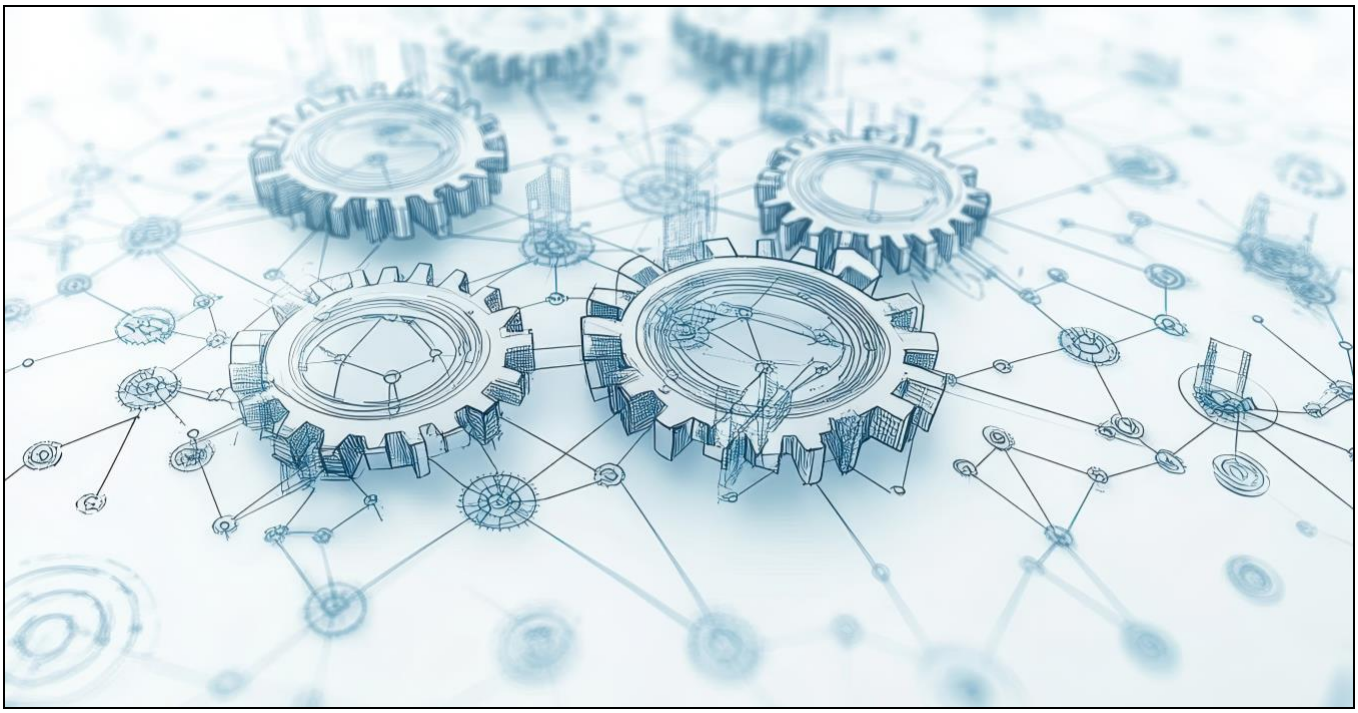
Level 4

Watsonx Orchestrate

Hands-on lab guide: Collaborating agents with Python and Langfuse



Gerry Baird
gerry.baird@uk.ibm.com
Learning Content Development, Data & AI

Version 2.0

Aug 2025

Document edition: June 2025

LEGAL NOTICES

Table of Contents

Introduction

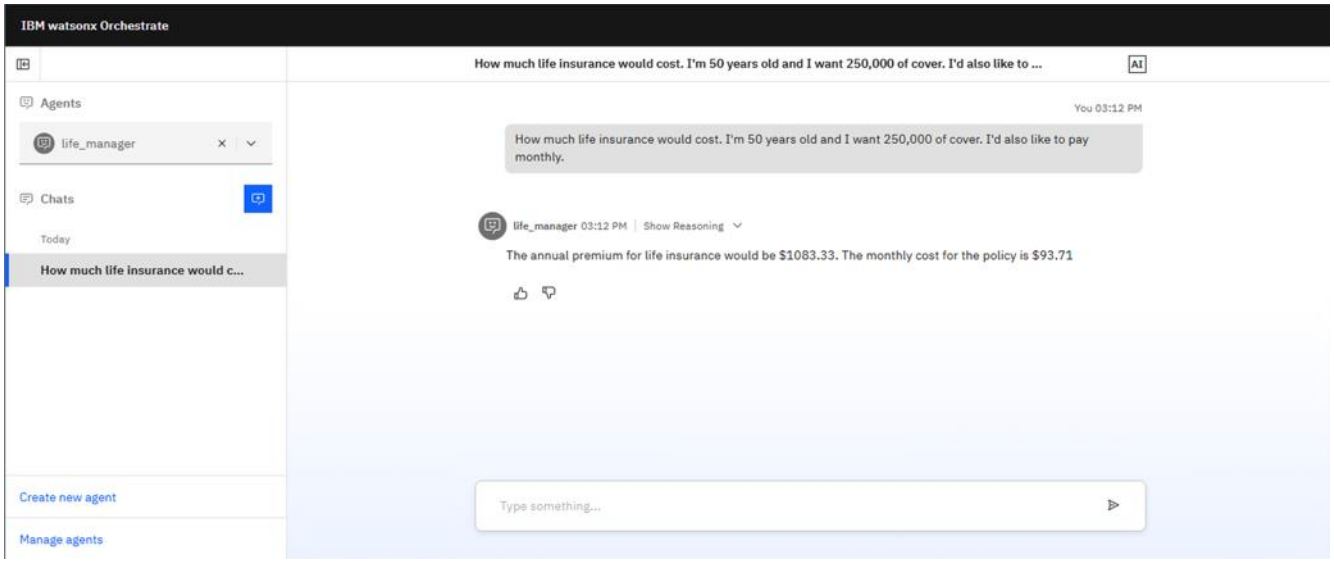
Lab overview

This lab will show you how to create an agentic solution that utilizes collaborating agents and three simple Python tools to provide coverage quotations and payment illustrations for a hypothetical insurance provider.

When complete, the top-level agent will be able to respond to complex requests that contain multiple instructions. The agent will analyze the request and work towards a solution, using its collaborating agents and their tools to produce a response. In the example below, the agent has identified the two parameters needed to calculate the cost of the insurance cover that have been included the users request, these are:

- The amount of cover required
- The customers age

Additionally, the agent has determined that the customer has asked to pay for the product monthly. To respond to this additional requirement, the agent will transfer control to another collaborating agent that has specific knowledge and tools to provide monthly payment illustrations. Once the monthly payment illustration has been calculated, control is passed back to the top-level agent and the response below is generated.



What you will learn

Once you have completed this lab you will have developed skills enabling you to complete the following tasks:

1. Create tools from Python functions and import them into Orchestrate.
2. Create an agent that uses Python based tools.
3. Create a manager agent that routes to collaborating.
4. Review agent description and instructions that define their behavior.
5. Review trace information that explains the agents reasoning and chain of thought.

Environment

This lab has been tested with the Agent Development Kit (ADK) version 1.8.1.

How to get support

If you encounter an issue with this lab, you can request assistance through the following channels:

- Review the Troubleshooting appendix in this guide first.
- IBMers can use the #ba-techlcd-support Slack channel.
- If you are an IBM Business Partner and require assistance, please open a support case at IBM Technology Zone Help.

Prerequisites

Make sure the following prerequisites are satisfied before proceeding with this lab:

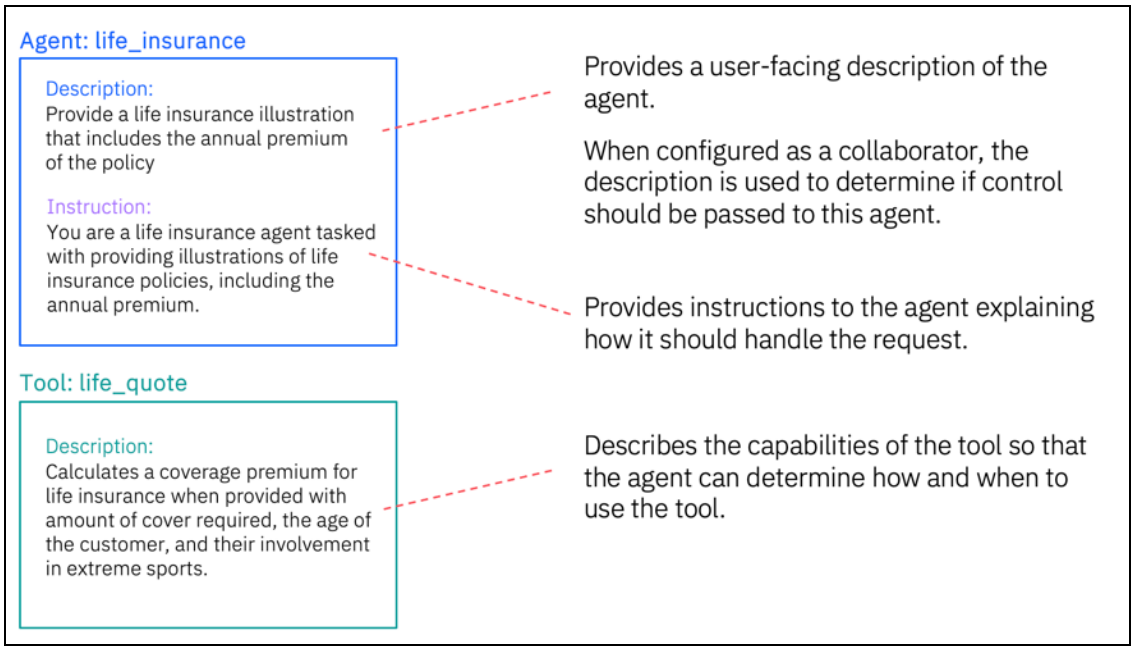
- You have performed the Preparation Lab guide including the environment validation step
- You have your ADK environment up and running (Orchestrate server started after validating the environment in the preparation guide OR after having performed another ADK related lab)

Note: If for some reason your server is not started correctly, reset your server following the Reset your environment appendix.

