

# **Day 1**

- [Inference Hands-on](#)
- [Vectors Hands-on](#)
- [Webcrawler Hands-on](#)

# Inference Hands-on

## Inference Connector Hands-on

As the Inference Connector is being certified as we deliver this training, it is not available on the Anypoint Exchange Portal.

Will will be using the latest open source release 0.5.7

### OpenAI API KEY

As part of the exercise, the MAC team will sponsor an **API Key from OpenAI** to be used during the exercise.

`sk-proj-IhCPniMttidFmGGZHrptPqQBxD80gmdF0ocVMESXrsUCgmSyGI8o1ifMg9jlaUY9fUDOXd2NpT3B1bkFJmavxgM7aSJj7820zF-J-9R54DwgkWgCHQPa6N7j2nTrr7kb6YtFXZwLW05-om4nfCGFMEeEYEA`

**This API Key will expire on the 30th of June.**

Please consider provisioning an Azure Open AI API Key through

[\*\*ACB \(Inference\)\*\*](#)

# **ACB (Inference)**

# **Inference Connector Hands-on**

## **Exercise 1 - Simple Chat App**

The purpose of this exercise is to demonstrate, how you can use the chat completions operation of LLM to implement a workflow in MuleSoft. In this exercise we are going to build a simple headless chat app with an LLM.

## Step 1

Create a Mule Project [hands-on-mule-inference](#)

### Develop an Integration

Create and test a Mule application project that integrates existing services.

**Project Name**

**Project Location**

 Browse

**Create**

**Empty Project**

*Create an integration project from scratch.*

**Template or Example Project**

*Start an integration with a template or example project from Anypoint Exchange.*

**Mule Runtime** ⓘ

 ▼

**Java Version** ⓘ

 ▼

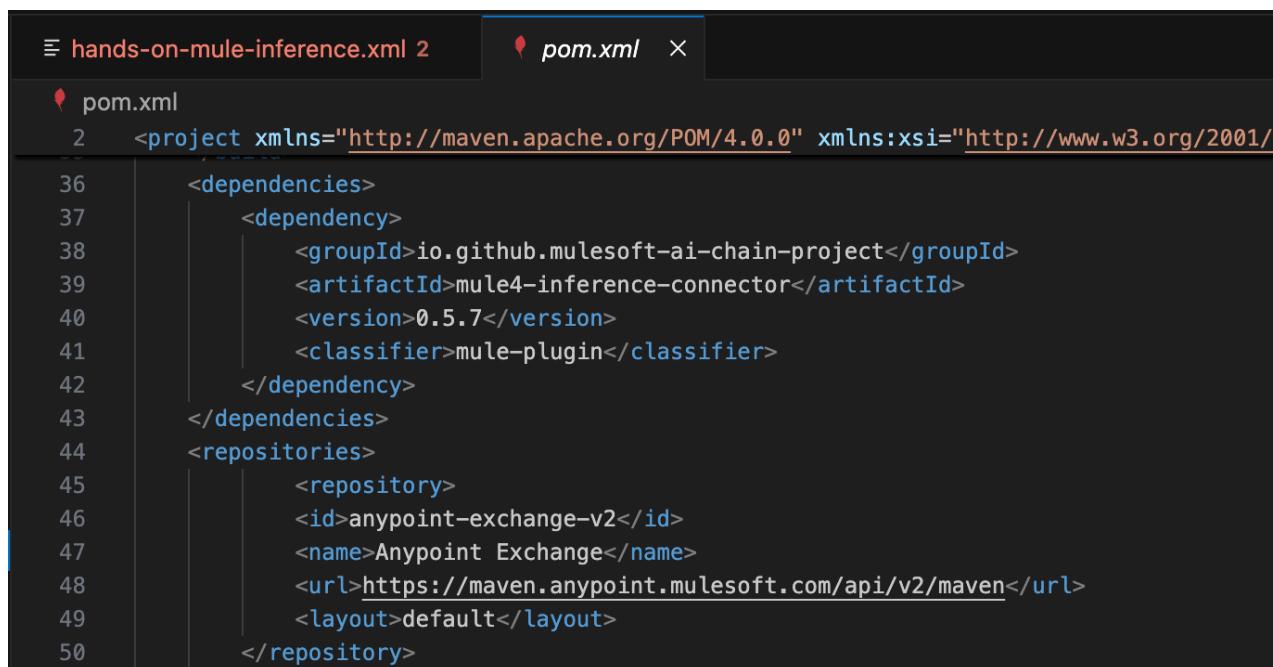
Cancel Create Project

## Step 2

Add the following dependency in the pom.xml  
hands-on-mule-inference

```
<dependency>
<groupId>io.github.mulesoft-ai-chain-project</groupId>
<artifactId>mule4-inference-connector</artifactId>
<version>0.5.7</version>
<classifier>mule-plugin</classifier>
</dependency>
```

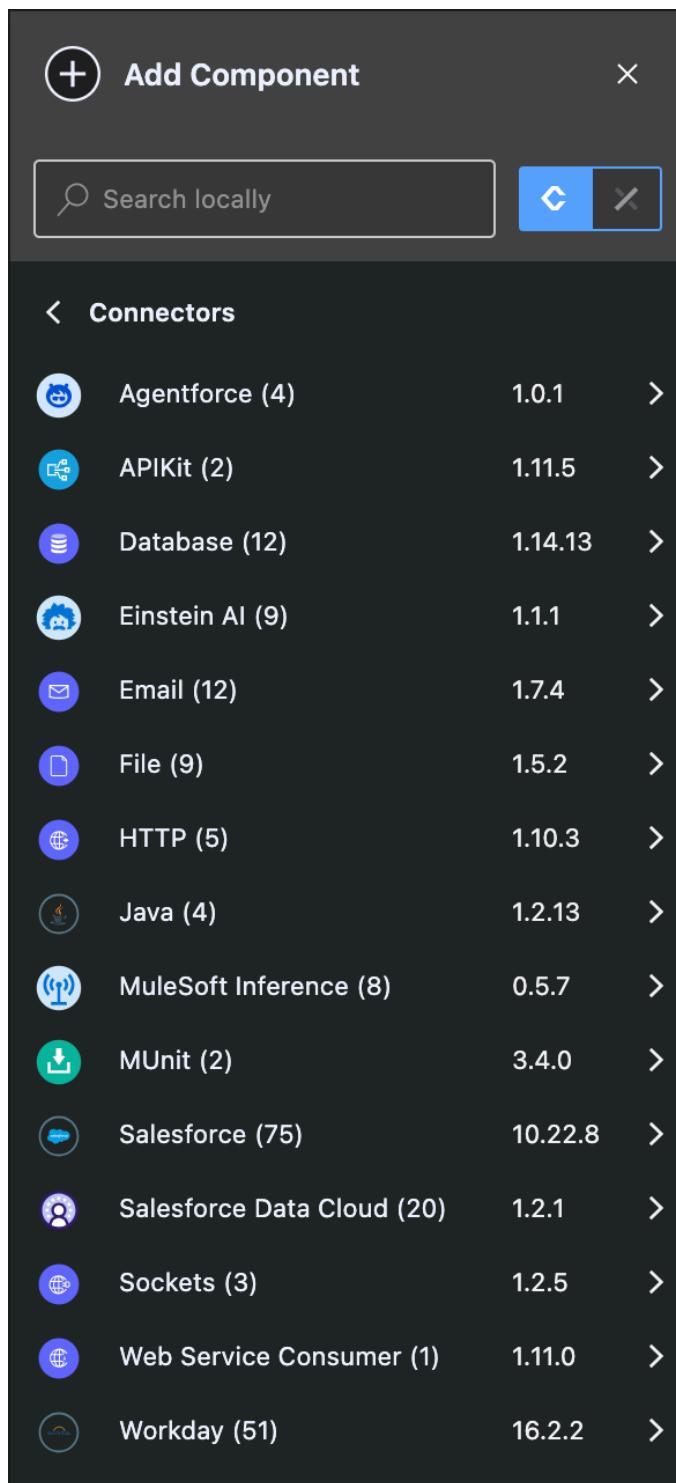
Save the project.



```
≡ hands-on-mule-inference.xml 2   pom.xml ×
  pom.xml
  2   <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
  36      <dependencies>
  37          <dependency>
  38              <groupId>io.github.mulesoft-ai-chain-project</groupId>
  39              <artifactId>mule4-inference-connector</artifactId>
  40              <version>0.5.7</version>
  41              <classifier>mule-plugin</classifier>
  42          </dependency>
  43      </dependencies>
  44      <repositories>
  45          <repository>
  46              <id>anypoint-exchange-v2</id>
  47              <name>Anypoint Exchange</name>
  48              <url>https://maven.anypoint.mulesoft.com/api/v2/maven</url>
  49              <layout>default</layout>
  50      </repository>
```

## Step 3

Check that MuleSoft Inference Connector is available in the Mule Palette



## Step 4

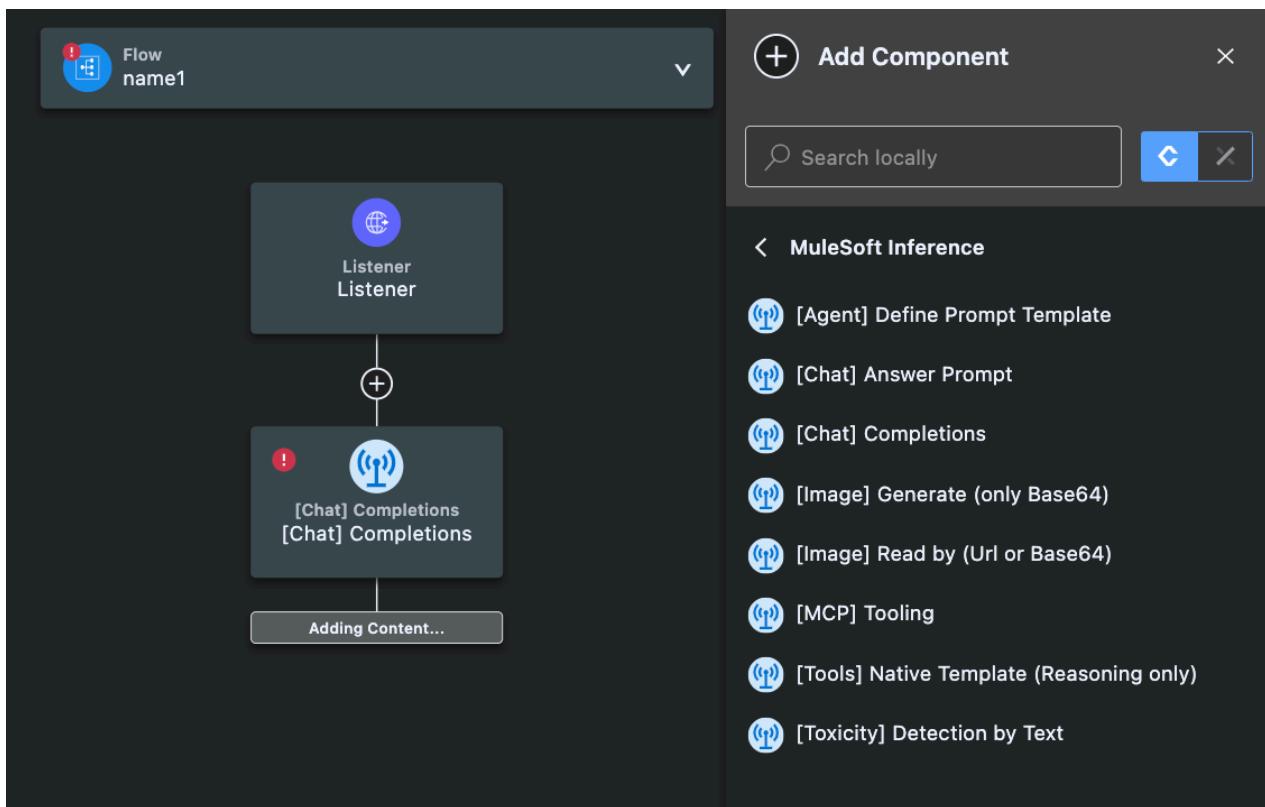
Drop a http listener (/inference/chat) to build a simple chat interaction using postman. Note: For ACB, please create a flow before dropping a processor.

The screenshot shows the Mule Studio interface with two main windows open:

- Configuration Window (Top):** This window is titled "Listener-config" and is used to configure an "Http - Listener". It includes sections for "Connection" (Protocol: HTTP (Default), Host: 0.0.0.0, Port: 8081, Read timeout: 30000), "General" (Base path: /inference), and "Listener interceptors" (Configure Listener interceptors, Reject invalid transfer encoding). A blue "Add" button is visible at the bottom right.
- Flow Editor (Bottom):** This window shows a flow named "name1". It contains a single "Listener" component (represented by a blue circle icon) which is connected to a dashed "Placeholder" box. To the right of the flow, the "Http - Listener" configuration is displayed again, showing the "Path" field set to "/chat".

## Step 5

Drop a [Chat] Completions operation from Inference into the canvas.



## Step 6

Add a new configuration (Text-Generation) for Inference Connector for OpenAI.

Inference Type: OPENAI

API Key: [[copy from here](#)]

Model Name: 'gpt-4o-mini'

leave the rest as is.

Click on the Test Connection and we should get 'Test connection successful' or 'Connection is valid' on ACB.

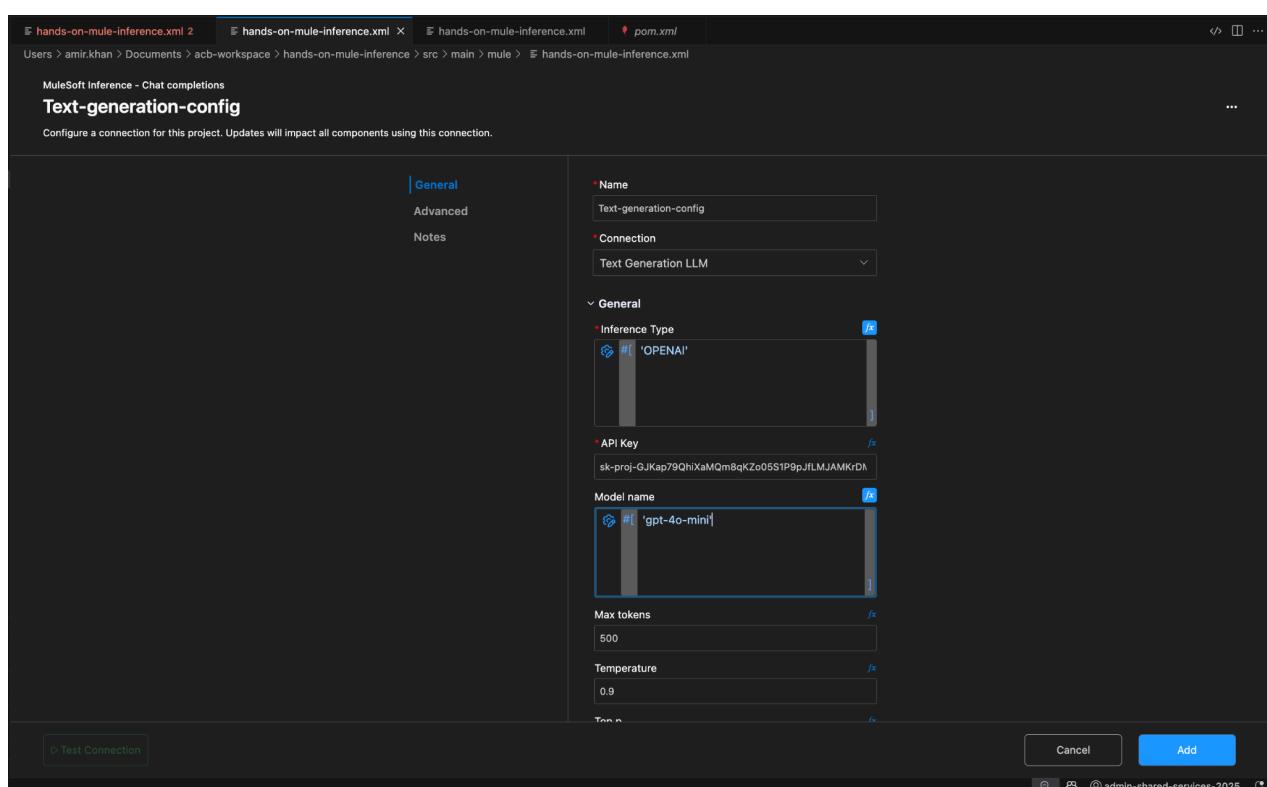
*Note: If in ACB the drop down list doesn't populate, please use the expression mode to fill-in the data. For Anypoint Studio, after specifying the Inference Type, sometimes the list is not displayed (Model name parameter). And you are getting the below while trying to refresh the list:*  
"

*Metadata resolution is taking longer than expected.*

*Click Studio services indicator to resolve.*

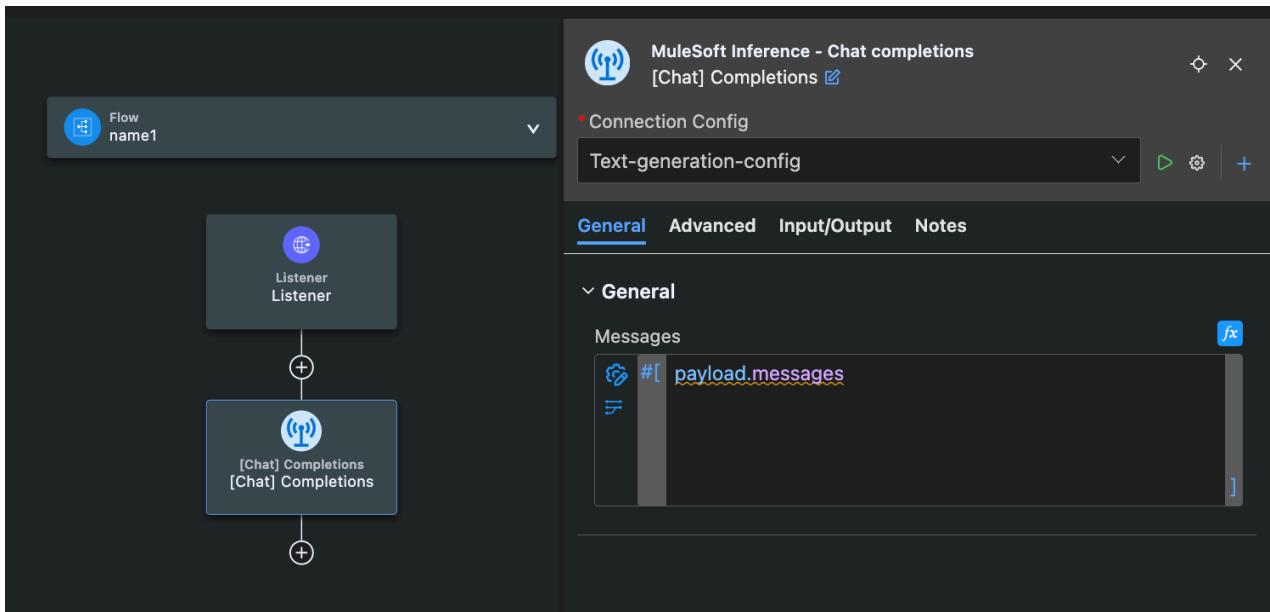
"

*The quickest fix is to restart Studio to restart the metadata resolution service.*



## Step 7

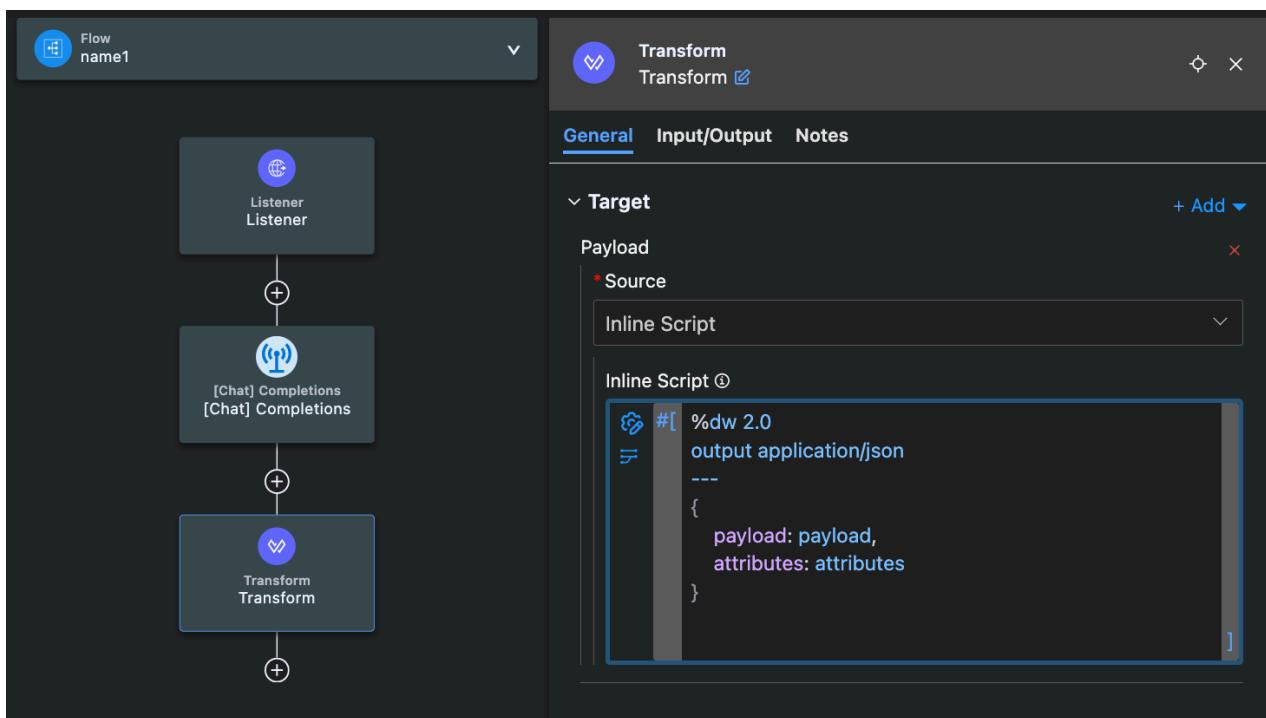
Select the [Chat] Completions operations and change the messages field to `payload.messages`



## Step 8

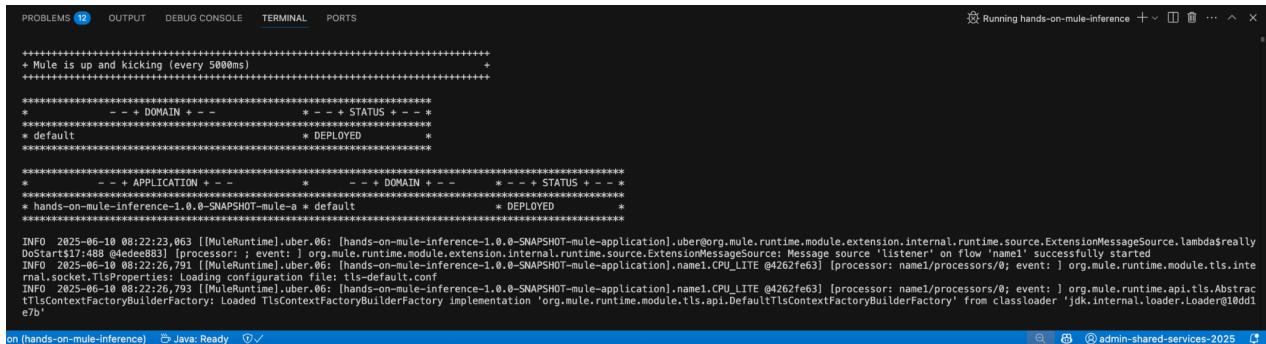
Add a Transform Message processor after [Chat] Completions with the following dataweave expression.

```
%dw 2.0
output application/json
---
{
  payload: payload,
  attributes: attributes
}
```



## Step 9

Deploy the application locally using the IDE (Studio or ACB)



The screenshot shows a terminal window titled "Running hands-on-mule-inference". The window displays deployment logs for a Mule application named "hands-on-mule-inference-1.0.0-SNAPSHOT-mule-a". The logs indicate that the application has been successfully deployed to a default domain. The output includes several INFO messages from the Mule runtime, such as the start of the application and the configuration of a TLS socket.

```
+ Mule is up and kicking (every 5000ms)
+-----+
*   - + DOMAIN + - - - + STATUS + - - *
*****+-----+-----+-----+-----+-----+
* default                                * DEPLOYED *
*****+-----+-----+-----+-----+-----+
*   - + APPLICATION + - - - + DOMAIN + - - - + STATUS + - - *
*****+-----+-----+-----+-----+-----+-----+
* hands-on-mule-inference-1.0.0-SNAPSHOT-mule-a * default          * DEPLOYED *
*****+-----+-----+-----+-----+-----+-----+
INFO 2025-06-10 08:22:23.963 [[MuleRuntime].uber.0@] [hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application].uber@org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource.lambda$reallyDoStart$17:488 @4edeab83] [processor: ; event: ] org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource: Message source 'listener' on flow 'name1' successfully started
INFO 2025-06-10 08:22:26.791 [[MuleRuntime].uber.0@] [hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application].name1.CPU_LITE @4262fe63] [processor: name1/processors/0; event: ] org.mule.runtime.module.tls.intern.socket.TlsProperties: Loading configuration file: tls-default.conf
INFO 2025-06-10 08:22:26.793 [[MuleRuntime].uber.0@] [hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application].name1.CPU_LITE @4262fe63] [processor: name1/processors/0; event: ] org.mule.runtime.api.tls.AbstractTlsContextBuilderFactory: Loaded TlsContextBuilderFactory implementation 'org.mule.runtime.module.tls.api.DefaultTlsContextBuilderFactory' from classloader 'jdk.internal.loader.Loder@10dd1e7b'
```

## Step 10

Use postman or any other rest api client to fire the following request:

```
curl --location 'localhost:8081/inference/chat' \
--header 'Content-Type: application/json' \
--data '{
  "messages": [
    {
      "role": "user",
      "content": "What is the capital of Switzerland!"
    }
  ]
}'
```

The screenshot shows the Postman interface with the following details:

- Request URL:** `localhost:8081/inference/chat`
- Method:** POST
- Body:** JSON (selected)  
Content:  
```

```
1 {
  "messages": [
    {
      "role": "user",
      "content": "What is the capital of Switzerland!"
    }
  ]
}
```

```
- Response Status:** 200 OK
- Response Body:** JSON (selected)  
Content:  
```

```
1 {
  "payload": {
    "response": "The capital of Switzerland is Bern."
  },
  "attributes": {
    "tokenUsage": {
      "outputCount": 7,
      "totalCount": 21,
      "inputCount": 14
    },
    "additionalAttributes": {
      "finish_reason": "stop",
      "model": "gpt-4o-mini-2024-07-18",
      "id": "chatcmpl-BgmfMwuaB0a2n7edxmUXRwYcPDRgc"
    }
  }
}
```

```

## Step 11

Use postman or any other rest api client to fire the following request:

```
curl --location 'localhost:8081/inference/chat' \
--header 'Content-Type: application/json' \
--data '{
  "messages": [
    {
      "role": "user",
      "content": "My Name Is Amir"
    }
  ]
}'
```

The screenshot shows the Postman interface with the following details:

- Request URL:** `localhost:8081/inference/chat`
- Method:** POST
- Headers:** Headers (10) - `Content-Type: application/json`
- Body:** Raw JSON payload:

```
1 {
  "messages": [
    {
      "role": "user",
      "content": "my Name is Amir"
    }
  ]
}
```
- Response Status:** 200 OK
- Response Body:** JSON output showing the response payload and token usage.

