Educational Secure Data Service

Runbook

Document History

Version	Date	Revised by	Description

Table of Contents

This page intentionally left blank.

1 Preparing for a Deployment of the Educational Secure Data Services

1.1 Introduction

Educational Secure Data Services (ESDS) is a multi-tenant, transaction-based database. The ESDS is run either as Software as a Service (SaaS) where ESDS manages the operation or as an Operator model, where a provider opts to maintain their own instance of the software.

An ESDS managed deployment offers a secure, cloud-hosted data store designed for states and school districts. An Operator instance is responsible for maintaining the service with the same expectations for performance, security, and availability as ESDS has for a managed instance. Operators are encouraged to align with an established level of technical and business standards to offer stakeholders a secure and optimal experience and are expected not to change the provided ESDS APIs so that applications can use them no matter which instance of the service their customers opt to use.

Whether a SaaS or an Operator model, the ESDS securely maintains data about state and district's organizational structure, schools, and employees, as well as information on student enrollment, biographical and achievement data. In implementing ESDS solution, states and districts load data to the SDS data store through either bulk ingestion or the API.

The goal of ESDS is to provide a level of data interoperability that allows states, districts, schools and teachers to tailor resources and learning applications to their local and individual needs securely by providing an API. The Secure Data Service API makes the process of sharing individual learning tools and strategies easier and more efficient. Overcoming these data interoperability barriers is the ultimate objective of ESDS and are core to enabling the products and strategies that drive personalized learning in the classroom.

This Runbook provides requirements, recommendations and procedures for installing, configuring, running, and troubleshooting the ESDS. Additionally, this Runbook should be used to understand the practices and procedures for the efficient use of ESDS.

1.1.1 Scope

The ESDS Runbook provides all processes and tasks required to install, configure, run and troubleshoot the ESDS system. This guide covers the instructions and supporting information you need to deploy an ESDS production environment.

1.1.2 Audience

This guide is intended for system operators and system administrators responsible for specific tasks related to the deployment of the ESDS, third party components and applications.

1.1.3 Dependencies, Assumptions & Constraints

This section provides the known dependencies, assumptions and constraints of deploying the ESDS. A successful deployment is dependent on the positive installation of third party components, SDS core components and applications.

It is assumed that the knowledge and skill-levels required by an IT Administrator or Operator in order to successfully deploy the ESDS include 3 to 5 years of Linux experience as well as familiarity with load balancing and distributed file systems.

Successful implementation of the ESDS is constrained by the regional or local education agency fully identifying all requirements for installation. This input is dependent on the participation of individuals from the stakeholder groups who are empowered to identify and agree on these requirements.

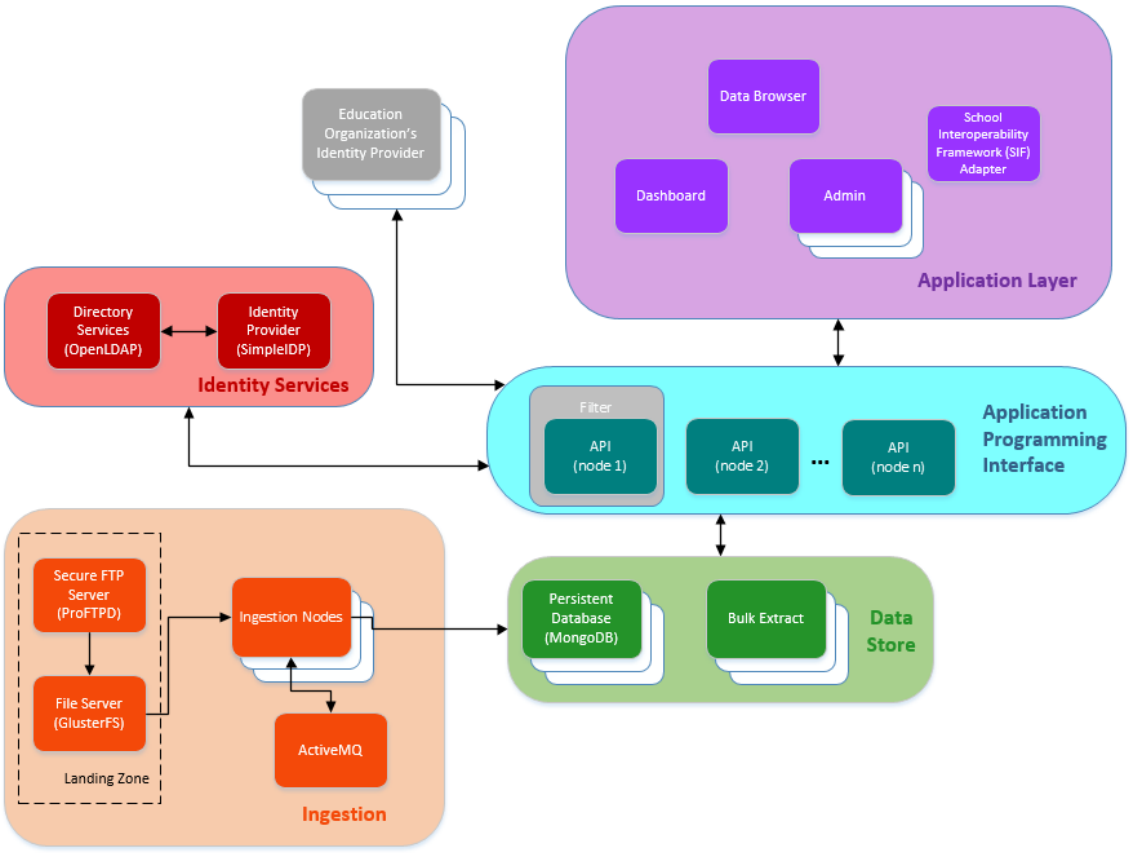
It is assumed that the target audience of this guide has experience in the following areas:

- Installing and configuring enterprise software.
- Working on a UNIX command line.
- Installing or managing software and services on a Linux system.

1.2 SDS Architecture

This section defines the ESDS architecture and the hardware and software required for an SDS deployment.

ESDS technology is logically divided into a series of subsystems that serve specific purposes in the infrastructure. The diagram below provides a look at the subsystems, their components, and how information flows between the subsystems and components.



Major components shown:

- **Applications** - All web-based ESDS applications as well as any third-party web or mobile applications that an education organization adds to its ESDS implementation. A user's defined role determines the access level to applications.
- **Application Programming Interface** - Applications that use the API to interact with the ESDS. It consists of one or more nodes hosting the ESDS RESTful API.
- **Identity Services** – Identity management solution for ESDS. SimpleIDP is the identity provider for ESDS administrator accounts. SimpleIDP is backed by openLDAP for account storage.

- **Data Store** – ESDS database that consist of educational data from the state and local education agencies plus other data necessary for ESDS operation. This also includes the Bulk Extract service which can be deployed to provide daily extracts to authorized applications in the event that the API is not suitable.
- **Ingestion** – Infrastructure required to add large amounts of data in bulk to the ESDS. This includes an SFTP server to allow ingestion users to connect and upload files.

The ESDS architecture is a scalable deployment of Linux servers. These servers fulfill the roles of API nodes, ingestion processors, application hosts, data store nodes, and background service hosts. Each part of the system can be horizontally scaled to support the high-levels of concurrency and large data sets that the SDS manages.

Two of the most critical aspects for a system administrator to consider are security and scalability. By segmenting the infrastructure tiers you are able to provide greater security. In addition, a multi-tiered environment allows for ease of scalability.

The following guidelines provide hardware and software requirements to execute a full scale deployment of ESDS that will support multiple customers at scale.

1.2.1 Hardware

Hardware needs will vary depending on the volume of data and number of students your program aims to support. Maintaining an efficient system will require the use of numerous physical or virtual servers of a caliber that would meet typical business operations for handling large datasets and frequent, concurrent connections by external parties via the internet. Operators are expected to have N+1 redundancy in an active-active configuration for all servers, as well as appropriate firewall, intrusion detection, and load balancing technologies.

An Operator can expect to provide hardware to enable and maintain the following software components of the ESDS service offering:

Database Servers

MongoDB

LDAP

Application Servers

API

Data Browser

Dashboard

Admin

SimpleIDP

Data Ingestion

Landing zone

Ingestion

Bulk Extract

ActiveMQ

GlusterFS

1.2.2 Software

ESDS utilizes a variety of open source software in the creation of the SDS. Best practices should be followed with the installation and configuration of these software components. Any specific configuration necessary to stand up the SDS will be noted in the software specific sections of this document.

An Operator can expect to install the following software to enable and maintain the ESDS service offering:

MongoDB - Document database used to store education data and system configuration

ActiveMQ - Messaging system for asynchronous processing

GlusterFS - Distributed file system for batch-processing XML ingestion files

OpenLDAP - Directory server for configuration of system-wide tools and administrators

Tomcat - Java application server for API and other applications

Nginx – Web Server used with Ruby on Rails applications

Ubuntu 12.04 – Operating system that has been tested with the SDS

1.3 Platform Sizing

The table below provides a summary of platform sizing details by server use case.

