Kylo Documentation

Release 0.8.4

Think Big, a Teradata Company

Dec 22, 2017

About

Kylo website:

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

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

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	

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	

Keberos 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

FAQ

2.1 About Kylo

2.1.1 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.

2.1.2 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.

2.1.3 What does Kylo mean?

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

2.1.4 What software license is Kylo provided under?

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

2.1.5 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.

2.1.6 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.

2.2 Enterprise Support

2.2.1 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.

2.2.2 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.

2.2.3 Is enterprise training available for Kylo?

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

2.2.4 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.

2.3 Architecture

2.3.1 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.

2.3.2 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

2.3.3 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.

2.3.4 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.

2.3.5 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.

2.3.6 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.

2.4 Metadata

2.4.1 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.

2.4.2 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.

2.4.3 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:

2.4.4 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.

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

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

2.4.6 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**

2.4.7 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.

2.4.8 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.

2.5 Development Lifecycle

2.5.1 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.

2.5.2 Does deployment of new templates or feeds require restart?

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

2.5.3 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).

2.6 Tool Comparisons

2.6.1 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.

2.6.2 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.

2.6.3 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.

2.6.4 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.

2.6.5 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.

2.7 Scheduler

2.7.1 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.

2.7.2 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.

2.7.3 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".

2.7.4 Does Kylo support 3rd party schedulers

Yes, feeds can be triggered via JMS or REST.

2.7.5 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.

2.8 Security

2.8.1 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.

2.8.2 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*.

2.8.3 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.

2.8.4 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.

2.9 Data Ingest

2.9.1 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.

2.9.2 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.

2.9.3 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.

2.9.4 Which target formats for Hive does Kylo support?

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

2.9.5 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.

2.9.6 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.

Terminology

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.

3.1 Apache NiFi Terminology

3.1.1 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.

3.1.2 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.

3.1.3 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.

3.1.4 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.

3.1.5 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.

3.1.6 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.

3.1.7 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.

3.1.8 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.

3.2 Kylo Terminology

3.2.1 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.

3.2.2 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

3.2.3 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.

3.2.4 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.

3.2.5 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.

3.2.6 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.

3.2.7 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.

Release Notes

4.1 Latest Stable Release

4.1.1 Release 0.8.4.1 (December 21, 2017)

Highlights

- Enhanced feed stepper plugin capabilities (Documentation & Examples)

- 11 Issues fixed.

Download Links

- Visit the *Downloads* page for links.

Upgrade Instructions from v0.8.4

1. Backup any Kylo plugins

   When Kylo is uninstalled it will backup configuration files, but not the */plugin* jar files. If you have any custom plugins in either *kylo-services/plugin* or *kylo-ui/plugin* then you will want to manually back them up to a different location.

2. Uninstall Kylo:

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

3. Install the new RPM:

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

4. Restore previous application.properties files. If you have customized the the application.properties, copy the backup from the 0.8.3 install.

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/conf/application.properties /opt/kylo/
↪kylo-services/conf/application.properties.0_8_3_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 Copy the /bkup-config/TIMESTAMP/kylo-ui/application.properties file to */opt/kylo/kylo-ui/conf*

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. **NOTE:** Kylo no longer ships with the default **dladmin** user. You will need to re-add this user only if you're using the default authentication configuration:

- Uncomment the following line in `/opt/kylo/kylo-services/conf/application.properties` and `/opt/kylo/kylo-ui/conf/application.properties`:

```
security.auth.file.users=file:///opt/kylo/users.properties
security.auth.file.groups=file:///opt/kylo/groups.properties
```

- Create a file called `users.properties` file that is owned by kylo and replace **dladmin** with a new username and **thinkbig** with a new password:

```
echo "dladmin=thinkbig" > /opt/kylo/users.properties
chown kylo:users /opt/kylo/users.properties
chmod 600 /opt/kylo/users.properties
```

- Create a file called `groups.properties` file that is owned by kylo and set the default groups as below. Replace **dladmin** in line 1 with the username populated in `users.properties` in step above.

```
vi /opt/kylo/groups.properties
```

```
dladmin=admin,user
analyst=analyst,user
designer=designer,user
operator=operations,user
```

```
chown kylo:users /opt/kylo/groups.properties
chmod 600 /opt/kylo/groups.properties
```

6. Start Kylo

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

4.2 Previous Releases

4.2.1 Release 0.8.4 (November 29, 2017)

Highlights

- NiFi-1.4.0 support

- Enhanced Operations Manager dashboard with better performance

- SLA improvements including customizable SLA email templates

- Enhanced operations streaming statistics supporting more viewing options

- Ability to clone an existing Feed

- Visual query enhancements. The Transform Data step has been improved with UI enhancements including a context menu when clicking on a row or highlighting text.

- Preview validation errors. Apply domain types in a Data Transformation feed and preview which rows are invalid.

- Secure installation. Default usernames and passwords can be customized during installation to ensure a secure environment.

- Global search enhancements. Deleting a feed will remove its data from search results. Re-processing same data via a feed will not duplicate search results.

- 136 Issues fixed

Download Links

- Visit the *Downloads* page for links.

Upgrade Instructions from v0.8.3

1. Stop NiFi:

```
service nifi stop
```

2. Backup any Kylo plugins

   When Kylo is uninstalled it will backup configuration files, but not the */plugin* jar files. If you have any custom plugins in either *kylo-services/plugin* or *kylo-ui/plugin* then you will want to manually back them up to a different location.

3. Uninstall Kylo:

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

4. Install the new RPM:

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

5. Global search configuration (only applicable if using Elasticsearch):

   5.1. This step to create kylo indexes may already have been performed as part of v0.8.3 installation. If indexes already exist, Elasticsearch will report an `index_already_exists_exception`. It is safe to ignore this and continue.

   Change the host and port if necessary. The last two parameters define *num-shards* and *num-replicas*, and can be kept as 1 for development environment.

   ```
   /opt/kylo/bin/create-kylo-indexes-es.sh localhost 9200 1 1
   ```

   5.2. If using Elasticsearch v5, update the **Index Text Service** feed. This step should be done once Kylo services are started and Kylo is up and running. [Note: This requires NiFi 1.3 or later]

   Import the feed `index_text_service_v2.feed.zip` file available at `/opt/kylo/setup/data/feeds/nifi-1.3`. Click 'Yes' for these options during feed import (a) Overwrite Feed (b) Replace Feed Template (c) Replace Reusable Template.

   5.3. If using Elasticsearch v2, install an additional plugin to support deletes. If required, change the location to where Elasticsearch is installed.

   ```
   sudo /usr/share/elasticsearch/bin/plugin install delete-by-query
   service elasticsearch restart
   ```

6. Restore previous application.properties files. If you have customized the the application.properties, copy the backup from the 0.8.3 install.

   6.1 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:

   6.2 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/conf/application.properties /opt/kylo/
   ↪kylo-services/conf/application.properties.0_8_3_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
   ```

   6.3 Copy the /bkup-config/TIMESTAMP/kylo-ui/application.properties file to */opt/kylo/kylo-ui/conf*

   6.4 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=
   ```

7. JMS configuration:

It was previously possible to provide ActiveMQ and AmazonSQS configuration in their respective configuration files called `activemq.properties` and `amazon-sqs.properties`. It is no longer possible and these properties should be moved over to standard Kylo configuration file found in `<KYLO_HOME>/kylo-services/conf/application.properties`.

8. **NOTE:** Kylo no longer ships with the default **dladmin** user. You will need to re-add this user only if you're using the default authentication configuration:

   - Uncomment the following line in `/opt/kylo/kylo-services/conf/application.properties` and `/opt/kylo/kylo-ui/conf/application.properties`:

     ```
     security.auth.file.users=file:///opt/kylo/users.properties
     security.auth.file.groups=file:///opt/kylo/groups.properties
     ```

   - Create a file called `users.properties` file that is owned by kylo and replace **dladmin** with a new username and **thinkbig** with a new password:

     ```
     echo "dladmin=thinkbig" > /opt/kylo/users.properties
     chown kylo:users /opt/kylo/users.properties
     chmod 600 /opt/kylo/users.properties
     ```

   - Create a file called `groups.properties` file that is owned by kylo and set the default groups:

     ```
     vi /opt/kylo/groups.properties
     ```

     ```
     dladmin=admin,user
     analyst=analyst,user
     designer=designer,user
     operator=operations,user
     ```

     ```
     chown kylo:users /opt/kylo/groups.properties
     chmod 600 /opt/kylo/groups.properties
     ```

9. 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 <NIFI_HOME> <KYLO_SETUP_FOLDER> <NIFI_
   →LINUX_USER> <NIFI_LINUX_GROUP>

   Example:  /opt/kylo/setup/nifi/update-nars-jars.sh /opt/nifi /opt/kylo/setup nifi␣
   →users
   ```

10. Start NiFi and Kylo

    ```
    service nifi start

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

    10.1 Once Kylo is up and running, refer back to step 5.2 to update the **Index Text Service** feed if using Elasticsearch v5.

4.2.2 Release 0.8.3.3 (October 16, 2017)

Highlights

- New configuration option added to the *auth-ad* security profile to control user details filtering (addresses Windows 365 issues)

- Fixes KYLO-1281 missing Kylo Upgrade Version

Download Links

- RPM : http://bit.ly/2yMUbjb

- Debian : http://bit.ly/2yrdL1o

- TAR : http://bit.ly/2ylM5NR

Upgrade Instructions from v0.8.3 & v0.8.3.1

1. Install the new RPM:

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

2. If using the `auth-ad` profile and having problems with accessing user info in AD (experienced by some Windows 365 deployments), add the following property to the existing AD properties in both kylo-services and kylo-ui application.properties files:

```
security.auth.ad.server.searchFilter=(&(objectClass=user)(sAMAccountName={1}
↪))
```

4.2.3 Release 0.8.3.2 (October 10, 2017)

Note: A later version, 0.8.3.3 exists that fixes an issue with this release. Please visit *Release 0.8.3.3 (October 16, 2017)* for the latest version

Highlights

- New configuration option added to the *auth-ad* security profile to control user details filtering (addresses Windows 365 issues)

Download Links

Please visit *Release 0.8.3.3 (October 16, 2017)* for download links

Upgrade Instructions from v0.8.3 & v0.8.3.1

Please visit *Release 0.8.3.3 (October 16, 2017)* for download links and install instructions

4.2.4 Release 0.8.3.1 (September 20, 2017)

Highlights

- Optimize feed creation in NiFi and improve NiFi usability when there is a large number of feeds

- Ability to skip NiFi auto alignment when saving feeds

- Fix bug in operations manager that didn't correctly fail jobs

- Support for 'failure connection' detection in feeds that contain sub process groups

- Fixes KYLO-823, KYLO-1202 setting controller service properties in feed/reusable templates

- Follow targetURL when logging in

- Fix Hive impersonation bug

- Additional metadata indexing to increase Kylo performance

Download Links

- RPM : http://bit.ly/2xgHsUM

- Debian : http://bit.ly/2hhqKOG

- TAR : http://bit.ly/2xT9ExY

Upgrade Instructions from v0.8.3

Build or download the RPM

1. Uninstall Kylo:

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

2. Install the new RPM:

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

3. Copy the application.properties file from the 0.8.2 install. If you have customized the application.properties file you will want to copy the 0.8.2 version and add the new properties that were added for this release.

    3.1 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:

    3.2 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_2_2_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
    ```

    3.3 Optional: If you want to skip the auto alignment after saving feeds then add in the new properties to the /opt/kylo/kylo-services/application.properties file

    ```
    ## skip auto alignment after you create a feed.
    ##You can always manually align your flows in NiFi via a
    →Kylo Rest Endpoint
    nifi.auto.align=false
    ```

    Optional: At startup Kylo inspects NiFi to build a cache of NiFi flow data. It now does this with multiple threads. By default it uses 10 threads. You can modify this by setting the following property:

```
## Modify the number of threads used by Kylo at startup to␣
↪inspect and build the NiFi flow cache.  Default is 10 if␣
↪not specified
nifi.flow.inspector.threads=10
```

3.4 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. 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 <NIFI_HOME> <KYLO_SETUP_FOLDER> <NIFI_
↪LINUX_USER> <NIFI_LINUX_GROUP>

Example:  /opt/kylo/setup/nifi/update-nars-jars.sh /opt/nifi /opt/kylo/setup nifi␣
↪users
```

5. After you startup you may need to re-index the Kylo metadata. You can do this via a REST endpoint after you login to Kylo at the following url:

http://localhost:8400/proxy/v1/metadata/debug/jcr-index/reindex

4.2.5 Release 0.8.3 (Aug 30, 2017)

Highlights

- Pluggable JMS implementation with out-of-the-box support for ActiveMQ and Amazon SQS. Refer to *JMS Providers* for details

- Pluggable REST client for *Elasticsearch*. This is now used by default in lieu of transport client.

- Cloudera Services Monitor as Kylo plugin. Refer to *Service Monitor Plugins* for details

- Business domain types for columns. Define rules to auto-apply domain types during feed creation or manually select the domain type to apply predefined standardization and validation rules.

- Column-level tagging. Apply tags to columns and search column tags using Global Search.

- Schema changes for column descriptions. The Hive schema is updated when modifying the column description of a feed. The column description is also available on the Visual Query page when hovering over a column name.

- Alerts improvement. User Interface enhancements and additional alerts capabilities. The Alerts page has been improved and the alerts on the dashboard are now in sync with the alerts page and adhere to entity access controls

- Category-level feed role memberships. Ability to manage feed access control of all feeds under a category by assigning feed role memberships at the category level

- Ability to query/filter Service Level Assessments against the Service Level Agreements

- IE & Safari browser support

- *Elasticsearch 5* support

- New angular UI module plugin support. Ability to create entirely new user interface modules and plug them into the UI navigation. Refer to

- Spark Jobserver processors for NiFi. Reuse a SparkContext between multiple Spark jobs for increased performance. Requires an existing .

- Pluggable Spark functions. Custom Spark functions can be added to the Visual Query page by providing a json file with the function definitions. Refer to *Writing Spark Function Definitions*.

- MS SQL support

- Maven Central support

Download Links

Visit the *Downloads* page for links.

Upgrade Instructions from v0.8.2

1. Stop 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 0.8.2 install. If you have customized the application.properties file you will want to copy the 0.8.2 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/conf/application.properties /opt/kylo/
    →kylo-services/conf/application.properties.0_8_3_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 new properties to the /opt/kylo/kylo-services/conf/application.properties file

    - The following properties allow Kylo to inspect the database schema when creating database feeds

    ```
    #Kylo MySQL controller service configuration
    nifi.service.kylo_mysql.database_user=root
    nifi.service.kylo_mysql.password=hadoop
    ```

- Flow Aggregation Stats

```
##when getting aggregate stats back for flows if errors are␣
↪detected kylo will query NiFi in attempt to capture matching␣
↪bulletins.
## by default this data is stored in memory.  Setting this to␣
↪true will store the data in the MySQL table
kylo.ops.mgr.stats.nifi.bulletins.persist=false
## if not perisiting (above flag is false) this is the limit to␣
↪the number of error bulletins per feed.
## this is a rolling queue that will keep the last # of errors␣
↪per feed
kylo.ops.mgr.stats.nifi.bulletins.mem.size=30
```

- New NiFi version 1.1 profile

  Previous versions of Kylo were compatible with Nifi v110 when using the nifiv1.0 profile. If you are using NiFi v1.1 in your environment then going forward you should use the nifi-1.1 profile.

```
spring.profiles.include=<other-profiles-as-required>,nifi-v1.1
```

- New configuration for JMS

  Previous versions of Kylo did not have a profile based method of configured the queue services. With new SQS support, the profile must be stated explicitly. See section 8 for more info.

```
spring.profiles.include=<other-profiles-as-required>,jms-
↪activemq
```

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. 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 <NIFI_HOME> <KYLO_SETUP_FOLDER> <NIFI_
↪LINUX_USER> <NIFI_LINUX_GROUP>

Example:  /opt/kylo/setup/nifi/update-nars-jars.sh /opt/nifi /opt/kylo/setup nifi␣
↪users
```

6. 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_8_2_backup.sql
```

7. 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 doesnt have priviledges to make schema changes to the kylo database. Please follow this *Database Upgrades* on how to manually apply the additional database updates.

8. Update NiFi to use default ActiveMQ JMS provider. Kylo now supports two JMS providers out-of-the-box: ActiveMQ and Amazon SQS. A particular provider is selected by active Spring profile in `/opt/nifi/ext-config/config.properties`.

   8.1. Edit `/opt/nifi/ext-config/config.properties`

   8.2. Add following line to enable ActiveMQ

   ```
   spring.profiles.active=jms-activemq
   ```

   Please follow this *JMS Providers* on how to switch active JMS Provider.

9. If using Elasticsearch as the search engine, go through steps 9.1 to 9.5. If using Solr, go to step 10 and also refer to *Solr plugin section*.

   9.1. Modify Elasticsearch rest client configuration (if required) in `/opt/kylo/kylo-services/conf/elasticsearch-rest.properties`. The defaults are provided below.

   ```
   search.rest.host=localhost
   search.rest.port=9200
   ```

   9.2. Verify `search-esr` profile in existing list of profiles in `/opt/kylo/kylo-services/conf/application.properties`

   ```
   spring.profiles.include=<other-profiles-as-required>,search-esr
   ```

   9.3. Create Kylo Indexes

   Execute a script to create kylo indexes. If these already exist, Elasticsearch will report an `index_already_exists_exception`. It is safe to ignore this and continue. Change the host and port if necessary.

   ```
   /opt/kylo/bin/create-kylo-indexes-es.sh localhost 9200 1 1
   ```

   9.4. Import updated Index Text Service feed. This step should be done once Kylo services are started and Kylo is up and running.

   9.4.1. **[Elasticsearch version 2]** Import the feed `index_text_service_elasticsearch.feed.zip` file available at `/opt/kylo/setup/data/feeds/nifi-1.0`

   9.4.2. **[Elasticsearch version 5] [This requires NiFi 1.3 or later]** Import the feed `index_text_service_v2.feed.zip` file available at `/opt/kylo/setup/data/feeds/nifi-1.3`

   9.5. For additional details, refer to *this document* under Rest Client section.

---

10. If using Solr as the search engine, go through steps 10.1 to 10.5. Also refer to *Solr plugin section*

    10.1. Create the collection in Solr

    ```
    bin/solr create -c kylo-datasources -s 1 -rf 1
    ```

    10.2. Navigate to Solr's

    10.3. Select the `kylo-datasources` collection from the drop down in the left nav area

    10.4. Click *Schema* on bottom left of nav area

    10.5. Click *Add Field* on top of right nav pane

    - name: *kylo_collection*

    - type: *string*

    - default value: *kylo-datasources*

    - index: *no*

    - store: *yes*

11. If Kerberos has been enabled in `spark.properties` then make the below edits and disable the kylo-spark-shell service. The service will be started as needed by kylo-services.

    ```
    # Changes for kylo-services/conf/spark.properties with Kylo 0.8.3
    #spark.shell.server.host = localhost
    #spark.shell.server.port = 8450
    spark.shell.deployMode = local
    ```

    ```
    # RedHat: disable kylo-spark-shell service
    chkconfig kylo-spark-shell off

    # Debian: disable kylo-spark-shell service
    update-rc.d kylo-spark-shell disable
    ```

12. Start NiFi and Kylo

    ```
    service nifi start

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

13. Migrate Hive schema indexing to Kylo. The indexing of Hive schemas is now handled internally by Kylo instead of using a special feed.

    12.1. Remove the Register Index processor from the `standard_ingest` and `data_transformation` reusable templates

    12.2. Delete the Index Schema Service feed

14. Import updated Index Text Service feed as mentioned in earlier step 9.4. At this point, Kylo should be up and running and hence 9.4 can be completed.

4.2.6 Release 0.8.2.6 (October 16, 2017)

Highlights

- New configuration option added to the *auth-ad* security profile to control user details filtering (addresses Windows 365 issues)

- Fixed KYLO-1264 ExecuteHQLStatement does not route to failure
- Fixed KYLO-940 ThriftConnectionPool doesn't reconnect on Hive restart
- Fixes KYLO-1281 missing Kylo Upgrade Version

Download Links

- RPM : http://bit.ly/2xK1Z0k
- Debian : http://bit.ly/2yqtlup
- TAR : http://bit.ly/2yn7y9c

Upgrade Instructions from v0.8.2

1. Uninstall Kylo:

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

2. Install the new RPM:

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

3. Copy the application.properties file from the 0.8.2 install. If you have customized the application.properties file you will want to copy the 0.8.2 version and add the new properties that were added for this release.

   3.1 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:

   3.2 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_2_2_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
   ```

   3.3 If using the `auth-ad` profile and having problems with accessing user info in AD (experienced by some Windows 365 deployments), add the following property to the existing AD properties in both kylo-services and kylo-ui application.properties files:

   ```
   security.auth.ad.server.searchFilter=(&
   →(objectClass=user)(sAMAccountName={1}))
   ```

4. 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 <NIFI_HOME> <KYLO_SETUP_FOLDER> <NIFI_
→LINUX_USER> <NIFI_LINUX_GROUP>

Example:  /opt/kylo/setup/nifi/update-nars-jars.sh /opt/nifi /opt/kylo/setup nifi
→users
```

5. Optional: To increase performance in Kylo you can choose to add indexes to the `metadata-repository.json` file. Add the following json snippet to the `/opt/kylo/kylo-services/conf/metadata-repository.json`

5.1 make a directory that kylo has read/write acess to:

```
mkdir -p /opt/kylo/modeshape/modeshape-local-index/
```

5.2. Edit the `/opt/kylo/kylo-services/conf/metadata-repository.json` and add in this snippet of JSON. Please ensure the "directory" in the json is the same that you created above.

```
"indexProviders": {
    "local": {
        "classname": "org.modeshape.jcr.index.local.
↪LocalIndexProvider",
        "directory": "/opt/kylo/modeshape/modeshape-
↪local-index/"
    }
    },
    "indexes": {
        "feedModificationDate": {
            "kind": "value",
            "provider": "local",
            "nodeType": "tba:feed",
            "columns": "jcr:lastModified(DATE)"
        },
        "feedState": {
            "kind": "value",
            "provider": "local",
            "nodeType": "tba:feedData",
            "columns": "tba:state(NAME)"
        },
        "categoryName": {
            "kind": "value",
            "provider": "local",
            "nodeType": "tba:category",
            "columns": "tba:systemName(STRING)"
        },
        "titleIndex": {
            "kind": "value",
            "provider": "local",
            "nodeType": "mix:title",
            "columns": "jcr:title(STRING)"
        },
        "nodesByName": {
            "kind": "value",
            "provider": "local",
            "synchronous": "true",
            "nodeType": "nt:base",
            "columns": "jcr:name(NAME)"
        },
        "nodesByDepth": {
            "kind": "value",
            "provider": "local",
            "synchronous": "true",
            "nodeType": "nt:base",
```

```
            "columns": "mode:depth(LONG)"
        },
        "nodesByPath": {
            "kind": "value",
            "provider": "local",
            "synchronous": "true",
            "nodeType": "nt:base",
            "columns": "jcr:path(PATH)"
        },
        "nodeTypes": {
            "kind": "nodeType",
            "provider": "local",
            "nodeType": "nt:base",
            "columns": "jcr:primaryType(STRING)"
        }
    },
```

Note: After you start you may need to re-index kylo. You can do this via a REST endpoint after you login to Kylo at the following url:

http://localhost:8400/proxy/v1/metadata/debug/jcr-index/reindex

4.2.7 Release 0.8.2.5 (October 11, 2017)

Note: A later version, 0.8.2.6 exists that fixes an issue with this release. Please visit *Release 0.8.2.6 (October 16, 2017)* for the latest version

Highlights

- New configuration option added to the *auth-ad* security profile to control user details filtering (addresses Windows 365 issues)
- Fixed KYLO-1264 ExecuteHQLStatement does not route to failure
- Fixed KYLO-940 ThriftConnectionPool doesn't reconnect on Hive restart

Download Links

Please visit *Release 0.8.2.6 (October 16, 2017)* for download links

Upgrade Instructions from v0.8.2

Please visit *Release 0.8.2.6 (October 16, 2017)* for download links and install instructions

4.2.8 Release 0.8.2.4 (September 18, 2017)

Highlights

- Fixes KYLO-1214 Feed Lineage
- Refer to previous 0.8.2.x releases for additional notes.

Additional Features in the 0.8.2.x Patch Releases

- Optimize feed creation in NiFi and improve NiFi usability when there is a large number of feeds
- Ability to skip NiFi auto alignment when saving feeds
- Fix bug in operations manager that didn't correctly fail jobs
- Support for 'failure connection' detection in feeds that contain sub process groups
- Fixes KYLO-823, KYLO-1202 setting controller service properties in feed/reusable templates

Download Links

- RPM : http://bit.ly/2xeDCcx
- Debian : http://bit.ly/2hfIiHm
- TAR : http://bit.ly/2f81QNv

Upgrade Instructions from v0.8.2

Build or download the rpm

1. Uninstall Kylo:

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

2. Install the new RPM:

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

3. Copy the application.properties file from the 0.8.2 install. If you have customized the application.properties file you will want to copy the 0.8.2 version and add the new properties that were added for this release.

   3.1 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:

   3.2 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_2_2_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
```

   3.3 Optional: If you want to skip the auto alignment after saving feeds then add in the new properties to the /opt/kylo/kylo-services/application.properties file

```
## skip auto alignment after you create a feed.
##You can always manually align your flows in NiFi via a
↪Kylo Rest Endpoint
nifi.auto.align=false
```

Optional: At startup Kylo inspects NiFi to build a cache of NiFi flow data. It now does this with multiple threads. By default it uses 10 threads. You can modify this by setting the following property:

```
## Modify the number of threads used by Kylo at startup to␣
→inspect and build the NiFi flow cache.  Default is 10 if␣
→not specified
nifi.flow.inspector.threads=10
```

3.4 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. 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 <NIFI_HOME> <KYLO_SETUP_FOLDER> <NIFI_
→LINUX_USER> <NIFI_LINUX_GROUP>

Example:  /opt/kylo/setup/nifi/update-nars-jars.sh /opt/nifi /opt/kylo/setup nifi␣
→users
```

5. Optional: To increase performance in Kylo you can choose to add indexes to the `metadata-repository.json` file. Add the following json snippet to the `/opt/kylo/kylo-services/conf/metadata-repository.json`

5.1 make a directory that kylo has read/write acess to:

```
mkdir -p /opt/kylo/modeshape/modeshape-local-index/
```

5.2. Edit the `/opt/kylo/kylo-services/conf/metadata-repository.json` and add in this snippet of JSON. Please ensure the "directory" in the json is the same that you created above.

```
"indexProviders": {
    "local": {
        "classname": "org.modeshape.jcr.index.local.
→LocalIndexProvider",
        "directory": "/opt/kylo/modeshape/modeshape-
→local-index/"
    }
    },
    "indexes": {
        "feedModificationDate": {
            "kind": "value",
            "provider": "local",
            "nodeType": "tba:feed",
            "columns": "jcr:lastModified(DATE)"
        },
        "feedState": {
            "kind": "value",
            "provider": "local",
            "nodeType": "tba:feedData",
            "columns": "tba:state(NAME)"
```

```
            },
            "categoryName": {
                "kind": "value",
                "provider": "local",
                "nodeType": "tba:category",
                "columns": "tba:systemName(STRING)"
            },
            "titleIndex": {
                "kind": "value",
                "provider": "local",
                "nodeType": "mix:title",
                "columns": "jcr:title(STRING)"
            },
            "nodesByName": {
                "kind": "value",
                "provider": "local",
                "synchronous": "true",
                "nodeType": "nt:base",
                "columns": "jcr:name(NAME)"
            },
            "nodesByDepth": {
                "kind": "value",
                "provider": "local",
                "synchronous": "true",
                "nodeType": "nt:base",
                "columns": "mode:depth(LONG)"
            },
            "nodesByPath": {
                "kind": "value",
                "provider": "local",
                "synchronous": "true",
                "nodeType": "nt:base",
                "columns": "jcr:path(PATH)"
            },
            "nodeTypes": {
                "kind": "nodeType",
                "provider": "local",
                "nodeType": "nt:base",
                "columns": "jcr:primaryType(STRING)"
            }
        },
```

Note: After you start you may need to re-index kylo. You can do this via a REST endpoint after you login to Kylo at the following url:

http://localhost:8400/proxy/v1/metadata/debug/jcr-index/reindex

4.2.9 Release 0.8.2.3 (September 15, 2017)

Highlights

- This releases further optimizes Kylo and NiFi integration.

- Fixes KYLO-823, KYLO-1202 setting controller service properties in feed/reusable templates

- Reduces the verbose logging output (in 0.8.2.2) to debug when creating feeds

Download Links

- RPM : http://bit.ly/2x7BB3q

- Debian : http://bit.ly/2wuQgSA

- TAR : http://bit.ly/2h81kiK

Upgrade Instructions from v0.8.2

Build or download the rpm

1. Uninstall Kylo:

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

2. Install the new RPM:

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

3. Copy the application.properties file from the previous install If you have customized the application.properties file you will want to copy the 0.8.2 version and add the new properties that were added for this release.

   3.1 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:

   3.2 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_2_2_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
```

   3.3 Optional: At startup Kylo inspects NiFi to build a cache of NiFi flow data. It now does this with multiple threads. By default it uses 10 threads. You can modify this by setting the following property:

```
## Modify the number of threads used by Kylo at startup to inspect␣
↪and build the NiFi flow cache.  Default is 10 if not specified
nifi.flow.inspector.threads=10
```

   3.4 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. 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 <NIFI_HOME> <KYLO_SETUP_FOLDER> <NIFI_
→LINUX_USER> <NIFI_LINUX_GROUP>

Example:  /opt/kylo/setup/nifi/update-nars-jars.sh /opt/nifi /opt/kylo/setup nifi␣
→users
```

5. Optional: To increase performance in Kylo you can choose to add indexes to the `metadata-repository.` `json` file.   Add the following json snippet to the `/opt/kylo/kylo-services/conf/` `metadata-repository.json`

    5.1 make a directory that kylo has read/write acess to:

    ```
    mkdir -p /opt/kylo/modeshape/modeshape-local-index/
    ```

    5.2. Edit the `/opt/kylo/kylo-services/conf/metadata-repository.json` and add in this snippet of JSON. Please ensure the "directory" in the json is the same that you created above.

    ```
    "indexProviders": {
        "local": {
            "classname": "org.modeshape.jcr.index.local.
    →LocalIndexProvider",
            "directory": "/opt/kylo/modeshape/modeshape-
    →local-index/"
        }
        },
        "indexes": {
            "feedModificationDate": {
                "kind": "value",
                "provider": "local",
                "nodeType": "tba:feed",
                "columns": "jcr:lastModified(DATE)"
            },
            "feedState": {
                "kind": "value",
                "provider": "local",
                "nodeType": "tba:feedData",
                "columns": "tba:state(NAME)"
            },
            "categoryName": {
                "kind": "value",
                "provider": "local",
                "nodeType": "tba:category",
                "columns": "tba:systemName(STRING)"
            },
            "titleIndex": {
                "kind": "value",
                "provider": "local",
                "nodeType": "mix:title",
                "columns": "jcr:title(STRING)"
            },
            "nodesByName": {
                "kind": "value",
                "provider": "local",
                "synchronous": "true",
                "nodeType": "nt:base",
                "columns": "jcr:name(NAME)"
            },
    ```

```
        "nodesByDepth": {
            "kind": "value",
            "provider": "local",
            "synchronous": "true",
            "nodeType": "nt:base",
            "columns": "mode:depth(LONG)"
        },
        "nodesByPath": {
            "kind": "value",
            "provider": "local",
            "synchronous": "true",
            "nodeType": "nt:base",
            "columns": "jcr:path(PATH)"
        },
        "nodeTypes": {
            "kind": "nodeType",
            "provider": "local",
            "nodeType": "nt:base",
            "columns": "jcr:primaryType(STRING)"
        }
    },
```

Note: After you start you may need to re-index kylo. You can do this via a REST endpoint after you login to Kylo at the following url:

> http://localhost:8400/proxy/v1/metadata/debug/jcr-index/reindex

4.2.10 Release 0.8.2.2 (September 12, 2017)

Highlights

- Optimize feed creation in NiFi and improve NiFi usability when there is a large number of feeds

- Ability to skip NiFi auto alignment when saving feeds

- Fix bug in operations manager that didn't correctly fail jobs

- Support for 'failure connection' detection in feeds that contain sub process groups

- For a complete list of issues resolved visit: ReleaseNotes8.2.2.issues

Download Links

- RPM : http://bit.ly/2fhpVSq

- Debian : http://bit.ly/2eUBtKA

- TAR : http://bit.ly/2x0g7FD

Upgrade Instructions from v0.8.2

Build or download the rpm

1. Uninstall Kylo:

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

2. Install the new RPM:

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

3. Copy the application.properties file from the 0.8.2 install. If you have customized the application.properties file you will want to copy the 0.8.2 version and add the new properties that were added for this release.

