EFM Linux Configuration Guide

Kinetic - Edge & Fog Processing Module (EFM) 1.7.0

Revised: July 9, 2019

Table of Contents

Introduction

The Cisco Edge and Fog Processing Module (EFM) allows you to create a reliable data communications messaging system on top of your data networking infrastructure. This system provides data delivery and allows you to rapidly deploy applications, where needed, that can be at the edge or fog or in the data center. The EFM is an open platform that allows for the addition of micro services or applications by anyone, allowing for unlimited capability and growth by adding software components that optimize the results of the application, system, or outcome.

The EFM addresses the complexity of building an enterprise-ready scalable data messaging system upon which one or many applications can reside. The EFM comes with a series of tools to manage the system, the EFM system administrator, and the EFM system monitor.

Features and functions

The system's key capabilities include:

- A framework for edge and fog processing. High performance.
- Reusable micro services for collecting data from, and providing control over, devices and machines, as well as processing the data prior to delivery to its destination.
- Different options for reliable transport of data through the system, encompassing both batch and real-time streaming options.
- Flexible mechanisms for integration with IT systems, reporting, and analytics.
- An architectural framework to extend fog processing to multiple tiers: east west (fog to fog) and north south (hierarchical processing leveraging network topology).
- Easy-to-use GUI tools to simplify development, deployment, and operation for all aspects of the system.
- A pervasive control paradigm and flow of information back to micro services, devices and machines for management, control, optimization, and specific actions.
- A completely open and polyglot system where third parties can provide devices, processing storage, software modules, analytics, applications, or any combination thereof.

This is the technology that makes IoT possible, and leads to faster industry adoption of the IoT vision.

The Edge and Fog Processing Module components

EFM Message Broker (DART Broker)	Provides reliable and flexible data delivery between any devices and micro services. The sources can be devices like devices or other micro services and consumers can be micro services or user applications. The EFM Message Broker is a small footprint component working with other brokers to form a message bus. This is also know as the EFM server.
EFM C++ Broker	A multi-threaded high-performance message broker with very low footprint in order to leverage the multi-core capability of different platforms.
EFM Data Flow Editor	Defines message paths between devices and micro services.
EFM Data Flow Engine	Executes message paths between devices and micro services. We recommend installing it adjacent to the EFM Message Broker for performing data transformation and inputting sources that are not in the canonical data format of the system.
EFM System Administrator	Configures and manages the message broker and micro services.
EFM System Monitor	A standalone tool for operators to obtain real-time functional status of a deployed solution.
Cisco ParStream (Historian Database)	Purpose-built database for scale to handle the massive volumes and high velocity of IoT data as well as analytics at the Edge.
Links	DQL Link – DSA Query Language System Link – System Information ParStream Link – ParStream Historian Database JDBC Link – Java Database Connectivity Alarming Link Asset Link – EFM Manager Device Simulation – demonstration device simulator
Smart License Tool	The Smart License Agent client that allows system users to manage license registration
EFM Manager	Detects and manages Assets and brokers throughout the EFM Messaging system
User Management	The User management is a component that allows managing users, groups, permissions and roles for the EFM Web based components.
DGLux5	Data Visualization Tool

IOT-DSA Message Brokers differences

The Cisco Edge and Fog Processing Module allows for the installation of two distinct message brokers.

- IOT-DSA DART Message Broker - Supports message broker functions as well as web based UI projects that include the EFM System Administrator, EFM Data Flow Engine and EFM System Monitor.
- IOT-DSA C++ Message Broker - Supports message broker functions, but web based UI projects are not supported. For Dataflow editing to be performed on the dataflow link running with a C++ broker, a connection must be performed via an upstream or downstream dataflow editor that does support the UI interface (DART broker) and navigate the node tree until the dataflow link is found.

Hardware requirements

EFM Message Broker (DART Broker) EFM Data Flow Engine DQL Link System Link ParStream Link	Red Hat Linux 7, CentOS 7, 1GB RAM, Windows 2016 Server, 10 GB HD - Recommended on the same system/VM
EFM C++ Broker	Red Hat Linux 7, CentOS 7 or Ubuntu 16.04
EFM Data Flow Editor	Automatically installs with EFM Message Broker and EFM Tools Runtime Engine. Access via a web browser
EFM System Administrator	Project installs on the same system as the EFM Dart Message Broker and EFM Tools Runtime Engine. Accessed via a web browser
EFM System Monitor	Project installs on the same system as the EFM Dart Message Broker and EFM Tools Runtime Engine. Accessed via a web browser
User Management	Installs on the same system as the EFM Dart Message Broker. Accessed via a web browser.
Cisco ParStream (Historian Database)	Red Hat Linux 7, CentOS 7 , 6 CPU cores with 2GB RAM per core, 500 GB HD
Smart License Agent Tool	Redhat Linux 7, CentOS 7, with 1GB RAM, 10 GB HD.
EFM Manager	4GB RAM, 10 GB HD - Recommended on the same system/VM
DGLux5	Red Hat Linux 7, CentOS 7, 1GB RAM, Windows 2016 Server, 10 GB HD

EFM components' protocols and ports

The protocols and ports used by the EFM Broker, EFM Manager and the EFM Historian Database. The port values are configurable after installation.

TCP Port No.	Description
443	Default https value for inbound connections to the Message Broker (DART Broker), System Administrator, System Monitor, Dataflow Editor, EFM Manager and User Management. This port is used when EFM Manager and the proxy are installed and configured for the components listed.
8080 and 8443	Default http and https ports for inbound connections to the Dart Message Broker. Note that the EFM Server when using the EFM Server, uses the proxy server to map to port 443 above.
EFM Administrator Defined for ParStream	Connection to ParStream Historian Server (when installed) from a local or remote ParStream DSLink. This includes the registrationPort and server(s) port in the server INI file.
9080 and 9443	DGlux5 message broker and user interface port defined in the server.json configuration file. See the EFM Message Broker server.json configuration file for parameter details.
8100 and 8463	Default http and https ports for inbound connections to the C++ Message Broker on Linux only

Additional ports post installation

The EFM DSLinks are microservices. They may expose additional protocol ports that are listening for incoming connections. It is necessary to verify these ports in the DSLink documentation and configure the host to allow the incoming connections as desired. For example, the MQTT DSLink provides an optional server that can listen on port TCP 1883 to MQTT clients. If the MQTT Server DSLink functionality is desired, the host must allow for the proper firewall access for this to receive the connections.

Licensing installation and requests

This product uses the Smart License Agent Tool (for Nodes and Devices) to manage the corresponding smart licenses. After installation, refer to the Kinetic – Edge and Fog Processing Module Smart License Agent User Guide.

Secure Mode operation

The EFM can operate in Secure Mode to enhance the security features available for the EFM Dart message broker and web server. Secure Mode provides these following enhacements:

- HTTPS Strict Transport Security (HSTS), which automatically redirects inbound http connections to https for message broker and web traffic
- System dslink cannot execute “system command”
- Login page won't allow browser to remember password
- Prevents the pages from being embedded in iframes
- Prevents the command action that allows shell exection by the System Link

Secure Mode is configured by default during the installation of the message broker or placing the hidden file “.secureMode” in the efm_server or dglux_server directory.

Using Secure Mode HSTS only affects inbound connections; outbound http and https connections are still supported.

Basic component installation scenarios

The EFM has many components, allowing for a diverse manner of architecting a solution. While no single deployment architecture exists, we will explain the basic deployment scenarios.

The first and simplest installation is a complete install on a single host. This scenario allows for development and testing, but is not typical for a production system.

The EFM architecture can be divided into six main building blocks and are typically on different hosts. We can separate them as follows:

- EFM Smart License Agent(s) – this is the only node in the system that is required to connect to Cisco.com, either directly or via the Smart License Satellite. The Smart License Agent allows for license activation, revocation, and renewals. Without license activation or periodic communication over the Internet to Cisco.com, the EFM is out of license compliance.
- EFM (DART) Message Broker with core DSLinks (DQL, System, ParStream, and Dataflow engine). The message broker is deployed on all nodes and is responsible for communications between all components across the system. If the EFM tools such as EFM System Administrator, Dataflow Editor, System Monitor or EFM Manager then the Dart message broker is required, Otherwise, it is recommended the installation and use of the C++ Message broker.
- EFM C++ Message broker is a A multi-threaded high-performance message broker with very low footprint in order to leverage the multi-core capability of different platforms. No user interface to

service the EFM System Administrator, Dataflow Editor, System Monitor or Asset Manager; if needed install the Dart Broker.