Server Role	Minimal Servers (no redundancy)	Recommend Size
Portal	1	Small
Portal MySQL	1	Small
API	1	Small
Dashboard	1	Small
Databrowser	1	Small
Admin	1	Small
SIDP	1	Small
MongoDB Config Servers	3	Medium
MongoDB Servers (1 shard)	3	Extra Large
Ingestion	1	Large
Ingestion ActiveMQ	1	Small
GlusterFS	1	Medium
Bulk Extract	1	Large
LDAP	1	Small
Landing Zone	1	Small
Search-Indexer	1	Small
ElasticSearch	1	Medium

Instance Sizing	Cores	Memory (Approximate)
Small	2	8GB
Medium	4	32GB
Large	8	64GB
Extra Large	32	248GB

2 Deploying Components

2.1 Cryptographic Keys

This section describes how to generate the cryptographic keys used to encrypt data in two locations:

- PII in the data store
- Sensitive information within the config files

Important! The JCE Unlimited strength policy must be installed on the JVM or you may get the following exception when encrypting or decrypting the password:

```
java.security.InvalidKeyException: Illegal key size or default parameters
```

To install the JCE Unlimited Strength download the appropriate file below and follow the included readme file for setup instructions.

Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6

Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7 Download

There are two different encryption keys:

- A 256bit AES secret key used by the API and Ingestion servers to encrypt data in the SDS and used by SimpleIDP and Bulk Extract for read operations. This key must be the same across all services and servers that require reading or writing directly from MongoDB.
- A private key is used by the SimpleIDP to sign SAML assertions. An x509 certificate generated using this private key is then utilized by the API to verify the integrity of the SAML assertions from the SimpleIDP.

There are two types of Java keystores that contain this information.

- A keystore used to store the AES secret key and the SimpleIDP's private key.
- A truststore used to store a trusted x509 certificate that is utilized by the API to verify the integrity of SAML messages sent from an IDP (such as the SimpleIDP) along with the client certificates to support bulk extract operations.

2.2 Configuration Files

The configuration files in the table below are used to configure the various components of the platform.

Important! These configuration files must be maintained so that all app servers that depend on these config file are running the same version.

Configuration File	Component
sli.properties	SimpleIDP, API, Ingestion, Dashboard, BulkExtract
config.yml	Admin
config.yml	DataBrowser

2.3 Operating System

It is strongly recommended that you deploy the ESDS on Ubuntu 12.04 LTS as this is the operating system that has been tested against the SDS. It is recommended that this operating system also have the appropriate security hardening established to reduce the risk of compromise.

2.4 Application Servers

The SDS platform requires two types of application servers on which the various components are then deployed.:

- Tomcat with Java 7
- Nginx with Ruby 2.0.0

The following sections cover the procedures for deploying these servers as well as any specific configuration required for the components to function properly.

2.4.1 Tomcat

2.4.1.1 Requirements

The requirements include the following:

Both JDK 7 and Tomcat 7 have been tested with the ESDS.

Important! Ensure that the Tomcat installation does not run as root.

2.4.1.2 Configuration

Confirm the following configuration parameters are set:

sli.trust.certificates=$TOMCAT_HOME/

2.4.2 Rails

Note: It is recommended that you install Ruby 2.0.0 via RVM as this provides several benefits such as making future upgrades easier to install as well as ensuring stability of the system by leaving Ruby in place.

1. Install Ruby 2.0.0.
2. Install Phusion Passenger gem.
3. Install Nginx and compile in Phusion passenger support.

Important! It is highly recommended that you ensure the Tomcat installation does not run as root.

No application specific configuration is required.

2.4.3 Bulk Extract Server

2.4.3.1 Setting Up Bulk Extract Files

Use the following steps to install and configure the bulk extract service:

Note: Based on your ESDS setup, be sure that the bulk extract service (installed here) has permission to both read and write to the new destination directory and that the REST API service has permission to read the directory.

1. Create the directory to serve as the destination for extracted data. Default configuration sets this to be /bulkextract/extracts, which is in the GlusterFS volume created at /bulkextract:

   ```
   mkdir /bulkextract/extracts
   ```

2. Create the install target for the bulk extract files /opt/bulkextract:

   ```
   mkdir /bulkextract/extracts
   ```

3. Retrieve bulk-extract-scripts.tgz from the files provided by ESDS and copy them to the new bulk extract server.
4. On the bulk extract server, extract bulk_extract.tar.gz to the newly-created /opt/bulkextract directory.
5. Confirm that the following contents now reside in /opt/bulkextract/:
   - Three directories: config, scripts, and target
   - Inside the target directory, a file named bulk_extract.tgz that contains the compiled binary objects used to run the bulk extract
   - Inside the scripts directory, two files named local_bulk_extract.sh and schedule_bulk_extracts.sh; local_bulk_extract.sh is used by scheduled extraction jobs created with schedule_bulk_extracts.sh.
6. Update sli.properties to include the entries.
7. Using your preferred text editor, edit schedule_bulk_extracts.sh so that it has the appropriate tenant names to include for bulk extraction.
   These must line up with one or more of the tenant names that ESDS administrators add to the system. Each represents a level of the education organization hierarchy served by this ESDS deployment. These are created during tenant on-boarding activities, usually following the initial ESDS deployment.
8. Review the default bulk extract properties in local_bulk_extract.sh, as shown below. If necessary, override these properties by creating a separate configuration file in /etc/sysconfig/ named bulk-extract:

   ```
   DEFAULT_CHECK_SLI_CONF="$ROOT/../config/properties/sli.properties"

   DEFAULT_CHECK_KEYSTORE="$ROOT/../data-access/dal/keyStore/dal-keystore.jks"

   DEFAULT_BULK_EXTRACTOR_JAR="$ROOT/target/bulk-extract-1.0-SNAPSHOT.jar"

   DEFAULT_TENANT="Midgar"
   ```

```
DEFAULT_MAX_MEMORY="1024m"
DEFAULT_MIN_MEMORY="1024m"

JAVA_OPT="-Dfile.encoding=UTF-8"

CHECK_SLI_CONF=0
CHECK_KEYSTORE=0
CHECK_SEARCH_INDEXER_TAR=0

RUN_EXTRACT=1
RUN_HELP=0

SLI_CONF="sli.conf"
SLI_ENCRYPTION_KEYSTORE="sli.encryption.keyStore"

BULK_EXTRACTER_LOG="bulk-extracter.log"
```

2.4.3.2 Adding & Updating sli.properties Entries for Bulk Extract

This section defines the sli.properties entries associated with the bulk extract service. Add or update these values as necessary for your ESDS deployment:

Property and Default Value	Description and Alternate Values
sli.bulk.extract.output.directory=/bulkextract/extracts	The output directory for bulk extract operations. Both the bulk extract service and the REST API service must have permission to read from this directory. This directory must be expressed as an absolute path.
sli.bulk.extract.log.path = /var/log/tomcat	The directory for log files for the bulk extract service. This directory must be expressed as an absolute path.
sli.bulk.extract.log.level = INFO	The log level for the bulk extract service.

Property and Default Value	Description and Alternate Values
	Refer to the Tomcat documentation for a list of log level values that could be used here.
sli.be.mongo.failOnPrimary=true	The value is Boolean. By default, bulk extract operations read from secondary MongoDB servers. Use this property to customize that behavior. When the property is set to true (default), any attempt by the bulk extract operation to read from a primary MongoDB server will fail.
sli.be.mongo.tagSet=	The value is a list of MongoDB replica set tag sets. By default, this value is not set. If this value is not set, the bulk extract operation reads from any available secondary MongoDB server. If it is set, bulk extract reads from a secondary server respective of the tags provided. Otherwise, it fails. Refer to *Configuring MongoDB for Bulk Extract* for more information about how to use this property.
sli.security.truststore.path = /tmp/trustey.jks	The path to the truststore used for two-way TLS communicatation.
sli.security.truststore.password = password	The password for the truststore.

2.4.3.3 Configuring MongoDB for Bulk Extract

By default, bulk extract operations read only from secondary MongoDB servers (sli.be.mongo.failOnPrimary=true) and from any of the available secondary servers (sli.be.mongo.tagSet is unset).

To control which secondary MongoDB servers should serve bulk extract operations, additional configuration is needed in MongoDB. Specifically, MongoDB replica sets must use tag sets as described in the following MongoDB documentation:

http://docs.mongodb.org/v2.2/applications/replication/

With tag sets configured in MongoDB, set the sli.be.mongo.tagSet property to target specific secondary MongoDB servers to fulfill bulk extract operations. The following is an example configuration showing two tag sets:

```
sli.be.mongo.tagSet = [{ "env": "prod", "geo": "east", "use" : "InUse" },
```

```
{ "env": "test", "geo": "west", "use": "InUse" }]
```

2.4.3.4 Scheduling Bulk Extractions

To schedule bulk extractions, run the schedule_bulk_extracts.sh script using the following syntax:

```
./schedule_bulk_extracts.sh /opt/bulkextract/scripts /opt/bulkextract/config
```

This script places entries in the crontab for the user who runs the script. When a scheduled extract is complete, look for the results in the target directory for extraction, configured in sli.properties using sli.bulk.extract.output.directory.

2.4.3.5 Extracting Tenant Data Without the API

Though the bulk extract feature is primarily exposed using the REST API, you can extract the data for a tenant without the API (manually) using the local_bulk_extract.sh script.

The following example depicts calling the script as a command line with the tenant name, the location of the sli.properties configuration file, the DAL encryption keystone, and the bulk_extract.tar.gz file package.

