



Administrator's Guide

Version **2019.0**

Copyright

Copyright © 2000-2019, NICE s.r.l.

All right reserved.

We'd Like to Hear from You

You can help us make this document better by telling us what you think of the content, organization, and usefulness of the information. If you find an error or just want to make a suggestion for improving this document, please address your comments to <documentation@nice-software.com>. Please send only comments regarding NICE documentation.

For product support, contact <helpdesk@nice-software.com>.

Although the information in this document has been carefully reviewed, NICE s.r.l. ("NICE") does not warrant it to be free of errors or omissions. NICE reserves the right to make corrections, updates, revisions, or changes to the information in this document.

UNLESS OTHERWISE EXPRESSLY STATED BY NICE, THE PROGRAM DESCRIBED IN THIS DOCUMENT IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL NICE BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION ANY LOST PROFITS, DATA, OR SAVINGS, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS PROGRAM.

Document Redistribution and Translation

This document is protected by copyright and you may not redistribute or translate it into another language, in part or in whole, without the express written permission of NICE s.r.l.

Trademarks

EnginFrame, Neutro, Remote File Browsing, Service Definition File, EnginFrame Agent are registered trademarks or trademarks of NICE s.r.l. in Italy and other countries.

Amazon™ is a registered trademark of Amazon.com, Inc.

Apache®, Apache Derby®, Tomcat® are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries.

Oracle®, Sun®, MySQL®, JavaScript® and Java™ are registered trademarks of Oracle and/or its affiliates.

Unix® is a registered trademark of The Open Group in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Microsoft®, Windows® and Internet Explorer® are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.

Firefox® and Mozilla® are trademarks or registered trademarks of the Mozilla Foundation in the United States and/or other countries.

Apple®, Mac®, Mac® OS X® and Apple® Safari® are trademarks or registered trademarks of Apple, Inc. in the United States and other countries.

Citrix®, XenDesktop®, Citrix Receiver™ are trademarks of Citrix Systems, Inc. and/or one or more of its subsidiaries, and may be registered in the United States Patent and Trademark Office and in other countries.

IBM®, IBM® Platform™ LSF® are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Altair® PBS Professional® is a trademark of Altair Engineering, Inc.

Univa® and Univa® Grid Engine® (UGE) are trademarks of Univa Corporation.

SLURM™ is a trademark of SchedMD LLC.

RealVNC® and VNC® are trademarks of RealVNC Limited and are protected by trademark registrations and/or pending trademark applications in the European Union, United States of America and other jurisdictions.

Adaptive Computing®, Moab® and other Adaptive Computing® products are either registered trademarks or trademarks of Adaptive Computing Enterprises, Inc.

HP® is a registered trademark of HP Inc.

Google™ and Chrome™ are trademarks of Google Inc.

Red Hat® is a trademark of Red Hat, Inc.

SUSE® is a registered trademark of SUSE Linux AG.

Other names mentioned in this document may be trademarks of their respective owners.

Last Update

February 27, 2019 (rev. 915)

Latest Version

<https://www.nice-software.com/download/enginframe>

毅碩科技 (CAX-CLOUD)
info@cax-cloud.com

聚辰科技(CAX-CLOUD)
info@cax-cloud.com

Contents

Welcome	ix
About This Guide	ix
Who Should Read This Guide	ix
What You Should Already Know	ix
Learn About NICE Products	x
World Wide Web	x
NICE EnginFrame Training	x
NICE EnginFrame Documentation	x
Get Technical Support	xi
NICE Support Contacts	xi
Collect Support Information	xi
I. Getting Started	1
1. About NICE EnginFrame	3
Architectural Overview	3
Basic Execution Flow	4
Basic Deployment	6
Distributed Deployment	8
File Downloads	10
Interactive Session Broker	11
EnginFrame Plugins	13
EnginFrame Enterprise	14
Architecture	14
Software Distribution and License	15
Deployment	16
2. Obtaining NICE EnginFrame	19
Downloading EnginFrame	19
Obtaining a License	19
Licensed Plug-ins	19
3. Planning NICE EnginFrame Deployment	21
Prerequisites	21
System Requirements	21
Third-party Software Prerequisites	21
Network Requirements	32
Supported Browsers	32
Interactive Plugin Requirements	33
EnginFrame Enterprise System Requirements	34
Deployment Strategies	35
Installation Directories	36
Special Users	38
Authentication	38
DRM Configuration for Interactive Plugin	38
NICE Neutro	39
IBM® Platform™ LSF® or OpenLava	39
Torque or PBS Professional®	42

SGE, Oracle® Grid Engine (OGE), Son of Grid Engine (SoGE) or Univa® Grid Engine® (UGE)	43
Adaptive Computing® Moab®	44
SLURM™	44
4. Installing NICE EnginFrame	45
Installing	45
Batch Installation	46
Fine Tuning Your Installation	46
Spooler Download URL	47
Optimizing JDK Options	47
Distributed Resource Manager Options	47
Interactive Plugin	48
EnginFrame Enterprise Installation	49
Load Balancer Setup	49
DBMS Setup	50
EnginFrame Service Configuration	53
EnginFrame Start	54
5. Running NICE EnginFrame	55
Start, Stop, and Check Status	55
Accessing the Portal	58
Demo Sites	58
Administration Portal	58
Monitor Services	59
Troubleshooting Services	59
EnginFrame Statistics	59
Applications Portal	60
Admin's Portal	60
User's Portal	60
Views Portal	61
Admin's Portal	61
User's Portal	61
II. Administration	63
6. Common Administration Tasks	65
Main Configuration Files	65
Deploying a New Plugin	68
NICE's Official Plugins	68
Custom Plugins	68
Changing Java™ Version	69
Changing Default Agent	69
Managing Internet Media Types	70
Customizing Error Page	73
Limiting Service Output	74
Configuring Agent Ports	74
Customizing User Switching	75
Customizing User Session Timeout	77
Apache®-Tomcat® Connection	77
Changing Charts Backend	79
Interactive Administration	79
Configuration Files	79
Interactive Session Life-cycle Extension Points	86

Session limits	90
Log files	91
Interactive Plugin Directory Structure	91
Views Administration	93
Configuration Files	93
Log files	98
VDI Plugin Directory Structure	98
Applications Administration	99
Configuration Files	99
Log files	102
Applications Directory Structure	102
7. Managing Spoolers	105
Spoolers Requirements	106
Spooler Security Permissions	106
Configuring EnginFrame Spoolers	108
Configuring Spoolers Default Root Directory	108
Download Files From Spoolers	108
Spooler Life Cycle	110
Overview	110
Change Repository Location	110
Configure Reaper Sleep Time	111
Spoolers Removal: Dead Spoolers	111
8. Managing Sessions Directory	113
Sessions Requirements	114
9. Customizing Logging	115
Tomcat® Logging	115
EnginFrame Server and Agent Logging	115
Configuration Files	116
Change Log Files Location	116
Change Log Files Size and Rotation Policy	117
Change Log Level	118
Fine Tune Logging	118
EnginFrame Scriptlet Logging	121
10. EnginFrame Licenses	123
License Files Management	123
Configuring License Files Location	123
License File Format	124
License Checking	125
License Token Count	125
Monitoring License Usage	127
Enable Debug Log Messages for Licenses	129
III. Security	131
11. Authentication Framework	133
Standard EnginFrame Authentication Authorities	133
Default Authority	133
User Mapping	134
Configuring NICE EnginFrame Authorities	135
PAM	135
LDAP	136
Active Directory	136

HTTP	136
Certificate	137
Custom Authentication Authority	137
The <authority>.login File	138
The ef.auth File	139
12. Authorization System	141
Configuring Authorization	141
Defining Actors	142
Defining Access Control Lists	143
Condition Based ACL	145
13. Configuring HTTPS	149
Change HTTPS certificates	149
Index	151

About This Guide

This guide provides information about installing, configuring, managing, and maintaining an instance of NICE EnginFrame portal.

Who Should Read This Guide

This guide is intended for system administrators who will install and administer an instance of NICE EnginFrame portal but are not necessarily creating new services.

What You Should Already Know

This guide assumes:

- You have knowledge of Unix® system administration tasks such as creating user accounts, sharing and mounting Network File System (NFS) partitions, backing up the system, etc.
- You have basic knowledge about web-related technologies like the HTTP protocol, the SSL protocol, the XML language, etc.

Learn About NICE Products

World Wide Web

You can find the latest information about NICE EnginFrame on its web site <https://www.nice-software.com>.

For more information about other NICE products and about the professional services provided by NICE you can refer to the company's web site <https://www.nice-software.com>.

Report problems accessing the aforementioned web sites to [<helpdesk@nice-software.com>](mailto:helpdesk@nice-software.com).

NICE EnginFrame Training

Training classes offered by NICE can help you master the skills needed to productively configure, manage, and maintain your EnginFrame Portal.

Classes are available at our corporate headquarters and other NICE locations.

Customized on-site classes are also available.

Find out more about NICE training at <https://www.nice-software.com> or for further details contact [<info@nice-software.com>](mailto:info@nice-software.com).

NICE EnginFrame Documentation

The latest NICE EnginFrame documentation is available at <https://www.nice-software.com/download/enginframe>.

Visit NICE web site at <https://www.nice-software.com> and get your personal access code to the documentation area or contact [<documentation@nice-software.com>](mailto:documentation@nice-software.com).

Get Technical Support

Contact NICE or your EnginFrame reseller for technical support.

NICE Support Contacts

Use one of the following to contact NICE technical support.

Email

[<helpdesk@nice-software.com>](mailto:helpdesk@nice-software.com)

World Wide Web

<https://www.nice-software.com>

Phone

+39 0141 901516

Mail

NICE Support
c/o NICE s.r.l.
Via Milliavacca, 9
14100 Asti
Italy

When contacting NICE, please include your company's full name.

Collect Support Information

Please use "*Support/Collect support info*" service described in the section called “Administration Portal” to collect some preliminary data that could help NICE support team speed up time to process your support request.

The output of this service is a compressed archive containing all the gathered information. Please send the compressed archive to NICE support team attached to your request.

聚辰科技(CAX-CLOUD)
info@cax-cloud.com

PART I

Getting Started

聚辰科技(CAX-CLOUD)
info@cax-cloud.com

About NICE EnginFrame

EnginFrame is the leading grid-enabled application portal for user-friendly HPC job submission, control, and monitoring. It includes sophisticated data management for all stages of job lifetime and is integrated with most important job schedulers and middleware tools to submit, monitor, and manage jobs.

EnginFrame provides a modular system where new functionality (e.g. application integrations, authentication sources, license monitoring, etc.) can be easily added. It also features a sophisticated Web Services interface which can be leveraged, for example, to enhance existing applications as well as developing custom solutions for your own environment.

A key advantage of EnginFrame is the rapid migration from command-line to *Computing Portal* paradigm, leveraging existing scripting solutions where available.

Based on the latest and most advanced Web 2.0 standards, it provides a flexible infrastructure to support current and future computing needs. As you can expect from a web-enabling solution, it is flexible in content presentation, providing personalized experience for users according to their role or operational context.

Architectural Overview

EnginFrame has an architecture layered into three tiers, as shown in Figure 1.1, “EnginFrame Architecture”:

- The *Client Tier* usually consists of the user's web browser. It provides an easy-to-use interface based on established web standards like XHTML and JavaScript®. This tier is independent from the specific software and hardware environment used by the end user. The Client Tier can also integrate remote visualization technologies like Virtual Network Computing (VNC®).
- The *Server Tier* consists of a *Server* that interacts with EnginFrame Agents and manages the interaction with users.
- The *Resource Tier* consists of one or more *Agents* deployed on the back-end infrastructure. Agents manage computing resources on user's behalf and interact with the underlying operating system, job scheduler, or grid infrastructure to execute EnginFrame services (e.g. starting jobs, moving data, retrieving cluster load, etc.)

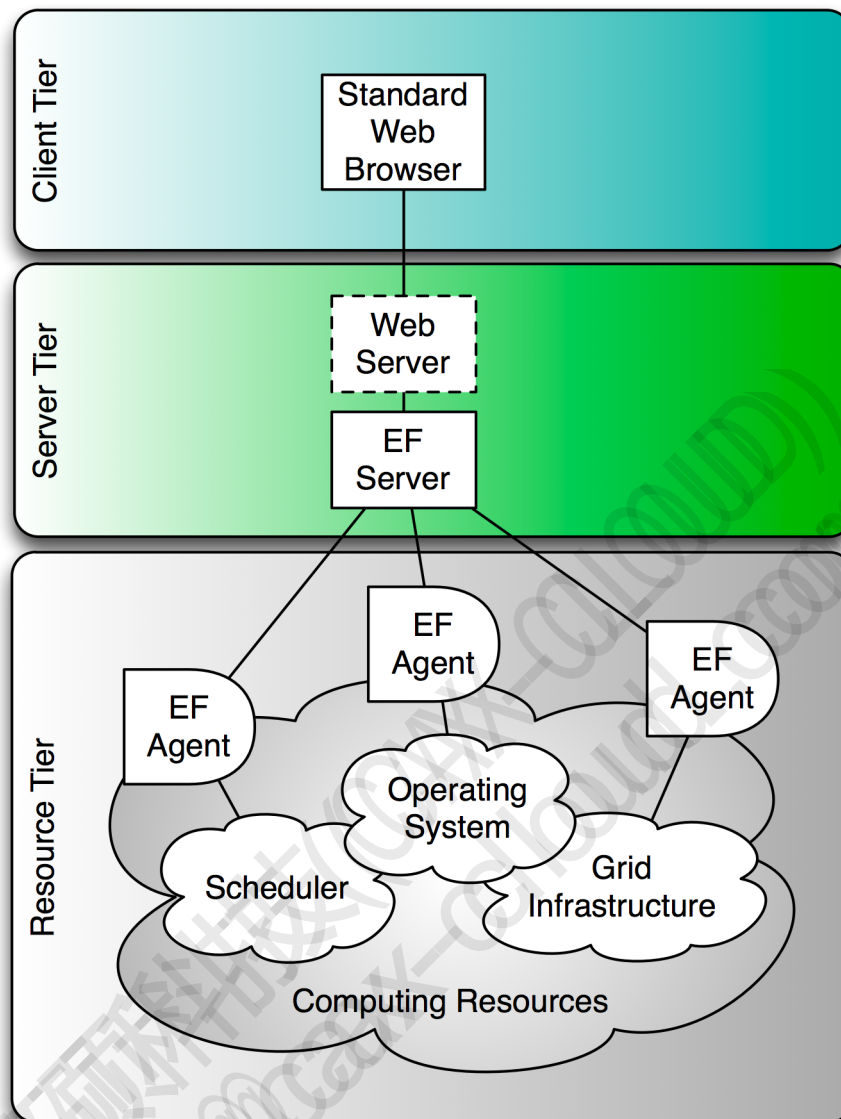


Figure 1.1. EnginFrame Architecture

Basic Execution Flow

EnginFrame abstracts computing resources and data management (*Resource Tier*) and exposes services to users (*Server Tier*) which in turn can use them directly from their browsers (*Client Tier*).

EnginFrame's internal structure reflects this high level architecture and revolves around two main software components: the EnginFrame Server and the EnginFrame Agent.

The EnginFrame Server

The EnginFrame Server is a Java™ Web Application and must be deployed inside a Java Servlet Container (EnginFrame ships *Apache Tomcat® 7.0.92*). It takes care of exposing services to users.

The EnginFrame Agent

The EnginFrame Agent is a stand-alone Java™ application which manages the computing resources and executes services on user's behalf (when running as `root`).

As you can see in Figure 1.2, “Interaction Diagram”, EnginFrame Server receives incoming requests from the Web Browser (1), authenticates and authorizes them and then asks an EnginFrame Agent (2) to execute the required actions.

Agents can perform different kind of actions, from the execution of a simple command on the underlying operating system to the submission of a job on the grid infrastructure (3).

The results of the executed action are gathered by the Agent (4) and sent back to the Server (5).

The Server applies some post processing transformations, filters output according to defined access control lists (ACL), and transforms the results into an HTML page (6).

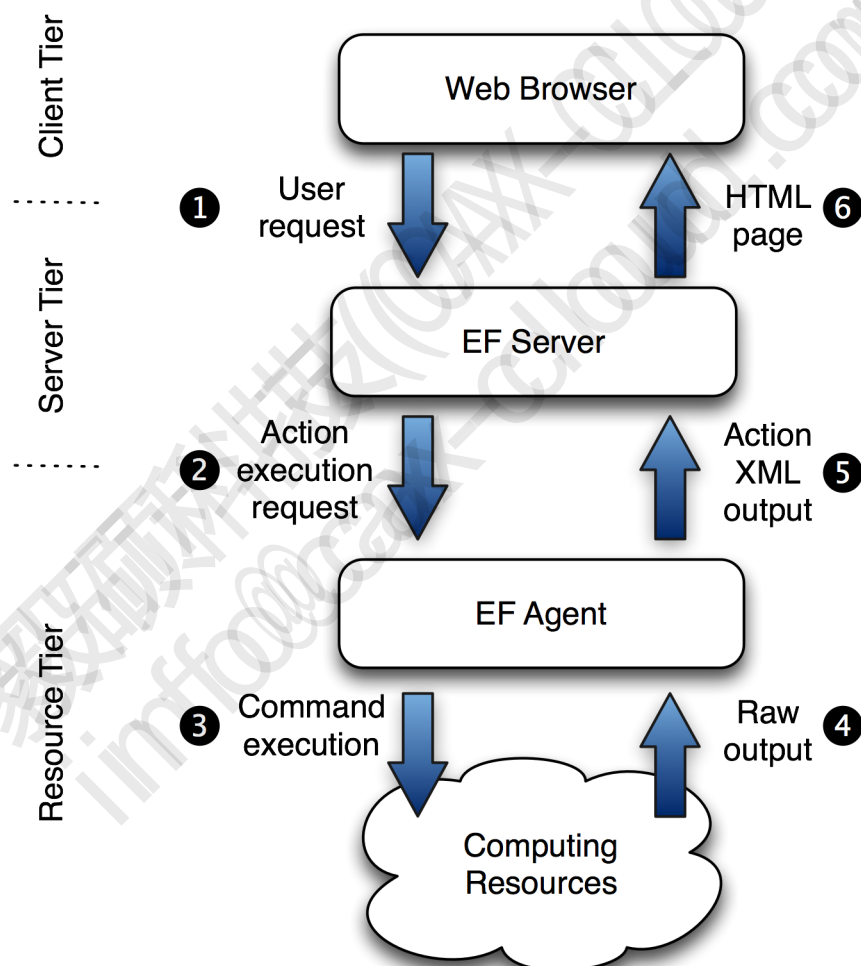


Figure 1.2. Interaction Diagram

EnginFrame creates (or reuses) a data area each time a new action is executed. This area is called *spooler*.

The spooler is the action's working directory. It contains files uploaded during action submission. Users can only download files from their spoolers.

The spooler is located on a file-system readable and writable by both Server and Agent. Refer to [Chapter 7, *Managing Spoolers*](#) for further details.

Basic Deployment

When EnginFrame Portal is installed on one host it is called *basic installation*. As shown in [Figure 1.3, “EnginFrame Deployed on One Host”](#), the Server Tier also contains the Agent used to access Resource Tier. User `efnobody` (default user chosen during installation procedure) runs the Server in this scenario. The EnginFrame Server contains a *local Agent* that is used when:

- Expressly configured in your service description.
- Submitting a scriptlet.

Normally the Server contacts a *remote Agent*, configured as the default one during setup procedure. Usually the remote Agent runs as `root` and can:

- Authenticate users using PAM/NIS.
- Create/delete spoolers on user's behalf.
- Execute services on user's behalf.
- Download files on user's behalf.

The remote Agent can also run as unprivileged user but you lose the main features of an Agent running as `root`: all spoolers, services, etc. are created/executed as this unprivileged user. It also implies that EnginFrame has to use an authentication module that does not require `root` privileges to check credentials (e.g., *ldap*, *activedirectory*, etc.)

The Server communicates with the Remote Agent using [Java™ RMI protocol](#), while the local Agent is reached directly since it lives inside the Server's JVM space.

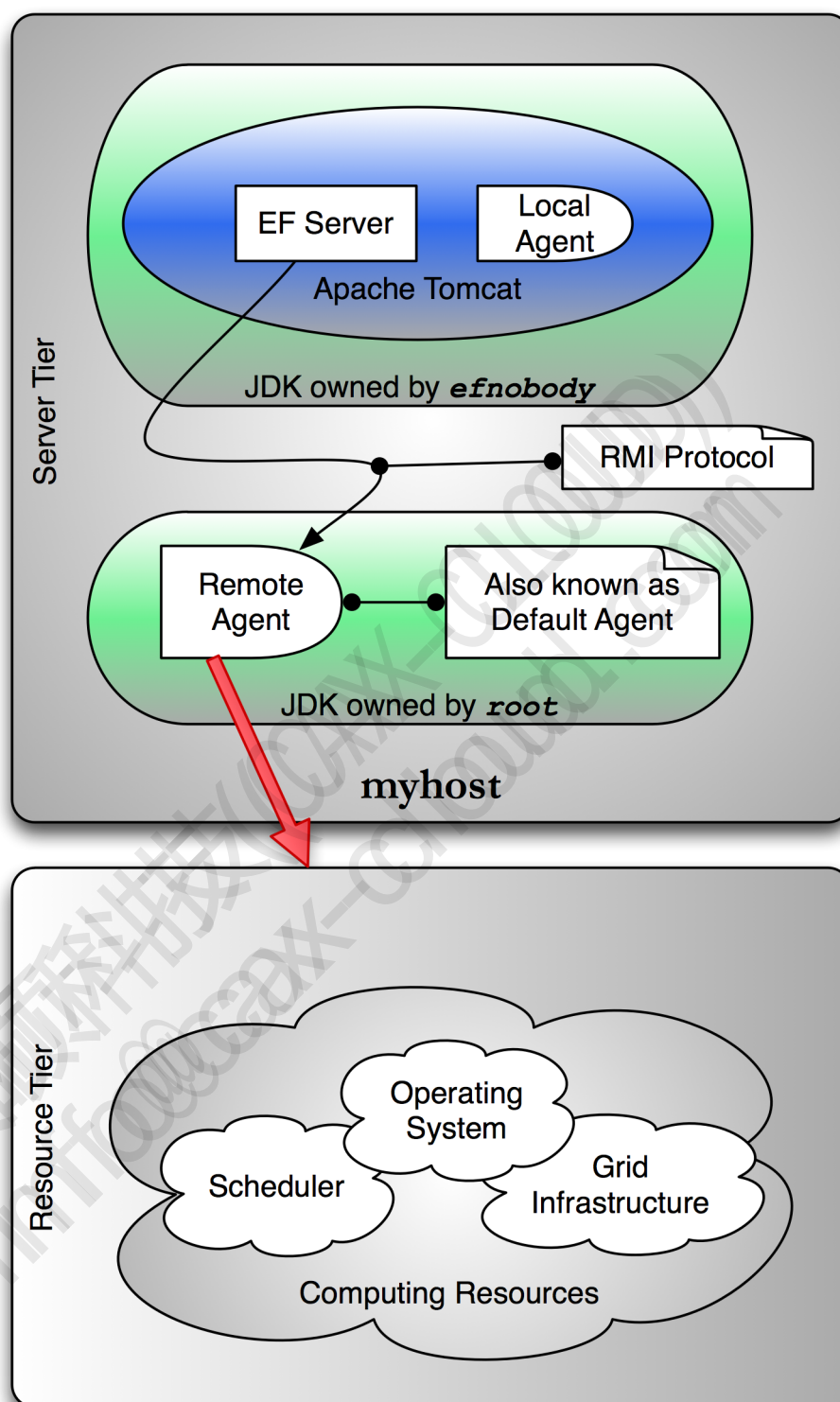


Figure 1.3. EnginFrame Deployed on One Host

Distributed Deployment

EnginFrame Server can be deployed in a demilitarized zone (DMZ) accessible from Intranet/Internet while the default EnginFrame Agent resides in your protected computing environment. EnginFrame Server and EnginFrame Agent reside on different hosts in this scenario called *distributed deployment*. You can see this in [Figure 1.4, “EnginFrame Deployed on More Hosts”](#).

In this scenario the following requirements have to be met:

- Server host reaches Agent host on ports specified during setup.
- Agent host reaches Server host via HTTP on port specified during setup.
- Spoolers are stored on a shared file-system.
- Spooler shared file-system is readable and writable by both `efnobody` and `root`.

The Server has to reach the Agent via RMI otherwise user's submissions fail.

The Agent has to reach the Server via HTTP otherwise user's downloads fail.



Why spoolers have to reside on a shared file-system

The Server saves files sent by users while the service executed on Agent needs to access them. Since files are written by the Server, `efnobody` needs read access to traverse the directory structures while creating new spoolers and write access to write files; since services are executed on user's behalf, `root` needs write permissions on spoolers area in order to give directory and files ownership to the user executing the service. This ownership change is necessary since the spooler and the files were created by `efnobody`.

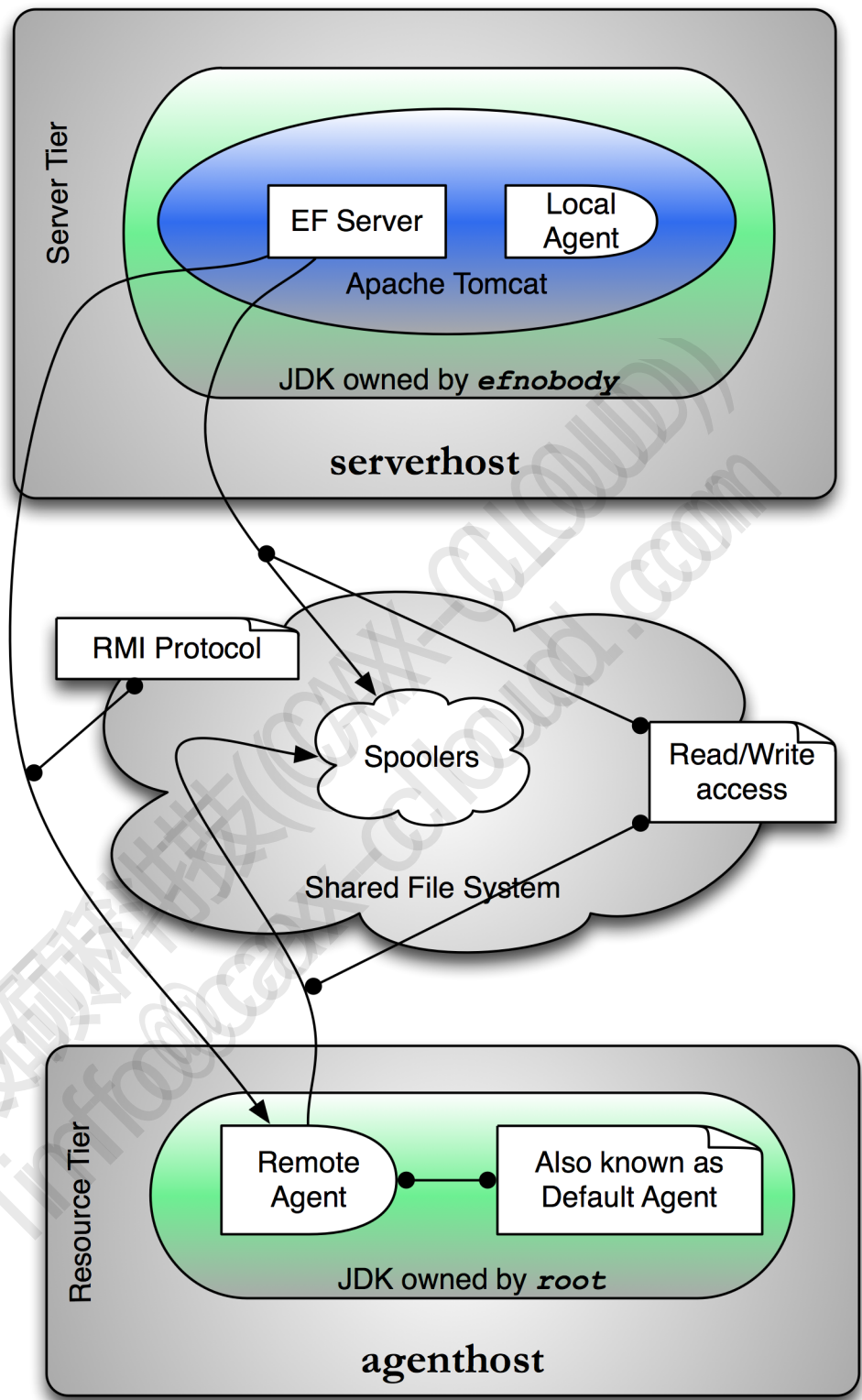


Figure 1.4. EnginFrame Deployed on More Hosts

File Downloads

Users can only download files contained in their spoolers and [Figure 1.5, “File Download Interaction”](#) explains this process flow.

EnginFrame Server receives incoming requests from the Web Browser (1) and forwards request to EnginFrame Agent (2) which downloads the remote file.

EnginFrame Agent forks a process as user which reads the file (3).

EnginFrame Agent connects back to EnginFrame Server via HTTP to send back bytes produced by forked process (4).

EnginFrame Server sends back the bytes as are to browser (5).

The browser displays the file or proposes to save it on disk depending on file mime-type and browser settings (6). Refer to [the section called “Managing Internet Media Types”](#) for further details.

Step (4) highlights why it is important for EnginFrame Agent to reach EnginFrame Server via HTTP.

EnginFrame also allows you to download files in *streaming* mode. File contents are shown while it grows. This is useful for files that grow during service execution. The flow is the same as the remote file download except that EnginFrame Server polls, at fixed intervals, EnginFrame Agent for some fresh data. This feature mimics Unix® **tail** that displays the last file portion while it grows.

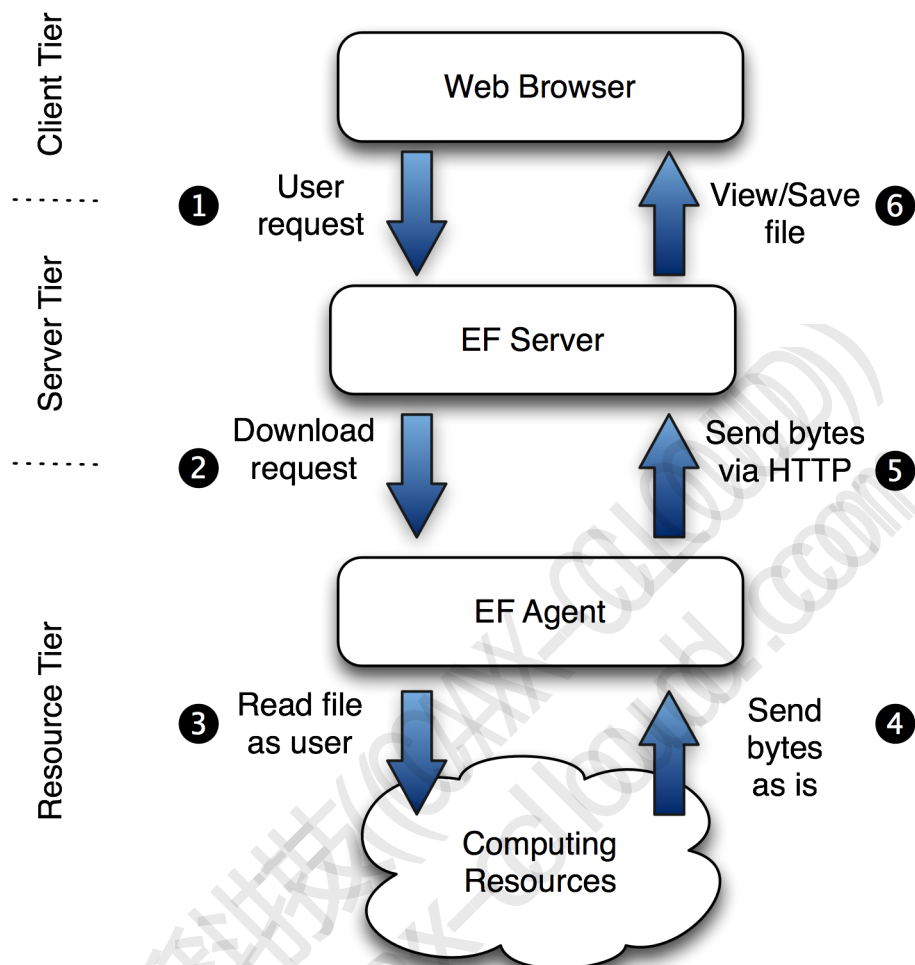


Figure 1.5. File Download Interaction

Interactive Session Broker

EnginFrame 2019.0 includes the *interactive* plugin, a session broker built ground up to be scalable and reliable. Its main purpose is to ease application delivery and manage interactive sessions.

