Koverse Manual

version 1.4

Contents

Introduction

Overview

Koverse is a scalable and secure platform designed for quickly and efficiently discovering the value of your data and incorporating derived insights into daily operations. Koverse is not designed around the problem set of a traditional database. Instead, Koverse enables new paradigms of ingest, processing, and query speeds at massive scale. Koverse provides the capability for organizations to become data-driven; using data to make better decisions and find new opportunities.

The Koverse Advantage

Koverse, Inc. separates signal from noise for data-driven organizations, turning information overload into an information advantage. We deliver meaningful insights critical for organizations to increase effectiveness in every aspect of business. The Koverse solution consolidates the elements necessary for deriving actionable insight from data:

Collect: Universal Ingest allows fast ingest of any data from any source.

Analyze: Adaptive Workflows unify disparate data sets and use embedded analytics for immediate insight across the entire organization.

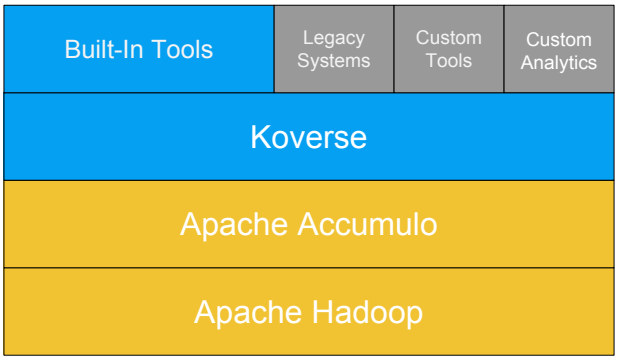
Act: Interactive data exploration and discovery makes it easier for business and technical users to quickly find information and extract meaning.

Koverse frees organizations from the time, expense, and risk of cobbling together multiple products and open source technologies to solve their data problems by providing all essential functionality right out of the box. Koverse provides all of this in a scalable, secure multi-tenant environment built on well-established technologies such as Hadoop.

Koverse Logical Architecture

The logical architecture of the Koverse platform was conceived to lower the barriers to entry that exist for many organizations who want to become data-driven. Big data solutions such as Hadoop and Accumulo are essential to big data success, but do not offer enough on their own to quickly and easily turn data into insights. Koverse fills this gap by building on well-established technologies and providing the additional capabilities that are necessary to deliver meaningful insights.

Below is a diagram of the Koverse platform's logical architecture. The foundation is formed by Apache Hadoop and Accumulo. Koverse builds on this foundation, offering built-in applications and analytics, in addition to tools for customers to create custom applications and analytics that support both legacy systems and revolutionary new use cases.



Koverse Logical Architecture

The *Extensibility* section offers more details about extending built-in Koverse capabilities to meet operational needs.

Physical Architecture

Koverse's software components are designed to run on top of Apache Hadoop - which is a scalable stack of software for data processing. Both Hadoop and Koverse's components are designed to allow for small to very large scale-up. Therefore, when describing the physical architecture one must consider that many of these software components are duplicated across many physical machines.

Koverse is comprised of two software components named Koverse Webapp and Koverse Server. The Koverse Webapp is a traditional Java WAR file that runs in a JBoss server. This webapp provides assets like HTML, Javascript, images, and CSS for web browser based clients. It also provides the REST API end point for third party integrations. The Koverse Webapp communicates with the Koverse Server to service requests for data. The Koverse Server provides the business logic for executing queries, applying data security, and monitoring systems. Koverse Server is a stand-alone Java process. The Koverse Server support 3rd party clients using an Apache Thrift API.

Koverse runs on Apache Hadoop, and therefore requires the base Hadoop components - which are Jobtracker and Namenode. The Hadoop Jobtracker maintains state around jobs that are executing across the Hadoop cluster, and communicates with many Tasktrackers. The Namenode maintains a file system index for the Hadoop Data File System (HDFS), and communicates with many Datanodes. See http://hadoop.apache.org/ for more information about Hadoop.

Koverse uses Apache Accumulo, which is a key-value NoSQL data store on Hadoop. Accumulo generally has single a "Master" process and many "Tablet Servers". See http://accumulo.apache.org/ for more information about Accumulo.

Koverse uses Apache Kafka for streaming data processing. Kafka is a distributed queue that provides guaranteed processing of data across many nodes. See http://kafka.apache.org/ for more information.

Network Requirements

The nodes in a Koverse installation should be connected via dedicated gigabit ethernet with a full bandwidth switching plane. The nodes should be physically and logically network isolated - as they perform network intensive operations.

DNS Requirements

Koverse, Hadoop, Accumulo, and Zookeeper have strict DNS requirements that must be met before an installation can begin. The following is a break down of a DNS entry for one physical machine in the Koverse cluster. Note the four part DNS name - where the first (lowest) part of the DNS name is the physical server "hostname", and the second part of the DNS name is the name of the "cluster".

hostname.cluster.domain.tld

Examples:

- koverse1.production.mycompany.com
- control.production.mycompany.com

In addition to external resolution of these DNS entries, the nodes in the cluster must be able to resolve the "hostname". For example, The koverse1 machine must resolve the dns entry for "control" to "control.cluster.mycompany.com". This is done through "search domains" - see the OS documentation for more details. The example search domain above is "production.mycompany.com".

The sections below define the hostnames that must resolve for each node.

Minimal Koverse Physical Architecture

This section describes the minimum necessary physical architecture for a distributed Koverse installation. The sections below name and describe the five physical servers in this configuration.

- 'Control' Node
    - Hadoop Namenode & Jobtracker Services
    - Accumulo Master, Monitor, Garbage Collector Services
    - DNS Hostnames: master, jobtracker, namenode
    - 16GB of RAM, 20GB of Disk Storage, 4 CPUs
- 'Koverse' Node
    - Koverse Server & Koverse Webapp
    - DNS Hostnames: koverse1, www
    - 16GB of RAM, 8GB of Disk Storage, 4 CPUs
- 'Worker' Nodes
    - Minimum of three of these nodes
    - Hadoop Datanode and Tasktracker Services
    - Zookeeper Server
        - Only 3 of these nodes should run zookeeper
    - Accumulo Tablet Server and Tablet Logger Services
    - Kafka Service
    - DNS Hostnames: worker1, worker2, worker3, zoo1, zoo2, zoo3
    - 16GB of RAM, One 8 GB root drive, Four Large (500GB - 2TB) Raw Disks, 4 CPUs

Production Koverse Physical Architecture

This section describes the typical physical architecture for a production Koverse installation. The sections below name and describe the physical servers necessary.

- 'Namenode' Node
    - Hadoop Namenode Services
    - DNS Hostnames: namenode
    - 64GB of RAM, 100GB of Disk Storage, 8 CPUs
- 'Jobtracker' Nodes
    - Hadoop Jobtracker Service
    - DNS Hostnames: jobtracker
    - 64GB of RAM, 100GB of Disk Storage, 8 CPUs
- 'Accumulo Master',

    - Accumulo Master, Monitor, Garbage Collector Services
    - DNS Hostnames: master,
    - 32GB of RAM, 100GB of Disk Storage, 8 CPUs
- 'Koverse' Node
    - Koverse Server & Koverse Webapp
    - DNS Hostnames: koverse1, www
    - 32GB of RAM, 8GB of Disk Storage, 8 CPUs
- 'Zookeeper' Nodes
    - Three of these running zookeeper servers
    - DNS Hostnames: zoo1, zoo2, zoo3
    - 8GB of RAM, One 8 GB root drive, 4 CPUs
- 'Worker' Nodes
    - Minimum of three of these nodes, but more realistically about 10
    - Hadoop Datanode and Tasktracker Services
    - Accumulo Tablet Server and Tablet Logger Services
    - Kafka Service
    - DNS Hostnames: worker1, worker2, worker3....
    - 64GB of RAM, One 8 GB root drive, Four Large (500GB - 2TB) Raw Disks, 4 CPUs

Security Model

Security is a fundamental component of the Koverse architecture because it allows Koverse to operate as a multi-tenant system. Multiple users can use the system, but with controlled access to all system resources. Koverse uses Authentication, Authorization, and Auditing at multiple levels to ensure that the correct users are given access to the correct resources.

Authentication is defined in this context as verifying the identity of a user, and **authorization** is granting an authenticated user access to specific resources. All actions within Koverse are permitted only by authorized users.

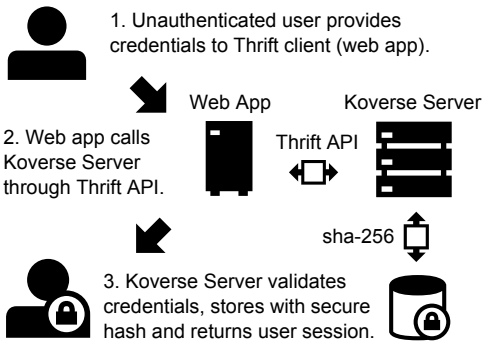
Auditing is the record-keeping of the actions that distinct users perform over time. These records can be reviewed to provide further assurance that only authorized users are accessing, or attempting to access, the appropriate resources.

Authentication

Authentication is the first step required in order to access Koverse resources. Access to Koverse resources can be requested either through built-in Koverse web apps, or through direct calls to the Koverse Thrift API. In both cases, the user is requesting access via a client of the Koverse server, so authentication will occur in the same manner.

Default Koverse Authentication

The *Koverse Default Authentication* figure below illustrates the steps which occur to authenticate a user who is requesting access to Koverse resources using the default method of authentication, which consists of a username and password.

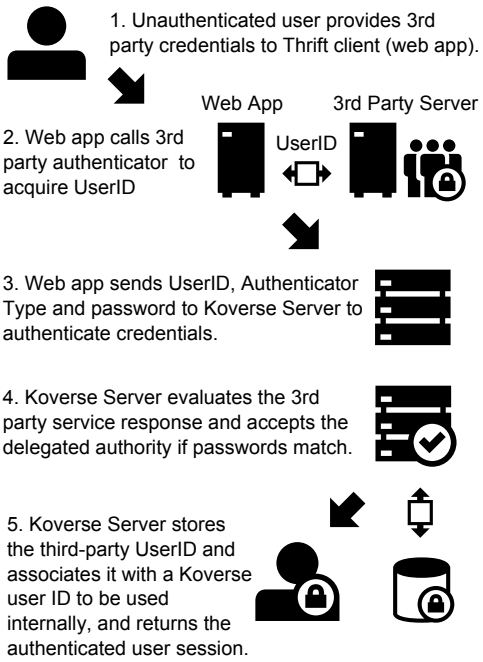


Koverse Default Authentication

1. User submits credentials (username and password) to web app.
2. Web app passes username and password to Koverse server.
3. If the password matches what is stored on the Koverse Server for that username, the user is authenticated. Note that user passwords are stored securely using a salted SHA-256 hash.

Third-Party Authentication

The *Third-Party Authentication* diagram shows the steps which occur to authenticate a user who is requesting access to Koverse resources using third-party authentication.



Third-Party Authentication

1. User submits third-party credentials to web app.

2. Web app authenticates to third-party system (identified within Koverse by it's 'Authenticator Type') and, if successful, retrieves user's third-party userID.
3. Web app passes in Authenticator Type, authenticator password, and third-party UserID to Koverse server.
4. If the third-party Authenticator Type and authenticator password match one that is registered with Koverse as a trusted service, the third-party authentication is automatically accepted by Koverse and no further authentication is required.
5. Koverse server stores the third-party UserID and associates it with a Koverse user ID to be used internally to Koverse.

See the *Extensibility* and *Koverse Developer Documentation* for more information on how to implement Third-Party Authentication and Authorization.

Authorization

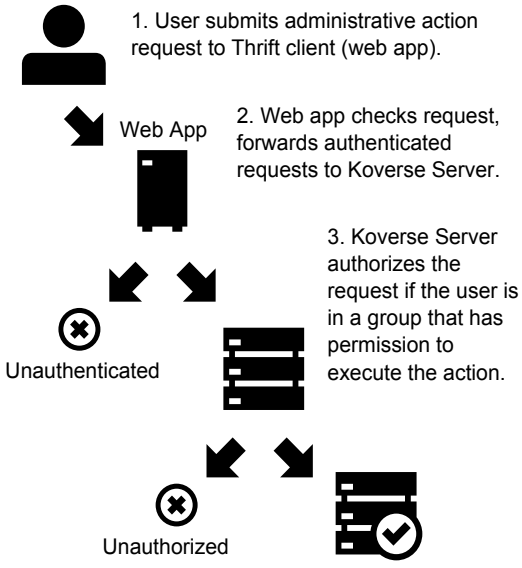
Koverse maintains the rules for which users have permission to access to which resources. Once a user has been authenticated, there are two main types of resources a user can request access to in Koverse: 1) The ability to perform administrative actions and 2) the ability to access data in the datastore. The process of authorization will determine whether access is granted.

Authorization to Perform Administrative Actions

Within Koverse, each user is assigned to one or more groups and each group is granted permissions to perform specific administrative actions, such as the ability to manage data collections, manage user accounts, view audit logs, etc. More details on configuring user groups can be found in the *Koverse User Guide*.

Groups may correspond to external user groups assigned by a third-party authorization system, which allows for easy integration when customers already have their users organized into various groups. In this case, the third-party authorization service is registered with Koverse as a trusted service and all group memberships the service associates with a particular user will be honored by the Koverse Server.

When a user attempts to perform an administrative action, either via built-in Koverse Apps, or via an API call, the Koverse Server will check to see which groups the user is a member of. The user's attempt will be successful only if the user is a member of a group that has permission to perform the action.



Administrative Action Authorization

Authorization to Access Data

Koverse provides fine-grained access control that allows multiple datasets to coexist in the same datastore without compromising sensitive information. The fundamental constructs that make up data stored in Koverse are Collections and Records. Details on these constructs can be found in the *Data Model* section, but for the purposes of this section, a Record can be thought of as an individual piece of data and a Collection as being made up of all of the Records that belong together as part of some logical dataset.

Each Record is assigned a security label that determines which permissions a user must possess in order to access that Record. This allows Koverse to control access to Collections in their entirety, in addition to providing even more precise access control on individual Records.

When a Collection is first created, user groups are mapped to have (or not have) permission to read the Collection. An important point to note is that permission to read a Collection does not guarantee access to read every Record within that collection. On top of the Collection-level read access granted to user groups, Koverse also honors additional, more restrictive, access controls that may apply to individual Records. Access to these restricted Records may be granted if a user has additional security tokens provided by a third-party authorization service. This is best illustrated through example:

Data Access Example

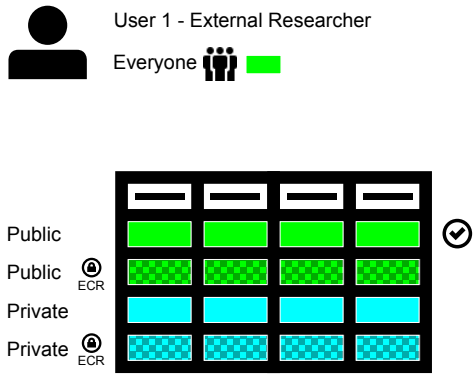
Suppose an academic institution wants to import all of its research findings into Koverse. Most of the findings are considered Public and would be shareable with any user. Some of the findings may be Private and perhaps should only be accessed by the institution's faculty. Another portion of the findings may relate to sensitive topics, such as chemical and biological agents, in which case they will be considered 'Export Controlled Research' and should only be exposed to researchers who are authorized U.S. citizens. Note that access to this last category of findings need not be restricted to researchers from the originating institution; students or faculty from other institutions who are authorized to access Export Controlled Research should be able to access them as well.

By default, Koverse automatically places all users in the 'Everyone' group. Suppose that the academic institution also has an authorization service that places its users into a 'Student' or 'Faculty' group if applicable. Furthermore, suppose that there is some Federal authorization service that will provide a registered user with an 'ECR' security token if they are authorized to access Export Controlled Research. Finally, suppose that both the institutional and the Federal authentication services are registered as trusted services with Koverse.

Consider the following scenarios:

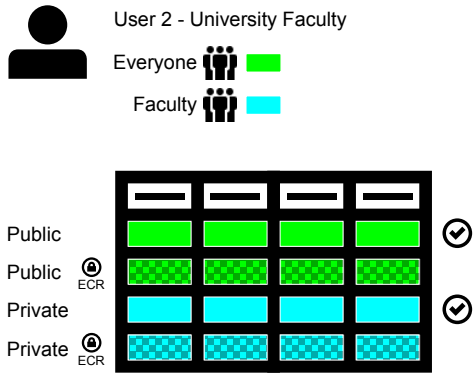
Scenario 1: User1 is from an another institution and does *not* have the authority to view Export Controlled Research.

User1 can access non-ECR rows in Public Findings.



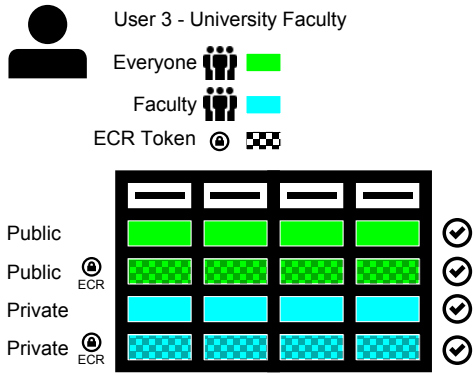
Scenario 2: User2 is a faculty member who does *not* have the authority to view Export Controlled Research.

User2 can access non-ECR rows in Public and Private Findings.



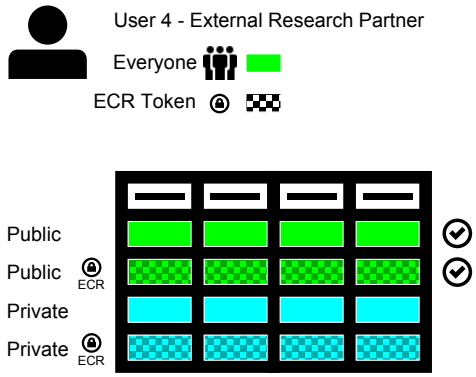
Scenario 3: User3 is a faculty member who has the authority to view Export Controlled Research.

User3 can access all rows in Public and Private Findings.



Scenario 4: User4 is from an outside institution and has the authority to view Export Controlled Research.

User4 Can access all rows in Public Findings.



Koverse's fine-grained access controls create an environment that is not only secure, but also ripe for information sharing and cross-corpus analytics. In the example above, users in scenarios 1 and 4 are allowed access to data that would otherwise be completely

unavailable to them. Equally as exciting is the fact that users in Scenarios 2 and 3 are able to perform queries and analytics across the Public and Private Collections, which would be impossible if the two datasets were physically segregated.

Auditing

Koverse keeps an audit log of user actions. Examples of actions that are audited are login attempts, queries, and changes to data flow configuration. The audit log provides an additional layer of assurance that users are not violating, or attempting to violate, the security rules of the system. Only administrators have access to the audit logs.

Built-In Capabilities

Koverse minimizes the amount of custom code that must be written to build big data applications by providing out-of-the-box capabilities for importing, querying, indexing, transforming, displaying, and exporting data, in addition to managing user access. The details of what is included in these powerful built-in capabilities are described below.

Data Collections

Data collections in Koverse are comprised of a set of Records. Records can be flat or hierarchical (e.g. a field can contain a list of values). Different records do not have to conform to the same schema. The following operations are automatically available for all data collections:

- Create, delete, and clear collections.
- Manage user access to individual collections.
- Tag collections.
- Lookup by name or tag.
- View count of total number of records in the collection.
- Browse representative samples of each collection.
- Browse details about the fields that exist in the data collection:
    - Field Name
    - Presence - how many times the field appears in the data collection
    - Average size in bytes
    - Estimate of cardinality - how many unique values exist for this field
    - Value types (e.g., string, integer, etc.) and relative frequency of each type
- Configure indexing to refine searchability of the collection.
- View data imports and transforms that affect the collection.

Imports

Koverse supports import of data from external data sources. User access to manage individual sources is configurable.

Built-In Support for Data Sources

Koverse handles the following import sources without requiring any custom code:

- Amazon S3

- Email Account (IMAP)
- File Transfer Protocol (FTP)
- Hadoop File System (HDFS)
- Kafka Queue 0.8
- MS SQL Server
- MySQL
- Newsfeed Source (RSS)
- Random Data (for testing)
- Twitter Streaming
- Twitter Timeline
- URL

Built-In Support for Data Formats

Koverse automatically parses the following file formats from file-based sources:

- XML
- JSON (either as an array of objects or a single object; each top-level object becomes a record)
- JSONSTREAM (JSON objects separated by newline characters, one record per line)
- CSV
- Text (.txt)
- RTF
- HTML
- Office (Word,PPT,Excel - OLE2 and OOXML versions)
- mbox email files
- PDF
- ePub

Built-In Import-Time Transforms

Koverse allows some processing to be done at import time. Import-time transforms apply a particular function to each record as it is ingested. Import-time transforms may be chained together.

- **Avro Deserialization** Allows users to deserialize a byte array in an incoming Record using an Avro schema as described by a JSON document provided by the user, and stores a Record structured according to the resulting Avro object.
- **Projection Transform** Allows a user to select a subset of fields to keep from incoming Records. Other fields not specified will be excluded.
- **Separated String Values** Allows a user to split a field containing a single string of separated values with a configurable value delimiter.
- **Uploaded File Processor** An import transform used to process files uploaded via the File Upload application.

Data Discovery

Users can submit queries and get back a set of Records. Koverse provides the following to support data discovery:

- Users can query one collection, a set of collections at once, or all collections they are authorized to read.
- Queries can be field-specific or can look for values appearing in any field.
- Indexing is controlled via the UI - no code needs to be written.
- Users can search for a range of values.
- Users can search multiple ranges simultaneously (multi-dimensional search) via composite indexes.
- Users can start typing and see suggested query terms (auto-complete).

Koverse automatically recognizes the following value types and will make them discoverable:

- Strings / free form text
- Numbers (integers and reals)
- URLs
- IP addresses
- Geos (lat/lon points)
- Dates
- Byte arrays with mime type

Transforms

Transforms allow users to glean valuable insights from one or more collections and store them in a new collection. They can be run once or set to run automatically. Transforms can be configured to process only new data or reprocess all the data in its input collections. There is also the option to transform data at the time of import. The output of a transform can either append to the output collection or replace the output collection every time the transform is run.

Because Koverse transforms are run using the Hadoop MapReduce framework, they benefit from all of the MapReduce features. For example, in addition to being massively parallel, transforms are fault-tolerant, and can be run using a configurable number of dedicated resources.

Koverse has a number of built-in transforms that can be run on collections without requiring any custom code:

Built-in Transforms

- **Close Graph** - The closed graph transform is a basic result which characterizes continuous functions in terms of their graphs.
- **Corpus Entity Stats** - Transform on an object-level which automatically ex-tract and integrate the semantic information about entities and return a list of ranked entities.
- **Corpus Word Stats** - Transform for finding the word frequency in giving recommendation for a spelling.
- **Document Similarity** - Transform over a set of documents or terms, where the idea of distance between them is based on the likeness of their meaning or semantic content as opposed to similarity which can be estimated.
- **Document Entities** - Transform to extract document entities from different datasets.
- **Entity Graph** - Transform to generate a graph of word or document entities.

- **Feature Extraction** - This transform extracts entities (locations, organizations, people, etc) found in structured fields and in text and counts up every time a pair of entities appears in a collection. This allows for looking up an entity and seeing all related entities and the relative strength of the relationship.
- **Geo Discovery** - Transform used to create a heat-map from geographical data.
- **Geo Location - Airport Codes** Allows users to augment a collection with a field containing airport codes with the latitude and longitude of the airport.
- **Geo Location - Canadian Postal Codes** Allows users to augment a collection with a field containing Canadian postal codes with the latitude and longitude of the center of the postal codes.
- **Geo Location - IPv4** Allows users to augment a collection with a field containing IPv4 address with the approximate latitude and longitude of IP address.
- **Geo Point Extraction** - Transform to extracts geo points from records.
- **Nearest Neighbors** - This analytic first extracts features as is done in the Feature Extraction application, and then proceeds to compare each entity to each other entity and calculates a score of similarity between two entities, based on features they have in common.
- **Pearson Correlation** - Transform to determine the correlation between sets of data and a measure of how well they are related.
- **Pig Transform** - Run a custom Apache Pig script to transform data. This can include User Defined Functions (UDFs) which can be packaged into jars and uploaded via the UI.
- **Python Transform** - Run a custom Python script to transform data.
- **Record Copy** - Copies incrementally all the records in a collection to a new collection.
- **Record Copy (Non-incremental)** - Copies all the records in a collection to a new collection.
- **Sentiment Analysis** - Calculates sentiment per a given field based on some text appearing with that field in a record. E.g. characterize the sentiment of locations, based on text associated with each location.
- **Sequence N-Grams** - Transform to find probability of an n-gram in a contiguous sequence of n items from a given sequence of text or speech.
- **Sequence Similarity** - Transform to sequence similarity of an empirical relationship between sequences.
- **Simple Regression Scoring Transform** - A transform to to fit a statistical regression model on one set of data and then evaluate the model on another set of data.
- **Spark Copy Transform** - Run a custom Spark copy transform to copy from one collection to another.
- **Spark SQL Transform** - Run a custom Spark SQL query script to transform data.
- **Summarize / Enumerate Fields** Calculates basic field statistics, such as the number of times each distinct value appears in the data, average size, max size, min size, etc.
- **Text Cleanup** Performs very simply text manipulations on field values, such as trimming, making upper case or lower case, etc.
- **TF-IDF vectors for documents** - Is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining.
- **Time Series** - Transform to extract time-series from different datasets.

Aggregations

Aggregation provides a means of turning billions of pieces of raw data into condensed, human-consumable information. Aggregations are arguably the first and most common analytic that people want to run on their data. To address this common use case, Koverse

provides a framework for building and querying aggregations without writing custom code. Besides the obvious aggregation of "Count", Koverse supports several other powerful functions like cardinality estimation, approximate top-k values, and approximate quantiles. Aggregations are maintained in near real-time on the Records of a Data Collection and are pre-computed so queries are sub-second regardless of how much data you have. These features provide a foundation for rapidly building products like interactive analytic dashboards that serve up-to-the-minute information. More detail on how to use Aggregations is seen in the *Koverse Developer Documentation*.

Apps

Koverse Web Applications, or Apps, provide a user interface to underlying Koverse capabilities. Koverse provides several built-in Apps so that users can start interacting with the system and gaining insight about their data immediately and without writing a single line of code.

Built-in Apps

Here is a list of the existing built-in Apps, and the capabilities they provide:

- **Audit Log** - Display and search details of all user activity, sorted in the order of the most recent events.
- **Configuration Manager** - Upload and download configuration for data Collections, Sinks, Sources, and Transforms.
- **Data Collections** - Manage and explore data Collections.
- **Data Flow** - Visualize, configure, and execute the movement of data within the Koverse system.
- **Search** - Query one or more Koverse Collections to find all Records that match search criteria.
- **File Upload** - Upload one or more files from the browser and import it into a collection.
- **System Administration** - Perform system administration activities, such as managing system configuration, user accounts, user groups, etc.
- **System Monitoring** - View health and status of the distributed cluster on which Koverse is running.

Exports

Koverse collections can easily be exported to external systems.

Records can be exported as JSON, as CSV when Records have a flat structure, or to relational database tables.

Built-in Export Sinks

- MySQL Database
- FTP
- HDFS

File-based export sinks will allow users to select a file format to use, the number of Records to include in each file, and a prefix to use when naming export files. Supported built-in file formats include:

Built-in Export File Formats

- CSV
- JSON

- XML

Export-Time Transforms

Users can apply one or more transforms to be applied to Records as they are exported. The following export-time transforms are built-in.

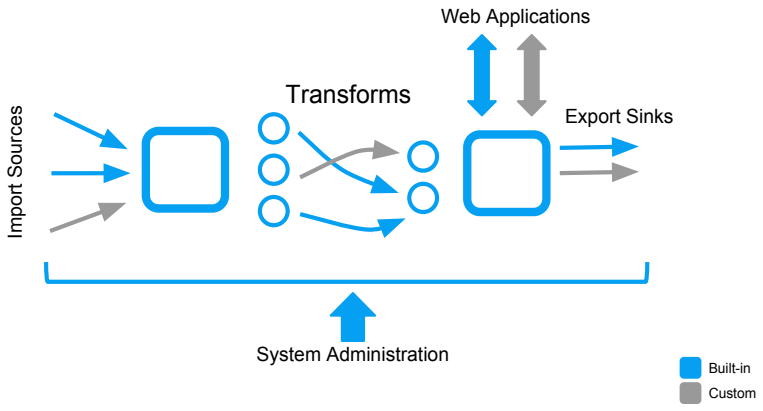
- **Record Flattening Transform** Converts Records containing complex fields such as lists or maps of fields to values to simple Records only containing one set of fields each with a simple value, which are suitable for exporting to a CSV file or Relational Database.
- **Sample Transform** Allows users to specify a percentage of Records to export, and a random number of records will be exported proportional to the percentage specified. For example if a Data Collection contains 10,000 Records, and the user specifies 5% of records to sample, the Export job will output approximately 500 randomly selected records.

Extensibility

In addition to many built-in capabilities, the Koverse platform can be extended by users who require custom functionality. In this way, analytic developers can spend more time on refining their analytics, rather than wasting time developing the underlying architecture. Not only does this reduce the time it takes to gain meaningful, tailored insights from raw data, but it ensures that new analytics are beholden to Koverse's robust security architecture.

There are several places Koverse can be extended to accommodate unique input/output data types, business-specific analytical logic, custom user interface requirements, and integration with enterprise identity management systems. Extension is accomplished by *Addons*, which can consist of one or more of the following:

- Custom Import Sources
- Custom Transforms (including Import-time and Export-time Transforms)
- Custom Export Sinks
- Customer Export File Formats
- Custom Web Apps Applications



Koverse Integration Architecture

Additionally, Koverse enables integration with existing enterprise identity management systems via extensible authentication and authorization modules.

Below is a list of the developer resources that are available for extending Koverse. Please see the *Koverse Developer Documentation* for details.

- Java SDK for Custom Sources,Transforms, and Sinks
- REST API
- Javascript REST API Library
- Java Thrift API Library
- Javascript App API

Additional Resources

Koverse software ships with documentation and SDKs available for direct download. Change the host name in the following URL to match your Koverse instance hostname.

`https://<hostname>/Koverse/docs`

Contacting Koverse Sales,Training, and Support

Koverse, Inc. offers 24/7 paid support and comprehensive training for all Koverse products.

Sales

1-855-403-1399

sales@koverse.com

Training

1-855-403-1399 (select sales for new training, or support for existing training)

training@koverse.com

Support

1-855-403-1399

support@koverse.com

** Service Level Agreements (SLAs) support requests can only be initiated via phone contact.

Koverse User Guide

The Koverse User Guide provides instructions for using the applications that are accessible via the Koverse Applications Dashboard. The User Guide Dashboard includes a number of icons which provide a link to all applications that have been deployed to the Koverse software platform.

> ***Note***
>
> All images displayed in this online document are 'clickable'. When clicked, the image open in a new windows in full size.

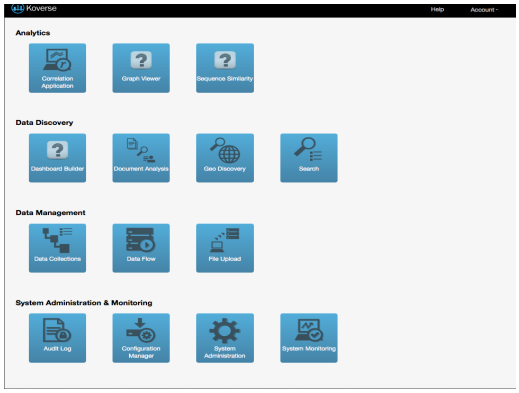
Utility Navigation Bar

For easy navigation, account maintenance and general help, the Koverse Dashboard provides a **Utilitiy Navigation Bar** and is always available when navigating through all the various Koverse applications. Please Note that clicking the Koverse logo at the left of the bar will take you home to the Dashboard from any child application page.

Navigation Bar Features:

- Application Navigation
- **Edit and view account details**
    - email address
    - first last name
    - authorization tokens
    - edit password
- **Help**
    - Koverse User Guide
    - Koverse Administration Guide
    - Koverse Operations Guide
    - Koverse Developers Documentation
    - Use Cases
    - Tutorials
- Logout of Koverse

The Koverse Application Dashboard



The Koverse dashboard categorizes the suite of features into the follow application groups:

- ***Analytics Application Group***
    - Correlation Application
    - Graph Viewer
    - Sequence Similarity
- ***Data Discovery Application Group***
    - Dashboard Builder
    - Document Analysis
    - Geo Discovery
    - Search
- ***Data Management Application Group***
    - Data Collections
    - Data Flow
    - File Upload
- ***System Administration Application Group***
    - Audit Log
    - Configuration Manager
    - System Administration
    - System Monitoring

Each application's features, functionality and usage are discussed and presented throughout this user guide.

Analytics Application Group

The **Analytics Application Group** provides a package of applications which provide abilities to examine big data to uncover hidden patterns, unknown correlations and other useful information that can be used to make better decisions. By using the Koverse analytic tools in this application group, data scientists and others can analyze huge volumes of data that conventional analytics and business intelligence solutions can't touch.

Correlation Application

Koverse's Correlation application was specifically designed to be used with the Pearson Correlation analytic.

1. Click the Koverse Logo on the black **Navigation Bar** at the top of the screen.
2. Click the **Correlation Application** Icon.
3. Click the Setup tab. This shows a list of collections that are appropriate for analysis using the Pearson Correlation transform.
4. Click the Analyze button. The Pearson Correlation transform is now running.
5. You can view the transform progress in the Data Flow Application by clicking on the black Navigation Bar and selecting Data Flow Icon.
6. Wait for the transform to complete - based on the current load of the system.
7. After the transform is complete Koverse will profile and index the analytical results. The progress and output of these jobs can be seen in the Data Collections app.

Data Discovery Application Group

The **Data Discovery Application Group** bundles all the visualization elements together to deliver indexing, search and query abilities, and advanced capability of exploring data.

Dashboard Builder

The **Dashboard Builder** application is a tool to visualize fields within a collection and view visual summaries of the entire collection or of search results.

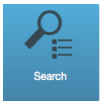
Document Analysis

The **Document Analysis** application enables you to collect and audit correlated data using the HP Fortify Runtime Hybrid Analysis technology. The Koverse platform comes with several build-in advanced analytics and machine learning algorithms.

These include, for example, the Pearson correlation, nearest neighbor calculation, simple regression scoring, and comprehensive in-0database near real-time aggregations such as TopK, sum, geobin, min, max, average, cardinality estimation, sets, quantile estimation, etc.

Geo Discovery

The **Geo Discovery** application is used to create a heat-map from geographical data that can be explored by zooming and panning.



Search Application

The Search Application provides the user the ability to interactively query one or more Koverse Collections.

- To learn more about **Search** functionality, click *Search Application*
- To learn more about **Search Query Syntax**, click *Query Syntax*

Data Management Application Group

The **Data Management Application Group** delivers all the abilities to load and manage the data collections where all your information resides.

Data Collections

The Data Collections Application provides the users the ability to manage and explore Data Collections. A Data Collection is simply a named collection of records. Collections are the primary mechanism by which data is tracked and managed in Koverse.

- To learn more about **Data Collection** functionality, click *Data Collection Application*
- A data collection is made up of data from one or more data sources
- Import jobs transfer data from outside sources, into a data collection
- All data in all data collections have the same meta-information:
    - Data Access rules and permissions
    - Field statistics
    - Record Samples
- Access to query or update a collection can be granted to one or more groups of users
- Data Collection capacity is limited only by the available disk space, which is shared amongst all data collections.
- Click the Collections tab
- Click Add Data Collection and follow the prompts
- Name: give the collection a name.
- Description: provide a textual description of the collection.
- Data Permissions: enter the data permission information by group.(see configuring collection permissions for more info)

- Data Model: in the data model section you can specify any information you want to record about the data model which applies to the data in the data collection. (see Configuring Collection Data Model for more info)

To learn more about the **Data Collections** application features and functionality, please refer to: *Data Collection Application*

Data Flow Application

Koverse **Data Flow** application provides a graphical rendering of the association between data sources and data sinks.

