eIDAS-Node Installation and Configuration Guide v2.8

Table of Contents

1 Introduction 7
1.1 Document structure 7
1.2 Purpose 7
1.3 Document aims 7
1.4 Other technical reference documentation 8
1.5 eIDAS Technical specifications and software provided 8
1.5.1 Further information 8

2 Product Overview 10
2.1 Package 10
2.2 Modules 10

3 Preparing the installation 14
3.1 Configuring the JVM 14
3.1.1 Oracle Java JCE Unlimited Strength Jurisdiction Policy 14
3.1.2 OpenJDK 14
3.1.3 IBM SDK Java 14
3.1.4 JDK17 14
3.1.5 Installing a security provider 15
3.2 Configuring the application server 17
3.2.1 Configuring Wildfly 23.0.2 Final Server (Servlet-Only Distribution) 17
3.2.2 Configuring WebSphere Liberty Profile 17
3.3 Enabling logging 18
3.3.1 Configuring audit logging 18
3.3.2 Organisation of logs 26

4 Configuring the software 28
4.1 JVM system properties 28
4.1.1 HTTP forwarding through Proxy 28
4.2 Configuring the project 28
4.2.1 Setup configuration directory 28
4.2.2 Setting up your Keystore 29
4.2.3 Configuring with Basic Setup 29
4.3 eIDAS-Node configuration files: eidas.xml 29
4.3.1 General purpose parameters 30
4.3.2 eIDAS-Node Connector configuration 31
4.3.3 eIDAS-Node Proxy Service configuration 34
4.3.4 Additional configuration — Security policies 36
4.4 eIDAS-Node Configuration Files: SAMLEngine.xml 38
4.4.1 SamlEngineConf 38
4.4.2 SignatureConf 38
4.4.3 EncryptionConf 40
4.4.4 ProtocolProcessorConf 40
4.4.5 ClockConf 41
4.5 Additional Configuration — SignModule_Service.xml and SignModule_Connector.xml . 41
4.6 Additional Configuration — Anti-replay Cache and Correlation Map Configuration 42
4.7 Error Codes and Error Messages 43
4.7.1 Different language files have different audiences in mind. 44
4.7.2 Multi language support 44
4.8 Configuring non-node components 44
4.8.1 Specific properties 44
4.8.2 Demo Service Provider 44
4.8.3 Demo Identity Provider 44

5 Building and deploying the software 45
5.1 Tomcat v9.0.85 server deployment 45
5.2 WildFly 23.0.2 Final Server (Servlet-Only Distribution) deployment 47
5.3 WebLogic Server 14.1.1.0.0 deployment 48
5.4 WebSphere Liberty Server v21.0.0.5 (WebProfile 8) deployment 49
5.5 Monolithic Deployment 51

6 Verifying the installation 52

6.1 Tomcat 9.0.85 52
6.2 WildFly 23.0.2 Final Server (Servlet-Only Distribution) 52
6.3 WebLogic 52
6.4 WebSphere Liberty Server 52
6.5 Configuration files 52

7 Advanced configuration for production environments 54
7.1 Clustering environment 54
7.1.1 Load balancer 54
7.2 Configuring Tomcat 54
7.2.1 Setting AJP ports 54
7.2.2 Apache HTTPD 55
7.3 Check your installation 56
7.4 eIDAS-Node compliance 56
7.5 Ignite shared map definitions 58

8 Appendix A: eIDAS Levels of Assurance 59

9 Appendix B: User consent 60

10 Appendix C: Ignite proposed configuration 61
10.1 Enable Cache Expiry Policy 69
10.2 Enable TLSv1.2 Ignite nodes 69
10.3 Enable Ignite Update Notifier 70

11 Appendix D: Installation Frequently Asked Questions 71

Document history

Version	Date	Reason for modification	Modified by
1.0	26/11/2015	Modifications to align with the eIDAS technical specifications.	DIGIT
1.1	09/09/2016	• Configuration improvements including support for Tomcat 8. • Removal of Attribute Provider. • Documentation of improvements included in Release 1.1 (see Release notes for eIDAS-Node version 1.1).	DIGIT
1.2	20/01/2017	• Configuration and stability improvements. • Documentation of improvements included in Release 1.2.0 (see Release notes for eIDAS-Node version 1.2.0).	DIGIT
1.3	08/06/2017	• Modifications to align with changes in Technical Specifications version 1.1. • Bug fixes and configuration improvements (for details please see the Version 1.3.0 Release Notes). • Documentation improvements to remove eIDAS-Nodes error codes and place in separate document *eIDAS Error Codes*.	DIGIT
1.4	06/10/2017	• Restructuring of reference documentation • Modifications to remove support for JBoss6. • Support WebLogic 12.2 family of servers. • Amend filename conventions to change '\' to '/'.	DIGIT
2.0	11/04/2018	• Changes in supported application servers; • Configuration and stability improvements; • Architectural changes (separation of Specific Connector and Specific Proxy Service). (for details see the Version 2.0 Release Notes and the *eIDAS-Node Migration Guide*)	DIGIT
2.1	05/07/2018	Reuse of document policy updated and version changed to match the corresponding Release.	DIGIT
2.2	14/09/2018	Document updated to reflect current installation and configuration.	DIGIT
2.3	20/06/2019	Document updated to reflect current installation and configuration.	DIGIT
2.4	06/12/2019	Document updated to reflect current installation and configuration.	DIGIT
2.5	11/12/2020	eIDAS-Node 2.5 release	DIGIT
2.6	15/04/2022	eIDAS-Node 2.6 release	DIGIT
2.7	01/09/2023	eIDAS-Node 2.7 release	DIGIT

Disclaimer

List of abbreviations

The following abbreviations are used within this document.

Abbreviation	Meaning
eIDAS	electronic Identification and Signature. The Regulation (EU) N°910/2014 governs electronic identification and trust services for electronic transactions in the internal market to enable secure and seamless electronic interactions between businesses, citizens and public authorities
IdP	Identity Provider. An institution that verifies the citizen's identity and issues an electronic ID.
LoA	Level of Assurance (LoA) is a term used to describe the degree of certainty that an individual is who they say they are at the time they present a digital credential.
MW	Middleware. Architecture of the integration of eIDs in services, with a direct communication between SP and the citizen's PC without any central server. The term also refers to the piece of software of this architecture that executes on the citizen's PC.
MS	Member State
SAML	Security Assertion Markup Language
SP	Service Provider

List of definitions

The following definitions are used within this document.

Term	Meaning
Audit	A function which seeks to validate that controls are in place, adequate for their purposes, and which reports inadequacies to appropriate levels of management.
Audit log	An audit log is a chronological sequence of audit records, each of which contains evidence directly as a result of the execution of a business process or system function
Basic Setup	The basic configuration and Demo tools provided in a package to setup and run an eIDAS-Node strictly for demo purposes only.
Demo tools	Demo tools comprise the Demo SP, Demo IDP, Specific Connector and Specific Proxy Service included in the integration package. These components are not

Term	Meaning
	production ready and should not be deployed or used in production environments.
eIDAS-Node	An eIDAS-Node is an application component that can assume two different roles depending on the origin of a received request. See eIDAS-Node Connector and eIDAS-Node Proxy Service.
eIDAS-Node Connector	The eIDAS-Node assumes this role when it is located in the **Service Provider's** Member State. In a scenario with a Service Provider asking for authentication, the eIDAS-Node Connector receives the authentication request from the Service Provider and forwards it to the eIDAS-Node of the citizen's country. This was formerly known as S-PEPS.
eIDAS-Node Proxy Service	The eIDAS-Node assumes this role when it is located in the **citizen's** Member State. The eIDAS-Node Proxy Service receives authentication requests from an eIDAS-Node of another MS (their eIDAS-Node Connector). The eIDAS-Node Proxy-Service also has an interface with the national eID infrastructure and triggers the identification and authentication for a citizen at an identity and/or attribute provider. This was formerly known as C-PEPS.

References

[1] ISO/IEC 27002 - Information technology -- Security techniques -- Code of practice for information security management, section 10.10, 2005 (www.iso.org)

[2] BSI PD008: Legal Admissibility and Evidential Weight of Information Stored Electronically, British Standards Institution, 1999

[3] COBIT (Control Objectives for Information and related Technology) from Information Systems Audit and Control Association (https://www.isaca.org/resources/cobit)

[4] ICT-PSP/2007/1 – STORK 1 : D5.7.3 Functional Design for PEPS, MW models and interoperability

[5] K. Kent, M. Souppaya. Guide to Computer Security Log Management. Recommendations of the National Institute of Standards and Technology, NIST Special Publication 800-92, September 2006

[6] SANS Consensus Policy Resource Community - Information Logging Standard, https://www.sans.org/information-security-policy/

[7] NIST: An Introduction to Computer Security: The NIST Handbook, NIST Special Publication 800-12 Rev.1, June 2017, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf

[8] Common Criteria: Common Criteria for Information Technology Security Evaluation, Version 3.1, revision 4, September.2012 Part 2: Security Functional Components, http://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R4.pdf

[9] ENISA: Privacy Features of European eID Card Specifications, Version 1.0.1, January 2009, https://www.enisa.europa.eu/publications/eid-cards-en

1 Introduction

This document is intended for a technical audience consisting of developers, administrators and those requiring detailed technical information on how to configure, build and deploy the eIDAS-Node application.

The document describes the steps involved when implementing a Basic Setup and goes on to provide detailed information required for customisation and deployment.

1.1 Document structure

This document is divided into the following sections:

- Chapter 1 − *Introduction*: this section.
- Chapter 2 − *Product overview* describes the binaries and source code to be installed plus the configuration files.
- Chapter 3 − *Preparing the installation* describes the prerequisites for a successful installation, including the correct Java version, supported application servers, environmental variables to be set, keystores etc.
- Chapter 4 − *Configuring the software* describes all configuration settings.
- Chapter 5 − *Building and deploying the software* describes the steps to build and then to deploy the software on the supported servers. There are two main types of eIDAS-Node: Connector and Proxy Service.
- Chapter 6 − *Verifying the installation* shows the final structure of your application server's relevant directories, so that you can confirm that you have made the proper configurations.
- Chapter 7 − *Advanced configuration for production environments* provides detailed descriptions of the configurations to enable you to change specific aspects as required.
- Appendix A − *eIDAS Levels of Assurance* provides information on the three Levels of Assurance described in the Implementing Regulation.
- Appendix B − *User consent* provides a brief overview of the meaning of 'user consent' in the context of privacy legislation.
- Appendix C − *Ignite proposed configuration* provides specific information related to configuration of a cluster environment using Ignite.
- Appendix D − *Installation Frequently Asked Questions* provides answers to questions that may arise during your installation.

1.2 Purpose

The purpose of this document is to give a comprehensive view of eID and its components (in terms of binaries, source code and configuration files).

1.3 Document aims

The aims of this document are to:

- guide you through the preliminary steps involved when setting up your servers;
- guide you through setting up, compiling and running a project for a basic configuration with one instance of your Application Server;

- cover detailed configuration of eIDAS-Nodes;
- provide a checklist of files for each application server;
- show how to ensure eIDAS regulation compliance and provide a checklist of recommendations;
- describe the technologies and configurations used for testing the eIDAS-Node in cluster mode.

1.4 Other technical reference documentation

We recommend that you also familiarise yourself with the following eID technical reference documents which are available on **Digital Home > eID**

- *eIDAS-Node Installation, Configuration and Integration Quick Start Guide* describes how to quickly install a demo Service Provider, eIDAS-Node Connector, eIDAS-Node Proxy Service and demo IdP from the distributions in the release package. The distributions provide preconfigured eIDAS-Node modules for running on each of the supported application servers.
- *eIDAS-Node National IdP and SP Integration Guide* provides guidance by recommending one way in which eID can be integrated into your national eID infrastructure.
- *eIDAS-Node Demo Tools Installation and Configuration Guide* describes the installation and configuration settings for Demo Tools (SP and IdP) supplied with the package for basic testing.
- *eIDAS-Node and SAML* describes the W3C recommendations and how SAML XML encryption is implemented and integrated in eID. Encryption of the sensitive data carried in SAML 2.0 Requests and Assertions is discussed alongside the use of AEAD algorithms as essential building blocks.
- *eIDAS-Node Error and Event Logging* provides information on the eID implementation of error and event logging as a building block for generating an audit trail of activity on the eIDAS Network. It describes the files that are generated, the file format, the components that are monitored and the events that are recorded.
- *eIDAS-Node Security Considerations* describes the security considerations that should be taken into account when implementing and operating your eIDAS-Node scheme.
- *eIDAS-Node Error Codes* contains tables showing the error codes that could be generated by components along with a description of the error, specific behaviour and, where relevant, possible operator actions to remedy the error.

1.5 eIDAS Technical specifications and software provided

This software package is provided as a reference implementation in accordance with the *eIDAS Technical Specifications v1.2* available at *https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eIDAS+eID+Profile*.

1.5.1 Further information

For further information on the practical implementation of the features listed above, please refer to section 7.5 — *eIDAS-Node compliance* which describes the production mode for ensuring eIDAS regulation compliance.

Disclaimer: The users of the eIDAS-Node sample implementation remain fully responsible for its integration with back-end systems (Service Providers and Identity Providers), testing, deployment and operation. The support and maintenance of the sample implementation, as well as any other auxiliary services, are provided by the European Commission according to the terms defined in the European Union Public License (EUPL) at *https://joinup.ec.europa.eu/sites/default/files/custom-page/attachment/eupl_v1.2_en.pdf*

2 Product Overview

2.1 Package

The main product deliverables are EidasNodeConnector.war and EidasNodeProxy.war. These are web applications that can be deployed to most Java web containers on the market. The actual functionality is activated by configuration.

2.2 Modules

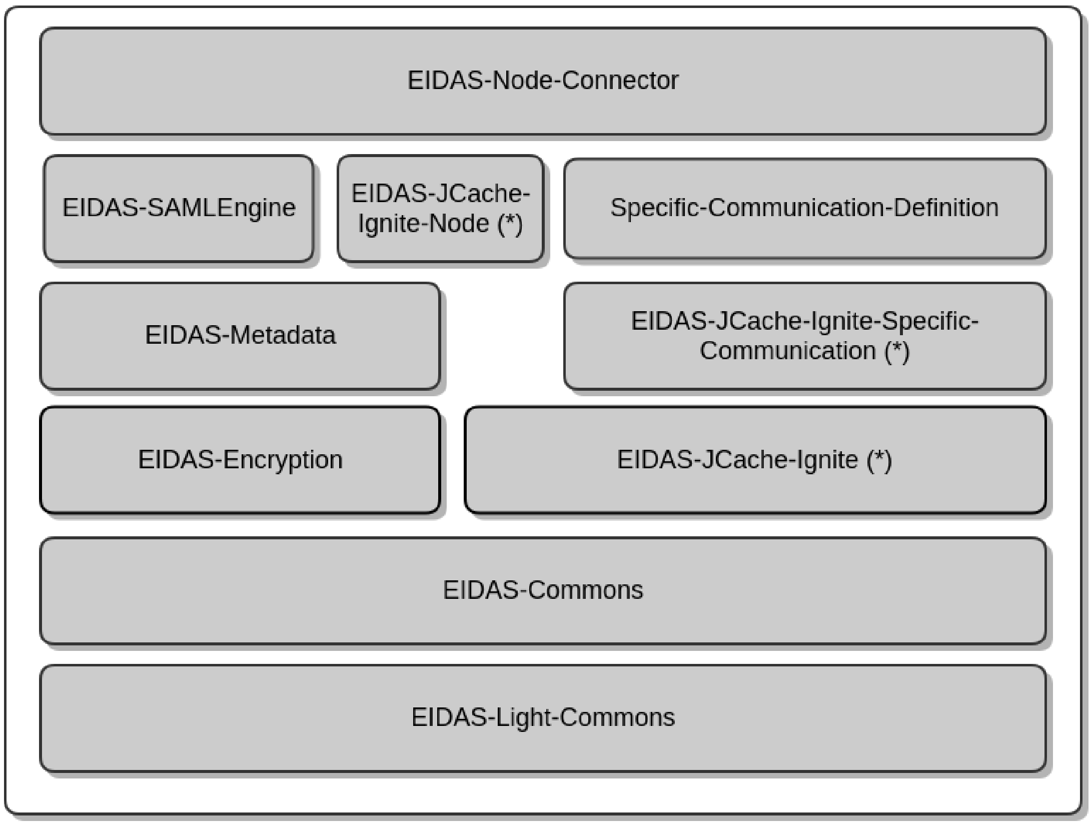
The software is composed of several modules. This section describes the binaries and source code to be installed plus the configuration files.

