Gateway Guide

/ ForgeRock Identity Gateway 6.5

Latest update: 6.5.4

Paul Bryan
Mark Craig
Jamie Nelson
Guillaume Sauthier
Joanne Henry

ForgeRock AS
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Copyright © 2011-2021 ForgeRock AS.

Abstract

Instructions for installing and configuring ForgeRock® Identity Gateway.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.

To view a copy of this license, visit https://creativecommons.org/licenses/by-nc-nd/3.0/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

ForgeRock® and ForgeRock Identity Platform™ are trademarks of ForgeRock Inc. or its subsidiaries in the U.S. and in other countries. Trademarks are the property of their respective owners.

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

DejaVu Fonts

Bitstream Vera Fonts Copyright

Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved. Bitstream Vera is a trademark of Bitstream, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Bitstream" or the word "Vera".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Bitstream Vera" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL BITSTREAM OR THE GNOME FOUNDATION BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the names of Gnome, the Gnome Foundation, and Bitstream Inc., shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from the Gnome Foundation or Bitstream Inc., respectively. For further information, contact: fonts at gnome dot org.

Arev Fonts Copyright

Copyright (c) 2006 by Tavmjong Bah. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the modifications to the Bitstream Vera Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Tavmjong Bah" or the word "Arev".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Tavmjong Bah Arev" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL TAVMJONG BAH BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the name of Tavmjong Bah shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from Tavmjong Bah. For further information, contact: tavmjong @ free . fr.

FontAwesome Copyright

Copyright (c) 2017 by Dave Gandy, http://fontawesome.io.

This Font Software is licensed under the SIL Open Font License, Version 1.1. See https://opensource.org/licenses/OFL-1.1.

Table of Contents

Preface vii
- 1. About This Guide vii
- 2. Formatting Conventions viii
- 3. Accessing Documentation Online viii
- 4. Using the ForgeRock.org Site ix
- 5. Getting Support and Contacting ForgeRock ix

1. About IG 1
- 1.1. About IG 1
- 1.2. IG Object Model 3
- 1.3. Configuration Directories and Files 3
- 1.4. Routing and Routes 3
- 1.5. Handlers, Filters, and Chains 4
- 1.6. Configuration Objects 8
- 1.7. Decorators 9
- 1.8. Configuration Parameters Declared as Property Variables 9
- 1.9. Using Comments in IG Configuration Files 9
- 1.10. Understanding IG APIs With API Descriptors 10

2. Installation in Detail 13
- 2.1. Configuring Deployment Containers 13
- 2.2. Preparing the Network 23
- 2.3. Installing IG 23
- 2.4. Changing the Default Location of the Configuration Folders 25
- 2.5. Preparing For Load Balancing and Failover 25
- 2.6. Configuring IG For HTTPS (Client-Side) 27
- 2.7. Setting Up Keys For JWT Encryption 29
- 2.8. Making the Configuration Immutable 31
- 2.9. Restricting Access to Studio in Development Mode 33
- 2.10. Setting Up AM 35

3. Getting Login Credentials From Data Sources 37
- 3.1. Before You Start 37
- 3.2. Log in With Credentials From a File 37
- 3.3. Log in With Credentials From a Database 40

4. Getting Login Credentials From AM 45
- 4.1. About Password Capture and Replay 45
- 4.2. Setting Up the Example 47

5. Single Sign-On and Cross-Domain Single Sign-On 52
- 5.1. About SSO Using the SingleSignOnFilter 52
- 5.2. About CDSSO Using the CrossDomainSingleSignOnFilter 54
- 5.3. Setting Up CDSSO 56
- 5.4. Using WebSocket Notifications to Evict the Session Cache 59

6. Enforcing Policy Decisions From AM 61
- 6.1. About IG As a PEP With AM As PDP 61
- 6.2. Enforcing AM Policy Decisions In the Same Domain 62
- 6.3. Enforcing AM Policy Decisions Cross-Domain 66

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

6.4. Using WebSocket Notifications to Evict the Policy Cache 71
7. Hardening Authorization With Advice From AM 72
7.1. Stepping Up the Authentication Level for a Session 72
7.2. Increasing Authorization for a Single Transaction 76
8. Acting As a SAML 2.0 Service Provider 80
8.1. About SAML 2.0 SSO and Federation 80
8.2. About SP-Initiated SSO 81
8.3. About IDP-Initiated SSO 81
8.4. Setting Up IDP-initiated SSO and SP-initiated SSO 82
8.5. Using a Non-Transient NameID Format 88
8.6. Federation Configuration Files 89
9. Acting as an OAuth 2.0 Resource Server 95
9.1. About IG As an OAuth 2.0 Resource Server 95
9.2. Validating Access-Tokens Through the Token Info Endpoint 97
9.3. Validating Access-Tokens Through the Introspection Endpoint 102
9.4. Validating Stateless Access_Tokens With the StatelessAccessTokenResolver 107
9.5. Validating Access_Tokens Obtained Through mTLS 118
9.6. Using the OAuth 2.0 Context to Log In To the Sample Application 128
10. Acting As an OAuth 2.0 Client or OpenID Connect Relying Party 132
10.1. About IG As an OAuth 2.0 Client 132
10.2. About IG As an OpenID Connect 1.0 Relying Party 132
10.3. Installation Overview 133
10.4. Setting Up AM for OpenID Connect 134
10.5. Setting Up IG As a Relying Party 135
10.6. Testing the Setup 138
10.7. Authenticating Automatically to the Sample Application 138
10.8. Using OpenID Connect Discovery and Dynamic Client Registration 139
11. Transforming OpenID Connect ID Tokens Into SAML Assertions 144
11.1. About Token Transformation 144
11.2. Setting Up AM for Token Transformation 146
11.3. Setting Up IG for Token Transformation 148
11.4. Testing the Setup 153
12. Supporting UMA Resource Servers 155
12.1. About IG As an UMA Resource Server 155
12.2. Sharing and Accessing Protected Resources 156
12.3. Limitations of This Implementation 158
12.4. Preparing the Tutorial 159
12.5. Setting Up AM As an Authorization Server 159
12.6. Setting Up IG As an UMA Resource Server 163
12.7. Test the Configuration 166
12.8. Editing the Example to Match Custom Settings 167
12.9. Understanding the UMA API With an API Descriptor 168
13. Configuring Routers and Routes 169
13.1. Configuring Routers 169
13.2. Configuring Routes 170
13.3. Creating and Editing Routes Through Common REST 172
13.4. Creating Routes Through Studio 175

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- 13.5. Preventing the Reload of Routes ... 175
- 13.6. Accessing Reserved Routes ... 176
14. Proxying WebSocket Traffic ... 177
  - 14.1. Configuring IG to Proxy WebSocket Traffic ... 179
15. Throttling the Rate of Requests to Protected Applications ... 183
  - 15.1. Configuring a Simple Throttling Filter ... 184
  - 15.2. Configuring a Mapped Throttling Filter ... 186
  - 15.3. Configuring a Scriptable Throttling Filter ... 194
16. Configuration Templates ... 199
  - 16.1. Proxy and Capture ... 199
  - 16.2. Simple Login Form ... 200
  - 16.3. Login Form With Cookie From Login Page ... 202
  - 16.4. Login Form With Password Replay and Cookie Filters ... 204
  - 16.5. Login Which Requires a Hidden Value From the Login Page ... 206
  - 16.6. HTTP and HTTPS Application ... 208
  - 16.7. AM Integration With Headers ... 209
17. Extending IG ... 212
  - 17.1. About Scripting ... 212
  - 17.2. Scripting Dispatch ... 213
  - 17.3. Scripting HTTP Basic Authentication ... 216
  - 17.4. Scripting Authentication to LDAP and LDAPS-enabled Servers ... 218
  - 17.5. Scripting SQL Queries ... 222
  - 17.6. Developing Custom Extensions ... 225
18. Auditing ... 232
  - 18.1. Recording Audit Events in CSV ... 233
  - 18.2. Recording Audit Events in Elasticsearch ... 234
  - 18.3. Recording Audit Events in JMS ... 236
  - 18.4. Recording Audit Events in JSON ... 239
  - 18.5. Recording Audit Events to Standard Output ... 242
  - 18.6. Recording Audit Events in Splunk ... 243
19. Monitoring ... 247
  - 19.1. Prometheus Scrape Endpoint ... 247
  - 19.2. Common REST Monitoring Endpoint ... 248
  - 19.3. Protecting the Monitoring Endpoints ... 250
20. Logging Events ... 252
  - 20.1. Default Logging Behavior ... 252
  - 20.2. Reference Logback Configuration ... 252
  - 20.3. Capturing Log Messages for Routes ... 254
  - 20.4. Changing the Logging Behavior ... 255
21. Tuning Performance ... 258
  - 21.1. Defining Performance Requirements and Constraints ... 259
  - 21.2. Tuning IG ... 260
  - 21.3. Tuning the ClientHandler/ReverseProxyHandler ... 263
  - 21.4. Tuning IG's Tomcat Container ... 264
  - 21.5. Tuning IG's JVM ... 264
22. Troubleshooting ... 266
  - 22.1. Timeout When Downloading Large Files ... 266

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

22.2. Requests Redirected to Access Management Instead of to the Resource ... 266
22.3. Troubleshooting the UMA Example ... 266
22.4. Can't Deploy Routes in Studio ... 267
22.5. Object not found in heap ... 268
22.6. Extra or missing character / invalid JSON ... 268
22.7. The values in the flat file are incorrect ... 268
22.8. Problem accessing URL ... 268
22.9. StaticResponseHandler results in a blank page ... 269
22.10. IG is not logging users in ... 269
22.11. Read timed out error when sending a request ... 269
22.12. IG does not use new route configuration ... 269
22.13. Make IG skip a route ... 270
A. SAML 2.0 and Multiple Applications ... 272
A.1. Installation Overview ... 272
A.2. Preparing the Network ... 273
A.3. Configuring the Circle of Trust ... 273
A.4. Configuring the Service Provider for Application One ... 273
A.5. Configuring the Service Provider for Application Two ... 279
A.6. Importing Service Provider Configurations Into AM ... 279
A.7. Preparing IG Configurations ... 279
A.8. Test the Configuration ... 283

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Preface

ForgeRock Identity Platform™ serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

1. About This Guide

IG integrates web applications, APIs, and microservices with the ForgeRock Identity Platform, without modifying the application or the container where they run. Based on reverse proxy architecture, it enforces security and access control in conjunction with the Access Management modules.

This guide is for access management designers and administrators who develop, build, deploy, and maintain IG for their organizations. It helps you to get started quickly, and learn more as you progress through the guide.

This guide assumes basic familiarity with the following topics:

- Hypertext Transfer Protocol (HTTP), including how clients and servers exchange messages, and the role that a reverse proxy (gateway) plays
- JavaScript Object Notation (JSON), which is the format for IG configuration files
- Managing services on operating systems and application servers
- Configuring network connections on operating systems
- Managing Public Key Infrastructure (PKI) used to establish HTTPS connections
- Access management for web applications

Depending on the features you use, you should also have basic familiarity with the following topics:

- Lightweight Directory Access Protocol (LDAP) if you use IG with LDAP directory services
- Structured Query Language (SQL) if you use IG with relational databases
- Configuring AM if you use password capture and replay, or if you plan to follow the OAuth 2.0 or SAML 2.0 tutorials

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- The Groovy programming language if you plan to extend IG with scripts
- The Java programming language if you plan to extend IG with plugins, and Apache Maven for building plugins

2. Formatting Conventions

Most examples in the documentation are created in GNU/Linux or Mac OS X operating environments. If distinctions are necessary between operating environments, examples are labeled with the operating environment name in parentheses. To avoid repetition file system directory names are often given only in UNIX format as in `/path/to/server`, even if the text applies to `C:\path\to\server` as well.

Absolute path names usually begin with the placeholder `/path/to/`. This path might translate to `/opt/`, `C:\Program Files\`, or somewhere else on your system.

Command-line, terminal sessions are formatted as follows:

```
$ echo $JAVA_HOME
/path/to/jdk
```

Command output is sometimes formatted for narrower, more readable output even though formatting parameters are not shown in the command.

Program listings are formatted as follows:

```
class Test {
    public static void main(String [] args)  {
        System.out.println("This is a program listing.");
    }
}
```

3. Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

  While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

4. Using the ForgeRock.org Site

The ForgeRock.org site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

If you are a *ForgeRock customer*, raise a support ticket instead of using the forums. ForgeRock support professionals will get in touch to help you.

5. Getting Support and Contacting ForgeRock

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details on ForgeRock's support offering, including support plans and service level agreements (SLAs), visit https://www.forgerock.com/support.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 1
About IG

This chapter sets out the essentials of using IG, including:

- What problems IG solves and where it fits in your deployment
- How IG acts on HTTP requests and responses
- How the configuration files for IG are organized
- The roles played by routes, filters, handlers, and chains, which are the building blocks of an IG configuration

1.1. About IG

Most organizations have valuable existing services that are not easily integrated into newer architectures. These existing services cannot often be changed. Many client applications cannot communicate as they lack a gateway to bridge the gap. "Missing Gateway" illustrates one example of a missing gateway.

Missing Gateway

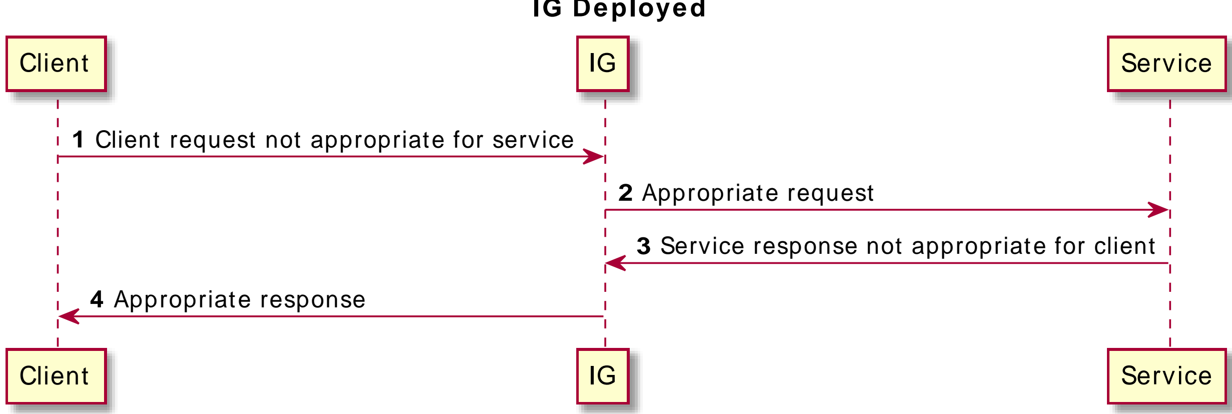
Missing Gateway

Client
Service
1 Client request not appropriate for service
2 Service response not appropriate for client
Client
Service

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

IG works as an HTTP gateway, based on reverse proxy architecture. IG is deployed on a network so it can intercept both client requests and server responses. "IG Deployed" illustrates a IG deployment.

IG Deployed



Clients interact with protected servers through IG. IG can be configured to add new capabilities to existing services without affecting current clients or servers.

You can add the following features to your solution by using IG:

- Access management integration
- Application and API security
- Credential replay
- OAuth 2.0 support
- OpenID Connect 1.0 support
- Network traffic control
- Proxy with request and response capture
- Request and response rewriting
- SAML 2.0 federation support
- Single sign-on (SSO)

IG supports these capabilities as out of the box configuration options. Once you understand the essential concepts covered in this chapter, try the additional instructions in this guide to use IG to add other features.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

1.2. IG Object Model

IG handles HTTP requests and responses in user-defined chains, making it possible to manage and to monitor processing at any point in a chain. The IG object model provides both access to the requests and responses that pass through each chain, and also context information associated with each request.

Contexts provide information about the client making the request, the session, the authentication or authorization identity of the principal, and any other state information associated with the request. Contexts provide a means to access state information throughout the duration of the HTTP session between the client and protected application, including when this involves interaction with additional services.

1.3. Configuration Directories and Files

By default, IG configuration files are located under `$HOME/.openig` on Linux, macOS, and UNIX systems, and `%appdata%\OpenIG` on Windows systems. For information about how to change the default locations , see "Changing the Default Location of the Configuration Folders".

IG uses the following configuration directories:

- `$HOME/.openig/config`, `%appdata%\OpenIG\config`

  IG administration and gateway configuration files. For information, see AdminHttpApplication(5) in the *Configuration Reference* and GatewayHttpApplication(5) in the *Configuration Reference*.

- `$HOME/.openig/config/routes`, `%appdata%\OpenIG\config\routes`

  IG route configuration files. For more information see "*Configuring Routers and Routes*".

- `$HOME/.openig/SAML`, `%appdata%\OpenIG\SAML`

  IG SAML 2.0 configuration files. For more information see "*Acting As a SAML 2.0 Service Provider*".

- `$HOME/.openig/scripts/groovy`, `%appdata%\OpenIG\scripts\groovy`

  IG script files, for Groovy scripted filters and handlers. For more information see "*Extending IG*".

- `$HOME/.openig/tmp`, `%appdata%\OpenIG\tmp`

  IG temporary files. This location can be used for temporary storage.

1.4. Routing and Routes

Routers are handlers that perform the following tasks:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Define the routes directory and loads routes into the IG configuration.
- Depending on the scanning interval, periodically scan the routes directory and updates the IG configuration when routes are added, removed, or changed.
- Route requests to the first route in the IG configuration whose condition is satisfied.

Routes are configuration files that you add to IG to manage requests. They are flat files in JSON format. You can add routes in the following ways:

- Manually into the filesystem.
- Through Common REST commands. For information, see " Creating and Editing Routes Through Common REST ".
- Through Studio. For information, see "*Configuring Routes With Studio*" in the *Getting Started Guide*.

Every route must call a handler to process the request and produce a response to a request.

When a route has a condition, it can handle only requests that meet the condition. When a route has no condition, it can handle any request.

Routes inherit settings from their parent configurations. This means that you can configure global objects in the `config.json` heap, for example, and then reference the objects by name in any other IG configuration.

For examples of route configurations see "*Configuring Routers and Routes*". For information about the parameters for routers and routes, see Router(5) in the *Configuration Reference* and Route(5) in the *Configuration Reference*.

1.5. Handlers, Filters, and Chains

Handlers and filters are chained together to modify a request, the response, or the context:

- *Handler*: Either delegates to another handler, or produces a response.

  One way to produce a response is to send a request to and receive a response from an external service. In this case, IG acts as a client of the service, often on behalf of the client whose request initiated the request.

  Another way to produce a response is to build a response either statically or based on something in the context. In this case, IG plays the role of server, generating a response to return to the client.

  For more information, see Handlers in the *Configuration Reference*.

- *Filter*: Either transforms data in the request, response, or context, or performs an action when the request or response passes through the filter.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

A filter can leave the request, response, and contexts unchanged. For example, it can log the context as it passes through the filter. Alternatively, it can change request or response. For example, it can generate a static request to replace the client request, add a header to the request, or remove a header from a response.

For more information, see Filters in the *Configuration Reference*.

- *Chain*: A type of handler that dispatches processing to an ordered list of filters, and then to the handler.

  A `Chain` can be placed anywhere in a configuration that a handler can be placed. Filters process the incoming request, pass it on to the next filter, and then to the handler. After the handler produces a response, the filters process the outgoing response as it makes its way to the client. Note that the same filter can process both the incoming request and the outgoing response but most filters do one or the other.

  For more information, see Chain(5) in the *Configuration Reference*.

- *Chain of Filters*: A type of filter that dispatches processing to an ordered list of filters without then dispatching the request to a handler. Use this filter to assemble a list of filters into a single filter that you can then use in different places in the configuration.

  A `ChainOfFilters` can be placed anywhere in a configuration that a filter can be placed.

  For more information, see ChainOfFilters(5) in the *Configuration Reference*.

"Flow Inside a Chain" shows the flow inside a `Chain`, where a request filter transforms the request, a handler sends the request to a protected application, and then a response filter transforms the response. Notice how the flow traverses the filters in reverse order when the response comes back from the handler.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Flow Inside a Chain

Flow Inside a Chain

IG Chain

Client
Request Filter
Response Filter
Handler
Protected application

Request
Transform request
Request, context
Request, context
Log request
Transformed request
Response
Log response
Request, context, response
Transform response
Request, context, response
Transformed response

Client
Request Filter
Response Filter
Handler
Protected application

The route configuration in "Chain to a Protected Application" demonstrates the flow through a chain to a protected application. With IG and the sample application set up as described in "*First Steps*" in the *Getting Started Guide,* access this route on http://openig.example.com:8080/home/chain.

Chain to a Protected Application

```
{
  "condition": "${matches(request.uri.path, '^/home/chain')}",
  "handler": {
    "type": "Chain",
    "comment": "Base configuration defines the capture decorator",
    "config": {
      "filters": [
        {
          "type": "HeaderFilter",
          "comment": "Add a header to all requests",
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
          "config": {
            "messageType": "REQUEST",
            "add": {
              "MyHeaderFilter_request": [
                "Added by HeaderFilter to request"
              ]
            }
          }
        },
        {
          "type": "HeaderFilter",
          "comment": "Add a header to all responses",
          "config": {
            "messageType": "RESPONSE",
            "add": {
              "MyHeaderFilter_response": [
                "Added by HeaderFilter to response"
              ]
            }
          }
        }
      ],
      "handler": {
        "type": "ReverseProxyHandler",
        "comment": "Log request, pass it to the sample app, log response",
        "capture": "all",
        "baseURI": "http://app.example.com:8081"
      }
    }
  }
}
```

The chain receives the request and context and processes it as follows:

- The first `HeaderFilter` adds a header to the incoming request.
- The second `HeaderFilter` manages responses not requests, so it simply passes the request and context to the handler.
- The `ReverseProxyHandler` captures (logs) the request.
- The `ReverseProxyHandler` passes the transformed request to the protected application.
- The protected application passes a response to the `ReverseProxyHandler`.
- The `ReverseProxyHandler` captures (logs) the response.
- The second `HeaderFilter` adds a header added to the response.
- The first `HeaderFilter` is configured to manage requests, not responses, so it simply passes the response back to IG.

"Requests and Responses in a Chain" list some of the HTTP requests and responses captured as they flow through the chain. You can search the log files for `MyHeaderFilter_request` and `MyHeaderFilter_response`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Requests and Responses in a Chain

```
### Original request from user-agent
GET http://openig.example.com:8080/home/chain HTTP/1.1
Accept: */*
Host: openig.example.com:8080

### Add a header to the request (inside IG) and direct it to the protected application
GET http://app.example.com:8081/home/chain HTTP/1.1
Accept: */*
Host: openig.example.com:8080
MyHeaderFilter_request: Added by HeaderFilter to request

### Return the response to the user-agent
HTTP/1.1 200 OK
Content-Length: 1809
Content-Type: text/html; charset=ISO-8859-1

### Add a header to the response (inside IG)
HTTP/1.1 200 OK
Content-Length: 1809
MyHeaderFilter_response: Added by HeaderFilter to response
```

1.6. Configuration Objects

Configuration objects have the following parts:

- *Name*: a unique string in the list of objects. When you declare inline objects, the name is not required.
- *Type*: the type name of the configuration object. IG defines many object types for different purposes.
- *Config*: additional configuration settings. The content of the configuration object depends on its type.

  If all of the configuration settings for the type are optional, the config field is also optional. The following configurations signify that the object uses default settings:

  - Omitting the config field
  - Setting the config field to an empty object, `"config": {}`
  - Setting `"config": null`

Filters, handlers, and other objects whose configuration settings are defined by strings, integers, or booleans, can alternatively be defined by expressions that match the expected type.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

1.7. Decorators

Decorators are additional heap objects to extend what another object can do. For example, a *CaptureDecorator* extends the capability of filters and handlers to log requests and responses. A *TimerDecorator* logs processing times. Decorate configuration objects with decorator names as field names.

IG defines the following decorators: `audit`, `baseURI`, `capture`, and `timer`. You can use these decorators without configuring them explicitly.

You can log requests, responses, and processing times by adding decorations as shown in the following example:

```
{
  "handler": {
  "type": "Router",
  "capture": [ "request", "response" ],
  "timer": true
  }
}
```

For more information, see Decorators in the *Configuration Reference*.

1.8. Configuration Parameters Declared as Property Variables

Configuration parameters, such as host names, port numbers, and directories, can be declared as property variables in the IG configuration or in an external JSON file. The variables can then be used in expressions in routes and in `config.json` to set the value of configuration parameters.

Properties can be inherited across the router, so a property defined in `config.json` can be used in any of the routes in the configuration.

Storing the configuration centrally and using variables for parameters that can be different for each installation makes it easier to deploy IG in different environments without changing a single line in your route configuration.

For more information, see Properties(5) in the *Configuration Reference*.

1.9. Using Comments in IG Configuration Files

The JSON format does not specify a notation for comments. If IG does not recognize a JSON field name, it ignores the field. As a result, it is possible to use comments in configuration files.

Use the following conventions when commenting to ensure your configuration files are easier to read:

- Use `comment` fields to add text comments. "Using a Comment Field" illustrates a CaptureDecorator configuration that includes a text comment.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Using a Comment Field

```
{
    "name": "capture",
    "type": "CaptureDecorator",
    "comment": "Write request and response information to the logs",
    "config": {
        "captureEntity": true
    }
}
```

- Use an underscore (`_`) to comment a field temporarily. "Using an Underscore" illustrates a `CaptureDecorator` that has `"captureEntity": true` commented out. As a result, it uses the default setting (`"captureEntity": false`).

Using an Underscore

```
{
    "name": "capture",
    "type": "CaptureDecorator",
    "config": {
        "_captureEntity": true
    }
}
```

1.10. Understanding IG APIs With API Descriptors

Common REST endpoints in IG serve API descriptors at runtime. When you retrieve an API descriptor for an endpoint, a JSON that describes the API for that endpoint is returned.

To help you discover and understand APIs, you can use the API descriptor with a tool such as Swagger UI to generate a web page that helps you to view and test the different endpoints.

When you start IG, or add or edit routes, registered endpoint locations for the routes hosted by the main router are written in `$HOME/.openig/logs/route-system.log`. Endpoint locations for subroutes are written to other log files. To retrieve the API descriptor for a specific endpoint, append one of the following query string parameters to the endpoint:

- `_api`, to represent the API accessible over HTTP. This OpenAPI descriptor can be used with endpoints that are complete or partial URLs.

  The returned JSON respects the OpenAPI specification and can be consumed by Swagger tools, such as Swagger UI.

- `_crestapi`, to provide a compact representation that is independent of the transport protocol. This ForgeRock® Common REST (Common REST) API descriptor cannot be used with partial URLs.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

The returned JSON respects a ForgeRock proprietary specification dedicated to describe Common REST endpoints.

For more information about Common REST API descriptors, see "Common REST API Documentation" in the *Configuration Reference*.

Retrieving API Descriptors for a Router

With IG running as described in "*First Steps*" in the *Getting Started Guide*, run the following query to generate a JSON that describes the router operations supported by the endpoint:

```
http://openig.example.com:8080/openig/api/system/objects/_router/routes?_api
{
     "swagger": "2.0",
     "info": {
     "version": "IG version",
     "title": "IG"
     },
     "host": "0:0:0:0:0:0:0:1",
     "basePath": "/openig/api/system/objects/_router/routes",
     "tags": [{
     "name": "Routes Endpoint"
     }],
     . . .
```

Alternatively, generate a Common REST API descriptor by using the `?_crestapi` query string.

Retrieving API Descriptors for the UMA Service

With the UMA tutorial running as described in "*Supporting UMA Resource Servers*", run the following query to generate a JSON that describes the UMA share API:

```
http://openig.example.com:8080/openig/api/system/objects/_router/routes/00-uma/objects/umaservice/share?
_api
{
     "swagger": "2.0",
     "info": {
     "version": "IG version",
     "title": "IG"
     },
     "host": "0:0:0:0:0:0:0:1",
     "basePath": "/openig/api/system/objects/_router/routes/00-uma/objects/umaservice/share",
     "tags": [{
     "name": "Manage UMA Share objects"
     }],
     . . .
```

Alternatively, generate a Common REST API descriptor by using the `?_crestapi` query string.

Retrieving API Descriptors for the Main Router

Run a query to generate a JSON that describes the API for the main router and its subsequent endpoints. For example:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
http://openig.example.com:8080/openig/api/system/objects/_router?_api
{
     "swagger": "2.0",
     "info": {
     "version": "IG version",
     "title": "IG"
     },
     "host": "openig.example.com:8080",
     "basePath": "/openig/api/system/objects/_router",
     "tags": [{
     "name": "Monitoring endpoint"
     }, {
     "name": "Manage UMA Share objects"
     }, {
     "name": "Routes Endpoint"
     }],
     . . .
```

Because the above URL is a partial URL, you cannot use the `?_crestapi` query string to generate a Common REST API descriptor.

Retrieving API Descriptors for an IG Instance

Run a query to generate a JSON that describes the APIs provided by the IG instance that is responding to a request. For example:

```
http://openig.example.com:8080/openig/api?_api
{
     "swagger": "2.0",
     "info": {
     "version": "IG version",
     "title": "IG"
     },
     "host": "openig.example.com:8080",
     "basePath": "/openig/api",
     "tags": [{
     "name": "Internal Storage for UI Models"
     }, {
     "name": "Monitoring endpoint"
     }, {
     "name": "Manage UMA Share objects"
     }, {
     "name": "Routes Endpoint"
     }, {
     "name": "Server Info"
     }],
     . . .
```

If routes are added after the request is performed, they are not included in the returned JSON.

Because the above URL is a partial URL, you cannot use the `?_crestapi` query string to generate a Common REST API descriptor.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 2

Installation in Detail

For information about how to quickly install and configure IG, see "*First Steps*" in the *Getting Started Guide*. This chapter contains information about how to do the following tasks:

- Prepare a deployment container for use with IG ("Configuring Deployment Containers").
- Prepare the network so that traffic passes through IG ("Preparing the Network").
- Download, deploy, and configure IG ("Installing IG").
- Change the locations of the configuration files ("Changing the Default Location of the Configuration Folders").
- Prepare for load balancing with IG ("Preparing For Load Balancing and Failover").
- Secure connections to and from IG ("Configuring IG For HTTPS (Client-Side)").
- Use IG JSON Web Token (JWT) Session cookies across multiple servers ("Setting Up Keys For JWT Encryption").
- Prevent further updates to the configuration ("Making the Configuration Immutable").

Before you begin to install or configure IG, make sure that you are using a supported container and version of Java. For information about requirements for running IG, see "*Before You Install*" in the *Release Notes*.

2.1. Configuring Deployment Containers

This section provides installation and configuration tips that you need to run IG in supported containers.

For the full list of supported containers see "Web Application Containers" in the *Release Notes*.

For information about advanced configuration for a container, see the container documentation.

2.1.1. About Securing Connections

IG is often deployed to replay credentials or other security information. In a real world deployment, that information must be communicated over a secure connection using HTTPS, meaning in effect

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

HTTP over encrypted Transport Layer Security (TLS). Never send real credentials, bearer tokens, or other security information unprotected over HTTP.

When IG is acting as a server, the web application container where IG runs is responsible for setting up TLS connections with client applications that connect to IG. For details, see "Configuring IG for HTTPS (Server-Side) in Jetty" or "Configuring Tomcat For HTTPS (Server-Side)".

When IG is acting as a client, the ReverseProxyHandler configuration governs TLS connections from IG to other servers. For details, see "Configuring IG For HTTPS (Client-Side)" and ReverseProxyHandler(5) in the *Configuration Reference*.

TLS depends on the use of digital certificates (public keys). In typical use of TLS, the client authenticates the server by its X.509 digital certificate as the first step to establishing communication. Once trust is established, then the client and server can set up a symmetric key to encrypt communications.

In order for the client to trust the server certificate, the client needs first to trust the certificate of the party who signed the server's certificate. This means that either the client has a trusted copy of the signer's certificate, or the client has a trusted copy of the certificate of the party who signed the signer's certificate.

Certificate Authorities (CAs) are trusted signers with well-known certificates. Browsers generally ship with many well-known CA certificates. Java distributions also ship with many well-known CA certificates. Getting a certificate signed by a well-known CA is often expensive.

It is also possible for you to self-sign certificates. The trade-off is that although there is no monetary expense, the certificate is not trusted by any clients until they have a copy. Whereas it is often enough to install a certificate signed by a well-known CA in the server keystore as the basis of trust for HTTPS connections, self-signed certificates must also be installed in all clients.

Like self-signed certificates, the signing certificates of less well-known CAs are also unlikely to be found in the default truststore. You might therefore need to install those signing certificates on the client-side as well.

This guide describes how to install self-signed certificates, which are certainly fine for trying out the software and okay for deployments where you manage all clients that access IG. If you need a well-known CA-signed certificate instead, see the documentation for your container for details on requesting a CA signature and installing the CA-signed certificate.

Once certificates are properly installed to allow client-server trust, also consider the cipher suites configured for use. The cipher suite used determines the security settings for the communication. Initial TLS negotiations bring the client and server to agreement on which cipher suite to use. Basically the client and server share their preferred cipher suites to compare and to choose. If you therefore have a preference concerning the cipher suites to use, you must set up your container to use only your preferred cipher suites. Otherwise the container is likely to inherit the list of cipher suites from the underlying Java environment.

The Java Secure Socket Extension (JSSE), part of the Java environment, provides security services that IG uses to secure connections. You can set security and system properties to configure the JSSE.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

For a list of properties you can use to customize the JSSE in Oracle Java, see the *Customization* section of the *JSSE Reference Guide*.

2.1.2. Apache Tomcat For IG

This section describes essential Tomcat configuration that you need in order to run IG.

Download and install a supported version of Tomcat from http://tomcat.apache.org/.

> **Important**
>
> If you use startup scripts to bootstrap the IG web container, the scripts can start the container process with a different user. To prevent errors, make sure that the location of the IG configuration is correct. Alternatively, adapt the startup scripts to specify the `IG_INSTANCE_DIR` env variable or `ig.instance.dir` system properties, taking care to set file permissions correctly.
>
> If you start and stop the IG web container yourself, the default location of the IG configuration files is correct. By default, IG configuration files are located under `$HOME/.openig` on Linux, Mac, and UNIX systems, and under `%appdata%\OpenIG` on Windows.

Configure Tomcat to use the same protocol as the application you are protecting with IG. If the protected application is on a remote system, configure Tomcat to use the same port as well. If your application listens on both an HTTP and an HTTPS port, then you must configure Tomcat to do so, too.

To configure Tomcat to use an HTTP port other than 8080, modify the defaults in `/path/to/tomcat/conf/server.xml`. Search for the default value of 8080 and replace it with the new port number.

2.1.2.1. Configuring Tomcat Cookie Domains

To use IG for multiple protected applications running on different hosts, set a cookie domain in Tomcat or `JwtSession`.

- To set a cookie domain for an HTTP session (the default if you're not using a JWT session), add a context element to `/path/to/conf/Catalina/server/root.xml`, as in the following example, and then restart Tomcat to read the configuration changes:

  ```
  <Context sessionCookieDomain=".example.com" />
  ```

- To set a cookie domain for a JWT session, set the `JwtSession`. property `domain`. For information, see JwtSession(5) in the *Configuration Reference*. When set, a JWT cookie can be accessed from different hosts in that domain. When not set, the JWT cookie can be accessed only from the host where the cookie was created.

2.1.2.2. Configuring Tomcat For HTTPS (Server-Side)

To get Tomcat up quickly on an SSL port, add an entry similar to the following in `/path/to/tomcat/conf/server.xml`:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
<Connector
 port="8443"
 protocol="HTTP/1.1"
 SSLEnabled="true"
 maxThreads="150"
 scheme="https"
 secure="true"
 address="127.0.0.1"
 clientAuth="false"
 sslProtocol="TLS"
 keystoreFile="/path/to/tomcat/conf/keystore"
 keystorePass="password"
/>
```

Also create a keystore holding a self-signed certificate:

```
$ keytool \
 -genkey \
 -alias tomcat \
 -keyalg RSA \
 -keystore /path/to/tomcat/conf/keystore \
 -storepass password \
 -keypass password \
 -dname "CN=openig.example.com,O=Example Corp,C=FR"
```

Note

Because KeyStore converts all characters in its key aliases to lower case, use only lowercase in alias definitions of a KeyStore.

Notice the keystore file location and the keystore password both match the configuration. By default, Tomcat looks for a certificate with alias `tomcat`.

Restart Tomcat to read the configuration changes.

Browsers generally do not trust self-signed certificates. To work with a certificate signed instead by a trusted CA, see the Tomcat documentation on configuring HTTPS.

2.1.2.3. Configuring Tomcat to Access MySQL Over JNDI

If IG accesses an SQL database, then you must configure Tomcat to access the database using Java Naming and Directory Interface (JNDI). To do so, you must add the driver .jar for the database, set up a JNDI data source, and set up a reference to that data source.

The following steps are for MySQL Connector/J:

1. Download the MySQL JDBC Driver Connector/J from http://dev.mysql.com/downloads/connector/j.

2. Copy the driver .jar to `/path/to/tomcat/lib/` so that it is on Tomcat's class path.

3. Add a JNDI data source for your MySQL server and database in `/path/to/tomcat/conf/context.xml`:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
<Resource
 name="jdbc/forgerock"
 auth="Container"
 type="javax.sql.DataSource"
 maxActive="100"
 maxIdle="30"
 maxWait="10000"
 username="mysqladmin"
 password="password"
 driverClassName="com.mysql.jdbc.Driver"
 url="jdbc:mysql://localhost:3306/databasename"
/>
```

4. Add a resource reference to the data source in `/path/to/tomcat/conf/web.xml`:

```
<resource-ref>
    <description>MySQL Connection</description>
    <res-ref-name>jdbc/forgerock</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

5. Restart Tomcat to read the configuration changes.

2.1.3. Jetty For IG

This section describes essential Jetty configuration that you need in order to run IG.

Download and install a supported version of Jetty from https://www.eclipse.org/jetty/download.html.

Configure Jetty to use the same protocol as the application you are protecting with IG. If the protected application is on a remote system, configure Jetty to use the same port as well. If your application listens on both an HTTP and an HTTPS port, then you must configure Jetty to do so as well.

To configure Jetty to use an HTTP port other than 8080, modify the defaults in `/path/to/jetty/etc/jetty.xml`. Search for the default value of 8080 and replace it with the new port number.

2.1.3.1. Configuring Jetty Cookie Domains

To use IG for multiple protected applications running on different hosts, set a cookie domain in Jetty or `JwtSession`:

- To set a cookie domain in Jetty, add a session domain handler element that specifies the domain to `/path/to/jetty/etc/webdefault.xml`, as in the following example:

```
<context-param>
  <param-name>org.eclipse.jetty.servlet.SessionDomain</param-name>
  <param-value>.example.com</param-value>
</context-param>
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Restart Jetty to read the configuration changes.

- To set a cookie domain for a JWT session, set the `JwtSession`. property `domain`. For information, see JwtSession(5) in the *Configuration Reference*. When set, a JWT cookie can be accessed from different hosts in that domain. When not set, the JWT cookie can be accessed only from the host where the cookie was created.

2.1.3.2. Configuring IG for HTTPS (Server-Side) in Jetty

This section describes how to set up Jetty to run IG over HTTPS. For information about the setup for HTTPS (client-side), see "Configuring IG For HTTPS (Client-Side)".

These instructions are for Jetty 9.4.21, and are not compatible with earlier versions of Jetty. For more information about Jetty and HTTPS, see http://www.eclipse.org/jetty/documentation/current/configuring-ssl.html#configuring-sslcontextfactory.

To Configure Jetty for HTTPS

1. Install Jetty, and set up the location for the Jetty distribution binaries:

    a. Download a supported version of Jetty server from its download page, and install it to `/path/to/jetty`.

    b. Set the environment variable JETTY_HOME for `/path/to/jetty`:

    ```
    $ export JETTY_HOME=/path/to/jetty
    ```

2. Set up the location for configurations and customizations to the Jetty distribution:

    a. Create a directory `/path/to/jetty_base`.

    b. Set the environment variable JETTY_BASE for `/path/to/jetty_base`:

    ```
    $ export JETTY_BASE=/path/to/jetty_base
    ```

3. Set up the keystore:

    a. Remove the built-in keystore:

    ```
    $ rm ${JETTY_HOME}/modules/ssl/keystore
    ```

    b. Generate a key pair with a self-signed certificate in the keystore:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ keytool \
-genkey
 \
-alias jetty
 \
-keyalg RSA
 \
-keystore ${JETTY_HOME}/modules/ssl/keystore
 \
-storepass password
 \
-keypass password
 \
-dname "CN=openig.example.com,O=Example Corp,C=FR"
```

> **Note**
>
> Because KeyStore converts all characters in its key aliases to lower case, use only lowercase in alias definitions of a KeyStore.

4. Create a directory to store local server customization and configurations in ${JETTY_BASE}:

   a. Delete the global `start.ini`:

   ```
   $ rm ${JETTY_HOME}/start.ini
   ```

   b. From ${JETTY_BASE}, create the `start.d` folder to hold the module `.ini` files:

   ```
   $ cd ${JETTY_BASE}
   $ java -jar ${JETTY_HOME}/start.jar --create-startd

   MKDIR : ${jetty.base}/start.d
   INFO  : Base directory was modified
   ```

5. From ${JETTY_BASE}, add the following Jetty configuration modules:

```
$ cd ${JETTY_BASE}
$ java  -jar ${JETTY_HOME}/start.jar
 \
--add-to-start=server,webapp,deploy,ssl,jstl,ext,jsp,resources,console-capture,http,https

INFO  : webapp          initialized in ${jetty.base}/start.d/webapp.ini
INFO  : ext             initialized in ${jetty.base}/start.d/ext.ini
INFO  : server          initialized in ${jetty.base}/start.d/server.ini
INFO  : mail            transitively enabled
INFO  : servlet         transitively enabled
INFO  : jsp             initialized in ${jetty.base}/start.d/jsp.ini
INFO  : annotations     transitively enabled
INFO  : resources       initialized in ${jetty.base}/start.d/resources.ini
INFO  : transactions    transitively enabled
INFO  : threadpool      transitively enabled, ini template available with --add-to-start=threadpool
INFO  : ssl             initialized in ${jetty.base}/start.d/ssl.ini
INFO  : plus            transitively enabled
INFO  : deploy          initialized in ${jetty.base}/start.d/deploy.ini
INFO  : jstl            initialized in ${jetty.base}/start.d/jstl.ini
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
INFO  : security        transitively enabled
INFO  : apache-jsp      transitively enabled
INFO  : jndi            transitively enabled
INFO  : console-capture initialized in ${jetty.base}/start.d/console-capture.ini
INFO  : apache-jstl     transitively enabled
INFO  : http            initialized in ${jetty.base}/start.d/http.ini
INFO  : client          transitively enabled
INFO  : https           initialized in ${jetty.base}/start.d/https.ini
INFO  : bytebufferpool  transitively enabled, ini template available with --add-to-
start=bytebufferpool
MKDIR : ${jetty.base}/lib
MKDIR : ${jetty.base}/lib/ext
MKDIR : ${jetty.base}/resources
MKDIR : ${jetty.base}/etc
COPY  : ${jetty.home}/modules/ssl/keystore to ${jetty.base}/etc/keystore
MKDIR : ${jetty.base}/webapps
MKDIR : ${jetty.base}/logs
INFO  : Base directory was modified
```

> **Note**
>
> IG depends on javax.websocket-api version 1.1, which is a later version than that provided by Jetty. To prevent errors related to WebSocket, do not include the websocket configuration modules when you configure Jetty.
>
> To change the default port for Jetty in HTTP, edit `http.ini`.
>
> To change the default port for Jetty in HTTPS, edit `server.ini`.

6. Replace `jetty-util-*.jar` with the version for your installation, and find the obfuscated form of the keystore password:

```
$ cd ${JETTY_HOME}/lib
$ ls jetty-util-*.jar
```

```
$ java -cp jetty-util-*.jar org.eclipse.jetty.util.security.Password password

password
OBF:1v2...v1v
MD5:5f4...cf99
```

7. In `${JETTY_BASE}/start.d/ssl.ini`, uncomment the following lines, and update the passwords with the OBF password returned in the previous step:

```
## Connector port to listen on
jetty.ssl.port=8443

## Keystore file path (relative to $jetty.base)
jetty.sslContext.keyStorePath=etc/keystore

## Keystore password
jetty.sslContext.keyStorePassword=OBF:1v2...v1v

## KeyManager password
jetty.sslContext.keyManagerPassword=OBF:1v2...v1v
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

8. Copy the IG .war file to `${JETTY_BASE}/webapps/IG-6.5.4.war`.
9. Go to ${JETTY_BASE}, and start Jetty:

```
$ cd ${JETTY_BASE}
$ java -jar ${JETTY_HOME}/start.jar
```

10. Access the IG welcome page on https://openig.example.com:8443.

    If you see warnings that the site is not secure, or that the self-signed certificate is not valid, respond to the warnings to access the site.

2.1.3.3. Configuring Jetty to Access MySQL Over JNDI

If IG accesses an SQL database, then you must configure Jetty to access the database over JNDI. To do so, you must add the driver .jar for the database, set up a JNDI data source, and set up a reference to that data source.

The following steps are for MySQL Connector/J:

1. Download the MySQL JDBC Driver Connector/J from http://dev.mysql.com/downloads/connector/j.
2. Copy the driver .jar to `/path/to/jetty/lib/jndi/` so that it is on Jetty's class path.
3. Add a JNDI data source for your MySQL server and database in `/path/to/jetty/etc/jetty.xml`:

```
<New id="jdbc/forgerock" class="org.eclipse.jetty.plus.jndi.Resource">
  <Arg></Arg>
  <Arg>jdbc/forgerock</Arg>
  <Arg>
    <New class="com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource">
      <Set name="Url">jdbc:mysql://localhost:3306/databasename</Set>
      <Set name="User">mysqladmin</Set>
      <Set name="Password">password</Set>
    </New>
  </Arg>
</New>
```

4. Add a resource reference to the data source in `/path/to/jetty/etc/webdefault.xml`:

```
<resource-ref>
    <description>MySQL Connection</description>
    <res-ref-name>jdbc/forgerock</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

5. Restart Jetty to read the configuration changes.

2.1.4. JBoss EAP For IG

This section describes a basic JBoss configuration to run IG.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Install IG in JBoss

1. Download and install a supported version of JBoss from https://developers.redhat.com/products/eap/download/.

   For information about supported versions, see "*Before You Install*" in the *Release Notes*.

2. Delete the JBoss welcome-content handler in the JBoss configuration file `/path/to/jboss/standalone/configuration/standalone.xml`:

```
<server name="default-server">
  <host name="default-host" alias="localhost">
    <location name="/" handler="welcome-content"/> <!-- Delete this line -->
```

3. Download `IG-6.5.4.war` from the ForgeRock BackStage download site and move it to the JBoss deployment directory:

```
$ cp IG-6.5.4.war /path/to/jboss/standalone/deployments/IG-6.5.4.war
```

4. (Optional) Add IG configuration:

   - To configure IG in the filesystem, edit `/path/to/jboss/bin/standalone.conf`. For example, add the IG base to the file:

```
export IG_INSTANCE_DIR="/path/to/openig"
```

   - To configure IG at startup, add the configuration to the startup command. For example, add the IG base to the startup command:

```
$ /path/to/jboss/bin/standalone.sh -Dig.instance.dir=/path/to/openig
```

5. Start JBoss as a standalone server:

```
$ /path/to/jboss/bin/standalone.sh
```

   JBoss deploys IG in the root context.

6. Make sure that IG is running:

   - Make sure that you can see Studio at http://localhost:8080/openig/studio
   - Make sure that you can see the IG welcome page at http://localhost:8080

2.1.4.1. Configuring JBoss Cookie Domains

To use IG to protect multiple applications running on different hosts, set a cookie domain in JBoss or `JwtSession`:

- To set a cookie domain in JBoss, see the Redhat documentation about *Cookie Domain*.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- To set a cookie domain for a JWT session, set the `JwtSession`. property `domain`. For information, see JwtSession(5) in the *Configuration Reference*. When set, a JWT cookie can be accessed from different hosts in that domain. When not set, the JWT cookie can be accessed only from the host where the cookie was created.

2.1.4.2. Configuring JBoss For HTTPS (Server-Side)

Configure JBoss for HTTPS, depending on the requirements of application you are protecting with IG.

If the protected application listens on both an HTTP and an HTTPS port, configure JBoss to listen on both ports.

2.2. Preparing the Network

Because IG uses reverse proxy architecture, you must configure the network so that that traffic from the browser to the protected application goes through IG.

Modify DNS or host file settings so that the host name of the protected application resolves to the IP address of IG on the system where the browser runs.

Restart the browser after making this change.

2.3. Installing IG

This section describes a basic installation in Jetty or Tomcat. For information about installing in JBoss, see see "JBoss EAP For IG".

Follow these steps to install IG:

1. Download `IG-6.5.4.war` from the ForgeRock BackStage download site .
2. Copy the .war file to the root context of the web application container:
   - For Tomcat, copy the file to `/path/to/tomcat/webapps/ROOT.war`.
   - For Jetty, copy the file to `/path/to/jetty/webapps/IG-6.5.4.war`. Jetty automatically deploys IG in the root context on startup.
   - For JBoss, see "JBoss EAP For IG".

   > **Important**
   >
   > If you use startup scripts to bootstrap the IG web container, the scripts can start the container process with a different user. To prevent errors, make sure that the location of the IG configuration is correct.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

> Alternatively, adapt the startup scripts to specify the `IG_INSTANCE_DIR` env variable or `ig.instance.dir` system properties, taking care to set file permissions correctly.
>
> If you start and stop the IG web container yourself, the default location of the IG configuration files is correct. By default, IG configuration files are located under `$HOME/.openig` on Linux, Mac, and UNIX systems, and under `%appdata%\OpenIG` on Windows.

3. Prepare your IG configuration files.

   For information about configuration files, see "Configuration Directories and Files". For information about how to change the default location of the configuration files, see "Changing the Default Location of the Configuration Folders".

   If you have not yet prepared configuration files, then start with the configuration described in "Trying IG With a Simple Configuration" in the *Getting Started Guide*.

   Copy the template to `$HOME/.openig/config/config.json`. Replace the baseURI of the DispatchHandler with that of the protected application.

   On Windows, copy the template to `%appdata%\OpenIG\config\config.json`. To locate the `%appdata%` folder for your version of Windows, open Windows Explorer, type `%appdata%` as the file path, and press Enter. You must create the `%appdata%\OpenIG\config` folder, and then add the configuration file.

4. Start the web container where IG is deployed.

5. Browse to the protected application.

   IG should now proxy all traffic to the application.

6. Make sure the browser is going through IG.

   Verify this in one of the following ways:

   - Follow these steps:

     1. Stop the IG web container.

     2. Verify that you cannot browse to the protected application.

     3. Start the IG web container.

     4. Verify that you can now browse to the protected application again.

   - Check the logs to see that traffic is going through IG.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

2.4. Changing the Default Location of the Configuration Folders

By default, the IG configuration files are in the directory `$HOME/.openig` (on Windows, `%appdata%\OpenIG`) Change the default location in the following ways:

- Set the `IG_INSTANCE_DIR` environment variable to the full path to the base location for IG files:

```
# On Linux, macOS, and UNIX using Bash
$ export IG_INSTANCE_DIR=/path/to/openig

# On Windows
C:>set IG_INSTANCE_DIR=c:\path\to\openig
```

- When you start the web application container where IG runs, set the `ig.instance.dir` Java system property to the full path to the base location for IG files.

