Lab 2: Using Java and JavaScript Adapters with MobileFirst Server

MobileFirst Adapters are Maven projects that contain server-side code implemented in either Java or JavaScript. Adapters are used to perform any necessary server-side logic, and to transfer and retrieve information from back-end systems to client applications and cloud services.

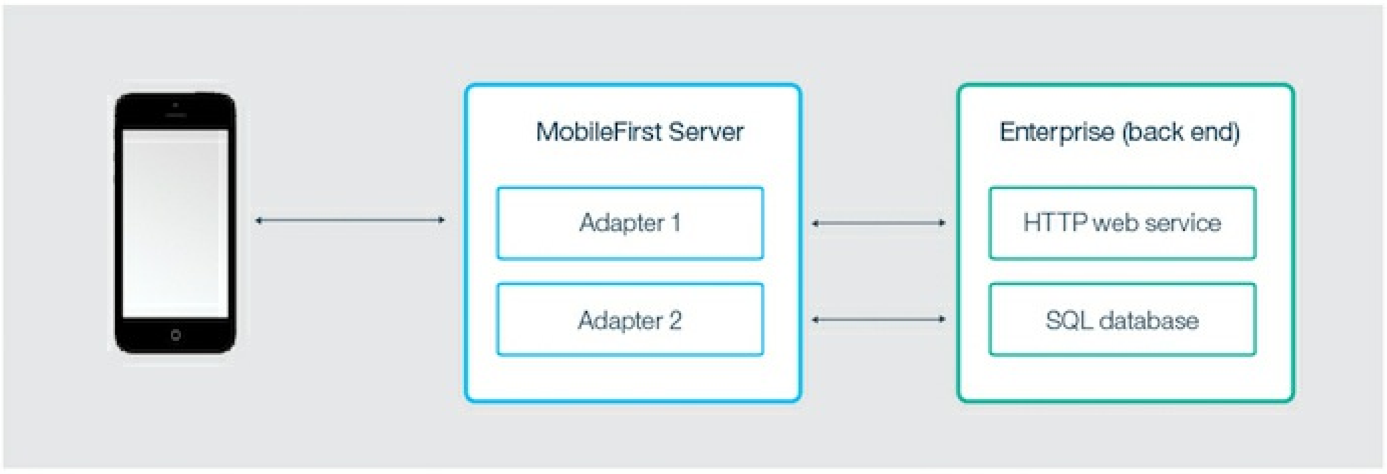
Each adapter has its own isolated sandbox, which runs without knowing about or interrupting other adapter sandboxes. That said, adapters can still communicate with one another by calling server-side API which makes "adapter mashup" possible, or the ability to aggregate calls to multiple backends and return a result.

It is also possible to include third-party Java libraries that are required by the adapter code by defining them as Maven dependencies in the adapter's **pom.xml** file.

Adapters also support DevOps needs:

- You can "hot deploy" adapters, meaning deploy, undeploy, and redeploy them at run time. This capability lends great flexibility to the MobileFirst server-side development process.
- An adapter can have user-defined properties, which can be configured by administration personnel without redeploying the adapter. This feature lets you customize adapter behavior for different environments, for example development, testing, and production.

Continue reading for an overview of Java and JavaScript adapters. Then follow the tutorials in the sidebar navigation to learn how to create, build, update, develop, test and deploy Java and JavaScript adapters, learn how to use IDE's for adapter development, collect server-side logs, as well as more advanced topics such as adapters mashup and Cloudant integration.



Benefits of using adapters

Universality

- Adapters support multiple integration technologies and back-end information systems.

Read-only and transactional capabilities

- Adapters support read-only and transactional access modes to back-end systems.

Fast development

- Adapters use simple XML syntax and are easily configured with JavaScript API or Java API.

Security

- Adapters use flexible authentication facilities to create connections with back-end systems.

- Adapters offer control over the identity of the connected user.

Transparency

- Data that is retrieved from back-end applications is exposed in a uniform manner, regardless of the adapter type.

Benefits specific to Java adapters

- Ability to fully control the URL structure, the content types, the request and response headers, content and encoding
- Ability to test the adapter by using a 3rd-party tool such as Postman
- Easy and fast deployment to a running MobileFirst Server instance with no compromise on performance and no downtime
- Security integration with the MobileFirst security model with no additional customization, by using simple annotations in the source code

JavaScript adapters

JavaScript adapters provide templates for communicating with HTTP and SQL back-ends. These templates contain a set of services called procedures. Mobile apps can call these procedures by issuing AJAX requests. The procedure retrieves information from the back-end application and return data to the application. Using a REST interface, you can benefit from the OAuth 2.0-based security framework provided by MobileFirst.

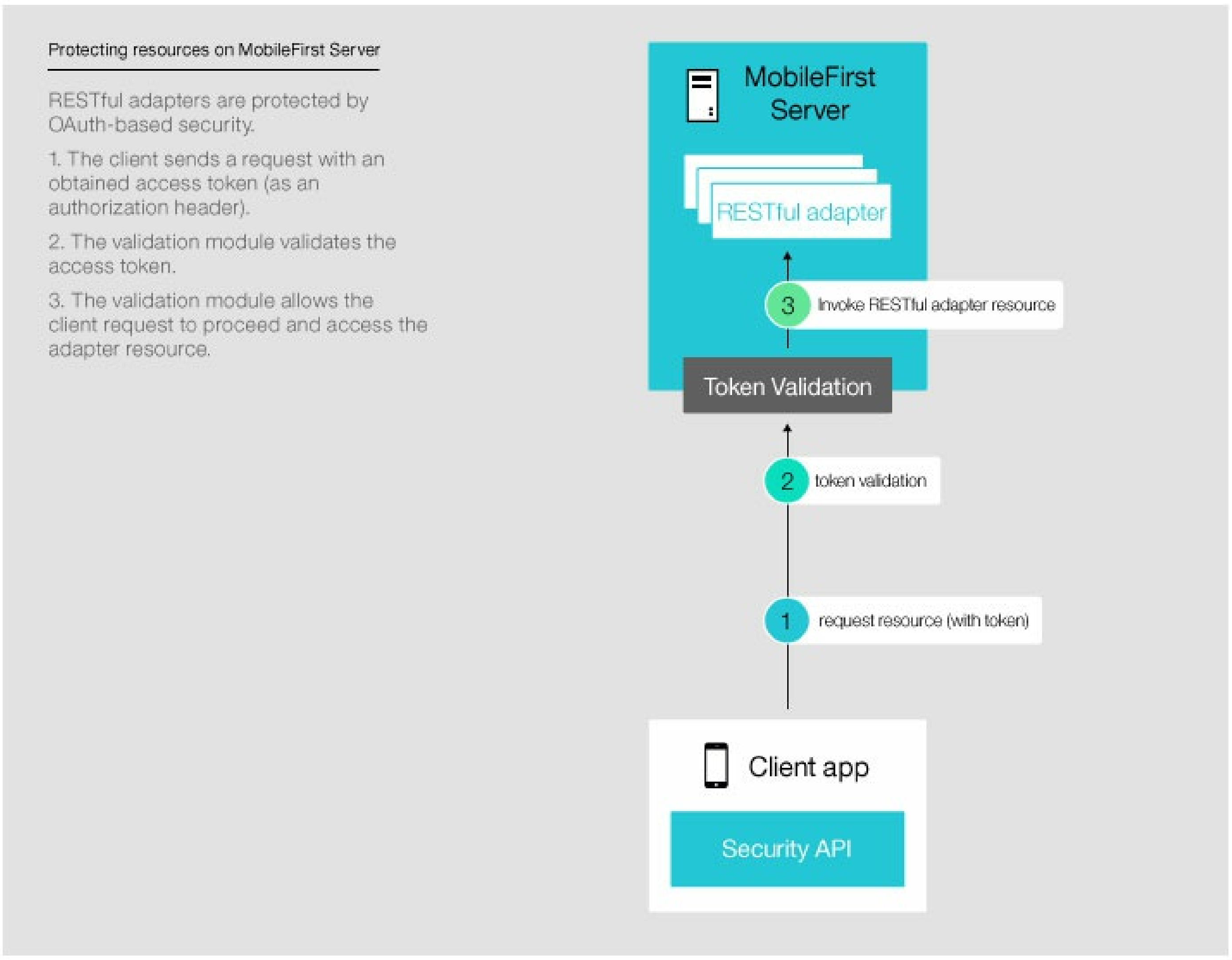
- If the format of the data is JSON then the MobileFirst Server keeps the data intact. Otherwise the MobileFirst Server automatically converts it to JSON.

- Alternatively you can provide XSL transformation to convert the data to JSON. In this case, the returned content type from the back-end must be XML. Then, you can use an XSLT transformation to filter the data based on requirements.
- With an HTTP adapter, you can send GET or POST HTTP requests and retrieve data from the response headers and body. HTTP adapters work with RESTful and SOAP-based services, and can read structured HTTP sources such as RSS feeds.
- With an SQL adapter, you can communicate with any SQL data source. You can use plain SQL queries or stored procedures.

Java adapters

Java adapters are based on the JAX-RS 2.0 specification and are thus written in Java, and expose a full REST API to the client. In Java adapters, it is up to the developer to define the returned content and its format, as well as the URL structure of each resource. The only exception is if the client sending the request supports GZip, then the returned content encoding of the Java adapter is compressed by GZip. All operations on the returned content are implemented and owned by the developer.

The diagram below depicts how a mobile device can access any Java adapter from its REST endpoint. The REST interface is protected by the MobileFirst OAuth security filter, meaning that the client needs to obtain an access token to access the adapter resources. Each of the resources of the adapter has its own URL, so it is possible to protect MobileFirst endpoints using any firewall. The REST interface invokes the Java code (JAX-RS service) to handle incoming requests. The Java code can perform operations on the server by using the Java MobileFirst Server API. In addition, the Java code can connect to the enterprise system to fetch data, update data, or perform any other operation that the enterprise system exposes.


Protecting resources on MobileFirst Server
RESTful adapters are protected by OAuth-based security.
1. The client sends a request with an obtained access token (as an authorization header).
2. The validation module validates the access token.
3. The validation module allows the client request to proceed and access the adapter resource.
MobileFirst Server
RESTful adapter
3 Invoke RESTful adapter resource
Token Validation
2 token validation
1 request resource (with token)
Client app
Security API

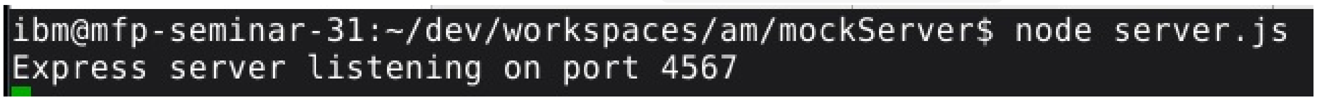
What you will be doing in this lab?

- Create a JavaScript Adapter
- Build and Deploy the JavaScript Adapter
- Test the JavaScript Adapter using Swagger
- Create a Java Adapter
- Build and Deploy the Java Adapter
- Test the Java Adapter using Swagger
- Modify the sample app to use the JavaScript and Java Adapter
- Test the sample app

Backend Services

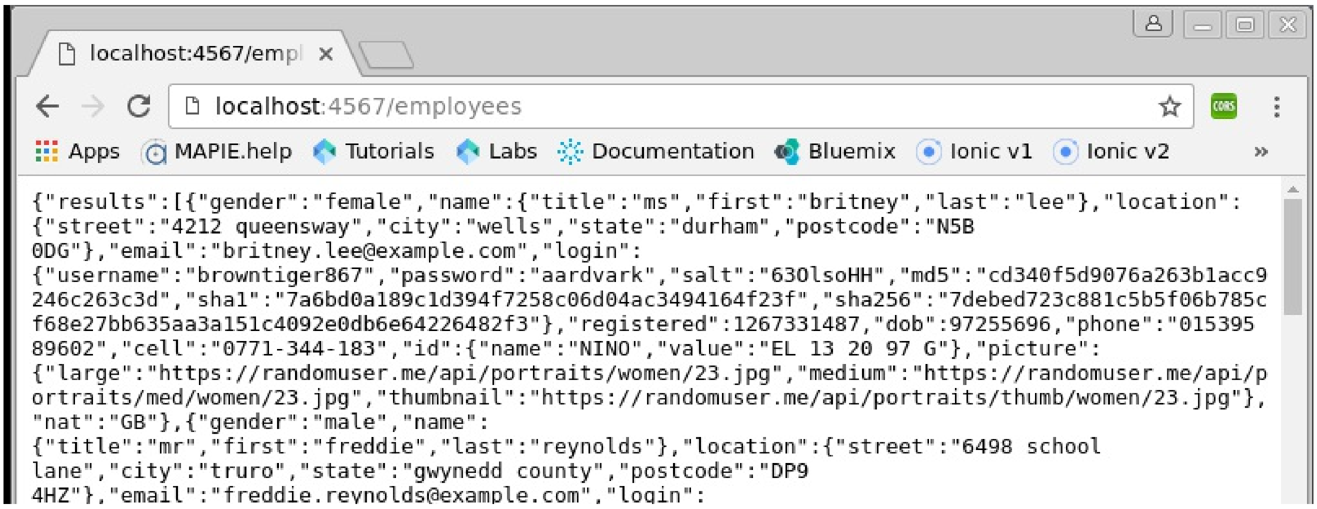
In this lab, we are going to create adapters that connect to our mock services. There are two mock services that we will be connecting. Let's test the services to make sure they are all working.

1. If your mock server is not already started, start it now by changing to the following directory using `cd ~/dev/workspaces/am/mockServer` then entering the following command in the terminal window: `node server.js`



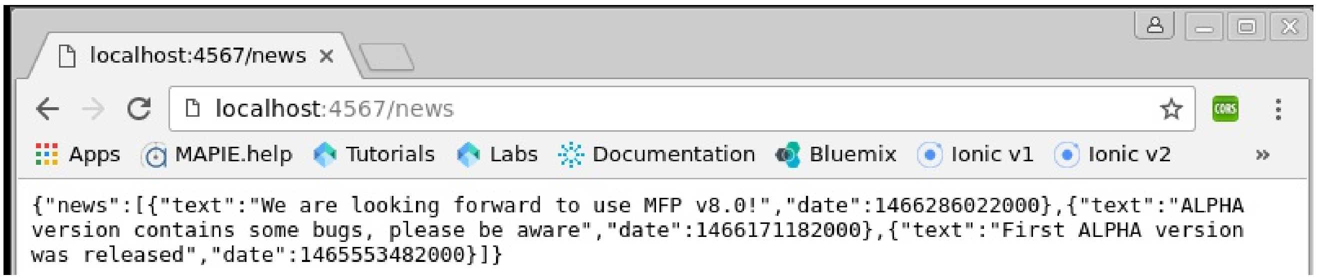
```
ibm@mfp-seminar-31:~/dev/workspaces/am/mockServer$ node server.js
Express server listening on port 4567
```

2. In the web browser, go to this URL which will bring back a list of employees: http://localhost:4567/employees.



```
{"results":[{"gender":"female","name":{"title":"ms","first":"britney","last":"lee"},"location":
{"street":"4212 queensway","city":"wells","state":"durham","postcode":"N5B
0DG"},"email":"britney.lee@example.com","login":
{"username":"browntiger867","password":"aardvark","salt":"63OlsoHH","md5":"cd340f5d9076a263b1acc9
246c263c3d","sha1":"7a6bd0a189c1d394f7258c06d04ac3494164f23f","sha256":"7debed723c881c5b5f06b785c
f68e27bb635aa3a151c4092e0db6e64226482f3"},"registered":1267331487,"dob":97255696,"phone":"015395
89602","cell":"0771-344-183","id":{"name":"NINO","value":"EL 13 20 97 G"},"picture":
{"large":"https://randomuser.me/api/portraits/women/23.jpg","medium":"https://randomuser.me/api/p
ortraits/med/women/23.jpg","thumbnail":"https://randomuser.me/api/portraits/thumb/women/23.jpg"},
"nat":"GB"},{"gender":"male","name":
{"title":"mr","first":"freddie","last":"reynolds"},"location":{"street":"6498 school
lane","city":"truro","state":"gwynedd county","postcode":"DP9
4HZ"},"email":"freddie.reynolds@example.com","login":
```

3. Now access this URL which will bring back a list of news items: http://localhost:4567/news.

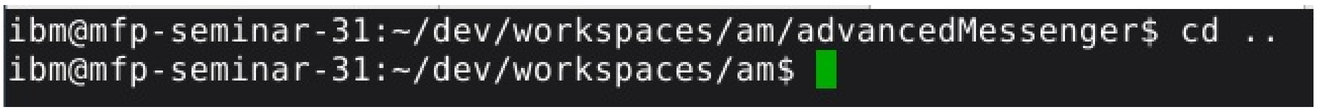
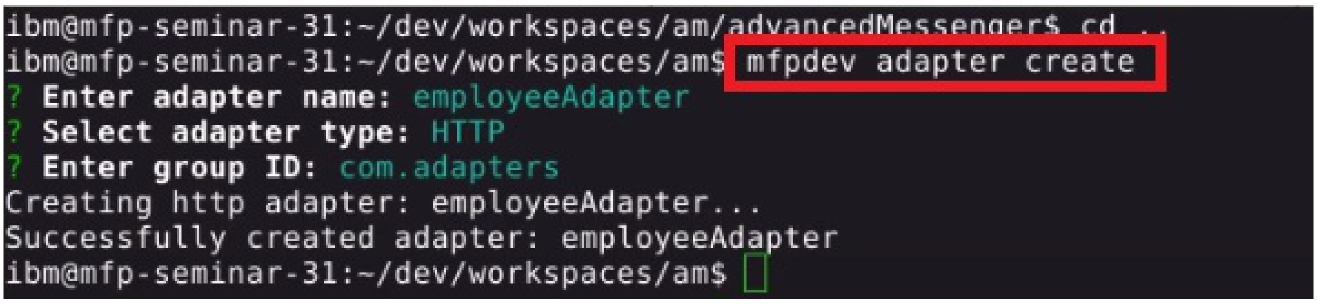


```
{"news":[{"text":"We are looking forward to use MFP v8.0!","date":1466286022000},{"text":"ALPHA
version contains some bugs, please be aware","date":1466171182000},{"text":"First ALPHA version
was released","date":1465553482000}]}
```

4. Minimize this terminal window for the remainder of this lab, so that

our Node.js mock services will remain running.

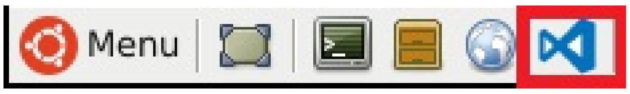
Create a JavaScript Adapter

1. Open a new terminal window, and ensure that you are in the **/dev/workspaces/am/** directory.
   
   ```
   ibm@mfp-seminar-31:~/dev/workspaces/am/advancedMessenger$ cd ..
   ibm@mfp-seminar-31:~/dev/workspaces/am$
   ```
2. Enter the following command to create a JavaScript adapter: `mfpdev adapter create`
3. Enter **employeeAdapter** as the **adapter name**.
4. Select **HTTP** as the **adapter type**.
5. Enter **com.adapters** as the **group ID**.
   
   ```
   ibm@mfp-seminar-31:~/dev/workspaces/am/advancedMessenger$ cd ..
   ibm@mfp-seminar-31:~/dev/workspaces/am$ mfpdev adapter create
   ? Enter adapter name: employeeAdapter
   ? Select adapter type: HTTP
   ? Enter group ID: com.adapters
   Creating http adapter: employeeAdapter...
   Successfully created adapter: employeeAdapter
   ibm@mfp-seminar-31:~/dev/workspaces/am$
   ```
   You will see that the adapter has been successfully created.

