

ACTIVIDADES PASO A PASO REALIZADAS EN PRUEBA TÉCNICA

AUTOR: DE LA CRUZ FLORES DANIEL

FECHA: 11/11/2023

Build your first Hello Mule application.

1. El primer paso será ejecutar la aplicación *AnypointStudio.exe* y posteriormente crear un “workspace” que será la ubicación dentro del equipo donde se almacenarán los archivos de Anypoint Studio. En la **Figura 1** se muestra la ventana donde ingresarás la ruta y el nombre de tu workspace.

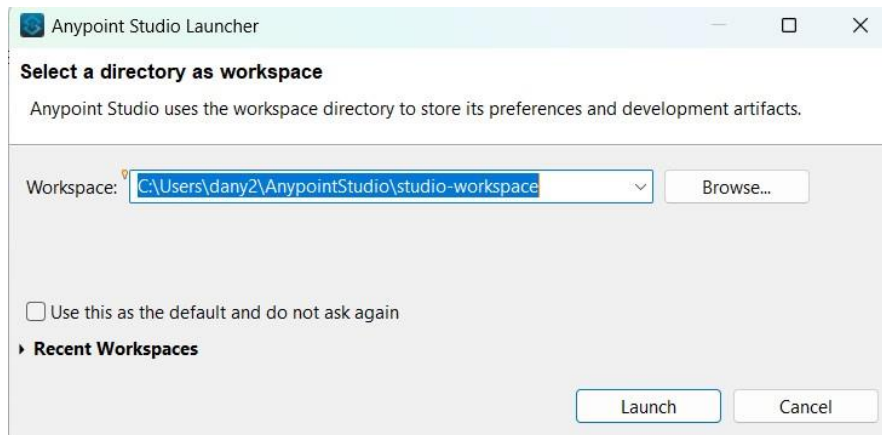


Figura 1. Crea tu workspace.

2. Posteriormente debes crear un proyecto en MuleSoft IDE, para ello, da clic en el botón *File* que se encuentra en la parte superior izquierda de la ventana de AnyPoint Studio, *posteriormente* en *New* y por último en *New Project*. Para mayor referencia puedes consultar la **Figura 2**.

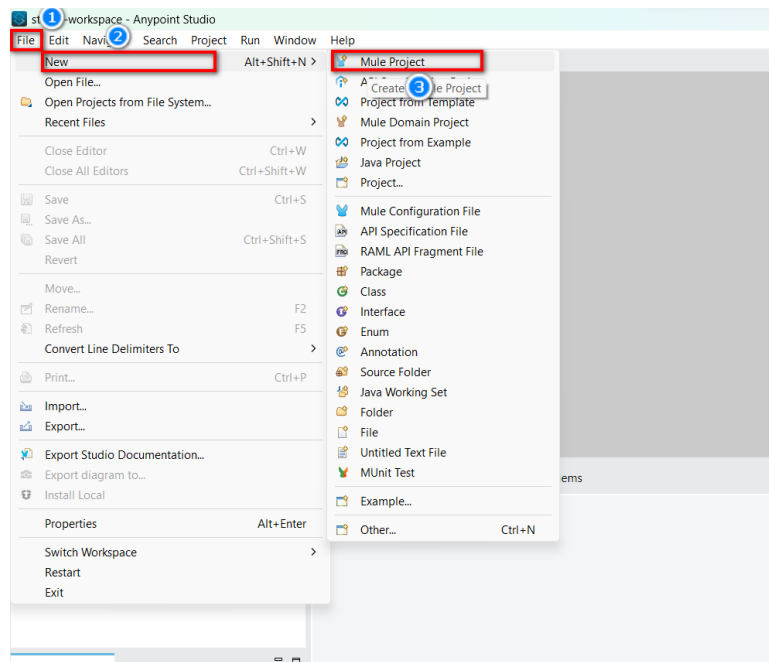


Figura 2. Crear nuevo proyecto.

3. Se desplegará una ventana como la que se muestra en la **Figura 3**, en ella deberás ingresar el nombre de tu proyecto en el campo *Project Name* y posteriormente dar clic en *Finalizar*.
- En la **Figura 3** se indica la localización de estos campos, así como el orden en el que se debe de realizar la ejecución

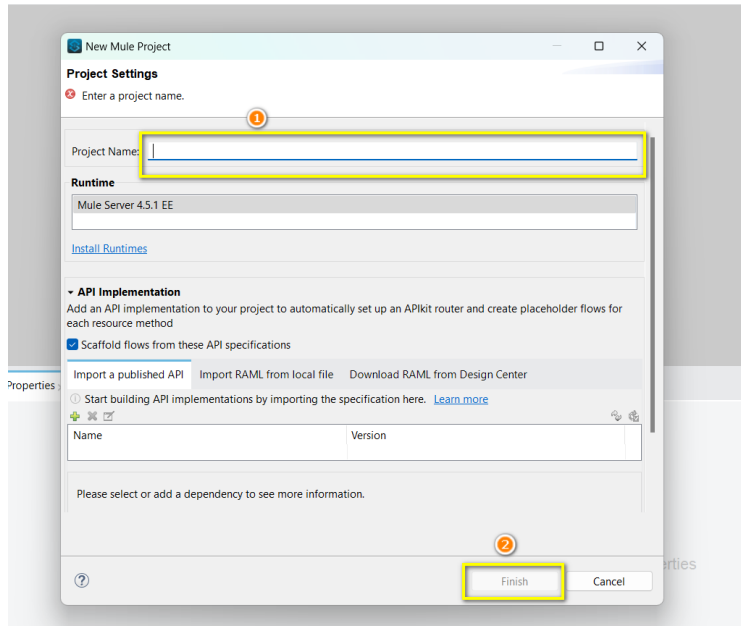


Figura 3. Asignar nombre proyecto.

4. Posteriormente dirígete a la *Mule Palette* que se encuentra del lado derecho del IDE, selecciona *HTTP*, selecciona *Listener* y arrástralo hacia el *Canvas*. En la **Figura 4** muestra una referencia.

Nota: Mule Palette contiene herramientas para la integración de aplicaciones. Estas herramientas representan diferentes componentes y conectores para construir flujos de integración.

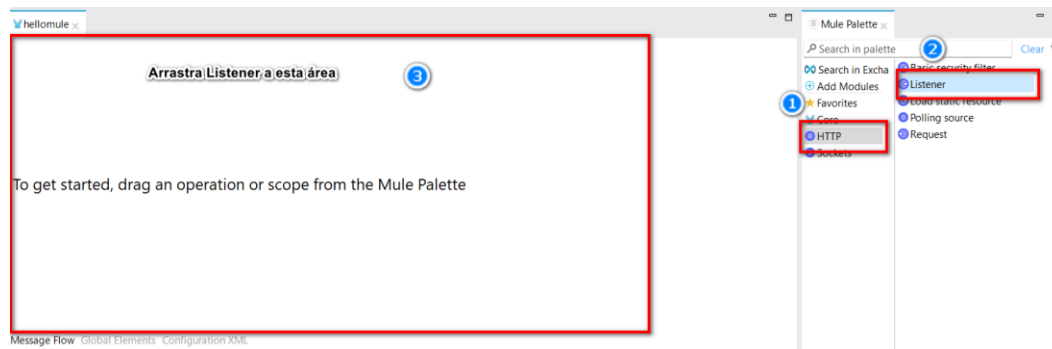


Figura 4. Agregar Endpoint HTTP Listener.

Nota: HTTP Listener permite que las aplicaciones reciban solicitudes HTTP entrantes y permite que los flujos de integración respondan a las solicitudes.

5. El siguiente paso es editar las propiedades del conector HTTP Listener, para ello, al dar doble clic en el conector que se colocó se muestran las propiedades del editor (véase **Figura 5**)

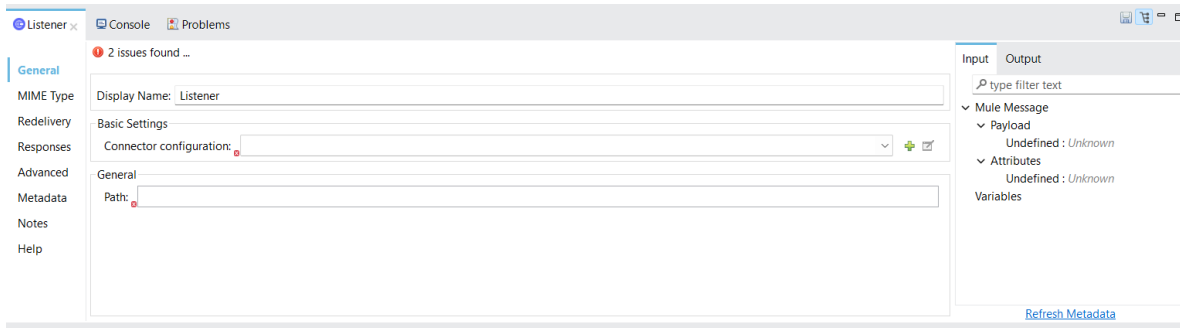


Figura 5. Propiedad del editor.

6. Posteriormente se crea el archivo de configuración en la configuración de elementos globales, el primer paso para ello es dar clic en el botón “+” de color verde que se indica en la **Figura 6**.

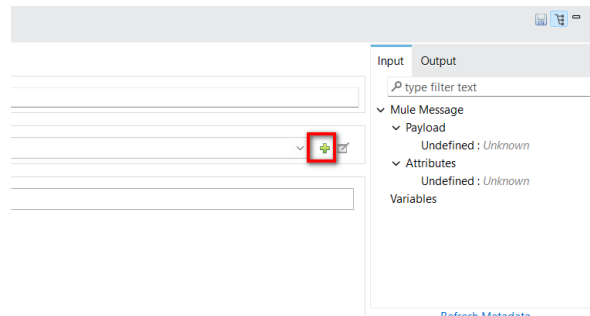


Figura 6. Crear archivo de configuración.

7. Esto muestra la configuración del elemento HTTP Listener. En esta configuración se dejan los valores que están por defecto. Para el Host se utilizó 0.0.0.0, mientras que para el puerto se establece el 8081. Tal como se muestra en la **Figura 7**.

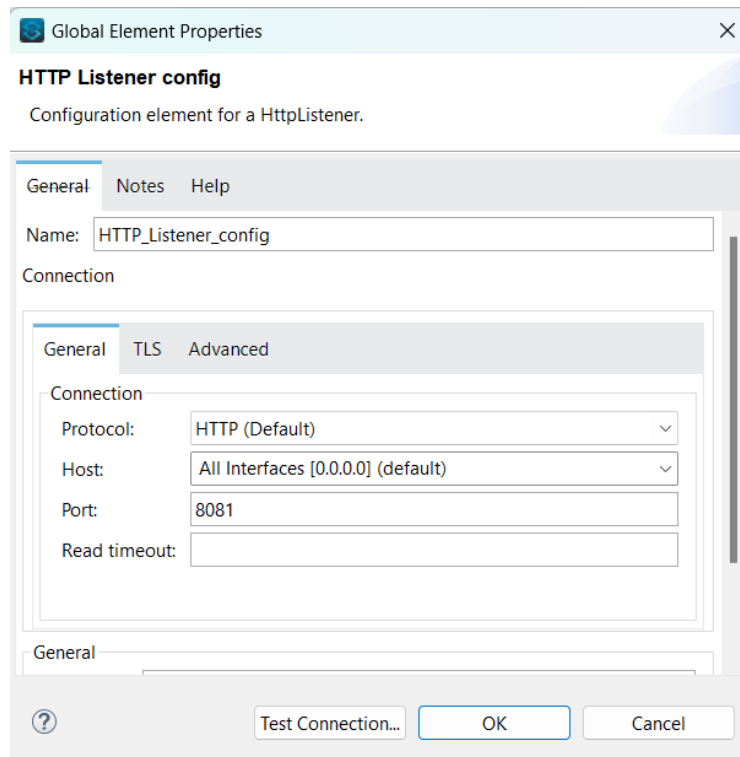


Figura 7. Configuración HTTP Listener.

8. Se añade el Path que representa el Endpoint que se ejecutará cuando se haga la petición HTTP hacia el HTTP Listener.

Nota: Con fines prácticos se añadió /helloMule como Path.



Figura 8. Se agrega el Path.

9. El siguiente paso es agregar el conector Set Payload en el flujo que estamos trabajando, el Set Payload se encargará de establecer el contenido del mensaje. Este conector se localiza en el Mule Palette dentro del apartado Core. Para mayor referencia véase la **Figura 9**.

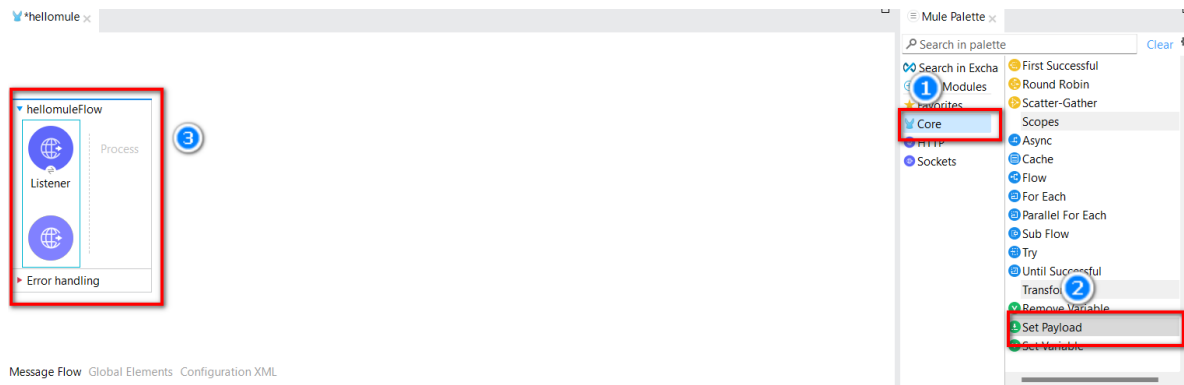


Figura 9. Agregar Set Payload.

10. Se despliega el apartado de propiedades de Set Payload en la parte inferior de la ventana, primero se deselectionará *fx* y posteriormente se insertará el valor del mensaje (véase **Figura 10**).

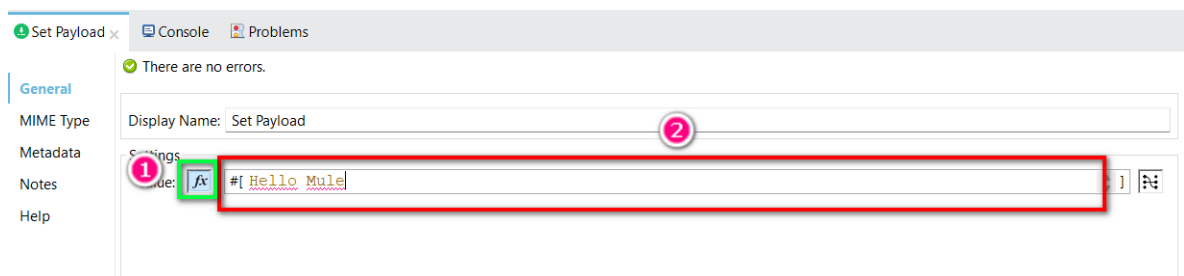


Figura 10. Establecer mensaje en Payload.

