ORACLE INSURANCE

Oracle Health Insurance Back Office

HTTP Service Layer (HSL)
Installation & Configuration Manual

Version 1.49

Part number: G49637-01

March 20, 2026

CHANGE HISTORY

Release	Version	Changes
10.16.2.2.0	1.0	• Creation
10.16.2.2.0	1.1	• Revision
10.17.1.0.0	1.2	• Changed grant instructions
10.17.2.0.0	1.3	• Documented hsl.<app>.developermode and hsl.developermode • Added reference to Doc[2] (Back Office HTTP Service Layer User Manual)
10.17.2.1.0	1.4	• Extended set of relevant properties.
10.17.2.2.0	1.5	• Minor revision of 'Creating a HSL database account' • Revised 'Security Configuration' • Removed 'Restricting access with custom roles' from Security Aspects • Renamed 'Security Aspects' to 'Additional Security Aspects' and revised contents. • Added 'Deployment Validation' • Added 'Appendix C – Testing with SoapUI' • Added 'Appendix D – Generating a WADL file' • Extended set of relevant properties for HSL_CLA.war deployment.
10.17.2.3.0	1.6	• Added paragraph 'Examining the Log File' • Added JDK version specific information regarding JSSE configuration.
10.18.1.0.0	1.7	• Added Appendix E – Authentication and Authorization • Added Appendix F – HSL_AUN and HSL_AUZ Services • Revised 2.1 including diagram • Revised introduction and document title
10.18.1.2.0	1.8	• Added Appendix G – PSL services • Use setUserOverrides.sh instead of modifying startManagedWebLogic.sh and Server Start arguments. Support for WLS 12.2.1.3.
10.18.1.3.0	1.9	• Added warning about patch **28278427**
10.18.1.3.0	1.10	• Revised installation of PSL services
10.18.1.4.0	1.11	• Updated 'Additional Security Aspects' Mentioned use of Basic Authentication for 'admin' requests. • Updated 'setting user context' in Appendix E • Added HTTP codes to 'Troublehooting'
10.18.1.4.0	1.12	• Updated HTTP codes overview in 'Troubleshooting' • Revised Appendix G – PSL services
10.18.2.0.0	1.13	• Corrected typo: hsl.auz, not hsl.aur. • Rewrote explanation of setUserOverrides.sh
10.18.2.0.0	1.14	• Services Components: added HSL_AUN and HSL_AUZ • WLS Preparation: fixed typos • Additional Security Aspects: removed remarks about patch 28278427 (already covered by certification matrix)
10.18.2.1.0	1.15	• Added example properties for new PSL service with shortname zvp • Added description of allowedorigins property to protect against cross site scripting attack
10.18.2.2.0	1.16	• Added additional deployment notes for HSL_C2B
10.18.2.3.0	1.17	• Added Appendix for HSL_JUP • Introduction of properties file templates
10.19.1.0.0	1.18	• No changes. Republished with different part nr.
10.19.1.4.0	1.19	• Added properties for logdirectory and logfilesuffix • The operation of developermode changed • Change in properties template is described • Updated description of data source creation
10.19.2.0.0	1.20	• Added description of deployment check verifyInterfaceVersion
10.20.1.0.0	1.21	• No changes, republished.
10.20.3.0.0	1.22	• Adapted for impact of DB 19c and FMW12.2.1.4 certification
10.20.4.0.0	1.23	• Added new recommended Initial Capacity configuration option for data sources in Creating a Data Source paragraph
10.20.7.0.0	1.24	• Introduced HSLBOWS.ear and adapted deployment instructions and references to individual web service deployments. • Introduced Appendix I as an example of limiting access to web applications through the use of a security policy
10.21.1.0.0	1.25	• No changes, republished with new part number
10.21.2.0.0	1.26	• Added descriptions about Security Models

Release	Version	Changes
10.21.5.0.0	1.27	• Added more information on the Custom Roles security model
10.21.7.0.0	1.28	• Custom Roles: added instructions • Added Statement Timeout
10.22.1.0.0	1.29	• Replaced base64 encoded strings with place holders; updated paragraph 3.7 for CustomRoles. Republished with new part number.
10.22.2.0.0	1.30	• Added property maxcontentlength
10.22.5.0.0	1.31	• Added remark about financial unit in hsl.usercontext in paragraph 5.2.3
10.22.8.0.0	1.32	• Added Authentication with Oracle Access Manager to appendix E.
10.22.3.0.0	1.33	• No changes, republished with a new part number.
10.23.1.0.0	1.34	• Removed reference to online swagger editor
10.23.3.0.0	1.35	• Corrected properties names for loglimit, logcount and logappend (paragraph 5.2.9, 5.2.10 and 5.2.11) • Recommended log level settings for Production (paragraphs 5.2.8, 13.1)
10.23.6.0.0	1.36	• Clarified impact of certification of Forms & Reports Services 12.2.1.19.0
10.23.6.0.0	1.37	• Added Authentication with OpenID to appendix E.
10.23.7.0.0	1.38	• Chapter 4.2 "template listing" (path */templates) removed.
10.23.7.0.0	1.39	• Added mentions of OpenApi 3.1 support. • Added information on the OpenApi 3.1 runtime documentation url. • Removed dots from the chapter numbers, e.g. ..3.5.2.1 to 3.5.2.1. • Added Appendix I as a regular numbered chapter instead of as an unnumbered appendix.
10.23.8.0.0	1.40	• Added restuserbsn and restuserbsn-role for the HSL_BSNCHECK service
10.24.1.0.0	1.41	• Republished with new part number. • Corrected some typing errors. • Removed mention of support for changing the language used for messages in a webservice based on the user context as this support has been removed.
10.24.2.0.0	1.42	• Added restuseragb and restuseragb-role for the HSL_AGBCHECK service (service will be delivered in the 10.24.3.0.0 release)
10.24.5.0.0	1.43	• Corrected some typing errors. • Added Appendix J with more information regarding the monitoring and diagnostics options provided by Jersey.
10.24.8.0.0	1.44	• Added a description of an HSL OAuth2.0 setup using Private Key JWT as a form of client authentication. As a result of this change the default authentication method for the HSL services has changed from BASIC Authentication to OAuth2.0. • The instructions for setting up the HSL services with BASIC Authentication have been moved to Appendix A. • Moved the following appendices to **Doc[3]** as these chapters are only applicable to the Private Service Layer (PSL) webservices that support the OHI JET web application: Appendix E – Authentication and Authorization Appendix F – HSL_AUN and HSL_AUZ Services Appendix G – HSL_JUP Service • Moved the following webservices to PSL: HSL_AUN → PSL_AUN HSL_AUZ → PSL_AUZ HSL_JUP → PSL_JUP • Moved the following properties to psl.properties and removed their use for HSL: hsl.sso.platform hsl.sso.openid.authorizationendpoint hsl.sso.openid.endsessionendpoint hsl.sso.openid.tokenendpoint hsl.sso.openid.clientid hsl.sso.openid.clientsecret • Removed the following properties in hsl.properties: hsl.<app>.tokenvalidation.rotor hsl.<app>.tokenvalidation.url hsl.<app>.tokenvalidation.method hsl.<app>.tokenvalidation.headerparams hsl.<app>.tokenvalidation.queryparams hsl.<app>.tokenvalidation.bodyparams hsl.<app>.tokenvalidation.authentication hsl.<app>.usercontext.control
10.25.1.0.0	1.45	• No changes, republished with a new part number.

Release	Version	Changes
10.25.4.0.0	1.46	• Changes in logging (paragraph 5.2.7) • Added paragraph outlining the importance of setting up confidential applications for the individual check webservices (paragraph 3.5.1) • Updated Appendix A – BASIC authentication, generalizing the setup of roles and users for the CHECK webservices (restuserbsn, restuserbsn-role, restuser<...>, restuser<...>-role)
10.25.7.0.0	1.47	• Added additional settings in 3.4.5 Creating a datasource
10.26.1.0.0	1.48	• No changes, republished with new part number.
10.26.3.0.0	1.49	• Changed for WLS 14.1.2 and WebLogic Remote Console

RELATED DOCUMENTS

A reference in the text (**doc[x]**) is a reference to another document about a subject that is related to this document.
Below is a list of related documents:

Doc[1]	Object Authorisation within OHI Back Office (docs.oracle.com)
Doc[2]	Back Office HTTP Service Layer User Manual (docs.oracle.com)
Doc[3]	OHI Back Office JET Application Installation & Configuration Manual (docs.oracle.com)
Doc[4]	Oracle Health Insurance Security Guide (docs.oracle.com)
Doc[5]	Oracle WebLogic Remote Console Online Help
Doc[6]	Oracle Health Insurance Back Office SOAP Service Layer (SVL) Installation & Configuration Manual (docs.oracle.com)

Contents

1 Introduction

Attention: OHI Back Office releases 10.26.3.0.x to 10.26.8.0.x are certified against Fusion Middleware 12.2.1.4.0 AND 14.1.2.0.0. In WebLogic 14.1.2.0.0, the Admin Console has been removed, and is replaced by the WebLogic Remote Console. This document will assume an installation on Fusion Middleware 14.1.2.0.0. For installation of HSL on Fusion Middleware 12.2.1.4.0, see version 1.49 of this document, as delivered with OHI 10.26.1.0.0.

The OHI Back Office HTTP Service Layer is an optional component to provide so-called Use Case services.

Use Case services constitute a group of specific operations aiming to support use cases that are common for Dutch healthcare payers. Examples of typical use cases: requesting a new policy, adding an insured member, changing insured products, changing payment method etc.

OHI BO Use Case Services are implemented through the HTTP Service Layer (HSL).

The services in this service layer are based on RESTful Services technology which has the following advantages for current web application frameworks (like AngularJS and Oracle JET):

- Accessible through HTTP (for example through JavaScript).
- Supports (JavaScript friendly) JSON as input and output formats.
- Standardized interface language through using HTTP verbs (GET, POST, PUT, PATCH, DELETE)'
- Standardized set of exceptions through HTTP error codes.

This HTTP Service Layer is intended to ease integration in a Service Oriented environment.

This document describes the generic technical details regarding the HTTP Service Layer, how to install and update it and how to change configuration settings.

1.1 Licenses

Customers are required to have the appropriate license for using the HTTP Service Layer. Customers who have acquired a Connect to Back Office (C2B) license or an OHI SOAP Service Layer (SVL) license are currently permitted to install and use the web service component of the HTTP Service Layer. This is valid until further notice.

The corresponding PL/SQL services may not be used when no Connect to Back Office license, SOAP Service Layer license or HTTP Service Layer license has been obtained.

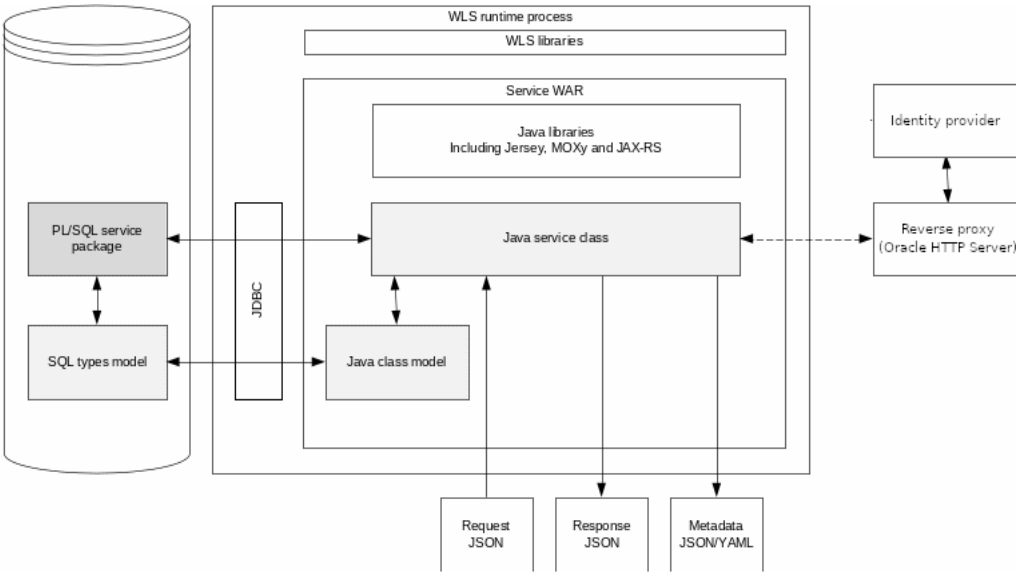
For further information please consult your OHI sales representative.

2 Architectural overview

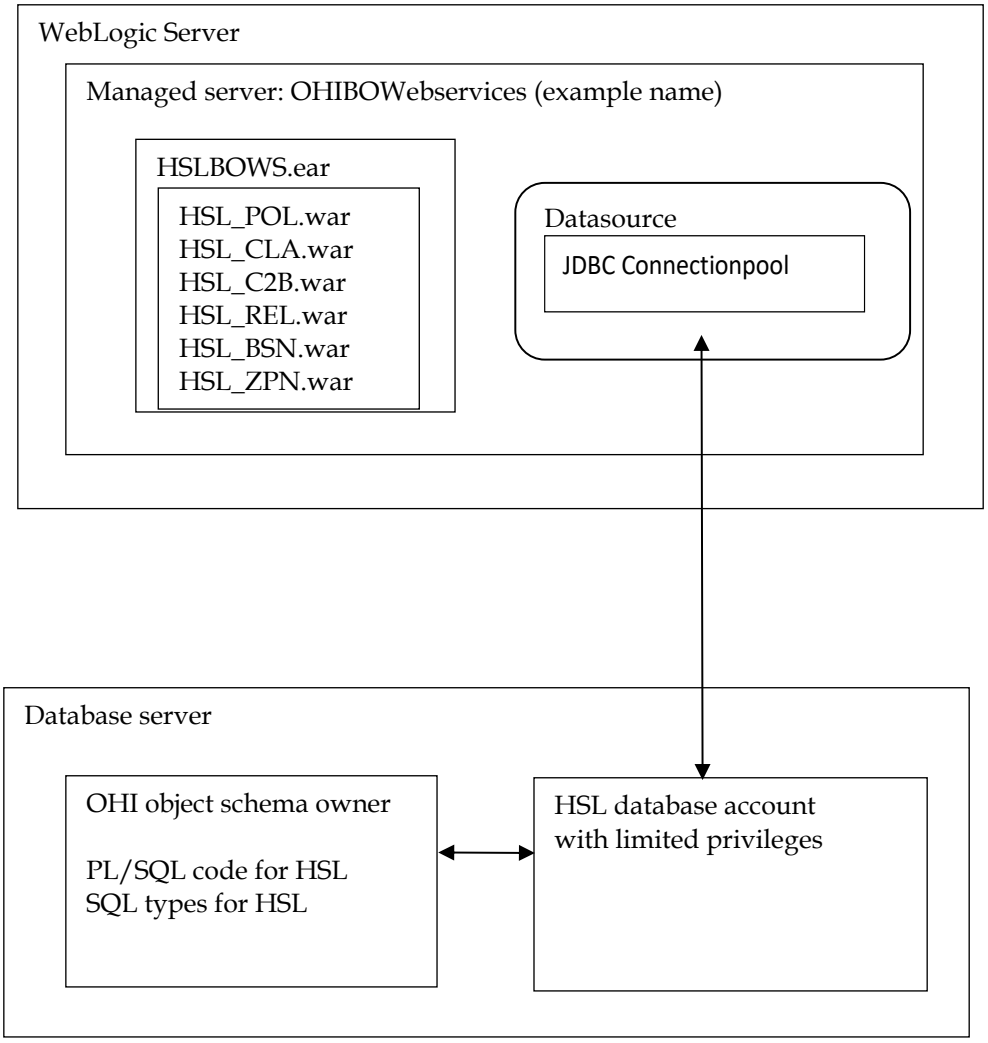
This chapter gives a high-level architectural overview of the current HTTP Service Layer implementation.

2.1 Services components

The functionality of each service is implemented through a PL/SQL service package. The service interface is provided through a Java layer.
Eclipse Jersey (JAX-RS reference implementation) and EclipseLink MOXy are used to serialize and deserialize JSON objects and for input validation.
JDBC is used to map Java objects to SQL objects and vice versa.
The PL/SQL service package performs the required operations, using operation parameters and inbound objects to communicate with the OHI Back Office database.



The high-level schema below shows how the services are deployed. It also shows the database connection to OHI which uses a database account with restricted access to execute the HSL service implementation in PL/SQL.

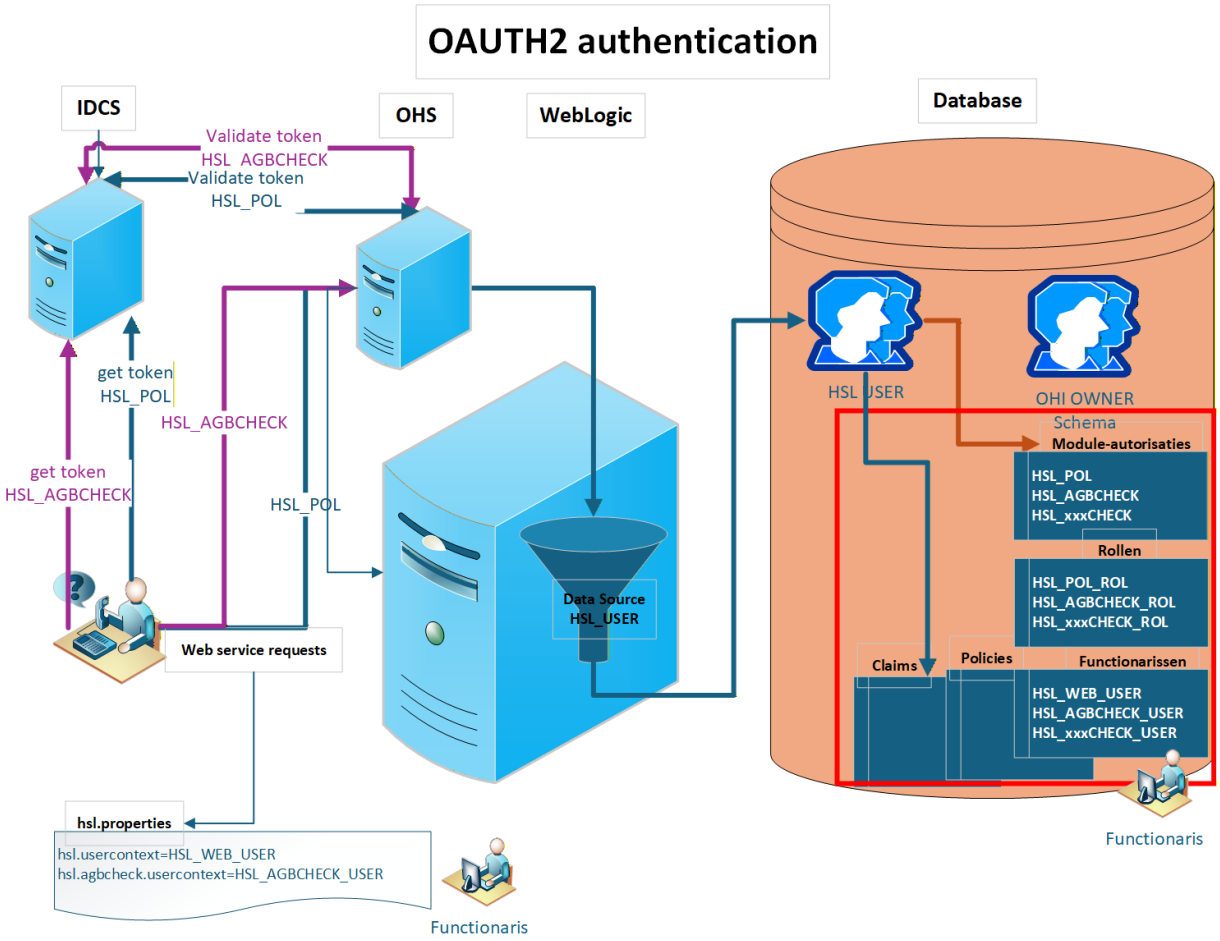


2.2 Security overview

The HSL web services are secured at various levels. The picture below shows the different mechanisms and accounts involved.