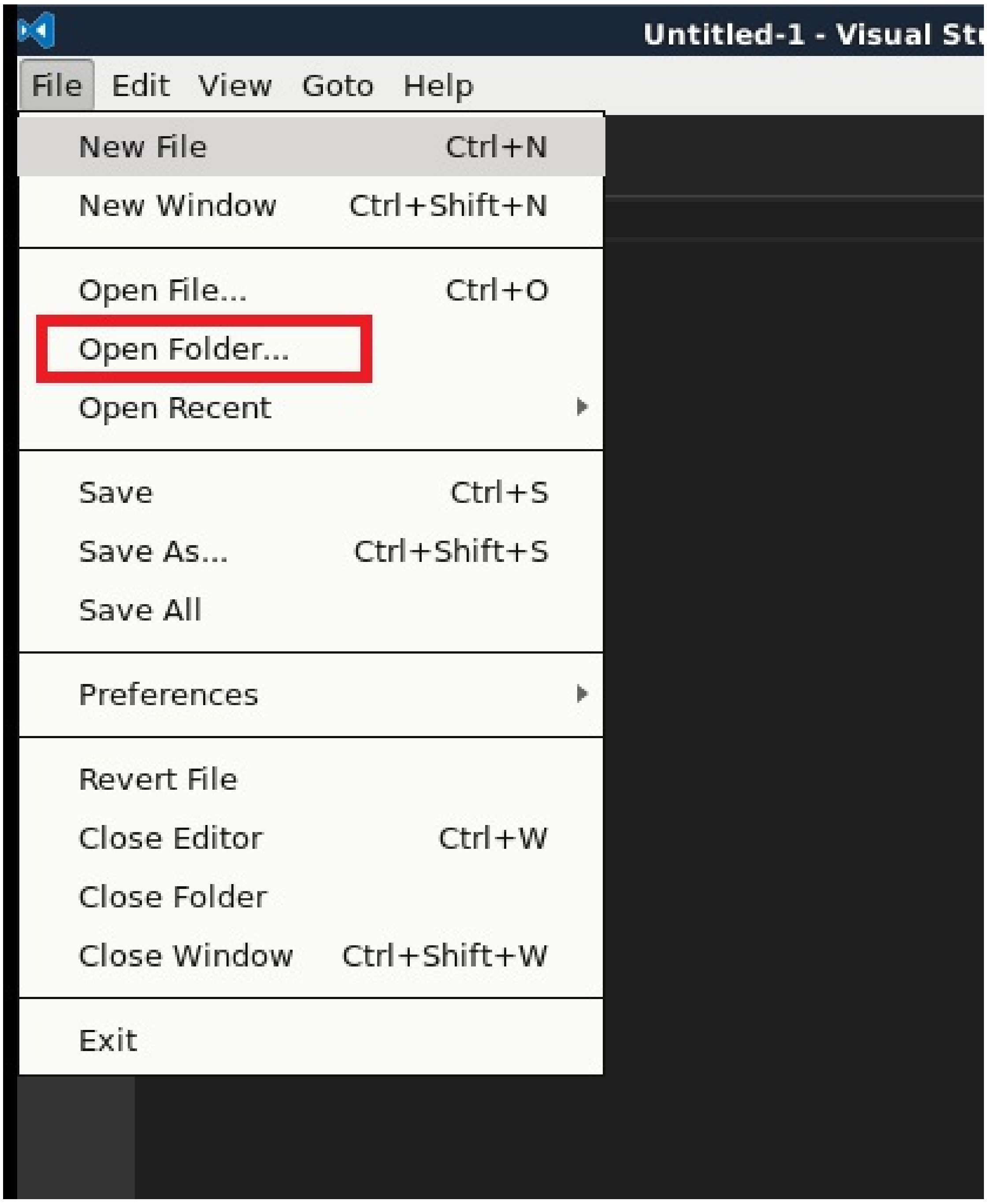
Modify the adapter.xml

In a JavaScript adapter, the host name and port number of the backend service is configured in an *adapter.xml* file. The actual path to the backend service is inside the JavaScript file. The *adapter.xml* file also specifies the name of the method we want to create for the mobile app to use.

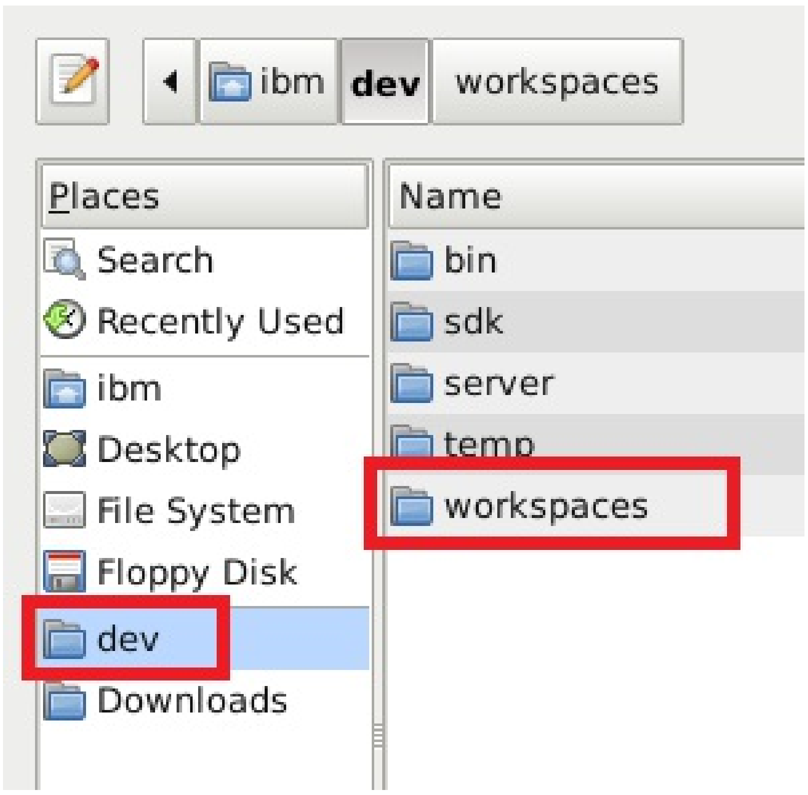
1. Open the Visual Studio editor by clicking on the blue icon on the bottom menu. You can also choose to use your favorite editor of choice such as Sublime Text, Atom, or Brackets if you wish.

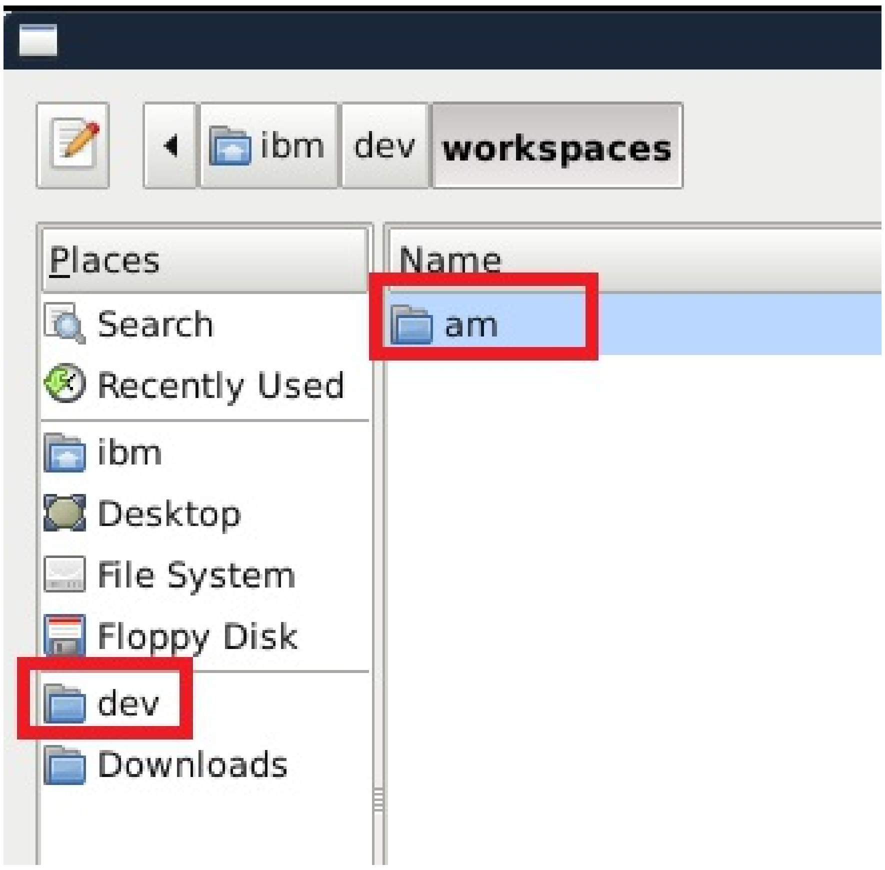


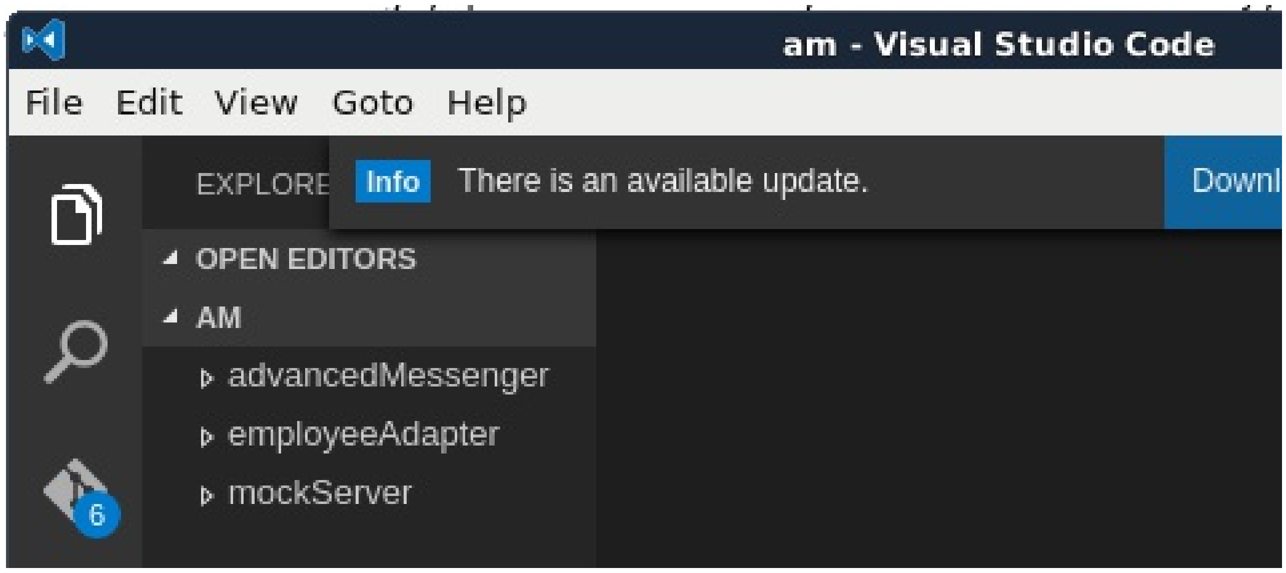
2. In the editor, select **File** > **Open Folder**.



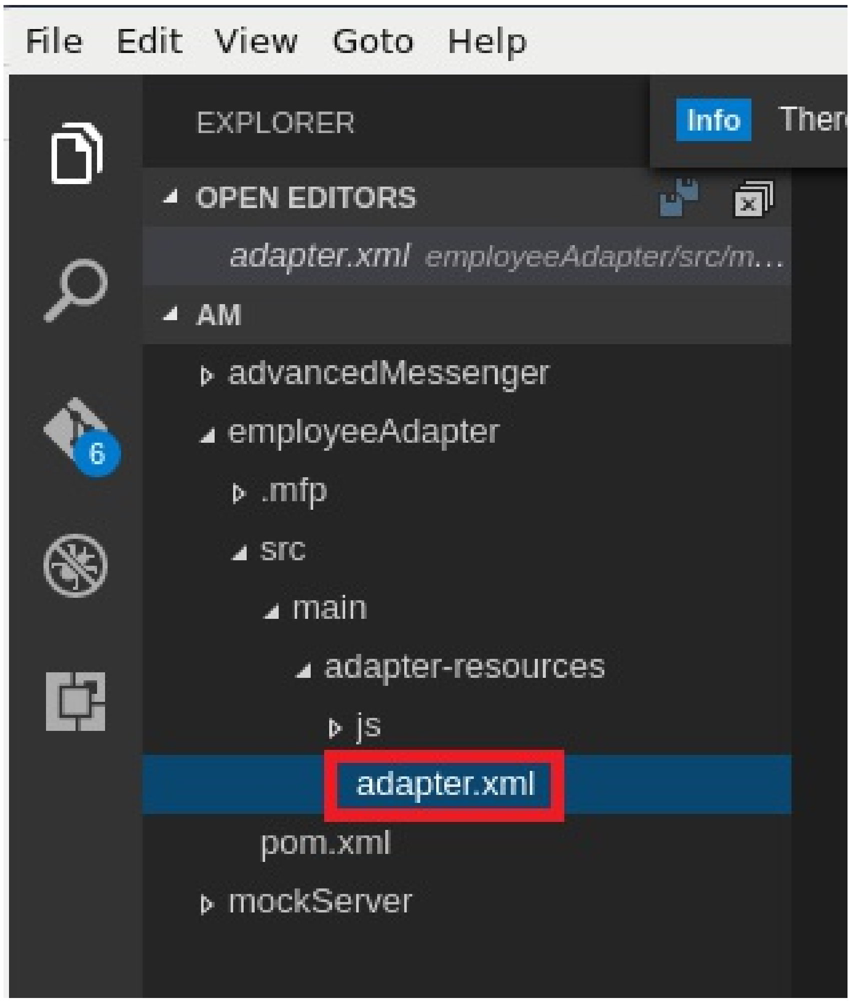
3. Select **dev** > **workspaces** > **am** folder. Click **OK.**


ibm
dev
workspaces
Places
Search
Recently Used
ibm
Desktop
File System
Floppy Disk
dev
Downloads
Name
bin
sdk
server
temp
workspaces

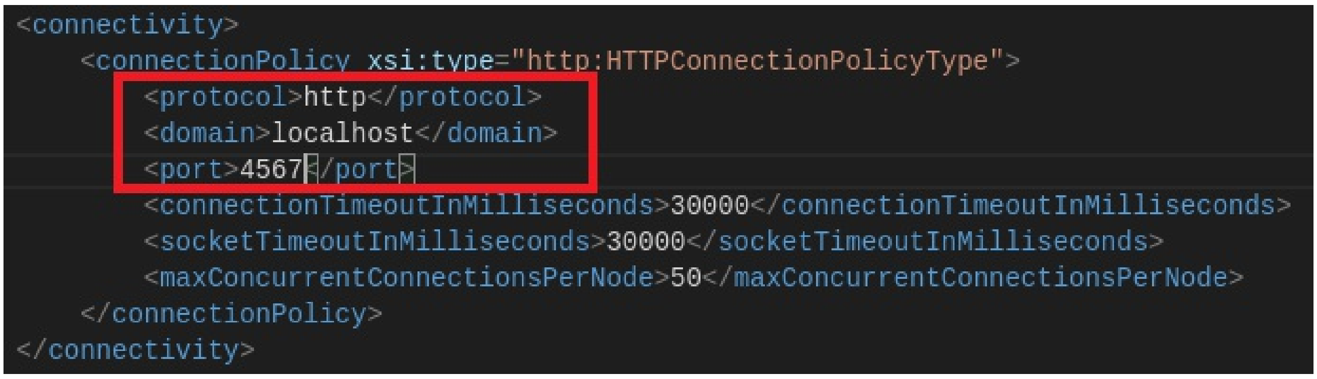

ibm
dev
workspaces
Places
Search
Recently Used
ibm
Desktop
File System
Floppy Disk
dev
Downloads
Name
am



4. Expand **employeeAdapter** > **src** > **main** > **adapter-resources** > **js** > click on **adapter.xml** to open it in an editor.

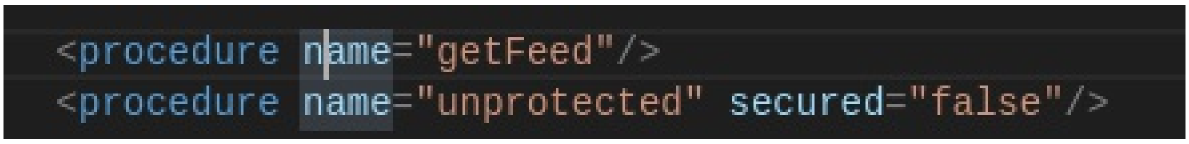


1. In the editor pane, change the protocol from *https* to **http**.
2. Change the domain to **localhost**.
3. Change the port to **4567**.

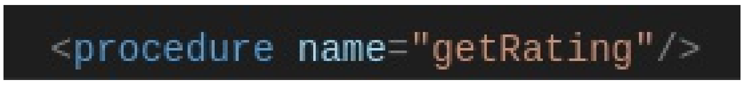

```
<connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
        <protocol>http</protocol>
        <domain>localhost</domain>
        <port>4567</port>
        <connectionTimeoutInMilliseconds>30000</connectionTimeoutInMilliseconds>
        <socketTimeoutInMilliseconds>30000</socketTimeoutInMilliseconds>
        <maxConcurrentConnectionsPerNode>50</maxConcurrentConnectionsPerNode>
    </connectionPolicy>
</connectivity>
```

4. Since our adapter will only create one procedure for the mobile app to call. Currently, the adapter.xml has default two procedures. We will delete the last procedure and replace the *getFeed* procedure to *getRating*. **Save** the file.