Replace "<TenantName>" with the name of the target tenant.

```
./local_bulk_extract.sh -t<TenantName> -Dsli.conf=/opt/tomcat/conf/sli.properties
-Dsli.encryption.keyStore=/opt/tomcat/encryption/DALKeyStore.jks -
f/opt/bulkextract/bulk_extract.tar.gz
```

2.4.3.6 Updating the Tomcat Configuration for Bulk Extract Encryption

When applications call a bulk extract endpoint, a client certificate is requested during the Transport Layer Security (TLS) handshake. The provided certificate is compared against the stored certificate for that application. This serves as a surrogate shared secret.

The TLS connection for bulk extract endpoints must propagate all the way to the API/bulk extract node and must not terminate at the load balancer/nginx.

Changes should be made to the Tomcat configuration to request, but not require, the client certificate during the handshake. Below is a sample configuration showing the change that needs to be made. The important parameter in this example is clientAuth="want".

```
<Connector port="8443" clientAuth="want" sslProtocol="TLS"
   keystoreFile="/etc/mykeystore.jks" keystorePass="password"
   truststoreFile="/etc/mytruststore.jks" truststorePass="password"/>
```

2.4.3.7 Removing Certificates from the ESDS Truststore

When an application developer no longer needs to apply bulk extract for their tenant, the developer's bulk extract application certificate must be removed from the ESDS truststore to disable bulk extract operations for that app. The following example can be used to remove the certificate from the truststore:

```
keytool -delete -keystore truststore.jks -alias <clientID>
```

The server's truststore for this example is 'truststore.jks'. The alias for the certificate to be removed must be the clientID of the Bulk Extract application for which bulk extract is to be disabled. The server must be restarted after the certificate is removed from the truststore.

2.4.3.8 Cleaning Up Bulk Extract Files

ESDS provides a bulk extract cleanup script to clean up bulk extract files and their associated database metadata over time.

The system running the bulk extract cleanup script must be running Ruby version 2.0.0.

Set up the script using the following steps:

1. Select or create a directory on the bulk extract server from which you want to run the bulk extract cleanup script. This is the install directory for the script.
2. Find bulk_extract_cleanup.tar in the files provided from ESDS, and copy the archive to the install directory.
3. At a command prompt to the bulk extract server, change to the directory where you copied the archive and extract it using the tar command:

   ```
   tar xf bulk_extract_cleanup.tar
   ```

4. For operator users that need to use this script, confirm that all extracted contents are readable and that the main script, cleanup_bulk_extract.rb, is executable.
5. Change to the scripts directory, and run bundle install to install the required ruby gems:

   ```
   > cd scripts
   > bundle install
   ```

6. Using your preferred text editor, edit the configuration file, bulk_extract_cleanup.yml, with local environment information for the script. The

default contents are shown below. Most default values are sufficient on various platforms. One possible exception is log_file_pathname for which the operator can choose another location.

```
# Bulk extract cleanup logging parameters.

log_file_pathname: logs/cleanup_bulk_extract.log

log_file_rotation: daily

log_level: INFO

# Bulk extract cleanup variables.

sli_database_name: sli

bulk_extract_host: localhost

bulk_extract_port: 27017

remove_db_record_retries: 3

remove_db_record_retry_interval_secs: 10
```

7. Run the script, cleanup_bulk_extract.rb, from a command prompt on the bulk extract server. The parameters you can use are:
   - tenant - The unique name for a specific tenant
   - date - A timestamp expressed in UTC or ISO8601 format
   - edOrg - The state unique ID value in the tenant database for a specific educational organization (its stateOrganizationId value)
   - file - The absolute path to a specific bulk extract file on the local filesystem

The table below defines the script options and how to use them for specific cleanup actions.

Command	Description
cleanup_bulk_extract.rb -h | -help	Print the help page for this script.
cleanup_bulk_extract.rb -t[tenant]	Remove all bulk extract files and their database metadata for this tenant (tenant).
cleanup_bulk_extract.rb -t[tenant] -d[date]	Remove all bulk extract files and their database metadata extracted up to this date (date) for this tenant (tenant).
cleanup_bulk_extract.rb -t[tenant] -e[edOrg]	Remove all bulk extract files and their database metadata for this educational organization (edOrg) belonging to this tenant (tenant).

Command	Description
cleanup_bulk_extract.rb -t[tenant] -e[edOrg] -d[date]	Remove all bulk extract files and their database metadata extracted up to this date (date) for this educational organization (edOrg) belonging to this tenant (tenant).
cleanup_bulk_extract.rb -t[tenant] -f[file]	Remove this specific bulk extract file (file) and its database metadata for this tenant (tenant).

Note: Use double quotes around any parameter containing whitespace.
For example: -e "Sunset Central High School"

The following occurs when the script calls to clean up bulk extract files

1. On-screen output indicates that the script is logging results to the configured log file. The log file reflects separate entries for each script execution, and it includes the exact command executed.
2. If the command syntax is correct, the script validates the provided parameter values (tenant, edOrg, date, filename). If any parameter value is invalid, the script exits with an error message.
3. If the parameters are valid, the script prompts for you to confirm the cleanup action, displaying the parameter values provided. If the user answers 'y' or 'yes', the script will perform the cleanup action. If the user provides any other response, the script exits with no action taken.
4. During the cleanup action, the script performs the following for each bulk extract file targeted for removal:
   - The corresponding record in the bulk extract database collection is removed.
   - The bulk extract file is deleted.
   - The actions taken are logged to the cleanup script log file.
5. When the script finishes processing, it displays a summary on screen indicating the number of files processed and removed, along with any failed processes.

2.4.3.9 Troubleshooting Errors in the Bulk Extract Cleanup Script

This section provides a summary of the errors that could occur when using the bulk extract cleanup script and how the script behaves when it encounters the error:

Error	Behavior
Errors in command syntax	Script exits with error
Invalid parameters when calling the script	Script exits with error
The target file to be deleted does not exist	Script reports a warning and continues
Database connection lost while processing -	Script attempts to reconnect and displays retry attempts. If unsuccessful, outputs a summary of cleanup to the point of failure and exits.

2.4.3.10 Required Maintenance Tasks for Bulk Extract Service

he tasks are required as part of maintaining both the bulk extract service and the ESDS infrastructure that uses it.

2.4.3.10.1 Required Actions During REST API Updates

When deploying updates that include a new version of the ESDS REST API, deployment tasks must include the following steps. These steps ensure that the bulk extract file contents reflect the new version of the API.

1. As part of stopping services prior to the REST API update, stop the cron process for scheduled bulk extract operations and be sure that no extracts are currently running.
2. Remove all previously extracted files from the bulk extract target directory. This is the directory identified in sli.properties for sli.bulk.extract.output.directory.
3. Remove all records in the bulkExtractFiles collection in the sli database.
4. If necessary, run an extract operation immediately following the API update to replace the files that were removed.
5. Resume the scheduled bulk extract operations in cron.

2.4.3.10.2 Required Database Updates After Deleting Files

Besides deleting files on a regular basis with the cleanup script, there may be other reasons that bulk extract files are deleted from the filesystem. When files have been deleted without the cleanup script, additional action is required to update a database record so that the API no longer references the deleted files. Use the following steps when deleting bulk extract files directly without the script:

1. Before deleting a bulk extract file, record its absolute path as it exists in the server's filesystem, such as:

   ```
   /bulk/extract/extract/02f7abaa9764db2fa3c1ad852247cd4ff06b2c0a/19cca28d-7357-4044-8df9-caad4b1c8ee4-Midgar-2013-05-06T12-41-59.tar
   ```

2. Access the command line of the MongoDB server that is responsible for the bulkExtractFiles collection in the "sli" database, and run MongoDB to launch the MongoDB Shell console.
3. Remove the corresponding record for the target file from the bulkExtractFiles collection using the absolute path you just recorded. For example:

   ```
   db.bulkExtractFiles.remove({"body.path" : \

   "/bulk/extract/extract/02f7abaa9764db2fa3c1ad852247cd4ff06b2c0a/19cca28d-7357-4044-8df9-caad4b1c8ee4-Midgar-2013-05-06T12-41-59.tar"})
   ```

4. Remove the file from the bulk extract server.

2.5 Ops Tools

The opstools package provides various utilities and scripts that are helpful to an operator. This package contains notable directories and files. Documentation for each item listed below resides in the script itself. Call the script with no parameters or open it in a text editor to access the associated documentation.

genAppKeys.rb – This Ruby script generates a set of client_id and client_secret keys which can be used when bootstrapping various applications within the platform.

Migration – This directory contains the required migration scripts for upgrading an SDS installation from one version to another. These scripts are required to be run when upgrading the platform from one version to another. Further details are provided in the appropriate documentation with the release.

Ingestion_trigger – This script is used as part of the proftpd solution to trigger ingestion after a file has been uploaded. This script can also be called manually to trigger ingestion.

2.6 MongoDB

An ESDS technology deployment consists of multiple MongoDB databases that live on a single MongoDB cluster. It includes a system database, an ingestion batch job database and a tenant database The ingestion and API servers automatically populate the ingestion batchjob and sli databases. However, you must apply the indexes ahead of time to ensure proper creation. The tenant database makes use of MongoDB sharding, a mechanism that increases the scalability of the MongoDB database and distributes the data across multiple shards.