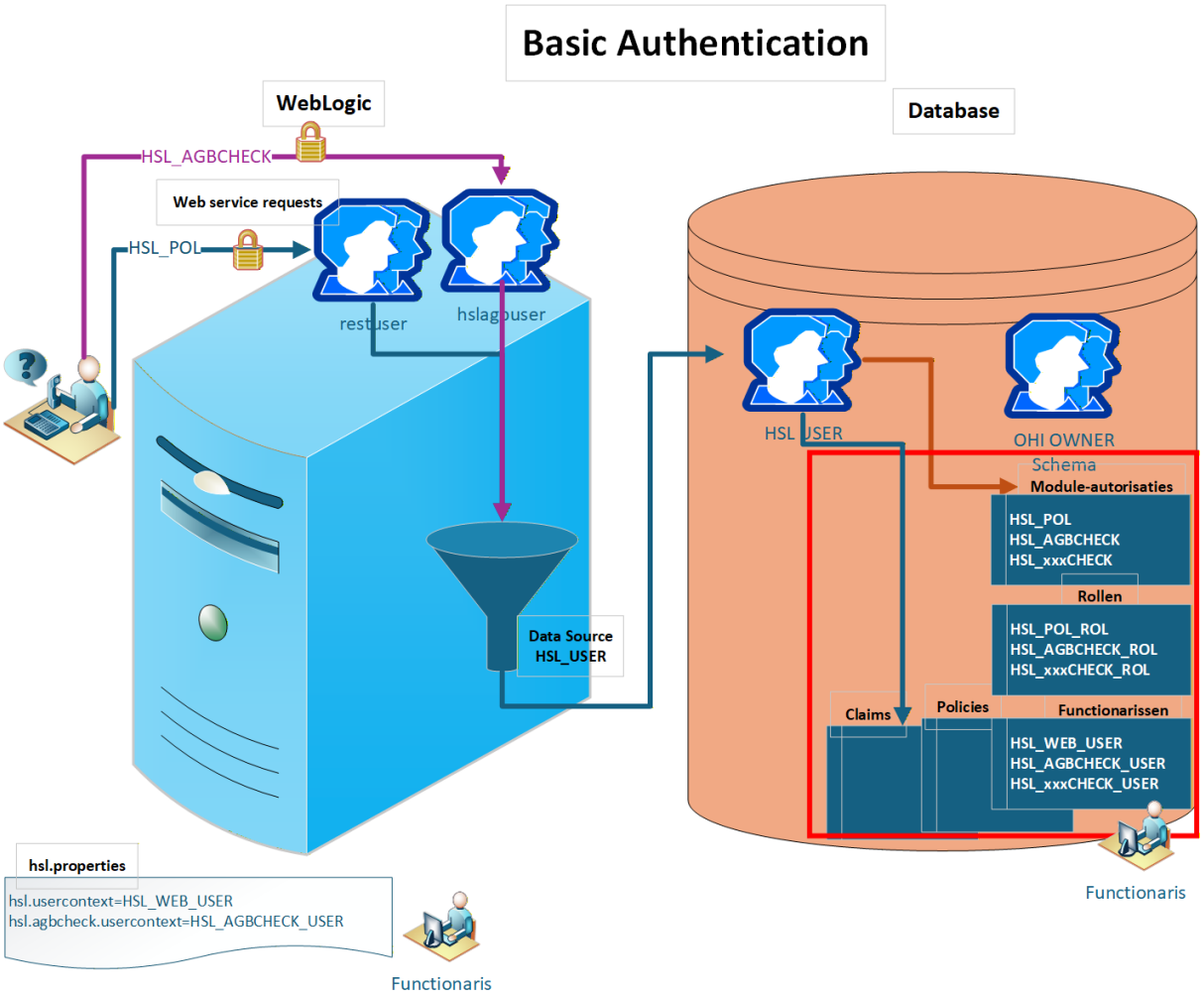
When a client application sends a web service request for a certain HSL web application, the client needs to send authentication data. This can either be OAuth2-based (see *0 OAuth2 Authorization*) or Basic Authentication (see *7 Appendix A*).

- For OAuth2, the web service request needs to be sent to a specially prepared web server. In this document, we will assume you use Oracle HTTP Server (OHS). The request must include a valid token that matches the web service that is requested (general HSL web service, AGBCHECK web services, etc.). Token validation is handled by the combination of OHS and an Identity Server. In this document, we will assume you use Oracle Identity Cloud Service (IDCS). Only if the token is valid and matches the requested web service, will the request be passed on by OHS to the WebLogic Managed Server. WebLogic does not check authentication or authorisation.



- For Basic Authentication, a WebLogic username and password are sent. These are checked by WebLogic. WebLogic also checks if the user is authorised to execute the web service, using either predefined usernames or roles (included in the HSL deployment) or checking if roles and policies linked to the web service in WebLogic have been granted to the username. In the picture, the WebLogic usernames are “restuser” and “hslagbuser”.

  So, for Basic Authentication, one or more WebLogic accounts need to be created, depending on the Security Model (see 7.1 *Background information).* Groups, Roles, and Policies may be needed too, again depending on the Security Model. For details on the WLS <…>CHECK users and roles, “functionarissen” and module authorisations, also see 7.1 *Background information*.



Once the OAuth2 Authentication or the Basic Authentication + WebLogic authorisation have been checked, the web service request is executed by the WebLogic Managed Server (Java layer of HSL). A username (or user context) is added to the request from the configuration file hsl.properties. That can either be a username that is generic for all HSL web services, or specific for a web service.

Note that this user context is neither a WebLogic user name/account, nor a database username/account. It must match a "functionaris" registered in OHI. Although "functionaris" was originally a registered end user of the OHI Back Office application, here it is a "service account" that must not have a database account. The "Functionaris"is used for 2 things in OHI:

- Authorisation of the web service (method) via module authorisations and a role
- Filling audit columns of records created or updated by the web service request (columns "created_by" and "last_modified_by").

The request (with the user context (Functionaris /user name) is passed on to the database via a Data Source. That data source uses a database account with just enough privileges to execute the HSL PL/SQL functionality in the database. In the picture, this is "HSL_USER" The PL/SQL functionality itself – and all data – is owned by the OHI Schema Owner.

The first thing the database session does is check if the user context (the "functionaris") has been authorized to execute the web service that is requested. The

authorization is checked against data in OHI tables using OHI module authorisation and role data.

Note that this authorization check is NOT:

- based on standard database authorization (username, grants, roles, etc. in the database dictionary)
- related to the privileges of the database account in the data source
- related to the OAUTH2 claim
- related to the Basic Authentication user, roles, groups and policies

Only if the OHI-specific authorisation check is passed will the PL/SQL functionality process the request and return a result to the Java layer, which will then return the result to the client application.

3 Installation of HSL services

This chapter describes the steps to (re)install the HSL web services.

This chapter contains the following parts to separate the various work areas:

- Sizing/load aspects
- Database installation
- WLS preparation
- Identity Provider configuration
- Oracle HTTP Server (OHS) preparation
- Setting up an OAuth2.0 resource server through the use of the mod_auth_openidc module for OHS.
- Security configuration
- (Re)deployment of the HSL application
- Additional Security aspects

3.1 Terminology

Note the following use of terminology:

- HSL stands for HTTP Service Layer. The underlying technology is based on RESTful service technology.
- A HSL service has one or more service operations.
- Each HSL service resides in its own HSL application (which is packaged as a WAR file).
- The HSL applications are packaged together as an EAR file, which is deployed to the WebLogic application server.

3.2 Sizing/load aspects

From the "Introduction" and the "Architectural overview" chapters it should be clear that the HSL services are functionally implemented through PL/SQL in the database.

The Java layer providing the REST interface handles request and response messages. It validates an incoming request, calls the PL/SQL service implementation to perform the required operation and transforms the result into a response message.

This choice means that the larger part of the processing is carried out on the database server and only a small part is handled on the application server.
Since the architecture for HSL is similar to the SVL services, the distribution of loads on the application and database server is expected to be comparable.

Tests with 10.24.x.x.x releases showed that for relatively small request and response messages and simple HSL service operations the related application and database CPU load are in the same order of magnitude. But when the size of a request and/or response message grows, the time needed for the related deserialization and serialization increases linearly with the size of the message, which may result in considerably more CPU load on the application server than on the database server. Conversely, service operations that demand a lot a of processing on the database server, such as registering or changing a policy and immediately, synchronously, processing all possible related acceptance checks, will generate more CPU load on the database server.

Based on experience, most of the simpler service operations on a well-sized and well-performing production environment should not take more than 0.2 up to 0.5 second

in total elapsed time when measured on the WebLogic Server. Of this elapsed time a comparable amount of time should be spent by the application server and the and database server handling the call, as mentioned before.

More complicated calls and service calls that return large data sets may take more time but usually should not exceed response times of more than a few seconds. As an example, a typical premium calculation call should be executed within a second and a fetching a large set of claim lines (several hundreds) should usually be returned within 5 to 10 seconds.

3.2.1 Deployment choices

The overall load on the OHI application resulting from HSL service calls is customer specific and may change over time.

HSL services are likely to be used by customer-facing applications. Although it may technically be possible to deploy HSL services to the same application server running the Forms GUI for internal users, you should be aware of the peak loads from HSL services during commercial campaigns. These loads may well exceed your normal capacity. You should devise your own strategy to cope with these extra loads. This strategy may include using separate application servers for internet users, potentially using a separate database with cached data for information requests, throttling inbound requests, etc.

If you choose to install HSL services on the application server for the Forms GUI it is advisable to actively monitor the respective loads of Forms processes, SVL processes, HSL processes and PSL processes (for the OHIJET application). This allows you to pick up trends to help you refine your infrastructure strategy.

Especially if you have multiple applications using the same HSL services, it may help to use a service bus to create a "separation of concerns". The service bus allows you to map the HSL interface specification to a customer-specific interface which means less maintenance on the client applications when deploying a new version of a HSL service. As long as the mapping on the service bus can be synchronized with the HSL service interface, the code of client applications can remain the same.

Stringent requirements for high availability and failover are also reasons to consider a service bus, combined with a load balancer, as a go-between.

3.3 Database installation

All database components of the HTTP Service Layer are owned by the OHI Back Office schema and are installed through the OHI Back Office release installation procedure.

To use the database components of the HTTP service layer, one or more database accounts must be created with HTTP Service Layer access privileges.

Before creating the account(s), check if you are licensed to use the HTTP Service Layer.

We expect that you are familiar with the DBA tasks of an Oracle database administrator in the instructions that follow.

Please check if you have a database object (package) HSL_UTIL_PCK in the OHI Back Office schema. If not, something went wrong regarding the installation of the HTTP Service Layer code. If this is incorrect, please contact the OHI Support department.

If the package is present in your database, you can continue with the database part of the installation.

3.3.1 Creating a HSL database account

The OHI Back Office schema owns the PL/SQL code to implement the HTTP Service Layer, but this account must never be directly used to execute the services.

The use of a separate database account to access the HTTP Service Layer components reduces the risk of accessing unauthorized OHI data and makes that account accountable for HSL actions. The HSL account(s) only need a minimum set of object privileges to the HSL database objects.

One or more HSL database accounts can be created:

- Default HSL account for use with WebLogic application server
  This account is configured in the HSL properties file as the default account for HSL service requests.
- Additional HSL account for use with bespoke PL/SQL code development by the customer. Please follow the directions in Doc[1].

The following steps are needed to setup a HTTP Service Layer database account:

1. Create a schema owner, for example HSL_USER. Determine the password policy, temporary tablespace, etc. according to your company standards but beware there is no interactive login which might show expiration messages for the password due to the enforced password policy. Do NOT use the same account as is used for the SVL services.
2. Grant *create session* system privilege to this account.
3. Grant the HTTP Service Layer object privileges: logon as the OHI Back Office schema owner, enable server output, and run

```
alg_security_pck.HSL_grants
(pi_owner   => '<ohibo_owner>'
,pi_grantee => '<hsl_user_account>'
)
```

Example:

```
execute
alg_security_pck.HSL_grants
(pi_owner   => 'OZG_OWNER'
,pi_grantee => 'HSL_USER');
```

This command does not have to be repeated after each new deployment of a new .ear file. During the database installation of OHI patches any existing grantees of the HSLL objects receive any required additional grants. However, if you run into ORA-01403 errors during a web service execution your first check should be to run this command in SQL*Plus, enabling server output before running, and see whether missing grant privileges were granted.

3.4 WLS Preparation

When the database account has been created and granted successfully, a WebLogic Server environment (software home) must be prepared for deploying the HSL application.

We expect that you are familiar with the WebLogic concepts like 'Domain', 'Managed Server', 'Cluster', etc.

These are your options:

- Use the same WebLogic environment (ORACLE_HOME) which is used for servicing the OHI Back Office user interface and batches. In this case you are required to a new WebLogic domain (with a new Admin Server) for the HSL applications, in order to prevent interference with the GUI application.

- Deploy the HSL application in a separate WebLogic environment (possibly on a separate server). This allows you to separately upgrade or patch the different WebLogic environments or implement a workload distribution.

  Deploy HSL applications to multiple environments for better scalability. Be sure to deploy each HSL application only once in a Managed Server or a cluster of Managed Servers.

- For testing purposes, you may want to have multiple versions within the same domain. In that case you should have a separate Managed Server for each deployment.

Some remarks about installing in a separate WebLogic environment:

- The OHI Back Office GUI application (Forms) installation requires a WebLogic Server "Infrastructure" installation. That means the domain created for Forms needs to have its own database schemas with OPSS and Audit database tables (created by RCU, the Repository Configuration Utility). For the HTTP Service Layer domain these schemas are not required, provided you do not select more components during the domain configuration than described.
- When installing in a separate WebLogic Server environment, use a different Installer: use the "Generic" installer instead of the "FMW Infrastructure" installer. When installing in a separate WebLogic environment make sure the correct components are installed when creating the Domain. You need at least:
  - `WebLogic Advanced Web Services for JAX-WS Extension - 14.1.2.0.0 [oracle_common]`
  - `WebLogic JAX-WS SOAP/JMS Extension - 14.1.2.0.0 [oracle_common]`

  When you have not installed these components your web services will respond with 'There are error messages.' All info in the functionalFaultType will contain question marks (???).

The instructions in the following paragraphs cover the setup of a new domain including the setting up of Managed Servers, a machine definition, data sources, etc.

This will support the following scenarios:

- ✓ Creating a separate domain with a single Managed Server
- ✓ Creating a separate domain with a cluster of 2 Managed Servers
- ✓ Adding a Managed Server to an existing domain

3.4.1 Using setUserOverrides.sh

To pass Server Start arguments to the WebLogic servers (like memory settings for the JVM, debug options and other Java options), create a file `$DOMAIN_HOME/bin/setUserOverrides.sh`

See the following documents on My Oracle Support for details:

KB581904	How to Customize Environment Parameters using 'setUserOverrides.sh' File
KB364001	How to Pass Unique Start Arguments to Different Managed Servers Using setUserOverrides.sh Script in WebLogic 12c

3.4.2 Requirements

The following requirements/limitations must be taken into account:

- ✓ A certified WebLogic Server version including JAX-WS (SOAP/JMS) extensions. The HSL services must be deployed on a single Managed Server or a cluster of Managed Servers (the 'target').
- ✓ The HSL services may not be deployed on a Managed Server which is also used for hosting the OHI GUI application (Forms). The Managed Server may not belong to a cluster used for deploying the GUI application.
- ✓ One deployment can only service one single OHI Back Office environment (it connects to a specific connection pool which accesses a specific OHI Back Office 'instance').

If the HSL application must be deployed more than once (for servicing different OHI Back Office environments) each deployment should be on its own Managed Server or Cluster.

HSL can be deployed on the same Managed Servers as SVL and PSL plus OHIJET.

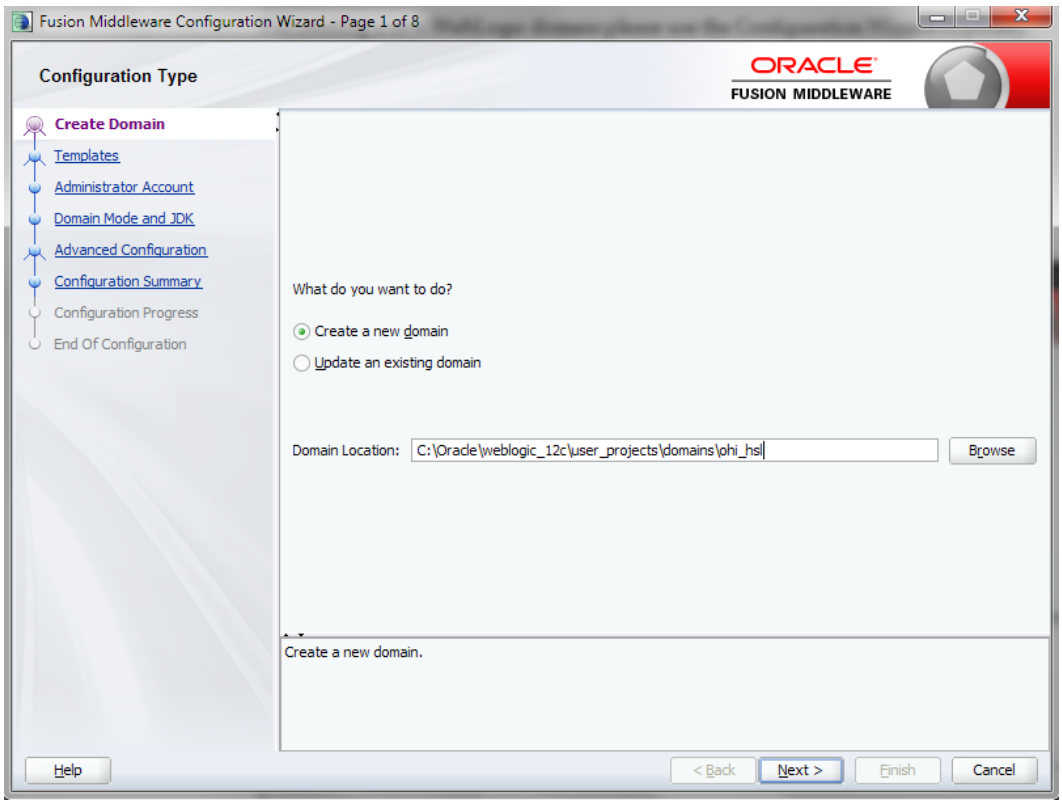
3.4.3 Creating a domain

Before creating a Domain, be sure to understand the difference between a "FMW Infrastructure" and a "Generic" WebLogic installation, and the consequences. Make sure the environment variable DOMAIN_HOME is not set.

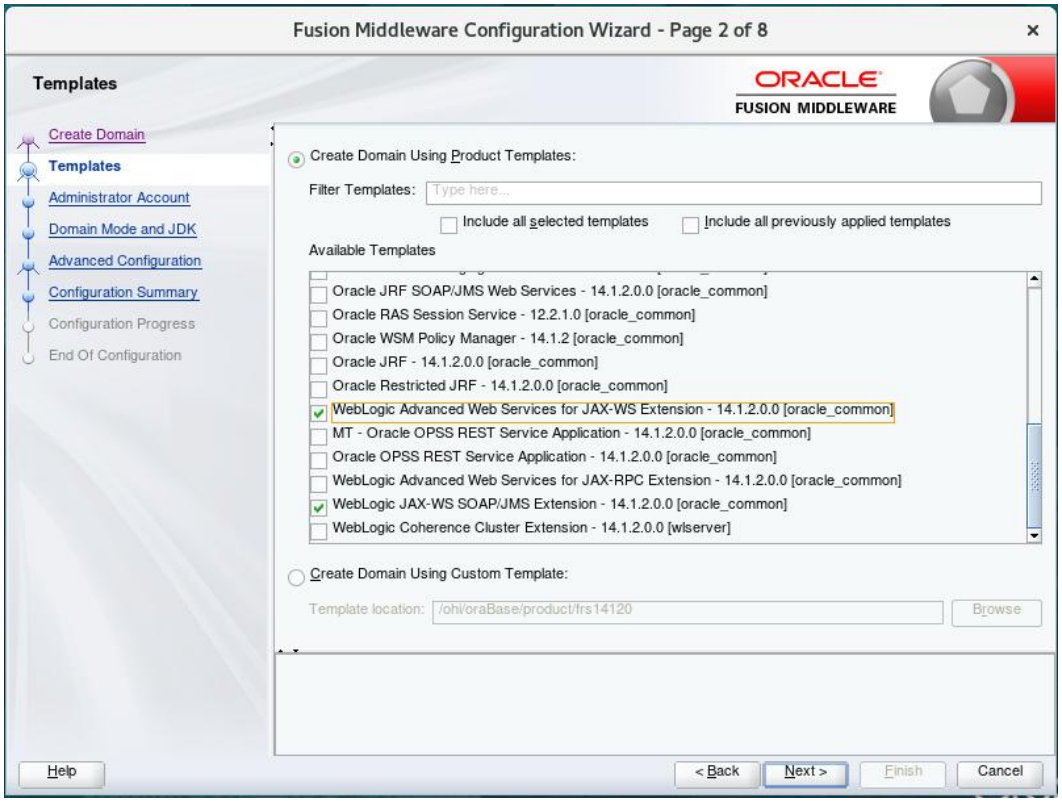
If you create the new WebLogic Domain from the same software home as the Forms Domain, you must choose the same "Domain Mode" (Development or Production), to avoid errors during startup of the new Managed Server(s).

For creating a new WebLogic domain, please use the Configuration Wizard (typically in the common/bin folder of the WebLogic Server home, so for example `$MW_HOME/oracle_common/common/bin/config.sh`) or use your own scripting.

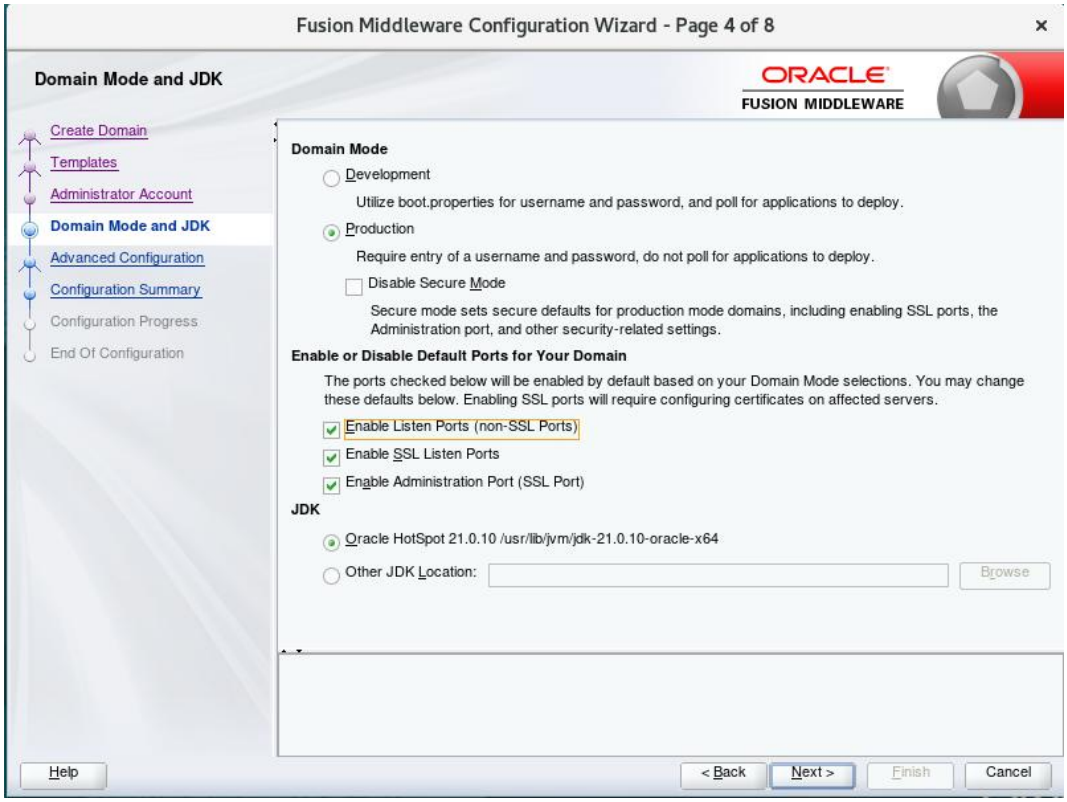
Specify the domain location. This is inside the WebLogic Home by default, but you can specify a location outside the WebLogic Home. The last part of the location will be the Domain Name.