To learn more about the **Data Flow** application features and functionality, please refer to: *Data Flow Application*

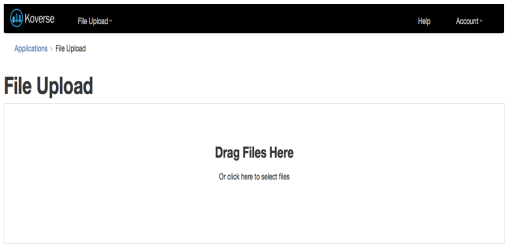
The Data Flow application consists of five logical operational tabs:

Tab	Link
Flow	*Flow Tab*
Imports	*Imports Tab*
Transforms	*Transforms Tab*
Exports	*Exports Tab*
Jobs	*Jobs Tab*

Each Data Flow Tab provides functionality for managing data flows.

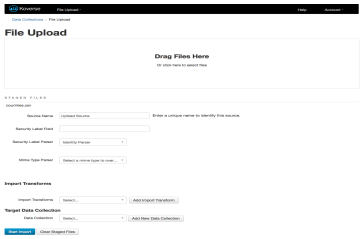
File Upload

File Upload Application provides the ability to ingest local files from the file system into Koverse, and load them into existing or new Data Collections. The user interface allows either the 'dragging' of a file, or the ability to ingest a file from a 'file system browser'.



Screen Snapshot: File Upload Drag and Drop Area

After selecting the file to load into the 'File Upload' staging cache, a series of parameters and configuration options are displayed to properly configure your file upload operation.



Screen Snapshot: File Upload Parameters

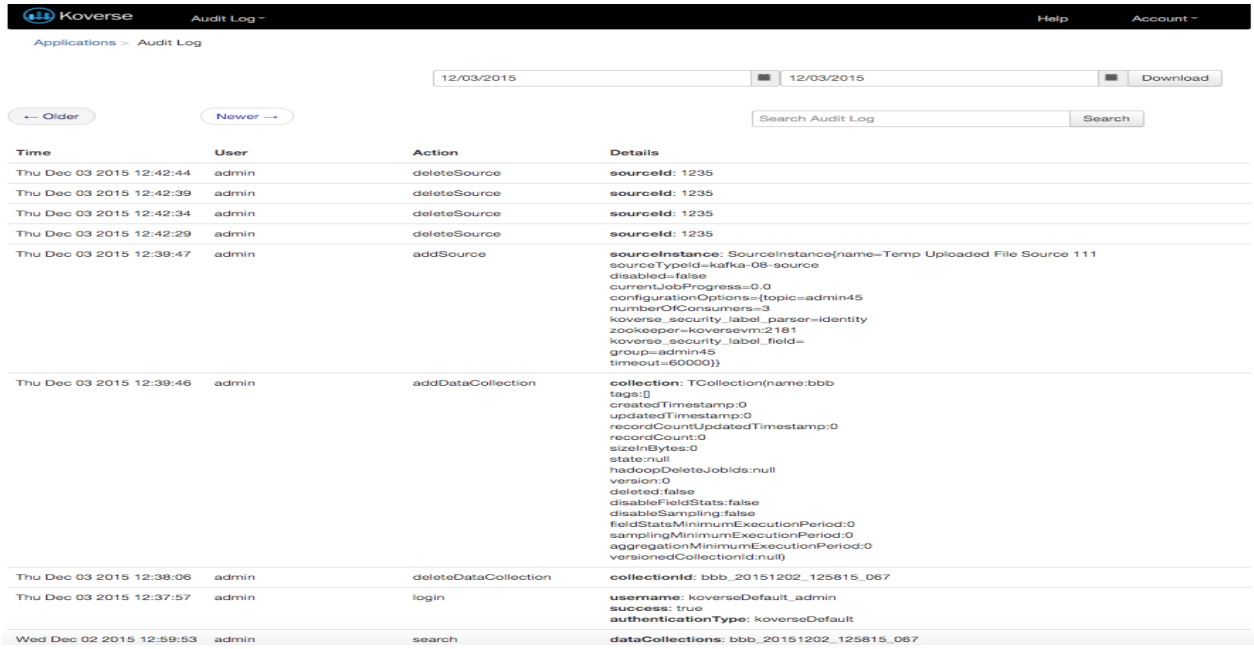
System Administration Application Group

Audit Log

The Audit Log application displays details of all user activity, sorted in the order of the most recent events.

For each event, the following information is displayed:

Column Name	Description
Time	Date and Time of the event
User	The name of the user who performed the activity
Action	The type of the event
Details	Event details

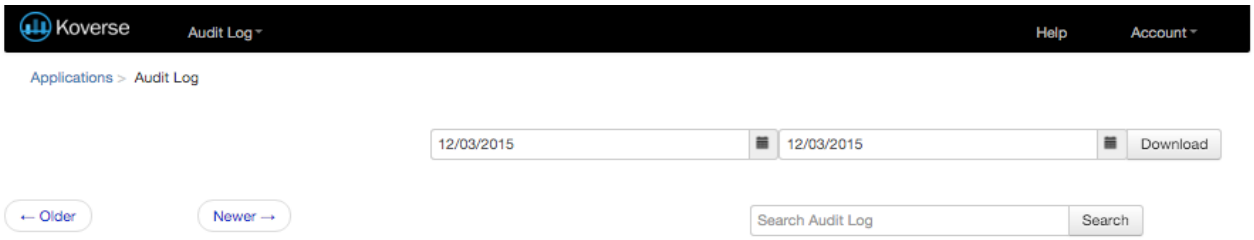


Screen snapshot: Audit Log Details

There are two features that allow the user to perform audit event analysis beyond scrolling forward and backward through the messages.

The first feature is the **Download** feature which allows the user to download a JSON formatted file that contains audit events for a date range.

The second feature provides key-word **Search** features for events of interest, such as a specific user's activity or a specific type of event.

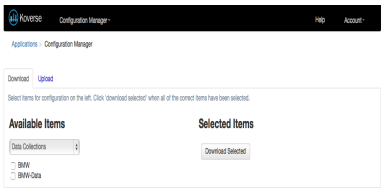


Screen snapshot: Audit Log Search and Download Feature

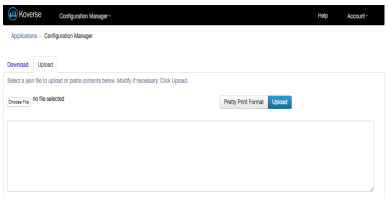
Configuration Manager

The Configuration Manager application gives users the ability to upload and download configuration for Data Collections, Sinks, Sources, and Transforms. In this manner, new Koverse instances can be stood up or reinitialized without the laborious task of reconfiguring all of these items. Note that no actual data will be uploaded or downloaded from within this app, only *configuration*.

The Configuration Manager consists of two tabs applications **(Download and Upload)**.



Screen snapshot: Configuration Manager Download Tab



Screen snapshot: Configuration Manager Upload Tab

System Administration

Koverse System Administration application provides access to common administrative tasks for the product. The administrative abilities are grouped into category tabs of 'Users', 'Groups', 'System', 'Add-Ons', 'Applications' and 'API'.

- To learn more about **System Administration** application, click *System Administration Application*

System Monitoring

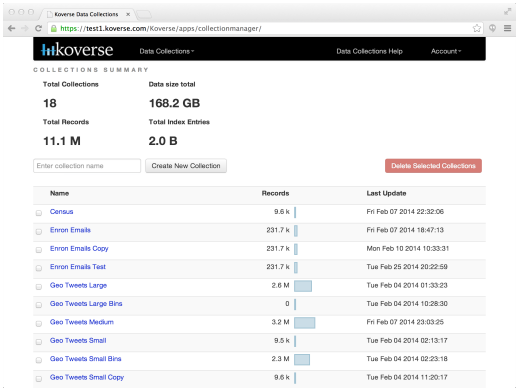
The System Monitoring App gives a view into the health and status of the distributed cluster on which Koverse is running.

- To learn more about **System Monitoring** application, click *System Monitoring App*

Data Collection Application

Overview

Users can view the status of the overall system on the first page of the Data Collections application. A list of all the Collections that the user is authorized to read are displayed. Metrics on the Collections are displayed including and the number of records in each collection.

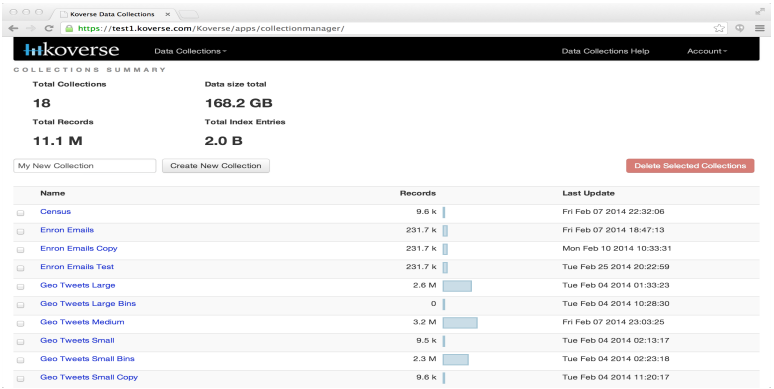


Screen snapshot: Data Collection overview

Create a New Collection

A new data collection can be created from the **Data Collections** application simply by typing in the new collection name and clicking the 'Create New Collection' button.

To add data to a collection. * First ensure that the data collection exists. All data collections can be seen by clicking on the collections tab and are listed on the main page. * Select the desired data source from the Import tab. and click load data source and select the desired collection as the destination of the import.



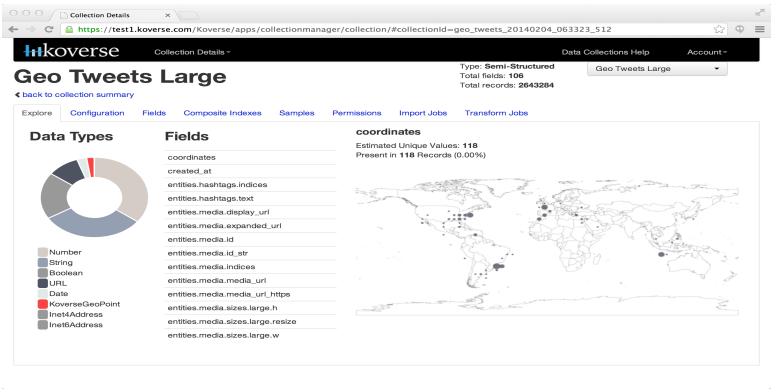
Screen snapshot: Data Collection Home Screen where a new collection can be created.

This new collection will be empty until some data is imported into it. This can be done in the Data Flow app, but also simply from the Import Jobs table of the Collection Details page, which is described next.

Viewing a Collection's Details

To view the details of a particular collection, click on the collection name in the list of collections on the overview page. This will show a new page with tabs for the various aspects of a collection:

- Explore - shows a breakdown of the data types found in a collection along with a list of fields and top values found for each field.



Screen snapshot: Viewing Collection details via the Explore option.

A new collection will be missing much of this information, as it is gathered from the data within the collection. To import data into a new collection, follow the instructions under 'Adding Data to a Collection' in the next section.

In the collections tab click on configure and then click on configure access. This will take you to the access control view of the collection. In this area a user can:

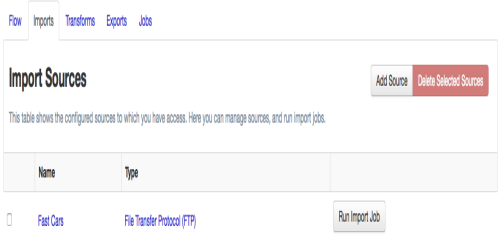
- Can control access to every column of every dataset(in later versions you can control access to each individual cell of every dataset).
- Define a default access control rule for un-identified elements in the data.
- View, and optionally mark, the observed data elements from within already loaded datasets.

Adding Data to a Collection

Data can be added to a collection from any Data Source that the user has access to. Creating new sources must be done in the Data Flow application *Data Flow Application*.

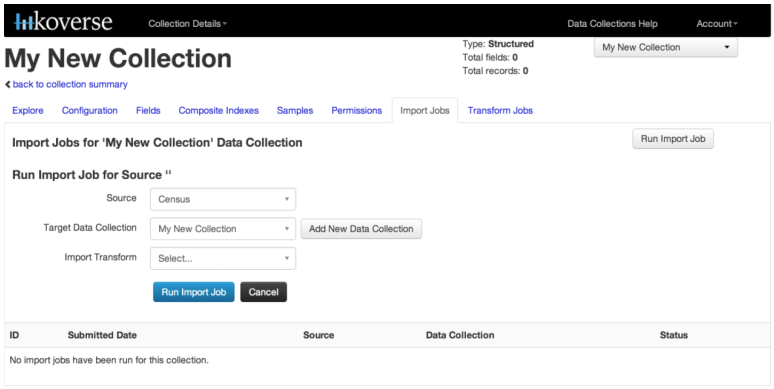
To import data from a Data Source into a collection:

- Click the **Imports** tab in the Data Flow application screen
- Click the **Run Import Job** button on the right.
- Select a Data Source from which to import
- Select the name of the Collection into which to import the data
- Optionally select any import-time transforms to apply to this import job
- Click the blue *Run Import Job* button



Screen snapshot: Imports Tab Screen

Editing Collection Details, Deleting, and Clearing a Collection



Screen snapshot: Collection Details screen.

Deleting a data collection removes all data contained within that data collection, and all meta data. Scheduled jobs that use the data collection will fail.

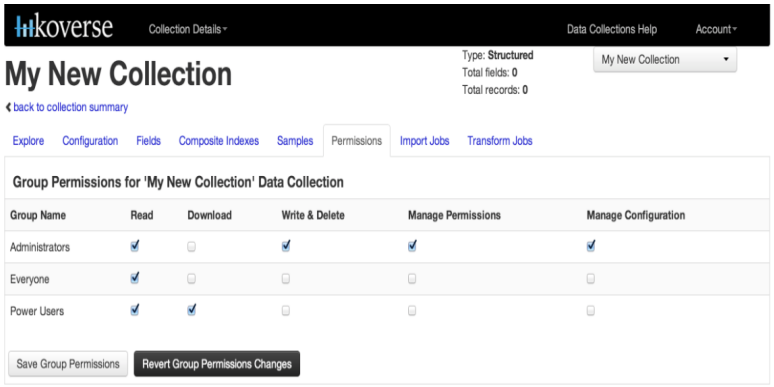
Clearing a collection removes all records within a collection but leaves the configuration information intact including indexing policy, user and group access.

Configuring Collection Permissions

Koverse supports Role-based Access Control for access to Collections. The 'Permissions' tab allows users to configure which groups of users can perform various actions on a collection including:

- Read (perform queries and use this collection as the input in transforms)
- Download
- Write & Delete
- Manage Permissions (change which groups can access)
- Manage Configuration (change details like indexing for this collection)

To affect changes, simply check the boxes that represent the type of access to grant for the desired group and click 'Save Group Permissions'.

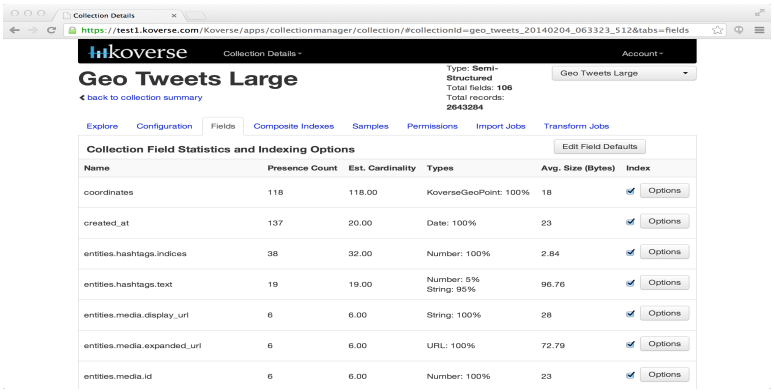


Screen snapshot: Collection permissions screen.

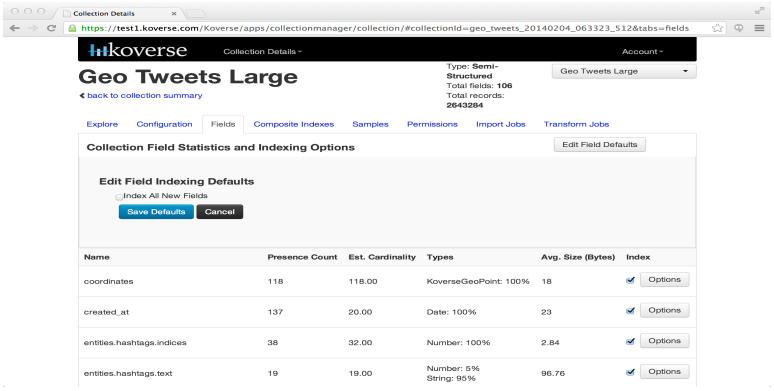
Configuring Indexing

In the Collection Details view click on the "Fields" tab to see the list of fields and indexing options. This information is gathered from the actual data that has be ingested into the collection and is updated when new data is imported. This field information for a collection is passed along with raw data to both internal Koverse processes and to processes accessing the data via the SDK. Within the Fields view user can:

- Choose fields to index
- Configure indexing options for a particular field
- Set the policy for how to index new fields

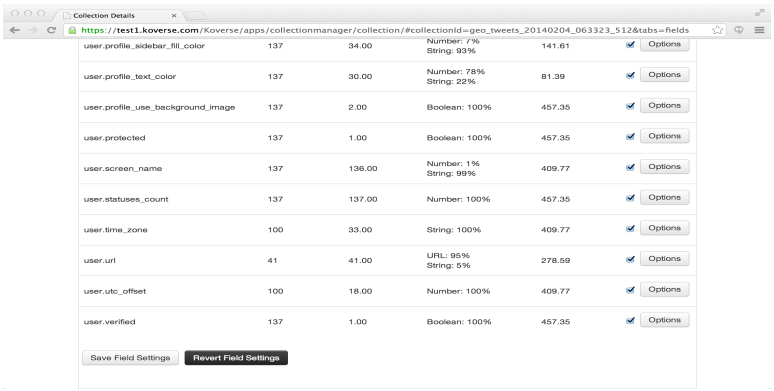


For collections with no data, Koverse can automatically index all fields in records that may be imported into this collection in the future by clicking the 'Edit Field Defaults' button and checking 'Index All New Fields' and clicking the 'Save Defaults' button.

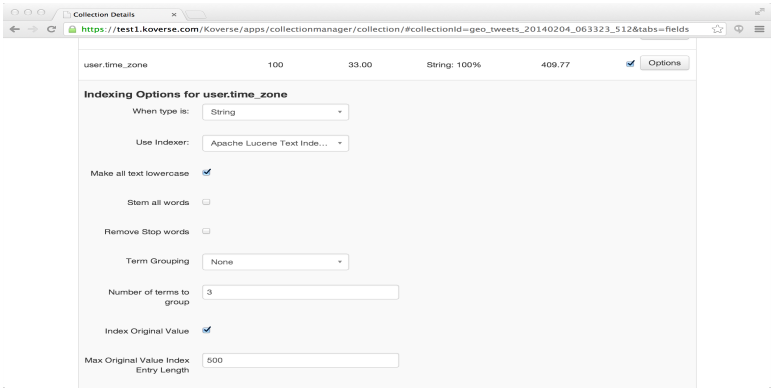


To save time and disk space, users may wish to choose to not index any fields until after the first import of data is complete. Koverse will then show a list of every field found in at least one record, including information about the type of the field, how often it is present in records, and estimates for things like cardinality (ie the number of unique values in this field), average size, etc.

This information is used to help users decide what fields to index, and to understand what kind of information is found in each field. Users can then choose to index particular fields by checking the box on the right of the table. After a number of fields are checked for indexing, users can put the new indexing policy into effect by clicking the 'Save' button at the bottom of the page.



Additional options for some types of fields are available by clicking the options button to the right of the check-box. For example, for String types, users can choose to index the whole field as it appears, or to tokenize the text found and index it in additional ways such as lowercase in addition to whatever case already exists, to remove stop words (common words like 'the'), and whether to index pairs or triples of tokens and so on (also known as n-grams). Clicking 'Save' after choosing these options will also put them into effect.



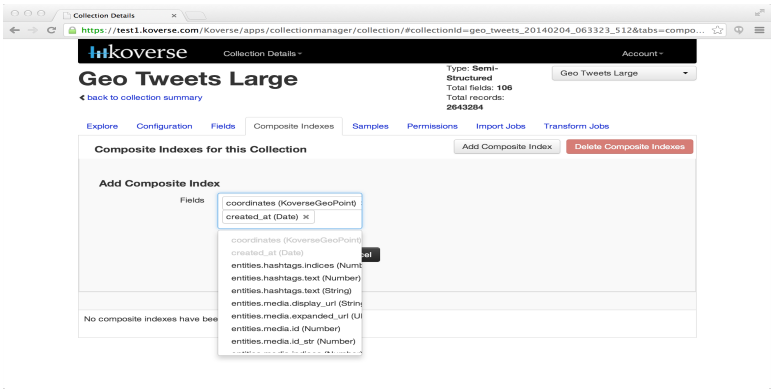
Whenever an indexing policy for a collection has changed, Koverse will automatically update the on-disk indexes using a MapReduce job. The status of this job can be viewed in the Health and Monitoring application.

Koverse automatically updates any indexes present for a collection as new data is imported.

Composite Indexes

Composite indexes are indexes built on two or more fields to enable querying combining ranges across those fields in queries. For example, in order query on a range of values in a field called 'height' and also a range of values in a field called 'age', a composite index must be created that will enable this query to run quickly, without doing expensive set operations on the server side.

Creating composite indexes is simple. In the Collection Details view, users can click on the 'Composite Indexes' tab to see a list of the composite indexes that already exist. New composite indexes can be created by clicking 'Add Composite Index' and selecting two or more fields from the drop down menu that appears.



Each entry in the drop down menu consists of a field name and a type. For example we might see 'height (string)' and 'height (float)'. This means that both string (text) and floating point number values have been observed in this field. It might be likely that the string values are erroneous. In any case, we wish to query for ranges across numerical values for the height field in our example so we choose 'height (float)'.

Users can choose up to four fields in one index, but beyond four users may start to see performance issues with queries.

Clicking the black 'Add Composite Index' button will cause Koverse to begin building this index on any data already available, and Koverse will update this index if any new data is imported into this collection.

- Click the Collections tab in the main menu.
- If you do not have a Collections tab in the main menu, your user account is not a member of a group with the Manage Data Collections permission.
- Find the data collection to modify or delete.
- Click the Edit or Delete button for the appropriate data collection.
- Confirm the edit or delete.

Data Flow Application

The Data Flow application gives users the ability to visualize, configure, and execute the movement of data within the Koverse system. It is important to note that users will only be able to view and interact with data and jobs for which they have the permission to do so.

Flow Tab

The flow tab shows a visualization of the Transforms that are configured. Users will be able to view Transforms on Collections they have permission to read. The flow starts on the right and moves left, showing how initial Collections are transformed into new Collections. Clicking on an individual section of the flow will take the user to the configuration details for that Transform.

If the user has appropriate permissions, the Flow tab also provides the ability to configure new Sources, Transforms, and Sinks:

Adding an Import Source

1. Click the "Add Import Source" button. If this button is not visible, you do not have permission to configure Sources.

2. Select the type of Source you wish to add from the dropdown. (Note that the list of available Sources will include all built-in Sources, in addition to any custom Sources that have been uploaded to Koverse as part of an Addon. See the *Addons* section for instructions on uploading Addons.)
3. Fill out the Source configuration fields. (See the tips below on configuring some of the more advanced Sources.)
4. To Optionally add Import Time Transforms - select a desired import time transfrom from the list, and then click "Add Import Transform". You may add more than one.
5. Select a output data collection for storing records from this source.
6. Select the type of flow - either manual, periodic, or continuous. Manual means that a user must kick off of jobs. Periodic means the job will be run on a defined schedule. Continuous means the job will be run continuously until this setting is changed.
7. Click the "Add Source" button. Note that data will not actually be imported until this source is run from the *Flow Tab*.

Tips for configuring particular Sources:

Configuring a Twitter Source

1. *Create Twitter Dev Application:*

    1. Log onto https://dev.twitter.com/
    2. Under your avatar, select My Applications
    3. Click Create a new application button

        - ex name: twitter koverse test
        - ex description: twitter koverse test
        - ex website: `http://localhost.com/Koverse`
        - callback: none
    4. Select Yes, I agree to the Developer Rules Of The Road
    5. Enter CAPTCHA displayed
    6. Click Create your Twitter application
    7. Your Twitter application properties should now be shown on the next page
    8. Click Create my access token
    9. Refresh page to have access tokens appear
    10. Leaving the details page open move onto Creating the Koverse Twitter Streaming Source
2. *Configuration Options for the Twitter Source in Koverse:*

    New Source Type: - Select Twitter Streaming"

    - Source Name: example 'twitter koverse test'
    - Security Field Label: (optional)
    - Security Label Parser: Identity Parser
    - Twitter App Consumer Key: Copy & Paste details from Twitter Dev App
    - Twitter App Consumer Secret: Copy & Paste details from Twitter Dev App

- Twitter App Access Token: Copy & Paste details from Twitter Dev App
- Twitter App Access Token Secret: Copy & Paste details from Twitter Dev App
- Keywords(optional)
- Locations (optional)

Note: Twitter Streaming Sources will continue to update every 10 minutes unless you stop the job.

Configuring an Email Account (IMAP) Source

- Input the following required fields:
  - Source Name - example: Personal Gmail
  - Server - example: imap.gmail.com
  - Username - example: youremail@gmail.com
  - Password - example: password123
  - Security Label Field (optional)
  - Security Label Parser (Default = Identity Parser)

Configuring an Newsfeed Source

- Input the following required fields:
  - Source Name - example: NY Times Business
  - Security Label Field (optional)
  - Security Label Parser (default: Identity Parser)
  - RSS Feed URL - example: http://www.nytimes.com/services/xml/rss/nyt/Business.xml
  - Polling Frequency (in minutes) - example: 5

Configuring an Amazon S3

- Source Parameters:
  - Source Name (Required)
  - Security Label Field (Optional)
  - Security Label Parser (Dropdown)
  - Access Key ID (Required) - This is actually a username. It is alphanumeric text string that uniquely identifies the user who owns the account.
  - Secret Key (Required) - This key plays the role of a password.
  - Mime Types (Optional) - This is optional however we recommend to always select a Mime Type.
  - Include files in subdirectories (Checkbox) - This is optional and if checked includes files in the subdirectories of the S3 Bucket you specify.
  - Import files with names matching regular expression - Specifying the name of the file(s) you want to import.
  - Date

Import Sources

All of Wikipedia Source

Wikipedia offers free copies of all available content to interested users. Enabling this source will stream the Wikipedia Records into your target collection.

Newsfeed Source

The Newsfeed source allows users to import information directly from RSS feeds.

Amazon S3

The Amazon S3 import source allows users to import files directly from Amazon's S3 service.

Apache Commons VFS

Commons VFS provides a single API for accessing various different file systems. It presents a uniform view of the files from various different sources, such as the files on local disk, on an HTTP server, or inside a Zip archive.

Email Account (IMAP)

IMAP is an Internet standard protocol used by email e-mail clients to retrieve e-mail messages from a mail server over TCP/IP connection.

File Transfer Protocol (FTP)

The File Transfer Protocol is a standard network protocol used to transfer computer files from one host to another host over a TCP-based network, such as the internet.

Hadoop Distributed File System (HDFS)

The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file-system written in Java for the Hadoop framework. A Hadoop cluster has nominally a single namenode plus a cluster of datanodes, although redundancy options are available for the namenode due to its criticality. Each datanode serves up blocks of data over the network using a block protocol specific to HDFS.

Kafka 0.8 Source

Apache Kafka is an open-source message broker project developed by the Apache Software Foundation written in Scala. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds.

MS SQL Server (Microsoft SQL Server)

Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications which may run either on the same computer or on another computer across a network (including the Internet).

MySQL

MySQL is a relational database management system (RDBMS), it was the world's second most widely used RDBMS, and the most widely used open-source RDBMS.

Oracle 11gR2

Oracle 11gR2 is the second and terminal release of the Oracle 11g database.

Oracle RAC 11gR2

In database computing, Oracle Real Application Clusters (RAC) - provides software for clustering and high availability in Oracle database environments.

PostgreSQL

PostgreSQL, often simply Postgres, is an object-relational database management system with an emphasis on extensibility and on standards-compliance. As a database server, its primary function is to store data securely, supporting best practices, and to allow for retrieval at the request of other software applications.

URL Source

URL source is our own custom source type that allows our users to select multiple comma separated list of URL's, the specifying file names, the date, and import them into your target collection.

Web Crawler

Web Crawler uses our Kafka streaming to allow you to specify the number of workers, the starting URL, the maximum levels to crawl, the broker list, and the zookeeper services to utilize.

Wikipedia Page Sources

Wikipedia offers free copies of all available content to interested users. Enabling this source will stream a single page source from the Wikipedia Records into your target collection.

Adding a Transform

1. Click the "Add Transform" button. If this button is not visible, you do not have permission to configure Transforms.
2. Select the type of Transform you wish to add from the dropdown. (Note that the list of available Transforms will include all built-in Transforms, in addition to any custom Transforms that have been uploaded to Koverse as part of an Addon. See the *Addons* section for instructions on uploading Addons.)
3. Assign a name to the Transform in the "Name" field.
4. Fill out the Transform configuration fields.
5. Click the "Add Transform" button. Note that the Transform will not actually execute until it is run from the *Transforms Tab*.

Adding a Sink

1. Click the "Add Export Sink" button. If this button is not visible, you do not have permission to configure Sinks.
2. Select the type of Sink you wish to add from the dropdown. (Note that the list of available Sinks will include all built-in Sinks, in addition to any custom Sinks that have been uploaded to Koverse as part of an Addon. See the *Addons* section for instructions on uploading Addons.)
3. Fill out the Sink configuration fields.
4. Click the "Add Sink" button. Note that the Sink will not actually export any data until it is run from the *Exports Tab*.

Imports Tab

Users will need permission to manage Sources in order to access all of the features in this tab.

Configuring an Existing Data Source

Locate the existing data source you wish to modify in the list on the main page and click on the link to be taken to the configuration page for that Source. The configuration page has several tabs:

- **Configuration Tab:** Shows the existing configuration for this import Source.

- If the configuration looks good, you can choose to import data at this point by hitting the "Run Import Job" button. See *Running an Import Job* for details.
- If the configuration needs to be modified, hit the "Edit Configuration" button and follow the instructions in the *Adding an Import Source* section.
- If you wish to delete this Source, hit the "Delete Source" button. Note that this will not delete any data that has already been imported, it will only delete the Source's definition.

- **Import Flows Tab:** Shows any imports flows for this Source. A source may have more than one flow. Flows connect sources to data collections, with specific import time transforms and schedules.
  - If you wish to create an additional import flow for this source
    1. Click the "Add Import Flow" button.
    2. Optionally add Import Time Transforms.
    3. Select the output Data Collection.
    4. Select the flow type - manual, periodic, or continuous.
    5. Click "Add Import Flow"
  - If you wish to edit an Import flow, click the "Edit" button on the desired import flow row.
  - Existing Import Flows can be removed by checking the flows to be deleted, and then clicking the "Delete Selected Import Flows" button.
- **Permissions Tab:** Use the check boxes to assign import, edit and delete permissions to various user groups for a given Source. Note that *import* permission will allow users to actually import data from the Source, *edit* permission will allow users to edit the configuration of the Source, and *delete* permission will allow users to delete the Source's definition. These permissions do not relate to permissions to access, edit or delete any particular data Collection and only relate to configuring and executing the Source itself.
- **Jobs Tab:** This tab displays a list of the historical import jobs that have run to import data from this Source. In order to run a new import job, click the "Run Import Job" button and follow the instructions in the *Running an Import Job* section

Running an Import Job

1. Click the "Run Import Job" button. If more than one Import Flow is present on a source, you will be presented with the option of which to run. If only one Import Flow is configured for the source, that Import Flow will be run - and you will be taken immediately to the import job progress page.

Deleting a Source

Check the box next to the Source(s) you wish to delete and hit the "Delete Selected Sources" button. Note that this will not delete any data from the Koverse data store, and will merely delete the Source definition(s).

Transforms Tab

Users will need permission to manage Transforms in order to access all of the features in this tab.

Configuring/Running an Existing Transform

Locate the existing Transform you wish to modify in the list on the main page and click on the link to be taken to the configuration page for that Transform. The configuration page has two tabs:

- **Configuration Tab:** Shows the existing configuration for this Transform.
    - If the configuration looks good, you can choose to transform data at this point by hitting the "Run Transform Job" button.
    - If the configuration needs to be modified, hit the "Edit Configuration" button and follow the instructions in the *Adding a Transform* section.
    - If you wish to delete this Transform, hit the "Delete Transform" button. Note that this will not delete any data that has already been processed, it will only delete the Transform's definition.
- **Jobs Tab:** This tab displays a list of the historical jobs that have run to execute this Transform. In order to run a new Transform job, click the "Run Transform Job" button and observe the progress of the Transform as it runs to completion.

Deleting a Transform

Check the box next to the Transform(s) you wish to delete and hit the "Delete Selected Transforms" button. Note that this will not delete any data from the Koverse data store, and will merely delete the Transform definition(s).

Exports Tab

Users will need permission to manage Sinks in order to access all of the features in this tab.

Configuring an Existing Export

Locate the existing export you wish to modify in the list on the main page and click on the link to be taken to the configuration page for that export. The configuration page has two tabs:

- **Configuration Tab:** Shows the existing configuration for this Sink.
- **Import Jobs Tab:** This tab displays a list of the historical export jobs that have run to import data from this Source. In order to run a new export job, click the "Run Export Job" button and follow the instructions in the *Running an Export Job* section

Running an Export Job

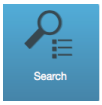
1. click the "Run Export Job" button.
2. Select the Collection you wish to export data from.
3. Select the Sink you wish to export data to.
4. Optionally select any Export Transforms to apply to this particular Export job and fill out any parameters they might have.
5. Click Run Export Job

Deleting a Sink

Check the box next to the Sink(s) you wish to delete and hit the "Delete Selected Sinks" button. Note that this will not delete any data, and will merely delete the Sink definition(s).

Jobs Tab

This tab displays a list of all the jobs that have run and are running on the system. You may choose to stop a job that is currently running by selecting the box next to the job and then clicking the "Cancel Selected Jobs" button. Choosing this option for a job that has already completed will do nothing.

