InterConnect
2017

March 19–23
MGM Grand &
Mandalay Bay
Las Vegas, NV

Hands-on Lab

Session 7502

API Connect Toolkit and DataPower working seamlessly together

Alex Irazabal, IBM

Tony French, IBM

Introduction

During lab 7502 you will experience the API Connect Tookit, API Designer and DataPower for Docker all working seamlessly together.

DataPower provides a rich set of API policies for transforming and securing API workloads and you will now get the opportunity to experiment with rate-limiting, JWT security and SOAP transformation during this lab.

The API Toolkit manages DataPower via Docker for you, so you can focus on the API definitions and implementation.

Some key benefits of this feature include:

- Develop and test APIs that use the full complement of IBM API Connect Assembly policies.
- Saved changes are instantly synchronized with DataPower running as a Docker container for rapid testing and feedback.
- Test product and plan level concepts such as rate-limiting.
- DataPower error logging is integrated into the API Designer logging console.
- Request/Response logging is also available from the logging console with latency information.

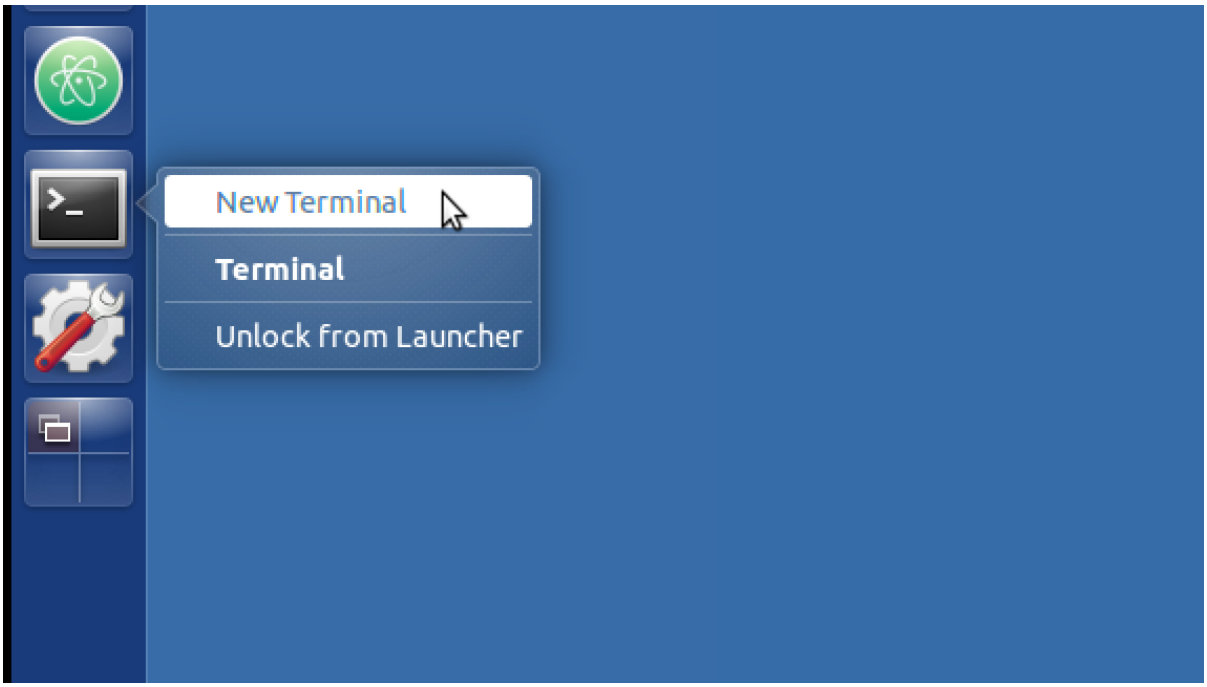
All source content for this lab is available from GitHub at https://github.com/ibm-datapower/interconnect-labs/tree/master/IC17-7502-DataPower-APIConnect-Toolkit so don't worry if you don't get it completed today.

Note: this capability requires API Connect version 5070.

Rate-limit a simple Loopback API

To get started we are going to use the API Connect Toolkit to scaffold a sample Loopback API application.

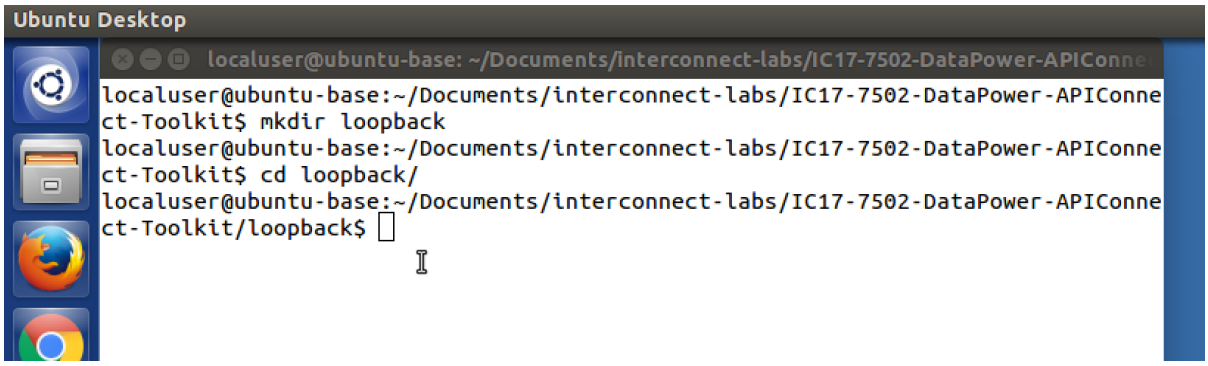
First open a new Terminal window



Create a new empty folder using the following commands:

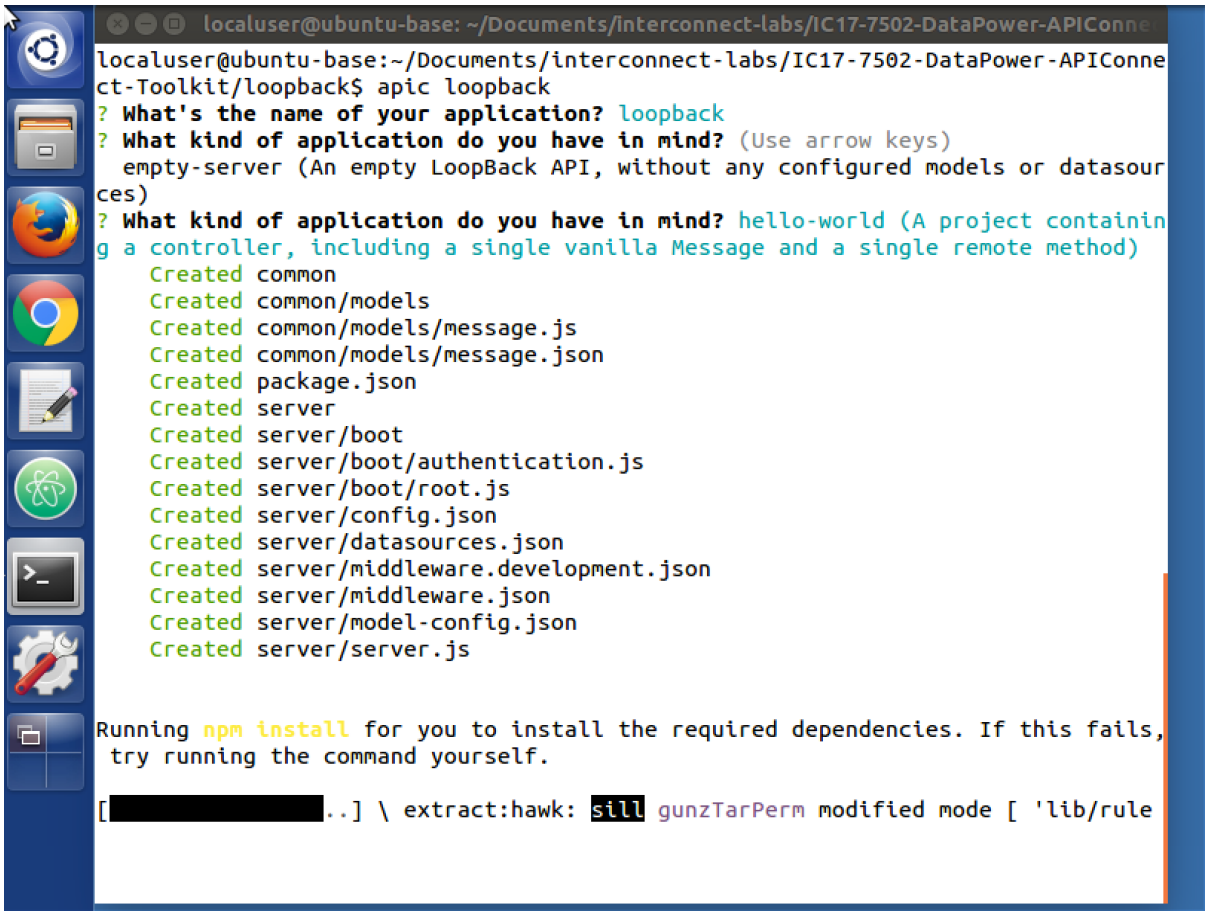
```
cd /home/localuser/Documents/interconnect-labs/IC17-7502-DataPower-
APIConnect-Toolkit/

mkdir loopback

cd loopback
```

Terminal window should look something like this:



Next, we will scaffold a sample loopback API application. From the loopback directory, execute the following command and select all the defaults