- A System Administrator node, typically one per system. The EFM System Administrator is the administrative console that allows for configuration and operation of the EFM System components. A message broker is installed on this node in addition to the EFM System Administrator project.
- A System Monitor node, also one per system. The EFM System Monitor allows operators to view the connectivity and operations of the message brokers and DSLinks deployed throught the system. The System Monitor is used as an operations console.[1]
- A Historian node, is deployed through a system to persist telemetry into a database. This is an add-on to a message broker. The ParStream DSLink is used to communicate between the message broker and the ParStream historian database.
- The EFM Manager is a component of the of Cisco Kinetic EFM that detects and manages assets and brokers throughout the EFM message system. The Device DSLinks expose the assets that are required to have been specified using the Cisco EFM Device Object Model Standard structure to allow for auto-discovery. The resulting list of approved assets not only appears in the EFM Manager graphical interface, but also creates a node structure in the EFM data path that can be used as input for other applications.

The System Administrator and System Monitor use the message brokers for communications to all the nodes and system dslinks. Message broker-to-broker communications need to be set up first before other tasks can be performed.

Securing the Installation

Every install of a Cisco Kinetic EFM instance will have to meet specific requirements for performance and security. It is generally advisable, to configure the underlying platform Linux OS as tight as possible by minimizing the number of amount and privileges of processes running and services offered. Suggested is adherence to general hardening guidelines as provided by the NSA hardening guide collection at https://www.nsa.gov/ or platform specific formulations. To enable educated decisions, when the grade of security impacts performance, and where to strike a balance acceptable for the local install, the sections in this guide offer helpful information and relations.

For additional information on hardening the underlying operating system some additional references are:

Red Hat: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/pdf/Security_Guide/Red_Hat_Enterprise_Linux-6-Security_Guide-en-US.pdf

[1] The operation of the System Monitor acts as a subscription to the System link in every broker for telemetry. If certain parts of the EFM system are bandwidth restricted, the use monitor rules should be reduced.

NSA hardening guide – https://www.nsa.gov/what-we-do/research/selinux/ for information on Security Enhanced Linux. See also SELinux SELinux in the RedHat Enterprise Security Guide above.

General concepts

Defining an EFM Administrator User per node for UI tools

It is important to note that we do not define default username and passwords for EFM tools such as EFM System Manager, Dataflow Editor, System Monitor or EFM Manager. The first user that is defined at install becomes the administrator of that node. After the install, using the System Administrator, the additional users may be added. At least one user requires administrative priviledges for that node.

Note that the C++ broker does not have an administrator user since it does not support the UI tools.

Defining a non-root Linux account for installation and operation

As a Linux security best practice, the installation RPM creates a non-root account for installing and operating the EFM. If the account does not exist, it will create the user “efm” and group “efm”. All examples throughout the documentation will reference this user name.

Download and Unzip the EFM package.

The software should be downloaded from CCO at www.cisco.com under “Support and Downloads.”

Unzip the image:

```
$ unzip EFM-1-7-0.zip
```

Change into unzipped folder:

```
$ cd EFM-1-7-0
```

EFM Component Installation Instructions for CentOS/RHEL

The EFM components for CentOS7 and RHEL7 are delivered in the form of RPM packages. The RPM packages are installed, updated and uninstalled using the YUM tool. During the installation of the RPM packages using YUM, third party dependencies are checked and if required downloaded and installed.

EFM RPM packages are delivered as a ZIP archive containing all RPM packages. This archive has to be installed on the target system before the RPM packages it contains can be installed with YUM.

Install local EFM repository of RPMs

Copy EFM archive from corresponding platform subfolder of the EFM-1-7-0.zip file to the target system.

The archive name follows the format repo-<major-version>.<minor-version>.<patch-version>-<build-number>.zip, i.e. " repo-1.7.0-34.zip"

The following steps must be carried out to define the local EFM repository.:

Step 1	`$ unzip repo-1.7.0-34.zip`	Unpack the archive in the **"user"** home directory. This creates the directory " /repo" , which contains all EFM RPM packages

Step 2	`$ cat << EOF > efm.repo`
	`[local]`
	`name=EFM Repository`
	`baseurl=file:///home/`**`user`**`/repo`
	`gpgcheck=0`
	`enabled=1`
	`EOF`
	
	`$ sudo mv efm.repo /etc/yum.repos.d/efm.repo`
		Register the local EFM archive for yum. Therefore create a text file with the following content under the path " /etc/yum.repos.d/efm.repo" (the directory " /etc/yum.repos.d" already exists)
		
		If you used a different folder in Step #1, adapt the path for *baseurl* correspondingly.

Step 3	`$ sudo yum updateinfo`	Update YUM in order get the latest package versions. Now yum is prepared for the installation of the EFM RPM packages.

Prerequisites for installation of EFM Authentication Server

The new EFM Authentication Server which enables Single Sign On for all EFM UI components, has additional dependencies which require the following installation steps to be executed before the RPM can be installed.

Step 1	`$ sudo yum install yum-utils`	Install yum-utils

Step 2	`For CentOS-7: $ sudo yum-config-manager --add-repo https://openresty.org/package/centos/openresty.repo`
	
	`For RHEL-7: $ sudo yum-config-manager --add-repo https://openresty.org/package/rhel/openresty.repo`
		Add repository openresty definition

Step 3	`$ sudo yum install -y openresty`	Install openresty

For RHEL the following addional steps must be exectuted.

- Set the IP hostname mapping in /etc/hosts
- Set SELinux mode to permissive:

```
$ sudo setenforce permissive
```

To make the the enforcement permanent for the next reboots, /etc/selinux/config must be modified by the administrator.

Install EFM Component RPM packages

In order to install an EFM Component RPM package in CentOS and RHEL invoke:

```
sudo yum install -y <package-name>
```

For example:

```
sudo yum install -y efm-server
```

This commnd installs the efm-server in the directory `/opt/cisco/kinetic/efm_server`. A user `efm` will automatically be created and will be owner of the corresponding installation files.

`/opt/cisco/kinetic` is the common home directory for all installed EFM packages.

Each package is installed in a separate subfolder, with the exception of additional modules for the EFM server, e.g the efm-server-admin or efm-server-monitor, which will be installed below the " efm_server" folder.

Required dependencies are automatically solved when the host is connected to the Internet, e.g. if System Administrator is installed using the command. If the host does not have connectivity to the RedHat or CentOS repositories, see the section “Preparing Third Party Packages for hosts without external repository access”.

```
sudo yum install -y efm-server-admin
```

The efm-server will also automatically be installed, if it is not already present.

Some packages also configure systemd services which can be used to start and stop the components. These services are by default disabled, but can be easily enabled and started.

RPM Packages

RPM Package name	Description
efm-applications	Package that installs EFM Server (DART message broker), System Administrator, Dataflow Editor, EFM Manager and EFM Authentication Server
efm-authentication-server	Package that installs EFM Authentication Server
efm-cpp-broker	Package that installs C++ Message Broker
efm-dart-runtime	Package that installs DART SDK
efm-demo	Metapackage for installing all of the above packages (not recommended for production use, for demo purposes only)
efm-dglux	Package that installs DGLux
efm-licensing	Package that installs EFM Licensing Tool
efm-manager	Package that installs EFM Manager
efm-server	Package that installs EFM Server (with DART message broker)
efm-server-admin	Package that installs EFM Sysem Administrator
efm-server-monitor	Package that installs EFM Sysem Monitor
efm-simulation-dslink	Package that installs EFM Simulation Link
ioxclient-1.7.0	Package that installs Cisco Linux CLI client
parstream-authentication	Package that installs Cisco ParStream Authentication
parstream-database	Package that installs Cisco ParStream dabase
parstream-client-examples	Package that installs Cisco ParStream client examples
parstream-database-examples	Package that installs Cisco ParStream dabase examples
parstream-jdbc	Package that installs Cisco ParStream jdbc
parstream-sii	Package that installs Cisco ParStream sii
parstream-ssl	Package that installs Cisco ParStream ssl

Preparing Third Party Packages for hosts without Internet repository access

For environments with limited access or security requirements that do not allow for access to RedHat or CentOS repositories via the Internet, the installation of dependencies will fail. As a workaround, the following steps will allow for the EFM RPMs to properly install.

It will be necessary you create a local repository with all the needed thirdparty packages. These packages will be transferred to the host machine where the EFM will be installed (in addition to the repo.zip file from the EFM package).