11. Se guarda el archivo seleccionando *File* en la parte superior izquierda y luego *Save* (**Figura 11**).

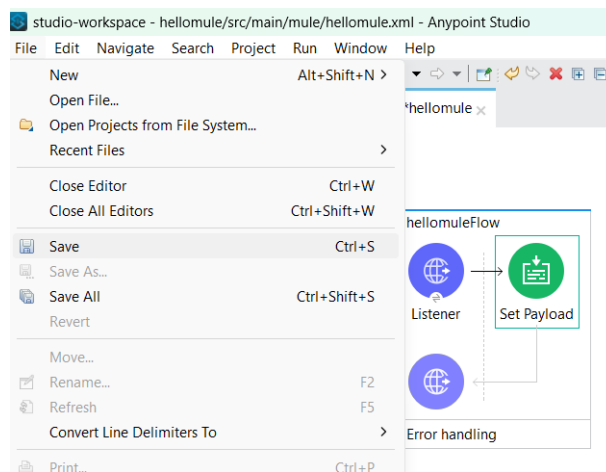


Figura 11. Guardar el proyecto.

12. El siguiente paso es ejecutar el proyecto, para ello se dará clic derecho en el elemento Canva y se selecciona la opción *Run project hellomule*. (**Figura 12**).

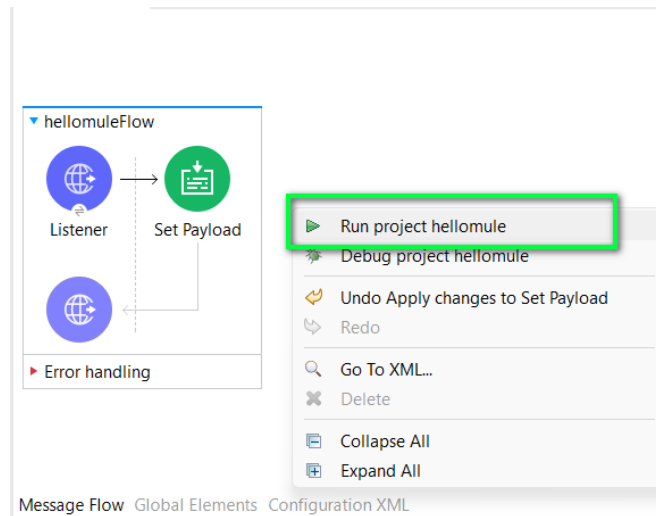


Figura 12. Ejecución de proyecto.

13. Se inicia la ejecución del programa, en el elemento consola se muestran los Logs para saber el estatus del proceso de ejecución. (**Figura 13**).

```
INFO 2023-11-11 00:12:39,296 [WrapperListener_start_runner] org.eclipse.jetty.server.AbstractConnector: Started ServerConnector@446d79532{HTTP/1.1, (h
INFO 2023-11-11 00:12:39,300 [WrapperListener_start_runner] org.mule.runtime.core.internal.logging.LogUtil:
*****
* - - + DOMAIN + - - * - - + STATUS + - - *
*****
* default * DEPLOYED *
*****
*****
* - - + APPLICATION + - - * - - + DOMAIN + - - * - - + STATUS + - - *
*****
* hellomule * default * DEPLOYED *
*****
INFO 2023-11-11 00:12:39,446 [[MuleRuntime].uber.03: [hellomule].uber@org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource
```

Figura 13. Logs proceso de ejecución.

14. Se abre Postman y se realiza la configuración correspondiente.

El primer punto para configurar es la URL. En este caso, la URL esta compuesta por el host 0.0.0.0 o localhost, el puerto 8081 y el path /hellomule

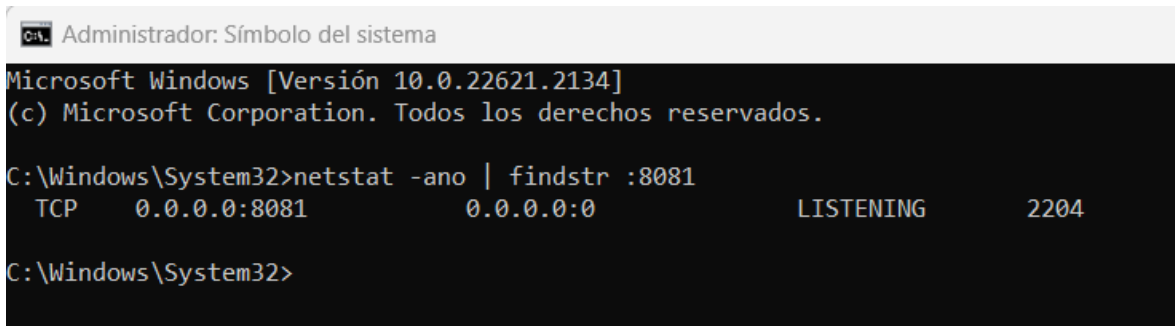
En la **Figura 14** se puede ver esto.



Se tienen distintos métodos dependiendo de lo que vayamos a realizar (GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS)

En este caso, debido a que nosotros vamos a hacer una petición para que nos devuelva información se utilizará GET

Nota: En este caso el puerto 8081 ya estaba siendo ocupado por otra aplicación, por lo tanto, se accedió al CMD para encontrar la aplicación que estaba haciendo uso del puerto y detenerla. En la **Figura 15** se muestra el comando que se utilizó para encontrar la tarea que estaba haciendo uso de este puerto.



```
Administrador: Símbolo del sistema
Microsoft Windows [Versión 10.0.22621.2134]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Windows\System32>netstat -ano | findstr :8081
TCP      0.0.0.0:8081          0.0.0.0:0             LISTENING        2204

C:\Windows\System32>
```

Figura 15. Puerto ocupado.

15. Al dar clic en *Send* se realiza la petición. Postman nos regresa la respuesta y se puede observar en la **Figura 16**.

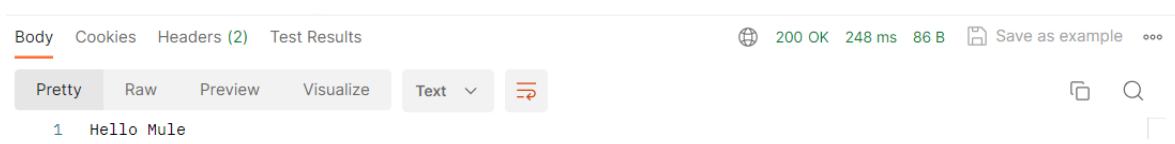


Figura 16. Respuesta petición.

En este caso se obtuvo la cadena esperada (Hello Mule), así como una respuesta de estado satisfactorio HTTP 200 OK, que indica que la solicitud ha tenido éxito.

16. El siguiente paso es desplegar la aplicación en CloudHub, para ello primero se detiene la aplicación.

Nos dirigimos al package explorer que es el panel que se encuentra del lado izquierdo de nuestra área de trabajo, ahí damos clic derecho a *hellomule.xml* que es nuestro flujo, seleccionamos *Anypoint Platform* y *Deploy to CloudHub*, tal como se indica en la **Figura 17**.

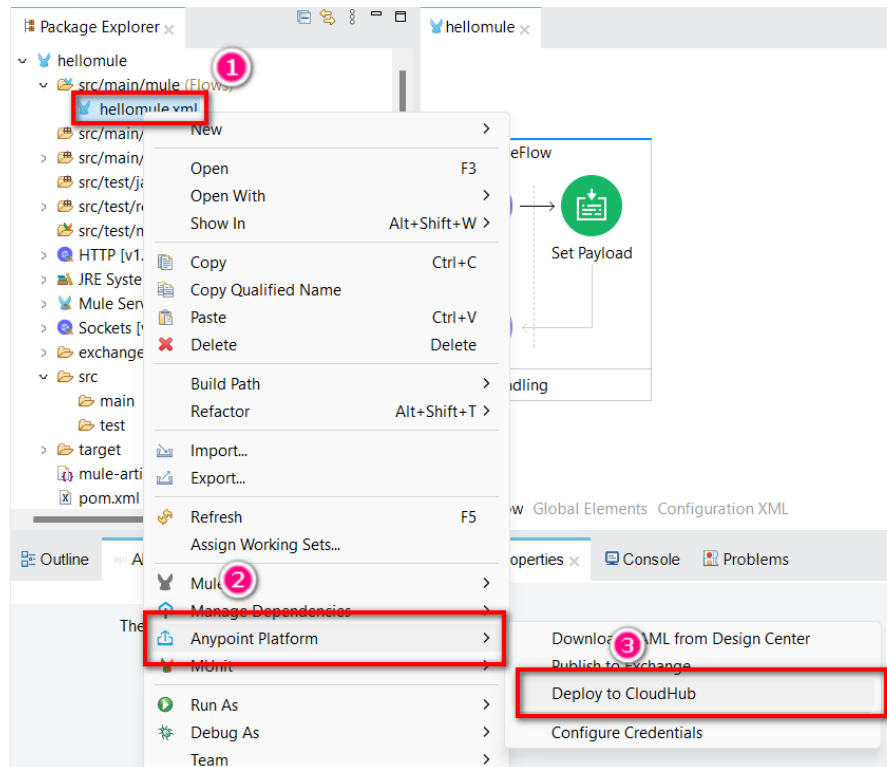


Figura 17. Desplegar en CloudHub.

17. El siguiente paso es llenar la información del formulario que se muestra en la **Figura 18**, se coloca el nombre que se desea que tenga la aplicación y se da clic en el botón **Deploy Application**.

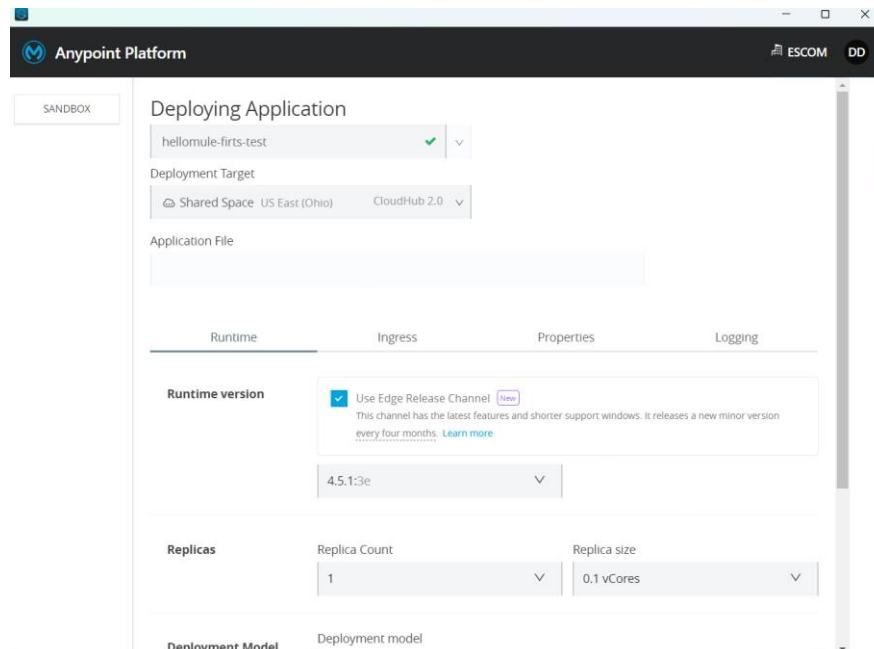


Figura 18. Configurar aplicación CloudHub.

Nota: Se modificó el *Deployment Target*, ya que originalmente se tenía en CloudHub 2.0

18. En la plataforma se ingresa a la opción de Runtime Manager, seleccionamos SandBox que es el tipo de ambiente de nuestra aplicación

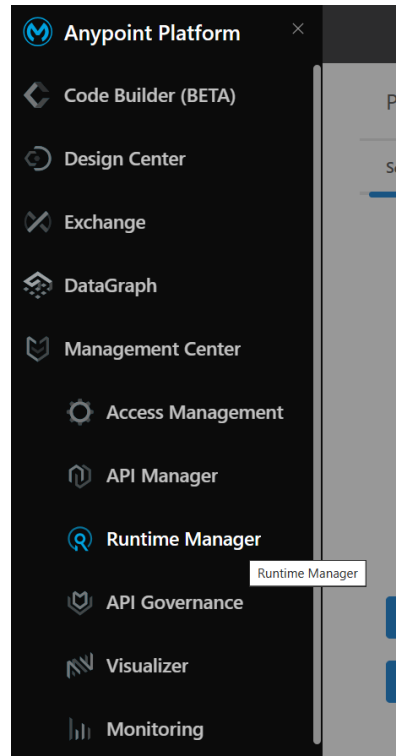


Figura 19. Runtime Manager.

19. Se accede a los logs de la aplicación para verificar que este en funcionamiento

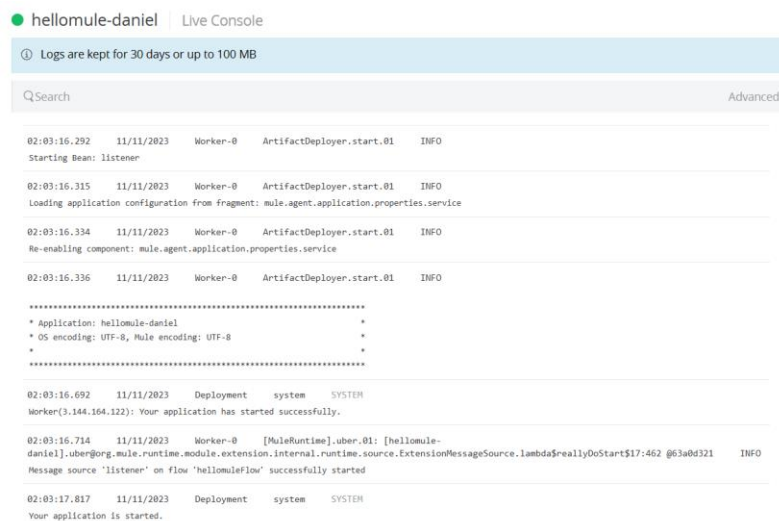


Figura 20. Logs aplicación.

20. En *settings* copiamos la URL (**Figura 21**) y la pegamos en Postman, tal como se muestra en la **Figura 22**.

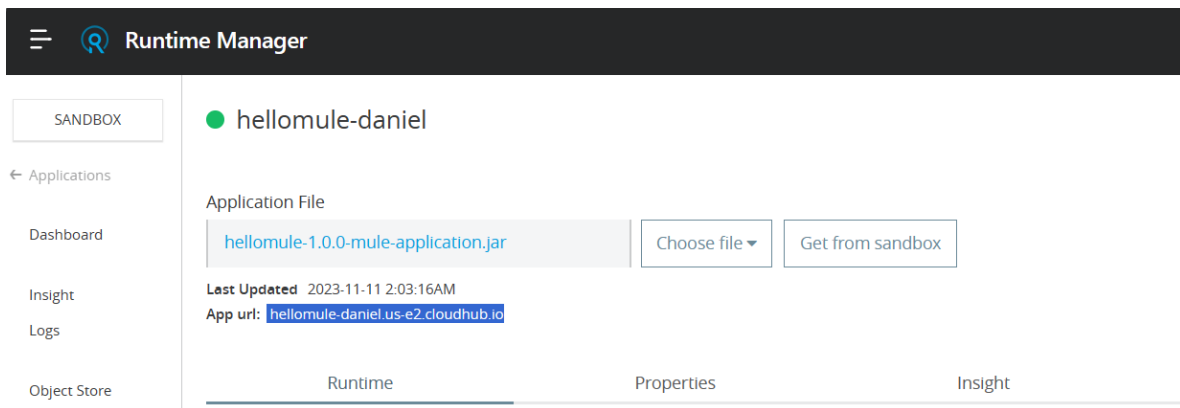


Figura 21. Copiar link App.

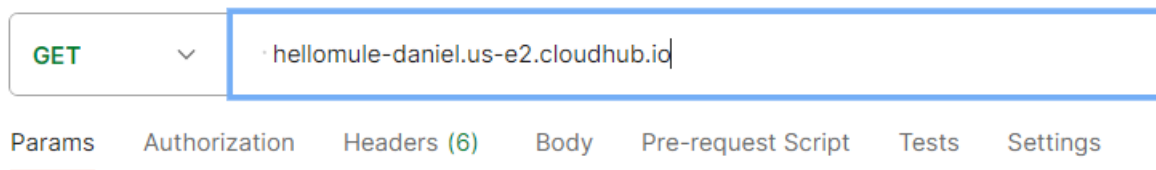


Figura 22. Pegar link en Postman.

