Cloudera Edge Management 1.2.0

Deploying an EFM Cluster

Date published: 2020-06-22
Date modified: 2020-06-22

https://docs.cloudera.com/

Legal Notice

Contents

Deploying Edge Flow Manager Cluster

The Edge Flow Manager (EFM) server component of CEM supports clustered deployments for horizontal scalability and high availability.

An EFM cluster deployment requires the following:

- Three or more EFM nodes

  Two-node clusters are not supported.
- A shared and external EFM database (MySQL or PostgreSQL)
- An EFM load balancer for Web UI users

  You can use any load balancer of your choice, provided it can proxy HTTP(s) traffic and can be configured as described in this document.
- MiNiFi agents that forward client requests to one of the EFM cluster nodes

In addition to communicating with the external, centralized, and shared EFM database, EFM cluster members also communicate with each other to establish ephemeral state, such as distributed locks and caches, within the cluster. Due to this, each EFM node must have network connectivity to every other EFM cluster member.

EFM Cluster Database Setup

Before you first run or launch the cluster, you must install and configure MySQL or PostgreSQL.

For more information, see *Installaing Databases*.

You can use an existing external database of a single and standalone EFM instance for the cluster deployment. None of the existing data will be lost.

However, you cannot use an H2 database of a standalone EFM instance and you must start over with an external database. When moving from H2 to MySQL or PostgreSQL, prior to stopping the standalone instance, export the existing flows from the EFM Flow Designer and import into the cluster to avoid data loss. For more information on exporting and importing flows, see the REST API Reference.

Related Information

Installing Databases

EFM Cluster Configuration

Every EFM cluster member must be configured to be part of the same cluster.

You can find the cluster configuration in the efm.properties file located in the conf directory of the EFM installation. The following is an example cluster configuration:

```
# Web Server Properties
#  address: the hostname or ip address of the interface to bind to; to bind
to all, use 0.0.0.0
efm.server.address=10.0.0.1
efm.server.port=10080
efm.server.servlet.contextPath=/efm
# Cluster Properties
#   address: the address (host:port) to bind to for the embedded Hazelcast
 instance that coordinates cluster state
#   memberAddress: the address (host:port) to advertise to other cluster m
embers, if different from the bindAddress
#   members: comma-separated list all cluster nodes; must be identical on
all nodes in the cluster, including order
```

```
#            format of node address is hostname or IP or hostname:port or
IP:port
#            port is optional (5701 the default port)
efm.cluster.enabled=true
efm.cluster.address=10.0.0.1:5701
#efm.cluster.memberAddress=
efm.cluster.members=10.0.0.1:5701,10.0.0.2:5701,10.0.0.3:5701
# Database Properties
efm.db.url=jdbc:mysql://database.example.come:3306/efm
efm.db.driverClass=com.mysql.cj.jdbc.Driver
efm.db.username=efm
efm.db.password=efmPassword
efm.db.maxConnections=50
efm.db.sqlDebug=false
```

Note: Except for the node-specific configurations, such as efm.server.address and efm.cluster.address, which can be a hostname or IP address, all other configurations must match for all nodes in the cluster. It is especially important that the nodes use the same database URL (efm.db.url), and the same list of cluster members (efm.cluster.members). If these values do not match on all cluster nodes, the cluster fails on startup.

EFM Cluster High Availability

In a clustered mode, EFM is highly available.

Even if individual nodes are stopped the cluster can continue operation. Nodes can stop or fail due to a variety of reasons, such as routine maintenance, loss of network connectivity, or underlying physical server host failure. By default, an EFM cluster can continue operating with up to two nodes down. If the failed nodes are restored, the cluster reforms. Note that any load that was served by a failed node is shifted to other cluster nodes. Hence capacity may fall if all nodes in the cluster were optimally utilized with regards to resources including network, CPU, memory, and disk.

Securing an EFM Cluster

This section describes how to secure intra-cluster communication.

By default, intra-cluster communication is through unsecure TCP. It is assumed that all EFM instances must be running in a Virtual Private Cloud and the cluster communication port must not be open to public networks.

Note: The default cluster communication port is 5701, and it is configurable as a part of the efm.cluster.address property value.