Table 1: List of modules

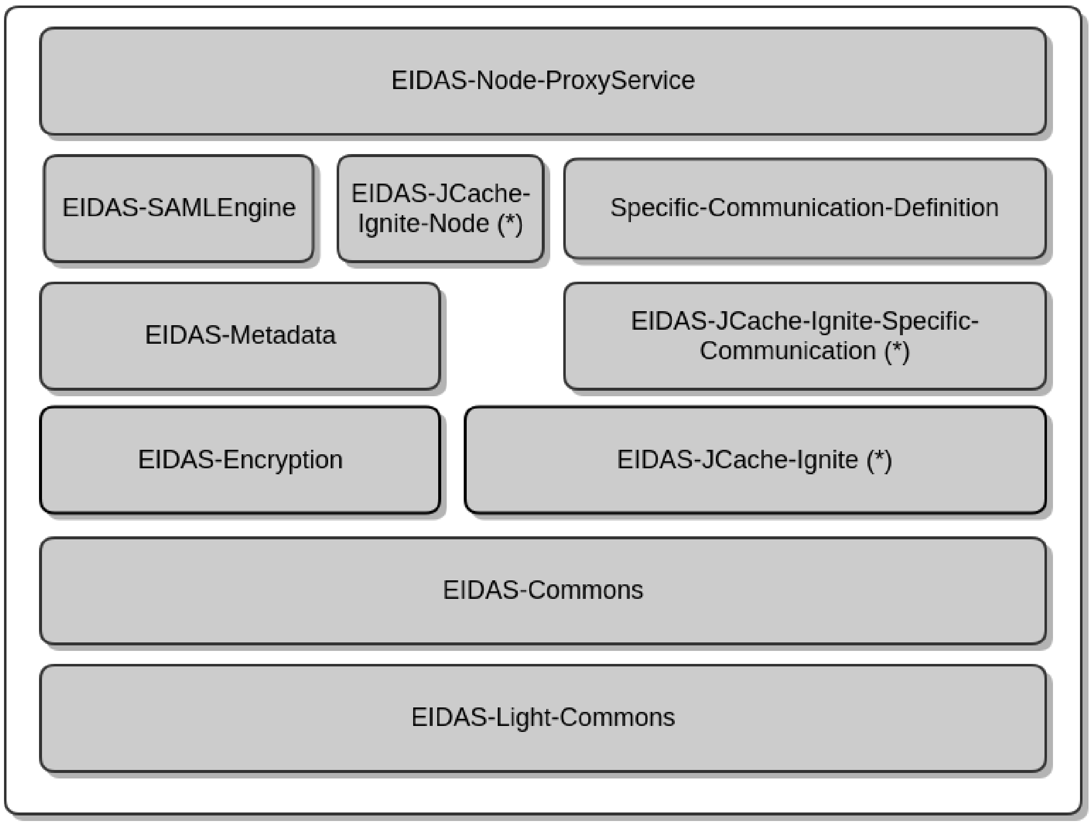
Module Name	Folder	Description
Parent	EIDAS-Parent	Module containing a consolidated and consistent location of the libraries and their version number to be used across the different modules.
Light Commons	EIDAS-Light-Commons	Light Common application component and utility classes used for implementing as basis for the EIDAS-Commons and MS Specific Connector and MS Specific Proxy Service modules.
Commons	EIDAS-Commons	Common Applications components and utility classes for implementing functionality of authentication service.
Encryption	EIDAS-Encryption	Encryption and signature dedicated module.
Metadata	EIDAS-Metadata	Implementation of metadata-related functionalities such as generation and fetching used in both EIDAS-SAMLEngine and eIDAS-Node.
SAMLEngine	EIDAS-SAMLEngine	Implementation of EIDAS SAML ProtocolEngine used in the eIDAS-Node.
JCache-Dev	EIDAS-JCache-Dev	Common code for Guava non-distributed JCache implementations
JCache-Dev-Node	EIDAS-JCache-Dev-Node	Adapts the implementation of Guava non-distributed maps to JCache used in eIDAS-Node caches.
JCache-Dev-Specific-Communication	EIDAS-JCache-Dev-Specific-Communication	Adapts the implementation of Guava non-distributed maps to

Module Name	Folder	Description
		JCache used in eIDAS-Node MS specific communication caches.
JCache-Ignite	EIDAS-JCache-Ignite	Common code for Ignite JCache implementations
JCache-Ignite-Node	EIDAS-JCache-Ignite-Node	Implementation of Ignite JCache for the eIDAS-Node caches
JCache-Ignite-Specific-Communication	EIDAS-JCache-Ignite-Specific-Communication	Implementation of Ignite JCache for the eIDAS-Node MS specific communication caches
Specific Communication Definition	EIDAS-SpecificCommunicationDefinition	The exchange definition (interfaces) and implementation used to formalise the exchange definition between the node and the Specific module.
Updater	EIDAS-Updater	Additional module used to change the configuration of a running eIDAS-Node in a testing environment. (To enable, web.xml must be updated.) Not to be used in production.
EidasNodeConnector	EIDAS-Node-Connector	eIDAS-Node Connector module
EidasNodeProxy	EIDAS-Node-Proxy	eIDAS-Node Proxy module
Basic Setup configuration	EIDAS-Config	Sample configuration as in 6.7

The figure below shows the dependencies between the installed modules.



* Interchangable by maven build profile



* Interchangable by maven build profile

Figure 1: Dependencies between the installed modules

3 Preparing the installation

This section provides instructions on how to deploy the project on Tomcat, Wildfly, WebLogic or WebSphere servers.

The appropriate JVM needs to be installed and configured first. If the selected application server includes an embedded JVM, the configuration still needs to be changed.

3.1 Configuring the JVM

The project is built by default using the **Java SDK** version **11.0.12**.

3.1.1 Oracle Java JCE Unlimited Strength Jurisdiction Policy

In Java 11, to verify the JCE security policy, the "crypto.policy" configuration in "JAVA_HOME/conf/security/java.security" file can be checked. By default, the configuration is "unlimited".

3.1.2 OpenJDK

Please be careful when using OpenJDK, OpenJDK release content is not guaranteed. Therefore some compatibility problems may be encountered while using OpenJDK with eIDAS.

3.1.3 IBM SDK Java

If the IBM-provided JVM is going to be used for the eIDAS-Node, it is necessary to ensure Java 11 SDK is selected.

3.1.4 JDK17

At this moment, no efforts have been done to build, run or test the software on JDK17.However it possible to run the connector and proxy JDK11 ignite build war under JDK17 after opening up following java modules:

```
# Ignite Shared Cache
export JAVA_OPTS="$JAVA_OPTS \
--add-opens=java.base/jdk.internal.access=ALL-UNNAMED \
--add-opens=java.base/jdk.internal.misc=ALL-UNNAMED \
--add-opens=java.base/sun.nio.ch=ALL-UNNAMED \
--add-opens=java.base/sun.util.calendar=ALL-UNNAMED \
--add-opens=java.management/com.sun.jmx.mbeanserver=ALL-UNNAMED \
--add-opens=jdk.internal.jvmstat/sun.jvmstat.monitor=ALL-UNNAMED \
--add-opens=java.base/sun.reflect.generics.reflectiveObjects=ALL-UNNAMED \
--add-opens=jdk.management/com.sun.management.internal=ALL-UNNAMED \
--add-opens=java.base/java.io=ALL-UNNAMED \
--add-opens=java.base/java.nio=ALL-UNNAMED \
--add-opens=java.base/java.net=ALL-UNNAMED \
--add-opens=java.base/java.util=ALL-UNNAMED \
--add-opens=java.base/java.util.concurrent=ALL-UNNAMED \
--add-opens=java.base/java.util.concurrent.locks=ALL-UNNAMED \
--add-opens=java.base/java.util.concurrent.atomic=ALL-UNNAMED \
--add-opens=java.base/java.lang=ALL-UNNAMED \
--add-opens=java.base/java.lang.invoke=ALL-UNNAMED \
--add-opens=java.base/java.math=ALL-UNNAMED \
--add-opens=java.sql/java.sql=ALL-UNNAMED \
--add-opens=java.base/java.lang.reflect=ALL-UNNAMED \
--add-opens=java.base/java.time=ALL-UNNAMED \
--add-opens=java.base/java.text=ALL-UNNAMED \
--add-opens=java.management/sun.management=ALL-UNNAMED \
--add-opens java.desktop/java.awt.font=ALL-UNNAMED"

# eIDAS Connector or Proxy
export JAVA_OPTS="$JAVA_OPTS \
--add-opens=java.xml/com.sun.org.apache.xalan.internal.xsltc.trax=ALL-
UNNAMED \
--add-opens=java.base/sun.security.x509=ALL-UNNAMED \
--add-opens=java.base/java.security.cert=ALL-UNNAMED \
--add-opens=java.xml/javax.xml.namespace=ALL-UNNAMED"
```

3.1.5 Installing a security provider

In order for the Node to be able to sign, verify and encrypt/decrypt data, one or more security providers need to be installed/configured that can perform those operations. In most cases this means that Bouncy Castle provider will have to be always installed/configured, since it provides operations others do not. However, this might be avoided if, for some reason, another security provider, that is installed/configured, performs all necessary cryptographic functionalities required by eIDAS technical specifications. Therefore, BouncyCastle is not installed directly through the code to allow this flexibility.

Additionally, when using Bouncy Castle as the primary security provider, specific issues may arise, such as compatibility issues with Ignite and revocation checking. To address this, it is necessary to add a new property to the Java options. The following property should be added:

```
--add-opens
org.bouncycastle.provider/org.bouncycastle.jcajce.provider.asymmetric.x509
=ALL-UNNAMED
```

3.1.5.1 PKCS12 or JKS Software Keystores with BouncyCastle

When using classic software keystores such as "PKCS12" or "JKS", the BouncyCastle provider will be needed, and to install the BouncyCastle provider via the `java.security` file (which can be found in the `$JAVA_HOME/conf/security` folder).

Note that in Java 11, additional to defining the provider in the java.security file, we have to define two Java options (`JAVA_OPTS`) in order to actually use the provider :

- *--module-path* to indicate the location/path of the provider jar
- *--add-modules* to add the module corresponding to the provider (defined by Java 9 standards in module-info file present in the jar)

For example, to install the BouncyCastle provider in Java 11, we would have to:

- Add a line to the `java.security` file (in this example 13th position, but could be another):

```
security.provider.13=org.bouncycastle.jce.provider.BouncyCastleProv
    ider
```

- Add the following Java options:

```
export JAVA_OPTS="$JAVA_OPTS --module-path /path/to/library/bcprov-
    jdk18on-176.jar"
export JAVA_OPTS="$JAVA_OPTS --add-modules
    org.bouncycastle.provider"
```

Recommended	**Bouncy Castle**	**bcprov-jdk18on-176.jar**	**1.76**	**Signed version obtained from https://www.bouncycastle.org/download/bcprov-jdk18on-176.jar**

3.1.5.2 PKCS11

When using keystores such as "PKCS11", a SunPKCS11 provider will be needed, this can also be configured by being added to the `java.security` file.

```
security.provider.1=SunPKCS11 /path/to/config/pkcs11.cfg
```

The content of the pkcs11.cfg will depend on the needs, here is an example of multiples attributes that can be used

```
name = providerSuffix
library = /path/to/lib/libpkcs11.so

slot = 0
attributes = compatibility
disabledMechanisms = {
  SecureRandom
}
```

PKCS11 keystore configuration can be found in the *eIDAS-Node and SAML 2.7* document.

More information about security providers can be found at

- **Oracle Security Developer's Guide: Configuring Java Security Providers** https://docs.oracle.com/en/java/javase/11/security/howtoimplaprovider.html#GUID-FB9C6DB2-DE9A-4EFE-89B4-C2C168C5982D
- **Oracle Security Developer's Guide: The SunPKCS11 provider** https://docs.oracle.com/en/java/javase/11/security/pkcs11-reference-guide1.html#GUID-97F1E537-CB59-4C7F-AB6B-05D4DBD69AC0

Note: When the private key is accessed through the SunPKCS11 provider for RSA signing, the Java Cryptographic Architecture should be able to handle this.
Both Bouncy Castle and SunPKCS11 use the same JCAName "RSASSA-PSS" accompanied by a PSSParameterSpec object that defines the algorithms.

Note: Beware that, if you add a new Security Provider you might be able to remove other providers, such as Bouncycastle, but only if the remaining set of configured providers can provide every algorithm that MUST be SUPPORTED in the eIDAS Cryptographic Requirements v.1.2.

3.2 Configuring the application server

The following is a list of the supported servers.

Table 2: Supported servers

Application Server	Supported version(s)
Tomcat	9.0.85
GlassFish	No version currently supported
WildFly	23.0.2 Final (Servlet-Only Distribution)
WebLogic	14.1.1.0.0 (14c)
WebSphere Liberty Profile	Liberty 21.0.0.5 (Web Profile 8)

3.2.1 Configuring Wildfly 23.0.2 Final Server (Servlet-Only Distribution)

In order for the node, to work correctly with Wildfly, a module for BouncyCastle has to be configured, therefore :

```
<module name="org.bouncycastle" xmlns="urn:jboss:module:1.9">
    <resources>
        <resource-root path="bcprov-jdk18on-176.jar"/>
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="org.jboss.logging"/>
        <module name="org.jboss.modules"/>
    </dependencies>
    <provides>
        <service name="java.security.Provider">
            <with-class
name="org.bouncycastle.jce.provider.BouncyCastleProvider"/>
            <with-class
name="org.bouncycastle.pqc.jcajce.provider.BouncyCastlePQCProvider"/>
        </service>
    </provides>
</module>
```

Code Block 1 ${WILDFLY_SERVER_HOME}/modules/system/layers/base/org/bouncycastle/main/module.xml

The module.xml file that points to a bcprov-jdk18on-176.jar downloaded in the same folder.

(Note the bcprov-jdk18on-176.jar is a signed version obtained from https://www.bouncycastle.org/download/bcprov-jdk18on-176.jar)

3.2.2 Configuring WebSphere Liberty Profile

The application may be deployed by copying the war files under *$SERVER_HOME/dropins* directory.

3.2.2.1 Websphere server feature

Webprofile feature must be disabled for Websphere Liberty Profile from server.xml to avoid error "Exception occurred during processing request".

Therefore, in the file "${SERVER_HOME}/usr/servers/${SERVER}/server.xml", comment the "webProfile-8.0" feature line, and add the "jsp-2.3" feature as in the example below :

```
<featureManager>
    <!--<feature>webProfile-8.0</feature>-->
    <feature>jsp-2.3</feature>
</featureManager>
```

3.2.2.2 Websphere additional configuration

For Websphere Liberty Profile, some additional configurations need to be added in the server.xml file, present at: ${SERVER_HOME}/usr/servers/${SERVER}/server.xml

In order for WebSphere, to correctly display eIDAS error messages, the property com.ibm.ws.webcontainer.enableErrorExceptionTypeFirst="true" has to be added to the server.xml configuration file.

This can be achieved by adding the following line :

```
<webContainer
com.ibm.ws.webcontainer.enableErrorExceptionTypeFirst="true"/>
```

If the configuration entry is not set, WebSphere will deal with error page handling by first giving preference to HTTP error code and not to exceptions, which causes it to display an error page without the eIDAS error code /message.

WebSphere also does not correctly handles context-root. In order to improve that behaviour, add this property in server.xml file of websphere:

```
<webContainer com.ibm.ws.webcontainer.redirectcontextroot="true"/>
```

If set to true, and a request is made to the context root of an application with a missing trailing slash, the WebContainer appends the trailing slash. The WebContainer redirects to the URL with the appended slash before it applies any servlet filters defined in the application.

3.3 Enabling logging

To enable audit logging of the communications between eIDAS-Node Proxy Service and eIDAS-Node Connector, please refer to the *eIDAS-Node Error and Event Logging* guide.

