Course Exercises Guide

AAA, OAuth, and OIDC in IBM DataPower V7.5

Course code WE753 / ZE753 ERC 1.0

IBM Training

April 2017 edition

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Trademarks

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

DataPower®
DB™
Domino®
Lotus®
Rational®
Tivoli®
WebSphere®

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Social® is a trademark or registered trademark of TWC Product and Technology, LLC, an IBM Company.

Other product and service names might be trademarks of IBM or other companies.

Exercises description

FLY airline case study

The exercises in this course build upon a common case study: the FLY airline services. The services are composed of a Booking Service web service and a Baggage Service web service. The services are implemented as a BookingServiceBackend MPGW and a BaggageStatusMockService MPGW, both running within the FLYService domain.

The Booking Service has one operation: BookTravel. The SOAP request that is named BookingRequest contains billing details, payment card details, booking type, and the reservation code. The SOAP response is a BookingResponse, which contains the confirmation code and much of the original message. The endpoint is:
`http://<dp_internal_ip>:9080/BookingService/`

The Baggage Service has two operations:

- BaggageStatus. The SOAP request that is named BaggageStatusRequest contains the passenger's last name and their reference number. The SOAP response is BaggageStatusResponse, which contains the status of each bag that is attached to the passenger's reference number.
- BagInfo. The SOAP request that is named BagInfoRequest contains the ID number of the bag in question. The SOAP response is BagInfoResponse, which contains the status of the bag and which passenger it belongs to. The Baggage Service does not have a WSDL, and cannot be proxied by a web service proxy. The endpoint is:
  `http://<dp_internal_ip>:2068/BaggageService/`

Technically, the FLY airline services are self-contained MPGWs that mimic a web services back end that might be on WebSphere Application Server.

This application minimizes its dependencies on data sources by relying on data from a flat file, and allowance of read-only operations.

Exercises

This course includes the following exercises:

- **Exercise 1**: Configuring authentication and authorization in a service

  Import an MPGW that converts JSON requests to the web service requests that the back-end application supports. Update the MPGW to use a AAA policy that works with a AAA information file for authentication and authorization. Use SoapUI to test the configuration. Change the AAA policy to use an LDAP server. Test with SoapUI.

- **Exercise 2**: Defining a three-legged OAuth scenario that uses DataPower services

  Configure the OAuth client profile to represent the OAuth client to the OAuth authorization server. Create a web token service to act as the authorization server. Configure an MPGW and its AAA policy to act as the enforcement point and resource server. Configure the Node.js parameters to work as the OAuth client. Start the Node.js program to act as the OAuth client. Use a browser to access the Node.js program and the resource server.

- **Exercise 3**: Implementing an OIDC client

  Create a social login policy object to represent the OAuth client to the social login provider. Create a web token service to act as the social login provider. Create an MPGW to act as the OIDC client. Use a browser to sign in to the social login provider and access the back-end application.

Note

The lab exercises were written on DataPower V7.5.1.3 firmware. A consistent problem at this level is a failure of the “save configuration” operation in the WebGUI. If this failure happens to you, a workaround exists. Instead of clicking **Save configuration**, click **Review changes** instead. Scroll to the bottom of the Review Configuration Changes page and click **Save Config**.

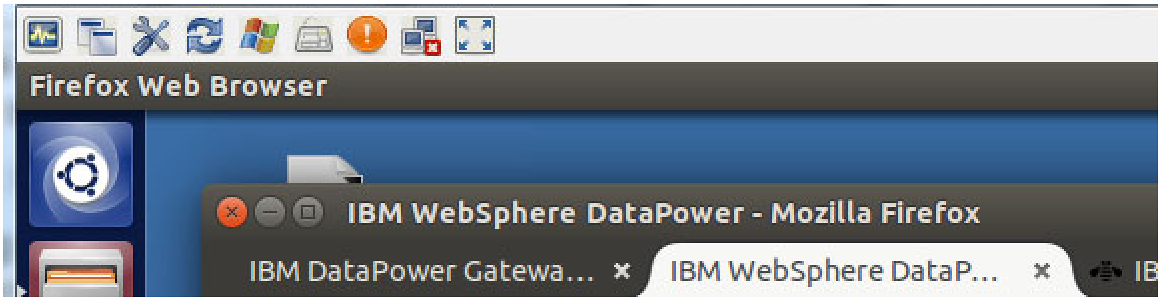
In the exercise instructions, you see that each step has a blank preceding it. You might want to check off each step as you complete it to track your progress.

Most exercises include required sections, which must always be completed. These exercises might be required before doing later exercises. Some exercises also include optional sections that you might want to do if you have sufficient time and want an extra challenge.

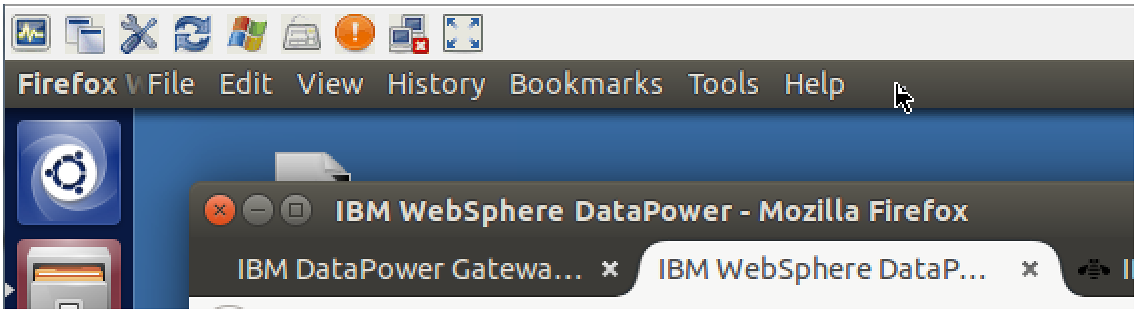
If you are using the IBM remote lab environment:

As of September 2016, the environment that is used to support the IBM-supplied images and DataPower gateways is Skytap. Each student is supplied an Ubuntu student image and a DataPower gateway.

- Ignore all offers to upgrade any of the software on the image. The image and exercise steps are designed to operate at the supplied levels of the contained software.
- The supplied image is Ubuntu 14.04 LTS. The desktop uses Unity, which is different than the more common Gnome desktop. Some hints on using Unity are at: http://www.howtogeek.com/113330/how-to-master-ubuntus-unity-desktop-8-things-you-need-to-know/
- A noticeable difference is that the menus on the windows within the desktop are not typically visible. When a window is the “active” window, that window does not have any menu items, but the application type is displayed in the black bar that spans the top of the desktop. In the following screen capture, observe that the browser window does not have a menu bar, and that its type of “Firefox Web Browser” is listed in the black bar.



If you hover the mouse over the black bar, the menu items for the active window are displayed.

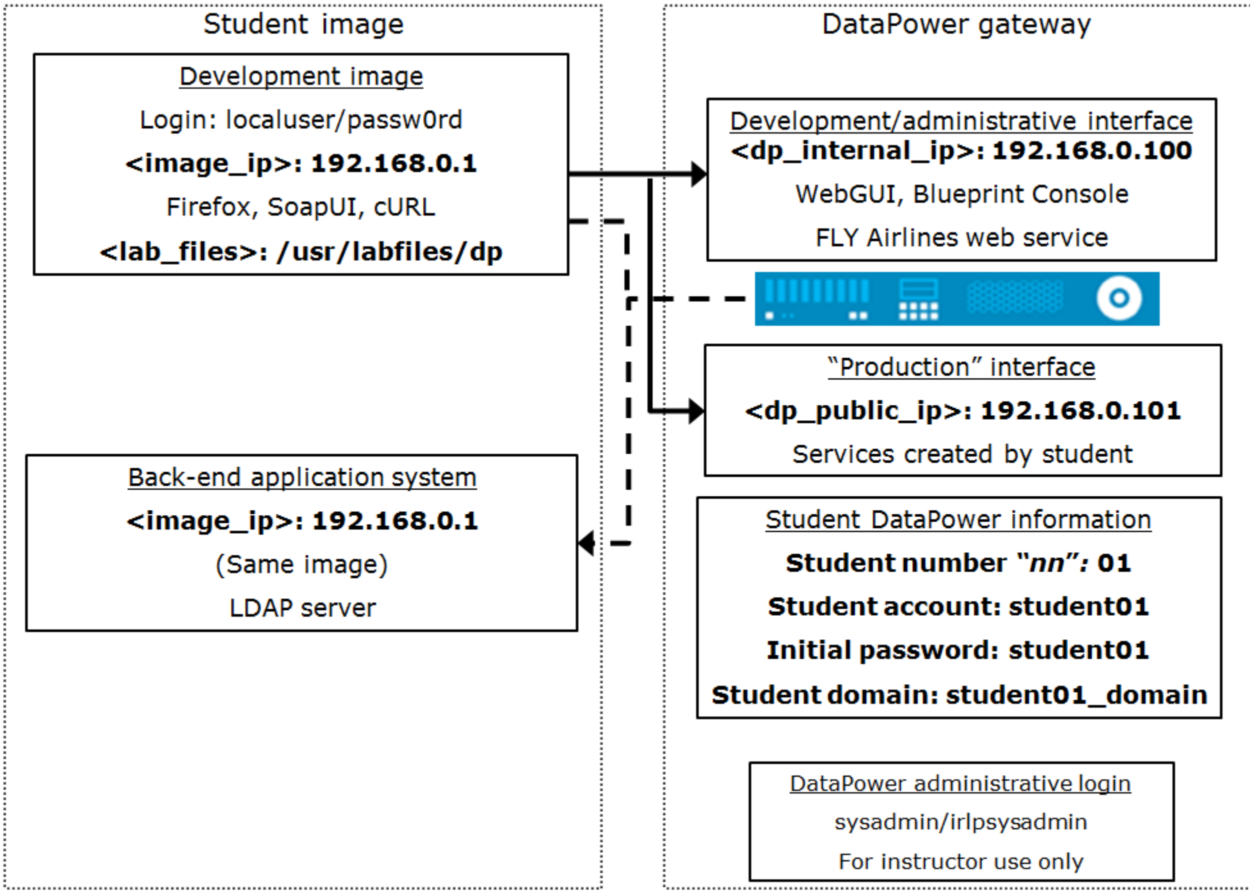


Another noticeable difference is that when a window is maximized, the Close, Minimize, and Restore buttons are not visible until you hover the mouse over the black bar.

This Unity behavior is discussed in the “Hidden Global Menus” section of the previously mentioned howtogeek.com page.

- The IBM supplied environment has pre-assigned values for some of the variables, such as the IP addresses of the student image and the DataPower gateway, the student number, and the initial password. The following graphic shows those assignments.

Variable values for DataPower courses on IBM/Skytap



Student image

Development image

Login: localuser/passw0rd

<image_ip>: 192.168.0.1

Firefox, SoapUI, cURL

<lab_files>: /usr/labfiles/dp

Back-end application system

<image_ip>: 192.168.0.1

(Same image)

LDAP server

DataPower gateway

Development/administrative interface

<dp_internal_ip>: 192.168.0.100

WebGUI, Blueprint Console

FLY Airlines web service

“Production” interface

<dp_public_ip>: 192.168.0.101

Services created by student

Student DataPower information

Student number “*nn*”: 01

Student account: student01

Initial password: student01

Student domain: student01_domain

DataPower administrative login

sysadmin/irlpsysadmin

For instructor use only

Important

Online course material updates might exist for this course. To check for updates, see the Instructor wiki at http://ibm.biz/CloudEduCourses.

Exercise 1. Configuring authentication and authorization in a service

Estimated time

00:45

Overview

This exercise covers the AAA capabilities of the IBM DataPower Gateway. To enforce client authentication and authorization, access to services is restricted to permitted clients for authorized operations.

Objectives

After completing this exercise, you should be able to:

- Configure a AAA action to enforce authentication and authorization policies that are in a AAA information file
- Configure a AAA action to enforce authentication and authorization policies that are in an LDAP server

Introduction

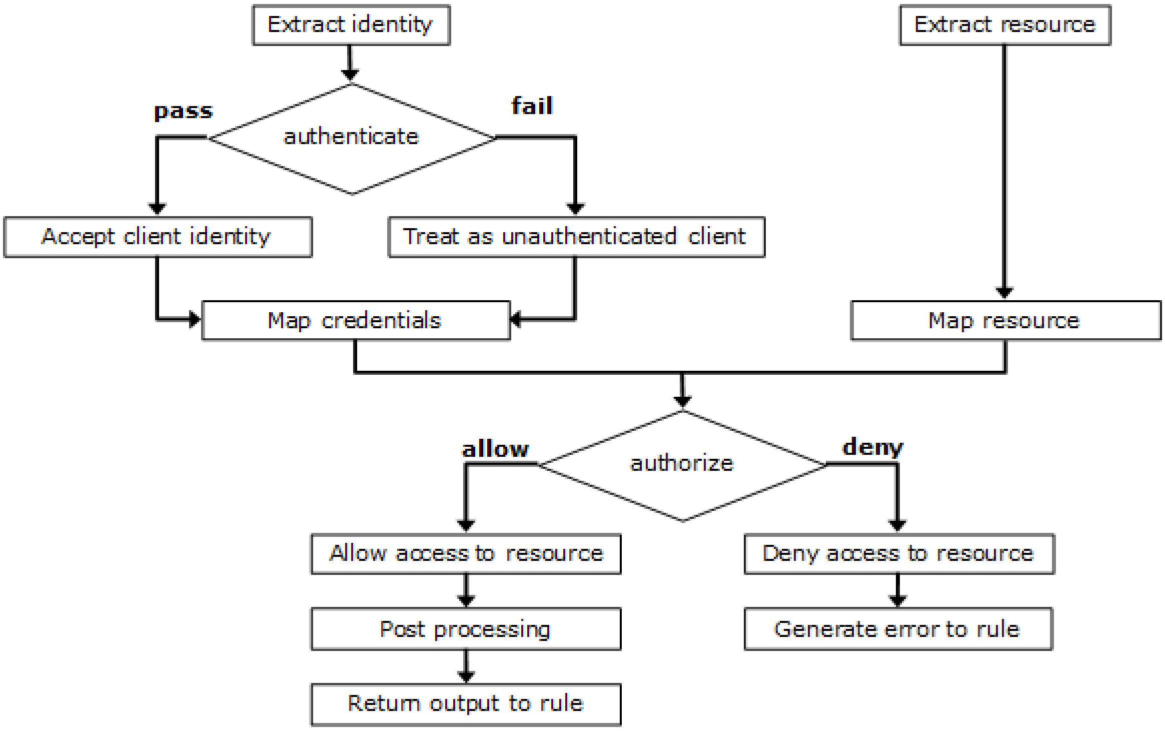
It is a common security practice to restrict access to back-end resources to specific clients. The “authentication” part of AAA controls authentication of the clients. Different clients might have access to different sets of back-end resources. The “authorization” part of AAA controls permission of a client to the requested resource. In this exercise, you are introduced to the access control framework that provides authentication, authorization, and audit services. Collectively, this framework is referred to as AAA.

A AAA policy identifies a set of resources and procedures that are used to determine whether a requesting client is granted access to a specific service, file, or document. AAA policies are thus filters in that they accept or deny a specific client request.

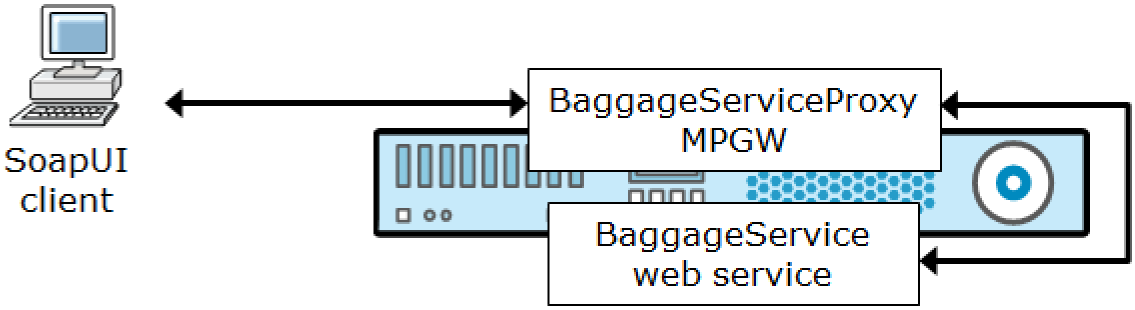
With a AAA policy, DataPower clients supply user credentials in the request message. As a policy enforcement point, the AAA policy verifies the user credentials and determines whether the user is authorized to access the requested operation.

Finally, the AAA policy provides logging and auditing features. Certain company policies, and existing laws, might mandate that the system track any access to private customer information.

This graphic reviews the AAA flow:



For this exercise, you import the BaggageServiceProxy MPGW. This service receives REST requests, converts them to SOAP requests, and passes the request to a back-end web service. The SOAP response from the web service gets converted to a REST response, and is returned to the client. You add a AAA action to the existing service policy. The topology of the exercise is provided here:



Requirements

To complete this exercise, you need:

- Access to the DataPower gateway
- SoapUI, to send requests to the DataPower gateway
- The BaggageStatusMockService web service that runs on the DataPower gateway in the `FLYServices` domain
- Access to the *<lab_files>* directory

Exercise instructions

Preface

- Remember to use the domain and port address that you were assigned in the exercise setup. *Do not* use the default domain.
- The references in exercise instructions refer to the following values:
  - *<lab_files>*: Location of the student lab files. Default location is: `/usr/labfiles/dp/`
  - <image_ip>: IP address of the student image (use `/sbin/ifconfig` from a terminal window to obtain value).
  - *<dp_internal_ip>*: IP address of the DataPower gateway development and administrative functions that are used by internal resources such as developers.
  - *<dp_public_ip>*: IP address of the public services on the gateway that is used by customer and clients.
  - *<dp_WebGUI_port>*: Port of the WebGUI. The default port is `9090`.
  - *<nn>*: Assigned student number. If there is no instructor, use “`01`”.
  - *<studentnn>*: Assigned user name and user account. If there is no instructor, use “`student01`”.
  - *<studentnn_password>*: Account password. In most cases, the initial value is the same as the user name. You are prompted to create a password on first use. Write it down.
  - *<studentnn_domain>*: Application domain that the user account is assigned to. If there is no instructor, use “`student01_domain`”.
  - *<FLY_baggage_port>*: Port number that the back-end BaggageServices web services listen on. The default port is 2068.
  - *<mpgw_baggage_port>*: `12`*`nn9`*, where “*nn*” is the two-digit student number. This port number is the listener port of the BaggageServiceProxy that mediates between the REST or JSON client and the Baggage Services back-end application.

1.1. Initialize the lab environment

Some setup activities are required to properly configure the lab environment and determine IP addresses and ports.

__ 1. Go to **Appendix B: Lab environment setup**. Complete those activities before proceeding.

These activities need to be performed only once for the course.

1.2. Import the BaggageServiceProxy service

If you do not have the BaggageServiceProxy MPGW already defined in your domain from the prerequisite Essentials course, then you need to import this service as your starting point.

__ 1. Use the WebGUI to log in to your domain.

__ a. Connect to the DataPower gateway: `https://<dp_internal_ip>:<dp_WebGUI_port>`

__ b. Log in with your student account `<studentnn>` and password `<studentnn_password>`, and select your student domain `<studentnn_domain>`.

__ 2. Verify that the host aliases are defined.

__ a. Switch to the default domain. You should have “read” access to this domain.

__ b. Open **Network > Interface > Host Alias**.

__ c. Verify that the **dp_internal_ip** alias refers to the `<dp_internal_ip>` address.

__ d. Verify that the **dp_public_ip** alias refers to the `<dp_public_ip>` address.

__ e. Switch back to your student domain.

__ 3. Use **Administration > Configuration > Import Configuration** to import `<lab_files>/AAA/RESTJOSE_REST_BaggageServiceProxy.zip` into your domain.

__ 4. Update the MPGW to use your student variables.

__ a. Edit the BaggageServiceProxy MPGW.

__ b. Verify that the **Default Backend URL** points to `http://dp_internal_ip:2068/BaggageService`. This URL is the Baggage Service web service back end that runs in the FLYServices domain.

__ c. Edit the **http_fsh_Baggage_12nn9** front side handler.

__ d. Set the **Port** to *<mpgw_baggage_port>*.

__ e. Click **Apply** to save the front side handler configuration.

__ f. Click **Apply** to save the MPGW configuration if needed.

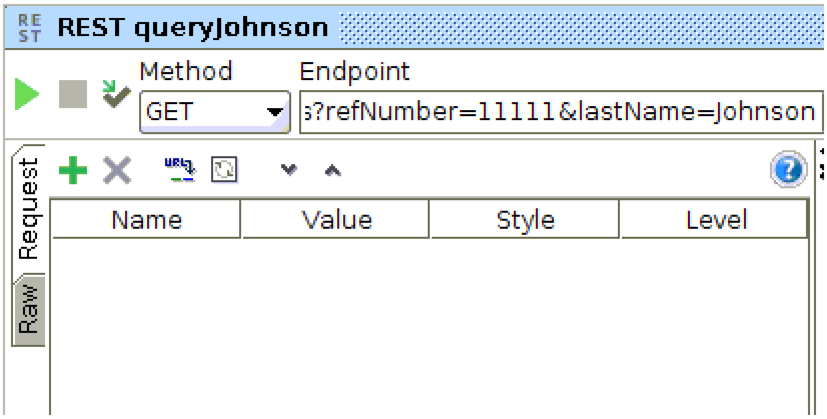
__ g. Save the configuration.

__ 5. Use SoapUI to test the service.

__ a. Open SoapUI.

__ b. Expand **BaggageServices > Baggage > Baggage REST GET**. Double-click **REST queryJohnson**.

__ c. Observe that it is an HTTP GET request that passes in a query parameter list of “refNumber=11111&lastName=Johnson”. No HTTP body is passed.



REST queryJohnson

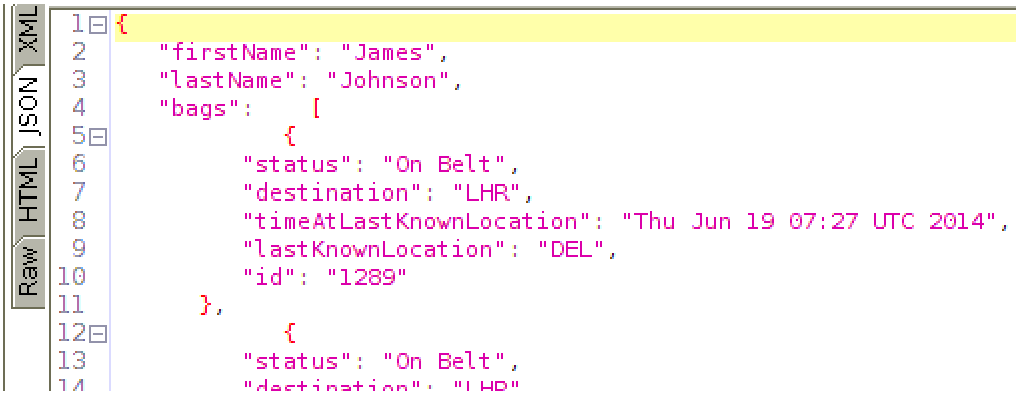
Method: GET

Endpoint: s?refNumber=11111&lastName=Johnson

Name	Value	Style	Level

__ d. Click the green **Submit** arrow.

__ e. The panel repaints and the response pane gets larger. Click the **JSON** tab. The JSON response for the Johnson bags is displayed.



```
{
    "firstName": "James",
    "lastName": "Johnson",
    "bags":    [
          {
          "status": "On Belt",
          "destination": "LHR",
          "timeAtLastKnownLocation": "Thu Jun 19 07:27 UTC 2014",
          "lastKnownLocation": "DEL",
          "id": "1289"
       },
          {
          "status": "On Belt",
          "destination": "LHR"
```

__ f. The base MPGW is set up. You can leave SoapUI open for later use.

1.3. Configure BaggageServiceProxy for authentication by using a AAA action

In this first section, you add a AAA action to the existing request rule that uses an HTTP authentication header and a `AAAInfo.xml` file to authenticate a client.

__ 1. Log in to the DataPower gateway WebGUI if you have logged out. Ensure that your are in your student domain.

__ 2. Click the **Multi-Protocol Gateway** icon.

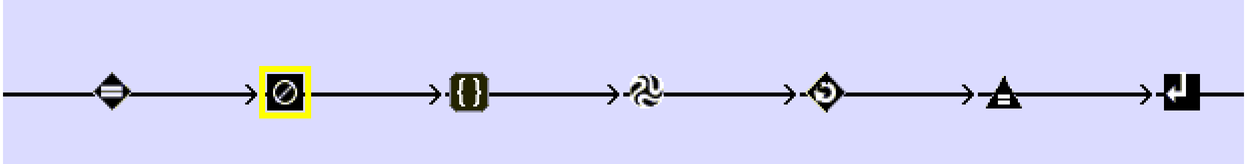
__ 3. Edit the `BaggageServiceProxy` multi-protocol gateway in your domain.

__ 4. Edit the `BaggageServicePolicy` service policy.

__ 5. The “queryJohnson” request invokes the **BaggageServicePolicy_BagsByPassenger_Req** rule. Select that rule in the Configured Rule section of the policy editor.

__ 6. Configure the request rule to use a AAA action as the first processing action in the rule.

__ a. Drag the **AAA** action from the action palette and drop it in front of the GatewayScript action.



__ b. Double-click the **AAA** action to configure it.

__ c. Click new (**+**) to create a AAA Policy.

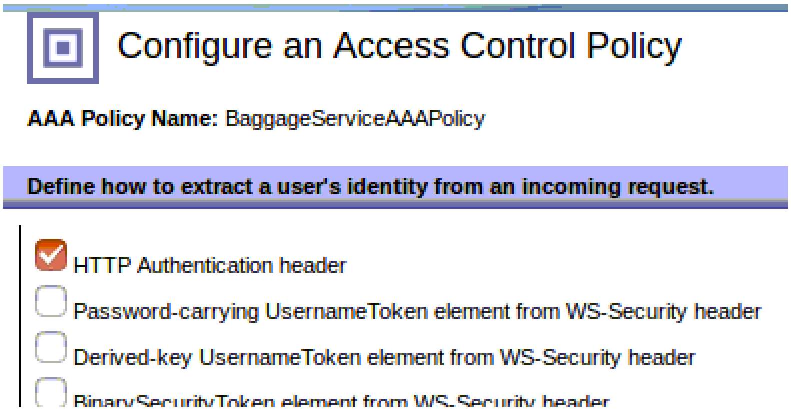
__ d. In the new window, enter the name `BaggageServiceAAAPolicy`.