On a host machine with the same Redhat or CentOS operating system as the EFM candicate host and with access to the internet available repository, the following steps need to be performed. The following steps must be carried out for this as user 'root' or sudo:

Step 1	`$ sudo yum install yum-utils`	Install yum-utils

Step 2	CentOS:
	`$ sudo yum-config-manager --add-repo https://openresty.org/package/centos/openresty.repo`
	RHEL:
	`$ sudo yum-config-manager --add-repo https://openresty.org/package/rhel/openresty.repo`
		Install the `openresty` repo

Step 3	`$ sudo yum install -y createrepo yum-utils`	Install the `createrepo` and needed tools
Step 4	`$ sudo yum install -y --downloadonly --downloaddir=/efm-thirdparty efm-demo`	Download all dependencies not yet in efm repo

Step 5	`ls /efm-thirdparty/*`
	
	For example:
	
	`/efm-thirdparty/copy-jdk-configs-3.3-10.el7_5.noarch.rpm`
	`/efm-thirdparty/dnsmasq-2.76-5.el7.x86_64.rpm`
	`...`
	`/efm-thirdparty/sysvinit-tools-2.88-14.dsf.el7.x86_64.rpm`
	`/efm-thirdparty/tzdata-java-2018f-2.el7.noarch.rpm`
		Now you should have all needed packages in `/efm-thirdparty`. Verify using ls.

Step 6	`$ sudo createrepo /efm-thirdparty`	Create a repository from these packages This will create the repository metainformation needed for yum.

Move the repository directory and contents to the target machine (i.e.via UsB stick, CD/DVD, etc.). The rest of the installation can be performed on the candidate host machine without internet access. On the EFM candidate host, perform the following as root so the new repository can be used for the EFM components installation. It is assumed that the repository is in /efm-repository.

Step 1	`$ cat << EOF > efm-thirdparty.repo`
	`[efm-thirdparty]`
	`name=EFM Thirdparty Repository`
	`baseurl=file:///efm-thirdparty`
	`gpgcheck=0`
	`enabled=1`
	`EOF`
	
	`$ sudo mv efm-thirdparty.repo /etc/yum.repos.d/efm-thirdparty.repo`
		Register the local third party repo for yum. Therefore create a text file with the following content under the path " /etc/yum.repos.d/efm-thirdparty.repo" (the directory " /etc/yum.repos.d" already exists)
		
		If you used a different folder for the repository, adapt the path for *baseurl* correspondingly.

Step 2	`$ sudo yum updateinfo`	Make the metadata available
Step 3	`$ sudo yum install efm-demo`	Install (the needed packages)

Troubleshooting tip, check all available repos with "`yum repolist all`". The installation will require the efm and efm-thirdparty repos.

Prerequisite to allow installation of Links from Git repositories

The EFM System Administrator allows for installing links from Git, or the system link in the (/sys/links/Install Link/from Git). The git command line application must be installed for this function to properly operate.

```
$ sudo yum install git
```

Installation of EFM Component Applications (efm-applications)

Some of the provided RPMs bundle several components. One of these is the efm-applications. This will install the following:

- EFM Server
- System Administrator
- Dataflow Editor
- EFM Manager
- EFM Authentication Server

Step 1	`$ sudo yum install -y efm-applications`	Install the following EFM components: • EFM System Administrator • Dataflow Editor • EFM Manager • EFM Authentication Server • Dart Runtime • EFM User Managemment
Step 2	`$ sudo /opt/cisco/kinetic/efm_manager/bin/configure_efm_applications.sh`	To complete the installation of EFM Authentication Server run the following script must be run.

The following systemd services are automatically added and started. They are not automatically endable for restarting:

```
efm-authenti cation-server.service
efm-manager.service
efm-server.service
openresty.service
```

A User `efm` is automatically created, if it doesn't exist, and owns the installations and is configured to run the services.

With completion of this step EFM Server and Applications are automatically started (via the registered services). See the "Newly Defined Services" section to enable services upon reboot.

To connect the central login page for all components, using a web browser go to

`https://[SERVER IP ADDRESS]:443`[2]

and log in with the admin user credentials you just registered.

Installation of EFM Monitor

Currently it is not recommended to run EFM System Monitor on the same instance as your remaining applications in production environments. For simple testing and demo purposes, you may install the System Monitor on the same instance though.

To install EFM System Monitor run the following command:

```
sudo yum install -y efm-server-monitor
```

User `efm` is automatically created and owns the installation.

The EFM Monitor is a project that runs with the EFM Server and must be running.

Installation of DGLux5

To install DGLux5 run the following command:

```
sudo yum install -y efm-dglux
```

To run EFM Server and DGLux5 Server in parallel, both need to be owned and executed by different system users. The installation process takes care of corresponding user creation and permissions. To run EFM applications and EFM Server the system user `efm` is used. To run DGLux Server the system user `dglux` is used. The user dglux is automatically created, owns the installation and is configured to run the corresponding service.

The following systemd service is automatically added:

```
efm-dglux.service
```

For improved security reasons, no default UI user is created. To create a DGLux user go to folder

```
/opt/cisco/kinetic/dglux_server
```


[2] Note that RHEL and CentOS have the firewall service on by default. See the Linux Firewall Issues section for recommendations.

and run the following command:

```
./bin/users.sh add -u [USERNAME] -p [PASSWORD]
```

If this user shall be configured as superuser the command needs to be:

```
./bin/users.sh add -u [USERNAME] -p [PASSWORD] -s
```

Starting DGLux:

```
sudo systemctl enable efm-dglux
```

To connect DGLux5, use a web browser and proceed to

```
https://[SERVER IP ADDRESS]:9443
```

and log in with the admin user credentials you just registered.

EFM Server and DGLux5 licensing

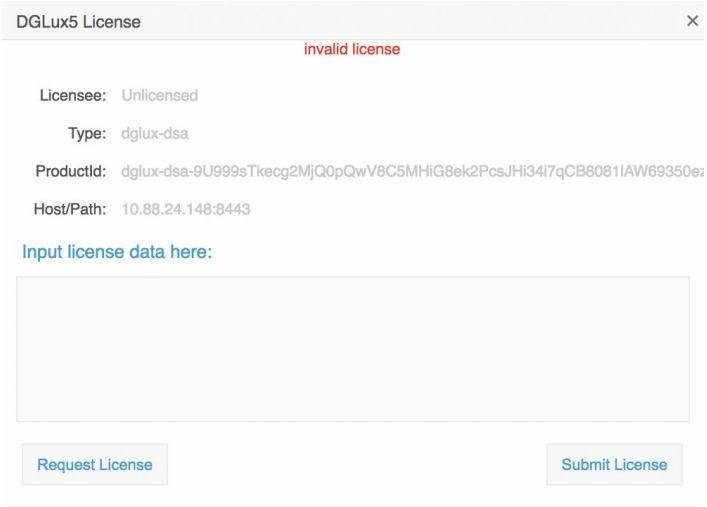
The EFM Server, upon startup, installs a .dglogik licensing file in the home directory of the efm user. This license enables the EFM to function as a unlimited node and supports the specific EFM projects: EFM System Administrator and EFM System Monitor. DGLux5 at startup also installs a .dglogik licensing file in the home directory of the dglux user.

Starting the appropriate services assures that each application is run as the correct user and licensing overlap is avoided.

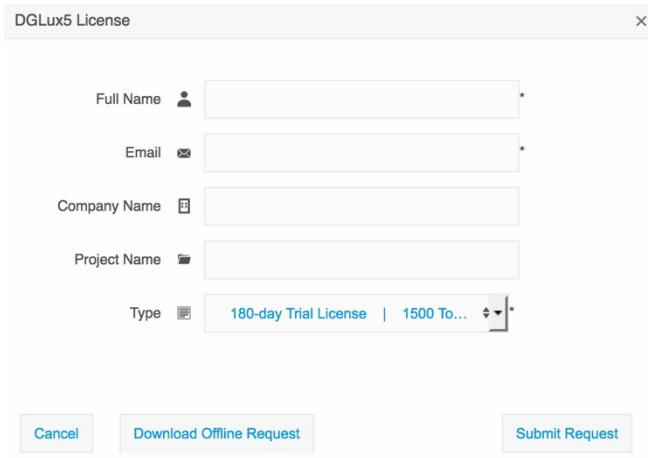
The DGLux5 also requires a license to operate. The trial license expires automatically after the 6-month trial. This license is not renewable and require the purchase of a additional license for operation.

Requesting a DGLux5 6-month trial license

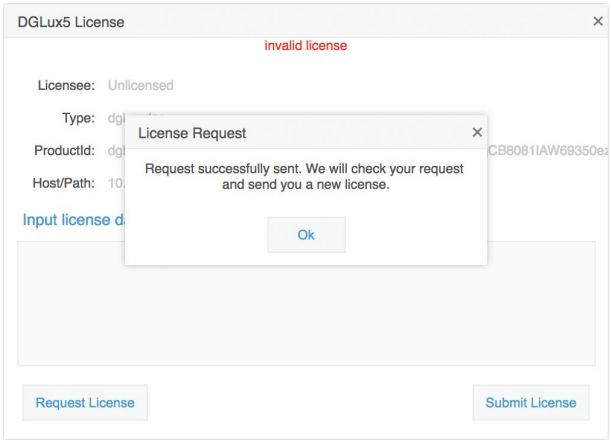
1. Using a web brower client, connect to the dglux port using **http://[Server IP address]: 9080** or **https://[Server IP address]:9443**.
2. Log in as administrator user and password defined at installation.