3.3.1 Configuring audit logging

Edit the project eIDAS-Node-Connector file: logback.xml (located in the resources directory) and add the following lines:

```xml
<?xml version="1.0" encoding="UTF-8" ?>

<!--
        NOTE :
            the environment variable LOG_HOME could be set to indicate the
directory containing the log files
            the log configuration files will be scanned periodically each
30 minutes
            LOG level is defined as below :
                Default level : INFO
                    Console appender (STDOUT)    : inherits from default
                    eIDASNodeConnectorDetail appender         : INFO
                    eIDASNodeConnectorSystem appender         : INFO
                    eIDASNodeConnectorSecurity appender       : INFO
-->

<configuration scan="true" scanPeriod="30 minutes">

    <!--
        This define the CONSOLE appender - the level of the console
appender is based on the root level
    -->
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d\{yyyy-MM-dd; HH:mm:ss.SSS\} \[%thread\] %-5level
%logger\{66\} %marker -%X\{sessionId\} -%X\{remoteHost\} -%msg%n</pattern>
        </encoder>
    </appender>

    <!--
        This define the FULL Detailed log file appender - the level of the
console appender is INFO by default
    -->
    <appender name="eIDASNodeConnectorDetail"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>$\{LOG_HOME\}/eIDASNodeConnectorDetail.log</file>

        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>INFO</level>
        </filter>
        <encoder
class="eu.eidas.node.logging.logback_integrity.HashPatternLayoutEncoder">
            <pattern>%d\{yyyy-MM-dd; HH:mm:ss.SSS\} \[%thread\] %-5level
%logger\{66\} %marker -%X\{sessionId\} -%X\{remoteHost\} -%msg%n</pattern>
        </encoder>
        <param name="Append" value="true" />
        <triggeringPolicy
class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
            <maxFileSize>500KB</maxFileSize>
        </triggeringPolicy>
        <!-- Support multiple-JVM writing to the same log file -->
        <prudent>true</prudent>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>$\{LOG_HOME\}/eIDASNodeConnectorDetail.%d\{yyyy-MM-
dd\}.log</fileNamePattern>
            <maxHistory>14</maxHistory>
        </rollingPolicy>
    </appender>

    <!--
        This define the SYSTEM Detailed log file appender - the default
Filter is inherited from root level
    -->
    <appender name="eIDASNodeConnectorSystem"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>$\{LOG_HOME\}/eIDASNodeConnectorSystem.log</file>

        <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
            <evaluator
class="ch.qos.logback.classic.boolex.OnMarkerEvaluator">
                <marker>SYSTEM</marker>
            </evaluator>
            <onMismatch>DENY</onMismatch>
            <onMatch>ACCEPT</onMatch>
        </filter>
        <encoder
class="eu.eidas.node.logging.logback_integrity.HashPatternLayoutEncoder">
            <pattern>%d\{yyyy-MM-dd; HH:mm:ss.SSS\} \[%thread\] %-5level
%logger\{66\} %marker -%X\{sessionId\} -%X\{remoteHost\} -%msg%n</pattern>
        </encoder>
        <param name="Append" value="true" />
        <!-- Support multiple-JVM writing to the same log file -->
        <prudent>true</prudent>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>$\{LOG_HOME\}/eIDASNodeConnectorSystem.%d\{yyyy-MM-
dd\}.log</fileNamePattern>
            <maxHistory>14</maxHistory>
        </rollingPolicy>
    </appender>

    <!--
        This define the SECURITY Detailed log file appender - the default
Filter is inherited from root level
    -->
    <appender name="eIDASNodeConnectorSecurity"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>$\{LOG_HOME\}/eIDASNodeConnectorSecurity.log</file>

        <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
            <evaluator
class="ch.qos.logback.classic.boolex.OnMarkerEvaluator">
                <marker>SECURITY_SUCCESS</marker>
                <marker>SECURITY_WARNING</marker>
                <marker>SECURITY_FAILURE</marker>
            </evaluator>
            <onMismatch>DENY</onMismatch>
            <onMatch>ACCEPT</onMatch>
        </filter>
        <encoder
class="eu.eidas.node.logging.logback_integrity.HashPatternLayoutEncoder">
            <pattern>%d\{yyyy-MM-dd; HH:mm:ss.SSS\} \[%thread\] %-5level
%logger\{66\} %marker -%X\{sessionId\} -%X\{remoteHost\} -%msg%n</pattern>
        </encoder>
        <param name="Append" value="true" />
        <!-- Support multiple-JVM writing to the same log file -->
        <prudent>true</prudent>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>$\{LOG_HOME\}/eIDASNodeConnectorSecurity.%d\{yyyy-MM-
dd\}.log</fileNamePattern>
            <maxHistory>14</maxHistory>
        </rollingPolicy>
    </appender>

    <!--
```

```
        This define the SAML exchange Detailed log file appender - the
default Filter is inherited from root level
    -->
    <appender name="eIDASNodeConnectorSAMLExchange"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>$\{LOG_HOME\}/eIDASNodeConnectorSAMLExchange.log</file>

        <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
            <evaluator
class="ch.qos.logback.classic.boolex.OnMarkerEvaluator">
                <marker>SAML_EXCHANGE</marker>
            </evaluator>
            <onMismatch>DENY</onMismatch>
            <onMatch>ACCEPT</onMatch>
        </filter>
        <encoder
class="eu.eidas.node.logging.logback_integrity.HashPatternLayoutEncoder">
            <pattern>%d\{yyyy-MM-dd; HH:mm:ss.SSS\} \[%thread\] %-5level
%logger\{66\} %marker -%X\{sessionId\} -%X\{remoteHost\} -%msg%n</pattern>
        </encoder>
        <param name="Append" value="true" />
        <!-- Support multiple-JVM writing to the same log file -->
        <prudent>true</prudent>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>$\{LOG_HOME\}/eIDASNodeConnectorSAMLExchange.%d\{yyyy-MM-
dd\}.log</fileNamePattern>
            <maxHistory>14</maxHistory>
        </rollingPolicy>
    </appender>

    <!--
        This define the log file appender for the logging of the full
request and response messages
    -->
    <appender name="eIDASNodeConnectorFullMessageExchange"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>$\{LOG_HOME\}/eIDASNodeConnectorFullMsgExchange.log</file>

        <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
            <evaluator
class="ch.qos.logback.classic.boolex.OnMarkerEvaluator">
                <marker>FULL_MESSAGE_EXCHANGE</marker>
            </evaluator>
            <onMismatch>DENY</onMismatch>
            <onMatch>ACCEPT</onMatch>
        </filter>
        <encoder
class="eu.eidas.node.logging.integrity.HashPatternLayoutEncoder">
            <pattern>%d\{yyyy-MM-dd'T'HH:mm:ss.SSS'Z',GMT\} \[%thread\] %-
5level %logger\{66\} %marker -%X\{remoteHost\}:
%n%replace(%msg)\{'\[\r\n\]+',''\}%n</pattern>
        </encoder>
        <param name="Append" value="true" />
        <!-- Support multiple-JVM writing to the same log file -->
        <prudent>true</prudent>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>$\{LOG_HOME\}/eIDASNodeConnectorFullMsgExchange.%d\{yyyy-
MM-dd,GMT\}.log</fileNamePattern>
            <maxHistory>14</maxHistory>
        </rollingPolicy>
    </appender>
    <!--
        This define the API fine grained level
    -->
    <logger name="org.opensaml">
        <level value="ERROR" />
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeConnectorDetail"/>
    </logger>
    <logger name="com.opensymphony.xwork2">
        <level value="WARN"/>
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeConnectorDetail"/>
    </logger>
    <logger name=" org.apache.struts2">
        <level value="WARN"/>
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeConnectorDetail"/>
    </logger>
    <logger name="org.springframework">
        <level value="WARN" />
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeConnectorDetail"/>
    </logger>
    <logger name="org.apache.xml.security">
        <level value="WARN" />
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeConnectorDetail"/>
    </logger>

    <logger name="eu.eidas.communication.requests">
        <level value="info" />
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeConnectorDetail"/>
    </logger>

    <logger name="eu.eidas.communication.responses">
        <level value="info" />
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeConnectorDetail"/>
    </logger>

    <!--
        The root level is set to debug for development purposes, for
production environment it could be set to INFO
    -->
    <root level="INFO">
        <appender-ref ref="STDOUT" />
        <appender-ref ref="eIDASNodeConnectorSystem" />
        <appender-ref ref="eIDASNodeConnectorSecurity" />
        <appender-ref ref="eIDASNodeConnectorDetail" />
        <appender-ref ref="eIDASNodeConnectorSAMLExchange" />
        <appender-ref ref="eIDASNodeConnectorFullMessageExchange" />
    </root>
</configuration>
```

Edit the project eIDAS-Node-Proxy file: logback.xml (located in the resources directory) and add the following lines:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!--
        NOTE :
            the environment variable LOG_HOME could be set to indicate the
directory containing the log files
            the log configuration files will be scanned periodically each
30 minutes
            LOG level is defined as below :
                Default level : INFO
                    Console appender (STDOUT)    : inherits from default
                    eIDASNodeProxyDetail appender         : INFO
                    eIDASNodeProxySystem appender         : INFO
                    eIDASNodeProxySecurity appender       : INFO
-->

<configuration scan="true" scanPeriod="30 minutes">

    <!--
        This define the CONSOLE appender - the level of the console
appender is based on the root level
    -->
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d\{yyyy-MM-dd; HH:mm:ss.SSS\} \[%thread\] %-5level
%logger\{66\} %marker -%X\{sessionId\} -%X\{remoteHost\} -%msg%n</pattern>
        </encoder>
    </appender>

    <!--
        This define the FULL Detailed log file appender - the level of the
console appender is INFO by default
    -->
    <appender name="eIDASNodeProxyDetail"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>$\{LOG_HOME\}/eIDASNodeProxyDetail.log</file>

        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>INFO</level>
        </filter>
        <encoder
class="eu.eidas.node.logging.logback_integrity.HashPatternLayoutEncoder">
            <pattern>%d\{yyyy-MM-dd; HH:mm:ss.SSS\} \[%thread\] %-5level
%logger\{66\} %marker -%X\{sessionId\} -%X\{remoteHost\} -%msg%n</pattern>
        </encoder>
        <param name="Append" value="true" />
        <triggeringPolicy
class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
            <maxFileSize>500KB</maxFileSize>
        </triggeringPolicy>
        <!-- Support multiple-JVM writing to the same log file -->
        <prudent>true</prudent>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>$\{LOG_HOME\}/eIDASNodeProxyDetail.%d\{yyyy-
MM-dd\}.log</fileNamePattern>
            <maxHistory>14</maxHistory>
        </rollingPolicy>
    </appender>

    <!--
        This define the SYSTEM Detailed log file appender - the default
Filter is inherited from root level
    -->
    <appender name="eIDASNodeProxySystem"
class="ch.qos.logback.core.rolling.RollingFileAppender">
```

```
        <file>$\{LOG_HOME\}/eIDASNodeProxySystem.log</file>

        <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
            <evaluator
class="ch.qos.logback.classic.boolex.OnMarkerEvaluator">
                <marker>SYSTEM</marker>
            </evaluator>
            <onMismatch>DENY</onMismatch>
            <onMatch>ACCEPT</onMatch>
        </filter>
        <encoder
class="eu.eidas.node.logging.logback_integrity.HashPatternLayoutEncoder">
            <pattern>%d\{yyyy-MM-dd; HH:mm:ss.SSS\} \[%thread\] %-5level
%logger\{66\} %marker -%X\{sessionId\} -%X\{remoteHost\} -%msg%n</pattern>
        </encoder>
        <param name="Append" value="true" />
        <!-- Support multiple-JVM writing to the same log file -->
        <prudent>true</prudent>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>$\{LOG_HOME\}/eIDASNodeProxySystem.%d\{yyyy-
MM-dd\}.log</fileNamePattern>
            <maxHistory>14</maxHistory>
        </rollingPolicy>
    </appender>

    <!--
        This define the SECURITY Detailed log file appender - the default
Filter is inherited from root level
    -->
    <appender name="eIDASNodeProxySecurity"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>$\{LOG_HOME\}/eIDASNodeProxySecurity.log</file>

        <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
            <evaluator
class="ch.qos.logback.classic.boolex.OnMarkerEvaluator">
                <marker>SECURITY_SUCCESS</marker>
                <marker>SECURITY_WARNING</marker>
                <marker>SECURITY_FAILURE</marker>
            </evaluator>
            <onMismatch>DENY</onMismatch>
            <onMatch>ACCEPT</onMatch>
        </filter>
        <encoder
class="eu.eidas.node.logging.logback_integrity.HashPatternLayoutEncoder">
            <pattern>%d\{yyyy-MM-dd; HH:mm:ss.SSS\} \[%thread\] %-5level
%logger\{66\} %marker -%X\{sessionId\} -%X\{remoteHost\} -%msg%n</pattern>
        </encoder>
        <param name="Append" value="true" />
        <!-- Support multiple-JVM writing to the same log file -->
        <prudent>true</prudent>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>$\{LOG_HOME\}/eIDASNodeProxySecurity.%d\{yyyy-MM-
dd\}.log</fileNamePattern>
            <maxHistory>14</maxHistory>
        </rollingPolicy>
    </appender>

    <!--
        This define the SAML exchange Detailed log file appender - the
default Filter is inherited from root level
    -->
    <appender name="eIDASNodeProxySAMLExchange"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>$\{LOG_HOME\}/eIDASNodeProxySAMLExchange.log</file>

        <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
            <evaluator
class="ch.qos.logback.classic.boolex.OnMarkerEvaluator">
                <marker>SAML_EXCHANGE</marker>
            </evaluator>
            <onMismatch>DENY</onMismatch>
            <onMatch>ACCEPT</onMatch>
        </filter>
        <encoder
class="eu.eidas.node.logging.logback_integrity.HashPatternLayoutEncoder">
            <pattern>%d\{yyyy-MM-dd; HH:mm:ss.SSS\} \[%thread\] %-5level
%logger\{66\} %marker -%X\{sessionId\} -%X\{remoteHost\} -%msg%n</pattern>
        </encoder>
        <param name="Append" value="true" />
        <!-- Support multiple-JVM writing to the same log file -->
        <prudent>true</prudent>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>$\{LOG_HOME\}/eIDASNodeProxySAMLExchange.%d\{yyyy-MM-
dd\}.log</fileNamePattern>
            <maxHistory>14</maxHistory>
        </rollingPolicy>
    </appender>

    <!--
        This define the log file appender for the logging of the full
request and response messages
    -->
    <appender name="eIDASNodeProxyFullMessageExchange"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>$\{LOG_HOME\}/eIDASNodeProxyFullMsgExchange.log</file>

        <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
            <evaluator
class="ch.qos.logback.classic.boolex.OnMarkerEvaluator">
                <marker>FULL_MESSAGE_EXCHANGE</marker>
            </evaluator>
            <onMismatch>DENY</onMismatch>
            <onMatch>ACCEPT</onMatch>
        </filter>
        <encoder
class="eu.eidas.node.logging.integrity.HashPatternLayoutEncoder">
            <pattern>%d\{yyyy-MM-dd'T'HH:mm:ss.SSS'Z',GMT\} \[%thread\] %-
5level %logger\{66\} %marker -%X\{remoteHost\}:
%n%replace(%msg)\{'\[\r\n\]+',''\}%n</pattern>
        </encoder>
        <param name="Append" value="true" />
        <!-- Support multiple-JVM writing to the same log file -->
        <prudent>true</prudent>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>$\{LOG_HOME\}/eIDASNodeProxyFullMsgExchange.%d\{yyyy-MM-
dd,GMT\}.log</fileNamePattern>
            <maxHistory>14</maxHistory>
        </rollingPolicy>
    </appender>

    <!--
        This define the API fine grained level
    -->
    <logger name="org.opensaml">
        <level value="ERROR" />
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeProxyDetail"/>
    </logger>
    <logger name="com.opensymphony.xwork2">
        <level value="WARN"/>
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeProxyDetail"/>
    </logger>
    <logger name=" org.apache.struts2">
        <level value="WARN"/>
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeProxyDetail"/>
    </logger>
    <logger name="org.springframework">
        <level value="WARN" />
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeProxyDetail"/>
    </logger>
    <logger name="org.apache.xml.security">
        <level value="WARN" />
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeProxyDetail"/>
    </logger>

    <logger name="eu.eidas.communication.requests">
        <level value="info" />
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeProxyDetail"/>
    </logger>

    <logger name="eu.eidas.communication.responses">
        <level value="info" />
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="eIDASNodeProxyDetail"/>
    </logger>

    <!--
        The root level is set to debug for development purposes, for
production environment it could be set to INFO
    -->
    <root level="INFO">
        <appender-ref ref="STDOUT" />
        <appender-ref ref="eIDASNodeProxySystem" />
        <appender-ref ref="eIDASNodeProxySecurity" />
        <appender-ref ref="eIDASNodeProxyDetail" />
        <appender-ref ref="eIDASNodeProxySAMLExchange" />
        <appender-ref ref="eIDASNodeProxyFullMessageExchange" />
    </root>
</configuration>
```

3.3.2 Organisation of logs

The root level of logging defines the detail of logged events, for testing and development purposes, this level should be set to DEBUG. In the production environment, it should be INFO.

Ten different log files are generated by the application, depending on the context of the event to log (please refer to the *eIDAS-Node Error and Event Logging* guide for more details):

- the Application System log (*eIDASNodeConnectorSystem and eIDASNodeProxySystem*);
- the Application Security log (*eIDASNodeConnectorSecurity and eIDASNodeProxySecurity*);

- the Message Exchange log (*eIDASNodeConnectorSAMLExchange and eIDASNodeProxySAMLExchange*);
- the Debug Message Exchange log (*eIDASNodeConnectorFullMsgExchange and eIDASNodeProxyFullMsgExchange*) and
- the Application Detailed log (*eIDASNodeConnectorDetail and eIDASNodeProxyDetail*).

For further information on logging please refer to the *eIDAS-Node Error and Event Logging* and the *eIDAS-Node Security Considerations* guides.

4 Configuring the software

This section describes the configuration settings. Keep in mind that in production you need to enforce the configuration described in section 7.5 ─ *eIDAS-Node compliance*. Before proceeding with these steps your server must be configured, as described in section 3 — *Preparing the installation*.

Note: For information on implementing the eIDAS-Node Protocol Engine, please refer to the *eID eIDAS-Node and SAML* document.

4.1 JVM system properties

In this section, are described the JVM system properties, that are used by or defined for the eIDAS-Node.

4.1.1 HTTP forwarding through Proxy

The eIDAS Node has support for HTTP Proxy. In order for the proxy to be used, the following JVM properties can be used, depending on the needs:

- http.proxyHost
- http.proxyPort
- http.proxyUser
- http.proxyPassword
- http.nonProxyHosts

Note that the property http.nonProxyHosts could be needed to avoid using proxy for remote caches like ignite.

Example of proxy configuration without authentication:

```
set "JAVA_OPTS=%JAVA_OPTS% -Dhttp.proxyHost=192.168.137.129 -Dhttp.proxyPort=8888
-Dhttp.nonProxyHosts=127.0.0.1"
```

4.2 Configuring the project

To configure the project in the Basic Setup, follow the steps shown below.

4.2.1 Setup configuration directory

The $EIDAS__CONNECTOR_CONFIG_REPOSITORY and $EIDAS_PROXY_CONFIG_REPOSITORY environment variables are used to locate the eIDAS-Node-Connector and eIDAS-Node-Proxy's directory of configuration files respectively. They can be defined as an OS environment variable or by setting them to the runtime environment (by –D switch to JVM or on the AS admin console).

- $EIDAS_CONNECTOR_CONFIG_REPOSITORY – used in applicationContext.xml and points to the configuration directory of the Connector(e.g. file:/C:/PGM/projects/configEidasConnector/).
- $EIDAS_PROXY_CONFIG_REPOSITORY – used in applicationContext.xml and points to the configuration directory of the Proxy Service(e.g. file:/C:/PGM/projects/configEidasProxy/).

By default, EIDAS_CONNECTOR_CONFIG_REPOSITORY and EIDAS_PROXY_CONFIG_REPOSITORY OS environment or JVM command line arguments (-

D option) must be set in order to specify the location of configuration files. It is possible to change or hardcode these variables in *environmentalContext.xml*. Please refer to *environmentalContext.xml* for more details on how to do it.

4.2.2 Setting up your Keystore

Copy your eidasKeystore.p12 (the key store with your eIDAS-Node keys, alternatively you can use the example key store provided with the application) into a directory of your own choice, and make sure that:

- the property keyStorePath on file:$EIDAS_PROXY_CONFIG_REPOSITORY/SignModule_Service.xml reflects the relative location of your Proxy Service eidasKeyStore.p12.
- the property keyStorePath on file:$EIDAS_CONNECTOR_CONFIG_REPOSITORY/SignModule_Connector.xml reflects the relative location of your eIDAS-Node Connector eidasKeyStore.p12.

If the eIDAS-Node is configured to use encryption (essential in the production environment), also ensure that:

- the property keyStorePath on file:$EIDAS_CONNECTOR_CONFIG_REPOSITORY/EncryptModule_Connector.xml reflects the relative location of your eIDAS-Node Connector eidasKeyStore.p12.

For more information see the eID eIDAS-Node and SAML manual.

4.2.3 Configuring with Basic Setup

The Basic Setup allows you to use predefined configuration supplied with the software package, only for demo purposes. Copy the provided configuration files to the predefined EIDAS_CONNECTOR_CONFIG_REPOSITORY and EIDAS_PROXY_CONFIG_REPOSITORY and then edit the file eidas.xml to specify the following eIDAS-Node Connector and eIDAS-Node Proxy Service configuration properties.

```
connector.assertion.url=
http://insert.your.ip.here:portGoesHere/EidasNodeConnector/ColleagueRespon
se
```

To configure the Demo Tools in order to test this Basic Setup, please read *eIDAS-Node Demo Tools Installation and Configuration Guide.*

4.3 eIDAS-Node configuration files: eidas.xml

This section provides a detailed description of the eIDAS-Node configuration files and their properties.

The *eidas.xml* file contains the properties to configure:

When a property is not defined by the eidas.xml in the configuration, it will use the value defined in resources/default/eidas.xml which is bundled with the code inside the war.
The default values that are defined in the default configuration are usually a good option.
Only when needed should you add these parameters to the eidas.xml of your configuration in order to override the value of the property.
Not every configuration property is present in the default configuration file, for some properties, it is required that you define them.

4.3.1 General purpose parameters

The following Table 3, lists general-purpose parameters which include additional checks and security configurations.

Table 3: General purpose parameters

Key	Description
metadata.activate	Allows activation/deactivation of SAML metadata (activates/deactivates metadata publishing and requesting on both Connector and Proxy Service) (More information about the SAML metadata published by the Node in the *eIDAS-Node and SAML* manual*)*
node.metadata.not.signed.descriptors	List of URLs corresponding to entity descriptors whose signatures do not have to be checked. The format to use is http://descriptorurl1; https://descriptorurl2 etc. Default: no URL value is defined.
nonDistributedMetadata.retention	Retention period for simple metadata cache in seconds. Default value is 86400 seconds
logging.hash.digest.algorithm	Sets the digest algorithm to be used in the logging of messages functionality, this is used to determine the *bltHash* of the message to log. Default digest algorithm is SHA-512.
logging.hash.digest.provider	Sets the provider that should be used in the logging of messages functionality, this is used to determine the provider that should be used to hash the *bltHash*. Default security providers are used if not specified.
metadata.file.repository	Path to folder where static SAML metadata configuration is stored. If static SAML metadata for a given url exists in this folder, it will be used and it will override the retrieval of new remote metadata using HTTP otherwise if it does not exist or refers an unavailable location, only dynamically retrieved metadata may be used.
metadata.http.retrieval	Boolean value (true | false), which indicates whether the application will activate the use of the metadata from the HTTP URLs or use the static metadata.
metadata.node.country	Value of the Node Country to be published in the metadata Values must be compliant with the ISO 3166-1 alpha-2 format. This value is mandatory to be set.
eidas.protocol.version	Value(s) of all the eIDAS protocol version followed by the node, e.g. "1.4;1.3;1.2". Technical specifications' guidelines on eidas versioning have to be followed to fill in this parameter. When configured, the value(s) must not be empty and will be published in the node's metadata URLs. **Note:** Nodes will not be interoperable if there is no matching protocol version. For instance, a node

Key	Description
	supporting only version 1.1 will not be able to communicate with a node supporting only version 1.2.
eidas.application.identifier	Value(s) of eIDAS protocol's application identifier relative to the node's code and version number., e.g. "CEF:eIDAS-ref:2.1". A comma or semicolon separated list of values is supported. Technical specifications' guidelines on eidas versioning have to be followed to fill this parameter. When not empty, the value(s) will be published in the node's metadata URLs.
tls.enabled.protocols	TLSv1.2: SSL/TLS enabled protocols
tls.enabled.ciphers	The eIDAS supported cipher suites have been consolidated according to the eIDAS-Crypto requirements and the eIDAS Technical Specifications v1.2. Default cipher suites for java 11 are: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384, TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, TLS_DHE_RSA_WITH_AES_256_CBC_SHA256, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, TLS_EMPTY_RENEGOTIATION_INFO_SCSV

4.3.2 eIDAS-Node Connector configuration

The eIDAS-Node Connector configuration is composed of the following parts:

- Service Provider configuration;
- eIDAS-Node Connector dedicated information; and
- Configuration of the recognised Connector.

4.3.2.1 eIDAS-Node Connector dedicated information

To identify the eIDAS-Node Connector, the following information needs to be provided.

Table 4: eIDAS-Node Connector dedicated information

Key	Description
connector.assertion.url	URL of the Action to be called when returning from eIDAS-Node Proxy Service. (This used as AssertionConsumerServiceURL in the Request also)
saml.connector	SAML ProtocolEngine instance name, as configured in SamlEngine.xml, used by this eIDAS-Node Connector.

Key	Description
metadata.sector	Value of the type of SP to be published in Connector's metadata, possible values: public and private. If no value is provided, the SPType element will not be published in Connector's metadata and the request originating from such a Connector will need to provide the SP type itself. The default value is no value is provided.
connector.contact.support.email	Email address of the support contact (for metadata)
connector.contact.support.company	Company name of the support contact (for metadata)
connector.contact.support.givenname	Given name of the support contact (for metadata)
connector.contact.support.surname	Surname of the support contact (for metadata)
connector.contact.support.phone	Phone number of the support contact (for metadata)
connector.contact.technical.email	Email address of the technical contact (for metadata)
connector.contact.technical.company	Company of the technical contact (for metadata)
connector.contact.technical.givenname	Given name of the technical contact (for metadata)
connector.contact.technical.surname	Surname of the technical contact (for metadata)
connector.contact.technical.phone	Phone number of the technical contact (for metadata)
connector.metadata.url	The URL at which the metadata of eIDAS-Node Connector will be made available, e.g. *http://server:port/EidasNodeConnector/ConnectorMetadata* Will be used as Issuer in the requests that eIDAS-Node Connector sends, but does not set or validate the physical listener binding, therefore can be a custom value, like a reverse proxy external URL.
connector.organization.name	Name of the organization displayed in metadata
connector.organization.displayname	Localised display name of the organization for metadata
connector.organization.url	URL of the organisation for metadata containing information
specific.connector.response.receiver	URL for Specific Connector response receiver used when Specific Connector is built/deployed as WAR *https://<specific ProxyService.yourHostname>:*<specific ProxyService*.yourPort*>/SpecificProxyService/ ConnectorResponse
validate.prefix.country.code.identifiers	True : enable or disable validation of prefixing identifier like attribute values
metadata.location.whitelist	Semicolon separated urls of the ServiceProxies that the Connector is allowed to connect to obtain metadata/certificate to verify signature of SAML requests. This property is located in metadata/MetadataFetcher_Connector.properties

Key	Description
	Non-existent or empty whitelist will prevent the node from retrieving any metadata from any ServiceProxy.
metadata.location.whitelist.use	True: enable or disable the issuer metadata url whitelist logic. This property is located in metadata/MetadataFetcher_Connector.properties
connector.optional.nameid.formats	Semicolon separated list of NameID Formats that will be added to the list of mandatory NameID support which is: urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified urn:oasis:names:tc:SAML:2.0:nameid-format:transient urn:oasis:names:tc:SAML:2.0:nameid-format:persistent

If you are running tests across the network, you must change the *connector.assertion.url* to reflect the IP address of the machine running the eIDAS-Node Connector to:

```
http://connector.ip.address:connector.port.number/node.deployment.name/Col
leagueResponse
```

4.3.2.2 Configuring the recognised eIDAS-Node Proxy Services in the Connector

The eIDAS-Node Connector recognises the eIDAS-Node Proxy Services listed in eidas.xml. Increment the *service.number*, add their keys and respective values.

Table 5: List of Accepted Proxy-Services in eidas.xml (Connector)

Key x = {1.. service.number}	Description
service.number	**Number eIDAS-Node Proxy Services in the list.**
service{x}.id	Id of the eIDAS-Node Proxy Service. (ISO 3166-1-alpha-2 Country code)
service{x}.name	Name of the eIDAS-Node Proxy Service.
service{x}.metadata.url	URL where the eIDAS-Node Proxy Service publishes its metadata. Note: This metadata url also needs to be in the metadata whitelist of the Saml Engine. (eg: MetadataFetcher_Connector.properties will have the metadata url for every whitelisted service) The URL must be in the format: http://service.ip.address:service.port.number/service.deployment.name/ServiceMetadata
service{x}.skew.notbefore	Time skew in milliseconds to adjust notBefore SAML condition in Connector. The actual value is added to the received time condition, negative value is possible.
service{x}.skew.notonorafter	Time skew in milliseconds to adjust notOnOrAfter SAML condition in Connector. The actual value is added to the received time condition. A negative value is possible.

4.3.3 eIDAS-Node Proxy Service configuration

To activate an eIDAS-Node Proxy Service the following properties need to be provided:

Table 6: eIDAS-Node Proxy Service setup

Key	Description
saml.service	Name of the configuration instance for the Proxy Service's SAML Engine. Default value: Service
service.countrycode	The eIDAS-Node Proxy Service country ID in ISO 3166-1 alpha-2 format e.g. PT is the ISO 3166 code for Portugal. Used when the eIDAS-Node Proxy Service constructs the unique identifier attributes.
service.contact.support.email	Email address of the support contact (for metadata)
service.contact.support.company	Company of the support contact (for metadata)
service.contact.support.givenname	Given name of the support contact (for metadata)
service.contact.support.surname	Surname of the support contact (for metadata)
service.contact.support.phone	Phone number of the support contact (for metadata)
service.contact.technical.email	Email address of the technical contact (for metadata)
service.contact.technical.company	Company name of the technical contact (for metadata)
service.contact.technical.givenname	Given name of the technical contact (for metadata)
service.contact.technical.surname	Surname of the technical contact (for metadata)
service.contact.technical.phone	Phone number of the technical contact (for metadata)
service.organization.name	Name of the organisation displayed in the metadata
service.organization.displayname	Localised display name of the organisation for metadata
service.organization.url	URL of the organisation for Metadata containing information
service.metadata.url	The URL under which the metadata of Proxy Service will be made available, e.g. http://*server:port*/EidasNodeProxy/ServiceMetadata. Will be used as Issuer in the requests that eIDAS-Node Proxy Service sends, but does not set or validate the physical listener binding. Therefore it can be a custom value, like a reverse proxy external URL.
service.LoA	Semicolon separated Levels of Assurance that are provided by the IdP of the service. All supported Levels must be provided. Notified levels of Assurance values are: *http://eidas.europa.eu/LoA/highhttp://eidas.europa.eu/LoA/substantialhttp://eidas.europa.eu/LoA/low*Non-notified levels of

Key	Description
	Assurance must start with a different prefix than "http://eidas.europa.eu/LoA" This list will be checked against the Levels of Assurance of the Request. By default, the Node will assume all notified levels of Assurance (and only notified) are supported, if it is not the case, this property should be added in external configuration with the correct list of supported levels of Assurance.
service.optional.nameid.formats	Semicolon separated list of NameID Formats that will be added to the list of mandatory NameID support which is: urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified urn:oasis:names:tc:SAML:2.0:nameid-format:transient urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
ssos.serviceMetadataGeneratorIDP.redirect.location	The URI for the metadata <md:SingleSignOnService> location attribute of the SingleSignOnService related to Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect. e.g. http://EidasNode:8888/EidasNodeProxy/ColleagueRequest Does not come with physical binding check, so it can be set up for a reverse proxy external endpoint.
ssos.serviceMetadataGeneratorIDP.post.location	The URI for the metadata <md:SingleSignOnService> location attribute of the *SingleSignOnService* related to Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST. e.g. http://EidasNode:8888/EidasNodeProxy/ColleagueRequest Does not come with physical binding check, so it can be set up for a reverse proxy external endpoint.
metadata.location.whitelist	Semicolon separated urls of the Connectors that the ServiceProxy is allowed to connect to obtain metadata/certificate to verify signature of SAML responses. This property is located in metadata/MetadataFetcher_Service.properties Non-existent or empty whitelist will prevent the node from retrieving any metadata from any Connector.
metadata.location.whitelist.use	True: enable or disable the issuer metadata url whitelist logic. This property is located in metadata/MetadataFetcher_Service.properties
specific.proxyservice.request.receiver	URL for Specific ProxyService requests receiver only used when Specific ProxyService is built/deployed as WAR https://<specific ProxyService.yourHostname>:<specific ProxyService.yourPort>/SpecificProxyService/ProxyServiceRequest

Key	Description
insert.prefix.identifiers.country.code	True: enable or disable the prefixing of identifiers with country codes in identifier attribute
requester.id.flag	Requester Id flag, to allow eIDAS Service that require the RequesterID to be present in the eIDAS Request, to notify the eIDAS Connectors. When set to true enables publishing of requester Id entity category attribute in Proxy-service's metadata. This attribute will not be published when the flag is not set, or if the flag is set to false (This is the value in default/eidas.xml).
unsupported.attributes	Comma-separeted list of NamedURI that are not supported by the proxyservice side (IDPs). If not configured or left empty, the node will suppose that all the common attributes and additional attributes can be processed and are to be published in the proxy service metadata. Otherwise, the values of the list will be removed from the attributes that will be published to the metadata.

4.3.3.1 Additional Configuration — Skew Time

It is possible for clocks to be out of synchronisation between eIDAS-Node instances (Proxy Service / Connector). To prevent validation errors occurring in the Connector you can configure a skew time for each Proxy Service. The skew time gives the Connector an additional tolerance window for validating the timestamps in the SAML Responses that are sent by the Proxy Service. Please refer to Table 5: Adding eIDAS-Node Proxy Service to Connector for more information.

4.3.4 Additional configuration — Security policies

This section describes several configuration entries related to security policies. For more information about the security features please refer to the eIDAS-Node Security Considerations guide.

Table 7: Security policies

Key	Description
max.requests.ip	Maximum limit of requests per IP within the time frame of *max.time.ip* (-1 = unlimited)
max.requests.sp	Maximum limit of requests per SP within the time frame of *max.time.sp* (-1 = unlimited)
max.time.ip	Time frame for IP requests (seconds)
max.time.sp	Time frame for SP requests (seconds)
trusted.sp.domains	Allowed SPs to communicate with the eIDAS-Node Connector Possible values are : • none • all • a point comma (";") list of domains The default value is : "all". All domains are trusted by default.
validation.bypass	Activate or not the bypass for the validation of the domain and amount of requests. The bypass is activated by default. (*true* | false)

Table 8: Security HTTP header parameters

Key	Description
security.header.CSP.enabled	Enable/disable sending the Content Security Policy (CSP) header. CSP protects against the injection of foreign content. Default value: true
security.header.CSP. includeMozillaDirectives	In the CSP, this additional directive can be added for backward compatibility with old Mozilla browsers. Default value: true
security.header. XXssProtection.block	This header enables the cross-site-scripting (XSS) filter built into most recent web browsers. Default value: true
security.header. XContentTypeOptions.noSniff	The only defined value 'nosniff' prevents Internet Explorer and Google Chrome from 'MIME-sniffing' by inspecting the content of a response. Default value: true
security.header. XFrameOptions.sameOrigin	Prevents the application from being propagated in a frame or iframe, which in turns protects against key logging, clickjacking and similar attacks. Setting this option to **true** will prevent the eIDAS-Node from being framed in another application. If the SP needs to frame the eIDAS-Node, the option has to be set to 'false' Default value: true
security.header. HSTS.includeSubDomains	HTTP Strict-Transport-Security (HSTS) instructs browsers to prefer secure connections to the server (HTTP over SSL/TLS) over insecure ones. Default value: true
security.header.CSP.report.uri	Prefix for **report_uri** header populated by the node in order to avoid retrieving the host/name from the request itself, that otherwise would have exposed the eidas flow to threats known as 'Host header poisoning'. The eidas node populated this header with a value similar to *http://eidasnode:8888/EidasNodeConnector/cspReportHandler* or *http://eidasnode:8888/EidasNodeProxy/cspReportHandler*

Table 9: Check on certificate security parameter

Key	Description
check.citizenCertificate. serviceCertificate	Flag to activate the check between the country code stored in the Proxy Service response signing certificate and the requested citizen country code. This flag is not used if the check is based on the country code published in the Proxy Service's Metadata Default value for this parameter is true.

4.3.4.1 SAML Binding method

Table 10: SAML binding parameters

Key	Description
allow.redirect.binding	Whether to allow the HTTP Redirect binding. Possible values are true/false. Default value is true

Key	**Description**
validate.binding	Whether to validate the actual binding (POST or GET/Redirect) against ProtocolBinding attribute value of the SAML request. Possible values are true/false. Default value is true.

Generally, eIDAS-Nodes operate using SAML POST Binding. The parameter *allow.redirect.binding* (set to true) instructs the eIDAS-Node to accept HTTP Redirect Binding SAML requests, normally coming as HTTP GET requests. When HTTP Redirect Binding is used the following items should be considered:

- Most browsers have a low limit for the size of GET requests.
- Most servers have a low limit for the size of an HTTP header (e.g. in Apache Tomcat v9.0.85 this limit is about 8k; in order to increase this limit, the connector element in server.xml should contain a *maxHttpHeaderSize* element with the new limit);
- When this binding is activated, an HTTP redirect binding request received by Connector will be forwarded also as a redirect to Proxy Service and further (to IdP);
- The response is always sent back through an HTTP POST operation.

When *allow.redirect.binding* is set to false, an error page will be displayed if the node receives a SAML or Light request with GET binding.

4.4 eIDAS-Node Configuration Files: SAMLEngine.xml

The SAMLEngine*.xml* file contains the structure to configure:

When a property is not defined by any of the configuration files of the samlEngine.xml,
it will use the value defined in src/main/resources/default/defaultSamlEngineProperties.xml of the EIDAS-SAMLEngine module which is bundled with the code inside the war.
The default values that are defined in the default configuration are usually a good option.

Depending on how you wish to configure the engine, should you override these parameters in their respective file?
Not every configuration property is present in the default configuration file, for some properties it is required that you define them.

Please refer to the eIDAS-Node and SAML Guide for more in depth information on SAMLEngine.xml.

4.4.1 SamlEngineConf

Defines the location of the SamlEngine properties configuration file. (eg. SamlEngine_Connector.xml).

4.4.2 SignatureConf

Defines what is used for signing and verifying signatures.

- The implementation class that defines the properties
- The location of the signature configuration file. (eg. SignModule_Connector.xml)

4.4.2.1 Signature Table 11: Signature algorithm

Key	**Description**
signature.algorithm	This is a SAMLEngine configuration entry, it defines the signing algorithm (SHA2 based) used by the default

Key	Description
	signer for outgoing requests and metadata. Possible values are: http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-MGF1 http://www.w3.org/2007/05/xmldsig-more#sha384-rsa-MGF1 http://www.w3.org/2007/05/xmldsig-more#sha512-rsa-MGF1 http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256 http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384 http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512 The default configuration value is:http://www.w3.org/2007/05/xmldsig-more#ecdsa-sha512 Note that only four values should be used to allow backward compatibility (All the ecdsa algorithms and the sha256-rsa-MGF1).
signature.algorithm.whitelist	This is a SAMLEngine configuration it consists of a list of allowed signature algorithms (in incoming requests). It contains OpenSAML's supported signing algorithms, separated by **;**. Currently the elements of the list may be picked from the following (which is the default whitelist) : http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-MGF1 http://www.w3.org/2007/05/xmldsig-more#sha384-rsa-MGF1 http://www.w3.org/2007/05/xmldsig-more#sha512-rsa-MGF1 http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256 http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384 http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512 But note that only four values of the list are also compatible with previous versions of the specifications. Also refer to the *eID eIDAS-Node and SAML* manual.
digest.method.algorithm	This allows you to configure the digest method algorithm that will be used to sign. If not specified the default digest method algorithm will be :http://www.w3.org/2001/04/xmlenc#sha512
digest.method.algorithm.whitelist	This allows you to configure the whitelist of digest method algorithms that will be used to verify digest method algorithms used If not specified, the following list of digest method algorithms will be used: • http://www.w3.org/2001/04/xmlenc#sha256 • http://www.w3.org/2001/04/xmldsig-more#sha384

Key	Description
	• http://www.w3.org/2001/04/xmlenc#sha512
enable.certificate.revocation.checking	When set to true, this enables revocation checking for metadata signature certificates. More information in *Appendix A: Certificate Revocation Validation* of document *eIDAS-Node and SAML v2.8* Default value is true
enable.certificate.revocation.soft.fail	When set to true, this enables the soft fail functionality for revocation checking of metadata signature certificates. More information in *Appendix A: Certificate Revocation Validation* of document *eIDAS-Node and SAML v2.8* Default value is false

4.4.3 EncryptionConf

Defines what is used for encrypting and/or decrypting Responses.

- The implementation class that defines the properties (eg. EncryptionOnlySW or EncryptionDecryptionSW)
- The location of the encryption configuration file. (eg. EncryptionModule_Connector.xml)

4.4.3.1 Encryption

Table 12: Configuring encryption algorithm

Key	Description
data.encryption.algorithm	This is a SAMLEngine configuration entry. Contains the encryption algorithm to be used by Proxy Service and Connector. Possible value must be : http://www.w3.org/2009/xmlenc11#aes128-gcm http://www.w3.org/2009/xmlenc11#aes256-gcm http://www.w3.org/2009/xmlenc11#aes192-gcm Default value: http://www.w3.org/2009/xmlenc11#aes256-gcm
encryption.algorithm.whitelist	This is a SAMLEngine configuration. Contains the encryption algorithms allowed in the responses received by eIDAS-Node components. As per specification, only the following values (which are the default whitelist) are allowed: http://www.w3.org/2009/xmlenc11#aes256-gcmhttp://www.w3.org/2009/xmlenc11#aes192-gcm http://www.w3.org/2009/xmlenc11#aes128-gcm

Note that additional configurations regarding the encryption can be done within the SamlEngine configuration (SamlEngine.xml) or in configuration files described/specify inside an instance declaration. For more information, please refer to the *eID eIDAS-Node and SAML* manual.

4.4.4 ProtocolProcessorConf

Defines what is used for the protocol processor

- The location of the addtional attributes configuration file. (eg. saml-engine-additional-attributes.xml)

- The implementation for the metadataFetcherClass

4.4.4.1 Attribute registry

Attribute registry holds and supplies information of types, value format and namespace for creating and validating requests and responses. The registry basically contains Attribute Definition objects built from custom XML files and hard coded lists of supported core attributes in *LegalPersonSpec*, *NaturalPersonSpec*, *RepresentativeLegalPersonSpec*, and *RepresentativeNaturalPersonSpec* collected together in *EidasSpec* class, found in the SAMLEngine module.

Each Protocol Engine has its own configuration files, specified by SamlEngine.xml files.

The following is an example code to introduce a new attribute to the XML configuration:

```
    <entry
key="19.NameUri">http://eidas.europa.eu/attributes/natural/NewSomething</e
ntry>
    <entry key="19.FriendlyName">NEW_SOMETHING</entry>
    <entry key="19.PersonType">NaturalPerson</entry>
    <entry key="19.Required">false</entry>
    <entry
key="19.XmlType.NamespaceUri">http://eidas.europa.eu/attributes/naturalper
son</entry>
    <entry key="19.XmlType.LocalPart">NewSomethingType</entry>
    <entry key="19.XmlType.NamespacePrefix">eidas-natural</entry>
```

For the *key* prefix number, take the last one and increment it. For eIDAS protocol the person type (natural or legal) must be specified and aligned with namespace.

4.4.4.1.1 Attribute registry validation and metadata support

Besides the Attribute Registry XML files there is a hard coded list of supported core attributes in *LegalPersonSpec*, *NaturalPersonSpec*, *RepresentativeLegalPersonSpec*, and *RepresentativeNaturalPersonSpec* collected together in *EidasSpec* class, can be found in the SAMLEngine module. This is necessary to get a reference of attribute definitions to perform business rule-based validations on requests and replies.

Supported attributes are published in the Metadata of the eIDAS-Node Proxy Service.

4.4.4.2 metadataFetcherClass

The implementation for the metadataFetcherClass

Note: Currently properties for the metadata Fetcher are being loaded based of the instance name in SamlEngine.xml,
for example an <instance name="Service"> will imply a file ./metadata/MetadataFetcher_Service.properties.

4.4.5 ClockConf

Defines to provide an implementation of timekeeping for the SAML Engine.

4.5 Additional Configuration — SignModule_Service.xml and SignModule_Connector.xml

It may be necessary to change the *keyStorePath* to reflect the location of your *eidasKeyStore.p12* and *eidasKeyStore_METADATA.p12* files, please see the *eIDAS-Node and SAML* manual for more information.

4.6 Additional Configuration — Anti-replay Cache and Correlation Map Configuration

To prevent a replay of SAML requests an anti-replay cache is implemented at the eIDAS-Node Connector and eIDAS-Node Proxy Service level. We provide two different implementations for these caches, which can be configured. By default, the eIDAS-Node is set up to use a distributed cache with expiration.

This implementation is provided for correlating request and reply pairs both for *AuthenticationRequests* and *LightRequests*.

By default, there is one distributed caches instance used by the Node for both correlation and anti-replay map purposes. This is loaded through the build depending on which EIDAS-JCache-XXXX-Node module is added as a dependency.

If EIDAS-JCache-Ignite-Node module/dependency is used for distributed caches at the node, *jCacheImplNodeBeans.xml* file contained in that module will provide the caches implementations based on Ignite.

The beans at the jCacheImplNodeBeans.xml file will provide the implementations of the caches used by the node.

```
<!-- production environment ignite initializer bean - injected into map
providers -->
<bean id="eidasIgniteInstanceInitializerNode"
class="eu.eidas.auth.cache.IgniteInstanceInitializerNode" init-
method="initializeInstance" lazy-init="true">
    <property name="configFileName"
value="#{eidasConfigRepository}/ignite/igniteNode.xml"/>
</bean>
```

Figure 2: Node's Ignite instance name

The Ignite instance is provided by the *eidasIgniteInstanceInitializerNode* bean.

```
<!-- production environment ignite initializer bean - injected into cache
providers -->
<bean id="eidasIgniteInstanceInitializerNode"
class="eu.eidas.auth.cache.IgniteInstanceInitializerNode" init-
method="initializeInstance" lazy-init="true">
    <property name="configFileName"
value="#{eidasConfigRepository}/ignite/igniteNode.xml"/>
</bean>
```

Figure 3: Node's Ignite instance provider bean

Note that #{eidasConfigRepository} value will originate from src/resources/environmentContext.xml.

This bean is injected into beans that have defined as class *ConcurrentMapServiceDistributedImpl* or *DistributedMetadataCaching*. If the distributed environment requires setup of multiple instances, the configuration can be done simply adding more of the above beans to *applicationContext*.

```
<bean id="springServiceCMapAntiReplayProviderImpl"
class="eu.eidas.auth.cache.ConcurrentCacheServiceIgniteNodeImpl" lazy-
init="true">
    <property name="igniteInstanceInitializer"
ref="eidasIgniteInstanceInitializerNode"/>
    <property name="cacheName" value="antiReplayCacheService"/>
</bean>
<bean id="springConnectorCMapAntiReplayProviderImpl"
class="eu.eidas.auth.cache.ConcurrentCacheServiceIgniteNodeImpl" lazy-
init="true">
    <property name="igniteInstanceInitializer"
ref="eidasIgniteInstanceInitializerNode"/>
    <property name="cacheName" value="antiReplayCacheConnector"/>
</bean>
```

Figure 4: Anti-replay cache configuration — Ignite —jCacheImplNodeBeans.xml (EIDAS-JCache-Ignite-Node module)

For correlation maps, there are two *AuthRequest* and one *LightRequest* type maps in *ApplicationContext*, one for the Proxy Service, two for the Connector one of which is for the Specific Connector.

```
<!-- Correlation maps provided by Ignite for distributed environment, use
these in productions! -->
<bean id="springConnectorCMapCorProviderImpl"
class="eu.eidas.auth.cache.ConcurrentCacheServiceIgniteNodeImpl" lazy-
init="true">
    <property name="igniteInstanceInitializer"
ref="eidasIgniteInstanceInitializerNode"/>
    <property name="cacheName"
value="connectorRequestCorrelationCacheService"/>
</bean>
<bean id="springServiceCMapCorProviderImpl"
class="eu.eidas.auth.cache.ConcurrentCacheServiceIgniteNodeImpl" lazy-
init="true">
    <property name="igniteInstanceInitializer"
ref="eidasIgniteInstanceInitializerNode"/>
    <property name="cacheName"
value="proxyServiceRequestCorrelationCacheService"/>
</bean>
<bean id="springConnectorCMapspecificLightCorProviderImpl"
class="eu.eidas.auth.cache.ConcurrentCacheServiceIgniteNodeImpl" lazy-
init="true">
    <property name="igniteInstanceInitializer"
ref="eidasIgniteInstanceInitializerNode"/>
    <property name="cacheName"
value="specificConnectorLtRequestCorrelationCacheService"/>
</bean>
```

Figure 5: Correlation caches configuration — Ignite — jCacheImplNodeBeans.xml (EIDAS-JCache-Ignite-Node module)

For more information about the Ignite product, please refer to Appendix C.

4.7 Error Codes and Error Messages

Error messages are defined in eidasErrors.properties and EidasErrorKey enum of the EIDAS-Commons module.

```
AUTHENTICATION_FAILED_ERROR("authenticationFailed"),
```

```
authenticationFailed.code=003002
authenticationFailed.message=authentication.failed
```

These keys are mapped to language files using the "message" key and {0} is interpolated with the "code".