  The following example starts Jetty server in the foreground and sets the value of `ig.instance.dir`:

```
$ java -Dig.instance.dir=/path/to/openig -jar start.jar
```

2.5. Preparing For Load Balancing and Failover

For a high scale or highly available deployment, you can prepare a pool of IG servers with nearly identical configurations, and then load balance requests across the pool, routing around any servers that become unavailable. Load balancing allows the service to handle more load.

Before you spread requests across multiple servers, however, you must determine what to do with state information that IG saves in the context, or retrieves locally from the IG server system. If information is retrieved locally, then consider setting up failover. If one server becomes unavailable, another server in the pool can take its place. The benefit of failover is that a server failure can be invisible to client applications.

IG can save state information in several ways:

- Handlers including a `SamlFederationHandler` or a custom `ScriptableHandler` can store information in the context. Most handlers depend on information in the context, some of which is first stored by IG.
- Some filters, such as `AssignmentFilters`, `HeaderFilters`, `OAuth2ClientFilters`, `OAuth2ResourceServerFilters`, `ScriptableFilters`, `SqlAttributesFilters`, and `StaticRequestFilters`, can store information in the context. Most filters depend on information in the request, response, or context, some of which is first stored by IG.

IG can also retrieve information locally in several ways:

- Some filters and handlers, such as `FileAttributesFilters`, `ScriptableFilters`, `ScriptableHandlers`, and `SqlAttributesFilters`, can depend on local system files or container configuration.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

By default, the context data resides in memory in the container where IG runs. This includes the default session implementation, which is backed by the HttpSession that the container handles. You can opt to store session data on the user-agent instead, however. For details and to consider whether your data fits, see JwtSession(5) in the *Configuration Reference*.

When you use the `JwtSession` implementation with a cookie domain set, be sure to share the encryption keys and the signature symmetric secret across all IG configurations so that any server can read or update JWT cookies from any other server in the same cookie domain.

If your data does not fit in an HTTP cookie, for example, because when encrypted it is larger than 4 KB, consider storing a reference in the cookie, and then retrieve the data by using another filter. IG logs warning messages if the `JwtSession` cookie is too large. Using a reference can also work when a server becomes unavailable, and the load balancer must fail requests over to another server in the pool.

If some data attached to a context must be stored on the server-side, then you have additional configuration steps to perform for session stickiness and for session replication. Session stickiness means that the load balancer sends all requests from the same client session to the same server. Session stickiness helps to ensure that a client request goes to the server holding the original session data. Session replication involves writing session data either to other servers or to a data store, so that if one server goes down, other servers can read the session data and continue processing. Session replication helps when one server fails, allowing another server to take its place without having to start the session over again. If you set up session stickiness but not session replication, when a server crashes, the client session information for that server is lost, and the client must start again with a new session.

How you configure session stickiness and session replication depends on your load balancer and on your container.

Tomcat can help with session stickiness, and a Tomcat cluster can handle session replication:

- If you choose to use the Tomcat connector (mod_jk) on your web server to perform load balancing, then see the *LoadBalancer HowTo* for details.

  In the HowTo, you configure the `jvmRoute` attribute in the Tomcat server configuration, `/path/to/tomcat/conf/server.xml`, to identify the server. The connector can use this identifier to achieve session stickiness.

- A Tomcat cluster configuration can handle session replication. When setting up a cluster configuration, the ClusterManager defines the session replication implementation.

Jetty has provisions for session stickiness, and also for session replication through clustering:

- Jetty's persistent session mechanism appends a node ID to the session ID in the same way Tomcat appends the `jvmRoute` value to the session cookie. This can be useful for session stickiness if your load balancer examines the session ID.

- Session Clustering with a Database describes how to configure Jetty to persist sessions over JDBC, allowing session replication.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Unless it is set up to be highly available, the database can be a single point of failure in this case.

- Session Clustering with MongoDB describes how to configure Jetty to persist sessions in MongoDB, allowing session replication.

  The Jetty documentation recommends this implementation when session data is seldom written but often read.

2.6. Configuring IG For HTTPS (Client-Side)

For IG to connect to a server securely over HTTPS, IG must be able to trust the server. The default settings rely on the Java environment truststore to trust server certificates. The Java environment default truststore includes public key signing certificates from many well-known Certificate Authorities (CAs). If all servers present certificates signed by these CAs, then you have nothing to configure.

If, however, the server certificates are self-signed or signed by a CA whose certificate is not trusted out of the box, then you can configure the following objects:

- KeyStore, to hold the servers' certificates or the CA's signing certificate. For information, see KeyStore(5) in the *Configuration Reference*.
- TrustManager, to allow IG to handle the certificates in the KeyStore when deciding whether to trust a server certificate. For information, see TrustManager(5) in the *Configuration Reference*.
- KeyManager, an optional object to allow IG to present its certificate from the keystore when the server must authenticate IG as client. This object is optional. For information, see KeyManager(5) in the *Configuration Reference*.
- ReverseProxyHandler can reference a TlsOptions object to configure options for connecting to TLS-protected endpoints. For information, see TlsOptions(5) in the *Configuration Reference*.

You can configure each of these objects globally for the IG server, or locally for a particular ReverseProxyHandler configuration.

The Java KeyStore holds the peer servers' public key certificates (and optionally, the IG certificate and private key). For example, suppose you have a certificate file, `ca.crt`, that holds the trusted signer's certificate of the CA who signed the server certificates of the servers in your deployment. In that case, you could import the certificate into a Java Keystore file, `/path/to/keystore.jks`:

```
$ keytool \
 -import \
 -trustcacerts \
 -keystore /path/to/keystore \
 -file ca.crt \
 -alias ca-cert \
 -storepass changeit
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Note

Because KeyStore converts all characters in its key aliases to lower case, use only lowercase in alias definitions of a KeyStore.

You could then configure the following KeyStore for IG to hold the trusted certificate. Notice that the url field takes an expression that evaluates to a URL, starting with a scheme such as `file://`:

```
{
  "name": "MyKeyStore",
  "type": "KeyStore",
  "config": {
    "url": "file:///path/to/keystore",
    "passwordSecretId": "keystore.secret.id"
  }
}
```

The KeyStore password is managed by the ForgeRock Commons Secrets Service.
For more information, see Secrets in the *Configuration Reference*. To use the default SystemAndEnvSecretStore to find the value of the password, export the following environment variable to the IG process:

```
$ export KEYSTORE_SECRET_ID='Y2hhbmdlaXQ='
```

The environment variable contains the base64-encoded value of the KeyStore password.

The TrustManager handles the certificates in the KeyStore when deciding whether to trust the server certificate. The TrustManager references your KeyStore:

```
{
  "name": "MyTrustManager",
  "type": "TrustManager",
  "config": {
    "keystore": "MyKeyStore"
  }
}
```

The `ReverseProxyHandler` configuration has the following security settings:

- `hostnameVerifier`: Defines how the ReverseProxyHandler verifies host names in server certificates. By default, this setting is `STRICT`.

- `tls`: References a TlsOptions object to configure options for connections to TLS-protected endpoints. For information, see TlsOptions(5) in the *Configuration Reference*

  If server certificates are not trusted out of the box, configure a TrustManager.

  If servers require IG to present its certificate as part of mutual authentication, configure a KeyManager. Generate a key pair for IG, and have the certificate signed by a well-known CA. For more information, see the documentation for the Java **keytool** command.

  You can use a different keystore for the `KeyManager` and `TrustManager`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

The following ReverseProxyHandler configuration sets strict host name verification, and references `MyTrustManager`:

```
{
  "name": "ReverseProxyHandler",
  "type": "ReverseProxyHandler",
  "config": {
    "hostnameVerifier": "STRICT",
    "tls": {
      "type": "TlsOptions",
      "config": {
        "trustManager": "MyTrustManager"
      }
    }
  }
}
```

2.7. Setting Up Keys For JWT Encryption

A JwtSession stores session information in JWT cookies on the user-agent, rather than storing the information in the container where IG runs.

In order to encrypt the JWTs, IG needs cryptographic keys. IG can generate its own key pair in memory, but that key pair disappears on restart and cannot be shared across IG servers. Alternatively, IG can use keys from a keystore.

The following procedure prepares a keystore for JWT encryption in a deployment with one IG instance. For a deployment with more than one IG instance, prepare a shared secret, distribute it amongst your instances, and then map the secret ID to this shared secret.

For information about sessions, see JwtSession(5) in the *Configuration Reference*. For information about the ForgeRock Commons Secrets Service, see Secrets in the *Configuration Reference*.

To Set Up Keys for JWT Encryption

1. Generate a keystore, where the keystore and the private key have the same password:

```
$ keytool \
  -genkey \
  -alias jwe-key \
  -keyalg rsa \
  -keystore /path/to/secrets/jwtsessionkeystore.pkcs12 \
  -storepass password \
  -keypass password \
  -dname "CN=openig.example.com,O=Example Corp"
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

> **Note**
>
> Because KeyStore converts all characters in its key aliases to lower case, use only lowercase in alias definitions of a KeyStore.

2. Create an IG environment variable with the value of the KeyStore password, `password`, and restart IG:

   ```
   $ export KEYSTORE_SECRET_ID='cGFzc3dvcmQ='
   ```

   The password is retrieved by the default SystemAndEnvSecretStore, and must be base64-encoded.

3. In IG, add the following route as `$HOME/.openig/config/routes/jwt-session.json` (on Windows, `%appdata%\OpenIG\config\routes\jwt-session.json`).

```
{
  "name": "jwt-session",
  "secrets": {
    "stores": [
      {
        "name": "KeyStoreSecretStore",
        "type": "KeyStoreSecretStore",
        "config": {
          "file": "/path/to/secrets/jwtsessionkeystore.pkcs12",
          "storeType": "PKCS12",
          "storePassword": "keystore.secret.id",
          "mappings": [{
            "secretId": "jwtsession.encryption.secret.id",
            "aliases": [ "jwe-key" ]
          }]
        }
      }
    ]
  },
  "session": {
    "type": "JwtSession",
    "config": {
      "encryptionSecretId": "jwtsession.encryption.secret.id",
      "cookie": {
        "name": "IG",
        "domain": ".example.com"
      }
    }
  },
  "handler": {
    "type": "StaticResponseHandler",
    "config": {
      "status": 200,
      "reason": "OK",
      "entity": "Hello, world!"
    }
  },
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
  "condition": "${matches(request.uri.path, '^/jwt-session')}"
}
```

4. Notice the following features of the route:

   - The route matches requests to `/jwt-session`.
   - The secrets object configures a KeyStoreSecretStore to manage secrets for session encryption. The KeyStoreSecretStore includes a secret ID mapping to the alias of the session encryption key.

     For information, see KeyStoreSecretStore(5) in the *Configuration Reference*.
   - The session refers to the encryption key to encrypt the JWT.

To Set Up Shared Secrets for Multiple Instances of IG

In deployments with more than one IG instance, configure a JWT signature to sign and verify the JWTs.

1. Create an IG environment variable with the value of a session signature secret, and restart IG: `this_is_a_session_signature_secret`:

   ```
   $ export JWTSESSION_SIGNATURE_SECRET_ID='dGhpc19pc19hX3Nlc3Npb25fc2lnbmF0dXJlX3NlY3JldA=='
   ```

   The secret must be at least 32-characters long and base64-encoded.

2. Add the `signatureSecretId` to the `session` object in the previous route:

   ```
   {
     "session": {
       "type": "JwtSession",
       "config": {
         "encryptionSecretId": "jwtsession.encryption.secret.id",
         "cookie": {
           "name": "IG",
           "domain": ".example.com"
         },
         "signatureSecretId": "jwtsession.signature.secret.id"
       }
     }
   }
   ```

2.8. Making the Configuration Immutable

IG operates in the following modes:

- **Development mode (mutable mode)**

  Use development mode to evaluate or demo IG, or to develop configurations on a single instance. This mode is not suitable for production.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

In development mode, by default all endpoints are open and accessible. You can create, edit, and deploy routes through IG Studio, and manage routes through Common REST, without authentication or authorization.

To protect specific endpoints in development mode, configure an ApiProtectionFilter in `admin.json` and add it to the IG configuration.

- **Production mode (immutable mode)**

  After you have developed your configuration, switch to production mode to test the configuration, to run the software in pre-production or production, or to run multiple instances of the software with the same configuration.

  In production mode, the `/routes` endpoint is not exposed or accessible. Studio is effectively disabled, and you cannot manage, list, or even read routes through Common REST.

  By default, other endpoints, such as `/share` and `api/info` are exposed to the loopback address only. To change the default protection for specific endpoints, configure an ApiProtectionFilter in `admin.json` and add it to the IG configuration.

After installation, IG is by default in production mode. While you evaluate IG or develop routes, it can be helpful to switch to development mode as described in "Switching Between Production Mode and Development Mode" in the *Getting Started Guide*. This section describes options for switching back to production mode to run IG in a production environment.

To Make the Configuration Immutable

1. In `$HOME/.openig/config/admin.json` (on Windows, `%appdata%\OpenIG\config`), change the value of `mode` from `DEVELOPMENT` to `PRODUCTION`:

   ```
   {
     "mode": "PRODUCTION"
   }
   ```

   The file changes the operating mode from development mode to production mode. For more information about the `admin.json` file, see AdminHttpApplication(5) in the *Configuration Reference*.

   The value set in `admin.json` overrides any value set by the configuration token `ig.run.mode` when it is used in an environment variable or system property. For information about `ig.run.mode`, see Configuration Tokens(5) in the *Configuration Reference*.

2. (Optional) Prevent routes from being reloaded after startup:

   - To prevent all routes in the configuration from being reloaded, add a `config.json` as described in "Trying IG With a Simple Configuration" in the *Getting Started Guide*, and configure the `scanInterval` of the main router.

   - To prevent individual routes from being reloaded, configure the `scanInterval` of the routers in those routes.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
{
  "type": "Router",
  "config": {
    "scanInterval": "disabled"
  }
}
```

For more information, see Router(5) in the *Configuration Reference*.

3. Restart IG.

   When IG starts up, the route endpoints are not displayed in the logs, and are not available. You can't access Studio on http://openig.example.com:8080/openig/studio.

2.9. Restricting Access to Studio in Development Mode

When IG is running in development mode, by default the Studio endpoint is open and accessible. To restrict access to Studio in development mode, configure a filter named StudioProtectionFilter in `admin.json`, and add it to the IG configuration.

For more information about StudioProtectionFilter, see "Provided Objects" in the *Configuration Reference*.

To Restrict Access to Studio in Development Mode

This procedure uses StudioProtectionFilter and SingleSignOnFilter to require users to authenticate with AM before they can access Studio. To allow access for specific users only, consider using StudioProtectionFilter with SingleSignOnFilter and PolicyEnforcementFilter.

Before you start, prepare IG and the sample application as described in "*First Steps*" in the *Getting Started Guide*, and access Studio as described in "Accessing Studio" in the *Getting Started Guide*. IG must be running in development mode.

1. (For AM 6.5.3 and earlier versions) Select Identities > demo, and set the demo user password to `Ch4ng31t`.

2. (For AM 6.5.3 and later versions) Select Services, and add a Validation Service with the following URLs for IG:

   - `http://openig.example.com:8080/*`
   - `http://openig.example.com:8080/*?*`

3. (For AM 6.5.3 and later versions) Select Applications > Agents > Identity Gateway, and add an agent with the following values:

   - Agent ID: `ig_agent`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Password: `password`

Leave all other values as default.

(For AM 6.5.2 and earlier versions) Set up an agent as described in "To Set Up a Java Agent in AM".

4. Set an IG environment variable for the Java agent password:

```
$ export AGENT_SECRET_ID='cGFzc3dvcmQ='
```

The password is retrieved by the default SystemAndEnvSecretStore, and must be base64-encoded.

5. In IG, add the following file as `$HOME/.openig/config/admin.json` (on Windows, `%appdata%\OpenIG\config\admin.json`):

```
{
  "prefix": "openig",
  "mode": "DEVELOPMENT",
  "properties": {
    "SsoTokenCookieOrHeader": "iPlanetDirectoryPro"
  },
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "agent" : {
          "username" : "ig_agent",
          "passwordSecretId" : "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "url": "http://openam.example.com:8088/openam/",
        "ssoTokenHeader": "&{SsoTokenCookieOrHeader}",
        "version": "${platform.version}"
      }
    },
    {
      "name": "StudioProtectionFilter",
      "type": "ChainOfFilters",
      "config": {
        "filters": [
          {
            "type": "SingleSignOnFilter",
            "config": {
              "amService": "AmService-1"
            }
          },
          {
            "type": "CsrfFilter",
            "config": {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
                "cookieName": "&{SsoTokenCookieOrHeader}",
                "failureHandler": {
                  "type": "StaticResponseHandler",
                  "config": {
                    "status": 403,
                    "headers": {
                      "Content-Type": [
                        "text/plain"
                      ]
                    },
                    "entity": "Request forbidden"
                  }
                }
              }
            }
          ]
        }
      }
    ]
}
```

Notice the following features of the file:

- The `prefix` sets the base of the administrative route to the default value `/openig`. The Studio endpoint is therefore `/openig/studio`.
- The `mode` is `development`, so by default the Studio endpoint is open and unfiltered.
- The SingleSignOnFilter called StudioProtectionFilter in the heap is used by IG to protect access to the Studio endpoint.

  When a request to access Studio doesn't have valid authentication, the SingleSignOnFilter redirects the request to AM for authentication.
- The AmService in the heap uses the Java agent configured on AM to act on behalf of IG for authentication.

6. Log out of AM, and restart IG.
7. Access Studio on http://openig.example.com:8080/openig/studio.

   The SingleSignOnFilter redirects the request to AM for authentication.
8. Log in to AM with username `demo`, password `Ch4ng31t`.

   The Studio welcome page is displayed.

2.10. Setting Up AM

This section contains procedures for setting up items in AM that you can use in many of the tutorials in this guide. For more information about setting up AM, see the *Access Management Docs*.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Set Up a Sample User In AM

Follow these steps to add an example user to the AM configuration:

1. In the AM console, select the top level realm, and then select Identities.
2. Click Add Identity and add a user with the following values:
   - ID/username: `george`
   - First name: `george`
   - Last name: `costanza`
   - Password: `C0stanza`
   - Email Address: `george@example.com`
   - Employee number: `123`

To Set Up a Java Agent in AM

Follow these steps to add an AM Java agent that acts on behalf of IG to authenticate with AM, get user profiles, and communicate WebSocket notifications from AM to IG:

1. In the AM console, select the top level realm, and then select select Applications > Agents > Java (or J2EE in earlier versions of AM).
2. Add an agent with the following values:
   - Agent ID: `ig_agent`
   - Agent URL: `http://openig.example.com:8080/agentapp`
   - Server URL: `http://openam.example.com:8088/openam`
   - Password: `password`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 3

Getting Login Credentials From Data Sources

In "*First Steps*" in the *Getting Started Guide* you learned how to configure IG to proxy traffic and capture request and response data. You also learned how to configure IG to use a static request to log in with hard-coded credentials. In this chapter, you will learn to:

- Configure IG to look up credentials in a file
- Configure IG to look up credentials in a relational database

3.1. Before You Start

Before you start, prepare IG and the sample application as described in "*First Steps*" in the *Getting Started Guide*.

3.2. Log in With Credentials From a File

This sample shows you how to configure IG to get credentials from a file.

The sample uses a comma-separated value file, `userfile`:

```
username,password,fullname,email
george,C0stanza,George Costanza,george@example.com
kramer,N3wman12,Kramer,kramer@example.com
bjensen,H1falutin,Babs Jensen,bjensen@example.com
demo,Ch4ng31t,Demo User,demo@example.com
kvaughan,B5ibery12,Kirsten Vaughan,kvaughan@example.com
scarter,S9rain12,Sam Carter,scarter@example.com
```

IG looks up the user credentials based on the user's email address. IG uses a `FileAttributesFilter` to look up the credentials.

Follow these steps to set up log in with credentials from a file:

1. Add the user file on your system:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ vi /tmp/userfile
$ cat /tmp/userfile
username,password,fullname,email
george,C0stanza,George Costanza,george@example.com
kramer,N3wman12,Kramer,kramer@example.com
bjensen,H1falutin,Babs Jensen,bjensen@example.com
demo,Ch4ng31t,Demo User,demo@example.com
kvaughan,B5ibery12,Kirsten Vaughan,kvaughan@example.com
scarter,S9rain12,Sam Carter,scarter@example.com
```

On Windows systems, use an appropriate path such as `C:\Temp\userfile`.

2. Add a new route to the IG configuration to obtain the credentials from the file.

   To add the route, add the following route configuration file as `$HOME/.openig/config/routes/02-file.json`:

```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "PasswordReplayFilter",
          "config": {
            "loginPage": "${true}",
            "credentials": {
              "type": "FileAttributesFilter",
              "config": {
                "file": "/tmp/userfile",
                "key": "email",
                "value": "george@example.com",
                "target": "${attributes.credentials}"
              }
            },
            "request": {
              "method": "POST",
              "uri": "http://app.example.com:8081/login",
              "form": {
                "username": [
                  "${attributes.credentials.username}"
                ],
                "password": [
                  "${attributes.credentials.password}"
                ]
              }
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  },
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
  "condition": "${matches(request.uri.path, '^/file')}"
}
```

On Windows, the file name should be `%appdata%\OpenIG\config\routes\02-file.json`.

Notice the following features of the new route:

- The `FileAttributesFilter` specifies the file to access, the key and value to look up to retrieve the user's record, and where to store the results in the request context attributes map.
- The `PasswordReplayFilter` creates a request by retrieving the username and password from the attributes map and replacing your browser's original HTTP GET request with an HTTP POST login request that contains the credentials to authenticate.
- The route matches requests to `/file`.

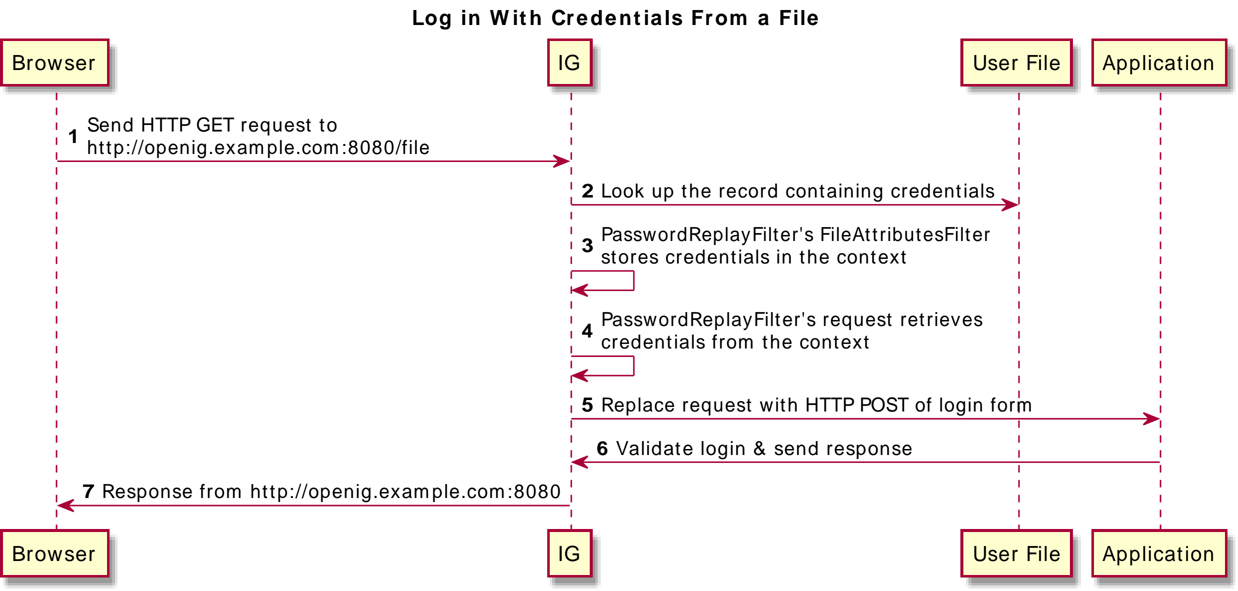
3. On Windows systems, edit the path name to the user file.

Now browse to http://openig.example.com:8080/file.

If everything is configured correctly, IG logs you in as George.

What's happening behind the scenes?

Log in With Credentials From a File



- IG intercepts the browser's HTTP GET request, which matches the route condition.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- The `PasswordReplayFilter`'s `FileAttributesFilter` looks up credentials in a file, and stores the credentials it finds in the request context attributes map.
- The `PasswordReplayFilter`'s request pulls the credentials out of the attributes map, builds the login form, and performs the HTTP POST request to the sample application.
- The sample application validates the credentials, and responds with a profile page. IG then passes the response from the sample application to your browser.

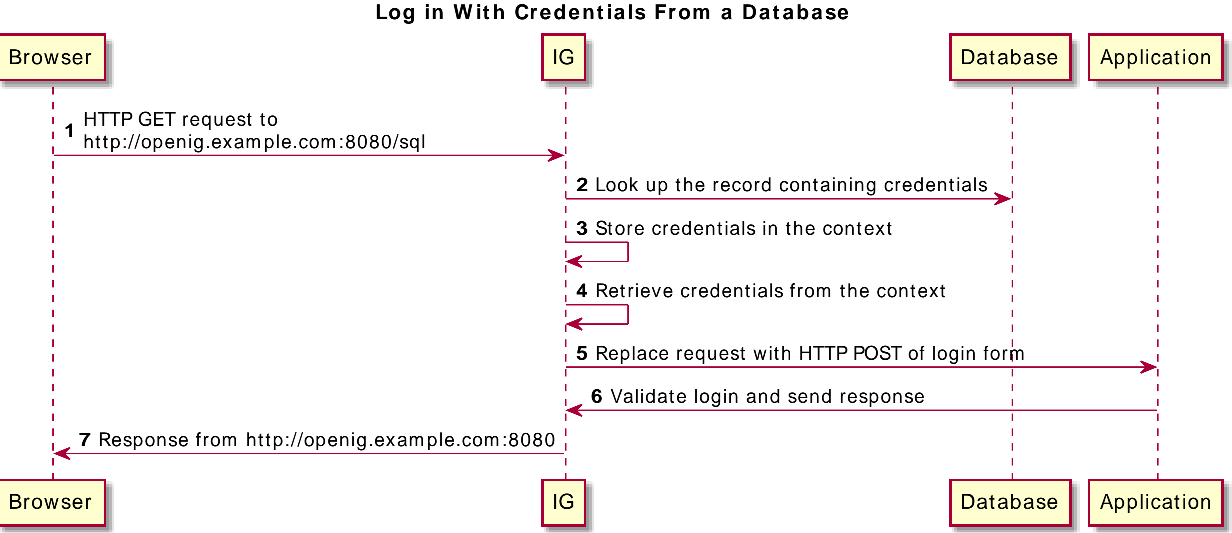
3.3. Log in With Credentials From a Database

This section describes how to configure IG to get credentials from H2. IG also works with other database software, but this example is tested with Jetty and H2 1.4.196.

IG relies on the application server where it runs to connect to the database. Configuring IG to retrieve data from a database is therefore a question of configuring the application server to connect to the database, and configuring IG to choose the appropriate data source, and to send the appropriate SQL request to the database. As a result, the IG configuration depends more on the data structure than on any particular database drivers or connection configuration.

The following flow chart illustrates the steps:

Log in With Credentials From a Database



- IG intercepts the browser's HTTP GET request, which matches the route condition.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- The `SqlAttributesFilter` in `PasswordReplayFilter` looks up credentials in H2, and stores them in the request context attributes map.
- The `request` in `PasswordReplayFilter` pulls the credentials out of the attributes map, builds the login form, and performs the HTTP POST request to the sample application.
- The sample application validates the credentials, and responds with a profile page. IG then passes the response to the browser.

To Prepare the Database

1. Create the following user data in `/tmp/userfile`:

```
username,password,fullname,email
george,C0stanza,George Costanza,george@example.com
kramer,N3wman12,Kramer,kramer@example.com
bjensen,H1falutin,Babs Jensen,bjensen@example.com
demo,Ch4ng31t,Demo User,demo@example.com
kvaughan,B5ibery12,Kirsten Vaughan,kvaughan@example.com
scarter,S9rain12,Sam Carter,scarter@example.com
```

2. Download and unpack the H2 database, and then start H2:

```
$ sh /path/to/h2/bin/h2.sh
```

H2 starts, listening on port 8082, and opens the H2 Console in a browser.

3. Select the following options in the H2 Console, and then select Connect to access the console:

   - Saved Settings: `Generic H2 (Server)`

     This option sets the Driver Class, `org.h2.Driver`, the JDBC URL, `jdbc:h2:tcp://localhost/~/test`, and the User Name, `sa`.

   - Password: `password`

4. In the console, create the user table, using `/tmp/userfile`:

```
DROP TABLE IF EXISTS USERS;
CREATE TABLE USERS AS SELECT * FROM CSVREAD('/tmp/userfile');
```

5. Run the following command in the console to check that the table contains the same users as the file:

```
SELECT * FROM users;
```

To Prepare Jetty's Connection to the Database

Follow these steps to enable Jetty to connect to the database:

1. Configure Jetty for JNDI.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Stop Jetty and add the following lines to `start.ini` in `/path/to/jetty/start.ini` to configure Jetty for JNDI:

```
# ===========================================================
# Enable JNDI
# -----------------------------------------------------------
OPTIONS=jndi

# ===========================================================
# Enable additional webapp environment configurators
# -----------------------------------------------------------
OPTIONS=plus
etc/jetty-plus.xml
```

For more information, see the Jetty documentation on *Configuring JNDI*.

2. Copy the H2 library to the classpath for Jetty:

```
$ cp /path/to/h2/bin/h2-*.jar /path/to/jetty/lib/ext/
```

3. Define a JNDI resource for H2 in `/path/to/jetty/etc/jetty.xml`:

```
<New id="jdbc/forgerock" class="org.eclipse.jetty.plus.jndi.Resource">
  <Arg></Arg>
  <Arg>jdbc/forgerock</Arg>
  <Arg>
    <New class="org.h2.jdbcx.JdbcDataSource">
      <Set name="Url">jdbc:h2:tcp://localhost/~/test</Set>
      <Set name="User">sa</Set>
      <Set name="Password">password</Set>
    </New>
  </Arg>
</New>
```

4. Add a resource reference to the data source in `/path/to/jetty/etc/webdefault.xml`:

```
<resource-ref>
    <res-ref-name>jdbc/forgerock</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

5. Restart Jetty to take the configuration changes into account.

To Prepare the IG Configuration

Add a new route to the IG configuration to look up credentials in the database:

1. Add the following route configuration file as `$HOME/.openig/config/routes/03-sql.json`:

```
{
  "handler": {
    "type": "Chain",
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    "config": {
      "filters": [
        {
          "type": "PasswordReplayFilter",
          "config": {
            "loginPage": "${true}",
            "credentials": {
              "type": "SqlAttributesFilter",
              "config": {
                "dataSource": "java:comp/env/jdbc/forgerock",
                "preparedStatement":
                "SELECT username, password FROM users WHERE email = ?;",
                "parameters": [
                  "george@example.com"
                ],
                "target": "${attributes.sql}"
              }
            },
            "request": {
              "method": "POST",
              "uri": "http://app.example.com:8081/login",
              "form": {
                "username": [
                  "${attributes.sql.USERNAME}"
                ],
                "password": [
                  "${attributes.sql.PASSWORD}"
                ]
              }
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  },
  "condition": "${matches(request.uri.path, '^/sql')}"
}
```

On Windows, add the file as `%appdata%\OpenIG\config\routes\03-sql.json`.

2. Notice the following features of the route:

    - The route matches requests to `/sql`.
    - The `SqlAttributesFilter` in `PasswordReplayFilter` specifies the data source to access, a prepared statement to look up the user's record, a parameter to pass into the statement, and where to store the search results in the request context attributes map.
    - The request in `PasswordReplayFilter` retrieves the username and password from the attributes map and replaces the browser's original HTTP GET request with an HTTP POST login request that contains the credentials to authenticate.

      The request is for `username, password`, but H2 returns the fields as `USERNAME` and `PASSWORD`. The configuration reflects this difference.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- The sample application validates the credentials, and responds with a profile page, which IG then passes to your browser.

To Test the Setup

Before you start, set up H2, Jetty, and IG as described in the previous procedures, make sure that sample application is running, and log out of AM.

- Access the route on http://openig.example.com:8080/sql.

  IG logs you in to the sample application as George.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 4

Getting Login Credentials From AM

Use IG with AM's password capture and replay feature to bring SSO to legacy web applications, without the need to edit, upgrade, or recode. This feature helps you to integrate legacy web applications with other applications, such as SharePoint, using the same user identity.

AM's password capture and replay feature captures the password during authentication, encrypts it, and adds to the session. IG's CapturedUserPasswordFilter then decrypts the password and uses it for authentication to a legacy web application.

For an alternative configuration using an AM policy agent instead of IG's CapturedUserPasswordFilter, see the documentation for earlier versions of IG.

4.1. About Password Capture and Replay

The following figure illustrates the flow of requests when an unauthenticated user accesses a protected application. After authenticating with AM, the user is logged into the application with the username and password from the AM login session.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Data Flow to Log In To a Protect Application With AM Credentials

Browser | IG | AM | Sample App

1 Send an HTTP GET request to http://openig.example.com:8080/replay

2 User not authenticated: -redirect request to AM for authentication

3 Authenticate and capture credentials in AM session

4 Redirect browser back to the protected application

5 Resend the HTTP GET request to http://openig.example.com:8080/replay

6 User authenticated: -retrieve encrypted AM password -decrypt it -place it in the context

7 Replace request with an HTTP POST of login form

8 Validate login

9 Return response page showing user is logged in

10 Return the response page showing user is logged in

Browser | IG | AM | Sample App

- IG intercepts the browser's HTTP GET request.
- Because the user is not authenticated, the SingleSignOnFilter redirects the user to AM for authentication.
- AM authenticates the user, capturing the login credentials, and storing the encrypted password in the user's AM session.
- AM redirects the browser back to the protected application.
- IG intercepts the browser's HTTP GET request again:
    - The user is now authenticated, so IG's SingleSignOnFilter passes the request to the CapturedUserPasswordFilter.
    - The CapturedUserPasswordFilter checks that the SessionInfoContext `${contexts.amSession.properties.sunIdentityUserPassword}` is available and not `null`. It then decrypts the password and stores it in the CapturedUserPasswordContext, at `${contexts.capturedPassword}`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- The PasswordReplayFilter uses the username and decrypted password in the context to replace the request with an HTTP POST of the login form.
- The sample application validates the credentials.
- The sample application responds with the user's profile page.
- IG then passes the response from the sample application to the browser.

4.2. Setting Up the Example

Before you start:

- Prepare IG and the sample app as described in "*First Steps*" in the *Getting Started Guide*
- Install and configure AM on http://openam.example.com:8088/openam, using the default configuration.

The following table summarizes the steps to set up this example.

Tasks for Configuring This Example

Task	See Section(s)
Create a key to decrypt the password shared by AM and IG. This example uses DES as the decryption algorithm. From AM 6, CapturedUserPasswordFilter can use the stronger algorithm AES to decrypt the AM password.	"To Create a DES Shared Key In IG"
Configure AM password capture, and add the DES key to the AM configuration.	"To Configure Password Capture in AM"
Create a route in IG to handle the requests.	"To Configure Password Replay in IG"
Test the setup.	"To Test the Setup"

To Create a DES Shared Key In IG

Before AM communicates passwords to IG, it encrypts them with a key. IG then uses the key to decrypt the shared passwords.

This example uses DES as the decryption algorithm. For more information about DES shared keys, see DesKeyGenHandler(5) in the *Configuration Reference*.

The shared key is sensitive information. If it is possible for others to inspect the response, make sure you use HTTPS to protect the communication.

Use this procedure to generate a DES shared key in IG that you can use later in the AM and IG configuration:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

1. Add the following route to the IG configuration as `$HOME/.openig/config/routes/04-keygen.json`. On Windows, add the route as `%appdata%\OpenIG\config\routes\04-keygen.json`.

```
{
  "name": "04-keygen",
  "handler": {
    "type": "DesKeyGenHandler"
  },
  "condition": "${matches(request.uri.path, '^/keygen') and (matches(contexts.client.remoteAddress,
 ':1') or matches(contexts.client.remoteAddress, '127.0.0.1'))}"
}
```

2. Call the route to generate a key:

```
$ curl http://localhost:8080/keygen
{"key":"1A+BCdEfGhI="}
```

   In this example, the key is `1A+BCdEfGhI=`.

3. Set an IG environment variable for the key:

```
export DESKEY='1A+BCdEfGhI='
```

   IG uses the default SecretsService to retrieve the value from the environment variable. For more information, see Secrets in the *Configuration Reference*.

To Configure Password Capture in AM

1. (For AM 6.5.3 and earlier versions) Select Identities > demo, and set the demo user password to `Ch4ng31t`.

2. (For AM 6.5.3 and later versions) Select Services, and add a Validation Service with the following URLs for IG:

   - `http://openig.example.com:8080/*`
   - `http://openig.example.com:8080/*?*`

3. (For AM 6.5.3 and later versions) Select Applications > Agents > Identity Gateway, and add an agent with the following values:

   - Agent ID: `ig_agent`
   - Password: `password`

   Leave all other values as default.

   (For AM 6.5.2 and earlier versions) Set up an agent as described in "To Set Up a Java Agent in AM".

4. Update the Authentication Post Processing Classes for password replay:

   a. Select Authentication > Settings > Post Authentication Processing.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

   b. Add the following class to Authentication Post Processing Classes: `com.sun.identity.authentication.spi.ReplayPasswd`.

5. Add the key you created in "To Create a DES Shared Key In IG" to the AM configuration:

   a. In the AM console, select DEPLOYMENT > Servers, and then select the AM server name. In some earlier releases, select Configuration > Servers and Sites.

   b. Select Advanced, and add the property `com.sun.am.replaypasswd.key` with the value of the DES shared key.

6. Add `sunIdentityUserPassword` to the session whitelist, so that the property can be read from the session:

   a. Select Services > Session Property Whitelist Service.

   b. Add `sunIdentityUserPassword` as a whitelisted session property name.

7. Add `example.com` as an AM cookie domain:

   a. In AM, select Configure > Global Services > Platform.

   b. Add the domain `example.com`.

   By default, AM sets host-based cookies. After authentication with AM, requests can be redirected to AM instead of to the resource.

8. Restart AM.

To Configure Password Replay in IG

1. Set an environment variable for the Java agent password, and then restart IG:

```
$ export AGENT_SECRET_ID='cGFzc3dvcmQ='
```

   The password is retrieved by the default SystemAndEnvSecretStore, and must be base64-encoded.

2. In IG, add the following route as `$HOME/.openig/config/routes/04-replay.json` (on Windows, `%appdata%\OpenIG\config\routes\04-replay.json`).

```
{
  "name": "04-replay",
  "condition": "${matches(request.uri.path, '^/replay')}",
  "heap": [
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "agent" : {
          "username" : "ig_agent",
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
          "passwordSecretId" : "agent.secret.id"
        },
        "url": "http://openam.example.com:8088/openam/",
        "version": "6.5"
      }
    },
    {
      "name": "CapturedUserPasswordFilter",
      "type": "CapturedUserPasswordFilter",
      "config": {
        "ssoToken": "${contexts.ssoToken.value}",
        "keySecretId": "DESKEY",
        "amService": "AmService-1"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "SingleSignOnFilter",
          "config": {
            "amService": "AmService-1"
          }
        },
        {
          "type": "PasswordReplayFilter",
          "config": {
            "loginPage": "${true}",
            "credentials": "CapturedUserPasswordFilter",
            "request": {
              "method": "POST",
              "uri": "http://app.example.com:8081/login",
              "form": {
                "username": [
                  "${contexts.ssoToken.info.uid}"
                ],
                "password": [
                  "${contexts.capturedPassword.value}"
                ]
              }
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

3. If necessary, change the value of `version` for AmService to your version of AM.
4. Notice the following features of the route:
   - The route matches requests to `/replay`.
   - The first filter in the chain is the SingleSignOnFilter.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

If the request does not have a valid iPlanetDirectoryPro cookie, the SingleSignOnFilter redirects the request to AM for authentication.

If the request already has a valid iPlanetDirectoryPro cookie, or after authenticating with AM to get a valid iPlanetDirectoryPro cookie, the SingleSignOnFilter passes the request to the next filter. The SingleSignOnFilter stores the cookie value in an `SsoTokenContext`.

See SingleSignOnFilter(5) in the *Configuration Reference*.

- The next filter in the chain is the PasswordReplayFilter.

  The PasswordReplayFilter uses the CapturedUserPasswordFilter declared in the heap to retrieve the AM password from session properties. The CapturedUserPasswordFilter uses the DES shared key to decrypt the password, and then makes it available in a CapturedUserPasswordContext.

  The PasswordReplayFilter retrieves the username and password from the context. It replaces the browser's original HTTP GET request with an HTTP POST login request containing the credentials to authenticate to the sample application.

  See PasswordReplayFilter(5) in the *Configuration Reference* and CapturedUserPasswordFilter(5) in the *Configuration Reference*.

To Test the Setup

1. Log out of AM if you are logged in.
2. Access the route on http://openig.example.com:8080/replay.

   You should be redirected to the AM login page.
3. Log in with username `demo`, password `Ch4ng31t`.

   The request is redirected to the sample application.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 5
Single Sign-On and Cross-Domain Single Sign-On

This chapter describes the support for single-sign on (SSO) and cross-domain single sign-on (CDSSO) provided by IG.

5.1. About SSO Using the SingleSignOnFilter

In SSO using the SingleSignOnFilter, IG processes a request using authentication provided by AM. IG and the authentication provider must run on the same domain.

For an example where SSO with the SingleSignOnFilter is used, see "Setting Up IG as a PEP".

The following sequence diagram shows the flow of information during SSO between IG and AM as the authentication provider.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Flow of Information for SSO

Browser | IG ig.example.com | AM am.example.com | Sample App app.example.com

1 Request to access sample app

1. Unauthenticated browser redirected to AM for authentication

2 Redirect browser to AM

3 Authentication

4 Create SSO token

2. Authenticated browser redirect to original URI

5 Redirect to original URI, with cookie containing SSO token

3. Request forwarded to application

6 Follow redirect, with cookie

7 Request SessionInfo and add to request
Store SSO token in SsoTokenContext

8 Forward request

9 Return response

10 Return response

Browser | IG ig.example.com | AM am.example.com | Sample App app.example.com

- The browser sends an unauthenticated request to access the sample app.
- IG intercepts the request, and redirects the browser to AM for authentication.
- AM authenticates the user, creates an SSO token.
- AM redirects the request back to the original URI with the token in a cookie, and the browser follows the redirect to IG.
- IG validates the token it gets from the cookie. It then adds the SessionInfo to the request, and stores the SSO token in the context for use by downstream filters and handlers.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- IG forwards the request to the sample app, and the sample app returns the requested resource to the browser.

5.2. About CDSSO Using the CrossDomainSingleSignOnFilter

The SSO mechanism described in the previous section can be used when IG and AM are running in the same domain. When IG and AM are running in different domains, AM cookies are not visible to IG because of the same-origin policy.

CDSSO using the CrossDomainSingleSignOnFilter, provides a mechanism to push tokens issued by AM to IG running in a different domain.

The following sequence diagram shows the flow of information between IG, AM, and the sample app during CDSSO. In this example, AM is running on `am.example.com`, and IG is running on `ig.ext.com`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Flow of Information for CDSSO

Browser | IG ig.ext.com | AM am.example.com | Sample App app.example.com

1 Request to access sample app

1. Unauthenticated browser redirected to AM for authentication

2 Create a nonce and store in session
Redirect browser to AM

3 Authentication

4 Create CDSSO token

2. Authenticated browser redirect to original URI

5 Autosubmit form to redirectEndpoint, and include CDSSO token as form parameter

6 Submit parameters to redirectEndpoint

7 Check nonce to confirm that
authentication was initiated by IG
Construct cookie containing CDSSO token

8 Redirect to original URI, with cookie

3. Request forwarded to application

9 Follow redirect, with cookie

10 Request SessionInfo and add to request
Store SSO token in SsoTokenContext
Store CDSSO token in CdSsoContext

11 Forward request

12 Return response

13 Return response

Browser | IG ig.ext.com | AM am.example.com | Sample App app.example.com

- The browser sends an unauthenticated request to access the sample app.
- IG intercepts the request, and redirects the browser to AM for authentication.
- AM authenticates the user and creates a CDSSO token.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- AM responds to a successful authentication with an HTML autosubmit form containing the issued token.
- The browser loads the HTML and autosubmit form parameters to the IG callback URL for the redirect endpoint.
- IG checks the nonce found inside the CDSSO token to confirm that the callback comes from an authentication initiated by IG. IG then constructs a cookie, and fulfills it with a cookie name, path, and domain, using the CrossDomainSingleSignOnFilter property `authCookie`. The domain must match that set in the AM J2EE agent.
- IG redirects the request back to the original URI, with the cookie, and the browser follows the redirect back to IG.
- IG validates the token it gets from the cookie. It adds SessionInfo to the request, and stores the SSO token and CDSSO token in the contexts for use by downstream filters and handlers.
- IG forwards the request to the sample app, and the sample app returns the requested resource to the browser.

5.3. Setting Up CDSSO

This section describes how to set up CDSSO, where AM on `openam.example.com` authenticates users that are processed by IG on `openig.ext.com`.

CDSSO in AM must be configured as a Java EE policy agent. For more information, see Implementing Cross-Domain Single Sign-On in the Access Management *Authentication and Single Sign-On Guide*.

Before you start, install and configure AM on http://openam.example.com:8088/openam, with the default configuration. If you use a different configuration, substitute in the tutorial accordingly.

To Set Up AM for CDSSO

1. (For AM 6.5.3 and earlier versions) Select Identities > demo, and set the demo user password to `Ch4ng31t`.
2. (For AM 6.5.3 and later versions) Select Services, and add a Validation Service with the following URLs for IG:
   - `http://openig.ext.com:8080/*`
   - `http://openig.ext.com:8080/*?*`
3. In the top-level realm, select Applications > Agents > Java (or J2EE in earlier versions of AM).
4. Add an agent with the following values:
   - Agent ID: `ig_agent_cdsso`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Agent URL: `http://openig.ext.com:8080/agentapp`
- Server URL: `http://openam.example.com:8088/openam`
- Password: `password`

5. Select the agent you just created, and on the SSO tab select set the following values:
    - Cross Domain SSO: Deselect this option
    - CDSSO Redirect URI: `/home/cdsso/redirect`
6. Add `example.com` as an AM cookie domain:
    a. In AM, select Configure > Global Services > Platform.
    b. Add the domain `example.com`.

    By default, AM sets host-based cookies. After authentication with AM, requests can be redirected to AM instead of to the resource.

To Set Up IG for CDSSO

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

Before you start, make sure that IG is running in development mode on `http://openig.ext.com:8080`, the sample application on `http://app.example.com:8081`, and AM on `http://openam.example.com:8088`.

1. In IG Studio, create a route:
    a. Browse to http://openig.example.com:8080/openig/studio, and select ⊕ Protect an Application.
    b. Choose ☰ Structured to use the predefined menus and templates.
2. Select Advanced options, and create a route with the following options:
    - Base URI: `http://app.example.com:8081`
    - Condition: Path: `/home/cdsso`
    - Name: `cdsso`
3. Configure authentication:
    a. Select 👤 Authentication.
    b. Select Cross-Domain Single Sign-On, and enter the following information to reflect the configuration in "To Set Up AM for CDSSO", and then save the settings:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- AM service: Configure an AM service to use for authentication:
  - URI: `http://openam.example.com:8088/openam`
  - Version: The version of the AM instance, for example, `6.5`.
  - Agent: The credentials of the agent you created in AM.
    - Username: `ig_agent_cdsso`
    - Password: `password`
- Redirect endpoint: `/home/cdsso/redirect`
- Authentication cookie:
  - Path: `/home`

Leave all other values as default.