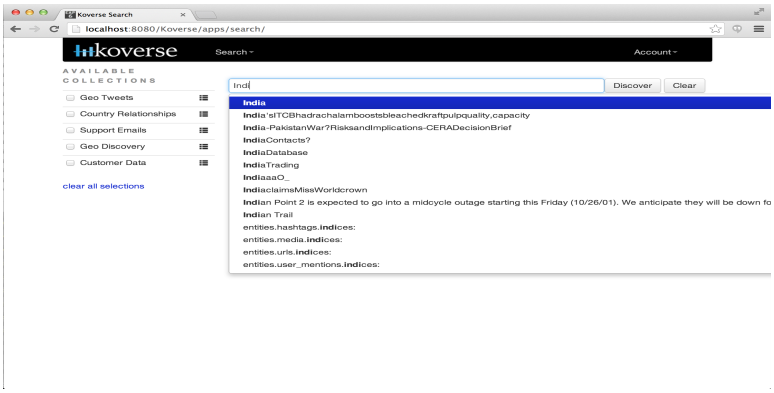


Search Application

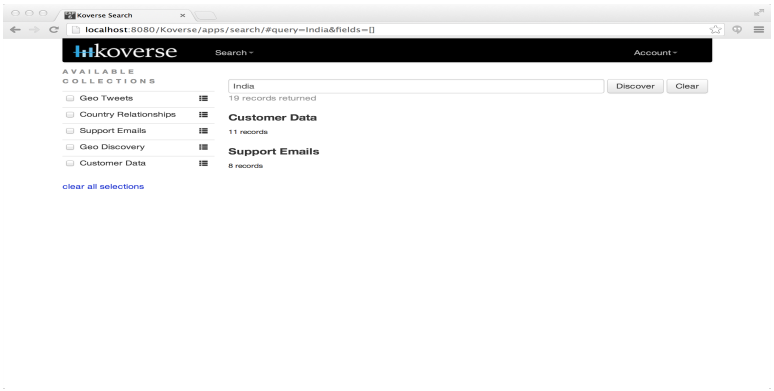
The Search application provides users the ability to interactively query one or more Koverse Collections to find all Records that match their search criteria. While many end-user analytics will be custom in nature and rely heavily on data model semantics, the data discovery application allows users to search in a schema-free manner, or to use the schema as necessary. Within the data data discovery tab a user can:

- Search across all collections or specific collections
- Search across any field or within specific fields
- Combine search terms
- Search for a range of values

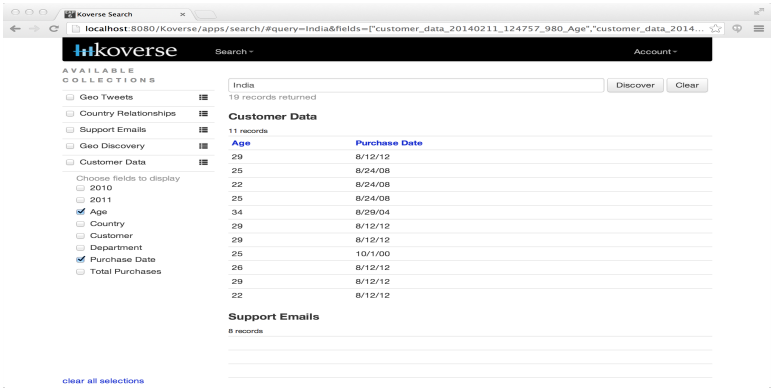
Initially, the Search application starts with nothing selected, and the application will return records containing search terms in any field of records from any collection. Users can start by typing search terms into the search bar. Koverse will auto-suggest terms to search based on the selected collections. The search bar will show terms from any field of any collection that match the prefix typed thus far, as well as any field names that can be searched.



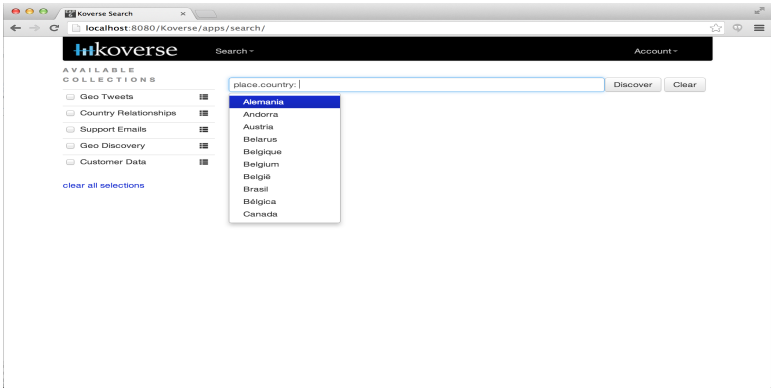
The Search application returns results grouped by collection.



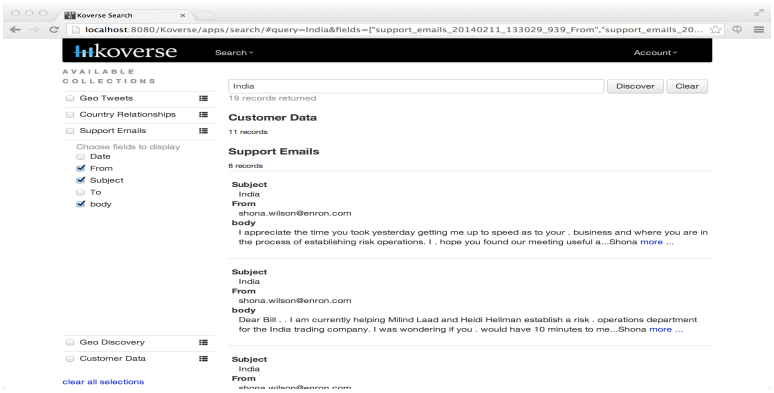
Users can then choose to show specific fields within those search results by expanding the set of field names for a collection and selecting fields to show on the left of the screen.



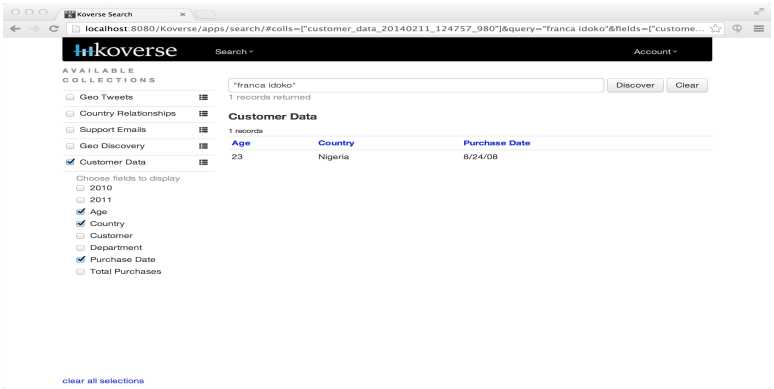
Koverse Search also auto-suggests fields that appear in all or selected collections, and users can choose a field in which to search. Auto-suggest will then only suggest terms that appear within the selected field.



Koverse handles search across structured (flat records), semi-structured, and unstructured (text) data. Fields containing large amounts of text are truncated to snippets. The full text can be viewed by clicking the 'more' link at the end of the snippet, and hidden again by clicking the 'close' link after expanding.



Users can choose to only search within a certain collections by checking the collection boxes on the left side of the screen. Auto-suggest will then only suggest terms and fields from those selected collections and results will only come from those collections.



A particular search can be bookmarked and shared with others by simply saving or sharing the URL from the address bar. However, if other users are not authorized to see any of the collections selected, they will simply not see those search results and will only see results from selected collections that they are authorized to see.

Query Syntax

The Search App is designed to be somewhat like Google in design. Users can simply type in terms and retrieve results that match all the terms. This means the terms are 'ANDed' together, so that records containing term1 AND term2 .. and so on are returned. There is no need to type the word AND into the search box.

Searching for records that contain a term in any field:

```
mary
```

To search for terms that contain spaces, use quotes around the terms:

```
"mary had a"
```

Searching for records that contain a term in a particular field:

```
name: mary
```

Combining Terms

Searching for records that contain a term in one field and another term in another field. This is like requesting records that match the first clause, AND the second:

```
name: mary occupation: shepherd
```

Two or more terms may be combined this way. Some terms can be field specific and others not. For example:

```
name: mary shepherd
```

Would return any records where the value "mary" appeared in the name field, and where the value "shepherd" appeared in any other field, including the name field.

Note that the difference between querying for a two-word phrase with containing a space and searching for one word within a field and one word in any field requires quotes. To search for a two-word phrase within a single field, use quotes around the two-word phrase:

```
name: "jane doe" shepherd
```

The preceding query would search for the entire string "jane doe" in the name field and the word "shepherd" in any field.

Range queries

To search for records that contain a value within a range, use square brackets and word 'TO':

```
height: [60 TO 70]
```

For an open-ended search, use an asterisk, * , to indicate positive or negative infinity. The following means return records with a value for the height field that is greater than or equal to 60:

```
height: [60 TO *]
```

The following returns all records with a value in the height field less than or equal to 60:

```
height: [* TO 60]
```

Searches can also be done across ranges of text values using wildcard syntax. Only trailing wildcards are supported. The following returns records with a value beginning with the letters 'ma' in any field:

```
ma*
```

Koverse understands the ordering of several types of values including numbers, text strings, URLs, dates, and IP addresses:

```
[192.168.1.0 TO 192.168.34.0]
```

To query a range of dates, the following formats are recognized:

```
"yyyyMMdd hh:mm:ss"
"EEE MMM d HH:mm:ss Z yyyy"
"EEE MMM d HH:mm:ss zzz yyyy"
"yyyy-MM-dd"
"yyyy-MM"
"yyyy/MM/dd HH:mm:ss"
"yyyy/MM/dd HH:mm:ss.SSS"
"MM/dd/yyyy HH:mm"
"ddHHmm'Z' MMM yy"
```

```
yyyy - four digit year
yy - two digit year
MM - two digit month
MMM - three letter month
dd - two digit day
d - one or two digit day
HH - two digit hour
mm - two digit minute
ss - two digit second
Z - time zone. such as -0800
zzz - time zone. such as Pacific Standard Time; PST; GMT-08:00
```

An example of a query for a date range is:

```
creation_date: ["20140211 11:28:08" TO "20140211 13:30:08"]
```

Another example date range is:

```
["2014-02-11" TO "2014-02-12"]
```

Note that a date format such as "20140211" is indistinguishable from a simple number, so dashes should be used if a date is meant.

Searching for records that contain a geographical point value.:

```
coordinate: [-60,-40 TO 30,35]
```

Searching a single range does not require that a composite index be built. To query multiple ranges at once or a range and other terms, a composite index must be built. These types of queries are described in the following section.

For additional information on Composite Indexes, please refer to: *Composite Indexes*

Combining Ranges

Koverse supports querying for multiple ranges or ranges and single terms simultaneously but requires that composite indexes be built first before such queries can be executed. This is because composite indexes reduce the work done at query time to just a few short scans without having to do any set operations so queries with multiple ranges can return quickly, without impacting other users of the system.

An example of a query that combines a range with a single term. To perform this query, a composite index of the height and name field is required. See *Composite Indexes* for how to build this type of index.:

```
height: [* TO 10] name: mary
```

An example of a query that combines multiple ranges. To perform this query, a composite index of the height and weight field is required.:

```
height: [* TO 10] weight: [70 TO 80]
```

To query across a range of geos and time simultaneously, do the following. To perform this query, a composite index on the geo field and time field is required.:

```
geo: [-60,-40 TO 30,35] time: ["20140211 11:28:08" TO "20140211 13:30:08"]
```

System Administration Application

The System Administration application provides a graphical user interface for system administration activities, such as system configuration, user accounts, user groups, etc. Only users with administration privileges will be able to access this App. The default administrator username and password are admin and admin, respectively. Be sure to change the default admin password.

Note that only a subset of the links shown above will be present if the current user does not have permission to perform all administrative actions.

Users

The links described below will only be present if the current user is a member of a group that has 'Manage Users & Groups' permissions.

Add User: Create a new user account.

Koverse uses email addresses as primary user IDs. To create a new user,

1. Click the Add User link.
2. Enter the new users email address.
3. Click the Add User button.

Edit User: Change a user's ID and/or manage the groups a user is in.

1. Click the Edit User link.
2. Select the target user from the drop down.
3. Enter the user's new email address, if applicable.
4. Check and uncheck the boxes next to the group names to add and remove the user from the groups.
5. Click Save when finished.

Set User Password: Initially set a new user's password, or reset the password of an existing user.

1. Click the Set User Password link.
2. Select the target user from the drop down.
3. Enter the user's new password once, and then again for confirmation.
4. Click Save when finished.

Delete User: Remove a specific user account and re-assigns responsibilities to another user who will assume all data collections, sources, jobs, and permissions management of the deleted user.

1. Click the Delete User link.
2. Select the user to be deleted from the drop down.
3. Select the user that will assume the responsibilities of the user to be deleted.
4. Click Delete User
5. Click Yes to confirm the deletion of the user.

Group/Roles

Koverse provides groups as a way to manage privileges for multiple users. Users are members of one or more group.

Add Group: Create a new group.

1. Click the Add Group link.
2. Enter the new groups name.
3. Click the Add Group button.

Edit Group Permissions: Provision system-wide permissions for the selected group. Note that if the "Add to All New Users" box is checked for a group, all new users to the system will automatically be assigned to the group and hence inherit all of the group's permissions. Also note that permissions for specific data collections are granted on the data collections themselves, and not in the System Administration App.

#. Click the Edit Group link. * Select a Group to edit from the drop down. * Check and uncheck the boxes next to the desired permissions group. * Click the Save button when finished.

Delete Group: Remove a group. It does not remove any users or data collections.

1. Click the Delete Group link.
2. Select the Group to be deleted from the drop down.
3. Click Delete Group.

System

System Configuration: Configure the system properties for Koverse, Hadoop, Accumulo, and SMTP. These properties will need to be configured correctly for Koverse to be fully functional.

In most cases, the fields are self-explanatory and the pre-populated defaults can be used. The main exception to this are the **Data Store** properties:

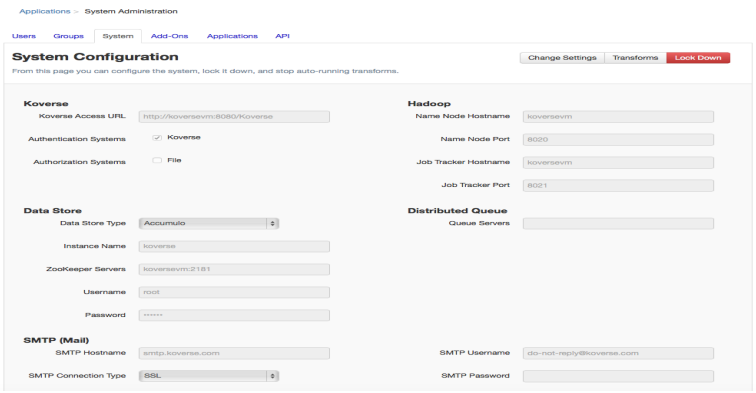


Figure: System Configuration Tab

- Data Store Type should be "Accumulo".
- Instance Name should be the name of the Accumulo instance that is configured in Accumulo.
- ZooKeeper Servers should be a comma separated list of ZooKeeper servers in the form <hostname>:<port>. *Example: zoo1:2181,zoo2:2181,zoo3:2181*
- Username is the Accumulo username that will be used to connect to Accumulo.

- Password is the corresponding password for the Accumulo user listed under Username.

When finished entering all System settings, hit the "Save" button. If there are any problems with the given settings, an error will pop up.

Lock Down: Lock down mode is used to disable all data interactions in this system. While lock down mode is enabled, only accounts with permissions to manage users, groups, view the audit log, and manage lock down mode will be able to interact with this system. If you choose to enter lock down mode, click in the "Lock Down" link and hit the "Enable Lock Down Mode" button. The same link is used to disable lock down mode.

Resources: Manage Auto-Running Transforms. Normally, transforms will run periodically on scheduled intervals. If you wish to disable this feature and only run transforms manually,

1. Select the "Resources" link.
2. Check the "Disable Auto-Running Transform" box.
3. Hit "Save".

Auto-Running Transforms can be re-enabled by un-checking the "Disable Auto-Running Transform" box.

Addons

Manage Addons: Install external jar files to extend Koverse.

1. Click the "Manage Addons" link.
2. Hit the "Choose File" button to browse files in your local file system.
3. Select a file with the .jar extension and hit "Upload".

Applications

Manage Applications: is used to configure permissions and parameters for Koverse Applications.

1. Click the "Manage Applications" link.
2. Click the Name of the Application you wish to configure.
3. Under the "Permissions" tab, check the boxes that correspond to the group permissions you wish to assign to this Application.
4. Hit "Save Permissions".
5. Under the "Parameters" tab, configure any parameter values you wish to set/change.

Deploy Application from Template: Deploy instances of Applications whose templates have been uploaded to Koverse as part of an Addon.

1. Click the "Deploy Application from Template" link.
2. Select an Application template to use from the dropdown.
3. Enter the Name of the Application. This will be the string displayed to the user under the application's icon on the Applications Dashboard.
4. Enter the Category Name for the Application. This is the string displayed for the category. Multiple Applications probably share the same string.
5. Enter the URL ID. This is the portion of the url path used to access the Application directly, as in /Koverse/apps/<url_id>/.
6. Hit "Deploy Application Template".

API

The Koverse REST API and SDK allow:

- Transform: data into specialized indexes, analytic summaries, and algorithmic processing of data within the system, from external systems and 3rd party plugins
- Data Discovery: indexing and query calls to quickly search and explore data.
- Data Collection Management: administer access permissions, data models, view provenance, and purge data.
- Import: import data into Koverse from a range of sources
- Data Export: download to external systems
- Direct access: to the data within the system, from external systems
- Auditing: access query activity of all users of the system.
- Advanced Data Discovery: apply high level analytic query logic to the data; entity chaining and disambiguation, classification etc.
- Authentication: an internal user registration and authentication service.
- Authorization: an internal authorization service.

These links are used to manage API tokens.

Add API Token: Create a new API token.

Enter a name for the new API token and click "Create API Token"

Edit API Token: Edit properties of an already existing API token.

1. Click the "Edit API Token" link.
2. Choose the token you wish to edit from the dropdown.
3. Optionally change the Token Name.
4. Optionally change the Responsible User by selecting a new user from the dropdown.
5. Optionally edit the group permissions you wish to assign to the token under "API Toke Group Membership".
6. Click "Update Token" when finished.

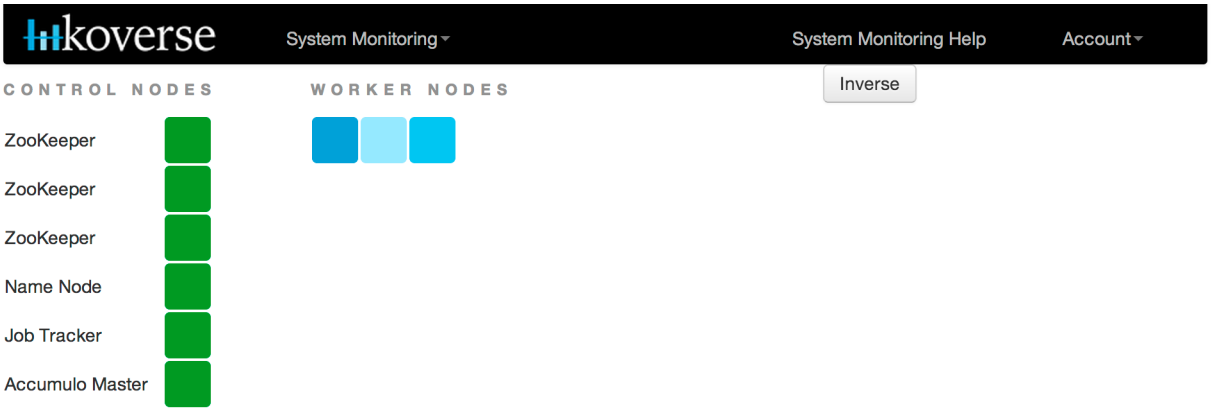
Delete API Token

Select the API token you wish to delete from the dropdown and click "Delete API Token".

System Monitoring App

The System Monitoring App gives a view into the health and status of the distributed cluster on which Koverse is running. There are several different sections of the monitoring view:

Control Nodes

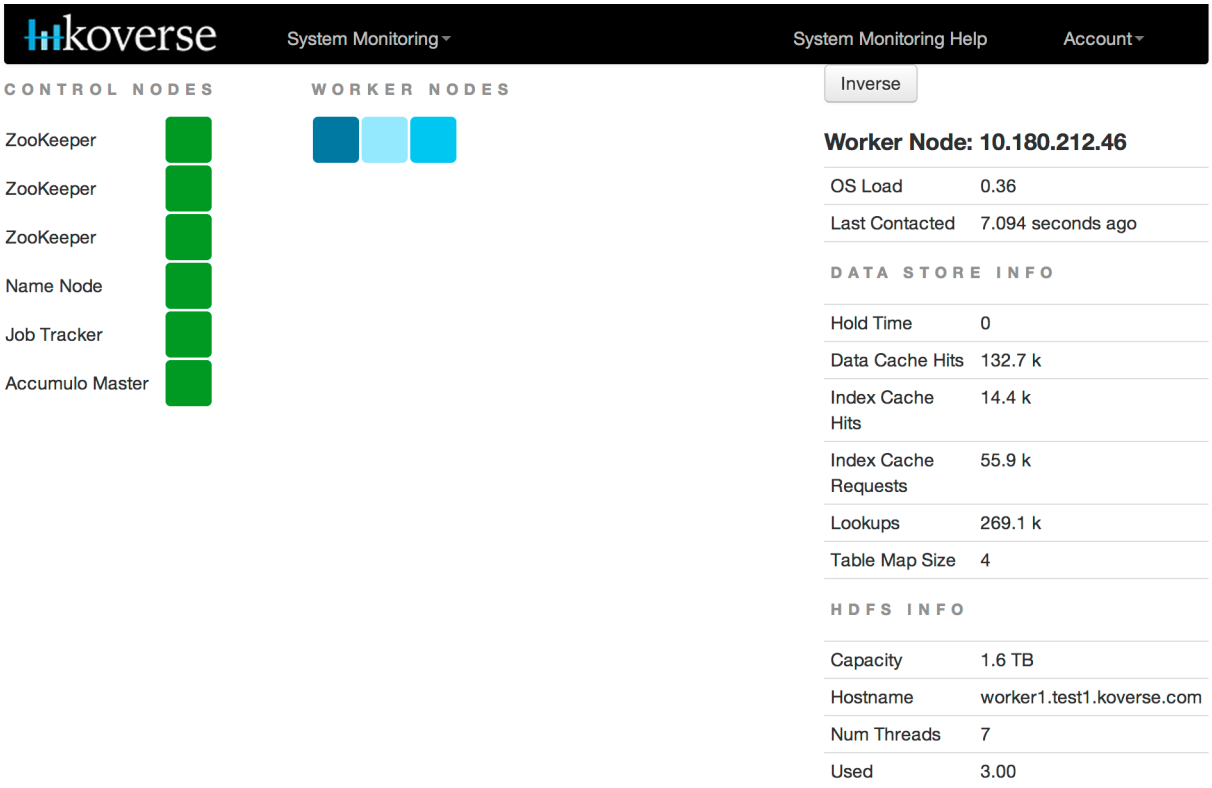


This displays a status of whether or not key processes are reachable.

- A **green** icon indicates that the process is up and reachable.
- A **red** icon indicates the the process is not reachable. In this case,

    1. Make sure the server that hosts the process is reachable over the network.
    2. Check to see that the process in question is still running on the control node.
    3. If it is verified that there are no hardware or network problems affecting the host, please see the *Troubleshooting* section, which has process-specific tips.

Worker Nodes

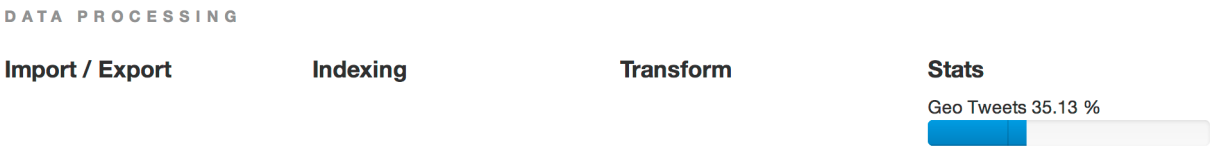


This displays the current workload of the worker nodes in the cluster. The shade of blue reflects the operating system load, with a lighter shade representing a higher load. Mousing over the individual nodes will display further details.

Ingest and Query Timelines

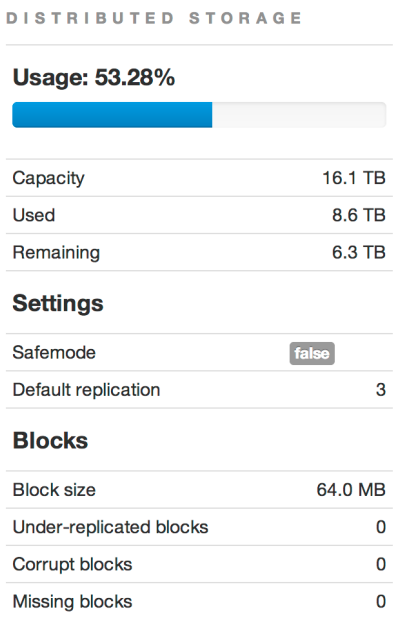
<insert screenshot of timelines> These timelines show the current ingest and query activity that the Koverse data store is handling. Large peaks are collapsed onto themselves and render as a darker shade of green in order to display high dynamic range in a small space.

Data Processing



This section shows the progress of any jobs that are currently running, organized by job type. All Koverse jobs run in the Hadoop MapReduce framework, so if you desire more detailed information about specific jobs, please see the *Checking Hadoop MapReduce Jobs* instructions.

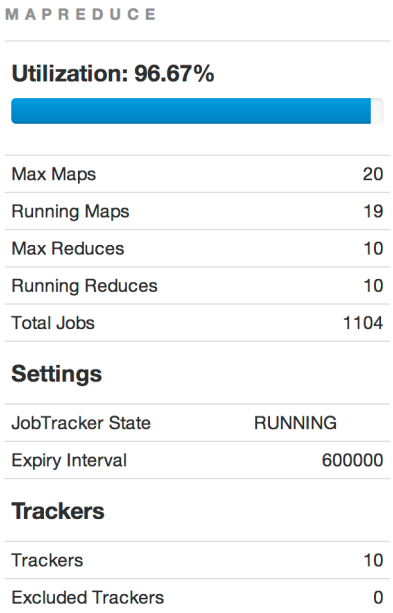
Distributed Storage



This section gives information about the state of the distributed storage system.

- This same information can be seen by *Checking the Hadoop Name Node*.
- In general, safemode should be set to "false". If it is not, see the *Hadoop Safe Mode* section of the troubleshooting guide.

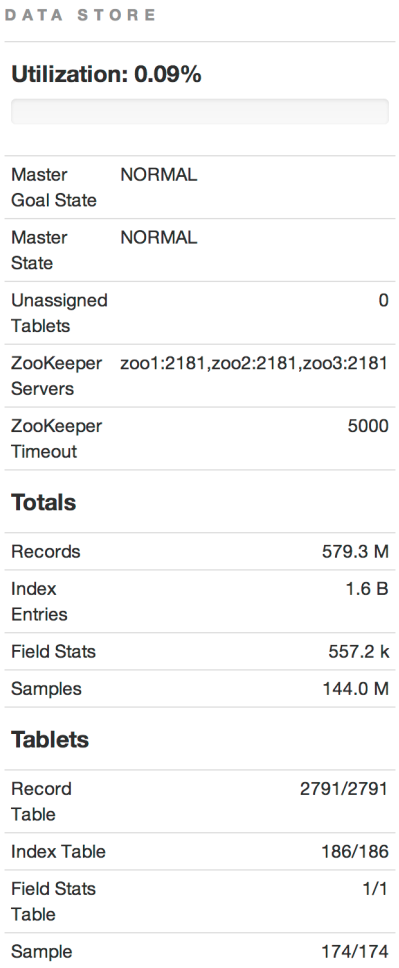
MapReduce



This section of the monitoring page shows the current state and configuration of the Hadoop MapReduce cluster.

- For more detailed information, see the *Checking Hadoop Jobtracker* and *Checking Hadoop Tasktracker* sections of the troubleshooting guide.

Data Store



This portion of the monitoring page displays information about the configuration and state of the Apache Accumulo instance that hosts Koverse's data store.

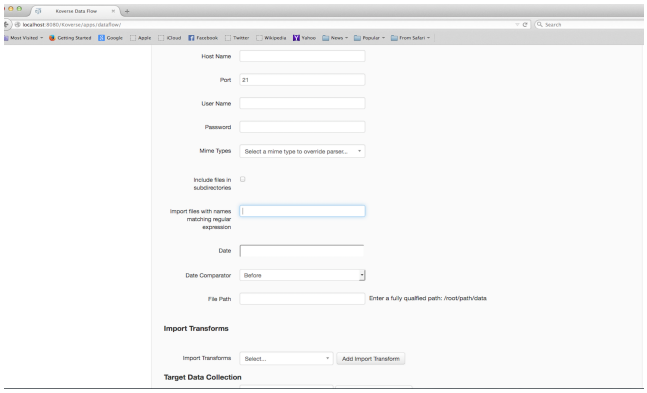
- More information can be found in the *Checking Accumulo* and *Checking Zookeeper* sections of the troubleshooting guide.

Uploading Addons

See the *Addons* section for instructions on managing Addons.

File Import Controls

Koverse now supports a new feature to import files with names matching a regular expression, filter files by date, mime-type to parser mapping, and a new recursive feature to include or exclude files in sub directories.



This screen snapshot displays the options in the *dataflow* portion of the Koverse user interface that filters import file names (please click on image to enlarge).

Details on the Mime-Type Parser Features

Koverse automatically detect the file format of each file in a file-based import, which is identified as a mime type.

Each of these files are then parsed by a specific mime-type parser. New mime-type parsers can be easily added by developers by writing a Java class that extends the `FileBasedRecordsProvider Class` (Please Note: Additional details on how to implement this feature will be documented in the Developers Guide)

It is not uncommon that there are multiple mime-type parsers that can handle a particular mime type. For example, **Plain Text** files (identified as the mime type 'text/plain') that often end in can the .txt extension may in fact contain comma-separated structured records. So rather than use the default Apache Tika parser to import the file as one record containing all the text contents of the file, users can choose to use the Separated Values Parser instead to break out the CSV lines into separate Koverse records.

Users may choose to use a specific parser for a mime-type when setting up import flow options.

Please Note: Koverse's automatic file format detection chooses the 'most reliable solution' during mime-type detection, but it is not perfect so the user must analyze the results after parsing to validate the correct mime-type was used during parsing.

For Example, sometimes a file such as CSV file is misidentified as text/plain during data ingest.

The users can use the mime-type parser User Interface Control to tell Koverse to override the default parser in these cases so the file is parsed correctly

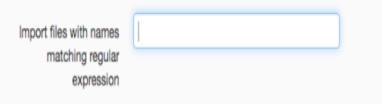
Example Usage:

Date Filtering

- If you enter a date before, or after, or equal to the Date Field; this value is used during a date comparison on the file's *Modified-Date*.

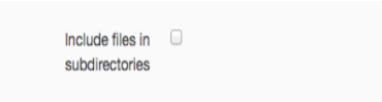
Regular Expression Matching on File Name

- If you enter any regular expression to match file names.



Recursive File Selection

- The user can select ***Include files in subdirectories*** which allows recursive navigation through the directory structure of the file system to retrieve files for ingest.
- Filtering matches are on an **OR** basis, so either date, or regular expression is used during the file matching process.
- Then that file is included in Map-Reduce import job....



Mime Type Override

- The user can select various `mime-type override` options to supersede the default parser used for a particular type of file.

Technical Workarounds

Converting Outlook .pst email messages to mbox-compatible format

Koverse currently does not support the direct import of Outlook .pst email messages but by following these instructions, a user with an Outlook .pst file can convert the email messages to .mbox format which Koverse 'does' support. Here is the process for converting the .pst email messages:

- The first step is to to take the .pst email messages file and convert it to mbox-compatible format.
- The conversion can be done by using the libpst.0.6.44 package utility. (note link below)
- The Libpst utilities includes a **readpst** command which can be used to convert .pst email messages to mailbox .mbox format.
- Run the following command to perform the conversion: **'readpst -r <outlook.pst file>'**
- The -r option changes the output format to Recursive. This will create folders as named in the PST file, and will put all emails in a file called "mbox" inside each folder. These files are then compatible with all mbox-compatible email clients.

Install libpst on Linux:

- **Download site for libpst**: http://rpm.pbone.net/index.php3/stat/4/idpl/30517395/dir/scientific_linux_6/com/libpst-0.6.44-3.el6.x86_64.rpm.html
- **Install rpm file**: rpm -i libpst-0.6.44-3.el6.x86_64.rpm
- **Run command**: readpst -r <outlook.pst file>

Install libpst on Mac OSX:

- **Run command**: ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)" < /dev/null 2> /dev/null
- **Run command**: brew install libpst

- **Run command**: readpst -r <outlook.pst file>

Man Page for readpst usage: http://linux.die.net/man/1/readpst

After the .pst messages have been converted to mbox-compatible format, the user can import the .mbox file into a Koverse collection. It appears that the 'File Import App' does not successfully process 'text/m-mailbox' properly. You must use the 'Data Flow App' and create a source with the mime type of 'text/m-mailbox'.

Glossary of Koverse Terminology

Data Collection

Data Collections are the basic container for data in Koverse. You can think of them like tables - but every record in a data collection can be completely unique in structure.

Configuration Manager

The Configuration Manager application gives users the ability to upload and download configuration for Data Collections, Sinks, Sources, and Transforms.

Data Collection

The Data Collections App gives users the ability to manage and explore Data Collections. A Data Collection is simply a named collection of records. Collections are the primary mechanism by which data is tracked and managed in Koverse.

Data Flow

Visualize, configure, and execute the movement of data within the Koverse system.

File Upload

Upload one or more files from the browser and import it into a collection.

Koverse Administration Guide

System Requirements and Dependencies

Supported Platforms

- GNU/Linux is supported as a development and production platform.

Required Software

- Jboss 7.1
- Hadoop 2.6
- Accumulo 1.6
- Zookeeper 3.4.5
- Java 1.7
- Kafka 0.8.0 (Scala 2.10)

JBoss

Version Supported: jboss-as-7.1.1.Final

It is likely that other Java servlet containers, such as Tomcat, will suffice - but they are not supported.

Hadoop

Koverse supports Cloudera's release CDH5.

http://www.cloudera.com

Koverse also supports HortonWorks' HDP.

http://www.hortonworks.com

Koverse support MapR 1.3.0.

http://www.mapr.com

Apache Hadoop is also supported.

http://hadoop.apache.com

Recommended Process Mapping

Recommendations for Process Mapping on Production Cluster

Some general suggestions:

- Keep YARN nodes and Koverse/Kafka nodes separate
- Worker nodes can be multi-tasked, e.g. a YARN worker can also be a Zookeeper node.
- On large clusters, use five node Zookeeper cluster, vice three. Seems to handle the load beter.
- Two (or more) node Kafka cluster, to allow replication/failover
- Tablet server - can be on any data node, but exclude from jobtracker and accumulo master (due to load)
- koverse-server can be memory-intensive, so make this a memory-heavy node
- Kafka - can be disk storage/IO intensive, depending on message retention settings, so make this a disk-heavy node

Here is a example suggested production cluster to service mapping:

- Koverse Node (1)
  - jboss - with koverse-webapp deployed
  - koverse-server
  - (optional) apache httpd or other reverse proxy
  - postgresql
- Kafka brokers (2+) (Note: If Kafka is used only for File Upload and not streaming imports, Kafka services could be run on shared nodes)
  - kafka-broker
- Accumulo Master (1)
  - accumulo-gc (garbage collector)
  - accumolo-master
  - accumulo-monitor
  - accumulo-tracer
  - hadoop-hdfs-secondarynamenode

- Hadoop/Spark Master (1)
    - hadoop-yarn-resourcemanager
    - hadoop-hdfs-namenode
    - spark-master
- Worker Nodes
    - accumulo-tablet-server
    - hadoop-hdfs-datanode
    - hadoop-yarn-nodemanager
    - spark-worker
- Zookeeper Nodes (3 or 5) (Note: On heavily loaded clusters, Zookeeper should be run on dedicated servers to reduce resource contention. If this is not possible, it should be given a dedicated disk on a shared server)
    - zookeeper-server

Accumulo

Accumulo 1.6 or later is supported.

http://accumulo.apache.org

Zookeeper

Zookeeper 3.4.5 is recommended - and is included in Cloudera's CDH5.

http://zookeeper.apache.org

System Requirements

Koverse can be installed on much smaller systems, particularly during, prototype, development and test and in normal operation will only take up a few Gigabytes of disk space. Koverse can be installed in a virtualized environment, but its recommended that Koverse be given high priority CPU and RAM access, for maximum processing efficiency.