```
authentication.failed={0} - Authentication Failed.
```

Will result in the string "003002 - Authentication Failed."

Note: The full list of eIDAS-Node error codes and related error messages is shown in the eIDAS-Node Error Codes document

4.7.1 Different language files have different audiences in mind.

sysadmin.properties	Used to create the error message presented in the user interface.
error.properties	Used to create the error message contained in a SAML Response.
eidastranslation.properties	Used to translate the error **code** contained in a SAML Response back to the SP.

4.7.2 Multi language support

Using a suffix to the filename, support can be added per language. For the UI this is infered from the "Accept-Language" in the citizen's browser.

eg. sysadmin_pt.properties (i.e. Portuguese language) **authentication.failed={0} - A autenticação falhou.** Will result in the string "003002 - A autenticação falhou."	eg. sysadmin_nl.properties (i.e. Dutch language) **authentication.failed={0} - De authenticatie is mislukt.** Will result in the string "003002 - De authenticatie is mislukt."

eidastranslation.properties will use the Locale from JVM, this is also the fallback behaviour for the other language files.

4.8 Configuring non-node components

4.8.1 Specific properties

For the Basic Setup, you might need to reconfigure MS-Specific module Configuration for that application as detailed in the *eIDAS-Node Demo Tool Installation and Configuration Guide*.

4.8.2 Demo Service Provider

For the Basic Setup, you might need to reconfigure Demo Service Provider. Configuration for that application is detailed in the *eIDAS-Node Demo Tool Installation and Configuration Guide*.

4.8.3 Demo Identity Provider

In order to proceed with Basic Setup, you might need to modify the configuration of Demo Identity Provider. The procedure and settings are detailed in the *eIDAS-Node Demo Tool Installation and Configuration Guide*.

5 Building and deploying the software

This section describes the steps to build and then to deploy the software on the supported servers. There are two main types of eIDAS-Node: Connector and Proxy Service.

The project build files are in **Maven3** format, so you need to install Maven. Download instructions are provided at http://maven.apache.org/run-maven/index.html). Recommended versions of Maven are 3.8.0 and above. Lower versions can result in exceptions.

There are two ways to build the binaries from sources:

1. **Parent build**

When building the project using the parent build, it is recommended to build from the root folder of the project using the maven *--file* (or -f) option to indicate the location of the *pom.xml* in the EIDAS-Parent module. This is necessary in order to allow the jaxb plugin, which generates the LightResponse and LightRequest java classes from xsd files, to work properly.

The *pom.xml* file in the EIDAS-Parent module is a common reference for all dependent module/external Maven artifact versions, and able to build all binaries related to EidasNode and/or Demo Tools.

There are various profiles to help tailoring the build to one's particular needs: these can be split in two main categories:

- First: profiles related to application server specifics, for instance profiles named weblogic.
- Second: two profiles related to the scope of modules to be built, specifically NodeOnly (this is active by default,) and DemoToolsOnly.

For instance issuing Maven "install" command with the appropriate activation profile (e.g. for *WebLogic*: -P *weblogic*,*NodeOnly*,*DemoToolsOnly*) will result in a full build.

- Third: several profiles are present in that switch between Jcache implementations for Node and for Specific Communication Caches. Those are profiles :
  -PnodeJcacheIgnite : Ignite JCache implementation
  -PnodeJcacheDev : Guava JCache implementation/adaptation (Only for non distributed case)
  -PspecificCommunicationJcacheIgnite : Ignite JCache implementation
  -PspecificCommunicationJcacheDev: Local Caches implementation (possible only for Monolithic deployment)

Note that among these, the build has to be executed with one of the nodeXXX profiles and one for the specificCommunicationXXX profiles so that the node works properly.

2. Module-based build

it is possible to build the artifacts one-by-one, which can be helpful if there is a need to build just one module. In this case please remember the dependencies between them. There is a certain order that needs to be followed.

Note : Paths with spaces should not be used in order to avoid problem during the execution of the JAXB plugin which generates java classes from xsd files.

The next sections detail the above two methods for supported application servers.

5.1 Tomcat v9.0.85 server deployment

You must compile, install and deploy the projects, either by compiling the parent project (Table 13: Parent project build for Tomcat Server) or by compiling each module separately in the order shown below (Table 14: Module-based build for Tomcat Server).

At a command prompt, navigate to the folder as shown below and enter the corresponding command line.

In order to deploy the project, after the build is complete, copy the artifacts (EIDAS-Node-Connector/target/EidasNodeConnector.war and EIDAS-Node-Proxy/target/EidasNodeProxy.war) to the deploy folder of the Server.

Deploy folder for tomcat is: $TOMCAT_HOME/webapps/

Table 13: Parent project build for Tomcat Server

Step	Folder	Command line
1	Project root folder	mvn clean install --file EIDAS-Parent/pom.xml -PNodeOnly[,DemoToolsOnly] -P nodeJcacheIgnite | nodeJcacheDev -P specificCommunicationJcacheIgnite | specificCommunicationJcacheDev [-DspecificJar] Note one of the nodeJcacheX profiles as well as one of specificCommunicationJcacheX profiles needs to be chosen.

Table 14: Module-based build for Tomcat Server

Step	Folder	Command line
1	Project root folder	mvn clean install --file EIDAS-Parent/pom.xml
2	EIDAS-Light-Commons	mvn clean install
3	EIDAS-Commons	mvn clean install
4	EIDAS-JCache-Dev	mvn clean install
5	EIDAS-JCache-Dev-Node	mvn clean install
6	EIDAS-JCache-Dev-Specific-Communication	mvn clean install
7	EIDAS-JCache-Ignite	mvn clean install
8	EIDAS-JCache-Ignite-Node	mvn clean install
9	EIDAS-JCache-Ignite-Specific-Communication	mvn clean install
10	EIDAS-SpecificCommunicationDefinition	mvn clean install -P nodeJcacheIgnite | nodeJcacheDev -P specificCommunicationJcacheIgnite | specificCommunicationJcacheDev [-DspecificJar] Note one of the nodeJcacheX profiles as well as one of specificCommunicationJcacheX profiles needs to be chosen.
11	EIDAS-Encryption	mvn clean install
12	EIDAS-Metadata	mvn clean install
13	EIDAS-SAMLEngine	mvn clean install

Step	Folder	Command line
14	EIDAS-Updater	mvn clean install
15	EIDAS-Node-Connector	mvn clean package
16	EIDAS-Node-Proxy	mvn clean package

5.2 WildFly 23.0.2 Final Server (Servlet-Only Distribution) deployment

You must compile, install and deploy the projects, either by compiling the parent project (Table 15: Parent project build for WildFly 23.0.2 Final Server (Servlet-Only Distribution)) or by compiling each module separately in the order shown below (Table 16: Module-based build for WildFly 23.0.2 Final Server (Servlet-Only Distribution)).

At a command prompt, navigate to the folder as shown below and enter the corresponding command line.

In order to deploy the project, after the build is complete, copy the artifacts (EIDAS-Node-Connector/target/EidasNodeConnector.war and EIDAS-Node-Proxy/target/EidasNodeProxy.war) to the deploy folder of the Server.

Deploy folder for wildfly is: $WILDFLY_HOME/standalone/deployments/

Table 15: Parent project build for WildFly 23.0.2 Final Server (Servlet-Only Distribution)

Step	Folder	Command line
1	Project root folder	mvn clean install --file EIDAS-Parent/pom.xml -PWildfly,NodeOnly[,DemoToolsOnly] -P nodeJcacheIgnite | nodeJcacheDev -P specificCommunicationJcacheIgnite | specificCommunicationJcacheDev [-DspecificJar] Note one of the nodeJcacheX profiles as well as one of specificCommunicationJcacheX profiles needs to be chosen.

Table 16: Module-based build for WildFly 23.0.2 Final Server (Servlet-Only Distribution)

Step	Folder	Command line
1	Project root folder	mvn clean install --file EIDAS-Parent/pom.xml
2	EIDAS-Light-Commons	mvn clean install
3	EIDAS-Commons	mvn clean install
4	EIDAS-JCache-Dev	mvn clean install
5	EIDAS-JCache-Dev-Node	mvn clean install
6	EIDAS-JCache-Dev-Specific-Communication	mvn clean install
7	EIDAS-JCache-Ignite	mvn clean install
8	EIDAS-JCache-Ignite-Node	mvn clean install

Step	Folder	Command line
9	EIDAS-JCache-Ignite-Specific-Communication	mvn clean install
10	EIDAS-SpecificCommunicationDefinition	mvn clean install -P nodeJcacheIgnite | nodeJcacheDev -P specificCommunicationJcacheIgnite | specificCommunicationJcacheDev [-DspecificJar] Note one of the nodeJcacheX profiles as well as one of specificCommunicationJcacheX profiles needs to be chosen.
11	EIDAS-Encryption	mvn clean install
12	EIDAS-Metadata	mvn clean install
13	EIDAS-SAMLEngine	mvn clean install
14	EIDAS-Updater	mvn clean install
15	EIDAS-Node-Connector	mvn clean package –P wildlfy
16	EIDAS-Node-Proxy	mvn clean package –P wildlfy

5.3 WebLogic Server 14.1.1.0.0 deployment

You must compile, install and deploy the projects, either by compiling the parent project (Table 17: Parent project build for WebLogic Server) or by compiling each module separately in the order shown below (Table 18: Module-based build for WebLogic Server).

At a command prompt, navigate to the folder as shown below and enter the corresponding command line.

In order to deploy the project, after the build is complete, copy the artifact (EIDAS-Node-Connector/target/EidasNodeConnector.war and EIDAS-Node-Proxy/target/EidasNodeProxy.war) to the deploy folder of the Server.

Deploy folder for Weblogic is: $MW_HOME/user_projects/domains/base_domain/autodeploy/

Table 17: Parent project build for WebLogic Server deployment

Step	Folder	Command line
1	Project root folder	mvn clean install --file EIDAS-Parent/pom.xml -P weblogic,NodeOnly[,DemoToolsOnly] -P nodeJcacheIgnite | nodeJcacheDev -P specificCommunicationJcacheIgnite | specificCommunicationJcacheDev [-DspecificJar] Note one of the nodeJcacheX profiles as well as one of specificCommunicationJcacheX profiles needs to be chosen.

Table 18: Module-based build for WebLogic Server deployment

Step	Folder	Command line
1	Project root folder	mvn clean install --file EIDAS-Parent/pom.xml
2	EIDAS-Light-Commons	mvn clean install
3	EIDAS-Commons	mvn clean install
4	EIDAS-JCache-Dev	mvn clean install
5	EIDAS-JCache-Dev-Node	mvn clean install
6	EIDAS-JCache-Dev-Specific-Communication	mvn clean install
7	EIDAS-JCache-Ignite	mvn clean install
8	EIDAS-JCache-Ignite-Node	mvn clean install
9	EIDAS-JCache-Ignite-Specific-Communication	mvn clean install
10	EIDAS-SpecificCommunicationDefinition	mvn clean install -P nodeJcacheIgnite | nodeJcacheDev -P specificCommunicationJcacheIgnite | specificCommunicationJcacheDev [-DspecificJar] Note one of the nodeJcacheX profiles as well as one of specificCommunicationJcacheX profiles needs to be chosen.
11	EIDAS-Encryption	mvn clean install
12	EIDAS-Metadata	mvn clean install
13	EIDAS-SAMLEngine	mvn clean install
14	EIDAS-Updater	mvn clean install
15	EIDAS-Node-Connector	mvn clean package –P weblogic
16	EIDAS-Node-Proxy	mvn clean package –P weblogic

5.4 WebSphere Liberty Server v21.0.0.5 (WebProfile 8) deployment

You must compile, install and deploy the projects, either by compiling the parent project (Table 19: Parent project build for WebSphere Server) or by compiling each module separately in the order shown below (Table 20: Module-based build for WebSphere Server).

At a command prompt, navigate to the folder as shown below and enter the corresponding command line.

In order to deploy the project, after the build is complete, copy the artifacts (EIDAS-Node-Connector/target/EidasNodeConnector.war and EIDAS-Node-Proxy/target/EidasNodeProxy.war) to the deploy folder of the Server.

Deploy folder for Websphere Liberty (WebProfile) is: ${WEBSPHERE_SERVER_HOME}/usr/servers/${SERVER}/dropins/

Table 19: Parent project build for WebSphere Server deployment

Step	Folder	Command line
1	Project root folder	mvn clean install --file EIDAS-Parent/pom.xml -PNodeOnly[,DemoToolsOnly] -P nodeJcacheIgnite | nodeJcacheDev -P specificCommunicationJcacheIgnite | specificCommunicationJcacheDev [-DspecificJar] Note one of the nodeJcacheX profiles as well as one of specificCommunicationJcacheX profiles needs to be chosen.

Table 20: Module-based build for WebSphere Server deployment

Step	Folder	Command line
1	Project root folder	mvn clean install --file EIDAS-Parent/pom.xml
2	EIDAS-Light-Commons	mvn clean install
3	EIDAS-Commons	mvn clean install
4	EIDAS-JCache-Dev	mvn clean install
5	EIDAS-JCache-Dev-Node	mvn clean install
6	EIDAS-JCache-Dev-Specific-Communication	mvn clean install
7	EIDAS-JCache-Ignite	mvn clean install
8	EIDAS-JCache-Ignite-Node	mvn clean install
9	EIDAS-JCache-Ignite-Specific-Communication	mvn clean install
10	EIDAS-SpecificCommunicationDefinition	mvn clean install -P nodeJcacheIgnite | nodeJcacheDev -P specificCommunicationJcacheIgnite | specificCommunicationJcacheDev [-DspecificJar] Note one of the nodeJcacheX profiles as well as one of specificCommunicationJcacheX profiles needs to be chosen.
11	EIDAS-Encryption	mvn clean install
12	EIDAS-Metadata	mvn clean install
13	EIDAS-SAMLEngine	mvn clean install
14	EIDAS-Updater	mvn clean install
15	EIDAS-Node-Connector	mvn clean package
16	EIDAS-Node-Proxy	mvn clean package

5.5 Monolithic Deployment

Besides the 'Basic Deployment' described in this document, a 'Monolithic Deployment' is possible. In this case, the EidasNodeConnector.war will include the SpecificConnector module and the EidasNodeProxy.war will include the SpecificProxyService module as JAR.

In this case add –D *specificJar* to the build commands for the following modules:

- EIDAS-SpecificCommunicationDefinition
- EIDAS-Node-Connector
- EIDAS-Node-Proxy

This also applies to Demo Tools modules, so please check the *Monolithic Deployment* section in the *Demo Tools Installation and Configuration Guide* for more details.

Lastly, if monolithic deployment will be performed, the operator will need to follow and take into consideration the document above (*Demo Tools Installation and Configuration Guide*), notably the configuration parameters such relaystate.randomize.null, etc.

In this case, the, EIDAS-JCache-Dev will be included by default as the Jcache implementation of the Communication Caches between Node and MS Specific parts.

It is possible to change the Jcache implementation to Jcache-Ignite, by activating profile *specificCommunicationJcacheIgnite*.

6 Verifying the installation

This section shows the final structure of your application server relevant directories, so that you can confirm that you have made the proper configurations. The structure of the application's 'war' files is also shown so you can verify that your applications were built successfully.

6.1 Tomcat 9.0.85

```
$TOMCAT_HOME/webapps/
EidasNodeConnector.war
EidasNodeProxy.war
(server specific directories were not included)
```

6.2 WildFly 23.0.2 Final Server (Servlet-Only Distribution)

```
$WILFDLY_HOME/standalone/Deployments/
EidasNodeConnector.war
EidasNodeProxy.war
```

Code Block 2 bash

6.3 WebLogic

```
$WLS_HOME/domain/autodeploy/
EidasNodeConnector.war
EidasNodeProxy.war
(server specific directories were not included)
```

6.4 WebSphere Liberty Server

```
$WEBSPHERE_SERVER_HOME/usr/servers/$SERVER/dropins/
EidasNodeConnector.war
EidasNodeProxy.war
(server specific directories were not included)
```

6.5 Configuration files

The below configuration and keystore files are needed for the installation of the eIDAS-Node-Connector and eIDAS-Node-ProxyService.
The layout itself can be different, depending on the environment variables, so this is just an example of Basic Setup:

Under EIDAS_CONNECTOR_CONFIG_REPOSITORY folder:

ignite/KeyStore/server.p12ignite/KeyStore/trust.p12ignite/igniteNode.xml (needed only in default build or if profile nodeJcacheIgnite is activated)

ignite/igniteSpecificCommunication.xml (needed only in default build or if profile specificCommunicationJcacheIgnite is activated) keystore/eidasKeyStore.p12 keystore/eidasKeyStore_METADATA.p12 metadata/MetadataFetcher_Connector.properties eidas.xml SamlEngine.xml EncryptModule_Connector.xml SamlEngine_Connector.xml SignModule_Connector.xml saml-engine-additional-attributes.xml
Under EIDAS_PROXY_CONFIG_REPOSITORY folder: ignite/KeyStore/server.p12ignite/KeyStore/trust.p12ignite/igniteNode.xml (needed only in default build or if profile nodeJcacheIgnite is activated) ignite/igniteSpecificCommunication.xml (needed only in default build or if profile specificCommunicationJcacheIgnite is activated) keystore/eidasKeyStore.p12 keystore/eidasKeyStore_METADATA.p12 metadata/MetadataFetcher_Service.properties eidas.xml SamlEngine.xml encryptionConf.xml EncryptModule_Service.xml SamlEngine_Service.xml SignModule_Service.xml saml-engine-additional-attributes.xml

7 Advanced configuration for production environments

This section provides detailed descriptions of the configurations to enable you to change specific aspects as required.

7.1 Clustering environment

This section describes the technologies and configurations used by the eIDAS-Node in cluster mode. The choice of technologies is proposed for testing purposes.

7.1.1 Load balancer

The configuration adopted is the following:

- One load balancer composed of two Tomcat 9 (version 9.0.85) servers including the eIDAS-Node;
- One Apache Http server to isolate SP/IDP request.

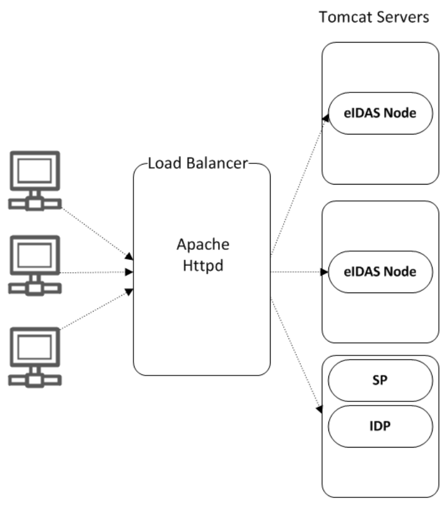


Figure 6: Clustering environment — Load balancer

The solution is to add one server in-front of all Tomcat clusters to accept all the requests and distribute to the cluster. So this server acts as a **load balancer**.

There are several servers available with load-balancing capability. Here we are going to use **Apache httpd** web server as a load balancer. With **mod_jk** module.

If one of the Tomcat instances fails then the load balancer dynamically reacts by ceasing to forward requests to that failed Tomcat instances. Other Tomcat instances continue as normal.

If the failed Tomcat is recovered from the failed state to normal state the load balancer will include it in the cluster to receive requests.

7.2 Configuring Tomcat

7.2.1 Setting AJP ports

Traffic is passed between Apache and Tomcat(s) uses the binary AJP 1.3 protocol.

Application Server	Http port	AJP port	Requests
Tomcat – instance 1	Tomcat 1 port	8209	Connector, Proxy Service
Tomcat – instance 2	Tomcat 2 port	8309	Connector, Proxy Service
Tomcat - instance 3	Tomcat 3 port	8409	SP, IDP

7.2.2 Apache HTTPD

In this section we will use **Apache httpd** web server as a Load Balancer.
To provide the load balancing capability to Apache httpd server we need to include the module **mod_jk**.

7.2.2.1 Install and configure mod_jk

The **mod_jk** module is downloaded from http://www.apache.org/dist/tomcat/tomcat-connectors/jk/binaries/.

mod_jk is the Apache HTTPD module that will be used to provide our cluster with its load balancing and proxy capabilities, by default it uses the 'round robin' algorithm to distribute the requests. It uses the AJP protocol to facilitate fast communication between Tomcat servers and the Apache Web Server that will receive the client requests.

Configuration consists of adding a few lines to the main Apache HTTPD configuration file httpd.conf:

```
JkMount /status stat
JkMount /EidasNode/* balancer
JkMount /SpecificConnector/* tomcat3
JkMount /SpecificProxyService/* tomcat3
JkMount /SP/* tomcat3
JkMount /IdP/* tomcat3
```

7.2.2.2 Configure the cluster workers

'Workers' is a blanket term used within **mod_jk** to refer to both real Tomcat servers that will process requests, and virtual servers included in the module to handle load balancing and monitoring.

<u>File: workers.properties</u>

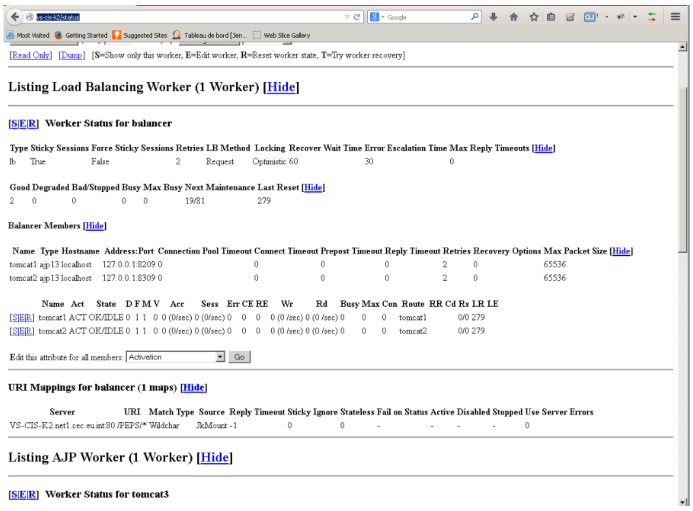
By default, mod_jk includes three additional load-balancing algorithms, some of which are more appropriate for certain situations, and can be configured with the 'method' directive:

```
worker.list=balancer,stat,tomcat3
worker.tomcat1.type=ajp13
worker.tomcat1.port=8209
worker.tomcat1.host=localhost
worker.tomcat2.type=ajp13
worker.tomcat2.port=8309
worker.tomcat2.host=localhost
worker.tomcat3.type=ajp13
worker.tomcat3.port=8409
worker.tomcat3.host=localhost
worker.balancer.type=lb
worker.balancer.balance_workers=tomcat1,tomcat2
```

7.3 Check your installation

The loadbalance configuration can be checked on the Apache status page. You can enable this by adding "worker.stat.type=status" (temporarily) to the workers.properties file described in 7.2.2.2.

After this restart and open the Apache status page: http://localhost/status and check that each node is up and running.



[Read Only] [Dump] [S=Show only this worker, E=Edit worker, R=Reset worker state, T=Try worker recovery]

Listing Load Balancing Worker (1 Worker) [Hide]

[S|E|R] Worker Status for balancer

Type	Sticky Sessions	Force Sticky Sessions	Retries	LB Method	Locking	Recover Wait Time	Error Escalation Time	Max Reply Timeouts [Hide]
lb	True	False	2	Request	Optimistic	60	30	0

Good	Degraded	Bad/Stopped	Busy	Max Busy	Next Maintenance	Last Reset [Hide]
2	0	0	0	0	19/81	279

Balancer Members [Hide]

Name	Type	Hostname	Address:Port	Connection Pool Timeout	Connect Timeout	Prepost Timeout	Reply Timeout	Retries	Recovery Options	Max Packet Size [Hide]
tomcat1	ajp13	localhost	127.0.0.1:8209	0	0	0	0	2	0	65536
tomcat2	ajp13	localhost	127.0.0.1:8309	0	0	0	0	2	0	65536

	Name	Act	State	D	F	M	V	Acc	Sess	Err	CE	RE	Wr	Rd	Busy	Max	Con	Route	RR	Cd	Rs	LR	LE
[S|E|R]	tomcat1	ACT	OK/IDLE	0	1	1	0	0 (0/sec)	0 (0/sec)	0	0	0	0 (0 /sec)	0 (0 /sec)	0	0	0	tomcat1			0/0	279	
[S|E|R]	tomcat2	ACT	OK/IDLE	0	1	1	0	0 (0/sec)	0 (0/sec)	0	0	0	0 (0 /sec)	0 (0 /sec)	0	0	0	tomcat2			0/0	279	

Edit this attribute for all members: Activation Go

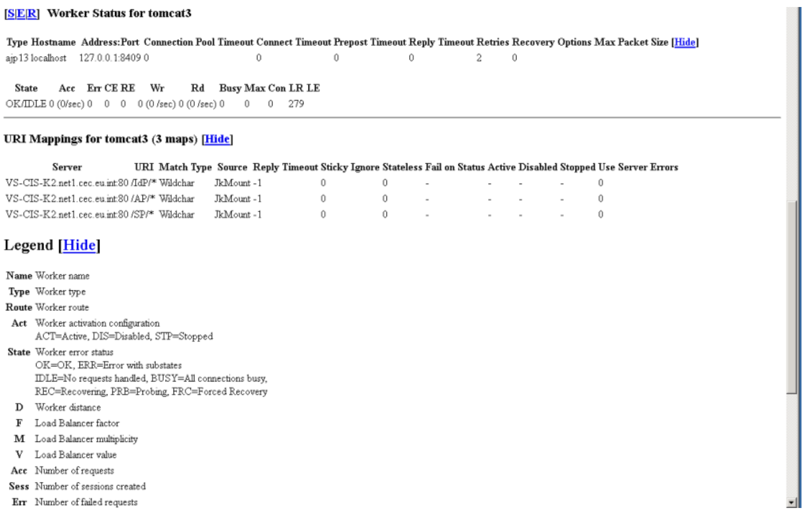
URI Mappings for balancer (1 maps) [Hide]

Server	URI	Match Type	Source	Reply Timeout	Sticky Ignore	Stateless	Fail on Status	Active	Disabled	Stopped	Use Server Errors
VS-CIS-K2.net1.cec.eu.int:80	/PEPS/*	Wildchar	JkMount	-1	0	0	-	-	-	-	0

Listing AJP Worker (1 Worker) [Hide]

[S|E|R] Worker Status for tomcat3

Figure 7: Apache status page



[S|E|R] Worker Status for tomcat3

Type	Hostname	Address:Port	Connection Pool Timeout	Connect Timeout	Prepost Timeout	Reply Timeout	Retries	Recovery Options	Max Packet Size [Hide]
ajp13	localhost	127.0.0.1:8409	0	0	0	0	2	0	

State	Acc	Err	CE	RE	Wr	Rd	Busy	Max	Con	LR	LE
OK/IDLE	0 (0/sec)	0	0	0	0 (0 /sec)	0 (0 /sec)	0	0	0	279	

URI Mappings for tomcat3 (3 maps) [Hide]

Server	URI	Match Type	Source	Reply Timeout	Sticky Ignore	Stateless	Fail on Status	Active	Disabled	Stopped	Use Server Errors
VS-CIS-K2.net1.cec.eu.int:80	/IdP/*	Wildchar	JkMount	-1	0	0	-	-	-	-	0
VS-CIS-K2.net1.cec.eu.int:80	/AP/*	Wildchar	JkMount	-1	0	0	-	-	-	-	0
VS-CIS-K2.net1.cec.eu.int:80	/SP/*	Wildchar	JkMount	-1	0	0	-	-	-	-	0

Legend [Hide]

Name	Worker name
Type	Worker type
Route	Worker route
Act	Worker activation configuration ACT=Active, DIS=Disabled, STP=Stopped

State	Worker error status
	OK=OK, ERR=Error with substates
	IDLE=No requests handled, BUSY=All connections busy,
	REC=Recovering, PRB=Probing, FRC=Forced Recovery

D	Worker distance
F	Load Balancer factor
M	Load Balancer multiplicity
V	Load Balancer value
Acc	Number of requests
Sess	Number of sessions created
Err	Number of failed requests

Figure 8: Apache status page (continued)

7.4 eIDAS-Node compliance

To ensure the eIDAS compliance, there is a list of parameters to specifically set. Those parameters are listed below.

Table 21: eIDAS-Node compliance

Parameter	Resulting value
disallow.self.signed.certificate	False: Allows self-signed certificates
check.certificate.validity.period	True: do not allow expired certificates
metadata.activate	True: specifies that metadata is generated
metadata.restrict.http	True : metadata must be only available via HTTPS
tls.enabled.protocols	TLSv1.2: SSL/TLS enabled protocols
tls.enabled.ciphers	The eIDAS supported cipher suites have been consolidated according to the eIDAS-Crypto requirements and the eIDAS Technical Specifications v1.2.. Default cipher suites for java 11 are: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384, TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, TLS_DHE_RSA_WITH_AES_256_CBC_SHA256, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, TLS_EMPTY_RENEGOTIATION_INFO_SCSV
metadata.check.signature	True: metadata received from a partner must be signed
metadata.validity.duration	Metadata validity period in seconds. Default=86400 (i.e. one day)
validate.binding	True: the bindings are validated
security.header.csp.enabled	True: the content-security and security checks are enabled (HSTS, Mozilla directives, X-content-Type-Options, X-frame-options)
check.citizencertificate.serviceCertificate	True: check if the CN of the certificate used for signing the response is the same as the requested citizen country

Note that to ensure compliance, the following checks are also made by the code and are not parametrized:

- One of the Levels of Assurance indicated in the Assertion matches or exceeds at least one of the requested Levels of Assurance (see Appendix A); and
- the Response will not be transmitted to a URL other than the AssertionConsumerServiceURL in the metadata of the eIDAS-Node Connector.

Remark: To improve the resilience of the application, we strongly recommend using the cache instances used for request anti-replay and SAML metadata using Ignite services. (please see Appendix C for further details).

7.5 Ignite shared map definitions

When using Ignite, the caches will have dedicated identifiers. When using the default configuration, these will have default values as detailed in the table below.

Table 22: Ignite shared maps default identifiers

Map name	Role
specificNodeConnectorRequestCache	Stores Requests sent from Specific Connector to Generic Connector
nodeSpecificConnectorResponseCache	Stores Responses sent from Generic Connector to Specific Connector.
nodeSpecificProxyserviceRequestCache	Stores Requests sent from Generic Proxy Service to Specific Proxy Service
specificNodeProxyserviceResponseCache	Stores Responses sent from Specific Proxy Service to Generic Proxy Service

However, it is possible to override the default values. This can be done in *specificCommunicationDefinitionConnector.xml* and in *specificCommunicationDefinitionProxyService.xml*. Note that if these values are changed, these changes must be reflected in the Ignite configuration.

8 Appendix A: eIDAS Levels of Assurance

Level of Assurance (LoA) is a term used to describe the degree of certainty that an individual is who they say they are at the time they present a digital credential.

The eIDAS implementing regulation determines three Levels of Assurance, known as Notified Level of Assurance:

- **Low** (*service.LoA=http://eidas.europa.eu/LoA/low*)
- **Substantial** (*service.LoA=http://eidas.europa.eu/LoA/substantial*)
- **High** (*service.LoA=http://eidas.europa.eu/LoA/high*)

(The eIDAS-Node Proxy Service *service.LoA* key is described in Table 5.)

Non-notified Levels of Assurance values can also be used, these values must start with a different prefix than "http://eidas.europa.eu/LoA".

At the SAML Request level, if no non-notified levels of assurance are requested, the level of assurance will limit the comparison attribute to 'minimum':

