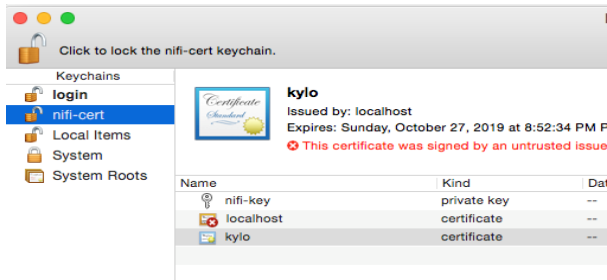
- (b) Give it the name “nifi-cert” and a password.



4. Once the keychain is created, click on it and select File -> import Items, and then find the .p12 file that you copied over in step 1.



Once complete you should have something that looks like this:

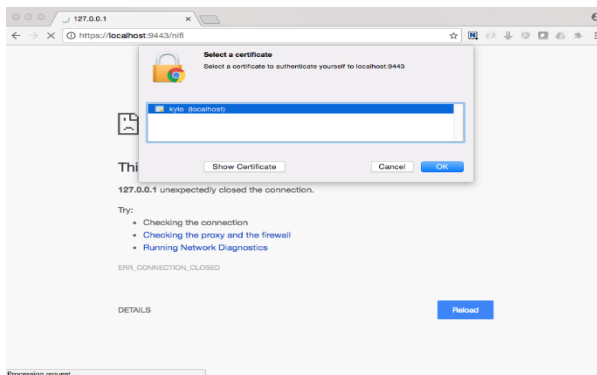


Accessing NiFi under SSL

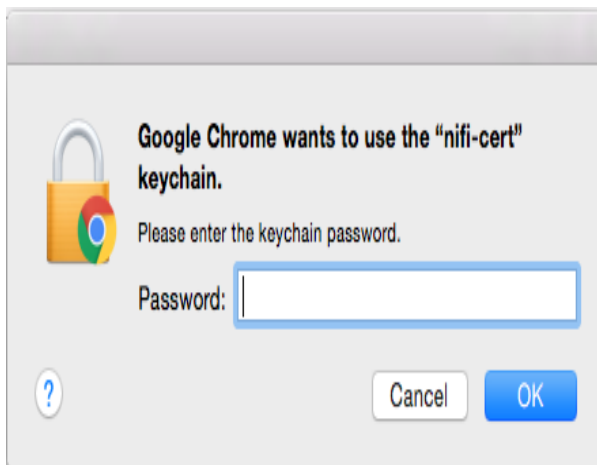
Open the port defined in the NiFi.properties above: 9443.

The first time you connect to NiFi (<https://localhost:9443/nifi>) you will be instructed to verify the certificate. This will only happen once.

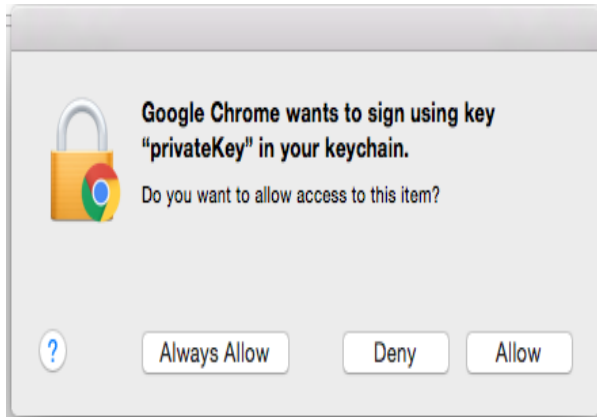
1. Click **OK** at the dialog prompt.



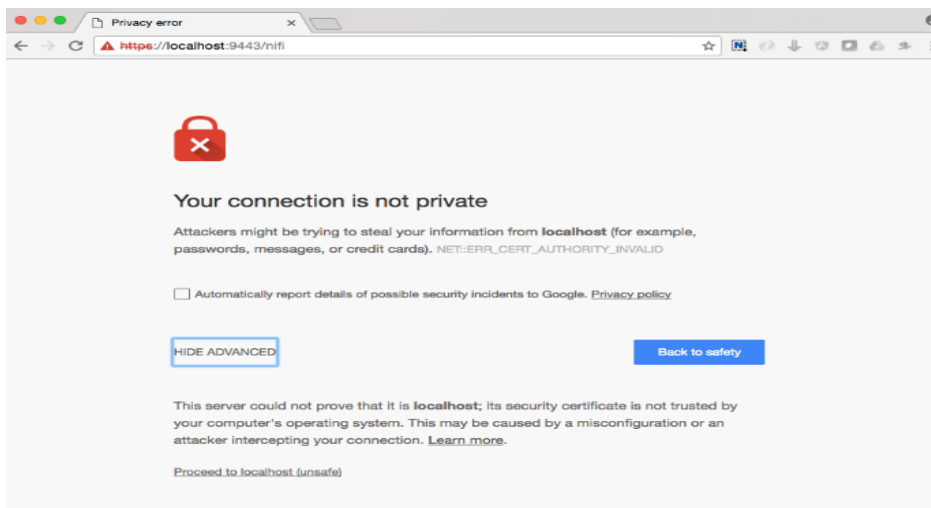
2. Enter the Password that you supplied for the keychain. This is the password that you created for the keychain in “Importing the Client Cert on the Mac” Step 3b.



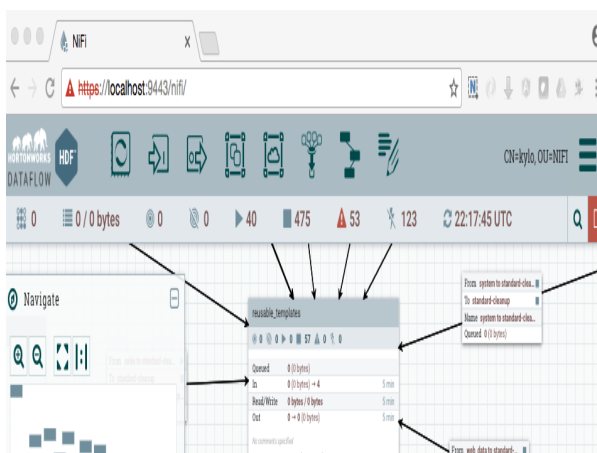
3. Click Always Verify.



4. Click AdvancKyloConfiguration.rsted and then Click Proceed. It will show up as “not private” because it is a self-signed cert.



5. NiFi under SSL. Notice the User name matches the one supplied via the certificate that we created: “CN=kylo, OU=NIFI”.



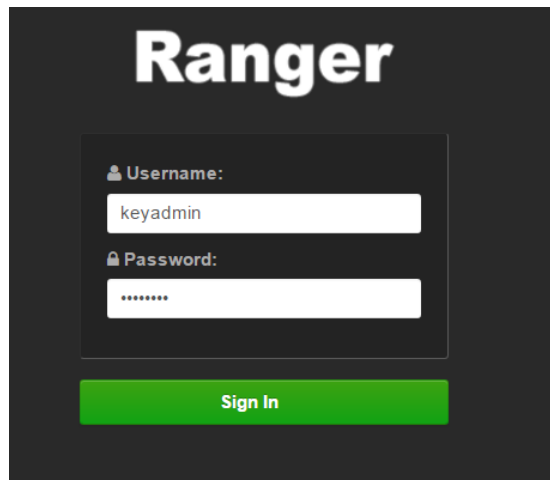
Refer to the Hortonworks documentation on Enabling SSL for NiFi:

Configuring NiFi for HDFS Encryption

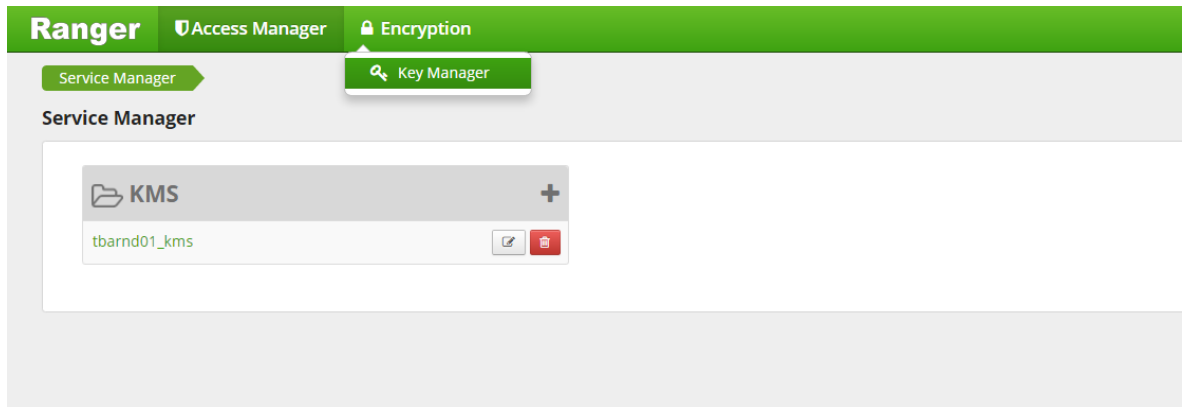
Key Creation Process

1. Log in to Ranger KMS UI.

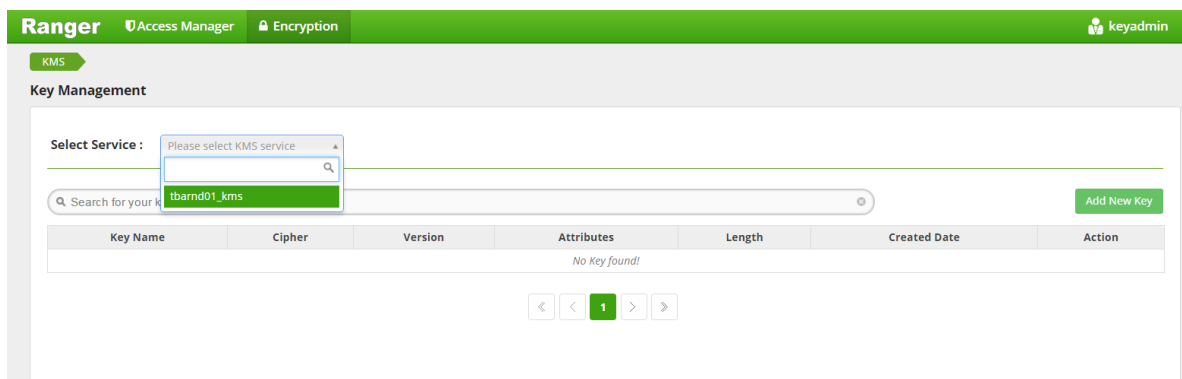
<Hostname>:6080



2. Provide **Username** as 'keyadmin' and password for user.
3. Go to the Encryption tab and click **Key Manager**.



4. Select the appropriate defined service from list.



5. Click **Add New Key**.
6. Fill out the Key Detail fields.

Ranger Access Manager Encryption

KMS > tbarnd01_kms > Key Create

Key Detail

Key Name *

Cipher

Length

Description

Attributes

Name	Value
<input type="text"/>	<input type="text"/>

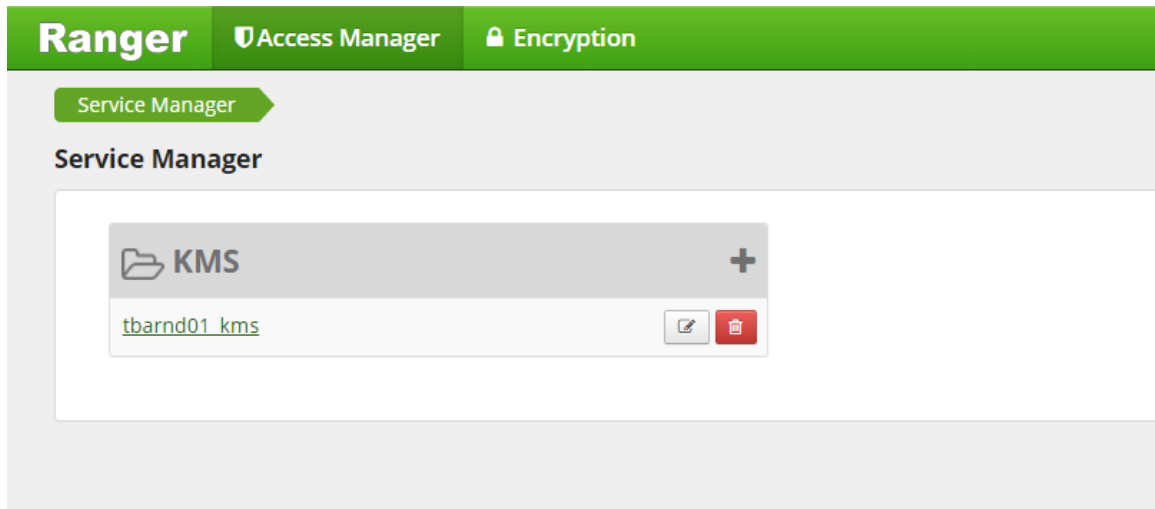
7. Click **Save**.

Now the Key has been successfully created, and it can be used for creating an encryption zone.

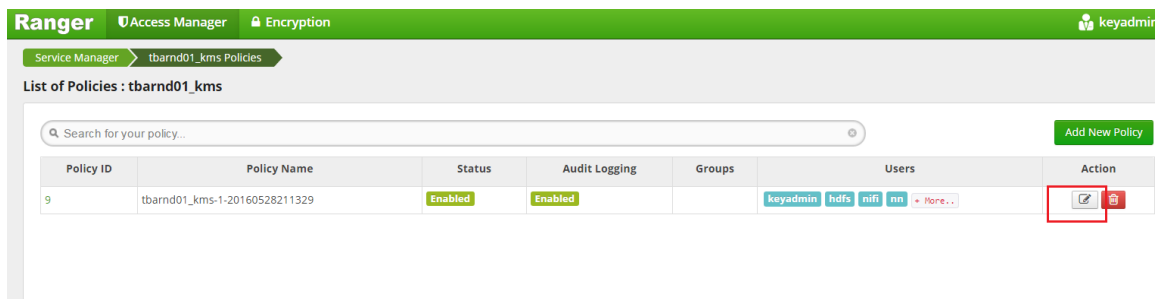
Permission Definition

The next task is to provide necessary permissions to a user who will run the NiFi application. In our case, we are using a NiFi user for running the application and HDFS as a super user operation.

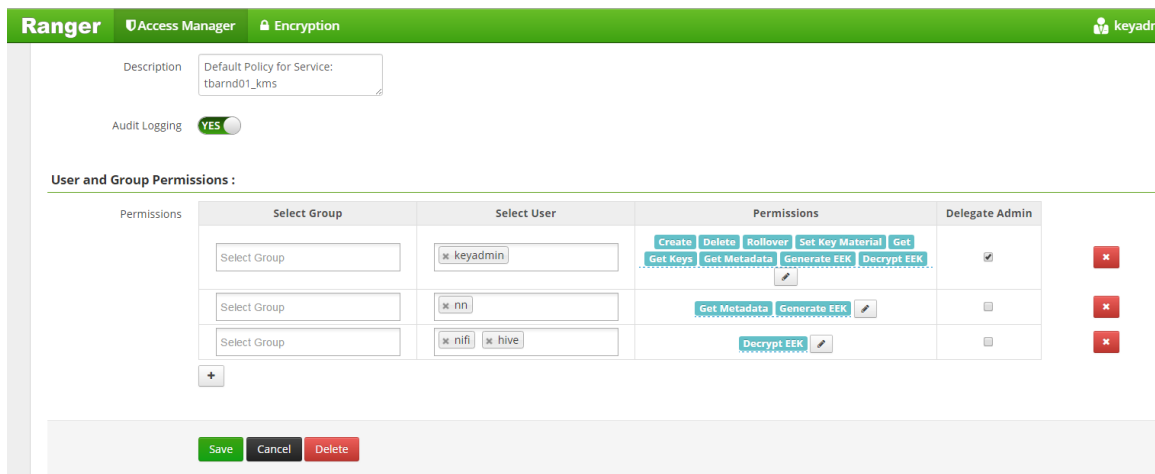
1. Click on **Service**.



- Click on the edit icon present at right side.



- Go to bottom of page, you will see User and Group Permissions tab.



- Provide appropriate permissions to the NiFi user.

Configure CreateHDFSFolder Processor

- Right-click **Processor** and select **Configure**.
- Configure the highlighted property for the processor.

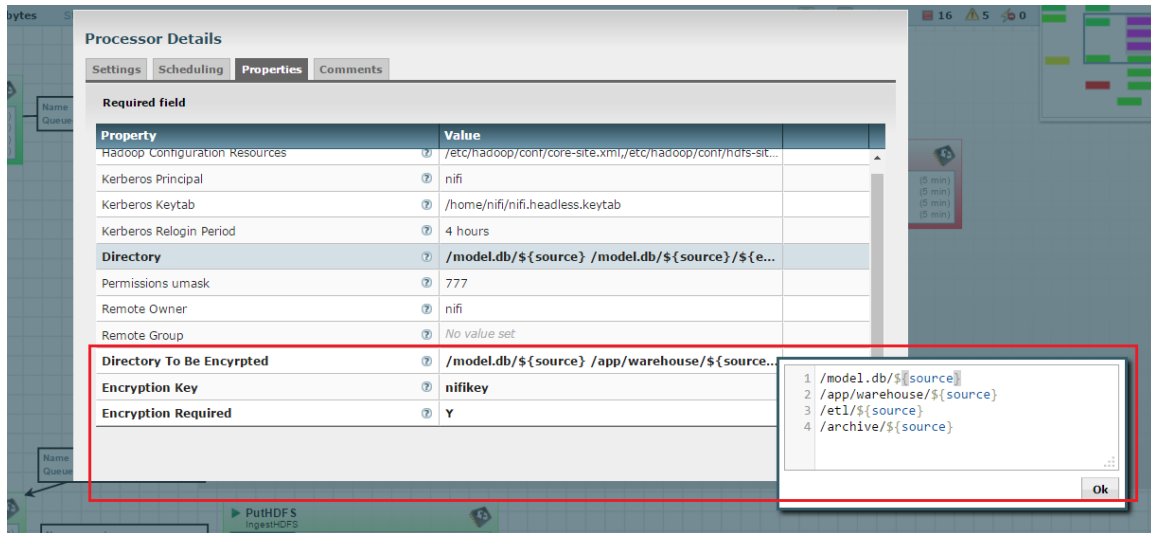
```

Directory To Be Encrypted:  /model.db/${source}
                           /app/warehouse/${source}
                           /etl/${source}
                           /archive/${source}

```

Encryption Key: nifikey

Encryption Required: Y



3. Click **OK** and start the processor.

You have successfully configured NiFi DataLake Platform for HDFS Encryption.