Deployment

The solution relies on the following systems:

- NICE EnginFrame, the kernel on top of which Interactive Plugin is built.
- A resource manager software to allocate and reserve resources according to the desired resource sharing policy or a third party session broker (Citrix® XenDesktop®).
- One or more remote visualization middlewares like NICE DCV, HP® RGS, VirtualGL or RealVNC®.

For a complete list of supported HPC workload managers, session brokers and visualization middlewares, please refer to the section called “Prerequisites”.

The visualization farm can be only Linux®, only Windows® or both.

The following picture represents the main components of the solution stack:

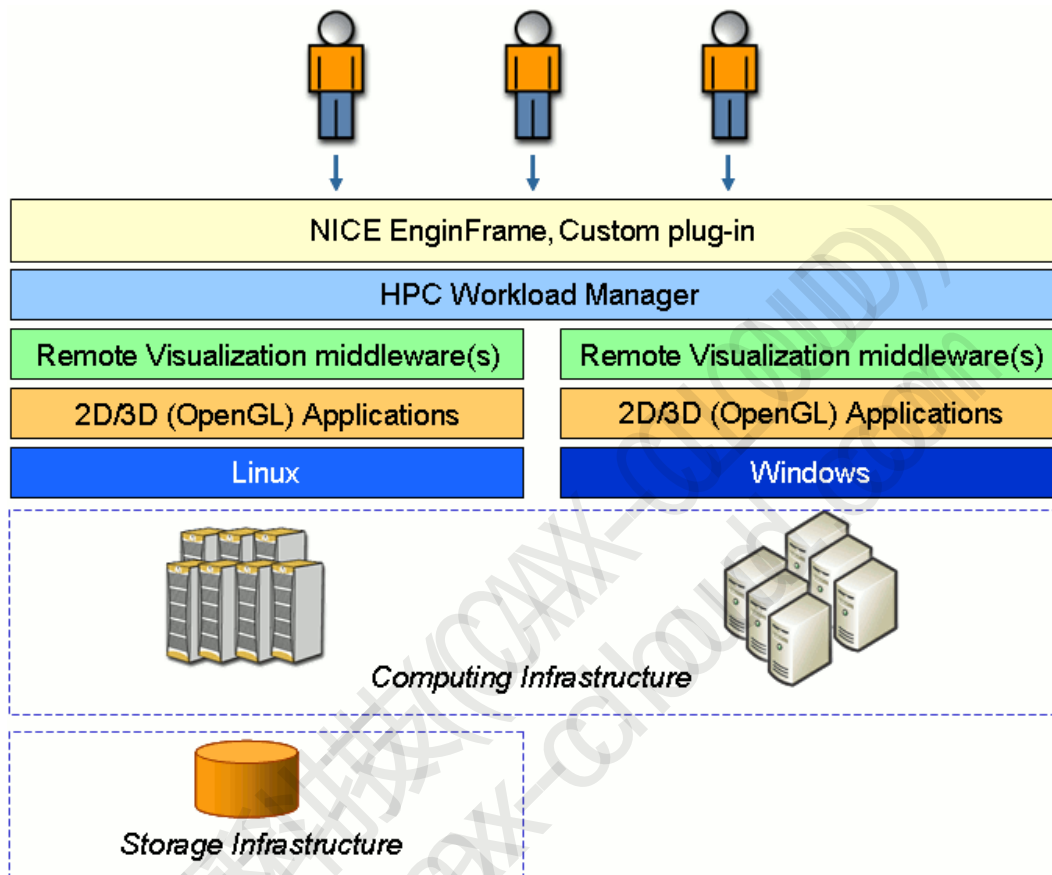


Figure 1.6. Interactive Plugin Architecture

Execution Flow

The following picture represents a real world example infrastructure including nodes with NICE DCV, HP® RGS and RealVNC®. This infrastructure was designed and deployed to deliver 3D applications on Linux® through NICE DCV, 3D applications on Windows® through HP® RGS, 2D applications on Windows® through XenDesktop® and 2D applications on both OSes through VNC®.

The following explains what happens at each step:

1. The user connects to Interactive Plugin to create a new session. Each session is a distinct job of the underlying HPC workload scheduler or a distinct session in the underlying third party session broker.
2. The resource manager or the session broker schedules the new session on the most appropriate node that complies with the application requirements and the resource sharing policies.
3. Once the session is created, Interactive Plugin sends a file to the web browser. This file contains information that allows the browser to kickoff the correct visualization client. The client uses this information to connect to the remote session.

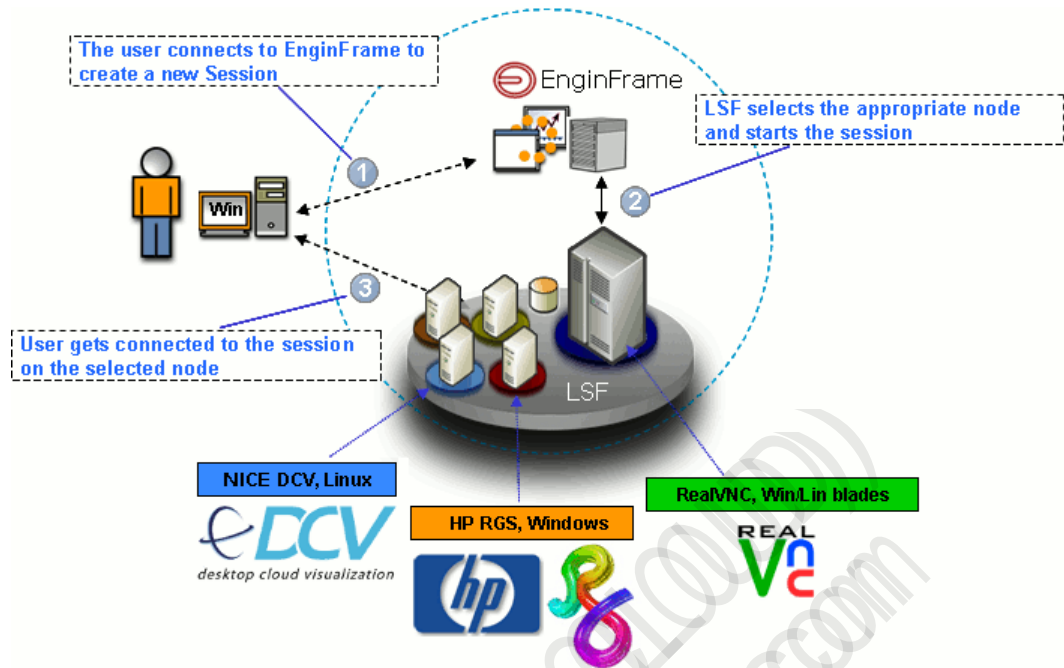


Figure 1.7. Interactive Plugin Use Model

EnginFrame Plugins

A plugin is a piece of software that extends EnginFrame Portal. NICE sells and gives for free many of these extensions. Please contact helpdesk@nice-software.com for a complete list of plugins and for pricing questions.

The plugins can extend EnginFrame in many different areas:

- **Bundle** - a full featured package containing other plug-ins
- **Kernel** - an extension that enhances EnginFrame core system, e.g. *WebServices*, *Interactive Plug-in*
- **Auth** - an extension that authenticates users against an authoritative source, e.g. *PAM Plug-in*
- **Data** - an extension that helps display data inside EnginFrame Portal, e.g. *File Manager*, *RSpooler Plug-in*
- **Grid** - an extension that connects EnginFrame Portal with a grid manager, e.g. *LSF Plug-in*
- **Util** - additional utility components, e.g. *Demo Portal*

NICE ships many plugins according to these conventions:

- **Certified extensions** - are developed and supported by NICE. They are available and supported as add-on products, which pass a quality assurance process at every new release of EnginFrame. Each extension is individually certified to work on the latest release of EnginFrame and guidelines are provided to evaluate how different groups of extensions may interact. No implicit commitment is taken about the compatibility between two different extensions.
- **Qualified extensions** - are developed or modified by NICE, which ensures a professional development and good functionality under some specific EnginFrame configuration. Qualified

extensions are available as project-accelerator solutions, to facilitate integration of your EnginFrame Portal in specific complex scenarios. Further support can be provided as Professional Services.

- **Contributed extensions** - are developed by third parties and made available by the respective authors. They are provided as-is and no additional endorsement is provided by NICE. Further support on such modules may be asked to the contributing authors, if available.

EnginFrame Enterprise

This section describes the EnginFrame Enterprise version, the solution aimed at enterprise environments where *load balancing* and *fault tolerance* are crucial requirements.

All the general concepts about EnginFrame explained in the previous sections apply also to EnginFrame Enterprise version. The following sections illustrate the characteristics of the Enterprise solution, describing the architecture, highlighting the differences with the architectures above, and suggesting the best approach for the deployment.

Architecture

The first aspect to highlight about EnginFrame Enterprise architecture as depicted in [Figure 1.8, “EnginFrame Enterprise Architecture”](#), is that it involves *multiple EnginFrame Servers* and *multiple EnginFrame Agents*. All the Servers and the Agents maintain the same role and functionalities as described in the previous sections but in an EnginFrame Enterprise infrastructure the EnginFrame Servers are also able to communicate each other by network to share and manage the system status.

The shared system status involves the following resources:

- users' spoolers and spoolers repository
- EnginFrame triggers
- logged-in users
- EnginFrame license tokens

Information are shared among EnginFrame Servers and managed in a distributed architecture where there is no "master" and so *no single point of failure* in the system. Each of the server alone can cover all the needed functionalities and, at the occurrence, it would be able to keep the whole system up and running, making the system more robust and fault-tolerant.

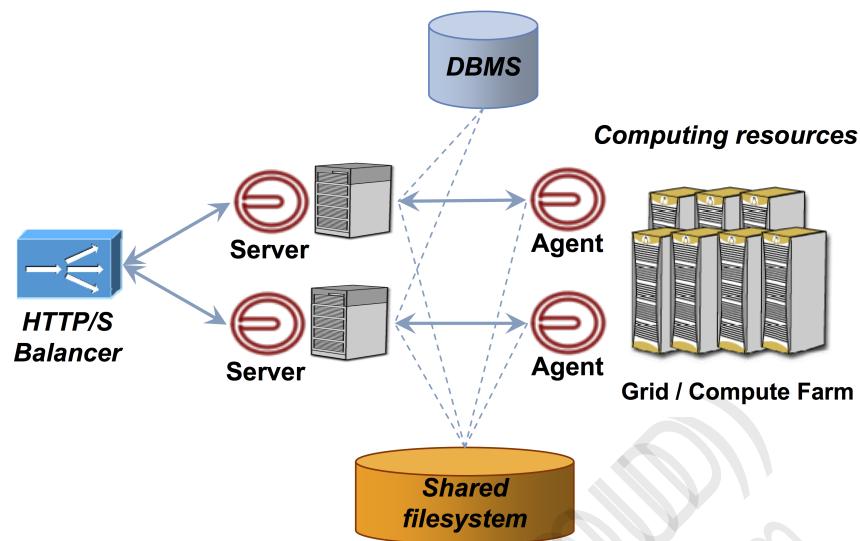


Figure 1.8. EnginFrame Enterprise Architecture

The EnginFrame Enterprise solution relies on a file-system that is not only shared between an EnginFrame Server and an EnginFrame Agent but it's shared among all the Servers and Agents. In more details, the EnginFrame Agents, as explained above, need access to the spoolers area while EnginFrame Servers have stronger requirements needing other file-system resources to be shared besides spoolers: the EnginFrame repository files containing server-side metadata about spoolers, the file upload cache, the plugins data directory tree, etc. Without going deeper into details, the [section called “Deployment”](#) describes the suggested and supported approach about file-system sharing.

Another important component to consider in the EnginFrame architecture is the Database Management System (DBMS). While in a standard EnginFrame installation you can rely on the database distributed with EnginFrame, that is Apache Derby®, in the EnginFrame Enterprise solution you must consider to have an external JDBC compliant DBMS. All the EnginFrame Servers then need to have access to the database.

The list of supported DBMS is summarized in [Table 3.1, “Supported Database Management Systems”](#).

As last component, in order to have a single point of access to EnginFrame, the architecture involves a front-end HTTP/S network load balancer. This component is not part of the EnginFrame Enterprise deployment but it's a third party solution, software or hardware (e.g. Cisco router 6500/7600 series), configured with the sticky session capability¹ that dispatches users' requests to the EnginFrame Servers in a balanced way.

NICE can provide and setup the network load balancer based on third party technology, e.g. Apache® Web server, as professional services activity according to specific projects with customers.

Software Distribution and License

EnginFrame Enterprise is distributed with the *same software package* of EnginFrame. It's the EnginFrame software license that actually enables EnginFrame Enterprise capabilities.

Here an example of an EnginFrame Enterprise license:

¹ Sticky session refers to the feature of many commercial load balancing solutions for Web-farms to route the requests for a particular session to the same physical machine that serviced the first request for that session. So the balancing occurs on Web sessions and not on the single received Web requests.

```
<?xml version="1.0"?>
<ef-licenses>
  <ef-license-group product="EnginFrame HPC ENT" release="2015.0" format="2">
    <ef-license
      component="EF Base"
      vendor="NICE"
      expiration="2015-12-31"
      ip="172.16.10.171,172.16.10.172"
      licensee="NICE RnD Team"
      type="DEMO"
      units="100"
      units-per-user="1"
      license-hosts="false"
      hosts-preemption="false"
      signature="MC0CFQCGPmb3lgpiGxxEr0DdyoYud..." <!-- Omitted -->
    />
  </ef-license-group>
</ef-licenses>
```

In particular please note the `product` attribute value: `EnginFrame HPC ENT`. This defines an EnginFrame license for HPC environments with the `ENT` string specifying the Enterprise version. Also note the `ip` attribute of tag `ef-license` with the list of the IP addresses of the licensed EnginFrame Servers nodes.

For information about how to get the EnginFrame software and license please refer to [Chapter 2, Obtaining NICE EnginFrame](#).

Deployment

Due to the inner distributed nature of its architecture the possible deployment scenarios of EnginFrame Enterprise can be quite different and vary in complexity.

You can have each component, EnginFrame Server or Agent, on a different node and you can decide to pick only the minimum parts of the file-system to share on each of the node. Remember that in an EnginFrame Enterprise deployment you also have file-system resources to be shared among EnginFrame Servers and in many cases, even when resources do not necessarily require sharing, they should anyway be replicated and maintained aligned among EnginFrame Servers.

Even if, in principle, it's possible to fine tune the installation of EnginFrame Enterprise taking into account the different factors as networking and file-system sharing, it's common practice to go for a much faster and easier-to-maintain deployment approach that is described here. A different approach from what is presented here, should be followed only in case of particular customer's needs and it requires to be discussed with NICE professional services.

The suggested and supported approach to EnginFrame Enterprise deployment involves

- one node for each pair of EnginFrame Server and Agent you want to install;
- a shared file-system for the whole `$EF_TOP` directory tree.

Where `$EF_TOP` is the top EnginFrame installation directory. For more details please refer to [the section called "Installation Directories"](#).

With this approach you are able to install and manage the software from just one node and all the binaries and data directories as spoolers, sessions, license, etc. will be shared among the installation nodes.

Please note that for those resources that are expected to be local and that would conflict in a shared environment, e.g. in the logging directory each Server (and Agent) writes log files with the same names, EnginFrame provides a *per-hostname* directory tree, making safe the sharing.

The external DBMS is suggested to reside on a different node(s) and possibly configured to be fault tolerant.

During installation of EnginFrame Enterprise you will be prompted to insert the JDBC URL to the EnginFrame database instance together with the username and password with which to access. The EnginFrame database instance must be previously created empty, then EnginFrame, at the first connection, will create all the needed tables.

The details that specifically concern an EnginFrame Enterprise deployment, its requirements and installation notes, are integrated where needed in the next chapters of this guide.

聚辰科技(CAX-CLOUD)
info@cax-cloud.com

Obtaining NICE EnginFrame

If you have not already received your NICE EnginFrame package from NICE or your EnginFrame reseller, you can download it from EnginFrame Web site.

Downloading EnginFrame

EnginFrame packages can be downloaded from:

<https://www.nice-software.com/download/enginframe>

You need a valid account to access the download area. If you do not have one yet, please contact [<helpdesk@nice-software.com>](mailto:helpdesk@nice-software.com) or your EnginFrame reseller.

Obtaining a License

You need a valid license to install and run EnginFrame. If you do not have one yet, please contact [<helpdesk@nice-software.com>](mailto:helpdesk@nice-software.com) or your EnginFrame reseller.

EnginFrame licenses are classified as:

- *Demo licenses* - demo licenses are usually not bound to any IP address and are valid for one month.
- *Full licenses* - full licenses have time-unlimited validity and are bound to one or more IP addresses.
- *Year licenses* - year licenses have time-limited validity and are bound to one or more IP addresses.

You have to contact [<helpdesk@nice-software.com>](mailto:helpdesk@nice-software.com) or your EnginFrame reseller to purchase, renew, or update a license, perform a license change or obtain a demo license.

Licensed Plug-ins

Some EnginFrame plug-ins require a specific license to work. The standard plug-ins included in EnginFrame installation which require a license are:

Plug-ins requiring a specific license

interactive

Enables basic functionalities for Interactive Session management

applications

Enables the Applications portal

neutro

Enables the NEUTRO grid functionalities

hpc-support

Enables the HPC functionalities

Additional plug-ins provided by NICE may require a license as well.

The license file provided by your sales contact should contain license components for all the plug-ins included in your EnginFrame bundle.

Please check with your NICE sales contact or with our support helpdesk@nice-software.com for details.

If you have a specific license file, it must be copied under `$EF_TOP/license` folder, and have an `.ef` extension.

It will be automatically read by the portal. There's no need to restart EnginFrame.



Note

Remove from `$EF_TOP/license` folder any older `.ef` license files containing expired or invalid licenses, since EnginFrame will not accept any license conflict.

Here's an example license for component *Interactive*:

```
<?xml version="1.0"?>
<ef-licenses>
<ef-license-group product="EnginFrame PRO" format="1.0" release="2014.0">
  <ef-license
    component="interactive"
    vendor="NICE"
    expiration="2014-12-31"
    ip="10.20.10.14"
    licensee="Acme.com"
    type="DEMO"
    units="20"
    signature="xxxxxx"
  />
</ef-license-group>
</ef-licenses>
```

Refer to [Chapter 10, EnginFrame Licenses](#) for detailed information about EnginFrame licenses.

Planning NICE EnginFrame Deployment

Setting up EnginFrame is a straightforward process. However it is very important to accurately plan your EnginFrame Portal deployment to achieve a seamless integration with your computing environment and to satisfy your organization's IT requirements.

Prerequisites

Your system has to satisfy the following requirements before deploying EnginFrame Portal.

System Requirements

NICE EnginFrame supports the following operating systems¹:

- Amazon™ Linux® release 2016.03 or above
- Red Hat® Enterprise Linux® 5.x, 6.x, 7.x (x86-64)
- SUSE® Linux® Enterprise Server 11 SP2, 12 SP3 (x86-64)

The installation machine must have at least 3 GB of RAM and one or more IP addresses (at least one of them reachable by each of the potential client machines, directly or via proxies).

To install EnginFrame you need at least 200 MB of free disk space, but 2 GB or more are suggested since, while operating, the software saves important data and logging information.

Please, make sure you have enough space for the service data stored inside the EnginFrame spoolers. By default, spoolers are located inside the EnginFrame installation directory (\$EF_TOP/spoolers).

Third-party Software Prerequisites

Besides the standard packages installed with your operating system, NICE EnginFrame requires some additional third-party software.

¹ Other Linux® distributions and compatible Java™ versions might work but are not officially supported. Contact [<helpdesk@nice-software.com>](mailto:helpdesk@nice-software.com) for more information.

Java™ Platform

NICE EnginFrame requires the *Linux® x64* version of *Oracle® Java™ Platform Standard Edition* (Java™ SE 7 or Java™ SE 8) or *OpenJDK Runtime Environment 7 or 8*.

From now on, we will call `JAVA_HOME` the Java™ installation directory.



Java™ and Security

NICE suggests you to use the latest version of Oracle® Java™ SE 8 or OpenJDK 8 since they contain important enhancements to improve security of your Java™ applications.

The same Java™ version *must* be used for both EnginFrame Server and EnginFrame Agent.

Database Management Systems

Since version 2013.0, EnginFrame requires a *JDBC-compliant* database. EnginFrame uses the RDBMS to manage *Triggers*, *Job-Cache* and *Applications* and *Views* users' groups. EnginFrame *Triggers* rely on Quartz² engine to schedule the execution of EnginFrame services. Triggers are used internally to execute periodic tasks as to check and update Interactive sessions status and to collect EnginFrame usage statistics informations. The *Job-Cache* feature is responsible for collecting and caching job statuses over time.

By default Apache Derby® 10.11 database is installed together with EnginFrame Professional, however using Apache Derby® in a production installation is not recommended.

Apache Derby® is not supported for EnginFrame Enterprise installations, it is strongly suggested to use an external JDBC-compliant RDBMS. Since EnginFrame Enterprise is part of a HA solution, also the RDBMS must have its own HA strategy. The external RDBMS is suggested to reside on a different node(s) than the EnginFrame servers and possibly configured to be fault tolerant.

Please refer to the following table to select the database which fits your needs.

Table 3.1. Supported Database Management Systems

Name	Version	Notes
Apache Derby®	10.11	<p>Included in the EnginFrame Professional edition. Can be used for small EnginFrame Professional installations. Not suggested for medium and large production installations.</p> <p>It is not supported by EnginFrame Enterprise edition.</p>
SQL Server®	2008 R2, 2012	<p>Requires installation of the JDBC driver.</p> <p>Microsoft® JDBC official driver can be downloaded from http://www.microsoft.com/en-us/download/details.aspx?id=11774.</p>

²<http://www.quartz-scheduler.org>

Name	Version	Notes
Oracle® Database	Enterprise Edition 11g Release 2	Requires installation of the JDBC driver Oracle® JDBC official driver can be downloaded from http://www.oracle.com/technetwork/database/features/jdbc .
MySQL® Database	5.5 - 5.7	Requires installation of the JDBC driver MySQL® JDBC official driver can be downloaded from http://dev.mysql.com/downloads/connector/j .
MariaDB®	5.5	Requires installation of the MySQL® JDBC driver MySQL® JDBC official driver can be downloaded from http://dev.mysql.com/downloads/connector/j .

EnginFrame provides the JDBC driver for Apache Derby® only. In case a different DBMS is used, the JDBC driver must be added after the installation to the `$EF_TOP/<VERSION>/enginframe/WEBAPP/WEB-INF/lib` directory.

Please refer to the DBMS documentation for instructions on how to get the proper JDBC driver and configure it.

Authentication Mechanisms

EnginFrame supports different authentication mechanisms. Some of them require third-party software components.

Refer to Table 3.2, “Supported Authentication Mechanisms” to select the most appropriate authentication method for your system and check its third-party software prerequisites (if any).

Table 3.2. Supported Authentication Mechanisms

Name	Prerequisites	Notes
PAM	Linux® PAM must be correctly configured	It is the most common authentication method. It allows a system administrator to add new authentication methods simply by installing new PAM modules, and to modify authentication policies by editing configuration files. At installation time, you will be asked to specify which PAM service to use, system-auth is the default.
LDAP	The ldapsrch command must be installed and working appropriately on the EnginFrame Agent host	These methods allow you to authenticate users against a LDAP or Active Directory server. The EnginFrame installer will ask you to specify the parameters needed by ldapsrch to contact and query your directory server.
Active Directory		

Name	Prerequisites	Notes
HTTP Authentication	External HTTP authentication system	This method relies on an external authentication system to authenticate the users. The external system then adds an HTTP authentication header to the user requests. EnginFrame will trust the HTTP authentication header.
Certificate	SSL Certificates need to be installed and exchanged between EnginFrame Server and clients.	This method relies on the authentication accomplished by the web server, which requires the client authentication through the use of SSL certificates.

The EnginFrame installer can optionally verify if you have correctly configured the selected authentication method.

NICE EnginFrame can be easily extended to add support for custom authentication mechanisms.

Distributed Resource Managers

EnginFrame supports different distributed resource managers (DRM).

At installation time, you will need to specify which DRMs you want to use and provide the information required by EnginFrame to contact them. A single EnginFrame instance can access more than one DRM at the same time.

Refer to Table 3.3, “Supported Distributed Resource Managers” for a list of supported DRMs.

Table 3.3. Supported Distributed Resource Managers

Name	Version	Notes
IBM® Platform™ LSF®	6.x - 10.x	The LSF/openlava client software must be installed on the EnginFrame Agent host. The installer will ask you to specify the LSF/openlava profile file.
OpenLava	2.x	
Adaptive Computing® Moab® Web Services (MWS)	7.2.x	The MWS server must be reachable from the EnginFrame Server host. The installer will ask you to specify the IP address of your MWS server.
Altair® Professional®	7.x - 14.x	The PBS Professional® client software must be installed on the EnginFrame Agent host. The installer will ask you to specify the directory where the PBS Professional® client software is installed.
Torque	3.x - 6.x	The Torque client software must be installed on the EnginFrame Agent host. The installer will ask you to specify the directory where the Torque client software is installed.

Name	Version	Notes
NICE Neutro	2013 or later	The NEUTRO master(s) must be reachable from the EnginFrame Server host. The installer will ask you to specify the IP address of your NEUTRO masters.
SLURM™	14.x - 17.x	SLURM™ binaries must be installed on the EnginFrame Server host. SLURM™ master host must be reachable from the EnginFrame Server host. The installer will ask you to specify the path where binaries are installed. On SLURM™ configuration, specifically related to compute nodes dedicated to interactive sessions, the Features: vnc,dcv,dcv2 and RealMemory parameters must be added to every required node. 'dcv2' stands for DCV since 2017.
Sun® Grid Engine (SGE)	6.2	The Grid Engine client software must be installed on the EnginFrame Agent host. The \$SGE_ROOT/\$SGE_CELL/common must be shared from SGE master to EF nodes. The installer will ask you to specify the Grid Engine shell settings file.
Oracle® Grid Engine (OGE)	7.0	
Univa® Grid Engine® (UGE)	8.x	
Son of Grid Engine (SoGE)	8.1.x	
Open Grid Scheduler	2011.x	
AWS Batch	The AWS Batch cluster must be created with AWS ParallelCluster 2.1.0 or later	The installer will ask you to specify the AWS ParallelCluster cluster name and the AWS region.

Some schedulers like Torque, PBS Professional® and Univa® Grid Engine® (UGE) 8.2.0 have job history disabled by default. This means that a job will disappear when finished. It is strongly suggested to configure these distributed resource managers to retain information about the finished jobs. For more information on the configuration check [the section called “Required DRM Configuration”](#).

Support for additional resource managers is available via optional plugins. Contact [<helpdesk@nice-software.com>](mailto:helpdesk@nice-software.com) for more information.

Required DRM Configuration

Altair® PBS Professional®

Applies to versions: 11, 12, 14

Altair® PBS Professional® by default does not show finished jobs. To enable job history, a server parameter must be changed:

```
qmgr -c "set server job_history_enable = True"
```

Once enabled, the default duration of the job history is 2 weeks.

Torque

Applies to versions: 4, 5, 6

Torque by default does not show finished jobs. To enable job history, a queue parameter must be changed:

```
qmgr -c "set queue batch keep_completed=120"
```

The `keep_completed` parameter specifies the number of seconds jobs should be held in the Completed state after exiting.

Once enabled, the default duration of the job history is 2 minutes.

Applies to versions: all

In order to get the *Display Output* functionality for *Torque* jobs, *qpeek* tool should be configured properly. By default *qpeek* uses the *rsh* command to remote access the so-called "mother superior" node.

Choose between installing *rsh* on the nodes or configure *qpeek* to use *ssh* and install *ssh* configured passwordless among the nodes.

Univa® Grid Engine® (UGE)

Applies to versions: 8.2.x

Univa® Grid Engine® (UGE) by default does not show finished jobs. To enable job history:

- (8.2.0 only) disable reader threads:

```
edit file SGE_ROOT/SGE_CELL/common/bootstrap
```

```
set reader_threads to 0 instead of 2
```

- enable finished jobs:

```
run
```

```
qconf -mconf
```

set `finished_jobs` to a non-zero value according to the rate of finishing jobs.

The `finished_jobs` parameter defines the number of finished jobs stored. If this maximum number is reached, the eldest finished job will be discarded for every new job added to the finished job list.

By default EnginFrame grabs the scheduler jobs every minute. The `finished_jobs` parameter must be tweaked so that a finished job stays in the job list for at least a minute. Depending on the number of jobs running in the cluster a reasonable value is in between *the medium number of running jobs* and *the amount of jobs ending per minute*.

- restart qmaster

SLURM™

Applies to versions: all

SLURM™ show finished jobs for a default period defined by the `MinJobAge` parameter in file `slurm.conf` (under `/etc/slurm` or the SLURM™ configuration directory). The default value is 300 seconds, i.e. five minutes, which is acceptable.

In case you changed this parameter, ensure it is not set to a value lower than 300.

Also check the `MaxJobCount` parameter is not set.

After changing this parameter restart SLURM™ with:

```
/etc/init.d/slurm stop
/etc/init.d/slurm start
```

The setting must be done on all SLURM™ nodes.

IBM® Platform™ LSF® / OpenLava

Applies to versions: all

IBM® Platform™ LSF® and OpenLava show finished jobs for a default period defined by the `CLEAN_PERIOD` parameter in file `lsb.params`. The default value is 3600 seconds, i.e. one hour, which is acceptable.

In case you changed this parameter, ensure it is not set to a value lower than 300.

After changing this parameter run:

```
badmin reconfig
```

AWS Batch

To integrate EnginFrame with AWS Batch it is required to create a Batch cluster with AWS ParallelCluster and give the user running the EnginFrame Server the permission to interact with the cluster. Here the details of the required steps:

- Install AWS ParallelCluster and configure it following the instruction [here](#).
AWS CLI will be installed as dependency of AWS ParallelCluster.
- Taking into account the [network requirements](#), create a new cluster for AWS Batch scheduler.
- Go to the CloudFormation console from the AWS Account and click on the created Stack.
Get the Stack ID from the Stack Info tab view. (e.g. `arn:aws:cloudformation:<REGION>:<ACCOUNT>:stack/<STACK_NAME>/<UID>`)
- Get the `BatchUserRole` from the Outputs tab view, using the AWS CloudFormation console, or through the `status` command of the AWS ParallelCluster command line. (e.g. `arn:aws:iam::<ACCOUNT>:role/<STACK_NAME>-suffix`).
- Use the `BatchUserRole` and the Stack ID (by replacing the latest UID with an asterisk) to create, through the Identity and Access Management (IAM) console, a new IAM Policy like the following:


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole",
        "cloudformation:DescribeStacks"
      ],
      "Resource": [
        "<BatchUserRole>",
        "<StackID with the UID replaced by *>"
      ]
    }
  ]
}
```

- Create a new IAM User and assign the created Policy to it.
- From the IAM Console, click on the created user and get the user credentials from the Security Credentials tab view. Use them to configure the AWS CLI for the user running the EnginFrame Server (e.g. efnobody).