Before


```
<procedure name="getFeed"/>
<procedure name="unprotected" secured="false"/>
```

After


```
<procedure name="getRating"/>
```

Final Adapter.xml file

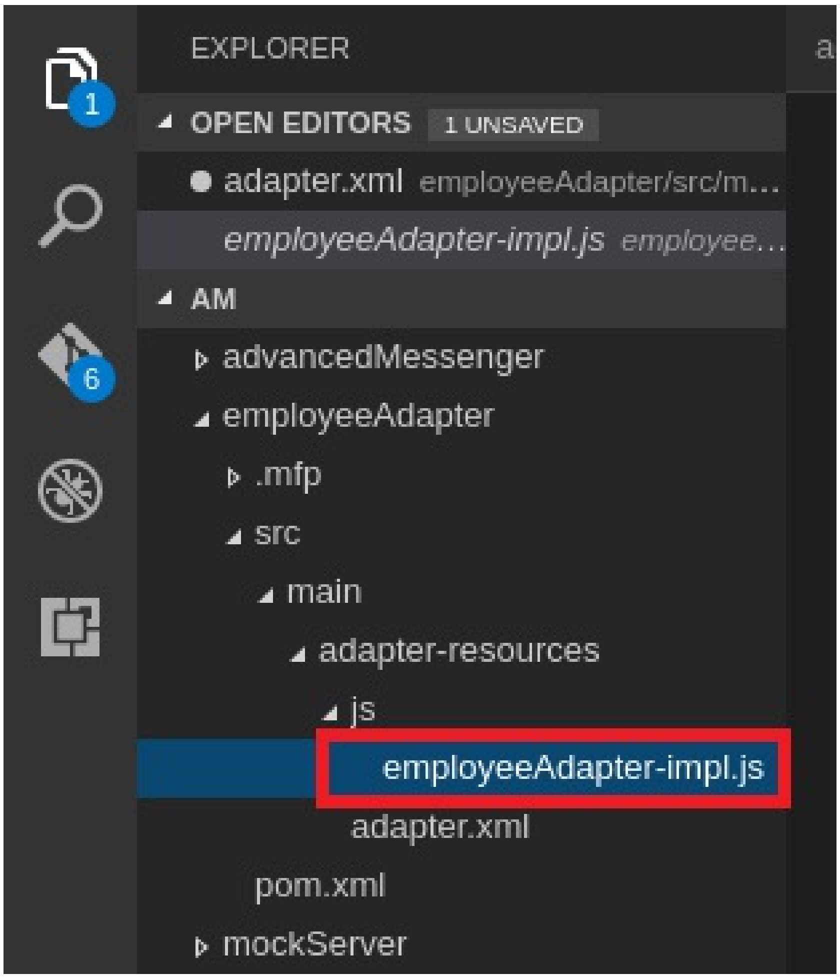
The final adapter.xml file looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Licensed Materials - Property of IBM
    5725-I43 (C) Copyright IBM Corp. 2011, 2016. All Rights
    US Government Users Restricted Rights - Use, duplication
    disclosure restricted by GSA ADP Schedule Contract with
-->
<mfp:adapter name="employeeAdapter"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
             xmlns:mfp="http://www.ibm.com/mfp/integration"
             xmlns:http="http://www.ibm.com/mfp/integration/
    <displayName>employeeAdapter</displayName>
    <description>employeeAdapter</description>
    <connectivity>
        <connectionPolicy xsi:type="http:HTTPConnectionPolic
            <protocol>http</protocol>
            <domain>localhost</domain>
            <port>4567</port>
            <connectionTimeoutInMilliseconds>30000</connecti
            <socketTimeoutInMilliseconds>30000</socketTimeou
            <maxConcurrentConnectionsPerNode>50</maxConcurre
        </connectionPolicy>
    </connectivity>

    <procedure name="getRating"/>

</mfp:adapter>
```

Implement the JavaScript Adapter

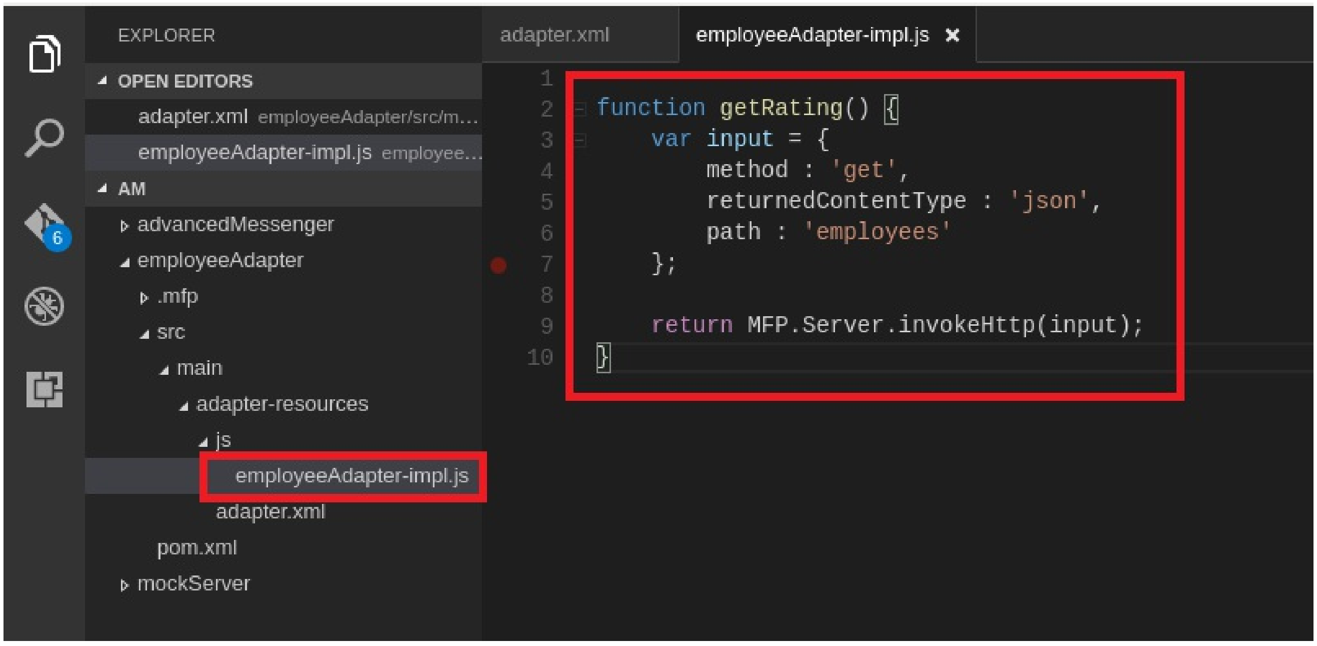
When we created a JavaScript Adapter, the tool automatically generated some sample code and the sample XML configuration. We will replace the sample code with our code in the adapter. The implementation is called **employeeAdapter-impl.js**.

1. In the editor, navigate to **am/employeeAdapter/src/main/adapter0resources/js/employeeAdapter-impl.js**. Open **employeeAdapter-impl.js** in the editor.



2. Delete all code from the file and replace with the following and **save** the file.

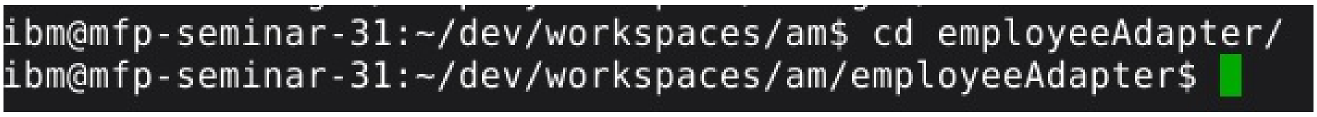
```
function getRating() {
      var input = {
      method : 'get',
      returnedContentType : 'json',
      path : 'employees'
      };

      return MFP.Server.invokeHttp(input);
 }
```



```
function getRating() {
    var input = {
        method : 'get',
        returnedContentType : 'json',
        path : 'employees'
    };

    return MFP.Server.invokeHttp(input);
}
```

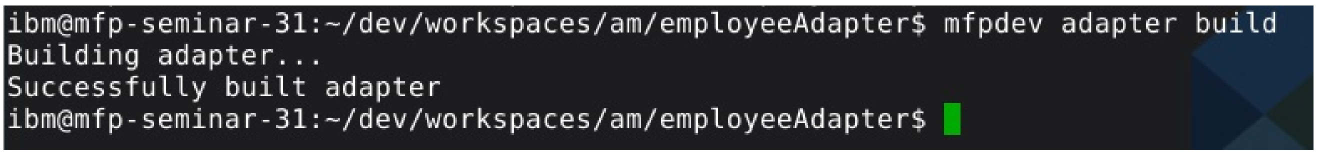
Build and Deploy the JavaScript Adapter

1. In a terminal window, make sure you are in the following directory: **dev/workspaces/am/employeeAdapters**



