IBM App Connect Enterprise

A Beginner’s Guide for Developers

Featuring:
ACE Toolkit, REST API, Applications, ESQL, Shared Libraries, Flow Debugger, Flow Exerciser, File Nodes, Kafka Nodes, Admin WebUI, Credential Vault

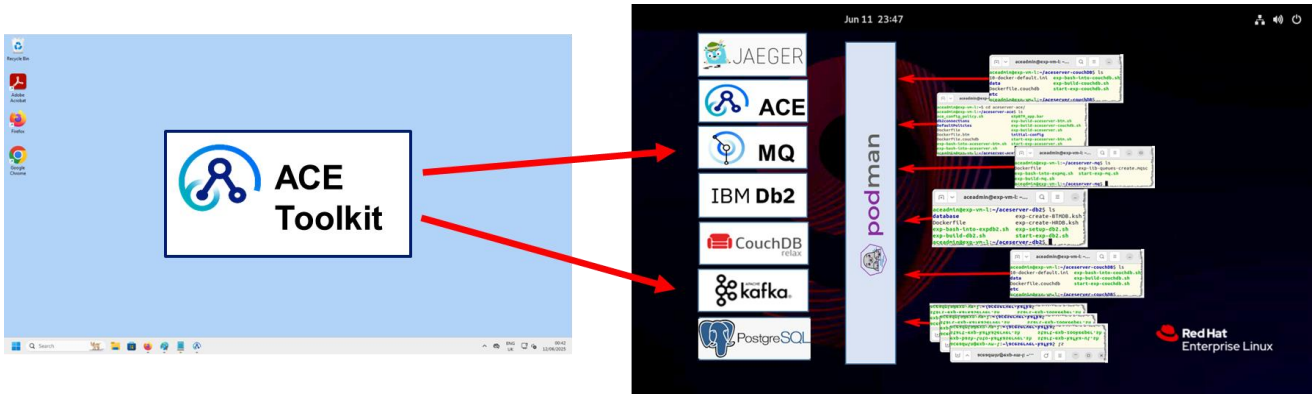
IBM early eXperience programs
Author: bthomps@uk.ibm.com

1. Introduction

The purpose of this Lab is to provide existing developers who do not know IBM App Connect Enterprise with a basic knowledge of the components of ACE so that they can begin their journey in using ACE to solve business problems.

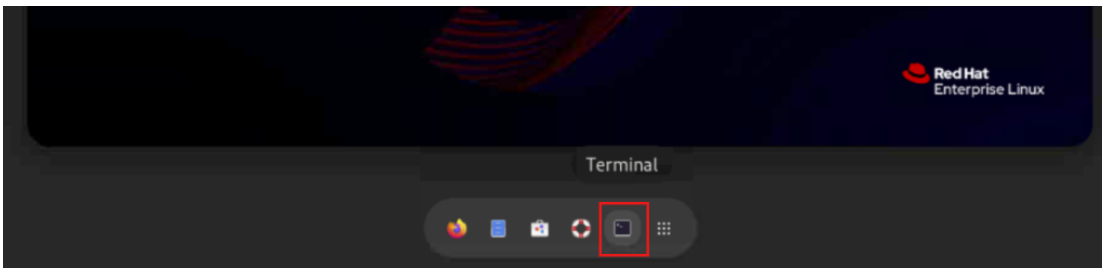
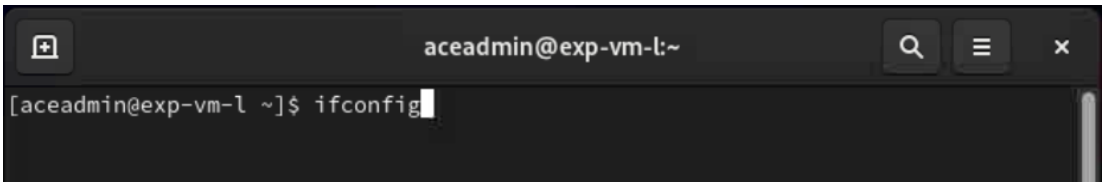
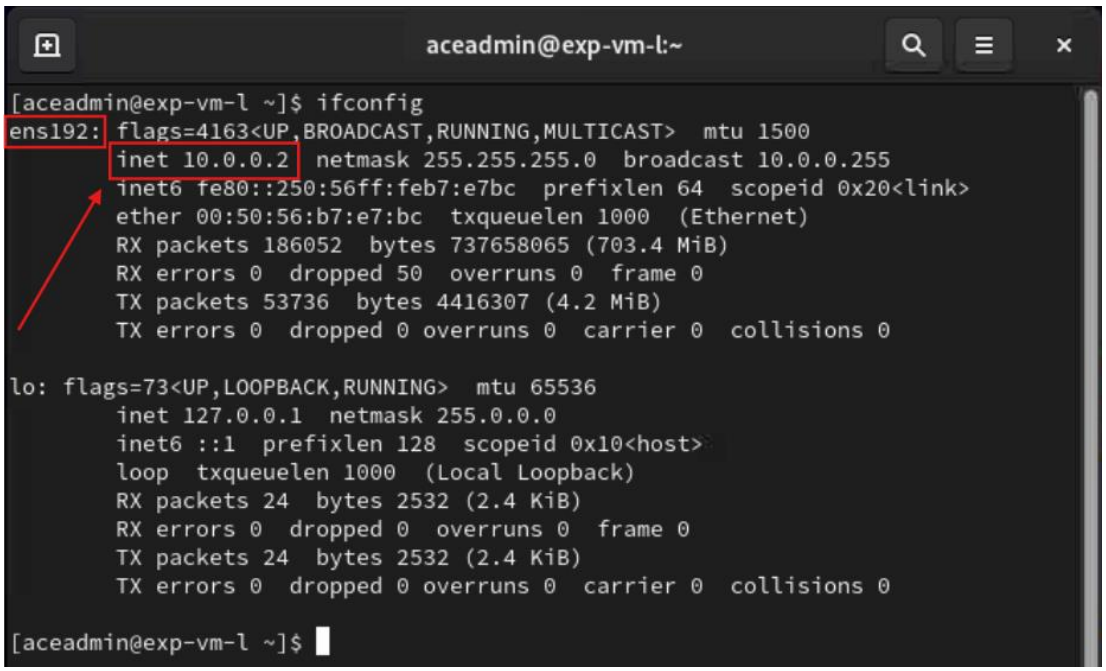
This lab assumes that you have at your disposal:

- A Windows VM Image which has ACE 13.0.3.0 installed
- A Linux VM Image which can support running the following containers:
  - IBM MQ Advanced for Developers v9.2.4
  - ZooKeeper, Kafka Server, UI for Apache Kafka



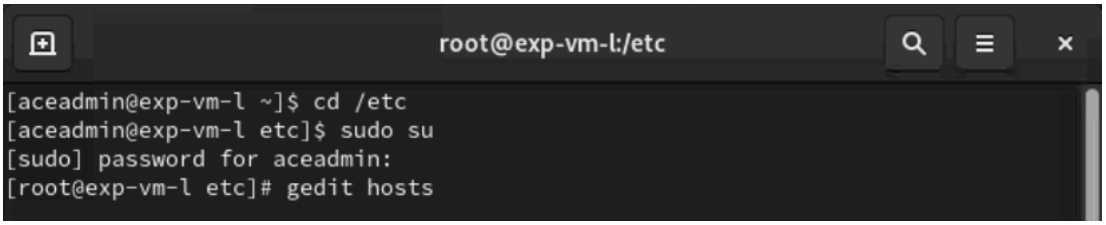
2. Configure access to MQ and Kafka on Linux

2.1 Obtain Linux Environment IP Address

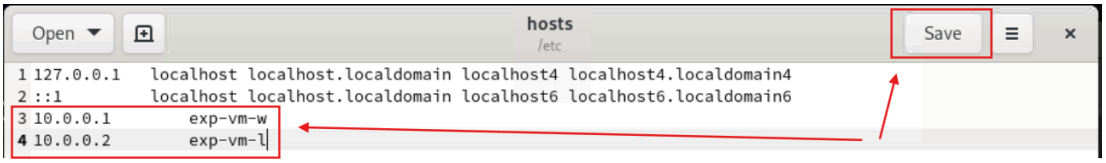
1.	In the Linux environment sign in using `aceadmin / IBMDem0s` and open a terminal window: 
2.	Enter the following command to establish the IP address of your Linux environment: `ifconfig` 
3.	The IP address of your Linux VM is the value of inet in ens192 (in this example **10.0.0.2**): 

4. Start a new terminal session and change directory to /etc and then we will edit the hosts file. To do this use the following commands:

```
cd /etc
sudo su
<type in the aceadmin password which is IBMDem0s>
gedit hosts
```



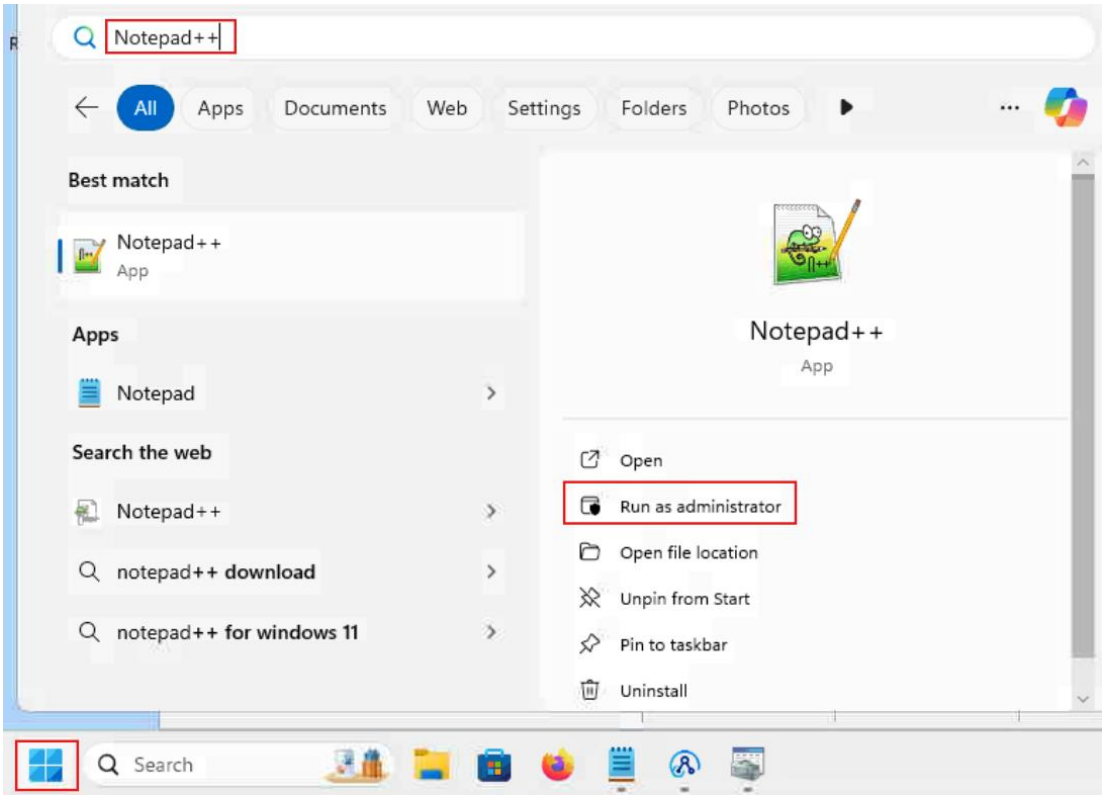
The hosts file will open. Check the IP addresses and hostnames for the Windows and Linux vm are set up correctly, and change them if necessary. Click the **Save** button:



2.2 Set the Hostname of the Linux VM in Windows

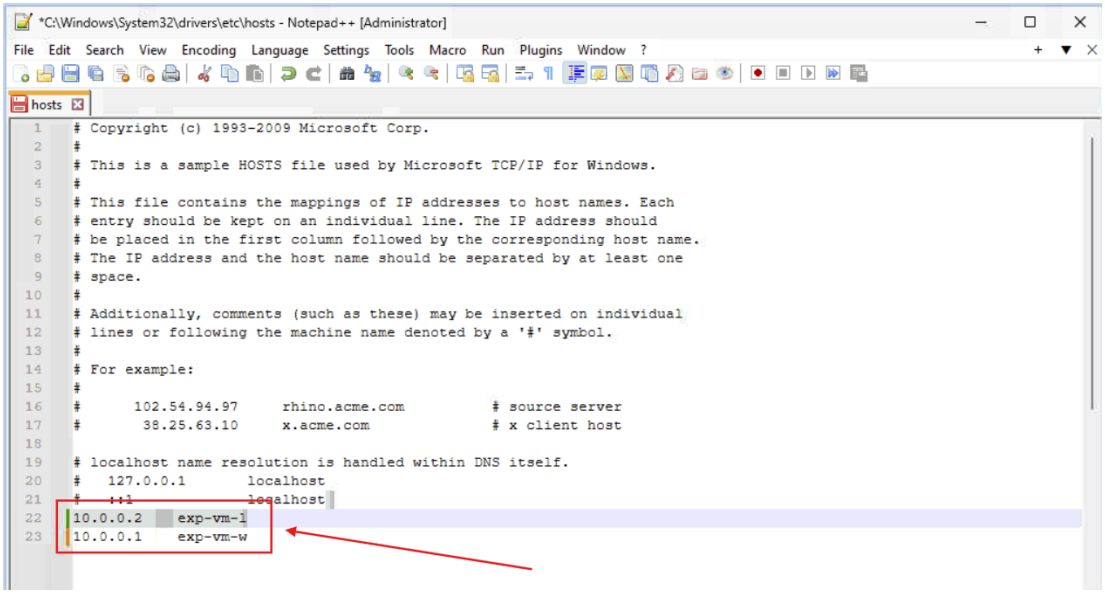
5. In the Windows environment sign in using `aceadmin / passw0rd`

   Click the **Windows Start** button, then search for **Notepad++** and choose the option to **Run as administrator**:



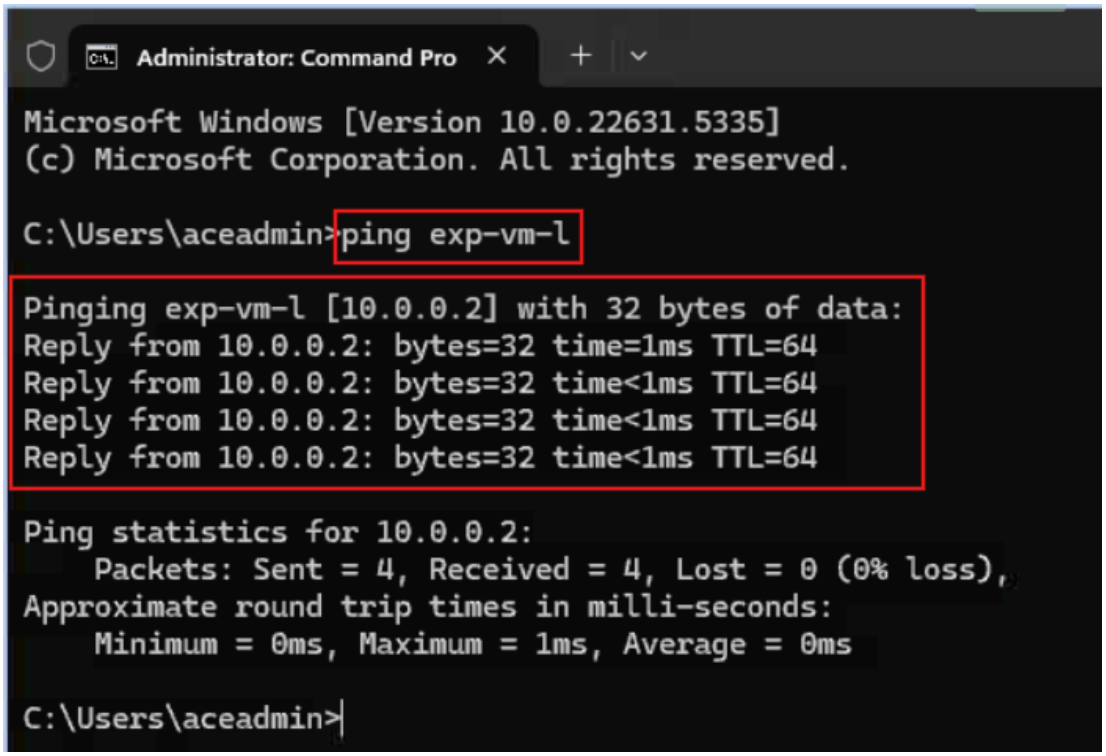
6. In Notepad++ open the file `C:\windows\System32\drivers\etc\hosts`

   Set the IP address of exp-vm-l to the IP address of your Linux VM (in this example 10.0.0.2). Whilst we are here, also set the IP address of the Windows VM (in this example 10.0.0.1):



7.	Save the file and close Notepad++

2.3 Check Connectivity using ping

8.	Open a command prompt and enter the command: `ping exp-vm-l` Ensure you receive a reply from pinging the host: 

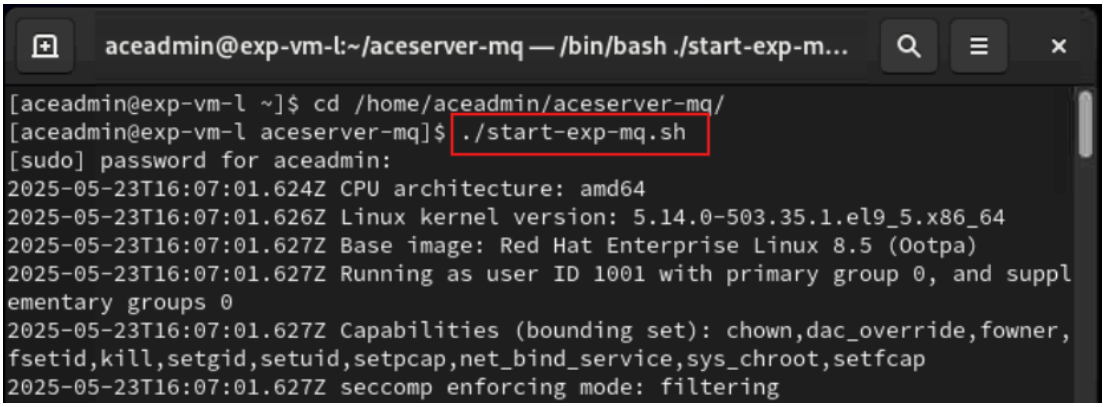
2.4 Start the MQ container environment

In this section you will run an IBM MQ container so that you can use it with ACE. The configuration uses the standard IBM MQ Advanced for Developers image on Docker hub: https://github.com/ibm-messaging/mq-container

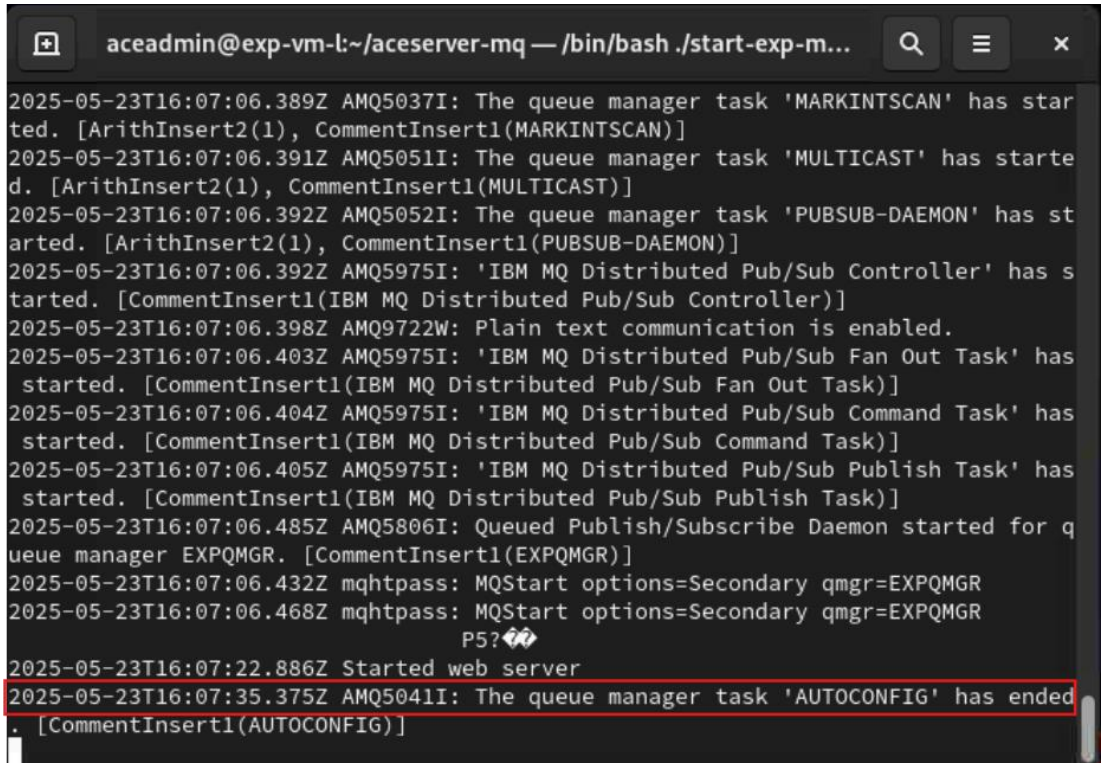
9.	In the Linux Environment, open a new terminal and navigate to: `/home/aceadmin/aceserver-mq`

10. Enter the following command to run an eXp configured MQ container:

```
./start-exp-mq.sh
```



11. The MQ container will run in this window and control will not be given back to this window until you stop the container. If you want to stop the container you can press **<ctrl> c** (note this will also delete the container as the podman run command used to start the container (in `start-exp-mq.sh`) has `"--rm"` specified. This helps with restarting a "clean" container):



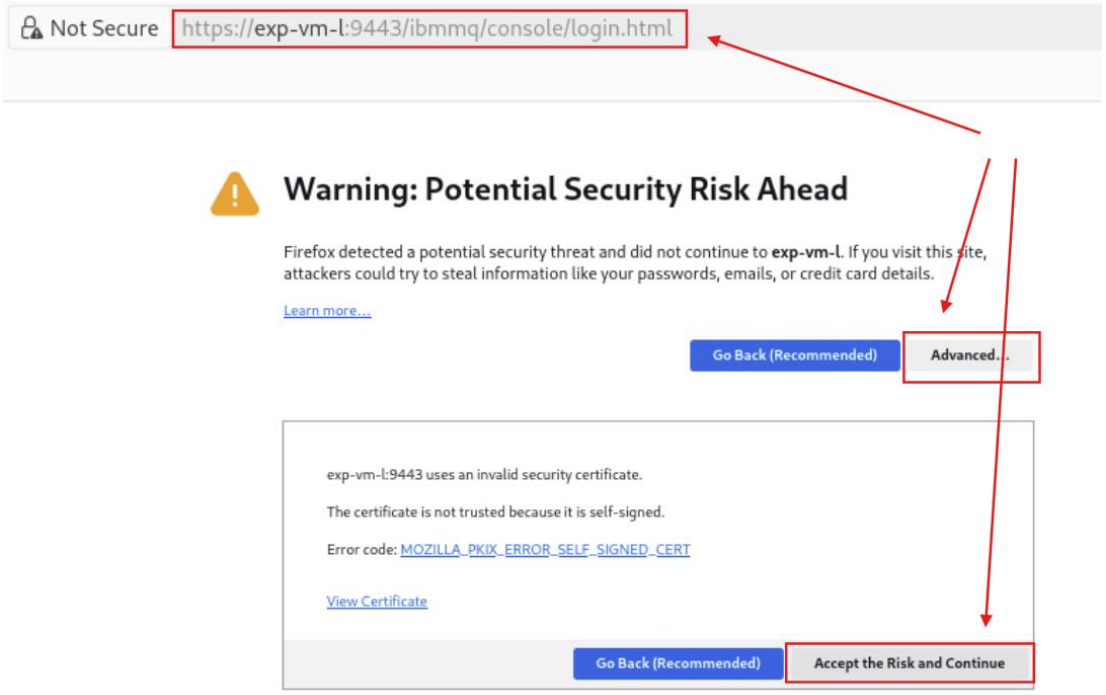
12. Leave the MQ container running in the terminal window.

    If you need to stop the container enter **<ctrl> c** in the container to stop it.

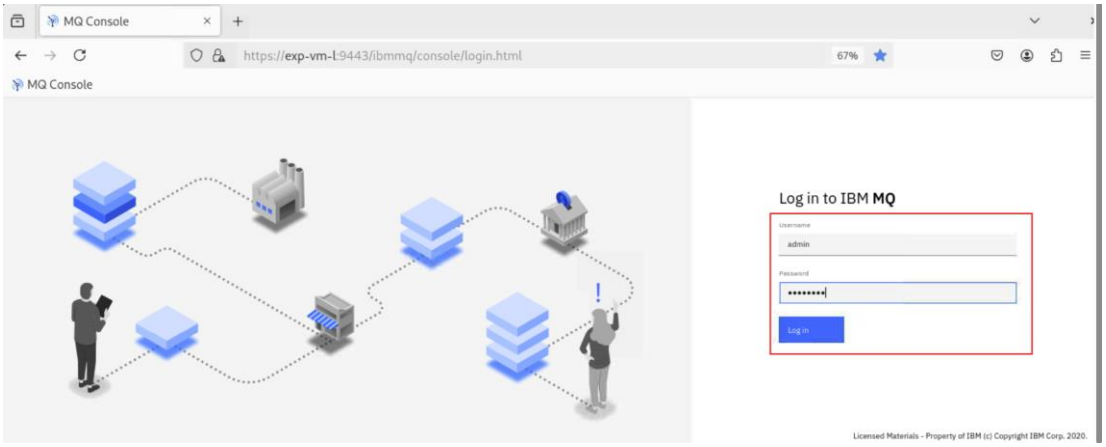
    NOTE: Any changes that you make to the environment will not be saved.

13. We will now verify the MQ container. Open the MQ Console in a web browser tab using the URL:
https://exp-vm-l:9443/ibmmq/console/login.html

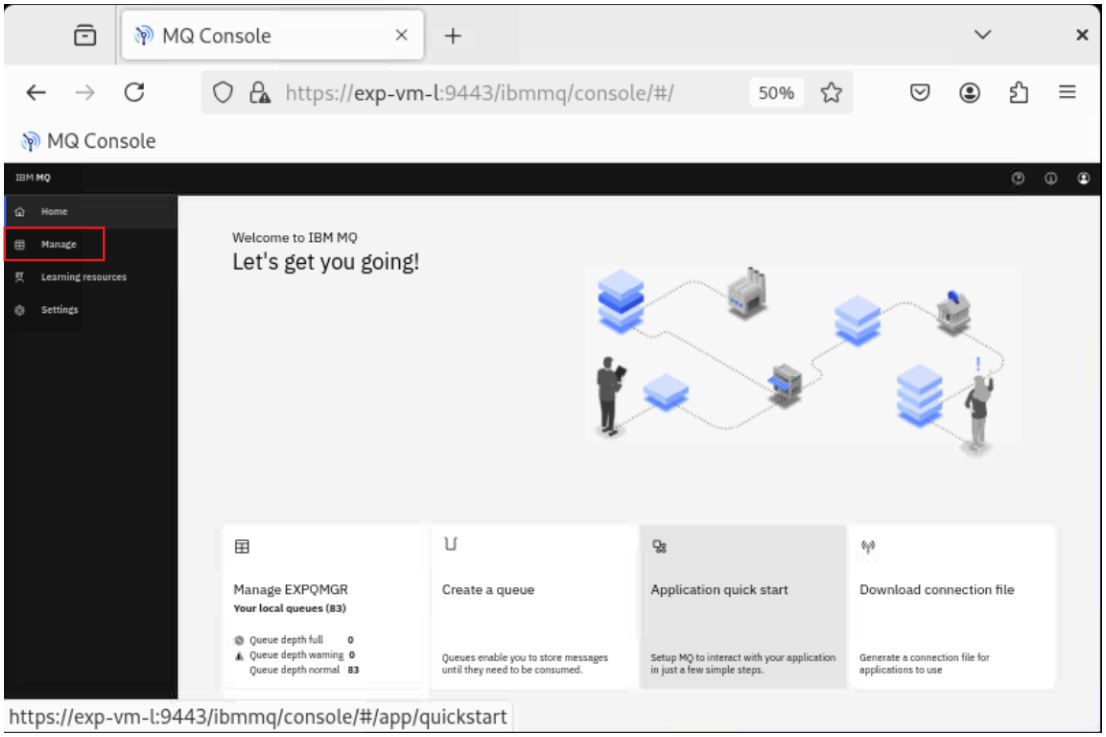
If you see a **Potential Security Risk Ahead** message, click the **Advanced** button and then **Accept the Risk and Continue**:

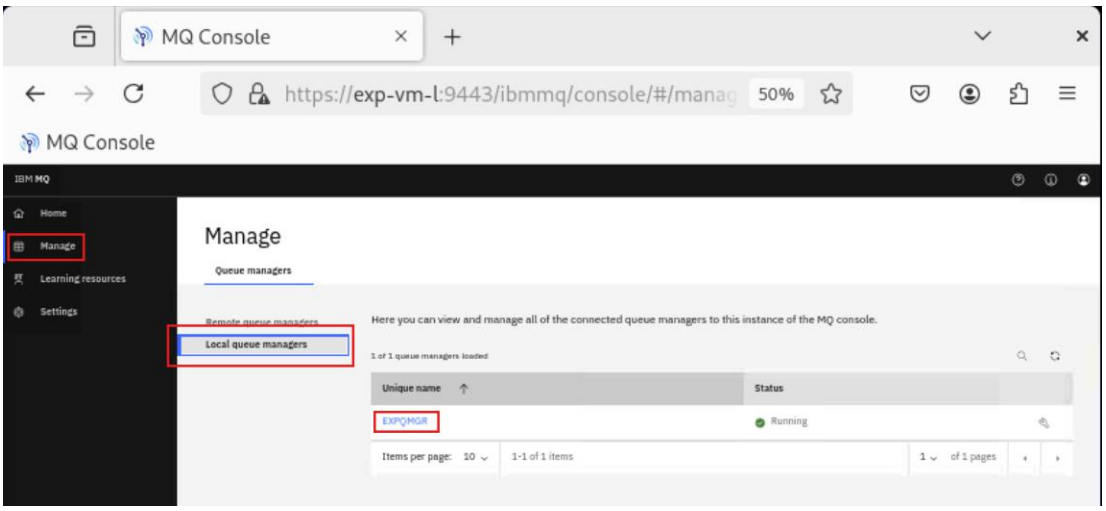


The login credentials are: `admin/passw0rd`



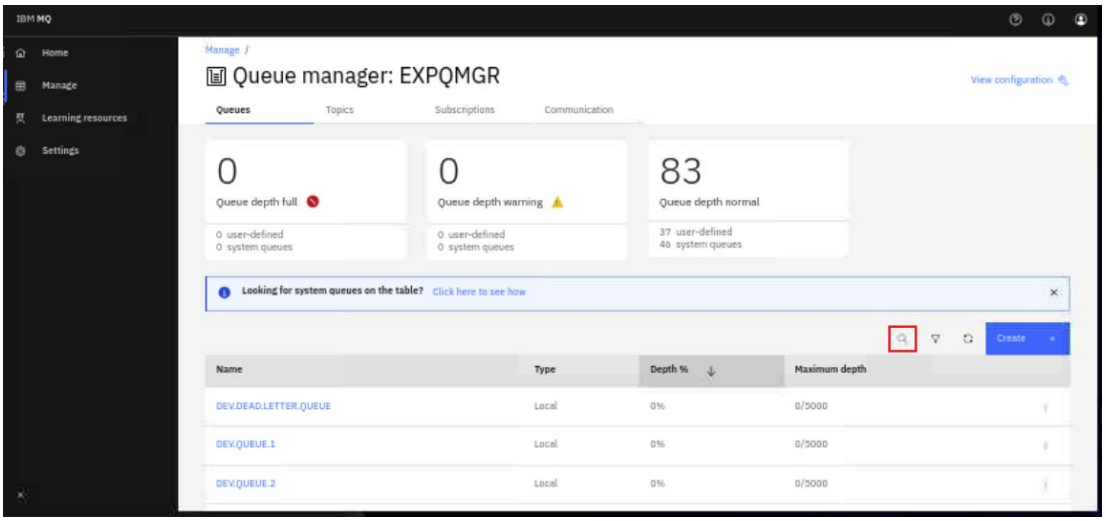
14. Select **Manage > Local queue managers > EXPQMGR** (make sure the Queue Manager is running):



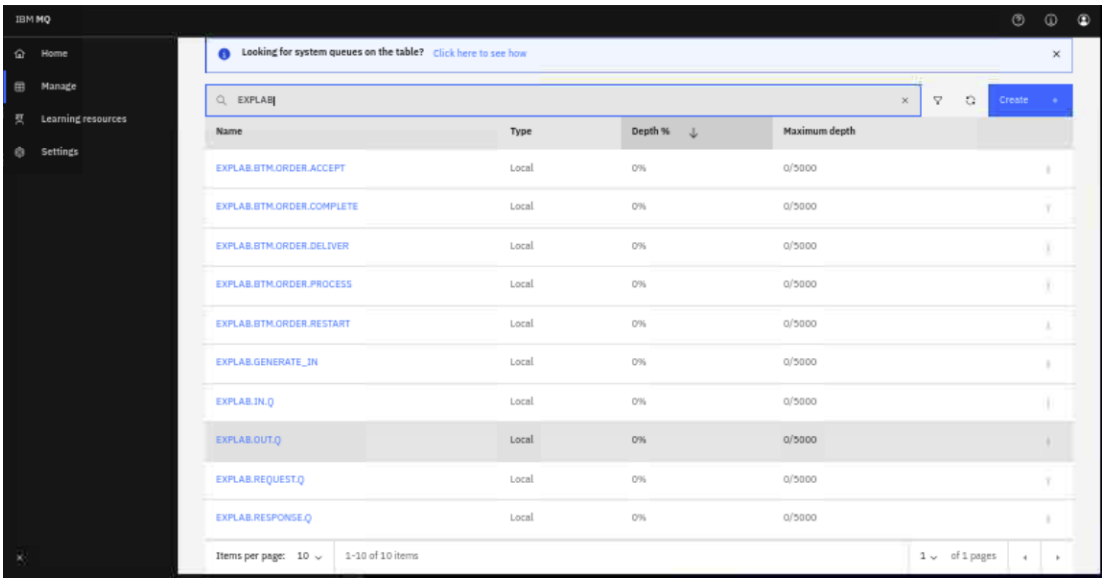


15. All the queues defined to the queue manager will be shown.

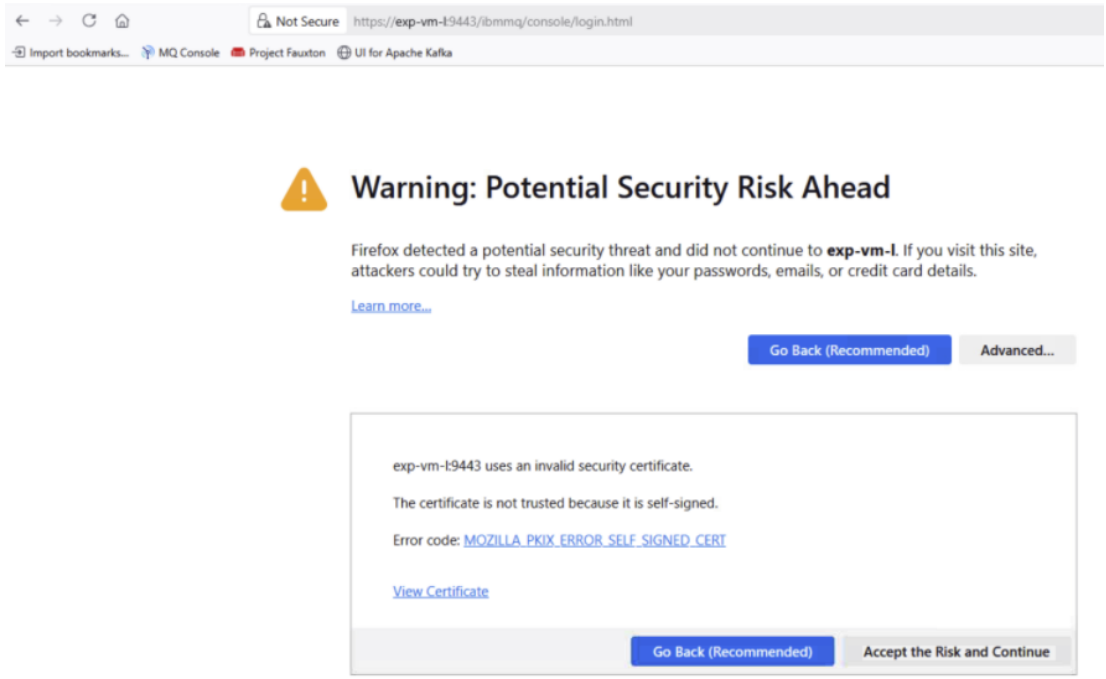
Click on the magnifying glass and search for **EXPLAB**:



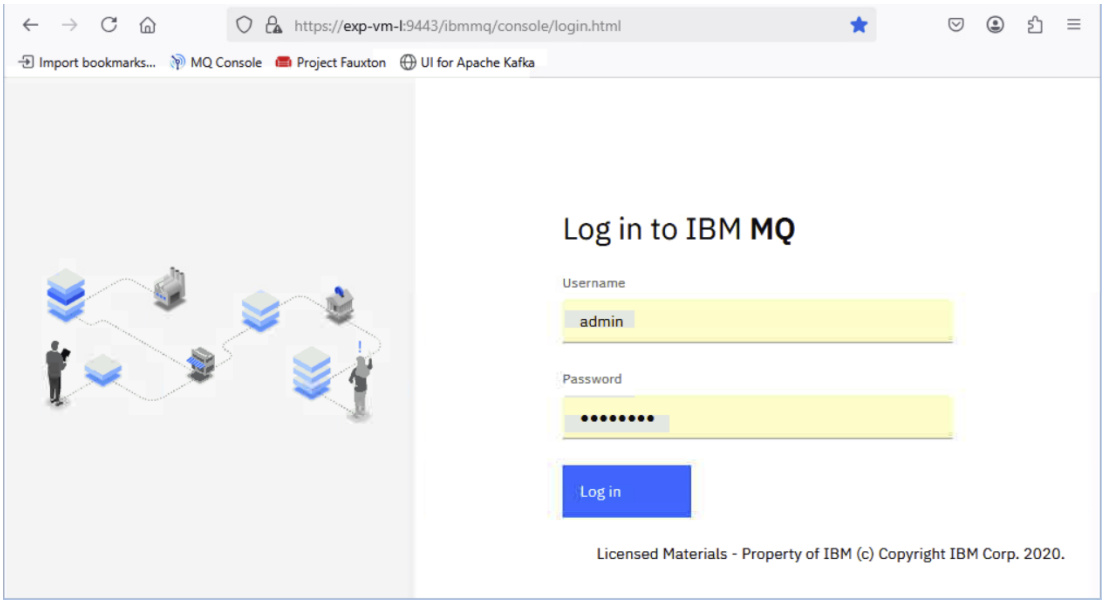
16. You will see queues defined prefixed **EXPLAB**



17. Switch over to the Windows environment and also check that you can open the MQ Console from there too (use the book-marked link in Firefox). If you see a **Potential Security Risk Ahead** message, click the **Advanced** button and then **Accept the Risk and Continue** just like before:



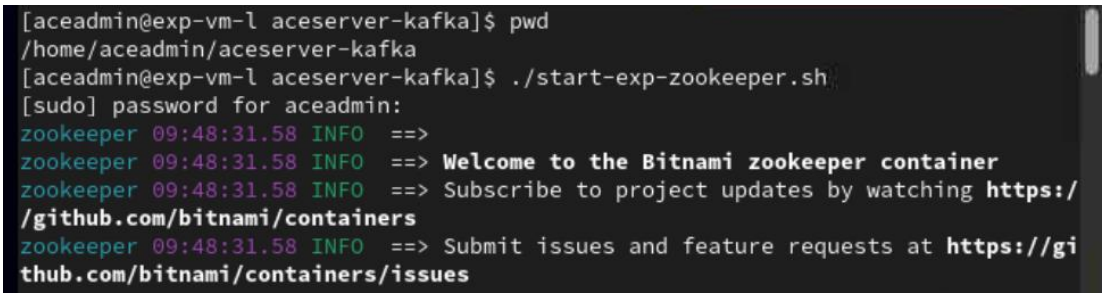
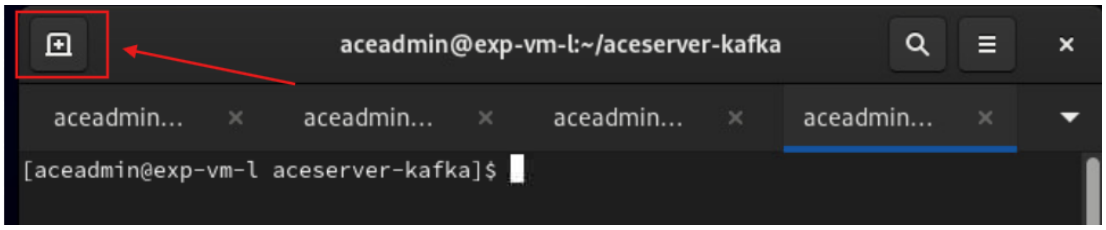
18. You should now see the IBM Console login:

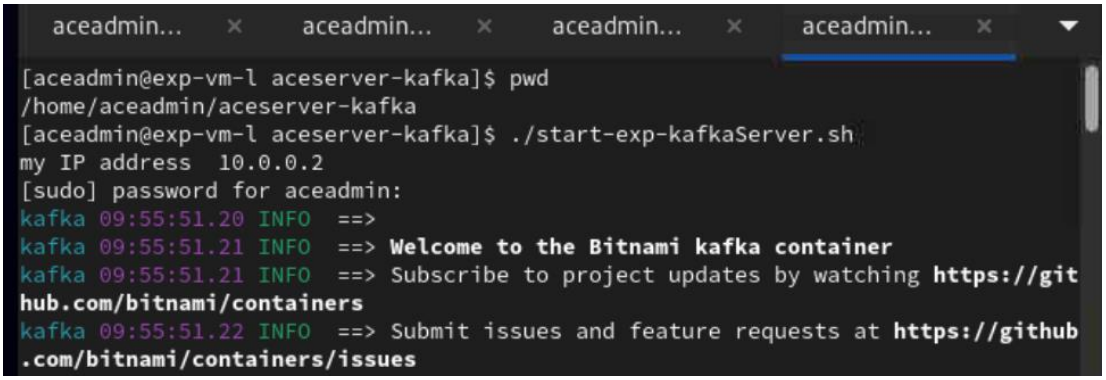
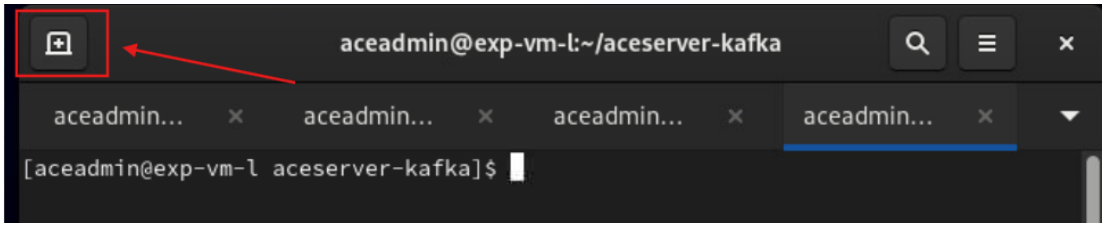


2.5 Start the Kafka container environment

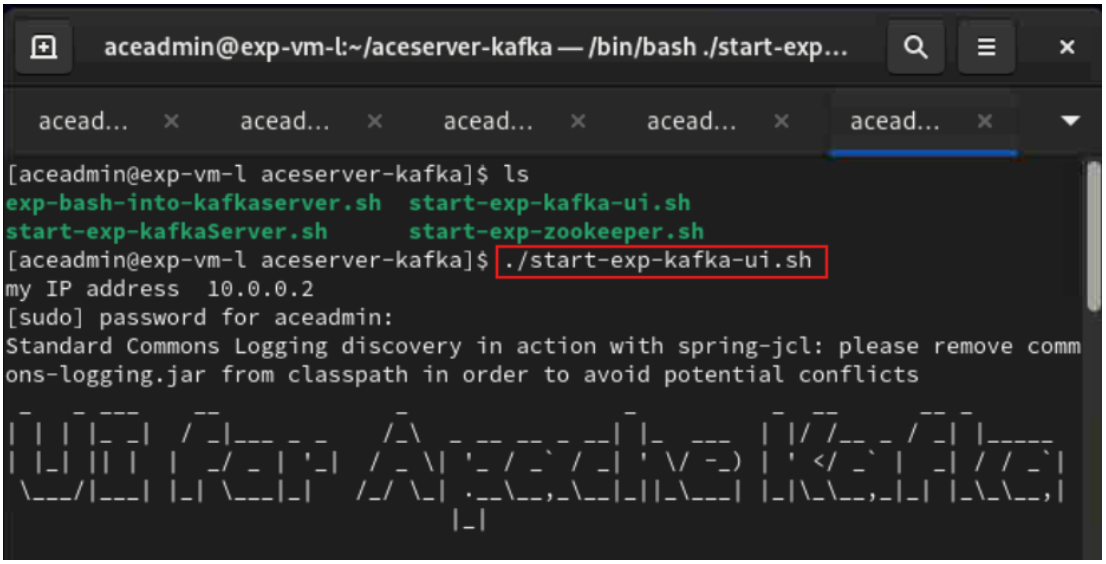
In this section you will run and configure a container based Kafka environment for use with eXp Labs. The configuration uses the Apache Kafka container images packaged by Bitnami on Docker hub, more information can be found here:

https://github.com/bitnami/bitnami-docker-kafka

19.	In the Linux Environment, open a new terminal and navigate to: `/home/aceadmin/aceserver-kafka`
20.	Enter the following command to run an eXp configured MQ container: `./start-exp-zookeeper.sh` The first time you start this command the image that the container will use to start will be downloaded into your environment (*make sure you have connectivity to the internet*). You will be prompted for the password for aceadmin (which as a reminder is `IBMDem0s`).
21.	The Zookeeper container will run in this window. You can stop the container using <ctrl> c. 
22.	In the Zookeeper terminal window, open a new tab.  Navigate into the `aceserver-kafka` directory and enter the following command: `./start-exp-kafkaServer.sh` The first time you start this command the image that the container will use to start will be downloaded into your environment (*make sure you are connected to the internet otherwise this will fail*).