Recommended Network Specifications Nodes in the Hadoop and Koverse cluster should be in a single IP subnet, with line-rate switched direct access ports. Racks of nodes should be connected via a switched 10GB connection.

Dev: * 4 core processor * 8+ GB RAM

Small: * Two Quad-core 2.15Ghz CPUs * 16 GB of RAM * Gigabit Ethernet * 2 to 8 Terabytes of non-RAID data storage * 1 disk for OS

Logical Architecture

Koverse runs on top of Accumulo and Hadoop, and requires a Postgres or H2 database for administrative state storage. Koverse connects to outside services, like FTP, Email, Websites, and Databases. Koverse also writes to similiar outside services. Koverse should be installed inside a typical network firewall.

More than one instance of Koverse can be used on a single Accumulo/Hadoop cluster. The "instancePrefix" setting in the <KOVERSE_HOME>/conf/settings.xml defines the unique instance name of the koverse server. Never change this setting after the koverse-server's first boot.

Installation and Configuration

Linux Configuration

Koverse is typically deployed on RedHat EL or Centos based systems, but Debian-based and other Linux distributions should work as well.

A 'koverse' user should be created for running the Koverse server. For example:

```
sudo useradd koverse
```

This user should also be added to the HDFS supergroup:

```
sudo usermod -a -G hadoop koverse
```

Before installing Koverse, its important to have a properly configured installation of Hadoop, Accumulo, Zookeeper, and Kafka to ensure proper operation. Please use the following URLs for more information about those packages.

http://hadoop.apache.org

http://accumulo.apache.org

http://zookeeper.apache.org

http://kafka.apache.org

Hadoop cluster information

You must know the Namenode and Jobtracker hostname and port. The Namenode and Jobtracker must report no errors.

Create a directory for Koverse that the koverse user can write to:

```
sudo -u hdfs hdfs dfs -mkdir /koverse
sudo -u hdfs hdfs dfs -chown koverse:hadoop /koverse
```

Zookeeper Servers

You must know the hostnames and server ports for the zookeeper servers. Zookeepers must all report "iamok" status, and be in a writeable state.

Accumulo location and credentials

A user account should be created in Accumulo for the Koverse application. The Koverse server will control access of individual users to its tables.

This user can be created in the accumulo shell via:

root@accumulo> createuser koverse Enter new password for 'koverse': ***** Please confirm new password for 'koverse': *****

Make a note of the username and password that Koverse will use to connect to Accumulo.

Next the koverse account will need the following permissions:

root@accumulo> grant -s System.CREATE_TABLE -u koverse root@accumulo> grant -s System.DROP_TABLE - koverse root@accumulo> grant -s System.ALTER_TABLE -u koverse root@accumulo> grant -s System.SYSTEM -u koverse

This will allow the koverse account to manage a set of tables.

Accumulo Iterators

In order to utilize the *aggregation* functions of Koverse, the koverse-aggreation-x.x.x.jar needs to be deployed to a location where Accumulo can load it. The default location would be in $ACCUMULO_HOME/lib/ext on all Accumulo tablet servers.

Installing the Koverse Server

Koverse server is distributed in following formats:

- GZipped Tar (.tar.gz)
- Zip File (.zip)
- RPM (.rpm).

To install the GZipped Tar and/or Zip file

- Copy the koverse-server archive into a directory on the target server. Unzip or untar the file into a directory on the server that will host the koverse-server service.

  Tar File Example: **tar -zxvpf koverse-server-x.x.x.tar.gz**

  Zip File Example: **unzip koverse-server-x.x.x.zip**

- Copy the "init.d" script for your platform from <KOVERSE_HOME>/scripts/... to the /etc/init.d/ directory.

  Example: **cp <KOVERSE_HOME>/scripts/centos-init.d/koverse-server**

- Add the service to the startup services.

  Example: **chkconfig koverse-server on**

Installing the Koverse Webapp

The Koverse webapp is a simple WAR file that is installed into a J2EE container. This example shows installing the Koverse webapp into a JBoss 7 installation.

Example: **unzip koverse-webapp-x.x.x.war -d <JBOSS_HOME>/standalone/deployments/**

Koverse Server Configuration Files

The Koverse Server's configuration files are available for editing in the <KOVERSE_HOME>/conf directory. Properties of interest include the JDBC connection string to the management database - which is an H2 file in <KOVERSE_HOME>/data directory.

Koverse Webapp

Koverse webapp's configuration files are available in the <JBOSS_HOME>/standalone/deployments/koverse-webapp-x.x.x.war/WEB-INF/conf directory. Properties of interest include the hostname and ports of the koverse-server. The default is localhost, assuming the koverse-server and koverse-webapp are running on the same host.

Postgres Configuration

> ***Note***
>
> If you create the koverse user with a password other then "password" you will need to execute the "Encrypting the Koverse Password" step.

CDH Environment

1. To get the current password run:

```
cat /var/lib/cloudera-scm-server-db/data/generated-password.txt
```

2. Then log into postgres as cloudera-scm:

```
psql -U cloudera-scm -h localhost -p 7432 -d postgres
```

3. To create the koverse user and use 'password' as the password:

```
postgres=# CREATE ROLE koverse LOGIN PASSWORD ‘password';
```

4. To create koverse database:

```
postgres=# CREATE DATABASE koverse OWNER koverse ENCODING ‘UTF-8';
```

Manual Installed Postgres instance

1. Change user to postgres:

```
su -u postgres
```

2. Create a new database (schema):

```
createdb koverse
```

3. Connect to the new database:

```
psql -s koverse
```

4. Create a new user:

```
create user koverse password 'password';
```

5. Give the new user permissions to modifiy the new database (schema):

```
GRANT ALL PRIVILEGES ON DATABASE koverse TO koverse;
```

6. update pg_hba.conf to set all connections METHOD to trust e.g.:

```
local  all  all  trust
```

Encrypting the Koverse Password

If you are changing the password from the default password you will need to run the koverse-squirrel utility to encrypt the password and store it in koverse-server.properties.

When Koverse runs, it uses the value in the com.koverse.license.verification property of the Koverse-Server and Koverse-Webapp property files as a symmetric key to encrypt decrypt the value of passwords (also located in those property files).

Go to your installation of Koverse server. In that directory, under the bin folder, there will be a command named**licensetool.sh**. Note that the passwords from this tool are used in both the Koverse Server and the Koverse Web-app. However, the licensetool.sh command in only found in the installation of Koverse server.

Step 1:

For the passwords used in Koverse Server, go to the conf directory of the Koverse Server installation (or fix the puppet files to do the same thing automatically).

View the file koverse-server.properties and copy the value of the field 'com.koverse.license.verification'. By default that value is **'5631524b62324648536e526152336856566a46564f513d3d'** but someone may have changed it. So, be sure to check.

To change the passwords using in the Koverse Webapp:

Navigate to the koverse.war/WEB-INF/conf directory in the file system, located in JBoss's deployment directory (e.g. the standalone/deployments folder).

The koverse-webapp.properties file you will also find the com.koverse.license.verification field, which you can use just like the above directions for Koverse server. The only difference being is that instead of changing the JDBC password for the server, you can change the thrift client password for the web app, whose value is located in the field 'com.server.webapp.thrift.client.password'.

Step 2:

Execute the Koverse License tool by changing to the Koverse server's bin directory and executing:

```
sh licensetool.sh
```

That will give you some quick feedback on how to use the program. The program has two modes of operation: create and encrypt. The create mode is used to determine the value of the 'com.koverse.license.verification' property. That is likely already done, so be cautious changing it.

To execute it, run:

```
sh licensetool.sh -m create
```

The mode you'll actually need to run is encrypt. To run that, execute:

```
sh licensetool.sh -m encrypt
```

The license tool will then ask you for the value of the com.koverse.license.verification I had you copy in the previous step. Paste it in (since it's so long and you're likely to mistype it).

Then, it will ask you for the password to encrypt, (e.g. the JDBC password or the Thrift client password). Enter the plaintext password. After that, it will work for a couple seconds and spit out the encrypted password (e.g. JewCeP3V+j5+KJulMqATQA==). Copy that password.

Step 3:

Go back to the koverse-server.properties (or koverse-webapp.properties) file on your server (or in the puppet files) and replace the encrypted password already filled into the 'com.koverse.server.jdbc.password property' (or any other password, whether in the server or the webapp) property with the one you just created.

Recommended changes to standard configurations

MapReduce

It is recommended that the "mapred-site.xml" have the following property and value added to enable an appropriate memory allocation for the task tracker processes.

mapred.java.child.opts 4096M

ZooKeeper Changes

It is recommended that the "zoo.conf" configuration file's "maxClientCnxns" property be changed to 200 - to accommodate the number of connections that are normal for a production Accumulo and Kafka installation.

maxClientCnxns=200

Secure Configuration

Access to /koverse and /accumulo directories in HDFS should be restricted to the Accumulo, Koverse Server, and Koverse WebApp processes.

Access to the Hadoop JobTracker should be restricted to administrators.

Starting Koverse Services

Koverse has two components, the Koverse Server and the Koverse Webapp.

To start the Koverse Server - use the <KOVERSE_HOME>/bin/startup.sh. Or if installed via an RPM, use the "/etc/init.d/koverse-server start" command.

When the Koverse Server startups correctly you should see this message:

```
"Koverse Server was started successfully! All services are ready and listening on ports."
```

To start the Koverse Webapp, start the J2EE container that contains the koverse-webapp...war file. For example:

```
if using JBoss, use the "/etc/init.d/jboss start" command. Or start jboss via the <JBOSS_HOME>bin/startup.sh script.
```

Koverse Default Administrator User

Koverse's default username and password are both 'admin'. You should change this on first access.

Koverse Hostname/Port

Koverse's Web UI is available via the hostname and port of the J2EE container - usually JBoss. Refer to the JBoss setup instructions.

`http://<hostname>:8080/Koverse`

Configuring Koverse's Data Store

Before Koverse can be used, the data store and related settings must be configured.

1. Access the Koverse Apps dashboard in the Koverse Web UI at `http://<koversehost>:<port>/Koverse/apps`
2. Click the "System Administration" App.
3. Click the "System" link.
4. Enter the required information.
5. Click Save.

Note: If the dialog does not close in a few seconds, check the koverse-server logs - usually at /var/log/koverse-server because there is probably a problem with the configuration.

Stopping Koverse Services

Koverse has two components - Koverse Server and Koverse Webapp - that are stopped independently.

To stop the Koverse Server - use the <KOVERSE_HOME>/bin/shutdown.sh. Or if installed via an RPM, use the "/etc/init.d/koverse-server stop" command.

To stop the Koverse Webapp, stop the J2EE container that contains the koverse-webapp...war or remove the koverse-webapp...war from the J2EE container. For example, if using JBoss, use the "/etc/init.d/jboss stop" command. Or stop jboss via the <JBOSS_HOME/bin/shutdown.sh script.

Monitoring

For monitoring of the Koverse platform, please see the *System Monitoring App* instructions.

Logging

Logging for Koverse Web Apps

In a standard installation, the logs for Koverse Web Apps, which run in JBoss, can be found in */opt/jboss/standalone/log/*.

By default, logging levels are set to "INFO". If logging levels need to be changed,

1. SSH to the jboss server(s)
2. vi /opt/jboss/standalone/deployments/Koverse.war/WEB-INF/classes/log4j.xml
3. Change the logging level. Below are examples of the 3 suggested log levels.
4. Restart JBoss Service (If you do not restart JBoss the new log level properties will not take effect.)

Logging levels may be set to one of the following:

DEFAULT:

```
<root>
        <priority value="INFO"/>
        <appender-ref ref="KoverseFile"/>
        <appender-ref ref="KoverseConsole"/>
</root>
```

WARN:

```
<root>
        <priority value="WARN"/>
        <appender-ref ref="KoverseFile"/>
        <appender-ref ref="KoverseConsole"/>
</root>
```

DEBUG:

```
<root>
        <priority value="DEBUG"/>
        <appender-ref ref="KoverseFile"/>
        <appender-ref ref="KoverseConsole"/>
</root>
```

Logging for the Koverse Server

In a standard installation, the logs for the Koverse Server can be found in */var/log/koverse-server/*.

By default, logging levels are set to "INFO". If logging levels need to be changed,

1. SSH to the Koverse Server host
2. vi /opt/koverse-server/conf/log4j.xml
3. Change the logging level. Below are examples of the 3 suggested log levels.
4. Restart Koverse Server (If you do not restart, the new log level property will not take effect.)

Logging levels may be set to one of the following:

DEFAULT:

```
<root>
    <priority value="INFO"/>
    <appender-ref ref="KoverseFile"/>
</root>
```

WARN:

```
<root>
    <priority value="WARN"/>
    <appender-ref ref="KoverseFile"/>
</root>
```

DEBUG:

```
<root>
    <priority value="DEBUG"/>
    <appender-ref ref="KoverseFile"/>
</root>
```

Backup and Recovery

Koverse relies on Hadoop Data File System (HDFS) for data storage, a relational database (either H2 or Postgres), and a set of configuration files. A production backup strategy must incorporate all three. Here are some suggestions for each.

Relational database

Use the tools that ship with the RDBMs. For postgres, use the pg_dump command. To restore, simply re-create the postgres database from the backup.

Accumulo

Use the Accumulo Export Tables feature to backup the "kv_*" tables.

Configuration Files

Copy the entire koverse-server directory - specifically the /conf directory must be included.

Automatic Support Reporting

Koverse Server has an automatic support reporting feature. This feature sends a status report to the Koverse Support Team every hour. This report can be disabled by uncommenting the documented line in /conf/settings.xml. These reports enable Kovers Support to provide better guidance for support issues.

The report includes:

1. The basic data store and jobtracker information seen in the System Monitor app.
2. The Nodes information seen in the System Monitor app.
3. The version, revision, and build date, seen in the system information details - click on the Koverse logo in the UI.
4. The IP address, date, and time from which the report was sent.

The report does not include:

1. The configuration or contents of any data collections, sources, transforms, and sinks.
2. User or groups information.
3. System level settings or environment configurations.
4. Audit logs, system level logs, or job details.

Configuring Spark to use YARN

The Koverse server can be configured to launch Spark Transform jobs using YARN. By default, Koverse is configured to use the built in Spark cluster manager.

To change to using YARN, there are prerequisites that must be completed on the machine that the Koverse server process will execute on. They are:

1. Hadoop must be installed and configured.
2. Spark must be installed and configured.

Both Hadoop and Spark must be installed and configured for the Koverse server to use YARN to execute spark jobs. The general rule is that if you can't execute a Spark job on the command line of the same machine that the Koverse server is installed due to an improper configuration of Hadoop or Spark, the Koverse server won't be able to do it either.

After configuring Hadoop and Spark, the following properties in the koverse-server.properties file must be examined and changed if necessary:

1. com.koverse.server.hadoop.conf.dir, which has the default value of "/etc/hadoop/conf" already set.
2. com.koverse.server.spark.dir, which has the default value of "/opt/spark" already set.

The final property in koverse-server.propeties file to set is "com.koverse.server.spark.mode" to "yarn" (instead of "master").

Troubleshooting

If ever you need assistance please submit questions to support@koverse.com. Please attach logs and steps to reproduce the general issue you are encountering.

Below are some troubleshooting tips to address specific issues,however, most of this section contains fairly advanced operations, so please do not hesitate to reach out to Koverse support.

Checking the Logs

In order to debug issues, it is often helpful to look in the logs of the Koverse Thrift Server, and/or JBoss logs. See the *Logging* section for instructions on how to do so.

Checking JBoss

On the JBoss server(s) in your Koverse cluster, the */etc/init.d/jboss* script can be used to start, stop, and check the status of JBoss.

Checking the Koverse Server

On the Koverse Server node in your Koverse cluster, the */etc/init.d/koverse-server* script can be used to start, stop, and check the status of the Koverse Server.

If you see evidence that applications are having problems connecting to the Koverse Server,

1. Check that Koverse Server is running and start it using the above script if it is not running.
2. Check the Server logs (see *Logging* for how to do so). If there are obvious problems being reported in the logs, try restarting the Server, or contact support@koverse.com.
3. If the Server appears to be running fine, check to make sure that the Thrift ports are open:

    1. On the box that hosts Koverse Server, perform the command 'telnet localhost 12320'.

        - If your connection is refused, the Thrift ports are not open. This generally means the system manager failed to start. Try restarting the Koverse Server.

Checking Hadoop MapReduce Jobs

To obtain detailed information about Hadoop MapReduce jobs, use the Hadoop jobtracker page.

This can be found at `http://<yourjobtrackerhost>:50030/jobtracker.jsp`

Checking Hadoop Jobtracker

In a CDH4 installation of Hadoop, one can start, stop, or check the status of the Jobtracker process from the command line using the script found in */etc/init.d/hadoop-0.20-mapreduce-jobtracker*. Other Hadoop installations have a similar executable.

Checking Hadoop Tasktracker

Status of the Hadoop Tasktracker can typically be found at `http://<yourtasktrackerhost>:50060/tasktracker.jsp` In a CDH4 installation of Hadoop, one can start, stop, or check the status of the Tasktracker process from the command line of the individual Tasktracker servers using the script found in */etc/init.d/hadoop-0.20-mapreduce-tasktracker*. Other Hadoop installations have a similar executable.

Checking the Hadoop Name Node

Status of the Hadoop Namenode can typically be found at `http://<yournamenodehost>:50070/dfshealth.jsp` In a CDH4 installation of Hadoop, one can perform operations such as start, stop, etc. on the Namenode process from the command line using the script found in */etc/init.d/hadoop-hdfs-namenode*. Other Hadoop installations have a similar executable.

Checking Accumulo

Status of the Accumulo Master and Tablet Servers can typically be found at `http://<yourmasterhost>:50095/` In a CDH4 installation of Hadoop, one can start, stop, or check the status of the Accumulo Master and Table Server processes from the command line using the scripts found in */etc/init.d/accumulo-master* and */etc/init.d/accumulo-tablest-server*, repectively. Other Hadoop installations have similar executables.

Checking Zookeeper

In a CDH4 installation of Hadoop, one can start, stop, or check the status of individual Zookeeper nodes from the command line using the script found in */etc/init.d/zookeeper-server*. Other Hadoop installations have a similar executable.

Failing Transforms

When you experience problems running transforms do the following:

- Look at your Job Tracker `http://<yourjobtrackerhost>:50030/`
- Go to the Hadoop Job for the Transform that failed, and identify any failed map or reduce tasks.
- Click on those failed tasks, there should be options to examine their individual runtime logs. Each log should be very short and contain a full exception stack trace.

Submit a support ticket with attached log if you need assistance from the support team.

Hadoop Safe Mode

If you ever run low on disk space Hadoop will automatically enter Safemode. In order to leave safe mode:

1. *hadoop dfsadmin -safemode leave*
2. Restart all services.

Persistent Login Screen

If users are unable to get past the login screen and there is not a warning that username/password are incorrect, this is an indication that the Koverse Web UI is not able to reach the Koverse Server. In this case, please follow the troubleshooting steps in *Checking the Koverse Server* to try to resolve the problem.

Waiting for Changelock

If the koverse-server shows a "Waiting for Changelog lock"... message on startup, the previous run of liquibase was killed during execution and left a DATABSECHANGELOGLOCK table that is keeping liquibase from executing.

To remove the DBCHANGELOGLOCK table, which will allow liquibase to run, do the following

1. cd <KOVERSE_SERVER_HOME>
2. java -cp lib/h2*.jar org.h2.tools.Shell
3. jdbc:h2:/tmp/koverse
4. DROP TABLE DATABSECHANGELOGLOCK;
5. exit
6. /etc/init.d/koverse-server start

High Availability Namenode and Jobtracker

Koverse can be configured to utilize high availability Namenodes and Jobtrackers by providing a `settings.xml` file that provides the appropriate Hadoop configuration values. The following sample values outline the configuration parameters used when running an HA namenode with two namenodes as well as an HA jobtracker with two job trackers:

```
<!-- HA namenode properties -->
<entry key="hadoopJobSetting.fs.defaultFS">
            hdfs://nameservice
    </entry>
<entry key="hadoopJobSetting.dfs.nameservices">
            myNameservice
    </entry>
<entry key="hadoopJobSetting.dfs.ha.namenodes.myNameservice">
            namenodeA,namenodeB
    </entry>
<entry key="hadoopJobSetting.dfs.namenode.rpc-address.myNameservice.namenodeA">
            namenodeA.address:port
    </entry>
```

```
<entry key="hadoopJobSetting.dfs.namenode.rpc-address.myNameservice.namenodeB">
            namenodeB.address:port
    </entry>
<entry key="hadoopJobSetting.dfs.client.failover.proxy.provider.myNameservice">
            org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
    </entry>

<!-- HA jobtracker properties -->
<entry key="hadoopJobSetting.mapred.job.tracker">
            myJobtracker
    </entry>
<entry key="hadoopJobSetting.mapred.jobtrackers.myJobtracker">
            jobtracker1,jobtracker2
    </entry>
<entry key="hadoopJobSetting.mapred.jobtracker.rpc-address.myJobtracker.jobtracker1">
            jobtracker1.address:rpc-port
    </entry>
<entry key="hadoopJobSetting.mapred.ha.jobtracker.rpc-address.myJobtracker.jobtracker1">
            jobtracker1.address:ha-rpc-port
    </entry>
<entry key="hadoopJobSetting.mapred.jobtracker.rpc-address.myJobtracker.jobtracker2">
            jobtracker2.address:rpc-port
    </entry>
<entry key="hadoopJobSetting.mapred.ha.jobtracker.rpc-address.myJobtracker.jobtracker2">
            jobtracker2.address:ha-rpc-port
    </entry>
<entry key="hadoopJobSetting.mapred.client.failover.proxy.provider.<%= jobtracker %>">
            org.apache.hadoop.mapred.ConfiguredFailoverProxyProvider
    </entry>
```

Restricting users from using Koverse

Koverse has the ability to be configured so that users' groups' may or may not be given access to use Koverse. This mode of operation is disabled by default. If enabled, then only users who are member of groups with the "useKoverse" permission will be able to use Koverse. Note that the users' groups' may be external, meaning that external systems are configured to allow access to Koverse.

Since this mode of operation is disabled by default, to enable it, configure koverse-server.properties so that the configuration:

com.koverse.server.auth.useKoversePermission.required=false

is set to true.

Note that by default the built in admin user has this permission, so logging in as admin should always work (unless the account is modified to remove it). However, if existing users are not members of groups that have this permission, making this change will lock them out.

Users who attempt to access Koverse but do not have the permission to do so are redirected to a static HTML page notifying them that they are forbidden from using Koverse.

Care should be taken with this configuration to ensure that external users and groups already contain the "useKoverse" permission, as appropriate for your requirements. This is typically done by adding in JSON files in to the server's conf/load-once directory, which specify such groups and their permissions.

Database cleanup

The Koverse configuration database can grow large with jobs running continuously and it is suggested that the jobs table are purged using a setting in koverse-server.properties:

com.koverse.server.purgeJobsDate=30d

By default this is set to 30 days and will delete statuses and history of jobs older than 30 days. The setting uses the format d(days), h(hours), and m(minutes) i.e. 1d, 1h, 1m. The purge job service runs every 10 minutes and will look for jobs to cleanup older than the duration entered.

Koverse Operations Guide

Architecture

Koverse sits on top of a complex set of interworking and distributed services. These include:

- Hadoop Distributed File System (HDFS)
- Hadoop MapReduce
- Spark
- ZooKeeper
- Accumulo
- Kafka
- Postgres

Total System Startup

There is an order to which the underlying systems should be brought online. When systems do not depend on each other they can be started at the same time.

1. **Data Storage and Coordination Layer - these can be started first after system boot.**
    - HDFS Data nodes
    - HDFS Namenode
    - ZooKeeper
    - Postgres
2. **Data Services Layer - all of these depend on one or more processes in the Storage and Coordination Layer.**
    - Hadoop MapReduce JobTracker
    - Hadoop MapReduce TaskTrackers
    - Spark
    - Kafka
    - Accumulo Tablet Servers
    - Accumulo Master
3. **Application Layer - all of these depend on one or more process in the Data Services Layer**
    - Accumulo Monitor
    - Accumulo Garbage Collector
    - Koverse Server
    - Koverse Web App

Total System Shutdown

Processes should be stopped in reverse of the startup layer order.

1. **Application Layer**

- Koverse Server
- Koverse Web App
- Accumulo Monitor
- Accumulo Garbage Collector

2. **Data Services Layer**

- Hadoop MapReduce JobTracker
- Hadoop MapReduce TaskTrackers
- Spark Executors
- Kafka
- Accumulo Tablet Servers
- Accumulo Master

3. **Data Storage and Coordination Layer**

- HDFS Data nodes
- HDFS Namenode
- ZooKeeper
- Postgres

If a process in say, the Data Storage and Coordination Layer, is stopped before all processes in the Data Services and Application Layers, system state may become unstable or corrupt. All processes in one layer should be stopped before stopping any processes in the next layer.

Sometimes a single worker process in a lower layer can be stopped and restarted without stopping higher layers. See "Fixing a simple, single-server failure".

System Recovery

Automatic Recovery Scenarios

Hadoop, Accumulo, Kafka, and ZooKeeper are distributed systems designed to recover automatically from single-server failure, often without administrator intervention. The following things can fail and the system should keep running indefinitely without admin intervention.

Single tablet server process - master will reassign its tablets to other servers and perform recovery of any data in memory. Clients will automatically detect the failed tablet server and find the tablets of interest on newly assigned servers.

Single data node - the Namenode will redirect remaining data nodes to create new replicas of the blocks on the failed machine. Clients will use replicas on remaining machines.

Single ZooKeeper node - remaining ZooKeeper nodes will handle load, optionally electing a new leader Clients will find the new leader automatically.

Accumulo monitor - the web UI will be unavailable but clients can continue to communicate with tablet servers to read and write data.

Accumulo gc - no garbage collection will be performed, but clients will continue to communicate with tablet servers.

Loss of one rack of servers - as long as HDFS rack-awareness is enabled and the servers consist only of worker processes, tablet servers, data nodes, kafka brokers

Fixing a simple, single worker failure

Most of the time, if a worker process stopped for a non-permanent reason (e.g. not out of disk space) it can simply be started again. In the case of permanent hardware failure, the server can simply be permanently left out of the cluster. Remaining servers will take over the failed machines workload, as long as remaining resources allow.

A new process can be started on a new machine for processes that do coordination, such as the master, gc, monitor etc.

Single Zombie Processes

Sometimes a process is still running but not responding to requests. Checking the logs of these processes can reveal problems such as running out of file handles to start new threads, or sockets to handle new requests. Sometimes servers just have a high workload queued up (such as lots of compactions scheduled) and will become responsive again after working down the queue. In some cases queued work, such as compactions, can be canceled and scheduled at a more opportune time.

If a server is inexplicably unresponsive despite still running, it can be stopped to cause its responsibilities to be taken over by another machine. When stopping a zombie process, time should be provided after stopping the process before stopping any other processes or trying to restart the process, in to allow other processes to absorb the workload transferred and optionally perform recovery.

Recoverable Failures Requiring Intervention

The following scenarios will result in data being unavailable until an administrator can intervene:

- More than 2 data node process failures in a short time resulting in all replicas for a block to be missing.
- Accumulo tablet server failure while no master is running - some tablets will be unassigned.
- More than one ZooKeeper server down - may result in a failure to form a quorum and accept writes.

When attempting to recover from a system failure involving more than one server, the following rules should be followed:

Lower layers should be online and healthy before attempting to fix higher layers.

Because system shutdown involves attempting to persist data to disk, starting some stopped processes is often required before shutdown can happen safely. This will allow the system to become healthy before shutting down.

If Accumulo is still running but some tablets are offline and can't be brought online, it may be that not all data nodes are healthy.

Failures Resulting in Potential Data Loss, or other Unrecoverable States

Loss of more than 2 hard drives at once - any data replicas living on those 3 or more hard drives will be lost.

While systems that depend on them are running:

- Unavailability of all ZooKeepers at once
- Unavailability of all tablet servers at once
- Unavailability of the Namenode (Single point of failure if not using HA Namenode)
- Loss of the Postgres DB (Single point of failure)

Koverse Developer Documentation

Introduction

Organizations are likely to require some custom software development to address their own unique set of data analytics requirements. These custom software features will require their own unique sources and methods that provide strategic insight and competitive advantage.

Each organization could have potentially some combination of unique data sets, mission-specific data processing requirement used to analyze and transform those data sets, and custom interactive user interfaces.

With this in mind, Koverse provides this developer documentation describing how to programmatically extend Koverse with additional functionality, and how to deploy that functionality into an operational Koverse instance.

References

Glossary of Koverse Terminology

Customizations Types

Developers can customize and extend Koverse in several ways, such as:

- Koverse Apps - Web Applications that are hosted by Koverse leverage the Javascript SDK to support interaction with Koverse for a large number of users. Apps may also include custom Transforms to help get Data Collections into a structure that the App expects.
- AddOns - these are packages that extend Koverse with custom Sources, Transforms, and Sinks.
- Koverse Clients - these are processes that interact with Koverse via an API and that can be embedded in other services etc.

Koverse Core Concepts

The following sections provide a basic introduction to the basic abstract concepts which build a foundation of knowledge for a developer before working with Koverse API.

Data Model

The Koverse data model has two main conceptual components: **Records**, and **Data Collections**. Logically, each Data Collection contains a set of Records.

For those familiar with relational database management systems such as Oracle or MySQL, the analogy is that a Data Collection is similar to a Table, a Record is similar to a Row, and the fields of a Record are similar to the Columns. However, unlike traditional relational databases, Records in a single Collection in Koverse do not have to all have the same fields, and fields can contain complex values, like lists and mappings of fields to values.

Records

The Koverse canonical unit of data is a Record. A Record is a map of keys/values, or fields, similar to a JSON document. Like a JSON document, a Record can have embedded lists or nested maps.

A Record belongs to a single Data Collection. Different Records within the same Data Collection do not have to have the same fields or structure. The values in a Record can be of many different types, including Strings, Doubles, geospatial points, and Dates. A Record also has an optional security label which can be used to provide Record-level access control.

Some key points to remember about Records are:

- Each record is present in one and only one Data Collection.
- **Records are maps of key/value pairs, similar to JSON**
    - Example: {key1: valueA, key2: valueB}

- **Value types may vary across records with matching keys**
    - Example Record A: { key1: stringValue}
    - Example Record B: { key1: 234 }
- Records do not have a user designated id field. It is up to the application to designate and populate an identifier field. The application can submit queries to look up records by any field, including a field to which it has assigned unique identifiers.
- The optional security label on a record is set programmatically through the Java API and effects how the record is stored and retrieved.
- Records can contain nested value objects: * Example: { name: parent, children: [{ name: child1}] }
- Records can contain the following native value types:

Native Value Type	Example
String	A string of text characters
Integer	15
Double	200.05
Date Time Strings - Unix, Epoch, DTG	Unix Timestamp: 1371277293 UTC (GMT) Epoch Timestamp: 1371277293 DTG: 271545ZFEB13 Other various date formats supported: • yyyyMMdd hh:mm:ss • EEE MMM d HH:mm:ss Z yyyy • EEE MMM d HH:mm:ss zzz yyyy • yyyy-MM-dd • yyyy/MM/dd • yyyy-MM • yyyy/MM/dd HH:mm:ss • yyyy-MM-dd HH:mm:ss • yyyy/MM/dd HH:mm:ss.SSS • yyyy-MM-dd HH:mm:ss.SSS • MM/dd/yyyy HH:mm • MM-dd-yyyy HH:mm • ddHHmm'Z' MMM yy
Geospatial Points	Well Known Text String Format: Point 1.23 60.423 Comma separated decimal lat,long: 1.23,60.423

Data Collection

Data Collections are the basic container for data in Koverse. You can think of them like tables - but every record in a data collection can be completely unique in structure.

A Koverse Data Collection is a named set of Records. A Data Collection has:

- Configurable indexes to enable queries to quickly and efficiently find Records.
- Permissions to control access to Records in the Data Collection.
- Automatically discovered statistics and samples to provide insight into the Records contained in the Data Collection.

Data Sources

A data source is simply the source of the data. It can be a file, a particular database on a DBMS, or even a live data feed. The data might be located on the same computer as the Koverse application, or on another computer somewhere on a network.

Koverse establishes the connection to these data sources and provides the ability to import data in Koverse, breaking the data into records according to the external format of the data (i.e. JSON, XML, CSV, relational records, etc).

Custom sources are only necessary when talking to a new type of server, often using a new protocol. For example, Koverse ships with an FTP source, and and IMAP source. New sources are not necessary simply for new file types and certainly not for specific uses of known physical formats such as a particular type of XML file.

Transforms

In Koverse, transforms are a process by which one or more data collections leverage re-usable, configurable, multi-stage MapReduce jobs for data manipulation. These are highly scalable and customizable analytics that are reusable across all of your data.

Built-In Example Transforms

The following list is an example of built-in Koverse transforms:

- Close Graph
- Extract Time Series
- Faceting
- Entity Extraction (using Apache OpenNLP)
- Geo Discovery
- Geo Location (IP Addresses, Airports, Postal Codes (Canada))
- Summarize Relationships
- Sentiment Analysis
- Copy Records
- Nearest Neighbors
- Text Cleanup
- Summarize Field Values

Import-time Transforms

Import-time Transforms are one-stage transforms that operate like a single map() phase and are applied to Records as they are imported from a Source. Import-time Transforms can be chained together during a particular Import job.

Export-time Transforms

Export-time Transforms are one-stage transforms that operate like a single map() phase and are applied to Records as they are exported to a Sink. Export-time Transforms can be chained together during a particular Export job.

Export File Formats

Export File Formats define how Records are written to file-based Sinks such as FTP and HDFS Sinks.

Sinks

Sinks represent external destinations to which Records from Data Collections may be sent. For example, one can write out Records as JSON objects to a remote file system.

Queries

Whether developing a Koverse App or building a custom source, Koverse queries conform to a specific format. There are two types of syntax supported: a Lucene-like syntax and a more Object-based structure.

Lucene-like Query Syntax

These queries are represented as strings and passed as such into query methods. The Lucene query syntax is described in the Usage Guide at *Query Syntax*.

Object-based Queries

Search Criteria	Query Syntax
Searching 'any' field for a value	{$any: fmv}
Search specific field for a value	{field.name: fmv}
Search AND	{$and: [{$any: fmv}, {$any: blue}]}
Search OR	{$or: [{$any: fmv}, {$any: blue}]}

Range Queries

Search Criteria	Query Syntax
Any value greater than or equal to 160	{$any: {$gte:160}}
Date field less than a specific date	{date_created: {$lt: "1980-01-01T00:00:00.000Z}}
Geo Range	{fieldName: {$box: [[sw-lat, sw-long],[ne-lat, ne-long]]}} {fieldName: {$box :[[39.5, -104.9],[40, -104.5]]}}

Note that queries that combine a range with any other criteria, and queries that combine multiple ranges require Composite Indexes on the fields involved. See _CompositeIndexes for information on building these.

Aggregations

Aggregations allow you to easily maintain near real-time statistics on the Records in a Data Collection. Aggregations run incrementally on new Records to maintain pre-computed, up-to-date results so that they can always be queried with sub-second latency.

Quick Start Java Project

Koverse ships with a koverse-sdk-project-<version>.zip file that contains an example Maven based Java project. This project defines some simple custom sources, sinks, transforms, and apps. The maven pom.xml file in this project builds an *Addon* that can be uploaded. Simply alter the Java and HTML/JS code in this project, then build and deploy the addon to Koverse.

GitHub Koverse SDK Project

Visit Koverse SDK Project to fork or download the latest koverse-sdk-project for your version of Koverse.