__ e. Click **Create**.

Information

The next page identifies how to extract the user’s identity (and optionally a password) from the message. For this exercise, you indicate that the identity is in the HTTP header.

__ f. Select **HTTP Authentication header**.



Configure an Access Control Policy

AAA Policy Name: BaggageServiceAAAPolicy

Define how to extract a user's identity from an incoming request.

☑ HTTP Authentication header
□ Password-carrying UsernameToken element from WS-Security header
□ Derived-key UsernameToken element from WS-Security header
□ BinarySecurityToken element from WS-Security header

__ g. Click **Next**.

Information

The “Define how to authenticate the user” page determines how the policy validates the user identity that is extracted from the request message.

In a production environment, an authorized users list usually exists in a corporate directory server, such as a Lightweight Directory Access Protocol (LDAP) server. The identity information in the request message must be verified against the corporate directory server.

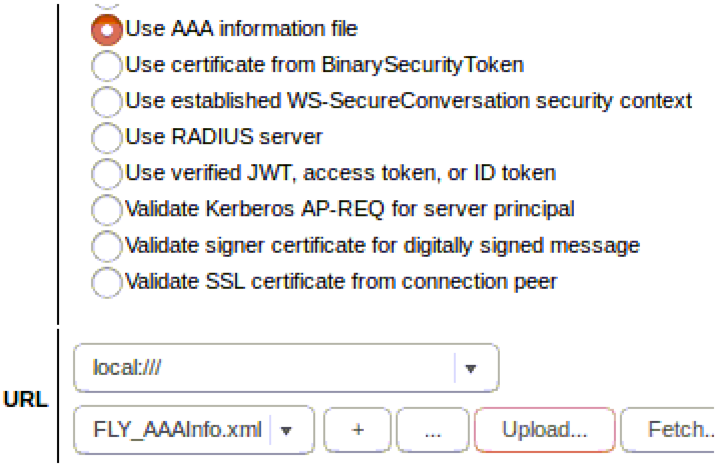
For the purposes of this case study, the list of authorized users exists in an XML file, `Fly_AAAInfo.xml`.

Next, you identify the user.

__ h. Select **Use AAA information file**. When you make the selection, the window is refreshed with a URL parameter field.

__ i. Click **Upload** and upload the file: `<`*`lab_files`*`>/AAA/FLY_AAAInfo.xml`

__ j. Ensure that the URL is set to **local:///** and **FLY_AAAInfo.xml**.



● Use AAA information file
○ Use certificate from BinarySecurityToken
○ Use established WS-SecureConversation security context
○ Use RADIUS server
○ Use verified JWT, access token, or ID token
○ Validate Kerberos AP-REQ for server principal
○ Validate signer certificate for digitally signed message
○ Validate SSL certificate from connection peer

URL: local:///
FLY_AAAInfo.xml

__ k. Click **Next**.

Information

`FLY_AAAInfo.xml` contains authentication information for user: `student` and password: `passw0rd`. You configure the client in SoapUI to include the user and password information in the HTTP header. DataPower authenticates the user for the requested rule in the policy against the `FLY_AAAInfo.xml` file.

Now you define how to extract the resource. Because the message is a REST request, you can expect that the HTTP method is important.

__ l. In the *Extract Resource* form, select **HTTP operation (GET or POST)**.

Information

The “Define how to extract the resource” page specifies how the access control policy determines what resource the client requested. In this scenario, the resource in question is the HHTP method and related REST operation.

__ m. Click **Next.**

__ n. For the authorization phase, select **Use AAA information file**.

__ o. Set the **URL** to the same file that you used for authentication: `local:///FLY_AAAInfo.xml`.

Information

The “Define how to authorize a request” page determines the access rules that are based on the resource and the user. The “output credential” from the authentication phase is “CurbsideServices”.The `FLY_AAAInfo.xml` file has an authorization entry that authorizes “CurbsideServices” if it uses a GET HTTP method.

__ p. Click **Next**.

Information

The last page of the AAA policy configuration wizard gives you the options of performing various post processing tasks. Such post-processing tasks might be to perform security protocol mediation such as creating a Kerberos/SPNEGO token or generating a signed SAML assertion. For this exercise, leave everything with the default values.

__ q. Click **Commit** to save the new AAA policy.

__ r. Click **Done**.

__ s. Ensure the AAA Policy `BaggageServiceAAAPolicy` is selected as the policy; then, click **Done** to save the AAA action.

__ 7. In the policy editor, click **Apply Policy**.

__ 8. Click the **Close Window** link to close the policy editor.

__ 9. Click **Apply** in the Multi-Protocol Gateway.

__ 10. Save the configuration.

1.4. Test that the BaggageServiceProxy can authenticate a client by using an HTTP authentication header and a AAAInfo file

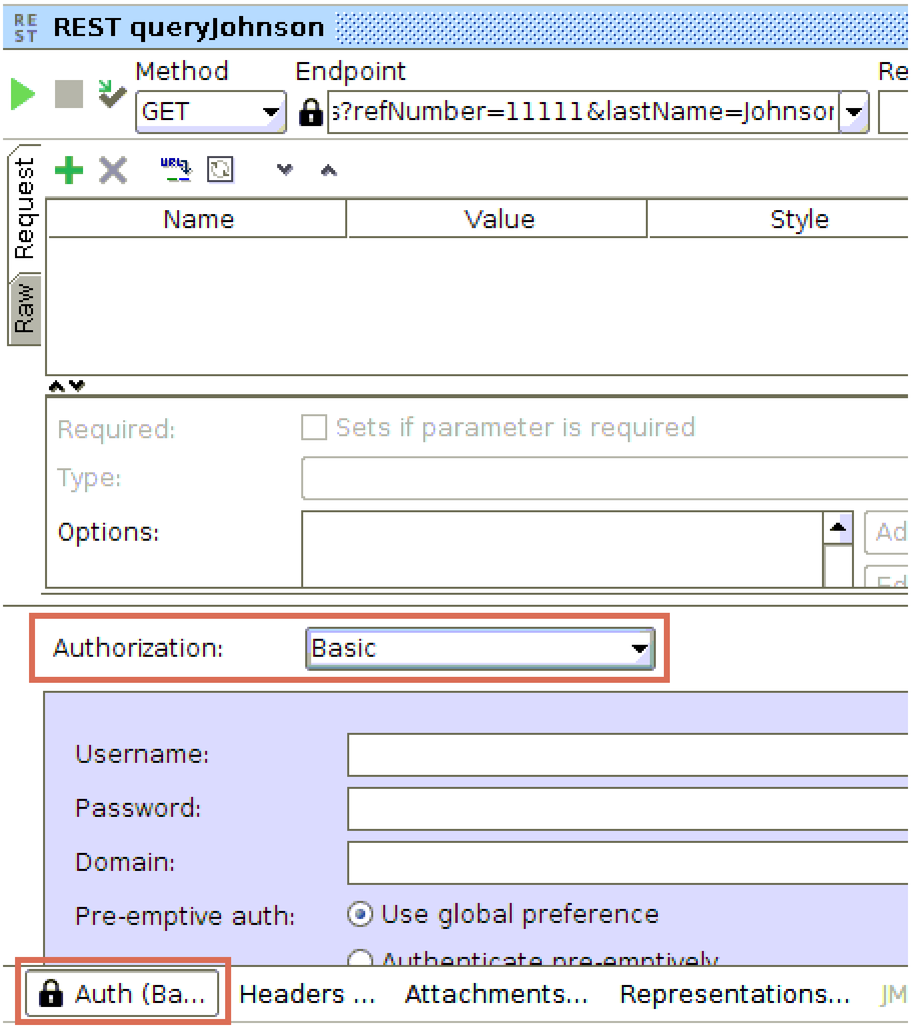
Use SoapUI to verify that the AAA action processes the client's HTTP basic authentication header login.

__ 1. Before testing, go to **ControlPanel > Troubleshooting** to verify that the log level is set to **debug**. If it is not, set it to that level.

__ 2. In **SoapUI**, open **REST queryJohnson** if it is not still open.

__ 3. Click the **Auth** tab that is in the bottom portion of the request tab.

__ 4. Set the **Authorization** type to `Basic`.



REST queryJohnson

Method: GET

Endpoint: s?refNumber=11111&lastName=Johnsoı

Required: □ Sets if parameter is required

Type:

Options:

Authorization: Basic

Username:

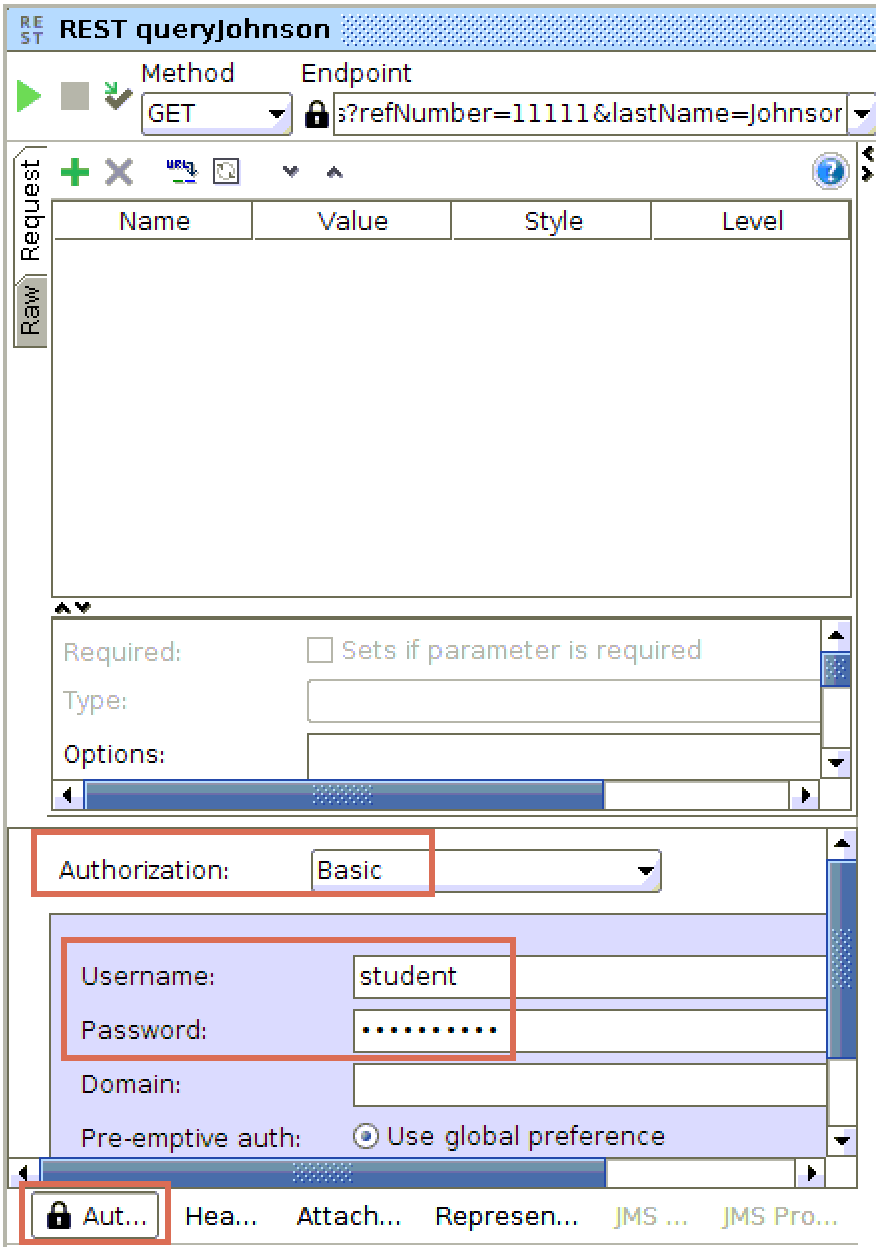
Password:

Domain:

Pre-emptive auth: ● Use global preference

○ Authenticate pre-emptively

__ 5. Enter a **Username** of `student` and a **Password** of `passw0rd`.



REST queryJohnson

Method: GET

Endpoint: s?refNumber=11111&lastName=Johnsor

Name	Value	Style	Level

Required: □ Sets if parameter is required

Type:

Options:

Authorization: Basic

Username: student

Password: ••••••••••

Domain:

Pre-emptive auth: ● Use global preference

__ 6. Ensure that the endpoint URL appears as:
`http://${dp_public_ip}:${mpgw_baggage_port}/BaggageService/Passenger/Bags?refNumber=11111&lastName=Johnson`

__ 7. Click the green **Submit** arrow to POST the message to BaggageServiceProxy. The request should return a successful baggage query.

__ 8. Confirm that the JSON structure of baggage information is received. Be sure to click the **JSON** tab.

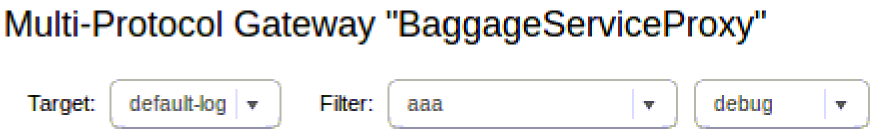
Optional

You can try several extra activities to further review this AAA behavior.

- Click the **http log** at the bottom of the SoapUI window. You can scroll back to see the Authorization header that SoapUI added to the HTTP GET request.
- Enable the probe for the MPGW. Run the request again. Open the probe and examine the **Context Variables** tab after the AAA action.

__ 9. On the Configure Multi-Protocol Gateway page, click **View log**.

__ 10. At the top of the log page, set the Filter to **aaa** and **debug**. This setting shows the entries that are at “debug” level or more severe and relate to the AAA category.



Multi-Protocol Gateway "BaggageServiceProxy"

Target: default-log

Filter: aaa debug

__ 11. You can examine the log for the authentication and authorization activity in the transaction.

__ 12. Close the log window.

1.5. Use an LDAP server for AAA

The previous configuration used the onboard AAA information file for the authentication and authorization information. In this section, you point the AAA policy to an LDAP server for this information.

__ 1. Edit the **BaggageServicePolicy** service policy of the **BaggageServiceProxy** MPGW.

__ 2. Open the AAA action so that you can edit the **BaggageServiceAAAPolicy**.

__ 3. You are not changing any identity extraction. Click **Next**.

__ 4. You are switching the authentication method. Select **Bind to LDAP server**.

__ 5. The page repaints to display more fields. Enter the following values:

- Host: `<image_ip>`
- Port: `389`
- LDAP Bind DN: `cn=admin,dc=ibm,dc=com`
- LDAP Version: `v3`
- LDAP Prefix: `cn=`
- LDAP Suffix: `dc=ibm,dc=com`

__ 6. Create an LDAP Bind Password Alias.

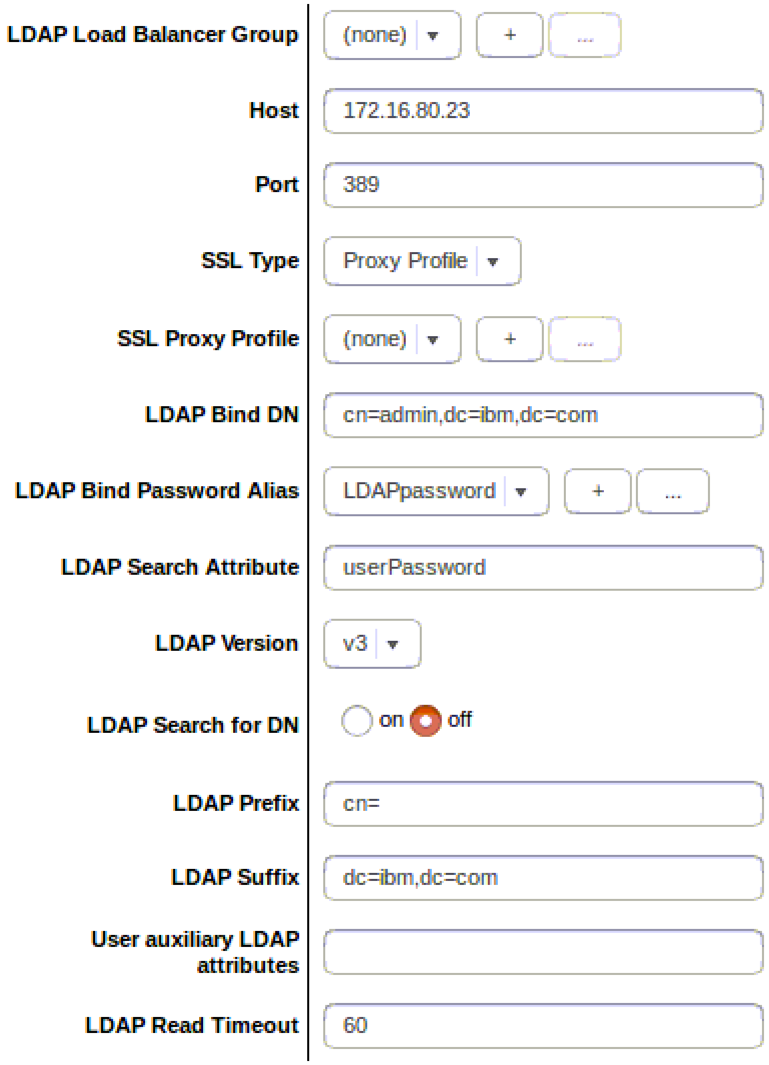
__ a. Click the **New (+)** button next to the LDAP Bind Password Alias field.

__ b. Enter a Name of `LDAPpassword`

__ c. Enter the password: `passw0rd`

__ d. Reenter the password.

__ e. Click **Apply**.



LDAP Load Balancer Group	(none) ▾ + ...
Host	172.16.80.23
Port	389
SSL Type	Proxy Profile ▾
SSL Proxy Profile	(none) ▾ + ...
LDAP Bind DN	cn=admin,dc=ibm,dc=com
LDAP Bind Password Alias	LDAPpassword ▾ + ...
LDAP Search Attribute	userPassword
LDAP Version	v3 ▾
LDAP Search for DN	○ on ● off
LDAP Prefix	cn=
LDAP Suffix	dc=ibm,dc=com
User auxiliary LDAP attributes	
LDAP Read Timeout	60

__ 7. Click **Next** to leave the authentication phase.

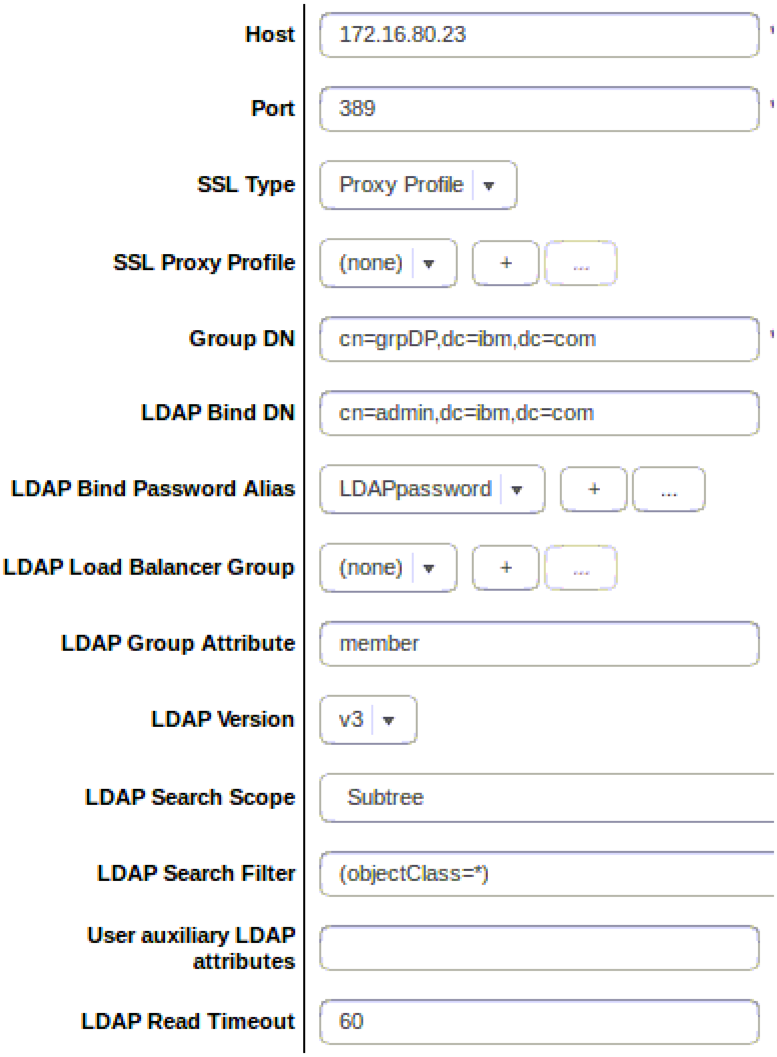
__ 8. Nothing is changing for extracting resources. Click **Next**.

__ 9. Authorization now uses an LDAP server. Select **Check membership in LDAP group**.

__ 10. The page repaints to display more fields. Enter the following values:

- Host: `<image_ip>`
- Port: `389`
- Group DN: `cn=grpDP,dc=ibm,dc=com`
- LDAP Bind DN: `cn=admin,dc=ibm,dc=com`
- LDAP Bind Password Alias: `LDAPpassword`

- LDAP Version: v3



Host	172.16.80.23
Port	389
SSL Type	Proxy Profile ▾
SSL Proxy Profile	(none) ▾ + ...
Group DN	cn=grpDP,dc=ibm,dc=com
LDAP Bind DN	cn=admin,dc=ibm,dc=com
LDAP Bind Password Alias	LDAPpassword ▾ + ...
LDAP Load Balancer Group	(none) ▾ + ...
LDAP Group Attribute	member
LDAP Version	v3 ▾
LDAP Search Scope	Subtree
LDAP Search Filter	(objectClass=*)
User auxiliary LDAP attributes	
LDAP Read Timeout	60

__ 11. Click **Next** to exit the authorization phase.

__ 12. Click **Commit** to end the AAA policy configuration.

__ 13. Click **Done**.

__ 14. Click **Done** to exit the AAA action.

__ 15. Click **Apply Policy**.

__ 16. Click **Apply** to save the MPGW if necessary.

1.6. Test the use of an LDAP server for authentication and authorization information

Use SoapUI to verify that the AAA action processes the client's HTTP basic authentication header login.

__ 1. In **SoapUI**, open **REST queryJohnson** if it is not still open.

__ 2. Click the **Auth** tab that is in the bottom portion of the request tab.

__ 3. Verify that the Basic Authorization is still present from the previous test.

__ 4. You can right-click in the response pane, click **Select all**, and then click **Clear** to remove the old response from the pane.

__ 5. Click the green **Submit** arrow to send the message to BaggageServiceProxy. The request should return a successful baggage query.

__ 6. Confirm that the JSON structure of baggage information is received. Be sure to click the **JSON** tab.

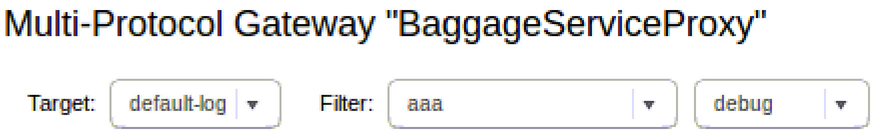
Optional

As before, you can try several extra activities to further review this AAA behavior.

- Click the **http log** at the bottom of the SoapUI window. You can scroll back to see the Authorization header that SoapUI added to the HTTP GET request.
- Enable the probe for the MPGW. Run the request again. Open the probe and examine the **Context Variables** tab after the AAA action.

__ 7. On the Configure Multi-Protocol Gateway page, click **View log**.

__ 8. At the top of the log page, set the Filter to **aaa** and **debug**. This setting shows the entries that are at "debug" level or more severe and relate to the AAA category.



Multi-Protocol Gateway "BaggageServiceProxy"

Target: default-log ▾ Filter: aaa ▾ debug ▾

__ 9. You can examine the log for the authentication and authorization activity in the transaction. Notice that the log entries mention that LDAP is being accessed.

__ 10. Close the log window.

__ 11. Save the configuration.

End of exercise

Exercise review and wrap-up

To restrict access to the `BaggageService` operations, an authentication, authorization, and auditing (AAA) action was applied to the processing rule. The first access control policy, `BaggageServiceAAApolicy`, allowed only clients with the correct user name and password in an HTTP Authorization header.

To demonstrate other forms of authentication and authorization, the AAA policy was modified to use an LDAP server to authenticate and authorize.

Exercise 2. Defining a three-legged OAuth scenario that uses DataPower services

Estimated time

01:30

Overview

In this exercise, you define the DataPower objects that are needed to implement a three-legged OAuth scenario: an OAuth client profile, an OAuth client group, a web token service, and a resource server. During the service creation, you create a AAA policy that specifies OAuth. Finally, you test the implementation by invoking a client that runs in a web server.

Objectives

After completing this exercise, you should be able to:

- Define an OAuth Client Profile and an OAuth Client Group object
- Create a AAA policy to support the OAuth protocol
- Configure a DataPower web token service
- Configure a DataPower implementation of an OAuth resource server

Introduction

DataPower provides multiple options and objects to support the OAuth 2.0 framework. It also supports the three-legged scenario for OAuth, which is the scenario for which OAuth was devised.

In this exercise, you configure the DataPower objects to support the three-legged scenario. You are supplied with the OAuth client code, which runs as JavaScript module in an IBM SDK for Node.js server. You use a browser to interact with the scenario as a resource owner. The back-end resource application is a multi-protocol gateway that was coded in the previous exercise. You create a web token service to act as the OAuth authorization server endpoint and the token endpoint. You also create a multi-protocol gateway to act as the OAuth resource server, which functions as the security gateway for the back-end application. The OAuth client, the JavaScript module, is defined to the BaggageServiceProxy by an OAuth client profile object.

