

Camunda 8 - Getting Started With Agentic Orchestration

About AI agents in Camunda

Agentic orchestration

Note: Camunda also offers agentic AI blueprints on the Marketplace.

Glossary of terms

AI agents

Camunda extended this capability with the introduction of the **AI agent outbound connector** and the **Embeddings Vector database connector**.

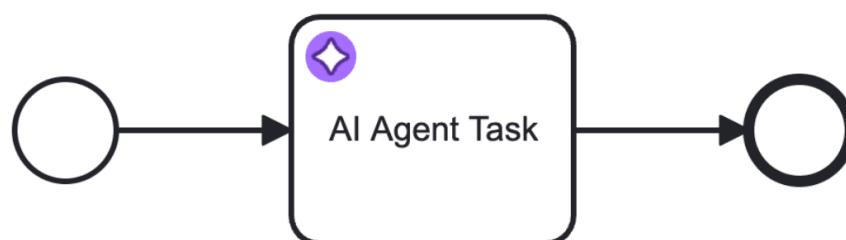
Together, they enable **full-spectrum agentic orchestration**, where workflows seamlessly combine deterministic flow control with dynamic, AI-driven decision-making.

This dual capability **supports both high-volume straight-through processing (STP) and adaptive case management**, empowering agents to plan, reason, and collaborate in complex environments. With Camunda's approach, the AI agents can add additional context for **handling exceptions** from STP.

This represents our next phase of AI agent support and Camunda intend to continue adding richer features and capabilities.

AI agent task

Handles individual tasks by aligning user requests with AI responses, **optimized for focused, single-goal queries**.



AI agent process

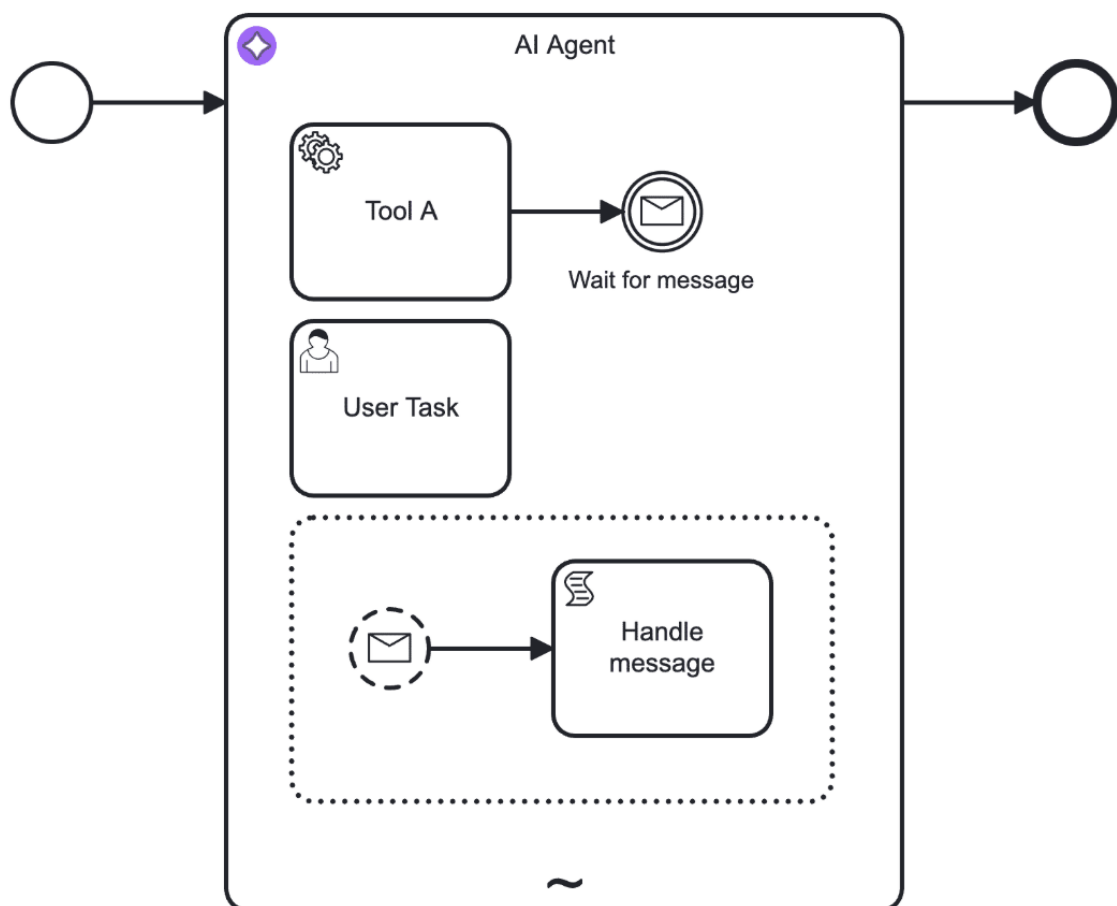
Processes user requests with an integrated, customizable toolbox and services for dynamic workflows.

An AI agent process is an automation within Camunda that leverages ad-hoc sub-processes to perform one or more tasks with **non-deterministic behavior**.

AI agent processes can:

- Make **autonomous decisions** about task execution.
- Adapt their **behavior based on context and input**.
- Handle complex scenarios that require **dynamic response**.
- Integrate with other process components through standard interfaces.

Both AI agents represent the practical implementation of agentic process orchestration within the Camunda ecosystem, combining the flexibility of AI with the reliability of traditional process automation.



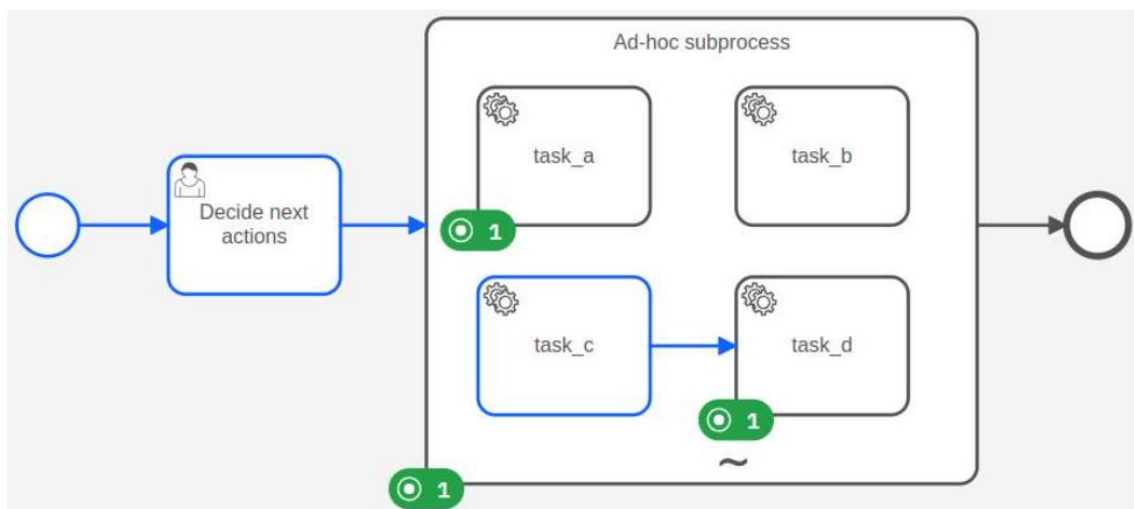
Ad-hoc sub-process (AHSP)

An **ad-hoc sub-process** in **BPMN** is a type of sub-process where **tasks do not follow a predefined sequence flow**.

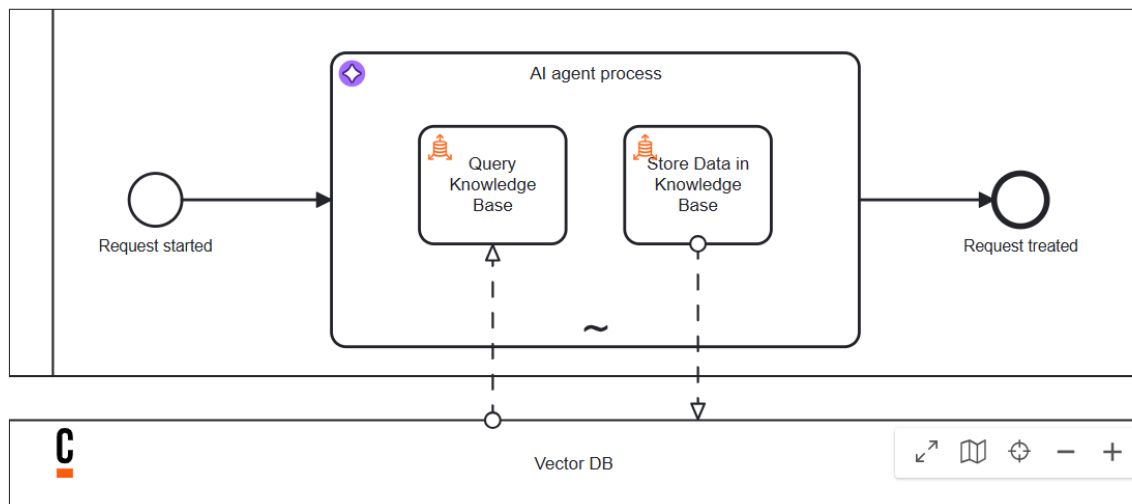
Instead, tasks within the sub-process can be **executed in any order, repeatedly, or skipped entirely**, all determined dynamically at runtime based on the context of the process instance.

This pattern is instrumental in introducing **dynamic (non-deterministic) behavior into otherwise deterministic process models**.

The **ad-hoc sub-process becomes the AI agent process decision workspace**: a flexible execution container where large language models (LLMs) can assess available actions and determine the most appropriate next steps in real time.



Support for agentic AI



Camunda embraces structured orchestration patterns to ensure AI orchestration remains adaptive, goal-oriented, and interoperable across complex systems.

A key evolution is the integration of **Retrieval-Augmented Generation (RAG)** into Camunda's orchestration fabric, enabling AI agents to retrieve relevant external knowledge for informed decisions. Durable, event-driven workflows support retrieval, reasoning, and human collaboration at scale.

Embeddings Vector database connector



The **Embeddings Vector Database Outbound connector** integrates RAG with long-term memory systems, supporting various vector databases (**OpenAI**, **AWS Bedrock**, **Azure OpenAI**, and **Google Vertex AI**). This setup allows agents to use semantically relevant data at runtime and evolve their knowledge base through continuous feedback.

AI agent connector



The **AI agent outbound connector** interfaces with large language models (**OpenAI**, **Anthropic**, **AWS Bedrock**, **Azure OpenAI**, **Google Vertex AI**, and more), enabling autonomous decision-making within BPMN-modeled orchestrations.

How this applies to our exercise

The exercise involves a BPMN process where an **AI agent process** integrates human-in-the-loop and REST calls.