Nota: Al termino de la URL se debe agregar el Path (/hellomule)

21. Se realiza la petición en Postman y se obtiene el resultado. (**Figura 23**)

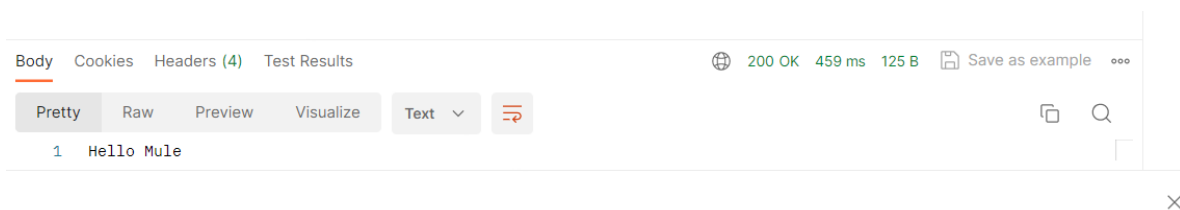


Figura 23. Resultado Postman.

Cómo configurar sus elementos globales y archivos de propiedades en Anypoint Studio

22. Se crea un archivo para almacenar todos los elementos globales (global.xml), así tendremos un mejor control de ellos pues todos estarán en un mismo archivo y no habrá múltiples. Véase **Figura 23**.

El nombre que se le asignará al archivo será *global.xml*

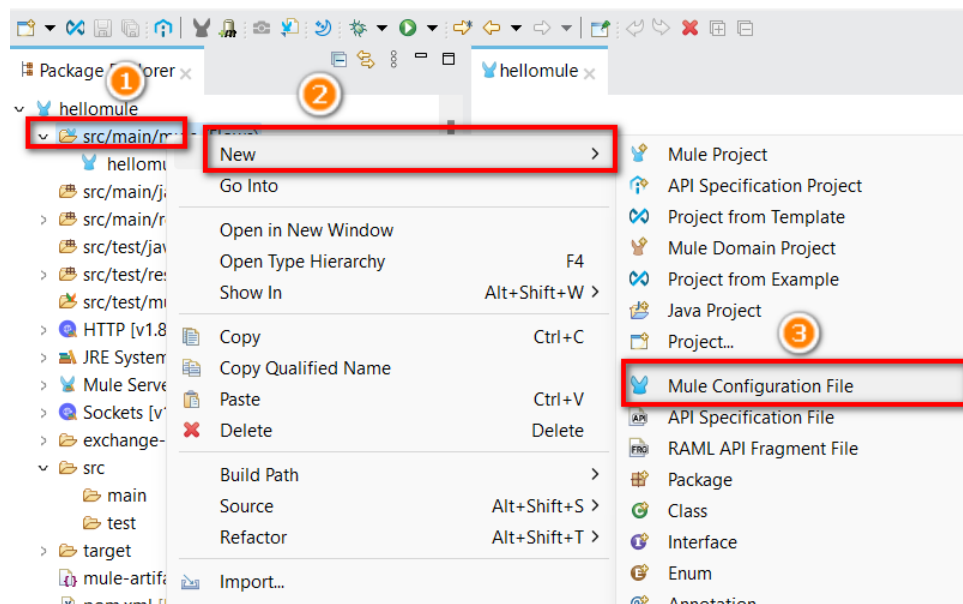


Figura 23. Crear archivo de configuración.

23. En el xml de hellomule se edita y se elimina todo el tag *http:listener-config*, tal como se indica en la **Figura 24**. Este tag se pegará en el archivo que se creo en el paso anterior.

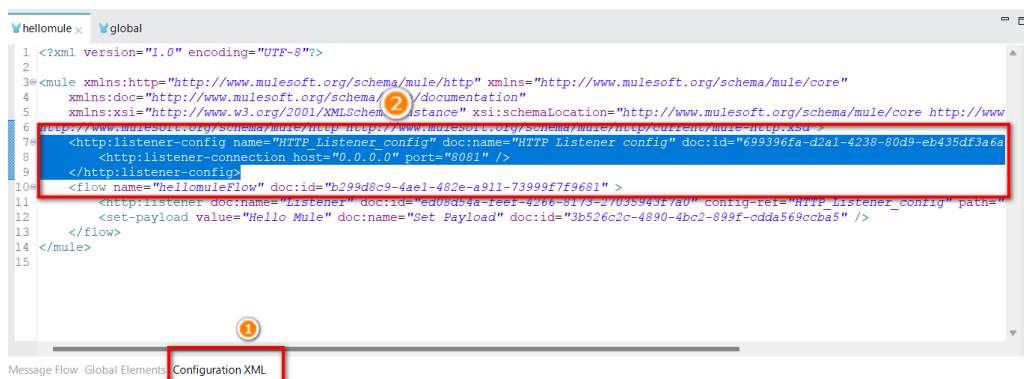


Figura 24. Eliminar tag hellomule xml.

24. Abre la configuración XML del archivo *global* y pega el tag que se eliminó de *hellomule* (Figura 25).



Figura 25. Agregar tag global xml.

25. Se crean archivos de propiedades para el elemento Listener, de esta manera no se eliminará el “hardcodeo” que existe al declarar el host y puerto. En la Figura 26 se muestra el proceso que se debe seguir para crear el archivo *.properties*. Se crearán dos archivos, uno llamado *local.properties* y *dev.properties*.

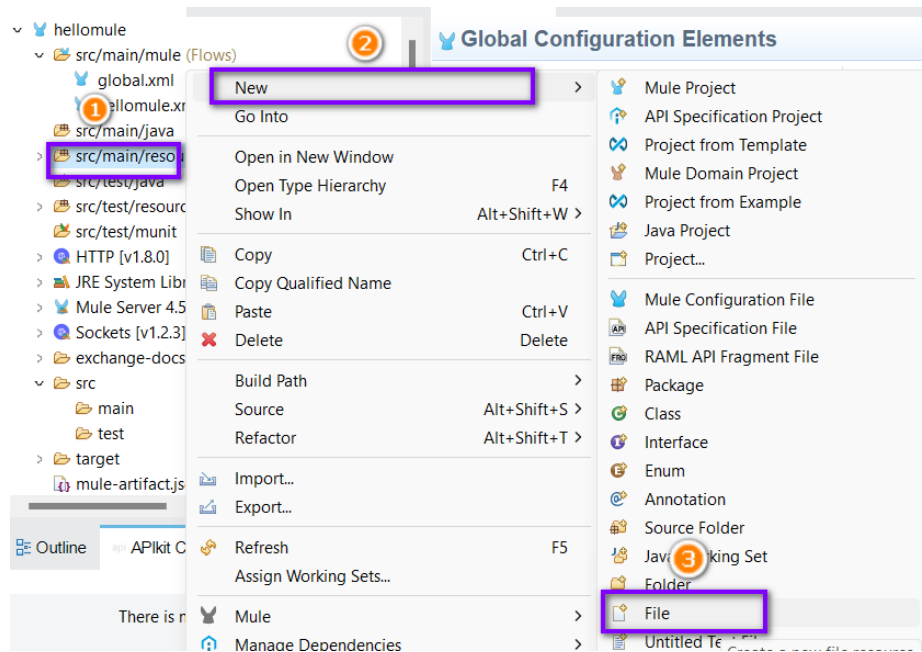
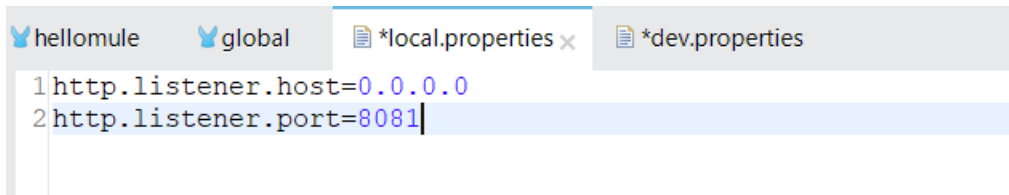


Figura 26. Crear archivo properties.

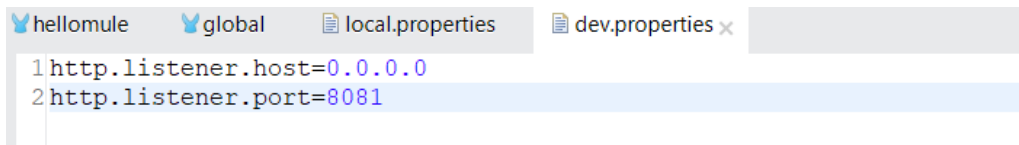
Cada uno de estos archivos tendrá una función diferente, en nuestro entorno local se hará uso de *local.properties*, mientras que al desplegarlo en CloudHub se hará uso de *dev.properties*.

26. Dentro de cada uno de estos archivos creados se deberán insertar los datos referentes al host y al puerto. Véase la **Figura 27** y **Figura 28** para referencia.



```
1 http.listener.host=0.0.0.0
2 http.listener.port=8081
```

Figura 27. Datos host y port, local.



```
1 http.listener.host=0.0.0.0
2 http.listener.port=8081
```

Figura 28. Datos host y port, dev.

27. Posteriormente se cambia el archivo de propiedades del elemento Listener, se edita el HTTP Edit Config.

Se reemplaza el campo host por: `${http.listener.host}`

Se reemplaza el campo port por: `${http.listener.port}`

En la **Figura 29** se muestra lo antes mencionado.

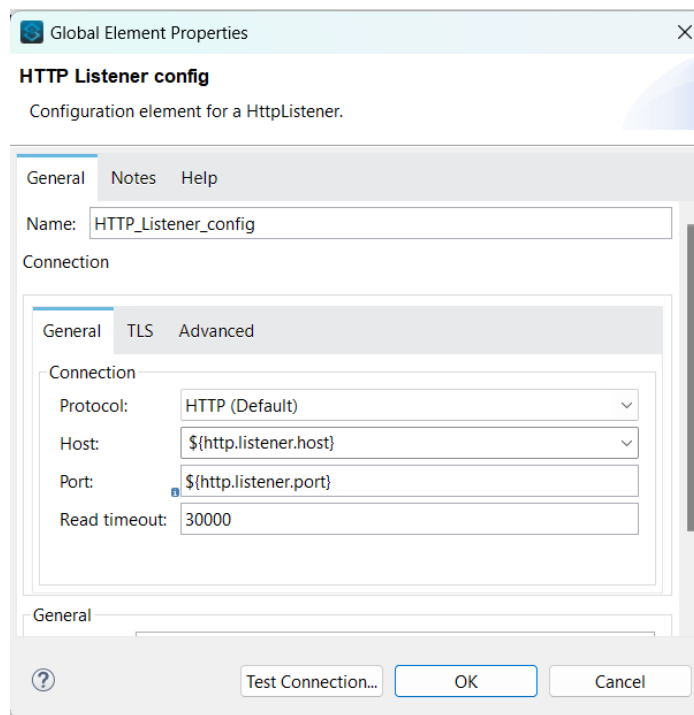


Figura 29. Cambio de valores en campo host y port.

28. El siguiente paso es añadir un elemento que permita que la aplicación Mule sepa en donde esta el archivo de propiedades al momento de ejecución. Para ello nos colocamos en la vista de elementos globales de global.xml (**Figura 30**) y añadimos un elemento dando clic en *Create* y posteriormente buscando y seleccionando *Configuration properties* (**Figura 31**)

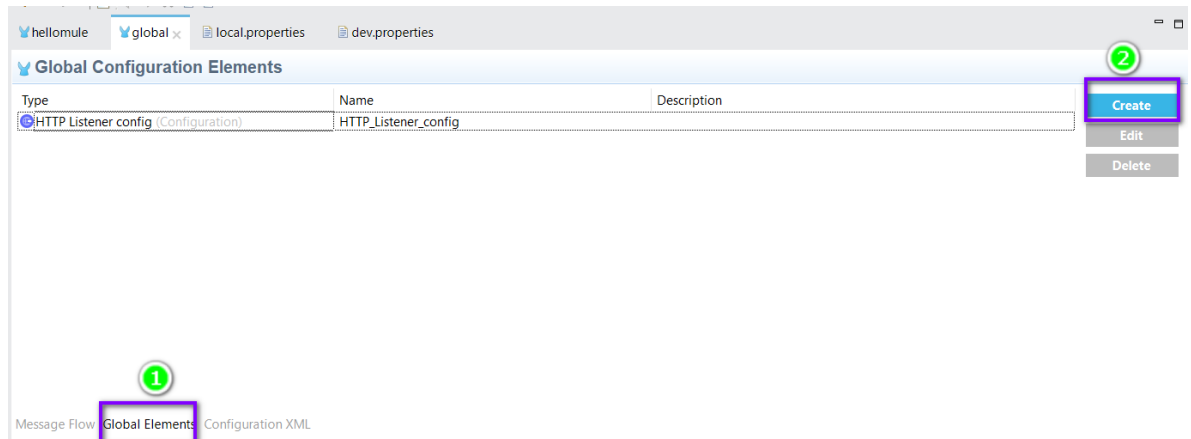


Figura 30. Elementos globales xml.

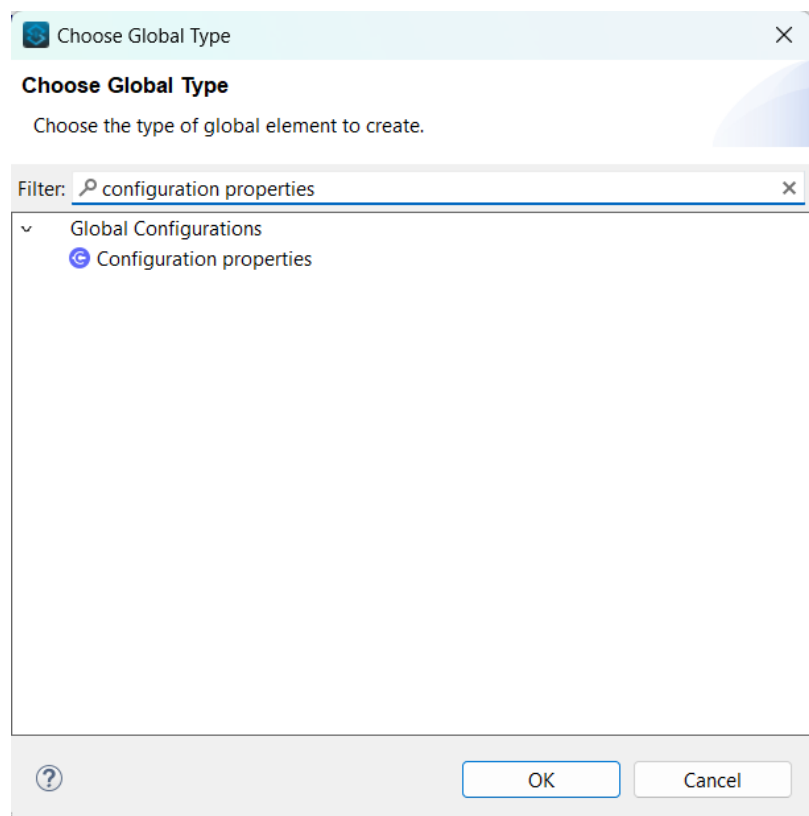


Figura 31. Configuration properties.