23.	The Kafka server container will run in this window. You can stop the container using <ctrl> c. 
24.	An (open source) UI for Apache Kafka is available packaged in a Docker container. The UI for Apache Kafka is a tool for viewing messages written to Kafka topics. More information can be found here: https://hub.docker.com/r/provectuslabs/kafka-ui In the terminal window running Zookeeper and your Kafka server, open another new tab.  Navigate into the aceserver-kafka directory and enter the following command: `./start-exp-kafka-ui.sh` The first time you start this command the image that the container will use to start will be downloaded into your environment (*make sure you are connected to the internet otherwise this will fail*).

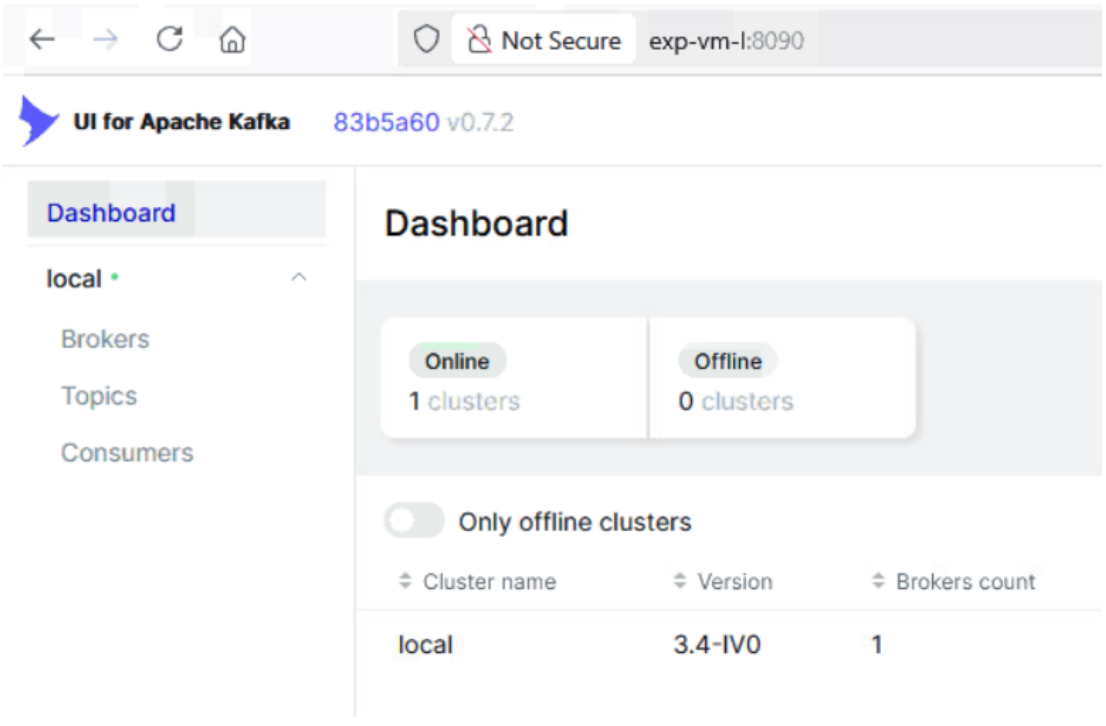
25. The Kafka UI container will run in this window. You can stop the container using <ctrl> c.



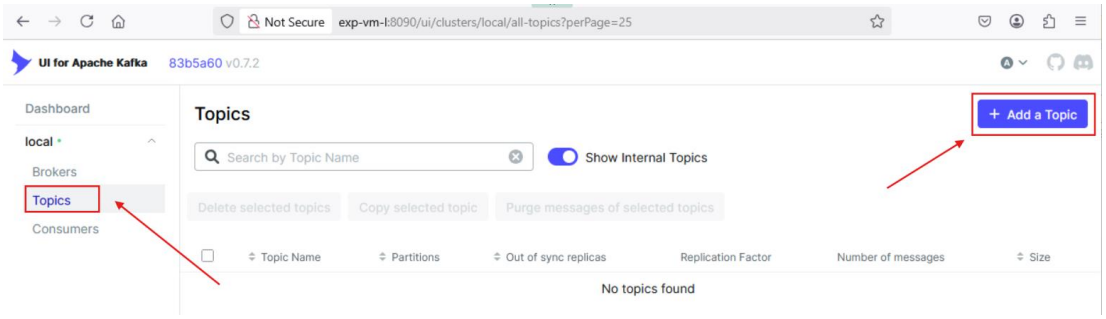
26. In your Windows Environment, open a new Firefox browser tab and go to the following URL:

http://exp-vm-l:8090/

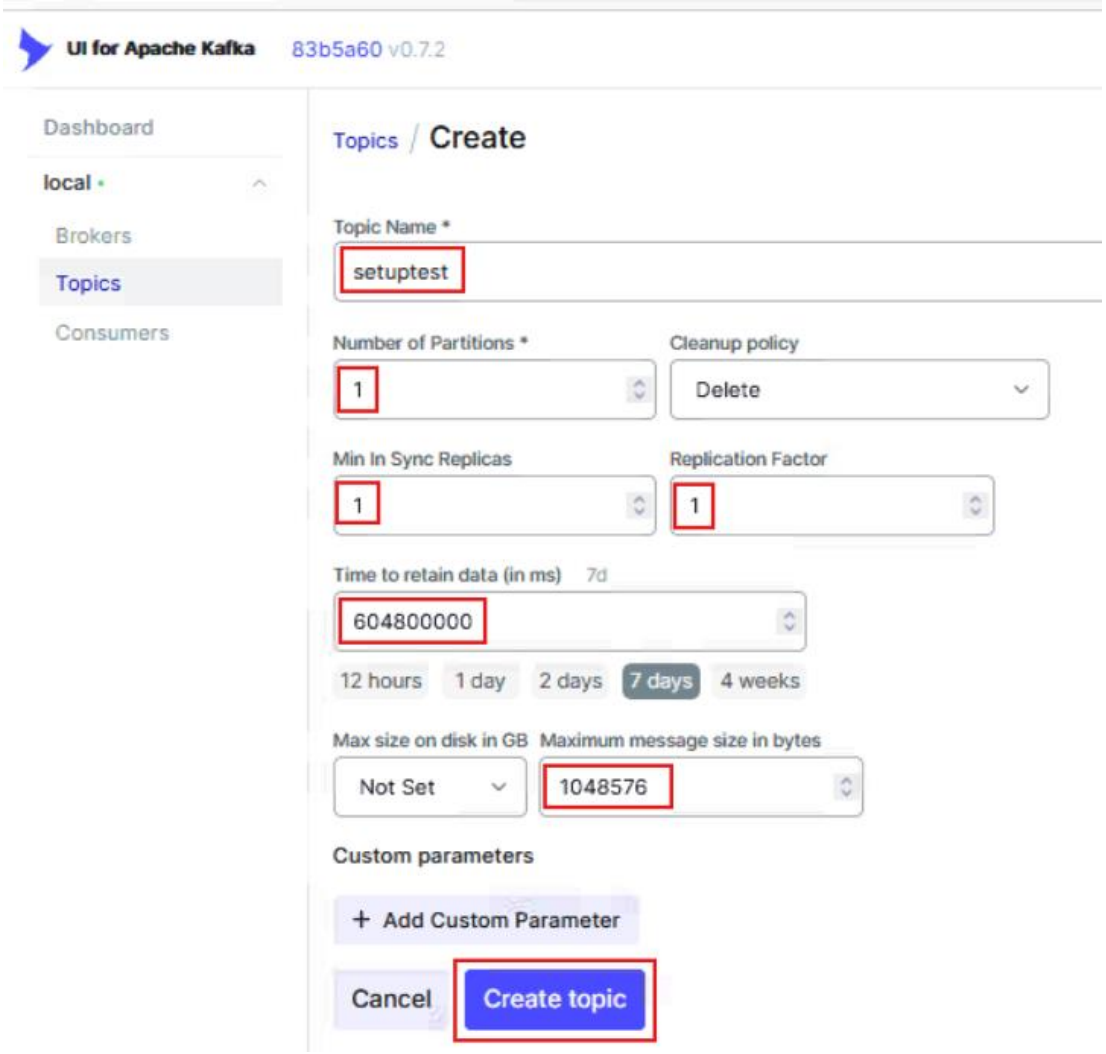
The UI for Apache Kafka will be shown in the browser:

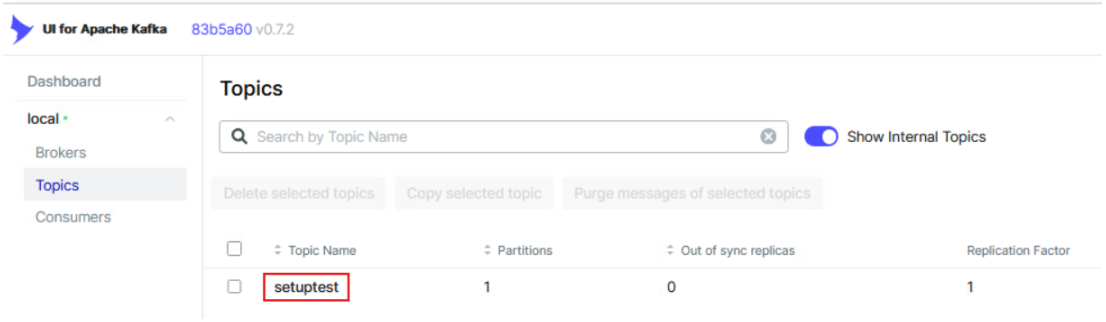


27. Click **Topics**, then the **Add a Topic** button:

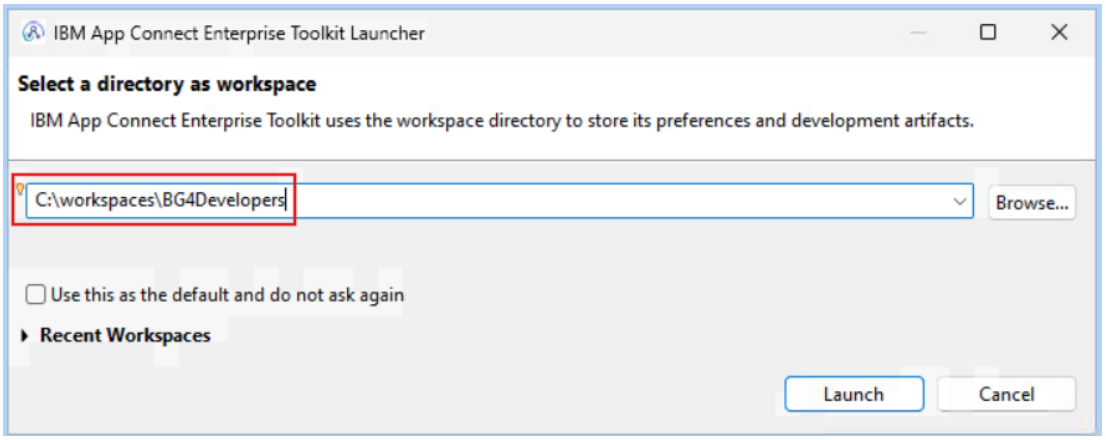


28. Supply a Topic Name, and set the following properties and then click **Send** to create a test topic:
    - **Topic Name** = `setuptest`
    - **Number of Partitions** = `1`
    - **Min In Sync Replicas** = `1`
    - **Replication Factor** = `1`
    - **Time to retain data (in ms)** = `604800000 (7 days)`
    - **Maximum message size in bytes** = `1048576`

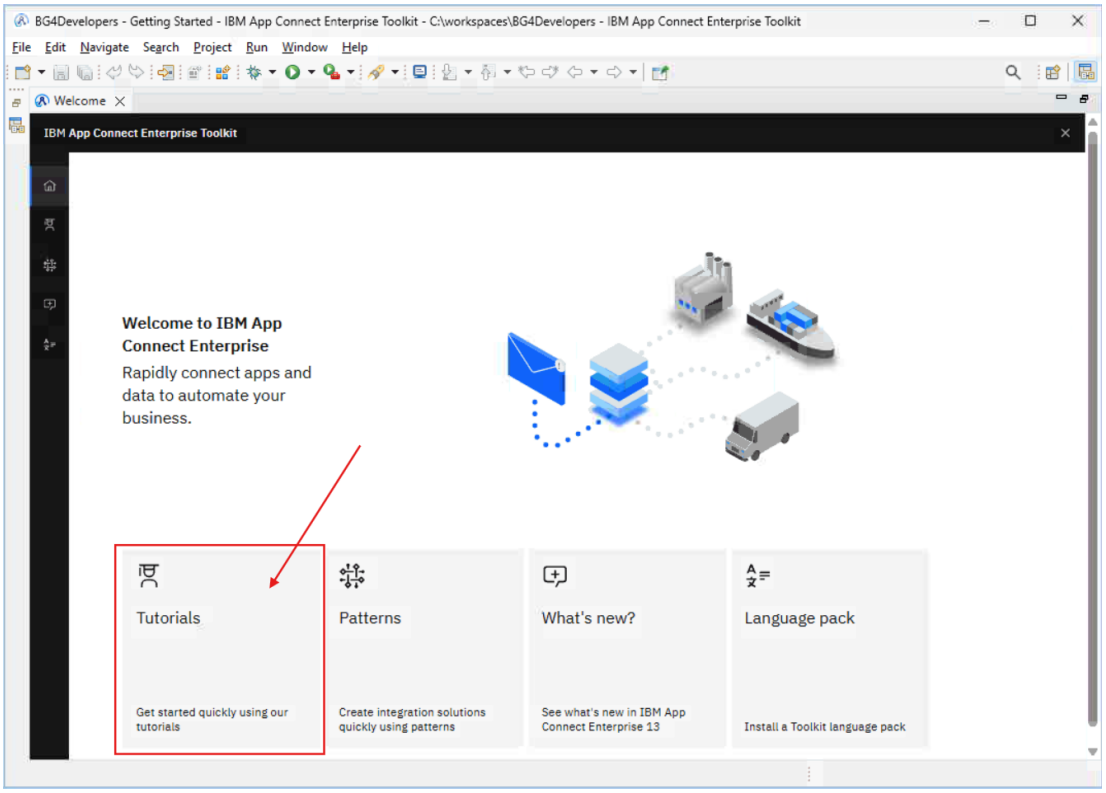


29.	The Topic should appear in the list of All Topics: 

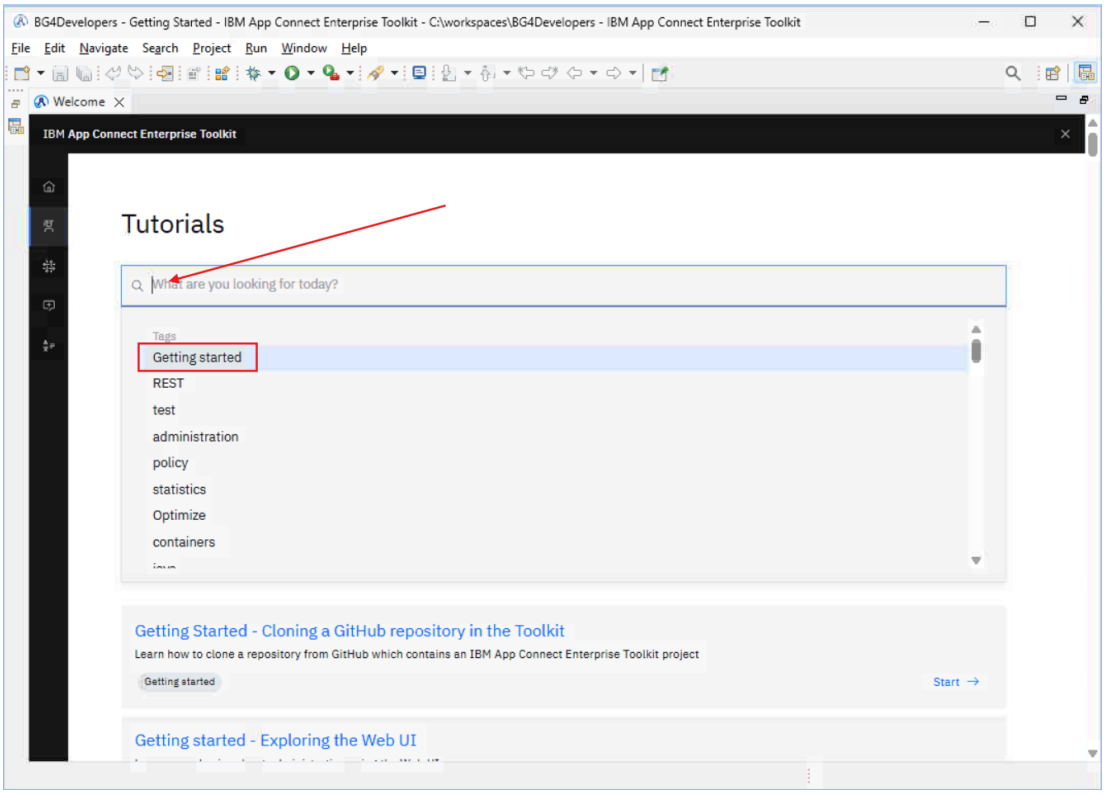
3. Configure ACE Toolkit Development environment.

30.	In the windows environment, open the v13.0.3.0 IBM App Connect Enterprise Toolkit. Create a new workspace for the work in this lab guide, call it BG4Developers: 

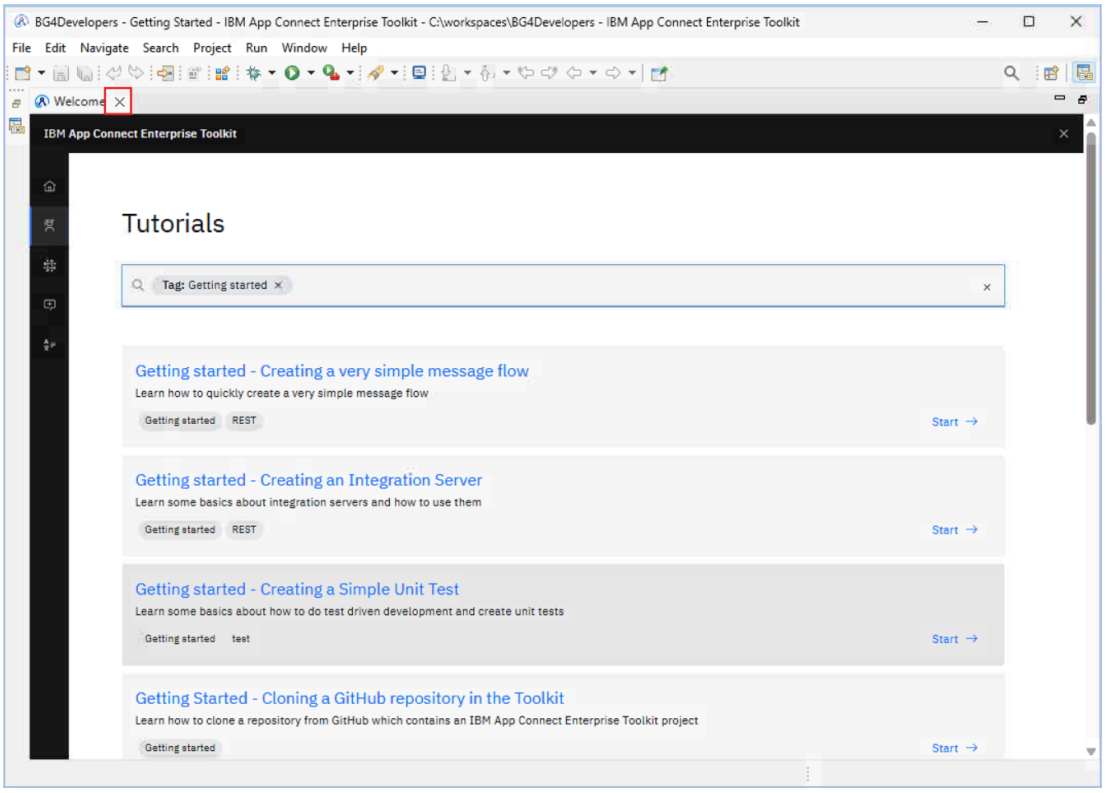
31. In the **Welcome to IBM App Connect Enterprise Toolkit** window, click the Tutorials tile (*Note: access to the internet is required in order to see the directory of Tutorials available in the ACE Toolkit*):



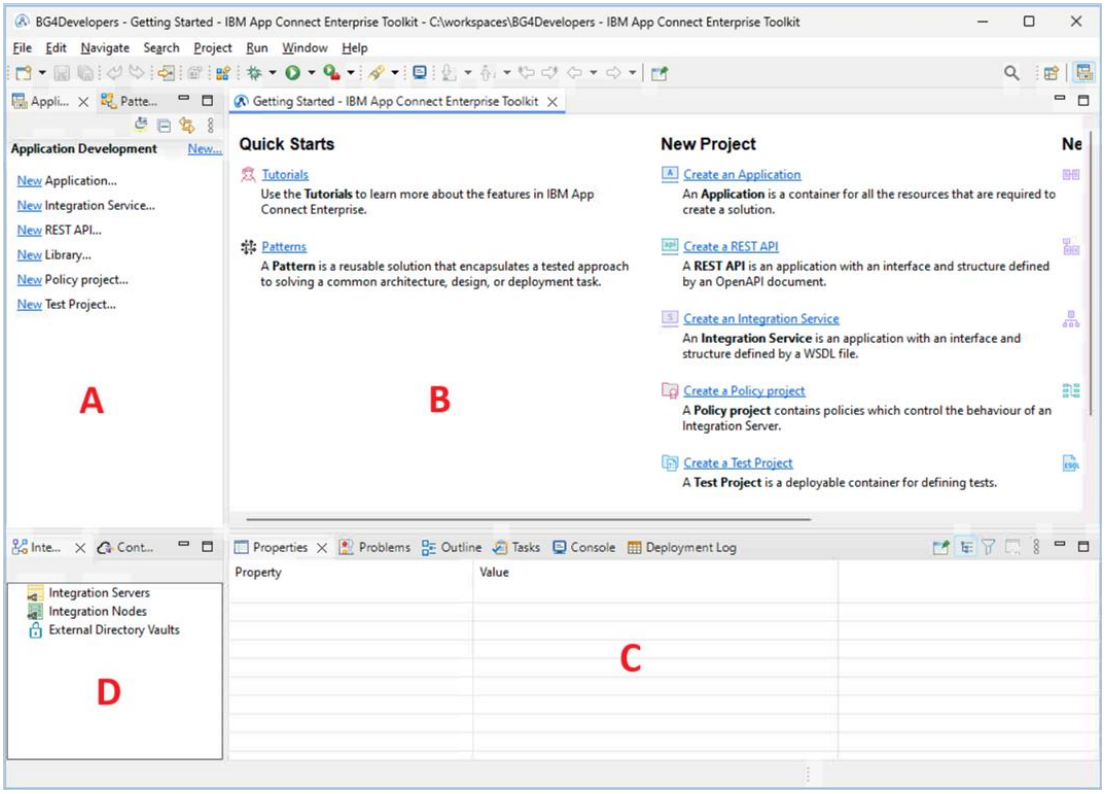
32. The list of Tutorials will show on the page. Click in the Tutorials search bar to see the list of pre-defined tags. Click the **Getting started** tag to see a subset of the tutorials related to Getting started:



33. The search filter will show all **Getting started** tutorials. Close the Welcome window.



34. The eclipse based App Connect Enterprise Toolkit will open:

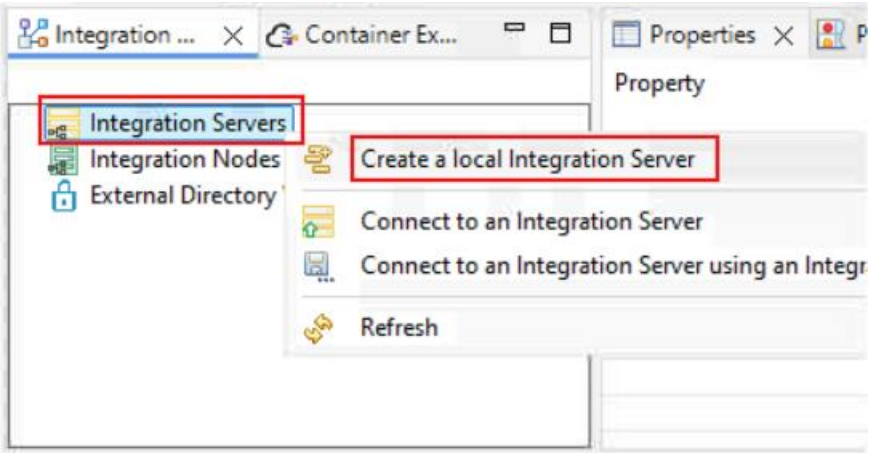


A) The **Application Development** view **(A)** is where your Applications, REST APIs etc will be shown in your workspace
B) View **(B)** is where resources that you open (for example message flows) will be shown.
C) View **(C)** is where properties of resources that you highlight in window **(B)** can be viewed
D) The **Integration Explorer** view **(D)** is where you can view and manage deployed assets (for example Applications and Message flows). Note assets are deployed to Integration Servers that are optionally managed by an Integration Node. When they are managed by an integration node the integration servers will appear “under” an integration node in this view. When not managed by an integration node they will be found under the “Integration Servers” view in this window.

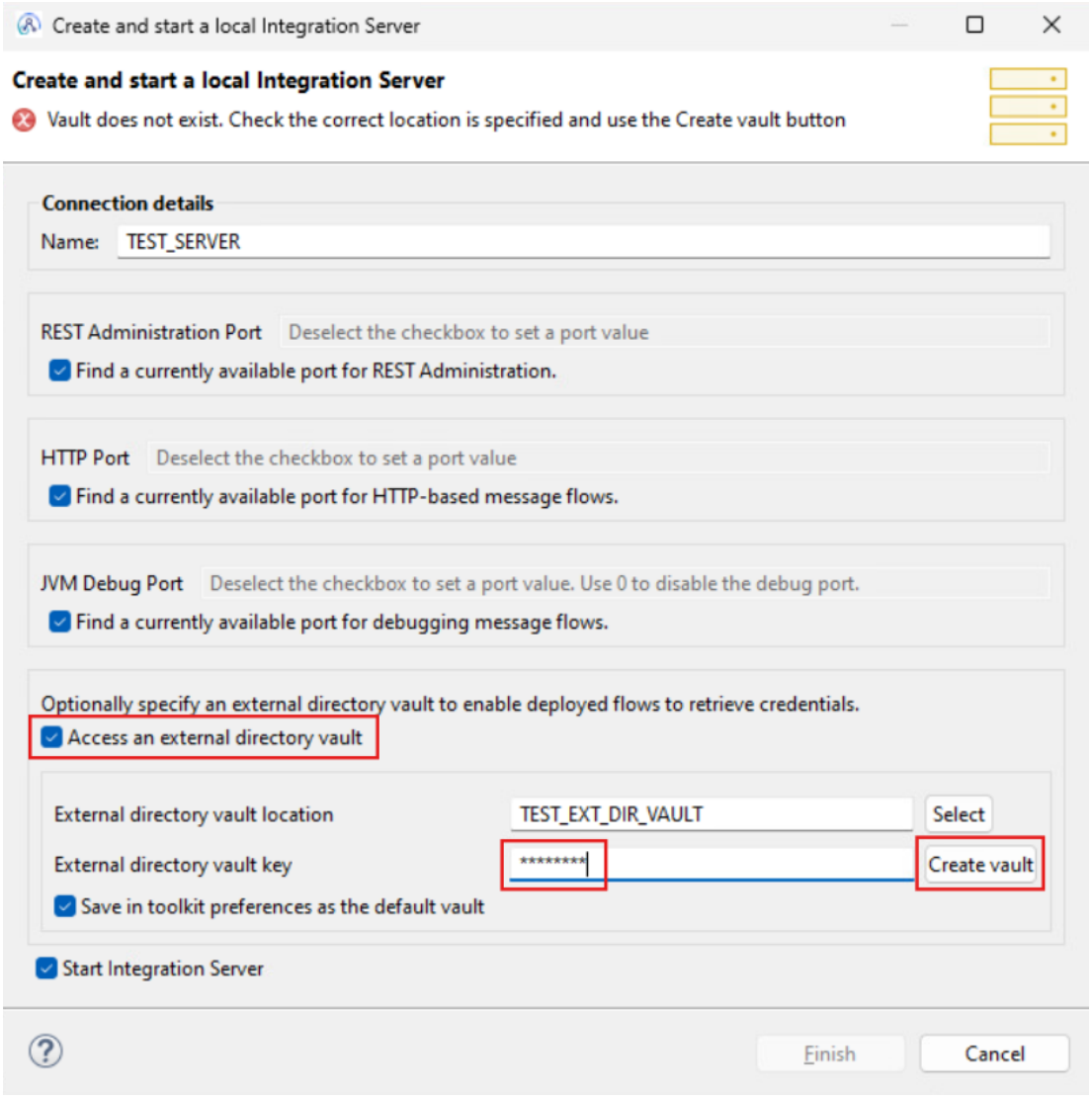
This collection of Views make up what is known as a Perspective. You can switch between Perspectives using the icons in the top right corner. For now, stay in the current Integration Development perspective.

3.1.1 Create an Integration Server (run-time) Environment

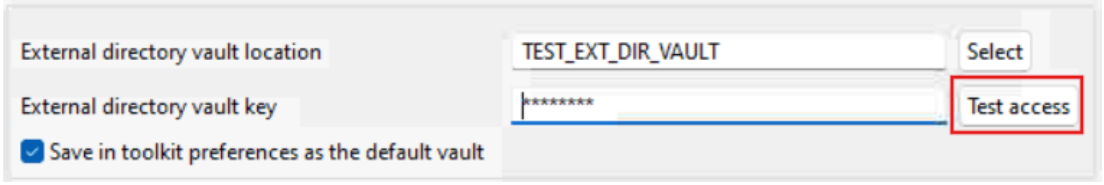
When you create Applications, REST APIs and other resources etc in IBM App Connect Enterprise, these resources need a place to run. These resources run inside an integration server. We often talk about “deploying” resources to an integration server. You will now create a local integration server that you will use to test the resources that you will create in this lab.

1.	In the Integration Explorer window **(D)** right click on Integration Servers and select “**Create a local Integration Server**”: 

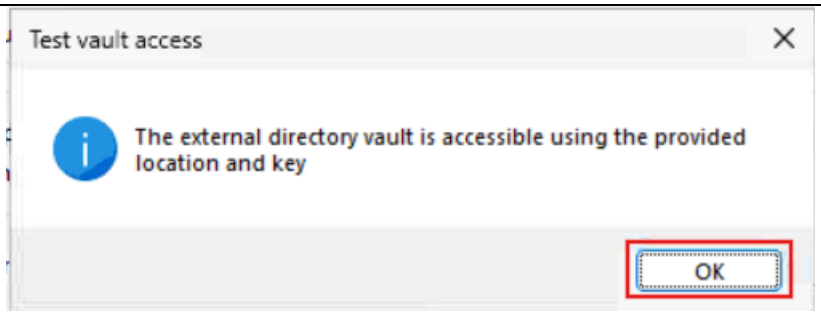
2. We will leave most of the settings with their defaults in the "***Create and start a local Integration Server***" dialog, however …
   - Select the tick box to "**Access an external directory vault**"
   - The "**external directory vault key property**" will become available. Type a value for the vault key. We recommend the value "**passw0rd**". if you choose your own value please remember it as you may need it later.
   - Click the "**Create vault**" button



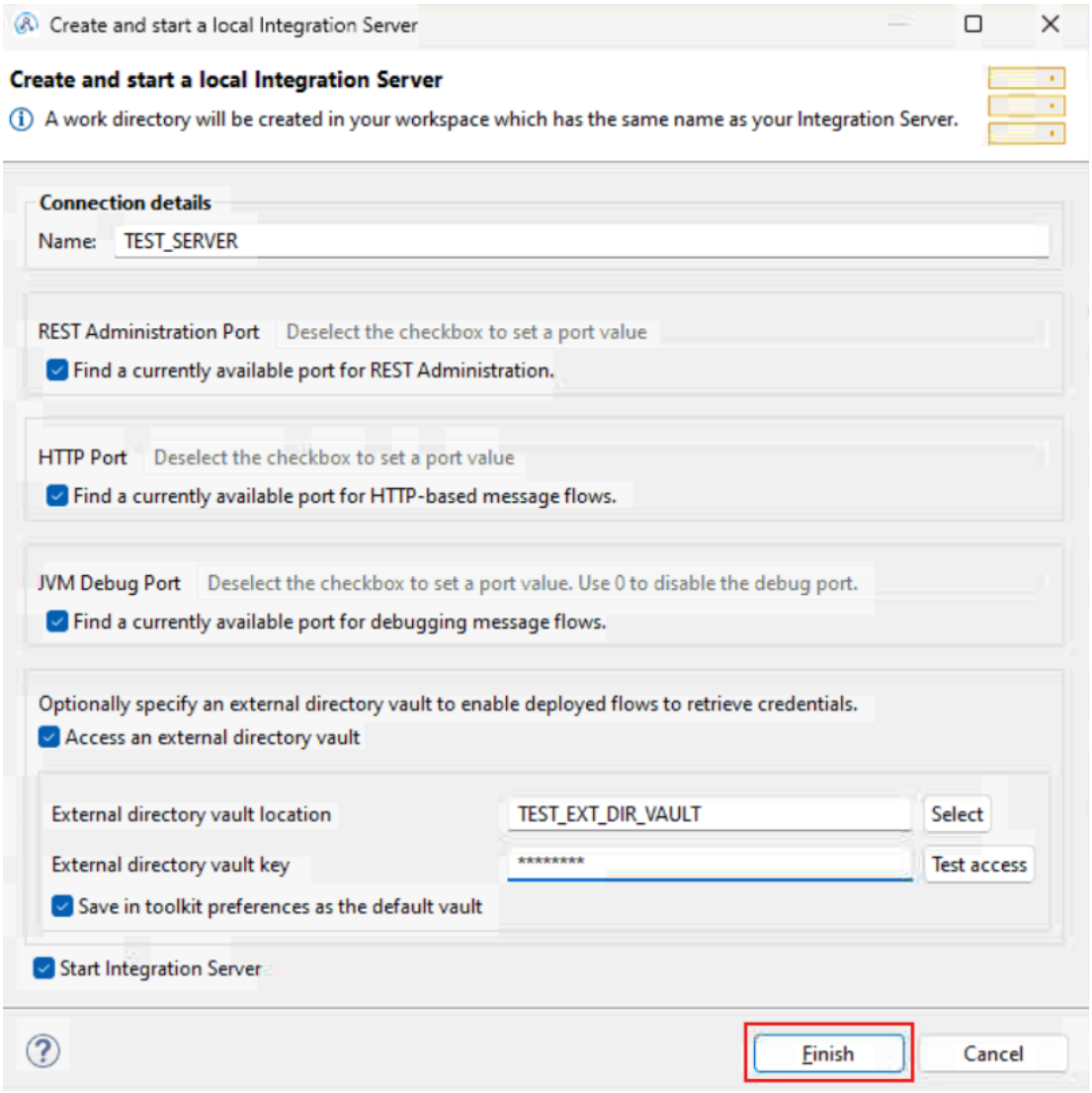
You will notice that the text on the button will change to say "**Test access**"



When you click the "Test access" button a new dialog box will open, showing a successful connection. Click OK.