3. Click **Request License**. Complete the following form, click **180-day Trial License | 1500 Topics**, and then click **Submit Request**.



The following results display:



In most instances, licensing will be automatic if connected to the Internet. Otherwise a license will be returned via email to the user's email address.

4. Connect to the dglux server port using **http://[Server IP address]:9080** *or* **https://[Server IP address]:9443**. Once the license has been approved, the following pop-up menu will display. If you agree to the End User License Agreement, click **I agree**.

Installation of EFM Smart Licensing Tool

To install EFM Smart Licensing Tool run the following command:

```
sudo yum install -y efm-licensing
```

User `efm` is automatically created and owns the installation. No service is registered.

For use of the EFM Smart Licensing tool, refer to the EFM Smart Licensing tool user guide.

Installation of EFM ParStream Historian Database

To install ParStream Database run the following command:

```
sudo yum install -y parstream-database
```

User `parstream` is automatically created, owns the installation and is configured to run the service.

The following systemd service is automatically added:

```
parstream-database@.service
```

This service configuration needs to be adapted according to your ParStream configuration. E.g. if you parstream working directory is `/data/parstream` the following must be added to the service:

```
[Unit]
 AssertPathExists=/data/parstream
[Service]
 WorkingDirectory=/data/parstream
 LimitNOFILE=131072
```

For ParStream Database, the service needs to be enabled with the corresponding instance name according to your configuration. For example, if you run a single node instance with ParStream server name `parstream1` (configured in your parstream.ini file) the command is:

```
sudo systemctl enable parstream-database@parstream1
```

If you don`t start and stop the database using the provided service, please ensure your `LD_LIBRARY_PATH` points to `/opt/cisco/kinetic/parstream-database/lib`

For use of the EFM ParStream Database, refer to the EFM ParStream manual.

Optional: Installation of EFM ParStream Database Authentication

To install ParStream Database Authentication run the following command:

```
sudo yum install -y parstream-authentication
```

No service is registered.

To use ParStream with user authentication, please configure a password for your parstream user running the command:

```
sudo passwd parstream
```

Please note, with activated user authentication you need to invoke pnc providing the username:
`pnc -U parstream`

Installation of the C++ Broker

The C++ Broker is a multi-threaded high-performance broker with very low footprint in order to leverage the multi-core capability of different platforms. This allows allow for performance on the edge. The C++ broker scales and performs so well that we recommend to use it on all other levels, except when UI is required, in a multi-tier architecture.

The C++ Broker is a standalone broker with no additional links. If needed, their installation will need to be performed after.

Installing runtime dependencies:

The C++ requires the libatomic dependency to be installed by running the following command:

```
sudo yum install -y libatomic
```

If the Dart runtime has not yet been installed, which is a prerequisite for Dataflow and DQL dslinks, run the following command to install:

```
sudo yum install -y efm-dart-runtime
```

If the Java SDK runtime environment has not yet been installed, which is a prerequisite for Java based dslinks, run the following command to install:

```
sudo yum install java-1.8.0-openjdk-headless
```

To install C++ run the following command:

```
sudo yum install -y efm-cpp-broker
```

User `efm` is automatically created, owns the installation and is configured to run the service.

The following systemd service are automatically added:

```
efm-cpp-broker.service
```

Starting the C++ Broker:

```
sudo systemctl enable efm-cpp-broker
```

Using Newly Defined Services

The registered services can be administrated, configured and used with usual systemctl and service commands.

If a new added service is not known by the system yet, run the following command to reload the services:

```
systemctl daemon-reload
```

All services are disabled by default for security reasons. To configure the services to automatically start and continue to run after a reboot, use the following commands:

For EFM Applications:

```
sudo systemctl enable openresty

sudo systemctl enable efm-server

sudo systemctl enable efm-authentication-server

sudo systemctl enable efm-manager
```

For DGLux:

```
sudo systemctl enable efm-dglux
```

For the C++ Broker:

```
sudo systemctl enable efm-cpp-broker
```

For ParStream Database, the service needs to be enabled with the corresponding instance name according to your configuration. For example, if you run a single node instance with ParStream server name `parstream1` (configured in your parstream.ini file) the command is:

```
sudo systemctl enable parstream-database@parstream1
```

Some other examples for service usage

To verify the status of the service, use the commands chkconfig and systemctl. For example:

```
$ sudo chkconfig efm-manager
Note: Forwarding request to 'systemctl is-enabled efm-manager.service'.
enabled

$ sudo systemctl status efm-cpp-broker
● efm-cpp-broker.service - Cisco Kinetic EFM IOT C++ Broker
   Loaded: loaded (/usr/lib/systemd/system/efm-cpp-broker.service; enabled; vendor preset:
disabled)
   Active: active (running) since Thu 2019-01-24 14:32:41 EST; 4min 14s ago
 Main PID: 741 (broker)
```

```
   CGroup: /system.slice/efm-cpp-broker.service
           └─741 /opt/cisco/kinetic/cpp_broker/bin/broker

Jan 24 14:32:41 efm93 systemd[1]: Started Cisco Kinetic EFM IOT C++ Broker.
Jan 24 14:32:41 efm93 systemd[1]: Starting Cisco Kinetic EFM IOT C++ Broker...
```

To verify the installation of service files:

```
$ ls /usr/lib/systemd/system/ | grep -i efm
```

To view the running services for efm:

```
$ sudo systemctl | grep -i openresty

$ sudo systemctl | grep -i efm

$ sudo systemctl | grep -i parstream
```

Upgrading to EFM version 1.7.0 from an existing installation on CentOS7 and RHEL7

EFM Upgrade and Migration on CentOS7 and RHEL7 from EFM 1.5.x to EFM 1.7.0

In the following section the upgrade/migration from an existing EFM 1.5 installation to EFM 1.7.0 is described.

Prerequisite: Validate or create required users

Before you start the migration, please ensure all required system users are created with correct settings.

Username	Home Directory	Command to create user
efm	Required	`useradd -r -U -m efm`
dglux	Required	`useradd -r -U -m dglux`
parstream	Not required	`useradd -r -U -M parstream`

Upgrade EFM Manager

There is no migration support for EFM Asset Manager to EFM Manager.

Please install the new EFM Manager following the steps from section Prerequisites for installation of EFM Authentication Server and Installation of EFM Applications. Assets need to be added manually again.

Upgrade EFM Server with Dataflow Editor

To upgrade EFM Server and Dataflow Editor run the following commands:

```
cp -r $EFM_ROOT/efm_server /opt/cisco/kinetic/efm_server
sudo chown -R efm:efm /opt/cisco/kinetic/efm_server
sudo yum install -y efm-server
service efm-server start
```

Upgrade EFM Server with System Administrator

To upgrade EFM Server with Dataflow Editor and System Administrator run the following commands:

```
cp -r $EFM_ROOT/efm_server /opt/cisco/kinetic/efm_server
sudo chown -R efm:efm /opt/cisco/kinetic/efm_server
sudo yum install -y efm-server-admin
service efm-server start
```

Upgrade EFM Server with System Monitor

To upgrade EFM Server with Dataflow Editor and System Monitor run the following commands:

```
cp -r $EFM_ROOT/efm_server /opt/cisco/kinetic/efm_server
sudo chown -R efm:efm /opt/cisco/kinetic/efm_server
sudo yum install -y efm-server-monitor
service efm-server start
```

If your setup contains EFM Server with Dataflow Editor, System Administrator and System Monitor, run the following commands:

```
cp -r $EFM_ROOT/efm_server /opt/cisco/kinetic/efm_server
sudo chown -R efm:efm /opt/cisco/kinetic/efm_server
sudo yum install -y efm-server-admin
sudo yum install -y efm-server-monitor
service efm-server start
```

Migrate existing EFM Installations to use new EFM Authentication Server

To upgrade an existing EFM Server installation and add the new EFM Authentication Server, first follow the steps to upgrade your EFM Server installation (depending on the components in your setup follow the instructions according to Upgrade EFM Server with sections System Administrator and Upgrade EFM Server with System Monitor.

Stop the EFM Server Service to proceed with installation:

```
service efm-server stop
```

Install prerequisites for EFM Authentication Server:

```
sudo yum install yum-utils
CentOS-7 sudo yum-config-manager --add-repo
https://openresty.org/package/centos/openresty.repo
RHEL-7 sudo yum-config-manager --add-repo
https://openresty.org/package/rhel/openresty.repo
```

```
sudo yum install -y openresty
```

Install EFM Applications:

```
sudo yum install -y efm-applications
```

To complete the installation of EFM Authentication Server run the following script:

```
sudo /opt/cisco/kinetic/efm_manager/bin/configure_efm_applications.sh
```

and follow the instructions to register your admin credentials for all UI components (default user name is: efmAdmin).