The tools created in this lab are based on Python functions, but knowledge of Python programming is not required for his lab.

Life insurance agent

In this section you will create and test the first life insurance agent. You will create a single agent and provide this agent with a single tool based on a Python function. The diagram below provides an overview of the configuration used in this section. Review the agent and tool configuration shown below before proceeding.



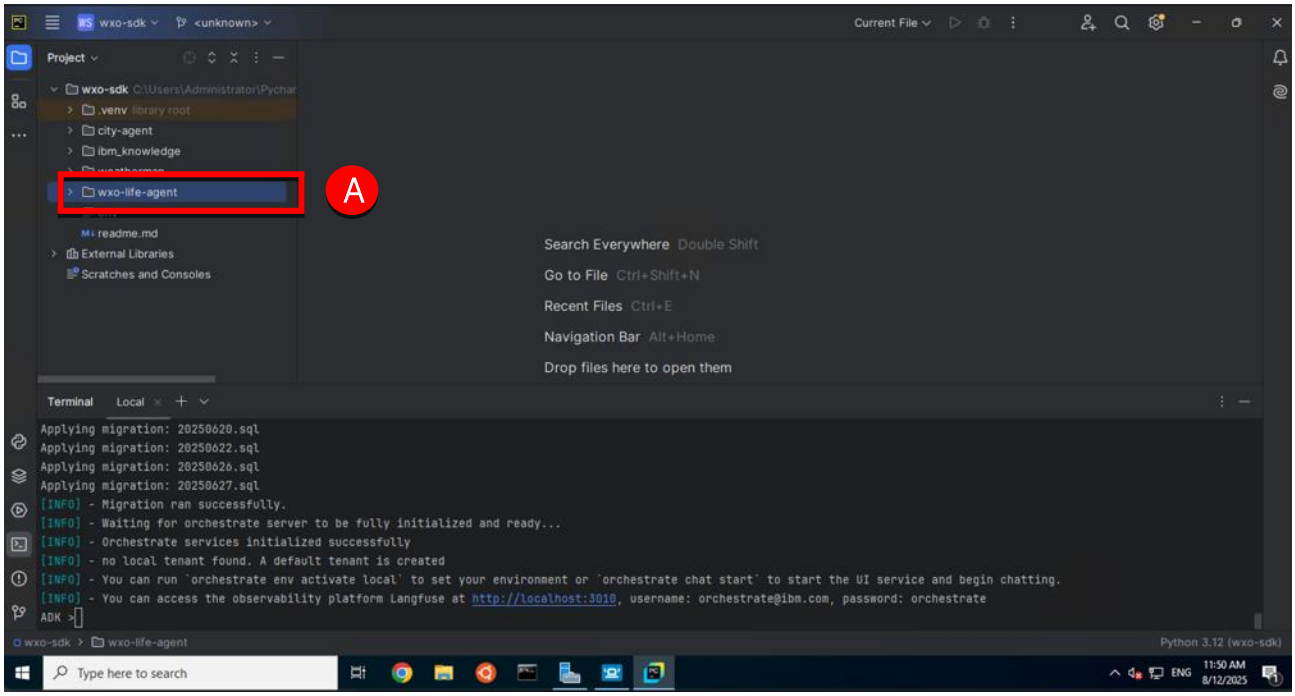
In addition to the tool description, any input and output parameters used by the tool are analyzed and added to the tool configuration. At runtime, the agent shares this configuration as part of the system prompt and is used for planning.

Tool and agent creation

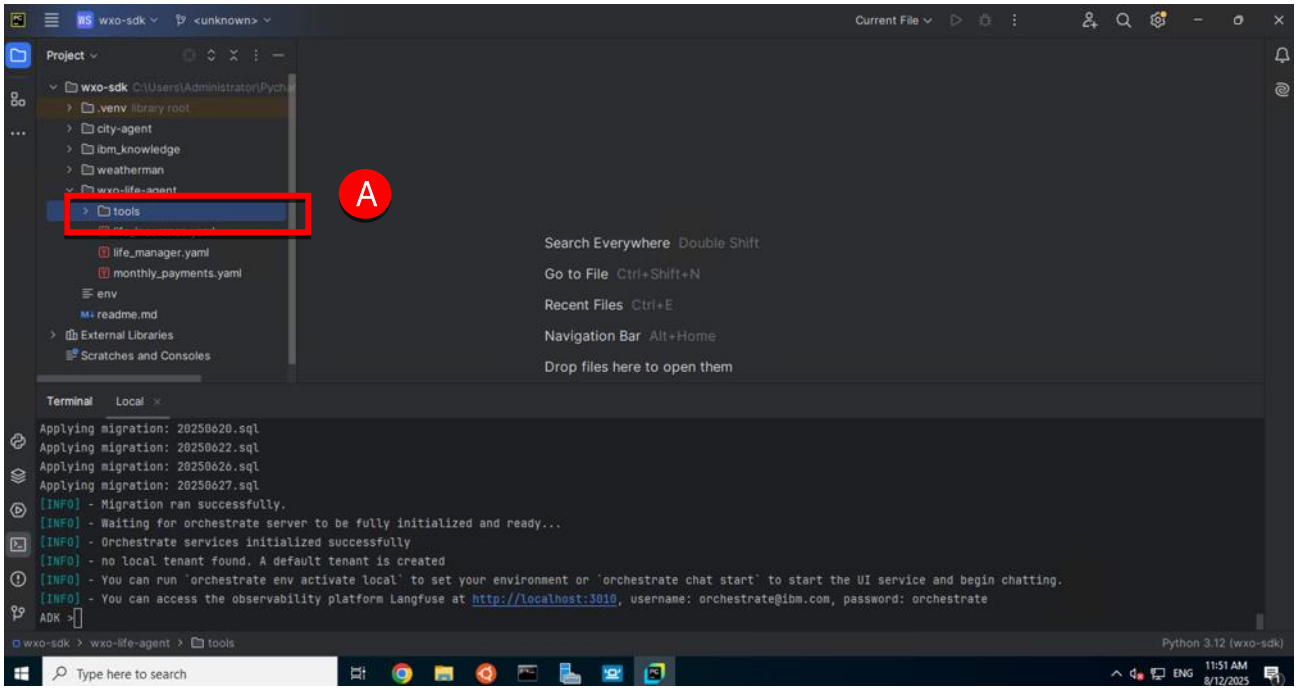
In this section you will review the Python code used by the tool and create the tool and agent in watsonx Orchestrate using the command line interface.

1. Open the ADK environment and start the watsonx Orchestrate server if not yet done. Refer to the Reset your environment appendix if your orchestrate server is not started.

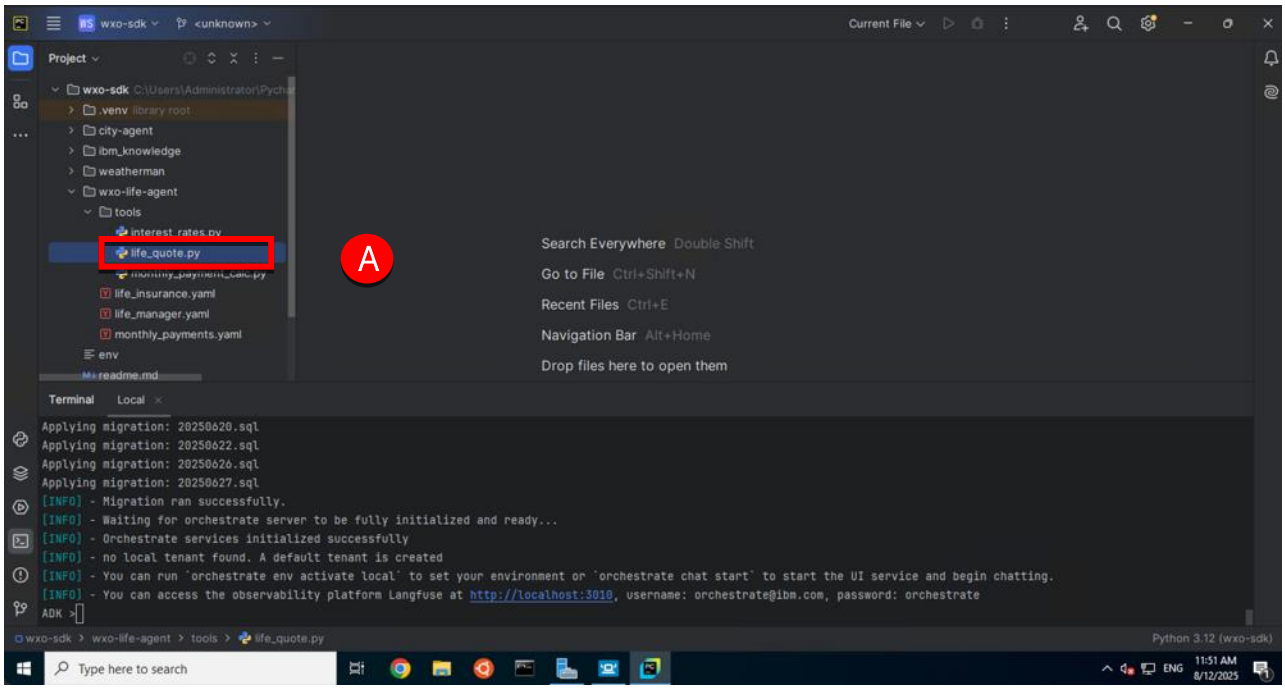
1. Double-click the **wxo-life-agent** folder (A).