```
ibm@mfp-seminar-31:~/dev/workspaces/am$ cd employeeAdapter/
ibm@mfp-seminar-31:~/dev/workspaces/am/employeeAdapter$
```

2. Enter the following command to build the adapter: `mfpdev adapter build`



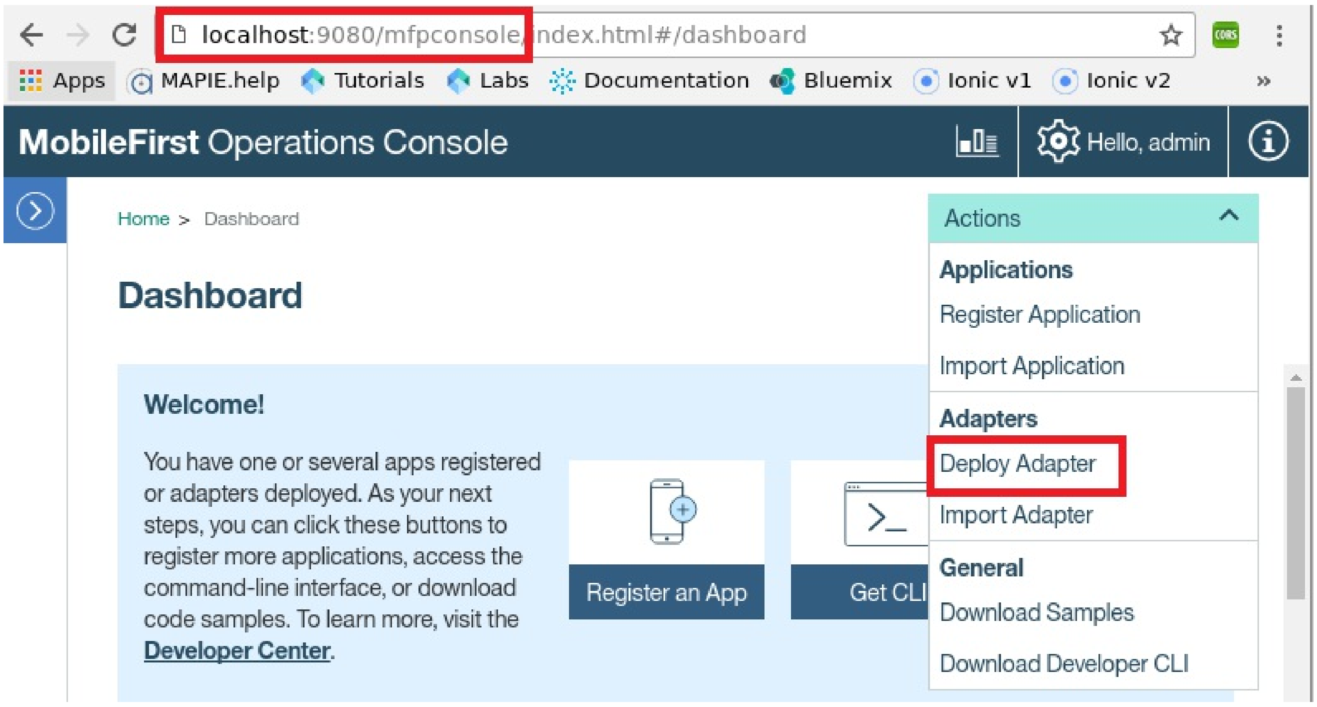
```
ibm@mfp-seminar-31:~/dev/workspaces/am/employeeAdapter$ mfpdev adapter build
Building adapter...
Successfully built adapter
ibm@mfp-seminar-31:~/dev/workspaces/am/employeeAdapter$
```

You will see the adapter was built successfully. This means there is now a compiled version of the adapter ready to deploy.

3. There are multiple ways to deploy the adapter. We could deploy an adapter using `mfpdev adapter deploy` from the MobileFirst Command Line (CLI) or we could deploy it using the MobileFirst Operations console. In this exercise, we will use the console to

deploy the adapter. In a web browser, go to the url http://localhost:9080/mfpconsole and login as **admin/admin**.

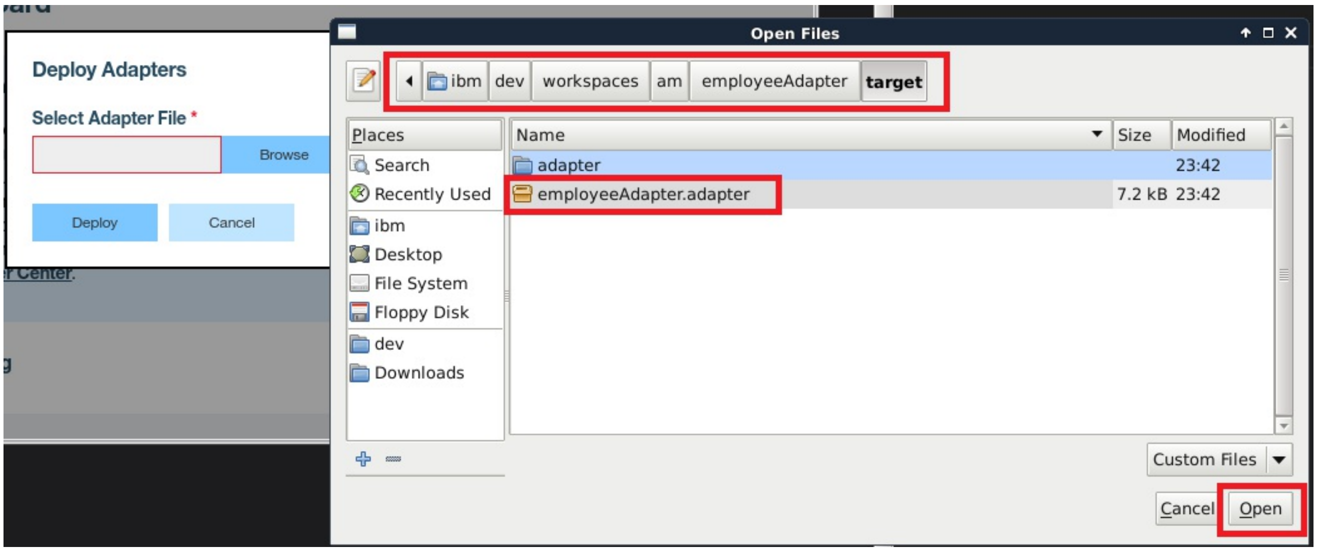
4. In the Actions menu on the right corner, select **Deploy Adapter**.



You have one or several apps registered or adapters deployed. As your next steps, you can click these buttons to register more applications, access the command-line interface, or download code samples. To learn more, visit the Developer Center.

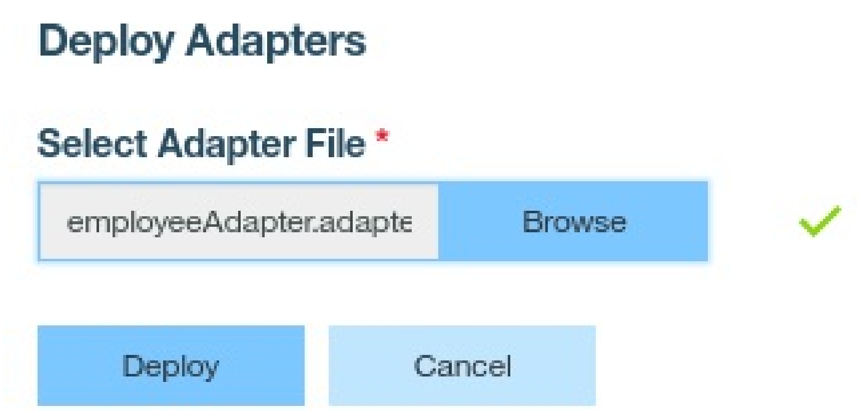
5. Click **Browse** to select the adapter to deploy.

6. Select **am** folder > **employeeAdapter** folder > **target** folder > select **employeeAdapter.adapter** file and click **Open**.

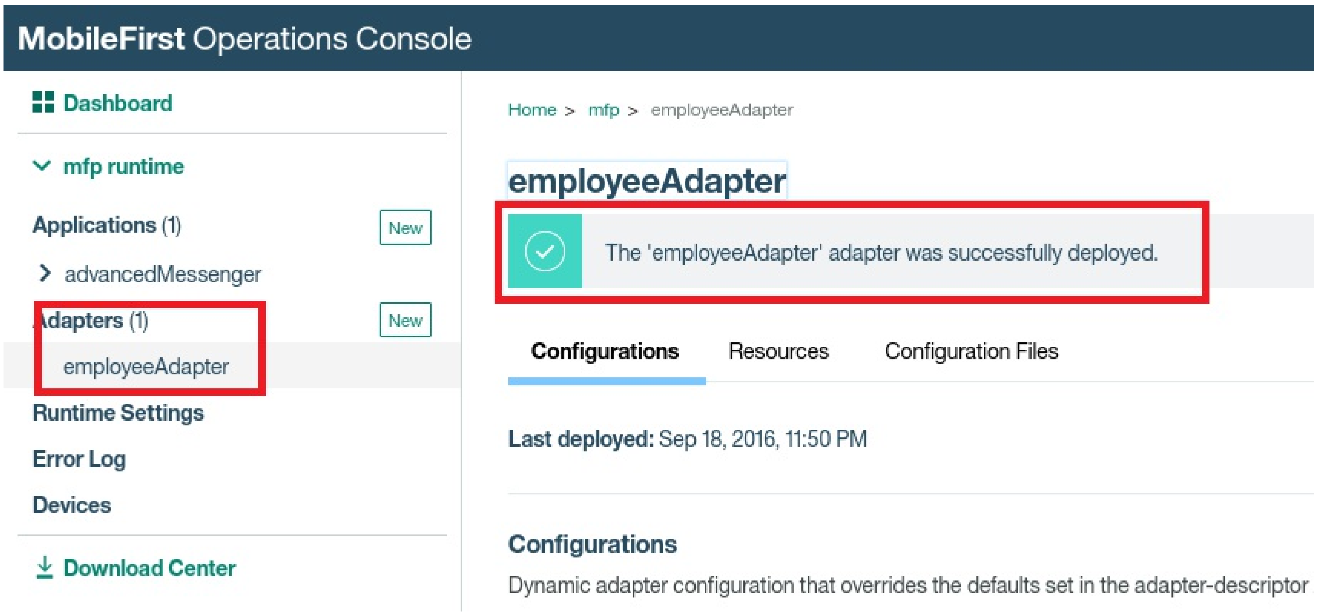


Name	Size	Modified
adapter		23:42
employeeAdapter.adapter	7.2 kB	23:42

7. Click **Deploy**.



8. You will see that the adapter has deployed successfully as shown.



The 'employeeAdapter' adapter was successfully deployed.

Last deployed: Sep 18, 2016, 11:50 PM

Configurations

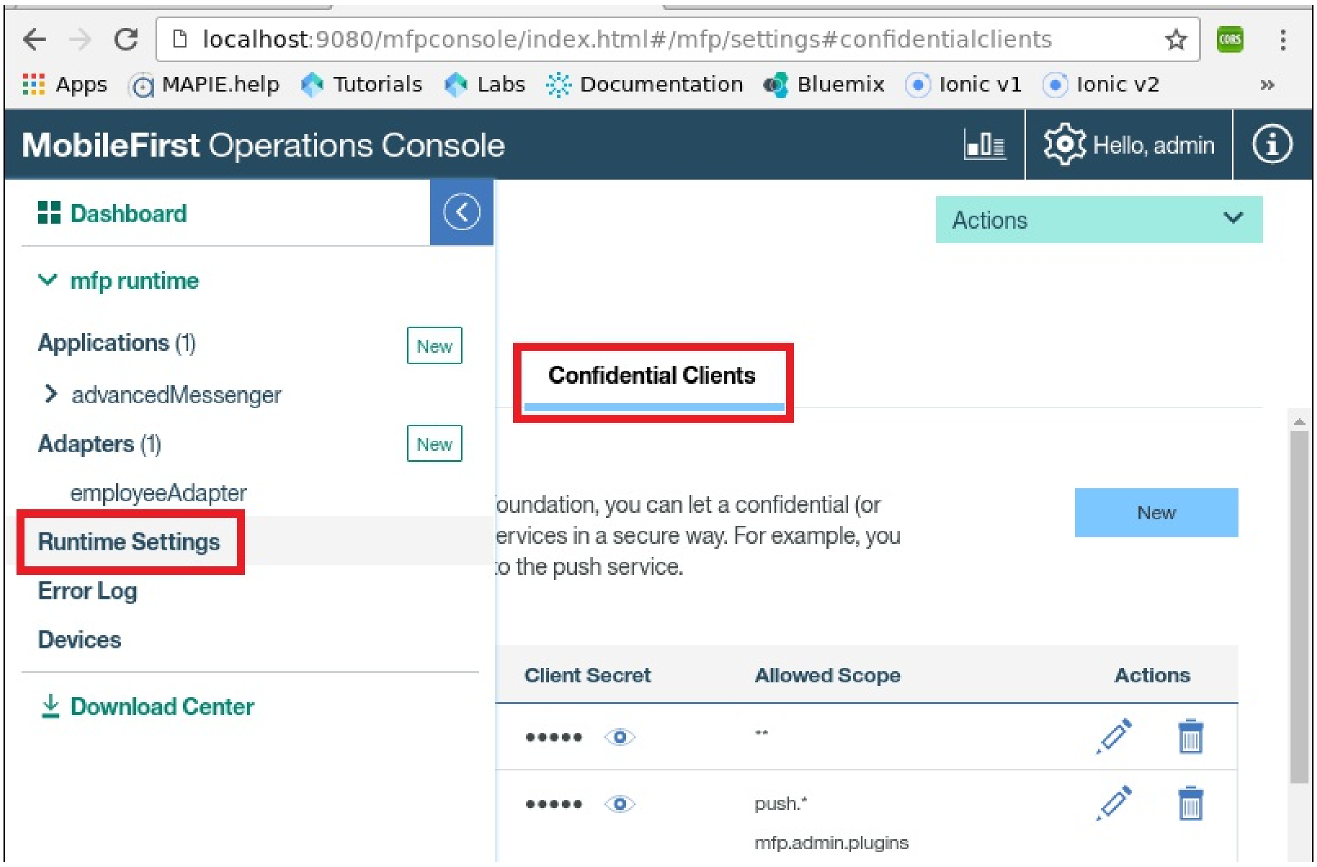
Dynamic adapter configuration that overrides the defaults set in the adapter-descriptor

Testing the Adapter using Swagger

IBM MobileFirst Platform comes with a Swagger web GUI for testing adapters. All adapters created on the platform are by default already secured with basic MobileFirst security. It means the adapters by default are not publicly callable API's - they are only callable from a MobileFirst app. This is extremely important since exposing your adapters completely without any security is an unacceptable level of risk to your backend resources. For example, random crawlers could call open adapters at high load and potentially bring down the server if the adapters were not secured by default.

The adapter can only be called from a valid MobileFirst app. In order to test the adapter from a web browser using the Swagger GUI, we will need to provide an authorization token for the Swagger GUI. This access token is called the *confidential client* which can be configured in the MobileFirst Operations Console, as we are about to do.

1. In the MobileFirst Operations console, select the **Runtime Settings** on the left menu. On the right pane, select the **Confidential Clients** tab.

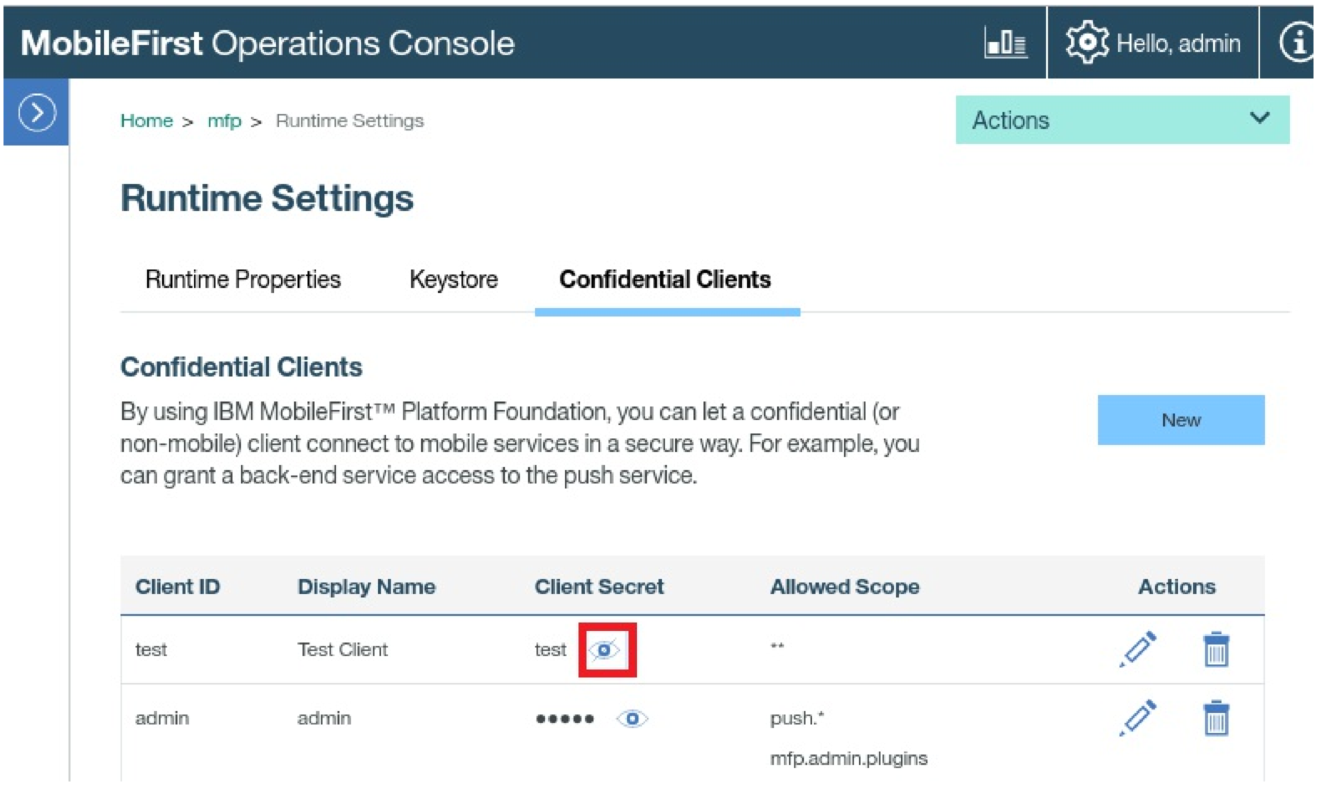


localhost:9080/mfpconsole/index.html#/mfp/settings#confidentialclients

oundation, you can let a confidential (or
ervices in a secure way. For example, you
o the push service.

Client Secret	Allowed Scope	Actions
•••••	**	
•••••	push.* mfp.admin.plugins	

2. You will notice that there are several confidential clients configured by default. On the first row with the **test** client ID, click on the eye button to see the password which is **test** as well. We will be using this client in the Swagger GUI for testing the adapter.



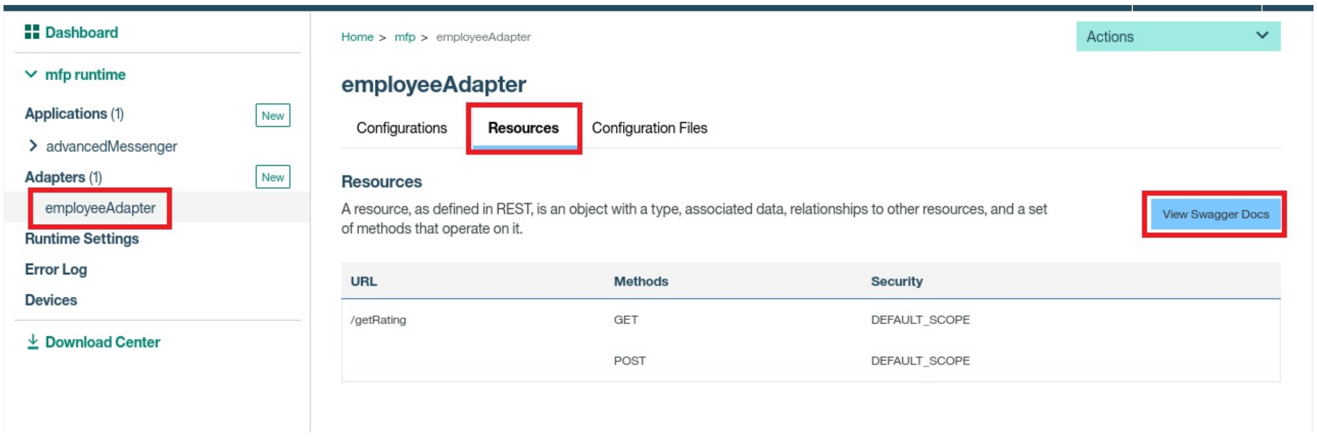
Runtime Settings

Confidential Clients

By using IBM MobileFirst™ Platform Foundation, you can let a confidential (or non-mobile) client connect to mobile services in a secure way. For example, you can grant a back-end service access to the push service.

Client ID	Display Name	Client Secret	Allowed Scope	Actions
test	Test Client	test	**	
admin	admin	●●●●●	push.* mfp.admin.plugins	

3. On the left side navigator, select **Adapter** > **employeeAdapter**. In the main pane, change to the **Resources** tab and select **View Swagger Docs** button.



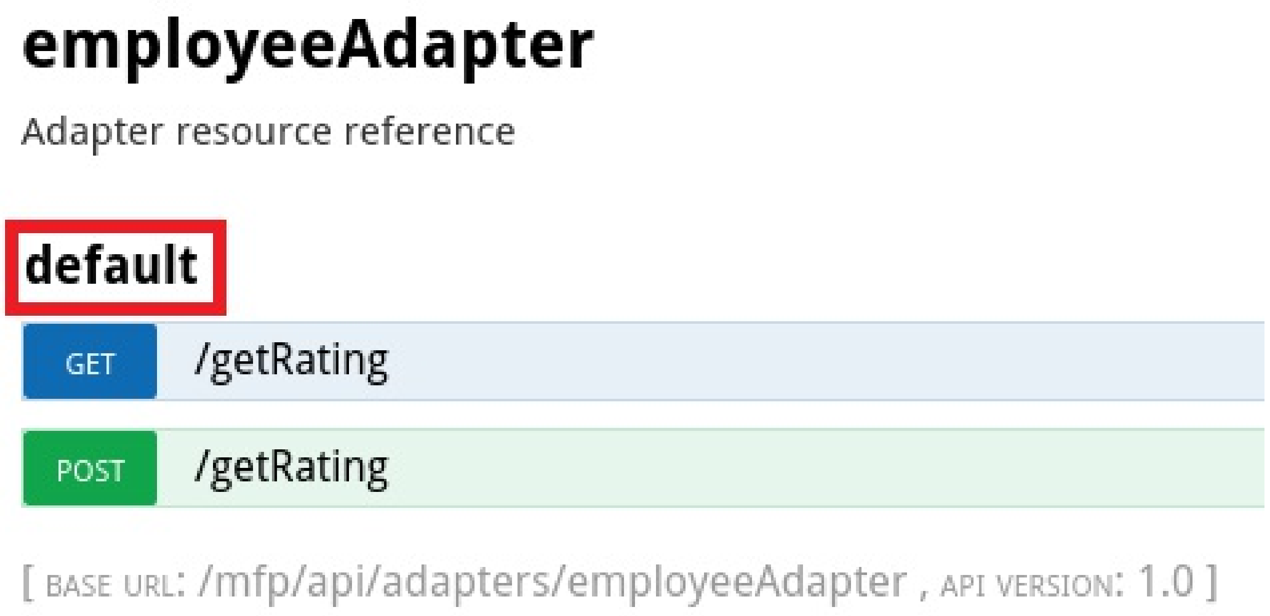
employeeAdapter

Resources

A resource, as defined in REST, is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it.

URL	Methods	Security
/getRating	GET	DEFAULT_SCOPE
	POST	DEFAULT_SCOPE

4. Click on the **default** link to expand it.

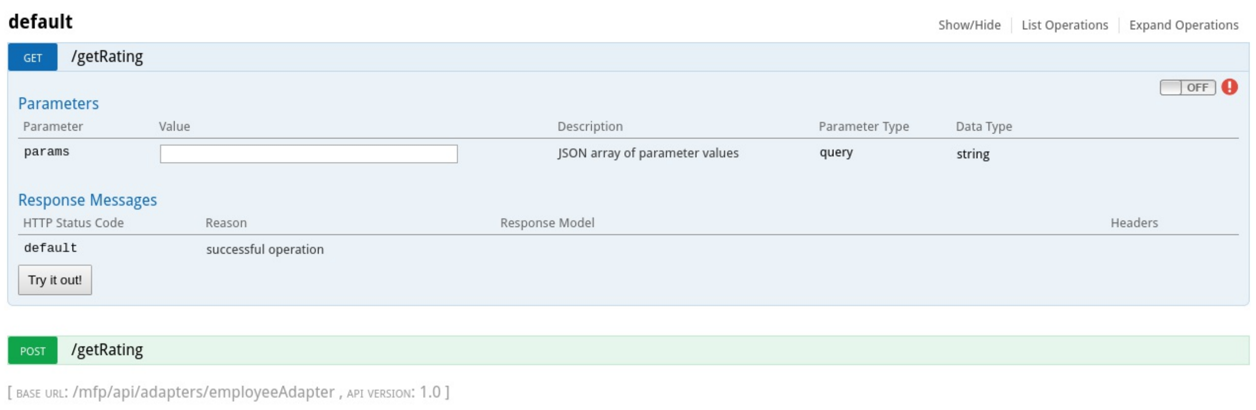

employeeAdapter

Adapter resource reference

default

[BASE URL: /mfp/api/adapters/employeeAdapter , API VERSION: 1.0]

5. Click on the **/getRating** link to expand it.


default

Parameters

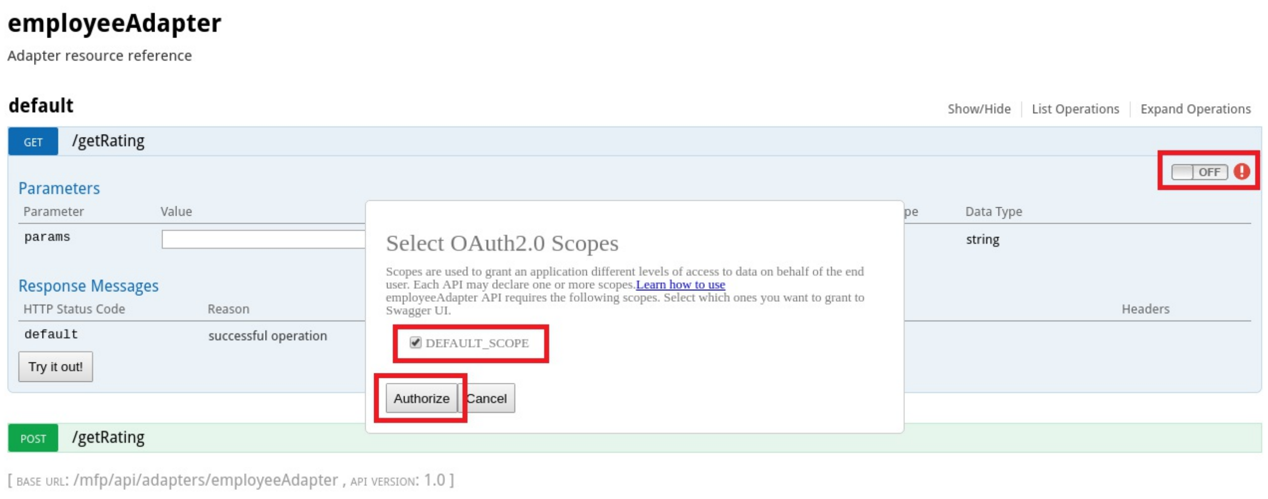
Parameter	Value	Description	Parameter Type	Data Type
params		JSON array of parameter values	query	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
default	successful operation		

[BASE URL: /mfp/api/adapters/employeeAdapter , API VERSION: 1.0]

6. On the far right, toggle the button and turn it on. As you toggle the button, a dialog will popup. Select **DEFAULT_SCOPE** in that dialog. Click **Authorize**.


employeeAdapter

Adapter resource reference

default

Parameters

Parameter Value pe Data Type

params string

Response Messages

HTTP Status Code Reason Headers

default successful operation

Select OAuth2.0 Scopes

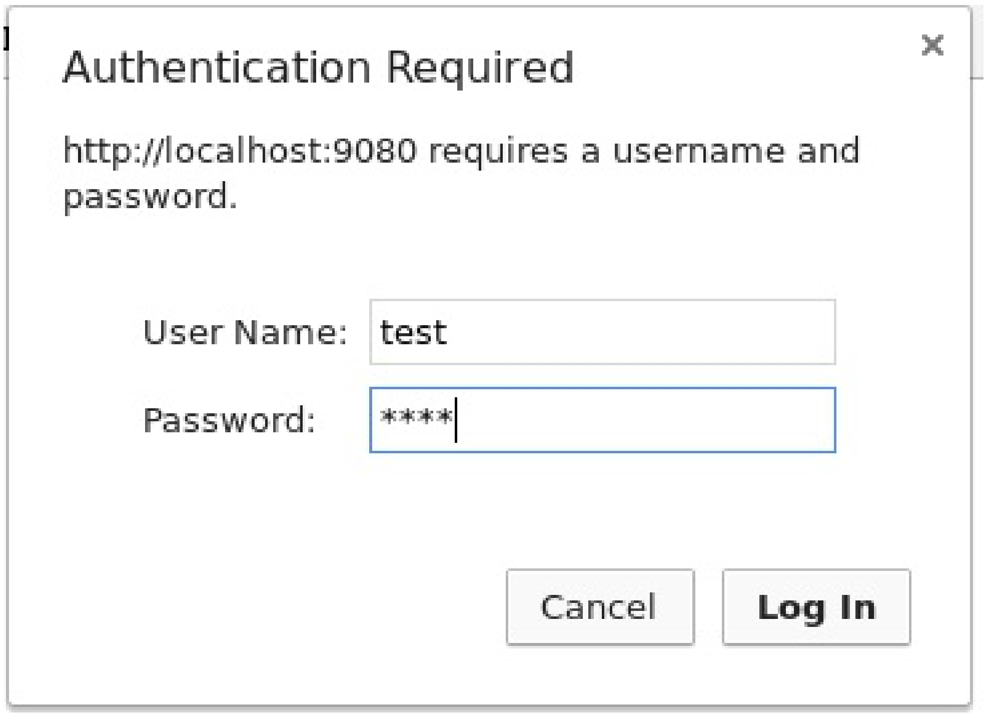
Scopes are used to grant an application different levels of access to data on behalf of the end user. Each API may declare one or more scopes.Learn how to use
employeeAdapter API requires the following scopes. Select which ones you want to grant to Swagger UI.

[✓ DEFAULT_SCOPE]

[BASE URL: /mfp/api/adapters/employeeAdapter , API VERSION: 1.0]

7. Enter **test / test** as the user name and password which is the pre-

defined *confidential client*. Click **Log in**.

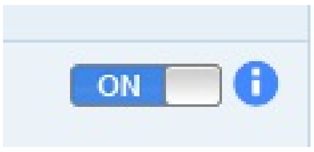

Authentication Required

http://localhost:9080 requires a username and password.

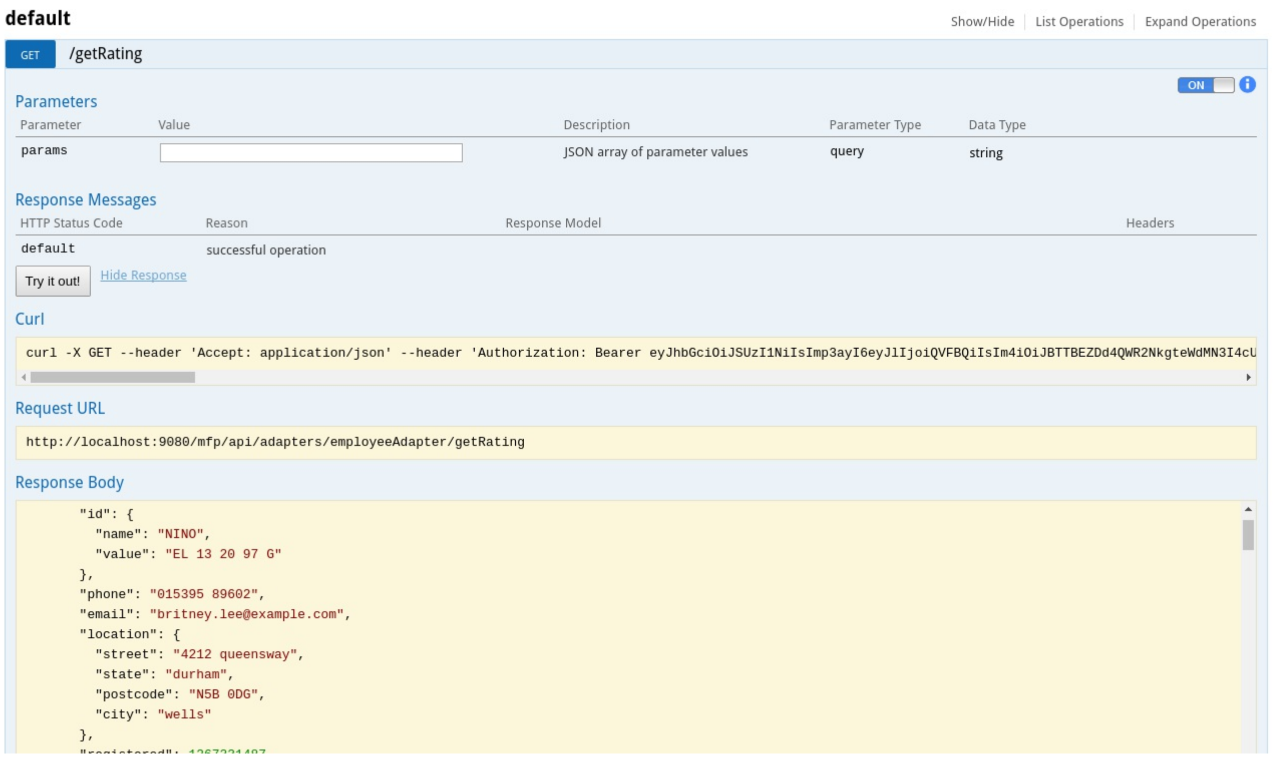
User Name: test

Password: ****

8. Once the login is successful, the toggle button becomes ON.



9. Now we will call the adapter method **getRating**. Click the **Try it Out** button. You will see a response being returned from the mock service and an HTTP 200 indicating a successful call. The adapter test is completed.


default

Show/Hide | List Operations | Expand Operations

GET /getRating

Parameters

Parameter	Value	Description	Parameter Type	Data Type
params		JSON array of parameter values	query	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
default	successful operation		

Curl

