TIBCO eFTL™ Administration

Software Release 3.3
October 2017

Two-Second Advantage®

Important Information

Contents

About this Product

TIBCO® is proud to announce the latest release of TIBCO eFTL ™ software.

This release is the latest in a long history of TIBCO products that leverage the power of Information Bus® technology to enable truly event-driven IT environments. To find out more about how TIBCO eFTL software and other TIBCO products are powered by TIB® technology, please visit us at www.tibco.com.

Product Editions

TIBCO eFTL and related products are now available in a community edition and an enterprise edition.

TIBCO eFTL - Community Edition is ideal for getting started with eFTL, for implementing application projects, including proof of concept projects, for testing, and for deploying applications in a production environment. Although the community license limits the number of production processes, you can easily upgrade to the enterprise edition as your use of TIBCO eFTL expands.

The community edition is available free of charge. It is a full installation of the TIBCO eFTL product, with the following limitations and exclusions:

- Users may run up to 1000 mobile clients in a production environment, per corporate entity.
- Users do not have access to TIBCO Support. However, you can use TIBCO Community as a resource (https://community.tibco.com).

TIBCO eFTL - Community Edition is compatible with both the enterprise and community editions of TIBCO FTL®.

TIBCO eFTL - Enterprise Edition is ideal for all application development projects, and for deploying and managing applications in an enterprise production environment. It includes all features presented in this documentation set, and access to TIBCO Support. Choose the enterprise edition for production deployments with more than 1000 mobile clients, and for enterprise monitoring using customizable status dashboards.

TIBCO eFTL - Enterprise Edition uses the enterprise edition of TIBCO FTL and includes a license for it.

TIBCO Documentation and Support Services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit https://docs.tibco.com.

Product-Specific Documentation

The following documents for this product can be found on the TIBCO Documentation site:

- *TIBCO eFTL Concepts*
- *TIBCO eFTL Administration*
- *TIBCO eFTL Development*
- *TIBCO eFTL Installation*
- *TIBCO eFTL API Reference* (HTML only)
- *TIBCO eFTL Release Notes*

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit http://www.tibco.com/services/support.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at https://support.tibco.com.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to https://support.tibco.com. If you do not have a user name, you can request one by clicking Register on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to https://community.tibco.com.

Server Administration

Administrators configure, run, and maintain server processes.

Administrators define eFTL clusters in the realm definition.

A cluster can run either as a singleton server process, or as a cluster of server processes for scalability and fault tolerance.

Client Authentication and Authorization

The TIBCO eFTL server can use an external authentication service to authenticate and authorize its eFTL clients.

The eFTL administrator configures the authorization groups that can publish and subscribe on each channel. That is, each channel can specify two of authorization groups: one for publishing and one for subscribing. When a user runs an eFTL application, that application can publish if the user name is in the channel's publish authorization group, and subscribe if the user name is in the channel's subscribe authorization group.

To enable authentication and authorization for a cluster of eFTL servers, see “Authorization” in Channel Details Panel.

To assign authorization groups to a channel, see Authorization.

One file configures authentication for both the realm server and the eFTL server. For an example, see the authentication configuration file in the samples directory.

JAAS Login Modules

TIBCO eFTL server supports JAAS login modules through the TIBCO FTL realm server.

See “JAAS Login Modules” in *TIBCO FTL Administration*.

Password File

If the realm server enables JAAS authentication and authorization, then you must configure credentials for TIBCO eFTL servers, which run as client processes of the realm server.

A TIBCO eFTL server process authenticates itself to the realm server using credentials that it reads from an ASCII password file, which you specify using the TIBCO eFTL server's --password-file command-line parameter.

The password file consists of two lines. The first line contains the user name. The second line contains the password string.

The user name must be in the JAAS authorization group `ftl`. For details, see *TIBCO FTL Administration*.

To hide the password from casual observers, you may first obfuscate the password using `tibrealmadmin --mangle`. For details, see *TIBCO FTL Administration*.

Option Names

The TIBCO eFTL server accepts parameter values in two ways: a short form and a long form.

Short-Form Command-line Option

A hyphen with one or two alphabetic characters. The short form is quick to type, but occasionally non-intuitive.

Long-Form Command-line Option

A double hyphen (--) introducing a descriptive name. The long form is longer to type, but its meaning is usually intuitive.

Running an eFTL Server

Complete these steps to start a TIBCO eFTL server.

Prerequisites

You have configured an eFTL cluster using the TIBCO FTL realm server GUI.

Procedure

1. Add TIBCO FTL libraries to the library path.

   If you invoke an eFTL server executable file, you must complete this step.

   If you invoke an eFTL server start script in the `bin` directory (see step 3), you may omit this step, as the scripts set the library path automatically.

Platform	Description
Linux	Ensure that `LD_LIBRARY_PATH` includes *`TIBCO_HOME`*`/ftl/`*`version`*`/lib`.
Windows	Ensure that `PATH` includes *`TIBCO_HOME`*`\ftl\`*`version`*`\bin`.

2. Optional. Add TIBCO Enterprise Message Service libraries to the library path.

   If you configure EMS channels in the TIBCO eFTL server, you must complete this step.

Platform	Description
Linux	Ensure that `LD_LIBRARY_PATH` includes both *`TIBCO_HOME`*`/ems/`*`version`*`/lib` and *`TIBCO_HOME`*`/ems/`*`version`*`/lib/64`.
Windows	Ensure that `PATH` includes *`TIBCO_HOME`*`\ems\`*`version`*`\bin`.

