UNICORE GATEWAY

UNICORE Team

Document Version:	1.1.0
Component Version:	8.1.0
Date:	23 02 2021

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: https://www.unicore.eu

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

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

GitHub page: https://github.com/UNICORE-EU/gateway

1 Introduction

The Gateway is the entry point into a UNICORE site, routing HTTPS traffic to servers like UNICORE/X. It forwards client traffic to the intended destination, optionally authenticating the client. 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
The Gateway is not a complete HTTP reverse proxy implementation. For example, it is not possible to run a full, complex web application "behind" the Gateway, especially not if protocols like WebSocket are used.

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 (optional) authentication of incoming requests. Connections to the Gateway are made using SSL, so the Gateway can be configured to check whether the caller presents a certificate issued by a trusted authority. Information about the client is forwarded to services behind the Gateway in UNICORE proprietary format (as a SOAP or HTTP header).

The Gateway will forward the IP address of the client to the back-end server.

Last not least, the Gateway can be configured as a HTTP load balancer.

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/logs
```

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

2 Installation

The UNICORE Gateway is distributed in the following formats:

1. As a part of platform independent installation bundle called UNICORE Server bundle. The UNICORE Server bundle is provided as a tar package and includes 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. SL, Centos, or Fedora.

To run, the Gateway requires Java (JRE headless is sufficient) in version 8 or later. We recommend using OpenJDK.

2.1 Installation from the Server bundle

Download the server bundle from the UNICORE project website.

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.

You should create and use a system user (e.g. *unicore*) to install and run the gateway. For security reasons, do not run the Gateway as the *root* user.

2.2 Installation from a Linux package (rpm or deb)

Use your distribution's package manager to install.

3 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.

4 Configuration

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

4.1 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.

4.2 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 4.3.3 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.

4.3 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.

4.3.1 Credential and truststore settings

The Gateway credential and truststore is configured using the following properties

Property name	Type	Default value / mandatory	Description
`gateway.credential.path`	filesystem path	*mandatory to be set*	Credential location. In case of *jks*, *pkcs12* and *pem* store it is the only location required. In case when credential is provided in two files, it is the certificate file path.
`gateway.credential.format`	[jks, pkcs12, der, pem]	-	Format of the credential. It is guessed when not given. Note that *pem* might be either a PEM keystore with certificates and keys (in PEM format) or a pair of PEM files (one with certificate and second with private key).
`gateway.credential.password`	string	-	Password required to load the credential.

Property name	**Type**	**Default value / mandatory**	**Description**
`gateway.credential.keyPath`	string	-	Location of the private key if stored separately from the main credential (applicable for *pem* and *der* types only),
`gateway.credential.keyPassword`	string	-	Private key password, which might be needed only for *jks* or *pkcs12*, if key is encrypted with different password then the main credential password.
`gateway.credential.keyAlias`	string	-	Keystore alias of the key entry to be used. Can be ignored if the keystore contains only one key entry. Only applicable for *jks* and *pkcs12*.

Property name	**Type**	**Default value / mandatory**	**Description**
`gateway.truststore.allowProxy`	[ALLOW, DENY]	`ALLOW`	Controls whether proxy certificates are supported.
`gateway.truststore.type`	[keystore, openssl, directory]	*mandatory to be set*	The truststore type.
`gateway.truststore.updateInterval`	integer number	`600`	How often the truststore should be reloaded, in seconds. Set to negative value to disable refreshing at runtime. *(runtime updateable)*
--- Directory type settings ---			
`gateway.truststore.directoryConnectionTimeout`	integer number	`15`	Connection timeout for fetching the remote CA certificates in seconds.

Property name	**Type**	**Default value / mandatory**	**Description**