## Step 12

Use postman or any other rest api client to fire the following request:

```
curl --location 'localhost:8081/inference/chat' \
--header 'Content-Type: application/json' \
--data '{
  "messages": [
    {
      "role": "user",
      "content": "What is my name"
    }
  ]
}'
```

The screenshot shows the Postman interface with the following details:

- Request URL:** `localhost:8081/inference/chat`
- Method:** POST
- Headers:** Headers (10) - `Content-Type: application/json`
- Body:** Raw JSON payload:

```
1 {
  "messages": [
    {
      "role": "user",
      "content": "What is my name"
    }
  ]
}
```
- Response Status:** 200 OK
- Response Time:** 2.53 s
- Response Size:** 520 B
- Response Content:** (JSON)

```
1 {
  "payload": {
    "response": "I'm sorry, but I can't determine your name based on the information provided."
  },
  "attributes": {
    "tokenUsage": {
      "outputCount": 15,
      "totalCount": 26,
      "inputCount": 11
    },
    "additionalAttributes": {
      "finish_reason": "stop",
      "model": "gpt-4-2024-08-06",
      "id": "chatcmpl-BgubUBNwiwrf386x9fVbb9DDCnGOF"
    }
  }
}
```

## **Exercise 2 - Enabling Chat History**

As we are using the Chat Completions operation of an LLM, we can also provide history about the chat with the user. You can decide which data to be considered in the chat history when extending the chat app.

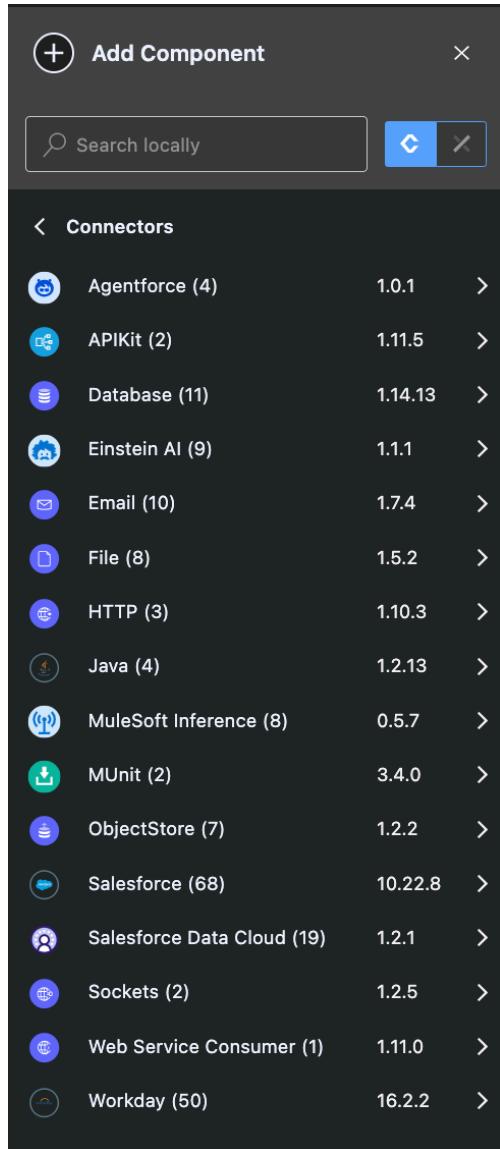
## Step 1

Open the [hands-on-mule-inference](#) app from exercise 1.

Drop the Object Store Connector into the Mule Palette or make sure it is available.

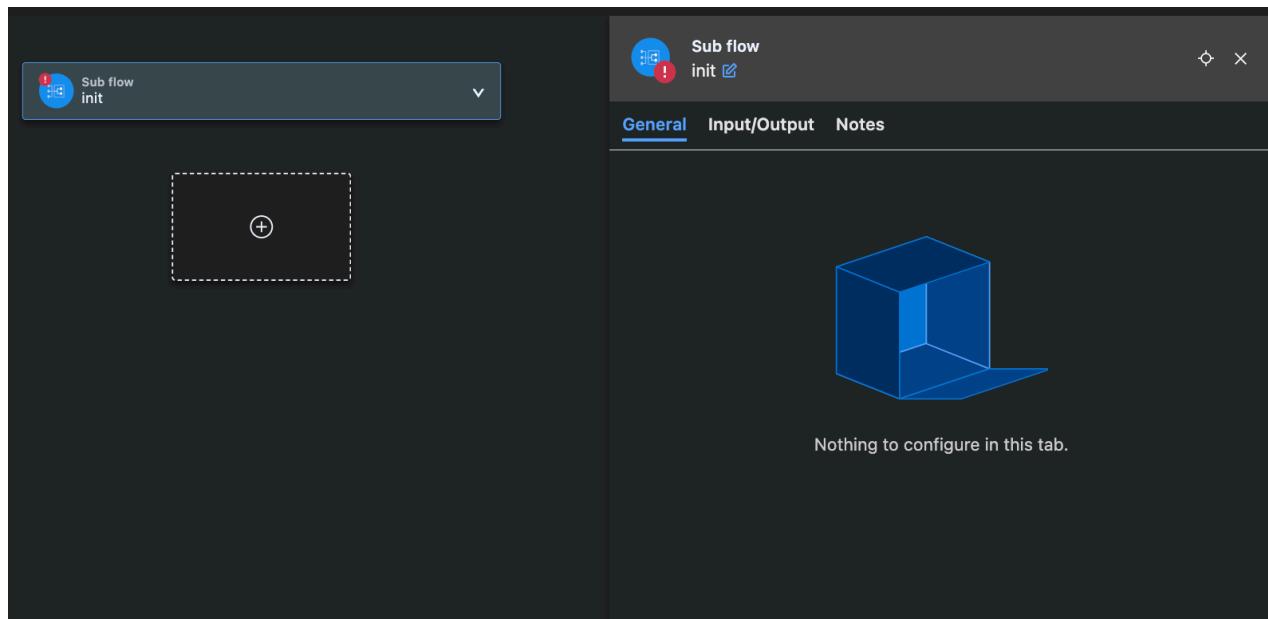
Once available, create a object store config either using UI or using the following expression in the xml.

```
<os:object-store name="Object_store" doc:name="Object store"  
doc:id="75bf104d-4263-4eb8-a2ac-7be3e619c4b8" persistent="false"/>
```



## Step 2

Create a new subflow called memory-retriever



## Step 3

Now we need to create a simple memory retriever for the chat history. The purpose of the simple memory retriever will be to evaluate the chat history by userId, and maxMessages, which are passed as Http Attributes to the app.

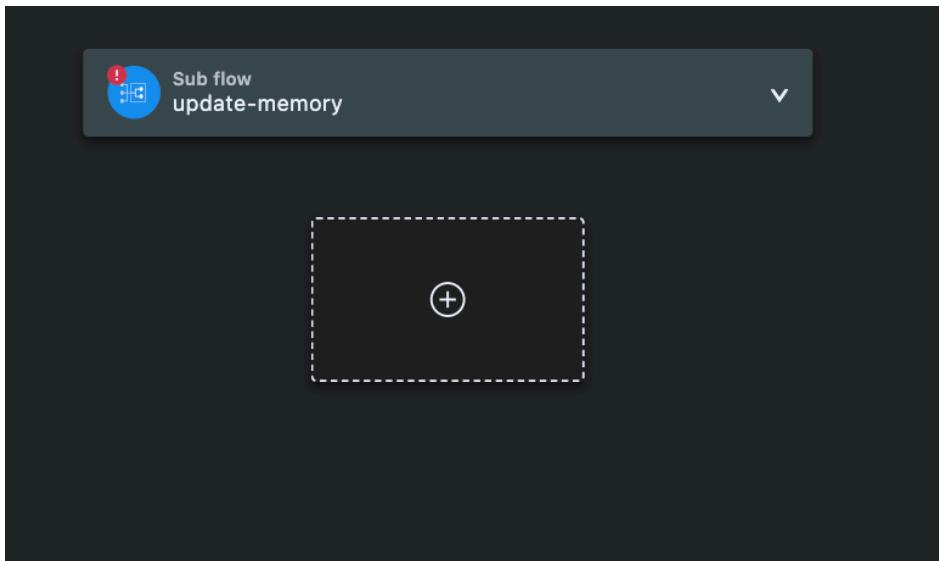
In order to achieve this, add the following variables and retriever (Object store) into the flow:

```
<set-variable value="#[payload.messages]" doc:name="User Prompt"
doc:id="76e70ed2-1b17-46fa-8e0a-327ae728158f"
variableName="userPrompt"/>
<set-variable value='#[attributes.headers."userId"]' doc:name="User Id"
doc:id="28824951-7f99-4313-b74f-50fd66fb94b" variableName="userId"/>
<os:retrieve doc:name="Load Memory"
doc:id="4e5a4fb2-ca70-44dc-84f7-f5de36676a44" key="#[vars.userId]"
objectStore="Object_store" target="memory">
<os:default-value ><![CDATA[#[output application/json
--->
{
chat_history: []
}]]></os:default-value>
</os:retrieve>
```



## Step 4

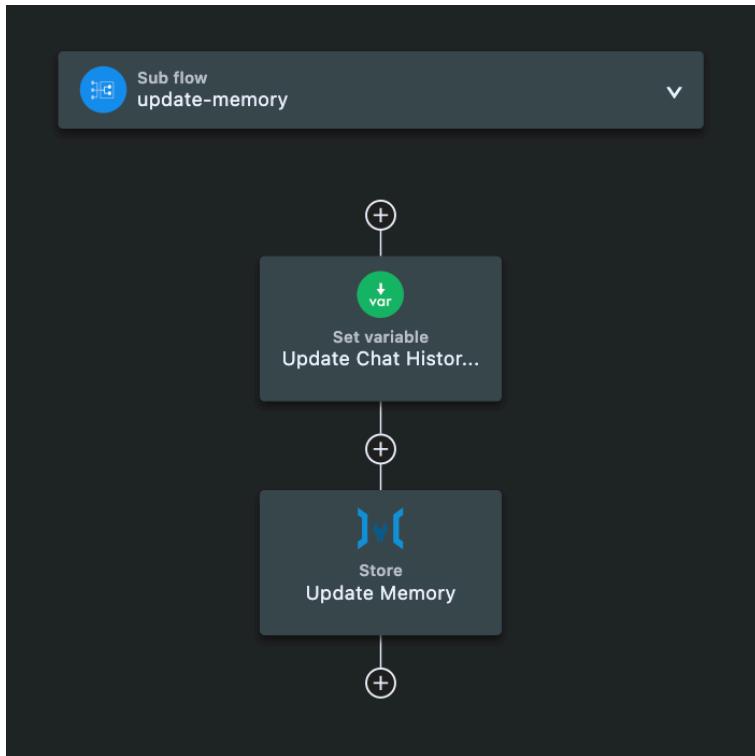
We need to create a new sub flow called "update-memory".



## Step 5

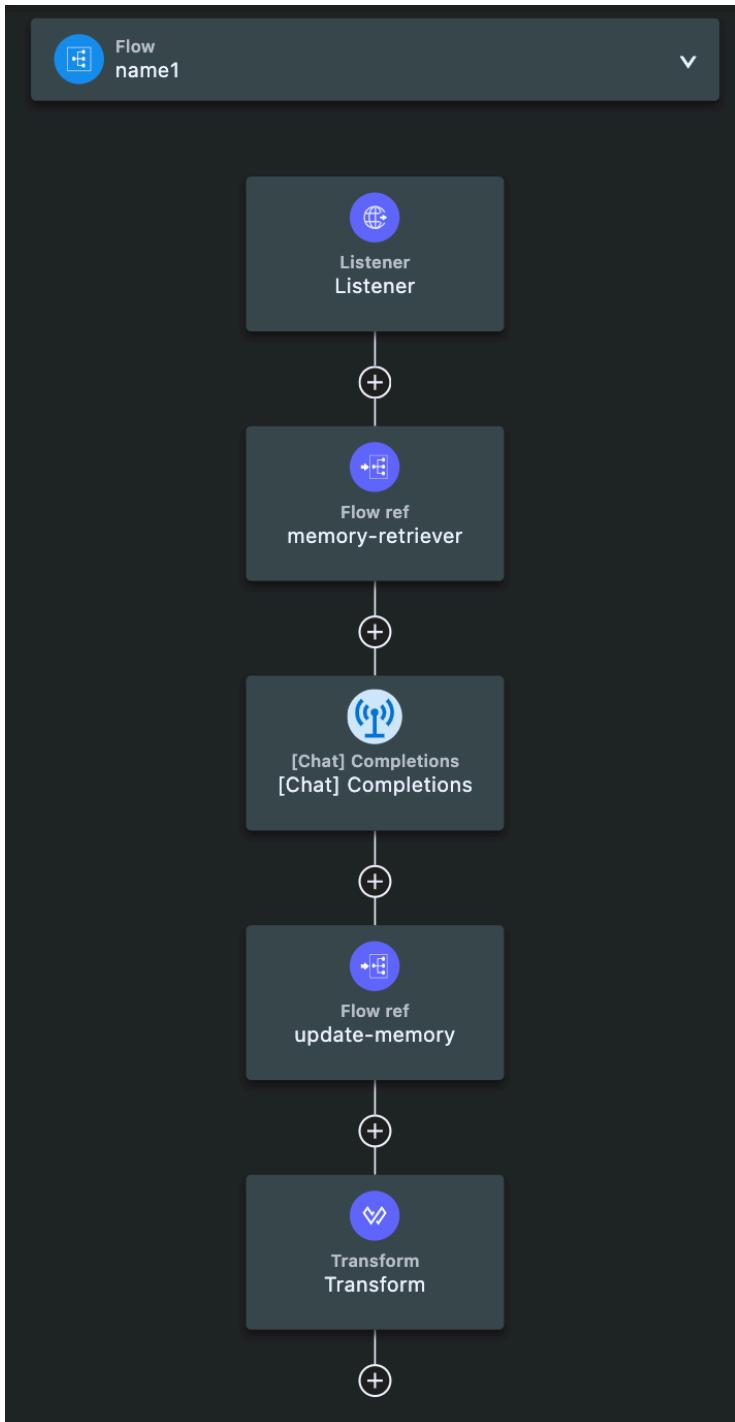
Now we need to update the memory retriever after the communication with the agent. This is done by copy / pasting the following dataweave code into the sub-flow "update-memory".

```
<set-variable value='#[output application/json---;vars.memory  
update {&#10; case history at .chat_history -> (history default []) ++  
[&#10; {&#10; role: "user",&#10; content: vars.userPrompt.content[0]&#10;  
},&#10; {&#10; role: "assistant",&#10; content: payload.response default  
""&#10; }]}' doc:name="Update Chat History"  
doc:id="e7e9a10b-363a-4464-85f3-496f5116e52b" variableName="memory" />  
<os:store doc:name="Update Memory"  
doc:id="bf81989c-f94d-4e3c-84b4-fc86d1c0d6ad" key='#[vars.userId]'  
objectStore="Object_store" >  
<os:value ><! [CDATA[# [vars.memory]]]></os:value>  
</os:store>
```



## Step 6

Add the flow references before (for memory-retriever) and after (for update-memory) the [Chat] Completions operation.



## Step 7

Update the [Chat] Completions operation with the following expression.

```
output application/json
---
vars.memory.chat_history ++ vars.userPrompt
```

**IMAGE MISSING**

## Step 8

Start the mule app locally.

```
*****+-----+
* Starting app
* 'hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application'
* Application plugins:
*   - MuleSoft Inference : 0.5.7
*   - ObjectStore : 1.2.2
*   - Sockets : 1.2.5
*   - HTTP : 1.10.3
*****+-----+
+Mule is up and kicking (every 500ms) +
*****+-----+
*****+-----+ + DOMAIN + - - + STATUS + - - *
*****+-----+ + DOMAIN + - - + STATUS + - - *
* default * DEPLOYED *
*****+-----+
*****+-----+ + APPLICATION + - - + + DOMAIN + - - + - + STATUS + - - *
*****+-----+ + APPLICATION + - - + + DOMAIN + - - + - + STATUS + - - *
* hands-on-mule-inference-1.0.0-SNAPSHOT-mule-a * default * DEPLOYED *
*****+-----+
INFO 2025-06-10 16:57:41.550 [[MuleRuntime].uber.06: [hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application].uber.org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource.lambda$really
DoStart$1@7:488 @6b749c47] [processor: ; event: ] org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource: Message source 'listener' on flow 'name1' successfully started
```

## Step 9

Use postman or any other REST API Client to fire the following request.

```
curl --location 'localhost:8081/inference/chat' \
--header 'userId: amir-khan' \
--header 'Content-Type: application/json' \
--data '{
  "messages": [
    {
      "role": "user",
      "content": "Whats my name"
    }
}'
```

The screenshot shows the Postman interface with the following details:

- Request Method:** POST
- URL:** localhost:8081/inference/chat
- Body:** JSON (selected)
- JSON Payload:**

```
1  {
2    "messages": [
3      {
4        "role": "user",
5        "content": "Whats my name"
6      }
7    ]
8 }
```
- Response Status:** 200 OK
- Response Body:**

```
1  {
2    "payload": {
3      "response": "I'm sorry, but I don't have access to personal data about
4          individuals unless it has been shared with me in the course of our
5          conversation. If you'd like to share your name, feel free to do so!"
6    },
7    "attributes": {
8      "tokenUsage": {
9        "outputCount": 41,
10       "totalCount": 51,
11       "inputCount": 10
12     },
13     "additionalAttributes": {
14       "finish reason": "stop",
15     }
16   }
17 }
```

## Step 10

Use postman or any other REST API Client to fire the following request.

```
curl --location 'localhost:8081/inference/chat' \
--header 'userId: amir-khan' \
--header 'Content-Type: application/json' \
--data '{
  "messages": [
    {
      "role": "user",
      "content": "My name is Amir"
    }
  ]
}'
```

The screenshot shows the Postman interface with the following details:

- Request URL:** `localhost:8081/inference/chat`
- Method:** POST
- Headers:** `Content-Type: application/json`
- Body (JSON):**

```
1 {
  "messages": [
    {
      "role": "user",
      "content": "My name is Amir"
    }
  ]
}
```
- Response Status:** 200 OK
- Response Headers:** 2.15 s, 499 B
- Response Body (JSON):**

```
1 {
  "payload": {
    "response": "Nice to meet you, Amir! How can I assist you today?"
  },
  "attributes": {
    "tokenUsage": {
      "outputCount": 14,
      "totalCount": 53,
      "inputCount": 39
    }
  },
  "additionalAttributes": {
    "finish_reason": "stop",
    "model": "gpt-4o-mini-2024-07-18",
    "id": "chatcmpl-BgV3g1r07ReK9SiYncrBIgF7KKMz"
  }
}
```

## Step 11

Use postman or any other REST API Client to fire the following request.

```
curl --location 'localhost:8081/inference/chat' \
--header 'userId: amir-khan' \
--header 'Content-Type: application/json' \
--data '{
  "messages": [
    {
      "role": "user",
      "content": "Whats my name"
    }
  ]
}'
```

The screenshot shows the Postman interface with a successful API call. The URL is `localhost:8081/inference/chat`. The method is `POST`. The request body is a JSON object:

```
1 {
  2   "messages": [
  3     {
  4       "role": "user",
  5       "content": "Whats my name"
  6     }
  7   ]
  8 }
```

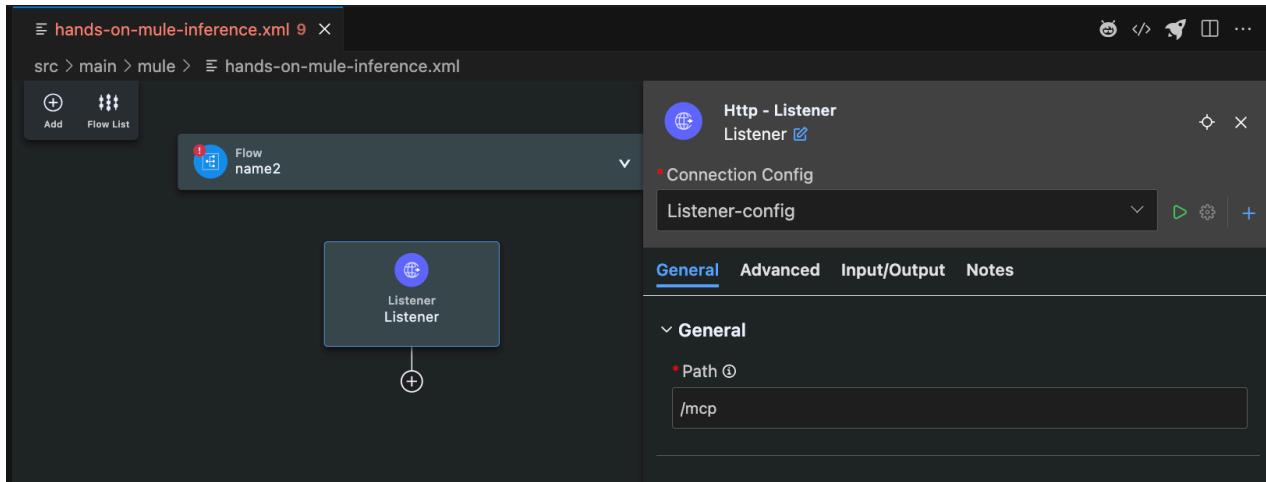
The response status is `200 OK` with a response time of `1.37 s` and a size of `498 B`. The response body is:

```
1 {
  2   "payload": {
  3     "response": "Your name is Amir. How can I help you today, Amir?"
  4   },
  5   "attributes": {
  6     "tokenUsage": {
  7       "outputCount": 14,
  8       "totalCount": 78,
  9       "inputCount": 64
  10    },
  11    "additionalAttributes": {
  12      "finish_reason": "stop",
  13      "model": "gpt-4o-mini-2024-07-18",
  14      "id": "chatcmpl-Bgv4Iirlc60XuuDZc1ES1BDsMCiH"
```

# Exercise 3 - Add Native MCP Support

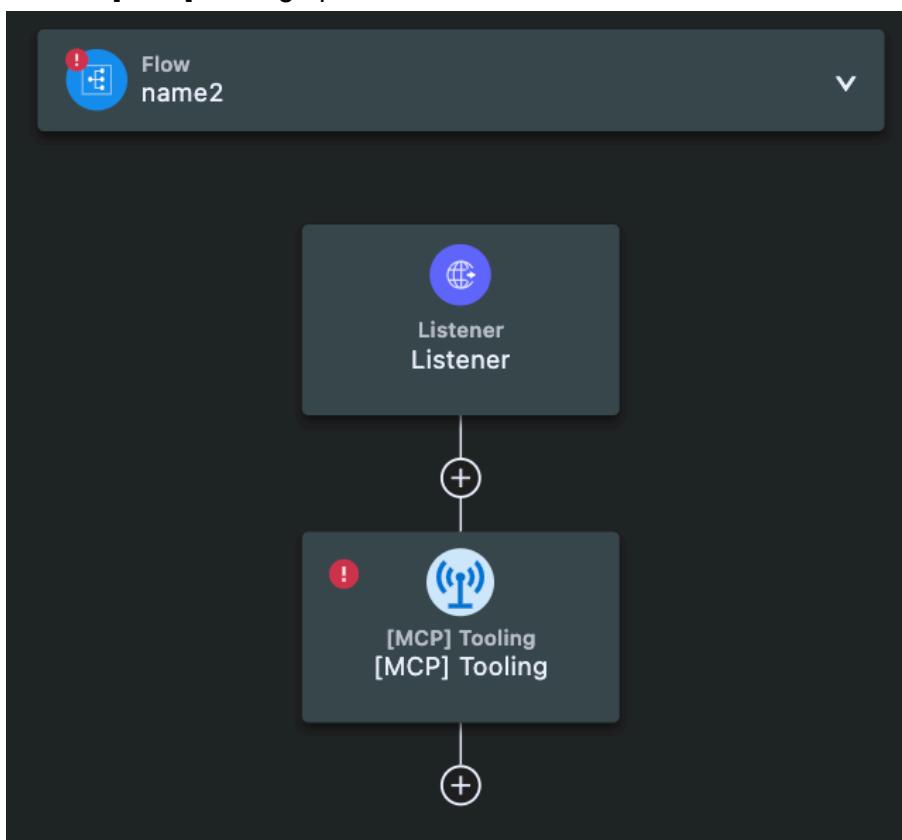
## Step 1

Drop a http listener with (/inference/mcp) into a new flow.