Requirements

To complete this exercise, you need:

- Access to the DataPower BaggageServiceProxy

- Completion of the previous exercises (see the Preface section in the exercise instructions for details)
- Node.js to run the OAuth Client code
- `oauth-client.js` and `DP_oauthclient.json`: The OAuth Client and its properties file
- The BaggageStatusMockService web service that runs on the DataPower BaggageServiceProxy in the `FLYServices` domain
- An HTTP server
- Access to the *`<lab_files>`* directory

Exercise instructions

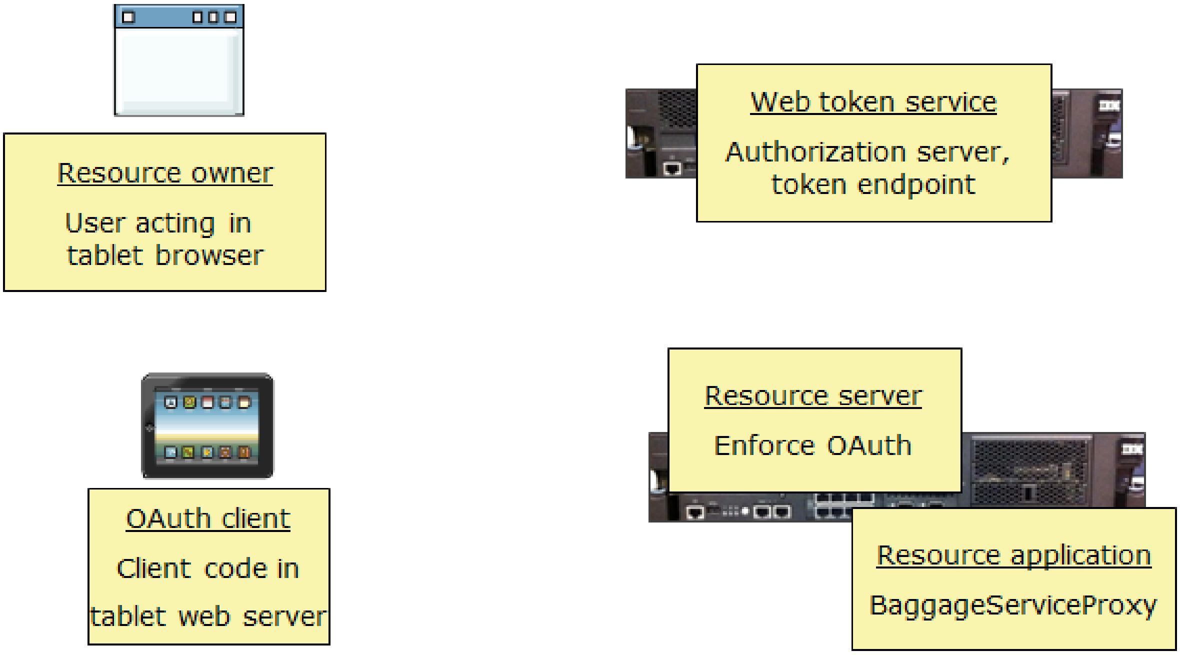
Preface

- Remember to use the domain and port address that you were assigned in the exercise setup. *Do not* use the default domain.
- This exercise depends on the completion of Exercise 1: Using DataPower to implement REST services.
- The references in exercise instructions refer to the following values:
  - *<lab_files>*: Location of the student lab files. Default location is: `/usr/labfiles/dp/`
  - <image_ip>: IP address of the student image (use `/sbin/ifconfig` from a terminal window to obtain value).
  - *<dp_internal_ip>*: IP address of the DataPower BaggageServiceProxy development and administrative functions that are used by internal resources such as developers.
  - *<dp_public_ip>*: IP address of the public services on the BaggageServiceProxy that is used by customer and clients.
  - *<dp_WebGUI_port>*: Port of the WebGUI. The default port is `9090`.
  - *<nn>*: Assigned student number. If there is no instructor, use “`01`”.
  - *<studentnn>*: Assigned user name and user account. If there is no instructor, use “`student01`”.
  - *<studentnn_password>*: Account password. In most cases, the initial value is the same as the user name. You are prompted to create a password on first use. Write it down.
  - *<studentnn_domain>*: Application domain that the user account is assigned to. If there is no instructor, use “`student01_domain`”.
  - *<FLY_baggage_port>*: Port number that the back-end BaggageServices web services listen on. The default port is 2068.
  - *<mpgw_baggage_port>*: 12*nn9*, where “*nn*” is the two-digit student number. This port number is the listener port of the BaggageServiceProxy that mediates between the REST or JSON client and the Baggage Services back-end application.
  - *<oauth_wts_port>*: 7*nn*0, where “*nn*” is the two-digit student number. This port number is the listener port of the web token service.
  - *<oauth_ep_port>*: 7*nn*3, where “*nn*” is the two-digit student number. This port number is the listener port of the multi-protocol gateway that acts as the OAuth enforcement point for the resource server.

2.1. The three-legged architecture for this exercise

For this exercise, you are implementing a three-legged OAuth architecture. It is used to support tablets at the baggage claim stations. A browser on each tablet can access the “find bag” REST service that runs on BaggageServiceProxy. The system designers wanted to secure the access from the tablets by using the OAuth protocol and OAuth clients that run on the tablets:

- The resource owner uses a browser as its user agent.
- The OAuth client is implemented in a JavaScript module under a Node.js web server.
- The authorization server is a web token service that supports the authorization and token endpoints.
- The resource server is implemented in a multi-protocol gateway. When the client’s access token is validated, it passes the resource request to the back-end resource application.
- The back-end resource application is the BaggageServiceProxy multi-protocol gateway that you coded in an earlier exercise. For this example, the resource application uses the bag ID that the user selected in the browser to return the current details on the bag, and the time that the response was created.



The `oauth-client.js` code that is used for the OAuth client is *not* intended to demonstrate good coding of an OAuth client or of a node.js server. For example, the OAuth client state that is sent to the web token service should be compared to the state value that the web token service returns. The redirect URL that the `oauth-client.js` sends to the web token service should use HTTPS, but this example code uses HTTP to simplify the setup.

2.2. Prepare the security objects

The OAuth exercise uses SSL, and a shared secret (symmetric) key. In this section, you generate a key-certificate pair and some security objects that are used in later steps.

__ 1. Log on to the DataPower WebGUI.

__ 2. Generate a key-certificate pair for the `oauth` crypto key and crypto certificate.

  __ a. Open **Administration > Miscellaneous> Crypto Tools.**

  __ b. Generate a 2048-bit RSA key with a **Common Name** of `oauth.` This generation creates an **oauth** crypto key and an **oauth** crypto credential.

Important

Ensure that you specify a **2048-bit** key, *not* the 1024-bit default. The SHA256 algorithm that is used later requires a minimum 2048-bit key. The error that is caused is not easy to debug.

__ 3. Define the identification credential object.

  __ a. On the **Control Panel**, enter `cred` in the search field.

  __ b. Select **Crypto Identification Credentials** from the resulting list.

  __ c. Click **Add** on the catalog list page.

  __ d. Enter the name as: `oauthIdCred`

  __ e. For Crypto Key, select **oauth**.

  __ f. For Certificate, select **oauth**.

  __ g. Click **Apply**.

__ 4. Define the shared secret key object.

  __ a. On the **Control Panel**, enter `shared` in the search field that is above the navigation bar.

  __ b. Click **Crypto Shared Secret Key** from the resulting list.

  __ c. From the **Configure Crypto Shared Secret Key** list, click **Add**.

  __ d. For the shared secret key object, enter `oauth-token-ssecret` for the name.

  __ e. For File Name, upload **sskey.txt** from `<lab_files>/oauth`.

  __ f. Click **Apply**.

__ 5. Create the SSL server profile object.

Information

This SSL server profile is used by the web token service and the resource server to manage SSL connections on each service’s front side.

__ a. On the **Control Panel**, enter `prof` in the search field.

__ b. Select **SSL Server Profile** from the resulting list.

__ c. Click **Add** on the catalog list page.

__ d. Enter the name as: `oauth-ssl-server`

__ e. For the Identification credentials, select **oauthIdCred**.

__ f. Leave the other fields at their defaults.

__ g. Click **Apply**.

__ 6. The objects that are needed for the security-related aspects are now defined. They are referenced in later steps as they are needed. Click **Save Configuration** to persist them.

2.3. Define the OAuth client

__ 1. On the Control Panel, enter `oauth` in the search field above the navigation bar.

__ 2. Click **OAuth Client Profile**.

__ 3. On the Configure OAuth Client Profile catalog list page, click **Add**.

__ 4. Define the OAuth client profile object.

Information

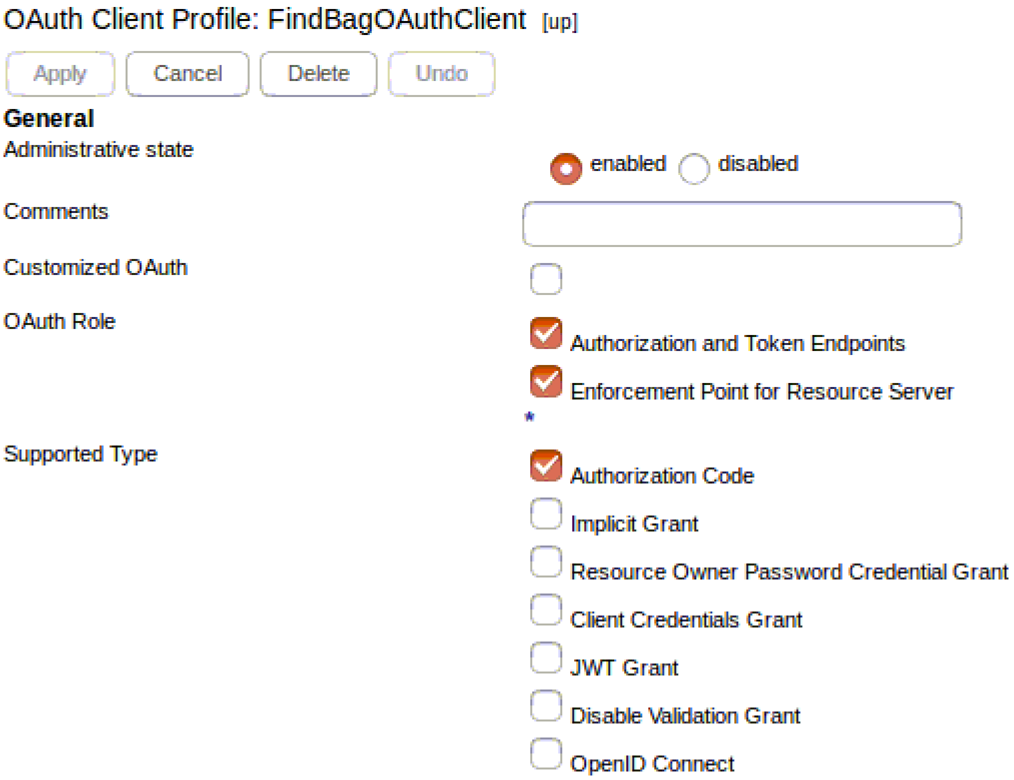
When a web token service and OAuth-related resource server are configured, they must define the group of predefined OAuth clients that can connect to them. This object defines the characteristics for a specific type of OAuth client. When the actual OAuth client contacts the OAuth-related service, it passes information that identifies itself. The service looks in its list of OAuth client profile objects to identify which one it is, determine its operating characteristics, and verify that it really is that type of client.

__ a. Enter the Name as: `FindBagOAuthClient`

The name of this object is also the **OAuth client ID** that the actual OAuth client sends to identify itself.

__ b. For the OAuth Role, select **Authorization and Token Endpoints** and **Enforcement Point for Resource Server**. This option specifies how the BaggageServiceProxy can act when this client contacts it.

__ c. For the Supported Type, select **Authorization Code**.



OAuth Client Profile: FindBagOAuthClient [up]

General

Administrative state ● enabled ○ disabled

Comments

Customized OAuth □

OAuth Role
☑ Authorization and Token Endpoints
☑ Enforcement Point for Resource Server

*

Supported Type
☑ Authorization Code
□ Implicit Grant
□ Resource Owner Password Credential Grant
□ Client Credentials Grant
□ JWT Grant
□ Disable Validation Grant
□ OpenID Connect

__ d. Set the Client Type to **Confidential**.

__ e. For Authentication Method, select **Client Secret**.

__ f. Clear the **Generate Client Secret** check box in the Authorization and Token Endpoints section of the page.

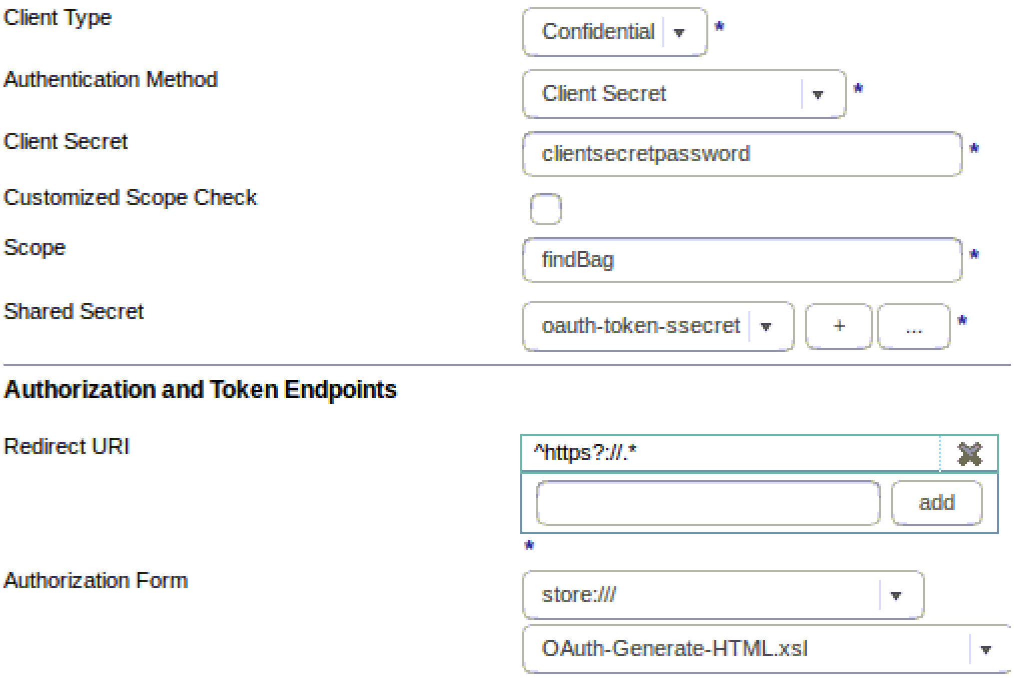
__ g. For Client Secret, enter: `clientsecretpassword`

__ h. Enter the Scope as: `findBag`

__ i. Select a Shared Secret of **oauth-token-ssecret**. This symmetric key is used to encrypt the token as it is passed around in the OAuth protocol.

__ j. For the Redirect URI, enter: `^https?://.*`
This regular expression allows any HTTP or HTTPS redirect URI to be acceptable for this OAuth client. Be sure to click **add**.

__ k. Select the Authorization Form **OAuth-Generate-HTML.xsl** that is in the **store:** directory. This sample stylesheet generates the “grant permission” challenge to the user.



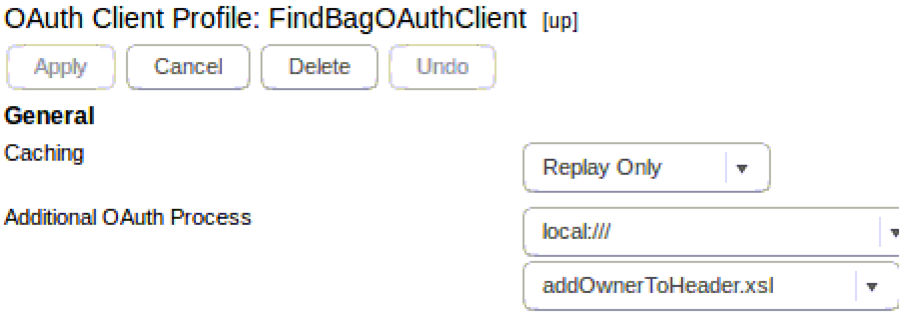
Client Type	Confidential *
Authentication Method	Client Secret *
Client Secret	clientsecretpassword *
Customized Scope Check	□
Scope	findBag *
Shared Secret	oauth-token-ssecret *

Authorization and Token Endpoints

Redirect URI	^https?://.* *
Authorization Form	store:/// OAuth-Generate-HTML.xsl

__ l. Click the **Advanced** tab to display more settings.

__ m. Specify an Additional OAuth Process stylesheet. Upload the **addOwnerToHeader.xsl** file from the `<lab_files>/oauth` directory into your domain’s **local:** directory. This sample stylesheet adds an HTTP header of “ResourceOwner” with a value of the verified resource owner name if the processing is during the enforcement point of the resource server.



OAuth Client Profile: FindBagOAuthClient [up]

General

Caching	Replay Only
Additional OAuth Process	local:/// addOwnerToHeader.xsl

__ n. Notice the default lifetimes of the grant code and access token.

__ o. Review the Enforcement Point for Resource Server section of the **Advanced** tab. These selections automatically add OAuth-related HTTP headers to the response from the resource server. The “`addOwnerToHeader.xsl`” that you specified for the Additional OAuth Process field does basically what the **Resource Owner** check box does. This approach was used to demonstrate the use of a stylesheet to do extra processing during the OAuth interactions.

__ p. Click **Apply**.

2.4. Define the OAuth client group

When a web token service or an OAuth-enforcing resource server is configured in DataPower, multiple types of OAuth clients might be involved. Rather than having to list each client type in the services, DataPower provides an OAuth client group object that gathers any related client types together.

__ 1. On the **Control Panel**, enter `oauth` in the search field above the navigation bar.

__ 2. Click **OAuth Client Group**.

__ 3. On the Configure OAuth Client Group catalog list page, click **Add**.

__ 4. Define the OAuth client group object:

  __ a. Enter the Name as: `myOAuthClientGroup`

  __ b. Select the OAuth Roles of **Authorization and Token Endpoints** and **Enforcement Point for Resource Server**. This option specifies what roles the associated DataPower services can perform when they are working with this group.

  __ c. For the Client list, select the OAuth client object you created: **FindBagOAuthClient**.

  __ d. Click **Add**.

  __ e. Click **Apply**.

__ 5. Click **Save Configuration** to persist the OAuth client-related objects.

2.5. Define the AAA policy for the web token service

In this section, you create the AAA policy that implements the authorization server and token endpoint AAA behavior for a web token service. This policy is used later in the service policy for the web token service.

The following approach defines the AAA policy object by using the Objects approach, rather than a wizard.

__ 1. On the **Control Panel**, enter `aaa` in the search field above the navigation bar.

__ 2. Click **AAA Policy**.

Information

When you created a AAA policy in earlier exercises, you used the AAA policy wizard that is part of the policy editor. In this case, you are directly accessing the stand-alone AAA policy object. Rather than being guided through the configuration of a AAA policy object, you are presented with a set of tabs that deal with specific aspects of a AAA policy. You must ‘lead’ yourself through the configuration.

__ 3. On the Configure AAA Policy catalog list page, click **Add**.

__ 4. Define the AAA policy object:

__ a. Enter the Name as: `oauthAZ`

__ b. Click the **Identity extraction** tab.

__ c. Select two Methods: **HTTP Authentication header** and **OAuth**.

__ d. Selecting **HTTP Authentication header** causes the HTTP Basic Authentication Realm field to appear. Leave it at its default value of **login**.

__ e. Selecting **OAuth** causes the Registered OAuth clients choice to display. Select the OAuth client group that you defined in a previous step: **myOAuthClientGroup**.

__ f. Click the **Authentication** tab.

__ g. For Method, select **Use AAA information file**.

__ h. When the method is selected, the remaining fields on the tab adjust. For the AAA information file URL, use the **FLY_AAAInfo.xml** file in the **local:** directory. If this XML file is not in the directory, upload it from the *`<lab_files>`*`/BookingServices` directory.

__ i. Click the **Credential mapping** tab. Nothing needs to be entered on this tab.

__ j. Click the **Resource extraction** tab.

__ k. Select **Processing metadata**.

__ l. The Processing metadata items choice appears. Select **oauth-scope-metadata**. This option is used in the stylesheets to retrieve the scope of the request.

__ m. Click the **Resource mapping** tab. Nothing needs to be entered on this tab.

__ n. Click the **Authorization** tab.

__ o. Select a Method of **Allow any authenticated client**.

__ p. Click the **Postprocessing** tab. Nothing needs to be done on this tab.

__ q. Click **Apply** to save the AAA policy for use in the web token service.

2.6. Define the web token service

DataPower provides a special service type, the web token service, that is designed to act as an OAuth authorization server and token endpoint. In this section, you configure a specific implementation of a web token service.

__ 1. On the **Control Panel**, enter `web tok` in the search field above the navigation bar.

__ 2. Click **New Web Token Service**.

__ 3. On the Create a Web Token Service page, enter the Web Token Service Name as: `myWTS`

__ 4. Click **Next**.

__ 5. On the next page, define the front side access to the service:

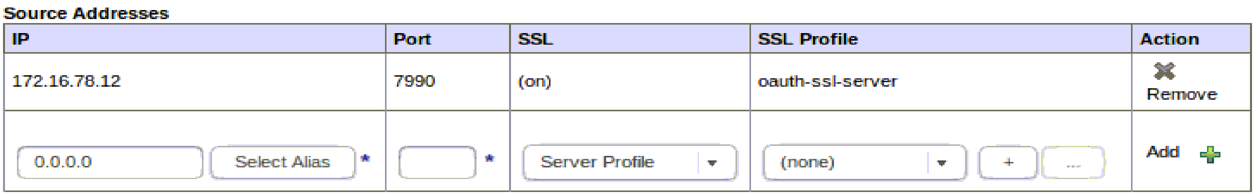
__ a. For the IP, use: *<dp_public_ip>*

__ b. For Port, enter: *<oauth_wts_port>*

__ c. For SSL, select **ServerProfile**.

__ d. For the SSL Profile, select the profile that you built in the earlier step: **oauth-ssl-server**.

__ e. Click the **Add** icon on the far right of the page.



Source Addresses

IP	Port	SSL	SSL Profile	Action
172.16.78.12	7990	(on)	oauth-ssl-server	Remove
0.0.0.0 Select Alias *	*	Server Profile	(none) + ...	Add

__ 6. Click **Next**.

__ 7. On the next page, select the AAA policy that you created in the earlier section: **oauthAZ**.

__ 8. Click **Next**.

__ 9. On the next page, review your changes, and click **Commit**.

__ 10. Click **Done**.

__ 11. Click **Save Configuration**.

2.7. Define the AAA policy for an enforcement point

In the next section, you will create the multi-protocol gateway service that acts as the enforcement point for OAuth-secured access to the real back-end application.

In this section, you create the AAA policy that implements the enforcement point behavior.

__ 1. Upload a stylesheet to the **local:** directory that is used for credential mapping.

__ a. On the Control Panel, start to enter `file` into the search field above the navigation bar.

__ b. Click **File Management**.

__ c. Select **Actions** to the right of the **local:** directory, and click **Upload Files**.

__ d. Browse to *`<lab_files>`*`/oauth`, and select **MapResourceToScope.xsl** to upload to the local: directory.

__ 2. Define the AAA policy object:

__ a. On the **Control Panel**, enter `aaa` into the search field above the navigation bar.

__ b. Click **AAA Policy**.

__ c. On the Configure AAA Policy catalog list page, click **Add**.

__ d. On the Configure AAA Policy page, enter the Name as: `oauthFindBagScopeEnforcement`

__ e. Click the **Identity extraction** tab.

__ f. Select the Method: **OAuth**.

__ g. Selecting OAuth causes the Registered OAuth clients choice to display. Select the OAuth client group that you defined in a previous step: **myOAuthClientGroup**.

__ h. Click the **Authentication** tab.

__ i. For Method, select **Pass identity token to authorization phase**.

__ j. Click the **Credential mapping** tab. Nothing needs to be entered on this tab.

__ k. Click the **Resource extraction** tab.

__ l. Select the **URL sent by client** and **Processing metadata** check boxes.

__ m. The Processing metadata items choice appears. Select **oauth-scope-metadata**. This option is used in the stylesheets to retrieve the scope of the request.

__ n. Click the **Resource mapping** tab.

__ o. Set the Method as **Custom**.

__ p. Enter a reference to the file that you uploaded: **local:///MapResourceToScope.xsl**. This stylesheet verifies that the scope that is sent by the client matches the scope that was verified as part of the OAuth checking.

Note

The activity to specify a custom method and stylesheet for the “map resources” phase is only used for demonstration purposes. Since V6.0, DataPower automatically verifies that the requested scope matches the authorized scope if the “extract resource” phase specifies “URL sent to client” and “Processing metadata” of oauth-scope-metadata.

__ q. Click the **Authorization** tab.

__ r. Select a Method of **Allow any authenticated client**.

__ s. Click the **Postprocessing** tab. Nothing needs to be done on this tab.

__ t. Click **Apply** to save the AAA policy for later use.

2.8. Define the enforcement point and resource server

In this section, you define a multi-protocol gateway that acts as the enforcement point for OAuth processing, and calls the actual back-end application.

__ 1. On the **Control Panel**, enter `multi` in the search field above the navigation bar.

__ 2. Click **Edit Multi-Protocol Gateway**.

__ 3. On the Configure a Multi-Protocol Gateway catalog list page, click **Add**.

__ 4. On the Configure a Multi-Protocol Gateway page, enter the name as: `FindBagEnforcementServer`

__ 5. Enter the Default Backend URL as: `http://`*`<dp_public_ip>`*`:`*`<mpgw_baggage_port>`*. This URL is pointing to your BaggageServiceProxy in your domain.