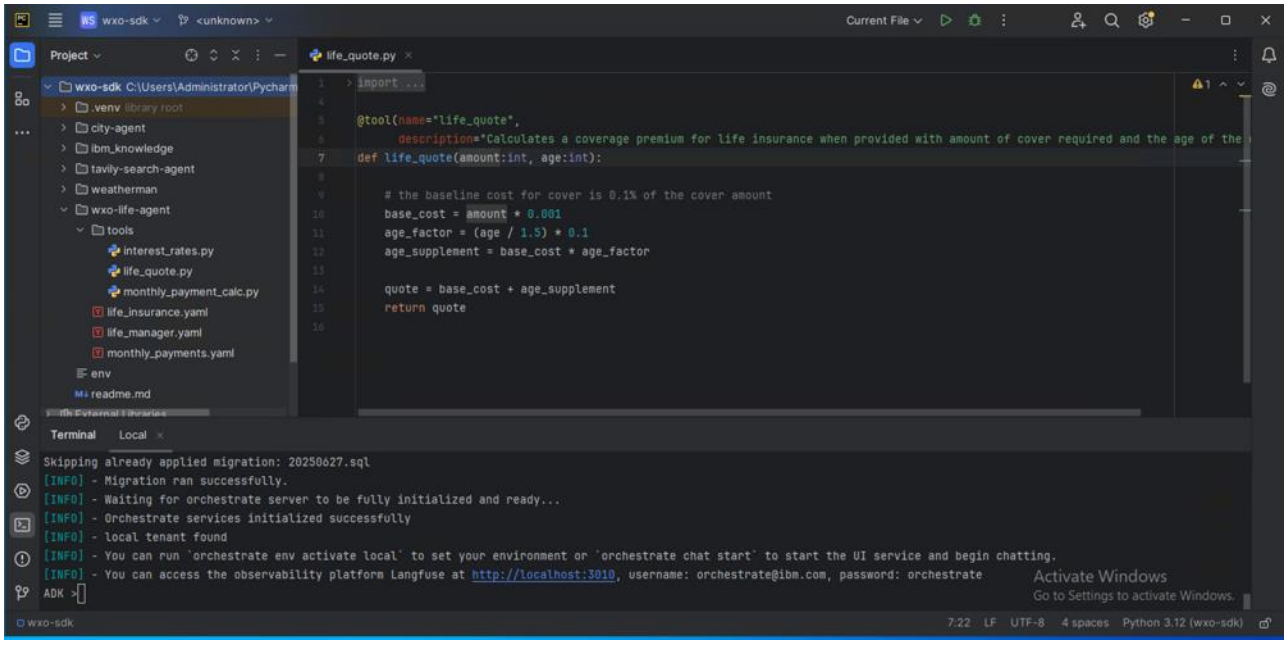
2. Double-click the **tools** folder (A).



3. Double-click the **life_quote.py** file (A).



4. Review the implementation of the Python tool (this is explained below).

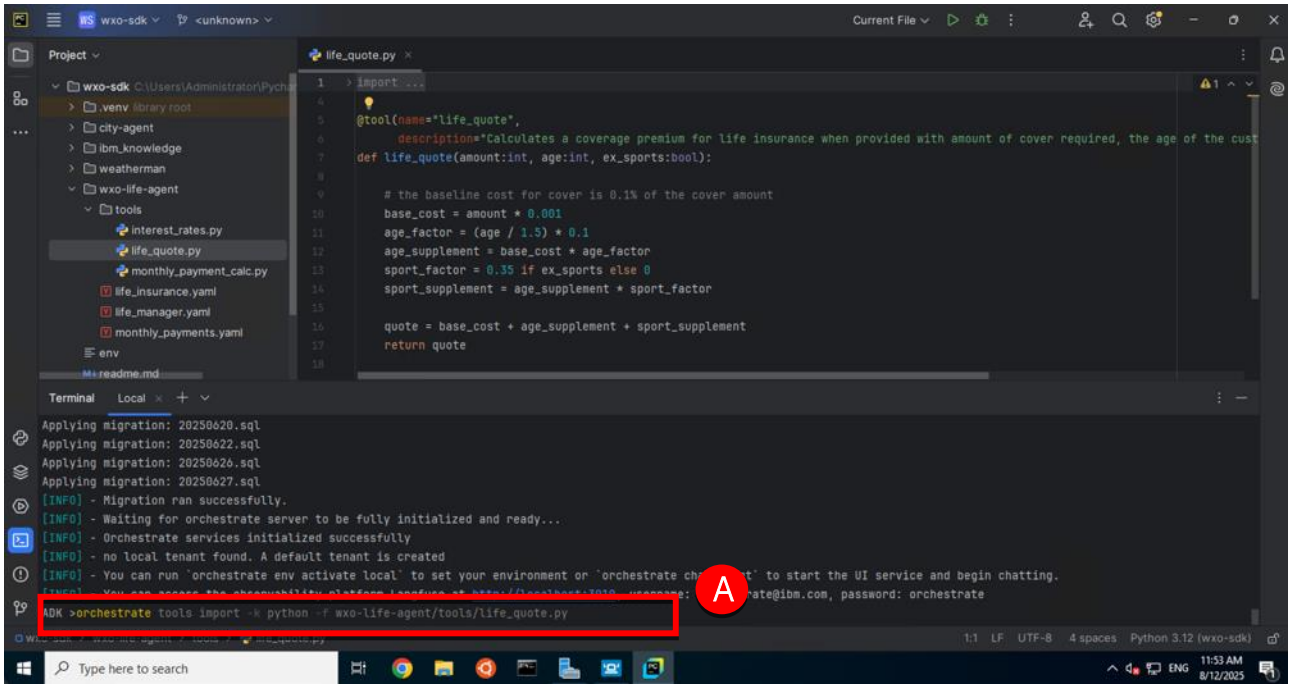


The **@tool** annotation on line 5 identifies this function as an Orchestrate tool, this annotation also provides the **name** of the tool and a **description** of its capabilities. The Python function starts on line 7 and implements a simplistic quotation calculation based on 0.1% of the amount of cover with supplements added for age.

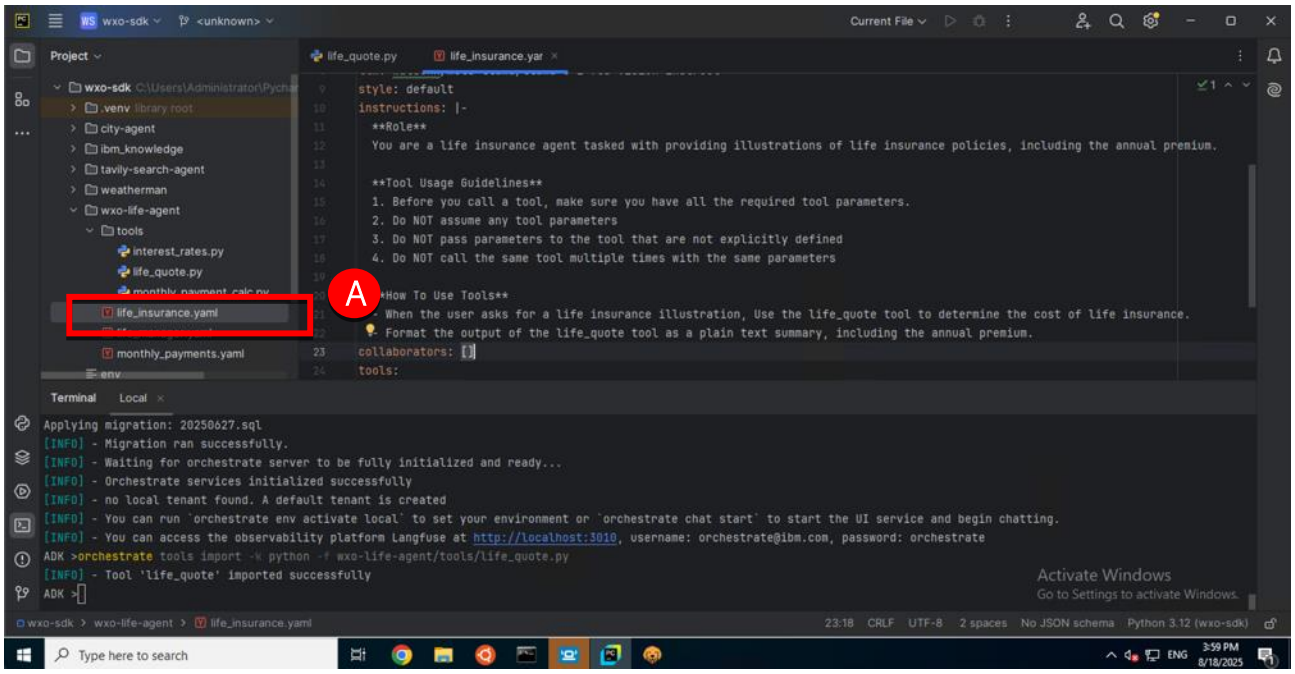
The function receives two parameters, the **amount** of cover required, and the **age** of the customer. When the tool is created, information about these parameters will be added to the configuration and available to any agent using this tool.

5. Enter the following command into the terminal (A) to create the tool.

```
orchestrate tools import -k python -f wxo-life-agent/tools/life_quote.py
```