Koverse SDK Project Maven Archetype

A Maven Archetype project is available for easy deployment. Modify the version number (KOVERSE-VERSION-HERE) in the command below to configure and create a new instance of a Koverse project:

```
mvn archetype:generate  \
  -DarchetypeRepository=http://nexus.koverse.com/nexus/content/groups/public/  \
  -DarchetypeGroupId=com.koverse.sdk.project \
  -DarchetypeArtifactId=koverse-sdk-project-archetype \
  -DarchetypeVersion=KOVERSE-VERSION-HERE \
  -DkoverseVersion=KOVERSE-VERSION-HERE
```

Building the Koverse SDK Project

The koverse-sdk-project is a standard *Apache Maven <https://maven.apache.org>* file that produces a shaded JAR - which means that it collapses all of its runtime dependencies into a single JAR file. This is necessary for running jobs in Koverse.

Use the following command from the root directory of the unzipped koverse-sdk-project:

```
mvn clean package
```

After a successful build, the resulting Addon JAR file is in the koverse-sdk-project/target/ directory. By default it is named koverse-sdk-project-<version>.jar

Modifying the Koverse SDK Project

You should modify the koverse-sdk-project to fit your needs. Here are some good starting points.

1. Change the <artifactGroup> and <artifactId> values in the pom.xml file to match your organization and project.
2. Change the Java package name from com.koverse.foo to your organization and project names.
3. Change the <artifactGroup> and <artifactId> values in the pom.xml file to match your organization and project.
4. Change the Java package name from com.koverse.foo to your organization and project names.
5. Modify the Java classes to create your own custom sources, transforms, sinks, and application definitions.
6. Delete any unused Java classes.
7. Modify the /src/main/resources/classesToInspect.example file to match your Java classes and rename the file to classesToInspect.
8. Modify the /src/main/resources/apps/ contents for your custom application.
9. Modify the LICENSE and README file

Deploying the Addon to a Koverse Server

Addons can be deployed via a Maven command, or via the Koverse web interface.

Maven Addon Deployment

1. Login to your Koverse server
2. Navigate to the "System Administration" application
3. Click the "API" tab
4. Click "Add API Token" button
5. Add a name such as "developer"
6. Click "Administrators" button
7. Click "Create Token" button
8. Note the API Token that was created.
9. Add the following settings to your ~/.m2/settings.xml profile:

```
<properties>
        <koverse.apitoken>API-TOKEN-HERE</koverse.apitoken>
        <koverse.serverurl>KOVERSE-URL-HERE (ex: http://koversevm/Koverse)</koverse.serverurl>
</properties>
```

10. Use this single command to build and deploy the plugin for testing:

```
mvn clean package org.apache.maven.plugins:koverse-maven-plugin:deploy
```

Web interface Addon Deployment

1. Navigate to the "System Administration App"
2. Click the "Addons" tab
3. Click "Browse" or "Choose File", and select the addon file from the <basedir>/target for your maven project.
4. Click upload

Addons

Any custom code, whether it be one or more applications, transforms, or custom sources or sinks, can be packaged up into a simple JAR - referred to in Koverse as an Addon. Addons are uploaded to Koverse, via the System Administration app, for deployment.

Koverse reads the contents of the JAR file and extracts necessary metadata from any classes extending Koverse known types, such as Application, Transform, Source, and Sink.

Creating an Addon

Addons are simply JAR files with some specific files and a well formed directory structure. The koverse-sdk-project provides a complete example maven project that builds an appropriately constructed Addon JAR. You may use any assembly framework you like to produce a JAR file with the following attributes

- Java binary .class files in the normal Java package directory structure.
- Koverse Application HTML and JavaScript should be placed in the /apps/<applicationId> folder - where applicationId matches the string your CustomApplication.getApplicationId() method returns.

- A file named classesToInspect can optionally be placed at the root level of the JAR. This file is a line separated list of all Applications, Transforms, Sources, and Sink Classes. Including this file causes Koverse to inspect only the classes listed in this file. This is useful when your Addon includes classes whose dependencies are not present in the JAR.

Example Addon JAR directory structure:

```
MyCustomAddon.jar
|
| -- classesToInspect
| -- com
          | -- mycompany
               | -- myproject
                      | -- MyCustomTransform.class
                      | -- MyCustomApplication.cass
|-- some
        | -- other
               | -- dependency
                      | -- OtherDependency.class


| -- apps
        | -- myApplicationId
               | -- index.html
               | -- css
                      |-- index.css
               | -- javascript
                      |-- index.js
        | -- mySecondApplicationId
               | -- index.html
               | -- someFolder
                      | -- some.file
```

Uploading an Addon to Koverse

See the *Addons* section.

Applications may be auto deployed, and immediately ready for use - if so defined by the developer of the application. Sources, Transforms, and Sinks are also now ready for immediate use as well.

Managing Versions for Custom Components

The applicationId, sourceTypeId, transformTypeId, and sinkTypeId property of the Applications, Sources, Transforms, and Sinks, are used by Koverse to identify these custom components across their versions. This means that, except in extreme cases, all versions of a custom component should share a single typeId string. This allows Koverse to identify when a newly installed custom component should override an exisisting custom component.

Here is an example life cycle of a single Addon containing a single custom source.

1. An administrator or developer user uploads a MyCustomAddon-1.0.0.jar Addon into a Koverse installation. This JAR contains a MyCustomSource with a sourceTypeId of myCustomSource.
2. The source is used by many other end users over time to import data from various systems.
3. A developer releases a new updated version of the Source. This source is now named My New Custom Source, has a sourceTypeId of myCustomSource, and is in an Addon named MyNewCustomAddon-2.0.0.jar.
4. An administrator or developer uploads this new Addon JAR file.
5. Koverse inspects the MyNewCustomAddon at install time, and discovers that the MyNewCustomSource has the same sourceTypeId as the existing MyCustomSource.
6. Koverse automatically disables the old MyCustomSource. All instances of this source now execute the MyNewCustomSource code. This means end users may need to consider the changes in parameters or behavior.

7. When all of the components of an Addon have been disabled, either manually or via uploading of new overlapping components, the old addon itself is disabled - and is therefore removed from the administration interface. In this case, MyCustomAddon-1.0.0.jar is disabled.
8. Koverse does not discard the logging or reference to the old Addon. These items remain for auditing and provenance purposes.

The Version Property

The version properties of these custom components are simply used to identify the active installed version for troubleshooting and verification purposes. Koverse uses a last installed methodology when selecting the implementation version for custom Application, Source, Transforms, and Sinks. This means that the end user can simply upload any version of an Addon, and be assured they are using the last installed. The version string itself has no affect on which version is executed.

Change Control Across Versions

Developers should consider that customers upgrading from one version to the next, or down grading, may have already established Source, Transform, or Sink instances that have existing parameter values. This means the developer may need to handle outdated parameter configurations. The most appropriate method to handle changing parameter sets across versions is to inform the user that new configuration is needed, when simple error checking of parameters fails.

HTML/JS Apps

The ability to quickly build new applications to address specific mission and business needs is a primary objective of Koverse. Applications built on Koverse can take advantage of the powerful data management, indexing, and query capabilities Koverse provides. In conjunction with custom Transforms, Koverse applications can achieve a large breadth of functionality.

Apps are built using HTML and Javascript and interact with Koverse via the Javascript SDK found in the koverse.js file.

Koverse Javascript SDK

Koverse ships with a few .js files that should be included in your custom apps. Open and inspect them each for a list of their properties. The list below describes these files in detail.

/Koverse/js/koverse.js

The koverse.js file contains all of the AJAX functions for calling Koverse REST API methods. Use these methods to manipulate services, perform CRUD operations on components, and query for data in koverse. This file requires that JQuery is also included in your project.

/Koverse/js/apps/apps-common.js

The apps-common.js file contains all of the logic for the Koverse common look-and-feel - including the top navigation bar and menu. Use this file in your native app so that it is well integrated with the koverse JS UI framework. This file requires the /Koverse/css/apps-common.css style sheet.

/Koverse/js/koverse-util.js

The koverse-util.js file contains many helper functions for common uses cases in Koverse apps - like number formatting, date parsing, and URL hash/anchor value manipulation, etc. Including this file in your app is optional, but you will likely find it very helpful.

Defining Custom Apps in Addons

Addons enable developers to deliver custom "Apps" that are managed and deployed in Koverse installations. When a system administrator uploads an Addon JAR file, it is inspected for custom Application definitions. The custom application contents are included included in the JAR, so that it's contents can then be delivered to the end user.

Application Definition

See the koverse-sdk-project/src/main/com/koverse/foo/MyCustomApplication.java file for an example of defining a custom application. That file defines the presence of a custom application type.

HTML/JS code in Addons

See the *Creating an Addon* section for the structure of an HTML/JS app in side an addon. The top directory name of the app's html/js code should match the output of getApplication() method.

Sources API

Koverse Sources are designed to read data from a specific type of data source, such as a relational database or a remote file system.

Koverse uses MapReduce to import from sources when there are multiple items to be read and when processing those items on multiple machines will speed up the overall import. Examples include having many map workers read and process one file each from a directory on a remote file system.

Other sources are read from a single thread running on the same server on which Koverse is installed. These are referred to as inline sources.

Once a connection to a source is established, the next job of the source is to create a RecordStream that produces a set of Java objects, representing raw records or files obtained from the source.

One example is a CSV file. When a source is processing a CSV file, it simply breaks the file into distinct lines by tokenizing on the newline character, and provides those as Java Strings to the next phase.

Finally, Sources employ RecordFactory classes to convert Java Objects into Koverse Records. Often RecordFactories can be shared across sources, such as a factory used to convert lines from a CSV file to a Koverse Record. There are many types of sources that may provide CSV files: NFS, local file systems, remote HDFS instances.

To use the RecordFactory classes as well as others, be sure to inlcude the following dependency in your pom.xml:

```
<dependency>
        <groupId>com.koverse.addon</groupId>
        <artifactId>koverse-addon-file-source-deps</artifactId>
        <version>${project.parent.version}</version>
</dependency>
```

Sources are configured through defining parameters that are presented to users via the User Interface. This way the source can obtain necessary information such as the hostname and port of the server containing the source data, or a username and password.

See the *Quick Start SDK Project* section for details about a ready made project for creating custom sources.

Source Types

SimpleSource.java

The SimpleSource class should be extended when users would like the ability to import one or more records or files from a single external server. The see Koverse SDK Project contains an example MyCustomSource that extends SimpleSource.

The methods to implement are the following:

```
/* Using the end-user supplied parameter values, establish a connection to the outside source */
public abstract void connect();

/* Retrieve the next record from the outside source. Return null when no more records exist */
public abstract Map<String, Object> getNext();


// Cleanup the connection to the outside source
public abstract void disconnect();
```

```
/* Return a list of Parameter objects that describe the
 * end-user supplied parameters necessary for establishing
 * a connection and producing records from this source. */
public abstract List<Parameter> listParameters();
```

Transforms API

Koverse Transforms can operate over one or more data collections to perform advanced algorithmic processing or create analytic summaries. Koverse tracks all transform relationships between input and output Data Collections so the provenance of any given Data Collection is traceable to its derivative Data Collections or Import Sources.

Koverse uses Apache Hadoop MapReduce to execute Transforms over data collections and handles all the details of scheduling, running, stopping, and monitoring the individual Hadoop jobs. To transform a data set, users implement a simplified MapReduce API that allows for reading records from one or more input Data Collections, potentially filtered according to user authorizations and writes output to a new Data Collection, applying security labels appropriately.

Using the Koverse Transform API his has several advantages over using Hadoop directly:

- Developers can focus on the details of their algorithm, rather than worrying about the details of handling many different input and output data formats and managing multiple jobs.
- Transforms are parameterized so that, once a Transform is written, it can be configured and run by non-developers on the Data Collections they are authorized to read.
- The details of how a transformed result data set is stored and labeled are handled by the Transform framework. This ensures that result sets will be automatically queryable and that access control policies are maintained.

Koverse transforms build on the MapReduce processing functions map(), combine(), and reduce().

See the Koverse SDK Project section for details about a ready made project for creating custom transforms.

Transform Stages

Many algorithms involve more than just one map() and one reduce() function. Koverse Transforms are organized into stages which are run one after the other, the output of the previous stage becoming the input to the following stage.

Transforms are specified by the developer defining the individual stages, and then specifying the order in which the stages should be run. The Transform framework handles the details of scheduling map, combine, and reduce stages into jobs to submit to Hadoop.

For example, if the first stage of a Transform is a reduce stage, the framework knows to set up an identity mapper in the first Hadoop job created to pass records directly to the reducer.

The only restriction on the order in which stages are run is that Combine stages must be followed by a Reduce stage.

Another item to note is that the first Map stage of a transform receives Record objects from the input Koverse Data Collections. Subsequent stages receive whatever objects are emitted by previous stages.

If a stage fails, the errors are reported to the User Interface and subsequent stages are cancelled.

Stages are defined by subclassing one of the Stage types described below.

RecordMapStage

This type of stage operates on Records from the input Data Collections specified when the Transform was configured.

```
public void map(Record record)
```

KVMapStage

This type of stage is used when mapping over the output of a previous stage.

```
public void map(Object key, Object value)
```

CombineStage

A CombineStage is used to locally combine the output of a previous map stage before the keys and values are sent to a ReduceStage. A CombineStage must be followed by a ReduceStage

```
public void combine(Object key, Iterable<Object> values)
```

ReduceStage

A ReduceStage takes a key and a set of values and emits one or more new key value pairs for consumption by a subsequent Stage, or writes Koverse Records to the output Collection in the data store.

```
public void reduce(Object key, Iterable<Object> values)
```

Emitter

The emitter is used to either send key value pairs to the next Stage or to write Records to the output collection. Usually all but the last Stage emit key value pairs and the last Stage writes Records.

Key value pairs emitted by emit() are sent to HDFS where they are read by a subsequent Stage and then deleted whereas Records emitted from writeRecord are written to the output Collection of the Transform and are indexed and made searchable according to the configuration of the output Collection.:

```
emit(Object key, Object value)

writeRecord(Record record)
```

Transform Runner

The transform runner is reponsible for assembling MapReduce jobs out of stages and incrementing a given transform job's current stage. The runner will peek at proceeding stages in an attempt to execute map, combine and reduce stages as parts of a single job. After configuring a job, it will submit the job to the cluster.

Transform class

Stages are packaged up into a single Transform by defining a subclass of the Transform class.

Security

Koverse ensures that a Transform only reads records from collections from which the submitting user is authorized to read. In addition, any restrictions on the imported with additional security labels is applied so that individual records that the user is not authorized to see are not delivered to the Transform for processing.

The output Records of each Transform are labeled by the framework so that access to them is controlled.

Tips and Tricks

- When writing transform logic, keep in mind that Koverse Records may vary in structure. As such, one cannot assume that certain fields will be present, or that the content of fields will conform to any particular format. Code must be defensive against variation in fields and their values.

Import Transforms API

Koverse ImportTransforms allow Records to be transformed during an Import job.

ImportTransforms can be parameterized to allow users to configure the ImportTransform at runtime. Parameters can be accessed via the setup method thus:

```
public void setup(Map<String, Object> params) throws IOException
```

Developers can grab the values of Parameters and store them for use in the transform method.

The core of an ImportTransform is the transform method:

```
public Iterable<SimpleRecord> transform(SimpleRecord inputRecord)
```

The transform method takes one input SimpleRecord and returns zero or more SimpleRecords.

Export Transforms API

Koverse ExportTransforms can be used to transform Records as they are being written to a Koverse Sink.

ExportTransforms can be parameterized to allow users to configure the ExportTransform at runtime. Parameters can be accessed via the setup method thus:

```
public void setup(Map<String, Object> params) throws IOException
```

Developers can grab the values of Parameters and store them for use in the transform method.

The core of an ExportTransform is the transform method:

```
public Iterable<SimpleRecord> transform(SimpleRecord inputRecord)
```

The transform method takes one input SimpleRecord and returns zero or more SimpleRecords.

Sinks API

Koverse Sinks are designed to write Koverse Records to external data stores. For example, customers often want transformed data exported into HDFS for follow-on processing by down stream systems. Java developers can create custom Sinks to support specific destination data stores.

Sinks are executed as MapReduce jobs with only a map phase. The sinks API provides an interface that allows the developer to open a connection to an outside system, deliver records, and then close that connection.

See the Koverse SDK Project section for details about a ready made project for creating custom sinks.

Export File Formats API

Developers can extend ExportFileFormat to easily create new ways to export Koverse Records to file-based Sinks. ExportFileFormats are parameterized like other classes.

There are three primary methods to define when creating an ExportFileFormat:

```
public void startFile()
```

startFile is used to do initialization. The method getOutputStream() can be used to get a reference to the OutputStream to which SimpleRecords are written. Some ExportFileFormats wrap the OutputStream object with other objects to make it easier to output records.

This method can also be used to write out header information to the output file.

```
public void writeRecordToFormat(SimpleRecord record) throws IOException
```

This writeRecordToFormat method is used to output individual records to the output file. SimpleRecord objects can be converted into the bytes that the file format requires.

```
public void endFile()
```

The endFile function is used to write out any footer information required by the file format. It is not necessary to close the OutputStream as this is done automatically by the super class.

Parameters

Koverse Transforms, Sources, and Sinks are all configured via Parameters. Parameters are defined by the developer and allow specific instances of Transforms, Sources, and Sinks to be configured and deployed into varying environments by authorized non-developer users.

When creating a specific implementation of a Transform, Source, or Sink, developers provide a list of Parameters to present to the end-user via the User Interface.

Parameters are created with the following fields:

- **String parameterName** (required) - uniquely identifies the parameter within the class.
- **String displayName** (required) - the name of the parameter that is shown to the user.
- **String type** (required) - one of the possible types defined in Parameter (see below).
- **String defaultValue** (optional) - a value set as the default.
- **String referencedParameterNames** (optional) - any parameterName that should be referenced. For example, for Parameters of the type TYPE_COLLECTION_FIELD, the possible values presented to the user in the UI are taken from the parameter defined in the referencedParameterName.
- **Boolean required** (optional) - whether the parameter must be set by the user. The default is false
- **Boolean hideInput** (optional) - whether the value of the parameter should be hidden in the UI. Used for sensitive parameters such as passwords.
- **String hint** (optional) - a string of text to be shown to the user as an additional hint for applying a value to the parameter.

For example, a Source may define a parameter in its constructor as follows:

```
private static final String URL_PARAMETER = url;


public NewsFeedSource() {
inputParameters.add(
new Parameter(
URL_PARAMETER,
"RSS Feed URL",
Parameter.TYPE_STRING,
"http://rssfeedurl.xml"));
}
```

Parameters can be of the following types:

- TYPE_STRING - for passing in single line short strings such as a hostname or URL.
- TYPE_TEXT - for passing in longer multi-line strings, such as an entire script.
- TYPE_BOOLEAN - presents a checkbox to the user and is set to true or false.
- TYPE_INTEGER - allows the user to specify an integer value.
- TYPE_FILE - Allows the to user choose a file from the local file system. The file is uploaded, and its contents are made available as a stream at execution time to the custom component.

- TYPE_COLLECTION_FIELD - allows the user to select a single field from a collection. The referencedParameterName must be equal to the parameterName of an TYPE_INPUT_COLLECTION or TYPE_OUTPUT_COLLECTION parameterName. This is useful for informing classes of a specific field to use.
- TYPE_COLLECTION_MULTIPLE_FIELD - allows the user to choose a set of fields from a collection selected as an input or output collection parameter. This is useful for informing classes of a specific set of fields to use.

There are additional Parameter types used primarily by the system:

- TYPE_INPUT_COLLECTION - an input collection parameter presents the user with a list of collections from which the user is authorized to read. The UI then fills in this parameter with the internal unique ID of the collection the user chose. This component generally allows the end-user to select multiple input collections. The contents of all input collections are read into transform and export jobs for example.
- TYPE_OUTPUT_COLLECTION - an output collection parameter presents the user with a list of collections to which the user is authorized to write. The UI then fills in this parameter the internal ID of the collection the user chose. This parameter generally only allows the user to select a single collection.
- TYPE_SECURITY_LABEL_PARSER - presents the user with a list of Security Label parser options. Security label parsers are responsible for translating from a source security label to a Koverse record security label.

Transforms are pre-configured with parameters for input and output Data Collections. Sources and Sinks are pre-configured with output or input collections, respectively.

REST API

Note: Clients written in Javascript can use the Javascript SDK rather than interacting directly with the REST API.

Koverse provides an HTTP REST API for providing access to third party tools and integrations. This documentation explains how to access the REST API, and provide third party integrations such as widgets and data management. All responses, and HTTP payload requests are encoded in JSON.

See the REST API generated documentation for a complete list of methods and their signatures. The REST API documentation is hosted in koverse itself open https://{yourKoverseInstance}/Koverse/docs/rest/

Response Messages

All response messages from the REST API are encoded in JSON, and include common attributes on the base response object. The most important attribute is the success boolean flag, that indicates whether the requested operation was successful. If the success value is false, then there will be a failureMessage attribute that provides a plain english statement as to the reason.

Example:

{"success": false, "failureMessage": "Something went wrong."}

Commonly used methods

Almost all applications will require the following functionality

- User Authentication and Authorization
- Fetching Data Collections
- Performing Queries

Additional Methods

- User management

- Collection management
- Index management
- Kicking off imports, transforms, exports
- Many others

API Tokens

Koverse Administrators can create API Tokens, which are used by outside systems to authenticate. These are generally unused outside of the context of a direct users request. For example, a server that periodically updates it's own cache using a Koverse query.

All REST API methods can be called using an API token to authenticate. The API Token takes precedence over any other method of authentication. Here is an example of using an API token to authenticate:

```
``http://<host:port>/Koverse/api/system/status?apiToken=API-TOKEN-HERE``
```

Example REST API Methods

Ping

```
http://<host:port>/Koverse/api/ping
```

A ping request shows that the Koverse HTTP REST API is available, and responsive. Use the ping response method to monitor basic system availability.

Example Ping Request

The following URL shows a ping request, for a Koverse server running on localhost.

```
http://localhost:8080/Koverse/api/ping
```

Example Ping Response

```
{"success":true}
```

System Status

```
http://<host:port>/Koverse/api/status
```

The system status method provides basic system status information. Use this method to integrate against system feature availability. For example, while in lock down mode, Koverse will not provide accesss to requests for data. Therefore it is important to know the basic status of the Koverse system to provide reasonable requests.

Example Status Request

```
http://localhost:8080/Koverse/api/status
```

Example Status Response

```
{"success":true,"systemStatus":{"lockDown":false}}
```

Session Authentication (Login)

```
http://<host:port>/Koverse/api/login/<name>/<password>
```

Example login failure response:

```
{"success":false,"failureMessage":"Login denied. Check username and password"}
```

Example login success response:

```
{"success":true,"user":{"id":1,"emailAddress":"admin","groups":
[{"id":1,"name":"Administrators","staticPermissions":["manageUsersAndGroups","manageDataCollections"
,"manageSystemSettings","audit","manageSources","manageLockDown","manageMapReduceJobs"]}]}}
```


Before using other REST API methods, an HTTP session must be established. Below is the URL and an example for login. The HTTP response to the login will include a JSESSIONID cookie that must be included in all future REST API calls.

Example Login URL

The following would retrieve an HTTP response with a JSESSIONID token for the default administrative user and password.

```
http://localhost:8080/Koverse/api/login/admin/admin
```

Querying for data

The most basic feature of the Koverse REST API is to provide query/search access to data collections. Below is an example of querying all data collections for a logged-in user.

```
http://<host:port>/Koverse/api/query/<queryHere>
```

Example Query

The following would query a Koverse instance running on localhost, port 8080, for the term test.

```
http://localhost:8080/Koverse/api/query/test
```

Additional Methods

See the Koverse REST API Generated Docs for details about the many other methods available.

Pig Scripts

Koverse supports using *Pig <http://pig.apache.org>* as a transform. Pig transforms are simple pig scripts - where Koverse defines the load and store functions. To use Pig, follow these steps.

1. Open the Data Flow app.
2. Click Add Transform
3. Choose 'Pig' from the transform type drop down.
4. Choose Input and Output collections.
5. Write the Pig script in provided text area.

Koverse automatically provides the "load" and "store" functions. You'll simply need to write a Pig script that references the input collections by name, and assigns a value to the output collection by name. Pig variables are case sensitive, and have some restrictions. Therefore Koverse transforms Data Collection names to use only case sensitive alphanumeric and underscore characters. Also, Pig table names cannot start with a non Alpha character (A-Z or a-z) - therefore Koverse prepends the character A when a data collection name starts with a non alpha character. Here are some example data collection name conversions.

- "My 1st Data Collection" = My_1st_Data_Collection
- "P*22" = P_22
- "cAsE sEnSiTiVe" = cAsE_sEnSiTiVe
- "9Items" = "A9Items"
- "_Items" = "A_Items"

Koverse records are converted into tuples. The field names are applied as the Pig field names with the same conversion as above.

While Koverse allows unstructured data, Pig requires highly structured data. The schema defined for Pig fields is derived using the data type(s) seen in the Collection Details Field's page in Koverse. If a field has only a single type detected, the conversions in the table below are used directly. If a field has more than one type detected, and one of those types is a String, all values for that field will take a chararray type in Pig. Otherwise, if more than one type is detected but none are Strings, for example, if a field is 90% Number and 10% Date, it will be defined as a double value type in Pig. Here are the conversions of Koverse data types to Pig data types.

Koverse	Pig
String	chararray
Number	double
Date	DateTime
KoverseGeoPoint	[double,double]
Byte[]	bytearray
Object	map

Example Pig Scripts

The following is a simple pig script that would copy the contents of DataCollection1 to DataCollection2:

```
DataCollection2 = DataCollection1
```

This more complex Pig script would perform a Group By operation on fieldA with a sum on fieldb:

```
A = GROUP DataCollection1 BY fieldA;
DataCollection2 = FOREACH A GENERATE FLATTEN(group) as fieldA, SUM(fieldB) as fieldBSum;
```

Pig Transforms Special Considerations

Pig transforms are executed as multiple stage map reduce jobs. They're considered "non-incremental" transforms in Koverse. Never restart the koverse-server process while a Pig transform is executing - as the job's state will be lost and the job will never finish.

3rd Party Authentication and Authorization

Koverse can be extended to integrate with existing enterprise authentication and authorization systems that may be required for a given production environment. While an extensible component that is built against the Koverse SDK, these authentication and authorization modules are not like other extensible components like Sources and Transform and packaged into a Koverse AddOn. Instead, these modules need to be built into a JAR and placed in the classpath of the Koverse Webapp. Additionally, the koverse-webapp.properties needs to be modified to identify the module(s) that Koverse should use for authentication and authorization.

Implementing an Authentication and Authorization Module To implement an authentication and authorization module, a developer will extend the `AbstractWebAppAuthModule` class. This is a Guice module that enables the injection of new authentication and authorization implementations. There are two ways to implement authentication, either with the `HttpServletRequestAuthenticator` or the `WebAppParameterAuthenticator`. The `HttpServletRequestAuthenticator` enables authentication based on information in the HttpServletRequest, such as an X.509 certificate. The `WebAppParameterAuthenticator` enables authentication based on custom, named parameters. To pass external groups or security tokens to Koverse, implement a `WebAppAuthorizer`.

Full examples of these classes can be found in the Koverse SDK Project .

Application Server Configuration The module and implementations described above need to be built into a JAR file which is placed in the classpath of the Koverse Webapp. This could be done through a mechanism like a JBoss Module or by simply putting the JAR into the WEB-INF/lib directory of the Koverse Webapp.

Koverse Webapp Configuration To update the active authentication and authorization modules used by the Koverse Webapp, set the `com.koverse.webapp.auth.modules` property in koverse-webapp.properties to a comma separated list of Guice modules.

Java Client

Introduction

The Java Client allows JVM based software to connect to and interact directly with Koverse through the REST API. It is capable of interacting with the REST API using plain HTTP connections, SSL, and SSL with client PKI certificates.

The general concept is to instantiate an implementation of the `com.koverse.client.java.KoverseClient` interface and invoke its methods. There are two such implementations, one for interacting with the Koverse web application via REST and another directly to the Koverse server using Thrift.

These instructions will focus on the REST based implementation because the Thrift based implementation is still a work-in-progress.

Basics

To use the Java client in your software, modify your project's Maven pom.xml to include the koverse java client dependency. First, include a repositories section in your `pom.xml` file and add the koverse repository, for example:

```
<repositories>
        <repository>
        <id>koverse</id>
        <name>Koverse Public Repo</name>
        <url>http://nexus.koverse.com/nexus/content/groups/public/</url>
        <layout>default</layout>
        </repository>
</repositories>
```

This will allow maven to download the Koverse java client dependency. This dependency has the groupId `com.koverse` and the artifactId `koverse-client-java`. Set the version of that dependency to match the version of Koverse your software will be communicating with. Your `pom.xml` file should now have a section similar to:

```
<dependencies>
        <dependency>
        <groupId>com.koverse</groupId>
        <artifactId>koverse-client-java</artifact>
        <version>1.2.0</version>
        </dependency>
</dependencies>
```

Note that if your IDE integrates with Maven, it should be able to download JavaDocs for the koverse client software and display them for you. If you'd like to download the JavaDocs yourself, visit http://nexus.koverse.com/nexus/content/groups/public/com/koverse/koverse-client-java/ in your browser, select the folder for your version of Koverse, and download the Javadoc archive.

Lastly:

In your Java code, you will be instantiating an instance of `com.koverse.client.java.KoverseConnection`. Note that the constructor takes an argument of a `com.koverse.client.java.KoverseConnection`. You choose which implementation of KoverseConnection to use in order to specify whether to use plain un-encrypted HTTP or encrypted HTTP (e.g. HTTP over SSL, TLS).

Unencrypted HTTP Connections

Begin by creating an instance of `com.koverse.client.java.PlainKoverseConnector`. Note that its constructor requires you to provide a valid Koverse API Token and the base URL of Koverse (e.g. http://www.myserver/Koverse). The, create an instance of KoverseConnection, supplying the PlainKoverseConnector you just created as the sole constructor argument.

Now, you may use the create KoverseConnection object to perform operations such as:

- Retrieve collection, including names and collection identifiers.
- Insert, update, and delete records.
- Get user and system information.
- Retrieve collection statistics and download records in bulk.
- Perform queries.
- Perform auto-complete queries.

Please view the JavaDocs for the interface `com.koverse.client.java.KoverseClient` for further details on these operations.

Encrypted HTTP Connections

Configuring the KoverseClient to use SSL is somewhat more involved. IT can be further complicated by using client side certificate authentication. As such, let's being with just setting up a SSL connection for now. Client side certificate authentication will be explained in the next section.

Before I begin, please note that this information is also documented in the JavaDocs for the `com.koverse.client.java.SecureKoverseConnector` class. Please feel free to reference that as well.

The important thing to realize is that the use of SSL is configured through the standard JVM mechanism of using special system properties and a Java Key Store.

Since it is most likely the case that the server is not assigned a certificate issued by a trusted CA (Certificate Authority), we must configure your Java software to use a self-signed certificate used by the Koverse server.

As such, the first thing you must do is create a Java keystore that will contain the certificate for the Koverse server. That is done by using the Java `keytool` command, such as so:

```
keytool -import -alias koverseserver -file koverseserver.crt -storepass $PASS -keystore koverseserver.keystore
```

In the above example, we are creating an entry named `koverseserver` in the keystore located in the file `koverseserver.keystore` from the contents of the certificate file `koverseserver.crt`. Additionally, we are protecting the contents of the keystore by encrypting it with the password stored in the environment variable $PASS.

Getting this certificate stored into the keystore is the first step.

The next step is to define special Java system properties when your program is executed so that Java will use the information in the keystore. Those system properties are:

`-Djavax.net.ssl.trustStoreType=jks`

`-Djavax.net.ssl.trustStore=koverseserver.keystore`

`-Djavax.net.ssl.trustStorePassword=$PASS`

Your program must either be run with the above command line properties or you must programmatically add them to the JVM's System Properties at runtime.

With this done, your software should be capable of interacting with a SSL enabled Koverse Server. However, in the case that things don't seem to be working for you, there are some tips that can help.

1. Be sure to contact us for support
2. Apply the system property `-Djavax.net.debug=all` to get lots of good SSL debugging output.

Encrypted HTTP Connections with Client Side Certificates

To use client side certificates, do the same as in the previous section, but also make sure the following system properties are set in your software as well:

```
-Djavax.net.ssl.keyStoreType=pkcs12
```

```
-Djavax.net.ssl.keyStore=clientcertificate.p12
```

```
-Djavax.net.ssl.keyStorePassword=$PASS
```

Where you are specifying your client certificate that is located in the file `clientcertificate.p12`. This file is a `pkcs12` formatted file, protected by the password stored in the system environment variable $PASS

Spark SQL Introduction

Koverse 1.4 supports Transforms written using the Apache Spark API, including Spark SQL. Spark SQL allows Koverse records to be processed using the popular SQL language, which is useful for many common operations such as reshaping data, filtering, combining, and aggregation.

Spark SQL can be used in two ways in Koverse Transforms: first, using the generic Spark SQL transform, developers can simply paste a SQL script into a new instance of a Spark SQL Transform in the Koverse UI.

Second, transform developers can create Koverse AddOns which include Spark SQL statements as part of a Java or Scala class. These can be packaged, uploaded to the Koverse platform, and reused to transform multiple input collections.

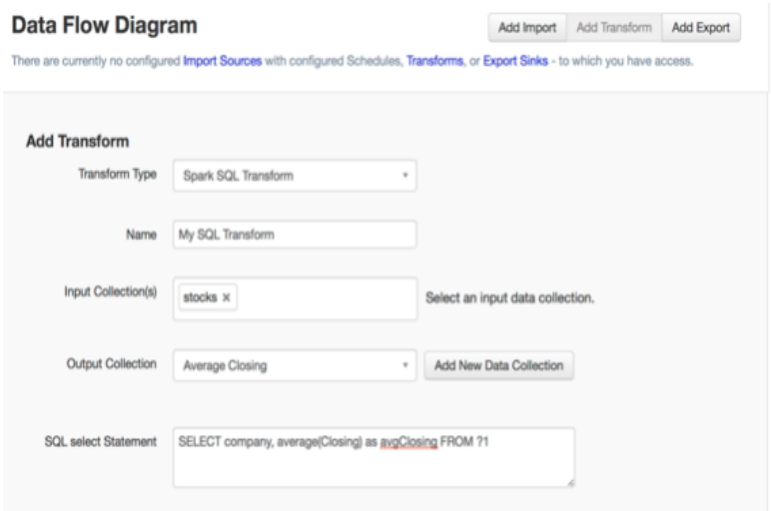
Using the generic Spark SQL Transform

Koverse ships with a generic Spark SQL Transform that allows users to simply paste a Spark SQL statement into a text parameter and applies that script to the input collection specified.

To create a transform like this, start in the Data Flow application from the main menu. Click ‘Add Transform’ and select ‘Spark SQL Transform’ from the drop down list.

Configure the input collections desired.

In the text input marked ‘SQL select statement’, paste in a SQL statement. When specifying a table, use position parameters to identify which input collection should be used. For example, if you’ve selected an input collection ‘stocks’ in the input collections control it will be referenced in the SQL statement as $1. The second input collection is referenced as $2 and so on.



For a description of the SQL statements that are supported see https://spark.apache.org/docs/latest/sql-programming-guide.html

Building a new AddOn that includes Spark SQL statements

Developers can also build Koverse AddOns that can leverage the Spark SQL API in transforms. To create a Spark SQL transform, create a Java class that extends JavaSparkSQLTransform and implement the required methods.