__ 6. Set the Request Type as **Non-XML**.

__ 7. Set the Response Type as **JSON**.

__ 8. Configure an HTTPS front side handler.

  __ a. At the Front Side Protocol field, click the new (**+**) button.

  __ b. Select **HTTPS Front Side Handler**.

  __ c. For the Name, enter: `FindBagEnforcement_HTTPS_7nn3`

  __ d. For the Local IP Address, enter the value as: *`<dp_public_ip>`*

  __ e. For the Port Number, enter: *`<oauth_ep_port>`*

  __ f. Select the **GET method** check box.

  __ g. Accept the other default methods and versions.

  __ h. Set the SSL server type to **Server Profile**.

  __ i. Select the SSL server profile to the one that you configured earlier: **oauth-ssl-server**.

  __ j. Click **Apply**.

__ 9. Configure the Multi-Protocol Gateway Policy:

  __ a. Click the new (**+**) button.

  __ b. Enter the Policy Name as: `FindBagEnforcement`

  __ c. Click **Apply Policy**.

__ 10. Configure the request rules:

  __ a. Click **New Rule**.

  __ b. Set the direction as **Client to Server**.

  __ c. Double-click the **Match action** to configure it.

__ d. Configure a Matching Rule that matches the URL `/favicon.ico`. This rule matches an HTTP request for the favicon that might show in the address bar of the browser. Give it any name that you want.

__ e. Drag an **Advanced action** to the rule configuration path.

__ f. Configure it as a **Set Variable** action.

__ g. Set the **var://service/mpgw/skip-backside** to **1**.

__ h. Drag a **Results action** to the path.

__ i. Configure the action to pass the **INPUT** input context to the **OUTPUT** output context.

__ j. Click **Apply Policy** to save the rule and processing policy.

__ k. Click **New Rule**.

__ l. Set the direction as **Client to Server**.

__ m. Double-click the **Match action** to configure it.

__ n. Configure a Matching Rule that matches on all URLs.

__ o. Drag a **AAA action** to the path.

__ p. Configure it with the AAA policy that you created earlier: **oauthFindBagScopeEnforcement**.

__ q. Click **Done**.

__ r. Drag an **Advanced action** to the path.

__ s. Configure this action as a **Convert Query Params to XML** action. This action converts non-XML encoded input (an HTTP POST of HTML form or URI parameters) into an equivalent XML message.

__ t. No further configuration is needed on this action, so click **Done**.

__ u. Drag a **Transform** action to the rule.

__ v. For the Transform File, upload **reformatBagIdURI.xsl** from `<lab_files>/oauth` into the **local:** directory. This stylesheet reformats the input from the OAuth client into a RESTful request.

__ w. Click **Done**.

__ x. Click **Apply Policy**.

__ 11. Configure the response rule:

__ a. Click **New Rule**.

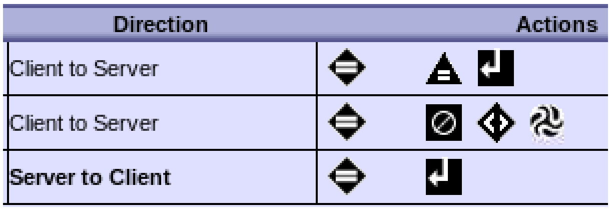
__ b. Set the direction as **Server to Client**.

__ c. Double-click the **Match** action to configure it.

__ d. Configure a Matching Rule that matches on all URLs.

__ e. Drag a **Results** action to the path.

__ f. Configure the action to pass the **INPUT** input context to the **OUTPUT** output context.



Direction	Actions
Client to Server	
Client to Server	
Server to Client	

__ g. Click **Apply Policy**.

__ h. Close the policy editor window.

__ 12. Click **Apply** to save the MPGW.

__ 13. Click **Save Configuration**.

2.9. Configure the OAuth client properties

The OAuth client is running as `oauth-client.js`, a JavaScript server program that runs under Node.js. The server code calls a properties file, `DP_oauthclient.json`, which specifies the values that are used for the URLs and the like. This external properties file makes it easy to “configure” `oauth-client.js` for different environments.

In this section, you set and verify the values in the file.

__ 1. Use gedit to open `localuser/DP_oauthclient.json` on the Linux file system. This JSON structure is a set of name-value pairs.

__ 2. Set the “localip” value to: *`<image_ip>`*

This value is the IP address of the OAuth client that is running under Node.js.

__ 3. Set the “oauthip” value to: *`<dp_public_ip>:<oauth_wts_port>`*

This value is the IP address and port of your web token service.

__ 4. Set the “oauthrs” value to: *`<dp_public_ip>:<oauth_ep_port>`*

This value is the IP address and port of your enforcement point service.

__ 5. Set the “scope” value to: `findBag`

This value is the scope within which your request operates.

__ 6. Set the “client_id” value to: `FindBagOAuthClient`

This value is the “client ID” that the OAuth client sends to the web token service. It is also the name of the OAuth Client Profile object that represents this actual client on the BaggageServiceProxy.

__ 7. Set the “client_secret” value to: `clientsecretpassword`

This value is the password that is associated with the OAuth client. It is also specified in the OAuth Client Profile object.

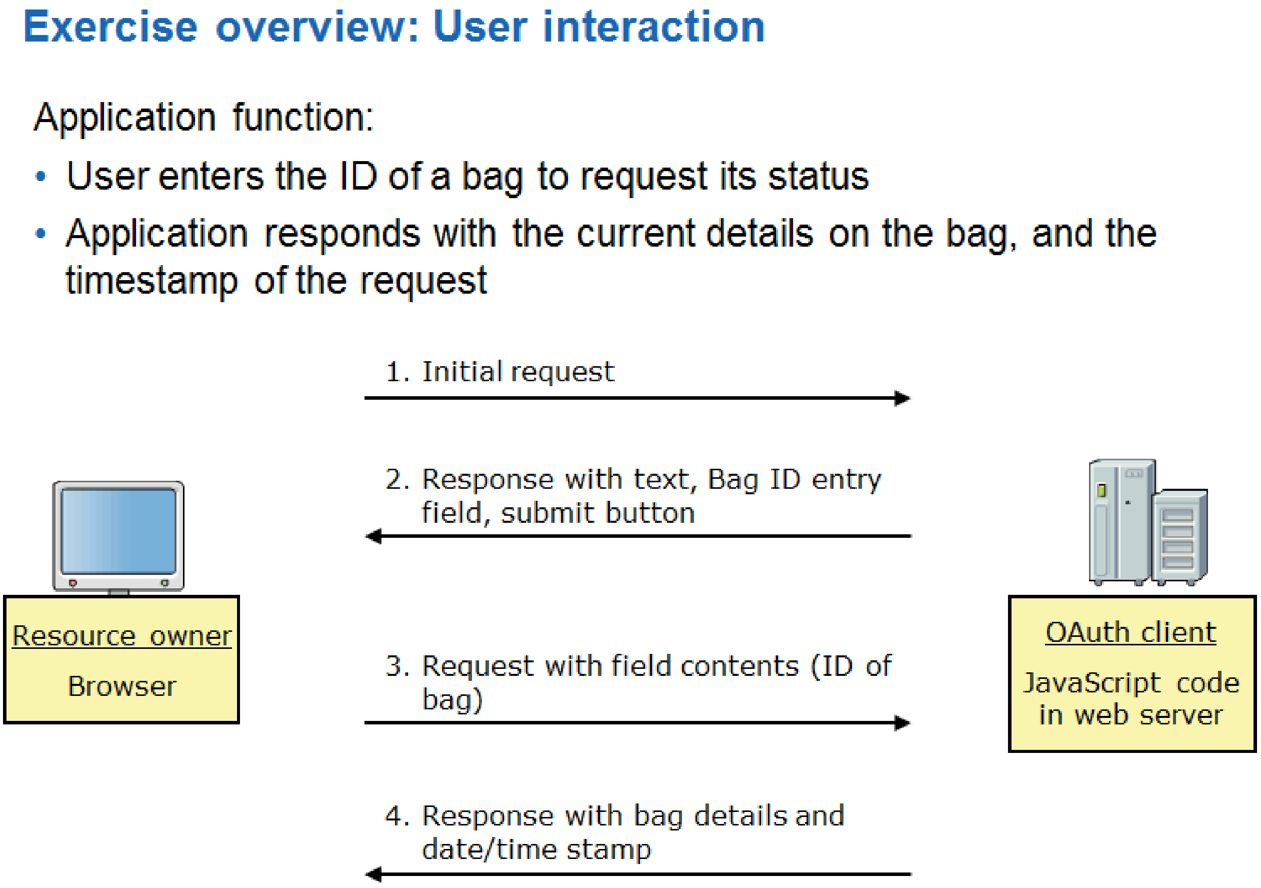
__ 8. Set the “localport” value to: `4000`

This value is the port on your image that Node.js listens on for your OAuth client oath-client.js.

__ 9. A copy of the JSON file is also in *`<lab_files>`*`/oauth/oauthclient`.

2.10. Test the solution

In this section, you are the resource owner and use the browser as the user agent. The “client” view of the interaction is provided in the following figure:



Exercise overview: User interaction

Application function:

- User enters the ID of a bag to request its status
- Application responds with the current details on the bag, and the timestamp of the request

Resource owner

Browser

OAuth client

JavaScript code in web server

1. Initial request
2. Response with text, Bag ID entry field, submit button
3. Request with field contents (ID of bag)
4. Response with bag details and date/time stamp

__ 1. Start the OAuth client.

__ a. Open a terminal window.

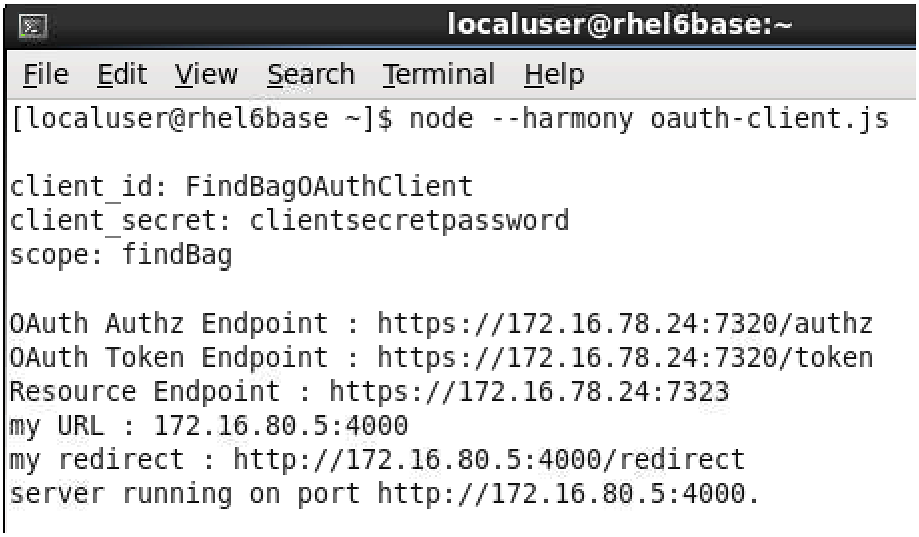
__ b. Start the OAuth client:

```
node --harmony oauth-client.js
```

Information

This command starts `oauth-client.js` on Node.js. The “`--harmony`” flag enables the newer ECMA features in JavaScript.

__ 2. The console writes some of the variable values in `oauth-client.js`:



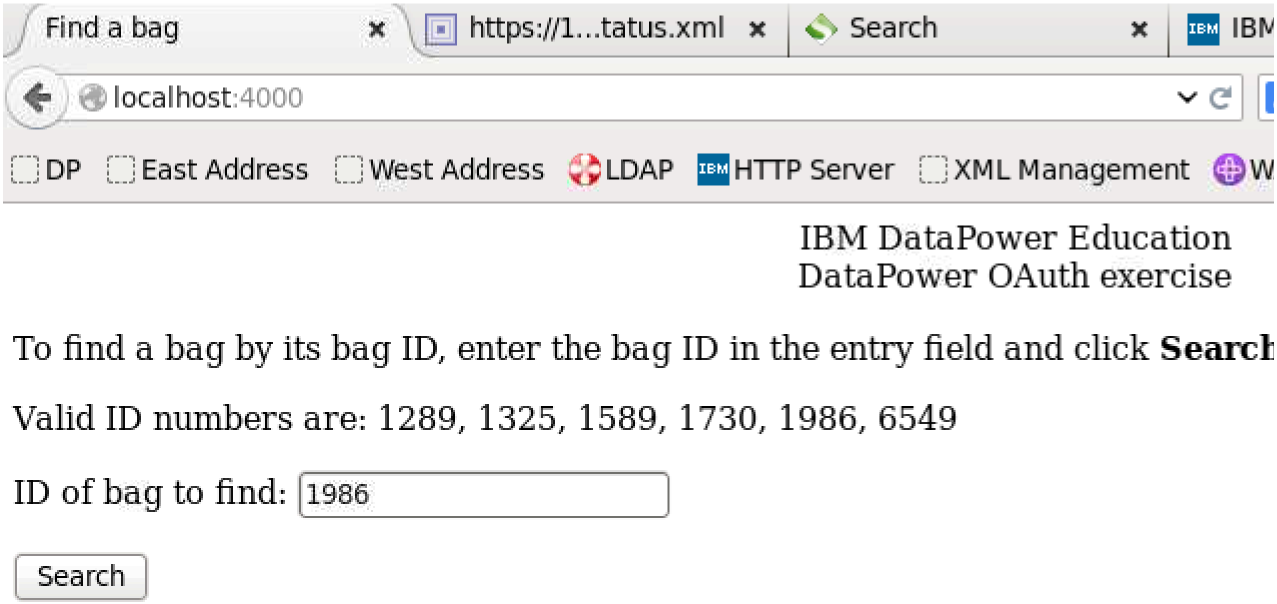
```
[localuser@rhel6base ~]$ node --harmony oauth-client.js

client_id: FindBagOAuthClient
client_secret: clientsecretpassword
scope: findBag

OAuth Authz Endpoint : https://172.16.78.24:7320/authz
OAuth Token Endpoint : https://172.16.78.24:7320/token
Resource Endpoint : https://172.16.78.24:7323
my URL : 172.16.80.5:4000
my redirect : http://172.16.80.5:4000/redirect
server running on port http://172.16.80.5:4000.
```

__ 3. In a browser, enter:

`http://localhost:4000`

This URL invokes `oauth-client.js`.

__ 4. The OAuth client returns a form. In the screen capture, “1986” is entered:



IBM DataPower Education
DataPower OAuth exercise

To find a bag by its bag ID, enter the bag ID in the entry field and click **Search**

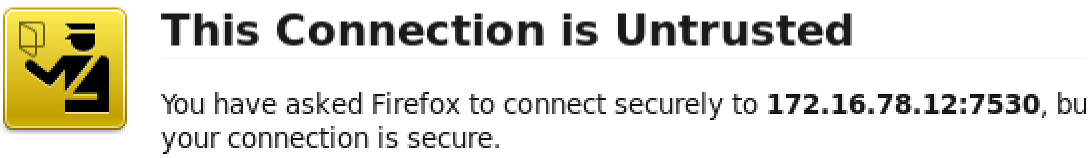
Valid ID numbers are: 1289, 1325, 1589, 1730, 1986, 6549

ID of bag to find: 1986

Search

__ 5. Click **Search**.

__ 6. A redirection occurs to the web token service. If this attempt is the first time for the browser to access the web token service, you get an “untrusted connection” dialog box:

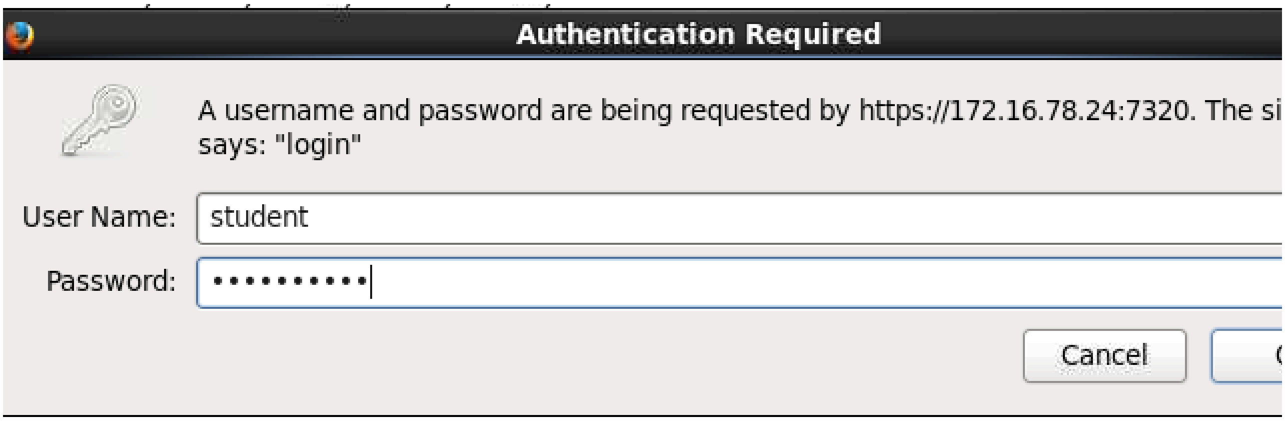


This Connection is Untrusted

You have asked Firefox to connect securely to **172.16.78.12:7530**, bu
your connection is secure.

__ 7. Accept the exception.

__ 8. If this attempt is the first time for the browser to connect to the web token service, you are prompted to authenticate. Enter `student` for the **User Name** and `passw0rd` for the **Password**:



Authentication Required

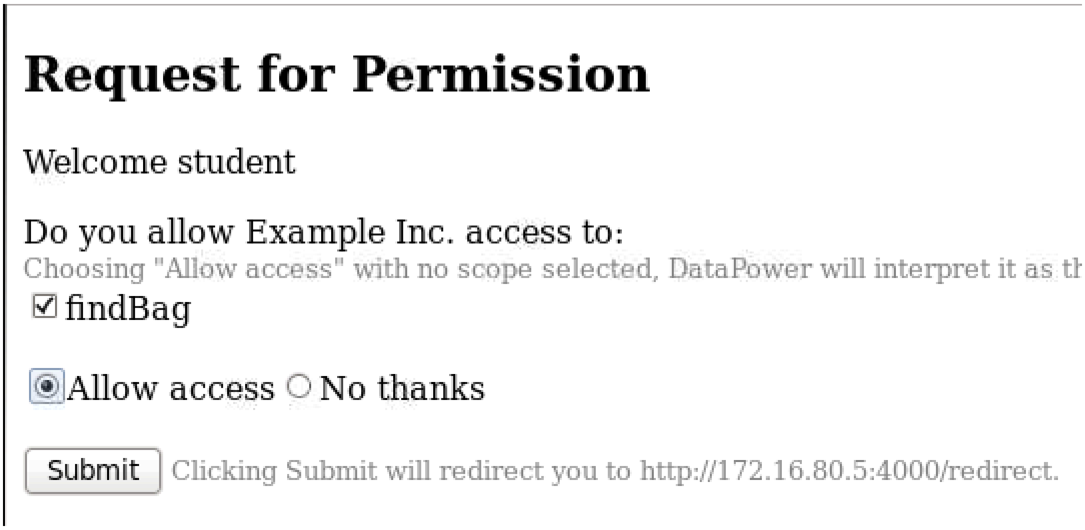
A username and password are being requested by https://172.16.78.24:7320. The si
says: "login"

User Name: student

Password: ••••••••••

__ 9. Click **OK**.

__ 10. Now the web token service asks for your permission for the OAuth client to access a resource within a scope:



Request for Permission

Welcome student

Do you allow Example Inc. access to:
Choosing "Allow access" with no scope selected, DataPower will interpret it as tl
☑ findBag

● Allow access ○ No thanks

Clicking Submit will redirect you to http://172.16.80.5:4000/redirect.

__ 11. Verify that the scope **findBag** is selected. Select **Allow access**.

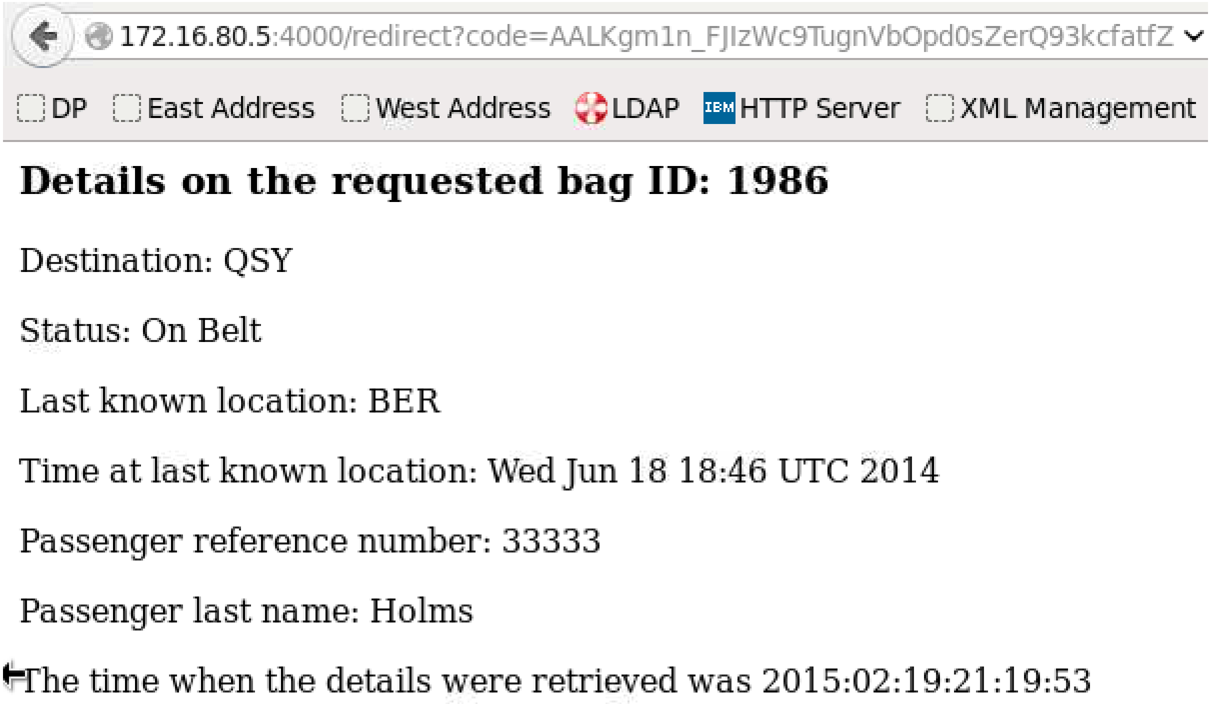
__ 12. Click **Submit**.

__ 13. The OAuth client then receives the authorization grant code from the web token service. It uses the grant code to request the access token from the web token service. Now that it has the access token, the OAuth client calls the enforcement point and resource server, passing the access token, the scope, and the bag ID that was selected from the list.

__ 14. The resource server verifies the access token and scope, and if successful, it calls BaggageServiceProxy.

__ 15. The response returns to the resource server as JSON data. The resource server passes that JSON data to the OAuth client.

__ 16. The OAuth client reformats the JSON data into HTML text, and presents the results to the browser:



172.16.80.5:4000/redirect?code=AALKgm1n_FJlzWc9TugnVbOpd0sZerQ93kcfatfZ

Details on the requested bag ID: 1986

Destination: QSY

Status: On Belt

Last known location: BER

Time at last known location: Wed Jun 18 18:46 UTC 2014

Passenger reference number: 33333

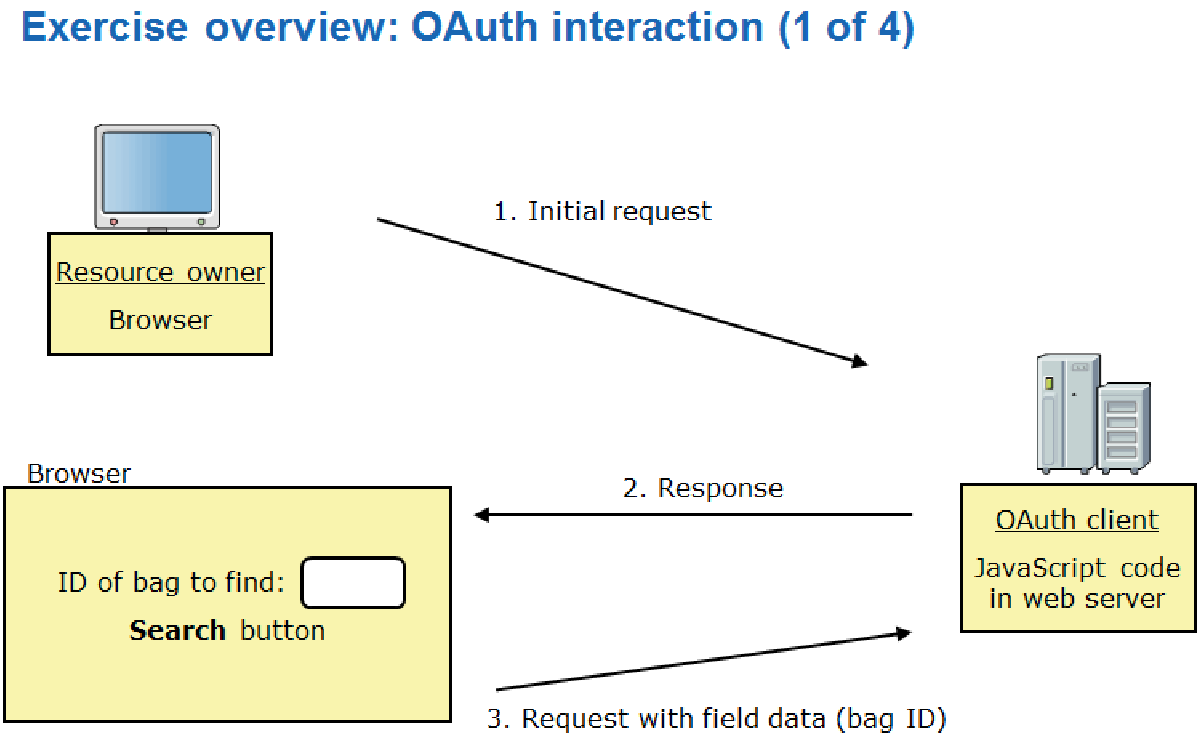
Passenger last name: Holms

The time when the details were retrieved was 2015:02:19:21:19:53

If you have errors, be sure to check the terminal window for details on the protocol flow and the DataPower logs.

2.11. The underlying OAuth interactions

Most of the OAuth interactions are hidden from the client. In these steps, you can review what occurred when you ran the test. These figures show the high-level view of what happens during the interaction.