   3.1 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:

   3.2 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_2_2_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
```

   3.3 Optional: If you want to skip the auto alignment after saving feeds then add in the new properties to the /opt/kylo/kylo-services/application.properties file

```
## skip auto alignment after you create a feed.
##You can always manually align your flows in NiFi via a Kylo Rest
→Endpoint
nifi.auto.align=false
```

   3.4 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. 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 <NIFI_HOME> <KYLO_SETUP_FOLDER> <NIFI_
→LINUX_USER> <NIFI_LINUX_GROUP>

Example:  /opt/kylo/setup/nifi/update-nars-jars.sh /opt/nifi /opt/kylo/setup nifi
→users
```

5. Optional: To increase performance in Kylo you can choose to add indexes to the `metadata-repository.json` file. Add the following json snippet to the `/opt/kylo/kylo-services/conf/metadata-repository.json`

   5.1 make a directory that kylo has read/write acess to:

```
mkdir -p /opt/kylo/modeshape/modeshape-local-index/
```

5.2. Edit the `/opt/kylo/kylo-services/conf/metadata-repository.json`
and add in this snippet of JSON. Please ensure the "directory" in the json is the same that you
created above.

```
"indexProviders": {
    "local": {
        "classname": "org.modeshape.jcr.index.local.
→LocalIndexProvider",
        "directory": "/opt/kylo/modeshape/modeshape-
→local-index/"
    }
    },
    "indexes": {
        "feedModificationDate": {
            "kind": "value",
            "provider": "local",
            "nodeType": "tba:feed",
            "columns": "jcr:lastModified(DATE)"
        },
        "feedState": {
            "kind": "value",
            "provider": "local",
            "nodeType": "tba:feedData",
            "columns": "tba:state(NAME)"
        },
        "categoryName": {
            "kind": "value",
            "provider": "local",
            "nodeType": "tba:category",
            "columns": "tba:systemName(STRING)"
        },
        "titleIndex": {
            "kind": "value",
            "provider": "local",
            "nodeType": "mix:title",
            "columns": "jcr:title(STRING)"
        },
        "nodesByName": {
            "kind": "value",
            "provider": "local",
            "synchronous": "true",
            "nodeType": "nt:base",
            "columns": "jcr:name(NAME)"
        },
        "nodesByDepth": {
            "kind": "value",
            "provider": "local",
            "synchronous": "true",
            "nodeType": "nt:base",
            "columns": "mode:depth(LONG)"
        },
        "nodesByPath": {
            "kind": "value",
            "provider": "local",
            "synchronous": "true",
            "nodeType": "nt:base",
            "columns": "jcr:path(PATH)"
        },
```

```
        "nodeTypes": {
            "kind": "nodeType",
            "provider": "local",
            "nodeType": "nt:base",
            "columns": "jcr:primaryType(STRING)"
        }
    },
```

Note: After you start you may need to re-index kylo. You can do this via a REST endpoint after you login to Kylo at the following url:

http://localhost:8400/proxy/v1/metadata/debug/jcr-index/reindex

4.2.11 Release 0.8.2 (July 12, 2017)

Highlights

- 109 issues resolved

- NiFi 1.3.0 support

- Global search enhancements. Auto-indexing of feed, category, tags, and improved UI.

- Ability to use *Solr vs ElasticSearch* (ES default)

- Streaming visualization improvements. New Ops UI for monitoring streaming feeds.

- Provenance event performance tune-up. Fixed lag that could occur for high throughput streaming feeds.

- Pluggable UI. Ability to add dynamic new user interface components. See: *Plugin APIs*

- Wrangler support for Spark yarn-cluster mode

- Wrangler supports *user impersonation*. There are a few different run modes depending on which *configuration properties* are specified.

- TAR file installation support. This allows installation in different folder locations and to be ran as different linux users/groups

- Example S3 data ingest template. Ability to process data without bringing the data into the edge node. See: *S3 Standard Ingest Template*

- SPNEGO bug fixes and improvements with Active Directory integration

Download Links

- RPM : http://bit.ly/2uT8bTo

- Debian : http://bit.ly/2uSTvUv

- TAR : http://bit.ly/2ug8ZUz

Upgrade Instructions from v0.8.1

Build or download the RPM

1. Uninstall Kylo:

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

2. Install the new RPM:

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

3. Update NiFi to use the new KyloPersistentProvenanceRepository. Kylo no longer uses the NiFi reporting task to capture provenance events. Instead it uses a modified ProvenanceRepository.

3.1. In NiFi stop and delete the Kylo Reporting Task and its associated Controller Service.

3.2. Stop NiFi

3.3. Follow the guide *NiFi & Kylo Provenance* to setup the KyloPersistentProvenanceRepository

4. Copy the application.properties file from the 0.8.1 install. If you have customized the application.properties file you will want to copy the 0.8.1 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_2_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 new properties to the /opt/kylo/kylo-services/application.properties file

- ActiveMQ properties: Redelivery processing properties are now available for configuration. If Kylo receives provenance events and they have errors are are unable to attach NiFi feed information (i.e. if NiFi goes down and Kylo doesnt have the feed information in its cache) then the JMS message will be returned for redelivery based upon the following parameters. Refer to the ActiveMQ documentation, http://activemq.apache.org/redelivery-policy.html, for assigning these values:

```
## retry for xx times before sending to DLQ (Dead Letter Queue)␣
↪set -1 for unlimited redeliveries
jms.maximumRedeliveries=100
##The initial redelivery delay in milliseconds.
jms.initialRedeliveryDelay=1000
##retry every xx seconds
jms.redeliveryDelay=5000
##Sets the maximum delivery delay that will be applied if the␣
↪useExponentialBackOff option is set (use value -1 for no max)
jms.maximumRedeliveryDelay=600000
##The back-off multiplier.
jms.backOffMultiplier=5
##Should exponential back-off be used, i.e., to exponentially␣
↪increase the timeout.
jms.useExponentialBackOff=false
```

- **NiFi 1.3 support** If you are using NiFi 1.2 or 1.3 you need to update the spring profile to point to the correct nifi version.

  Example NiFi 1.2 or 1.3 support

  ```
  ### Indicate the NiFi version you are using with the correct␣
  →spring profile.
  ###  - For NiFi 1.0.x or 1.1.x:    nifi-v1
  ###  - For NiFi 1.2.x or 1.3.x:    nifi-v1.2
  spring.profiles.include=native,nifi-v1.2,auth-kylo,auth-file
  ```

  Example NiFi 1.0 or 1.1 support

  ```
  spring.profiles.include=native,nifi-v1,auth-kylo,auth-file
  ```

- **Global search support** Elasticsearch is the default search provider. Add `search-es` to spring profiles:

  ```
  spring.profiles.include=<all your existing profiles>,search-es
  ```

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 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_8_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 doesnt have priviledges to make schema changes to the kylo database. Please follow this *Database Upgrades* on how to manually apply the additional database updates.

7. 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 <NIFI_HOME> <KYLO_SETUP_FOLDER> <NIFI_
→LINUX_USER> <NIFI_LINUX_GROUP>

Example:  /opt/kylo/setup/nifi/update-nars-jars.sh /opt/nifi /opt/kylo/setup nifi␣
→users
```

8. **Update configuration for using Elasticsearch as the search engine**

   (a) **Provide cluster properties**

      i. Update cluster properties in `/opt/kylo/kylo-services/conf/elasticsearch.properties` if different from the defaults provided below.

      ```
      search.host=localhost
      search.clusterName=demo-cluster
      search.restPort=9200
      search.transportPort=9300
      ```

      Kylo services must be restarted if the above file has been changed to pick up the new values.

      ```
      service kylo-services restart
      ```

   (b) **Steps to import updated Index Schema Service feed**

      i. Feed Manager -> Feeds -> + orange button -> Import from file -> Choose file

      ii. Pick the `index_schema_service_elasticsearch.feed.zip` file available at `/opt/kylo/setup/data/feeds/nifi-1.0`

      iii. Leave *Change the Category* field blank (It defaults to *System*)

      iv. Click *Yes* for these two options (1) *Overwrite Feed* (2) *Replace Feed Template*

      v. (optional) Click *Yes* for option (3) *Disable Feed upon import* only if you wish to keep the indexing feed disabled upon import (You can explicitly enable it later if required)

      vi. Click *Import Feed*.

      vii. Verify that the feed imports successfully.

      viii. If your Hive metastore is in a schema named something other than `hive`, edit the feed and set `hive.schema` to the schema name. This configuration value may be available with the key `config.hive.schema` in `/opt/kylo/kylo-services/conf/application.properties`

   (c) **Steps to import updated Index Text Service feed**

      i. Feed Manager -> Feeds -> + orange button -> Import from file -> Choose file

      ii. Pick the `index_text_service_elasticsearch.feed.zip` file available at `/opt/kylo/setup/data/feeds/nifi-1.0`

      iii. Leave *Change the Category* field blank (It defaults to *System*)

      iv. Click *Yes* for these two options (1) *Overwrite Feed* (2) *Replace Feed Template*

      v. (optional) Click *Yes* for option (3) *Disable Feed upon import* only if you wish to keep the indexing feed disabled upon import (You can explicitly enable it later if required)

      vi. Click *Import Feed*.

      vii. Verify that the feed imports successfully.

9. Re-import the templates.

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

   • Re-import Data Transformation template (data_transformation.zip)

   • Re-import Data Confidence template (data_confidence_invalid_records.zip)

10. NiFi 1.2/1.3 breaking change.

- NiFi introduced a change to their UpdateAttributes processor that prevents empty strings from being set to the dynamic properties unless the state is saved.

- The templates (in step 7 above) already have this change made. Any feeds you have from a previous NiFi version that have empty strings in the UpdateAttributes processors will be broken and need fixed. You can fix them by importing the new templates and then saving the feed, or you will need to manually fix the feed/template. If you need to manually fix feed flows in NiFi do the following:

    1. Modify the UpdateAttributes processors and change the "Store State" property to be "Store state locally"

    2. Change the "Stateful Variables Initial Value" and check the box "Set empty string"

    3. Go to the Settings for the processor and Auto terminate the "set state fail" route.

1

Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	COME

Required field

Property		Value
Delete Attributes Expression	❓	No valu
Store State	❓	**Store s**
Stateful Variables Initial Value	❓	Empty
feedts	❓	${now(
hdfs.ingest.root	❓	${hdfs.
hive.ingest.root	❓	${hive.i

2

Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	C

Required field

Property		Value
Delete Attributes Expression	❓	No v

Store State		
Stateful Variables Initial Value		

4.2.12 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 *Clustering Kylo* 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 kylp 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 doesnt have priviledges to make schema changes to the kylo database. Please follow this *Database Upgrades* on how to manually apply the additional database updates.

7. 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
```

8. 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)

4.2.13 Release 0.8.0 (Apr 19, 2017)

Highlights

- 90+ issues resolved

- Automatic and manual database upgrades. See *Database Upgrades*

- 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 applica-tion.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 kylp 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 doesnt have priviledges to make schema changes to the kylo database. Please follow this *Database Upgrades* 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/
```

4.2.14 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
```

4.2.15 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
```

4.2.16 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:

../how-to-guides/NiFiKyloProvenanceReportingTask

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.

4.2.17 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:

../how-to-guides/NiFiKyloProvenanceReportingTask

4.2.18 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 the 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 where 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:

../how-to-guides/NiFiKyloProvenanceReportingTask

4.2.19 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

4.2.20 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, PAR-QUET).
- 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.

4.2.21 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 *Encrypting Configuration Properties*
- 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.

4.2.22 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.

4.2.23 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 havent done so, you need to ensure the latest nifi-provenance-repo.nar file is in the /opt/nifi/data/lib folder.

4.2.24 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

4.2.25 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"

4.2.26 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

4.2.27 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/

Downloads

This page contains links to the commons files you might want to download

5.1 Latest Kylo Distribution (0.8.4.1)

Type	Link
RPM	http://bit.ly/2DrO9Ur
DEB	http://bit.ly/2DrLXfz
TAR	http://bit.ly/2kEwFfU

5.2 Kylo Distribution (0.8.3.3)

Type	Link
RPM	http://bit.ly/2yMUbjb
DEB	http://bit.ly/2yrdL1o
TAR	http://bit.ly/2ylM5NR

5.3 Kylo Distribution (0.8.3)

Type	Link
RPM	http://bit.ly/2xOA8wd
DEB	http://bit.ly/2gkYmr1
TAR	http://bit.ly/2wk1kVH

5.4 Plugins

Plugins can be downloaded from the maven central repository https://search.maven.org/

Overview

The best way to get started with Kylo is to keep it simple at first. Get Kylo up and running with a single node and test a simple feed before enabling features such as clustering, SSL, encryption,etc. This installation section will help you get Kylo up and running, then give you some guidance on where to go from there.

6.1 Installation Methods

Kylo has 3 build distributions:

- **RPM** - An easy and opinionated way of installing Kylo on Redhat based systems

- **DEB** - An easy and opinionated way of installing Kylo on Debian based systems

- **TAR File** – Available for those who want to install Kylo in a folder other than /opt/kylo, or want to run Kylo as a different user.

Once the binary is installed Kylo can be configured a few different ways:

- **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.

- **Manually Run Shell Scripts** - 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.

- **Configuration Management Tools** – Kylo installation is designed to be automated. You can leverage tools such as Ansible, Chef, Puppet, and Salt Stack

6.2 Installation Components

Installing Kylo inlcudes 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/SOLR**: Kylo can use either Elasticsearch or SOLR, as a distributed, multi-tenant capable full-text search engine.

6.3 Default 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

6.4 Demo Sandbox

If you are interested in running a working example of Kylo you might consider running one of our demo sandboxes located on the http://kylo.io/quickstart.html website

Review Dependencies

This page can be used as a guide to prepare you environment for installation.

7.1 Supported Operating Systems

Operating System	Versions
RHEL,CentOs	6.x, 7.x
SUSE	v11
Ubuntu	16.x,17.x

7.2 Supported Hadoop Distributions

Platform	Sandbox URL	Version
Hortonworks	https://hortonworks.com/products/sandbox/	HDP 2.3+
Cloudera	https://www.cloudera.com/downloads/quickstart_vms/5-12.html	5.8+

7.3 Edge Node Hardware Requirements

Although the hardware requirements depend on the volume of data that will be processed here are some general recommendations:

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

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

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

7.4 Kylo Stack Dependencies

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

Category	Item	Version	Description
Persistence	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
Persistence	Postgres	9.x	Used to store both the Modeshape (JCR 2.0) metadata and the Operational Relational (Kylo Ops Manager) metadata
Persistence	MS SQL Server	Azure	Used to store both the Modeshape (JCR 2.0) metadata and the Operational Relational (Kylo Ops Manager) metadata
JMS	ActiveMq	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 - 1.3,(HDF 2.0)	Either HDF or open source NiFi work.
Spark	Spark Client	1.5.x, 1.6.x, 2.x	NiFi and Kylo have routines that leverage Spark.
Hive	Hive	1.2.x+	Required if using Hive and the standard ingest template
Hadoop	HDFS	2.7.x+	Required if using Hive and the standard ingest template
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	Elasticsearch	2.3.x, 5.x	For index and search of Hive metadata and indexing feed data when selected as part of creating a feed
Search	Solr	6.5.1 (SolrCloud mode)	For index and search of Hive metadata and indexing feed data when selected as part of creating a feed

7.5 Linux Tools

Below are tools required to be installed on the Linux box before installing the Kylo components

Tool
Curl (for downloading installation files)
RPM or dpkg(for install)

7.6 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 API Server	kylo	kylo, hdfs or super-group	/etc/security/keytabs/kylo.service.keytab	kylo...@EXAMPLE.COM*	
kylo-ui	Provides Kylo feed and operations user interface	kylo	kylo, hdfs or super-group			
nifi	Orchestrate data flows	nifi	nifi, hdfs or supergroup	/etc/security/keytabs/nifi.service.keytab	nifi...@EXAMPLE.COM*	
activemq	Broker messages between components	activemq	activemq			
elastic-search	Manages searchable index	elastic-search	elastic-search			
mysql or postgres	Metastore for Kylo feed manager and operational metadata	mysql or postgres	mysql or postgres			

Note: You have the flexibility to change the installation locations and service accounts when using the TAR installation method

7.7 Network Ports

Kylo relies heavily on integration with other services. Below is a list of network ports that are required for the standard ingest to work

Required

Port	From Service	To Service
8400	Browser/NiFi	kylo-ui
8079	Browser/kylo-services	NiFi
61616	kylo-services/NiFi	ActiveMQ
3306	kylo-services/NiFi	MySQL
9200	kylo-services/NiFi	Elasticsearch
9300	kylo-services/NiFi	Elasticsearch 2.x
8983	kylo-services/NiFi	SOLR
9983	kylo-services/NiFi	SOLR
10000	kylo-services/NiFi	HiveServer2
ALL	kylo-spark-shell	Yarn, data nodes

Optional

Port	From Service	To Service
8420	REST Client	kylo-services
8161	Browser	ActiveMQ Admin

7.8 Default HDFS Locations (for standard ingest)

The below locations are configurable. If you plan on using the default locations they will be create here.

HDFS Location	Description
/archive	Archive original files
/etl	Feed processing file location
/model.db	Hive feed, invalid, valid, profile location
/app/warehouse	Hive feed table final location

Prepare Install Checklist

This checklist will help you prepare for an enterprise deployment and is valuable if you require approvals ahead of time. Please refer to the *Review Dependencies* guide for more details in each section

- **Pre-installation**
    - [ ] Determine data throughput requirements based on expected feeds
    - [ ] Will I use an existing Elasticsearch/SOLR instance?
    - [ ] Will I use an existing ActiveMQ instance?
    - [ ] Review library dependencies to ensure HDFS/Hive/Spark is current enough
    - [ ] Obtain approvals for Linux service users (If not, you must install using TAR method)
    - [ ] Obtain approvals for network ports
    - [ ] Determine if I want to leverage liquibase to automatically install database scripts and upgrades for Kylo
    - [ ] Request or generate SSL certificates if required
- **Hardware/OS Provisioning**
    - [ ] Provision Edge Nodes
    - [ ] Install supported operating system
- **General Configuration Preparation**
    - [ ] Hive Hostname/IP Address:
    - [ ] Ambari/Cloudera Manager IP Hostname/IP Address (if used):
    - [ ] Ambari/Cloudera Manager "kylo" user username/password (if used):
    - [ ] Kylo Edge Hostname/IP Address:
    - [ ] NiFi Edge Hostname/IP Address:
    - [ ] MySQL Kylo 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):

- [ ] MySQL Hive Hostname/IP Address:

- [ ] Hive MySQL application username/password:

- [ ] HDFS root folder location (if different than default:

- **Kerberos Configuration Preparation**

    - [ ] KDC Hostname/IP Address (if used):

    - [ ] Kerberos Principal for "kylo":

    - [ ] Kerberos Principal for "nifi":

    - [ ] Kerberos Principal for "hive" on the Hive Server2 Host:

Create Service Accounts

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.

9.1 Option 1: Install all users/groups on single node

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

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

9.2 Option 2: Run individual useradd commands

If you are installing the Kylo components on different nodes you will need to run the commands individually. To create individual users, run the following commands on the appropriate machines:

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

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

```
grep 'nifi\|kylo\|activemq' /etc/group /etc/passwd
```

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:

```
groupadd -f kylo
groupadd -f nifi
groupadd -f activemq
```

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

```
groupadd -f kylo && groupadd -f nifi && groupadd -f activemq
```

Prepare Offline TAR

The OPTIONAL offline TAR file can be useful in two scenarios:

1. You are installing ActiveMQ, Elasticsearch, Java, or NiFi on nodes with no external network access.

2. You plan on installing ActiveMQ, Elasticsearch, Java, or NiFi on separate nodes than Kylo and want to take advantage of the setup files you will want to generate an

The offline TAR file will include the binaries required to install the 4 services mentioned above.

10.1 Generate the TAR file

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

```
$ rpm -ivh kylo-<version>.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. You will need to reference the setup location when manually running the shell scripts

Install Kylo

Choose one of the installation methods below to install Kylo.

11.1 RPM Install

Download the latest RPM (*Downloads*) , 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.

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

11.2 DEB Install

Download the latest DEB file (*Downloads*) , 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.

```
$ dpkg -i kylo-<version>.deb
```

11.3 TAR File Install

The TAR file method is useful when you need more control over where you can install Kylo and you need the flexibility to run Kylo as a different service user. In this example we will assume you want to install Kylo in the /apps folder, run

it as the "ad_kylo" user and "users" group

1. Download the latest TAR (*Downloads*) , and place it on the host Linux machine that you want to install Kylo services on.

2. **Untar the file**

```
$ sudo mkdir /apps/kylo
$ sudo tar -xvf /tmp/kylo-<version>-dependencies.tar.gz -C /apps/kylo
```

3. Run the post-install script

```
$ sudo /apps/kylo/setup/install/post-install.sh /apps/kylo ad_kylo users
```

11.4 TAR File Upgrade

If you are performing an upgrade please see the TAR file upgrade page for instructions

../installation/TarFileUpgrade

Install Additional Components

Now that Kylo has been installed you have a few different option to install the database scripts, ActiveMQ, Elasticsearch, Java, and NiFi

Note: The setup wizard currently doesnt autodetect that its on a SUSE. Therefore you should skip the Elasticsearch installation step and download/install the DEB distribution manually.

12.1 Database Preparation

If you would like to run Kylo as a non-privileged user you should create a kylo database user and configure the appropriate permissions.

12.1.1 Create Kylo Database and User

If you prefer to run Kylo as a non-privileged user and want to create the database schema yourself please do the following.

Note: These commands need to be ran as a database administrator

Create the kylo database

This must be done as a database administrator

Postgres

```
$ sudo -u postgres psql

> CREATE DATABASE kylo;
```

Create the kylo database user

Postgres

```
$ sudo -u postgres psql

> CREATE USER kylo WITH PASSWORD 'abc123';
```

Grant Kylo user access to DB

Postgres

```
$ sudo -u postgres psql -d kylo

> grant usage on schema public to kylo;
> GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA PUBLIC TO kylo;
> GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA PUBLIC TO kylo;
> grant execute on all functions in schema public to kylo;
> alter default privileges in schema public grant execute on functions to kylo;
```

If you plan to generate and run the SQL scripts manually (turn off liquibase), please see the "Manual Upgrades" section in *Database Upgrades*

12.2 Option 1: Setup Wizard Installation

This is the easiest method and will allow you to choose which components to install on that node.

12.2.1 Setup Wizard Deployment Guide

Note that you will need a database user with schema create privileges if allowing the setup wizard to create the database. If you prefer to create the "kylo" database yourself and/or create a "kylo" user please refer to *Create Kylo Database and User* first

Step 1: Run the Setup Wizard

> **Warning:** If Java 8 is not the system Java 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 and ActiveMQ services start when the wizard is finished.

12.3 Option 2: Manual Installation

This option shows you how to run the scripts manually and will allow you to make customizations as you go.

12.3.1 Manual Deployment Guide

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.

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

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
```

You might install some of these components on a differnet edge node than where Kylo is installed. In this case, copy the setup folder or offline TAR file to those nodes that do not have the Kylo applications installed. In that case, use this SETUP_DIR command:

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

Step 2: Create the "dladmin" user

Before logging into Kylo for the first time you must create a password for the "dladmin" user. To created the password please do the following:

1. Create a users.properties file and add the username/password

---

```
$ vi /opt/kylo/users.properties
    dladmin=myPassword
```

2. Modify the /opt/kylo/kylo-ui/conf/application.properties file

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

    # uncomment this line
    security.auth.file.users=file:///opt/kylo/users.properties
```

3. Modify the /opt/kylo/kylo-services/conf/application.properties file

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

    # uncomment this line
    security.auth.file.users=file:///opt/kylo/users.properties
```

Please see *Configure Access Control* for information about configuring users and groups

Step 3: 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 unstall Java 8:

Online Mode

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

Offline Mode

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

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

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 <KYLO_HOME>
$ /opt/kylo/setup/java/change-kylo-java-home.sh <JAVA_HOME> <KYLO_HOME>
```

Step 4: 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 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 <KYLO_SETUP_FOLDER> <JAVA_8_HOME>
```

2. Offline Mode

```
$ <OFFLINE_SETUP_DIR>/elasticsearch/install-elasticsearch.sh <OFFLINE_SETUP_DIR>
→<JAVA_8_HOME> -o

  Example:  /tmp/kylo-install/setup/elasticsearch/install-elasticsearch.sh /tmp/kylo-
→install/setup /opt/java/current -o
```

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.

- Execute a script to create kylo indexes. If these already exist, Elasticsearch will report an `index_already_exists_exception`. It is safe to ignore this and continue. Change the host and port if necessary.

```
/opt/kylo/bin/create-kylo-indexes-es.sh localhost 9200 1 1
```

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 <INSTALLATION_FOLDER> <LINUX_USER> <LINUX_
↪GROUP> <JAVA_8_HOME>
```

2. Offline Mode

```
 $ <OFFLINE_SETUP_DIR>/activemq/install-activemq.sh <INSTALLATION_FOLDER> <LINUX_USER>
↪ <LINUX_GROUP> <JAVA_8_HOME> <OFFLINE_SETUP_DIR> -o

Example: /tmp/kylo-install/setup/activemq/install-activemq.sh /opt/activemq activemq
↪activemq /opt/java/current /tmp/kylo-install/setup -o
```

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

     jms.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
```

Step 7: Install NiFi

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

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 <NIFI_BASE_FOLDER> <NIFI_LINUX_USER> <NIFI_LINUX_
↪GROUP>
```

   Offline Mode

```
$ <OFFLINE_SETUP_DIR>/nifi/install-nifi.sh  <NIFI_BASE_FOLDER> <NIFI_LINUX_USER>
↪<NIFI_LINUX_GROUP> <OFFLINE_SETUP_DIR> -o
```

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

```
$ <SETUP_DIR>/java/change-nifi-java-home.sh <JAVA_HOME> <NIFI_BASE_FOLDER>/current
```

3. Install Kylo specific components.

   Online Mode

```
$ <SETUP_DIR>/nifi/install-kylo-components.sh <NIFI_BASE_FOLDER> <KYLO_HOME> <NIFI_
↪LINUX_USER> <NIFI_LINUX_GROUP>
```

   Offline Mode

```
$ <SETUP_DIR>/nifi/install-kylo-components.sh <NIFI_BASE_FOLDER> <KYLO_HOME> <NIFI_
↪LINUX_USER> <NIFI_LINUX_GROUP> <OFFLINE_SETUP_DIR> -o
```

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
nifi.provenance.repository.implementation=com.thinkbiganalytics.nifi.provenance.repo.
↪KyloPersistentProvenanceEventRepository
```

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.

Note: See *NiFi & Kylo Provenance* for more information on provenance.

---

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.

---

11. Add additional System Property to NiFi boostrap.conf for the kylo ext-config location.

    (a) Add the next java.arg.XX in <NIFI_HOME>/conf/bootstrap.conf set to: -Dkylo.nifi.configPath=<NIFI_INSTALL>/ext-config

    Example: java.arg.15=-Dkylo.nifi.configPath=/opt/nifi/ext-config

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.

Enable Kerberos

If the cluster Kylo and NiFi will talk to has Kerberos enabled you will need to make a few additional configuration changes before starting Kylo for the first time.

13.1 Enable Kerberos for NiFi

Enable Kerberos for NiFi

13.2 Enable Kerberos for Kylo

Enable Kerberos for Kylo

13.3 Test Client

If your cluster is Kerberized its a good idea to test the keytabs generated for Kylo and NiFi to make sure they work in the JVM. Kylo provides a test client to make this easy.

1. Download the Test Client

Downloads

2. Run the test client

Follow the instructions in the test client to validate connectivity in the JVM

```
$ java -jar /opt/kylo-kerberos-test-client-VERSION.jar
```

Additional Configuration

Before starting Kylo you will want to make sure the configuration is correct. Some common cases of when you would want to change the defaults is

1. Database configuration

2. Hive thrift connection configuration

3. Set all required passwords

Note: Kylo no longer includes default passwords

14.1 Edit the Properties Files

There are 3 main properties files for Kylo

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

$ vi /opt/kylo/kylo-services/conf/spark.properties

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

For more details on the properties please see *Configuration Properties*

14.2 Kylo HDP Demo Sandbox Example

Here is an example of the properties that need to be changed to work on the Kylo HDP demo sandbox

```
spring.datasource.username=<REPLACE_ME_WITH_USERNAME>
spring.datasource.password=<REPLACE_ME_WITH_PASSWORD>

hive.datasource.username=<REPLACE_ME_WITH_USERNAME>
hive.metastore.datasource.username=<REPLACE_ME_WITH_USERNAME>
```

```
hive.metastore.datasource.password=<REPLACE_ME_WITH_PASSWORD>

nifi.service.mysql.database_user=<REPLACE_ME_WITH_USERNAME>
nifi.service.mysql.password=<REPLACE_ME_WITH_PASSWORD>
nifi.service.kylo_mysql.database_user=<REPLACE_ME_WITH_USERNAME>
nifi.service.kylo_mysql.password=<REPLACE_ME_WITH_PASSWORD>

#Note: The value for this property is the password for the dladmin user.
nifi.service.kylo_metadata_service.rest_client_password=<REPLACE_ME_WITH_PASSWORD>

modeshape.datasource.username=${spring.datasource.username}
modeshape.datasource.password=${spring.datasource.password}
metadata.datasource.username=${spring.datasource.username}
metadata.datasource.password=${spring.datasource.password}
```

14.3 Kylo Cloudera Demo Sandbox Example

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

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

```
spring.datasource.username=<REPLACE_ME_WITH_USERNAME>
spring.datasource.password=<REPLACE_ME_WITH_PASSWORD>

hive.datasource.username=<REPLACE_ME_WITH_PASSWORD>
hive.metastore.datasource.username=REPLACE_ME_WITH_USERNAME
hive.metastore.datasource.password=<REPLACE_ME_WITH_PASSWORD>
hive.metastore.datasource.url=jdbc:mysql://localhost:3306/metastore
config.hive.schema=metastore

nifi.service.mysql.database_user=REPLACE_ME_WITH_USERNAME
nifi.service.mysql.password=<REPLACE_ME_WITH_PASSWORD>
nifi.service.kylo_mysql.database_user=REPLACE_ME_WITH_USERNAME
nifi.service.kylo_mysql.password=<REPLACE_ME_WITH_PASSWORD>
nifi.service.kylo_metadata_service.rest_client_password=<REPLACE_ME_WITH_PASSWORD>
nifi.executesparkjob.sparkhome=/usr/lib/spark
config.spark.validateAndSplitRecords.extraJars=/usr/lib/hive-hcatalog/share/hcatalog/
↪hive-hcatalog-core.jar

modeshape.datasource.username=${spring.datasource.username}
modeshape.datasource.password=${spring.datasource.password}
metadata.datasource.username=${spring.datasource.username}
metadata.datasource.password=${spring.datasource.password}
```

2. Spark configuration