## Step 2

Add the [MCP] Tooling operations into the flow



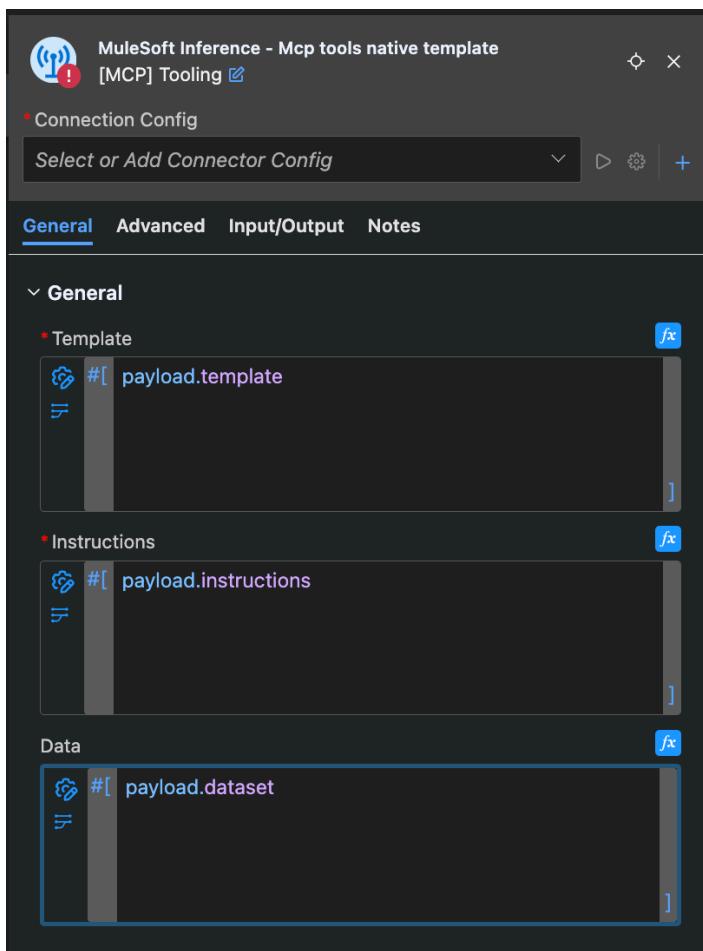
## Step 3

Add the following expression to the fields of the processor:

**Template:** payload.template

**Instructions:** payload.instructions

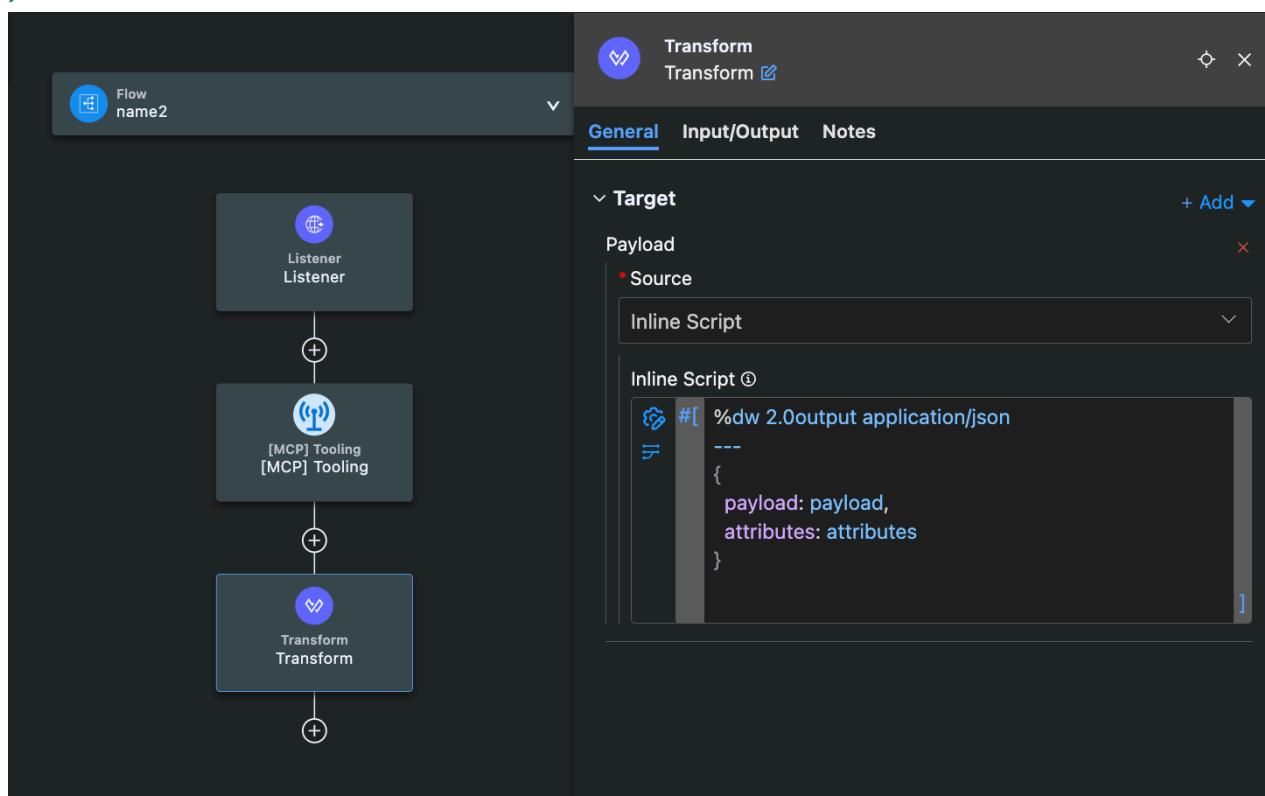
**Dataset:** payload.dataset



## Step 4

Drop a **Transform Message** after the MCP Tooling operation and copy the following expression.

```
%dw 2.0
output application/json
---
{
  payload: payload,
  attributes: attributes
}
```



## Step 5

Now configure the MCP Servers provided above in the Inference Text Generation configuration.

ERP Server: <http://mcp-server-demo-erp-ch1-application.us-e2.cloudhub.io>

CRM Server: <http://mcp-server-demo-crm-ch1-application.us-e2.cloudhub.io> DO NOT USE (you may face a timeout error)

The screenshot shows the 'Text-generation-config' page in the MuleSoft Inference interface. The left sidebar has tabs for 'General' (selected), 'Advanced', and 'Notes'. The main area contains configuration settings:

- Top p:** A field containing '0.9'.
- Timeout (milliseconds):** A field containing '#[ '60000']'.
- MCP ServerUrls (SSE over HTTP):** A section with two entries:
  - Key:** CRM Server
  - Value:** <http://mcp-server-demo-crm-b2jb0y.1d6nel.usa-e1.cloudhub.io>
- MCP ServerUrls (SSE over HTTP):** A section with two entries:
  - Key:** ERP Server
  - Value:** <http://mcp-server-demo-b2jb0y.1d6nel.usa-e1.cloudhub.io>

At the bottom, there are buttons for '+Add' and 'Remove All'.

## Step 6

Deploy the application locally.

```
*****
* Started app
*   'hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application'
*     Application plugin: *
*       MuleSoft Inference : 0.5.7
* - ObjectStore : 1.2.2
* - Socket : 1.2.5
* - HTTP : 1.10.3
*****
+-----+
+ Mule is up and kicking (every 5000ms) +
+-----+
*****
*   - + DOMAIN +           * - + STATUS +
*****                         *           *
* default                      * DEPLOYED  *
*****                         *           *
*****
*   + APPLICATION +           * + DOMAIN +           * + STATUS +
*****                         *           *           *
* hands-on-mule-inference-1.0.0-SNAPSHOT-mule-a * default      * DEPLOYED
*****                         *           *
*****
INFO 2025-06-11 06:08:32,898 [[MuleRuntime]]:uber.10: [hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application].uber.org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource:lambda$realLyDbStart$17:488 @245bcf0f1 [processor: ; event: ] org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource: Message source 'listener' on flow 'name1' successfully started
INFO 2025-06-11 06:08:32,898 [[MuleRuntime]]:uber.00: [hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application].uber.org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource:lambda$realLyDbStart$17:488 @06324cf53 [processor: ; event: ] org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource: Message source 'listener' on flow 'name2' successfully started
```

## Step 7

Use Postman or any other API Client to perform the following request:

```
curl --location 'http://localhost:8081/inference/mcp' \
--header 'Content-Type: application/json' \
--data '{
  "template": "You are an helpful assistant",
  "instructions": "Answer the request with politeness.",
  "dataset": "What is the capital of Switzerland"
}'
```

The screenshot shows the Postman interface with the following details:

- Request URL:** POST <http://localhost:8081/inference/mcp>
- Body:** JSON (Raw code shown)
- Response Status:** 200 OK
- Response Body (JSON):**

```
1 {
2   "payload": {
3     "response": "The capital of Switzerland is Bern.",
4     "tools": []
5   },
6   "attributes": {
7     "tokenUsage": {
8       "outputCount": 8,
9       "totalCount": 419,
10      "inputCount": 411
11    },
12    "additionalAttributes": {
13      "finish_reason": "stop",
14      "model": "gpt-4o-2024-08-06",
15      "id": "chatcmpl-Bh7Cjg0VTpKazqgqVL0Cw0JNb1qYs"
16    }
--}
```

## Step 8

Use Postman or any other API Client to perform the following request:

```
curl --location 'http://localhost:8081/inference/mcp' \
--header 'Content-Type: application/json' \
--data '{
  "template": "You are an helpful assistant",
  "instructions": "Answer the request with politeness.",
  "dataset": "Check Inventory for MULETEST0"
}'
```

The screenshot shows the Postman interface with the following details:

- Request URL:** POST <http://localhost:8081/inference/mcp>
- Body:** JSON (shown in the preview tab)
- Response Status:** 200 OK
- Response Body (JSON):**

```
1 {  
2   "payload": {  
3     "toolsExecutionReport": [  
4       {  
5         "result": {  
6           "productId": "MULETEST0",  
7           "availableQuantity": 1650521  
8         },  
9         "serverUrl": "http://mcp-server-demo-b2jb0y.1d6nel.usa-e1.cloudhub.  
io",  
10        "serverName": "ERP Server",  
11        "tool": "get_inventory",  
12        "timestamp": "2025-06-11T04:20:50.019617Z"  
13      }  
14    ],  
15    "tools": [  
16      {  
17        "function": {  
18          "name": "Inventory Check",  
19          "status": "Success",  
20          "details": "Available quantity: 1650521"  
21        }  
22      }  
23    ]  
24  }  
25}
```

Red arrows highlight specific fields in the response body:

- An arrow points to the "productId" field: "productId": "MULETEST0",
- Two arrows point to the "serverUrl" field: "serverUrl": "http://mcp-server-demo-b2jb0y.1d6nel.usa-e1.cloudhub.io",
- Two arrows point to the "serverName" field: "serverName": "ERP Server",

## Step 9

Use Postman or any other API Client to perform the following request:

```
curl --location 'http://localhost:8081/inference/mcp' \
--data '{
  "template": "You are an helpful assistant",
  "instructions": "Answer the request with politeness.",
  "dataset": "Get all CRM accounts for InnovateX"
}'
```

The screenshot shows a Postman request to `http://localhost:8081/inference/mcp`. The request method is POST. The request body is a JSON object:

```
1 {
  2   "template": "You are an helpful assistant",
  3   "instructions": "Answer the request with politeness.",
  4   "dataset": "Get all CRM accounts for InnovateX"
}
```

The response status is 200 OK. The response body is a JSON object:

```
1 {
  2   "payload": {
  3     "toolsExecutionReport": [
  4       {
  5         "result": {
  6           "accounts": [...]
  7         }
  8       },
  9       {
 10         "serverUrl": "http://mcp-server-demo-crm-b2jb0y.1d6nel.usa-e1.cloudhub.io",
 11         "serverName": "CRM Server",
 12         "tool": "get_accounts",
 13         "timestamp": "2025-06-11T04:22:18.732633Z"
 14       }
 15     ],
 16     "tools": [
 17       {
 18         "function": {
 19           "name": "get_accounts",
 20         }
 21       }
 22     ]
 23   }
 24 }
```

Three specific fields in the response body are highlighted with red arrows:

- `serverUrl`: `http://mcp-server-demo-crm-b2jb0y.1d6nel.usa-e1.cloudhub.io`
- `serverName`: `CRM Server`
- `tool`: `get_accounts`

## **Exercise 4 - Implement A2A Standard with your Agent**

**A2A Connector 0.1.0-BETA** can be added through Anypoint Exchange.

## Step 1

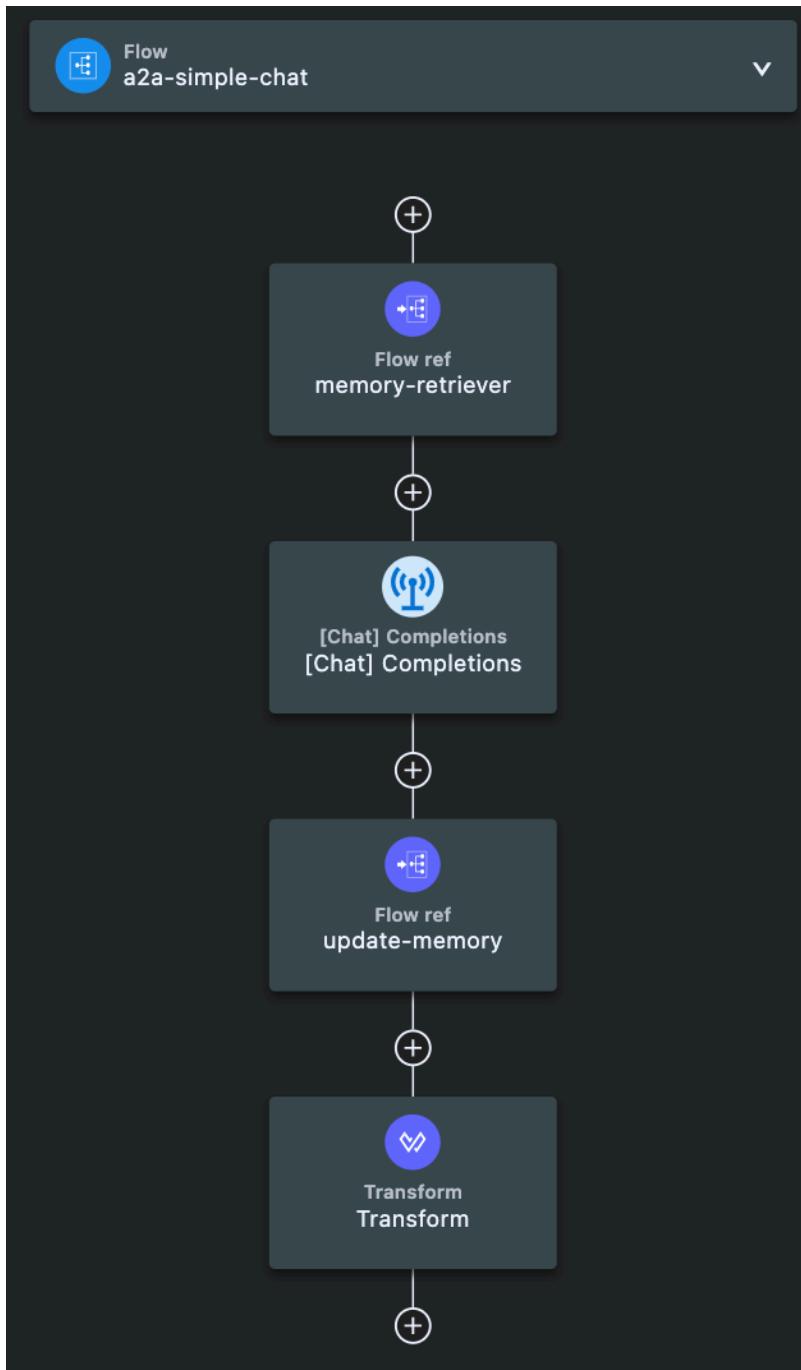
Delete the Http Listener from the flow with the listener (/inference/chat) and rename it to a2a-simple-chat.

The screenshot shows the 'Add Component' dialog with the 'Connectors' category selected. The list includes various connectors with their counts and versions:

Connector	Count	Version	Action
A2A (3)	3	0.1.0-BETA	>
Agentforce (4)	4	1.0.1	>
APIKit (2)	2	1.11.5	>
Database (11)	11	1.14.13	>
Einstein AI (9)	9	1.1.1	>
Email (10)	10	1.7.4	>
File (8)	8	1.5.2	>
HTTP (3)	3	1.10.3	>
Java (4)	4	1.2.13	>
MuleSoft Inference (8)	8	0.5.7	>
MUnit (2)	2	3.4.0	>
ObjectStore (7)	7	1.2.2	>
Salesforce (68)	68	10.22.8	>
Salesforce Data Cloud (19)	19	1.2.1	>
Sockets (2)	2	1.2.5	>
Web Service Consumer (1)	1	1.11.0	>
Workday (50)	50	16.2.2	>

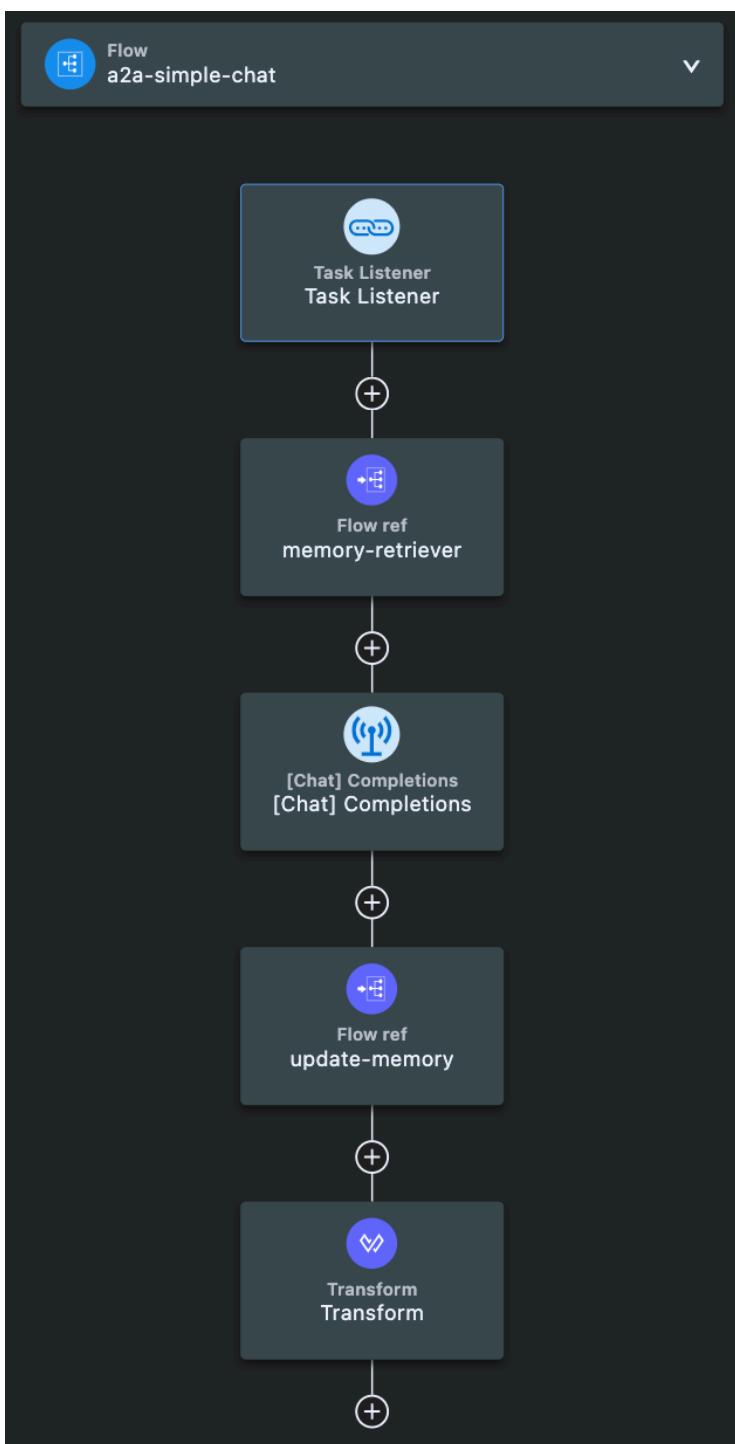
## Step 2

Delete the Http Listener from the flow with the listener (/inference/chat) and rename it to a2a-simple-chat.



## Step 3

Add A2A Task Listener instead into the new flow as a source.



## Step 4

Now add the following 2 configuration to your app.

Here we are adding a dedicated Http Listener for the A2A Task Listener and its Agent Configuration including Agent Card.

```
<http:listener-config name="HTTP_Listener_config-a2a-server"
doc:name="HTTP Listener config"
doc:id="133e35cd-f00a-4e7e-8e9f-cb125a4af2a" >
<http:listener-connection host="0.0.0.0" port="9081" />
</http:listener-config>
<a2a:server-config name="A2A_Server" doc:name="A2A Server"
doc:id="ac631edc-6db4-4f8f-8ea1-868d8badaedd" >
<a2a:connection listenerConfig="HTTP_Listener_config-a2a-server"
agentPath="/a2a-agent" />
<a2a:card name="A2A Agent" url="http://localhost:9081/a2a-agent"
version="1.0.0" >
<a2a:description ><![CDATA[This is a simple chat
agent]]></a2a:description>
<a2a:skills >
<a2a:agent-skill id="1" name="ChatMemory" >
<a2a:description ><![CDATA[This agent has the ability to remember
conversation]]></a2a:description>
</a2a:agent-skill>
</a2a:skills>
</a2a:card>
</a2a:server-config>
```

A 2 a - Task listener

## A2A\_Server

...

Configure a connection for this project. Updates will impact all components using this connection.

### General

### Advanced

### Notes

\* Name

A2A\_Server

\* Connection

Connection

✓ General

\* Connection Config

HTTP\_Listener\_config-a2a-server



\* Agent path

/a2a-agent

\* Card

\* Agent Name

A2A Agent

\* Agent URL

http://localhost:9081/a2a-agent

\* Agent Version

1.0.0

Agent Description

## Step 5

Update the memory-retriever sub-flow with the following expression as replacement.

We are not anymore looking for the payload.messages for the userPrompt, but instead getting the userPrompt from an standarad a2a structured payload and addiitonally also setting the task Id, as well as the session Id as variables.

```
<sub-flow name="memory-retriever"
doc:id="72bf2565-f2b4-4533-ba5e-14f385b6b1c1" >
<set-variable value="#[payload.message.parts[0].text default
payload.messages]" doc:name="Set task user prompt"
doc:id="c15af2c5-dd80-4d99-b4e5-9df0aee6898b" variableName="userPrompt"
/>
<set-variable value="#[payload.id]" doc:name="Set Task Id"
doc:id="c4602681-f811-4d86-8088-56ad30b2b4bf" variableName="task_id" />
<set-variable value='#[(payload.sessionId) default ""]' doc:name="Set
Session Id" doc:id="5d513a8f-e069-4edf-a3c8-1fd88b0c1373"
variableName="session_id" />
<set-variable value='#[attributes.headers."userId" default
"amir-khan"]' doc:name="User Id"
doc:id="28824951-7f99-4313-b74f-50fd66fb94b" variableName="userId" />
<os:retrieve doc:name="Load Memory"
doc:id="4e5a4fb2-ca70-44dc-84f7-f5de36676a44" key="#[vars.userId]"
objectStore="Object_store" target="memory">
<os:default-value ><![CDATA[#[output application/json
---<
{
chat_history: []
}]]></os:default-value>
</os:retrieve>
</sub-flow>
```

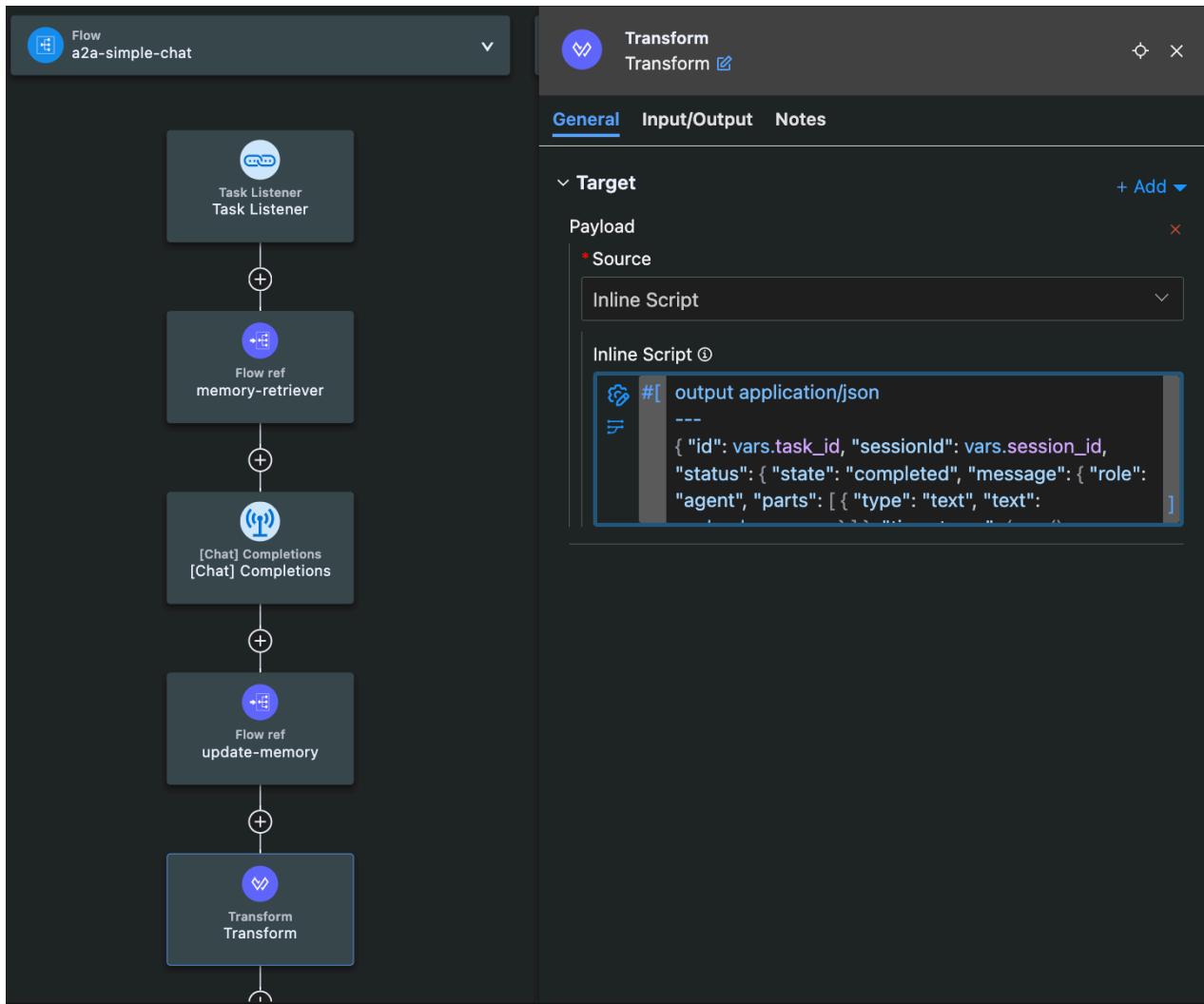


## Step 6

The Transform Message processor needs to be updated with the following dataweave expression as a replacement. This is the A2A response format to comply to the A2A protocol.