29. Debido a que se hará de una manera “dinámica” el archivo deberá poder ir cambiando, dependiendo del entorno en el que se esté utilizando.
Se añade `${env}.properties` dentro de las propiedades de configuración (**Figura 32**)

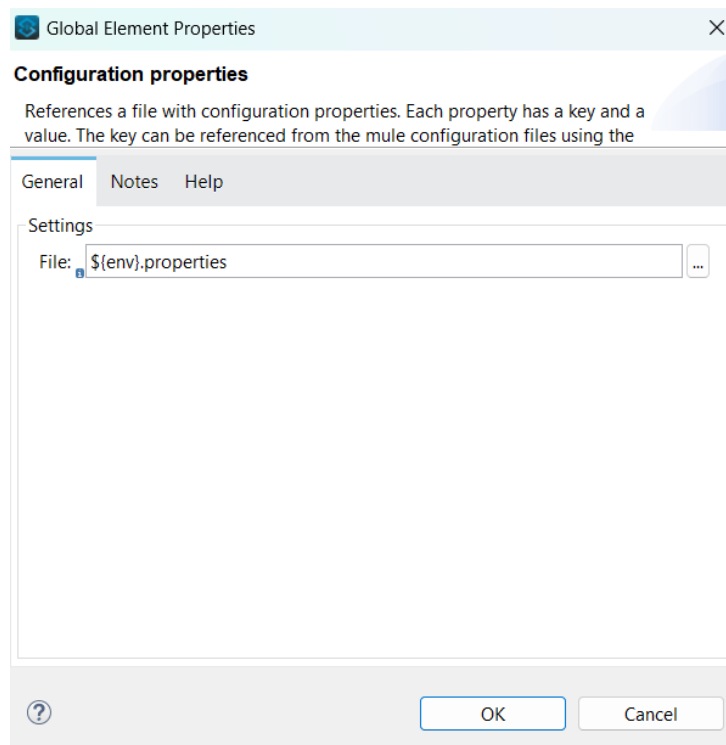


Figura 32. Configuration properties.

30. El siguiente paso es añadir la propiedad `env`, para ello se creará un nuevo elemento, en este caso será *Global property* (**Figura 33**)

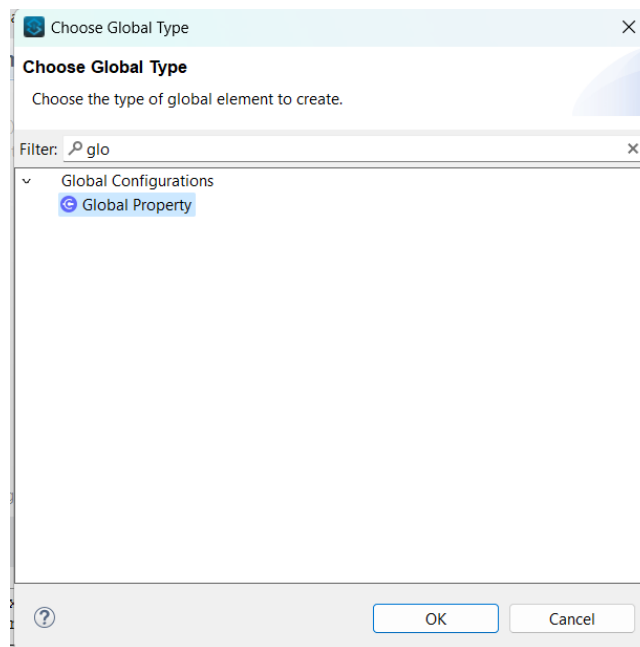


Figura 33. Global property

31. En ella ingresamos el valor *env* y *local* como se muestra en la **Figura 34** que será la configuración que tendrá.

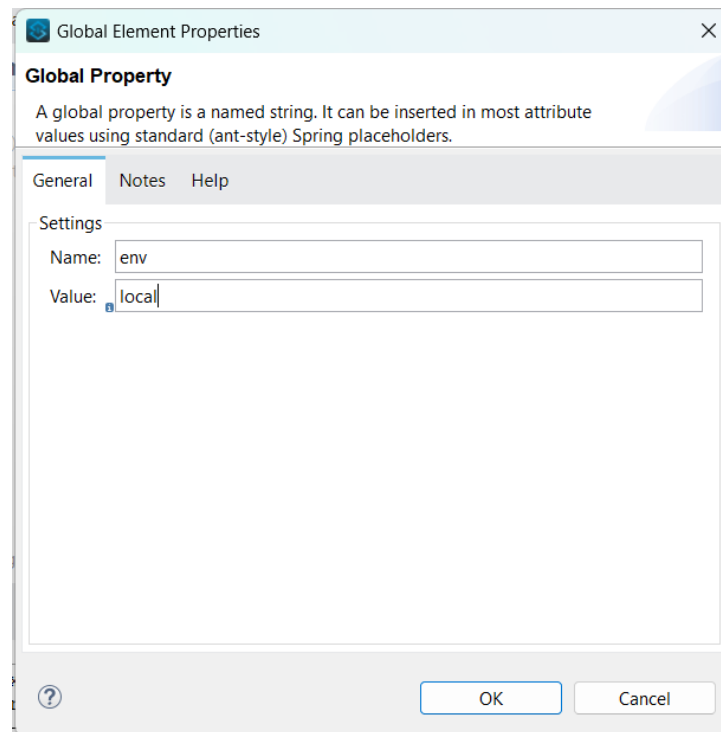


Figura 34. Global property settings.

32. Se vuelve a realizar la ejecución del proyecto Mule (**Paso 12 y 13**)
33. Ingresamos a Postman y se ingresa la URL localhost:8081/hellomule (**Figura 35**) y podemos observar que se realiza la petición de manera correcta, se recibe un código 200 OK y el mensaje mostrado es Hello Mule.

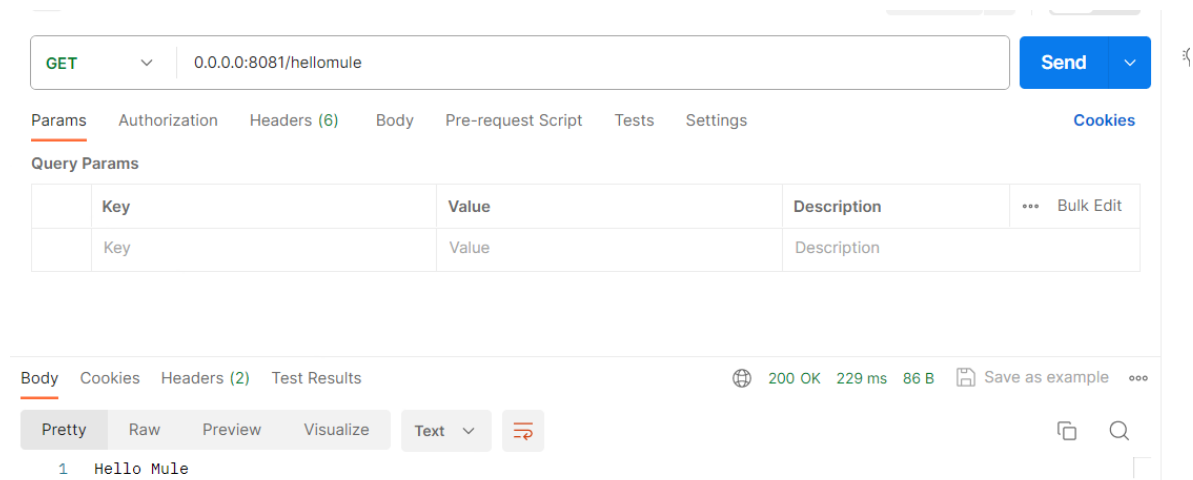


Figura 35. Petición ostman dinámica.

Create your environments' secure properties files

34. Se crearan dos archivos nuevos *.properties* como en el **paso 25**, sin embargo en este caso será llamado *local.secure* y *dev.secure*. En la **Figura 36** se observa la creación del primer archivo.

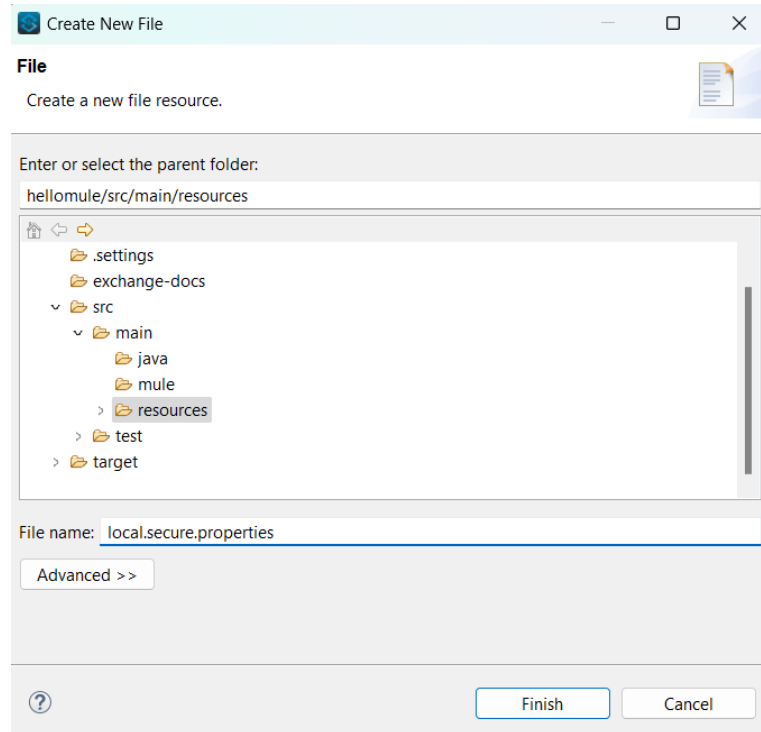


Figura 36. Creación de archivo local secure.

35. Dentro de estos archivos se colocan las propiedades de un “usuario” y “contraseña”, para ello se colocan los valores `example.username=myUsernameLocal` y `example.password=myPasswordLocal` y `example.username=myUsernameDev` y `example.password=myPasswordDev`, dependiendo del archivo. Véase **Figura 37**.

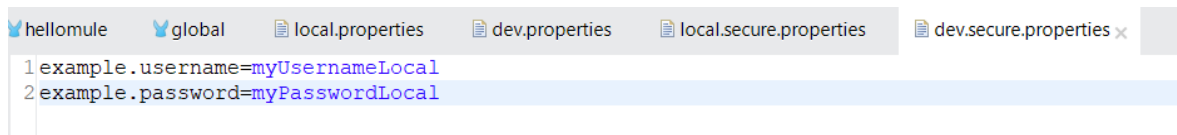


Figura 37. Asignación de propiedades usuario y contraseña.

36. Posteriormente se busca *Mule Secure Configuration Properties* en el *Mule Palette* se selecciona *Search in Exchange* y en el panel que se despliega se ingresa el nombre del módulo y se añade como se muestra en la **Figura 38**.

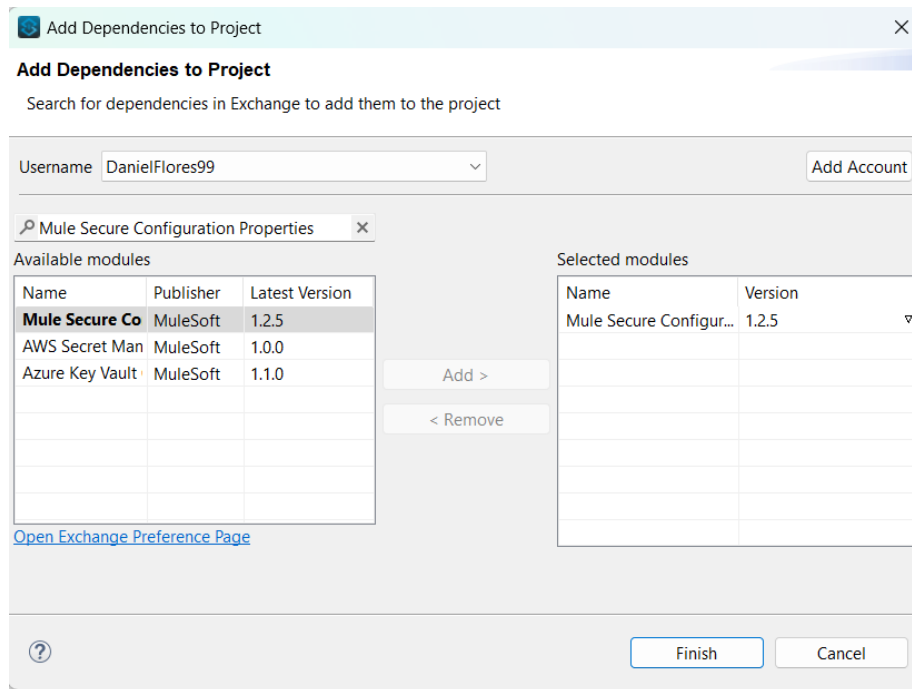


Figura 38. Se añade modulo Mule Secure Configuration.

37. Se crea un *Secure Properties Config* (**Figura 39**).

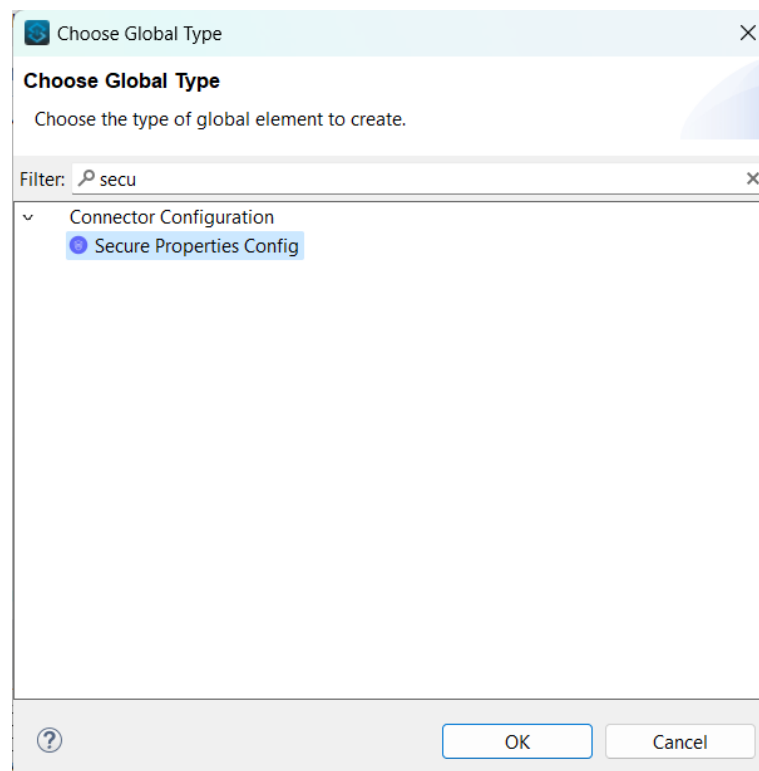


Figura 39. Secure Properties Config.

38. Se realiza la configuración añadiendo los valores `${env}.secure.properties` y `${secure.key}` en los campos File y Key respectivamente, esto hará que sean dinámicos y se pueda adaptar a cualquier entorno. También se selecciona Blowfish que es un algoritmo de encriptación, con la finalidad de mantener protegida la información y no esté al alcance de todos.