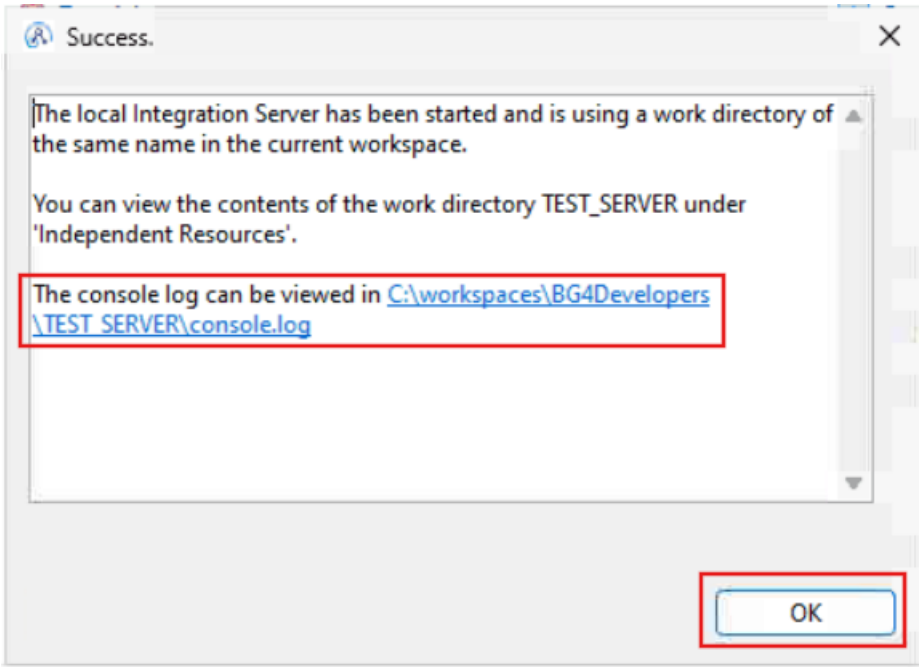
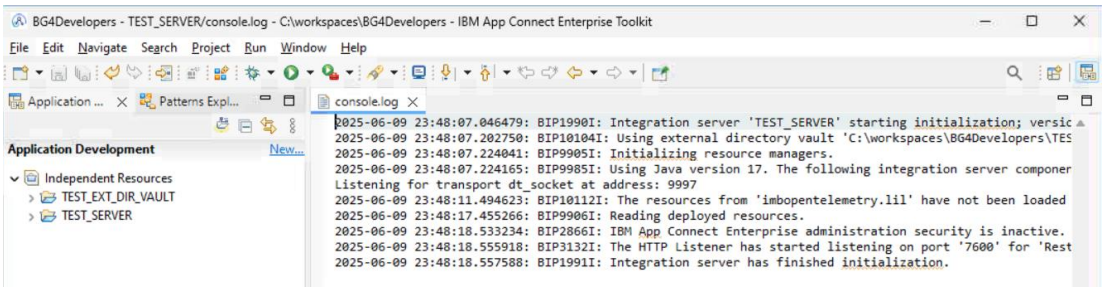


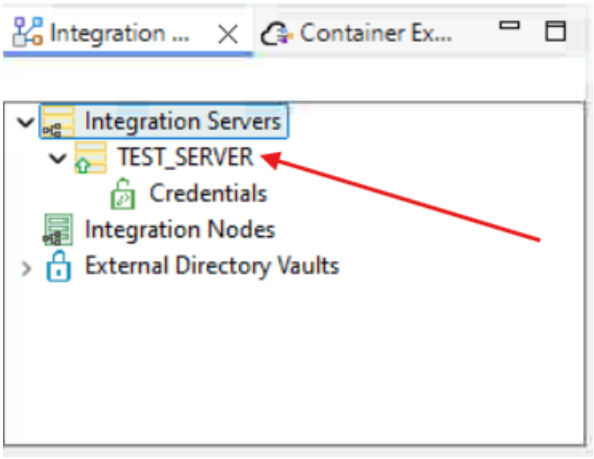
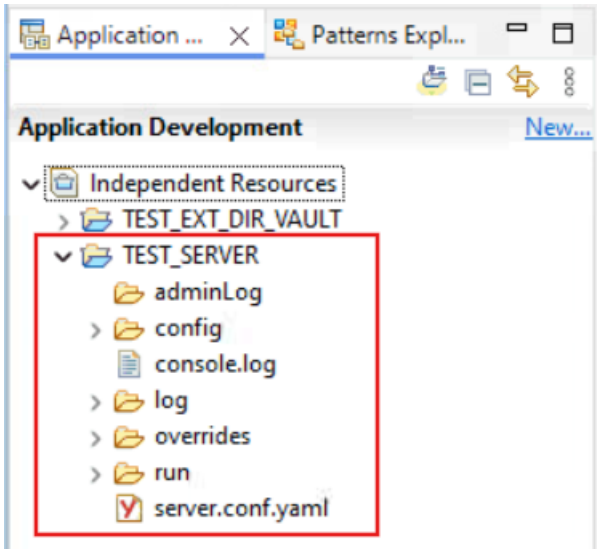
You will be returned to the "**Create and start a local Integration Server**" dialog, where you should click the **Finish** button:



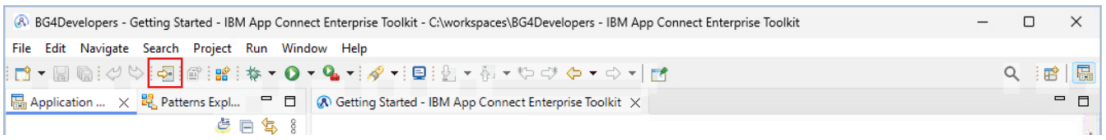
This will trigger a process to configure a local integration server (not managed by a node) and to start it.

Note: the default port for Administration of the server is **7600**. Since the "**Find a currently available port for REST administration**" is ticked, the process will start from 7600 and if not available, add one to this port number until an available port is found (*equivalent logic applies for the HTTP Port and JVM Debug Port*).

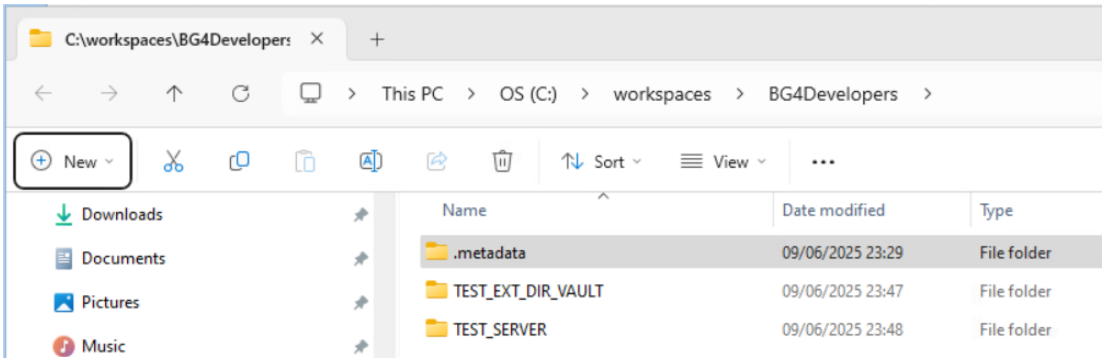
3.	On successful start of the local Integration Server you will see a message similar to the picture below. **<Before>** dismissing the message, you can click the hyperlink and the console.log file will open behind the Success window. Click the **OK** button to dismiss the window: 
4.	The console.log file will provide feedback about the start sequence of the integration server:  Close the **console.log** file.

5.	A connection to your new local Integration Server will appear in the **Integration Explorer** Window **(D)** – the green arrow (pointing upwards) to the left of the server name indicates that the server is running:  The local Integration Server should be started. If the server isn't showing as started, try right-clicking TEST_SERVER and refreshing. If this doesn't cause it to show as started, there is likely to be a problem with the default ports configured in your server.conf.yaml. If you cannot get the server started, seek help from your instructor.
6.	In the Application Development window **(A)**, the configuration associated with your local Integration Server can be found under Independent Resources: 

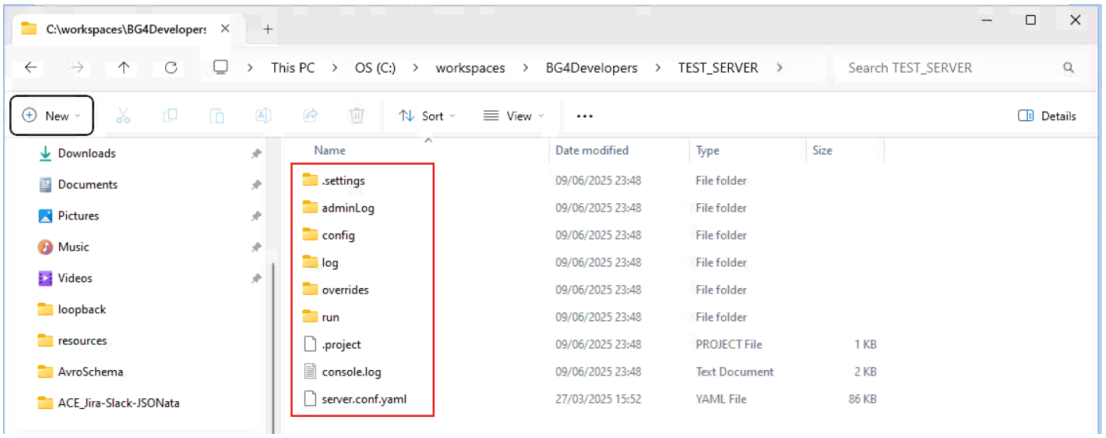
7. The configuration for a local integration server is stored directly in your App Connect Enterprise Toolkit workspace folder. To see the files and folders in the directory you could open Windows Explorer and navigate to the directory (`C:\workspaces\BG4Developers\TEST_SERVER`) but another easier way is to click the icon highlighted below in the ACE Toolkit task bar:



Windows Explorer will open at the root of the Eclipse workspace:

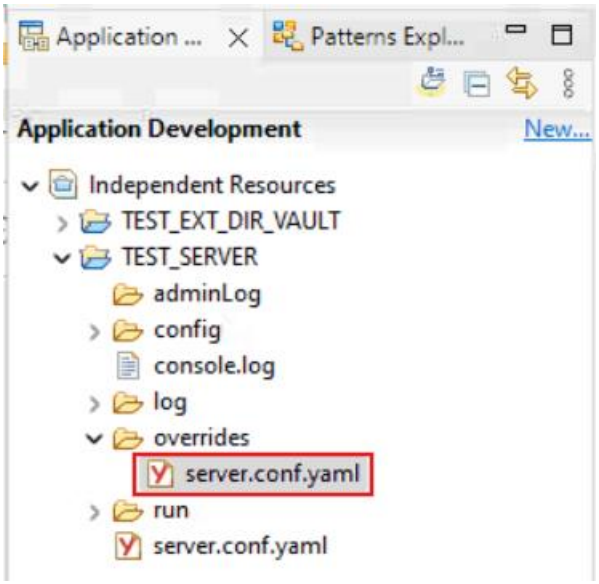
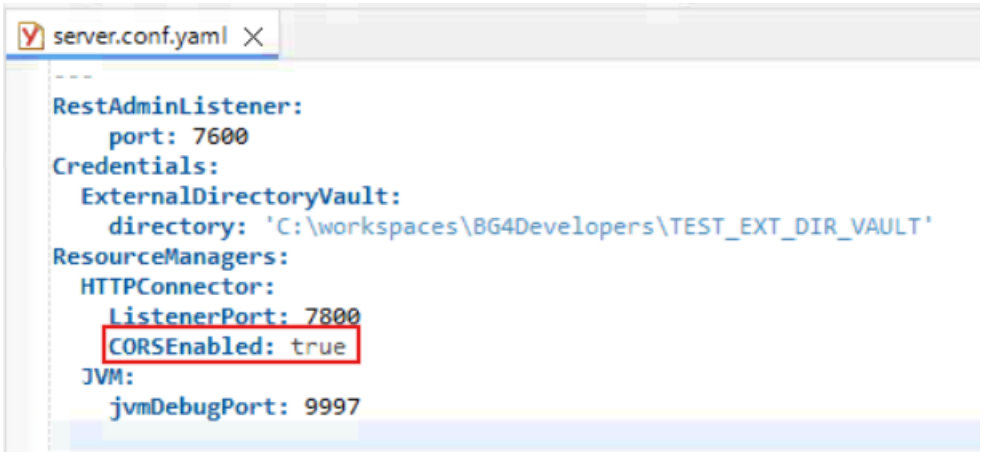


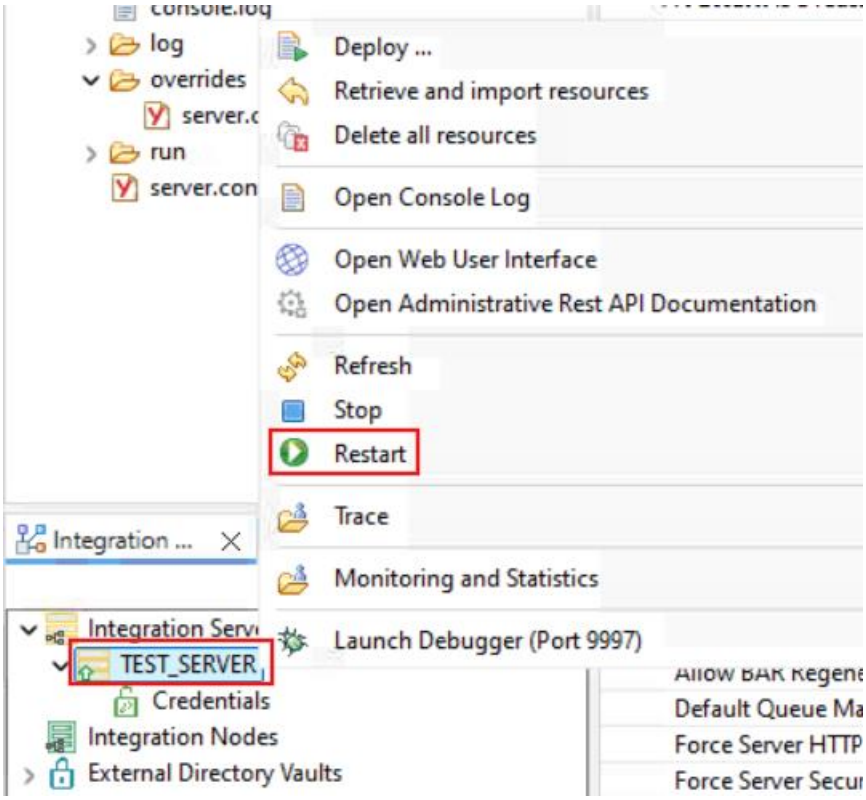
Descending into the **TEST_SERVER** directory will show the files underpinning the runtime integration server which we just created:



3.1.2 Configure CORSEnabled

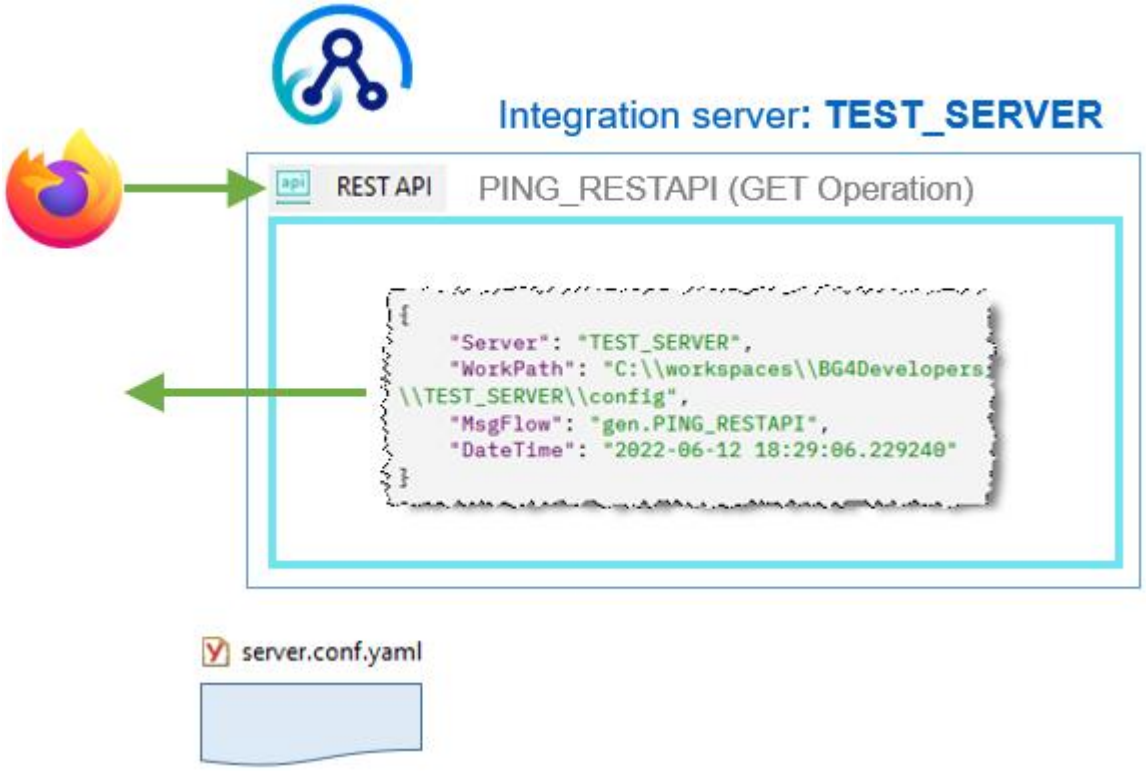
In this lab you will use the ACE Admin WebUI to test REST APIs that you create and run in the runtime server. In order to test these REST APIs, the runtime server needs to be configured with `CORSEnabled=true` for the HTTP port that the REST API runs on (*note the default value is false*).

1.	In the Application Development window **(A)**, expand the configuration associated with your local Integration Server (**Independent Resources> TEST_SERVER**). You will notice that there are two separate files with the name `server.conf.yaml`, one in the main working directory of TEST_SERVER and another in the **overrides** subdirectory. As the subdirectory name suggests, values in that copy of `server.conf.yaml` will take precedence. Open (double click) the `server.conf.yaml` inside the **overrides** subdirectory: 
2.	You will find that the override server.conf.yaml file already contains values for the ports (for REST administration, for the HTTPConnector and for the JVM Debug Port) which were selected when the server was created. In the HTTPConnector section, add the `CORSEnabled` property and set its value to `true` as shown highlighted below: 
3.	Save the `server.conf.yaml` file and close the editor.

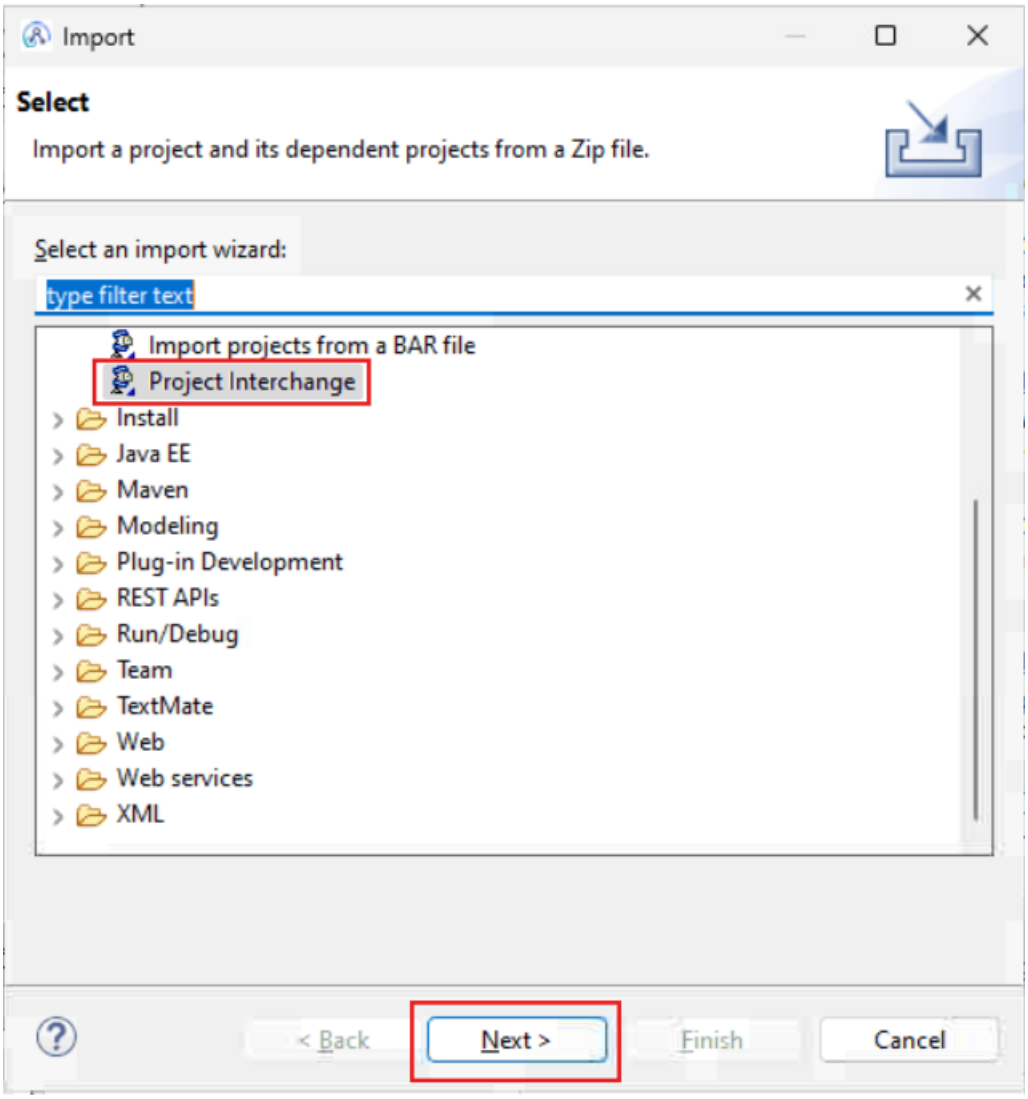
4.	To make this change effective on **TEST_SERVER** you will need to restart the integration server. In the Integration Explorer, right click on **TEST_SERVER** and select the option to **Restart**. If you prefer you could separately select the actions to Stop and then Start: 

4. Running a Basic REST API

In this section you will import a Project Interchange (PI) file (this is a way of sharing development resources between different users of the ACE Toolkit). The PI file contains a very simple REST API. You will see how you can test this REST API using the ACE Admin WebUI and using the ACE Toolkit “debugger”.

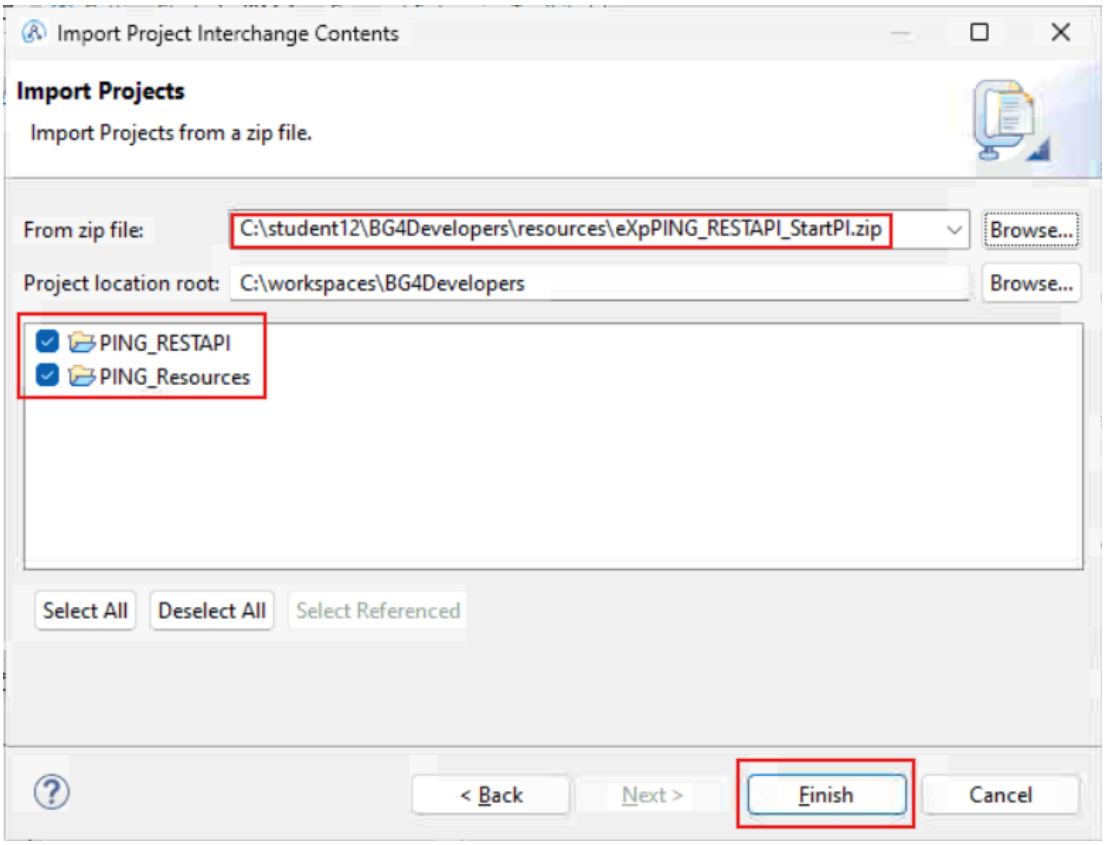


4.1 Import Basic REST API

1.	Right click on the background of the Application Development window **(A)**, in the Integration Toolkit, select **Import** and click **Next**> 

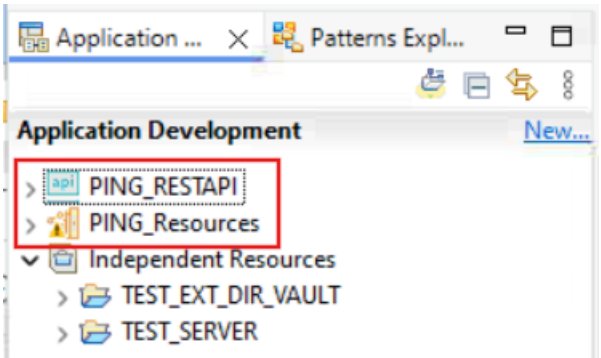
2. Import the project interchange file **eXpPING_RESTAPI_StartPI.zip** located in `C:\student12\BG4Developers\resources`

   The PI file contains 2 projects:

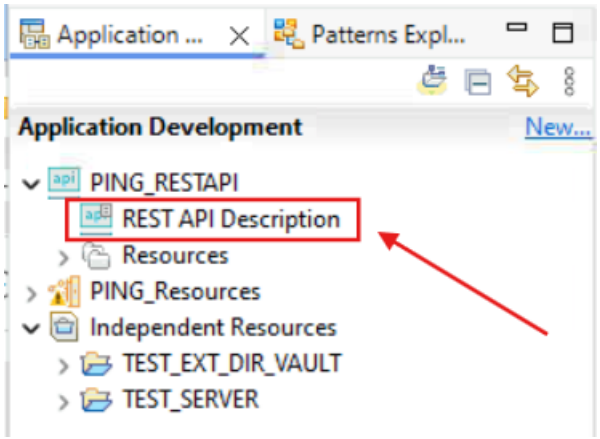
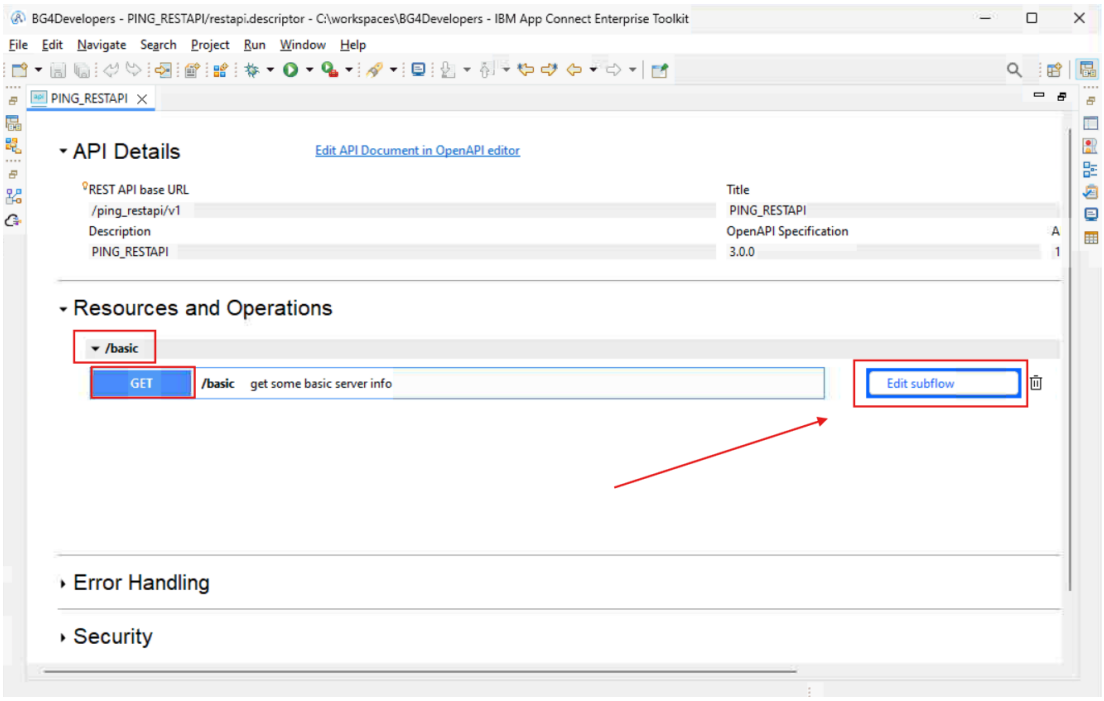


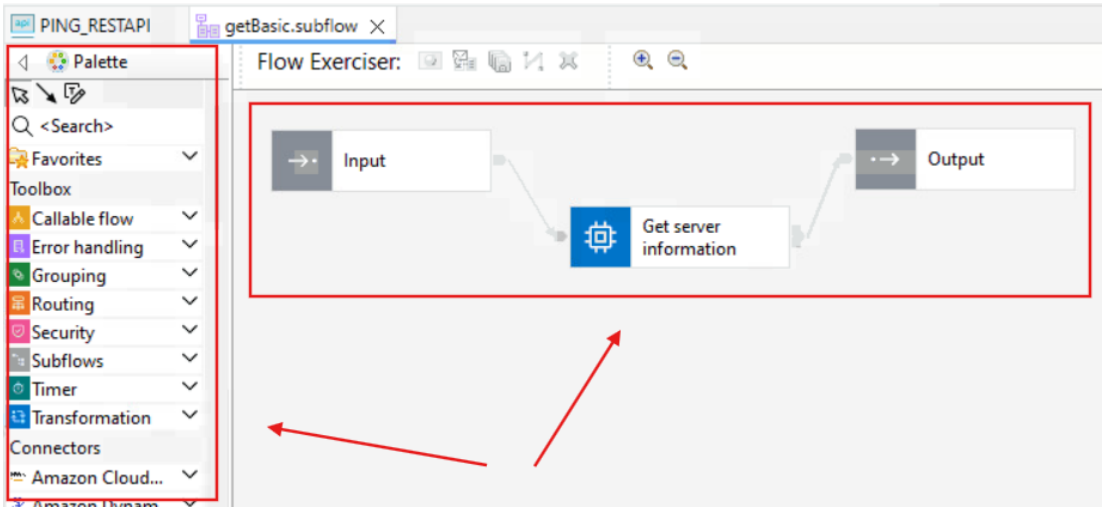
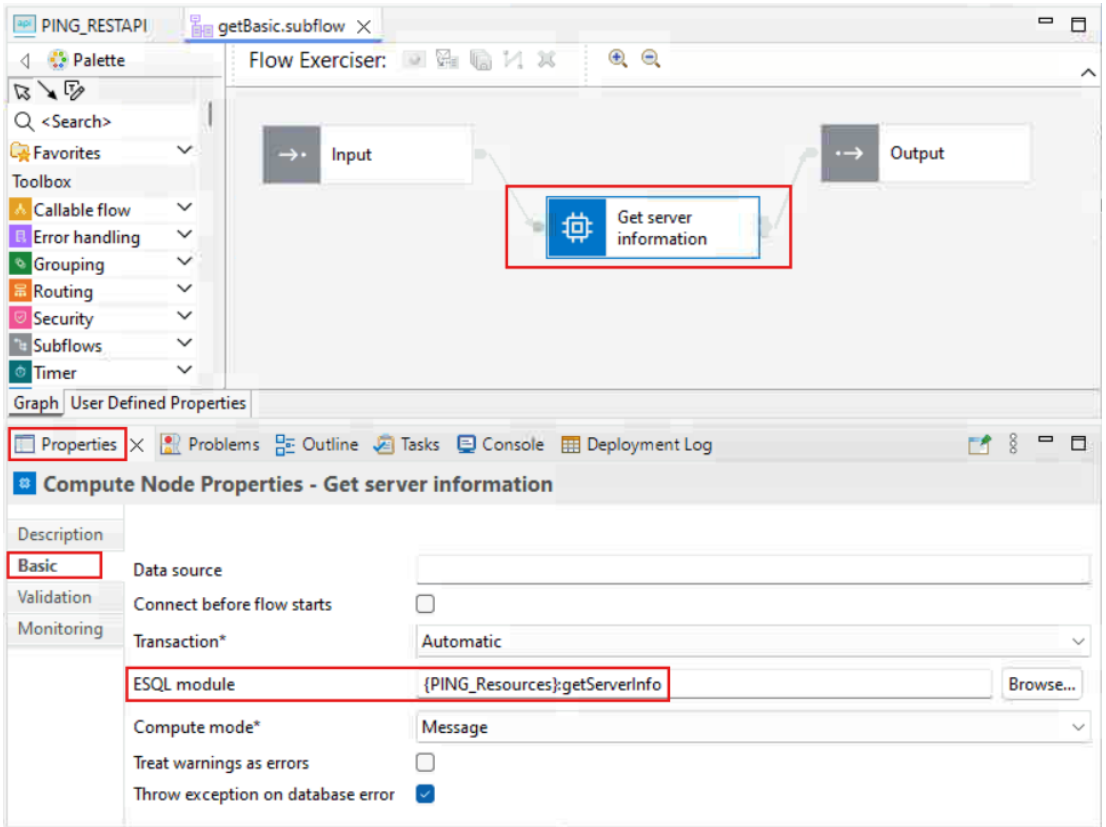
3. The PI will make the following available in your workspace:

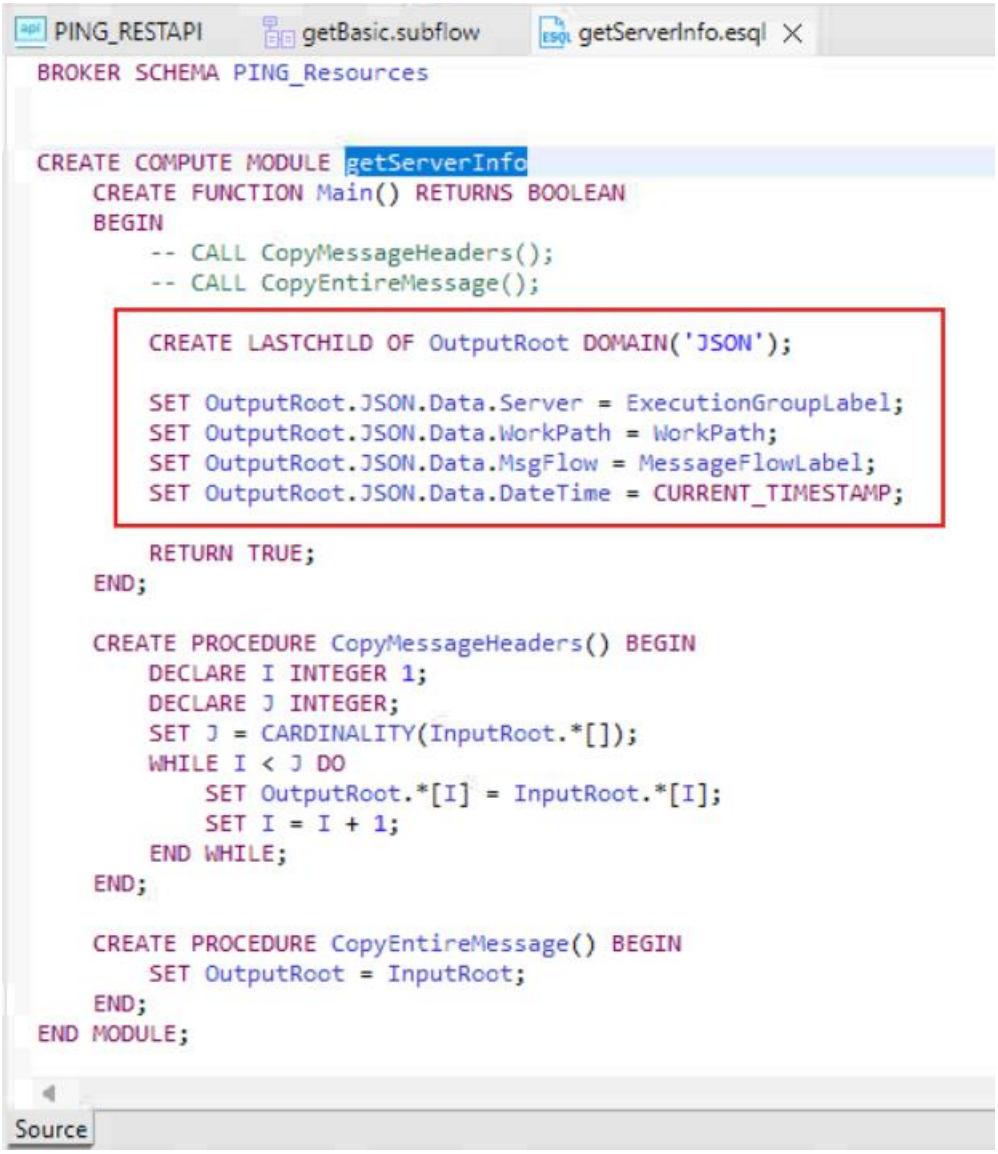
   a) An OpenAPI specification 3.0 REST API called **PING_RESTAPI**.
   b) A shared library (required by the REST API) called **PING_Resources**.