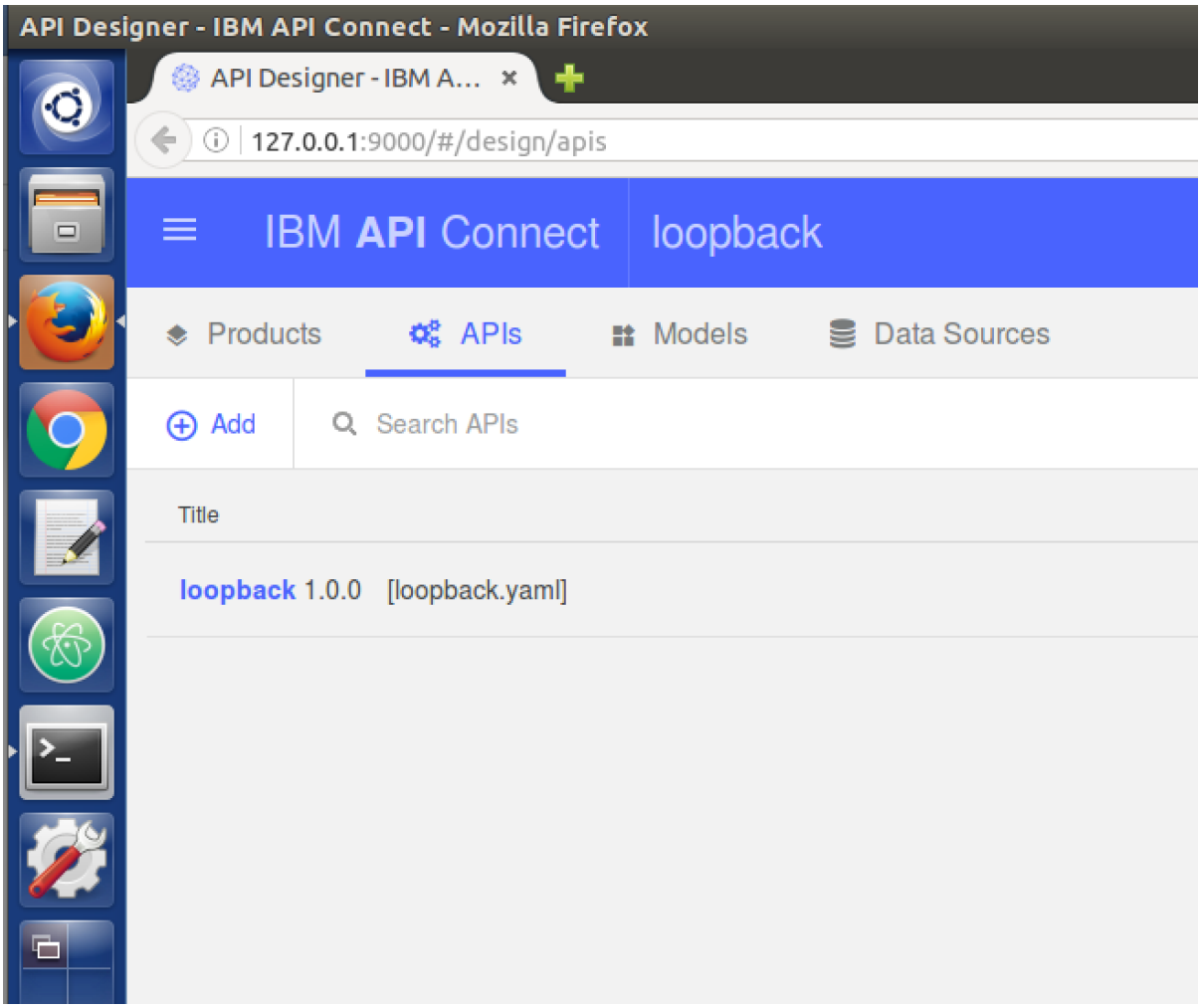
```
apic loopback
```



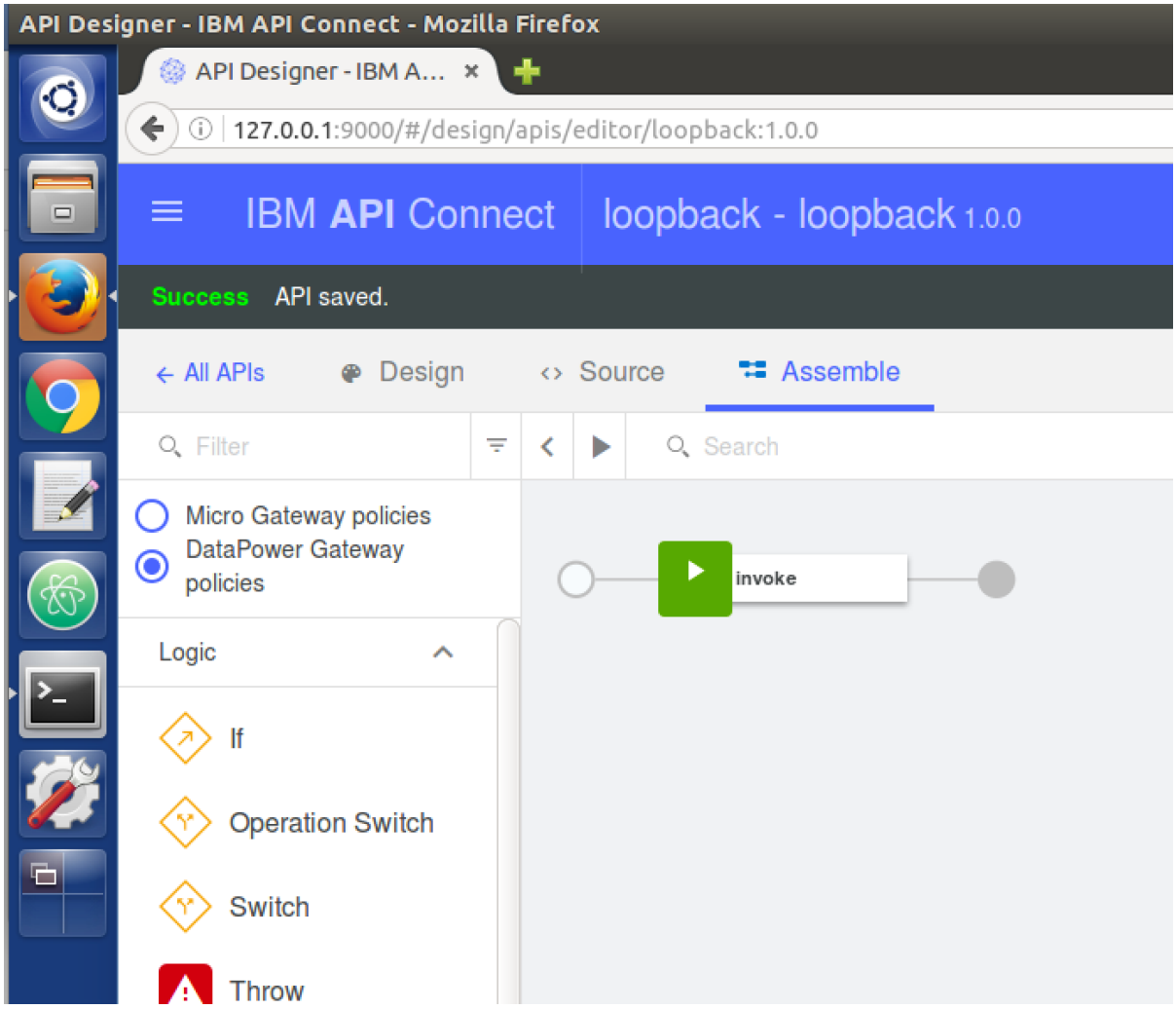
Next type

```
apic edit
```

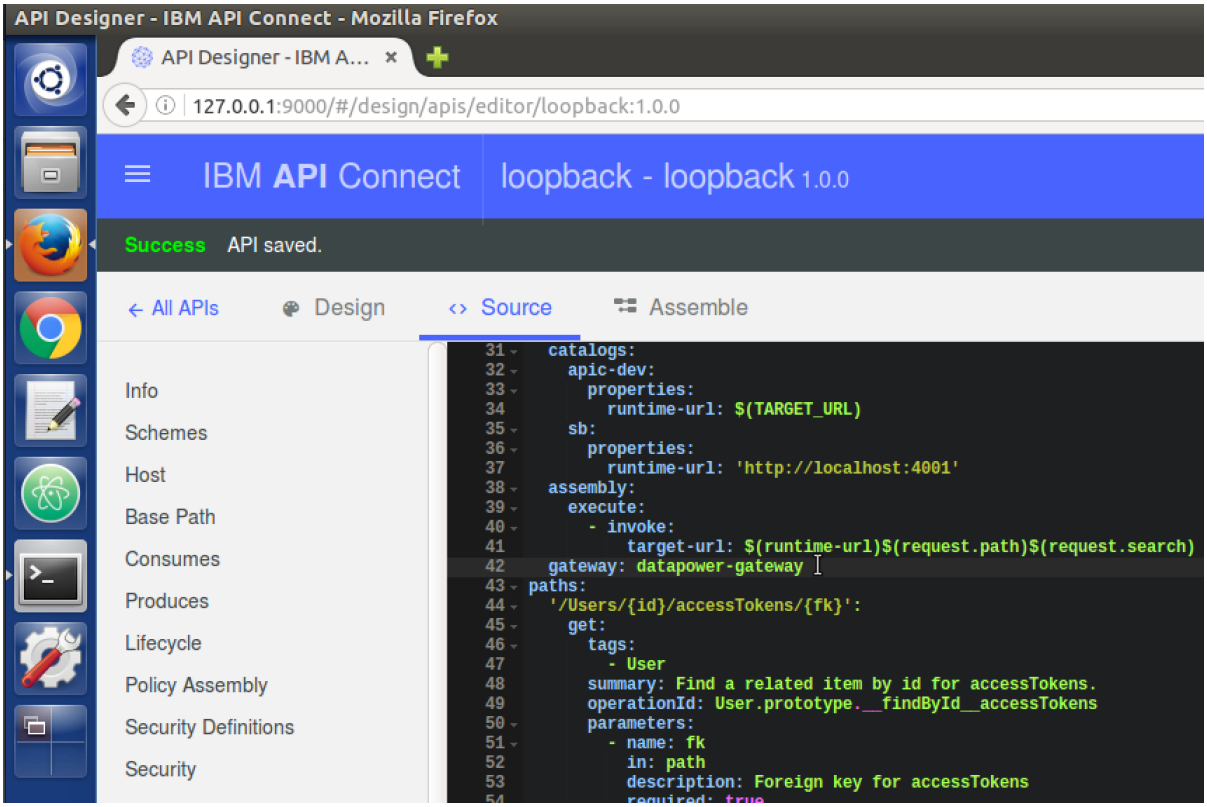
to launch the API Designer



select the loopback API, click the “Assembly” tab and select “DataPower Gateway policies” and click the save icon on the right. This will update the API definition file in the current project to specify DataPower as the default gateway type.

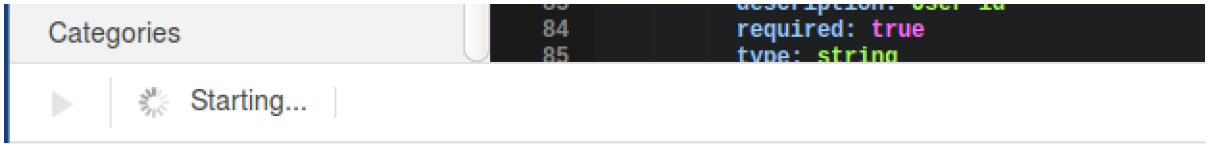

API Designer - IBM API Connect - Mozilla Firefox
API Designer - IBM A...
127.0.0.1:9000/#/design/apis/editor/loopback:1.0.0
IBM API Connect
loopback - loopback 1.0.0
Success API saved.
All APIs
Design
Source
Assemble
Filter
Search
Micro Gateway policies
DataPower Gateway policies
Logic
If
Operation Switch
Switch
Throw
invoke

you can also see in the “Source” view that the following “gateway: datapower-gateway” was added on line 42:

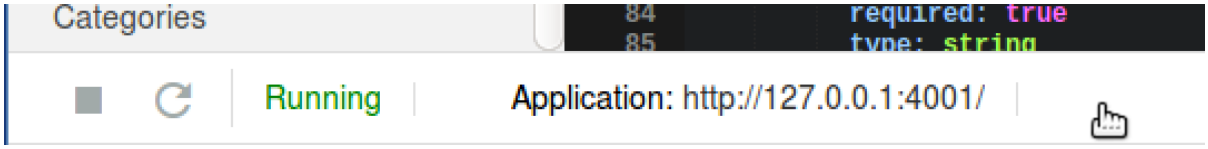


Now that we have specified DataPower as the gateway type, it’s time to start both the Loopback API application and the gateway.

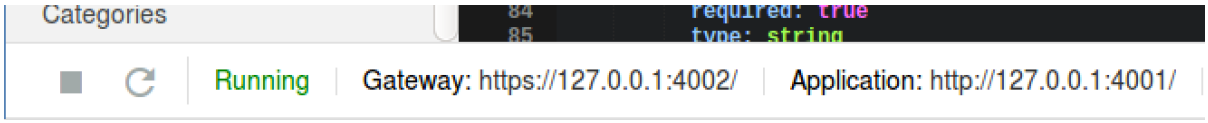
Press the play button and wait approximately 2 minutes to allow DataPower to start and retrieve the configuration.



the application will complete the start procedure before the gateway:

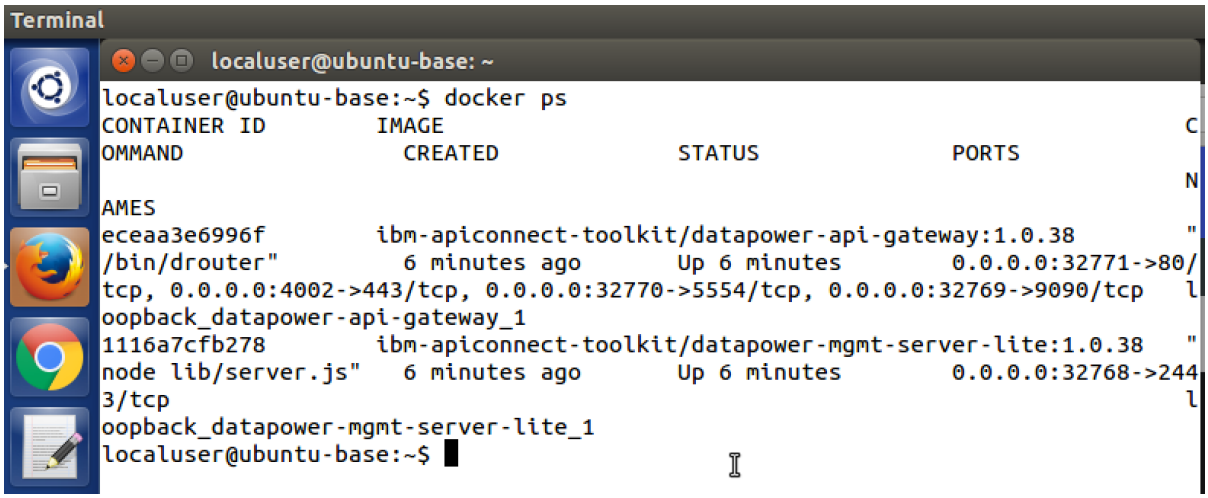


be patient and wait for the following (clicking stop, start or restart will only prolong the operation)

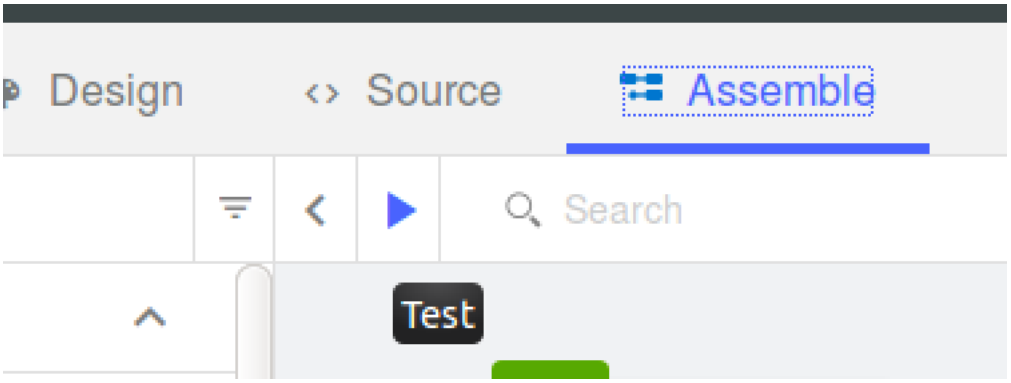


This indicates that the Gateway and Application are both running and listing on ports 4002 and 4001 respectively.