2.6.1 Cluster Setup

The MongoDB cluster should consist of three config servers and at least one replica set consisting of three servers. Refer to the MongoDB documentation for more information on setting up and managing a MongoDB cluster.

2.6.2 Databases

The following sections describe the three types of required databases for an ESDS deployment: the system database, the ingestion batch job database and the tenant databases.

2.6.2.1 System Database

The system database for the ESDS is called *sli*. It requires setting up indexes by running the MongoDB script *sli_indexes.js* against the sli database. This script is provided by ESDS and can be found in the indexes package. To create the database and apply the appropriate indexes run the following command:

```
mongo sli < sli_indexes.js
```

2.6.2.2 Ingestion Batch Job Database

The ingestion batch job database is called ingestion_batch_job. It requires setting up indexes by running the MongoDB script *ingestion_batch_job.js* against the ingestion_batch_job database. This script is provided by ESDS and can be found in the indexes package. To create the database and apply the appropriate indexes run the following command.

```
mongo ingestion_batch_job < ingestion_batch_job_indexes.js
```

2.6.2.3 Tenant Databases

The ESDS tenant databases make use of sharding. This sharding is configured automatically by the ingestion process when the tenant ingests data for the first time. You are not required to create or manage the tenant database in any way.

2.6.3 Configuration

Running the following command allows you to view the current state of the MongoDB balancer:

```
sh.getBalancerState()
```

If the current state is *true* then you should disable it by running the following command:

```
sh.setBalancerState(false).
```

You can then run the `sh.getBalancerState()` command to verify that it has been disabled.

Important! This process disables the MongoDB balancer, which should remain in a disabled state. In the event that you wish to add an additional shard or remove one, you must stop the bulk ingestion process before re-enabling the balancer in order to facilitate chunk migration. Once this process is complete, the balancer should be disabled again and bulk ingestion re-enabled.

2.7 ActiveMQ

Apache ActiveMQ (Apache 2.0 licensed) is an open source message broker that fully implements the Java Message Service 1.1 (JMS). It provides enterprise features like clustering, multiple message stores, and the ability to use any database as a JMS persistence provider besides VM, cache, and journal persistence.

ActiveMQ is used by the ingestion service as part of the ingestion process.

2.7.1 Requirements

ActiveMQ requires the following:

- Oracle Java JDK
- Apache AcitveMQ 5.6.0

2.7.2 Installation

1. Install the Apache ActiveMQ software and create an activemq user:

   ```
   wget http://www.eng.lsu.edu/mirrors/apache/activemq/apache-activemq/5.6.0/apache-activemq-5.6.0-bin.tar.gz

   useradd activemq -d /opt/activemq
   ```

2. Expand tar file to /opt, set permissions for activemq user and symlink activemq:

   ```
   tar -xzf apache-activemq-5.6.0-bin.tar.gz -C /opt/

   chown -R activemq: /opt/apache-activemq-5.6.0

   cd /opt && ln -s apache-activemq-5.6.0 activemq
   ```

3. Edit and copy the provided init script:

   ```
   cp /opt/activemq/bin/linux-x86-64/activemq  /etc/init.d/
   ```

4. Edit the following fields to point to the ActiveMQ home directory locations and enable running as the activemq user:

   ```
   vi /etc/init.d/activemq

   ACTIVEMQ_HOME="/opt/activemq"

   WRAPPER_CMD="/opt/activemq/bin/linux-x86-64/wrapper"

   WRAPPER_CONF="/opt/activemq/bin/linux-x86-64/wrapper.conf"

   RUN_AS_USER=activemq

   PIDDIR="/tmp"
   ```

5. Edit wrapper.conf to include the current ActiveMQ home directory locations:

   ```
   vi /opt/activemq/bin/linux-x86-64/wrapper.conf

   set.default.ACTIVEMQ_HOME=/opt/activemq/

   set.default.ACTIVEMQ_BASE=/opt/activemq
   ```

6. Enable ActiveMQ to start at startup:

   ```
   chkconfig --add activemq
   ```

7. Set memoryLimit (under destinationPolicy) for queue to 750mb.
8. Set memoryLimit (under destinationPolicy) for topic to 750mb.
9. Set memoryUsage (under systemUsage) for broker to 1gb.
10. Set producerFlowControl (under destinationPolicy) for topic to false.
11. Set producerFlowControl (under destinationPolicy) for queue to false.
12. Enable ActiveMQ broker for stomp protocol. This is used by the SARJE Oplog Agent and the ProFTPD+publish_file_uploaded.rb script configured in ???
13. Add the following inside of <transportConnectors> in activemq.xml:

```
<transportConnector name="stomp" uri="stomp://0.0.0.0:61613"/>
```

14. Ensure that the port configured for stomp (e.g. 61613) is opened to the landing zone servers that will be running ProFTPD.
15. Start ActiveMQ.

```
service activemq start
```

2.7.2.1 Setting ActiveMQ Redundancy

When setting ActiveMQ redundancy, the JMS transports for connecting to ActiveMQ support the use of the Failover Transport mechanism on the JMS URI. Information on the failover transport URI for ActiveMQ can be found at:

http://activemq.apache.org/failover-transport-reference.html

Information on configuring ActiveMQ for a distributed cluster can be found at:

http://activemq.apache.org/networks-of-brokers.html

2.7.2.2 Configuring Ingestion to Utilize ActiveMQ Redundancy

Since ActiveMQ is a JMS message queuing provider and ingestion makes use of the JMS interface, a failover transport definition may be used to failover to an alternative ActiveMQ instance.

To use a redundant ActiveMQ message queueing system, update the sli.properties configuration file to represent the four example configuration entries listed below.

In the example below, sli.ingestion.queue.workItem.secondaryhost and sli.ingestion.queue.workItem.secondaryport variables represent the secondary server, and the sli.ingestion.queue.options and sli.ingestion.queue.brokerUrl options inform the client to use the secondaryhost and secondaryport options.

```
sli.ingestion.queue.workItem.secondaryhost= <Add host name here>

sli.ingestion.queue.workItem.secondaryport= <Add port here>

sli.ingestion.queue.options=
randomize=false&jms.prefetchPolicy.queuePrefetch=0&wireFormat.maxInact
ivityDurationInitalDelay=60000&keepAlive=true&trackMessages=true

sli.ingestion.queue.brokerUrl=
failover:(tcp://${sli.ingestion.queue.workItem.host}:${sli.ingestion.queue.w
orkItem.port}?keepAlive=true,tcp://${sli.ingestion.queue.workItem.secondar
yhost}:${sli.ingestion.queue.workItem.secondaryport}?keepAlive=true)?${sli.
ingestion.queue.options}
```

2.8 RESTful API

The RESTful API provides the interface between MongoDB and the applications that need to read or write data. Applications like the Dashboard and Data Browser interact with the SDS through the RESTful API.

2.8.1 Requirements

The following components are required for RESTful API to function properly and should be configured and operational before you attempt to deploy this service.

- MongoDB
- ActiveMQ
- sli.properties file (For more information, refer to the Configuration section.)
- Keystore and truststore (For more information, refer to the Cryptographic Keys section.)

2.8.2 Configuration

Deploying the REST API requires a large number of properties to be configured in the sli.properties file:

- Bootstrap properties – A Java properties file containing all the configurable parameters for bootstrap applications. It defines values such as the client id, secret, and application URL.
- Application templates – JSON files containing application values that either should not need to be modified, or cannot easily be represented in a properties file.
- ESDS developer account - An account with the name slcdeveloper and application developer role. Refer to the sections SimpleIDP and LDAP for more information on creating users.
- Bootstrap properties location - The bootstrap properties are referenced through the bootstrap.app.conf property. This property should be added to the canonical config (sli.properties). If the value is blank or if the file it points to does not exist on the file system, bootstrapping of applications will be skipped when the API is loaded.
- Bootstrap properties keys - The bootstrap properties contain the following keys:
    - bootstrap.app.keys - comma-separated list of app keys that will be used elsewhere in the properties file to reference specific applications. For example, the key for the admin tools application could be "admin".
    - bootstrap.app.<key>.template - location within the API of the JSON template file for the application with the given key. The templates are resolved relative to the directory containing the bootstrap properties file.
    - bootstrap.app.<key>.guid - (optional) use this property to set a predefined value for the application's Mongo ID. This should not be needed in a Production environment.

- Any arbitrary property can exist, and templates can reference it using the ${...} notation, as seen in the following sample template.