An abbreviated example is as follows. Create a new class extending JavaSparkSQLTransform:

```
…
import com.koverse.sdk.transform.spark.sql.JavaSparkSqlTransform;
import com.koverse.sdk.transform.spark.sql.JavaSparkSqlTransformContext;
import org.apache.spark.sql.DataFrame;
import org.apache.spark.sql.SQLContext;
…

public class MySparkSqlTransform extends JavaSparkSqlTransform {
```

Provide any parameters you wish to expose to users to configure this transform, and the basic information about the transform that will help users identify it in the UI:

```
public Iterable<Parameter> getParameters()
public String getName()
public String getTypeId()
public Version getVersion()
.....
```

Now in the execute() method, access is provided to a SqlContext. Input collections are already loaded as data frames into the Sql Context and only need to be referenced in SQL statements. The input collection IDs can be accessed via the JavaSparkSqlTransformContext:

```
protected DataFrame execute(JavaSparkSqlTransformContext context) {{

final SQLContext sql = context.getSqlContext();

final List<String> inputCollections =

context.getJavaSparkTransformContext()

.getInputCollectionIds();

…
```

Collection Schemas are also available, but they do not need to be used to execute SQL statements. They are there in case additional DataFrames need to be created.

```
Map<String, CollectionSchema> schemas =

    context.getJavaSparkTransformContext().getInputCollectionSchemas();
```

SQL statements should be edited to reference the input collection IDs, and can be executed simply by passing the SQL string to the SqlContext. The resulting data frame should be returned and Koverse will persist the output as a new collection.

```
return sql.sql(sqlStatement)
```

Spark Transform API

Introduction

Koverse now supports the Apache Spark cluster computing framework through a set of native Koverse APIs that leverage much of the Spark primitives. The Koverse Spark APIs allow a developer of Koverse a set of routines, protocols, and tools for building software applications based upon the Koverse technology.

See the *Addons* section for information about building an addon that contains a class that uses the Koverse Spark API.

The following is a high-level outline of the Koverse Spark API framework:

Interface SparkTransform

com.koverse.sdk.transform.spark

@ThreadSafe

@Immutable

public interface SparkTransfor

Description: The following methods, when executed in order, obtain information on how to execute the transform: *getName(), getVersion() and getParameters()*. These methods are used to configure the transform before performing execution using `execute(com.koverse.sdk.transform.spark.SparkTransformContext)` which is passed a **SparkTransformContext** to give it the information needed to run the spark transform.

Modifier and Type	Method	Description
org.apache.spark.rdd.RDD <SimpleRecord>	getName()	Koverse calls this method to execute your transform.
String	getName()	Get the name of this transform.
Iterable<Parameter>	getParameters()	Get the parameters of this transform.
String	getTypeId()	Get a programmatic identifier for this transform.
Version	getVersion()	Get the version of this transform.
boolean	supportsIncrementalProcessing()	Whether the transform supports incremental output.

Example

```
final RDD<SimpleRecord> actual; actual = se.execute(sparkTransformContext);
```

Interface SparkTransformContext

com.koverse.sdk.transform.spark

@NotThreadSafe

@Immutable

public interface SparkTransformContext

Description: Given to a SparkTransform when it is executed. Provides context information to assist in the execution.

Modifier and Type	Method	Description
Map<String,org.apache.spark.rdd.RDD <SimpleRecord>>	getInputCollectionRDDs()	Get all Koverse input collection RDDs from the parameters that were input by the user.
Map<String,CollectionSchema>	getInputCollectionSchemas()	Get the schemas for all input collections.

Map<String,String>	getInputCollectionSchemas()	Get all parameters that is input by the user, with the exception of collection parameters (which are given as RDDs). None of the keys or values in the returned map will be null.
org.apache.spark.SparkContext	getSparkContext()	Get the spark context to use during execution

Class JavaSparkTransform

com.koverse.sdk.transform.spark

@ThreadSafe

@Immutable

public abstract class JavaSparkTransform extends Object implements SparkTransform, Serializable

Description: A version of of spark transforms that are easier to work with when the spark code is written in Java.

Modifier and Type	Method	Description
protected abstract org.apache.spark.api.java.JavaRDD<SimpleRecord>	execute(JavaSparkTransformContext sparkTransformContext)	Koverse calls this method to execute your transform
org.apache.spark.rdd.RDD<SimpleRecord>	execute(SparkTransformContext sparkTransformContext)	Invokes execute(com.koverse.sdk.transform.spark.JavaSparkTransformContext) after wrapping up the Scala specific types into Java friendly types.
boolean	supportsIncrementalProcessing()	Override this method if transform supports incremental processing - i.e.

Class JavaSparkTransformContext

com.koverse.sdk.transform.spark

@Immutable

@NotThreadSafe

public final class JavaSparkTransformContext extends Object

Description: A version of the Spark Transform Context more tailored for use with pure Java Spark code.

Modifier and Type	Method	Description
Map<String,org.apache.spark.api.java.JavaRDD<SimpleRecord>>	getInputCollectionRDDs()	Get all Koverse input collection RDDs from the parameters that were input by the user.
Map<String,CollectionSchema>	getInputCollectionSchemas()	Get the schemas for all input collections.
Map<String,String>	getParameters()	Get all parameters that is input by the user, with the exception of collection parameters (which are given as RDDs) None of the keys or values in the returned map will be null.

org.apache.spark.api.java.JavaSparkContext	getSparkContext()	Get the spark context to use during execution.

Class SparkTransformLoader

com.koverse.sdk.transform.spark

public class SparkTransformLoader extends Object

Description:

Modifier and Type	Method	Description
String	getName()	Get name
List<Parameter>	getParmeters()	Get all the parameters input by user
String	getTypeId()	Get Type Id
Version	getVersion()	Get the spark version

Spark SQL Transform API

Koverse now supports the Apache Spark SQL via a set of native Koverse Spark SQL APIs that let the user query structured data as a distributed dataset (RDD). This makes it easy to run SQL queries.

See the *Addons* section for information about building an addon that contains a class that uses the Koverse Spark SQL API.

The following is a high-level outline of the Koverse Spark SQL API framework:

Class JavaSparkSqlTransform

com.koverse.sdk.transform.spark.JavaSparkTransform

@Immutable

@ThreadSafe

public abstract class JavaSparkSqlTransform extends JavaSparkTransform

Description: A transform for executing Spark SQL query transforms

Modifier and Type	Method	Description
protected abstract org.apache.spark.sql.DataFrame	execute(JavaSparkSqlTransformContext context)	Execute the Spark SQL query.
protected org.apache.spark.api.java.JavaRDD<SimpleRecord>	execute(JavaSparkTransformContext sparkTransformContext)	Koverse calls this method to execute your transform

Class JavaSparkSqlTransformContext

com.koverse.sdk.transform.spark.JavaSparkTransform

@NotThreadSafe

public final class JavaSparkSqlTransformContext extends Object
Description: The context for a JavaSparkSqlTransform

Modifier and Type	Method	Description
JavaSparkTransformContext	getSparkTransformContext()	Get the JAva spark tranform context, if needed.
org.apache.spark.sql.SQLContext	getSqlContext()	Get the SQL context, which is ready to go and loaded with the schemas for the input collections.

Class KoverseSparkSql

com.koverse.sdk.transform.spark.sql

public class KoverseSparkSql extends Object

Description:

Modifier and Type	Method	Description
static org.apache.spark.sql.DataFrame	createDataFrame(org.apache.spark.api.java.JavaRDD<org .apache.spark.sql.Row> rowRdd, org.apache.spark.sql.SQLContext sqlContext, org.apache.spark.sql.types.StructType schema)	Create a new DataFrame from an RDD of rows, a SQL Context, and a struct type (the Spark SQL schema)
static org.apache.spark.sql.DataFrame | getSqlContext()	createDataFrame(org.apache.spark.api.java.JavaRDD<SimpleRecord> recordRdd, org.apache.spark.sql.SQLContext sqlContext, FlatCollectionSchema collectionSchema)	Create a new DataFrame from an RDD of records, a SQL Context, and a flat collection schema
static org.apache.spark.api.java.JavaRDD<org.apache.spark.sql.Row>	createRowRdd(org.apache.spark.api.java.JavaRDD<SimpleRecord> recordRdd, FlatCollectionSchema collectionSchema)	Converts a RDD of records and a flat collection schema into a RDD of rows.
static org.apache.spark.sql.SQLContext | getSqlContext()	createSqlContext(org.apache.spark.SparkContext sparkContext, Map<String,org.apache.spark.api.java.JavaRDD<SimpleRecord>> recordRdds, Map<String,FlatCollectionSchema> collectionSchemas)	Converts two maps keyed by collection name, one containing record RDDs and the other containing collection schema, into a SQLContext ready for query.
static org.apache.spark.sql.types.StructType	createSqlSchema(FlatCollectionSchema collectionSchema)	cGiven a flat collection schema, create s Spark SQL Struct type, which the SQL schema.

For a reference of the supported query syntax in the Spark Java SQL see:

http://savage.net.au/SQL/sql-99.bnf.html

http://docs.datastax.com/en/datastax_enterprise/4.6/datastax_enterprise/spark/sparkSqlSupportedSyntax.html

Spark Scala Transform API with Examples

These examples give a quick overview of the Koverse Spark Scala Transform API. Koverse 1.4 currently provides a single transform API in Scala.

The Koverse transform class called: **SimpleSparkTransform()**

To use the Scala Transform, simply run the **SimpleSparkTransform.execute()** method with the proper arguments; JavaSparkContext and org.apache.spark.api.java.JavaRDD.

Please refer to JavaDoc's for full detailed usage description.

The transform consists of the following high level steps:

1. Get the 'projectField' field property from the JavaSparkContext
2. Map the input collection RDDs to Scala map
3. Get the collection from the Scala map
4. Scan and pull out java records/objects from RDD
5. Output the total record count for Java records
6. Output the total record count for Scala records

Here is an example of a Spark Scala execute() method:

```
  protected def execute(context: JavaSparkTransformContext): JavaRDD[SimpleRecord] = {
      val field = context.getParameters.get(C.FIELD_PARAM)
      println(s"looking for field $field in the records")

      val map = mapAsScalaMap(context.getInputCollectionRDDs)
      println("mapped input collection RDDs to scala map")

      val collectionKV = map.iterator.next
      println(s"got collection ${collectionKV._1} from map")

      val rdd = JavaRDD.toRDD(collectionKV._2)
      println("pulled out RDD from tuple")

      val transformRDD = rdd
          .filter(r => r.containsKey(field))
          .map(r => {
          val outputRecord: SimpleRecord = new SimpleRecord

          if(r.containsKey(field)) {
          outputRecord.put(field, r.get(field))
          } else {
          outputRecord.put(field, "NOTHING")
          }

          println(s"${field} => ${r.get(field)}")
          outputRecord
    })
  println(s"total java records ${transformRDD.count()}")

  val output = JavaRDD.fromRDD(transformRDD)
  println(s"total scala records ${output.count}")

  output
}
```

You can run Java and Scala examples by passing the class name to Spark's bin/run-example script; for instance:

`./bin/run-example <scala class>`

For a description of the Spark Scala statements that are supported see the Scala Docs at:

https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.package

Spark Java API with Examples

Koverse 1.4 supports Transforms written using the Apache Spark API. Koverse APIs leverages much of the Spark primitive abilities that can be applied by writing a custom Transform or use an existing Transform provided by the Koverse API.

`public class MySparkSqlTransform extends JavaSparkSqlTransform {`

Provide any parameters you wish to expose to users to configure this transform, and the basic information about the transform that will help users identify it in the UI:

public Iterable<Parameter> getParameters()

Example:

```
@Override
    public Iterable<Parameter> getParameters() {
            ArrayList<Parameter> params = new ArrayList<Parameter>();
            params.add(new Parameter(FIELD_PARAM, "Field to project", Parameter.TYPE_COLLECTION_FIELD));
            return params;
    }
```

public String getName()

Example:

```
@Override
    public String getName() {
            return "Spark Java";
    }
```

public String getTypeId()

Example:

```
@Override
    public String getTypeId() {
            return "Spark Java Transform";
    }
```

public Version getVersion()

Example:

```
@Override
    public Version getVersion() {
            return new Version(0, 1, 0);
    }
```

Here is an example of usage (**Create a new class extending JavaSparkTransform()**):

```
final JavaSparkTransform javaSparkTransform;
final SparkTransformContext sparkTransformContext
final RDD<SimpleRecord> actual;

javaSparkTransform = new JavaSparkTransform() {
        @Override
        public Iterable<Parameter> getParameters() {
            throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public String getName() {
            throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public String getTypeId() {
            throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public Version getVersion() {
            throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        protected JavaRDD<SimpleRecord> execute(JavaSparkTransformContext sparkTransformContext) {
            return sparkTransformContext.getInputCollectionRDDs().get("input");
        }
   };

   actual = javaSparkTransform.execute(sparkTransformContext);
}
```

For a complete description of the Spark Java APIs that are supported see the Spark Java Docs at: https://spark.apache.org/docs/latest/api/java/index.html

Custom Transforms Code Examples

This code example is provided as a bootstrap to developing your own 'custom transform'. The 'companyTransform' class presented here can be used as a template.

Custom Transform Example:

```
package com.company.transform;

import com.koverse.sdk.data.Parameter;
import com.koverse.sdk.data.Record;
import com.koverse.sdk.data.TermTypeDetector;
import com.koverse.sdk.transform.AbstractRecordMapStage;
import com.koverse.sdk.transform.AbstractReduceStage;
import com.koverse.sdk.transform.AbstractTransform;
import com.koverse.sdk.transform.TransformStage;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.net.InetAddress;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Map.Entry;

public class CompanyTransform extends AbstractTransform {

private static Logger logger = LoggerFactory
            .getLogger(CompanyTransform.class);

public static class CompanyCustomValue implements Writable {

     public String companyCustomValue;

     public CompanyCustomValue() {
     }

     public CompanyCustomValue(String myCustomValue) {
            this.companyCustomValue = myCustomValue;
     }

    @Override
     public void write(DataOutput out) throws IOException {
            out.writeUTF(companyCustomValue);
     }

     @Override
     public void readFields(DataInput in) throws IOException {
            companyCustomValue = in.readUTF();
     }
}

private static final String PARAM_MY_CUSTOM = "myCustomParam";

public CompanyTransform() {
} // Necessary for Hadoop

public static class CompanyCustomMapStage extends AbstractRecordMapStage {

     private String companyCustomParam = null;

     /**
      * Perform Mapper setup here. Read parameters, setup data structures,
      * etc
      */
     @Override
```

Class KoverseSparkSql

```
	public void setup() {
		companyCustomParam = getStageContext().getParameterValue(
				PARAM_MY_CUSTOM);
	}

	/*
	 * This mapper will takes in a list of IP address for each record and
	 * create all unique combinations in any direction i.e.
	 * 127.0.0.1,255.255.255.255 is the same as 255.255.255.255, 127.0.0.1
	 */
	public void map(Record inputRecord) throws IOException,
			InterruptedException {

		HashMap<String, ArrayList<String>> mapOfIps = new HashMap<String, ArrayList<String>>();

		ArrayList<String> ipsArrayList = new ArrayList<String>();

		// System.out.println("********This is the mapper running!*********");

		for (Entry<String, Object> fields : inputRecord.fields.entrySet()) {

			Object value = inputRecord.get(fields.getKey());

			// Get the record value and nested values
			checkIP(value, ipsArrayList);

		}

		// call to get unique pairs
		uniquePairs(ipsArrayList, mapOfIps);

		/*
		 * emit resulting map using key and custom class in the format of
		 * {"127.0.0.1,255.255.255.255", count} The sort and group function
		 * will then combine all identical keys and create larger lists,
		 * which are then sent to reducer to do the final count for each
		 * grouping
		 */
		CompanyCustomValue myCustomValueClass = null;

		for (Entry<String, ArrayList<String>> fields : mapOfIps.entrySet()) {

			// System.out.println("this is the new data structure");

			String key = fields.getKey();

			ArrayList<String> ips = (ArrayList<String>) mapOfIps.get(key);

			// System.out.println("for emit new key is:" + key);

			myCustomValueClass = new CompanyCustomValue(
					Integer.toString(ips.size()));

			getStageContext().emit(new Text(key.toString()),
					myCustomValueClass);
		}
	}

	@Override
	public Class<Text> getMapOutputKeyClass() {
		return Text.class;
	}

	@Override
	public Class<CompanyCustomValue> getMapOutputValueClass() {
		return CompanyCustomValue.class;
	}

	// recursive function takes record and then continues to iterate through
	public void checkIP(Object value, ArrayList<String> ipsArrayList) {
		if (value instanceof List) {

			try {
				Iterator<?> iterator = ((List<?>) value).iterator();

				while (iterator.hasNext()) {
					Object listValue = (Object) iterator.next();
					checkIP(listValue, ipsArrayList);

				}

				// System.out.println("this value is instance of list");
```

```
                } catch (Exception e) {
                        e.printStackTrace();
                }

        } else if (value instanceof Map) {

                try {

                        Map<?, ?> result = (Map<?, ?>) value;

                        Iterator<?> iterator = result.keySet().iterator();

                        while (iterator.hasNext()) {
                                Object resultValue = result.get(iterator.next());

                                checkIP(resultValue, ipsArrayList);
                        }

                        // System.out.println("this value is instance of map");

                } catch (Exception e) {
                        e.printStackTrace();
                }

        } else if (value instanceof InetAddress) {
                ipsArrayList.add(((InetAddress) value).getHostAddress());

                // System.out.println("check it is INET:" + ((InetAddress)
                // value).getHostAddress());

        } else if (value instanceof String) {

                String removedSlash = ((String) value).replace("/", "");

                if (TermTypeDetector.typify(removedSlash) instanceof InetAddress) {
                        ipsArrayList.add(removedSlash);
                } else {
                        // System.out.println("This is not INET!:" + removedSlash);
                }
        }
}

public void uniquePairs(ArrayList<String> ipsArrayList,
                HashMap<String, ArrayList<String>> mapOfIps) {

        // go through list and build unique ip address pairs
        String ipAddress = "";
        String ipAddress2 = "";
        String pair = "";

        ArrayList<String> pairs = new ArrayList<String>();

        for (int i = 0; i < ipsArrayList.size(); i++) {
                ipAddress = (String) ipsArrayList.get(i);

                for (int j = i; j < ipsArrayList.size(); j++) {
                        if (j == i)
                                continue;

                        ipAddress2 = (String) ipsArrayList.get(j);

                        pair = ipAddress + "," + ipAddress2;

                        // System.out.println(pair);

                        pairs.add(pair);
                }

        }

        // take unique list of pairs that is any directional and build a
        // HashMap with ArrayList of ip pairs
        for (int i = 0; i < pairs.size(); i++) {
                String testPair = (String) pairs.get(i);

                String[] indIps = testPair.split(",");
                String firstPart = (String) indIps[0];
                String secondPart = (String) indIps[1];
                String testReversePair = secondPart + "," + firstPart;

                if (mapOfIps.get(testPair) != null) {
                        ArrayList<String> testList = (ArrayList<String>) mapOfIps
                                        .get(testPair);
```

```
                        testList.add(testPair);
                } else if (mapOfIps.get(testReversePair) != null) {
                        ArrayList<String> testList = (ArrayList<String>) mapOfIps
                                        .get(testReversePair);
                        testList.add(testReversePair);
                } else {
                        ArrayList<String> testList = new ArrayList<String>();
                        testList.add(testPair);
                        mapOfIps.put(testPair, testList);
                }

        }

    }

}

/*
 * The reduce will count all of the ip pairs and write them through Record.
 * The count for each grouping will occur and then records will be written
 * out independent of other reduce tasks.
 */
public static class CompanyCustomReduceStage extends AbstractReduceStage {

    private String companyCustomParam;

    /** Perform setup here */
    public void setup() {
        companyCustomParam = getStageContext().getParameterValue(
                        PARAM_MY_CUSTOM);
    }

    /** Perform main work here */
    @Override
    public void reduce(Object feature, Iterable<Object> entities)
                    throws IOException {

        // System.out.println("******This is the reduce running!*******");

        // System.out.println("feature: " + feature.toString());

        Iterator<Object> i = entities.iterator();

        int mergedCount = 0;

        while (i.hasNext()) {
                CompanyCustomValue count = (CompanyCustomValue) i.next();

                mergedCount += Integer.parseInt(count.companyCustomValue);

                // System.out.println("count: " + count.companyCustomValue);
                // System.out.println("merge count: " + mergedCount);

        }

        String[] splitIPs = feature.toString().split(",");
        ArrayList<String> listIPs = new ArrayList<String>();

        for (int l = 0; l < splitIPs.length; l++) {
                listIPs.add("/" + splitIPs[l]);
        }

        Record myCustomRecord = new Record();
        myCustomRecord.addField("IP_ADDRESS", listIPs);
        myCustomRecord.addField("count", mergedCount);

        try {
                // Write the record to the data store, if this is the last stage
                getStageContext().writeRecord(myCustomRecord);

        } catch (InterruptedException e) {
                logger.error(e.getMessage(), e);
        }

    }

    public Class<Text> getMapOutputKeyClass() {
        return Text.class;
    }

    public Class<CompanyCustomValue> getMapOutputValueClass() {
        return CompanyCustomValue.class;
    }
```

```
  }

  @Override
  protected void fillInParameters(List<Parameter> parameters) {
       // Add custom parameters
       parameters.add(new Parameter(PARAM_MY_CUSTOM, "Custom Parameter",
                       Parameter.TYPE_STRING));
  }

  @Override
  public String getName() {
       return "Company IP Address Transform";
  }

  @Override
  public String getJobTypeId() {
       return "companyTransform";
  }

  @Override
  protected void fillInStages(List<Class<? extends TransformStage>> stages) {
       /**
        * Add all stages in order here
        */
       stages.add(CompanyCustomMapStage.class);
       stages.add(CompanyCustomReduceStage.class);
  }

  @Override
  public String getVersion() {
       return "1.0.0";
  }
}
```

Aggregations

Aggregations are configured on a Data Collection so that custom statistics on Records in the Collection can be maintained over time. Aggregations are pre-computed on the Records in a Data Collection so that they can be queried with sub-second response time, regardless of the number of Records in a collection. Unlike Transforms which output new immutable Records, Aggregations can update previous values as they run, for instance updating the count of some event in a period of time. This characteristic makes Aggregations a perfect solution for use cases like analytics dashboards. The sections below will go into more detail of how Aggregations are configured and then queried, in the context of a web log analytics use case.

Example Aggregations Use Case

While certainly not required, Aggregations are often used in conjunction with streaming Imports to maintain near real-time statistics on a constant stream of Records. In this example, assume web log events are streaming into Koverse via a Kafka Source. These web log event Records look like the following JSON:

```
{
  timestamp: 1440785470000
  userId: 4581
  location: "Denver, CO, USA"
  url: "/docs/index.html"
}
```

The fields in these Records are:

- timestamp: UNIX timestamp in milliseconds
- userId: unique integer id field for the logged in user
- location: string denoting the city, state, and country of the user
- url: the page requested

The questions we want to answer are:

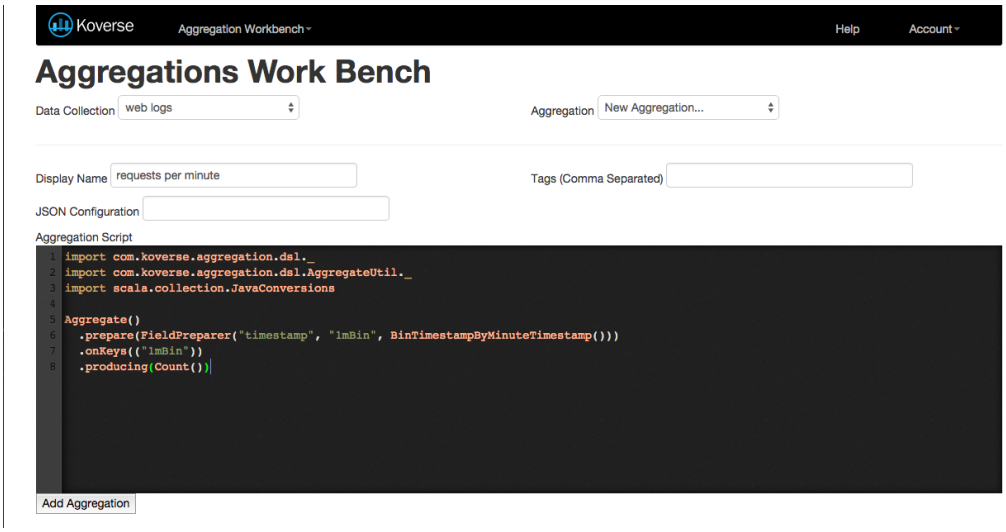
1. How many requests per minute is the web site receiving?
2. How many unique users are visiting the site each day?
3. How many unique users are visiting the site each day by country?

Creating Aggregations

Aggregations can be created in two ways:

1. Using the Aggregation Workbench App at /Koverse/apps/aggregationworkbench
2. Creating a JSON configuration file that is automatically loaded from *$KOVERSE_SERVER_HOME/conf/load-every-time*

The image below shows how to create an Aggregation using the Aggregation Workbench App.



The steps to using the app are:

1. Select the Data Collection you want to create an Aggregation on
2. Set a Display Name that describes the Aggregation
3. Write your Aggregation Script. The Aggregation script is a Scala DSL that defines how Aggregations are built on the Records in the Data Collection. More detail is provided below.
4. Click the Add Aggregation button. This will save the Aggregation and create an Aggregation job to process any existing Records in the Data Collection.

The alternative way of creating Aggregations is through a JSON configuration file that is loaded by Koverse Server. The Aggregation seen in the screenshot above would look like the following JSON:

```
{
  "aggregates": [
    {
      "dataCollection" : {
        "name": "web logs"
      },
      "configurationOptions": {},
      "displayName": "timeseries",
      "tags":[""],
      "definition":"import com.koverse.aggregation.dsl._\nimport com.koverse.aggregation.dsl.AggregateUtil._\nimport scala.collection.JavaConversions\n\nAggreg
    }
```

```
  ]
}
```

The most important part is the Aggregation Script/definition. It is described in sections below.

These imports should be at the beginning of every script. The Aggregation Script is compiled on the server when it is saved, so add additional imports that would be required for the Scala code to compile as needed. Currently Aggergations don't support the use of 3rd party libraries, so only imports from standard Java and Scala packages are supported:

```
import com.koverse.aggregation.dsl._
import com.koverse.aggregation.dsl.AggregateUtil._
import scala.collection.JavaConversions
```

The creation and configuration of a *com.koverse.aggregation.dsl.Aggregate* object forms the body of the code:

```
Aggregate()
.prepare(FieldPreparer("timestamp", "1mBin", BinTimestampByMinuteTimestamp()))
.onKeys(("1mBin"))
.producing(Count())
```

The *prepare* method allows you to create new fields in your Records for the purpose of using them as dimensions in the following *onKeys* method. This doesn't actually add new fields to your Records stored in Koverse, but creates temporary fields just for the Aggregation. The *prepare* method takes 0 or more *FieldPreparers* which map an existing field from the Records, in this case "timestamp", to a new field, in this case "1mBin", by applying the Function found in the final argument, in this case *BinTimestampByMinuteTimestamp()*:

```
.prepare(FieldPreparer("timestamp", "1mBin", BinTimestampByMinuteTimestamp()))
```

The *onKeys* method provides the dimensions to build the Aggregation on. This is similar to a GROUP BY in SQL. There can be more than one field name listed as dimensions as seen in additional examples below:

```
.onKeys(("1mBin"))
```

The *producing* method lists the Aggregation function(s) to apply to each Record:

```
.producing(Count())
```

To summarize this example, the UNIX timestamp of each Record is mapped into 1-minute bins, and then the number of Records within each 1-minute bin is counted:

FieldPreparer

As seen in the example above, the *prepare* method takes 0 or more *FieldPreparers*. The definition of the *FieldPreparer* class looks like:

```
class FieldPreparer[I: ClassTag, O](inputFieldName: String, outputFieldName: String, function: Function1[I, Option[O]]) extends Preparer
```

The first two parameters are straightforward, the name of the existing field in the Record and the name of the new field that will be created, respectively. The last argument is a function that does the projection from the value of inputFieldName to the value for outputFieldName. The system uses the type information to only apply the function if the input value is of type **I**. For example, if you pass in the function:

```
.prepare(FieldPreparer("text", "textLength", { text: java.lang.String => Some(text.length()) })
```

then the FieldPreparer will only operate on input values of type *java.lang.String*. This is helpful if your Records have different typed values for the same inputFieldName.

There are several off-the-shelf functions you can drop into FieldPreparers, for example the *BinTimestampByMinuteTimestamp()* is an off-the-shelf function for taking an input UNIX timestamp and rounding it down to the minute and returning that UNIX timestamp. This function, as well as others like *BinTimestampByHourTimestamp()*, are useful for binning events for timeseries analysis. Below is a table documenting the existing functions that may be dropped into a *FieldPreparer*

Object Name	Input Type	Output Type	Description
BinTimestampByMinuteTimestamp()	java.lang.Long	java.lang.Long	Rounds UNIX timestamp down to the minute
BinTimestampByNMinuteTimestamp(n: Long)	java.lang.Long	java.lang.Long	Rounds UNIX timestamp down to the Nth minute
BinTimestampByHourTimestamp()	java.lang.Long	java.lang.Long	Rounds UNIX timestamp down to the hour
BinTimestampByDayTimestamp(tz: Option[TimeZone] = None)	java.lang.Long	java.lang.Long	Rounds UNIX timestamp down to the day. TimeZone is optional. Default to GMT
BinTimestampByMonthTimestamp(tz: Option[TimeZone] = None)	java.lang.Long	java.lang.Long	Rounds UNIX timestamp down to the month. TimeZone is optional. Default to GMT
BinDateByMinute(tz: Option[TimeZone] = None)	java.util.Date	java.lang.String	Formats date into yyyy_MM_dd_HH_mm. TimeZone is optional. Default to GMT
BinDateByHour(tz: Option[TimeZone] = None)	java.util.Date	java.lang.String	Formats date into yyyy_MM_dd_HH. TimeZone is optional. Default to GMT
BinDateByDay(tz: Option[TimeZone] = None)	java.util.Date	java.lang.String	Formats date into yyyy_MM_dd. TimeZone is optional. Default to GMT
BinDateByMonth(tz: Option[TimeZone] = None)	java.util.Date	java.lang.String	Formats date into yyyy_MM. TimeZone is optional. Default to GMT
BinDateByYear(tz: Option[TimeZone] = None)	java.util.Date	java.lang.String	Formats date into yyyy. TimeZone is optional. Default to GMT
Tokenize(regex: String)	java.lang.String	java.util.List[java.lang.String]	Splits input String by the supplied regex

You can also write your own and pass it in as an anonymous function as was seen in the String length example above. Remember the return value of the function is an *Option* which can be used in case your function can't or doesn't want to return a value. The input and output types of these functions must be ones that are supported by Koverse Records.

Aggregation Functions

In the example above, we saw the aggregation function *Count()*. Counting, while the most popular, is certainly not the only aggregation function. The table below describes each of the available aggregation functions. Currently there is no means for supplying User Defined Aggregation Functions (UDAF).

Function	Description

Count()	Counts the number of Records
CountMap(field: String)	Builds a Map[String, Long] with keys being the Record's value (as a String) for the given field, and the value being the count. For example, we could have CountMap("url") which would maintain a single aggregate value with the counts for each URL instead of maintaining seperate counts for each URL. The benefit is that all the counts are kept together and all of the keys are enumerated. The drawback is that if you have too many distinct keys (>1000s), this Map will grow too large.
TopK(field: String)	Similar to CountMap, but an approximate Top-K, so it won't store all keys and counts so it is safe to use when the number of keys is very large. It will return the top-25.
SumIntegeger(field: String)	Sums the integer values for the given field.
SumDecimal(field: String)	Sums the decimal values for the given field.
Min(field: String)	Maintains the minimum numeric value for the given field
Max(field: String)	Maintains the maximum numeric value for the given field
Average(field: String)	Calculates the average over the numeric values for the given field
StringSet(field: String)	Maintains a Set of the distinct values for the given field
CardinalityEstimate(field: String)	Estimates the cardinality of the values for the given field. This uses the HyperLogLog+ algorithm internally.

QuantileEstimate(field: String)	Estimates the distribution of the values for the given field. Returns the follow percentiles: .01%, .1%, 1%, 2%, 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 95%, 98%, 99%, 99.9%, 99.99%

Additional Examples

The first example answered the question, "How many requests per minute is the web site receiving?" by counting events in 1-minute bins. Below you can find the Aggregations for the second and third questions we wanted to answer in this web log analytic use case.

"How many unique users are visiting the site each day?":

```
import com.koverse.aggregation.dsl._
import com.koverse.aggregation.dsl.AggregateUtil._
import scala.collection.JavaConversions

Aggregate()
  .prepare(FieldPreparer("timestamp", "1dBin", BinTimestampByDayTimestamp(java.util.TimeZone.getTimeZone("EST"))))
  .onKeys(("1dBin"))
  .producing(CardinalityEstimate("userId"))
```

In the example above, we round the timestamp down to the day, based on the EST timezone. This will group all of the events that happened on the same day, and then use the CardinalityEstimate to get the approximate number of distinct users.

"How many unique users are visiting the site each day by country?":

```
import com.koverse.aggregation.dsl._
import com.koverse.aggregation.dsl.AggregateUtil._
import scala.collection.JavaConversions

Aggregate()
  .prepare(FieldPreparer("timestamp", "1dBin", BinTimestampByDayTimestamp(java.util.TimeZone.getTimeZone("EST")),
           FieldPreparer("location", "country", { location: String => location.split(raw"\s")(2) })))
  .onKeys(("1dBin", "country"))
  .producing(CardinalityEstimate("userId"))
```

In the example above, we create the same 1-day bins, but also add second dimension which is the Country, parsed from the Record's location field. The parsing here is simplified and has no error handling for brevity.

Aggregation Query API

The sections above have gone into detail about how to configure Aggregations on the Records in a Data Collection. As originally stated, the primary use case for Aggregations is to maintain precomputed statistics over time to support interactive (sub-second) queries from applications such as analytic dashboards. This section will provide detail on the query API. The REST API will be discussed, but a Thrift API is also available and is very similar.

Queries are submitted via HTTP POST requests to /Koverse/api/query/aggregate. The Content-Type header should be set to "application/json". An example query for the first example above might look like:

```
{
  "collectionId":"web_logs_20150828_212035_291",
  "dimensionValuesPairs":[
    {
      "dimensionValues":[{"field":"1mBin","value":"1440785460000"}],
      "producer":{"type":"count"}
    }
  ],
  "generateTotal":true,
}
```

This will query the web log Data Collection for the event count in the 1-min bin of 1440785460000. This would have been the events that occured between 18:11:00 and 18:12:00 GMT on Fri, 28 Aug 2015. The dimensionValuesPairs property is an array so a single query may contain many dimensionValues which enables you to batch requests which can be useful when pulling the data for a timeseries graph for example. There currently is no range query, so instead you would batch together all of the 1mBin values that you need to render your graph. The requests are also batched on the server so this ends up being fast even if your query has 100s of dimensionValues.

Below we show another query, but this one is for the 3rd Aggregation example from above, the number of unique users per country per day:

```
{
  "collectionId":"web_logs_20150828_212035_291",
  "dimensionValuesPairs":[
    {
      "dimensionValues":[{"field":"1dBin","value":"1441166400000"}, {"field":"country", "value":"USA"}],
      "producer":{"type":"cardest", "relation":"userId"}
    }
  ],
  "generateTotal":true,
}
```

Here we see how the field (or relation) is specified in conjunction with the aggregation function. We also see how additional dimensions can be added to the query easily. Below is a table mapping the Scala aggregation function to the type used in the query API.

Function	Type String
Count	count
CountMap	countmap
TopK	topk
SumInteger	sumint
SumDecimal	sumdec
Min	min
Max	max
Average	ave
StringSet	set
CardinalityEstimate	cardest
QuantileEstimate	quantest

The generateTotal property above enables the query to request a final reduction on the server for when the query returns more than one value. This can be very useful in certain cases where the client can't perform the reduction itself. For example, you could aggregate and query for the individual event counts for each day of a week and then add these values together on the client to get a total number of events for the week. What if you were trying to get the total number of unique users for the week? You are likely to get a very wrong answer if you simply add up the unique users for each day of the week as the same users may access the web site on several days during the week. By requesting the final reduction on the server, it can properly merge the data structures that hold the cardinality estimates and then return the total.