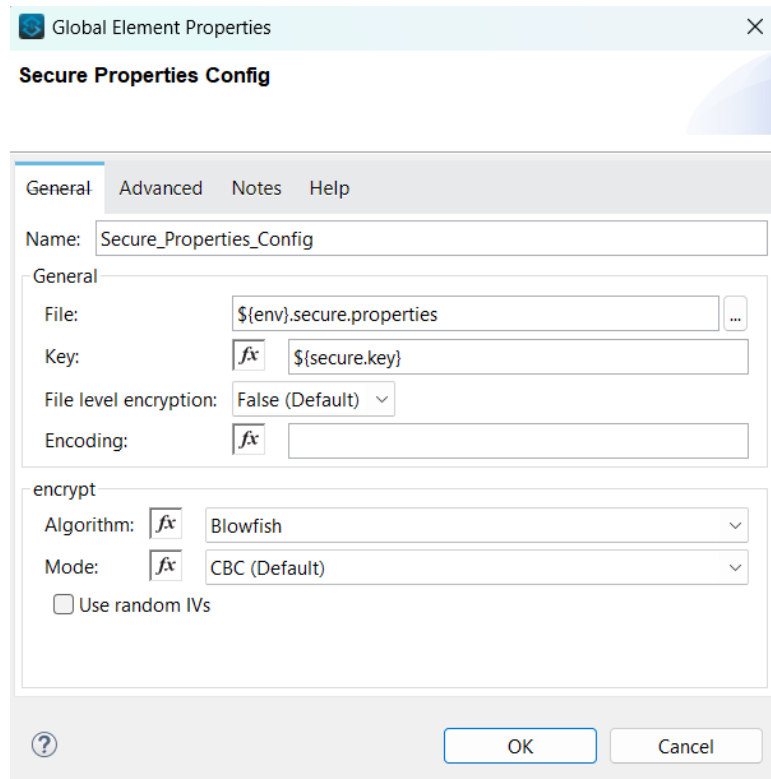


Figura 40. Configuración Secure Properties.

39. De manera similar a cuando se añadió HTTP Listener, se añadirá el componente Logger a nuestro flujo *hellomule*. (**Figura 41**).

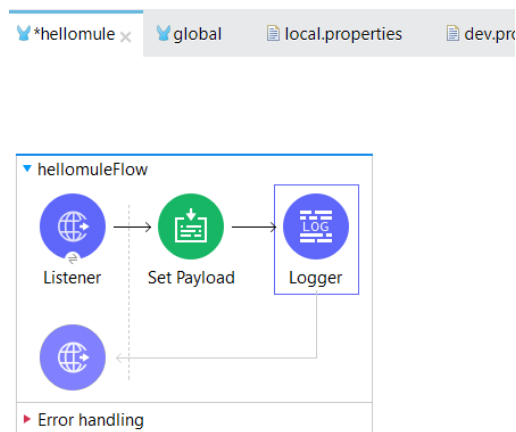


Figura 41. Logger.

40. Se accede a *Show Graphical View* haciendo primero clic en el botón fx

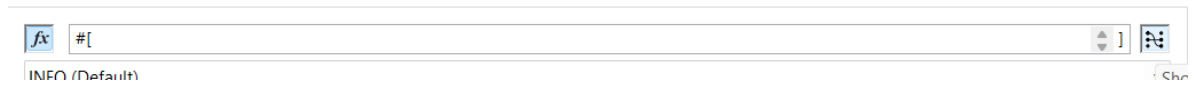


Figura 41. Show Graphical View.

41. Esto nos despliega un panel en el que se añade el código que mostrará las propiedades que fueron configuradas previamente

```
1 output application/java
2 ---
3 "Username: " ++ Mule::p("secure::example.username")
4 ++ " - " ++
5 "Password: " ++ Mule::p("secure::example.password")
```

Figura 42. Output.

42. Se ejecuta el siguiente comando que se encargará de encriptar nuestro usuario y que nos servirá para cifrar y descifrar.

```
java -cp secure-properties-tool.jar com.mulesoft.tools.SecurePropertiesTool \
string \
encrypt \
Blowfish \
CBC \
MyMuleSoftKey \
"myUsernameLocal"
```

```
C:\Users\dany2\Downloads>java -cp secure-properties-tool.jar com.mulesoft.tools.SecurePropertiesTool string encrypt Blow
fish CBC MyMuleSoftKey "myUsernameLocal"
HbsuWJRjiubchmzQREGdsA==
```

Figura 43. Ejecución en consola.

Nota: Esto se repite también con la contraseña que fue colocada en los archivos local.secure y dev.secure.

43. Posteriormente se cambian los valores que estaban colocados en los archivos local.secure y dev.secure por los valores cifrados que se obtuvieron en el paso anterior. (**Figura 44**)

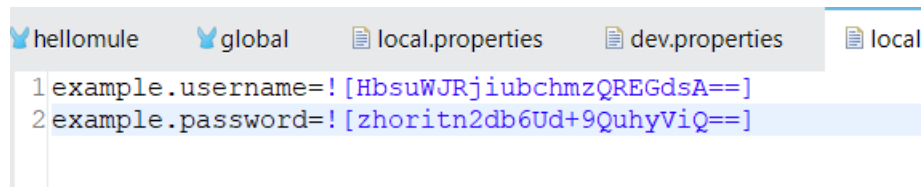


Figura 44. Asignación valores cifrados.

44. Se realiza una ejecución de la configuración dando clic derecho en el proyecto > Run As > Run Configurations...
45. Esto nos despliega un panel en el que nos colocaremos en la pestaña *Environment*, damos clic en *Add* y colocamos los valores *secure.key* y *MyMuleSoftKey*. (**Figura 45**)

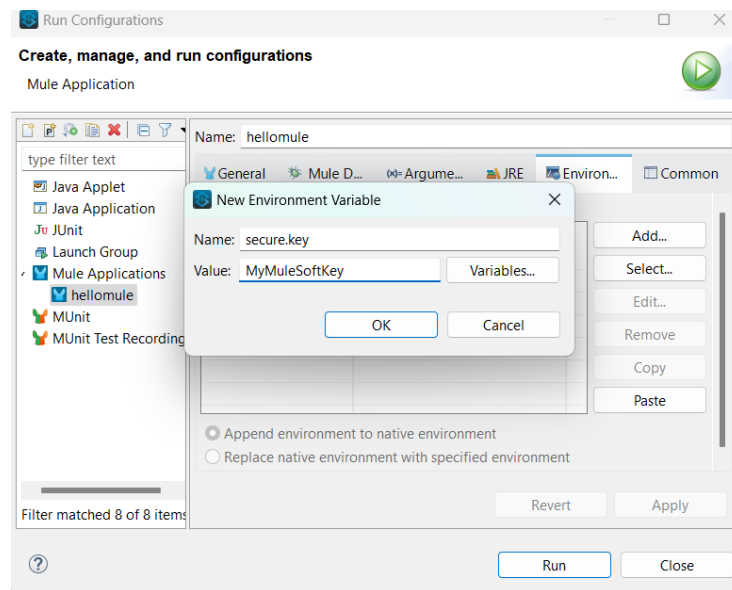


Figura 45. Nuevas variables de entorno.

46. Una vez que se realiza la ejecución de la aplicación, realizamos la ejecución en Postman de la petición (0.0.0.0:8081/hellomule)
Y en consola podemos ver el Log que se colocó, en el que se observa el usuario y contraseña (**Figura 46**)

```
extension.internal.runtime.source.ExtensionMessageSource: Message s
?rocessor: Username: myUsernameLocal - Password: myPasswordLocal
```

Figura 46. Usuario y contraseña logger.

La siguiente prueba es realizar el despliegue en CloudHub, en esta ocasión se hará uso de los secure key y los valore env que fueron configurados previamente.

47. Se coloca nuevamente en la aplicación que desplegamos en el **paso 17**, nos dirigimos al apartado properties y se añaden dos nuevas propiedades *env – dev* y *secure.key – MyMuleSoftKey* (**Figura 47**)

Properties	
env	dev
secure.key	MyMuleSoftKey
key	value

⚠ Deployments with less than 0.1 vCPU available may take up to 10 minutes to start

Cancel Deploy Application

Figura 47. Add properties CloudHub.

48. Lo siguiente es ocultar el valor de *secure.key* para mantenerlo seguro, por lo tanto, nos dirigimos a la carpeta target del proyecto y en el documento *mule-artifact.json* añadimos la línea "*secureProperties*": ["*secure.key*"] Esto ocultará nuestra contraseña (**Figura 48**).

Properties	
env	dev
secure.key	*****
key	value

Figura 48. Key oculta.

49. Se realiza nuevamente la petición en postman apuntando a la URL de CloudHub (**paso 20**) y en la página de CloudHub podemos observar los logs de nuestra aplicación y verificar el log que contiene el usuario y contraseña. (**Figura 49**)

```
*****
* Application: hellomule-daniel
* OS encoding: UTF-8, Mule encoding: UTF-8
*
*****

11:57:34.980 11/11/2023 Worker-0 [MuleRuntime].uber.01: [hellomule-
daniel].uber@org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource.lambda$reallyDoStart$17:462 @6d0873f4 INFO
Message source 'listener' on flow 'hellomuleFlow' successfully started

11:57:34.997 11/11/2023 Deployment system SYSTEM
Worker(3.144.2.233): Your application has started successfully.

11:58:38.704 11/11/2023 Worker-0 [MuleRuntime].uber.01: [hellomule-daniel].hellomuleFlow.CPU_LITE @56db510 INFO
event:f0d98150-80bb-11ee-8ad9-0242980f30ab Username: myUsernameLocal - Password: myPasswordLocal
```

Figura 49. Logs CloudHub usuario y contraseña

Create a new API in API Manager

50. Se ingresa a API Manager en la página de AnyPoint, seleccionamos Add API > Add new API. Seleccionamos Mule Gateway y dejamos la configuración predeterminada (**Figura 50**).

Select runtime

☐ Flex Gateway **NEW**
Ultrafast API gateway designed to manage and secure APIs running anywhere.

☒ Mule Gateway
API gateway embedded in Mule runtime. Connect directly to an existing Mule app or deploy a new proxy app.

☐ Service Mesh
Manage Kubernetes-based non-Mule microservices with Anypoint Service Mesh.

Proxy type *

☒ Connect to existing application (basic endpoint)
Connect your API to a Mule application using Autodiscovery.

☐ Deploy a proxy application
Select a deployment target and deploy a new Mule application to serve as a proxy.

Mule version *

☒ Mule 4 (recommended)

☐ Mule 3 or below

Cancel Next

Figura 50. Mule Gateway API.

51. Seleccionamos Create New API, ingresamos el nombre y el Asset Type (hellomule y HTTP API)

API

Select the API you want to manage.

☐ Select API from Exchange

☒ Create new API

i Once the API is created it will be published in Exchange in stable state.

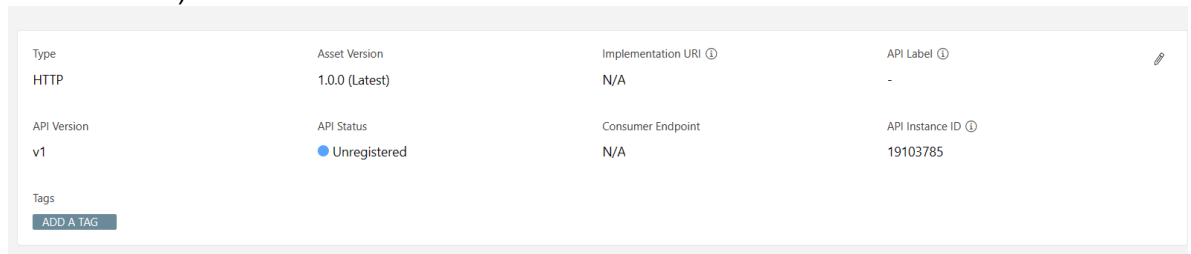
Name hellomule

Asset types ⁱ HTTP API

Advanced >

Figura 51. Nombre y tipo de API.

52. Nos despliega la información de la API y obtenemos el ID de la API. (API Instance ID 19103785)

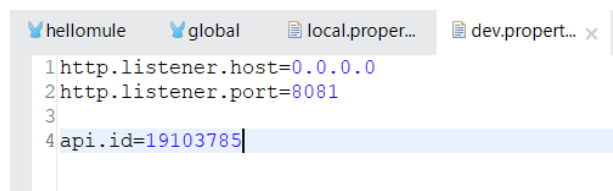


The screenshot shows a table with API details. The first row has columns: Type (HTTP), Asset Version (1.0.0 (Latest)), Implementation URI (N/A), and API Label (-). The second row has columns: API Version (v1), API Status (Unregistered), Consumer Endpoint (N/A), and API Instance ID (19103785). Below the table is a 'Tags' section with an 'ADD A TAG' button.

Type	Asset Version	Implementation URI ⓘ	API Label ⓘ
HTTP	1.0.0 (Latest)	N/A	-
API Version	API Status	Consumer Endpoint	API Instance ID ⓘ
v1	Unregistered	N/A	19103785
Tags			
ADD A TAG			

Figura 52. Información API.

53. Dentro de los archivos dev.properties y local.properties se agrega ip.id con el id de nuestra aplicación

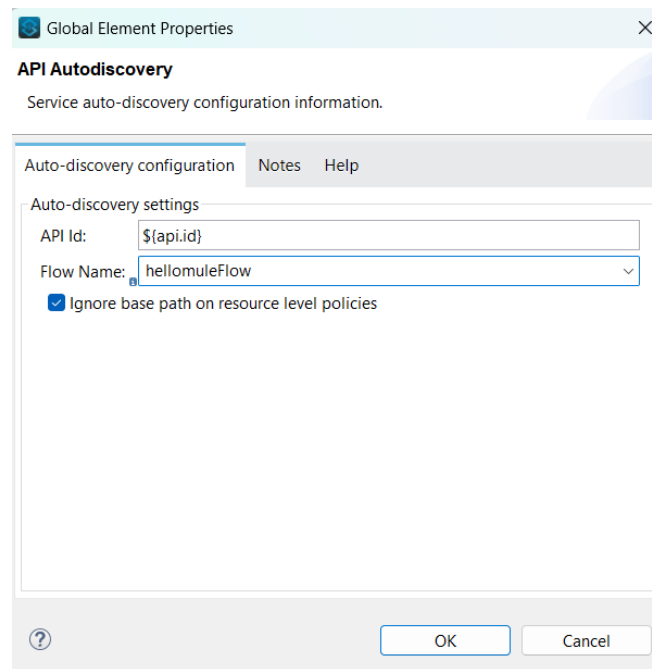


The screenshot shows an IDE with two tabs: 'local.properties' and 'dev.properties'. The 'dev.properties' tab is active, showing the following content:

```
1 http.listener.host=0.0.0.0
2 http.listener.port=8081
3
4 api.id=19103785
```

Figura 53. Dev properties API ID.

54. Se crea un nuevo modulo dentro de global.xml, en este caso es el API Autodiscovery, una vez instalado se cambia en las propiedades del elemento el API id por `${api.id}`, de esta manera obtendrá el valor de nuestras propiedades de desarrollo (dev) y se selecciona el flujo de hellomule (**Figura 54**).



The screenshot shows the 'Global Element Properties' dialog box for 'API Autodiscovery'. The 'Auto-discovery configuration' tab is selected. The 'API Id' field contains the value `${api.id}`. The 'Flow Name' dropdown is set to 'hellomuleFlow'. The checkbox 'Ignore base path on resource level policies' is checked. The dialog has 'OK' and 'Cancel' buttons at the bottom.

Global Element Properties

API Autodiscovery

Service auto-discovery configuration information.

Auto-discovery configuration | Notes | Help

Auto-discovery settings

API Id: `${api.id}`

Flow Name: hellomuleFlow

☒ Ignore base path on resource level policies

OK Cancel

Figura 54. Propiedades API Autodiscovery.

API Autodiscovery servirá para detectar las API que se encuentran disponibles en todo momento en lugar de requerir la configuración manual.

55. Posteriormente dentro de la página de AnyPoint nos colocamos en el apartado AccesManagement > Business Groups > Seleccionamos el nombre de nuestro grupo (**Figura 55**)


Name	Environments	Total vCores
	2	2

Figura 55. Business groups.

56. Seleccionamos Environments y posteriormente Sandbox, esto nos despliega un panel que contiene las credenciales de Sandbox. Véase **Figura 56**.