Exercise overview: OAuth interaction (1 of 4)

Resource owner
Browser

1. Initial request

OAuth client
JavaScript code in web server

2. Response

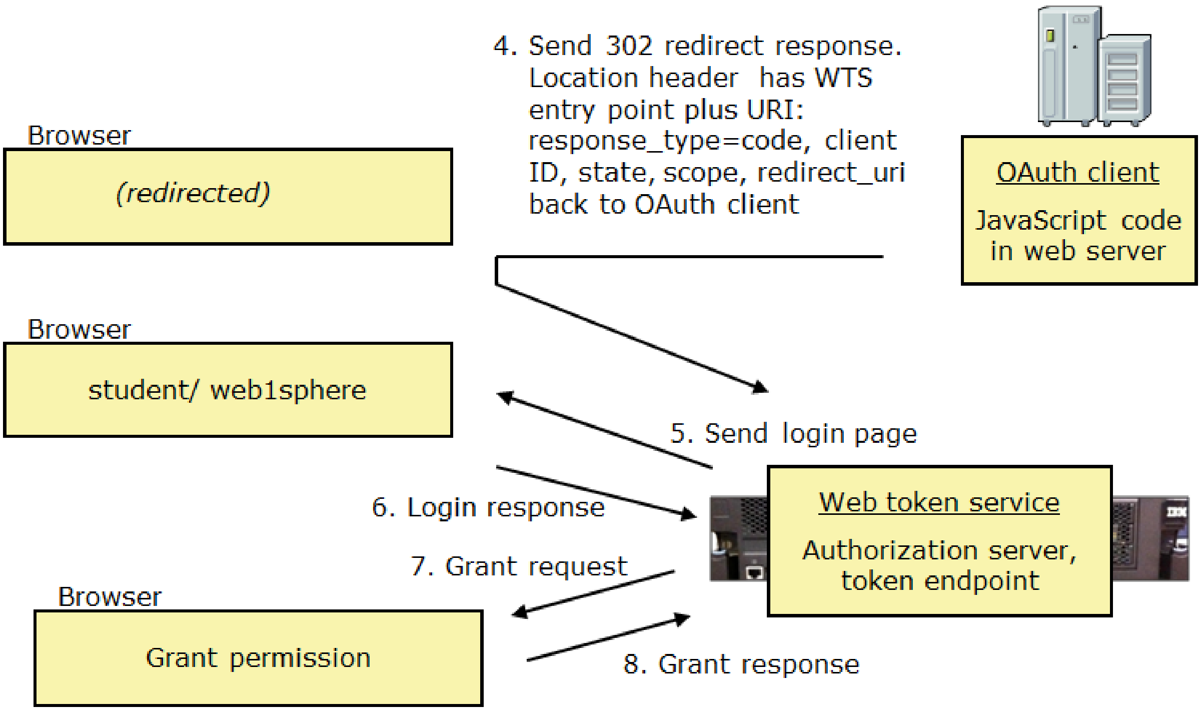
Browser

ID of bag to find:

Search button

3. Request with field data (bag ID)

Exercise overview: OAuth interaction (2 of 4)



4. Send 302 redirect response. Location header has WTS entry point plus URI: response_type=code, client ID, state, scope, redirect_uri back to OAuth client

Browser

(redirected)

OAuth client

JavaScript code in web server

Browser

student/ web1sphere

5. Send login page

6. Login response

Web token service

Authorization server, token endpoint

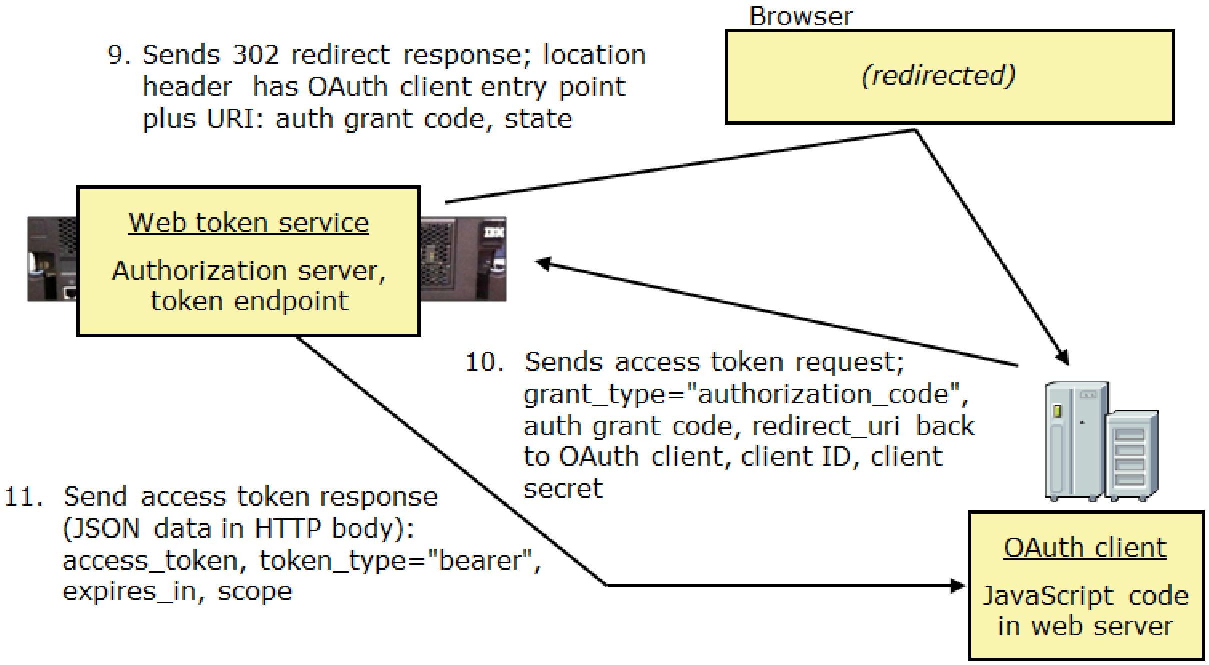
7. Grant request

Browser

Grant permission

8. Grant response

Exercise overview: OAuth interaction (3 of 4)



Browser

(redirected)

9. Sends 302 redirect response; location header has OAuth client entry point plus URI: auth grant code, state

Web token service

Authorization server, token endpoint

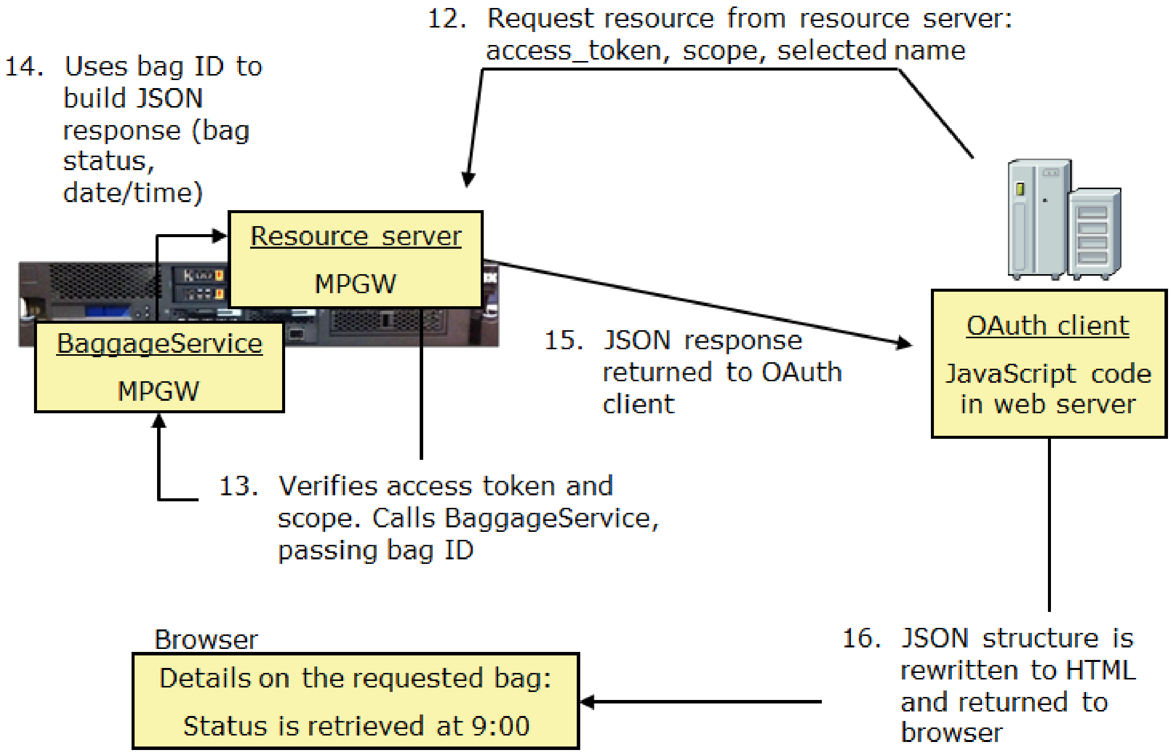
10. Sends access token request; grant_type="authorization_code", auth grant code, redirect_uri back to OAuth client, client ID, client secret

11. Send access token response (JSON data in HTTP body): access_token, token_type="bearer", expires_in, scope

OAuth client

JavaScript code in web server

Exercise overview: OAuth interaction (4 of 4)



12. Request resource from resource server: access_token, scope, selected name

14. Uses bag ID to build JSON response (bag status, date/time)

Resource server

MPGW

BaggageService

MPGW

15. JSON response returned to OAuth client

OAuth client

JavaScript code in web server

13. Verifies access token and scope. Calls BaggageService, passing bag ID

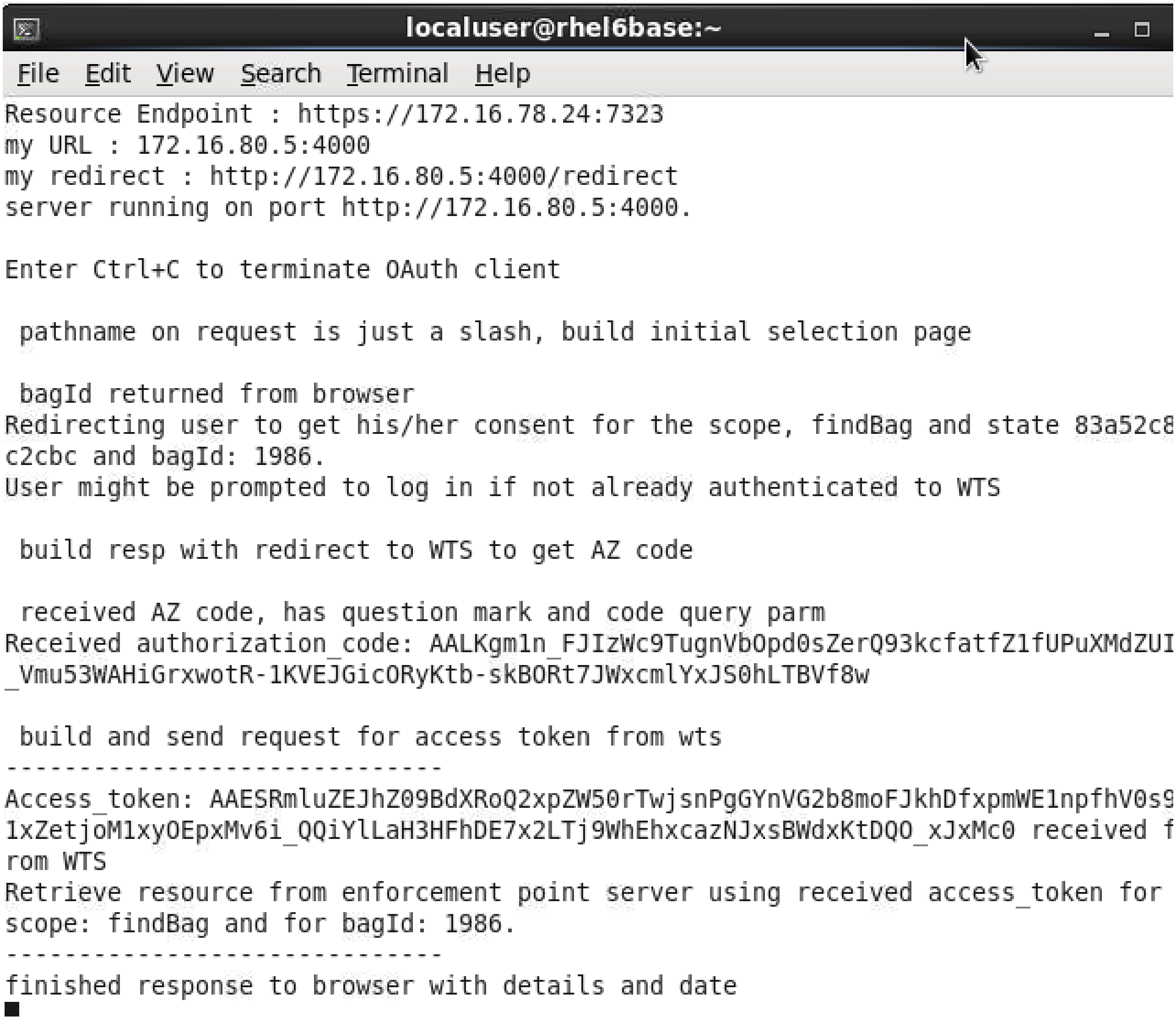
Browser

Details on the requested bag:

Status is retrieved at 9:00

16. JSON structure is rewritten to HTML and returned to browser

__ 1. If you look at the console log, you see some of the underlying activity for `oauth-client.js`:



```
Resource Endpoint : https://172.16.78.24:7323
my URL : 172.16.80.5:4000
my redirect : http://172.16.80.5:4000/redirect
server running on port http://172.16.80.5:4000.

Enter Ctrl+C to terminate OAuth client

 pathname on request is just a slash, build initial selection page

 bagId returned from browser
Redirecting user to get his/her consent for the scope, findBag and state 83a52c8
c2cbc and bagId: 1986.
User might be prompted to log in if not already authenticated to WTS

 build resp with redirect to WTS to get AZ code

 received AZ code, has question mark and code query parm
Received authorization_code: AALKgm1n_FJIzWc9TugnVbOpd0sZerQ93kcfatfZ1fUPuXMdZUI
_Vmu53WAHiGrxwotR-1KVEJGicORyKtb-skBORt7JWxcmlYxJS0hLTBVf8w

 build and send request for access token from wts
------------------------------
Access_token: AAESRmluZEJhZ09BdXRoQ2xpZW50rTwjsnPgGYnVG2b8moFJkhDfxpmWE1npfhV0s9
1xZetjoM1xyOEpxMv6i_QQiYlLaH3HFhDE7x2LTj9WhEhxcazNJxsBWdxKtDQO_xJxMc0 received f
rom WTS
Retrieve resource from enforcement point server using received access_token for
scope: findBag and for bagId: 1986.
------------------------------
finished response to browser with details and date
■
```

__ 2. You can go back to the browser and initialize another request for a different bag.

Optional

You can perform several activities to see more of what is happening within the calls between `oauth-client.js` and the services on the BaggageServiceProxy. If you rerun the testing, you might want to clear the browser history so that the login challenge is presented again.

__ 1. Enable the probe on the web token service. Examine the incoming URL, the headers, any body content, and resource services. Also, examine the outgoing information.

__ 2. Enable the probe on the resource server. Review the incoming request from `oauth-client.js`, and the outgoing request to the BaggageServiceProxy. Look at the outgoing HTTP headers. You should notice a ResourceOwner header with a value of "`CurbsideService`". "`addOwnerToHeader.xsl`" added this header and value. Why is the resource owner identified as "`CurbsideService`"? The reason is because in `FLY_AAAinfo.xml` the output credential for "student/passw0rd" is "`CurbsideService`".

__ 3. Enable the probe on the BaggageServiceProxy. Observe the incoming request and its response.

__ 4. Update the OAuth client profile for myOAuthClient. On the **Advanced** tab, create an HTTP header for the Client ID. Send a new request to the OAuth client. Look in the probe for BaggageServiceProxy and find the new HTTP header that contains the Client ID.

__ 5. Press Ctrl+C in the terminal window to stop the OAuth server code.

End of exercise

Exercise review and wrap-up

In this exercise, you defined an OAuth client profile that represented an OAuth client. You specified this client profile as a member of an OAuth client group that is referenced in AAA policies. You created a AAA policy that specifies HTTP basic authorization and OAuth, which you added to a web token service. You created another AAA policy that specifies OAuth to validate an access token, which you used in a resource server. Finally, you tested the configuration by using a browser to interact with an OAuth client that is running under a Node.js web server.

Exercise 3. Implementing an OIDC client

Estimated time

01:00

Overview

In this exercise, you configure and test an OIDC client.

Objectives

After completing this exercise, you should be able to:

- Configure an OIDC client

Introduction

OpenID Connect (OIDC) provides the authentication capability that OAuth 2.0 lacked. It adds an authentication layer on top of OAuth 2.0's authorization framework. DataPower provides multiple options and objects to support OIDC.

The purpose of this exercise is to create an authentication gateway to a back-end baggage information application. You use the OIDC protocol to support a social login authentication requirement.

You create a multi-protocol gateway (MPGW) that acts as the OIDC client. You also create a web token service that acts as the OIDC social login provider.

Requirements

To complete this exercise, you need:

- Access to the DataPower BaggageServiceProxy
- Completion of the previous exercises (see the Preface section in the exercise instructions for details)
- The BaggageStatusMockService web service that runs on the DataPower BaggageServiceProxy in the `FLYServices` domain
- Access to the *`<lab_files>`* directory

Exercise instructions

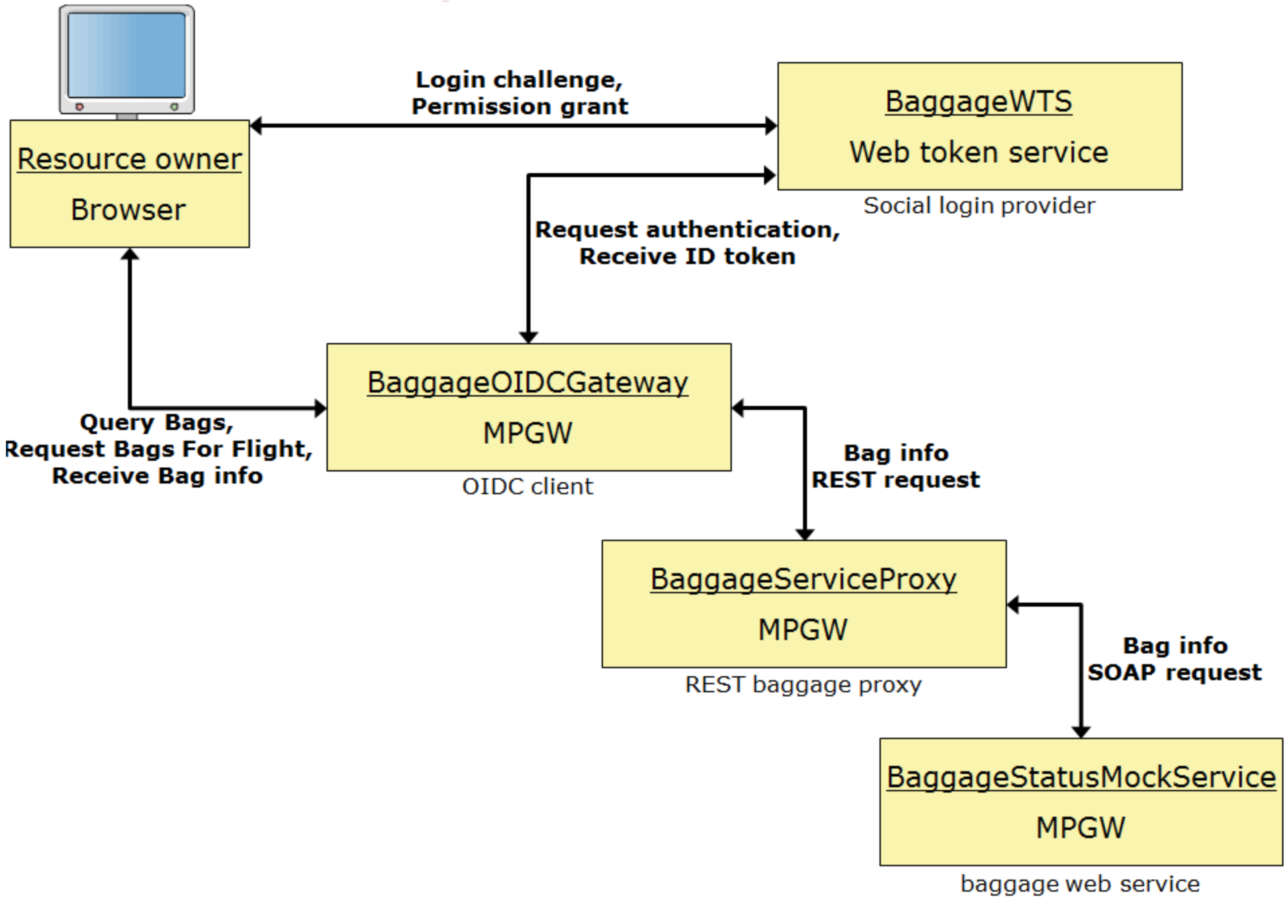
Preface

- Remember to use the domain and port address that you were assigned in the exercise setup. *Do not* use the default domain.
- This exercise depends on the completion of Exercise 1: Using DataPower to implement REST services.
- The references in exercise instructions refer to the following values:
    - *<lab_files>*: Location of the student lab files. Default location is: `/usr/labfiles/dp/`
    - <image_ip>: IP address of the student image (use `/sbin/ifconfig` from a terminal window to obtain value).
    - *<dp_internal_ip>*: IP address of the DataPower BaggageServiceProxy development and administrative functions that are used by internal resources such as developers.
    - *<dp_public_ip>*: IP address of the public services on the BaggageServiceProxy that is used by customer and clients.
    - *<dp_WebGUI_port>*: Port of the WebGUI. The default port is `9090`.
    - *<nn>*: Assigned student number. If no instructor exists, use “`01`”.
    - *<studentnn>*: Assigned user name and user account. If no instructor exists, use “`student01`”.
    - *<studentnn_password>*: Account password. In most cases, the initial value is the same as the user name. You are prompted to create a password on first use. Write it down.
    - *<studentnn_domain>*: Application domain that the user account is assigned to. If no instructor exists, use “`student01_domain`”.
    - *<FLY_baggage_port>*: Port number that the back-end BaggageServices web services listen on. The default port is 2068.
    - *<mpgw_baggage_port>*: 12*nn*9, where “*nn*” is the two-digit student number. This port number is the listener port of the BaggageServiceProxy that mediates between the REST or JSON client and the Baggage Services back-end application.
    - *<oidc_provider_port>*: 7*nn*5, where “*nn*” is the two-digit student number. This port number is the listener port of the web token service that acts as a social login provider.
    - *<oauth_ep_port>*: 7*nn*3, where “*nn*” is the two-digit student number. This port number is the listener port of the multi-protocol gateway that acts as the OAuth enforcement point for the resource server.
    - *<oidc_client_port>*: 7*nn4*, where “*nn*” is the two-digit student number. This port number is the listener port of the multi-protocol gateway that acts as the OIDC client.

3.1. The OIDC topology for this exercise

For this exercise, you are implementing an OIDC client. It is used to support agents at the baggage claim stations. A browser at the station can issue a request to “query bags”. This request goes to the BaggageOIDCGateway MPGW. It returns a page that allows for the entry of a bag ID. The agent enters a bag ID. That request initiates a “request bags for flight” request to retrieve the details on a specific bag. The system designers wanted to secure the access from the agents by using the OIDC protocol and a social login provider:

- The resource owner uses a browser as its user agent.
- The requests from the users go to the BaggageOIDCGateway MPGW. It interacts with the social login provider to secure access, and with the BaggageServiceProxy to retrieve the bag information.
- The social login provider is the BaggageWTS web token service. It uses a AAA information XML file to authenticate users.
- The BaggageServiceProxy MPGW receives REST requests for bag information, and converts them to SOAP requests for the actual baggage service BaggageStatusMockService.
- The back-end resource application is the BaggageStatusMockService MPGW. It uses the SOAP request to retrieve the bag information. For this example, the resource application uses the bag ID that the user selected in the browser to return the current details on the bag, and the time that the response was created.



3.2. Configure the objects to create the social login provider

In this section, you configure the objects that are needed for a social login provider.

The DataPower service type that supplies a social login provider solution is a web token service (WTS). The WTS refers to a AAA policy object to authenticate the users. Because this approach is an OAuth-based implementation, the AAA policy refers to an OAuth Client Profile object. And because this approach is also OIDC, the OAuth Client Profile object refers to a JWT Generator object. This object creates a JWT that is returned as part of the OIDC interactions.

__ 1. Log on to the DataPower WebGUI. Ensure that you are in your student domain.

__ 2. Create the **JWT Generator** object.

__ a. On the **Control Panel**, enter `jwt` in the Search field.

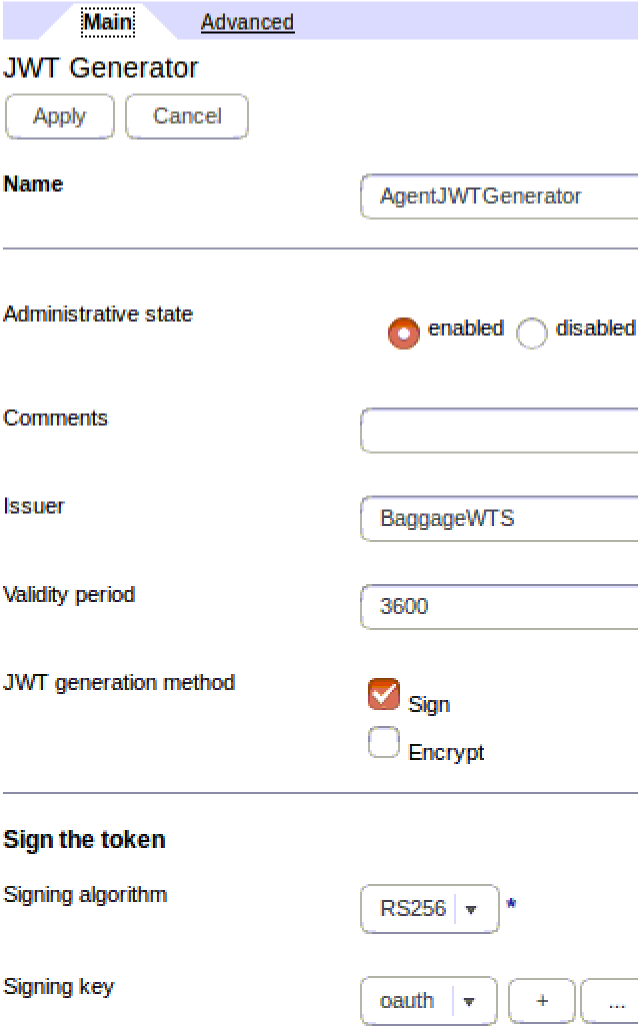
__ b. Select **JWT Generator** from the resulting list.