The query response looks very similar to the query, but with values:

```
{
  "recordCountEstimate": 0,
  "responseTime": 0,
  "success": true,
```

```
  "recordsWritten": 0,
  "aggregateQueryResult": {
      "collectionId": "web_logs_20150828_212035_291",
      "aggregateValues": [
          {
              "dimensionValuesProducerPair": {
                  "dimensionValues": [
                      {
                          "field": "1mBin",
                          "value": "1440785460000"
                      }
                  ],
                  "producer": {
                      "type": "count"
                  }
              },
              "value": "3"
          }
      ],
      "total": "38",
      "lastAggregationExecuted": 1440797400467
  }
}
```


Here we see there were 38 events for the 1-minute bin that was queried. The query response also shows the last time an aggregation job was run and completed, which provides a "freshness" to the results.

Using Python with Koverse

Python is a popular interpreted programming language.

Koverse ships with a Python client to allow Python scripts to access the Koverse API.

The Koverse Python client uses Apache Thrift to communicate with the Koverse server. It is possible to generate clients for other languages as well.

To use the Koverse Python client, do the following:

```
sudo pip install koverse
Downloading/unpacking koverse
 Downloading koverse-1.4.0-py2.py3-none-any.whl (144kB): 144kB downloaded
Requirement already satisfied (use --upgrade to upgrade): thrift in /Library/Python/2.7/site-packages (from koverse)
Requirement already satisfied (use --upgrade to upgrade): kafka-python in /Library/Python/2.7/site-packages (from koverse)
Requirement already satisfied (use --upgrade to upgrade): six in /Library/Python/2.7/site-packages (from kafka-python->koverse)
Installing collected packages: koverse
Successfully installed koverse
Cleaning up...
```

The Koverse Python client can then be used in Python scripts by importing the koverse module:

```
$ python
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from koverse import client
```

Connecting to the Koverse Server

The Python client can connect to the hostname of the Koverse Server (note: this is not the address of the Koverse Web App).:

```
>>> client.connect('localhost')
```

If for some reason the client loses the connection to the Koverse Server, such as when the Koverse Server is restarted, the client can reconnect simply by calling client.connect() again.

Users can authenticate themselves to the Koverse server using their username and base-64 encoded passwords:

```
>>> import base64
>>> client.authenticateUser('myusername', base64.b64encode('mypassword'))
>>>
```

If the authentication is unsuccessful an exception is raised:

```
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
 File "/Library/Python/2.7/site-packages/koverse/client.py", line 93, in authenticateUser
   tUser = ugClient.authenticateUser(auth, None, parameters)
 File "/Library/Python/2.7/site-packages/koverse/thriftgen/usergroup/UserGroupService.py", line 782, in authenticateUser
   return self.recv_authenticateUser()
 File "/Library/Python/2.7/site-packages/koverse/thriftgen/usergroup/UserGroupService.py", line 807, in recv_authenticateUser
   raise result.ke
koverse.thriftgen.ttypes.TKoverseException: TKoverseException(_message='No authenticated user found')
```

Querying Koverse Collections

The Koverse Python client can be used to interactively query collections, fetch samples, create collections and run transforms.

To query one or more collections, use the client's query() method. In this example, we'll query Koverse for any collection that has a value above 100 in a field named 'Close'.:

```
>>> results = client.query({'Close': {'$gt': 100.0}})
>>> len(results)
736
```

Results are returned as a list of Python dicts, each representing a record from a Koverse collection:

```
>>> import pprint
>>> pprint.pprint(results[0])
{'AdjClose': 34.9,
 'Close': 256.88,
 'Date': time.struct_time(tm_year=42304, tm_mon=11, tm_mday=6, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=6, tm_yday=311, tm_isdst=0),
 'High': 267.88,
 'Low': 199.25,
 'Open': 263.84,
 'Volume': 236228300}
```

Koverse records contain fields and values. Values may be of a simple type such as int and date, but may also contain lists or dicts.

To query a specific set of collections, specify an optional parameter with a list of collection names to query:

```
>>> client.query({'Close': {'$gt': 100.0}}, ['stocks'])
```

or, by using the name parameter 'collections':

```
>>> client.query({'Close': {'$gt': 100.0}}, collections=['stocks'])
```

Clients can also request that the results be limited to a set number, and can request that the Koverse server deliver results beginning at a specified offset. For example:

```
>>> client.query({'Close': {'$gt': 100.0}}, collections=['stocks'], limit=10, offset=100)
```

Clients can also request that the Koverse Server return only a subset of the fields in each record by specifying a list of field names to include:

```
>>> pprint.pprint(client.query({'Close': {'$gt': 100.0}}, collections=['stocks'], limit=10, offset=100, fields=['Close']))
[{'Close': 110.88},
```

```
 {'Close': 111.56},
 {'Close': 111.25},
 {'Close': 110.75},
 {'Close': 111.63},
 {'Close': 111.25},
 {'Close': 111.5},
 {'Close': 111.25},
 {'Close': 111.5},
 {'Close': 111.5}]
```

Fetching Collection Samples

Because Python runs on a single machine, and because Koverse collections may contain a large volume of records, it can be useful to work with a sample of a collection's records, especially when building statistical models designed to be trained on a representative sample.

Koverse maintains representative samples for all collections by default. These samples can be retrieved by the client using the getSamples() method:

```
>>> samples = client.getSamples('stocks')
>>> len(samples)
1000
```

Uploading resource files

One advantage of Python is that is has a number of well supported libraries for doing sophisticated data analysis , such as numpy (http://www.numpy.org), scipy (http://www.scipy.org), nltk for natural language processing (http://nltk.org), pandas for data manipulation and analysis http://pandas.pydata.org, scikit-learn for machine learning (http://scikit-learn.org/stable/), etc.

For this simple example, we'll model the distribution of day to day changes in stock prices so we can identify anomalous jumps or dips in price. We can pull a sample of the stock prices from Koverse using the getSamples() method:

```
>>> samples = client.getSamples('stocks')
```

We'll model the day-to-day changes in price as a gaussian random walk (https://en.wikipedia.org/wiki/Random_walk#Gaussian_random_walk).:

```
>>> differences = [r['Close'] - r['Open'] for r in samples]
>>> import numpy
>>> mean = numpy.mean(differences)
>>> mean
-0.085472972972972849
>>> stddev = numpy.std(differences)
>>> stddev
8.6134268092274517
```

Now we'll store our 'model', which just consists of these two numbers, the mean and standard deviation, in a file that we can upload and use in a transform. Typically we wouldn't do this for such a simple model, we could pass those numbers as parameters to a transform. But for more complicated models using a file is much more convenient. The storeResourceFile() method will upload the model data to a file in HDFS so that it can be accessed by workers in parallel:

```
>>> import cPickle
>>> modelData = base64.b64encode(cPickle.dumps((mean, stddev)))
>>> modelFilename = client.storeResourceFile('model1',modelData)
>>> modelFilename
'1438664105966model1'
```

Note: we used the numpy package to obtain these parameters, which means numpy must also be installed on our MapReduce worker nodes.

The storeResourceFile() method returns a unique filename that Transform scripts can reference. Now we can use it to score all the daily changes in price to look for anomalous changes, for example: changes that are greater than two standard deviations from the mean. We'll do that in the next section.

Developing an XML Transform (XSLT) to import your XML data as Koverse Records

XML can be imported into Koverse as any file can. However, the format of the imported records in the collection may not be what you are expecting, as it is more of a raw import of generic XML data. To enable the import of your XML data into proper Koverse records, an XSLT must be used to convert your XML into proper Koverse Record XML.

For example, let's say you have the following XML file which you wish to import:

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
       xmlns='http://www.koverse/xml/example/books/1.0.0'>

  <book id="1">
    <author>Miguel De Cervantes</author>
    <title>Don Quixote</title>
  </book>

  <book id="2">
    <author>John Bunyan</author>
    <title>Pilgrim's Progress</title>
  </book>

  <book id="3">
    <author>Daniel Defoe</author>
    <title>Robinson Crusoe</title>
  </book>

</books>
```

For this example, this XML file would conform to your XML schema (XSD):

```
<?xml version="1.0"?>
<xs:schema version="1.0" elementFormDefault="qualified"
           xmlns:tns="http://www.koverse/xml/example/books/1.0.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.koverse/xml/example/books/1.0.0">

  <xs:element name="books" type="tns:books"/>

  <xs:complexType name="books" final="extension restriction">
    <xs:sequence>
      <xs:element name="book" type="tns:book" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="book" final="extension restriction">
    <xs:sequence>
      <xs:element name="author" type="xs:string"/>
      <xs:element name="title" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string"/>
  </xs:complexType>

</xs:schema>
```

Now, to transform this XML into XML that represents Koverse records, the following XSLT would be used:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:b="http://www.koverse/xml/example/books/1.0.0"
  xmlns="http://www.koverse/xml/records/1.0.0">

  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/">
```

```
    <records>

      <xsl:for-each select="b:books/b:book">
        <record>

          <securityLabel></securityLabel>

          <fields>

            <entry>
              <entry-name>id</entry-name>
              <entry-value xsi:type="decimal">
                <value>
                  <xsl:value-of select="@id"/>
                </value>
              </entry-value>
            </entry>

            <entry>
              <entry-name>author</entry-name>
              <entry-value xsi:type="string">
                <value>
                  <xsl:value-of select="b:author"/>
                </value>
              </entry-value>
            </entry>

            <entry>
              <entry-name>title</entry-name>
              <entry-value xsi:type="string">
                <value>
                  <xsl:value-of select="b:title"/>
                </value>
              </entry-value>
            </entry>

          </fields>

        </record>
      </xsl:for-each>

    </records>
  </xsl:template>

</xsl:stylesheet>
```

Which would produce the following XML file that conforms to the Koverse Record XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<records xmlns:b="http://www.koverse/xml/example/books/1.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://www.koverse/xml/records/1.0.0">

  <record>
    <securityLabel/>
    <fields>
      <entry>
        <entry-name>id</entry-name>
        <entry-value xsi:type="decimal">
          <value>1</value>
        </entry-value>
      </entry>
      <entry>
        <entry-name>author</entry-name>
        <entry-value xsi:type="string">
          <value>Miguel De Cervantes</value>
        </entry-value>
      </entry>
      <entry>
        <entry-name>title</entry-name>
        <entry-value xsi:type="string">
          <value>Don Quixote</value>
        </entry-value>
      </entry>
    </fields>
  </record>

  <record>
    <securityLabel/>
    <fields>
```

```
      <entry>
        <entry-name>id</entry-name>
        <entry-value xsi:type="decimal">
          <value>2</value>
        </entry-value>
      </entry>
      <entry>
        <entry-name>author</entry-name>
        <entry-value xsi:type="string">
          <value>John Bunyan</value>
        </entry-value>
      </entry>
      <entry>
        <entry-name>title</entry-name>
        <entry-value xsi:type="string">
          <value>Pilgrim's Progress</value>
        </entry-value>
      </entry>
    </fields>
  </record>

  <record>
    <securityLabel/>
    <fields>
      <entry>
        <entry-name>id</entry-name>
        <entry-value xsi:type="decimal">
          <value>3</value>
        </entry-value>
      </entry>
      <entry>
        <entry-name>author</entry-name>
        <entry-value xsi:type="string">
          <value>Daniel Defoe</value>
        </entry-value>
      </entry>
      <entry>
        <entry-name>title</entry-name>
        <entry-value xsi:type="string">
          <value>Robinson Crusoe</value>
        </entry-value>
      </entry>
    </fields>
  </record>

</records>
```

Finally, for your reference, here is the complete Koverse XML Schema:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema elementFormDefault="qualified" version="1.0"
           xmlns:tns="http://www.koverse/xml/records/1.0.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.koverse/xml/records/1.0.0" >

  <xs:element name="records" type="tns:records"/>

  <xs:complexType name="boolean" final="extension restriction">
    <xs:complexContent>
      <xs:extension base="tns:scalar-field">
        <xs:sequence>
          <xs:element name="value" type="xs:boolean"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="scalar-field" abstract="true">
    <xs:complexContent>
      <xs:extension base="tns:field">
        <xs:sequence/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="field" abstract="true">
    <xs:sequence/>
  </xs:complexType>

  <xs:complexType name="date" final="extension restriction">
```

```
    <xs:complexContent>
      <xs:extension base="tns:scalar-field">
        <xs:sequence>
          <xs:element name="value" type="xs:dateTime"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="array" final="extension restriction">
    <xs:complexContent>
      <xs:extension base="tns:container-field">
        <xs:sequence>
          <xs:element name="element" type="tns:field" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="container-field" abstract="true">
    <xs:complexContent>
      <xs:extension base="tns:field">
        <xs:sequence/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="map" final="extension restriction">
    <xs:complexContent>
      <xs:extension base="tns:container-field">
        <xs:sequence>
          <xs:element name="entry" type="tns:entry" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="entry" final="extension restriction">
    <xs:sequence>
      <xs:element name="entry-name" type="xs:string"/>
      <xs:element name="entry-value" type="tns:field"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="records" final="extension restriction">
    <xs:sequence>
      <xs:element name="record" type="tns:record" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="record" final="extension restriction">
    <xs:sequence>
      <xs:element name="securityLabel" type="xs:string"/>
      <xs:element name="fields" type="tns:map"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="string" final="extension restriction">
    <xs:complexContent>
      <xs:extension base="tns:scalar-field">
        <xs:sequence>
          <xs:element name="value" type="xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="decimal" final="extension restriction">
    <xs:complexContent>
      <xs:extension base="tns:scalar-field">
        <xs:sequence>
          <xs:element name="value" type="xs:decimal"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

Running a Python Script as a Transform

Koverse supports running Python scripts in Transforms. These transforms are simple map-only transforms.

We'll write our Python script for scoring daily stock changes based on our model. The list of any resource files included will be passed in as an argument to our script. In our case, we have one model filename. If there are multiple resource files, they will be separated by commas:

```
>>> script = '''
#/usr/bin/python

import numpy
import cPickle
import base64
import sys
import json

# load our model
modelFile = sys.argv[1]
f = open('/tmp/' + modelFile)
mean, stddev = cPickle.loads(base64.b64decode(f.read()))
f.close()

# records from input collections are delivered as JSON objects via stdin
for line in sys.stdin:

       record = json.loads(line.strip())

       # calculate price change
       change = record['Close'] - record['Open']

       # if change is more than two standard deviations from the mean
       # consider it anomalous and output the record
       if abs(change - mean) / stddev > 2.0:
              print json.dumps(record)
              sys.stdout.flush()

'''
```

Be sure to call sys.stdout.flush() after outputting a new record.

Any libraries our script needs to use should be installed on all the MapReduce worker nodes before hand. Care should be taken to ensure the proper versions of libraries are installed. See instructions on this site https://www.digitalocean.com/community/tutorials/how-to-set-up-python-2-7-6-and-3-3-3-on-centos-6-4 for tips on installing python 2.7 packages on CentOS.

In our example, workers will need the popular numpy package, which can be installed via:

```
sudo /usr/local/bin/pip install numpy
```

once Python 2.7 and pip are installed.

To get a description of a Transform use the getTransformDescription() method. This will tell us the parameters we need to fill out to create a transform. We're using the Python script Transform that ships with Koverse, identified by the name 'python-transform':

```
>>> desc = client.getTransformDescription('python-transform')
>>> for param in desc.parameters:
...     print param.parameterName + ': ' + param.displayName
...
inputCollection: Input Collection(s)
outputCollection: Output Collection
pythonPathParam: Path to Python Executable
scriptParam: Python script
resourceFiles: Comma separated resource file paths
```

The pythonPathParam should reference the path to the Python executable on MapReduce workers. This allows us to use a particular version of the Python interpreter if necessary.

Define the options we'll pass to our Transform, which includes the Python script and the model filename we stored in the previous section. We don't need to specify the input and output collections here, we'll do that later in the call to create the transform.:

```
>>> options = {
       'pythonPathParam': '/usr/local/bin/python2.7',
       'scriptParam': script,
       'resourceFiles': modelFilename
}
```

Create a collection to store the output:

```
>>> client.createCollection('anomalous changes')
```

To setup a transform, use the createTransform() method.:

```
>>> transform = client.createTransform(
               'python-transform',
               'score daily changes',
               ['stocks'],
               'anomalous changes',
               options)
```

This returns a Transform object. To obtain a list of Transforms that have already been created, use the listTransforms() method.

To run the transform we'll use its run() method:

```
>>> job = transform.run()
```

This will instantiate a MapReduce job that executes our Python script on all of the MapReduce worker nodes in parallel. This way we can process a large amount of data efficiently.

Note that Transforms are configured by default to not run sooner than once per hour. Any jobs submitted earlier than that will be blocked until an hour has passed.

The output will be stored in the output collection we specified. We can examine a sample of the output to verify our results:

```
>>> sampleOutput = client.getSamples('anomalous changes')
>>> first = sampleOutput[0]
>>> print first['Close'] - first['Open']
-22.44
```

This shows an example of a day when a stock dropped by 22.44 points, which is more than two standard deviations from the typical daily change.

The Python client can also be used in the context of Python tools such as iPython Notebook (http://ipython.org/notebook.html). Simply use the same methods described above in iPython Notebooks.

Using Koverse with PySpark

PySpark is the name of Apache Spark's Python API and it includes an interactive shell for analyzing large amounts of data with Python and Spark.

Koverse supports processing data from Koverse Collections using PySpark and storing Resilient Distributed Datasets (RDDs) into Koverse Collections.

To use Koverse with PySpark, follow these steps.

Set the following environment variables:

```
export SPARK_HOME=[your Spark installation directory]
export ACCUMULO_HOME=[your Accumulo installation directory]
export KOVERSE_HOME=[your Koverse installation directory]
export PYSPARK_PYTHON=/usr/local/bin/python2.7
```

Copy the following JAR files into a the Spark installation directory:

```
cd $SPARK_HOME

cp $ACCUMULO_HOME/lib/accumulo-core.jar .
cp $ACCUMULO_HOME/lib/accumulo-fate.jar .
cp $ACCUMULO_HOME/lib/accumulo-tracer.jar .
cp $ACCUMULO_HOME/lib/accumulo-trace.jar .
cp $ACCUMULO_HOME/lib/guava.jar .

cp $KOVERSE_HOME/lib/koverse-sdk-xml*.jar koverse-sdk-xml.jar
cp $KOVERSE_HOME/lib/koverse-sdk-1*.jar koverse-sdk.jar
cp $KOVERSE_HOME/lib/koverse-server-base*.jar koverse-server-base.jar
cp $KOVERSE_HOME/lib/koverse-shaded-deps*.jar koverse-shaded-deps.jar
cp $KOVERSE_HOME/lib/koverse-thrift*.jar koverse-thrift.jar
```

Install Koverse python files. As described above, the Koverse Python client can be installed using:

```
pip install koverse
```

Start PySpark:

```
bin/pyspark --deploy-mode client \
--jars koverse-sdk.jar,koverse-sdk-xml.jar,koverse-thrift.jar, \
accumulo-core.jar,guava.jar,accumulo-fate.jar,accumulo-trace.jar, \
koverse-server-base.jar,koverse-shaded-deps.jar

Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.3.0
      /_/

Using Python version 2.7.6 (default, Sep  9 2014 15:04:36)
SparkContext available as sc, HiveContext available as sqlCtx.
```

To access Koverse's Spark functionality import the following:

```
>>> from koverse.spark import *
```

A KoverseSparkContext object is used to obtain Spark RDDs for specified Koverse collections. Simply pass in the pre-created SparkContext object, the hostname of the Koverse Server, and your username and password:

```
>>> import base64
>>> ksc = KoverseSparkContext(sc, 'localhost', 'username', base64.b64encode('password'))
```

To get an RDD for a Koverse Collection, call the koverseCollection() method:

```
>>> rdd = ksc.koverseCollection('stocks')
```

This rdd can be used like other RDDs.

```
>>> rdd.take(1)
[{u'Volume': 26765000, u'High': 25.42, u'AdjClose': 25.17, u'Low': 24.46, u'Date': datetime.datetime(2014, 9, 1, 20, 0), u'Close': 25.17, u'Open': 24.94}]
```

If, for example, we wanted to repeat our previous analysis of this example data set, we could build a model using a few simple functions:

```
>>> differences = rdd.map(lambda r: {'Date': r['Date'], 'Change': r['Close'] - r['Open']})

>>> sum  = differences.map(lambda r: r['Change']).reduce(lambda a, b: a + b)
>>> mean = sum / differences.count()
>>> mean
-0.08547297297297289

>>> ssq = differences.map(lambda r: (r['Change'] - mean) ** 2).reduce(lambda a, b: a + b)
>>> var = ssq / differences.count()
>>> import math
>>> stddev = math.sqrt(var)
>>> stddev
8.613426809227452
```

Now we can apply our model directly to our differences RDD.

```
>>> anomalies = differences.flatMap(lambda r: [r] if (abs(r['Change']) - mean) / stddev > 2.0 else [])
>>> anomalies.count()
12
>>> anomalies.first()
{'Date': datetime.datetime(1998, 8, 31, 20, 0), 'Change': -22.439999999999998}
```

Note that, unlike the previous example, here we are not setting up a Koverse Transform which means this analysis workflow will only exist during this PySpark session. We can persist the output, but if we want to repeat this process we'll need to run these commands again.

If we wish to persist these anomalies in a Koverse collection to that applications and users can access and search these results we can use the saveAsKoverseCollection() method.

```
>>> ksc.saveAsKoverseCollection(anomalies, 'anomalies')
```

This will create a collection called 'anomalies' and store the information from our RDD into it.

If the collection already exists and we wish to simply add new data to it, we can specify append=True

```
>>> ksc.saveAsKoverseCollection(anomalies, 'anomalies', append=True)
```

Using PySpark and iPython Notebook with Koverse

iPython Notebook is a popular tool for creating Python scripts that can display results and be shared with others.

PySpark can be used in the context of iPython Notebook to create repeatable workflows.

First, follow the steps to configure PySpark to work with Koverse as described in the previous section.

To use Koverse with PySpark and iPython Notebook, create a new iPython profile:

```
ipython profile create pyspark
```

This will create a profile in ~/.ipython/profile_pyspark. In that directory, create a file called ipython_config.py with the following contents:

```
c = get_config()

c.NotebookApp.ip = '*'
c.NotebookApp.open_browser = False
c.NotebookApp.port = 8880
```

Next, in ~/.ipython/profile_pyspark/startup create a file called 00-pyspark-setup.py with the following contents:

```
import os
import sys
```

```
spark_home = os.environ.get('SPARK_HOME', None)
if not spark_home:
   raise ValueError('SPARK_HOME environment variable is not set')
sys.path.insert(0, os.path.join(spark_home, 'python'))
sys.path.insert(0, os.path.join(spark_home, 'python/lib/py4j-0.8.2.1-src.zip'))

execfile(os.path.join(spark_home, 'python/pyspark/shell.py'))

from koverse.spark import *
```

Export the following env vars:

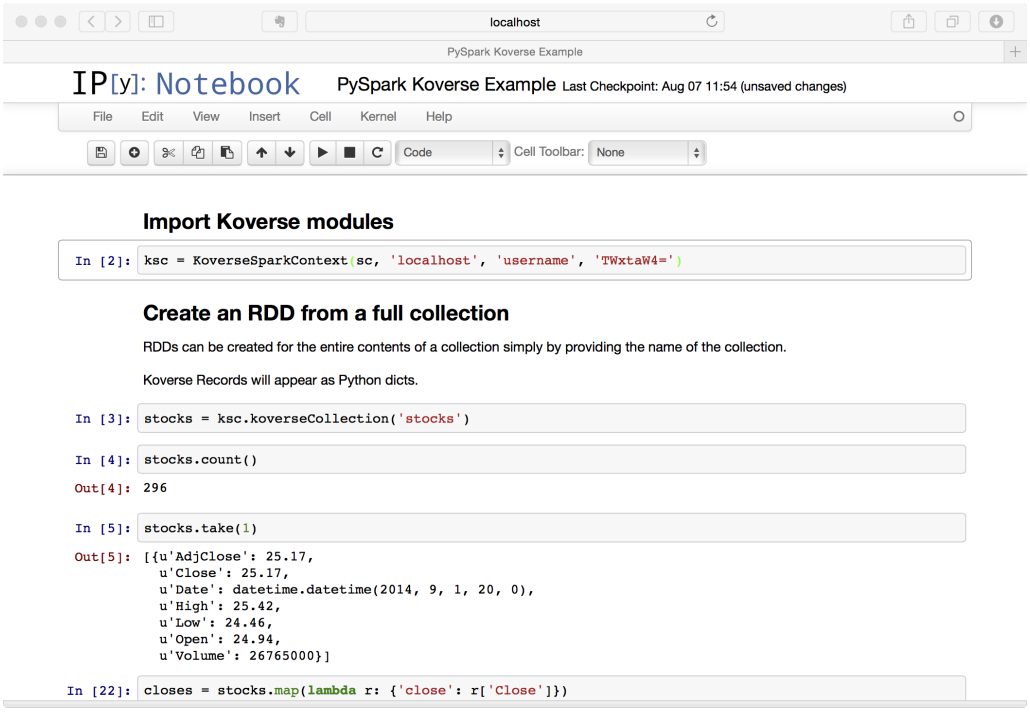
```
export SPARK_HOME=[path to your spark installation]

export PYSPARK_PYTHON=/usr/local/bin/python2.7

export PYSPARK_SUBMIT_ARGS="--deploy-mode client --jars koverse-sdk.jar,koverse-sdk-xml.jar,koverse-thrift.jar,accumulo-core.jar,guava.jar,accumulo-fate.jar,ac

export KOVERSE_HOME=[path to your Koverse installation]
```

Now iPython Notebook can be started from the Spark installation directory:

```
ipython notebook --profile=pyspark
```

Visit http://localhost:8880 in a web browser to access iPython Notebook and create a new notebook. In this new notebook, everything should be imported and initialized for us to start using PySpark with Koverse.

Use the same methods described in the previous section on PySpark in iPython notebooks to obtain RDDs from Koverse collections, process them, and persist RDDs to Koverse collections.



Using PySpark and Jupyter with Koverse

Jupyter is a development tool that allows users to create notebooks containing comments and code, like iPython Notebook. Jupyter supports other languages via the use of 'kernels'.

To use Jupyter with Koverse and PySpark, first create a kernel.json file in a folder called 'koverse'

Configure the kernel.json file as follows by setting the right value for SPARK_HOME:

```
{
 "display_name": "Koverse PySpark",
 "language": "python",
 "argv": [
  "/usr/bin/python",
  "-m",
  "ipykernel",
  "-f",
  "{connection_file}"
 ],
 "env": {
  "SPARK_HOME": "",
  "PYTHONPATH": "$SPARK_HOME/python/:$SPARK_HOME/python/lib/py4j-0.8.2.1-src.zip",
  "PYTHONSTARTUP": "$SPARK_HOME/bin/pyspark",
  "PYSPARK_SUBMIT_ARGS": "--deploy-mode client --jars koverse-sdk.jar,koverse-sdk-xml.jar,koverse-thrift.jar,koverse-server-base.jar,koverse-shaded-deps.jar,ac
 }
}
```

Install the kernel file via the command:

```
ipython kernelspec install koverse/
```

Place the following jars into the $SPARK_HOME folder:

```
accumulo-core.jar
accumulo-trace.jar
commons-validator-1.4.0.jar
koverse-sdk-xml.jar
koverse-server-base.jar
koverse-thrift.jar
accumulo-fate.jar
accumulo-tracer.jar
guava.jar
koverse-sdk.jar
koverse-shaded-deps.jar
```

Install the Koverse python module via:

```
pip install koverse
```

Then you can fire up Jupyter and create a new notebook using the newly installed Koverse kernel.

In that notebook, you can connect to a Koverse instance via:

```
import pyspark
from koverse.spark import *
import base64
sc = SparkContext()
ksc = KoverseSparkContext(sc, 'localhost', 'your-username', base64.b64encode('your-password'))
```

You can create an RDD from a Koverse instance as follows, for example:

```
rentals = ksc.koverseCollection('Customer Rentals')
rentals.take(1)

[{u'email': u'DIANNE.SHELTON@sakilacustomer.org',
  u'first_name': u'DIANNE',
  u'title': u'ACADEMY DINOSAUR'}]
```

You can process the RDD the same as other Spark RDDs:

```
pairs = rentals.map(lambda r: (r['first_name'].lower(), 1))
nameCount = pairs.reduceByKey(lambda a, b: a + b)
nameCount.count()
591
nameCount.take(1)
[(u'sheila', 18)]
```

When you want to write an RDD to Koverse, convert it to be a set of Python dicts and save:

```
ncRecords = nameCount.map(lambda nc: {'name': nc[0], 'count': nc[1]})
ksc.saveAsKoverseCollection(ncRecords, 'name count', append=True)
```

Glossary of Koverse Terminology

- Data Collection - Data Collections are the basic container for data in Koverse. You can think of them like tables - but every record in a data collection can be completely unique in structure.
- Configuration Manager App - The Configuration Manager App gives users the ability to upload and download configuration for Data Collections, Sinks, Sources, and Transforms.
- Data Collections App - The Data Collections App gives users the ability to manage and explore Data Collections. A Data Collection is simply a named collection of records. Collections are the primary mechanism by which data is tracked and managed in Koverse.
- Data Flow - Visualize, configure, and execute the movement of data within the Koverse system.
- File Upload - Upload one or more files from the browser and import it into a collection.

Introduction

Use Cases

General Use Cases

Compliance: The past few years have given rise to a nearly countless number of government and industry regulations that organizations must adhere to. In the financial area there is Sarbanes-Oxley (SOX) and Dodd-Frank. Health care has given us HIPAA. Web sites that handle sensitive or payment data need to follow PCI DSS. Compliance with these regulations requires organizations to have visibility into many parts of the business. This level of visibility is only possible when the organization can access and process a wide range of data generated by every employee, every application, and every customer-facing activity - and convert that into straightforward summarizations that address specific compliance issues. Koverse gives organizations the access, visibility, and analytic capabilities they need to generate these summaries in a secure and controlled manner, resulting in clear and more complete answers to compliance questions in less time.

Organizational Knowledge-base: Organizational data is generated by many users using multiple applications. To further complicate matters users may not be members of the organization. Consider comments to user forums or blog posts, customer emails, and Tweets that are captured for future use. Koverse provides the flexibility needed to search across all organizational data without being restricted by structured vocabularies. This means users are more likely to find the sum total of the information they need to answer questions and make decisions.

Cross-organizational Information Sharing: Groups within organizations create their own data. To achieve strategic objectives, different groups are frequently required to share information. While necessary, this is not always easy. Existing database technologies make it difficult to integrate data from different sources, in addition to the fact that they do not adequately address access control and security; some groups may not want to share confidential information such as customer name or credit card number with people from other groups who do not need it. Koverse provides the tools necessary to collocate and selectively share data, yielding significant and new value.

Industry-Specific Use Cases

Pharma

Much attention has been given recently to opportunities that may exist in the pharmaceutical industry to take advantage of advanced data analytics and data sharing. While a lot of this attention is focused on opportunities--and potentially even mandates--for data transparency during the pharmaceutical research and development phase, there is another much-less-mentioned arena of opportunity: The arena of pharmaceutical sales and marketing.

In every industry, information about customers is invaluable; how to find them, how to reach them, and how to make use of their feedback. Unlike other industries in which the customer is direct, in the pharmaceutical industry, a customer can be one of many indirect entities. This makes sales and marketing in pharma much more complex. Luckily, we are in an era where there is more information than ever available to draw upon to understand customer behavior.

Koverse is the natural place to collect and analyze these massive amounts of data, providing the ability to rapidly consume both structured and unstructured data and perform analysis that would have previously taken months or been impossible; turning data into an advantage. This has the potential to benefit pharmaceutical companies and patients alike.

The Koverse advantage in Reaching Prescribers

Typically, physicians are "deciled" based on their writing patterns and higher deciles (i.e., physicians who prescribe a drug most often) are more aggressively targeted. But what if there is a smarter way to reach the right prescribers?

Koverse can leverage multiple heterogeneous data sets that are available to pharmaceutical companies, including both traditional and nontraditional data sets such as physician prescription data, physician profile data, sales rep activity, census data, and disease statistics. The combination and analysis of these data sets can shed new light on which prescribers should receive attention. For example:

- Regional anomalies could highlight patient populations that could be benefitting from a particular drug, but that currently are not. More attention could then be focused on prescribers in that region.
- Physicians that have characteristics like high prescribers, but are not themselves high prescribers, could present opportunities for education and outreach. In fact, it may be more important to target prescribers in this category, than to focus on those who are already reliably using a drug.

The Koverse Advantage in Reaching Patients

Again, much benefit can be gained by combining data sets in one place and applying advanced analytics on the entire corpus of data. For example, combining aggregate prescription data and data from wholesalers with census and social network data can help to:

- Discover untapped markets or surprising patient trends.
- Track drug launches, patient referral patterns, drug switches and off-label use, and disease trends and locations.

Koverse can also open up unprecedented new opportunities in data sharing. Multiple pharma companies can use the same instance of Koverse securely and decide at a very granular level which data to share. The result is that all parties can mutually benefit from the availability of more data. For example, pooled data on patient prescriptions and disease states can:

- Detect patients who are prescribed a drug manufactured by one company, but whose disease state or drug side effects indicate that they would benefit from a treatment plan that included drugs produced by multiple companies.

The Koverse Advantage in Distributor Interactions

Pharmaceutical manufacturers can realize immense benefits by using Koverse to leverage the massive amounts of electronic transactional data that is gathered from different points in the supply chain. This can provide visibility into what is happening in the marketplace and how various channels, such as hospital, retail pharmacy, and mail order are performing. Within Koverse, supply chain data can augment more traditional pharma data sets to:

- Show not only how revenue is accruing, but also how specific wholesaler contracts are performing.

- Provide opportunities to share data with other pharmaceutical manufacturers and/or wholesalers to ensure compliance with inventory management agreements and detect anomalies such as the introduction of counterfeit drugs into the market. Without Koverses ability to share and integrate data securely, such analyses would not be possible.

Tutorials

> ***Note***
>
> These tutorials are meant to guide first time users through the common activities in Koverse. Each of the End User, System Administrator, and Developer tutorials are a progression of work and rely on the previous sections. Start with the End User tutorials, and continue until you believe the topics are no longer applicable.

Tutorial Overview

This tutorial will provide you step-by-step instructions on how to import a data set into Koverse, view the content of the data set, search for records from the data set, perform a transform that data set, and view the output results of the data set after the transform.

The best way to understand Koverse is by diving in and seeing working examples in action. The following sections of documentation provide step-by-step tutorial of examples on how to use Koverse.

Logging Into Koverse

Koverse application provides a web browser based login that may be customized to your organization's needs. If your organization uses a Single Sign On (SSO) authentication system you may have the ability to be logged-in automatically.

Login Steps:

1. Open a web browser and navigate to http://<server>/Koverse
    - Example: http://mycompany.koverse.com/Koverse
2. Enter the user name and password given to you by your system administrator. The default user name and password is admin.
3. After successful login, you will be presented with the Application Dashboard.

Accessing Online Documentation

Koverse provides online documentation for many of the applications displayed on the Dashboard. You may need to refer to this documentation during the course of this tutorial so it's a good idea to become familiar with how to navigate the online help.

1. Click the "Help" link in the Navigation Bar at the top of the Koverse Dashboard.
    - A new browser window or tab will open, and you will be presented with the following sections of the online documentation
        - **Help**

- Introduction
- Koverse User Guide
- Koverse Administration Guide
- Koverse Operations Guide
- Koverse Developers Documentation
- Use Cases
- Tutorials

2. Click the "Table of Contents" header on the left.

    - Here you can see the full contents of the Koverse documentation.

3. Close the help tab or window in your browser. You should be back to the Applications Dashboard.

Manage your User Account

1. Click the "Account" link in the Navigation Bar at the top of the Koverse Dashboard.
2. Click the "My Account" link.

    - Here you can change your password, edit your account details, and view your permissions and credentials.

3. Click the "My Account" page title on the left side of the menu bar.

    - This menu shows the same list of Applications that are available on the Applications Dashboard.

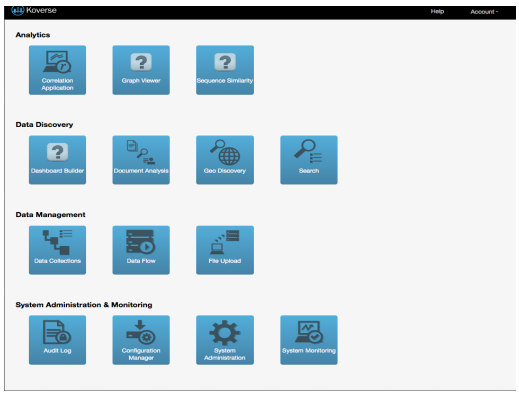
4. Click "Applications Dashboard"

Use those two steps to return to the Applications Dashboard at any time.

Koverse Applications

Koverse hosts many applications that help you manage your organizations data. The applications made available to you depend on the permissions granted by your system administrator.

Example Application Dashboard:



Data Collections

Data Collections are the basic container for data in Koverse. You can think of them like tables - but every record in a data collection can be completely unique in structure.

Create a Data Collection

The following set of instructions will show you how to create a **New Data Collection**.

1. Click the "Data Collections" application icon under the 'Data Management' application group.
2. Enter the name "BMW Z4M Cars" into the "data collection name" text box above the table.
3. Click the "Create Data Collection" button.

You have just created an empty Data Collection!

Edit a Data Collection

1. Click the newly created Data Collection name in the table.
    - You will be taken to the "Data Collection Details" for that data collection.
    - Note that no data has been imported into this Data Collection yet, so there are no details to view.
2. Click the "Configuration" tab.
    - Here you see the basic configuration of a data collection, including it's name, tags, and processing options.
3. Click the **Edit Data Collection** button.
4. Change the name of the Data Collection to "BMW Z4M Automobiles".
5. Click the "Save" button.

Share a Data Collection

When a Data Collection is created, it's presence, content, and configuration are visible only to the user that created it. That user must grant the permissions to other groups before other users will be able to use it. Note that this means even system administrators can not see the Data Collection until it is shared.

1. Click the "Permissions" tab in the Data Collection details page.
    - Here you will see a list of groups, which are defined by your system administrator - or your organizations user directory.
2. Click the check-box in the "Read" column for the "Everyone" row.
3. Click "Save Group Permissions"

Your Data Collection is now visible and readable for all authenticated users.

Upload Data from a Web Browser into Data Collection

Koverse provides users the ability to simply upload data files from their web browser. Koverse automatically parses several formats of files including CSV, XLS, XML, JSON, plus many other formats. Koverse even parses unknown files for basic information - and makes them available for retrieval or further processing.

1. Click the link below to download an example CSV file. Be sure to note where it is saved on your computer (usually in your Downloads folder).
    - http://data.koverse.com/bmw/z4mproduction.csv

2. Click the "Collection Details" link - which has a down arrow to the right of it, in the black menu bar at the top of the Data Collection details page.
3. Click the "File Upload" application link.
4. Drag the z4mproduction.csv file, which you downloaded above, into the box in the screen.
    - You can optionally drag more than one file at a time into the box.
5. Allow the file upload to complete.
    - You can optionally drag additional files onto the screen.
6. Select the "BMW Z4M Automobiles" from the Data Collection drop down at the bottom.
7. Click "Start Import". This will import all of the files you staged into the selected Data Collection.

The import has started, and will take a few moments to complete - based on the current work load and file sizes.

View Field Statistics of a Data Collection

After your file upload import has completed, Koverse evaluates the newly imported data and makes available to you a list of the fields and value type present in your data.

1. Click the "File Upload" page title in the menu bar.
2. Click the "Data Collections" application link from the drop down.
3. Click the "BMW Z4 Automobiles" link in the table.
    - You have now returned to the BMW Z4M Automobiles Collection details page. If the field statistics job has completed, you will see a a chart and a list of field names.
4. Click the the "BodyType" field name - 6th from the top.
    - On the right, you will see the details of the field, including the values discovered, and in how many records the field is present.
    - Here we see that there are two body types for the Z4M - ROADST and COUPE.
    - Remember that each record in a Data Collection may contain a unique set of fields.
5. Scroll down the field list and click the InteriorColor field.
    - Here we see that there are 6 unique values, of which one which starts with LEDERAUS... is far more prevalent than the others.
6. Click the Fields tab
    - Here you see a complete table of fields and their given size, type, presence and some indexing options.
7. Find the ProductionDate field.
    - Note that it is always a Date value, is present in every record, and has an Estimated Cardinality of 682. This means that there are an estimated 682 unique production date values.
8. Find the ShortVIN field.
    - Note the checkboxes in the "Index" column. For a field to be searchable, this checkbox must be checked.

9. Uncheck the "Index" checkbox for the ShortVIN column.
10. Scroll to the bottom of the table, and click "Save Field Settings".

    - The ShortVIN column will no longer be searchable. Doing this reduces the amount of processing necessary at import time, and query time - but is generally only necessary for very large real-time data sets or field values that are extremely large.

View Sample Records of a Data Collection

1. Click the "Samples" tab at the top.
2. Scroll to the right and find the "PaintColor" field.
3. Scroll down the page to view the paint colors.

    - Note that the colors are listed in a mix of German and English. Remember that there are 6 unique values.
4. Scroll back to the top, and further to the right to find the "VIN" and "ShortVIN" columns.

    - Note that the ShortVIN values are the last 7 digits of the VIN value.
    - Note that most of the columns contain simply ones and zeros. These indicate the presence of a feature, such as an alarm system.

Now you have a very clear understanding of the contents of the data you uploaded - even though you've likely never seen or used this data before.

Search the contents of a Data Collection

Koverse allows users to search indexed Data Collections using a simple search syntax.

1. Click the "Collection Details" page title link in the menu bar.
2. Click the "Search" link.

    - Optionally, click the check box next to "BMW Z4M Automobiles" on the left. This will limit your search to that single Data Collection.
3. Type ROAD into the search text box.

    - Notice the autosuggest drop down with the value "ROADST" which starts with what you entered.
4. Click the "ROADST" value in the drop down.
5. Click the "Discover" Button.

    - The results are disabled in a table, with pagination. Notice that only records containing ROADST in the BodyType are returned.
    - Notice the record count of 3,041 in the top right. This means there were 3,041 Z4M roadsters in the data.
6. Click the three result formatting buttons that are group together at the top right below the records count text.

    - You can view the results in one of three formats, for easier viewing.
7. Click the number 2 in the page control above the table.

- You are now viewing page two of the results.

8. Enter the lowercase word roadst in the search bar and click Discover.

- Notice that the same results are returned. Based on the field index options in the collection details, your searches are case insensitive.

9. Enter the following in to the search box.

- BodyType: coupe AND PaintColor: silbergrau
- Note that the AND must be in upper case.
- The "BodyType:" portion of this search term means 'search only in the BodyType field for this value'.

10. Click Discover

- Note that 209 records were returned - meaning 209 cars are both coupes and the color silbergrau (silvery grey).

11. Change the search term AND to OR.
12. Click Discover

Note that 2,118 records were returned. This means that there are 2,118 cars of body type "coupe" or paint color silbergrau.

Import Data from an External Source into Data Collection

Koverse "import sources" can load data from external systems into a Data Collection.

1. Click the "Search" title in the menu bar.
2. Click the "Data Flow" application link.
3. Click "Add Import Source" to start defining a new source.
4. Choose "URL Source" from the drop down.
5. Enter a name of "DC Website"
6. Skip the Security Label options
7. Enter the following URL

- http://data.koverse.com/dc/Purchase_order_FY13.csv

8. Do not check 'Process files in directory'
9. Click "Add New Data Collection"
10. Enter the name "DC 2013 Purchase Orders"
11. Click "Add Source"

- Your new import source is now configured.

12. Click on the line between "DC Website" and "DC 2013 Purchase Orders" in the Data Flow diagram.

- This takes you to the import flow details for the source you just added.

13. Click the "Run" button - which is under the "Import" tab.

- Your import job is now running

14. Wait for the job to finish.

- Notice the record count, and records per second imported over time.

15. When the job has completed, click the "DC 2013 Purchase Orders" Data Collection link.

- This takes you to the Data Collection details page.

16. Review the Explore and Fields tabs.

- Notice the presence of the ZipCodes field

Running a Transform on a Data Collection

Transforms read data from one or more input data collections, and writes resulting data to an output data collection. Each Transform is a type of analytic or filter that produces usable results for apps, or user friendly searches.

Here we will use a "Pearson Correlation" transform to discover the correlation between the available options on the BMW Z4. For example, are heated seats correlated, not correlated, or anti-correlated with the presence of GPS navigation. We will use the "Correlation" application to view the results of the transform and discover the relative correlation between features.

1. Click the "Collection Details" link in the menu bar at the top.
2. Click the "Data Flow" link in the drop down.
3. Click the "Add Transform" button at the top right.
4. Choose "Pearson Correlation" from the drop down.
5. Enter the name "BMW Z4 Feature Correlation".
6. Select the "BMW Z4M Production" collection in the Input Collections.
7. Click the "Add New Data Collection" on the Output Collection line.
8. Enter the name "BMW Z4 Feature Correlations"
9. Do not enter any values in the Number Fields.

- This will cause the transform to operate over all fields.

10. Leave the flow type to automatic, and minimum seconds between jobs to its defaults.
11. Click the Add Transform button.

- At this point, if new data were added to the BMW Z4M Production data collection, the transform would be automatically run.
- For now we must manually execute the transform one time.

12. Click the line between the BMW Z4M Production and Correlations nodes in the diagram.
13. Click the "Run Transform Job".
14. Allow the job to complete.

Viewing the output of a Transform

The output of a transform is a data collection, and you could use the data collection details page to view the results. But here we will use the "Correlation" application - which is specifically designed to be used with the output of the "Pearson Correlation" that we configured above - to view the results.

1. Click the "Transform Job Details" link in the menu bar.
2. Click the "Correlation" link in the drop down.
3. Choose the BMW Z4M Production correlation from the drop down at the top.
4. Move your mouse over the correlogram on the left to view the correlation value and sample correlations on the right.

    - Bright blue means highly correlated
    - Bright red means highly anti-correlated
    - Dark means no correlation

Downloading Data Collections

Reasonably sized data collection, such as those with relatively few original records or the output of a transform that summarizes a large data collection into a much smaller data collection, can be downloaded. There is no hard limit to the records that can be downloaded, but the network connection speed and available disk space on your computer are limited.

1. Click the "Correlation" link in the top menu.
2. Click the "Data Collections" link in the drop down.
3. Click the "BMW Z4M Feature Correlations" link.
4. Click "Configuration" tab.
5. Click the "Download Collection Contents" button.

You now have the output of the data collection saved as a JSON stream file on your local computer. You can use this in external tools, or upload it to another Koverse instance.

Exporting Data Collections

Koverse enables you to export data collections to outside systems. Use this for periodic backups, or integrations for data work flows.

1. Click the "Collection Details" link at the top.
2. Click "Data Flow" link in the drop down.
3. Click the "Exports" tab at the top.
4. Click the "Add Sink" button at the top right.
5. Select the "FTP Server"
6. Enter the name "My FTP Server"
7. Enter a valid host name, port, username and password.

    - If you do not have an FTP server, enter fictional values.
8. Click "Add Sink"
9. Click the "Run Export Job"
10. Select the "BWM Z4M Feature Correlation" Data Collection

11. Enter 100000000 for the number of records per file.
    - Note that this is a maximum, and not a strict count. Even though there are fewer records in the data collection, multiple files will be created.
12. Enter the file name prefix of "bmw-z4m-feature-correlations"
13. Select the CSV File Format.
14. Leave the fields to export empty
15. Leave the character options as default.
16. Check the "Write Header Line".
    - This will ensure each file has a header line
17. Enter a relative path for saving the files.
    - Example: /bmw-z4m-data
18. Select the "GZip" compression option
19. Do not add any Export Transforms.
20. Click "Run Job"
21. Allow the export job to complete.

Your data is now exported to the FTP server. You could re-import this data into another koverse server, or use it in an external tool.

Transferring Configuration from one Koverse Instance

After configuring a Koverse instance with source, transforms, exports, and data collection you may wish to move that configuration. Here you will learn how to move the configuration.

1. Click the "Export Job Details" link in the menu at the top.
2. Click the "Configuration Manager" link.
3. Select the "BMW Z4M Production" and "BMW Z4M Feature Correlations" data collections checkboxes.
4. Change the Available Items drop down to Transforms.
5. Select the "BMW Z4M Feature Correlation" transform.
6. Click the "Download Selected".
7. Save this file to your local computer.
    - Now that you have downloaded the configuration, you can upload to another koverse server using the configuration manager application.
8. Be sure to do these next steps in another koverse server, or simply read them for reference.
9. Click the "Upload" tab in the configuration manager application.
10. Click the "Choose File" button, and select the previously downloaded configuration file.

11. Validate the contents of the configuration in the text box.
12. Click the "Upload" button.

Your configuration is now transfered. Upload the BMW Z4M data to this new koverse-server will allow you to use the correlation application.

Administrator Tutorial

> ***Note***
>
> Finish the End User Tutorial before starting this section of the tutorial. In this tutorial you will learn how to manage users and groups, define permissions, manage addons, monitor system status, and view user activity.

Viewing System Health

1. From the Applications Dashboard, or in the applications menu, click the "System Monitoring" link.
2. See the colored squares along the top. Click on a few to view their statuses.
3. See the Ingest and Query charts in the middle. These will show activity only when jobs are running.
4. See the Job progress bars below the data charts.
5. See the resource details at the bottom.
    - The Distributed Storage shows the remaining disk space and the health of the file system.
    - The Map Reduce section shows the available compute resources and their usage.
    - The Data Store section shows the data base resources and health.

Viewing the Audit Logs

1. From the Applications Dashboard, or in the applications menu, click the "Audit Log" link.
2. Enter the search term login and click search.
3. Click the "Older" and "Newer" links to navigate back in time.

Managing Users and Groups

1. From the Applications Dashboard, or in the applications menu, click the "System Administration" link.
2. Click the "Add User" button.
3. Enter an email address and name.
4. Click the "Add User" button.
    - The user will be in the "Everyone" group.
    - The user will receive an email with a randomly generated password.
5. Click the "Edit User" button on the user to be edited.

6. Check the groups you desire, or change a user attribute.
7. Click the "Save" button.
8. Click the checkbox to the left of the user you created.
9. Click the "Delete Selected Users" button.
10. Click the "Delete" button.
11. Click the "Groups" tab at the top.
12. Click "Add Group"
13. Enter the group name "Test"
14. Optionally check the "Add to All New Users" check box"
15. Check the "Manage Data Collections" check box.
16. Click the "Add Group" button.
17. Click the "Edit Group" button on the "Test" group row.
18. Make changes as you see fit.
19. Click the "Save" button.
20. Click the checkbox to the left of the "Test" group.
21. Click the "Delete Selected Groups" button.
22. Click the "Delete" button.

Managing the System Settings

System settings are for configuring the Hadoop and data store settings.

1. Click the "System" tab in the "System Administration" application.
2. View but do not change these system settings.

Managing Lock Down Mode

Lock down mode is used stop all inbound and outbound data streams from the system, and to keep users from accessing data. Use lock down mode when data leaks such as incorrect user and group permissions are discovered.

1. Click the "Lock Down" button.
2. Click the "Enable LockDown Mode" button.
3. Click the "Unlock" button.
4. Click the "Disable LockDown" button.

Managing Addons

Addons are created by developers and installed by system administrators. Addons expand the available sources, transforms, sinks and applications for users.

1. Click the "Add-Ons" tab in the "System Administration" application.
2. If you had an addon to deploy, you would use the Install addon section to do so.

3. Click the koverse-addon-geo-discovery... link.
    - This addon detail view is shown directly after uploading an addon.
4. See the details about the addon, and the download and disable addon buttons.
5. Click the tabs at the top to view the contents of this addon.

Managing Applications

Developers can create applications for Koverse that are delivered via addons. System administrators can grant permissions for users to access applications, and deploy instances of applications.

1. Click the "Applications" tab in the "System Administration" application.
2. Click the "Geo Discovery" link.
3. See the details of the Geo Discovery application. Note the Change and Delete Application buttons.
    - Do not change or delete any settings.
4. Click the "permissions" tab at the top.
    - Here you can define permissions for groups to access and manage this application.
5. Click the "parameters" tab at the top
    - Here you can manage any configuration parameters for the application.
    - The geo discovery application does not have any permissions to manage.

Managing API Tokens

System Administrators can create API Tokens for use by developers.

1. Click the "API" tab in the "System Administration" application.
2. Click the "Add API Token" button.
3. Enter a name for the API Token, such as "Test Token".
4. Check the "Administrators" checkbox.
    - This gives the API token the same permissions as an administrator.
5. Click the "Create Token" button.
    - The new token string is now listed in the table below.
6. Click the "Edit Token" button in the newly created row.
7. Change the token name to "Test Token Renamed".
8. Click the "Everyone" group checkbox.
9. Click the "Update Token" button.
    - The token string stays the same, but the permissions and name changes.
10. Click the checkbox to the left of the "Test Token Renamed".

11. Click the "Delete Selected API Tokens" button.
12. Click the "Delete" button.

Developer Tutorial

> ***Note***
>
> Finish the End User and Administrators tutorials before beginning this Developer tutorial. In this tutorial you will learn how to create a custom Import Source, Transform, and Application using the Koverse SDK Project.

Setup your Koverse SDK based Project

The Koverse SDK is available as a Maven archetype, and comes with complete examples of sources, transforms, sinks and apps. Here we will create a local copy of the Koverse SDK project.

1. Download and install Apache Maven for your operating system.
    - http://maven.apache.org
    - Version 3.0.5 recommended
    - Eclipse and Netbeans both have Maven integrations.
2. Create a directory on your local computer to store the project.
3. Download the Koverse SDK Project from http://github.com/Koverse/koverse-sdk-project
    - Be sure to switch to the branch for your version of Koverse
    - Use the download button, or fork if you're familiar with Git.

Add your API Token to your Maven Settings.xml

This will allow you to build, test, and deploy your addon in a single maven command.

1. Create an API Token in the administrators group.
    - See the API Token section of the Administrators Tutorial for steps.
2. Add the following profile to your ~/.m2/settings.xml file
    - Note the koverse.serverurl property may need to start with HTTP or HTTPS

```
<settings>
        <profiles>
                <profile>
                        <activation>
                                <activeByDefault>true</activeByDefault>
                        </activation>
                        <properties>
                                <koverse.apitoken>API TOKEN HERE</koverse.apitoken>
                                <koverse.serverurl>http://<KOVERSE-SERVER-HERE>/Koverse</koverse.serverurl>
                        </properties>
                </profile>
```

```
        </profiles>
</settings>
```

Test your SDK Project

You are now ready to perform a test build and deploy.

1. Use the following command from the project directory
    - `mvn clean package koverse:deploy`
2. Check that there were no errors.
    - If there is an error deploying, ensure that the koverse.serverurl and apitoken settings in your ~/.m2/settings.xml are correct.
3. Check that the destination Koverse server received and has the addon installed.
    - In "System Administration" app under Addons.

Explore the project structure

The Koverse SDK example project includes working source code examples, and build structure. You'll simply delete the unnecessary files and modify the ones you want to fit your use case.

1. See the Java classes in /src/main/java/...
    - There are sources, transforms, sinks, application definitions, import time transforms, and a query client.
2. See the classesToInspect file. An example is shown at /src/main/resources/classesToInspect.example
    - This important file tells the Koverse server which classes to evaluate when the addon is uploaded.
    - Rename and edit this file after you change the Java classes.
    - List all of your source, transform, sink, application, importTransform, securityLabelParser classes.
    - Do not list dependency classes.
3. See the unit testing code in /src/test/java/...
    - The fastest way to debug will always be using the provided test runners, as shown in this code.
    - These tests will also be run during the maven test phase.
4. Perform a `mvn clean package`, and then view the `/target` directory.
    - The addon file is named <artifactId>-<version>.jar in the `/target` directory

Create a custom Simple Source

Here we will create a source that returns a single record that contains two fields with user defined values.

1. From the project directory, open the `/src/main/java/your/package/MyCustomSource.java` file.
    - The source code in here is a fully functional, but extremely minimal Koverse import Source.

2. Note that this source extends SimpleSource.
    - Simple source is for simple single connection "pull" models - like databases or twitter streams.
    - We will cover more advanced multi-connection sources later.
3. Refactor the name of the class to TutorialSource. Change the package if you wish as well.
4. Review the methods in MyCustomSource, and see their JavaDocs more information about their use.
    - **Set the following**
        - name = "Tutorial Simple Source"
        - description = "Source from a tutorial"
        - **sourceTypeId = "tutorialSource"**
            - Needs to be universally unique for your source. Change as necessary.
        - version = "1.0.0-alpha1"
        - isContinuous = false
5. Give the source two parameters, one string and one number.
    - Parameters are the user defined configuration for the source.
    - See the getSourceParameters() method.
    - **First Parameter**
        - parameterName = "stringParameter"
        - displayName = "String"
        - type = Parameter.TYPE_STRING
        - defaultValue = null
    - **Second Parameter**
        - parameterName = "integerParameter"
        - displayName = "Integer"
        - type = Parameter.TYPE_INTEGER
        - defaultValue = "100"
6. Edit the connect() method to read the parameter values.
    - Be sure to cast the values with error checking.
    - Note that the getSourceParameters()
7. Edit the getNext() method to return only one record, which contains the two parameter values given.
    - The getNext method is called repeatedly until it returns null.
    - This instance getNext() is called, keep state so the second getNext() returns null.

8. Leave the disconnect() method empty.
    - Normally this is where you would clean up connections, but this example has none.
9. Edit the `/src/main/resources/classesToInspect` file, list only your new TutorialSource class by package name.
    - Example: `com.company.project.TutorialSource`
10. Find the `/src/main/test/java/MyCustomSourceTest.java` file and rename it `/src/test/java/TutorialSourceTest.java`.
11. Edit the TutorialSourceTest.java file to validate that your source is working correctly.
    - Add the string and integer parameter values.
    - Change the assert record count to equal 1.
    - Add an assert to evaluate the values of the string and the integer.
12. Compile and upload your new addon
    - `mvn clean package koverse:deploy`
    - The test from above will be run, and will fail the build if they do not pass.
13. Test your addon in Koverse
    - In the Data Flow application, add an import source that uses the "Tutorial Source"
    - Run an import into a new data collection.
    - View the data collection details after the import has been run, to view the record.

Create a custom ListMapReduceSource

ListMapReduce based sources allow many connections to be used to import data in parallel. This is done by first producing a list of the items to processed - for example a list of files. And then each item is processed by an individual task in MapReduce. Note only the Map phase is used, the reduce phase is not available.

ListMapReduce sources should be used when the outside data source an handle many connections well, and when the data import logic is parallelizable.

Here we will make a list based source that will allow the user to define how many mappers are used, and how many records to return from each mapper. You will give the records any values that you like.

1. Rename the `/src/main/java/com/koverse/foo/MyCustomListSource.java` class to TutorialListSource.
2. Edit the file and apply the following settings.
    - name = "Tutorial List Source"
    - description = "List based Source from a tutorial"
    - **sourceTypeId = "tutorialListSource"**
        - Needs to be universally unique for your source. Change as necessary.
    - version = "1.0.0-alpha1"
    - isContinuous = false

3. Give the source two parameters, one string and one number.
    - Parameters are the user defined configuration for the source.
    - See the getSourceParameters() method.
    - **First Parameter**
        - parameterName = "mapperCount"
        - displayName = "Mapper Count"
        - type = Parameter.TYPE_INTEGER
        - defaultValue = "3"
    - **Second Parameter**
        - parameterName = "recordsPerMapper"
        - displayName = "Records Per Mapper"
        - type = Parameter.TYPE_INTEGER
        - defaultValue = "100"
4. Edit the initialize() method to pull the user defined parameter values and place them in local variables.
    - Be sure to use a safe cast to an integer
5. Edit the enumerateList() method to produce strings of numbers from 1 to the user defined MapperCount
    - The list returned should have N elements, where N is the number from the mapperCount parameter.
6. Edit the recordsForItem(String item) method to produce the user configured number of elements for each item.
    - Ideally you should return an Iterable that uses a streaming method of return values.
    - Even though a Collection (List) is an Iterable, no not return a Collection (List). This is because Collections require that all values are in memory - and that's not scalable.
7. Note the close() method
    - This would normally be used to cleanup connections, etc.
8. Edit the `/src/main/resources/classesToInspect` file, list only your new TutorialSource class by package name.
    - Example: `com.company.project.TutorialSource`
9. Find the `/src/main/test/java/MyCustomListSourceTest.java` file and rename it `/src/test/java/TutorialListSourceTest.java`.
10. Edit the TutorialListSourceTest.java file to validate that your source is working correctly.
    - Add the mapperCount and recordsPerMapper parameter values.
    - Change the reference to the TutorialListSource class in the TestRunner line.
    - Change the assert record count to equal 1.
    - Add an assert to evaluate the number and contents of the values returned.

11. Compile and upload your new addon
    - `mvn clean package koverse:deploy`
    - The test from above will be run, and will fail the build if they do not pass.
12. Test your addon in Koverse
    - In the Data Flow application, add an import source that uses the "Tutorial Source"
    - Run an import into a new data collection.
    - View the data collection details after the import has been run, to view the record.

Use Pig in a Transform

Koverse has a "Pig Transform", that uses Apache Hadoop Pig script to execute a transform. Pig is useful for joining, group, filtering, and performing basic aggregations on data collections.

Here you will enter a very simple "copy records" Pig script. For more information about Pig, and examples, see http://pig.apache.org

1. Navigate to the Data Flow App
2. Click the "Add Transform" button
3. Select "Pig Transform" from the list.
4. Enter the name "Pig Z4M Copy"
5. Select the "BMW Z4M Production" input data collection
6. Create a "BMW Z4M Production Copy" output data collection.
7. Do not enter a UDF
8. Enter the following Pig script.
    - `BMW_Z4M_Production_Copy = FOREACH BMW_Z4M_Production GENERATE *;`
9. Leave the default flow settings.
10. Click the "Add Transform" button
11. Click the line between "BMW Z4M Production" and "BMW Z4M Production Copy"
12. Click the "Run Transform" button
13. Allow the Transform to complete.
14. Click the output data collection name to view the results.

Understanding Map Reduce

Map-Reduce is a two phase framework for parallel processing with many useful applications. The Map phase takes an arbitrarily long list of input values, and emits zero,one, or more key/value pairs to the Reduce phase. The reduce phase taskes a key and a list of values for that key, and can output key/value pairs. In either of these phases, you can write out Koverse records. There is a Combine phase available between the Map and the Reduce phase for the purpose of reducing the network congestion between the map and reduce phases, by collapsing associative aglorithm output immediately after the Map phase output - befor enetwork transfer.

In concrete terms, the Map phase is used for extracting features from input data, and the reduce phase is used for summarizing those extracted features. For example, in the map phase the MyCustomTransform in the Koverse SDK extracts each word from a paragraph.

Then in the Combine and Reduce stage, the word counts are summarized via summation. For more information about Map Reduce patterns, search the web for "MapReduce Algorithms".

Create a custom Transform

Koverse Transforms are implemented as Map-Combine-Reduce logic in one or many steps. Koverse makes Map Reduce much easier, by easily defining many map and reduce stages in series. The first stage reads records, the subsequent stages pass key/value pairs. Some familiarity with Map Reduce is necessary.

Here we will alter the MyCustomTransform to create a "word phrase" counter, that reports the counts of series of words.

1. Rename the `/src/main/java/com/koverse/foo/MyCustomTransform.java` class to TutorialPhraseCountTransform.
2. Edit the file and apply the following settings.
    - name = "Tutorial Phrase Count"
    - description = "Counts the phrases of words"
    - **jobTypeId = "tutorialPhraseCount"**
        - Needs to be universally unique.
    - version = "1.0.0-alpha1"
    - isIncrementalProcessingSupported = false
3. Give the transform two parameters.
    - Parameters are the user defined configuration for the transform.
    - See the fillInParameters() method.
    - **First Parameter**
        - parameterName = "textFiled"
        - displayName = "Text Field"
        - type = Parameter.TYPE_COLLECTION_FIELD
    - **Second Parameter**
        - parameterName = "phraseWordCount"
        - displayName = "Phrase Word Count"
        - type = Parameter.TYPE_INTEGER
        - defaultValue = "3"
4. Ensure that the fillInStages() method adds one AbstractKVRecordMapperStage class, one AbstractCombinerStage, and one AbstractReducerStage - in that order.
5. Edit the inner static class MyCustomMapStage.setup() method to read both parameters from the context.
    - Use type safe checking to ensure valid values.
6. Edit the MyCustomMapStage.getMapOutputKeyClass() and .getMapOutputValueClass() to ensure they are Text and IntWritable respectively.

- These define the types of keys and values that are output in the map phase.

7. Edit the MyCustomMapStage.map() method to output keys that contain exactly the number of words defined by the user in the phraseWordCount configuration.
    - For example, the phrase "The fox is fast" contains two 3 word phrases. "The fox is" and "fox is fast".
    - Output each phrase as the key, and the number one as the value.
8. Edit the MyCustomCombinerStage.combine() method to ensure that it sums all of the received values for a key.
    - The combiner runs on the same machine as the mapper, and is used to reduce the amount of network traffic.
    - Combiners can only be used if the algorithm is both associative and commutative (see Google for more info).
9. Note the MyCustomReduceStage.setup() should be empty.
    - Additional parameter reading or job setup could be performed here.
10. Edit the MyCustomReduceStage.getMapOutputKeyClass() and .getMapOutputValueClass() methods. They should both return NullWritable
    - NullWritable is preferable for the last Transform stage - in which we expect to output no data to Hadoop.
11. Edit the MyCustomReduceStage.reduce() method.
    - Change the Koverse record key field name to "phrase".
    - Note the use of the getStageContext().writeRecord(r) method. This is out records are output to koverse.
12. Edit the `/src/test/java/MyCustomTransformTest.java` class to match the expected behavior of the TutorialPhraseCountTransform.
    - Change the input data to create a few records with unique phrases.
    - Change the Assert lines to evaluate the record count and phrases expected.
    - Change the TransformTestRunner.runTest() method to use the TutorialPhraseCountTransform class.
13. Compile and upload your new addon
    - `mvn clean package koverse:deploy`
    - The test from above will be run, and will fail the build if they do not pass.
14. Test your transform.
    - Create a new Data Collection named "Text"
    - Using the file upload app, upload a .txt file that contains the phrase "The fox is fast" to the Text data collection.
    - **In the Data Flow App add your new transform:**
        - Input Data Collection: Text
        - Output Data Collection: Phrase Count
        - Text Field: body
        - Phrase Word Count: 3
        - Flow Type: Automatic
        - Minimum Seconds Between Jobs: 60
    - Click Add New Transform

- Run your new transform by clicking on the line between Text and Phrase Count data collections.
- Click "Run Transform"
- Allow the transform to run. Then view the output of that transform.

Creating a Custom Koverse HTML/JS Application

Koverse applications are written in HTML and JS, and are delivered via addons. The Koverse SDK Project, that you pulled from a maven archetype above, has an example application. Here we will edit this application to perfom a few basic queries of data collections in Koverse.

1. View the Koverse Javascript API at http(s)://<KOVERSE_SERVER>/Koverse/docs/javascript/
2. Open the `/src/main/java/MyCustomApplication.java` class.
    - This class defines the presence of an application in this addon.
    - **Edit the class as follows:**
        - Rename the class to TutorialApplication
        - applicationId = "tutorialApplication"
        - displayName = "Tutorial Application"
        - defaultCategoryName = "Tutorials"
        - version = "1.0.0-alpha1"
        - No parameters
        - autoDeploy = true
3. Rename the `/src/main/resources/apps/mycustomapp` folder to match your new applicationId
    - `/src/main/resources/apps/tutorialApplication`
4. Edit the `/src/main/resources/classesToInspect` file.
    - Ensure that the TutorialApplication file is listed in that file.
5. Open the `/src/main/resources/apps/tutorialApplication/index.html` file.
    - Remove the collection list section.
    - Add a text input box with an id of "searchTextInput"
    - Add a button with an id of "searchButton"
    - Add a div with an id of "searchResults"
6. Open the `/src/main/resources/apps/tutorialApplication/js/index.js` file.
    - Store the results of Koverse.getDataCollections method in an array.
    - Add a jQuery event listener for when the searchButton is pressed.
    - Add logic that reads the value from the searchTextInput box
    - **Use the Koverse.performRecordQuery method to perform a search.**

- Method signature looks like Koverse.performRecordQuery(query, dataCollectionIdsCommaSeparated, callBack, numRecords, offset, fields)
- Example: Koverse.performRecordQuery("text", "dataCollectionId1,dataCollectionId2", searchResultHandler, 100, 0, [])

- **Show the count and the contents of the records returned**

- Use the response.records array
- Be sure to check the response.success boolean to ensure the request was successful.

7. Compile and upload your new addon

- `mvn clean package koverse:deploy`
- The test from above will be run, and will fail the build if they do not pass.

8. Test your application by clicking on it in the Application dashboard.
9. When you are ready, set the application permissions for access by others.

- Navigate to the System Administration App
- Click the Applications Tab
- Click the Permissions tab
- Click the Edit Permissions button
- Click the Use check box in the Everyone row.
- Click Save permissions.

Creating a custom Record Based Export Sink

Record Based sinks export to record based systems, such as databases or other non-file based systems.

1. Open the `/src/main/java/com/koverse/foo/MyCustomRecordSink.java`
2. Edit the class as follows:

- Rename the class to TutorialSink
- sinkTypeId = "tutorialSink"
- name = "Tutorial Sink"
- version = "1.0.0-alpha1"
- getParameters = return one string parameter named "test"
- getJobParameters = return one string parameter named "job test"
- executionMethod = Sink.ExecutionMethod.MapReduce;

3. Rename the MyCustomRecordWritter to TutorialRecordWritter
4. Change the RecordWritter implementation to count records written.

- Normally, you will use a third party API to connect and write records to an external system.

5. Open the `/src/test/java/MyCustomRecordSinkTest.java`
    - Change the test to use the TutorialSink
    - Change the test to output to records
    - Change the Assert to test number of records written.
6. Compile and upload your new addon
    - `mvn clean package koverse:deploy`
    - The test from above will be run, and will fail the build if they do not pass.
7. Test your sink in the Koverse Data Flow app

Creating a custom File Based Export Sink

The FileBasedSink abstract class can be extend to create custom file sink destinations. These sinks simply open a file output stream. They do not define a file format.

1. Open the `/src/main/java/com/koverse/foo/MyCustomFileSink.java` class
2. Edit the class as follows
    - Rename the class to TutorialFileSink
    - name = "Tutorial File Sink"
    - sinkTypeID = "tutorialFileSink"
    - description = "Sink from a Tutorial"
    - version = "1.0.0-alpha1"
    - fileSinkParamaters = one string parameter named "test"
    - fileSinkJobParameters = one string parameter named "jobTest"
3. Change openOutputStream method to count the number of times it is opened.
4. Open the `/src/test/java/MyCustomFileSinkTest.java`
    - Change the test to use the TutorialSink
    - Change the test to output to records
    - Change the Assert to test number of records written.
5. Compile and upload your new addon
    - `mvn clean package koverse:deploy`
    - The test from above will be run, and will fail the build if they do not pass.
6. Test your sink in the Koverse Data Flow app

Congratulations!!!

Congrats on finishing the Koverse Developer Training. As a reward, please contact support@koverse.com and ask for your special Koverse cloud instance access.

Index