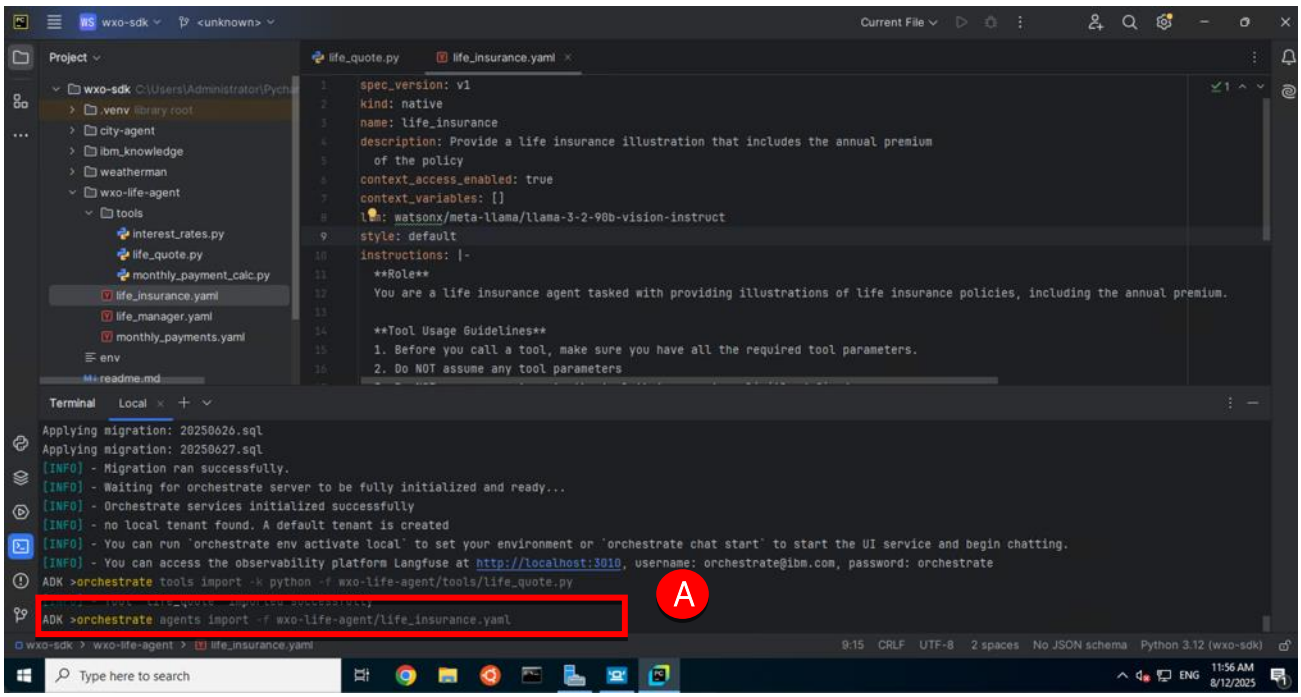
6. Double-click **life_insurance_.yaml** (A) and review the definition of the agent.



This yaml file defines the agent, its description behavior/instructions, and the LLM it will use for planning and reasoning. This agent has no collaborator, but a single tool is declared.

7. Enter the following command into the terminal (A) to create the agent.

```
orchestrate agents import -f wxo-life-agent/life_insurance.yaml
```

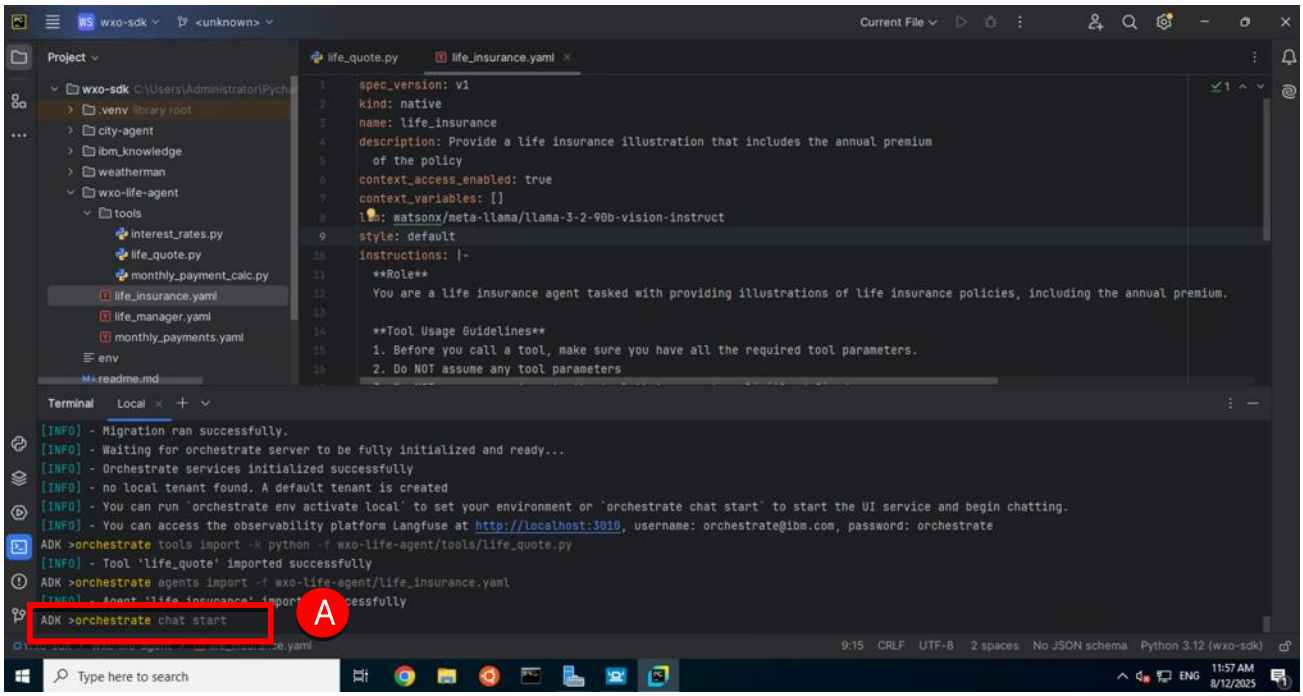


Agent testing

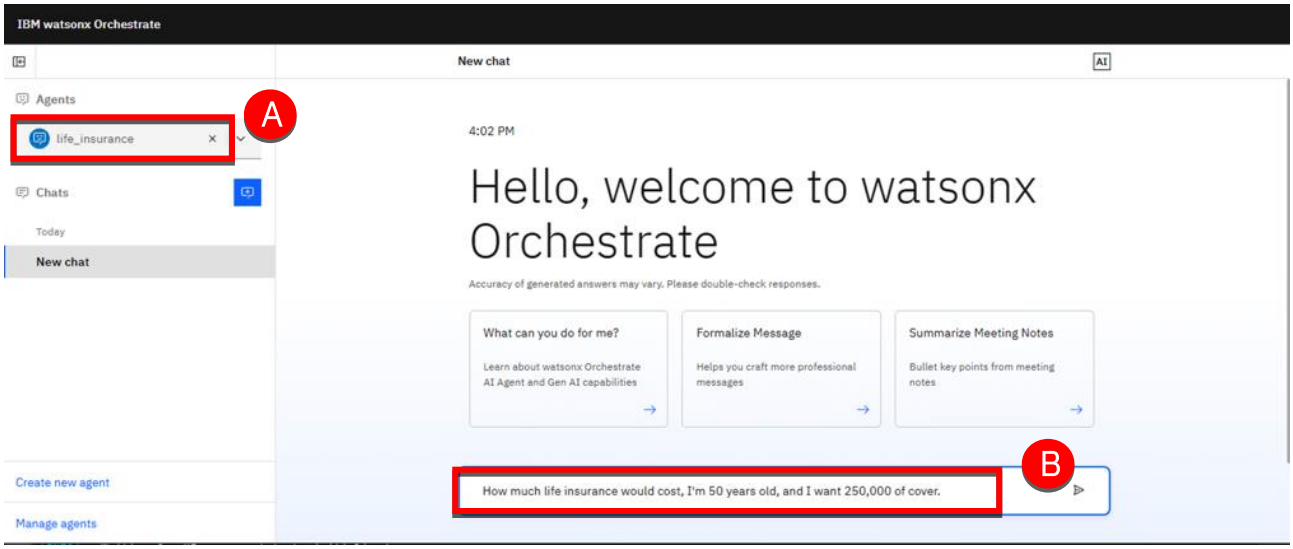
In this section you will start the chat interface, test the agent and review its reasoning in the chat UI.