We are setting 3 A2A specific attributes with taskId, sessionId, and the text in the parts as the response.

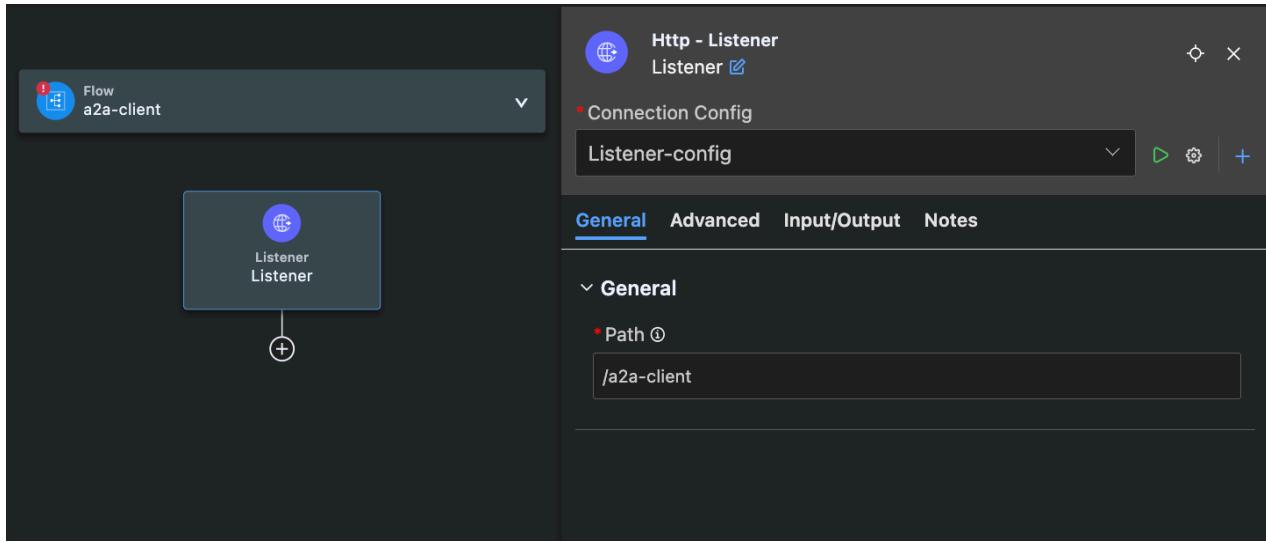
```
%dw 2.0
output application/json
---
{
  "id": vars.task_id,
  "sessionId": vars.session_id,
  "status": {
    "state": "completed",
    "message": {
      "role": "agent",
      "parts": [
        {
          "type": "text",
          "text": payload.response
        }
      ]
    },
    "timestamp": (now() >> 'UTC') as String {format:
      "yyyy-MM-dd'T'HH:mm:ss'Z'"
    },
    "artifacts": [
      {
        "name": "Answer",
        "index": 0,
        "parts": [
          {
            "type": "text",
            "text": payload.response
          }
        ]
      }
    ]
  }
}
```



## Step 7

Now our A2A Server Agent is ready. We need to add now a A2A client call to the same app in order to communicate with the A2A server agent. We drop a new http listener (/inference/a2a-client) in to a new flow (a2a-client).

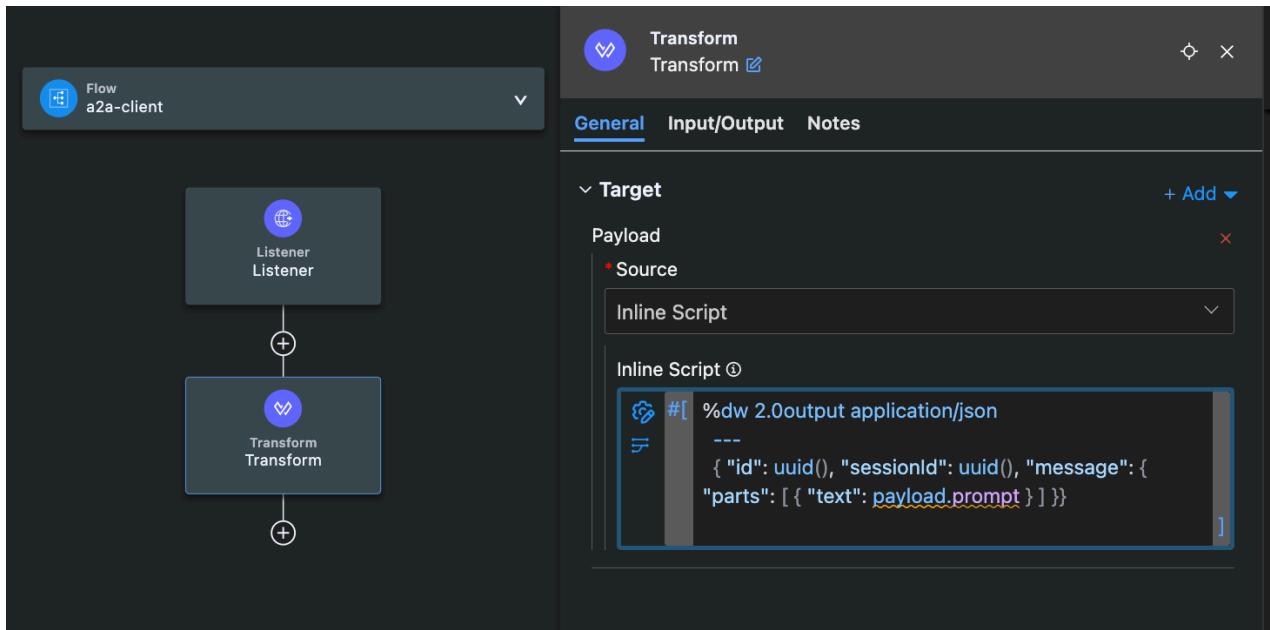
Make sure to reuse the http configuration running on 8081 and not the a2a server http configuration.



## Step 8

Add a Transform Message into the a2a-client flow with the following dataweave expression. This is the standard way of doing a client call to an A2A server. Key attributes as TaskId, SessionId and message must be provided.

```
%dw 2.0
output application/json
---
{
  "id": uuid(),
  "sessionId": uuid(),
  "message": {
    "parts": [
      {
        "text": payload.prompt
      }
    ]
  }
}
```

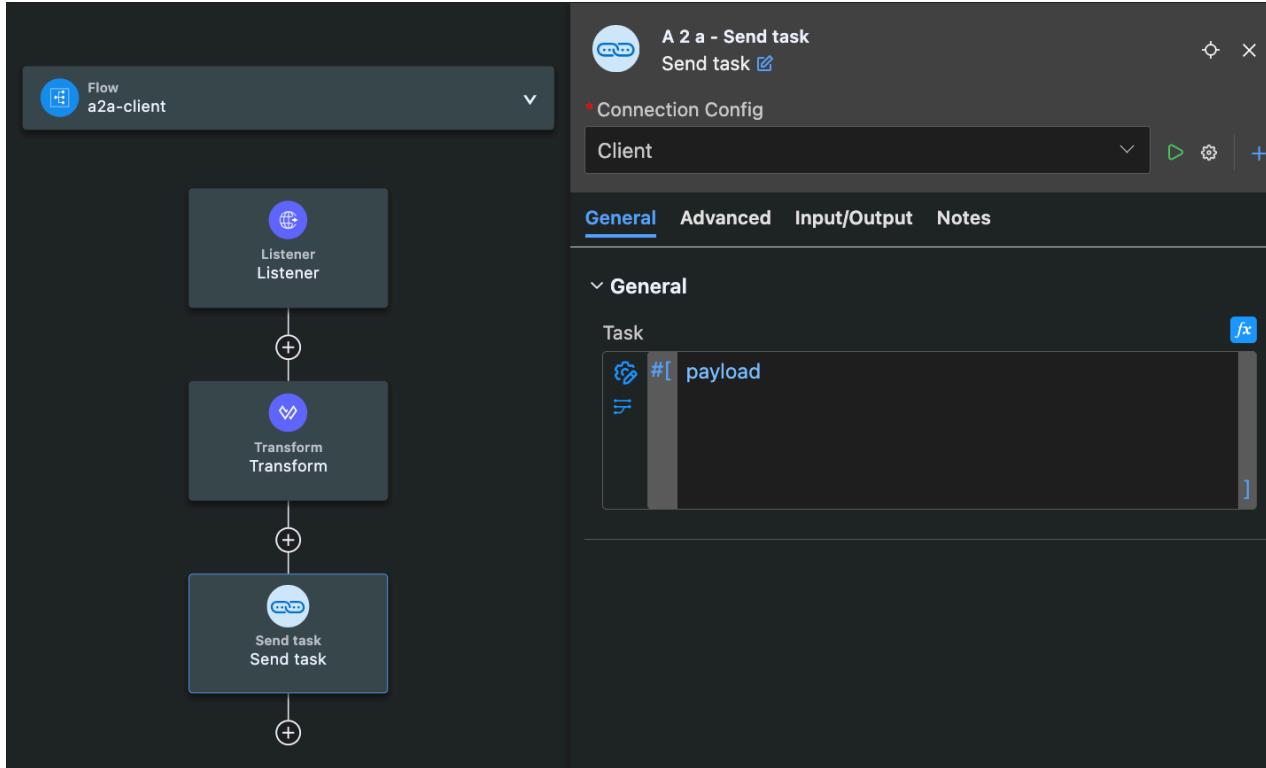


## **Step 9**

Finally add a Send Task processor from the A2A Connector after the Transform Message operation.

In the configuration add the following agent url:

<http://localhost:9081/a2a-agent>



## Step 10

Deploy the application locally

```
PROBLEMS 82 OUTPUT DEBUG CONSOLE TERMINAL PORTS Running hands-on-mule-inference
* Application plugins:
*   - MuleSoft Inference : 0.5.7
*   - ObjectStore : 1.2.2
*   - Sockets : 1.2.5
*   - HTTP : 1.10.3
*   - AZA : 0.1.0-BETA
*****
+ Mule is up and kicking (every 5000ms)
*****
***** + DOMAIN + - - + STATUS + - - *
***** + hands-on-mule-inference-1.0.0-SNAPSHOT-mule-a + default * DEPLOYED *
*****
***** + DOMAIN + - - + STATUS + - - *
***** + hands-on-mule-inference-1.0.0-SNAPSHOT-mule-a + default * DEPLOYED *
*****
INFO 2025-06-11 08:04:27,500 [[MuleRuntime].uber.09: [hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application].uber.org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource.lambda$readAll$0@71d188a} : event] [uber.org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource$readAll$0@71d188a] successfully started
INFO 2025-06-11 08:04:27,500 [[MuleRuntime].uber.02: [hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application].uber.org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource$readAll$0@204b692] [processor: ; event:] [org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource: Message source 'listener' on flow 'aza-client' successfully started]
INFO 2025-06-11 08:04:27,500 [[MuleRuntime].uber.05: [hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application].uber.org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource$lambda$readAll$0@71d47fb] [processor: ; event:] [org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource: Message source 'listener' on flow 'mcp-tooling' successfully started]
INFO 2025-06-11 08:04:27,500 [[MuleRuntime].uber.04: [hands-on-mule-inference-1.0.0-SNAPSHOT-mule-application].uber.org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource$lambda$readAll$0@6af4f36] [processor: ; event:] [org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource: Message source 'TaskListener' on flow 'aza-simple-chat' successfully started]
```

## Step 11

Use postman or any other API Client to execute the following request:

```
curl --location 'localhost:8081/inference/a2a-client' \
--header 'userId: amir-khan' \
--header 'Content-Type: application/json' \
--data '{
  "prompt": [
    {
      "role": "user",
      "content": "What is the capital of Switzerland"
    }
  ]
}'
```

The screenshot shows a Postman request configuration and its response.

**Request Configuration:**

- Method: POST
- URL: localhost:8081/inference/a2a-client
- Body tab is selected.
- JSON selected under "raw".
- Body content:

```
1 {
  2   "prompt": [
  3     {
  4       "role": "user",
  5       "content": "What is the capital of Switzerland"
  6     }
  7   ]
}
```

**Response:**

- Status: 200 OK
- Time: 2.02 s
- Size: 691 B
- Timestamp: 2025-06-11T06:04:32Z
- Message content:

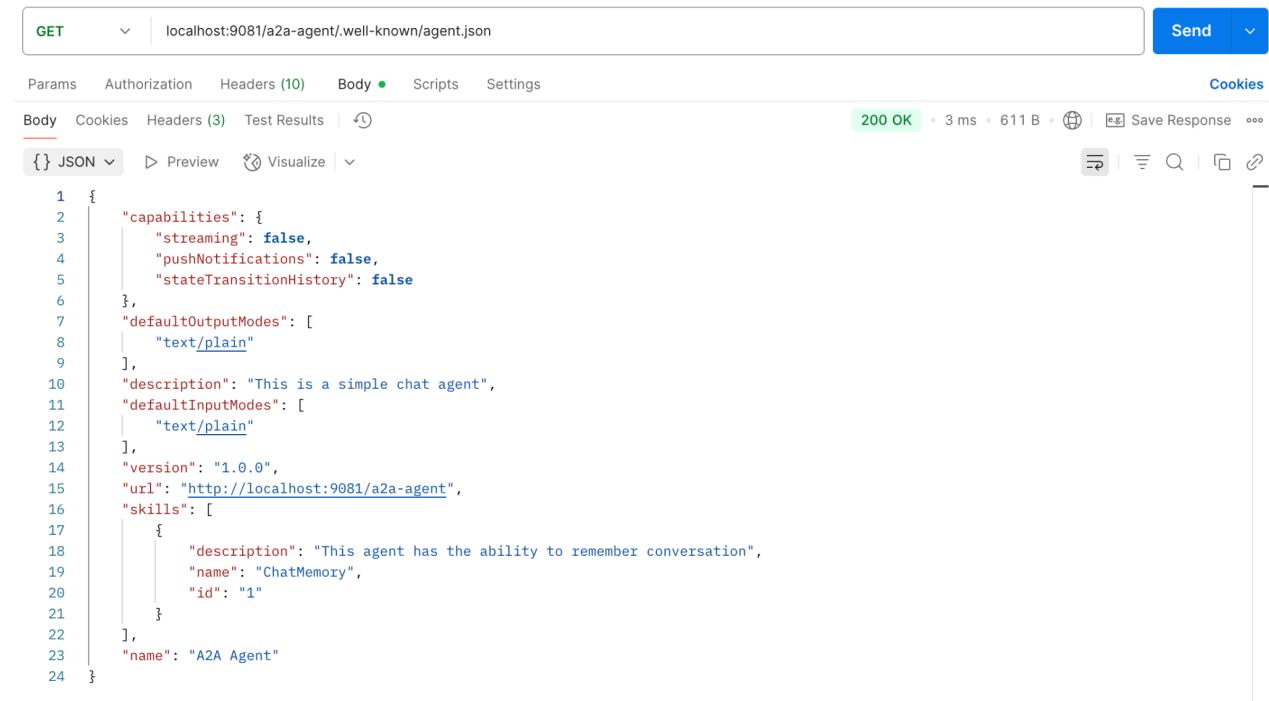
```
1 {
  2   "id": "1374efcf-410b-40f2-b3d7-63e177dcae05",
  3   "sessionId": "ae812219-59f7-4911-a7db-71b24fc090e8",
  4   "status": {
  5     "state": "completed",
  6     "message": {
  7       "role": "agent",
  8       "parts": [
  9         {
  10           "type": "text",
  11           "text": "The capital of Switzerland is Bern."
  12         }
  13       ],
  14     },
  15     "timestamp": "2025-06-11T06:04:32Z"
  16   }
}
```

## Step 12

Get the **agent card** by using the following request.

```
curl --location --request GET  
'localhost:9081/a2a-agent/.well-known/agent.json' \  
--header 'Content-Type: application/json' \  

```



The screenshot shows a browser-based API testing interface. At the top, there is a URL input field containing "localhost:9081/a2a-agent/.well-known/agent.json". Below the URL, there are tabs for "Params", "Authorization", "Headers (10)", "Body", "Scripts", and "Settings". The "Body" tab is selected and displays the JSON response. The response is a multi-line JSON object with line numbers from 1 to 24 on the left. The JSON structure includes fields like "capabilities", "defaultOutputModes", "description", "defaultInputModes", "version", "url", and "skills". The "skills" field contains a single skill named "ChatMemory" with an ID of "1". The "name" field at the bottom is "A2A Agent". Above the JSON, there is a status bar showing "200 OK", "3 ms", "611 B", and other icons. On the right side of the interface, there are various buttons and icons for managing the response.

```
1 {  
2   "capabilities": {  
3     "streaming": false,  
4     "pushNotifications": false,  
5     "stateTransitionHistory": false  
6   },  
7   "defaultOutputModes": [  
8     "text/plain"  
9   ],  
10  "description": "This is a simple chat agent",  
11  "defaultInputModes": [  
12    "text/plain"  
13  ],  
14  "version": "1.0.0",  
15  "url": "http://localhost:9081/a2a-agent",  
16  "skills": [  
17    {  
18      "description": "This agent has the ability to remember conversation",  
19      "name": "ChatMemory",  
20      "id": "1"  
21    }  
22  ],  
23  "name": "A2A Agent"  
24 }
```

# **Vectors Hands-on**

# Vectors Connector Hands-on

## Prerequisites

- [Setup PostgreSQL from SE Platform](#)

## ACB (Vectors)

```
#heroku-hands-on configurations
herokuhandson:
  host: cloud-services.demos.mulesoft.com
  port: '30389'
  username: postgres
  databaseName: sampledb
  password: TVK4pah_tze3dqe5qvx
  rootPassword: ''
```

# **ACB (Vectors)**

# Exercise 1- Data Load

## Step 1

Create a new mule app `hands-on-vectors-app` using Studio or ACB.

### Develop an Integration

Create and test a Mule application project that integrates existing services.

Project Name

Project Location  
 Browse

Create

Empty Project  
*Create an integration project from scratch.*

Template or Example Project  
*Start an integration with a template or example project from Anypoint Exchange.*

Mule Runtime ①

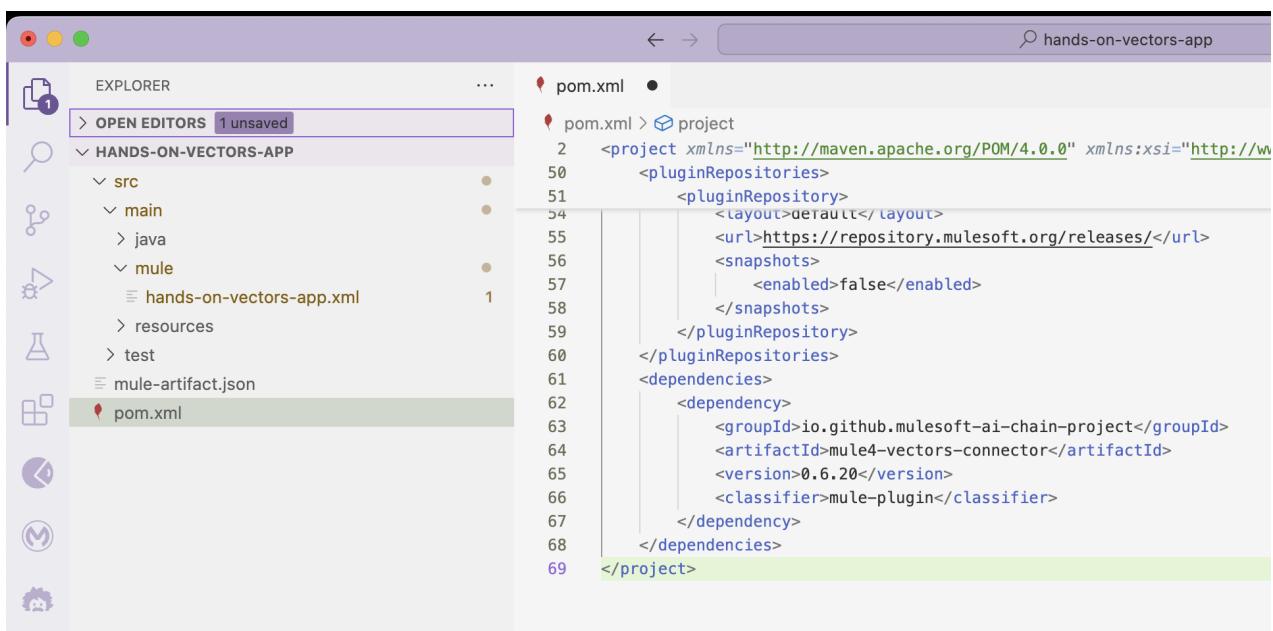
Java Version ①

Cancel Create Project

## Step 2

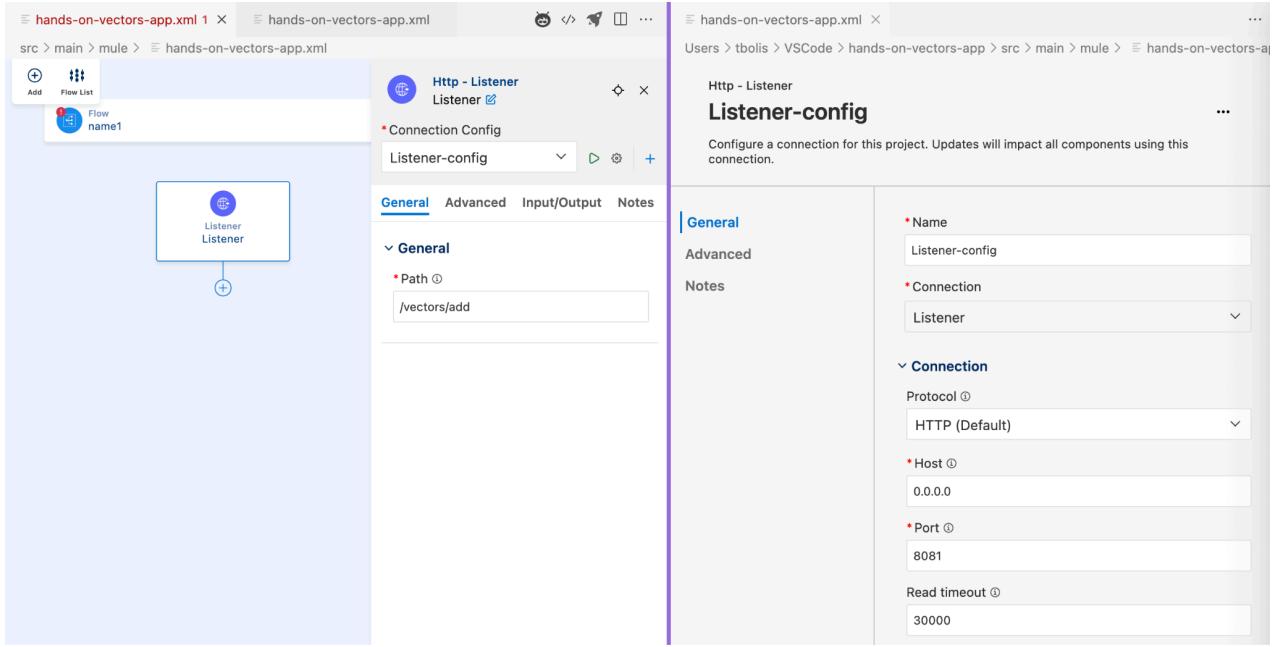
Add Vectors connector dependency to your pom.xml

```
...
<dependencies>
...
<dependency>
<groupId>io.github.mulesoft-ai-chain-project</groupId>
<artifactId>mule4-vectors-connector</artifactId>
<version>0.6.20</version>
<classifier>mule-plugin</classifier>
</dependency>
...
</dependencies>
...
```



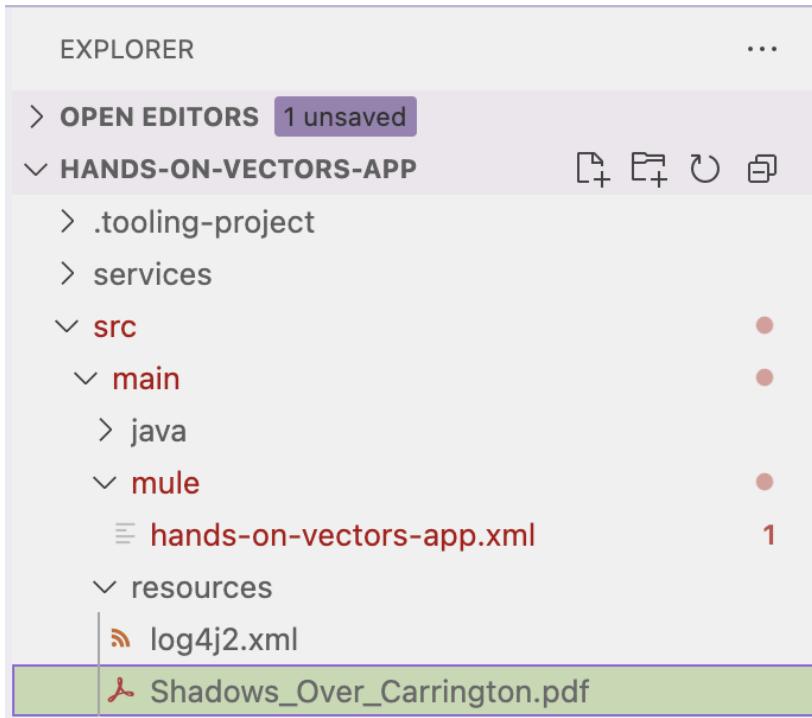
## Step 3

Drop http listener (`/vectors/add`) in a new flow, you'll use it to load data into store using postman



## Step 4

Save the [test file](#) under your src/main/resources folder.



## Step 5

Drop [Document] Load single operation after listener. Leave segmentation parameters empty.