With completion of this step, the EFM Server and Applications are automatically started (via the registered services).

Please note, that EFM Server now changed from usage of file based user management to proxy based user management. Users created for file based user management will still be visible in the configuration, but will not have access to the system. To re-enable access for these users, they must be added in EFM Authentication User Management.

If your setup contains links running externally of your broker, please ensure to update the upstream connections used for these links. These now need to connect to port 443.

Upgrade DGLux5

To upgrade DGLux5 Server run the following commands:

```
cp -r $EFM_ROOT/dglux_server /opt/cisco/kinetic/dglux_server
sudo chown -R dglux:dglux /opt/cisco/kinetic/dglux_server
sudo yum install -y efm-dglux
service efm-dglux start
```

See Installation of DGLux5 for more details on requesting a trial license if necessary.

Upgrade EFM Smart Licensing Tool

An upgrade or migration of EFM Smart License Tool is not required, a simple installation following the step from section Installation of EFM Smart Licensing Tool is sufficient.

Upgrade ParStream Historian Database

ParStream configurations should never be stored below the installation folder. To upgrade ParStream Historian Database install ParStream following the steps from Installation of EFM ParStream Historian

Database and configure the system script according to your ParStream configuration (see ParStream Manual, chapter “2.10 Installation – Systemd” for detailed information).

Upgrade C Broker to new C++ Broker

The C Broker is replaced by a new C++ Broker with EFM 1.7.0. Generally, the C++ Broker is compatible to the C Broker configurations. To upgrade an existing C Broker installation to the new C++ Broker run the following commands:

```
cp -r $EFM_ROOT/efm_cbroker /opt/cisco/kinetic/cpp_broker
sudo chown -R efm:efm /opt/cisco/kinetic/cpp_broker
sudo yum install -y efm-cpp-broker
service efm-cpp-broker start
```

The C++ Broker `data.json` format is not compatible to the C Broker `data.json`. The C++ Broker will migrate the `data.json` file automatically. Afterwards, it should not be used anymore with a C Broker. The C++ Broker will make a backup of the `data.json` file before merging it. It will be called `data.json.bak`.

Installing runtime dependencies:

The C++ requires the libatomic dependency to be installed by running the following command:

```
sudo yum install -y libatomic
```

If the Dart runtime has not yet been installed, which is a prerequisite for Dataflow and DQL dslinks, run the following command to install:

```
sudo yum install -y efm-dart-runtime
```

If the Java SDK runtime environment has not yet been installed, which is a prerequisite for Java based dslinks, run the following command to install:

```
sudo yum install java-1.8.0-openjdk-headless
```

EFM Upgrade and Migration on CentOS7 and RHEL7 from EFM 1.6.x to EFM 1.7.0

In the following section the upgrade/migration from an existing EFM 1.6.X installation to EFM 1.7.0 is described.

Install local EFM repository of RPMs

Copy EFM archive from corresponding platform subfolder of the EFM-1-7-0.zip file to the target system.

The archive name follows the format repo-<major-version>.<minor-version>.<patch-version>-<build-number>.zip, i.e. " repo-1.7.0-34.zip"

If the existing /repo directory exists from version 1.6.X, it can be renamed or erased. The following steps will create the new /repo directory for the upgrade.

```
$ mv repo repo_1.6.0
```

The following steps must be carried out to define the local EFM repository.:

Step 1	`$ unzip repo-1.7.0-34.zip`	Unpack the archive in the **"user"** home directory. This creates the directory " /repo" , which contains all EFM RPM packages

Step 2	`$ sudo cat << EOF > /etc/yum.repos.d/efm.repo`
	`[local]`
	`name=EFM Repository`
	`baseurl=file:///home/`**`user`**`/repo`
	`gpgcheck=0`
	`enabled=1`
	`EOF`
		Register the local EFM archive for yum. Therefore create a text file with the following content under the path " /etc/yum.repos.d/efm.repo" (the directory " /etc/yum.repos.d" already exists)
		
		If you used a different folder in Step #1, adapt the path for *baseurl* correspondingly.

Step 3	`$ sudo yum updateinfo`	Update YUM in order get the latest package versions. Now yum is prepared for the installation of the EFM RPM packages.

Update EFM application packages

Before updating, it is important to stop the efm services. For example:

```
systemctl stop openresty
systemctl stop efm-manager
systemctl stop efm-server
systemctl stop efm-authentication-server
```

ParStream example:

```
systemctl stop parstream-database@parstream1
```

Step 4	`$ sudo yum update efm-*` `$ sudo yum update parstream-*`	Update all EFM applications packages

After updating, restart the efm services. For example:

ParStream example:

```
systemctl start parstream-database@parstream1

systemctl start efm-authentication-server
systemctl start efm-server
systemctl start efm-manager
systemctl start openresty
```

EFM Component Installation Instructions for Ubuntu

On Ubuntu16.04 EFM only supports the new C++ Broker, no other components are supported on Ubuntu. It is necessary that the user migrate to RHEL7 or CentOS7 for EFM 1.7.0.

The EFM components for Ubuntu are delivered in the form of DEB packages. The DEB packages are installed, updated and uninstalled using the APT tool.

Installation of the C++ Broker

To install C++ run the following command:

```
sudo apt install ./efm-cpp-broker-1.1.2-Ubuntu16.04.deb
```

User `efm` is automatically created, owns the installation and is configured to run the service.

Installing runtime dependencies if needed:

If the Dart runtime has not yet been installed, which is a prerequisite for Dataflow and DQL dslinks. Download the latest DART SDK release and unzip into the /opt/cisco/kinetic. The C++ Broker Lifecycle Manager will automatically use the dart runtime.

```
wget https://storage.googleapis.com/dart-
archive/channels/stable/release/1.21.1/sdk/dartsdk-linux-x64-release.zip
```

```
sudo unzip dartsdk-linux-x64-release.zip /tmp -d /opt/cisco/kinetic
```

If the Java SDK runtime environment has not yet been installed, which is a prerequisite for Java based dslinks, run the following command to install:

```
sudo apt install openjdk-8-jre-headless
```

The following systemd service are automatically added:

```
efm-cpp-broker.service
```

Starting the C++ Broker:

```
service efm-cpp-broker start
```

Using Newly Defined Services

The registered services can be administrated, configured and used with usual systemctl and service commands.

If a new added service is not known by the system yet, run the following command to reload the services:

```
systemctl daemon-reload
```

All services are disabled by default for security reasons. To configure the services to automatically start on reboot use the following commands:

For the C++ Broker:

```
systemctl enable efm-cpp-broker
```

Troubleshooting

Linux Firewall issues

Redhat and CentOS initially is configured by default with the firewall service turned on and blocks all incoming connections. It is necessary to consult the Operating System Guide to turn off or allow only the known service ports for the EFM connections. The proper configuration needs to be defined by the host administrator.[3]

For the firewall to allow for incoming connections on the 443 and on Redhat/CentOS, the following commands can be executed:

```
$ sudo firewall-cmd --add-port=443/tcp --permanent
```

If using unencrypted connections to the broker on the default 8080 port, this will need to be added:

```
$ sudo firewall-cmd --add-port=8080/tcp --permanent
```

You must restart the firewall to implement the changes.

```
$ sudo firewall-cmd --reload
```

Note that if any incoming connections for DSLinks to the ParStream database, etc. exist, those specific ports should be configured to allow incoming connections.

Proxy Server challenges and the EFM Message Broker

In some environments, it might be necessary to define a proxy server to access the Internet due to security restrictions. The EFM message broker uses a localhost communication to connect to the DSLinks on the same host and usually any proxy server configuration inhibits some of this functionality from functioning properly.

We have observed in the System Administrator that some DSLinks connect to the message broker, while others do not if there is a proxy server configured. The same can be said for DGLux5.

In order to successfully connect to all the DSLinks it stopping the Message Broker be necessary, remove the proxy settings and start again the message broker. For example:

- Stop the message broker with `/opt/cisco/kinetic/efm_server/bin/daemon.sh stop` or `/opt/cisco/kinetic/dglux_server/bin/daemon.sh stop`
- Remove the proxy server settings in the environment or system configuration

[3] Unless properly configured and made permanent, on RedHat and CentOS, the firewall service will restart in the default configuration.

- Start the message broker with `/opt/cisco/kinetic/efm_server/bin/daemon.sh start` or `/opt/cisco/kinetic/dglux_server/bin/daemon.sh start`

An alternative to removing the use of the proxy server is to define an exclusion list that includes the localhost. In this manner at least the localhost will not be forwarded to the proxy server and communications between the Message Broker and the DSLinks that are on the local host form a connection.[4]

Dart Broker DSLinks take long to start when no Internet connection is available or it is unstable

When starting or restarting the DART broker when there is no Internet connectivity or the Internet connection is unstable for the host operating system, the DSLinks start with a long delay. To prevent this issue, the server.json parameter "isAlwaysOffline" should be set to true.