```
[efnobody]$ aws configure
```

- Follow EnginFrame installer steps to configure AWS Batch EnginFrame plugin to contact the created cluster.

Session Brokers

Starting from version 2017.0, EnginFrame supports Citrix® XenDesktop® as Session Broker.

At installation time you can choose to use XenDesktop® as session broker and provide the configuration parameters required by EnginFrame to contact the XenDesktop® Server.

Table 3.4. Supported Session Brokers

Name	Version	Notes
XenDesktop®	7.6	see the section called “Required Session Brokers Configuration”

For detailed instructions on how to install and configure the session broker please refer to its manuals.

Contact <helpdesk@nice-software.com> for more information.

Required Session Brokers Configuration

XenDesktop®

Applies to versions: 7.6

EnginFrame plugin for XenDesktop® requires NICE Neutro to submit delegate sessions (see the section called “Distributed Resource Managers” for more info about NICE Neutro).

- Neutro Agent must run as XenDesktop® administrator user on the same host where the XenDesktop® Delivery Controller is running.
- The host where XenDesktop® Delivery Controller is running must be tagged as XenDesktopController in the Neutro Master configuration file `$NEUTRO_ROOT/conf/hosttags.conf`.
- XenDesktop® tasks must be added into the NICE Neutro Master installation tree.
Copy `task-interactive-xendesktop.xml` file into the Neutro Master task repository and set right permissions:

```
cp $EF_ROOT/plugins/xendesktop/etc/neutro_tasks/task-repository/
task-interactive-xendesktop.xml \
$NEUTRO_ROOT/task-repository/task-interactive-xendesktop.xml

chown <neutroadmin>:root \
$NEUTRO_ROOT/task-repository/task-interactive-xendesktop.xml
```

- Create the directory to store the files required by XenDesktop® task:

```
mkdir $NEUTRO_ROOT/task-repository/task-interactive-xendesktop
```

- Copy XenDesktop® task files and set right permissions:

```
cp $EF_ROOT/plugins/xendesktop/etc/neutro_tasks/task-repository/
task-interactive-xendesktop/* \
$NEUTRO_ROOT/task-repository/task-interactive-xendesktop

chown -R <neutroadmin>:root \
$NEUTRO_ROOT/task-repository/task-interactive-xendesktop
```

- XenDesktop® Web Interface library must be added to the EnginFrame XenDesktop® plugin installation tree.

Download the Web Interface 5.4 for Java Application Servers library from [Citrix® Website](#) (registration or login is required).

- Uncompress the `WebInterface.jar` and look for the `PNAgent.war` on it.

```
jar -xvf WebInterface.jar
```

- Uncompress `PNAgent.war` file, copy jar files from `PNAgent.war` archive into XenDesktop® plugin jars folder and set right permissions:

```
jar -xvf PNAgent.war
```

```
cp <uncompressed PNAgent folder>/WEB-INF/lib/*.jar \
$EF_ROOT/plugins/xendesktop/lib/jars/
```

```
chmod +r $EF_ROOT/plugins/xendesktop/lib/jars/*
```

- EnginFrame users must be in the same Active Directory domain used by XenDesktop®. EnginFrame uses users provided password to log in to XenDesktop®.
- Clients require Citrix Receiver™ application to connect to XenDesktop® sessions.

Remote Visualization Technologies

EnginFrame supports different remote visualization technologies, and the same EnginFrame instance can manage multiple of them. Please refer to the following table for the supported ones.

Table 3.5. Supported Remote Visualization Technologies

Name	Version	Notes
RealVNC® Enterprise Edition	4.x or 5.x	It allows to share sessions both in full access or view only mode.
TigerVNC	1.x	Linux® only (server side).
TurboVNC	1.x or 2.x	Linux® only (server side).
RealVNC® Free Edition	4.x	Linux® only (server side).
NICE DCV	2012.0 or later	It allows to share sessions both in full access or view only mode.
VirtualGL	2.1 or later	
HP® RGS	5.x, 6.x or 7.x	Please refer to EnginFrame Administrator's Guide for more information on how to set up your DRM system on Linux® nodes to work with HP® RGS.
Citrix Receiver™	4.5	Windows® only (server side).

For detailed instructions on how to install and configure these remote visualization technologies please refer to their respective manuals.

Support for additional remote visualization technologies is available via optional plugins. Contact [<helpdesk@nice-software.com>](mailto:helpdesk@nice-software.com) for more information.

Remote Visualization Technologies Configuration

NICE DCV 2017.0 or later on Linux

For Linux environments the configuration of the authentication to use with NICE DCV must correspond to the authentication system set on the DCV server in the remote visualization hosts.

On EnginFrame the authentication to use with DCV on Linux can be set in the `INTERACTIVE_DEFAULT_DCV2_LINUX_AUTH` configuration parameter inside the `$EF_TOP/conf/plugins/interactive/interactive.efconf` file.

Default value and documentation can be found in the static configuration file `$EF_TOP/<VERSION>/enginframe/plugins/interactive/conf/interactive.efconf`.

The auto authentication system, providing seamless authentication with self-generated strong passwords, requires the following configuration on the visualization hosts running the DCV server:

- The DCV simple external authenticator provided with NICE DCV must be installed and running.

The simple external authenticator installation package is distributed as an rpm, e.g. `nice-dcv-simple-external-authenticator-2017.x...x86_64.rpm`.

Once installed you can manage the service as root user:

- On systems using SystemD (e.g. RedHat 7):

```
systemctl [start|stop|status] dcvsimpleextauth
```

- On systems using SysVInit (e.g. RedHat 6):

```
/etc/init.d/dcvsimpleextauth [start|stop|status]
```

- The DCV server must be configured to use the simple external authenticator `dcvsimpleextauth` instance running on the same host, e.g. inside `/etc/dcv/dcv.conf`, under the `security` section, there should be a setting like this:

```
[security]
auth-token-verifier="http://localhost:8444"
```

- Restart the DCV server after any changes made to `/etc/dcv/dcv.conf` configuration file.

NICE DCV 2017.0 or later on Windows

For Windows environments the configuration of the authentication to use with NICE DCV must be configured on EnginFrame in the `INTERACTIVE_DEFAULT_DCV2_WINDOWS_AUTH` configuration parameter inside the `$EF_TOP/conf/plugins/interactive/interactive.efconf` file.

Default value and documentation can be found in the static configuration file `$EF_TOP/<VERSION>/enginframe/plugins/interactive/conf/interactive.efconf`.

The auto authentication system, providing seamless authentication with self-generated strong passwords, does not require any other configuration on the visualization hosts running the DCV server.

The DCV server service is managed by the interactive session job landing on the node:

- If the DCV server service is not running, it will be started.
- If the DCV server service is running but with different authentication configuration than the one set on the EnginFrame side, the configuration will be changed and the service restarted. This includes the case when the DCV server is configured to automatically launch the console session at system startup. This setting will be removed by the interactive session job.

- If DCV session is running but there is no logged user, the session will be closed by the interactive session job.
- It is not possible to submit an interactive session to a node with a DCV session running and a user logged in.

Network Requirements

EnginFrame is a distributed system. Your network and firewall configuration must allow EnginFrame components to communicate with each other and with user's browsers.

The specific requirements depend on how EnginFrame is deployed on your system. Please refer to *EnginFrame Administrator's Guide* for more detailed information. The following table summarizes network requirements for a basic EnginFrame deployment.

Table 3.6. Network Requirements

Port (Default)	Protocol	From Host	To Host	Mandatory
8080/8443	HTTP/HTTPS	User's clients	EnginFrame Server	Mandatory
9999 and 9998	RMI (TCP)	EnginFrame Server	EnginFrame Agent	Optional ¹
8080/8443	HTTP/HTTPS	EnginFrame Agent	EnginFrame Server	Optional ¹
7800	TCP	EnginFrame Server	EnginFrame Server	Mandatory only for EnginFrame Enterprise ²

¹Required if EnginFrame Agent and EnginFrame Server run on separate hosts

²EnginFrame Servers use the port to communicate with each other

Supported Browsers

NICE EnginFrame produces HTML which can be viewed with most popular browsers. NICE EnginFrame has been tested with the browsers listed in Table 3.7, "Supported Browsers".

Table 3.7. Supported Browsers

Name	Version	Notes
Microsoft® Internet Explorer®	10 and 11	
Mozilla Firefox®	3.6 and above	
Apple® Safari®	6.0 and above and iOS 6 version	Tested on Mac® OS X® and iPad® only.
Google™ Chrome™	25 and above	

JavaScript® and Cookies must be enabled on browsers.

Interactive Plugin Requirements

Interactive Plugin requires the following components to be successfully installed and configured:

- at least one supported resource manager software, see [the section called “Distributed Resource Managers”](#) or a session broker software, see [the section called “Session Brokers”](#)
- at least one supported remote visualization middleware, see [the section called “Remote Visualization Technologies”](#)

To use the Interactive Plugin, a proper license must be installed on the EnginFrame Server.

Each node running interactive sessions should have all the necessary software installed. On Linux® this usually means the packages for the desired desktop environment (gnome, kde, xfce, etc).

In addition, to let the portal show screen thumbnails in the session list, the following software must be installed and available in the system PATH on visualization nodes:

- Linux®: *ImageMagick tool* (<http://www.imagemagick.org>) and the `xorg-x11-apps`, `xorg-x11-utils` packages
- Windows®: *NICE Shot tool* (`niceshot.exe` available under `$EF_TOP/<VERSION>/enginframe/plugins/interactive/tools/niceshot`). Not required on NICE Neutro hosts since Neutro Agent installer already includes it.

Single Application Desktop Requirements (Linux®)

Sometimes you may prefer to run a minimal session on your interactive nodes consisting in a minimal desktop and a single application running. In that case, instead of installing a full desktop environment like GNOME or KDE, you may want to only install some basic required tools, a Window manager, a dock panel and the applications you intend to use.

For this intent the `minimal.xstartup` script can be configured to be a Window Manager choice for the Applications and Views service editors.

Here is a reference list of the tools used by the `minimal.xstartup` file provided by EnginFrame under `$EF_TOP/<VERSION>/enginframe/plugins/interactive/conf`:

- basic tools: `bash`, `grep`, `cat`, `printf`, `gawk`, `xprop`
- window managers: `metacity`, `kwin` (usually provided by package `kdebase`), `xfwm4`
- dock panels: `tint2`, `fluxbox`, `blackbox`, `mwm` (usually provided by package `openmotif` or `lesstif` or `motif`)

Shared File System Requirements

Depending on the deployment strategy, EnginFrame may require some directories to be shared between the cluster and EnginFrame nodes. This guide covers the simplest scenario where both EnginFrame Server and EnginFrame Agent run on the same host. For more complex configurations or to change the mount points of the shared directories, please check the *"Deployment Strategies"* section in the EnginFrame Administrator's Guide.

In this scenario the EnginFrame Server, EnginFrame Agent and visualization nodes may require the `$EF_TOP/sessions` directory to be shared. Please refer to the following table to check if you need to share this directory or not.

Table 3.8. Shared File-System Requirement

Distributed Resource Manager	Linux®	Windows®
NICE Neutro	-	Not required
IBM® Platform™ LSF®	Not required	Not required
OpenLava	Not required	-
SLURM™	Required	-
Adaptive Computing® Moab® Web Services (MWS)	Required	-
Torque	Required	-
Altair® PBS Professional®	Required	-
Grid Engine (SGE, SoGE, OGE, UGE)	Required	-

EnginFrame Enterprise System Requirements

This documents lists the hardware and software prerequisites for an EnginFrame Enterprise installation.

Shared File System

As explained in the section called “Architecture” the suggested and supported approach to EnginFrame Enterprise deployment involves a shared file-system for the whole `$EF_TOP` directory tree. With this approach you are able to install and manage the software from just one node and all the binaries and data directories as spoolers, sessions, license, etc. will be shared among the installation nodes. High-Availability of the Shared File System shall be consistent with the overall HA/Disaster Recovery strategy.

The NFS `no_root_squash` or equivalent feature must be active in order to allow the correct management of permissions and ownership of deployed files.

It is strongly suggested to enable on the shared file system the NFS `no_wdelay` or equivalent feature (server-side) in order to minimize the file writing delay between clients.

Network Load Balancing

In order to ensure automated load balancing and HA of the EnginFrame services it is necessary to setup a network load balancer that dispatches users' requests to the EnginFrame Servers in a balanced way.

EnginFrame requires the load balancer to implement a *sticky session* strategy. There are many open source and commercial solutions to implement a network load balancer.

Please refer to the section called “Load Balancer Setup” for examples of configuration with Apache® front-end.

Deployment Strategies

If the prerequisites are met, you must decide how to deploy NICE EnginFrame on your system.

As described in [the section called “Architectural Overview”](#) EnginFrame is made of two main software components: the EnginFrame Server and the EnginFrame Agent.

These two components can be deployed on the same host or on different hosts that communicate across the network. The choice depends on your computational resources organization, on your network architecture, and on your security and performance requirements, as long as these constraints are met:

- EnginFrame Server host must be reachable via HTTP(S) by the clients and the EnginFrame Agents.
- EnginFrame Agent host must have access to your computational resources and your grid infrastructure (e.g. submitting jobs to your scheduler).
- EnginFrame Server and EnginFrame Agent must be installed on a shared storage area.
- For the interactive sessions, EnginFrame Server and EnginFrame Agent must have read/write access to a storage area shared among them and with the visualization nodes too.

In the simplest scenario both EnginFrame Server and EnginFrame Agent run on the same host, which ensures that communication between the two is reliable and minimizes administration efforts. This scenario is also suggested for an EnginFrame Enterprise deployment where you have multiple EnginFrame installations on different hosts but on each of the hosts you install both an EnginFrame Server and Agent, as described in [the section called “Deployment”](#). Network communication among EnginFrame Servers must also be assured.

In other cases you may want to install EnginFrame Server and EnginFrame Agent on separate hosts. For instance you want to run the EnginFrame Server in DMZ and EnginFrame Agent on the head node of your cluster. In these cases you must consider some extra requirements:

- Agent and Server must be able to communicate through a TCP connection using the [Java™ RMI protocol](#). The relevant TCP ports that are 9999 for RMI Registry and 9998 for Remote Object, must be free on the Agent's host and reachable from the Server host.
- Agent and Server must be able to communicate through a TCP connection using the HTTP(S) protocol. The relevant TCP port (which by default is 8080 / 8443 for HTTP/HTTPS respectively) on the Server's host must be reachable from the Agent host.
- Agent and Server must be installed on a shared storage area. In particular:
 - the spoolers directory must reside on a storage area satisfying the requirements described in [the section called “Spoolers Requirements”](#)
 - the sessions directory must reside on a storage area satisfying the requirements described in [the section called “Sessions Requirements”](#)

In a scenario where you need to access the system through a DMZ, it's a common practice to use an Apache® Web Server as front-end in the DMZ while EnginFrame is deployed on the intranet. This scenario comes even more natural with EnginFrame Enterprise if you're going to use Apache® Server as network load balancer in front of a battery of EnginFrame Servers. In all these cases the EnginFrame Server is behind the Apache® Web Server that forwards all the EnginFrame requests to the Tomcat® servlet container of the EnginFrame deployment. Refer to [the section called “Apache®-Tomcat® Connection”](#) for details on this configuration.

About security in the communications, EnginFrame by default uses Java RMI over SSL protocol between Server and Agent and allows to setup Tomcat® at installation time to use HTTPS. It's also possible to enable HTTPS protocol at a later time as described in [Chapter 13, Configuring HTTPS](#).

Installation Directories

The next thing to consider are the directories where NICE EnginFrame is deployed.

The *installation directory* is the location, hereafter referred to as \$NICE_ROOT, where EnginFrame binaries, configuration files, and logs are placed.

The installation creates under \$NICE_ROOT the following directory structure:

```
NICE_ROOT
|-- enginframe
|   |-- 2019.0-rXXXXX
|   |   |-- enginframe
|   |-- current-version
|   |-- bin
|   |   |-- enginframe
|   |-- install
|   |   |-- 2019.0-rXXXXX
|   |-- license
|   |   |-- license.ef
|   |-- conf
|   |   |-- enginframe.conf
|   |   |-- enginframe
|   |   |   |-- certs
|   |   |   |-- server.conf
|   |   |   |-- agent.conf
|   |   |-- tomcat
|   |   |   |-- conf
|   |   |   |   |-- certs
|   |   |-- derby
|   |   |   |-- derby.properties
|   |   |   |-- server.policy
|   |   |-- plugins
|   |-- data
|   |   |-- cache
|   |   |-- derby
|   |   |   |-- EnginFrameDB
|   |   |-- plugins
|   |-- logs
|   |   |-- <HOSTNAME>
|   |   |   |-- *.log
|   |   |   |-- tomcat
|   |   |   |-- derby
|   |-- repository
|   |-- sessions
|   |-- spoolers
|   |-- temp
|   |   |-- <HOSTNAME>
|   |   |   |-- dumps
|   |   |   |-- errors
|   |   |   |-- tomcat
```

The following names will be used in this guide to refer to the different parts of the EnginFrame installation tree:

NICE_ROOT

The directory containing NICE products, the default is `/opt/nice`

EF_TOP

The directory containing the EnginFrame product, the default is `NICE_ROOT/enginframe`

EF_LICENSE_PATH

The directory containing the EnginFrame license files, the default is `EF_TOP/license`

EF_CONF_ROOT

The directory containing the EnginFrame configuration files, the default is `EF_TOP/conf`

EF_DATA_ROOT

The directory containing the data files, the default is `EF_TOP/data`

EF_LOGS_ROOT

The directory containing the log files, the default is `EF_TOP/logs`

EF_TEMP_ROOT

The directory containing the temporary files, the default is `EF_TOP/tmp`

EF_REPOSITORYDIR

The directory containing the EnginFrame repository files, the default is `EF_TOP/repository`

EF_SPOOLERDIR

The directory containing the EnginFrame spoolers, the default is `EF_TOP/spoolers`

INTERACTIVE_SHARED_ROOT

The directory containing the EnginFrame interactive sessions, the default is `EF_TOP/sessions`

EF_ROOT

The directory containing the EnginFrame binaries and system files, the default is `EF_TOP/<VERSION>/enginframe`

**suid binaries**

PAM based authentication method shipped with EnginFrame requires that some binaries have the `suid` bit set in order to interact with the underlying system to authenticate users.

If you plan to use this authentication method, ensure the file system hosting EnginFrame is mounted with `nosuid` flag unset.

The *EF_SPOOLERDIR* directory is used to hold all the data supplied as input and created as output by EnginFrame services.

As already mentioned, the spooler directory *must* be accessible by both the EnginFrame Server and EnginFrame Agent and *must* be readable and writable by unprivileged users (see below) and by `root`.

By default, the spooler directory is placed in a sub-directory of the installation directory.

Chapter 7, *Managing Spoolers* contains a detailed description of the [system requirements](#) for the spoolers directory.

Special Users

You have to choose which *system accounts* EnginFrame has to use.

The EnginFrame Administrator is a special system account owning some privileges, for instance the possibility to access the EnginFrame monitor/administration portal and some of the configuration files. From now on, this account is referred as EF_ADMIN.

You have also to choose the account that should run Tomcat®, from now on referred as EF_NOBODY in the document.

The configuration files with sensitive information that are meant to be read only by EnginFrame Server are owned by EF_NOBODY with no allowed permissions for the group and other users.

Restricted ownership and permissions apply also to the \$EF_TOP/logs directory since logs files may also contain sensitive information.

Any existing system account *excluding root* can be specified, however it is good practice to set up two new dedicated users for these roles.

Conventionally eadmin and enobody are respectively used for EF_ADMIN and EF_NOBODY.



EnginFrame Accounts

EF_ADMIN and EF_NOBODY must be operating system valid accounts: you must be able to login to the system with those accounts and they *must not* be disabled.

Authentication

The last aspect you need to consider before installing EnginFrame is which authentication method to use.

EnginFrame is able to authenticate users using many different mechanisms, among which PAM, LDAP, ActiveDirectory, HTTP Basic Authentication and Certificate.

You can also write your own authentication module if the ones shipped by EnginFrame do not suit your needs.

Chapter 11, *Authentication Framework* contains a detailed information about EnginFrame Authentication System.

DRM Configuration for Interactive Plugin

The following sections describe the additional requirements of the Interactive Portal.



Important

EnginFrame periodically checks the status of the interactive jobs using the EF_ADMIN user account.

Therefore this account *MUST* be able to get information from all the DRMs.

Usually the simplest way to achieve it is by making this account one of the resource manager administrators.

A misconfiguration of this account can lead to interactive session data loss.



Note

When the resource manager controls mixed Windows® clusters, Interactive Plugin will submit interactive session jobs on Windows® hosts as the user running EnginFrame Server (efnobody by default). Consequently this *MUST* be a valid Windows® user.

NICE Neutro

Interactive Plugin relies on the NICE Neutro workload manager, to allocate and reserve resources to run the interactive sessions. The integration does not require any extra setting. For details about NICE Neutro, please refer to NICE Neutro installation guide.

IBM® Platform™ LSF® or OpenLava

Interactive Plugin relies on the LSF® workload manager, to allocate and reserve resources to run the interactive sessions. Installation and configuration instructions for LSF® are out of the scope of this document, however Interactive Plugin requires some specific LSF® settings.

Here is a minimal configuration necessary to run Interactive Plugin sessions on your LSF® cluster. If needed they can be enhanced or combined with your existing LSF® configuration to achieve more complex resource sharing policies.

Configuring Queues

Interactive Plugin uses resource manager's queues to submit and manage interactive sessions. You can set up Interactive Plugin to use a default queue and set different services to use different queues, it is however important that the queues used for any visualization middleware or target system have the following settings:

- The EnginFrame Administrator account (usually eadmin) must be queue administrator of any queue used by Interactive Plugin.
- Queues for HP® RGS sessions need to have HJOB_LIMIT set to one, since only one HP® RGS session can run on each host.
- Queues for HP® RGS Linux® sessions need to have PRE_EXEC and POST_EXEC respectively set to the rgs.preexec.sh and rgs.postexec.sh scripts, located under \$EF_TOP/<VERSION>/enginframe/plugins/interactive/tools folder.

Here is a configuration snippet for these queues in lsb . queues. You might not need to have all of these queues configured. You can adapt the parameters as you wish, given the above requirements.

```
Begin Queue
QUEUE_NAME      = int_linux
PRIORITY        = 50
EXCLUSIVE       = y
NEW_JOB_SCHED_DELAY = 0
JOB_ACCEPT_INTERVAL = 0
ADMINISTRATORS  = efaadmin
HOSTS           = viz1 vizlin01 vizlin02
DESCRIPTION = Queue for linux interactive applications
End Queue

Begin Queue
QUEUE_NAME      = int_windows
PRIORITY        = 50
EXCLUSIVE       = y
NEW_JOB_SCHED_DELAY = 0
JOB_ACCEPT_INTERVAL = 0
ADMINISTRATORS  = efnobody
HOSTS           = win_grp
DESCRIPTION = Queue for windows interactive applications
End Queue

Begin Queue
QUEUE_NAME      = rgs_linux
PRIORITY        = 50
EXCLUSIVE       = y
NEW_JOB_SCHED_DELAY = 0
JOB_ACCEPT_INTERVAL = 0
HJOB_LIMIT      = 1
ADMINISTRATORS  = efaadmin
HOSTS           = viz2 vizlin03 vizlin04
PRE_EXEC        = /opt/nice/enginframe/plugins/interactive/tools/rgs.preexec.sh
POST_EXEC       = /opt/nice/enginframe/plugins/interactive/tools/rgs.postexec.sh
DESCRIPTION = Queue for RGS linux sessions
End Queue

Begin Queue
QUEUE_NAME      = rgs_windows
PRIORITY        = 50
EXCLUSIVE       = y
NEW_JOB_SCHED_DELAY = 0
JOB_ACCEPT_INTERVAL = 0
HJOB_LIMIT      = 1
ADMINISTRATORS  = efnobody
DESCRIPTION = Queue for RGS windows sessions
End Queue
```

Finally the pre and post execution scripts for HP® RGS Linux® sessions need to run as root. This means that the `/etc/lsf.sudoers` file on all the LSF® nodes must contain the following line:

```
LSB_PRE_POST_EXEC_USER=root
```



Note

Make sure `/etc/lsf.sudoers` is owned by root and has permissions 600 otherwise LSF® will ignore its contents.

After you modify `/etc/lsf.sudoers`, you must run

```
badadmin hrestart all
```

to restart `sbatchd` on all nodes in the cluster.



Note

To specify the default `rgs` queues inside interactive you can edit the `$EF_TOP/conf/plugins/interactive/interactive.efconf` file and add the following two lines.

```
INTERACTIVE_DEFAULT_RGS_LINUX_QUEUE=rgs_linux  
INTERACTIVE_DEFAULT_RGS_WINDOWS_QUEUE=rgs_windows
```



Important

By default, every pre/post exec script runs with the credentials of the owner of the job. Once this configuration is applied, all the pre/post execution scripts configured in LSF® at queue level (`lsb.queues`) or at application level (`lsb.applications`) will be executed with the root account. The impact on security and functionality must be analyzed case by case.

In alternative it is also possible to configure the **sudo** command to run pre and post execution scripts as a normal user with privileges to run as `root` only specific operations.

Requirements on scheduler tools

In order to properly operate with interactive sessions on the target operating systems, EnginFrame relies on some tools provided by LSF® and OpenLava schedulers. The scheduler must then be properly configured in order to make these tools working effectively on the hosts of the cluster.

Here the list of LSF® and OpenLava tools required by EnginFrame:

- Windows sessions scheduled with LSF® require `lsrun` and `lsrscp`.
- Linux® sessions via LSF® or OpenLava require `lsrun`.



Important

In the recent LSF® versions (i.e. from 9.1) the `lsrun` command comes disabled by default. This configuration can be changed by editing file `LSF_TOP/conf/lsf.conf` and setting:

```
LSF_DISABLE_LSRUN=N
```

After this change the LIM daemon must be asked to reload the configuration, as scheduler administrator running the following command:

```
lsadmin reconfig
```

Additional requirements for Windows® sessions

To be able to launch Windows® interactive sessions through LSF®, there are additional requirements for the user running EnginFrame server (default `efnobody`). In all Windows® hosts, this user must be able to:

- read and write Windows® registry keys.
- start and stop Windows® services.
- read and write under LSF top directory.

These privileges enable that user to start, stop and control remote visualization middlewares servers launched on the Windows® hosts.

These requirements do not imply that the user running EnginFrame server must be a Windows® administrator, even if they are automatically granted to a Windows® XP one.



Important

Under Windows® 7 or later, these requirements are *not* automatically granted to administrators. For this reason, Windows® 7 system administrators should either manually grant these privileges to that user only, or completely disable the User Account Control (UAC) for everybody.

OpenLava 3.0 or above

From version 3.0 OpenLava introduced `bpost` command that is not compatible with the same LSF command.

To use OpenLava 3.0 or above with EnginFrame please configure the system to use a shared file system for the `$INTERACTIVE_SHARED_ROOT` directory whose default value is `$EF_TOP/sessions`.

Then you need to edit the configuration file `$EF_TOP/conf/plugins/lsf/ef.lsf.conf` to apply the following setting:

```
LSF_INTERACTIVE_USE_SHARED_FS="true"
```

Alternatively, to avoid using a shared file system for the `sessions` directory, you need to remove or rename the `bpost` executable in the OpenLava binary directory.

Torque or PBS Professional®

Interactive Plugin relies on the PBS Professional® workload manager, to allocate and reserve resources to run the interactive sessions. Installation and configuration instructions for PBS

Professional® are out of the scope of this document, however Interactive Plugin requires some specific PBS Professional® settings.

Here is a minimal configuration needed to run Interactive Plugin sessions on your PBS Professional® cluster. If needed they can be enhanced or combined with your existing PBS Professional® configuration to achieve more complex resource sharing policies.

Configuring Queues

Interactive Plugin uses resource manager's queues to submit and manage interactive sessions. You can set up Interactive Plugin to use a default queue and set different services to use different queues, it is however important that the queues used for any visualization middleware or target system have the following settings:

- The EnginFrame Administrator account (usually `efadmin`) must be able to see, start and kill jobs of any queue used by Interactive Plugin.
- You need to force the limit of 1 job per host for HP® RGS queues, since only one HP® RGS session can run on each host.
- Queues for HP® RGS Linux® sessions need to have `prolog` and an `epilog` respectively set to the `rgs.preexec.sh` and `rgs.postexec.sh` scripts, located under `$EF_TOP/<VERSION>/enginframe/plugins/interactive/tools` folder. So you might want to copy or link them into `${PBS_HOME}/mom_priv` of each execution host.

Here is a sample configuration of the interactive queue

```
# qmgr
Max open servers: 49
Qmgr: create queue interactive
set queue interactive queue_type = Execution
set queue interactive resources_default.arch = linux
set queue interactive enabled = True
set queue interactive started = True
```

SGE, Oracle® Grid Engine (OGE), Son of Grid Engine (SoGE) or Univa® Grid Engine® (UGE)

Interactive Plugin relies on the SGE workload manager, to allocate and reserve resources to run the interactive sessions. Installation and configuration instructions for SGE are out of the scope of this document, however Interactive Plugin requires some specific SGE settings.

Here is a minimal configuration needed to run Interactive Plugin sessions on your SGE cluster. If needed they can be enhanced or combined with your existing SGE configuration to achieve more complex resource sharing policies.

Configuring Queues

Interactive Plugin uses resource manager's queues to submit and manage interactive sessions. You can set up Interactive Plugin to use a default queue and set different services to use different queues, it is however important that the queues used for any visualization middleware or target system have the following settings:

- The EnginFrame Administrator account (usually `efadmin`) must be queue administrator of any queue used by Interactive Plugin.
- To make available to Interactive Plugin scripts the necessary system command line tools and environment, SGE queues used also need the following requirement: either `shell_start_mode` queue parameter has to be set to `unix_behavior` or, if the same parameter is set to `posix_compliant`, then the `shell` parameter must be set to `/bin/bash`.
- You need to force the limit of 1 job per host for HP® RGS queues, since only one HP® RGS session can run on each host.
- Queues for HP® RGS Linux® sessions need to have `prolog` and an `epilog` respectively set to the `rgs.preexec.sh` and `rgs.postexec.sh` scripts, located under `$EF_TOP/<VERSION>/enginframe/plugins/interactive/tools` folder. They must also be run as root, since HP® RGS needs to operate on runlevels, so you might want to have something like:

```
prolog root@/path/to/enginframe/plugins/interactive/tools/rgs.preexec.sh
```

in the queue configuration.

Adaptive Computing® Moab®

In this scenario, Interactive Plugin relies on Moab® workload manager to allocate and reserve resources to run the interactive sessions.

Interactive Plugin requires that the user running EnginFrame Server (usually `efadmin`) is a Moab® administrator, in particular it must be able to check the status and query all Moab® jobs.

For example, it is possible to define `efadmin` as *Level 2 Moab® Admin* (Operator Access) by adding:

```
ADMINCFG[3]          USERS=efadmin
```

in `moab.cfg` configuration file.

EnginFrame Interactive Plugin requires Moab® Web Services (MWS) to communicate with Moab®.

SLURM™

In this scenario, Interactive Plugin relies on SLURM™ Workload Manager to allocate and reserve resources to run the interactive sessions.

Interactive Plugin requires that Features `vnc`, `dcv`, `dcv2` have been defined on SLURM™ configuration and that every allowed user must be able to check the status and query all SLURM™ jobs and partitions (alias queues). `'dcv2'` stands for DCV since 2017.

Also, Interactive Plugin requires that `RealMemory` (in MB units) parameter has been defined on SLURM™ configuration for every execution node, in order to show the correct maximum value of Memory.

Installing NICE EnginFrame

EnginFrame is distributed with an installer that guides you through the setup procedure making the installation straightforward. The installer is the EnginFrame package itself. Refer to [the section called “Downloading EnginFrame”](#) for instructions on getting your EnginFrame package.

Installing

You can use a graphical and a text based installer, where the latter is useful when installing on machines where you do not have access to an X Window System.



Unprivileged User Installation

Normally you would install EnginFrame as `root`.

Installation as an unprivileged user is possible, but has the following limitations:

- You cannot use authentication mechanisms that need `root` privileges, e.g. PAM.
- All services are executed as the user installing EnginFrame Agent.

Please be sure the `umask` is `022` before launching the installation commands.

Assuming Java™ is available in your `PATH`, to start the graphical installer simply run as `root`:

```
# java -jar enginframe-2019.0.x.y.jar
```

The graphical installer guides you through the installation procedure. If the X Window System is not available, the installer falls back to the text based one.

The user interface of the text based installer can be started by simply specifying `text` argument on the command line:

```
# java -jar enginframe-2019.0.x.y.jar --text
```

The installer shows you the terms of the license agreement and prompts you for a valid license file. If you do not have a valid license file, refer to [the section called “Obtaining a License”](#).

The installer then prompts you with some questions to tailor the EnginFrame deployment to your needs.

After asking all the questions the installer shows you a summary of your answers; this is when you can change values before installing. Once summary is accepted, the installer proceeds to set up EnginFrame on current host.



Note

When installing using a Java™ Runtime Environment the following warning may appear in the output:

```
Unable to locate tools.jar. Expected to find it in [...]
```

This warning is harmless and can be safely ignored.

After installation finishes EnginFrame is ready to be used. Refer to [Chapter 5, Running NICE EnginFrame](#) to learn how to start your EnginFrame Portal and test your installation.

A file named `efinstall.config` is saved in the directory from which you launched the installer. This file contains the options you specified during installation. This file may be useful to document how you installed EnginFrame as well as to replicate installation in *batch* mode without requiring user interactions.



Distributed Deployment

If you want to run Server and Agent on different hosts, launch the installer on the Server host and select a shared installation directory. The installer will ask if the Server and Agent will run on different hosts.

Batch Installation

Batch installation allows to easily replicate an installation on different hosts taking as input a configuration file created during a normal install. To run the EnginFrame installer in batch mode simply run

```
# java -jar enginframe-2019.0.x.y.jar --batch -f efinstall.config
```

if you do not specify the `-f` option the installer searches for a file named `efinstall.config` in the current directory.

Unless errors occur, EnginFrame installer completes the installation procedure without requiring further user input.

Fine Tuning Your Installation

Sometimes it is useful to fine tune your installation to make sure EnginFrame is performing at its best on your system or to customize EnginFrame behavior to match your needs.

Spooler Download URL

The EnginFrame Agent needs to communicate with EnginFrame Server when downloading a file from a spooler. In some network configurations and architectures - especially in the case a Web Server is placed in front of the EnginFrame Server - this callback URL has to be explicitly configured.

If downloading files from your EnginFrame Portal does not work properly, refer to [the section called “Configure Download URL on Agent”](#) for configuring EnginFrame Agent callback URL.

Optimizing JDK Options

The `SERVER_JAVA_OPTIONS` and `AGENT_JAVA_OPTIONS` options in `$EF_TOP/conf/enginframe.conf` contain command line options passed to JDKs starting respectively EnginFrame Server and EnginFrame Agent. Sometimes tweaking some of those values might be necessary, for instance in some cases it is useful to expand the size of the JVM heap (memory used for dynamic data allocation): this can be done by changing the `-Xmx` option. The following example sets heap size for Agent to 200 MB

```
AGENT_JAVA_OPTIONS="--Xms200m -Xmx200m -XX:MaxPermSize=50m \
-Djava.rmi.server.hostname=localhost -Djava.net.preferIPv4Stack=true \
-Dsun.rmi.dgc.client.gcInterval=3600000 \
-Dsun.rmi.dgc.server.gcInterval=3600000"
```

Distributed Resource Manager Options

Cluster Name Label

All grid plug-ins in EnginFrame support the option to specify a custom label to be used by the portal to show the name of the clusters.

Change the grid plug-in cluster name by specifying in the plug-in configuration file the `[PLUGIN-ID]_CLUSTER_LABEL` options:

- `$EF_TOP/conf/plugins/lsf/ef.lsf.conf: LSF_CLUSTER_LABEL=...`
- `$EF_TOP/conf/plugins/moabws/moabws.efconf: MOABWS_CLUSTER_LABEL=...`
- `$EF_TOP/conf/plugins/pbs/ef.pbs.conf: PBS_CLUSTER_LABEL=...`
- `$EF_TOP/conf/plugins/torque/ef.torque.conf: TORQUE_CLUSTER_LABEL=...`
- `$EF_TOP/conf/plugins/sge/ef.sge.conf: SGE_CLUSTER_LABEL=...`
- `$EF_TOP/conf/enginframe/plugins/slurm/ef.slurm.conf: SLURM_CLUSTER_LABEL=...`

- `$EF_TOP/conf/plugins/neutro/neutro.efconf:`
`NEUTRO_CLUSTER_LABEL=...`

Interactive Plugin

Distributed Resource Manager

The EnginFrame installer configures the Interactive Plugin depending on the resource manager selection at installation time. In case you want to change this setting the related configuration parameters are located into Interactive Plugin main configuration file: `$EF_TOP/conf/plugins/interactive/interactive.efconf`.

The parameters are named `INTERACTIVE_DEFAULT_LINUX_JOBMANAGER` and `INTERACTIVE_DEFAULT_WINDOWS_JOBMANAGER`. You can set their values to:

- `lsf` - sessions are scheduled by LSF® (*both Windows® and Linux®*) or OpenLava (*Linux®-only*).
- `neutro` - sessions are scheduled by NICE Neutro. (*Windows®-only*)
- `pbs` - sessions are scheduled by PBS Professional®.
- `moabws` - sessions are scheduled by Moab® Web Services. (*Linux®-only*)
- `torque` - sessions are scheduled by Torque. (*Linux®-only*)
- `sge` - sessions are scheduled by Sun® Grid Engine, Oracle® Grid Engine (OGE), Univa® Grid Engine® (UGE), Son of Grid Engine (SoGE) or Open Grid Scheduler.

You can override this behaviour on per-EnginFrame service basis by using `--jobmanager` option of `interactive.submit` session starter script.

Remote Visualization Technology

By default, Interactive Plugin is set to use VNC® remote visualization technology. In case you want to change this setting, the related configuration parameter is located into Interactive Plugin main configuration file: `$EF_TOP/conf/plugins/interactive/interactive.efconf`.

The parameter is named `INTERACTIVE_DEFAULT_REMOTE`. You can set its value to one of the following:

- `vnc` - manage RealVNC®, TurboVNC, TigerVNC sessions.
- `dcdv` - manage NICE DCV (up to 2016.0) 3D accelerated sessions.
- `dcdv2` - manage NICE DCV (since 2017.0) 3D accelerated sessions.
- `rgs` - manage HP® RGS sessions.
- `virtualgl` - manage VirtualGL sessions.

You can override this behaviour on per-EnginFrame service basis by using `--remote` option of `interactive.submit` session starter script.

EnginFrame Enterprise Installation

Load Balancer Setup

EnginFrame Enterprise requires a load balancer implementing the *sticky session* strategy.

The following sections describe a common solution based on AJP connector and Apache® Web Server frontend with `mod_proxy_balancer` module.

Configure AJP Connector

Enable the AJP Connector on Tomcat®:

- Login as root on the node of EnginFrame Server:

```
# cd $EF_TOP/conf/tomcat/conf
```

- Open `server.xml` file and uncomment the section that defines the AJP 1.3 connector.

```
<Connector
  port="8009"
  enableLookups="false"
  redirectPort="8443"
  protocol="AJP/1.3"
  URIEncoding="utf-8" />
```

If port 8009 is used by another application, then you must choose another value.

Configure Apache® Proxy

To enable Reverse Proxy Support in Apache® append the following lines to the main Apache® configuration file (`APACHE_TOP/conf/httpd.conf`) and reload the Apache® service.

```
<Location "/enginframe">
  DefaultType None
  ProxyPass      ajp://127.0.0.1:8009/enginframe flushpackets=on
  ProxyPassReverse ajp://127.0.0.1:8009/enginframe
</Location>
```

If your context is not `enginframe` then you have to change the `<Location>` and those two lines accordingly.

Configure Apache® `mod_proxy_balancer`

This configuration is required to balance the traffic over many EnginFrame instances.

Create a specific file, for instance `ef-ent.conf`, and add to the Apache® configuration directory (usually `/etc/httpd/conf.d`).

An example of the `ef-ent.conf` file content is:

```
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e;
path=/enginframe" env=BALANCER_ROUTE_CHANGED

<Proxy balancer://enterprise>
  BalancerMember ajp://<ip of EnginFrame Server 1>:8009 route=1
  BalancerMember ajp://<ip of EnginFrame Server 2>:8009 route=2
  ProxySet lbmethod=bybusyness
  ProxySet stickysession=ROUTEID
</Proxy>

<Location "/enginframe">
  ProxyPass          balancer://<enterprise hostname>/enginframe
  ProxyPassReverse  balancer://<enterprise hostname>/enginframe
</Location>
```

The enterprise alias is an internal parameter. Instead, the enginframe context can be changed if needed. The Header setup is needed in order to store internally the route id to pass it to *stickysession* variable, useful for cookie automatic management.

DBMS Setup

An empty database instance named EnginFrameDB (case sensitive) must be created *prior* to EnginFrame first startup. At the first startup, EnginFrame will create all the needed tables.

The following sections describe the steps to create the EnginFrameDB database instance on the supported DBMS.



Note

Do not start EnginFrame server(s) before the following steps are completed.

EnginFrame Configuration

During installation of EnginFrame Enterprise you will be prompted to insert the JDBC URL to the EnginFrame database instance together with the username and password with which to access. i.e.:

```
JDBC URL [default: jdbc:derby://localhost:1527/EnginFrameDB]
> jdbc:mysql://172.16.10.216:3306/EnginFrameDB
Username [default: dbadmin]
> enginframedb
Password
> efdpassword
```

MySQL® (version 5.1.x and higher)

- Use the command

```
# mysql -p
```

on the MySQL® server host to login as root;

- Create a new database by executing the following SQL query:

```
# CREATE DATABASE EnginFrameDB;
```

- Create a new user by executing the following SQL queries, using single quotes as listed below

```
# CREATE USER '<username>'@'<host>' IDENTIFIED BY '<password>';
```

E.g.:

```
# CREATE USER 'efdbadmin' IDENTIFIED BY 'efdbpassword';
# CREATE USER 'efdbadmin'@'%' IDENTIFIED BY 'efdbpassword';
```

The first statement allows access to localhost, i.e., for maintenance actions; in the second one, '%' is the wildcard used to allow all hosts. You can modify the latest statement to restrict the access to the EnginFrame Servers only;

- Grant privileges to the new user on the previously created DB by executing the following SQL query:

```
# GRANT ALL PRIVILEGES ON EnginFrameDB.* TO <username>
IDENTIFIED BY '<password>';
```

E.g.:

```
# GRANT ALL PRIVILEGES ON EnginFrameDB.* TO 'efdbadmin'
IDENTIFIED BY 'efdbpassword';
# GRANT ALL PRIVILEGES ON EnginFrameDB.* TO 'efdbadmin'@'%'
IDENTIFIED BY 'efdbpassword';
```

- Flush privileges in order to activate them on the created DB:

```
# flush privileges;
```

- Test the connection to the EnginFrameDB database. From one of server on which will be installed the EnginFrame server instance, using MySQL® client only, check the connection to the created database:

```
# mysql -h <mysql server hostname/ip address>[:<port>]
-u <username>
-p <password>
```

E.g.:

```
# mysql -h mysqlserver -u efdbadmin -p efpassword
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 87653
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Oracle DB (10 and higher)

- Login to the DB as admin using a SQL client like, for instance, Squirrel SQL Client, using the uri

```
jdbc:oracle:thin:@<hostname>:<port>:XE
```

E.g.:

```
jdbc:oracle:thin:@efentserverdb:1521:XE
```

- Create a new user by executing the following SQL query:

```
# CREATE USER <username> IDENTIFIED BY <password>
```

E.g.:

```
# CREATE USER enginframedb IDENTIFIED BY efdbpassword
```

a new schema with name equals to the username will be automatically created by the db engine: logging into the db with the new user, the user schema will be automatically used for that user;

- Grant privileges to the new user by executing the following SQL query:

```
# GRANT ALL PRIVILEGES TO <username>
```

E.g.:

```
# GRANT ALL PRIVILEGES TO enginframedb
```

If you need to restrict some privileges for the new user, please see the Oracle documentation, just be sure that the user has read/write permissions on his schema (the one automatically created, see above).

- Alter db user profile to avoid automatic expiration, issuing the following SQL query:

```
# ALTER PROFILE DEFAULT LIMIT PASSWORD_LIFE_TIME UNLIMITED;
```


This sets the *no password expiration* for all the default user's profile.

SQL Server® (2012, 2008)

- Enable TCP/IP networking and remote connections:
 - run SQL Server Configuration Manager;
 - go to SQL Server Network Configuration > Protocols for SQLEXPRESS;
 - make sure TCP/IP is enabled;
 - right-click on TCP/IP and select Properties;
 - verify that, under IP2, the IP Address is set to the computer's IP address on the local subnet;
 - scroll down to IPAll;
 - make sure that TCP Dynamic Ports is blank;
 - make sure that TCP Port is set to 1433.
- Create a new database named EnginFrameDB using the SQL Server Management Studio with default settings.
- Create a user with SQL Server® authentication using the SQL Server Management Studio, for instance, username: enginframedb and password: efdpassword
 - Make sure the user has at least the db_owner role on the EnginFrameDB, otherwise it will not be able to read/create tables. Check this in the Object Explore. Go to Security > Logins, right click on the username and choose Properties > User Mapping from the left panel.
 - EnginFrameDB database must be checked and with the right role membership checked (i.e.: checked db_owner and public);
 - (optional) Make EnginFrameDB the default database for the user, go to Security > Logins, right click on the username and choose Properties > General from the left panel.
- Allow SQL Server® to be accessed through "SQL Server and Windows Authentication mode" and not only through "Windows authentication mode"
 - Right-click on the DB server instance where EnginFrameDB has been created, e.g. SQLEXPRESS;
 - Select Properties, then Security, Server authentication and select "SQL Server and Windows Authentication mode";

At this point EnginFrame should be able to access SQL Server® DB and to create tables.

EnginFrame Service Configuration

The installer can configure the EnginFrame service to start at boot time on the node where it is run.

Assuming the installation was performed on a shared file system, to configure the EnginFrame service to start on all other dedicated nodes, log in on each node and run:

```
# $EF_TOP/bin/enginframe host-setup --roles=server,agent --boot=y
```

This configuration will start both server and agent components at node startup.

For more information on the `host-setup` options, run:

```
# $EF_TOP/bin/enginframe host-setup --help
```

EnginFrame Start

Once the EnginFrame service is installed, to manually start start it on every node, run:

```
# service enginframe start
```

Check if every database EnginFrameDB instance (MySQL®, Oracle® or SQL Server®) has been populated with EnginFrame tables.

Running NICE EnginFrame

This chapter describes how to start, stop, and check EnginFrame Portal's status. The administration monitoring and self checking services are also described.

Start, Stop, and Check Status

You control EnginFrame execution with `$EF_TOP/bin/enginframe` command line script.

Execute the following to start EnginFrame:

```
# $EF_TOP/bin/enginframe start
```

Execute the following to stop EnginFrame:

```
# $EF_TOP/bin/enginframe stop
```



Starting server/agent separately

If EnginFrame Server and EnginFrame Agent are on separate hosts, issue the following command on the host running EnginFrame Server:

```
# $EF_TOP/bin/enginframe <start|stop> server
```

Issue the following command on the host running EnginFrame Agent:

```
# $EF_TOP/bin/enginframe <start|stop> agent
```



Starting EnginFrame Enterprise

In the EnginFrame Enterprise deployment scenario, as described in [the section called “Deployment”](#), you will have to run the EnginFrame start (stop) command on each host of the EnginFrame Enterprise infrastructure.

```
# $EF_TOP/bin/enginframe <start|stop>
```

This control script also checks EnginFrame's status:

```
# $EF_TOP/bin/enginframe status
```

An example output of the previous commands follows:

```
# /opt/nice/enginframe/bin/enginframe start

Reading EnginFrame version from: /opt/nice/enginframe/current-version
Current version: 2017.0-r41442

EnginFrame Control Script
Loading configuration from:
- "/opt/nice/enginframe/conf/enginframe.conf"
Using EnginFrame in "/opt/nice/enginframe/2017.0-r41442"
Tomcat started.

[OK] EnginFrame Server started

[OK] EnginFrame Agent started
```

```

# /opt/nice/enginframe/bin/enginframe status
Reading EnginFrame version from: /opt/nice/enginframe/current-version
Current version: 2017.0-r41442

EnginFrame Control Script
Loading configuration from:
- "/opt/nice/enginframe/conf/enginframe.conf"
Using EnginFrame in "/opt/nice/enginframe/2017.0-r41442"

---- Server PID Information ----
USER      PID  PPID %CPU %MEM STIME      TIME COMMAND
efnobody  1674    1 69.9 17.4 19:34 00:01:47 /usr/lib/jvm/jre/bin/java
-Xms1024m -Xmx1024m -XX:HeapDumpPath=/[...]/dumps/server.pid1549.hprof
-Djava.protocol.handler.pkgs=com.enginframe.common.utils.xml.handlers
-XX:ErrorFile=/[...]/dumps/server.hs_err_pid1549.log
-DjvmRoute=efserver1 -DEF_LICENSE_PATH=/opt/nice/enginframe/license
-DDERBY_DATA=/opt/nice/enginframe/data/derby -DEF_ERRORS_DIR=/opt/nice/
enginframe/data/errors -Def.repository.dir=/opt/nice/enginframe/repository
-XX:+HeapDumpOnOutOfMemoryError -DEF_ROOT=/opt/nice/enginframe/2017.0-r41442/
enginframe -DEF_DYNAMIC_ROOT=/opt/nice/enginframe/2017.0-r41442/enginframe
-DEF_CONF_ROOT=/opt/nice/enginframe/conf -DEF_DATA_ROOT=/opt/nice/enginframe/
data -Def.tmp.dir=/opt/nice/enginframe/tmp/efserver1 -DEF_SPOOLER_DIR=/opt/
nice/enginframe/spoolers -DEF_SESSION_SPOOLER_DIR=/opt/nice/enginframe/
spoolers -DEF_LOGDIR=/opt/nice/enginframe/logs/efserver1
-Dfile.encoding=UTF -classpath :/opt/nice/enginframe/2017.0-r41442/tomcat/lib/
sdftree-handler.jar:/opt/nice/enginframe/2017.0-r41442/tomcat/bin/
bootstrap.jar:/opt/nice/enginframe/2017.0-r41442/tomcat/bin/tomcat-juli.jar
[...] org.apache.catalina.startup.Bootstrap start

---- Server Port Information ----
INFO: Starting ProtocolHandler ["http-bio-8443"]

---- Agent PID Information ----
root      1677    1  6.5  5.2 19:34 00:00:10 /usr/lib/jvm/jre/bin/java
-Xms512m -Xmx512m -XX:HeapDumpPath=/[...]/dumps/agent.pid1549.hprof
-XX:ErrorFile=/[...]/dumps/agent.hs_err_pid1549.log
-DEF_ROOT=/opt/nice/enginframe/2017.0-r41442/enginframe -DEF_DYNAMIC_ROOT=
/opt/nice/enginframe/2017.0-r41442/enginframe -DEF_CONF_ROOT=/opt/nice/
enginframe/conf -DEF_DATA_ROOT=/opt/nice/enginframe/data -DEF_SPOOLER_DIR=
/opt/nice/enginframe/spoolers -DEF_SESSION_SPOOLER_DIR=/opt/nice/enginframe/
spoolers -DEF_LOGDIR=/opt/nice/enginframe/logs/efserver1 -Dfile.encoding=UTF-8
-Djava.security.policy==/[...]/2017.0-r41442/enginframe/conf/ef_java.policy
[...] -jar /opt/nice/enginframe/2017.0-r41442/enginframe/agent/agent.jar

# /opt/nice/enginframe/bin/enginframe stop

Reading EnginFrame version from: /opt/nice/enginframe/current-version
Current version: 2017.0-r41442

EnginFrame Control Script
Loading configuration from:
- "/opt/nice/enginframe/conf/enginframe.conf"
Using EnginFrame in "/opt/nice/enginframe/2017.0-r41442"
Tomcat stopped.

[OK] EnginFrame Agent is down

```

Accessing the Portal

EnginFrame Portal can be accessed by a browser once EnginFrame daemons are running. Type EnginFrame Server's host name in your browser's address bar, followed by ':', and EnginFrame Server's port number.

EnginFrame Server port number was selected during installation and can be viewed with the command:

```
# $EF_TOP/bin/enginframe status
```

described in previous section.

For instance if EnginFrame Server host is named *myhost* and EnginFrame Server port number is 7070, type in your browser's address bar:

`http://myhost:7070`

If *myhost* name is not resolved by your DNS, you can specify the corresponding IP address:

`http://192.168.0.10:7070`



DNS Issues

For any issue about DNS name and domain resolution and IP address numbers, you should contact your network administrator.

EnginFrame Server is installed correctly if you see the welcome page. If your browser reports errors such as *Cannot find the requested page*, *Server not found*, *Problem loading page*, check EnginFrame is installed correctly by inspecting status as explained above.

Demo Sites

If you choose to install the EnginFrame *Developer's Documentation* during installation, the welcome page, together with the production portals Applications, Views and Administration Portal, will also display a link to the Technology Showcase that points to a set of demo services to illustrate EnginFrame services capabilities.



Demo and Administration Access

By default, only EF_ADMIN user can access administration and tutorial demo sites.

Administration Portal

EnginFrame includes an *Administration Portal* to monitor and manage some of its aspects directly from a web browser.

The Administration portal is linked from the EnginFrame welcome page or can be reached directly at

`http://host:port/context/admin`

The Administration portal offers a set of services divided into the following categories:

- Monitor
- Develop
- Troubleshooting

Monitor Services

The *Monitor* folder contains useful services to check logged users, to monitor usage statistics information and to get historical data and trigger information.

- *Server Load* service shows CPU usage, Java™ Virtual Machine memory usage, repository and spoolers file system size and i-node usage.
- *Usage Statistics* service shows current and historical data that involves the number of logged users, number of jobs with their status, number of interactive sessions and spoolers.
- *License Status* service enables the administrator to display current and historical usage of EnginFrame licenses.
- *Installed Components* displays the installed EnginFrame plugin list with their version.
- *Logged Users* service shows the users logged into the portal and gives the possibility to force user logout.
- *Triggers* service allows to check and manage scheduled triggers in EnginFrame.
- *ACL Actors* service shows EnginFrame ACL actors defined in `authorization.xconf` files loaded by the system.

Troubleshooting Services

The *Troubleshooting* folder contains useful services for checking portal status and health.

- *Run Self checks* service performs several operations to exercise different functions and aspects of EnginFrame. Every test outputs a result and, in most cases, a quick hint to correct the problem.
- *View Error Files* service enables the administrator to display error files generated by EnginFrame when services produce a wrong output. In those cases EnginFrame provides an error number that can be used here to open the associated error file.
- *Collect Support Info* service gathers a wide range of information about EnginFrame Portal and its configuration. This service output is a compressed archive containing all gathered information. Please attach this package when sending your request to EnginFrame support.

EnginFrame Statistics

To collect information about license usage, jobs usage and others useful statistics, EnginFrame uses RRD4J as round-robin database.

RRD4J is a high performance data logging and graphing system for time series data, implementing [RRDTool's](#) functionality in Java™. It follows much of the same logic and uses the same data sources, archive types and definitions as RRDTool does.

EnginFrame creates a database for general usage information named `efstatistics.rrd` and a database for each license file named `license_<component>_<expiration>_<maxToken>.rrd`. A new database will be created when license file changes.

In the `admin.statistics.efconf` configuration file you can configure some RRD4J specific parameter to change archive time intervals or to configure historical charts. See [RRD4J web site](#) for more information.

Database files are created by the `update.statistics` trigger at the very first run and updated at each succeeding iteration every 60 seconds by default.

Applications Portal

EnginFrame includes the *Applications Portal* to create, manage and submit both Batch and Interactive services.

The Applications portal is linked from the EnginFrame welcome page or can be reached directly at

`http://<host>:<port>/<context>/applications`

Applications portal offers two interfaces for two different users' roles: *Admin's Portal* and *User's Portal*.

Admin's Portal

The *Admin's Portal* contains useful services to monitor interactive sessions, jobs, hosts and to manage services, Applications users and portal appearance.

- *Monitor » All Sessions* service enables the administrator to manage interactive sessions for all Applications users.
- *Monitor » All Jobs* service allows the administrator to monitor and manage DRM jobs for all Applications users. In order to control the jobs of other users Applications administrator should also have the proper rights in the underlying DRM.
- *Monitor » Hosts* service enables the administrator to monitor the status of the hosts of the configured DRMs.
- *Manage » Services* service allows the administrator to fully manage, i.e. create, delete, edit, publish etc., batch and interactive services.
- *Manage » Users* service allows the administrator to register, import and manage Applications Users.
- *Manage » Appearance* service enables the administrator to change the company logo and the Portal's color theme.

User's Portal

The *User's Portal* exposes useful services to monitor user's data, sessions, jobs and hosts, together with Batch and Interactive services published by the Applications administrators.

- *Data » Spoolers* service shows the user's EnginFrame Spoolers and provides rename and delete operations.
- *Data » Files* service allows the user to browse and manage files in his home directory.
- *Monitor » Sessions* service enables the user to monitor and manage his interactive sessions.
- *Monitor » Jobs* service enables the user to monitor and manage his jobs.
- *Monitor » Hosts* service enables the user to monitor the status of the hosts of the configured DRMs.

Applications administrators can create and publish new services through the Admin's Portal. Service publishing allows to expose different services to different groups of users.



Applications License

Applications portal requires a specific license. You have to contact [<helpdesk@nice-software.com>](mailto:helpdesk@nice-software.com) or your EnginFrame reseller to purchase a license, perform a license change or obtain a demo license.

Views Portal

EnginFrame includes the *Views Portal* to create, manage and submit Interactive services.

The Views portal is linked from the EnginFrame welcome page or can be reached directly at

`http://<host>:<port>/<context>/vdi`

Views portal offers two interfaces for two different users' roles: *Admin's Portal* and *User's Portal*.

Admin's Portal

The *Admin's Portal* contains useful services to monitor interactive sessions, hosts and to manage Interactive services, Views users and portal appearance.

- *Monitor » All Sessions* service enables the administrator to manage interactive sessions for all Views users.
- *Monitor » Hosts* service enables the administrator to monitor the status of the hosts of the configured DRMs.
- *Manage » Interactive Services* service allows the administrator to fully manage, i.e. create, delete, edit, publish etc., Interactive services.
- *Manage » Users* service allows the administrator to register, import and manage Views users.
- *Manage » Appearance* service allows the administrator to change the company logo and the portal color theme.

User's Portal

The *User's Portal* exposes services to monitor user's sessions and cluster hosts, together with the Interactive services published by Views administrators.

- *Monitor » Sessions* service allows the user to monitor and manage his interactive sessions.
- *Monitor » Hosts* service enables the user to monitor the status of the hosts of the configured DRMs.

Views administrators can create and publish new services through the Admin's Portal. Service publishing allows to expose different services to different groups of users.

禁发网科技(CAXX-CLOUD)
info@caxx-cloud.com

PART II

Administration

教习网科技(CAXX-CLOUD)
info@caxx-cloud.com

聚辰科技(CAX-CLOUD)
info@cax-cloud.com

Common Administration Tasks

Most of the tasks an EnginFrame administrator has to perform involve editing configuration files. This chapter provides an overview of the main EnginFrame configuration files and then focuses on some common administration tasks, explaining in detail how to accomplish them.

This chapter describes the following tasks:

- Deploying a New Plugin
- Changing Java™ Version
- Changing Default Agent
- Managing Internet Media Types
- Customizing Error Page
- Limiting Service Output
- Configuring Agent Ports
- Customizing User Switching
- Customizing User Session Timeout
- Changing Charts Back-end

Other important administration tasks regarding EnginFrame Portal's specific sub-components (like spooler management, logging, etc.) are described in the next chapters.

Main Configuration Files

In this section the main EnginFrame configuration files are described. Further details can be found throughout this guide.

Starting from EnginFrame 2015, configuration files are isolated from the rest of EnginFrame installation in `$EF_TOP/conf`. Configuration files in `$EF_TOP/conf` are preserved during updates.

EnginFrame still uses internal configuration files located under `$EF_TOP/<VERSION>` and organized according to the pre-EnginFrame 2015 directory-tree layout (i.e. `$EF_TOP/<VERSION>/enginframe/conf` and `$EF_TOP/<VERSION>/enginframe/plugins/<plug-in>/conf,...`).

Some of these files define default values which can be overridden using files with the same name under the `$EF_TOP/conf` tree. Note any modifications to files under `$EF_TOP/<VERSION>` are discouraged and the files under this directory are subject to change in the next EnginFrame versions without notice.

禁发科技(CAXX-CLOUD)
info@caxx-cloud.com

`enginframe.conf`

It is located in `$EF_TOP/conf` directory.

As already seen in the section called “Fine Tuning Your Installation”, this file configures the JDK running EnginFrame Server and EnginFrame Agent and the execution options passed to the JVM. It also configures other execution environment parameters like locale or user running Tomcat® (referred to as `EF_NOBODY`).

`server.conf`

This is the server's main configuration file.

It is located in `$EF_TOP/<VERSION>/enginframe/conf` directory. Its contents are merged with `$EF_TOP/conf/enginframe/server.conf` if present. In case the same property is defined in both files the latter wins.

It also contains some parameters used by the *local agent* when executing services on EnginFrame Server's host on `EF_NOBODY`'s behalf.

`agent.conf`

This is the agent's main configuration file.

It is located in `$EF_TOP/<VERSION>/enginframe/conf` directory. Its contents are merged with `$EF_TOP/conf/enginframe/agent.conf` if present. In case the same property is defined in both files the latter wins.

`mime-types.xml`

It associates content types to files downloaded through the portal without requiring any change to the JDK settings.

It is located in `$EF_TOP/<VERSION>/enginframe/conf` directory. Its contents are merged, extended or overridden, with `$EF_TOP/conf/enginframe/mime-types.xml` if present. In the case the same MIME type is defined in both files the latter overrides the mapping.

Refer to the section called “Managing Internet Media Types” for more details.

`log.server.xconf`

It configures EnginFrame Server's logging.

It is located in `$EF_TOP/<VERSION>/enginframe/conf` directory. Overridden by `$EF_TOP/conf/enginframe/log.server.xconf` if present.