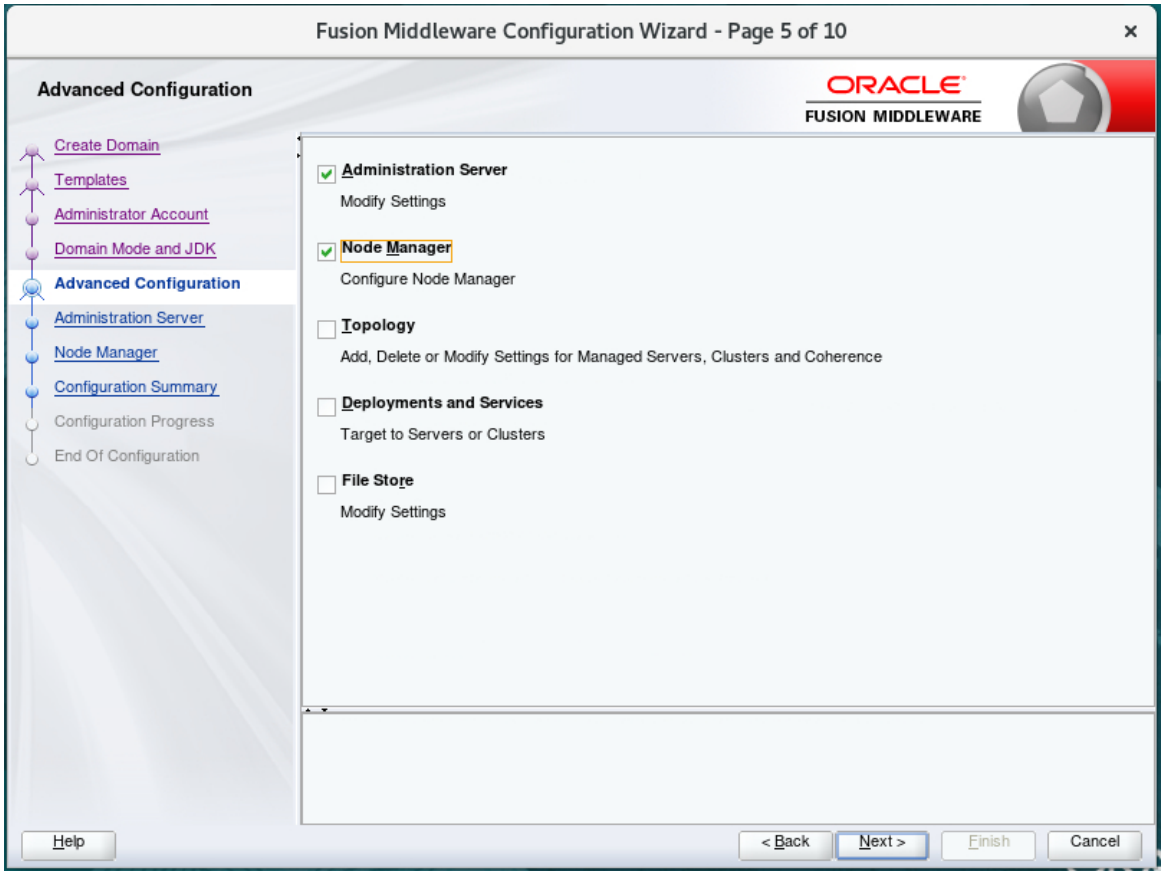
When creating a new domain select at least the options as shown below.



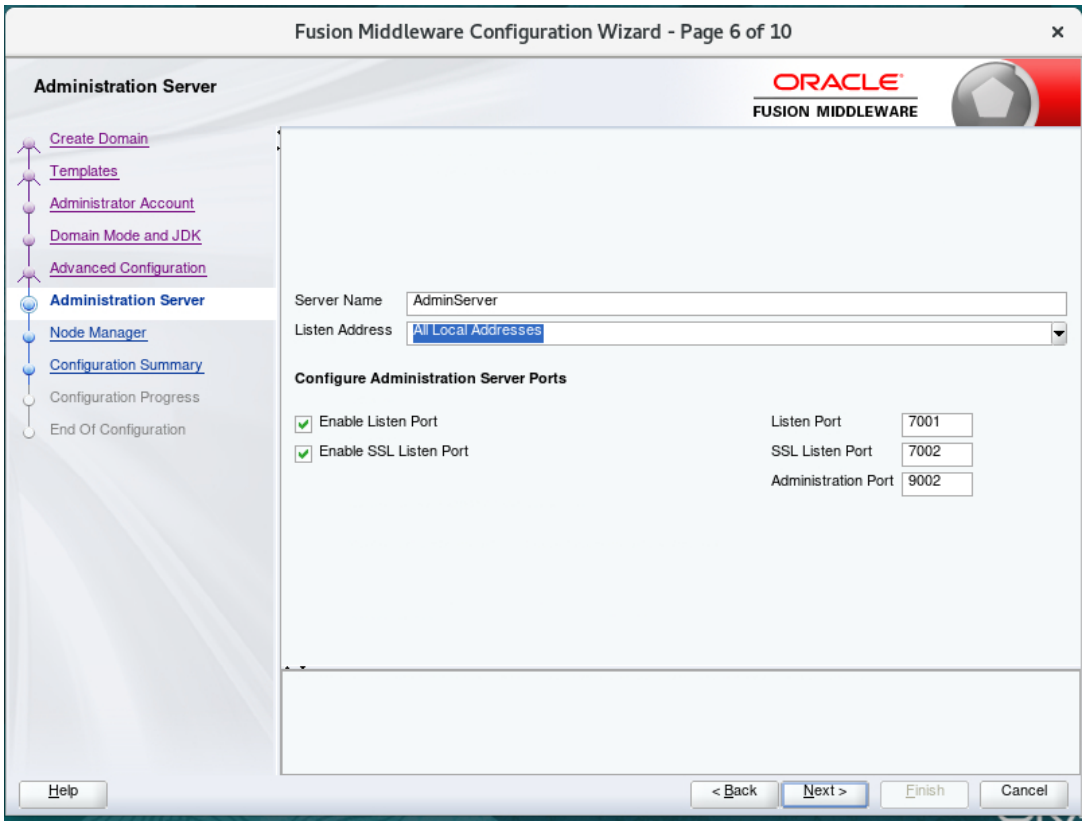
In the next screens, specify the *username* and *password* for the domain administrator account. When prompted for developer or production mode choose *production mode* and pick a JDK.



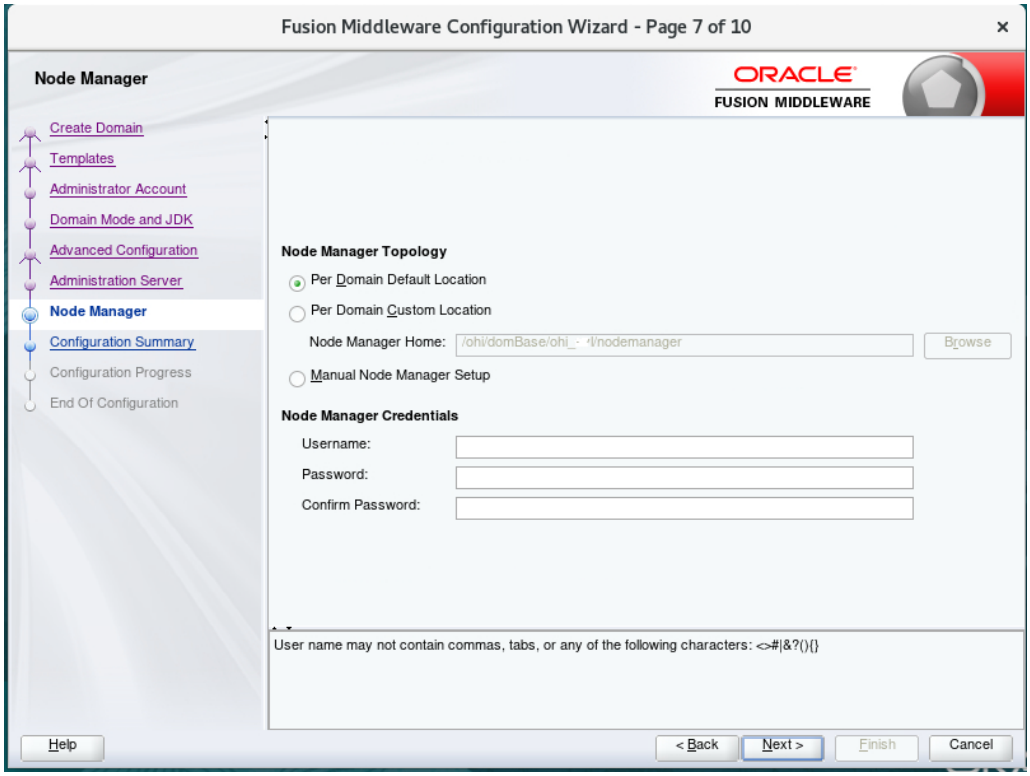
In this documentation we choose to configure only the Administration Server and the Node Manager using the wizard. The Administration Server can be used as the starting point for additional configuration options you may want to choose later:

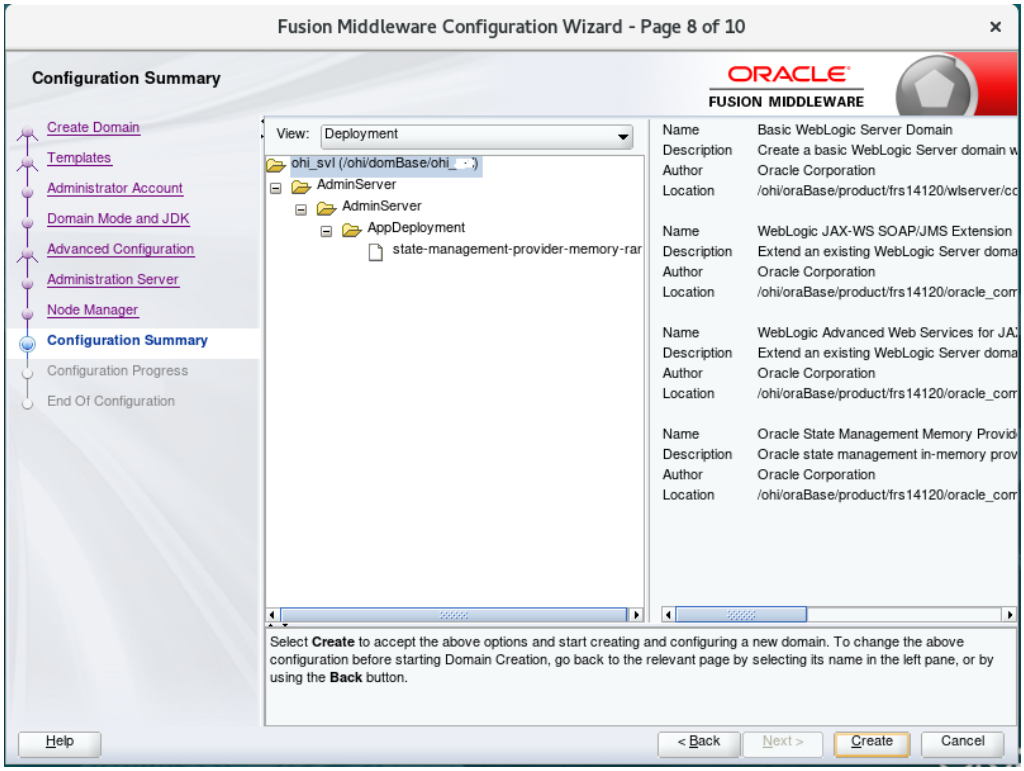


For the Administration Server free port numbers must be specified. Enable SSL to support secure connections. An example using non default ports is shown below.



In the next screen, enter a username and password for the Node Manager:





Finish the domain creation by the Configuration Wizard.

3.4.4 Creating Managed Server(s)

The other WebLogic objects can be created with the wizard too. Here, we use the Administration Server to create one or more Machines, a Cluster (optionally) and one or more Managed Servers.

If you start a Node Manager, you can use the WebLogic Remote Console to start the Managed Servers later on. You need to associate the machine with the Node Manager

so that the Node Manager can start the Managed Server within the domain of the machine definition.

Start the Administration Server (of the existing or newly created domain) using the startWebLogic.sh script (this is present in the root folder of the domain folder, which you created through the Configuration Wizard).

When the Administration Server is running, either:

- deploy the Hosted WebLogic Remote Console according to paragraph 4.9. *DEPLOYING THE HOSTED WEBLOGIC REMOTE CONSOLE* in the "Oracle Health Insurance Back Office Installation, Configuration and DBA Manual"

  or

- download and install the local WebLogic Remote Console according to paragraph 4.1.2 in the "Oracle Health Insurance Back Office Installation, Configuration and DBA Manual".

- Log in in the Hosted WebLogic Remote Console.
- Connect to the newly created Admin Server.
- Select the "Edit Tree".
- Select "Environment".
- At this point you should decide whether or not to make the Managed Servers part of a Cluster. If no Cluster exists, you can create one first, using the "Clusters" option.
- Select "Servers"
- Click on "New"
- Enter a name for the new Managed Server. For easy reference you may want to include the domain name and the OHI environment name in the name of the Managed Server, for example "MS_HSL1420_OHIPRD".
- Click on "Create".
- In the next screen, in the tab "General":
    - Click on "Show Advanced Fields"
    - Select a Cluster or create one using the dots behind the field "Cluster" (if desired)
    - Create a Machine using the dots behind the field "Machine"
        - Enter a name
        - Select Type = "Unix Machine"

      and select the new Machine in the main screen.
    - In the "Listen Address" field, fill in the Fully Qualified Domain Name (FQDN) of the application server. Using "localhost" or an IP number may result in errors in SSL/TLS communication.
    - (At least initially, until SSL has been configured and tested) keep the "Listen Port Enabled" and specify a "Listen Port".
    - Set the "SSL Listen Port Enabled" and specify a "SSL Listen Port".
    - Set the "WebLogic Plug-In Enabled" on

  Make sure the listen address is the actual listen address that is used by the Node Manager. This is passed as first parameter to the `$WL_HOME/server/bin/startNodeManager.sh` shell script. The correct value can be found as ListenAddress in the file nodemanager.properties.

This address can be changed in the file nodemanager.properties which is located in the <domain home>/nodemanager folder. This is necessary when you have a node manager per domain.

- Optionally, change the settings in the tab "Logging".

You need to create a boot.properties file for the new Managed Server for the domain in the domain home Managed Server ../data/nodemanager.

Because you are running in Production Mode, you need to create the file yourself, in the $DOMAIN_HOME/servers/AdminServer/security folder. This file is used when the AdminServer is started by the script startWebLogic.sh. If the file is not present, the script prompts for the username/password. The same goes for the Managed Servers when you start them through a script.

3.4.5 Creating a data source

The HSL application needs a data source to connect with the OHI Back Office database.

To create a data source in the Weblogic Remote Console:

- Connect to the newly created Admin Server.
- Select the "Edit Tree".
- Select "Services"
- Select "Data Sources"
- Click on "New"
- Enter a name for the data source that reflects its purpose. For example, you may want to reference the database name: DS_OHI_HSL_PRD.
- Enter a JNDI name. The JNDI name will be used in the properties file for starting the HSL application.
- Select a Managed Server as Target, e.g., MS_HSL1420_OHIPRD
- Select "Data Source Type" = "Generic Data Source" for single instance databases. If the data source connects to a RAC data source it is more useful to choose "GridLink Data Source", as this can respond to instance state changes.
- The following fields appear for the different types of data source:

	Generic Data Source	GridLink Data Source
Database Driver	Oracle's Driver (Thin) for Service connections; Versions: Any. Do not use the XA driver.	Oracle's Driver (Thin) for GridLink Connections, Versions: Any
Global Transactions Protocol	N.A.	One-Phase Commit
Listeners	N.A.	The SCAN address of the RAC database listener
Service Name	N.A.	The Service Name of the RAC database
Protocol	N.A.	TCP
FAN Enabled	N.A.	<empty>

ONS Nodes	N.A.	<empty>
ONS Wallet File	N.A.	<empty>
ONS Wallet Password	N.A.	<empty>
Database Type	Oracle	N.A.
Database Name	The (service) name of the database	N.A.
Host Name	The name of the database server	N.A.
Port	1521	N.A.

- For "Database User Name", enter the user name of the Service Layer database account you created in paragraph 3.3 *Database installation* (e.g. HSL_USER)
- For "Password", enter the password of the Service Layer database account.
- Click on "Create".

The next page shows the results. You can test the connection using the "Test Configuration" icon.

- Open the "Connection Pool" tab, and then the sub-tab "General".
- Consider setting the "Initial Capacity" to 0. During the setup of new connections, a health check, if you have configured this (for more information see the OHI Release Installation manual), claims a shared lock that might stall (patch) sessions and vice versa. Setting this option to zero implies no connections are set up after the connection pool is initialized but only on demand.

- Open the "Connection Pool" tab, and then the sub-tab "Advanced".
- Ensure that the option 'Wrap Data Types' is off. This setting is needed for passing CLOB objects to and from the database. If it is checked, it slows down execution.
- Consider activating the "Statement Timeout"'. This is the time after which a statement currently being executed will time out. This can be used to limit the impact of run-away queries (e.g., if a bad execution plan is chosen or a wildcard search is not selective). Especially in scenarios where the requestor (e.g., OSB) stops listening after a certain period – and possibly retries the same operation several times – continued execution of these long running queries can overload the database. This can have serious effects on the performance of other web services. In those cases, it is better to cancel the query (after a period that is a little longer than the timeout of the requestor).

  Queries that are cancelled will result in a technical fault:

  ```
  ORA-01013: user requested cancel of current operation
  ```
  or
  ```
  ORA-03111: break received on communication channel
  ```

  Note that it is possible to specify different data sources for different web services (see 5.2 *properties file for the HSL web services*). This can be used to specify different timeouts for different web services. You could create a data source with a long timeout for web services that usually take longer and one for web services that are normally quick.

  Be aware that the 'Statement Timeout' is only about database processing. Any processing of output by the Middle Tier is not included in this period.

- When “Statement Timeout” (see below) is set an execution of a statement can be interrupted. In rare cases, this could lead to the following error in subsequent calls:

  “The current transaction is not allowed, because the current PL/SQL definition is defined as 'Query only'!'”

  We recommend setting the following properties only if you have encountered this error.

- To prevent this error for subsequent calls, set the following properties in the advanced settings of the Connection Pool tab:
  - Test Connection On Reserve: On
  - Test Frequency: 0 seconds
  - Test Table Name: SQL begin alg_public_pck.session_health_check(); end;
  - Seconds to Trust an Idle Pool Connection: 0 seconds

- Click on “Save”
- You may have to restart your Managed Server to activate this setting.

3.4.6 Enable SSL

The HSL services are preconfigured to use a default policy which uses SSL. Therefore, you need to enable SSL for every Managed Server to which you deploy the HSL application.

Check if SSL was enabled when the Managed Server was created.

- In the “Edit Tree”, go to the Managed Server configuration.
- Open the “General” tab.
- Set “SSL Listen Port Enabled” = On
- Set “SSL Listen Port” to a value that is free.

3.4.7 Setting up a key store

For testing purposes, you may want to use the Demo keystore that is provided.

Note that in a production environment it is not safe to use the demo keystore.

You should create your own Custom Identity Key Store and import your certificates.

To register your Custom Identity Key Store:

- In the “Edit Tree”, go to the Managed Server configuration.
- Open the “Security” tab.

For more information about configuring keystores please read the WebLogic documentation. As a starter you can use this: Configuring Keystores

It contains references to pages that describe in more detail how to obtain private keys, digital certificates, etc.

You should take action and not rely on the demo keystore!

3.4.8 Configure Managed Server logging level

The standard logging level for a Managed Server regarding security issues is intentionally non-informative to discourage fraudulent users.

A typical security-related error message looks like:

'Unknown exception, internal system processing error.'

If you are trying to setup the SVL application to work with SSL and basic authentication in a non-production environment you can configure verbose logging with the following Server Start argument for the Managed Server:

```
-Dweblogic.wsee.security.debug=true
```

Create or edit a file $DOMAIN_HOME/bin/setUserOverrides.sh and add the following text:

```
#!/bin/bash
echo Adding Settings from UserOverrides.sh

# global settings (for all servers)
# this will decrease start up times
JAVA_OPTIONS="-Djava.security.egd="file:/dev/./urandom" ${JAVA_OPTIONS}"
export JAVA_OPTIONS
CONFIG_JVM_ARGS="-Djava.security.egd=file:/dev/./urandom ${CONFIG_JVM_ARGS}"
export CONFIG_JVM_ARGS

# specify additional java command line options for the Admin Server
#if [ "${SERVER_NAME}" = "${AS_NAME}" ]
#then
#
#fi
#export JAVA_OPTIONS

# specify additional java command line options for specific servers
if [ "${SERVER_NAME}" = "MS_HSL1420_OHIPRD" ]
then
  # add settings for HSL
  # Custom Setting for MS_HSL1420_OHIPRD to set debug level for SSL
  JAVA_OPTIONS="-Dweblogic.wsee.security.debug="true" ${JAVA_OPTIONS}"
fi
export JAVA_OPTIONS
```

Replace the server name MS_HSL1420_OHIPRD with your server name.