```
curl -X GET --header 'Accept: application/json' --header 'Authorization: Bearer eyJhbGciOiJSUzI1NiIsImp3ayI6eyJlIjoiQVFBQiIsIm4iOiJBTTBEZDd4QWR2NkgteWdMN3I4cU
```

Request URL

```
http://localhost:9080/mfp/api/adapters/employeeAdapter/getRating
```

Response Body

```
        "id": {
          "name": "NINO",
          "value": "EL 13 20 97 G"
        },
        "phone": "015395 89602",
        "email": "britney.lee@example.com",
        "location": {
          "street": "4212 queensway",
          "state": "durham",
          "postcode": "N5B 0DG",
          "city": "wells"
        },
```

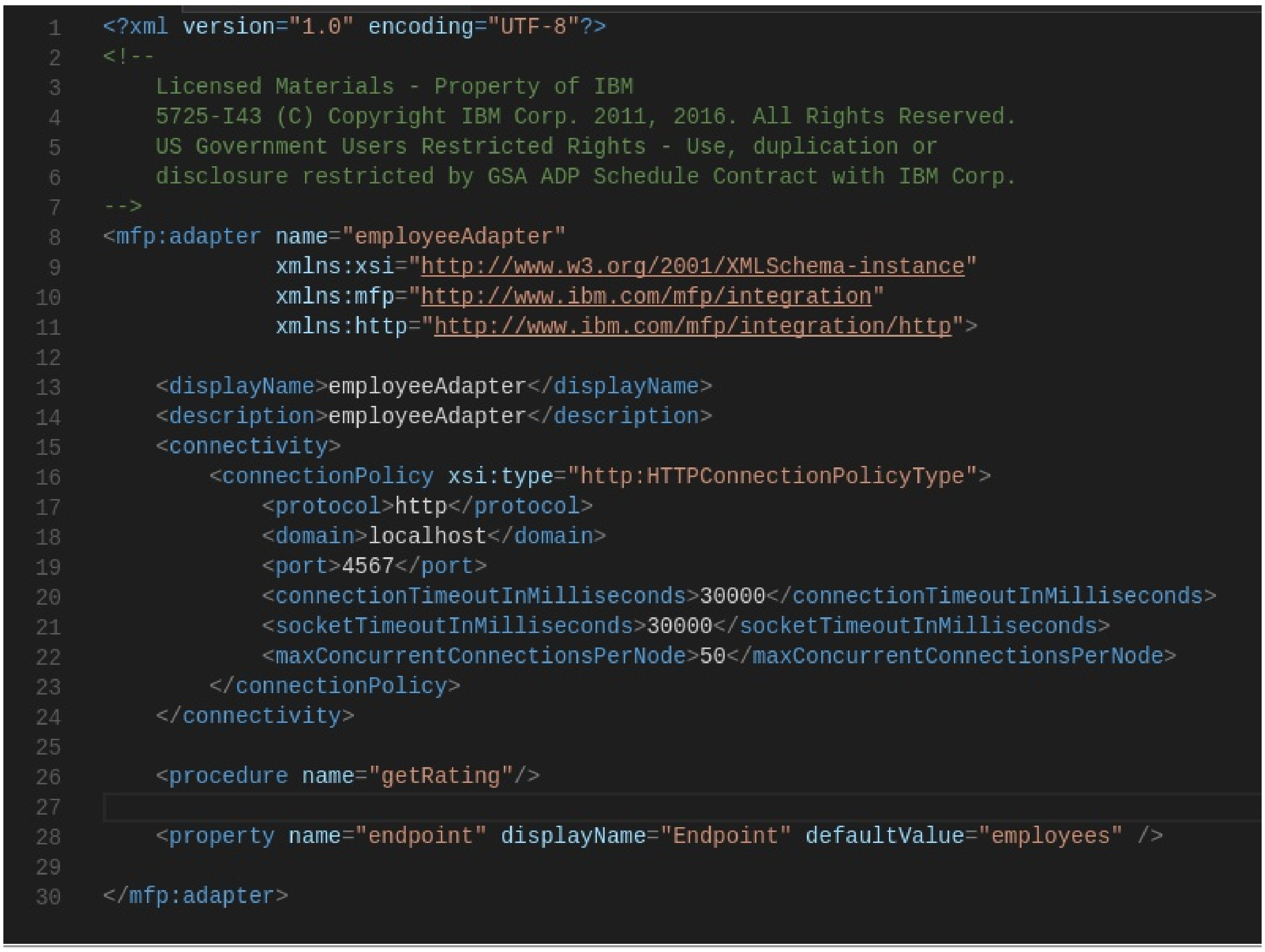
Creating a Dynamic Property in the Adapter

In the adapter, we can create a property that is dynamically changeable during runtime in the Operations console. This is an extremely useful feature as you move applications between different environments such as Dev, QA, Test, and Production, and configure resources or settings specific to each environment dynamically.

1. Back in the text editor, navigate the folders **employeeAdapter** > **src** > **main** > **adapter-resources** > **js** > and open the file **adapter.xml**.

2. Enter the following line in the XML file right after the `<procedure name="getRating">` and before `</mfp:adapter>` :

   `<property name="endpoint" displayName="Endpoint" defaultValue="employees" />`

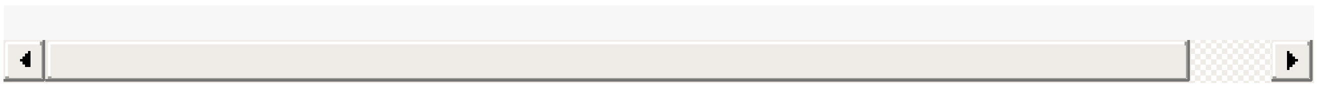


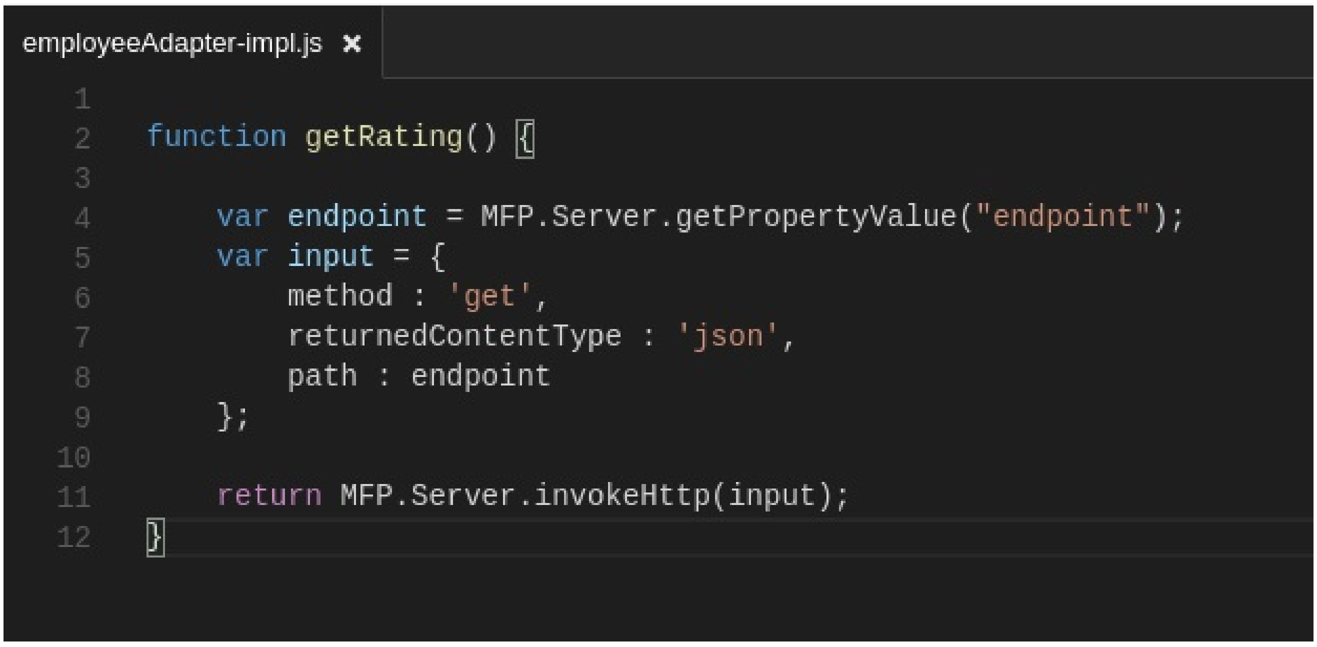
```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Licensed Materials - Property of IBM
    5725-I43 (C) Copyright IBM Corp. 2011, 2016. All Rights Reserved.
    US Government Users Restricted Rights - Use, duplication or
    disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<mfp:adapter name="employeeAdapter"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:mfp="http://www.ibm.com/mfp/integration"
             xmlns:http="http://www.ibm.com/mfp/integration/http">

    <displayName>employeeAdapter</displayName>
    <description>employeeAdapter</description>
    <connectivity>
        <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
            <protocol>http</protocol>
            <domain>localhost</domain>
            <port>4567</port>
            <connectionTimeoutInMilliseconds>30000</connectionTimeoutInMilliseconds>
            <socketTimeoutInMilliseconds>30000</socketTimeoutInMilliseconds>
            <maxConcurrentConnectionsPerNode>50</maxConcurrentConnectionsPerNode>
        </connectionPolicy>
    </connectivity>

    <procedure name="getRating"/>

    <property name="endpoint" displayName="Endpoint" defaultValue="employees" />

</mfp:adapter>
```

Save the file.

3. Open the file **employeeAdapter-impl.js** in the editor. Replace the getRating() method with the following code and save the file. This code can be found in the Lab02_Snippets.txt file located in the snippets directory for this lab. This modification will cause the adapter to use the dynamic property value set in the MFP Server.

```javascript
function getRating() {
    var endpoint = MFP.Server.getPropertyValue("endpoint
    var input = {
        method : 'get',
        returnedContentType : 'json',
        path : endpoint
    };

    return MFP.Server.invokeHttp(input);
}
```





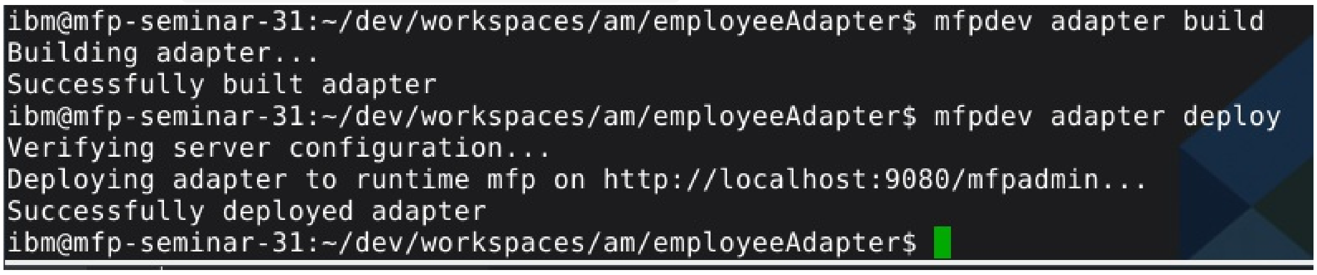
```javascript

function getRating() {

    var endpoint = MFP.Server.getPropertyValue("endpoint");
    var input = {
        method : 'get',
        returnedContentType : 'json',
        path : endpoint
    };

    return MFP.Server.invokeHttp(input);
}
```

Testing the Adapter

1. Return to the terminal window. We will now build and deploy the adapter using the following command:

   `mfpdev adapter build`

2. This time we will use the MobileFirst CLI to deploy the adapter. Deploy the adapter by entering the following command in the terminal: `mfpdev adapter deploy`

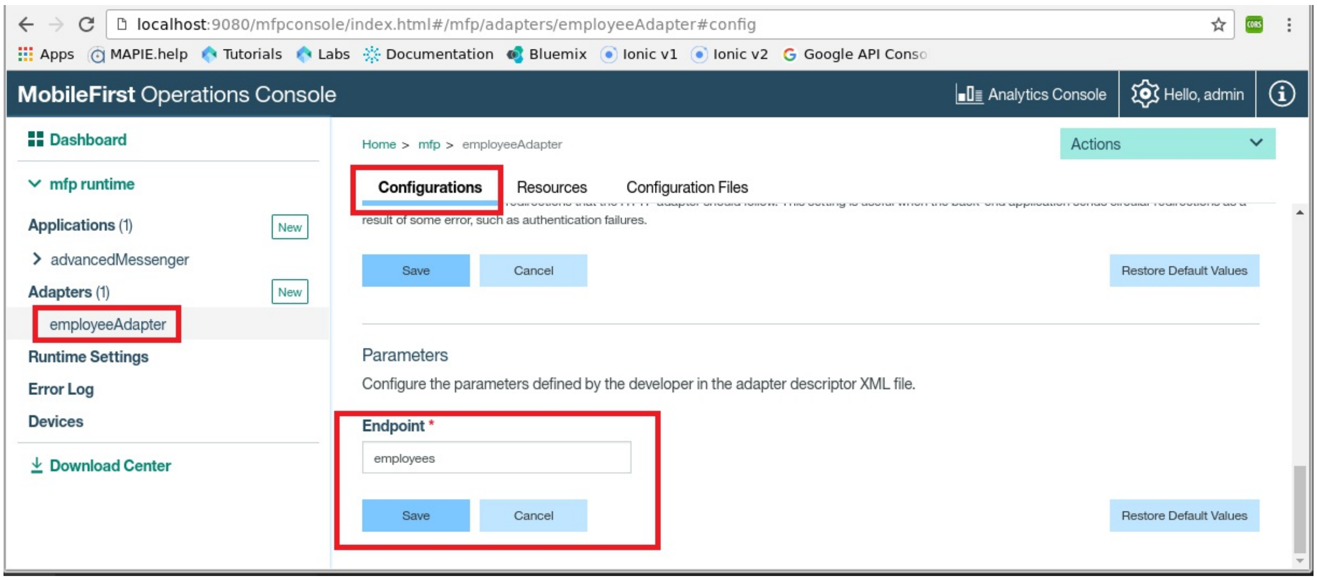