3. Start the server using either a script or an executable file.

Platform	Description

Linux	Script:
	`TIBCO_HOME/eftl/version/bin/tibeftlserver.sh command_line_args`
	Executable File:
	`TIBCO_HOME/eftl/version/bin/tibeftlserver command_line_args`

Windows	Script:
	`TIBCO_HOME\eftl\version\bin\tibeftlserver.bat command_line_args`
	Executable File:
	`TIBCO_HOME\eftl\version\bin\tibeftlserver.exe command_line_args`

Server Command Line Reference

Administrators use `tibeftlserver`, the TIBCO eFTL server command-line executable program, to start a server process.

Most command-line parameters and options have both a short and a long form. The command-line parser accepts either form.

Help

Parameter	Arguments	Description
`-h` **`--help`**		Display a help message describing the command-line parameters and options.

Realm Server

Parameter	Arguments	Description
`--realmserver` **`-rs`**	*URL*	Optional. URL of the realm server. The TIBCO eFTL server contacts the realm server at this location to receive its realm definition. Supply the location of the primary realm server or a satellite server. When absent, the default value is `http://localhost:8080`.
`--secondary-realmserver` **`-s`**	*URL*	Optional. URL of the backup realm server. If the regular realm server is unavailable, the TIBCO eFTL server contacts the backup realm server at this location to receive its realm definition. Supply the location of the backup realm server.
`--name` **`-n`**	*cluster_name*	Optional. Cluster name. A server process belongs to exactly one cluster. You may supply the name of the cluster to which this process belongs. When absent, the default value is `Cluster`.

Clients

Parameter	Arguments	Description

`--listen`
 	
`-l`
	URI	Optional.
		
		The TIBCO eFTL server listens for client connections at this URI. Supply a URI in either of two forms:
		
		`ws://`*`host`*`[:`*`port`*`]`
		`wss://`*`host`*`[:`*`port`*`]`
		
		When this parameter is absent, the server listens on `ws://localhost:9191`.
		
		If you omit the *port*, the server listens for non-TLS (ws:) connections on port 9191, or TLS (wss:) connections on port 9291.

Authentication and Authorization Service

Parameter	Arguments	Description
`--auth.url`	*URL*	Optional. The eFTL server contacts the authentication service at this URL. When present, authentication is enabled, so that the eFTL server requires and verifies user name and password credentials from clients. When absent, authentication is disabled, so that the eFTL server neither requires nor verifies credentials.
`--auth.user`	*user_name*	Optional. When the authentication service URL uses the `https://` protocol, the eFTL server identifies itself to the authentication service using this user name credential.
`--auth.password`	*password*	Optional. When the authentication service URL uses the `https://` protocol, the eFTL server identifies itself to the authentication service using this password credential.
`--auth.trust`	*path*	Optional. When the authentication service URL uses the `https://` protocol, use this parameter to specify the location of the authentication service's public certificate file (in PEM format). The eFTL server uses the certificate to verify the identify of the authentication service.

Security: Realm Server

Parameter	Arguments	Description
--password-file **-pf**	*path*	Required if the realm server requires password authentication from its clients. The TIBCO eFTL server uses credentials in this password file to authenticate itself to the realm server. See also, Password File.
--trust-file	*path*	Optional. (Required for TLS communication with a secure realm server.) When present, the eFTL server reads a trust file from *path*, and uses that trust data in communications with the secure realm server. See "Running a Secure Realm Server" in *TIBCO FTL Administration*.
--trust-everyone		Optional. The eFTL server trusts any realm server without verifying trust in the server's certificate. ⚠ Do not use this parameter except for convenience in development and testing. It is *not* secure.

Security: Clients

Parameter	Arguments	Description
--server-cert	*path*	Required if the --listen argument specifies TLS, that is, it uses the wss protocol scheme. The TIBCO eFTL server reads a public TLS certificate from this file. It uses the certificate to authenticate itself to clients. The certificate file must be in PEM format. See also, Clients Trust the Server.
--private-key	*path*	Required if the --listen argument specifies TLS, that is, it uses the wss protocol scheme. The TIBCO eFTL server reads an encrypted TLS private key from this file. It uses the key to authenticate itself to clients, and to encrypt TLS communication with clients. The key file must be in PEM format. See also, Clients Trust the Server.

Parameter	Arguments	Description
`--private-key-password`	*password*	Required if you specify `--private.key`. The TIBCO eFTL server decrypts the private key using this password.

Security: EMS Server

Parameter	Arguments	Description
`--ssl-params`	*path*	Required for secure connections to an EMS server. The eFTL server reads parameters from this file, and uses them when connecting to a secure EMS server. For details, see SSL Parameters for EMS Connections Reference.

Messages

The TIBCO eFTL server can automatically append user and client identification information to the messages that clients publish. See "Client Information Fields" in *TIBCO eFTL Development*.

Parameter	Arguments	Description
`--publish-client-id`		Optional. When present, the server appends the `_client_id` field to every message that any eFTL client publishes. See "Client ID Field" in *TIBCO eFTL Concepts*.
`--publish-user`		Optional. When present, the server appends the `_user` field to every message that any eFTL client publishes. See "User Field" in *TIBCO eFTL Concepts*.

Logging

Parameter	Arguments	Description
`--trace` **`-t`**	*level*	Optional. When present, the TIBCO eFTL server logs at this level of detail. Supply one of these strings: • `off` • `severe` • `warn` • `info` • `verbose` • `debug` The output from `debug` and `verbose` can result in very large log files. This level of detail is generally not useful unless TIBCO staff specifically requests it. When this option is absent, the default value is `info`.
`--verbose` **`-v`**		Optional. When present, the TIBCO eFTL server produces verbose log output related to clients. This level of tracing is appropriate only during client program development and testing.
`--logfile`	*logfile_prefix*	Optional. When present, the TIBCO eFTL server logs to a rolling set of log files instead of the console. The *logfile_prefix* argument may denote a path. If so, then all of the directories in the path must already exist. For more information about rotating log files, see "Log Output Targets" in TIBCO FTL Development. When absent, the TIBCO eFTL server sends log output to the console, and ignores the parameters --max.log.size and --max.logs.
`--max-log-size`	*size*	Optional. Limits the maximum size (in bytes) of log files. The value must be greater than 100 kilobytes, that is, 102400 bytes. The default value is 2 megabytes, that is, 2×1024×1024.