When startup times of your service calls are important and the security of the connection is less important you may consider to specify an alternative for retrieving cryptographically strong random numbers (included above):

```
JAVA_OPTIONS="-Djava.security.egd="file:/dev/./urandom" ${JAVA_OPTIONS}"
```

Restart the Managed Server to get the new verbose messages later on. After you have deployed the services and are testing them (through for example SoapUI as described later), you might get a message like this in the Response message:

weblogic.xml.crypto.wss.WSSecurityException: Timestamp validation failed.

This would indicate that you forgot to add a timestamp when calling the HSL application.

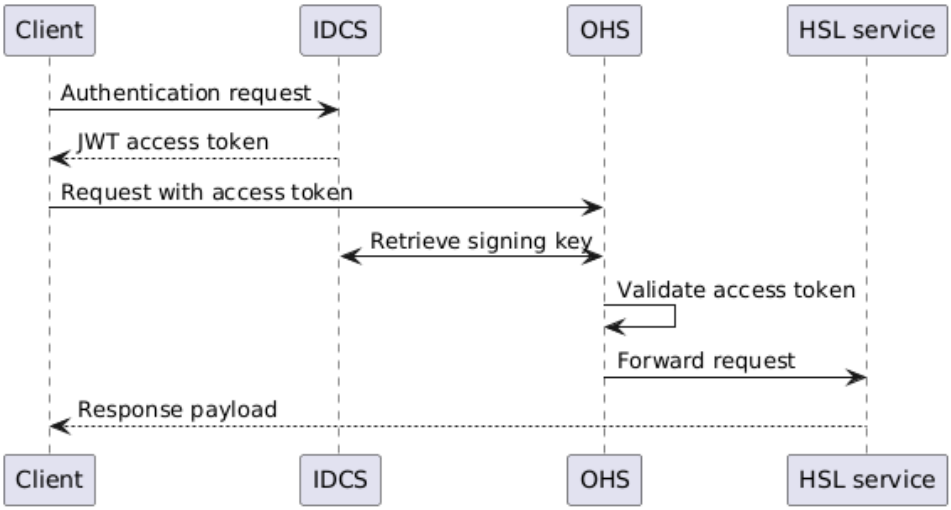
3.5 OAuth2.0 Authorization and OpenID Connect Authentication

As part of OHI release 10.24.8.0.0 a new way of securing the HSL webservices has been introduced. This setup is based on the OAuth2.0 authorization framework. As an alternative the services can be secured using Basic authentication, details for which can be found in Appendix A. The setup described in this chapter is the preferred option for securing the HSL services.

The setup uses a commonly used open-source module for the Apache HTTP server. This module can also be used for the Oracle HTTP server (OHS). The module allows OHS to act as an OAuth2.0 Resource Server, protecting access to the HSL services.

After client applications authenticate against an Authorization Server (which for the purposes of this manual will be Oracle Identity Cloud Service, in short IDCS, but could also be Microsoft Entra ID, formerly known as Azure AD) they will receive an access token. The Resource Server validates the access tokens the client received from the Identity Provider, sets headers and forwards the request to the HSL services, or denies access based on the validation result.

After setting up all the required components you will have a setup that allows client applications to authenticate themselves to IDCS, receiving an access token in the form of a JSON Web Token (JWT) as a part of this interaction. The JWT can then be added as part of the Authorization HTTP header to requests to the OAuth2.0 Resource Server (OHS in this manual). The resource server provides the only means of accessing the HSL services, acting as an intermediary (also known as a reverse proxy) that forwards requests to the HSL services upon successful validation of the access token. Schematically, a successful first interaction will look like this:



The description of the required configuration will start with the client and will continue inward, toward the HSL services. Along the way the components themselves, and the interactions between them will be discussed in more detail.

3.5.1 Confidential applications for services with stringent security requirements

OHI Web Services require a non-guessable UUID as an identifier of the object the caller wants information about.

To start the conversation, the caller therefore needs to get the UUID of the object. For certain types of objects, it is not sufficient to supply a single guessable identifier, as that is deemed insecure.

For example, relations (persons) can be identified by their 'BurgerServiceNummer' (BSN)(Social Security number), but these BSNs come from a publicly known range, so a malicious caller could simply do repeated calls with random numbers from the BSN range, get the UUIDs for each and use those to retrieve a lot of confidential information about the persons.

To prevent this abuse, two security measures have been implemented:

- To start the conversation with the regular OHI Web Services, a second data element is required in addition to the first, guessable identifier. For persons, the date of birth must be supplied in addition to the BSN. Only if these match will the regular OHI Web Service return the UUID of the person.
- The second data element can only be retrieved from a special check service that requires a separate authentication. For persons, this check service will return the date of birth for the supplied BSN.

It is important that these check operations are authenticated separately from the regular webservice operations, to implement this additional line of defense.

Each individual check service should have an individual authentication, to make sure a breach of authentication of one check service does not compromise the other check services.

When using OpenID Connect specification (OIDC) for authentication, it is of vital importance to set up so-called 'confidential applications' with unique private keys to support separate authentication for the individual check services in your Identity Cloud Service. See paragraph 3.5.3.

For more information refer to Doc[2] paragraph 3.4.

When using Basic authentication this separate authentication can be realized through separate sets of credentials as outlined in Appendix A. A future version of this document will provide steps outlining the setup of confidential applications for services with more stringent security requirements.

3.5.2 Client configuration

Before a client may request access to a protected resource like the HSL services it must first authenticate itself. OAuth2.0 only describes authorization and leaves authentication of clients to be implemented by other specifications. The OpenID Connect specification (OIDC) builds on the OAuth2.0 framework standardizing mechanisms for clients to authenticate against Authorization Servers. What follows is a description of how a client application authenticates itself to IDCS using the Private Key JWT method of OIDC.

Public-key cryptography is used as a means for the client to prove to the Authorization Server it is communicating with a known client. This works by having the client application sign a JSON Web Token (JWT) with its private key. The JWT serves as an intermediate credential that can be sent to an Authorization Server along with a request for a specific scope of access to a protected resource. The protected resource here being an HSL service. The received JWT (called the authorization grant in OAuth2.0 parlance) is verified by IDCS by checking the signature with the public key counterpart that has been uploaded in advance as part of the IDCS setup (described in the next chapter). Note that values between '<>' are placeholder values and need to be replaced with values obtained from IDCS.

The header of the JWT consists of the following:

```
{
   "alg": "RS256",
   "typ": "JWT",
   "kid": "<KEY ID>"
}
```

The body of the JWT consists of a number of claims:

```
{
   "iss": "<CLIENT_ID>",
   "sub": "<CLIENT_ID>",
   "aud": "https://identity.oraclecloud.com/",
   "jti": "<UUIDv4>",
   "iat": "<Unix time>",
   "exp": "<Unix time + 5 minutes>"
}
```

Explanation:

- alg: algorithm used to encrypt the JWT
- typ: Token type.

- kid: Key ID. The alias given to the uploaded public key (configured in the next chapter).
- iss: Issuer. As the client creates the authorization grant, the client itself is the issuer.
- sub: Subject. Refers to the client the grant is used to request an access token for.
- aud: Audience. Identifies the Authorization Server as an intended audience.
- jti: JWT ID. Unique identifier for the token.
- iat: Issued At. Time at which the JWT is issued.
- exp: Expiration time after which the JWT must not be accepted for processing

To create a validly signed JWT it is recommended to use one of the readily available libraries that offer such functionality. See https://jwt.io/libraries for several options in a variety of programming languages.

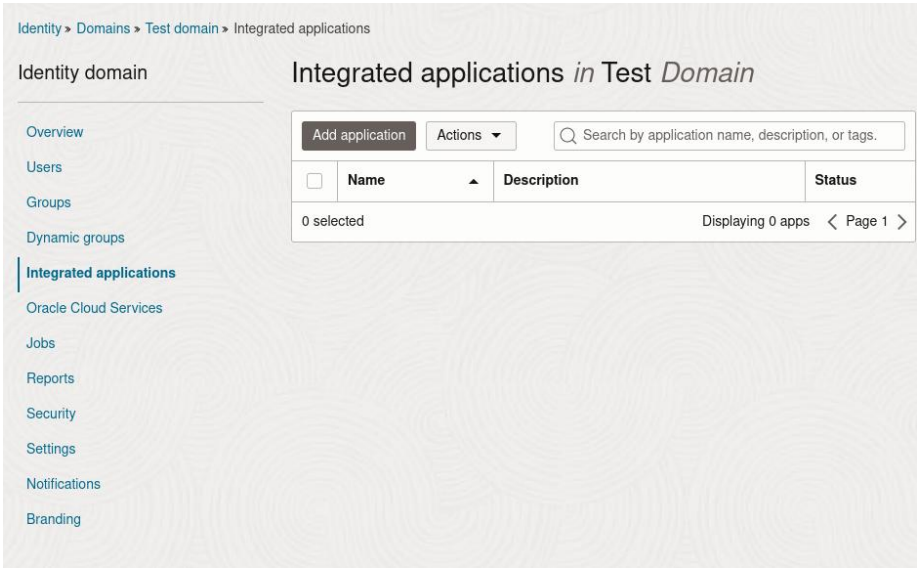
Before the signed JWT can be used, IDCS needs to be configured to receive it.

3.5.3 IDCS – confidential application setup

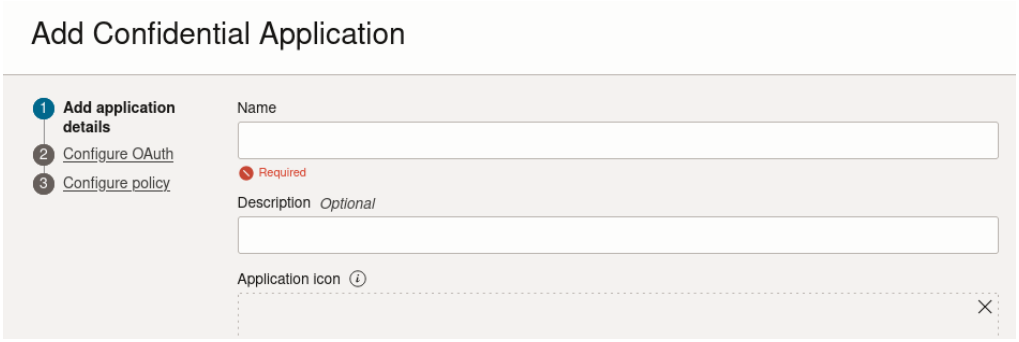
To be able to access protected resources the client application first needs an access token from the authorization server. In this case IDCS. The server needs to be setup beforehand. This is done by setting up a confidential application for an HSL client application.

A confidential client is considered confidential for its ability to keep a secret. Specifically, to keep its private key secret as we're using public key cryptography for signing and verifying JWT's.

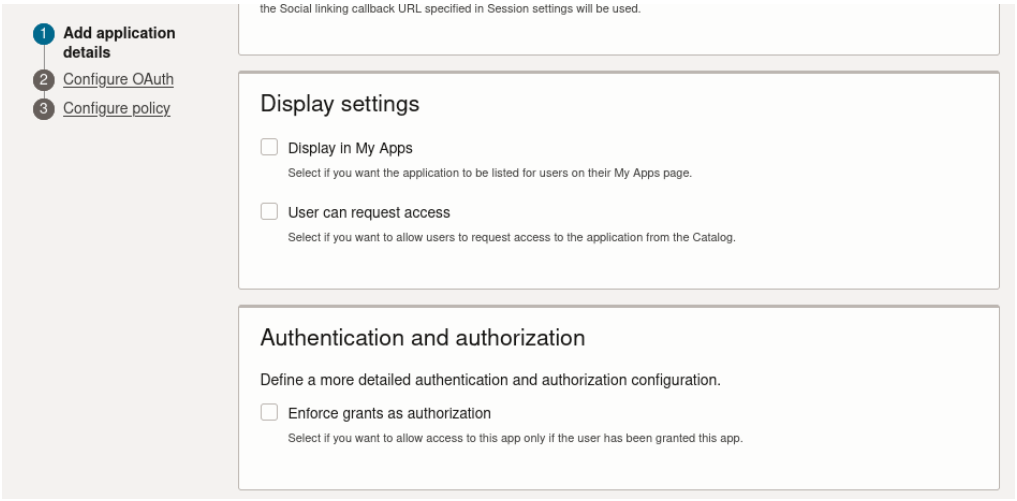
To add a new confidential application, navigate from the cloud console to the relevant identity domain ("Test domain" in the following description). From the domain overview navigate to "Integrated application." From there the "Add application" button can be used to start the setup.



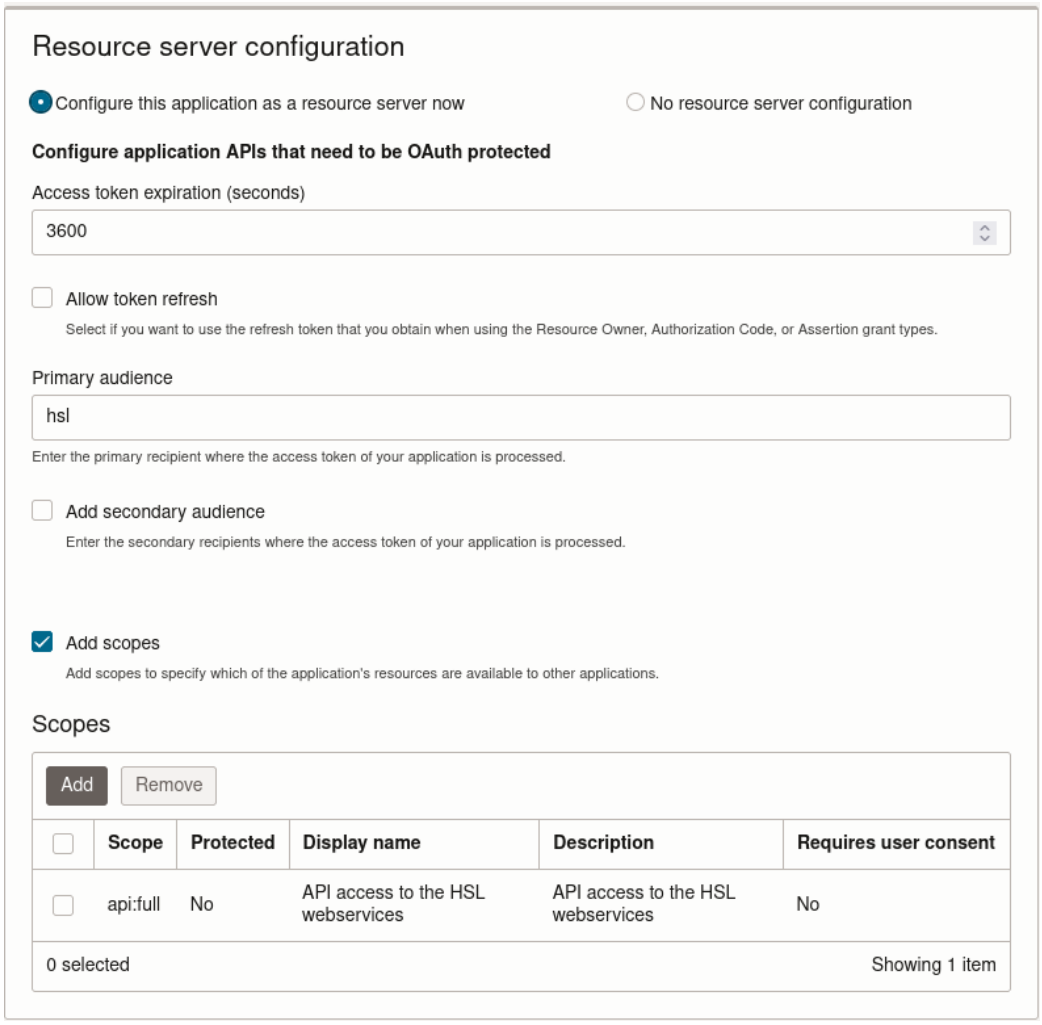
Provide a name and description for the new application.



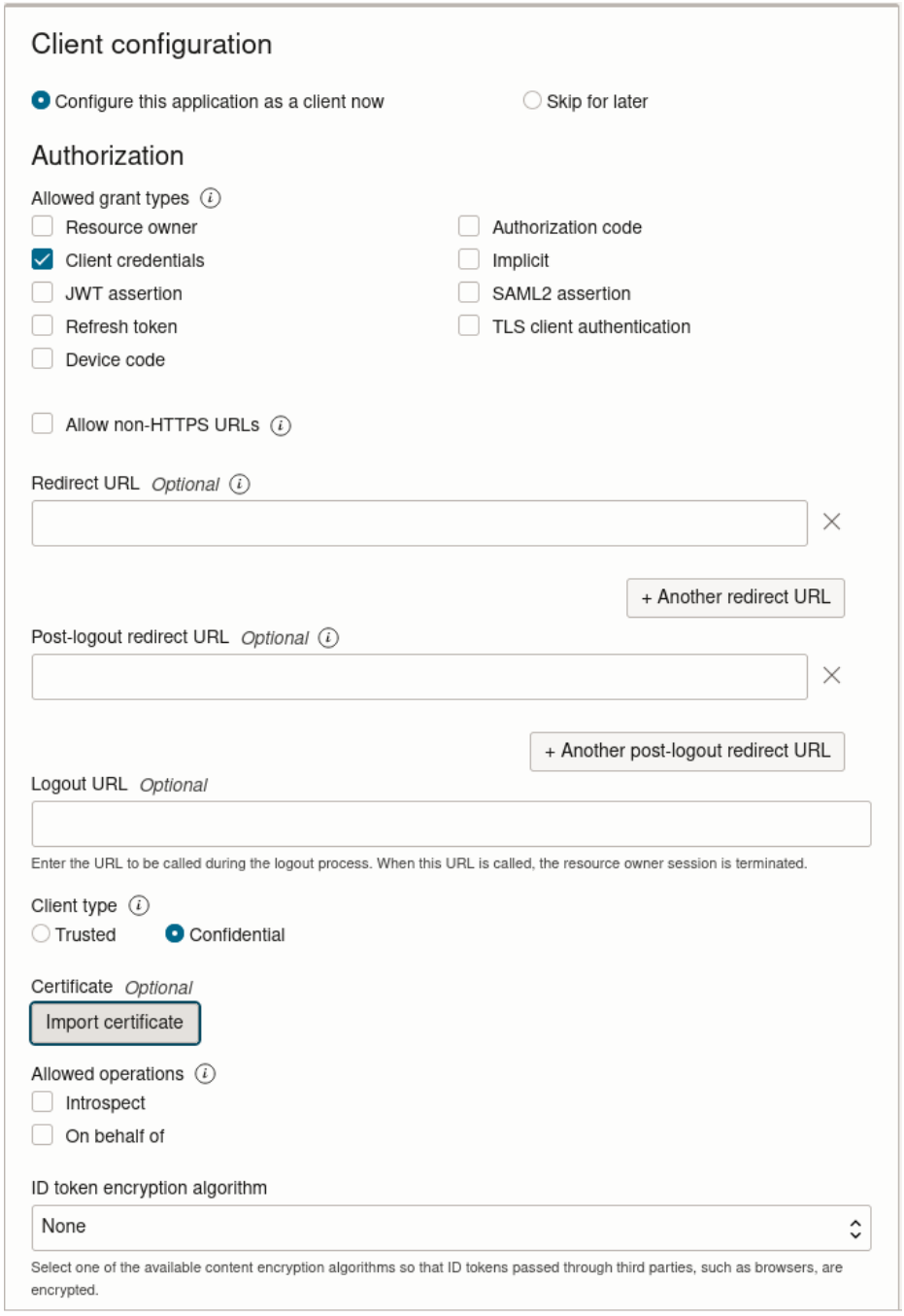
If the service account that's used to access the HSL applications is managed through IDCS then consider enabling the "Enforce grants as authorization" option.



Press next and add a Resource Server configuration.



Enter a primary audience and create relevant scopes. As we're configuring access to the HSL services 'hsl' is used here as the primary audience (primary recipient of access tokens generated by IDCS). Scopes describe what kinds of resources are available to other applications. We'll use the scope(s) defined here to restrict or allow access to resources by client applications by adding them in the context of specific resources in the configuration of OHS later on.

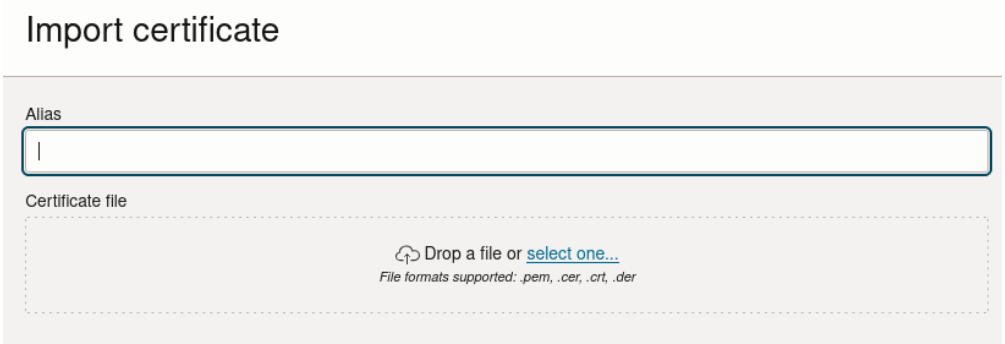


Moving on to the client configuration, starting with the allowed grant types.

