UNICORE Gateway

UNICORE Team

Document Version:	1.1.0
Component Version:	7.8.0
Date:	23 03 2017

This work is co-funded by the EC EMI project under the FP7 Collaborative Projects Grant Agreement Nr. INFSO-RI-261611.

Contents

This user manual provides information on running and using the UNICORE gateway. Please note also the following places for getting more information:

UNICORE Website: http://www.unicore.eu

Support list: unicore-support@lists.sf.net

Developer's list: unicore-devel@lists.sf.net

Introduction

The Gateway is the entry point into a UNICORE site, routing HTTPS traffic to servers like UNICORE/X. It is installed in front of any networking firewall. It authenticates incoming messages and forwards them to their intended destination. The Gateway receives the reply and sends it back to the client. In this way, only a single open port in a site's firewall has to be configured.

LIMITATIONS
This forwarding process only works for “most” HTTP requests, and is not a complete HTTP reverse proxy implementation. For example, it is not possible to run a full, complex web application like the UNICORE portal "behind" the gateway. Check the respective components's manual whether it can be run behind the Gateway.

In effect, traffic to a *virtual* URL, e.g. *https://mygateway:8088/Alpha* is forwarded to the real URL, e.g. *https://host1:7777*.

The mappings of virtual URL to real URL for the available sites are listed in a configuration file `connections.properties`. Additionally, the gateway supports dynamic registration of sites.

The second functionality of the gateway is authentication of incoming requests. Connections to the gateway are made using client-authenticated SSL, so the gateway will check whether the caller presents a certificate issued by a trusted authority. Information about the authenticated client is forwarded to services behind the gateway in UNICORE proprietary format (as SOAP header element).

IMPORTANT NOTE ON PATHS
The UNICORE Gateway is distributed either as a platform independent and portable bundle (as a part of the UNICORE core server package) or as an installable, platform dependent package format such as RPM.
Depending on the installation method, the paths to various Gateway files are different. If installing using a distribution-specific package the following paths are used:

```
CONF=/etc/unicore/gateway
BIN=/usr/sbin
LOG=/var/log/unicore/gateway
```

If installing using the portable bundle all Gateway files are installed under a single directory. Path prefixes then are as follows, where INST is a directory where the Gateway was installed:

```
CONF=INST/conf
BIN=INST/bin
LOG=INST/log
```

The above variables (CONF, BIN and LOG) are used throughout the rest of this manual.

Installation

The UNICORE Gateway is distributed in the following formats:

1. As a part of platform independent installation bundle called UNICORE core server bundle. The UNICORE core server bundle is provided in two forms: one with a graphical installer and one with a command line installer.
2. As a binary, platform-specific package available currently for RedHat (Centos) and Debian platforms. Those packages are not tested on all possible platforms, but should work without any problems with other versions of similar distributions, e.g. SL6, Centos, or Fedora.

Installation from the core server bundle

Download the core server bundle from the UNICORE project website.

If you use the graphical installer, follow the on-screen instructions and do not forget to enable the Gateway checkbox when prompted.

If you use the console installer, please review the README file available after extracting the bundle. You don't have to change any defaults as the Gateway is installed by default.

Installation from a Linux package (rpm or deb)

Use your distribution's package manager to install.

Upgrading

The general update procedure is presented below, with possible variations:

1. Stop the old Gateway.
2. Update the server package. This step mostly applies for RPM/DEB managed installations. For Quickstart installation it is enough to replace the *.jar files with the new ones.
3. Start the newly installed Gateway.
4. Verify log file and fix any problems reported.

Configuration

The gateway is configured using a set of configuration files, which reside in the `CONF` subdirectory.

Java and environment settings: startup.properties

This file contains settings related to the Java VM, such as the Java command to use, memory settings, library paths etc.

Configuring sites: connections.properties

This is a simple list connecting the names of sites and their physical addresses. An example is:

```
DEMO-SITE = https://localhost:7777
REGISTRY = https://localhost:7778
```

If this file is modified, the gateway will re-read it at runtime, so there is no need to restart the gateway in order to add or remove sites.