4. On the top-right of the screen, select ⋮ and Display to review the route.

The following route should be displayed:

```
{
  "name": "cdsso",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/cdsso')}",
  "heap": [
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "http://openam.example.com:8088/openam",
        "realm": "/",
        "ssoTokenHeader": "iPlanetDirectoryPro",
        "version": "6.5",
        "agent": {
          "username": "ig_agent_cdsso",
          "password": "password"
        },
        "sessionCache": {
          "enabled": false
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "CrossDomainSingleSignOnFilter-1",
          "type": "CrossDomainSingleSignOnFilter",
          "config": {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
            "redirectEndpoint": "/home/cdsso/redirect",
            "authCookie": {
              "path": "/home",
              "name": "ig-token-cookie"
            },
            "amService": "AmService-1"
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

> **Note**
>
> If necessary, change the value of `version` for AmService to your version of AM.

5. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

To Test the Setup

1. Log out of AM if you are logged in, and clear any cookies.
2. Browse to http://openig.ext.com:8080/home/cdsso.

   The CrossDomainSingleSignOnFilter redirects the request to AM for authentication.
3. Log in to AM as user `demo`, password `Ch4ng31t`.

   When you have authenticated, AM calls `/home/cdsso/redirect`, and includes the CDSSO token.

   The CrossDomainSingleSignOnFilter then passes the request to sample app, which returns the profile page.

5.4. Using WebSocket Notifications to Evict the Session Cache

When WebSocket notifications are enabled, IG receives notifications whenever a user logs out of AM, or when an AM session is modified, closed, or times out.

This section describes how to change the configuration of the previous examples to evict entries related to the event from the session cache. For information about WebSocket notifications, see "WebSocket Notification Service" in the *Configuration Reference*.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Evict Entries From the Session Cache

Before you start, set up and test one of the previous examples for SSO.

- In the AmService heap object of your route, enable `sessionCache`:

  ```
  "sessionCache": {
    "enabled" : true
  }
  ```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 6
Enforcing Policy Decisions From AM

This chapter describes how to set up IG as a policy enforcement point, with AM as a policy decision point. It provides an example of how to enforce a policy decision from AM, and an example of upgrading a session to a higher authentication level.

For more information about authentication and session upgrade, see AM's *Authentication and Single Sign-On Guide* .

6.1. About IG As a PEP With AM As PDP

The following terms are used in access management:

- *Policy Decision Point* (PDP): Entity that evaluates access rights and then issues authorization decisions.
- *Policy Enforcement Point* (PEP): Entity that intercepts a request for a resource and then enforces policy decisions from a PDP.

IG as a PEP intercepts requests for a resource, and provides information about the request to AM as a PDP.

AM evaluates requests based on their context and the configured policies. AM then returns decisions that indicate what actions are allowed or denied, as well as any advices, subject attributes, or static attributes for the specified resources.

After a policy decision, IG continues to process requests as follows:

- If the request is allowed, processing continues.
- If the request is denied with advices, IG checks whether it can respond to the advices. If IG can respond, it sends a redirect and information about how to meet the conditions in the advices.

  By default, the request is redirected to AM. If the SingleSignOnFilter property `loginEndpoint` is configured, the request is redirected to that endpoint.
- If the request is denied without advice, or if IG cannot respond to the advice, IG forwards the request to a `failureHandler` declared in the `PolicyEnforcementFilter`. If there is no `failureHandler`, IG returns a 403 Forbidden.
- If an error occurs during the process, IG returns 500 Internal Server Error.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

In AM, administrators can maintain centralized, fine-grained, declarative policies to manage who can access what resources, and under what conditions. Policies can be managed separately by AM realm and by AM application.

AM provides a REST API for authorized users to request policy decisions. IG provides a PolicyEnforcementFilter that uses the REST API. For information, see PolicyEnforcementFilter(5) in the *Configuration Reference*.

6.2. Enforcing AM Policy Decisions In the Same Domain

This section gives an example of how to set up IG as a PEP, requesting policy decisions from AM as a PDP.

Tasks for Configuring Policy Enforcement

Task	See Section(s)
Set up an AM policy to allow authenticated users to access the sample application, and set up an AM agent with permission to request policy decisions.	"To Set Up AM As a PDP"
Configure IG as a PEP in Studio.	"To Set Up IG as a PEP"
Test the setup.	"To Test the Setup"

6.2.1. Setting Up AM As a PDP

This section describes how to create a policy in AM and configure an agent that can request policy decisions.

Before you start, install and configure AM on http://openam.example.com:8088/openam, with the default configuration. If you use a different configuration, substitute in the tutorial accordingly.

To Set Up AM As a PDP

Before you start, install and configure AM on http://openam.example.com:8088/openam, with the default configuration. If you use a different configuration, substitute in the tutorial accordingly.

1. (For AM 6.5.3 and earlier versions) Select Identities > demo, and set the demo user password to `Ch4ng31t`.

2. (For AM 6.5.3 and later versions) Select Services, and add a Validation Service with the following URLs for IG:

   - `http://openig.example.com:8080/*`

   - `http://openig.example.com:8080/*?*`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

3. (For AM 6.5.3 and later versions) Select Applications > Agents > Identity Gateway, and add an agent with the following values:

   - Agent ID: `ig_agent`
   - Password: `password`

   Leave all other values as default.

   (For AM 6.5.2 and earlier versions) Set up an agent as described in "To Set Up a Java Agent in AM".

4. Add a policy:

   a. Select Authorization > Policy Sets.

   b. Add a policy set with the following values, and then select Create:

      - Id: `PEP-SSO`
      - Resource Types: `URL`

   c. In the new policy set, add a policy with the following values, and then select Create:

      - Name: `IG Policy SSO`
      - Resource Type: `URL`
      - Resource pattern: `*://*:*/*`
      - Resource value: `http://app.example.com:8081/home/pep-sso`

      This policy protects the home page of the sample application.

   d. On the Actions tab, add an action to allow HTTP `GET`, and then save your changes.

   e. On the Subjects tab, remove any default subject conditions, add a subject condition for all `Authenticated Users`, and then save your changes.

5. Add `example.com` as an AM cookie domain:

   a. In AM, select Configure > Global Services > Platform.

   b. Add the domain `example.com`.

   By default, AM sets host-based cookies. After authentication with AM, requests can be redirected to AM instead of to the resource.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

6.2.2. Setting Up IG as a PEP

This section describes how to set up IG to configure policy enforcement, where the user-agent is redirected to AM for authentication.

To configure IG without using Studio, add the route in the following procedure to the IG configuration as `$HOME/.openig/config/routes/04-pep.json`. On Windows, add the route as `%appdata%\OpenIG\config\routes\04-pep.json`.

For an example route that uses `claimsSubject` instead of `ssoTokenSubject` to identify the subject, see "Example Policy Enforcement Using claimsSubject" in the *Configuration Reference*.

To Set Up IG as a PEP

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

1. In IG Studio, create a route:
    a. Browse to http://openig.example.com:8080/openig/studio, and select ⊕ Protect an Application.
    b. Choose ☰ Structured to use the predefined menus and templates.
2. Select Advanced options, and create a route with the following options:
    - Base URI: `http://app.example.com:8081`
    - Condition: Path: `/home/pep-sso`
    - Name: `pep-sso`
3. Configure authentication:
    a. Select 👤 Authentication.
    b. Select Single Sign-On, and enter the following information:
        - AM service:
            - URI: `http://openam.example.com:8088/openam`
            - Version: The version of the AM instance, for example, `6.5`.
            - Agent: The credentials of the Java agent you created in AM.
                - Username: `ig_agent`
                - Password: `password`

        Leave all other values as default.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

4. Configure a PolicyEnforcementFilter:

   a. Select Authorization.

   b. Select AM Policy Enforcement, and then select the following options:

      - Access Management configuration:
        - AM server: `http://openam.example.com:8088/openam (/)`.
      - Access Management policy endpoint:
        - Policy set: `PEP-SSO`
        - AM SSO token: `${contexts.ssoToken.value}`

      Leave all other values as default.

5. On the top-right of the screen, select and Display to review the route.

   The following route should be displayed:

```
{
  "name": "pep-sso",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/pep-sso')}",
  "heap": [
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "http://openam.example.com:8088/openam",
        "realm": "/",
        "ssoTokenHeader": "iPlanetDirectoryPro",
        "version": "6.5",
        "agent": {
          "username": "ig_agent",
          "password": "password"
        },
        "sessionCache": {
          "enabled": false
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "SingleSignOnFilter-1",
          "type": "SingleSignOnFilter",
          "config": {
            "amService": "AmService-1"
          }
        },
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
        {
          "name": "PolicyEnforcementFilter-1",
          "type": "PolicyEnforcementFilter",
          "config": {
            "pepRealm": "/",
            "application": "PEP-SSO",
            "ssoTokenSubject": "${contexts.ssoToken.value}",
            "amService": "AmService-1"
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

6. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

6.2.3. Testing the Setup

To Test the Setup

1. If you are logged in to AM, log out.

2. Go to http://openig.example.com:8080/home/pep-sso.

   Because you are not authenticated to AM, the request does not contain a cookie with an SSO token. The SingleSignOnFilter redirects you to AM for authentication.

3. Log in to AM as user `demo`, password `Ch4ng31t`.

   When you have authenticated, AM redirects you back to the request URL, and IG requests a policy decision using the iPlanetDirectoryPro cookie value.

   AM returns a policy decision that grants access to the sample application.

6.3. Enforcing AM Policy Decisions Cross-Domain

6.3.1. Setting Up AM As a PDP

This section describes how to create a policy in AM and configure an agent that can request policy decisions, when IG and AM are in different domains.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Set Up AM As a PDP In a Different Domain

Before you start, install and configure AM on http://openam.example.com:8088/openam, with the default configuration. If you use a different configuration, substitute in the tutorial accordingly.

1. (For AM 6.5.3 and earlier versions) Select Identities > demo, and set the demo user password to `Ch4ng31t`.
2. (For AM 6.5.3 and later versions) Select Services, and add a Validation Service with the following URLs for IG:
   - `http://openig.ext.com:8080/*`
   - `http://openig.ext.com:8080/*?*`
3. In the AM console, add a Java agent:
   a. In the top-level realm, select Applications > Agents > Java (or J2EE in earlier versions of AM).
   b. Add an agent with the following values:
      - Agent ID: `ig_agent_cdsso`
      - Agent URL: `http://openig.ext.com:8080/agentapp`
      - Server URL: `http://openam.example.com:8088/openam`
      - Password: `password`
   c. On the SSO tab, select set the following values:
      - Cross Domain SSO: Deselect this option
      - CDSSO Redirect URI: `/home/pep-cdsso/redirect`
4. Add a policy:
   a. Select Authorization > Policy Sets.
   b. Add a policy set with the following values, and then select Create:
      - Id: `PEP-CDSSO`
      - Resource Types: `URL`
   c. In the new policy set, add a policy with the following values, and then select Create:
      - Name: `IG Policy CDSSO`
      - Resource Type: `URL`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Resource pattern: `*://*:*/*`
- Resource value: `http://app.example.com:8081/home/pep-cdsso`

This policy protects the home page of the sample application.

d. On the Actions tab, add an action to allow HTTP `GET`, and then save your changes.

e. On the Subjects tab, remove any default subject conditions, add a subject condition for all `Authenticated Users`, and then save your changes.

5. Add `example.com` as an AM cookie domain:

a. In AM, select Configure > Global Services > Platform.

b. Add the domain `example.com`.

By default, AM sets host-based cookies. After authentication with AM, requests can be redirected to AM instead of to the resource.

6.3.2. Setting Up IG as a PEP

This section describes how to set up IG to configure policy enforcement, where the user-agent is redirected to AM for authentication.

To configure IG without using Studio, add the route in the following procedure to the IG configuration as `$HOME/.openig/config/routes/04-pep.json` (on Windows, `%appdata%\OpenIG\config\routes\04-pep.json`).

To Set Up IG as a PEP for CDSSO

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

Before you start, make sure that IG is running in development mode on `http://openig.ext.com:8080`, the sample application on `http://app.example.com:8081`, and AM on `http://openam.example.com:8088`.

1. In IG Studio, create a route:

a. Browse to http://openig.example.com:8080/openig/studio, and select ⊕ Protect an Application.

b. Choose ☰ Structured to use the predefined menus and templates.

2. Select Advanced options, and create a route with the following options:

- Base URI: `http://app.example.com:8081`
- Condition: Path: `/home/pep-cdsso`
- Name: `pep-cdsso`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

3. Configure authentication:

   a. Select Authentication.

   b. Select Cross-Domain Single Sign-On, and enter the following information:

      - AM service:
        - URI: `http://openam.example.com:8088/openam`
        - Version: The version of the AM instance, for example, `6.5`.
        - Agent: The credentials of the agent you created in AM.
          - Username: `ig_agent_cdsso`
          - Password: `password`
      - Redirect endpoint: `/home/pep-cdsso/redirect`
      - Authentication cookie:
        - Path: `/home`

      Leave all other values as default.

4. Configure a PolicyEnforcementFilter:

   a. Select Authorization.

   b. Select AM Policy Enforcement, and select the following options to reflect the configuration of the AM Java agent:

      - Access Management configuration:
        - AM server: `http://openam.example.com:8088/openam (/)`.
      - Access Management policy endpoint:
        - Policy set: `PEP-CDSSO`
        - AM SSO token ID: `${contexts.cdsso.token}`

      Leave all other values as default.

5. On the top-right of the screen, select and Display to review the route.

   The following route should be displayed:

```
{
  "name": "pep-cdsso",
  "baseURI": "http://app.example.com:8081",
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
  "condition": "${matches(request.uri.path, '^/home/pep-cdsso')}",
  "heap": [
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "http://openam.example.com:8088/openam",
        "realm": "/",
        "ssoTokenHeader": "iPlanetDirectoryPro",
        "version": "6.5",
        "agent": {
          "username": "ig_agent_cdsso",
          "password": "password"
        },
        "sessionCache": {
          "enabled": false
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "CrossDomainSingleSignOnFilter-1",
          "type": "CrossDomainSingleSignOnFilter",
          "config": {
            "redirectEndpoint": "/home/pep-cdsso/redirect",
            "authCookie": {
              "path": "/home",
              "name": "ig-token-cookie"
            },
            "amService": "AmService-1"
          }
        },
        {
          "name": "PolicyEnforcementFilter-1",
          "type": "PolicyEnforcementFilter",
          "config": {
            "pepRealm": "/",
            "application": "PEP-CDSSO",
            "ssoTokenSubject": "${contexts.cdsso.token}",
            "amService": "AmService-1"
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

6. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

6.3.3. Testing the Setup

To Test the Setup

1. Log out of AM.
2. Browse to http://openig.ext.com:8080/home/pep-cdsso.

   IG redirects you to AM for authentication.
3. Log in to AM as user `demo`, password `Ch4ng31t`.

   When you have authenticated, AM redirects you back to the request URL, and IG requests a policy decision.

   AM returns a policy decision that grants access to the sample application.

6.4. Using WebSocket Notifications to Evict the Policy Cache

When WebSocket notifications are enabled, IG receives notifications whenever AM creates, deletes, or changes a policy.

This section describes how to change the configuration of the previous examples to evict outdated entries from the policy cache. For information about WebSocket notifications, see "WebSocket Notification Service" in the *Configuration Reference*.

To Evict Entries From the Policy Cache

Before you start, set up and test one of the previous examples for enforcing policy decisions.

- In the PolicyEnforcementFilter, enable `cache`:

```
"cache": {
  "enabled" : true
}
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 7
Hardening Authorization With Advice From AM

To protect sensitive resources, AM policies can be configured with additional conditions to harden the authorization. When AM communicates these policy decisions to IG, the decision includes advices to indicate what extra conditions the user must meet.

Conditions can include requirements to access the resource over a secure channel, access during working hours, or to reauthenticate at a higher authentication level. For more information, see AM's *Authorization Guide*.

The chapter gives examples of how to step up the authentication level for a session, and how to use transactional authorization to increase the authorization required for a single transaction. The examples build on the policies in "*Enforcing Policy Decisions From AM*".

7.1. Stepping Up the Authentication Level for a Session

When you step up the authentication level for a session, the authorization is verified and then captured as part of the session, and the user-agent is authorized to that authentication level for the duration of the session.

This section uses the policies you created in "Enforcing AM Policy Decisions In the Same Domain" and "Enforcing AM Policy Decisions Cross-Domain", adding an authorization policy with a Authentication by Service environment condition. Except for the paths where noted, procedures for single domain and cross-domain are the same.

After the user-agent redirects the user to AM, if the user is not already authenticated they are presented with a login page. If the user is already authenticated, or after they authenticate, they are presented with a second page asking for a verification code to meet the `AuthenticateToService` environment condition.

To Set Up the Authentication Level for a Session

Before you start, configure IG and AM:

- For SSO, as described in "Enforcing AM Policy Decisions In the Same Domain".
- For CDSSO, as described in "Enforcing AM Policy Decisions Cross-Domain".

1. In the AM console, add an environment condition to the policy:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

a. Select a policy set:

- For SSO, select Authorization > Policy Sets > PEP-SSO.
- For CDSSO, select Authorization > Policy Sets > PEP-CDSSO.

b. In the IG policy, select Environments, and add the following environment condition:

- `All of`
- Type: `Authentication by Service`
- Authenticate to Service: `VerificationCodeLevel1`

2. Set up client-side and server-side scripts:

a. Select Scripts > Scripted Module - Client Side, and replace the default script with one of the following scripts:

- For AM 6 and later versions, use this script:

```
/*
 * Copyright 2018 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */
autoSubmitDelay = 60000;

function callback() {
    var parent = document.createElement("div");
    parent.className = "form-group";

    var label = document.createElement("label");
    label.className = "sr-only separator";
    label.setAttribute("for", "answer");
    label.innerText = "Verification Code";
    parent.appendChild(label);

    var input = document.createElement("input");
    input.className = "form-control input-lg";
    input.type = "text";
    input.placeholder = "Enter your verification code";
    input.name = "answer";
    input.id = "answer";
    input.value = "";
    input.oninput = function(event) {
        var element = document.getElementById("clientScriptOutputData");
        if (!element.value) element.value = "{}";
        var json = JSON.parse(element.value);
        json["answer"] = event.target.value;
        element.value = JSON.stringify(json);
    };
    parent.appendChild(input);
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    var fieldset = document.forms[0].getElementsByTagName("fieldset")[0];
    fieldset.prepend(parent);
}

if (document.readyState !== 'loading') {
    callback();
} else {
    document.addEventListener("DOMContentLoaded", callback);
}
```

- For earlier versions of AM, use this script:

```
spinner.hideSpinner();
autoSubmitDelay = 60000;
$(document).ready(function() {
  fs = $(document.forms[0]).find("fieldset");
  strUI = '<div class="form-group"> \
           <label class="sr-only separator" for="answer"> \
           Verification Code</label><input onchange="s=$(\'#clientScriptOutputData\')[0]; \
           if (!s.value) s.value=\'{}\'; d=JSON.parse(s.value); d[\'answer\']=value; \
           s.value=JSON.stringify(d);" id="answer" class="form-control input-lg" type="text" \
           placeholder="Enter your verification code" value="" name="answer"></input></div>';
  $(fs).prepend(strUI);
});
```

Leave all other values as default.

This client-side script adds a field to the AM form, in which the user is required to enter a verification code. The script formats the entered code as a JSON object, as required by the server-side script.

b. Select Scripts > Scripted Module - Server Side, and replace the default script with the following script:

```
username = 'demo'
logger.error('username: ' + username)

// Test whether the user 'demo' enters the correct validation code
data = JSON.parse(clientScriptOutputData);
answer = data.answer;

if (answer !== '123456') {
  logger.error('Authentication Failed !!')
  authState = FAILED;
} else {
  logger.error('Authenticated !!')
  authState = SUCCESS;
}
```

Leave all other values as default.

This server-side script tests that the user has entered `123456` as the verification code.

3. Add an authentication module:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

a. In the top level realm, select Authentication > Modules, and add a module with the following settings:

- Name: `VerificationCodeLevel1`
- Type: `Scripted Module`

b. In the authentication module, enable the option for client-side script, and select the following options:

- Client-side Script: `Scripted Module - Client Side`
- Server-side Script: `Scripted Module - Server Side`
- Authentication Level: `1`

4. Add the authentication module to an authentication chain:

a. Select Authentication > Chains, and add a chain called `VerificationCodeLevel1`.

b. Add a module with the following settings:

- Select Module: `VerificationCodeLevel1`
- Select Criteria: `Required`

To Test the Setup

1. Log out of AM.

2. Access the route:

- For SSO, browse to http://openig.example.com:8080/home/pep-sso.
- For CDSSO, browse to http://openig.ext.com:8080/home/pep-cdsso.

If you have not previously authenticated to AM, the SingleSignOnFilter redirects the request to AM for authentication.

3. Log in to AM as user `demo`, password `Ch4ng31t`.

AM creates a session with the default authentication level `0`, and IG requests a policy decision.

The updated policy requires authentication level `1`, which is higher than the session's current authentication level. AM issues a redirect with a `AuthenticateToServiceConditionAdvice` to authenticate at level `1`.

4. In the session upgrade window, enter the verification code `123456`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

AM upgrades the authentication level for the session to 1, and grants access to the sample application. If you try to access the sample application again in the same session, you don't need to provide the verification code.

7.2. Increasing Authorization for a Single Transaction

Transactional authorization improves security by requiring a user to perform additional actions when trying to access a resource protected by an AM policy. For example, they must reauthenticate to an authentication module or respond to a push notification on their mobile device.

Performing the additional action successfully grants access to the protected resource, but only once. Additional attempts to access the resource require the user to perform the configured actions again.

This section builds on the example in "Stepping Up the Authentication Level for a Session", adding a simple authorization policy with a `Transaction` environment condition. Each time the user-agent tries to access the protected resource, the user must reauthenticate to an authentication module by providing a verification code.

This feature is supported with AM 5.5 and later versions.

To Use Transactional Authorization

Before you start, configure AM as described in "To Set Up the Authentication Level for a Session". The IG configuration is not changed.

1. In the AM console, add a new Environment condition:
    a. Select the policy set:
        - For SSO, select Authorization > Policy Sets > PEP-SSO.
        - For CDSSO, select Authorization > Policy Sets > PEP-CDSSO.
    b. In the IG policy, select Environments and add another environment condition:
        - `All of`
        - Type: `Transaction`
        - Authentication strategy: `Authenticate To Module`
        - Strategy specifier: `TxVerificationCodeLevel5`
2. Set up client-side and server-side scripts:
    a. Select Scripts > New Script, and add the following client-side script:
        - Name: `Tx Scripted Module - Client Side`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Script Type: `Client-side Authentication`

```
/*
 * Copyright 2018 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */
autoSubmitDelay = 60000;

function callback() {
    var parent = document.createElement("div");
    parent.className = "form-group";

    var label = document.createElement("label");
    label.className = "sr-only separator";
    label.setAttribute("for", "answer");
    label.innerText = "Verification Code";
    parent.appendChild(label);

    var input = document.createElement("input");
    input.className = "form-control input-lg";
    input.type = "text";
    input.placeholder = "Enter your TX code";
    input.name = "answer";
    input.id = "answer";
    input.value = "";
    input.oninput = function(event) {
        var element = document.getElementById("clientScriptOutputData");
        if (!element.value) element.value = "{}";
        var json = JSON.parse(element.value);
        json["answer"] = event.target.value;
        element.value = JSON.stringify(json);
    };
    parent.appendChild(input);

    var fieldset = document.forms[0].getElementsByTagName("fieldset")[0];
    fieldset.prepend(parent);
}

if (document.readyState !== 'loading') {
    callback();
} else {
    document.addEventListener("DOMContentLoaded", callback);
}
```

This client-side script adds a field to the AM form, in which the user is required to enter a TX code. The script formats the entered code as a JSON object, as required by the server-side script.

b. Select Scripts > New Script, and add the following server side script:

- Name: `Tx Scripted Module - Server Side`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Script Type: `Server-side Authentication`

```
username = 'demo'
logger.error('username: ' + username)

// Test whether the user 'demo' enters the correct validation code
data = JSON.parse(clientScriptOutputData);
answer = data.answer;

if (answer !== '789') {
  logger.error('Authentication Failed !!')
  authState = FAILED;
} else {
  logger.error('Authenticated !!')
  authState = SUCCESS;
}
```

This server-side script tests that the user has entered `789` as the verification code.

3. Add an authentication module:

   a. Select Authentication > Modules, and add a module with the following settings:

      - Name: `TxVerificationCodeLevel5`
      - Type: `Scripted Module`

   b. In the authentication module, enable the option for client-side script, and select the following options:

      - Client-side Script: `Tx Scripted Module - Client Side`
      - Server-side Script: `Tx Scripted Module - Server Side`
      - Authentication Level: `5`

To Test the Setup

1. Log out of AM.
2. Browse to your route:

   - For SSO, browse to http://openig.example.com:8080/home/pep-sso.
   - For CDSSO, browse to http://openig.ext.com:8080/home/pep-cdsso.

   If you have not previously authenticated to AM, the SingleSignOnFilter redirects the request to AM for authentication.

3. Log in to AM as user `demo`, password `Ch4ng31t`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

AM creates a session with the default authentication level `0`, and IG requests a policy decision.

4. Enter the verification code `123456` to upgrade the authorization level for the session to `1`.

   The authentication module you configured for transactional authorization requires authentication level `5`, so AM issues a `TransactionConditionAdvice`.

5. In the transaction upgrade window, enter the verification code `789`.

   AM upgrades the authentication level for this policy evaluation to `5`, and then returns a policy decision that grants a one-time access to the sample application. If you try to access the sample application again, you must enter the code again.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 8
Acting As a SAML 2.0 Service Provider

The following sections describe IG's role as a SAML 2.0 service provider, and give an example of how to set up IG, with AM as an identity provider:

For information about how to set up multiple service providers, see "*SAML 2.0 and Multiple Applications*".

8.1. About SAML 2.0 SSO and Federation

The IG federation component implements SAML 2.0, to validate users and log them in to protected applications.

The SAML 2.0 standard describes the messages that providers exchange, and the way that they exchange them. SAML 2.0 enables web single sign-on (SSO), for example, where the service managing the user's identity does not belong to the same organization and does not use the same software as the service that the user wants to access.

The following terms are used in SAML and federation:

- *Identity Provider* (IDP): The service that manages the user identity.
- *Service Provider* (SP): The service that a user wants to access. IG acts as a SAML 2.0 SP for SSO, providing users with an interface to applications that don't support SAML 2.0.
- *Circle of trust* (CoT): An IDP and SP that participate in the federation.

When an IDP and an SP participate in federation, they agree on what security information to exchange, and mutually configure access to each other's services.

After an IDP authenticates a user, it provides the SP with SAML assertions that attest to which user is authenticated, when the authentication succeeded, how long the assertion is valid, and so on. The SP uses the SAML assertions to make authorization decisions, for example, to let an authenticated user complete a purchase that gets charged to the user's account at the IDP.

The IDP and SP usually communicate about a user identified by a name identifier. In SP-initiated SSO and IDP-initiated SSO, the NameID format can be any format supported by the IDP. For more information, see "Using a Non-Transient NameID Format".

SAML assertions can be signed and encrypted. For a list of allowed algorithms for signing XML documents, see Algorithms in the AM *SAML v2.0 Guide*. ForgeRock recommends using *SHA-256 variants (rsa-sha256 or ecdsa-sha256).

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

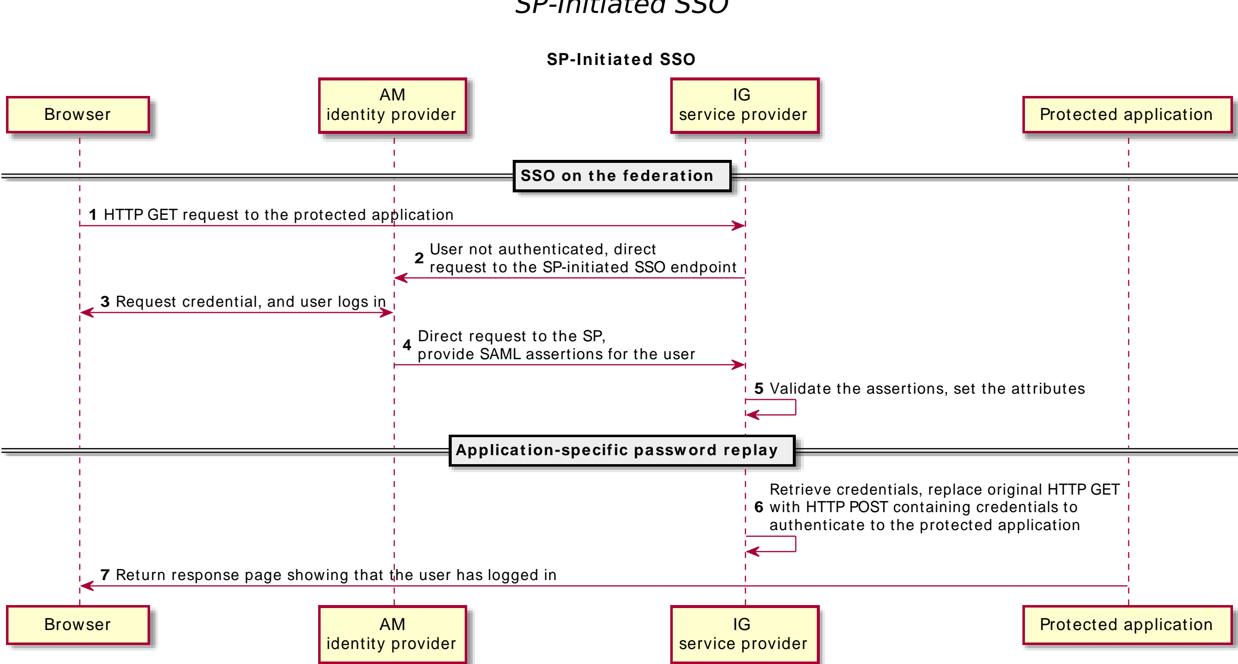
SAML assertions can contain configurable attribute values, such as user meta-information or anything else provided by the IDP. The attributes of a SAML assertion can contain one or more values, made available as a list of strings. Even if an attribute contains a single value, it is made available as a list of strings.

8.2. About SP-Initiated SSO

SP-initiated SSO occurs when a user attempts to access a protected application directly through the SP. Because the user's federated identity is managed by the IDP, the SP sends a SAML authentication request to the IDP. After the IDP authenticates the user, it provides the SP with a SAML assertion for the user.

The following sequence diagram shows the flow of information in SP-initiated SSO, when IG acts as a SAML 2.0 SP:

SP-Initiated SSO



8.3. About IDP-Initiated SSO

IDP-initiated SSO occurs when a user attempts to access a protected application, using the IDP for authentication. The IDP sends an unsolicited authentication statement to the SP.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Before IDP-initiated SSO can occur:

- The user must access a link on the IDP that refers to the remote SP.
- The user must authenticate to the IDP.
- The IDP must be configured with links that refer to the SP.

The following sequence diagram shows the flow of information in IDP-initiated SSO when IG acts as a SAML 2.0 SP:

IDP-Initiated SSO

IDP-Initiated SSO

Browser
AM identity provider
IG service provider
Protected application
SSO on the federation
1 HTTP GET request to the protected application through IDP-initiated SSO endpoint
2 Request credentials, and user logs in
3 Direct the request to the SP, provide SAML assertions for the user
4 Validate the assertions, set the attributes
Application-specific password replay
5 Retrieve credentials, replace original HTTP GET with HTTP POST containing credentials to authenticate to the protected application.
6 Return response page showing that the user has logged in
Browser
AM identity provider
IG service provider
Protected application

8.4. Setting Up IDP-initiated SSO and SP-initiated SSO

The following sections describe how to set up AM as an IDP, and IG as an SP to protect an application:

For examples of the federation configuration files, see "Federation Configuration Files". You can copy and edit these files to create new configurations.

To set up multiple SPs, work through this section, and then consider the explanation in "*SAML 2.0 and Multiple Applications*".

This tutorial does not address PKI configuration for validation and encryption, although IG is capable of handling both, just as any AM Fedlet can handle both.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Prepare the Network

Configure the network so that browser traffic to the application hosts is proxied through IG. The example in this chapter uses the host name `sp.example.com`.

- Add `sp.example.com` to your `/etc/hosts` file:

```
127.0.0.1    localhost openam.example.com openig.example.com app.example.com sp.example.com
```

To Configure AM as An IDP for SAML 2.0

Before you start this tutorial:

- Prepare IG as described in "*First Steps*" in the *Getting Started Guide*.
- Install and configure AM on http://openam.example.com:8088/openam, with the default configuration. If you use a different configuration, substitute in the tutorial accordingly.

1. In AM, Select Identities, and add a user with the following values:
   - ID/username: `george`
   - First name: `george`
   - Last name: `C0stanza`

     Note that, for this example, the last name must be the same as the password.
   - Password: `C0stanza`
2. Set up a hosted IDP:
   a. In the top level realm select Configure SAMLv2 Provider > Create Hosted Identity Provider.
   b. Create an IDP with the following settings, and then select Configure:
      - Name: `openam`
      - Signing Key: `test`
      - Circle of Trust: `Circle of Trust`
      - Attribute Mapping: `cn` to `cn`, and `sn` to `sn`

        The mapping indicates that IG (the SP) wants AM (the IDP) to get the value of the `cn` and `sn` attributes from the user profile and send them to IG. IG can use the attribute values to log the user in to the application it protects. In a real deployment, you would use other attributes.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

A confirmation page is displayed. You can start to create a Fedlet from this page or go back to the top level realm, as described in the following step.

3. Create a fedlet configuration to act as a lightweight SAML v2.0 SP.

   a. In the top level realm, select Create Fedlet Configuration.

   b. Select the following options, and then select Create:

      - Name: `sp`
      - Destination URL: `http://sp.example.com:8080/saml`
      - Attribute Mapping: Map `cn` to `cn`, and `sn` to `sn`.

   The federation configuration files are created in a directory similar to this: `$HOME/openam/myfedlets/`*`openig-fedlet`*`/Fedlet.zip`. In versions of AM before 13.5, the configuration files are provided in a `war` directory or .zip file.

   By default, AM as an IDP uses the NameID format `urn:oasis:names:tc:SAML:2.0:nameid-format:transient` to communicate about a user. For information about using a different NameID format, see "Using a Non-Transient NameID Format".

To Configure IG as An SP

Before you start, set up AM as described in "To Configure AM as An IDP for SAML 2.0".

1. Retrieve the Fedlet configuration files:

   a. Unpack the fedlet files created in the previous procedure into the IG configuration. For example:

```
$ cd $HOME/openam/myfedlets/sp
$ unzip Fedlet.zip
$ mkdir $HOME/.openig/SAML
$ cp conf/* $HOME/.openig/SAML
$ ls -l $HOME/.openig/SAML

FederationConfig.properties
fedlet.cot
idp-extended.xml
idp.xml
sp-extended.xml
sp.xml
```

   b. If the following header is defined in `sp-extended.xml`, comment it out to prevent issues with timeout:

```
<!--
  <Attribute name="fedletAdapter">
    <Value>com.sun.identity.saml2.plugins.DefaultFedletAdapter</Value>
  </Attribute>
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
-->
```

c. The SAML library component validates the SP's AssertionConsumerService Location against the incoming IDP SAML Assertion, based on the request information, including the port. In `sp.xml`, always specify the port in the `Location` value of `AssertionConsumerService`, even when using defaults of `443` or `80`, as follows:

```
<AssertionConsumerService isDefault="true" index="0"
 Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Location="https://sp.example.com:443/
fedletapplication"/>
```

d. Restart IG.

2. Add the following route for credential injection as `$HOME/.openig/config/routes/05-saml.json` (on Windows, `%appdata%\OpenIG\config\routes\05-saml.json`):

```
{
  "handler": {
    "type": "SamlFederationHandler",
    "config": {
      "assertionMapping": {
        "username": "cn",
        "password": "sn"
      },
      "subjectMapping": "sp-subject-name",
      "redirectURI": "/federate"
    }
  },
  "condition": "${matches(request.uri.path, '^/saml')}",
  "session": "JwtSession"
}
```

Notice the following features of the route:

- The route matches requests to `/saml`.
- After authentication, the SamlFederationHandler extracts `cn` and `sn` from the SAML assertion, and maps them to the session fields `session.username` and `session.password`.
- The handler stores the subject name as a string in the session field `session.sp-subject-name`, which is named by the `subjectMapping` property. By default, the subject name is stored in the session field `session.subjectName`.
- The handler redirects the request to the `/federate` route.
- The route uses the `JwtSession` implementation, meaning it stores encrypted session information in a browser cookie. The name is a reference to the `JwtSession` object defined in `config.json`. For details, see JwtSession(5) in the *Configuration Reference*.

3. Add the following route for SAML federation as `$HOME/.openig/config/routes/05-federate.json` (on Windows, `%appdata%\OpenIG\config\routes\05-federate.json`):

```
{
  "handler": {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    "type": "DispatchHandler",
    "config": {
      "bindings": [
        {
          "condition": "${empty session.username}",
          "handler": {
            "type": "StaticResponseHandler",
            "config": {
              "status": 302,
              "reason": "Found",
              "headers": {
                "Location": [
                  "http://sp.example.com:8080/saml/SPInitiatedSSO"
                ]
              }
            }
          }
        },
        {
          "handler": {
            "type": "Chain",
            "config": {
              "filters": [
                {
                  "type": "StaticRequestFilter",
                  "config": {
                    "method": "POST",
                    "uri": "http://app.example.com:8081/login",
                    "form": {
                      "username": [
                        "${session.username[0]}"
                      ],
                      "password": [
                        "${session.password[0]}"
                      ]
                    }
                  }
                }
              ],
              "handler": "ReverseProxyHandler"
            }
          }
        }
      ]
    }
  },
  "condition": "${matches(request.uri.path, '^/federate')}",
  "session": "JwtSession"
}
```

Notice the following features of the route:

- The route matches requests to `/federate`.
- If the user is not authenticated with AM, the username is not populated in the context. The DispatchHandler then dispatches the request to the StaticResponseHandler, which redirects it to the SP-initiated SSO endpoint.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

If the credentials are in the context, or after successful authentication, the DispatchHandler dispatches the request to the Chain.

- The StaticRequestFilter retrieves the first value for the `username` and `password` attributes of the SAML assertion. It replaces the original HTTP GET request with an HTTP POST login request that contains the credentials to authenticate.
- The route uses the `JwtSession` implementation, meaning it stores encrypted session information in a browser cookie. The name is a reference to the `JwtSession` object defined in `config.json`. For details, see JwtSession(5) in the *Configuration Reference*.

Tip

For more control over the URL where the user agent is redirected, use the `RelayState` query string parameter in the URL of the redirect `Location` header. `RelayState` specifies where to redirect the user when the SAML 2.0 web browser SSO process is complete. It overrides the `redirectURI` set in the SamlFederationHandler.

The `RelayState` value must be URL-encoded. When using an expression, use a function to encode the value. For example, use `${urlEncodeQueryParameterNameOrValue(contexts.router.originalUri)}`.

In the following example, the user is finally redirected to the original URI from the request:

```
"headers": {
  "Location": [
    "http://openig.example.com:8080/saml/SPInitiatedSSO?RelayState=
${urlEncodeQueryParameterNameOrValue(contexts.router.originalUri)}"
  ]
}
```

To Test the Setup

1. Test IDP-initiated SSO:
   a. Log out of AM, and select this link to AM for IDP-initiated SSO.

      Because the user is not authenticated with AM, AM displays its login page.
   b. Log in to AM as user `george`, password `C0stanza`.

      After authentication, the request is processed as follows:
      - AM directs the request to the `/saml` route on IG, providing SAML assertions for the user.
      - The SamlFederationHandler in the `/saml` route validates the assertion, sets the attributes, and redirects the request to the route.
      - The DispatchHandler in the `/federate` route directs the request to the StaticRequestFilter.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- The StaticRequestFilter retrieves the credentials and replaces the original HTTP GET request with an HTTP POST login request that contains the credentials to authenticate to the protected application.
- IG returns the response page showing that the user has logged in.

2. Test SP-initiated SSO:

   a. Log out of AM, and browse to http://openig.example.com:8080/federate.

      Because the user is not authenticated, the session username is not populated in the context. The DispatchHandler in the `/federate` route directs the request to AM for SP-initiated SSO.

   b. Log in to AM as user `george`, password `C0stanza`.

      After authentication, the request is processed as follows:

      - AM directs the request to the `/saml` route on IG, providing SAML assertions for the user.
      - The SamlFederationHandler in the `/saml` route validates the assertion, sets the attributes, and redirects the request to the route.
      - The DispatchHandler in the `/federate` route directs the request to the StaticRequestFilter.
      - The StaticRequestFilter retrieves the credentials and replaces the original HTTP GET request with an HTTP POST login request that contains the credentials to authenticate to the protected application.
      - IG returns the response page showing that the user has logged in.

8.5. Using a Non-Transient NameID Format

By default, AM as an IDP uses the NameID format `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`. For information about configuring AM to use a different NameID format, see SAML v2.0 Configuration Properties in AM's *SAML v2.0 Guide*.

When the IDP uses another NameID format, configure IG to use that NameID format by editing the Fedlet configuration file `sp-extended.xml` retrieved in "To Configure IG as An SP":

- To use the NameID value provided by the IDP, add the following attribute:

```
<Attribute name="useNameIDAsSPUserID">
  <Value>true</Value>
</Attribute>
```

- To use an attribute from the assertion, add the following attribute:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
<Attribute name="autofedEnabled">
  <Value>true</Value>
</Attribute>
<Attribute name="autofedAttribute">
  <Value>sn</Value>
</Attribute>
```

This example uses the value in `SN` to identify the subject.

Although IG supports the `persistent` NameID format, IG does not store the mapping. To configure this behavior, edit the file `sp-extended.xml`:

- To disable attempts to persist the user mapping, add the following attribute:

```
<Attribute name="spDoNotWriteFederationInfo">
  <Value>true</Value>
</Attribute>
```

- To enable attempts to persist the user mapping, add the following attribute:

```
<Attribute name="spDoNotWriteFederationInfo">
  <Value>false</Value>
</Attribute>
```

If a login request doesn't contain a NameID format query parameter, the value is defined by the presence and content of the NameID format list for the SP and IDP. For example, an SP-initiated login can be constructed with the binding and `NameIDFormat` as a parameter, as follows:

```
http://fedlet.example.org:7070/fedlet/SPInitiatedSSO?binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST&NameIDFormat=urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified
```

When the NameID format is provided in a list, it is resolved as follows:

- If both the IDP and SP have a list, the first matching NameID format in the lists.
- If either the IDP or SP list is empty, the first NameID format in the other list.
- If neither the IDP nor SP has a list, then AM defaults to `transient`, and IG defaults to `persistent`.

8.6. Federation Configuration Files

The following section summarizes and give examples of the files that are used in SAML 2.0 SSO and federation:

8.6.1. Summary of Federation Configuration Files

The following table summarizes the files that are used in SAML 2.0 SSO and federation:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Fedlet Configuration Files

File	Description
`fedlet.cot`	Circle of trust for IG and the IDP.
`idp.xml`	Standard metadata. This file is usually generated by the IDP.
`idp-extended.xml`	Metadata extensions. When using AM as the IDP, this file is generated by AM. When using a third-party IDP, create this file from a template or example.
`sp.xml`	Standard metadata for the IG SP. When using AM as the IDP, and the Fedlet wizard generates base standard/extended XML, this file is generated by AM. When using a third-party IDP, create this file from a template or example.
`sp-extended.xml`	Metadata extensions for the IG SP. This file is usually generated by the IDP.

8.6.2. Example Circle of Trust File

The following example of `$HOME/.openig/SAML/fedlet.cot` defines a CoT between AM as the IDP and an IG SP:

```
cot-name=Circle of Trust
sun-fm-cot-status=Active
sun-fm-trusted-providers=openam,sp
sun-fm-saml2-readerservice-url=
sun-fm-saml2-writerservice-url=
```

8.6.3. Example SAML Configuration File

The following example of `$HOME/.openig/SAML/sp.xml` defines a SAML configuration file for an IG SP, `sp`:

```
<EntityDescriptor
  entityID="sp"
  xmlns="urn:oasis:names:tc:SAML:2.0:metadata">
  <SPSSODescriptor
    AuthnRequestsSigned="false"
    WantAssertionsSigned="false"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <SingleLogoutService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
      Location="http://sp.example.com:8080/saml/fedletSloRedirect"
      ResponseLocation="http://sp.example.com:8080/saml/fedletSloRedirect"/>
    <SingleLogoutService
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
      Location="http://sp.example.com:8080/saml/fedletSloPOST"
      ResponseLocation="http://sp.example.com:8080/saml/fedletSloPOST"/>
    <SingleLogoutService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
      Location="http://sp.example.com:8080/saml/fedletSloSoap"/>
    <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:transient</NameIDFormat>
    <AssertionConsumerService
      isDefault="true"
      index="0"
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
      Location="http://sp.example.com:8080/saml/fedletapplication"/>
    <AssertionConsumerService
      index="1"
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"
      Location="http://sp.example.com:8080/saml/fedletapplication"/>
  </SPSSODescriptor>
  <RoleDescriptor
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:query="urn:oasis:names:tc:SAML:metadata:ext:query"
    xsi:type="query:AttributeQueryDescriptorType"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
  </RoleDescriptor>
  <XACMLAuthzDecisionQueryDescriptor
    WantAssertionsSigned="false"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
  </XACMLAuthzDecisionQueryDescriptor>
</EntityDescriptor>
```

8.6.4. Example Extended Configuration File

The following example of `$HOME/.openig/SAML/sp-extended.xml` defines a SAML configuration file for an IG SP, `sp`:

```
<EntityConfig xmlns="urn:sun:fm:SAML:2.0:entityconfig"
    xmlns:fm="urn:sun:fm:SAML:2.0:entityconfig"
    hosted="1"
    entityID="sp">

    <SPSSOConfig metaAlias="/sp">
        <Attribute name="description">
            <Value></Value>
        </Attribute>
        <Attribute name="signingCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="encryptionCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="basicAuthOn">
            <Value>false</Value>
        </Attribute>
        <Attribute name="basicAuthUser">
            <Value></Value>
        </Attribute>
        <Attribute name="basicAuthPassword">
            <Value></Value>
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
        </Attribute>
        <Attribute name="autofedEnabled">
            <Value>false</Value>
        </Attribute>
        <Attribute name="autofedAttribute">
            <Value></Value>
        </Attribute>
        <Attribute name="transientUser">
            <Value>anonymous</Value>
        </Attribute>
        <Attribute name="spAdapter">
            <Value></Value>
        </Attribute>
        <Attribute name="spAdapterEnv">
            <Value></Value>
        </Attribute>
        <!--
        <Attribute name="fedletAdapter">
            <Value>com.sun.identity.saml2.plugins.DefaultFedletAdapter</Value>
        </Attribute>
        -->
        <Attribute name="fedletAdapterEnv">
            <Value></Value>
        </Attribute>
        <Attribute name="spAccountMapper">
            <Value>com.sun.identity.saml2.plugins.DefaultLibrarySPAccountMapper</Value>
        </Attribute>
        <Attribute name="useNameIDAsSPUserID">
            <Value>false</Value>
        </Attribute>
        <Attribute name="spAttributeMapper">
            <Value>com.sun.identity.saml2.plugins.DefaultSPAttributeMapper</Value>
        </Attribute>
        <Attribute name="spAuthncontextMapper">
            <Value>com.sun.identity.saml2.plugins.DefaultSPAuthnContextMapper</Value>
        </Attribute>
        <Attribute name="spAuthncontextClassrefMapping">
            <Value>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport|0|default</Value>
        </Attribute>
        <Attribute name="spAuthncontextComparisonType">
           <Value>exact</Value>
        </Attribute>
        <Attribute name="attributeMap">
           <Value>cn=cn</Value>
           <Value>sn=sn</Value>
        </Attribute>
        <Attribute name="saml2AuthModuleName">
           <Value></Value>
       </Attribute>
       <Attribute name="localAuthURL">
           <Value></Value>
       </Attribute>
       <Attribute name="intermediateUrl">
           <Value></Value>
       </Attribute>
       <Attribute name="defaultRelayState">
           <Value></Value>
       </Attribute>
       <Attribute name="appLogoutUrl">
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
            <Value>http://sp1.example.com:8080/saml/logout</Value>
        </Attribute>
        <Attribute name="assertionTimeSkew">
            <Value>300</Value>
        </Attribute>
        <Attribute name="wantAttributeEncrypted">
            <Value></Value>
        </Attribute>
        <Attribute name="wantAssertionEncrypted">
            <Value></Value>
        </Attribute>
        <Attribute name="wantNameIDEncrypted">
            <Value></Value>
        </Attribute>
        <Attribute name="wantPOSTResponseSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="wantArtifactResponseSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="wantLogoutRequestSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="wantLogoutResponseSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="wantMNIRequestSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="wantMNIResponseSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="responseArtifactMessageEncoding">
            <Value>URI</Value>
        </Attribute>
        <Attribute name="cotlist">
        <Value>Circle of Trust</Value></Attribute>
        <Attribute name="saeAppSecretList">
        </Attribute>
        <Attribute name="saeSPUrl">
            <Value></Value>
        </Attribute>
        <Attribute name="saeSPLogoutUrl">
        </Attribute>
        <Attribute name="ECPRequestIDPListFinderImpl">
            <Value>com.sun.identity.saml2.plugins.ECPIDPFinder</Value>
        </Attribute>
        <Attribute name="ECPRequestIDPList">
            <Value></Value>
        </Attribute>
        <Attribute name="ECPRequestIDPListGetComplete">
            <Value></Value>
        </Attribute>
        <Attribute name="enableIDPProxy">
            <Value>false</Value>
        </Attribute>
        <Attribute name="idpProxyList">
            <Value></Value>
        </Attribute>
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
        <Attribute name="idpProxyCount">
            <Value>0</Value>
        </Attribute>
        <Attribute name="useIntroductionForIDPProxy">
            <Value>false</Value>
        </Attribute>
        <Attribute name="spSessionSyncEnabled">
            <Value>false</Value>
        </Attribute>
         <Attribute name="relayStateUrlList">
         </Attribute>
    </SPSSOConfig>
    <AttributeQueryConfig metaAlias="/attrQuery">
        <Attribute name="signingCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="encryptionCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="wantNameIDEncrypted">
            <Value></Value>
        </Attribute>
        <Attribute name="cotlist">
            <Value>Circle of Trust</Value>
        </Attribute>
    </AttributeQueryConfig>
    <XACMLAuthzDecisionQueryConfig metaAlias="/pep">
        <Attribute name="signingCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="encryptionCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="basicAuthOn">
            <Value>false</Value>
        </Attribute>
        <Attribute name="basicAuthUser">
            <Value></Value>
        </Attribute>
        <Attribute name="basicAuthPassword">
            <Value></Value>
        </Attribute>
        <Attribute name="wantXACMLAuthzDecisionResponseSigned">
            <Value>false</Value>
        </Attribute>
        <Attribute name="wantAssertionEncrypted">
            <Value>false</Value>
        </Attribute>
        <Attribute name="cotlist">
            <Value>Circle of Trust</Value>
        </Attribute>
    </XACMLAuthzDecisionQueryConfig>
</EntityConfig>
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 9
Acting as an OAuth 2.0 Resource Server

The following sections describe how IG acts as an OAuth 2.0 Resource Server, to resolve and validate access_tokens, and inject them into the context:

- "Validating Access-Tokens Through the Token Info Endpoint"
- "Validating Access-Tokens Through the Introspection Endpoint"
- "Validating Stateless Access_Tokens With the StatelessAccessTokenResolver"
- "Validating Access_Tokens Obtained Through mTLS"
- "Using the OAuth 2.0 Context to Log In To the Sample Application"

For information about allowing third-party applications to access users' resources without having users' credentials, see OAuth 2.0 Authorization Framework.

For information about the context, see OAuth2Context(5) in the *Configuration Reference*. For examples that use fields in OAuth2Context to throttle access to the sample application, see "Configuring a Mapped Throttling Filter" and " Configuring a Scriptable Throttling Filter ".

9.1. About IG As an OAuth 2.0 Resource Server

OAuth 2.0 includes the following entities:

- *Resource owner*: A user who owns protected resources on a resource server. For example, a resource owner can store photos in a web service.
- *Resource server*: A service that gives authorized client applications access to the resource owner's protected resources. In OAuth 2.0, an authorization server grants authorization to a client application, based on the resource owner's consent. For example, a resource server can be a web service that holds a user's photos.
- *Client*: An application that requests access to the resource owner's protected resources, on behalf of the resource owner. For example, a client can be a photo printing service requesting access to a resource owner's photos stored on a web service, after the resource owner gives the client consent to download the photos.
- *Authorization server*: A service responsible for authenticating resource owners, and obtaining their consent to allow client applications to access their resources. For example, AM can act as the

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

OAuth 2.0 authorization server to authenticate resource owners and obtain their consent. Other services, such as Google and Facebook can provide OAuth 2.0 authorization services.

The following image illustrates the steps for a client application to access a user's protected resources, with AM as the authorization server and IG as the resource server:

Handling OAuth 2.0 Requests as an OAuth 2.0 Resource Server

Resource Owner
Client Application
Authorization Server AM
Resource Server IG
1 Authorization request
2 Authorization grant
3 Authorization grant
4 Access token
5 Access token
6 Grab the token from the request header
7 Ask for token information
8 Validate that the token is active and has sufficient scopes
9 Create new context for the authorization server response, at ${ contexts.oauth2}
10 Protected resources
Resource Owner
Client Application
Authorization Server AM
Resource Server IG

- The application obtains an *authorization grant*, representing the resource owner's consent. For information about the different OAuth 2.0 grant mechanisms supported by AM, see OAuth 2.0 Authorization Grant in the AM *OAuth 2.0 Guide*.
- The application authenticates to the authorization server and requests an *access_token*. The authorization server returns an access_token to the application.

  An OAuth 2.0 access_token is an opaque string issued by the authorization server. When the client interacts with the resource server, the client presents the access_token in the `Authorization` header. For example:

  ```
  Authorization: Bearer 7af41ddd-47a4-40dc-b530-a9aa9f7ceda9
  ```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Access_tokens are the credentials to access protected resources. The advantage of access_tokens over passwords or other credentials is that access_tokens can be granted and revoked without exposing the user's credentials.

The access_token represents the authorization to access protected resources. Because an access_token is a bearer token, anyone who has the access_token can use it to get the resources. Access_tokens must therefore be protected, so that requests involving them go over HTTPS.

In OAuth 2.0, the token scopes are strings that identify the scope of access authorized to the client, but can also be used for other purposes.