Parameter	Arguments	Description
`--max-logs`	*logs*	Optional. Limits the maximum number of rolling log files. The default is 50.

SSL Parameters for EMS Connections

If you configure any EMS channels in the eFTL server, you must also arrange a configuration file with parameters from the following table. The eFTL server uses these parameter values when connecting to a EMS server.

For information about these parameters, see "Configuring SSL in EMS Clients" in *TIBCO Enterprise Message Service User's Guide*.

For syntax, see the example configuration file in the `samples` directory.

Supply SSL parameters in a file, and specify the file name as the value of the command line parameter --ssl-params when you start the eFTL server.

All the EMS channels in a cluster must use the same set of SSL parameter values for connecting to their respective EMS servers.

Parameter	Note
`ssl_issuer_cert`=*filename*	
`ssl_trusted_cert`=*filename*	
`ssl_auth_only`=`true	false`	
`ssl_ciphers`=*cipher_suites*	
`ssl_expected_hostname`=*hostname*	
`ssl_identity`=*filename*	
`ssl_private_key`=*filename*	
`ssl_password`=*private_key_password*	You may obfuscate the password using `tibrealmadmin --mangle`. For details, see *TIBCO FTL Administration*.
`ssl_rand_data`=*data*	
`ssl_rand_file`=*filename*	
`ssl_rand_egd`=*filename*	
`ssl_verify_host`=`true	false`	
`ssl_verify_hostname`=`true	false`	

Cluster and Server Configuration

Administrators can configure a cluster of TIBCO eFTL servers using the realm server's interfaces: its GUI and its web API. When a TIBCO eFTL server connects to the realm server, the realm server supplies its configuration.

The following topics present tasks for configuring eFTL definitions, and the details of the realm server GUI.

For web API details, see eFTL Objects in the FTL Realm Server Web API .

Defining a Cluster

A cluster definition contains the configuration information for the eFTL server processes.

Procedure

1. Create a new cluster definition.
   a) In the eFTL clusters grid, add a new cluster definition row.
   b) Enter the name of the new cluster Cluster Name column.
   c) Optional. In the cluster details panel, add a description of the cluster.
2. Add channels and configure their settings.

   You must add at least one channel.

   For details, see Channel Details Panel.
3. Optional. Connect FTL channels to FTL applications.

   See Connecting FTL Channels to Applications.
4. Optional. Configure authentication.
   a) Select the check box in the **Authorization** column of eFTL Clusters grid.
   b) Configure the authentication service.

      - If the realm server uses a JAAS authentication service, configure the eFTL authentication service with a `tibeftlserver` entry in the JAAS realm configuration file.
      - If the realm server uses a flat-file authentication service, configure the eFTL user authentication data in a file, and supply the URL location of that file to the realm server command line parameter **`--auth.eftl.url`**.

      See the sample authentication file. See the “Realm Server Authentication" and its sub-topics in *TIBCO FTL Administration*.
5. For each channel, configure the publish and subscribe authorization groups.

Connecting FTL Channels to Applications

The initial configuration of a new FTL channel automatically includes a dynamic TCP transport for the channel's application-facing endpoint. You can extend that transport on the FTL side in either of these two ways.

(For background information, see Cluster-Facing and Application-Facing Endpoints.)

Procedure

- Optional. Connect FTL Applications.

  You can connect an FTL channel to applications on the FTL side in either of two ways:

Option	Description
Preferred	Share the dynamic TCP transport that implements the channel's application-facing endpoint, so that it also implements the endpoint of the FTL application: 1. In the eFTL cluster grid, locate the channel's application-facing endpoint in the Transport Name column, and note its name. Each automatically generated name concatenates the prefix `eftl_channel_app_facing_transport_`, the cluster name, and the channel name. 2. In the applications grid, locate the FTL application definition or application instance definition. Then locate the endpoint and select the transport you noted from the drop-down menu.
Alternative	Add a transport from the FTL application to the channel's application-facing endpoint: 1. In the applications grid, locate the FTL application definition or application instance definition. Then locate the endpoint and note its transport name. 2. In the eFTL cluster grid, locate the channel, and select **Add Transport** from the drop-down menu. 3. In the Transport Name column of the new row, select the transport you noted from the drop-down menu.

Avoid crosstalk by ensuring that the FTL application does not share transports with more than one channel, nor with FTL applications on other channels. For more information, see Cluster-Facing and Application-Facing Endpoints.

eFTL Clusters Grid

The eFTL Clusters grid presents eFTL cluster definitions in the realm. In edit mode you can create new cluster definitions and modify existing clusters.

Levels

- Cluster
- Channel
- Transport

Cluster Level

Column	Description
eFTL Cluster	Required. Name of the eFTL cluster. Cluster names must be unique within the realm. All names are limited to maximum length of 256 characters.

Column	Description
Maximum Connections	This maximum limits the number of clients that each eFTL server in the cluster can support simultaneously. Zero is a special value, which does not limit the number of clients.
Last Modified	This timestamp indicates the date and time of the most recent change to this eFTL cluster definition.
Authorization	When enabled, eFTL client applications must authenticate to the eFTL server with user name and password credentials. The eFTL server must access an authentication service that authenticates and authorizes eFTL client applications. When disabled, the eFTL server does not require authentication and authorization security for its clients. (However, if the realm server requires authentication security for its clients, the eFTL server, as a realm server client, must nonetheless comply.)