Optionally administrator can enable a possibility for dynamic site registration at runtime, see Section 4.4.2 for details. Then this file should contain only the static entries (or none if all sites register dynamically).

Further options for back-end sites configuration are presented in Section 6.

Main server settings: gateway.properties

Use the `gateway.hostname` property to configure the network interface and port the gateway will listen on. You can also select between https and http protocol, though in almost all cases https will be used.

Example:

```
gateway.hostname = https://192.168.100.123:8080
```

Note
If you set the host to `0.0.0.0`, the gateway will listen on all network interfaces of the host machine, else it will listen only on the specified one.

If the scheme of the hostname URL is set to https, the Gateway uses the configuration data from `security.properties` to configure the HTTPS settings.

Certificate-less end-user access

With UNICORE 7, it is possible that end-users do not have client certificates. To enable them to connect, the Gateway needs to accept TLS connections without a client certificate. To configure this, set the following in `gateway.properties`

```
gateway.httpServer.requireClientAuthn=false
```

Scalability settings

To fine-tune the operational parameters of the embedded Jetty server, you can set advanced HTTP server parameters. See Section 4.5 for details. Among others you can use the non-blocking IO connector offered by Jetty, which will scale up to higher numbers of concurrent connections than the default connector.

The gateway acts as a https client for the VSites behind it. The number of concurrent calls is limited, and controlled by two parameters:

```
# maximum total number of concurrent calls to Vsites
gateway.client.maxTotal=100
# total number of concurrent calls per site
gateway.client.maxPerService=20
```

You can also control the limit on the maximum SOAP header size which is allowed by the gateway. **Typically you don't have to touch this parameter**. However if your clients do produce very big SOAP headers and gateway blocks them, you can increase the limit. Note that such a giant SOAP header usually means that the client is not behaving in a sane way, e.g. is trying to perform a DoS attack.

```
# maximum size of an accepted SOAP header, in bytes
gateway.soapMaxHeader=102400
```

Note that gateway may consume this amount of memory (plus some extra amount for other data) for each opened connection. Therefore, this value multiplied by the number of maximum allowed connections, should be significantly lower, then the total memory available for the gateway.

Dynamic registration of Vsites

Dynamic registration is controlled by three properties in `CONF/gateway.properties` file:

```
gateway.registration.enable=true
```

If set to true, the gateway will accept dynamic registrations which are made by sending a HTTP POST request to the URL /VSITE_REGISTRATION_REQUEST

Filters can be set to forbid access of certain hosts, or to require certain strings in the Vsite addresses. For example:

```
gateway.registration.deny=foo.org example.org
```

will deny registration if the remote hostname contains *foo.org* or *example.org*. Conversely,

```
gateway.registration.allow=mydomain.org
```

will only accept registrations if the remote address contains *mydomain.org*. These two (deny and allow) can be combined.

Web interface ("monkey page")

For testing and simple monitoring purposes, the gateway displays a website showing detailed site information (the details view can be disabled). Once the Gateway is running, open up a browser and navigate to https://<gateway_host>:8080 (or whichever URL the gateway is running on). As the gateway usually runs using SSL, you will need to import a suitable client certificate into your web browser.

A HTML form for testing the dynamic registration is available as well, by clicking the link in the footer of the main gateway page.

To disable the Vsite details page, set

```
gateway.disableWebpage=true
```

Main options reference

Property name	**Type**	**Default value / mandatory**	**Description**
`gateway.hostname`	string	*mandatory to be set*	external gateway bind address
`gateway.registration.allow`	string	-	Space separated list of allowed hosts for dynamic registration.
`gateway.registration.deny`	string	-	Space separated list of denied hosts for dynamic registration.
`gateway.registration.enable`	[true, false]	`false`	Whether dynamic registration of sites is enabled.
	--- Passing Consignor info ---		
`gateway.consignorTokenTimeTolerance`	integer >= 0	`30`	The validity time of the authenticated client information passed to backend sites will start that many seconds before the real authentication. It is used to mask time synchronization problems between machines.
`gateway.consignorTokenValidity`	integer >= 1	`60`	What is the validity time of the authenticated client information passed to backend sites. Increase it if there machines clocks are not synhronized.
`gateway.signConsignorToken`	[true, false]	`false`	Controls whether information about the authenticated client (the consignor) passed to backend sites should be signed, or not. Signing is slower, but is required when sites may be reached directly, bypassing the Gateway.
	--- Gateway $\rightarrow$ Site client ---		
`gateway.client.chunked`	[true, false]	`true`	Controls whether chunked passing of HTTP requests to backend sites is supported.