Refer to the section called “EnginFrame Server and Agent Logging” for more details.

`log.agent.xconf`

It configures EnginFrame Agent's logging.

It is located in `$EF_TOP/<VERSION>/enginframe/conf` directory. Overridden by `$EF_TOP/conf/enginframe/log.agent.xconf` if present.

Refer to the section called “EnginFrame Server and Agent Logging” for more details.

`authorization.xconf`

It's a configuration file for the EnginFrame authorization system. It defines users' groups and access control lists (ACLs).

Refer to the section called “Configuring Authorization”.

Deploying a New Plugin

Two types of plugins exist from a deployment point-of-view: the official ones distributed by NICE and the custom ones produced in-house or distributed by third parties.



Important

Plug-ins designed for pre-2015 EnginFrame versions cannot be installed on newer EnginFrame. Please contact NICE and check if updated plug-ins are available.

NICE's Official Plugins

NICE's official plugins are distributed with an installer that sets up the plugin and deploys it inside EnginFrame.

If NICE *eftoken* plugin (sold separately) were to be installed, it would be done by executing:

```
# java -jar eftoken-X.Y.Z.jar
```

The installer asks EnginFrame's root directory, plugin specific configuration options, and then installs the code.

If EnginFrame Server and EnginFrame Agent are installed on two different hosts, unless otherwise specified in plugin's documentation, the plugin has to be installed on both hosts.

Unless stated otherwise in plugin's documentation, once installed, the plugin is immediately available through EnginFrame Portal without requiring a restart.

Custom Plugins

If you have a custom plugin or are deploying a third party plugin that is not distributed with an installer, a manual deployment is necessary. All EnginFrame plugins *must be* placed inside `$EF_TOP/<VERSION>/plugins` and follow the internal structure described below.

A plugin directory structure example is:

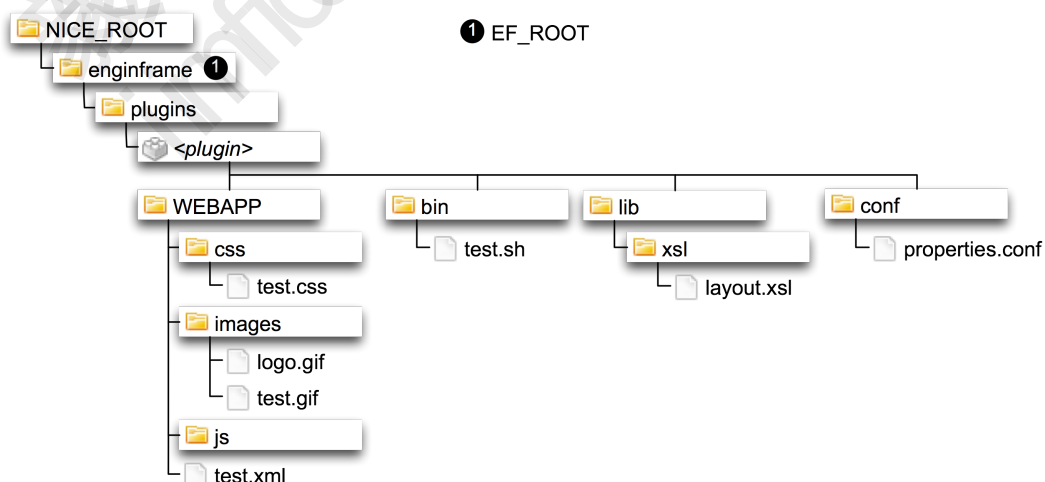


Figure 6.1. EnginFrame Plugin Structure

Some plugins may need additional setup operations to work properly. Read and follow the instructions distributed with plugin's documentation.

Changing Java™ Version

In some cases, e.g. an important security fix is shipped by Java™ vendor, changing the Java™ Platform running EnginFrame Portal is necessary.

The Java™ Platform running EnginFrame Server and EnginFrame Agent is defined using `JAVA_HOME` parameter inside `$EF_TOP/conf/enginframe.conf`. Changing this value is the only step necessary to use a different Java™ version to run these components.

Using Java™ installed in the directory `/opt/java` is done as follows:

```
JAVA_HOME="/opt/java"
```

EnginFrame Server and EnginFrame Agent restart is necessary to make changes effective.

Refer to [the section called “Java™ Platform”](#) for further information on supported Java™ versions.

Changing Default Agent

A single EnginFrame Server can connect to many EnginFrame Agents to execute services.

The remote *Default Agent* is specified during installation. When both server and agent are installed on the same host, this agent automatically becomes the default one. When the server is installed alone, the installer asks default agent's hostname and TCP port.

The default agent is defined in `server.conf` with two properties:

- `EF_AGENT_HOST`

Specifies default Agent hostname or IP address. The default value is `localhost`.

- `EF_AGENT_PORT`

Specifies default TCP port where Agent listens for incoming connections (i.e. parameter `ef.agent.port` in Agent configuration, see [the section called “Configuring Agent Ports”](#) for more details). Default value is `9999`.

It is located in `$EF_TOP/<VERSION>/enginframe/conf` directory. Its contents are merged with `$EF_TOP/conf/enginframe/server.conf` if present. In case the same property is defined in both files the latter wins.

Service Definition Files use both properties.

So if an EnginFrame Server wants to set its default agent as `agent1.nice` listening on port `7777` the parameters mentioned above become:

```
EF_AGENT_HOST=agent1.nice
EF_AGENT_PORT=7777
```

EF_AGENT_HOST can also be set to an IP address, e.g. 192.168.1.16.

Changes to these parameters do not require EnginFrame Server's restart.



Important

You *cannot* remove EF_AGENT_HOST and EF_AGENT_PORT from `server.conf`. If those properties are empty/missing, EnginFrame uses the default values.

Managing Internet Media Types

When a file is downloaded from EnginFrame Portal the browser tries to open it with the application configured to manage its media type content. For example, if an image is downloaded the browser displays it inline, while if a Word document is downloaded, it launches Office.

EnginFrame suggests to browsers the best method to handle files by sending the *Internet media type* in the download response.

An Internet media type, originally called *MIME type*, is a two-part identifier for file formats on the Internet. The identifiers were originally defined in [RFC 2046](#) for use in e-mails sent through SMTP, but their use has expanded to other protocols such as HTTP.

It is very useful to link uncommon file extensions to specific MIME types, so browsers can handle them correctly. Not all browsers use MIME type information in the same way. Older *Microsoft® Internet Explorer®* versions relies more on content type detection provided by *Windows® operating system* than on MIME type specified by the server's response message, especially for some of the most common file name extensions.

EnginFrame uses a built-in list of MIME types provided by the JDK. This list is defined in `JAVA_HOME/jre/lib/content-types.properties`.

EnginFrame overrides and extends this list of MIME types with `$EF_TOP/<VERSION>/enginframe/conf/mime-types.xml` in the static installation directory and with the optional `$EF_TOP/conf/enginframe/mime-types.xml` in the EnginFrame custom configuration directory tree. These files, owned by EnginFrame administrator, are used across the whole system unless more specific settings are found.

Each plugin has the possibility to extend and overrides the EnginFrame MIME types settings, by defining its own static `$EF_TOP/<VERSION>/enginframe/plugins/plugin_name/conf/mime-types.xml` and the associated customizable version `$EF_TOP/conf/plugins/plugin_name/mime-types.xml` in the EnginFrame configuration directory tree. MIME types defined in the plugin `mime-types.xml` files are used when downloading files from spoolers generated by services defined into `plugin_name` plugin.

When EnginFrame receives a download request from the browser, it tries to associate a MIME type to the file. It first looks in the plugin specific `mime-types.xml` file, then in EnginFrame `mime-types.xml`. In the case it cannot associate a MIME type from its own configuration files, it uses the default definitions specified in JDK's `content-types.properties`.

More in details, when looking for a MIME type of a file EnginFrame checks resources in the following order:

1. Custom plug-in MIME types, `$EF_TOP/conf/plugins/plugin_name/mime-types.xml`
2. Static plug-in MIME types, `$EF_TOP/<VERSION>/enginframe/plugins/plugin_name/conf/mime-types.xml`
3. Custom EnginFrame system-wide MIME types, `$EF_TOP/conf/plugins/plugin_name/mime-types.xml`
4. Static EnginFrame system-wide MIME types, `$EF_TOP/<VERSION>/enginframe/conf/mime-types.xml`
5. Default JDK MIME types, `JAVA_HOME/jre/lib/content-types.properties`

If this chain of lookup for a specific MIME type fails, the default MIME type, if defined, is returned, otherwise an empty MIME type is sent back in the HTTP response by EnginFrame.

EnginFrame Server dynamically reloads the `mime-types.xml` files when they are modified, so no restart is necessary.

EnginFrame ships this `$EF_TOP/<VERSION>/enginframe/conf/mime-types.xml`:

```

<?xml version="1.0"?>
<ef:mime-types xmlns:ef="http://www.enginframe.com/2000/EnginFrame">

  <ef:default type="text/plain" />

  <ef:mime-type>
    <ef:type type="text/plain"/>
    <ef:desc desc="ASCII Text File"/>
    <ef:extensions>
      <ef:extension ext=".log"/>
    </ef:extensions>
    <ef:match-list>
      <ef:match expr="[A-Z0-9]*.ef" />
      <ef:match expr="README" casesensitive="false" />
    </ef:match-list>
  </ef:mime-type>

  <ef:mime-type>
    <ef:type type="application/x-javascript"/>
    <ef:desc desc="JavaScript File"/>
    <ef:match-list>
      <ef:match expr="^.*[Jj][Ss]" />
    </ef:match-list>
  </ef:mime-type>

  <ef:mime-type>
    <ef:type type="application/json"/>
    <ef:desc desc="JSON File"/>
    <ef:match-list>
      <ef:match expr="^.*\.[Jj][Ss][Oo][Nn]" />
    </ef:match-list>
  </ef:mime-type>

  <ef:mime-type>
    <ef:type type="text/css"/>
    <ef:desc desc="CSS File"/>
    <ef:match-list>
      <ef:match expr="^.*[Cc][Ss][Ss]" />
    </ef:match-list>
  </ef:mime-type>

  <ef:mime-type>
    <ef:type type="image/vnd.microsoft.icon"/>
    <ef:desc desc="ICO File"/>
    <ef:match-list>
      <ef:match expr="^.*[Ii][Cc][Oo]" />
    </ef:match-list>
  </ef:mime-type>
</ef:mime-types>

```

The `<ef:extensions>` section contains exact matches. Thus, in this example, EnginFrame associates `text/plain` MIME type to any file ending with extension `.log` or `.patch`.

The `<ef:match-list>` section contains regular expressions for matching a file name. Thus, in this example, EnginFrame associates `text/plain` MIME type to all files whose name contains only alphanumeric characters and whose extension is `.ef` and to all files named `README`.

Since the `casesensitive` attribute is set to `false` in the first `<ef:match>` tag, a caseless matching is performed. This means, for example, that both files named `license.ef` or

LICENSE.EF are matched. If `casesensitive` attribute is not explicitly set to `false`, a case sensitive match is performed. So files named `readme` or `ReadMe` are not matched by the regular expression defined in the second `<ef:match>` tag.

`<ef:default>` is a child of `<ef:mime-types>` tag. It specifies a default MIME type for those cases where a MIME type cannot be guessed. The syntax is:

```
<ef:default type="expected_mime_type" forward-guess="[true/false]" />
```

Attribute `forward-guess` set to `true` allows to interrupt MIME type lookup at the current `mime-types.xml` file without considering upstream MIME type settings. Its default value is `false`.

The default value is also overridable following the same lookup order for `mime-types.xml` files as reported in the list above.



Important

There are two settings concerning security and MIME types into `server.conf` configuration file that are important to be described here: `ef.download.mimetype.mapping.text` and `ef.download.mimetype.mapping.octetstream`.

`ef.download.mimetype.mapping.text`: it's a comma separated list of MIME types that, for security reasons, are mapped to `text/plain` in the HTTP response to clients when downloading a file. This further MIME type mapping prevents browsers from interpreting and rendering the downloaded files protecting against malicious code that could be executed on the client browser (cross-site scripting attack).

`ef.download.mimetype.mapping.octetstream`: it's a comma separated list of MIME types that, for security reasons, are mapped to `application/octet-stream` in the HTTP response to clients when downloading a file. This further MIME type mapping prevents browsers from taking any action on the downloaded files protecting against malicious code that could be executed on the client host.

Refer to *EnginFrame Administrator's Reference* for more details on these XML tags.

Customizing Error Page

Whenever EnginFrame encounters an error during service execution, it displays an error message on the browser using a well known layout. All errors that end up on browser are displayed with the same look and feel.

Error page layout customization is achieved by changing `ef.error.layout` value inside `server.conf`. This value must be an absolute path to an XSL file containing customized stylesheets. `$EF_TOP/<VERSION>/enginframe/lib/xsl/com.enginframe.error.xsl` is the default value for `ef.error.layout`. This file can be used as a starting point to create customized templates.

So, if `$EF_TOP/<VERSION>/enginframe/plugins/mycompany/lib/xsl/mycompany.error.xsl` contains the customized XSL templates, `ef.error.layout` is set as follows:

```
ef.error.layout=${EF_ROOT}/plugins/mycompany/lib/xsl/mycompany.error.xsl
```

Changes to `ef.error.layout` do not require a server restart.

Limiting Service Output

EnginFrame's usual client is a web browser. Limiting amount of data sent to browsers saves resources on client-side. When a service execution produces a big amount of XML/HTML, the browser could have trouble rendering the page.

To avoid overloading the clients (and server/agent that have to produce/process the data), the maximum amount of data that services can produce is definable using `ef.output.limit`. If the limit is exceeded, the service's output is truncated and an error message is displayed on the browser.

`ef.output.limit` is specified as number of bytes and the default value is 10485760, i.e. 10 MB.

Since services are usually executed by a *remote agent*, this property is set inside the agent's `agent.conf`.

The following example shows how to set this property to limit service's output to 2 KB (2048 bytes) of data:

```
ef.output.limit=2048
```

However, since *local agent* can also execute services, `ef.output.limit` is also defined inside `server.conf`.

EnginFrame Agent and/or the EnginFrame Server restart is not required when changing this property.



Note

The service execution is not influenced in any way by the specified limit.
The service output is truncated on agent before sending it back to the server.

Configuring Agent Ports

EnginFrame Agent and EnginFrame Server communicate using Java™ RMI over SSL protocol. Technically speaking, EnginFrame Agent is an RMI server that exposes a remote object whose methods are invoked by EnginFrame Server.

For this reason, EnginFrame Agent needs to open two TCP ports on its host: one port is used by an *RMI Registry* while the other one is used by an RMI server. These ports are chosen during installation (by default they are respectively 9999 and 9998).

`$EF_TOP/conf/enginframe/agent.conf` contains the values specified during installation. Edit this file to change these values:

- `ef.agent.port`

Specifies TCP port on which RMI Registry is listening.

If this property is empty, the default port 9999 is used.

The specified value must be a valid TCP port that is not used by other processes on the same host.

- `ef.agent.bind.port`

Specifies TCP port on which RMI server is listening.

If this property is empty or is 0, a random free port is chosen at EnginFrame Agent startup.

The specified value must be 0 or a valid TCP port that is not used by other processes on the same host. Furthermore, the specified value must be different from `ef.agent.port`.

For example, using port 7777 for RMI Registry and port 7778 for RMI server, the two parameters must be set in the following way:

```
ef.agent.port=7777
ef.agent.bind.port=7778
```

EnginFrame Agent must be restarted to make changes effective.



Firewall Issues

If there is a firewall between EnginFrame Server and EnginFrame Agent then `ef.agent.bind.port` has to be set to a value different from zero. The firewall has to be configured to allow EnginFrame Server to open TCP connections towards EnginFrame Agent using the ports specified by `ef.agent.port` and `ef.agent.bind.port`.

If `ef.agent.port` is changed, then all `<ef:location>`'s port attributes have to change accordingly inside *Service Definition Files*.

Modify `EF_AGENT_PORT` inside `$EF_TOP/conf/enginframe/server.conf` if default agent's ports changed. Refer to [the section called “Changing Default Agent”](#) for more details.

Customizing User Switching

Unless EnginFrame was installed by an unprivileged user, every time EnginFrame Agent runs a service it *impersonates* the system user associated to portal user requesting service execution. This ensures service execution is performed as a regular system user (`root` is not allowed to run services) and the files created/modified have proper ownerships and permissions.

EnginFrame allows modifying how *user switching* is done to affect service execution environment and ultimately service execution itself.

EnginFrame Agent user switching mechanism is based on `su` shipped with every Linux®.

`$EF_TOP/conf/enginframe/agent.conf` configures `ef.switch.user.params` parameter used to specify options passed to `su`. Multiple parameters must be separated by a space without using quotes or double quotes like in the following example:

```
ef.switch.user.params=-f -m
```


An empty `ef.switch.user.params` means no options are passed to **su**:

```
ef.switch.user.params=
```

A missing `ef.switch.user.params` is automatically set to `-` which results in the user's profile being sourced when **su** is executed. By default, `ef.switch.user.params` property is not set.



Tip

If user profiles on EnginFrame Agent's host are complicated and sourcing them affects service execution performance, it is suggested to set `ef.switch.user.params` to avoid sourcing them when **su** is executed. You can, for example, set `ef.switch.user.params` to the empty string.

EnginFrame Agent does not have to be restarted when changing this parameter.

Customizing User Session Timeout

A session defines a user's working period within EnginFrame Portal. A session starts at user login and ends either when user logs out or when EnginFrame Server invalidates it.

Session timeout specifies the number of minutes a user can remain idle before the portal terminates the session automatically. If a user does not interact with EnginFrame within the configured timeout, the session is automatically invalidated and user has to reauthenticate to access EnginFrame Portal.

The default session timeout, defined for all users, is set to *30 minutes*.

Session timeout can be changed in EF_ROOT/WEBAPP/WEB-INF/web.xml by changing the `session-timeout` value. This value is expressed in a whole number of minutes. If the timeout is 0 or less, the container ensures the default behaviour of sessions is never to time out.

Changing session timeout to two hours, can be achieved modifying `session-timeout` value in the following way:

```
<session-config>
  <session-timeout>120</session-timeout>
</session-config>
```

Changes to session timeout require EnginFrame Server's restart.

Apache®-Tomcat® Connection

There are many reasons to integrate Tomcat® with Apache®. And there are reasons why it should not be done too. Starting with newer Tomcat (EnginFrame ships version 7.0.92), performance reasons are harder to justify. So here are the issues to discuss in integrating vs not:

- *Encryption* - The Apache HTTP Server module `mod_ssl` is an interface to the OpenSSL library, which provides Strong Encryption using the Secure Sockets Layer and Transport Layer Security protocols. Tomcat is able to provide a similar encryption using the JVM, which needs to be cross platform, so it is somehow less efficient than Apache. Moreover, Apache has a longer experience on this field.
- *Clustering* - By using Apache as a front end you can let Apache act as a front door to your content to multiple Tomcat instances. If one of your Tomcats fails, Apache ignores it and your Sysadmin can sleep through the night. This point could be ignored if you use a hardware loadbalancer and the clustering capabilities of EnginFrame Enterprise Edition.
- *Clustering/Security* - You can also use Apache as a front door to different Tomcats for different URL namespaces (`/app1/`, `/app2/`, `/app3/`, or virtual hosts). The Tomcats can then be each in a protected area and from a security point of view, you only need to worry about the Apache server. Essentially, Apache becomes a smart proxy server.
- *Security* - This topic can sway one way or another. Java™ has the security manager while Apache has a larger mindshare and more tricks with respect to security. Details will not be given here, but let Google™ be your friend. Depending on your scenario, one might be better than the other. But also keep in mind, if you run Apache with Tomcat you have two systems to defend, not one.
- *Add-ons* - Adding on CGI, perl, PHP is very natural to Apache. It's slower and more of a kludge for Tomcat. Apache also has hundreds of modules that can be plugged in at will. Tomcat can have this ability, but the code has not been written yet.

- *Decorators* - With Apache in front of Tomcat, you can perform any number of decorators that Tomcat does not support or does not have the immediate code support. For example, `mod_headers`, `mod_rewrite`, and `mod_alias` could be written for Tomcat, but why reinvent the wheel when Apache has done it so well?
- *Speed* - Apache is faster at serving static content than Tomcat. But unless you have a high traffic site, this point is useless. But in some scenarios, Tomcat can be faster than Apache. So benchmark *your* site.
- *Socket handling/system stability* - Apache has better socket handling with respect to error conditions than Tomcat. The main reason is that Tomcat must perform all its socket handling via the JVM which needs to be cross platform. The problem is that socket optimization is a platform specific ordeal. Most of the time the Java™ code is fine, but when you are also bombarded with dropped connections, invalid packets, invalid requests from invalid IPs, Apache does a better job at dropping these error conditions than JVM based program. (YMMV)

[Source: [Tomcat Wiki](#)]

There are at least two ways to configure an Apache Web Server as a frontend to Tomcat according to the protocol used:

- HTTP
- AJP [see [Protocol Reference](#)].

The connection between Apache and Tomcat using protocol AJP can follow two different strategies:

- Apache Module `mod_proxy_ajp` (Apache version 2.2 or higher)
- Tomcat Connector JK

Changing Charts Backend

Charts can be embedded dinamically into EnginFrame web pages.

By default, the internal charts provider is used, but is possible to use any other service compatible with Google™ Chart API.

The chart backend can be changed in two ways:

- Globally for all charts that EnginFrame produces.
- Locally for specific chart.

In the first case, edit `$EF_TOP/conf/enginframe/server.conf` specifying `ef.charts.base.url`:

```
ef.charts.base.url=http://chart.apis.google.com/chart
```

In the second case, set `base` attribute inside chart root tag:

```
<ch:chart ... base="http://chart.apis.google.com/chart" ... >
```

Interactive Administration

Configuration Files

Most of the times the values defined during the Interactive Plugin installation provide all the information necessary to have a working setup. However sometimes further configuration is needed to tailor the session broker to specific system and network conditions or to change the values defined during the installation.

All the Interactive Plugin configuration files are located in the `conf` subdirectory.



Note

All the parameters in the configuration files with extension different from `.efconf` comply with the following format from Bourne shell:

```
PARAMETER_NAME="parameter value"
```

In particular,

- There are *no spaces* before and after the `=` (equals).
- You can use shell variable references with the usual syntax `$variable`. Always enclose variable names with curly braces, for example: `${HOME}`.

- Bourne shell escaping and quoting syntax apply. Be sure to enclose values containing spaces within the most appropriate quotes.



Important

Configuration parameters are automatically loaded upon saving. No need to restart EnginFrame or logout.

interactive.efconf

This file contains Interactive Plugin's main default configuration parameters, that can be usually overridden by each portal service.

Default Parameters

INTERACTIVE_DEFAULT_OS

- value: *required*
- default: *linux*

By default, interactive session will be launched on the operating system stated by `INTERACTIVE_DEFAULT_OS` parameter.

Available values:

- `linux` - schedule on Linux® operating systems
- `windows` - schedule on Windows® operating systems

This behaviour can be overridden by each service itself by using `--os <system>` option of `interactive.submit`

Example:

```
INTERACTIVE_DEFAULT_OS=linux
```

INTERACTIVE_DEFAULT_JOBMANAGER

- value: *optional*
- default: *lsf*

Default job manager for submitting interactive session jobs. Each session will be scheduled as a single job.

This behaviour can be overridden by each service itself by using `--jobmanager <jobmanager>` option of `interactive.submit`



Note

Your EnginFrame installation requires the related grid middleware plugin to be installed and configured. Interactive Plugin will use it to submit and manage interactive session jobs.

Example:

```
INTERACTIVE_DEFAULT_JOBMANAGER=lsf
```

INTERACTIVE_DEFAULT_REMOTE

- value: *optional*
- default: *vnc*

Default visualization middleware to use.

Available values:

- `dcv` - use NICE DCV (up to 2016.0) visualization middleware (over RealVNC®)
- `dcv2` - use NICE DCV (since 2017.0) visualization middleware
- `vnc` - use VNC® visualization middlewares (RealVNC®, TigerVNC, TurboVNC and VirtualGL)
- `rgs` - use HP® RGS visualization middleware

Example:

```
INTERACTIVE_DEFAULT_REMOTE=dcv
```

INTERACTIVE_DEFAULT_VNC_QUEUE

- value: *optional*
- default: *(not set)*

Sets the default resource manager queue to use. Interactive session jobs will be submitted on that queue.

This behaviour can be overridden by each service itself by using `--queue <queue name>` option of `interactive.submit`

Example:

```
INTERACTIVE_DEFAULT_VNC_QUEUE=int_windows
```

Limits

INTERACTIVE_DEFAULT_MAX_SESSIONS

- value: *optional*

- default: *undefined (no limits)*

The maximum number of interactive sessions per interactive class.

If you set this default limit to X, each user will be able to start up to X sessions of the same interactive class.

For more informations about interactive classes and sessions limits, please refer to [the section called “Session limits”](#)

Example:

```
INTERACTIVE_DEFAULT_MAX_SESSIONS=3
```

interactive.<remote>.resolutions.conf

Inside this file you can specify some presets of the Remote Visualization Technology desktop geometry as a four-valued colon-separated string plus a label. The label must be separated by the previous fields by one or more spaces

```
widthxheight:fullscreen:allmonitors label
```

width and height are integer numbers and express the size in pixels, fullscreen and allmonitors are boolean flags {true|false} (case insensitive) and label is a human readable string describing the preset.

You can use the keyword `auto` to let the system guess the current screen resolution. If the list of presets includes a line containing the string `custom` (no other content on the same line), the user will be able to specify a custom resolution.



Note

Flag `allmonitors` is meaningful only when `fullscreen` is true. If you set `allmonitors=true` while `fullscreen=false`, then `allmonitors` parameter will be automatically converted to false.

Default content of the file:

```
auto          Fullscreen on single monitor (autodect resolution)
5120x1600:true:true  Fullscreen on two 30' monitors (5120x1600)
3840x1200:true:true  Fullscreen on two 24' monitors (3840x1200)
2560x1600:true:false Fullscreen on single 30' monitor (2560x1600)
1920x1200:true:false Fullscreen on single 24' monitor (1920x1200)
1024x768:false:false Window-mode on singla XGA monitor (1024x768)
custom
```

authorization.xconf

This file contains the ACL (Access Control List) definitions specific to Interactive Plugin. It defines some ACLs that are used in the demo portal to allow or deny access to the different visualization middlewares to different users.

For more details on EnginFrame ACL system, general EnginFrame authorization and its configuration, please refer to EnginFrame Administrator's Guide, *Security* section, *Authorization System* chapter.

nat.conf

If you set up NAT (Network Address Translation) so that the client machines connect to the cluster nodes through a different IP:PORT pair, this file allows to map IP:PORT pairs for services running on a node to the corresponding public IP:PORT pair.

**Note**

Some visualization middleware clients require that the actual port of the service equals the NATted port

The syntax consists of a line made of two pairs: the real IP:PORT pair followed by the public IP:PORT pair. It is possible to specify a group of ports using the fromPORT-toPORT syntax.

Example:

```
node01 mycompany.com
node01 10.100.0.101
node12:42976 mycompany.com:42976
node01:7900-7910 mycompany.com:5900-5910
node05:7900-7910 10.100.0.101:5900-5910
```

A session starting on host node01, port 7901 would be returned to the client as mycompany.com:5901

proxy.conf

If you give access to cluster nodes through a proxy, you can configure this file to assign for each connection a specific proxy server to use.

**Note**

This configuration applies only to VNC® and DCV connections.

The default configuration is to have a direct connection from any client to any server, so no proxy for all connections.

The syntax consists of a table, each line has the following columns: PRIORITY, CLIENT-FILTER, SERVER-FILTER, PROXY-TYPE, PROXY-ADDRESS:

PRIORITY

a number to rank the proxy list, 0 is the highest priority

CLIENT-FILTER

the range of IP addresses in the format: NETWORK/PREFIX

Examples:

```
10.20.0.0/16
0.0.0.0/0 (matches any IP address)
```

SERVER-FILTER

a glob pattern matching the server hostname

Examples:

```
node*           (matches any node starting with "node")
node0[1-9]      (matches hosts from node01 to node09)
```

PROXY TYPE

the proxy type to use, can be:

HTTP

proxy must support HTTP Connect protocol

SOCKS

proxy must support SOCKS5 protocol

DIRECT

special value to specify no proxy

PROXY-ADDRESS

the proxy hostname and port in the format *host:port* (not used in case proxy type is DIRECT).

Examples:

```
squidproxy.domain:3128
danteproxy:80
10.20.1.1:3128
```

In case multiple proxies with the same priority match, one of them is selected using an internal strategy.

In case no proxy for a priority matches, the proxies in the next priority are checked.

In case no proxy line matches, an error is returned to the client.

Example: no connection will receive a proxy configuration

```
99      0.0.0.0/0      *      DIRECT
```

Example: only connections to node01 will pass through proxyserver:3128, all other connections will be direct.

```
1      0.0.0.0/0      node01      SOCKS      proxyserver:3128
99     0.0.0.0/0      *      DIRECT
```

Example: connections from IP 10.20.3.20 to node01 will pass through proxyserver:80, other connections to node01 will pass through proxyserver:3128, all other connections will get an error.

```
0      10.20.3.20/32  node01      HTTP      proxyserver:80
1      0.0.0.0/0      node01      SOCKS      proxyserver:3128
```

url.mapping.conf

The new URL mapping configuration allows EnginFrame administrators to configure the target endpoints that will be used by clients to connect to the NICE DCV (since 2017.0) remote servers.

The configuration file to define the target DCV servers URLs endpoints is `${EF_CONF_ROOT}/plugins/interactive/url.mapping.conf`.

In this configuration file the administrator can write multiple mappings each one defining a matching rule and a target endpoint. Each rule can match one or more DCV servers as provided upstream by the system by using a set of predefined variables and glob expression. For each match the configuration provides a mapped endpoint that is a triple that includes the host, port and web URL path that will be used by clients to connect to the target DCV server.

Inside the `${EF_CONF_ROOT}/plugins/interactive/url.mapping.conf` configuration file it is possible to use the usual set of EnginFrame environment variables available during a service execution (e.g. `EF_*`, session variables) together with the interactive session metadata and a new set of noteworthy variables:

- `${server_host}` - the remote DCV server host as provided by the system in the upstream process;
- `${server_port}` - the remote DCV server port as configured on the DCV server node;
- `${server_web_url_path}` - the DCV server web URL path as configured on the DCV server node;
- `${session_id}` - the DCV session ID;
- `${nat_server_host}` - the value of the DCV server host coming from `${EF_CONF_ROOT}/plugins/interactive/nat.conf`;
- `${nat_server_port}` - the value of the DCV server port coming from `${EF_CONF_ROOT}/plugins/interactive/nat.conf`;
- `${proxy_host}` - the proxy host coming from `${EF_CONF_ROOT}/plugins/interactive/proxy.conf`;
- `${proxy_port}` - the proxy port coming from `${EF_CONF_ROOT}/plugins/interactive/proxy.conf`;

Every single value of the tuple, target host, target port and target web URL path, is evaluated separately. Variables are expanded and command substitution executed.



Important

Parameters evaluation is performed on behalf of the user running the Apache Tomcat® server (e.g. `efnobody`), on the host where EnginFrame runs.



Note

The only supported protocol for the mapped URL is HTTPS and cannot be changed.

For further information and examples consult directly the `${EF_CONF_ROOT}/plugins/interactive/url.mapping.conf` configuration file.

xstartup files

The configuration directory also contains a collection of sample `xstartup` files named `*.xstartup` that may be used in your service definitions to start a X session with the specified Window Manager.

