LUX Operations Guide

Last modified: 1 JUN 19

1 Introduction

The LUX Operations Guide describes operations, routine maintenance tasks, and common issues and fixes. This guide is meant to enable tier 1 operations support without detailed knowledge of LUX internals.

1.1 System Summary

The LUX System ingests event data (events), passes the events to event streams, and enriches and runs analytics on the events in the event streams. LUX Users write “rules” against the event streams and analytic outputs that generate alerts when a rule’s conditions are met. LUX passes the alerts to a browser display for various forms of visualization, as well as to a configured set of alert sinks.

1.2 System-wide User Roles and Responsibilities

This guide is directed toward the operation of the software and servers responsible for operation of the LUX System. The role responsible for this is termed the System Administrator or “SA”. The SA must have knowledge of the hardware, networking, operating systems, database server (mongo), web server (tomcat) and java virtual machines used by this system.

The SA must have access to these systems at the administrator or root level. This involves knowledge of the linux root password (or having sudo permission) as well as knowledge of the root or admin password for mongo. These passwords are set at installation time.

A second class of privileged user is one who can edit the configuration of the LUX engine and change the data sources and sinks that the engine interfaces, and one who can modify the configuration of enrichments and analytics of the engine pipeline. Such a user must have knowledge of the engine ingest, enrichment, analytic, alerter and outputter plugins. Unless the Engine Admin Console's file manager has been enabled, this user must also have write access to the Host's LUXEngine/EngineMain/data folder and folders below that.

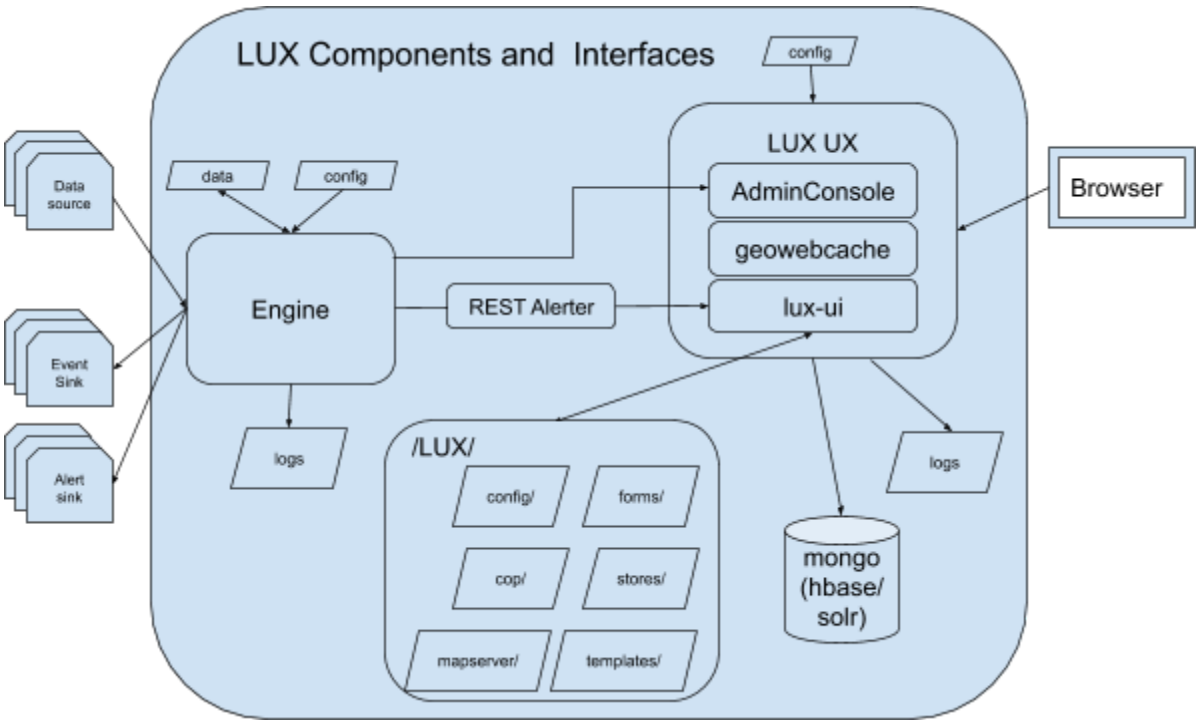
A third class of privileged user is one who is an admin on the LUX UI. A LUX UI Admin is denoted as such by being a member of the LUX Administrators group. A LUX UI Admin can effect changes on the LUX UI by editing the Rule Forms, Display Templates, Data Stores, etc. A LUX UI Admin must have knowledge of the specific form (schema) of the LUX forms, and also have knowledge of the freemarker template language. The changes that a LUX UI Admin can make can affect LUX system operations, and so provision must be made to backup the underlying data.