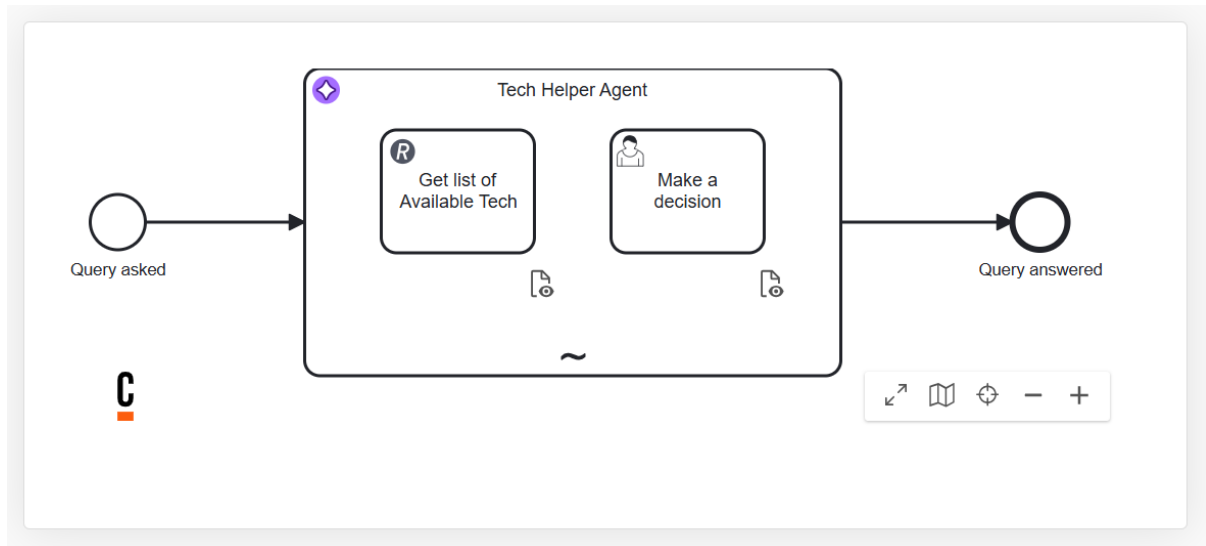
Key Components

- **AI agent process:** This agent receives a goal, instructions, limits, and a chain of thought to decide how to accomplish the goal. It also contains tasks that can be activated based on the prompt provided to the AI agent.

Model overview

Tech Helper Agent

Our example model for this process is the **Tech Helper Agent** as shown below.



Process flow

User inputs: The process starts with a user request.

Decision making: The AI agent process processes the user inputs as the user prompt and decides which tasks to execute within the ad-hoc sub-process.

Execution: The AI agent process executes the appropriate tasks.

Completion: The process completes when no further action is required.

Tasks

The tasks that can be performed are located in the ad-hoc sub-process (AI agent process) and are:

REST call (Get list of Available Tech), with a REST connector task to get the list of the available products

Human-in-the-loop (Make a decision), with a user task and its corresponding form

Process breakdown

Let's take a little deeper dive on the components of the BPMN process before jumping in to build and execute it.

AI agent process

The AI agent process for this exercise uses **AWS Bedrock's Claude 4 Sonnet** model for processing requests. The agent makes decisions on which tools to use based on the context. You can alternatively use other models.

Assumptions and initial configuration

A few assumptions are made for those individuals who will be using this step-by-step guide to create their first AI agent process. These are outlined in this section.

Proper Environment

In order to take advantage of the latest and greatest functionality provided by Camunda, you will need to have a **Camunda 8.8-alpha8** cluster or higher available for use. You will be using Web Modeler and Forms to create your model and human task interface, and then Tasklist when executing the process.

Connector Secrets

Separating sensitive information from the process model is a best practice. You will need to create the appropriate connector secrets within your cluster.

If you do not have existing accounts for the connectors that will be used, you can create them.

You will need to have an AWS with the proper credentials for **AWS Bedrock**. If you do not have this, you can follow the instructions on the **AWS site** to accomplish this and obtain the required keys:

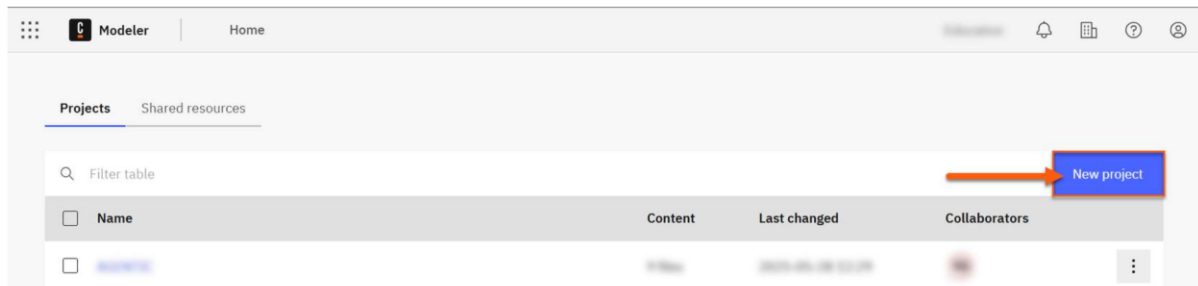
- AWS Region
- AWS Access key
- AWS Secret key

The secrets will be referenced in your model using `{{secrets.yourSecretHere}}` where *yourSecretHere* represents the name of your connector secret.

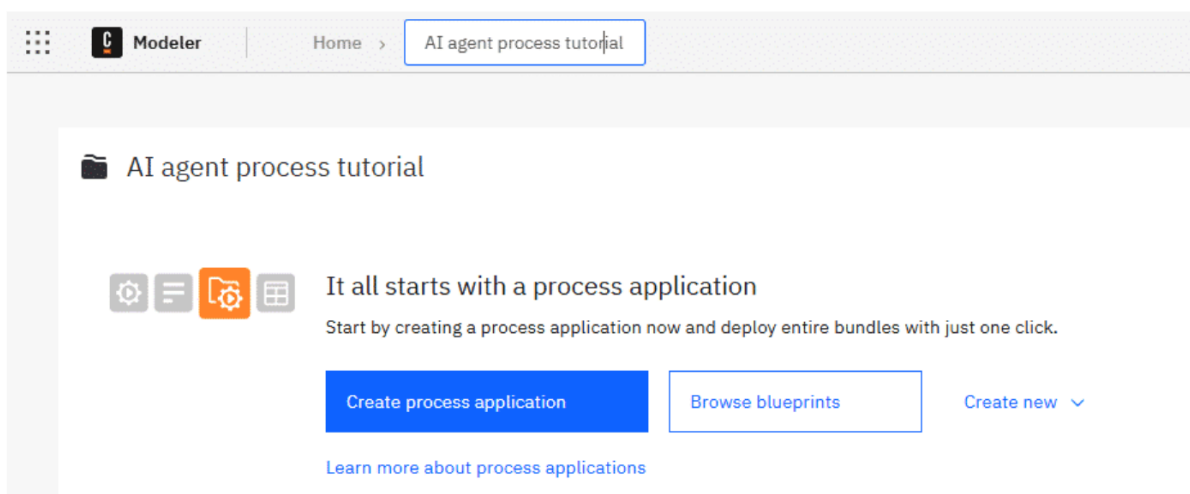
Build your process

Creating your process application

The first step is to create a process application for your process model and any other associated assets.



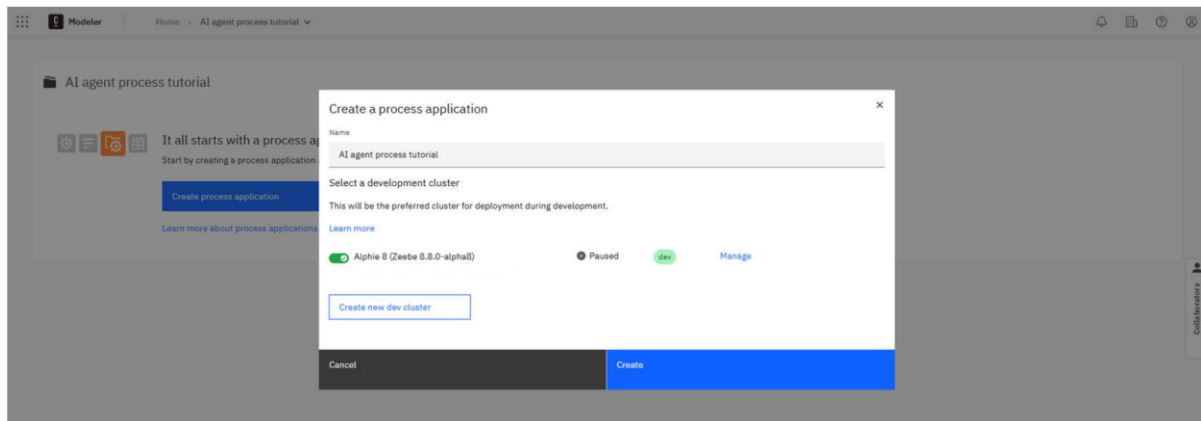
Create a **new project** using the blue button at the top right of your Modeler environment.



Enter the name for your project.

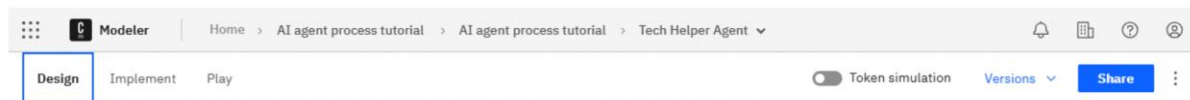
In this case we have used “**AI agent process tutorial**”.

Next, **create your process application** using the blue button provided.



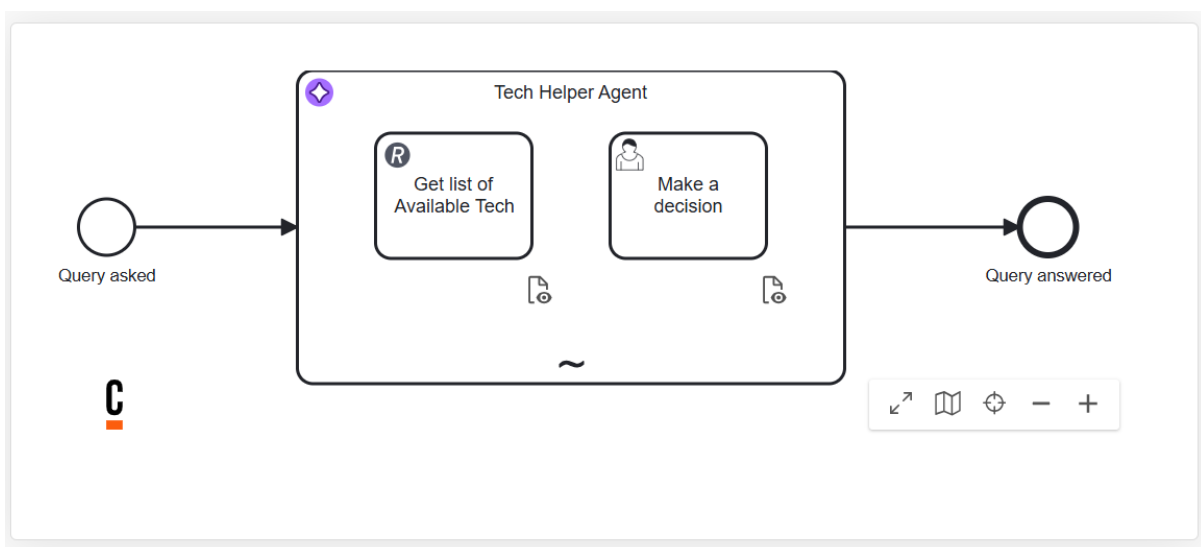
1. Enter the name of your process application, in this example “**AI agent process tutorial**”.
2. Select the **Camunda 8.8** (or greater) cluster that you will be using for your project.
3. Click **Create**.