NiFi & Kylo Reporting Task

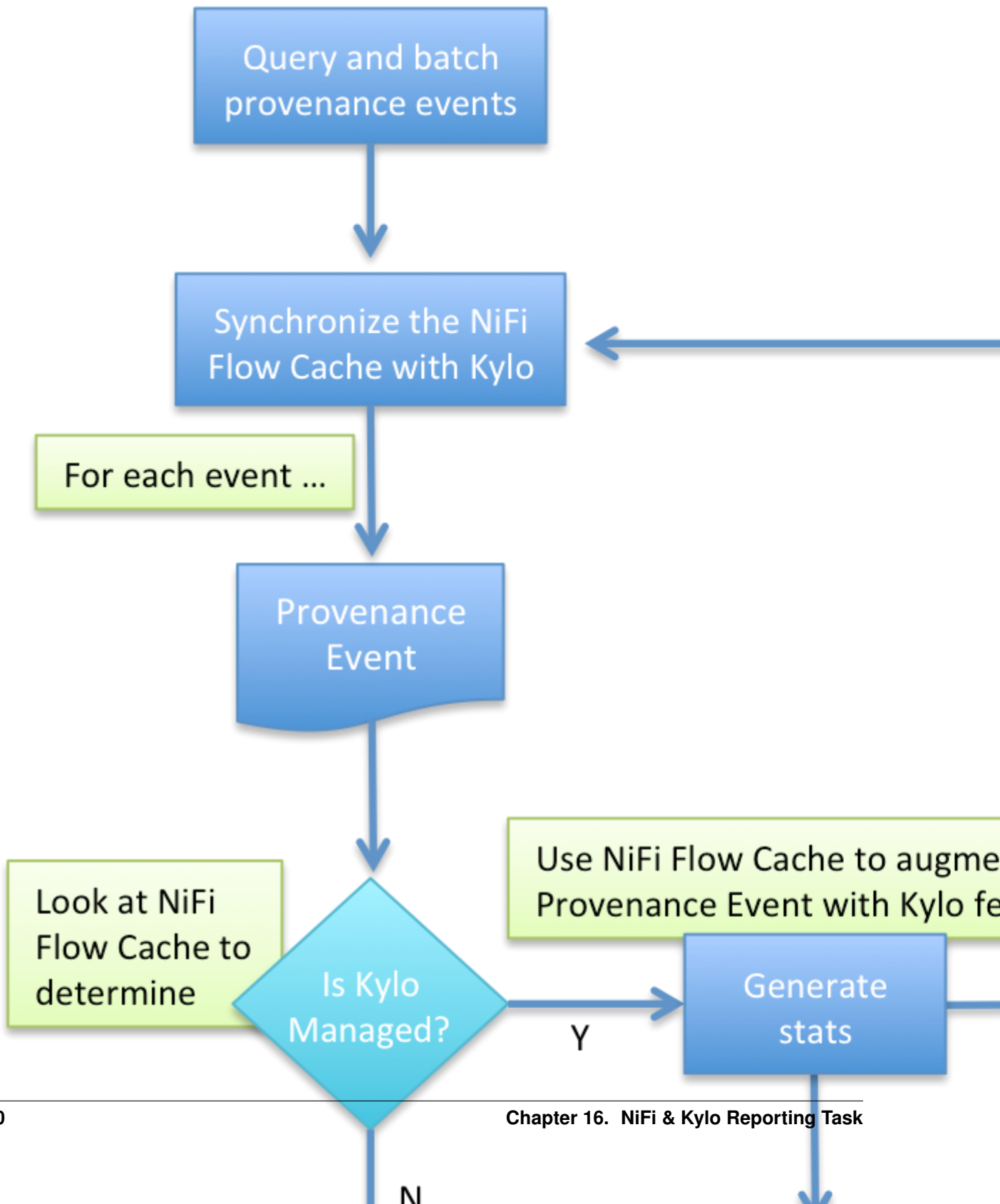
Introduction

Kylo communicates with NiFi via a NiFi reporting task. As flow files run through NiFi, each processor creates provenance events that track metadata and status information of a running flow. A NiFi reporting task is used to query for these provenance events and send them to Kylo to display job and step executions in Kylo's operations manager.

Processing Provenance Events

The Kylo reporting task relies on Kylo to provide feed information, which it uses to augment the provenance event giving the NiFi event feed context. It does this through a cache called the "NiFi Flow Cache", which is maintained by the Kylo Feed manager and kept in sync with the NiFi reporting task. As feeds are created and updated, this cache is updated and synchronized back to the NiFi reporting task upon processing provenance events. The cache is exposed through a REST API, which is used by the reporting task.

NiFi Reporting Task



The NiFi Flow Cache REST API

NiFi - Provenance : event reporting

[Show/Hide](#)

GET	/v1/metadata/nifi-provenance/max-event-id	Gets the maximum event id
GET	/v1/metadata/nifi-provenance/nifi-flow-cache/available	
GET	/v1/metadata/nifi-provenance/nifi-flow-cache/cache-summary	
GET	/v1/metadata/nifi-provenance/nifi-flow-cache/get-cache	
GET	/v1/metadata/nifi-provenance/nifi-flow-cache/get-flow-updates	
GET	/v1/metadata/nifi-provenance/nifi-flow-cache/preview-flow-updates	
GET	/v1/metadata/nifi-provenance/nifi-flow-cache/reset-cache	
GET	/v1/metadata/nifi-provenance/nifi-flow-cache/reset-flow-updates	Resets the updates

The above REST endpoints allow you to manage the cache. Kylo and the reporting task will automatically keep the cache in sync. If needed you can use these REST endpoints to manage, view, and reset the cache.

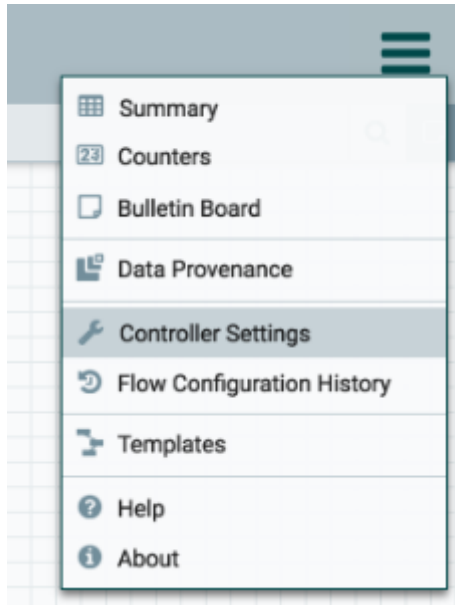
Note: If for some reason the reporting task is reporting Kylo as “not available”, you can try to reset the cache to fix the problem using the “reset-cache” endpoint.

Reporting Task Creation

When Kylo starts up, it will attempt to auto create the controller service and reporting task in NiFi that is needed to communicate with Kylo. If this process doesn’t work, or if you want more control, you can manually create it following the steps below.

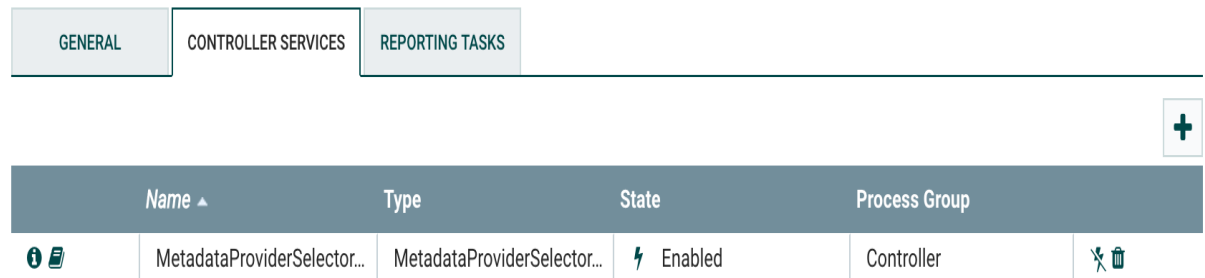
Manual Setup

1. To setup the reporting task, click the menu icon on the top right and click the “Controller Settings” link.



- From there we need to setup a **Controller Service** before adding the Reporting task. The Controller Service is used to allow NiFi to talk to Kylo REST endpoints that gather feed information needed for processing NiFi events. Setup a new **Metadata Provider Selection Service** and set the properties to communicate with your Kylo instance.

NiFi Settings



SETTINGS	PROPERTIES	COMMENTS
----------	------------	----------

Required field

Property		Value
Implementation	?	REST API
REST Client URL	?	http://localhost:8400/proxy/metadata
REST Client User Name	?	dladmin
REST Client Password	?	Sensitive value set
SSL Context Service	?	No value set

3. Next add the reporting task.

Configure Reporting Task

SETTINGS	PROPERTIES	COMMENTS
----------	------------	----------

Required field

Property		Value
Metadata Service	?	MetadataProviderSelectorService
Max batch feed events per second	?	10
JMS event group size	?	50
Rebuild cache on restart	?	false
Last event id not found value	?	MAX_EVENT_ID
Initial event id value	?	LAST_EVENT_ID
Processing batch size	?	500

A rundown of the various properties can be found by hovering over the ? icon or at the bottom of this page: **Kylo Provenance Event Reporting Task Properties**.

4. Set the schedule on the reporting task.

It is recommended to set the schedule between 5 and 15 seconds. On this interval the system will run and query for all events that haven't been processed.

Configure Reporting Task

SETTINGS	PROPERTIES	COMMENTS
Name	KyloProvenanceEventReportingTask	Scheduling Strategy ? Timer driven
Id	99c327c4-0159-1000-6a02-2300b816d66f	Run Schedule ? 5 sec
Type	KyloProvenanceEventReportingTask	

Reporting Task Properties

Name	Default Value	Allowable Values	Description
Metadata Service		Controller Service API: MetadataProviderService Implementation:	Kylo metadata service
Max batch feed events per second	10		The maximum number of events/second for a given feed allowed to go through to Kylo. This is used to safeguard Kylo against a feed that starts acting like a stream Supports Expression Language: true
JMS event group size	50		The size of grouped events sent over to Kylo. This should be less than the Processing Batch Size Supports Expression Language: true
Rebuild cache on restart	false		Should the cache of the flows be rebuilt every time the Reporting task is restarted? By default, the system will keep the cache up to date; however, setting this to true will force the cache to be rebuilt upon restarting the reporting task. Supports Expression Language: true
Last event id not found value	KYLO	KYLO ZERO MAX_EVENT_ID	If there is no minimum value to start the range query from (i.e. if this reporting task has never run before in NiFi) what should be the initial value?" KYLO: It will attempt to query Kylo for the last saved id and use that as the latest id ZERO: this will get all events starting at 0 to the latest event id. MAX_EVENT_ID: this is set it to the max provenance event. This is the default setting
Initial event id value	LAST_EVENT_ID	LAST_EVENT_ID KYLO MAX_EVENT_ID	Upon starting the Reporting task what value should be used as the minimum value in the range of provenance events this task should query?
156		Chapter 16. NiFi & Kylo Reporting Task	
			LAST_EVENT_ID: will use the last event successfully processed from this

ImportSqoop Processor

About

The ImportSqoop processor allows loading data from a relational system into HDFS. This document discusses the setup required to use this processor.

Starter template

A starter template for using the processor is provided at:

```
samples/templates/nifi-1.0/template-starter-sqoop-import.xml
```

Configuration

For use with Kylo UI, configure values for the two properties (**nifi.service.<controller service name in NiFi>.password**, **config.sqoop.hdfs.ingest.root**) in the below section in the properties file: **/opt/kylo/kylo-services/conf/application.properties**

```
### Sqoop import
# DB Connection password (format: nifi.service.<controller service name in NiFi>.
↪password=<password>
#nifi.service.sqoop-mysql-connection.password=hadoop
# Base HDFS landing directory
#config.sqoop.hdfs.ingest.root=/sqoopimport
```

Note: The **DB Connection password** section will have the name of the key derived from the controller service name in NiFi. In the above snippet, the controller service name is called **sqoop-mysql-connection**.

Drivers

Sqoop requires the JDBC drivers for the specific database server in order to transfer data. The processor has been tested on MySQL, Oracle, Teradata and SQL Server databases, using Sqoop v1.4.6.

The drivers need to be downloaded, and the .jar files must be copied over to Sqoop's /lib directory.

As an example, consider that the MySQL driver is downloaded and available in a file named: **mysql-connector-java.jar**.

This would have to be copied over into Sqoop's /lib directory, which may be in a location similar to: **/usr/hdp/current/sqoop-client/lib**.

The driver class can then be referred to in the property **Source Driver** in **StandardSqoopConnectionService** controller service configuration. For example: **com.mysql.jdbc.Driver**.

Tip: Avoid providing the driver class name in the controller service configuration. Sqoop will try to infer the best connector and driver for the transfer on the basis of the **Source Connection String** property configured for **StandardSqoopConnectionService** controller service.

Passwords

The processor's connection controller service allows three modes of providing the password:

1. Entered as clear text
2. Entered as encrypted text
3. Encrypted text in a file on HDFS

For modes (2) and (3), which allow encrypted passwords to be used, details are provided below:

Encrypt the password by providing the:

1. Password to encrypt
2. Passphrase
3. Location to write encrypted file to

The following command can be used to generate the encrypted password:

```
/opt/kylo/bin/encryptSqoopPassword.sh
```

The above utility will output a base64 encoded encrypted password, which can be entered directly in the controller service configuration via the **SourcePassword** and **Source Password Passphrase** properties (mode 2).

The above utility will also output a file on disk that contains the encrypted password. This can be used with mode 3 as described below:

Say, the file containing encrypted password is named: **/user/home/sec-pwd.enc**.

Put this file in HDFS and secure it by restricting permissions to be only read by **nifi** user.

Provide the file location and passphrase via the **Source Password File** and **Source Password Passphrase** properties in the **StandardSqoopConnectionService** controller service configuration.

During the processor execution, password will be decrypted for modes 2 and 3, and used for connecting to the source system.

TriggerFeed

Trigger Feed Overview

In Kylo, the TriggerFeed Processor allows feeds to be configured in such a way that a feed depending upon other feeds is automatically triggered when the dependent feed(s) complete successfully.

Obtaining the Dependent Feed Execution Context

Required field +

Property	Value
Metadata Service	Think Big Metadata Service →
Feed Precondition Event Service	JmsFeedPreconditionEventService →
System feed category	\${metadata.category.systemName}
System feed name	\${metadata.systemFeedName}
Matching Execution Context Keys	export.kylo

Comma separated list of Execution context keys or key fragments that will be applied to each of the dependent feed execution context data set. Only the execution context values starting with keys this set will be included in the flow file JSON content. Any key (case insensitive) starting with one of these supplied keys will be included

Default value: export.kylo

Supports expression language: false

CANCEL APPLY

To get dependent feed execution context data, specify the keys that you are looking for. This is done through the “Matching Execution Context Keys” property. The dependent feed execution context will only be populated the specified matching keys.

For example:

Feed_A runs and has the following attributes in the flow-file as it runs:

```
-property.name = "first name"
-property.age=23
-feeds=1478283486860
-another.property= "test"
```

Feed_B depends on Feed A and has a Trigger Feed that has “Matching Execution Context Keys” set to “property”.

It will then get the ExecutionContext for Feed A populated with 2 properties:

```
"Feed_A": {property.name: "first name", property.age: 23}
```

Trigger Feed JSON Payload

The FlowFile content of the Trigger feed includes a JSON string of the following structure:

```
{
  "feedName": "string",
  "feedId": "string",
  "dependentFeedNames": [
    "string"
  ],
  "feedJobExecutionContexts": {

  },
  "latestFeedJobExecutionContext": {

  }
}
```

JSON structure with a field description:

```
{
  "feedName": "<THE NAME OF THIS FEED>",
  "feedId": "<THE UUID OF THIS FEED>",
  "dependentFeedNames": [<array of the dependent feed names>],
  "feedJobExecutionContexts": {<dependent_feed_name>: [
    {
      "jobExecutionId": <Long ops mgr job id>,
      "startTime": <millis>,
      "endTime": <millis>,
      "executionContext": {
        <key,value> matching any of the keys defined as being "exported" in
        this trigger feed
      }
    }
  ]
},
  "latestFeedJobExecutionContext": {
    <dependent_feed_name>: {
      "jobExecutionId": <Long ops mgr job id>,
      "startTime": <millis>,
      "endTime": <millis>,
      "executionContext": {
        <key,value> matching any of the keys defined as being "exported" in
        this trigger feed
      }
    }
  }
}
```

Example JSON for a Feed:

```
{
  "feedName": "companies.check_test",
  "feedId": "b4ed909e-8e46-4bb2-965c-7788beabf20d",
  "dependentFeedNames": [
    "companies.company_data"
  ],
  "feedJobExecutionContexts": {
    "companies.company_data": [
      {
        "jobExecutionId": 21342,
        "startTime": 1478275338000,
```



```

        "endTime":1478275500000,
        "executionContext":{
        }
    }
]
},
"latestFeedJobExecutionContext":{
    "companies.company_data":{
        "jobExecutionId":21342,
        "startTime":1478275338000,
        "endTime":1478275500000,
        "executionContext":{
        }
    }
}
}
}

```

Example Flow

The screenshot shown here is an example of a flow in which the inspection of the payload triggers dependent feed data.



The EvaluateJSONPath processor is used to extract JSON content from the flow file.

Refer to the Data Confidence Invalid Records flow for an example:

Accessing S3 from the Data Wrangler

Problem

You would like to access S3 or another Hadoop-compatible filesystem from the data wrangler.

Solution

The Spark configuration needs to be updated with the path to the JARs for the filesystem.

To access S3 on HDP, the following must be added to the `spark-env.sh` file:

```
export SPARK_DIST_CLASSPATH=$(hadoop classpath)
```

Additional information is available from the Apache Spark project:

<https://spark.apache.org/docs/latest/hadoop-provided.html>

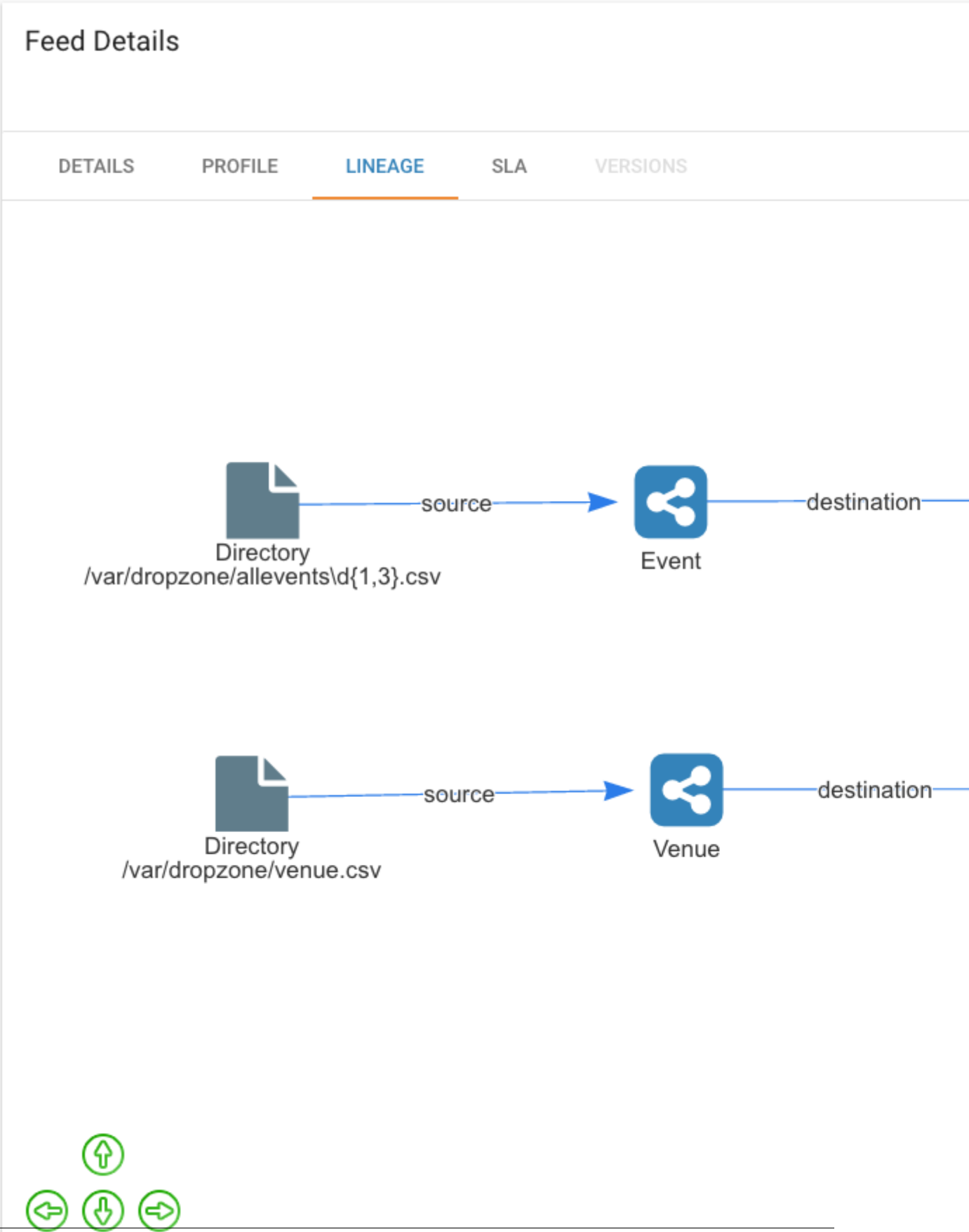
Feed Lineage Configuration

Introduction

Feeds track and display dependencies to other feeds and also their connections through their datasources.