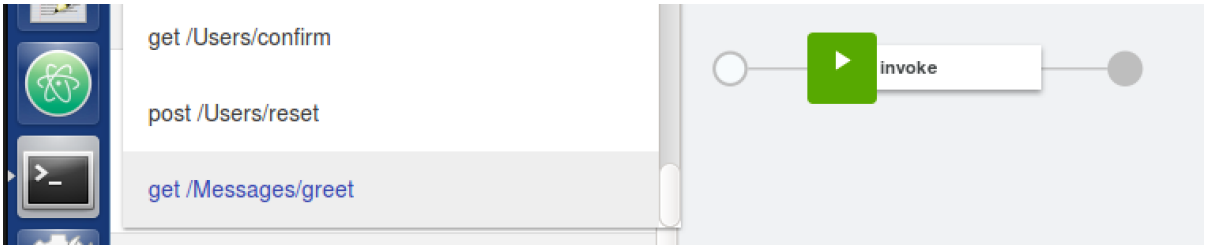
You can switch to a new Terminal screen and see that Docker is running the DataPower image and another image (datapower-mgmt-server-lite) to map the current project to DataPower configuration:



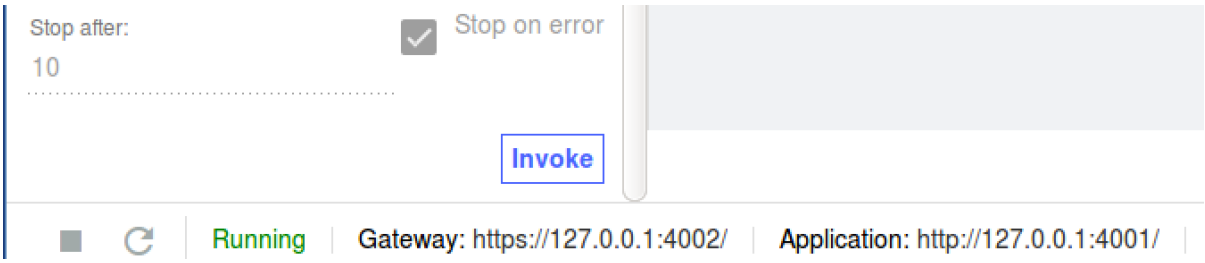
The next step is to test the running API using the Assembly Test Tool. Click on the “Assembly” tab and click top “play” button:



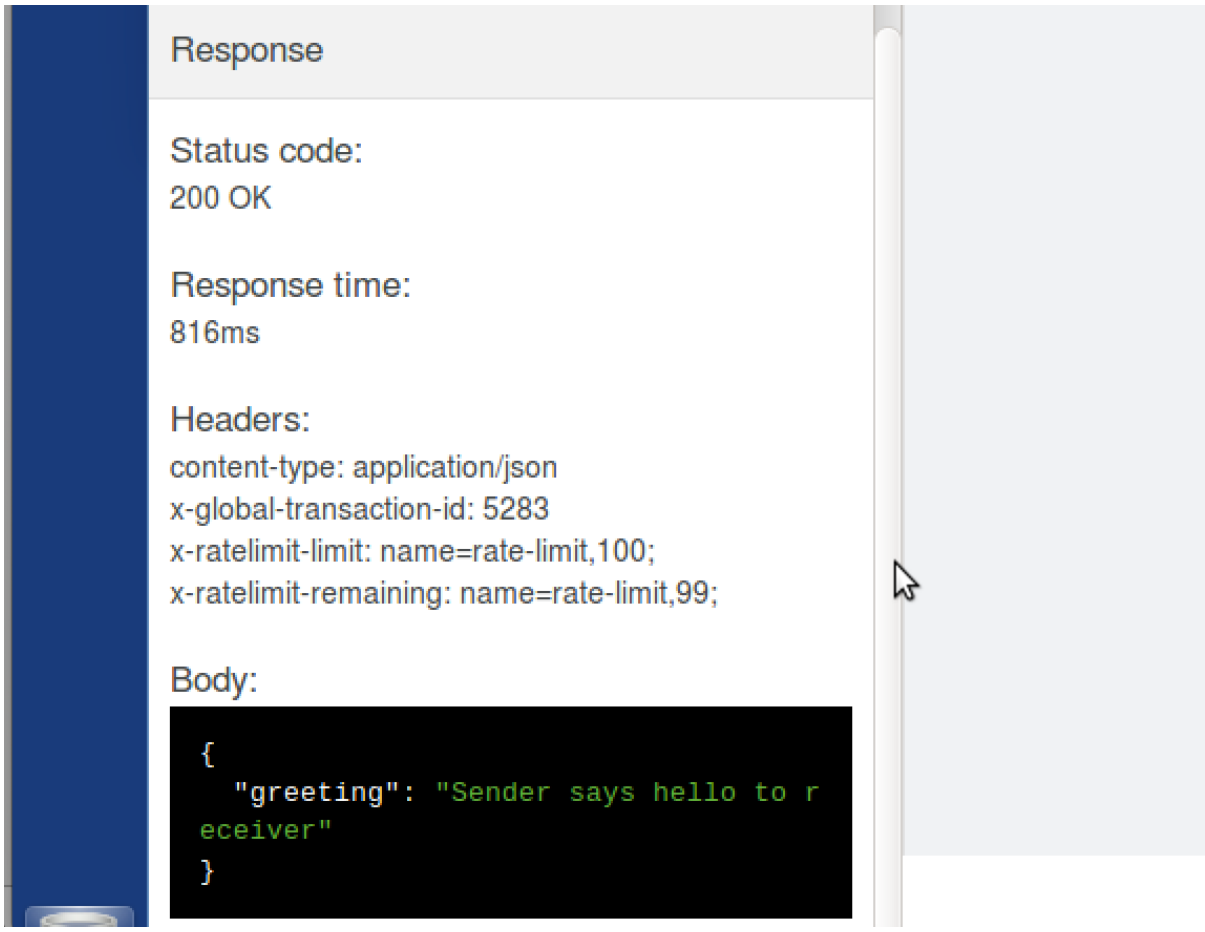
select the get /Messages/greet operation (located at the end of the operation list)



and scroll down to the bottom of the page to find the “Invoke” button (note: you might need to use both scrollbars, left and right to get this button to display, depending on the screen resolution)

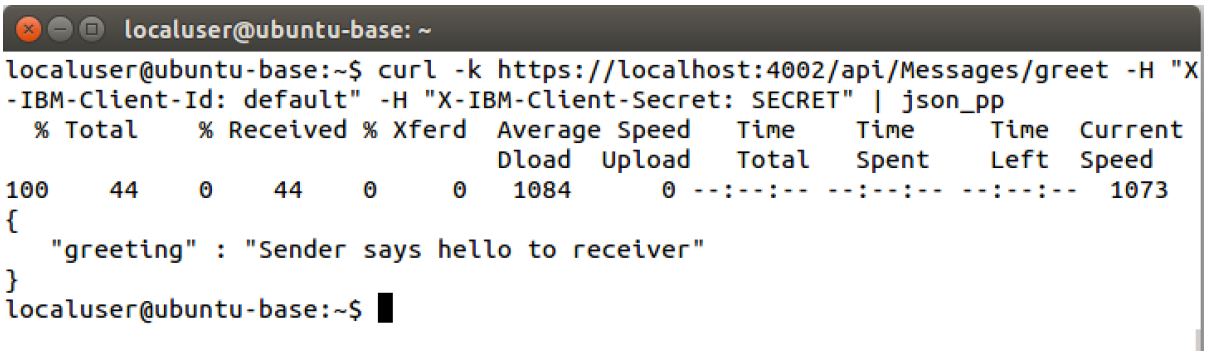


click “Invoke” to see the results:



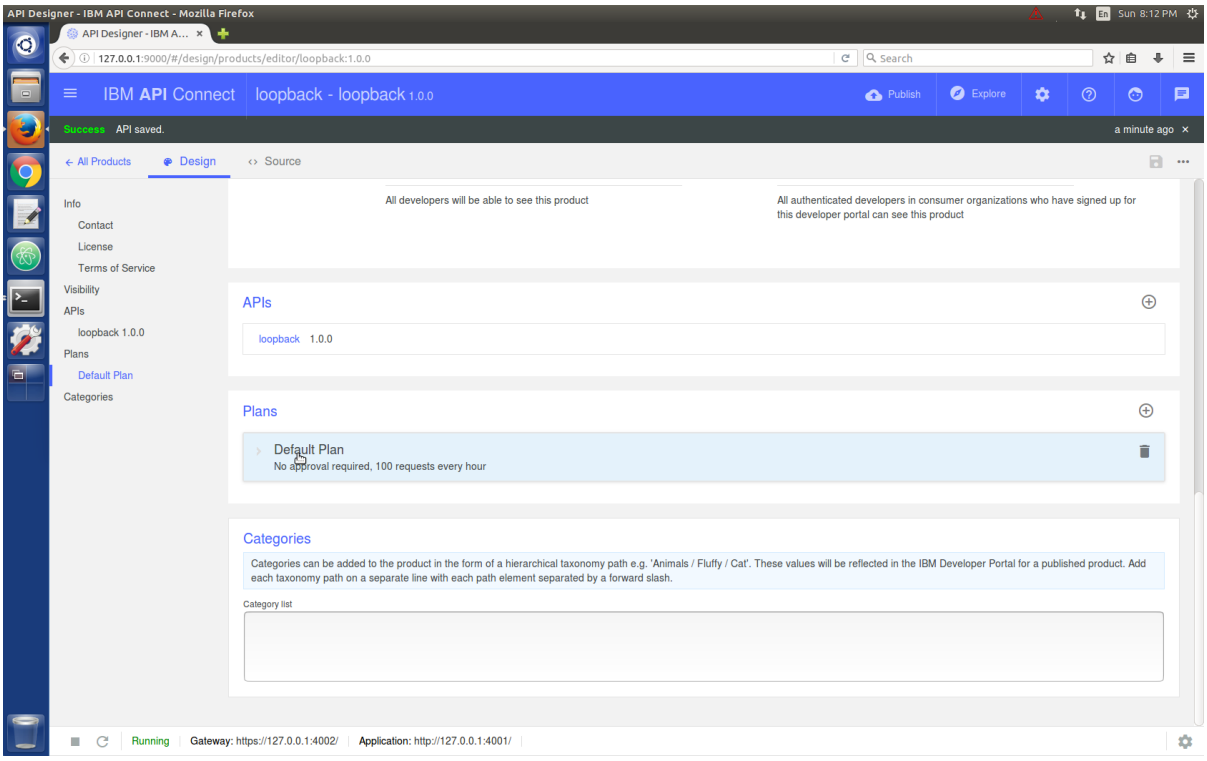
To test the rate-limit feature, its quicker to issue this request using curl. Open a new terminal window and issue the following command:

```
curl -k https://localhost:4002/api/Messages/greet -H "X-IBM-Client-Id:
default" -H "X-IBM-Client-Secret: SECRET" | json_pp
```

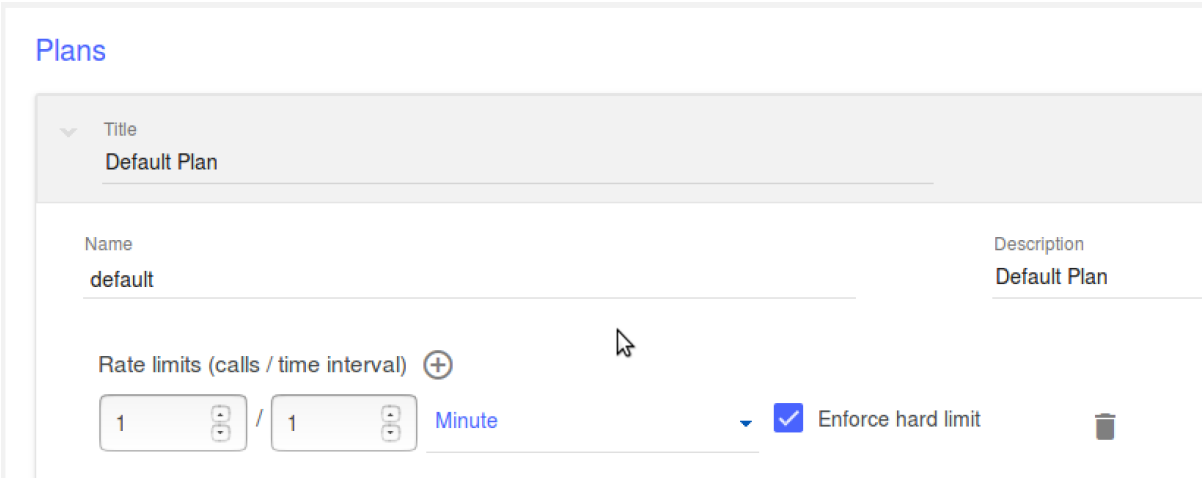


Ok now to the fun part…

Click the “All APIs” tab, click the “Products” tab and select the loopback product. Click the default plan.



Expand the “Default Plan”, select 1 request per minute, click “Enforce hard limit” and click the “Save” button on the top right of the screen.



Switch back to the previous Terminal window or create a new one and issue the previous curl command twice. You should observe the second request being refused with a 429 response:

```
localuser@ubuntu-base:~$ curl -k https://localhost:4002/api/Messages/greet -H "X
-IBM-Client-Id: default" -H "X-IBM-Client-Secret: SECRET" | json_pp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    44    0    44    0     0    688      0 --:--:-- --:--:-- --:--:--   698
{
   "greeting" : "Sender says hello to receiver"
}
localuser@ubuntu-base:~$ curl -k https://localhost:4002/api/Messages/greet -H "X
-IBM-Client-Id: default" -H "X-IBM-Client-Secret: SECRET" | json_pp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    96    0    96    0     0    354      0 --:--:-- --:--:-- --:--:--   355
{
   "httpCode" : "429",
   "moreInformation" : "Rate Limit exceeded",
   "httpMessage" : "Too Many Requests"
}
localuser@ubuntu-base:~$
```

This part of the lab will teach you to expose a REST API (JSON data) which calls a standard web service backend (SOAP). The exercise exemplifies a common use case: JSON -> SOAP and SOAP -> JSON conversion.

Lab 1 - REST to SOAP (and back again)

In this section, you'll set up an API acting like a back-end service. This service will take a SOAP request as input and extract certain fields from it returning a different response payload.

Click on "terminal" to open a new window. Make a new directory in your home directory: **mkdir lab1**.

This directory will be used to build the solution. The actual lab (and solution) is in the /home/localuser/interconnect-labs/IC17-7502-DataPower-Toolkit/labx directory.

You'll be using some of the files there.

Now let's change directory to lab1: **cd lab1.** Follow the directions below.

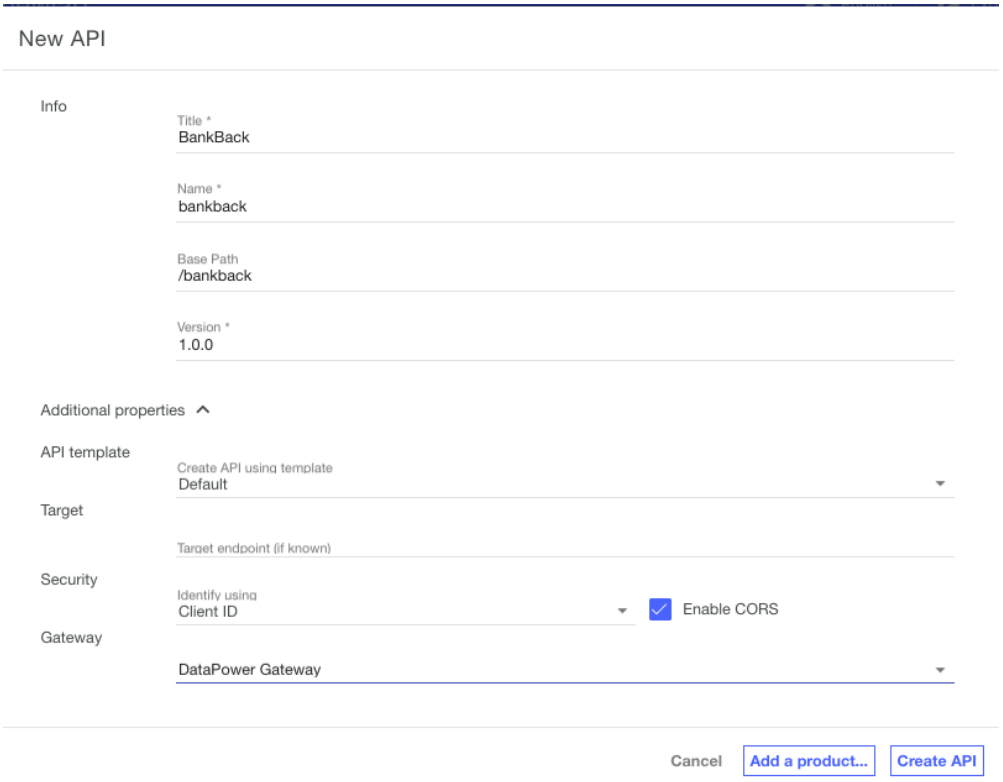
Step 1: Open API Designer

On the command line, type: **apic edit**, to start the API Designer. A browser window will open up on port 9000.