Change the name of your main process to “**Tech Helper Agent**” and then switch to **Design mode** as shown below.



BPMN diagram

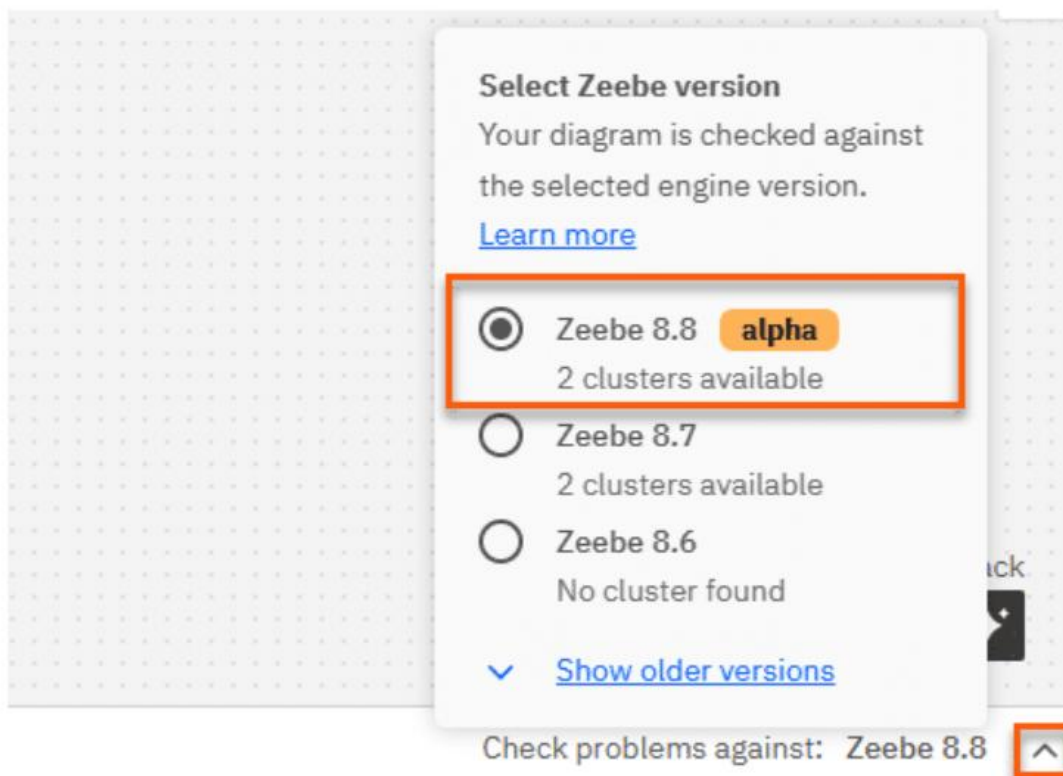
At the end, you will be building the model represented below.



Steps for creating your initial model

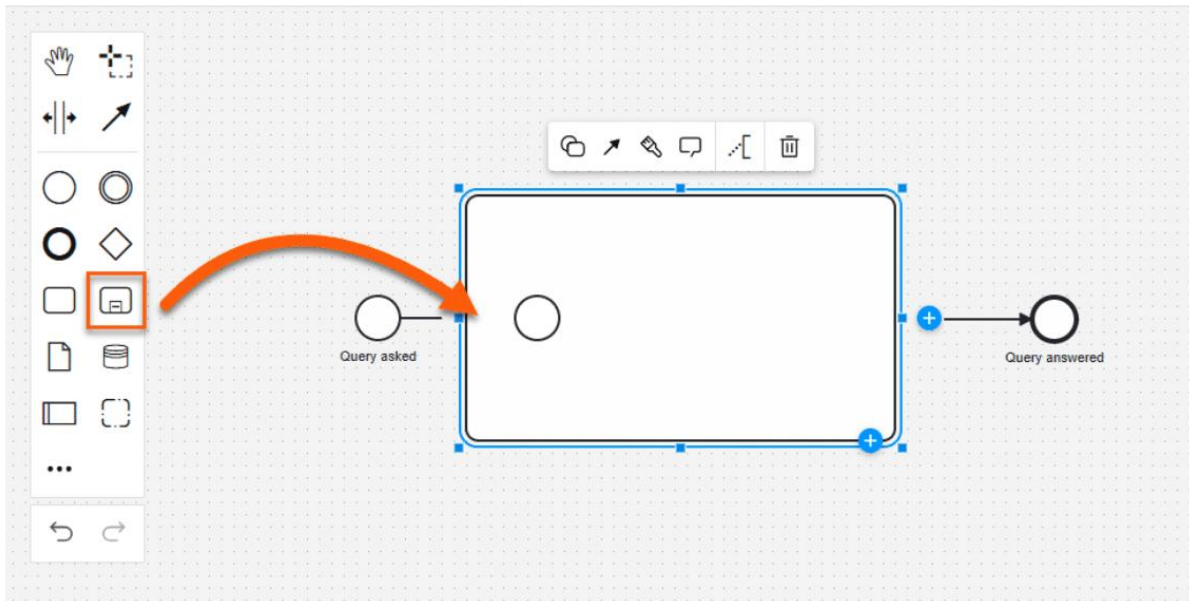


- Name your start event “**Query asked**”
- Name your end event “**Query answered**”

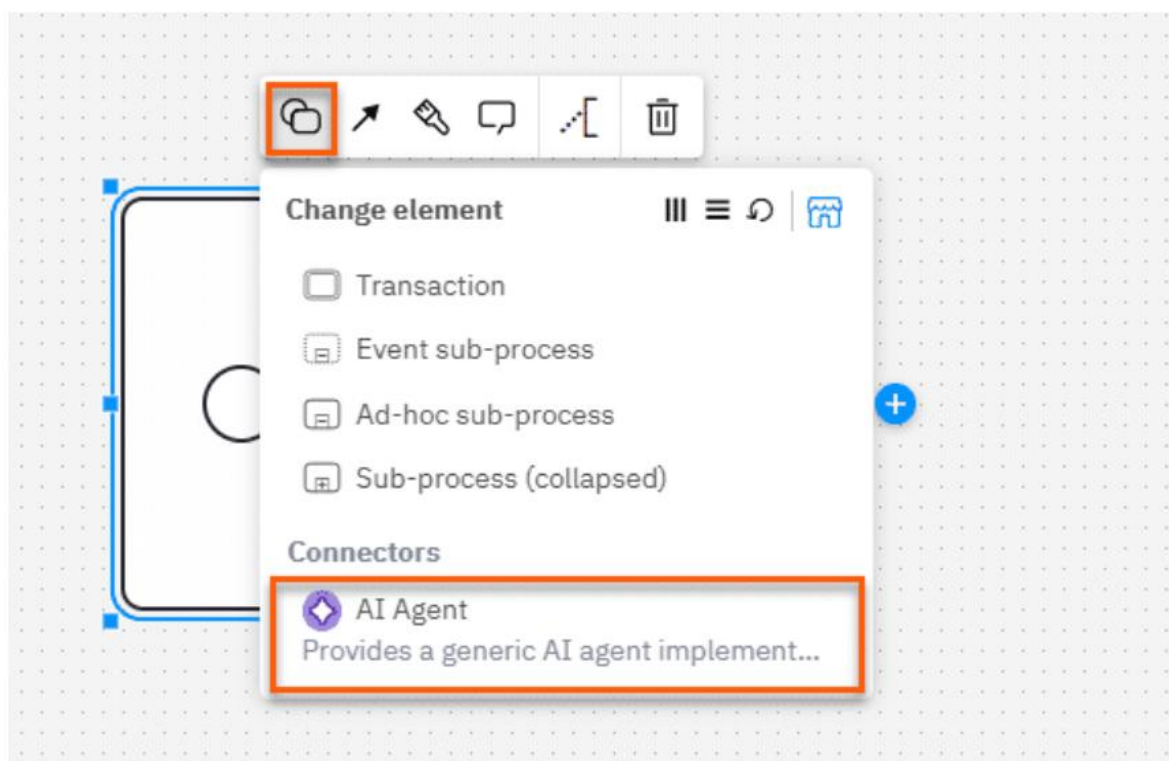


Confirm that you are using the **proper cluster version**.

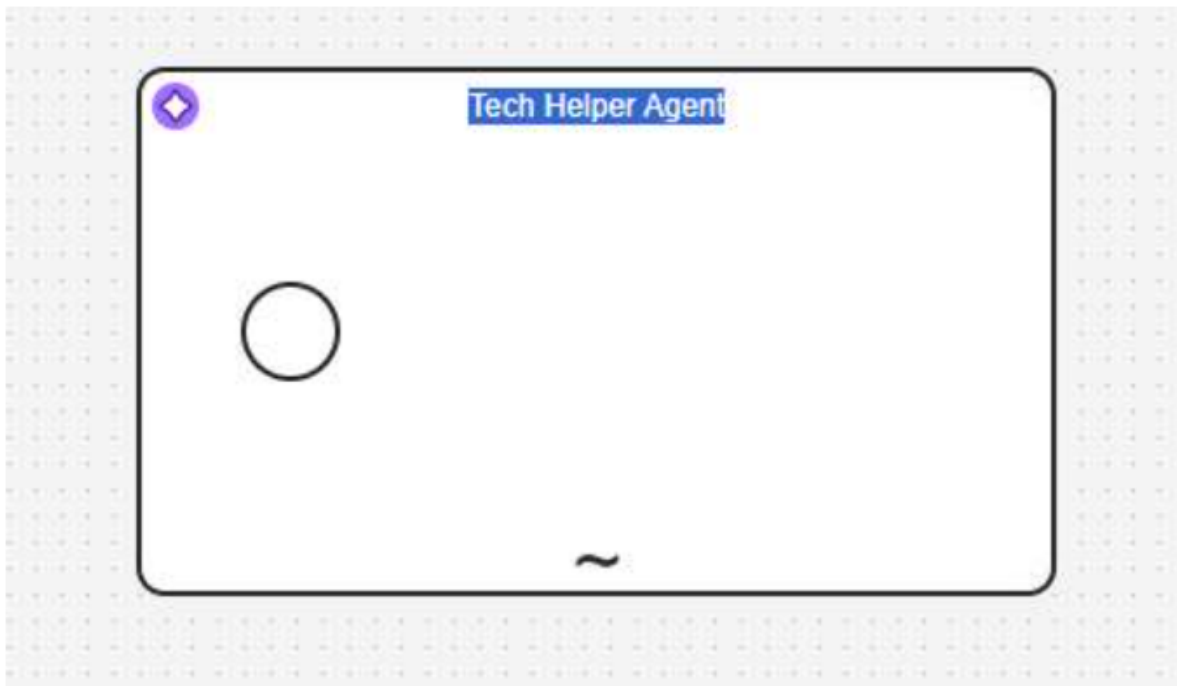
You can do this in **Implement mode**. On the lower right-hand side of Web Modeler be sure to select a cluster that is at least **Camunda 8.8** or higher.