The Lineage view on the Feed Details page is an interactive canvas that allows the user to analyze and inspect the feeds relationships.

The Designer must indicate NiFi processors that represent a source or sink to be tracked. The following guide describes how lineage is tracked and the role of designers.



Feed Connections

Connected by Preconditions

When a feed depends upon another feed(s) via a precondition (a TriggerFeed), then it will be assigned as “depends on” relationship in the Feed Lineage graph.

Connected through Datasources

Feeds are also connected through their datasources. If FeedA writes to a table and FeedB uses that same table as its source then it will be connected.

Getting Started

In order to get your feed to see its lineage you will need to do 2 things.

1. Assign the datasources to the template. See the section **Registering Datasources with a Template** below.
2. Save the Feed. Once the template has been registered you will need to save the feed. Go to the feed details. Click the Pencil icon on any section. Click **Save**.

How it works

Datasource Definitions

NiFi processors and their properties are defined as datasources in the system. These definitions are stored in the Kylo metadata repository and they can be registered 2 ways.

Registration on Startup

Kylo read the file *datasource-definitions.json* found in the classpath on startup and will update the datasource definitions. This will be in the */opt/kylo/kylo-services/conf* directory. Kylo ships with many of the NiFi processors defined, but you may find you want to alter or add new ones.

Registration via REST

If you need to update or add new datasource definitions there is a REST endpoint that allows you to post the new definition data.

POST `/v1/feedmgr/feeds/update-datasource-definitions`

Updates the datasource definitions

POST `/v1/feedmgr/feeds/update-feed-lineage-styles`

Updates the feed lineage styles

To list the metadata store of defined datasources you can use this REST call

`http://localhost:8400/proxy/v1/metadata/datasource/datasource-definitions`

Datasource Definition Structure

A datasource definition is defined with the following attributes in JSON:

```
{
  "processorType": "The Path to the NiFi processor Class Name",
  "datasourcePropertyKeys": ["Array of NiFi Property Names that identify Uniqueness"],
  "datasourceType": "A Common String identifying the Type. See the section Datasource_
↳Types below",
  "connectionType": "Either SOURCE or DESTINATION",
  "identityString": "<optional> <supports expressions> A string identifying uniqueness.
You reference any 'datasourcePropertyKey' above via expressions ${key}
(see the example GetFile below), If not defined it will use all the
↳'datasourcePropertyKeys' for its identityString",
  "description": "<optional> <supports expressions> A string describing this source",
  "title": "<optional> <supports expressions> A Title that will be displayed on the Feed_
↳Lineage page.
If not supplied it will use the 'identityString' property"
}
```

Example for the GetFile processor in NiFi:

```
{
  "processorType": "org.apache.nifi.processors.standard.GetFile",
  "datasourcePropertyKeys": ["Input Directory", "File Filter"],
  "datasourceType": "DirectoryDatasource",
  "connectionType": "SOURCE",
  "identityString": "${Input Directory}/${File Filter}",
  "description": "Directory or File source"
}
```

Datasource Types

A datasource is made unique by using its 'identityString' and its 'datasourceType'. The predefined types shipping with Kylo are:

- "HiveDatasource"
- "JMSDatasource"
- "KafkaDatasource"
- "DirectoryDatasource"
- "HDFSDatasource"
- "S3Datasource"
- "FTPDatasource"
- "HBaseDatasource"
- "HTTPDatasource"
- "DatabaseDatasource"

Refer to the datasource-definitions.json file for more details.

Registering Datasources with a Template

Templates need to be configured to identify the datasources that it should track. When registering a template that last step will show the available datasources it found in your flow. Kylo reads the template and then matches each processor with the datasource definition (see above). You will then need to select the datasources you wish to track.

This step is necessary because you may have a variety of processors in the flow that match a processor type in the datasource definition (i.e. PutFile for failed flows), but those don't define the true destination of the flow.

Feed Lineage Datasources

Select datasources that should track feed lineage

- ☒ Fetch RDBMS Data - DatabaseDatasource - SOURCE
- ☒ Filesystem - DirectoryDatasource - SOURCE
- ☒ Merge Table - HiveDatasource - DESTINATION
- ☐ Upload to HDFS - HDFSDatasource - DESTINATION
- ☐ Archive Originals - HDFSDatasource - DESTINATION
- ☐ Failed Flow - DirectoryDatasource - DESTINATION

PREVIOUS STEP

REGISTER

Styling the Feed Lineage User Interface

Feed Lineage uses a JavaScript framework [*http://visjs.org/*](http://visjs.org/) to build the interactive canvas.

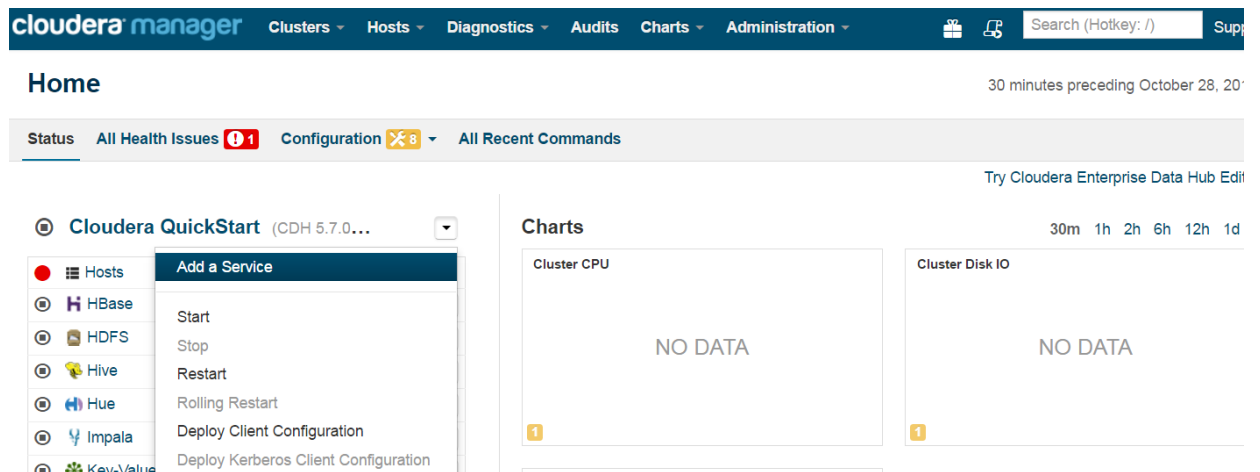
If needed you can adjust the styles of the feeds and each type of datasource. Kylo reads styles on startup from the `/opt/kylo/kylo-services/conf/datasource-styles.json`. This file can be found in `/opt/kylo/kylo-services/conf`. Styles are not stored in the metadata. They are read from this file on startup. You can alter styles using the REST endpoint below, but to persist it for the next time you will want to update this JSON file.

@TODO: image of REST ENDPOINTS

Sentry Installation Guide

Installation Steps

1. Log in to Cloudera Manager and click **Add Service**.



2. Select **Sentry** from list.

activated or the Kafka package is installed.

<input type="radio"/>	Key-Value Store Indexer	Key-Value Store Indexer listens for changes in data inside tables contained in HBase and indexes them using Solr.
<input type="radio"/>	MapReduce	Apache Hadoop MapReduce supports distributed computing on large data sets across your cluster (requires HDFS). YARN (MapReduce 2 Included) is recommended instead. MapReduce is included for backward compatibility.
<input type="radio"/>	Oozie	Oozie is a workflow coordination service to manage data processing jobs on your cluster.
<input checked="" type="radio"/>	Sentry	Sentry service stores authorization policy metadata and provides clients concurrent and secure access to this metadata.
<input type="radio"/>	Solr	Solr is a distributed service for indexing and searching data stored in HDFS.
<input type="radio"/>	Spark	Apache Spark is an open source cluster computing system. This service runs Spark as an application on YARN.
<input type="radio"/>	Spark (Standalone)	Apache Spark is an open source cluster computing system. This is the standalone version of the service which does not use YARN for resource management. Cloudera recommends using Spark on YARN instead of this standalone version.
<input type="radio"/>	Sqoop 1 Client	Configuration and connector management for Sqoop 1.
<input type="radio"/>	Sqoop 2	Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases. The version supported by Cloudera Manager is Sqoop 2 .
<input type="radio"/>	YARN (MR2 Included)	Apache Hadoop MapReduce 2.0 (MRv2), or YARN, is a data computation framework that supports MapReduce applications (requires HDFS).
<input type="radio"/>	ZooKeeper	Apache ZooKeeper is a centralized service for maintaining and synchronizing configuration data.

Back Continue

3. Click **Select hosts** to add a Sentry service.

cloudera manager Support cloudera

Add a Sentry Service to Cloudera QuickStart

Customize Role Assignments for Sentry

You can customize the role assignments for your new service here, but note that if assignments are made incorrectly, such as assigning too many roles to a single host, performance will suffer.

You can also view the role assignments by host. [View By Host](#)

ss Sentry Server × 1 New **g** Gateway

quickstart.cloudera Select hosts

Back 1 2 3 4 5 6 Continue

192.168.56.101:7180/cm/clusters/1/add-service/index?serviceType=SENTRY#

4. Provide Sentry database information and test connection.

- Database Type : mysql
- Database Name : sentry
- Username : sentry
- Password : cloudera

Add a Sentry Service to Cloudera QuickStart

Database Setup

Configure and test database connections. Create the databases first according to the [Installing and Configuring an External Database](#) section of the [Installation Guide](#).

Sentry

✓ Successful

Database Host Name: *

quickstart.cloudera

Database Type:

MySQL

Database Name: *

sentry

Username: *

sentry

Password:

☐ Show Password

Test Connection

Notes:

- The value in the **Database Host Name** field must match the value you used for the host name when creating the database. [Learn more](#)
- If the database is not running on its default port, specify the port number using **host:port** in the **Database Host Name** field.
- It is highly recommended that each database is on the same host as the corresponding role instance.

Back

1 2 3 4 5 6

Continue

5. Click **Continue** once all services are started.

Add a Sentry Service to Cloudera QuickStart

✓ First Run Command

Status: **Finished** Start Time: Oct 28, 6:55:55 AM Duration: 2m

Finished First Run of the following services successfully: Sentry.

Details [Completed 5 of 5 step\(s\).](#)

☒ All ☐ Failed Only ☐ Running Only

Step	Context	Start Time	Duration	Actions
▶ ✓ Deploy Client Configuration Successfully deployed all client configurations.	Cloudera QuickStart	Oct 28, 6:55:55 AM	15.84s	
▶ ✓ Start Successfully completed 1 steps.	ZooKeeper	Oct 28, 6:56:11 AM	30.29s	
▶ ✓ Start Successfully started HDFS service	HDFS	Oct 28, 6:56:41 AM	30.52s	
▶ ✓ Create Sentry Database Tables Successfully created Sentry database tables.	Sentry	Oct 28, 6:57:11 AM	15.38s	
▶ ✓ Start Successfully started service.	Sentry	Oct 28, 6:57:27 AM	30.29s	

Back

1 2 3 4 5 6

Continue

6. Click **Finish**.

Add a Sentry Service to Cloudera QuickStart

Congratulations!

Your new service is installed and configured on your cluster.

Note: You may still have to start your new service. It is recommended that you restart any dependency services with outdated configurations before doing so. You can perform these actions on the main page by clicking **Finish** below.

Back

1 2 3 4 5 6

Finish

Sentry is installed successfully.

SUSE Configuration Changes

Overview

The deployment guide currently addresses installation in a Red Hat Enterprise Linux (RHEL or variant, CentOS, Fedora) based environment. There are a couple of issues installing Elasticsearch and ActiveMQ on SUSE. Below are some instructions on how to install these two on SUSE.

ActiveMQ

When installing ActiveMQ you might see the following error.

Error: Configuration variable JAVA_HOME or JAVACMD is not defined correctly.
(JAVA_HOME='', JAVACMD='java')

For some reason ActiveMQ isn't properly using the system Java that is set. To fix this issue I had to set the JAVA_HOME directly.

1. Edit /etc/default/activemq and set JAVA_HOME at the bottom
2. Restart ActiveMQ (service activemq restart)

Elasticsearch

RPM installation isn't supported on SUSE. To work around this issue we created a custom init.d service script and wrote up a manual procedure to install Elasticsearch on a single node.

<https://www.elastic.co/support/matrix>

We have created a service script to make it easy to start and stop Elasticsearch, as well as leverage chkconfig to automatically start Elasticsearch when booting up the machine. Below are the instructions on how we installed Elasticsearch on a SUSE box.

1. Make sure Elasticsearch service user/group exists
2. `mkdir /opt/elasticsearch`
3. `cd /opt/elasticsearch`
4. `mv /tmp/elasticsearch-2.3.5.tar.gz`
5. `tar -xvf elasticsearch-2.3.5.tar.gz`
6. `rm elasticsearch-2.3.5.tar.gz`
7. `ln -s elasticsearch-2.3.5 current`
8. `cp elasticsearch.yml elasticsearch.yml.orig`
9. Modify `elasticsearch.yml` if you want to change the cluster name. Our copy that is installed the wizard scripts is located in `/opt/kylo/setup/elasticsearch`
10. `chown -R elasticsearch:elasticsearch /opt/elasticsearch/`
11. `vi /etc/init.d/elasticsearch` - paste in the values from `/opt/kylo/setup/elasticsearch/init.d/sles/elasticsearch`
12. Uncomment and set the java home on line 44 of the `init.d` file in step #10
13. `chmod 755 /etc/init.d/elasticsearch`
14. `chkconfig elasticsearch on`
15. `service elasticsearch start`

Configuration Properties

Overview

This guide provides details on how to configure Kylo Templates and Feeds with properties from different sources. The sources can be the following:

1. Configuration from `application.properties`
2. Configuration from Feed Metadata
3. Configuration from Nifi environment variables

There are two property resolution options:

1. Design-time resolution
2. Runtime resolution

1. Configuration Sources

1.1 Configuration from `application.properties`

When creating Kylo feeds and templates one can refer to configuration properties which appear in `/opt/kylo/kylo-services/conf/application.properties` file. Property names must begin with word `config.` and they should be referenced by following notation in Kylo UI `${config.<property-name>}`

Here is an example of how we use this in `application.properties`

```
config.hive.schema=hive
config.props.max-file-size=3 MB
```

Here is how you would refer to `config.props.max-file-size` in Kylo template:

Additional Properties	
<input type="checkbox"/>	Maximum File Age
<input checked="" type="checkbox"/>	Maximum File Size <small>Default Value (Supports Expressions)</small> <code>\${config.props.max-file-size}</code> 3 MB <div> <input checked="" type="checkbox"/> Allow user input? <div>Render as</div> <div>Text</div> </div>
<input type="checkbox"/>	Minimum File Age
<input type="checkbox"/>	Minimum File Size

Setting NiFi Processor Properties

There is a special property naming convention available for NiFi Processors and Services in application. properties too.

For Processor properties four notations are available:

1. `nifi.<processor_type>.<property_key>`
2. `nifi.all_processors.<property_key>`
3. `nifi.<processor_type>[<processor_name>].<property_key>` (Available in Kylo 0.8.1)
4. `$nifi{nifi.property}` will inject the NiFi property expression into the value. (Available in Kylo 0.8.1)

where `<processor_type>`, `<property_key>`, `<processor_name>` should be all lowercase with spaces replaced by underscores. The `<processor_name>` is the display name of the processor set in NiFi. Starting in Kylo 0.8.1 you can inject a property that has NiFi Expression Language as the value. Since Spring and NiFi EL use the same notation (`${property}`) Kylo will detect any nifi expression in the property value if it start with `$nifi{property}`

- Setting properties matching the NiFi Processor Type. Here is an example of how to set ‘Spark Home’ and ‘Driver Memory’ properties on all ‘Execute Spark Job’ Processors:

```
nifi.executesparkjob.sparkhome=/usr/hdp/current/spark-client
nifi.executesparkjob.driver_memory=1024m
```

- Setting properties for a named NiFi Processor (starting in Kylo 0.8.1). Here is an example setting the property for just the ExecuteSparkJob processor named “Validate and Split Records”:

```
nifi.executesparkjob[validate_and_split_records].number_of_executors=3
nifi.executesparkjob[validate_and_split_records].driver_memory=1024m
```

- Setting a property with NiFi expression language as a value (starting in Kylo 0.8.1). Here is an example of injecting a value which refers to a NiFi expression

```
nifi.updateattributes[my_processor].my_property=/path/to/$nifi{my.nifi.
↪expression.property}
```

The “my property” on the UpdateAttribute processor named “My Processor” will get resolved to `/path/to/${my.nifi.expression.property}` in NiFi.