Common tasks for `xstartup` scripts are, e.g. launching `dbus` daemon, opening an `xterm` window or setting specific Window Manager parameters.

An example `xstartup` script:

```
#!/bin/bash
[ -r $HOME/.Xresources ] && xrdp $HOME/.Xresources
xsetroot -solid grey
vncconfig -iconic &
xterm -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
```

The `xstartup` files shipped with Interactive Plugin are:

- `gnome.xstartup`, `xstartup` script for GNOME window manager.
- `icewm.xstartup`, `xstartup` script for ICE window manager.
- `kde.xstartup`, `xstartup` script for KDE window manager.
- `mate.xstartup`, `xstartup` script for MATE window manager.
- `metacity.xstartup`, `xstartup` script for Metacity window manager.
- `mwm.xstartup`, `xstartup` script for Motif window manager.
- `xfce.xstartup`, `xstartup` script for Xfce window manager.
- `xfwm4.xstartup`, `xstartup` script for Xfwm4 window manager.

Default Xstartup are `gnome.xstartup` for desktop sessions and `mwm.xstartup` for standalone interactive applications.

mime-types.xml

This file defines some mime-types useful for Interactive Plugin. Mime-types in this context are used to associate client viewers like VNC® Viewer to files generated by Interactive Plugin with specific extensions.

File extensions specified in this file are `.dcm`, `.vnc`, `.efrfs`, `.rgreceiver` and `.ica`.

For more details on EnginFrame mime-types configuration and customization, please refer to EnginFrame Administrator's Guide, *Administration* section, *Common Administration Tasks* chapter.

Interactive Session Life-cycle Extension Points

Interactive Session Dynamic Hooks

EnginFrame, starting from version 2017.2, adds two new extension points (hooks) to the interactive session life cycle.

The first customisable hook, basically a shell script, it's called when the session has been successfully setup on the remote host and it's ready to pass to the “Running” state. This hook is meant to execute some simple setup operations at session startup, e.g. to dynamically configure a gateway technology as the AWS™ Application Load Balancer (ALB) or an Nginx instance, enabling the clients to access the underlying dynamic infrastructure.

The hook has also the capability to add custom metadata to the session and to configure the target host, port and web URL path tuple to be used by the clients to connect to the session.

In order to set the connection parameters to be used by the clients to connect to the interactive session, the hook script has to export the following variables in the environment:

- `INTERACTIVE_SESSION_TARGET_HOST`
- `INTERACTIVE_SESSION_TARGET_PORT`
- `INTERACTIVE_SESSION_TARGET_WEBURLPATH`

These variables will be set as session metadata and used to forge the session URL and connection .dcv file upon a client request through the EnginFrame portal. The configuration of these interactive session settings will have the precedence over the static configuration files (e.g. `nat.conf`, `url.mapping.conf`) in defining the connection parameter for the clients.

Inside the hook script, it is possible to use the usual EnginFrame environment variables together with the session metadata variables. Noteworthy variables that can be useful to the hook logic are:

- `${INTERACTIVE_SESSION_REMOTE_SESSION_ID}` - the ID for a DCV 2017 session;
- `${INTERACTIVE_SESSION_EXECUTION_HOST}` - the execution host of the DCV session as determined internally by the system;
- `${INTERACTIVE_SESSION_DCV2_WEBURLPATH}` - the web URL path of the DCV server, as configured in `/etc/dcv/dcv.conf` on the DCV server node;
- `${INTERACTIVE_DEFAULT_DCV2_WEB_PORT}` - the web port of the DCV server, as configured in `${EF_CONF_ROOT}/plugins/interactive/interactive.efconf`;

In a specular way, the second customisable hook it's executed when the session goes in the “Closed” or “Failed” state.

The locations of the starting and closing hooks are configured in `${EF_CONF_ROOT}/plugins/interactive/interactive.efconf` through the variables `INTERACTIVE_SESSION_STARTING_HOOK` and `INTERACTIVE_SESSION_CLOSING_HOOK` respectively.

Hooks execution is done by the user running the Apache Tomcat® server (e.g. `efnobody`), and their standard output and standard error are logged into two log files in the interactive session spooler. They are accessible via web from the session details view.



Important

In case of errors the starting hook will block the session startup avoiding the session to go in the “Running” state. If the starting hook fails, it will keep the

session in the “Starting” status, and the session will be flagged with a warning message.

The result of the closing hook instead doesn't prevent the session to go in the terminal state. If the closing hook fails the session will anyway terminate and it will be flagged with a warning message.



Warning

The execution of the hooks can be triggered either by a user action (e.g. submission or closing operation) or by the EnginFrame internal process that updates the interactive sessions status. At the moment there is no mutual-exclusion mechanism in place and hooks may run concurrently on the same session. It's up to the hook scripts to be concurrency-safe.

Hooks also allow to set custom metadata to the interactive session. Any environment exported variable with prefix `SESSION_` will be added as metadata to the interactive session. All the session metadata are available to the hook scripts.

Sample Starting and Closing Hooks to Configure an AWS™ ALB

Sample starting and closing hooks to dynamically configure the AWS™ Application Load Balancer (ALB) on session creation and session closing, are provided in:

- `${EF_ROOT}/plugins/interactive/bin/samples/sample.alb.session.starting.hook.sh`
- `${EF_ROOT}/plugins/interactive/bin/samples/sample.alb.session.closing.hook.sh`

It is suggested to copy the scripts under `${EF_DATA_ROOT}/plugins/interactive/bin` before configuring or modifying them.

These scripts configure an AWS™ ALB to enable a connection to a host where a NICE DCV (since 2017.0) interactive session is running.

The starting hook script creates a new Target Group containing the instance where the Session is running and adds a new Listener Rule for the HTTPS listener of the ALB.

The Listener Rule has the role to associate the input URL path to the Target Group. This path must be the web URL path of the DCV server running on the execution node.



Important

Since it not possible to do URL path translations with an ALB, every DCV server must have an unique web URL path configured. It is suggested to use the hostname of the node as web URL path for the DCV server running on that node.

The maximum number of Listener Rule(s) per ALB is 100, hence a single ALB can handle at most 100 interactive sessions running concurrently. To increase this limit, please consider to add more ALBs in the infrastructure and to implement a rotation in the starting hook script.

Prerequisites for using the sample hook scripts provided:

- On EnginFrame node:
 - AWS™ Command Line Interface (CLI) must be installed;
 - Since this script is going to be executed by the user running the EnginFrame Server, i.e. the Apache Tomcat® user, an AWS™ CLI profile must be configured for that user, having the permissions to list instances and to manage load balancers. (see [CLI Getting Started](#)). Alternatively, if EnginFrame is installed into an EC2 instance, a valid AWS™ role to perform the above mentioned operations should be added to this instance;
- On AWS™ account:
 - An AWS™ Application Load Balancer with an HTTPS listener with a Default Target Group must be already configured and running;

On DCV server nodes:

- Each DCV server node must be configured with a unique web URL path (see `/etc/dcv/dcv.conf` configuration file);

The following is an example of the steps to do in order to use the sample AWS™ ALB hook scripts provided:

- copy the samples from `${EF_ROOT}/plugins/interactive/bin/samples` to `${EF_DATA_ROOT}/plugins/interactive/bin` and be sure that are executable;
- add the two configuration variables inside `${EF_CONF_ROOT}/plugins/interactive/interactive.efconf`:
 - `INTERACTIVE_SESSION_STARTING_HOOK=${EF_DATA_ROOT}/plugins/interactive/bin/sample.alb.session.starting.hook.sh`;
 - `INTERACTIVE_SESSION_CLOSING_HOOK=${EF_DATA_ROOT}/plugins/interactive/bin/sample.alb.session.closing.hook.sh`
- modify the hooks to change the value of the AWS™ ALB Public DNS name, through the variable `ALB_PUBLIC_DNS_NAME`;
- configure the AWS™ role to let the EC2 instance where EnginFrame is running to manage the ALB. From the AWS™ EC2 Console, select EC2 instance -> Instance Settings -> Attach/Replace IAM Role. The following is just an example, more restrictive rules can be used instead:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "elasticloadbalancing:*"
      ],
      "Resource": "*"
    }
  ]
}
```

Common errors from the hook execution:

- “An error occurred (AccessDenied) when calling the <xyz> operation: User: <abc> is not authorized to perform <xyz>”. The user running the hook (i.e. the user running the Apache Tomcat® server) is not authorized to perform the required operation. The AWS™ CLI profile must be configured for that user, having the permissions to list instances and to manage load balancers (see [CLI Getting Started](#)) or alternatively, if EnginFrame is installed into an EC2 instance, configure the correct AWS™ role for that instance.
- Getting “502 Bad Gateway” when connecting to the interactive session. It often means the Target Group of the ALB listener rule is not yet initialized with the target instance. In this case the system usually requires few more instants before establishing the connection with the instance.

Session limits

Number of sessions

The number of interactive sessions can be limited according to:

- the *interactive session class*. An interactive session class is a group of interactive services. Classes defined and customizable by EnginFrame administrators. Interactive classes are defined by setting the following metadata:
 - `INTERACTIVE_CLASS` - an unique identifier for the class.
 - `INTERACTIVE_CLASS_LABEL` - a label for the class (optional).

The maximum number of interactive sessions for each class is defined by setting `INTERACTIVE_MAX_SESSIONS` parameter. The default value is defined into *interactive.efconf* configuration file.

- the *interactive service*. Each interactive service can be assigned to a certain class and can define the `INTERACTIVE_MAX_SESSIONS` parameter within its code.
- the *user or user group*. With the use of EnginFrame ACLs (Access control lists) combined with the first two items. For detailed documentation about ACLs, please refer to EnginFrame Administrator's Guide.

A full example follows:

```
<ef:service id="interactive.xterm">
  <ef:name>XTerm</ef:name>
  <ef:metadata attribute="INTERACTIVE_CLASS">xterm</ef:metadata>
  <ef:metadata attribute="INTERACTIVE_CLASS_LABEL">Xterm</ef:metadata>
  <ef:metadata attribute="INTERACTIVE_MAX_SESSIONS">3</ef:metadata>
  <ef:option id="project" label="Project" type="text">Interactive</ef:option>
  <ef:option id="jobmanager" label="Job Manager" type="list">
    <ef:embed id="grid.plugins"/>
  </ef:option>
  <ef:action id="submit" label="Start" result="text/xml">
    "${EF_ROOT}/plugins/interactive/bin/interactive.submit" \
    --name "XTerm" \
    --os "linux" \
    --jobmanager "${jobmanager}" \
    --project "${project}" \
    --remote "vnc" \
    --vnc-xstartup "${EF_ROOT}/plugins/interactive/conf/metacity.xstartup" \
    --close-on-exit \
    --command "xterm"
  </ef:action>
</ef:service>
```

In the example above, the maximum number of sessions of the "xterm" class will be limited to 3 for each user



Note

Interactive session name and interactive class are not dependent to each other.

Log files

Main interactive log file is located under EnginFrame log directory: `${EF_LOGDIR}/interactive.log`.

All others log files (session, debug, authentication) can be found inside each interactive session spooler, and are available via the portal user interface, by reaching *session details* page.

Interactive Plugin Directory Structure

This section describes the directory structure. Please refer to the EnginFrame Administrator's Guide for details about the formats and the purpose of the files.

Interactive Plugin is installed in `${EF_ROOT}/plugins/interactive`.

These are the most important contents of the folder:


```
interactive/  
|-- WEBAPP  
|-- bin  
|-- conf  
|   |-- mappers  
|   |   |-- interactive.duplicated.sessions.xconf  
|   |   |-- interactive.list.sessions.xconf  
|   |-- dcv.gpu.balancer.conf  
|   |-- dcv2.gpu.balancer.conf  
|   |-- gnome.xstartup  
|   |-- icewm.xstartup  
|   |-- interactive.efconf  
|   |-- interactive.vnc.resolutions.conf  
|   |-- kde.xstartup  
|   |-- log.xconf  
|   |-- lxde.xstartup  
|   |-- mate.xstartup  
|   |-- metacity.xstartup  
|   |-- mime-types.xml  
|   |-- minimal.xstartup  
|   |-- mwm.xstartup  
|   |-- nat.conf  
|   |-- proxy.conf  
|   |-- template.dcv  
|   |-- template-dcv2.dcv  
|   |-- template.efrgs  
|   |-- template.rgreceiver  
|   |-- template.vnc  
|   |-- xfce.xstartup  
|   |-- xfwm4.xstartup  
|-- etc  
|-- lib  
|-- tools
```

Interactive Plugin follows the conventional EnginFrame plug-in structure and in particular the following directories contain Interactive Plugin system files:

- **WEBAPP** - top level service definition files.
- **bin** - scripts and executables.
- **conf** - configuration files.
- **etc** - Interactive Plugin metadata and EnginFrame descriptor files.
- **lib** - internal files used by Interactive Plugin: XML support services and XSL files.
- **tools** - Interactive Plugin integration and interface tools.

Each resource manager plugin supported by Interactive Plugin, includes an *interactive* subdirectory which contains the the related interface code with Interactive Plugin:


```

interactive/
|-- interactive.close
|-- interactive.is.session.ready
|-- interactive.log.data
|-- interactive.retrieve.screenshot
|-- interactive.submit
|-- linux.jobscript.functions
|-- services
|   |-- interactive.lsf.linux.xml
|   |-- interactive.lsf.shared.xml
|   `-- interactive.lsf.windows.xml
`-- windows.jobscript.bat

```

in particular:

- `interactive.submit` - the session submission script.
- `interactive.close` - the operations to be performed to close the session.
- `services` - various service definitions.

Views Administration

Views portal is implemented by the VDI plugin that defines all the services that interact with the backend to provide the high level, user functionalities.

VDI plugin provides a front-end portal that gives EnginFrame administrators an easy way to create, publish and manage Interactive services. End-users instead are provided with a portal to easily access the company Interactive services.

This section explains the configuration files and settings of the VDI plugin.

Configuration Files

Most of the times the values defined during the VDI plugin installation provide all the information necessary to have a working setup. However the administrator may have the need to change the folders where services files are stored or to change other settings of the system.

All the VDI plugin configuration files are located in the `$EF_TOP/<VERSION>/enginframe/plugins/vdi/conf` subdirectory.



Important

As for the other EnginFrame plugins the correct way to change default configuration is to copy the target configuration file under the `$EF_TOP/conf`, into the `$EF_TOP/conf/plugins/vdi` directory, if it doesn't already exist, and edit the copied file.



Note

Configuration parameters are automatically loaded upon saving. No need to restart EnginFrame or logout.

vdi.conf

This file contains VDI main default configuration parameters.

Users Access Parameters

VDI_ALLOW_ALL_USERS

- value: *required*
- default: *true*

Enables Views portal access to all the users able to log into the system. If *true*, the users will be added to the VDI default group at login time.

Available values:

- *true* - all user are allowed to access to the Views portal, i.e. VDI plugin services
- *false* - Views portal users should be explicitly added or imported by a Views administrator

Example:

```
VDI_ALLOW_ALL_USERS=true
```

service-manager.efconf

This file contains VDI plugin configuration parameters useful for service management.

General Parameters

VDI_SERVICES_ROOT

- value: *required*
- default: *\${EF_DATA_ROOT}/plugins/vdi/services* where *EF_DATA_ROOT* is *\$EF_TOP/data*

Sets the root folder where services files are stored. Changing this value, you have to change also

- *sdfree* URL value inside *href* attribute of *xi:include* tags declared in *vdi.xml*, *vdi.admin.xml* XML files.
- *load-conf* value of *ef:action* tags in the services XML files

Example:

```
VDI_SERVICES_ROOT=${EF_DATA_ROOT}/plugins/vdi/services
```

SM_TEMPLATES_ROOT

- value: *required*
- default: `${EF_ROOT}/plugins/vdi/templates`

Sets the root folder where templates files for service creation are stored.

Example:

```
SM_TEMPLATES_ROOT=${EF_ROOT}/plugins/vdi/templates
```

Interactive Services Parameters

SM_CATALOG_INTERACTIVE

- value: *required*
- default: `${VDI_SERVICES_ROOT}/catalog`

Sets the folder where unpublished interactive services files are stored.

Example:

```
SM_CATALOG_INTERACTIVE=${VDI_SERVICES_ROOT}/catalog
```

SM_PUBLISHED

- value: *required*
- default: `${VDI_SERVICES_ROOT}/published`

Sets the folder where published interactive services files are stored.

Example:

```
SM_PUBLISHED=${VDI_SERVICES_ROOT}/published
```

interactive.editor.efconf

This file contains configuration parameters for the interactive service editor in the Views portal.



Note

In EnginFrame version 2015.0 this file was named `vdi.editor.efconf` and had a slightly different set of configuration parameters.

During the installation of a newer version, EnginFrame makes a copy of the old configuration file under `$EF_TOP/conf/plugins/vdi` directory, naming it as `vdi.editor.efconf.backup`.

Interactive Editor Parameters

VDI_EDITOR_OS

- value: *optional*
- default: *windows,linux*

Sets supported operating systems for interactive sessions. Comma separated list without any blank space.

Available values:

- windows - Windows® Desktop
- linux - Linux® Desktop

Example:

```
VDI_EDITOR_OS=windows,linux
```

VDI_EDITOR_CLUSTERS

- value: *optional*
- default: Cluster ids retrieved from system

Sets cluster ids to list on service editor. Comma separated list without any spaces.

Example:

```
VDI_EDITOR_CLUSTERS=clusterid1,clusterid2
```

VDI_EDITOR_CLUSTERS_ARCH_clusterId

- value: *optional*
- default: linux for all the clusters id, except for neutro, which is windows and except for lsf which is windows and linux

Sets supported operating systems for interactive sessions scheduled in a specific cluster. Comma separated list without any spaces. `clusterId` is one of the cluster id defined in the `VDI_EDITOR_CLUSTERS` list.

Available values:

- windows - Windows® Desktop

- linux - Linux® Desktop

Example:

```
VDI_EDITOR_CLUSTERS_ARCH_openlavaCluster=linux
VDI_EDITOR_CLUSTERS_ARCH_myCluster=windows,linux
```

VDI_EDITOR_REMOTES

- value: *optional*
- default: *vnc,dcv,dcv2,virtualgl,rgs*

Sets the list of supported remote visualization technologies to display in the service editor. Comma separated list without any spaces.

Available values:

- vnc - Virtual Network Computing®
- dcv - NICE Desktop Cloud Visualization (up to 2016.0)
- dcv2 - NICE Desktop Cloud Visualization (since 2017.0)
- virtualgl - VirtualGL
- rgs - HP® Remote Graphics Software

Example:

```
VDI_EDITOR_REMOTES=vnc,dcv,dcv2,virtualgl,rgs
```

VDI_EDITOR_REMOTES_ARCH_remoteId

- value: *optional*
- default: linux, windows for vnc, dcv and dcv2, linux and windows for rgs, linux for virtualgl

Sets supported session types for a specific remote id. Comma separated list without any spaces. remoteId is one of the remote id defined in the VDI_EDITOR_REMOTES list.

Available values:

- windows - Windows® Desktop
- linux - Linux® Desktop
- linux-app - Linux® Desktop application

Example:

```
VDI_EDITOR_REMOTES_ARCH_vnc=linux
VDI_EDITOR_REMOTES_ARCH_rgs=windows
```

VDI_EDITOR_DESKTOP_MANAGERS

- value: *optional*
- default: *none*

Sets the list of supported desktop manager ids to display on the service editor. Comma separated list without any spaces.

For each desktop manager, the name to display could also be specified in the configuration parameter `VDI_EDITOR_DESKTOP_MANAGER_NAME_desktopManagerId`. If omitted it will be equal to the id.

For each desktop manager, the path to the `xstartup` file must be specified in the configuration parameter `VDI_EDITOR_DESKTOP_MANAGER_XSTARTUP_desktopManagerId`. Interactive plugin already provides a set of preconfigured `xstartup` files for supported desktop managers under `${EF_ROOT}/plugins/interactive/conf/` directory.

Example:

```
VDI_EDITOR_DESKTOP_MANAGERS=gnome,kde
VDI_EDITOR_DESKTOP_MANAGER_NAME_gnome=GNOME
VDI_EDITOR_DESKTOP_MANAGER_XSTARTUP_gnome=${EF_ROOT}/path_to_gnome_xstartup
VDI_EDITOR_DESKTOP_MANAGER_XSTARTUP_kde=/path_to_kde_xstartup
```

Log files

Main VDI log file is located under EnginFrame log directory, `${EF_LOGDIR}/vdi.log`, where `EF_LOGDIR` is `${EF_TOP}/logs/<hostname>`.

VDI Plugin Directory Structure

This section describes the directory structure.

VDI plugin is installed in `${EF_ROOT}/plugins/vdi`.

These are the most important contents of the folder:

```
vdi/
|-- WEBAPP
|-- bin
|-- conf
|   |-- authorization.xconf
|   |-- log.xconf
|   |-- service-manager.efconf
|   |-- interactive.editor.efconf
|   `-- vdi.conf
|-- etc
|-- lib
`-- templates
```

VDI follows the conventional EnginFrame plug-in structure and in particular the following directories contain VDI system files:

- WEBAPP - top level service definition files and web resources.

- `bin` - scripts and executables.
- `conf` - configuration files.
- `etc` - VDI plugin metadata and EnginFrame descriptor files.
- `lib` - internal files used by VDI plugin: XML support services and XSL files.
- `templates` - Interactive services templates.

Interactive services are installed in `$EF_TOP/data/plugins/vdi/services`.

These are the important contents of the folder:

```
services/
|-- catalog
|-- published
`-- extra
```

The following directories contain VDI plugin services files:

- `catalog` - root folder for unpublished services
- `published` - root folder for published services
- `extra` - root folder for custom extra services to be included in the Views portal.

Applications Administration

Applications plugin provides a front-end portal that gives EnginFrame administrators an easy way to create, publish and manage batch and interactive services. End-users instead are provided with a portal to easily access the company HPC services.

This section explains the configuration files and settings of the Applications plugin.



Important

The service examples provided by the Applications portal make use of a `JOB_WORKING_DIR` and assume it is mounted by both EnginFrame hosts and execution hosts.

By default the `JOB_WORKING_DIR` is set to the `EF_SPOOLER` directory of the submitted service, so in order to use the examples the root spoolers directory should be shared with the execution hosts.

This is not a requirement of EnginFrame Applications portal but a simplification used by the examples.

Configuration Files

Most of the times the values of the settings collected during the Applications plugin installation provide all the information necessary to have a working setup.

However the administrator may have the need to change the folders where services files are stored or to change other settings of the system.

All the Applications Plugin configuration files are located in the `$EF_TOP/<VERSION>/enginframe/plugins/applications/conf` subdirectory.



Important

As for the other EnginFrame plugins the correct way to change default configuration is to copy the target configuration file under the `$EF_TOP/conf`, into the `$EF_TOP/conf/plugins/applications` directory, if it doesn't already exist, and edit the copied file.



Note

Configuration parameters are automatically loaded upon saving. No need to restart EnginFrame or logout.

`applications.conf`

This file contains Applications's main default configuration parameters.

Users Access Parameters

`APPLICATIONS_ALLOW_ALL_USERS`

- value: *required*
- default: *true*

Enables Applications portal access to all the users able to log into the system. If `true`, the users will be added to the Applications default group at login time.

Available values:

- `true` - all user are allowed to access to the Applications portal
- `false` - Applications portal users should be explicitly added or imported by an Applications administrator

Example:

```
APPLICATIONS_ALLOW_ALL_USERS=true
```

`service-manager.efconf`

This file contains Applications plugin configuration parameters useful for service management.

General Parameters

APPLICATIONS_SERVICES_ROOT

- value: *required*
- default: `${EF_DATA_ROOT}/plugins/applications/services` where `EF_DATA_ROOT` is `$EF_TOP/data`

Sets the root folder where services files are stored. A change to this value requires changes also to

- `sdf-tree` URL value inside `href` attribute of `xi:include` tags declared in *applications.xml*, *applications.admin.xml* XML files
- `load-conf` value of `ef:action` tags in the services XML files

Example:

```
APPLICATIONS_SERVICES_ROOT=${EF_DATA_ROOT}/plugins/applications/services
```

SM_TEMPLATES_ROOT

- value: *required*
- default: `${EF_ROOT}/plugins/applications/templates`

Sets the root folder where service templates files are stored.

Example:

```
SM_TEMPLATES_ROOT=${EF_ROOT}/plugins/applications/templates
```

SM_PUBLISHED

- value: *required*
- default: `${APPLICATIONS_SERVICES_ROOT}/published`

Sets the root folder where Applications stores files for published services.

Example:

```
SM_PUBLISHED=${APPLICATIONS_SERVICES_ROOT}/published
```

Batch Services Parameters

SM_CATALOG_BATCH

- value: *required*
- default: `${APPLICATIONS_SERVICES_ROOT}/catalog/batch`

Sets the folder where unpublished batch services files are stored.

Example:

```
SM_CATALOG_BATCH=${APPLICATIONS_SERVICES_ROOT}/catalog/batch
```

Interactive Services Parameters

SM_CATALOG_INTERACTIVE

- value: *required*
- default: `${APPLICATIONS_SERVICES_ROOT}/catalog/interactive`

Sets the folder where unpublished interactive services files are stored.

Example:

```
SM_CATALOG_INTERACTIVE=${APPLICATIONS_SERVICES_ROOT}/catalog/interactive
```

interactive.editor.efconf

This file contains configuration parameters for the interactive service editor in the Applications portal.

The file syntax and parameters is the same of the homonymous file of the VDI plugin. Please refer to the section called “`interactive.editor.efconf`” of VDI plugin for the description of this configuration file.

Log files

Main Applications log file is located under EnginFrame log directory: `${EF_LOGDIR}/applications.log`, where `EF_LOGDIR` is `$EF_TOP/logs/<hostname>`.

Applications Directory Structure

This section describes the directory structure.

Applications is installed in `${EF_ROOT}/plugins/applications`.

These are the most important contents of the folder:

```
applications/
|-- WEBAPP
|-- bin
|-- conf
|   |-- authorization.xconf
|   |-- log.xconf
|   |-- service-manager.efconf
|   |-- interactive.editor.efconf
|   `-- applications.conf
|-- etc
|-- lib
`-- templates
```

Applications follows the conventional EnginFrame plug-in structure and in particular the following directories contain Applications system files:

- WEBAPP - top level service definition files and web resources.
- bin - scripts and executables.
- conf - configuration files.
- etc - Applications plugin metadata and EnginFrame descriptor files.
- lib - internal files used by Applications plugin: XML support services and XSL files.
- templates - Services templates.

By default both batch and interactive services are installed in `$EF_TOP/data/plugins/applications/services`.

These are the most important contents of the folder:

```
services/  
|-- catalog  
|   |-- batch  
|   |-- interactive  
|-- published  
|-- extra
```

The following directories contain Applications services files:

- catalog - root folder for batch and interactive unpublished services
- published - root folder for published services
- extra - root folder for custom extra services to be included in the Applications portal.

聚辰科技(CAX-CLOUD)
info@cax-cloud.com

Managing Spoolers

This chapter illustrates the basic concepts concerning EnginFrame spoolers and their management.

A spooler is a dedicated data container created by EnginFrame to host files provided by users (e.g. input files uploaded using a Web browser) or generated by the services (e.g. output or temporary files).

Every service execution (unless explicitly configured) triggers EnginFrame to create a new spooler with appropriate user permissions allowing services to read from and write to spooler's directory.

EF_SPOOLER_DIR is organized per-user. Under EF_SPOOLER_DIR, EnginFrame creates a directory for each user the first time the system is accessed. These directories are named with the user's names. Under each <username> directory EnginFrame creates its spoolers, one for each service execution.

The picture below illustrates EnginFrame's spooler directory structure.

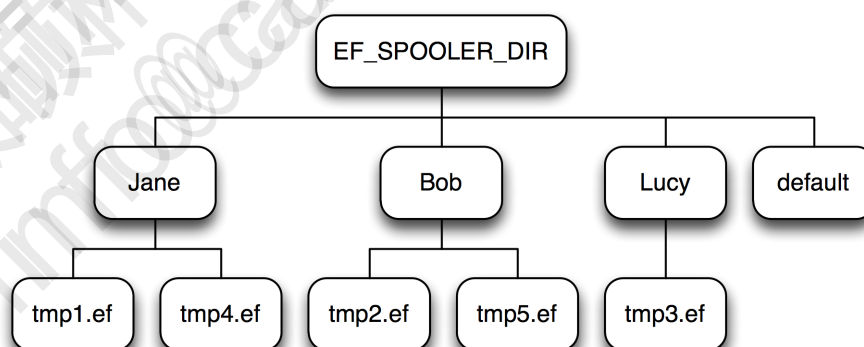


Figure 7.1. Spoolers Directory Structure

Jane, Bob, Lucy are per-user directories and tmp1.ef, tmp2.ef, tmp3.ef, tmp4.ef, tmp5.ef are spooler directory names EnginFrame creates dynamically.

Spoolers Requirements

Spoolers used by agents running on hosts different from the server's one must reside on a *shared file system* that must also be mounted from the EnginFrame Server's host. This shared area has to be readable/writable by both components. EnginFrame provides a mapping mechanism enabling these file-systems to be mounted with different paths on server and agent hosts.



Agent Mount Point

Please note that if you are using multiple agents the same mount point path must be used for all of them.

EF_SPOOLER_DIR directory should be owned by the user running EnginFrame Server (e.g. `efnobody`). `efnobody` must have read/write permissions since EnginFrame Server initially creates spoolers for users. All other users must be able to traverse, i.e. read and execute permissions, spooler root directory. Summarizing, spooler root directory should have the following ownerships and permissions:

```
rwX r-x r-x  efnobody:efnogroup
```

where `efnogroup` is `efnobody`'s primary group.

The spoolers area must be placed on a file-system where agent's owner must have complete read/write privileges. Depending on who runs agent daemon, this can be achieved by:

- Avoiding *root squashing* on NFS server when agent owner is *root*.
- Using same user running server when agent owner is a *normal user*.



EnginFrame Professional

Please note that distinct EnginFrame deployments *should not share* the same Spooler area.

Spooler Security Permissions

EnginFrame Server starts with user file-creation mask, i.e. **umask**, set to 022 meaning that:

- Files are created with 644 `rw-r--r--` permissions.
- Directories are created with 755 `rwXr-xr-x` permissions.

This assures EnginFrame Server user can read/write files from/into spoolers. This might lead to a weak security environment since everybody could read files from spoolers owned by other user accounts. For this reason EnginFrame Agent changes spooler permission to 750 `rwXr-x---` just before service execution. Final spooler permissions are thus read/write/execute for spooler's owner and read/execute for EnginFrame Server's owner. This occurs because user `efnobody` with group `efnogroup` creates the spooler. EnginFrame Agent changes spooler's owner executing `chown <username> <spooler>` and spooler ownership becomes `<username>:efnogroup`. This

allows spooler's owner to perform whatever action is needed inside spooler and that just `efnogroup` members are allowed to read data inside spoolers. A security best practice is to have only `efnobody` as `efnogroup` group's member.

禁发科技(CAX-Cloud)
info@cax-cloud.com

Configuring EnginFrame Spoolers

This section describes how to customize basic spoolers settings concerning directory paths and file download aspects.

Configuring Spoolers Default Root Directory

`EF_SPOOLER_DIR` directory is chosen during installation. All spoolers are created by EnginFrame under this directory. The default value is `$EF_TOP/spoolers`.

Spoolers root directory can be changed by editing `EF_SPOOLERDIR` parameter inside `$EF_TOP/conf/enginframe.conf`. Please note that the startup parameter name `EF_SPOOLERDIR` is slightly different from the property `EF_SPOOLER_DIR` that is set and used inside EnginFrame.