__ c. Click **Add** on the JWT Generator catalog list.

__ d. On the **Main** tab, enter:

- Name: `AgentJWTGenerator`
- Issuer: `BaggageWTS`
- Validity period: `3600`
- JWT generation method: `Sign` (enable)
- Signing algorithm: `RS256`

- Signing key: `oauth`



JWT Generator

Name: AgentJWTGenerator

Administrative state: ● enabled ○ disabled

Comments:

Issuer: BaggageWTS

Validity period: 3600

JWT generation method: ☑ Sign ☐ Encrypt

Sign the token

Signing algorithm: RS256 *

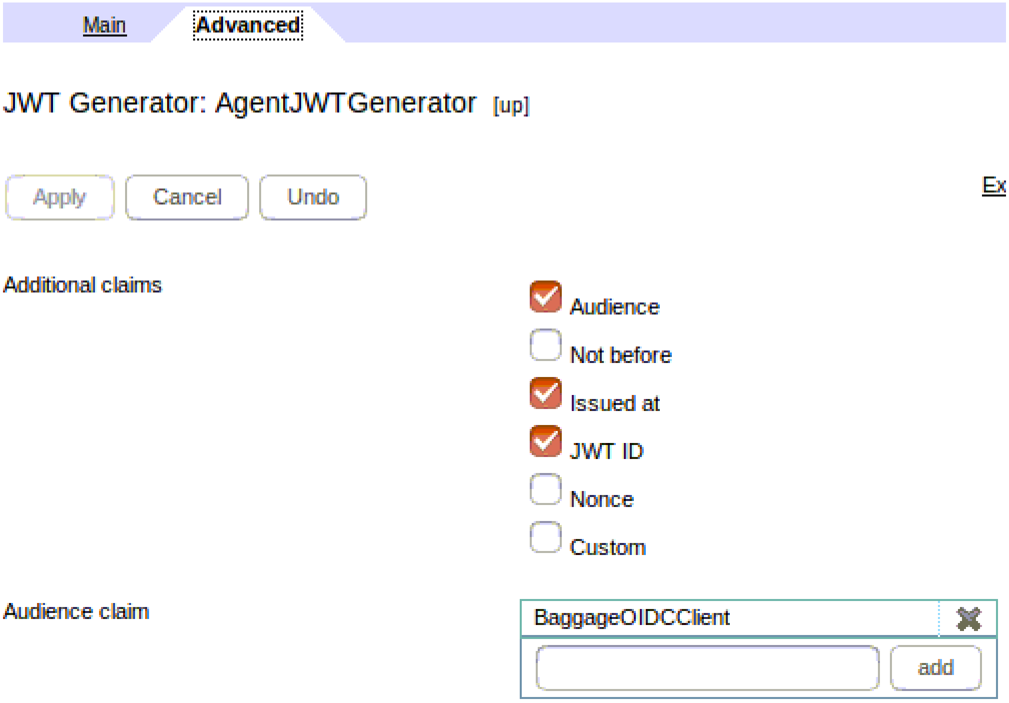
Signing key: oauth

__ e. Click the **Advanced** tab.

__ f. For the Additional claims, select **Audience**, **Issued at**, and **JWT ID**.

__ g. For the Audience claim, enter **BaggageOIDCClient**, and click **add**.

__ h. Click **Apply**.



JWT Generator: AgentJWTGenerator [up]

Additional claims

☑ Audience
□ Not before
☑ Issued at
☑ JWT ID
□ Nonce
□ Custom

Audience claim: BaggageOIDCClient

__ 3. Create the **OAuth Client Profile** object.

__ a. On the **Control Panel**, enter `oauth` in the Search field.

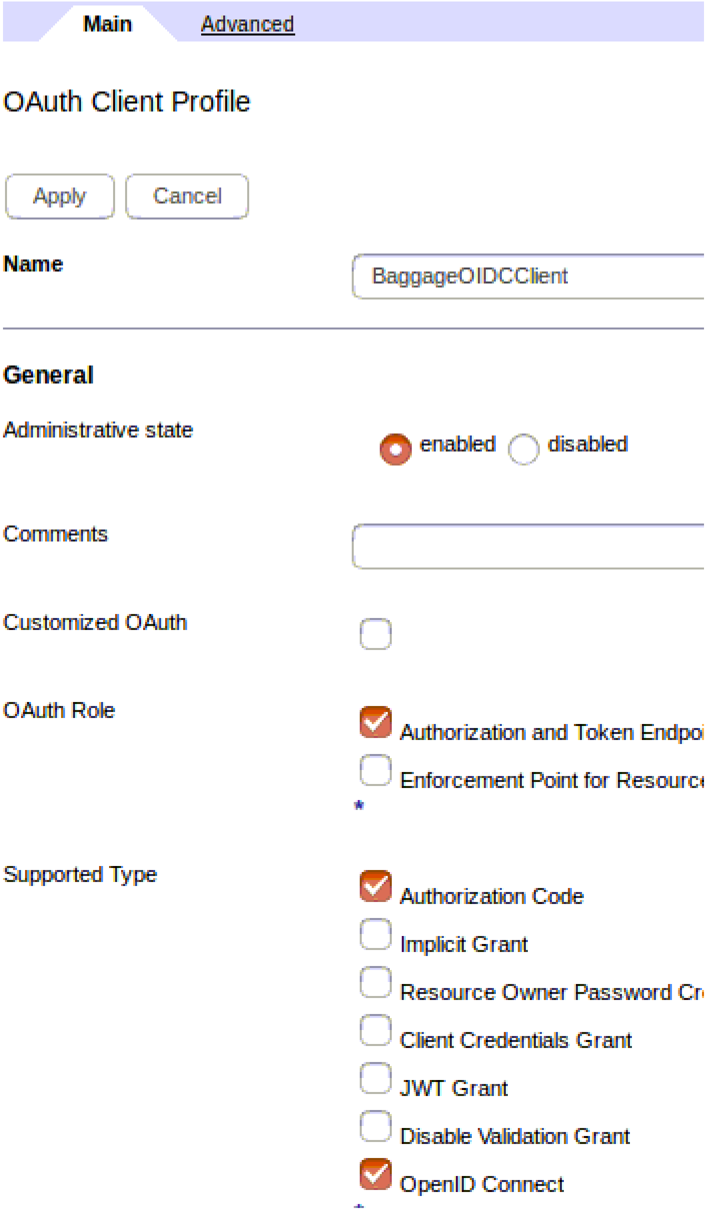
__ b. Select **OAuth Client Profile** from the resulting list.

__ c. Click **Add** on the OAuth Client Profile catalog list.

__ d. On the **Main** tab, enter a Name of `BaggageOIDCClient`

__ e. For OAuth Role, enable **Authorization and Token Endpoints** only.

__ f. For Supported Type, select **Authorization Code** and **OpenID Connect**.



OAuth Client Profile

Name: BaggageOIDCClient

General

Administrative state: ● enabled ○ disabled

Comments:

Customized OAuth: □

OAuth Role:
☑ Authorization and Token Endpoi
□ Enforcement Point for Resourc

Supported Type:
☑ Authorization Code
□ Implicit Grant
□ Resource Owner Password Cr
□ Client Credentials Grant
□ JWT Grant
□ Disable Validation Grant
☑ OpenID Connect

__ g. Several fields further down, disable **Generate Client Secret**.

__ h. For Client Type, select **Confidential**.

__ i. For Authentication Method, select **Client Secret**.

__ j. For Client Secret, enter `passw0rd`.

__ k. For Scope, enter `RequestBagInfo`.

Information

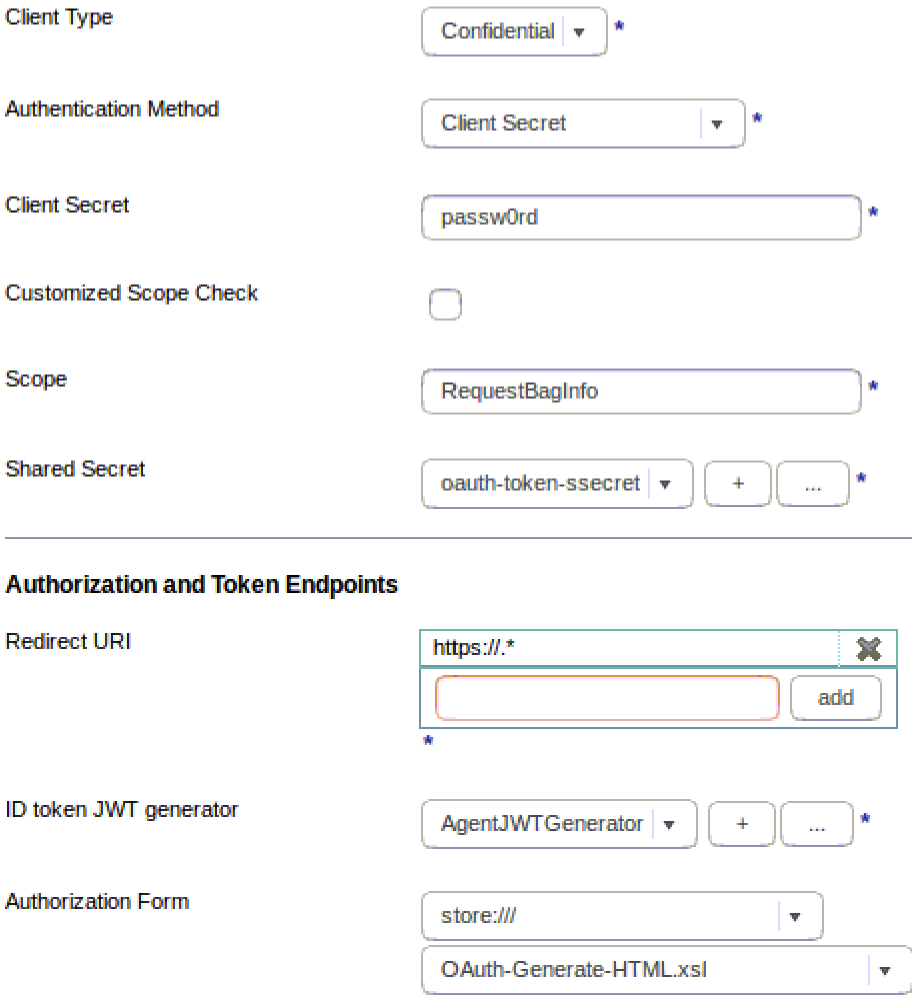
Because **OpenID Connect** is selected for a Supported Type, **openid** is automatically added to the Scope field.

__ l. For Shared Secret, select `oauth-token-ssecret`.

__ m. For Redirect URI, enter `https://.*` and click **add**.

__ n. For ID token JWT generator, select **AgentJWTGenerator**.

__ o. For the Authorization Form, use the default **store:///OAuth-Generate-HTML.xsl**.



Client Type	Confidential *
Authentication Method	Client Secret *
Client Secret	passw0rd *
Customized Scope Check	□
Scope	RequestBagInfo *
Shared Secret	oauth-token-ssecret + ... *

Authorization and Token Endpoints

Redirect URI	https://.* ✖ add *
ID token JWT generator	AgentJWTGenerator + ... *
Authorization Form	store:/// OAuth-Generate-HTML.xsl

__ p. Click **Apply**.

__ 4. Define the **OAuth client group** object.

__ a. On the **Control Panel**, enter `oauth` in the Search field above the navigation bar.

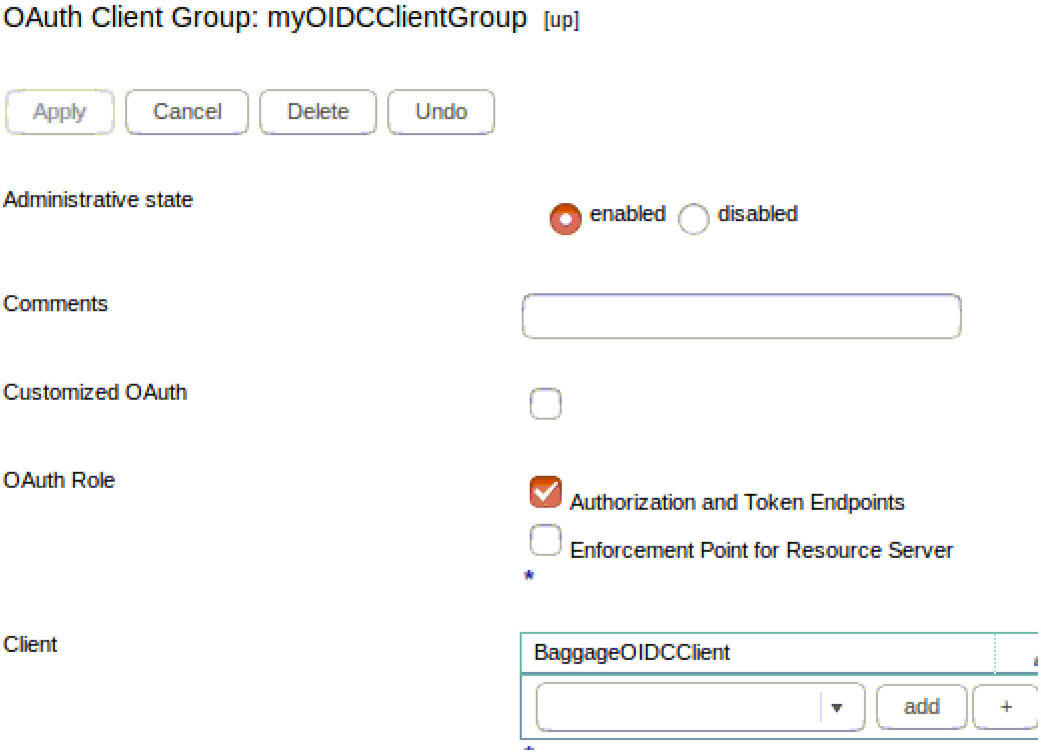
__ b. Click **OAuth Client Group**.

__ c. On the Configure OAuth Client Group catalog list page, click **Add**.

__ d. Enter the Name as: `myOIDCClientGroup`

__ e. Select the OAuth Roles of **Authorization and Token Endpoints** only.

__ f. For the Client list, select **BaggageOIDCClient** and click **add**.



OAuth Client Group: myOIDCClientGroup [up]

Administrative state ● enabled ○ disabled

Comments

Customized OAuth □

OAuth Role ☑ Authorization and Token Endpoints
□ Enforcement Point for Resource Server
*

Client BaggageOIDCClient

__ g. Click **Apply**.

__ 5. Define the **AAA policy** object that is used in the web token service.

__ a. On the **Control Panel**, enter `aaa` in the Search field above the navigation bar.

__ b. Click **AAA Policy**.

__ c. On the Configure AAA Policy catalog list page, click **Add**.

__ d. On the **Main** tab of the Configure AAA Policy page, enter a Name of **ProviderAAAPolicy**.

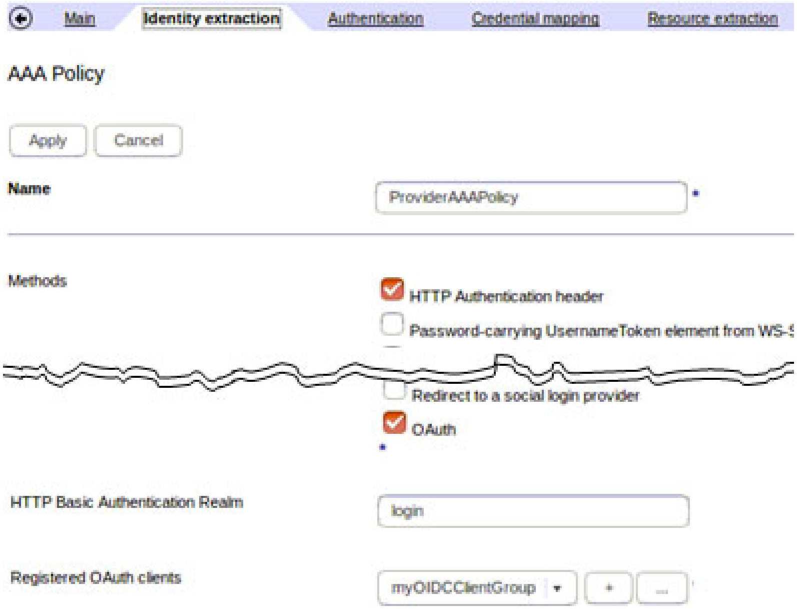
Information

When you created a AAA policy as part of a AAA action in a policy editor, you used the AAA policy wizard that is part of the policy editor. In this case, you are directly accessing the stand-alone AAA policy object. Rather than being guided through the configuration of a AAA policy object, you are presented with a set of tabs that deal with specific aspects of a AAA policy. You must “lead” yourself through the configuration.

__ e. On the **Identity extraction** tab, enable **HTTP Authentication header** and **OAuth** as methods.

__ f. Because “HTTP Authentication header” was enabled, the “HTTP Basic Authentication Realm” field is displayed. Leave it at its default value of **login**.

__ g. Because “OAuth” was enabled, the “Registered OAuth clients” field is displayed. Select **myOIDCClientGroup**.



AAA Policy

Name: ProviderAAAPolicy *

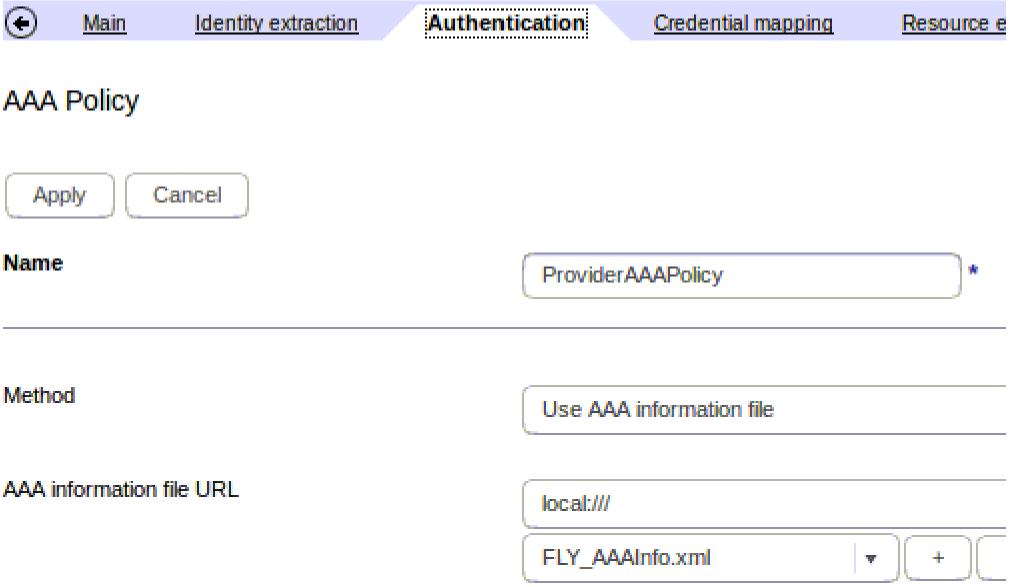
Methods
☑ HTTP Authentication header
□ Password-carrying UsernameToken element from WS-S
□ Redirect to a social login provider
☑ OAuth

HTTP Basic Authentication Realm: login

Registered OAuth clients: myOIDCClientGroup

__ h. On the **Authentication** tab, select a Method of **Use AAA information file**.

__ i. Select a AAA information file URL of **local:///FLY_AAAInfo.xml**.



AAA Policy

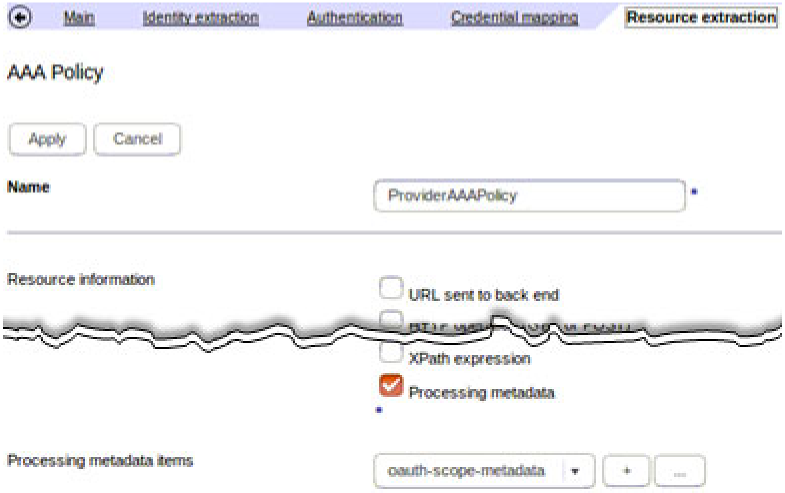
Name: ProviderAAAPolicy *

Method: Use AAA information file

AAA information file URL: local:/// FLY_AAAInfo.xml

__ j. On the **Resource extraction** tab, enable **Processing metadata** as the resource information.

__ k. Select the Processing metadata items as **oauth-scope-metadata**.



Name: ProviderAAAPolicy

Resource information:
□ URL sent to back end
□ XPath expression
☑ Processing metadata

Processing metadata items: oauth-scope-metadata

__ l. For the **Authorization** tab, leave the Method at its default value of **Allow any authenticated client**.

__ m. The AAA policy configuration is complete. Click **Apply**.

__ 6. Define the **web token service** object that acts as the social login provider.

__ a. On the **Control Panel**, enter `web tok` in the Search field above the navigation bar.

__ b. Click **New Web Token Service**.

__ c. Enter a Web Token Service Name of **BaggageWTSProvider**.

__ d. Click **Next**.

__ e. Enter an IP of *<dp_public_ip>* and a Port *<oidc_provider_port>*.

__ f. Select an SSL **Server Profile**.

__ g. Select an SSL Profile of **oauth-ssl-server**.

__ h. Click **Add** on the right side of the page.

__ i. Click **Next**.

__ j. On the next page, select a AAA Policy of **ProviderAAAPolicy**.

__ k. Click **Next**.

__ l. Click **Commit**.

__ m. Click **Done**.

__ 7. Click **Save Configuration** at the top of the page.

3.3. Configure the support objects for an OIDC client

In this section, you create the objects that are needed to implement your OIDC client. The actual OIDC client is implemented as an MPGW. As part of the MPGW configuration, you need a AAA policy for the service policy, and a Social Login Policy object to represent the interactions with the provider. A JWT Validator object is used by the social login policy to validate the JWT that is returned from the provider.

__ 1. Create the objects that are needed for the client side of the SSL connection to the web token service.

__ a. Create a Crypto Validation Credential named `oauthValCred` that points to the **oauth** certificate.

__ b. Create an SSL Client Profile named `oauth-ssl-client` that points to the **oauthValCred** validation credential.

__ 2. Create the **JWT Validator** object.

__ a. On the **Control Panel**, enter `jwt` in the Search field.

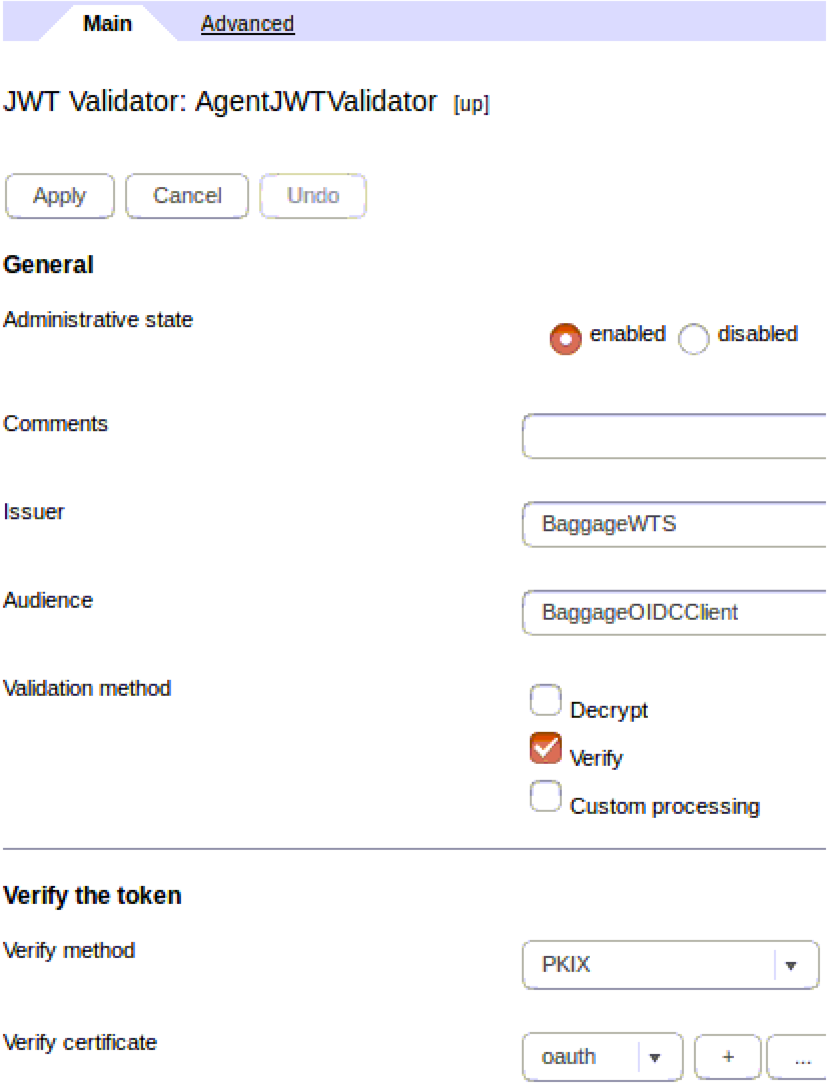
__ b. Select **JWT Validator** from the resulting list.