Edit environment

Name

Sandbox

Client ID

697b74ae87aa4290adc0e5d57c2626bc

Client Secret

859f6fBd4298464499BA18539112A89E

Hide

Delete Environment

Cancel

Update

Figura 56. Credenciales Sandbox.

57. Estas credenciales las colocaremos en nuestro IDE. Para ello iremos a Window > AnyPoint Studio > API Manager y en el apartado Environment Credentials pegaremos los datos de Sandbox y daremos clic en *Validate* (**Figura 57**). Con esto configuramos el entorno SandBox con nuestro IDE.

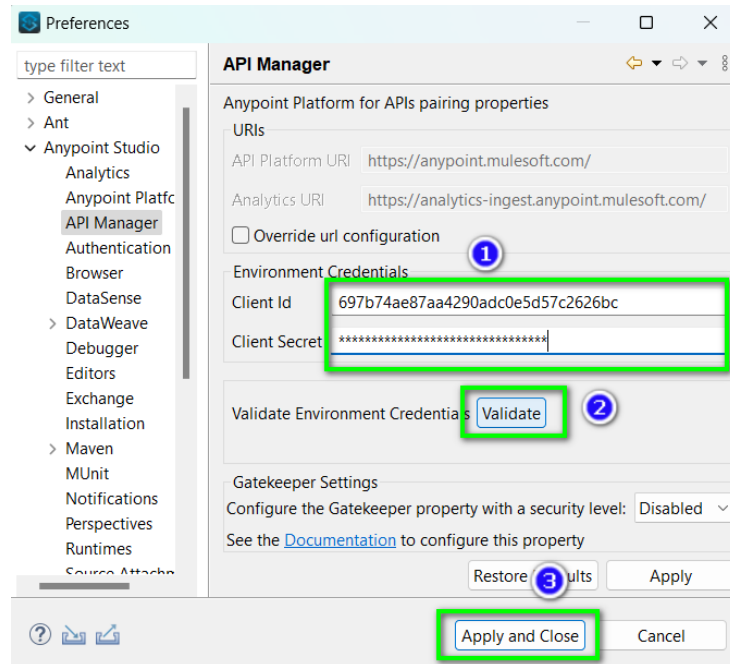


Figura 57. Preferencias AnyPoint Studio – Sandbox.

58. Al ejecutar la aplicación nos indica que la aplicación ya se ha levantado (**Figura 58**).

```
hellomule [Mule Applications] [pid: 24000]
*****
INFO 2023-11-11 12:50:10,906 [WrapperListener_start_runner] org.eclipse.jetty.server.AbstractConnector: Starte
INFO 2023-11-11 12:50:10,910 [WrapperListener_start_runner] org.mule.runtime.core.internal.logging.LogUtil:
*****
* - - + DOMAIN + - - * - - + STATUS + - - *
*****
* default * DEPLOYED *
*****

*****
* - - + APPLICATION + - - * - - + DOMAIN + - - * - - + STATUS + - - *
*****
* hellomule * default * DEPLOYED *
*****
```

Figura 58. Aplicación levantada.

59. Realizamos una petición a través de Postman a nuestro entorno local y observamos la respuesta obtenida. En este caso se tiene una respuesta correcta que nos confirma que está funcionando correctamente.

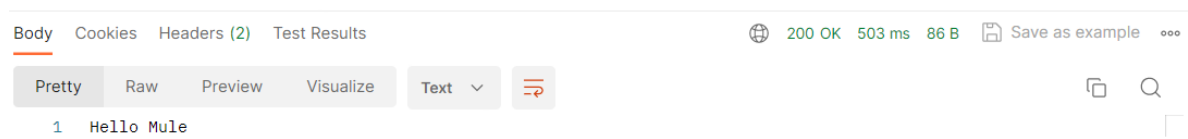


Figura 59. Petición Postman crea API.

60. El siguiente paso es recrear esto en CloudHub, para ello primero se agregaran campos a mule-artifact.json, en específico la información referente a cliente_id y la contraseña, estos datos son los obtenidos de Sandbox.

61. En CloudHub se ingresan las dos nuevas variables, mismas que se colocaron en el json en el punto anterior

anypoint.platform.client_id= 697b74ae87aa4290adc0e5d57c2626bc

anypoint.platform.client_secret= 859f6fBd4298464499BA18539112A89E

62. Una vez que se realizó la implementación en API Manager se puede ver que el estatus de la aplicación es que se encuentra disponible.
Por lo tanto, ahora se pueden aplicar nuevas implementaciones desde API Manager sin necesidad de volver a realizar la implementación de la aplicación Mule (**Figura 60**)

Type	Asset Version	Implementation URI ⓘ	API Label ⓘ	
HTTP	1.0.0 (Latest)	N/A	-	
API Version	API Status	Consumer Endpoint	API Instance ID ⓘ	
v1	● Active	N/A	19103785	
Mule Version				
4.5.1				
Tags				
ADD A TAG				

Figura 60. Aplicación Activa.

63. Si realizamos una petición en Postman y observamos los logs podemos observar que está funcionando correctamente (**Figura 61**).

Starting ResourceManager				
13:29:42.230	11/11/2023	Worker-0	qtp2071594616-39	INFO
Started ResourceManager				
13:29:42.234	11/11/2023	Worker-0	qtp2071594616-39	INFO
Starting Bean: proxy-policy-a95c0a20-80c8-11ee-964d-0a772f1ad9b7				
13:29:42.235	11/11/2023	Worker-0	qtp2071594616-39	INFO

* Policy: analytics-policy-19103785 @ hellomule-daniel-hellomuleFlow *				
* OS encoding: UTF-8, Mule encoding: UTF-8 *				
* *				

13:29:42.341	11/11/2023	Deployment	system	SYSTEM
Worker(3.14.81.226): Your application has started successfully.				
13:30:54.772	11/11/2023	Worker-0	[MuleRuntime].uber.03: [hellomule-daniel].hellomuleFlow.CPU_LITE @732ca66	INFO
event:d473afb0-80c8-11ee-964d-0a772f1ad9b7 Username: myUsernameLocal - Password: myPasswordLocal				

Figura 61. Petición realizada correctamente.

How to apply Client ID enforcement policy to your Mule app in API Manager

64. En la página de AnyPoint nos dirigimos a API Manager > Seleccionamos la API (hellomule) > Policies

65. Ahí seleccionamos Add Policy > Buscamos *Client ID Enforcement* y damos clic en Next (**Figura 62**)

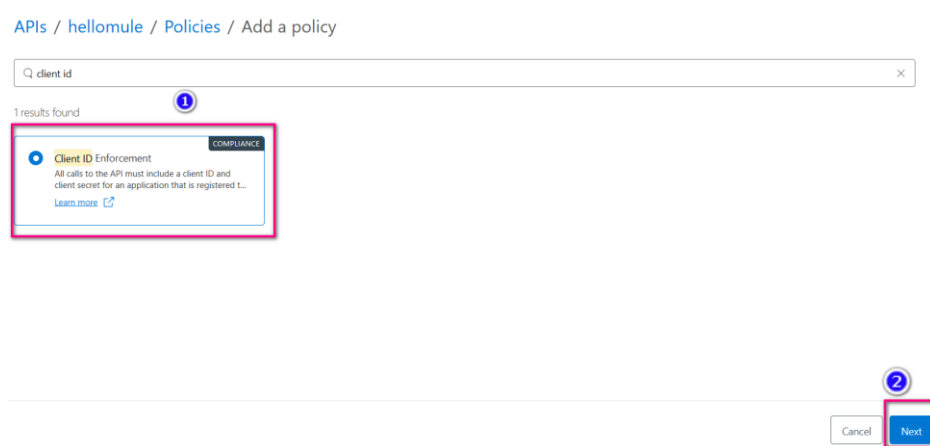


Figura 62. Agregar Client ID Enforcement.

66. Seleccionamos *HTTP Basic Authentication Header*, y la versión más actual disponible. Véase la **Figura 63**.

Credentials origin
Origin of the Client ID and Client Secret credentials.

☒ HTTP Basic Authentication Header
☐ Custom Expression

Advanced options ▾
Configure policy version, methods and resources

Policy version *
1.3.2 (latest) ▾ ⓘ

Method & resource conditions
☒ Apply configuration to all API method & resources ☐ Apply configuration to specific API method & resources

Figura 63. Configuración Policies.

67. Realizamos la petición a postman apuntando a la URL de CloudHub
Obtenemos un error 401 Unauthorized y nos indica que nuestra autenticación falló.

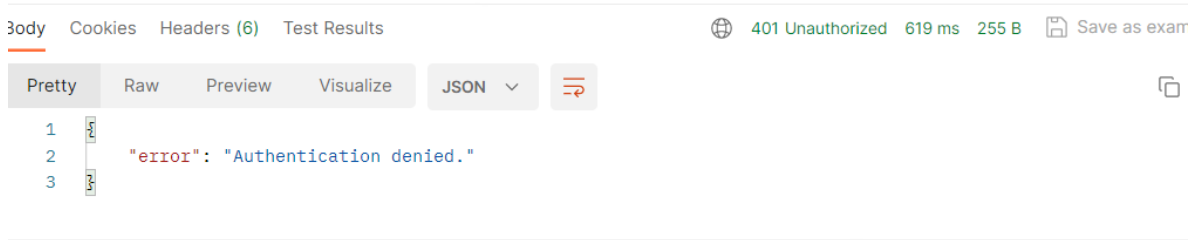


Figura 64. Petición errónea.

68. En AnyPoint Platform nos dirigimos a *Exchange* y nos muestra que hay una aplicación publicada (hellomule). Damos clic sobre la aplicación y nos despliega detalles sobre ella (**Figura 65**)

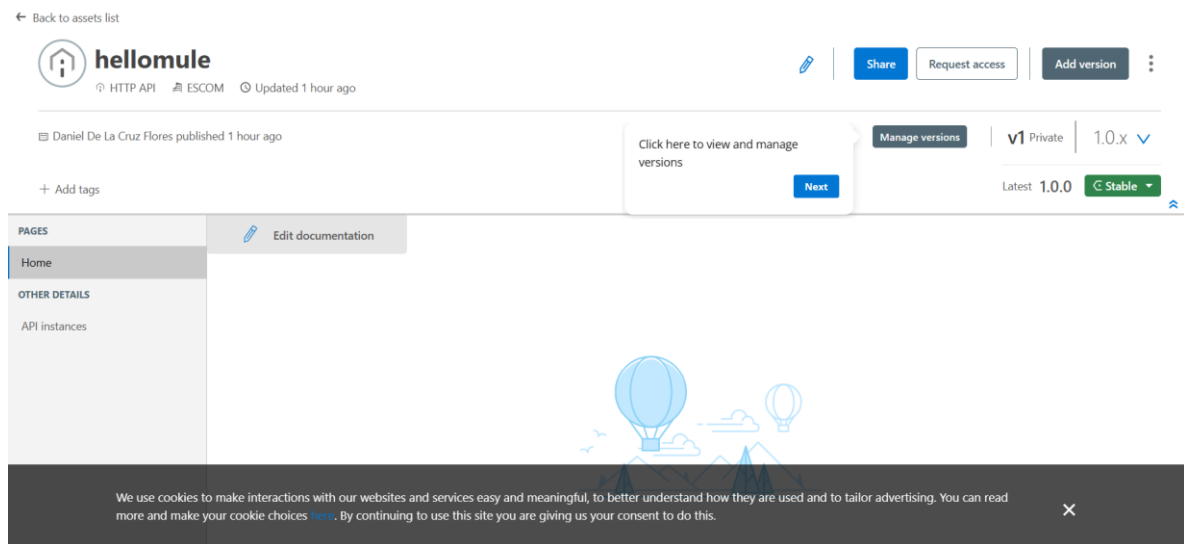
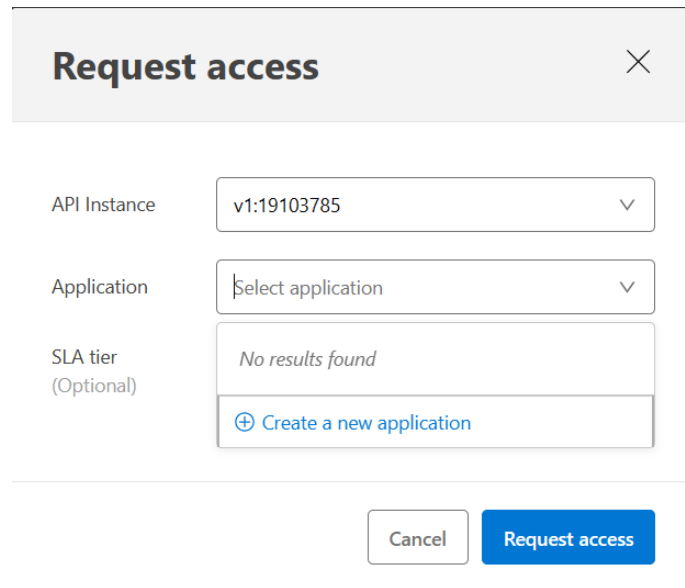


Figura 65. Detalles Aplicación hellomule.

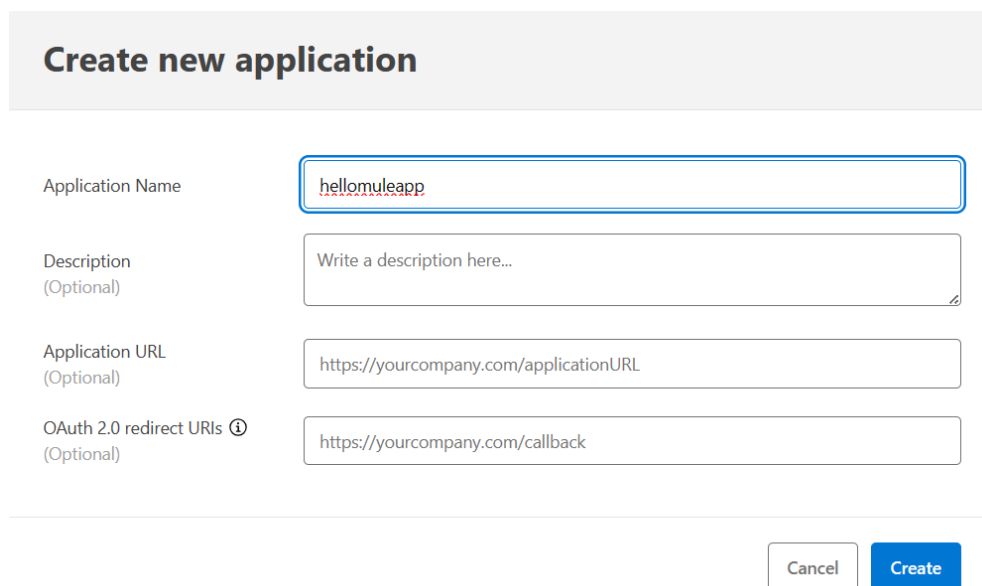
69. Damos clic en Request Access > seleccionamos la instancia de la API > en el campo aplicación seleccionamos *Create a new application*.



The 'Request access' dialog box features a title bar with a close button (X). It contains three dropdown menus: 'API Instance' with the value 'v1:19103785', 'Application' with the placeholder 'Select application', and 'SLA tier (Optional)' with the text 'No results found'. Below the 'SLA tier' dropdown is a link that says '+ Create a new application'. At the bottom right, there are two buttons: 'Cancel' and 'Request access'.

Figura 66. Request acces.

70. Se despliega un panel en el que ingresamos únicamente el nombre de la aplicación, en este caso hellomuleapp y damos clic en Create. Véase **Figura 67**.



The 'Create new application' form has a title bar. It includes four input fields: 'Application Name' containing 'hellomuleapp', 'Description (Optional)' with the placeholder 'Write a description here...', 'Application URL (Optional)' containing 'https://yourcompany.com/applicationURL', and 'OAuth 2.0 redirect URIs (Optional)' with an information icon and containing 'https://yourcompany.com/callback'. At the bottom right, there are 'Cancel' and 'Create' buttons.