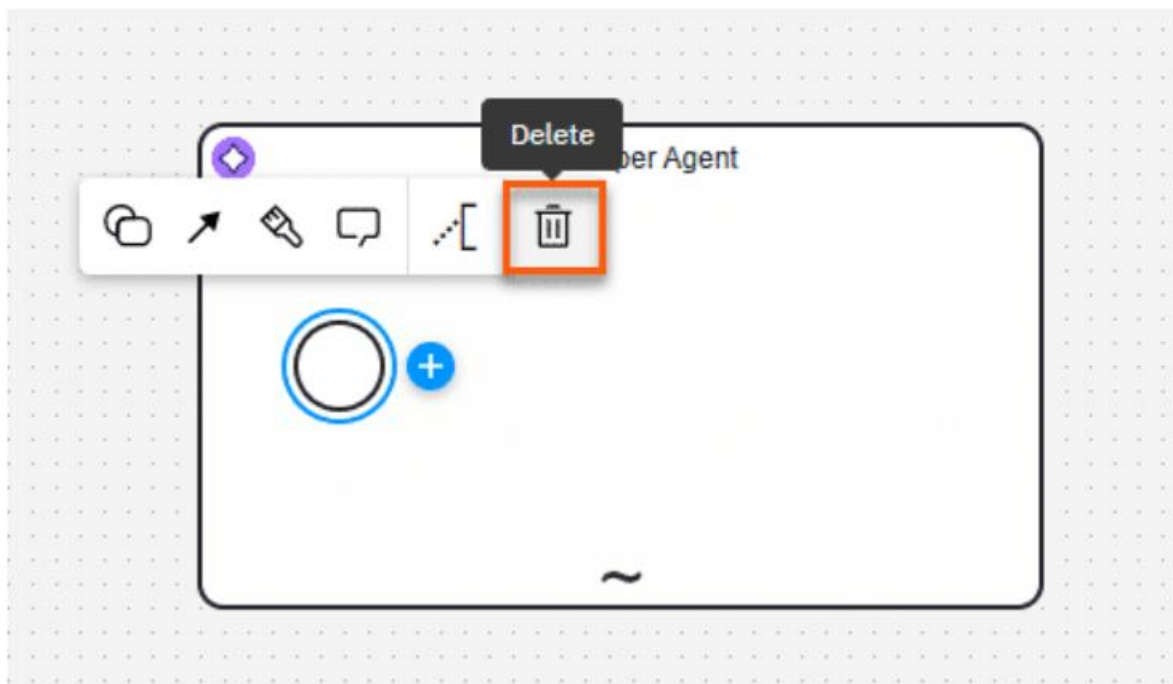
Drag and drop an **expanded sub-process** component between both start and end events.



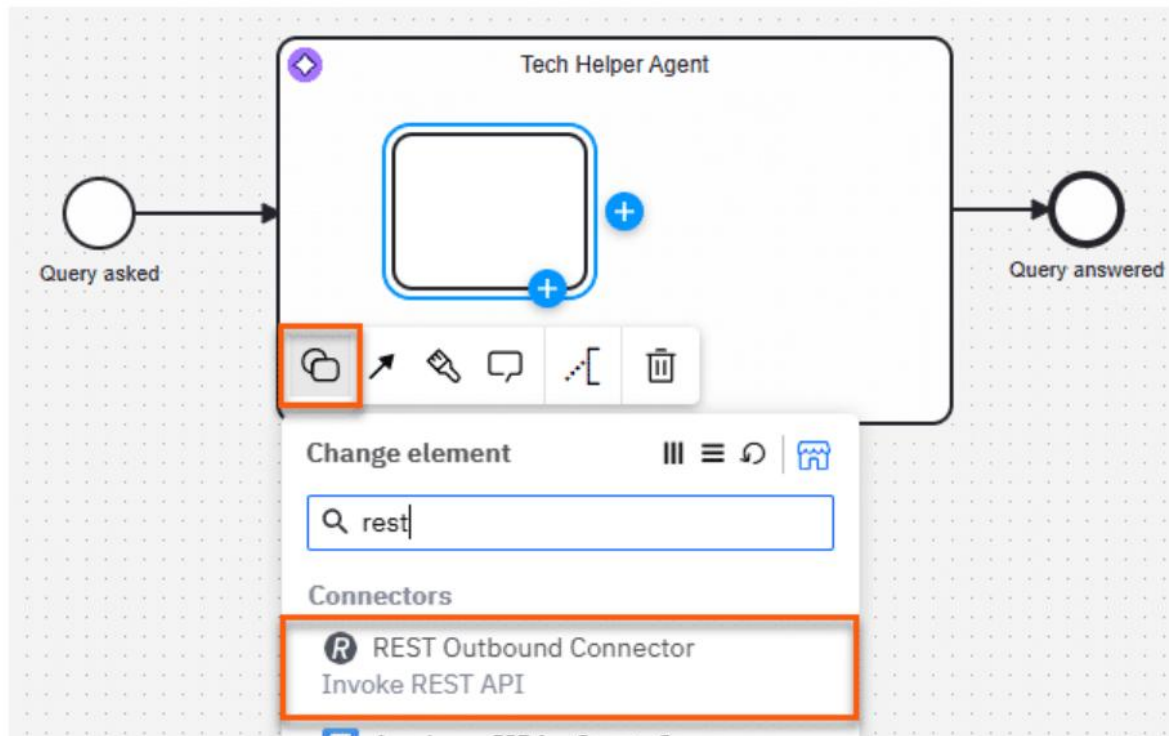
Change element to **AI Agent connector**



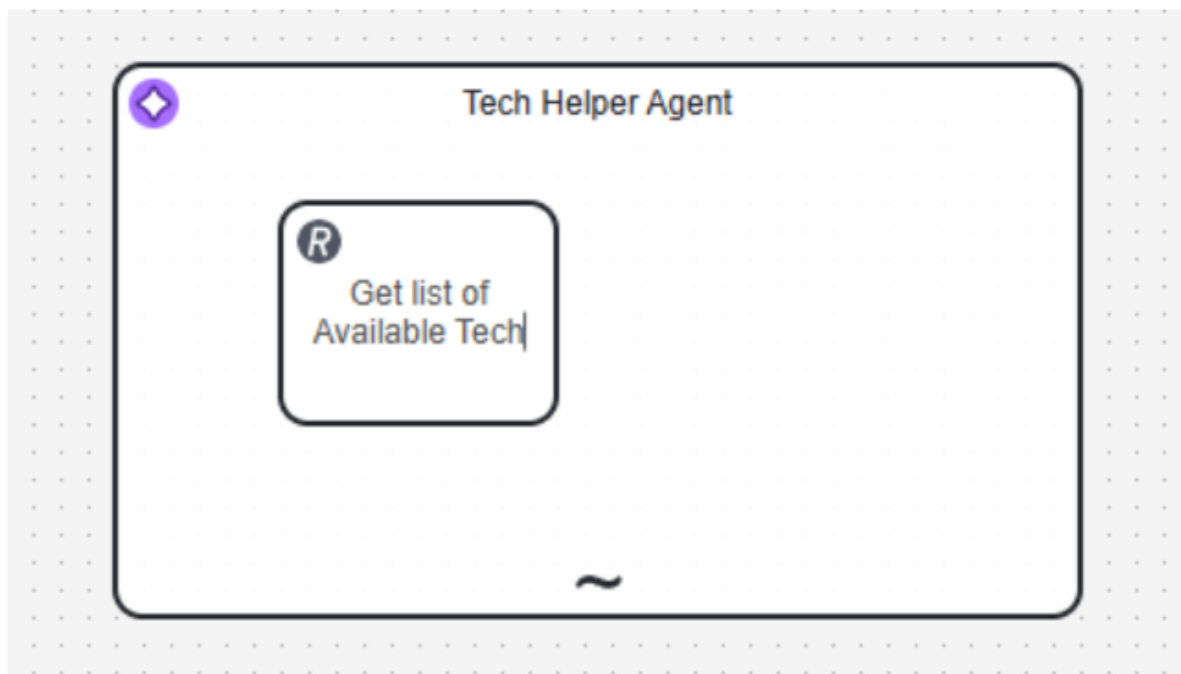
Double click and name it "**Tech Helper Agent**"



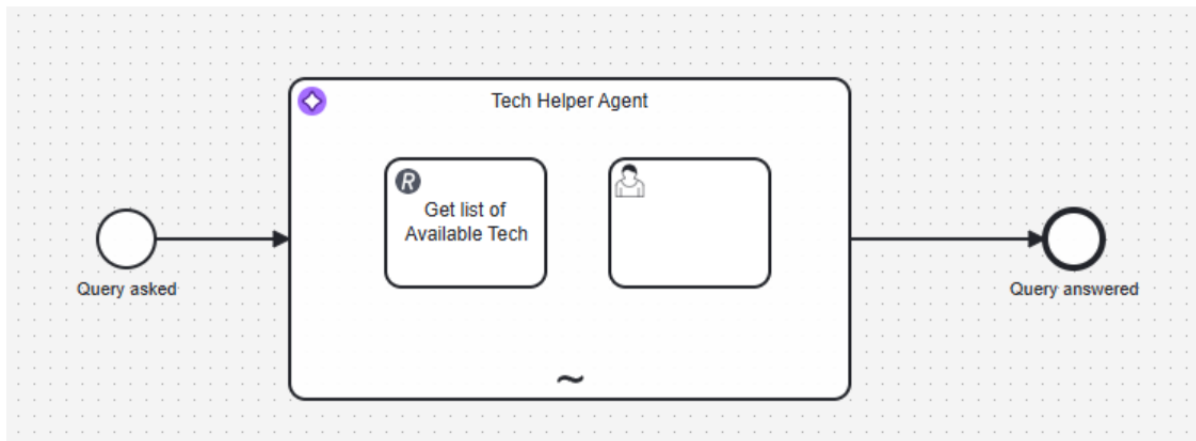
Delete the start event



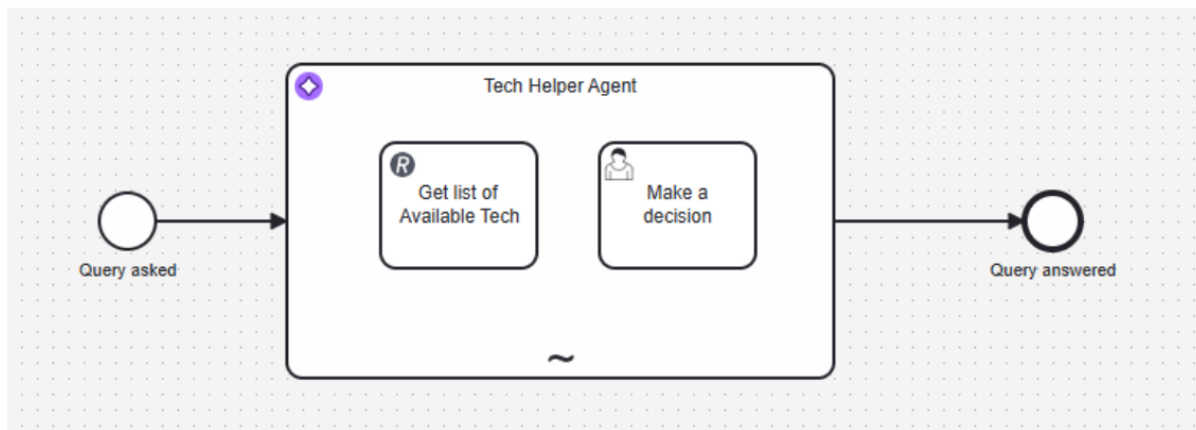
Add a task and change the element to **Rest outbound connector**



Double click and name it "**Get list of Available Tech**"



Add a user task



Double click and name it "**Make a decision**"