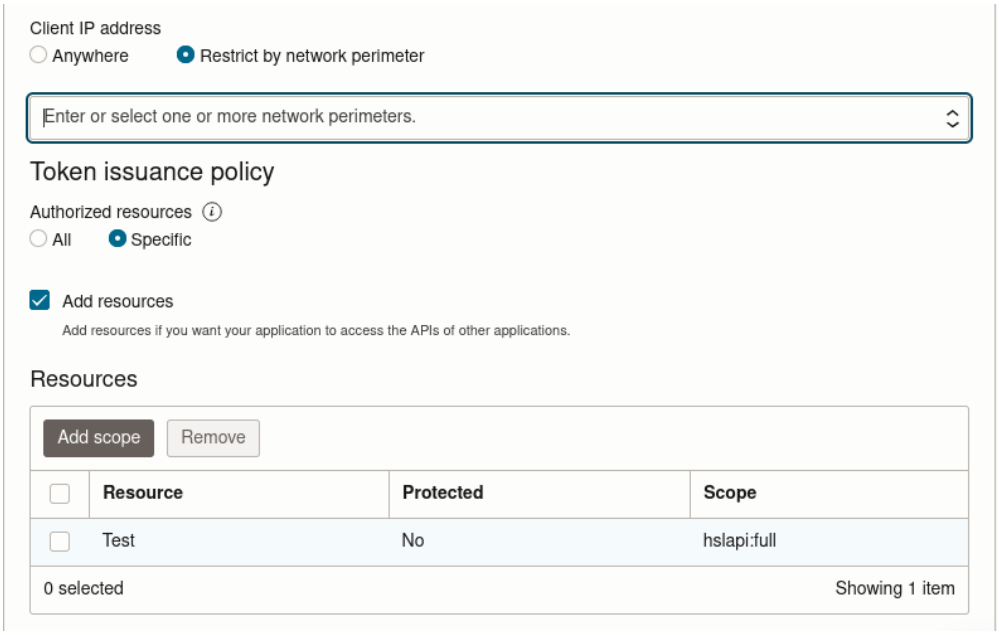
The only allowed option in this case is the client credentials flow. The client credentials grant flow permits a client application to use its own credentials (signed JWT) to authenticate when calling IDCS endpoints, instead of having to impersonate a user, for example.

You'll note the URLs in the configuration below are left empty. As the application is capable of requesting access to protected resources based on its own credentials there is no use for logout or redirect URLs. The application would simply be able to request a new token with which it can access a resource.

The client type option is set to "Confidential" as the client application using this configuration is capable of keeping its private key secret.

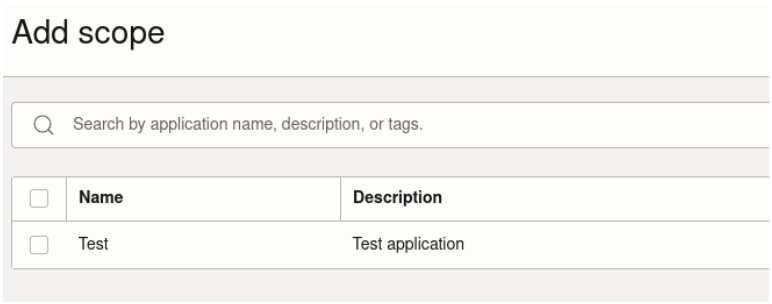


Through the "Certificate" option the public key counterpart of the private key of the client application can be uploaded and provided an alias. The alias is the `kid` or Key Id used in the JWT in paragraph 3.5.1. IDCS uses the alias/key id to be able to verify using the correct public key that requests for access tokens using the JWT signed by the client applications private key are in fact signed with the private key that only the client application should have access to.



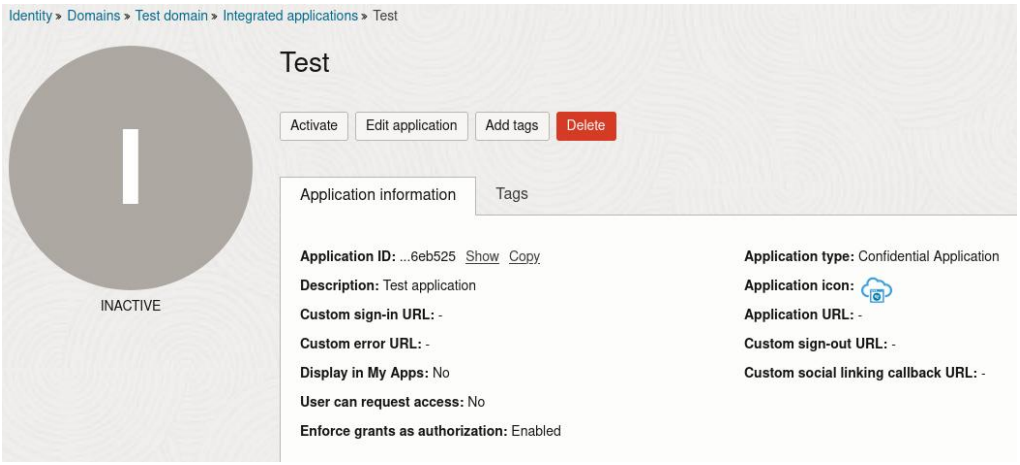
If able, it is advisable to restrict access to the IDCS application through the "Client IP Address" option by entering one or more network perimeters. Ask your network engineers for help setting this up.

Lastly, tokens issued by IDCS on behalf of this configuration should be restricted to specifically authorized resources. The resources previously configured must be added here.



Note that upon going through the workflow and configuring the client configuration for the first time, the "Resource server configuration" that was previously set up hasn't yet been saved, so it won't show up in the "Add scope" window. Press next and on the "Web tier policy" window press previous. The scopes configured in the resource server configuration are now saved and can be added through the "Add scope" window.

Setting up a Web tier policy and/or app gateway in OCI is considered out of scope here. Reference the OCI documentation for more information on this component through: https://docs.oracle.com/en-us/iaas/Content/Identity/applications/add-confidential-application.htm.



The configuration of the confidential application in IDCS is now complete. The application can be activated after pressing finish.

With IDCS set up, the signed JWT created in the previous section can be used to retrieve an access token. To find out where the JWT should be sent to, reference the *well-known* endpoint. Every application registration in IDCS is provided a publicly accessible endpoint that serves its OpenID configuration document. To determine the URI of the configuration document's endpoint for your application, append the *well-known OpenID configuration* path to your applications registration's *URL:* https://idcs-abcdef123456ghijk7890.identity.oraclecloud.com/.well-known/openid-configuration

The signed JWT is sent via an HTTP POST request to the activated IDCS application, e.g.

```
curl --header 'Accept: application/json' --header 'Content-
Type: application/x-www-form-urlencoded' --request POST
https://idcs- idcs-
abcdef123456ghijk7890.identity.oraclecloud.com/oauth2/v1/token?
grant_type=client_credentials&scope=<SCOPE>&client_assertion_ty
pe=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-
bearer&client_assertion=<SIGNED JWT>
```

SCOPE: specific scope (of access) that the client wants to receive an access token for. In the case of IDCS this is a combination of the primary audience and the specific scope access is requested for. Based on the configuration outlined above that would be 'hslapi:full'.

On a successful request an access token should be returned. The access token can be used as a bearer token for requests to the OHS instance and indirectly the HSL services, e.g.

```
curl -s -H 'Accept: application/json; Charset=utf-8' -H
'Accept-Language: en-US' -H 'Authorization: Bearer
eyJ4NXQjUzI[... omitted for brevity ...]' --get
'https://example.com:7580/HSL_CLA/cla/v1/vervoerDeclaratie/100'
```

3.5.3.1 *Key management and rotation*

Care should be taken to protect the private key used by the client application. As access to this key constitutes access to protected resources. The resources a key provides access to, should be tailored to the application and should be as narrow in

scope as possible. A way to achieve that for any given application is by setting up and configuring a specific confidential application in IDCS (with its own private key and scope of access) for that application.

In addition, a key needs to be rotated periodically as part of a healthy security posture. Factors that should be considered when defining a key management policy (and specifically a key rotation period) are considered out of scope for this document. For a more in-depth discussion regarding key management see the following report by the US National Institute of Standards and Technology (NIST): https://doi.org/10.6028/NIST.SP.800-57pt1r5. With regards to the key rotation period (also known as the cryptoperiod), chapter 5.3 of the report describes 1 to 2 years per rotation to be acceptable.

3.5.4 OHS – mod_auth_openidc installation and configuration

The next step is to configure OHS to act as an OAuth2.0 Resource Server for HSL services. In this example we will use the OHS component as it is installed in combination with the Oracle Forms product. Note that without OHS serving as a reverse proxy in front of the HSL services, the services are left unprotected! It is vitally important to have a reverse proxy that includes OAuth2 token verification, like the setup outlined in this paragraph, in front of the HSL webservices. The HSL services will only execute rudimentary checks on incoming requests to verify they are forwarded by a proxy-like application.

The apache module mod_auth_openidc should be installed on the OHI BO application tier. It is available from the ol8_appstream dnf/yum repository. To install it, login as root and run: `dnf install mod_auth_openidc`

After a successful installation locate and record the location of the installed module. The module needs to be registered in the OHS configuration for OHS to be able to load it as a part of its startup. Typically, the module can be found using: `rpm -ql mod_auth_openidc | grep mod_auth_openidc.so`

The module might be installed at something like:
`/usr/lib64/httpd/modules/mod_auth_openidc.so`

Next

Login as oracle to the OHI BO Application Tier and go to the OHS moduleconf directory:

```
oraset DOMFRS14120
cd
$DOMAIN_HOME/config/fmwconfig/components/OHS/instances/ohs14120
/moduleconf
```

Create or edit an OHS configuration file, ending with .conf, in this moduleconf directory as these are normally all included by the OHS httpd.conf file. Name it e.g. OHIBO.HSL.conf:

```
LoadModule auth_openidc_module
/usr/lib64/httpd/modules/mod_auth_openidc.so
<IfModule auth_openidc_module>

  OIDCOAuthVerifyJwksUri https://idcs- idcs-
abcdef123456ghijk7890.identity.oraclecloud.com/admin/v1/Signing
Cert/jwk
  <LocationMatch "^/HSL" >
    AuthType oauth20
    <RequireAll>
      Require valid-user
      Require claim "scope:api:full"
      Require claim aud:hsl
```

```
    </RequireAll>

    SetHandler weblogic-handler
    WebLogicHost localhost
    WebLogicPort 9522
    SecureProxy ON
    WLSSLWallet
"${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_TYP
E}/instances/${COMPONENT_NAME}/keystores/default"
    WLProxySSL ON
  </LocationMatch>
</IfModule>
```

This configuration manages a couple of things. It specifies where the mod_auth_openidc module can obtain the public key counterpart of the key IDCS used to sign access tokens through the 'OIDCOAuthVerifyJwksUri' option. This key is obtained the first time a request hits a location protected by OHS, in this case all locations starting with '/HSL'. The signing key is cached for a configurable amount of time, which is an hour by default. Incoming requests that send along JWT's will be verified to have been signed by IDCS are additionally required to contain a configurable set of claims. Here the scope and audience claim are used as an example.

Through standard Apache Require-directives access can be managed to resources. See https://httpd.apache.org/docs/2.4/howto/access.html for more information.

3.5.4.1 *mod_auth_openidc and mod_oauth2*

In an upstream version of the mod_auth_openidc module the developer has split-off the OAuth2.0 Resource Server functionality into its own module called mod_oauth2. This module might in the future be made available via the OL dnf/yum repositories. It offers identical functionality to what is currently provided by the 2.4.9.4-5 version of mod_auth_openidc in the OL8 and OL9 dnf/yum repositories with regards to an OAuth2.0 Resource Server implementation and can be used interchangeably.

3.5.5 WLS – enable proxy plug-in

For WebLogic to be able to proxy requests back and forth to OHS the WebLogic Plug-In must be enabled. To check:

In the WebLogic Remote Console:

- Connect to the Admin Server.
- Select the "Edit Tree".
- Select "Environment".
- Select "Servers"
- For the target server(s) for the HSL application
- In the next screen, in the tab "General":
- Click on "Show Advanced Fields"
- Make sure the "WebLogic Plug-In Enabled" option is on.

3.5.6 WLS – OHS as the only means of accessing HSL services

As the access to the HSL webservices is secured through OHS, it is vitally important that there are no other ways of accessing the HSL webservices without going through OHS.

If OHS and the WLS managed server should run on the same host the following configuration can be used to allow for the WLS managed server to be only accessible locally on that host. This is achieved through setting the "Listen Address" in the configuration of the managed server to localhost.

In the WebLogic Remote Console:

- Connect to the Admin Server.
- Select the "Edit Tree".
- Select "Environment".
- Select "Servers"
- For the target server(s) for the HSL application
- In the next screen, in the tab "General":
- Set "Listen Address" to "localhost".

If OHS (or another application serving the same purpose) and the HSL services don't reside on the same host, access to the HSL services should still be routed via the reverse proxy to protect the services. Whether this could be setup transparently via firewall rules or as part of a cloud-based solution is considered out-of-scope for this manual. Include your network administrator in this conversation.

3.5.7 HSL service configuration

The HSL services are configured through the hsl.properties file. Chapter 5 describes the possible configuration properties the file might contain in more detail. For the purposes of the OAuth2.0 setup it should be noted that the following properties must be present in the configuration file:

- `hsl.<app>.authorization` (or when this property isn't set the fallback `hsl.authorization` property). The property must have the value TOKEN.
- `hsl.<app>.allowedorigins` (or when this property isn't set the fallback `hsl.allowedorigins` property). This property must contain the host and port combination of the OHS reverse proxy, e.g., https://example.com:7580

3.5.8 Troubleshooting OHS and mod_auth_openidc

To troubleshoot OHS the following statements can be added to the HSL OHS configuration (httpd.conf) to enable debug logging:

```
LogLevel info auth_openidc:debug weblogic:debug
```

The LogLevel-directive sets a default log level of info for all log messages except the mod_auth_openidc module and weblogic. They get a level of debug.

3.5.9 OHS log message - nzos_Handshake: mod_wl failed

If the OHS log should contain something like the following messages:

```
[2000-01-01T12:01:22.1000+02:00] [OHS] [ERROR:32] [OH99999]
[weblogic] [host_id: example.com] [host_addr: 100.100.100.1]
[pid: 1059400] [tid : 140178737800960] [user: oracle] [ecid:
0067NOLxtkk9d_eUtaXvWH0042d8000002] [rid: 0] [VirtualHost:
main] nzos_Handshake: mod_wl failed (29024)
[2000-01-01T12:01:22.2000+02:00] [OHS] [ERROR:32] [OH99999]
[weblogic] [host_id: example.com] [host_addr: 100.100.100.1]
[pid: 1059400] [tid : 140178737800960] [user: oracle] [ecid:
0067NOLxtkk9d_eUtaXvWH0042d8000002] [rid: 0] [VirtualHost:
main] wl_ssl_open : SSL Handshake failed onserror : Success,
error : 29024, status : 2
[2000-01-01T12:01:22.3000+02:00] [OHS] [ERROR:32] [OH99999]
[weblogic] [client_id: 10.10.10.1] [host_id: example.com]
[host_addr: 100.100.100.1] [pid: 1059400] [tid:
140178737800960] [user: oracle] [ecid:
0067NOLxtkk9d_eUtaXvWH0042d8000002] [rid: 0] [VirtualHost:
example.com:7580] <0067NOLxtkk9d_eUtaXvWH0042d8000002>
*******Exception type [NO_RESOURCES] (Could not open secure
```

```
connection) raised at line 1829 of URL.cpp
[2000-01-01T12:01:22.4000+02:00] [OHS] [ERROR:32] [OH99999]
[weblogic] [client_id: 10.10.10.1] [host_id: example.com]
[host_addr: 100.100.100.1] [pid: 1059400] [tid:
140178737800960] [user: oracle] [ecid:
0067NOLxtkk9d_eUtaXvWH0042d8000002] [rid: 0] [VirtualHost:
example.com:7580] Trying GET
/OHI/EXAMPLE/HSL_CLA/cla/v1/vervoerDeclaratie/1000 at backend
host '::1/9522; got exception 'NO_RESOURCES: [os error=0,  line
1829 of URL.cpp]: Could not open secure connection'
```

and if OHS is configured for an SSL connection to WLS refer to Doc ID 1665711.1 on My Oracle Support for more information and a likely fix.

3.6 (Re)deployment of the HSL Application

The HSL web applications are packaged in a single archive named 'HSLBOWS.ear'. This EAR file must be deployed to WLS.

The EAR file containing the HSL applications resides in the $OZG_BASE/java directory on the application server containing the OHI Back Office software.

You can copy this to another location if required.

The WebLogic Remote Console can deploy files located on your local PC and on the Application Server where the WebLogic Remote Console runs (for the Hosted WebLogic Remote Console).

To automate deployments, you can use java weblogic.Deployer. See A weblogic.Deployer Command-Line Reference.

Note that you cannot use an older EAR file with a newer OHI Back Office release and vice versa.

The following scenarios are discussed:

- Deploy to a single Managed Server
- Deploy to multiple Managed Servers
- Deploy to a WebLogic cluster
- Deploy for DTAP (development, test, acceptance, production)

3.6.1 Deploy to a single Managed Server

3.6.1.1 *Deploy EAR file*

- Log in in the Weblogic Remote Console.
- Connect to the Admin Server of the HSL web services Domain.
- Select the "Edit Tree".
- Select "Deployments"
- Select "App Deployments"
- Click on "New"
- Enter a name that includes the OHI environment name, to help identify the deployment, e.g., HSL_VOHI.
- Select the Managed Servers that are the Target(s)
- If you deploy the ear file from your laptop, set "Upload" to On, and select the ear file using the popup after "Source".
- If you deploy an ear file that is already located on the application server, set "Upload" to Off. Enter the file name (including the directory) in "Source Path".

- Set “Security Model” to the desired option. Select the Security Model as discussed in 7.1 *Background information*.
- Set “Staging Mode” to “Stage” so the ear file will be available on the Managed Server for redeployment.
- Set “On Deployment” to “Start Application”.
- Click on “Create”.
- In the next page, review the settings. At this moment the version of the .ear file is also shown.
- If needed, change values and click on “Save”.
- Click on the Shopping Cart, top right, and select “Commit Changes”.
- Go to the “Monitoring Tree” (via “Home” or via the icon in the left margin).
- Click on ”Deployments”.
- Click on “Application Management”.
- Check the “State” of your deployed application. This should be “Active” if the hsl.properties file has been specified and can be found.
- In the “Monitoring Tree”, click on “Application Runtime Data”.
- Click on your deployed application.
- Check the “Health State”. This should be “Okay”.

3.6.1.2 *Specify configuration file*

Before using the HSL web services, implement the following actions as described below. These actions have to be executed only once. There is no need to repeat them when you update a deployment or delete and install it again.

Add a Server Start argument by adding a line to the file $DOMAIN_HOME/bin/setUserOverrides.sh you created earlier. Add the line to the part for the HSL Managed Server, as indicated below:

```
# specify additional java command line options for specific servers
if [ "${SERVER_NAME}" = "MS_HSL1420_OHIPRD"1
then
  # add settings for HSL
  # Custom Setting for MS_HSL1420_OHIPRD to set debug level for SSL
  JAVA_OPTIONS="-Dweblogic.wsee.security.debug="true" ${JAVA_OPTIONS}"
  # Set location for HSL properties file
  JAVA_OPTIONS="-Dhsl.properties="/u01/app/oracle/product/OHI/vohi/hsl.properties" ${JAVA_OPTIONS}"
fi
export JAVA_OPTIONS
```

- Make sure to keep the parts with ${JAVA_OPTIONS} on the same line.

This example uses a properties file with the name hsl.properties which is located in the $OZG_BASE folder of your OHI Back Office application server environment, but you can specify any name and location.

The contents of this file are discussed in Chapter 5 *Configuration files for HSL services*).

When completed, (re)start the Managed Server. This can be done from the WebLogic Remote Console, or from the command line with the following commands:

```
cd $DOMAIN_HOME/bin
./startManagedWebLogic.sh MS_HSL1420_OHIPRD http://<FQDN>:7061
```

The example above contains the Managed Server’s name as first parameter and the listen address of the Admin Server of the domain as second parameter.

Check in the <ManagedServer>.log file and/or in the <ManagedServer>.out file in $DOMAIN_HOME/servers/<ManagedServer>/logs directory of your Managed Server whether the command line contains the arguments as specified above.

NOTE: If the file specified by hsl.properties cannot be read, messages as below will show up in the WebLogic console and in the <ManagedServer>.log file:

```
java.lang.RuntimeException: Property file could not be loaded.
….
java.io.FileNotFoundException:
/u01/domains/OHIDEV01/conf/psl.properties (No such file or directory)
```

3.6.2 Deploy to multiple Managed Servers

You may deploy the application to more than one target.

Example: if you choose to target the application to Managed Servers MS1 and MS2 runnig on the same application server, the application will be available on separate end points. The URLs of these end points will only differ in port number. You will have to configure a load balancer or Oracle HTTP server to distribute the load and/or do failover.

If you choose this rather unlikely scenario, be aware that each Managed Server should have different Server Start argument values (for hsl.properties).

3.6.3 Deploy to a WebLogic cluster

You may deploy the application on all the Managed Servers of a WebLogic cluster. This may be needed for better scalability. Be aware to use some form of load balancing to allow the use of a single end point by the client applications.

The best way to implement this type of deployment depends on your specific situation.

If you are planning a load balanced environment with multiple Managed Servers in a WebLogic cluster, it is vital that the configuration of every Managed Server is aligned with the others.

3.6.4 Deploy for multiple environments (DTAP)

If you use several OHI-related environments to support the various DTAP (Develop-Test-Accept-Production) stages, you may want to have different versions of the HSL application running at the same time.