Figura 67. Create new application.

71. Regresamos al panel de la **Figura 66**, damos clic en Request acces y con esto generamos el acceso para nuestra API.

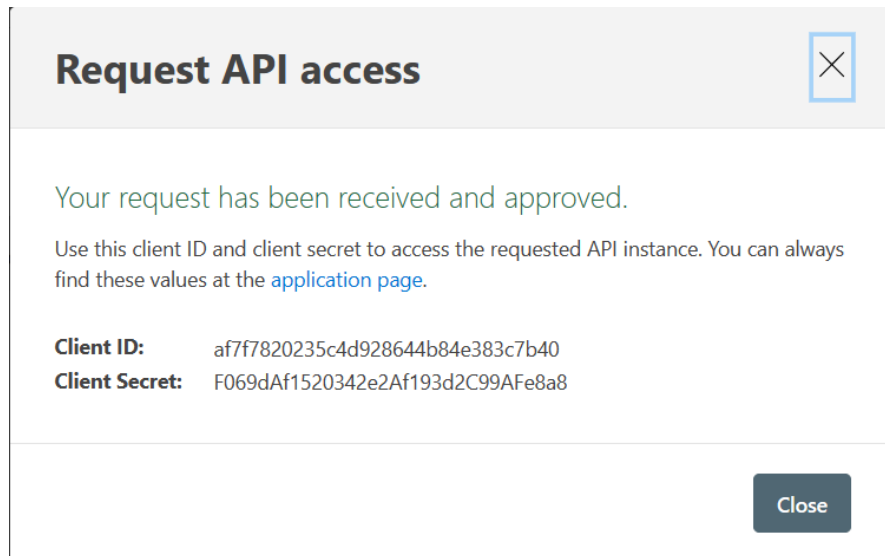


Figura 68. Datos acceso a la API.

72. El siguiente paso es configurar Postman con los valores de las credenciales (**Paso 71**), estos valores se insertan en el apartado *Authorization* de Postman, seleccionamos Basic Auth y ahí ingresamos los valores de Client ID y Client Secret, tal como se muestra en la **Figura 69**.

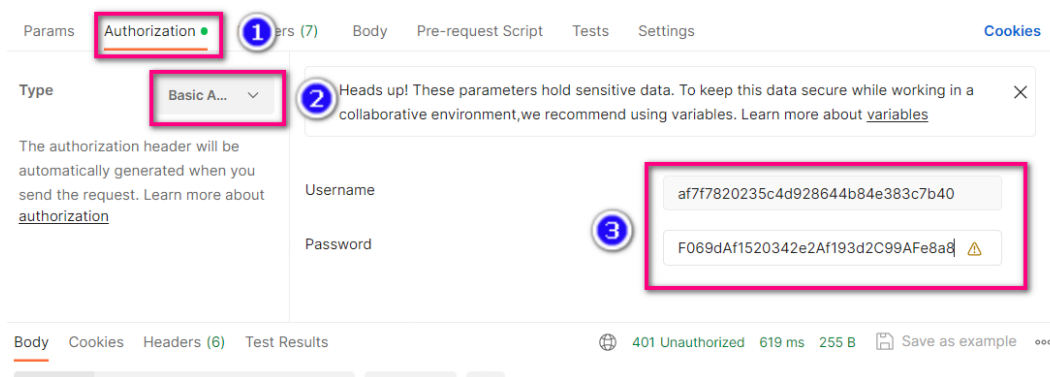


Figura 69. Configuración Authentication Postman.

73. Al volver a lanzar la aplicación podemos observar que el resultado es 200 Ok, y nos devuelve el mensaje esperado.

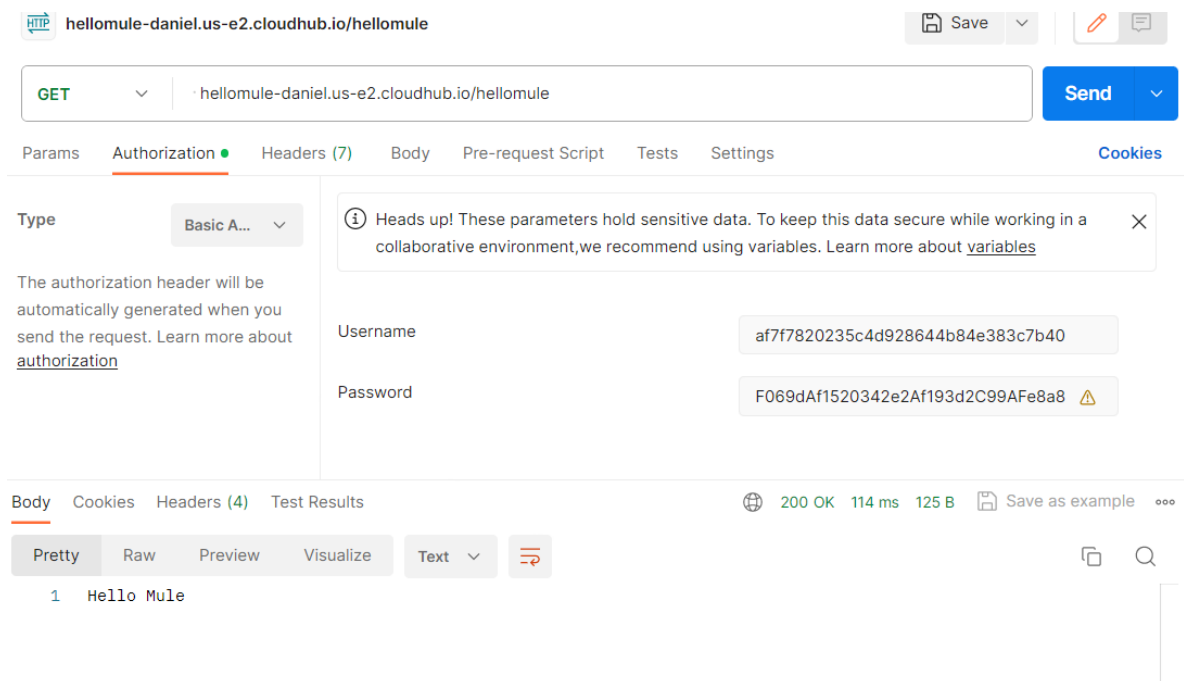


Figura 70. Petición Postman con Autenticación.

Build your first API Specification with API Designer

74. Para construir la especificación de la API el primer paso es posicionarse en *Design Center* en AnyPoint Platform > Create > New API Specification (**Figura 71**)

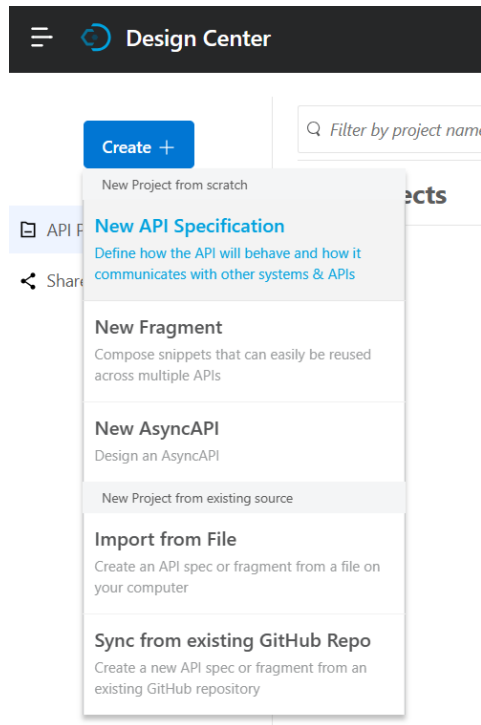


Figura 71. Crear especificación API.

75. En el panel que se despliega se ingresa el nombre el proyecto (En este caso para la práctica se utilizó NTO Customer Database API y se selecciona Guide me through it.

New API specification

Project name (required)

NTO Customer Database API

How do you want to draft the API Spec?

☐ **I'm comfortable designing it on my own**
A complete code editing experience with interactive documentation

Specification Language
RAML 1.0

☒ **Guide me through it**
Use a visual interface scaffolding the API Specification (can generate both RAML & OAS)

Use GitHub to store, manage, and collaborate on API specifications.
To get started authorize the MuleSoft application. [Learn more](#)

Authorize

Figura 72. Especificación Nueva API.

76. Se insertan los datos requeridos conforme se muestra en la **Figura 73**. Del lado izquierdo se pueden observar los datos que establecerán la configuración de la especificación, del lado derecho se tiene una previsualización del modelado de la API que se está realizando en RAML y OAS.

API Summary

Title New API **Version** 1.0.0

Protocols HTTP x HTTPS x **Media type** application/json x

Base URI api.samplebaseuri.com **Base URI Parameters (0)** >

Description

A simple API to retrieve information from Northern Trail Outfitters' customer database.

Secured By Select...

Documentation (0)

Edit RAML **Download**

```

#%RAML 1.0
title: New API
baseUri: api.samplebaseuri.com
description: A simple API to retrieve
information from Northern Trail Outfitters'
customer database.
mediaType:
- application/json
version: 1.0.0
protocols:
- HTTP
- HTTPS
  
```

RAML **OAS**

Figura 73. Configuración especificación-.

77. Posteriormente se selecciona el símbolo “+” de *Data Types*, en esta ventana se agregarán diferentes campos, el primero será agregar datos de tipo Address, tal como se muestra en la **Figura 74**.

Property Name	Required	Type	Union
street	<input type="checkbox"/>	String	<input type="checkbox"/>
city	<input type="checkbox"/>	String	<input type="checkbox"/>
postalCode	<input type="checkbox"/>	String	<input type="checkbox"/>
state	<input type="checkbox"/>	String	<input type="checkbox"/>
country	<input type="checkbox"/>	String	<input type="checkbox"/>

Figura 74. Campos Address

78. Se agrega otro tipo de datos, en este caso serán datos de tipo *Contacto*, en la **Figura 75** se muestran los datos ingresados.

Property Name	Required	Type	Union
firstName	<input type="checkbox"/>	String	<input type="checkbox"/>
lastName	<input checked="" type="checkbox"/>	String	<input type="checkbox"/>
phone	<input type="checkbox"/>	String	<input type="checkbox"/>
email	<input type="checkbox"/>	String	<input type="checkbox"/>
deliveryAddress	<input type="checkbox"/>	Address	<input type="checkbox"/>

Figura 75. Campos Contact.

79. Al final de la página se encuentra un campo llamado *Example*, en este campo se inserta un ejemplo de los campos que fueron agregados previamente. El código agregado es el siguiente:

```
{
  "firstName": "Example",
  "lastName": "Example",
  "phone": "Example",
  "email": "Example",
  "deliveryAddress": {
    "street": "Example",
    "city": "Example",
    "postalCode": "Example",
    "state": "Example",
    "country": "Example"
  },
  "postalAddress": {
    "street": "Example",
    "city": "Example",
    "postalCode": "Example",
    "state": "Example",
    "country": "Example"
  }
}
```

80. Posteriormente se agregarán dos recursos, para ello se dará clic en el botón *Add Resource*, se inserta información en los campos mostrados del panel, específicamente */customers* en el resource Path (será la ruta para realizar la llamada), en resource Method se establecerá el método GET y en Name será *Get list of costumers*. En la **Figura 76**, se muestra el llenado de los campos.

The screenshot shows the configuration for a new API resource. The 'Resource path' is set to `/customers`. The 'GET' method is selected from the available options (GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD). The 'Name' of the resource is 'Get list of costumers'. The 'Secured By' dropdown is set to 'Select...'. The interface includes tabs for 'Summary', 'Responses (0)', 'Query Parameters (0)', and 'Headers (0)'. A 'URI Parameters (0)' button is also visible.

Figura 76. Resources API.

81. Se realiza un response para la respuesta a la petición, se establece primero el estatus a configurar (200 Ok), el Media Type se coloca application/json, el tipo Array y en Items Type se establece Contact, que es uno de los Data Types que se crearon en los puntos anteriores.

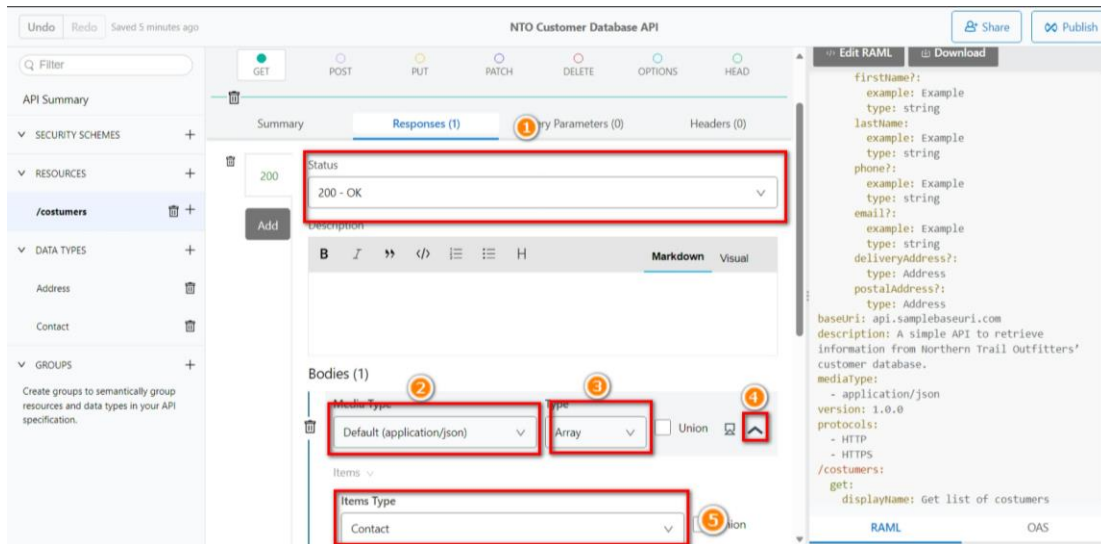


Figura 77. Configuración Response 200 Ok.

82. Se crea un nuevo Resource, el Resource Path es /customers/{customerId}, el Resource Method GET y Name Get costumer by ID

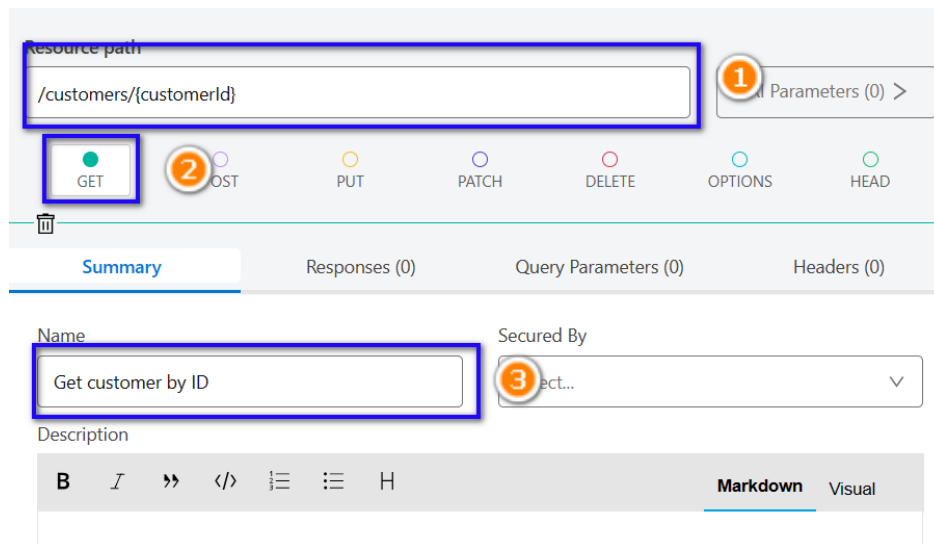


Figura 78. Resource Get costumer by ID

83. Se da clic en URI Parameters y Add URI Parameter, se agrega costumerId en URI Parameter Name, se habilita el campo Required y el tipo String. Véase **Figura 79**.