4.2 Explore PING_RESTAPI

1.	In the Application Development window, expand **PING_RESTAPI** and double click on the **REST API Description**: 
2.	The REST API Description will open. Expand Resources and Operation and note that REST API has one path (/basic) and one operation (GET). Click the "**Edit subflow**" button *(this indicates that the operation has an implementation)*: 

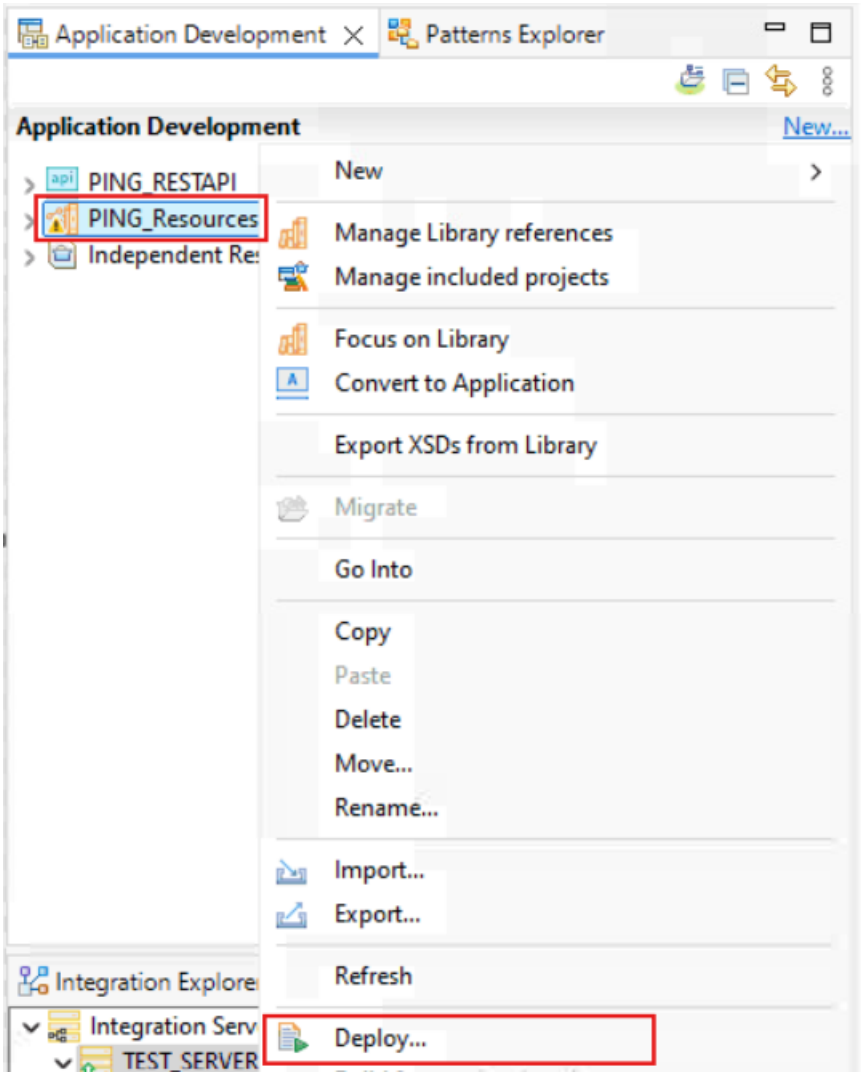
3.	The **getBasic.subflow** will open in the subflow editor. The editor has a “Palette” of nodes on the left and a “canvas” on the right. The canvas shows the implementation of the operation, in this case a simple “Compute” node called **Get server information** 
4.	***Single* click** on the Compute node **Get server information**, to show the configuration of the node (this is shown in node properties): Note: you can see from the Basic tab that the code for the ESQL module is stored in the **PING_Resources** shared library: 

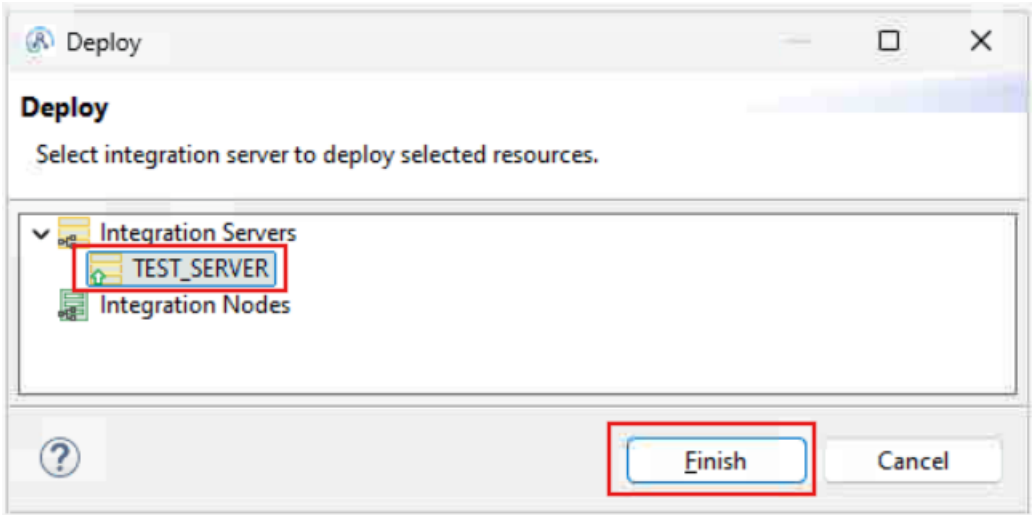
5.	Double click on the Compute node to open the ESQL code that will run in the node. This code obtains some server information and adds it to an (output) JSON message – which will then be returned to the caller: 
6.	Close the ESQL editor without making any changes. Leave the editor showing getBasic.subflow open as you will come to that in a section below.

```
PING_RESTAPI   getBasic.subflow   getServerInfo.esql
BROKER SCHEMA PING_Resources


CREATE COMPUTE MODULE getServerInfo
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        -- CALL CopyMessageHeaders();
        -- CALL CopyEntireMessage();

        CREATE LASTCHILD OF OutputRoot DOMAIN('JSON');

        SET OutputRoot.JSON.Data.Server = ExecutionGroupLabel;
        SET OutputRoot.JSON.Data.WorkPath = WorkPath;
        SET OutputRoot.JSON.Data.MsgFlow = MessageFlowLabel;
        SET OutputRoot.JSON.Data.DateTime = CURRENT_TIMESTAMP;

        RETURN TRUE;
    END;

    CREATE PROCEDURE CopyMessageHeaders() BEGIN
        DECLARE I INTEGER 1;
        DECLARE J INTEGER;
        SET J = CARDINALITY(InputRoot.*[]);
        WHILE I < J DO
            SET OutputRoot.*[I] = InputRoot.*[I];
            SET I = I + 1;
        END WHILE;
    END;

    CREATE PROCEDURE CopyEntireMessage() BEGIN
        SET OutputRoot = InputRoot;
    END;
END MODULE;

Source
```

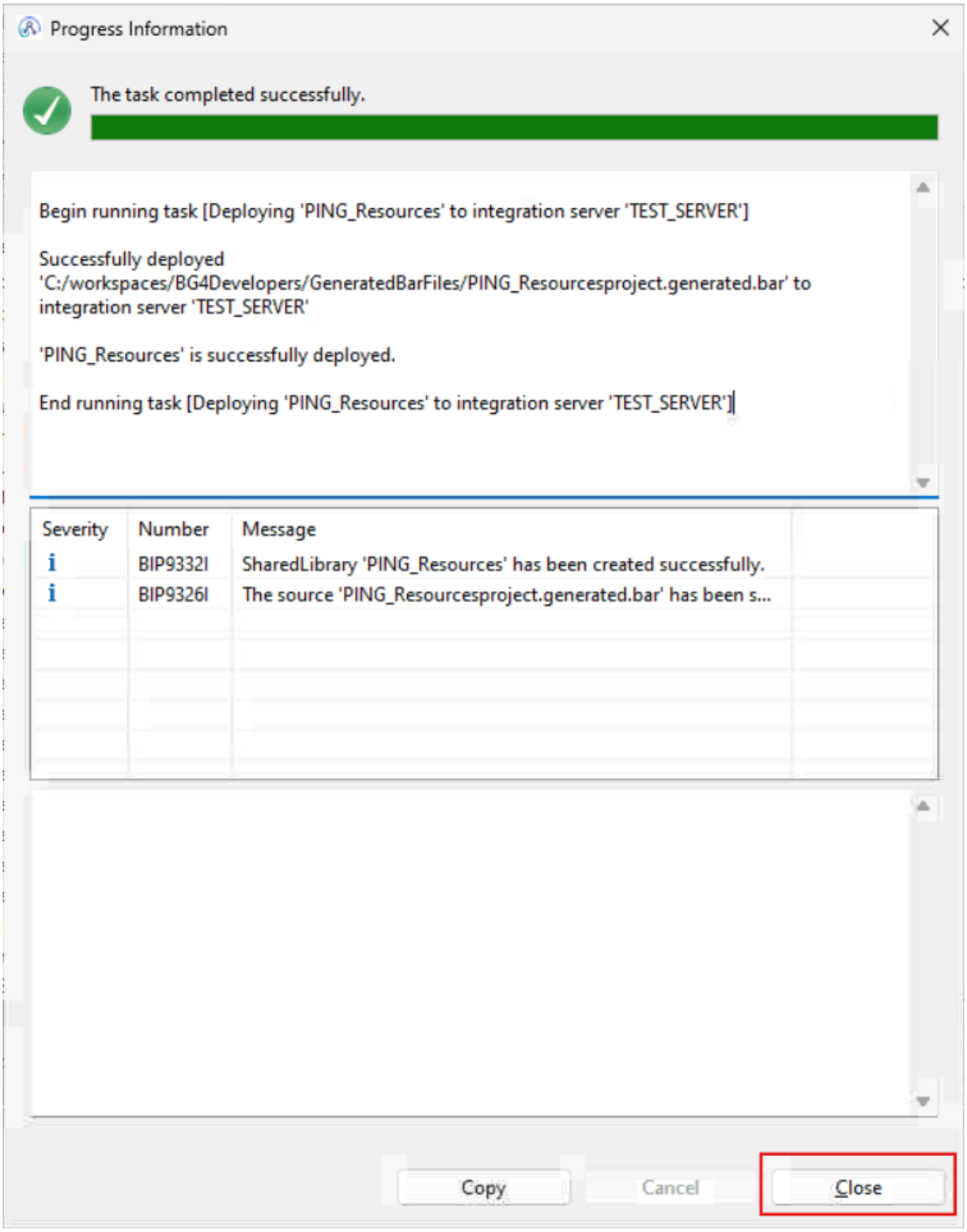
4.3 Deploy Resources

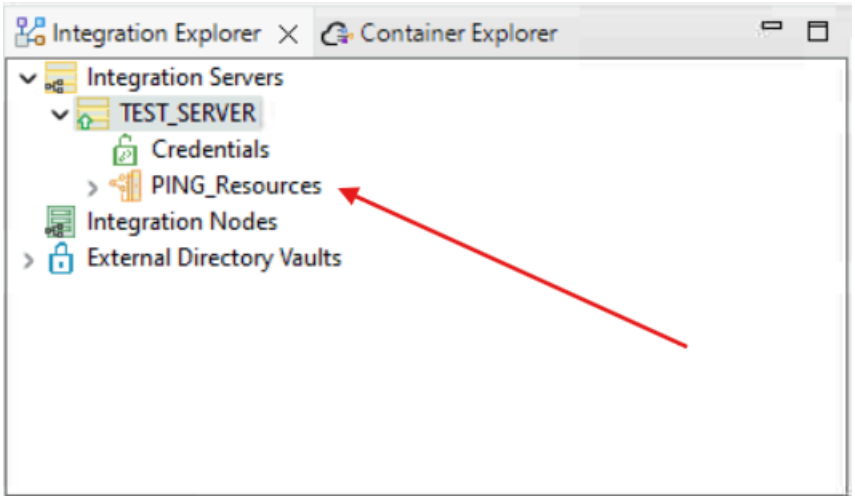
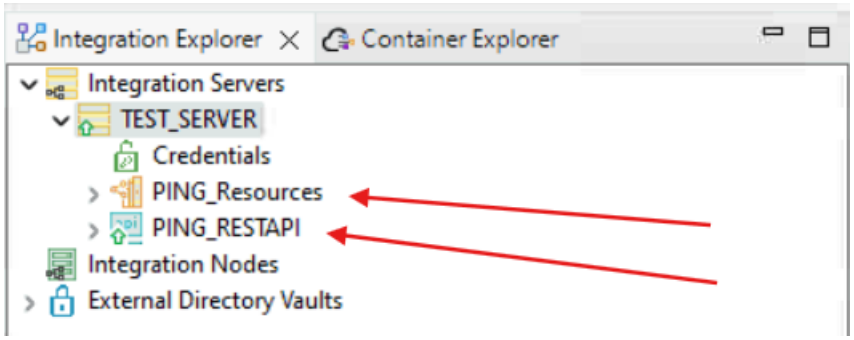
In this next section you will test the **PING_RESTAPI** on the Local **TEST_SERVER** you created earlier.

1.	In the Application Development window, right click on **PING_Resources** and select Deploy. 

2.	When prompted to select an integration server to deploy to, select **TEST_SERVER**, and click **Finish**: 

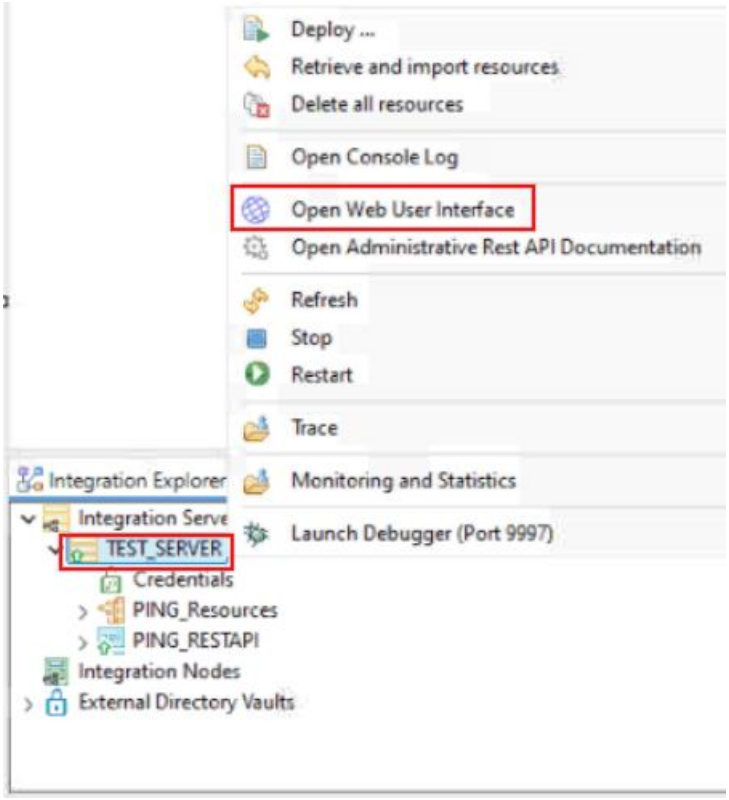
3.	Ensure the Shared Library deploys successfully *(Click Close on the progress information window)*:

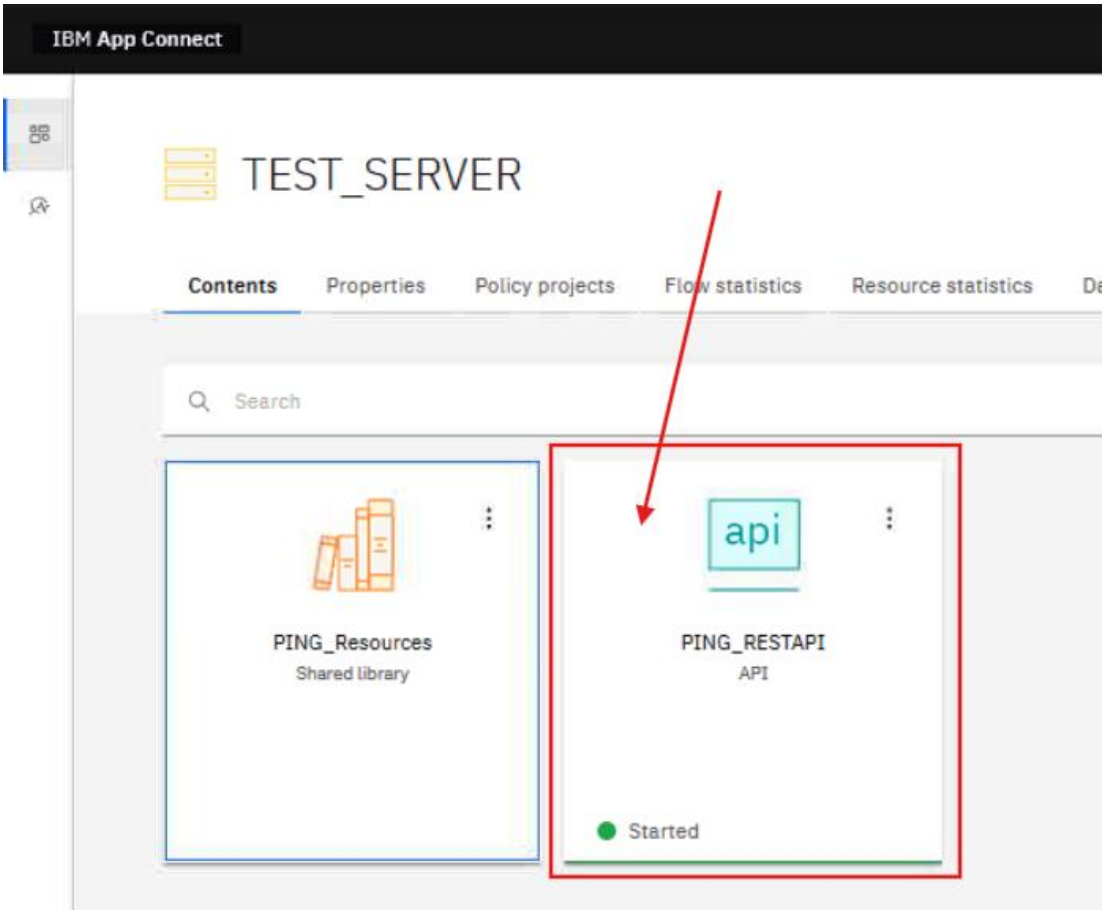


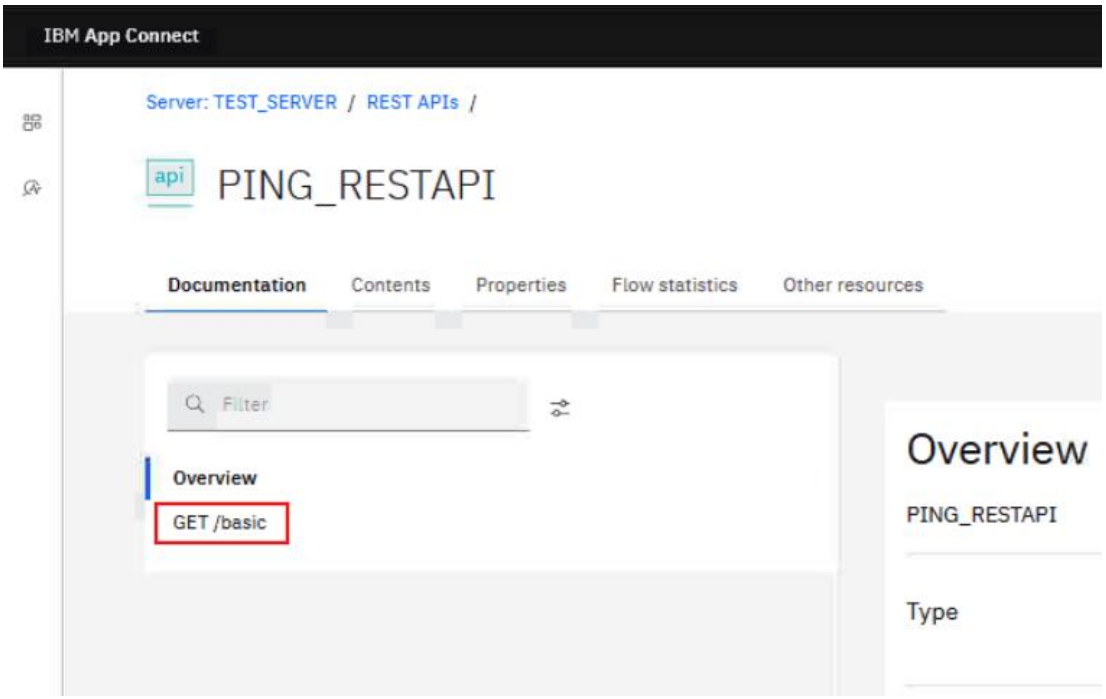
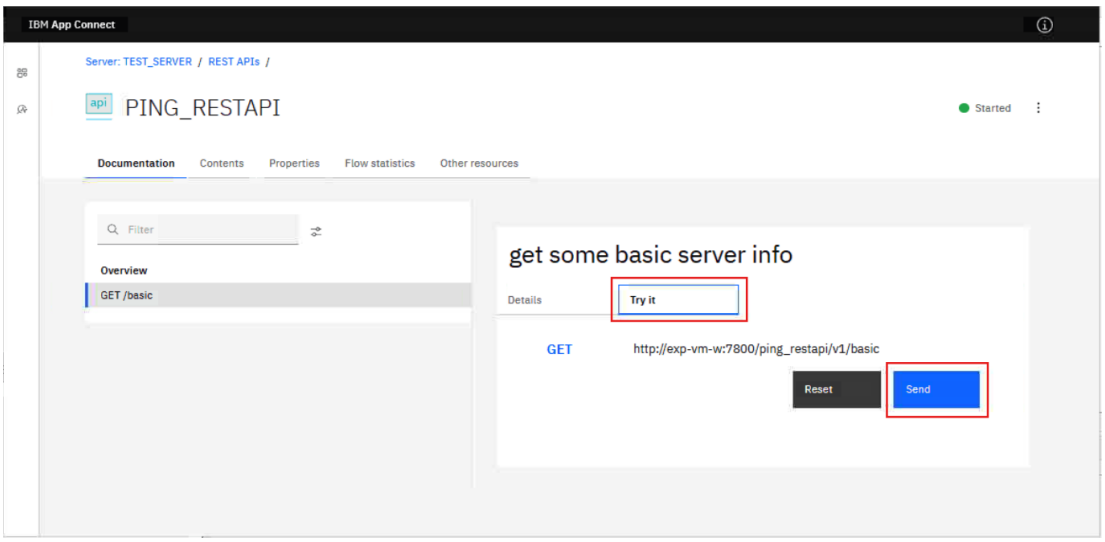
4.	In the Integration Explorer window (window D), note that the Shared library now shows as being deployed to **TEST_SERVER**: 
5.	Repeat the above steps and deploy **PING_RESTAPI** (right click on the REST API name to select **Deploy**).
6.	When both are deployed successfully, they will both appear in the Integration Explorer window under **TEST_SERVER**: 

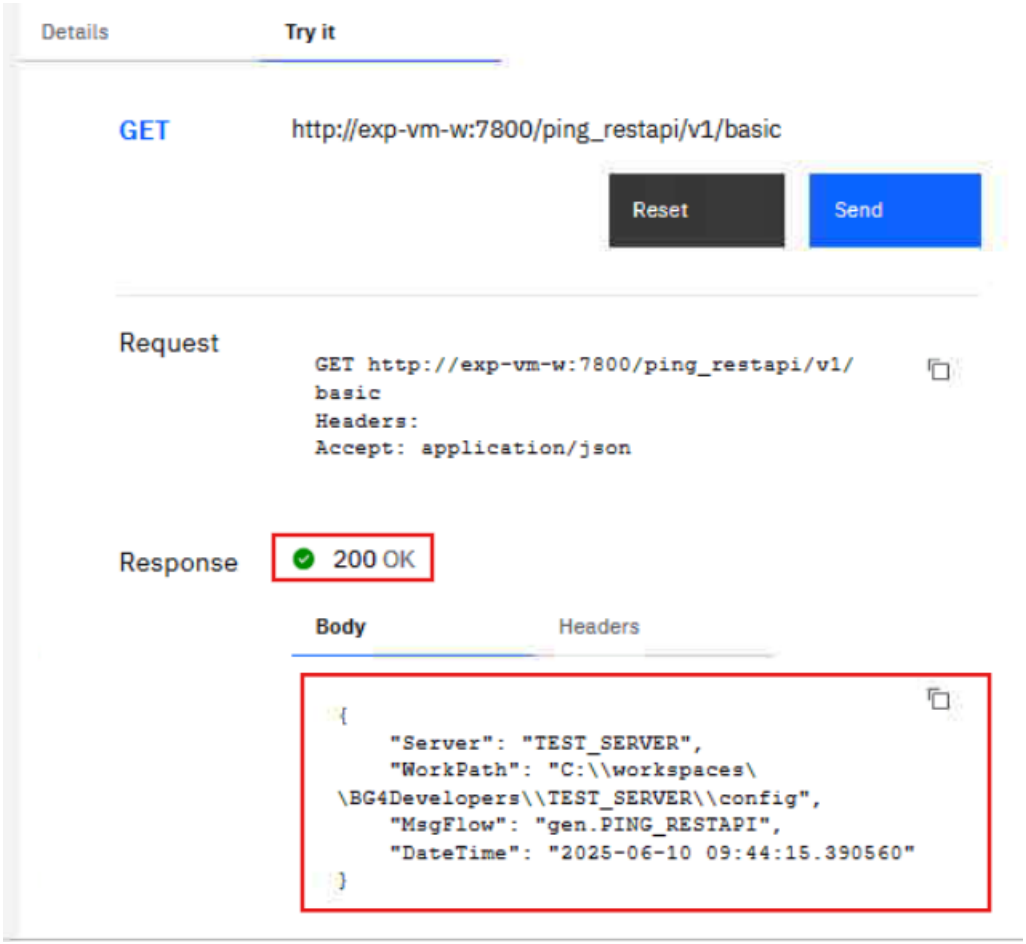
4.4 Test using ACE Admin Web UI

1.	In the Integration Explorer window right click on TEST_SERVER and select "**Open Web User Interface**":



2.	A browser window will open at http://exp-vm-w:7600 (7600 is the default admin port number for an integration server) Click on the PING_RESTAPI tile: 

3.	Click on the GET /basic operation (under overview): 
4.	In the GET /basic operation select the “Try it” tab and click the send button: 

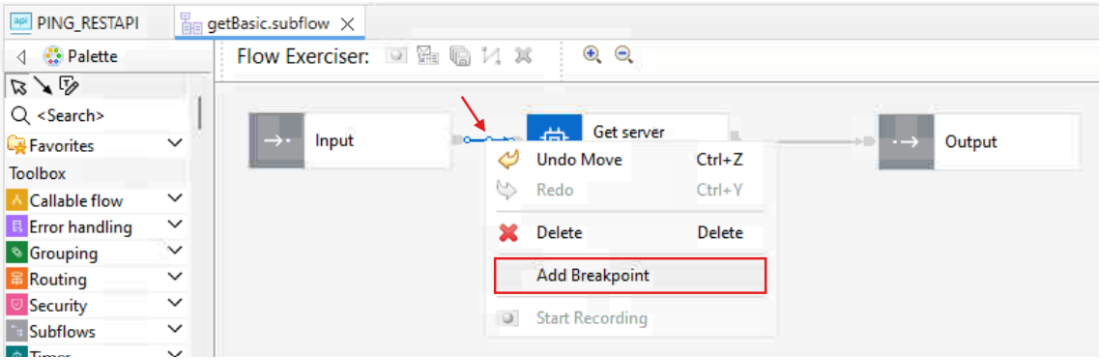
5.	After a few seconds you will receive a **Response** code **200 OK**. The body of the response will show the details of the integration server obtained by the ESQL compute node: 
6.	Keep this browser window open as you will use it in the next section when using the Graphical debugger.

4.5 Using the ACE Toolkit Flow Debugger

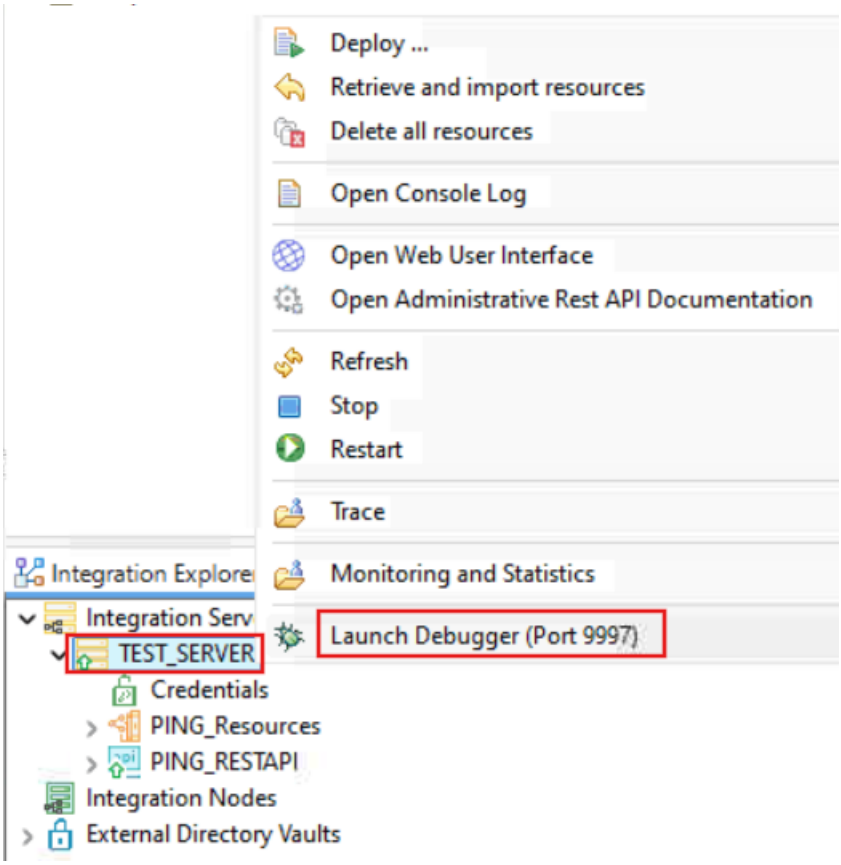
In this next section you will see how to use the ACE (graphical) flow debugger tool with the simple REST API.

You will add a Break point to the subflow used to implement the GET operation and start the Flow Debugger on the independent integration server TEST_SERVER.

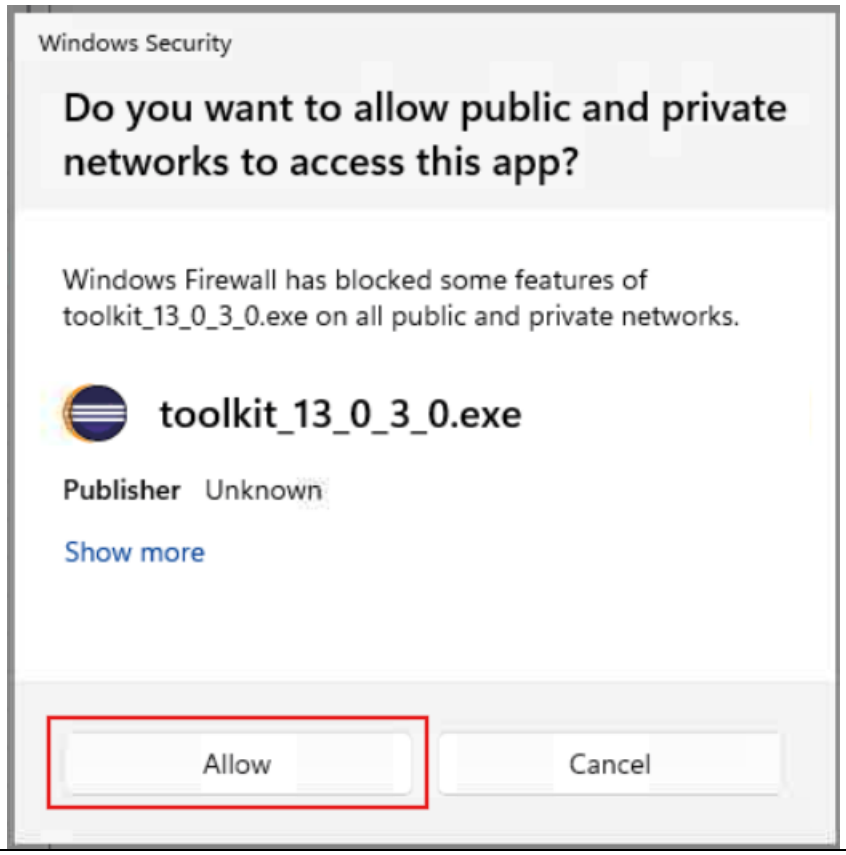
4.5.1 Add Breakpoints and Launch Flow Debugger

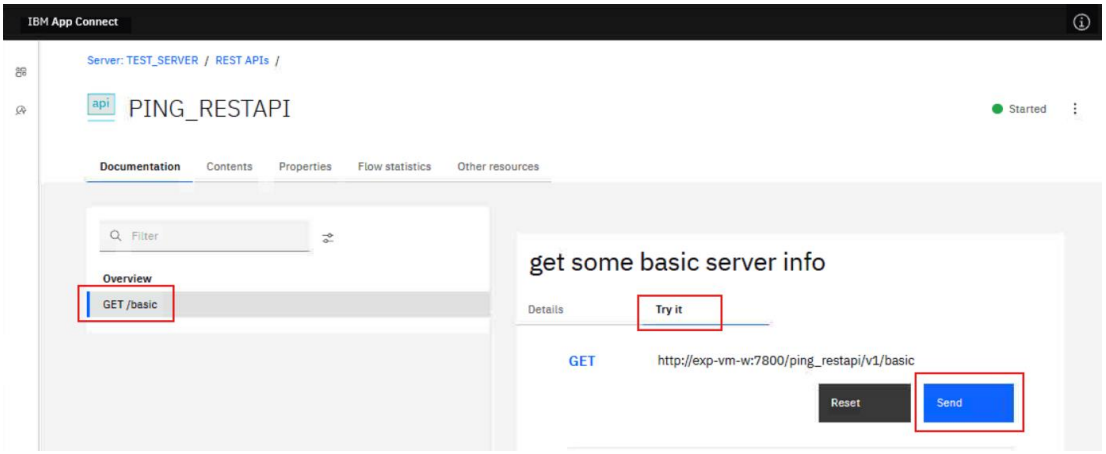
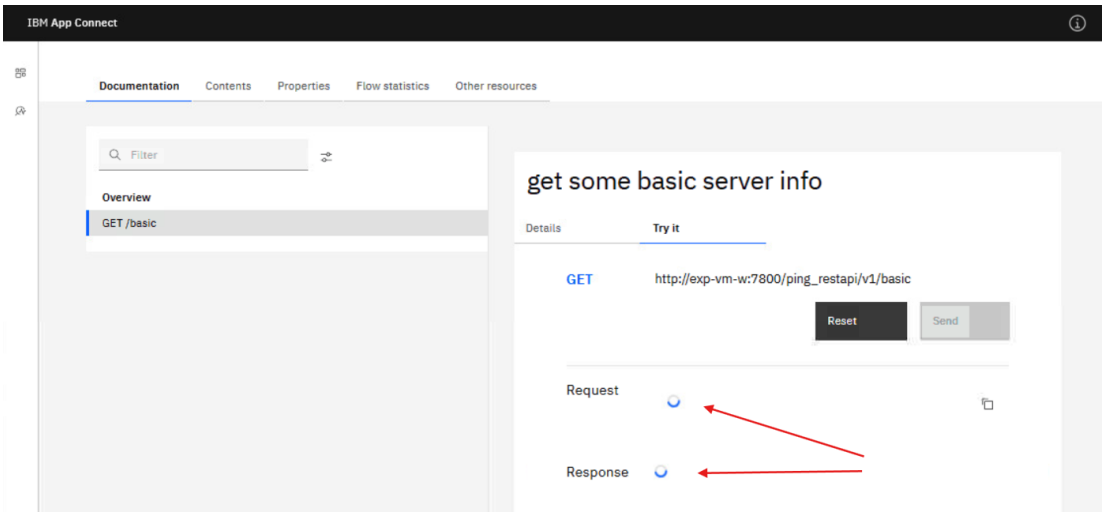
1.	Switch back to the ACE Toolkit and focus on getBasic.subflow (you left this editor open in an earlier section). Right click on the connector between the Input node and the Get server information compute node, Select “Add Breakpoint” from the list of options: 

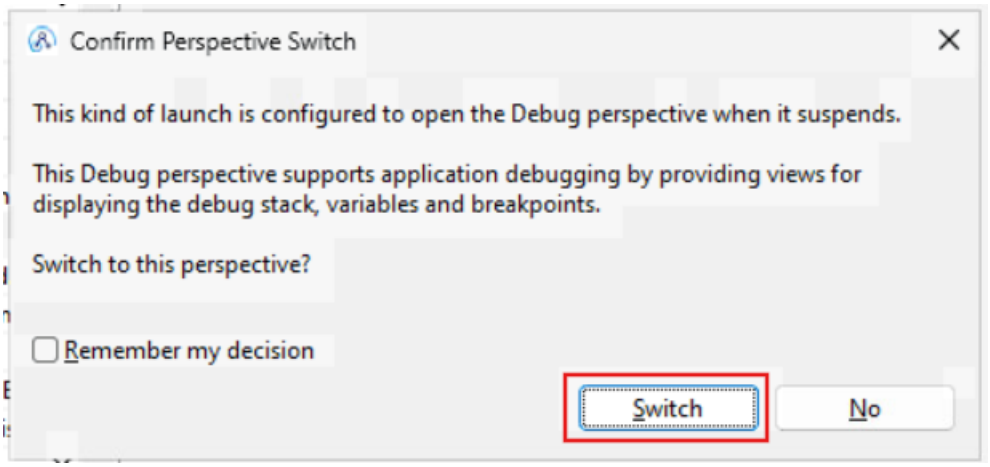
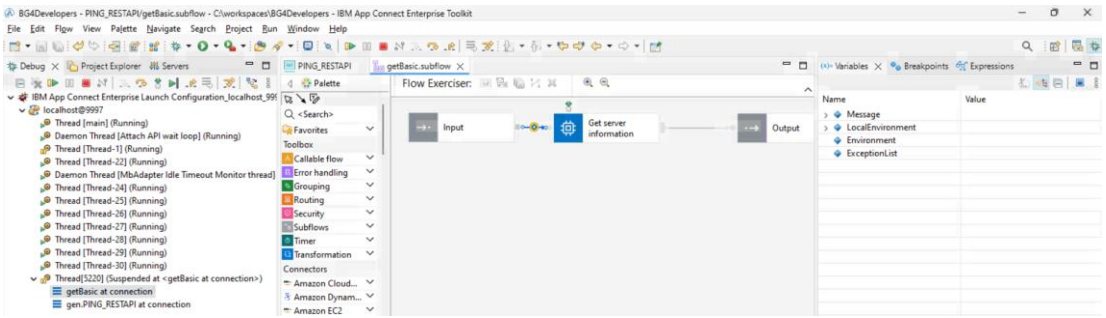
2. In the Integration Explorer window, right click on the TEST_SERVER and select “Launch Debugger”

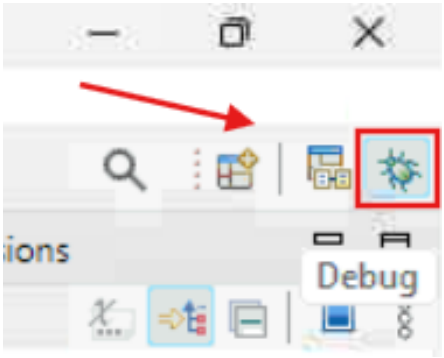
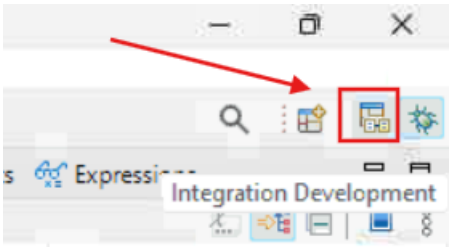


You may also see the following warning. If so, click **Allow**:

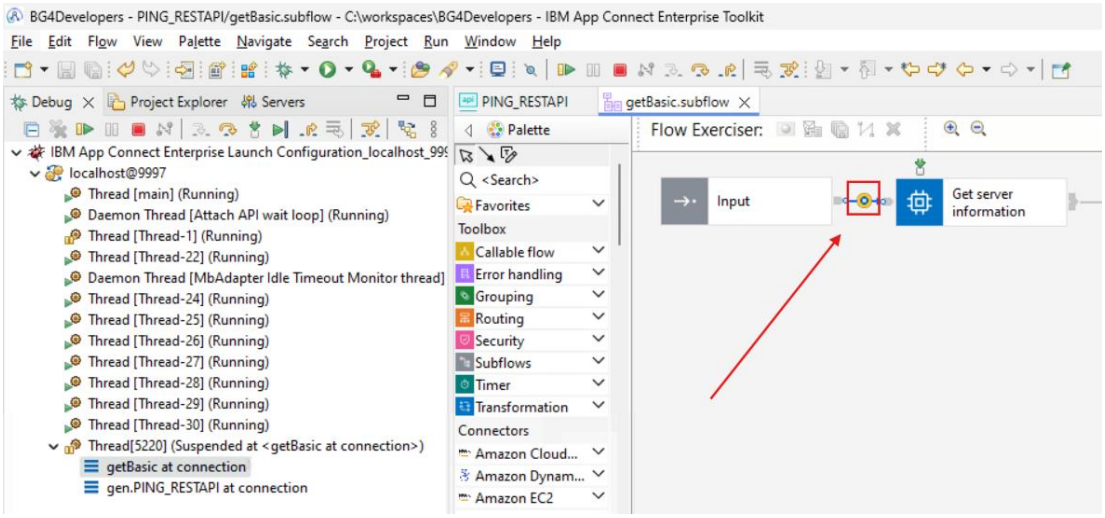
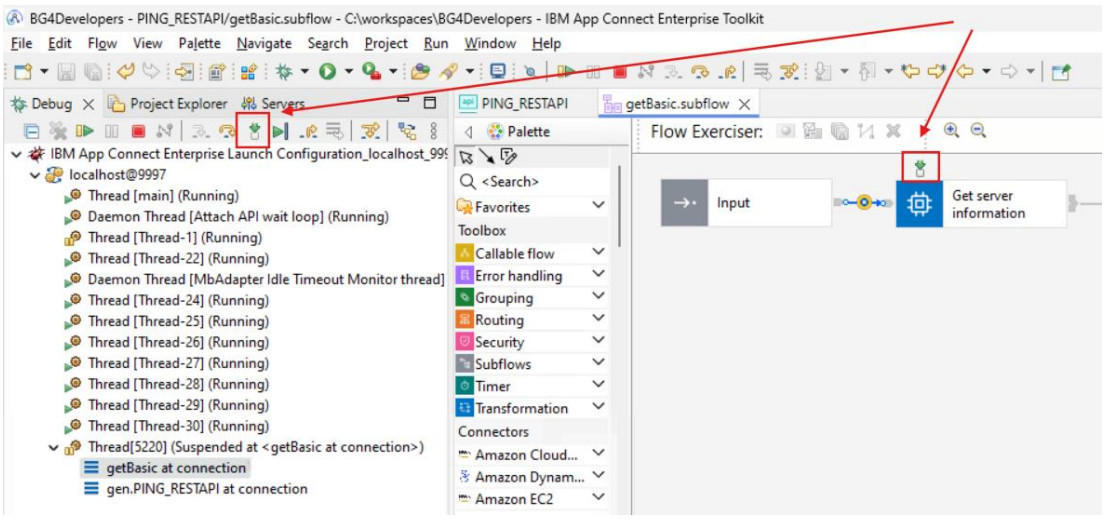


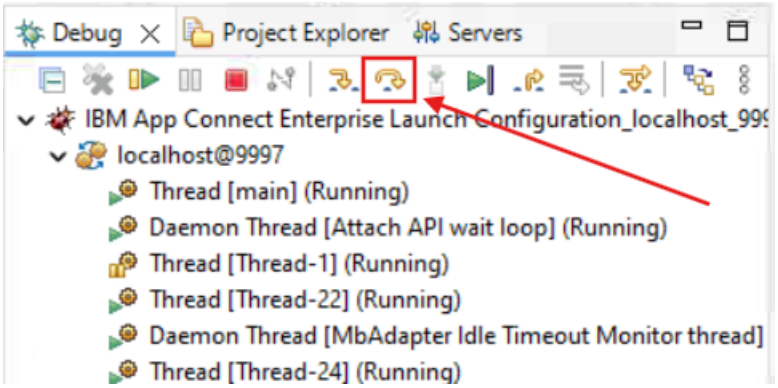
	(Note: Once you have attached the debugger, if you right click again on TEST_SERVER you will see the option to **Launch Debugger** will have turned into **Terminate Debugger**)
3.	Switch back to the browser where you tested the REST API using the ACE Web Admin UI. In the **GET /basic** operation (“**Try it**” tab), click the **Send** button: 
4.	This time the response will not be displayed until the debugger has Launched and you have stepped through the flow that implements the GET operation. You will see the rotating circle indicating that the response has not yet been received: 