To implement this, you need to:

- Create a Managed Server for each of the DTAP stages.
- Create a data source for each of the DTAP OHI Back Office databases and deploy that data source only to the corresponding Managed Server.
- Create an hsl.properties file for each Managed Server.
- Configure each Managed Server to start up with the appropriate hsl.properties. Add multiple tests for Managed Server names in `setUserOverrides.sh` to specify a different file name for each Managed Server.
- Deploy the appropriate version of the HSL application to its corresponding Managed Server(s) and give it a unique deployment name to identify its deployment.

3.6.5 Validate deployment

Be aware that any URLs displayed in the WebLogic Remote Console cannot be used to test or validate the deployment.

Also note that, even with the correct URLs, you cannot use a browser to test, because the request needs to include a Request Header "Accept:application/json".

You may get no response, or a reply like this:

```
<exceptionResponse>
<internalStatus>Not Acceptable</internalStatus>
<message>Wrong value for Accept</message>
</exceptionResponse>
```

Instead, use curl, as described in chapter 4 *Deployment validation* or SoapUI, as described in 7.4 *Testing with SoapUI*.

3.7 Additional Security Aspects

The HSL services provide an additional channel to retrieve and change OHI Back Office data.

Therefore, you must prevent unauthorized use of the HSL applications.

Please consult the "Oracle Health Insurance Security Guide" Doc[4] for more information about OHI Back Office security aspects.

To prevent the exposure of sensitive data or unauthorized changes to the OHI Back Office data, access of the HSL applications should be limited to trusted systems and interfaces. When access to the HSL applications is secured through OHS (as it is for Single Sign-On using OpenID Connect), it is vitally important that there are no other ways of accessing the HSL webservices without going through OHS.

Finally, note that it is your responsibility as an administrator to secure the HSL services within your organization.

3.7.1 Cross-Site scripting protection

As a measure to prevent against potential cross site scripting attacks you can limit the callers of the REST services by specifying a set of trusted origins. The set of origins can be added as a space-separated list to the property 'allowedorigins' in the file `hsl.properties`.

4 Deployment validation

Especially when deploying a HSL application for the first time, it makes sense to validate that the HSL application is in working order.

Before you begin, check in the WebLogic Remote Console that the deployment status of the HSL application is "Active".

The following validation tests can be performed by the administrator:

- getDatabaseInfo operation
- Get Runtime Swagger definition

The getDatabaseInfo operation requires a JDBC connection between the HSL application and the OHI BO database, so you are not only testing the deployment itself but also the integration between the HSL application and the OHI database.

If a validation test fails, see the paragraph '*Troubleshooting*' below to find and resolve the problem.

The validation tests described below assume that you test with 'curl'.

4.1 Testing with Curl

An operation of the HSL application can be invoked with many HTTP client tools. One of these tools is curl, which is present on any Linux/Unix server.
Assuming that you have terminal access to the Linux server running the reverse proxy in front of the HSL services, curl is a good tool to run the deployment validation tests.

Use '`curl --version`' to check the curl version. Ensure that you are running curl 7.35.0 or higher as that supports the required SSL implementation.

First an access token needs to be retrieved from an identity provider. Continuing with the setup outlined in chapter 3 for OAUTH2, we'll use IDCS as an example.

```
curl --header 'Accept: application/json' --header 'Content-
Type: application/x-www-form-urlencoded' --request POST
https://idcs-
250703492t3452395h234985236eb5525.identity.oraclecloud.com/oaut
h2/v1/token?grant_type=client_credentials&scope=hslapi%3Afull&c
lient_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-
assertion-type%3Ajwt-bearer&client_assertion=<SIGNED JWT>
```

Explanation of the used options and placeholders:

- `--header 'Accept: application/json'`
  We'd like IDCS to send the access token and accompanying information back in json format.
- `--header 'Content-Type: application/x-www-form-urlencoded'`
  This header specifies the way in which the information sent as part of the IDCS url has been formatted.
- `--request POST`
  add HTTP POST verb

If, or rather when, IDCS accepts the signed JWT, the access token is sent as part of a json payload that includes metadata about the requested access and token, e.g. the expiry time of the access token or the scope of access. The access token itself is contained in the 'access_token' field of the payload. We'll use this token in the upcoming curl requests in this chapter.

A typical invocation of an HSL operation using curl would look like this

```
curl --dump-header - --request <verb> --insecure --header
"Accept:application/json" --header "Authorization: Bearer
<access token>" <url>
```

Explanation of the used options and placeholders:

- `--dump-header -`
  Dump response headers to stdout
- `--request <verb>`
  add HTTP verb (GET/PUT/POST/PATCH/DELETE)
- `--insecure`
  Allow curl to run HTTP requests without checking SSL certificates.
- `--header "Accept:application/json"`
  Add request header to require a response in application/json format.
  This is required for every HSL operation.
- `--user <user>`
  The username of the WLS user used for Basic Authentication.
  The <user> must refer to an existing WLS user.
  Note that curl will prompt for a password if it is not given on the command line.
- <access token>
  Access token retrieved from an identity provider.
- <url>
  The path to the HSL operation.

The url format is https://server:port/application/path, where

server	**This must be one of the managed servers protected by a reverse proxy that includes a OAuth2 resource server implementation, like the OHS setup as specified in chapter 3 of this manual. In addition, the managed server must be listed in the WebLogic Remote Console as an active target for the HSL application.**
port	The SSL port of the reverse proxy running the HSL application.
application	This is the name of the HSL application as listed on the WLS deployment page. For example, HSL_POL, HSL_REL, HSL_CLA or HSL_C2B.
path	The path to this operation. Each operation is uniquely identified by a <path> + <verb> combination. Path examples: '/dbinfo' or 'api/swagger.json'

In the following example, information about the database connection is requested from the HSL_POL application on the local WLS host running a managed server accessed via an OHS instance that serves as a reverse proxy at SSL port 7094:

```
curl --dump-header - --insecure --header "Authorization: Bearer
yJ4NXQjUzI1NiI6IlRyYmlwdUM0YjJGbzBBeXJ6dm9IVGpZRlhLQX...[omitte
d for brevity]" --header "Accept:application/json"
https://localhost:7094/HSL_POL/dbinfo
```

4.2 getDatabaseInfo

This operation provides information about the database connection between the HSL application and the OHI BO database.
If you are familiar with OHI BO's SVL services, note that the getDatabaseInfo operation is comparable with the 'isAlive' operation implemented in every SVL service.

This operation requires a working database connection and invokes the PL/SQL implementation package specific to the HSL application. Additionally, the value of "hsl.usercontext" in hsl.properties must match a "functionaris".

The getDatabaseInfo operation is invoked through

https://**server:port/application**/dbinfo

4.2.1 OAUTH2

In the following example the getDatabaseInfo of the HSL_POL application accessed via an OHS instance that serves as a reverse proxy on SSL port 7094 in front of our local WLS host is invoked through curl.

```
curl --dump-header - --insecure --header "Authorization: Bearer
yJ4NXQjUzI1NiI6IlRyYmlwdUM0YjJGbzBBeXJ6dm9IVGpZRlhLQX...[omitte
d for brevity]" --header "Accept:application/json"
https://localhost:7094/HSL_POL/dbinfo
```

The output is a JSON object with information about the database connection and the PL/SQL package implementing the HSL application in the OHI BO database:

```
{
   "basePath": https://localhost:7094/HSL_POL/pol,
   "database": "BDDEV1722",
   "instance": "CDB02",
   "jndiName": "HSL_BDDEV1722",
   "plsqlPackage": "hsl_pol_sp_pck $Revision: 4.21 $",
   "user": "HSL_USER",
   "userContext": "HSL_WEB_USER"
}
```

4.2.2 Basic Authentication

The same call, using Basic Authentication and the Security Model "DD Only":

```
curl --insecure -vX GET -H "Content-Type: application/json" -H
"Accept:application/json" --user restuser
https://localhost:9542/HSL_POL/dbinfo
```

4.3 verifyInterfaceVersion

This operation checks whether the deployed version at the application server and the corresponding objects as installed in the database do match with each other.

The verifyInterfaceVersion operation is invoked through

https://**server:port/application**/api/verifyInterfaceVersion

In the example below the operation is called for application HSL_CLA:

```
curl --dump-header - --insecure --header "Authorization: Bearer
yJ4NXQjUzI1NiI6IlRyYmlwdUM0YjJGbzBBeXJ6dm9IVGpZRlhLQX...[omitte
d for brevity]" --header "Accept:application/json"
https://localhost:7410/HSL_CLA/api/verifyInterfaceVersion
```

The output is a JSON object like:

```
{
  "items":
    [{"name":"plsqlSwaggerRevision","value":"42719"}
    ,{"name":"warSwaggerRevision","value":"42719"}
    ,{"name":"match","value":"true"}
    ]
}
```

It states whether both versions do match (true) or not (false). They should match and if not, you should find out what went wrong: is the problem at the database side of the application server deployment side?

4.4 Get Runtime Swagger definition

Each HSL application has an operation to generate a Swagger definition which documents the operations and the objects of the HSL service.
This documentation is not only useful to client application developers but can also be used as the basis for code generation.

The Swagger 2.0 and OpenApi 3.1 standards are supported by many leading software vendors including Oracle. They are documented on https://www.openapis.org.

The Swagger 2.0 definition can be retrieved as follows:

- `https:/`**`server:port/application`**`/api/swagger.json`
  Returns the Swagger definition in JSON format
- `https:/`**`server:port/application`**`/api/swagger`
  Returns the Swagger definition in JSON format
- `https:/`**`server:port/application`**`/api/swagger.yaml`
  Returns the Swagger definition in YAML format

The OpenApi 3.1 definition can be retrieved as follows:

- `https:/`**`server:port/application`**`/api/openapi.json`
  Returns the OpenApi definition in JSON format
- `https:/`**`server:port/application`**`/api/openapi`
  Returns the OpenApi definition in JSON format
- `https:/`**`server:port/application`**`/api/openapi.yaml`
  Returns the OpenApi definition in YAML format

In the following example we retrieve the runtime Swagger definition of the POL service:

```
curl --dump-header - --insecure --header "Authorization: Bearer
yJ4NXQjUzI1NiI6IlRyYmlwdUM0YjJGbzBBeXJ6dm9IVGpZRlhLQX...[omitte
d for brevity]" --header "Accept:application/json"
https://localhost:7094/HSL_POL/api/swagger.json
```

The output is a JSON object containing the Swagger definition of the deployed HSL application.

For retrieving the YAML format beware that you specify x-yaml in the -H argument:

```
curl --dump-header - --insecure --header "Authorization: Bearer
yJ4NXQjUzI1NiI6IlRyYmlwdUM0YjJGbzBBeXJ6dm9IVGpZRlhLQX...[omitte
d for brevity]" --header "Accept:application/x-yaml" -header
"Content-Type: application/json"
https://localhost:7094/HSL_POL/api/swagger.yaml
```

4.4.1 Saving the Swagger definition to a file

Curl can also write the response of an invocation to a file, using the '`--output <filename>`' option

In the example below we save the online Swagger definition of the POL service running on localhost at SSL port 7094 to a file called 'saved_swagger.json':

```
curl --dump-header - --insecure --header "Authorization: Bearer
yJ4NXQjUzI1NiI6IlRyYmlwdUM0YjJGbzBBeXJ6dm9IVGpZRlhLQX...[omitte
d for brevity]" --header "Accept:application/json" --output
"saved_swagger.json"
https://localhost:7094/HSL_POL/api/swagger.yaml
```

4.5 Troubleshooting

If the deployment validation fails, first check that the following items have been configured correctly:

- **hsl.properties configuration file (see next chapter)**
  This sets the data source for your HSL application.
- **hsl.properties Server Start argument**
  This parameter tells the HSL application where to find the hsl.properties file. If the hsl.properties parameter refers to a non-existing file, the HSL application cannot be started by WLS and its state will be 'Failed'. Check the file $DOMAIN_HOME/bin/setUserOverrides.sh.
- **Data source configured in hsl.properties configuration file**
  This data source is used to create the JDBC connection between the HSL application and the OHI BO database
- **HSL database account**
  This account in the OHI BO database that is used in the data source and has access to the PL/SQL components used by the HSL application.
- **hsl.usercontext**

  The value of this parameter in hsl.properties must be a "functionaris" (registered application user, employee). For functional web service requests, the functionaris must also have module authorization.
- **The configuration of the reverse proxy (OHS in the context of this manual)**
  See paragraph 3.5.7 for more information on how to enable debug output for OHS.

Troubleshooting tips:

- Edit the hsl.properties file and set 'developermode=true' for your HSL application.
- Restart the managed server for your application.
  Error messages will now be included in the output (normally they are suppressed from the output).
- Reproduce the request with curl. Be sure to use the 'dumpheader' option (`--dump-header`) to dump the response headers.

The table below may help you to pinpoint the problem:

HTTP	Message	Problem	Action
	WebLogic Remote Console and Managed Server logs: java.lang.RuntimeException: Property file could not be loaded.	The configuration file could not be loaded when starting the HSL application.	Restart application after ensuring that the file referred to by the 'hsl.properties' Server Start argument exists and is readable.
500	Unable to resolve 'xyz'	The jndiname property for this application does not refer to a valid data source.	Examine hsl.properties and ensure that the application's jndiname points to a valid datasource.
500	ORA-06550:line 1..	The required HSL objects cannot be accessed by the database user related to the data source.	Verify that the data source points to a HSL database account. Verify that the HSL database account has access to the HSL objects (see

			'Creating a HSL database account' above).
401	Missing Authentication Scheme	No authorization header has been supplied as part of the request	Add an authorisation header that includes the access token that was retrieved from the identity provider to the request.
401	Unauthorized or Forbidden	Invalid or expired access token	Add a valid access token to the request
422	Functional Error	A syntactically correct request could not be completed due to a functional error or business rule violation.	Adapt the request or the database situation so the cause of the failure is prevented.
412	Precondition Failed	Another user already updated these data.	Refresh data and try again.

5 Configuration Files for HSL services

In the previous chapter a properties file was referenced in the web service application server deployment description. This chapter provides more information about that file.

5.1 Properties file templates

With the OHI Back Office release installation, a properties file template called hsl.properties.template is distributed to the $OZG_BASE/conf/Back-Office directory. Each OHI Back Office release may overwrite this template with an updated version. The hsl.properties.template can be used as an example to create your own hsl.properties file (for example in $OZG_BASE/conf).

Please note that all values are examples. You should consider if these values are appropriate for your installation and replace them with your own values if needed. Values indicated with <<SOME_NAME>> in the templates are placeholders and must be replaced. This notation is intended to make scripted deployment easier.

Also make sure not to set log level to FINE, FINER or FINEST in production mode, use SEVERE or WARNING instead.

5.2 Back Office HSL properties file

The location of the Back Office properties file for the HSL services is specified as a Server Start argument for a Managed Server with:

```
-Dhsl.properties=<filename>
```

See paragraph *3.6.1Deploy to a single Managed Server* to set the Server start argument using the file setUserOverrides.sh.

This file contains properties to configure the various deployed HSL applications:

- Datasource to connect the HSL application to the OHI database.
- Default OHI employee (Dutch: functionaris) on whose behalf a request is executed. This can (should?) be generic name (e.g. HSL_FUNC_USER) without a corresponding database account. It is only used for the auditing columns in the OHI BO tables. The functionaris must have module authorisations for the Web Services. These must be registered via the OHI Back Office Forms application.
- Logging configuration.
  Note that HSL services use Java Util Logging (JUL). You may find more information about the configuration options of JUL on the internet.

In the subsection below, the properties are described in more detail, where "<app>" is a placeholder for the service name, like "rel", "pol" or "polis" (see the properties file template for more examples).

5.2.1 hsl.<app>.jndiname /hsl.jndiname

The JNDI name of the data source to connect this HSL application to the OHI database is configured in `hsl.<app>.jndiname`. If not set, this value defaults to the value of the `hsl.jndiname` property.

There is no default value. If `hsl.jndiname` is not set, this will result in an error.

Setting `hsl.<app>.jndiname` allows you to use different data sources for different HSL applications. A different data source may connect to the same database using a different account, or to a different database altogether.
As an example, you may want to use HSL_PRD for the REL service and HSL_RO ('read only') for the POL service to avoid changes to the policies in the production database.

Note that you must use // for each forward slash in the JNDI name.

Example:

```
hsl.jndiname=jdbc//DS_OHI_HSL_PRD
```

5.2.2 hsl.<app>.authorization /hsl.authorization

By default, HSL service operations require OAUTH2 authentication. The setting must be set to TOKEN for this form of authentication to take effect.

As a fallback, Basic authentication is provided as a minimal policy to reduce the risk of unauthorized access and network sniffing. Basic authentication requires HTTPS communication and providing a WebLogic username/password with each call.

To enforce basic authentication, the value of `hsl.<app>.authorization` or the default `hsl.authorization` property, should be set to BASIC.

5.2.3 hsl.<app>.usercontext /hsl.usercontext

The OHI employee (Dutch: functionaris) on whose behalf a request is executed.

Setting `hsl.<app>.usercontext` allows you to set an OHI employee per HSL application. If not set, this value defaults to the value of the `hsl.usercontext` property.

The user context is inserted in the call context which is included in the call to the PL/SQL implementation procedure. Note that the PL/SQL implementation may set a different OHI employee based on the request data.

This user context determines the user identity that is used for authorising the web service request in the database and for logging changes to the data. The value must be the Oracle username of a registered BackOffice user (in Dutch: “Functionaris”). This user context will also be used to determine the access to the financial units when applicable (in Dutch: "Bedrijfstoegang").

Example:

```
hsl.usercontext=HSL_FUNC_USER
```

the usercontext must always be set, even if `hsl.<app>.usercontext.control` is set to TOKEN

the user context from the hsl.properties file may be overwritten in the SQL at the HSL application level, for the specific service or even operation. This should be documented in the functional specification(s) which apply to the given HSL application.

Do not use the technical account (HSL_USER) that is used for the data source. That account should not be registered as a “functionaris”.

Do not use the name of a Back Office user (”functionaris”) that also has a database account.

Do not use the value “MANAGER”. Records created and updated by HSL functionality should be recognizable as such. Using MANAGER will make it impossible to distinguish those records from records created or updated by batch procedures run by MANAGER and conversion scripts run during release installations, as these latter ones use hardcoded MANAGER.

☞ You need to specify a specific user context for the hsl.<…>CHECK web services, and authorize the "functionaris" for that web service. Do NOT use the same value that you use for `hsl.usercontext`.

hsl.agbcheck.usercontext=<<<HSL_AGBCHECK_CONTEXT_USER>>

hsl.bsncheck.usercontext=<<<HSL_BSNCHECK_CONTEXT_USER>>

hsl.ccocheck.usercontext=<<<HSL_CCOCHECK_CONTEXT_USER>>

hsl.tupcheck.usercontext=<<<HSL_TUPCHECK_CONTEXT_USER>>

5.2.4 hsl.<app>.developermode /hsl.developermode

For security reasons, a response for a failed request contains minimal information so that potential hackers cannot use this information to misuse the HSL services. For functional errors, being either a BAD REQUEST or an OHI business rule violation, the functional error is returned. For all other errors, the error message will be replaced with 'Non-disclosed'. The original error message is written to the log file.

If `hsl.<app>.developermode` is set to 'true', the response for a failed request contains the original error message for all errors. If not set, this value defaults to the value of the `hsl.developermode` property. If `hsl.developermode` is not set, developermode is disabled.

Note that in production mode it is strongly advised to delete the `hsl.developermode` and `hsl.<app>.developermode` property from the `hsl.properties` file.

See **Doc[2]** ('Error Handling') for the differences in error handling between developer mode and non-developer mode.

5.2.5 hsl.<app>. allowedorigins /hsl.allowedorigins

For security reasons, to provide protection from cross-site scripting attacks, you can limit the callers of the REST services by specifying a set of trusted origins. See also the separate paragraph with 'Additional Security Aspects' earlier in this document describing this property.

If `hsl.<app>.allowedorigins` is not set, this value defaults to the value of the `hsl.allowedorigins` property. If `hsl.allowedorigins` is not set, protection is not enabled.

Note that in production mode it is strongly advised to set explicitly trusted origins through this property.

An example:

```
hsl.allowedorigins=https://server.domain.com:7430, https://localhost:8000
```

5.2.6 hsl.<app>.logdirectory, logfilesuffix, logfile / hsl.logdirectory, logfilesuffix, logfile

hsl.<app>.logdirectory / hsl.logdirectory

hsl.<app>.logfilesuffix / hsl.logfilesuffix

hsl.<app>.logfile hsl.logfile

To minimize maintenance, the properties `hsl.logdirectory` and `hsl.logfilesuffix` can be used instead of the `hsl.<app>.logfile` property.

With `hsl.logdirectory`, a generic logdirectory can be defined. When there is no fully qualified logfile defined in `hsl.<app>.logfile`, the log file for the HSL application will be written to this location.

With `hsl.logfilesuffix` a generic logfile name suffix can be defined. When no `hsl.<app>.logfile` property is defined, it will default to "hsl.<app>.log", but when the suffix is set the logfile will default to "hsl.<app>.<logfilesuffix>.log".

When `hsl.logdirectory` is set, the `hsl.<app>.logfile` property no longer has to be set. The HSL application will be able to determine the log file name and location from `hsl.logdirectory` (and optionally `hsl.logfilesuffix`). This means that you will not have to add properties for new HSL applications.

These properties can also be set at app level: `hsl.<app>.logdirectory` and `hsl.<app>.logfilesuffix`, but that will not help to reduce the configuration effort.

`hsl.<app>.logfile` can only be set at <app> level.

Default values:

- `hsl.<app>.logdirectory`: no default value
- `hsl.<app>.logfilesuffix`: no default value
- `hsl.<app>.logfile`: "hsl.<app>.log"

Note:

- The directory referenced in `hsl.<app>.logdirectory` / `hsl.logdirectory` must exist and must be writable by the OS user running the WLS application server.
- The `hsl.<app>.logfile` property can be a filename only or a fully qualified path including a filename.