- The application supplies the access_token to the resource server, which then resolves and validates the access_token by using an access_token resolver, as described in Access Token Resolvers in the *Configuration Reference*.

  If the access_token is valid, the resource server permits the client access to the requested resource.

9.2. Validating Access-Tokens Through the Token Info Endpoint

This section sets up IG as an OAuth 2.0 resource server, using the AM token info endpoint to resolve and validate access_tokens.

To Set Up AM As an Authorization Server for the Token Info Endpoint

Before you start, install and configure AM on http://openam.example.com:8088/openam, with the default configuration. If you use a different configuration, substitute in the tutorial accordingly.

For more information about configuring AM as an OAuth 2.0 authorization service, see Configuring the OAuth 2.0 Authorization Service in the AM *OAuth 2.0 Guide*.

1. In AM, add a Java agent as described in "To Set Up a Java Agent in AM".
2. Add a user as described in "To Set Up a Sample User In AM".
3. Configure an OAuth 2.0 Authorization Server:
   a. In the top level realm, select Configure OAuth Provider > Configure OAuth 2.0.
   b. Accept all of the default values and select Create.
4. Create an OAuth2 Client to request OAuth 2.0 access_tokens:
   a. Select Applications > OAuth 2.0.
   b. Add a client with the following values:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Client ID: `client-application`
- Client secret: `password`
- Scope(s): `mail`, `employeenumber`

c. (From AM 6.5) On the Advanced tab, select the following option:

- Grant Types: `Resource Owner Password Credentials`

To Set Up IG As a Resource Server Using the Token Info Endpoint

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

Before you start, prepare IG and the sample application as described in "*First Steps*" in the *Getting Started Guide*, and access Studio as described in "Accessing Studio" in the *Getting Started Guide*. IG must be running in development mode.

To configure IG without using Studio, add the route in the following procedure as `$HOME/.openig/config/routes/rs-tokeninfo.json` (on Windows, `%appdata%\OpenIG\config\routes\rs-tokeninfo.json`).

1. In IG Studio, create a route:

   a. Browse to http://openig.example.com:8080/openig/studio, and select Protect an Application.

   b. Choose Structured to use the predefined menus and templates.

2. Select Advanced options, and create a route with the following options:

   - Base URI: `http://app.example.com:8081`
   - Condition: Path: `/rs-tokeninfo`
   - Name: `rs-tokeninfo`

3. Configure authorization:

   a. Select Authorization.

   b. Select OAuth 2.0 Resource Server, and enter the following information to reflect the configuration in AM:

      - Token resolver configuration:
        - Access token resolver: `AM token info endpoint`
        - AM service:
          - URI: `http://openam.example.com:8088/openam`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Version: The version of the AM instance, for example, `6.5`.
- Agent: The credentials of the Java agent you created in AM.
  - Username: `ig_agent`
  - Password: `password`

- Scope configuration:
  - Evaluate scopes: `Statically`
  - Scopes: `mail`, `employeenumber`
- OAuth 2.0 Authorization settings:
  - Require HTTPS: Deselect this option
  - Enable cache: Deselect this option

Leave all other values as default.

4. Add a StaticResponseHandler:

   a. On the top-right of the screen, select ⋮ and Editor mode to switch into editor mode.

   Warning

   After switching to Editor mode, you cannot go back. You will be able to use the JSON file editor to manually edit the route, but will no longer be able use the full Studio interface to add or edit filters.

   b. Replace the last ReverseProxyHandler in the route with the following StaticResponseHandler, and then save the route:

```
"handler": {
  "type": "StaticResponseHandler",
  "config": {
   "status": 200,
   "entity": "<html><body><h2>Decoded access_token: ${contexts.oauth2.accessToken.info}</h2></
body></html>"
  }
}
```

5. On the top-right of the screen, select ⋮ and Display to review the route.

   Make sure that the following route is displayed:

```
{
  "name": "rs-tokeninfo",
  "baseURI": "http://app.example.com:8081",
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
  "condition": "${matches(request.uri.path, '^/rs-tokeninfo')}",
  "heap": [
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "http://openam.example.com:8088/openam",
        "realm": "/",
        "ssoTokenHeader": "iPlanetDirectoryPro",
        "version": "6.5",
        "agent": {
          "username": "ig_agent",
          "password": "password"
        },
        "sessionCache": {
          "enabled": false
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "OAuth2ResourceServerFilter-1",
          "type": "OAuth2ResourceServerFilter",
          "config": {
            "scopes": [
              "mail",
              "employeenumber"
            ],
            "requireHttps": false,
            "realm": "OpenIG",
            "accessTokenResolver": {
              "name": "token-resolver-1",
              "type": "OpenAmAccessTokenResolver",
              "config": {
                "amService": "AmService-1"
              }
            }
          }
        }
      ],
      "handler": {
        "type": "StaticResponseHandler",
        "config": {
          "status": 200,
          "entity": "<html><body><h2>Decoded access_token: ${contexts.oauth2.accessToken.info}</h2></
body></html>"
        }
      }
    }
  }
}
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

> **Note**
>
> If necessary, change the value of `version` for AmService to your version of AM.

Notice the following features of the route:

- The route matches requests to `/rs-tokeninfo`.
- The `OAuth2ResourceServerFilter` expects an OAuth 2.0 access_token in the header of the incoming authorization request, with the scopes `mail` and `employeenumber`.

  The `accessTokenResolver` uses the AM server declared in the heap. The token info endpoint to validate the access token is extrapolated from the URL of the AM server.

  For convenience in this test, `"requireHttps"` is false. In production environments, set it to true.
- After the filter successfully validates the access_token, it creates a new context from the authorization server response. The context is named `oauth2`, and can be reached at `contexts.oauth2` or `contexts['oauth2']`.

  The context contains information about the access_token, which can be reached at `contexts.oauth2.accessToken.info`. Filters and handlers further down the chain can access the token info through the context. For an example that uses the scopes `email` and `employeenumber` to log the user in to the sample application, see "Using the OAuth 2.0 Context to Log In To the Sample Application".

  If there is no access_token in the request, or token validation does not complete successfully, the filter returns an HTTP error status to the user-agent, and IG does not continue processing the request. This is done as specified in the RFC, OAuth 2.0 Bearer Token Usage.
- The StaticResponseHandler returns the content of the access_token from the context `${contexts.oauth2.accessToken.info}`.

6. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

To Test the Setup

The following configuration gets an access_token from AM and use it to access IG, which then uses the OAuth 2.0 resource owner password credentials authorization grant.

1. In a terminal window, use a **curl** command similar to the following to retrieve an access_token:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ mytoken=$(curl -s
 \
--user "client-application:password"
 \
--data "grant_type=password&username=george&password=C0stanza&scope=mail%20employeenumber" \
http://openam.example.com:8088/openam/oauth2/access_token | jq -r ".access_token")
```

2. Validate the access_token returned in the previous step:

```
$ curl -v http://openig.example.com:8080/rs-tokeninfo --header "Authorization: Bearer ${mytoken}"

{
  access_token=gofyKg3ulxhEMfiFQ3H-0CBsoXc,
  employeenumber=123,
  mail=george@example.com,
  grant_type=password,
  auth_level=0,
  scope=[employeenumber, mail],
  realm=/,
  token_type=Bearer,
  expires_in=3548,
  client_id=client-application
}
```

Note that the token info endpoint returns the scopes, `employeenumber` and `mail`.

9.3. Validating Access-Tokens Through the Introspection Endpoint

This section sets up IG as an OAuth 2.0 resource server, using the introspection endpoint.

If you have already followed the steps in "To Set Up AM As an Authorization Server for the Token Info Endpoint", do only the last step of this procedure.

To Set Up AM As an Authorization Server for the Introspection Endpoint

Before you start, install and configure AM on http://openam.example.com:8088/openam, with the default configuration. If you use a different configuration, substitute in the tutorial accordingly.

For more information about configuring AM as an OAuth 2.0 authorization service, see Configuring the OAuth 2.0 Authorization Service in the AM *OAuth 2.0 Guide*.

1. Add a user as described in "To Set Up a Sample User In AM".
2. Configure an OAuth 2.0 Authorization Server:
   a. In the top level realm, select Configure OAuth Provider > Configure OAuth 2.0.
   b. Accept all of the default values and select Create.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

3. Create an OAuth2 Client to request OAuth 2.0 access_tokens:
   a. Select Applications > OAuth 2.0.
   b. Add a client with the following values:
      - Client ID: `client-application`
      - Client secret: `password`
      - Scope(s): `mail`, `employeenumber`
   c. (From AM 6.5) On the Advanced tab, select the following option:
      - Grant Types: `Resource Owner Password Credentials`
4. Create an OAuth2 Client authorized to examine (introspect) tokens:
   a. In the top level realm, select Applications > OAuth 2.0.
   b. Add a client with the following values:
      - Client ID: `resource-server`
      - Client secret `password`
      - Scope(s): `am-introspect-all-tokens`
   c. (From AM 6.5) On the Advanced tab, select the following option:
      - Grant Types: `Resource Owner Password Credentials`

To Set Up IG As a Resource Server Using the Introspection Endpoint

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

Before you start, prepare IG and the sample application as described in "*First Steps*" in the *Getting Started Guide*, and access Studio as described in "Accessing Studio" in the *Getting Started Guide*. IG must be running in development mode.

To configure IG without using Studio, add the route in the following procedure to the IG configuration as `$HOME/.openig/config/routes/rs-introspect.json` (on Windows, `%appdata%\OpenIG\config\routes\rs-introspect.json`).

1. In IG Studio, create a route:
   a. Browse to http://openig.example.com:8080/openig/studio, and select ⊕ Protect an Application.
   b. Choose ☰ Structured to use the predefined menus and templates.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

2. Select Advanced options, and create a route with the following options:
   - Base URI: `http://app.example.com:8081`
   - Condition: Path: `/rs-introspect`
   - Name: `rs-introspect`
3. Configure authorization:
   a. Select Authorization.
   b. Select OAuth 2.0 Resource Server, and enter the following information to reflect the configuration in AM:
      - Token resolver configuration:
        - Access token resolver: `OAuth 2.0 introspection endpoint`
        - Introspection endpoint URI: `http://openam.example.com:8088/openam/oauth2/introspect`
        - Client name and Client secret: `resource-server` and `password`

          This is the name and password of the OAuth 2.0 client with the scope to examine (introspect) tokens, configured in AM.
      - Scope configuration:
        - Evaluate scopes: `Statically`
        - Scopes: `mail`, `employeenumber`
      - OAuth 2.0 Authorization settings:
        - Require HTTPS: Deselect this option
        - Enable cache: Deselect this option

      Leave all other values as default.
4. Add a StaticResponseHandler:
   a. On the top-right of the screen, select ⋮ and Editor mode to switch into editor mode.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Warning

After switching to Editor mode, you cannot go back. You will be able to use the JSON file editor to manually edit the route, but will no longer be able use the full Studio interface to add or edit filters.

b. Replace the last ReverseProxyHandler in the route with the following StaticResponseHandler, and then save the route:

```
"handler": {
  "type": "StaticResponseHandler",
  "config": {
   "status": 200,
   "entity": "<html><body><h2>Decoded access_token: ${contexts.oauth2.accessToken.info}</h2></
body></html>"
  }
}
```

5. On the top-right of the screen, select ⋮ and Display to review the route.

   Make sure that the following route is displayed:

```
{
  "name": "rs-introspect",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/rs-introspect')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "OAuth2ResourceServerFilter-1",
          "type": "OAuth2ResourceServerFilter",
          "config": {
            "scopes": [
              "mail",
              "employeenumber"
            ],
            "requireHttps": false,
            "realm": "OpenIG",
            "accessTokenResolver": {
              "name": "token-resolver-1",
              "type": "TokenIntrospectionAccessTokenResolver",
              "config": {
                "endpoint": "http://openam.example.com:8088/openam/oauth2/introspect",
                "providerHandler": {
                  "type": "Chain",
                  "config": {
                    "filters": [
                      {
                        "type": "HeaderFilter",
                        "config": {
                          "messageType": "request",
                          "add": {
                            "Authorization": [
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
                              "Basic ${encodeBase64('resource-server:password')}"
                            ]
                          }
                        }
                      }
                    ],
                    "handler": "ForgeRockClientHandler"
                  }
                }
              }
            }
          }
        }
      ],
      "handler": {
        "type": "StaticResponseHandler",
        "config": {
          "status": 200,
          "entity": "<html><body><h2>Decoded access_token: ${contexts.oauth2.accessToken.info}</h2></
body></html>"
        }
      }
    }
  }
}
```

Notice the following features of the route compared to `rs-tokeninfo.json`:

- The route matches requests to `/rs-introspect`.
- The `accessTokenResolver` uses the token introspection endpoint to validate the access_token.
- The `providerHandler` is a chain that adds an authorization header to the request. The header contains the username and password of the OAuth 2.0 client with the scope to examine (introspect) access_tokens.

6. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

To Test the Setup

The following configuration gets an access_token from AM and use it to access IG, which then uses the OAuth 2.0 resource owner password credentials authorization grant.

1. In a terminal window, use a **curl** command similar to the following to retrieve an access_token:

```
$ mytoken=$(curl -s
 \
--user "client-application:password"
 \
--data "grant_type=password&username=george&password=C0stanza&scope=mail%20employeenumber" \
http://openam.example.com:8088/openam/oauth2/access_token | jq -r ".access_token")
```

2. Validate the access_token returned in the previous step:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ curl -v http://openig.example.com:8080/rs-introspect --header "Authorization: Bearer ${mytoken}"

{
  ...
  active=true,
  scope=employeenumber mail,
  client_id=client-application,
  user_id=george,
  token_type=access_token,
  exp=1534867911,
  sub=george,
  iss=http://openam.example.com:8088/openam/oauth2, auth_level=0
 ...
}
```

Note that the token introspection endpoint returns different information than the token info endpoint.

9.4. Validating Stateless Access_Tokens With the StatelessAccessTokenResolver

The StatelessAccessTokenResolver confirms that stateless access_tokens provided by AM are well-formed, have a valid issuer, have the expected access_token name, and have a valid signature.

After the StatelessAccessTokenResolver resolves an access_token, the OAuth2ResourceServerFilter checks that the token is within the expiry time, and that it provides the required scopes. For more information, see StatelessAccessTokenResolver(5) in the *Configuration Reference*. This feature is supported with OpenAM 13.5, and AM 5 and later versions.

The following sections provide examples of how to validate signed and encrypted access_tokens:

9.4.1. Validating Signed Access_Tokens With the StatelessAccessTokenResolver

This section provides an example of how to validate a signed access_token with the StatelessAccessTokenResolver.

To Set Up Key Stores

Before you start, find the following information:

- The directory where the IG keystore is located (`ig_keystore_directory`)
- The directory where the AM keystore is located (`am_keystore_directory`)
- The AM keystore password (`am_storepass`)
- The AM key password (`am_keypass`)

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

For information about configuring keystores, see Setting Up Keys and Keystores in the AM *Setup and Maintenance Guide*.

1. Configure verification keys:

   a. Create a `signature-key` in the AM JCEKS keystore:

   ```
   $ keytool -genkey
    \
   -alias signature-key
    \
   -dname "CN=openig.example.com, OU=example, O=com, L=fr, ST=fr, C=fr"
    \
   -keystore "<am_keystore_directory>/keystore.jceks"
    \
   -storetype JCEKS
    \
   -storepass "<am_storepass>"
    \
   -keypass <am_keypass>
    \
   -keyalg RSA -keysize 2048
   ```

   b. Export the `signature-key` to `.pem`:

   ```
   $ keytool -exportcert
    \
   -rfc
    \
   -alias signature-key
    \
   -file "<ig_keystore_directory>/verification-key-cert.pem"
    \
   -keystore "<am_keystore_directory>/keystore.jceks"
    \
   -storetype JCEKS
    \
   -storepass "<am_storepass>"
    \
   -keypass <am_keypass>

   Certificate stored in file .../verification-key-cert.pem
   ```

   c. Import the `.pem` to the IG PKCS12 keystore:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ keytool -import
 \
-trustcacerts
 \
-rfc
 \
-alias verification-key
 \
-file "<ig_keystore_directory>/verification-key-cert.pem"
 \
-keystore "<ig_keystore_directory>/IG_keystore.p12"
 \
-storetype PKCS12
 \
-storepass "keystore"

Trust this certificate? [no]: yes
Certificate was added to keystore
```

2. Restart AM to add the new keys to the AM keystore cache.

To Set Up AM

1. (For AM 6.5.3 and earlier versions) Select Identities > demo, and set the demo user password to `Ch4ng31t`.
2. Configure an OAuth 2.0 Authorization Provider with token encryption disabled:
   a. Select Configure OAuth Provider > Configure OAuth 2.0, accept the default values, and select Create.
   b. Select Services > OAuth2 Provider.
   c. On the Core tab, select the following option:
      - Use Client-Based Access & Refresh Tokens: `on`
   d. On the Advanced tab, select the following options:
      - Supported Scopes: `myscope`
      - OAuth2 Token Signing Algorithm: `RS256`
      - **tokenEncryptionEnabled: Deselect this option**
3. Create an OAuth2 Client to request OAuth 2.0 access_tokens:
   a. Select Applications > OAuth 2.0 > Clients.
   b. Add a client with the following values:
      - Client ID: `client-application`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Client secret: `password`
- Scope(s): `myscope`

c. (From AM 6.5) On the Advanced tab, select the following options:

- Grant Types: `Resource Owner Password Credentials`
- Response Types: `code token`

d. On the Signing and Encryption tab, make sure the following settings are included:

- ID Token Signing Algorithm: `RS256`

4. Add a mapping for the verification keystore:

   a. Select CONFIGURE > SECRET STORES.

   b. Select `default-keystore`, and then select the Mappings tab.

   c. Select `am.services.oauth2.stateless.signing.RSA`, remove the default mapping, and add a mapping for the alias `signature-key`.

> **Important**
>
> In the `default-keystore`, secret IDs are mapped to demo keys provided with AM. Use the `default-keystore` for demo and test purposes only. In production environments, replace the secrets and create mappings for them in an AM secret store.
>
> For information about managing secret stores and mapping secret IDs to aliases, see the AM *Setup and Maintenance Guide*.

To Set Up IG

1. In the IG configuration, set an environment variable for the KeyStore password, and then restart IG:

```
$ export KEYSTORE_SECRET_ID='a2V5c3RvcmU='
```

   The password is retrieved by the default SystemAndEnvSecretStore, and must be base64-encoded.

2. Add the following route to the IG configuration as `$HOME/.openig/config/routes/rs-stateless-signed.json` (on Windows, `%appdata%\OpenIG\config\routes\rs-stateless-signed.json`).

```
{
  "name": "rs-stateless-signed",
  "condition" : "${matches(request.uri.path, '/rs-stateless-signed')}",
  "heap": [
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    {
      "name": "SecretsProvider-1",
      "type": "SecretsProvider",
      "config": {
        "stores": [
          {
            "name": "KeyStoreSecretStore-1",
            "type": "KeyStoreSecretStore",
            "config": {
              "file": "<ig_keystore_directory>/IG_keystore.p12",
              "storeType": "PKCS12",
              "storePassword": "keystore.secret.id",
              "keyEntryPassword": "keystore.secret.id",
              "mappings": [
                {
                  "secretId": "stateless.access.token.verification.key",
                  "aliases": [ "verification-key" ]
                }
              ]
            }
          }
        ]
      }
    }
  ],
  "handler" : {
    "type" : "Chain",
    "capture" : "all",
    "config" : {
      "filters" : [ {
        "name" : "OAuth2ResourceServerFilter-1",
        "type" : "OAuth2ResourceServerFilter",
        "config" : {
          "scopes" : [ "myscope" ],
          "requireHttps" : false,
          "accessTokenResolver": {
            "type": "StatelessAccessTokenResolver",
            "config": {
              "secretsProvider": "SecretsProvider-1",
              "issuer": "http://openam.example.com:8088/openam/oauth2",
              "verificationSecretId": "stateless.access.token.verification.key"
            }
          }
        }
      } ],
      "handler": {
        "type": "StaticResponseHandler",
        "config": {
          "status": 200,
          "entity": "<html><body><h2>Decoded access_token: ${contexts.oauth2.accessToken.info}</h2></
body></html>"
        }
      }
    }
  }
}
```

3. Notice the following features of the route:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- The route matches requests to `/rs-stateless-signed`.
- A SecretsProvider in the heap declares a KeyStoreSecretStore to manage secrets for signed access_tokens.
- The OAuth2ResourceServerFilter expects an OAuth 2.0 access_token in the header of the incoming authorization request, with the scope `myscope`.
- The `accessTokenResolver` uses a `StatelessAccessTokenResolver` to resolve and validate the access_token, which references the SecretsProvider.
- After the OAuth2ResourceServerFilter validates the access_token, it creates the `OAuth2Context` context. For more information, see OAuth2Context(5) in the *Configuration Reference*.
- If there is no access_token in a request, or token validation does not complete successfully, the filter returns an HTTP error status to the user-agent, and IG does not continue processing the request. This is done as specified in the RFC, OAuth 2.0 Bearer Token Usage.
- The StaticResponseHandler returns the content of the access_token from the context.

Test the Setup For a Signed Access_Token

1. Get an access_token for the demo user, using the scope `myscope`:

```
$ mytoken=$(curl -s
 \
--user "client-application:password"
 \
--data "grant_type=password&username=demo&password=Ch4ng31t&scope=myscope" \
http://openam.example.com:8088/openam/oauth2/access_token | jq -r ".access_token")
```

2. Display the token:

```
$ echo ${mytoken}
```

Note that the token is structured as a signed token.

3. Access the route by providing the token returned in the previous step:

```
$ curl -v http://openig.example.com:8080/rs-stateless-signed --header "Authorization: Bearer
 ${mytoken}"
...
  Decoded access_token: {
  sub=demo,
  cts=OAUTH2_STATELESS_GRANT,
  ...
```

9.4.2. Validating Encrypted Access_Tokens With the StatelessAccessTokenResolver

This section provides an example of how to validate an encrypted access_token with the StatelessAccessTokenResolver.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Set Up Key Stores

Before you start, find the following information:

- The directory where the IG keystore is located (`ig_keystore_directory`)
- The directory where the AM keystore is located (`am_keystore_directory`)
- The AM keystore password (`am_storepass`)
- The AM key password (`am_keypass`)

For information about configuring keystores, see Setting Up Keys and Keystores in the AM *Setup and Maintenance Guide*.

1. Configure an encryption key:

   a. Create `encryption-key-1` in the AM JCEKS keystore:

   ```
   $ keytool -genseckey
    \
   -alias encryption-key-1
    \
   -dname "CN=openig.example.com, OU=example, O=com, L=fr, ST=fr, C=fr"
    \
   -keystore "<am_keystore_directory>/keystore.jceks"
    \
   -storetype JCEKS
    \
   -storepass "<am_storepass>"
    \
   -keypass <am_keypass>
    \
   -keyalg AES
    \
   -keysize 256
   ```

   b. Import `encryption-key-1` into the IG PKCS12 keystore, with the alias `decryption-key-1`:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ keytool -importkeystore
 \
-srcalias encryption-key-1
 \
-srckeystore "<am_keystore_directory>/keystore.jceks"
 \
-srcstoretype JCEKS
 \
-srcstorepass "<am_storepass>"
 \
-destkeystore "<ig_keystore_directory>/IG_keystore.p12"
 \
-deststoretype PKCS12
 \
-destalias decryption-key-1
 \
-deststorepass "keystore"
 \
-destkeypass "keystore"

Enter key password for <encryption-key-1> <am_keypass>
```

2. Restart AM to add the new keys to the AM keystore cache.

To Set Up AM

1. (For AM 6.5.3 and earlier versions) Select Identities > demo, and set the demo user password to `Ch4ng31t`.
2. Configure an OAuth 2.0 Authorization Provider with token encryption enabled:
    a. Select Configure OAuth Provider > Configure OAuth 2.0, accept the default values, and select Create.
    b. Select Services > OAuth2 Provider.
    c. On the Core tab, select the following option:
        - Use Client-Based Access & Refresh Tokens: `on`
    d. On the Advanced tab, select the following options:
        - Supported Scopes: `myscope`
        - OAuth2 Token Signing Algorithm: `RS256`
        - **tokenEncryptionEnabled: Select this option**
3. Create an OAuth2 Client to request OAuth 2.0 access_tokens:
    a. Select Applications > OAuth 2.0 > Clients.
    b. Add a client with the following values:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Client ID: `client-application`
- Client secret: `password`
- Scope(s): `myscope`

c. (From AM 6.5) On the Advanced tab, select the following options:

- Grant Types: `Resource Owner Password Credentials`
- Response Types: `code token`

d. On the Signing and Encryption tab, make sure the following settings are included:

- ID Token Signing Algorithm: `RS256`

4. Add a mapping for the encryption keystore:

a. Select CONFIGURE > SECRET STORES.

b. Select `default-keystore`, and then select the Mappings tab.

c. Select `am.services.oauth2.stateless.token.encryption`, remove the default mapping, and add a mapping for the alias `encryption-key-1`.

> **Important**
>
> In the `default-keystore`, secret IDs are mapped to demo keys provided with AM. Use the `default-keystore` for demo and test purposes only. In production environments, replace the secrets and create mappings for them in an AM secret store.
>
> For information about managing secret stores and mapping secret IDs to aliases, see the AM *Setup and Maintenance Guide*.

To Set Up IG

1. In the IG configuration, set an environment variable for the KeyStore password, and then restart IG:

```
$ export KEYSTORE_SECRET_ID='a2V5c3RvcmU='
```

The password is retrieved by the default SystemAndEnvSecretStore, and must be base64-encoded.

2. Add the following route to the IG configuration as `$HOME/.openig/config/routes/rs-stateless-encrypted.json` (on Windows, `%appdata%\OpenIG\config\routes\rs-stateless-encrypted.json`).

```
{
  "name": "rs-stateless-encrypted",
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
  "condition" : "${matches(request.uri.path, '/rs-stateless-encrypted')}",
  "heap": [
    {
      "name": "SecretsProvider-1",
      "type": "SecretsProvider",
      "config": {
        "stores": [
          {
            "name": "KeyStoreSecretStore-1",
            "type": "KeyStoreSecretStore",
            "config": {
              "file": "<ig_keystore_directory>/IG_keystore.p12",
              "storeType": "PKCS12",
              "storePassword": "keystore.secret.id",
              "keyEntryPassword": "keystore.secret.id",
              "mappings": [
                {
                  "secretId": "stateless.access.token.decryption.key",
                  "aliases": [ "decryption-key-1" ]
                }
              ]
            }
          }
        ]
      }
    }
  ],
  "handler" : {
    "type" : "Chain",
    "capture" : "all",
    "config" : {
      "filters" : [ {
        "name" : "OAuth2ResourceServerFilter-1",
        "type" : "OAuth2ResourceServerFilter",
        "config" : {
          "scopes" : [ "myscope" ],
          "requireHttps" : false,
          "accessTokenResolver": {
            "type": "StatelessAccessTokenResolver",
            "config": {
              "secretsProvider": "SecretsProvider-1",
              "issuer": "http://openam.example.com:8088/openam/oauth2",
              "decryptionSecretId": "stateless.access.token.decryption.key"
            }
          }
        }
      } ],
      "handler": {
        "type": "StaticResponseHandler",
        "config": {
          "status": 200,
          "entity": "<html><body><h2>Decoded access_token: ${contexts.oauth2.accessToken.info}</h2></
body></html>"
        }
      }
    }
  }
}
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

3. Notice the following features of the route:

   - The route matches requests to `/rs-stateless-encrypted`.
   - A SecretsProvider in the heap declares a KeyStoreSecretStore to manage secrets for encrypted access_tokens.
   - The OAuth2ResourceServerFilter expects an OAuth 2.0 access_token in the header of the incoming authorization request, with the scope `myscope`.
   - The `accessTokenResolver` uses a `StatelessAccessTokenResolver` to resolve and validate the access_token, which references the SecretsProvider.
   - After the OAuth2ResourceServerFilter validate the access_token, it creates the `OAuth2Context` context. For more information, see OAuth2Context(5) in the *Configuration Reference*.
   - If there is no access_token in a request, or token validation does not complete successfully, the filter returns an HTTP error status to the user-agent, and IG does not continue processing the request. This is done as specified in the RFC, OAuth 2.0 Bearer Token Usage.
   - The StaticResponseHandler returns the content of the access_token from the context.

Test the Setup For an Encrypted Access_Token

1. Get an access_token for the demo user, using the scope `myscope`:

```
$ mytoken=$(curl -s
 \
--user "client-application:password"
 \
--data "grant_type=password&username=demo&password=Ch4ng31t&scope=myscope" \
http://openam.example.com:8088/openam/oauth2/access_token | jq -r ".access_token")
```

2. Display the token:

```
$ echo ${mytoken}
```

   Note that the token is structured as an encrypted token.

3. Access the route by providing the token returned in the previous step:

```
$ curl -v http://openig.example.com:8080/rs-stateless-encrypted --header "Authorization: Bearer
 ${mytoken}"
...
Decoded access_token: {
 sub=demo,
 cts=OAUTH2_STATELESS_GRANT,
 ...
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

9.5. Validating Access_Tokens Obtained Through mTLS

Clients can authenticate to AM through mutual TLS (mTLS) and X.509 certificates. Certificates must be self-signed or use public key infrastructure (PKI), as per version 12 of the draft OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens.

When a client obtains an access_token from AM through mTLS, AM can bind the access_token to the registered client certificate with a confirmation key. The confirmation key is the certificate thumbprint, computed as `base64url-encode(sha256(der(certificate)))`.

When the client connects to IG using the same certificate, IG can use the ConfirmationKeyVerifierAccessTokenResolver to verify that the confirmation key corresponds to the certificate thumbprint.

This proof-of-possession interaction ensures that only the client in possession of the key corresponding to the certificate can use the access_token to access protected resources.

For more information about ConfirmationKeyVerifierAccessTokenResolver, see ConfirmationKeyVerifierAccessTokenResolver(5) in the *Configuration Reference*.

The following sections describe how to set up an example where IG resolves access_tokens acquired from AM through mTLS. The following image illustrates the connections and certificates required by the example.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Client Authentication and Token Resolution Through mTLS

Client registration
Token aquisition
AM
Client
Introspection
Certificate
TLS connection
- Client authentication
- No certificate validation
- Same client certificate presented
Token presentation
IG

9.5.1. Preparing Deployment Containers for HTTPS

This section gives the following pointers for setting up the deployment containers for HTTPS.

Before you start, find the keystore directories to use in the following procedures:

- AM keystore directory: *am_keystore_directory*
- IG keystore directory: *ig_keystore_directory*
- Client keystore directory: *oauth2_client_keystore_directory*

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Prepare the Tomcat Deployment Containers for HTTPS

1. Add the following connector configuration to the container `server.xml`, and then restart AM:

   ```
   <Connector port="8445" protocol="HTTP/1.1" SSLEnabled="true" scheme="https" secure="true">
     <SSLHostConfig sslProtocol="TLS" certificateVerification="optionalNoCA"
       truststoreFile="oauth2_client_keystore_directory/cacerts.p12"
       truststorePassword="changeit"
       truststoreType="PKCS12">
     <Certificate certificateKeystoreFile="am_keystore_directory/keystore.p12"
       certificateKeystorePassword="changeit"
       certificateKeystoreType="PKCS12" />
     </SSLHostConfig>
   </Connector>
   ```

   The `optionalNoCA` setting requires the TLS connector to accept client connections even when a certificate is not present in the specified truststore.

   For more information about setting up AM to run over HTTPS, see Securing Installations in the AM Installation Guide.

2. Add the following connector configuration to the container `server.xml`, and then restart IG:

   ```
   <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true" scheme="https" secure="true">
     <SSLHostConfig sslProtocol="TLS" certificateVerification="optionalNoCA"
       truststoreFile="oauth2_client_keystore_directory/cacerts.p12"
       truststorePassword="changeit"
       truststoreType="PKCS12">
     <Certificate certificateKeystoreFile="ig_keystore_directory/keystore.p12"
       certificateKeystorePassword="changeit"
       certificateKeystoreType="PKCS12" />
     </SSLHostConfig>
   </Connector>
   ```

   The `optionalNoCA` setting requires the TLS connector to accept client connections even when a certificate is not present in the specified truststore.

   For information about setting up IG to run over HTTPS in other container types, see "Configuring IG for HTTPS (Server-Side) in Jetty" and "Configuring JBoss For HTTPS (Server-Side)".

9.5.2. Preparing Key Stores and Trust Stores

This section describes how to prepare the keystores and trust stores required for the example.

To Set Up Key Stores and Trust Stores

1. Create self-signed RSA key pairs for AM, IG, and the client:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ keytool -genkeypair
 \
-alias openam-server
 \
-keyalg RSA
 \
-keysize 2048
 \
-keystore am_keystore_directory/keystore.p12
 \
-storepass changeit
 \
-storetype PKCS12
 \
-keypass changeit
 \
-validity 360
 \
-dname CN=openam.example.com,O=Example,C=FR
```

```
$ keytool -genkeypair
 \
-alias openig-server
 \
-keyalg RSA
 \
-keysize 2048
 \
-keystore ig_keystore_directory/keystore.p12
 \
-storepass changeit
 \
-storetype PKCS12
 \
-keypass changeit
 \
-validity 360
 \
-dname CN=openig.example.com,O=Example,C=FR
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ keytool -genkeypair
 \
-alias oauth2-client
 \
-keyalg RSA
 \
-keysize 2048
 \
-keystore oauth2_client_keystore_directory/keystore.p12
 \
-storepass changeit
 \
-storetype PKCS12
 \
-keypass changeit
 \
-validity 360
 \
-dname CN=test
```

2. Export the certificates to .pem so that the **curl** client can verify the identity of the AM and IG servers:

```
$ keytool -export
 \
-rfc
 \
-alias openam-server
 \
-keystore am_keystore_directory/keystore.p12
 \
-storepass changeit
 \
-storetype PKCS12
 \
-file am_keystore_directory/openam-server.cert.pem

Certificate stored in file .../openam-server.cert.pem
```

```
$ keytool -export
 \
-rfc
 \
-alias openig-server
 \
-keystore ig_keystore_directory/keystore.p12
 \
-storepass changeit
 \
-storetype PKCS12
 \
-file ig_keystore_directory/openig-server.cert.pem

Certificate stored in file openig-server.cert.pem
```

3. Extract the certificate and client private key to .pem so that the **curl** command can identity itself as the client for the HTTPS connection:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ keytool -export
 \
-rfc
 \
-alias oauth2-client
 \
-keystore oauth2_client_keystore_directory/keystore.p12
 \
-storepass changeit
 \
-storetype PKCS12
 \
-file oauth2_client_keystore_directory/oauth2-client.cert.pem

Certificate stored in file .../oauth2-client.cert.pem
```

```
$ openssl pkcs12 \
-in oauth2_client_keystore_directory/keystore.p12
 \
-nocerts
 \
-nodes
 \
-passin pass:changeit
 \
-out oauth2_client_keystore_directory/oauth2-client.key.pem

...verified OK
```

You can now delete the client keystore.

4. Create the CACerts truststore so that AM can validate the client identity:

```
$ keytool -import
 \
-noprompt
 \
-trustcacerts
 \
-file oauth2_client_keystore_directory/oauth2-client.cert.pem
 \
-keystore oauth2_client_keystore_directory/cacerts.p12
 \
-storepass changeit
 \
-storetype PKCS12
 \
-alias client-cert

Certificate was added to keystore
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

9.5.3. Setting Up AM As an Authorization Server With mTLS

This section describes how to set up AM for this example. For more information about configuring AM as an OAuth 2.0 authorization service for clients using mTLS, see Authenticating Clients Using Mutual TLS in the AM *OAuth 2.0 Guide*.

To Set Up AM As an Authorization Server With mTLS

Before you start, install and configure AM on http://openam.example.com:8088/openam, with the default configuration. If you use a different configuration, substitute in the tutorial accordingly.

1. In AM, export an environment variable for the base64-encoded value of the password for the `cacerts.p12` truststore:

```
$ export PASSWORDSECRETID='Y2hhbmdlaXQ='
```

2. Configure an OAuth 2.0 Authorization Server:

    a. In the top level realm, select Configure OAuth Provider > Configure OAuth 2.0.

    b. Accept all of the default values and select Create.

    c. In Services on the left, select the OAuth2 Provider, and on the Advanced tab select the following options:

        - Support TLS Certificate-Bound Access Tokens: enabled
        - Supported Scopes: `test`

3. Create an OAuth2 Client to request OAuth 2.0 access_tokens:

    a. Select Applications > OAuth 2.0.

    b. Add a client with the following values:

        - Client ID: `client-application`
        - Scope(s): `test`

    c. On the Advanced tab, select the following options:

        - Grant Types: `Client Credentials`
        - Token Endpoint Authentication Method: `tls_client_auth`

    d. On the Signing and Encryption tab, select the following options:

        - mTLS Subject DN: `CN=test`
        - Use Certificate-Bound Access Tokens: Enabled

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

4. Create an OAuth2 Client authorized to examine (introspect) tokens:

    a. In the top level realm, select Applications > OAuth 2.0.

    b. Add a client with the following values:

        - Client ID: `resource-server`
        - Client secret `password`
        - Scope(s): `am-introspect-all-tokens`

    c. On the Advanced tab, select the following options:

        - Grant Types: none

5. Set up secret stores:

    a. Select Secret Stores, and add a store with the following values:

        - Secret Store ID: `trusted-ca-certs`
        - Store Type: `Keystore`
        - File: `<oauth2_client_keystore_directory>/cacerts.p12`
        - Keystore type: `PKCS12`
        - Store password secret ID: `passwordSecretId`

    b. Select Mappings and add the following mapping:

        - Secret ID: `am.services.oauth2.tls.client.cert.authentication`
        - Aliases: `client-cert`

        When the token endpoint authentication method is `tls_client_auth`, this secret is used to validate the client certificate. Add an alias in this list for each client that uses `tls_client_auth`. For certificates signed by a CA, add the CA certificate to the list.

9.5.4. Setting Up IG As a Resource Server With mTLS

To Set Up AM As a Resource Server With mTLS

1. Add the following route to the IG configuration as `$HOME/.openig/config/routes/mtls.json` (on Windows, `%appdata%\OpenIG\config\routes\mtls.json`).

```
{
  "name": "mtls",
  "condition": "${matches(request.uri.path, '/mtls')}",
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
  "handler": {
    "type": "Chain",
    "capture": "all",
    "config": {
      "filters": [
        {
          "name": "OAuth2ResourceServerFilter-1",
          "type": "OAuth2ResourceServerFilter",
          "config": {
            "scopes": [
              "test"
            ],
            "requireHttps": false,
            "accessTokenResolver": {
              "type": "ConfirmationKeyVerifierAccessTokenResolver",
              "config": {
                "delegate": {
                  "name": "token-resolver-1",
                  "type": "TokenIntrospectionAccessTokenResolver",
                  "config": {
                    "endpoint": "http://openam.example.com:8088/openam/oauth2/realms/root/introspect",
                    "providerHandler": {
                      "type": "Chain",
                      "config": {
                        "filters": [
                          {
                            "type": "HeaderFilter",
                            "config": {
                              "messageType": "request",
                              "add": {
                                "Authorization": [
                                  "Basic ${encodeBase64('resource-server:password')}"
                                ]
                              }
                            }
                          }
                        ],
                        "handler": "ForgeRockClientHandler"
                      }
                    }
                  }
                }
              }
            }
          }
        }
      ],
      "handler": {
        "name": "StaticResponseHandler-1",
        "type": "StaticResponseHandler",
        "config": {
          "status": 200,
          "entity": "mTLS\n Valid token: ${contexts.oauth2.accessToken.token}\n Confirmation keys:
${contexts.oauth2}"
        }
      }
    }
  }
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
}
```

2. Notice the following features of the route:

   - The route matches requests to `/mtls`.
   - The ConfirmationKeyVerifierAccessTokenResolver delegates the token resolution to the TokenIntrospectionAccessTokenResolver.

     The `providerHandler` adds an authorization header to the request, containing the username and password of the OAuth 2.0 client with the scope to examine (introspect) access_tokens.
   - If the ConfirmationKeyVerifierAccessTokenResolver validates the token, it passes the request to the StaticResponseHandler, which returns the content of the access_token from the context `${contexts.oauth2.accessToken.info}`. Otherwise, an error occurs.

9.5.5. Testing the Setup

To Test the Setup

1. Get an access_token from AM, over TLS:

```
$ mytoken=$(curl --request POST
 \
--cacert <am_keystore_directory>/openam-server.cert.pem
 \
--cert <oauth2_client_keystore_directory>/oauth2-client.cert.pem
 \
--key <oauth2_client_keystore_directory>/oauth2-client.key.pem
 \
--header 'cache-control: no-cache'
 \
--header 'content-type: application/x-www-form-urlencoded'
 \
--data 'client_id=client-application&grant_type=client_credentials&scope=test' \
https://openam.example.com:8445/openam/oauth2/access_token | jq -r .access_token)
```

2. Introspect the access_token on AM:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ curl --request POST
 \
-u resource-server:password
 \
--header 'content-type: application/x-www-form-urlencoded'
 \
--data token=${mytoken} \
http://openam.example.com:8088/openam/oauth2/realms/root/introspect | jq

{
  "active": true,
  "scope": "test",
  "client_id": "client-application",
  "user_id": "client-application",
  "token_type": "Bearer",
  "exp": 1550590833,
  "sub": "client-application",
  "iss": "http://openam.example.com:8088/openam/oauth2",
  "cnf": {
    "x5t...256": "T4u...R9Q"
  }
}
```

The `cnf` property indicates the value of the confirmation code.

3. Access the IG route to validate the confirmation key with the ConfirmationKeyVerifierAccessTokenResolver:

```
$ curl --request POST
 \
--cacert <ig_keystore_directory>/openig-server.cert.pem
 \
--cert <oauth2_client_keystore_directory>/oauth2-client.cert.pem
 \
--key <oauth2_client_keystore_directory>/oauth2-client.key.pem
 \
--header "authorization: Bearer ${mytoken}" \
https://openig.example.com:8443/mtls

mTLS
 Valid token: 2Bp...s_k
 Confirmation keys: {
  ...
}
```

The validated token and confirmation keys are displayed.

9.6. Using the OAuth 2.0 Context to Log In To the Sample Application

The token info endpoint returns the scopes `employeenumber` and `mail` in the context. This section contains an example route that retrieves the scopes, assigns them as the session username and password, and uses them to log the user directly in to the sample application.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

For information about the context, see OAuth2Context(5) in the *Configuration Reference*.

To Log In To the Sample Application By Using the Token Info

Before you start, run the example in "To Set Up IG As a Resource Server Using the Token Info Endpoint".

1. (For AM 6.5.3 and later versions) Select Services, and add a Validation Service with the following URLs for IG:

   - `http://openig.example.com:8080/*`
   - `http://openig.example.com:8080/*?*`

2. Edit the user created in "To Set Up a Sample User In AM", to set:

   - Email address: `george`
   - Employee number: `C0stanza`

   The IG route in this procedure assigns the value of `mail` as the username, and `employeenumber` as the password to log in to the sample application.

3. Edit the route, `rs-tokeninfo.json`, created in "To Set Up IG As a Resource Server Using the Token Info Endpoint":

   - Remove the StaticResponseHandler.
   - Add an AssignmentFilter to access the token info through the context, and inject the username and password into `session`.
   - Add a StaticRequestFilter to retrieve the username and password from `session`, and replace the original HTTP GET request with an HTTP POST login request that contains the credentials to authenticate.
   - Add a ReverseProxyHandler to submit the request to the sample application.

   An example route with a different name and condition is as follows:

```
{
  "name" : "rs-pwreplay",
  "baseURI" : "http://app.example.com:8081",
  "condition" : "${matches(request.uri.path, '^/rs-pwreplay')}",
  "heap" : [ {
    "name" : "AmService-1",
    "type" : "AmService",
    "config" : {
      "url" : "http://openam.example.com:8088/openam",
      "realm" : "/",
      "ssoTokenHeader" : "iPlanetDirectoryPro",
      "version": "6.5",
      "agent": {
        "username": "ig_agent",
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
        "password": "password"
      },
      "sessionCache": {
        "enabled": false
      }
    }
  } ],
  "handler" : {
    "type" : "Chain",
    "config" : {
      "filters" : [ {
        "name" : "OAuth2ResourceServerFilter-1",
        "type" : "OAuth2ResourceServerFilter",
        "config" : {
          "scopes" : [ "mail", "employeenumber" ],
          "requireHttps" : false,
          "realm" : "OpenIG",
          "accessTokenResolver" : {
            "name" : "token-resolver-1",
            "type" : "OpenAmAccessTokenResolver",
            "config" : {
              "amService" : "AmService-1"
            }
          }
        }
      },
        {
          "type": "AssignmentFilter",
          "config": {
            "onRequest": [{
              "target": "${session.username}",
              "value": "${contexts.oauth2.accessToken.info.mail}"
            },
              {
                "target": "${session.password}",
                "value": "${contexts.oauth2.accessToken.info.employeenumber}"
              }
            ]
          }
        },
        {
          "type": "StaticRequestFilter",
          "config": {
            "method": "POST",
            "uri": "http://app.example.com:8081/login",
            "form": {
              "username": [
                "${session.username}"
              ],
              "password": [
                "${session.password}"
              ]
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
}
```

> **Note**
>
> If necessary, change the value of `version` for AmService to your version of AM.

4. In a terminal window, use a **curl** command similar to the following to retrieve an access_token:

```
$ mytoken=$(curl -s
 \
--user "client-application:password"
 \
--data "grant_type=password&username=george&password=C0stanza&scope=mail%20employeenumber" \
http://openam.example.com:8088/openam/oauth2/access_token | jq -r ".access_token")
```

5. Validate the access_token returned in the previous step:

```
$ curl -v http://openig.example.com:8080/rs-pwreplay --header "Authorization: Bearer ${mytoken}"
```

HTML for the sample application is displayed.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 10

Acting As an OAuth 2.0 Client or OpenID Connect Relying Party

IG helps integrate applications into OAuth 2.0 and OpenID Connect deployments. In this chapter, you will:

- Configure IG as an OpenID Connect 1.0 relying party
- Configure IG to use OpenID Connect discovery and dynamic client registration

For an example configuration with two identity providers, AM and Google, and a login handler, see "Example Configuration For Multiple Identity Providers" in the *Configuration Reference*

10.1. About IG As an OAuth 2.0 Client

As described in "*Acting as an OAuth 2.0 Resource Server*", an OAuth 2.0 client is a third-party application that needs access to a user's protected resources.

IG can act as an OAuth 2.0 client when you configure an OAuth2ClientFilter as described in OAuth2ClientFilter(5) in the *Configuration Reference*. The OAuth2ClientFilter handles the process of allowing the user to select a provider, and redirecting the user through the authentication and authorization steps of an OAuth 2.0 authorization code grant. The code grant results in the authorization server returning an access token to the filter.

When an authorization grant succeeds, the OAuth2ClientFilter injects the access token data into a configurable target in the context so that subsequent filters and handlers have access to the access token. Subsequent requests can use the access token without reauthentication. If an authorization grant fails, the `failureHandler` is invoked.

If the protected application is an OAuth 2.0 resource server, then IG can send the access token with the resource request.

10.2. About IG As an OpenID Connect 1.0 Relying Party

The specifications available through the OpenID Connect site describe the OpenID Connect 1.0 authentication layer built on OAuth 2.0.

OpenID Connect 1.0 is a specific implementation of OAuth 2.0, where the identity provider holds the protected resource that the third-party application aims to access.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

OpenID Connect 1.0 has the following key entities:

- *End user*: An OAuth 2.0 resource owner whose user information the application needs to access.

  The end user wants to use an application through an existing identity provider account without signing up and creating credentials for another web service.

- *Relying Party* (RP): An OAuth 2.0 client that needs access to the end user's protected user information.

  For example, an online mail application needs to know which end user is accessing the application in order to present the correct inbox.

  As another example, an online shopping site needs to know which end user is accessing the site in order to present the right offerings, account, and shopping cart.

- *OpenID Provider* (OP): An OAuth 2.0 authorization server and also resource server that holds the user information and grants access.

  The OP has the end user consent to provide the RP with access to some of its user information. Because OpenID Connect 1.0 defines unique identification for an account (subject identifier + issuer identifier), the RP can use this as a key to its own user profile.

  For the online mail application, this key could be used to access the mailboxes and related account information. For the online shopping site, this key could be used to access the offerings, account, shopping cart and others. The key makes it possible to serve users as if they had local accounts.

- *UserInfo*: The protected resource that the third-party application aims to access. The information about the authenticated end-user is expressed in a standard format. The user-info endpoint is hosted on the authorization server and is protected with OAuth 2.0.

When IG acts as an OpenID Connect relying party, its ultimate role is to retrieve user information from the OpenID provider, and then to inject that information into the context for use by subsequent filters and handlers.

10.3. Installation Overview

The folllowing tasks set up AM as an OpenID Connect provider and IG as an OpenID Connect relying party.

Tasks for Configuring OpenID Connect

Task	See Section(s)
Set up AM as an OpenID Connect provider.	" Setting Up AM for OpenID Connect "
Set up IG as a relying party for browser requests to the home page of the sample application.	"Setting Up IG As a Relying Party"

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Task	See Section(s)
Test the configuration.	"Testing the Setup"

10.4. Setting Up AM for OpenID Connect

To Set Up AM as an OpenID Connect Provider

Before you start, install and configure AM on http://openam.example.com:8088/openam, with the default configuration. If you use a different configuration, substitute in the tutorial accordingly.

1. In AM, add a user as described in "To Set Up a Sample User In AM".
2. (For AM 6.5.3 and later versions) Select Services, and add a Validation Service with the following URLs for IG:
   - `http://openig.example.com:8080/*`
   - `http://openig.example.com:8080/*?*`
3. Configure an OAuth 2.0 Authorization Server and OpenID Connect Provider.
   a. In the top level realm, select Configure OAuth Provider > Configure OpenID Connect.
   b. Accept all of the default values and select Create.
4. Create an OAuth 2.0 Client to enable IG to communicate as an OAuth 2.0 relying party with AM.
   a. Select Applications > OAuth 2.0.
   b. Add a client with the following values:
      - Client ID: `oidc_client`
      - Client secret: `password`
      - Redirection URIs: `http://openig.example.com:8080/home/id_token/callback`
      - Scope(s): `openid`, `profile`, and `email`.
   c. (From AM 6.5) On the Advanced tab, select the following options:
      - Grant Types: `Authorization Code` and `Resource Owner Password Credentials`
   d. On the Signing and Encryption tab, change ID Token Signing Algorithm to `HS256`, `HS384`, or `HS512`.

      The algorithm must be HMAC.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

5. Log out of AM.

10.5. Setting Up IG As a Relying Party

This section describes how to use Studio to configure IG as a relying party for browser requests to the home page of the sample application. The example refers to the provider configuration in " Setting Up AM for OpenID Connect ".

To configure IG without using Studio, add the route in the following procedure to the IG configuration as `$HOME/.openig/config/routes/07-openid.json` (on Windows, `%appdata%\OpenIG\config\routes\07-openid.json`).

To Set Up IG As a Relying Party

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

1. In IG Studio, create a route:
   a. Browse to http://openig.example.com:8080/openig/studio, and select ⊕ Protect an Application.
   b. Choose ☰ Structured to use the predefined menus and templates.
2. Select Advanced options, and create a route with the following options:
   - Base URI: `http://app.example.com:8081`
   - Condition: Path: `/home/id_token`
   - Name: `07-openid`
3. Configure authentication:
   a. Select 👤 Authentication.
   b. Select OpenID Connect, and then select the following options to reflect the configuration in " To Set Up AM as an OpenID Connect Provider ":
      - Client Filter:
        - Client Endpoint: `/home/id_token`
        - Require HTTPS: Deselect this option
      - Client Registration:
        - Client ID: `oidc_client`
        - Client secret: `password`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Scopes: `openid`, `profile`, and `email`
- Basic authentication: Select this option

- Issuer:
  - Well-known Endpoint: `http://openam.example.com:8088/openam/oauth2/.well-known/openid-configuration`

Leave all other values as default.

4. On the top-right of the screen, select ⋮ and Display to review the route.

The following route should be displayed:

```
{
  "name": "07-openid",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/id_token')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "OAuth2ClientFilter-1",
          "type": "OAuth2ClientFilter",
          "config": {
            "clientEndpoint": "/home/id_token",
            "failureHandler": {
              "type": "StaticResponseHandler",
              "config": {
                "status": 500,
                "headers": {
                  "Content-Type": [
                    "text/plain"
                  ]
                },
                "entity": "An error occurred during the OAuth2 setup."
              }
            },
            "registrations": [
              {
                "name": "oidc-user-info-client",
                "type": "ClientRegistration",
                "config": {
                  "clientId": "oidc_client",
                  "clientSecret": "password",
                  "issuer": {
                    "name": "Issuer",
                    "type": "Issuer",
                    "config": {
                      "wellKnownEndpoint": "http://openam.example.com:8088/openam/oauth2/.well-known/
openid-configuration"
                    }
                  },
                  "scopes": [
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
                    "openid",
                    "profile",
                    "email"
                  ],
                  "tokenEndpointAuthMethod": "client_secret_basic"
                }
              }
            ],
            "requireHttps": false,
            "cacheExpiration": "disabled"
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

Notice the following features about the route:

- The route matches requests to `/home/id_token`.
- The `OAuth2ClientFilter` enables IG to act as a relying party. It uses a single client registration that is defined inline and refers to the AM server configured in " Setting Up AM for OpenID Connect ".
- The filter has a base client endpoint of `/home/id_token`, which creates the following service URIs:
  - Requests to `/home/id_token/login` start the delegated authorization process.
  - Requests to `/home/id_token/callback` are expected as redirects from the OAuth 2.0 Authorization Server (OpenID Connect provider). This is why the redirect URI in the client profile in AM is set to `http://openig.example.com:8080/home/id_token/callback`.
  - Requests to `/home/id_token/logout` remove the authorization state for the end user, and redirect to the specified URL if a `goto` parameter is provided.