__ c. Click **Add** on the JWT Validator catalog list.

__ d. On the **Main** tab, enter:

- Name: `AgentJWTValidator`
- Issuer: `BaggageWTS`
- Audience: `BaggageOIDCClient`
- Validation method: `Verify` (enable)
- Verify method: `PKIX`

- Verify certificate: `oauth`



JWT Validator: AgentJWTValidator [up]

General

Administrative state ● enabled ○ disabled

Comments

Issuer BaggageWTS

Audience BaggageOIDCClient

Validation method
□ Decrypt
☑ Verify
□ Custom processing

Verify the token

Verify method PKIX

Verify certificate oauth

__ e. Click **Apply**.

__ 3. Create the **Social Login Policy** object.

__ a. On the **Control Panel**, enter `social` in the Search field.

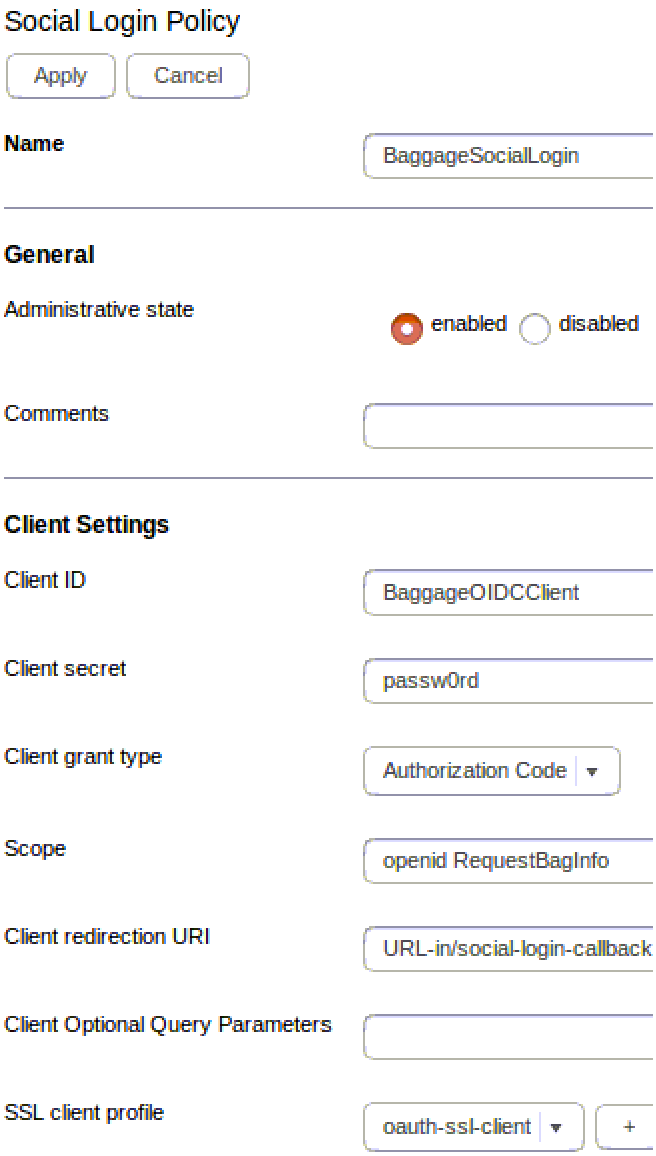
__ b. Select **Social Login Policy** from the resulting list.

__ c. Click **Add** on the Social Login Policy catalog list.

__ d. On the **Main** tab, enter:

- Name: `BaggageSocialLogin`
- Client ID: `BaggageOIDCClient`
- Client secret: `passw0rd`
- Client grant type: `Authorization Code`
- Scope: `openid RequestBagInfo`
- Client redirection URI: `URL-in/social-login-callback`

- SSL client profile: `oauth-ssl-client`



Name: BaggageSocialLogin

General

Administrative state: ● enabled ○ disabled

Comments:

Client Settings

Client ID: BaggageOIDCClient

Client secret: passw0rd

Client grant type: Authorization Code

Scope: openid RequestBagInfo

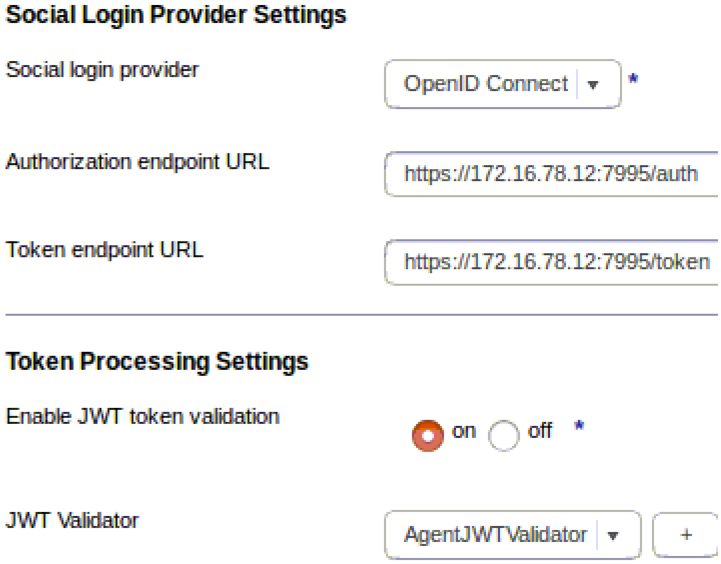
Client redirection URI: URL-in/social-login-callback

Client Optional Query Parameters:

SSL client profile: oauth-ssl-client +

- Social login provider: `OpenID Connect`
- Authorization endpoint URL: `https://<dp_public_ip>:<oidc_provider_port>/auth`
- Token endpoint URL: `https://<dp_public_ip>:<oidc_provider_port>/token`

- JWT Validator: `AgentJWTValidator`



Social Login Provider Settings

Social login provider	OpenID Connect ▾ *
Authorization endpoint URL	https://172.16.78.12:7995/auth
Token endpoint URL	https://172.16.78.12:7995/token

Token Processing Settings

Enable JWT token validation	● on ○ off *
JWT Validator	AgentJWTValidator ▾ +

__ e. Click **Apply**.

__ 4. Configure the **AAA policy** to use in the OIDC client MPGW.

__ a. On the **Control Panel**, enter `aaa` in the Search field above the navigation bar.

__ b. Click **AAA Policy**.

__ c. On the Configure AAA Policy catalog list page, click **Add**.

__ d. Define the AAA policy object:

__ e. Enter the Name as: `oidcAAApolicy`

__ f. Click the **Identity extraction** tab.

__ g. Select the Method: **Redirect to a social login provider**.

__ h. Selecting **Redirect** causes the Social login policy choice to display. Select the social login policy that you defined in a previous step: **BaggageSocialLogin**.

__ i. Click the **Authentication** tab.

__ j. For Method, select **Use verified JWT, access token, or ID token**.

__ k. Click the **Resource extraction** tab.

__ l. Select **URL sent to back end**.

__ m. Click the **Authorization** tab.

__ n. Select a Method of **Allow any authenticated client**.

__ o. Click **Apply** to save the AAA policy for use in the OIDC client.

__ 5. Configure the HTTPS front side handler that is used by the OIDC client MPGW.

__ a. On the **Control Panel**, enter `https` in the Search field above the navigation bar.

__ b. Click **HTTPS Front Side Handler**.

__ c. On the Configure HTTPS Front Side Handler catalog list page, click **Add**.

__ d. Enter the Name as: `https_OIDCClient_7nn4`

__ e. Set the Local IP address to <*dp_public_ip*>.

__ f. Set the Port to <*oidc_client_port*>.

__ g. For the Allowed methods and versions, select **GET method** and **URL with ..** along with the other default selections.

Information

The **URL with ..** choice is required if you use Google as the provider.

__ h. Set the SSL server type to **Server Profile**.

__ i. Set the SSL server profile to **oauth-ssl-server**.

__ j. Click **Apply**. Since the handler is not yet associated with a service, it is in the “down” state.

3.4. Define the OIDC client MPGW

In this section, you define a multi-protocol gateway that acts as the OIDC client. It also acts as a proxy to the back-end application.

__ 1. On the **Control Panel**, enter `multi` in the Search field above the navigation bar.

__ 2. Click **Edit Multi-Protocol Gateway**.

__ 3. On the Configure a Multi-Protocol Gateway catalog list page, click **Add**.

__ 4. On the Configure a Multi-Protocol Gateway page, enter the name as: `BaggageOIDCGateway`

__ 5. Enter the Default Backend URL as: `http://`*`<dp_public_ip>`*`:`*`<mpgw_baggage_port>`*. This URL is pointing to the BaggageServiceProxy in your domain.

__ 6. Set the Request Type as **Non-XML**.

__ 7. Set the Response Type as **Non-XML**.

__ 8. Set the Front Side Protocol to **https_OIDCClient_7nn4**.

__ 9. Start the Multi-Protocol Gateway Policy configuration:

__ a. Click the new (**+**) button.

__ b. Enter the Policy Name as: `BaggageOIDCPolicy`

__ c. Click **Apply Policy**.

__ 10. Configure the request rule that builds the bag ID selection page:

__ a. Click **New Rule**.

__ b. Set the direction as **Client to Server**.

__ c. Double-click the **Match** action to configure it.

__ d. Configure a Matching Rule that matches the URL `/QueryBags`. This rule matches an HTTP request for the bag ID selection page. Give it any name that you want.

__ e. Drag a **GatewayScript** action to the rule configuration path.

__ f. Configure it by uploading **BuildBagQueryPage.js** from *`<lab_files>`*`:/oidc`. Put it in the **local**: directory.

__ g. Drag an **Advanced** action to the rule configuration path.

__ h. Configure it as a **Set Variable** action.

__ i. Set the **var://service/mpgw/skip-backside** to **1**.

__ j. Drag a **Results** action to the path.

__ k. Configure the action to pass the output context of the GatewayScript action to the **OUTPUT** output context.

__ l. Click **Apply Policy** to save the rule and processing policy.

__ 11. Configure the request rule that handles a favicon request.

__ a. Click **New Rule**.

__ b. Set the direction as **Client to Server**.

__ c. Double-click the **Match** action to configure it.

__ d. Configure a Matching Rule that matches the URL `/favicon.ico`. This rule matches an HTTP request for the favicon that might show in the address bar of the browser. Give it any name that you want.

__ e. Drag an **Advanced** action to the rule configuration path.

__ f. Configure it as a **Set Variable** action.

__ g. Set the **var://service/mpgw/skip-backside** to **1**.

__ h. Drag a **Results** action to the path.

__ i. Configure the action to pass the **INPUT** input context to the **OUTPUT** output context.

__ j. Click **Apply Policy** to save the rule and processing policy.

__ 12. Configure the request rule that acts as an OIDC client.

__ a. Click **New Rule**.

__ b. Set the direction as **Client to Server**.

__ c. Double-click the **Match** action to configure it.

__ d. Configure a Matching Rule that matches the URL `/RequestBagInfo*`. This rule matches an HTTP request for the bag information. Give it any name that matches on all URLs.

__ e. Drag an **Advanced** action to the path.

__ f. Configure this action as a **Convert Query Params to XML** action. This action converts non-XML encoded input (an HTTP POST of HTML form or URI parameters) into an equivalent XML message.

__ g. No further configuration is needed on this action, so click **Done**.

__ h. Drag a **AAA** action to the path.

__ i. Configure it with the AAA policy that you created earlier: **oidcAAAPolicy**.

__ j. Click **Done**.

__ k. Drag a **GatewayScript** action to the rule.

__ l. For the GatewayScript File, upload **setNewPath.js** from *`<lab_files>`*`/oidc` into the **local:** directory. This GatewayScript retrieves the original URL from the AAA context and reformats it into a RESTful request that the back end expects.

Important

Because of the way that the AAA policy manages the OIDC redirects, the original path and query string does not come out of the AAA policy. The path and query string that is returned is from the callback from the provider. However, the “social-login-url-in” variable in the AAA policy’s context does contain the original URL. This GatewayScript shows the use of this variable.

__ m. Click **Done**.

__ n. Drag a **Results** action to the path.

__ o. Configure it to pass the output context of the GatewayScript action to the **OUTPUT** output context.

__ p. Click **Apply Policy**.

__ 13. Configure the response rule:

__ a. Click **New Rule**.

__ b. Set the direction as **Server to Client**.

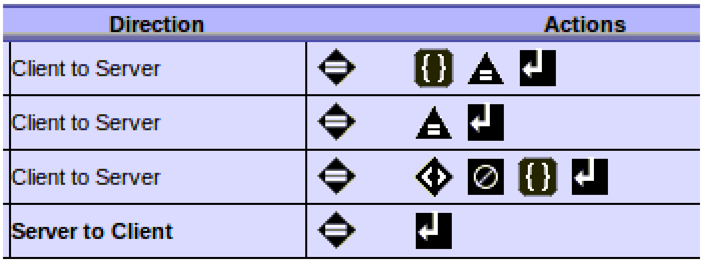
__ c. Double-click the **Match** action to configure it.

__ d. Configure a Matching Rule that matches on all URLs.

__ e. Drag a **Results** action to the path.

__ f. Configure the action to pass the **INPUT** input context to the **OUTPUT** output context.

__ g. Click **Apply Policy**.



Direction		Actions
Client to Server	◆	{} ▲ ↵
Client to Server	◆	▲ ↵
Client to Server	◆	◈ ⊘ {} ↵
Server to Client	◆	↵

__ h. Close the policy editor window.

__ 14. On the **Advanced** tab, set the Process Messages Whose Body Is Empty to **on**.

__ 15. Click **Apply** to save the MPGW.

__ 16. Click **Save Configuration**.

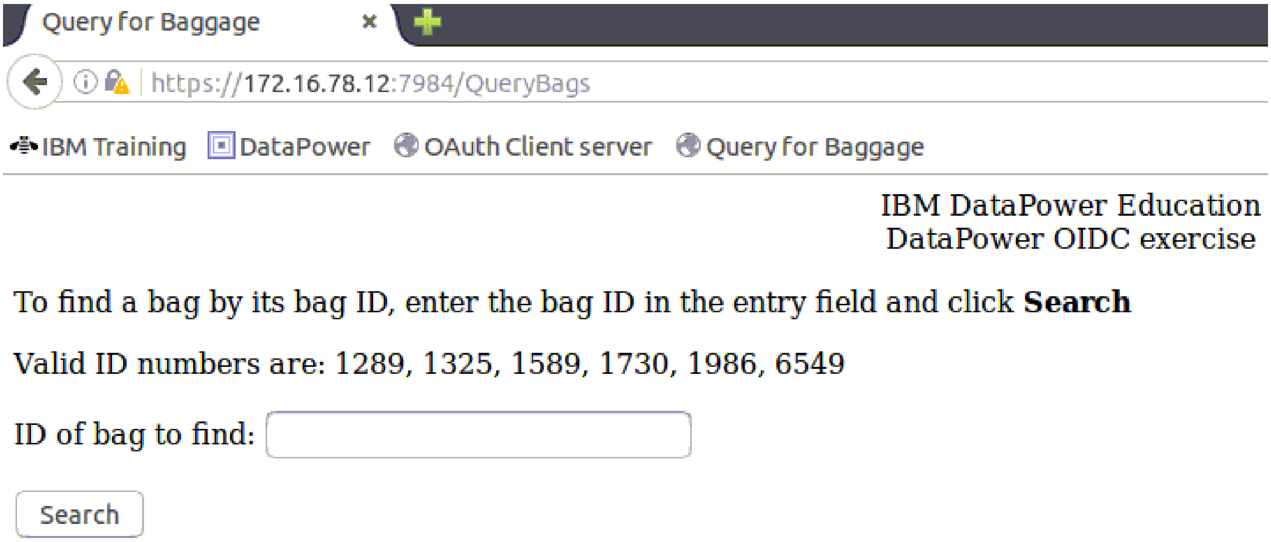
3.5. Test the solution

In this section, you are the resource owner and use the browser as the user agent.

__ 1. Open a browser. Enter a URL of `https://<dp_public_ip>:<oidc_client_port>/QueryBags`

__ 2. If this is the first access of this browser to the BaggageOIDCGateway, you should get an "untrusted connection" warning. Accept the exception.

__ 3. The BaggageOIDCGateway, the OIDC client, returns a form.



IBM DataPower Education
DataPower OIDC exercise

To find a bag by its bag ID, enter the bag ID in the entry field and click **Search**

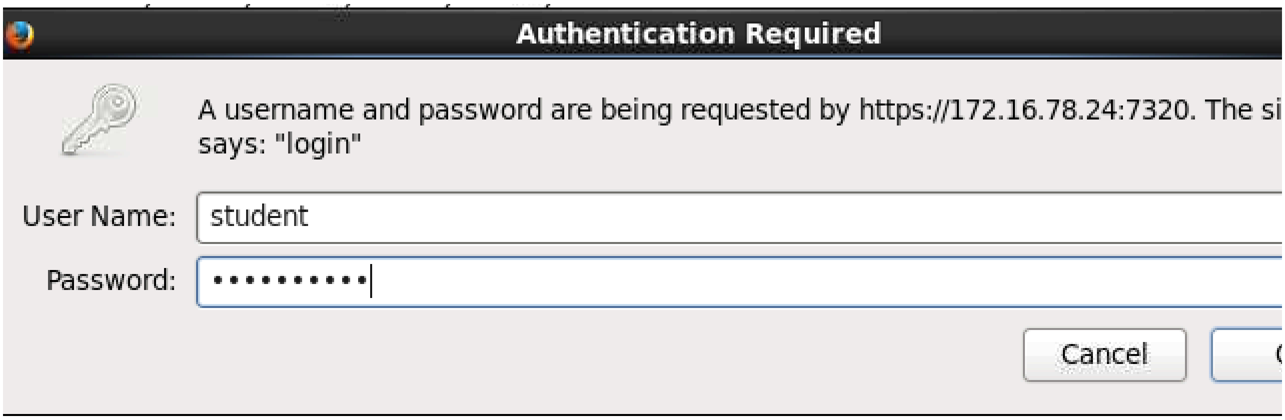
Valid ID numbers are: 1289, 1325, 1589, 1730, 1986, 6549

ID of bag to find:

__ 4. Enter a valid bag ID and click **Search**.

__ 5. If this attempt is the first time for the browser to access the web token service, you get an "untrusted connection" warning. Accept the exception.

__ 6. A redirection occurs from the OIDC client to the provider, the web token service. If this attempt is the first time for the browser to connect to the web token service (via the redirection), you are prompted to authenticate. Enter `student` for the **User Name** and `passw0rd` for the **Password**:



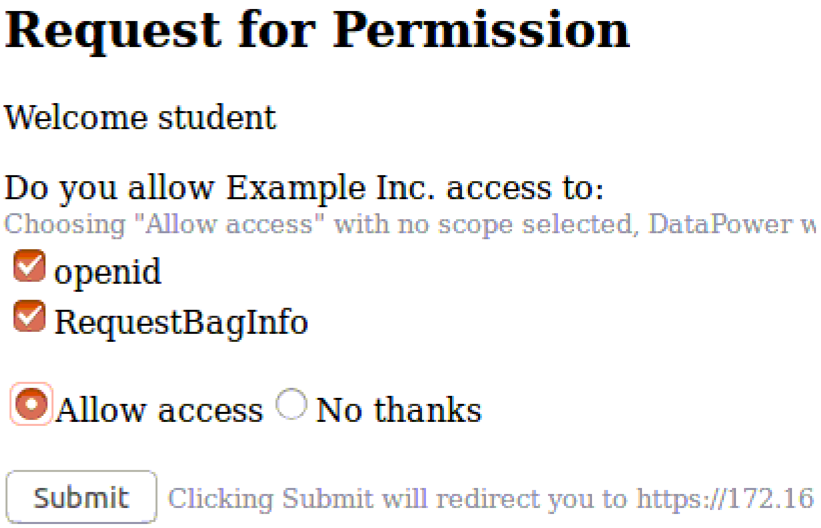
A username and password are being requested by https://172.16.78.24:7320. The si
says: "login"

User Name: student

Password: ••••••••••

__ 7. Click **OK**.

__ 8. Now the web token service asks for your permission for the OIDC client to access a resource within a scope:



Request for Permission

Welcome student

Do you allow Example Inc. access to:

Choosing "Allow access" with no scope selected, DataPower w

☑ openid

☑ RequestBagInfo

● Allow access ○ No thanks

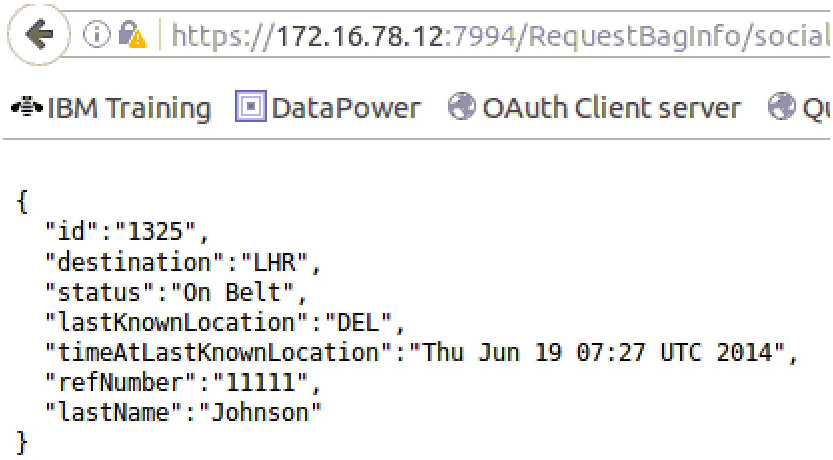
Clicking Submit will redirect you to https://172.16

__ 9. Verify that the scopes **openid** and **RequestBagInfo** are selected. Select **Allow access**.

__ 10. Click **Submit**.

__ 11. The web token service sends the authorization grant code to the OIDC client via another redirection. It uses the grant code and the web token service's token endpoint to request the access token and ID token (JWT) from the web token service. The access token and ID token (JWT) are returned to the OIDC client and validated. The OIDC client then reformats the original URL and calls the back-end application.

__ 12. The application response returns bag information to the OIDC client. The OIDC client returns the bag information to the browser.



https://172.16.78.12:7994/RequestBagInfo/social

```
{
  "id":"1325",
  "destination":"LHR",
  "status":"On Belt",
  "lastKnownLocation":"DEL",
  "timeAtLastKnownLocation":"Thu Jun 19 07:27 UTC 2014",
  "refNumber":"11111",
  "lastName":"Johnson"
}
```

Troubleshooting

If the request fails, review the DataPower logs. Common problems are incorrect IP addresses, and mismatches of the issuer and audience claims. If the `aaapolicy.js` script (internal GatewayScript that executes the AAA policy) fails, it might be because of an incorrect key length (less than 2048-bit).

__ 13. The OIDC client (MPGW) used the social login provider (web token service) to authenticate the user and get consent, and accessed the back-end application to get information on a specific bag.

Optional

You can perform several activities to see more of what is happening within the calls between the OIDC client and the social login provider. If you rerun the testing, you might want to clear the browser history so that the login challenge is presented again.

__ 1. Enable the probe on the BaggageOIDCGateway. Rerun the test. Notice that the request for the bag information goes through the request rule twice. The first time is to handle the bag information request from the browser. The second time is to handle the redirect from the web token service.

__ 2. For the first pass through the rule, examine the context variables after the AAA action completes. Look for the “social-login-url-in” variable. It does not exist yet. Look at the same point in the second pass through the request rule. The “social-login-url-in” variable now exists, and it contains the original request from the browser. Examine the service variables. Notice that the “URI” and “URL-in” and “URL-out” reflect the callback from the web token service, not the original request.

__ 3. Enable the probe on the web token service. Review the incoming requests from the OIDC client. Look at the context variables after the AAA action. Notice that the user credential is not “student01”, but “CurbsideService”. Why is the resource owner identified as “`CurbsideService`”? The reason is because in `FLY_AAAinfo.xml` the output credential for “student/passw0rd” is “`CurbsideService`”.

__ 4. Examine the **EI** AAA context variable in the BaggageOIDCGateway for the transaction after the scope permission has been given. Examine the <entry type=”social-login”> node. You can see the returned JWT.

__ 5. Enable the GatewayScript CLI debugger for the **setNewPath.js**. Examine the execution within the GatewayScript. If you are not familiar with the GAtewayScript debugger, you can look in the IBM DataPower Knowledge Center: https://www.ibm.com/support/knowledgecenter/SS9H2Y_7.5.0/com.ibm.dp.doc/debugger.html.

End of exercise

Exercise review and wrap-up

In this exercise, you defined an MPGW that acted as an OIDC client. You also created a web token service that acted as the social login provider for the OIDC client. You created several supporting objects such as a social login policy object, JWT generator and validator objects, and an OAuth client profile.

Appendix A. Exercise solutions

This appendix describes:

- The dependencies between the exercises and other tools.
- How to load the sample solution configurations for the various exercises. The solutions were exported from the appliance into a `.zip` file. You can import a sample solution into your domain.

Part 1: Dependencies

Certain exercises depend on previous exercises and on other resources, such as the need for the back-end application server to support service calls. The back-end application server is a multi-protocol gateway that runs in another application domain within the DataPower gateway.

Table 1. Dependencies

Exercise	Depends on exercise	Uses cURL	Uses SoapUI	Uses Baggage web service	Uses Booking web service
1: Configure authentication and authorization in a service		No	Yes	Yes	
2: Defining a three-legged OAuth scenario that uses DataPower services	1	No	No	Yes	
3: Implement an OIDC client	1	No	No	Yes	

If the class is using the standard images and setup, the LDAP server is running on the student image. The Baggage and Booking services are running as services on the DataPower gateway. Therefore, each student is using a different IP address for the student image. Assuming that each student has their own DataPower virtual gateway, each student also has different IP addresses for the gateway.

If the exercises are run in the IBM remote lab environment, like Skytap, the IP addresses might be the same for each student because each student has a unique entry point into the virtualized environment.