```
cp /etc/hive/conf/hive-site.xml /etc/spark/conf/hive-site.xml

# Snappy isn't working well for Spark on Cloudera
echo "spark.io.compression.codec=lz4" >> /etc/spark/conf/spark-defaults.conf
```

Kylo Application Properties

Below you can find all the properties used for the kylo-services application

15.1 Common Configuration Properties

Property	Default Value	Description
spring.profiles.include	native,nifi-v1.2,auth-kylo,auth-file,search-esr,jms-activemq	Profiles that should be used. Different profiles will enable certain behaviors in Kylo. Indicate the NiFi version you are using with the correct spring profile. - For NiFi 1.0.x: nifi-v1 - For NiFi 1.1.x: nifi-v1.1 - For NiFi 1.2.x or 1.3.x: nifi-v1.2 Additionally you can separate properties into separate files with the notation *application-<ProfileName>*.You can separate properties into separate files and load them Then add the ProfileName to the active profile property to use/override properties.
server.port	8420	The port Kylo runs on
liquibase.enabled	true	Liquibase allows Kylo to automatically update the database to ensure the Kylo metastore is current. If this is set to false you will need to manually run any SQL scripts when upgrading Kylo.
liquibase.change-log	classpath:com/thinkbiganalytics/db/master.xml	the location of the liquibase scripts

15.2 Kylo Operations

Property	Default Value	Description
kylo.cluster.jgroupsConfigFile		*Only for Clustered Kylo* The name of the kylo jgroups configuration file (i.e. 'kylo.cluster.jgroupsConfigFile=kylo-cluster-jgroups-config.xml')
kylo.feed.mgr.cleanup.timeout	60000	The amount of time to wait when removing feeds before issuing a Timeout error. Sometimes it can take a while to remove a feed and its data. Increase this value if you need more time to cleanup a feed.
kylo.ops.mgr.query.nifi.bulletins	true	If a failure event is detected query NiFi for any related bulletins and add them to the Job details
kylo.ops.mgr.stats.nifi.bulletins.mem.size	20	The limit to the number of bulletins to store for streaming feed failures. If statistics for streaming feeds detects a failure it will store any related NiFi bulletins in memory. This is a rolling queue that will keep the last # of errors per feed *Since 0.8.3*
kylo.ops.mgr.stats.nifi.bulletins.persist	false	When getting aggregate stats back for flows if errors are detected Kylo will query NiFi in attempt to capture matching bulletins. By default this data is stored in memory. Setting this to true will store the data in the MySQL table *Since 0.8.3*
kylo.provenance.retry.unregistered.events.enabled	true	*Only for Clustered Kylo* When receiving provenance data when Kylo is clustered, sometimes the JMS message will come through before the Cluster notification is sent to all nodes. When receiving JMS provenance events if the events are not found to match a Kylo feed set this to true to have it retry and process the events *Since 0.8.4*
kylo.provenance.retry.unregistered.events.maxRetries		

15.3 Database Connection

15.3.1 Kylo

Property	Default Value	Description
spring.datasource.driverClassName	org.mariadb.jdbc.Driver	The database driver to use. The default is for MariaDB. Be sure this matches your database (i.e. Postgres: org.postgresql.Driver, MySQL: com.mysql.jdbc.Driver)
spring.datasource.maxActive	30	Max number of connections that can be allocated by the pool at a given time
spring.datasource.username		the user name to connect to the database
spring.datasource.password		the database password
spring.datasource.testOnBorrow	true	true/false if the connection should be validated before connecting
spring.datasource.url	jdbc:mysql://localhost:3306/kylo	URL for the database
spring.datasource.validationQuery	SELECT 1	Query used to validate the connection is valid.
spring.jpa.database-platform	org.hibernate.dialect.MySQL5InnoDBDialect	Default uses MySQL. Change this to the specific database platform (i.e. for Postgres use: *org.hibernate.dialect.PostgreSQLDialect*
spring.jpa.open-in-view	true	true/false if spring should attempt to keep the connection open while in the view
metadata.datasource.driverClassName	${spring.datasource.driverClassName}	Connection to Modeshape database. This defaults to the standard Kylo *spring.datasource* property
metadata.datasource.testOnBorrow	true	Connection to Modeshape database. This defaults to the standard Kylo *spring.datasource* property
metadata.datasource.url	${spring.datasource.url}	Connection to Modeshape database. This defaults to the standard Kylo *spring.datasource* property
metadata.datasource.validationQuery	SELECT 1	Query used to validate the connection is valid.
modeshape.datasource.driverClassName	${spring.datasource.driverClassName}	Connection to Modeshape database. This defaults to the standard Kylo *spring.datasource* property
modeshape.datasource.url	${spring.datasource.url}	Connection to Modeshape database. This defaults to the standard Kylo *spring.datasource* property
modeshape.index.dir	/opt/kylo/modeshape/modeshape-local-index	Directory on this node that will store the Modeshape index files. Indexing Modeshape speeds up access to the metadata. The indexes are defined in the *metadata-repository.json* file

15.3.2 Hive

Property	Default Value	Description
hive.datasource.driverClassName	org.apache.hive.jdbc.HiveDriver	The driver used to connect to Hive
hive.datasource.url	jdbc:hive2://localhost: 10000/default	The Hive Url
hive.datasource.username		The username used to connect to Hive
hive.datasource.password		The password used to connect to Hive
hive.datasource.validationQuery	show tables 'test'	Validation Query for Hive.
hive.userImpersonation.enabled	false	true/false to indicate if user impersonation is enabled
kerberos.hive.kerberosEnabled	false	true/false to indicate if kerberos is enabled
hive.metastore.datasource.driverClassName	org.mariadb.jdbc.Driver	The driver used to connect to the Hive metastore
hive.metastore.datasource.url	jdbc:mysql://localhost: 3306/hive	The Hive metastore location
hive.metastore.datasource.username		the username used to connect to the Hive metastore
hive.metastore.datasource.password		the password used to connect to the Hive metastore
hive.metastore.datasource.testOnBorrow	true	true/false if the connection should be validated before connecting
hive.metastore.datasource.validationQuery	SELECT 1	Query used to validate the connection is valid.

15.4 JMS

More details about these properties can be found here *JMS Providers*

Property	Default Value	Description
jms.activemq.broker.url	tcp://localhost:61616	The JMS url
jms.connections.concurrent	1-1	The MIN-MAX threads to have listening for events. By default its set to 1 thread. Example. A value of 3-10 would create a minimum of 3 threads, and if needed up to 10 threads *Since: 0.8.1*
jms.client.id	thinkbig.feedmgr	The name of the client connecting to JMS

15.4.1 JMS - ActiveMQ

More detail about the ActiveMQ redelivery properties can be found here: http://activemq.apache.org/redelivery-policy.html

Property	Default Value	Description
jms.activemq.broker.username		The username to connect to JMS *Since: 0.8*
jms.activemq.broker.password		The password to connect to JMS *Since: 0.8*
jms.backOffMultiplier	5	The back-off multiplier *Since: 0.8.2*
jms.maximumRedeliveries	100	Sets the maximum number of times a message will be redelivered before it is considered a poisoned pill and returned to the broker so it can go to a Dead Letter Queue. Set to -1 for unlimited redeliveries. *Since: 0.8.2*
jms.maximumRedeliveryDelay	600000L	Sets the maximum delivery delay that will be applied if the useExponentialBackOff option is set. (use value -1 to define that no maximum be applied) (v5.5). *Since: 0.8.2*
jms.redeliveryDelay	1000	Redeliver policy for the Listeners when they fail (http://activemq. apache.org/redelivery-policy.html) *Since: 0.8.2*
jms.useExponentialBackOff	false	Should exponential back-off be used, i.e., to exponentially increase the timeout. *Since: 0.8.2*

15.4.2 JMS - Amazon SQS

Note: To use SQS you need to replace the spring profile, *jms-activemq*, with *jms-amazon-sqs*

```
spring.profiles.include=[other profiles],jms-amazon-sqs
```

Property	Default Value	Description
sqs.region.name		the sqs region, example: eu-west-1 *Since: 0.8.2.2*

15.5 Kylo SSL

The following should be set if you are running Kylo under SSL

Property	Default Value	Description
server.ssl.key-store		
server.ssl.key-store-password		
server.ssl.key-store-type	jks	
server.ssl.trust-store		
server.ssl.trust-store-password		
server.ssl.trust-store-type	JKS	

15.6 Security

Property	Default Value	Description
security.entity.access.controlled	false	To enable entity level access control change this to "true". *WARNING*: Enabling entity access control is a one-way operation; you will not be able to set this poperty back to "false" once Kylo is started with this value as "true".
security.jwt.algorithm	HS256	JWT algorithm
security.jwt.key	<insert-256-bit-secret-key-here>	The encrypted jwt key. This needs to match the same key in the kylo-ui/conf/application.properties file
security.rememberme.alwaysRemember	false	
security.rememberme.cookieDomain	localhost	
security.rememberme.cookieName	remember-me	
security.rememberme.parameter	remember-me	
security.rememberme.tokenValiditySeconds	1209600	How long to keep the token active. Defaults to 2 weeks.
security.rememberme.useSecureCookie		

15.6.1 Security - Authentication

Below are properties for the various authentication options that Kylo supports. Using an option below requires you to use the correct spring profile and configure the associated properties. More information on the different authentication settings can be found here: *Authentication*

Security - auth-simple

The following should be set if using the auth-simple profile

Property	Default Value	Description
authenticationService.username		
authenticationService.password		

Security - auth-file

Property	Default Value	Description
security.auth.file.password.hash.algorithm	MD5	
security.auth.file.password.hash.enabled	false	
security.auth.file.password.hash.encoding	base64	
security.auth.file.groups	file:///opt/kylo/groups.properties	Location of the groups file
security.auth.file.users	file:///opt/kylo/users.properties	Location of the users file

Security - auth-ldap

Property	Default Value	Description
security.auth.ldap.authenticator.userDnPatterns	uid={0},ou=people	user DN patterns are separated by '|'
security.auth.ldap.server.authDn		
security.auth.ldap.server.password		
security.auth.ldap.server.uri	ldap://localhost:52389/dc=example, dc=com	
security.auth.ldap.user.enableGroups	true	
security.auth.ldap.user.groupNameAttr	ou	
security.auth.ldap.user.groupsBase	ou=groups	

Security - auth-ad

Property	Default Value	Description
security.auth.ad.server.domain	test.example.com	
security.auth.ad.server.searchFilter	(&(objectClass=user)(sAMAccountName={1}))	
security.auth.ad.server.uri	ldap://example.com/	
security.auth.ad.user.enableGroups	true	
security.auth.ad.user.groupAttributes		group attribute patterns are separated by '|'

15.7 NiFi Rest

These properties allow Kylo to interact with NiFi

Property	Default Value	Description
nifi.rest.host	localhost	The hose NiFi is running on
nifi.rest.port	8079	The port NiFi is running on. The port should match the port found in the /opt/nifi/current/conf/nifi.properties (nifi.web.https.port)

15.7.1 NiFi Rest SSL

The following properties need to be set if you interact with NiFi under SSL Follow the document *NiFi and SSL* for more information on setting up NiFi to run under SSL

Property	Default Value	Description
nifi.rest.https	false	Set this to true if NiFi is running under SSL
nifi.rest.keystorePassword		
nifi.rest.keystorePath		
nifi.rest.keystoreType		The keystore type i.e. PKCS12
nifi.rest.truststorePassword		the truststore password needs to match that found in the nifi.properties file (nifi.security.truststorePasswd)
nifi.rest.truststorePath		
nifi.rest.truststoreType		The truststore type i.e JKS
nifi.rest.useConnectionPooling	false	Use the Apache Http Connection Pooling client instead of the Jersey Rest Client when connecting.

15.8 NiFi Flow/Template Injection

Kylo will inject/populate NiFi Processor and Controller Service properties using Kylo environment properties. Refer to this document *Configuration Properties* for details as Kylo has a number of options allowing it to interact and set properties in NiFi. Below are the default settings Kylo uses.

Property	Default Value	Description
config.category.system.prefix		A constant string that is used to prefex the category reference. This is useful if you have separate dev,qa,prod that might use the same hadoop cluster and want to prefex the locations with the environment.
config.elasticsearch.jms.url	tcp://localhost:61616	the JMS url that will be used to send/receive notification when something should be indexed in Elastic Search
config.hdfs.archive.root	/archive	Location used by the standard-ingest template to archive the data
config.hdfs.ingest.root	/etl	Location used by the standard-ingest template to land the data
config.hive.ingest.root	/model.db	Location used by the standard-ingest template for the Hive tables
config.hive.master.root	/app/warehouse	description
config.hive.profile.root	/model.db	Location used by the standard-ingest template for the Hive _profile table
config.hive.schema	hive	Schema used to query the JDBC Hive metastore. Note for Cloudera this is *metastore*
config.metadata.url	http://localhost:8400/proxy/v1/metadata	JDBC url for the Hive Metastore
config.nifi.home	/opt/nifi	Location of NiFi
config.spark.validateAndSplitRecords.extra.jars	/usr/hdp/current/hive-webhcat/share/hcatalog/hive-hcatalog-core.jar	Location of the extra jars needed for the Spark Validate/Split processor in standard-ingest template
config.spark.version	1	The spark version. Used in the Data Transformation template
nifi.executesparkjob.driver_memory	1024m	Memory setting for all *ExecuteSparkJob* processors
nifi.executesparkjob.executor_cores	1	Spark Executor Cores for all *ExecuteSparkJob* processors
nifi.executesparkjob.number_of_executors	1	Spark Number of Executors for all *ExecuteSparkJob* processors
nifi.executesparkjob.sparkhome	/usr/hdp/current/spark-client	Spark Home for all *ExecuteSparkJob* processors
nifi.executesparkjob.sparkmaster	local	Spark master setting for all *ExecuteSparkJob* processors
nifi.service.hive_thrift_service.database_connection_url	jdbc:hive2://localhost:10000/default	Controller Service named, Hive Thirft Service, default url
nifi.service.kylo_metadata_service.rest_client_password		Controller Service named, Kylo Metadata Service, Rest client password. This controller service is used for NiFi to talk to Kylo
nifi.service.kylo_metadata_service.rest_client_url	http://localhost:8400/proxy/v1/metadata	Controller Service named, Kylo Metadata Service, Rest Url. This controller service is used for NiFi to talk to Kylo
nifi.service.kylo_mysql.database_user		Controller Service named, Kylo Mysql, database user
nifi.service.kylo_mysql.password		Controller Service named, Kylo Mysql, database password

15.8 NiFi Flow/Template Injection

15.9 Unused properties

Property	Default Value	Description
application.debug	true	
application.mode	STANDALONE	
spring.batch.job.enabled	false	
spring.batch.job.names		

Grant HDFS Privileges

Kylo and NiFi requires access to HDFS and Hive. NiFi needs to write to both Hive and HDFS. There are three approaches for granting the required access to Kylo and NiFi

1. Grant the "kylo" and "nifi" service users super user privileges to access resources on the cluster

2. Control access through Ranger or Sentry

3. Manage the HDFS permissions yourself

16.1 Option 1: Grant super user privileges

This is useful in a sandbox environment where you do not need security enabled. This allows Kylo and NiFi the ability to create/edit HDFS and Hive objects.

16.1.1 Grant Superuser HDFS Privileges

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
```

Kylo 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 kylo and hdfs to that group:
$ usermod -a -G supergroup kylo
$ 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
```

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** and **Data Nodes** on a cluster.

> **Hortonworks (HDP)**

```
    $ useradd kylo

    $ useradd nifi

    $ usermod -G hdfs nifi

    $ usermod -G hdfs kylo

**Cloudera (CDH)**
```

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

16.2 Option 2: Control access through Ranger or Sentry

Instructions coming soon !!

16.3 Option 3: Manage the HDFS permissions yourself

This option is rarely used and we do not have documentation at this time

Start Services

17.1 Start Kylo and NiFi

```
$ kylo-service start
$ service nifi start
```

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

```
$ kylo-service status
$ service nifi status
$ service elasticsearch status
$ service activemq status
```

17.2 Test 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
curl localhost:9200

ActiveMQ Admin
http://127.0.0.1:8161/admin

Import Templates

The Kylo installation includes some sample ingestion templates to get you started. You can import them either through the command line or in the UI

18.1 Import from the command line

The setup folder includes a script to import the templates locally.

```
$ <KYLO_HOME>/setup/data/install-templates-locally.sh
```

18.2 Import from the Kylo UI

1. 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:

    ```
    – <kylo_project>/samples/templates/nifi-1.0/data_ingest.zip
    – /opt/kylo/setup/data/templates/nifi-1.0/data_ingest.zip
    ```

    (a) Go to the templates page in the Admin section

    (b) Click on the plus icon on the top left

    (c) Click on "Import from file" and choose the data_ingest.zip

    (d) If this is the first time you are importing the template you do not need to check any of the additional options

    (e) Click "Import Template"

2. Import Index Text Template (For Elasticsearch or SOLR).

    (a) Locate the file. You will need the file locally to upload it. You can find it in one of two places:

If you are using a version of NiFi prior to 1.3:

```
- <kylo_project>/samples/feeds/nifi-1.0/index_text_service_<TYPE>.zip
- /opt/kylo/setup/data/feeds/nifi-1.0/index_text_service_<TYPE>.zip
```

If you are using NiFi 1.3 or later:

```
- <kylo_project>/samples/feeds/nifi-1.3/index_text_service_v2.feed.zip
- /opt/kylo/setup/data/feeds/nifi-1.3/index_text_service_v2.feed.zip
```

    (a) Go to the the Feeds page in Kylo.

    (b) Click on the plus icon to add a feed.

    (c) Select "Import from a file".

    (d) Choose the file from above.

    (e) Click "Import Feed".

2. Import the data transformation template.

    (a) Locate the data_transformation.zip file. You will need the file locally to upload it. You can find it in one of two places:

```
- <kylo_project>/samples/templates/nifi-1.0/data_transformation.zip
- /opt/kylo/setup/data/templates/nifi-1.0/data_transformation.zip
```

    (a) Go to the templates page in the Admin section

    (b) Click on the plus icon on the top left

    (c) Click on "Import from file" and choose the data_transformation.zip

    (d) If this is the first time you are importing the template you do not need to check any of the additional options

    (e) Click "Import Template"

2. Import the data confidence template.

    (a) Locate the data_confidence_invalid_records.zip file. You will need the file locally to upload it. You can find it in one of two places:

```
- <kylo_project>/samples/templates/nifi-1.0/data_confidence_invalid_
→records.zip
- /opt/kylo/setup/data/templates/nifi-1.0/data_confidence_invalid_records.
→zip
```

    (a) Go to the templates page in the Admin section

    (b) Click on the plus icon on the top left

    (c) Click on "Import from file" and choose the data_confidence_invalid_records.zip

    (d) If this is the first time you are importing the template you do not need to check any of the additional options

    (e) Click "Import Template"

Create Sample Feed

Before performing any more configuration with Kylo you should create and test a simple feed to make sure all of the integration configuration is correct

Below is an example on how to create a simple feed using one of the provided CSV files.

19.1 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.

19.2 Create a category in Kylo

If you have not created a category in Kylo you can do so by going to the "Categories" page in the Feed Manager

1. Click on the plus icon.
2. Create a category called "users" and save it.

19.3 Create a data ingest feed

Next we need to create a feed under the "users" category.

1. Go to the "Feeds" page in Feed Manager

2. Click the plus icon and choose the "Data Ingest" feed type.

3. Under "General Info" give the feed a name. For example, "Test Feed 1".

4. Choose the "users" category then click "Continue to Step 2".

5. Leave the source type as "Filesystem" but change the file filter to be "userdata1.csv". The click "Continue to Step 3".

6. "Sample File" should be selected. Click on "Choose File" and find the userdata1.csv file. It is located in two places:

   - \<kylo_project>/samples/sample-data/csv/userdata1.csv

   - /opt/kylo/setup/data/sample-data/csv/userdata1.csv

7. Click the upload button to upload the file

8. Change the data type for the "registration_dttm" field name to be "timestamp" instead of "string".

9. Change the data type for the "cc" field name to be "String". Then continue to Step 4 .

10. Under field policies check the "index" box for the "id", "first_name", and "last_name" fields to index the data in Elasticsearch. Click "Continue to Step 6"

11. Continue to step 7. Change the Timer to be 5 seconds instead of 5 minutes. Then click the "Create" button

19.4 Run the sample feed

Now lets try running the feed.

1. Copy the file to the drop zone folder

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

2. You can watch the feed from both the Operations Manger page in Kylo and in NiFi to verify the file is being processed.
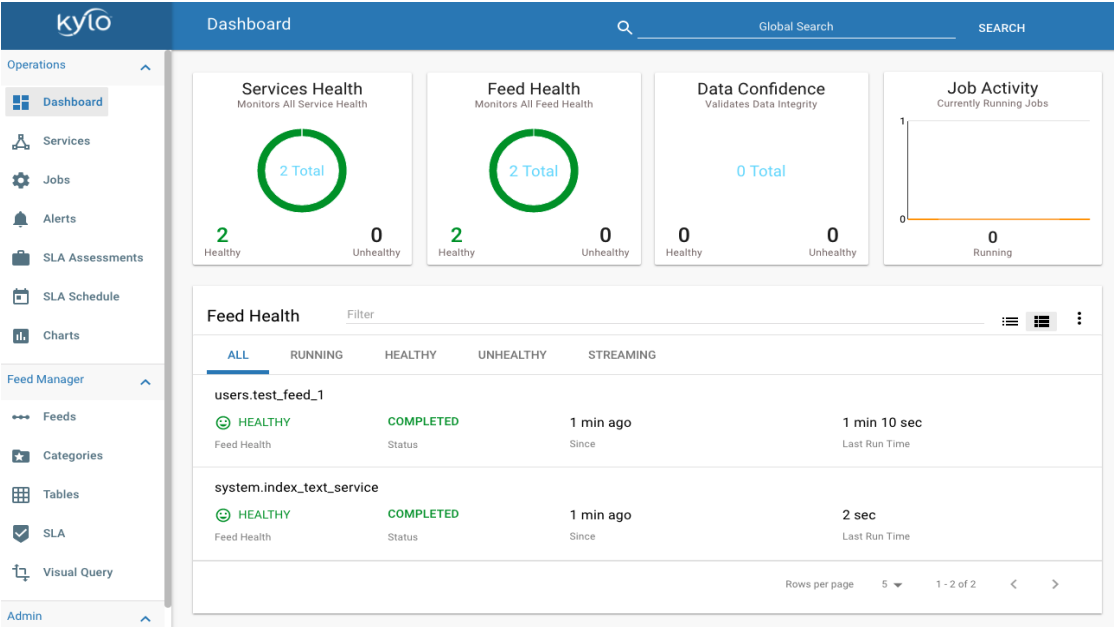
Next we will show you can to validate all of the integration is configured correctly in Kylo

Validate Configuration

Kylo has many integration points. For example, Hive, NiFi, MySQL, Spark, ActiveMQ, Elasticsearch, etc. Now that we ran a feed through we can test Kylo's integration with all of these components.

By successfully running Kylo you have validated the MySQL configuration, as well as integration with ActiveMQ and NiFi.

20.1 Validate Hive Thrift Connection

1. Test profile statistics:

    (a) Go to the "Feeds" page in feed manager and click on your test feed.

    (b) Go to the "PROFILE" tab and click "view" for one of the rows.

    (c) Go to the "VALID" and "INVALID" tabs and verify data is being returned.

2. Test the Tables page:

    (a) Go to the "Tables" page in Feed Manager.

    (b) Click on the table for your test feed. In our example it is "test_feed_1".

    (c) Click the "PREVIEW" and "QUERY" tabs to ensure data is being returned.
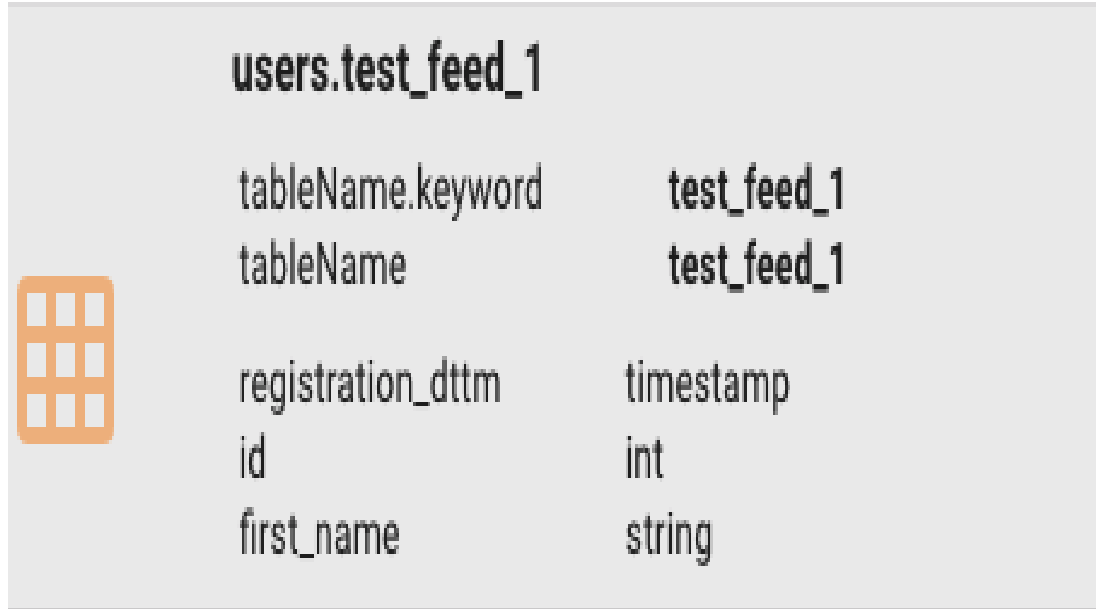
20.2 Validate Spark Shell

1. Go to the "Visual Query" page in Feed Manager.

2. In the search box type "test_feed_1" then click "add table".

3. Click "Continue to step 2". Validate you can see data.

4. Apply a transformation. An easy way to do this is to click on the "id" column, choose Filter -> "> 600". Validate you only see numbers greater than 600

20.3 Validate Search

1. Validate the schema information is there:

    (a) Enter the feed name in the global search box at the top. For example, "test_feed_1". Then click
        enter

    (b) Verify the schema metadata exists.

2. Validate the indexed data is there:

    (a) Enter the feed name in the global search box at the top. For example, "test_feed_1". Then click
        enter.

    (b) Verify the data you selected to index exists. If you remember we chose the id, first_name, and
        last_name columns.

users.test_feed_1

kylo_table.keyword		**test_feed_1**
kylo_table		**test_feed_1**
id	157	
first_name		
last_name	Rose	

users.test_feed_1

kylo_table.keyword		**test_feed_1**
kylo_table		**test_feed_1**
id	282	
first_name		
last_name	Patterson	

users.test_feed_1

kylo_table.keyword		**test_feed_1**
kylo_table		**test_feed_1**
id	292	
first_name		
last_name	Fuller	

users.test_feed_1

kylo_table.keyword		**test_feed_1**
kylo_table		**test_feed_1**
id	398	
first_name		
last_name	Flores	

HDP 2.5 Kerberos/Ranger Cluster Deployment Guide

21.1 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.

21.2 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).

21.3 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.

21.4 Prepare a Checklist

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

Prepare Install Checklist

21.5 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
```

21.6 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".

21.7 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"

21.8 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.

21.9 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
```

21.10 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
```

21.11 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 /
```

21.12 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
```

21.13 Import Kylo Templates

1. 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".

2. 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.

3. 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 superuser).

---

4. 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>.

5. Import the transform feed (Optional).

21.14 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/
```

Overview

Now that Kylo is installed and you can run a simple feed successfully, you can now get famliar wtih some of the common configuration. Some of the common configuration changes include

- Adding memory
- Changing the Java home
- Starting and stopping services
- Viewing log files
- Common cluster changes required for feeds
- Enabling yarn cluster mode
- Configure Kylo Spark Mode
- Configure to use Postgres database

Adjust Memory

23.1 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
```

Change 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
```

Log Files

25.1 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
/opt/kylo/kylo-services/conf/log4j-spark.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
```

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

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

Yarn Cluster Mode Configuration

26.1 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.

26.2 Requirements

You must have Kylo installed.

26.3 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)
```

26.4 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.
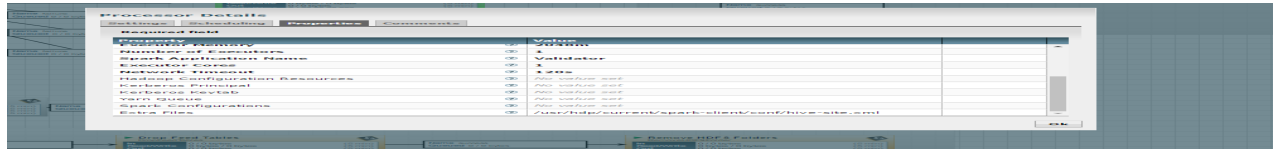
26.4.1 Step 3: Validate and Split Records Processor

If using the "Validate and Split Records" processor in the standard-ingest template, pass the JSON policy file as well.
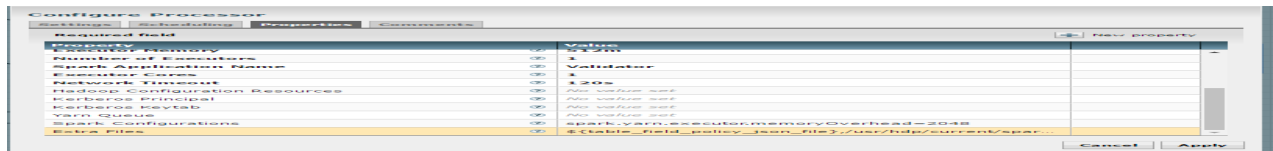
Kylo Spark Properties

27.1 Overview

The kylo-spark-shell process compiles and executes Scala code for schema detection and data transformations.

27.2 Configuration

The default location of the configuration file is at `/opt/kylo/kylo-services/conf/spark.properties`.

The process will run in one of three modes depending on which properties are defined. The default mode is *Server* which requires the process to be started and managed separately, typically using the included init script. In the other two modes, *Managed* and *Multi-User*, the kylo-services process will start and manage the kylo-spark-shell processes. Typically *Server* mode will only be used in development environments and *Managed* or *Multi-User* will be used in production. The following sections further describe these modes and their configuration options.

27.2.1 Server Mode

The kylo-spark-shell process will run in *Server* mode when the below properties are defined in `spark.properties`. In this mode the process should be started by using the included init script. When using the Kylo sandbox it is sufficient to run `service kylo-spark-shell start`.

The properties from the other sections in this document are ignored when *Server* mode is enabled. To modify the Spark options, edit the `run-kylo-spark-shell.sh` file and add them to the `spark-submit` call on the last line. Note that only the local Spark master is supported in this configuration.

Property	Type	De-fault	Description
server.port	Number	8450	Port for kylo-spark-shell to listen on.
spark.shell.server.host	String		Host name or address where the kylo-spark-shell process is running as a server.
spark.shell.server.port	Number	8450	Port where the kylo-spark-shell process is listening.
spark.ui.port	Number	8451	Port for the Spark UI to listen on.

Advanced options are available by using .

Example `spark.properties` configuration:

```
spark.shell.server.host = localhost
spark.shell.server.port = 8450
```

27.2.2 Managed Mode

In *Managed* mode, Kylo will start one kylo-spark-shell process for schema detection and another for executing data transformations.

Once the process has started it will call back to kylo-services and register itself. This allows Spark to run in yarn-cluster mode as the driver can run on any node in the cluster.

The *auth-spark* Spring profile must be enabled for the Spark client to start.

Property	Type	Default	Description
spark.shell.appResource	String		Path to the kylo-spark-shell-client jar file. This is only needed if Kylo is unable to determine the location automatically. The default location for Spark 1.x is `/opt/kylo/kylo-services/lib/app/kylo-spark- shell-client-v1-*.jar`. There is a v2 jar for Spark 2.x.
spark.shell.deployMode	String		Whether to launch a kylo-spark-shell process locally (`client`) or on one of the worker machines inside the cluster (`cluster`). Set to `cluster` when enabling user impersonation.
spark.shell.files	String		Additional files to be submitted with the Spark application. Multiple files should be separated with a comma.
spark.shell.javaHome	String		The `JAVA_HOME` for launching the Spark application.
spark.shell.idleTimeout	Number	900	Indicates the amount of time in seconds to wait for a user request before terminating a kylo-spark-shell process. Any user request sent to the process will reset this timeout. This is only used in `yarn-cluster` mode.
spark.shell.jars	String		Additional jars to be submitted with the Spark application. Multiple jars should be separated with a comma.
spark.shell.master	String		Where to run Spark executors locally (`local`) or inside a YARN cluster (`yarn`). Set to `yarn` when enabling user impersonation.
spark.shell.portMin	Number	45000	Minimum port number that a kylo-spark-shell process may listen on.
spark.shell.portMax	Number	45999	Maximum port number that a kylo-spark-shell process may listen on.
spark.shell.propertiesFile	String		A custom properties file with Spark configuration for the application.
spark.shell.registrationKeystorePassword	String		Password to keystore when registrationUrl uses SSL.
spark.shell.registrationKeystorePath	String		Path to keystore when registrationUrl uses SSL.
spark.shell.registrationUrl	String		Kylo Services URL for registering the Spark application once it has started. This defaults to `http://<server-address>:8400/ proxy/v1/spark/shell/register`
spark.shell.sparkArgs	String		Additional arguments to include in the Spark invocation.
spark.shell.sparkHome	String		A custom Spark installation location for the application.
spark.shell.verbose	Boolean	false	Enables verbose reporting for Spark Submit.

The default property values should work on most systems. An error will be logged if Kylo is unable to determine the correct value from the environment. Example `spark.properties` configuration:

```
#spark.shell.server.host = localhost
#spark.shell.server.port = 8450
spark.shell.deployMode = cluster
spark.shell.master = yarn
```

27.2.3 Multi-User Mode

Kylo will start a separate process for each Kylo user in *Multi-User* mode. This ensures that users only have access to their own tables and cannot interfere with each other.

The *auth-spark* Spring profile must be enabled for the Spark client to start.

In a Kerberized environment Kylo will need to periodically execute *kinit* to ensure there is an active Kerberos ticket. Spark does not support supplying both a keytab and a proxy user on the command-line. See *Spark User Impersonation Configuration* for more information on configuring user impersonation in a Kerberized environment.

The options from *Managed Mode* are also supported.

Property	Type	Default	Description
spark.shell.proxyUser	Boolean	false	Set to `true` to enable *Multi-User* mode.

Example `spark.properties` configuration:

```
#spark.shell.server.host = localhost
#spark.shell.server.port = 8450
spark.shell.deployMode = cluster
spark.shell.master = yarn
spark.shell.proxyUser = true
spark.shell.sparkArgs = --driver-java-options -Djavax.security.auth.
→useSubjectCredsOnly=false
```

Hadoop must be configured to allow the kylo user to proxy users:

```
$ vim /etc/hadoop/conf/core-site.xml

<property>
  <name>hadoop.proxyuser.kylo.groups</name>
  <value>*</value>
</property>
<property>
  <name>hadoop.proxyuser.kylo.hosts</name>
  <value>*</value>
</property>
```

27.2.4 Kerberos

Kerberos is supported in both *Managed* and *Multi-User* modes.

Property	Type	Default	Description
kerberos.spark.kerberosEnabled	Boolean	false	Indicates that an active Kerberos ticket is needed to start a kylo-spark-shell process.
kerberos.spark.kerberosPrincipal	String		Name of the principal for acquiring a Kerberos ticket.
kerberos.spark.keytabLocation	String		Local path to the keytab for acquiring a Kerberos ticket.
kerberos.spark.initInterval	Number	43200	Indicates the amount of time in seconds to cache a Kerberos ticket before acquiring a new one. Only used in *Multi-User* mode. A value of 0 disables calling `kinit`.
kerberos.spark.initTimeout	Number	10	Indicates the amount of time in seconds to wait for `kinit` to acquire a ticket before killing the process. Only used in *Multi-User* mode.
kerberos.spark.retryInterval	Number	120	Indicates the amount of time in seconds to wait before retrying to acquire a Kerberos ticket if the last try failed. Only used in *Multi-User* mode.
kerberos.spark.realm	String		Name of the Kerberos realm to append to usernames.

Example `spark.properties` configuration:

```
kerberos.spark.kerberosEnabled = true
kerberos.spark.kerberosPrincipal = kylo
kerberos.spark.keytabLocation = /etc/security/keytabs/kylo.headless.keytab
```

Postgres Metastore Configuration

28.1 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.

28.2 Kylo Services Configuration

28.2.1 Step 1: Ensure the Postgres driver is on the classpath

Ensure the postgres jdbc driver jar file is included in the kylo-services classpath. Copy the driver jar file to the `kylo-services/lib` folder.

28.2.2 Step 2: Update the application.properties

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
```

28.3 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:

28.3.1 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
```

28.3.2 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.

28.3.3 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 can be configured to run as a super user in a non-secure cluster or can be configured to work with secure clusters in order meet certain compliance guidelines (ex, PCI). This section includes guides on how to secure different components of the Kylo stack. We recommend following the list in order to configure security.

Encrypting Configuration Properties

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.

30.1 Encrypting Configuration Property Values with Spring CLI

1. Install the Spring CLI client Mac example. In this example we will use Home Brew to install it on a Mac:

   - Install JCE: http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html

   - Install Homebrew: http://brew.sh/

   - Install Spring Boot CLI:

```
$ brew tap pivotal/tap
$ brew install springboot
$ spring install org.springframework.cloud:spring-cloud-cli:1.0.0.BUILD-SNAPSHOT
```

2. Install the Spring CLI client Linux example:

```
$ wget http://repo.spring.io/release/org/springframework/boot/spring-boot-cli/1.5.3.
↪RELEASE/spring-boot-cli-1.5.3.RELEASE-bin.tar.gz
$ sudo mkdir /apps/spring-boot
$ sudo tar -xvf /tmp/spring-boot-cli-1.5.3.RELEASE-bin.tar.gz -C /apps/spring-boot/

$ sudo vi  /etc/profile
export SPRING_HOME=/apps/spring-boot/spring-1.5.3.RELEASE
export JAVA_HOME=/usr/lib/jvm/jre-1.8.0
export PATH=$SPRING_HOME/bin:$JAVA_HOME/bin:$PATH


$ source /etc/profile

$ sudo chown -R centos:centos /apps/spring-boot/
$ spring install org.springframework.cloud:spring-cloud-cli:1.3.1.RELEASE
```

3. Copy the /apps/kylo/encrypt.key file to the computer with the Spring CLI client (if different)

4. Encrypt the values. Note: Make sure to use single quotes around the password. If not special characters like $ will cause issues:

```
$ spring encrypt 'Pretend$Password' --key ./encrypt.key
dda0202d65ac03d250b1bc77afcf1097954wee08fc118b0f804a66xx286f61ae
```

5. Decrypt values

```
$ spring decrypt dda0202d65ac03d250b1bc77afcf1097954wee08fc118b0f804a66xx286f61ae --
↪key encrypt.key
```

Enable Kerberos for Kylo

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

31.1 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

31.2 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
```

2. Validate that the Keytabs Work.

```
[root]$ su - kylo

[kylo]$ kinit -kt /etc/security/keytabs/kylo.headless.keytab kylo

[kylo]$ klist

[root]$ su - hive

[hive]$ kinit -kt /etc/security/keytabs/hive-kylo.headless.keytab hive/sandbox.
↪hortonworks.com

[hive]$ klist
```

3. Modify the kylo-spark-shell configuration. If the *spark.shell.server* properties are set in *spark.properties* then the *run-kylo-spark-shell.sh* script will also need to be modified.