Channels

You must configure *at least one* channel in each eFTL cluster.

Parameter	Description
Channel Name	Required for each channel. Name of the channel. Channel names must be unique within each cluster. All names are limited to a maximum length of 256 characters.
Messaging	Required. A channel can use either FTL or EMS messaging infrastructure. Select one of these options. This choice of messaging infrastructure for the channel determines the set of parameters that you can configure in the following columns.
FTL Format	Optional for FTL messaging infrastructure. When present, the channel converts messages from eFTL clients into this *exchange format* as it forwards them to FTL clients. Select a format name from the drop-down menu of formats defined in the realm. When absent, the channel converts messages from eFTL clients into self-describing dynamic-format messages as it forwards them to FTL clients.

Parameter	Description
FTL Store	Optional for FTL messaging infrastructure. You may associate a persistence store with an FTL channel. To prevent crosstalk, use unique stores: that is, ensure that no two channels use the same store.
FTL Durable Template	Optional for FTL messaging infrastructure. When eFTL clients create dynamic durable subscriptions on the channel, they use this template. You can associate at most one dynamic durable template with each channel. The menu offers the templates defined in the persistence store associated with the channel.
EMS Server	Required for EMS messaging infrastructure. The eFTL server connects to the EMS server at this URL. For fault tolerance supply a comma-separated list of URLs: first the primary EMS server, then the backup server.
EMS Delivery Mode	Required for EMS messaging infrastructure. Delivery mode instructs the EMS server concerning messages forwarded by the eFTL server. Select persistent or non-persistent from the drop-down menu.
EMS Topic Prefix	Optional for EMS messaging infrastructure. Channels isolate message streams. However, the EMS server merges message streams that share a topic name. A topic prefix can prevent crosstalk among channels through the EMS server. To prevent crosstalk, supply a distinct prefix string for each EMS channel. If present, the channel prepends this prefix string to EMS topic names at each publish and subscribe operation. Topic prefix strings are transparent to eFTL client applications. EMS administrators must explicitly allow these prefix strings in topic names. If absent, separate channels could carry crosstalk.
Push Notifications	Reports whether you have configured push notifications on the channel. For more information, see Configuring Push Notifications.

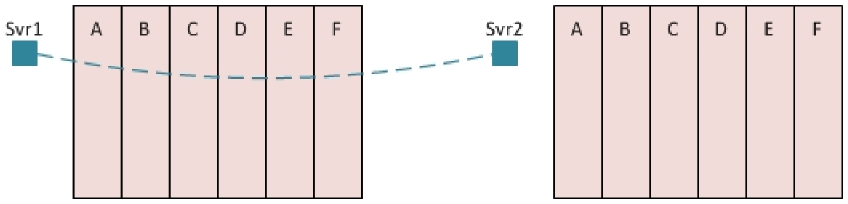
Transport Level

Parameter	Description
Transport Name	Applies only to FTL channels. This FTL transport implements the channel's application-facing endpoint. When you create an FTL channel, its initial configuration includes an automatically generated transport. Before modifying the initial configuration, read the background information and guidelines in Cluster-Facing and Application-Facing Endpoints. To add more transports to a channel's application facing-endpoint, select **Add Transport** from the drop-down menu at the channel level. To apply a specific transport definition to a channel's application-facing endpoint, select the name of that transport definition from the drop-down menu in this column.

Cluster-Facing and Application-Facing Endpoints

Each server in the cluster has a single *cluster-facing endpoint*, which it uses to exchange protocols with other servers in the cluster. In addition, each channel has a single *application-facing endpoint*, which the channel uses to exchange message data across the servers in the cluster (but within the channel), and also with eFTL clients of the channel.

In the diagram the blue square under each server's name represents its cluster-facing endpoint.



When defining a cluster, the configuration must include a separate FTL transport bus for the each of these endpoints. When you create a cluster or a channel, the realm server automatically generates a dynamic TCP transport (DTCP) for each of these endpoints as part of the initial configuration.

The initial configuration is often sufficient. It is good practice to use these automatically generated transports without modification, except for the following needs:

- To attach additional transport connectors to application-facing endpoints
- To define application instances to meet specific constraints of your network or hardware

If you choose to define your own transports for these endpoints, use these guidelines:

- The bus *must* be a full mesh topology: that is, a complete graph.
- The best practice is to use a dynamic TCP transport, such as those automatically generated in the initial configuration.

  Alternatively, you could use any transport protocol, but for clusters of two or more server processes it is convenient to use a symmetric transport definition: either a dynamic TCP transport or a symmetric multicast transport.

  (To use a fragmented transport, such as RDMA, you must configure separate application instances for each eFTL server process, and specify listen and connect roles for each server in the cluster.)

- The cluster-facing endpoint requires its own *dedicated* transport bus. You must *not* use that same transport to implement any FTL application endpoints: otherwise, you risk duplicate message delivery. You must *not* use that same transport to implement any application-facing endpoints: otherwise, you risk crosstalk among channels.

  Similarly, each application-facing endpoint requires its own *dedicated* transport bus. You must *not* use the same transport to implement more than one application-facing endpoint, nor any cluster-facing endpoint: otherwise you risk crosstalk among channels.

Use Unique Transports

If a transport is bound to either the cluster-facing endpoint, or any application-facing endpoint of a cluster, then you may *not* bind the same transport to any other endpoint in any eFTL cluster or channel.

⚠ Reusing a transport can result in unwanted crosstalk between channels, or even between clusters.