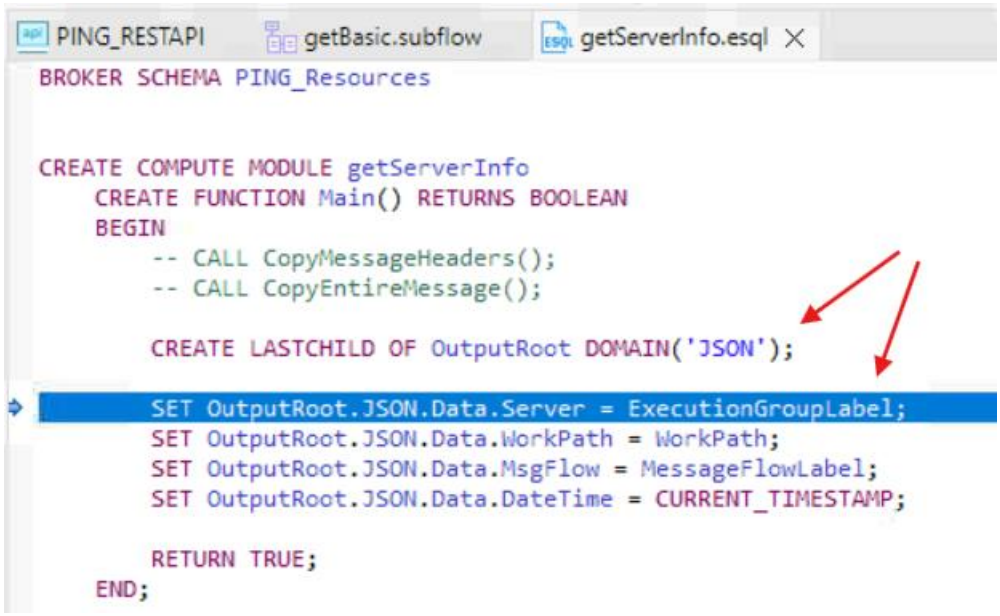
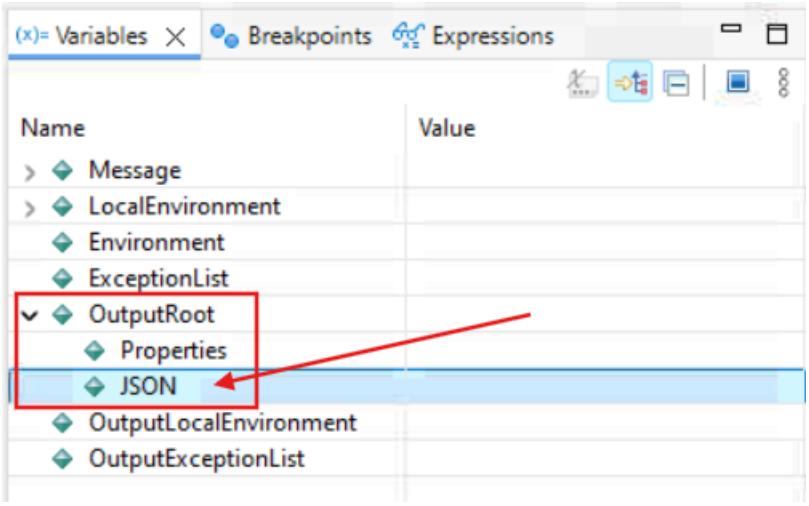
5.	Switch back to the ACE Toolkit and note the **Confirm Perspective Switch** message, click **Switch**: 
6.	The Debug “perspective” will now open:  *An Eclipse perspective “is a named collection of windows, layouts, views, toolbars, and actions in the Eclipse Workbench window”.*

7.	Note you can switch between the **Debug** and the **Integration Development** perspectives using the icons at the top right of the ACE Toolkit (hover over the icons to see the meaning of the icon):  

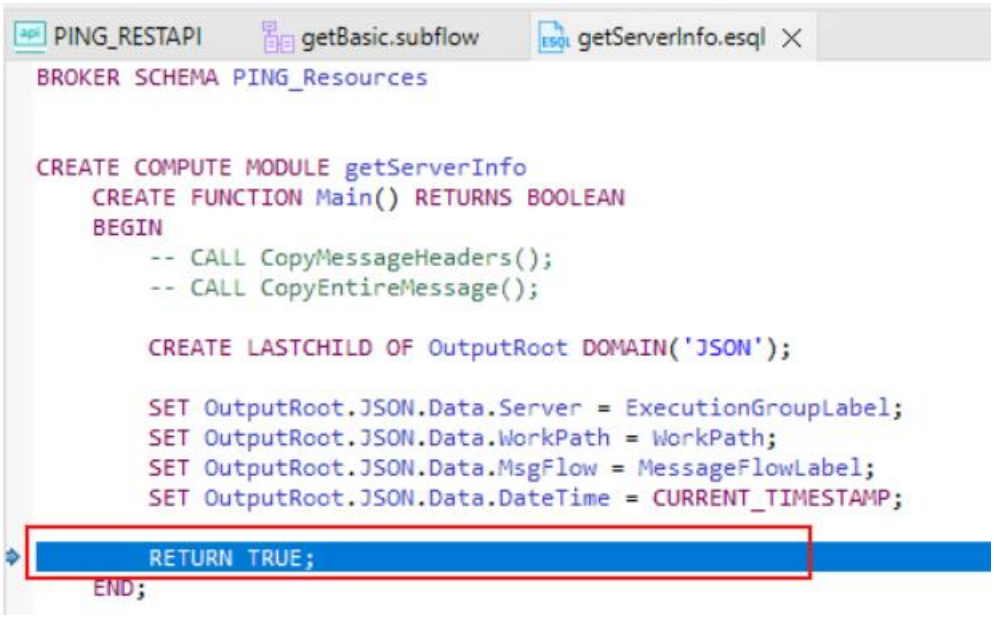
4.5.2 Step through message flow

1.	Keep in the Debug perspective and note that in the subflow editor showing getBasic.subflow you will see a (yellow) circle around the “Breakpoint” that you added earlier indicating this is where the debugger has temporarily “stopped” the ACE runtime: 
2.	The green arrow pointing down above the Compute node indicates you can step through the source code associated with the node. Click the “Step into Source” icon (on the Debug window to the left): 

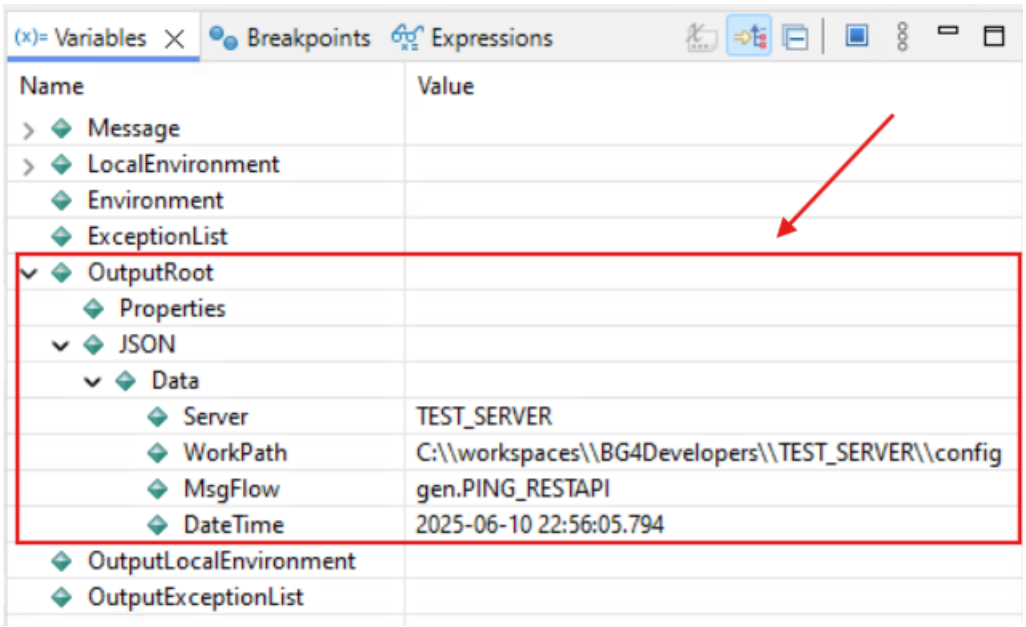
3.	The ESQL editor will open showing the source code associated with the Compute node. The next instruction that the server will execute is shown as a highlighted line of code, in the margin there is a (blue arrow): 
4.	To progress to the next instruction in the code, select the **Step Over** icon from the **Debug** window: 

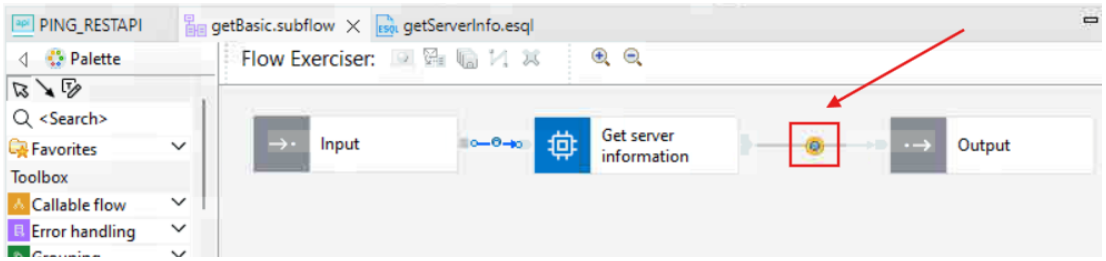
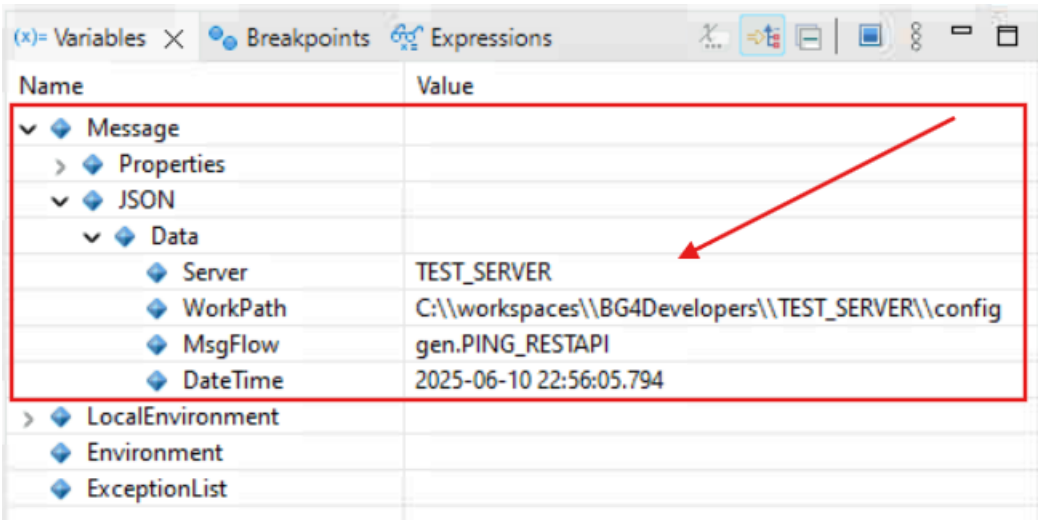
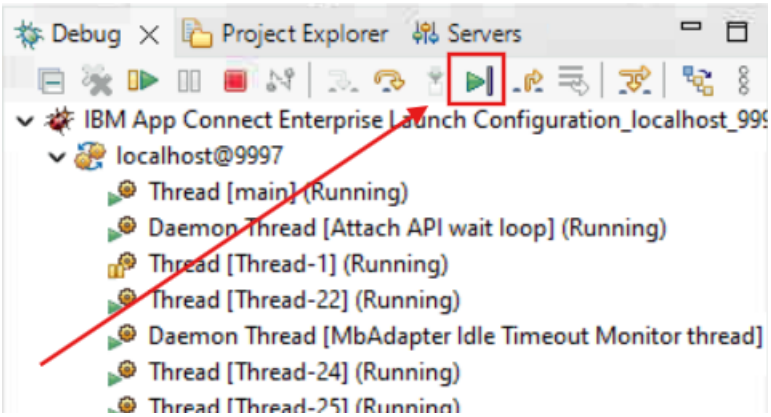
5.	The instruction "`CREATE LASTCHILD OF OutputRoot DOMAIN('JSON');` has now been executed and the next statement about to be executed is highlighted. 
6.	Note the Variable window. Expand `OutputRoot` and note that this now has a `JSON` branch: 

7. Using the **Step Over** icon, step through the source code until the "`RETURN TRUE;`" statement is about to be executed:



8. Expand OutputRoot and note that the JSON branch now has the fields: `Server`; `Workpath`; `MsgFlow`; and `DateTime` fields created and showing values:

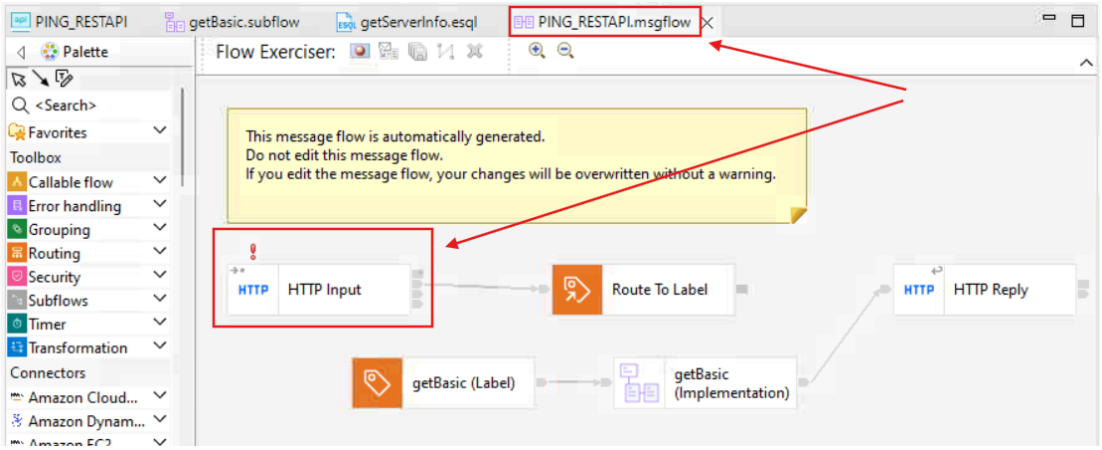


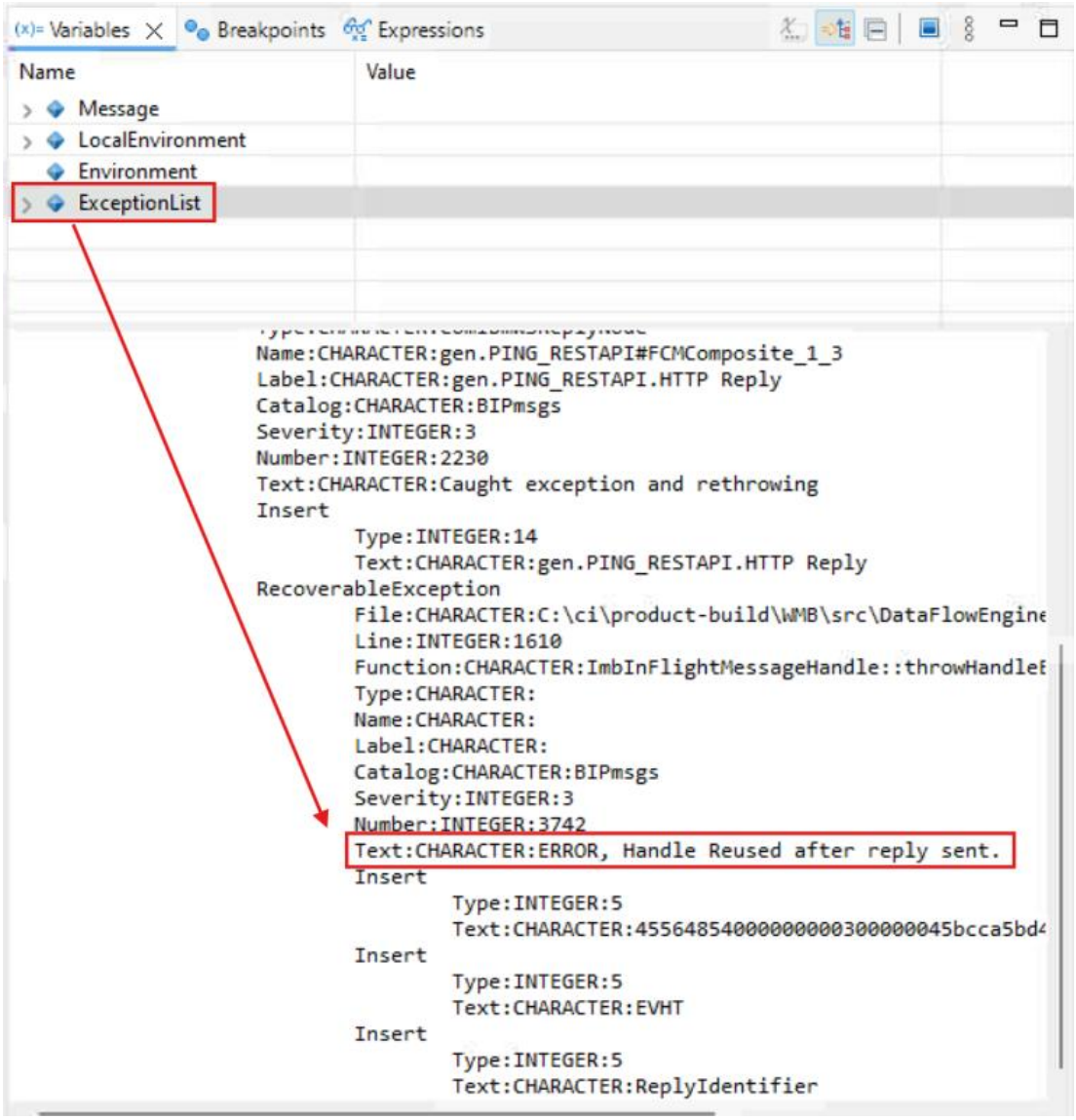
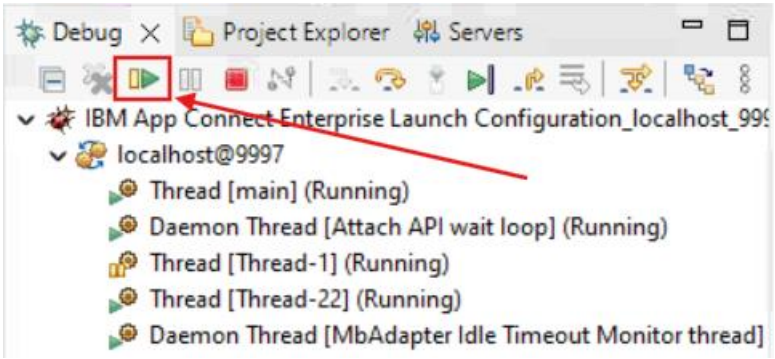
9.	Click the Step through source icon again to execute the "`RETURN TRUE`" statement. You will be automatically switched to the subflow view. The yellow circle now shows the run time position in the message flow (ie you have exited the Compute node: 
10.	Note in the Variables tab, OutputRoot is no longer shown, however "Message" now has the JSON branch that was created in the Compute node. Expand Message > JSON > Data to see the content and variables: 
11.	In the Debugger window click Run to completion icon: 

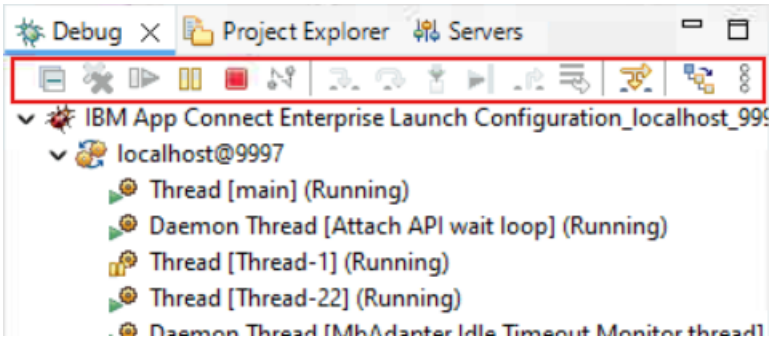
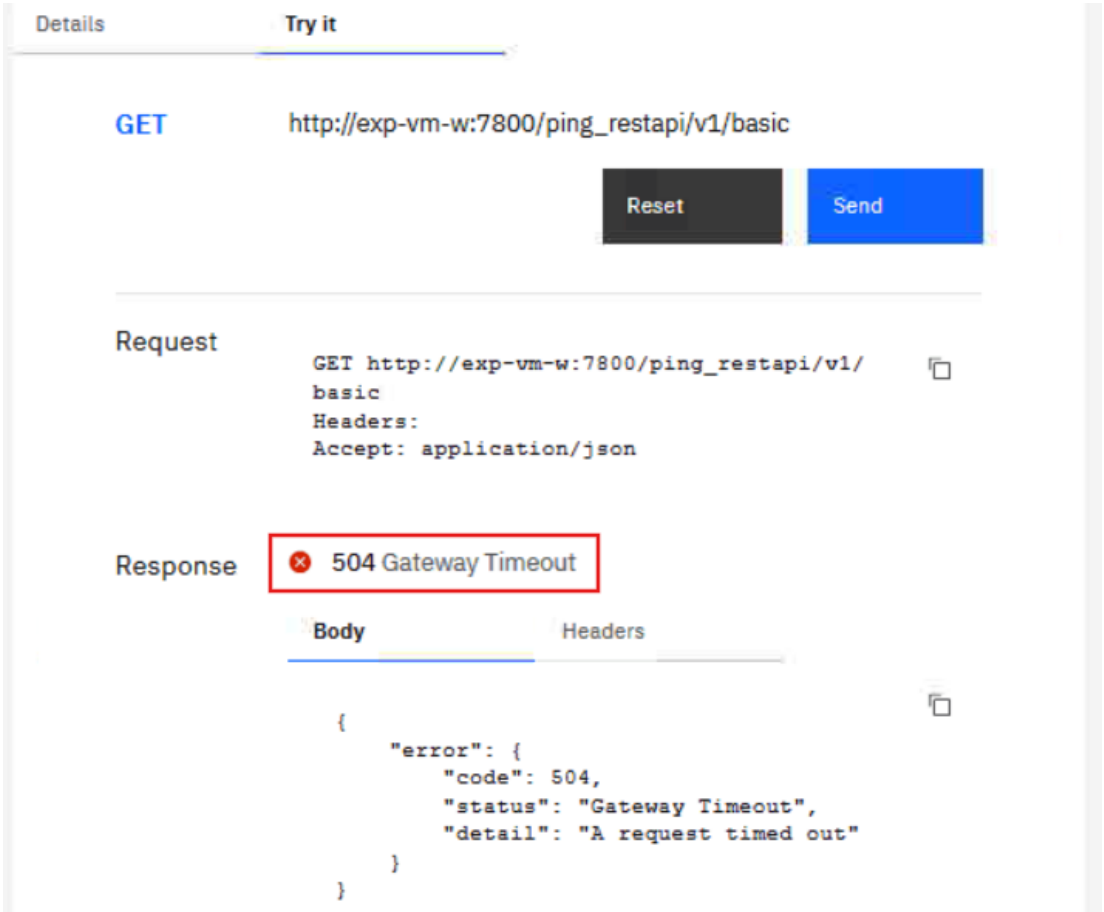
4.5.3 Likely Error Response when using Flow Debugger

When using the Flow Debugger with http-based requests normal processing can be very slow as you investigate the values in the message tree at Breakpoints. It is likely that the REST API request initiated from the ACE Web Admin UI will time out.

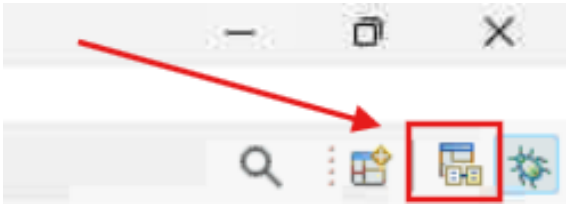
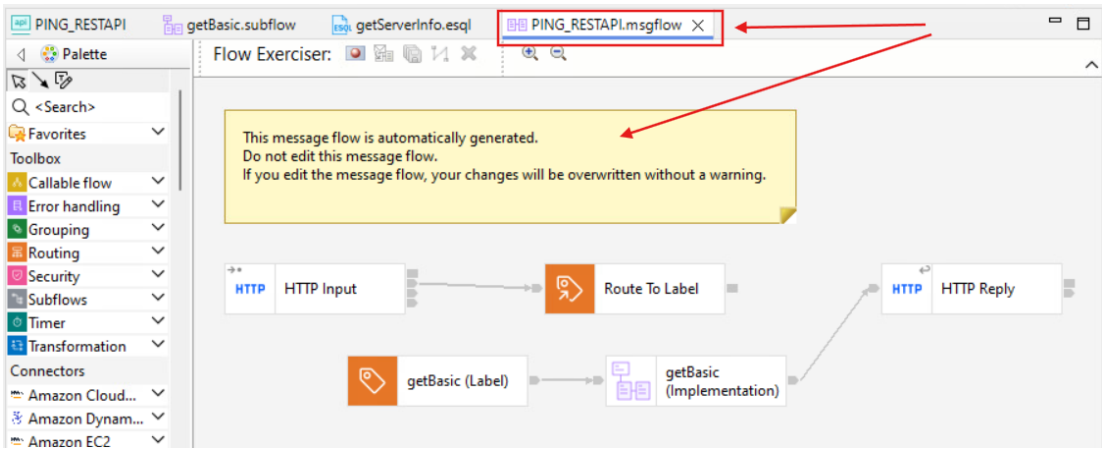
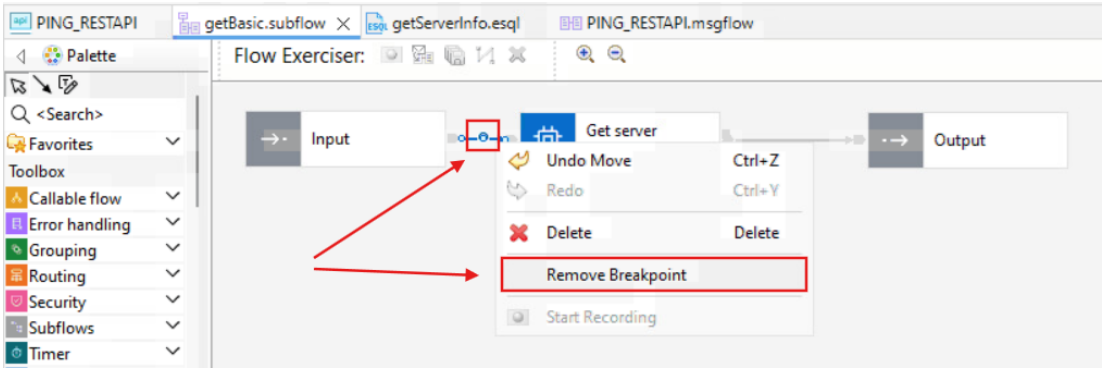
The following screen captures show an example of this and are entirely normal and expected when using the Flow Debugger.

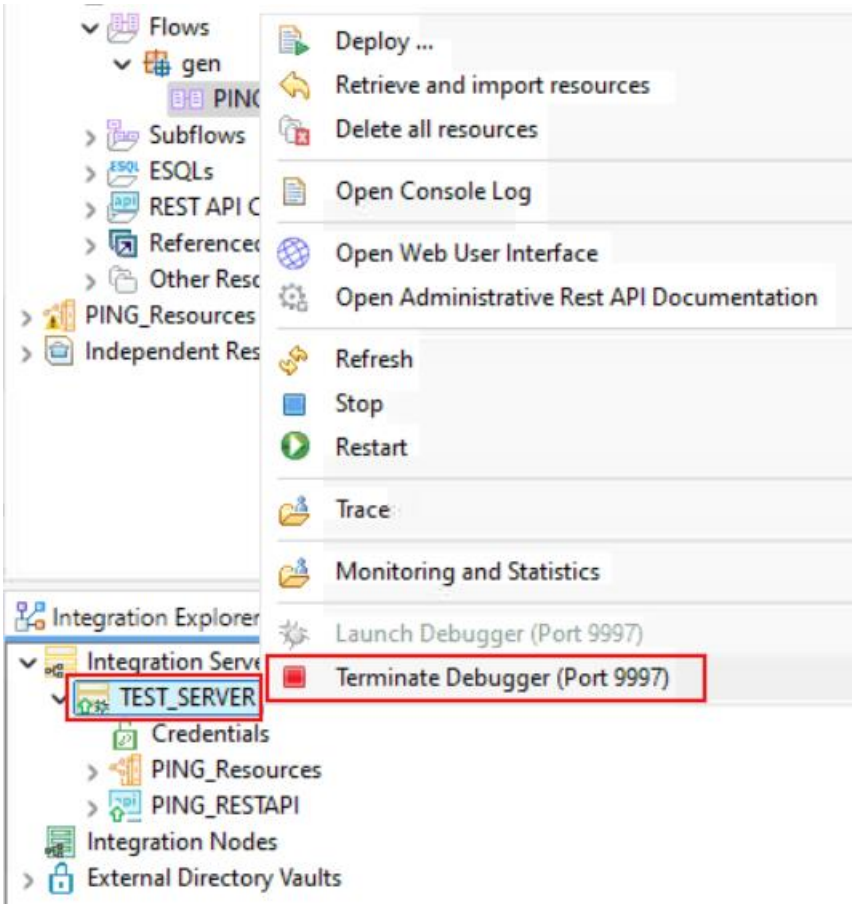
1.	If you see the following, the request has very likely timed out and caused an exception (which is likely normal if you are using the Flow Debugger). 

2.	Click on `ExceptionList` in the Variables tab. The value of `ExceptionList` can been seen in the window below, scroll to the bottom of this window to see the error which has occurred (in this case "`ERROR Handle Reused after reply sent`" (ie a timeout occurred and the Flow Debugger attempted to use the same Reply Identifier: 
3.	Click on the Resume button on the Debug window to ensure the Flow Debugger completes: 

4.	**HINT:** if the Flow Debugger does not complete (ie there are steps yet to execute), and you attempt to deploy the REST API to TEST_SERVER, for example as a result of a change to the REST API implementation, the deploy will not complete and will appear to “hang”. If this happens switch back to the Debug perspective and select the Stop icon (a red square) and retry the deploy. When the Debug window has all of its options greyed out, a deploy on TEST_SERVER should work fine: 
5.	Switch back to the browser window to see the timeout that occurred because you were using the Flow Debugger: 
6.	Keep this browser window open as you will use it later in the lab.

4.5.4 Stopping the Flow Debugger

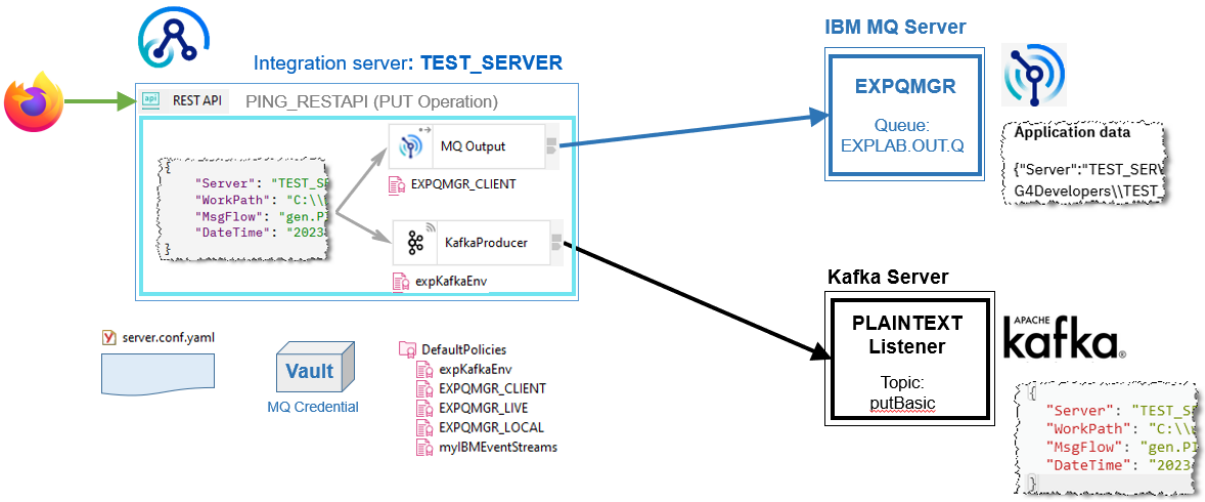
1.	In the ACE Toolkit, switch back to the **Integration Development perspective**. (On the top right of the ACE Toolkit, hover over the icons and switch back to the Integration Development perspective): 
2.	If the PING_RESTAPI.msgflow is open, note this message flow **in a REST API** *(only)* is automatically generated and any changes made will be over written when the REST API is modified (for example you add an operation to the REST API). Close the editor: 
3.	In getBasic.subflow right click on the connector with the Breakpoint configured and remove the Breakpoint: 
4.	Close the subflow editor.

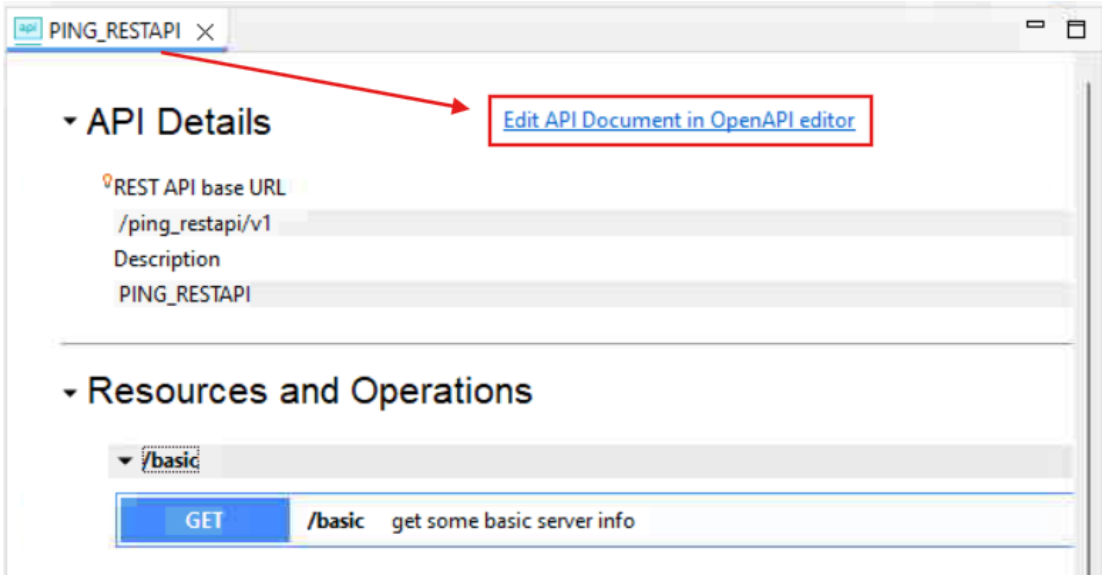
5.	In the Integration Explorer window, right click on **TEST_SERVER** and select **Terminate Debugger**: 

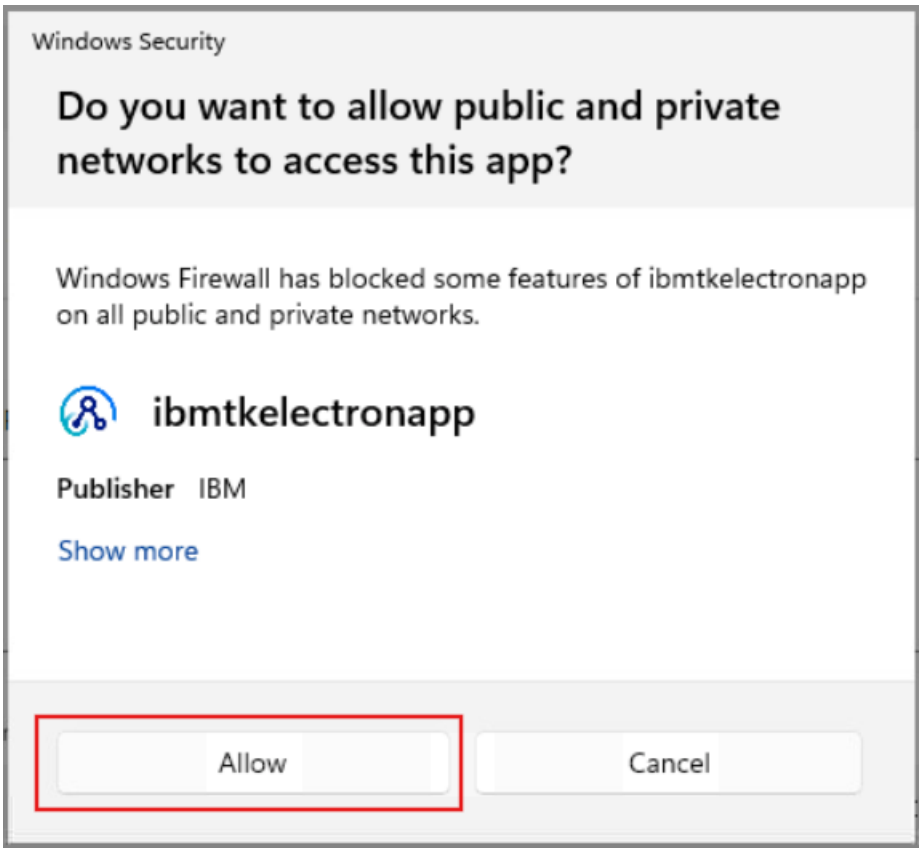
5. Adding an Operation to a REST API

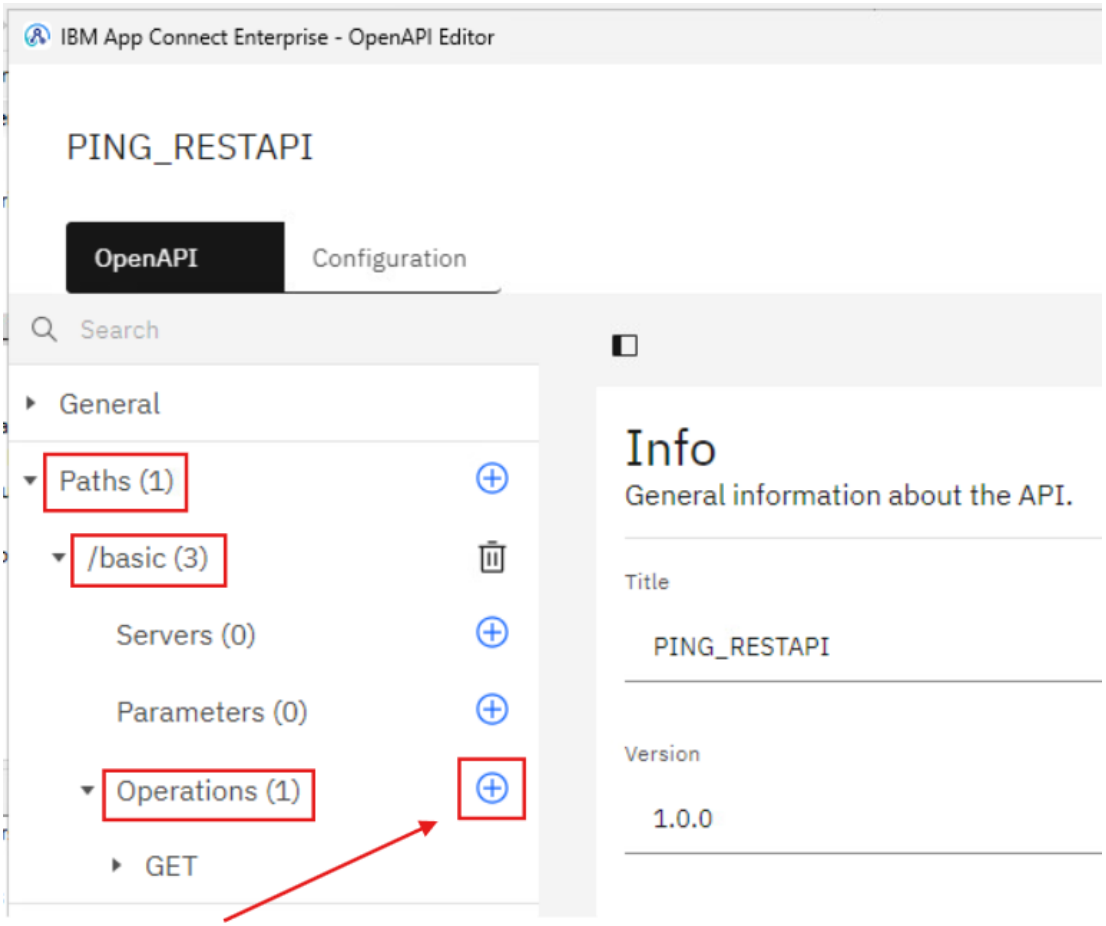
5.1 Modifying the Configuration

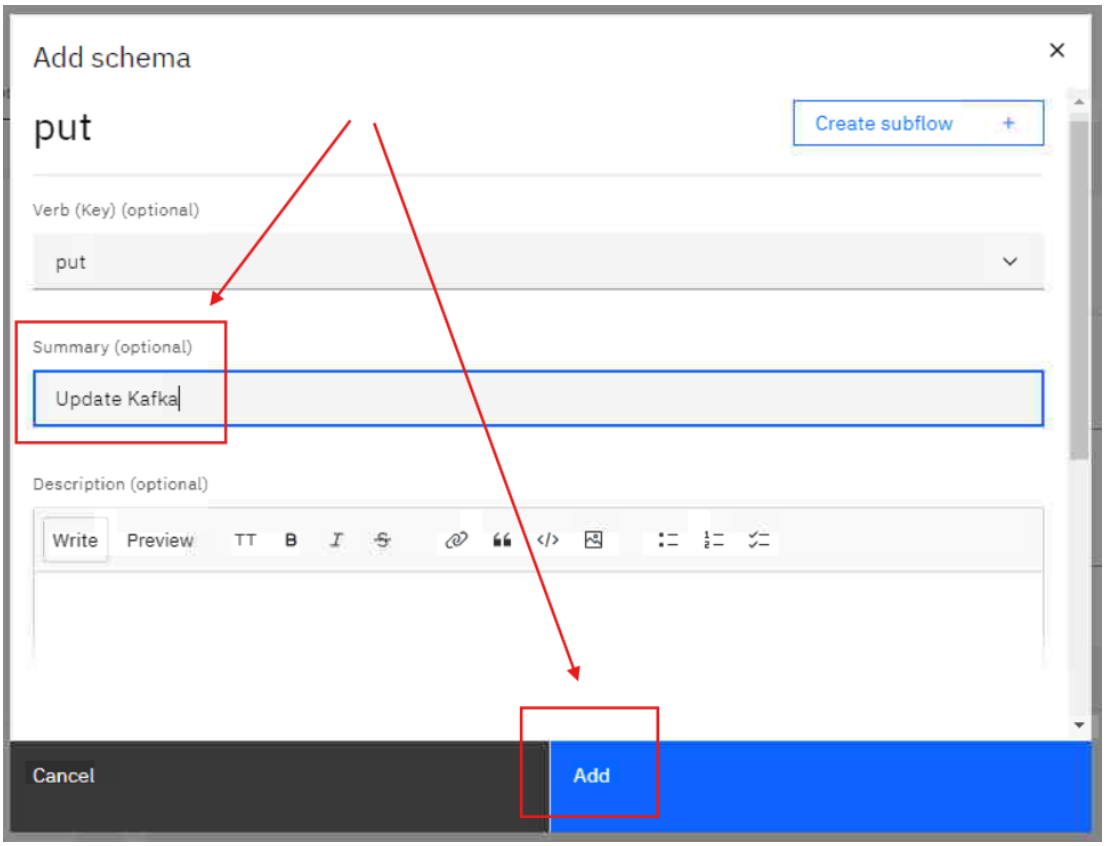
In this next section you will see how to add a PUT operation to the REST API. You will use the OpenAPI Editor to make the change to the REST API. The OpenAPI Editor is an Electron app that will open in a new window. Note when the OpenAPI Editor is open, the REST API Description is “locked” and becomes “Read-only”. At the end of this section you will have sent data to your local MQ Advance and Apache Kafka containers.