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

kerberos.spark.kerberosEnabled = true
kerberos.spark.keytabLocation = /etc/security/keytabs/kylo.headless.keytab
kerberos.spark.kerberosPrincipal = kylo@sandbox.hortonworks.com

[root]$ vi /opt/kylo/kylo-services/bin/run-kylo-spark-shell.sh

spark-submit --principal 'kylo@sandbox.hortonworks.com' --keytab /etc/security/
↪keytabs/kylo.headless.keytab ...
```

4. 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
```

5. 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.

Enable Kerberos for NiFi

32.1 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.

32.2 Types of Processors to be Configured

32.2.1 HDFS

- IngestHDFS

- CreateHDFSFolder

- PutHDFS

32.2.2 Hive

- TableRegister

- ExecuteHQLStatement

- TableMerge

32.2.3 Spark

- ExecuteSparkJob

32.3 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
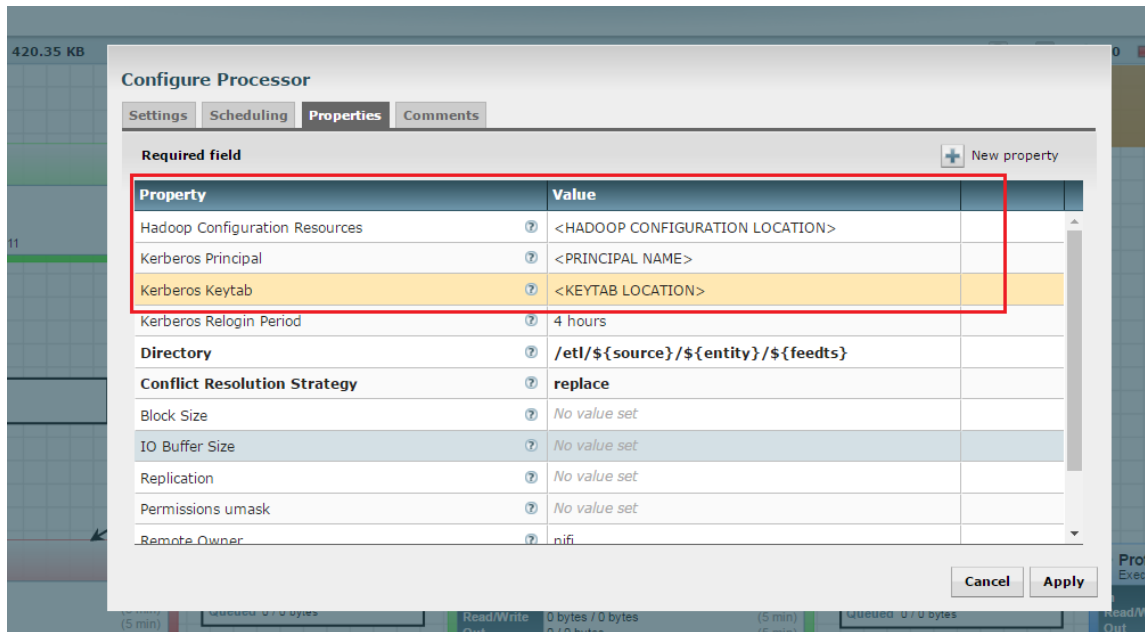
5. SPARK Processor Configuration : Log in to NiFi UI and select HDFS processor and set properties which is highlighted in red box.
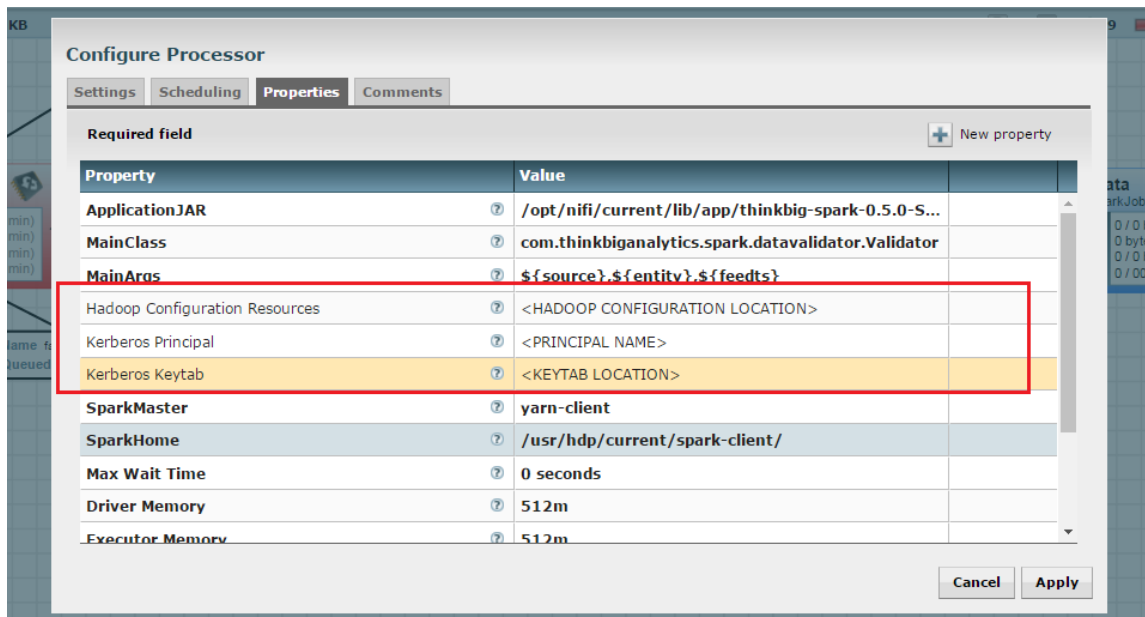
6. Hive Processor Configuration : Log in to NiFi UI and go to toolbar.
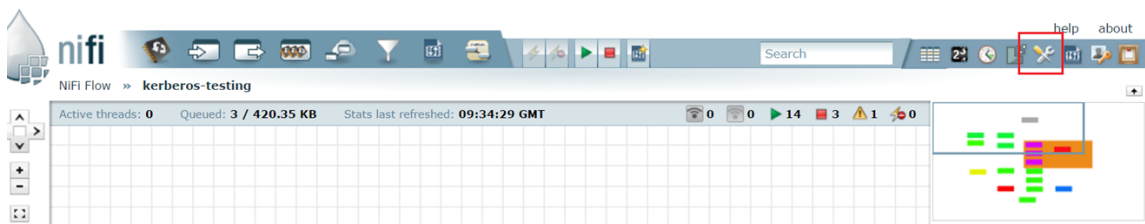
7. Go to Controller Service Tab and disable Thrift Controller Services if already running which highlighted in red box.

8. Make sure everything has stopped properly like below.

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'
```
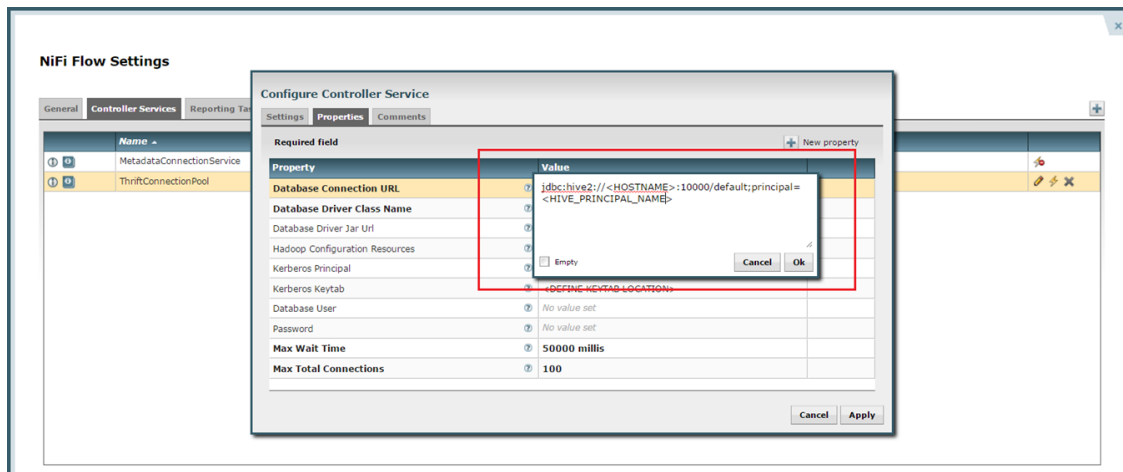
10. Update Kerberos user information and Hadoop Configuration. Apply Changes and start controller services.

    You have successfully configured NiFi DataLake Platform with Kerberos.

Enable Ranger Authorization

33.1 Prerequisite

33.2 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/
```

33.3 Kylo

This documentation assumes that you have Kylo installed and running on a cluster.

33.4 Optional: Delete/Disable HDFS/HIVE Global Policy

If you are using HDP sandbox, remove all HDFS/HIVE global policy.

Disable the HDFS Policy.

Disable the HIVE policy.

33.5 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

33.6 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 authorization is configured successfully. Now create a feed from the Kylo UI and create feed for testing.

Enable Sentry Authorization

34.1 Prerequisite

34.1.1 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/
```

34.1.2 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.

34.1.3 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**.

34.2 Enabling Sentry for Hive

34.2.1 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
```

34.2.2 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.

34.2.3 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.

34.2.4 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.

34.2.5 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;

34.2.6 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.

Sentry authorization is configured successfully. Now create a feed from the Kylo UI and test it.

Kylo UI and SSL

35.1 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

35.1.1 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
```

35.1.2 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

Add StandardSSLContextService in Nifi at the root level next to all other controller services.

Set following properties on SSL Context Service:

> **Truststore Filename** /opt/nifi/data/ssl/kylo-ui-truststore.jks

Truststore Password changeit

Truststore Type JKS

2.3 Update MetadataProviderSelectorService

Update MetadataProviderSelectorService. 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

NiFi and SSL

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".
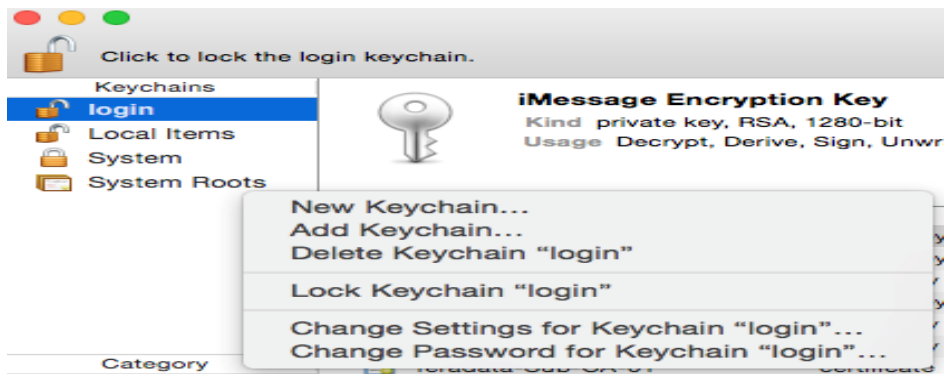
   (b) Give it the name "nifi-cert" and a password.
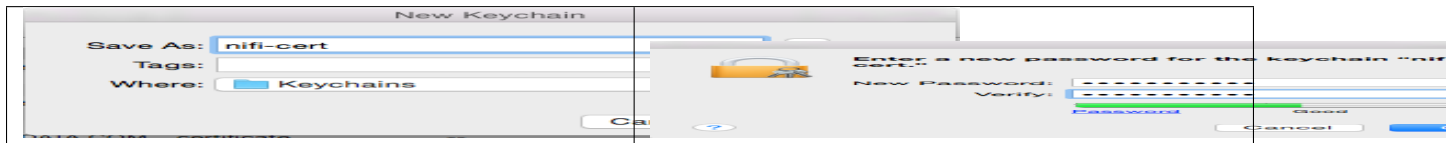
4. Once the keychain is created, click on it and select File -> import Items, and then find the .p12 file that you copied over in step 1.
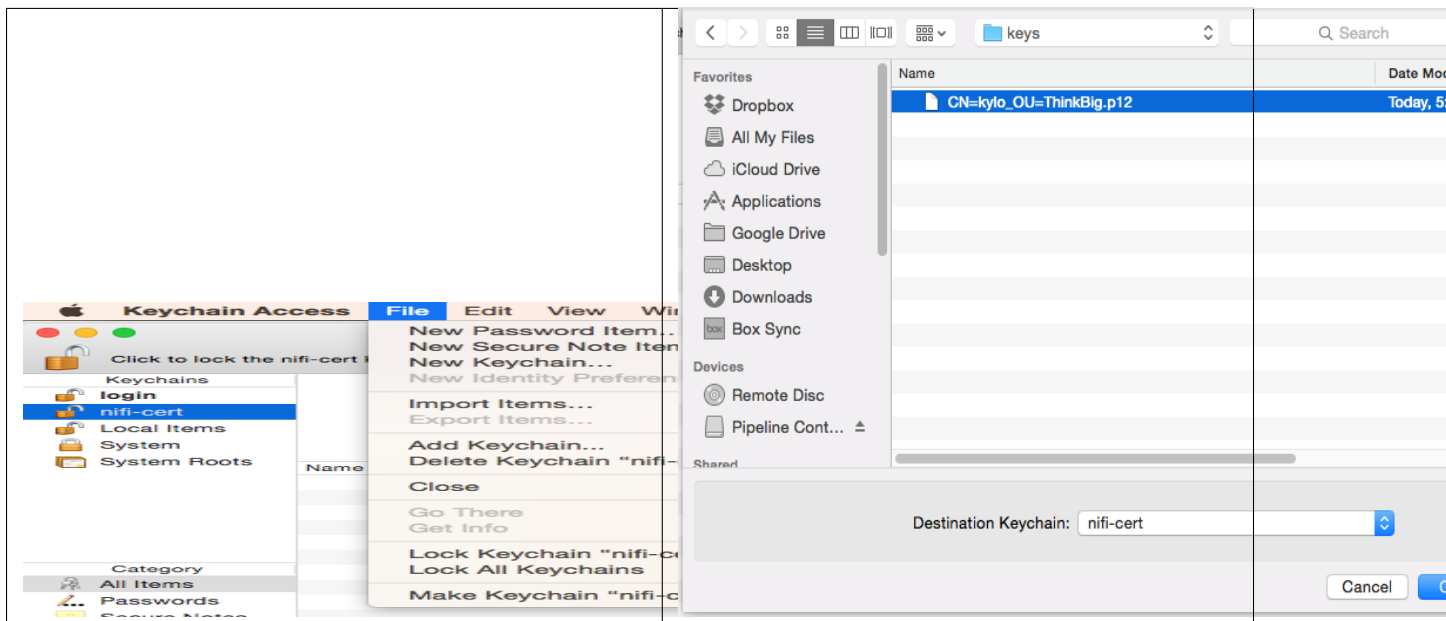
   Once complete you should have something that looks like this:
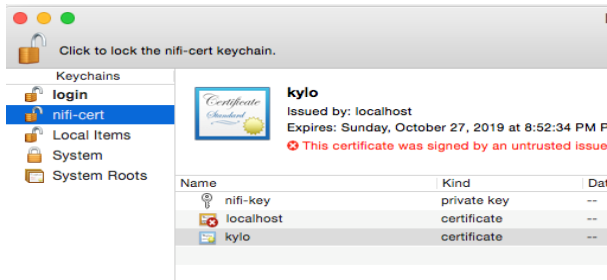
Accessing NiFi under SSL

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.
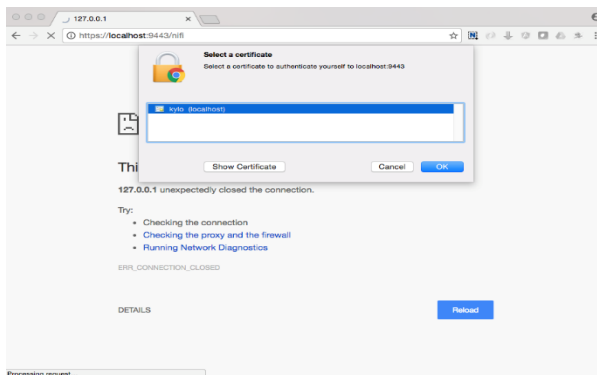
2. Enter the Password that you supplied for the keychain. This is the password that you created for the keychain in "Importing the Client Cert on the Mac" Step 3b.
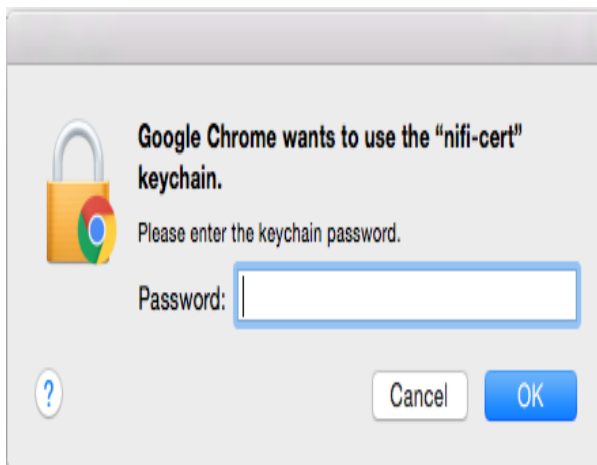
3. Click Always Verify.
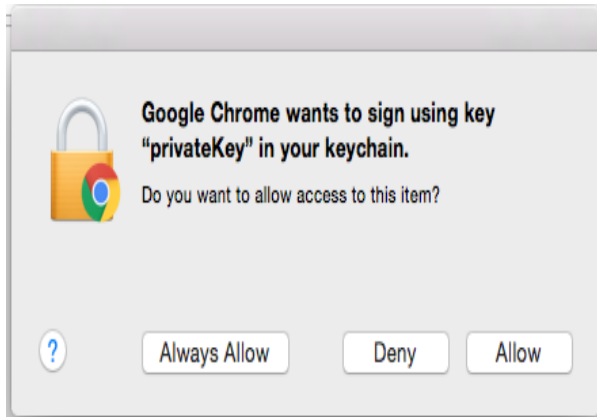
4. Click AdvancKyloConfiguration.rsted and then Click Proceed. It will show up as "not private" because it is a self-signed cert.
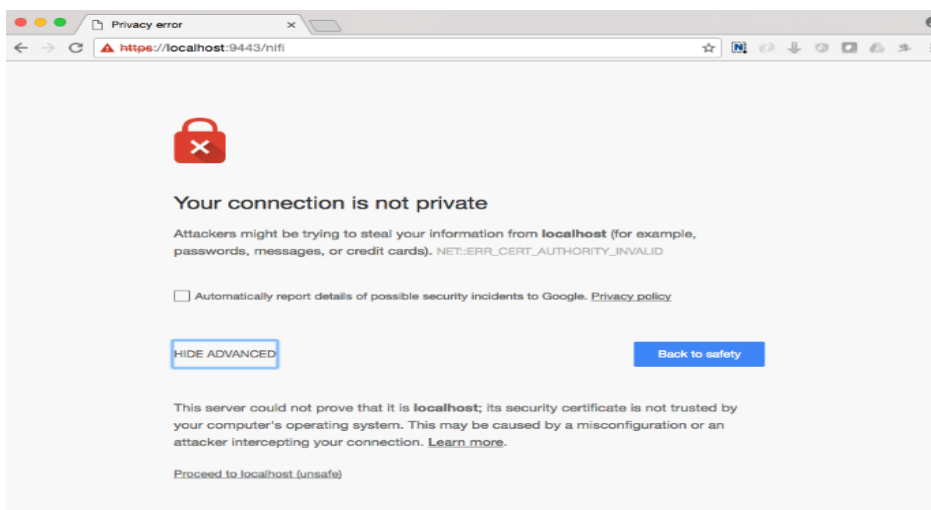
5. NiFi under SSL. Notice the User name matches the one supplied via the certificate that we created: "CN=kylo, OU=NIFI".
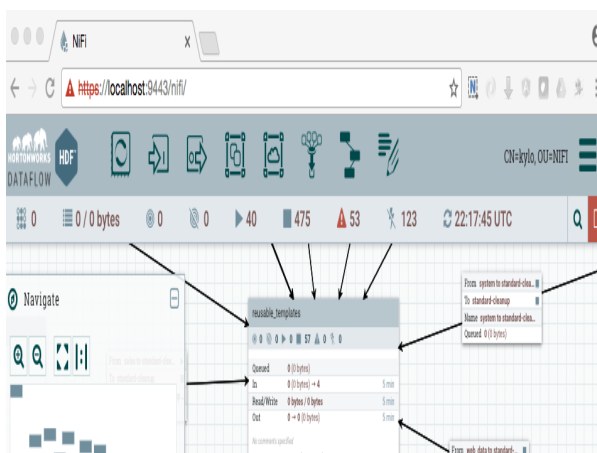
Refer to the Hortonworks documentation on Enabling SSL for NiFi:

Authentication

37.1 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.

37.2 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	*auth-kylo*	Authenticates users against the Kylo user/group store
LDAP	*auth-ldap*	Authenticates users stored in LDAP
Active Directory	*auth-ad*	Authenticates users stored in Active Directory
Users file	*auth-file*	Authenticates users in a file users.properies (typically used in development only)
Simple	*auth-simple*	Allows only one admin user defined in the configuration properties (development only)
Cached credentials	*auth-cache*	Short-cicuit, temporary authentication after previous user authentication by other means

37.2.1 *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	Required	Example	Description
security.auth.kylo.login.services	No	`required`	Corresponds to the control flag for LoginModule configurations: *required*, *requisite*, *sufficient*, and *optional*. Possible values are *required*, *requisite*, *sufficient*, and *optional*

37.2.2 *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 top 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	Required	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 *file:///*, 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
```

37.2.3 *auth-ldap*

This profile configures a LoginModule that authenticates the username and password against an LDAP server.

Property	Required	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 substitued 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

37.2.4 *auth-ad*

This profile configures a LoginModule that authenticates the username and password against an Active Directory server. If the properties `security.auth.ad.server.serviceUser` and `security.auth.ad.server.`

`servicePassword` are set then those credentials will be used to autheticate with the AD server and only the username will be validated to exist in AD; loading the user's groups load (when configured) if the user is present.

Property	Re-quired	Example Value	Description
secu-rity.auth.ad.server.uri	Yes	`ldap://example.com/`	The URI to the AD server
secu-rity.auth.ad.server.domain	Yes	`test.example.com`	The AD domain of the users to authenticate
secu-rity.auth.ad.server.searchFilter	No	`(&(objectClass=user)` `(sAMAccountName={1}))`	Specifies the filter to use to find AD entries for the login user; default: `(&(objectClass=user)(userPrincipalName={0}))`
secu-rity.auth.ad.server.serviceUser	No	`admin`	A service account used to authenticate with AD rather than the user logging in (typically used with auth-spnego)
secu-rity.auth.ad.server.servicePassword	No		A service account password used to authenticate with AD rather than that of the user logging in (typically used with auth-spnego)
secu-rity.auth.ad.user.enableGroups	No	`true`	Activates user group loading; default: `false`

37.2.5 *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

37.2.6 *auth-cache*

Kylo's REST API is stateless and every request must be authenticated. In cases where the REST API is heavily used and/or the primary means of authetication is expensive, this profile can be used to reduce the amount of times the primary authentication mechanism is consulted. This is achieved by inserting a LoginModule a the head of the login sequence, flagged as Sufficient, that reports a login success if the user credential for the current request is present in its cache. Another LoginModule, flagged as Optional, is inserted at the end of the sequence to add the credential to the cache whenever a successful login is committed.

Property	Re-quired	Example Value	Description
secu-rity.auth.cache.spec	No	`expireAfterWrite=30s,` `maximumSize=512`	The cache specification (entry expire time, cache size, etc.)

37.3 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.

37.4 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.

37.4.1 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.

Kylo Kerberos SPNEGO

38.1 Configuration

38.1.1 auth-krb-spnego

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 either the `auth-kylo` or `auth-ad` profile must be used as well. This is because requests reaching Kylo when SPNEGO is used will already be authenticated but the groups associated with the requesting user must still be associated during Kylo authentication. Both the configurations activated by `auth-kylo` and `auth-ad` are SPNEGO-aware and allow serice accounts properties to be set for use in looking up the groups of user from the Kylo user store or Active Directory.

Once SPNEGO is configured in *kylo-services* the services' REST API will begin to accept SPNEGO `Authorization:   Negotiate` headers for authentication. The REST API will continue to accept HTTP BASIC authentication requests as well.

When `auth-krb-spnego` is activated, the following properties are required to configure Kerberos SPNEGO:

Property	Description	Example
security.auth.krb.service-principal	Names the service principal used to access Kylo	HTTP/kylo.domain.com@EXAMPLE.COM
security.auth.krb.keytab	Specifies path to the keytab file containing the service principal	/opt/kylo/kylo.keytab

38.1.2 auth-kylo

If the `auth-kylo` profile is activated with SPNEGO then the *kylo-ui/conf/appplication.properties* file must contain the credential properties specified in the table below to allow access to the Kylo user store via the kylo-services' REST API using BASIC auth. The authentication configuration for *kylo-services* can be anything that accepts the credentials specified in these properties.

Property	Description
security.auth.kylo.login.username	Specifies a Kylo username with the rights to retrieve all of the Kylo groups of which the authenticating user is a member
security.auth.kylo.login.password	Specifies the password of the above username retrieving the authenticating user's groups

38.1.3 auth-ad

If the `auth-ad` profile is activated with SPNEGO then the properties in the table below must be set in *kylo-ui/conf/appplication.properties* and *kylo-services/conf/appplication.properties* (if the profile is used in *kylo-services*).

Property	Description
security.auth.ad.user.enableGroups	This should be set to `true` as group loading would be the only purpose of activating *auth-ad* with SPNEGO
security.auth.ad.server.serviceUser	Specifies a username in AD with the rights to retrieve all of the groups of which the authenticating user is a member
security.auth.ad.server.servicePassword	Specifies the password of the above AD username retrieving the authenticating user's groups

38.1.4 Kerberos Configuration

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 properties as specified above.

38.2 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)

38.3 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

Access Control

39.1 Overview

A goal is to support authentication and authorization seamlessly between the Kylo applications and the Hadoop cluster.

39.2 Authorization

Authorization within Kylo uses access control lists (ACL) to control what users can do and see. 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. These actions are organized into a hierarchies and permission to perform an action may be granted at any level in that hierarchy.

Authorization in Kylo is divided into two layers: service-level (Kylo-wide) permissions and (when enabled) entity-level permissions. 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 only by an external authentication source (such as LDAP) via a plugin module then user groups may not be visible in the Users page.

39.2.1 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.

39.3 Service-Level Authorization

Service-level access controla 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 heirarchical 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.)

39.4 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

---

39.4.1 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 verion 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

Category-Wide Feed Role Memberships

Kylo supports adding users and groups to feed roles at the category level that apply to all feeds under that category. This is useful when you wish to organize your feed access control around feeds grouped by category and apply all feed access control changes in one place. Assigning feed role memberships at the category level does not prevent adding additional memberships on each individual feed however. The members of the roles of a particular feed are the union of all memberships assigned at the individual feed level and at the level of the category containing that feed.

In Kylo feed role memberships are managed by editing them in the category details page just below where the category role memberships are managed.

39.5 Why Two Levels of Access Control?

Kylo support two levels acces 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.

39.6 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 reqired to perform those actions

39.6.1 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

39.6.2 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

39.6.3 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

39.6.4 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

Spark User Impersonation Configuration

40.1 Overview

Users in Kylo have access to all Hive tables accessible to the `kylo` user by default. By configuring Kylo for a secure Hadoop cluster and enabling user impersonation, users will only have access to the Hive tables accessible to their specific account. A local spark shell process is still used for schema detection when uploading a sample file.

40.2 Requirements

This guide assumes that Kylo has already been setup with Kerberos authentication and that each user will have an account in the Hadoop cluster.

40.3 Kylo Configuration

Kylo will need to launch a separate spark shell process for each user that is actively performing data transformations. This means that the `kylo-spark-shell` service should no longer be managed by the system.

1. Stop and disable the system process.

```
$ service kylo-spark-shell stop
$ chkconfig kylo-spark-shell off
```

2. Add the auth-spark profile in `application.properties`. This will enable Kylo to create temporary credentials for the spark shell processes to communicate with the kylo-services process.

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

spring.profiles.include = auth-spark, ...
```

3. Enable user impersonation in `spark.properties`. It is recommended that the yarn-cluster master be used to ensure that both the Spark driver and executors run under the user's account. Using the local or yarn-client masters are possible but not recommended due the Spark driver running as the kylo user.

```
$ vim /opt/kylo/kylo-services/conf/spark.properties:

# Ensure these two properties are commented out
#spark.shell.server.host
#spark.shell.server.port

# Executes both driver and executors as the user
spark.shell.deployMode = cluster
spark.shell.master = yarn
# Enables user impersonation
spark.shell.proxyUser = true
# Reduces memory requirements and allows Kerberos user impersonation
spark.shell.sparkArgs = --driver-memory 512m --executor-memory 512m --driver-java-
→options -Djavax.security.auth.useSubjectCredsOnly=false

kerberos.spark.kerberosEnabled = true
kerberos.spark.kerberosPrincipal = kylo
kerberos.spark.keytabLocation = /etc/security/keytabs/kylo.headless.keytab
```

4. Redirect logs to `kylo-spark-shell.log`. By default the logs will be written to `kylo-services.log` and include the output of every spark shell process. The below configuration instead redirects this output to the `kylo-spark-shell.log` file.

```
$ vim /opt/kylo/kylo-services/conf/log4j.properties

log4j.additivity.org.apache.spark.launcher.app.SparkShellApp=false
log4j.logger.org.apache.spark.launcher.app.SparkShellApp=INFO, sparkShellLog

log4j.appender.sparkShellLog=org.apache.log4j.DailyRollingFileAppender
log4j.appender.sparkShellLog.File=/var/log/kylo-services/kylo-spark-shell.log
log4j.appender.sparkShellLog.append=true
log4j.appender.sparkShellLog.layout=org.apache.log4j.PatternLayout
log4j.appender.sparkShellLog.Threshold=INFO
log4j.appender.sparkShellLog.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %t:
→%c{1}:%L - %m%n
```

5. Configure Hadoop to allow Kylo to proxy users.