Nonetheless, you may extend a transport that is bound to an application-facing endpoint, by binding it to an endpoint of an FTL application. This type of extension is the preferred method for connecting an FTL application to an FTL channel.

Enabling Persistence

You can enable persistence for individual channels.

EMS Channel

EMS channels do not require special configuration to enable persistence. Instead, the EMS administrator must enable appropriate durables. To determine requirements, consult the application developer, and together complete the forms:

- TIBCO eFTL Application Coordination Form
- TIBCO eFTL EMS Channel Coordination Form

FTL Channel

To enable persistence for an FTL channel, complete the steps that follow.

Procedure

1. Define requirements.
   Consult the application developer and the FTL administrator, and together complete the forms:
   - TIBCO eFTL Application Coordination Form
   - TIBCO FTL Durable Coordination Form
2. Ensure that an FTL persistence store is configured.
3. Ensure that an FTL persistence server cluster is running.
4. Configure a persistence store for the channel, and select it in the eFTL clusters grid.
5. Configure the store's **`Publisher Mode`** to Store -> Send -> No Confirm.
   For best throughput performance, use this publisher mode to prevent the eFTL server from blocking when it publishes a message to the store.
6. Optional. Configure the channel's dynamic durable template.
   Set the **`Acknowledgment Mode`** of the template to Asynchronous.

Asynchronous acknowledgments prevent the eFTL server from blocking when it acknowledges inbound messages from the store. (This setting does not apply to last-value durables and templates.)

7. Optional. Configure a mapping from durable names to static durables for the channel.

   Subscriber name mapping is an advanced topic.

   The static durable must already be defined in the store.

   For each of those static durables, set the **`Acknowledgment Mode`** to Asynchronous. (For details, see the preceding note.)

eFTL Cluster Details Panel

The eFTL cluster details panel presents the details of a TIBCO eFTL cluster. In edit mode you can modify the definition.

All eFTL servers in a cluster receive the same definition from the realm server.

Optional. You can set the cluster description string in this panel.

Channel Details Panel

The Channel details panel presents the details of an individual channel definition. In edit mode, you can modify the definition.

Push Settings

See Configuring Push Notifications.

Parameter	Description
APNS Certificate	Enter the PEM-encoded private certificate data.
APNS Certificate Password	Enter the password for the private certificate.
Use APNS Sandbox	When enabled, use Apple's development sandbox network. When disabled, use Apple's production network.

Authorization

When security is enabled for the cluster, the TIBCO eFTL server authenticates and authorizes clients. Clients can use channel functionality based on their authorization groups. See also Client Authentication and Authorization.

These parameters are available if you enable security in the cluster settings; see eFTL Clusters Grid.

Parameter	Description
Publish Group	Only eFTL clients in this authorization group can publish messages on this channel.
Subscribe Group	Only eFTL clients in this authorization group can subscribe on this channel.

Parameters Specific to FTL Messaging Infrastructure

Parameter	Description
Dynamic Durable Template	Optional. When eFTL clients create dynamic durable subscriptions on the channel, they use this template. You can associate at most one dynamic durable template with each channel. The menu offers the templates defined in the persistence store associated with the channel.
Subscriber Name Mapping for Static Durables	Optional. Subscriber name mapping is an advanced topic. If you configure a persistence store for a channel, then you can configure a mapping from durable names (in subscribe calls) to static durable names (defined in the store). For the semantics of durable subscriptions, see "Persistence" in *TIBCO eFTL Development*.

Parameters Specific to EMS Messaging Infrastructure

Parameter	Description
User Name Password	Required for EMS infrastructure. The eFTL server authenticates itself to the EMS server using this user name and password.

Heartbeats and Timeouts

These parameters have factory default values, which you can override.

Parameter	Description
Heartbeat : eFTL Server to Realm Server	The TIBCO eFTL server sends heartbeats at this interval (in seconds) to clients on the channel. Clients respond to heartbeats, indicating that they are still connected. Zero is a special value, which prevents the server from sending heartbeats. If you set this parameter to zero, you must also set Client Timeout to zero.
Client Timeout	The TIBCO eFTL server disconnects a client after this interval (in seconds) since the last communication from the client, including heartbeat responses and message publishing calls. Zero is a special value, instructing the server to not disconnect clients.

Parameter	Description
Client Reconnect Timeout	When a client disconnects from the eFTL server, the server buffers outbound messages so that the client can receive them when it reconnects. The server buffers message for this interval. After this interval elapses, the server deletes outbound messages it was buffering for the clientL

Limits

These parameters have factory default values, which you can override.

Parameter	Description
Maximum Pending Acknowledgments	When the backlog of unacknowledged messages outbound to an eFTL client on the channel exceeds this limit, the server stops transferring messages to the client. When the server receives acknowledgments, it resumes sending messages to the client.
Maximum Message Size	You can limit the size (in bytes) of inbound messages. Publish calls in eFTL clients fail when a message exceeds this limit.
Maximum Queue Size	You can limit the growth of eFTL client message queues to conserve memory in the eFTL server. For background information and complete details, see Maximum Queue Size.

Maximum Queue Size

The eFTL server maintains a message queue for each eFTL client subscription. These queues hold messages for transfer to clients. If many subscriptions are present, and clients frequently disconnect, then queue growth can strain memory resources in the eFTL server. **`Maximum Queue Size`** limits queue growth to conserve memory.

When distributing a message to a queue would overflow this maximum, two behaviors are possible:

- If the channel is *not* configured for persistence, the queue discards the oldest messages to make room for new messages.
- If the channel *is* configured for persistence, the queue discards the new message. The persistence store redelivers discarded messages to the queue at a later time.

The default maximum is 100 messages. You can decrease this value to reduce memory requirements even further.