`gateway.truststore.directoryDiskCachePath`	filesystem path	-	Directory where CA certificates should be cached, after downloading them from a remote source. Can be left undefined if no disk cache should be used. Note that directory should be secured, i.e. normal users should not be allowed to write to it.
`gateway.truststore.directoryEncoding`	[PEM, DER]	PEM	For directory truststore controls whether certificates are encoded in PEM or DER. Note that the PEM file can contain arbitrary number of concatenated, PEM-encoded certificates.
`gateway.truststore.directoryLocations.*`	list of properties with a common prefix	-	List of CA certificates locations. Can contain URLs, local files and wildcard expressions. *(runtime updateable)*
--- *Keystore type settings* ---			
`gateway.truststore.keystoreFormat`	string	-	The keystore type (jks, pkcs12) in case of truststore of keystore type.
`gateway.truststore.keystorePassword`	string	-	The password of the keystore type truststore.
`gateway.truststore.keystorePath`	string	-	The keystore path in case of truststore of keystore type.
--- *Openssl type settings* ---			
`gateway.truststore.opensslNewStoreFormat`	[true, false]	false	In case of openssl truststore, specifies whether the trust store is in openssl 1.0.0+ format (true) or older openssl 0.x format (false)

Property name	Type	Default value / mandatory	Description
`gateway.truststore.opensslNsMode`	[GLOBUS_EUGRIDPMA, EUGRIDPMA_GLOBUS, GLOBUS, EUGRIDPMA, GLOBUS_EUGRIDPMA_REQUIRE, EUGRIDPMA_GLOBUS_REQUIRE, GLOBUS_REQUIRE, EUGRIDPMA_REQUIRE, EUGRIDPMA_AND_GLOBUS, EUGRIDPMA_AND_GLOBUS_REQUIRE, IGNORE]	`EUGRIDPMA_GLOBUS`	In case of openssl truststore, controls which (and in which order) namespace checking rules should be applied. The *REQUIRE* settings will cause that all configured namespace definitions files must be present for each trusted CA certificate (otherwise checking will fail). The *AND* settings will cause to check both existing namespace files. Otherwise the first found is checked (in the order defined by the property).
`gateway.truststore.opensslPath`	filesystem path	`/etc/grid-security/certificates`	Directory to be used for opeenssl truststore.
--- Revocation settings ---			
`gateway.truststore.crlConnectionTimeout`	integer number	`15`	Connection timeout for fetching the remote CRLs in seconds (not used for Openssl truststores).
`gateway.truststore.crlDiskCachePath`	filesystem path	-	Directory where CRLs should be cached, after downloading them from remote source. Can be left undefined if no disk cache should be used. Note that directory should be secured, i.e. normal users should not be allowed to write to it. Not used for Openssl truststores.
`gateway.truststore.crlLocations.*`	list of properties with a common prefix	-	List of CRLs locations. Can contain URLs, local files and wildcard expressions. Not used for Openssl truststores. *(runtime updateable)*

Property name	**Type**	**Default value / mandatory**	**Description**
`gateway.truststore.crlMode`	[REQUIRE, IF_VALID, IGNORE]	`IF_VALID`	General CRL handling mode. The IF_VALID setting turns on CRL checking only in case the CRL is present.
`gateway.truststore.crlUpdateInterval`	integer number	`600`	How often CRLs should be updated, in seconds. Set to negative value to disable refreshing at runtime. *(runtime updateable)*
`gateway.truststore.ocspCacheTtl`	integer number	`3600`	For how long the OCSP responses should be locally cached in seconds (this is a maximum value, responses won't be cached after expiration)
`gateway.truststore.ocspDiskCache`	filesystem path	-	If this property is defined then OCSP responses will be cached on disk in the defined folder.
`gateway.truststore.ocspLocalResponders.<NUMBER>`	list of properties with a common prefix	-	Optional list of local OCSP responders
`gateway.truststore.ocspMode`	[REQUIRE, IF_AVAILABLE, IGNORE]	`IF_AVAILABLE`	General OCSP ckecking mode. REQUIRE should not be used unless it is guaranteed that for all certificates an OCSP responder is defined.
`gateway.truststore.ocspTimeout`	integer number	`10000`	Timeout for OCSP connections in miliseconds.
`gateway.truststore.revocationOrder`	[CRL_OCSP, OCSP_CRL]	`OCSP_CRL`	Controls overal revocation sources order
`gateway.truststore.revocationUseAll`	[true, false]	`false`	Controls whether all defined revocation sources should be always checked, even if the first one already confirmed that a checked certificate is not revoked.

4.3.2 Scalability settings

To fine-tune the operational parameters of the embedded Jetty server, you can set advanced HTTP server parameters. See [?informaltable] 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 the 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.

4.3.3 Dynamic registration of Vsites

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