```
ibm@mfp-seminar-31:~/dev/workspaces/am/employeeAdapter$ mfpdev adapter build
Building adapter...
Successfully built adapter
ibm@mfp-seminar-31:~/dev/workspaces/am/employeeAdapter$ mfpdev adapter deploy
Verifying server configuration...
Deploying adapter to runtime mfp on http://localhost:9080/mfpadmin...
Successfully deployed adapter
ibm@mfp-seminar-31:~/dev/workspaces/am/employeeAdapter$
```

3. Back in the web browser, refresh the MobileFirst Operations Console. The URL is http://localhost:9080/mfpconsole.

4. Select the **employeeAdapter** on the left navigation. Select the **Configurations** tab in the main pane. Scroll down to the bottom to

see the newly created property in the main pane. You will see the **Endpoint** property.



localhost:9080/mfpconsole/index.html#/mfp/adapters/employeeAdapter#config

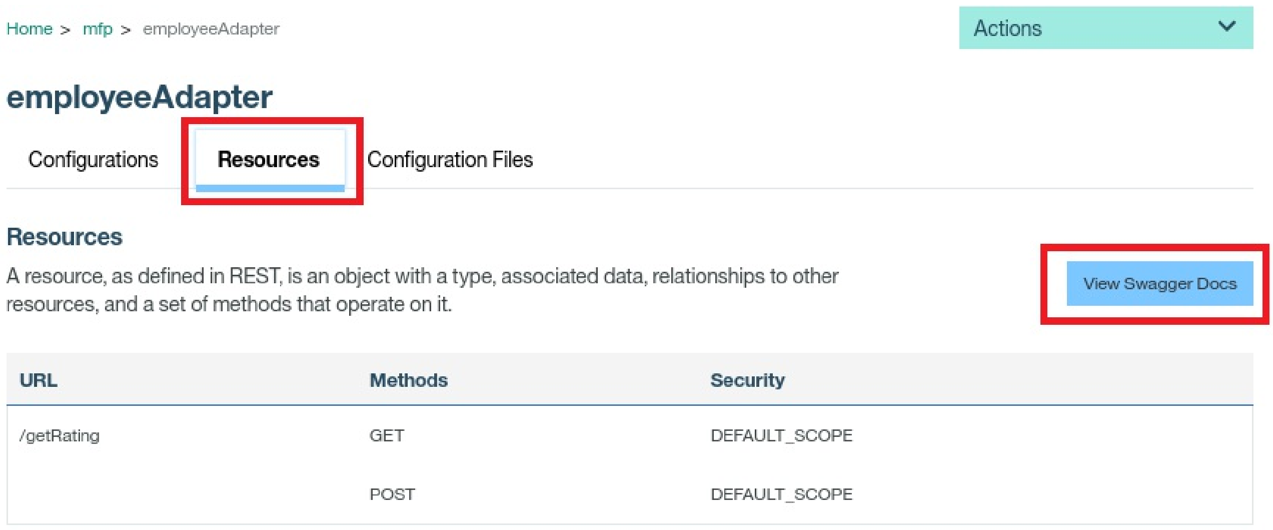
result of some error, such as authentication failures.

Parameters

Configure the parameters defined by the developer in the adapter descriptor XML file.

Endpoint *: employees

5. Go to the **Resources** tab and select **View Swagger Docs** button.



Resources

A resource, as defined in REST, is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it.

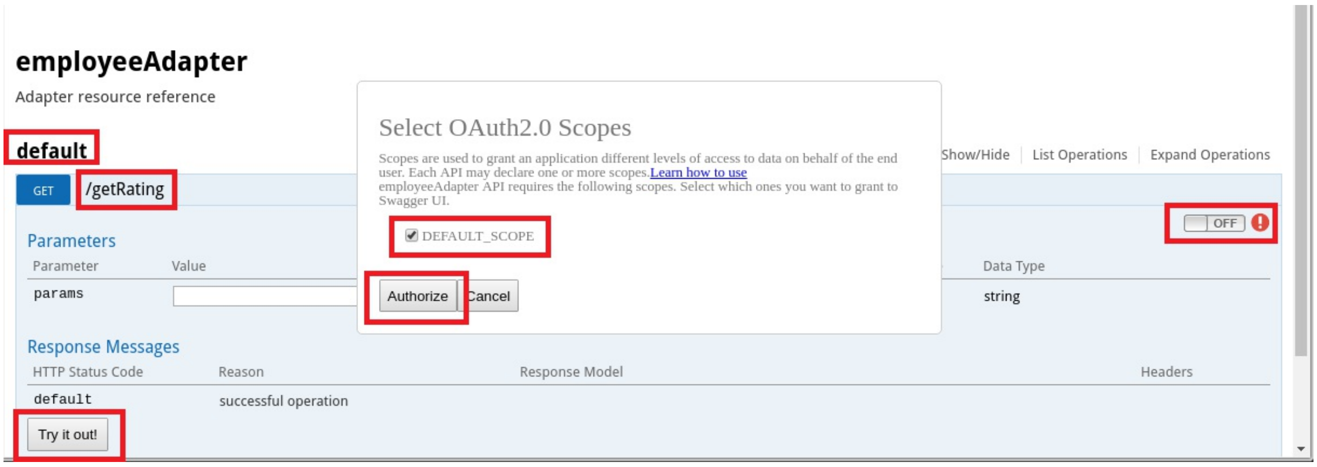
URL	Methods	Security
/getRating	GET	DEFAULT_SCOPE
	POST	DEFAULT_SCOPE

6. Test the adapter as before. Expand on the **default** link > expand on the **getRating** link.

7. Select the toggle for authorization to turn it on.

8. Check the **DEFAULT_SCOPE** check box and click **Authorize**.

9. Click on **Try it out** button to invoke the adapter.


employeeAdapter

Adapter resource reference

default

GET /getRating

Parameters

Parameter	Value	Data Type
params		string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
default	successful operation		

Select OAuth2.0 Scopes

Scopes are used to grant an application different levels of access to data on behalf of the end user. Each API may declare one or more scopes.Learn how to use employeeAdapter API requires the following scopes. Select which ones you want to grant to Swagger UI.

☑ DEFAULT_SCOPE

10. You will see that it's calling the **employees** endpoint on our mock server.

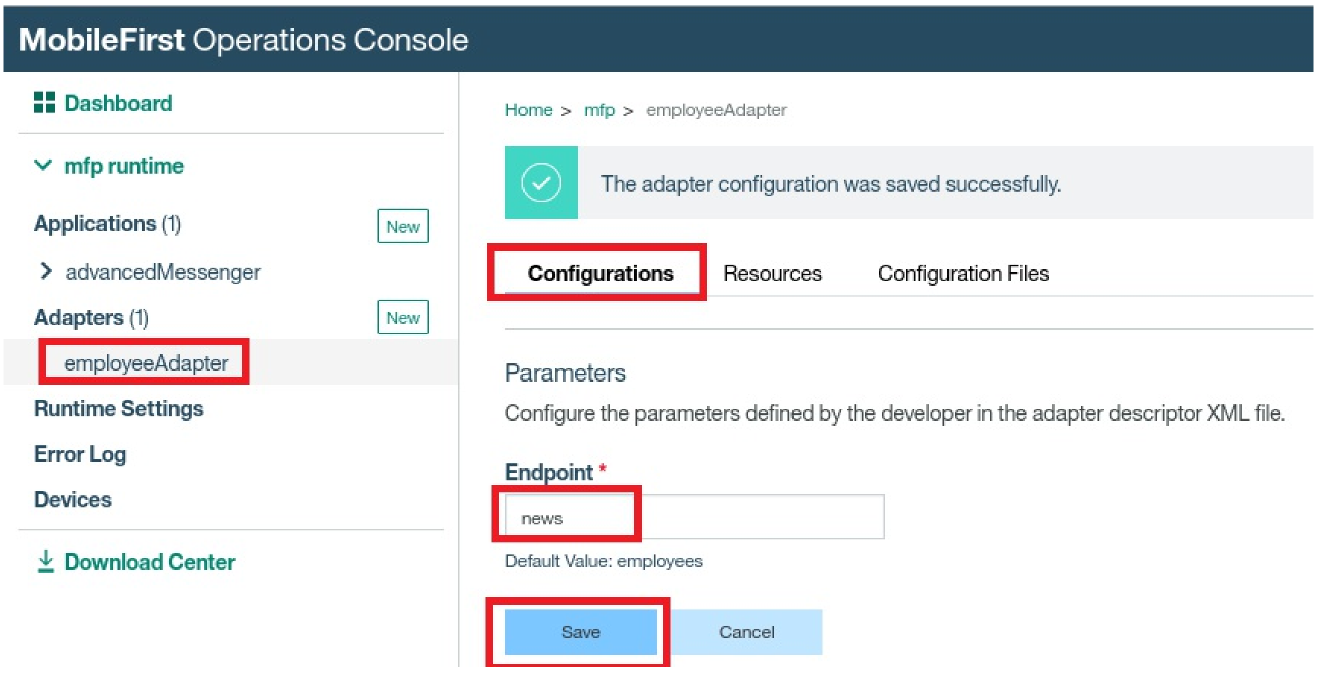
```
{
  "statusCode": 200,
  "isSuccessful": true,
  "results": [
    {
      "picture": {
        "thumbnail": "https://randomuser.me/api/portraits/thumb/women/23.jpg",
        "medium": "https://randomuser.me/api/portraits/med/women/23.jpg",
        "large": "https://randomuser.me/api/portraits/women/23.jpg"
      },
      "id": {
        "name": "NINO",
        "value": "EL 13 20 97 G"
      },
      "phone": "015395 89602",
      "email": "britney.lee@example.com",
      "location": {
        "street": "4212 queensway",
        "state": "durham",
        "postcode": "N5B 0DG",
```

Change the Dynamic Property in Admin Console

1. In the Operations Console, go to the **Configurations** tab of the employeeAdapter.

2. Change the endpoint property from **employees** to **news**. Click

Save.



The adapter configuration was saved successfully.

Parameters

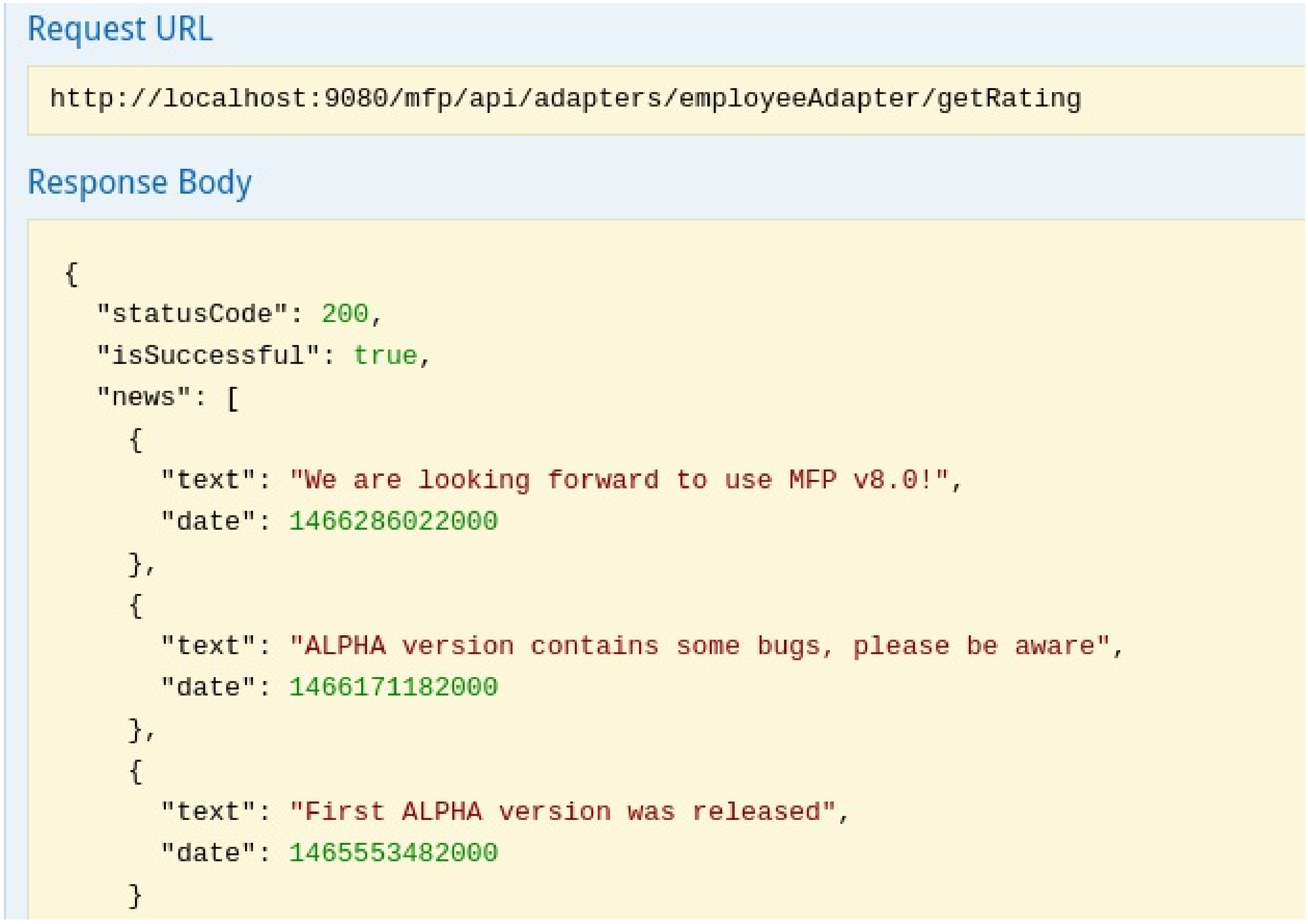
Configure the parameters defined by the developer in the adapter descriptor XML file.

Endpoint *

news

Default Value: employees

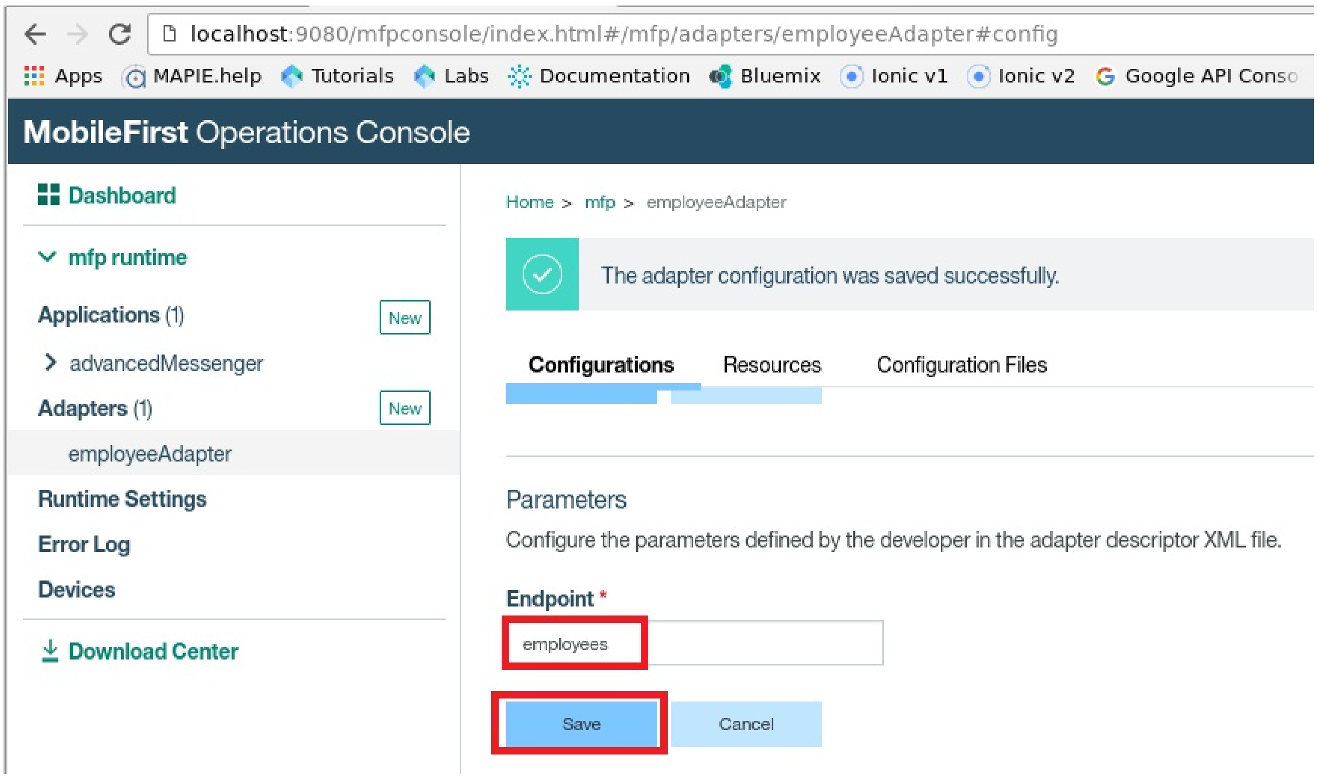
3. Go back to the Swagger GUI and click on the **Try it out** button. Notice that the adapter is now calling the **news** endpoint.



```
http://localhost:9080/mfp/api/adapters/employeeAdapter/getRating
```

```
{
  "statusCode": 200,
  "isSuccessful": true,
  "news": [
    {
      "text": "We are looking forward to use MFP v8.0!",
      "date": 1466286022000
    },
    {
      "text": "ALPHA version contains some bugs, please be aware",
      "date": 1466171182000
    },
    {
      "text": "First ALPHA version was released",
      "date": 1465553482000
    }
```

4. To clean up, change the property back to **employees**. Click **Save**.


localhost:9080/mfpconsole/index.html#/mfp/adapters/employeeAdapter#config
Apps MAPIE.help Tutorials Labs Documentation Bluemix Ionic v1 Ionic v2 Google API Conso
MobileFirst Operations Console
Dashboard
mfp runtime
Applications (1) New
advancedMessenger
Adapters (1) New
employeeAdapter
Runtime Settings
Error Log
Devices
Download Center
Home > mfp > employeeAdapter
Configurations
Resources
Configuration Files
Save
Cancel

The adapter configuration was saved successfully.

Parameters

Configure the parameters defined by the developer in the adapter descriptor XML file.

Endpoint *

employees

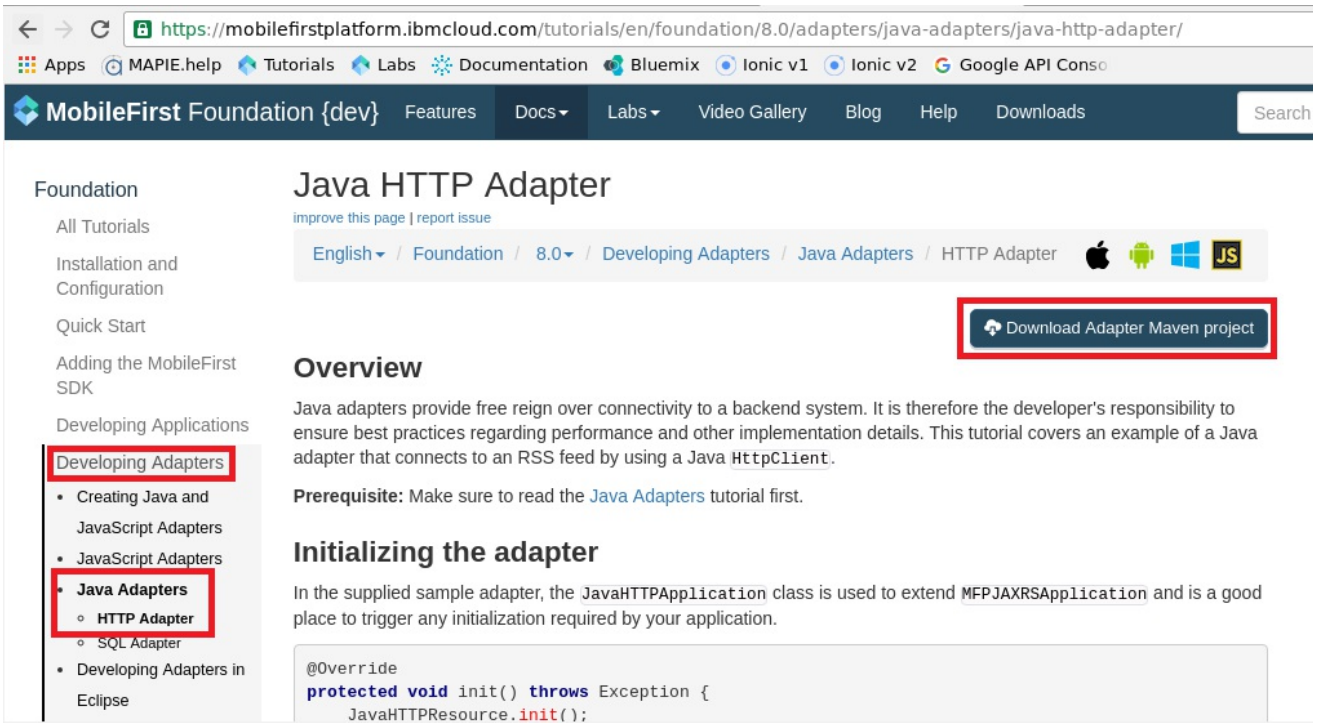
Create a Java Adapter

We will now create a Java Adapter for the **news** endpoint. First we will download the Java Adapter from the MobileFirst Developer Center tutorial and then modify it to suit our needs.

Although we will not be creating a dynamic property inside the Java Adapter in this lab, this feature is available just like we have used with the JavaScript adapter previously. You can use the `ConfigurationAPI` to obtain the dynamic property from the admin console from within the Java adapter code.

1. In a web browser, browse to the website URL: *https://mobilefirstplatform.ibmcloud.com/tutorials/en/foundation/8.0/all-tutorials/*
2. In the left navigator, go to **Developing Adapters** > **Java Adapters** > **HTTP Adapter**.
3. Click on **Download Adapter Maven Project** button. You will be

taken to a GitHub repository.



https://mobilefirstplatform.ibmcloud.com/tutorials/en/foundation/8.0/adapters/java-adapters/java-http-adapter/

Apps
MAPIE.help
Tutorials
Labs
Documentation
Bluemix
Ionic v1
Ionic v2
Google API Conso
MobileFirst Foundation {dev}
Features
Docs
Labs
Video Gallery
Blog
Help
Downloads
Search
Foundation
All Tutorials
Installation and Configuration
Quick Start
Adding the MobileFirst SDK
Developing Applications
Developing Adapters
Creating Java and JavaScript Adapters
JavaScript Adapters
Java Adapters
HTTP Adapter
SQL Adapter
Developing Adapters in Eclipse
improve this page | report issue
English / Foundation / 8.0 / Developing Adapters / Java Adapters / HTTP Adapter
Download Adapter Maven project

Java HTTP Adapter

Overview

Java adapters provide free reign over connectivity to a backend system. It is therefore the developer's responsibility to ensure best practices regarding performance and other implementation details. This tutorial covers an example of a Java adapter that connects to an RSS feed by using a Java `HttpClient`.

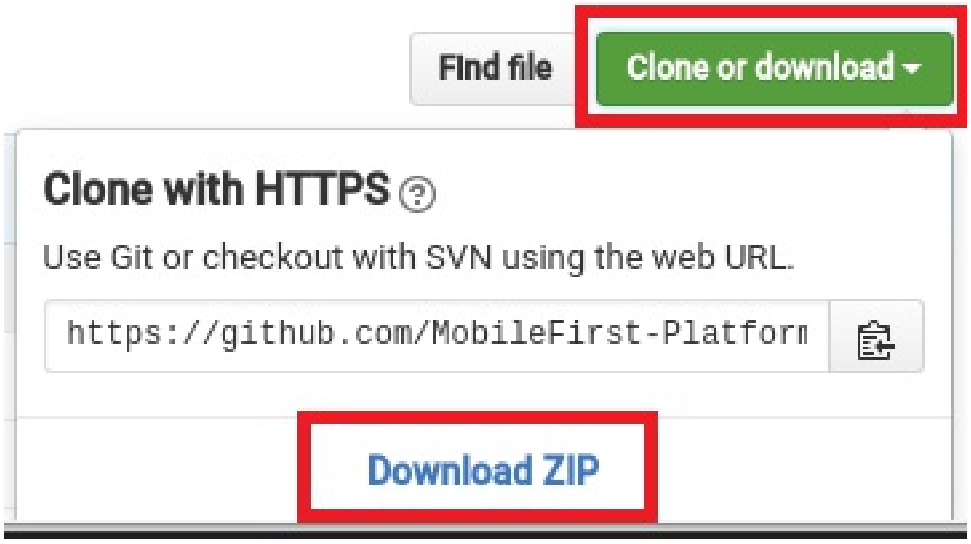
Prerequisite: Make sure to read the Java Adapters tutorial first.

Initializing the adapter

In the supplied sample adapter, the `JavaHTTPApplication` class is used to extend `MFPJAXRSApplication` and is a good place to trigger any initialization required by your application.