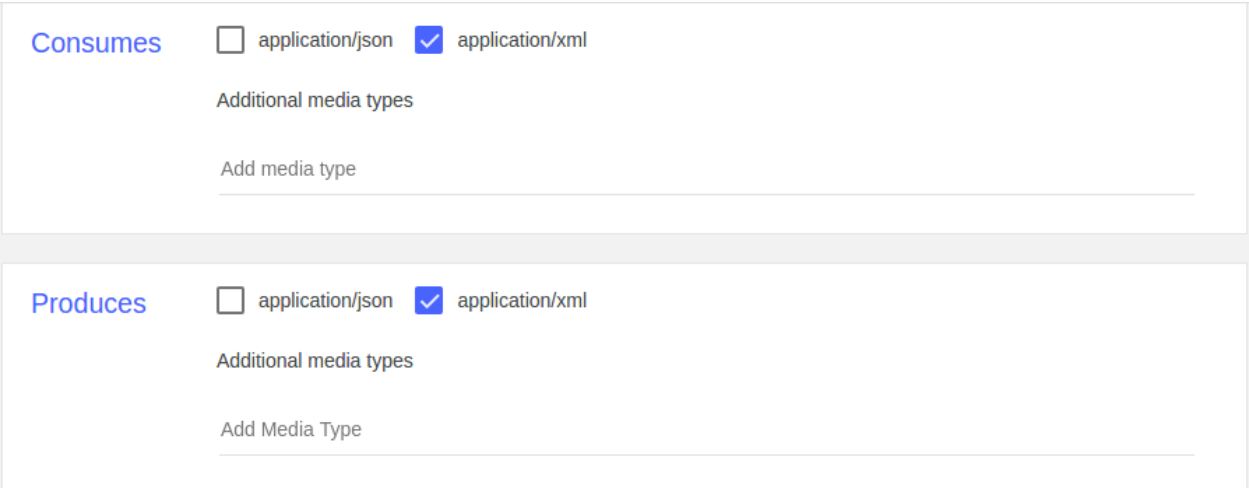
Step 2: Create new API BankBack

Create a new api in Drafts called BankBack. Click “+” to add a “New API”. Enter the title, and additional properties as depicted below. Setting the “Gateway” property will enable the local DataPower option we will be testing.

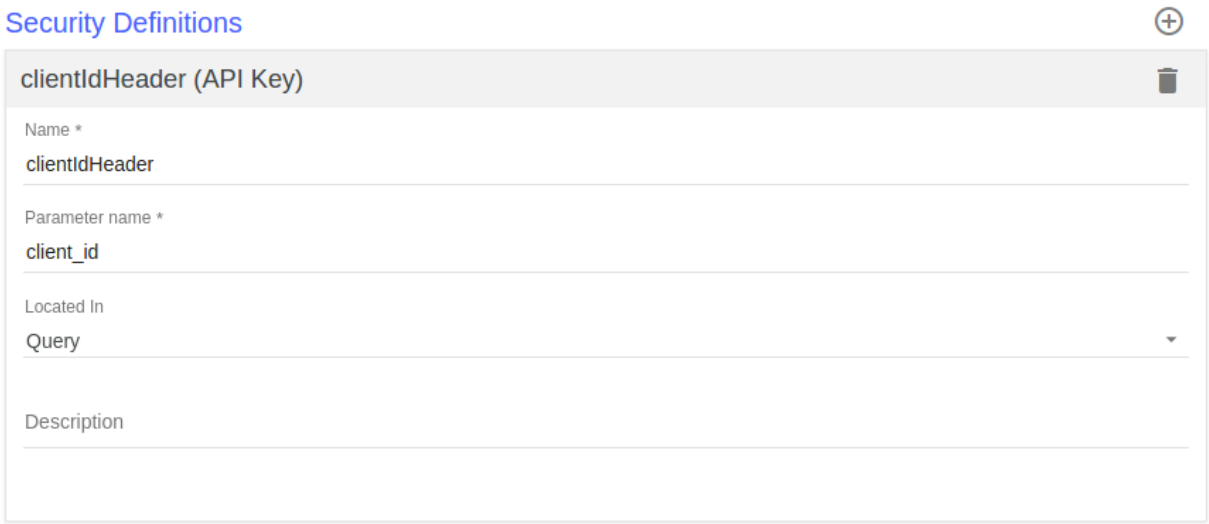
Before creating the api, go ahead and “Add a product”. Take the defaults. Unselect “Publishing”, since we will not be doing that yet. Then hit “Create API” to finish and create our API.



Set the API “Consumes/Produces” to application/xml as shown below:

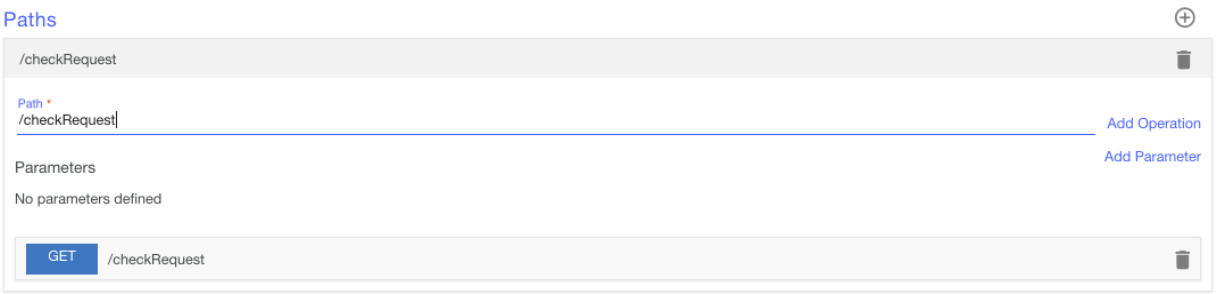


Change the default “Located In” field of ClientIDHeader (APIKey) from “Header” to “Query” in “Security Definitions”.



Step 3: Add POST operation

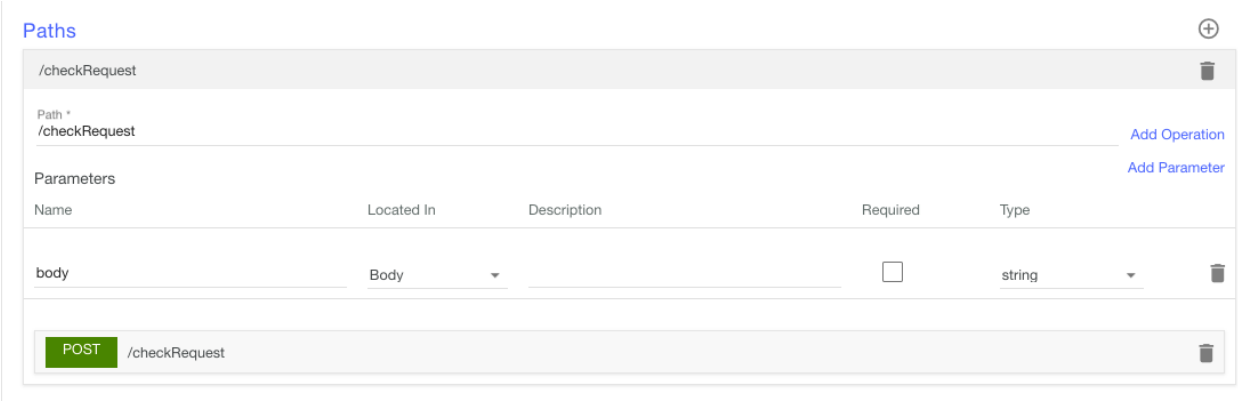
Add a new Path to the API. Click on “Paths” on the left menu. Click “+” to add a new path. Enter “/checkRequest”. The screen should look like this now.



However, we need a POST operation, since this is our back end and will be receiving a SOAP payload. Delete the “GET” by clicking on the trash can icon to the right of the “GET”.

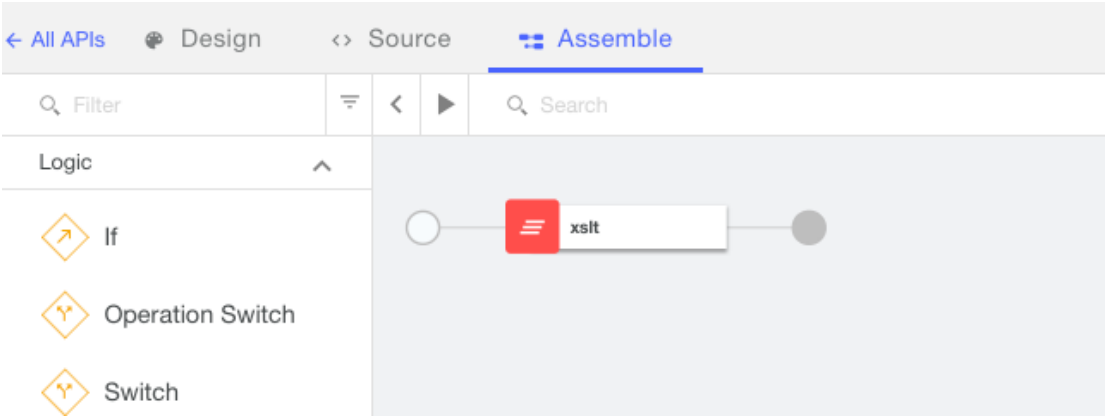
Add a new Operation (top left link under Paths). Select “POST”. Now we need to add the body of the request so we can test it locally (DataPower via test tool).

Click “Add Parameter”, add new parameter. For name enter “body” and “Located in”, select “Body”. The screen should now look like this:



Step 4: Add XSLT Policy

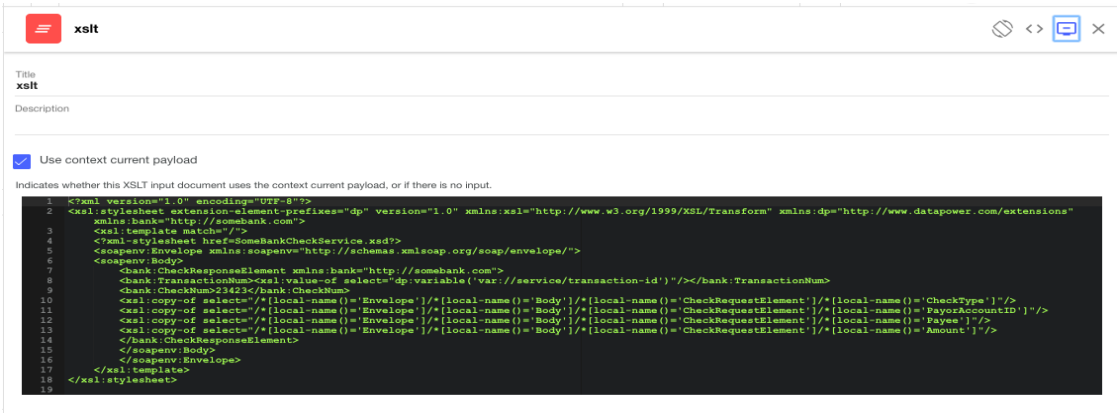
Head over to “Assemble” (top of the screen next to Design/Source) and delete the default invoke policy (click the trash can as you hover over the policy. Now we will add an XSLT policy by dragging the policy from the left palette onto the canvas. When dragging the policy don't let go until you place it on the yellowish square between the two dots. This policy will parse incoming SOAP and extract the data we want returned. The screen should now look like this:



if you click on the xslt policy, you'll get a empty screen. This is where normally you would enter (or copy) your logic. We are going to copy it from the lab files.

Step 5: Copy XSLT

Select the file under lab1 resources folder named: BankBack.xsl. Double click to edit, select all (ctrl-a) and copy (cntl-c). Go back to the empty editor open in API Designer and paste the xslt code (ctrl-v). The screen should look like this now:



Make sure the "Use context current payload" is checked off. Otherwise we won't see any payload.

We will be extracting CheckType, PayorAccountID, Payee, and Amount for our SOAP response.

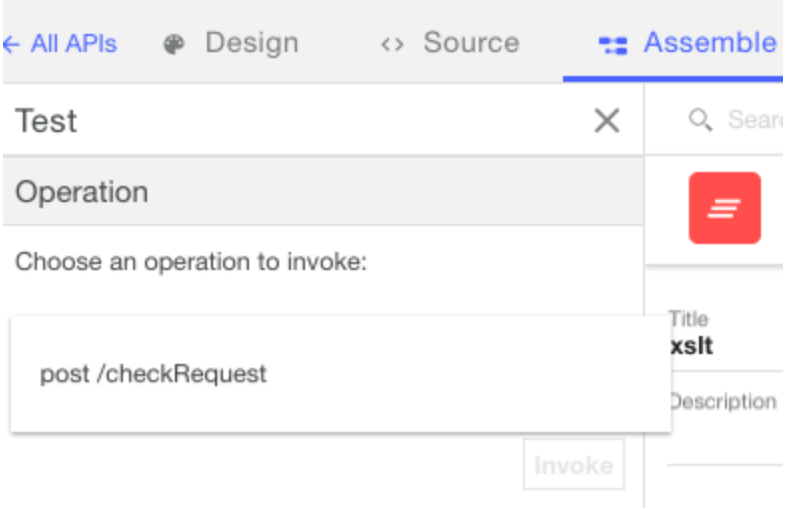
Step 6: Test

At the bottom of the screen you'll notice a "play" (>) button and the text: "Stopped"

Click on the arrow to start your DataPower gateway.