- Setting all properties matching the property key. Here is an example of how to set Kerberos configuration for all processors which support it:

```
nifi.all_processors.kerberos_principal=nifi
nifi.all_processors.kerberos_keytab=/etc/security/keytabs/nifi.headless.
↪keytab
```

Setting Controller Service Properties

For Services use following notation: `nifi.service.<service_name>.<property_name>`. Anything prefixed with `nifi.service` will be used by the UI. Replace spaces in Service and Property names with underscores and make it lowercase. Here is an example of how to set 'Database User' and 'Password' properties for MySQL Service:

```
nifi.service.mysql.database_user=root
nifi.service.mysql.password=hadoop
```

1.2 Configuration from Feed Metadata

When creating Kylo feeds and templates you can also refer to Feed Metadata, i.e. set property values based on known information about the feed itself. These properties start with word 'metadata', e.g. `${metadata.<property-name>}`

Here is how you would refer to Category name and Feed name in Kylo template:

Additional Properties	
<input type="checkbox"/>	Ignore Hidden Files
<input checked="" type="checkbox"/>	Input Directory <small>Default Value (Supports Expressions)</small> <code>/var/\${metadata.category.systemName}/\${metadata.systemFeedName}</code> <small>Render as</small> <input checked="" type="checkbox"/> Allow user input? <small>Text</small> ▼
<input type="checkbox"/>	Keep Source File

1.3 Configuration from Nifi environment variables

TODO - Help us complete this section

2. Property Resolution Options

2.1 Design-time Resolution


These properties will be resolved at design-time during Feed creation from Template. They use the following notation `${property-name}`. If you had `property-name=value` in `application.properties` and `${property-name}` in Template then static value would be placed into Processor field in Nifi on Feed creation.

You can also provide nested properties or properties which refer to other properties `${property-name2.${property-name1}}`. If you had `property-name1=value1` and `property-name2.value1=value2` in `application.properties` and `${property-name1.${property-name2}}` in Template then static value2 would be placed into Processor field in Nifi on Feed creation.

Note: This type of resolution is great for properties which do not support Nifi's Expression Language.

2.2 Runtime or Partial Resolution

If you don't want to resolve properties at design time and would rather take advantage of property resolution at runtime by Nifi's Expression Language then you can still refer to properties in Kylo Feeds and Template, just escape them with a dollar sign \$ like so: `$$${config.${metadata.feedName}.input-dir}`. Notice the double dollar sign at the start. This property will be resolved at design-time to `${config.<feed-name>.input-dir}` and will be substituted at runtime with a value from `application.properties` file. So if you had a feed called `users` and `config.users.input-dir=/var/dropzone/users` in `application.properties` then at runtime the feed would take its data from `/var/dropzone/users` directory.

 **Feed Details**

Choose a Feed Input

GetFile

Input Directory

`$$${config.${metadata.feedName}.input-dir}`

Note: This type of resolution is great for creating separate configurations for multiple feeds created from the same template

Setting RDD Persistence Level

The Validator allows specifying the RDD persistence level via command line argument.

To use this feature in the standard ingest flow, perform these steps:

1. In NiFi, navigate to 'reusable_templates -> standard_ingest'.
2. Stop 'Validate And Split Records' processor.
3. Open configuration for 'Validate And Split Records' processor. Add two arguments at the end for the *MainArgs* property.

```
<existing_args>, --storageLevel, <your_value>
```

<your_value> can be any valid Spark persistence level (e.g. MEMORY_ONLY, MEMORY_ONLY_
→SER)

4. Start 'Validate And Split Records' processor.

Note: If not specified, the default persistence level used is MEMORY_AND_DISK.

Specifying Number of RDD Partitions

The Validator allows specifying the number of RDD partitions via command line argument. This can be useful for processing large files.

To use this feature in the standard ingest flow, perform these steps:

1. In NiFi, navigate to 'reusable_templates -> standard_ingest'.
2. Stop 'Validate And Split Records' processor.

3. Open configuration for 'Validate And Split Records' processor. Add two arguments at the end for the *MainArgs* property.

```
<existing_args>,--numPartitions,<your_value>  
<your_value> should be positive integer.
```

4. Start 'Validate And Split Records' processor.

Note: If not specified, Spark will automatically decide the partitioning level.

Developer Getting Started Guide

This guide should help you get your local development environment up and running quickly. Development in an IDE is usually done in conjunction with a Hortonworks sandbox in order to have a cluster with which to communicate.

Dependencies

To run the Kylo project locally the following tools must be installed:

- Maven 3
- RPM (for install)
- Java 1.8 (or greater)
- Hadoop 2.3+ Sandbox
- Virtual Box or other virtual machine manager

The assumption is that you are installing on a Mac or Linux box. You can do most activities below on a Windows box, except to perform a Maven build with the RPM install. At some point, we could add a Maven profile to allow you to build but skip the final RPM step.

Install Maven 3

This project requires Maven to execute a build. Use this link to download to the Maven installation file:

Note: For instructions on installing Apache Maven see the [Installing Apache Maven](#) docs at the Apache Maven project site.

Optional - Add Java 8 to Bash Profile

To build from the command line, you need to add Java 8 and Maven to your \$PATH variable.

Edit ~/.bashrc and add the following:

```
export MVN_HOME=/Users/<HomeFolderName>/tools/apache-maven-3.3.3
export MAVEN_OPTS="-Xms256m -Xmx512m"
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home
export PATH=$JAVA_HOME/bin:$MVN_HOME/bin:$PATH
```

To test, run the following:

```
$ mvn -v
$ java -version
```

Install Virtual Box

Use this link to download and install the DMG file to install Virtual Box:

[*https://www.virtualbox.org/wiki/Downloads*](https://www.virtualbox.org/wiki/Downloads)

Install the RPM Tool on your Mac

The RPM library is required for building the RPM file as part of the Maven build. This can be done using Home Brew or Mac Ports.

```
$ brew install rpm
```

Clone Project from Github

Clone the Kylo project to your host. You can do this in your IDE or from the command line.

1. From the command line, run the “git clone” command.
 - (a) cd to the directory you want to install the project to.
 - (b) Type “git clone [*https://github.com/kyloio/kylo.git*](https://github.com/kyloio/kylo.git)”.
2. Import from your IDE using the “[*https://github.com/kyloio/kylo.git*](https://github.com/kyloio/kylo.git)” URL.

Import the Project into your IDE

Import the project into your favorite IDE as a Maven project.

Note: Configure the project to use Java 8.

Perform a Maven Build

Perform a Maven build to download all of the artifacts and verify that everything is setup correctly.

```
$ mvn clean install
```

Note: If you receive an `OutOfMemoryError` try increasing the Java heap space: `$ export MAVEN_OPTS="-Xms2g -Xmx4g"`

Tip: For faster Maven builds you can run in offline mode and skip unit testing: `$ mvn clean install -o -DskipTests`

Install and Configure the Hortonworks Sandbox

Follow the guide below to install and configure the Hortonworks sandbox:

[Hortonworks Sandbox Configuration](#)

Install the Kylo Applications

To install the Kylo apps, NiFi, ActiveMQ, and Elasticsearch in the VM you can use the deployment wizard instructions found here:

[Setup Wizard Deployment Guide](#)

Instead of downloading the RPM file copy the RPM file from your project folder after running a Maven build.

```
$ cd /opt
$ cp /media/sf_kylo/install/target/rpm/kylo/RPMS/noarch/kylo-<version>.noarch.rpm.
$ rpm -ivh kylo-<version>.noarch.rpm
```

Follow the rest of the deployment wizard steps to install the rest of the tools in the VM.

You now have a distribution of the stack running in your Hortonworks sandbox.

Running in the IDE

You can run kylo-ui and thinkbig-services in the IDE. If you plan to run the apps in the IDE, you should shut down the services in your sandbox so you aren't running two instances at the same time.

```
$ service kylo-services stop
$ service kylo-ui stop
```

The applications are configured using Spring Boot.

IntelliJ Configuration

1. Install the Spring Boot plugin.
2. Create the kylo-services application run configuration.
 - (a) Open the Run configurations.
 - (b) Create a new Spring Boot run configuration.
 - (c) Give it a name like “KyloServerApplication”.
 - (d) Set “use classpath of module” property to “kylo-service-app” module.
 - (e) Set the “Main Class” property to “com.thinkbiganalytics.server.KyloServerApplication”.
3. Create the kylo-ui application run configuration.
 - (a) Open the Run configurations.
 - (b) Create a new Spring Boot run configuration.
 - (c) Give it a name like “KyloDataLakeUiApplication”.
 - (d) Set “use classpath of module” property to “kylo-ui-app” module.
 - (e) Set the “Main Class” property to “com.thinkbiganalytics.KyloUiApplication”.
4. Run both applications.

Eclipse Configuration

1. Open Eclipse.
2. Import the Kylo project.
 - (a) File - Import
 - (b) Choose “maven” and “Existing Maven Projects” then choose next
 - (c) Choose the Kylo root folder. You should see all Maven modules checked
 - (d) Click finish
 - (e) Import takes a bit - if you get an error about scala plugin, just click finish to ignore it.
3. Find and open the “com.thinkbiganalytics.server.KyloServerApplication” class.
4. Right click and choose to debug as a Java application.
5. Repeat for “com.thinkbiganalytics.KyloUiApplication”.

OPTIONAL: Install the spring tools suite and run as a spring boot option

Note: Consult the Spring Boot documentation for [Running Your Application](#) for additional ways to run with spring boot.

Introduction

We gladly welcome contributions to help make Kylo better! This document describes our process for accepting contributions and the guidelines we adhere to as a team. Please take a moment to review before submitting a pull request.

Why Contribute

Think Big originally developed Kylo based on experience gained on over 150 big data projects. Many of the best improvements came from exercising the technology in the field on the huge variety of situations faced by customers. Contributing to Kylo allows you to influence the roadmap and evolution of Kylo and contribute back to the community at large.

Reporting Issues

We monitor [Group Groups](#) for questions. If you're not sure about something then please search on [Group Groups](#) first and ask a new question if necessary. Bug reports, feature requests, and pull requests can be submitted to our [JIRA](#) for tracking. If you find an issue:

1. Search in [JIRA](#) to see if the issue has already been reported. You can add to the existing discussion or see if someone else is already working on it.
2. If the issue has been fixed then try reproducing the issue using the latest *master* branch.
3. If the issue persists then try to isolate the cause and create a new [JIRA](#).
 - For bug reports, please include a description of the issue, the steps to reproduce, the expected results, and the actual results.
 - For feature requests, please give as much detail as possible including a design document if available.

Introducing New Functionality

Before contributing new functionality or bug fixes please consider how these changes may impact other people using Kylo, and whether these changes can be considered overall enhancements or merely enhancements needed by your particular project. New functionality can be introduced either as a plugin or through a pull request.

Plugins

Plugins are the preferred way of adding, swapping, or enhancing functionality that is only relevant to specific users. Our components and services have well-defined interfaces that can be extended by adding a new JAR to the *plugin* directory. Create a new Spring `@Configuration` class to add your classes to the Spring context.

A separate git repository should be used for your plugins. You can reference Kylo's API artifacts in Maven.

Pull Requests

Changes that apply to every Kylo user should be submitted as a pull request in GitHub. You should do your work in a fork of Kylo and submit a request to pull in those changes. Don't forget to confirm the target branch (master or point release) before submitting the request. Please continue reading for instructions on creating a pull request.

Development Guidelines

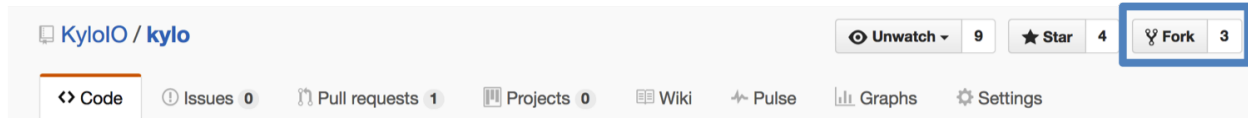
We adhere to the following guidelines to ensure consistency in our code:

- Source code should be formatted according to our IntelliJ or Eclipse formatter. Formatter markers in comments are enabled but should be used sparingly.
 - To import our standard IntelliJ formatter:
 - Download the template from here: `thinkbig-googlestyle-intellij-v2-1.xml`.
 - Preferences -> Editor -> Code Style -> Manage
 - Select “Import” and choose the downloaded preferences file
 - Make sure the “scheme” shows `thinkbig-googlestyle-intellij-vX.Y`
 - To import our standard Eclipse formatter:
 - Download the template from here: `thinkbig-googlestyle-eclipse-v2-1.xml`.
 - Preferences -> Java -> Code Style -> Formatter
 - Select “Import” and choose the downloaded preferences file
 - Make sure the “Active Profile” shows `thinkbig-googlestyle-eclipse-v2-1.xml`
- Public API methods should be documented. Use Swagger annotations for REST endpoints.
- Ensure tests are passing for the modified classes. New tests should use JUnit and Mockito.
- Prefer to throw runtime exceptions instead of checked exceptions.
- Dependency versions should be declared in the root pom and can be overridden using pom properties.
- Module names should be in all lowercase. Words should be singular and separated by a hyphen. For example, `kylo-alert` is preferred over `kylo-alerts`.
- Logging should use SLF4j:

```
private static final Logger log = LoggerFactory.getLogger(MyClass.class);
```

Pull Requests

To get started go to GitHub and fork the [Kylo](#) repository.



This will create a copy of the repository under your personal GitHub account. You will have write permissions to your repository but not to the official Kylo repository.

Before you start

The easiest way to contribute code is to create a separate branch for every feature or bug fix. This will allow you to make separate pull requests for every contribution. You can create your branch off our *master* branch to get the latest code, or off a *release* branch if you need more stable code.

```
git clone https://github.com/<your-username>/kylo.git
cd kylo
git checkout -b my-fix-branch master
```

Every change you commit should refer to a [JIRA issue](#) that describes the feature or bug. Please open a JIRA issue if one does not already exist.

Committing your change

Ensure that your code has sufficient unit tests and that all unit tests pass.

Your commit message should reference the JIRA issue and include a sentence describing what was changed. An example of a good commit message is “PC-826 Support for schema discovery of Parquet files.”

```
git commit -a -m "<my-commit-message>"
git push origin my-fix-branch
```

Submitting a pull request

Once you are ready to have us add your changes to the Kylo repository, go to your repository in GitHub and select the branch with your changes. Then click the *New pull request* button.



GitHub will generate a diff for your changes and determine if they can be merged back into Kylo. If your changes cannot be automatically merged, please try rebasing your changes against the latest *master* branch.

```
git fetch --all
git rebase origin/master
git push --force-with-lease origin my-fix-branch
```

We will review your code and respond with any necessary changes before pulling in your changes. After your pull request is merged you can safely delete your branch and pull in the changes from the official Kylo repository.

CHAPTER 26

Kylo REST API

Kylo uses swagger to document its REST API.

When running Kylo, you can access the documentation at <http://localhost:8400/api-docs/index.html>.

A sample PDF `Kylo REST API Sample` shows you some of the operations Kylo exposes..

Purpose

This guide provides instructions for operating and maintaining the Kylo solution. The information is used by the Operations and Support Team in the deployment, installation, updating, monitoring and support of Kylo.

Scope

This guide is not a step-by-step process for the Operations Team, but a set of examples that we have assembled from our previous experiences.

Audience

This guide assumes its user to be knowledgeable in IT terms and skills. As an operations and maintenance (O&M) runbook, it describes the information necessary to effectively manage:

- Production processing
- Ongoing maintenance
- Performance monitoring

This document specifically serves to guide those who will be maintaining, supporting, and using the Kylo solution in day-to-day operational basis.

Abbreviations

Abbreviations/Key term	Definition
O&M	Operations and Maintenance
CLI	Command Line Interface
ES	ElasticSearch

Introduction

Kylo is a software application that provides scheduling, monitoring, and control for data processing jobs. Kylo includes its own web-based interface intended for an Operations user to visualize status of processing and assist with troubleshooting problems.

Please note, this Operations Guide is provided in its entirety, despite the fact that not all features may be utilized within a particular solution.

Common Definitions

The following terms are used in this document or are relevant to understanding the nature of Kylo processing.

Term	Definition
Job	A Job consists of a sequence of processing tasks called <i>steps</i> . A Job has both status and state that indicate its outcome.
Feed	A feed is a pipeline, jobs are run for feeds. The “health” status of a feed (regardless of its running state) can be visualized on the Kylo Overview page.
Check Data Job	An optional job type employed for independent data quality checks against customer data with results contributing to a “Data Confidence” metric visible on the Overview page.
Step	A unit of processing in a job sequence. A job consists of one or more steps. Each step also has both status and state, similar to that of a job. Steps may capture meta-data, stored in Postgres and viewable in the application.
Job Instance Id	The Job Instance and its corresponding Job Instance Id refer to a logical Job run (i.e. A Job with a set of Job Parameters). A Job Instance can have multiple Job Executions, but only one successful Job Execution.
Job Execution Id	The Job Execution and corresponding Job Execution Id refer to a single attempt to run a Job Instance. A Job Instance can have multiple Job Executions if some fail and are restarted.

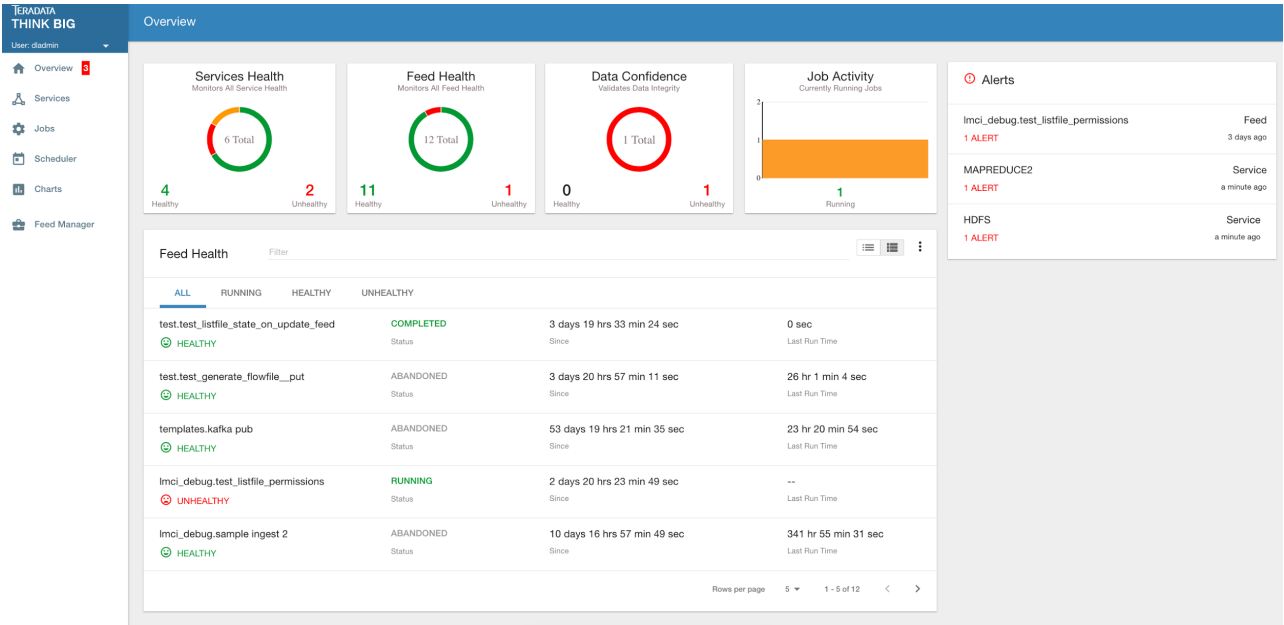
User Interface

Kylo has a web-based user interface designed for an Operations user to monitor and managing data processing. The default URL is `http://<hostname>:8400/`, however the port may be configured via the application.properties.

The following sections describe characteristics of the user interface.

Overview Page

The Overview tab performs the role of an Operations Dashboard. Content in the page automatically refreshes showing real-time health and statistics about data feeds and job status.