Property name	Type	Default value / mandatory	Description
`gateway.client.connectionTimeout`	integer number	`30000`	Connection timeout, used when connecting to backend sites.
`gateway.client.expectContinue`	[true, false]	`true`	Controls whether the HTTP expec-continue mechanism is enaled on connections to backend sites.
`gateway.client.gzip`	[true, false]	`true`	Controls whether support for compression is announced to backend sites.
`gateway.client.keepAlive`	[true, false]	`true`	Whether to keep alive the connections to backend sites.
`gateway.client.maxPerService`	integer number	`20`	Maximum allowed number of connections per backend site.
`gateway.client.maxTotal`	integer number	`100`	Maximum total number of connections to backend sites allowed.
`gateway.client.socketTimeout`	integer number	`30000`	Connection timeout, used when connecting to backend sites.
--- *Advanced* ---			
`gateway.disableWebpage`	[true, false]	`false`	Whether the (so called monkey) status web page should be disabled.
`gateway.externalHostname`	string	*not set*	External address of the gateway, when it is accessible through a frontend server as Apache HTTP.
`gateway.soapMaxHeader`	integer [1024 — 1024000000]	`102400`	Size in bytes of the accepted SOAP header. In the most cases you don't need to change it.

security.properties

In the `security.properties` file, the trust store and gateway credential is configured.

Proxy certificate support

The UNICORE gateway optionally accepts proxy certificates as used by other Grid middleware systems. In general, we think proxies are a bad idea, but for interoperability purposes, proxies support can be enabled. If enabled, the clients using proxies are authenticated as the initial issuer of the presented proxy certificates chain. See the above reference properties table for the actual setting.

The most important, root log categories used by the Gateway's logging are:

unicore.gateway	General gateway logging
unicore.connections	Log IPs of clients, and the DN after the SSL handshake
unicore.httpserver	HTTP processing, Jetty server
unicore.security	Certificate details and other security

The Gateway uses the so called MDC (Mapped Diagnostic Context) to provide additional information on the client which is served. In the MDC the client's IP address and client's Distinguished Name is stored. You can control whether to attach MDC to each line by the `%X{entry}` log4j pattern entry. As the `entry` you can use: `clientIP` or `clientName`. For example:

```
log4j.appender.A1.layout.ConversionPattern=%d [%t] [%X{clientIP} %X ↩
    {clientName}] %-5p %c{1} - %m%n
```

Using Apache httpd as a frontend

You may wish to use the Apache webserver (httpd) as a frontent for the gateway (e.g. for security or fault-tolerance reasons).

Requirements

- Apache httpd
- mod_proxy for Apache httpd

External references

- https://wiki.eclipse.org/Jetty/Howto/Configure_mod_proxy

Using the Gateway for failover and/or loadbalancing of UNICORE sites

The Gateway can be used as a simple failover solution and/or loadbalancer to achieve high availability and/or higher scalability of UNICORE/X sites without additional tools.

A site definition (in `CONF/connections.properties`) can be extended, so that multiple physical servers are used for a single virtual site.

An example for such a so-called multi-site declaration in the connections.properties file looks as follows:

```
#declare a multisite with two physical servers

MYSITE=multisite:vsites=https://localhost:7788 https://localhost ↩
    :7789
```

This will tell the gateway that the virtual site "MYSITE" is indeed a multi-site with the two given physical sites.