Sample admin.json:

```
{
  "name": "${bootstrap.app.admin.name}",
  "description":"${bootstrap.app.admin.description}",
  ...
}
```

The API application will look for its configuration in *$TOMCAT_HOME/conf/sli.properties* and will look for the following properties to be set (example values shown):

```
# used to specify the location of performance log (only used in API)
api.perf.log.path = target/apilogs/logs

# Security  (only used in API)
sli.security.noSession.landing.url =
http://local.example.com:8080/api/oauth/authorize?response_type=code

# Security SAML (only used in API)
sli.security.sp.issuerName = http://local.example.com:8080
sli.security.idp.url:
https://shibboleth.slidev.org/idp/profile/SAML2/SOAP/ArtifactResolution

# Security grace period for viewing sections and enrollments (only use in
API)
# This property must be set to 0 in a live Production environment
sli.security.gracePeriod = 2000

# Maximum number of returns for a response (only used in API)
sli.security.in_clause_size = 100000
```

```
# Trusted Certificate Authority Store (used in common)
sli.trust.certificates = ../common/common-util/trust/trustedCertificates

# Support Email (only used in API)
sli.support.email = sli@example.com

# Session lengths in milliseconds (5 minutes - only in API)
sli.session.length = 300000

# Session hard logout (8 hours - only in API)
sli.session.hardLogout = 28800000

# Mongo settings
sli.mongodb.database = sli
sli.mongodb.host = localhost:27017
sli.mongodb.keyencoding: \%:\%25,\\.:\%2E

# Encryption settings.  Should be the same across all API and Ingestion
nodes
sli.encryption.keyStore = /path/to/api-keystore.jks
sli.encryption.keyStorePass = storepass
sli.encryption.dalKeyAlias = dalkey
sli.encryption.dalKeyPass = dalpass
sli.encryption.dalInitializationVector = 32RandomHexCharacters

# Landing Zone SFTP server
sli.landingZone.server = rclz.example.com

# Used to bootstrap the admin realm
bootstrap.admin.realm.name = ESDS
```

bootstrap.admin.realm.tenantId = SLI

bootstrap.admin.realm.idpId = https://idp.example.com:443/sp

bootstrap.admin.realm.redirectEndpoint = https://idp.example.com:443/sp/SSORedirect/metaAlias/idp

#Used to bootstrap the application developer realm

bootstrap.developer.realm.name = ESDS App Developers

bootstrap.developer.realm.uniqueId = DeveloperIDP

bootstrap.developer.realm.idpId = <sandbox bootstrap.admin.realm.idpId>&developer=true

bootstrap.developer.realm.redirectEndpoint = <sandbox bootstrap.admin.realm.redirectEndpoint>&developer=true

API kesytore settings

sli.api.keyStore: /path/to/api-keystore.jks

sli.api.keystore.password: encryptedStorepass

sli.api.digital.signature.keyAlias: entryAlias

sli.api.digital.signature.keyPass: encryptedEntrypass

sli.api.client.certificate.keyAlias: entryAlias

sli.api.client.certificate.keyPass: encryptedEntrypass

sli.api.encryption.certificate.keyAlias: encryptionAlias

sli.api.encryption.certificate.keyPass: encryptionPass

Sandbox settings

sli.autoRegisterApps = false

sli.simple-idp.sandboxImpersonationEnabled = false

#Search engine cluster URL

sli.search.url = http://<Internal ElasticSearch Load Balancer>:9200

```
#Search engine cluster username (user)

sli.search.username = 17BF72EAF6893034F8FB4AE35BF3A567


#Search engine cluster password (searchme)

sli.search.password = 4284B82ACF30963CA2315839AC939C62


#Search engine query limits (only used in API)

sli.search.maxUnfilteredResults = 15000

sli.search.maxFilteredResults = 250

sli.search.maxFilteredResultsOverride = 5000


#This setting should always be set to false in a production environment.  It is
intended to only be used in conjunction with localized development testing.

sli.search.embedded = false


#set to true when username and password are encrypted

sli.search.encryption = true


# These settings are required, however they should remain the defaults
listed below.

sli.mongodb.connections = 30

sli.perf.mongodb.database = apiPerf

sli.perf.mongodb.host = localhost

sli.perf.mongodb.port =27017

sli.api.performance.tracking = false
```

You may use the encryption tool to generate encrypted values. To create the encrypted values using the encryption tool, run the following command:

```
java -jar encryption-tool-1.0-SNAPSHOT.jar <keystore_location>
<keystore_password> <key_alias> <key_password> <value to encrypt>
```

API uses the following properties from sli.properties for the parameters above:

keystore_location : sli.encryption.keyStore

keystore_password : sli.encryption.keyStorePass

key_alias : sli.encryption.ldapKeyAlias

key_password : sli.encryption.ldapKeyPass

This example assumes that the encryption tool jar is in current working directory.

The Grace Period prevents data from becoming unavailable by extending its contextual date range by the number of days specified. It is important to set *sli.security.gracePeriod* to 0 (zero) before allowing public access to a Production environment. To leave the Grace Period set to a value greater than 0 (zero) in a live Production environment could expose data to users who would not normally have permission to view it.

2.8.3 Installation

Use the following procedure to install the REST API service:

1. Create a *bootstrap.properties* file on the machine that will run the API.
2. Place the template files in the same directory as the *bootstrap.properties*. Reference versions of the template files for each application are located in the *sli/config/applications/* directory.
3. Add the following properties to the file:

   bootstrap.app.keys=admin,portal,dashboard,databrowser

   bootstrap.app.vendor = ESDS

   # Admin-specific properties

   bootstrap.app.admin.template = ./admin.json

   bootstrap.app.admin.name=Admin Apps

   bootstrap.app.admin.description = The ESDS Administration Application allows you to change a variety of system settings.

   bootstrap.app.admin.url = <app url, e.g. https://admin.localhost>

   bootstrap.app.admin.client_id = <random 10-character string>

   bootstrap.app.admin.client_secret = <random 48-character string>

```
bootstrap.app.admin.version = 0.0

# Portal-specific properties

bootstrap.app.portal.template = ./portal.json

bootstrap.app.portal.name=Portal

bootstrap.app.portal.description = The ESDS Portal application is the
primary access portal.

bootstrap.app.portal.url = <portal url, e.g.
https://portal.localhost/portal/login/c>

bootstrap.app.portal.client_id = <random 10-character string>

bootstrap.app.portal.client_secret = <random 48-character string>

bootstrap.app.portal.version = 0.0

bootstrap.app.databrowser.template: applications/databrowser.json

bootstrap.app.databrowser.name: ESDS Data Browser

bootstrap.app.databrowser.description: The ESDS Data Browser allows
developers and administrators to access all available information in the
ESDS Data Store.

bootstrap.app.databrowser.version: 0.0

bootstrap.app.dashboard.name: ESDS Dashboards

bootstrap.app.dashboard.description: The ESDS Dashboards allow you to see
information about students in lists and profiles.

bootstrap.app.dashboard.template: applications/dashboard.json

bootstrap.app.dashboard.version: A.0
```

4. Generate a sample *client_id* using *mkpasswd -s 0 -l 10*, and a sample *client_secret* can using *mkpasswd -s 0 -l 48*.
5. Add the *bootstrap.app.conf* property to the *sli.properties* to point to the bootstrap.properties, (For example: bootstrap.app.conf = /home/user/bootstrap.properties)
6. Start the API server.
7. Verify bootstrapping was successful by querying the application collection in MongoDB as shown below:

```
> db.application.find().pretty()
```

Optionally, you may add other applications to the API bootstrapping as follows:

a. Add your application name to the bootstrap.app.keys property.

b. Copy the bootstrap.app.dashboard.* properties, paste the properties and replace "dashboard" using the unique name for your application. Putting the properties in the bootstrap.properties file for specific environments allows using the same template then adjusting the client_id, client_secret and callback url for each different environment.
c. Adjust the value of the properties in the bootstrap.properties file to reflect your application.
d. Copy the dashboard.json application template file, rename and adjust the attributes as required.

8. (Optional) Delete the bootstrap.properties file to avoid having plain-text copies of the client id and secret on the filesystem.

2.9 GlusterFS

GlusterFS is a distributed filesystem that allows for a storage pool to be mirrored across multiple servers. This pool can be mounted on multiple servers without complication. The SDS system uses GlusterFS between the landing zone servers and the ingestion servers.

2.9.1 Configuration

A single volume named *ingestion* must be created. It is recommended that this volume be 4x the size of the largest decompressed ingestion job that you would upload.

Important! It is highly recommended that you restrict the hosts that can access the volume using GlusterFS built in the ip restriction policies. The only hosts that should have access to the volume are ingestion and landing zone servers.

2.9.2 Installation

GlusterFS is a distributed filesystem which allows a user to take various storage "bricks" and assemble them into one larger storage pool of storage. GlusterFS should be deployed across at least two storage bricks.

The procedure below assumes that on each server, a volume named */gluster* has been mounted and represents the local disk storage. It also assumes that you are using GlusterFS specifically for the landing zones service.