1. Enter the following command into the terminal (A) to start the chat server.

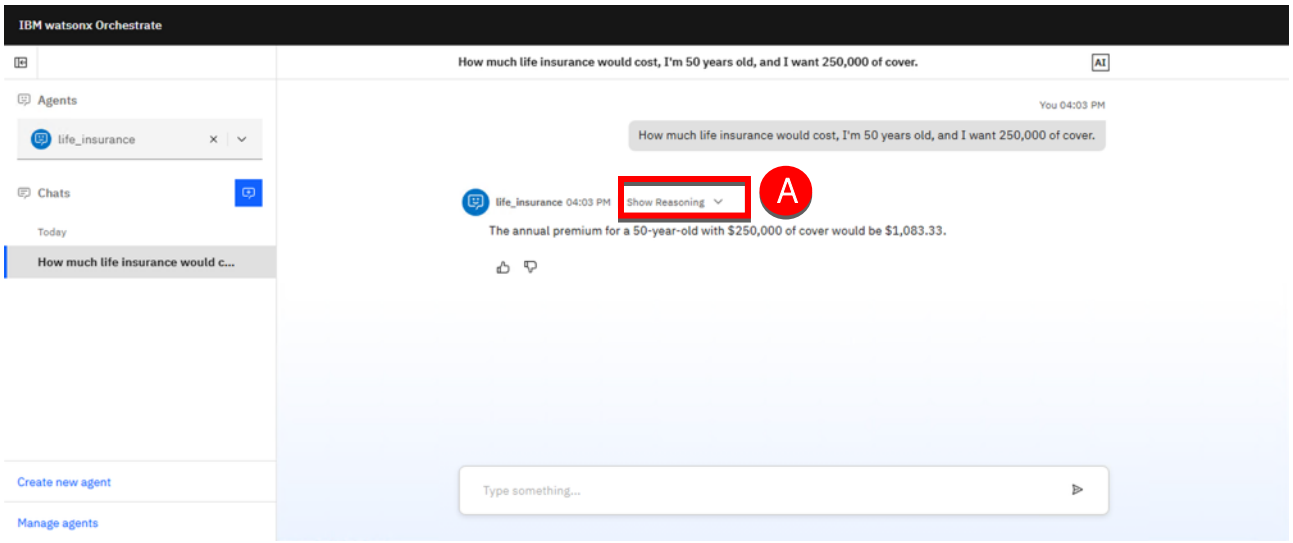
```
orchestrate chat start
```



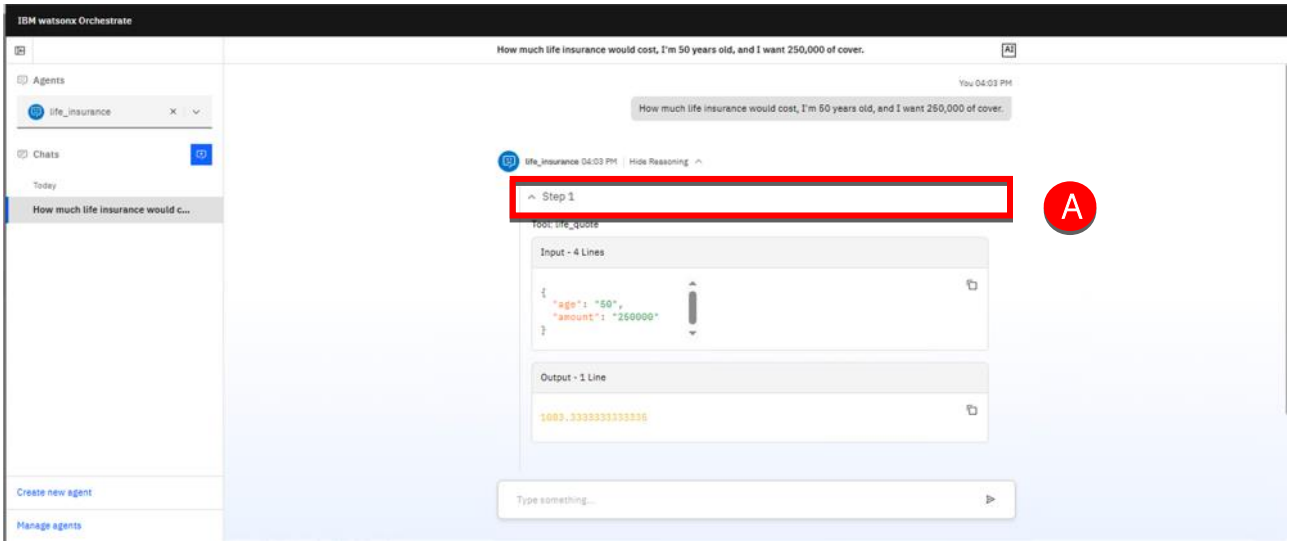
2. When the browser opens, ensure **life_insurance** is selected (A) then enter "`How much life insurance would cost, I'm 50 years old, and I want 250,000 of cover.`" into the chat window (B), then press Enter. If the agent is not available, refresh your browser.



The user input is interpreted as a request for a quotation for life insurance. When the agent responds with a quotation, expand the **Show Reasoning** panel (A).



3. Expand the **Step 1** panel (A).

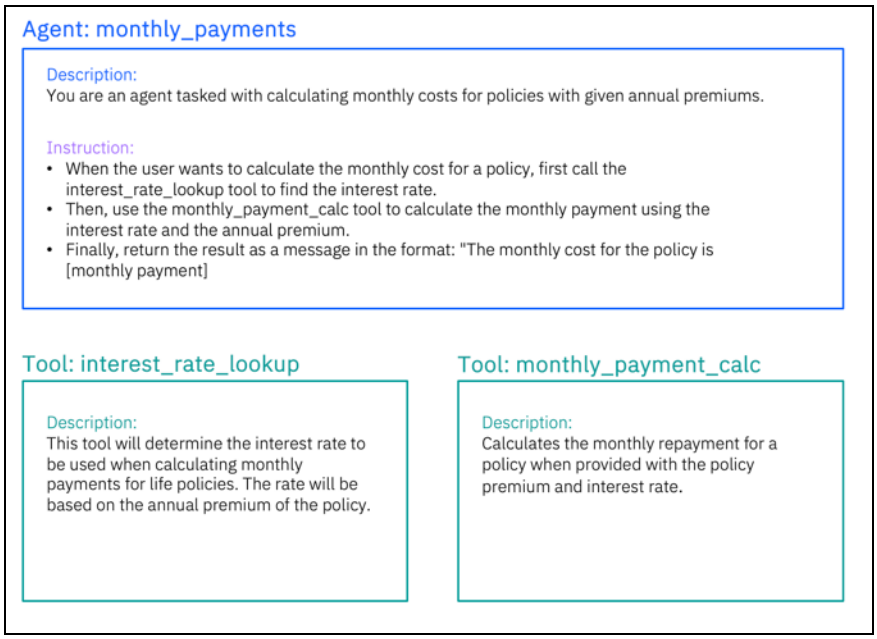


The reasoning panel shows the input and output parameters passed between the agent and the tool.

Monthly payment agent

The solution will now be extended. You will create a second agent that specializes in creating payment illustrations should a customer want to pay for their policies in monthly instalments. This second agent will use two tools, to do this, first, the interest_rate_lookup tool will be used to determine the amount of interest to be charged, then the monthly_payment_calc tool will be used to calculate the monthly payment.

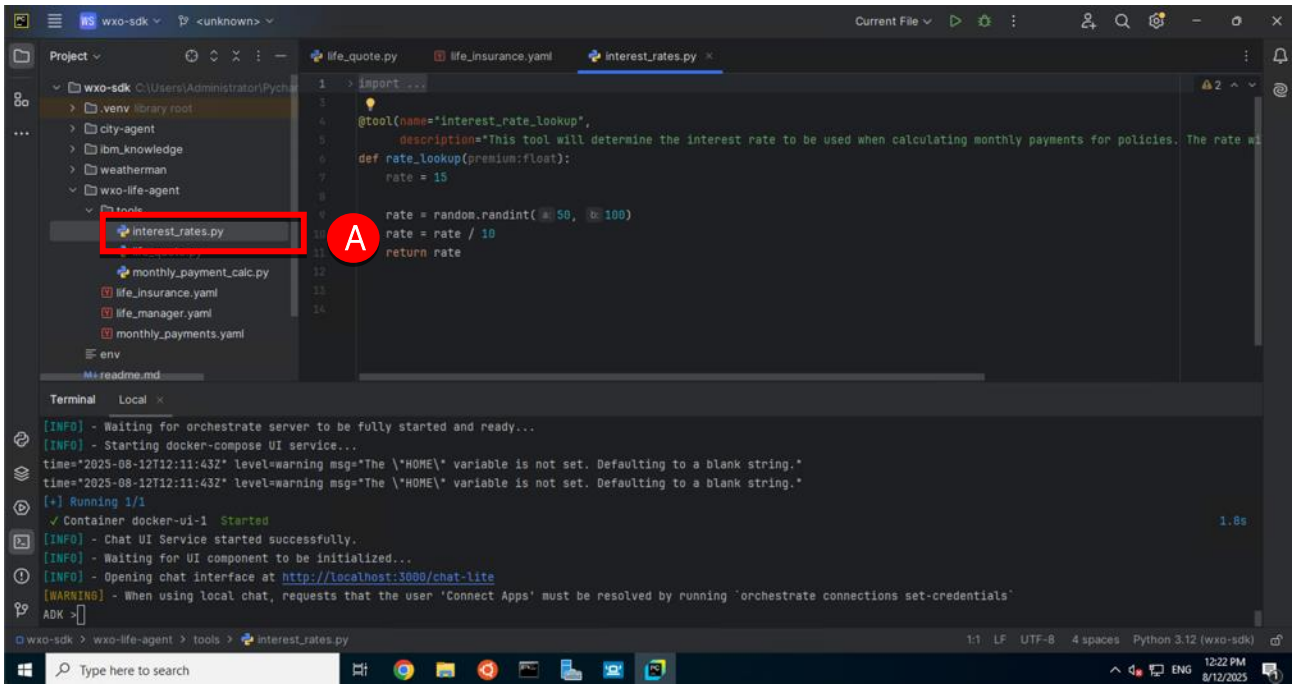
The diagram below provides an overview of the new agent configuration.



Tool and agent creation

In this section you will review the Python code used by the tools and create them in watsonx Orchestrate using the command line interface.

1. In PyCharm, double-click **interest_rates.py** (A).

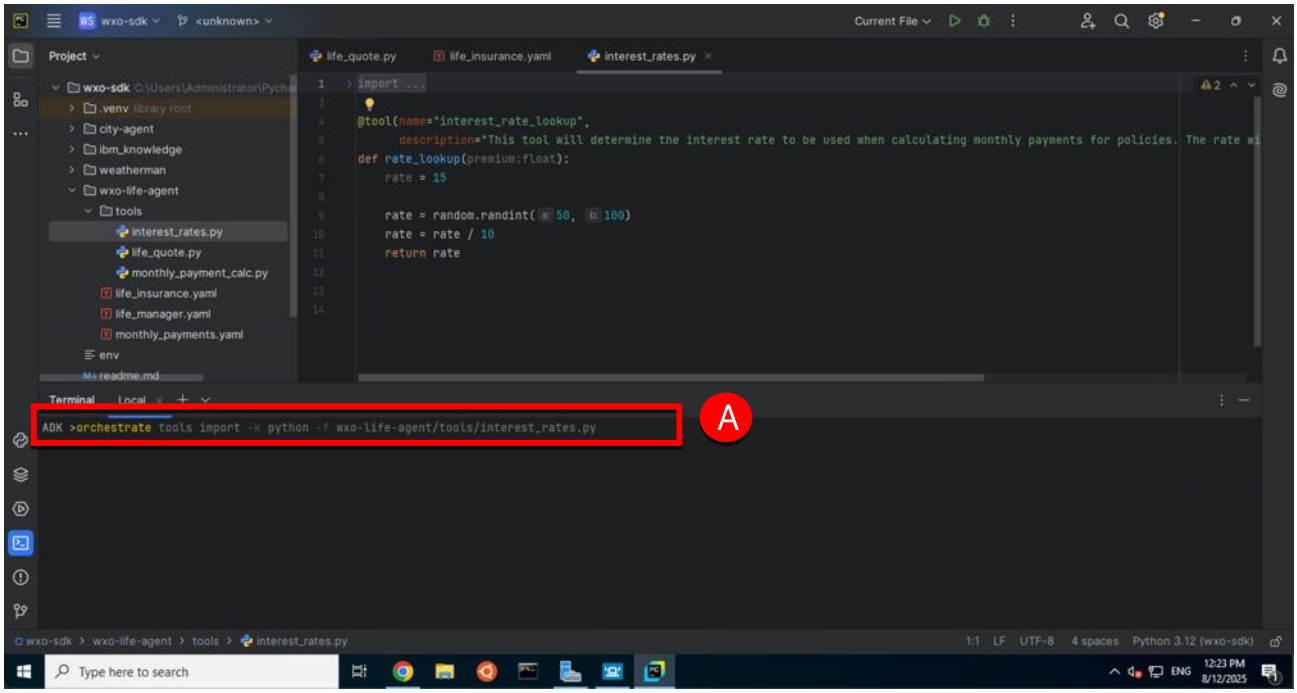


The @tool annotation on line 4 identifies the function on line 6 as an Orchestrate tool. This annotation also provides the **name** of the tool and a **description** of its capabilities. The Python function generates a random interest rate lookup between 5-10%. The policy premium is passed as a parameter into the function, but it is not used.