Kylo Overview Page

Key Performance Indicators

The Overview page has multiple indicators that help you quickly assess the health of the system:

 <p>Services Health Monitors All Service Health</p> <p>6 Total</p> <p>4 Healthy</p> <p>2 Unhealthy</p>	Provides a health status of external dependencies such as MySQL or Postgres, Hadoop services.
 <p>Feed Health Monitors All Feed Health</p> <p>12 Total</p> <p>11 Healthy</p> <p>1 Unhealthy</p>	Provides a summary health status of all data feeds. Details of these feeds are shown in a table, Feed Summary, also on the Overview Page
 <p>Data Confidence Validates Data Integrity</p> <p>1 Total</p> <p>0 Healthy</p> <p>1 Unhealthy</p>	Optional. Displays a confidence metric updated by any Data Quality Check jobs.
 <p>Job Activity Currently Running Jobs</p> <p>2</p> <p>1</p> <p>0</p> <p>1 Running</p>	Displays all running jobs.
 <p>Alerts</p> <p>Imcl_debug.test_listfile_permissions 1 ALERT</p> <p>MAPREDUCE2 1 ALERT</p> <p>HDFS 1 ALERT</p> <p>Feed 3 days ago</p> <p>Service a minute ago</p> <p>Service a minute ago</p>	Displays alerts for services and feeds. Click on them for more information.

Feed Summary

The Feed Summary Table provides the state and status of each data feed managed by Kylo. The state is either HEALTHY or UNHEALTHY. The status is the status of the most recent job of the feed. You can drill into a specific feed and see its *history* by clicking on the name of the feed in the table.

Feed Health Filter				
ALL	RUNNING	HEALTHY	UNHEALTHY	
test.test_listfile_state_on_update_feed	COMPLETED	3 days 19 hrs 34 min 56 sec	0 sec	
test.test_generate_flowfile__put	ABANDONED	3 days 20 hrs 58 min 43 sec	26 hr 1 min 4 sec	
templates.kafka pub	ABANDONED	53 days 19 hrs 23 min 7 sec	23 hr 20 min 54 sec	
Imci_debug.test_listfile_permissions	RUNNING	2 days 20 hrs 25 min 21 sec	--	
Imci_debug.sample ingest 2	ABANDONED	10 days 16 hrs 59 min 21 sec	341 hr 55 min 31 sec	
				Rows per page 5 1 - 5 of 12 < >

Active Jobs

The Active Jobs table shows currently running jobs as well as any failed jobs that require user attention. The table displays all jobs. A user may drill-in to view **Job Details** by clicking on the corresponding Job Name cell. Jobs can be controlled via action buttons. Refer to the **Controlling Jobs** section to see the different actions that can be performed for a Job.

Understanding Job Status

Jobs have two properties that indicate their status and state, Job Status and Exit Code respectively.

Job Status

The Job Status is the final outcome of a Job.

- COMPLETED – The Job finished.
- FAILED – The Job failed to finish.
- STARTED – The Job is currently running.
- ABANDONED – The Job was abandoned.

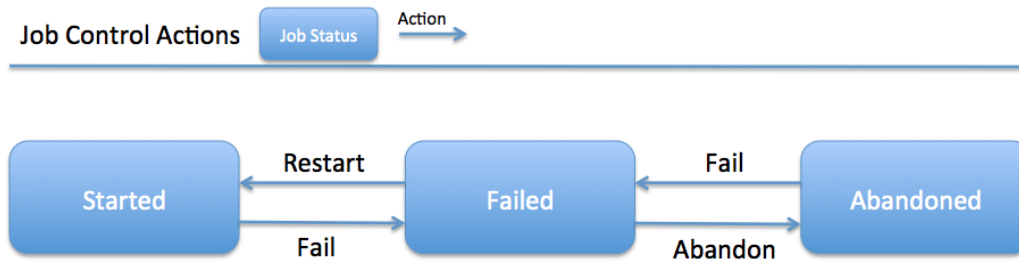
Job Exit Codes

The Exit Code is the state of the Job.

- COMPLETED – The Job Finished Processing
- EXECUTING - The Job is currently in a processing state
- FAILED – The Job finished with an error
- ABANDONED – The Job was manually abandoned

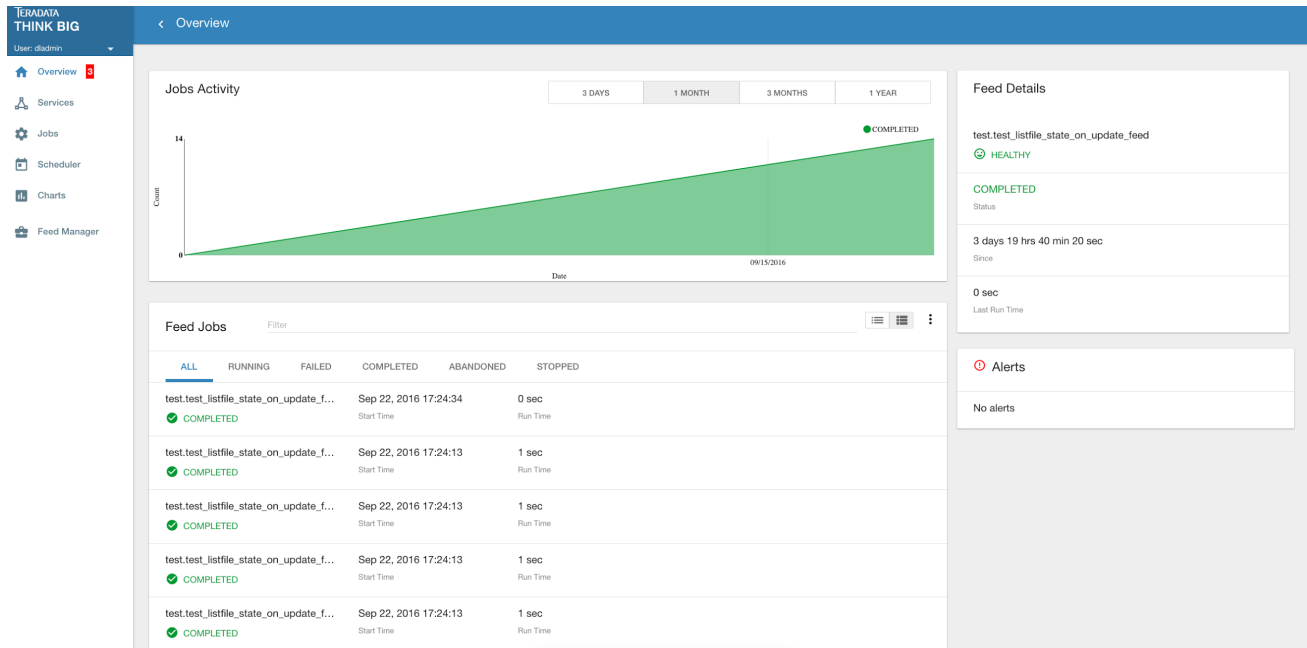
Controlling Jobs

The image below illustrates the different *actions* that can be performed based on its Job Status:



Feed History Page

Kylo stores history of each time a feed is executed. You can access this data by clicking on the specific feed name in the Feed Summary table on the Overview page. Initially the Feeds table provides high-level data about the feed.



You can get more data by clicking on a job in the Feed Jobs table. This will go into the Job Details page for that job.

Job History Page

Job history can be accessed in the Jobs Tab.

TERADATA THINK BIG

User: dladmin

Overview 2

Services

Jobs

Scheduler

Charts

Feed Manager

The Job History page provides a searchable table displaying job information, seen below. You can click on the Job Name to view the **Job Details** for the selected Job.

Jobs

Filter

ALL






RUNNING

FAILED

COMPLETED

ABANDONED

STOPPED

demo.kafka_pub_demo_feed_2	demo.kafka_pub_demo_feed_2	Sep 27, 2016 12:32:19	104 hrs 52 min 19 sec	STOP	FAIL
 STARTED	Feed	Start Time	Run Time		
Imci_debug.test_listfile_permissions	Imci_debug.test_listfile_permi...	Sep 27, 2016 10:22:25	107 hrs 2 min 13 sec	STOP	FAIL
 STARTED	Feed	Start Time	Run Time		
Imci_debug.test_listfile_permissions	Imci_debug.test_listfile_permi...	Sep 27, 2016 10:22:25	107 hrs 2 min 13 sec	STOP	FAIL
 STARTED	Feed	Start Time	Run Time		
Imci_debug.test_listfile_permissions	Imci_debug.test_listfile_permi...	Sep 27, 2016 10:22:25	107 hrs 2 min 13 sec	STOP	FAIL
 STARTED	Feed	Start Time	Run Time		
Imci_debug.test_listfile_permissions	Imci_debug.test_listfile_permi...	Sep 27, 2016 10:22:25	107 hrs 2 min 13 sec	STOP	FAIL
 STARTED	Feed	Start Time	Run Time		

Rows per page

5

1 - 5 of 1317

<

>

Job Detail Drill-Down

Clicking on the Job Name from either the Jobs Tab or Feeds Tab accesses the Job Details. It shows all information about a job including any metadata captured during the Job run.

The detail page is best source for troubleshooting unexpected behavior of an individual job.

Job Execution

Job 1 of 1

JOBSTEP 1STEP 2STEP 3

test.test_listfile_state_on_update_feed

✓ COMPLETED

Exit Description

No description available.

Sep 22, 2016 17:24:34

Start Time

0 sec

Run Time

COMPLETED

Exit Code

JOB PARAMETERS

EXECUTION CONTEXT DATA

Parameters

Values

file.group

thinkbig

file.lastModifiedTime

2016-09-20T10:27:08-0400

file.size

413376

file.permissions

rw-r--r--

uuid

076560cb-b322-446e-aa47-eff91ae03301

absolute.path

/tmp/

path

./

Job Details

test.test_listfile_state_on_update_feed

✓ COMPLETED

FEED

Type

Sep 22, 2016 17:24:34

Start Time

0 sec

Run Time

COMPLETED

Exit Code

Related Jobs

Job

1. Sep 22, 2016 17:24:34

Job Status Info

Job Status information such as start and run time, along with any control actions, are displayed on the right.

Job Details

RESTARTABANDON

Imci_debug.test_listfile_permissions

⊘ FAILED

FEED

Type

Sep 27, 2016 10:22:24

Start Time

24 hrs 56 min 56 sec

Run Time

EXECUTING

Exit Code

Related Jobs

Job

1. Sep 27, 2016 10:22:24

Job Parameters

A Job has a set of parameters that are used as inputs into that job. The top section of the Job Details page displays these

		JOB PARAMETERS	EXECUTION CONTEXT DATA
Parameters	Values		
file.group	thinkbig		
file.lastModifiedTime	2016-09-20T10:27:08-0400		
file.size	413376		
file.permissions	rw-r--r--		
uuid	076560cb-b322-446e-aa47-eff91ae03301		
absolute.path	/tmp/		
path	./		
feed	test.test_listfile_state_on_update_feed		
filename	import_template_14743816284854385705068247225005.xml		
file.creationTime	2016-09-20T10:27:08-0400		
file.lastAccessTime	2016-09-20T10:27:08-0400		
file.owner	thinkbig		
feedIsParent	true		
jobType	FEED		

parameters.

Job Context Data

As a Job runs operational metadata is captured and step status is visible in the Job page.

This metadata is stored in the Job Context section. Access this section by clicking on the **Execution Context Data** button next to the Job Parameters button in the previous figure.

Step Context Data

A job can have multiple steps, each of which capture and store metadata as it relates to that step.

Job Execution				Job 1 of 1
JOB	STEP 1	STEP 2	STEP 3	
ListFile		Sep 22, 2016 17:24:34	0 sec	COMPLETED
✓ COMPLETED		Start Time	Run Time	Exit Code
Exit Description No description available.				
Context Parameters		Values		
file.group		thinkbig		
file.lastModifiedTime		2016-09-20T10:27:08-0400		
file.size		413376		
Event Id		2251		
file.permissions		rw-r--r--		
uuid		076560cb-b322-446e-aa47-eff91ae03301		
absolute.path		/tmp/		
Flow File Id		076560cb-b322-446e-aa47-eff91ae03301		
path		./		

Scheduler Page

The scheduling of SLAs can be viewed and via the “Scheduler” tab.

This allows a user to pause the entire Scheduler, pause specific SLAs, and even manually trigger SLAs to execute.

Scheduler

Scheduled Jobs

Completion Time

SLA

in a few seconds

Cron Expression

PAUSE

FIRE NOW

🕒 SCHEDULED

Group

Next Fire

Scheduler Details

PAUSE SCHEDULER

✓ RUNNING

Status

149 hrs 54 min 38 sec

Up Time

09/26/2016 04:23:18 pm

Start Time

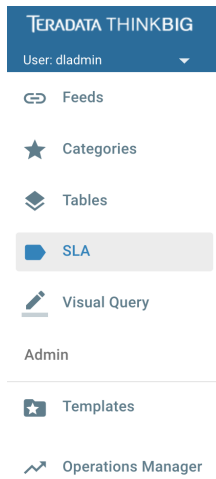
284

Jobs Executed

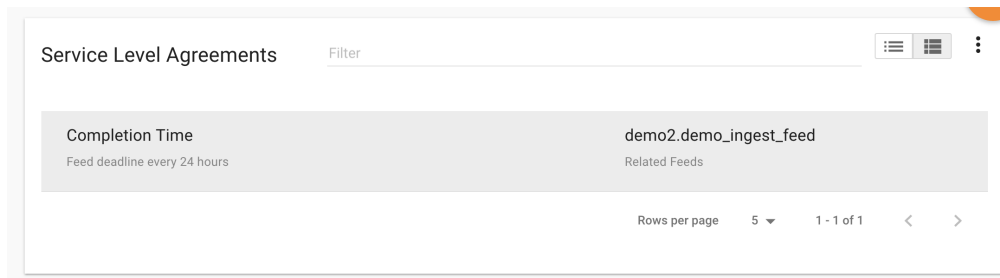
Changing an SLA

To change the schedule of a given SLA :

1. Click on the SLA tab in the Feed Manager site.



2. Select the SLA whose schedule you would like to change.



3. Edit the configurations and click Save SLA.

SLA Conditions

1. Feed Processing deadline

Ensure a Feed processes data by a specified time

FeedName

demo2.Demo ingest feed

Expected Delivery Time

0 0 12 1/1 * ? *

Cron Expression for when you expect to receive this data

Cron Preview

10/03/2016 12:00:00 PM

10/04/2016 12:00:00 PM

10/05/2016 12:00:00 PM

No later than time

2

Units

Hours

Number specifying the amount of time allowed after the Expected Delivery Time

ADD CONDITION

SLA Actions

1. Email

Email user(s) when the SLA is violated

Warning Configuration Error! You can still assign this action, but it may not fire due to configuration issues. Email connection information is not setup. Please contact an administrator to set this up.

Email addresses

admin@lmci.com

comma separated email addresses

ADD ACTION

DELETE

CANCEL

SAVE SLA

Filtering Job History

The following section describes how to filter the job and feed history tables. Kylo provides a dynamic filter capability for any table displaying multiple rows of information.

Data Table Operations

Sorting Content

All tables allow for the columns to be sorted. An arrow will appear next to the column indicating the sort direction. Click on the column header to change the sort.

Filtering Tables


All Tables in Kylo have a Filter bar above them. The rows can be filtered using the search bar at the top.

Feed Health

Filter
demo

ALL	RUNNING	HEALTHY	UNHEALTHY
demo2.demo_ingest_feed HEALTHY	ABANDONED Status	17 days 15 hrs 55 min 47 sec Since	893 hr 34 min 12 sec Last Run Time
demo.kafka_pub_demo_feed_2 HEALTHY	RUNNING Status	5 days 23 hrs 24 min 13 sec Since	-- Last Run Time
demo.kafka_pub_demo_feed_1 HEALTHY	COMPLETED Status	61 days 18 hrs 32 min 13 sec Since	0 sec Last Run Time

Rows per page: 5 1 - 3 of 3 < >

Clicking on the  icon in the top right of the table will display the table so that you can sort by column.

Filter

demo




Feed Health

ALL

RUNNING

HEALTHY

UNHEALTHY

Feed ↓	Health	Status	Since	Last Run Time
demo2.demo_ingest_feed	 HEALTHY	ABANDONED	17 days 15 hrs 57 min 24 sec	893 hr 34 min 12 sec
demo.kafka_pub_demo_feed_2	 HEALTHY	RUNNING	5 days 23 hrs 25 min 50 sec	--
demo.kafka_pub_demo_feed_1	 HEALTHY	COMPLETED	61 days 18 hrs 33 min 50 sec	0 sec


Rows per page:

5 ▼

1 - 5 of 12

<

>

Click on any of the column headers, or click on the  icon in the top right of the table, to sort.


Charts and Pivot Tables


The Charts tab allows you to query and perform data analysis on the Jobs in the system. The right panel allows you to provide filter input that will drive the bottom Pivot Chart panel.

Filter Chart

Showing 1 jobs

Feed
data sources.GetFile source ▼

 Start date ▼

 End date ▼

Limit
500 ▼

Update

The Pivot Charts panel is a rich drag and drop section that allows you to create custom tables and charts by dragging attributes around. The drop down at the top left allows you to choose how you want to display the data

Chart Type

✓ Table

Table Barchart

Heatmap

Row Heatmap

Col Heatmap

Line Chart

Bar Chart

Stacked Bar Chart

Area Chart

Scatter Chart

The data attributes at the top can be dragged into either Column Header or Row level attributes for the rendered pivot.

Pivot Charts

Chart Type: Table

Aggregator: Count

Attributes (drag and drop to customize the chart)

Job Name Exit Code

Start Date

Table:

Job Name	Exit Code	Start Date	2016-09-01	Totals
Imci_debug.ingest 3	COMPLETED		1	1
	EXECUTING		1	1
Totals			2	2

Clicking the down arrow on each attribute allows you to filter out certain fields.

Aggregator: Count

Start Date

Job Name

Exit Code

Exit Code (2)

Select All Select None

Filter results

☒ COMPLETED (15)

☒ EXECUTING (1)

OK

This interface allows you to filter the job data and create many different combinations of tables and charts.

Software Components

The following provides a basic overview of the components and dependencies for Kylo:

- Web-based UI (tested with Safari, Firefox, Chrome)
- Embedded Tomcat web container (configurable HTTP port)
- Java 8
- Stores job history and metadata in Postgres or MySQL
- NiFi 1.x+
- ActiveMQ

- Elasticsearch (optional, but required for full feature set)

Installation

Please refer to the installation guide for Kylo installation procedures.

Application Configuration

Configuration files for Kylo are located at:

```
/opt/kylo/kylo-services/conf/application.properties
/opt/kylo/kylo-ui/conf/application.properties
```

Application Properties

The *application.properties* file in *kylo-services* specifies most of the standard configuration in pipeline.

Note: Any change to the application properties will require an application restart.