Valid Realm Modifications Reference

Some configuration changes that affect the eFTL server require the server to restart. The following table describes the ways in which you can modify eFTL configuration within the realm definition, and the deployment consequences of each modification.

eFTL Modifications

Modification	Details
Add or delete a cluster	You can add or delete an eFTL cluster.
Rename a cluster	If you rename a cluster, then all eFTL servers in the cluster require restart using the new cluster name.
Modify cluster settings	You can modify any settings of an eFTL cluster.
Modify cluster endpoint	If you modify the cluster endpoint, then all eFTL servers in the cluster require restart.
Add, delete, or rename a channel	You can add, delete, or rename a channel.
Modify channel settings	If you modify any settings of any channel, then all eFTL servers in the cluster require restart. However, if you *add* a new channel and configure its settings for the first time, the eFTL server does *not* require restart.
Modify channel endpoint	If you modify a channel endpoint, then all eFTL servers in the cluster require restart. However, if you *add* a new channel and configure its endpoint for the first time, the eFTL server does *not* require restart.
Modify Authorization Groups	If you modify the authorization groups of a channel, then all eFTL servers in the cluster require restart. However, if you *add* a new channel and configure its authorization groups for the first time, the eFTL server does *not* require restart.
Modify channel advanced parameters	If you modify any parameters in the **Advanced** panel of a channel, then all eFTL servers in the cluster require restart. However, if you *add* a new channel and configure its advanced parameters for the first time, the eFTL server does *not* require restart.

Server Monitoring

Administrators can monitor and manage a cluster of TIBCO eFTL servers using the realm server's monitoring interfaces: its GUI and its web API.

The following topics present the details of the realm server GUI that apply to monitoring and managing eFTL clients.

For web API details, see eFTL Objects in the FTL Realm Server Web API .

eFTL Clusters Status Table and Servers List

The eFTL clusters status table presents the status eFTL clusters. Cluster rows expand to present the status of individual servers.

Clicking a cluster row expands a servers list, detailing the eFTL servers in the cluster.

Cluster Table Columns

Field	Description
Status	The health of the cluster based on the aggregated status values of its servers. See the status values in the table Servers List Columns.
Name	Name of the eFTL cluster.
Servers	Number of servers in the cluster, and their combined state.

The **Attention** tally in the GUI top bar does not include persistence servers or eFTL servers unless *at least one* server from the cluster is present. Nonetheless, even in a situation where a cluster is configured but none of its servers are present, the status table indicates that the cluster and its absent servers are offline.

Servers List Columns

Field	Description
Status	**Running** The server process is running, and its local realm definition is up-to-date: that is, it is the latest realm deployment. **Offline** The eFTL server process is not running, and the realm server has no data about it. **Needs Restart** The server process is running, but it cannot accept a proposed realm deployment. To accept the realm deployment, you must restart the process instance. **Timed Out** The realm server has lost the eFTL server process' heartbeat signal. Either the TIBCO eFTL server process has stopped, or a network segmentation obstructs the heartbeat signal. **Exception** The server process is running, but its realm definition is out-of-date. The deployment caused an error in the server, so the server process continues to use its old realm definition. Restarting the server process might *not* be sufficient to resolve the issue. For example, the deployment specifies a new TCP transport, but the port is already bound in some other process. **Out-of-Sync** The server process' realm definition is a different revision than the realm server's.
ID	Client ID of the eFTL server process. To see more detailed information about the server process, click this ID.
Host	Host name of the eFTL server's host computer.

Server Commands

Icons at the right of each client row trigger commands:

- Change logging level of an eFTL server process.

  For the meaning of log level values, see "Log Level Reference" in *TIBCO FTL Development*.

Clients Trust the Server

When TLS security is enabled, client applications connect to the eFTL server using the WSS protocol. Clients must trust the server’s certificate.

The Java and Objective C APIs include a method to register the server certificates that the application trusts.

In .NET, install the server's certificate into the Microsoft certificate store before starting an eFTL client program. For details, see Microsoft documentation.

When running JavaScript applications in a Node.js environment, the application must register the server certificates that the application trusts. (However, the JavaScript method that registers certificates applies only when running in a Node.js environment. It is not effective when running in a browser.)

When running JavaScript applications in a browser, the browser is responsible for trusting server certificates. Unless the browser already trusts the eFTL server’s certificate, it will not connect to the server. To register trust in the server, users must complete the *Accepting a Self-Signed Server Certificate* task listed in the *TIBCO eFTL ™ Administration* guide.

Accepting a Self-Signed Server Certificate

To instruct your browser to accept the eFTL server’s certificate, complete this task.

Procedure

1. Enter the server’s URL in the browser’s URL pane, using this template:
   ```
   https://eFTL_server_host:port
   ```
   The browser displays a security dialog.
2. Inspect the server’s certificate to ensure that it is genuine and correct.

   If it is not, then do not complete the remaining steps of this task.
3. Resolve the security issue by trusting the server’s certificate.

   Some browsers use alternate terminology, such as *accepting* the certificate, or *allowing an exception* for this server or certificate.
4. Connect the browser to the server using this address template:
   ```
   wss://eFTL_server_host:port/channel
   ```

Configuring Push Notifications

To enable push notifications for an application's channel, complete these configuration steps in the TIBCO eFTL server.

Prerequisites

The application already has an ID and authentication with the platform vendor's push notification service (PNS).

Procedure

1. If your application communicates over more than one channel, complete the following steps for each relevant channel.
2. If users can run the application in conjunction with more than one PNS, complete this step for each relevant PNS.