1. Install the GlusterFS packages:

```
rpm -ihv http://download.gluster.org/pub/gluster/glusterfs/LATEST/RHEL/\

glusterfs-core-3.2.6-1.x86_64.rpm
http://download.gluster.org/pub/gluster/\

glusterfs/LATEST/RHEL/glusterfs-fuse-3.2.6-1.x86_64.rpm \
```

```
http://download.gluster.org/pub/gluster/glusterfs/LATEST/RHEL/\

glusterfs-geo-replication-3.2.6-1.x86_64.rpm
```

2. Setup the services to start on boot:

```
chkconfig glusterfsd on

chkconfig glusterd on
```

3. Join the servers together using the Gluster peer probe. This needs to be performed for every server after the first one. If you are deploying Gluster across four nodes, you would need to execute the command initially and then three times for the three additional servers.

```
[root@gluster01 ~]# gluster peer probe gluster02

Probe successful
```

4. Confirm that a peer exists using `gluster peer status`.
5. Create your GlusterFS volume:

```
gluster volume create ingestion replica 2 transport tcp \

gluster01:/gluster gluster02:/gluster
```

6. Set IP level permissions for Gluster volume access:

```
gluster volume set ingestion auth.allow 10.10.10.11,\

10.10.10.12,10.10.11.11,10.10.11.12
```

7. View the volume settings via the `gluster volume info` command.
8. Start the GlusterFS volume.

```
[root@gluster01 ~]# gluster volume start ingestion

Starting volume ingestion has been successful
```

2.9.2.1 Attaching GlusterFS to the Ingestion and Landing Zone Servers

Use the following procedure to attach the GlusterFS volume (created in the installation procedure above) to the ingestion and landing zone servers:

1. Install the packages needed for attaching the volume:

```
yum -y install opensm fuse fuse-libs libibverbs
```

2. Install GlusterFS Packages:

```
rpm -ihv http://download.gluster.org/pub/gluster/glusterfs/LATEST/RHEL/\

glusterfs-core-3.2.6-1.x86_64.rpm
http://download.gluster.org/pub/gluster/\

glusterfs/LATEST/RHEL/glusterfs-fuse-3.2.6-1.x86_64.rpm \

http://download.gluster.org/pub/gluster/glusterfs/LATEST/RHEL/\

glusterfs-geo-replication-3.2.6-1.x86_64.rpm
```

3. Make a directory to mount the folder, such as */ingestion*:

```
mkdir /ingestion
```

4. Add an entry in */etc/fstab* as follows:

```
echo "gluster01:/ingestion /ingestion glusterfs
defaults,_netdev,transport=tcp,\

log-level=WARNING,log-file=/var/log/gluster.log 0 0" >> /etc/fstab
```

5. Use the mount command to mount your new volume:

```
mount -a
```

2.9.2.2 Setting up the Landing Zone Servers

Use the following procedure to complete setting up the landing zone service following the successful completion of the GlusterFS setup and attachment.

The scripts provided via the links below should be deployed to the landing zone servers:

opstools/ingestion_trigger/publish_file_uploaded.rb

opstools/ingestion_trigger/ftpwrapper.sh

To deploy, copy the scripts to */opt/sli/bin/*.

The ActiveMQ Stomp rubygem must be installed for this script to function: https://rubygems.org/gems/stomp gem install stomp

1. Install the LDAP packages required to complete this process:

   ```
   yum install -y pam_ldap nss-pam-ldapd
   ```

2. Create or modify the */etc/ldap.conf* to match your environment. (Depending on the Linux OS version, this may be */etc/pam_ldap.conf*.) This should be configured to point to the same environment and organizational unit structure as the admin application and the SimpleIDP. Specifically, users and group memberships will need to be accessed by the landing zone server.
3. Edit the */etc/proftpd.conf* file to match your environment and security requirements. At a minimum:
   a. Enable SFTP
   b. Ensure that the DefaultRoot for users is set to "~".
   c. Add the mod_exec configuration for the ingestion trigger.

   Depending on the authentication configuration, you will likely have ProFTPD point to its own PAM configuration via the AuthPAMConfig option.
4. The following configuration will trigger ingestion upon file upload completion and should be added to the */etc/proftpd.conf* file:

   Note: You must replace the #ACTIVEMQSERVENAME# placeholder with the top level domain of the ActiveMQ server.

   ```
   LoadModule mod_exec.c

   <IfModule mod_exec.c>

    ExecEngine on

    ExecLog /var/log/proftpd/mod_exec.log

    ExecOnCommand STOR /opt/sli/bin/ftpwrapper.sh %U %m %f
   #ACTIVEMQSERVENAME#

   </IfModule>
   ```

   Note: ESDS requires mod_exec and mod_rewrite. These modules have been validated and included in the *proftpd* packages provided by Red Hat and Ubuntu.
5. Add the following configuration to the /etc/proftpd.conf file. This adds a timestamp component to uploaded .zip files.

   ```
   LoadModule mod_rewrite.c

   <IfModule mod_rewrite.c>
   ```

```
    RewriteEngine on
    RewriteCondition %m STOR
    RewriteRule ^(.*?)\.(zip)$ $1.%t.$2
</IfModule>
```

6. The following configuration controls the file types that can be uploaded and should be added to the */etc/proftpd.conf* file:

   ```
   PathAllowFilter \.(zip)
   ```

   Note: Do not use multiple PathAllowFilter directives; proftpd only reads the first one.

7. In the PAM configuration stack used by ProFTPD, perform group membership validation to ensure that the user is a member of the "ingestion_user" group. This can be performed via the pam_lsitfile.so module:

   ```
   auth required pam_listfile.so onerr=fail item=group sense=allow \
   file=/etc/proftpd_allowed_group
   ```

8. Ensure the */etc/proftpd_allowed_group* file scontain "ingestion_user", which matches the group name in LDAP.
9. Depending on your ProFTPD configuration, you will need to add /sbin/nologin to your /etc/shells file. Users created in the LDAP Directory get a default loginShell of /sbin/nologin. This is especially important when operating in Sandbox mode, where developer accounts have the intended ability to log in directly for test data ingestion.

At this point, if all is working correctly, a user should be able to log in to the SFTP server via the LDAP server.

Note: ProFTPD will not allow a user to log in that has a nonexistent home directory. You may want to create a temporary test user that is rooted at an upper level folder such as */ingestion* for initial testing purposes only.

2.9.3 Troubleshooting

When attempting to mount the volume on a client, most connectivity issues will be firewall related. If you receive any errors relating to the status of the volume that do not seem connectivity related, try unmounting the volume on all hosts, stop/start the volume, and then attempt to remount.

2.10 Ingestion

The Ingestion process provides a tenant the ability to perform bulk ingestion into the SDS. Generally, the ingestion process allows a zip file to be uploaded that contains xml files along with a control file. More details on how to perform ingestion can be found in the *Ingestion Guide*. This section focuses on the setup of the Ingestion process. Ingestion and landing zone share a common GlusterFS filesystem. It is very important to have the GlusterFS system in place before launching landing zone and ingestion services.

2.10.1 Requirements

Ingestion requires the following components:

- GlusterFS
- ActiveMQ
- LDAP
- MongoDB
- Tomcat Application Server

2.10.2 Configuration

The main configuration file for ingestion is the sli.properties file, which is kept in $TOMCAT_HOME/conf/sli.properties.

1. Modify the contents to suit your own environment:

```
sli.tenant.ingestionServers = SET_LIST_OF_INGESTION_SERVERS

sli.tenant.landingZoneMountPoint = /ingestion/lz

sli.ingestion.batchjob.mongodb.database = ingestion_batch_job

sli.ingestion.batchjob.mongodb.host = SET_INGESTIONMONGO_HOST

sli.ingestion.batchjob.mongodb.port = 27017

sli.ingestion.errors.tracking = true

sli.ingestion.warnings.tracking = true

sli.ingestion.securityEvent.capSize =

sli.ingestion.healthcheck.user = admin

sli.ingestion.healthcheck.pass = admin

landingzone.inbounddir = /ingestion/lz/

sli.ingestion.topic.command = activemq:topic:ingestion.command

sli.ingestion.exception.message.log = true
```

sli.ingestion.log.level = info

sli.ingestion.queue.workItem.host = SET_ACTIVEMQ_HOST

sli.ingestion.queue.workItem.port = 61616

sli.ingestion.queue.workItem.secondaryhost =

sli.ingestion.queue.workItem.secondaryport =

sli.ingestion.queue.options = keepAlive=true&jms.prefetchPolicy.queuePrefetch=0&wireFormat.maxInactivityDurationInitalDelay=60000&trackMessages=true

sli.ingestion.queue.brokerUrl = tcp://${sli.ingestion.queue.workItem.host}:${sli.ingestion.queue.workItem.port}?${sli.ingestion.queue.options}

sli.ingestion.queue.maxConnections = 25

sli.ingestion.queue.maximumActive = 500

sli.ingestion.queue.workItem.queueURI = seda:IngestionWorkItem

sli.ingestion.queue.workItem.concurrentConsumers = 4

sli.ingestion.queue.pit.uriOptions = &transferExchange=true

sli.ingestion.queue.parser.queueURI: activemq:queue:IngestionParser

sli.ingestion.queue.parser.concurrentConsumers: 8

sli.ingestion.queue.parser.uriOptions:

sli.ingestion.parser.batch.size = 1000

sli.ingestion.purge.batch.size = 30000

sli.ingestion.nodeType = standalone

sli.ingestion.tenant.loadDefaultTenants = true

sli.ingestion.tenant.tenantPollingRepeatInterval = 5s

sli.ingestion.referenceSchema.referenceCheckEnabled = false

sli.ingestion.errorsCountPerInterchange = 10000

sli.ingestion.warningsCountPerInterchange = 10000

sli.ingestion.totalRetries = 5

sli.ingestion.dataset.sample = {"small":["SmallSampleDataSet.zip"],"medium":["MediumSampleDataSet.zip"]}

sli.mongodb.connections = 30

sli.stagingmongodb.connections = 20

sli.ingestion.batchjobmongodb.connections = 20

sli.mongodb.keyencoding: \%:\%25,\\.:\%2E

sli.ingestion.queue.landingZone.queueURI:
activemq:queue:ingestion.landingZone

2. Update the API configuration in the sli.properties file to include the following ingestion parameters. These settings are used when a landing zone is created. The ingestion service must be configured to have -Xmx40G available.

   sli.tenant.ingestionServers = server1,server2,server3,server4

   sli.tenant.landingZoneMountPoint = /ingestion/lz

3. Edit the *catalina.sh* or the *setenv.sh* scripts located in $TOMCAT_HOME to apply certain environment variables that ingestion uses on start up. Add the following into the JAVA_OPTS string:

   -Xms40G -Xmx40G -XX:+UseParallelGC -XX:PermSize=512m -
   XX:MaxPermSize=512m

   It should look like this:

   JAVA_OPTS="-Xms40G -Xmx40G -XX:+UseParallelGC -XX:PermSize=512m

   -XX:MaxPermSize=512m -Dsli.env=rc

   -Dsli.encryption.keyStore=/opt/tomcat/encryption/ciKeyStore.jks

   -Dsli.encryption.properties=/
   opt/tomcat/encryption/ciEncryption.properties

   -Dsli.trust.certificates=/opt/tomcat/trust/ trustedCertificates -
   Dsli.conf=/opt/tomcat/apache-tomcat-7.0.27/conf/sli.properties

   -Dlogging.path=/ var/log/tomcat/ -Dapi.perf.log.path=/var/log/tomcat/"

2.10.3 Installation

Place the ingestion.war file in $TOMCAT_HOME/webapps/.