Click the Floppy disk icon on the top right of the screen to save the API. On the left hand side of the screen there is a "play" icon (>). This is the test button. Click it to open the test pane on the left.

Select operation "post /checkRequest".



We need to provide a "body" parameter to emulate a payload request coming into the API. One of the files provided is called SBCRequest.xml. Open that file with an editor and copy paste into here.

Test

body


```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/env
elope/">
    <soap:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
<wsse:UsernameToken>
<wsse:Username>fred</wsse:Username>
<wsse:Password>smith</wsse:Password>
</wsse:UsernameToken>
</wsse:Security>
</soap:Header>
    <soap:Body>
        <bank:CheckRequestElement
xmlns:bank="http://somebank.com">
        <bank:PayorAccountName>Cartoon
Studios</bank:PayorAccountName>

<bank:PayorAccountID>8458jf8757275234</bank:P
ayorAccountID>

<bank:CheckType>P</bank:CheckType>
            <bank:Payee>Elmer
Funn</bank:Payee>
            <bank:PayeeAddr>124 East Sunset
Drive; Sunset, AL 64846</bank:PayeeAddr>

<bank:PayeeRouteNo>03849032874908</bank:Pay
eeRouteNo>

<bank:Amount>999.00</bank:Amount>
        </bank:CheckRequestElement>
    </soap:Body>
</soap:Envelope>
```


Show schema | Generate

Repeat

Repeat the API invocation a set number of times, or

Then scroll down and press “Invoke”. If you get something like this:

Response

Status code:
-1

No response received. Causes include a lack of CORS support on the target server, the server being unavailable, or an untrusted certificate being encountered.

Clicking the link below will open the server in a new tab. If the browser displays a certificate issue, you may choose to accept it and return here to test again.
https://localhost:4001/bankback/checkRequest

This is because is the first time going to this address and the certificates are not trusted. Click on the link and accept the exception. Then click invoke again.

This time, you should get this instead:

Body:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlso
ap.org/soap/envelope/"
    xmlns:bank="http://somebank.com">
    <soapenv:Body>
        <bank:CheckResponseElement>
            <bank:TransactionNum>248581
14</bank:TransactionNum>
            <bank:CheckNum>23423</bank:
CheckNum>
            <bank:CheckType
                xmlns:soap="http://sche
mas.xmlsoap.org/soap/envelope/"
                xmlns:xsd="http://www.w
3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w
3.org/2001/XMLSchema-instance">P
            </bank:CheckType>
            <bank:PayorAccountID
                xmlns:soap="http://sche
mas.xmlsoap.org/soap/envelope/"
                xmlns:xsd="http://www.w
3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w
3.org/2001/XMLSchema-instance">8458jf87
57275234
            </bank:PayorAccountID>
            <bank:Payee
                xmlns:soap="http://sche
mas.xmlsoap.org/soap/envelope/"
                xmlns:xsd="http://www.w
3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w
3.org/2001/XMLSchema-instance">Elmer Fu
nn
            </bank:Payee>
            <bank:Amount
                xmlns:soap="http://sche
mas.xmlsoap.org/soap/envelope/"
                xmlns:xsd="http://www.w
3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w
3.org/2001/XMLSchema-instance">999.00
            </bank:Amount>
        </bank:CheckResponseElement>
    </soapenv:Body>
</soapenv:Envelope>
```

Debug

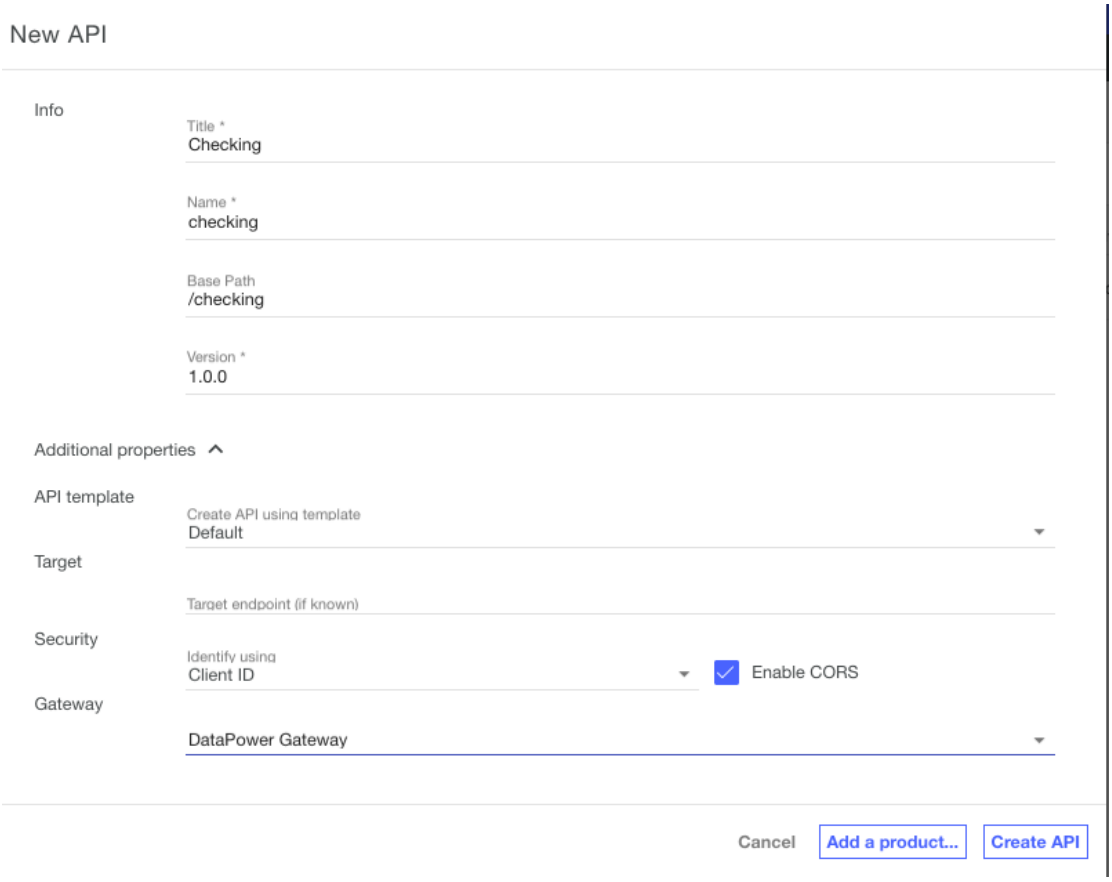
In this section, we will create another API which will be calling the back end we just created. This new rest API takes a JSON request converts it to SOAP, calls the BankBack service and converts the response back to JSON.

You'll need to use the terminal app to get a command line. In the left hand of the screen there is a tool bar. Click on "terminal" to open up a new window. In this lab we will use the actual directory that contains the yaml for the api. Change directory to /home/localuser/interconnect-labs/IC17-7502-DataPower-Toolkit/lab2.

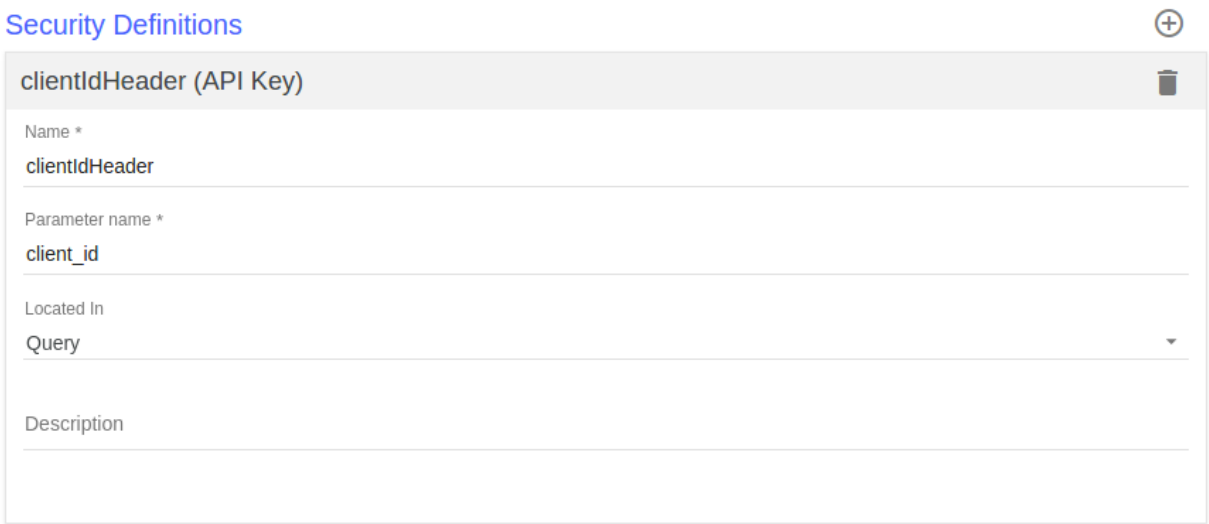
Step 1: Create new API - Checking

Click "all APIs" on the left of the API Designer page to go back to the list of APIs. Click "Add New API". The new API will be called "Checking". Do not add a product, since we are going to piggy back on the one we already created. Do change the "Gateway" to DataPower.

Click "Create API".

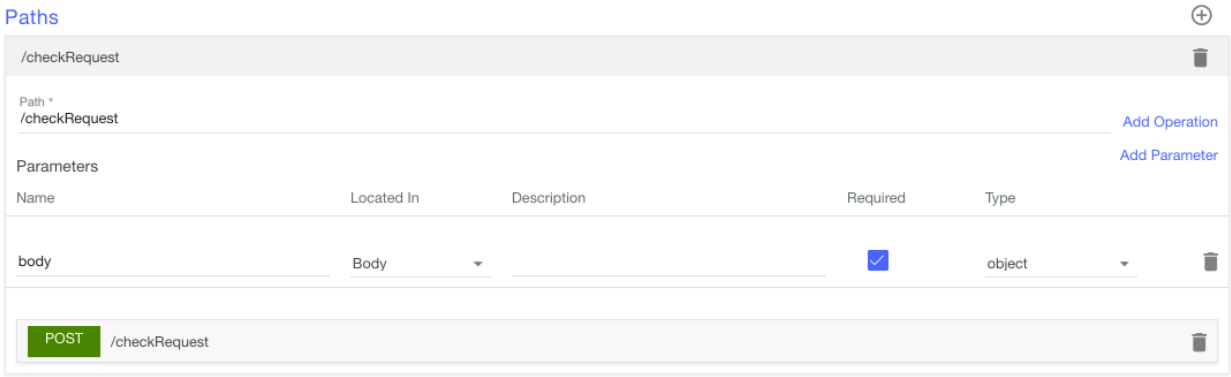


Change the default “Located In” field of clientIDHeader in “Security Definitions” from “Header” to “Query” as shown below. We will be passing it in the invoke to the BankBack service.



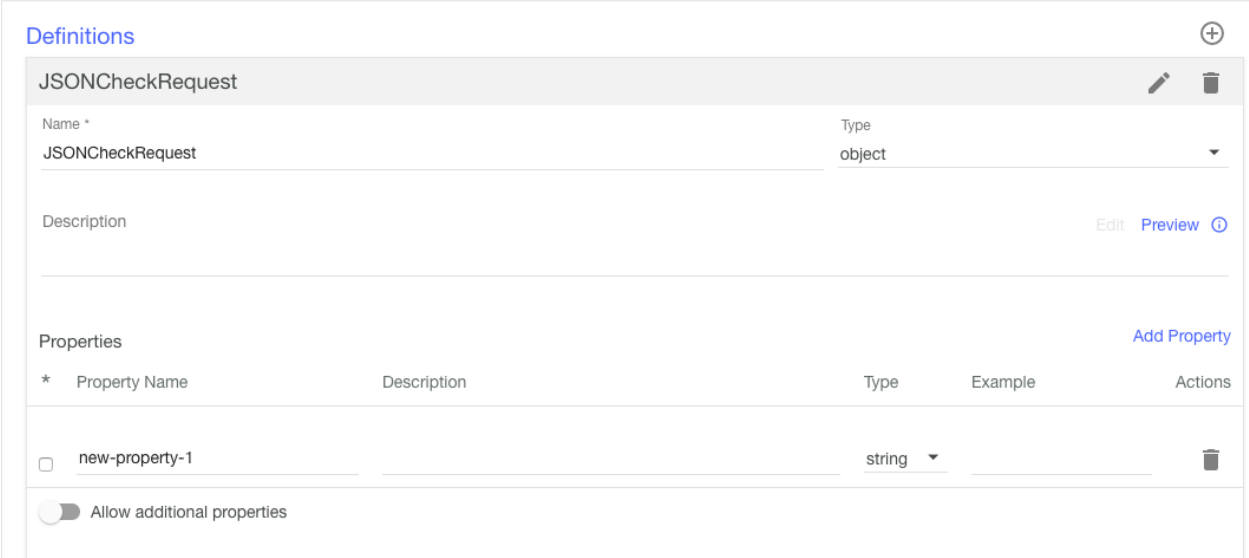
Step 2: Add POST /checkRequest

Add a new path by clicking “+” in Paths. Name the new path: “/checkingRequest”. Add a POST operation to the path just created. Add a “body” parameter. The screen should look like this:



Step 3: Add Definitions for request/response.

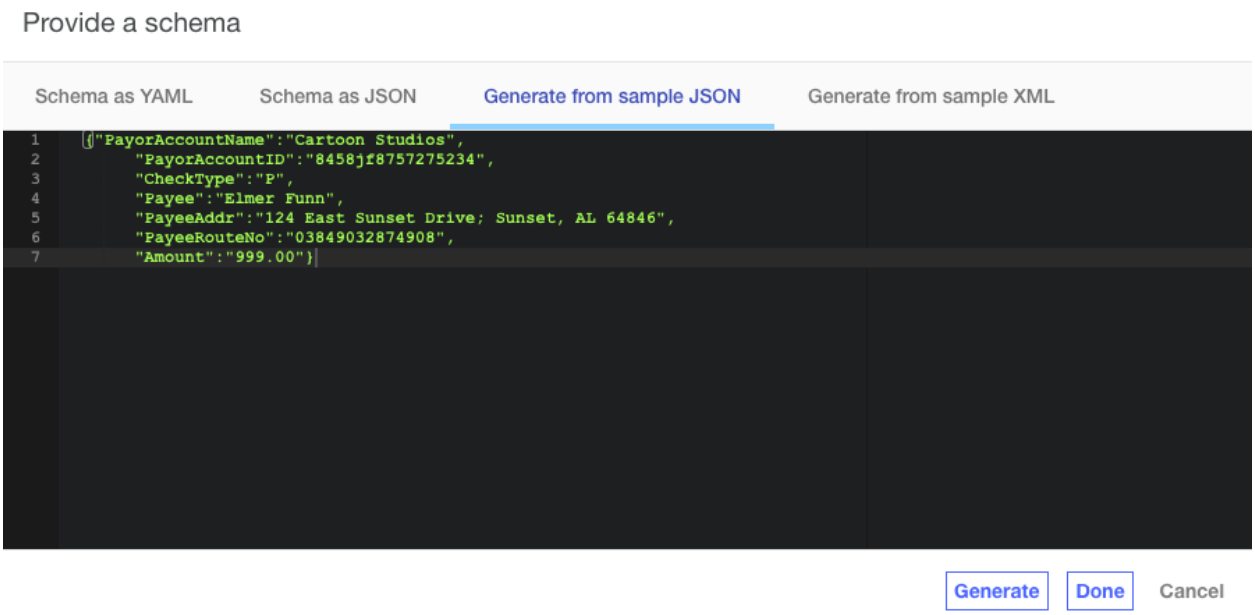
Click Definitions on the left hand pane. Click “+” to add a new Definition. Under “Name” enter “JSONCheckRequest”. Select “Type” as object.



We have a choice here: we can enter the attributes as properties one by one, or we can have API Designer do it for us. We will take the easy way. Open the “checkingRequest.json” file in the resources folder and copy it.

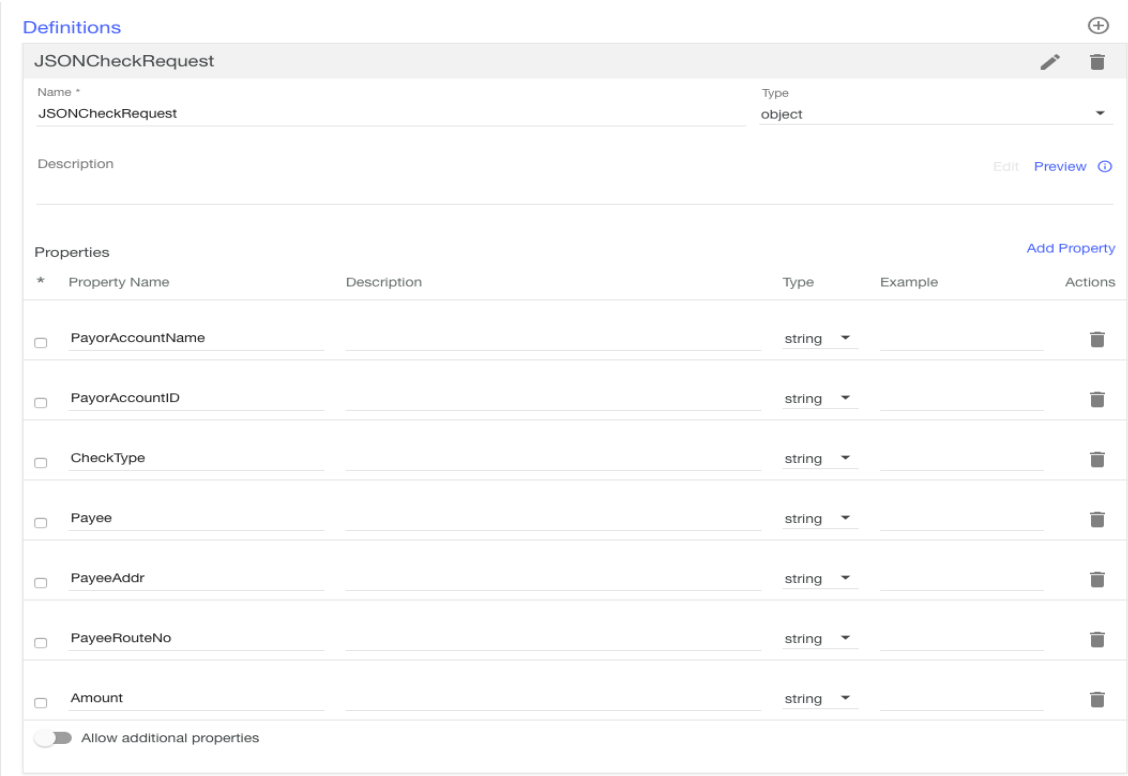
Click the “edit” icon (pencil) next to the trash can icon on the upper right corner of Definitions.

Select “Generate from sample JSON” tab and paste the JSON from the checkingRequest.json file into here.



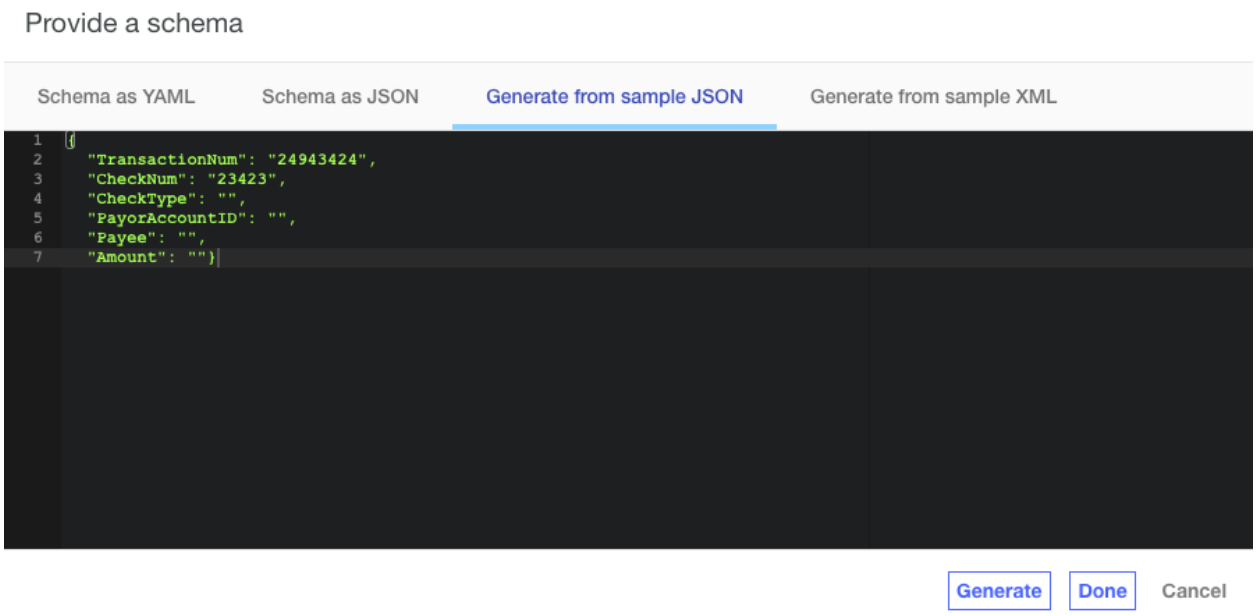
Click “Generate”. The screen changes to the first tab: “Schema as YAML”. Click “Done”.

Coming back to the “Definitions” screen, we see the properties defined.

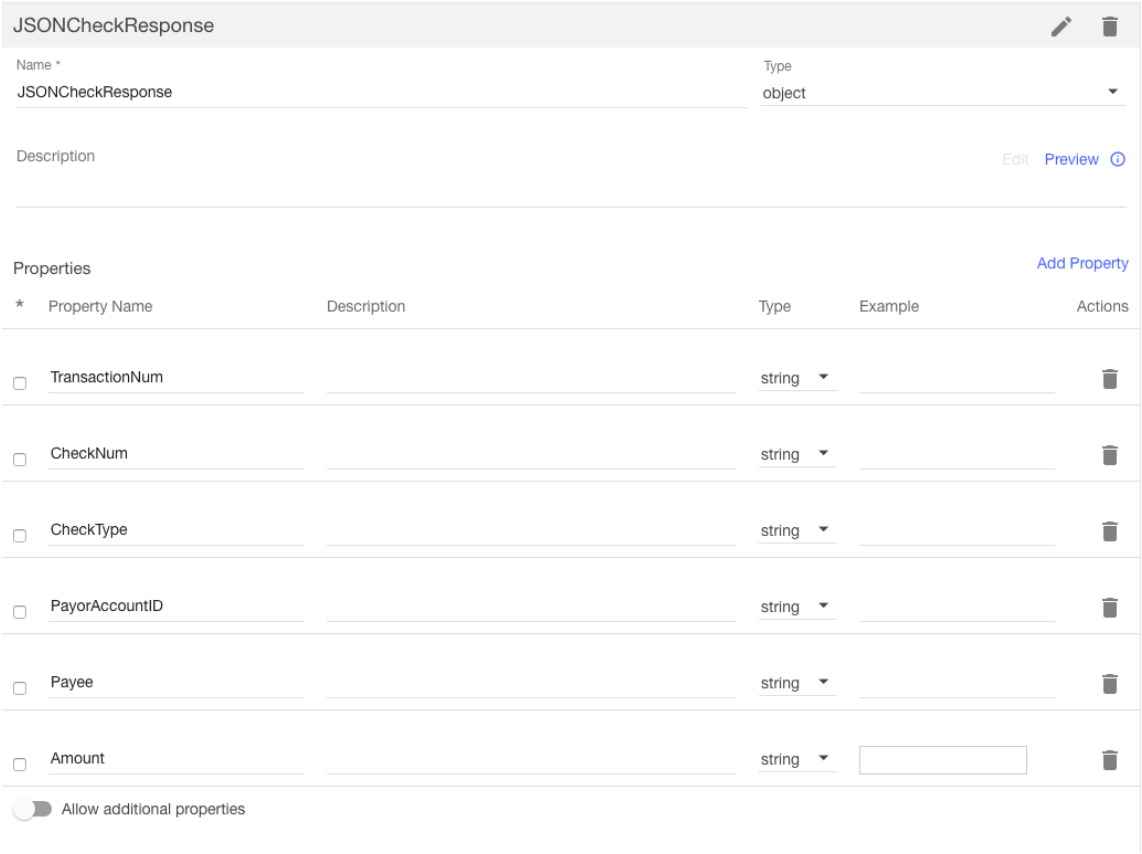


We need to do the same for the response.

Add a new Definition: JSONCheckResponse. Copy the contents of the checkResponse.json file into the schema editor and press “Generate”.

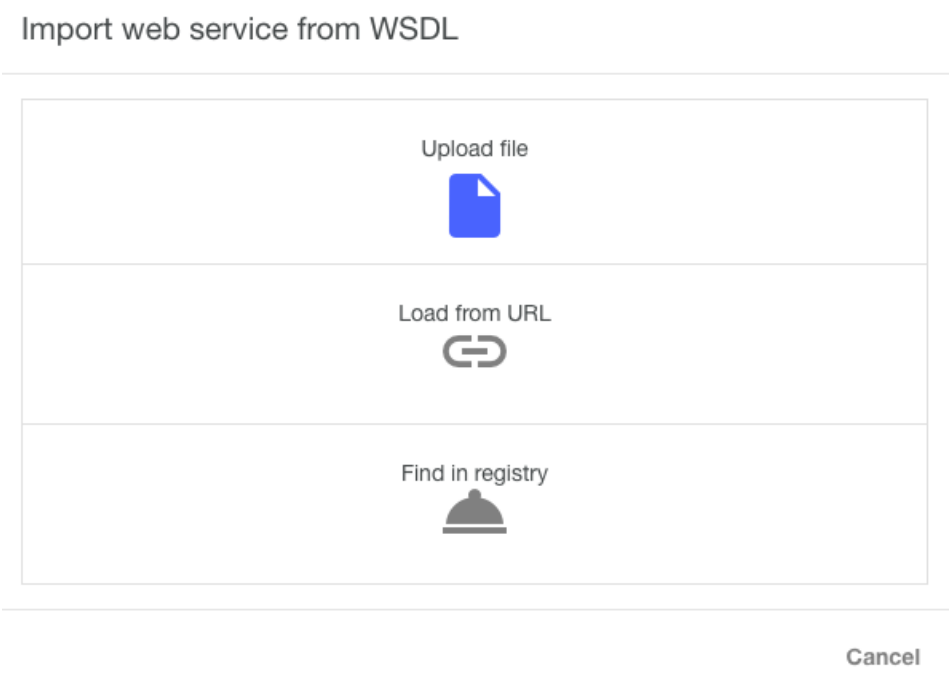


Your screen should look like this:

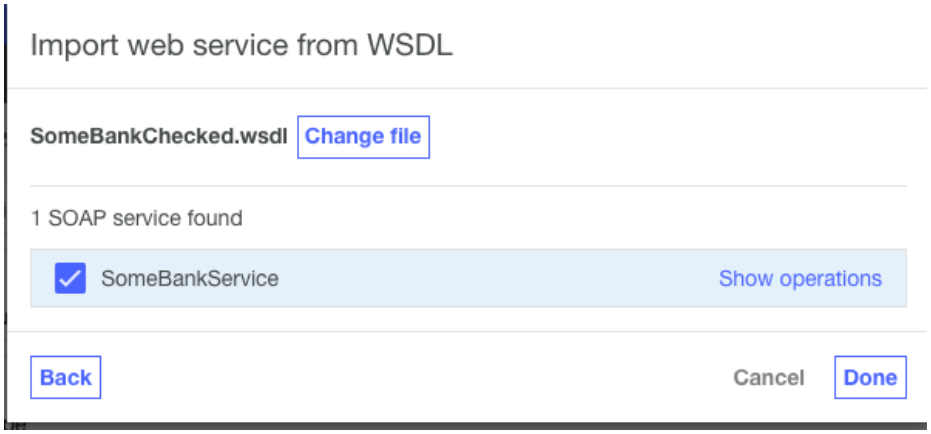


Step 4: Add Web Service Definitions for back end.