Below is a sample properties file with Spring Datasource properties for spring batch and the default data source:

Note: Cloudera default password for root access to mysql is “cloudera”.

```
spring.datasource.url=jdbc:mysql://localhost:3306/kylo
spring.datasource.username=root
spring.datasource.password=
spring.datasource.maxActive=10
spring.datasource.validationQuery=SELECT 1
spring.datasource.testOnBorrow=true
spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.open-in-view=true
#
#Postgres datasource configuration
#
#spring.datasource.url=jdbc:postgresql://localhost:5432/pipeline_db
#spring.datasource.driverClassName=org.postgresql.Driver
#spring.datasource.username=root
#spring.datasource.password=thinkbig
#spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
###
# Current available authentication/authorization profiles:
# * auth-simple - Uses authenticationService.username and authenticationService.
#                  ↳password for authentication (development only)
# * auth-file - Uses users.properties and roles.properties for authentication and
#                  ↳role assignment
#
spring.profiles.active=auth-simple
authenticationService.username=dladmin
```



```

authenticationService.password=thinkbig
###Ambari Services Check
ambariRestClientConfig.username=admin
ambariRestClientConfig.password=admin
ambariRestClientConfig.serverUrl=http://127.0.0.1:8080/api/v1
ambari.services.status=HDFS,HIVE,MAPREDUCE2,SQOOP
###Cloudera Services Check
#clouderaRestClientConfig.username=cloudera
#clouderaRestClientConfig.password=cloudera
#clouderaRestClientConfig.serverUrl=127.0.0.1
#cloudera.services.status=
##HDFS/[DATANODE,NAMENODE,SECONDARYNAMENODE],HIVE/[HIVEMETASTORE,HIVESERVER2],YARN,
↳SQOOP
# Server port
#
server.port=8420
#
# General configuration - Note: Supported configurations include STANDALONE, BUFFER_
↳NODE_ONLY, BUFFER_NODE, EDGE_NODE
#
application.mode=STANDALONE
#
# Turn on debug mode to display more verbose error messages in the UI
#
application.debug=true
#
# Prevents execution of jobs at startup. Change to true, and the name of the job that
↳should be run at startup if we want that behavior.
#
spring.batch.job.enabled=false
spring.batch.job.names=
#spring.jpa.show-sql=true
#spring.jpa.hibernate.ddl-auto=validate
# NOTE: For Cloudera metadata.datasource.password=cloudera is required
metadata.datasource.driverClassName=com.mysql.jdbc.Driver
metadata.datasource.url=jdbc:mysql://localhost:3306/kylo
metadata.datasource.username=root
metadata.datasource.password=
metadata.datasource.validationQuery=SELECT 1
metadata.datasource.testOnBorrow=true

# NOTE: For Cloudera hive.datasource.username=hive is required.

hive.datasource.driverClassName=org.apache.hive.jdbc.HiveDriver
hive.datasource.url=jdbc:hive2://localhost:10000/default
hive.datasource.username=
hive.datasource.password=
# NOTE: For Cloudera hive.metastore.datasource.password=cloudera is required.
##Also Clouder url should be /metastore instead of /hive
hive.metastore.datasource.driverClassName=com.mysql.jdbc.Driver
hive.metastore.datasource.url=jdbc:mysql://localhost:3306/hive
#hive.metastore.datasource.url=jdbc:mysql://localhost:3306/metastore
hive.metastore.datasource.username=root
hive.metastore.datasource.password=
hive.metastore.validationQuery=SELECT 1
hive.metastore.testOnBorrow=true
nifi.rest.host=localhost
nifi.rest.port=8079

```

```

elasticsearch.host=localhost
elasticsearch.port=9300
elasticsearch.clustername=demo-cluster
## used to map Nifi Controller Service connections to the User Interface
## naming convention for the property is nifi.service.NIFI_CONTROLLER_SERVICE_NAME.
↪NIFI_PROPERTY_NAME
##anything prefixed with nifi.service will be used by the UI. Replace Spaces with
↪underscores and make it lowercase.
nifi.service.mysql.password=
nifi.service.example_mysql_connection_pool.password=
jms.activemq.broker.url:tcp://localhost:61616
jms.client.id=thinkbig.feedmgr
## nifi Property override with static defaults
##Static property override supports 2 usecases
# 1) store properties in the file starting with the prefix defined in the
↪"PropertyExpressionResolver class" default = config.
# 2) store properties in the file starting with "nifi.<PROCESSORTYPE>.<PROPERTY_KEY>"
↪where PROCESSORTYPE and PROPERTY_KEY are all lowercase and the spaces are
↪substituted with underscore
##Below are Ambari configuration options for Hive Metastore and Spark location
config.hive.schema=hive
nifi.executesparkjob.sparkhome=/usr/hdp/current/spark-client
##cloudera config
#config.hive.schema=metastore
#nifi.executesparkjob.sparkhome=/usr/lib/spark
## how often should SLAs be checked
sla.cron.default=0 0/5 * 1/1 * ? *

```

Kylo Metadata

Kylo stores its metadata in the database configured in `/opt/kylo/kylo-services/conf/application.properties` in the following lines:

```

metadata.datasource.driverClassName=com.mysql.jdbc.Driver
metadata.datasource.url=jdbc:mysql://localhost:3306/kylo
metadata.datasource.username=root
metadata.datasource.password=

```

The metadata database needs to be configured in order to have Kylo metadata backed up and recovered.

For example, MySQL backup can be configured using the methods provided at <http://dev.mysql.com/doc/refman/5.7/en/backup-methods.html>.

NiFi Data

Data and metadata in NiFi is intended to be transient, and depends on the state of the flows in NiFi. However, NiFi can be configured to keep metadata and data in certain directories, and those directories can be backed up as seen fit. For example, in the `nifi.properties` file, changing

```
nifi.flow.configuration.file=/opt/nifi/data/conf/flow.xml.gz
```

will have NiFi store its flows in `/opt/nifi/data/conf/flow.xml.gz`.

With a default Kylo installation, NiFi is configured to put all of its flows, templates, data in the content repository, data in the flowfile repository, and data in the provenance repository in `/opt/nifi/data`. For more information about these

configurations, the NiFi system administrator's guide is the authority.

Startup and Shutdown

Kylo service automatically starts on system boot.

- Manual startup and shutdown from command-line:

```
$ sudo /etc/init.d/kylo-services start
$ sudo /etc/init.d/kylo-ui start
$ sudo /etc/init.d/kylo-spark-shell start

$ sudo /etc/init.d/kylo-services stop
$ sudo /etc/init.d/kylo-ui stop
$ sudo /etc/init.d/kylo-spark-shell stop
```

Log Files

Kylo uses Log4J as its logging provider.

- Default location of application log file is:

```
/var/log/kylo-<ui, services, or spark-shell>/
```

- Log files roll nightly with pipeline-application.log.<YYYY-MM-DD>
- Log levels, file rotation, and location can be configured via:

```
/opt/kylo/kylo-<ui, services, or
spark-shell>/conf/log4j.properties
```

Additional Configuration

The following section contains additional configuration that is possible.

Configuring JVM Memory

You can adjust the memory setting of the Kylo Service using the KYLO_SERVICES_OPTS environment variable. This may be necessary if the application is experiencing OutOfMemory errors. These would appear in the log files.

```
export KYLO_SERVICES_OPTS="-Xmx2g"
```






The setting above would set the Java maximum heap size to 2 GB.






Service Status Configuration

The Overview page displays Service Status as a Key Performance Indicator. The list of services is configurable using the following instructions:

Viewing Service Details

Within Kylo on the Overview tab the “Services” indicator box shows the services it is currently monitoring. You can get details of this by clicking on the Services tab:

Service Health <small>Filter</small>			
database  HEALTHY	1 Component(s)	None Alerts	10/03/2016 at 9:09 Last Checked
HDFS  WARNING	7 Component(s)	18 Alerts Alerts	10/03/2016 at 9:09 Last Checked
HIVE  HEALTHY	6 Component(s)	3 Alerts Alerts	10/03/2016 at 9:09 Last Checked
MAPREDUCE2  HEALTHY	2 Component(s)	4 Alerts Alerts	10/03/2016 at 9:09 Last Checked
Nifi  HEALTHY	1 Component(s)	None Alerts	10/03/2016 at 9:09 Last Checked
Rows per page 5 1 - 5 of 6			

Service Components <small>Filter</small>			
DATANODE  HEALTHY	STARTED Message	5 Alerts Alerts	10/03/2016 at 9:10 Last Checked
HDFS_CLIENT  HEALTHY	INSTALLED Message	0 Alerts Alerts	10/03/2016 at 9:10 Last Checked
JOURNALNODE  WARNING	UNKNOWN Message	0 Alerts Alerts	10/03/2016 at 9:10 Last Checked
NAMENODE  HEALTHY	STARTED Message	10 Alerts Alerts	10/03/2016 at 9:10 Last Checked
NFS_GATEWAY  WARNING	UNKNOWN Message	0 Alerts Alerts	10/03/2016 at 9:10 Last Checked
Rows per page 5 1 - 5 of 7			

Service Component Alerts Filter			
DataNode Process ✓ OK	TCP OK - 0.000s response on port 50010 Message	10/03/2016 at 9:10 Time	
DataNode Web UI ✓ OK	HTTP 200 response in 0.000s Message	10/03/2016 at 9:10 Time	
DataNode Unmounted D... ✓ OK	Data dir(s) are fine, /hadoop/hdfs/data . Message	10/03/2016 at 9:09 Time	
DataNode Storage ✓ OK	Remaining Capacity:[13935746895], Total Capacity:[73% Used, 52587134976] Message	10/03/2016 at 9:09 Time	
DataNode Heap Usage ✓ OK	Used Heap:[10%, 96.34919 MB], Max Heap: 1004.0 MB Message	10/03/2016 at 9:09 Time	
			Rows per page 5 1 - 5 of 5 < >

The Services Indicator automatically refreshes every 15 seconds to provide live updates on service status.

Example Service Configuration

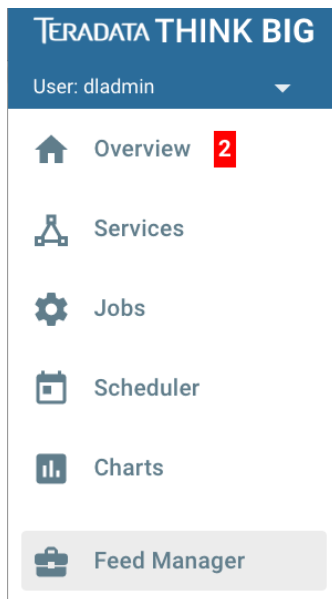
The below is the service configuration monitoring 4 services:

```
ambari.services.status=HDFS,HIVE,MAPREDUCE2,SQOOP
```











Migrating Templates and Feeds

Exporting Registered Templates

In Kylo, a template can be exported from one instance of Kylo to another. To export a template, navigate to the Feed Manager site by clicking Feed Manager on the left pane.



Then navigate to the Templates tab. All of the templates that have been registered in this instance of Kylo will be listed

Registered Templates		Filter		
 Data Confidence Invalid Records	07/28/2016 @ 2:57:09PM	Last Updated		Export
 Data Ingest	08/23/2016 @ 5:20:31PM	Last Updated		Export
 Data Transformation	07/28/2016 @ 2:53:09PM	Last Updated		Export
 Example kafka put	08/11/2016 @ 6:33:33PM	Last Updated		Export
 generate flow file source	08/23/2016 @ 3:16:49PM	Last Updated		Export
<div>Rows per page 5 1 - 5 of 14 < ></div>				


here.

To export a template, click the Export button for that template. This will download a zip archive of the template.


Importing Registered Templates

To import a registered template, on the Templates tab click on the  button in the top right. Select Import from File.

Register a new template


Create from Nifi


Register a new template that currently resides in Nifi

Import from file


Register a new template that you exported from a different Pipeline Controller Environment

Browse for the zip archive of the registered template, select whether or not to overwrite any existing registered templates with the same name, and click upload.

Import a Template


Choose Import a Nifi Template or Pipeline Controller Archive.

Type	File type	Description
Nifi Template	XML	Importing a Nifi Template will validate and import the template into Nifi.
Archive	ZIP	An archive contains both Nifi and Pipeline Controller data. This will import into Nifi and register the template in Pipeline Controller.

CHOOSE FILE

data_transformation.zip






☐ Overwrite
If template already exists it will be replaced.

IMPORT TEMPLATE


The template is now in the list of registered templates, and a feed can be created from it. This will also import the associated NiFi template into NiFi.

Exporting Feeds

To export a feed for deployment in another instance of Kylo, click on the **Feeds** tab. Similarly to the templates page, there will be a list, this time with feeds instead of templates. Click the export button to export a feed as a zip archive.

Feeds Filter			
Demo ingest feed <small>Feed Name</small>	demo2 <small>Category</small>	Data Ingest <small>Type</small>	 Export
generate flowfile to hdfs continue <small>Feed Name</small>	Sources <small>Category</small>	generate flow file source <small>Type</small>	 Export
Generated FlowFile <small>Feed Name</small>	Sources <small>Category</small>	generate flow file source <small>Type</small>	 Export
GetKafkaPutHdfs1 <small>Feed Name</small>	Imci_debug <small>Category</small>	GetKafkaPutHDFS <small>Type</small>	 Export
HDFS put <small>Feed Name</small>	Sinks <small>Category</small>	putfile sink <small>Type</small>	 Export
<div> Rows per page 5 1 - 5 of 17 </div>			

Importing Feeds

To import a feed, click the  button in the top right of the Feeds page. Click “Import” text at the top of the screen.

Select the type of feed or [Import](#) from an archive

Data Ingest



Data Ingest

Data Transformation



Data Transformation

Data Confidence Invalid Records



Data Confidence Check - Invalid record count percentage

[More](#)

Browse for the exported feed and then click **Import Feed**.

Import a Feed

Import a Pipeline Controller Feed Archive.

Type	File type	Description
Archive	ZIP	An archive contains both Nifi and Pipeline Controller Feed Data. This will import into Nifi and register the feed and respective template in Pipeline Controller .

CHOOSE FILE

☐ Overwrite
 If the Feed already exists it will be replaced.

IMPORT FEED

If the import is successful, you should now see a running feed in the Feeds tab.

Altering Feed Configurations

A feed that has been imported may have configurations specific to an environment, depending on its registered template. To change configurations on a feed, click on the **Feeds** tab in the Feed Manager site and then click on the name of the feed you want to update. A list of configurations will be present.

Feed Details

DETAILS

PROFILES

RELATED

SLA

VERSIONS

✓

Feed Definition

✎


Feed Name	Demo ingest feed
System Name	demo_ingest_feed
Description	sample description
Feed Type	Data Ingest

✓

Feed Details

✎

Source	Poll filesystem
Input Directory	/var/dropzone
File Filter	userdata\d{1,3}.csv

Click on the  icon to allow editing the fields. When done editing the fields for a section, click **Save**.

✓ Feed Details
✕

Choose a Feed Input

☒ Poll filesystem
☐ Poll database

Input Directory

/var/dropzone

The input directory from which to pull files

File Filter

userdata\d{1,3}.csv

Only files whose names match the given regular expression will be picked up

CANCEL
SAVE

Kylo recreates the flow in NiFi with the new values. Keep in mind that the values that are configurable here are determined by the registered template, so registered templates need to expose environment-specific properties if they are to be configured or updated at a feed level.

Updating Sensitive Properties in NiFi

Some NiFi processors and controller services have properties that are deemed sensitive, and are therefore not saved when exporting from Kylo. Because of this, some Kylo templates and feeds are not directly portable from one instance of Kylo to another, without some changes in NiFi. In these situations, sensitive values need to be entered directly into NiFi running on the target environment, and then the changes must be saved in a new NiFi template and used to overwrite the imported NiFi template. If the sensitive properties are only within controller services for the imported artifact, then the controller service must be disabled, the sensitive value entered, and the controller service re-enabled, but a new NiFi template does not need to be made.

It is uncommon for NiFi processors to have sensitive properties, and is most often seen in controller services, such as a DBCPConnectionPool for connection to a database. If the controller services used by a template or feed are already in existence in NiFi in the target environment, then Kylo uses those controller services. This issue only exists when importing a template or feed that has NiFi processors with sensitive properties or that use controller services that do not exist in the target environment.

Continuous Integration / Continuous Deployment (CICD)

Kylo currently does not have built-in or integrated CICD. However, Kylo allows you to export both templates (along with any registered properties) and feeds that can then be imported to any environment.

The following approach for CICD should be incorporated:

1. Build a flow in Nifi and get it configured and working in a dev instance of Nifi and Kylo as a Feed.

Once its ready to be tested export that Feed from Kylo. This export is a zip containing the feed metadata along with the categories and templates used to create the feed.

Have a separate VM running Kylo and NiFi. This would be where the scripts would create, run, and test the feeds and flows.

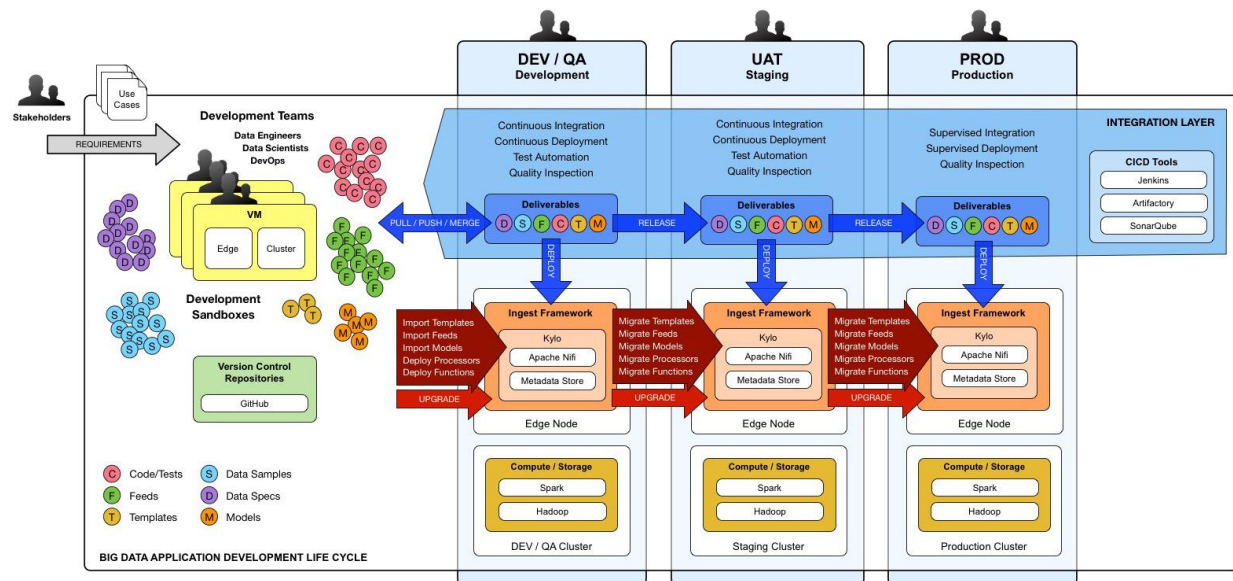
Have a separate Script/Maven project running to instantiate this feed and run it. This could look something like the following: Have a Maven module running that has a TestCase that looks for these exported feed zip files and then uses NiFi and Kylos Rest apis to create them, run the feed, verify the results, and then tear down the flow.

Kylo operates over REST and has many rest endpoints that can be called to achieve the same results as you see in the Kylo UI. For example importing a feed can be done by posting the zip file to the endpoint:

```
/v1/feedmgr/admin/import-feed
```

2. Once the tests all are passed you could take that exported Feed/Template, save it in a version control system (i.e. git), and import it into a different environment.