  These endpoints are implicitly reserved. Attempts to access them directly can cause undefined errors.
- For convenience in this test, `"requireHttps"` is false. In production environments, set it to true. So that you see the delegated authorization process when you make a request, `"requireLogin"` has the default value `true`.
- The target for storing authorization state information is `${attributes.openid}`. This is where subsequent filters and handlers can find access tokens and user information.

5. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

10.6. Testing the Setup

To Test the Setup

1. Log out of AM.
2. Browse to http://openig.example.com:8080/home/id_token.

   The AM login page is displayed.
3. Log in as user `george`, password `C0stanza`, and then allow the application to access user information.

   The home page of the sample application is displayed.

   What's happening behind the scenes?

   - When IG gets the browser request, the OAuth2ClientFilter redirects you to authenticate with AM and authorize access to user information. AM then returns an access token to the filter.
   - The OAuth2ClientFilter uses the access token to get the user information, and then injects the authorization state information into `attributes.openid`.
   - The ReverseProxyHandler redirects the request to the home page of the sample application.

10.7. Authenticating Automatically to the Sample Application

To authenticate automatically to the sample application, change the family name of the user `george` to match the password `C0stanza`, and add a StaticRequestFilter like the following to the end of the chain in `07-openid.json`:

```
{
  "type": "StaticRequestFilter",
  "config": {
    "method": "POST",
    "uri": "http://app.example.com:8081/login",
    "form": {
      "username": [
        "${attributes.openid.user_info.sub}"
      ],
      "password": [
        "${attributes.openid.user_info.family_name}"
      ]
    }
  }
}
```

The StaticRequestFilter retrieves the username and password from the context, and replaces the original HTTP GET request with an HTTP POST login request containing credentials.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

10.8. Using OpenID Connect Discovery and Dynamic Client Registration

OpenID Connect defines mechanisms for discovering and dynamically registering with an identity provider that is not known in advance. These mechanisms are specified in OpenID Connect Discovery and OpenID Connect Dynamic Client Registration. IG supports discovery and dynamic registration. In this section you will learn how to configure IG to try these features with AM.

Although this section focuses on OpenID Connect dynamic registration, IG also supports dynamic registration as described in RFC 7591, *OAuth 2.0 Dynamic Client Registration Protocol*.

This section builds on the previous procedures in this chapter. Before you start, perform the steps in "Installation Overview". This tutorial requires a recent sample application, as the newer versions include a small WebFinger service that is used here.

To Set Up AM for OpenID Connect Dynamic Registration

Before you start, set up AM as described in " Setting Up AM for OpenID Connect ".

1. Select the user `george`, and change the last name to `C0stanza`. Note that, for this example, the last name must be the same as the password.
2. Select Services > OAuth2 Provider.
3. On the Client Dynamic Registration tab, enable Allow Open Dynamic Client Registration. In earlier versions of AM, this option is in Advanced OpenID Connect.

Preparing IG for Discovery and Dynamic Registration

1. Add the following route to the IG configuration as `$HOME/.openig/config/routes/07-discovery.json` (on Windows, `%appdata%\OpenIG\config\routes\07-discovery.json`):

```
{
  "heap": [
    {
      "name": "DiscoveryPage",
      "type": "StaticResponseHandler",
      "config": {
        "status": 200,
        "reason": "OK",
        "entity": "
          <!doctype html>
          <html>
          <head>
            <title>OpenID Connect Discovery</title>
            <meta charset='UTF-8'>
          </head>
          <body>
            <form id='form' action='/discovery/login?'>
              Enter your user ID or email address:
            <input type='text' id='discovery' name='discovery'
                   placeholder='george or george@example.com' />
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
            <input type='hidden' name='goto'
                   value='${contexts.router.originalUri}' />
            </form>
            <script>
            // The sample application handles the WebFinger request,
            // so make sure the request is sent to the sample app.
            window.onload = function() {
              document.getElementById('form').onsubmit = function() {
                // Fix the URL if not using the default settings.
                var sampleAppUrl = 'http://app.example.com:8081/';
                var discovery = document.getElementById('discovery');
                discovery.value = sampleAppUrl + discovery.value.split('@', 1)[0];
              };
            };
            </script>
          </body>
        </html>"
      }
    }
  ],
  "name": "07-discovery",
  "condition": "${matches(request.uri.path, '^/discovery')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "DynamicallyRegisteredClient",
          "type": "OAuth2ClientFilter",
          "config": {
            "clientEndpoint": "/discovery",
            "requireHttps": false,
            "requireLogin": true,
            "target": "${attributes.openid}",
            "failureHandler": {
              "type": "StaticResponseHandler",
              "config": {
                "comment": "Trivial failure handler for debugging only",
                "status": 500,
                "reason": "Error",
                "entity": "${attributes.openid}"
              }
            },
            "loginHandler": "DiscoveryPage",
            "metadata": {
              "client_name": "My Dynamically Registered Client",
              "redirect_uris": [
                "http://openig.example.com:8080/discovery/callback"
              ],
              "_scope": "openid profile email",
              "scopes": [
                "openid",
                "profile",
                "email"
              ]
            }
          }
        },
        {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
          "type": "StaticRequestFilter",
          "config": {
            "method": "POST",
            "uri": "http://app.example.com:8081/login",
            "form": {
              "username": [
                "${attributes.openid.user_info.sub}"
              ],
              "password": [
                "${attributes.openid.user_info.family_name}"
              ]
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

2. Consider the differences with `07-openid.json`:

   - The route matches requests to `/discovery`.
   - The StaticResponseHandler, `DiscoveryPage`, is used as a login handler for the client filter (there is no issuer or client registration), and serves an HTML page that provides the following information to IG:

     - The value of a `discovery` parameter.

       IG uses the value to perform OpenID Connect discovery. Examples from the specification include `acct:joe@example.com`, `https://example.com:8080/`, and `https://example.com/joe`. First, IG extracts the domain host and port from the value, and attempts to find a match in the `supportedDomains` lists for any issuers that are already configured for the route. If it finds a match, then it can potentially use the issuer's registration end point and avoid an additional request to look up the user's issuer using the WebFinger protocol. If there is no match in the supported domains lists, IG uses the `discovery` value as the `resource` for a WebFinger request according to the OpenID Connect Discovery protocol.

       On success, IG has either found an appropriate issuer in the configuration, or found the issuer using the WebFinger protocol. IG can thus proceed to dynamic client registration.

       The small JavaScript function in the HTML page transforms user input into a useful `discovery` value for IG. This is not a requirement for deployment, only a convenience for the purposes of this example. Alternatives are described in the discovery protocol specification.

     - The value of a `goto` parameter.

       The `goto` parameter takes a URI that tells IG where to redirect the end user's browser once the process is complete and IG has injected the OpenID Connect user information into the context. In this case, the user is redirected back to this route so that the innermost chain of the configuration can log the user in to the protected application.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- The OAuth2ClientFilter specifies the following configuration:
  - Login handler to point to the login page described above.
  - Metadata that IG uses to prepare the dynamic registration request, including:
    - Client name
    - Redirection URIs

      As set out in RFCs for OAuth2 and OpenID, redirection URIs are mandatory. One of the registered redirection URI values **must** exactly match the clientEndpoint/callback URI.
    - Scopes:
      - Use `scope` for dynamic client registration with AM 5.5 and later versions, or with identity providers that support RFC 7591, *OAuth 2.0 Dynamic Client Registration Protocol*.
      - Use `scopes` for dynamic client registration with all AM versions.
      - Use both `scope` and `scopes` to dynamically register with a wider range of identity providers.

To Test the Setup

1. Log out of AM.
2. Browse to http://openig.example.com:8080/discovery.
3. Enter the following email address: `george@example.com`.

   The AM login page is displayed.
4. Log in as user `george`, password `C0stanza`, and then allow the application to access user information.

   If successful, IG logs you in to the sample application as George Costanza, and the sample application returns George's page.

   What is happening behind the scenes?

   After IG gets the browser request, it returns the example page for discovery. You provide a user ID or email address, and the page transforms that into a `discovery` value. The value is tailored to let IG use the sample application as a WebFinger server. (The WebFinger server is more likely to be a service on the issuer's domain, not part of the protected application. For the purposes of this tutorial the WebFinger service has been embedded in the sample application to avoid leaving you with another server to manage during the tutorial.)

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

IG learns from the WebFinger service that AM is the issuer for the user. IG retrieves the OpenID Provider configuration from AM, and registers itself dynamically with AM, using the redirection URIs and scopes specified in the OAuth 2.0 client filter configuration.

Once the issuer and client registration are properly configured, the OAuth 2.0 client filter redirects the browser to AM for authentication and authorization to access to the user information. The rest is the same as the previous tutorial in this chapter. For details, see "Testing the Setup".

IG reuses issuer and client registration configurations that it builds after discovery and dynamic registration. These dynamically generated configuration objects are held in memory, and do not persist when IG is restarted.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 11

Transforming OpenID Connect ID Tokens Into SAML Assertions

This chapter builds on the example in "*Acting As an OAuth 2.0 Client or OpenID Connect Relying Party*" to use an IG TokenTransformationFilter to transform OpenID Connect ID tokens (id_tokens) issued by AM into SAML 2.0 assertions.

Many enterprises use existing or legacy, SAML 2.0-based SSO, but many mobile and social applications are managed by OpenID Connect. Use the IG TokenTransformationFilter to bridge the gap between OpenID Connect and SAML 2.0 frameworks.

11.1. About Token Transformation

The following figure illustrates the flow of information between a request, IG, the AM OAuth 2.0 and STS modules, and an application. For a more detailed view of the flow, see "Flow of Events".

Token Transformation

The process is as follows:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

1. A user tries to access to a protected resource.

2. If the user is not authenticated, the OAuth2ClientFilter redirects the request to AM. After authentication, AM asks for the user's consent to give IG access to private information.

3. If the user consents, AM returns an id_token to the OAuth2ClientFilter. The filter opens the id_token JWT and makes it available in `attributes.openid.id_token` and `attributes.openid.id_token_claims` for downstream filters.

4. The TokenTransformationFilter calls the AM STS to transform the id_token into a SAML 2.0 assertion.

5. The STS validates the signature, decodes the payload, and verifies that the user issued the transaction. The STS then issues a SAML assertion to IG on behalf of the user.

6. The TokenTransformationFilter makes the result of the token transformation available to downstream handlers in the `issuedToken` property of the `${contexts.sts}` context.

The following sequence diagram shows a more detailed view of the flow:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Flow of Events

Token Transformation Code Flow for Unauthenticated User Agent

IG
Relying Party

AM
Identity Provider

User Agent
OAuth2ClientFilter
TokenTransformationFilter
Authorization Server
User Info End Point
STS

OpenID Connect Authorization Flow

1 Request to access to route.
2 Redirect for authorization.
3 Request authorization.
4 User agent not authenticated
5 Request authentication.
6 Provide authentication.
7 Request consent to share private information with IG.
8 Give consent.
9 Redirect request and include authorization code.
10 Redirect authorization code.
11 Exchange authorizationcode for access_token and id_token
12 Validate id_token.
13 Use access_token to get other user info.
14 Return other user info.
15 Insert user info and tokens into the request context.
16 Display id_token

Add the token transformation filter to the route, and access the route again.

Token Transformation Code Flow for Authenticated User Agent

17 Request to access route.
18 Session valid so forward request.
19 Provide id_token and request transformation into SAML assertion.
20 Transform id_token.
21 Return the SAML assertion.
22 Insert SAML assertion into the dedicated context.
23 Display id_token and SAML assertion.

User Agent
OAuth2ClientFilter
TokenTransformationFilter
Authorization Server
User Info End Point
STS

11.2. Setting Up AM for Token Transformation

In this section, you configure an AM Security Token Service (STS), where the subject confirmation method is Bearer.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

The OpenID Connect ID token bearer module expects an id_token in an HTTP request header. It validates the id_token, and, if successful, looks up the AM user profile corresponding to the end user for whom the id_token was issued. Assuming the id_token is valid and the profile is found, the module authenticates the AM user.

The token bearer module specifies how AM gets the information to validate the id_token, which request header contains the id_token, the identifier for the provider who issued the id_token, and how to map the id_token claims to an AM user profile. The REST STS instance exposes a preconfigured transformation under a specific REST endpoint.

For more information about setting up a REST STS instance, see the AM *Security Token Service Guide* . For information about configuring keystores, see Setting Up Keys and Keystores in the AM *Setup and Maintenance Guide*.

To Set Up AM for Token Transformation

1. Set up AM as described in " Setting Up AM for OpenID Connect ". IG authenticates with AM and retrieves an OpenID Connect ID token (id_token) to be transformed by STS.
2. Add a Java agent as described in "To Set Up a Java Agent in AM".
3. Create a Bearer Module:
   a. In the top level realm, select Authentication > Modules.
   b. Add a module with the following values:
      - Module name: `oidc`
      - Type: `OpenID Connect id_token bearer`
   c. In the configuration page, enter the following values:
      - OpenID Connect validation configuration type: `Client Secret`
      - OpenID Connect validation configuration value: `password`

        This is the password of the OAuth 2.0/OpenID Connect client.
      - Client Secret: `password`
      - Name of OpenID Connect ID Token Issuer: `http://openam.example.com:8088/openam/oauth2`
      - Audience name: `oidc_client`

        This is the name of the OAuth 2.0/OpenID Connect client.
      - List of accepted authorized parties: `oidc_client`

   Leave all other values as default, and save your settings.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

4. Create an instance of STS REST.

   a. In the top level realm, select STS.

   b. Add a Rest STS instance with the following values, and then select Create:

      - Deployment Url Element: `openig`

        This value identifies the STS instance and is used by the `instance` parameter in the TokenTransformationFilter.

      - SAML2 TOKEN

        - SAML2 issuer Id: `OpenAM`

        - Service Provider Entity Id: `openig_sp`

        - NameIdFormat: Select `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`

        > **Note**
        >
        > For STS, it isn't necessary to create a SAML SP configuration in AM.

      - OPENID CONNECT TOKEN

        - OpenIdConnect Token Provider Issuer Id: `oidc`

        - Token signature algorithm: Enter a value that is consistent with " To Set Up AM as an OpenID Connect Provider ", for example, `HMAC SHA 256`

        - Client Secret (for HMAC-signed-tokens): `password`

        - Issued Tokens Audience: `oidc_client`

   c. Select the SAML2 Token tab, add the following values, and then save your changes:

      - Attribute Mappings: Add `password=mail` and `userName=uid`

5. Log out of AM.

11.3. Setting Up IG for Token Transformation

To configure IG without using Studio, add the route in the following procedure to the IG configuration as `$HOME/.openig/config/routes/50-idtoken.json`. On Windows, add the route as `%appdata%\OpenIG\config\routes\50-idtoken.json`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Set Up IG For Token Transformation

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

Before you start, prepare IG and the sample application as described in "*First Steps*" in the *Getting Started Guide*, and access Studio as described in "Accessing Studio" in the *Getting Started Guide*. IG must be running in development mode.

1. In IG Studio, create a route:

   a. Browse to http://openig.example.com:8080/openig/studio, and select ⊕ Protect an Application.

   b. Choose ☰ Structured to use the predefined menus and templates.

2. Select Advanced options, and create a route with the following options:

   - Base URI: `http://app.example.com:8081`
   - Condition: Path: `/home/id_token`
   - Name: `50-idtoken`

3. Configure authentication:

   a. Select 👤 Authentication.

   b. Select OpenID Connect, and enter the following information:

      - Client Filter:
        - Client Endpoint: `/home/id_token`
        - Require HTTPS: Deselect this option
      - Client Registration:
        - Client ID: `oidc_client`
        - Client secret: `password`
        - Scopes: `openid`, `profile`, and `email`
        - Basic authentication: Select this option
      - Issuer:
        - Well-known endpoint: `http://openam.example.com:8088/openam/oauth2/.well-known/openid-configuration`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Leave all other values as default, and save your settings.

4. Set up token transformation:

   a. Select ⇄ Token transformation, and then enable it.

   b. Enter the following information:

      - AM service: Configure an AM service to use for authentication and REST STS requests.
        - URI: `http://openam.example.com:8088/openam`
        - Version: The version of the AM instance, for example, `6.5`.
        - Agent: The credentials of the agent you created in AM.
          - Username: `ig_agent`
          - Password: `password`
      - Username: `oidc_client`
      - Password: `password`
      - id_token: `${attributes.openid.id_token}`
      - Instance: `openig`

5. Add a StaticResponseHandler:

   a. On the top-right of the screen, select ⋮ and Editor mode to switch into editor mode.

      Warning

      After switching to Editor mode, you cannot go back. You will be able to use the JSON file editor to manually edit the route, but will no longer be able use the full Studio interface to add or edit filters.

   b. Replace the last ReverseProxyHandler in the route with the following StaticResponseHandler, and then save the route:

```
"handler": {
  "type": "StaticResponseHandler",
  "config": {
    "entity": "{\"id_token\":\n\"${attributes.openid.id_token}\"} \n\n\n{\"saml_assertions\":\n
\"${contexts.sts.issuedToken}\"}",
      "reason": "Found",
      "status": 200
  }
}
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

6. On the top-right of the screen, select ⋮ and Display to review the route.

   Make sure that the following route is displayed:

```
{
  "name": "50-idtoken",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/id_token')}",
  "heap": [
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "http://openam.example.com:8088/openam",
        "realm": "/",
        "ssoTokenHeader": "iPlanetDirectoryPro",
        "version": "6.5",
        "agent": {
          "username": "ig_agent",
          "password": "password"
        },
        "sessionCache": {
          "enabled": false
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "OAuth2ClientFilter-1",
          "type": "OAuth2ClientFilter",
          "config": {
            "clientEndpoint": "/home/id_token",
            "failureHandler": {
              "type": "StaticResponseHandler",
              "config": {
                "status": 500,
                "headers": {
                  "Content-Type": [
                    "text/plain"
                  ]
                },
                "entity": "An error occurred during the OAuth2 setup."
              }
            },
            "registrations": [
              {
                "name": "oidc-user-info-client",
                "type": "ClientRegistration",
                "config": {
                  "clientId": "oidc_client",
                  "clientSecret": "password",
                  "issuer": {
                    "name": "Issuer",
                    "type": "Issuer",
                    "config": {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
                      "wellKnownEndpoint": "http://openam.example.com:8088/openam/oauth2/.well-known/
openid-configuration"
                    }
                  },
                  "scopes": [
                    "openid",
                    "profile",
                    "email"
                  ],
                  "tokenEndpointAuthMethod": "client_secret_basic"
                }
              }
            ],
            "requireHttps": false,
            "cacheExpiration": "disabled"
          }
        },
        {
          "name": "TokenTransformationFilter-1",
          "type": "TokenTransformationFilter",
          "config": {
            "username": "oidc_client",
            "password": "password",
            "idToken": "${attributes.openid.id_token}",
            "instance": "openig",
            "amService": "AmService-1"
          }
        }
      ],
      "handler": {
        "type": "StaticResponseHandler",
        "config": {
          "entity": "{\"id_token\":\n\"${attributes.openid.id_token}\"} \n\n\n{\"saml_assertions\":\n
\"${contexts.sts.issuedToken}\"}",
          "reason": "Found",
          "status": 200
        }
      }
    }
  }
}
```

Note

If necessary, change the value of `version` for AmService to your version of AM.

Notice the following features of the route:

- The route matches requests to `/home/id_token`.
- The AmService in the heap is used for authentication and REST STS requests.
- The OAuth2ClientFilter enables IG to act as an OpenID Connect relying party:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- The client endpoint is set to `/home/id_token`, so the service URIs for this filter on the IG server are `/home/id_token/login`, `/home/id_token/logout`, and `/home/id_token/callback`.
- For convenience in this test, `requireHttps` is false. In production environments, set it to true. So that you see the delegated authorization process when you make a request, `requireLogin` is true.
- The target for storing authorization state information is `${attributes.openid}`. Subsequent filters and handlers can find access tokens and user information at this target.

- The ClientRegistration holds configuration provided in " Setting Up AM for OpenID Connect ", and used by IG to connect with AM.
- The TokenTransformationFilter transforms an id_token into a SAML assertion:
  - The `id_token` parameter defines where this filter gets the id_token created by the `OAuth2ClientFilter`.

    The TokenTransformationFilter makes the result of the token transformation available to downstream handlers in the `issuedToken` property of the `${contexts.sts}` context.
  - The `instance` parameter must match the `Deployment URL Element` for the REST STS instance.

  Errors that occur during token transformation cause an error response to be returned to the client and an error message to be logged for the IG administrator.
- When the request succeeds, a StaticResponseHandler retrieves and displays the id_token from the target `{attributes.openid.id_token}`.

7. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

11.4. Testing the Setup

To Test the Setup

1. Browse to http://openig.example.com:8080/home/id_token.

   The AM login screen is displayed.

2. Log in to AM as user `george`, password `C0stanza`.

   An OpenID Connect request to access private information is displayed.

3. Select Allow.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

The id_token and SAML assertions are displayed:

```
{"id_token": "eyAidHlwIjogIkpXVCIsICJhbGciOiAiSFMyNTYiIH0.eyAiYXRfaGFzaCI6ICJ . . ."}

{"saml_assertions":
<"saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version= . . ."}
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 12

Supporting UMA Resource Servers

This chapter describes the experimental support that IG provides for building a User-Managed Access (UMA) resource server.

IG 5.5 and AM 5.5 add support for the User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization specifications. The text and examples in this chapter describe UMA 2.0, and are relevant for IG 5.5 and later versions used with AM 5.5 and later versions.

The examples in this chapter do not work for versions of IG or AM below 5.5. Refer to the documentation for those versions.

For information about UMA, see the AM *User-Managed Access (UMA) 2.0 Guide*

12.1. About IG As an UMA Resource Server

This section illustrates the position of IG as a resource server in an UMA environment, with AM as an authorization server.

For more information about the actions that form the UMA workflow, and the process flows for protecting a resource and performing an UMA 2.0 grant, see UMA 2.0 Actors and Actions in the AM *User-Managed Access (UMA) 2.0 Guide*.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

UMA Architecture

Requesting Party
Client
(on behalf of Requesting Party)
Requesting Party Token
Access
Redirect to Auth Server
Authorize
Protected Resource (requires RPT)
ForgeRock Identity Gateway Resource Server
Protect
Protection API Access Token
UMA Grant
Protection API (requires PAT)
ForgeRock Access Management Authorization Server
Manage
Control
Resource Owner

12.2. Sharing and Accessing Protected Resources

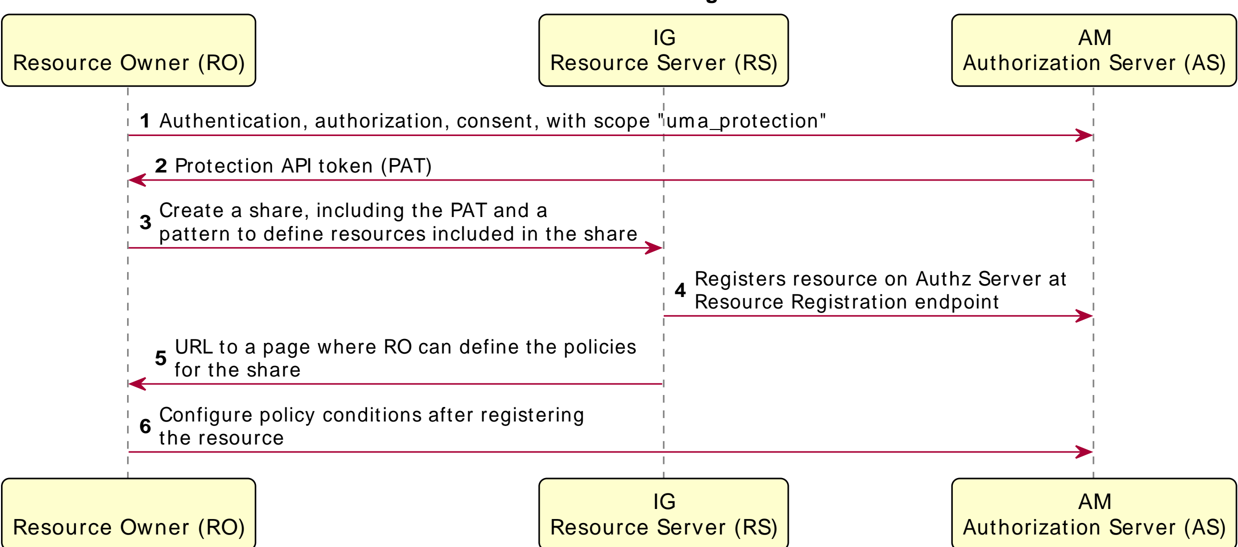
This section describes the data flows for registering protected resources and accessing protected resources with a Requesting Party Token (RPT).

For more information about the UMA 2.0 process flow, see UMA 2.0 Process Flow in the AM *User-Managed Access (UMA) 2.0 Guide*.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

The following sequence diagram outlines the data flow for a successful registration of a protected resource:

UMA 2.0 Protecting a Resource Flow



The following sequence diagram outlines the data flow for a successful UMA 2.0 grant flow, where the client accesses the protected resource:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

UMA 2.0 Grant Flow Process

UMA 2.0 Grant Flow

Client (On Behalf of Requesting Party) | IG Resource Server (RS) | AM Authorization Server (AS)

1 Request access to a shared resource

2 Find resource set in the shares configured for the request

3 Request a permission ticket for the resource being accessed

4 Permission ticket

5 HTTP 401 Unauthorized Response, including permission ticket and URL of UMA Authz Server

6 Authenticate with OpenID Connect

7 id_token

8 Request an RPT, including permission ticket and id_token

9 RPT

10 Request access to a shared resource, with the RPT

11 Introspect the RPT

12 Permissions associated with the RPT

alt **[If requesting party authorized for resource]**

13 Return the resource

[If requesting party not authorized for resource]

14 HTTP 401 Unauthorized Response, including permission ticket and URL of UMA Authz Server

Client (On Behalf of Requesting Party) | IG Resource Server (RS) | AM Authorization Server (AS)

12.3. Limitations of This Implementation

When using IG as an UMA resource server, note the following points:

- IG depends on the resource owner for the PAT.

  When a PAT expires, no refresh token is available to IG. The resource owner must repeat the entire share process with a new PAT in order to authorize access to protected resources. The resource owner should delete the old resource and create a new one.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Data about PATs and shared resources is held in memory.

  IG has no mechanism for persisting the data across restarts. When IG stops and starts again, the resource owner must repeat the entire share process.

- UMA client applications for sharing and accessing protected resources must deal with UMA error conditions and IG error conditions.

- By default, the REST API to manage share objects exposed by IG is protected only by CORS.

- When matching protected resource paths with share patterns, IG takes the longest match.

  For example, if resource owner Alice shares `/photos/.*` with Bob, and `/photos/vacation.png` with Charlie, and then Bob attempts to access `/photos/vacation.png`, IG applies the sharing permissions for Charlie, not Bob. As a result, Bob can be denied access.

12.4. Preparing the Tutorial

This tutorial describes one way of setting up IG as an UMA resource server. It uses AM as an authorization server for OAuth 2.0 and for UMA, and uses the sample application as a resource to protect and for files that serve as a basic UMA client.

Tasks for Configuring UMA

Task	See Section(s)
Modify the AM configuration to allow cross-site access.	"To Enable CORS Support for AM"
Configure AM as an authorization server.	"To Configure AM As an OAuth 2.0 Authorization Server and UMA Authorization Server"
Register client profiles in AM for OAuth 2.0 and UMA.	"To Register Client Profiles in AM"
Create a subject to act as a resource owner and a subject to act as a requesting party.	"To Create a Resource Owner and Requesting Party"
Set up the IG configuration for an UMA resource server	"To Set Up IG As an UMA Resource Server"
If you use a configuration that is different from that described in this chapter, adjust the sample to your configuration.	"Editing the Example to Match Custom Settings"

12.5. Setting Up AM As an Authorization Server

This section describes the following tasks to set up AM as an authorization server:

- Enabling cross-origin resource sharing (CORS) support in AM

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Configuring AM as an authorization server
- Registering UMA client profiles with AM
- Setting up a resource owner (Alice) and requesting party (Bob)

Before you start, install and configure AM on http://openam.example.com:8088/openam, with the default configuration. If you use a different configuration, substitute in the tutorial accordingly.

To Enable CORS Support for AM

For information about CORS support, see the AM product documentation. This procedure describes how to modify the AM configuration to allow cross-site access.

> **Caution**
>
> The settings in this section are suggestions for this tutorial, and are not intended as documentation for setting up AM CORS support on a server in production.
>
> If you need to accept all origins, by allowing the use of `Access-Control-Allowed-Origin=*`, do not allow `Content-Type` headers. Allowing the use of both types of header exposes AM to cross-site request forgery (CSRF) attacks.

1. In the `WEB-INF/web.xml` file of AM, edit the URL pattern for `CORSFilter` to match all endpoints:

```
<filter-mapping>
    <filter-name>CORSFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

2. In the same file, edit the `CORSFilter` configuration to authorize cross-site access for origins, hosts, and headers, as shown in the following excerpt:

```
<filter>
    <filter-name>CORSFilter</filter-name>
    <filter-class>org.forgerock.openam.cors.CORSFilter</filter-class>
    <init-param>
        <description>
            Accepted Methods (Required):
            A comma separated list of HTTP methods for which to accept CORS requests.
        </description>
        <param-name>methods</param-name>
        <param-value>POST,GET,PUT,DELETE,PATCH,OPTIONS</param-value>
    </init-param>
    <init-param>
        <description>
            Accepted Origins (Required):
            A comma separated list of origins from which to accept CORS requests.
        </description>
        <param-name>origins</param-name>
        <param-value>http://app.example.com:8081,http://openig.example.com:8080,http://
openam.example.com:8088</param-value>
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    </init-param>
    <init-param>
        <description>
            Allow Credentials (Optional):
            Whether to include the Vary (Origin)
            and Access-Control-Allow-Credentials headers in the response.
            Default: false
        </description>
        <param-name>allowCredentials</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <description>
            Allowed Headers (Optional):
            A comma separated list of HTTP headers
            which can be included in the requests.
        </description>
        <param-name>headers</param-name>
        <param-value>
          Authorization,Content-Type,iPlanetDirectoryPro,X-OpenAM-Username,X-OpenAM-
Password,Accept,Accept-Encoding,Connection,Content-Length,Host,Origin,User-Agent,Accept-
Language,Referer,Dnt,Accept-Api-Version,If-None-Match,Cookie,X-Requested-With,Cache-Control,X-Password,X-
Username,X-NoSession
        </param-value>
    </init-param>
    <init-param>
        <description>
            Expected Hostname (Optional):
            The name of the host expected in the request Host header.
        </description>
        <param-name>expectedHostname</param-name>
        <param-value>openam.example.com:8088</param-value>
    </init-param>
    <init-param>
        <description>
            Exposed Headers (Optional):
            The comma separated list of headers
            which the user-agent can expose to its CORS client.
        </description>
        <param-name>exposeHeaders</param-name>
        <param-value>Access-Control-Allow-Origin,Access-Control-Allow-Credentials,Set-Cookie,WWW-
Authenticate</param-value>
    </init-param>
    <init-param>
        <description>
            Maximum Cache Age (Optional):
            The maximum time that the CORS client can cache
            the pre-flight response, in seconds.
            Default: 600
        </description>
        <param-name>maxAge</param-name>
        <param-value>600</param-value>
    </init-param>
</filter>
```

3. Restart AM.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Configure AM As an OAuth 2.0 Authorization Server and UMA Authorization Server

1. Log in to the AM console as administrator.
2. In the top level realm, select Configure OAuth Provider > Configure OAuth 2.0, accept the default values, and select Create.

   The AM service `OAuth2 Provider` is created for the authorization endpoint.

   The PAT is obtained through the OAuth 2.0 access token endpoint. The RPT is obtained through the UMA endpoint.

   If you plan to build your own examples or modify the sample clients, consider extending the default token lifetimes.
3. Select Configure OAuth Provider > Configure User Managed Access, accept the default values, and select Create.

   The AM service `UMA Provider` is created.

To Register Client Profiles in AM

Follow these steps to register client profiles for OAuth 2.0 and UMA:

1. In the top level realm, select Applications > OAuth 2.0.
2. Add an OAuth 2.0 client to use for UMA protection:
   a. Add a client with the following values:
      - Client ID: `OpenIG`
      - Client secret: `password`
      - Scope: `uma_protection`

   b. (From AM 6.5) On the Advanced tab, select the following option:
      - Grant Types: `Resource Owner Password Credentials`
3. Add an OAuth 2.0 client to use for accessing protected resources:
   a. Add a client with the following values:
      - Client ID: `UmaClient`
      - Client secret: `password`
      - Scope: `openid`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

b. (From AM 6.5) On the Advanced tab, select the following option:

- Grant Types:
  - `Resource Owner Password Credentials`
  - `UMA`: The client (on behalf of the requesting party) needs the UMA grant type to access the protected resources.

To Create a Resource Owner and Requesting Party

Follow these steps to create identities in the top level realm:

1. In the top level realm, select Identities.
2. Add a new identity to act as a resource owner, with the following values:
   - ID: `alice`
   - Password: `UMAexample`
3. Add a new identity to act as a requesting party, with the following values:
   - ID: `bob`
   - Password: `UMAexample`

12.6. Setting Up IG As an UMA Resource Server

To Set Up IG As an UMA Resource Server

Before you start, prepare IG and the sample application as described in "*First Steps*" in the *Getting Started Guide*.

1. Add the following file as `$HOME/.openig/config/admin.json` (on Windows, `%appdata%\OpenIG\config\admin.json`):

```
{
  "heap": [
    {
      "name": "ClientHandler",
      "type": "ClientHandler"
    },
    {
      "name": "ApiProtectionFilter",
      "type": "ScriptableFilter",
      "config": {
        "type": "application/x-groovy",
        "file": "CorsFilter.groovy"
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
      }
    }
  ],
  "prefix": "openig"
}
```

To allow CORS support for the UMA share API, this route overrides the default ApiProtectionFilter that protects the reserved administrative route. By default, the administrative route is under `/openig`. For information, see AdminHttpApplication(5) in the *Configuration Reference*.

2. Add the following script to the IG configuration as `$HOME/.openig/scripts/groovy/CorsFilter.groovy` (on Windows, `%appdata%\OpenIG\scripts\groovy\CorsFilter.groovy`):

```
import org.forgerock.http.protocol.Response
import org.forgerock.http.protocol.Status

if (request.method == 'OPTIONS') {
    /**
     * Supplies a response to a CORS preflight request.
     *
     * Example response:
     *
     * HTTP/1.1 200 OK
     * Access-Control-Allow-Origin: http://app.example.com:8081
     * Access-Control-Allow-Methods: POST
     * Access-Control-Allow-Headers: Authorization
     * Access-Control-Allow-Credentials: true
     * Access-Control-Max-Age: 3600
     */

    def origin = request.headers['Origin']?.firstValue
    def response = new Response(Status.OK)

    // Browsers sending a cross-origin request from a file might have Origin: null.
    response.headers.put("Access-Control-Allow-Origin", origin)
    request.headers['Access-Control-Request-Method']?.values.each() {
        response.headers.add("Access-Control-Allow-Methods", it)
    }
    request.headers['Access-Control-Request-Headers']?.values.each() {
        response.headers.add("Access-Control-Allow-Headers", it)
    }
    response.headers.put("Access-Control-Allow-Credentials", "true")
    response.headers.put("Access-Control-Max-Age", "3600")

    return response
}

return next.handle(context, request)
/**
 * Adds headers to a CORS response.
 */
        .thenOnResult({ response ->
    if (response.status.isServerError()) {
        // Skip headers if the response is a server error.
    } else {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
        def headers = [
                "Access-Control-Allow-Origin": request.headers['Origin']?.firstValue,
                "Access-Control-Allow-Credentials": "true",
                "Access-Control-Expose-Headers": "WWW-Authenticate"
        ]
        response.headers.addAll(headers)
    }
})
```

This script adds a CORS filter to include headers for cross-origin requests.

The tutorial involves JavaScript clients that are served by the sample application, and so not from the same origin as AM or IG. The route uses a CORS filter to include appropriate response headers for cross-origin requests.

The CORS filter handles pre-flight (HTTP OPTIONS) requests, and responses for all HTTP operations. The logic for the filter is provided through the script.

The filter adds the appropriate headers to CORS requests. Pre-flight requests are diverted to a dedicated handler, which returns the response directly to the user agent. For all other requests, the headers are added to the response.

For information about scripting filters and handlers, see "*Extending IG*".

3. Add the following route to the IG configuration as `$HOME/.openig/config/routes/00-uma.json` (on Windows, `%appdata%\OpenIG\config\routes\00-uma.json`).

```
{
  "heap": [
    {
      "name": "UmaService",
      "type": "UmaService",
      "config": {
        "protectionApiHandler": "ClientHandler",
        "wellKnownEndpoint": "http://openam.example.com:8088/openam/uma/.well-known/uma2-configuration",
        "resources": [
          {
            "comment": "Protects all resources matching the following pattern.",
            "pattern": ".*",
            "actions": [
              {
                "scopes": [
                  "#read"
                ],
                "condition": "${request.method == 'GET'}"
              },
              {
                "scopes": [
                  "#create"
                ],
                "condition": "${request.method == 'POST'}"
              }
            ]
          }
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
        ]
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ScriptableFilter",
          "config": {
            "type": "application/x-groovy",
            "file": "CorsFilter.groovy"
          }
        },
        {
          "type": "UmaFilter",
          "config": {
            "protectionApiHandler": "ClientHandler",
            "umaService": "UmaService"
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  },
  "condition": "${request.uri.host == 'app.example.com'}"
}
```

Notice the following features of the route:

- The `UmaService` describes the resources that a resource owner can share, using AM as the authorization server.

  The `UmaService` provides a REST API to manage sharing of resource sets.

- The handler for the route chains together the CORS filter, the `UmaFilter`, and the default handler.

  The `UmaFilter` manages requesting party access to protected resources, using the `UmaService`. Protected resources are on the sample application, which responds to requests on port 8081.

- The route matches requests to `app.example.com`.

4. Restart IG to reload the configuration.

12.7. Test the Configuration

Follow these steps to test the configuration and demonstrate IG acting as an UMA resource server:

1. If necessary, log out of AM, and then browse to http://app.example.com:8081/uma/.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

If you used the settings described in this chapter and "Installing the Sample Application" in the *Getting Started Guide*, your configuration should match the displayed configuration. If you used other settings, you might need to edit the sample files to match your settings. For information, see "Editing the Example to Match Custom Settings".

2. Select the link to demonstrate Alice sharing resources.

3. On Alice's page, select Share with Bob to simulate Alice sharing resources as described in "Sharing and Accessing Protected Resources".

   The following items are displayed:

   - The PAT that Alice receives from AM
   - The metadata for the resource set that Alice registers through IG
   - The result of Alice authenticating with AM in order to create a policy
   - The successful result when Alice configures the authorization policy attached to the shared resource.

   If the step fails, get help in "Troubleshooting the UMA Example".

4. Go back to the first page, and select the link to demonstrate Bob accessing resources.

5. On Bob's page, select Get Alice's resources to simulate Bob accessing one of Alice's resources.

   The following items are displayed:

   - The WWW-Authenticate Header
   - The OpenID Connect Token that Bob gets to obtain the RPT
   - The RPT that Bob gets in order to request the resource again
   - The final response containing the body of the resource

12.8. Editing the Example to Match Custom Settings

If you use a configuration that is different to that described in this chapter, consider the following tasks to adjust the sample to your configuration:

1. Unpack the UMA files from the sample application described in "Installing the Sample Application" in the *Getting Started Guide* to temporary folder:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ mkdir /tmp/uma
$ cd /tmp/uma
$ jar -xvf /path/to/IG-sample-application-6.5.4.jar uma
 created: uma/
inflated: uma/alice.html
inflated: uma/bob.html
inflated: uma/common.js
inflated: uma/index.html
inflated: uma/style.css
```

2. Edit the configuration in `common.js`, `alice.html`, and `bob.html` to match your settings.

3. Repack the UMA sample client files and then restart the sample application:

```
$ jar -uvf /path/to/IG-sample-application-6.5.4.jar uma
adding: uma/(in = 0) (out= 0)(stored 0%)
adding: uma/index.html(in = 1698) (out= 880)(deflated 48%)
adding: uma/common.js(in = 4265) (out= 1319)(deflated 69%)
adding: uma/bob.html(in = 5427) (out= 1811)(deflated 66%)
adding: uma/style.css(in = 1403) (out= 696)(deflated 50%)
adding: uma/alice.html(in = 5494) (out= 1762)(deflated 67%)
```

4. If necessary, adjust the CORS settings for AM.

12.9. Understanding the UMA API With an API Descriptor

The UMA share endpoint serves API descriptors at runtime. When you retrieve an API descriptor for the endpoint, a JSON that describes the API for the endpoint is returned.

You can use the API descriptor with a tool such as Swagger UI to generate a web page that helps you to view and test the endpoint. For information, see "Understanding IG APIs With API Descriptors".

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 13
Configuring Routers and Routes

IG provides routers and routes to handle requests and their context. In this chapter, you will learn about:

- How routers and routes are configured
- How to read, create, edit, or delete routes through Common REST or Studio
- How to prevent changes to routes when IG is running

13.1. Configuring Routers

The following `config.json` file configures a Router:

```
{
  "handler": {
    "type": "Router",
    "name": "_router",
    "baseURI": "http://app.example.com:8081",
    "capture": "all"
  },
  "heap": [
    {
      "name": "JwtSession",
      "type": "JwtSession"
    },
    {
      "name": "capture",
      "type": "CaptureDecorator",
      "config": {
        "captureEntity": true,
        "_captureContext": true
      }
    }
  ]
}
```

In this configuration, all requests are passed with the default settings to the Router. The Router scans `$HOME/.openig/config/routes` at startup, and rescans the directory every 10 seconds. If routes have been added, deleted, or changed, the router applies the changes.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

The main router and any subrouters are used to build the monitoring endpoints. For information about monitoring endpoints, see Monitoring Endpoints(5) in the *Configuration Reference*. For information about the parameters of a router, see Router(5) in the *Configuration Reference*.

13.2. Configuring Routes

Routes are JSON configuration files that handle requests and their context, and then hand off any request they accept to a handler. Another way to think of a route is like an independent dispatch handler, as described in DispatchHandler(5) in the *Configuration Reference*.

Routes can have a base URI to change the scheme, host, and port of the request.

For information about the parameters of routes, see Route(5) in the *Configuration Reference*.

13.2.1. Configuring Objects Inline or In the Heap

If you use an object only once in the configuration, you can declare it inline in the route and do not need to name it. However, when you need use an object multiple times, declare it in the heap, and then reference it by name in the route.

The following route shows an inline declaration for a handler. The handler is a router to route requests to separate route configurations:

```
{
    "handler": {
        "type": "Router"
    }
}
```

The following example shows a named router in the heap, and a handler references the router by its name:

```
{
    "handler": "My Router",
    "heap": [
        {
            "name": "My Router",
            "type": "Router"
        }
    ]
}
```

Notice that the heap takes an array. Because the heap holds all configuration objects at the same level, you can impose any hierarchy or order when referencing objects. Note that when you declare all objects in the heap and reference them by name, neither hierarchy nor ordering are obvious from the structure of the configuration file alone.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

13.2.2. Setting Route Conditions

When a route has a condition, it can handle only requests that meet the condition. When a route has no condition, it can handle any request.

A condition can be based on almost any characteristic of the request, context, or IG runtime environment. Conditions are defined using IG expressions, as described in Expressions(5) in the *Configuration Reference*.

Because routes define the conditions on which they accept a request, the router does not have to know about specific routes in advance. In other words, you can configure the router first and then add routes while IG is running.

The following example shows a route with no condition. This route accepts any request:

```
{
  "name": "myroute",
  "handler": {
    "type": "ReverseProxyHandler"
  }
}
```

The following example shows the same route with a condition. This route accepts only requests whose path starts with `mycondition`:

```
{
  "name": "myroute",
  "handler": {
    "type": "ReverseProxyHandler"
  },
  "condition": "${matches(request.uri.path, '^/mycondition')}"
}
```

The following table lists some of the conditions used in routes in this guide:

Route Conditions

Route Condition	Some Example Requests That Meet the Condition
`"${matches(request.uri.path, '^/login')}"`	`http://app.example.com/login`, . . .
`"${request.uri.host == 'api.example.com'}"`	`http://api.example.com/`, `https://api.example.com/home`, `http://api.example.com:8080/home`, . . .
`"${matches(contexts.client.remoteAddress, '127.0.0.1')}"`	`http://localhost:8080/keygen`, `http://127.0.0.1:8080/keygen`, . . . Where `/keygen` is not mandatory and could be anything else.
`"${matches(request.uri.query, 'demo=simple')}"`	`http://openig.example.com:8080/login?demo=simple`, . . . For information about URI query, see `query` in URI(5) in the *Configuration Reference*.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Route Condition	Some Example Requests That Meet the Condition
`"${request.uri.scheme == 'http'}"`	http://openig.example.com:8080, . . .
`"${matches(request.uri.path, '^/dispatch') or matches(request.uri.path, '^/mylogin')}"`	http://openig.example.com:8080/dispatch, http://openig.example.com:8080/mylogin, . . .
`"${request.uri.host == 'sp1.example.com' and not matches(request.uri.path, '^/saml')}"`	http://sp1.example.com:8080/, http://sp1.example.com/mypath, . . . Not http://sp1.example.com:8080/saml, http://sp1.example.com/saml, . . .

13.2.3. Configuring Route Names, IDs, and Filenames

The filenames of routes have the extension `.json`, in lowercase.

The Router scans the routes folder for files with the `.json` extension, and uses the route's `name` property to order the routes in the configuration. If the route does not have a `name` property, the Router uses the route ID.

The route ID is managed as follows:

- When you add a route manually to the routes folder, the route ID is the value of the `_id` field. If there is no `_id` field, the route ID is the filename of the added route.
- When you add a route through the Common REST endpoint, the route ID is the value of the mandatory `_id` field.
- When you add a route through Studio, you can edit the default route ID.

> **Caution**
>
> The filename of a route cannot be `default.json`, and the route's `name` property and route ID cannot be `default`.

13.3. Creating and Editing Routes Through Common REST

> **Note**
>
> When IG is in production mode, you cannot manage, list, or even read routes through Common REST. For more information, see "Making the Configuration Immutable".

> **Note**
>
> If an AM policy agent is configured in the same container as IG, by default the policy agent intercepts requests to manage routes. When you try to add a route through Common REST, the policy agent redirects the request to AM and the route is not added.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To override this behavior, add the URL pattern `/openig/api/*` to the list of not-enforced URI in the policy agent profile. For more information about configuring policy agents in AM, see the *Java Agents Guide* .

Through Common REST, you can read, add, delete, and edit routes on IG without manually accessing the file system. You can also list the routes in the order that they are loaded in the configuration, and set fields to filter the information about the routes.

The following examples show some ways to manage routes through Common REST. For more information, see "About ForgeRock Common REST" in the *Configuration Reference*.

To Manage Routes Through Common REST

Before you start, prepare IG as described in "*First Steps*" in the *Getting Started Guide*.

1. Add the following route to the IG configuration as `$HOME/.openig/config/routes/00-crest.json`.

   On Windows, add the route as `%appdata%\OpenIG\config\routes\00-crest.json`.

   ```
   {
     "name": "crest",
     "handler": {
       "type": "StaticResponseHandler",
       "config": {
         "status": 200,
         "reason": "OK",
         "entity": "Hello, world!"
       }
     },
     "condition": "${matches(request.uri.path, '^/crest')}"
   }
   ```

   To check that the route is working, access the route on: http://openig.example.com:8080/crest.

2. To read a route through Common REST:

   - Enter the following command in a terminal window:

     ```
     $ http GET http://openig.example.com:8080/openig/api/system/objects/_router/routes/00-crest
     ```

     The route is displayed. Note that the route `_id` is displayed in the JSON of the route.

3. To add a route through Common REST:

   - Move `$HOME/.openig/config/routes/00-crest.json` to `/tmp/00-crest.json`.
   - Check in `$HOME/.openig/logs/route-system.log` that the route has been removed from the configuration. To double check, access http://openig.example.com:8080/crest. You should get an HTTP 404.
   - Enter the following command in a terminal window:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ http PUT http://openig.example.com:8080/openig/api/system/objects/_router/routes/00-crest < /tmp
/00-crest.json
```

This command posts the file in `/tmp/00-crest.json` to the `routes` directory.

- Check in `$HOME/.openig/logs/route-system.log` that the route has been added to configuration. To double-check, access http://openig.example.com:8080/crest. You should see the "Hello, world!" message.