Note: All ESDS Java web applications should be deployed to the context specified by the *.war* filename. They should not be deployed to the root context in Tomcat.

2.10.4 Troubleshooting

If on startup you receive an error in ingestion.log relating to indexes, refer to the MongoDB section and apply the ingestion_batch_job and sli database indexes.

2.11 Landing Zone

The landing zone is a location that a user can log into, place files for the ingestion service to process, and begin populating the database with an educational organization and tenant information. The SDS utilizes ProFTPD to provide these services.

2.11.1 Requirements

Landing zone requires the following components:

- GlusterFS
- ActiveMQ
- Ingestion
- LDAP

2.11.2 Configuration

For standard best practices and setup, refer to the ProFTPD documentation. The following ProFTPD modifications are required for SDS deployment.

1. ProFTPD must be configured to use LDAP for authentication. It is recommended executing this function this via *pam-ldap*.
2. Jail users into their home directory.
3. Configure mod_exec with the following paramaters:

```
LoadModule mod_exec.c

    <IfModule mod_exec.c>

       ExecEngine on

      ExecLog /var/log/proftpd/mod_exec.log

      ExecOnCommand STOR /opt/sli/bin/ftpwrapper.sh %U
%m %f #ACTIVEMQSERVENAME#

 </IfModule>
```

mod_rewrite should be configured with the following parameters:

```
LoadModule mod_rewrite.c
```

```
<IfModule mod_rewrite.c>
    RewriteEngine on
     RewriteCondition %m STOR
     RewriteRule ^(.*?)\.(zip)$ $1.%t.$2
</IfModule>
```

4. Restrict the types of files that to be uploaded to the following:

   ```
   PathAllowFilter \.(zip)
   ```

   Ensure the filename ends with .zip. This does not prevent a user from uploading another file type ending as .zip. In the event that this occurs, ingestion will return an error as it attemptsto unzip the file.

5. In the PAM configuration stack used by ProFTPD, perform group membership validation to ensure that the user is a member of the "ingestion_user" group. This can be performed via the pam_lsitfile.so mod- ule.

   ```
   auth required pam_listfile.so onerr=fail item=group sense=allow \ file=/etc/
       proftpd_allowed_group
   ```

   The /etc/proftpd_allowed_group file should contain "ingestion_user", which matches the group name in LDAP.

6. Depending on your ProFTPD configuration, you will need to add /sbin/nologin to your `/etc/shells` file. Users created in the LDAP Directory get a default loginShell of `/sbin/nologin`. This is especially important when operating in Sandbox mode, where developer accounts have the intended ability to log in directly for test data ingestion.

2.11.3 Installation

Install and configure ProFTPD using standard best practices and the required configuration paramaters as discussed in the sections above.

1. Create a directory located at `/opt/sli/bin/`
2. Place the following two files in this directory:

   ```
   opstools/ingestion_trigger/publish_file_uploaded.rb
   ```

   ```
   opstools/ingestion_trigger/ftpwrapper.sh
   ```

3. Install the stomp gem:

   ```
   gem install stomp
   ```

2.11.4 Troubleshooting

Troubleshooting the landing zone will primarily be an exercise in ensuring that the following statements are true.

- A shared filesystem (GlusterFS) is mounted and is allowing the same folder path to be visible between the servers.
- A landing zone user can be authenticated via the Admin application's LDAP server, can log in via the ProFTPD daemon operating in SFTP mode, and can access a home directory that resides on the shared filesystem.
- Application and User operating umasks, as well as group memberships are established so that a file written to the landing zone server can be read and written to by the user running the ingestion service.

2.12 LDAP

SimpleIDP uses OpenLDAP for storage of administrative accounts. Only accounts such as SEA Administrators, LEA Administrators, Operators, and Ingestion Users are stored in LDAP. Tenant-specific accounts for teachers and staff would reside in that tenants IDP. Password Policy is managed by the policy overlay.

An installation of at least two OpenLDAP servers is recommended with N-Way Multi-Master Replication.

OpenLDAP installation documentation can be found at:
http://www.openldap.org/doc/admin24/install.html

OpenLDAP documentation on replication can be found at:
http://www.openldap.org/doc/admin24/replication.html

OpenLDAP documentation on password policy controls can be found in the OpenLDAP Administration Guide. The specific information can be found here:
http://www.openldap.org/doc/admin24/overlays.html#Password Policies

OpenLDAP documentation on SSL setup can be found at:
http://www.openldap.org/doc/admin24/tls.html

Important! A non-manager account MUST be utilized for all interaction between the software and the LDAP directory because the OpenLDAP Manager account does not have the "ppolicy" overlay applied to its interactions. Therefore. a generic account object

must be created and granted elevated privileges through the Access/olcAccess method. This allows the non-manager account to Read/Write across the Organizational Unit that the software is configured to utilize.

The default OpenLDAP query size limit (olcSizeLimit) of 500 must be relaxed. The suggested value for this option is "unlimited". Two functions of the system (ESDS Operator's account management tool and the developer account registration) may be impacted if the number of ESDS Hosted Users (including Operators, SEA Administrators, LEA Administrators, Ingestion Users, and Application Developers) exceeds this value:.

2.13 SimpleIDP

SimpleIDP is an identity provider that speaks the Security Assertion Markup Language (SAML). SimpleIDP handles admin account authentication using LDAP for account storage. Prior to deployment, you should be aware of the following points about SimpleIDP:

- SimpleIDP provides Web authentication for ESDS operators and other administrators when they use SDS administrative applications.
- All back-end authentication and authorization membership storage for SimpleIDP must be supplied by OpenLDAP.

2.13.1 Requirements

The following components are required for SimpleIDP

- Tomcat
- MongoDB
- OpenLDAP
- API
- sli.properties file (For more information, refer to the Configuration section.)
- Keystore and truststore (For more information, refer to Cryptographic Keys section.)

2.13.2 Configuration

The SimpleIDP is configured using the common sli.properties file. The following table defines the configuration properties that are used.

SimpleIDP Configuration Properties

Property	Description & Example
sli.simple-idp.issuer-base	Corresponds to the idp.id in the API. This is the identifier of the SimpleIDP (as a SAML identity provider) that the API (as a SAML service provider) needs to trust. Example: https://idp-server.example.com/simple-idp
sli.simple-idp.cot	Defines the Service Providers this SimpleIDP will trust. The value is a comma separated list of key=value pairs where the key is the issuer and value is the redirect endpoint URI. If this simple idp instance is running in Sandbox mode, it needs to add Production API service. Example: https://api.example.com=http://api.example.com/api/rest/saml/sso/post
sli.simple-idp.userSearchAttribute	LDAP attribute used to uniquely search for and authenticate users. Example: uid
sli.simple-idp.userObjectClass	LDAP class used to uniquely search for and authenticate users. Example: inetOrgPerson
sli.simple-idp.groupSearchAttribute	LDAP attribute used to uniquely search for a user's group membership. Example: memberUid
sli.simple-idp.groupObjectClass	LDAP class used to uniquely search for a user's group membership Example: posixGroup
sli.simple-idp.sandboxImpersonationEnabled	Used when switching to Sandbox mode as described in the Appendix. Should always be false in Production mode. Example: false
sli.simple-idp.sandbox.users	SmallDatasetUsers,Small Sample Dataset,MediumDatasetUsers,Medium Sample Dataset
sli.simple-idp.ldap.urls	A list of LDAP URLs for the available LDAP servers that can be used to authenticate against. Example: ldaps://ldap01.example.com:636,ldaps://ldap02.example.com:636
sli.simple-idp.ldap.base	Base DN for the LDAP server which defines where the SimpleIDP will be looking for the various structures that it requires. Example: dc=domain,dc=tld
log.path	Location where log files are written.