The graphic below depicts an example of an overall CICD ecosystem that could be implemented with Kylo with an approach similar to what Think Big R&D has put forward.



Migrating Kylo and NiFi Extensions

If custom NiFi or Kylo plugins/extensions have been built, they must be copied to all instances of NiFi and Kylo where you wish to use them. Custom NiFi extensions are packaged in .nar format, and must be placed in NiFi's lib directory. With a default Kylo installation, this directory is /opt/nifi/current/lib. Place all custom .nar files there, and restart the NiFi service.

Custom Kylo plugins belong in the /opt/kylo/kylo-services/plugin directory in a default Kylo installation. Place the .jar files for custom plugins in this directory and manually start and stop the kylo-services service.

Operational Considerations

When considering promoting Kylo/NiFi metadata you will need to restart Kylo:

- Upon changing/adding any new NiFi processors/services (changing code that creates a new NiFi plugin .nar file) you will need to bounce NiFi
- Upon changing/adding any new Kylo plugin/extension (changing the java jar) you will need to bounce Kylo (kylo-services)

Tuning the ExecuteSparkJob Processor

Problem

By default, the ExecuteSparkJob processor is configured to run in *local* or *yarn-client* mode. When a Hadoop cluster is available, it is recommended that the properties be updated to make full use of the cluster.

Solution

Your files and jars should be made available to Spark for distributing across the cluster. Additional configuration may be required for Spark to run in *yarn-cluster* mode.

1. Add the DataNucleus jars to the “Extra Jars” parameter:
 - (a) `/usr/hdp/current/spark-client/lib/datanucleus-api-jdo-x.x.x.jar`
 - (b) `/usr/hdp/current/spark-client/lib/datanucleus-core-x.x.x.jar`
 - (c) `/usr/hdp/current/spark-client/lib/datanucleus-rdbms-x.x.x.jar`
2. Add the hive-site.xml file to the “Extra Files” parameter:
 - (a) For Cloudera, this file is at `/etc/hive/conf.cloudera.hive/hive-site.xml`.
 - (b) For Hortonworks, this file is at `/usr/hdp/current/spark-client/conf/hive-site.xml`.
3. The “Validate and Split Records” and “Profile Data” processors from standard-ingest require access to the json policy file. Add “`${table_field_policy_json_file}`” to the “Extra Files” properties to make this file available.

Configure Processor

Settings | Scheduling | **Properties** | Comments

Required field + New property

Property	Value
Executor Memory	512m
Number of Executors	1
Spark Application Name	Validator
Executor Cores	1
Network Timeout	120s
Hadoop Configuration Resources	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
Yarn Queue	No value set
Spark Configurations	spark.yarn.executor.memoryOverhead=2048
Extra Files	\${table_field_policy_json_file}/usr/hdp/current/spar...

Cancel Apply

4. The “Execute Script” processor from the data-transformation reusable template requires access to the Scala script.

- (a) Change “MainArgs” to: `${transform_script_file:substringAfterLast('/')}`
- (b) Add the following to “Extra Files”: `${transform_script_file}`

Additionally, you can update your Spark configuration with the following:

1. It is ideal to have 3 executors per node minus 1 used by the manager:
 - (a) $\text{num-executor} = 3 * (\text{number of nodes}) - 1$
2. Executor cores should be either 4, 5, or 6 depending on the total number of available cores. This should be tested. Starting with 6 tends to work well:
 - (a) `spark.executor.cores = 6`
3. Determine the total memory using the following equation:
 - (a) $\text{total.memory (GB)} = \frac{\text{yarn.nodemanager.resource.memory-mb}}{\text{yarn.nodemanager.resource.cpu-vcores}} * (\text{spark.executor.cores} / \text{yarn.nodemanager.resource.cpu-vcores})$
4. Use total.memory and split it between `spark.executor.memory` and `spark.yarn.executor.memoryOverhead` (15-20% of total memory):
 - (a) `spark.yarn.executor.memoryOverhead = total.memory * (0.15)`
 - (b) `spark.executor.memory = total.memory - spark.yarn.executor.memoryOverhead`

Dealing with non-standard file formats

Problem

You need to ingest a file with a non-standard format.

Solution

There are two possible solutions:

1. You may write a custom SerDe and register that SerDe in HDFS. Then specify the use of the SerDe in the source format field of the schema tab during feed creation.
 - (a) Here's an example SerDe that reads ADSB files: <https://github.com/gm310509/ADSBSerDe>
 - (b) The dependencies in the pom.xml file may need to be changed to match your Hadoop environment.
2. You can use two feeds: 1) ingest; 2) use the wrangler to manipulate the fields into columns:
 - (a) Create an ingest field, manually define the schema as a single field of type string. You can just call that field "data".
 - (b) Make sure the format specification doesn't conflict with data in the file, i.e., tabs or commas which might cause it to get split.
 - (c) Once ingested, create a data transform feed to wrangle the data using the transform functionsHi.
 - (d) Here's an example of converting the weird ADSB format into JSON then converting into fields:

```

1 select (regexp_replace(data, "([\\w-.]+)\\t([\\w-.]+)", "\\$1\\":\\$2\\").as("data"))
2 select (regexp_replace(data, "\\\" *\\t\\\"", "\\\",\\\"").as("data"))
3 select (concat("{", data, "}").as("data"))
4 select (json_tuple(data, "clock", "hexid", "ident", "squawk", "alt", "speed",
→ "airGround", "lat", "lon", "heading"))
5 select (c0.as("clock"), c1.as("hexid"), c2.as("ident"), c3.as("squawk"), c4.as("alt"),
→ c5.as("speed"), c6.as("airGround"), c7.as("lat"), c8.as("lon"), c9.as("heading"))

```

Indexing Categories and Feeds

Problem

You need to index specific fields of categories and/or feeds.

Solution

This is a two-step process involving adding the required Maven dependencies to the Kylo project and updating the kylo-services configuration.

1. Plugins are available for indexing with Elasticsearch or Lucene (including Solr). One of these must be added as a dependency to the metadata/metadata-modeshape/pom.xml file.
 - (a) For Elasticsearch:

```

<dependency>
  <groupId>org.modeshape</groupId>
  <artifactId>modeshape-elasticsearch-index-provider</artifactId>
  <version>${modeshape.version}</version>
</dependency>

```

1. For Lucene (including Solr):

```
<dependency>
  <groupId>org.modeshape</groupId>
  <artifactId>modeshape-lucene-index-provider</artifactId>
  <version>${modeshape.version}</version>
</dependency>
```

2. Indexes are defined in the `/opt/kylo/kylo-services/conf/metadata-repository.json` file. Each index must specify a node type (`tba:category` or `tba:feed`) and a comma-separated list of columns to be indexed. User-defined properties must be URL-encoded and prefixed with `usr:.` As an example, add the following properties to the `metadata-repository.json` file:

```
{
  "indexes": {
    "category": {
      "columns": "jcr:title(String), jcr:description(String)",
      "kind": "text",
      "nodeType": "tba:category",
      "provider": "local"
    },
    "feed": {
      "columns": "jcr:title(String), jcr:description(String), tba:tags(String)",
      "kind": "text",
      "nodeType": "tba:feed",
      "provider": "local"
    }
  },
  "indexProviders": {
    "local": {
      "classname": "org.modeshape.jcr.index.elasticsearch.EsIndexProvider",
      "host": "localhost",
      "port": 9200
    }
  }
}
```

Merge Table fails when storing as Parquet using HDP

Problem

There is a bug with Hortonworks where a query against a Parquet backed table fails while using single or double quotes in the value names. For example:

```
hive> select * from users_valid where processing_dttm='1481571457830';
OK
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Failed with exception java.io.IOException:java.lang.IllegalArgumentException: Column_
↪[processing_dttm] was not found in schema!
```


Solution

You need to set some Hive properties for queries to work in Hive. These forum threads explain how to set the correct property:

1. <https://community.hortonworks.com/questions/47897/illegalargumentexception-when-select-with-where-cl.html>
2. <https://community.hortonworks.com/questions/40445/querying-a-partition-table.html>
3. On the Hive command line you can set the following property to allow quotes:

```
set hive.optimize.ppd = false;
```

NiFi becomes non-responsive

Problem

NiFi appears to be up but the UI is no longer functioning. NiFi may be running low on memory. There may be PID files in the `/opt/nifi/current` directory.

Solution

Increase memory to NiFi by editing `/opt/nifi/current/conf/bootstrap.conf` and setting the following line:

```
java.arg.3=-Xmx3g
```

Additionally, it may also be necessary to create swap space but this is not recommended by NiFi for performance reasons.

Automated Feed and Template Importing

Problem

Feeds and templates should be automatically imported into the staging or production environment as part of a continuous integration process.

Solution

The Kylo REST API can be used to automate the importing of feeds and templates.

Templates can be imported either as an XML or a ZIP file. Set the *overwrite* parameter to *true* to indicate that existing templates should be replaced otherwise an error will be returned. Set the *createReusableFlow* parameter to *true* if the template is an XML file that should be imported as a reusable template. The *importConnectingReusableFlow* parameter indicates how to handle a ZIP file that contains both a template and its reusable flow. The *NOT_SET* value will cause an error to be returned if the template requires a reusable flow. The *YES* value will cause the reusable flow to be imported along with the template. The *NO* value will cause the reusable flow to be ignored and the template to be imported as normal.

```
curl -F file=@<path-to-template-xml-or-zip> -F overwrite=false -F   
↪createReusableFlow=false -F importConnectingReusableFlow=NOT_SET -u <kylo-user>:  
↪<kylo-password> http://<kylo-host>:8400/proxy/v1/feedmgr/admin/import-template
```

Feeds can be imported as a ZIP file containing the feed metadata and NiFi template. Set the *overwrite* parameter to *true* to indicate that an existing feed and corresponding template should be replaced otherwise an error will be returned. The *importConnectingReusableFlow* parameter functions the same as the corresponding parameter for importing a template.

```
curl -F file=@<path-to-feed-zip> -F overwrite=false -F   
↪importConnectingReusableFlow=NOT_SET -u <kylo-user>:<kylo-password> http://<kylo-  
↪host>:8400/proxy/v1/feedmgr/admin/import-feed
```

Spark job failing on sandbox with large file

Problem

If running on a sandbox (or small cluster) the spark executor may get killed due to OOM when processing large files in the standard ingest flow. The flow will route to failed flow but there will be no error message. Look for Exit Code 137 in `/var/log/nifi/nifi-app.log`. This indicates an OOM issue.

Solution

On a single-node sandbox it is better to run Spark in *local* mode than *yarn-client* mode and simply give Spark enough memory to perform its task. This eliminates all the YARN scheduler complications.

1. In the standard-ingest flow, stop and alter the ExecuteSparkJob processors:
 - (a) Set the SparkMaster property to *local* instead of *yarn-client*.
 - (b) Increase the Executor Memory property to at least 1024m.
2. Start the processors.

NiFi hangs executing Spark task step

Problem

Apache NiFi flow appears to be stuck inside the Spark task such as “Validate and Split Records” step. This symptom can be verified by viewing the YARN jobs. The Spark job appears to be running and there is a Hive job queued to run but never launched: <http://localhost:8088/cluster>

So what is happening? Spark is executing a Hive job to insert data into a Hive table but the Hive job never gets YARN resources. This is a configuration problem that leads to a deadlock. Spark will never complete because the Hive job will never get launched. The Hive job is blocked by the Spark job.

Solution

First you will need to clean up the stuck job then re-configure the YARN scheduler.

To clean up the stuck job, from the command-line as root:

1. Obtain the PID of the Spark job:

```
ps -ef | grep Spark | grep Validator
```

2. Kill the Spark job:

```
kill <pid>
```

Configure YARN to handle additional concurrent jobs:

1. Increase the maximum percent with the following parameter (see: <https://hadoop.apache.org/docs/r0.23.11/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html>):

```
yarn.scheduler.capacity.maximum-am-resource-percent=0.8
```

2. Restart the cluster or all affected services.
3. Restart Apache NiFi to re-initialized Thrift connection pool:

```
service nifi restart
```

Note: In Ambari, find this under Yarn | Configs (advanced) | Scheduler.

Spark SQL fails on empty ORC and Parquet tables

Problem

Your spark job fails when running in HDP 2.4 or 2.5 while interacting with an empty ORC table. A likely error that you will see is:

```
ExecuteSparkJob[id=1fblb9a0-e7b5-4d85-87d2-90d7103557f6] java.util.  
↪NoSuchElementException: next on empty iterator
```

This is due to a change Hortonworks added that modified how it loads the schema for the table.

Solution

To fix the issue, you can take these steps:

1. On the edge node, edit the file: /usr/hdp/current/spark-client/conf/spark-defaults.conf
2. Add these configuration entries to the file:

```
spark.sql.hive.convertMetastoreOrc false  
spark.sql.hive.convertMetastoreParquet false
```

See

High Performance NiFi Setup

Problem

The NiFi team published an article on how to extract the most performance from Apache NiFi.

Solution

See

RPM install fails with ‘cpio: read’ error

Problem

Kylo rpm install fails giving a ‘cpio: read’ error.

Solution

This problem occurs if the rpm file is corrupt or not downloaded properly. Try re-downloading the Kylo rpm from the Kylo website.

Accessing Hive tables from Spark

Problem

You receive a NoSuchTableException when trying to access a Hive table from Spark.

Solution

Copy the hive-site.xml file from Hive to Spark.

For Cloudera, run the following command:

```
cp /etc/hive/conf/hive-site.xml /usr/lib/spark/conf/
```

Compression codec not found for PutHDFS folder

Problem

The PutHDFS processor throws an exception like:

```
java.lang.IllegalArgumentException: Compression codec com.hadoop.compression.lzo.  
↳LzoCodec not found.
```

Solution

Edit the `/etc/hadoop/conf/core-site.xml` file and remove the failing codec from the `io.compression.codecs` property.

Creating a cleanup flow

Problem

When deleting a feed it is sometimes useful to run a separate NiFi flow that will remove any HDFS folders or Hive tables that were created by the feed.

Solution

1. You will need to have a controller service of type `JmsCleanupEventService`. This service has a `Spring Context Service` property that should be connected to another service of type `SpringContextLoaderService`.
2. In your NiFi template, create a new input processor of type `TriggerCleanup`. This processor will be run automatically when a feed is deleted.
3. Connect additional processors such as `RemoveHDFSFolder` or `DropFeedTables` as needed.

Accessing S3 from the data wrangler

Problem

You would like to access S3 or another Hadoop-compatible filesystem from the data wrangler.

Solution

The Spark configuration needs to be updated with the path to the JARs for the filesystem.

To access S3 on HDP, the following must be added to the `spark-env.sh` file:

```
export SPARK_DIST_CLASSPATH=$(hadoop classpath)
```

Additional information is available from the .

Dealing with XML files

Problem

You need to ingest an XML file and parse into Hive columns.

Solution

1. You can use two feeds: 1) ingest; 2) use the wrangler to manipulate the fields into columns:
 - (a) Create an ingest field and manually define the schema as a single field of type string. You can just call that field “data”.
 - (b) Make sure the format specification doesn’t conflict with data in the file, i.e. tabs or commas which might cause it to get split.
 - (c) Once ingested, create a data transform feed to wrangle the data using the transform functions.
 - (d) Here’s an example of converting XML to columns using wrangler functions:

XML Explode

```
1  select (regexp_replace(contents, "(?s).*<TicketDetails>\\s*<TicketDetail>\\s*", "").  
    ↪as("xml"))  
2  select (regexp_replace(xml, "(?s)</TicketDetails>.*", "").as("xml"))  
3  select (split(xml, "<TicketDetail>\\s*").as("TicketDetails"))  
4  select (explode(TicketDetails).as("TicketDetail"))  
5  select (concat("<TicketDetail>", TicketDetail).as("TicketDetail"))  
6  xpath_int(TicketDetail, "//Qty").as("Qty")  
7  xpath_int(TicketDetail, "//Price").as("Price")  
8  xpath_int(TicketDetail, "//Amount").as("Amount")  
9  xpath_int(TicketDetail, "//NetAmount").as("NetAmount")  
10 xpath_string(TicketDetail, "//TransDateTime").as("TransDateTime")  
11 drop("TicketDetail")
```

Dealing with fixed width files

Problem

You need to load a fixed-width text file.

Solution

This is possible to configure with the schema tab of the feed creation wizard. You can set the SerDe and properties:

1. Create an ingest feed.
2. When at the schema tab look for the field (near bottom) specifying the source format.
3. Manually build the schema since Kylo won’t detect the width.
4. Place text as follows in the field substituting regex based on the actual columns:

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'  
WITH SERDEPROPERTIES ("input.regex" = "({10})({20})({20})({20})({5}).\\s*")
```

Dealing with custom SerDe or CSV files with quotes and escape characters

Problem

You need to load a CSV file with surrounding quotes and don't want those quotes removed.

Solution

This is possible to configure within the schema tab of the ingest feed creation, you can set the SerDe and properties:

1. Create an ingest feed.
2. When at the schema tab look for the field (near bottom) specify the source format.
3. See the Apache wiki .
4. Place text as follows in the field:

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar"      = "\"\"",
  "escapeChar"="\"\\\\\\\\\"");
)
```

Notice the double escape required!

Configuration on a Node with Small Root Filesystem

Problem

The node that Kylo will run on has a small root filesystem. There are other mounts that contain larger space but in particular, the following directories contain 30GB or less.

- **/opt** which is used for libraries, executables, configs, etc
- **/var** which is used for logs, storage, etc
- **/tmp** which is used for processing data

For Kylo, these directories get filled up very quickly and this causes all processes on the edge node to freeze.

Solution

In general, the solution is to move all the large files onto the larger data mount. For this solution, the /data directory is considered to be the largest and most ideal location to contain Kylo artifacts (logs, storage, etc).

To alleviate the disk space issues, these steps were taken to move items to the /data directory

Relocate MySQL

The default location of MySQL is /var/lib/mysql. MySQL will fill up the root partition with the default configuration so the storage volumes for MySQL must be migrated to /data/mysql.

1. Stop MySQL: **service mysql stop**

2. Copy data over to new location: **rsync -av /var/lib/mysql /data/**
3. Backup the existing data: **mv /var/lib/mysql /var/lib/mysql.bak**
4. Backup the existing my.cnf: **cp /etc/my.cnf /etc/my.cnf.bak**
5. Update MySQL config with new location with the values below: **vi /etc/my.cnf**
 - (a) Under [mysqld], set datadir = /data/mysql
6. Start MySQL: **service mysql start**
7. Back up old MySQL directory: **tar -zcvf mysql_bak.tar.gz mysql.bak**

Change properties to point to /data

1. Kylo
 - (a) Update /opt/kylo-services/log4j.properties
 - i. log4j.appender.file.File=/data/log/kylo-services/kylo-services.log
 - (b) Update /opt/kylo-services/log4j-spark.properties
 - i. log4j.appender.file.File=/data/log/kylo-services/kylo-spark-shell.log
 - (c) Update /opt/kylo-ui/log4j.properties
 - i. log4j.appender.file.File=/data/log/kylo-ui/kylo-ui.log
2. Nifi
 - (a) Update /opt/nifi/nifi.properties
 - i. nifi.flow.configuration.file=/data/opt/nifi/data/conf/flow.xml.gz
 - ii. nifi.flow.configuration.archive.dir=/data/opt/nifi/data/conf/archive/
 - iii. nifi.authorizer.configuration.file=/data/opt/nifi/data/conf/authorizers.xml
 - iv. nifi.login.identity.provider.configuration.file=/data/opt/nifi/data/conf/login-identity-providers.xml
 - v. nifi.templates.directory=/data/opt/nifi/data/conf/templates
 - vi. nifi.flowfile.repository.directory=/data/opt/nifi/data/flowfile_repository
 - vii. nifi.content.repository.directory.default=/data/opt/nifi/data/content_repository
 - viii. nifi.provenance.repository.directory.default=/data/opt/nifi/data/provenance_repository
3. Elasticsearch
 - (a) Update /opt/elasticsearch/elasticsearch.yml
 - i. path.data: /data/elasticsearch
 - ii. path.logs: /data/log/elasticsearch

GetTableData vs ImportSqoop Processor

Problem

You need to load data from a structured datastore.