1.3 Help

Additional documentation may be found online at
https://icgsolutions.com/documentation2_8.html.

2 System Overview

The following figure shows all of the major LUX System Components and their interfaces.



3 Software

3.1 Engine

3.1.1 Operation

3.1.1.1 Starting the Engine

Start the engine by cd'ing into the engine's bin folder and running start-lite.sh. This will run the "full" engine in a normal operation mode.

```
$ cd lux/engine/bin
$ ./start-lite.sh
```

3.1.1.2 Stopping the Engine

Stop the engine (started as above) by cd'ing into the engine's bin folder and running stop-lite.sh.

```
$ cd lux/engine/bin
```

```
$ ./stop-lite.sh
```

3.1.1.3 Running the Engine as a Service

Edit the script **`/etc/init.d/luxengine`** and fill in the JAVA_HOME and LUX_ENGINE_HOME paths at the top of the script. Make sure the file has permissions 744. You can now control the engine as a service:

```
# service luxengine start
# service luxengine stop
# service luxengine restart
```

If supported by the system OS, the engine may be started by calling it via systemd.

```
$ systemctl start luxengine
```

You may also set the engine to start when the system boots.

```
# chkconfig luxengine --add
# chkconfig luxengine on
```

Or via systemd:

```
# systemctl enable luxengine
```

3.1.2 Folders and Files

3.1.2.1 lux/engine/EngineMain/data

This folder is the file system base folder for data files used by enrichments and analytics. Many enrichments and analytics name a data file or files in their configuration. The file system base for these data files is lux/EngineMain/data. The data files may be read-only or read-write, depending on their specific function. Refer to the appropriate plugin documentation for further details. While it makes sense to make a backup copy of the data files, be careful restoring a read-write data file. Read-write files may contain state that will be lost if the file is overwritten.

3.1.2.2 lux/engine/EngineMain/data/conf

The engine is primarily configured by two files located in this folder - engine.properties, and lux.properties. There are other engine configuration files located in this folder, but these two files are the principal configuration files of interest.

The engine.properties file configures the alerters and track manager. The lux.properties file configures the engine's connections to the tomcat lux webapp and certain features of the engine. Refer to specific engine plugin documentation for details.

3.1.2.3 lux/engine/EngineMain/data/logs

This folder contains a set of rolling log4j log files of the engine operations. The most recent file is always log4j.log. You can find an engine start in a log file (if the log file contains the engine start) by searching from the bottom of the file for "Access" (as part of the string "Access control disabled.") This string occurs very near the engine start in the log file ("Initialized logging from logging.properties" is the true start).

3.1.3 Network Connections

The engine makes any number of connections to data sources, event sinks and alert sinks, as configured in the engine.properties and ae.xml files.

The engine makes the following connections to the other internal components of the LUX system.

Name	Description	IP address and protocol
Engine REST	The engine's access to REST services offered by the LUX UI. These are configured in lux.properties.	https://luxui/lux/rest/v2/engine tcp or ssl as configured JWT authorization
Engine Status	The engine's status. This connection is configured in engine.properties.	https://luxui/AdminConsole/rest/

3.1.4 Monitoring

The primary monitor of the engine's operation is the Engine Admin Console. This is accessible to a LUX Admin User from the UI's "LUX Administration" menu item, "Engine Administration Console" link. In general, green is good and red is bad. If the engine pipeline backs up, you will see the queues go red from back to front. Eventually, back pressure will stop or seriously rate limit the ingestors.

The secondary monitor of the engine's operation is the Engine log file in EngineMain/data/logs. `tail` this file and look for evidence of a heartbeat. Alternately, view the file and search for exceptions and errors.

3.1.5 Maintenance

The engine's data folder and conf folder should be backed up. The files should be selectively restored only if there has been a data loss or data corruption issue. There are no run-away files (files that grow ever larger) to worry about, unless the engine has been started in a non-standard way. Check the bin folder for a nohup.out file. If that file exists and is large, restart the engine in the standard way.

3.3 Tomcat

Tomcat is the web server for two LUX webapps - LUX UI & the AdminConsole.

3.3.1 Operation

Tomcat should have been simultaneously installed and configured during the installation of the LUX UI. Refer to the installation guide.

Checking that tomcat is running via the LUX UI using ps and grep:

```
$ ps -ef | grep catalina
```

3.3.2 Files and Folders

3.3.2.1 Configuration File

The tomcat configuration is in **`/usr/local/lux/ui/conf/server.xml`**. The primary configuration is the Connector, which is typically configured for SSL as an https port with a keystore validating the server name.