See the section *Configuring the EFM Dart Message Broker or DGLux5 server configuration via the server.json file* for more detail.

Using IPv6 with EFM Manager Landing page, DART Broker, System Administrator, System Monitor and User Management

IPv6 is supported on all of the EFM components, but for troubleshooting purposes there are some recommendations for use with the EFM Manger and the reverse proxy that serves the EFM Manager Landing page, DART Broker, System Administrator, System Monitor and User Management if installed.

- IPv6 works properly using literal addressing such as `https://[2003:1:2:3::1]`
- If hostnames are used, locally or dns, they must be also resolved properly by the EFM host. Otherwise an error " You are not allowed to access this resource. Please contact your system administrator."

[4] See http://xmodulo.com/how-to-configure-http-proxy-exceptions.html for examples on Linux.

Configuring the EFM Dart Message Broker or DGLux5 server configuration via the server.json file

The EFM Dart broker is configured in the server.json file. The system administrator can edit the text file. Modifications to this file should be performed when the broker is not running to avoid the content being overwritten by the message broker. The new configuration will take effect after startup.

The Dart Broker configuration file server.json are the same for EFM Dart Broker and the DGLux5.

An example server.json configuration file located in the $EFM_ROOT/efm_server or $EFM_ROOT/dglux_server folder and does not necessarily contain all parameters:

```
{
  "debug": false,
  "host": "0.0.0.0",
  "port": 8080,
  "httpsPort": 8443,
  "certName": "cert.pem",
  "certKeyName": "key.pem",
  "certPassword": "",
  "enableHSTS": false,
  "enableCSRFProtection": false,
  "strictFileUpload": {
    "enabled": false,
    "useClamAV": false,
    "extensions": [
      "dg5",
      "dgi",
      "crt",
      "key",
      "woff",
      "ttf",
      "gif",
      "svg",
      "png",
      "jpg",
      "xml",
      "json",
      "sql",
      "csv"
    ]
  },
  "disableFileSecurity": false,
  "isAlwaysOffline": false,
  "broadcast": false,
  "workers": 1,
  "updateInterval": 200,
  "static": {
    "/.well-known": "/opt/cisco/kinetic/efm_server/.well-known"
  },
  "linkConfig": {},
  "disabledLinks": [],
  "enableUptimeChecker": true,
  "uptimeCheckUrl": null,
  "upstream": {},
```

```
"strictTls": false,
"quarantine": false,
"allowAllLinks": true,
"defaultPermission": [
  [
    ":config",
    "config"
  ],
  [
    ":write",
    "write"
  ],
  [
    ":read",
    "read"
  ],
  [
    ":user",
    "read"
  ],
  [
    ":trustedLink",
    "config"
  ],
  [
    "default",
    "none"
  ]
],
"useRuntimeManager": false,
"useDartRuntimeManager": false,
"useJavaRuntimeManager": false,
"passwordHasherIterations": 10000,
"passwordHasherKeyLength": 32,
"loginRedirectPath": "/",
"guestLoginRedirectPath": "/assets/",
"authType": "proxy",
"twoFactorAuth": "none",
"runPortChecks": true,
"storageDriver": "simple",
"downstreamName": "downstream",
"loggers": [],
"proxies": {},
"hooks": {},
"distributionUrl": "NO",
"linkRepositoryUrl": "https://dsa.s3.amazonaws.com/links/links.json",
"serverVmFlags": [],
"userTimeout": 525600,
"allowBrowserCaching": false,
"serverLogLevel": "INFO",
"enableLogCompression": true,
"logRotationInterval": 0,
"enableIPv6": true,
"dartRuntimeManagerVmFlags": [],
"javaRuntimeManagerVmFlags": [],
"allowPasswordChanges": true,
"keepCustomAssets": true,
"linkManagerEnvironment": {},
"timeHttpRequests": false,
```



```
  "generatedCertificateSubject": "/C=US/ST=California/L=Oakland/O=DGLogik
Inc./OU=Customers/CN=*",
  "enableCertificateGeneration": true,
  "alternativeBrokerUrl": null,
  "httpPathClassification": {},
  "corsProxyRules": "",
  "enableGit": false,
  "enableSingleSignOnServer": false,
  "maxQueueSize": 256,
  "ssoProviderUrl": null,
  "formatDg5": false,
  "allowedCorsRegexString": null,
  "loginAuditFileName": "audit.log",
  "loginAudit": false,
  "blockOutsideGuests": false,
  "brokerName": "broker-",
  "runBrokerInMain": true
}
```

In the following table, the default values are listed that are assumed by the server, if the parameter is not present in the server.json.

Option	Description	Default Value	Comments
allowAllLinks	When the value is true, all incoming DSLink connections will be accepted to /downstream. When the value is false, an incoming DSLink without proper authentication will be rejected unless quarantine is enabled.	true	
allowBrowserCaching	When enabled, this value will add Cache-Control headers for 300 seconds on static files such as .dg5, images, etc.	false	
allowedCorsRegexString	If you wish to allow, but restrict, the access of external sites to interface with your EFM server, you can set a Regular Expression string here which much match for the external server requests to be completed.	null	
allowPasswordChanges	When true, this value will enable passwords to be updated via the `/change_password` URL (after the user has logged in). This is only work if supported by the current (authType)[#authtype]	true	
alternativeBrokerUrl	If you wish for all DSLink connections to be forwarded to a separate broker rather than the default broker, you would specify the URI of the alternative broker here. This was primarily used for legacy installations.	null	
authType	Determines the authentication provider to use.	file	
blockOutsideGuests	Enable this value if you wish to require a valid user login to view all projects.	false	

broadcast	When this value is true, the server's broker is broadcast to the local network for discovery by other machines. When this value is false, the broadcast service is not enabled.	true	
certKeyName	SSL private key file name. Leave blank to disable HTTPS		
certName	SSL certificate file name. Leave blank to disable HTTPS		
certPassword	SSL certificate password. Set to null to disable HTTPS		
corsProxyRules	The EFM server may also be used to proxy requests to external servers. To limit the locations which the proxy can access, a list of addresses, separated by new lines, may be added to the string.	" "	
dartRuntimeManagerVmFlags	When the useRuntimeManager or useDartRuntimeManager options are enabled and the platform supports the use of a runtime manager, then the flags provided here are passed to the Dart VM prior to starting the DSLink manager.	[]	
debug	Enable/Disable Debugging Mode	false	For production site, this should always be false, debug:true may result in memory leak and bugs.

defaultPermission	Default permission setting for the root node. When this value is null, permissions are disabled, and everything has the config permission.	[[":config", "config"], [":write", "write"], [":read", "read"], [":user", "read"], [":trustedLink", "config"], ["default", "none"]]	
disableFileSecurity	When this value is true, then any user can access any file. When this is false, file permissions are checked.	false	
distributionURL	This value is the url used to check for updates of the EFM server. This value can be managed in the `/sys/config` nodes (generally should not change from default).	NO	
downstreamName	This value is the name of the downstream connections node. Previously releases used a downstream name of `conns`. However it is recommended to leave this as the default value, as other Requester DSLinks may make assumptions of the correct path.	downstream	
enableCertificateGeneration	When this option is set to true, the server will attempt to generate self-signed SSL certificates prior to launching the server. This will set the appropriate certName, certKeyName. If these values are not empty, then certificate generation will be skipped.	true	
enableCSRFProtection	When this value is true, the HTTP server will add specific headers and cookies to help mitigate Cross-Site Request Forgery attacks.	false	

enableGit	This value will enable git version control over your project directory. When enabled, modifications to files in the project will be committed to a git repository at the same file path, and can be used in project management to see a history of changes and even revert changes.	false	
enableHSTS	When this value is true, the HTTP server will always redirect to the HTTPS server, and the HTTPS server will have HSTS enabled to route requests automatically to the HTTPS server.		
enableIPv6	Toggles support for IPv6 connections	true	
enableLogCompression	If this value is true, then when log files reach approximately 5MB in size, they will be rotated and compressed. Log files will be renamed to `<logfile>.<millisecondTimeStamp>.gz`	true	
enableSingleSignOnServer	In an environment where there are multiple instances of the EFM Server installed on the network, it is possible to allow all instances to refer to one primary server for authentication. When this option is enabled, this server will act as a primary server and allow other EFM Server instances to query this server for a user session on this server and if found, share it with the other instance. This requires the ssoProviderUrl be supplied to the other EFM Server instances.	false	
enableUptimeChecker	The server also comes with a checker which will periodically check to verify that the server is still up and running and responsive. Setting this value to false will disable the uptime checker.	true	
formatDg5	When this value is true, EFM client will save dg5 in a formatted and json with key sorted, makes it easy to track changes.	false	
generatedCertificateSubject	If the option enableCertificateGeneration is enabled, this is the subject used when generating the self-signed certificate.	/C=US/ST=California/L=Oakland/O=Acuity Brands Inc./OU=Customers/CN=*	
guestLoginRedirectPath	Determines the URI that a user is redirected to when login is complete.	/	
hooks	This value is designed to execute a specific command line program at various server states. Currently the only supported state is `ready` which executes when the server has finished loading. The Map contains keys of state (eg ready) and value of a list of command line programs to execute.	{}	