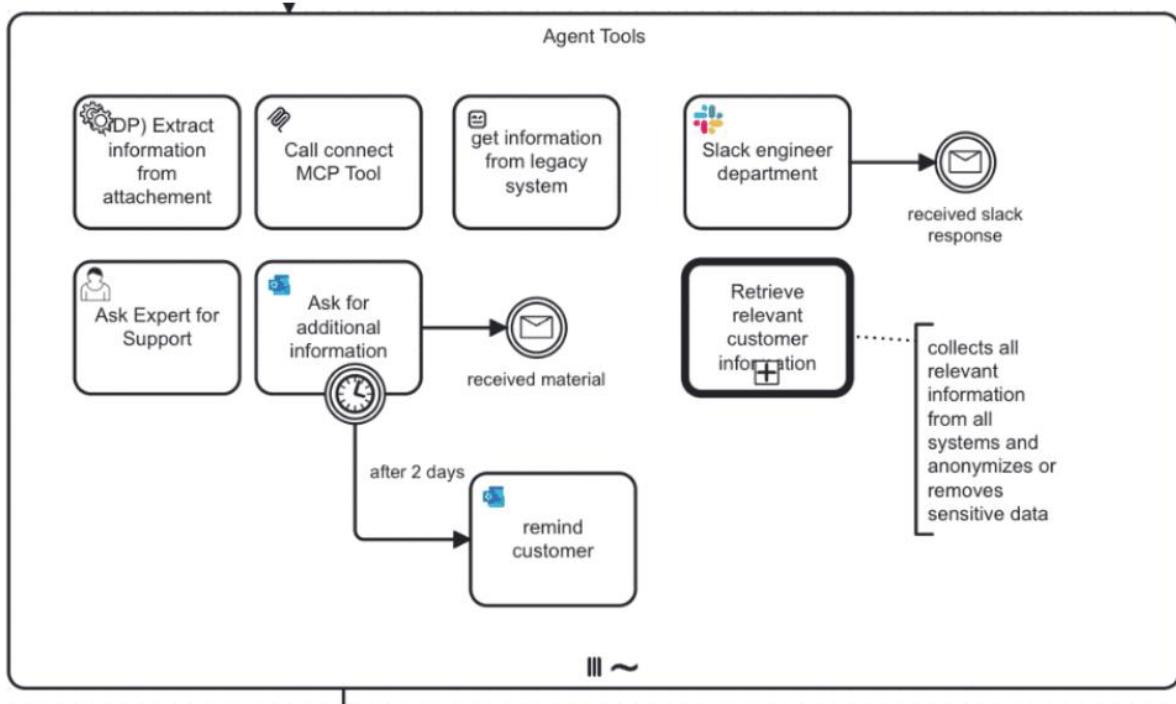
Configure the AI agent process

Introduction

The **AI agent task** and the **AI agent process** can be both configured similarly. The difference lies in the absence of the ad-hoc sub-process (AHSP) on the part of the AI agent task, which is not included by default, although it can be referenced if necessary.


Usually, the **AI agent task** is used to evaluate final LLM outcomes, acting as a judge, or to catalogue those results.



AI agent process Tools



Inside the AHSP you can define the tools as activities. They could be of any kind.

Model provider

 AI AGENT
AI Task Agent

Save as  

General

Documentation

Template Applied

Model provider

Provider

AWS Bedrock

Specify the LLM provider to use.

Region *fx*

{{secrets.AWS_REGION}}

Specify the AWS region (example: eu-west-1)

Endpoint *fx*

Optional custom API endpoint

Authentication

Credentials

Specify the AWS authentication strategy. Learn more at the [documentation page](#)

Access key *fx*

{{secrets.AWS_ACCESS_KEY}}

Provide an IAM access key tailored to a user, equipped with the necessary permissions

Secret key *fx*


{{secrets.AWS_SECRET_KEY}}



Provide a secret key of a user with permissions to invoke specified AWS Lambda function

Different model providers are available:

- Anthropic
- AWS Bedrock
- Azure OpenAI
- Google Vertex AI
- OpenAI


Model ID

 AI AGENT
AI Task Agent


Save as  

General

Documentation

Template Applied 

Model provider

Model 

Model *fx*

us.anthropic.claude-sonnet-4-20250514-v1:0

Specify the model ID. Details in the [documentation](#).

Maximum tokens *fx*

= 5000

Temperature *fx*

= 0.2

top P *fx*

= 0.1

You can specify the model ID you want to use and set some properties:

- **Maximum tokens:** sets the limit on the number of tokens the LLM is allowed to generate for a single response per request.
- **Temperature:** determines the randomness of the model's output.
- **top P:** sets a threshold for the cumulative probability of tokens being considered when generating each word in the output.

Note: Usually, you only need to configure Temperature; Top P is recommended for expert use cases.

System & user prompts

 AI AGENT
AI Task Agent

Save as  

System prompt  

System prompt *fx*

You are **TaskAgent**, a helpful, generic chat agent that can handle a wide variety of customer requests using your own domain knowledge **and** any tools explicitly provided to you at runtime.

0. CONTEXT – WHO IS “USER”?

- **Every incoming user message is from the customer.**
- Treat “user” and “customer” as the same person throughout the conversation.
- Internal staff or experts communicate only through the expert-communication tool(s).

System prompt parameters *fx*

=

Use `{{parameter}}` format in the prompt to insert values defined in this map.

User prompt  

User prompt *fx*

= "someone want to send a message to " + *person* + " and i want to say " + *message* + " The sender of the message is " + *sender*

User prompt parameters *fx*

=

Use `{{parameter}}` format in the prompt to insert values defined in this map.

Documents *fx*

=

Documents to be included in the user prompt.

System prompt


A collection of fundamental instructions provided to a model prior to any user input is known as a system prompt. It guarantees that responses stay consistent and in line with the AI agent's intended purpose by defining the agent's function, behavior, tone, and communication style. Throughout the interaction, the model's interpretation and response to human input are influenced by these instructions.



User prompt

The input message or query you send to the AI to initiate or carry on a discussion is known as a user prompt. It lets the AI know what you need, be it information, task assistance, or just a conversation. Your prompt helps the AI figure out how to react.

You can also use document references to be included into the user prompt.

AI agent task tools

 AI AGENT
AI Task Agent

Save as  

Tools

Ad-hoc sub-process ID *fx*

agentTools

ID of the sub-process that contains the tools the AI agent can use.

Tool call results *fx*

= toolCallResults

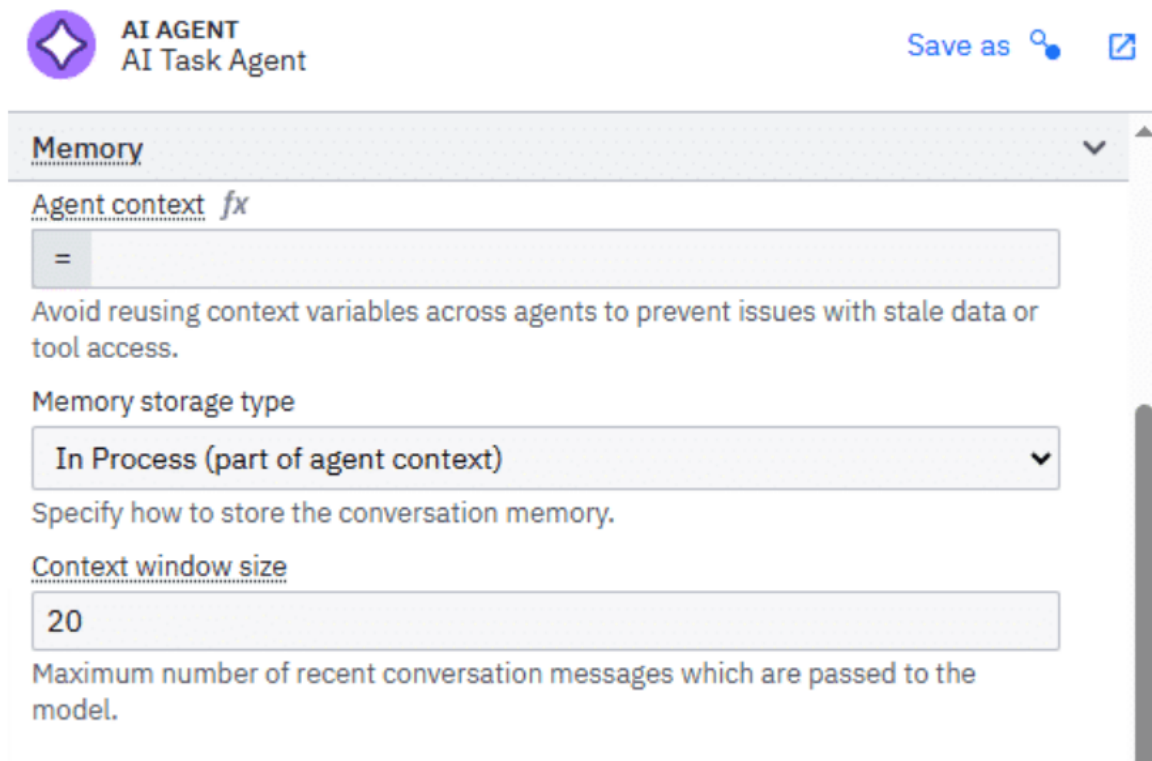
Tool call results as returned by the sub-process.

In order to **connect the AI agent task to the tools**, add an **ad-hoc sub-process ID**. This is **only required for the AI agent task**.

A tools feedback loop that routes into the ad-hoc sub-process and back to the AI agent task connector should be a part of your workflow.

The **Tool call results** specifies where the ad-hoc sub-process's tool call results should be handled. Include this in the tools feedback loop and model it as part of your process.

Memory (1/3)



The screenshot shows the 'Memory' configuration panel for an 'AI AGENT AI Task Agent'. At the top, there's a purple star icon and the text 'AI AGENT AI Task Agent'. To the right are 'Save as' and a share icon. The panel has a title bar 'Memory' with a dropdown arrow. Below it, the 'Agent context' section is labeled 'fx' and contains an equals sign in a text box, with a note: 'Avoid reusing context variables across agents to prevent issues with stale data or tool access.' The 'Memory storage type' section has a dropdown menu set to 'In Process (part of agent context)' with a note: 'Specify how to store the conversation memory.' The 'Context window size' section has a text box containing '20' with a note: 'Maximum number of recent conversation messages which are passed to the model.'

All pertinent information needed for the agent to maintain the feedback loop between tool calls, user requests, and LLM answers is contained in the **agent context**. It is only necessary to configure the agent context when the agent should pick up an existing conversation. Otherwise, it should be used in combination with the **Include agent context** setting in the **Response** section and be aligned with the used result variable. Use this option if you need to re-inject the previous agent context into a future agent execution, for example when modeling a user feedback loop between an agent and a user task.

Example: =agent.context, =anotherAgent.context


Different memory storage types are available:



In Process (part of agent context)



- Great for prototyping as it is visible in Operate
- Variable size limits

Context window size: This can be used to restrict how many messages are sent to the model. Only the most recent communications up to the set limit will be sent to the LLM by the agent. Older messages won't be sent to the model, but they will be stored in the discussion storage.

Memory (2/3)

 **AI AGENT**
Handle student request

Save as  


Memory  

Agent context *fx*

=

Avoid reusing context variables across agents to prevent issues with stale data or tool access.

Memory storage type

Camunda Document Storage 

Specify how to store the conversation memory.