Examples:

- `hsl.logdirectory=/u01/log`
  `hsl.logfilesuffix=BOPROD_AS1`
  `hsl.<app>.logfile` is not set
  This wil result in logfiles for all HSL applications as follows:
  `/u01/log/hsl.<app>.BOPROD_AS1.log`

  This is the recommended setup.

- `hsl.logdirectory=/u01/log`
  `hsl.logfilesuffix` is not set
  `hsl.<app>.logfile` is not set

  This wil result in logfiles for all HSL applications as follows:
  `/u01/log/hsl.<app>.log`

  NOTE: do not use this setup for multiple OHI Environments and/or multiple Managed Servers (e.g. in a clustered environment) to log to the same directory. Locking issues will occur. Use different (sub) directories for each OHI Environment and Managed Server or use the logfilesuffix to create unique file names.

- `hsl.logdirectory` is not set
  `hsl.logfilesuffix` is not set
  `hsl.<app>.logfile=/u01/log/hsl.myownname.log`

  This wil result in logfiles per HSL applications as follows:
  `/u01/log/hsl.myownname.log`

  HSL applications that have no setting for `hsl.<app>.logfile` will try to create a logfile in the WLS domain directory (which is likely to cause issues):
  `/<WLS domain directory>/hsl.<app>.log`

  This setup is not recommended.

5.2.7 hsl.<app>.loglevel /hsl.loglevel

The severity level that determines which log messages should be written is controlled by `hsl.<app>.loglevel`. If not set, this value defaults to the value of the `hsl.loglevel` property. If the global property is also not set, the default value is SEVERE.

Logging levels: SEVERE, WARNING, INFO, CONFIG, FINE, FINER or FINEST.

The following logging levels are currently used: SEVERE, WARNING, FINER and FINEST. These levels are used as follows:

Level	Usage
SEVERE	Severe is used for technical error, where it is expected that the user cannot do anything to solve the issue and that a system administrator is needed to solve the issue.
WARNING	Warning is used for functional errors, where it is expected that the user can change something and try again. This log-level is used when an error occurs with one of the following HTTP response codes: • HTTP 400: Bad Request • HTTP 401: Unauthorized • HTTP 404: Not Found • HTTP 412: Optimistic lock • HTTP 422: OHI business rule violation • HTTP 423: Locked
FINER	Finer is used for tracking and tracing. This can be used when an unexpected error occurs, and tracing information is required for debugging purposes.
FINEST	Also used for tracking and tracing, but for the low-level objects that have a very high frequency.

The logging levels FINER and FINEST should only be used for debugging.

Example:

```
hsl.loglevel=SEVERE
```

☞ WARNING: When setting the loglevel to FINER or FINEST this may lead to extensive log messages being recorded which can slow down the processing of service requests considerably. Response times measured while using such detailed log levels are clearly affected and should not be considered as representative for regular use.

☞ WARNING: for production environments, set the loglevel to SEVERE or WARNING, to avoid logging of sensitive data.

5.2.8 hsl.<app>.loglimit /hsl.loglimit

The maximum size of the log file (in bytes) is controlled by `hsl.<app>.loglimit`. If not set, this value defaults to the value of the `hsl.loglimit` property. If the global property is also not set, the default value is 1000000 (1Mb).

When the size of the log file reaches this limit, the log is rolled over to the next log file.

Note that a value of 0 means 'unlimited'.

Example:

```
hsl.loglimit=5000000
```

5.2.9 hsl.<app>.logcount /hsl.logcount

The number of log files to use in the log file rotation is controlled by `hsl.<app>.logcount`. If not set, this value defaults to the value of the `hsl.logcount` property. If the global property is also not set, the default value is 1.

A value of 1 means that only 1 log file is created, and no log rotation takes place. When the log.limit is reached, the log file is overwritten and its previous contents are lost.

Set the log.count to 2 or higher to avoid overwriting the log file once it is full.

Example

```
hsl.logcount=2
```

5.2.10 hsl.<app>.logappend /hsl.logappend

Configure `hsl.<app>.logappend` if logging can be appended to existing log files. If not set, this value defaults to the value of the `hsl.logappend` property. If the global property is also not set, the default value is true.

If false, a new log file will be created when rotating log files.

Example:

```
hsl.logappend=false
```

5.2.11 hsl.<app>.maxcontentlength /hsl.maxcontentlength

For security reasons, to provide denial of service attack protection, the maximum size on an incoming request is limited to the maxcontentlength. If the request exceeds this length, HTTP 406 Not Acceptable is returned.

If `hsl.<app>.maxcontentlength` is not set, this value defaults to the value of the `hsl.maxcontentlength` property. If `hsl.maxcontentlength` is also not set, a default of 100.000 is applied.

It is advised not to set this property to a value higher than 500.000.

An example:

```
hsl.maxcontentlength=100000
```

5.2.12 Activating changes to hsl.properties

To activate changes to hsl.properties you must restart the managed server.

5.2.13 Troubleshooting hsl.properties

Note the following if you have trouble starting up with a new hsl.properties file:

- an empty value for ANY property will block any HSL application from starting up.
  Example:
  `hsl.jndiname=`
- lines starting with '#' are ignored.
- empty lines are ignored.

- do not use whitespace characters in property lines (even at the end). Whitespace characters are tabs and spaces. Inserting whitespace characters may result in a malfunction in the operation of HSL services.

5.2.14 Keeping hsl.properties up to date

When new HSL properties are released through (patch) releases of OHI Back Office, the installation instructions will tell you to change the hsl.properties file if required. Also, an updated properties file template will be released, as described in the previous section 'Properties file templates'.

5.3 Examining the Log File

When encountering long-running HSL operations, examining the log file allows you to break down the roundtrip into different components.

Ensure that the log level for the HSL application is set to FINE.

If the log level is set to FINEST, writing log messages alone may require significant time and may account for much of the time spent in the HSL operation.

If you changed the log level, you must restart the Managed Server to activate the new log properties.

Next, look up the long-running operation in the log file. The example shows log messages of a fictitious operation:

```
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: begin getDossierRegels
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService getLanguage
FINE: getLanguage() returns: nl-NL
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: expand=all
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: limit=10000
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: number=35
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: offset=0
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.CCallContext
toJDBCObject
FINE: enter toJDBCObject
Mar 08, 2018 5:09:40 PM com.oracle.insurance.ohibo.hpo.CCallContext
toJDBCObject
FINE: leave toJDBCObject
Mar 08, 2018 5:09:41 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: Before calling PL/SQL operation
Mar 08, 2018 5:09:59 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: After calling PL/SQL operation
Mar 08, 2018 5:10:00 PM com.oracle.insurance.ohibo.exception.ExceptionUtil
handleReturnContext
FINE: start handleReturnContext
Mar 08, 2018 5:10:00 PM com.oracle.insurance.ohibo.exception.ExceptionUtil
handleReturnContext
FINE: end handleReturnContext
Mar 08, 2018 5:10:00 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: Before mapping SQL object to Java object
Mar 08, 2018 5:10:21 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: After mapping SQL object to Java object
Mar 08, 2018 5:10:21 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: http code=200
Mar 08, 2018 5:10:21 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: Before creating response
Mar 08, 2018 5:10:22 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: After creating response
```

```
Mar 08, 2018 5:10:22 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: end getDossierRegels
```

From this fragment we may derive the following information:

- Total roundtrip is about 43s (5:09:39 - 5:10:22)
- PL/SQL execution: 18s (5:09:41 - 5:09:59)
- Mapping SQL object to Java object:<1s
- Creating response with JSON string:<1s

5.3.1 Changing the log format

The default format for logging timestamps is not suitable for sub-second operations. Logging timestamps in milliseconds since 01-01-1970 is needed if you want to analyse sub-second operations.

To override the default format, create a configuration file with the following contents:

```
# override default format for timestamps in milliseconds since 01-01-1970.
java.util.logging.SimpleFormatter.format=%1$tQ %2$s%n%4$s: %5$s%6$s%n
```

You now need to activate this configuration for the managed server to which the HSL application is deployed:

- Add a line to the file $DOMAIN_HOME/bin/setUserOverrides.sh you created earlier. Add the line to the part for the HSL server:

```
JAVA_OPTIONS="-Djava.util.logging.config.file=”your_config_file” {JAVA_OPTIONS}"
```

- Restart the Managed Server.
- Call the HSL operation and check that the subsequent log messages show log messages in milliseconds since 01-01-1970

The output should now look like:

```
1520867075960 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: begin getDossierRegels
1520867075971 com.oracle.insurance.ohibo.hpo.HpoService getLanguage
FINE: getLanguage() returns: nl-NL
1520867075974 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: expand=all
1520867075974 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: limit=10000
1520867075975 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: number=11
1520867075975 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: offset=0
1520867075976 com.oracle.insurance.ohibo.hpo.CCallContext toJDBCObject
FINE: enter toJDBCObject
1520867075977 com.oracle.insurance.ohibo.hpo.CCallContext toJDBCObject
FINE: leave toJDBCObject
1520867075978 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: Before calling PL/SQL operation
1520867092723 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: After calling PL/SQL operation
1520867092728 com.oracle.insurance.ohibo.exception.ExceptionUtil
handleReturnContext
FINE: start handleReturnContext
1520867092729 com.oracle.insurance.ohibo.exception.ExceptionUtil
handleReturnContext
FINE: end handleReturnContext
1520867092730 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: Before mapping SQL object to Java object
1520867093066 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: After mapping SQL object to Java object
1520867093068 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: http code=200
1520867093072 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: Before creating response
1520867093073 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: After creating response
1520867093074 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: end getDossierRegels
```

This log output of a fictitious operation gives us the following information:

- Total roundtrip is 17114 ms (1520867093074 - 1520867075960)
- PL/SQL execution: 16745 ms (1520867092723 – 1520867075978)
- Mapping SQL object to Java object: 336 ms (1520867093066 - 1520867092730)
- Creating response with JSON string: 1 ms

6 Upgrading HSL services

Future OHI releases may include a new version of the EAR file for HSL services (HSLBOWS.ear).

To deploy a new version of an existing HSL application, follow the steps below:

- Check your web service properties file (typically hsl.properties) and implement necessary changes for your release. For information about the contents please see the previous chapter.
- Log in in the Hosted Weblogic Remote Console.
- Connect to the Admin Server of the web services Domain.
- Select the "Edit Tree".
- Select "Deployments"
- Select "App Deployments"
- If you already have a version of the SVLBOWS deployment, mark the check box in front of that deployment and delete it.
- Click on the Shopping Cart, top right, and select "Commit Changes".
- Now execute the steps described in paragraph 3.6 *(Re)deployment of the HSL Application.* Determine whether the same source path still applies (typically a new version is delivered in the $OZG_BASE/java folder of your environment but your organisation may have additional distribution methods implemented).

After this the deployment state of the web services should be Active again (be sure the Managed Server(s) is/are running, otherwise start it/them to get this result).

If not, check whether your OHI database environment and deployed version are correct, meaning that their version levels correspond with each other.

7 BASIC authentication

By default, OAUTH2 is used as the authorization method of the HSL webservices. BASIC authentication is offered as a fallback option. This form of authentication involves sending cleartext credentials over a secure encrypted connection (for the purposes of this document, HTTPS). Setting up the HSL webservices for basic authentication involves multiple parts.

The following steps are needed to set up minimal security for the HSL application:

- Set up the WebLogic security realm
- Setup a WebLogic user for accessing the HSL application
- Configure the key store

7.1 Background information

Before you can install the HSL application using BASIC authentication, you need to decide on a Security Model for the HSL Webservices. See Comparison of Security Models for Web Applications and EJBs for an introduction of the different Security Models. Your choice will depend on the security standards set by your company.

A very short summary:

- **DD Only** (Deployment Descriptors Only): choose this option if the default policies and role-mapping are desired and the WebLogic user "restuser" is allowed to execute the HSL Web Services (or "restuserbsn" for the HSL_BSNCHECK service, or "restuseragb" for the HSL_AGBCHECK service, or for other CHECK services "restuser<...>" for the HSL_<...>CHECK service).
- **Custom Roles:** choose this option if the default policies are desired, but with a customizable role. The default policies are linked to the role "restuser-role" (or "restuserbsn-role" for the HSL_BSNCHECK service, "restuser<...>-role" for the HSL_<...>CHECK service, etc.), which needs to be created using the WebLogic Remote Console. If this role is not created and assigned, all requests will be refused with HTTP 403 "Unauthorized".
- **Custom Roles and Policies**: choose this option if you want to overrule the default policy of each web service and create customizable roles, e.g. ,to limit access to certain of the HSL Webservices.

NOTE: The HSL application must be deployed with Custom Roles and Policies to use token validation.

Within the context of the 'DD Only' security model the following WebLogic users hold special significance:

- "restuser" (for all services except HSL_BSNCHECK and HSL_<...>CHECK)
- "restuserbsn" (specific for the HSL_BSNCHECK service) or
- "restuser<...>" (specific for the HSL_<...>CHECK service).

Within the context of the 'Custom Roles' security model the following WebLogic roles hold special significance:

- "restuser-role" (for all services except HSL_BSNCHECK and HSL_<...>CHECK)
- "restuserbsn-role" (specific for the HSL_BSNCHECK service) or
- "restuser<...>-role" (specific for the HSL_<...>CHECK service).

In the rest of this chapter the special 'check' services, discussed hereafter, will only be documented for the 'bsn' and '...' service. The same instructions apply to the other check services as well.

The specific users and roles for check services like the HSL_BSNCHECK and HSL_<...>CHECK (restuserbsn, restuserbsn-role, restuser<...> and restuser<...>-role) were created as an extra security measure to prevent users that gain unauthorized access to the restuser or a WebLogic user with the restuser-role, to easily crawl data from a lot of our services.

Most of the newer HSL services require a relation UUID (or another UUID that was retrieved using the relation UUID first) to retrieve data. In order to retrieve this relation UUID of an insured member it is required to supply the social security number of the relation as well as the relation's date of birth. The social security number can be retrieved using the HSL_BSNCHECK service.

In order to retrieve the relation UUID of a care provider it is required to supply the AGB-code and the provider id of the relation. The provider id can be retrieved using the HSL_AGBCHECK service.

Because the HSL_BSNCHECK and the "normal" services are authorized to different WebLogic users, a person with bad intentions needs to compromise multiple different accounts to be able to crawl all services (without causing many failing calls, which should notify a system administrator that there is suspicious activity).

7.2 Setup security realm

Create a security realm if this has not already been done (normally realm 'myrealm' will already be present).

- Log in in the Weblogic Remote Console.
- Connect to the Admin Server of your web services domain.
- Select the "Edit Tree".
- Select "Security".
- Select "Realms".
- Select "myrealm".

The security realm 'myrealm' will be used to configure the security at application level. If there are no other security realms, this will be the default security realm.

Unlesss you want to use the default "weblogic" user to make the web service requests (which is not a good idea), you need to create dedicated users in this security realm.

7.3 Setup WebLogic users for accessing the HSL application

Each operation of an HSL service requires basic authentication. This means that each call must be made as an authenticated WebLogic user.

Depending on the Security Model you choose for deployment, the WebLogic user that makes the Web Services requests must have additional authorization.

7.3.1 DDOnly

The name of the default WebLogic user to access the HSL services is defined in a preconfigured deployment descriptor in the EAR file. This default name is 'restuser' for all services except HSL_BSNCHECK and HSL_<...>CHECK for which the names are 'restuserbsn' and 'restuser<...>'.

If the HSL application is deployed with the security model DD Only, the WebLogic user 'restuser' must be created and used as authentication when invoking the HSL service operations (or 'restuserbsn' for the HSL_BSNCHECK service or 'restuser<...>' for the HSL_<...>CHECK service).

To set up the WebLogic users 'restuser', 'restuserbsn' and 'restuser<...>':

- Log in in the Weblogic Remote Console.

- Connect to the Admin Server of your web services domain.
- Select the “Security Data Tree”.
- Select “Realms”.
- Select “myrealm”.
- Select “Authentication Providers”
- Select “DefaultAuthenticator”
- Select “Users”
- Repeat for the usernames restuser’, ‘restuserbsn’ and ‘restuser<...>’:
  - In the screen on the right, click on “New”.
  - Enter values for “Name”, “Description” and “Password”.
  - Click on “Create”

7.3.2 CustomRoles

If you choose Security Model “Custom Roles”, create and/or customize the predefined roles “restuser-role”, “restuserbsn-role” and “restuser<...>-role” now. See Securing Resources Using Roles and Policies for Oracle WebLogic Server for more information on how to setup a secure role.

For each of the roles, you need to create a Group, assign the correct role to it via a Policy and add WebLogic users to that Group. Make sure you create a separate Group for each Role.

Create the Groups:

- Log in in the Weblogic Remote Console.
- Connect to the Admin Server of your web services domain.
- Select the “Security Data Tree”.
- Select “Realms”.
- Select “myrealm”.
- Select “Authentication Providers”.
- Select “DefaultAuthenticator”.
- Select “Groups”.
- For the groups, e.g., HSLCRUsersGroup, HSLCRBSNUsersGroup, HSLCR<…>UsersGroup, execute:
  - In the screen on the right, click on “New”.
  - Enter values for “Name” and “Description”.
  - Click on “Create”.

Create the Roles and Policies:

- Select the “Security Data Tree”.
- Select “Realms”.
- Select “myrealm”.
- Select “Role Mappers”.
- Select “XACMLRoleMapper”.
- Select “Global”.
- Select “Roles”.

- For the predefined roles “restuser-role”, “restuserbsn-role” “restuser<...>-role”
  - In the screen on the right, click on “New”.
  - Enter the value for “Name”.
  - Click on “Create”.
  - Click on the Role.
  - In the next screen, in the tab “Policy”, click on “Add Condition”.
  - In the popup, in “Predicate List”, select “Group”.
  - In “Group Argument Name” fill in the correct Group Name.
  - Click on “Save”.

Now add existing or new users to the Group to authorize that user for HSL, using the Security Model “CustomRoles“. To create a new user, see the step above, for DDOnly.

To add a user to a Group, first navigate to the detail page of the user:

- Select the “Security Data Tree”.
- Select “Realms”.
- Select “myrealm”.
- Select “Authentication Providers”.
- Select “DefaultAuthenticator”.
- Select “Users”.
- Click on the user.
- In the screen on the right, click on the tab “Membership” and move the desired Group to the right.
- Click on “Save”.

☞ WARNING: Do not make the same user member of the different Groups. This defeats the purpose of the separate authentication and authorisation for the HSL_<…>CHECK web services.

7.3.3 CustomRolesAndPolicies

If you do not want to authenticate the HSL application using the predefined weblogic users ‘restuser’, ‘restuserbsn’ and ‘restuser<...>’ and/or the predefined roles ‘restuser-role’, ‘restuserbsn-role’ and ‘restuser<...>-role’, and instead you want to change the default permissions for individual Web Services or Web Service methods in the HSL application, you can choose to deploy using ‘Custom Roles and Policies’.

☞ **Attention:** deploying HSL with ‘Custom Roles and Policies’ but not adding your own roles and policies will allow any authenticated WebLogic user to execute the application.

7.3.3.1 *Using a custom security policy for a deployed application*

The WebLogic console allows the administrator to specify a custom security policy for HSL applications deployed using ‘Custom Roles and Policies’.
For example, a custom security policy can be used:

- to limit access to a specified list of named WebLogic users; or

- to limit access to a group of WebLogic users; or
- to limit access to WebLogic users with a specific role; or
- to limit access to a specific web module; or
- to limit access to a specific web service operation inside a module (see Appendix E for an example)
- or a combination of the above.

7.3.4 Set user lockout

While setting up HSL services for testing you may want to disable user lockout. In a production environment you should enable user lockout to discourage fraudulent use.

- In the "Edit Tree", navigate to the Security Realm "myrealm" (see above).
- Open the "User Lockout" tab.
- Set "Lockout Enabled" to On.
- Choose values for the other fields.

7.4 Testing with SoapUI

SoapUI is a tool for testing web services which can be downloaded from http://www.soapui.org.

It is especially useful for functional testing of the HSL application.

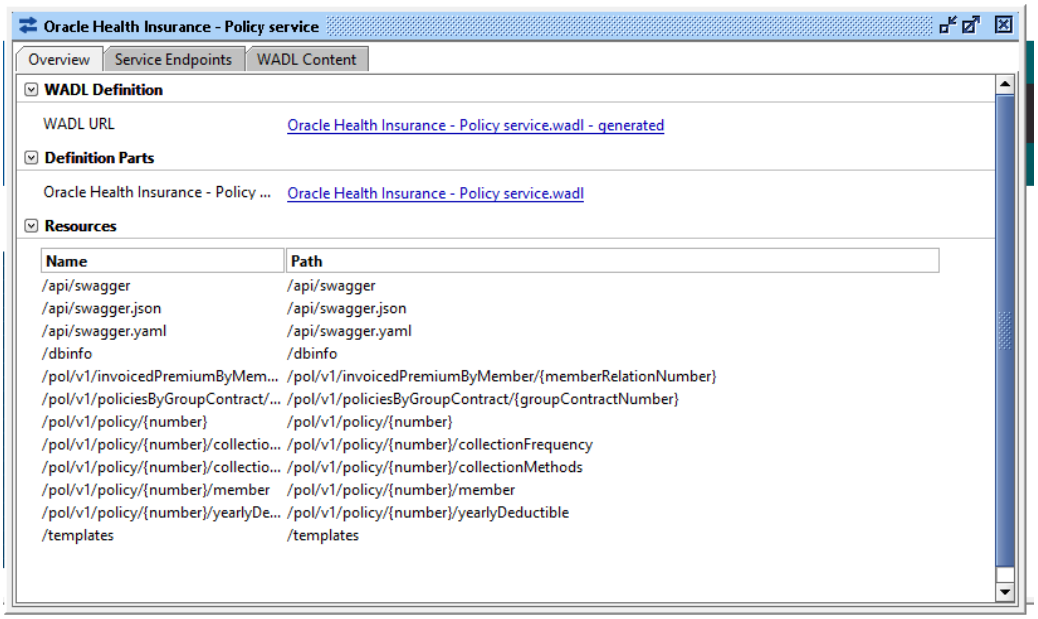
SoapUI is not only useful for testing the functionality of the HSL services, but it is also suitable for testing their security settings.