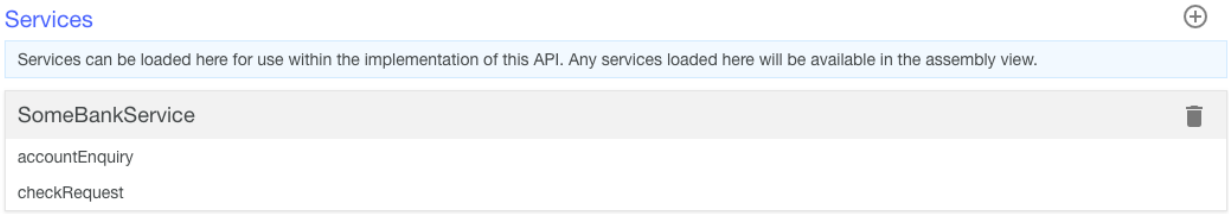
The easiest way to add a service definition (interface) is to import its WSDL. We are going to do just that. Head down to “Services” on the bottom of the left pane menu. Click “+”. We want to upload the file (from the resources folder): SomeBankChecked.wsdl.



Click the checkbox next to “SomeBankService”. Click Done.

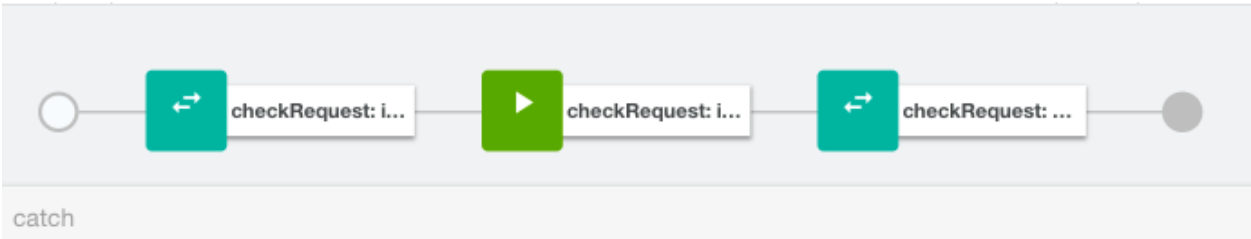


We now see two operations under “Services”.



Step 5: Add the services to the “Assemble” section

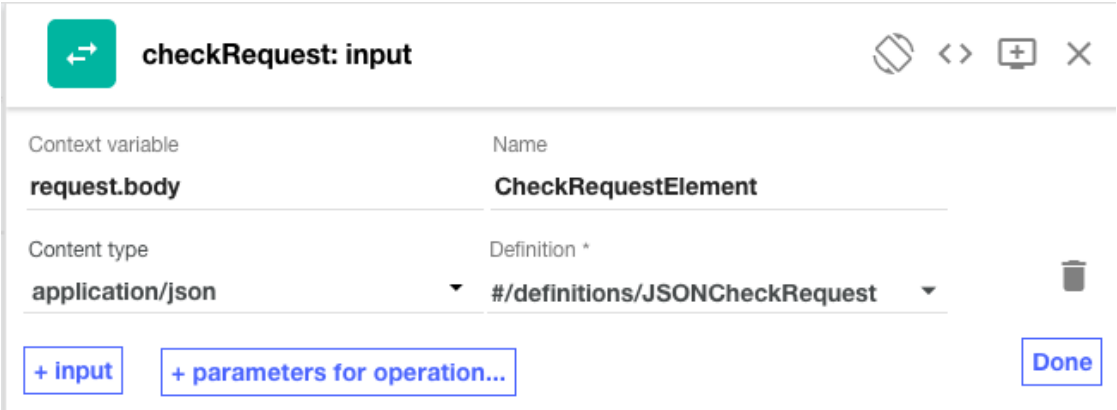
Delete the default invoke policy and drag the “checkRequest” Web Service Operation onto the canvas (bottom of left palette). It should look like this:



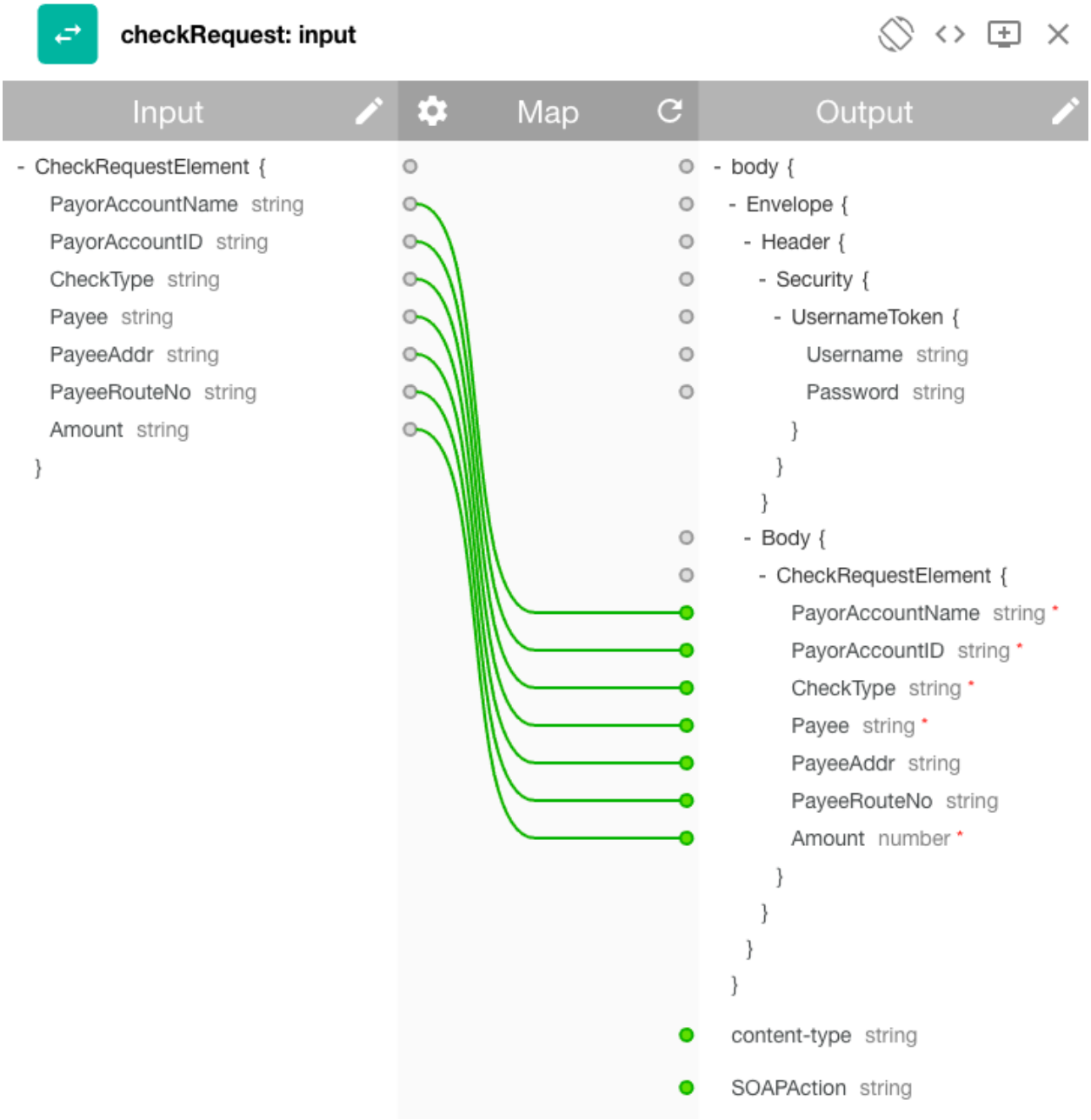
We now must edit the first map to take the input JSON request and convert it to SOAP and then modify the second map to take the response from BankBack service and convert it back to JSON.

Step 6: Edit first map

Click on the first map – the input should be empty. Click the edit icon (pencil) to add a new input definition. Click “+input” to add anew input definition. The input should look like this when you are done:



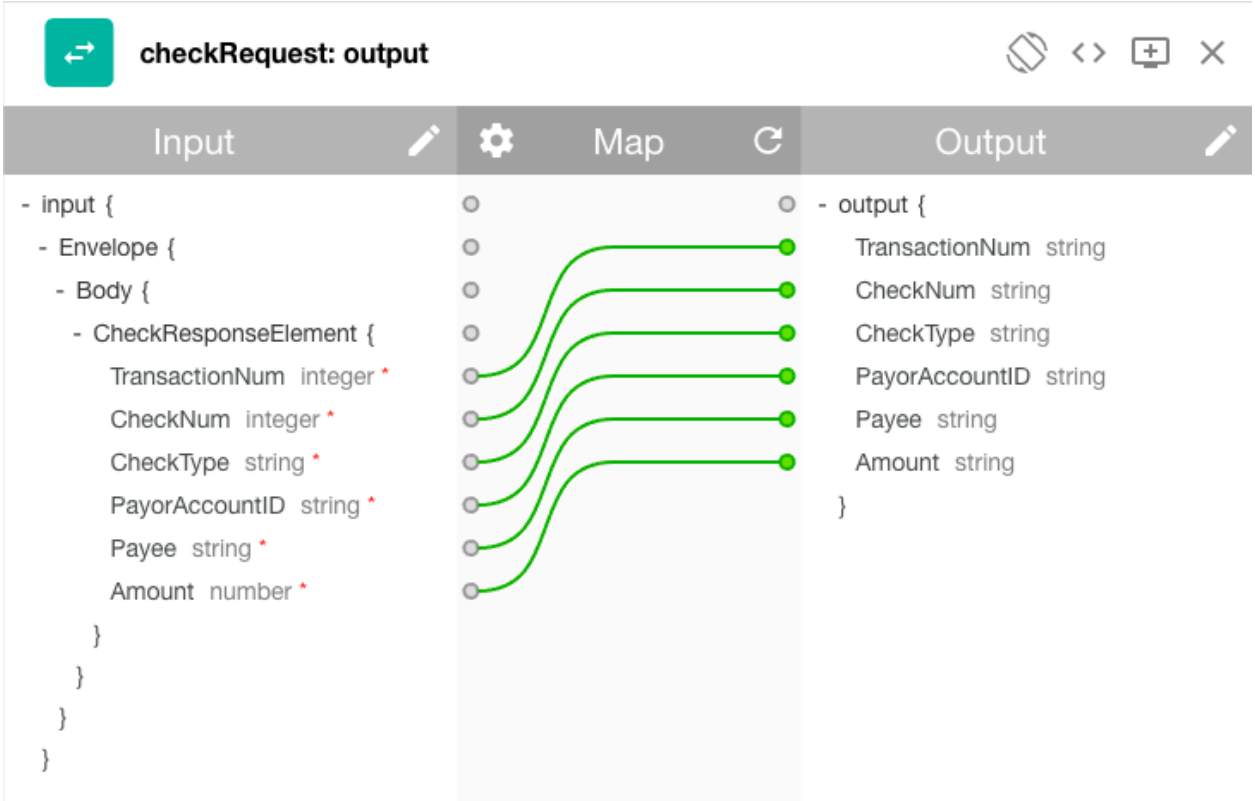
Then map the inputs to the outputs by clicking on the Input node (PayorAccountName) and dragging it to the corresponding Output node. The screen should look like this when done.


checkRequest: input
Input
Map
Output
- CheckRequestElement {
PayorAccountName string
PayorAccountID string
CheckType string
Payee string
PayeeAddr string
PayeeRouteNo string
Amount string
}
- body {
- Envelope {
- Header {
- Security {
- UsernameToken {
Username string
Password string
}
}
}
- Body {
- CheckRequestElement {
PayorAccountName string *
PayorAccountID string *
CheckType string *
Payee string *
PayeeAddr string
PayeeRouteNo string
Amount number *
}
}
}
}
content-type string
SOAPAction string

Step 7: Edit the 2nd map

The Output map should be empty. This is the response we want to return. Click the edit (pencil) icon to create a new Output definition. We will be using JSONCheckResponse.

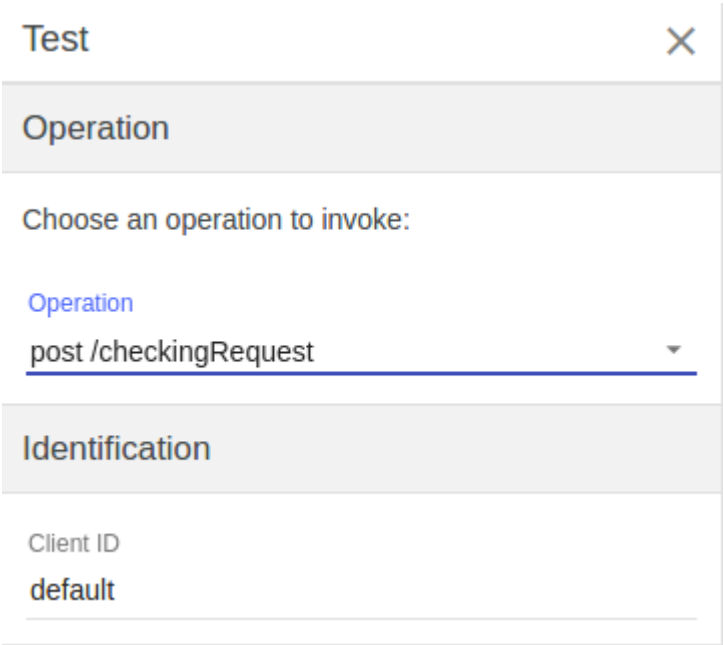
When Done we must map the Input nodes to the Output nodes.



Step 8: Modify Invoke to point to the correct URL.

We need to pass the client_id to the BankBack service as a query parameter. The easiest way to get that is to go into the Test tool and see what the client_id for the test tool is. For the local gateway it is always: default.

Click the “play” (>) button to open the test tool pane. Select the POST /checkRequest operation and the client_id shows up.



We need to copy this so we can add it to the end of the routing url in the invoke policy.

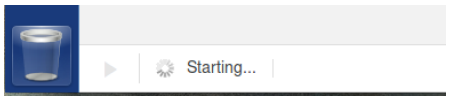
Click on the checkRequest invoke to open the pane. Change the URL field to:

Error! Hyperlink reference not valid.default

Where the client_id is the client_id copied from the test tool. Click the save icon.

Step 8: Test.

If the gateway is not running, click on the start icon at the bottom left of the screen to start it now. It should change to:

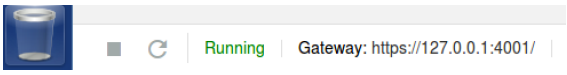


***** Please READ ******

It will take a few minutes after you click on the start icon and actually seeing the status change.

DO NOT hit the start again since it will act as the stop button and will cancel the start. So you'll have to wait until it stops and try it again.

At the end the status should look like:



Open the test tool again and select the checkRequest operation. Paste the JSON sample request from the resources folder (checkingRequest.json) into the "body" parameter on the test pane.

Parameters

Content-Type

application/json

Accept

application/json

body *

{"PayorAccountName":"Cartoon Studios",
"PayorAccountID":"8458jf8757275234",
"CheckType":"P",
"Payee":"Elmer Funn",
"PayeeAddr":"124 East Sunset Drive; Sunset, AL 64846",
"PayeeRouteNo":"03849032874908",
"Amount":"999.00"}

Click “Invoke”. You should see the following screen:

Response

Status code:
200 OK

Response time:
4590ms

Headers:
apim-debug-trans-id: 58a3672ce4b0e13332d303d4-6515d49e-9613-485b-a6c3-d1f032fb9e65
x-global-transaction-id: 25374594
content-type: application/json

Body:


```
{
  "TransactionNum": "25282896",
  "CheckNum": "23423",
  "CheckType": "P",
  "PayorAccountID": "8458jf8757275234",
  "Payee": "Elmer Funn",
  "Amount": "999.00"
}
```


Click on the “stop” (square) at the bottom of the screen to stop the gateway. Exit the browser and

This concludes lab 1.

Lab 2 – creating a secure API with JWT

Description

This lab will generate and validate a JWT token using Bluemix.

Change directory to /home/localuser/interconnect-labs/IC17-7502-DataPower-Toolkit/lab2 in a terminal window.

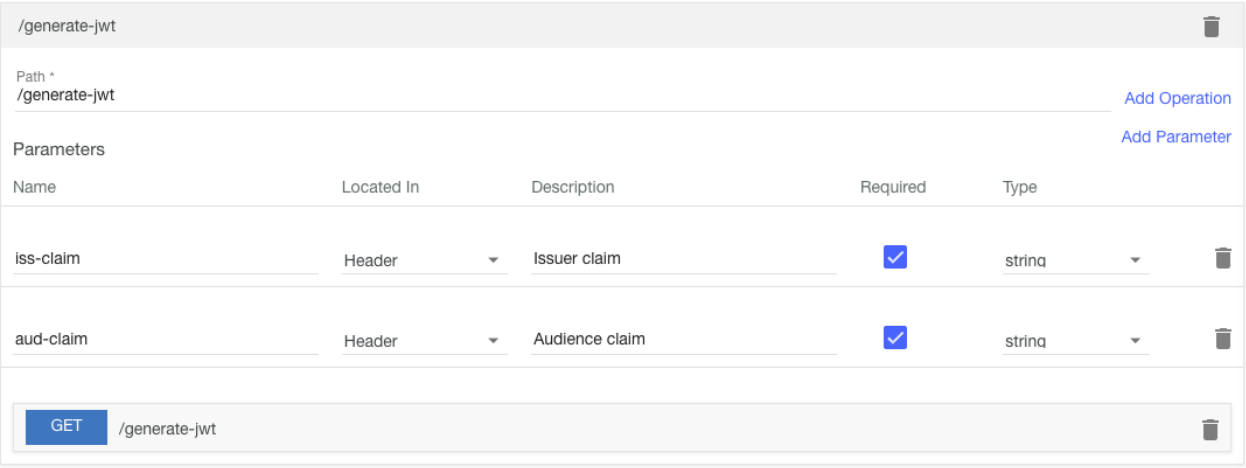
Step 1: import JWTAPI.yaml from the lab2 folder

On the desktop open a command prompt and navigate to the lab2 folder. In this folder you'll see the JWTAPI.yaml file.

Start the API Designer by typing: `apic edit`.

Once API Designer starts you'll see the api already loaded in the API window. Click on the api to display the API editor page. You'll notice that it has 2 paths: /validate-jwt and /generate-jwt.

Click on generate-jwt. It should look like this:



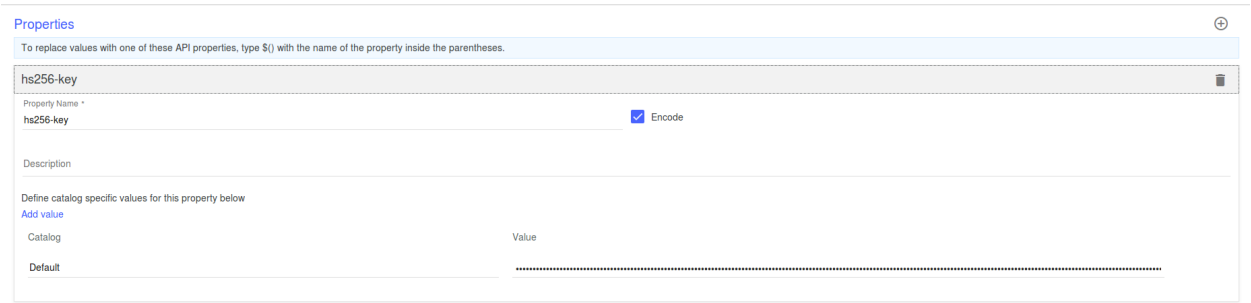
It has 2 Parameters: iss-claim (issuer) and aud-claim (audience). These are passed in in the header to generate a claim.

Step 2: Assemble

The logic here is pretty straight forward. A property variable was created to set the values of the key used to generate the JWT token. The property is encoded so that it will not be visible. The variable is hs256-key and its value is:

{ "alg": "HS256", "kty": "oct", "use": "sig", "k": "o5yErLaE-dbgVpSw65Rq57OA9dHyaF66Q_Et5azPa-XUjbyP0w9iRWhR4kru09aFfQLXeIODIN4uhjElYKXt8n76jt0Pjkd2pqk4t9abRF6tnL19GV4pflfL6uvVKkP4weOh39tqHt4TmkBgF2P-gFhgssZpjwq6l82fz3dUhQ2nkzoLA_CnyDGLZLd7SZ1yv73uzfE2Ot813zmig8KTMEMWVcWSDvy61F06vs_6LURcq_IEEevUiubBxG5S2akNnWigfpbhWYjMI5M22FOCpdcDBt4L7K1-yHt95Siz0QUb0MNIT_X8F76wH7_A37GpKKJGqeaiNWmHkgWdE8QWDQ", "kid": "hs256-key" }

This key was created using an online JWT key generator. There are other ways to create these keys.



On the Assemble tab, the switch statement determines the flow of the api based on either a HTTP protocol/path combination or a condition.

If the incoming request is for /generate-jwt, then the flow would do a jwt-generate policy to create the JWT token. If the incoming request is for /validate-jwt, it checks to make sure that the authorization request header has a “Bearer” token. Then it calls the jwt-validate policy. Otherwise, it sets the message.status.code to “302” and the message.status.reason to the value of the error which will be sent back to the consumer. Additionally the message.headers.Location is also entered to supply the location for the redirect. The other GWS policies are there to provide visual feedback of what has occurred during the flow.

Generate-JWT uses the hs-256-key so encrypt the token.

jwt-generate

Title

jwt-generate

Description

JSON Web Token (JWT)

generated-jwt

Runtime variable in which to place the generated JWT. If not set, the JWT is placed in the Authorization Header as a Bearer token.

☐ JWT ID Claim

Indicates whether a JWT ID (jti) claim should be added to the JWT. If selected, the jti claim value will be a UUID.

Issuer Claim

request.headers.iss-claim

Runtime variable from which the Issuer (iss) claim string can be retrieved. This claim represents the Principal that issued the JWT.

Subject Claim

Runtime variable from which the Subject (sub) claim string can be retrieved.

Audience Claim

request.headers.aud-claim

Runtime variable from which the Audience (aud) claim string can be retrieved. Multiple variables are set via a comma-separated string.

Validity Period

36000

The length of time (in seconds), that is added to the current date and time, in which the JWT is considered valid.

Private Claims

Runtime variable from which a valid set of JSON claims can be retrieved.

Sign JWK variable name

hs256-key

Runtime variable containing the JWK to use to sign the JWT.

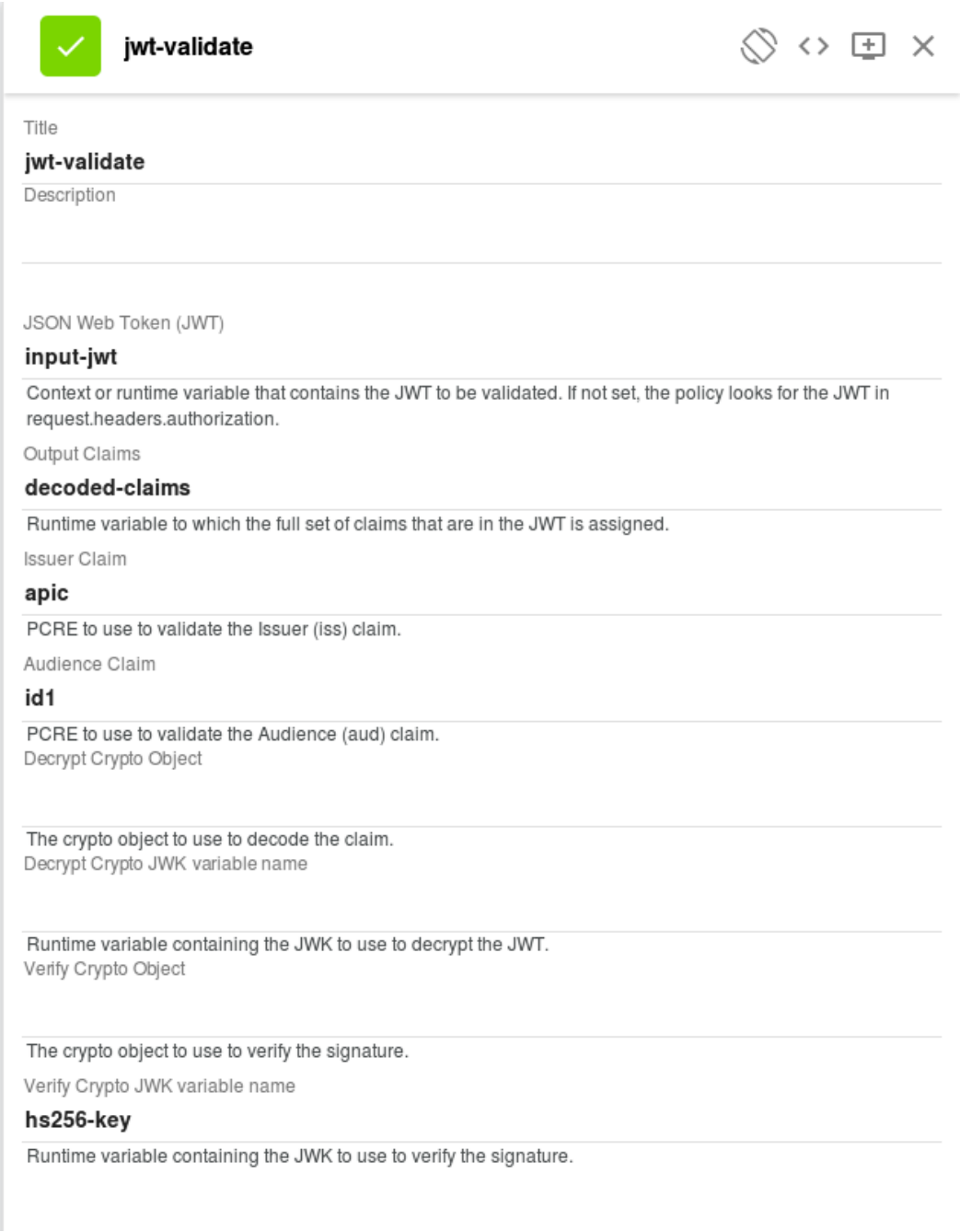
Cryptographic Algorithm *

HS256

The iss-claim and aud-claim is passed in as well.

Validate-JWT

The validate policy validates against a specific issuer and audience claim (apic/id1). This is done via the following configuration:



Notice that both generate and validate share the same hs256-key.

Step 3: Test

If the gateway is not running, press the run button at the bottom of the screen.

In order to test this locally just click on the test tool icon (>) on the screen. Select the /generate-jwt operation and enter: APIC for iss-claim and id1 for aud-claim.

The result is the JWT token as shown below:

Test

application/json

Accept

application/xml

Issuer claim

iss-claim *

apic

Generate

Audience claim

aud-claim *

id1

Generate

Repeat

Repeat the API invocation a set number of times, or until the stop button is clicked

Stop after:

10

Stop on error

Invoke

Response

Status code:

200 OK

Response time:

9351ms

Headers:

apim-debug-trans-id: 58a3672ce4b0e13332d303d4-097af6f1-3cc3-4472-bec7-6142c3154bc1
x-global-transaction-id: 26051632
content-type: unknown

Body:

eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJhcGljIiwic3ViIjoiIiwiYXVkIjoiaWQxIiwiZXhwIjoxNDg5MTI0MDg4LCJpYXQiOjE0ODkwODgwODh9.NfQ84arvOY8saTC0gvMrG6bv8-Yfou9G_-gYTf_27CY

Debug

To test the validate, just copy the token and select the /validate-jwt operation at the top of the test tool and paste the token in the form of: Bearer XXX; where XXX is the token generated in the previous step.

Bearer <jwt-token>

Authorization *

Bearer eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJhcGljIiwic

Generate

Repeat

Repeat the API invocation a set number of times, or until the stop button is clicked

Stop after:

10

Stop on error

Invoke

Response

Status code:

200 OK

Response time:

10557ms

Headers:

apim-debug-trans-id: 58a3672ce4b0e13332d303d4-625ccebd-b3c8-4616-bd00-226be761128f

x-global-transaction-id: 26068128

content-type: unknown

Body:


```
{
  "iss": "apic",
  "sub": "",
  "aud": "id1",
  "exp": 1489124088,
  "iat": 1489088088
}
```


Debug

As you can see we can extract the iss and aud claims from the token.