Document TTL *fx*

P30D

How long to retain the conversation document as ISO-8601 duration (example: P14D).

Custom document properties *fx*

=

An optional map of custom properties to be stored with the conversation document.

Camunda Document Storage

- Serialized JSON representation
- Writes a new document per iteration
- Deletes older documents
- Limited to Document TTL

Memory (3/3)

Memory storage type

Custom Implementation (Hybrid/Self-Managed only) ▼

Specify how to store the conversation memory.

Implementation type *fx*

my-conversation

Parameters *fx*


=



Parameters for the custom memory storage implementation.


Custom Implementation (Hybrid/Self Managed only)

- E.g. RDBMS

Limits, Event handling, Response, and Output Mapping

 AI AGENT
AI agent process


Save as  

Limits 


Maximum model calls

10


Maximum number of calls to the model as a safety limit to prevent infinite loops.

Event handling 


Event handling behavior

Wait for tool call results 

Behavior in combination with an event sub-process.

Response 

Response format


Text 

Specify the response format. Support for JSON mode varies by provider.

☐ Parse text as JSON
Tries to parse the LLM response text as JSON object.

☐ Include assistant message
Include the full assistant message as part of the result object.

☒ Include agent context
Include the agent context as part of the result object.

Output mapping 

Result variable

agent

Name of variable to store the response in

Limits

The maximum number of calls to the model, set to 10 by default.

Event handling

It can wait for all tool calls to complete before handling the event, or interrupt the tool call results.

Response

The **Text response format** can:

- Be plain or parsed as JSON

- Include the full **assistant message** and/or the **agent context** as part of the result object

The **JSON response format** lets you add a JSON Schema to instruct the model how to structure the JSON output.

Output mapping

The whole AI Agent response will be stored in the **agent** result variable.

Configure the AI agent process

Now you need to configure the brains of the operation, the AI agent process. This task takes care of accepting the user prompt and sending the request to the LLM to determine next steps. In this section, you will configure this agent with specific variables and values based on your model and using some default values where appropriate.

You will now move into setting up the details for each construct to implement the model, so switch to the Implement mode in your Web Modeler.

Model Provider

First, you need to pick the “**Provider**” that you will use for the exercise, in our case, we are selecting “AWS Bedrock”.

The next field is the “**Region**” for AWS. A secret was created for the region (AWS_REGION) which will be used in this field.

Remember the secrets will be referenced in your model using {{secrets.yourSecretHere}} where yourSecretHere represents the name of your connector secret.

Update the authorization credentials with your AWS **Access Key** and **Secret key** from your connector secrets.



General


Documentation

Template

Applied 

Model provider

Provider

AWS Bedrock 

Specify the LLM provider to use.

Region *fx*


{{secrets.AWS_REGION}}

Specify the AWS region (example: `eu-west-1`)

Endpoint *fx*

Optional custom API endpoint

Authentication

Credentials 

Specify the AWS authentication strategy. Learn more at the [documentation page](#)

Access key *fx*

{{secrets.AWS_ACCESS_KEY}}

Provide an IAM access key tailored to a user, equipped with the necessary permissions

Secret key *fx*

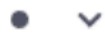
{{secrets.AWS_SECRET_KEY}}

Provide a secret key of a user with permissions to invoke specified AWS Lambda function

Model ID

Select the model ID you will be using from your Model Provider. We are using `us.anthropic.claude-sonnet-4-20250514-v1:0` at the time of creating this course.

Model



Model *fx*

`us.anthropic.claude-sonnet-4-20250514-v1:0`

Specify the model ID. Details in the [documentation](#).

Maximum tokens *fx*

=

Temperature *fx*

=

top P *fx*

=

Configure the System prompt

Default system prompt is good enough to start with, but you are welcome to create your own.

Hint: If you are creating your own prompt, try taking advantage of tools like ChatGPT or other AI tools to help you build a strong prompt.

System prompt

System prompt *fx*

```
= "You are **TaskAgent**, a helpful, generic chat agent that  
can handle a wide variety of customer requests using your  
own domain knowledge **and** any tools explicitly provided  
to you at runtime.
```

```
  
If tools are provided, you should prefer them instead of  
guessing an answer. You can call the same tool multiple  
times by providing different input values. Don't guess any  
tools which were not explicitly configured. If no tool  
matches the request, try to generate an answer. If you're  
not able to find a good answer, return with a message  
stating why you're not able to.
```

```
  
Wrap minimal, inspectable reasoning in *exactly* this XML  
template:
```

```
<thinking>  
<context>...briefly state the customer's need and current  
state...</context>  
<reflection>...list candidate tools, justify which you will  
call next and why...</reflection>  
</thinking>
```

```
  
Reveal **no** additional private reasoning outside these  
tags."
```

Configure the User prompt

In the User prompt field, enter: request.

The user prompt will use the **request** input used when the process instance starts.

User prompt

User prompt *fx*

= request

Documents *fx*

=

Documents to be included in the user prompt.

Configure the Response

The response in **text** format will include:

- Assistant message
- Agent context.

Response

Response format

Text

Specify the response format. Support for JSON mode varies by provider.

☐ Parse text as JSON

Tries to parse the LLM response text as JSON object.

☒ Include assistant message

Include the full assistant message as part of the result object.

☒ Include agent context

Include the agent context as part of the result object.

Configure the REST connector

Set the REST outbound connector with the <https://restful-api.dev/> GET call and store the result.

Change element ID

First, change the ID to "**Activity_GetListOfAvailableTech**"

 **REST OUTBOUND CONNECTOR**
Get list of Available Tech

Save as 🔍 [

General ● ▼

Name

Get list of Available Tech

ID

Activity_GetListOfAvailableTech

Documentation

Describe the usage of this tool task to the AI agent.

For example:

This is the tool to get all the available technology products.

Documentation ● ▼

Element documentation

This is the tool to get all the available technology products

Authentication

No authentication is needed.

Authentication ▼

Type

None ▼

Choose the authentication type. Select 'None' if no authentication is necessary

HTTP endpoint

Use a **GET** method to get the list of objects from this **URL**: <https://api.restful-api.dev/objects>

HTTP endpoint

Method

GET

URL *fx*

<https://api.restful-api.dev/objects>

Headers *fx*

=

Map of HTTP headers to add to the request

Query parameters *fx*

=

Map of query parameters to add to the request URL

☐ Store response

Store the response as a document in the document store

Output mapping

Set a callback result by using the `toolCallResult` variable.

Output mapping

Result variable

`toolCallResult`

Name of variable to store the response in

Result expression *fx*

=

Expression to map the response into process variables

Configure the user task

The user task will be useful to present the results of the REST call with the list of technology available and to get the instructions from the user on what to do next.

First, you need to provide some **inputs** and **outputs** to the user task. Then, **create a form** to map them accordingly to display the REST call result list and to gather the user instructions.

The REST call results will be set as an input and the user instructions as an output.

Input parameters

Input parameters are defined as **fromAI()** FEEL functions.

The AI agent connector parses the FEEL expressions, extracts function calls, and merges them into a schema. The LLM decides the value of the parameters when calling the tool.

fromAI() function

The screenshot shows a configuration form for an HTTP endpoint. The 'Method' is set to 'GET'. The 'URL' field contains a FEEL expression: `"https://jsonplaceholder.typicode.com/users/" + string(fromAI(toolCall.id, "The user ID", "number"))`. Below the URL field, there are sections for 'Headers' and 'Query parameters', both with empty input fields. At the bottom, there is a checkbox labeled 'Store response' with the description 'Store the response as a document in the document store'.

HTTP endpoint

Method

GET

URL *fx*

= "https://jsonplaceholder.typicode.com/users/" +
string(fromAI(toolCall.id, "The user ID", "number"))

Headers *fx*

=

Map of HTTP headers to add to the request

Query parameters *fx*

=

Map of query parameters to add to the request URL

☐ Store response
Store the response as a document in the document store

In the picture, the **URL property** of a REST connector activity is a FEEL expression which builds the URL, using the **fromAI()** function to complete the user id.

The **fromAi()** function has:

- the input parameter **name** (toolCall.id)
- the parameter **description** ("The user ID")
- the optional parameter **type** ("number").

fromAi() - Example 1

▼ aSimpleValue

Local variable name

aSimpleValue

Variable assignment value *fx*

= fromAi(toolCall.aSimpleValue, "A simple value")

▼ anEnumValue

Local variable name

anEnumValue

Variable assignment value *fx*

= fromAi(toolCall.anEnumValue, "An enum value", "string", { enum: ["A", "B", "C"] })

▼ anArrayValue

Local variable name

anArrayValue

Variable assignment value *fx*

= fromAi(toolCall.anArrayValue, "An array value", "array", {
 items: {
 type: "string",
 enum: ["foo", "bar", "baz"]
 }
})

fromAi() - Example 2

▼ aCombinedValue

Local variable name

aCombinedValue

Variable assignment value *fx*

= "https://example.com/" + fromAi(toolCall.urlPath, "The URL path to use", "string")

▼ multipleParametersInDifferentFormats

Local variable name

multipleParametersInDifferentFormats

Variable assignment value *fx*

= {
 comment: "Multiple params, positional & named, simple & complex",
 foo: [fromAi(toolCall.firstValue), fromAi(toolCall.secondValue, "The second value",
 "integer")],
 bar: {
 baz: fromAi(description: "The third value to add", value: toolCall.thirdValue),
 qux: fromAi(toolCall.fourthValue, "The fourth value to add", "array", {
 "items": {
 "type": "string",
 "enum": ["foo", "bar", "baz"]
 }
 })
 }
}

Change element ID

First, change the ID to "**Activity_MakeADecision**"



General

Name

Make a decision

ID

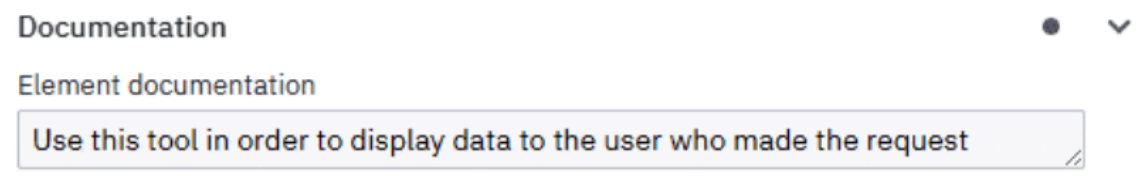
Activity_MakeADecision

Documentation

Describe the usage of this tool task to the AI agent.

For example:

Use this tool in order to display data to the user who made the request



Documentation

Element documentation

Use this tool in order to display data to the user who made the request

Create a new input

Add a new local variable named "**queryResult**".

The AI agent connector sets the value thanks to the fromAI(toolCall) function.

Copy & paste the following code:

```
fromAi(toolCall.queryResult, "This is the result of the query, format this in markdown and add some icons")
```

Inputs

+

1

▼

▼ queryResult

Local variable name

queryResult

Variable assignment value *fx*

= fromAi(toolCall.queryResult, "This is the result of the query, format this in markdown and add some icons")

Create an new output

Add a new process variable named "**toolCallResult**".

The value contains the response from the user, that you will map later in the task form.