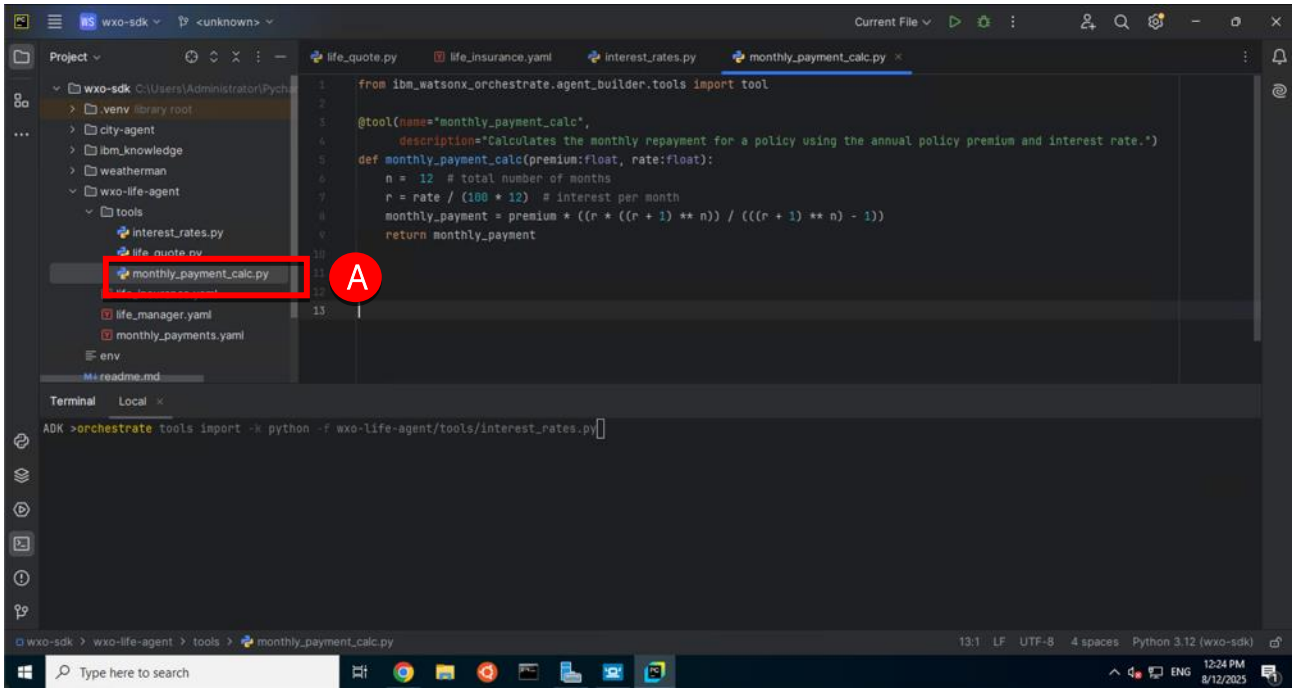
This function is simply a random number generator, and as such, the calculations performed using its response will vary.

2. Enter the following command into the terminal (A) to import the interest rate lookup tool.

```
orchestrate tools import -k python -f wxo-life-agent/tools/interest_rates.py
```



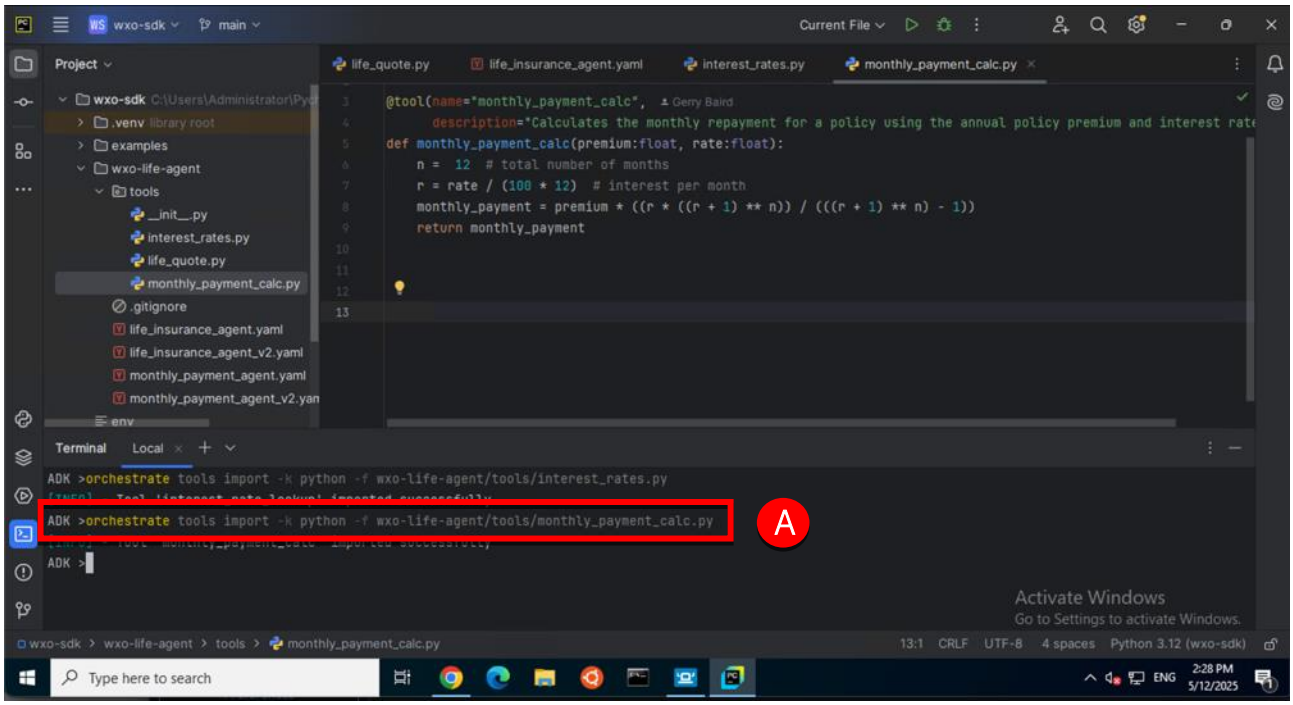
3. Double-click **monthly_payment_calc.py** (A).



This function calculates a monthly payment based on the premium and interest rate.

4. Enter the following command into the terminal (A) to import the interest rate lookup tool.

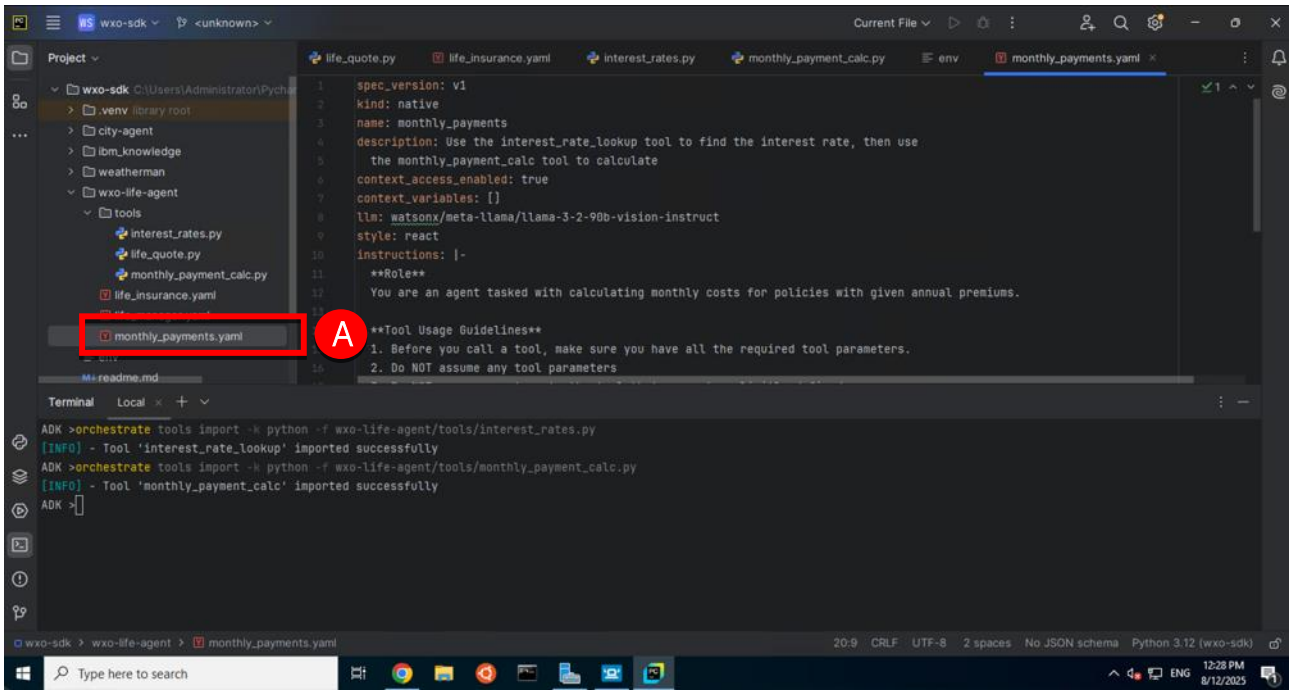
```
orchestrate tools import -k python -f wxo-life-
agent/tools/monthly_payment_calc.py
```



Agent creation and testing

In this section you will review the definition of the agent and create it in watsonx Orchestrate using the command line interface. Once the agent has been created, you will test the agent and review its reasoning.