```
<saml2p:RequestedAuthnContext Comparison="minimum">
```

Otherwise, the default comparison ("exact") will be used, and if a notified Level of Assurance was specified in the LightRequest, higher notified Levels of Assurance will be added to the RequestedAuthnContext.

Validations made:

The eIDAS-Node Proxy Service will reject the request if:

- in the case request's Comparison is Exact and none of the request's LoA(s) matches the published LoAs in the Proxy-Service Metadata.
- in the case Comparison is Minimum and if the request's notified LoA is higher than the published notified LoA in the Proxy-Service Metadata.

For nodes supporting the eIDAS specification 1.2, since all LoAs have to be published, at the Proxy-Service metadata, the Connector supporting the eIDAS specification 1.2, will apply exact matching LOA validation to incoming responses.

At the eIDAS specification 1.2 Proxy-Service's, exact matching LOA validation will also be applied, for the incoming Light Response towards the Proxy-Service metadata published LOAs.

The legal definitions of the Level of Assurance can be found at http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:JOL_2015_235_R_0002&from=EN.

9 Appendix B: User consent

In most Member States (MS), the privacy legislation requires that the user gives consent to the use of personal data. But the explanation of this requisite, and thus its implementation may be very different from one MS to another MS. So this general objective to request the consent of the user to send his/her attributes to a Service Provider in another Member State leads to the following consent schemes. The consent is requested by the eIDAS-Node or by the Middleware of the user's MS.

There are three possible cases:

- The requested attributes are displayed and the user's consent is given by choosing only the attributes that he/she allows to transfer.
- The obtained values of the requested attributes are displayed and the user's consent is given by choosing only the attributes that he/she allows to transfer.
- The requested attributes are not displayed because the user's consent is not required as it was given (for example) when the user registered to the ID Provider.

10 Appendix C: Ignite proposed configuration

For the communication caches, an Ignite implementation can be used. The configuration of the product is done via its configuration file ignite.xml located by EIDAS_CONNECTOR_CONFIG_REPOSITORY and EIDAS_PROXY_CONFIG_REPOSITORY. A default configuration is provided with the application, e.g. in igniteNode.xml file, we can see two simple configurations for the connector and proxy service respectively using Ignite Jcache as copied below. Note that, for more support on Ignite configuration, the Ignite Documentation should be used.

for the Connector:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  ~ Copyright (c) 2023 by European Commission
  ~
  ~ Licensed under the EUPL, Version 1.2 or - as soon they will be
  ~ approved by the European Commission - subsequent versions of the
  ~ EUPL (the "Licence");
  ~ You may not use this work except in compliance with the Licence.
  ~ You may obtain a copy of the Licence at:
  ~ https://joinup.ec.europa.eu/page/eupl-text-11-12
  ~
  ~ Unless required by applicable law or agreed to in writing, software
  ~ distributed under the Licence is distributed on an "AS IS" basis,
  ~ WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
  ~ implied.
  ~ See the Licence for the specific language governing permissions and
  ~ limitations under the Licence.
  -->

<!--
    Ignite Spring configuration file to startup Ignite cache.

    This file demonstrates how to configure cache using Spring. Provided
cache
    will be created on node startup.

    Use this configuration file when running HTTP REST examples (see
'examples/rest' folder).

    When starting a standalone node, you need to execute the following
command:
    {IGNITE_HOME}/bin/ignite.{bat|sh} examples/config/ignite-cache.xml

    When starting Ignite from Java IDE, pass path to this file to
Ignition:
    Ignition.start("examples/config/ignite-cache.xml");
-->


<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="igniteNode.cfg"
class="org.apache.ignite.configuration.IgniteConfiguration">

        <property name="igniteInstanceName" value="igniteNode"/>

        <property name="cacheConfiguration">
            <list>
                <!-- Partitioned cache example configuration (Atomic
mode). -->
                <bean
class="org.apache.ignite.configuration.CacheConfiguration">
                    <property name="name"
value="antiReplayCacheConnector"/>
                    <property name="atomicityMode" value="ATOMIC"/>
                    <property name="backups" value="1"/>
                    <property name="expiryPolicyFactory"
ref="3_hours_duration"/>
                </bean>
                <!-- Partitioned cache example configuration (Atomic
mode). -->
```

```
                <bean
class="org.apache.ignite.configuration.CacheConfiguration">
                    <property name="name"
value="connectorRequestCorrelationCacheService"/>
                    <property name="atomicityMode" value="ATOMIC"/>
                    <property name="backups" value="1"/>
                    <property name="expiryPolicyFactory"
ref="7_minutes_duration"/>
                </bean>
                <!-- Partitioned cache example configuration (Atomic
mode). -->
                <bean
class="org.apache.ignite.configuration.CacheConfiguration">
                    <property name="name"
value="specificConnectorLtRequestCorrelationCacheService"/>
                    <property name="atomicityMode" value="ATOMIC"/>
                    <property name="backups" value="1"/>
                    <property name="expiryPolicyFactory"
ref="7_minutes_duration"/>
                </bean>
                <bean
class="org.apache.ignite.configuration.CacheConfiguration">
                    <property name="name"
value="connectorFlowIdCacheService"/>
                    <property name="atomicityMode" value="ATOMIC"/>
                    <property name="backups" value="1"/>
                    <property name="expiryPolicyFactory"
ref="7_minutes_duration"/>
                </bean>
                <!-- Partitioned cache example configuration (Atomic
mode). -->
                <bean
class="org.apache.ignite.configuration.CacheConfiguration">
                    <property name="name" value="eidasmetadata"/>
                    <property name="atomicityMode" value="ATOMIC"/>
                    <property name="backups" value="1"/>
                    <property name="expiryPolicyFactory"
ref="24_hours_duration"/>
                </bean>
            </list>
        </property>

        <!--Explicitly configure TCP discovery SPI to provide list of
initial nodes from the first cluster.-->
        <property name="discoverySpi">
            <bean
class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
                <!-- Initial local port to listen to. -->
                <property name="localPort" value="48500"/>

                <!-- Changing local port range. This is an optional
action. -->
                <property name="localPortRange" value="4"/>

                <!-- Setting up IP finder for this cluster -->
                <property name="ipFinder">
                    <bean
class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFin
der">
                        <property name="addresses">
                            <list>
                                <!--
                                Addresses and port range of nodes from
                                the first cluster.
                                127.0.0.1 can be replaced with actual IP
addresses
```

```
                                or host names. Port range is optional.
                                -->
                                <value>127.0.0.1:48500..48503</value>
                            </list>
                        </property>
                    </bean>
                </property>
            </bean>
        </property>

        <!--
        Explicitly configure TCP communication SPI changing local
        port number for the nodes from the first cluster.
        -->
        <property name="communicationSpi">
            <bean
class="org.apache.ignite.spi.communication.tcp.TcpCommunicationSpi">
                <property name="localPort" value="48100"/>
            </bean>
        </property>

        <!-- Ssl/Tls context. -->
        <!--IMPORTANT: THIS IS A DEMO CONFIGURATION AND DEMO KEYSTORES, IT
NEEDS TO BE CHANGED FOR PRODUCTION ALSO THE KEYS IN THE KEYSTORES NEED TO
BE CREATED AS WELL-->
        <property name="sslContextFactory">
            <bean class="org.apache.ignite.ssl.SslContextFactory">
                <!--uncomment the below keyAlgorithm when IBM JRE is used
e.g. websphere 8.5.5-->
                <!--<property name="keyAlgorithm" value="IBMX509" />-->
                <property name="keyStoreFilePath"
value="${EIDAS_CONNECTOR_CONFIG_REPOSITORY}/ignite/KeyStore/server.p12"/>
                <property name="keyStorePassword"
value="${IGNITE_NODE_CONNECTOR_KEY_STORE_PASSWORD:123456}"/>
                <property name="trustStoreFilePath"
value="${EIDAS_CONNECTOR_CONFIG_REPOSITORY}/ignite/KeyStore/trust.p12"/>
                <property name="trustStorePassword"
value="${IGNITE_NODE_CONNECTOR_TRUST_STORE_PASSWORD:123456}"/>
                <property name="protocol" value="TLSv1.2"/>
            </bean>
        </property>

        <!-- how frequently Ignite will output basic node metrics into the
log-->
        <property name="metricsLogFrequency" value="#{60 * 10 * 1000}"/>

    </bean>

    <!--
        Initialize property configurer so we can reference environment
variables.
    -->
    <bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigu
rer">
        <property name="systemPropertiesModeName"
value="SYSTEM_PROPERTIES_MODE_FALLBACK"/>
        <property name="searchSystemEnvironment" value="true"/>
    </bean>

    <!--
        Defines expiry policy based on moment of creation for ignite
cache.
    -->
    <bean id="7_minutes_duration"
class="javax.cache.expiry.CreatedExpiryPolicy" factory-method="factoryOf"
scope="prototype">
        <constructor-arg>
            <bean class="javax.cache.expiry.Duration">
                <constructor-arg value="MINUTES"/>
                <constructor-arg value="7"/>
            </bean>
        </constructor-arg>
    </bean>

    <!--
    Defines expiry policy based on moment of creation for ignite cache.
-->
    <bean id="3_hours_duration"
class="javax.cache.expiry.CreatedExpiryPolicy" factory-method="factoryOf"
scope="prototype">
        <constructor-arg>
            <bean class="javax.cache.expiry.Duration">
                <constructor-arg value="HOURS"/>
                <constructor-arg value="3"/>
            </bean>
        </constructor-arg>
    </bean>

    <!--
    Defines expiry policy based on moment of creation for ignite cache.
-->
    <bean id="24_hours_duration"
class="javax.cache.expiry.CreatedExpiryPolicy" factory-method="factoryOf"
scope="prototype">
        <constructor-arg>
            <bean class="javax.cache.expiry.Duration">
                <constructor-arg value="HOURS"/>
                <constructor-arg value="24"/>
            </bean>
        </constructor-arg>
    </bean>
</beans>
```

For the Proxy Service:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  ~ Copyright (c) 2023 by European Commission
  ~
  ~ Licensed under the EUPL, Version 1.2 or - as soon they will be
  ~ approved by the European Commission - subsequent versions of the
  ~ EUPL (the "Licence");
  ~ You may not use this work except in compliance with the Licence.
  ~ You may obtain a copy of the Licence at:
  ~ https://joinup.ec.europa.eu/page/eupl-text-11-12
  ~
  ~ Unless required by applicable law or agreed to in writing, software
  ~ distributed under the Licence is distributed on an "AS IS" basis,
  ~ WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
  ~ implied.
  ~ See the Licence for the specific language governing permissions and
  ~ limitations under the Licence.
  -->

<!--
    Ignite Spring configuration file to startup Ignite cache.

    This file demonstrates how to configure cache using Spring. Provided
cache
    will be created on node startup.

    Use this configuration file when running HTTP REST examples (see
'examples/rest' folder).

    When starting a standalone node, you need to execute the following
command:
    {IGNITE_HOME}/bin/ignite.{bat|sh} examples/config/ignite-cache.xml

    When starting Ignite from Java IDE, pass path to this file to
Ignition:
    Ignition.start("examples/config/ignite-cache.xml");
-->


<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="igniteNode.cfg"
class="org.apache.ignite.configuration.IgniteConfiguration">

        <property name="igniteInstanceName" value="igniteNode"/>

        <property name="cacheConfiguration">
            <list>
                <!-- Partitioned cache example configuration (Atomic
mode). -->
                <bean
class="org.apache.ignite.configuration.CacheConfiguration">
                    <property name="name" value="antiReplayCacheService"/>
                    <property name="atomicityMode" value="ATOMIC"/>
                    <property name="backups" value="1"/>
                    <property name="expiryPolicyFactory"
ref="3_hours_duration"/>
                </bean>
                <!-- Partitioned cache example configuration (Atomic
mode). -->
```

```
                <bean
class="org.apache.ignite.configuration.CacheConfiguration">
                    <property name="name"
value="proxyServiceRequestCorrelationCacheService"/>
                    <property name="atomicityMode" value="ATOMIC"/>
                    <property name="backups" value="1"/>
                    <property name="expiryPolicyFactory"
ref="7_minutes_duration"/>
                </bean>
                <bean
class="org.apache.ignite.configuration.CacheConfiguration">
                    <property name="name"
value="proxyServiceFlowIdCacheService"/>
                    <property name="atomicityMode" value="ATOMIC"/>
                    <property name="backups" value="1"/>
                    <property name="expiryPolicyFactory"
ref="7_minutes_duration"/>
                </bean>
                <!-- Partitioned cache example configuration (Atomic
mode). -->
                <bean
class="org.apache.ignite.configuration.CacheConfiguration">
                    <property name="name" value="eidasmetadata"/>
                    <property name="atomicityMode" value="ATOMIC"/>
                    <property name="backups" value="1"/>
                    <property name="expiryPolicyFactory"
ref="24_hours_duration"/>
                </bean>
            </list>
        </property>

        <!--Explicitly configure TCP discovery SPI to provide list of
initial nodes from the first cluster.-->
        <property name="discoverySpi">
            <bean
class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
                <!-- Initial local port to listen to. -->
                <property name="localPort" value="48500"/>

                <!-- Changing local port range. This is an optional
action. -->
                <property name="localPortRange" value="4"/>

                <!-- Setting up IP finder for this cluster -->
                <property name="ipFinder">
                    <bean
class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFin
der">
                        <property name="addresses">
                            <list>
                                <!--
                                Addresses and port range of nodes from
                                the first cluster.
                                127.0.0.1 can be replaced with actual IP
addresses

                                or host names. Port range is optional.
                                -->
                                <value>127.0.0.1:48500..48503</value>
                            </list>
                        </property>
                    </bean>
                </property>
            </bean>
        </property>

        <!--
        Explicitly configure TCP communication SPI changing local
        port number for the nodes from the first cluster.
        -->
        <property name="communicationSpi">
            <bean
class="org.apache.ignite.spi.communication.tcp.TcpCommunicationSpi">
                <property name="localPort" value="48100"/>
            </bean>
        </property>

        <!-- Ssl/Tls context. -->
        <!--IMPORTANT: THIS IS A DEMO CONFIGURATION AND DEMO KEYSTORES, IT
NEEDS TO BE CHANGED FOR PRODUCTION ALSO THE KEYS IN THE KEYSTORES NEED TO
BE CREATED AS WELL-->
        <property name="sslContextFactory">
            <bean class="org.apache.ignite.ssl.SslContextFactory">
                <!--uncomment the below keyAlgorithm when IBM JRE is used
e.g. websphere 8.5.5-->
                <!--<property name="keyAlgorithm" value="IBMX509" />-->
                <property name="keyStoreFilePath"
value="${EIDAS_PROXY_CONFIG_REPOSITORY}/ignite/KeyStore/server.p12"/>
                <property name="keyStorePassword"
value="${IGNITE_NODE_PROXY_KEY_STORE_PASSWORD:123456}"/>
                <property name="trustStoreFilePath"
value="${EIDAS_PROXY_CONFIG_REPOSITORY}/ignite/KeyStore/trust.p12"/>
                <property name="trustStorePassword"
value="${IGNITE_NODE_PROXY_TRUST_STORE_PASSWORD:123456}"/>
                <property name="protocol" value="TLSv1.2"/>
            </bean>
        </property>

        <!-- how frequently Ignite will output basic node metrics into the
log-->
        <property name="metricsLogFrequency" value="#{60 * 10 * 1000}"/>

    </bean>

    <!--
        Initialize property configurer so we can reference environment
variables.
    -->
    <bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigu
rer">
        <property name="systemPropertiesModeName"
value="SYSTEM_PROPERTIES_MODE_FALLBACK"/>
        <property name="searchSystemEnvironment" value="true"/>
    </bean>

    <!--
        Defines expiry policy based on moment of creation for ignite
cache.
    -->
    <bean id="7_minutes_duration"
class="javax.cache.expiry.CreatedExpiryPolicy" factory-method="factoryOf"
scope="prototype">
        <constructor-arg>
            <bean class="javax.cache.expiry.Duration">
                <constructor-arg value="MINUTES"/>
                <constructor-arg value="7"/>
            </bean>
        </constructor-arg>
    </bean>

    <!--
    Defines expiry policy based on moment of creation for ignite cache.
```

```
-->
    <bean id="3_hours_duration"
class="javax.cache.expiry.CreatedExpiryPolicy" factory-method="factoryOf"
scope="prototype">
        <constructor-arg>
            <bean class="javax.cache.expiry.Duration">
                <constructor-arg value="HOURS"/>
                <constructor-arg value="3"/>
            </bean>
        </constructor-arg>
    </bean>

    <!--
    Defines expiry policy based on moment of creation for ignite cache.
-->
    <bean id="24_hours_duration"
class="javax.cache.expiry.CreatedExpiryPolicy" factory-method="factoryOf"
scope="prototype">
        <constructor-arg>
            <bean class="javax.cache.expiry.Duration">
                <constructor-arg value="HOURS"/>
                <constructor-arg value="24"/>
            </bean>
        </constructor-arg>
    </bean>
</beans>
```

10.1 Enable Cache Expiry Policy

One feature to add to each one of the caches is the duration adding the expiryPolicyFactory property cache by cache.

```
<!-- Partitioned cache example configuration (Atomic mode). -->
<bean class="org.apache.ignite.configuration.CacheConfiguration">
…
    <property name="expiryPolicyFactory">
        <bean class="javax.cache.expiry.CreatedExpiryPolicy" factory-
method="factoryOf">
            <constructor-arg>
                <bean class="javax.cache.expiry.Duration">
                    <constructor-arg value="MINUTES"/>
                    <constructor-arg value="7"/>
                </bean>
            </constructor-arg>
        </bean>
    </property>
</bean>
```

This is now default as shown by the bean "7_minutes_duration" above.

10.2 Enable TLSv1.2 Ignite nodes

In the configuration files, the TLSv1.2 is enabled for Ignite Nodes if at ignite.xml, the following is present:

```
<!-- Ssl/Tls context. -->
    <!-IMPORTANT: THIS IS JUST A DEMO CONFIGURATION AND DEMO KEYSTORES,
NEEDS TO BE CHANGED FOR PRODUCTION ALSO THE KEYS IN THE KEYSTORES NEED TO
BE CREATED AS WELL->
    <property name="sslContextFactory">
        <bean class="org.apache.ignite.ssl.SslContextFactory">
            <property name="keyStoreFilePath"
value="${EIDAS_CONNECTOR_CONFIG_REPOSITORY}/ignite/KeyStore/server.p12"/>
            <property name="keyStorePassword"
value="${IGNITE_NODE_CONNECTOR_KEY_STORE_PASSWORD:123456}"/>
            <property name="trustStoreFilePath"
value="${EIDAS_CONNECTOR_CONFIG_REPOSITORY}/ignite/KeyStore/trust.p12"/>
            <property name="trustStorePassword"
value="${IGNITE_NODE_CONNECTOR_TRUST_STORE_PASSWORD:123456}"/>
            <property name="protocol" value="TLSv1.2"/>
        </bean>
    </property>
```

Of course the corresponding keystores:

Will need to be present at

EIDAS-Config/server/connector/ignite/KeyStore/server.p12

EIDAS-Config/server/connector/ignite/KeyStore/trust.p12

EIDAS-Config/server/proxy/ignite/KeyStore/server.p12

EIDAS-Config/server/proxy/ignite/KeyStore/trust.p12

This default configuration works with an Oracle (Sun) Java Runtime. In the case you are using IBM Java Runtime, you will need the additional "keyAlgorithm" property with the value "IBMX509" in the SslContextFactory configuration.

```
<property name="sslContextFactory">
    <bean class="org.apache.ignite.ssl.SslContextFactory">
        <property name="keyAlgorithm" value="IBMX509" />
…
        <property name="protocol" value="TLSv1.2" />
    </bean>
</property>
```

For further information regarding this topic please refer to Ignite documentation pages, e.g. *https://apacheignite.readme.io/docs/ssltls*

10.3 Enable Ignite Update Notifier

The default behaviour of Ignite to verify if the version of Ignite, which is used, is the latest has been deactivated for the Node.

To activate this, the system property "IGNITE_UPDATE_NOTIFIER" needs to be set to true, otherwise, it will not be active.

11 Appendix D: Installation Frequently Asked Questions

Q: How can I compile the project using external properties (Tomcat)?

A: First you compile EIDAS-NODE and EIDAS-Specific without the "-P embedded" argument. This will generate the packages without specific properties. Now you need to place all the properties files in one folder and tell Tomcat to lookup that folder.

If in Linux:
Edit $TOMCAT_HOME/bin/catalina.sh and change
"C*LASSPATH="$CLASSPATH""$CATALINA_HOME"/bin/bootstrap.jar*" to
"*CLASSPATH="$CLASSPATH""$CATALINA_HOME"/bin/bootstrap.jar:/path/to/config/folder/*
"

If in Windows:
Edit $TOMCAT_HOME/bin/catalina.bat and change
"*CLASSPATH="$CLASSPATH""$CATALINA_HOME"/bin/bootstrap.jar*" to
"*CLASSPATH="$CLASSPATH""$CATALINA_HOME"/bin/bootstrap.jar:/path/to/config/folder/*
"

Q: I'm getting an error that says "Failed to load class org.slf4j.impl.StaticLoggerBinder" .

A: This error is reported when the *org.slf4j.impl.StaticLoggerBinder* class could not be loaded into memory. In this case, you should recompile your projects to ensure that Maven includes the appropriate jars.

Q: I'm getting an error that says "com.opensymphony.xwork2.DefaultActionInvocation.invokeAction (DefaultActionInvocation.java)" .

A: The *DefaultActionInvocation* class is responsible for calling the user action, if an error occurs, generally due to missing libraries or missing properties file, the struts framework will not be able to render the result of the action, thus producing that error message. However, in the logs or the stack trace you can usually find another exception. That exception is the reason for this error, perhaps you can solve it by making sure:

- you have the properties files in the right place
- you have the right privileges to access p12 file (you may need to install JCE and allow Java to read the file outside the webapp context)
- you have all the required libraries.