Copy & paste the following code:

```
{
  "responseFromUser": responseFromUser
}
```

Outputs

+

1

▼

▼ toolCallResult

Process variable name

toolCallResult

Variable assignment value *fx*

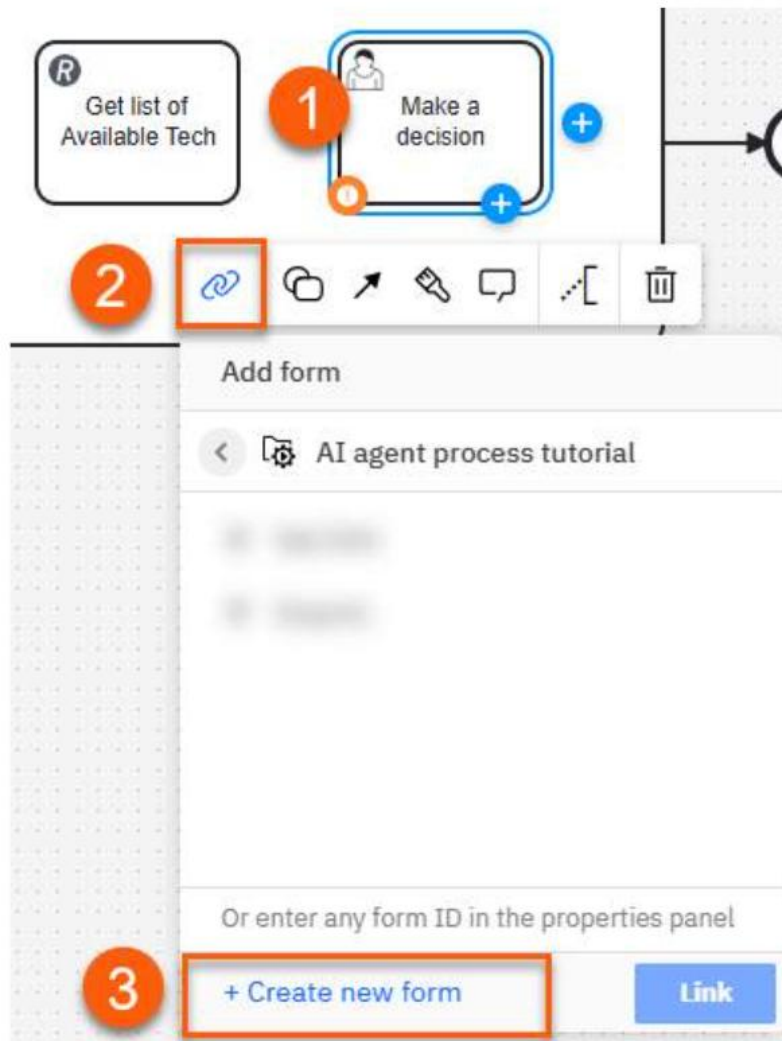
= {
 "responseFromUser": responseFromUser
}

All set in the user task to create a form to interact with the end-user!

Now create a form to map the input and the output.

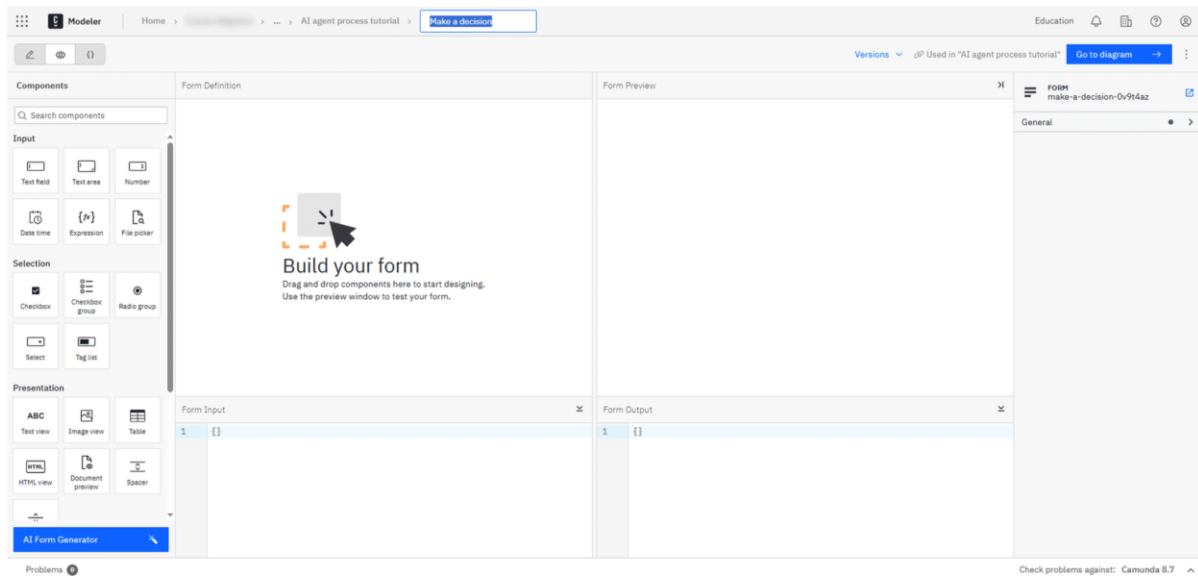
Access the Form Builder

1. Select the user task
2. Click the chain link icon
3. Select **+ Create new form**



Name your form

Give it a name (e.g. Make a decision).



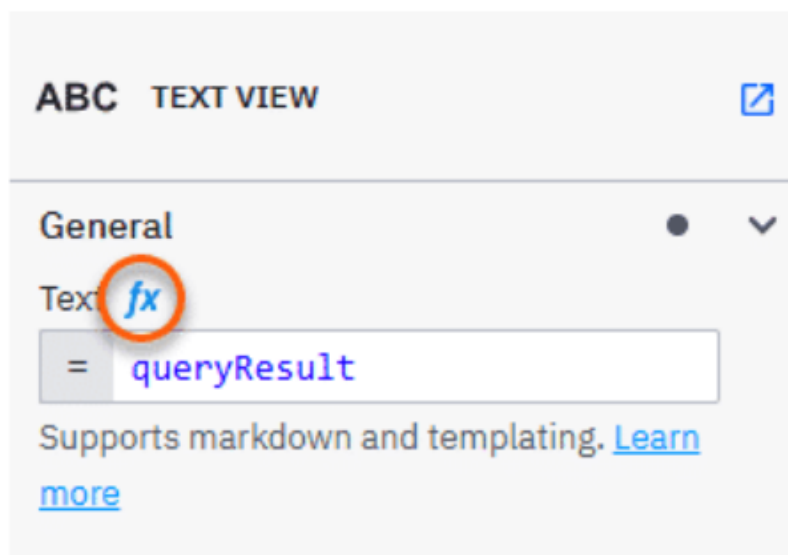
You will need the following fields on this form:

Component	Text/Label	Required?	Key
Text view	=queryResult	N	
Text field	What to do next?	Y	responseFromUser

Dynamic value

The queryResult is the input variable that will be used to display the list of technology available, so this value property must be **dynamic (fx)**.

Set dynamic value with FEEL expression.



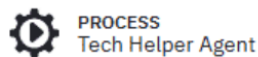
The screenshot shows the AI Form Generator interface. On the left is a 'Components' panel with categories: Input (Text field, Text area, Number), Selection (Checkbox, Checkbox group, Radio group, Select, Tag list), and Presentation (Text view, Image view, Table, HTML view, Document preview, Spacer). The main area is divided into 'Form Definition' and 'Form Preview'. The 'Form Definition' shows a text field with the label 'What to do next?' and a note 'ABC Text view is populated by an expression'. The 'Form Preview' shows the rendered text field. On the right, a 'TEXT FIELD' configuration panel is open, showing fields for 'Field label', 'Field description', 'Key' (set to 'responseFromUser'), 'Default value', 'Disabled' (toggle), and 'Read only' (toggle). Below these are expandable sections for 'Condition', 'Layout', 'Appearance', and 'Validation'. At the bottom of the main area are 'Form Input' and 'Form Output' sections. 'Form Input' contains a single row with an empty text field. 'Form Output' contains three rows: a variable declaration '1 v {', a value assignment '2 "responseFromUser": **', and a closing bracket '3 }'.

Once you have completed your form, click **Go to Diagram ->** to return to your model.

This screenshot shows the same AI Form Generator interface as the previous one, but with the 'TEXT FIELD' configuration panel closed. The 'Form Definition' and 'Form Preview' sections remain the same. The 'Form Input' and 'Form Output' sections are also visible. The 'Form Output' section now shows the completed configuration for the text field, including the variable declaration and the value assignment.

Deploy and Run

If you haven't already, you might want to give your process a better **name and ID** before deploying.



General

Name

Tech Helper Agent

ID

Process_TechHelperAgent

Version tag

☒ Executable

Start a process instance

1. Click **Deploy & run**
2. **Select** your cluster.
3. Add a variable (copy & paste below)

```
{
```

```
"request": "Can you show me the list of all tech stuff?"
```

```
}
```

4. Click **Deploy & run**

The screenshot shows a modal window titled "Deploy & run diagram" with a close button (X) in the top right corner. Below the title is a "Select a cluster" section. It states "Your diagram will be deployed in cluster:" and lists a cluster named "Alphie 8 (Zeebe 8.8.0-alpha8)" which is marked as "Healthy" and has a "dev" environment tag. A "Manage" link is next to it. Below this is a "Related Forms" section, stating "The following form will be automatically deployed with the diagram:" and showing a form titled "Make a decision" with a right arrow icon. The "Add variables (optional)" section explains that JSON data can be added to the instance and provides a link to "variables". A "Variables" section contains a text area with a JSON object: {"request": "Can you show me the list of all tech stuff?"}. At the bottom are two buttons: "Cancel" and "Deploy & run".

Deploy & run diagram

Select a cluster

Your diagram will be deployed in cluster:

☒ Alphie 8 (Zeebe 8.8.0-alpha8) Healthy dev [Manage](#)

Related Forms

The following form will be automatically deployed with the diagram:

Make a decision →

Add variables (optional)

Optionally specify JSON data to add to the newly created instance. Variables can be accessed inside the process instance. Learn more about [variables](#).

Variables

```
{
  "request": "Can you show me the list of all tech stuff?"
}
```

Cancel Deploy & run

Instance follow-up

Open the instance in **Operate**.

Enable the **Show Execution Count**. This will visualize the number of executions of each BPMN elements.

The Tech Help Agent executed the REST call and it's currently waiting for the execution of the user task.

Process Name: Tech Helper Agent
Process Instance Key: 4503599628713639
Version: 4
Start Date: 2025-10-02 13:42:16
End Date: ---
Parent Process Instance Key: None
Called Process Instances: None

Instance History: ☐ Show End Date ☒ Hide Execution Count

Name	Value
agentContext	{ "state": "READY", "metrics": { "modelCalls": 2, "tokenUsage": { "inputTokenCount": 2272, "outputTokenCount": 582, "toolDefinitions": { "name": "Activity_GetListOfAvailableTech", "description": "This is the tool to get all the available technology products", "inputSchema": {} } } } }
request	"Can you show me the list of all tech stuff?"
toolCallResults	[]

Instance History

Click the **Tech Helper Agent** in the Instance History section to extend its content.