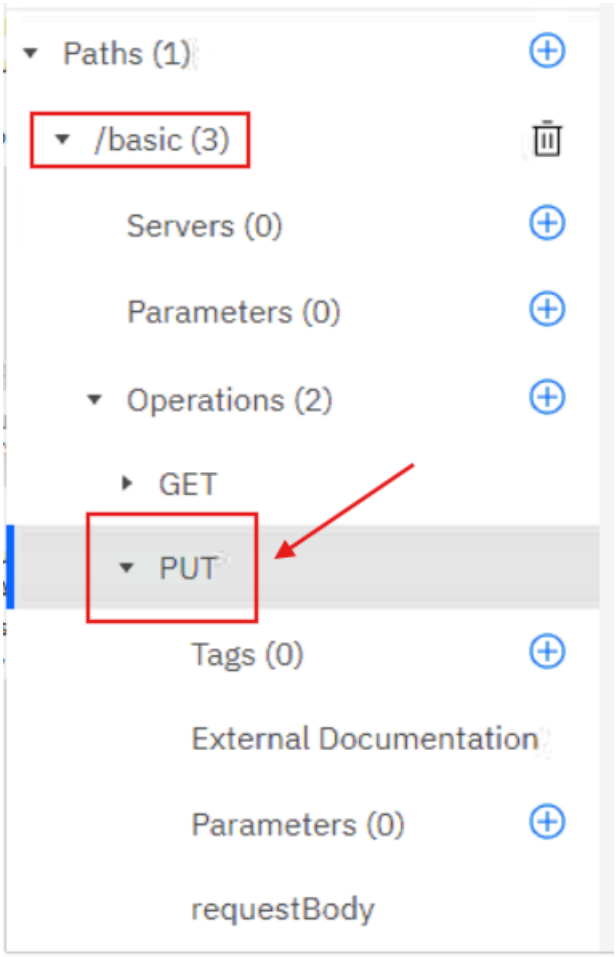
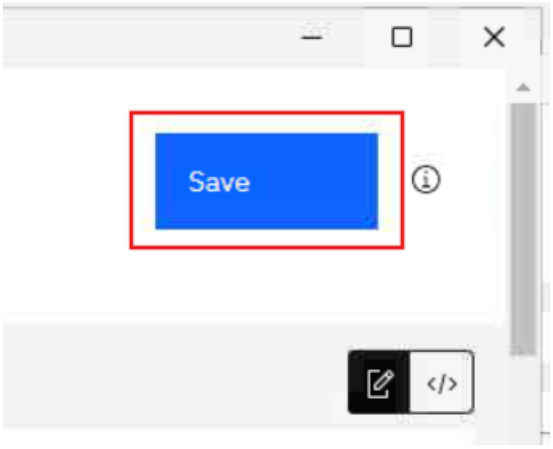


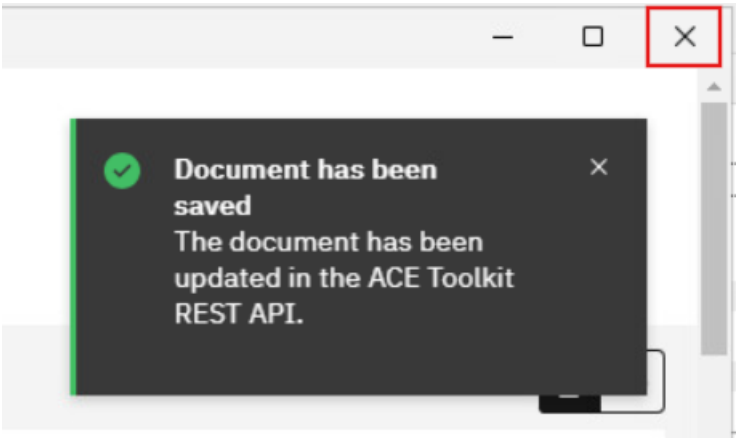
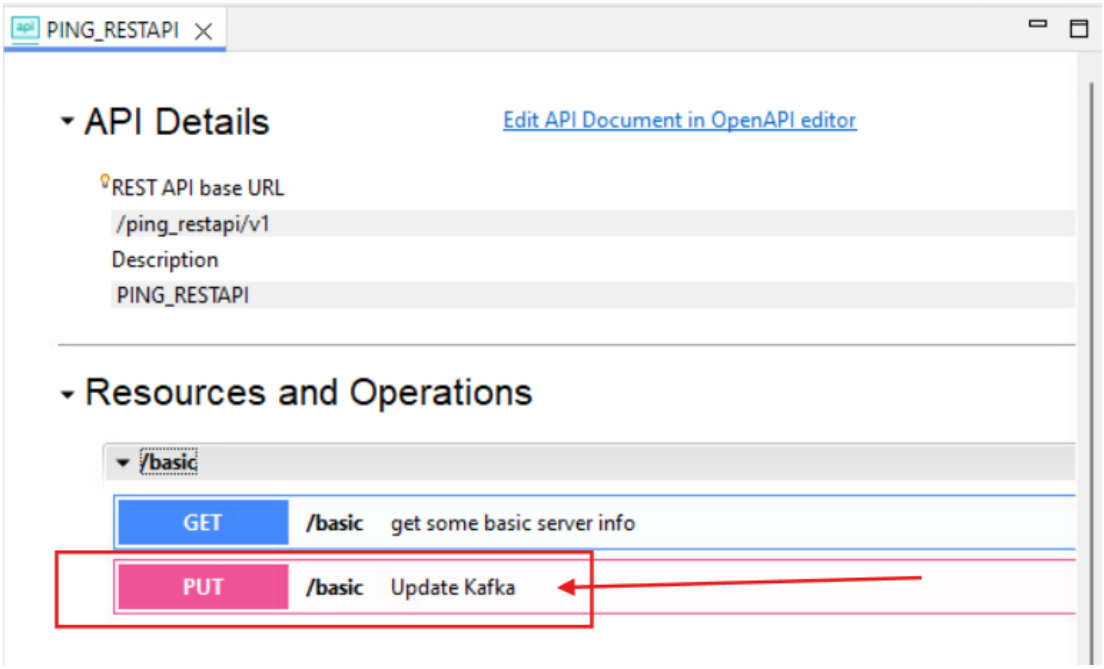
1.	In the PING_RESTAPI Description (Integration Development perspective), Click the link **Edit API Document in OpenAPI Editor**: 

2.	If you see the following pop-up window, click **Allow**: 

3.	In the OpenAPI Editor, a) Collapse the General section b) Expand Paths > /basic > Operations. Note the GET operation you tested earlier. c) Click the Plus sign to add another operation: 

4.	In the **Add object** window, add a summary “Update Kafka” and Click Create: 

5.	Note the PUT operation is now shown in the /basic path. 
6.	Scroll to the top of the page and click the Save button: 

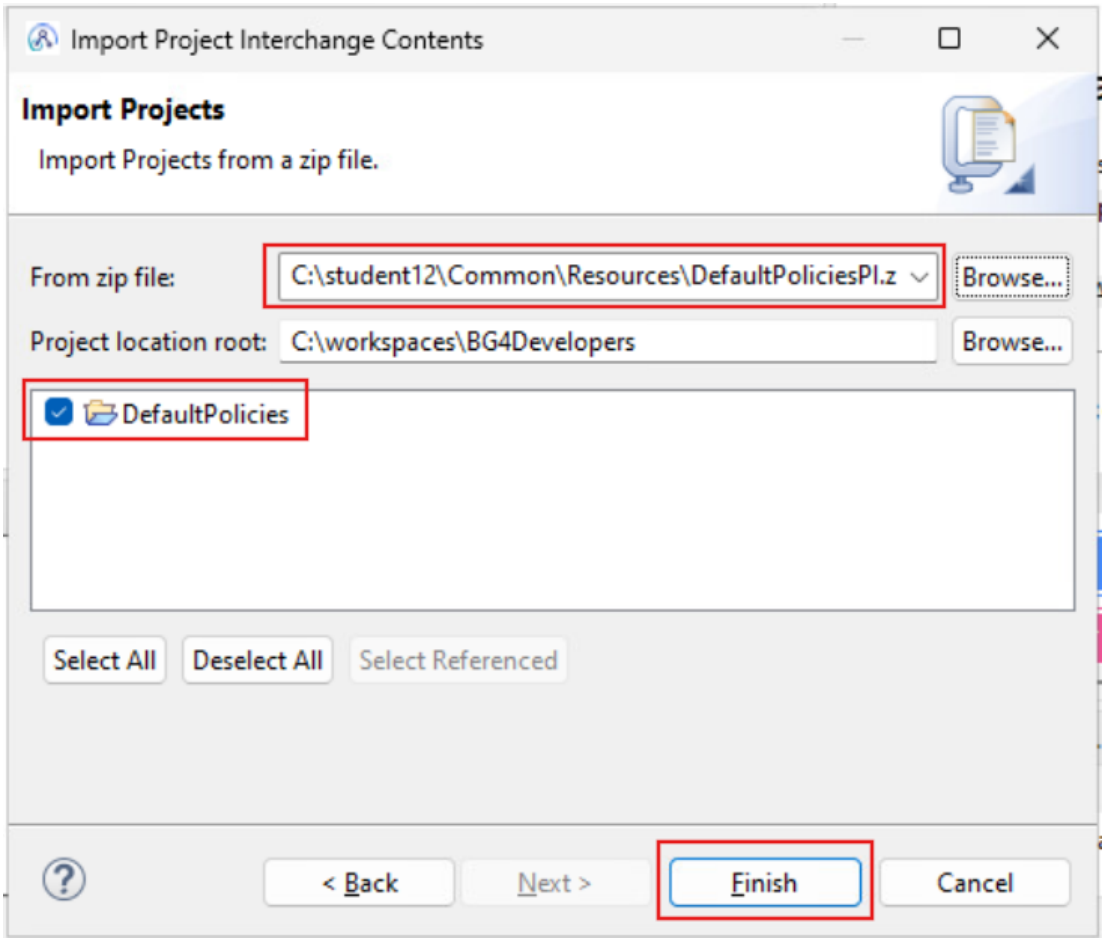
7.	Close the OpenAPI Editor: 
8.	In the PING_RESTAPI Description the **PUT operation** will now be shown under the **/basic** path: 

5.2 Implement the PUT Operation

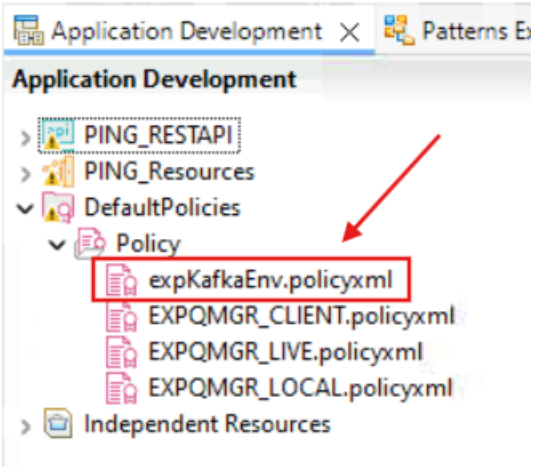
In this next section you will create a message flow to implement the PUT operation. The message flow will obtain some server data and send the data to Kafka and to MQ. Connectivity configuration will be done using Policies.

A supplied **DefaultPolicies** Policy Project with previously configured **Kafka Policy** and **MQEndpoint Policy** will be used to configure the connectivity details on each of the two nodes. Policies are useful artifacts when sharing information and ensure all nodes using a Policy are configured with the same information.

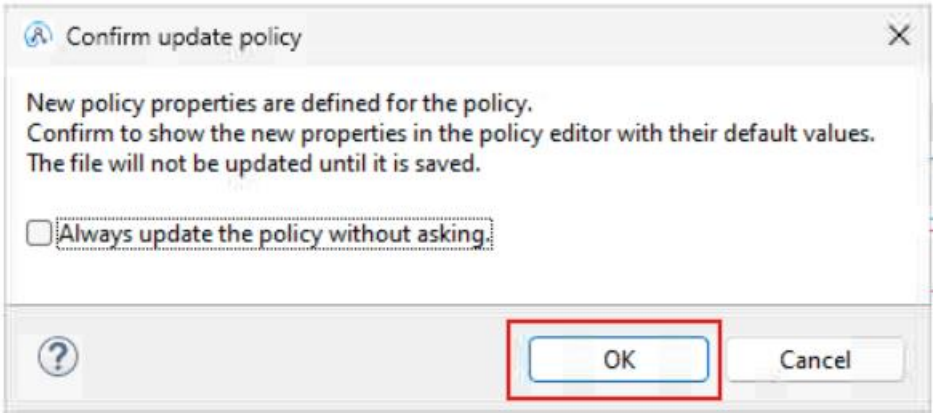
5.2.1 Import Policy Project: DefaultPolicies

1.	Right click on the Application Development window and select Import.
	
	Import the Project Interchange file:
	
	`DefaultPoliciesPI.zip`
	
	from
	
	`C:\student12\common\resources\`
	
	 

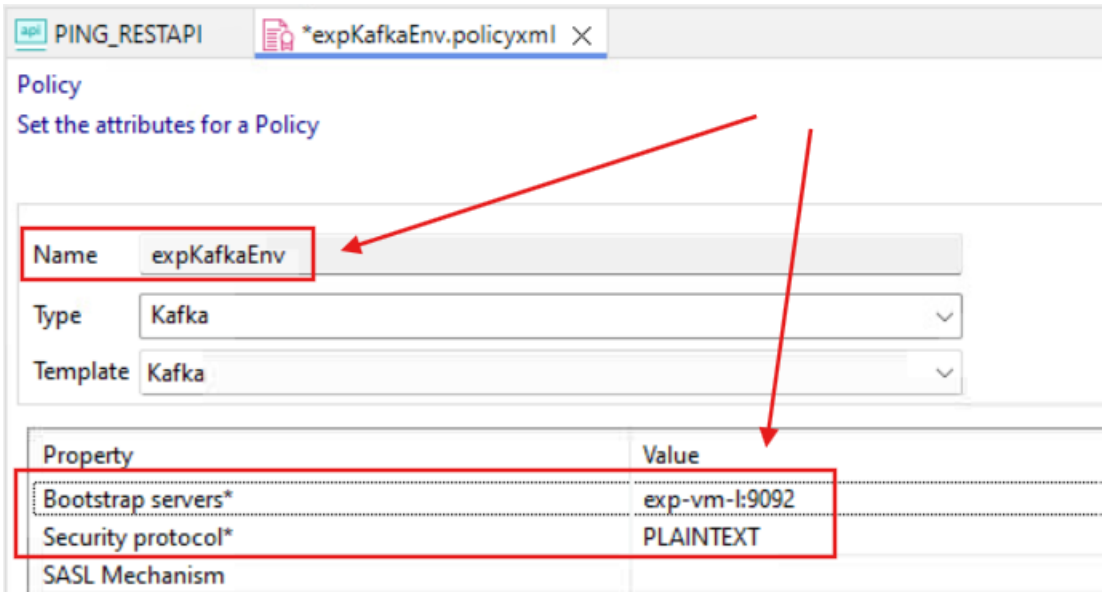
2. In the Application Development window expand DefaultPolicies > Policy and double click on the **eXpKafkaEnv** policy:

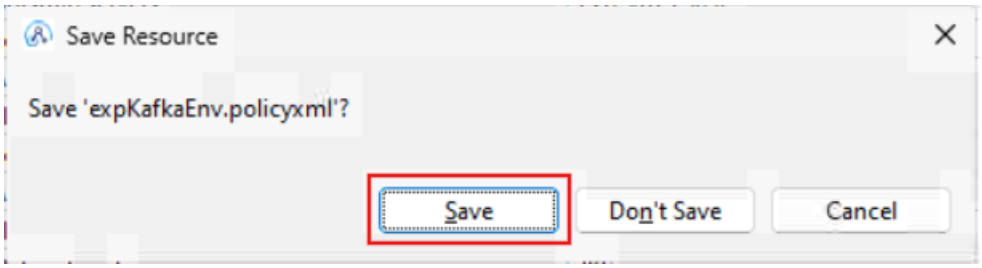


Click OK to dismiss the warning (the PI file we are using for this Lab was initially created some time ago using ACE12 and due to new properties being added in ACE since then, this warning is letting you know!):

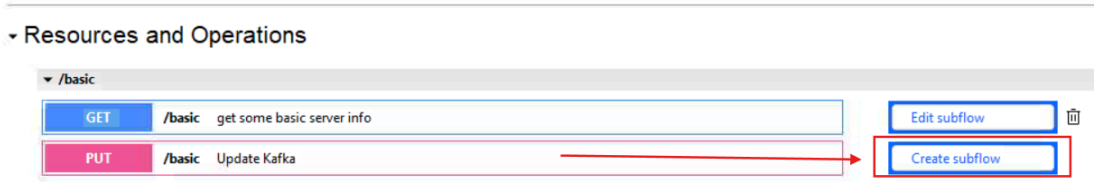


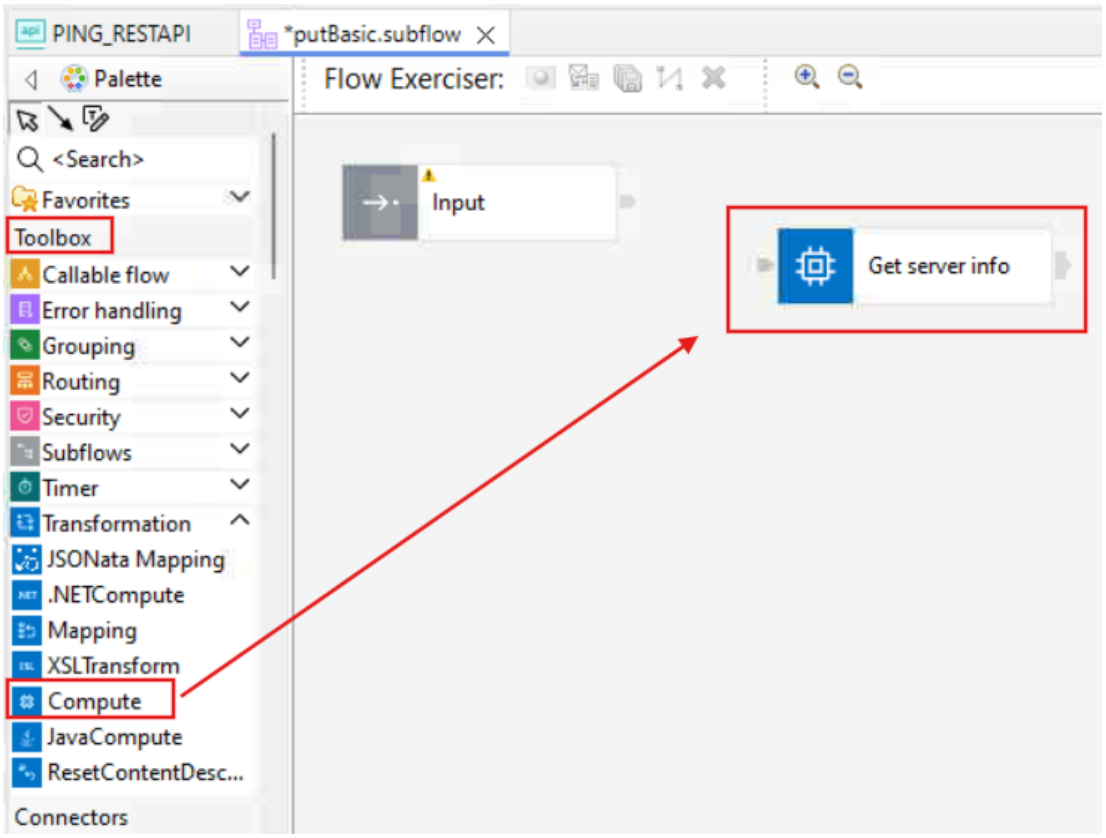
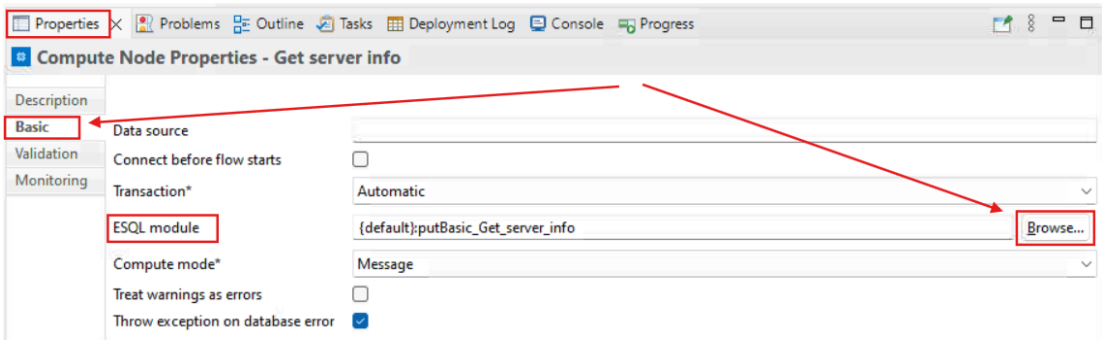
3. The Policy Editor will open, note the Bootstrap servers and Security Protocol (in this example PLAINTEXT) are configured in the Policy:

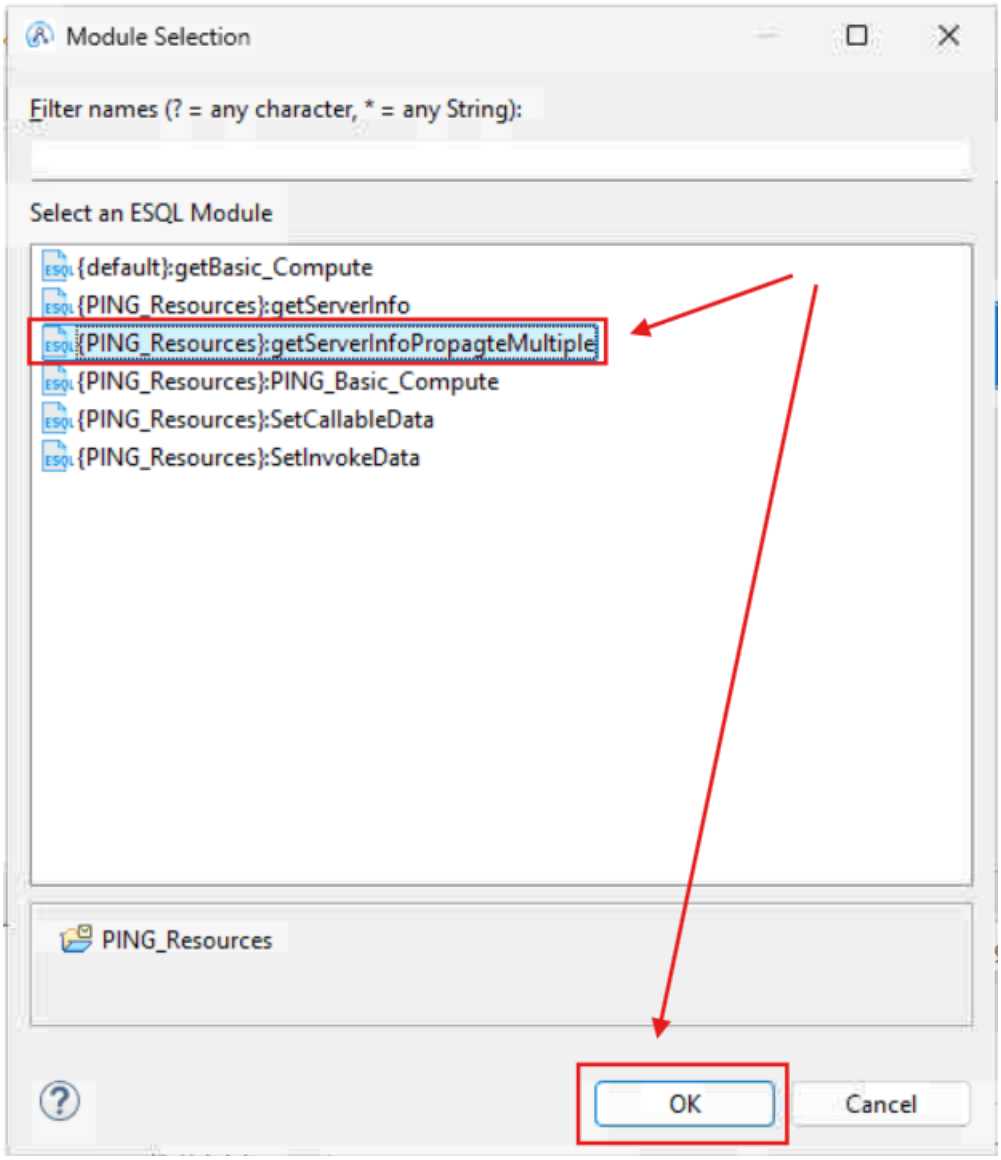


4.	Close the Policy Editor, agreeing to **Save** the policy: 

5.2.2Create the subflow

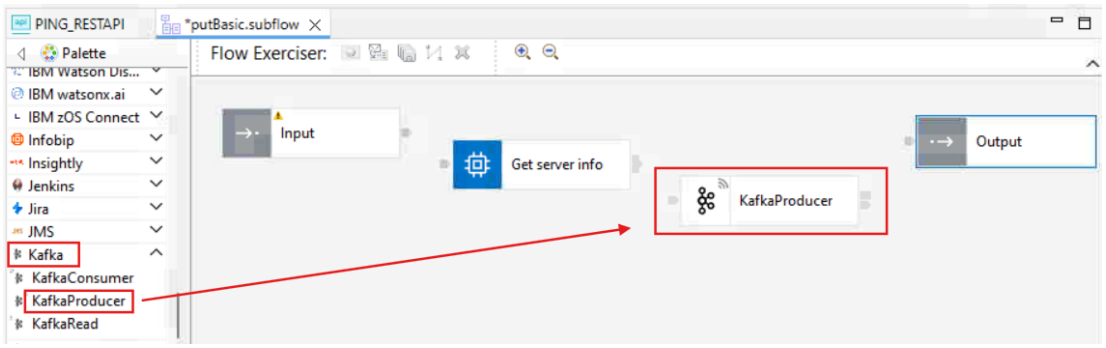
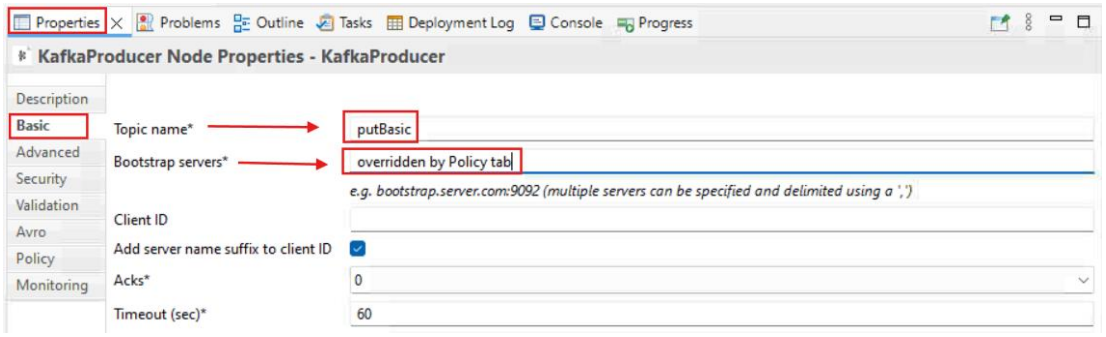
1.	Click the **Create subflow** button on the same line as the PUT operation *(the subflow implements the operation)*: 

2.	The subflow editor will open. In the Palette, expand the **Transformation** folder (under the Toolbox heading). Click the Compute node, then click the canvas to add the node to the subflow *(alternatively drag and drop the Compute node onto the canvas).* Call the Compute node “Get server info” (click the canvas when you have entered the name of the node) 
3.	Click the Compute to show the node properties. In the node properties (Basic tab), click the **Browse** button on ESQL module: 

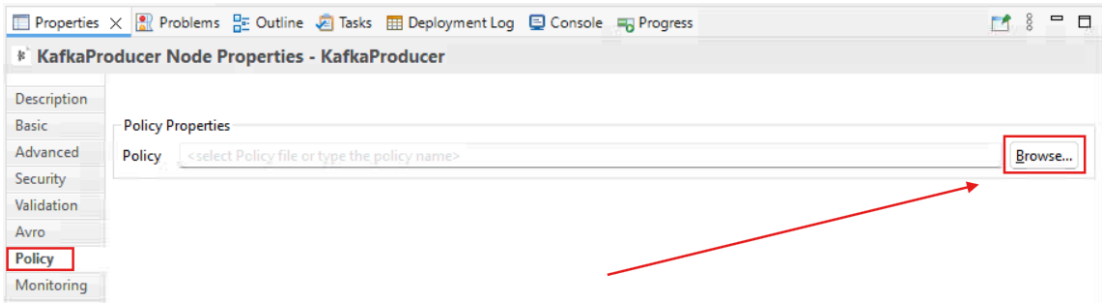
4.	In the Module Selection window select **getServerInfoPropagateMultiple** in the **PING_Resources** library (the library is specified in curly brackets). Click OK: 

5.2.3 Add a KafkaProducer node

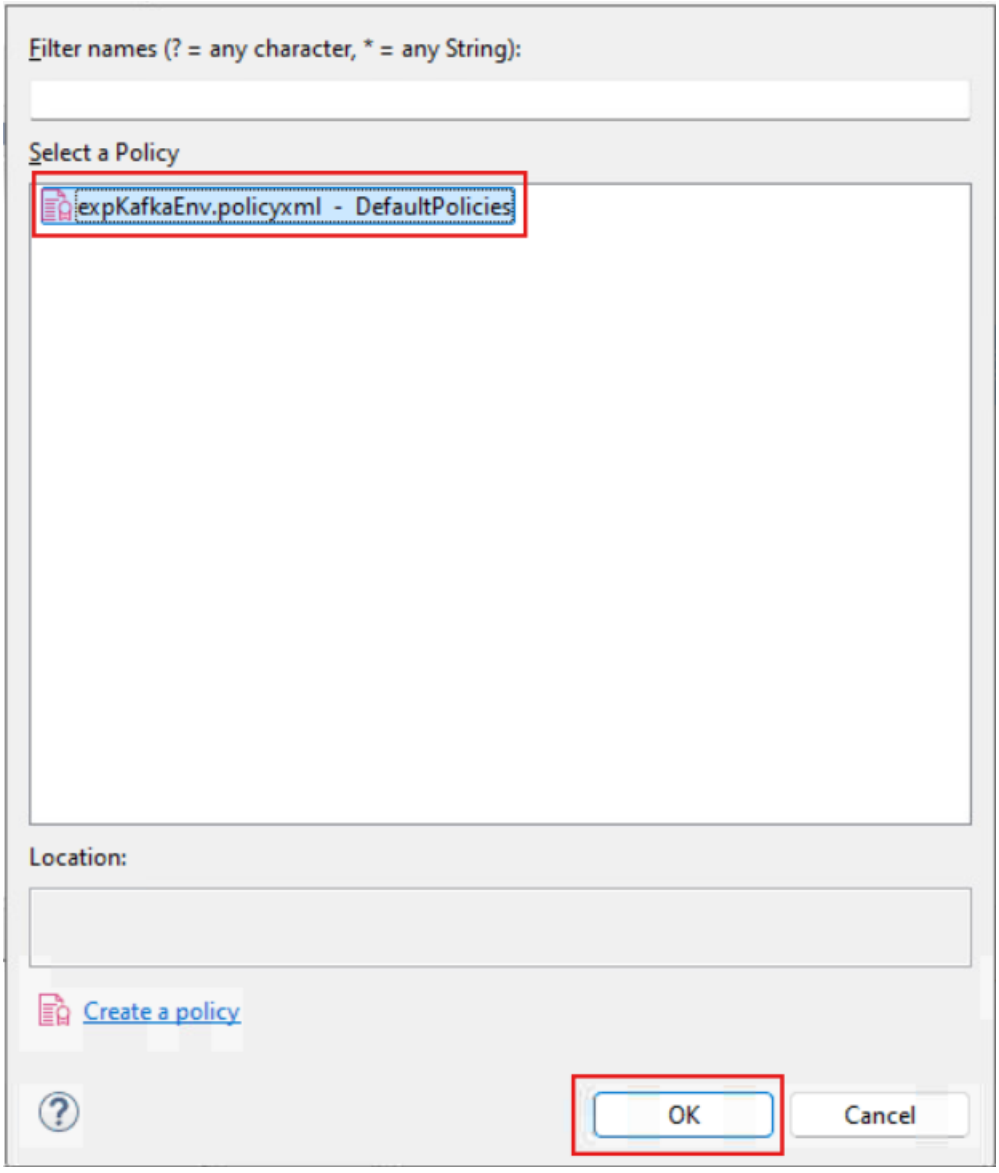
In this next section you will add a KafkaProducer node to the message flow. The Server Information will be written to a Kafka Topic using this node.

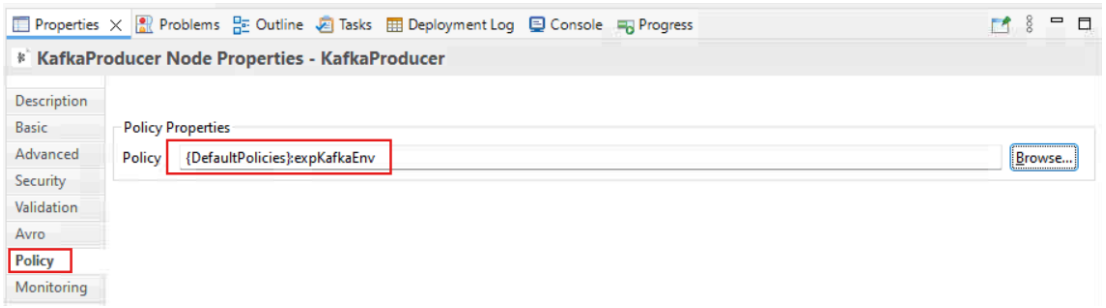
1.	Under the Connectors heading in the palette, expand the **Kafka** folder. Click **KafkaProducer** and then click the canvas to add the node to the message flow: 
2.	In the **Basic Tab** of the **KafkaProducer node** properties configure the following: a) Topic name: `putBasic` b) Bootstrap servers: `overridden by Policy Tab` 

3. In the Policy Tab, click the Browse button

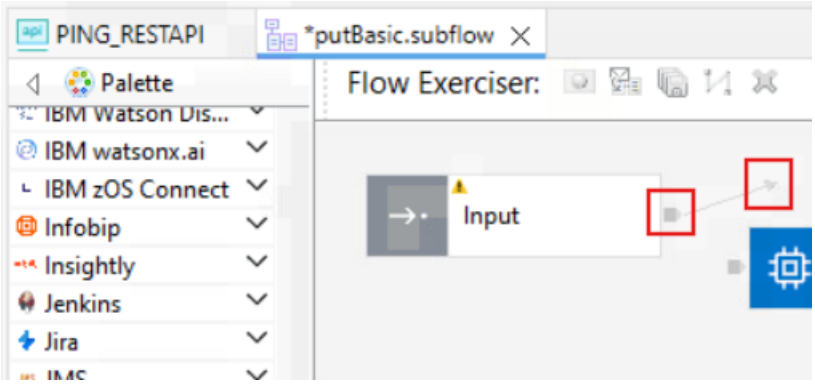
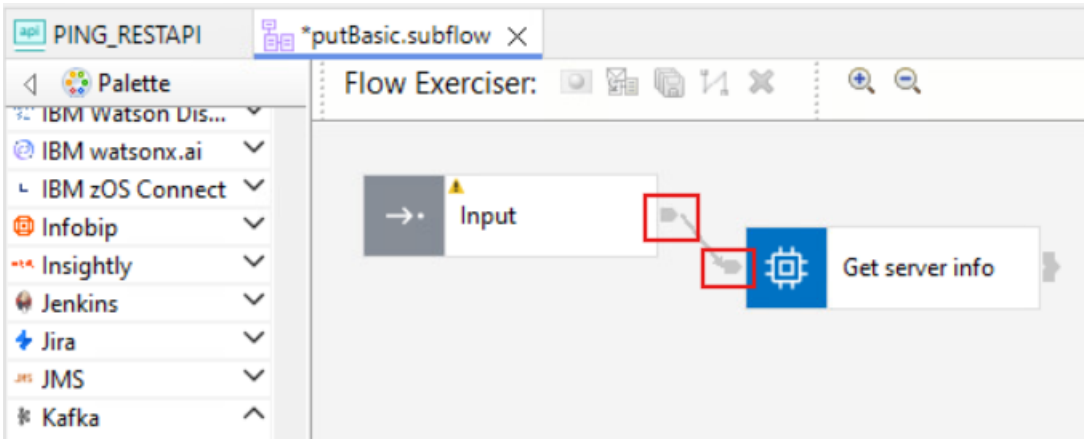
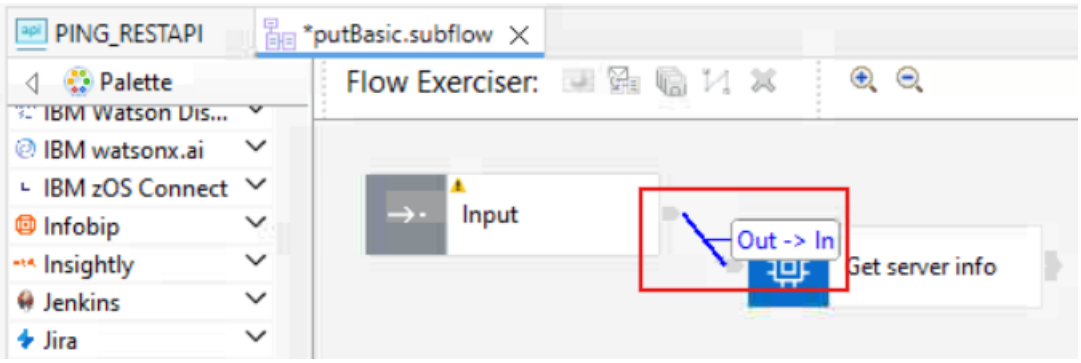


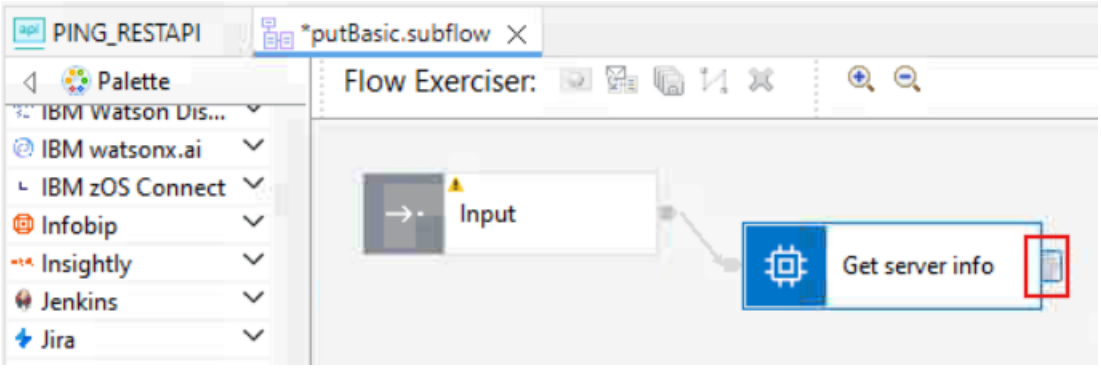
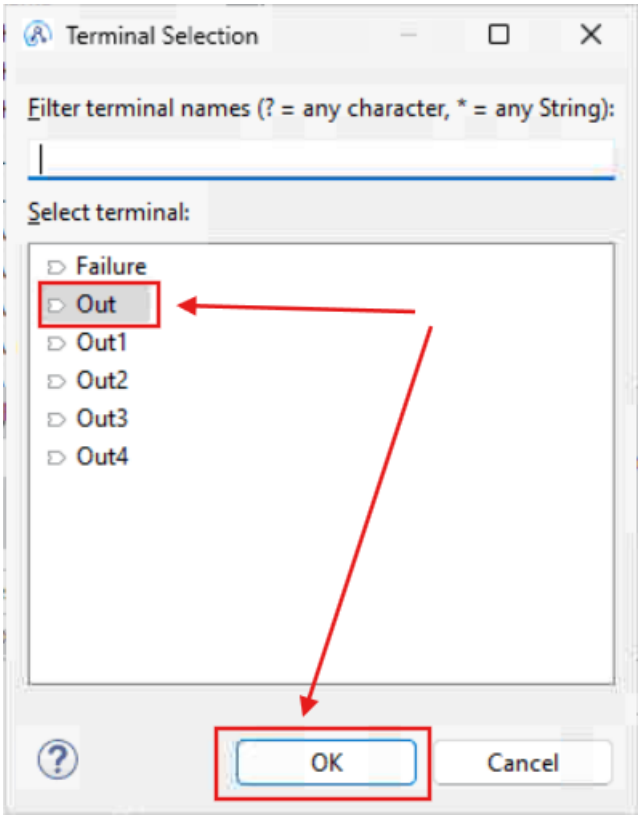
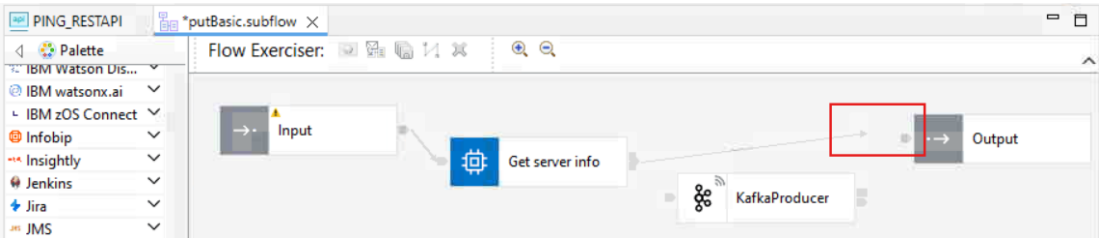
Select **expKafkaEnv.Policyxml** in **DefaultPolicies** and click OK:

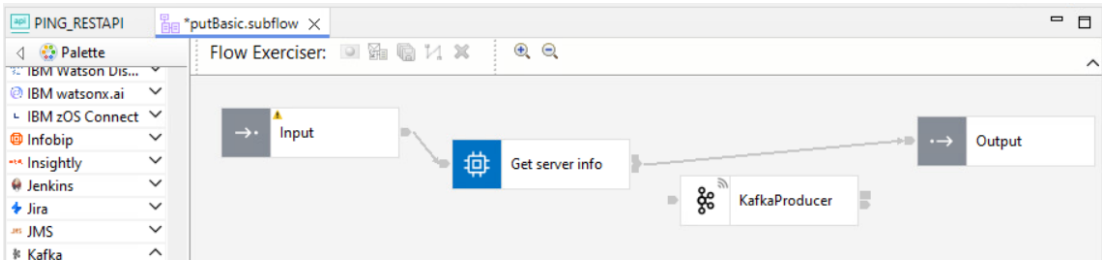
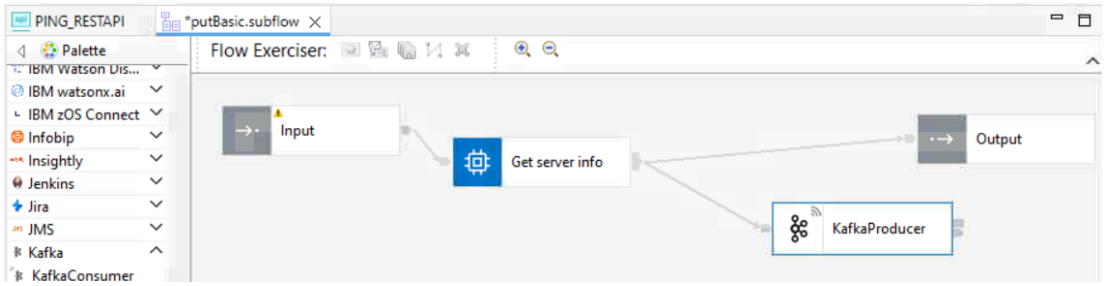


4.	The Policy tab will look like this when complete: 

5.2.4 Connect the Kafka nodes

1.	At runtime when the subflow is triggered, message data will travel down the connections between nodes. Click on the **Out** terminal of the **Input** node to begin the connection, you will see a line attached to your cursor:  Click the **In** terminal of the "**Get server info**" Compute node to connect to the two nodes: 
2.	Try hovering your cursor over the connection wire of the two connected nodes, you will see the names of the two connected terminals displayed (this is useful when a node has multiple terminals: 

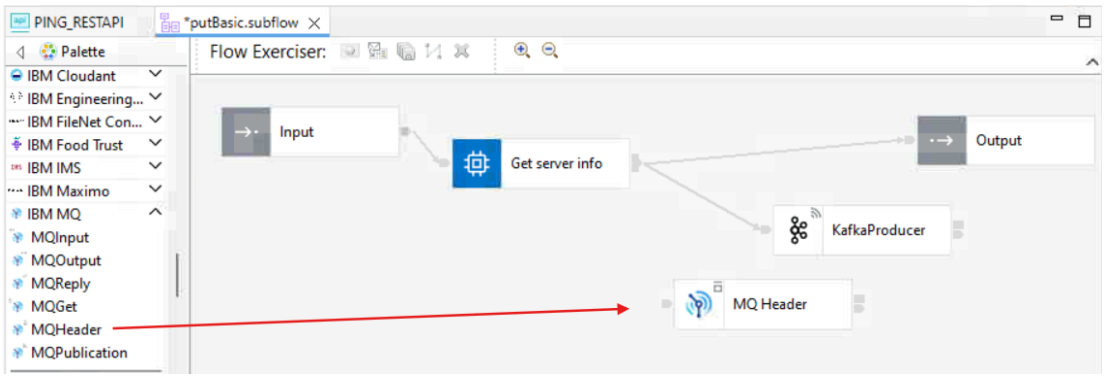
3.	Click on the multiple terminals icon on the **Get server info** Compute node: 
4.	In the **Terminal Selection** window select the **Out** terminal and click OK: 
5.	This will create a connection wire for the Out terminal. The wire will connect to the terminal that you next click on. (**HINT**: you can actually click anywhere on the node and the In terminal will get selected) Click the “in” terminal of the Output node: 

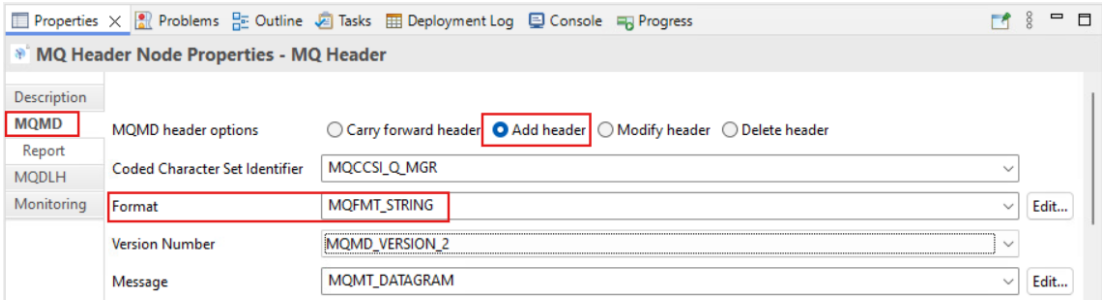
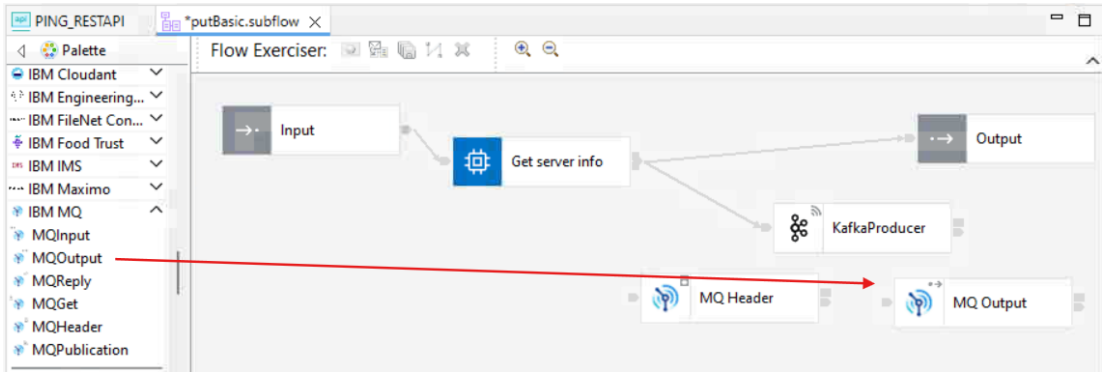
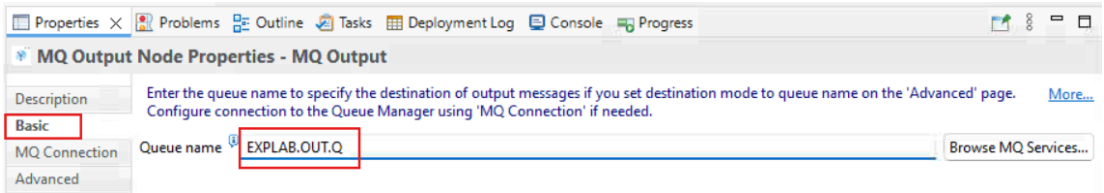
6.	The connections between the two nodes will look like this when complete: 
7.	Repeat the above steps to connect terminal **Out1** to the **In** terminal of the **KafkaProducer** node. The connection between these nodes will look like this when complete: 
8.	Save the message flow changes so far (<ctrl> s)

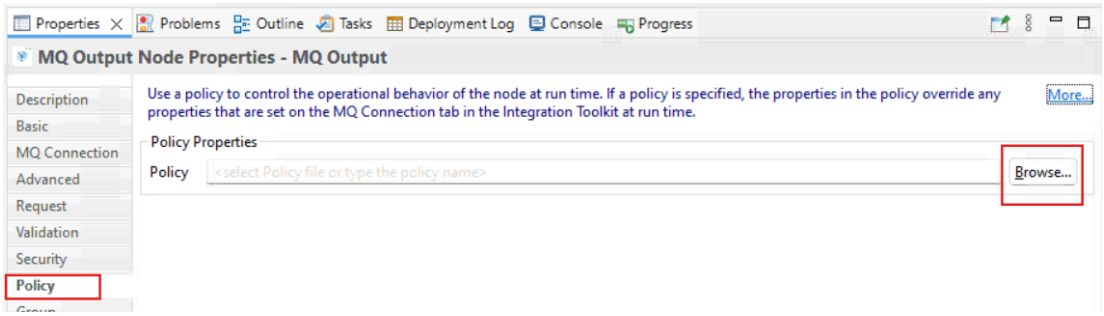
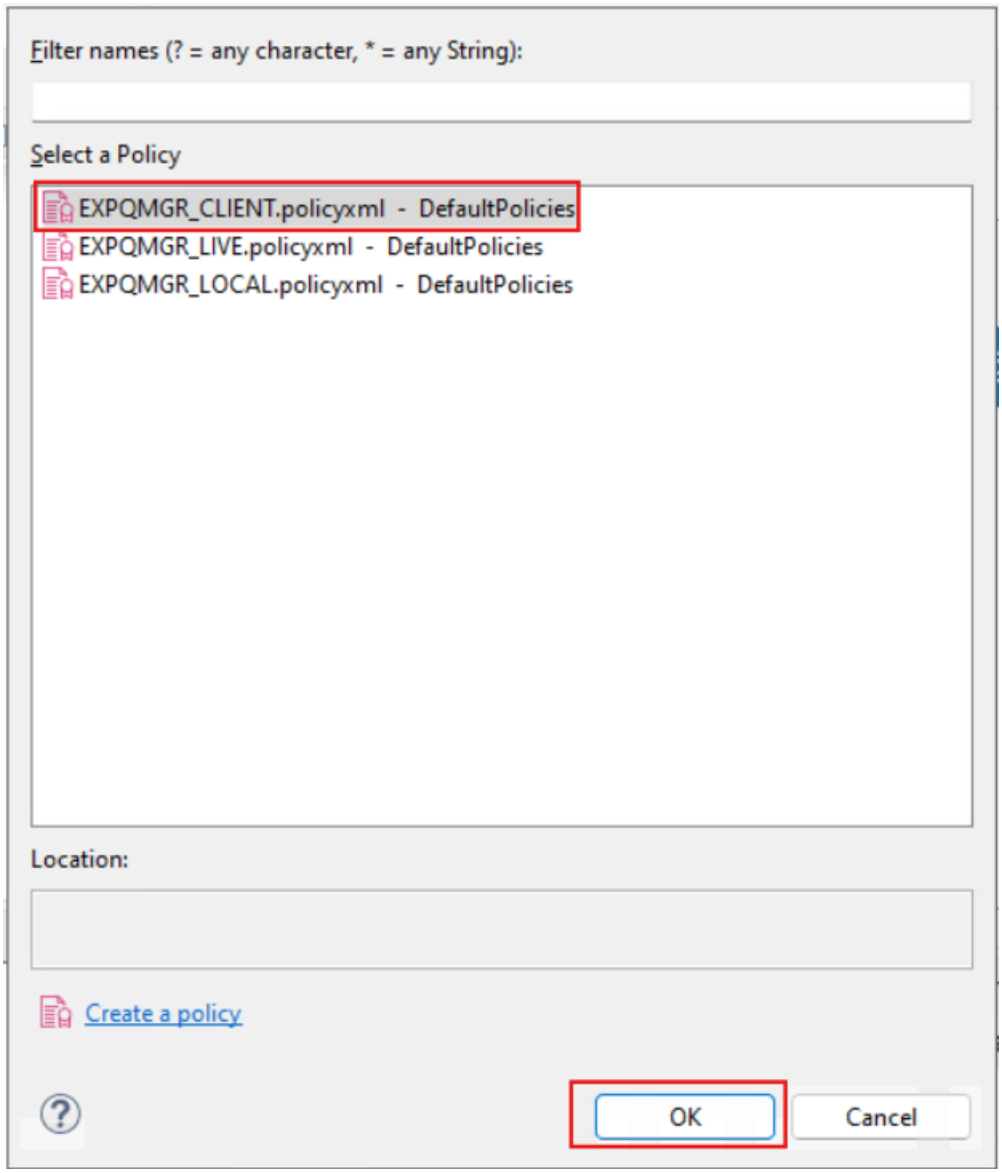
5.2.5 Add an MQ Output node

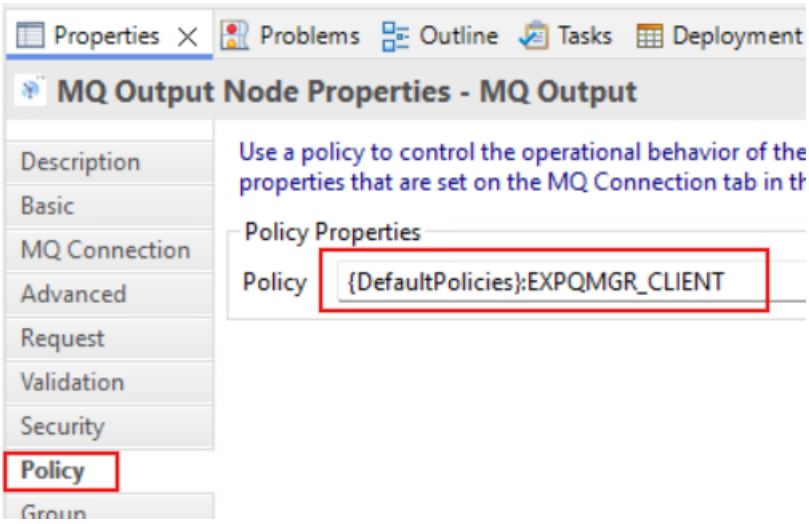
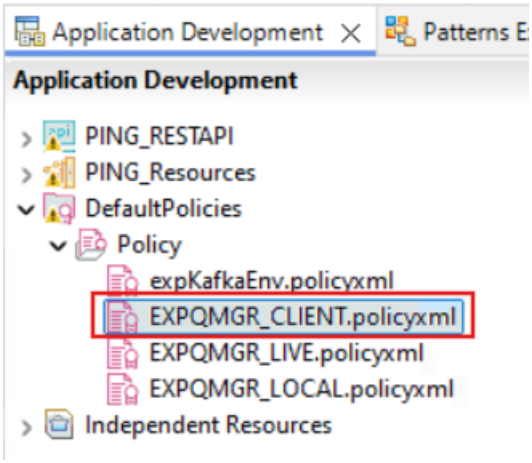
In this next section the message that is sent down the Out2 terminal as a result of the `PROPAGATE` statement in the **Get server info Compute node** will be sent to an IBM MQ Queue called `EXPLAB.OUT.Q`

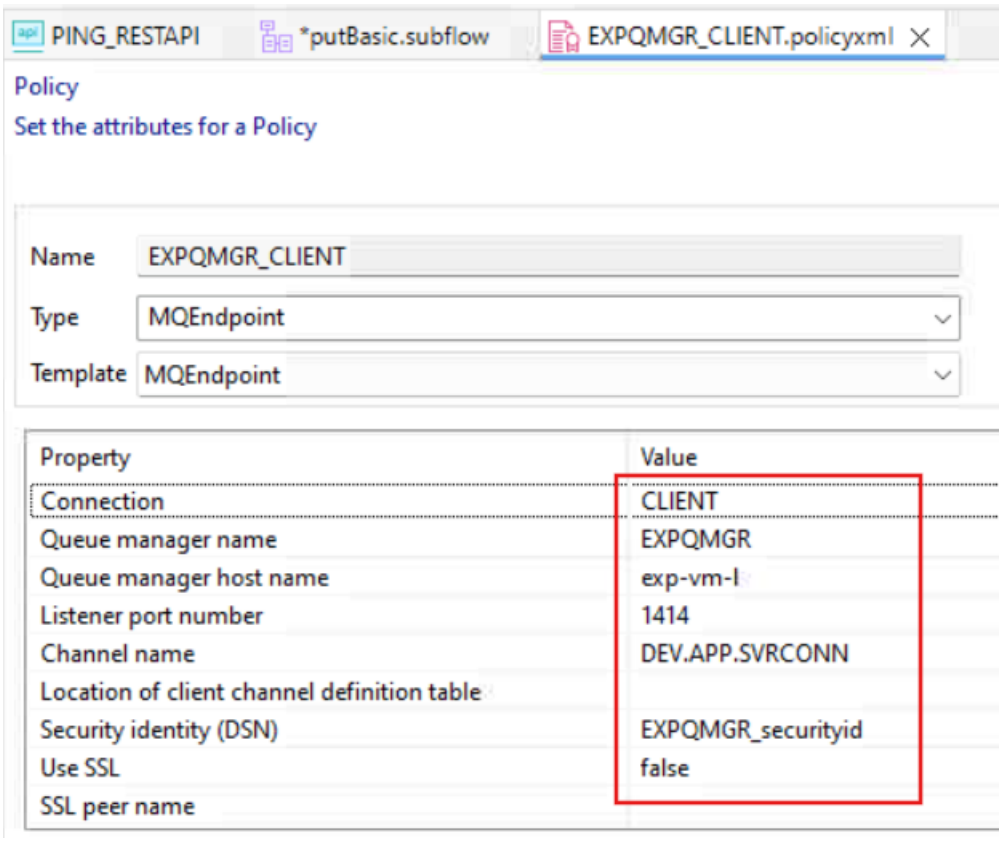
You will use a previously created **MQEndpoint Policy** to configure the connection details for the MQ environment.

1.	In the palette, expand the **IBM MQ** folder. Select the **MQHeader** node and click the canvas to add the node to the message flow. 

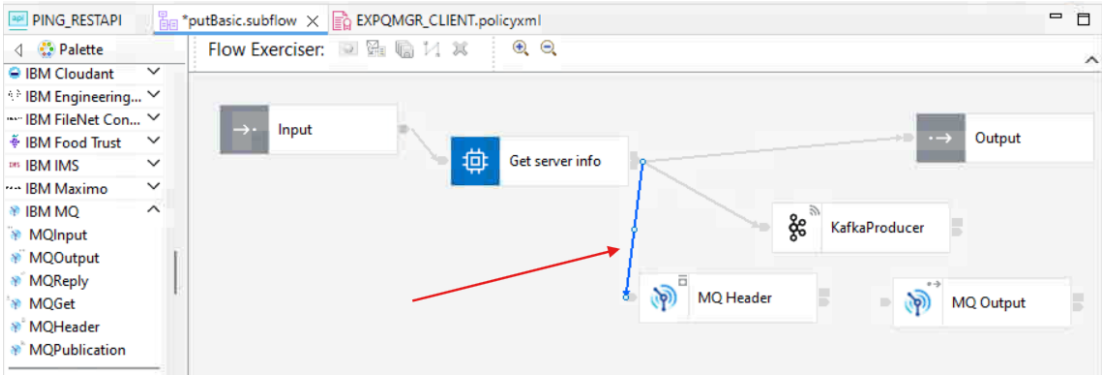
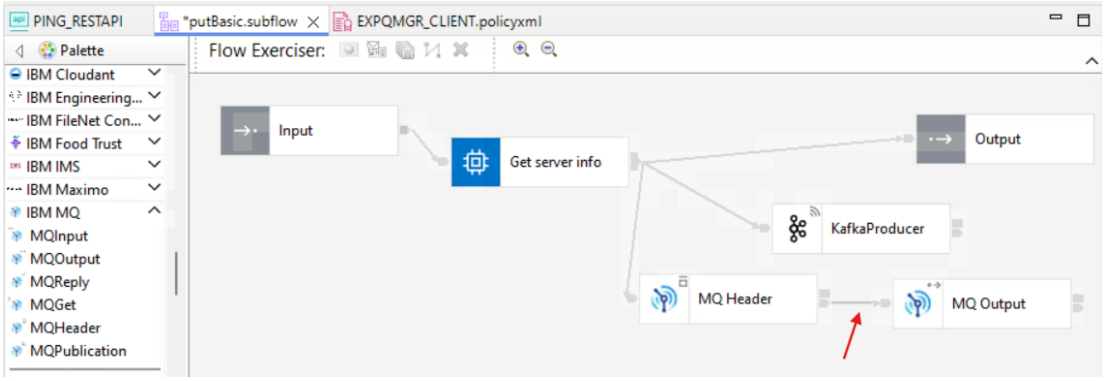
2.	In the properties tab for the MQ Header node (MQMD tab), configure the following: MQMD header options: **Add header** Format : **MQFMT_STRING**  *Adding the MQ Header with these settings will allow the MQ tools to display the content of the message.*
3.	Select the **MQOutput** node from the IBM MQ Folder in the palette and click the canvas to add the node to the message flow: 
4.	In the MQ Output node properties (Basic tab), configure the following: Queue name: **EXPLAB.OUT.Q** 

5.	The MQ Endpoint connection Policy that you will use to configure the MQ Output node has already been created in the DefaultPolicies policy project. The MQ Output node just needs configuring to use it. Click the Browse button on the Policy tab: 
6.	Select **EXPQMGR_CLIENT** in the Policy Selection window and click OK: 

7.	The MQ Output node **Policy Properties** will look like this when complete: 
8.	In the **Application Development** window expand DefaultPolicies > Policy and double click on the **EXPQMGR_CLIENT** policy: 

9.	The Policy Editor will open, note the MQ Connection configuration for the Queue manager EXPQMGR. The Security Identity `EXPQMGR_securityid` will need to be defined in a Credential Vault *(you will do this later in the lab guide)*: 
10.	Close the Policy Editor.

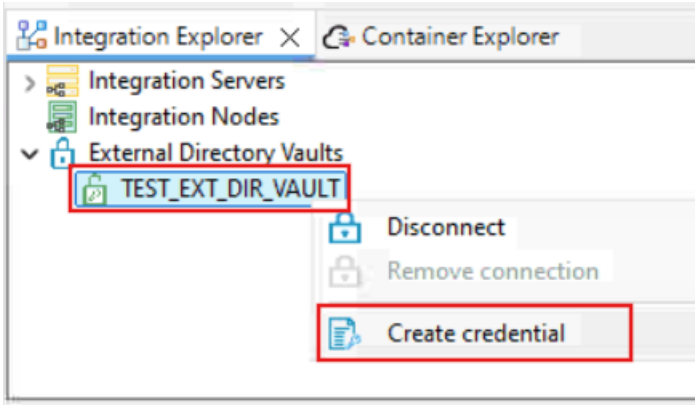
5.2.6 Connect the MQ nodes

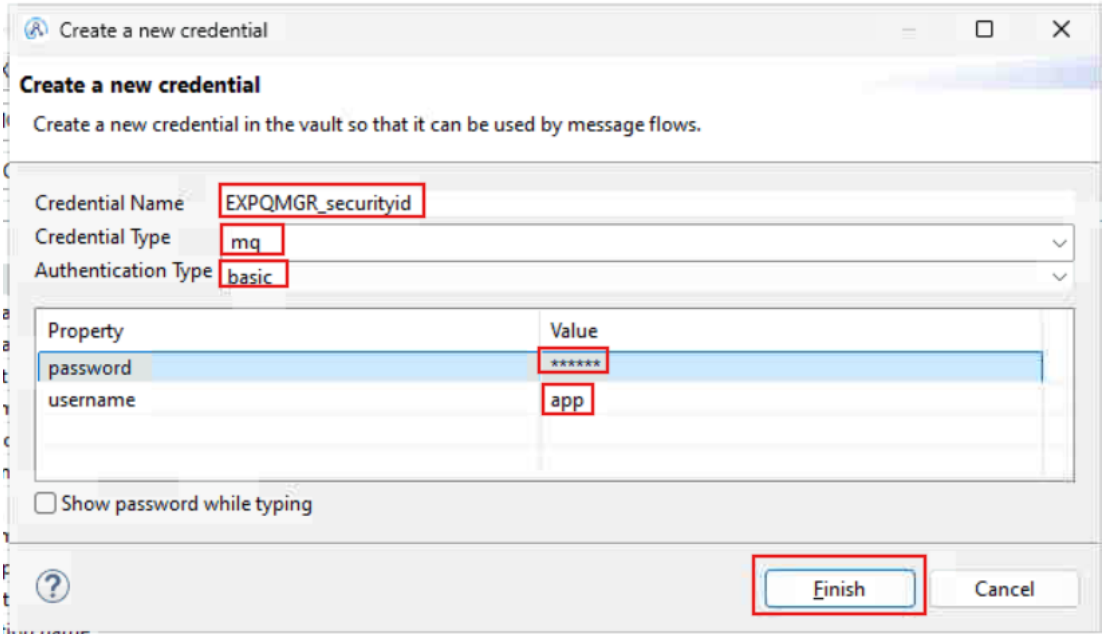
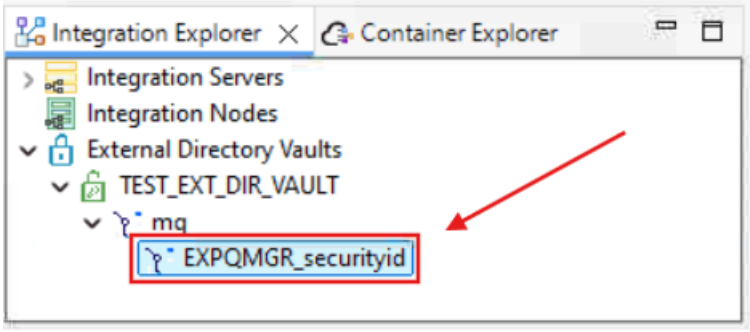
1.	Connect the **Out2** terminal of the Get server info compute node to the **In** terminal of the MQ Header node: 
2.	Connect the **Out** terminal of the MQ Header node to the **In** terminal of the MQ Output note: 
3.	Save the subflow <ctrl> s
4.	Close the subflow editor

5.3 Configure Security Vault

In this section you will configure the security identity that the MQ Output node will use. You will store this identity in an **ACE External Directory Vault** that TEST_SERVER was configured to use. An ACE External Directory Vault is a capability that comes as part of the ACE installation, which enables users to safely store (symmetrically encrypt) credentials that are used for communicating with 3rd party applications. The term “external directory” aims to describe that the file location where the vault is stored can be anywhere on your local disk and can be separated from the integration server’s main working directory. An External Directory Vault can also be shared between more than one server.

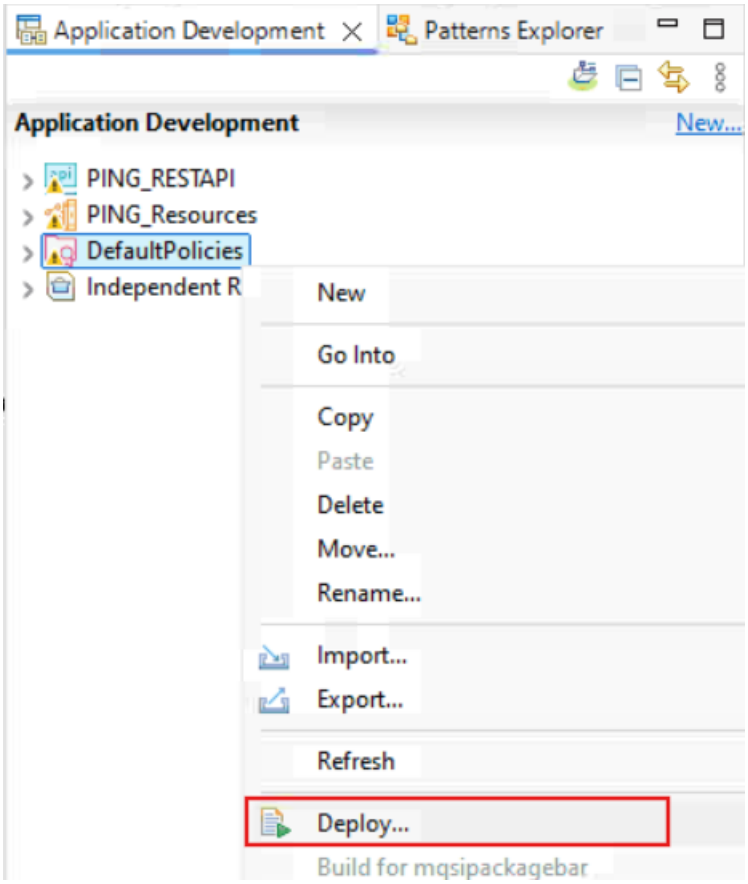
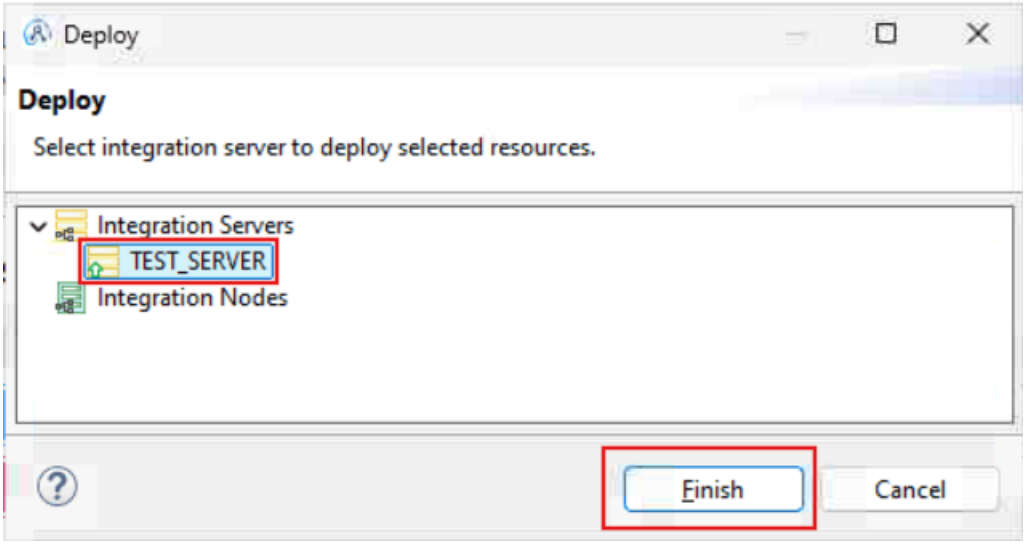
More information on Security identities can be found in the ACE Documentation at this URL: https://www.ibm.com/docs/en/app-connect/13.0.x?topic=vault-configuring-external-directory

1.	In the Integration Explorer window, right click on **TEST_EXT_DIR_VAULT** and select **Create credential**. 

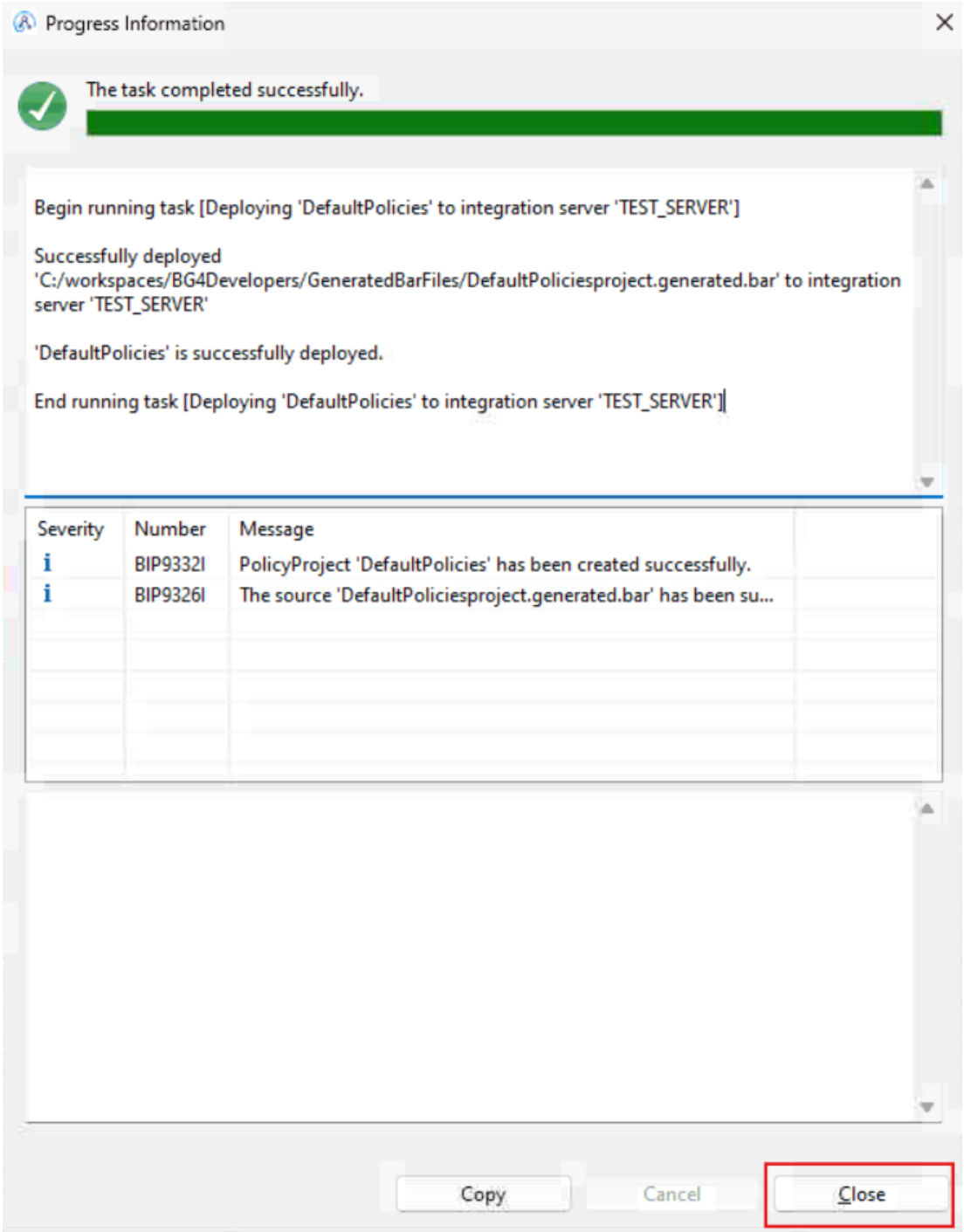
2.	In the **Create a new credential** window, set the following properties: • Set **Credential Name** = `EXPQMGR_securityid` • Set **Credential Type** = mq • Set **Authentication Type** = `basic` • Set **username** = `app` • Set **password** = `passw0rd` Click **Finish**: 
3.	The mq credential **EXPQMGR_securityid** will now appear in the Integration Explorer window within **TEST_EXT_DIR_VAULT**: 
4.	MQ Credentials are now dynamic within ACE, so there is no need to restart the integration server for this credential to be available to the message flows which are deployed to the integration server.

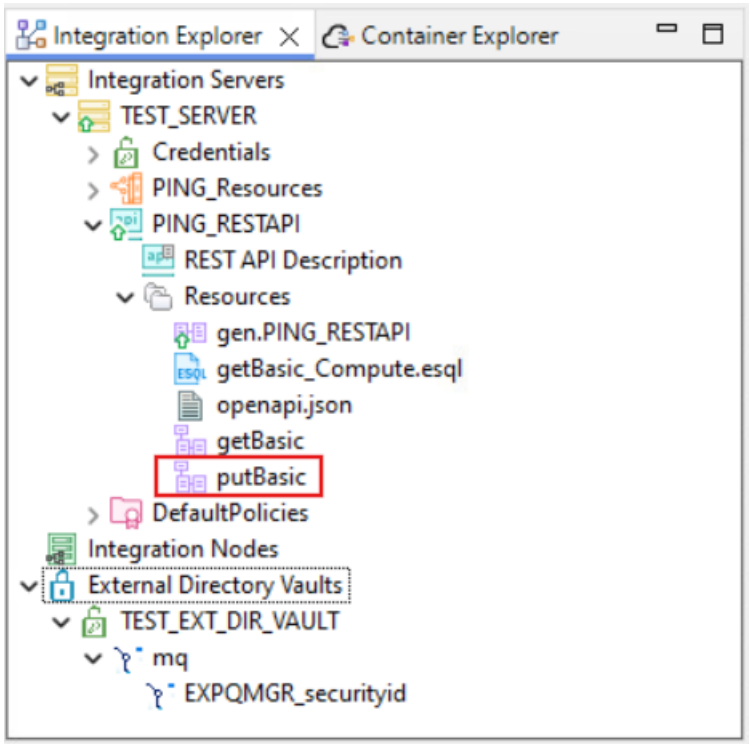
5.4 Deploy all Updated Resources

In this next section you will see how to use the ACE Toolkit Flow Exerciser with a REST API.