```
$ vim /etc/hadoop/conf/core-site.xml

<property>
  <name>hadoop.proxyuser.kylo.groups</name>
  <value>*</value>
</property>
<property>
  <name>hadoop.proxyuser.kylo.hosts</name>
  <value>*</value>
</property>
```

Setup A NiFi Cluster in a Kylo Sandbox

41.1 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.

41.2 Prerequisite

You will need to have set up a Kylo sandbox according to the *Setup Wizard Deployment Guide*.

41.3 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
```

41.4 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
```

41.5 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.

Clustering Kylo

Kylo Clustering is now available starting with version v0.8.1.

Kylo uses jgroups, http://jgroups.org/index.html, for cluster configuration. This is chosen because Kylo's metadata engine, Modeshape (http://modeshape.jboss.org/) uses jgroups internally for its cluster management.

Two jgroups configuration files are needed to be setup (One for ModeShape and one for Kylo)

42.1 ModeShape Configuration

1. Update the metadata-repository.json file and add the "clustering" section

```
"clustering": {
    "clusterName":"kylo-modeshape-cluster",
    "configuration":"modeshape-jgroups-config.xml",
    "locking":"db"
},
```

Make sure the name of the jgroups-config.xml file is in the */kylo-services/conf* folder. Refer sample files for setting up a jgroups configuration at */opt/kylo/setup/config/kylo-cluster*. Note if working in Amazon you need to refer to the "s3" jgroups configuration as it needs to use an S3Ping to have the nodes communicate with each other.

42.2 Kylo Configuration

We also have another jgroups configuration setup for Kylo nodes. We cannot use the ModeShape cluster configuration since that is internal to ModeShape.

1. Create a similar jgroup-config.xml file and add it to the */kylo-services/conf* file. Refer sample files for setting up a jgroups configuration at */opt/kylo/setup/config/kylo-cluster*. Ensure the ports are different between this xml file and the ModeShape xml file

2. Add a property to the kylo-services/conf/application.properties to reference this file

```
kylo.cluster.jgroupsConfigFile=kylo-cluster-jgroups-config.xml
```

3. Startup Kylo

   When starting up you should see 2 cluster configurations in the logs. One for the modeshape cluster and one for the kylo cluster

```
-------------------------------------------------------------
GMS: address=Kylo - MUSSR186054-918-31345, cluster=kylo-modeshape-cluster,␣
↪physical address=127.0.0.1:7800
-------------------------------------------------------------
```

```
-------------------------------------------------------------
GMS: address=Kylo - MUSSR186054-918-31345, cluster=internal-kylo-cluster,␣
↪physical address=127.0.0.1:7900
-------------------------------------------------------------
2017-05-04 06:17:06 INFO  pool-5-thread-1:JGroupsClusterService:120 -␣
↪Cluster membership changed: There are now 1 members in the cluster. [Kylo -␣
↪ MUSSR186054-918-31345]
2017-05-04 06:17:06 INFO  pool-5-thread-1:JGroupsClusterService:155 - ***␣
↪Channel connected Kylo - MUSSR186054-918-31345,[Kylo - MUSSR186054-918-␣
↪31345]
2017-05-04 06:17:06 INFO  pool-5-thread-1:NifiFlowCacheClusterManager:205 -␣
↪on connected 1 members exist.  [Kylo - MUSSR186054-918-31345]
```

42.3 Quartz Scheduler Configuration

When running in clustered mode you need to configure the Quartz SLA scheduler to be backed by the database and run it in clustered mode. Do the following:

1. Download and extract the Quartz distribution to a machine. http://d2zwv9pap9ylyd.cloudfront.net/quartz-2.2.3-distribution.tar.gz You just need this to get the database scripts.

2. Run the Quartz database scripts for your database found in the *docs/dbTables*

3. Create a *quartz.properties* file and put it in the */opt/kylo/kylo-services/conf* folder. Refer to a sample file */opt/kylo/setup/kylo-cluster/quartz-cluster-example.properties*

   (a) Do not specify datasource connection information in this file. The system will use the default *spring.datasource* property information found in the *application.properties* for the database connection

42.4 Service Monitoring

You can monitor the health of the kylo cluster by adding the *kylo-service-monitor-kylo-cluster.jar* to the */opt/kylo/kylo-services/plugins* folder.

1. Copy the file in the */opt/kylo/setup/plugins/kylo-service-monitor-kylo-cluster-VERSION.jar* to the /opt/kylo/kylo-services/plugins' folder

2. Add a new property to the application.properties to indicate the expected number of nodes you are running in your cluster. Below is an example expecting 2 nodes in the cluster

```
kylo.cluster.nodeCount=2
```

3. Now a new *Kylo Cluster* service will appear in the Kylo dashboard and show you cluster health status

Service Components

Filter

Kylo - sandbox-20892

☺ HEALTHY

All 2 nodes are connected. Currently connected to this node. T

Message

Kylo - sandbox-53398

☺ HEALTHY

All 2 nodes are connected. There are 2 members in the cluster.

Message

42.5 Testing - (as of Kylo 0.8.4)

Starting with Kylo 0.8.4 you can test your cluster configuration in the Kylo application.

1. Start your Kylo Cluster (both services and ui on all your nodes)

2. In a new web browser (i.e. Chrome), connect to a specific node in your cluster. Go to this webpage *http://localhost:8400/index.html#!/admin/cluster* (replace localhost and port with the direct host and port)

   • This page will show you Cluster information and allow you to send/receive test messages

3. In a new web browser (i.e. Firefox), connect to another node in your cluster. Follow the steps above and connect to a different node/port.

4. Send a message from one of the nodes. You should see the message appear in the other web browser. See screenshots below.

   The screenshots below are from a 2 node Kylo cluster.

   • Node 1: *kylo-sandbox-43958*

   • Node 2: *kylo-sandbox-59542*

Screenshot from *Node 1: kylo-sandbox-43958*

Screenshot from *Node 2: kylo-sandbox-59542*
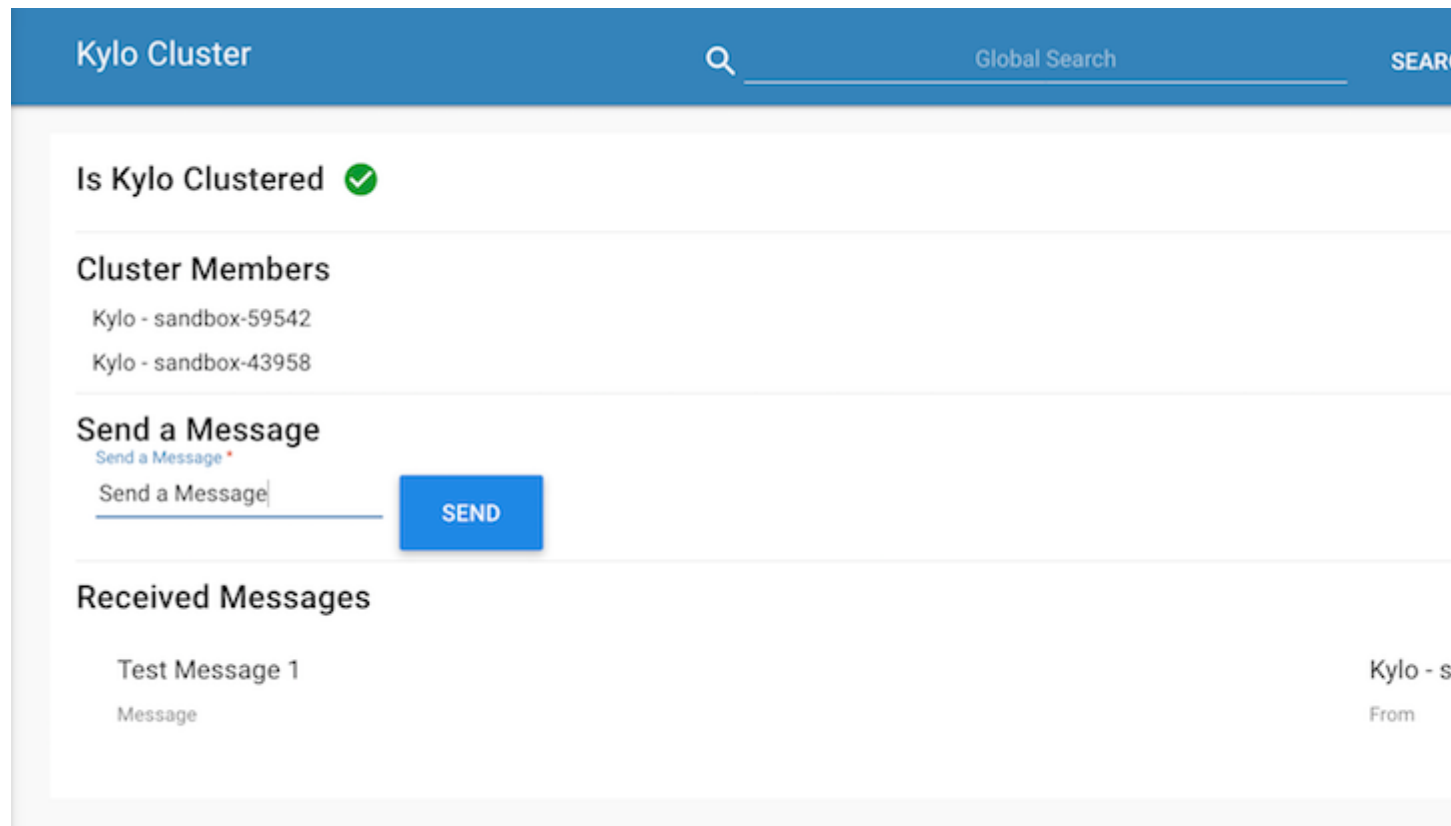
- If this page correctly shows your cluster members, but fails to send the message or receive it follow the *Troubleshooting* tips below. Most likely cause of this is the system parameter *-Dava.net.preferIPv4Stack=true* needs to be configured.

- If the cluster is not configured correctly this page will indicate and provide you with information on troubleshooting.

42.6 Troubleshooting

- If you are having issues identifying if the clustering is working you can modify the log4j.properties and have it show cluster events. This is especially useful for modeshape. Note: by doing this logs will be very verbose, so its recommended this is only done for initial setup/debugging

```
log4j.logger.org.modeshape.jcr.clustering.ClusteringService=DEBUG
log4j.logger.org.jgroups=DEBUG
```

- If you get a *Network is unreachable* error, below, you may need to do the following:

  - Network unreachable error

    ```
    SEVERE: JGRP000200: failed sending discovery request
    java.io.IOException: Network is unreachable
        at java.net.PlainDatagramSocketImpl.send(Native Method)
        at java.net.DatagramSocket.send(DatagramSocket.java:693)
        at org.jgroups.protocols.MPING.sendMcastDiscoveryRequest(MPING.
    →java:295)
        at org.jgroups.protocols.PING.sendDiscoveryRequest(PING.java:62)
    ```

```
      at org.jgroups.protocols.PING.findMembers(PING.java:32)
      at org.jgroups.protocols.Discovery.findMembers(Discovery.java:244)
```

– Modify the */opt/kylo/kylo-services/bin/run-kylo-services.sh*

– Add -Djava.net.preferIPv4Stack=true

```
java $KYLO_SERVICES_OPTS -Djava.net.preferIPv4Stack=true -cp /opt/kylo/
↪kylo-services/conf ....
```

- Multicast

  – Enabling multicast is done via the *<MPING .. />* xml node in the jgroups-configuration xml file. Multicast
    may not work in your environment. If you have issues you can remove the *<MPING ../>* node and ensure
    your host names are configured propertly in the *<TCPPING ../>* node. Refer to the jgroups documentation
    around MPING for more information: http://jgroups.org/manual-3.x/html/protlist.html#d0e4760

- Running the Multicast test program

  – Run the following to test 2 node communication. The below was taken from http://www.jgroups.
    org/manual/html/ch02.html#ItDoesntWork

  1. Stop kylo-services on both nodes

  2. On 1 node run the code below to act as a receiver. Replace the `bind_addr` and `port` argu-
     ments with your specific values

```
java -Djava.net.preferIP4Stack=true  -cp  /opt/kylo/kylo-services/
↪conf:/opt/kylo/kylo-services/lib/*:/opt/kylo/kylo-services/plugin/*␣
↪org.jgroups.tests.McastReceiverTest -bind_addr 127.0.0.1 -port 7900
```

  3. On another node run the code below to act as a sender. Replace the `bind_addr` and `port`
     arguments to match the values above

```
java -Djava.net.preferIP4Stack=true -cp  /opt/kylo/kylo-services/
↪conf:/opt/kylo/kylo-services/lib/*:/opt/kylo/kylo-services/plugin/*␣
↪org.jgroups.tests.McastSenderTest -bind_addr 127.0.0.1 -port 7900
```

  As a Sender you will get a prompt. Type in some string and then verify its received on the
  other node.

  Sender:

```
org.jgroups.tests.McastSenderTest -bind_addr 127.0.0.1 -
↪port 7900
Socket #1=0.0.0.0/0.0.0.0:7900, ttl=32, bind interface=/127.
↪0.0.1
> this is a test message
```

  Receiver:

```
this is a test message [sender=127.0.0.1:7900]
```

NiFi & Kylo Provenance

43.1 Introduction

Kylo uses a custom ProvenanceRepository (KyloPersistentProvenanceEventRepository) to send data from NiFi to Kylo. A custom NiFi nar file https://github.com/Teradata/kylo/tree/master/integrations/nifi/nifi-nar-bundles/nifi-provenance-repo-bundle is used for the ProvenanceRepository.

43.2 Setup

1. Edit the nifi.properties file (`/opt/nifi/current/conf/nifi.properties`) and change the `nifi.provenance.repository.implementation` property as below:

   ```
   # Provenance Repository Properties
   #nifi.provenance.repository.implementation=org.apache.nifi.provenance.
   ↪PersistentProvenanceRepository
   nifi.provenance.repository.implementation=com.thinkbiganalytics.nifi.
   ↪provenance.repo.KyloPersistentProvenanceEventRepository
   ```

2. Ensure the correct nars are available in the NiFi classpath. Depending upon the NiFi version there are 2 different nar files that are used. If you use the kylo wizard it will copy the nar files and setup the symlinks to point to the correct nar version for your NiFi installation.

   - For NiFi 1.0 or 1.1

     – kylo-nifi-provenance-repo-v1-nar-<version>.nar

   - For NiFi 1.2 or 1.3

     – kylo-nifi-provenance-repo-v1.2-nar-<version>.nar

3. **Configure the KyloPersistentProvenanceEventRepository properties: The Provenance Repository uses properties found in**
   Note: this location is configurable via the System Property `kylo.nifi.configPath` passed into NiFi when it launches. Below are the defaults which are automatically set if the file/properties are not found.

Note: the config.properties marked with ## *Supports dynamic update* below can be updated without restarting NiFi. Every 30 seconds a check is made to see if the config.properties file has been updated.

```
 ###
jms.activemq.broker.url=tcp://localhost:61616

## Back up location to write the Feed stats data if NiFi goes down
## *Supports dynamic update*
kylo.provenance.cache.location=/opt/nifi/feed-event-statistics.gz

## The maximum number of starting flow files per feed during the given␣
↪run interval to send to ops manager
## *Supports dynamic update*
kylo.provenance.max.starting.events=5

## The number of starting flow files allowed to be sent through until␣
↪the throttle mechanism in engaged.
# if the feed starting processor gets more than this number of events␣
↪during a rolling window based upon the kylo.provenance.event.
↪throttle.threshold.time.millis timefame events will be throttled␣
↪back to 1 per second until its slowed down
kylo.provenance.event.count.throttle.threshold=15

## Throttle timefame used to check the rolling window to determine if␣
↪rapid fire is occurring
kylo.provenance.event.throttle.threshold.time.millis=1000

## run interval to gather stats and send to ops manager
## *Supports dynamic update*
kylo.provenance.run.interval.millis=3000

## JSON string of the Event Type to Array of Processor classes
## These processors produce orphan child flow files that dont send␣
↪DROP provenance events for the children.
## Child flow files produced by events  matching the EventType and␣
↪processor class will not be processed
## *Supports dynamic update*
kylo.provenance.orphan.child.flowfile.processors={"CLONE":[
↪"ConvertCSVToAvro"]}
```

43.3 Event Processing

When NiFi runs the processors will send provenance events to JMS Queues. Kylo listens on these JMS queues and creates Jobs/Steps and Streaming statistics about each feed and job execution. These are displayed in the Operations Manager.

NiFi Processor Guide

44.1 ImportSqoop Processor

44.1.1 About

The ImportSqoop processor allows loading data from a relational system into HDFS. This document discusses the setup required to use this processor.

44.1.2 Starter template

A starter template for using the processor is provided at:

```
samples/templates/nifi-1.0/template-starter-sqoop-import.xml
```

44.1.3 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**.

44.1.4 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.

44.1.5 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.

44.2 TriggerFeed

44.2.1 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.

44.2.2 Obtaining the Dependent Feed Execution Context

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
-feedts=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}
```

44.2.3 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":{
            }
        }
    }
}
```

44.2.4 Example Flow

The screenshot shown here is an example of a flow in which the inspection of the payload triggers dependent feed data.
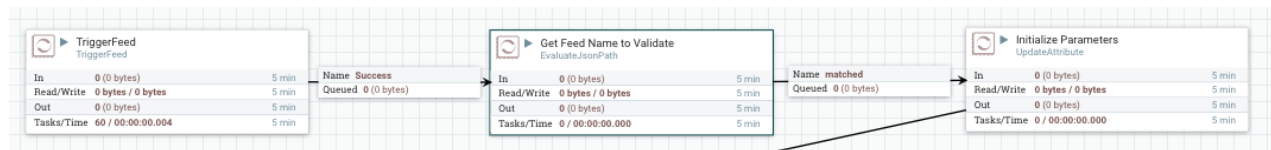
The EvaluateJSONPath processor is used to extract JSON content from the flow file.

Refer to the Data Confidence Invalid Records flow for an example:

Kylo Templates Guide

Templates facilitate the creation of data flows. They can be:

- normal (1 template for the whole flow)
- reusable (1 reusable template and 1 flow template)

Important: More on reusable flows here

45.1 Setup templates

45.1.1 Import Kylo template

1. Import template from file
2. Select file
3. Select overwrite + replace the reusable template option
4. Register the template

Note: The following sections apply only if you didn't import yet a template in Kylo, or are lacking a Kylo template archive.

45.1.2 Import reusable template

1. Import template from file.

> **Warning:** You can't import the reusable template from NiFi environment, as it has input/output ports which need to be connected.

2. Select file and select overwrite + replace the reusable template option

3. Register the template

45.1.3 Import flow template

1. Import template from NiFi environment (as we want to customize it)

2. Enable/Customize the available fields (steps 2 - 4)

3. Under *Connection Options* (step 5) - connect the output ports from the flow template to the input ports from reusable template

4. Customize the *Feed Lineage Datasources*

5. Register the template

45.2 Update template

1. Remember the template name <template_name> from NiFi

2. Create a new flow from the template <template_name>

3. Modify your flow for <template_name>

4. Delete <template_name> in NiFi template registry

5. Save flow with name <template_name>

6. In Kylo (if exists), from the Template menu, go through the edit wizard (click on the template name), so that it's reinitialized properly

45.3 Indicating Flow Failures

When Data is sent to Kylo Operations Manager it indicates if the flow file has been successful or has failed. Failures are indicated two ways

1. When the flow file passes through an 'Auto terminate on failure' relationship. In a processor in NiFi if you check the box 'Auto terminate on failure' and the flow file passes through this relationships and fails it will send the failure message to Kylo Operations Manager and fail the job/step.

2. If the NiFi connection has the word 'failure' in it and the flow files pass through that connection. The 'failure' connection name will be automatically applied by NiFi if you have a 'failure' relationship between your two processors. You can right click on a connection and edit it and change the name to include the word 'failure' if you want to always ensure that flow files which travel along that path fail the job in Kylo.
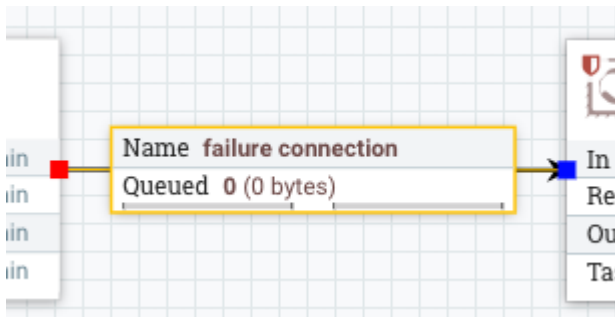
Additionally if you manually 'Empty the Queue' in NiFi it will fail those corresponding jobs in Kylo.

45.4 Available templates

Kylo provides some ready to be used templates in the Kylo repository

45.4.1 Data Ingest

Data Ingest template is used to import data from with various formats (CSV, JSON, AVRO, Parquet, ORC) into Hive tables.

S3

S3 Standard Ingest Template

JSON

There is a limitation with the JSON file format:

1. Ensure 'skip header' is turned OFF. This will allow all of the JSON data in file to be processed. Otherwise the first record will be skipped.

2. Ensure that this jar file is provided to the Validator step via the 'Extra JARs' parameter (HDP location shown for reference): /usr/hdp/current/hive-webhcat/share/hcatalog/hive-hcatalog-core.jar. Otherwise, an exception will be thrown: "java.lang.ClassNotFoundException Class org.apache.hive.hcatalog.data.JsonSerDe not found"

3. The JSON data in the file should be on one row per line.

Example: .. code-block:

```
{"id":"978-0641723445","cat":["book","hardcover"],"name":"The Lightning Thief","author
→":"Rick Riordan","series_t":"Percy Jackson and the Olympians","sequence_i":1,"genre_
→s":"fantasy","inStock":true,"price":12.50,"pages_i":384} {"id":"978-1423103349","cat
→":["book","paperback"],"name":"The Sea of Monsters","author":"Rick Riordan","series_
→t":"Percy Jackson and the Olympians","sequence_i":2,"genre_s":"fantasy","inStock
→":true,"price":6.49,"pages_i":304}
```

45.4.2 Data Transformation

Data Transformation is used to transform/wrangle data with various operations from Spark ML.

Several tables can be taken from a data source and be joined, denormalized or transformed together, to result a new data table.

Accesing S3 and other distributed filesystems

Accessing S3 from the Data Wrangler

Kylo Datasources Guide

46.1 Introduction

Kylo can manage the creation and usage of Nifi RDBMS data source configurations, through a simple Data Source UI.

46.2 JDBC

Note: Permissions for the jars are separate per the type of process running.

46.2.1 Locations

$NIFI_HOME/data/lib or any path accesible by NiFi	Needed by Nifi for the DBCPConnectionPool. The path might be erased at Nifi upgrade time.
$KYLO_HOME/kylo-services/plugin	Needed by Kylo in the schema discovery (Data Ingestion). Need to restart Kylo if added post-start
$KYLO_HOME/kylo-services/lib	Needed by Kylo wrangler (Visual Query / Data Transformation)

46.2.2 Spark configuration

While using the Visual Query / Data Transformation, you will need to make available the datasource jar. Recommended is to keep the datasource jar with the application (Kylo/Nifi), and pass it along to spark.

Depending on the Spark setup (server mode or the others), you will need to do different changes.

Server mode / Sandbox

- edit *$KYLO_HOME/kylo-services/bin/run-kylo-spark-shell.sh*
- update *KYLO_DRIVER_CLASS_PATH* with the path to the datasource jar (can be under $NIFI_HOME)

OR (not so recommended)

- update/append *$SPARK_HOME/conf/spark-defaults.conf* with the path value. Values can be appended with ":" .This file should be referenced by spark-submit, or it's referenced by */opt/kylo/kylo-services/bin/run-kylo-spark-shell.sh*, which passes the values like *spark-submit … –driver-class-path /path-to-oracle-jdbc/:/path-to-other-jars/*

Non-server mode

- edit *$KYLO_HOME/kylo-services/spark.properties*
- add to *spark.shell.sparkArgs* the *–jar /path-to-datasource-jdbc/*

You can find more *information here <http://kylo.readthedocs.io/en/latest/installation/KyloSparkProperties.html>*

46.3 Configuration examples

46.3.1 Oracle

```
Database Connection URL = jdbc:oracle:thin:@oracle:1521
Database Driver Class Name =  oracle.jdbc.OracleDriver
User = <user>
Password = <password>
Database Driver Location = /opt/nifi/oracle/oracle-jdbc.jar (needs to be accesible by␣
↪Nifi)
```

Note: Oracle tables are only in UPPERCASE

46.3.2 MariaDB / MySQL

```
Database Connection URL = jdbc:mariadb://mariadb:3306
Database Driver Class Name =  org.mariadb.jdbc.Driver
User = <user>
Password = <password>
Database Driver Location = /opt/nifi/mysql/maria-jdbc.jar (needs to be accesible by␣
↪Nifi)
```

(OPT) Specify the password in the Kylo application properties file Update /opt/kylo/kylo-services/conf/application.properties with nifi.service.<datasource_name>.password=<password>

46.4 Perfomance considerations while importing data

Consider to use the Sqoop import processor for performance gains

---

CHAPTER 47

Feed Lineage Configuration

47.1 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 Details

DETAILS PROFILE LINEAGE SLA VERSIONS

source → Event

destination

Directory
/var/dropzone/allevents\d{1,3}.csv

source → Venue

destination

Directory
/var/dropzone/venue.csv
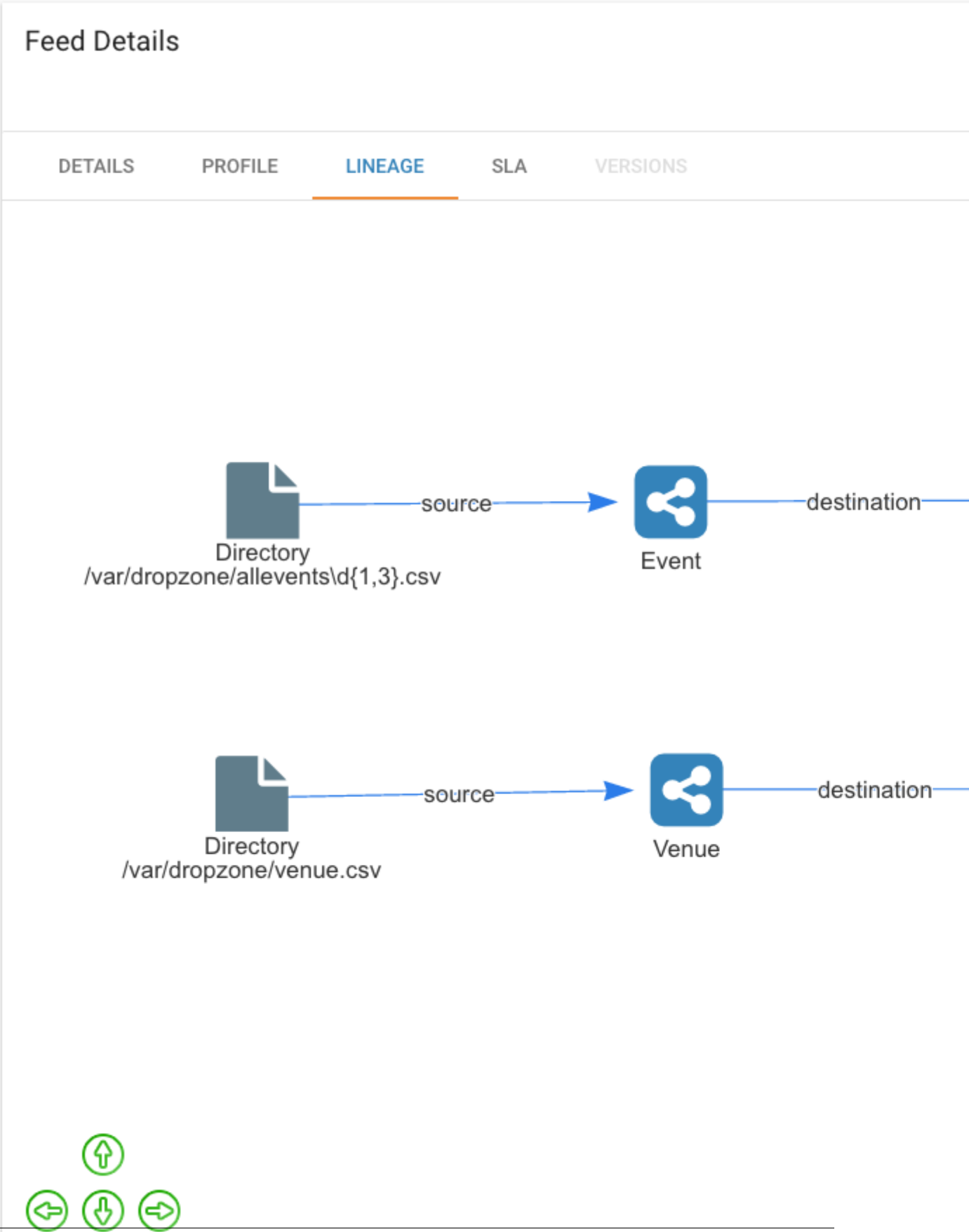
47.2 Feed Connections

47.3 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.

47.4 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 than it will be connected.

47.5 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**.

47.6 How it works

47.6.1 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.

47.6.2 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.

47.6.3 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 definitio |
| POST | /v1/feedmgr/feeds/update-feed-lineage-styles | Updates the feed lineage sty |

To list the metadata store of defined datasources you can use this REST call

http://localhost:8400/proxy/v1/metadata/datasource/datasource-definitions

47.6.4 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"
}
```

47.6.5 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.

47.7 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

47.8 Styling the Feed Lineage User Interface

Feed Lineage uses a JavaScript framework *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

Accessing S3 from the Data Wrangler

48.1 Problem

You would like to access S3 or another Hadoop-compatible filesystem from the data wrangler.

48.2 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

CHAPTER 49

S3 Standard Ingest Template

Table of Contents

49.1 Problem

You would like to ingest data from a S3 data source into Hive tables backed by S3 external folders without the data files traveling through the NiFi edge nodes.

49.2 Introduction

The Data Ingest S3 template is a variation of the standard Data Ingest template within Kylo. The standard template utilizes HDFS backed hive tables, accepts inputs from local files, and is designed to run on a Cloudera or Hortonworks sandbox. By contrast, the Data Ingest S3 template utilizes S3 backed hive tables, accepts inputs from an S3 bucket and is designed for use on an AWS stack utilizing EC2 and EMR. Additionally the template has improved performance in that data on s3 is not brought into the Nifi node.i In order to accommodate these changes, the ExecuteHQLStatement processor has been updated and a new processor, CreateElasticsearchBackedHiveTable, has been created.

49.3 1. S3 Data Ingest Template Overview

The template has two parts. The first is a non-reusable part that is created for each feed. This is responsible for getting the input location of the data in S3 as well as setting properties that will be used by the reusable portion of the template. The second is the reusable template. The reusable template creates the hive tables. It also merges, validates, profiles, and indexes the data.

The template is very similar to the HDFS standard ingestion template. The differences are outlined in the following sections.

49.3.1 1.1 Template processors pull defaults from application.properties

Creating feeds from the S3 template is simplified by adding default values into Kylo's /opt/kylo/kylo-services/conf/application.properties.

config.s3ingest.s3.protocol The protocol to use for your system. e.g. The hortonworks sandbox typically uses "s3a", EMR using an EMRFS may use "s3"

config.s3ingest.es.jar_url The location of the elasticsearch-hadoop jar. Use an S3 location accessible to the cluster.

config.s3ingest.apach-commons.jar_url The location of the commons-httpclient-3.1.jar. Use an S3 location accessible to the cluster.

config.s3ingest.hiveBucket This property is the name output bucket where the data ends up. Hive will generate the folder structure within it. Note: This bucket must have something in it. Hive cannot create folders within an empty S3 bucket.

config.s3ingest.es.nodes A comma separated list of Elasticsearch nodes that will be connected to.

For Example settings see below.

49.3.2 1.2 Non-reusable portion of template

1.2.1 List S3

Rather than fetching the data and bringing it into the Nifi node the first few properties get the location of the input data and pass the data location to subsequent processors.

Bucket This is the S3 bucket where the input data is located. Note: The data files should be in a folder at the root level of the bucket.

Region The region of the input S3 bucket.

Prefix The "path" or "sub directory" within the bucket that will receive input files. Be sure the value ends with a trailing slash.

1.2.2 Initialize Feed Parameters

Just like in the Standard ingestion template, this processor sets the attributes that will be used by the reusable portion of the template. There are several parameters that have been added to accommodate changes made to the template for S3 integration:

InputFolderName:=<the path portion of the filename> The input folder name will be used by the create feed partition processor in the reusable flow.

s3ingest.apache-commons.jar_url:=${config.s3ingest.apache-commons.jar_url} The location of the commons-httpclient.jar. Use an S3 location accessible to the cluster.

s3ingest.es.jar_url:=${config.s3ingest.es.jar_url} The location of the elasticsearch-hadoop.jar. Use an S3 location accessible to the cluster.

s3ingest.hiveBucket:=${config.3ingest.hiveBucket} This property is the name output bucket where the data ends up. Hive will generate the folder structures within it. Note: Hive cannot create folders into a fresh bucket that has not had objects written to it before. Prime the pump on new S3 buckets by uploading and deleting a file.

s3ingest.es.nodes:=${config.s3ingest.es.nodes} The comma separated list of node names for your elasticsearch nodes.

s3ingest.s3.protocol:=${config.s3ingest.s3.protocol} The protocol your cluster will use to access the S3 bucket. (e.g. 's3a')

1.2.3 DropInvalidFlowFile

When ListS3 scans a bucket, the first time it sees an object that represents the folder you specified in the Prefix it creates a flow file. Since this flow file is not a data file it will not process correctly in the flow and should be removed.