Set:

- File Type to **any**
- Context Path to **Shadows\_Over\_Carrington.pdf**

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, a flow diagram titled "Flow add-to-store" is displayed. It starts with a "Listener Listener" component, followed by a connector component labeled "[Document] Load single [Document] Load si...". On the right, a configuration panel for the "MuleSoft Vectors Connector - Document load single" is open. The "Connection Config" dropdown is set to "Storage-config". The "General" tab is selected, showing the "File Type" as "any" and the "Context Path" as "Shadows\_Over\_Carrington.pdf". Below the general tab, the "Segmentation" tab is visible, containing fields for "Max Segment Size (Characters)" and "Max Overlap Size (Characters)".

## Step 6

Create a Storage Configuration (with Local Connection). Set `<project-path>/src/main/resources` as working dir.

MuleSoft Vectors Connector - Document load single

## Storage-config

...

Configure a connection for this project. Updates will impact all components using this connection.

### General

Advanced

Notes

\* Name

Storage-config

\* Connection

Local

▼

### Connection

Working Directory

fx

/Users/tbolis/VSCode/hands-on-vectors-app/src/main/re 

 Test Connection

Cancel

Apply

## Step 7

Add Shared Libraries.

- ACB: directly edit the pom.xml

```
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-clean-plugin</artifactId>
<version>3.1.0</version>
</plugin>
```

```
<plugin>
<groupId>org.mule.tools.maven</groupId>
<artifactId>mule-maven-plugin</artifactId>
<version>${mule.maven.plugin.version}</version>
<extensions>true</extensions>
<configuration>
<sharedLibraries>
<sharedLibrary>
<groupId>dev.langchain4j</groupId>
<artifactId>langchain4j</artifactId>
</sharedLibrary>
<sharedLibrary>
<groupId>dev.langchain4j</groupId>
<artifactId>langchain4j-document-transformer-jsoup</artifactId>
</sharedLibrary>
<sharedLibrary>
<groupId>dev.langchain4j</groupId>
<artifactId>langchain4j-document-parser-apache-tika</artifactId>
</sharedLibrary>
...
</sharedLibraries>
</configuration>
</plugin>
</plugins>
</build>
...
<dependencies>
...
<dependency>
<groupId>dev.langchain4j</groupId>
<artifactId>langchain4j</artifactId>
<version>1.0.1</version>
</dependency>
<dependency>
<groupId>dev.langchain4j</groupId>
<artifactId>langchain4j-document-transformer-jsoup</artifactId>
<version>1.0.1-beta6</version>
</dependency>
<dependency>
<groupId>dev.langchain4j</groupId>
<artifactId>langchain4j-document-parser-apache-tika</artifactId>
<version>1.0.1-beta6</version>
```

```

</dependency>
...
</dependencies>
...

```

## Step 8

Start your application

Hit <http://localhost:8081/vectors/add> and check response.

The screenshot shows a POST request to <http://localhost:8081/vectors/add>. The response is 200 OK with a response time of 703 ms and a size of 5.02 KB. The response body is a JSON object:

```

1  {
2   "text-segments": [
3     {
4       "metadata": {
5         "absolute_directory_path": "/Users/tbolis/VSCode/hands-on-vectors-app/src/main/resources",
6         "index": "0",
7         "file_name": "Shadows_Over_Carrington.pdf",
8         "file_type": "any"
9       },
10      "text": "\n Shadows Over Carrington \n Carrington was the kind of town where nothing ever happened—until it did. On a chilled au
11    ]
12  }
13

```

## Exercise 2 - Data Splitting

Start from the exercise 1 app

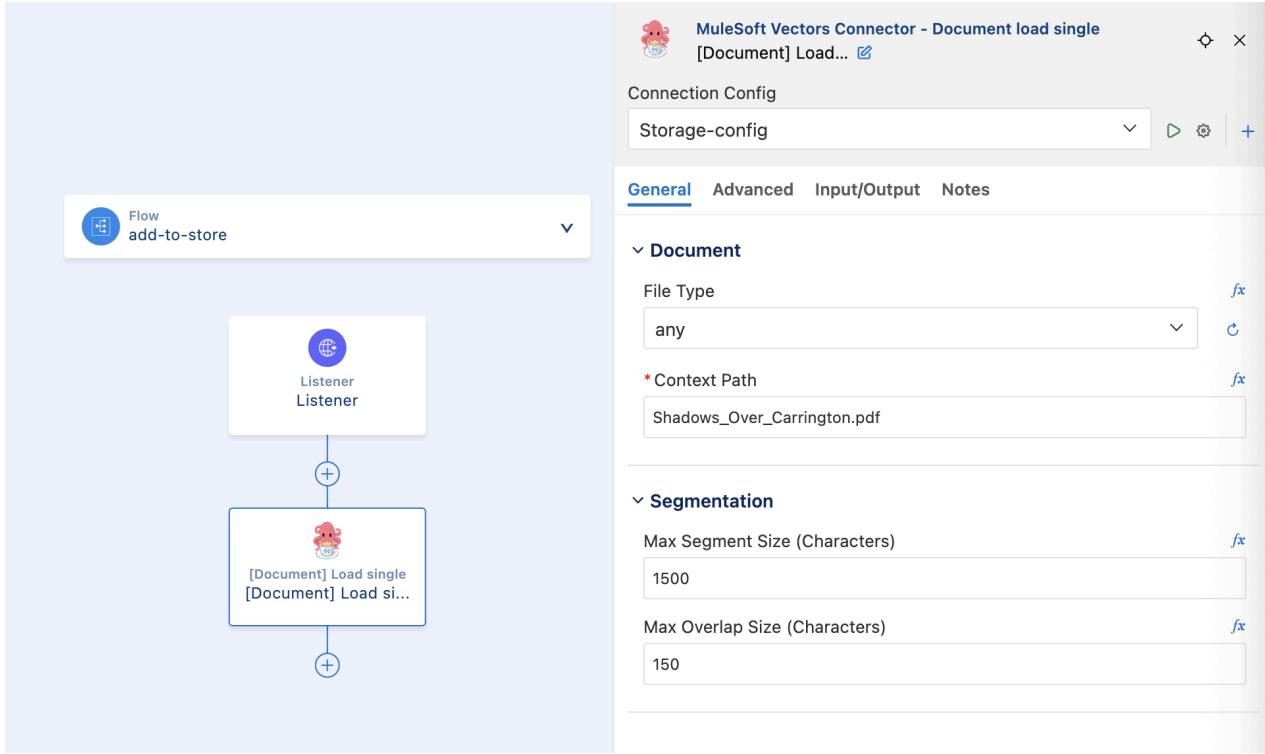
### Step 1

Update [Document] Load single operation you have after listener.

Set segmentation parameters as follow:

- \* Max segmentation chars = 1500

- \* Max overlaps chars=150



## Step 2

Restart your application

Hit <http://localhost:8081/vectors/add> and check response.

The screenshot shows a POST request to `http://localhost:8081/vectors/add`. The response status is `200 OK` with a duration of `1.80 s` and a size of `5.72 KB`. The JSON response body contains the following data:

```

1  {
2   "text-segments": [
3     {
4       "metadata": {
5         "absolute_directory_path": "/Users/tbolis/VSCode/hands-on-vectors-app/src/main/resources",
6         "index": "0",
7         "file_name": "Shadows_Over_Carrington.pdf",
8         "file_type": "any"
9       },
10      "text": "Shadows Over Carrington \n Carrington was the kind of town where nothing ever happened-until it did. On a chilled autumn evening, a young reporter named Emily Hartley arrived at the Carrington Police Department to interview the chief of police, Captain John Miller. Emily had been assigned to write a story about the department's recent string of missing persons cases. She was surprised to find that Captain Miller was not available for an interview. Instead, he had sent her a copy of his favorite book, 'Shadows Over Carrington'. Emily took the book back to her office and began reading it. As she turned the pages, she noticed that the text was printed in a dark, shadowy font that seemed to bleed through from the other side of the page. She was fascinated by the book and decided to use it as the basis for her article. She wrote a compelling piece about the book and its author, and it became a hit with readers. The book also inspired her to look into the mysterious disappearances that had plagued the town for years. She discovered that there had been several cases of people disappearing without a trace, and that the police had never solved any of them. Emily was determined to find out what had happened to these people, and she started investigating on her own. She soon learned that the town had a dark history of corruption and violence. She uncovered evidence of police officers being paid off by drug dealers and gangsters. She also found out that the police chief himself was involved in some shady dealings. Emily's investigation led her to a shocking discovery: the police chief had been responsible for many of the disappearances. He had been using his power to cover up the truth and protect the guilty. Emily's exposé shocked the town, and it led to a major scandal. The police chief was arrested and charged with corruption and obstruction of justice. The town was left reeling, and it was a reminder of the dark side of power and authority. The book 'Shadows Over Carrington' became a best-seller, and it inspired a new generation of readers to look beyond the surface and explore the hidden depths of society. It remains a classic work of fiction that continues to captivate readers to this day."
11    },
12    {
13      "metadata": {
14        "absolute_directory_path": "/Users/tbolis/VSCode/hands-on-vectors-app/src/main/resources",
15        "index": "1",
16        "file_name": "Shadows_Over_Carrington.pdf",
17        "file_type": "any"
18      },
19      "text": "Inside was a list of case numbers and a \n note: \"They said they'd come back.\"\\n\\nThe cases on the list were all linked to the same person: a man named John Miller. Emily was curious about this and decided to investigate further. She found out that John Miller was the police chief of Carrington. She was shocked to learn that he had been responsible for many of the disappearances. She decided to write a story about him and his corruption. Her story was published in a local newspaper and went viral online. It became a sensation and attracted the attention of a major publishing house. They offered her a contract to write a full-length novel based on her investigation. Emily accepted the offer and began writing. She spent months researching and writing, and finally completed the novel. It was a massive success and became a best-seller. Emily became a household name and received numerous awards for her work. She continued to write and publish books, becoming one of the most popular authors of her generation. Her legacy lives on through her books and the impact they had on society. She will always be remembered as a trailblazer who dared to expose the dark truths of power and corruption. Her story is a testament to the power of journalism and the importance of speaking truth to power. It serves as a reminder that even in the darkest of times, there is always hope and the possibility of change. The book 'Shadows Over Carrington' is a timeless classic that continues to inspire and educate readers to this day."
20  },
21  ...
22  ],
23  [
24    {
25      "metadata": {
26        "absolute_directory_path": "/Users/tbolis/VSCode/hands-on-vectors-app/src/main/resources",
27        "index": "3",
28        "file_name": "Shadows_Over_Carrington.pdf",
29        "file_type": "any"
30      },
31      "text": "In short:\\n\\nIt's a dark tale of justice, corruption, and long-buried secrets catching up with those who \\n thought they could get away with it. It's a reminder that the past never truly goes away, and that sometimes, the shadows still lurk in the corners. It's a powerful story that will stay with you long after you've finished reading it. If you're looking for a gripping, suspenseful read, then 'Shadows Over Carrington' is definitely worth checking out. You won't be disappointed."\\n\\n"
32    }
33  ]
34 }
35 ]
36 ]
37 ]
38 ]
39 ]
40 ]

```

## Exercise 3 - Embedding Generation

Start from the exercise 2 app

#hands-on-vectors configurations

handsonvectors:

Salesforce Org: storm-2fb9bac3f0ef38.my.salesforce.com

Consumer Key:

3MVG9kb26yEQGZW3tPydgyJYjCFxn9y\_ipKZvKTxaEP6Kk\_3Iwd5VlES3167etTkqMGFyJt

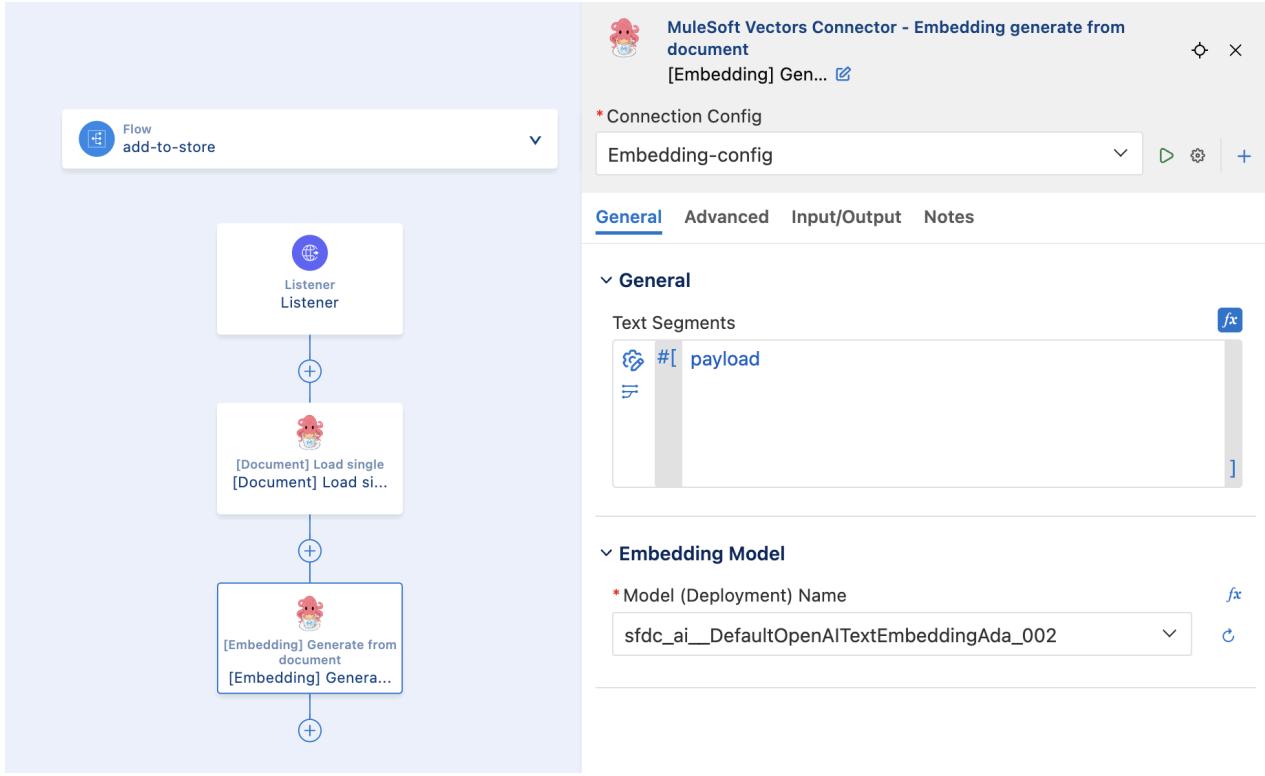
RUFerXj0tpeks2

Consumer Secret:

2CDA7023D43C6626B5E3A2BBF440D6EEE5CF0FD1708FABA8B24888F9B68A8E5A

### Step 1

Drop [Embedding] Generate from Document operation right after the [Document] Load single operation.



## Step 2

Create an Embedding Configuration using either Einstein or Azure Open AI (from Embark) as model connection.

## Embedding-config

...

Configure a connection for this project. Updates will impact all components using this connection.

### General

Advanced

Notes

\* Name

Embedding-config

\* Connection

Einstein

### Connection

\* Salesforce Org

fx

storm-2fb9bac3f0ef38.my.salesforce.com

\* Client id

fx

3MVG9kb26yEQGZW3tPydgyJYjCFxn9y\_ipKZvKTxaEP6KI

\* Client secret

fx

.....@

▷ Test Connection

Cancel

Apply

## Step 3

Start your application

Hit <http://localhost:8081/vectors/add> and check response.

```

1 {
2   "embeddings": [
3     [
4       0.02354035,
5       0.006446402,
6       0.009573727,
7       -0.0066032913,
8       -0.008743958,
9       0.018031796,
10      -0.0048775105,
11      -0.032939754,

```

## Exercise 4 - Store Embeddings

```
#heroku-hands-on configurations
herokuhandson:
  host: cloud-services.demos.mulesoft.com
  port: '30389'
  username: postgres
  databaseName: sampledb
  password: TVK4pah_tze3dqe5qvx
  rootPassword: ''
```

Start from the exercise 3 app

### Step 1

Drop [Store] Add operation right after the [Embedding] Generate from Document operation. Configure the operation by setting a store name.

Store name: `hands_on_vectors_<your-user>`

**MuleSoft Vectors Connector - Store add**

[Store] Add

Connection Config

Store-config

**General** Advanced Input/Output Notes

\* Store name ⓘ hands\_on\_vectors

Text Segments and Embeddings

#[ payload ]

**Custom Metadata**

Metadata entries

+ Add

## Step 2

Create a Store Configuration using PGVector as store connection.

MuleSoft Vectors Connector - Store add

## Store-config

Configure a connection for this project. Updates will impact all components using this connection.

**General**

**Advanced**

**Notes**

\* Name: Store-config

\* Connection: PGVector

**Connection**

\* Host: cloud-services.demos.mulesoft.com

\* Port: 30492

\* Database: sampledb

\* User: postgres

\* Password: ..... (obscured)

...

### Step 3

Add [Shared Libraries](#) by editing directly edit the pom.xml

```
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-clean-plugin</artifactId>
<version>3.1.0</version>
</plugin>
<plugin>
```

```
<groupId>org.mule.tools.maven</groupId>
<artifactId>mule-maven-plugin</artifactId>
<version>${mule.maven.plugin.version}</version>
<extensions>true</extensions>
<configuration>
<sharedLibraries>
<sharedLibrary>
<groupId>dev.langchain4j</groupId>
<artifactId>langchain4j-pgvector</artifactId>
</sharedLibrary>
...
</sharedLibraries>
</configuration>
</plugin>
</plugins>
</build>
...
<dependencies>
...
<dependency>
<groupId>dev.langchain4j</groupId>
<artifactId>langchain4j-pgvector</artifactId>
<version>1.0.1-beta6</version>
</dependency>
...
</dependencies>
...
```

## Step 4

Start your application

Hit <http://localhost:8081/vectors/add> and check response.

HTTP MAC SME Development / add-to-store

POST <http://localhost:8081/vectors/add>

Params Authorization Headers (8) Body Scripts Settings

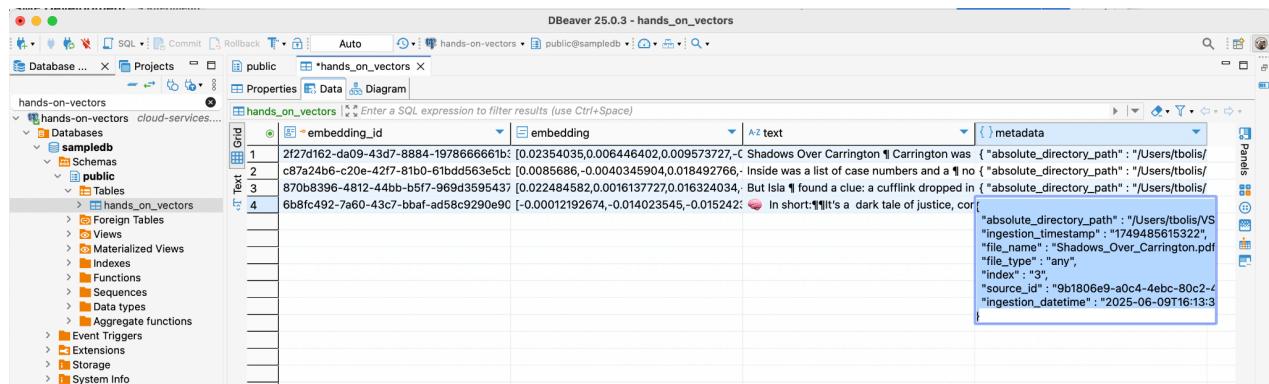
Body Cookies Headers (3) Test Results | ⏪

{ } JSON ▾ ▶ Preview ⚡ Visualize ▾

```
1 {  
2   "sourceId": "9b1806e9-a0c4-4ebc-80c2-4164c38c2dc2",  
3   "embeddingIds": [  
4     "2f27d162-da09-43d7-8884-1978666661b3",  
5     "c87a24b6-c20e-42f7-81b0-61bdd563e5cb",  
6     "870b8396-4812-44bb-b5f7-969d35954377",  
7     "6b8fc492-7a60-43c7-bbaf-ad58c9290e90"  
8   ],  
9   "status": "updated"  
10 }
```

## Step 5

Connect to your PostgreSQL database using a client as DBeaver and check the data.



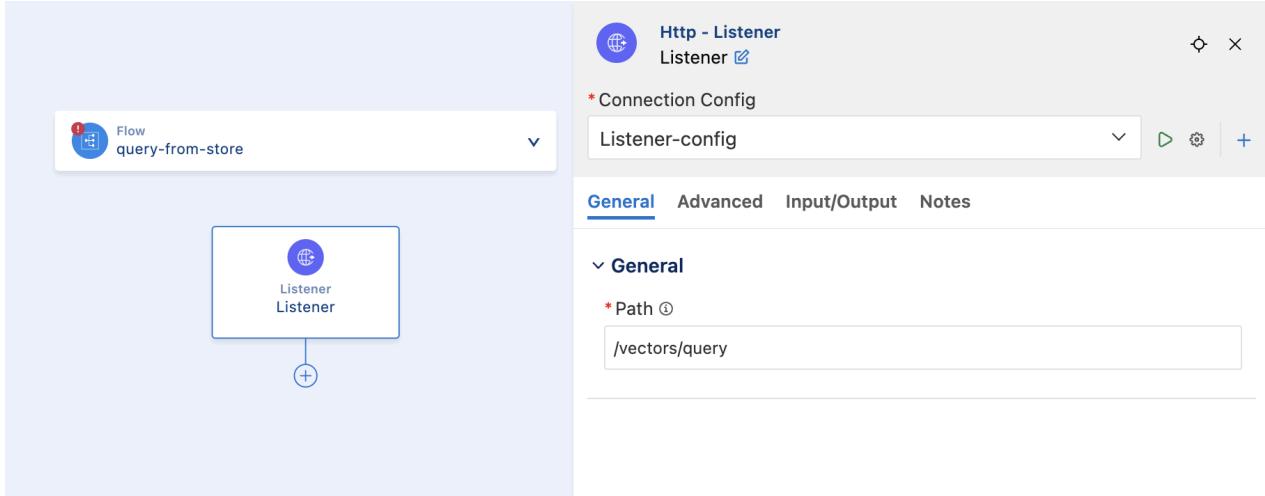
## Exercise 5 - Query Vector Store

Start from the exercise 4 app.

### Step 1

Drop http listener (/vectors/query) in a new flow, you'll use it to query the store using postman. It should accept a json payload like

```
{
  "prompt": "question"
}
```



## Step 2

Drop [Embedding] Generate from Text operation after the listener. Set payload.prompt as input.

The screenshot shows a MuleSoft Anypoint Studio interface with a flow named "query-from-store". The flow consists of a "Listener" component followed by an "[Embedding] Generate from text" component. The configuration for the embedding operation is displayed on the right, under the "General" tab. The "Text" input field contains the expression "#[ payload.prompt ]". Other tabs include "Advanced", "Input/Output", and "Notes".

### Step 3

Drop [Store] Query operation right after the [Embedding] Generate from Text operation. properly set the store name.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, a flow diagram titled "Flow query-from-store" is displayed. The flow consists of three main components connected sequentially: a "Listener Listener" (represented by a blue icon with a globe), a "[Embedding] Generate from text" component (represented by a red icon with a brain), and a "[Store] Query [Store] Query" component (represented by a blue icon with a brain). Each component has a plus sign (+) below it, indicating they can be expanded or modified.

On the right, a configuration window for the "[Store] Query" component is open. The title bar says "MuleSoft Vectors Connector - Query [Store] Query". The window is divided into tabs: General (selected), Advanced, Input/Output, and Notes. The "General" tab contains the following configuration settings:

- Store name**: hands\_on\_vectors
- Text Segment and Embedding**: #[ payload ]
- Max results**: 5
- Min score**: 0,5

Below the "General" tab, there is a section titled "Filter" with a "Metadata Condition" input field containing the placeholder text: "E.g. file\_name = 'example.pdf' AND (file\_type = 'any' OR file\_name = 'txt')".

## Step 4

Start your application

Hit <http://localhost:8081/vectors/query> and check response.

```

POST http://localhost:8081/vectors/query
Body (raw)
{
  "prompt": "Shadows Over Carrington"
}
200 OK
{
  "minScore": 0.5,
  "question": "#\n    \"prompt\": \"Shadows Over Carrington\"\n",
  "sources": [
    {
      "score": 0.8903093752357178,
      "metadata": {
        "absolute_directory_path": "/Users/tbolis/VSCode/hands-on-vectors-app/src/main/resources",
        "ingestion_timestamp": "1749485615322",
        "file_name": "Shadows_Over_Carrington.pdf",
        "file_type": "any",
        "index": "0",
        "source_id": "9b1806e9-a0c4-4ebc-80c2-4164c38c2dc2",
        "ingestion_datetime": "2025-06-09T16:13:35.321914Z"
      },
      "embeddingId": "2f27d162-da09-43d7-8884-1978666661b3",
      "text": "Shadows Over Carrington\n Carrington was the kind of town where nothing ever happened--until it did. On a chilled autu"
    },
    {
      "score": 0.8719614778722655,
      "metadata": {
        "absolute_directory_path": "/Users/tbolis/VSCode/hands-on-vectors-app/src/main/resources",
        "ingestion_timestamp": "1749485615322",
        "file_name": "Shadows_Over_Carrington.pdf",
        "file_type": "any",
        "index": "2",
        "source_id": "9b1806e9-a0c4-4ebc-80c2-4164c38c2dc2",
        "ingestion_datetime": "2025-06-09T16:13:35.321914Z"
      }
    }
  ]
}

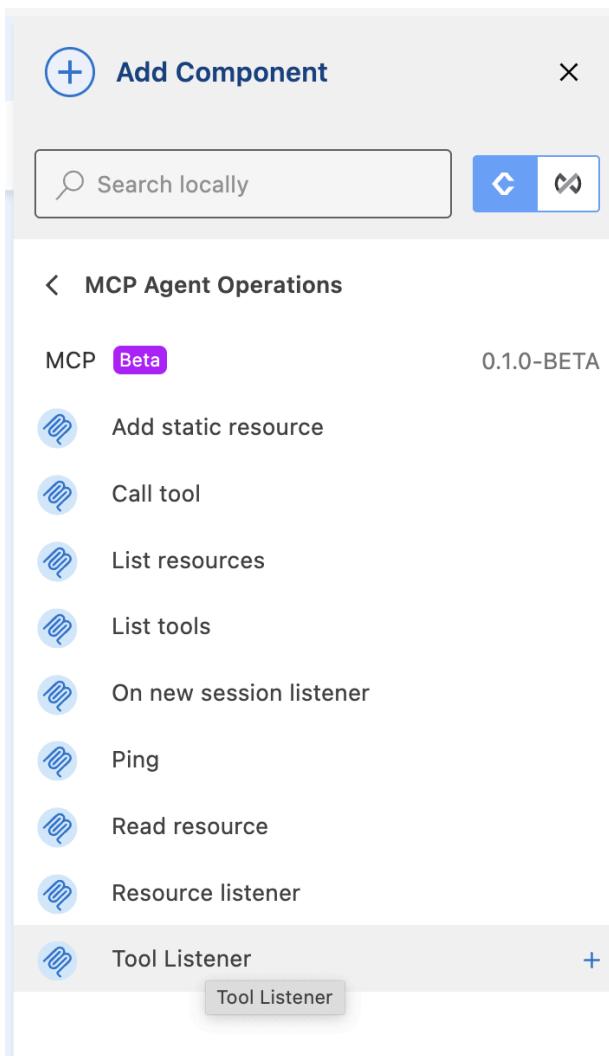
```

## Exercise 6 - Vector Store as MCP Tool

Start from the exercise 5 app.