```
@Override
protected void init() throws Exception {
    JavaHTTPResource.init();
```

4. In the github page, click on the **Clone or download** button, then click on **Download ZIP** button.

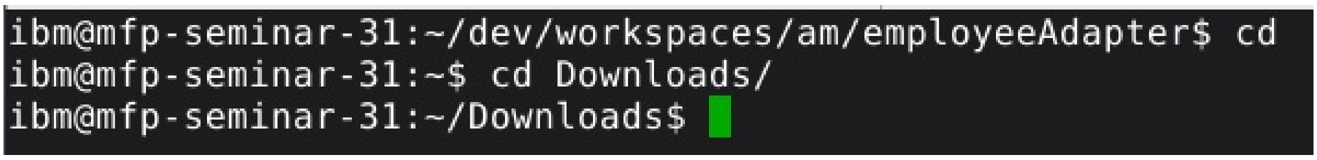


Use Git or checkout with SVN using the web URL.

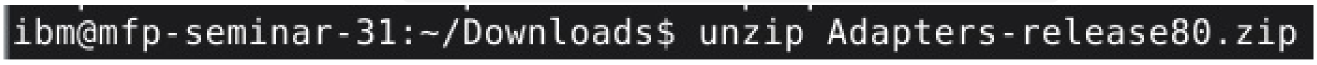
```
https://github.com/MobileFirst-Platform
```

5. Back in the terminal window, enter the following commands to change to the Downloads folder:

```
  cd
cd Downloads
```

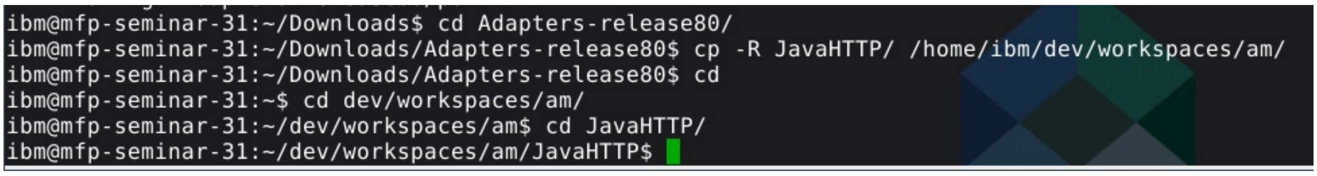

```
ibm@mfp-seminar-31:~/dev/workspaces/am/employeeAdapter$ cd
ibm@mfp-seminar-31:~$ cd Downloads/
ibm@mfp-seminar-31:~/Downloads$
```

6. Unzip the file using : `unzip Adapters-release80.zip`


```
ibm@mfp-seminar-31:~/Downloads$ unzip Adapters-release80.zip
```

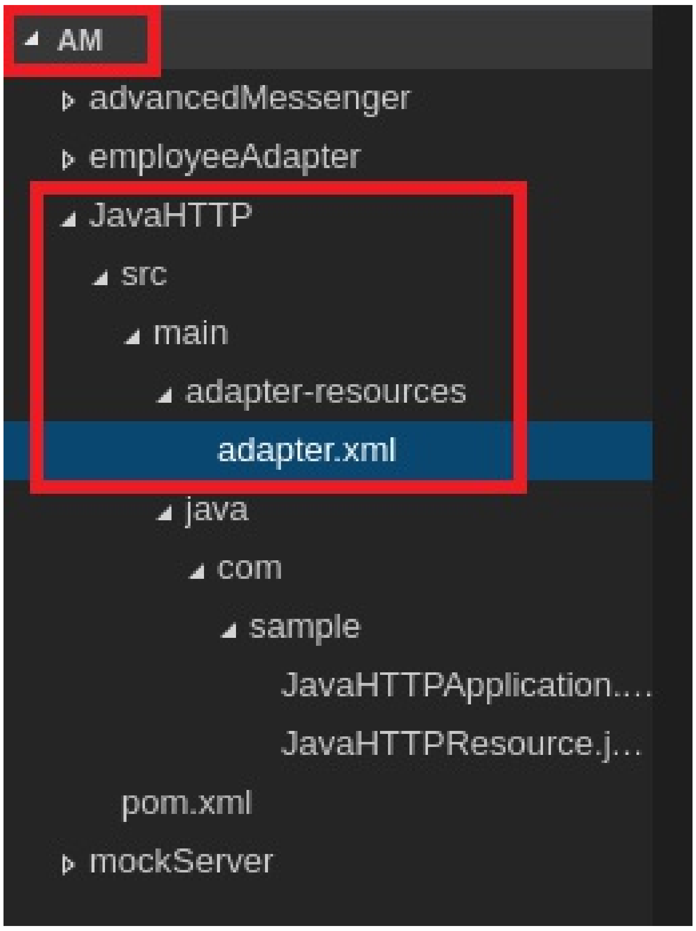
7. We will copy the unzipped content to our workspace directory. Enter the following command:

```
cd Adapters-release80/
cp -R JavaHTTP /home/ibm/dev/workspaces/am/
cd
cd dev/workspaces/am
cd JavaHTTP/
```

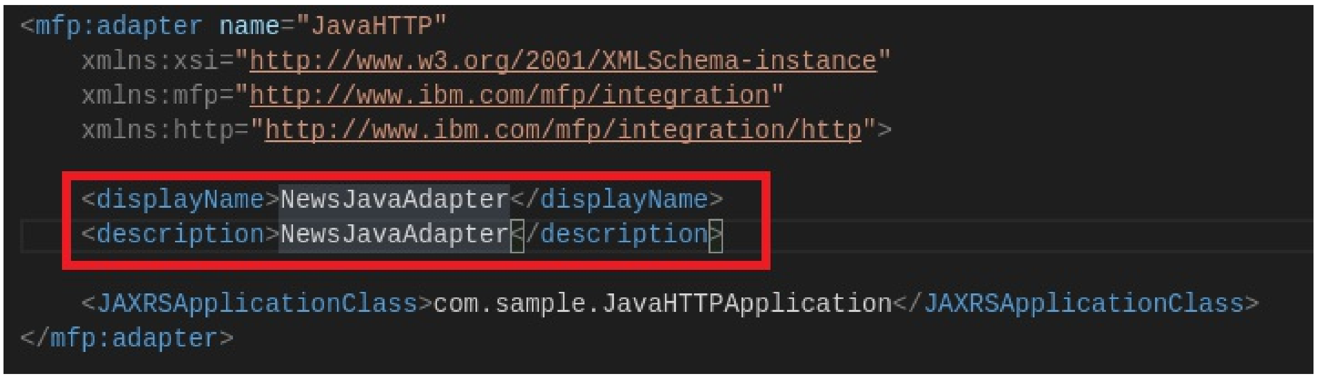

```
ibm@mfp-seminar-31:~/Downloads$ cd Adapters-release80/
ibm@mfp-seminar-31:~/Downloads/Adapters-release80$ cp -R JavaHTTP/ /home/ibm/dev/workspaces/am/
ibm@mfp-seminar-31:~/Downloads/Adapters-release80$ cd
ibm@mfp-seminar-31:~$ cd dev/workspaces/am/
ibm@mfp-seminar-31:~/dev/workspaces/am$ cd JavaHTTP/
ibm@mfp-seminar-31:~/dev/workspaces/am/JavaHTTP$
```

Modify the adapter.xml for the Java Adapter

1. In the text editor, expand the folders **am** > **JavaHTTP** > **src** > **main** > **adapter-resources**. Select **adapter.xml** and open it in the editor.



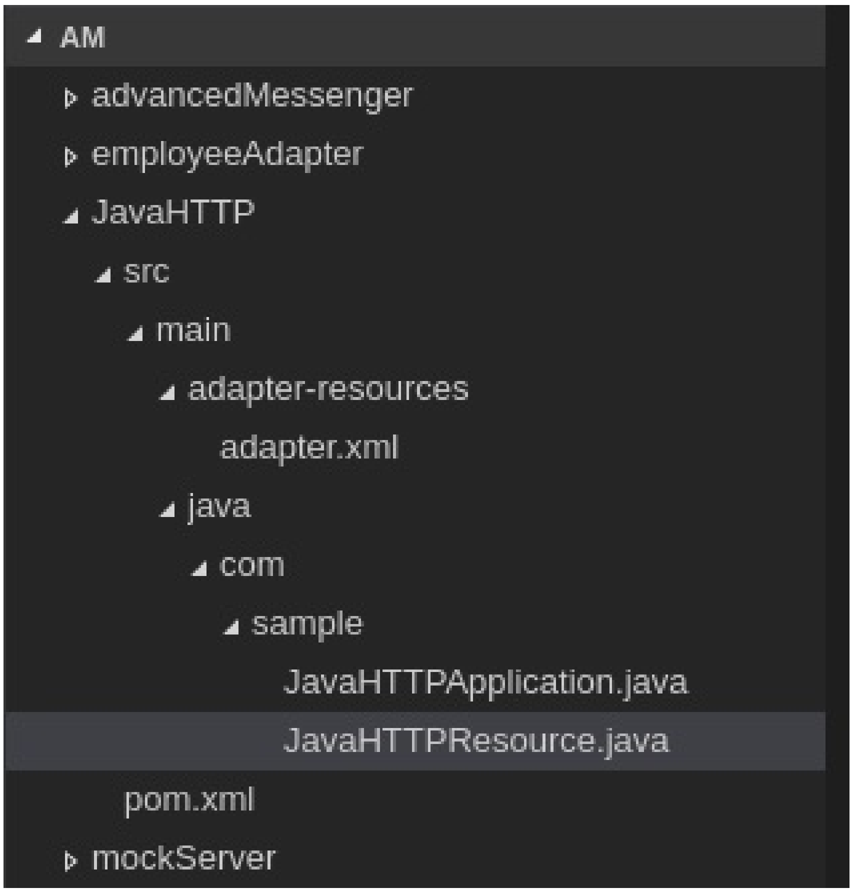
2. In the file, change the displayName and description to **NewsJavaAdapter**. **Save** the file.



```
<mfp:adapter name="JavaHTTP"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mfp="http://www.ibm.com/mfp/integration"
    xmlns:http="http://www.ibm.com/mfp/integration/http">

    <displayName>NewsJavaAdapter</displayName>
    <description>NewsJavaAdapter</description>

    <JAXRSApplicationClass>com.sample.JavaHTTPApplication</JAXRSApplicationClass>
</mfp:adapter>
```

Modify the Java Adapter Code

1. Expand the folders **am** > **JavaHTTP** > **src** > **main** > **java**> **com** > **sample**. Select the file **JavaHTTPResource.java** and open it in the editor.



2. Replace the entire class with the following code. We are changing the host name and port number to point to our mock server, and also simplifying the GET operation to go to the **/news** endpoint. **Save** the file. You can find the code in the snippet.

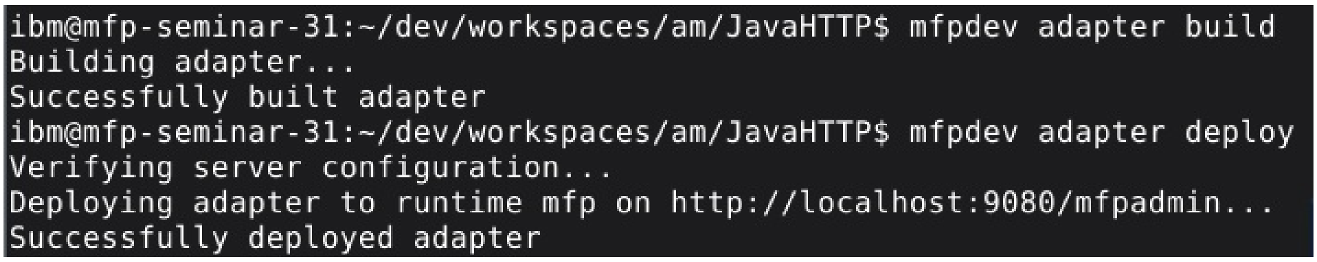
```
package com.sample;

import org.apache.http.HttpHost;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpUriRequest;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.wink.json4j.utils.XML;
import org.xml.sax.SAXException;

import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.GET;
```

```java
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import java.io.IOException;
import java.nio.charset.Charset;
import com.ibm.json.java.JSONObject;

@Path("/")
public class JavaHTTPResource {

    private static CloseableHttpClient client;
    private static HttpHost host;

    public static void init() {
        client = HttpClientBuilder.create().build();
        host = new HttpHost("localhost", 4567, "http");
    }

    public void execute(HttpUriRequest req, HttpServlet
            throws IOException,
            IllegalStateException, SAXException {
        HttpResponse JSONResponse = client.execute(host,
        ServletOutputStream os = resultResponse.getOutpu
        if (JSONResponse.getStatusLine().getStatusCode(
            resultResponse.addHeader("Content-Type", "a
            JSONObject result = JSONObject.parse(JSONRes
            String json = result.toString();
            os.write(json.getBytes(Charset.forName("UTF-

        }else{
            resultResponse.setStatus(JSONResponse.getSta
            JSONResponse.getEntity().getContent().close(
            os.write(JSONResponse.getStatusLine().getRea
        }
        os.flush();
        os.close();
    }
    @GET
    @Produces("application/json")
    public void get(@Context HttpServletResponse respons
            throws IOException, IllegalStateException, S

            execute(new HttpGet("/news"), response);

    }

}
```

Build and Deploy the Adapter

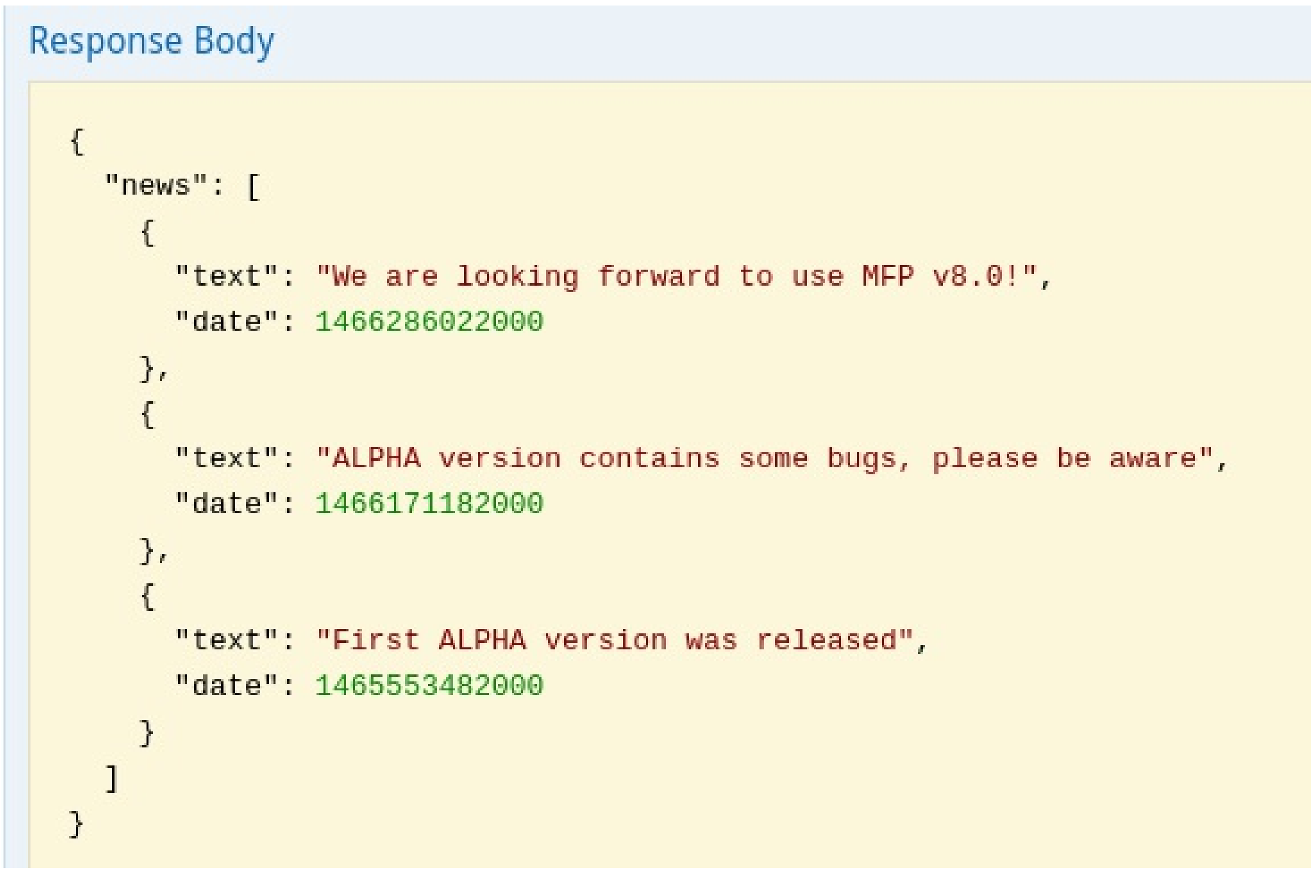
1. In the terminal window, ensure you are in the **/dev/workspaces/am/JavaHTTP** folder. Enter the following command to build and deploy the adapter:

```
 mfpdev adapter build
mfpdev adapter deploy
```



```
ibm@mfp-seminar-31:~/dev/workspaces/am/JavaHTTP$ mfpdev adapter build
Building adapter...
Successfully built adapter
ibm@mfp-seminar-31:~/dev/workspaces/am/JavaHTTP$ mfpdev adapter deploy
Verifying server configuration...
Deploying adapter to runtime mfp on http://localhost:9080/mfpadmin...
Successfully deployed adapter
```

2. Feel free to test it in Swagger as before.

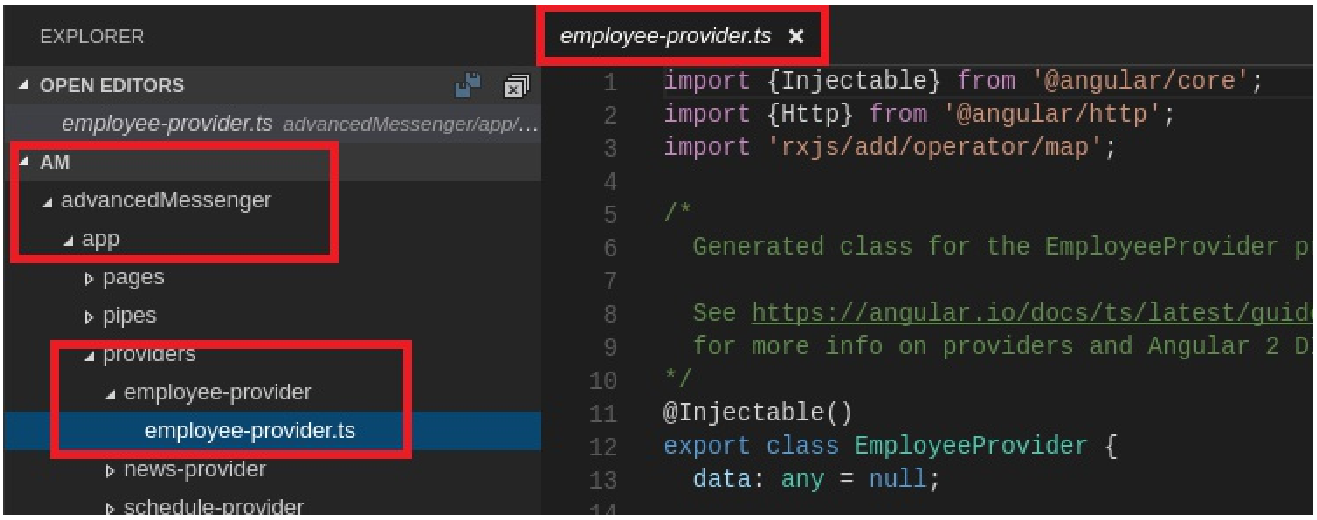

Response Body

```
{
  "news": [
    {
      "text": "We are looking forward to use MFP v8.0!",
      "date": 1466286022000
    },
    {
      "text": "ALPHA version contains some bugs, please be aware",
      "date": 1466171182000
    },
    {
      "text": "First ALPHA version was released",
      "date": 1465553482000
    }
  ]
}
```

Modify the App to use EmployeesAdapter

Now we will modify the demo Ionic app to actually use these adapters. The employees adapter will be used in the employee-provider.ts and the news adapter will be used in the news-provider.ts. These are both Angular providers, which the Ionic framework uses, typically implemented to implement the data model for the app. These providers will call the MobileFirst adapters to provide the data needed by the application.

1. In the editor, navigate to the following file and open it: `/am/advancedMessenger/app/providers/employee-provider/employee-provider.ts.`



```
import {Injectable} from '@angular/core';
import {Http} from '@angular/http';
import 'rxjs/add/operator/map';

/*
  Generated class for the EmployeeProvider p

  See https://angular.io/docs/ts/latest/guid
  for more info on providers and Angular 2 D
*/
@Injectable()
export class EmployeeProvider {
  data: any = null;
```

2. Replace the entire file with the following code (which is available in the snippets file). The code calls the MobileFirst employeesAdapter using a `WLResourceRequest` which is the MobileFirst API for accessing a protected backend resource. The URL for the adapter is `/adapters/employeeAdapter/getRating`. **Save** the file.

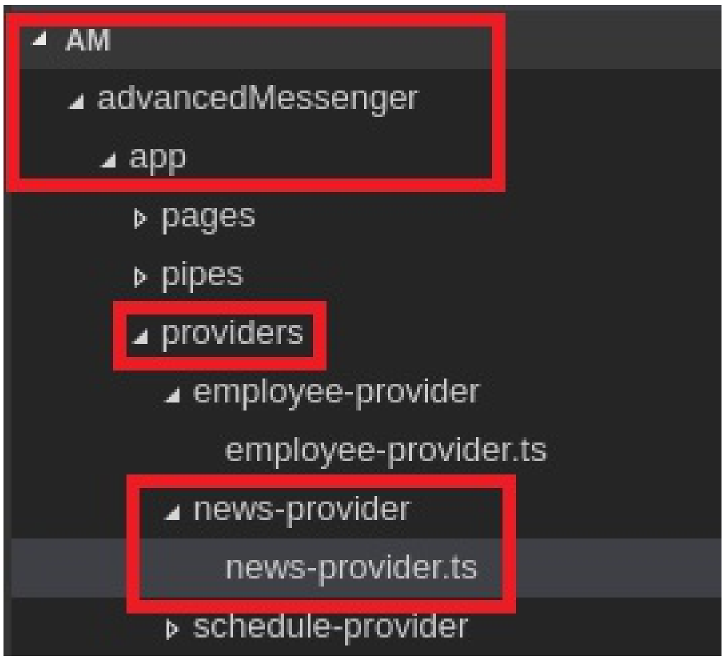
```
import {Injectable} from '@angular/core';

@Injectable()
export class EmployeeProvider {
  data: any = null;

  constructor() {}

  load() {
    console.log('---> called EmployeeProvider load');

    if (this.data) {
      // already loaded data
      return Promise.resolve(this.data);
    }

    // don't have the data yet
    return new Promise(resolve => {
      // We're using Angular Http provider to request th
      // then on the response it'll map the JSON data to
      // Next we process the data and resolve the promis
```

```
        let dataRequest = new WLResourceRequest ("/adapte
        dataRequest.send().then((response)=> {
          console.log ('--> data loaded from adapter', res
          this.data=response.responseJSON.results;
          resolve(this.data);
        }, (failure) => {
          console.log ('--> failed to load data', failure
          resolve ('error');
        })

      });
    }
  }
```

Modify the App to use NewsJavaAdapter

Next up, we will modify the news-provider.ts provider to use the NewsAdapter.

1. In the editor, navigate to the following file and open it: `/am/advancedMessenger/app/providers/news-provider/news-provider.ts`.



2. Replace the entire file with the following code (available in snippets file). Similar to the previous adapter, this code calls the NewsAdapter using a `WLResourceRequest`. The URL for the adapter is `/adapters/JavaHTTP/`. **Save** the file.

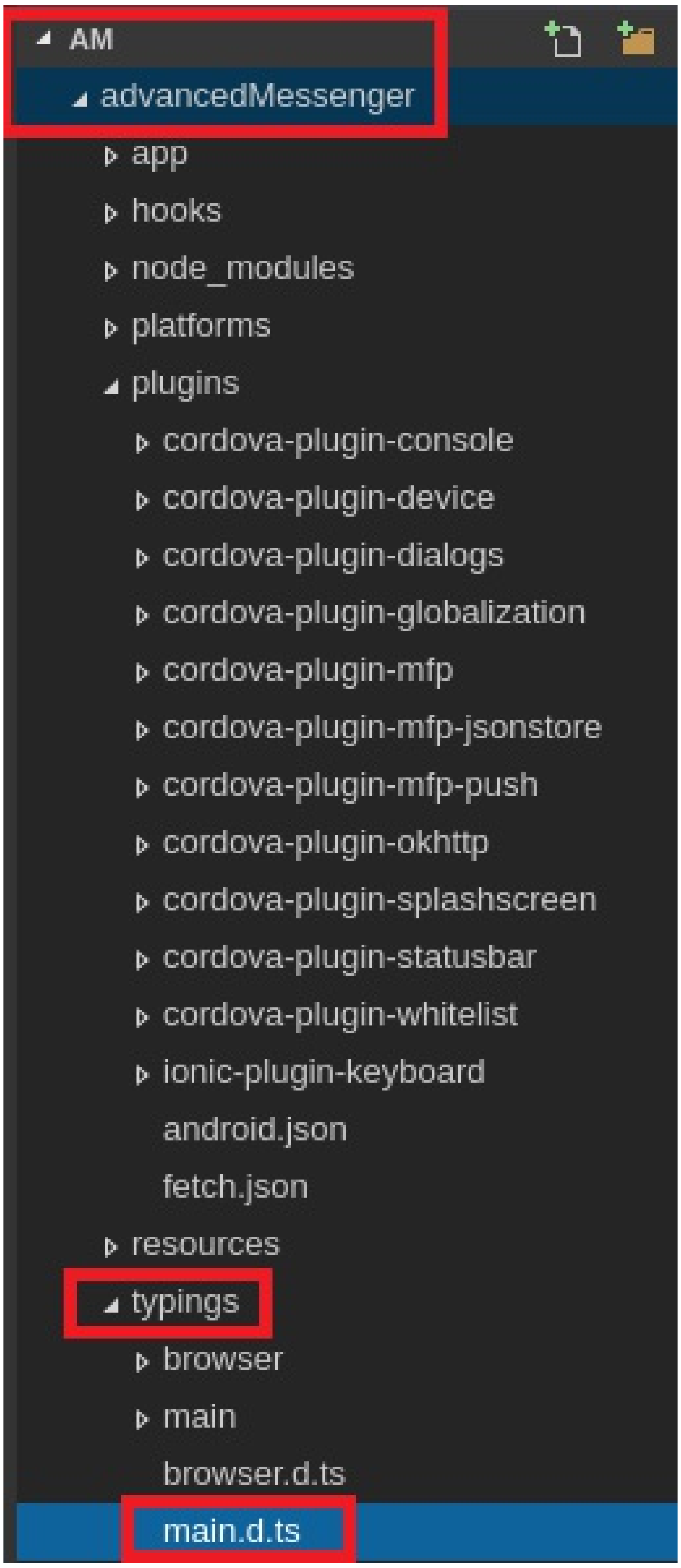
```
import {Injectable} from '@angular/core';

@Injectable()
export class NewsProvider {
  data: any = null;

  constructor() {}

  load() {
    console.log('---> called NewsProvider load');

    if (this.data) {
      // already loaded data
      return Promise.resolve(this.data);
    }

    // don't have the data yet
    return new Promise(resolve => {
      // We're using Angular Http provider to request th
```

```
         // then on the response it'll map the JSON data t
         // Next we process the data and resolve the promis
        let dataRequest = new WLResourceRequest ("/adapter
         dataRequest.send().then((response)=> {
           console.log ('--> data loaded from adapter', res
           this.data=response.responseJSON.news;
           resolve(this.data);
         }, (failure) => {
           console.log ('--> failed to load data', failure
           resolve ('error');
         })
       });
     }
   }
```

Add the MobileFirst Typings into the App

In this step, we will add the appropriate *typings* entries to our project's typings file. These files define the types in use by our project, including those used in the MobileFirst Cordova plugins, which are necessary to successfully compile our TypeScript-based project.

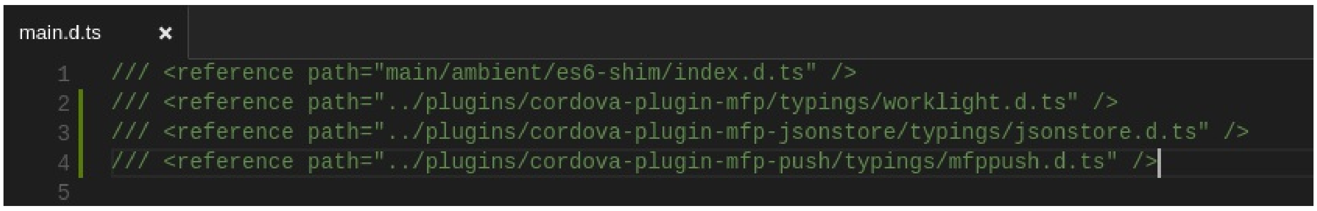
1. In the editor, navigate to the following file and open it to the editor: `/am/advancedMessenger/typings/main.d.ts.`



```
AM
  advancedMessenger
    app
    hooks
    node_modules
    platforms
    plugins
      cordova-plugin-console
      cordova-plugin-device
      cordova-plugin-dialogs
      cordova-plugin-globalization
      cordova-plugin-mfp
      cordova-plugin-mfp-jsonstore
      cordova-plugin-mfp-push
      cordova-plugin-okhttp
      cordova-plugin-splashscreen
      cordova-plugin-statusbar
      cordova-plugin-whitelist
      ionic-plugin-keyboard
      android.json
      fetch.json
    resources
    typings
      browser
      main
      browser.d.ts
      main.d.ts
```

2. Replace the entire file with the following lines. **Save** the file.

```
/// <reference path="main/ambient/es6-shim/index.d.ts" /
/// <reference path="../plugins/cordova-plugin-mfp/typin
/// <reference path="../plugins/cordova-plugin-mfp-jsons
/// <reference path="../plugins/cordova-plugin-mfp-push/
```

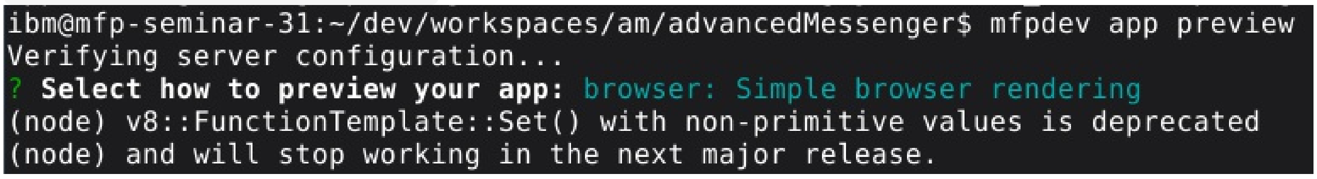


```
main.d.ts
/// <reference path="main/ambient/es6-shim/index.d.ts" />
/// <reference path="../plugins/cordova-plugin-mfp/typings/worklight.d.ts" />
/// <reference path="../plugins/cordova-plugin-mfp-jsonstore/typings/jsonstore.d.ts" />
/// <reference path="../plugins/cordova-plugin-mfp-push/typings/mfppush.d.ts" />
```

Preview the App

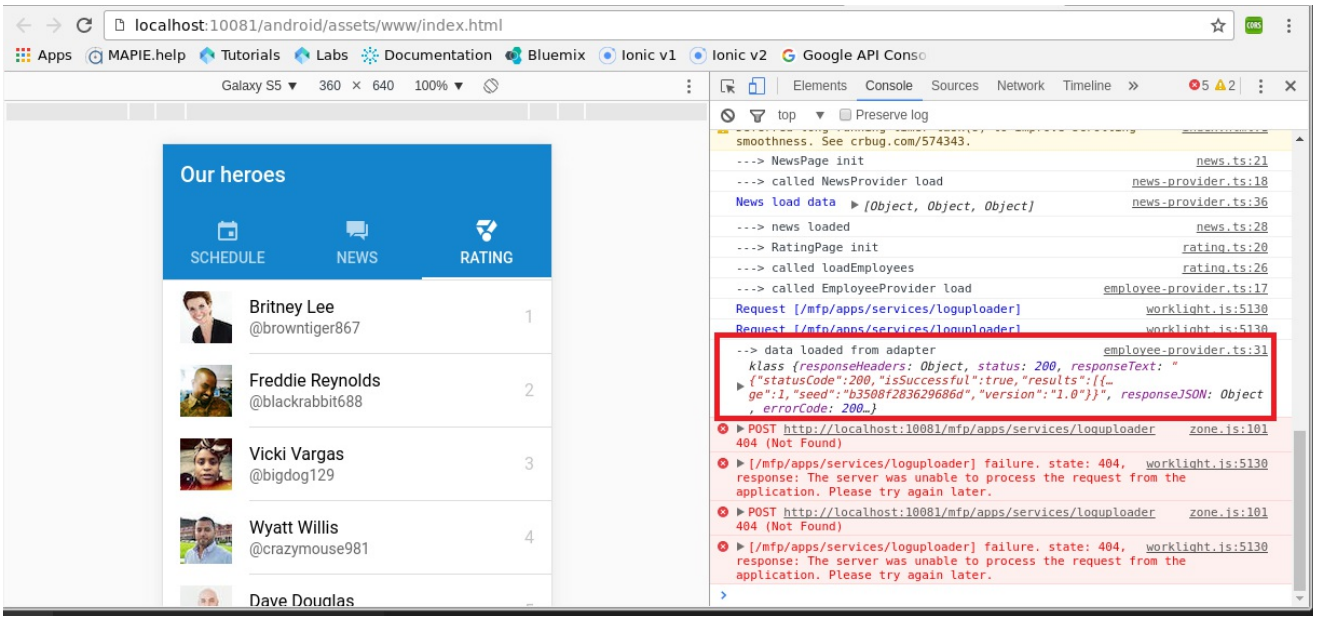
1. Return to the terminal window. Make sure the directory is **/dev/workspaces/am/advancedmessenger**. Enter the following command to preview the app and select the **Simple Browsing Rendering.**

```
mfpdev app preview
```



```
ibm@mfp-seminar-31:~/dev/workspaces/am/advancedMessenger$ mfpdev app preview
Verifying server configuration...
? Select how to preview your app: browser: Simple browser rendering
(node) v8::FunctionTemplate::Set() with non-primitive values is deprecated
(node) and will stop working in the next major release.
```

2. A new browser is opened. Open the **developer tools** to see the log. You can ignore the error messages, they will not appear when running in an actual device or emulator. Switch to the **Rating** tab and you can see that the employees-provider.ts is called to use the adapter content.



```
smoothness. See crbug.com/574343.
---> NewsPage init                                              news.ts:21
---> called NewsProvider load                          news-provider.ts:18
News load data  ▶ [Object, Object, Object]             news-provider.ts:36
---> news loaded                                                news.ts:28
---> RatingPage init                                          rating.ts:20
---> called loadEmployees                                     rating.ts:26
---> called EmployeeProvider load                  employee-provider.ts:17
Request [/mfp/apps/services/loguploader]                  worklight.js:5130
Request [/mfp/apps/services/loguploader]                  worklight.js:5130
--> data loaded from adapter                       employee-provider.ts:31
  klass {responseHeaders: Object, status: 200, responseText: "
▶ {"statusCode":200,"isSuccessful":true,"results":[{…
  ge":1,"seed":"b3508f283629686d","version":"1.0"}}", responseJSON: Object
  , errorCode: 200…}
▶ POST http://localhost:10081/mfp/apps/services/loguploader     zone.js:101
404 (Not Found)
▶ [/mfp/apps/services/loguploader] failure. state: 404,   worklight.js:5130
response: The server was unable to process the request from the
application. Please try again later.
▶ POST http://localhost:10081/mfp/apps/services/loguploader     zone.js:101
404 (Not Found)
▶ [/mfp/apps/services/loguploader] failure. state: 404,   worklight.js:5130
response: The server was unable to process the request from the
application. Please try again later.
```

Summary

In this lab, we have created two adapters, one implemented in JavaScript and the other in Java. Both adapters can be dynamically configured via the admin console and tested via a built-in Swagger interface. Adapters provide an easy and simple way to call to external resources and are enabled by default with basic security. They can be extended to more advanced use cases with multi-adapter mashup and sophisticated security policies including social login and multi-factor / step-up authentication.