httpPathClassification	This value is a map of paths which may match a specific classification string. The key is the classification and the value is a list of paths which match that classification. If enabled and a requested path to the server matches a path in that classification, then that request will be treated as that type of classification request even if not matching the original hardcoded path. Currently the only supported classification is `session`.	{}	
httpsPort	HTTPS port to listen on. If this is less than or equal to 0, and/or certName or certPassword is not provided, then the server does not listen on any port for HTTPS. Ensure that if you install a custom certificate, you fill in the certName, certKeyName and certPassword fields.	8443	At least one of port or httpsPort must have a valid port number assigned.
isAlwaysOffline	Indicates that a server is expected to never have a full internet connection. This will prevent the server from trying to download the list of DSLinks available in the remote repository.	false	
javaRuntimeManagerVmFlags	When the useRuntimeManager or useJavaRuntimeManager options are enabled and the platform supports the use of a runtime manager, then the flags provided here are passed to the Java VM prior to starting the DSLink manager.	[]	
keepCustomAssets	When the value is true, custom assets in www/assets are kept upon updating EFM Server.	false	
linkConfig	Each DSLink may optionally specify its own configuration parameters to use. These configuration parameters can be see under the `/sys/links/<linkName>/configs` node. If you modify one of those parameters, the value is updated in the server configuration, as opposed to directly modifying the DSLink's configuration file. This value will vary depending on the DSLinks installed, their given names and the configuration parameters they may provide. It should be modified from the DSA node tree rather than by hand.		
linkManagerEnvironment	This value is a map of environment variables to set when the DSLink manager is started.	{}	
linkRepository	This value is the url used to check for updates for any of the DSA Links installed via repository. This value can be managed in the `/sys/config` nodes (generally should not change from default).		

loggers	The server contains a number of specialized loggers, particularly for debugging, which may be added here to retrieve verbose logging information. Some examples include " File Service" and " Execute" . These would normally be advised to be enabled at the request of support.		
loginAuditFileName	This value only applies when *loginAudit* is enabled. This will be the filename, within the `/logs` directory, in which the login audits are recorded.	audit.log	
loginAudit	When enabled, this option will log to the *loginAuditFileName* any user logins, it will record the DateTime, username, and the IP address from which the request originated. It will also log any time a user's IP address changes during an active session.	false	
loginRedirectPath	Determines the URI that a user is redirected to when login is complete.	/	
logRotationInterval	This value is the number of seconds to wait before rotating log files. When this option is enabled (anything greater than 0) enableLogCompression will not be used. After the specified period, a log file will be renamed to `<filename>.<number>` the higher the number the older the log file. Any files greater than 2 will be removed.	0	
maxQueueSize	This value is the maximum number of items stored in the queue to be sent, if the queue reaches a volume greater the behaviour will vary depending on the QOS settings (merged, dropped etc).	256	
passwordHasherKeyLength	When using file based authType, this value determines the number of bytes that the encoded password should store.	32 bytes	
passwordHasherIteration	When using file based authType, passwords are encrypted locally using PBKDF2. This value determines the number of iterations of the PBKDF2 algorithm used to encode the password.	10000	
port	HTTP Port to listen on. If this is less than or equal to 0, then the server does not listen on any port for HTTP.	8080	At least one of port or httpsPort must have a valid port number assigned.
proxies	This value is a Map of path (key) and URI (value) pairs. Requests to the path will be forwarded to the URI	{}	

quarantine	** This setting has no effect when allowAllLinks is true ** When the value is true, a new incoming DSLink without a token will be put in /sys/quarantine. A quarantined DSLink can only work as a responder. Use the /sys/quarantine/authorize to move a quarantined DSlink to /downstream.	false	
runPortChecks	When set to true, this option will verify that the configured ports for the server (HTTP and HTTPS) are valid and available for use prior to actually starting the server.	true	
serverLogLevel	Sets the log level verbosity. Levels are: `NONE`; `SEVERE`; `WARNING`; `INFO`; `FINE`; `FINEST`; `ALL`; `DEBUG`. Each level will report its level and all prior to it. (Example: INFO will log all INFO, WARNING and SEVERE messages).	INFO	
serverVmFlags	This value is a list of flags to add to the server when being started. They only apply to the EFM server and not any managed links.		
ssoProviderUrl	When this value is supplied, it must be the URI of another EFM server instance. This server will request an existing session from the supplied server and if found grant access via that session. If no session is found, the user will be prompted to log into that server and will be redirected to this instance once successfully authenticated.	null	

static	Configures a static directory mapping. This is used to serve files and directories on the server. Example:
	{
	" /static" : " /srv/http/static"
	}
		{" /.well-known" : " /path/to/dsa/dglux-server/.well-known" }	

storageDriver	This option is available for future expansion for how data is persisted at various QOS levels. Currently only simple is supported.	simple	

strictFileUpload	**strictFileUpload** is a configuration map that contains 3 fields. When this option is enabled, it will affect various file upload capabilities. Notably, it prevents guest users from being able to upload a file; It will limit uploads to only explicitly permitted file extensions; and possibly scan uploaded files for viruses.
	
	The configuration options are:
	enabled
	When set to true, strictFileUpload is enabled. If false, it will disable to strictFileUpload checks.
	useClamAV
	When set to true, the server will attempt to find Clam Antivirus on the system and if located, it will try to utilize this to scan any file uploads the server receives from a user. If this value is false, or the ClamAV was not found on the system, antivirus scans will be skipped, but other strictFileUpload conditions still apply if enabled.
	extensions
	An allow list of permissible file extensions (omitting the leading .) When strictFileUpload is enabled, the filename must end in one of these extensions or the upload will be rejected.
		{
		"enabled": false,
		
		"useClamAV": false,
		
		"extensions": [
		"dg5",
		"dgi",
		"crt",
		"key",
		"woff",
		"ttf",
		"gif",
		"svg",
		"png",
		"jpg",
		"xml",
		"json",
		"sql",
		"csv"
		]
		}	

timeHttpRequests	If enabled, this value will cause all HTTP requests to the EFM Server to be timed and the log will be updated with the request and elapsed duration.	false	
twoFactorAuth	Determines the two factor authentication provider to use. Supported Two-Factor Authentication Providers • none: Don't enable two factor authentication. duo: Duo Two-Factor Authentication	none	
updateInterval	Only affects the responder. When this setting specified, a responder must not send stream updates to server more often than the minimum interval in milliseconds, value subscriptions in the responder should be cached. If a value subscription update is already cached, it must update the cache with the new value to prevent useless updates or updating an incorrect value. This value only affects the time between two updates of the same stream. If the responder does not respect the interval, the requestor might close the connection due to flooding.	200	

upstream	A list of upstream brokers that are defined with a locally referenced upstream name, locally referenced broker name, the connection URL that always has the suffix /conn, token and group.
	For example, the name “upstreamName”, local name "ThisName”, the URL https://192.168.22.93:443/conn,enable, no token and permission group “:config”.
	
	`"upstreamName": {`
	`"name": "ThisName",`
	`"url": "https://192.168.22.93:443/conn",`
	`"enabled": true,`
	`"token": null,`
	`"group": ":config"`
	`}`		

uptimeCheckURL	The server has a built-in checker to verify it is still running, and restart it if it goes offline or drops connections. By default the checker will attempt to connect to localhost. However if the server is bound to a different interface in the host parameter, you will need to specify the correct URL for the server. It should end in `/ping`. This only applies when enableUptimeChecker is enabled. Example: `"https://169.254.100.100/ping"`		
useDartRuntimeManager	When the value is true, the Dart Runtime Manager is used for Dart DSLinks. The Dart runtime manager reduces resource consumption by merging Dart DSLinks into a single process.	false	
useJavaRuntimeManager	When the value is true, the Java Runtime Manager is used for Java DSLinks. The Java runtime manager reduces resource consumption by merging Java DSLinks into a single process.	false	
userTimeout	Number of minutes of user inactivity (nothing being loaded from the server) after which session times out. This is a general setting, cannot be set per user.	525600	
useRuntimeManager	This value enables both the useDartRuntimeManager and useJavaRuntimeManager on the server if applicable. **WARNING: Setting this value has no effect on a Windows based server**		

workers	Number of Server Workers. For low end devices, this should stay at 1. For large machines, this can be set up to a maximum of 128. It is recommended that you do not exceed the number of logical processors on your machine.	For single-core machines, this is 1, for other devices, this is 2.	