5. Double-click **monthly_payments. yaml** (A) and review the definition of the agent.



This yaml file defines the agent, its description and behavior, and the LLM it uses for planning and reasoning.

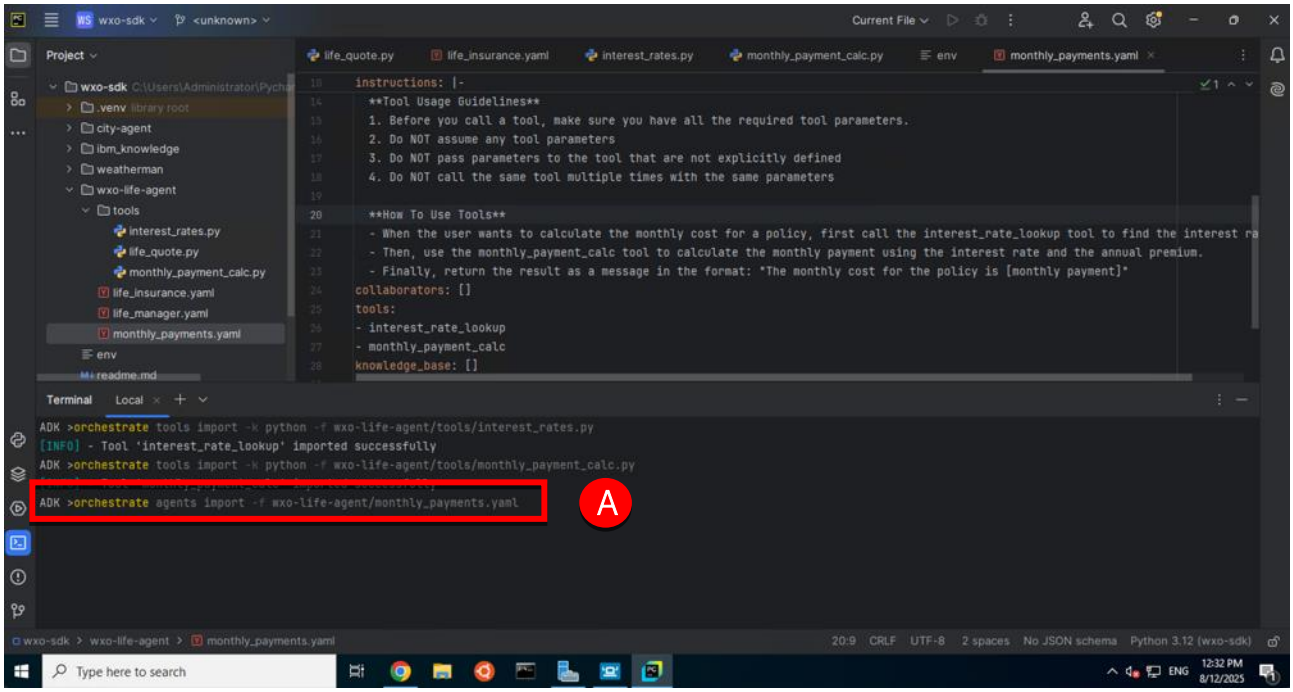
Most LLM's will try and generate an approximate monthly payment. This approximate calculation may appear reasonable, but it won't be based on an accurate amortization formula like the one used in the tool. As LLM's become more capable, identifying these hallucinations is increasingly difficult.

This agent includes additional instructions on how to use its tools correctly and format the response.

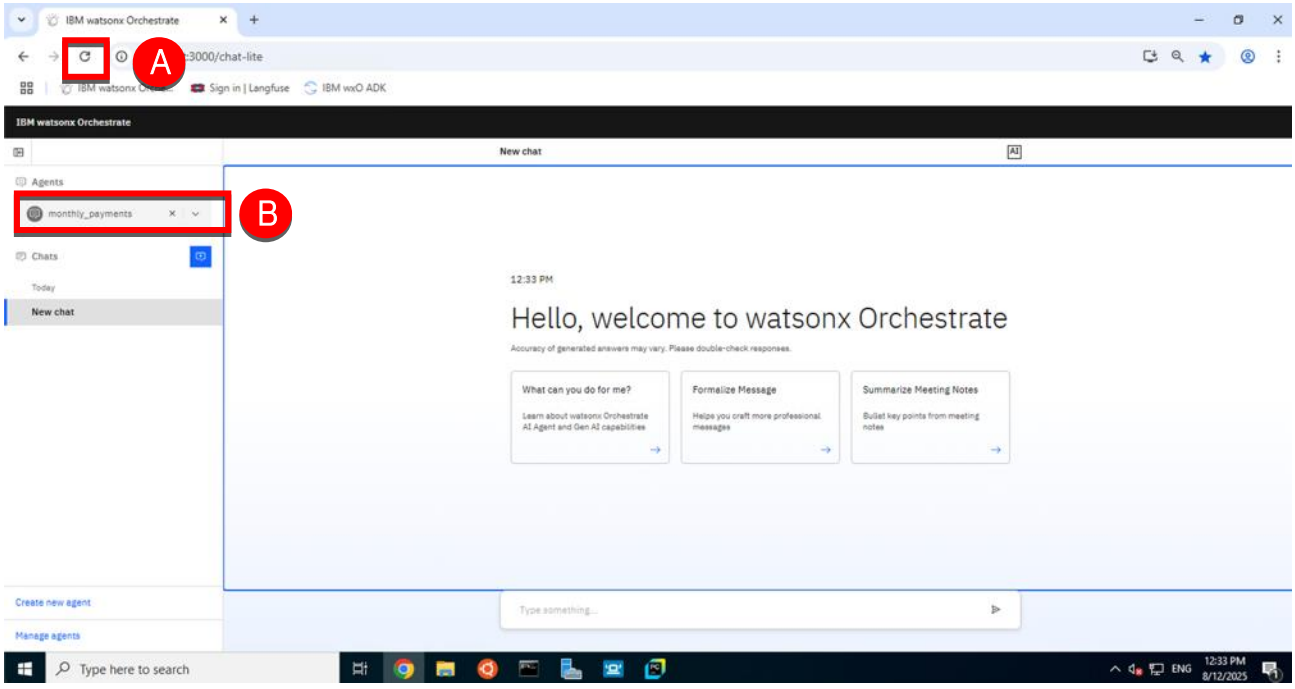
```
instructions: |-
  **Role**
  You are an agent tasked with calculating monthly costs for policies with given annual premiums.

  **Tool Usage Guidelines**
  1. Before you call a tool, make sure you have all the required tool parameters.
  2. Do NOT assume any tool parameters
  3. Do NOT pass parameters to the tool that are not explicitly defined
  4. Do NOT call the same tool multiple times with the same parameters

  **How To Use Tools**
  - When the user wants to calculate the monthly cost for a policy, first call the interest_rate_lookup tool to find the interest rate.
  - Then, use the monthly_payment_calc tool to calculate the monthly payment using the interest rate and the annual premium.
  - Finally, return the result as a message in the format: "The monthly cost for the policy is [monthly payment]"
```

6. Enter the following command into the terminal (A) to create the agent.

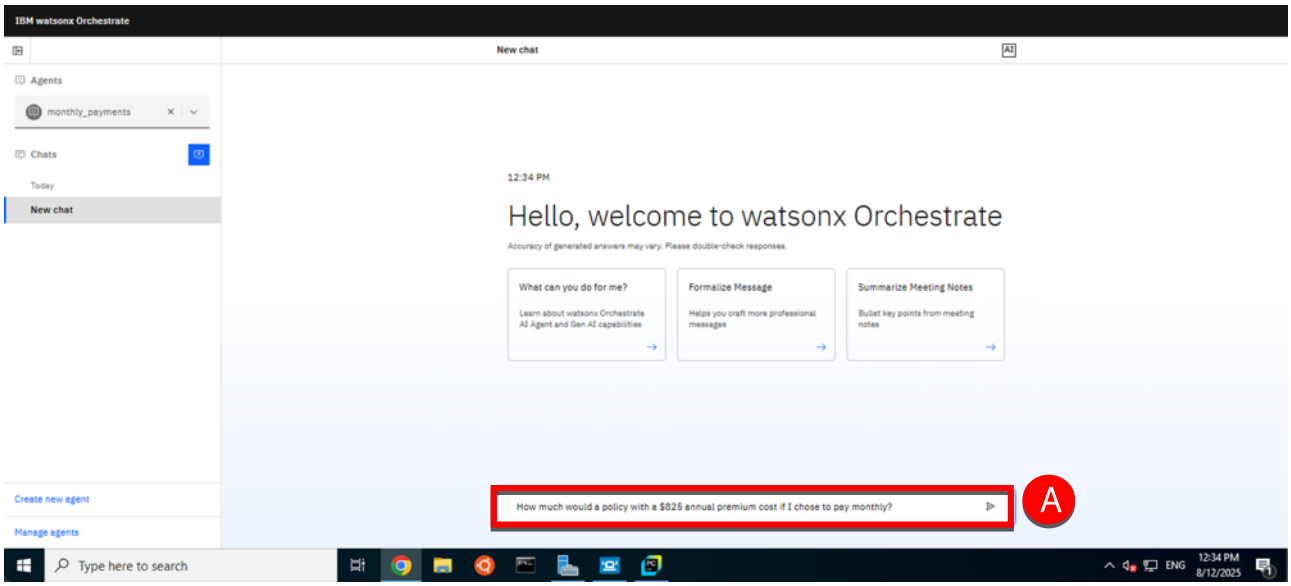
```
orchestrate agents import -f wxo-life-agent/monthly_payments.yaml
```