1.2.4 Initialize Cleanup Parameters

The clean up flow needs to know the name of the Hive bucket in order to clean it so the s3ingest.hiveBucket property has been added to this processor.

49.3.3 1.3 Reusable portion of Template

1.3.1 Register Tables

This processor creates S3 backed hive tables for storing valid, invalid, feed, profile, and master data. Feed Root Path, Profile Root Path, and Master Root Path define the location of their respective tables. Each of these properties will use the protocol you specified in s3ingest.protocol (s3, s3n, or s3a). The protocol must be supported by you cluster distribution.

1.3.2 Route if Data to Create ES Table

This processor routes the flow to the CreateElastisearchBackedHiveTable processor if the metadata.table.fieldIndexString property has been set. Otherwise, the CreateElastisearchBackedHiveTable processor is skipped.

1.3.3 CreateElasticsearchBackedHiveTable

This processor creates an elasticsearch backed hive table for indexing data that will be searchable from with in the Kylo UI. A description of this processor and it's properties can be found here: CreateElasticsearchBackedHiveTable Create Feed Partition In the statement for this processor the protocol for the s3 location may need to be updatad to use a protocol supported by the distribution being used.

1.3.4 Set Feed Defaults

The following property has been modified:

filename The filename property will later be used by Failed Flow processor when the flowfile is placed into the temp location. Since filename coming from S3List in the feed flow includes path information, it is stripped of that here.

1.3.5 Create Feed Partition

The ALTER TABLE statement has been modified to include the InputFolderName

1.3.6 ExecuteHQLStatement

We have updated the ExecuteHQLStatement processor to run Hive statements they just need to be separated by a semi-colon (";"). This allows us to add the elasticsearch-hadoop jar using the config.s3ingest.es.jar_url property. This particular processor inserts the data to be indexed into the elasticsearch backed hive table. It executes the following statements:

```
ADD JAR ${config.s3ingest.es.jar_url};
ADD JAR ${config.s3ingest.apache-commons.jar_url};
INSERT INTO TABLE ${category}.${feed}_index SELECT ${metadata.table.fieldIndexString},
↪processing_dttm FROM ${category}.${feed}_valid
```

1.3.5 Merge Table

The Merge Table processor will merge the incoming data with the master table, based on the merge strategy you choose.

1.3.4.1 Sync Merge Strategy

If you encounter an error similar to:

```
2017-06-21 20:50:42,430 ERROR [Timer-Driven Process Thread-4] c.t.ingest.
→TableMergeSyncSupport Failed to execute alter table `category_name`.`feed_name_
→1498078145646` RENAME TO `catgeory_name`.`feed_name` with error
java.sql.SQLException: Error while processing statement: FAILED: Execution Error,
→return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. Unable to alter table.
→Alter Table operation for <category_name>.<feed_name>_1498078145646 failed to move
→data due to: 'Renaming s3a://${hiveS3Bucket}/${hive.root.master}/<category_name>/
→<feed_name>_1498078145646 to s3a://hiveS3Bucket/${hive.metastore.warehouse.dir}/$
→{category_name}.db/<feed_name> failed' See hive log file for details.
```

Note that hive.root.master is a feed property and that hive.metastore.warehouse.dir is a property from your hive-site.xml. In versions of Hive prior to 2.2.0 the HDFS location of a managed table, with a LOCATION clause, will be moved and that Hive derives the new location using the hive.metastore.warehouse.dir and the schema_name with a .db suffix. Be sure that you have set the properties `mapred.input.dir.recursive=true` and `hive.mapred.supports.subdirectories=true` in your hive-site.xml.

1.3.6 DeleteS3Object

This processor replaces the RemoveHDFSFolder processor in standard ingest. It is analgous in that it takes the attributes from earlier in the flow and uses them to calculate the objects in the S3bucket that need to be removed and performs the delete operation.

49.4 2. Sandbox Walk-Through

49.4.1 2.1 Prerequisites

Download the required JARS for Hive to write table data to ElasticSearch. Using the links below find the jars need and place them in a folder within your hive bucket (or other S3 bucket). Make them public. In the end you should have jars available in S3 and the following commands should produce a good result:

```
aws s3 ls s3://hive-bucket/jars/elasticsearch-hadoop-5.4.0.jar
aws s3 ls s3://hive-bucket/jars/commons-httpclient-3.1.jar
```

49.4.2 2.2 Launch an EC2 instance using the Sandbox AMI

The S3 template was developed using the 0.8.1 sandbox but relies on code changes to be released in the 0.8.2 release. Go to AWS Market place and find the 0.8.2 or later sandbox for your region and launch the instance. Wait 15 minutes or more for nifi service and kylo services to start. Now shut down Nifi so we can change cluster configs and will need to refresh the NiFi connections to the cluster. Shut down Kylo so we change the application properties later.

```
service nifi stop
/opt/kylo/stop-kylo-apps.sh
```

49.4.3 2.3 Configuring core-site.xml and hive-site.xml

In the core-site.xml where your data is to be processed make sure that your fs.s3 properties are set.

Note:

- for s3 use `fs.s3.awsAccessKeyId` and `fs.s3.awsSecretAccessKey`

- for s3n use `fs.s3n.awsAccessKeyId` and `fs.s3n.awsSecretAccessKey`

- for s3a use `fs.s3a.access.key` and `fs.s3a.secret.key`

Depending on what distribution you are using the supported protocol may be different (s3, s3n) in which case you would need to use the equivalent property for that protocol. Import the template using kylo-ui making sure to import the reusable portion as well as overwriting any previous versions of the template.

Warning: There are times when AWS SDK will consult the 's3' properties for the keys, regardless of the protocol you use. To work around the problem define s3 properties in addition to your protocol properties.

Open Ambari and go to HDFS -> Configs -> Advanced -> Custom core-site section. Add the fs.s3a access properties.

```
fs.s3.awsAccessKeyId=XXX
fs.s3.awsSecretAccessKey=YYY
fs.s3a.access.key=XXX
fs.s3a.secret.key=YYY
```

Go to Hive -> Configs -> Advanced -> Custom hive-site section. Add the mapred.input.dir.recursive and hive.mapred.supports.subdirectories properties.

```
mapred.input.dir.recursive=true
hive.mapred.supports.subdirectories=true
```

Stop all services in the cluster. Start all services.

49.4.4 2.4 Get Nifi Ready

```
service nifi start
```

Go into Nifi UI and open up the Process Group Configuration and create a new AWSCredentialsProviderControllerSer-vice under the Controller Services tab. This service will be utilized by the various S3 processors to access the config-ured S3 buckets. Add your Access Key and Secret Key to the named parameters.

49.4.5 2.5 Get Kylo Ready

Edit /opt/kylo/kylo-services/conf/elasticsearch.properties and edit your settings.

Change elasticsearch.host to be same as your host in use by the template, if not already done. e.g.

```
search.host=localhost
search.clusterName=demo-cluster
```

Edit /opt/kylo/kylo-services/conf/application.properties and edit your settings. Append your template defaults. Example settings:

```
config.s3ingest.s3.protocol=s3a
config.s3ingest.hiveBucket=hive-bucket
config.s3ingest.es.jar_url=s3a://hive-bucket/jars/elasticsearch-hadoop-5.4.0.jar
config.s3ingest.apache-commons.jar_url=s3a://hive-bucket/jars/commons-httpclient-3.1.
↪jar
config.s3ingest.es.nodes=localhost
```

Start Kylo

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

49.4.6 2.6 Import the Template

Go to Admin -> Templates section of Kylo. Import the 'S3 Data Ingest' bundle from the kylo source repo path: *samples/templates/nifi-1.0/s3_data_ingest.template.zip*

49.4.7 2.7 Create the Data Ingest Feed

Create a category called "S3 Feeds" to place your new feed. Create a feed and provide the following feed inputs:

Bucket This is the name of your S3 bucket for input data. e.g. "myInputBucket"

Region This is the region where your servers operate. e.g. us-east-1

s3ingest.hiveBucket This is the name of your S3 bucket for the various hive tables e.g. "myHiveBucket". It appears twice as it will be initilaized for the feed flow and the cleanup flow. It should be defaulted to the value you set in application.properties.

prefix This is the folder in the S3 input bucket to search for input files. The default bucket will look in a folder with the same system name as the feed you are creating: "${metadata.systemFeedName}/"

49.4.8 2.8 Test the Feed

Put a data file in your input bucket. Check Kylo to ensure your feed ran successfully!

49.5 3. Further Reference

- Configure Apache Hive to Recursively Search Directories for Files
- Hadoop-AWS module: Integration with Amazon Web Services
- LanguageManual DDL: Rename Table
- Maven Central: Elasticsearch Haddop Jars
- Maven Central: Apache Commons HTTP Jars

Azure Standard Ingest Template

Table of Contents

50.1 Problem

You need to modify Kylo's standard ingest template for Hive tables backed by the Azure Blob (WASB) store.

50.2 Intro

This guide provides the basics for modifying Kylo's existing standard ingest template for use with the Azure Blob (WASB) store. For details on how to modify an existing template, please see the video tutorials on http://kylo.io.

50.3 Configuration

The resulting Data Ingest template will be a slightly modified version of the standard Data Ingest template included with Kylo.

Note:

- HD Insight clusters with Azure Blob Storage as a default storage has all the necessary properties set so default resource file, i.e. /etc/hadoop/conf/core-site.xml, can be used.

=============================-

To access Azure Blob Storage from HDFS, create an Hadoop Configuration XML file which, at minimum, will contain following properties:

Property	Value	Note
fs.azure.account.key.<your-account>.blob.core.windows.net	access key to your Azure Blob Storage's account (MSDN documentation)	See Hadoop Azure documentation for more options for how to specific access keys
fs.defaultFS	wasb://@.blob.core.windows.net	
fs.wasb.impl	org.apache.hadoop.fs.azure.NativeAzureFileSystem	

50.4 Modify HDFS processor in template

Load the existing reusable Data Ingest template into NiFi, locate the HDFS processor and modify the following properties and reimport the template into Kylo:

Property	Note
Hadoop Configuration Resources	Comma separated paths to XML configuration files. See Hadoop Azure documentation for more options for how to specific access keys
Directory	absolute / relative path, within default FS, for writing / reading files
Additional Classpath Resources	path to directory containing additional JARs needed by WASB (usually hadoop-azure-2.7.3.jar,azure-storage-2.0.0.jar

50.4.1 Known Limitations

50.4.2 Default FS

Because HDFS processors are using value of fs.defaultFS property in connection with processor's Directory property to figure out where to write/read files, this functionality can be limiting in terms when you need to copy/move files between various distributed file systems (DFS), using HDFS processors, within a single NiFi flow.

To overcome this limitation you can create minimal Hadoop configuration resource for the other FS and specify it in the list of files in Hadoop Configuration Resources property of HDFS processor. This will change the default FS for this single processor and thus allows to use a different DFS.

50.4.3 Single container

With previous limitation, Default FS, is a closely related a limitation on a single container - fs.defaultFS property contains also the container name. Way to overcome this limitation is the same as for default FS, i.e. create a copy of minimal Hadoop Configuration Resource file and change the fs.defaultFS property. Troubleshooting Server failed to authenticate the request. Make sure the value of Authorization header is formed correctly including the signature.

1. check that access keys are valid and you set them correctly in XML file (including the blob.core.windows.net after the storage account name)

2. if you run the NiFi on virtual machine make sure your OS time is synchronised (e.g. using NTP)

SUSE Configuration Changes

51.1 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.

51.2 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)

51.3 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/teradata-sles-11-hadoop/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

52.1 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

52.1.1 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:

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:

1.3 Configuration from Nifi environment variables

TODO - Help us complete this section

52.1.2 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

Validator Tuning

53.1 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.

53.2 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.

Configure Kylo & Global Search

Kylo supports Global search via a plugin-based design. Three plugins are provided out of the box:

1. Elasticsearch (rest client) [default]

2. Elasticsearch (native client)

3. Solr

54.1 Elasticsearch 5 support

Elasticsearch 5 is supported when using NiFi 1.3 (or later) and rest client. Kylo has been tested with version 5.5.1. Please refer to the rest client configuration for additional details.

54.2 Elasticsearch (rest client) [default]

Steps to configure Kylo with Elasticsearch engine (using rest client) are below. Both Elasticsearch versions 2 and 5 are supported via this plugin.

1. Include `search-esr` profile in existing list of profiles in `/opt/kylo/kylo-services/conf/application.properties`

   ```
   spring.profiles.include=native,nifi-v1,auth-kylo,auth-file,search-esr
   ```

2. Ensure that the plugin is available in `/opt/kylo/kylo-services/plugin`. This comes out-of-the-box at this location by default. It should have ownership as `kylo:users` and permissions `755`.

   ```
   kylo-search-elasticsearch-rest-0.8.3.jar
   ```

   Note: There should be only one search plugin in the /opt/kylo/kylo-services/plugin directory. If there is another search plugin (for example, kylo-search-elasticsearch-0.8.3.jar), move it to

/opt/kylo/setup/plugins/ for later use.

Reference commands to get the plugin, and change ownership and permissions:

```
mv /opt/kylo/kylo-services/plugin/kylo-search-*-0.8.3.jar /opt/kylo/setup/
↪plugins/
cp /opt/kylo/setup/plugins/kylo-search-elasticsearch-rest-0.8.3.jar /opt/
↪kylo/kylo-services/plugin/
cd /opt/kylo/kylo-services/plugin/
chown kylo:users kylo-search-elasticsearch-rest-0.8.3.jar
chmod 755 kylo-search-elasticsearch-rest-0.8.3.jar
```

3. Provide elasticsearch properties

   Update cluster properties in `/opt/kylo/kylo-services/conf/elasticsearch-rest.properties` if different from the defaults provided below.

   ```
   search.rest.host=localhost
   search.rest.port=9200
   ```

4. Create Kylo Indexes

   Execute a script to create kylo indexes. If these already exist, Elasticsearch will report an `index_already_exists_exception`. It is safe to ignore this and continue. Change the host and port if necessary.

   ```
   /opt/kylo/bin/create-kylo-indexes-es.sh localhost 9200 1 1
   ```

4. Restart Kylo Services

   ```
   service kylo-services restart
   ```

5. Steps to import updated Index Text Service feed

   (a) Feed Manager -> Feeds -> + orange button -> Import from file -> Choose file

   2a. **[Elasticsearch version 2]** Pick the `index_text_service_elasticsearch.feed.zip` file available at `/opt/kylo/setup/data/feeds/nifi-1.0`

   2b. **[Elasticsearch version 5] [This requires NiFi 1.3 or later]** Pick the `index_text_service_v2.feed.zip` file available at `/opt/kylo/setup/data/feeds/nifi-1.3`

   (a) Leave *Change the Category* field blank (It defaults to *System*)

   (b) Click *Yes* for these two options (1) *Overwrite Feed* (2) *Replace Feed Template*

   (c) (optional) Click *Yes* for option (3) *Disable Feed upon import* only if you wish to keep the indexing feed disabled upon import (You can explicitly enable it later if required)

   (d) Click *Import Feed*.

   (e) Verify that the feed imports successfully.

54.3 Elasticsearch (native client)

Steps to configure Kylo with Elasticsearch engine (using native client) are below. Only Elasticsearch version 2 is supported via this plugin.

1. Include `search-es` profile in existing list of profiles in `/opt/kylo/kylo-services/conf/application.properties`

   ```
   spring.profiles.include=native,nifi-v1,auth-kylo,auth-file,search-es
   ```

2. Ensure that the plugin is available in `/opt/kylo/kylo-services/plugin`. The plugin comes out-of-the-box at another location `/opt/kylo/setup/plugins`. It should have ownership as `kylo:users` and permissions `755`.

   ```
   kylo-search-elasticsearch-0.8.3.jar
   ```

   Note: There should be only one search plugin in the above plugin directory. If there is another search plugin (for example, kylo-search-solr-0.8.3.jar), move it to /opt/kylo/setup/plugins/ for later use.

   Reference commands to get the plugin, and change ownership and permissions:

   ```
   mv /opt/kylo/kylo-services/plugin/kylo-search-*-0.8.3.jar /opt/kylo/setup/
   ↪plugins/
   cp /opt/kylo/setup/plugins/kylo-search-elasticsearch-0.8.3.jar /opt/kylo/
   ↪kylo-services/plugin/
   cd /opt/kylo/kylo-services/plugin/
   chown kylo:users kylo-search-elasticsearch-0.8.3.jar
   chmod 755 kylo-search-elasticsearch-0.8.3.jar
   ```

3. Provide elasticsearch properties

   Update cluster properties in `/opt/kylo/kylo-services/conf/elasticsearch.properties` if different from the defaults provided below.

   ```
   search.host=localhost
   search.clusterName=demo-cluster
   search.restPort=9200
   search.transportPort=9300
   ```

4. Restart Kylo Services

   ```
   service kylo-services restart
   ```

5. Steps to import updated Index Text Service feed

   (a) Feed Manager -> Feeds -> + orange button -> Import from file -> Choose file

   (b) Pick the `index_text_service_elasticsearch.feed.zip` file available at `/opt/kylo/setup/data/feeds/nifi-1.0`

   (c) Leave *Change the Category* field blank (It defaults to *System*)

   (d) Click *Yes* for these two options (1) *Overwrite Feed* (2) *Replace Feed Template*

   (e) (optional) Click *Yes* for option (3) *Disable Feed upon import* only if you wish to keep the indexing feed disabled upon import (You can explicitly enable it later if required)

   (f) Click *Import Feed*.

   (g) Verify that the feed imports successfully.

54.4 Solr

Kylo is designed to work with Solr (SolrCloud mode) and tested with v6.5.1. This configuration assumes that you already have a running Solr instance. You can also get it from the official download page.

Steps to configure Kylo with Solr are below:

1. Include `search-solr` profile in existing list of profiles in `/opt/kylo/kylo-services/conf/application.properties`

   ```
   spring.profiles.include=native,nifi-v1,auth-kylo,auth-file,search-solr
   ```

2. Ensure that the plugin is available in `/opt/kylo/kylo-services/plugin`. The plugin comes out-of-the-box at another location `/opt/kylo/setup/plugins`. It should have ownership as `kylo:users` and permissions `755`.

   ```
   kylo-search-solr-0.8.3.jar
   ```

   Note: There should be only one search plugin in the /opt/kylo/kylo-services/plugin directory. If there is another search plugin (for example, kylo-search-elasticsearch-0.8.3.jar), move it to /opt/kylo/setup/plugins/ for later use.

   Reference commands to get the plugin, and change ownership and permissions:

   ```
   mv /opt/kylo/kylo-services/plugin/kylo-search-*-0.8.3.jar /opt/kylo/setup/
   ↪plugins/
   cp /opt/kylo/setup/plugins/kylo-search-solr-0.8.3.jar /opt/kylo/kylo-
   ↪services/plugin/
   cd /opt/kylo/kylo-services/plugin/
   chown kylo:users kylo-search-solr-0.8.3.jar
   chmod 755 kylo-search-solr-0.8.3.jar
   ```

3. Create a folder on the box where Kylo is running to store indexes for Kylo metadata. Ensure that Kylo can write to this folder.

   Reference commands to create this folder and give full permissions:

   ```
   mkdir /tmp/kylosolr
   chmod 777 /tmp/kylosolr
   ```

4. Provide solr properties

   Update cluster properties in `/opt/kylo/kylo-services/conf/solrsearch.properties` if different from the defaults provided below. The `search.indexStorageDirectory` should match with the folder location created in previous step.

   ```
   search.host=localhost
   search.port=8983
   search.indexStorageDirectory=/tmp/kylosolr
   search.zooKeeperPort=9983
   ```

5. Create collections in Solr that Kylo will use.

   Reference commands:

   ```
   bin/solr create -c kylo-datasources -s 1 -rf 1
   bin/solr create -c kylo-data -s 1 -rf 1
   ```

6. Configure Kylo collections created in previous step via Admin UI

    Reference steps:

    Navigate to Admin UI

    - http://localhost:8983/solr

    Configure collection for datasources

    (a) Select `kylo-datasources` collection from the drop down on left nav area

    (b) Click *Schema* on bottom left of nav area

    (c) Click *Add Field* on top of right nav pane

    - name: *kylo_collection*

    - type: *string*

    - default value: *kylo-datasources*

    - index: *no*

    - store: *yes*

    Configure collection for data

    (a) Select `kylo-data` collection from the drop down on left nav area

    (b) Click *Schema* on bottom left of nav area

    (c) Click *Add Field* on top of right nav pane

    - name: *kylo_collection*

    - type: *string*

    - default value: *kylo-data*

    - index: *no*

    - store: *yes*

7. Restart Kylo Services

```
service kylo-services restart
```

8. Steps to import updated Index Text Service feed

    (a) Feed Manager -> Feeds -> + orange button -> Import from file -> Choose file

    (b) Pick the `index_text_service_solr.feed.zip` file available at `/opt/kylo/setup/data/feeds/nifi-1.0`

    (c) Leave *Change the Category* field blank (It defaults to *System*)

    (d) Click *Yes* for these two options (1) *Overwrite Feed* (2) *Replace Feed Template*

    (e) (optional) Click *Yes* for option (3) *Disable Feed upon import* only if you wish to keep the indexing feed disabled upon import (You can explicitly enable it later if required)

    (f) Click *Import Feed*.

    (g) Verify that the feed imports successfully.

9. Ensure that the box running Kylo can connect to the box running Solr (if they are on separate machines). If required, open up these ports:

    - 8983

- 9983

Service Monitor Plugins

55.1 Introduction

Kylo supports pluggable Service Monitor implementations. There are a number of them available out-of-the-box, for example:

-

-

-

-

is available to implement additional Service Monitors

55.2 Monitor Services via Cloudera Manager

55.2.1 Installation

After you have installed Kylo, copy `/opt/kylo/setup/plugins/kylo-service-monitor-cloudera-service-<vers`
`jar` to Kylo plugins directory `/opt/kylo/kylo-services/plugin` and make sure plugin jar belongs to user
Kylo runs with:

```
cp /opt/kylo/setup/plugins/kylo-service-monitor-cloudera-service-<version>.
↪jar /opt/kylo/kylo-services/plugin
chown kylo:kylo /opt/kylo/kylo-services/plugin/kylo-service-monitor-cloudera-
↪service-<version>.jar
```

55.2.2 Configuration

Create service configuration file `/opt/kylo/kylo-services/conf/cloudera.properties` which belongs to user Kylo runs with and contains following properties. Do substitute values with what your Cloudera Manager is configured with:

```
clouderaRestClientConfig.username=cloudera
clouderaRestClientConfig.password=cloudera
clouderaRestClientConfig.serverUrl=127.0.0.1
clouderaRestClientConfig.port=7180
cloudera.services.status=HDFS/[DATANODE,NAMENODE,SECONDARYNAMENODE],HIVE/
↪[HIVEMETASTORE,HIVESERVER2],YARN,SQOOP
```

55.2.3 Restart Kylo

```
service kylo-services restart
```

55.3 Monitor Services via Ambari

55.3.1 Installation

After you have installed Kylo, copy `/opt/kylo/setup/plugins/kylo-service-monitor-ambari-<version>.jar` to Kylo plugins directory `/opt/kylo/kylo-services/plugin` and make sure plugin jar belongs to user Kylo runs with:

```
cp /opt/kylo/setup/plugins/kylo-service-monitor-ambari-<version>.jar /opt/
↪kylo/kylo-services/plugin
chown kylo:kylo /opt/kylo/kylo-services/plugin/kylo-service-monitor-ambari-
↪<version>.jar
```

55.3.2 Configuration

Create service configuration file `/opt/kylo/kylo-services/conf/ambari.properties` which belongs to user Kylo runs with and contains following properties. Do substitute values with what your Ambari is configured with:

```
ambariRestClientConfig.host=127.0.0.1
ambariRestClientConfig.port=8080
ambariRestClientConfig.username=admin
ambariRestClientConfig.password=admin
ambari.services.status=HDFS/[DATANODE,NAMENODE,SECONDARYNAMENODE],HIVE/
↪[HIVEMETASTORE,HIVESERVER2],YARN,SQOOP
```

55.3.3 Restart Kylo

```
service kylo-services restart
```

JMS Providers

56.1 Introduction

Kylo supports pluggable JMS implementations. There are two JMS implementations supported out-of-the-box: ActiveMQ and Amazon SQS. Both Kylo and Nifi should be configured with the same JMS implementation.

56.2 Kylo Configuration

56.2.1 ActiveMQ

ActiveMQ profile is selected by default. If you switched away from ActiveMQ and now want to restore default Kylo settings you can edit `/opt/kylo/kylo-services/conf/application.properties` and select ActiveMQ JMS implementation by adding `jms-activemq` profile to `spring.profiles.include` property, e.g.

```
spring.profiles.include=[other profiles],jms-activemq
```

In addition to selected profile, ActiveMQ configuration properties should be provided in `/opt/kylo/kylo-services/conf/application.properties`. Redelivery processing properties are now available for configuration. If Kylo receives provenance events and they have errors or are unable to attach NiFi feed information (i.e. if NiFi goes down and Kylo doesnt have the feed information in its cache) then the JMS message will be returned for redelivery based upon the following parameters. Refer to the ActiveMQ documentation, http://activemq.apache.org/redelivery-policy.html, for assigning these values

```
jms.activemq.broker.url=tcp://localhost:61616
#jms.activemq.broker.username=admin
#jms.activemq.broker.password=admin
##Redeliver policy for the Listeners when they fail (http://activemq.apache.
↪org/redelivery-policy.html)
#jms.maximumRedeliveries=100
#jms.redeliveryDelay=1000
#jms.maximumRedeliveryDelay=600000L
```

```
#jms.backOffMultiplier=5
#jms.useExponentialBackOff=false
```

56.2.2 Amazon SQS

ActiveMQ profile is selected by default. But you can switch over to Amazon SQS by replacing `jms-activemq` profile with `jms-amazon-sqs` in `/opt/kylo/kylo-services/conf/application.properties`, e.g.

```
spring.profiles.include=[other profiles],jms-amazon-sqs
```

In addition to that Amazon SQS specific properties should be provided in `/opt/kylo/kylo-services/conf/application.properties`.

```
sqs.region.name=eu-west-1
```

Amazon SQS uses `DefaultAWSCredentialsProviderChain` class to look for AWS credentials in the following order:

- Environment Variables - AWS_ACCESS_KEY_ID and AWS_SECRET_KEY
- Java System Properties - aws.accessKeyId and aws.secretKey
- Credential profiles file at the default location (~/.aws/credentials) shared by all AWS SDKs and the AWS CLI
- Instance profile credentials delivered through the Amazon EC2 metadata service

For example, add your AWS credentials to `/home/kylo/.aws/credentials`

```
[default]
aws_access_key_id=...
aws_secret_access_key=...
```

56.3 Nifi Configuration

56.3.1 Active MQ

Select `jms-activemq` profile and provide ActiveMQ specific configuration properties in `/opt/nifi/ext-config/config.properties`, e.g.

```
spring.profiles.active=jms-activemq

jms.activemq.broker.url=tcp://localhost:61616
#jms.activemq.broker.username=admin
#jms.activemq.broker.password=admin
##Redeliver policy for the Listeners when they fail (http://activemq.apache.
↪org/redelivery-policy.html)
#jms.maximumRedeliveries=100
#jms.redeliveryDelay=1000
#jms.maximumRedeliveryDelay=600000L
#jms.backOffMultiplier=5
#jms.useExponentialBackOff=false
```

56.3.2 Amazon SQS

Select `jms-amazon-sqs` profile and provide Amazon SQS specific configuration properties in `/opt/nifi/ext-config/config.properties`, e.g.

```
spring.profiles.active=jms-amazon-sqs

sqs.region.name=eu-west-1
```

Amazon SQS uses `DefaultAWSCredentialsProviderChain` class to look for AWS credentials in the following order:

- Environment Variables - AWS_ACCESS_KEY_ID and AWS_SECRET_KEY
- Java System Properties - aws.accessKeyId and aws.secretKey
- Credential profiles file at the default location (~/.aws/credentials) shared by all AWS SDKs and the AWS CLI
- Instance profile credentials delivered through the Amazon EC2 metadata service

For example, add your AWS credentials to `/home/nifi/.aws/credentials`

```
[default]
aws_access_key_id=...
aws_secret_access_key=...
```

There are four places where standard Kylo feeds need updating in Nifi to route JMS messages via Amazon SQS instead of ActiveMQ. Replace JMS processors with their Amazon SQS equivalents. Replace PublishJMS processors with PutSQS processors and ConsumeJMS processors with GetSQS processors in following feeds:

- reusable_templates -> standard-ingest
    - Register Index (PublishJMS)
    - Update Index (PublishJMS)
- system
    - index_schema_service -> Receive Schema Index Request (ConsumeJMS)
    - index_text_service -> Receive Index Request (ConsumeJms)

Database Upgrades

57.1 Overview

This guide provides details on how to update your database with each new Kylo version. Kylo supports two ways to upgrade your database:

1. Automatic upgrades

2. Manual upgrades

57.1.1 1. Automatic Upgrades

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. Just start Kylo services as normal and your database will be automatically upgraded to latest version if required.

57.1.2 2. Manual Upgrades

By default Kylo is set up to automatically upgrade its database. To manually upgrades your database:

1. Turn off automatic database upgrades

2. Generate update SQL script

3. Run generated SQL manually on your database

2.1 Turn off automatic database upgrades

Set `liquibase.enabled` to `false` in `/opt/kylo/kylo-services/conf/application.properties` if you don't want to automatically upgrade Kylo database. Also make sure your database connection properties are correct:

```
liquibase.enabled=false

spring.datasource.url=
spring.datasource.username=
spring.datasource.password=
spring.datasource.driverClassName=
```

2.2 Generate upgrade SQL script

Make sure that required database driver is on classpath in `/opt/kylo/kylo-services/lib` directory and run `/opt/kylo/setup/sql/generate-update-script.sh`.

```
/opt/kylo/setup/sql/generate-update-script.sh
```

This will generate database update SQL in your current directory called `kylo-db-update-script.sql`. This SQL script will contain all required SQL statements to update your database to next Kylo version.

2.3 Run generated SQL manually on your database

The process of executing `kylo-db-update-script.sql` SQL script will differ for each database vendor. Please consult documentation for your database on how to execute an SQL script.

Icons and Icon Colors

Icons and the colors can be configured using 2 JSON files found in the /opt/kylo/kylo-services/conf directory.

icons.json

This is an array of valid icon names. Valid names that can be used can be found here: https://klarsys.github.io/angular-material-icons/.

icon-colors.json

This is an array of objects indicating the display name and respective Hex color code.

Twitter Sentiment with Kafka and Spark Streaming Tutorial

59.1 About

This advanced tutorial will enable Kylo to perform near real-time sentiment analysis for tweets. Our Kylo template will enable user self-service to configure new feeds for sentiment analysis. The user will simply enter the list of twitter keywords to analyze (e.g. famous list of music artists). Tweets will be classified as positive, negative, or neutral based on analysis of the text. The tweet and sentiment results will be written to Hive. We will be able to monitor the streaming process in the Kylo Ops Manager module and explore the results.

59.2 How it Works

Once the user configures the new feed in Kylo, a pipeline will be generated in Apache NiFi. The tweet text will be extracted and published to a Kafka topic. A Spark streaming job will consume the message tweet from Kafka, performs sentiment analysis using an embedded machine learning model and API provided by the Stanford NLP project. The Spark streaming job then inserts result into Hive and publishes a Kafka message to a Kafka response topic monitored by Kylo to complete the flow.

In order to track processing though Spark, Kylo will pass the NiFi flowfile ID as the Kafka message key. Kylo passes the FlowFile ID to Spark and Spark will return the message key on a separate Kafka response topic. The latter utilizes the new Notify and Wait processors in NiFi 1.3.0+ which we will introduce with this tutorial.

59.3 Prerequisites

1. Download the latest Kylo sandbox. This tutorial requires NiFi 1.3.

2. Install/enable Kafka (if needed)

3. Create a twitter application account. and document your consumer key/secret and access token/secret pairs.

4. Download Stanford corenlp libraries. The specific library files are shown in the Spark configuration section (below). This tutorial used v3.7.0.

5. Download and build this useful Twitter Sentiment analysis utility. The specific library files needed are shown in the Spark configuration section (below).

59.4 Configuration

Your Twitter consumer key/secret and access token/secret pairs are needed in order to provision the template with the correct Twitter credentials. Add the following block to your Kylo application .properties file (typically located in /opt/kylo/kylo-services/conf/application.properties). Note that Kylo will automatically injects these properties into the NiFi pipeline when a new feed is provisioned:

```
### Twitter
nifi.gettwitter.consumer_key={REPLACE_WITH_YOUR_TWITTER_CONSUMER_KEY}
nifi.gettwitter.consumer_secret={REPLACE_WITH_YOUR_TWITTER_CONSUMER_SECRET}
nifi.gettwitter.access_token={REPLACE_WITH_YOUR_TWITTER_ACCESS_TOKEN}
nifi.gettwitter.access_secret={REPLACE_WITH_YOUR_TWITTER_ACCESS_SECRET}
```

Restart Kylo after applying changes to the property file.