Solution

There are two major NiFi processors provided by Kylo for importing data into Hadoop: GetTableData and ImportSqoop.

1. **GetTableData** leverages JDBC to pull data from the source into the flowfile within NiFi. This content will then need to be pushed to HDFS (via a PutHDFS processor).
2. **ImportSqoop** executes a Sqoop job to pull the content from the source and place it directly to HDFS. For details on how this is done, please refer to [Apache Sqoop](#).

In general, it is recommended to use the ImportSqoop processor due to performance. Using the GetTableData processors uses the edge node (where NiFi is running) as a middle-man. The ImportSqoop processor runs a MapReduce job that can be tuned to load the data efficiently. For example, a single mapper will be sufficient if you are loading a reference table but a table with billions of rows would benefit from multiple mappers.

The GetTableData processor should be used when the data being pulled is small. Other use cases are when certain pre-processing steps are required that benefit from being on the edge node. For instance, if the edge node resides behind a firewall and PII (personal identifiable information) fields need to be masked before being pushed to a more open HDFS environment.

Kylo's Data Ingest template comes with out-of-the-box support for the GetTableData processor. To use the ImportSqoop processor instead, the following changes should be made to the Data Ingest template and the standard-ingest reusable template:

1. Replace the GetTableData processor with the ImportSqoop processor
2. Remove the PutHDFS processor from the flow
3. Update the "Create Feed Partition" processor to point to the target location of the ImportSqoop processor
4. Create a new archive processor which will archive data from HDFS. One option is use the Hadoop streaming tool to take the files residing in the target location of the ImportSqoop processor and compress then store the data to the archive directory. For details on this, please refer to [Hadoop Streaming](#).

It is important to note that any other templates that output to standard-ingest would need to be updated because the changes above assumes data resides in HDFS. In general, adding a PutHDFS processor would be sufficient.

Using machine learning functions

Problem

You need to use a machine learning function in a data transformation feed.

Solution

Kylo provides many functions from the Spark ML package. Below is an example of using linear regression to estimate the number of tickets bought based on the price paid. The `run()` function performs both the fit and transform operations of the linear regression. It requires a `DataFrame` as a parameter which is used for the fit operation, in the case below it uses `limit(10)`.

```
1 vectorAssembler(["pricepaid"], "features")
2 qtySold.cast("double").as("label")
3 LinearRegression().setMaxIter(10).setRegParam(0.01).run(limit(10))
```

Sqoop requires JDK on Kylo sandbox

Problem

This issue is known to exist for Kylo sandbox version 0.7.1. The file name for the sandbox is kylo-hdp-sandbox-0.7.1.ova. Sqoop job throws an error “Sqoop requires a JDK that can compile Java code.”

Solution

Sqoop requires a JDK to compile Java code. The steps to install a JDK and fix this error are listed below:

1. Install Open JDK 7.

```
root@sandbox ~# yum install java-1.7.0-openjdk-devel
```

2. Verify JDK version.

```
root@sandbox ~# javac -version
javac 1.7.0_131
```

3. Verify actual location.

```
root@sandbox ~# ls -l /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.131-2.6.9.0.el7_3.x86_64/
↪ bin/javac
-rwxr-xr-x 1 root root 7368 Feb 13 17:16 /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.131-2.
↪ 6.9.0.el7_3.x86_64/bin/javac
```

4. Update /etc/hadoop/conf/hadoop-env.sh. (Find existing entry and update it)

```
root@sandbox ~# vi /etc/hadoop/conf/hadoop-env.sh
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.131-2.6.9.0.el7_3.x86_64/
```

5. Re-run Sqoop flow.

Validator is unable to process policy JSON file

Problem

Validator throws an error while trying to process the policy JSON file. This issue may be caused due to manual editing of the file in an editor and pasting the result back in NiFi.

Solution

Ensure that the policy file is correctly formatted. External editors can sometimes put in invalid characters. One way to do this verification is at: [JSON Pretty Print](#). Paste in the policy file in the text box and click ‘Pretty Print JSON’. If the JSON is valid, it will be shown in a more readable format. Otherwise, a `null` will be output.

The following document describes patterns and best practices particularly oriented to IT Designers and System Administrators.

Organizational Roles

Kylo supports the division of responsibility between IT designers, administrators, operations, and end-users.

Role separation

A key tenet of Kylo is IT governed self-service. Most activities such as data ingest and preparation are possible by data analysts who may have deep understanding of their data but not appreciate the advanced data processing concepts of Hadoop. It is the responsibility of the Designer to build models that incorporate best practices and maintain the ability for end-users to easily configure feeds.

Designers are responsible for developing templates for pipelines using Apache NiFi. When configured in Kylo provide the processing model for feeds created by end-users. System Administrators are responsible for activities such as install, configuration, connections, security, performance tuning and role-based security.

Designers

Designers are responsible for developing templates for pipelines using Apache NiFi. When configured in Kylo provide the processing model for feeds created by end-users. System Administrators are responsible for activities such as install, configuration, connections, security, performance tuning and role-based security.

Designers should limit the properties exposed to end-users and assume a user has limited knowledge of the internal working of the pipeline. For example, it is poor practice to expose Spark parameters, paths to libraries, memory settings, concurrency settings, etc. However, a user creating a feed should know the name of file(s) to load, whether they want to do a snapshot or merge, and target table names and business metadata.

Designers use the NiFi expression language and Kylo's built-in metadata properties to auto-wire processor components in the NiFi flow to the wizard UI.

Administrators

NiFi/Hadoop Administrators are typically system administrators who need to control resource utilization, such as memory and concurrency. These activities are typically configured directly in NiFi.

The Administrator is also responsible for configuring NiFi Controller Services, which may contain privileged database and services login configuration.

The Administrator must review new pipelines to understand how shared resources are utilized. For example, a flow may use excessive resources on the edge node or may need to be properly tuned for the size of the target cluster. Administrators may modify resource behavior such as concurrency, back-pressure settings, Spark driver memory, and number of mappers.

The Administrator should also evaluate new flows and understand security implications or security vulnerabilities introduced as NiFi operates as a privileged user.

Operations

An Operator uses the Operations Manager dashboard to monitor activity in the system and relies on alerts. The Designer should consider that an Operations user may need to respond to problems and recover from errors.

Users

Users can include data analytics, data scientists, and data stewards who interact with the Kylo application. Administrator determines what features are available to users based on roles. Designers determine how users are able to configure feeds based on templates.

Designers

Guidance for designers who design pipeline templates and enable self-service.

NiFi Template Design

The Designer is responsible for developing Apache NiFi templates, which provide the processing model for feeds in Kylo. Once a template has been registered with the Kylo framework through the administrative template UI, Kylo allows end-users to create and configure feeds (based on that model) through a user-friendly, guided wizard. The use of templates embodies the principle of “write-once, use-many”.

The Designer determines which parameters are settable by an end-user in the wizard UI, how the field is displayed (for example: picklist, SQL window, numeric field), and any defaults or constraints. The Designer may also wire parameters to environment-specific properties and any standard metadata properties provided by the UI wizard used by end-users.

After a template is registered in Kylo, an end-user will be able to create new feeds based on that template using the UI-wizard. End-users may only set parameters exposed by the template designer.

A well-written template may support many feeds. It should incorporate best practices and consider security, regulatory requirements, and error handling.

A good reference model is Kylo's standard ingest template. This can serve as a model for best practices and can be adapted to an organization's individual requirements.

Use Reusable Flows

When possible, consider using re-usable flows. A reusable flow is a template that creates just a single instance of a flow. A single instance simplifies administration and future updates. All feeds utilizing a reusable flow will inherit changes automatically.

A re-usable flow will require at least two templates: 1) The feed flow instance template, and 2) the re-usable flow template.

The feed flow instance will be generated each time a feed is created and will have the feed-specific configuration defined by the end-user. The feed-instance defines an output to the re-usable flow. The re-usable flow template will have an input from the feed-instance flow.

When a Designer registers the re-usable template and the feed instance template, the Designer is prompted to wire together the input and output. Kylo will take care of auto-wiring these each time a new feed is created.

Error Handling

Error handling is essential to building robust flows.

NiFi processors have the ability to route to success or failure paths. This allows the Designer to setup standard error handling. The Designer should ensure that data is never lost and that errors allow an Operator to recover.

Kylo is configured to look for any activity along standard failure paths and trigger alerts in Ops Mgr.

A best practice is to handle errors in consistent ways through a reusable "error flow". Potentially, a custom NiFi processor could be developed to make this convenient for Designers.

Some processors automatically support retries, providing a penalty to incoming flowfiles. An example of this case is when a resource is temporarily unavailable. Rather than failing, the flowfile will be penalized (delayed) and re-attempted at a later point.

Preserve Edge Resources

The edge node is a limited resource, particularly compared to the Hadoop cluster. The cluster will have a magnitude greater IO and processing capacity than the edge, so if possible avoid moving data through Apache NiFi. Strive to move data directly from source to Hadoop and performing any data processing in the cluster.

There may be good arguments to perform data processing through the edge node, in this case a single edge node may be insufficient and require a small NiFi cluster along the edge.

Note: The advantage of external Hive tables is the ability to simply mount an HDFS file (external partition). This means data can be moved to HDFS, and then surfaced in a table through a simple DDL (ADD PARTITION).

Generalize Templates

Templates allow the Designer to promote the "write-once,use-many" principle. That is, once a template is registered with Kylo, any feeds created will utilize the model provided. The Designer should consider parameterizing flows to support some derivative data use cases, while always striving to maintain ease of use for end-users, who have to create feeds and ensure their testability.

An example of this type of flexibility is a flow that allows the end-user to select from a set of sources (for example: kafka, filesystem, database) and write to different targets (for example: HDFS, Amazon S3). A single template could feasibly provide this capability. There is no need to write nxn templates for each possible case.

It may be necessary to write “exotic templates” that will only be used once by a single feed. This is also fine. The Designer should still consider other best practices, such as portability. See chaining feeds below for a possible alternative to this.

Chaining Feeds

Instead of creating long special-purposed pipelines, consider breaking the pipeline into a series of feeds. Each feed then represents a significant movement of data between source and sink (for example: ingest feed, transform feed A, transform feed B, export feed).

Kylo provides the ability to chain feeds together via *preconditions*. *Preconditions* define a rule for the “event” that will trigger a feed. Preconditions allow triggering based on the completion of one or more predecessor jobs. The ability to define *preconditions* can be enabled by a Designer and configured by a Data Analyst during the feed creation process. This allows for sophisticated chaining of feeds without resorting to the need to build specially-purpose pipelines.

One-Time Setup and Deletion

The Designer should incorporate any one-time setup, and any processing flow required for deletion of a feed. One time setup is referred to as *registration* within a feed. The metadata server can route a flow through a one-time registration process to setup Hive tables and HDFS paths.

A proper deletion routine should delete all the Hadoop artifacts created by a feed. Delete allows a user to test a feed and easily delete it if needed.

Lineage Tracking

Kylo framework only automatically maintains lineage at the “feed-level” and by any sources and sinks identified by the template designer when registering the template.

A Designer may utilize additional capabilities of Kylo’s metadata server by issuing REST calls from a NiFi flow. This can be done one time at registration, or for each feed instance. For example, the Designer may wish to track detailed lineage between a series of transforms and data sources. See Metadata Server REST API documentation.

Idempotence

Pipelines and template steps should be idempotent, such that if work is replayed it will produce the same result without a harmful side effect such as duplicates.

Environment Portability

NiFi Templates and associated Kylo configuration can be exported from one environment and imported into another environment. The Designer should ensure that Apache NiFi templates are designed to be portable across development, test and production environments .

Environment-specific settings such as library paths or URLs should be specified in the environment-specific settings file in Kylo. See documentation. Environment-specific variables can be set through an environment specific properties file. Kylo provides an expression syntax for a Designer to utilize these properties when registering the template. An Administrator typically maintains the environment-specific settings.

Data Confidence

In addition to NiFi templates for feeds, a Designer can and should create templates for performing Data Quality (DQ) verification of those feeds. Data Quality verification logic can vary but often can be designed to be generalized into a few common patterns.

Examples of a DQ template might evaluate the profile statistics from the latest run and use those statistics such as ratio of valid-to-invalid records. Another check could compare aggregates in the source table against Hadoop to verify that totals match at certain intervals (for example: nightly revenue roll-ups match).

A special field identifies the template as a DQ check related to a feed and used for Data Confidence KPI, alerts, and feed health by the Ops manager. See Manual.

Data Ingestion

Archival: It is best practice to preserve original raw content and consider regulatory compliance. Also, consider security and encryption at rest since raw data may contain sensitive information. After a retention period is passed, information may be deleted. ILM feeds can be created to do this type of house-keeping. Retention policies can optionally be defined by a feed or business metadata at the category-level.

Make sure to secure intermediate tables and HDFS locations used for data processing. These tables may contain views of raw, sensitive data. Intermediate tables may require different security requirements than the managed table. Additionally, the data may need to go on an encryption zone on HDFS. Administrators and Operators may need visibility for troubleshooting, but typical end-users should not see intermediate data.

Avoid “transformations” to raw. Best practice is to ingest the raw source (although consider protecting sensitive data) and avoid transformation of the data.

Cleanup Intermediate Data

The intermediate data generated by feed processing should be periodically deleted. It may be useful to have a brief retention period (for example: 72 hours) for troubleshooting. A single cleanup feed can be created to do this cleanup.

Data Cleansing and Standardization

Kylo includes a number of useful cleansing and standardization functions that can be configured by an end-user in the feed creation wizard UI.

Avoid using the cleansing and standardization capabilities to do complex “transformation” data. It should be primarily used for manipulating data into conventional or canonical formats (for example: simple datatype conversion such as dates, stripping special characters) or data protection (for example: masking credit cards, PII, etc.)

Kylo provides an extensible Java API for developing custom cleansing and standardization routines.

Validation

Hive is extremely tolerant of inconsistencies between source data and the HCatalog schema. Using Hive without additional validation will allow data quality issues to go unnoticed and extremely difficult to detect.

Kylo automatically provides schema validation, ensuring that source data conforms to target schema. For example, if a field contains alpha characters and is destined for a numeric column, Kylo will flag the record as invalid.

Additionally users can define field-level validation to protect against data quality issues.

Kylo provides an extensible Java API for developing custom validation routines.

Data Profiling

Kylo's Data profiling routine generates statistics for each field in an incoming dataset.

Beyond being useful to Data Scientists, profiling is useful for validating data quality (See Data Quality checking).

RDBMS Data

Joins in Hadoop are inefficient. Consider de-normalizing data during ingest. One strategy is to ingest data via views.

File Ingest

One common problem with files is ensuring they are fully written from a source before they are picked up for processing. A strategy for this is to set the process writing the file to either change permissions on the file after the write is complete, or append a suffix such as DONE.

Character Conversion and Hive

Hive works with UTF-8. Character conversion may be required for any records that should be queried from Hive. NiFi provides a character conversion processor that can be used for this. Kylo can detect source encoding using Tikka.

Development Patterns

Best practices and guidance oriented to the development process, release, and testing.

Development Process

NiFi templates should be developed and tested in a personal development environment. Do not develop NiFi templates in the production NiFi instance used by Kylo.

It is recommended to do initial testing in NiFi. Once the flow has been tested and debugged within NiFi, then register the template with Kylo in the development environment, where one can test feed creation.

Note: Controller Services that contain service, cluster, and database connection information should be setup by the Developer using their personal login information. In production, an Administrator manages these controller services, and they typically operate as an application account with elevated permissions.

Template Export/Import

As stated previously, it is recommended that Apache NiFi template development occur in a development environment. This is a best practice from a security and operations perspective. Kylo allows templates and the registration metadata to be exported to a ZIP file. This file can be imported into a new environment.

Feed Export/Import

Although Kylo can be used for self-service feed creation in production, some organizations prefer to lock this ability down and perform feed development and testing in a separate environment.

Version Control

It is recommended to manage exported templates and feeds through an SCM tool such as git, subversion, or CVS.

Users

Best practices and guidance oriented to end-users (users of the Kylo application).

When to Use Snapshot

Kylo allows users to configure feeds to do incremental updates or to enable the use of a snapshot (replacing the target with the entire contents). In the case of RDBMS, where there small source tables, it may be more efficient to simply overwrite (snapshot) the data each time. Tables with less than 100k records probably fit the snapshot pattern.

When to Use Timer (vs. Cron)

Timer is a good scheduling technique for lightweight polling behavior. Be aware, however, that all timers fire concurrently when NiFi starts. Avoid using for processors that place heavy demand on a source when triggered. For example: database sources or launching a transformation workflow. Cron is a more appropriate scheduling option for these resource-intensive processors.

Wrangling

The wrangling utility allows for users to do visual drag-drop SQL joins and apply transform functions to build complex transformations in a WYSIWG, Excel-like interface. This is a recommended method for performing transformations on raw data.

Service Level Agreements

Service level agreements are created by users to enforce service levels, typically related to feeds. An SLA may set a threshold tolerance for data arrival time or feed processing time. An SLA can enforce ratio of invalid data from a source.

SLAs are useful for alerting and measuring service level performance over-time.

Administrators

Back-Pressure

Administrators (and Designers) should understand NiFi capabilities regarding back-pressure. Administrators can configure backpressure limits at the processor level to control how many flow files can be queued before upstream processors start to throttle activity. This can assure that a problem with a service doesn't cause a huge queue or result in a large number of failed jobs.

Business Metadata

Business metadata is any information that enriches the usefulness of the data, or is potentially helpful for future processing or error handling.

Kylo allows an Administrator to setup business metadata fields that a user sees when creating a feed. These business metadata templates can be setup either globally or at the category-level. Once setup, the user is prompted to fill this information in the Properties step of the Ingest wizard.

Security

Guidance around security.

Security Vulnerabilities

Designers and Administrators should be aware of introducing a backdoor for malicious users, or even for developers. Although NiFi components are extremely powerful, be aware of SQL Injection or exposing the ability for a user to paste script.

Consider issues such as a malicious user configuring an ingestion path that accesses secure files on the file system.

When importing feeds from other environments, the Administrator should always ensure that the security group is appropriate to the environment. A security group that may be appropriate in a development environment might not be inappropriate for production.