PNS	Action
Google Cloud Messaging	Enter your application's API key.
Apple Push Notification Service	Enter your application's certificate and certificate password. Select either the production environment or the development sandbox.

eFTL Objects in the FTL Realm Server Web API

Administrators use the TIBCO FTL realm server GUI and web API to configure, monitor, and manage the TIBCO eFTL service.

The topics that follow present the details of the web API.

For background information about the web API, see "Realm Server Web API" in *TIBCO FTL Administration*.

For details of the realm server GUI that affect eFTL, see instead:

- Cluster and Server Configuration
- Server Monitoring

eFTL Definition Objects

The realm server web API represents each eFTL definition as a JSON object.

Example JSON Representation

```
{
  "eftl_clusters":[{
      "channels":[{
          "apns_cert":"",
          "apns_cert_pass":"",
          "apns_sandbox":false,
          "client_reconnect_timeout":60.0,
          "client_timeout":600.0,
          "description":"",
          "dynamic_durable_template":"",
          "ftl_exchange_format":"",
          "gcm_api_key":"",
          "max_message_size":"8kb",
          "max_pending_acks":100,
          "max_queue_size":100,
          "name":"channel",
          "persistence_store":"my_pstore",
          "provider_type":"ftl",
          "server_heartbeat":240.0,
          "static_durables": [
             {
              "name": "my_subscr_name",
              "durable": "a_static_dur_name"
             }
            ],
      "description":"",
      "jaas_required":false,
      "max_connections":500,
      "name":"Cluster"
    }]
}
```

JSON Attributes

For attributes and semantics, see Channel Details Panel.

GET realm/eftl

The web method `GET realm/eftl` retrieves a collection of eFTL cluster definitions.

Example Requests

```
curl GET http://host:port/api/v1/realm/eftl
```

GET realm/eftl/*clus_name*

The web method `GET realm/eftl/`*`clus_name`* retrieves the definition of an eFTL cluster by its name.

clus_name in the URI is the name of the eFTL cluster.

Example Requests

```
curl GET http://host:port/api/v1/realm/eftl/clus_name
```

GET realm/eftl/name/channels

The web method `GET realm/eftl/`*`clus_name`*`/channels` retrieves the definitions of all eFTL channels.

clus_name in the URI is the name of the eFTL cluster.

Example Requests

```
curl GET http://host:port/api/v1/realm/eftl/clus_name/channels
```

GET realm/eftl/*clus_name*/channels/*chan_name*

The web method `GET realm/eftl/`*`clus_name`*`/channels/`*`chan_name`* retrieves the definition of a channel by its name.

- *clus_name* in the URI is the name of the eFTL cluster.
- *chan_name* in the URI is the name of the channel.

Example Requests

```
curl GET http://host:port/api/v1/realm/eftl/clus_name/channels/
chan_name
```

POST realm/eftl

The web method `POST realm/eftl` creates a new eFTL cluster definition.

Input Data

Supply a JSON representation of the definition in the message body.

Example Requests

```
curl POST http://host:port/api/v1/realm/eftl json_defn
```

POST realm/eftl/*clus_name*/channels

The web method `POST realm/eftl/`*`clus_name`*`/channels` creates a new channel definition.

clus_name in the URI is the name of the eFTL cluster.

Input Data

Supply a JSON representation of the definition in the message body.

Example Requests

```
curl POST http://host:port/api/v1/realm/eftl/clus_name/channels
json_defn
```

DELETE realm/eftl/*clus_name*

The web method `DELETE realm/eftl/clus_name` deletes an eFTL cluster definition.

clus_name in the URI is the name of the eFTL cluster.

Example Requests

```
curl POST http://host:port/api/v1/realm/eftl/clus_name
```

DELETE realm/eftl/*clus_name*/channels/*chan_name*

The web method `DELETE realm/eftl/clus_name/channels/chan_name` deletes a channel from an eFTL cluster definition.

- *clus_name* in the URI is the name of the eFTL cluster.
- *chan_name* in the URI is the name of the channel.

Example Requests

```
curl POST http://host:port/api/v1/realm/eftl/clus_name/channels/
chan_name
```

PUT realm/eftl/*clus_name*

The web method `PUT realm/eftl/clus_name` updates an eFTL cluster definition.

clus_name in the URI is the name of the eFTL cluster.

Input Data

Supply a JSON representation of the new definition in the message body.

Example Requests

```
curl PUT http://host:port/api/v1/realm/eftl/clus_name json_defn
```

PUT realm/eftl/*clus_name*/channels/*chan_name*

The web method `PUT realm/eftl/clus_name/channels/chan_name` updates a channel definition within an eFTL cluster definition.

- *clus_name* in the URI is the name of the eFTL cluster.
- *chan_name* in the URI is the name of the channel.

Input Data

Supply a JSON representation of the new definition in the message body.

Example Requests

```
curl PUT http://host:port/api/v1/realm/eftl/clus_name/channels/
chan_name json_defn
```

eFTL Status Objects

The realm server web API represents the operating states of TIBCO eFTL clusters as a JSON object.

JSON Attributes of eFTL

TIBCO eFTL servers are client processes of the realm server. The set of JSON attributes for eFTL servers is identical to the set of attributes for FTL clients in general.

Client ID

The realm server assigns a unique client ID to each client. The ID is unique with the realm and across time. IDs are also unique across the different types of clients: ordinary application clients, and TIBCO FTL component clients, such as bridges, eFTL servers, and persistence servers.

See Also

For information about filtering and monitoring options, see the following topics in *TIBCO FTL Administration*:

- "Filters for Client Status Methods"
- "HTTP Parameters for Filtering"
- "HTTP Parameters for Client Monitoring"

GET eftl

The web method `GET eftl` retrieves a collection of eFTL cluster status objects.