Configuration

Configuration options for the multi-site can be passed in two ways. On the one hand they can go into the connections.properties file, by putting them in the multi-site definition, separated by ";" characters:

```
#declare a multisite with parameters

MYSITE=multisite:param1=value1;param2=value2;param3=value3;...
```

The following general parameters exist

vsites	List of physical sites
strategy	Class name of the site selection strategy to use (see below)
config	Name of a file containing additional parameters

Using the "config" option, all the parameters can be placed in a separate file for enhanced readability. For example you could define in connections.properties:

```
#declare a multisite with parameters read from a separate file

MYSITE=multisite:config=conf/mysite-cluster.properties
```

and give the details in the file "conf/mysite-cluster.properties":

```
#example multisite configuration
vsites=https://localhost:7788 https://localhost:7789

#check site health at most every 5 seconds
strategy.healthcheck.interval=5000
```

Available Strategies

A selection strategy is used to decide where a client request will be routed. By default, the strategy is "Primary with fallback", i.e. the request will go to the first site if it is available, otherwise it will go to the second site.

Primary with fallback

This strategy is suitable for a high-availability scenario, where a secondary site takes over the work in case the primary one goes down for maintenance or due to a problem. This is the default strategy, so nothing needs to be configured to enable it. If you want to explicitely enable it anyway, set

```
strategy=primaryWithFallback
```

The strategy will select from the first two defined physical sites. The first, primary one will be used if it is available, else the second one. Health check is done on each request, but not more frequently as specified by the "strategy.healthcheck.interval" parameter. By default, this parameter is set to 5000 milliseconds.

Changes to the site health will be logged at "INFO" level, so you can see when the sites go up or down.

Round robin

This strategy is suitable for a load-balancing scenario, where a random site will be chosen from the available ones. To enable it, set

```
strategy=roundRobin
```

Changes to the site health will be logged at "INFO" level, so you can see when the sites go up or down.

It is very important to be aware that this strategy requires that all backend sites used in the pool, share a common persistence. It is because Gateway does not track clients, so particular client requests may land at different sites. This is typically solved by using a non-default, shared database for sites, such as MySQL.

Note
Currently loadbalancing of target sites is an experimental feature and is not yet fully functional. It will be improved in future UNICORE versions.

Custom strategy

You can implement and use your own failover strategy, in this case, use the name of the Java class as strategy name:

```
strategy=your_class_name
```

Gateway failover and migration

The Section 6 covered usage of the Gateway to provide failover of backend services. However it may be needed to guarantee high-availabilty for the Gateway itself or to move it to other machine in case of the original one's failure.

Gateway's migration

The Gateway does not store any state information, therefore its migration is easy. It is enough to install the Gateway at the target machine (or even to simply copy it in the case of installation from the core server bundle) and to make sure that the original Gateway's configuration is preserved.

If the new machine uses a different address, it needs to be reflected in the server's configuration file (the listen address). Also, the configuration of sites behind the gateway must be updated accordingly.

Failover and loadbalancing of the Gateway

Gateway itself doesn't provide any features related to its own redundancy. However as it is stateless, the standard redundancy solutions can be used.

The simpliest solution is to use Round Robin DNS, where DNS server routes the Gateway's DNS address to a pool of real IP addresses. While easy to set up this solution has a significant drawback: DNS server doesn't care about machines being down.

The much better choice is to use the Linux-HA software suite, often known under the name of its principal component, the *heartbeat*. For details see http://www.linux-ha.org

Additionally a more advanced HTTP-aware software can be used, such as HA-Proxy (http://haproxy.1wt.eu). Currently Gateway and UNICORE don't maintain HTTP sessions so usage of the HTTP-aware load-balancer is not strictly required, but such solutions generally provide more general purpose features.

Building the Gateway from source

To checkout the latest version of the Gateway source code, do

```
svn co http://unicore.svn.sourceforge.net/svnroot/unicore/gateway/ ↩
    trunk gateway
```

You will need to install Maven from http://maven.apache.org Compile using

```
mvn install
```

Compiles the code and runs the tests.

The file "README-building.txt" contains instructions for building distributable packages.