59.5 Spark Configuration

The following JARs need to be available on the Spark2 classpath. There are different ways to achieve this but one way is to simply modify the /etc/spark2/conf/spark-defaults.conf as shown here:

```
# Add to /etc/spark2/conf/spark-defaults.conf
spark.driver.extraClassPath /path/to/lib.jar:/path/to/lib2.jar:/path/to/lib3.jar
```

The extra classpath libraries needed will depend on your specific Hadoop and Kafka installation. The following are required jar files for readability:

```
# From local machine
kafka-streams-0.10.0.2.5.5.0-157.jar
spark-examples_2.11-2.0.2.2.5.5.0-157.jar
spark-streaming-kafka-0-10_2.11-2.1.1.jar
spark-streaming_2.11-2.0.2.2.5.5.0-157.jar
ejml-0.23.jar

# From github.com/vspiewak/
twitter-sentiment-analysis_3.7.0.jar (must be built from source)
jsonic-1.2.0.jar (available in lib)
langdetect.jar (available in lib)

# From Stanford NLP
stanford-corenlp-3.7.0-models-english.jar
stanford-parser-3.7.0.jar
stanford-corenlp-3.7.0.jar
```

59.6 Twitter Sentiment template

The feed template for this tutorial is provided in Kylo github. This template will allow you to create a feed to monitor tweets based on keywords and write the sentiment results to a Hive table. Download and import the Twitter Sentiment template. into the Kylo templates UI.

59.7 Kafka response feed

This system feed will monitor a Kafka topic for flowfile ids that have been processed by our Spark job.

Download and import the Kafka Notifier feed. into the Kylo feeds.

59.8 Scripts

Create the following shell scripts in /opt/spark-receiver/ and ensure NiFi has execute permissions on the files:

1. The following shell script will start/stop our streaming application. It will only start the application if it is not currently running. Name the file: stream-submit-kafka.sh

```bash
#!/bin/bash

#extract script file then shift remaining args will be pased to scala script
arg_count="$#"
command=$1
app_name=$2
scala_file=$3
shift 3
arguments=$@

export SPARK_MAJOR_VERSION=2
spark_regex=".*SparkSubmit.*\s$app_name.*"

start() {
    if [ "$arg_count" -lt 10 ]; then
        echo "Illegal parameters. Usage ./stream-submit-kafka.sh start sentiment-app
↪path/to/script.scala {window secs} {hive table} {twitter keywords,comma-delim}
↪{kafka read topic} {kafka write topic} {broker} {zookeeper} {kafka group}
        echo "Example: ./stream-submit-kafka.sh start sentiment-app /opt/spark-
↪receiver/sentiment-job-kafka.scala 15 sentiment_17 @ArianaGrande,@justinbieber,
↪@MileyCyrus topicC topicB sandbox.kylo.io:6667 sandbox.kylo.io:2181 groupA
        exit 1
    fi
    echo "Starting process $app_name with $arguments"
    if pgrep -f "$spark_regex" > /dev/null
    then
        echo "$app_name already running"
    else
        nohup spark-shell --name "$app_name" --master local[2] --deploy-mode client \
          --queue default \
          --driver-memory 4G --executor-memory 4G \
          -i <(echo 'val args = "'$arguments'".split("\\s+")' ; cat $scala_file) &>
↪$app_name.out &
    fi
}

stop() {
    if [ "$arg_count" -lt 2 ]; then
        echo "Illegal parameters. Usage ./stream-submit.sh kill appName"
        exit 1
    fi
    if pgrep -f "$spark_regex" > /dev/null
    then
```

```bash
        echo "Killing $app_name"
        pkill -f "$spark_regex"
    else
        echo "$app_name not running"
    fi
}

status() {
    if [ "$arg_count" -lt 2 ]; then
        echo "Illegal parameters. Usage ./stream-submit.sh status appName"
        exit 1
    fi

    if pgrep -f "$spark_regex" > /dev/null
        then echo "$app_name running"
        else echo "$app_name not running"
    fi
}

case "$command" in
    status)
        status
    ;;
    start)
        start
    ;;
    stop)
        stop
    ;;
    restart)
        echo "Restarting $app_name"
        stop
        sleep 2
        start
        echo "$app_name started"
    ;;
     *)
        echo $"Usage: $0 {start|stop|restart|status|"
        exit 1
esac
exit 0
```

2. The following Scala script is our sentiment analysis Spark job. Please name the file: sentiment-job-kafka.scala

```scala
import java.util.HashMap

import org.apache.spark.examples.streaming._
import kafka.serializer.StringDecoder

import org.apache.spark.streaming._
import org.apache.spark.streaming.kafka010._
import org.apache.spark.SparkConf

import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization._
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
```

```scala
import java.util.HashMap

import org.apache.kafka.clients.producer.{KafkaProducer, ProducerConfig,
→ProducerRecord}

import org.apache.spark.SparkConf
import org.apache.spark.streaming._
import java.io._

import java.nio.charset.StandardCharsets
import scala.collection.mutable.ListBuffer

import org.apache.spark.rdd.RDD
import org.apache.spark.sql.SQLContext
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.{Seconds, StreamingContext, Time}
import com.github.vspiewak.util._
import edu.stanford.nlp.sentiment._
import java.io.IOException
import java.util.Properties

    case class TweetRecord(time: Integer, topic: String, sentiment: String, tweet:
→String)

    val durationSecs = args(0).toLong
    val tableName = args(1)
    val keywords = args(2)
    val readerTopic = args(3)
    val writerTopic = args(4)
    val brokers = args(5)
    val zookeeper = args(6)
    val group = args(7)
    println("durationSecs: " + durationSecs)
    println("tableName: " + tableName)
    println("keywords: " + keywords)

    val bKeywords = sc.broadcast(keywords.split(","))

    val clientParams = Map[String, Object](
     "bootstrap.servers" -> brokers,
     "zookeeper.connect" -> zookeeper,
     "key.deserializer" -> classOf[StringDeserializer],
     "value.deserializer" -> classOf[StringDeserializer],
     "group.id" -> group,
     "auto.offset.reset" -> "latest",
     "enable.auto.commit" -> (false: java.lang.Boolean)
    )

   val producerProps =  new java.util.Properties()

   producerProps.put("bootstrap.servers", brokers)
   producerProps.put("zookeeper.connect", zookeeper)
   producerProps.put("key.serializer", classOf[StringSerializer])
   producerProps.put("value.serializer", classOf[StringSerializer])
   producerProps.put("key.deserializer", classOf[StringDeserializer])
   producerProps.put("value.deserializer", classOf[StringDeserializer])
```

```
    StreamingExamples.setStreamingLogLevels()

    val producer = new KafkaProducer[String, String](producerProps)

    spark.sql("CREATE TABLE IF NOT EXISTS "+tableName+" (`time` int, `topic` string,␣
→`sentiment` string, `tweet` string)")

    // Create direct kafka stream with brokers and topics
    // Create context with specified batch interval
    @transient val ssc = new StreamingContext(sc, Seconds(durationSecs))

    val topics = Array(readerTopic)
    @transient val tweetStream = KafkaUtils.createDirectStream[String, String](
        ssc,
        PreferConsistent,
        Subscribe[String, String](topics, clientParams)
    )

  @transient val uuids = tweetStream.map(_.key)

  @transient val tweetStreamMapped = tweetStream.map { record: org.apache.kafka.
→clients.consumer.ConsumerRecord[String,String] =>
                                   val tweet = record.value
                                   println(tweet)
                                   // Create record for each match so tweets with␣
→multiple matches will be counted multiple times
                                   val keywords = bKeywords.value
                                   val matches = keywords.filter { (term) => (tweet.
→contains(term)) }
                                   val matchArray =  matches.map { (keyword) =>␣
→(keyword, tweet) }
                                   // Convert to listbuffer so we can flatten
                                   val matchLB = ListBuffer(matchArray: _ *)
                                   matchLB.toList
                               }.
                               flatMap(identity).
                               map { (tuple) =>
                                       val topic = tuple._1
                                       val tweet = tuple._2
                                       // Clean hashtags, emoji's, hyperlinks, and␣
→twitter tags which can confuse the model. Replace @mention with generic word Foo
                                       val tweet_clean = tweet.replaceAll(
→"(\\b\\w*RT)|[^a-zA-Z0-9\\s\\.\\,\\!,\\@]", "").replaceAll("(http\\S+)","").
→replaceAll("(@\\w+)","Foo").replaceAll("^(Foo)","")
                                       try {
                                           val sentiment = SentimentAnalysisUtils.
→detectSentiment(tweet_clean).toString.toLowerCase
                                               (topic, sentiment,tweet)

                                       } catch {
                                           case e: IOException => e.
→printStackTrace(); (tuple._1, "unknown", tweet)
                                       }
                                   }


    println("Writing results to Hive "+tableName)
    tweetStreamMapped.foreachRDD { (rdd: RDD[(String, String, String)], time: org.
→apache.spark.streaming.Time) => rdd.map( t => TweetRecord(( time.milliseconds /␣
→1000).toInt, t._1, t._2, t._3) )
```

```
                                                               .toDF()
                                                               .filter(
↪"sentiment is not null")

                                                               .write
                                                               .
↪insertInto(tableName)
                      }
 println("Sending results to Kafka topic:"+writerTopic)
   uuids.foreachRDD { rdd =>
     rdd.collect().foreach { key =>
        producer.send( new ProducerRecord[String, String](writerTopic, key, "done"))
     }
   }

   ssc.start()
   ssc.awaitTermination()

   ssc.stop()
```

59.9 Create your feed

After importing the template in Kylo, you are ready to create a feed. Create a new feed and select 'Sentiment Analysis'. Now provide the keywords as comma separated strings. Note that because the Twitter account used by the template is a free account, you are limited to filtering on specific keywords. This template has hardcoded a set of keywords of common twitter accounts: @katyperry, @justinbieber,@taylorswift13,@rihanna,@realDonaldTrump. Your feed may include any subset or combination of these. You can alter the superset of keywords in the template. If you have a full Twitter account, you could use the Firehose endpoint and then perform your filtering in Spark.

59.10 Monitor your feed in Kylo

From the Ops Manager, your feed will appear as a Streaming feed.
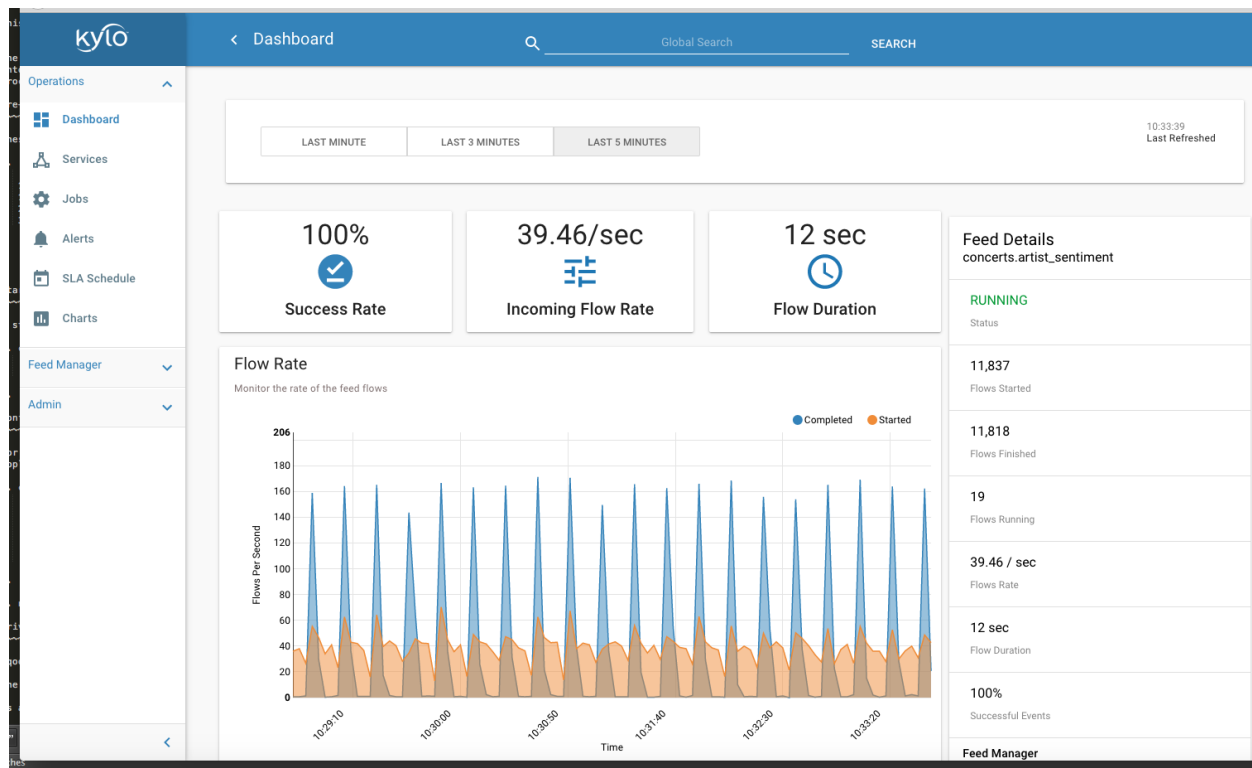
Ambari Service Monitor Plugin

60.1 Purpose

The Ambari Service Monitor reports the status of Ambari services in Kylo Operations Manager.

60.2 Enable Plugin

To enable the plugin please do the following

1. Create a new file /opt/kylo/kylo-services/conf/ambari.properties. Ensure the owner of the file is kylo

2. Add the following to the ambari.properties file.

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

3. Copy the /opt/kylo/setup/plugins/kylo-service-monitor-ambari-<<VERSION>>.jar to /opt/kylo/kylo-services/plugin

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

4. Restart kylo-services

Contributing to Kylo

61.1 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.

61.2 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.

61.3 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.

61.4 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.

61.4.1 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.

61.4.2 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.

61.5 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);
```

61.6 Pull Requests

To get started go to GitHub and fork the Kylo repository.
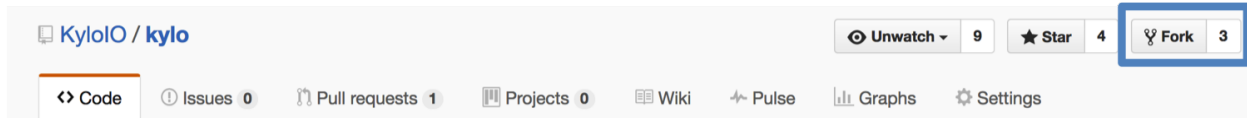
This will create a copy of the repository under your personal GitHub account. You will have write permissions to your repository but not to the official Kylo repository.

61.6.1 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.

61.6.2 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
```

61.6.3 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.

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.

62.1 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.

62.2 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.

62.3 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
```

62.4 Install Virtual Box

Use this link to download and install the DMG file to install Virtual Box:

> *https://www.virtualbox.org/wiki/Downloads*

62.5 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
```

62.6 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"*.

2. Import from your IDE using the "*https://github.com/kyloio/kylo.git*" URL.

62.7 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.

62.8 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`

62.9 Install and Configure the Hortonworks Sandbox

Follow the guide below to install and configure the Hortonworks sandbox:

> *Hortonworks Sandbox Configuration*

62.10 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>.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.

62.11 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.

62.12 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.

62.13 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.

62.14 Web Development

Most of the Kylo UI depends on and but a few parts have been upgraded to and . New plugins should be written in Typescript and use Angular 2 for future compatibility.

NPM should be used to configure and start your web development environment:

1. Install NPM in your development environment:

   - apt-get install npm (Debian / Ubuntu)

   - brew install npm (Mac)

2. Install the development packages:

```
$ cd kylo/ui/ui-app
$ npm install
```

3. Start Kylo and the development server:

```
$ service kylo-services start
$ service kylo-ui start
$ npm run start
```

4. A new browser window will open showing the Kylo UI. Any changes you make will automatically refresh the page with the new changes.

If you will be editing stylesheets then you will need to additionally configure IntelliJ to compile SCSS files:

1. Install the File Watchers plugin and restart IntelliJ.

2. Go to Preferences -> Tools -> File Watchers.

3. Click the + plus sign at the bottom and create a SCSS file watcher.

4. Change the Arguments to: `--output $FileDir$ $FilePath$`

5. Change the Output paths to refresh to: `$FileNameWithoutExtension$.css`

62.15 Angular Material Notes

There are a few notes worth mentioning about using AngularJS Material:

1. Do not use `layout-row` and `layout-wrap` with percents. It has been broken on Safari for a while now with current plan to be fixed only in Angular 4.x.

2. Do not refer to Angular model in plain HTML `style` element, it is broken on IE. Instead use Angular `ng-style` element which works on all browsers like so `ng-style="{'fill':controller.fillColor}"`

3. Do not use `flex` element where you don't have to. Browsers will usually flex elements correctly. This is to minimise the occurrence of `flex` being required by Safari while breaking layout on IE.

CHAPTER 63

Plugin APIs

63.1 Kylo UI

63.1.1 Writing Spark Function Definitions

Tern defines the definitions file format for displaying the list of functions, providing auto-completion, and showing hints. Kylo extends this format by providing additional fields that describe how to convert the function into Scala code.

The definitions are loaded from json files matching `*spark-functions.json` in the Kylo classpath and merged into a single document to be used by the Kylo UI. Duplicate functions are ignored.

Data Types

An expression may consist of may different data types but the end result is to produce a *DataFrame*.

Arrays

An array is a collection of zero or more literals of the same type.

Booleans

A Boolean value is either *true* or *false*.

Columns

A *Column* is an object that represents a *DataFrame* column. It has an optional *alias* property which defines the name of the column.

Numbers

Numbers can be either literal integers or floating-point values. They will be automatically converted to a *Column* if required.

Objects

An Object is any Scala class type. No conversions are performed on objects.

Strings

Strings should be enclosed in double quotes. They are automatically converted to a *Column* if required.

Definitions

Function definitions are declared in a JSON document that maps a function name to a definition. Each definition is an object with special directives indicating the function arguments, return value, documentation, and a Spark conversion string. The JSON document also has a special directive with the name of the document.

```
{
  "!name": "ExampleDefinition",
  "add": {
    "!type": "fn(col1: Column, col2: Column) -> Column",
    "!doc": "Add two numbers together.",
    "!spark": "%c.plus(%c)"
    "!sparkType": "column"
  }
}
```

The above document is named *ExampleDefinition* as declared by the `!name` directive. It contains a single function named *add* and the `!type` directive indicates it takes two *Column* arguments and outputs a *Column*. The strings for the `!doc` and `!type` directives will be displayed in the autocomplete menu. The `!spark` directive defines the Spark conversion string for converting the expression to Spark code, and the `!sparkType` directive indicates is produces a Column object.

Spark Conversion String Syntax

The conversion string consists of literal characters that are copied as-is to the Spark code and conversion specifications that either consume one of the function arguments.

The conversion specifications have the following syntax:

```
%[flags]conversion
```

Conversion

The following conversions are supported:

Type Specifier	Description	Example Spark Result
b	Expects the argument to be a literal boolean, either *true* or *false*. The result is a literal boolean.	true
c	The result is a *Column* object. All input types are supported.	new Column("mycol")
d	Expects the argument to be a literal integer. The result is a literal integer.	123
f	Expects the argument to be a literal floating-point number. The result is a literal double.	123.5
o	The result is a Scala object.	
r	The result is a *DataFrame* object.	
s	Expects the argument to be a literal of any type. The result is a literal string.	"myval"

Flags

The following flags are supported:

Flag	Description	Example Spark Result
?	The conversion is optional and will be ignored if there are no more arguments left to consume.	
*	The conversion should consume all remaining arguments, if any. Useful for var-arg functions.	new Column("arg1"), new Column("arg2")
,	The conversion should begin with a comma.	, new Column("arg1")
@	The result is an array of the specified type.	Array("value1", "value2")

Spark Types

The `!sparkType` directive indicates the type produced by the `!spark` directive.

Type	Description
array	A Scala array.
column	A Spark SQL *Column* object.
dataframe	A Spark SQL *DataFrame* object.
literal	A Scala literal value.
transform	A function that takes a *DataFrame* and returns a *DataFrame*.

Any other type is assumed to be a class type.

Column Functions

These functions are instance methods of the *Column* class.

as fn (alias: string) -> Column Gives the column an alias.

cast fn (to: string) -> Column Casts the column to a different type.

over fn (window: WindowSpec) -> Column Define a windowing column.

Resources

Additional information on the Tern JSON format is available in the section of the Tern docs.

63.2 Kylo Services

Kylo REST API

64.1 Documentation

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..

64.2 Authentication

REST API calls require basic authorization header.

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
```

Cloudera Docker Sandbox Deployment Guide

66.1 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.

66.2 Prerequisites

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

66.3 Installation

66.3.1 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>
```

66.3.2 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. Create Cloudera docker script:

```
$ 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 755 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.

66.3.3 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.

66.3.4 Step 4: Install Kylo in the Docker Container

1. Follow the Setup Wizard guide

Setup Wizard Deployment Guide

2. Login to Kylo at <EC2_HOST>:8400, and NiFi at <EC2_HOST>:8079.

66.4 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.

66.5 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.

Hortonworks Sandbox Configuration

67.1 Introduction

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

67.2 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

67.3 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.

```
VBox GUI > Settings > Shared Folders > Add
```

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

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

67.4 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:

67.5 Startup the Sandbox

1. Start the sandbox.

2. SSH into the sandbox.

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

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.

Kerberos Installation Example - Cloudera

Note: This document was developed by going through these steps in a base CDH Sandbox 5.4, not the newer Kylo sandbox on Cloudera distribution.

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.

68.1 Prerequisite

68.1.1 Java

All client node should have java installed on it.

```
$ java -version
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
```

68.1.2 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
```

68.1.3 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
```

68.2 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

[libdefaults]
   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 /
```

68.3 Install Kerberos on Cloudera Cluster

1. Login to Cloudera Manager and Select Security option from Administration tab.

2. Click on Enable Kerberos.
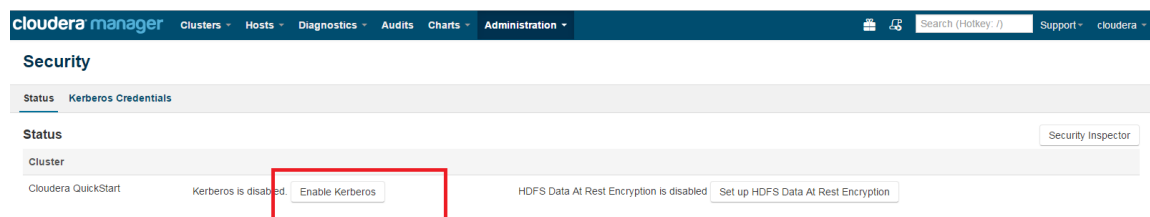
3. Select each item and click on continue.

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
```


5. Select checkbox Manage krb5.conf through cloudera manager.

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.

68.4 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
```

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.

69.1 Prerequisite

69.2 Java

Java must be installed on all client nodes.

```
$ java -version
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
```

69.3 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
```

69.4 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
```

69.5 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 sandbox.hortonworks.com. 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
```

```
[libdefaults]
  default_realm = sandbox.hortonworks.com
  dns_lookup_realm = false
  dns_lookup_kdc = false
  ticket_lifetime = 24h
  renew_lifetime = 7d
  forwardable = true

[realms]
  sandbox.hortonworks.com = {
  kdc = sandbox.hortonworks.com
  admin_server = sandbox.hortonworks.com
  }
```

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

or

systemctl start krb5kdc.service
systemctl start kadmin.service
```

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

or
```

```
systemctl enable krb5kdc.service
systemctl enable kadmin.service
```

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:

```
*/admin@sandbox.hortonworks.com *
```

12. After editing and saving the kadm5.acl file, restart the kadmin process.

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

or

systemctl restart kadmin.service
systemctl restart krb5kdc.service
```

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 /
```

69.6 Install Kerberos on an HDP Cluster

1. Open Ambari and then go to admin tab and select Kerberos.
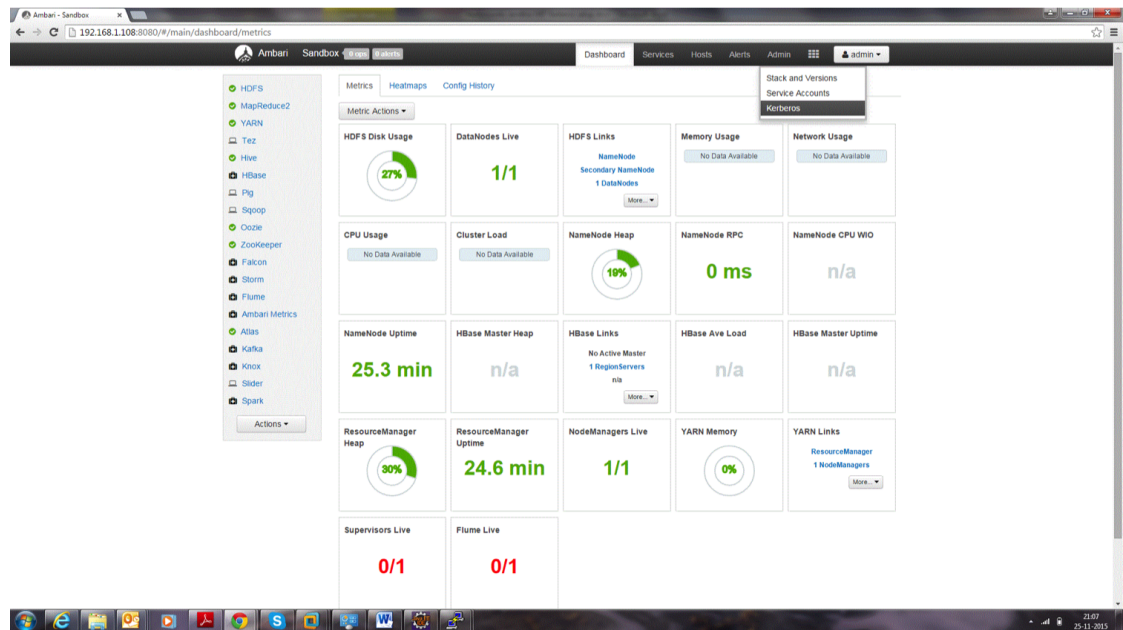
2. Click on enable Kerberos. Then following screen will display. Tick the checkboxes as shown in this screenshot, then click Next.
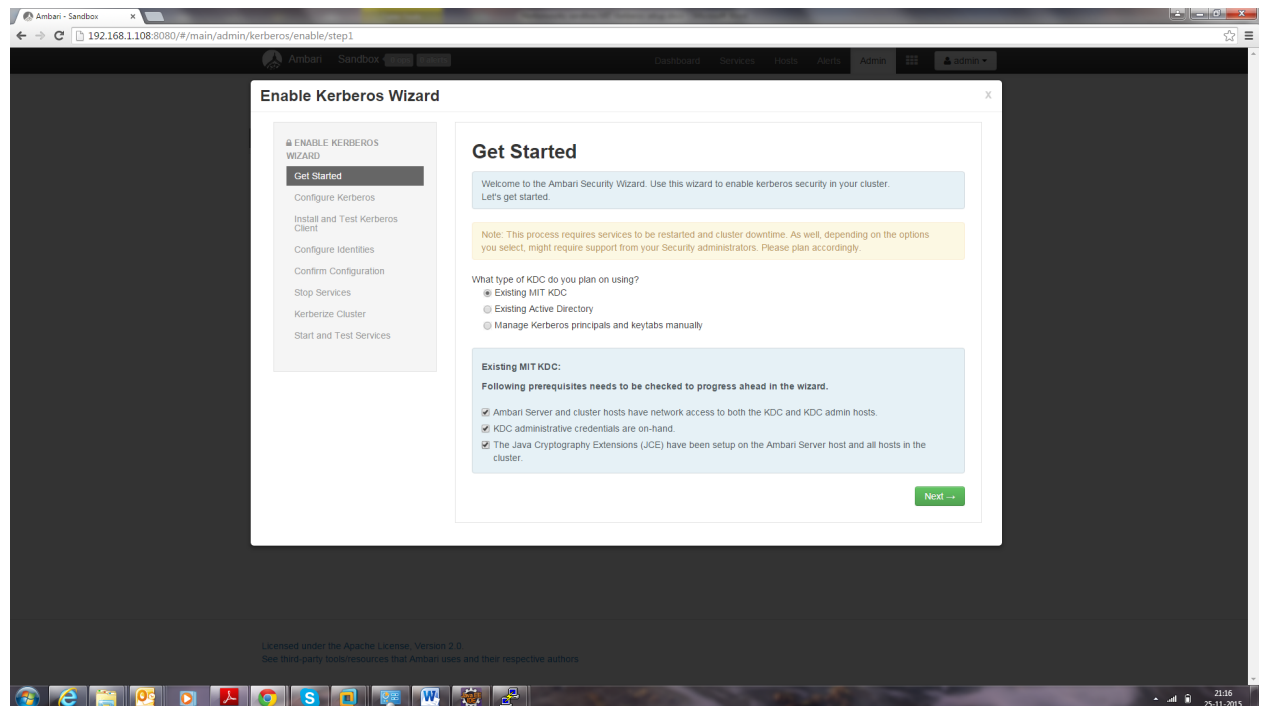
3. Put sandbox.hortonworks.com in the KDC tab and click to test the KDC connection. Then, in Kadmin, put sandbox.hortonworks.com as host and admin principal as *admin/admin@sandbox.hortonworks.com*, and enter the password created in step 7.

   Leave the advanced Kerberos-env and advanced krb5-conf as it is. And click **Next**.

4. Download the .csv file and save it.
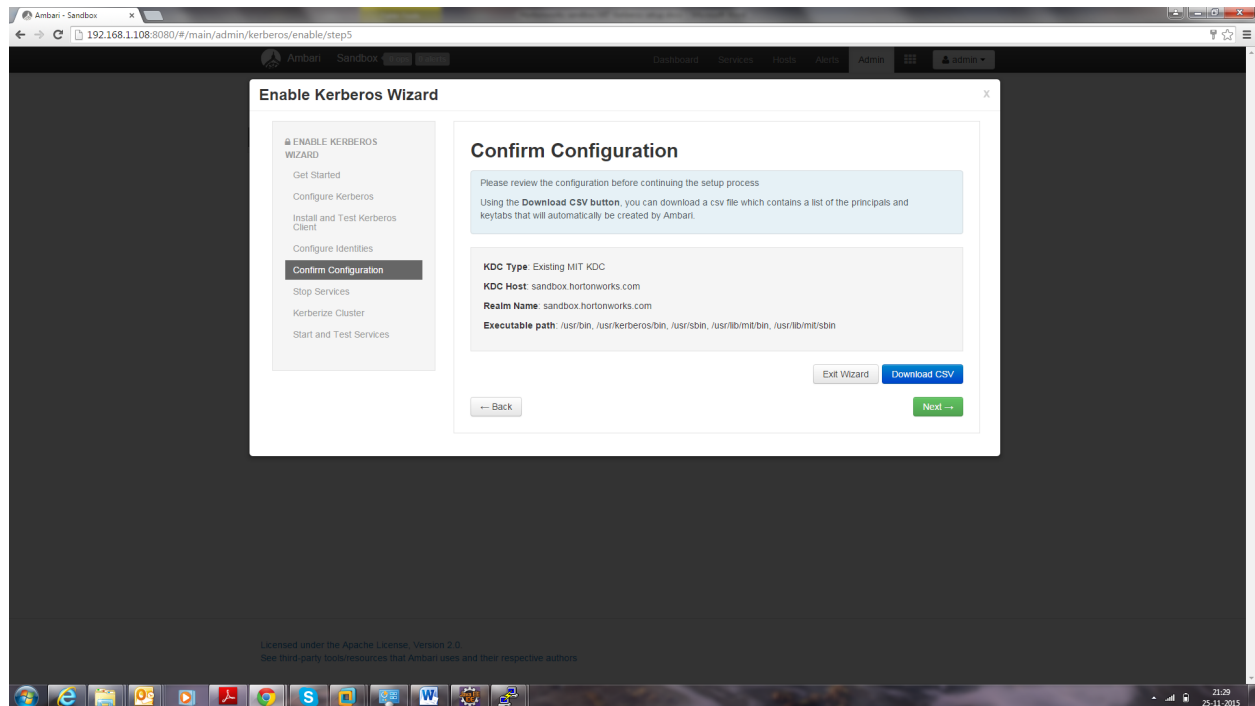
5. Click Next through the end of the process, until finally you can click **Complete**.
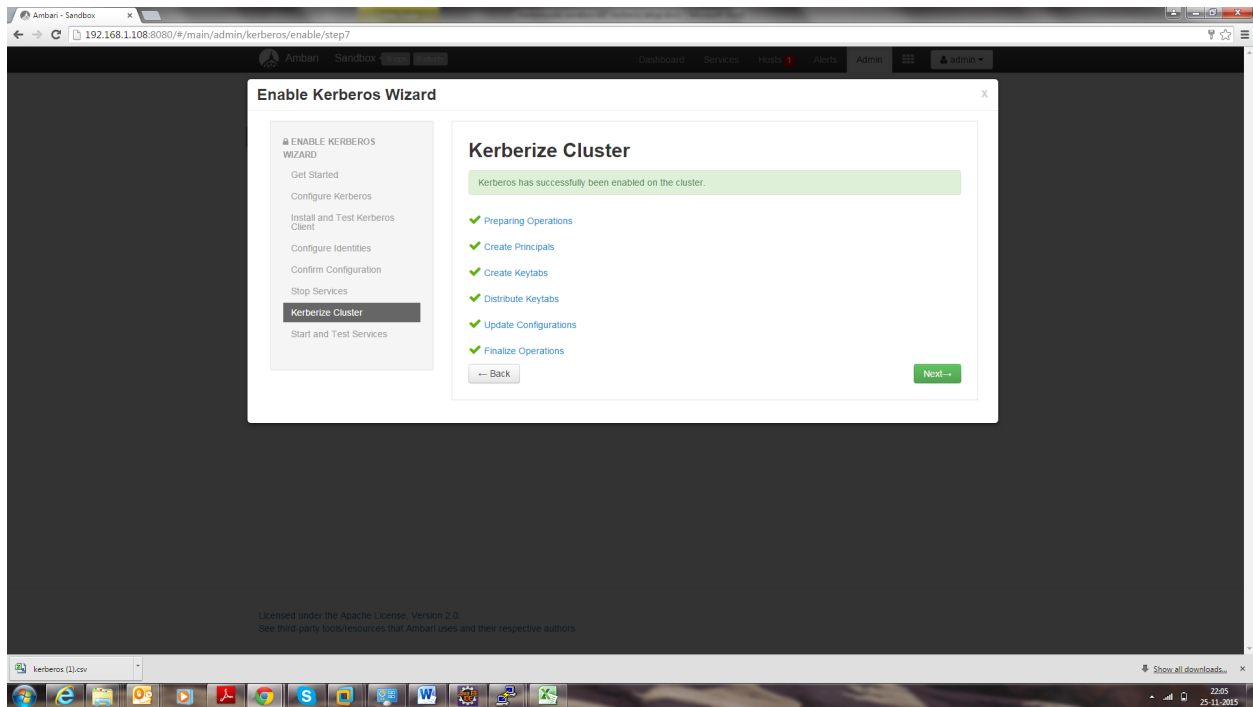
Kerberos Installation Example - Cloudera

Make sure all services started properly. Kerberos is successfully installed on the cluster.

Events

Kylo publishes events to a message bus that you can subscribe to and react to changes in the system with custom plugins.

Below is a listing of the events Kylo publishes.

Event	Description
CategoryChangeEvent	Called when a Category is created, updated, or deleted
FeedChangeEvent	Called when a Feed is created, updated, or deleted
FeedProperty-ChangeEvent	Called when a user updates the generic set of properties on a feed. See the Nifi processor PutFeedMetadata
PreconditionTrig-gerEvent	Called when a precondition is fired for a feed
CleanupTriggerEvent	Called when a feed is being cleaned up after a delete
FeedOperationStatu-sEvent	Called when a Job for a feed has started, stopped, succeeded, failed, or been abandoned
TemplateChangeEvent	Called when a Template is created, updated, or deleted
ServiceLevelAgree-mentEvent	As of Kylo 0.8.4, Called only when an SLA is deleted.

70.1 Event Source

All events extends the *AbstractMetadataEvent* object. You can find relevant source code for the events here: https://github.com/Teradata/kylo/tree/master/metadata/metadata-api/src/main/java/com/thinkbiganalytics/metadata/api/event

70.2 Example Feed Change Listener