4. To edit a route through Common REST:

   - Edit `/tmp/00-crest.json` to change the message displayed by the response handler in the route.
   - Enter the following command in a terminal window:

   ```
   $ http PUT http://openig.example.com:8080/openig/api/system/objects/_router/routes/00-crest If-
   Match:* < /tmp/00-crest.json
   ```

   This command deploys the route with the new configuration. Because the changes are persisted into the configuration, the existing `$HOME/.openig/config/routes/00-crest.json` is replaced with the edited version in `/tmp/00-crest.json`.

   - Check in `$HOME/.openig/logs/route-system.log`, that the route has been updated. To double-check, access http://openig.example.com:8080/crest to confirm that the displayed message has changed.

5. To delete a route through Common REST:

   - Enter the following command in a terminal window:

   ```
   $ $ http DELETE http://openig.example.com:8080/openig/api/system/objects/_router/routes/00-crest
   ```

   - Check in `$HOME/.openig/logs/route-system.log` that the route has been removed from the configuration. To double-check, access http://openig.example.com:8080/crest. You should get an HTTP 404.

6. To list the routes deployed on the router, in the order that they are tried by the router:

   - Enter the following command in a terminal window:

   ```
   $ http "http://openig.example.com:8080/openig/api/system/objects/_router/routes?_queryFilter=true"
   ```

   The list of loaded routes is displayed.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

13.4. Creating Routes Through Studio

> **Note**
>
> When IG is in production mode, Studio is effectively disabled. For more information, see "Making the Configuration Immutable".

Studio is a user interface to configure and deploy routes in IG. You can use Studio to create routes for tasks such as authenticating users, and authorizing access to APIs, throttling the rate of requests to protected applications, capturing messages, and collecting statistics. New features will be added as Studio evolves.

When IG is installed and running as described in this guide, access Studio on http://openig.example.com:8080/openig/studio.

For help to get started with Studio, see "*Configuring Routes With Studio*" in the *Getting Started Guide*. For examples of how to use Studio to configure routes, see the following sections of this guide:

- To configure IG to enforce AM policy decisions, see "To Set Up IG as a PEP".
- To configure IG as a resource server using the token info endpoint, see "To Set Up IG As a Resource Server Using the Token Info Endpoint".
- To configure IG as a relying party for OpenID Connect 1.0, see "To Set Up IG As a Relying Party".
- To configure a simple throttling filter, see "To Configure a Simple Throttling Filter".

13.5. Preventing the Reload of Routes

To prevent routes from being reloaded after startup, stop IG, edit the router `scanInterval`, and restart IG. When the interval is set to `disabled`, routes are loaded only at startup:

```
{
    "name": "Router",
    "type": "Router",
    "config": {
        "scanInterval": "disabled"
    }
}
```

The following example changes the location where the router looks for the routes:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
{
    "name": "Router",
    "type": "Router",
    "config": {
        "directory": "/path/to/safe/routes",
        "scanInterval": "disabled"
    }
}
```

13.6. Accessing Reserved Routes

IG uses an `ApiProtectionFilter` to protect the reserved routes. By default, the filter allows access to reserved routes only from the loopback address. To override this behavior, declare a custom `ApiProtectionFilter` in the top-level heap. For an example, see the CORS filter described in "To Set Up IG As an UMA Resource Server".

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 14
Proxying WebSocket Traffic

When a user agent requests an upgrade from HTTP or HTTPS to the WebSocket protocol, IG detects the request and performs an HTTP handshake request between the user agent and the protected application.

If the handshake is successful, IG upgrades the connection and provides a dedicated tunnel to route WebSocket traffic between the user agent and the protected application.

The tunnel remains open until it is closed by the user agent or protected application. When the user agent closes the tunnel, the connection between IG and the protected application is automatically closed.

The following sequence diagram shows the flow of information when IG proxies WebSocket traffic:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Flow of Information to Proxy WebSocket Traffic

WebSocket client | IG | WebSocket server

Perform handshake and create tunnel

WebSocket handshake request

HTTP filtering
(for example, execute IG filters for authentication)

WebSocket handshake request

Creation of WebSocket tunnel

WebSocket handshake response

Upgrade Client-IG connection from
HTTP to WebSocket protocol

WebSocket handshake response

Send message

WebSocket frame

Push through tunnel

WebSocket frame

Receive message

WebSocket frame

Push through tunnel

WebSocket frame

WebSocket client | IG | WebSocket server

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To set up IG to proxy WebSocket traffic, configure the `websocket` property of ReverseProxyHandler. By default, IG does not proxy WebSocket traffic. For more information, see ReverseProxyHandler(5) in the *Configuration Reference*.

14.1. Configuring IG to Proxy WebSocket Traffic

The following procedures provide examples of how to set up and test proxying for WebSocket traffic. The example uses the WebSocket server provided in the sample app, and uses the SingleSignOnFilter for authentication.

To Configure Proxying for WebSocket Traffic

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

Before you start:

- Prepare IG and the sample app as described in "*First Steps*" in the *Getting Started Guide*
- Install and configure AM on http://openam.example.com:8088/openam, using the default configuration.

1. Set up AM:

   a. (For AM 6.5.3 and earlier versions) Select Identities > demo, and set the demo user password to `Ch4ng31t`.

   b. (For AM 6.5.3 and later versions) Select Services, and add a Validation Service with the following URLs for IG:

      - `http://openig.example.com:8080/*`
      - `http://openig.example.com:8080/*?*`

   c. (For AM 6.5.3 and later versions) Select Applications > Agents > Identity Gateway, and add an agent with the following values:

      - Agent ID: `ig_agent`
      - Password: `password`

      Leave all other values as default.

      (For AM 6.5.2 and earlier versions) Set up an agent as described in "To Set Up a Java Agent in AM".

2. In IG Studio, create a route:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

a. Browse to http://openig.example.com:8080/openig/studio, and select Protect an Application.

b. Choose Structured to use the predefined menus and templates.

3. Select Advanced options, and create a route with the following options:

   - Base URI: `http://app.example.com:8081`
   - Condition: Path: `/websocket`
   - Name: `websocket`
   - Enable WebSocket: Select this option

4. Configure authentication:

   a. Select Authentication.

   b. Select Single Sign-On, and enter the following information:

   - AM service:
     - URI: `http://openam.example.com:8088/openam`
     - Version: The version of the AM instance, for example, `6.5`.
     - Agent: The credentials of the Java agent you created in AM.
       - Username: `ig_agent`
       - Password: `password`

   Leave all other values as default.

5. On the top-right of the screen, select and Display to review the route.

   The following route should be displayed:

```
{
  "name": "websocket",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/websocket')}",
  "heap": [
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "http://openam.example.com:8088/openam",
        "realm": "/",
        "ssoTokenHeader": "iPlanetDirectoryPro",
        "version": "6.5",
        "agent": {
          "username": "ig_agent",
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
          "password": "password"
        },
        "sessionCache": {
          "enabled": false
        }
      }
    },
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "config": {
        "websocket": {
          "enabled": true
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "SingleSignOnFilter-1",
          "type": "SingleSignOnFilter",
          "config": {
            "amService": "AmService-1"
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

6. Notice the following features of the route:
   - The route matches requests to `/websocket`, the endpoint on the sample app that exposes a WebSocket server.
   - The SingleSignOnFilter redirects unauthenticated requests to AM for authentication.
   - The ReverserProxyHandler enables IG to proxy WebSocket traffic, and, after IG upgrades the HTTP connection to the WebSocket protocol, passes the messages to the WebSocket server.
7. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

To Test the Setup

1. If necessary log out of AM, and then browse to http://openig.example.com:8080/websocket.

   The SingleSignOnFilter redirects you to AM for authentication.
2. Log in to AM as user `demo`, password `Ch4ng31t`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

AM authenticates the user, creates an SSO token, and redirects the request back to the original URI, with the token in a cookie.

The request then passes to the ReverseProxyHandler, which routes the request to the HTML page `/websocket/index.html` of the sample app. The page initiates the HTTP handshake for connecting to the WebSocket endpoint `/websocket/echo`.

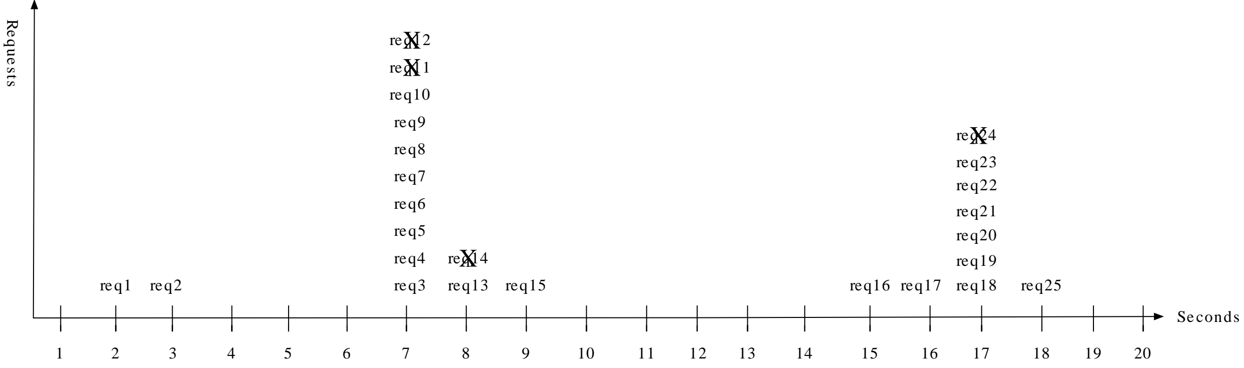
3. Enter text on the WebSocket echo screen, and note that the text is echoed back.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 15
Throttling the Rate of Requests to Protected Applications

To protect applications from being overused by clients, use a throttling filter to limit how many requests can be made in a defined time. The maximum number of requests that are allowed in a defined time is called the *throttling rate*.

The throttling filter uses the token bucket algorithm, allowing some unevenness or bursts in the request flow. The following image shows how IG manages requests for a throttling rate of 10 requests/10 seconds:



- At 7 seconds, 2 requests have previously passed when there is a burst of 9 requests. IG allows 8 requests, but disregards the 9th because the throttling rate for the 10-second throttling period has been reached.
- At 8 and 9 seconds, although 10 requests have already passed in the 10-second throttling period, IG allows 1 request each second.
- At 17 seconds, 4 requests have passed in the previous 10-second throttling period, and IG allows another burst of 6 requests.

When the throttling rate is reached, IG issues an HTTP status code 429 `Too Many Requests` and a `Retry-After` header like the following, where the value is the number of seconds to wait before trying the request again:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
GET http://openig.example.com:8080/home/throttle-scriptable HTTP/1.1

. . .

HTTP/1.1 429 Too Many Requests
Retry-After: 10
```

This section describes how to set up simple, mapped, and scriptable throttling filters. For more configuration options, see ThrottlingFilter(5) in the *Configuration Reference*

15.1. Configuring a Simple Throttling Filter

This section describes how to use Studio to configure a simple throttling filter that applies a throttling rate of 6 requests/10 seconds. When an application is protected by this throttling filter, no more than 6 requests, irrespective of their origin, can access the sample application in a 10 second period.

Simple Throttling Filter

User
All requests
IG
Throttling rate
6 req/10 sec
Throttled requests
Sample application

To try this example without using Studio, add the route in the following procedure to the IG configuration as `$HOME/.openig/config/routes/simple-throttling.json`. On Windows, add the route as `%appdata%\OpenIG\config\routes\simple-throttling.json`.

To Configure a Simple Throttling Filter

1. In IG Studio, create a route:

   a. Browse to http://openig.example.com:8080/openig/studio, and select ⊕ Protect an Application.

   b. Choose ☰ Structured to use the predefined menus and templates.

2. Select Advanced options, and create a route with the following options:

   - Base URI: `http://app.example.com:8081`
   - Condition: Path: `/home/throttle-simple`
   - Name: `00-throttle-simple`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

3. Select Throttling, and then enable throttling.
4. In GROUPING POLICY, select to apply the rate to a single group.

   All requests are grouped together, and the default throttling rate is applied to the group. By default, no more than 100 requests can access the sample application each second.
5. In RATE POLICY, select Fixed, and Edit, and then allow 6 requests each 10 seconds.
6. On the top-right of the screen, select and Display to review the route.

   The following route should be displayed:

```
{
  "name": "00-throttle-simple",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/throttle-simple')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ThrottlingFilter",
          "name": "ThrottlingFilter-1",
          "config": {
            "requestGroupingPolicy": "",
            "rate": {
              "numberOfRequests": 6,
              "duration": "10 s"
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

   Notice the following features of the route:

   - The route matches requests to `/home/throttle-simple`.
   - The ThrottlingFilter contains a request grouping policy that is blank. This means that all requests are in the same group.
   - The rate defines the number of requests allowed to access the sample application in a given time.
7. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Test the Setup

1. With IG and the sample application running, use **curl**, a bash script, or another tool to access the following route in a loop: http://openig.example.com:8080/home/simple-throttle.

   Accessing the route in a loop runs the request multiple times in quick succession, allowing you to test the throttling rate.

```
$ curl -v http://openig.example.com:8080/home/throttle-simple/\[01-10\] \
> /tmp/simple-throttle.txt 2>&1
```

2. Search the output file to see the result:

```
$ grep "< HTTP/1.1" /tmp/simple-throttle.txt | sort | uniq -c

6 < HTTP/1.1 200 OK
4 < HTTP/1.1 429 Too Many Requests
```

   Notice that the first six requests returned a success response, and the following four requests returned an HTTP 429 `Too Many Requests`. This result demonstrates that the throttling filter has allowed only six requests to access the application, and has blocked the other requests.

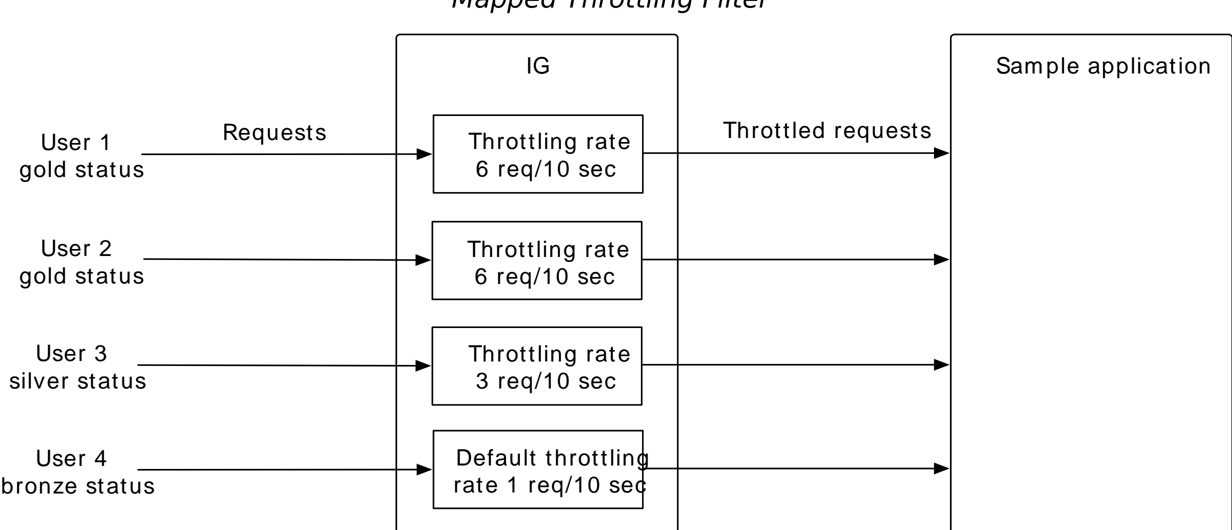
15.2. Configuring a Mapped Throttling Filter

This section describes how to configure a mapped throttling policy, where the grouping policy defines criteria to group requests, and the rate policy defines the criteria by which rates are mapped.

The following image illustrates how different throttling rates can be applied to users.

The following image illustrates how each user with a `gold` status has a throttling rate of 6 requests/10 seconds, and each user with a `silver` status has 3 requests/10 seconds. The `bronze` status is not mapped to a throttling rate, and so a user with the `bronze` status has the default rate.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.



In the following example, the grouping policy and rate policy are taken from fields in the OAuth2Context, as described in "Validating Access-Tokens Through the Token Info Endpoint".

To Set Up AM for Mapped Throttling

1. Follow the steps in "To Set Up AM As an Authorization Server for the Token Info Endpoint".
2. Select Scripts > OAuth2 Access Token Modification Script, and replace the default script as follows:

```
import org.forgerock.http.protocol.Request
import org.forgerock.http.protocol.Response

def attributes = identity.getAttributes(["mail", "employeeNumber"].toSet())
accessToken.setField("mail", attributes["mail"][0])
def mail = attributes['mail'][0]
if (mail.endsWith('@example.com')) {
     status = "gold"
} else if (mail.endsWith('@other.com')) {
     status = "silver"
} else {
     status = "bronze"
}
accessToken.setField("status", status)
```

   The AM script adds user profile information to the access_token, and defines the content of the users `status` field according to the email domain.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Configure a Mapped Throttling Filter

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

1. In IG Studio, create a route:

    a. Browse to http://openig.example.com:8080/openig/studio, and select ⊕ Protect an Application.

    b. Choose Structured to use the predefined menus and templates.

    c. Select Advanced options, and create a route with the following options:

    - Base URI: `http://app.example.com:8081`
    - Condition: Path: `/home/throttle-mapped`
    - Name: `00-throttle-mapped`

2. Configure authorization:

    a. Select Authorization.

    b. Select OAuth 2.0 Resource Server, and enter the following information to reflect the configuration in AM:

    - Token resolver configuration:
        - Access token resolver: `AM token info endpoint`
        - AM service:
            - URI: `http://openam.example.com:8088/openam`
            - Version: The version of the AM instance, for example, `6.5`.
            - Agent: The credentials of the Java agent you created in AM.
                - Username: `ig_agent`
                - Password: `password`
    - Scope configuration:
        - Evaluate scopes: `Statically`
        - Scopes: `mail`, `employeenumber`
    - OAuth 2.0 Authorization settings:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Require HTTPS: Deselect this option
- Enable cache: Deselect this option

Leave all other values as default.

3. Configure throttling:

   a. Select Throttling, and then enable throttling.

   b. Set up the grouping policy:

      i. In GROUPING POLICY, apply the rate to independent groups of requests.

         Requests are split into different groups according to criteria, and the throttling rate is applied to each group.

      ii. Select to group requests by custom criteria.

      iii. Enter `${contexts.oauth2.accessToken.info.mail}` as the custom expression.

         This expression defines the subject in the OAuth2Context.

   c. Set up the rate policy:

      i. In RATE POLICY, select Mapped.

      ii. Select to map requests by custom criteria.

      iii. Enter the custom expression `${contexts.oauth2.accessToken.info.employeenumber}`.

      iv. In Default Rate, select Edit and change default rate to 1 request each 10 seconds.

      v. In Mapped Rates, add the following rate for `gold` status:

         - Match Value: `gold`
         - Number of requests: `6`
         - Period: `10 seconds`

      vi. Add a different rate for `silver` status:

         - Match Value: `silver`
         - Number of requests: `3`
         - Period: `10 seconds`

      vii. Add a different rate for `bronze` status:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Match Value: `bronze`
- Number of requests: `1`
- Period: `10 seconds`

viii. Save the rate policy.

4. Select Chain, and change the order of the filters so that Throttling comes after Authorization.

5. On the top-right of the screen, select and Display to review the route.

   The following route should be displayed:

```
{
  "name": "00-throttle-mapped",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/throttle-mapped')}",
  "heap": [
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "http://openam.example.com:8088/openam",
        "realm": "/",
        "ssoTokenHeader": "iPlanetDirectoryPro",
        "version": "6.5",
        "agent": {
          "username": "ig_agent",
          "password": "password"
        },
        "sessionCache": {
          "enabled": false
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "OAuth2ResourceServerFilter-1",
          "type": "OAuth2ResourceServerFilter",
          "config": {
            "scopes": [
              "mail",
              "employeenumber"
            ],
            "requireHttps": false,
            "realm": "OpenIG",
            "accessTokenResolver": {
              "name": "token-resolver-1",
              "type": "OpenAmAccessTokenResolver",
              "config": {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
                "amService": "AmService-1"
              }
            }
          }
        },
        {
          "name": "ThrottlingFilter-1",
          "type": "ThrottlingFilter",
          "config": {
            "requestGroupingPolicy": "${contexts.oauth2.accessToken.info.mail}",
            "throttlingRatePolicy": {
              "name": "MappedPolicy",
              "type": "MappedThrottlingPolicy",
              "config": {
                "throttlingRateMapper": "${contexts.oauth2.accessToken.info.status}",
                "throttlingRatesMapping": {
                  "gold": {
                    "numberOfRequests": 6,
                    "duration": "10 s"
                  },
                  "silver": {
                    "numberOfRequests": 3,
                    "duration": "10 s"
                  },
                  "bronze": {
                    "numberOfRequests": 1,
                    "duration": "10 s"
                  }
                },
                "defaultRate": {
                  "numberOfRequests": 1,
                  "duration": "10 s"
                }
              }
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

Notice the following features of the route:

- The route matches requests to `/home/throttle-mapped`.
- The OAuth2ResourceServerFilter validates requests with the OpenAmAccessTokenResolver, and makes it available for downstream components in the `oauth2` context.
- The ThrottlingFilter bases the request grouping policy on the AM user's mail. The throttling rate is applied independently to each email address.

  The throttling rate is mapped to the AM user's `status`, which is defined by the email domain, in the AM script.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

6. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

To Test the Setup

1. Get an access_token for George from AM:

```
$ george_token=$(curl -s \
  --user "client-application:password" \
  --data "grant_type=password&username=george&password=C0stanza&scope=mail%20employeenumber" \
  http://openam.example.com:8088/openam/oauth2/access_token | jq -r ".access_token")
```

2. Using the access_token for authentication, access the route multiple times. The following example accesses the route 10 times, and writes the output to a file:

```
$ curl -v http://openig.example.com:8080/home/throttle-mapped/\[01-10\] --header "Authorization:Bearer
 ${george_token}" > /tmp/george.txt 2>&1
```

3. Search the output file to see the result:

```
$ grep "< HTTP/1.1" /tmp/george.txt | sort | uniq -c

6 < HTTP/1.1 200
4 < HTTP/1.1 429
```

   Notice that with a `gold` status, George can access the route 6 times in 10 seconds.

4. In AM, change George's email to `george@other.com`, and then run the last steps again to see how the access is reduced.

15.2.1. Considerations for Dynamic Throttling

The following image illustrates what can happen when the throttling rate defined by `throttlingRateMapping` changes frequently or quickly:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Dynamic Throttling Rate

IG
Sample applicati
Throttled requests
0 sec
2 sec
George gold status
George sliver status
George gold status
Request 1
Request 2
Request 3
Request 4
Request 5
Request 6
Request 7
Request 8
Request 9
Request 10
Request 11
Request 12
Throttling rate 6 req/10 sec
Throttling rate 3 req/10 sec
Throttling rate 6 req/10 sec
X

In the image, George starts out with a `gold` status. In a two second period, he sends five requests, is downgraded to silver, sends four requests, is upgraded back to `gold`, and then sends three more requests.

After making five requests with a `gold` status, George has almost reached his throttling rate. When his status is downgraded to silver, those requests are disregarded and the full throttling rate for `silver` is applied. George can now make three more requests even though he had nearly reached his throttling rate with a `gold` status.

After making three requests with a `silver` status, George has reached his throttling rate. When he makes a fourth request, the request is refused.

George is now upgraded back to `gold` and can now make six more requests even though he had reached his throttling rate with a `silver` status.

When you configure `requestGroupingPolicy` and `throttlingRateMapper`, bear in mind what happens when the throttling rate defined by the `throttlingRateMapper` is changed.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

15.3. Configuring a Scriptable Throttling Filter

This section builds on the example in "Configuring a Mapped Throttling Filter". It creates a scriptable throttling filter, where the script applies a throttling rate of 6 requests/10 seconds to requests from gold status users. For all other requests, the script returns `null`, and applies the default rate of 1 request/10 seconds.

For information about scripting, see "Scripting in Studio". To try this example without using Studio, add the route in the following procedure to the IG configuration as `$HOME/.openig/config/routes/throttle-scriptable.json` (on Windows, `%appdata%\OpenIG\config\routes\throttle-scriptable.json`).

To Configure a Scriptable Throttling Filter

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

1. Set up AM as described in "To Set Up AM for Mapped Throttling".
2. In IG Studio, create a route:
   a. Browse to http://openig.example.com:8080/openig/studio, and select Protect an Application.
   b. Choose Structured to use the predefined menus and templates.
   c. Select Advanced options, and create a route with the following options:
      - Base URI: `http://app.example.com:8081`
      - Condition: Path: `/home/throttle-scriptable`
      - Name: `00-throttle-scriptable`
3. Configure authorization:
   a. Select Authorization.
   b. Select OAuth 2.0 Resource Server, and enter the following information to reflect the configuration in AM:
      - Token resolver configuration:
        - Access token resolver: `AM token info endpoint`
        - AM service:
          - URI: `http://openam.example.com:8088/openam`
          - Version: The version of the AM instance, for example, `6.5`.
          - Agent: The credentials of the Java agent you created in AM.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

        - Username: `ig_agent`
        - Password: `password`
    - Scope configuration:
        - Evaluate scopes: `Statically`
        - Scopes: `mail`, `employeenumber`
    - OAuth 2.0 Authorization settings:
        - Require HTTPS: Deselect this option
        - Enable cache: Deselect this option

    Leave all other values as default.

4. Configure throttling:

    a. Select Throttling, and then enable throttling.

    b. Set up the grouping policy:

        i. In GROUPING POLICY, apply the rate to independent groups of requests.

        Requests are split into different groups according to criteria, and the throttling rate is applied to each group.

        ii. Select to group requests by custom criteria.

        iii. Enter `${contexts.oauth2.accessToken.info.mail}` as the custom expression.

    c. Set up the rate policy:

        i. In RATE POLICY, select Scripted.

        ii. Select to create a new script, and name it `X-User-Status`. So that you can easily identify the script, use a name that describes the content of the script.

        iii. Add the following argument/value pairs:

        - argument: `status`, value: `"gold"`
        - argument: `rate`, value: `6`
        - argument: `duration`, value: `"10 seconds"`

        iv. Replace the default script with the content of a valid Groovy script. For example, enter the following script:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
if (contexts.oauth2.accessToken.info.employeenumber == status) {
  return new ThrottlingRate(rate, duration)
} else {
  return null
}
```

> **Tip**
>
> Alternatively, you can skip the step where you define arguments, and add the following script instead:
>
> ```
> if (contexts.oauth2.accessToken.info.employeenumber == 'gold') {
>   return new ThrottlingRate(6, '10 seconds')
> } else {
>   return null
> }
> ```

> **Note**
>
> Studio does not check the validity of the Groovy script.

v. Enable the default rate, and set it to 1 request each 10 seconds.

vi. Save the rate policy. The script is added to the list of reference scripts available to use in scriptable throttling filters.

5. Select Chain, and change the order of the filters so that Throttling comes after Authorization.

6. On the top-right of the screen, select and Display to review the route.

   The following route should be displayed:

```
{
  "name": "00-throttle-scriptable",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/throttle-scriptable')}",
  "heap": [
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "http://openam.example.com:8088/openam",
        "realm": "/",
        "ssoTokenHeader": "iPlanetDirectoryPro",
        "version": "6.5",
        "agent": {
          "username": "ig_agent",
          "password": "password"
        },
        "sessionCache": {
          "enabled": false
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "OAuth2ResourceServerFilter-1",
          "type": "OAuth2ResourceServerFilter",
          "config": {
            "scopes": [
              "mail",
              "employeenumber"
            ],
            "requireHttps": false,
            "realm": "OpenIG",
            "accessTokenResolver": {
              "name": "token-resolver-1",
              "type": "OpenAmAccessTokenResolver",
              "config": {
                "amService": "AmService-1"
              }
            }
          }
        },
        {
          "name": "ThrottlingFilter-1",
          "type": "ThrottlingFilter",
          "config": {
            "requestGroupingPolicy": "${contexts.oauth2.accessToken.info.mail}",
            "throttlingRatePolicy": {
              "type": "DefaultRateThrottlingPolicy",
              "config": {
                "delegateThrottlingRatePolicy": {
                  "name": "ScriptedPolicy",
                  "type": "ScriptableThrottlingPolicy",
                  "config": {
                    "type": "application/x-groovy",
                    "source": [
                      "if (contexts.oauth2.accessToken.info.status == status) {",
                      "  return new ThrottlingRate(rate, duration)",
                      "} else {",
                      "  return null",
                      "}"
                    ],
                    "args": {
                      "status": "gold",
                      "rate": 6,
                      "duration": "10 seconds"
                    }
                  }
                },
                "defaultRate": {
                  "numberOfRequests": 1,
                  "duration": "10 s"
                }
              }
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
              }
            }
          }
        ],
        "handler": "ReverseProxyHandler"
      }
    }
}
```

Notice the following features of the route:

- The route matches requests to `/home/throttle-scriptable`.
- The DefaultRateThrottlingPolicy delegates the management of throttling to the ScriptableThrottlingPolicy.
- The script applies a throttling rate to requests from users with gold status. For all other requests, the script returns null and the default rate is applied.

7. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

To Test the Setup

1. Get an access_token for George from AM:

```
$ george_token=$(curl -s \
  --user "client-application:password" \
  --data "grant_type=password&username=george&password=C0stanza&scope=mail%20employeenumber" \
  http://openam.example.com:8088/openam/oauth2/access_token | jq -r ".access_token")
```

2. Using the access_token for authentication, access the route multiple times. The following example accesses the route 10 times, and writes the output to a file:

```
$ curl -v http://openig.example.com:8080/home/throttle-scriptable/\[01-10\] --header
 "Authorization:Bearer ${george_token}" > /tmp/george.txt 2>&1
```

3. Search the output file to see the result:

```
$ grep "< HTTP/1.1" /tmp/george.txt | sort | uniq -c

6 < HTTP/1.1 200
4 < HTTP/1.1 429
```

   Notice that with a `gold` status, George can access the route 6 times in 10 seconds.

4. In AM, change George's email to `george@other.com`, and then run the last two steps again to see how the access is reduced.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 16
Configuration Templates

This chapter contains template routes for common configurations.

Before you use these templates, install and configure IG with a router and default route as described in "*First Steps*" in the *Getting Started Guide*.

Next, take one of the templates and then modify it to suit your deployment. Read the summary of each template to find the right match for your application.

When you move to use IG in production, be sure to turn off DEBUG level logging, and to deactivate `CaptureDecorator` use to avoid filling up disk space. Also consider locking down the `Router` configuration.

16.1. Proxy and Capture

If you installed and configured IG with a router and default route as described in "*First Steps*" in the *Getting Started Guide*, then you already proxy and capture the application requests coming in and the server responses going out.

This template route uses a `DispatchHandler` to change the scheme to HTTPS on login:

Proxy and Capture

```
{
  "heap": [
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "comment": "Testing only: blindly trust the server cert for HTTPS.",
      "config": {
        "tls": {
          "type": "TlsOptions",
          "config": {
            "trustManager": {
              "type": "TrustAllManager"
            }
          }
        },
        "hostnameVerifier": "ALLOW_ALL"
      }
    }
  ],
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
  "handler": {
    "type": "DispatchHandler",
    "config": {
      "bindings": [
        {
          "condition": "${request.uri.path == '/login'}",
          "handler": "ReverseProxyHandler",
          "baseURI": "https://app.example.com:8444"
        },
        {
          "condition": "${request.uri.scheme == 'http'}",
          "handler": "ReverseProxyHandler",
          "baseURI": "http://app.example.com:8081"
        },
        {
          "handler": "ReverseProxyHandler",
          "baseURI": "https://app.example.com:8444"
        }
      ]
    }
  },
  "condition": "${matches(request.uri.query, 'demo=capture')}"
}
```

To try this example with the sample application:

1. Save the file as `$HOME/.openig/config/routes/20-capture.json`.

2. Browse to http://openig.example.com:8080/login?demo=capture.

   The login page of the sample application is displayed.

To use this as a default route with a real application:

1. Replace the test ReverseProxyHandler with one that is configured to trust the application's public key server certificate. Otherwise, use a ReverseProxyHandler that references a truststore holding the certificate.

   Configure the ReverseProxyHandler to strictly verifiy hostnames for outgoing SSL connections.

   In production, do not use `TrustAllManager` for TrustManager, or `ALLOW_ALL` for hostname verification. For information, see ReverseProxyHandler(5) in the *Configuration Reference*.

2. Change the baseURI settings to match the target application.

3. Remove the route-level condition on the handler that specifies a `demo` query string parameter.

16.2. Simple Login Form

This template route intercepts the login page request, replaces it with a login form, and logs the user into the target application with hard-coded username and password:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Simple Login Form

```
{
  "heap": [
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "comment": "Testing only: blindly trust the server cert for HTTPS.",
      "config": {
        "tls": {
          "type": "TlsOptions",
          "config": {
            "trustManager": {
              "type": "TrustAllManager"
            }
          }
        },
        "hostnameVerifier": "ALLOW_ALL"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "PasswordReplayFilter",
          "config": {
            "loginPage": "${request.uri.path == '/login'}",
            "request": {
              "method": "POST",
              "uri": "https://app.example.com:8444/login",
              "form": {
                "username": [
                  "MY_USERNAME"
                ],
                "password": [
                  "MY_PASSWORD"
                ]
              }
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  },
  "condition": "${matches(request.uri.query, 'demo=simple')}"
}
```

To try this example with the sample application:

1. Save the file as `$HOME/.openig/config/routes/21-simple.json`.
2. Replace `MY_USERNAME` with `demo`, and `MY_PASSWORD` with `Ch4ng31t`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

3. Browse to http://openig.example.com:8080/login?demo=simple.

   The sample application profile page for the demo user displays the following information about the request:

```
Method POST
URI /login
Cookies
Headers content-type: application/x-www-form-urlencoded
content-length: 31
host: app.example.com:8444
connection: Keep-Alive
user-agent: Apache-HttpAsyncClient/4.1.2 (Java/1.8.0_144)
```

To use this as a default route with a real application:

1. Replace the test ReverseProxyHandler with one that is configured to trust the application's public key server certificate. Otherwise, use a ReverseProxyHandler that references a truststore holding the certificate.

   Configure the ReverseProxyHandler to strictly verifiy hostnames for outgoing SSL connections.

   In production, do not use `TrustAllManager` for TrustManager, or `ALLOW_ALL` for hostname verification. For information, see ReverseProxyHandler(5) in the *Configuration Reference*.

2. Change the `uri`, `form`, and `baseURI` to match the target application.

3. Remove the route-level condition on the handler that specifies a `demo` query string parameter.

16.3. Login Form With Cookie From Login Page

Like the previous route, this template route intercepts the login page request, replaces it with the login form, and logs the user into the target application with hard-coded username and password. This route also adds a CookieFilter to manage cookies.

The route uses a default `CookieFilter` to manage cookies. In this default configuration, cookies from the protected application are intercepted and stored in the IG session. They are not sent to the browser. For information, see CookieFilter(5) in the *Configuration Reference*.

Login Form With Cookie From Login Page

```
{
  "heap": [
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "comment": "Testing only: blindly trust the server cert for HTTPS.",
      "config": {
        "tls": {
          "type": "TlsOptions",
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
          "config": {
            "trustManager": {
              "type": "TrustAllManager"
            }
          }
        },
        "hostnameVerifier": "ALLOW_ALL"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "PasswordReplayFilter",
          "config": {
            "loginPage": "${request.uri.path == '/login'}",
            "request": {
              "method": "POST",
              "uri": "https://app.example.com:8444/login",
              "form": {
                "username": [
                  "MY_USERNAME"
                ],
                "password": [
                  "MY_PASSWORD"
                ]
              }
            }
          }
        },
        {
          "type": "CookieFilter"
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  },
  "condition": "${matches(request.uri.query, 'demo=cookie')}"
}
```

To try this example with the sample application:

1. Save the file as `$HOME/.openig/config/routes/22-cookie.json`.
2. Replace `MY_USERNAME` with `kramer`, and `MY_PASSWORD` with `N3wman12`.
3. Browse to http://openig.example.com:8080/login?demo=cookie.

   The sample application page is displayed.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
Method    POST
URI    /login
Cookies
Headers  content-type: application/x-www-form-urlencoded
         content-length: 31
         host: app.example.com:8444
         connection: Keep-Alive
         user-agent: Apache-HttpAsyncClient/... (Java/...)
```

4. Refresh your connection to http://openig.example.com:8080/login?demo=cookie.

   Compared to the example in "Login Form With Cookie From Login Page", this example displays additional information about the session cookie:

   ```
   Cookies  session-cookie=123...
   ```

To use this as a default route with a real application:

1. Replace the test ReverseProxyHandler with one that is configured to trust the application's public key server certificate. Otherwise, use a ReverseProxyHandler that references a truststore holding the certificate.

   Configure the ReverseProxyHandler to strictly verifiy hostnames for outgoing SSL connections.

   In production, do not use `TrustAllManager` for TrustManager, or `ALLOW_ALL` for hostname verification. For information, see ReverseProxyHandler(5) in the *Configuration Reference*.

2. Change the `uri` and `form` to match the target application.

3. Remove the route-level condition on the handler that specifies a `demo` query string parameter.

16.4. Login Form With Password Replay and Cookie Filters

When a user without a valid session tries to access a protected application, this template route works with an application to return a login page.

The route uses a PasswordReplayFilter to find the login page by using a pattern that matches a mock AM Classic UI page.

The CookieFilter removes cookies from the response, except for the session cookie added by the container.

The route uses a default `CookieFilter` to manage cookies. In this default configuration, cookies from the protected application are intercepted and stored in the IG session. They are not sent to the browser. For information, see CookieFilter(5) in the *Configuration Reference*.

Login Form With Password Replay and Cookie Filters

```
{
  "handler": {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "PasswordReplayFilter",
          "config": {
            "loginPageContentMarker": "OpenAM\\s\\(Login\\)",
            "request": {
              "comments": [
                "An example based on OpenAM classic UI: ",
                "uri is for the OpenAM login page; ",
                "IDToken1 is the username field; ",
                "IDToken2 is the password field; ",
                "host takes the OpenAM FQDN:port.",
                "The sample app simulates OpenAM."
              ],
              "method": "POST",
              "uri": "http://app.example.com:8081/openam/UI/Login",
              "form": {
                "IDToken0": [
                  ""
                ],
                "IDToken1": [
                  "demo"
                ],
                "IDToken2": [
                  "Ch4ng31t"
                ],
                "IDButton": [
                  "Log+In"
                ],
                "encoded": [
                  "false"
                ]
              },
              "headers": {
                "host": [
                  "app.example.com:8081"
                ]
              }
            }
          }
        },
        {
          "type": "CookieFilter"
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  },
  "condition": "${matches(request.uri.query, 'demo=classic')}"
}
```

To try this example with the sample application:

1. Save the file as `$HOME/.openig/config/routes/23-classic.json`.
2. Use the following **curl** command to check that it works:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ curl -D- http://openig.example.com:8080/login?demo=classic

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: IG_SESSIONID=24446BA29E866F840197C8E0EAD57A89; Path=/;
 HttpOnly
...
```

To use this as a default route with a real application:

1. Change the `uri` and `form` to match the target application.
2. Remove the route-level condition on the handler that specifies a `demo` query string parameter.

16.5. Login Which Requires a Hidden Value From the Login Page

This template route extracts a hidden value from the login page, and includes it the static login form that it then POSTs to the target application.

Login Which Requires a Hidden Value From the Login Page

```
{
  "properties": {
    "appBaseUri":  "https://app.example.com:8444"
  },
  "heap": [
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "comment": "Testing only: blindly trust the server cert for HTTPS.",
      "config": {
        "tls": {
          "type": "TlsOptions",
          "config": {
            "trustManager": {
              "type": "TrustAllManager"
            }
          }
        },
        "hostnameVerifier": "ALLOW_ALL"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "PasswordReplayFilter",
          "config": {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
            "loginPage": "${request.uri.path == '/login'}",
            "loginPageExtractions": [
              {
                "name": "hidden",
                "pattern": "loginToken\\s+value=\"(.*)\""
              }
            ],
            "request": {
              "method": "POST",
              "uri": "${appBaseUri}/login",
              "form": {
                "username": [
                  "MY_USERNAME"
                ],
                "password": [
                  "MY_PASSWORD"
                ],
                "hiddenValue": [
                  "${attributes.extracted.hidden}"
                ]
              }
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  },
  "condition": "${matches(request.uri.query, 'demo=hidden')}",
  "baseURI": "${appBaseUri}"
}
```

The parameters in the PasswordReplayFilter form, `MY_USERNAME` and `MY_PASSWORD`, can have string values or can use expressions.

To try this example with the sample application:

1. Save the file as `$HOME/.openig/config/routes/24-hidden.json`.
2. Replace `MY_USERNAME` with `scarter`, and `MY_PASSWORD` with `S9rain12`.
3. Browse to http://openig.example.com:8080/login?demo=hidden.

To use this as a default route with a real application:

1. Replace the test ReverseProxyHandler with one that is configured to trust the application's public key server certificate. Otherwise, use a ReverseProxyHandler that references a truststore holding the certificate.

   Configure the ReverseProxyHandler to strictly verifiy hostnames for outgoing SSL connections.

   In production, do not use `TrustAllManager` for TrustManager, or `ALLOW_ALL` for hostname verification. For information, see ReverseProxyHandler(5) in the *Configuration Reference*.
2. Change the `loginPage`, `loginPageExtractions`, `uri`, and `form` to match the target application.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

3. Remove the route-level condition on the handler that specifies a `demo` query string parameter.

16.6. HTTP and HTTPS Application

This template route proxies traffic to an application with both HTTP and HTTPS ports. The application uses HTTPS for authentication and HTTP for the general application features. Assuming that all login requests are made over HTTPS, you must add the login filters and handlers to the chain.

HTTP and HTTPS Application

```
{
  "heap": [
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "comment": "Testing only: blindly trust the server cert for HTTPS.",
      "config": {
        "tls": {
          "type": "TlsOptions",
          "config": {
            "trustManager": {
              "type": "TrustAllManager"
            }
          }
        },
        "hostnameVerifier": "ALLOW_ALL"
      }
    }
  ],
  "handler": {
    "type": "DispatchHandler",
    "config": {
      "bindings": [
        {
          "condition": "${request.uri.scheme == 'http'}",
          "handler": "ReverseProxyHandler",
          "baseURI": "http://app.example.com:8081"
        },
        {
          "condition": "${request.uri.path == '/login'}",
          "handler": {
            "type": "Chain",
            "config": {
              "comment": "Add one or more filters to handle login.",
              "filters": [],
              "handler": "ReverseProxyHandler"
            }
          },
          "baseURI": "https://app.example.com:8444"
        },
        {
          "handler": "ReverseProxyHandler",
          "baseURI": "https://app.example.com:8444"
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
        }
      ]
    }
  },
  "condition": "${matches(request.uri.query, 'demo=https')}"
}
```

To try this example with the sample application:

1. Save the file as `$HOME/.openig/config/routes/25-https.json`.
2. Browse to http://openig.example.com:8080/login?demo=https.

   The login page of the sample application is displayed.

To use this as a default route with a real application:

1. Replace the test ReverseProxyHandler with one that is configured to trust the application's public key server certificate. Otherwise, use a ReverseProxyHandler that references a truststore holding the certificate.

   Configure the ReverseProxyHandler to strictly verifiy hostnames for outgoing SSL connections.

   In production, do not use `TrustAllManager` for TrustManager, or `ALLOW_ALL` for hostname verification. For information, see ReverseProxyHandler(5) in the *Configuration Reference*.
2. Change the `loginPage`, `loginPageExtractions`, `uri`, and `form` to match the target application.
3. Remove the route-level condition on the handler that specifies a `demo` query string parameter.

16.7. AM Integration With Headers

This template route logs the user into the target application by using headers such as those passed in from an AM policy agent. If the passed in header contains only a user name or subject and requires a lookup to an external data source, you must add an attribute filter to the chain to retrieve the credentials.

AM Integration With Headers

```
{
  "heap": [
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "comment": "Testing only: blindly trust the server cert for HTTPS.",
      "config": {
        "tls": {
          "type": "TlsOptions",
          "config": {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
            "trustManager": {
              "type": "TrustAllManager"
            }
          }
        },
        "hostnameVerifier": "ALLOW_ALL"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "PasswordReplayFilter",
          "config": {
            "loginPage": "${request.uri.path == '/login'}",
            "request": {
              "method": "POST",
              "uri": "https://app.example.com:8444/login",
              "form": {
                "username": [
                  "${request.headers['username'][0]}"
                ],
                "password": [
                  "${request.headers['password'][0]}"
                ]
              }
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  },
  "condition": "${matches(request.uri.query, 'demo=headers')}"
}
```

To try this example with the sample application:

1. Save the file as `$HOME/.openig/config/routes/26-headers.json`.
2. Use the **curl** command to simulate the headers being passed in from an AM policy agent, as in the following example:

```
$ curl \
--header "username: kvaughan"
 \
--header "password: B5ibery12" \
http://openig.example.com:8080/login?demo=headers

...
<title id="welcome">Howdy, kvaughan</
title>
...
```

To use this as a default route with a real application:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

1. Replace the test ReverseProxyHandler with one that is configured to trust the application's public key server certificate. Otherwise, use a ReverseProxyHandler that references a truststore holding the certificate.

   Configure the ReverseProxyHandler to strictly verifiy hostnames for outgoing SSL connections.

   In production, do not use `TrustAllManager` for TrustManager, or `ALLOW_ALL` for hostname verification. For information, see ReverseProxyHandler(5) in the *Configuration Reference*.

2. Change the `loginPage`, `uri`, and `form` to match the target application.

3. Remove the route-level condition on the handler that specifies a `demo` query string parameter.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 17
Extending IG

This chapter describes how to extend IG, taking you through the steps to:

- Write scripts to create custom filters and handlers
- Plug additional Java libraries into IG for further customization

For when you can't achieve complex server interactions or intensive data transformations with scripts or existing handlers, filters, or expressions, IG allows you to develop custom extensions in Java and provide them in additional libraries that you build into IG. The libraries allow you to develop custom extensions to IG.

Important

When you are writing scripts or Java extensions, never use a `Promise` blocking method, such as `get()`, `getOrThrow()`, or `getOrThrowUninterruptibly()`, to obtain the response.

A promise represents the result of an asynchronous operation. Therefore, using a blocking method to wait for the result can cause deadlocks and/or race issues.

17.1. About Scripting

IG supports the Groovy dynamic scripting language through the use the scriptable objects. For information about scriptable object types, their configuration, and properties, see Scripts in the *Configuration Reference*.

Scriptable objects are configured by the script's Internet media type, and either a source script included in the JSON configuration, or a file script that IG reads from a file. The configuration can optionally supply arguments to the script.

IG provides global variables to scripts at runtime, and provides access to Groovy's built-in functionality. Scripts can access the request and the context, store variables across executions, write messages to logs, make requests to a web service or to an LDAP directory service, and access responses returned in promise callback methods.

Before trying the scripts in this chapter, install and configure IG as described in "*First Steps*" in the *Getting Started Guide*.

When developing and debugging your scripts, consider configuring a capture decorator to log requests, responses, and context data in JSON form. You can then turn off capturing when you move to production. For details, see CaptureDecorator(5) in the *Configuration Reference*.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

17.1.1. Using a Reference File Script

The following example defines a ScriptableFilter written in Groovy, and stored in a file named `$HOME/.openig/scripts/groovy/SimpleFormLogin.groovy` (`%appdata%\OpenIG\scripts\groovy\SimpleFormLogin.groovy` on Windows):

```
{
  "name": "SimpleFormLogin",
  "type": "ScriptableFilter",
  "config": {
    "type": "application/x-groovy",
    "file": "SimpleFormLogin.groovy"
  }
}
```

Relative paths in the file field depend on how IG is installed. If IG is installed in an application server, then paths for Groovy scripts are relative to `$HOME/.openig/scripts/groovy`.

The base location `$HOME/.openig/scripts/groovy` is on the classpath when the scripts are executed. If some Groovy scripts are not in the default package, but instead have their own package names, they belong in the directory corresponding to their package name. For example, a script in package `com.example.groovy` belongs under `$HOME/.openig/scripts/groovy/com/example/groovy/`.

17.1.2. Scripting in Studio

You can use Studio to configure a ScriptableFilter or ScriptableThrottlingPolicy, or use scripts to configure scopes in OAuth2ResourceServerFilter.

During configuration, you can enter the script directly into the object, or you can use a stored reference script. Note the following points about creating and using reference scripts:

- When you enter a script directly into an object, the script is added to the list of reference scripts.
- You can use a reference script in multiple objects in a route, but if you edit a reference script, all objects that use it are updated with the change.
- If you delete an object that uses a script, or remove the object from the chain, the script that it references remains in the list of scripts.
- If a reference script is used in an object, you can't rename or delete the script.

For an example of creating a ScriptableThrottlingPolicy in Studio, see " Configuring a Scriptable Throttling Filter ". For information about using Studio, see "Adding Filters to a Route" in the *Getting Started Guide*.

17.2. Scripting Dispatch

In order to route requests, especially when the conditions are complicated, you can use a `ScriptableHandler` instead of a `DispatchHandler` as described in DispatchHandler(5) in the *Configuration Reference*.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

The following script demonstrates a simple dispatch handler:

```
/*
 * Copyright 2014-2020 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */


/*
 * This simplistic dispatcher matches the path part of the HTTP request.
 * If the path is /mylogin, it checks Username and Password headers,
 * accepting bjensen:H1falutin, and returning HTTP 403 Forbidden to others.
 * Otherwise it returns HTTP 401 Unauthorized.
 */

// Rather than return a Promise of a response from an external source,
// this script returns the response itself.
response = new Response(Status.OK);

switch (request.uri.path) {

    case "/mylogin":

        if (request.headers.Username.values[0] == "bjensen" &&
                request.headers.Password.values[0] == "H1falutin") {

            response.status = Status.OK
            response.entity = "<html><p>Welcome back, Babs!</p></html>"

        } else {

            response.status = Status.FORBIDDEN
            response.entity = "<html><p>Authorization required</p></html>"

        }

        break

    default:

        response.status = Status.UNAUTHORIZED
        response.entity = "<html><p>Please <a href='./mylogin'>log in</a>.</p></html>"

        break

}

// Return the locally created response, no need to wrap it into a Promise
return response
```

To try this handler, save the script as `$HOME/.openig/scripts/groovy/DispatchHandler.groovy` (`%appdata%\OpenIG\scripts\groovy\DispatchHandler.groovy` on Windows).

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Next, add the following route to your configuration as `$HOME/.openig/config/routes/98-dispatch.json` (`%appdata%\OpenIG\config\routes\98-dispatch.json` on Windows):

```
{
  "heap": [
    {
      "name": "DispatchHandler",
      "type": "DispatchHandler",
      "config": {
        "bindings": [{
          "condition": "${matches(request.uri.path, '/mylogin')}",
          "handler": {
            "type": "Chain",
            "config": {
              "filters": [
                {
                  "type": "HeaderFilter",
                  "config": {
                    "messageType": "REQUEST",
                    "add": {
                      "Username": [
                        "bjensen"
                      ],
                      "Password": [
                        "H1falutin"
                      ]
                    }
                  }
                }
              ],
              "handler": "Dispatcher"
            }
          }
        },
          {
            "handler": "Dispatcher",
            "condition": "${matches(request.uri.path, '/dispatch')}"
          }
        ]
      }
    },
    {
      "name": "Dispatcher",
      "type": "ScriptableHandler",
      "config": {
        "type": "application/x-groovy",
        "file": "DispatchHandler.groovy"
      }
    }
  ],
  "handler": "DispatchHandler",
  "condition": "${matches(request.uri.path, '^/dispatch') or matches(request.uri.path, '^/mylogin')}"
}
```

The route sets up the headers required by the script when the user logs in.