Configuring the EFM C++ Message Broker configuration files

The newly introduced EFM C++ message broker the C message broker option. The C++ message is meant as an option for users to instead the full EFM Server Dart message broker version by providing several benefits:

- Improved performance compared to DART Message Broker and C Broker
- Smaller memory footprint than DART Message Broker
- Feature compatibility between DART Message Broker and C++ Message Broker
- Configuration consistency across all installation platforms (Linux/Windows/IOx)

The EFM C++ message broker allows for configuration of three different files rather than a single server.json file for the Dart broker. The system administrator can edit the text files. Modifications to this file should be performed when the broker is not running to avoid the content being overwritten by the message broker. The new configuration will take effect after startup.

Configuration files are located in the $EFM_ROOT/cpp_broker folder and does not necessarily contain all parameters:

The system administrator can edit the text files broker.json, manager.json and upstream.json. Modifications to these files should be performed when the broker is not running to avoid the content being overwritten by the message broker. The new configuration will take effect after startup.

broker.json example and parameters.

```
{
    "http": {
        "enabled": false,
        "port": 8100,
        "protocol": "dualstack"
    },
    "https": {
        "enabled": true,
        "port": 8463,
        "protocol": "dualstack",
        "certName": "server.pem",
        "certKeyName": "key.pem",
        "cert_chain_file": "server.ca-bundle",
        "tmp_dh_file": "dhparams.pem",
        "cipher_list": "HIGH:!aNULL"
    },
    "allowAllLinks": true,
    "workers": 1,
    "logging": {
        "log_level": "info",
        "debug_level": "no"
    },
    "ssl": {
        "self_signed_tls_certificate_allowed": true,
        "certs_path": "ca",
        "ca_file": "ca/ca-bundle.crt",
        "cipher_list": "HIGH:!aNULL",
        "verify_peer": true
```

```
    },
    "redo_log": {
        "path": ".redo",
        "max_entries_per_file": 1024,
        "max_size_per_file_bytes": 0,
        "max_files_per_log": 0,
        "flush_after_write": true,
        "automatic_recovery": true,
        "write_encrypted_values": true,
        "min_available_disk_space_threshold_mb": 50
    },
    "qos": {
        "default_queue_length": 1024
    },
    "max_send_queue_length": 8,
    "serializer": {
        "serialization_frequency": 1000,
        "serialize_values": true
    }
}
```

Section	Option	Default	Description
qos	default_queue_length	1024	Length of internal value queue
	max_send_queue_length	8	Specifies the maximum length of the internal send queue. If this number of send messages has not been acknowledged yet, the sending will be paused until at least some of these messages have been acknowledged. The default is 8.
serializer	serialization_frequency	1000	The node serialization will be called intermittently with this frequency in ms.
serializer	serialize_values	true	Controls if node values shall also be serialized. If set to false no values will be serialized.
redo_log	path	.redo	Path to the storage directory
redo_log	max_entries_per_file	1024	The maximum entries of each redo log file. The log will be cycled when this is reached. This limitation does not apply if set to 0.
redo_log	max_size_per_file_bytes	0	The maximum size (in bytes) of each redo log file.
redo_log	max_files_per_log	0	The maximum number of files in each redo log folder. The latest log will be deleted if this is reached. This limitation does not apply if set to 0.
redo_log	flush_after_write	true	Controls if a flush is performed after each write operation.
redo_log	automatic_recovery	true	Controls if consistencies issues of the redo log are being resolved automatically on start-up.
redo_log	write_encrypted_values	true	Controls if the data written for values is being encrypted.
redo_log	min_available_disk_space_threshold_mb	50	The minimum available disk space threshold (in MB). If the available disk space drops below the threshold the oldest log will be deleted when the log is cycled. This limitation does not apply if set to 0.

ssl	self_signed_tls_certificate_allowed	true	Specifies if self signed certificates are allowed or not.
ssl	certs_path	ca	Specifies the location the certificate verification will look for certificates.
ssl	ca_file	ca/ca-bundle.crt	Specifies the location of the CA certificates files.
ssl	cipher_list	HIGH:!AES256-SHA:!DHE-RSA-AES256-SHA:!ECDHE-RSA-AES256-SHA:!CAMELLIA:!aNULL	Specifies the cipher list string. See https://www.openssl.org/docs/man1.0.2/apps/ciphers.htmlfor more information.
ssl	verify_peer	true	Specifies if the certificate of the peer should be verified.
logging	log_level	info	The log level to use. Can be one of fatal,error,warning,info,debug.
logging	debug_level	no	Sets the debug log level. Can be one of NO,L1,L2,L3,L4,L5.
	workers	Number of cores	Sets the number of worker threads to use for processing messages.
https	enabled	true	Specifies if https is enabled.
https	port	8463	Specifies the port to use.
https	protocol	dualstack	Specifies the protocol to use. Dualstack support ipv4 and ipv6. One of dualstack,ipv6,ipv4.
https	certName	server.pem	Specfies the name of the server certificate file. In contrast to the C Broker, the C++ Broker supports to specify the certificate with a complete path. Nevertheless, the C++ Broker will check the certs directory used by the C Broker for cerificates, if no path was specified.
https	certKeyName	key.pem	Specifies the name of the server key file. In contrast to the C Broker, the C++ Broker supports to specify the key with a complete path. Nevertheless, the C++ Broker will check the certs directory used by the C Broker for keys, if no path was specified.
https	cert_chain_file	server.ca-bundle	The ca file to use to validate certificates. Can be specified with a complete path.
https	tmp_dh_file	dhparams.pem	The dhparams file to use. Can be specified with a complete path.
https	cipher_list	HIGH:!AES256-SHA:!DHE-RSA-AES256-SHA:!ECDHE-RSA-AES256-	Specifies the cipher list string. See https://www.openssl.org/docs/man1.0.2/apps/ciphers.htmlfor more information.

		SHA:!CAMELLIA:!aNULL	
http	enabled	false	Specifies if https is enabled.
http	port	8100	Specifies the port to use.
http	protocol	dualstack	Specifies the protocol to use. Dualstack support ipv4 and ipv6. One of dualstack,ipv6,ipv4.
	defaultPermission	null	Specifies the default permissions for the connected links. Recommended setting: [[":config","config"], [":write","write"], [":read","read"], [":user","read"], [":trustedLink","config"], ["default","none"]]

manager.json example and parameters.

```
{
    "enabled_links": {
        "modbus": true,
        "System": true,
        "Serial": true
    }
}
```

Option	Default	Description

enabled_links		Will be managed automatically by the lifecycle manager.
		Lists the installed links. Each link can be enabled or disabled. Example:
		`"enabled_links": {`
		`"dataflow": true,`
		`"c-serial": false`
		`}`

link_repository_url	https://dsa.s3.amazonaws.com/links/links.json	Url from which to download the link repository. Has to be a https address schema.
log_dir_path	logs	Where to put the link and broker log files
ssl_verify_peer	true	If the link_repository_url ssl certificates shall be verified.
certs_path	/etc/ssl/certs	Specifies the location the certificate verification will look for certificates.

ca_file	/etc/ssl/certs/ca-bundle.crt	Specifies the location of the CA certificates files.

configs		Will be managed automatically by the lifecycle manager.
		Individual link configs overridden by user settings.
		Example:
		`"configs": {`
		`  "Responder": {`
		`  "log": "info"`
		`  }`
		`}`

broker	depends on the brokers http and https configuration	Will be managed automatically by the lifecycle manager. The broker url to which the links shall connect.

upstream.json example and parameters.

```
{"name": "efmFogNode", "brokerName": "efmIR829edge", "url": "https://192.168.14.101:443/conn",
"enabled": true}
```

The upstreams are normally managed by the C++ Broker, but if need a file can be put into the upstream folder in the broker folder.

Option	Description
brokerName	The name of the current broker that will be shown under the downstream node of the upstream broker.
enabled	If this upstream is enabled or not. One of true, or false.
group	The permission group that the current broker will give to the upstream broker.
name	The name of the upstream broker, must be same as the file name in the upstream folder. This name will be shown under the upstream folder of this broker. It will also be shown under the `/sys/upstream` node.
token	The token to be used by the connecting broker when it connects to upstream broker.
url	The connection url of the upstream broker.

Obtaining documentation and submitting a service request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Subscribe to the *What's New in Cisco Product Documentation* as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS Version 2.0.