The screenshot displays the 'URI Parameters' configuration section of an API tool. At the top, the 'Resource path' is set to '/customers/{customerId}'. Below this, the 'URI Parameters (1)' section shows a single parameter named 'customerId'. The 'Required' checkbox is checked, and the 'Type' is set to 'String'. A 'Union' checkbox is also present but unchecked. At the bottom, there is an 'Add URI Parameter' button and a row of HTTP method icons: GET, POST, PUT, PATCH, DELETE, OPTIONS, and HEAD.

Figura 79. Configuración parámetros.

84. Se realiza la configuración de Response de la misma manera que se realizó en el punto anterior.

The screenshot shows the 'Response' configuration section. The 'Status' is set to '200 - OK'. The 'Description' field is empty, with a rich text editor toolbar visible above it. Below the description, the 'Bodies (1)' section shows a single body configuration. The 'Media Type' is set to 'Default (application/json)' and the 'Type' is set to 'Array'. A 'Union' checkbox is also present but unchecked. At the bottom, the 'Items Type' is set to 'Contact'.

Figura 80. Response petición con parámetros.

85. Una vez establecidos todos los parámetros se realiza una prueba, para ello se da clic en el botón del lado derecho de la página *Try it*, esto despliega un panel del que se seleccionará la opción *Mocking Service* que realiza una simulación como si la aplicación estuviera activa (**Figura 81**)

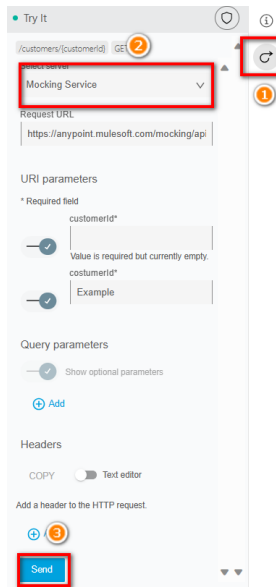


Figura 81. Test de peticiones

86. Se realizan ambas peticiones. Y se obtienen los resultados de cada una de las peticiones. Véase **Figura 82** y **Figura 83**.



Figura 82. Response costumers



Figura 83. Response by Id.

87. Se habilita el link publico para que puedan consultar el servicio de manera pública.
Esto se hace habilitando *Make Public* y agregándole al final del link el servicio

Link:anypoint.mulesoft.com/mocking/api/v1/links/b5b95b3d-27f0-4ca6-90c7-39c4dd4698ce/customers/1

Link:<https://anypoint.mulesoft.com/mocking/api/v1/links/b5b95b3d-27f0-4ca6-90c7-39c4dd4698ce/costumers>

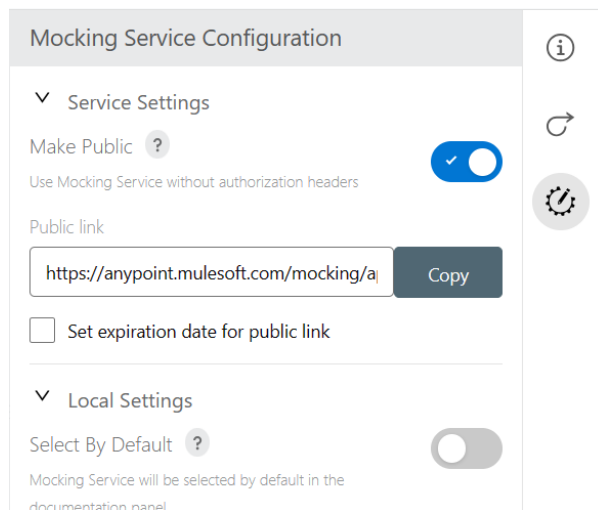
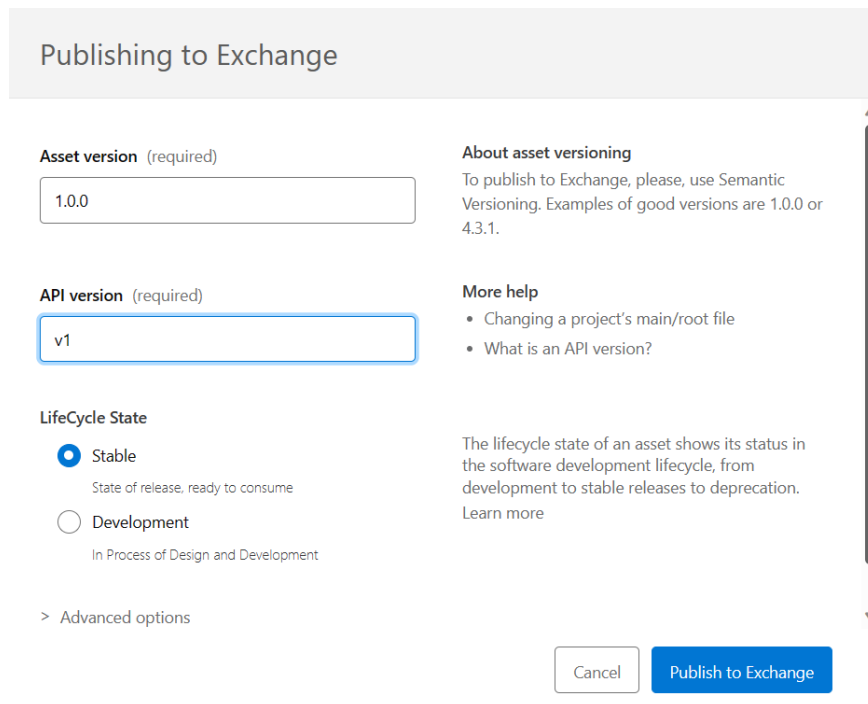


Figura 84. Link público.

88. Das clic en Publish y seleccionas la configuración predeterminada



The image shows a 'Publishing to Exchange' dialog box. It has a title bar at the top. Below the title bar, there are two input fields: 'Asset version (required)' with the value '1.0.0' and 'API version (required)' with the value 'v1'. To the right of these fields, there is a section titled 'About asset versioning' with text explaining Semantic Versioning and examples of good versions. Below the input fields, there is a 'LifeCycle State' section with two radio buttons: 'Stable' (selected) and 'Development'. To the right of this section, there is a 'More help' section with two bullet points. At the bottom right, there are two buttons: 'Cancel' and 'Publish to Exchange'.

Publishing to Exchange

Asset version (required)
1.0.0

API version (required)
v1

LifeCycle State

☒ **Stable**
State of release, ready to consume

☐ **Development**
In Process of Design and Development

> Advanced options

About asset versioning
To publish to Exchange, please, use Semantic Versioning. Examples of good versions are 1.0.0 or 4.3.1.

More help

- Changing a project's main/root file
- What is an API version?

The lifecycle state of an asset shows its status in the software development lifecycle, from development to stable releases to deprecation. [Learn more](#)

Cancel Publish to Exchange

Figura 85. Publishing to Exchange.

89. Si nos dirigimos a Exchange podemos ver la aplicación publicada.

Desde aquí los usuarios que tengan acceso a estas aplicaciones pueden exportarlo o consultar detalles de las aplicaciones para continuar con el desarrollo.

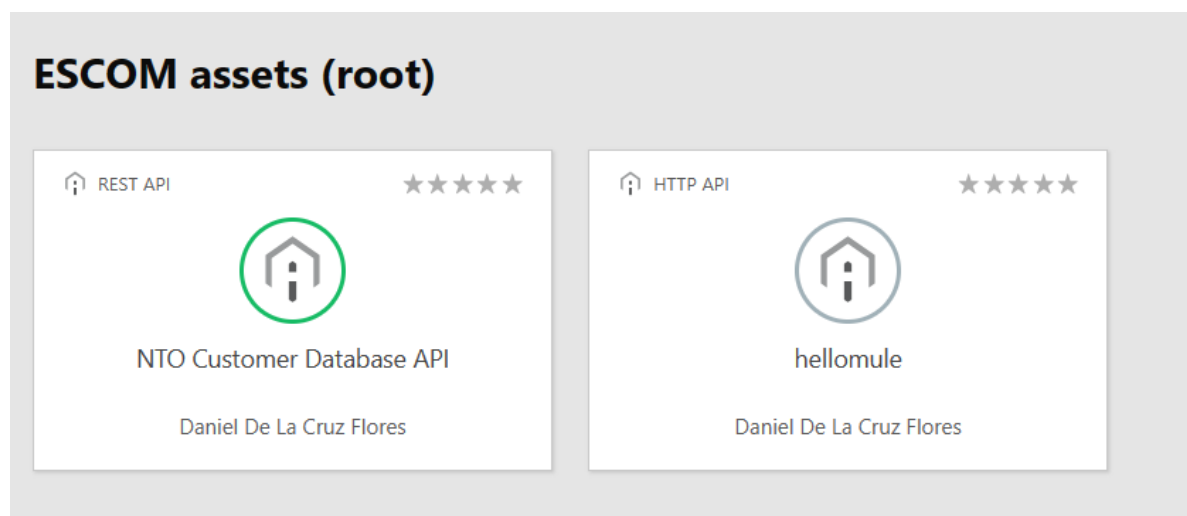


Figura 86. Aplicaciones Exchange.

How to configure an HTTPS endpoint in Anypoint Studio

90. Se generará un archive keystore.jks para poder habilitar HTTPS en el host local, para ello se ejecuta una línea de comando en la terminal

```
keytool -genkey -alias mule -keystore mule-2.jks -keyalg RSA -storetype JKS
```

91. Posteriormente se ingresa a AnyPoint Studio, nos dirigimos al conector HTTP Listener y abrimos la configuración, aquí se modificará el protocolo de HTTP a HTTPS y el puerto a 8082

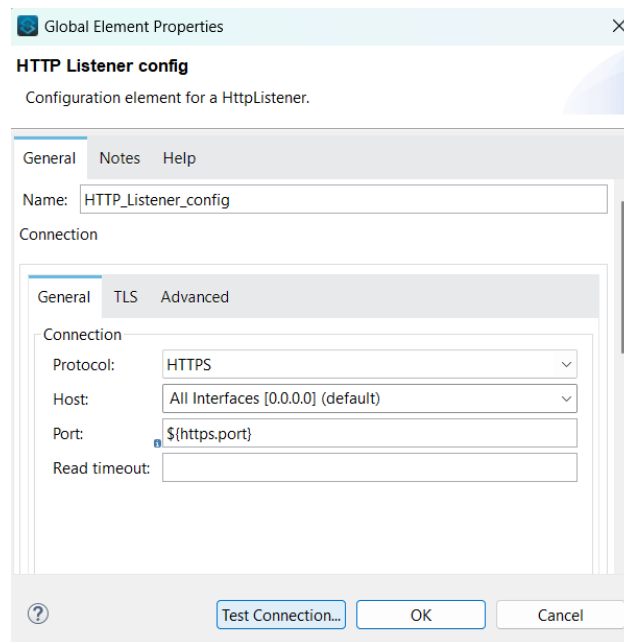


Figura 87. Configuración HTTPS.

92. Después nos dirigimos a TLS para realizar la configuración con las claves generadas.

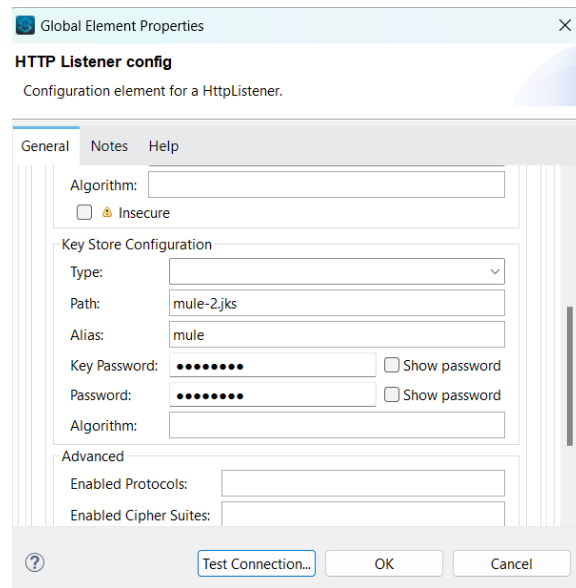


Figura 88. Configuración TLS.

93. Realizamos un test de la conexión para verificar que todo esté bien.

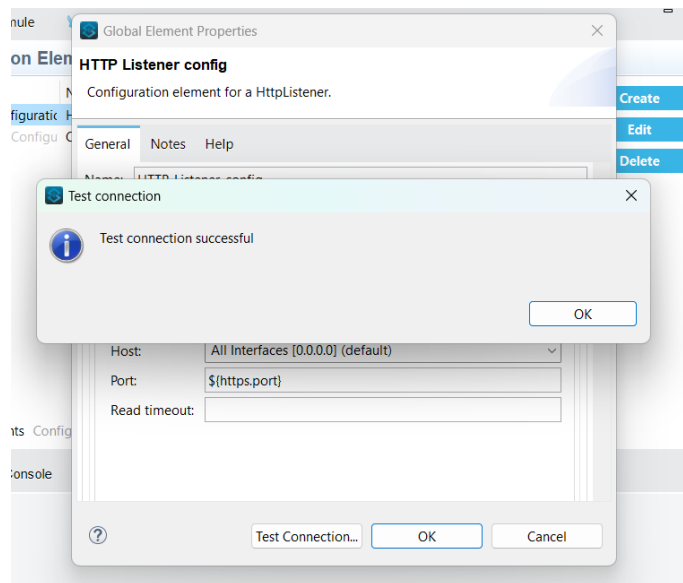


Figura 89. Test de conexión.

94. En el caso de Maven yo ya contaba con Maven instalado, ejecuté el comando mvn -v para verificar que todo estuviera correctamente. Véase **Figura 90**.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\dany2> mvn -v
Apache Maven 3.9.1 (2e178502fcd0b0c0b4b4cc58f8)
Maven home: C:\Program Files\Apache Software Foundation\apache-maven-3.9.1
Java version: 1.8.0_202, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_202\jre
Default locale: es_MX, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
PS C:\Users\dany2>
```

Figura 90. Maven instalado.

95. Se configura settings.xml en AnyPoint Studio

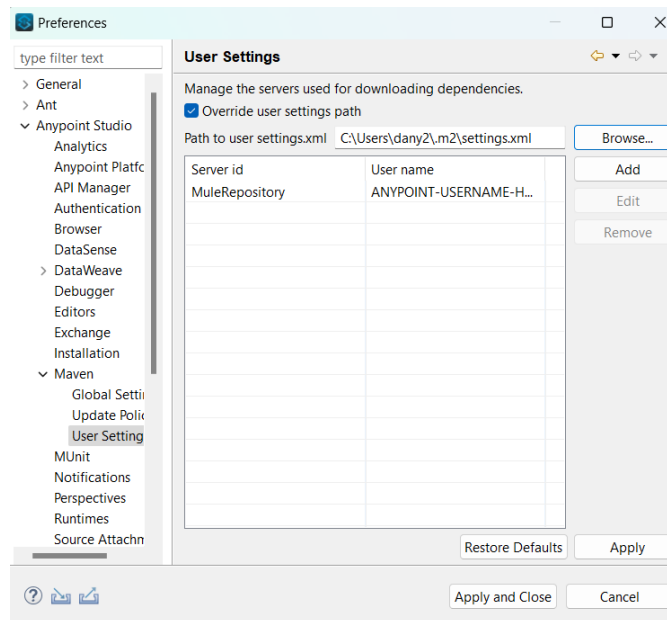


Figura 91. Configuración settings IDE.