You have to test and use the OHI Web Services with a client that uses TLS 1.2 or higher.

7.4.1 Create REST project and import Swagger definiton

- Follow the instructions in paragraph 4.4 *Get Runtime Swagger Definition* to retrieve the runtime Swagger definition from the HSL application and save the output to a file (for example 'saved_swagger.json')
- Create a new REST project (empty value for URL)
- Choose 'Project > Import Swagger' and select the saved Swagger definition.

The operations of the HSL application are now discovered:



You can now create requests for the operations provided by this HSL application.

NOTE: Upon release of this document there was no SoapUI version available that provided the 'Import Swagger' functionality for OpenApi 3.1 documents. Resources/testcases and mockservers can't be automatically generated but can be entered manually.

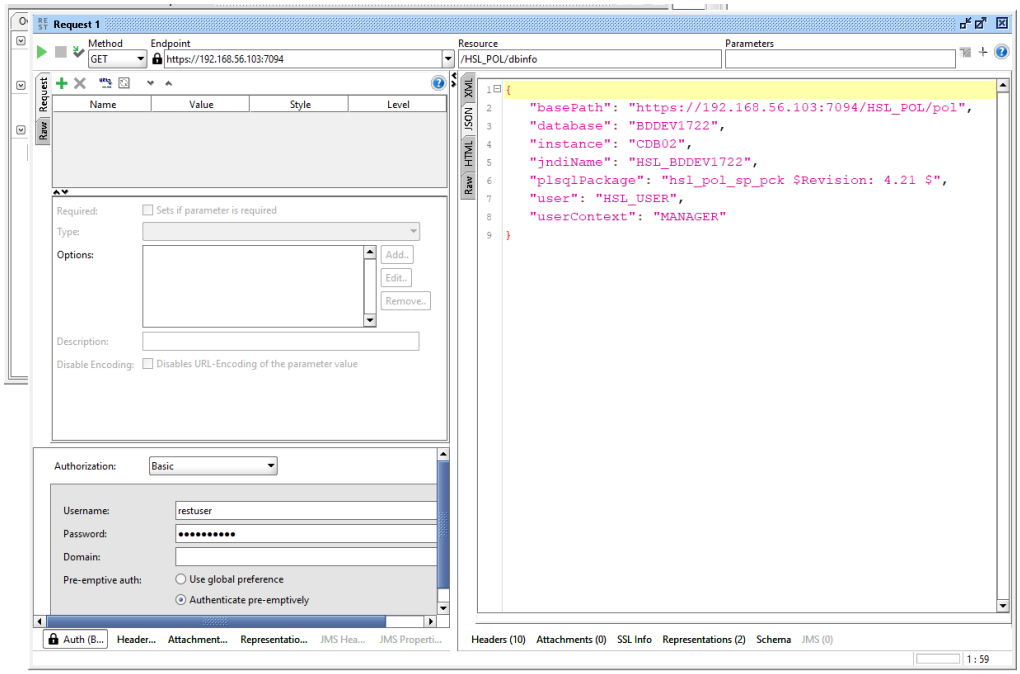
7.4.2 Create a request

Once you have imported the Swagger definition you may create a request for each of the operations.

In the example below we create a request for the getDatabaseInfo operation:

- Double-click on 'Request 1' of the requested operation
  (in our case /dbinfo > getDatabaseInfo).
- Set the endpoint for the request to `https://<server>:<port>`
  For example https://127.0.0.1:7094.
- Select 'Headers' and add a HTTP request header with Header value 'Accept' and with Value value 'application/json'.
- Add other HTTP request headers as required (not needed for this example)
- Select 'Auth' to add Basic Authentication for the WLS user.
  If you deployed with the 'DD Only' deployment model the WLS user should be 'restuser' (or 'restuserbsn' for the HSL_BSNCHECK service or 'restuser<...>' for the HSL_<...>CHECK service).
- Set 'pre-emptive authentication'.
- Run the request.

The request window should now look like this:



7.5 Generating a WADL file

A WADL (Web Application Description Language) file may be required by Oracle Service Bus or other middleware to describe your HSL application.

The current HSL applications cannot be used to generate WADL files directly.

However, a WADL file can be easily generated from the online Swagger definition using SoapUI.

This involves the following steps:

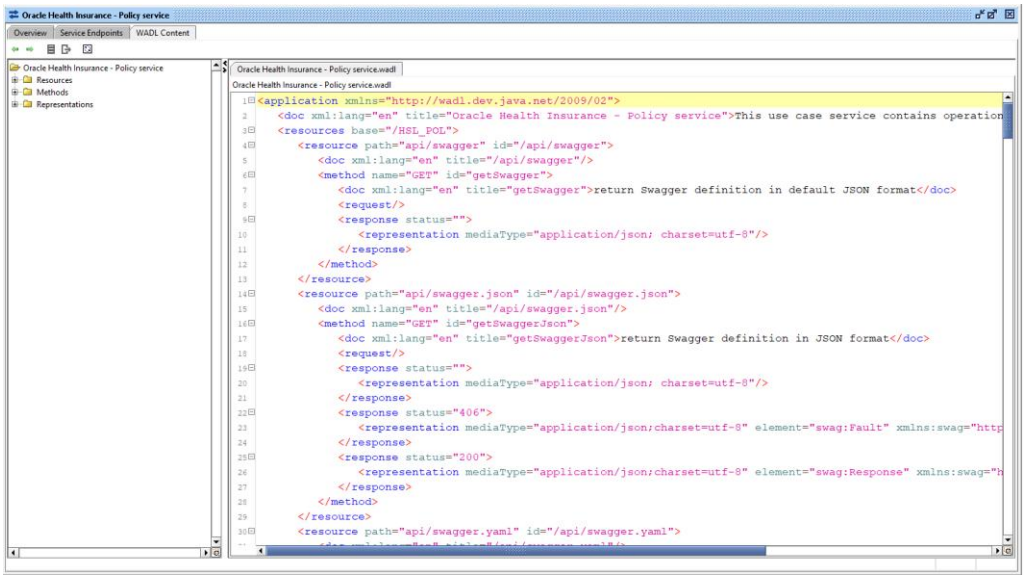
- Create a REST project in SoapUI for your HSL application
- Open the Service Viewer for the REST project
- Export WADL from your REST project

7.5.1 Create a REST project in SoapUI for your HSL application

Follow the instructions in 7.4 *Testing with SoapUI* to set up SoapUI for testing with your HSL application.

7.5.2 Open the Service Viewer for the REST Project

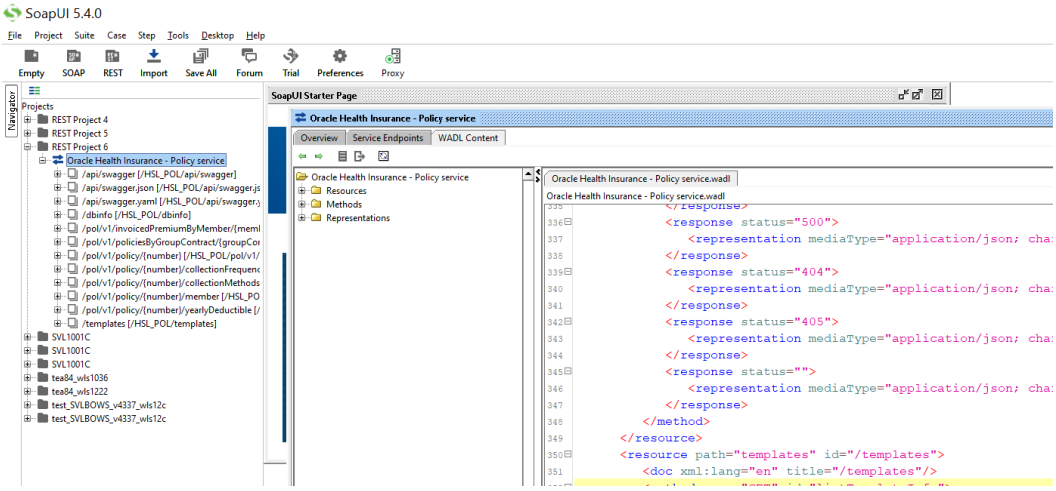
Click on the 'WADL Content' to see the WADL description.
Your screen may look like this:



7.5.3 Export WADL from your REST project

Save the buffer to a WADL file.

Alternatively, you may right-click on the service within the REST project (highlighted in the screen shot below):



And select 'Export WADL' to create the WADL for this application.

8 Appendix B – Service Information

The following URI provides version information about a running HSL application:

`https://`**`server:port/application`**`/dbinfo`

For example:

https://localhost:7002/HSL_POL/dbinfo

This will return a JSON object like below:

```
{
  "basePath":"https://localhost:7002/HSL_POL/pol",
  "database":"OHI_PRD",
  "instance":"CDB19BO",
  "jndiName":"jdbc//DS_HSL_BTSRCNL",
  "plsqlPackage":"hsl_pol_sp_pck $Revision: 45869 $",
  "user":"HSL_USER",
  "userContext":"HSL_FUNC_USER"
}
```

Information:

- basePath
  Format: `https://`**`server:port/application/context`**
  This is the base URI for all operations in this service.
- database
  The name of the database associated with the current database connection
- instance
  Instance name of the database associated with the current database connection.
- jndiName
  The JNDI name of the database connection (specified in the hsl.properties file)
- plsqlPackage
  The PL/SQL package which implements the operations of the HSL service.
  The revision number refers to the version of the PL/SQL package.
- user
  The database account used to log on to the database by the data source
- userContext
  The OHI employee on whose behalf service operations are performed, as specified in the hsl.properties file.

9 Appendix C – Removing a WLS domain

In case you want to restructure your environment or recreate a domain you can remove an existing domain.

In order to do this, make sure all servers for the domain are stopped and make sure there is no Node Manager process running which 'guards' this domain.

Next perform the following actions:

- ✓ Completely remove your domain directory including all contents.
- ✓ Remove any reference in start and stop scripts to this domain.
- ✓ Remove, if present, the domain from the file <WebLogic home>\oracle_common\common\nodemanager\nodemanager.domains.
- ✓ Remove the domain from the file domain-registry.xml which is located in the Middleware home folder ($MW_HOME).

For more information, please reference the standard WebLogic documentation.

10 Appendix D – HSL C2B services – deployment points of attention

Starting with OHI Back Office release 10.17.2.2.0, a set of web service operations has been made available in the HTTP Service Layer to replace the 'old' Connect to Back Office (C2B) services that modified policies and relations. These SOAP envelope-based services were first delivered in 2006 and have been used for a long time by many OHI customers to send all kinds of policy and relation modification requests to OHI Back Office.

However, as years went by, these services were kept up to date from a functional perspective but received no technical updates. From a technical and security perspective, they could no longer be supported, because they were based on by now obsolete Java libraries and did not support any form of security.

In 2015 it was announced the C2B services would become obsolete and be replaced by more use case driven HSL services. This intention was frustrated because customers could not agree on a uniform set of commonly accepted service definitions to replace the C2B services. It also became clear that OHI customers could not easily say goodbye to the functionally well-known behaviour of the old C2B services. The impact of implementing a new set of functionally different services was perceived as difficult and as a change that would have a large impact on the business.

So, as more strict security requirements demanded phasing out of the old C2B services, time ran out to develop completely new C2B use case services. This resulted in a compromise, where OHI Development would implement a new set of C2B business operations according to the technology used for the existing, modern HTTP Service Layer services, but their functional behaviour would be kept identical, as much as possible.

As a result, the C2B 'Use Case services' clearly differ from the other Use Case services that were developed from scratch. The C2B Use Case services should be seen as 'classic C2B operations' being implemented with more modern technology but with a minimum of effort. No new functionality has been added and no new requirements were set, to keep the implementation costs low. It was deemed better to invest later on in new functionality.

This clearly influences the way the C2B Use Case services should be deployed and used. This paragraph focuses on these aspects. Please bear in mind that the old C2B services descend from an OHI era in which it was agreed documentation was not part of the contractual obligations and usage knowledge was transferred as part of consulting activities.

10.1 HSL_C2B deployment aspects

The 'old' C2B operations were made available in 2 different versions:

- A synchronous deployment
- An asynchronous deployment

The asynchronous deployment version required a hard-coded JMS queue and contained a Message Driven Bean that dequeued the message and called the service. This bean could only be deployed to one WebLogic Managed Server and as such was not scalable. Any submitted messages were processed in First in First out (FIFO) order. This had a fortunate side-effect: 2 changes on the same policy were always processed in the correct order provided they were put on the queue in the correct order.

In the new C2B implementation the asynchronous deployment is no longer supported by code in the application. Modern queueing and service implementations offer this functionality out of the box and as such only a synchronous implementation can be deployed. Queues with delayed asynchronous processing should be implemented through standard middleware functionality and put in front of the synchronous implementation.

This has deployment implications:

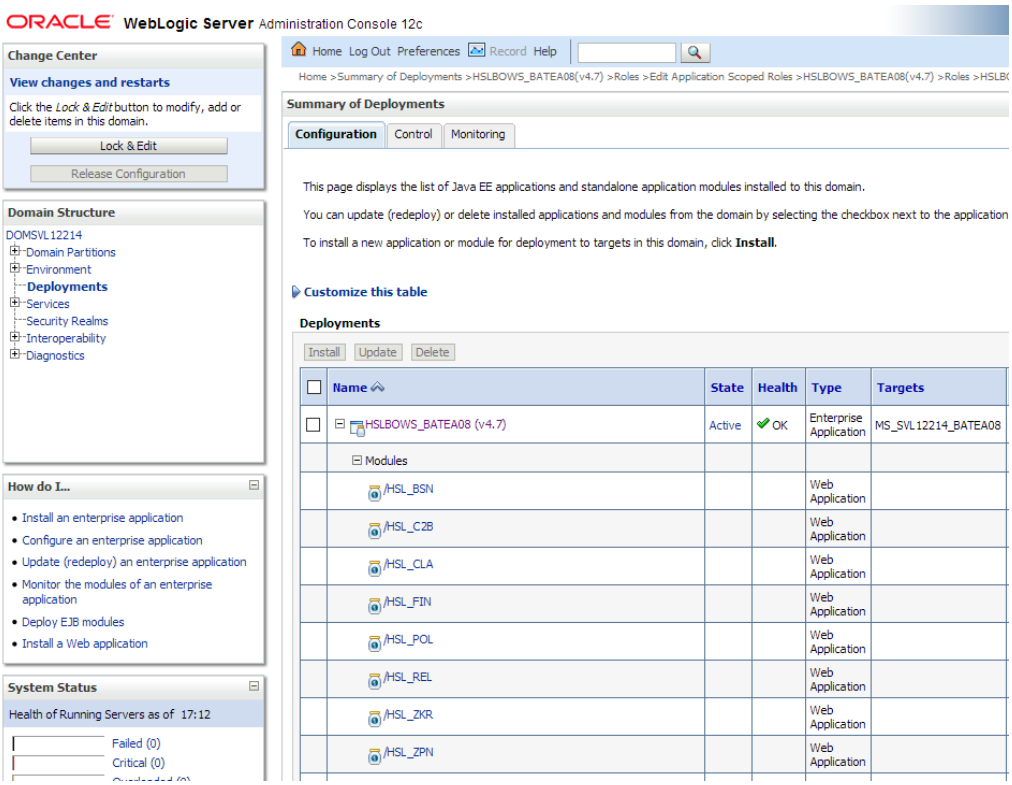
- Scalability is still relatively poor as it is expected messages are processed in FIFO order and it is assumed that changes on the same policy are offered and processed non-concurrently. This means the number of processing threads must be kept to a minimum, preferably one or only a few, provided that dequeuing and processing takes care of the order of processing. A C2B operation for adding a new policy and possibly ending an existing one (a typical functionality from the old C2B SOAP Services) is a relatively resource consuming operation, in which up to a several hundred standard and custom 'policy checks' and other business rules are fired and during which a full copy of the policy to be approved may be created, changed and dropped. This may lead to a response time of several seconds even on modern infrastructure. This response time may increase when synchronous policy processing calls also lead to synchronous webservice callouts towards external providers, like 'BRP' (formerly 'GBA') and 'VECOZO'.
- Large amounts of stored messages that may have piled up during a maintenance downtime of OHI Back Office should not be processed through a burst of many parallel threads but must be digested in an ordered manner of (at most) a few parallel processes.

11 Appendix E – Limiting access to a specific application inside a module

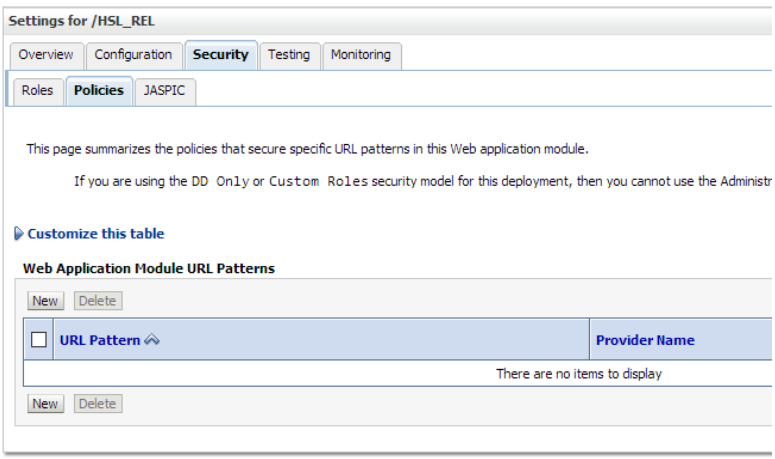
The WebLogic Remote Console currently does not appear to support the display or editing of Web Service Policies. For now, scripting with WSLT appears to be the only option.

When HSL applications are deployed using 'Custom Roles and Policies' additional options are available to limit the access to a web application or operation. WebLogic can be configured to limit access through the use of a security policy. To give an idea as to what configuration might involve, an example is shown below.

First, expand the HSLBOWS EAR file, revealing the individual HSL applications.

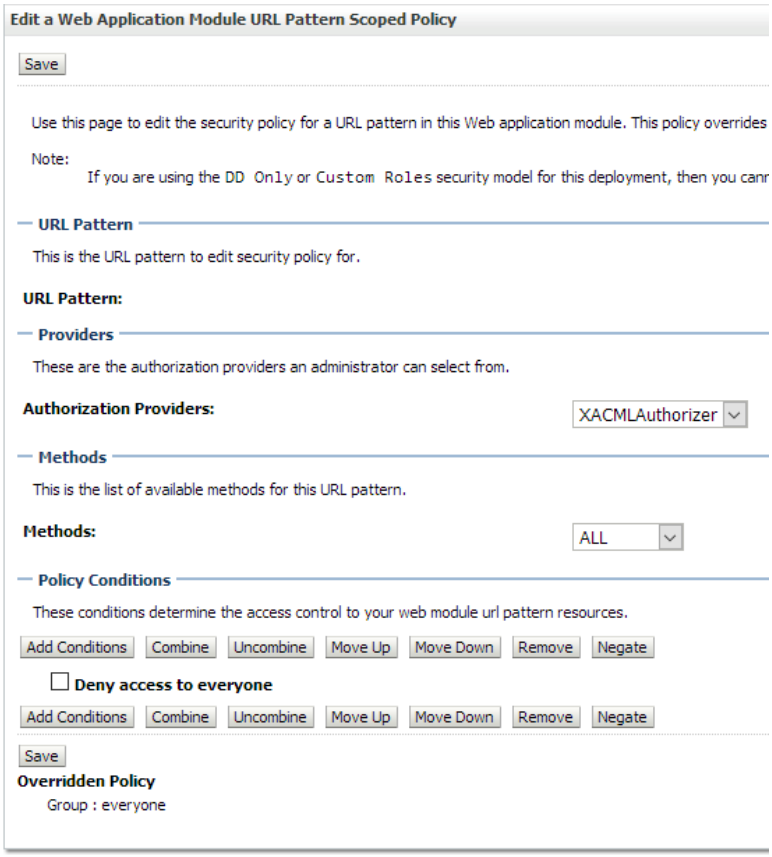


Now to fully disable the HSL_REL service, enter its configuration page by clicking on the link. Click on the tab 'Security' in the resulting page. Finally enter the 'Policies' sub-tab.



As we want to disable access to this specific application in its entirety, adding a policy with an URL pattern of '*' is enough. After the creation of the policy, rules and conditions need to be defined that will be enforced through the policy. Enter the policy's configuration page through clicking on the link named after the URL pattern of the newly created policy.

A number of options are available on the following page to narrow, limit or deny access to a web application. Add a condition and choose the predicate 'Deny access to everyone' from the predicate list. The added condition becomes active immediately after saving the policy at which point all access to the HSL_REL web application is denied.



Note that the created policies survive an update of the ear-file but not a complete reinstall (deletion and subsequent installation of the ear-file). This allows for setting up policies as a one-time configuration step and having all future updates adhere to the same (strict) set of policies.

12 Appendix F – Eclipse Jersey Monitoring and Diagnostics

The HSL and PSL webservices use Jersey to expose data from a OHI Back Office application through a RESTful interface. Jersey provides functionality for monitoring JAX-RS/Jersey applications. Application monitoring is useful when you need to identify the performance hot-spots in a JAX-RS application or observe execution statistics of particular resources. The information collected in this manner is accessible through multiple applications and/or interfaces. See "Monitoring RESTful Web Services and Clients" for more information.

WARNING: RESTful webservice monitoring is enabled by default by WLS. In some cases, this may result in increased memory consumption. You can disable the monitoring feature at the WLS domain level, and at the application level. See Disabling RESTful Web Service Application Monitoring.