```
import com.thinkbiganalytics.metadata.api.event.MetadataChange;
import com.thinkbiganalytics.metadata.api.event.MetadataEventListener;
```

```
import com.thinkbiganalytics.metadata.api.event.MetadataEventService;
import com.thinkbiganalytics.metadata.api.event.feed.FeedChangeEvent;
import com.thinkbiganalytics.metadata.api.event.feed.
↪FeedOperationStatusEvent;
import com.thinkbiganalytics.metadata.api.feed.Feed;
import com.thinkbiganalytics.metadata.api.op.FeedOperation;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import java.util.Optional;

import javax.annotation.Nonnull;
import javax.annotation.PostConstruct;
import javax.inject.Inject;

@Component
public class ExampleFeedListener {

    private static final Logger log = LoggerFactory.
↪getLogger(ExampleFeedListener.class);

    @Inject
    private MetadataEventService metadataEventService;

    /**
     * Listen for when feeds change
     */
    private final MetadataEventListener<FeedChangeEvent>␣
↪feedPropertyChangeListener = new FeedChangeEventDispatcher();


    /**
     * Listen for when feed job executions change
     */
    private final MetadataEventListener<FeedOperationStatusEvent>␣
↪feedJobEventListener = new FeedJobEventListener();


    @PostConstruct
    public void addEventListener() {
        metadataEventService.addListener(feedPropertyChangeListener);
        metadataEventService.addListener(feedJobEventListener);
    }

    private class FeedChangeEventDispatcher implements MetadataEventListener
↪<FeedChangeEvent> {

        @Override
        public void notify(@Nonnull final FeedChangeEvent metadataEvent) {

            //feedName will be the 'categorySystemName.feedSystemName'
            Optional<String> feedName = metadataEvent.getData().
↪getFeedName();

            //the id for the feed
            Feed.ID feedId = metadataEvent.getData().getFeedId();
```

```java
            //feed state will be NEW, ENABLED, DISABLED, DELETED
            Feed.State feedState = metadataEvent.getData().getFeedState();

            if (feedName.isPresent()) {
                log.info("Feed {} ({}) has been {} ", feedName.get(), feedId,
 metadataEvent.getData().getChange());
                Feed.State state = metadataEvent.getData().getFeedState();
                if (metadataEvent.getData().getChange() == MetadataChange.
ChangeType.CREATE) {
                    //Do something on Create
                } else if (metadataEvent.getData().getChange() ==
MetadataChange.ChangeType.UPDATE) {
                    //Do something on Update
                } else if (metadataEvent.getData().getChange() ==
MetadataChange.ChangeType.DELETE) {
                    //Do something on Delete
                }



            }
        }
    }

    private class FeedJobEventListener implements MetadataEventListener
<FeedOperationStatusEvent> {

        @Override
        public void notify(FeedOperationStatusEvent event) {

            //feedName will be the 'categorySystemName.feedSystemName'
            String feedName = event.getData().getFeedName();

            //the id for the feed
            Feed.ID feedId = event.getData().getFeedId();

            //This is the Job Execution Id
            FeedOperation.ID jobId = event.getData().getOperationId();

            //this is {STARTED, SUCCESS, FAILURE, CANCELED, ABANDONED}
            FeedOperation.State jobState = event.getData().getState();

            //this is CHECK  or  FEED.   CHECK refers to a Data Confidence
Job
            FeedOperation.FeedType feedType = event.getData().getFeedType();

            //a string message of what the event is for
            String statusMessage = event.getData().getStatus();

            if(event.getData().getState() == FeedOperation.State.SUCCESS){
                // Do something if a Job successfully completes
            }
        }
    }


}
```

Operations Guide

71.1 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.

71.2 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.

71.3 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.

71.4 Abbreviations

Abbreviations/Key term	Definition
O&M	Operations and Maintenance
CLI	Command Line Interface
ES	ElasticSearch

71.5 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.

71.6 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 *steps*. 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 Dashboard 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 Dashboard 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 metadata, 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.

71.7 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.

71.7.1 Dashboard Page

The Dashboard 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.
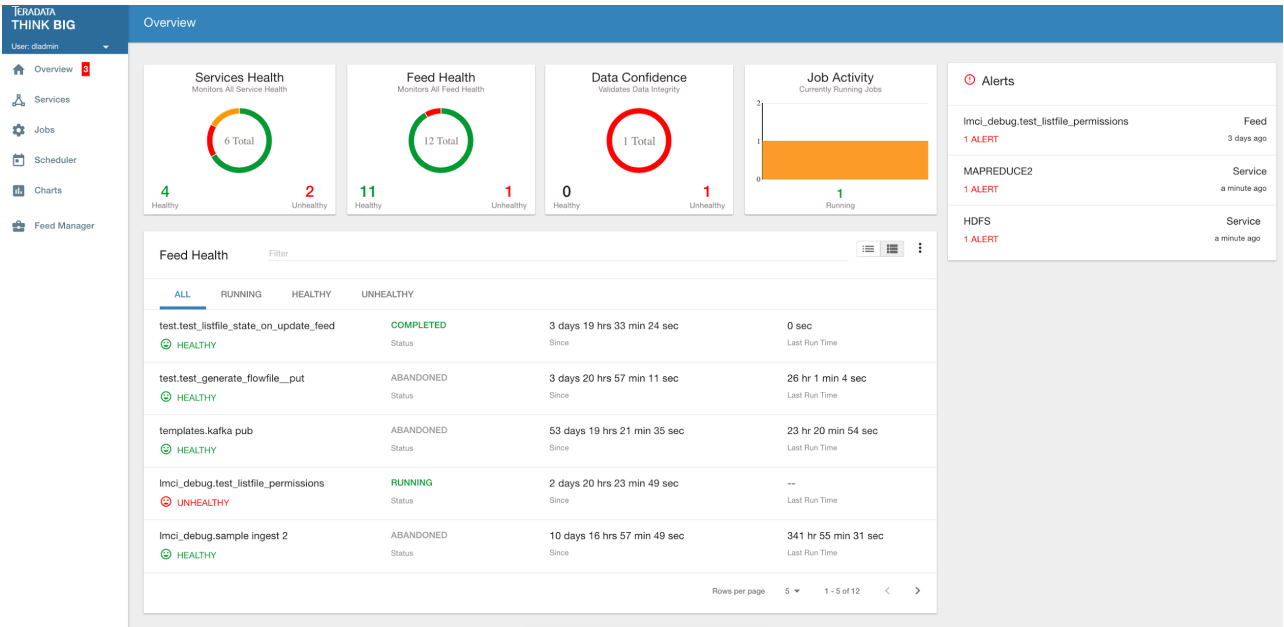
Kylo Dashboard Page

71.7.2 Key Performance Indicators

The Dashboard page has multiple indicators that help you quickly assess the health of the system:

Services Health Monitors All Service Health 6 Total **4** Healthy **2** Unhealthy	Provides a health status of external dependencies such as MySQL or Postgres, Hadoop services.
Feed Health Monitors All Feed Health 12 Total **11** Healthy **1** Unhealthy	Provides a summary health status of all data feeds. Details of these feeds are shown in a table, Feed Health, also on the Dashboard Page
Data Confidence Validates Data Integrity 1 Total **0** Healthy **1** Unhealthy	Optional. Displays a confidence metric updated by any Data Quality Check jobs.
Job Activity Currently Running Jobs **1** Running	Displays all running jobs.
Alerts lmci_debug.test_listfile_permissions Feed 1 ALERT 3 days ago MAPREDUCE2 Service 1 ALERT a minute ago HDFS Service 1 ALERT a minute ago	Displays alerts for services and feeds. Click on them for more information.

71.7.3 Feed Health

The Feed Health 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.

71.7.4 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.

71.7.5 Understanding Job Status

Jobs have two properties that indicate their status and state, Job Status and Exit Code respectively.

71.7.6 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.

71.7.7 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

71.7.8 Controlling Jobs

The image below illustrates the different *actions* that can be performed based on its Job Status:
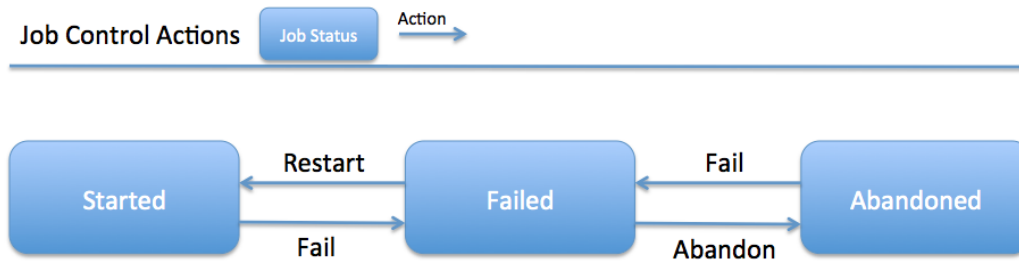
71.7.9 Feed History Page

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 Health table on the Dashboard page. Initially the Feeds table provides high-level data about the feed.
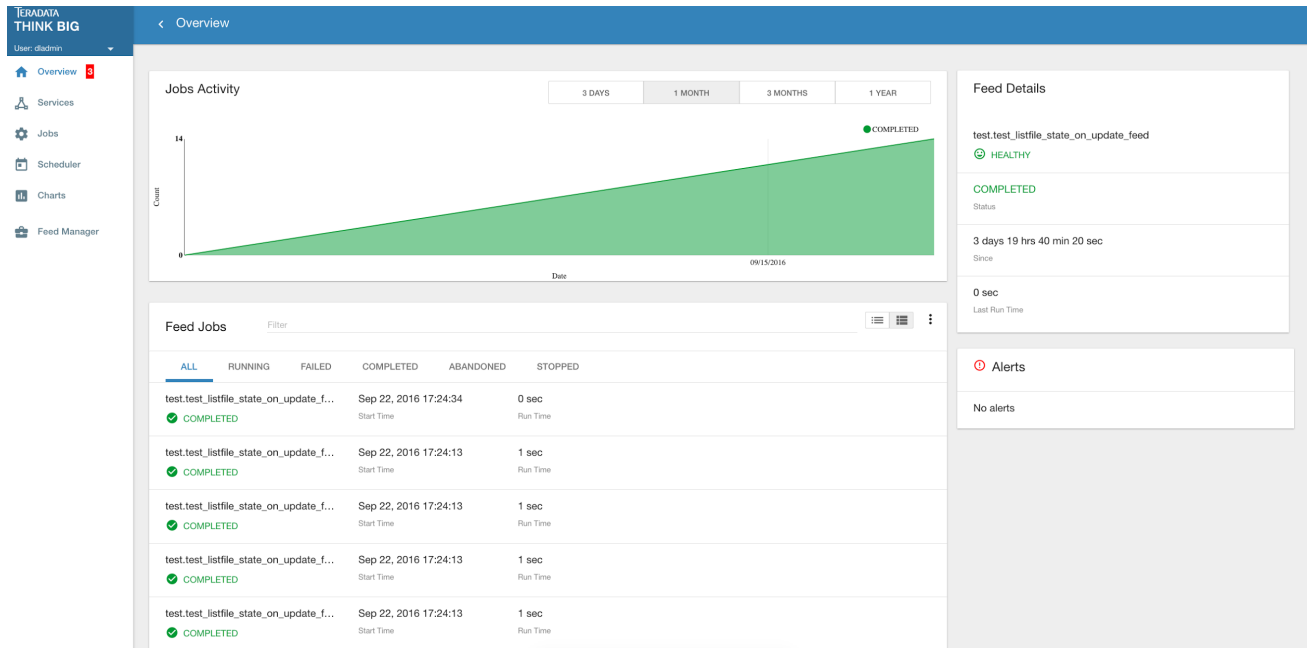
You can get more data by clicking on a job in the Feed Jobs table. This will go into the Job Details page for that job.

71.7.10 Job History Page

Job history can be accessed in the Jobs Tab.

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.

71.7.11 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

JOB   STEP 1   STEP 2   STEP 3

test.test_listfile_state_on_update_feed   Sep 22, 2016 17:24:34   0 sec   COMPLETED
✅ COMPLETED   Start Time   Run Time   Exit Code

Exit Description
No description available.

                                          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   ▾

71.7.12 Job Status Info

Job Status information such as start and run time, along with any control actions, are displayed on the right.

Job Details   ⦾

RESTART   ABANDON

lmci_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   ▾

71.7.13 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_1474381628485438570506824722505.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.

71.7.14 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.

71.7.15 Step Context Data

A job can have multiple steps, each of which capture and store metadata as it relates to that step.

71.7.16 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.

71.7.17 Changing an SLA

To change the schedule of a given SLA :

1. Click on the SLA tab in the Feed Manager site.
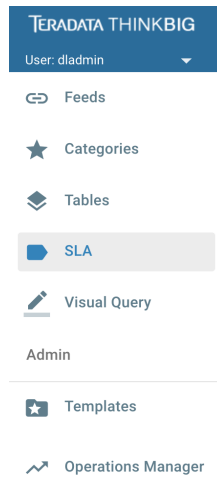
2. Select the SLA whose schedule you would like to change.
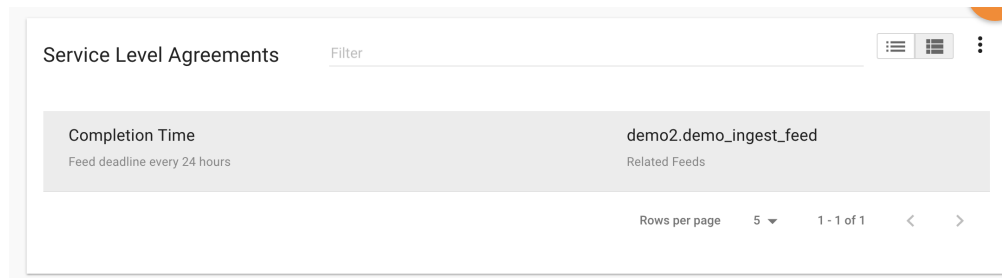
3. Edit the configurations and click Save SLA.

71.7.18 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.

71.7.19 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.

Clicking on the ☰ icon in the top right of the table will display the table so that you can sort by column.

Click on any of the column headers, or click on the ⋮ icon in the top right of the table, to sort.

71.7.20 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.

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

The data attributes at the top can be dragged into either Column Header or Row level attributes for the rendered pivot.

Clicking the down arrow on each attribute allows you to filter out certain fields.

This interface allows you to filter the job data and create many different combinations of tables and charts.

71.8 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)

71.9 Installation

Please refer to the installation guide for Kylo installation procedures.

71.10 Application Configuration

Configuration files for Kylo are located at:

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

71.10.1 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 * ? *
```

71.10.2 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.*

71.10.3 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.

71.11 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
```

71.12 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
```

71.13 Additional Configuration

The following section contains additional configuration that is possible.

71.13.1 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.

71.13.2 Service Status Configuration

The Dashboard 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 Dashboard 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 Component Alerts	Filter		
DataNode Process ✓ OK	TCP OK - 0.000s response on port 50010		10/03/2016 at 9:10
	Message		Time
DataNode Web UI ✓ OK	HTTP 200 response in 0.000s		10/03/2016 at 9:10
	Message		Time
DataNode Unmounted D... ✓ OK	Data dir(s) are fine, /hadoop/hdfs/data .		10/03/2016 at 9:09
	Message		Time
DataNode Storage ✓ OK	Remaining Capacity:[13935746895], Total Capacity:[73% Used, 52587134976]		10/03/2016 at 9:09
	Message		Time
DataNode Heap Usage ✓ OK	Used Heap:[10%, 96.34919 MB], Max Heap: 1004.0 MB		10/03/2016 at 9:09
	Message		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
```

71.14 Migrating Templates and Feeds

71.14.1 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

here.

To export a template, click the Export button for that template. This will download a zip archive of the template.

71.14.2 Importing Registered Templates

To import a registered template, on the Templates tab click on the button in the top right. Select Import from File.

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.

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.

71.14.3 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.

71.14.4 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.

Browse for the exported feed and then click **Import Feed**.

If the import is successful, you should now see a running feed in the Feeds tab.

71.14.5 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.

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.

71.14.6 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.

71.14.7 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.
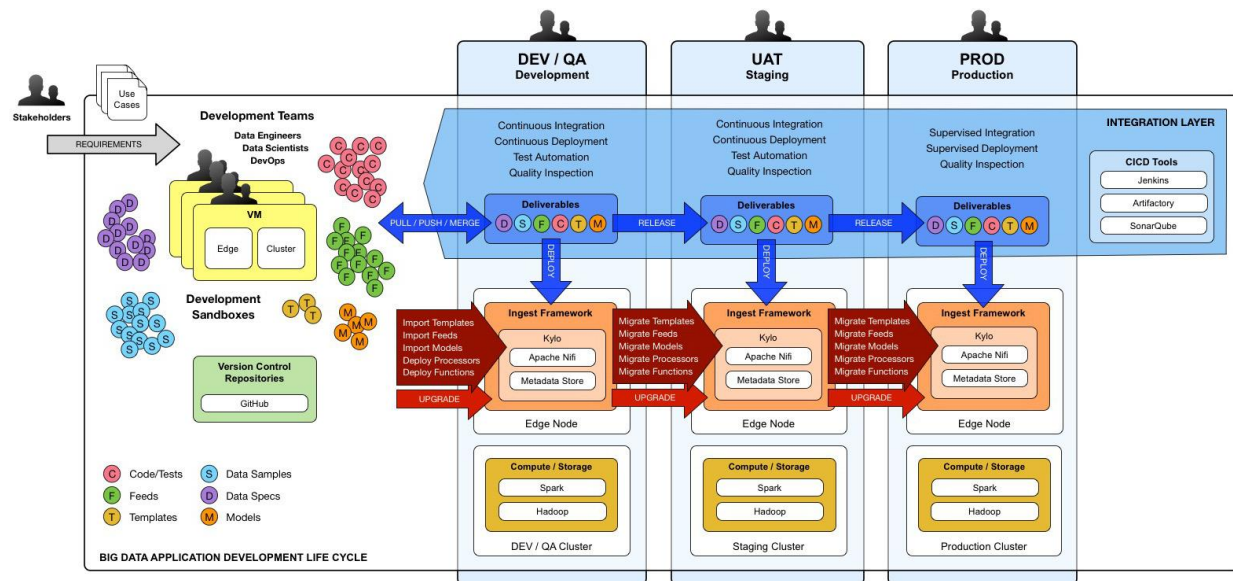
71.14.8 Migrating Kylo and NiFi Extensions

If custom NiFi or Kylo plugins/extensions have been built, they must 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 place 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.

71.15 Operational Considerations

When considering promoting Kylo/NiFi metatdata 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)

Troubleshooting & Tips

72.1 Tuning the ExecuteSparkJob Processor

72.1.1 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.

72.1.2 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.

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) num-executor = 3 * (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) total.memory (GB) = yarn.nodemanager.resource.memory-mb * (spark.executor.cores / 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

72.2 Dealing with non-standard file formats

72.2.1 Problem

You need to ingest a file with a non-standard format.

72.2.2 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"))
```

72.3 Merge Table fails when storing as Parquet using HDP

72.3.1 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!
```

72.3.2 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;
```

72.4 NiFi becomes non-responsive

72.4.1 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.

72.4.2 Solution

Increase memory to NiFi by editing /opt/nifi/current/conf/boostrap.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.

72.5 Automated Feed and Template Importing

72.5.1 Problem

Feeds and templates should be automatically imported into the staging or production environment as part of a continuous integration process.

72.5.2 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
```

72.6 Spark job failing on sandbox with large file

72.6.1 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.

72.6.2 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.

72.7 NiFi hangs executing Spark task step

72.7.1 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.

72.7.2 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.

72.8 Spark SQL fails on empty ORC and Parquet tables

72.8.1 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=1fb1b9a0-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.

72.8.2 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

72.9 High Performance NiFi Setup

72.9.1 Problem

The NiFi team published an article on how to extract the most performance from Apache NiFi.

72.9.2 Solution

See

72.10 RPM install fails with 'cpio: read' error

72.10.1 Problem

Kylo rpm install fails giving a 'cpio: read' error.

72.10.2 Solution

This problem occurs if the rpm file is corrupt or not downloaded properly. Try re-downloading the Kylo rpm from the Kylo website.

72.11 Accessing Hive tables from Spark

72.11.1 Problem

You receive a NoSuchTableException when trying to access a Hive table from Spark.

72.11.2 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/
```

72.12 Compression codec not found for PutHDFS folder

72.12.1 Problem

The PutHDFS processor throws an exception like:

```
java.lang.IllegalArgumentException: Compression codec com.hadoop.compression.lzo.
→LzoCodec not found.
```

72.12.2 Solution

Edit the /etc/hadoop/conf/core-site.xml file and remove the failing codec from the io.compression.codecs property.

72.13 Creating a cleanup flow

72.13.1 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.

72.13.2 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.

72.14 Accessing S3 from the data wrangler

72.14.1 Problem

You would like to access S3 or another Hadoop-compatible filesystem from the data wrangler.

72.14.2 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 .

72.15 Dealing with XML files

72.15.1 Problem

You need to ingest an XML file and parse into Hive columns.

72.15.2 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:

72.15.3 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")
```

72.16 Dealing with fixed width files

72.16.1 Problem

You need to load a fixed-width text file.

72.16.2 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}).\*")
```

72.17 Dealing with custom SerDe or CSV files with quotes and escape characters

72.17.1 Problem

You need to load a CSV file with surrounding quotes and don't want those quotes removed.

72.17.2 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!

72.18 Configuration on a Node with Small Root Filesystem

72.18.1 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.

72.18.2 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

72.19 GetTableData vs ImportSqoop Processor

72.19.1 Problem

You need to load data from a structured datastore.

72.19.2 Solution

There are two major NiFi processors provided by Kylo for importing data into Hadoop: GetTableData and Import-Sqoop.

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 Import-Sqoop processor instead, the following changes should to 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.

72.20 Using machine learning functions

72.20.1 Problem

You need to use a machine learning function in a data transformation feed.

72.20.2 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))
```

72.21 Sqoop requires JDK on Kylo sandbox

72.21.1 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."

72.21.2 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.

72.22 Validator is unable to process policy JSON file

72.22.1 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.

72.22.2 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.

72.23 Creating a feed fails due to java.sql.BatchUpdateException

72.23.1 Problem

When using MySQL/MariaDB as the database for Kylo, creating a feed with large number of columns can lead to an exception in the last step (Setting the feed schedule and saving it). Sample exception below:

```
java.sql.BatchUpdateException: (conn:330) Could not send query: stream size 1652321
↪is >= to max_allowed_packet (1048576)
```

72.23.2 Solution

Increase the *max_allowed_packet* configuration parameter for the database server.

1. Add this entry to file **/etc/my.cnf** under the **[mysqld]** section.

```
[mysqld]
max_allowed_packet=16M
```

2. Restart the database server. Choose command as per your database.

```
service mariadb restart
service mysql restart
```

3. Verify the change by executing this in the database client console.

```
show variables like 'max_allowed_packet';
```

4. Save the feed now.

72.24 When using Solr, indexing schema with large number of fields throws exception

72.24.1 Problem

When using Solr as the search engine, indexing of the feed schema can throw an exception similar to below. This happens when the feed contains a large number of columns.

```
Exception writing document id a1e41cbc-d550-49cc-bc20-49fc981e767e to the index;␣
→possible analysis error: Document contains at least one immense term in field=
→"hiveColumns" (whose UTF8 encoding is longer than the max length 32766), all of␣
→which were skipped. Please correct the analyzer to not produce such terms.
```

72.24.2 Solution

1. Execute this command. Replace **localhost** if necessary.

```
curl -X POST -H 'Content-type:application/json' --data-binary '{
"replace-field":{
 "name":"hiveColumns",
 "type":"text_general" } }' "http://localhost:8983/solr/kylo-datasources/schema?
→wt=json&indent=true"
```

2. Restart Solr server.

3. Create the feed again.

72.25 Prioritize jobs based on an attribute value

72.25.1 Problem

You have many feeds in Kylo that all execute at once but there are a few high-priority feeds that should be executed before the others.

72.25.2 Solution

The connection to a processor can be configured to prioritize flow files using a few different methods:

- **FirstInFirstOutPrioritizer**: Given two FlowFiles, the one that reached the connection first will be processed first.

- **NewestFlowFileFirstPrioritizer**: Given two FlowFiles, the one that is newest in the dataflow will be processed first.

- **OldestFlowFileFirstPrioritizer**: Given two FlowFiles, the one that is oldest in the dataflow will be processed first. This is the default scheme that is used if no prioritizers are selected.

- **PriorityAttributePrioritizer**: Given two FlowFiles that both have a "priority" attribute, the one that has the highest priority value will be processed first. Note that an UpdateAttribute processor should be used to add the "priority" attribute to the FlowFiles before they reach a connection that has this prioritizer set. Values for the "priority" attribute may be alphanumeric, where "a" is a higher priority than "z", and "1" is a higher priority than "9", for example.

See the for more information.

72.26 EsIndexException in Kylo services logs

72.26.1 Problem

Kylo services log contains errors similar to this: *org.modeshape.jcr.index.elasticsearch.EsIndexException: java.io.IOException: Not Found*

72.26.2 Solution

Pre-create the indexes used by Kylo in Elasticsearch. Execute this script:

```
/opt/kylo/bin/create-kylo-indexes-es.sh
```

The script takes 4 parameters.

```
<host> <rest-port> <num-shards> <num-replicas>
Examples values:
 host: localhost
 rest-port: 9200
 num-shards: 1
 num-replicas: 1
 Note: num-shards and num-replicas can be set to 1 for development environment
```

Best Practices

The following document describes patterns and best practices particularly oriented to IT Designers and System Administrators.

73.1 Organizational Roles

Kylo supports the division of responsibility between IT designers, administrators, operations, and end-users.

73.1.1 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.

73.1.2 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 would 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.

73.1.3 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.

73.1.4 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.

73.1.5 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.

73.2 Designers

Guidance for designers who design new pipeline templates and enable self-service.

73.2.1 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 template 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.

73.2.2 Template re-use

Templates should be designed for maximum re-use and flexibility. Kylo's standard ingest serves as an example of this. There are two types of templates Kylo uses this to promote this objective:

- Feed Template. Kylo generates a clone of this template as a unique running instance per feed. This means for every feed, there is a copy of the pipeline as defined by the template. Kylo uses the template to make the clone and injects any metadata configured in the feed (e.g. data source selections, schema configuration, etc). The feed template should be composed of the set of initial datasource connectors, an UpdateAttribute processor where Kylo can inject common metadata configured by the wizard, and an output port connected to a re-usable flow (below). The feed-based template should include minimal logic. The bulk of logic should be contained in the re-usable flow.

- Reusable-flow Template. This template is used to create a single running instance of the flow that can supports multiple connected feeds through a NiFi input port. The core logic for your pipelines should be centralized into re-usable flows. This allows one to update the pipeline for many feeds in just one place.

Again, both types of templates are exemplified in Kylo's standard ingest template included with Kylo. More about reusable flows is discussed below.

73.2.3 Reusable Flows

When possible, consider using re-usable flows for the majority of pipeline workflow and logic. A reusable flow is a special template that creates just a single instance of a flow shared by other feed flows through a NiFi process port. 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.

Please see Kylo's standard ingest templates for an example of this in action.

73.2.4 Streaming Templates

Kylo can support batch and streaming feeds. In a batch feed, each dataset is processed and tracked as a job from start to finish. The entire job fails if the dataset is not processed successfully.

Streaming feeds typically involve continuous data processing of very frequent, discrete packets of data. Data can be flowing through different portions of the pipeline. Tracking each record in a streaming feed as a job would add significant overhead and could be meaningless. Imagine consuming millions of JMS messages and viewing each records journey through the pipeline as a job. This would be impractical. Instead, Kylo treats a streaming feed as a constant running job, gathering aggregate statistics such as success and failure rates, throughput, etc.

A template can be registered as a streaming template by checking the 'Streaming template" checkbox on the last step of the template registration wizard.

73.2.5 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.

73.2.6 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).

73.2.7 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.

73.2.8 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.

73.2.9 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. The cleanup-up flow is described below.

73.2.10 Clean-up

When creating a template, ensure you have the appropriate clean-up activity associated. If using the standard ingest, you can also use the standard clean-up to remove HDFS, Hive tables and the feed itself. This is triggered when the delete feed option is clicked on the Kylo UI.

Clean up flows should be configured to start with a TriggerCleanup trigger processor and the attribute variables set to specify that feed. When you register the template in Kylo, be sure to set the attributes for the Trigger Cleanup processor to take the metadata systemNames of the feed.

For each client, think about what a clean-up best practice will be when you design the template as this may be different per client.

Clean-ups could also be triggered through a JMS message using the publish and consumeJMS processors. In t this way you could start a clean-up activity on the completion of a feed for instance

73.2.11 Lineage Tracking

Kylo automatically maintains lineage at the "feed-level" and by any sources and sinks identified by the template designer when registering the template.

Kylo relies on the designer specifying the roles of processors as sources or sinks when registering the flow. The default or stereotype role of processors can be defined by a system administrator conf/datasource-definitions.json.

73.2.12 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.

73.2.13 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.

Application properties override template attribute settings and can be very useful for setting environment specific settings and also to set specific controller related settings. Application properties can be set encrypted and should be when setting sensitive information.

Note: You should NOT add your processor attributes to application properties unless they are ENVIRONMENT specific. It is an anti-pattern to try to bring all attributes out into "configuration property files".

73.2.14 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.

73.2.15 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.

73.2.16 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.

73.2.17 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.

73.2.18 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.

73.2.19 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).

73.2.20 RDBMS Data

Joins in Hadoop are inefficient. Consider de-normalizing data during ingest. One strategy is to ingest data via views.

73.2.21 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.

73.2.22 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.

73.3 Development Patterns

Best practices and guidance oriented to the development process, release, and testing.

73.3.1 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.

73.3.2 Automated Deployment

Building an automated deployment scripts is the best practice approach to deploying feeds and templates and this should be delivered along with your other deployment scripts. Importing of templates and feeds can be carried out via the REST API of Kylo.

73.3.3 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.

73.3.4 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.

73.3.5 Version Control

It is recommended to manage exported templates and feeds through an SCM tool such as git, subversion, or CVS.

73.3.6 General Deployment Guidelines

Regardless of whether deploying manually or using automated scripts, ensure the following:

- Deploy any reusable templates first
- Configure controller services (in NiFi) on the first time a template is imported or if any new controllers are introduced
- Smoke test your pipeline

73.4 Users

Best practices and guidance oriented to end-users (users of the Kylo application).

73.4.1 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.

73.4.2 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.

73.4.3 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.

73.4.4 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.

73.5 Administrators

73.5.1 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.

73.5.2 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.

73.6 Security

Guidance around security.

73.6.1 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 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.