To secure intra-cluster communication, EFM supports an external, third-party utility for TCP TLS tunneling called Stunnel. The following section is an example configuration for enabling Stunnel communication.

Install Stunnel

Stunnel is available on most major Linux distributions, including those supported for EFM.

Stunnel version 5.x is required. EFM is tested with Stunnel 5.56 specifically.

To install Stunnel, run yum install stunnel or apt-get update && apt-get install stunnel.

Configure EFM to use Stunnel

Following is an example of configuring EFM to use Stunnel:

```
# Cluster TLS/SSL Tunnel Properties
#  enabled: enable secure communication within the cluster via a stunnel pr
oxy
```

```
#  command: the command or path to executable for stunnel, which must be in
stalled, e.g., /usr/bin/stunnel
#  logLevel: the level of stunnel debug output: emerg|alert|crit|err|warnin
g|notice|info|debug
#  logFile: (optional) if specified, the file to use for stunnel logs. if n
ot specified, output is to EFM App Log

#  caFile: The file containing Certificate Authority certificates. Must be P
EM format.

#  cert: The file containing this cluster node's public certificate. Must be
 PEM format.

#  key: The file containing this cluster node's private key. Must be PEM f
ormat. Can be encrypted or unencrypted

#  keyPassword: (optional) If the key file is encrypted with a password, the
 password to decrypt the key file.

#  proxyServerPort: the port that will receive the TLS traffic and redirect
to Hazelcast (default 10090)

#  proxyClientPortStart: starting with the given port, the ports used to p
roxy communication with other cluster members
#  over the secure TLS tunnel (default 10091). The number of ports used is
 one fewer than the number of cluster members.

#  For additional Stunnel configuration options, see https://www.stunnel.org
/static/stunnel.html
#  global options, service level options, or client-/server-specific server
 options can be specified as
#  key-value pairs with the appropriate prefix efm.cluster.stunnel.[global|
service|clientService|serverService].*

efm.cluster.stunnel.enabled=true
efm.cluster.stunnel.command=/usr/bin/stunnel
efm.cluster.stunnel.caFile=/path/to/keys/ca-cert.pem
efm.cluster.stunnel.cert=/path/to/keys/efm-node-cert.pem
efm.cluster.stunnel.key=/path/to/keys/efm-nod-key.pem
efm.cluster.stunnel.keyPassword=keyPemFilePassword

# The proxy server port that will receive the TLS traffic and redirect to
local hazelcast instance (default is 9000)
efm.cluster.stunnel.proxyServerPort=9000
# Starting with the given port,the ports are used to proxy the TLS traffic
to other cluster members (default is 9001)

# the number of ports used will be equal to cluster members -1
efm.cluster.stunnel.proxyClientPortStart=9001
```

Note: Unlike the EFM web server and REST API settings for TLS/SSL, which support JKS or PKS12 keystores, Stunnel only supports PEM files. You must extract keys, certificates, and CA certificates and convert to a PEM file format, using standard tools such as keytool and OpenSSL. Here is an example of obtaining a PEM key and certificate file from a JKS keystore:

```
# Convert JKS keystore to PKCS12 keystore
keytool -importkeystore \
    -srckeystore "/path/to/keystore.jks" -srcstoretype jks -srcstorepas
s <keystore_pass> -srckeypass <key_pass> \
    -destkeystore "/path/to/keystore.p12" -deststoretype pkcs12 -dest
storepass <keystore_pass>
# Extract PEM key from PKCS12 keystore
openssl pkcs12 -in "/path/to/keystore.p12"  -passin pass:<keystore_pass
> -out "key.pem" -passout pass:<keystore_pass>
# Extract PEM cert from PKCS12 keystore
openssl pkcs12 -in "ketstore.p12"  -passin pass:<keystore_pass> -out
 "cert.pem" -nokeys
```

Note: For certain versions of OpenSSL and Stunnel on certain Linux distributions, password protected PEM key files are not supported. If you have trouble starting Stunnel due to failure to load the key, remove the password protection from the keyfile (taking care to set filesystem permissions to restrict access to the plaintext key) and omit the efm.cluster.stunnel.keyPassword property from efm.properties.

.