To try it out, browse to http://openig.example.com:8080/dispatch.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

The response from the script says, "Please log in." When you click the log in link, the `HeaderFilter` sets `Username` and `Password` headers in the request, and passes the request to the script.

The script then responds, `Welcome back, Babs!`

17.3. Scripting HTTP Basic Authentication

HTTP Basic authentication calls for the user agent such as a browser to send a user name and password to the server in an `Authorization` header. HTTP Basic authentication relies on an encrypted connection to protect the user name and password credentials, which are base64-encoded in the `Authorization` header, not encrypted.

The following script, for use in a `ScriptableFilter`, adds an `Authorization` header based on a username and password combination:

```
/*
 * Copyright 2014-2020 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */

/*
 * Perform basic authentication with the user name and password
 * that are supplied using a configuration like the following:
 *
 * {
 *     "name": "BasicAuth",
 *     "type": "ScriptableFilter",
 *     "config": {
 *         "type": "application/x-groovy",
 *         "file": "BasicAuthFilter.groovy",
 *         "args": {
 *             "username": "bjensen",
 *             "password": "H1falutin"
 *             }
 *         }
 * }
 */

def userPass = username + ":" + password
def base64UserPass = userPass.getBytes().encodeBase64()
request.headers.add("Authorization", "Basic ${base64UserPass}" as String)

// Credentials are only base64-encoded, not encrypted: Set scheme to HTTPS.

/*
 * When connecting over HTTPS, by default the client tries to trust the server.
 * If the server has no certificate
 * or has a self-signed certificate unknown to the client,
 * then the most likely result is an SSLPeerUnverifiedException.
 *
 * To avoid an SSLPeerUnverifiedException,
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
 * set up HTTPS correctly on the server.
 * Either use a server certificate signed by a well-known CA,
 * or set up the gateway to trust the server certificate.
 */
request.uri.scheme = "https"

// Calls the next Handler and returns a Promise of the Response.
// The Response can be handled with asynchronous Promise callbacks.
next.handle(context, request)
```

To try this filter, save the script as `$HOME/.openig/scripts/groovy/BasicAuthFilter.groovy` (`%appdata%\OpenIG\scripts\groovy\BasicAuthFilter.groovy` on Windows).

Next, add the following route to your configuration as `$HOME/.openig/config/routes/09-basic.json` (`%appdata%\OpenIG\config\routes\09-basic.json` on Windows):

```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ScriptableFilter",
          "config": {
            "type": "application/x-groovy",
            "file": "BasicAuthFilter.groovy",
            "args": {
              "username": "bjensen",
              "password": "H1falutin"
            }
          },
          "capture": "filtered_request"
        }
      ],
      "handler": {
        "type": "StaticResponseHandler",
        "config": {
          "status": 200,
          "reason": "OK",
          "entity": "Hello, Babs!"
        }
      }
    }
  },
  "condition": "${matches(request.uri.path, '^/basic')}"
}
```

When the request path matches `/basic` the route calls the `Chain`, which runs the `ScriptableFilter`. The capture setting captures the request as updated by the `ScriptableFilter`. Finally, IG returns a static page.

To try it out, browse to http://openig.example.com:8080/basic.

The captured request in the console log shows that the scheme is now HTTPS, and that the `Authorization` header is set for HTTP Basic:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
GET https://openig.example.com:8080/basic HTTP/1.1
Authorization: Basic YmplbnNlbjpoaWZhbHV0aW4=
```

17.4. Scripting Authentication to LDAP and LDAPS-enabled Servers

Many organizations use an LDAP directory service to store user profiles including authentication credentials. The LDAP directory service securely stores user passwords in a highly-available, central service capable of handling thousands of authentications per second.

17.4.1. Scripting Authentication to an LDAP Server

The following script, for use in a `ScriptableFilter`, performs simple authentication against an LDAP server, based on request form fields `username` and `password`:

```
/*
 * Copyright 2014-2017 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */

import org.forgerock.opendj.ldap.*

/*
 * Perform LDAP authentication based on user credentials from a form.
 *
 * If LDAP authentication succeeds, then return a promise to handle the response.
 * If there is a failure, produce an error response and return it.
 */

username = request.form?.username[0]
password = request.form?.password[0]

// For testing purposes, the LDAP host and port are provided in the context's attributes.
// Edit as needed to match your directory service.
host = attributes.ldapHost ?: "localhost"
port = attributes.ldapPort ?: 1389

client = ldap.connect(host, port as Integer)
try {

    // Assume the username is an exact match of either
    // the user ID, the email address, or the user's full name.
    filter = "(|(uid=%s)(mail=%s)(cn=%s))"

    user = client.searchSingleEntry(
            "ou=people,dc=example,dc=com",
            ldap.scope.sub,
            ldap.filter(filter, username, username, username))
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    client.bind(user.name as String, password?.toCharArray())

    // Authentication succeeded.

    // Set a header (or whatever else you want to do here).
    request.headers.add("Ldap-User-Dn", user.name.toString())

    // Most LDAP attributes are multi-valued.
    // When you read multi-valued attributes, use the parse() method,
    // with an AttributeParser method
    // that specifies the type of object to return.
    attributes.cn = user.cn?.parse().asSetOfString()

    // When you write attribute values, set them directly.
    user.description = "New description set by my script"

    // Here is how you might read a single value of a multi-valued attribute:
    attributes.description = user.description?.parse().asString()

    // Call the next handler. This returns when the request has been handled.
    return next.handle(context, request)

} catch (AuthenticationException e) {

    // LDAP authentication failed, so fail the response with
    // HTTP status code 403 Forbidden.

    response = new Response(Status.FORBIDDEN)
    response.entity = "<html><p>Authentication failed: " + e.message + "</p></html>"

} catch (Exception e) {

    // Something other than authentication failed on the server side,
    // so fail the response with HTTP 500 Internal Server Error.

    response = new Response(Status.INTERNAL_SERVER_ERROR)
    response.entity = "<html><p>Server error: " + e.message + "</p></html>"

} finally {
    client.close()
}

// Return the locally created response, no need to wrap it into a Promise
return response
```

For the list of methods to specify which type of objects to return, see the OpenDJ LDAP SDK Javadoc for `AttributeParser`.

To Authenticate to an LDAP Server

1. Install an LDAP directory server such as ForgeRock Directory Server, and generate or import some sample users who can authenticate over LDAP.

   For information about setting up DS and importing sample data, see To Set Up a Directory Server in the DS *Install Guide*.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

2. Save the previous script as `$HOME/.openig/scripts/groovy/LdapAuthFilter.groovy` (`%appdata%\OpenIG\scripts\groovy\LdapAuthFilter.groovy` on Windows).

   If your directory server installation does not match the values in the script, adjust the script to match your installation.

3. Add the following route to your configuration as `$HOME/.openig/config/routes/10-ldap.json`, or `%appdata%\OpenIG\config\routes\10-ldap.json` on Windows:

```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ScriptableFilter",
          "config": {
            "type": "application/x-groovy",
            "file": "LdapAuthFilter.groovy"
          }
        }
      ],
      "handler": {
        "type": "ScriptableHandler",
        "config": {
          "type": "application/x-groovy",
          "source": [
            "dn = request.headers['Ldap-User-Dn'].values[0]",
            "entity = '<html><body><p>Ldap-User-Dn: ' + dn + '</p></body></html>'",
            "",
            "response = new Response(Status.OK)",
            "response.entity = entity",
            "return response"
          ]
        }
      }
    }
  },
  "condition": "${matches(request.uri.path, '^/ldap')}"
}
```

   The route calls the `LdapAuthFilter.groovy` script to authenticate the user over LDAP. On successful authentication, it responds with the the bind DN.

4. Browse to a URL where query string parameters specify a valid username and password.

   If you set up DS and imported the DS sample data as described in step 1, browse to the following link which specifies credentials for the sample user `abarnes`: http://openig.example.com:8080/ldap?username=abarnes&password=chevron.

   The script responds with the DN of the user:

```
Ldap-User-Dn: uid=abarnes,ou=People,dc=example,dc=com
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

17.4.2. Scripting Authentication to an LDAPS-enabled Server

To authenticate to LDAPS, you must define the configuration for an SSL context builder, specify the enabled protocols, define a trust manager, and use the LDAPS port.

For convenience, this example uses a special TrustManager to blindly accept all certificates. In a production environment, use a TrustManager that is strictly configured to accept only appropriate certificates.

To test authentication to an LDAPS enabled server, follow the procedure in "Scripting Authentication to an LDAP Server", but save the following script as `LdapAuthFilter.groovy` instead of the script in that procedure:

```
import org.forgerock.opendj.ldap.*
import org.forgerock.opendj.security.SslOptions;
import org.forgerock.opendj.security.TrustManagers;

/* Perform LDAP authentication based on user credentials from a form,
 * connecting to an LDAPS enabled server.
 *
 * If LDAP authentication succeeds, then return a promise to handle the response.
 * If there is a failure, produce an error response and return it.
 */

username = request.form?.username[0]
password = request.form?.password[0]

// Update port number to match the LDAPS port of your directory service.
host = attributes.ldapHost ?: "localhost"
port = attributes.ldapPort ?: 1636

// Include options for SSL. In this example, the LDAP secure protocol is TLSv1.2,
// and the TrustAllManager blindly trusts all server certificates.
// In a production environment, replace TrustAllManager with a properly configured TrustManager.
ldapOptions = ldap.defaultOptions(context)
SslOptions sslOptions = SslOptions.newSslOptions(null, TrustManagers.trustAll())
                                  .enabledProtocols("TLSv1.2");
ldapOptions = ldapOptions.set(CommonLdapOptions.SSL_OPTIONS, sslOptions);

// Include SSL options in the LDAP connection
client = ldap.connect(host, port as Integer, ldapOptions)
try {

    // Assume the username is an exact match of either
    // the user ID, the email address, or the user's full name.
    filter = "(|(uid=%s)(mail=%s)(cn=%s))"

    user = client.searchSingleEntry(
            "ou=people,dc=example,dc=com",
            ldap.scope.sub,
            ldap.filter(filter, username, username, username))

    client.bind(user.name as String, password?.toCharArray())

    // Authentication succeeded.

    // Set a header (or whatever else you want to do here).
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    request.headers.add("Ldap-User-Dn", user.name.toString())

    // Most LDAP attributes are multi-valued.
    // When you read multi-valued attributes, use the parse() method,
    // with an AttributeParser method
    // that specifies the type of object to return.
    attributes.cn = user.cn?.parse().asSetOfString()

    // When you write attribute values, set them directly.
    user.description = "New description set by my script"

    // Here is how you might read a single value of a multi-valued attribute:
    attributes.description = user.description?.parse().asString()

    // Call the next handler. This returns when the request has been handled.
    return next.handle(context, request)

} catch (AuthenticationException e) {
    // LDAP authentication failed, so fail the response with
    // HTTP status code 403 Forbidden.
    response = new Response(Status.FORBIDDEN)
    response.headers['Content-Type'] = "text/html; charset=utf-8"
    response.entity = "<html><p>Authentication failed: " + e.message + "</p></html>"

} catch (Exception e) {
    // Something other than authentication failed on the server side,
    // so fail the response with HTTP 500 Internal Server Error.
    response = new Response(Status.INTERNAL_SERVER_ERROR)
    response.headers['Content-Type'] = "text/html; charset=utf-8"
    response.entity = "<html><p>Server error: " + e.message + "</p></html>"

} finally {
    client.close()
}

// Return the locally created response, no need to wrap it into a Promise
return response
```

17.5. Scripting SQL Queries

You can use a `ScriptableFilter` to look up information in a relational database and include the results in the request context.

The following filter looks up user credentials in a database given the user's email address, which is found in the form data of the request. The script then sets the credentials in headers, making sure the scheme is HTTPS to protect the request when it leaves IG:

```
/*
 * Copyright 2014-2017 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
/*
 * Look up user credentials in a relational database
 * based on the user's email address provided in the request form data,
 * and set the credentials in the request headers for the next handler.
 */

def client = new SqlClient()
def credentials = client.getCredentials(request.form?.mail[0])
request.headers.add("Username", credentials.Username)
request.headers.add("Password", credentials.Password)

// The credentials are not protected in the headers, so use HTTPS.
request.uri.scheme = "https"

// Calls the next Handler and returns a Promise of the Response.
// The Response can be handled with asynchronous Promise callbacks.
next.handle(context, request)
```

The previous script demonstrates a `ScriptableFilter` that uses a `SqlClient` class defined in another script. The following code listing shows the `SqlClient` class:

```
/*
 * Copyright 2014-2017 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */

import groovy.sql.Sql

import javax.naming.InitialContext
import javax.sql.DataSource

/**
 * Access a database with a well-known structure,
 * in particular to get credentials given an email address.
 */
class SqlClient {

    // Get a DataSource from the container.
    InitialContext context = new InitialContext()
    DataSource dataSource = context.lookup("jdbc/forgerock") as DataSource
    def sql = new Sql(dataSource)

    // The expected table is laid out like the following.

    // Table USERS
    // ----------------------------------------
    // | USERNAME  | PASSWORD |   EMAIL   |...|
    // ----------------------------------------
    // | <username>| <passwd> | <mail@...>|...|
    // ----------------------------------------

    String tableName = "USERS"
    String usernameColumn = "USERNAME"
    String passwordColumn = "PASSWORD"
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    String mailColumn = "EMAIL"

    /**
     * Get the Username and Password given an email address.
     *
     * @param mail Email address used to look up the credentials
     * @return Username and Password from the database
     */
    def getCredentials(mail) {
        def credentials = [:]
        def query = "SELECT " + usernameColumn + ", " + passwordColumn +
                " FROM " + tableName + " WHERE " + mailColumn + "='$mail';"

        sql.eachRow(query) {
            credentials.put("Username", it."$usernameColumn")
            credentials.put("Password", it."$passwordColumn")
        }
        return credentials
    }
}
```

To try the script, follow these steps:

1. Follow the tutorial in "Log in With Credentials From a Database".

   When everything in that tutorial works, you know that IG can connect to the database, look up users by email address, and successfully authenticate to the sample application.

2. Save the scripts as `$HOME/.openig/scripts/groovy/SqlAccessFilter.groovy` (`%appdata%\OpenIG\scripts\groovy\SqlAccessFilter.groovy` on Windows), and as `$HOME/.openig/scripts/groovy/SqlClient.groovy` (`%appdata%\OpenIG\scripts\groovy\SqlClient.groovy` on Windows).

3. Add the following route to your configuration as `$HOME/.openig/config/routes/11-db.json` (`%appdata%\OpenIG\config\routes\11-db.json` on Windows):

```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ScriptableFilter",
          "config": {
            "type": "application/x-groovy",
            "file": "SqlAccessFilter.groovy"
          }
        },
        {
          "type": "StaticRequestFilter",
          "config": {
            "method": "POST",
            "uri": "http://app.example.com:8081/login",
            "form": {
              "username": [
                "${request.headers['Username'][0]}"
              ],
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
              "password": [
                "${request.headers['Password'][0]}"
              ]
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  },
  "condition": "${matches(request.uri.path, '^/db')}"
}
```

The route calls the `ScriptableFilter` to look up credentials over SQL. It then uses calls a `StaticRequestFilter` to build a login request. Although the script sets the scheme to HTTPS, the `StaticRequestFilter` ignores that and resets the URI. This makes it easier to try the script without additional steps to set up HTTPS.

To try the configuration, browse to a URL where a query string parameter specifies a valid email address, such as http://openig.example.com:8080/db?mail=george@example.com.

If the lookup and authentication are successful, you see the profile page of the sample application.

17.6. Developing Custom Extensions

IG includes a complete Java application programming interface to allow you to customize IG to perform complex server interactions or intensive data transformations that you cannot achieve with scripts or the existing handlers, filters, and expressions described in Expressions(5) in the *Configuration Reference*.

17.6.1. Key Extension Points

Interface Stability: Evolving (For information, see "ForgeRock Product Interface Stability" in the *Release Notes*.)

The following interfaces are available:

Decorator

A `Decorator` adds new behavior to another object without changing the base type of the object.

When suggesting custom `Decorator` names, know that IG reserves all field names that use only alphanumeric characters. To avoid clashes, use dots or dashes in your field names, such as `my-decorator`.

ExpressionPlugin

An `ExpressionPlugin` adds a node to the `Expression` context tree, alongside `env` (for environment variables), and `system` (for system properties). For example, the expression `${system['user.home']}` yields the home directory of the user running the application server for IG.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

In your `ExpressionPlugin`, the `getKey()` method returns the name of the node, and the `getObject()` method returns the unified expression language context object that contains the values needed to resolve the expression. The plugins for `env` and `system` return Map objects, for example.

When you add your own `ExpressionPlugin`, you must make it discoverable within your custom library. You do this by adding a services file named after the plugin interface, where the file contains the fully qualified class name of your plugin, under `META-INF/services/org.forgerock.openig.el.ExpressionPlugin` in the .jar file for your customizations. When you have more than one plugin, add one fully qualified class name per line. For details, see the reference documentation for the Java class ServiceLoader. If you build your project using Maven, then you can add this under the `src/main/resources` directory. As described in "Embedding the Customization in IG", you must add your custom libraries to the `WEB-INF/lib/` directory of the IG .war file that you deploy.

Be sure to provide some documentation for IG administrators on how your plugin extends expressions.

Filter

A `Filter` serves to process a request before handing it off to the next element in the chain, in a similar way to an interceptor programming model.

The `Filter` interface exposes a `filter()` method, which takes a Context, a Request, and the Handler, which is the next filter or handler to dispatch to. The `filter()` method returns a Promise that provides access to the Response with methods for dealing with both success and failure conditions.

A filter can elect not to pass the request to the next filter or handler, and instead handle the request itself. It can achieve this by merely avoiding a call to `next.handle(context, request)`, creating its own response object and returning that in the promise. The filter is also at liberty to replace a response with another of its own. A filter can exist in more than one chain, therefore should make no assumptions or correlations using the chain it is supplied. The only valid use of a chain by a filter is to call its `handle()` method to dispatch the request to the rest of the chain.

Handler

A `Handler` generates a response for a request.

The `Handler` interface exposes a `handle()` method, which takes a Context, and a Request. It processes the request and returns a Promise that provides access to the Response with methods for dealing with both success and failure conditions. A handler can elect to dispatch the request to another handler or chain.

ClassAliasResolver

A `ClassAliasResolver` makes it possible to replace a fully qualified class name with a short name (an alias) in an object declaration's type.

The `ClassAliasResolver` interface exposes a `resolve(String)` method to do the following:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Return the class mapped to a given alias
- Return `null` if the given alias is unknown to the resolver

All resolvers available to IG are asked until the first non-null value is returned or until all resolvers have been contacted.

The order of resolvers is nondeterministic. To prevent conflicts, don't use the same alias for different types.

17.6.2. Implementing a Customized Sample Filter

The `SampleFilter` class implements the `Filter` interface to set a header in the incoming request and in the outgoing response.

In the following example, the sample filter adds an arbitrary header:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
package org.forgerock.openig.doc;

import org.forgerock.http.Filter;
import org.forgerock.http.Handler;
import org.forgerock.http.protocol.Request;
import org.forgerock.http.protocol.Response;
import org.forgerock.openig.heap.GenericHeaplet;
import org.forgerock.openig.heap.HeapException;
import org.forgerock.services.context.Context;
import org.forgerock.util.promise.NeverThrowsException;
import org.forgerock.util.promise.Promise;

/**
 * Filter to set a header in the incoming request and in the outgoing response.
 */
public class SampleFilter implements Filter {

    /** Header name. */
    String name;

    /** Header value. */
    String value;

    /**
     * Set a header in the incoming request and in the outgoing response.
     * A configuration example looks something like the following.
     *
     * <pre>
     * {
     *     "name": "SampleFilter",
     *     "type": "SampleFilter",
     *     "config": {
     *         "name": "X-Greeting",
     *         "value": "Hello world"
     *     }
     * }
     * </pre>
     *
     * @param context           Execution context.
     * @param request           HTTP Request.
     * @param next              Next filter or handler in the chain.
     * @return A {@code Promise} representing the response to be returned to the client.
     */
    @Override
    public Promise<Response, NeverThrowsException> filter(final Context context,
                                                          final Request request,
                                                          final Handler next) {

        // Set header in the request.
        request.getHeaders().put(name, value);

        // Pass to the next filter or handler in the chain.
        return next.handle(context, request)
                   // When it has been successfully executed, execute the following callback
                   .thenOnResult(response -> {
                       // Set header in the response.
                       response.getHeaders().put(name, value);
                   });
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    }

    /**
     * Create and initialize the filter, based on the configuration.
     * The filter object is stored in the heap.
     */
    public static class Heaplet extends GenericHeaplet {

        /**
         * Create the filter object in the heap,
         * setting the header name and value for the filter,
         * based on the configuration.
         *
         * @return                  The filter object.
         * @throws HeapException    Failed to create the object.
         */
        @Override
        public Object create() throws HeapException {

            SampleFilter filter = new SampleFilter();
            filter.name  = config.get("name").as(evaluatedWithHeapProperties()).required().asString();
            filter.value = config.get("value").as(evaluatedWithHeapProperties()).required().asString();

            return filter;
        }
    }
}
```

The corresponding filter configuration is similar to this:

```
{
  "name": "SampleFilter",
  "type": "org.forgerock.openig.doc.SampleFilter",
  "config": {
    "name": "X-Greeting",
    "value": "Hello world"
  }
}
```

Note how `type` is configured with the fully qualified class name for `SampleFilter`. To simplify the configuration, implement a class alias resolver, as described in "Implementing a Class Alias Resolver".

17.6.3. Implementing a Class Alias Resolver

To simplify the configuration of a customized object, implement a `ClassAliasResolver` to allow the use of short names instead of fully qualified class names.

In the following example, a `ClassAliasResolver` is created for the `SampleFilter` class:

```
package org.forgerock.openig.doc;

import org.forgerock.openig.alias.ClassAliasResolver;

import java.util.HashMap;
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
import java.util.Map;

/**
 * Allow use of short name aliases in configuration object types.
 *
 * This allows a configuration with {@code "type": "SampleFilter"}
 * instead of {@code "type": "org.forgerock.openig.doc.SampleFilter"}.
 */
public class SampleClassAliasResolver implements ClassAliasResolver {

    private static final Map<String, Class<?>> ALIASES =
            new HashMap<>();

    static {
        ALIASES.put("SampleFilter", SampleFilter.class);
    }

    /**
     * Get the class for a short name alias.
     *
     * @param alias Short name alias.
     * @return      The class, or null if the alias is not defined.
     */
    @Override
    public Class<?> resolve(String alias) {
        return ALIASES.get(alias);
    }
}
```

With this `ClassAliasResolver`, the filter configuration in "Implementing a Customized Sample Filter" can use the alias instead of the fully qualified class name, as follows:

```
{
  "name": "SampleFilter",
  "type": "SampleFilter",
  "config": {
    "name": "X-Greeting",
    "value": "Hello world"
  }
}
```

To create a customized `ClassAliasResolver`, add a services file with the following characteristics:

- Name the file after the class resolver interface.
- Store the file under `META-INF/services/org.forgerock.openig.alias.ClassAliasResolver`, in the customization .jar file.

  If you build your project using Maven, you can add the file under the `src/main/resources` directory.
- In your ClassAliasResolver file, add a line for the fully qualified class name of your resolver as follows:

  ```
  org.forgerock.openig.doc.SampleClassAliasResolver
  ```

  If you have more than one resolver in your .jar file, add one line for each fully qualified class name.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

17.6.4. Configuring the Heap Object for the Customization

Objects are added to the heap and supplied with configuration artifacts at initialization time. To be integrated with the configuration, a class must have an accompanying implementation of the Heaplet interface. The easiest and most common way of exposing the heaplet is to extend the GenericHeaplet class in a nested class of the class you want to create and initialize, overriding the heaplet's `create()` method.

Within the `create()` method, you can access the object's configuration through the `config` field.

17.6.5. Embedding the Customization in IG

After building your customizations into a .jar file, you can include them in the IG .war file for deployment. You do this by unpacking `IG-6.5.4.war`, including your .jar library in `WEB-INF/lib`, and then creating a new .war file.

For example, if your .jar file is in a project named `sample-filter`, and the development version is `1.0.0-SNAPSHOT`, you might include the file as in the following example:

```
$ mkdir root && cd root
$ jar -xf ~/Downloads/IG-6.5.4.war
$ cp ~/Documents/sample-filter/target/sample-filter-1.0.0-SNAPSHOT.jar WEB-INF/lib
$ jar -cf ../custom.war *
```

In this example, the resulting `custom.war` contains the custom sample filter. You can deploy the custom .war file as you would deploy `IG-6.5.4.war`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 18
Auditing

The ForgeRock Common Audit Framework is a platform-wide infrastructure to handle audit events by using common audit event handlers.

Each audit event is identified by a unique transaction ID that can be communicated across products and recorded for each local event. By using the transaction ID, requests can be tracked as they traverse the platform, making it easier to monitor activity and to enrich reports.

Transaction IDs from other services in the ForgeRock platform are sent as `X-ForgeRock-TransactionId` header values. By default, IG does not trust transaction ID headers from client applications.

Note

If you trust transaction IDs sent by client applications, and want monitoring and reporting systems consuming the logs to allow correlation of requests as they traverse multiple servers, then set the boolean system property `org.forgerock.http.TrustTransactionHeader` to `true` in the Java command to start the container where IG runs.

Important

In your AuditService configuration, consider using the following filter policy to exclude sensitive data from log files:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
{
  "name": "MyAuditService",
  "type": "AuditService",
  "config": {
    "config": {
      "filterPolicies": {
        "field": {
          "excludeIf": [
            "/access/http/request/cookies/iPlanetDirectoryPro",
            "/access/http/request/headers/iPlanetDirectoryPro",
            "/access/http/request/headers/AMAuthCookie",
            "/access/http/request/headers/authorization",
            "/access/http/request/headers/proxy-authorization",
            "/access/http/request/headers/X-OpenAM-Password",
            "/access/http/request/headers/X-OpenIDM-Password",
            "/access/http/request/queryParameters/access_token",
            "/access/http/request/queryParameters/id_token_hint",
            "/access/http/request/queryParameters/IDToken1",
            "/access/http/request/queryParameters/Login.Token1",
            "/access/http/request/queryParameters/redirect_uri",
            "/access/http/request/queryParameters/requester",
            "/access/http/request/queryParameters/sessionUpgradeSSOTokenId",
            "/access/http/request/queryParameters/tokenId",
            "/access/http/response/headers/Set-Cookie"
          ]
        }
      }
    },
    "event-handlers": MyEventHandler
  }
}
```

If you use a different name for `iPlanetDirectoryPro` or `AMAuthCookie`, edit the list to include the correct name.

18.1. Recording Audit Events in CSV

This section describes how to record audit events in a CSV file. For information about the CSV audit event handler, see CsvAuditEventHandler(5) in the *Configuration Reference*.

Important

The CSV handler does not sanitize messages when writing to CSV log files.

Do not open CSV logs in spreadsheets and other applications that treat data as code.

To Record Audit Events in a CSV File

Before you start, prepare IG and the sample application as described in "*First Steps*" in the *Getting Started Guide*.

1. Add the following route to the IG as `$HOME/.openig/config/routes/30-audit.json`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

On Windows, add the route as `%appdata%\OpenIG\config\routes\30-audit.json`.

```
{
  "handler": "ForgeRockClientHandler",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/audit')}",
  "auditService": {
    "type": "AuditService",
    "config": {
      "config": {},
      "event-handlers": [
        {
          "class": "org.forgerock.audit.handlers.csv.CsvAuditEventHandler",
          "config": {
            "name": "csv",
            "logDirectory": "/tmp/logs",
            "buffering": {
              "enabled": "true",
              "autoFlush": "true"
            },
            "topics": [
              "access"
            ]
          }
        }
      ]
    }
  }
}
```

The route calls an audit service configuration for publishing log messages to the CSV file, `/tmp/logs/access.csv`. When a request matches `audit`, audit events are logged to the CSV file.

The route uses the `ForgeRockClientHandler` as its handler, to send the `X-ForgeRock-TransactionId` header with its requests to external services.

2. Access the route on http://openig.example.com:8080/home/audit.

   The home page of the sample application should be displayed and the file `/tmp/logs/access.csv` should be updated.

18.2. Recording Audit Events in Elasticsearch

This section describes how to record audit events with an Elasticsearch audit event handler. For information about configuring the Elasticsearch event handler, see ElasticsearchAuditEventHandler(5) in the *Configuration Reference*.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Record Audit Events in Elasticsearch

Before you start, prepare IG and the sample application as described in "*First Steps*" in the *Getting Started Guide*, and access Studio as described in "Accessing Studio" in the *Getting Started Guide*. IG must be running in development mode.

1. Make sure that Elasticsearch is installed and running.

   For Elasticsearch downloads and installation instructions, see the Elasticsearch *Getting Started* document. For information about configuring the Elasticsearch event handler, see ElasticsearchAuditEventHandler(5) in the *Configuration Reference*.

2. Browse to http://openig.example.com:8080/openig/studio, and select Protect an Application.

3. Choose Structured to use the predefined menus and templates.

4. Select Advanced options, and create a route with the following options:

   - Base URI: `http://app.example.com:8081`
   - Condition: Path: `/home/audit`
   - Name: `30-elasticsearch`

5. Configure auditing:

   a. Select Audit, and then enable it.

   b. Select New event handler and then Elasticsearch event handler.

   c. Enter the following information, and then save the settings:

      - Name: `elasticsearch`

      Leave the other fields with their default values and save.

   d. In the event handlers frame, enable the event handler.

6. On the top-right of the screen, select and Display to review the route.

   The following route should be displayed:

```
{
  "name": "30-elasticsearch",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/audit')}",
  "auditService": {
    "name": "AuditService",
    "type": "AuditService",
    "config": {
      "event-handlers": [
        {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
          "class": "org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEventHandler",
          "config": {
            "name": "elasticsearch",
            "indexMapping": {
              "indexName": "audit"
            },
            "connection": {
              "host": "localhost",
              "port": 9200,
              "useSSL": false
            },
            "topics": [
              "access"
            ]
          }
        }
      ]
    }
  },
  "handler": "ReverseProxyHandler"
}
```

7. (Optional) Select Edit and set options to manage the connection, index mapping, and buffering. Use information in ElasticsearchAuditEventHandler(5) in the *Configuration Reference*.
8. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

To Test the Setup

1. Access the route on http://openig.example.com:8080/home/audit.

   The home page of the sample application is displayed and events are logged in Elasticsearch.
2. Access the ElasticSearch URL, `http://localhost:9200/audit/access/_search?q='"OPENIG-HTTP-ACCESS"'`

   The audit events logged in Elasticsearch are displayed.
3. Repeat the previous steps to confirm that each time you access the IG, Elasticsearch is updated.

18.3. Recording Audit Events in JMS

> **Important**
>
> This procedure is an example of how to record audit events with a JMS audit event handler configured to use the ActiveMQ message broker. This example is not tested on all configurations, and can be more or less relevant to your configuration.

For information about configuring the JMS event handler, see JmsAuditEventHandler(5) in the *Configuration Reference*.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Record Audit Events With a JMS Audit Event Handler

Before you start, prepare IG as described in "*First Steps*" in the *Getting Started Guide*.

1. Add ActiveMQ client dependencies to IG:

   a. Download the following `.jar` files from :

      - `geronimo-j2ee-management_1.1_spec-1.0.1.jar`
      - `hawtbuf-1.11.jar`
      - `activemq-client-5.13.3.jar`

   b. Add the `.jar` files to the IG container, at `/path/to/jetty/webapps/ROOT/WEB-INF/lib`.

2. Download and install the ActiveMQ message broker from http://activemq.apache.org/. For help, see the the ActiveMQ documentation on the same site.

3. Create a consumer that subscribes to the `audit` topic.

   From the ActiveMQ installation directory, run the following command:

```
$ ./bin/activemq consumer --destination topic://audit
```

4. Add the following route to the IG as `$HOME/.openig/config/routes/30-jms.json`. On Windows, add the route as `%appdata%\OpenIG\config\routes\30-jms.json`.

```
{
  "MyCapture" : "all",
  "auditService" : {
    "config" : {
      "event-handlers" : [
        {
          "class" : "org.forgerock.audit.handlers.jms.JmsAuditEventHandler",
          "config" : {
            "name" : "jms",
            "topics" : [ "access" ],
            "deliveryMode" : "NON_PERSISTENT",
            "sessionMode" : "AUTO",
            "jndi" : {
              "contextProperties" : {
                "java.naming.factory.initial" :
 "org.apache.activemq.jndi.ActiveMQInitialContextFactory",
                "java.naming.provider.url" : "tcp://openam.example.com:61616",
                "topic.audit" : "audit"
              },
              "topicName" : "audit",
              "connectionFactoryName" : "ConnectionFactory"
            }
          }
        }
      ],
      "config" : { }
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    },
    "type" : "AuditService"
  },
  "handler" : {
    "type" : "StaticResponseHandler",
    "config" : {
      "status" : 200,
      "headers" : {
        "content-type" : [ "text/plain" ]
      },
      "reason" : "found",
      "entity" : "Message from audited route"
    }
  },
  "condition" : "${request.uri.path == '/activemq_event_handler'}"
}
```

When a request matches the `/activemq_event_handler` route, this configuration publishes JMS messages containing audit event data to an ActiveMQ managed JMS topic, and the StaticResponseHandler displays a message.

5. Access the route on http://openig.example.com:8080/activemq_event_handler.

   Depending on how ActiveMQ is configured, audit events are displayed on the ActiveMQ console or written to file. For example, the following log message can be written to a log file in the folder where you installed ActiveMQ:

```
{
  "auditTopic": "access",
  "event": {
    "eventName": "OPENIG-HTTP-ACCESS",
    "timestamp": "2016-11-28T14:39:30.004Z",
    "transactionId": "882918f9-f7c3-47ee-9f87-5e3cfcfb98be-37",
    "server": {
      "ip": "0:0:0:0:0:0:0:1",
      "port": 8080
    },
    "client": {
      "ip": "0:0:0:0:0:0:0:1",
      "port": 56095
    },
    "http": {
      "request": {
        "secure": false,
        "method": "GET",
        "path": "http://openig.example.com:8080/activemq_event_handler",
        "queryParameters": {},
        "headers": {
          "accept": ["*/*"],
          "accept-encoding": ["gzip, deflate"],
          "Connection": ["keep-alive"],
          "host": ["openig.example.com:8080"],
          "user-agent": ["python-requests/2.9.1"]
        },
        "cookies": {}
      },
      "response": {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
        "headers": {
          "Content-Length": ["26"],
          "Content-Type": ["text/plain"]
        }
      }
    },
    "response": {
      "status": "SUCCESSFUL",
      "statusCode": "200",
      "elapsedTime": 73,
      "elapsedTimeUnits": "MILLISECONDS"
    },
    "_id": "882918f9-f7c3-47ee-9f87-5e3cfcfb98be-38"
  }
}
```


18.4. Recording Audit Events in JSON

This section describes how to record audit events with a JSON audit event handler. For information about configuring the JSON event handler, see JsonAuditEventHandler(5) in the *Configuration Reference*.

To Record Audit Events With a JSON Audit Event Handler

Before you start, prepare IG and the sample application as described in "*First Steps*" in the *Getting Started Guide*, and access Studio as described in "Accessing Studio" in the *Getting Started Guide*. IG must be running in development mode.

1. Browse to http://openig.example.com:8080/openig/studio, and select Protect an Application.
2. Choose Structured to use the predefined menus and templates.
3. Select Advanced options, and create a route with the following options:
   - Base URI: `http://app.example.com:8081`
   - Condition: Path: `/home/audit`
   - Name: `30-json`
4. Configure auditing:
   a. Select Audit, and then enable it.
   b. Select New event handler and then JSON audit event handler.
   c. Enter the following information, and then save the settings:
      - Name: `json`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Log directory: `/tmp/logs`

  Leave the other fields with their default values and save.

5. In the event handlers frame, enable the event handler
6. On the top-right of the screen, select ⋮ and Display to review the route.

   The following route should be displayed:

```
{
  "name": "30-json",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/audit')}",
  "auditService": {
    "name": "AuditService",
    "type": "AuditService",
    "config": {
      "event-handlers": [
        {
          "class": "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
          "config": {
            "name": "json",
            "logDirectory": "/tmp/logs",
            "elasticsearchCompatible": false,
            "topics": [
              "access"
            ],
            "fileRetention": {
              "rotationRetentionCheckInterval": "1 minute"
            },
            "buffering": {
              "maxSize": 100000,
              "writeInterval": "100 ms"
            }
          }
        }
      ]
    }
  },
  "handler": "ReverseProxyHandler"
}
```

   The route calls an audit service configuration for publishing log messages to the JSON file, `/tmp/audit/access.audit.json`. When a request matches `/home/json-audit`, a single line per audit event is logged to the JSON file.

   The route uses the `ForgeRockClientHandler` as its handler, to send the `X-ForgeRock-TransactionId` header with its requests to external services.

7. (Optional) Select Edit and set options to manage compatibility with the ElasticSearch JSON format, or file rotation, retention, and buffering. Use information in JsonAuditEventHandler(5) in the *Configuration Reference*.
8. Select Deploy to push the route to the IG configuration.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

To Test the Setup

- Access the route on http://openig.example.com:8080/home/audit.

  The home page of the sample application is displayed and the file `/tmp/logs/access.audit.json` is created or updated with a message. The following example message is formatted for easy reading, but it is produced as a single line for each event:


```
{
  "eventName": "OPENIG-HTTP-ACCESS",
  "timestamp": "2016-11-08T15:39:59.128Z",
  "transactionId": "a386a21c-0ceb-4c6b-af77-167bd71f0161-1",
  "server": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 8080
  },
  "client": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 34066
  },
  "http": {
    "request": {
      "secure": false,
      "method": "GET",
      "path": "http://openig.example.com:8080/home/json-audit",
      "queryParameters": {},
      "headers": {
        "accept": ["text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"],
        "accept-encoding": ["gzip, deflate"],
        "Accept-Language": ["en-US;q=1"],
        "cache-control": ["max-age=0"],
        "Connection": ["keep-alive"],
        "host": ["openig.example.com:8080"],
        "upgrade-insecure-requests": ["1"],
        "user-agent": ["Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6)
                        AppleWebKit / 602.2 .14(KHTML, like Gecko)
                        Version / 10.0 .1 Safari / 602.2 .14 "]
      },
      "cookies": {
        "i18next": "en"
      }
    },
    "response": {
      "status": "SUCCESSFUL",
      "statusCode": "200",
      "elapsedTime": 104,
      "elapsedTimeUnits": "MILLISECONDS"
    },
    "_id": "a386a21c-0ceb-4c6b-af77-167bd71f0161-2"
  }
}
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

18.5. Recording Audit Events to Standard Output

This section describes how to record audit events to standard output. For more information about the event handler, see JsonStdoutAuditEventHandler(5) in the *Configuration Reference*.

To Record Audit Events to Standard Output

Before you start, prepare IG and the sample application as described in "*First Steps*" in the *Getting Started Guide*.

- Add the following route to the configuration as `$HOME/.openig/config/routes/30-jsonstdout.json` (on Windows, `%appdata%\OpenIG\config\routes\30-jsonstdout.json`):

```
{
  "name": "30-jsonstdout",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/jsonstdout-audit')}",
  "auditService": {
    "name": "AuditService-1",
    "type": "AuditService",
    "config": {
      "event-handlers": [
        {
          "class": "org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHandler",
          "config": {
            "name": "jsonstdout",
            "elasticsearchCompatible": false,
            "topics": ["access"]
          }
        }
      ]
    }
  },
  "handler": "ReverseProxyHandler"
}
```

  Notice the following features of the route:

  - The route matches requests to `/home/jsonstdout-audit`.
  - The route calls the audit service configuration for publishing access log messages to standard output. When a request matches `/home/jsonstdout-audit`, a single line per audit event is logged.

To Test the Setup

- Access the route on http://openig.example.com:8080/home/audit.

  The home page of the sample application is displayed, and a message like this is published to standard output:

```
{
  "eventName": "OPENIG-HTTP-ACCESS",
  "timestamp": "2018-09-12T11:43:44.853Z",
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
  "transactionId": "bac95a21-fa8a-4e73-ab4c-ca95919856eb-149",
  "server": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 8080
...
  },
  "_id": "bac95a21-fa8a-4e73-ab4c-ca95919856eb-150",
  "source": "audit",
  "topic": "access",
  "level": "INFO"
}
```

18.6. Recording Audit Events in Splunk

This section describes how to set up a Splunk audit event handler to log IG events to a Splunk system. For information about configuring the Splunk event handler, see SplunkAuditEvenHandler(5) in the *Configuration Reference*.

To Set Up Splunk

This procedure assumes a Splunk instance running on the same host as IG. Adjust the instructions for your Splunk system.

1. Download Splunk from http://www.splunk.com, and install it with the default configuration. If you don't already have a Splunk account, create one.

   Tip

   Splunk currently uses the following ports by default: `8000`, `8065`, `8088`, `8089`, and `8091`. Before you install Splunk, make sure that these ports are free. Alternatively, change the Splunk installation and IG route to use other ports.

   To find port numbers and other settings used by Splunk, select Server settings > General settings in the Splunk web interface.

2. With Splunk running, create a new source type and associate it with log data from IG:

   a. In the Splunk web interface, select Settings > Source Types > New Source Type.

   b. In the Create Source Type window, enter a name for the source type, for example, `openig`.

   c. In the Event Breaks panel of the same window, select Regex... and enter `^{` to indicate how the bulk messages are separated.

   d. Accept all of the other values as default and select Save.

3. Create an HTTP Event Collector to provide an authorization token so that IG can log events to Splunk:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

a. Select Settings > Data Inputs > HTTP Event Collector > New Token.

b. Enter a Name for the token, for example, `openig`, leave the other fields with their default values, and select Next.

c. In the Input Settings screen, select Select > Select Source Type > Custom, and then select the source type you created in the previous step.

d. Select Review and then Submit.

An authorization token is displayed. Make a note of the value or keep it on the screen so that you use it as the value of `authzToken` in "To Set Up IG for the Splunk Audit Event Handler".

4. In the HTTP Event Collector window, check that the Global Settings are configured correctly. For example, make sure that all tokens are enabled and that SSL is not enabled.

   The HTTP port number displayed in these global settings is used as the value of `port` in "To Set Up IG for the Splunk Audit Event Handler".

To Set Up IG for the Splunk Audit Event Handler

Before you start, prepare IG and the sample application as described in "*First Steps*" in the *Getting Started Guide*, and access Studio as described in "Accessing Studio" in the *Getting Started Guide*. IG must be running in development mode.

1. In IG Studio, create a route:

   a. Browse to http://openig.example.com:8080/openig/studio, and select Protect an Application.

   b. Choose Structured to use the predefined menus and templates.

2. Select Advanced options, and create a route with the following options:

   - Base URI: `http://app.example.com:8081`
   - Condition: Path: `/home/audit`
   - Name: `30-splunk`

3. Configure auditing:

   a. Select Audit, and then enable it.

   b. Select New event handler and then Splunk event handler.

   c. Enter the following information, and then save the settings:

      - Name: `splunk`

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

- Authorization token: enter the value of the authorization token returned in "To Set Up Splunk".

Leave the other fields with their default values and save.

4. In the event handlers frame, enable the event handler
5. On the top-right of the screen, select ⋮ and Display to review the route.

   The following route should be displayed:

```
{
  "name": "30-splunk",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/audit')}",
  "auditService": {
    "name": "AuditService",
    "type": "AuditService",
    "config": {
      "event-handlers": [
        {
          "class": "org.forgerock.audit.handlers.splunk.SplunkAuditEventHandler",
          "config": {
            "name": "splunk",
            "enabled": true,
            "authzToken": "<splunk-authorization-token>",
            "connection": {
              "host": "localhost",
              "port": 8088,
              "useSSL": false
            },
            "topics": [
              "access"
            ],
            "buffering": {
              "maxSize": 10000,
              "maxBatchedEvents": 500,
              "writeInterval": "100 ms"
            }
          }
        }
      ]
    }
  },
  "handler": "ReverseProxyHandler"
}
```

   The route calls an audit service configuration for publishing log messages to Splunk.

6. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Test the Setup

1. Access the route on http://openig.example.com:8080/home/audit.

   The home page of the sample application is displayed and events are logged in Splunk.

2. Access the Splunk web interface on http://localhost:8000, and select Search & Reporting > Data Summary.

   Depending on how Splunk is configured, audit events are displayed on the web interface.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 19
Monitoring

IG can collect metrics on requests and responses, and expose metrics over the following HTTP endpoints:

- Prometheus Scrape Endpoint
- Common REST Monitoring Endpoint

19.1. Prometheus Scrape Endpoint

All ForgeRock products automatically expose a monitoring endpoint where Prometheus can scrape metrics, in a standard Prometheus format. For information about the endpoint and links to available metrics, see "Prometheus Scrape Endpoint" in the *Configuration Reference*. This section gives an example query of the Prometheus Scrape Endpoint.

To Monitor the Prometheus Scrape Endpoint

1. Add the following route to the IG configuration as `$HOME/.openig/config/routes/myroute1.json` (on Windows, `%appdata%\OpenIG\config\routes`):

```
{
  "name": "myroute1",
  "handler": {
    "type": "StaticResponseHandler",
    "config": {
      "status": 200,
      "reason": "OK",
      "entity": "Hello, world from myroute1!"
    }
  },
  "condition": "${matches(request.uri.path, '^/myroute1')}"
}
```

   The route contains a StaticResponseHandler to display a simple message.

2. Access the route a few times, on http://openig.example.com:8080/myroute1.

3. Query the Prometheus Scrape Endpoint at http://openig.example.com:8080/openig/metrics/prometheus.

   Metrics for `myroute1` and `_router` are displayed:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
# HELP ig_router_deployed_routes Generated from Dropwizard metric import
 (metric=gateway._router.deployed-routes, type=gauge)
# TYPE ig_router_deployed_routes gauge
ig_router_deployed_routes{fully_qualified_name="gateway._router",heap="gateway",name="_router",} 1.0
# HELP ig_route_request_active Generated from Dropwizard metric import
 (metric=gateway._router.route.default.request.active, type=gauge)
# TYPE ig_route_request_active gauge
ig_route_request_active{name="default",route="default",router="gateway._router",} 0.0
# HELP ig_route_request_active Generated from Dropwizard metric import
 (metric=gateway._router.route.myroute1.request.active, type=gauge)
# TYPE ig_route_request_active gauge
ig_route_request_active{name="myroute1",route="myroute1",router="gateway._router",} 0.0
# HELP ig_route_request_total Generated from Dropwizard metric import
 (metric=gateway._router.route.default.request, type=counter)
# TYPE ig_route_request_total counter
ig_route_request_total{name="default",route="default",router="gateway._router",} 0.0
# HELP ig_route_response_error Generated from Dropwizard metric import
 (metric=gateway._router.route.default.response.error, type=counter)
# TYPE ig_route_response_error counter
ig_route_response_error{name="default",route="default",router="gateway._router",} 0.0
# HELP ig_route_response_null Generated from Dropwizard metric import
 (metric=gateway._router.route.default.response.null, type=counter)
# TYPE ig_route_response_null counter
ig_route_response_null{name="default",route="default",router="gateway._router",} 0.0
# HELP ig_route_response_status_total Generated from Dropwizard metric import
 (metric=gateway._router.route.default.response.status.client_error, type=counter)
# TYPE ig_route_response_status_total counter
ig_route_response_status_total{family="client_error",name="default",route="default",router="gateway._router",}
 0.0
...
```

19.2. Common REST Monitoring Endpoint

All ForgeRock products expose a monitoring endpoint where metrics are exposed as a JSON format monitoring resource. For information about the endpoint and links to available metrics, see "Common REST Monitoring Endpoint" in the *Configuration Reference*. This section gives an example query of the Common REST Monitoring Endpoint.

To Monitor the Common REST Monitoring Endpoint

Before you start, prepare IG as described in "*First Steps*" in the *Getting Started Guide*.

1. Set up IG and some example routes, as described in the first few steps of "To Monitor the Prometheus Scrape Endpoint".
2. Query the Common REST Monitoring Endpoint at http://openig.example.com:8080/openig/metrics/api?_prettyPrint=true&_sortKeys=_id&_queryFilter=true

   Metrics for `myroute1` and `_router` are displayed:

```
{
  "result" : [ {
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    "_id" : "gateway._router.deployed-routes",
    "value" : 1.0,
    "_type" : "gauge"
    }, {
    "_id" : "gateway._router.route.default.request",
    "count" : 204,
    "_type" : "counter"
    }, {
    "_id" : "gateway._router.route.default.request.active",
    "value" : 0.0,
    "_type" : "gauge"
    }, {

   . . .

   _id" : "gateway._router.route.myroute1.response.status.unknown",
   "count" : 0,
   "_type" : "counter"
    }, {
    "_id" : "gateway._router.route.myroute1.response.time",
    "count" : 204,
    "max" : 0.420135,
    "mean" : 0.08624678327176545,
    "min" : 0.045079999999999995,
    "p50" : 0.070241,
    "p75" : 0.096049,
    "p95" : 0.178534,
    "p98" : 0.227217,
    "p99" : 0.242554,
    "p999" : 0.420135,
    "stddev" : 0.046611762381930474,
    "m15_rate" : 0.2004491450567003,
    "m1_rate" : 2.8726563452698075,
    "m5_rate" : 0.5974045160056258,
    "mean_rate" : 0.010877725092634833,
    "duration_units" : "milliseconds",
    "rate_units" : "calls/second",
    "total" : 17.721825,
    "_type" : "timer"
    } ],
    "resultCount" : 11,
    "pagedResultsCookie" : null,
    "totalPagedResultsPolicy" : "EXACT",
    "totalPagedResults" : 11,
    "remainingPagedResults" : -1
}
```


3. Change the query to access metrics only for `myroute1`: http://openig.example.com:8080/openig/metrics/api?_prettyPrint=true&_sortKeys=_id&_queryFilter=_id+sw+"gateway._router.route.myroute1".

   Note that metric for the router, `"_id" : "gateway._router.deployed-routes"`, is no longer displayed.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

19.3. Protecting the Monitoring Endpoints

By default, everyone has read-access to the Prometheus Scrape Endpoint and Common REST Monitoring Endpoint. No special credentials or privileges are required.

To protect these monitoring endpoints, add an `admin.json` file to your configuration, with a filter declared in the heap and named `MetricsProtectionFilter`. The following procedure gives an example of how to manage access to the monitoring endpoints:

To Protect the Monitoring Endpoints

1. Add the following script to the IG configuration as `$HOME/.openig/scripts/groovy/BasicAuthResourceServerFilter.groovy` (on Windows, `%appdata%\OpenIG\scripts\groovy\BasicAuthResourceServerFilter.groovy`): this script is a simple implementation of the HTTP Basic Authentication mechanism.