Property	Description & Example
	Example: /var/log/tomcat
sli.mongodb.host	Server hosting the mongo db instance and port used to connect to the host used to persist security events. Example: localhost:27017
sli.mongodb.database	Database where security events are persisted. Example: sli
sli.mongodb.keyencoding	Tells the MongoDB layer to replace the given patterns found in keys with the replace string. Example: sli.mongodb.keyencoding: \%:\%25,\\.:\%2E
sli.api.ldap.user	User account in the LDAP directory that has sufficient privileges to create and modify accounts. Example: cn=Admin_User,dc=domain,dc=tld
sli.api.ldap.pass	Password for the LDAP user indicated above that has permission to create and modify LDAP users. Example: 76F2FD74E0CEF4808FFD476EC604CFB4 Refer to the Cryptographic Keys section for details.
sli.encryption.ldapKeyAlias	The alias used in the keystore to retrieve the key used for encryption and decryption. Example: This value should match what was configured in the keystore. Refer to the Cryptographic Keys section for details.
sli.encryption.ldapKeyPass	The corresponding password of the alias. Example: This value should match what was configured in the keystore. Refer to the Cryptographic Keys section for details.

Note: The SimpleIDP can service multiple LDAP structures at the same level of ou=SLIAdmin, and provide authentication for additional unique organizational units as needed for testing. This can be helpful if you want to run a test tenant in production The IDP URL used to leverage this is:

https://<IDP FQDN>/simple-idp?realm=<unique LDAP organizational unit common name>

For example: https://<IDP FQDN>/simple-idp?realm=SLIAdmin.

For more information, refer to *Creating and Managing Realms* in the Administration Guide.

2.13.3 Installation

Use the following procedure to install the SimpleIDP service for the SDS:

1. Ensure that an sli.properties file is in the location matching the location defined as sli.conf within your Tomcat installation.
2. Copy the simple-idp.war file that is provided into the Tomcat webapps folder.

   Important! All ESDS java web applications should be deployed to the context specified by the .war filename. They should not be deployed to the root context in Tomcat.

   The SimpleIDP service should auto-deploy and be available within a few minutes.

2.13.4 Troubleshooting

If you have issues during SimpleIDP deployment, check the following troubleshooting points:

- Ensure that the .war file is readable by the Tomcat user, and that the Tomcat user is able to extract the .war file, creating a folder matching the first part of the name of the war file.
- Review the Tomcat catalina.out file to determine if the SimpleIDP application is failing to start.
- If the SimpleIDP successfully starts, but fails to recognize users by prompting them with appropriate login window when the users are re-directed to it, then a misconfiguration of the trust variable is likely. Review your configuration and ensure that your API name matches the name that is configured within the circle of trust.

2.14 Admin Tool

The Admin Tool provides a number of management functions. This tool provides functions such as admin account management, tenant creation, and tenant management.

2.14.1 Requirements

The components required for the Admin tool are:

- Rails
- API
- SimpleIDP

2.14.2 Configuration

To configure the Admin tool, you must perform the following steps:

Encryption Keys

1. Generate a key file with the script generateRailsKey.rb.
2. Generate encrypted LDAP password by using the keyfile generated in the previous step.
3. Generate encrypted SMTP password by using the keyfile generated in step 1.
4. Set the key file read permissions to the rails apps only.
5. Add the properties generated by the script to the relevant profile in $RAILS_HOME/config/config.yml:
   a. encryption_keyfile
   b. encryption_iv

Application Configuration

1. Update ldap_pass in $RAILS_HOME/config/config.yml.
2. Update the email_password field in $RAILS_HOME/config/config.yml.
3. Next, you will need to have a configuration file that represents your environment placed on top of $RAILS_HOME/config/config.yml.
   Note: Pre-deployment verification is always recommended.
4. Ensure that your configuration matches that of the API configuration file, which should be bootstrapped into the Mongo database upon API startup if no previous application entries are found in the database. Once the API has been started, and the application entry has been created in the Mongo database, you are ready to proceed.
5. Access the server webroot with a web browser. You will be re-directed to the API, which will re-direct you to the configured IDP. If validated correctly, you will then be redirected back to the API, and then back to the admin web application.

2.14.3 Installation

Follow the steps below to install the Admin Tool:

1. Extract the Admin software package to $RAILS_HOME.

   ```
   tar xvzf -C $RAILS_HOME admin-package.tgz
   ```

   Change your current directory to $RAILS_HOME.

1. Execute the following commands to install all of the prerequisite Ruby Gems and compile the assets:

   ```
   bundle install --deployment
   ```

   ```
   bundle exec rake assets:precompile
   ```

2. Modify the $RAILS_HOME/config/deploy/team.rb file to define the name of the environment. This will allow multiple configurations to be placed in the config.yml file, although only one configuration can be used at a time.

   ```
   server "admin.example.com", :app, :web, :db, :primary => true
   ```

   ```
   set :rails_env, "production"
   ```

2.14.4 Troubleshooting

If you encounter any internal server errors look at the production.log file in the admin-rails directory.

2.15 Dashboard

The Dashboard is a proof-of-concept application that leverages the rich ESDS RESTful API to demonstrate educator access to a broad range of student information. The Dashboard is implemented as a Java web application.

2.15.1 Requirements

The components required for Dashboard are:

- TomCat
- API

2.15.2 Configuration

Dashboard configuration is driven by the sli.properties configuration. Tomcat finds the sli.properties configuration file through the sli.conf environment variable.

2.15.3 Installation

To install the Dashboard, place the dashboard.war file into the $TOMCAT_HOME/webapps folder. If Tomcat is configured for hot deployment, replacing this file will update the application. If so, Tomcat should be restarted periodically or post-deployment.

Note: All ESDS java web applications should be deployed to the context specified by the .war filename. They should not be deployed to the root context in Tomcat.

2.15.4 Troubleshooting

There are two variables in the sli.properties file that are used by Dashboard for logging purposes:

- log.path is used to specify where the Dashboard log file (dashboard.log) resides.
- log.level is used to set the log level of slf4j logger used by the Dashboard.

The Dashboard creates a dashboard.log file in the location defined by log.path and that file's contents can be reviewed to identify problems. Ensure the Dashboard application is able to access the API.

2.16 Data Browser

The Data Browser is a web application that allows the end user to traverse data that is available through the RESTful API. The Data Browser is a read-only application, meaning that it cannot be used to modify any of the data viewed.

Designated administrators for education organizations will likely use the Data Browser to validate ingested data. As described in detail the ESDS Administration Guide, the Data Browser is one of the available applications from the administrator home page for users with permission to use it. Data Browser users are part of the ESDS hosted user directory and in the federated directories for state and local education agencies.

2.16.1 Requirements

The Data Browser requires the following components:

- Rails
- API

2.16.2 Configuration

Follow the steps below to configure the Data Browser:

1. Create a configuration file that represents your environment, named $RAILS_HOME/config/config.yml. Use the following example as a guide.

   ```
   production:

   api_base: https://api.example.com/api/rest

   redirect_uri: https://databrowser.example.com/callback

   portal_url: https://portal.example.com/headerfooter-
   portlet/api/secure/jsonws/headerfooter

   client_id: 10CharIden

   client_secret: 48CharacterRandomSecret
   ```

   - api_base - The URL where the application will need to connect in order to work with the API, as well as direct the user for authentication.
   - The below can be
   - client_id - A 10 character random string [a-z|A-Z|0-9] that represents the application for authentication to the API service. Be sure to use the encrypted client_id. This can be generated using the genAppKeys.rb script located in the opstools package.
   - client_secret - A 48 character random string [a-z|A-Z|0-9] that is used as a shared secret between the API and the application. Be sure to use the

encrypted client_secret. This can be generated using the genAppKeys.rb script located in the opstools package.

2.16.3 Installation

To install the Data Browser:

1. Extract the admin software package to $RAILS_HOME.

   ```
   tar xvzf -C $RAILS_HOME databrowser-package.tgz
   ```

2. Change your current directory to $RAILS_HOME.
3. Execute the following commands to install all of the pre-requisite Ruby gems and compile the assets:

   ```
   bundle install --deployment
   ```

   ```
   bundle exec rake assets:precompile
   ```

3 Appendix

3.1 Sandbox

The Sandbox environment is a Production-like environment for use by application developers. It contains the same infrastructure as the Production environment with minor changes to the runtime configuration.

Sandbox Mode vs. the Sandbox Environment

The SDS can run in two modes: Production and Sandbox. The Sandbox environment is a separate deployment that replicates the resources used in Production. This separate environment is deployed in Sandbox mode, and it remains in Sandbox mode throughout its existence. By using this isolated Sandbox environment, you can keep the Production environment operating separately and securely while still offering Sandbox services to application developers. When operated in sandbox mode, users will be able to register and create their own tenant. They can also choose to ingest a sample data set automatically or upload one manually. Additionally, the Sandbox mode provides impersonation functionality.

3.1.1 Requirements

The environment should be hosted on separate hardware to prevent interactions between Production and Sandbox systems.

Software requirements are also similar to that of Production. Specifically, the Sandbox environment needs Tomcat with ESDS applications running, Rails running the Admin tool and Data Browser, a MongoDB sharded cluster, and an OpenLDAP instance. These should be separate instances from Production.

3.1.2 Installation

To install the ESDS Sandbox environment, follow the same procedures as the Production environment installation with the following configuration differences:

- In the API node sli.properties:

  ```
  sli.sandbox.enabled=true

  sli.autoRegisterApps=true

  bootstrap.sandbox.createSandboxRealm=true
  ```

- In the admin application node config.yml:

  ```
  is_sandbox: true
  ```

- In the IDP server node sli.properties:

  ```
  sli.simple-idp.sandboxImpersonationEnabled=true
  ```