1.	In the Application development window, right click on DefaultPolicies and select Deploy. 
2.	Deploy to TEST_SERVER when prompted to choose a server: 

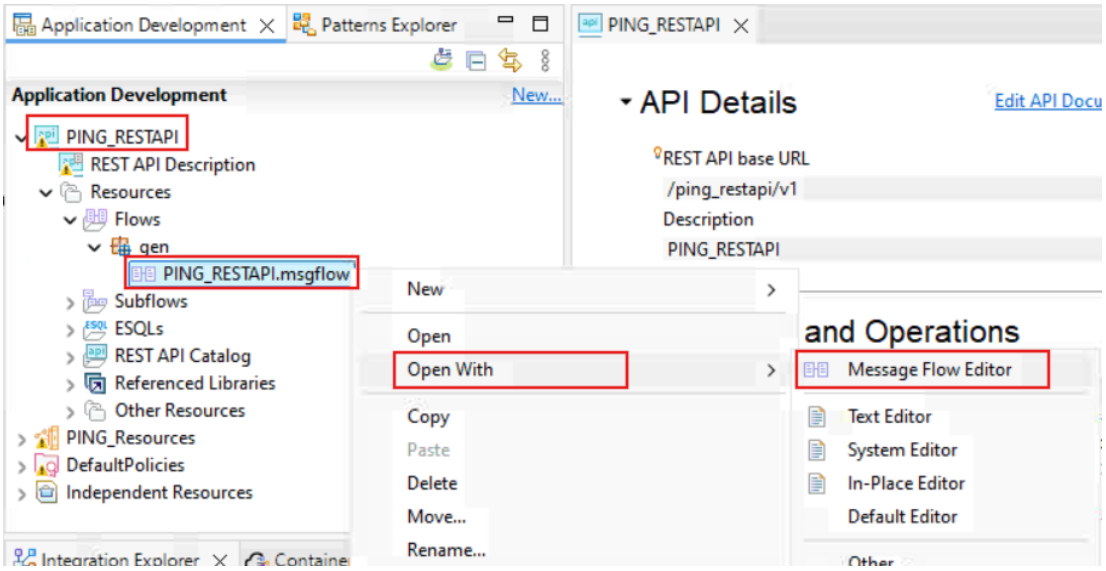
3.	Ensure the Deploy works successfully. Close the Progress Information window:

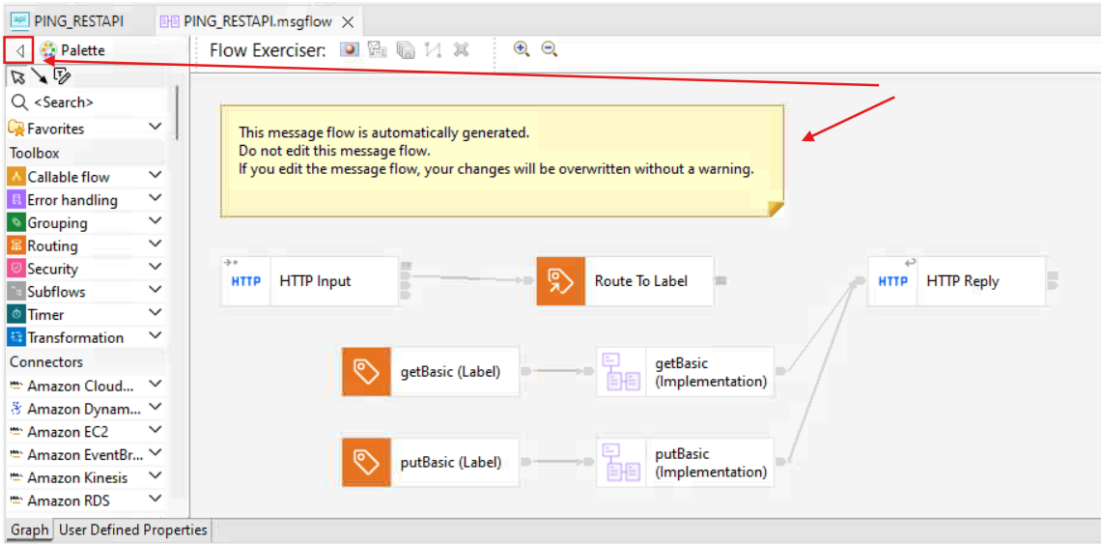
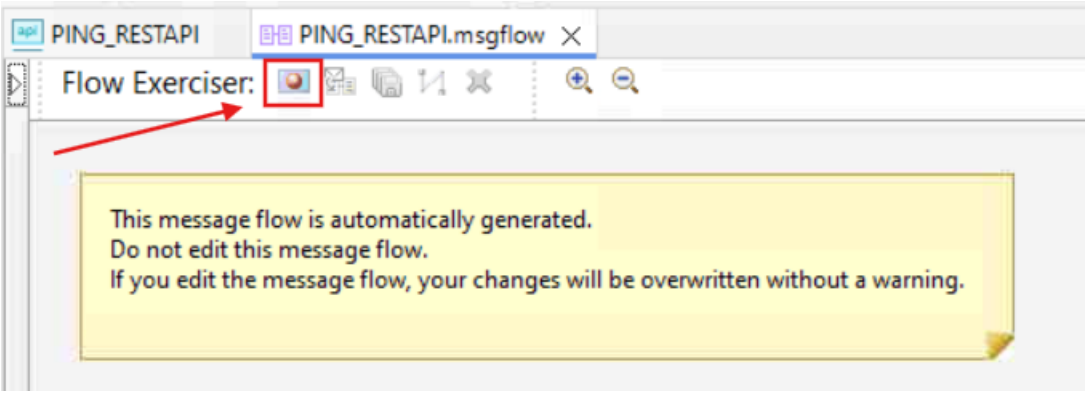
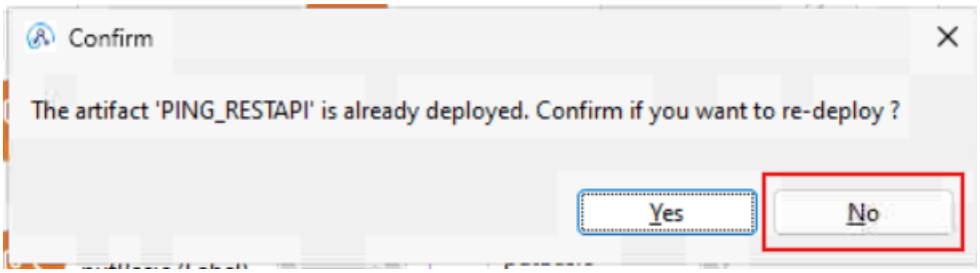


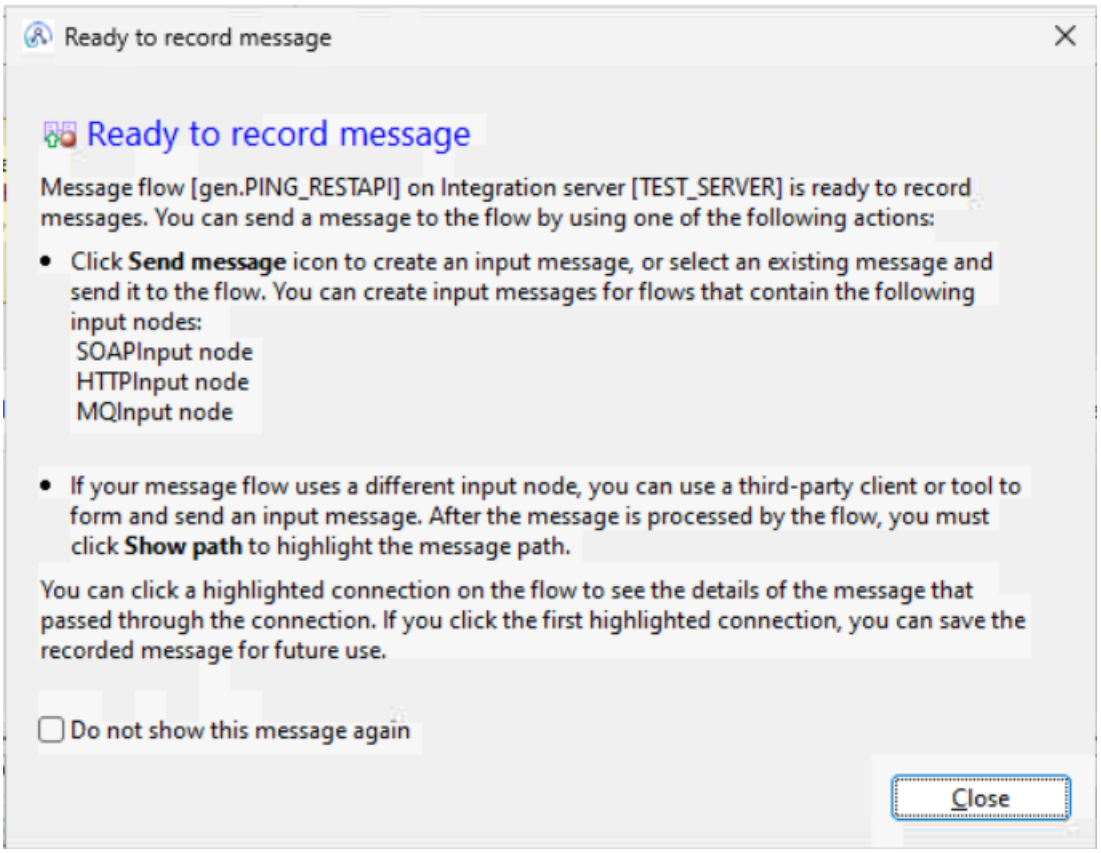
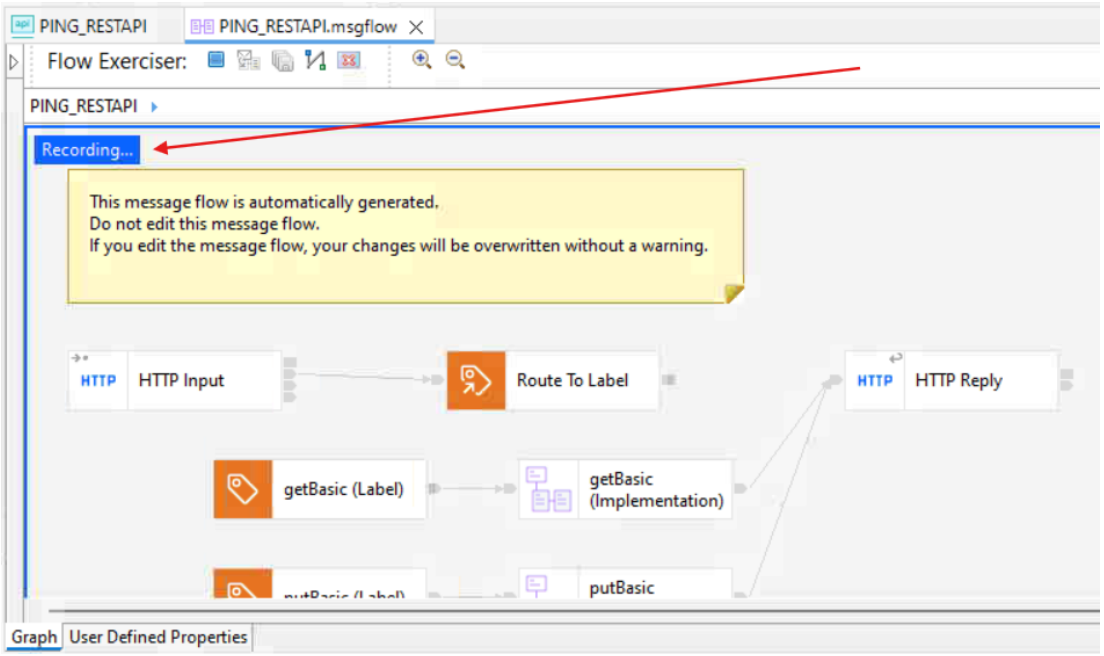
4.	Repeat the above step for PING_RESTAPI so that the new operation is available in the runtime environment: To check that the putBasic operation is now available in the TEST_SERVER runtime environment, expand PING_RESTAPI until you see the putBasic message flow: 

5.5 Test using Flow Exerciser

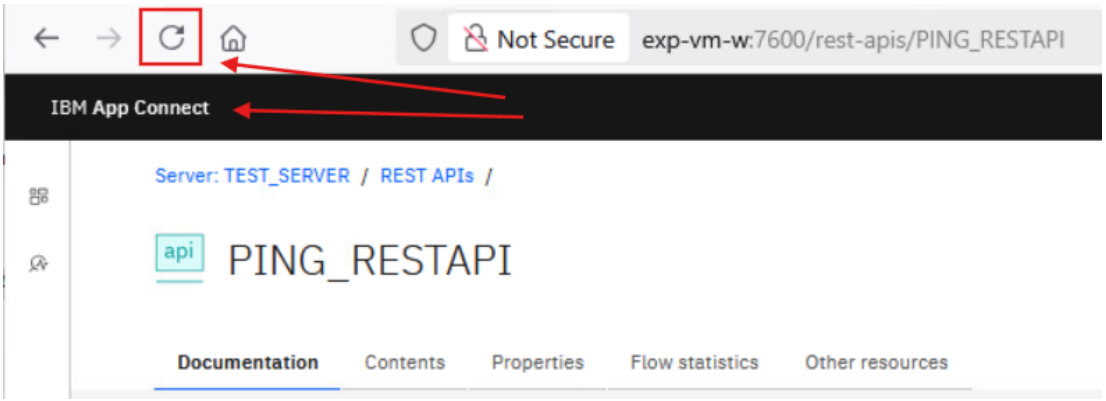
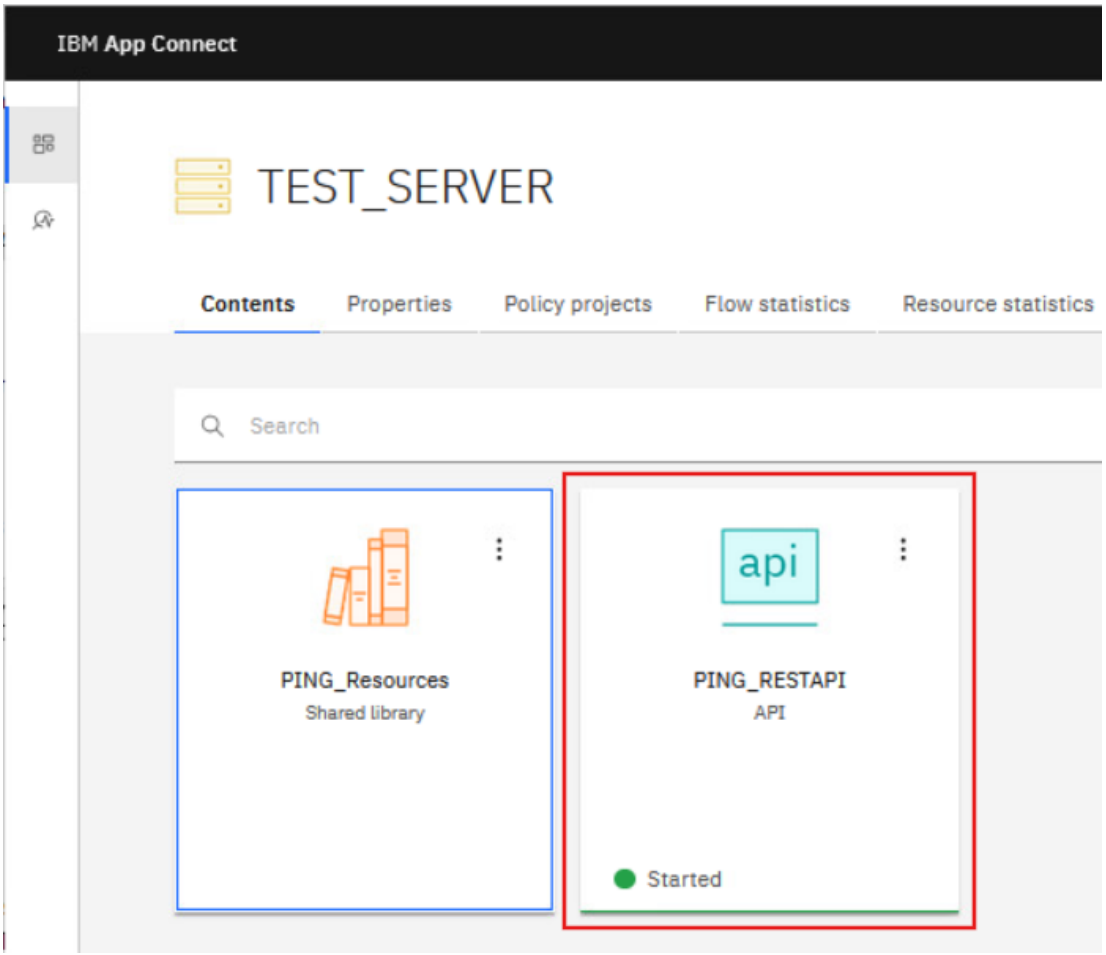
In this next section you will see how to use the ACE Toolkit Flow Exerciser with a REST API.

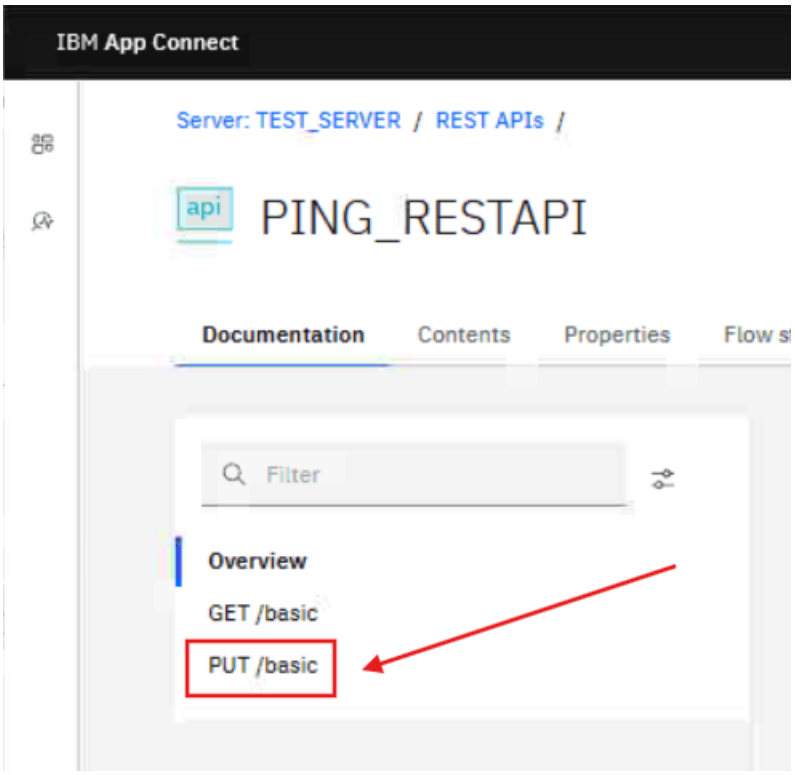
1.	In the Application development window, expand PING_RESTAPI until you see the PING_RESTAPI.msgflow file. Right click on `PING_RESTAPI.msgflow` and select **Open With** > **Message Flow Editor** from the list of options: 

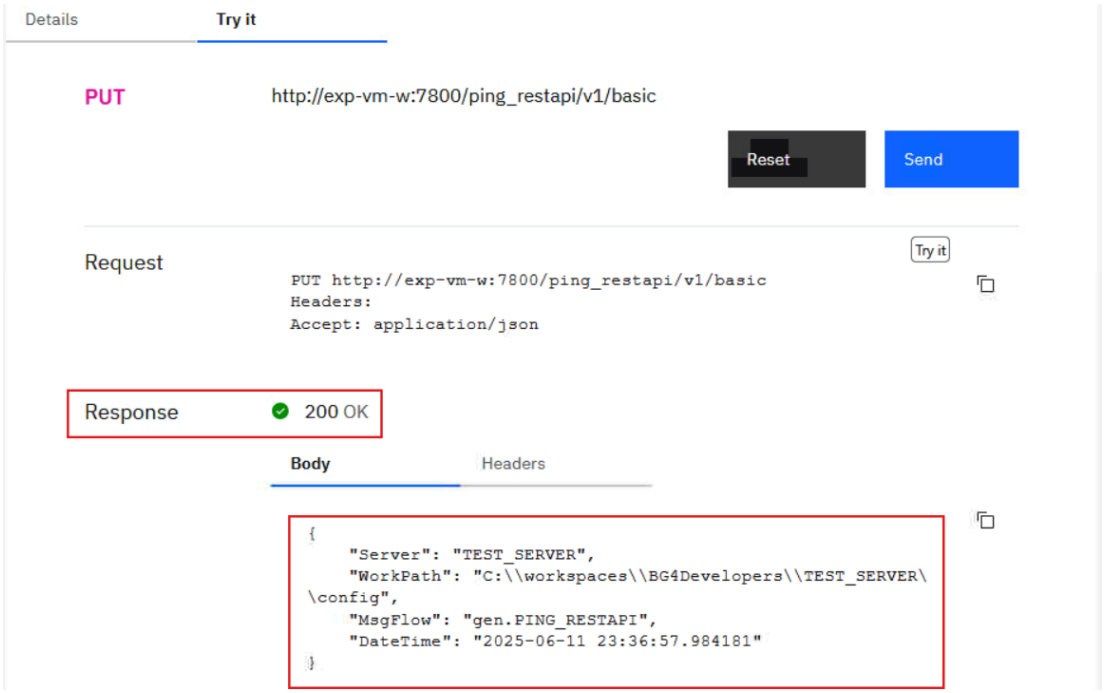
2.	The message flow implementation for the REST API will open. **Note:** The warning about not updating this file. This message flow is automatically created (and recreated) when the configuration and implementation of the REST API is changed using the ACE Toolkit. Click the triangle shaped icon at the top of the palette to close it: 
3.	Click the Flow Exerciser record button 
4.	Select No from the Confirm redeploy prompt: 

5.	Review the content of the “Ready to record message” window and click Close: 
6.	The message flow will now have a blue line around the perimeter of the window and it will say it is “Recording…”: 

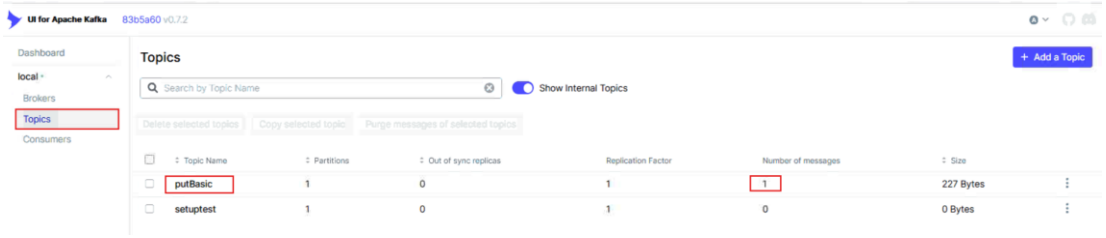
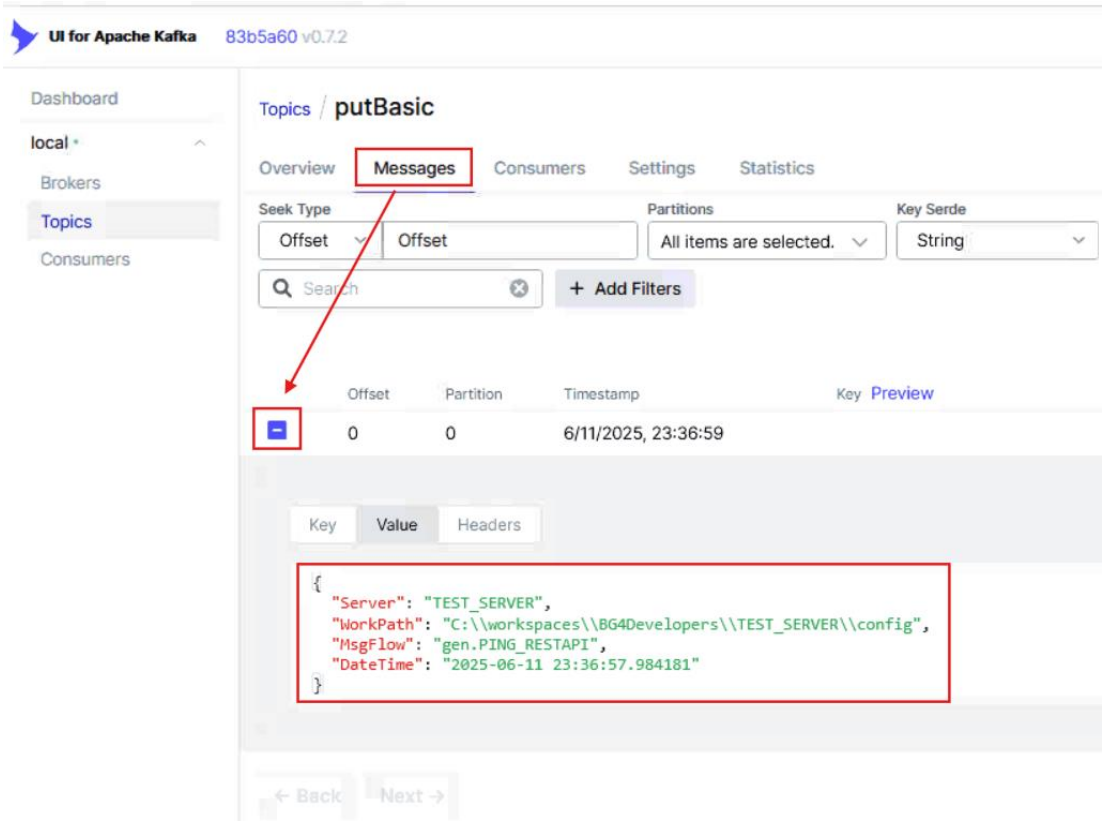
5.5.1 Use Admin WebUI to Send a PUT Request

1.	Switch to the browser window you left open with the ACE Web Admin UI running and reload the current page (Ctrl +R). Return to the main IBM App Connect window (http://exp-vm-w:7600): 
2.	Click the PING_RESTAPI tile: 

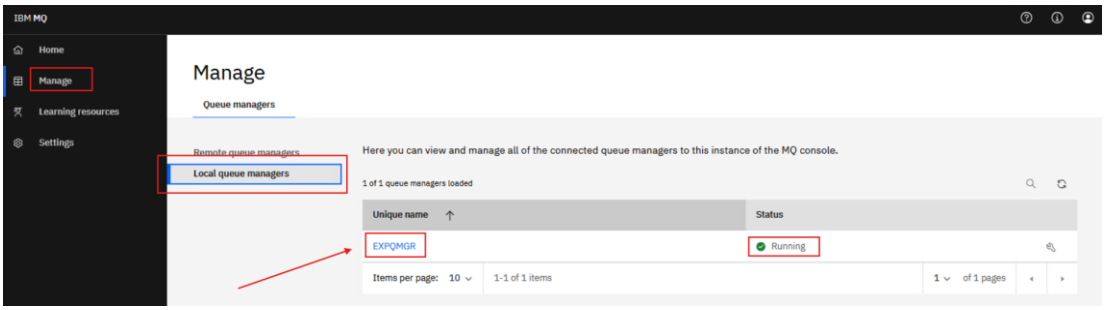
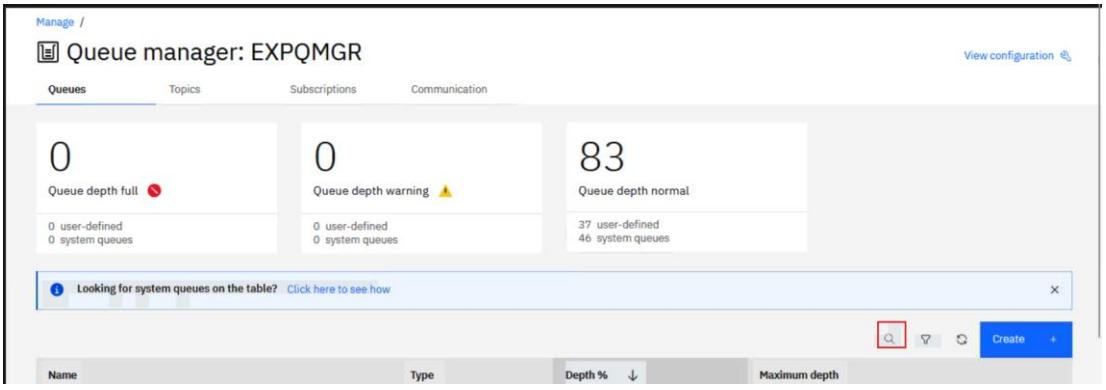
3.	Note the Put /basic operation now exists. Click the PUT operation: 
4.	On the **Try it** tab, click the **Send** button: 

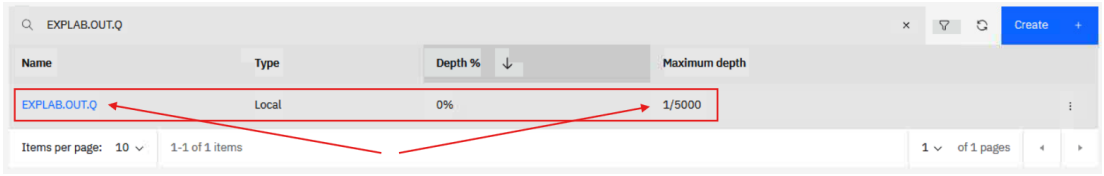
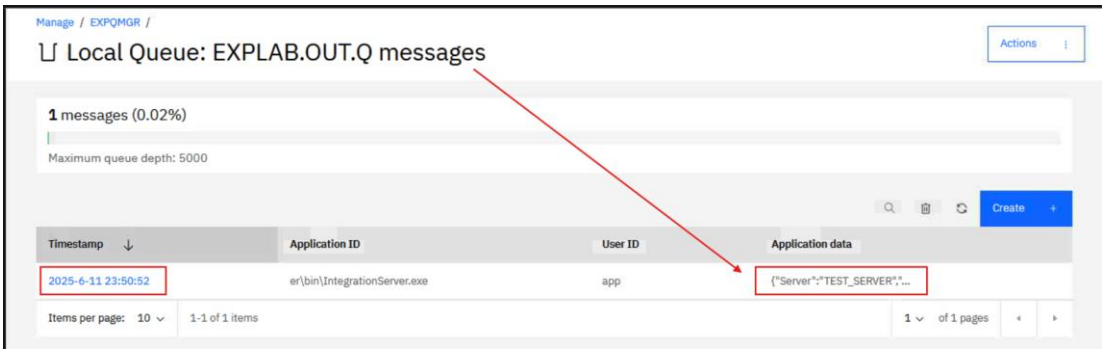
5.	After a few seconds you should receive a Response 200 OK from the request, with the body of the message containing the Server information: 

5.5.2 Verify Kafka Topic

1.	Switch to the browser tab that you have open with the UI for Apache Kafka: Reload the UI page. Navigate to **All Topics** > **putBasic** and click on the putBasic Topic name: 
2.	Click the Messages tab to see the content of the response message that was also written to the Kafka Topic: 

5.5.3 Verify MQ Queue

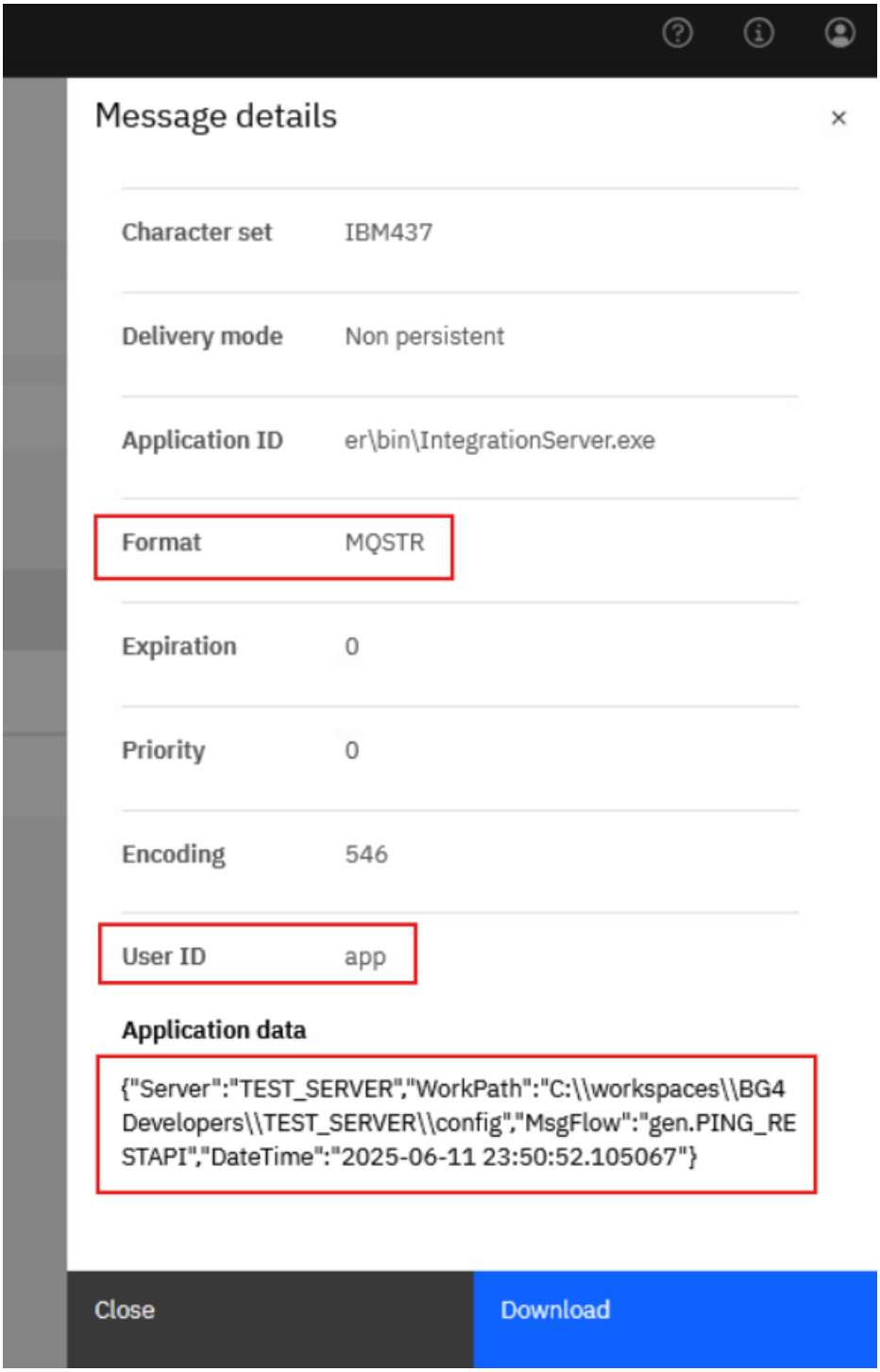
1.	Switch to the MQ Console (if you need to log in again use admin/passw0rd) (URL: https://exp-vm-l:9443/ibmmq/console/login.html) 
2.	Select Manage > Local queue managers > EXPQMGR (make sure the Queue manager is running): 
3.	All the queues defined to the queue manager will be shown. Click on the magnifying glass and search for `EXPLAB.OUT.Q` (the queue defined on the MQ Output node): 

4.	Note the queue depth is 1. Click the queue name to see the messages in the queue: 
5.	The same message will be shown on the queue: 

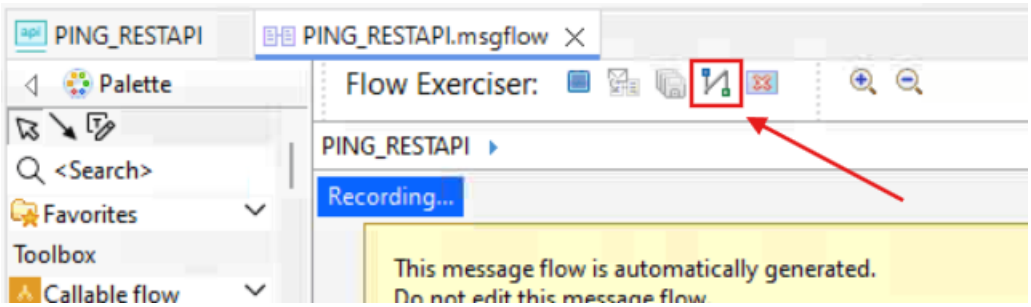
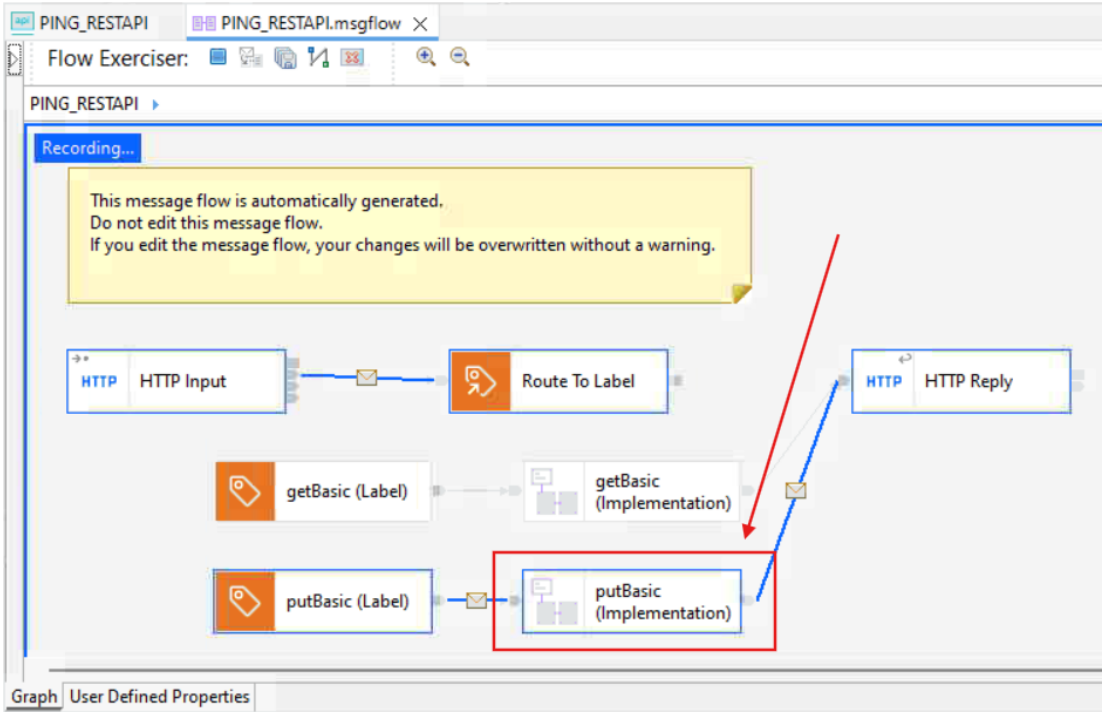
6. Click the message to see the content:

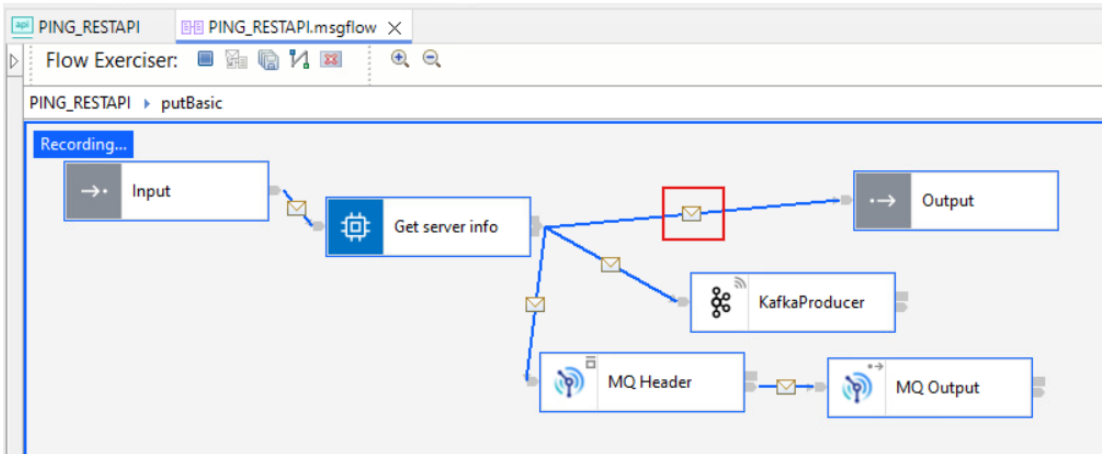
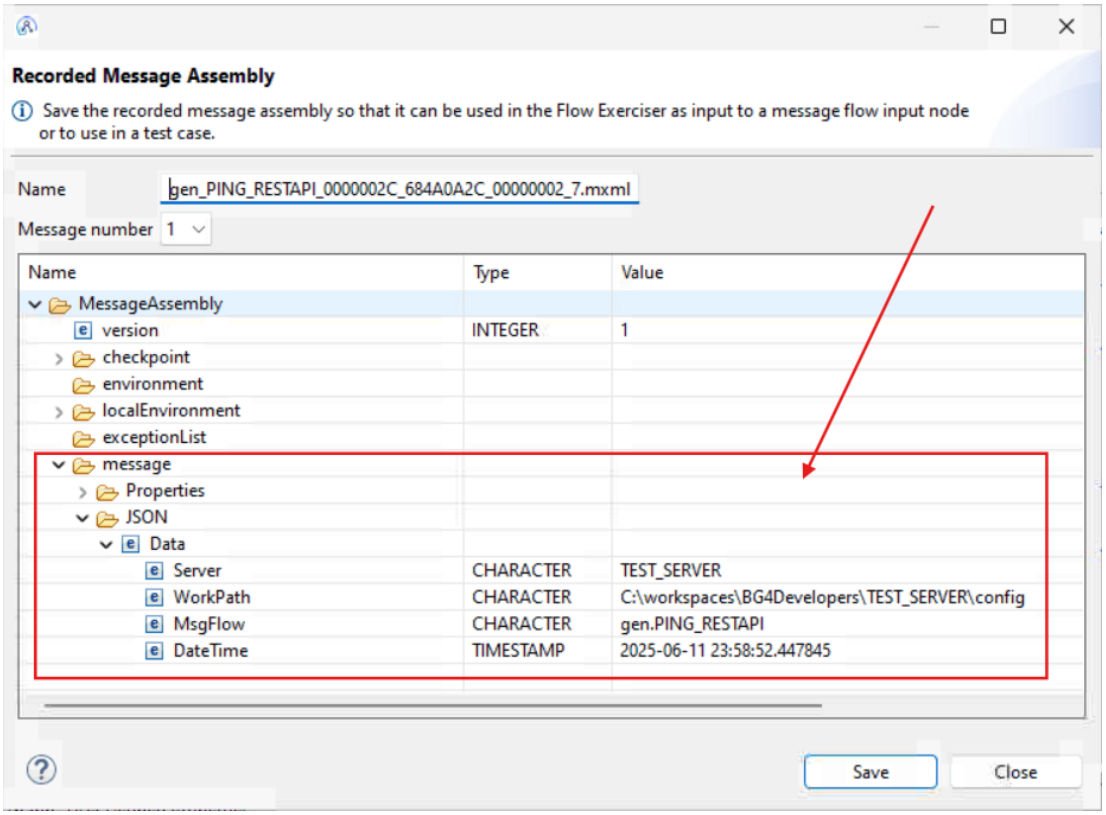
The value of MQSTR for the Format message property is from the MQ Header configuration in the PUT subflow flow.

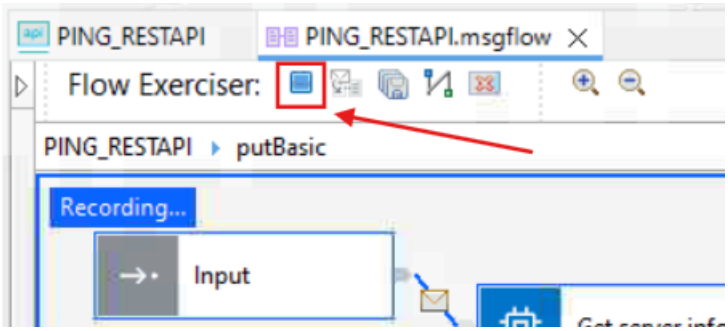
The User ID of app was configured in the EXPQMGR_CLIENT MQEndpoint Connection Policy configured on the MQ Output node:



5.5.4 Review recorded Messages in Flow Exerciser

1.	In the Flow Exerciser window (ACE Toolkit), click on the icon to show the path of recorded messages: 
2.	The route the message took through the message flow is highlighted in blue. Double click the putBasic (Implementation) subflow: 

3.	The route that the message took thorough this subflow is also highlighted: Click the envelope on the connection between the Get server info Compute node and the Output node: 
4.	A window will open showing the Recorded Message Assembly:  Note it is possible to save this message assembly so that you can use it in Unit Testing (a deeper subject than will be covered in this lab).
5.	Close the Recorded Message Assembly window.

6.	Stop the Flow Exerciser: 
7.	Close the PING_RESTAPI.msgflow file.

END OF LAB GUIDE