It clearly shows:

- A first `Activity_TechHelperAgent#innerInstance` already completed, representing the first iteration where the AI agent connector retrieved the list of available tech
- And a second `Activity_TechHelperAgent#innerInstance` still waiting for execution, representing the current active user task

Process Name: Tech Helper Agent
Process Instance Key: 4503599628713639
Version: 4
Start Date: 2025-10-02 13:42:16
End Date: ---
Parent Process Instance Key: None
Called Process Instances: None

Instance History: ☐ Show End Date ☒ Hide Execution Count

Details:

- Element Instance Key: 4503599628713644
- Execution Duration: 7 minutes (running)
- Retries Left: 3

Name	Value
adHocSubProcessElements	[{ "elementId": "Activity_GetListOfAvailableTech", "elementName": "Get list of Available Tech", "documentation": "This is the tool to get all the available technology products", "elementId": "Activity_MakeADecision", "elementName": "Make a decision" }]
data	{ "systemPrompt": "You are **TaskAgent**, a helpful, generic chat agent that can handle a wide variety of customer requests using your own domain knowledge **and** any tools explicitly provided to you at runtime.\n\nIf tools are provided, you should prefer them instead of your own domain knowledge.", "prompt": "You are **TaskAgent**, a helpful, generic chat agent that can handle a wide variety of customer requests using your own domain knowledge **and** any tools explicitly provided to you at runtime.\n\nIf tools are provided, you should prefer them instead of your own domain knowledge." }
provider	{ "type": "bedrock", "region": "us-east-1", "credentials": { "type": "credentials", "accessKey": "AKIA...", "secretKey": "..." } }

Open Tasklist

Click the **Make a decision** user task and **Open Tasklist**

Process Name: Tech Helper Agent
Process Instance Key: 4503599628713639
Version: 4
Start Date: 2025-10-02 13:42:16
End Date: --
Parent Process Instance Key: None
Called Process Instances: None

Details for 'Make a decision' task:
Element Instance Key: 4503599628713683
Execution Duration: 20 minutes (running)

Instance History:
Tech Helper Agent
Query asked
Tech Helper Agent
Activity_TechHelperAgent#innerInstance
Activity_TechHelperAgent#innerInstance

Variables:
Name: queryResult
Value: [{"@type": "Smartphones", "name": "Google Pixel 6 Pro", "color": "Cloudy White", "storage": "128 GB", "price": 599}, {"@type": "Smartphones", "name": "Apple iPhone 12 Mini", "color": "Blue", "storage": "128 GB", "price": 699}, {"@type": "Smartphones", "name": "Apple iPhone 12 Pro Max", "color": "Cloudy White", "storage": "512 GB", "price": 1199}, {"@type": "Smartphones", "name": "Apple iPhone 11", "color": "Purple", "storage": "64 GB", "price": 399}, {"@type": "Smartphones", "name": "Samsung Galaxy Z Fold2", "color": "Brown", "storage": "512 GB", "price": 1999}, {"@type": "Audio Accessories", "name": "Apple AirPods (3rd Generation)", "color": "White", "price": 129}, {"@type": "Audio Accessories", "name": "Beats Studio3 Wireless", "color": "Red", "price": 349}, {"@type": "Computers", "name": "Apple MacBook Pro 16 (2019)", "color": "Space Gray", "storage": "1 TB", "price": 1849}, {"@type": "Wearables", "name": "Apple Watch Series 8", "color": "Elderberry", "size": "41mm", "price": 399}, {"@type": "Tablets", "name": "Apple iPad Mini 5th Gen (64 GB)", "color": "Pink", "storage": "64 GB", "price": 399}, {"@type": "Tablets", "name": "Apple iPad Mini 5th Gen (256 GB)", "color": "Pink", "storage": "256 GB", "price": 599}, {"@type": "Tablets", "name": "Apple iPad Air 4th Gen (64 GB)", "color": "Blue", "storage": "64 GB", "price": 499}, {"@type": "Tablets", "name": "Apple iPad Air 4th Gen (256 GB)", "color": "Blue", "storage": "256 GB", "price": 699}]]

Make a decision form

Select the **Make a decision** task.

The form displays the list of available technology retrieved from the REST call

Task: Make a decision
Tech Helper Agent
Unassigned
Assign to me

Available Technology Products

Here's the complete list of all tech items we have available:

- Smartphones**
 - Google Pixel 6 Pro - Cloudy White, 128 GB
 - Apple iPhone 12 Mini - 256GB, Blue
 - Apple iPhone 12 Pro Max - Cloudy White, 512 GB
 - Apple iPhone 11 - 64GB, Purple - \$389.99
 - Samsung Galaxy Z Fold2 - Brown - \$1699.99
- Audio Accessories**
 - Apple AirPods - 3rd Generation - \$120
 - Beats Studio3 Wireless - Red, high-performance wireless noise cancelling headphones
- Computers**
 - Apple MacBook Pro 16 (2019) - Intel Core i9, 1 TB storage - \$1,849.99
- Wearables**
 - Apple Watch Series 8 - Elderberry strap, 41mm case
- Tablets**
 - Apple iPad Mini 5th Gen (64 GB) - 7.9" screen
 - Apple iPad Mini 5th Gen (256 GB) - 7.9" screen
 - Apple iPad Air 4th Gen (64 GB) - \$419.99
 - Apple iPad Air 4th Gen (256 GB) - \$519.99

Total: 13 different tech products available

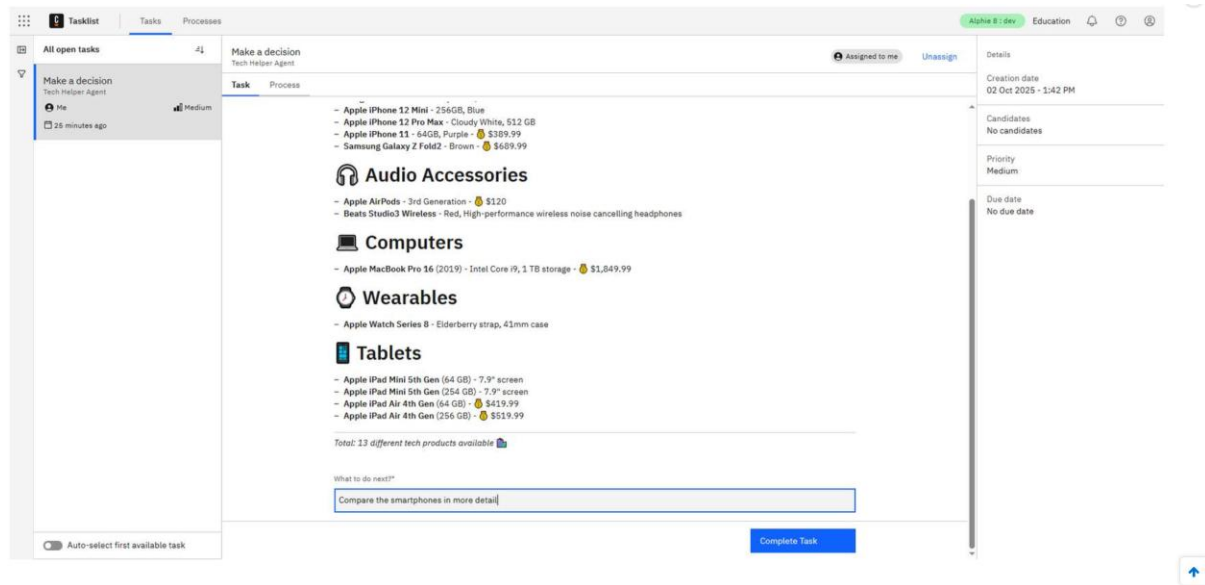
Complete the task

Assign the task to you.

Add the next instruction.

E.g. Compare the smartphones in more detail

Click **Complete Task**

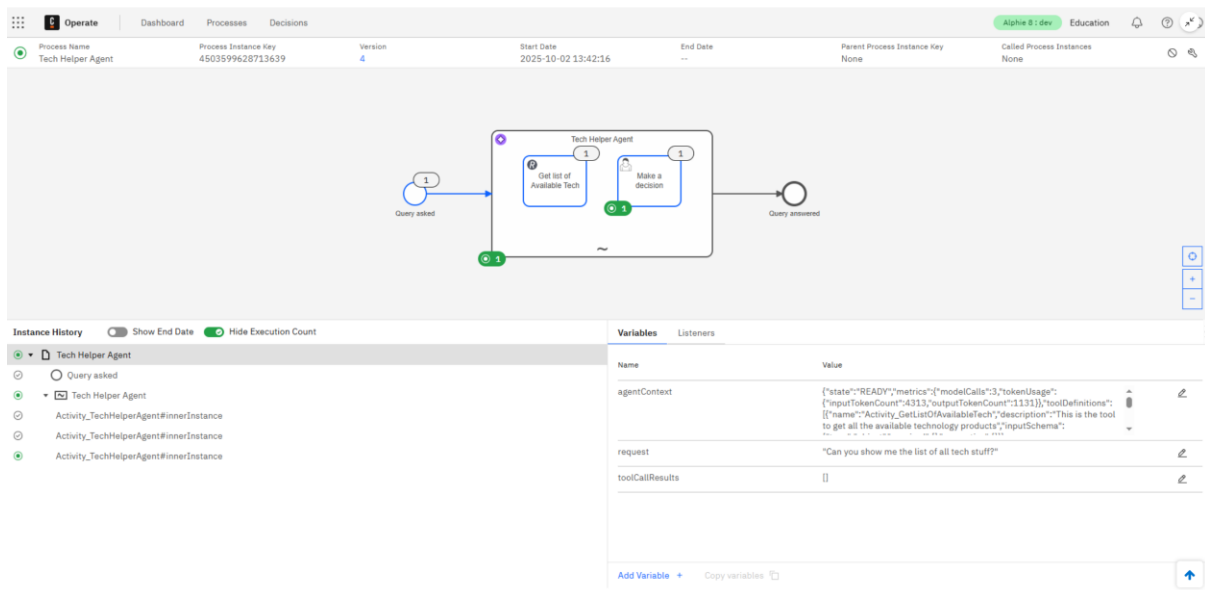


AI agent thinking

The AI agent is now deciding what to do.

It doesn't need a tool to perform the next instruction (compare the smartphones) because it already has what it needs to do it.

So it comes back to the user with the comparison result.



Smartphone comparison

Back to Tasklist, a new Make a decision task appears with the smartphone comparison.

Tasklist

Task

Processes

Alpha 8 : dev

Education

🔔

?

⌵

All open tasks

4

Make a decision

Tech Helper Agent

Unassigned

6 minutes ago

Medium

Auto-select first available task

Make a decision

Tech Helper Agent

Unassigned

Assign to me

Task

Process

📱

Smartphone Comparison

Here's a detailed comparison of all available smartphones:

🔍

Detailed Smartphone Breakdown

📱

Apple iPhone 12 Pro Max

📱 Color: Cloudy White

💾 Storage: 512 GB

🏷️ Tier: Premium/Pro model

🔥 Features: Largest iPhone 12 series, Pro camera system

📱

Samsung Galaxy Z Fold2

📱 Color: Brown

💰 Price: \$6699.99

🏷️ Type: Foldable smartphone

🔥 Features: Dual screen, innovative folding design

📱

Apple iPhone 12 Mini

📱 Color: Blue

💾 Storage: 256 GB

🏷️ Tier: Compact flagship

🔥 Features: Smallest iPhone 12 series

📱

Apple iPhone 11

📱 Color: Purple

💾 Storage: 64 GB

💰 Price: \$389.99

🏷️ Tier: Previous generation, budget-friendly

🔥 Features: Proven performance, lower price point

Details

Creation date

02 Oct 2025 - 2:08 PM

Candidates

No candidates

Priority

Medium

Due date

No due date

⬆