If running behind a proxy, the following changes will need to be made to /usr/local/lux/config/context.xml to permit LUX to accurately detect incoming connection source IPs.

*Note: this will not work if running behind an anonymous proxy.

```
<Valve
  className="org.apache.catalina.valves.RemoteIpValve"
  internalProxies="place.proxy.ip.here"
  remoteIpHeader="x-forwarded-for"
  proxiesHeader="x-forwarded-by"
  protocolHeader="x-forwarded-proto"
/>
```

3.3.2.2 Log File

The principal tomcat log file is **`/usr/local/lux/ui/logs/catalina.out`**. This file is unmanaged and should be monitored for size. Alternatively, this log can be managed through **`logrotate`** configuration. Problems with the LUX UI will often show up in this file as errors or exceptions.

3.3.3 Webapps

3.3.3.1 lux

The lux webapp is the principal LUX UI application. The service commands and general operation are similar to the Engine Admin console noted above.

3.3.3.1.1 LUX Operation

LUX UI should have been installed and configured during the baseline LUX installation. Refer to the installation guide.

Checking that lux is running using ps and grep:

```
$ ps -ef | grep lux
```

Manually starting and stopping tomcat is accomplished by executing commands on the lux service:

```
# service lux start
# service lux stop
```

Or alternatively:

```
# systemctl start lux
# systemctl stop lux
```

3.3.3.1.2 Files and Folders

The lux webapp is in `/usr/local/lux/ui/webapps/lux`. The initial installation creates the `lux.war` file in webapps. The next time tomcat runs, it expands `lux.war` into the lux folder. At that time, a system admin can configure lux, by modifying the files in `/usr/local/lux/ui/webapps/lux/WEB-INF/classes` and `/usr/local/lux/ui/webapps/lux/WEB-INF/spring`. Once these files are configured, the system admin should make a copy of these files outside of the `/usr/local/lux/ui` folder, as the classes and spring folders will be replaced by another installation of LUX from RPM.

In addition, the admin should copy certain folders from `/usr/local/lux/ui/webapps/lux/` to `/usr/local/lux/`. These are:

Name	Access	Description
`config`	RW	Contains the configuration for the entities and layers.
`cop`	RW	Contains the configuration for the Common Operating Picture.
`forms`	RW	Contains the xml files that define the rule forms.
`mapserver`	RO	Contains the xml files that define the map servers used.
`stores`	RO	Contains the xml files that define the data store values (e.g., the display names and values for UI combo boxes and other name/value pairs)..
`templates`	RO	Contains the Freemarker Template Language (ftl) files that define the alert details display html generation.

On a subsequent RPM install or update, the RPM will overwrite the files in the folders in `/usr/local/lux/ui/webapps/lux`, but the files in use will be safely held in `/usr/local/lux/`… Because the RPM update may have updated these files, the admin should follow any migration instructions that come with the update RPM.

3.3.3.1.3 Network Connections

The lux webapp makes network connections to the following LUX internal components.

Name	Description	Ip address and protocol
Alerts via ActiveMQ	The UI's non-default alert queue input from ActiveMQ if configured. The default alerter uses REST.	<luxuiHostname>:61616 tcp or ssl as configured Client cert or name/password
Alert database	The mongo (or hbase/solr) database for storing alerts and related data (e.g., AOIs).	<luxdbHostname>:27017/lux
Engine REST	The UI's REST services specifically for the LUX engine.	https://<luxuiHostname>/lux/rest/v2/engine tcp or ssl as configured JWT authorization
General REST	The UI's REST services for developmental use. See the LUX Web Services Guide.	https://<luxuiHostname>/lux/rest/v2/ tcp or ssl as configured JWT authorization

https://<luxuiHostname>/lux	The LUX UI	https
https://<luxuiHostname>/AdminConsole	The LUX Engine Admin Console	https

3.3.3.2 AdminConsole

The AdminConsole webapp displays the LUX engine status.

3.3.4 Monitoring

The operation of the lux (tomcat) web server and webapp container is best monitored by viewing the catalina.out log file. This file should evidence a heartbeat. Problems are indicated by errors or exceptions.

3.3.5 Maintenance

See notes contained in lux webapp discussion above about files. The principal file that can grow large and require maintenance is /usr/local/lux/ui/logs/catalina.out. This file should be recycled occasionally as it gets large (>100MB) by truncating the file, restarting the lux webapp, or by configuring logrotate.