Part 2: Importing solutions

Note

The solution files use port numbers that might already be in use. You must change the port numbers of the imported service. You might also find it necessary to update the location of the back-end application server that provides the web services.

__ 1. Determine the `.zip` file to import from the following table:

Table 2. Exercise solution files

Exercise	Compressed solution file name

1: Configure authentication and authorization in a service	`AAA_AAA_xmlFile.zip,`
	`AAA_AAA_LDAP.zip`

2: Defining a three-legged OAuth scenario that uses DataPower services	`AAA_OAuth.zip`
3: Implement an OIDC client	`AAA_OIDC.zip`

__ a. The `.zip` file names begin with the naming convention `AAA_aaa`, where “aaa” represents the particular lab exercise.

__ b. To import a solution to begin a new exercise, import the solution for the previous exercise. Import the `.zip` solution file into your application domain.

__ c. From the **Control Panel**, in the vertical navigation bar, click **Administration > Configuration > Import Configuration**.

__ d. Make sure that the selection for **From** is **ZIP Bundle** and the selection for **Where** is **File**.

__ e. Click **Browse** and navigate to your respective `.zip` solution file.

__ f. Click **Next**.

__ g. In the next page, leave the files selected. Scroll down and click **Import**.

__ h. Make sure that the import is successful. Click **Done**.

__ 2. *Be sure to update the port numbers and application server location* to your local values. Because private keys (key files) are not exported, *you also must create keys and certificates*. In some exercise solutions, the key files are exported in the `local:` directory. After import, you move those files into the `cert:` directory.

__ 3. The lab exercises call one back-end web service, **Baggage Service**. This web service is in the FLY service domain. To do the labs on another DataPower gateway, be sure to import the `dev_FLYservices_domain.zip` file in the **FLYServices** domain.

Appendix B. Lab environment setup

The appendix instructs how to set up the lab environment, including:

- Defining the literal variable values in SoapUI
- Testing the Booking and Baggage web service back ends
- Identifying the IP address of your student image
- Populating a convenient table with all the required variables that are used in this course

Part 1: Configure the SoapUI variables for use

The SoapUI tool supports specification of properties to reduce the redundant entry of the same value for testing. For these exercises, the client testing usually accesses the public interface of the student-created DataPower services. Rather than requiring the students to constantly enter the same value, the public IP address of the appliance is configured as a SoapUI property.

__ 1. Obtain the required variables for this course. The variable information can be found in at least on one of the following locations, based on the type of the course you are taking:

- On the image desktop as a background
- On the image background from the SPVC you logged in to
- In an email you received with instructions for this course
- From your instructor if you are in a classroom (virtual or literal) environment
- In the exercise guide itself

The variables are:

- The DataPower appliance’s public IP address `<dp_public_ip>`
- The DataPower appliance’s internal IP address `<dp_internal_ip>`
- The (your) student image IP address `<image_ip>`
- Your student number (it is a two-digit number) `<nn>`

__ 2. Open SoapUI by using the icon on the desktop.

Attention

If you get a “new version available” message, close the message window and do not download or install any upgrades.

__ 3. Click **File > Preferences**.

__ 4. Click the **Global Properties** choice.

__ 5. Update the values for the following variables.

__ a. Double-click the **Value** cell for the **dp_internal_ip** property.

__ b. Replace the value (`x.x.x.x` or `1.2.3.4`) with the literal value of the *`<dp_internal_ip>`* address in the cell. That is, replace 1.2.3.4 with the IP address of the DataPower appliance that is being used for your class.

__ c. Click **Enter** while the cursor is in the cell. This action registers the new value.

__ d. Double-click the **Value** cell for the **dp_public_ip** property.

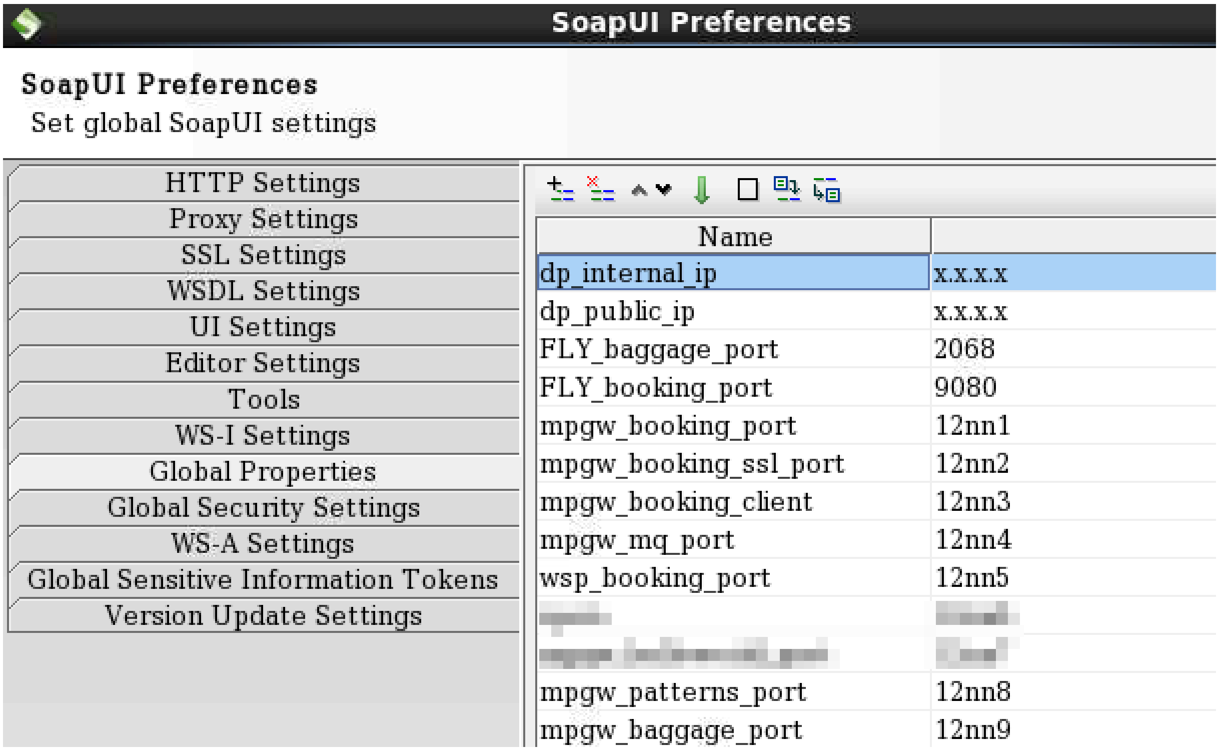
__ e. Replace the value (`x.x.x.x` or `1.2.3.4`) with the literal value of the *`<dp_public_ip>`* address in the cell. That is, replace 1.2.3.4 with the IP address of the DataPower appliance that is being used for your class.

__ f. Click **Enter** while the cursor is in the cell. This action registers the new value.

__ g. Double-click the **Value** cell for the **mpgw_booking_port** property.

__ h. Replace "*nn*" with your appropriate student number. For example, if you are `student 01`, the value for `mpgw_booking_port` of `12`*`nn`*`1` is updated to `12011`.

__ i. Click **Enter** while the cursor is in the Value cell. This action registers the new value.



Name	
dp_internal_ip	x.x.x.x
dp_public_ip	x.x.x.x
FLY_baggage_port	2068
FLY_booking_port	9080
mpgw_booking_port	12nn1
mpgw_booking_ssl_port	12nn2
mpgw_booking_client	12nn3
mpgw_mq_port	12nn4
wsp_booking_port	12nn5
[redacted]	[redacted]
[redacted]	[redacted]
mpgw_patterns_port	12nn8
mpgw_baggage_port	12nn9

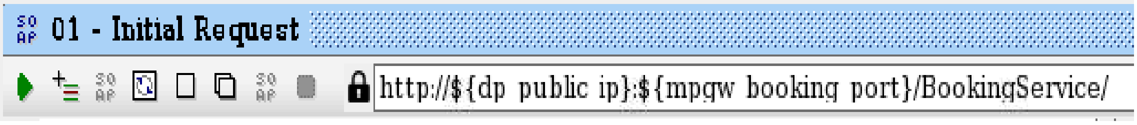
__ j. Repeat the previous steps g – i for the remaining values.

__ k. Click **OK**.

__ l. Click **File > Save Preferences**.

SoapUI is now configured for all exercises in this course. The messages that are sent to DataPower when using SoapUI reference these variables. No further SoapUI configuration is required, unless stated in the specific exercise.

When SoapUI recognizes the `dp_public_ip` reference in a request (“`${dp_public_ip}` ”), it substitutes the correct IP address into the URL.

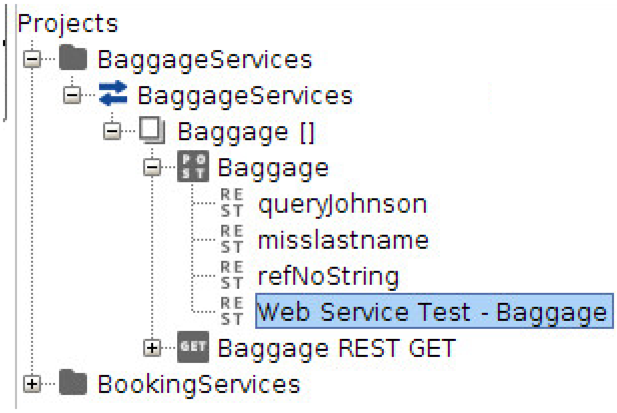


http://${dp public ip}:${mpgw booking port}/BookingService/

Part 2: Confirm that the Booking and Baggage web services are up

Test the Booking web service and the Baggage web service. The following steps ensure that the back-end web services are operational. In addition to testing the availability of the web service, it is also a useful troubleshooting technique to verify network connectivity to the back-end web service.

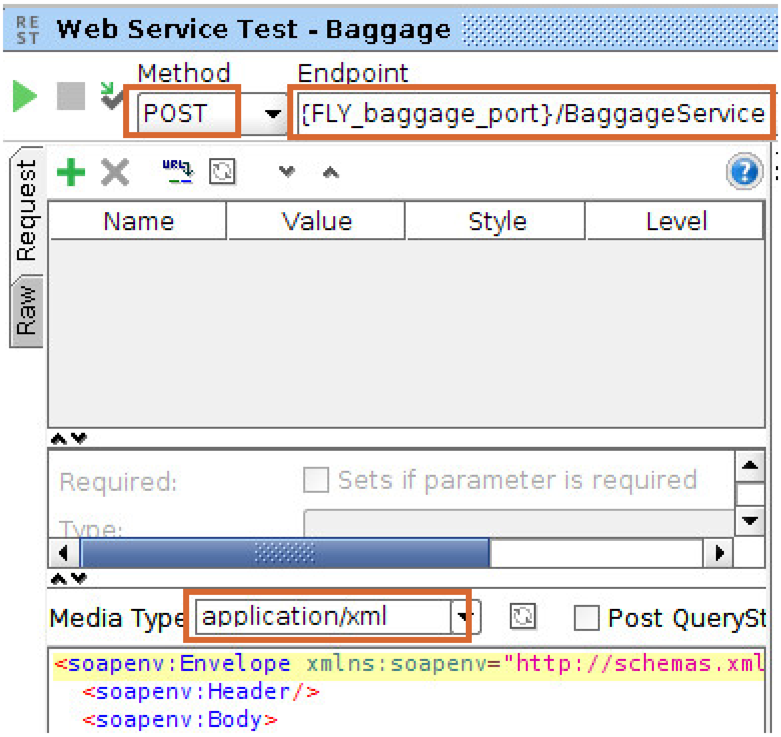
__ 1. In the project tree, expand the **BaggageServices** project until **Web Service Test - Baggage** is visible.



__ 2. Double-click **Web Service Test - Baggage** to open the request window. If a double-click does not work, right-click the request and click **Show Request Editor**.

__ 3. Ensure that the following information is preconfigured in the request message:

- Method: **POST**
- Endpoint: `http://${dp_internal_ip}:${FLY_baggage_port}/BaggageService`
- Media Type: **application/xml**



Method: POST

Endpoint: {FLY_baggage_port}/BaggageService

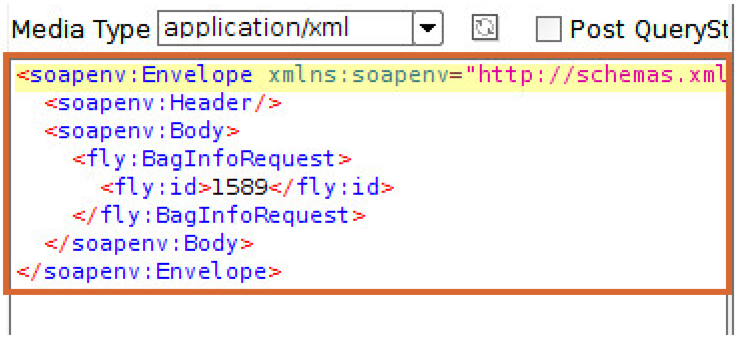
Required: □ Sets if parameter is required

Type:

Media Type: application/xml

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xml
  <soapenv:Header/>
  <soapenv:Body>
```

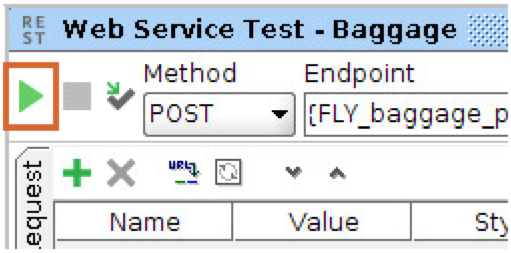
- Soap message:



Media Type: application/xml

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xml
  <soapenv:Header/>
  <soapenv:Body>
    <fly:BagInfoRequest>
      <fly:id>1589</fly:id>
    </fly:BagInfoRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

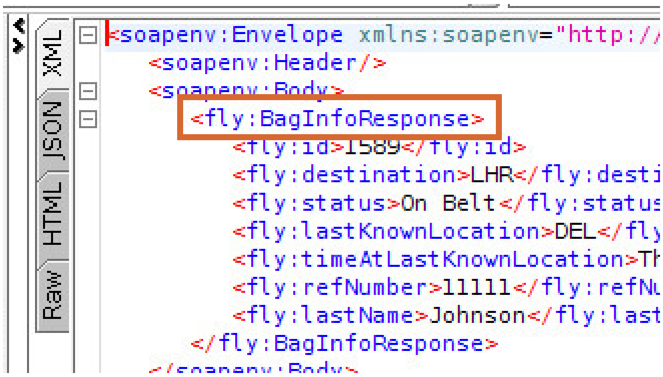
__ 4. Click the green **Submit** arrow to send the request message directly to the **BaggageService** web service for FLY airlines.



Method: POST

Endpoint: {FLY_baggage_p

__ 5. Confirm that a successful **BagInfoResponse** response is returned in the response tab.



```
<soapenv:Envelope xmlns:soapenv="http://
   <soapenv:Header/>
   <soapenv:Body>
      <fly:BagInfoResponse>
         <fly:id>1589</fly:id>
         <fly:destination>LHR</fly:desti
         <fly:status>On Belt</fly:status
         <fly:lastKnownLocation>DEL</fly
         <fly:timeAtLastKnownLocation>Th
         <fly:refNumber>11111</fly:refNu
         <fly:lastName>Johnson</fly:last
      </fly:BagInfoResponse>
   </soapenv:Body>
```

Important

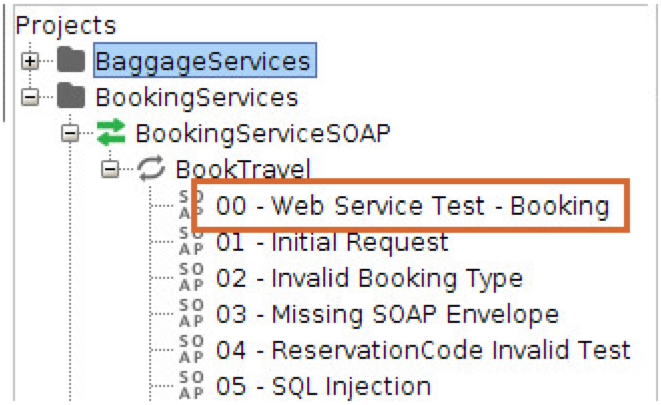
If you do not get the correct response, the failure can be due to several reasons:

- The variables that are entered in SoapUI General Preferences are not installed on the DataPower appliance.
- The DataPower appliance is unreachable from your student image due to some network connectivity issue.

Verify that you entered the correct values for the SoapUI variables. If the values are correct, escalate for assistance.

__ 6. Close the **Web Service Test - Baggage** window.

__ 7. In the project tree, expand the **BookingServices** project until **00 – Web Service Test - Booking** is visible.

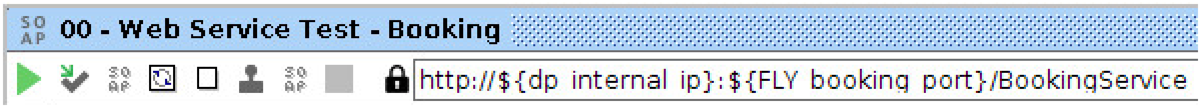


Projects
- BaggageServices
- BookingServices
  - BookingServiceSOAP
    - BookTravel
      - 00 - Web Service Test - Booking
      - 01 - Initial Request
      - 02 - Invalid Booking Type
      - 03 - Missing SOAP Envelope
      - 04 - ReservationCode Invalid Test
      - 05 - SQL Injection

__ 8. Double-click **00 – Web Service Test - Booking** to open the request window. If a double-click does not work, right-click the request and click **Show Request Editor**.

__ 9. Confirm that the URL address field contains:

`http://${dp_internal_ip}:${FLY_booking_port}>/BookingService`

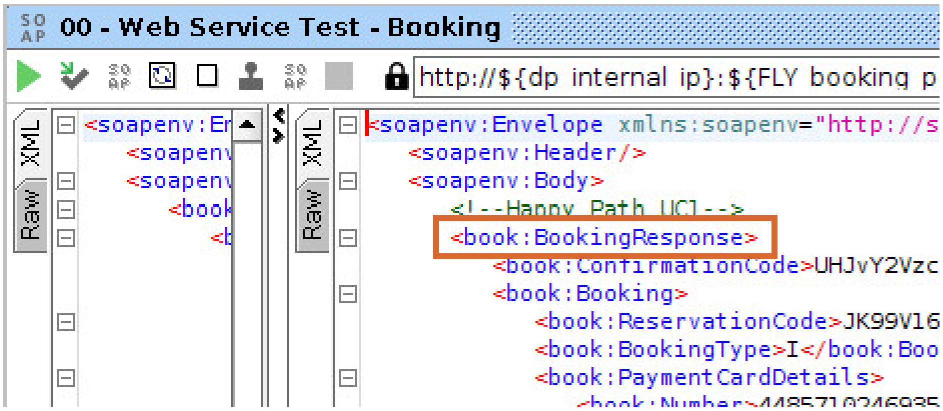


00 - Web Service Test - Booking

http://${dp_internal_ip}:${FLY_booking_port}/BookingService

__ 10. Click the green **Submit** arrow that is to the left of the URL address field to send the SOAP request test message directly to the FLY Airlines Booking web service.

__ 11. If everything worked properly, you should see the `<book:BookingResponse>` XML tree on the **Response** tab.



```
http://${dp internal ip}:${FLY booking p
```

```
<soapenv:En
    <soapen
    <soapen
        <book
            <b
```

```
<soapenv:Envelope xmlns:soapenv="http://s
   <soapenv:Header/>
   <soapenv:Body>
      <!--Happy Path UC1-->
      <book:BookingResponse>
         <book:ConfirmationCode>UHJvY2Vzc
         <book:Booking>
            <book:ReservationCode>JK99V16
            <book:BookingType>I</book:Boo
            <book:PaymentCardDetails>
               <book:Number>4485710246935
```

Important

If you do not get the correct response, the failure can be due to several causes:

- The variables that are entered in SoapUI General Preferences are wrong.
- The FLYService domain that contains the web services is not installed on the DataPower appliance.
- The DataPower appliance is unreachable from your student image due to a network connectivity issue.

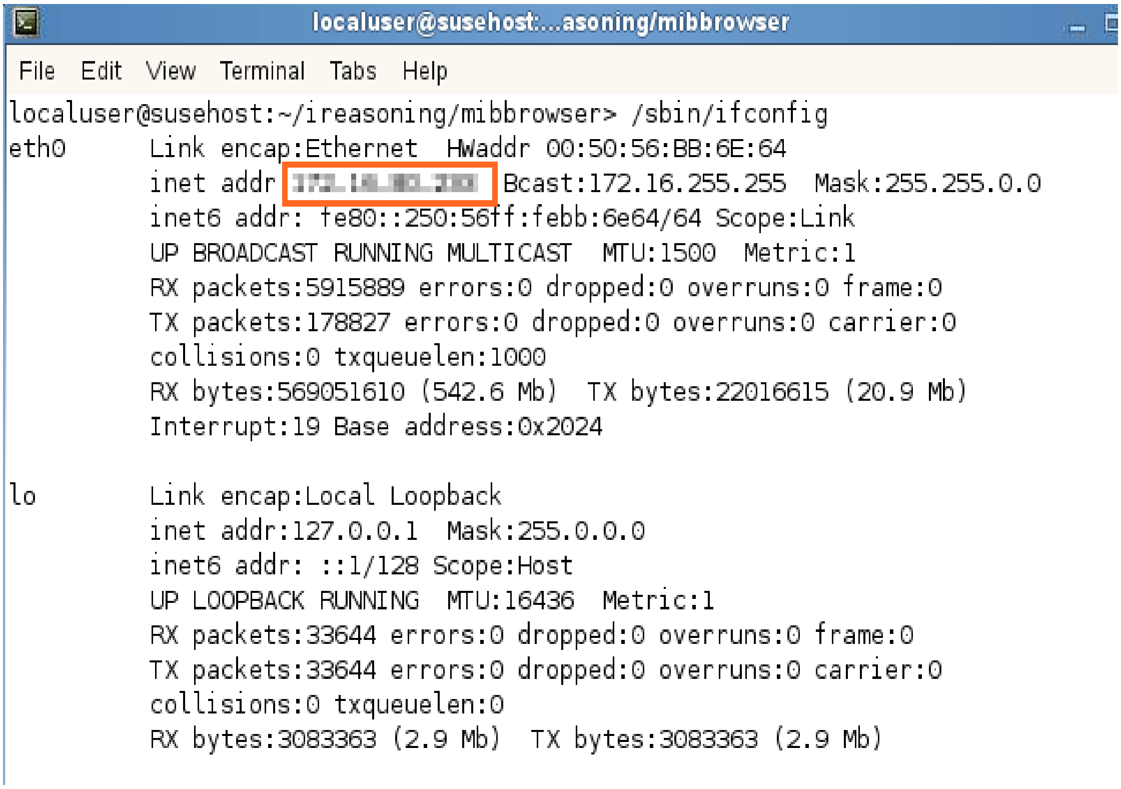
Verify that the values that you entered for the SoapUI variables are correct. If you still have problems, you must contact whatever support you have for the class.

__ 12. Close the **00 - Web Service Test - Booking** window.

Part 3: Identify the student image IP address

On Linux, you can discover the IP address by running the `ifconfig` command. Open a terminal window. The terminal window is available from the icon on the desktop. From within a terminal window, run the `ifconfig` command that includes the full path, as follows:

```
> /sbin/ifconfig
```


localuser@susehost:...asoning/mibbrowser
File Edit View Terminal Tabs Help

```
localuser@susehost:~/ireasoning/mibbrowser> /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:50:56:BB:6E:64
          inet addr [redacted] Bcast:172.16.255.255  Mask:255.255.0.0
          inet6 addr: fe80::250:56ff:febb:6e64/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5915889 errors:0 dropped:0 overruns:0 frame:0
          TX packets:178827 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:569051610 (542.6 Mb)  TX bytes:22016615 (20.9 Mb)
          Interrupt:19 Base address:0x2024

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:33644 errors:0 dropped:0 overruns:0 frame:0
          TX packets:33644 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3083363 (2.9 Mb)  TX bytes:3083363 (2.9 Mb)
```

When the IP address of the local student image is obtained, update the information in table B1 for the variable *<image_ip>*.

Close the terminal window.

Part 4: Port and variable table values

If you want to have a single reference for all variables that are used in this course, the following table is supplied. You might want to tear these two pages out of your book, or if you have a PDF file, you can print both pages as a quick reference point.

__ 1. Complete the following table with the values that are supplied by the instructor.

Table B-1. Developers course variable and port assignments table

Object	**Value (default)**
Lab files location	
<lab_files> Location of student lab files for this course	/usr/labfiles/dp
Student information	
<nn>	
<student*nn*>	
<student*nn*_domain>	
<student*nn*_password>	student*<nn>*
<student*nn*_updated_password>	
<image_ip> IP address of the student image	
Logins that are not DataPower	
<linux_user>	localuser
<linux_user_password>	passw0rd
<linux_root_user>	
<linux_root_password>	passw0rd
DataPower information	
<dp_public_ip> IP address of the public services on the appliance	
<dp_internal_ip> IP address of the DataPower appliance development and administrative functions	
<dp_WebGUI_port> Port number for the WebGUI	9090
<dp_FLY_baggage_port>	2068
<dp_FLY_booking_port>	9080
Server information	
<SoapUI_keystores>	/usr/labfiles/dp/WSSecurity/Client.jks

<SoapUI_keystores_password>	myjkspw
<ldap_password>	passw0rd
<ldap_server_root_dir>	/var/lib/ldap/ibm-com/
<ldap_user_name>	cn=admin,dc=ibm,dc=com
<http_server_port>	80
<http_server_root_dir>	/var/www/html/
<logger_app_port>	1112
Student-built DataPower services	
<mpgw_booking_port>	12*nn*1
<mpgw_booking_ssl_port>	12*nn*2
<mpgw_ssl_booking_port>	12*nn*3
<mpgw_mq_port>	12*nn*4
<wsp_booking_port>	12*nn*5
<mpgw_helloworld_port>	12*nn*7
<mpgw_patterns_port>	12*nn*8
<mpgw_baggage_port>	12*nn*9
<oidc_provider_port>	7*nn5*
<oidc_client_port>	7*nn4*

End of appendix

IBM Training