### Step 1

Add MCP Connector to your app taking it from exchange.



## Step 2

Drop MCP Tool Listener and configure it as follow:

- Name: `search_docs`
- Description: `Search into a vector store which contains documentation about Shadows Over Carrington story.`
- Parameters schema:

```
{  
  "$schema": "http://json-schema.org/draft-07/schema#",  
  "type": "object",  
  "properties": {  
    "prompt": {
```

```
"type": "string",
"description": "The user prompt or question for which you need
information from Mule AI Chain documentation."
},
},
"required": [ "prompt" ],
"additionalProperties": false
}
```

- Responses:
- Text: `payload.^raw`
- Audience: ASSISTANT
- Priority: 1

The screenshot shows the Mule Studio interface with a flow named "query-as-mcp-tool" on the left. On the right, a configuration dialog for "Mcp - Tool listener" is open, specifically in the "General" tab.

**Mcp - Tool listener**

**\* Connection Config**

**Server**

**General**   Advanced   Input/Output   Notes

**\* Name**: search\_docs

**\* Description**: Search into a vector store which contains documentation about Shadows Over Carrington story.

**\* Parameters schema**

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "prompt": {
      "type": "string",
      "description": "The user prompt or question for which you need information from Mule AI Chain documentation."
    }
  },
  "required": [
    "prompt"
  ],
  "additionalProperties": false
}
```

**\* Responses**

**Responses**

**Responses**

**\* Text**: payload.^raw

**Audience**

**Audience**

ASSISTANT

+ Add   × Remove All

**Priority**: 1

+ Add   × Remove All

## Step 3

Create a new HTTP Listener configuration using port 8083

Http

## MCP-Listener-config

Configure a connection for this project. Updates will impact all components using this connection.

<p><b>General</b></p> <p>Advanced</p> <p>Notes</p>	<p>* Name MCP-Listener-config</p> <p>* Connection Listener</p> <p><b>Connection</b></p> <p>Protocol ⓘ HTTP (Default)</p> <p>* Host ⓘ 0.0.0.0</p> <p>* Port ⓘ 8083</p> <p>Read timeout ⓘ 30000</p>
--	---

## Step 4

Create a new MCP Server Configuration using Http Listener configuration created at previous step

Mcp - Tool listener

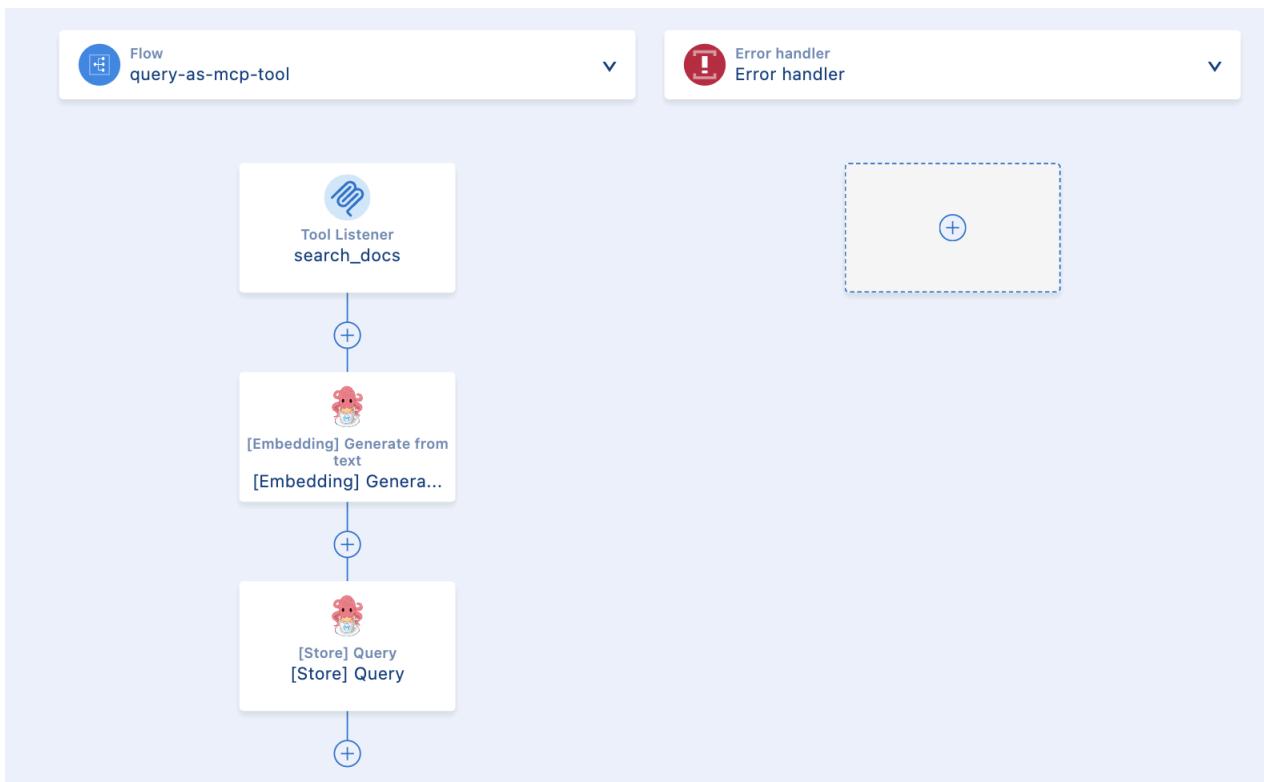
## Server

Configure a connection for this project. Updates will impact all components using this connection.

<b>General</b>	<p>* Name Server</p> <p>* Connection Sse server</p> <p><b>General</b></p> <p>* Connection Config MCP-Listener-config</p> <p>Sse endpoint path /sse</p> <p>Messages path /message</p> <p>* Server name mule-mcp-server</p> <p>* Server version 1.0.0</p> <p><b>Resource Capabilities</b></p> <p>List changed <input checked="" type="checkbox"/> List changed</p> <p>Subscribe <input checked="" type="checkbox"/> Subscribe</p> <p><b>Tools Capabilities</b></p> <p>List changed <input checked="" type="checkbox"/> List changed</p>
----------------	---

## Step 5

After MCP Tool Listener add same logic implemented in previous exercise for querying



## Step 6

Start your application



## Step 7

Test your MCP Server with URL <http://localhost:8083/sse> using one client as Witsy or the MCP Inspector.

- MCP Inspector available at <https://github.com/modelcontextprotocol/inspector>
- Witsy available at <https://witsyai.com/>

## MCP Inspector v0.14.0

Transport Type

SSE

URL

http://localhost:8083/sse

> Authentication

> Configuration

Connected

Resources Prompts Tools Ping Sampling Roots Auth

Tools

List Tools

Clear

search\_docs

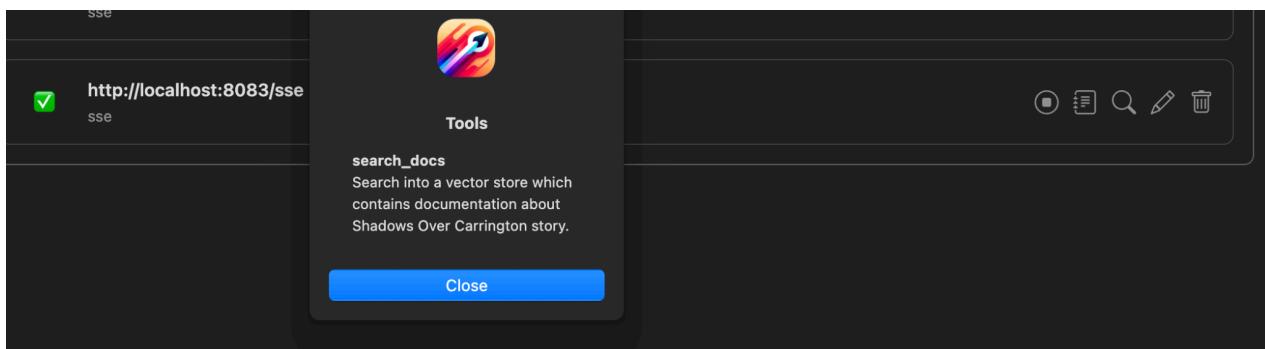
Search into a vector store which contains documentation about Shadows Over Carrington story.

prompt

What's the story about?

Tool Result: Success

```
{  
  minScore: 0.5  
  question: "What's the story about?"  
  sources: [  
    0: {  
      score: 0.8953082914926089  
      metadata: { ... } 7 items  
      embeddingId: "6b8fc492-7a60-43c7-bbaf-ad58c9290e90"  
      text: "To short."  
    }  
  ]  
}
```



# **WebCrawler Hands-on**

## **WebCrawler Connector Hands-on**

[Anypoint Studio \(WebCrawler\)](#)

[ACB \(WebCrawler\)](#)

# **ACB (WebCrawler)**

# Table of Contents

Exercise 1 - Get Static Page.....	1
Exercise 2 - Get Dynamic Page.....	1
Exercise 3 - Crawl Website.....	1
Exercise 4 - Crawl and ingest into Store.....	1
Exercise 5 - Page Crawl as MCP Tool.....	1

## Exercise 1 - Get Static Page

### Step 1

Create a new mule app `hands-on-webcrawler-app` using Studio or ACB.

or

Start from the `hands-on-vectors-app` if you plan to do Exercise 4

## Develop an Integration

Create and test a Mule application project that integrates existing services.

Project Name

Project Location

Browse

Create

Empty Project

*Create an integration project from scratch.*

Template or Example Project

*Start an integration with a template or example project from Anypoint Exchange.*

Mule Runtime ⓘ



Java Version ⓘ

CancelCreate Project

## Step 2

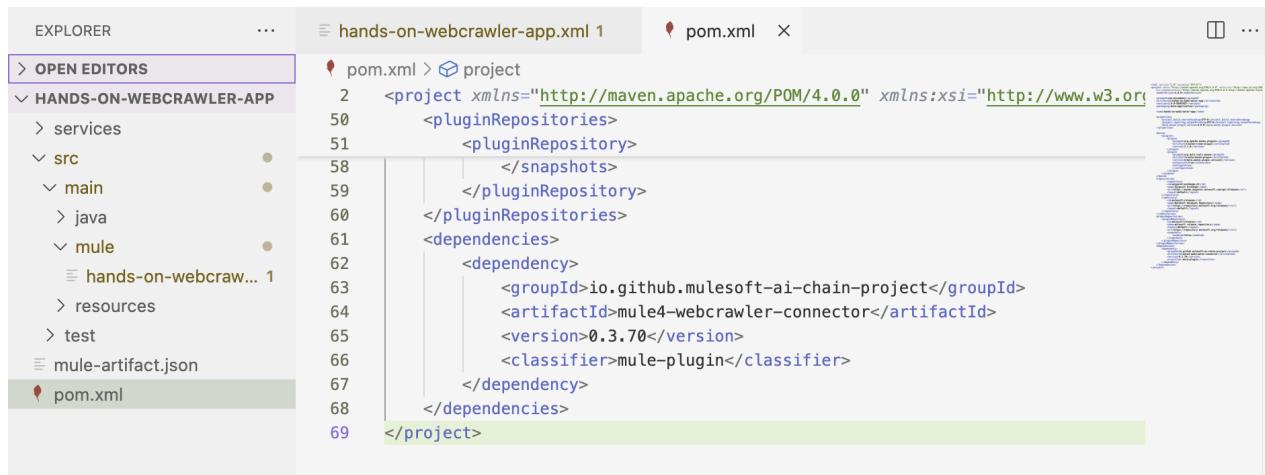
Add WebCrawler dependency to your pom.xml

```
<dependency>
<groupId>io.github.mulesoft-ai-chain-project</groupId>
<artifactId>mule4-webcrawler-connector</artifactId>
```

```

<version>0.3.70</version>
<classifier>mule-plugin</classifier>
</dependency>

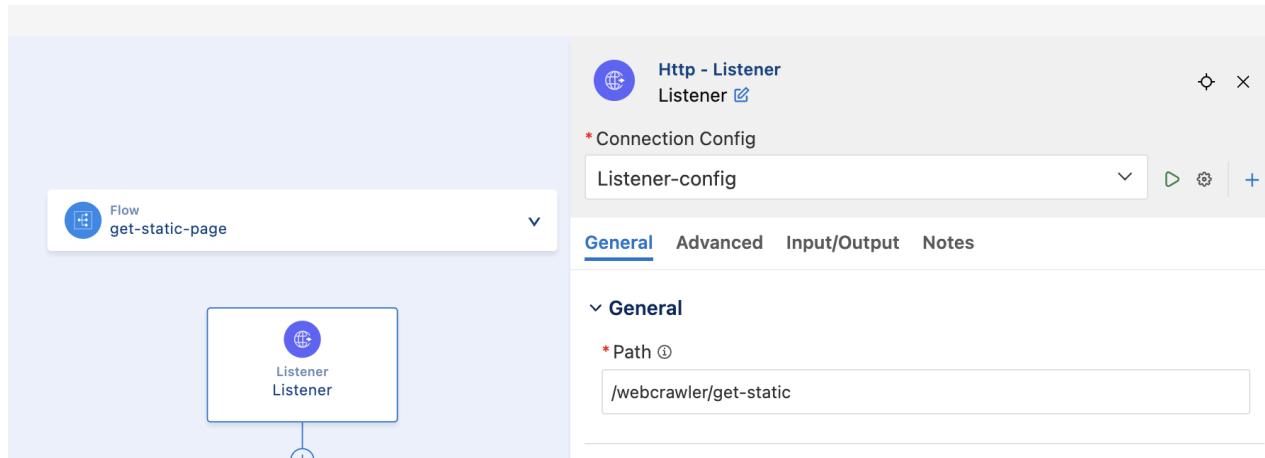
```



## Step 3

Drop http listener (/webcrawler/get-static) in a new flow, you'll use it to get page content in a static way. You'll use it to get page content using postman. It should accept a json payload like

```
{
"url": "https://mac-project.ai/",
"outputFormat": "MARKDOWN"
}
```



## Step 4

Create Listener Configuration

Http - Listener

## Listener-config

...

Configure a connection for this project. Updates will impact all components using this connection.

### General

Advanced

Notes

\* Name

Listener-config

\* Connection

Listener

### Connection

Protocol ⓘ

HTTP (Default)

\* Host ⓘ

0.0.0.0

\* Port ⓘ

8081

Read timeout ⓘ

30000

▷ Test Connection

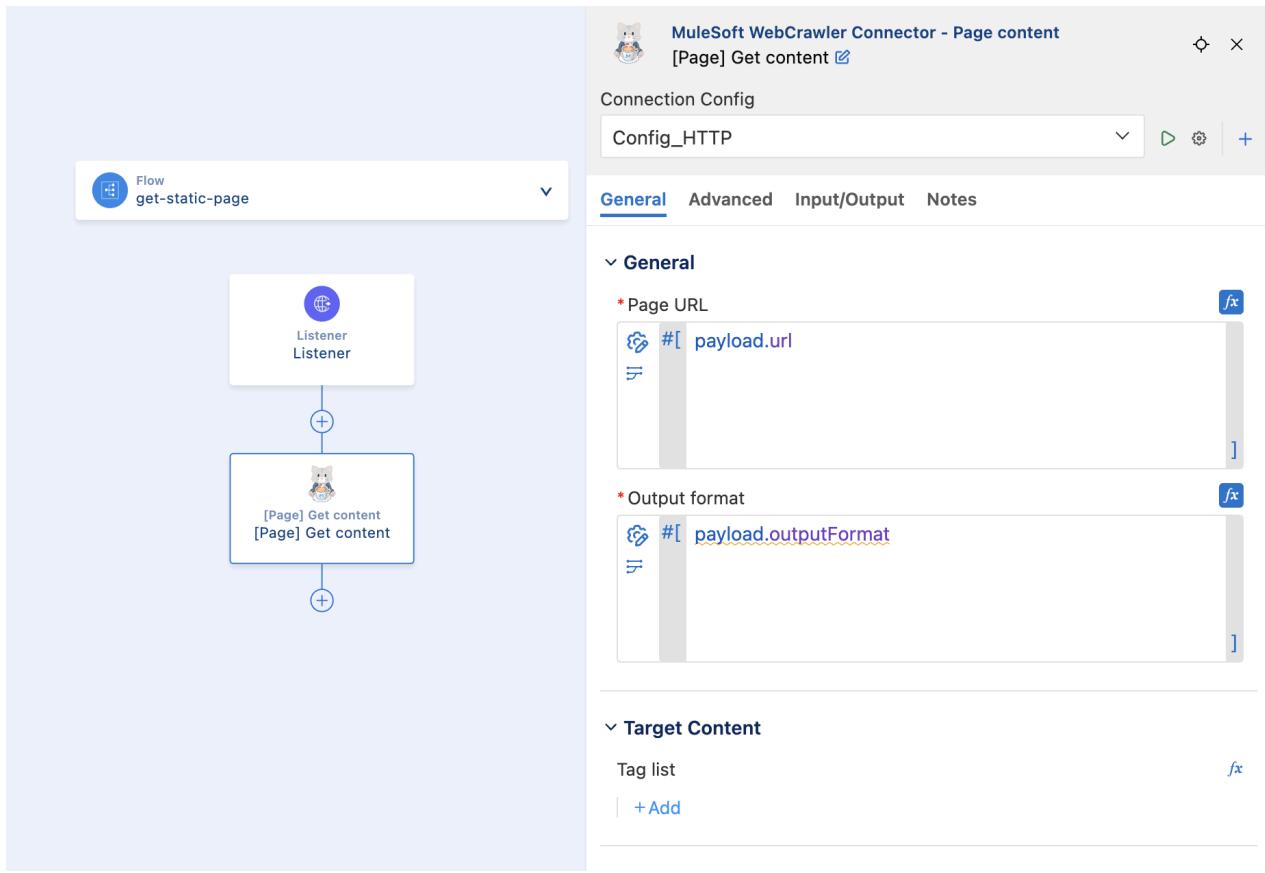
Cancel

Apply

## Step 5

Drop [Page] Get content operation after listener.

Set url as payload.url and output format as payload.outputFormat



## Step 6

Create WebCrawler Configuration (with HTTP Connection).

\* User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36

## Config\_HTTP

Configure a connection for this project. Updates will impact all components using this connection.

### General

Advanced

Notes

\* Name

Config\_HTTP

\* Connection

HTTP

#### General

User agent

fx

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4369.90 Safari/537.36

copy

Referrer

fx

www.google.com

Timeout

fx

10000

#### Crawler Options

▷ Test Connection

Cancel

Apply

## Step 7

Start your application

Hit <http://localhost:8081/webcrawler/get-static> and check response.

Request payload:

```
{  
  "url": "https://mac-project.ai/",  
  "outputFormat": "MARKDOWN"  
}
```

```

POST http://localhost:8081/webcrawler/get-static
{
  "url": "https://mac-project.ai",
  "outputFormat": "MARKDOWN"
}
200 OK
{
  "title": "The MuleSoft AI Chain (MAC) Project",
  "url": "https://mac-project.ai",
  "content": "\n\n[](/)[Docs] (/docs)[About] (/about)\u2026"
}

```

## Exercise 2 - Get Dynamic Page

Start from the exercise 1 app.

### Step 1

Copy the flow from previous exercise.

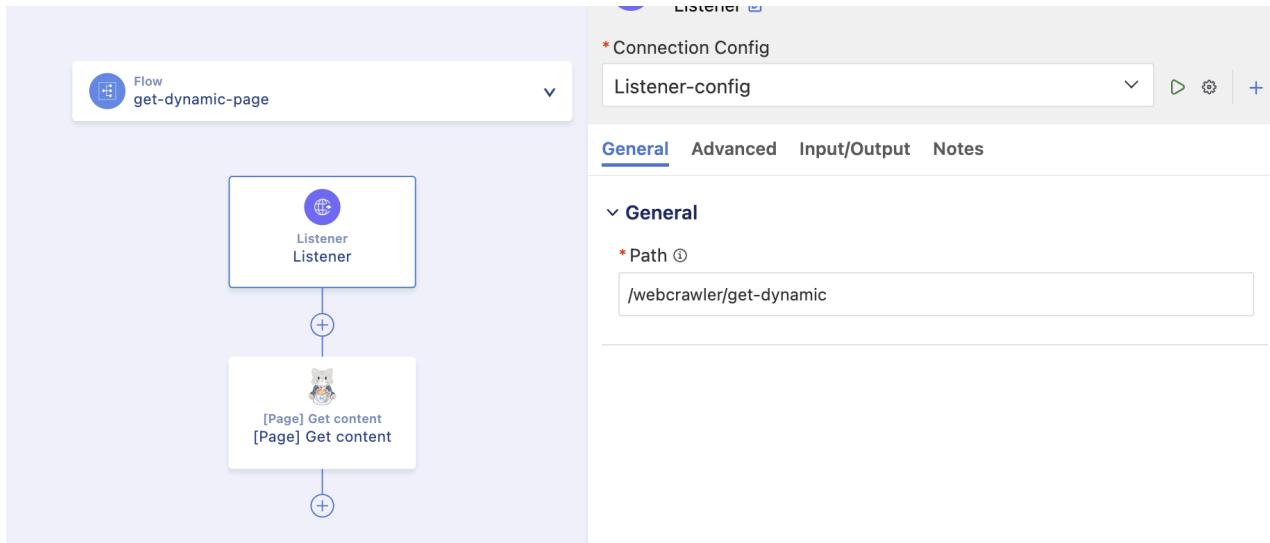
```

<mule xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="http://www.mulesoft.org/schema/mule/documentation" testConnection="true">
  <http:listener-config name="Listener-config">
    <http:listener-connection host="0.0.0.0" port="8081"/>
  </http:listener-config>
  <ms-webcrawler:config name="Config_HTTP">
    <ms-webcrawler:http-connection referrer="www.google.com" userAgent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.102 Safari/537.36" />
  </ms-webcrawler:config>
  <flow name="get-static-page">
    <http:listener config-ref="Listener-config" doc:id="mtcdxf" doc:name="HTTP Listener" doc:type="HTTP Listener"/>
    <ms-webcrawler:page-content doc:id="gabfat" doc:name="Get content" doc:type="Page Content" referrer="www.google.com" userAgent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.102 Safari/537.36" />
  </flow>
  <flow name="get-dynamic-page">
    <http:listener config-ref="Listener-config" doc:id="erwdgr" doc:name="HTTP Listener" doc:type="HTTP Listener"/>
    <ms-webcrawler:page-content doc:id="werssr" doc:name="Get content" doc:type="Page Content" referrer="www.google.com" userAgent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.102 Safari/537.36" />
  </flow>
</mule>

```

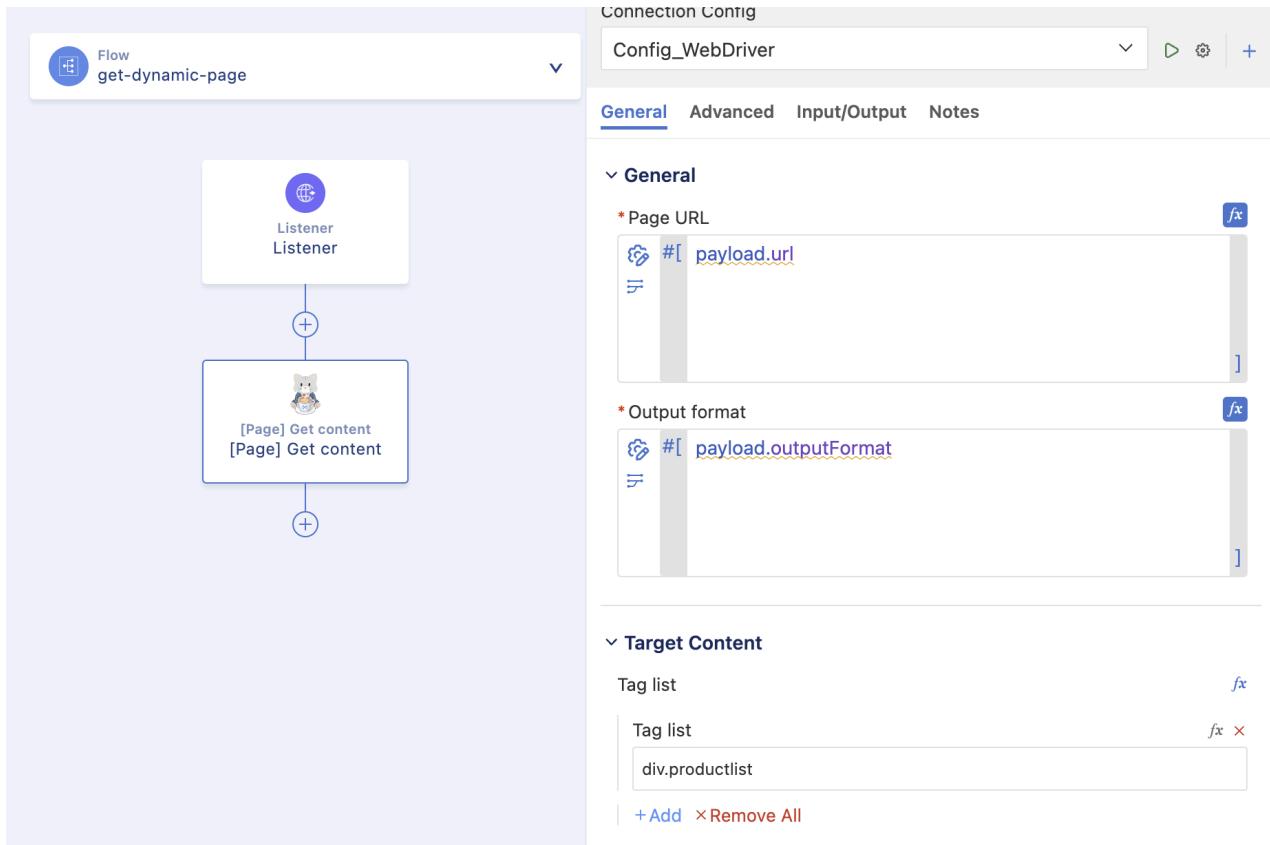
### Step 2

Update listener with path /webcrawler/get-dynamic



### Step 3

Update [Page] Get content operation and set a new Tag `div.productlist`



## **Step 4**

Create WebCrawler Configuration (with WebDriver Connection). **User Agent drop down only working with Anypoint Studio**

\* User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36

\* Referrer <https://www.google.com>

\* Set Wait on page load (millisec) to 120000.

\* Set Wait for xPath to //div[@class="productlist"]

MuleSoft WebCrawler Connector - Page content

## Config\_WebDriver

Configure a connection for this project. Updates will impact all components using this connection.

<ul style="list-style-type: none"> <li><b>General</b></li> <li>Advanced</li> <li>Notes</li> </ul>	<p>* Name Config_WebDriver</p> <p>* Connection WebDriver</p> <p><b>General</b></p> <p>User agent Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4369.90 Safari/537.36</p> <p>Referrer https://www.google.com</p> <p><b>Crawler Options</b></p> <p>Delay between pages (millisecs) 0</p> <p>Enforce robots.txt <input type="checkbox"/> Enforce robots.txt</p> <p><b>Page Load Options (WebDriver)</b></p> <p>Wait on page load (millisecs) 120000</p> <p>Wait for XPath //div[@class="productlist"]</p> <p>Extract Shadow DOM <input type="checkbox"/> Extract Shadow DOM</p> <p>Shadow Host(s) XPath E.g. //results</p>
<input type="button" value="▷ Test Connection"/> <input type="button" value="Cancel"/> <input type="button" value="Apply"/>	

## Step 5

Start your application

Hit <http://localhost:8081/webcrawler/get-dynamic> and check response.

Request payload:

```
{
"url": "https://www.brenntag.com/en-us/products",
"outputFormat": "MARKDOWN"
```

}

MAC SME Development / get-dynamic-page

POST <http://localhost:8081/webcrawler/get-dynamic>

Params Authorization Headers (8) Body Scripts Settings

Body (raw JSON)

```
1 {  
2   "url": "https://www.brenntag.com/en-us/products/",  
3   "outputFormat": "MARKDOWN"  
4 }
```

200 OK 15.35 s 56.97 KB Save Response

Body Cookies Headers (3) Test Results

{ } JSON Preview Visualize

```
1 {  
2   "title": "Product overview | Brenntag",  
3   "url": "https://www.brenntag.com/en-us/products/",  
4   "content": "\n\n # A B C D E F G H I J K L M N O P Q R S T U V W X Y Z\n\n * [ 1,4-Butanediol\n\n Formula: C4H10O2 | CAS: 110-63-4 ](/en-us/p:  
5 }
```

1,4-Butanediol  
Formula: C4H10O2 | CAS: 110-63-4

2272 Sericite LQ-15  
CAS: -

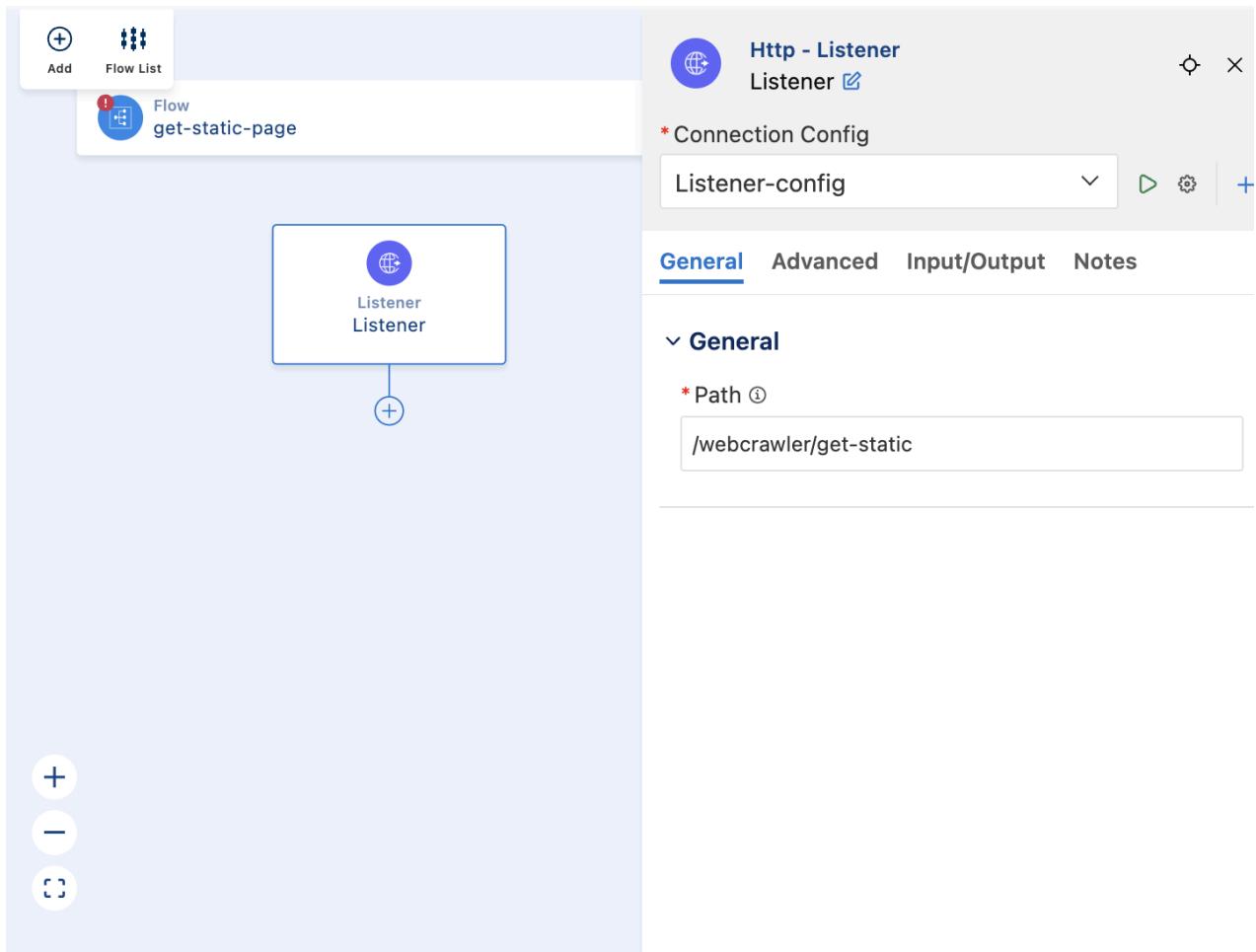
Postbot Runner Start Proxy Cookies Vault

## Exercise 3 - Crawl Website

Start from the exercise 2 app.

### Step 1

Drop http listener (with path `/webcrawler/crawl`) in a new flow, you'll use it to download the site content.



## Step 2

Drop [Crawl] Website (Full Scan) operation after listener. Set: url, output format, download location, mac depth, etc...

Use the configuration we created for exercise 1.

- \* url: <https://mac-project.ai>
- \* output format: MARKDOWN
- \* download location: /Users/<your-user>/Desktop/mac-project.ai
- \* depth: 1
- \* Restrict crawl under URL:false
- \* Download images: true

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, there is a flow diagram titled "Flow crawl". The flow starts with a "Listener" component, followed by a connector component labeled "[Crawl] Website (Full Scan) [Crawl] Website (F...)".

On the right, a configuration window is open for the "MuleSoft WebCrawler Connector - Crawl website full scan" configuration. The window has tabs for "General", "Advanced", "Input/Output", and "Notes". The "General" tab is selected.

**General** tab settings:

- \* Website URL: https://mac-project.ai/
- \* Output format: MARKDOWN
- \* Download location: /Users/tbolis/Desktop/mac-project.ai

**Target Pages** section:

- \* Maximum depth: 1
- Restrict crawl under URL:  (unchecked)
- Regex URLs filter logic: -- Empty --
- Regex URLs List:

**Target Content** section:

- Tag list:
- Retrieve meta tags:  (unchecked)
- Download images:  (checked)
- Max image number for each page:

## Step 3

Start your application  
Hit <http://localhost:8081/webcrawler/crawl>.

HTTP MAC SME Development / [get-dynamic-page](#) Copy

Save Share

POST http://localhost:8081/webcrawler/crawl

Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Body (8) x-www-form-urlencoded raw binary GraphQL JSON

```
1 {}
```

Body Cookies Headers (3) Test Results

200 OK 29.63 s 1.09 KB Save Response

{ } JSON Preview Visualize

```

1 {
2   "url": "https://mac-project.ai",
3   "filename": "TheMuleSoftAIChain(MAC)Project_20250611090135943.json",
4   "children": [
5     {
6       "url": "https://www.linkedin.com/groups/13047000/",
7       "filename": "LinkedInLogin,Signin_LinkedIn_20250611090136687.json"
8     },
9     {
10       "url": "https://mac-project.ai/docs",
11       "filename": "Introduction_20250611090138761.json"
12     },
13     {
14       "url": "https://mac-project.ai/",
15       "filename": "TheMuleSoftAIChain(MAC)Project_20250611090139611.json"
16     },
17     {
18       "url": "https://mac-project.ai/docs/mulechain-ai/getting-started",
19       "filename": "GettingStarted_20250611090141187.json"
20     },
21     {
22       "url": "https://github.com/MuleSoft-AI-Chain-Project",
23     }
24   ]
25 }
```

## Step 4

check download location.

mac-project.ai		View	Group	Share	Add Tags	Action	Search
Name		Date Modified		Size		Kind	
About_20250611090203349.json		Today at 09:02		25 KB		JSON	
Contribute_20250611090156416.json		Today at 09:01		9 KB		JSON	
GettingStarted_20250611090141187.json		Today at 09:01		17 KB		JSON	
images		Today at 09:02		--		Folder	
44559.jpg		Today at 09:01		2 KB		JPEG image	
13334073.jpg		Today at 09:01		7 KB		JPEG image	
16238095.jpg		Today at 09:01		2 KB		JPEG image	
19755082.jpg		Today at 09:01		2 KB		JPEG image	
29606687.jpg		Today at 09:01		11 KB		JPEG image	
33336374.jpg		Today at 09:01		2 KB		JPEG image	
41447877.jpg		Today at 09:01		2 KB		JPEG image	
86777111.jpg		Today at 09:01		9 KB		JPEG image	
87764828.jpg		Today at 09:01		2 KB		JPEG image	
89086831.jpg		Today at 09:01		2 KB		JPEG image	
95849087.jpg		Today at 09:01		2 KB		JPEG image	
142403274.jpg		Today at 09:01		2 KB		JPEG image	

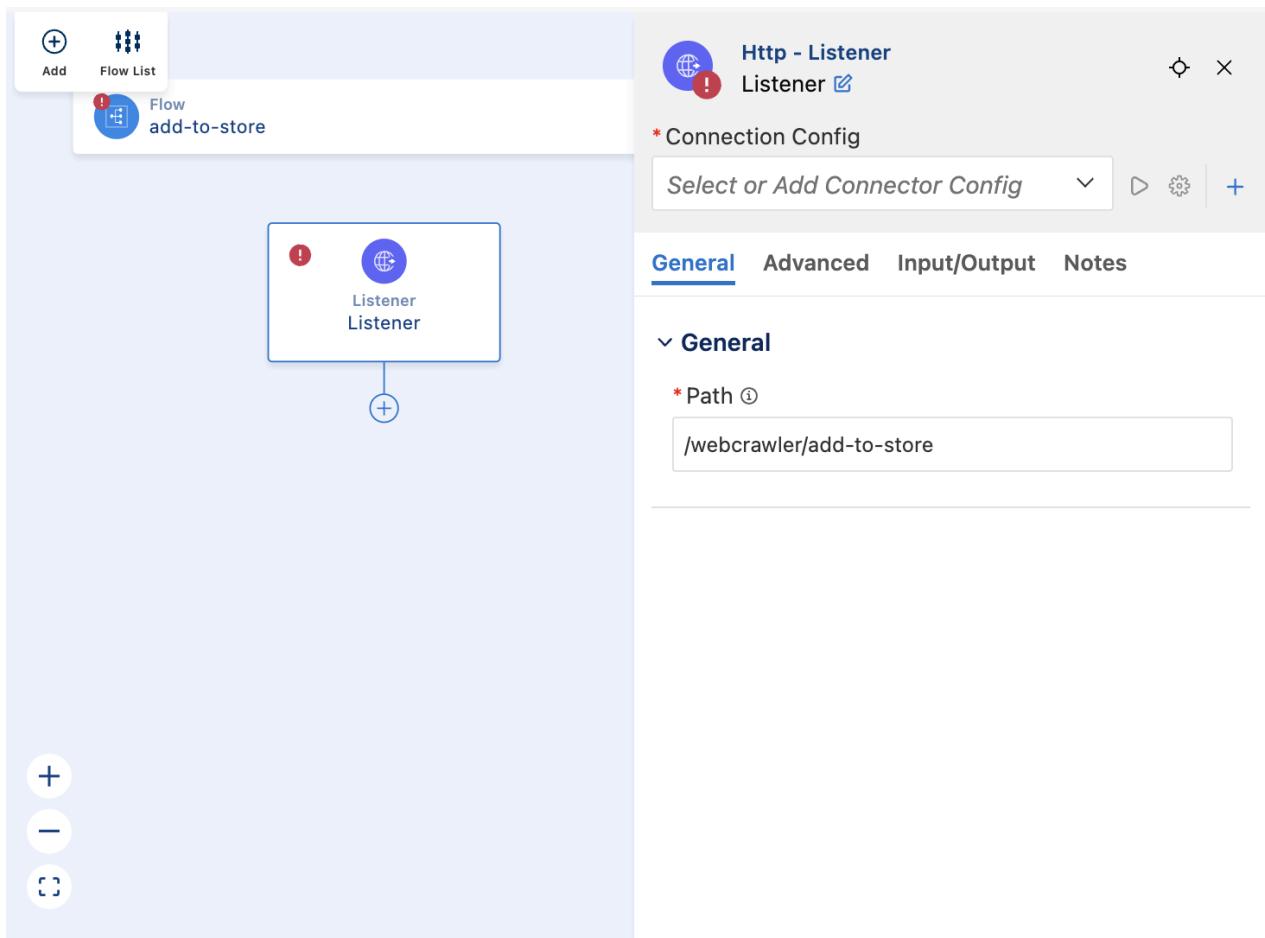
# Exercise 4 - Crawl and ingest into Store

## Prerequisites

- [Setup SDO for Einstein Model APIs](#)
- [Setup PostgreSQL from SE Platform](#)
- [Vectors Hands-on](#)

## Step 1

Drop http listener (/webcrawler/add-to-store) in a new flow, you'll use it load site content inside vector store.



## Step 2

Drop [Crawl] Website (Streaming)

- \* url: <https://mac-project.ai/>
- \* output format: MARKDOWN
- \* download location: /Users/<your-user>/Desktop/mac-project.ai
- \* depth: 1
- \* Restrict crawl under URL: true

MuleSoft WebCrawler Connector - Crawl website streaming  
[Crawl] Website (...)

Connection Config  
Select or Add Connector Config

**General** Advanced Input/Output Notes

**General**

\* Website URL  [fx](#)

\* Output format  [fx](#)

**Target Pages**

\* Maximum depth  [fx](#)

Restrict crawl under URL  [fx](#)

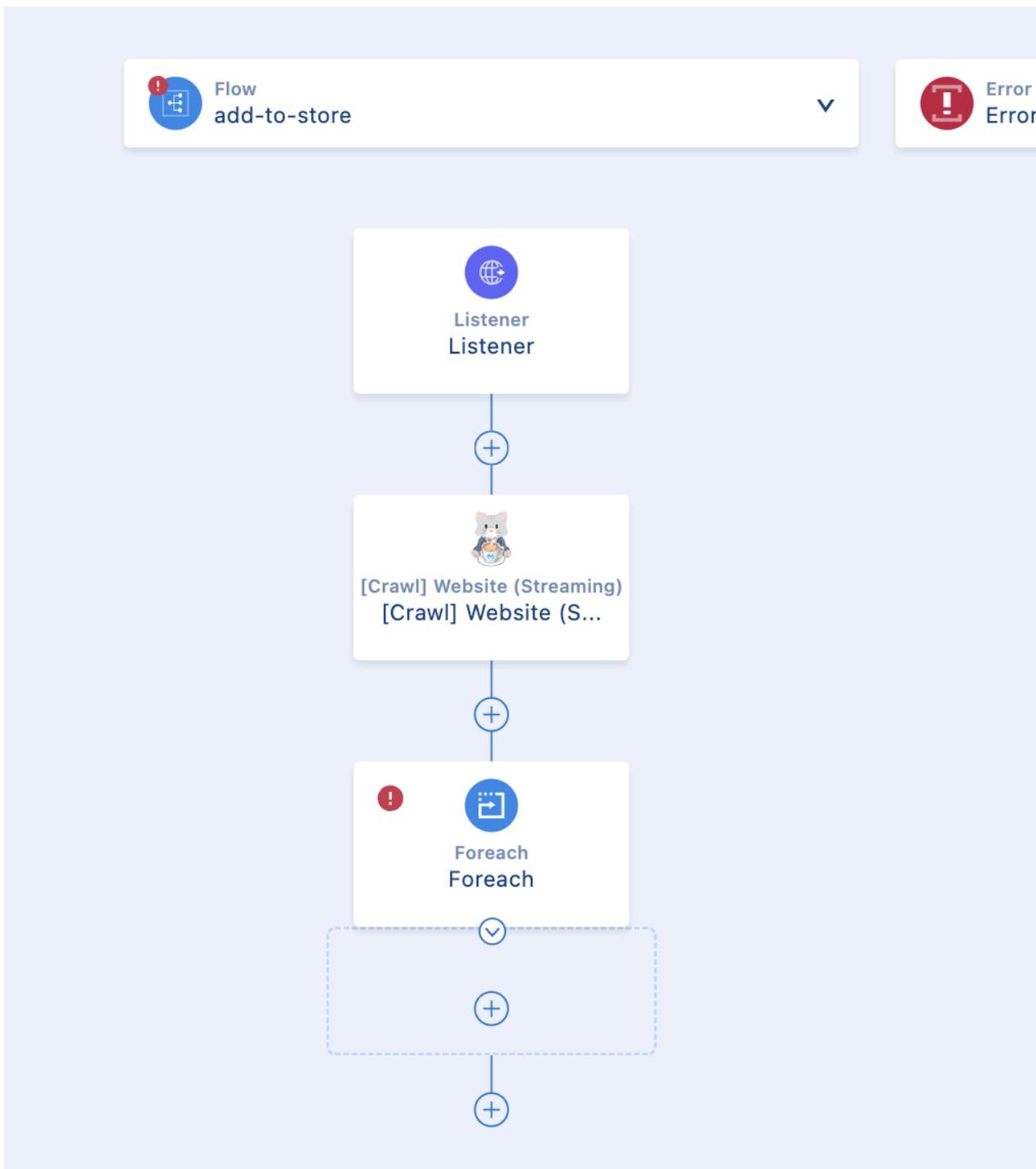
Regex URLs filter logic  [fx](#)

+ Add

Regex URLs List [fx](#)

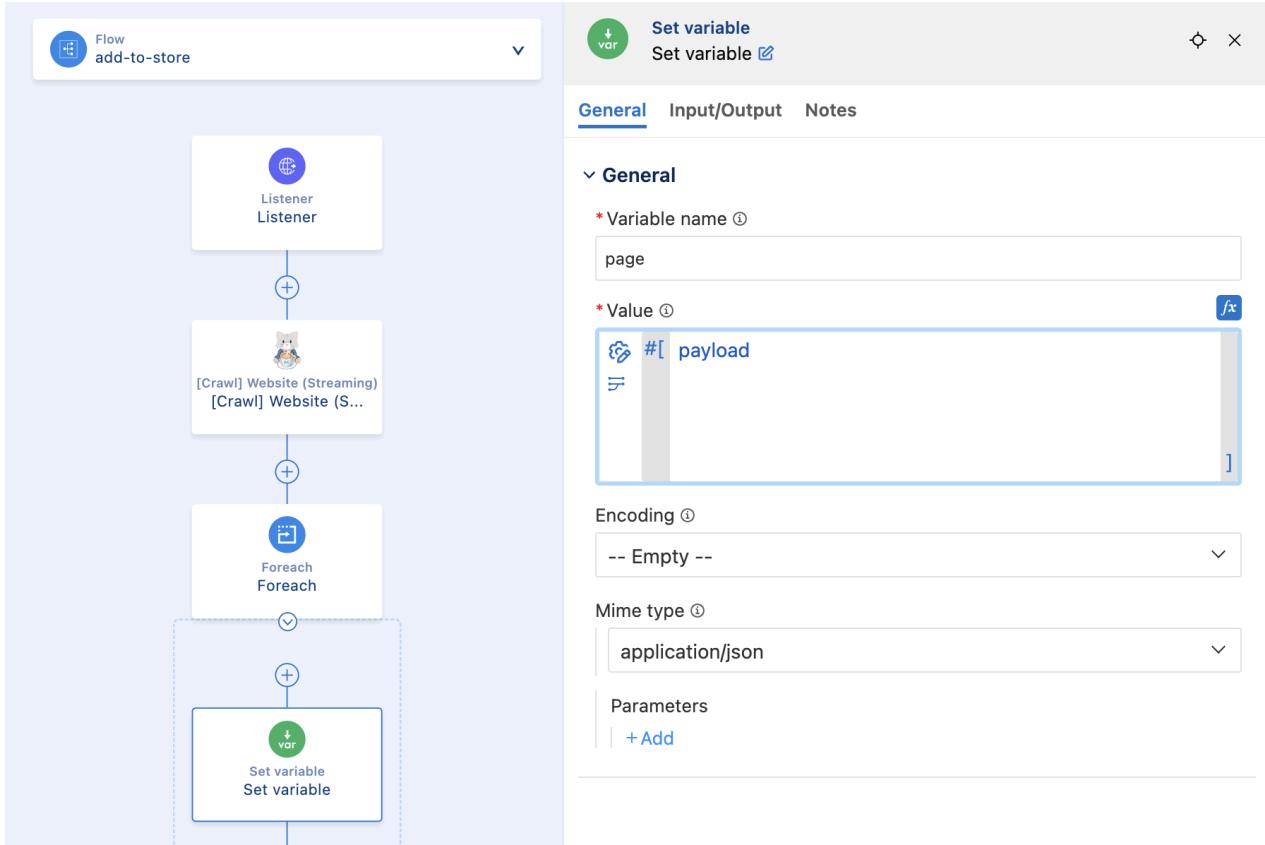
## Step 3

Drop For Each



## Step 4

Set payload into a `page` variable



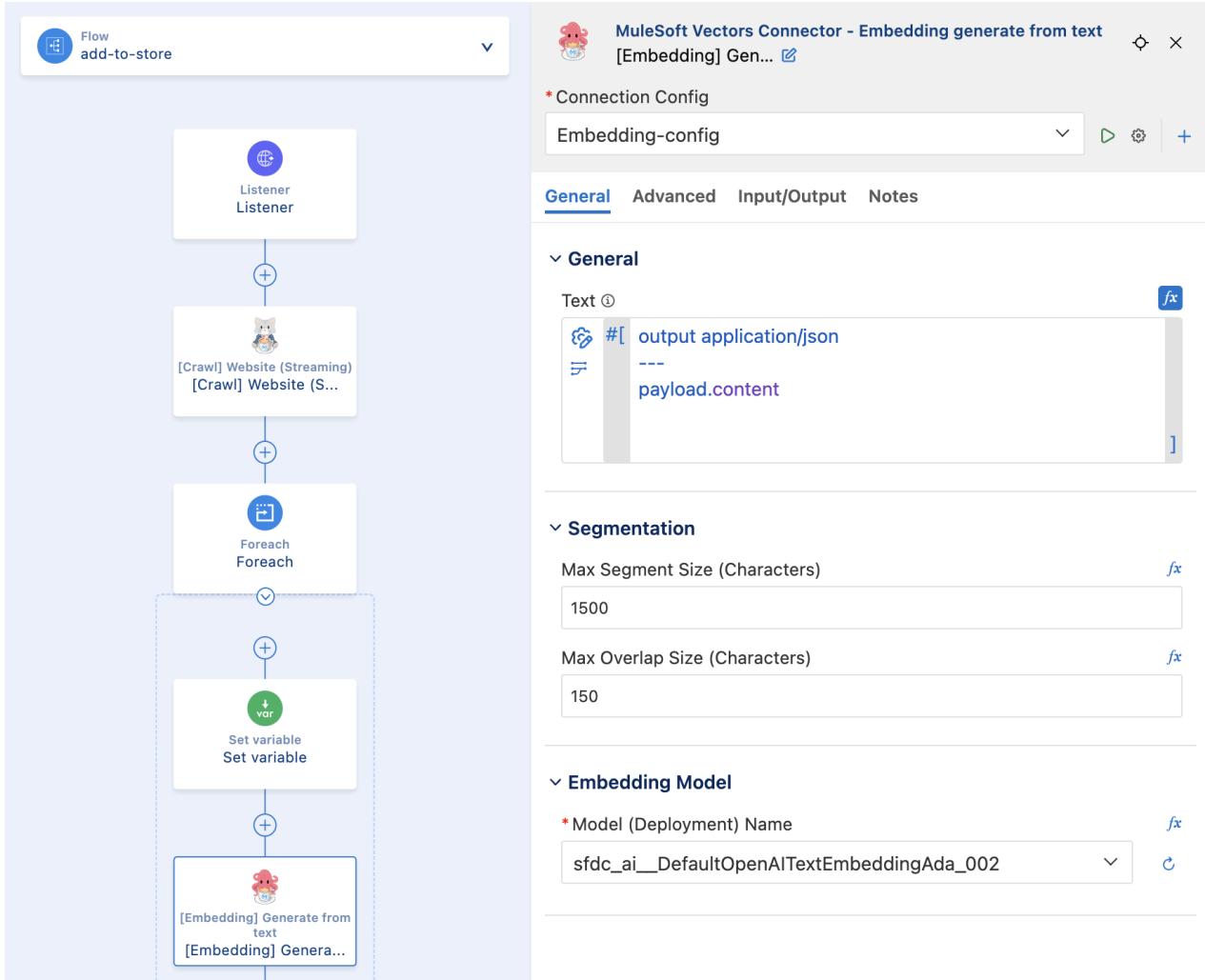
## Step 5

Drop [Embedding] Generate from Text (inside for each)

*Use configuration from vectors exercise*

Set payload to:

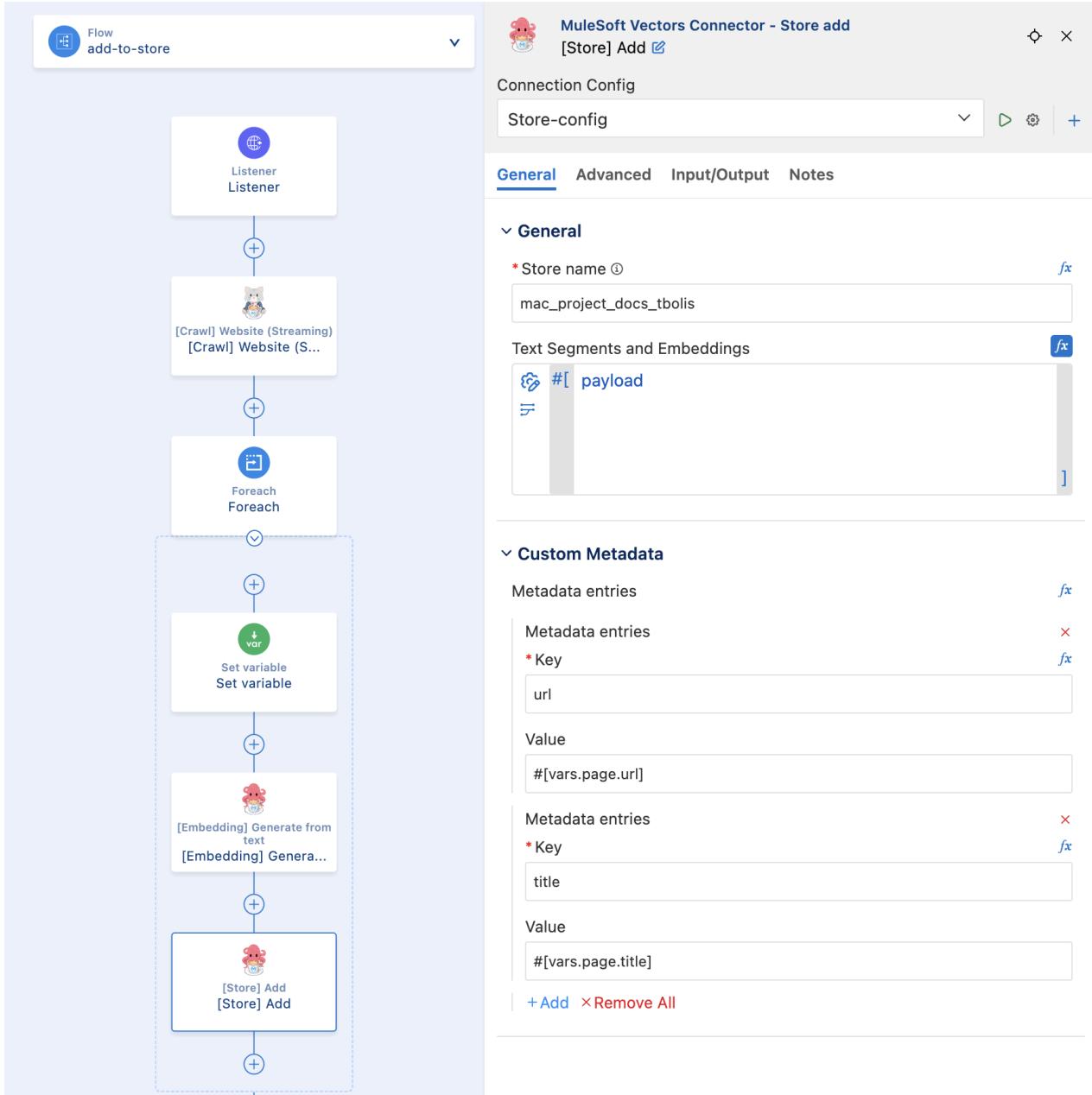
```
output application/json
---
payload.content
```



## Step 6

Drop [Store] Add (inside for each).

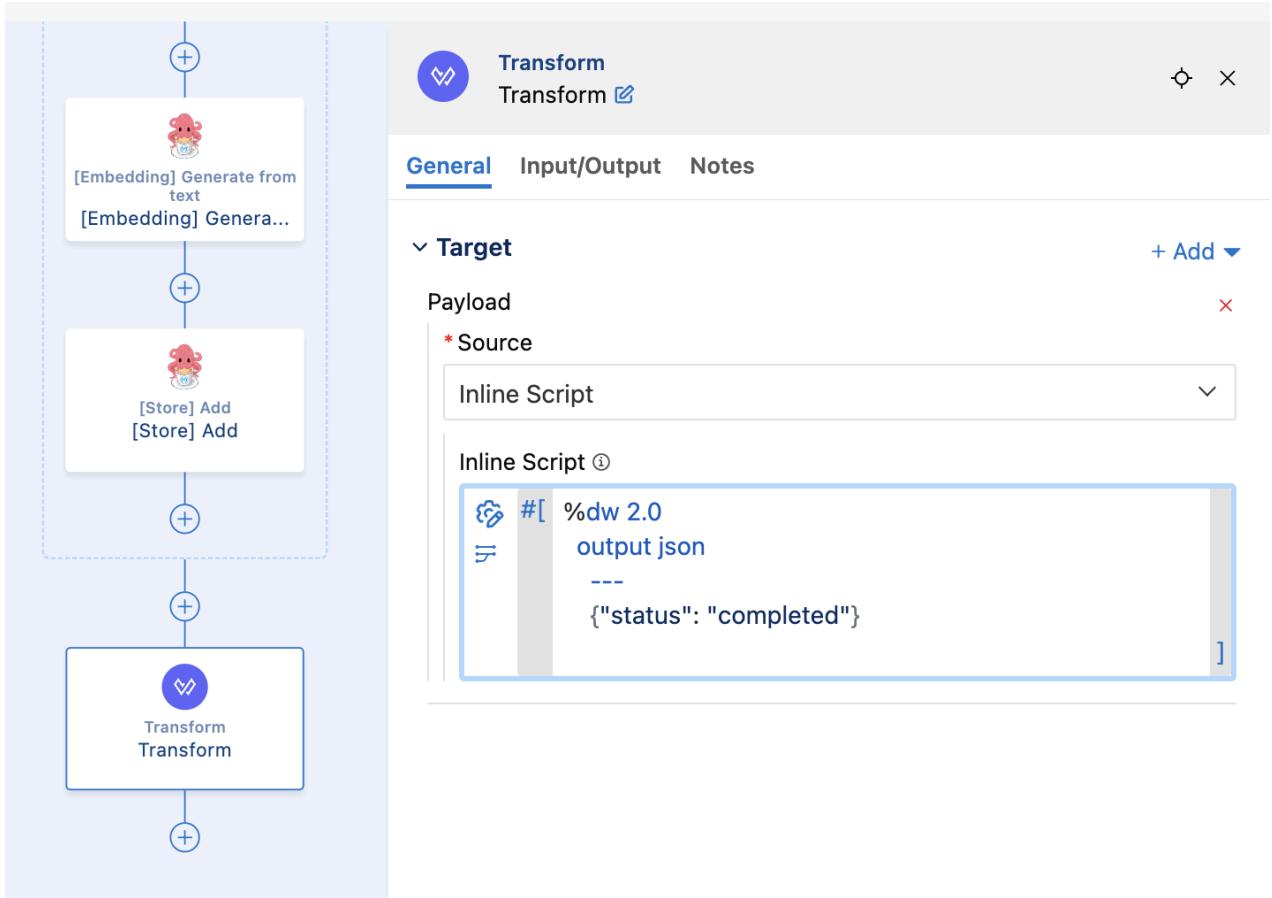
\* storeName: `mac_project_docs_<your-user>`



## Step 7

Drop a Transform with:

```
%dw 2.0
output json
---
{"status": "completed"}
```



## Step 8

Start your application

Hit <http://localhost:8081/webcrawler/add-to-store> and check the response.

The screenshot shows a Postman collection named "MAC SME Development / crawl". A POST request is made to "http://localhost:8081/webcrawler/add-to-store". The "Body" tab is selected, showing a JSON payload:

```
{}
{
  "status": "completed"
}
```

The response received is a 200 OK status with the following details:

- Time: 25.71 s
- Size: 150 B
- Save Response

## **Step 9**

## Check rows inserted into DB

The screenshot shows the DBBeaver interface with a complex database structure. The left sidebar lists databases, tables, and views. The main area displays a table named 'mac\_project\_docs\_tbolis' with columns like 'embedding\_id', 'text', and 'embedding'. A search bar at the top right contains the query 'Enter a SQL expression to filter results (use Ctrl+Space)'. The status bar at the bottom indicates 'Database Navig... Projects ... public hands\_on\_vectors ... mac\_project\_docs\_tbolis ... Properties Data Diagram'.

## Exercise 5 - Page Crawl as MCP Tool

## **Step 1**

Add MCP Connector to your app taking it from exchange.

The screenshot shows a modal window titled "Add Component" with a search bar and two buttons. Below the search bar, there's a breadcrumb navigation with "MCP Agent Operations". The main list contains several items, each with a blue circular icon and a name:

- MCP Beta 0.1.0-BETA
- Add static resource
- Call tool
- List resources
- List tools
- On new session listener
- Ping
- Read resource
- Resource listener
- Tool Listener +

A tooltip "Tool Listener" is visible near the bottom left of the list.

## Step 2

Drop MCP Tool Listener and configure it as follow:

- Name: `get_page`
- Description: `Retrieve page content from url.`
- Parameters schema:

```
{  
"$schema": "http://json-schema.org/draft-07/schema#",  
"type": "object",
```

```
"properties": {  
  "url": {  
    "type": "string",  
    "description": "The url to get content from."  
  }  
},  
"required": [ "url" ],  
"additionalProperties": false  
}
```

- Responses:

- Text: `payload.^raw`
- Audience: ASSISTANT
- Priority: 1

The screenshot shows the MCP (Multi-Protocol Client) interface. On the left, a flow named "Flow get-page-as-mcp-tool" is displayed, containing a single step: "Tool Listener get\_page". On the right, a detailed configuration window for "Mcp - Tool listener" is open, specifically for the "get\_page" step.

**Mcp - Tool listener**

**\* Connection Config**

Server

**General** Advanced Input/Output Notes

**get\_page**

**\* Description**

Retrieve page content from url.

**\* Parameters schema**

```
{ "$schema": "http://json-schema.org/draft-07/schema#", "type": "object", "properties": { "url": { "type": "string", "description": "The url to get content from." } }, "required": ["prompt"], "additionalProperties": false}
```

**\* Responses**

**Responses**

Text tool response content

\* Text

```
#[ payload.^raw ]
```

**Audience**

Audience

ASSISTANT

+ Add × Remove All

**Priority**

1

+ Add × Remove All

### Step 3

Create a new HTTP Listener configuration using port 8083

Http

## MCP-Listener-config

Configure a connection for this project. Updates will impact all components using this connection.

<p><b>General</b></p> <p>Advanced</p> <p>Notes</p>	<p>* Name MCP-Listener-config</p> <p>* Connection Listener</p> <p><b>Connection</b></p> <p>Protocol ⓘ HTTP (Default)</p> <p>* Host ⓘ 0.0.0.0</p> <p>* Port ⓘ 8083</p> <p>Read timeout ⓘ 30000</p>
--	---

## Step 4

Create a new MCP Server Configuration using Http Listener configuration created at previous step

Mcp - Tool listener

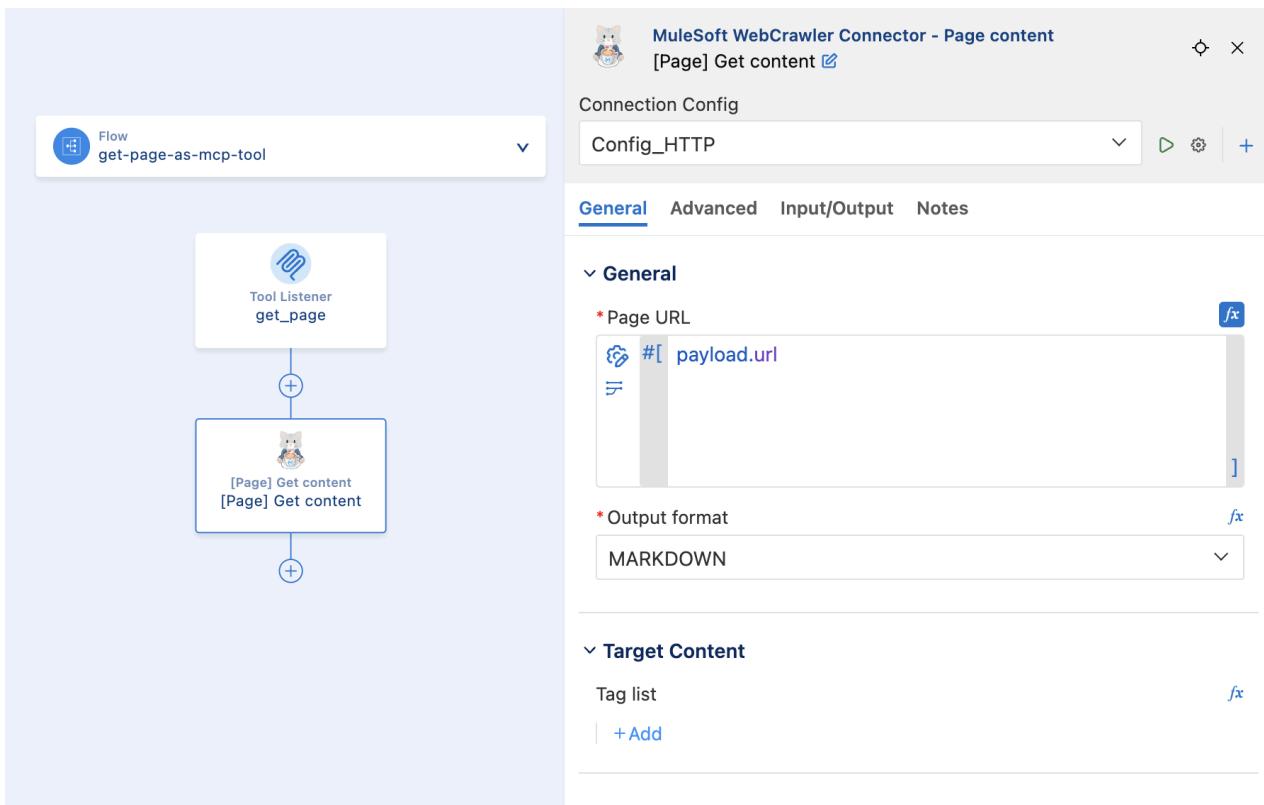
## Server

Configure a connection for this project. Updates will impact all components using this connection.

<b>General</b>	<p>* Name Server</p> <p>* Connection Sse server</p> <p><b>General</b></p> <p>* Connection Config MCP-Listener-config</p> <p>Sse endpoint path /sse</p> <p>Messages path /message</p> <p>* Server name mule-mcp-server</p> <p>* Server version 1.0.0</p> <p><b>Resource Capabilities</b></p> <p>List changed <input checked="" type="checkbox"/> List changed</p> <p>Subscribe <input checked="" type="checkbox"/> Subscribe</p> <p><b>Tools Capabilities</b></p> <p>List changed <input checked="" type="checkbox"/> List changed</p>
----------------	---

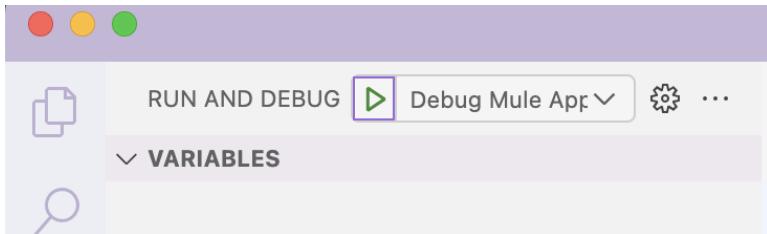
## Step 5

After MCP Tool Listener add same logic implemented in exercise 1 to get page content in a static way.



## Step 6

Start your application



## Step 7

Test your MCP Server with URL <http://localhost:8083/sse> using one client as Witsy or the MCP Inspector.

- MCP Inspector available at <https://github.com/modelcontextprotocol/inspector>
- Witsy available at <https://witsyai.com/>

**MCP Inspector v0.14.0**

Transport Type  
SSE

URL  
<http://localhost:8083/sse>

> Authentication

[Server Entry](#) [Servers File](#)

> Configuration

Connected

Resources Prompts Tools Ping Sampling Roots Auth

Tools

List Tools

Clear

get\_page  
Retrieve page content from url.

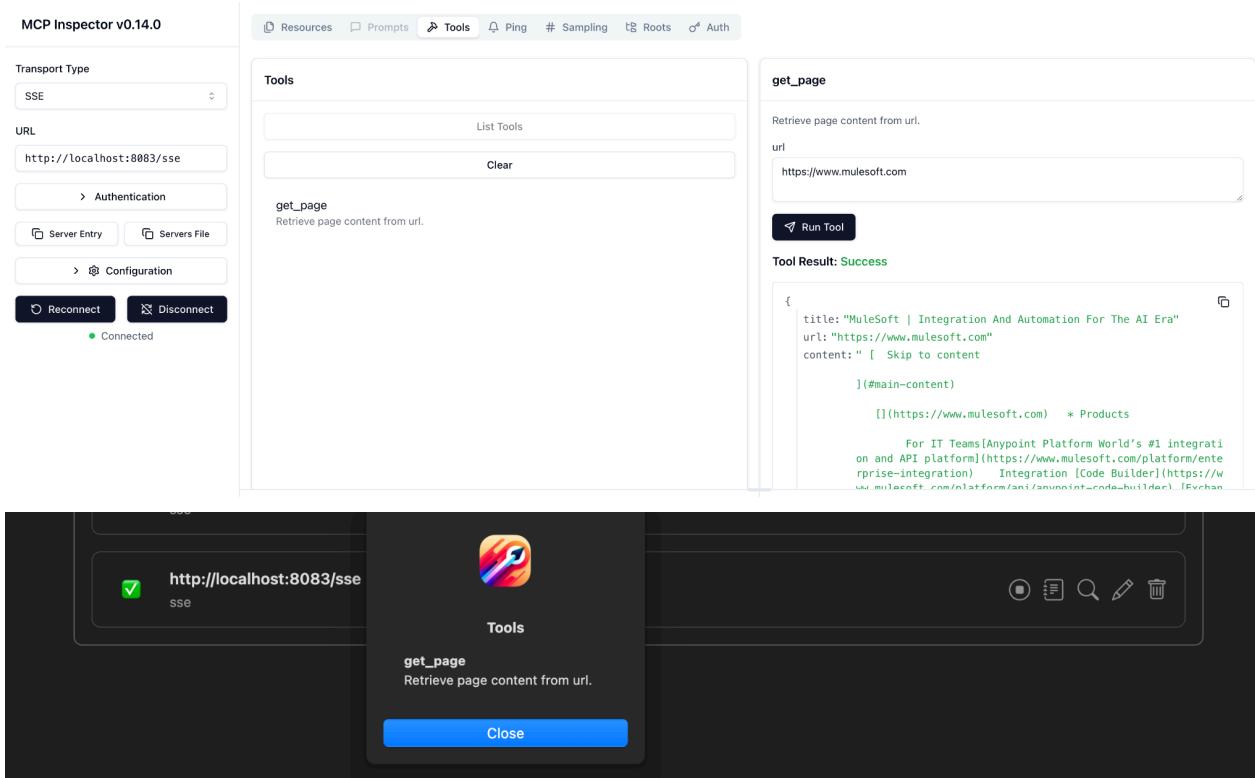
get\_page

Retrieve page content from url.

Run Tool

Tool Result: Success

```
{  
  title: "MuleSoft | Integration And Automation For The AI Era"  
  url: "https://www.mulesoft.com"  
  content: " [ Skip to content ](#main-content)  
  [](https://www.mulesoft.com) * Products  
  For IT Teams [Anypoint Platform World's #1 integration and API platform](https://www.mulesoft.com/platform/enterprise-integration) Integration [Code Builder](https://www.mulesoft.com/platform/api/anypoint-code-builder). [Exchange
```



http://localhost:8083/sse  
sse

Tools

get\_page  
Retrieve page content from url.

Close

Close

Search

Edit

Delete

# **Pre-work**

# Setup PostgreSQL from SE Platform

```
#heroku-hands-on configurations
```

```
herokuhandson:
```

```
  host: cloud-services.demos.mulesoft.com
  port: '30389'
  username: postgres
  databaseName: sampledb
  password: TVK4pah_tze3dqe5qvx
  rootPassword: ''
```

## Step 1

On SE Platform create a new Cloud Service.

Select `PostgreSQL17 with vector similarity...`

Select a service type

X

Q PostgreSQL X

Total: 2

Service	Description ^	Version	Provider
	PostgreSQL DB	11.9.0.	Field EKS
	PostgreSQL17 with vector similarity	17.2.0	Field EKS

Next →

## Step 2

Enter required info

Creating an instance of *PostgreSQL 17 with vector similarity search* X

**Need help creating the instance?** [Go to the documentation](#)

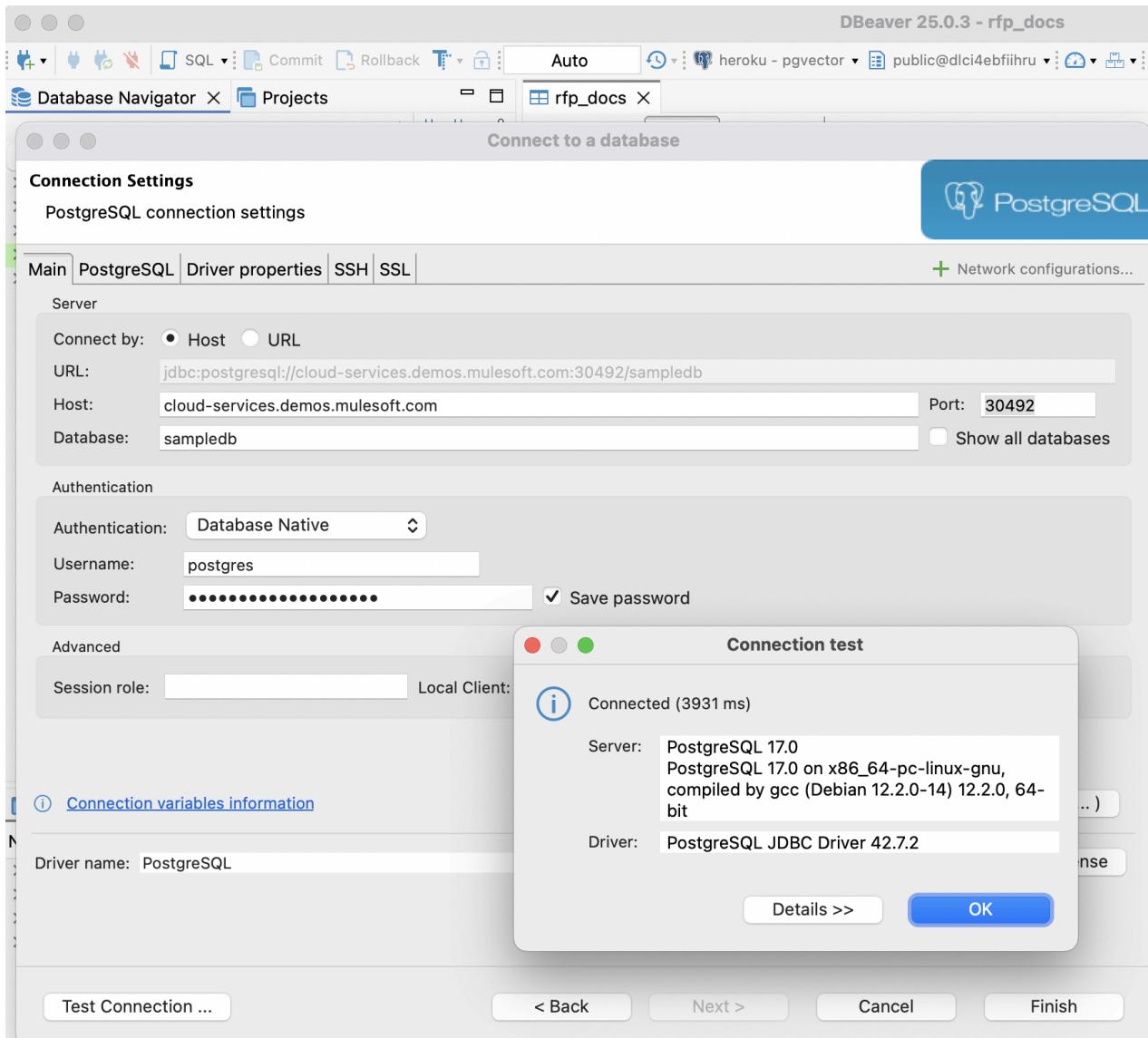
Name *	<input type="text" value="hands-on-vectors"/>
Service end date *	<input type="text" value="06/19/2025"/>
Purpose *	<input type="text" value="Dry Run"/> <span style="float: right;">▼</span>
DB Schema	<input type="text"/>
Database Password *	<input type="password" value="....."/> <span style="float: right;">Show</span>

← Back Create

## Step 3

Verify connection to Store using DBeaver

<https://dbeaver.io/download/>



## Step 4

Once connected check your are able to see database and schema.