```
EF_SPOOLERDIR=/mnt/scratch/spoolers
```

Example 7.1. Change Spoolers Default Root Directory

After this change all spoolers are created under `/mnt/scratch/spoolers` directory.



Note

Changes to this parameter require an EnginFrame Server restart.



Note

Spooler location is defined at *creation time*, so changing `EF_SPOOLERDIR` does not affect existing spooler's location. EnginFrame Agent retrieves old spoolers from the old location and new spoolers from the new location.

Refer to the section called “Spoolers Requirements” when setting up a new spooler root directory.

Download Files From Spoolers

File are downloaded from spoolers on user's behalf involving both server and agent. EnginFrame Server receives a download request forwarding it to the agent that actually accesses the file. The agent posts the file back to server via HTTP(S).

This process implies EnginFrame Agent is able to connect back to server via HTTP(S) for sending data. So network and firewall configurations should be carefully considered.



Note

If EnginFrame Server is configured to accept requests through *HTTP over SSL* - HTTPS - protocol, then refer to [Chapter 13, Configuring HTTPS](#) to properly configure server and agent.

It could be important to set a *mime-type* for downloaded files to let browsers correctly identify the file's type. You can customize and configure the mime-type EnginFrame should use when

downloading files. the section called “Managing Internet Media Types” explains you how to setup mime-type configurations.

Configure Download URL on Agent

The download process highlights how an agent has to connect back to the server for sending downloaded data. Usually the server HTTP endpoint, i.e. host and port, to which agent should connect is automatically detected from server's request.

There might be network configurations or architectural scenarios, e.g. web access is configured with an HTTP server in front of EnginFrame Server, for which agent should use a different host and port to connect to server. In this case you have to explicitly configure the complete URL to which agent has to connect to EnginFrame Server.

The `ef.download.server.url` parameter inside `$EF_TOP/conf/enginframe/agent.conf` (or `$EF_TOP/<VERSION>/enginframe/conf/agent.conf`) sets the URL agent uses to connect to server. It is defined as:

```
ef.download.server.url=http[s]://<host>:<port>/<web-context>/download
```

where:

- `host` and `port` identifies EnginFrame Server network endpoint.
- `web-context` is EnginFrame's root context chosen during installation; it defaults to `enginframe`.

Example:

```
ef.download.server.url=http://localhost:8080/enginframe/download
```

Configure Streaming Download Timeout

Streaming downloads have a timeout. If the file being streamed does not change during this interval, EnginFrame interrupts its stream to client. Its default value is 300 seconds.

This value is specified in `$EF_TOP/conf/enginframe/server.conf` (or `$EF_TOP/<VERSION>/enginframe/conf/server.conf`). You change this value by editing `ef.download.stream.inactivity.timeout` property inside `$EF_TOP/conf/enginframe/server.conf`. The value is expressed in seconds.

Example:

```
ef.download.stream.inactivity.timeout=600
```

to set a 10 minutes timeout.

Configure Streaming Download Sleep Time

EnginFrame's file streaming download feature works using a pull model. The server periodically queries agent for available data.

The interval between two subsequent checks can be configured. Its default value is 5 seconds. Consider that a lower interval might improve user experience while increasing system load.

This value is specified in `$EF_TOP/conf/enginframe/server.conf` (or `$EF_TOP/<VERSION>/enginframe/conf/server.conf`). You change this value by editing `ef.download.stream.sleep.time` parameter inside `$EF_TOP/conf/enginframe/server.conf`. The value is expressed in seconds.

Example:

```
ef.download.stream.sleep.time=20
```

to set a 20 second sleep between two subsequent checks.

Spooler Life Cycle

This section outlines EnginFrame's spooler lifecycle. Besides outlining spooler's lifecycle, each subsection describes common customizations you can apply to EnginFrame Portal.

Overview

Spoolers are created for each service submission. More precisely spoolers are created for all those services whose spooler definition has a TTL different from `-1`.

Spoolers are initially created by EnginFrame Server that also associates its defined time-to-live. Together with spooler directory the server also creates an entry in EnginFrame's spooler database called *repository*. The repository is file-system based and each entry is a file that contains all information necessary to recreate a spooler, for example:

- The owner
- The physical location path on both server and agent
- The display name
- And other properties

The server also has the responsibility to save user's files into a spooler before contacting agent for service execution. After the spooler setup tasks have been accomplished the server contacts agent for service execution. The Agent first changes spooler's and its contents ownership and then uses this spooler as execution working directory.

EnginFrame removes a spooler when its life-time expires, deleting spooler's directory with its content and its repository entry. An EnginFrame thread called *reaper* periodically checks if there are expired spoolers ready to be removed.

Change Repository Location

The default EnginFrame repository path is `$EF_TOP/repository`. EnginFrame gives you the flexibility to change the location where repository entries should be stored.

For example you may want to save repository entries on a high speed/reliability file-system or on an area where to keep "dynamic" file-system paths, i.e. directory whose contents change often, in order to conform to your company's policies.

The repository location is configured by `EF_REPOSITORYDIR` parameter inside

`$EF_TOP/conf/enginframe.conf`. It specifies an absolute path in the file-system. You can use other variables defined in `enginframe.conf` for path definition.



Note

Changes to `EF_REPOSITORYDIR` require EnginFrame Server restart.

Some configuration examples

```
EF_REPOSITORYDIR=$EF_TOP/repository
```

defines repository location using `$EF_TOP` variable.

```
EF_REPOSITORYDIR=/mnt/nas/ef-repository
```

sets an absolute path for repository directory.

Configure Reaper Sleep Time

As explained in [the section called “Overview”](#) the EnginFrame reaper is a thread that periodically wakes-up to check if there are expired spoolers in the system needing cleanup.

You can configure reaper's thread sleep interval determining how frequently this check should be done.

This value is specified in `$EF_TOP/conf/enginframe/server.conf` (or `$EF_TOP/<VERSION>/enginframe/conf/server.conf`). You change this value by editing `ef.reaper.sleep.time` property inside `$EF_TOP/conf/enginframe/server.conf`. The value is expressed in minutes. The default value is 30 minutes.

Example:

```
ef.reaper.sleep.time=60
```

to set one hour sleep between each thread's reap.

Spoolers Removal: Dead Spoolers

If for any reason spooler cleanup fails, spooler's directory is renamed prepending `DEAD_` prefix to its original name.

For example if EnginFrame is unable to remove spooler `tmp32062.ef`, it is renamed to `DEAD_tmp32062.ef`. Dead spoolers can be safely removed from your system.

If you have setup deadspooler logging target (refer to [the section called “Fine Tune Logging”](#)) once a dead spooler has been created, the event is logged into `$EF_TOP/logs/DEAD_spoolers.{agent|server}.log` depending on which side, server or agent, the error occurred.

聚辰科技(CAX-CLOUD)
info@cax-cloud.com

Managing Sessions Directory

This chapter illustrates the basic concepts concerning EnginFrame interactive session data and their management.

An interactive session data is a dedicated data container created by EnginFrame to host files required for the interactive session lifecycle, such as the thumbnail of the screenshot.

INTERACTIVE_SHARED_ROOT is organized in the same way as the EnginFrame spoolers, see [Chapter 7, Managing Spoolers \[105\]](#). Under INTERACTIVE_SHARED_ROOT, EnginFrame creates a directory for each user the first time an interactive session is created. These directories are named with the user's names. Under each <username> directory EnginFrame creates its session data directory, one for each interactive session.

Sessions Requirements

Sessions must reside on a *shared file system* that must be mounted from the EnginFrame Server's host, EnginFrame Agent's host and visualization nodes. This area may not require to be shared when submitting sessions to some Distributed Resource Managers, see [the section called “Shared File System Requirements”](#) for more info.

The INTERACTIVE_SHARED_ROOT directory must be owned by the user running EnginFrame Server (e.g. efnobody) and by efnobody's primary group. It must have permissions 3777. Summarizing it must have the following permissions:

```
d rwx rws rwt efnobody:efnogroup
```

where efnogroup is efnobody's primary group.

The EnginFrame installer takes care of creating this directory with the proper permissions. The default location is \$EF_TOP/sessions. This directory can be configured changing the INTERACTIVE_SHARED_ROOT value in \$EF_TOP/conf/plugins/interactive/interactive.efconf.

This shared area has to be readable/writable by EnginFrame nodes and visualization nodes. EnginFrame provides a mapping mechanism enabling these file-systems to be mounted with different paths on EnginFrame and visualization hosts. This configuration is specific to the Distributed Resource Manager used for the session and can be configured in the specific plugin configuration file:

- On OpenLava, use LSF_INTERACTIVE_SHARED_ROOT_EXEC_HOST in \$EF_TOP/conf/plugins/lsf/ef.lsf.conf
- On Moab®, use MOABWS_INTERACTIVE_SHARED_ROOT_EXEC_HOST in \$EF_TOP/conf/plugins/moabws/moabws.efconf
- On PBS Professional®, use PBS_INTERACTIVE_SHARED_ROOT_EXEC_HOST in \$EF_TOP/conf/plugins/pbs/ef.pbs.conf
- On Torque, use TORQUE_INTERACTIVE_SHARED_ROOT_EXEC_HOST in \$EF_TOP/conf/plugins/torque/ef.torque.conf
- On SLURM™, use SLURM_INTERACTIVE_SHARED_ROOT_EXEC_HOST in \$EF_TOP/conf/plugins/slurm/ef.slurm.conf
- On SGE, use SGE_INTERACTIVE_SHARED_ROOT_EXEC_HOST in \$EF_TOP/conf/plugins/sge/ef.sge.conf
- This configuration does not apply to Neutro

Customizing Logging

Logging is an integral component to any software development project. During the development stages it offers a valuable source of debugging information for the developer. During deployment it can provide valuable operational data that allows administrators to diagnose problems as they arise.

Tomcat® Logging

EnginFrame is shipped with Apache Tomcat® servlet container. Tomcat® log files are the first source of information concerning EnginFrame's status. Log files are located on EnginFrame Server's host under `$EF_TOP/logs/<HOSTNAME>/tomcat`.

These are the most interesting log files:

- `catalina.out`
Contains Tomcat®'s Java™ process standard output and standard error. Tomcat® startup and shutdown messages are written here.
- `catalina.[yyyy]-[mm]-[dd].log`
Contains Tomcat®'s and its libraries logging information on a day-by-day basis.
- `localhost_access_log.[yyyy]-[mm]-[dd].txt`
Contains all web accesses on a day-by-day basis. Any resource served by Tomcat is logged here with information about the amount of transferred data. Also HTTP status codes like 404 Not Found, 403 Forbidden are logged here.
- `enginframe.[yyyy]-[mm]-[dd].log`
Contains Tomcat®'s error logs about EnginFrame Portal not caught by EnginFrame itself.

The standard Tomcat® configuration fits the most common installations. In case of specific needs you can modify the default `$EF_TOP/<VERSION>/enginframe/conf/logging.properties` configuration. Refer to the official [Tomcat® documentation](#) for more details.

EnginFrame Server and Agent Logging

Configuration Files

The default logging configurations are located under `$EF_TOP/<VERSION>/enginframe/conf`:

- `log.server.xconf`
Contains server's configuration.
- `log.agent.xconf`
Contains agent's configuration.

Both are XML files with the same syntax.

In case you need to modify the defaults, you can specify a new configuration in the `log.server.xconf` and `log.agent.xconf` files under the configuration directory `$EF_TOP/conf/enginframe`.

In these files you can configure the location where to store log files, their verbosity level, rotation policies. By default EnginFrame is configured to write only warning and error messages and to keep a history of 4 log files with a 10 MB maximum size.

The log's default location is under `$EF_TOP/logs/<HOSTNAME>` on server and agent hosts:

- `ef.log`
Contains server's logging. It also contains the local agent's logs. It is located on server's host.
- `agent.remote.stdout`
Contains agent's process standard output. It is located on agent's host.
- `agent.remote.stderr`
Contains agent's process standard error. It is located on agent's host.
- `agent.remote.log`
Contains agent's process logging. It is located on agent's host.

If you need to quickly switch the log configuration to produce a verbose logging, you can use `log.server.verbose.xconf` and `log.agent.verbose.xconf` files located in `$EF_TOP/<VERSION>/enginframe/conf`.

You just need to backup the previous `log.server.xconf` and `log.agent.xconf` files and replace them with these verbose configurations.

It is not recommended to use a verbose logging configuration when you are in a production environment. The performance impact of logging can be significant, depending on the logging threshold that is configured and especially if a large number of users are accessing the portal.

Change Log Files Location

The EnginFrame logging system gives you complete flexibility on where to store log files. For example you may want to move log files from their default location to store them on a high speed file-system or to conform to your company's policies.

The location configuration is done in the `<targets>` section by setting the `<filename>` tag. The text inside this tag represents the path to the log file.

For example this XML text sets `${EF_LOGDIR}/ef.log` for the core components:

```
<targets>
  <enginframe id="core">
    <filename>${EF_LOGDIR}/ef.log</filename>
    ...
```

You can also decide whether to append or overwrite existing files by using the `<append>` tag.

In the following example the log file will be overwritten on every server restart:

```
<targets>
  <enginframe id="core">
    <filename>${EF_LOGDIR}/ef.log</filename>
    <append>false</append>
    ...
```

Change Log Files Size and Rotation Policy

Each EnginFrame Server and EnginFrame Agent is configured to write log files up to 10 MB and then to create another file to a maximum of 4 files. When a new log file is written the oldest one is deleted. This configuration guarantees that each server and agent uses no more than 40 MB of disk space for log files. You may want to change this rotation policy for example because you need more history. A change may also be helpful if you have increased log verbosity.

Changing the `<rotation>` tag inside the `<targets>` section configures the rotation policy. The `max` attribute defines the maximum number of files, while the `<size>` tag contains each file's size.

For example the following XML text defines a rotation of 5 files each one of 5 MB:

```
<rotation type="revolving" max="5">
  <size>5m</size>
</rotation>
```

Change Log Level

EnginFrame logging allows to have a fine grain control over which statements are printed. In a development environment you may wish to enable all logging statements while in a production environment it is suggested to disable debug messages to avoid performance issues. You can configure this behavior by setting the appropriate *log level*. A *log level* describes the urgency of a message. Below is a list of log levels that are usable within the EnginFrame logging system:

- **NONE**: No messages are emitted.
- **DEBUG**: Developer oriented messages, usually used during development of the product.
- **INFO**: Useful information messages such as state changes, client connection, user login etc.
- **WARN**: A problem or conflict has occurred but it may be recoverable, then again it could be the start of the system failing.
- **ERROR**: A problem has occurred but it is not fatal. The system still functions.
- **FATAL_ERROR**: Something caused whole system to fail. This indicates that an administrator should restart the system and try to fix the problem that caused the failure.

Each logger instance is associated with a log level. This allows you to limit each logger so that it only displays messages greater than a certain level. So if a **DEBUG** message occurred and the logger's log level was **WARN**, the message would be suppressed.

Changing the `log-level` attribute of a `<category>` tag sets log verbosity for the associated category.

For example this XML tag defines a default category whose log-level is **INFO**:

```
<category name="" log-level="INFO">
  <log-target id-ref="core"/>
</category>
```

Fine Tune Logging

Define New Categories and Targets

In a complex system it is often not enough to suppress logging based on log-level. For instance you may wish to log the network subsystem with **DEBUG** log-level while the simulator subsystem with **WARN** log-level. To accomplish this EnginFrame uses *categories*. Each *category* is a name, made up of name components separated by a `.`. So a category named `"network.interceptor.connected"` is made up of three name components `"network"`, `"interceptor"` and `"connected"`, ordered from left to right. Every logger is associated with a category at creation. The left-most name component is the most generic category while the right-most name component is the most specific. So `"network.interceptor.connected"` is a child category of `"network.interceptor"`, which is in turn a child category of `"network"`. There is also a root category `""` that is hidden. The main reason for structuring logging namespace in a hierarchical manner is to allow inheritance. A logger will inherit its parent log-level if it has not been explicitly set. This allows you to set the `"network"` logger to have **INFO** log-level and unless the `"network.interceptor"` has had its log-level set it will inherit the **INFO** log-level.

Categories send messages to log targets. Decoupling log message generation from handling allows developers to change destinations of log messages dynamically or via configuration files. Possible destinations include writing to a database, a file, an IRC channel, a syslog server, an instant messaging client, etc.

Adding a `<category>` tag inside the `<categories>` section defines a new category. You must specify its name and `log-target` to which log messages are written. The name of the category acts like a filter: all messages matching the category name are sent to the specified log-target.

For example if you want more information from EnginFrame's download component you can use the category named `com.enginframe.server.download`:

```
<category name="com.enginframe.server.download" log-level="DEBUG">
  <log-target id-ref="core"/>
</category>
```

This configuration sends all download messages to the core target.

Another useful category is the `deadspooler`. The `deadspooler` category is used by EnginFrame to write messages concerning spoolers that could not be deleted and were renamed as DEAD. NICE support team recommends to turn this feature on. There is no overhead and it could be very useful to know why an EnginFrame spooler could not be deleted. This category is activated by setting its log-level to INFO:

```
<category name="deadspooler" log-level="INFO">
  <log-target id-ref="deadspooler"/>
</category>
```

Refer to *EnginFrame Administrator's Reference* for a complete list of categories EnginFrame uses.

Change Message Format

Log targets that write to a serial or unstructured store (i.e., file-system or network based targets) need some method to serialize the log message before writing to the store. The most common way to serialize the log message is to use a `formatter`.

The format specified consists of a string containing raw text combined with pattern elements. Each pattern element has the generalized form:

```
%[+|-]#.#{field:subformat}
```

- `+|-` indicates whether the pattern element should be left or right justified (defaults to left justified if unspecified.)
- `#.#` indicates the minimum and maximum size of output, if unspecified the output is neither padded nor truncated.
- `field` indicates the field to be written and must be one of `category`, `context`, `user`, `message`, `time`, `rtime` (time relative to start of application), `throwable` or `priority`. This parameter is mandatory.
- `subformat` specifies which piece of field is interesting for log messages.

Use the `<format>` tag to set log message printing.

For example the following code shows format setting for the `core` target:

```
<enginframe id="core">
  <filename>${EF_LOGDIR}/ef.log</filename>
  <format type="enginframe">
    %7.7{priority} %5.5{rttime} [%8.8{category}]:%{message}\n%{throwable}
  </format>
  ...
```

A number of examples for format and actual output follows.

- Example

Format:

```
%7.7{priority} %5.5{rttime} [%8.8{category}]:%{message}\n%{throwable}
```

Output:

```
DEBUG    123    [network.]: This is a debug message
```

- Example

Format:

```
%7.7{priority} %5.5{rttime} [%{category}]:%{message}\n
```

Output:

```
DEBUG    123    [network.interceptor.connected]: This is a debug message
DEBUG    123    [network]: This is another debug message
```

- Example

Format:

```
%7.7{priority} %5.5{rttime} [%10.{category}]:%{message}\n
```

Output:

```
DEBUG    123    [network.interceptor.connected]: This is a debug message
DEBUG    123    [network      ]: This is another debug message
```

EnginFrame Scriptlet Logging

EnginFrame modularity lets developers add new features by deploying their plugins. If your plugin uses scriptlets, you have the same logging support on which EnginFrame is based. As an administrator you must know how to set up scriptlet logging for both development and production environments.

Creating `$EF_TOP/<VERSION>/enginframe/plugins/<myplugin>/conf/log.xconf` or `$EF_TOP/conf/plugins/<myplugin>/log.xconf` on EnginFrame Server's host enables scriptlet logging. This file has the same syntax as the ones shipped by EnginFrame under `$EF_TOP/<VERSION>/enginframe/conf`, so all the information in [the section called “EnginFrame Server and Agent Logging”](#) applies here too.

The scriptlet code can emit logging messages with any category and log level.

This is an example configuration file:

```
<?xml version="1.0"?>

<logkit>
  <factories>
    <factory
      type="enginframe"
      class="com.enginframe.common.utils.log.EnginFrameTargetFactory"/>
    </factories>

  <targets>
    <enginframe id="plugin">
      <filename>
        ${EF_LOGDIR}/myplugin.log
      </filename>
      <format type="enginframe">
        %7.7{priority} %5.5{rttime} [%{category}]:%{message}\n
      </format>
    </enginframe>
  </targets>

  <categories>
    <category name="myplugin" log-level="DEBUG">
      <log-target id-ref="plugin"/>
    </category>
    <category name="" log-level="WARN">
      <log-target id-ref="plugin"/>
    </category>
  </categories>
</logkit>
```

`${EF_LOGDIR}/<myplugin>.log` contains the log messages for this configuration. It exposes only two categories: *myplugin* and the *root one* (it is the one that has an empty name.) If your code uses *myplugin* category, then it logs DEBUG messages; if your code uses a category that is not defined, then it logs WARN messages.

If the plugin `log.xconf` configuration files do not exist, then EnginFrame defaults to the core `log.server.xconf` to define *myplugin*'s logging categories.

聚辰科技(CAX-CLOUD)
info@cax-cloud.com

EnginFrame Licenses

This chapter describes how EnginFrame manages its licenses, where license files are located, the meaning of license fields, how license tokens are counted, and how to check EnginFrame's license token usage.

License Files Management

EnginFrame loads its license files from `$EF_TOP/license`. All files ending with `.ef` extension are loaded.



Replace Old License

If you want to replace an EnginFrame license keeping the old one, you should rename the license changing the `.ef` extension to something else, e.g. appending `.OLD` extension to the original file. Otherwise EnginFrame would also load the old license. Having two license for the same EnginFrame component leads to a conflict issue

The EnginFrame license management system reads licenses dynamically from the file-system and it is able to detect any changes you may apply to licenses even to recognize if license files were added or removed.

Configuring License Files Location

As mentioned above `$EF_TOP/license` is the default directory where EnginFrame loads licenses from. This directory can be changed making EnginFrame load licenses from another location in your file-system.

Modifying `EF_LICENSE_PATH` parameter inside `$EF_TOP/conf/enginframe/server.conf` and defining an absolute file-system path changes the directory EnginFrame uses to load licenses.

Example:

```
EF_LICENSE_PATH=/mnt/server/ef-licenses
```

License File Format

An EnginFrame license is an XML file describing one or more EnginFrame licensed components or plugins. EnginFrame's main component, the one that activates core functionalities, is EF Base.

The license fields are:

- **product**: what is being licensed, e.g. EnginFrame PRO
- **release**: EnginFrame's major release number, e.g. 2019.0
- **format**: license file format, e.g. 2.0
- **component**: component being licensed, example of components are EF Base, webservices, etc.
- **expiration**: license expiration date. The value never means a perpetual license.
- **ip**: licensed IP addresses where EnginFrame can be deployed. It can be a single host IP, a range of IPs, or a list of IP numbers. The license is valid if EnginFrame is running on one of the mentioned hosts.
- **type**: one of DEMO, YEAR, or FULL.
- **units**: number of license tokens.
- **units-per-user**: number of tokens EnginFrame locks for each concurrent user.
- **license-hosts**: if true EnginFrame locks a token for each host in the underlying grid infrastructure.
- **hosts-preemption**: if true EnginFrame releases tokens grabbed by hosts for new users accessing the system. This happens when all tokens have been used.
- **signature**: license signature accounting all the license fields values.

An EnginFrame license example:

```
<?xml version="1.0"?>
<ef-licenses>
  <ef-license-group product="EnginFrame PRO" release="2017.0" format="2.0">
    <ef-license
      component="EF Base"
      vendor="NICE"
      expiration="2017-08-15"
      ip="80.20.156.116"
      licensee="RnD Team"
      type="DEMO"
      units="20"
      units-per-user="1"
      license-hosts="true"
      hosts-preemption="true"
      signature="..." <!-- Omitted for space -->
    />
  </ef-license-group>
</ef-licenses>
```


License Checking

The most important license fields are `component`, `ip`, `expiration`, and `units`. The first three fields can be statically verified and basically depend on your EnginFrame deployment setup:

- `Component` unlocks core (value `EF Base`) but you must one for each licensed component you have installed, e.g. `webservices`.

Send questions to [<helpdesk@nice-software.com>](mailto:helpdesk@nice-software.com) for your requirements and to get help on understanding which components you actually need to satisfy your goals.

- `IP` address depends on EnginFrame's host.
- `Expiration` date states how long your license is valid. Evaluation licenses last one month, nevertheless NICE is flexible enough to release licenses with a life-time that satisfies your evaluation needs. Please express them inside the request form at the download web site or via email at [<helpdesk@nice-software.com>](mailto:helpdesk@nice-software.com).



EnginFrame Server's Host IP Address

You have to determine EnginFrame Server's host IP address for a valid license.

You can verify which IP address is correct for the license request by executing the following command:

```
ping `hostname`
```

which pings the host name returned by `hostname` and displays the same IP address EnginFrame would use.

EnginFrame does not accept loopback IP address, like `127.0.0.1`. If `ping` returned a loopback address, EnginFrame Server's host name resolution has to be properly configured, either editing `/etc/hosts` or changing DNS or NIS or LDAP configurations.

The `units` field expresses the total number of license tokens. A detailed explanation of how EnginFrame counts tokens follows.

License Token Count

As it could be guessed from the section called “License File Format” different scenarios have to be considered when EnginFrame counts license tokens. The scenarios change according to different combinations of some license fields.

License tokens are dynamically consumed by EnginFrame on the base of system usage and load, e.g. the number of concurrent users or the number of hosts in the grid environment accessed through EnginFrame. When EnginFrame consumes a token, it subtracts it from the total number of available tokens expressed by the `units` license field.

Tokens acquired by a user are always released once the user logs out of the system or on his working session expires. Users logging out (or on session expiry) release their tokens. Tokens acquired by hosts are released only on pre-emption (if expressed in the license, for the needed amount) or when EnginFrame is restarted (all tokens are released).

The following scenarios depict how EnginFrame acquires and releases license tokens:

1.

```
license-hosts="true"
hosts-preemption="true"
```

EnginFrame acquires 1 token for each concurrent user accessing the system and 1 token for each host in the underlying computing environment.

Since pre-emption on hosts tokens is switched on, when all the tokens are consumed, further users logging into system are granted access by reclaiming one token from those acquired by hosts. The host whose token has been reclaimed is now unlicensed.

If all tokens are consumed and there are no more tokens to pre-empt, system access is denied.

Available tokens are dynamically acquired by users or hosts.

EnginFrame does not show unlicensed hosts details (kind of host, status, memory consumption, etc.)

2.

```
license-hosts="true"
hosts-preemption="false"
```

As in the previous scenario, EnginFrame acquires 1 token for each concurrent user accessing the system and 1 token for each host in the underlying computing environment.

Differently from the previous scenario, tokens consumed by hosts are never released. Only restarting EnginFrame resets host's license token status.

When all the tokens are consumed the system denies further accesses.

Available tokens are dynamically acquired by users or hosts.

Unlicensed hosts are managed as in scenario 1.

3.

```
license-hosts="false"
units-per-user="<n>"
```

In this scenario EnginFrame acquires n tokens for each concurrent user, where n is a positive integer. No tokens are acquired by hosts meaning they are all licensed.

This kind of license fits all those cases where you deal with big or growing clusters and it is more convenient to have a flat license for the number of hosts. Or you just do not want to bother with license issues concerning hosts.

When all the tokens are consumed (by users) the system denies further accesses. In this case there is no token reclaiming since hosts do not acquire tokens.

List of Licensed Hosts

An optional list of licensed hosts for scenarios 1 and 2 can be created. This list spares license tokens though limiting cluster view through EnginFrame.

Create `license.hostlist` in `EF_LICENSE_PATH` declaring the list of licensed hosts. The file has to contain the host names, one per line. The host names must be reported as the job scheduler does.

A `$EF_TOP/license/license.hostlist` example:

```
host-linux1.nice
host-linux2.nice
host-linux3.nice
host-unix1.nice
host-win1.nice
host-win2.nice
```

Monitoring License Usage

License token consumption is an important aspect for an EnginFrame administrator.

The EnginFrame administration portal provides administrators with installed licenses view and with used license token details. Loaded licenses with their field values are shown as follows:

Details	
Component:	EF Base
Expiration:	2015-12-31
Address:	172.16.10.197,172.16.10.198
Licensee:	TestEnv@NICE RnD
Product:	EnginFrame HPC ENT
Type:	DEMO
Vendor:	NICE
Hosts preemption:	false
License hosts:	false
Units per user:	1

Figure 10.1. EnginFrame License Details

The license token consumption details is also presented:

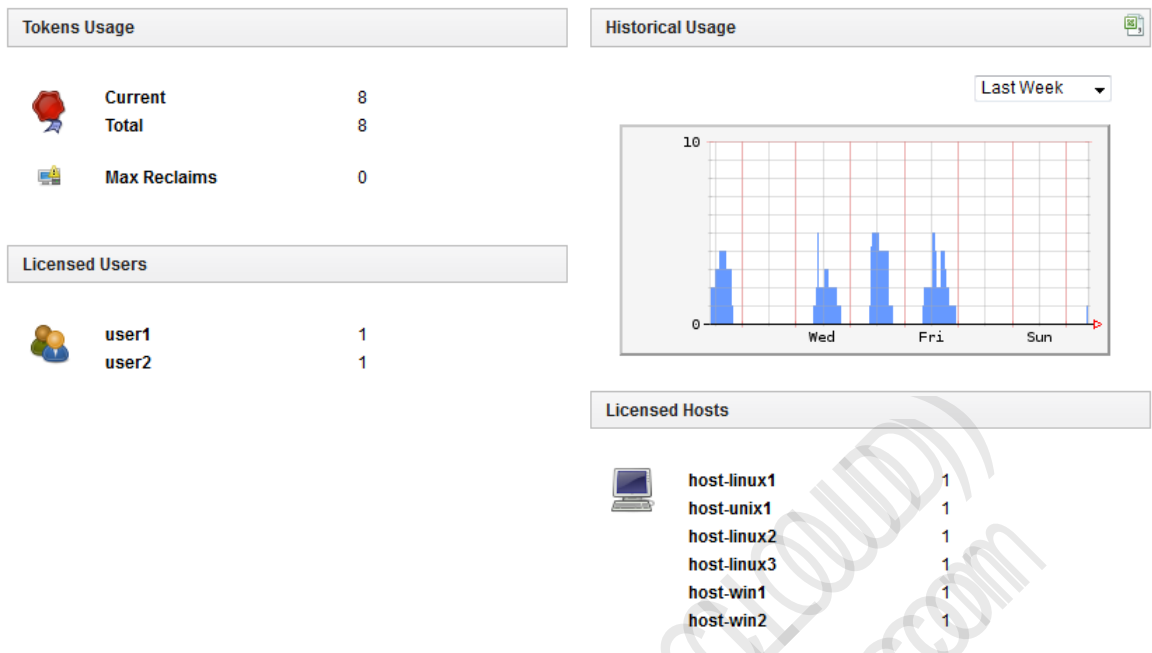



Figure 10.2. EnginFrame License Tokens Status

You can monitor the total number of used tokens, which users have logged in, and which hosts are currently licensed. In the preceding example EnginFrame acquired a license token for all reported users and hosts.



Note

The same user name (but not necessarily the same physical person) connecting from a different workstation or different browser at the same time, consumes an extra token. Tokens are not acquired nominally but per concurrent user.

You can also check if there were token reclaims in the system. The value reported is the max number of reclaimed tokens since EnginFrame's last restart:

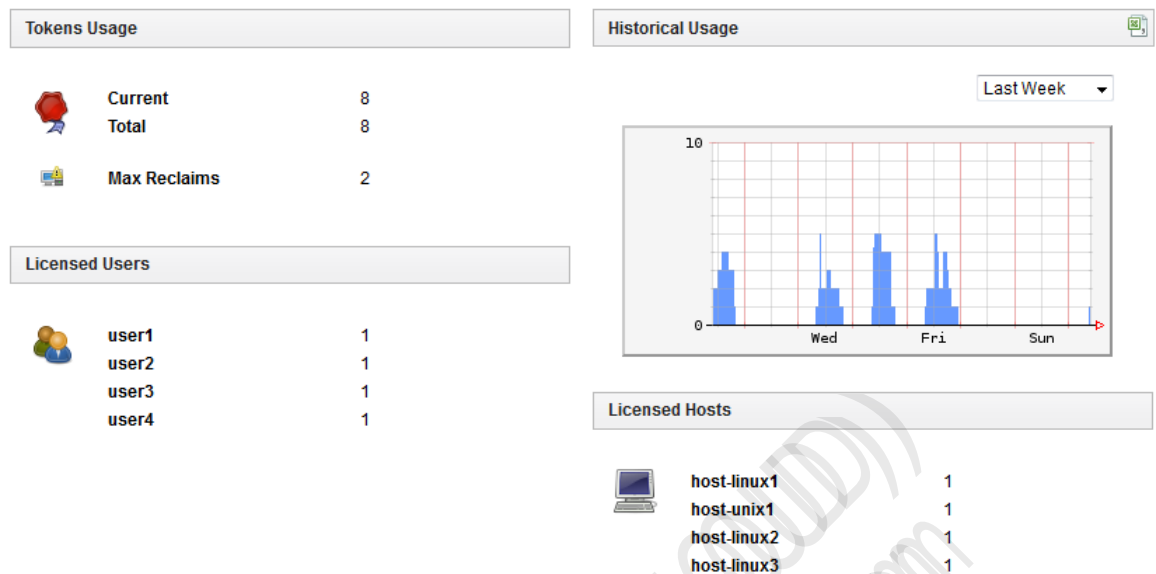


Figure 10.3. EnginFrame License Tokens Status With Reclaims

In the picture above there were two token reclaims. All 8 tokens were consumed by 6 hosts and 2 users. The next two requests from paolo and aware were satisfied but required 2 token reclaims from the hosts.

Enable Debug Log Messages for Licenses

If you are experiencing problems with the EnginFrame license system and you need support from the NICE Support Team or even if you want to check by yourself the details of license management, it is important to rise license management module's log level.

You can enable license module's debug log level by editing `log.server.xconf` and adding the category:

```
<category name="com.enginframe.common.license" log-level="DEBUG">
  <log-target id-ref="core"/>
</category>
```

Otherwise just use `log.server.verbose.xconf` which has this category's log level already set to DEBUG.

Refer to [Chapter 9, Customizing Logging](#) for more details about logging configuration.

聚辰科技(CAX-CLOUD)
info@cax-cloud.com

PART III

Security

聚云科技(CAX-CLOUD)
info@cax-cloud.com

聚辰科技(CAX-CLOUD)
info@cax-cloud.com

Authentication Framework

The EnginFrame authentication framework is extremely flexible and supports out of the box many different mechanisms.

Furthermore, you can easily write your own custom mechanism to authenticate users when standard methods do not meet your company rules.

Standard EnginFrame Authentication Authorities

EnginFrame is shipped with these standard authentication mechanisms

- PAM
- LDAP
- HTTP
- Active Directory
- Certificate

Default Authority

You choose default authentication method used to access services during installation. This value can be changed setting `EF_DEFAULT_AUTHORITY` property in `server.conf`, like in this example:

```
EF_DEFAULT_AUTHORITY=pam
```

Default authentication method is used by all services having `ef:agent's authority` attribute set to `${EF_DEFAULT_AUTHORITY}` as in this example:

```
<ef:agent xmlns:ef="http://www.enginframe.com/2000/EnginFrame"
  id="tutorial"
  authority="${EF_DEFAULT_AUTHORITY}">
```

The authority can also be expressed at `ef:service` level to override the default one defined in the root `ef:agent` tag.

User Mapping

In some deployments users log into the EnginFrame Portal with an username that differs from their user account on the underlying computing environment. EnginFrame supports this situation providing a way to configure *User Mapping*, which allows to *map* a username provided at login time to a username of the underlying operating system.

For instance, the user *John Smith* could log into the EnginFrame Portal using the whole string `John Smith` and submit a job which is executed as the user `jsmith` on the underlying Unix® computing environment.

In some cases user mapping can also be used to map different users of the portal to the same system account or vice versa to map the same user to different accounts on separate systems.



Mapping root Account

You cannot map users to the `root` account.

All authentication modules shipped with EnginFrame support user mapping.

To enable it you need to

- set to `true` the `EFAUTH_USERMAPPING` parameter in the `$EF_TOP/conf/plugins/<authority>/ef.auth.conf` file
- add a script called `ef.user.mapping` in the `EF_ROOT/plugins/<authority>/bin` directory of the authentication module
- set the ownership of file `ef.user.mapping` to `root:root` and its permissions to `755` (`rwxr-xr-x`)

The `ef.user.mapping` script must produce as output the username to which the user being authenticated is mapped. This mechanism provides full custom flexibility to user mapping mechanism.

A trivial example of the `ef.user.mapping` script could be

```
#!/bin/sh
echo "jsmith"
```

which maps all the portal users to the unique `jsmith` user.

A more significative example of user mapping involves reading a file where each line specifies a mapping using the `Login Name=username` format.

For instance let's consider a file named `EF_ROOT/plugins/<authority>/conf/user.mapping` with the following content:

```
# simple mapping file
#
# Syntax: loginname=unixaccount
#

Lucy Johnson=ljohnson
John Smith=jsmith
```

Our `ef.user.mapping` script tries to match the name provided by the user during login with the ones present on the left-hand side of the equal sign in the `user.mapping` file and map it to the corresponding account name:

```
#!/bin/sh

# read login name from command line
_loginname="$1"

# mapping file
_mappingfile=`dirname "$0"`"/../conf/user.mapping"

# transform login name so it can be used by sed in a safe way
_happysed=`echo "${_loginname}" | sed \
-e 's#\|\\|#\|\\\\\|#g' \|
-e 's#/#\|\\/#g' \|
-e 's/\\/.\|\\\\.\|g' \|
-e 's/\\[/\|\\\\[/g' \|
`

# extract mapped unix account
_mapping=`sed -n 's/^'"${_happysed}"'=\(.*\)$/\1/p' "${_mappingfile}"`

# check with case insensitive flag if necessary
if [ -z "${_mapping}" ] ; then
    _mapping=`sed -n 's/^'"${_happysed}"'=\(.*\)$/\1/pi' "${_mappingfile}"`
fi

# print first result
if [ -n "${_mapping}" ] ; then
    echo "${_mapping}" | sed 'q'
else
    exit 1
fi
```

Configuring NICE EnginFrame Authorities

Besides the user mapping feature described in the previous section, some of the authorities shipped with EnginFrame have additional configuration parameters that can be changed to tailor the authentication process to your environment.

PAM

The pluggable authentication modules (PAM) authority authenticates a user using the PAM method of the Operating System.

The `PAM_SERVICE` parameter in the `$EF_TOP/conf/plugins/pam/ef.auth.conf` file on your Agent host specifies which PAM service is used for the authentication.

LDAP

This authority authenticates users querying a LDAP database. The settings in the `$EF_TOP/conf/plugins/ldap/ef.auth.conf` file on your Agent host allows you to specify the location of the LDAP server to use and customize access to the database.

The following parameters are available

- `LDAP_LDAPSEARCH`: the absolute path to the **ldapsearch** executable
- `LDAP_SERVER`: LDAP Server name or IP address
- `LDAP_PORT`: LDAP Server port
- `LDAP_BASE`: the base DN (Distinguished Name) for the search operation in the LDAP database

Active Directory

This authority authenticates users querying an Active Directory database. The settings in the `$EF_TOP/conf/plugins/activedirectory/ef.auth.conf` file on your Agent host allows you to specify the location of the Active Directory server to use and customize access to the database.

The following parameters are available

- `AD_LDAPSEARCH`: the absolute path to the **ldapsearch** executable
- `AD_SERVER`: LDAP Server name or IP address
- `AD_PORT`: LDAP Server port
- `AD_BASE`: the base DN (Distinguished Name) for the search operation in the Active Directory database
- `AD_BINDAS`: user that has permissions to bind to Active Directory Server for queries
- `AD_BINDPWD`: password for user binding to Active Directory Server

HTTP

HTTP authentication is slightly different from the other EnginFrame authentication methods: HTTP authentication is actually accomplished by the Web Server.

Once the user has been authenticated by the Web Server, the EnginFrame HTTP authentication authority allows you to perform some initialization steps. In particular it allows you to take advantage of the User Mapping feature described in [the section called “User Mapping”](#) in order to map the EnginFrame Portal users to the underlying operating system usernames.



Useful Software

When using HTTP authentication, in the case you need to configure a user mapping that retrieves information from an LDAP Server the `openldap` software can be useful.

Certificate

Certificate authority relies on X.509 certificates to encrypt the channel, check client identity and finally to authenticate users.

The web server in front of EnginFrame must be configured to use X.509 certificates for HTTPS channel encryption and client authentication while user identity is then determined by the Certificate authority within EnginFrame.

The web server configured to use certificates can be either Apache Tomcat® shipped with EnginFrame or an external web server as Apache® Web Server connected to NICE EnginFrame Tomcat® using the the AJP connector.

The client in order to successfully authenticate to EnginFrame must provide a valid certificate the web server recognizes and trusts, and a username to be used by EnginFrame to log in.

EnginFrame can be configured to retrieve the username from the first Common Name (CN) field of the Distinguished Name (DN) certificate property or from the HTTP request parameter named `_username`.

The parameter to configure where EnginFrame should look for the client username is `authorization.certificate.userCertificate` into `server.conf` configuration file. When `true` the username is retrieved from the CN field of the DN in the client certificate, `false` for retrieving the username from the client HTTP authentication request.

Even with the Certificate authority it is possible to take advantage of the User Mapping feature described in [the section called “User Mapping”](#) in order to map the EnginFrame Portal users to the underlying operating system usernames.

Custom Authentication Authority

If none of the standard mechanisms included in EnginFrame satisfies your requirements, you can easily create your own authentication algorithm.

The process of writing a custom EnginFrame authentication module involves the development of the following components

- The `EF_ROOT/plugins/<authority>/etc/<authority>.login` file to specify the fields the users must fill in order to authenticate. The field values will then be passed to the following authentication module at login time
- The `EF_ROOT/plugins/<authority>/bin/ef.auth` authentication module script. It receives authentication field values filled by the user as input in order to accomplish the authentication task

Once the new authority is ready it can be used in the `authority` attribute for `ef:agent` and `ef:service` tags in the SDF as usual.



Warning

If one of the authentication authorities included in EnginFrame does not completely satisfy your requirements, *do not modify EnginFrame system files*: create your own authority, using the ones provided by EnginFrame as a starting point.

Modifying one or more of the EnginFrame system files could corrupt the system. You are strongly recommended to leave EnginFrame system files as is. You should modify EnginFrame system files only if you are sure of what you are doing. By no means NICE or one of its partners responds for EnginFrame unresponsiveness if a system file has been modified. Besides this, a future EnginFrame update could override (without warning) your modifications.

The <authority>.login File

This XML file defines the authentication parameters required by the authentication script to check user credentials.

This file specifies the information (e.g. username, token, password, ...) prompts for the logon pages associated with the authentication module.

Login file must match the authority name (the name that is going to be used in the EnginFrame SDF) and must be located under the directory EF_ROOT/plugins/<authority>/etc. For security reasons the file ownership must be set to root:root and its permissions must be 644 (rw-r--r--).

For instance the login file for authority *pam* must be: EF_ROOT/plugins/pam/etc/pam.login

The <authority>.login file has the following structure:

```
<ef:login title="login_form_title"
  xmlns:ef="http://www.enginframe.com/2000/EnginFrame">;
  <ef:signature label="login_field_label"
    type="text|password"
    id="authentication_parameter_name" />;
  <!-- [<ef:signature ... />] -->
</ef:login>
```

Here an example (taken from *pam* authority):

```
<ef:login title="Login to EnginFrame">
  <ef:signature label="Username: "
    type="text"
    id="_username"/>
  <ef:signature label="Password: "
    type="password"
    id="_password"/>
</ef:login>
```

The ef:login entry corresponds to an authentication HTML page. Referring to the example, the HTML logon page asks users to enter a text token (the username) and a password.



Note

The authentication parameters <ef:signature> with id=_username and id=_password (the <ef:signature> tags of the pam.login

example) are strongly suggested. If the parameter `_username` is not used, there *must* be the `<ef:user-mapping>` XML in the authentication process output in order to set a proper user name in EnginFrame.

Please refer to EnginFrame Administrator's Reference for details on the XML format used by the `<authority>.login` file.

The ef.auth File

The script `ef.auth` actually implements the authentication procedure.

It must reside under the directory `EF_ROOT/plugins/<authority>/bin`

where `<authority>` must match the authority name that is going to be used in the EnginFrame SDF. For security reasons the file ownership must be set to `root:root` and its permissions must be `755` (`rwxr-xr-x`).

For instance the authentication script for the LDAP authority is `EF_ROOT/plugins/ldap/bin/ef.auth`.

The authentication script receives as input the login form parameters values filled by the user. Such values, separated by `'\0'` character (ASCII code 0), are directly passed to the standard input of the script in the same order, as they have been defined in the `<authority>.login` file.

Let's use the *pam* authority as an example. When user `demo` with password `secret` logs into EnginFrame, the string `demo\0secret\0` is passed to the standard input of the `ef.auth` script.

The authentication script checks the credentials passed as input and writes the response to the standard output. In case of a positive answer it must emit the following XML structure:

```
<?xml version="1.0"?>
<ef:auth xmlns:ef="http://www.enginframe.com/2000/EnginFrame">

  <ef:result>
    <ef:grant />
  </ef:result>

</ef:auth>
```

For a negative answer:

```
<?xml version="1.0"?>
<ef:auth xmlns:ef="http://www.enginframe.com/2000/EnginFrame">

  <ef:result>
    <ef:deny />
  </ef:result>

  <ef:error> <!-- Not mandatory -->
    <ef:message>error_message</ef:message>
  </ef:error>

</ef:auth>
```

If you want to support user mapping in your custom authentication authority, in case of positive answer the `ef.auth` script should include the `ef:user-mapping` tag in its output:

```
<?xml version="1.0"?>
<ef:auth xmlns:ef="http://www.enginframe.com/2000/EnginFrame">

  <ef:result>
    <ef:grant />
    <ef:user-mapping name="target_username"/>
  </ef:result>

</ef:auth>
```

where `target_username` is the username that should be used on the underlying operating system.

It is usually a good practice to move the user mapping logic to a separate script that can be changed without editing `ef.auth` itself. As described in [the section called “User Mapping”](#), authentication modules shipped with EnginFrame follow the convention of using `EF_ROOT/plugins/<authority>/bin/ef.user.mapping`.

If you follow this advice, your `ef.auth` script will have the following structure:

```
[Get credentials]

[Verify credentials]

[If User is authenticated]
  ACTUAL_USERID=[Custom User Mapping procedure]

  [Emit]
    <?xml version="1.0"?>
    <ef:auth xmlns:ef="http://www.enginframe.com/2000/EnginFrame">
      <ef:grant/>
      <ef:user-mapping name="$ACTUAL_USERID"/>
    </ef:auth>
  [end]
[end]
```



Tip

You can refer to the authentication plugins shipped with EnginFrame to see some examples of how the `ef.auth` scripts may be implemented

Authorization System

The EnginFrame Authorization System allows you to control in a fine-grained fashion which *users* can access the EnginFrame *resources*, granting or denying operations according to a set of predefined *policies*.

In the EnginFrame Authorization System *users* and *groups* are called Actors, the *resources* are EnginFrame SDF folders, services, service options, service actions and service output, while *policies* defining the permissions are specified using access control lists (ACL).

The authorization framework defines *who* can do *what* on *which resources*. By configuring the authorization system, it is possible to give different views of the EnginFrame Portal to different users and user groups.

Configuring Authorization

EnginFrame authorization settings are specified in the following configuration files. The ACLs and Actors defined in all of them are merged. In case of multiple definitions of the same ACLs or Actors (same id), the file priority is used to resolve conflicts.

- `$EF_TOP/conf/enginframe/authorization.xconf` - highest priority
- `$EF_TOP/<VERSION>/enginframe/conf/authorization.xconf`
- `$EF_TOP/conf/plugins/<plug-in>/authorization.xconf`
- `$EF_TOP/<VERSION>/enginframe/plugins/<plug-in>/conf/authorization.xconf` - lowest priority

Modifications to these files are automatically picked up by a running EnginFrame Server, without requiring a restart.

The `authorization.xconf` file is an XML file consisting of two sections

- An *Actors* section introduced with the tag `<ef:acl-actor-list>`
- An *Access Control Lists* section starting with the tag `<ef:acl-list>`

```

<ef:authorization>
  <!-- EnginFrame Authorization Actors section -->
  <ef:acl-actor-list>
    ...
  </ef:acl-actor-list>
  <!-- EnginFrame Authorization ACL section -->
  <ef:acl-list>
    ...
  </ef:acl-list>
</ef:authorization>

```

Defining Actors

Actors are entities able to perform actions on resources. In the EnginFrame infrastructure an Actor can be a user, a group of users, or a group of groups of users and so on.

Since an Actor can be a single user or a group, we refer to each component with the term *member*.

There are three ways to define an actor

- *efgroup Actor*: is an explicit list of members. It can contain one or more users or even other Actors already defined.
- *osgroup Actor*: the members will be the users belonging to the Operating System group with the same id of this actor
- *user Actor*: an actor for a single EnginFrame user. It exists just for "renaming" purposes because usually there is no need to wrap an EnginFrame user id in an Actor

The XML syntax to define actors inside `authorization.xconf` is the following

```

<ef:acl-actor id="unique_id" type="efgroup|osgroup">

```

and an actor of type `efgroup` must include members defined with

```

<ef:acl-member type="efuser|acl-actor">...</ef:acl-member>

```



Note

When using a member of type `acl-actor` the actor must be defined in the `authorization.xconf`

Here follows an example of an Actors section:

```

...
<!-- EnginFrame Authorization Actors section -->
<ef:acl-actor-list>
  <!-- Actor made up of two simple users and two Actors -->
  <ef:acl-actor id="nice" type="efgroup">
    <ef:info>NICE people</ef:info>
    <ef:acl-member type="efuser">andrea</ef:acl-member>
    <ef:acl-member type="efuser">beppe</ef:acl-member>
    <ef:acl-member type="acl-actor">
      developers
    </ef:acl-member>
    <ef:acl-member type="acl-actor">efadmin</ef:acl-member>
  </ef:acl-actor>
  <ef:acl-actor id="developers" type="efgroup">
    <ef:info>EnginFrame Developers</ef:info>
    <ef:acl-member type="efuser">antonio</ef:acl-member>
    <ef:acl-member type="efuser">mauri</ef:acl-member>
    <ef:acl-member type="acl-actor">goldrake</ef:acl-member>
  </ef:acl-actor>
  <!--
    Member of this Actor are dynamically loaded from the
    Operating system group "efadmin" using the script:
    NICE_ROOT/enginframe/plugins/myplugin/bin/ef.load.users
  -->
  <ef:acl-actor id="efadmin" type="osgroup" plugin="myplugin"/>
</ef:acl-actor-list>
...

```

Defining Access Control Lists

ACL define policies to be applied to Actors when they try to perform actions on resources.

The purpose of an ACL is to grant or deny permissions on actions an EnginFrame Actor can perform without reference to the resource on which the ACL is applied.

An ACL consists of three main parts. The first section it is used to "bias" the ACL towards the allow or the deny directive. The following two parts define allow and deny directives, in which Actors are bound with the actions they can or cannot perform.

The ACL structure explanation follows

- *ACL priority*, defines if the allow or deny directive has priority for this ACL
 - *allow priority*: access is allowed by default. The deny directives are evaluated before the allow ones. Any Actor which does *not* match a deny directive *or* does match an allow directive will be allowed access to the resource;
 - *deny priority*: access is denied by default. The allow directives are evaluated before the deny ones. Any Actor which does *not* match an allow directive *or* does match a deny directive will be denied access to the resource
- *ACL allow*, it contains the allow directives, a list of Actors in which a set of actions specifies the operations the Actor can actually perform on a generic resource guarded by this ACL.
- *ACL deny*, it contains the deny directives, a list of Actors in which a set of actions specifies the operations the Actor cannot perform on a generic resource guarded by this ACL.

The definition of an ACL adheres to this XML structure:

```
...
<ef:acl id="unique_id">
  <ef:acl-priority>allow | deny</ef:acl-priority>
  <ef:acl-allow>
    <ef:actor id="actor_id">
      <ef:action-list>
        <ef:read/>
        <ef:execute/>
        ...
      </ef:action-list>
    </ef:actor>
  </ef:acl-allow>
  <ef:acl-deny>
    <ef:actor id="actor_id">
      <ef:action-list>
        <ef:read/>
        <ef:execute/>
        ...
      </ef:action-list>
    </ef:actor>
  </ef:acl-deny>
</ef:acl>
...
```

The `id` attribute of an `ef:actor` can refer to a predefined `ef:acl-actor` or directly to an EnginFrame user id.

There are four kinds of actions EnginFrame can accept. The action meaning and the consequent EnginFrame behavior depends on the type of the resource the ACL is applied to

- `<ef:read/>`
- `<ef:write/>`
- `<ef:execute/>`
- `<ef:delete/>`

Here an example of an ACL definition follows:

```

...
<ef:acl-list>
  ...
  <ef:acl id="priv-exec">
    <ef:info>Privileged permissions for Admins</ef:info>
    <ef:acl-priority>deny</ef:acl-priority>
    <ef:acl-allow>
      <ef:actor id="efadmin">
        <ef:action-list>
          <ef:read/>
          <ef:write/>
          <ef:execute/>
          <ef:delete/>
        </ef:action-list>
      </ef:actor>
    </ef:acl-allow>
  </ef:acl>
  ...
</ef:acl-list>
...

```

Condition Based ACL

An ACL can include some extra conditions that have to be met in order to grant access to a resource.

These conditions can take into account the value of session variables, system properties, and xpath expressions and be combined using logical the operators or, and, not, and equals.

The concept is best explained with an example:

```

...
<ef:acl-list>
  ...
  <ef:acl id="project-acme">
    <ef:info>
      Privileged permissions for Project ACME
    </ef:info>
    <ef:acl-priority>deny</ef:acl-priority>
    <ef:acl-allow>
      <ef:actor id="company-users">
        <ef:condition>
          <ef:or>
            <ef:and>
              <ef:equals type="session"
                id="project"
                value="acme"
                casesensitive="true" />
              <ef:equals type="session"
                id="{project}_responsible"
                value="true"
                casesensitive="false" />
            </ef:and>
            <ef:and>
              <ef:equals type="session"
                id="administrator"
                value="true"
                casesensitive="false" />
              <ef:not>
                <ef:equals type="property"
                  id="{EF_USER}"
                  value="jack"
                  casesensitive="true" />
              </ef:not>
            </ef:and>
          </ef:or>
        </ef:condition>
        <ef:action-list>
          <ef:read/>
          <ef:write/>
          <ef:execute/>
          <ef:delete/>
        </ef:action-list>
      </ef:actor>
    </ef:acl-allow>
  </ef:acl>
  ...
</ef:acl-list>

```

The current user can access resources guarded by the ACL "project-acme" only if one of the following conditions is true

- he belongs to "company-users" *and* the session variable "project" is present and its value is "acme" *and* the session variable "acme_responsible" is set to "true" independently from the case of letters
- the session variable "administrator" is "true" independently from case of letters *and* is *not* named "jack"

The `ef:and` and `ef:or` tags are condition containers and they evaluate to true respectively when all the included conditions evaluate to true or when at least one of them does.

The `ef:not` may contain only one condition and it negates the result of its evaluation.

The `ef:equals` tag checks two arguments for equality. The arguments to check are defined by the `type` and by the `id` attributes. The `casesensitive` attribute specifies if the letter case should be taken into account or not.

The `type` can refer to three different kinds of values:

Session variables

The value to be compared is the one of a session variable with the specified `id`.

For instance in the example above

```
<ef:equals type="session"
           id="administrator"
           value="true"
           casesensitive="false"/>
```

EnginFrame checks if a session variable names `administrator` is defined and its value is `true`.

System properties

The value to be compared is the one of a system property with the specified `id`. EnginFrame system properties are the ones loaded by the JVM, the ones passed to the JVM via command line (i.e. `-Dname=value`), and the ones loaded from the EnginFrame configuration files.

```
<ef:equals type="property"
           id="${EF_USER}"
           value="mary"
           casesensitive="true"/>
```

EnginFrame checks if the system property named `${EF_USER}` has value `mary`.

XPath expressions

The value to be compared is the one extracted from the current DOM by the xpath expression specified in the `id` attribute.

For instance

```
<ef:equals type="xpath"
           id="starts-with(//ef:profile/ef:user/., 'br')"
           value="true"
           casesensitive="true"/>
```

checks if the DOM element `//ef:profile/ef:user/text()` starts with `br`.

聚辰科技(CAX-CLOUD)
info@cax-cloud.com

13

Configuring HTTPS

HTTPS, [Hypertext Transfer Protocol over Secure Socket Layer](#), is a URI scheme used for secure HTTP connections.

This system was designed to provide authentication and encryption and is widely used for security-sensitive communication such as payment transactions and corporate logons.

During the installation process, EnginFrame installer allows to automatically configure Apache Tomcat® to use HTTPS connector instead of plain HTTP. By default EnginFrame installs configuring HTTP protocol in the Apache Tomcat® web server.

Choosing HTTPS, the installer will automatically create self-signed certificates under the directory `$EF_TOP/conf/tomcat/conf/certs` and will configure the Apache Tomcat® connector to use them.

Change HTTPS certificates

If self-signed certificates do not suit your needs or you already have valid certificates to setup an HTTPS web server, you should refer to the [Apache Tomcat® official documentation](#) in order to properly configure the web server to use your certificates.

Depending on the format of the available certificate there are different procedure to follow to setup the Apache Tomcat® connector.

For example, starting from a PEM private key and PEM certificate, they need to be converted first in order to be handled by Java™ `keytool` and keystores, as recommended by the Apache Tomcat® documentation.

In this case the first step would be to convert PEM key and certificate into PKCS12 format:

```
$ openssl pkcs12 -export -in <your_CA_signed_PEM_cert>
-inkey <your_PEM_private.key> -out <your_certificate_name>.p12
-name tomcat -chain -CAfile <your_root_CA_certificate>
```

Next, the newly created PKCS12 certificate should be imported into a Java™ keystore file:

```
$ $JAVA_HOME/bin/keytool -importkeystore -deststorepass <password>
-destkeypass <password> -destkeystore tomcat.keystore
-srckeystore <exported_private_key_and_cert.p12> -srcstoretype PKCS12
-srcstorepass <password> -alias tomcat
```

If your CA has intermediate certificates, you should import them into the keystore file. It is very likely that your CA provides instructions on how to do this and how the certificates should be named. For example, assuming a CA intermediate certificate is already in a format supported by Java™ `keytool`, it can be simply imported in this way:

```
$ $JAVA_HOME/bin/keytool -import -alias intermed -keystore tomcat.keystore
-trustcacerts -file gd_intermediate.crt
```

You may also need to import the root CA certificate into the keystore in the case it doesn't come from one of the well known CAs whose root certificates are pre-configured in the Java™ system.

For reference on how to use the [keytool](#) and [openssl](#) commands, please look at the official documentation.

Symbols

<authority>.login, 138

A

Access Control Lists, 141, 143

actions, 144

allow, 143

conditional operators, 145, 146

Condition Based, 145

Defining, 143

deny, 143

priority, 143

Actors, 141

Defining, 142

efgroup, 142

osgroup, 142

user, 142

Agent

agent.conf, 67, 74, 75

configuration, 67, 74, 75

log.agent.xconf, 67, 115

logging, 67, 115

ports, 74

agent.conf, 67, 74, 75

Apache®

connection with Tomcat®, 77

Authentication, 38, 133

Authorities, 133

Authority

Custom, 137

Default, 133

LDAP, 139

ef.user.mapping, 134

EFAUTH_USERMAPPING, 134

user mapping

Default, 134

authorization

authorization.xconf, 67

configuration, 67

Authorization, 141

authorization.xconf, 141

Configuring, 141

policies, 141

resources, 141

users, 141

authorization.xconf, 67, 141

C

Charts

configuration, 79

configuration, 65

Agent, 67

ports, 74

agent.conf, 67, 74, 75

authorization, 67

Charts, 79

default Agent, 69

enginframe.conf, 67

error page, 73

files, 65

Java™, 69

log.agent.xconf, 67

log.server.xconf, 67

logging, 67, 67

media types, 70

mime-types, 70

mime-types.xml, 67, 70

plugin, 68

ports, 74

Server, 67

server.conf, 67

service output, 74

session timeout, 77

users, 75

D

Deployment Strategies, 35

Downloading, 19

E

EF_SPOOLER_DIR, 105

configuring, 108

permissions, 106

ef.auth, 139

ef.download.server.url, 109

ef.download.stream.inactivity.timeout, 109

ef.download.stream.sleep.time, 109

ef.reaper.sleep.time, 111

- ef.repository.dir, 110
- enginframe.conf, 67
- EnginFrame Portal, 55
 - accessing, 58
 - script, 55
 - start, 55
 - status, 56
 - stop, 55
- error page, 73

H

- HTTPS
 - configuring, 149
 - Tomcat®, 149

I

- Installation
 - Batch, 46
 - Fine Tuning, 46
 - Unix®, 45
- Installation Directories, 36
- INTERACTIVE_SHARED_ROOT
 - permissions, 114

J

- Java™
 - configuration, 69

L

- License, 19
- licenses, 123
 - checking, 125
 - directory, 123
 - files format, 124
 - files location, 123
 - host, 126
 - host list, 126
 - location, 123
 - logging, 129
 - monitoring, 127
 - token, 125
- log.agent.xconf, 67, 115
- log.server.xconf, 67, 115
- logging, 115
 - Agent, 115
 - configuration, 67, 67, 115
 - configuring categories, 118
 - file location, 116
 - file size, 117
 - level, 118
 - licenses, 129

- message format, 119
- policy, 117
- Scriptlet, 121
- Server, 115
- Tomcat®, 115

M

- mime-types.xml, 67, 70

P

- plugin, 68
 - configuration, 68
 - custom, 68
 - official, 68
- Prerequisites, 21

R

- Requirements
 - Network, 32
- RMI, 74

S

- Scriptlet
 - logging, 121
- Server
 - configuration, 67
 - log.server.xconf, 67, 115
 - logging, 67, 115
 - server.conf, 67
- server.conf, 67
- service output
 - configuration, 74
- session
 - configuration, 77
- sessions, 113
- session-timeout, 77
- spooler, 105
 - configuring, 108
 - dead, 111
 - download, 109
 - download files, 109
 - EF_SPOOLER_DIR, 105
 - ef.download.server.url, 109
 - ef.download.stream.inactivity.timeout, 109
 - ef.reaper.sleep.time, 111
 - ef.repository.dir, 110
 - life cycle, 110
 - permissions, 106
 - reaper, 111
 - repository location, 110
 - requirements, 106

- root directory, 108, 108
- security, 106
- structure, 105

T

- token
 - license, 125
- Tomcat®
 - connection with Apache®, 77
 - logging, 115

U

- Users
 - Special, 38
- users
 - configuration, 75, 77
 - session timeout, 77
 - switching, 75

W

- web.xml
 - session-timeout, 77

聚辰科技(CAX-CLOUD)
info@cax-cloud.com