```
gateway.registration.enable=true
gateway.registration.secret=<your secret>
```

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 This request must contain a parameter "secret" which matches the value configured in the gateway.properties file

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.

4.3.4 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). If the Gateway is configured to do SSL authentication, 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
```

4.3.5 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.
`gateway.registration.secret`	string	-	Required secret for dynamic registration.
--- *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.

Property name	**Type**	**Default value / mandatory**	**Description**
`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 → Site client ---	
`gateway.client.chunked`	[true, false]	`true`	Controls whether chunked passing of HTTP requests to backend sites is supported.
`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.

Property name	**Type**	**Default value / mandatory**	**Description**
		--- 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.

Property name	**Type**	**Default value / mandatory**	**Description**
`gateway.httpServer.CORS_allowedHeaders`	string	`*`	CORS: comma separated list of allowed HTTP headers (default: any)
`gateway.httpServer.CORS_allowedMethods`	string	`GET,PUT,POST,DELETE,HEAD`	CORS: comma separated list of allowed HTTP verbs.
`gateway.httpServer.CORS_allowedOrigins`	string	`*`	CORS: allowed script origins.
`gateway.httpServer.CORS_chainPreflight`	[true, false]	`false`	CORS: whether preflight OPTION requests are chained (passed on) to the resource or handled via the CORS filter.
`gateway.httpServer.CORS_exposedHeaders`	string	`Location,Content-Type`	CORS: comma separated list of HTTP headers that are allowed to be exposed to the client.

Property name	Type	Default value / mandatory	Description
`gateway.httpServer.disabledCipherSuites`	string	*empty string*	Space separated list of SSL cipher suites to be disabled. Names of the ciphers must adhere to the standard Java cipher names, available here: http://docs.oracle.com/-javase/8/docs/technotes/-guides/security/-SunProviders.html#SupportedCipherSuites
`gateway.httpServer.enableCORS`	[true, false]	`false`	Control whether Cross-Origin Resource Sharing is enabled. Enable to allow e.g. accesing REST services from client-side JavaScript.
`gateway.httpServer.enableHsts`	[true, false]	`false`	Control whether HTTP strict transport security is enabled. It is a good and strongly suggested security mechanism for all production sites. At the same time it can not be used with self-signed or not issued by a generally trusted CA server certificates, as with HSTS a user can't opt in to enter such site.
`gateway.httpServer.fastRandom`	[true, false]	`false`	Use insecure, but fast pseudo random generator to generate session ids instead of secure generator for SSL sockets.
`gateway.httpServer.gzip.enable`	[true, false]	`false`	Controls whether to enable compression of HTTP responses.
`gateway.httpServer.gzip.minGzipSize`	integer number	`100000`	Specifies the minimal size of message that should be compressed.
`gateway.httpServer.highLoadConnections`	integer number	`200`	deprecated
`gateway.httpServer.lowResourceMaxIdleTime`	integer >= 1	`100`	deprecated

Property name	Type	Default value / mandatory	Description
`gateway.httpServer.maxConnections`	integer >= 0	`0`	Maximum number of incoming connections to this server. If set to a value larger than 0, incoming connections will be limited to that number. Default is 0 = unlimited.
`gateway.httpServer.maxIdleTime`	integer >= 1	`200000`	Time (in ms.) before an idle connection will time out. It should be large enough not to expire connections with slow clients, values below 30s are getting quite risky.
`gateway.httpServer.maxThreads`	integer number	`255`	Maximum number of threads to have in the thread pool for processing HTTP connections. Note that this number will be increased with few additional threads to handle connectors.
`gateway.httpServer.minThreads`	integer >= 1	`1`	Minimum number of threads to have in the thread pool for processing HTTP connections. Note that this number will be increased with few additional threads to handle connectors.
`gateway.httpServer.requireClientAuthn`	[true, false]	`true`	Controls whether the SSL socket requires client-side authentication.
`gateway.httpServer.useNIO`	[true, false]	`true`	DEPRECATED, no effect
`gateway.httpServer.wantClientAuthn`	[true, false]	`true`	Controls whether the SSL socket accepts (but does not require) client-side authentication.
`gateway.httpServer.xFrameAllowed`	string	`http://localhost`	URI origin that is allowed to embed web interface inside a (i)frame. Meaningful only if the xFrameOptions is set to *allowFrom*. The value should be in the form: *http[s]://host[:port]*

Property name	Type	Default value / mandatory	Description
`gateway.httpServer.xFrameOptions`	[deny, sameOrigin, allowFrom, allow]	deny	Defines whether a clickjacking prevention should be turned on, by insertionof the X-Frame-Options HTTP header. The *allow* value disables the feature. See the RFC 7034 for details. Note that for the *allowFrom* you should define also the xFrameAllowed option and it is not fully supported by all the browsers.

4.4 Require end-user certificates

Using client certificates for end-user authentication are not required or recommended. If you still want to require end-users to have a certificate, end-user certificates, the Gateway can be configured accordingly. Set the following in `gateway.properties`

```
gateway.httpServer.requireClientAuthn=true
```

4.4.1 Logging

UNICORE uses Log4j (version 2) as its logging framework, and comes with an example configuration file (CONF/logging.properties)

Please refer to the Log4j documentation for more information https://logging.apache.org/log4j/2.x/manual/configuration.html

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

unicore.gateway	General Gateway logging
unicore.security	Certificate details and other security
org.apache.http	Outgoing HTTP to the backend services

5 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

6 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.

6.1 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
```

6.2 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
```

7 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.

7.1 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.

7.2 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.

8 Building the Gateway from source

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

```
git clone https://github.com/UNICORE-EU/gateway.git
cd gateway
```

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

```
mvn install
```

which compiles the code and runs the tests.

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