3.4 Mongo (or HBase/Solr)

LUX uses a mongo database (or other databases, e.g., hbase/solr) to store alerts and related information such as AOIs.

3.4.1 Operation

Mongo should have been configured during installation to run as a service and to start at boot time. Refer to the installation guide.

Checking that mongo is running using ps and grep:

```
$ ps auxw | grep mongod
```

Manually starting and stopping mongo is accomplished by the service commands:

```
# service mongod start
# service mongod stop
```

3.4.2 Files and Folders

3.4.2.1 Database files

The mongo database is in the folder configured for dbPath in /etc/mongod.conf, typically `/var/lib/mongo/`. This database folder contains a number of files managed by mongod. Depending on the number of alerts stored, the database folder can get big.

The retention for alert files in mongo is specified by a line in the `setupMongo.js` script that is run at installation:

```
db.alerts.createIndex({“createdDate” : 1},
{“expireAfterSeconds”: 432000})
```

The default sets the expiration to 432000 seconds, or 5 days. This can be changed, most easily at install time, but it is possible to change it after the installation.

3.4.2.2 Log files

The mongo log file is configured in `/etc/mongod.conf` by `systemLog: path`, and is `/var/log/mongodb/mongod.log` by default. This is an unmanaged file that can get large and so should be monitored. If the log becomes too large, consider configuring logrotate and truncate the log.

3.4.3 Monitoring

Monitor the operation of mongodb by tailing the mongo log file.

3.4.4 Maintenance

Other than managing the log file, no maintenance should be required. A few simple commands that you can execute on the database server:

```
$ mongo -ulux -plux lux
> show collections
> db.alerts.count()
> quit()
```

4 Hardware

4.1 Operations

LUX is intended to operate 24x7, monitoring event sources and generating alerts. Consequently, the hardware is required to operate continuously.

4.2 Monitoring

The problems that typically arise are network issues (loss of connectivity, loss of DNS) and disk full situations. Ideally these should be automatically monitored.

4.3 Maintenance

Maintenance may be required to remove log files of the software as some logs grow over time. This is specifically discussed in the software sections for each server.

5 Operations

5.1 Backup/Restore

The following should be backed up and only restored selectively.

Component	Folder	Notes
LUXEngine	`LUXEngine/EngineMain/data/`	Backup files whenever engine configuration is changed.
LUXEngine	`LUXEngine/EngineMain/data/conf`	Backup files whenever engine configuration is changed.
LUXUI	`/usr/local/lux/ui/`	Backup after installation and if configuration changes
LUXDB	`mongo`	Backup all relevant dbs and collections.

6 Operational Procedures

6.1 System Restart

Occasionally it may be necessary to restart the LUX system, whether in part or in entirety. The effect of restarting the various components is shown in the table below.

Component	Reason	Impact
Engine	configuration change	Event loss while engine is off
UI/Tomcat	UI performance or flakiness Alert queue not being consumed	Total UI loss. Alerts will be dropped if using the REST alerter.
MongoDB	maintenance	UI alert display loss. Tomcat will emit errors in the log.

Generally speaking, the system will recover from an individual component being restarted. If a full system restart is determined to be necessary, the best procedure is:

1. Stop LUXEngine
2. Stop LUX
3. Stop MongoDB
4. Start the MongoDB
5. Start LUX
6. Start LUXEngine

6.2 Diagnostics

Suggestions for diagnosing system problems.

Identifying the root cause.

Total LUX failure:

1. Disk full? Use df on the servers to see if a server has run out of disk.
2. Network failure? Use ping and telnet to see if the servers are connected
3. Process failure? Use “ps -ef | grep …” to see if required process is running.

Data source failure:

1. Use ping or telnet to see if network connectivity to data source

Component	Tool
LUXEngine	Use "`$ ps auxw|grep engine`" to see if the engine is running on server Use "`df -h`" to see if disk is available on server Use Engine Admin Console to see if pipelines are running Tail Engine log file and look for exceptions
ActiveMQ (if the REST alerter has been replaced).	Use "`$ ps auxw|grep activemq`" to see if activemq is running on server Use "`$ df -h`" to see if disk is available on server View alerts queue on activemq admin page to see if alerts are flowing Tail activemq log file and look for exceptions
UI/tomcat	Use "`$ ps auxw|grep lux`" to see if tomcat is running on UI server Use "`$ df -h`" to see if disk is available on UI server Tail tomcat log file (`catalina.out`) and look for exceptions
mongod	Use "`$ ps auxw|grep mongod`" to see if mongod is running on db server Use "`$ df -h`" to see if disk is available on db server Tail mongod log and look for errors Use "`$ mongo -ulux -plux lux`" to see if database is available