Example Requests

```
curl GET http://host:port/api/v1/eftl
```

GET eftl/*clus_name*

The web method `GET eftl/clus_name` retrieves the status of a specific eFTL cluster.

clus_name in the URI is the cluster name.

Example Requests

```
curl GET http://host:port/api/v1/eftl/clus_name
```

GET eftl/*name*/clients

The web method `GET eftl/clus_name/clients` retrieves the status of the eFTL clients in a specific eFTL cluster.

clus_name in the URI is the cluster name.

HTTP Parameters

Syntax	Description
`client_id=eFTL_client_ID`	Optional. When present, limit the query to a specific client ID.

Syntax	Description
`user=`*`user_name`*	Optional. When present, limit the query to clients with a specific user name.

Example Requests

```
curl GET http://host:port/api/v1/eftl/clus_name/clients
```

GET eftl/*name*/channels

The web method `GET eftl/`*`clus_name`*`/channels` retrieves the status of the eFTL channels in a specific eFTL cluster.

clus_name in the URI is the cluster name.

Example Requests

```
curl GET http://host:port/api/v1/eftl/clus_name/channels
```

GET eftl/*name*/channels/*name*/clients

The web method `GET eftl/`*`clus_name`*`/channels/`*`chan_name`*`/clients` retrieves the status of the eFTL clients in a specific eFTL channel.

clus_name in the URI is the cluster name.

chan_name in the URI is the channel name.

HTTP Parameters

Syntax	Description
`client_id=`*`eFTL_client_ID`*	Optional. When present, limit the query to a specific client ID.
`user=`*`user_name`*	Optional. When present, limit the query to clients with a specific user name.

Example Requests

```
curl GET http://host:port/api/v1/eftl/clus_name/channels/chan_name/
clients
```

GET eftl/*clus_name*/servers

The web method `GET eftl/`*`clus_name`*`/servers` retrieves the status of the eFTL servers in a specific eFTL cluster.

clus_name in the URI is the cluster name.

Example Requests

```
curl GET http://host:port/api/v1/eftl/clus_name/servers
```

GET eftl/*clus_name*/servers/*ID*

The web method `GET eftl/clus_name/servers/ID` retrieves the status of a specific eFTL server in a specific eFTL cluster.

- *clus_name* in the URI is the cluster name.
- *ID* in the URI is the client ID of the eFTL server.

Example Requests

```
curl GET http://host:port/api/v1/eftl/clus_name/servers/ID
```

POST eftl/*name*

The web method `POST eftl/clus_name` sends a command concerning a specific eFTL cluster.

- *clus_name* in the URI is the cluster name.

Commands

Command	Description
`{"cmd":"shutdown"}`	Shutdown the servers of the eFTL cluster.

Example Requests

```
curl POST http://host:port/api/v1/realm/eftl/clus_name json_cmd
```

```
curl POST http://host:port/api/v1/realm/eftl/clus_name -d
'{"cmd":"shutdown"}'
```

POST eftl/*name*/channels/*name*

The web method `POST eftl/clus_name/channels/chan_name` sends a command concerning a specific eFTL channel.

- *clus_name* in the URI is the cluster name.
- *chan_name* in the URI is the channel name.

Commands

Command	Description
`{"cmd":"disable",{"args":[{"reason":"string"}]}}`	Disable the channel so clients cannot connect to it. The eFTL server includes the `reason` string in failure responses to clients.
`{"cmd":"enable"}`	Enable the channel so clients can connect to it again.

Example Requests

```
curl POST http://host:port/api/v1/realm/eftl/clus_name/channels/
chan_name json_cmd
```

```
curl POST http://host:port/api/v1/realm/eftl/clus_name/channels/
chan_name -d '{"cmd":"enable"}'
```

POST eftl/*name*/channels/*name*/clients/*ID*

The web method `POST eftl/`*`clus_name`*`/channels/`*`chan_name`*`/clients/`*`ID`* sends a command concerning a specific eFTL client.

- *clus_name* in the URI is the cluster name.
- *chan_name* in the URI is the channel name.
- *ID* in the URI is the client ID of the eFTL client.

Commands

Command	Description
`{"cmd":"disconnect"}`	Disconnect the client from the eFTL server.

HTTP Parameters

Syntax	Description
`client_id=`*`eFTL_client_ID`*	Optional. When present, limit the query to a specific client ID.
`user=`*`user_name`*	Optional. When present, limit the query to clients with a specific user name.

Example Requests

```
curl POST http://host:port/api/v1/realm/eftl/clus_name/channels/
chan_name/clients/ID json_cmd
```

```
curl POST http://host:port/api/v1/realm/eftl/clus_name/channels/
chan_name/clients/ID -d '{"cmd":"disconnect"}'
```

POST eftl/*clus_name*/servers/*ID*

The web method `POST eftl/`*`clus_name`*`/servers/`*`ID`* changes the log level of a specific eFTL server in a specific eFTL cluster.

- *clus_name* in the URI is the cluster name.
- *ID* in the URI is the client ID of the eFTL server.

Input Data

Supply the log level command in JSON format. For example:

```
{"cmd":"setloglevel",
 "args":[{"level","element:level"}]}
```

.

For details, see "Tuning the Log Level" in *TIBCO FTL Development*.

Example Requests

```
curl POST http://host:port/api/v1/realm/eftl/clus_name/servers/ID
json_cmd
```

Coordination Forms

Coordination forms help developers and administrators to agree upon application details and to document those details.

You can duplicate these PDF forms, renaming the copies, and complete them online. Alternatively, you can print them and complete them on paper.

- TIBCO eFTL Application Coordination Form
- TIBCO eFTL EMS Channel Coordination Form
- TIBCO eFTL FTL Channel Coordination Form
- TIBCO eFTL Server Coordination Form