   With such a filter, the endpoint `/openig/metrics` is not now protected through the standard HTTP Basic Authentication mechanism. It is possible to use here any kind of filter to protect that endpoint.

```
/**
 * This scripts is a simple implementation of HTTP Basic Authentication on
 * server side.
 * It expects the following arguments:
 *  - realm: the realm to display when the user-agent prompts for
 *    username and password if none were provided.
 *  - username: the expected username
 *  - password: the expected password
 */

import static org.forgerock.util.promise.Promises.newResultPromise;

import java.nio.charset.Charset;
import org.forgerock.util.encode.Base64;

String authorizationHeader = request.getHeaders().getFirst("Authorization");
if (authorizationHeader == null) {
    // No credentials provided, reply that they are needed.
    Response response = new Response(Status.UNAUTHORIZED);
    response.getHeaders().put("WWW-Authenticate", "Basic realm=\"" + realm + "\"");
    return newResultPromise(response);
}

String expectedAuthorization = "Basic " + Base64.encode((username + ":" +
 password).getBytes(Charset.defaultCharset()))
if (!expectedAuthorization.equals(authorizationHeader)) {
    return newResultPromise(new Response(Status.FORBIDDEN));
}
// Credentials are as expected, let's continue
return next.handle(context, request);
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

For information about scripting filters and handlers, see "*Extending IG*".

2. Add the following file as `$HOME/.openig/config/admin.json` (on Windows, `%appdata%\OpenIG\config\admin.json`):

```
{
  "heap": [{
    "name": "ClientHandler",
    "type": "ClientHandler"
  }, {
    "name": "MetricsProtectionFilter",
    "type": "ScriptableFilter",
    "config": {
      "type": "application/x-groovy",
      "file": "BasicAuthResourceServerFilter.groovy",
      "args": {
        "realm": "/",
        "username": "metric",
        "password": "password"
      }
    }
  }],
  "prefix": "openig"
}
```

3. Restart IG to reload the configuration.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 20
Logging Events

Log messages in IG and third-party dependencies are recorded using the Logback implementation of the Simple Logging Facade for Java (SLF4J) API.

20.1. Default Logging Behavior

Log messages are recorded with the following default configuration:

- When IG starts, log messages for IG and third-party dependencies, such as the ForgeRock Common Audit framework, are displayed on the console and written to `$HOME/.openig/logs/route-system.log`.
- When a route is accessed, log messages for requests and responses passing through the route are written to a log file in `$HOME/.openig/logs`, and named by the route name or filename.

  For more information, see "Capturing Log Messages for Routes" and CaptureDecorator(5) in the *Configuration Reference*.
- By default, log messages with the level `INFO` or higher are recorded, with the titles and the top line of the stack trace. Messages on the console are highlighted with a color related to their logging level.

20.2. Reference Logback Configuration

The content and format of logs is defined by the reference `logback.xml` delivered with IG. This file defines the following configuration items for logs:

- A root logger to set the overall level to `INFO`, and to write all log messages to the `SIFT` and `STDOUT` appenders.
- A `STDOUT` appender to define the format of log messages on the console.
- A `SIFT` appender to separate log messages according to the key `routeId`, to define when log files are rolled, and to define the format of log messages in the file.
- An exception logger, called `LogAttachedExceptionFilter`, to write log messages for exceptions attached to responses.

```
<?xml version="1.0" encoding="UTF-8"?><!--
  Copyright 2016-2018 ForgeRock AS. All Rights Reserved
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
  Use of this code requires a commercial software license with ForgeRock AS.
  or with one of its affiliates. All use shall be exclusively subject
  to such license between the licensee and ForgeRock AS.
--><configuration>

  <!--
    Avoid logs to be flooded in case of repetitive messages.

    Configuration properties:
     * AllowedRepetitions (int): threshold above which same messages won't be logged anymore
     * CacheSize (int): Remove eldest entry when hitting max size
   -->
  <!--<turboFilter class="ch.qos.logback.classic.turbo.DuplicateMessageFilter" />-->

  <!-- Allow configuration of JUL loggers within this file, without performance impact -->
  <contextListener class="ch.qos.logback.classic.jul.LevelChangePropagator"/>

  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%nopex[%thread] %highlight(%-5level) %boldWhite(%logger{35}) @%mdc{routeId:-system} -
 %message%n%highlight(%rootException{short})</pattern>
    </encoder>
  </appender>

  <appender name="SIFT" class="ch.qos.logback.classic.sift.SiftingAppender">
    <discriminator>
      <key>routeId</key>
      <defaultValue>system</defaultValue>
    </discriminator>
    <sift>
      <!-- Create a separate log file for each <key> -->
      <appender name="FILE-${routeId}" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${ig.instance.dir}/logs/route-${routeId}.log</file>

        <rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
          <!-- Rotate files daily -->
          <fileNamePattern>${ig.instance.dir}/logs/route-${routeId}-%d{yyyy-MM-dd}.%i.log</
fileNamePattern>

          <!-- each file should be at most 100MB, keep 30 days worth of history, but at most 3GB -->
          <maxFileSize>100MB</maxFileSize>
          <maxHistory>30</maxHistory>
          <totalSizeCap>3GB</totalSizeCap>
        </rollingPolicy>

        <encoder>
          <pattern>%date{"yyyy-MM-dd'T'HH:mm:ss,SSSXXX", UTC} | %-5level | %thread | %logger{20} | @
%mdc{routeId:-system} | %message%n%xException</pattern>
        </encoder>
      </appender>
    </sift>
  </appender>

  <!-- Disable logs of exceptions attached to responses by defining 'level' to OFF -->
  <logger name="org.forgerock.openig.filter.LogAttachedExceptionFilter" level="INHERITED"/>

  <root level="INFO">
    <appender-ref ref="SIFT"/>
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
    <appender-ref ref="STDOUT"/>
  </root>
</configuration>
```

20.3. Capturing Log Messages for Routes

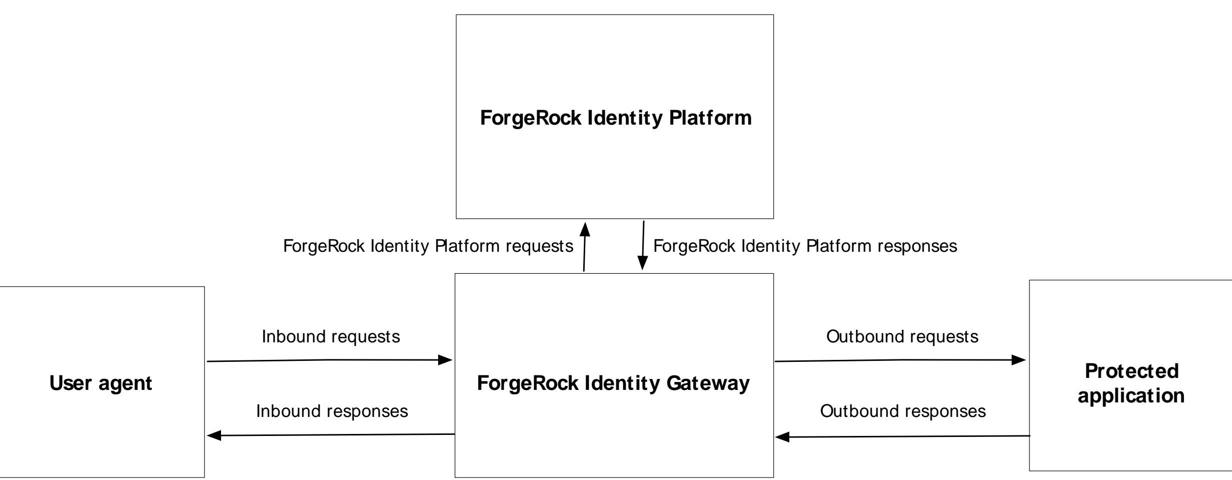
IG provides a CaptureDecorator to capture messages for requests and responses. The decorator can capture the context and entity of inbound and outbound messages for the route, or for the individual handlers and filters in the route.

> **Important**
>
> During debugging, consider using a CaptureDecorator to capture the entity and context of requests and responses. However, increased logging consumes resources, such as disk space, and can cause performance issues. In production, reduce logging by disabling the CaptureDecorator properties `captureEntity` and `captureContext`, or setting `maxEntityLength`.

Captured information is written to SLF4J logs. For more information about the decorator configuration, see CaptureDecorator(5) in the *Configuration Reference*.

Studio provides an easy way to capture messages while developing your configuration. The following image illustrates the capture points where you can log messages on a route:

Capturing Log Messages for Routes



To Capture Messages on a Route in Studio

1. In Studio, select ROUTES and then select a route.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

2. On the left side of the screen, select Capture, and then select capture options. You can capture the body and context of messages passing to and from the user agent, the protected application, and the ForgeRock Identity Platform.
3. Select Deploy to push the route to the IG configuration.

   You can check the `$HOME/.openig/config/routes` folder to see that the route is there.
4. Access the route, and then check `$HOME/.openig/logs` for a log file named by the route. The log file should contain the messages defined by your capture configuration.

20.4. Changing the Logging Behavior

The Logback configuration is very flexible, providing a wide range of options for logging. For a full description of its parameters, see the Logback website. The following examples show some simple changes that you can make.

To change the logging behavior, create a new Logback file at `$HOME/.openig/config/logback.xml`. This Logback file overrides the default configuration.

To take into account edits to `logback.xml`, stop and restart IG or edit the `configuration` parameter to add a scan and interval:

```
<configuration scan="true" scanPeriod="5 seconds">
```

The configuration `scan="true"` requires `logback.xml` to be scanned for changes. The file is scanned after both of the following criteria are met:

- The specified number of logging operations have occurred, where the default is 16.
- The scan period has elapsed, where the example specifies 5 seconds.

> **Note**
>
> If your custom `logback.xml` contains errors, messages like these are displayed on the console but log messages are not recorded:
>
> ```
> 14:38:59,667 |-ERROR in ch.qos.logback.core.joran.spi.Interpreter@20:72 ...
> 14:38:59,690 |-ERROR in ch.qos.logback.core.joran.action.AppenderRefAction ...
> ```

20.4.1. Formatting Log Messages

You can format log messages in many ways. For example, to add the date to the log message, edit `logback.xml` to change the pattern of the log messages in the `encoder` part of the SIFT appender:

```
%d{yyyyMMdd-HH:mm:ss} | %-5level | %thread | %logger{20} | %message%n%xException
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

20.4.2. Logging for Different Object Types

You can change the log messages for a single object type without changing them for the rest of the configuration.

For example, to record log messages with the level `ERROR` or higher for the ClientHandler, edit `logback.xml` to add a logger defined by the fully qualified class name of the ClientHandler, and to set its logging level to `ERROR`:

```
<logger name="org.forgerock.openig.handler.ClientHandler" level="ERROR"/>
```

Log messages with a level lower than `ERROR` are no longer recorded for the ClientHandler but continue to be recorded for the rest of the configuration.

20.4.3. Logging for Scripts

The logger object provides access to a unique SLF4J logger instance for scripts. Events are logged as defined in by a dedicated logger in `logback.xml`, and are included in the logs with the name of with the scriptable object.

To log events for scripts:

- Add logger objects to the script to enable logging at different levels. For example add some of the following logger objects:

  ```
  logger.error("ERROR")
  logger.warn("WARN")
  logger.info("INFO")
  logger.debug("DEBUG")
  logger.trace("TRACE")
  ```

- Add a logger to `logback.xml` to reference the scriptable object and set the logging level. The logger is defined by the type and name of the scriptable object that references the script, as follows:

  - ScriptableFilter: `org.forgerock.openig.filter.ScriptableFilter.`*`filter_name`*
  - ScriptableHandler: `org.forgerock.openig.handler.ScriptableHandler.`*`handler_name`*
  - ScriptableThrottlingPolicy: `org.forgerock.openig.filter.throttling .ScriptableThrottlingPolicy.`*`throttling_policy_name`*
  - ScriptableAccessTokenResolver: `org.forgerock.openig.filter.oauth2 .ScriptableAccessTokenResolver.`*`access_token_resolver_name`*

  For example, the following logger logs trace-level messages for a ScriptableFilter named `cors_filter`:

  ```
  <logger name="org.forgerock.openig.filter.ScriptableFilter.cors_filter" level="TRACE"/>
  ```

  The resulting messages in the logs contain the name of the scriptable object:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
14:54:38:307 | TRACE | http-nio-8080-exec-6 | o.f.o.f.S.cors_filter | TRACE
```

20.4.4. Logging for the BaseUriDecorator

During setup and configuration, it can be helpful to display log messages from the BaseUriDecorator.

For example, to record a log message each time a request URI is rebased, edit `logback.xml` to add a logger defined by the fully qualified class name of the BaseUriDecorator appended by the name of the baseURI decorator:

```
<logger name="org.forgerock.openig.decoration.baseuri.BaseUriDecorator.baseURI" level="TRACE"/>
```

Each time a request URI is rebased, a log message similar to this is created:

```
12:27:40| TRACE | http-nio-8080-exec-3 | o.f.o.d.b.B.b.{Router}/handler
        | Rebasing request to http://app.example.com:8081
```

20.4.5. Switching Off Exception Logging

To stop recording log messages for exceptions, edit `logback.xml` to set the level to `OFF`:

```
<logger name="org.forgerock.openig.filter.LogAttachedExceptionFilter" level="OFF"/>
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

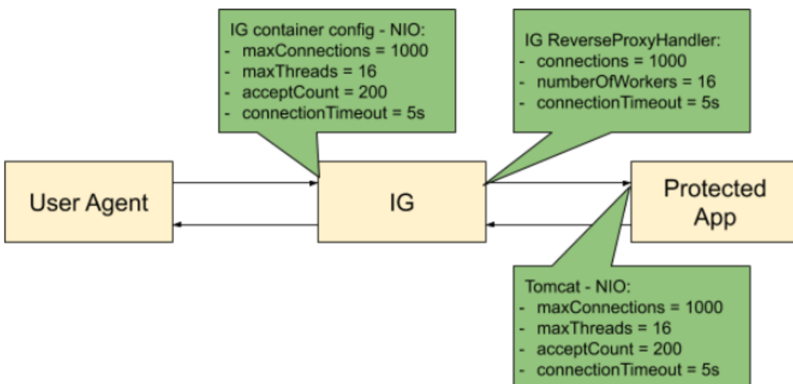
Chapter 21
Tuning Performance

Tune deployments in the following steps:

1. Consider the issues that impact the performance of a deployment. See "Defining Performance Requirements and Constraints".
2. Tune and test the downstream servers and applications:
    - Tune the downstream web container and JVM to achieve performance targets.
    - Test downstream servers and applications in a pre-production environment, under the expected load, and with common use cases.
    - Make sure that the configuration of the downstream web container can form the basis for IG and its container.
3. Tune IG and its web container:
    - Optimize IG performance, throughput, and response times. See "Tuning IG".
    - Configure IG connections to downstream services and protected applications. See "Tuning the ClientHandler/ReverseProxyHandler".
    - Configure connections in the IG web container. See "Tuning IG's Tomcat Container".
    - Configure the IG JVM to support the required throughput. See "Tuning IG's JVM".
4. Increase hardware resources as required, and then re-tune the deployment.

The following figure shows an example configuration for IG, its container, and the container for the protected app:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.



21.1. Defining Performance Requirements and Constraints

When you consider performance requirements, bear in mind the following points:

- The capabilities and limitations of downstream services or applications on your performance goals.
- The increase in response time due to the extra network hop and processing, when IG is inserted as a proxy in front of a service or application.
- The constraint that downstream limitations and response times places on IG and its container.

21.1.1. Service Level Objectives

A service level objective (SLO) is a target that you can measure quantitatively. Where possible, define SLOs to set out what performance your users expect. Even if your first version of an SLO consists of guesses, it is a first step towards creating a clear set of measurable goals for your performance tuning.

When you define SLOs, bear in mind that IG can depend on external resources that can impact performance, such as AM's response time for token validation, policy evaluation, and so on. Consider measuring remote interactions to take dependencies into account.

Consider defining SLOs for the following metrics of a route:

- Average response time for a route.

  The response time is the time to process and forward a request, and then receive, process, and forward the response from the protected application.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

The average response time can range from less than a millisecond, for a low latency connection on the same network, to however long it takes your network to deliver the response.

- Distribution of response times for a route.

  Because applications set timeouts based on worst case scenarios, the distribution of response times can be more important than the average response time.

- Peak throughput.

  The maximum rate at which requests can be processed at peak times. Because applications are limited by their peak throughput, this SLO is arguably more important than an SLO for average throughput.

- Average throughput.

  The average rate at which requests are processed.

Metrics are returned at the monitoring endpoints. For information and examples of monitoring, see Monitoring in the *Configuration Reference*.

21.1.2. Available Resources

With your defined SLOs, inventory the server, networks, storage, people, and other resources. Estimate whether it is possible to meet the requirements, with the resources at hand.

21.1.3. Benchmarks

Before you can improve the performance of your deployment, establish an accurate benchmark of its current performance. Consider creating a deployment scenario that you can control, measure, and reproduce.

For information about running benchmark tests on IG, see *ForgeOps*' Performance Benchmarks. Benchmark test results are given for throughput and response times in an AM password grant flow, and for IG resource server flows with and without cache.

21.2. Tuning IG

Consider the following recommendations for improving performance, throughput, and response times. Adjust the tuning to your system workload and available resources, and then test suggestions before rolling them out into production.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

21.2.1. Logs

Log messages in IG and third-party dependencies are recorded using the Logback implementation of the Simple Logging Facade for Java (SLF4J) API. By default, logging level is INFO.

To reduce the number of log messages, consider setting the logging level to `error`. For information, see "*Logging Events*".

21.2.2. Buffering Message Content

IG creates a TemporaryStorage object to buffer content during processing. For information about this object and its default values, see TemporaryStorage(5) in the *Configuration Reference*.

Messages bigger than the buffer size are written to disk, consuming I/O resources and reducing throughput.

The default size of the buffer is 64 KB. If the number of concurrent messages in your application is generally bigger than the default, consider allocating more heap memory or changing the initial or maximum size of the buffer.

To change the values, add a TemporaryStorage object named `TemporaryStorage`, and use non-default values.

21.2.3. Cache

When caches are enabled, IG can reuse cached information without making additional or repeated queries for the information. This gives the advantage of higher system performance, but the disadvantage of lower trust in results.

During service downtime, the cache is not updated, and important notifications can be missed, such as for the revocation of tokens or the update of policies, and IG can continue to use outdated tokens or policies.

When caches are disabled, IG must query a data store each time it needs data. This gives the disadvantage of lower system performance, and the advantage of higher trust in results.

When you configure caches in IG, make choices to balance your required performance with your security needs.

IG provides the following caches:

Session cache

After a user authenticates with AM, this cache stores information about the session. IG can reuse the information without asking AM to verify the session token (SSO token or CDSSO token) for each request.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

If WebSocket notifications are enabled, the cache evicts entries based on session notifications from AM, making the cache content more accurate (trustable).

By default, the session information is not cached. To increase performance, consider enabling and configuring the cache. For more information, see `sessionCache` in AmService(5) in the *Configuration Reference*.

Policy cache

After an AM policy decision, this cache stores the decision. IG can reuse the policy decision without repeatedly asking AM for a new policy decision.

If WebSocket notifications are enabled, the cache evicts entries based on policy notifications from AM, making the cache content more accurate (trustable).

By default, policy decisions are not cached. To increase performance, consider enabling and configuring the cache. For more information, see PolicyEnforcementFilter(5) in the *Configuration Reference*.

User profile cache

When the UserProfileFilter retrieves user information, it caches it. IG can reuse the cached data without repeatedly querying AM to retrieve it.

By default, profile attributes are not cached. To increase performance, consider enabling and configuring the cache. For more information, see UserProfileFilter(5) in the *Configuration Reference*.

Access token cache

After a user presents an access_token to the OAuth2ResourceServerFilter, this cache stores the token. IG can reuse the token information without repeatedly asking the authorization server to verify the access_token for each request.

By default, access_tokens are not cached. To increase performance by caching access_tokens, consider configuring the `cache` property of OAuth2ResourceServerFilter. For more information, see OAuth2ResourceServerFilter(5) in the *Configuration Reference*.

Open ID Connect user information cache

When a downstream filter or handler needs user information from an OpenID Connect provider, IG fetches it lazily. By default, IG caches the information for 10 minutes to prevent repeated calls over a short time.

For more information, see `cacheExpiration` in OAuth2ClientFilter(5) in the *Configuration Reference*.

All caches provide similar configuration properties for timeout, defining the duration to cache entries. When the timeout is lower, the cache is evicted more frequently, and consequently, the performance is lower but the trust in results is higher. Consider your requirements for performance and security when you configure the timeout properties for each cache.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

21.2.4. WebSocket Notifications

By default, IG receives WebSocket notifications from AM for the following events:

- When a user logs out of AM, or when the AM session is modified, closed, or times out. IG can use WebSocket notifications to evict entries from the session cache.

  For an example of setting up session cache eviction, see "Using WebSocket Notifications to Evict the Session Cache".

- When AM creates, deletes, or changes a policy decision. IG can use WebSocket notifications to evict entries from the policy cache.

  For an example of setting up cache eviction, see "Using WebSocket Notifications to Evict the Session Cache".

To disable WebSocket notifications, or change any of the parameters, configure the `notifications` property in AmService. For information, see AmService(5) in the *Configuration Reference*.

If the WebSocket connection is lost for a time, notifications that occur during that time are lost. If a session ends or a policy decision changes while the WebSocket connection is lost, IG is not notified, and can continue to operate on outdated data.

By default, IG waits for five seconds before trying to re-establish the connection. If it can't re-establish the connection, it keeps trying every five seconds.

21.3. Tuning the ClientHandler/ReverseProxyHandler

The ClientHandler/ReverseProxyHandler communicates as a client to a downstream third-party service or protected application. The performance of the communication is determined by the following parameters:

- The number of available connections to the downstream service or application.
- Number of IG worker threads allocated to service inbound requests, and manage propagation to the downstream service or application.
- The connection timeout, or maximum time to connect to a server-side socket, before timing out and abandoning the connection attempt.
- The socket timeout, or the maximum time a request is expected to take before a response is received, after which the request is deemed to have failed.

Configure IG in conjunction with the Tomcat container, as follows:

- For BIO Connector (Tomcat 3.x to 8.x), configure `maxThreads` in Tomcat to be close to the number of configured Tomcat connections.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Because IG uses an asynchronous threading model, the `numberOfWorkers` in ClientHandler/ReverseProxyHandler can be much lower. The asynchronous threads are freed up immediately after the request is propagated, and can service another blocking Tomcat request thread.

To take advantage of IG's asynchronous thread model, configure Tomcat to use a non-blocking, NIO or NIO2 connector, instead of a BIO connector.

- For NIO connectors, align `numberOfWorkers` in IG with `maxThreads` in Tomcat.

  Because NIO connectors use an asynchronous threading model, the `maxThreads` in Tomcat can be much lower than for a BIO connector.

To identify the throughput plateau, test in a pre-production performance environment, with realistic use cases. Increment `numberOfWorkers` from its default value of one thread per JVM core, up to a large maximum value based on the number of concurrent connections.

21.4. Tuning IG's Tomcat Container

Configure the Tomcat container in conjunction with IG, as described in "Tuning the ClientHandler/ReverseProxyHandler".

To take advantage of IG's asynchronous thread model, configure Tomcat to use a non-blocking, NIO or NIO2 connector. Consider configuring the following connector attributes:

- `maxConnections`
- `connectionTimeout`
- `soTimeout`
- `acceptCount`
- `executor`
- `maxThreads`
- `minSpareThreads`

For more information, see Apache Tomcat 9 Configuration Reference and Apache Tomcat 8 Configuration Reference.

21.5. Tuning IG's JVM

Start tuning the JVM with default values, and monitor the execution, paying particular attention to memory consumption, and GC collection time and frequency. Incrementally adjust the configuration, and retest to find the best settings for memory and garbage collection.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Make sure that there is enough memory to accommodate the peak number of required connections, and make sure that timeouts in IG and its container support latency in downstream servers and applications.

IG makes low memory demands, and consumes mostly YoungGen memory. However, using caches, or proxying large resources, increases the consumption of OldGen memory. For information about how to optimize JVM memory, see the Oracle documentation.

Consider these points when choosing a JVM:

- Find out which version of the JVM is available. More recent JVMs usually contain performance improvements, especially for garbage collection.
- Choose a 64-bit JVM if you need to maximize available memory.

Consider these points when choosing a GC:

- Test GCs in realistic scenarios, and load them into a pre-production environment.
- Choose a GC that is adapted to your requirements and limitations. Compare the following GCs in typical business use cases:
  - Concurrent-Mark-Sweep garbage collector (CMS)
  - Garbage-First Collector (G1)

    The G1 is targeted for multi-processor environments with large memories. It provides good overall performance without the need for additional options. The G1 is designed to reduce garbage collection, through low-GC latency. It is largely self-tuning, with an adaptive optimization algorithm. For more information, see Garbage-First Garbage Collector and Garbage-First Garbage Collector Tuning
  - Parallel GC

    The Parallel GC aims to improve garbage collection by following a high-throughput strategy, but it requires more full garbage collections. For more information, see Available Collectors.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Chapter 22 Troubleshooting

This chapter covers common problems and their solutions.

Get the product version and build information for the running instance of IG from the `/api/info` endpoint. If IG is set up as described in "*First Steps*" in the *Getting Started Guide*, access the information at http://openig.example.com:8080/openig/api/info.

22.1. Timeout When Downloading Large Files

If `SocketTimeoutException` errors occur in the logs when you try to download large files, in your ReverseProxyHandler or ClientHandler, increase `soTimeout` and set `asyncBehavior` to `streaming`.

22.2. Requests Redirected to Access Management Instead of to the Resource

By default, ForgeRock Access Management 5 and later writes cookies to the fully qualified domain name of the server, for example, `openam.example.com`. Therefore a host-based cookie rather than a domain-based cookie is set.

Consequently, after authentication through Access Management, requests can be redirected to Access Management instead of to the resource.

To resolve this issue, add a cookie domain to the Access Management configuration. For example, in the Access Management console, go to Configure > Global Services > Platform, and add the domain `example.com`.

22.3. Troubleshooting the UMA Example

You have set up and are testing the example in "About IG As an UMA Resource Server".

If you have problems creating shares for Alice, perform the following steps to see if you can get a PAT from AM:

1. With AM running, run the following command to authenticate Alice to AM:

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
$ mytoken=$(curl --request POST
 \
--header "Accept-API-Version: resource=2.1"
 \
--header "X-OpenAM-Username: alice"
 \
--header "X-OpenAM-Password: UMAexample"
 \
--header "Content-Type: application/json"
 \
--data "{}"  \
http://openam.example.com:8088/openam/json/authenticate)
```

2. Run the following command to get an access token for Alice:

```
$ curl -X POST
 \
-H "Cache-Control: no-cache"
 \
-H "Cookie: iPlanetDirectoryPro=${mytoken}"
 \
-H "Content-Type: application/x-www-form-urlencoded"
 \
-d
 'grant_type=password&scope=uma_protection&username=alice&password=UMAexample&client_id=OpenIG&client_secret=pa
 \
http://openam.example.com:8088/openam/oauth2/access_token

{"access_token":"AQIC5wM2LY . . . Dg5AAJTMQAA*","scope":"uma_protection","token_type":"Bearer"
,"expires_in":3599}
```

If you fail to get an access token, check that AM is configured as described in "Setting Up AM As an Authorization Server".

If you continue to have problems, make sure that your IG configuration matches that shown when you are running the test on http://app.example.com:8081/uma/.

22.4. Can't Deploy Routes in Studio

Studio deploys and undeploys routes through a main router named `_router`, which is the name of the main router in the default configuration. If you use a custom `config.json`, make sure that it contains a main router named `_router`.

For information about Studio, see "Creating Routes Through Studio ".

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

22.5. Object not found in heap

```
org.forgerock.json.fluent.JsonValueException: /handler:
   object Router2 not found in heap
    at org.forgerock.openig.heap.HeapImpl.resolve(HeapImpl.java:351)
    at org.forgerock.openig.heap.HeapImpl.resolve(HeapImpl.java:334)
    at org.forgerock.openig.heap.HeapImpl.getHandler(HeapImpl.java:538)
```

You have specified `"handler": "Router2"` in `config.json`, but no handler configuration object named Router2 exists. Make sure you have added an entry for the handler and that you have correctly spelled its name.

22.6. Extra or missing character / invalid JSON

When the JSON for a route is not valid, IG does not load the route. Instead, a description of the error appears in the log:

```
16:09:50 | ERROR | openig.example.com-startStop-1 | o.f.o.h.r.RouterHandler |
The file '/Users/me/.openig/config/routes/zz-default.json' is not a valid route configuration.
```

Use a JSON editor or JSON validation tool such as JSONLint to make sure that your JSON is valid.

22.7. The values in the flat file are incorrect

Ensure the flat file is readable by the user running the container for IG. Values are all characters including space and tabs between the separator, so make sure the values are not padded with spaces.

22.8. Problem accessing URL

```
HTTP ERROR 500

Problem accessing /myURL . Reason:

java.lang.String cannot be cast to java.util.List
Caused by:
java.lang.ClassCastException: java.lang.String cannot be cast to java.util.List
```

This error is typically encountered when using an `AssignmentFilter` as described in AssignmentFilter(5) in the *Configuration Reference* and setting a string value for one of the headers. All headers are stored in lists so the header must be addressed with a subscript.

For example, rather than trying to set `request.headers['Location']` for a redirect in the response object, you should instead set `request.headers['Location'][0]`. A header without a subscript leads to the error above.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

22.9. StaticResponseHandler results in a blank page

Define an entity for the response as in the following example:

```
{
    "name": "AccessDeniedHandler",
    "type": "StaticResponseHandler",
    "config": {
        "status": 403,
        "reason": "Forbidden",
        "entity": "<html><body><p>User does not have permission</p></body></html>"
    }
}
```

22.10. IG is not logging users in

If you are proxying to more than one application in multiple DNS domains, you must make sure your container is enabled for domain cookies. For details on your specific container, see "Configuring Deployment Containers".

22.11. Read timed out error when sending a request

If a baseURI configuration setting causes a request to come back to IG, IG never produces a response to the request. You then observe the following behavior.

You send a request and IG seems to hang. Then you see a failure message, `HTTP Status 500 - Read timed out`, accompanied by IG throwing an exception, `java.net.SocketTimeoutException: Read timed out`.

To fix this issue, make sure that `baseURI` configuration settings use a different host and port than the host and port for IG.

22.12. IG does not use new route configuration

IG loads all configuration at startup. By default, it then periodically reloads changed route configurations.

If you make changes to a route that result in an invalid configuration, IG logs errors, but it keeps the previous, correct configuration, and continues to use the old route.

IG only uses the new configuration after you save a valid version or when you restart IG.

Of course, if you restart IG with an invalid route configuration, then IG tries to load the invalid route at startup and logs an error. In that case, if there is no default handler to accept any incoming request for the invalid route, then you see an error, `No handler to dispatch to`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

22.13. Make IG skip a route

If you have copied routes from another IG server, those routes might depend on environment or container configuration that you have not yet configured locally.

You can work around this problem by changing the route file extension. A router ignores route files that do not have the `.json` extension.

For example, suppose you copy route all sample route configurations from the documentation, and then start IG without first configuring your container. This can result in an error such as the following:

```
/handler/config/filters/0/config/dataSource: javax.naming.NameNotFoundException;
    remaining name 'jdbc/forgerock'
[       JsonValueException] > /handler/config/filters/0/config/dataSource:
    javax.naming.NameNotFoundException; remaining name 'jdbc/forgerock'
[    NameNotFoundException] > null

org.forgerock.json.fluent.JsonValueException:
    /handler/config/filters/0/config/dataSource:
    javax.naming.NameNotFoundException; remaining name 'jdbc/forgerock'
 at org.forgerock.openig.filter.SqlAttributesFilter$Heaplet.create(
    SqlAttributesFilter.java:211)
 at org.forgerock.openig.heap.GenericHeaplet.create(GenericHeaplet.java:81)
 at org.forgerock.openig.heap.HeapImpl.extract(HeapImpl.java:316)
 at org.forgerock.openig.heap.HeapImpl.get(HeapImpl.java:281)
...
```

This arises from the route in `03-sql.json`, which defines an SqlAttributesFilter that depends on a JNDI data source configured in the container:

```
{
    "type": "SqlAttributesFilter",
    "config": {
        "dataSource": "java:comp/env/jdbc/forgerock",
        "preparedStatement":
          "SELECT username, password FROM users WHERE email = ?;",
        "parameters": [
            "george@example.com"
        ],
        "target": "${attributes.sql}"
    }
}
```

To prevent IG from loading the route configuration until you have had time to configure the container, change the file extension to render the route inactive:

```
$ mv ~/.openig/config/routes/03-sql.json ~/.openig/config/routes/03-sql.inactive
```

If necessary, restart the container to force IG to reload the configuration.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

When you have configured the data source in the container, change the file extension back to `.json` to render the route active again:

```
$ mv ~/.openig/config/routes/03-sql.inactive ~/.openig/config/routes/03-sql.json
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Appendix A. SAML 2.0 and Multiple Applications

"*Acting As a SAML 2.0 Service Provider*" describes how to set up IG as a SAML 2.0 service provider for a single application, using AM as the identity provider. This chapter describes how to set up IG as a SAML 2.0 service provider for two applications, still using AM as the identity provider.

Before you try the samples described here, familiarize yourself with IG SAML 2.0 support by reading and working through the examples in "*Acting As a SAML 2.0 Service Provider*". Before you start, you should have IG protecting the sample application as a SAML 2.0 service provider, with AM working as identity provider configured as described in that tutorial.

A.1. Installation Overview

In this chapter you use the Fedlet configuration from "*Acting As a SAML 2.0 Service Provider*" to create a configuration for each new protected application. You then import the new configurations as SAML 2.0 entities in AM. If you subsequently edit a configuration, import it again.

In the following examples, the first application has entity ID `sp1` and runs on the host `sp1.example.com`, the second application has entity ID `sp2` and runs on the host `sp2.example.com`. To prevent unwanted behavior, the applications must have different values.

Tasks for Configuring SAML 2.0 SSO and Federation

Task	See Section(s)
Prepare the network.	"Preparing the Network"

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

Task	See Section(s)
Prepare the configuration for two IG service providers.	"Configuring the Circle of Trust" "Configuring the Service Provider for Application One" "Configuring the Service Provider for Application Two"
Import the service provider configurations into AM.	"Importing Service Provider Configurations Into AM"
Add IG routes.	"Preparing the Base Configuration File" "Preparing Routes for Application One" "Preparing Routes for Application Two"

A.2. Preparing the Network

Configure the network so that browser traffic to the application hosts is proxied through IG.

Add `sp1.example.com` and `sp2.example.com` to your `/etc/hosts` file:

```
127.0.0.1    localhost openam.example.com openig.example.com app.example.com sp1.example.com sp2.example
.com
```

A.3. Configuring the Circle of Trust

Edit the `$HOME/.openig/SAML/fedlet.cot` file created in "*Acting As a SAML 2.0 Service Provider*" to include the entity IDs `sp1` and `sp2`:

```
cot-name=Circle of Trust
sun-fm-cot-status=Active
sun-fm-trusted-providers=openam,sp1,sp2
sun-fm-saml2-readerservice-url=
sun-fm-saml2-writerservice-url=
```

A.4. Configuring the Service Provider for Application One

To configure the service provider for application one, you can use the example files "Configuration File for Application One" and "Extended Configuration File for Application One", saving them as `sp1.xml` and `sp1-extended.xml`. Alternatively, follow the steps below to use the files you created in "*Acting As a SAML 2.0 Service Provider*".

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To Configure the Service Provider for Application One By Using Files Created In "Acting As a SAML 2.0 Service Provider"

1. Copy the SAML configuration files `sp.xml` and `sp-extended.xml` you created in "*Acting As a SAML 2.0 Service Provider*", and save them as `sp1.xml` and `sp1-extended.xml`.
2. Make the following changes in `sp1.xml`:
   - For `entityID`, change `sp` to `sp1`. The `entityID` must match the application.
   - On each line that starts with `Location` or `ResponseLocation`, change `sp.example.com` to `sp1.example.com`, and add `/metaAlias/sp1` at the end of the line.

   For an example of how this file should be, see "Configuration File for Application One".
3. Make the following changes in `sp1-extended.xml`:
   - For `entityID`, change `sp` to `sp1`.
   - For `SPSSOConfig metaAlias`, change `sp` to `sp1`.
   - For `appLogoutUrl`, change `sp` to `sp1`.
   - For `hosted=`, make sure that the value is `1`.

   For an example of how this file should be, see "Extended Configuration File for Application One".

Configuration File for Application One

```
<!--
  - sp1-xml.txt
  - Set the entityID
  - Set metaAlias/<sp-name> at the end of each of the following lines:
     - Location
     - ResponseLocation
  - Note that AssertionConsumerService Location attributes include the metaAlias.
-->
<EntityDescriptor
  entityID="sp1"
  xmlns="urn:oasis:names:tc:SAML:2.0:metadata">
  <SPSSODescriptor
    AuthnRequestsSigned="false"
    WantAssertionsSigned="false"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <SingleLogoutService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
      Location="http://sp1.example.com:8080/saml/fedletSloRedirect/metaAlias/sp1"
      ResponseLocation="http://sp1.example.com:8080/saml/fedletSloRedirect/metaAlias/sp1"/>
    <SingleLogoutService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
      Location="http://sp1.example.com:8080/saml/fedletSloPOST/metaAlias/sp1"
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
      ResponseLocation="http://sp1.example.com:8080/saml/fedletSloPOST/metaAlias/sp1"/>
    <SingleLogoutService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
      Location="http://sp1.example.com:8080/saml/fedletSloSoap/metaAlias/sp1"/>
    <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:transient</NameIDFormat>
    <AssertionConsumerService
      isDefault="true"
      index="0"
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
      Location="http://sp1.example.com:8080/saml/fedletapplication/metaAlias/sp1"/>
    <AssertionConsumerService
      index="1"
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"
      Location="http://sp1.example.com:8080/saml/fedletapplication/metaAlias/sp1"/>
  </SPSSODescriptor>
  <RoleDescriptor
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:query="urn:oasis:names:tc:SAML:metadata:ext:query"
    xsi:type="query:AttributeQueryDescriptorType"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
  </RoleDescriptor>
  <XACMLAuthzDecisionQueryDescriptor
    WantAssertionsSigned="false"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
  </XACMLAuthzDecisionQueryDescriptor>
</EntityDescriptor>
```

Extended Configuration File for Application One

```
<!--
  - sp1-extended.xml
  - Set the entityID.
  - Set the SPSSOConfig metaAlias attribute.
  - Set the value of appLogoutUrl.
  - Set the value of hosted to 1.
  - Comment out the attribute "com.sun.identity.saml2.plugins.DefaultFedletAdapter".
-->
<EntityConfig xmlns="urn:sun:fm:SAML:2.0:entityconfig"
    xmlns:fm="urn:sun:fm:SAML:2.0:entityconfig"
    hosted="1"
    entityID="sp1">

    <SPSSOConfig metaAlias="/sp1">
        <Attribute name="description">
            <Value></Value>
        </Attribute>
        <Attribute name="signingCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="encryptionCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="basicAuthOn">
            <Value>false</Value>
        </Attribute>
        <Attribute name="basicAuthUser">
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
            <Value></Value>
        </Attribute>
        <Attribute name="basicAuthPassword">
            <Value></Value>
        </Attribute>
        <Attribute name="autofedEnabled">
            <Value>false</Value>
        </Attribute>
        <Attribute name="autofedAttribute">
            <Value></Value>
        </Attribute>
        <Attribute name="transientUser">
            <Value>anonymous</Value>
        </Attribute>
        <Attribute name="spAdapter">
            <Value></Value>
        </Attribute>
        <Attribute name="spAdapterEnv">
            <Value></Value>
        </Attribute>
        <!--
        <Attribute name="fedletAdapter">
            <Value>com.sun.identity.saml2.plugins.DefaultFedletAdapter</Value>
        </Attribute>
        -->
        <Attribute name="fedletAdapterEnv">
            <Value></Value>
        </Attribute>
        <Attribute name="spAccountMapper">
            <Value>com.sun.identity.saml2.plugins.DefaultLibrarySPAccountMapper</Value>
        </Attribute>
        <Attribute name="useNameIDAsSPUserID">
            <Value>false</Value>
        </Attribute>
        <Attribute name="spAttributeMapper">
            <Value>com.sun.identity.saml2.plugins.DefaultSPAttributeMapper</Value>
        </Attribute>
        <Attribute name="spAuthncontextMapper">
            <Value>com.sun.identity.saml2.plugins.DefaultSPAuthnContextMapper</Value>
        </Attribute>
        <Attribute name="spAuthncontextClassrefMapping">
            <Value>
            urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport|0|default
            </Value>
        </Attribute>
        <Attribute name="spAuthncontextComparisonType">
           <Value>exact</Value>
        </Attribute>
        <Attribute name="attributeMap">
           <Value>cn=cn</Value>
           <Value>sn=sn</Value>
        </Attribute>
        <Attribute name="saml2AuthModuleName">
           <Value></Value>
       </Attribute>
       <Attribute name="localAuthURL">
           <Value></Value>
       </Attribute>
       <Attribute name="intermediateUrl">
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
            <Value></Value>
        </Attribute>
        <Attribute name="defaultRelayState">
            <Value></Value>
        </Attribute>
        <Attribute name="appLogoutUrl">
            <Value>http://sp1.example.com:8080/saml/logout</Value>
        </Attribute>
        <Attribute name="assertionTimeSkew">
            <Value>300</Value>
        </Attribute>
        <Attribute name="wantAttributeEncrypted">
            <Value></Value>
        </Attribute>
        <Attribute name="wantAssertionEncrypted">
            <Value></Value>
        </Attribute>
        <Attribute name="wantNameIDEncrypted">
            <Value></Value>
        </Attribute>
        <Attribute name="wantPOSTResponseSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="wantArtifactResponseSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="wantLogoutRequestSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="wantLogoutResponseSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="wantMNIRequestSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="wantMNIResponseSigned">
            <Value></Value>
        </Attribute>
        <Attribute name="responseArtifactMessageEncoding">
            <Value>URI</Value>
        </Attribute>
        <Attribute name="cotlist">
        <Value>Circle of Trust</Value></Attribute>
        <Attribute name="saeAppSecretList">
        </Attribute>
        <Attribute name="saeSPUrl">
            <Value></Value>
        </Attribute>
        <Attribute name="saeSPLogoutUrl">
        </Attribute>
        <Attribute name="ECPRequestIDPListFinderImpl">
            <Value>com.sun.identity.saml2.plugins.ECPIDPFinder</Value>
        </Attribute>
        <Attribute name="ECPRequestIDPList">
            <Value></Value>
        </Attribute>
        <Attribute name="ECPRequestIDPListGetComplete">
            <Value></Value>
        </Attribute>
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
        <Attribute name="enableIDPProxy">
            <Value>false</Value>
        </Attribute>
        <Attribute name="idpProxyList">
            <Value></Value>
        </Attribute>
        <Attribute name="idpProxyCount">
            <Value>0</Value>
        </Attribute>
        <Attribute name="useIntroductionForIDPProxy">
            <Value>false</Value>
        </Attribute>
        <Attribute name="spSessionSyncEnabled">
            <Value>false</Value>
        </Attribute>
         <Attribute name="relayStateUrlList">
         </Attribute>
    </SPSSOConfig>
    <AttributeQueryConfig metaAlias="/attrQuery">
        <Attribute name="signingCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="encryptionCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="wantNameIDEncrypted">
            <Value></Value>
        </Attribute>
        <Attribute name="cotlist">
            <Value>Circle of Trust</Value>
        </Attribute>
    </AttributeQueryConfig>
    <XACMLAuthzDecisionQueryConfig metaAlias="/pep">
        <Attribute name="signingCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="encryptionCertAlias">
            <Value></Value>
        </Attribute>
        <Attribute name="basicAuthOn">
            <Value>false</Value>
        </Attribute>
        <Attribute name="basicAuthUser">
            <Value></Value>
        </Attribute>
        <Attribute name="basicAuthPassword">
            <Value></Value>
        </Attribute>
        <Attribute name="wantXACMLAuthzDecisionResponseSigned">
            <Value>false</Value>
        </Attribute>
        <Attribute name="wantAssertionEncrypted">
            <Value>false</Value>
        </Attribute>
        <Attribute name="cotlist">
            <Value>Circle of Trust</Value>
        </Attribute>
    </XACMLAuthzDecisionQueryConfig>
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
</EntityConfig>
```

A.5. Configuring the Service Provider for Application Two

To Configure the Service Provider for Application Two

1. Copy the SAML configuration files `sp1.xml` and `sp1-extended.xml` you created in "Configuring the Service Provider for Application One", and save them as `$HOME/.openig/SAML/sp2.xml` and `$HOME/.openig/SAML/sp2-extended.xml`.
2. In both files, replace all incidences of `sp1` with `sp2`. To prevent unwanted behavior. application two must have different values to application one.

A.6. Importing Service Provider Configurations Into AM

For each new protected application, import a SAML 2.0 entity into AM. If you subsequently edit a service provider configuration, import it again.

To Import the Service Provider Configurations Into AM

1. Log in to AM console as administrator.
2. Select Applications > WS-Fed, and click Import Entity.

   In some versions of AM, this option is on a Federation tab.
3. Import the entity provider:
   - For the metadata file, select File and upload `sp1.xml`.
   - For the extended data file, select File and upload `sp1-extended.xml`.
4. Repeat the previous steps to upload `sp2.xml` and `sp2-extended.xml` for `sp2`.
5. Log out of the AM console.

A.7. Preparing IG Configurations

For each new protected application, prepare an IG configuration. The configurations in this section follow the example in "*Acting As a SAML 2.0 Service Provider*".

> **Tip**
>
> To prevent unspecified behavior, use different keys for session-stored values in the routes for each application. For example, use different keys for `session.sp1Username` and `session.sp2Username`.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.

To prevent configurations from overwriting each others' data, use the `subjectMapping` property of SamlFederationHandler to define a different session field for the subject name of each application. The two applications must not map data into the same session field.

A.7.1. Preparing the Base Configuration File

Edit `config.json` to comment the baseURI in the top-level handler. The handler declaration appears as follows:

```
{
    "handler": {
        "type": "Router",
        "name": "_router",
        "_baseURI": "http://app.example.com:8081",
        "capture": "all"
    }
}
```

Restart IG for the changes to take effect.

A.7.2. Preparing Routes for Application One

Set up the following routes for application one:

- `$HOME/.openig/config/routes/05-federate-sp1.json`, to redirect the request for SAML authentication. After authentication, this route logs the user in to the application.
- `$HOME/.openig/config/routes/05-saml-sp1.json`, to map attributes from the SAML assertion into the context, and then redirect the request back to the first route.

`05-federate-sp1.json`

```
{
    "handler": {
        "type": "DispatchHandler",
        "config": {
            "bindings": [
                {
                    "condition": "${empty session.sp1Username}",
                    "handler": {
                        "type": "StaticResponseHandler",
                        "config": {
                            "status": 302,
                            "reason": "Found",
                            "headers": {
                                "Location": [
                                    "http://sp1.example.com:8080/saml/SPInitiatedSSO?metaAlias=/sp1"
                                ]
                            }
                        }
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
                    }
                },
                {
                    "handler": {
                        "type": "Chain",
                        "config": {
                            "filters": [
                                {
                                    "type": "StaticRequestFilter",
                                    "config": {
                                        "method": "POST",
                                        "uri": "http://app.example.com:8081/login",
                                        "form": {
                                            "username": [
                                                "${session.sp1Username[0]}"
                                            ],
                                            "password": [
                                                "${session.sp1Password[0]}"
                                            ]
                                        }
                                    }
                                }
                            ],
                            "handler": "ReverseProxyHandler"
                        }
                    }
                }
            ]
        }
    },
    "condition": "${matches(request.uri.host, 'sp1.example.com') and not matches(request.uri.path, '^/
saml')}"
}
```

05-saml-sp1.json

```
{
    "handler": {
        "type": "SamlFederationHandler",
        "config": {
            "comment": "Use unique session properties for this SP.",
            "assertionMapping": {
                "sp1Username": "cn",
                "sp1Password": "sn"
            },
            "authnContext": "sp1AuthnContext",
            "sessionIndexMapping": "sp1SessionIndex",
            "subjectMapping": "sp1SubjectName",
            "redirectURI": "/sp1"
        }
    },
    "condition": "${matches(request.uri.host, 'sp1.example.com') and matches(request.uri.path, '^/saml')}"
}
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

A.7.3. Preparing Routes for Application Two

Set up the following routes for application two:

- `$HOME/.openig/config/routes/05-federate-sp2.json`, to redirect the request for SAML authentication. After authentication, this route logs the user in to the application.
- `$HOME/.openig/config/routes/05-saml-sp2.json`, to map attributes from the SAML assertion into the context, and then redirect the request back to the first route.

05-federate-sp2.json

```
{
   "handler": {
       "type": "DispatchHandler",
       "config": {
           "bindings": [
               {
                   "condition": "${empty session.sp2Username}",
                   "handler": {
                       "type": "StaticResponseHandler",
                       "config": {
                           "status": 302,
                           "reason": "Found",
                           "headers": {
                               "Location": [
                                   "http://sp2.example.com:8080/saml/SPInitiatedSSO?metaAlias=/sp2"
                               ]
                           }
                       }
                   }
               },
               {
                   "handler": {
                       "type": "Chain",
                       "config": {
                           "filters": [
                               {
                                   "type": "StaticRequestFilter",
                                   "config": {
                                       "method": "POST",
                                       "uri": "http://app.example.com:8081/login",
                                       "form": {
                                           "username": [
                                               "${session.sp2Username[0]}"
                                           ],
                                           "password": [
                                               "${session.sp2Password[0]}"
                                           ]
                                       }
                                   }
                               }
                           ],
                           "handler": "ReverseProxyHandler"
                       }
```


Copyright © 2011-2021 ForgeRock AS. All rights reserved.

```
                    }
                }
            ]
        }
    },
    "condition": "${matches(request.uri.host, 'sp2.example.com') and not matches(request.uri.path, '^/
saml')}"
}
```

05-saml-sp2.json

```
 {
    "handler": {
        "type": "SamlFederationHandler",
        "config": {
            "comment": "Use unique session properties for this SP.",
            "assertionMapping": {
                "sp2Username": "cn",
                "sp2Password": "sn"
            },
            "authnContext": "sp2AuthnContext",
            "sessionIndexMapping": "sp2SessionIndex",
            "subjectMapping": "sp2SubjectName",
            "redirectURI": "/sp2"
        }
    },
    "condition": "${matches(request.uri.host, 'sp2.example.com') and matches(request.uri.path, '^/saml')}"
}
```

A.8. Test the Configuration

If you use the example configurations described in this chapter, try the SAML 2.0 web single sign-on profile with application one by selecting either of the following links and logging in to AM as user `george`, password `C0stanza`:

- The link for SP-initiated SSO.
- The link for IDP-initiated SSO.

Similarly, try the SAML 2.0 web single sign-on profile with application two by selecting either of the following links and logging in to AM as user `george`, password `C0stanza`:

- The link for SP-initiated SSO.
- The link for IDP-initiated SSO.

If you have not configured the examples exactly as shown in this guide, then adapt the SSO links accordingly.

Copyright © 2011-2021 ForgeRock AS. All rights reserved.