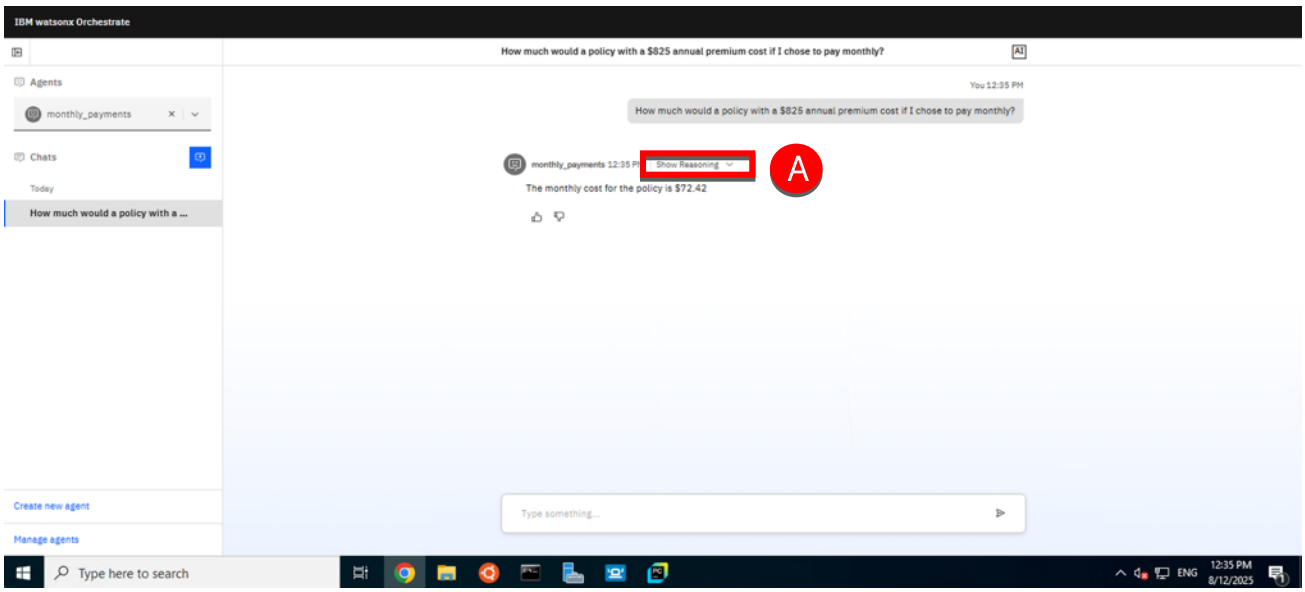
7. In the browser window click the refresh icon (A) then select **monthly_payments** from the **Agents** list (B).



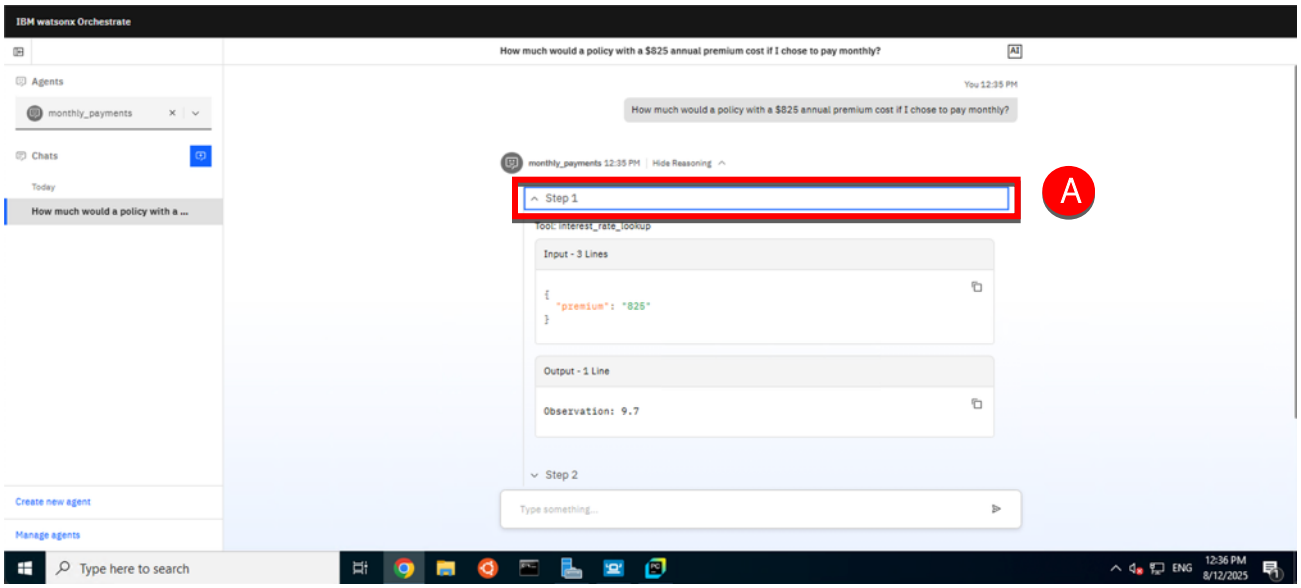
8. Enter “How much would a policy with a $825 annual premium cost if I chose to pay monthly?” into the chat window (A), then press Enter.



9. When the agent responds expand the Show Reasoning panel (A).

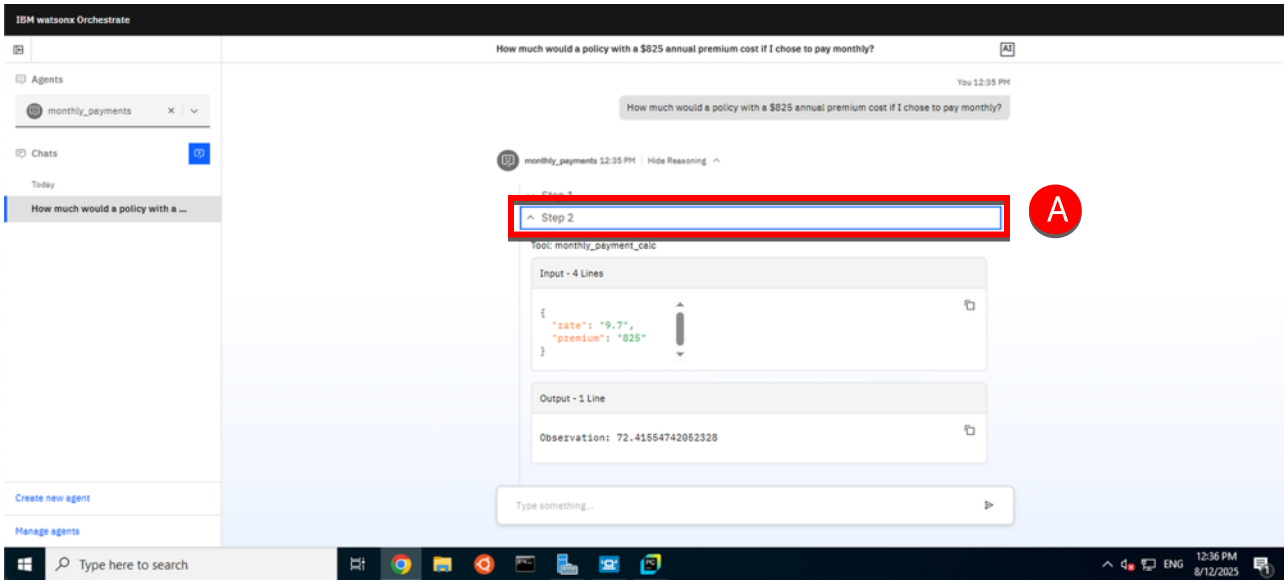


10. Expand **Step 1** (A).



The agent has begun by using the **interest_rate_lookup** tool (this is a random number).

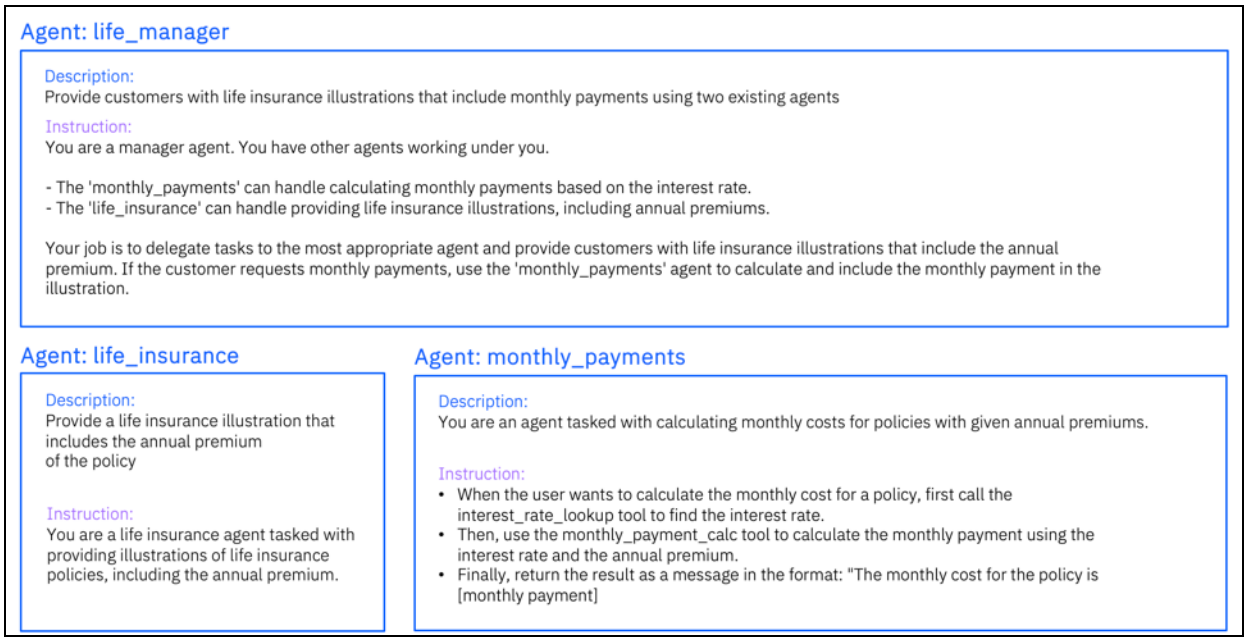
11. Expand **Step 2** (A).



The agent has used the **monthly_payment_calc** tool to determine the monthly payment based on the premium and interest rate and formatted the response according to the instructions.

Collaborating agents

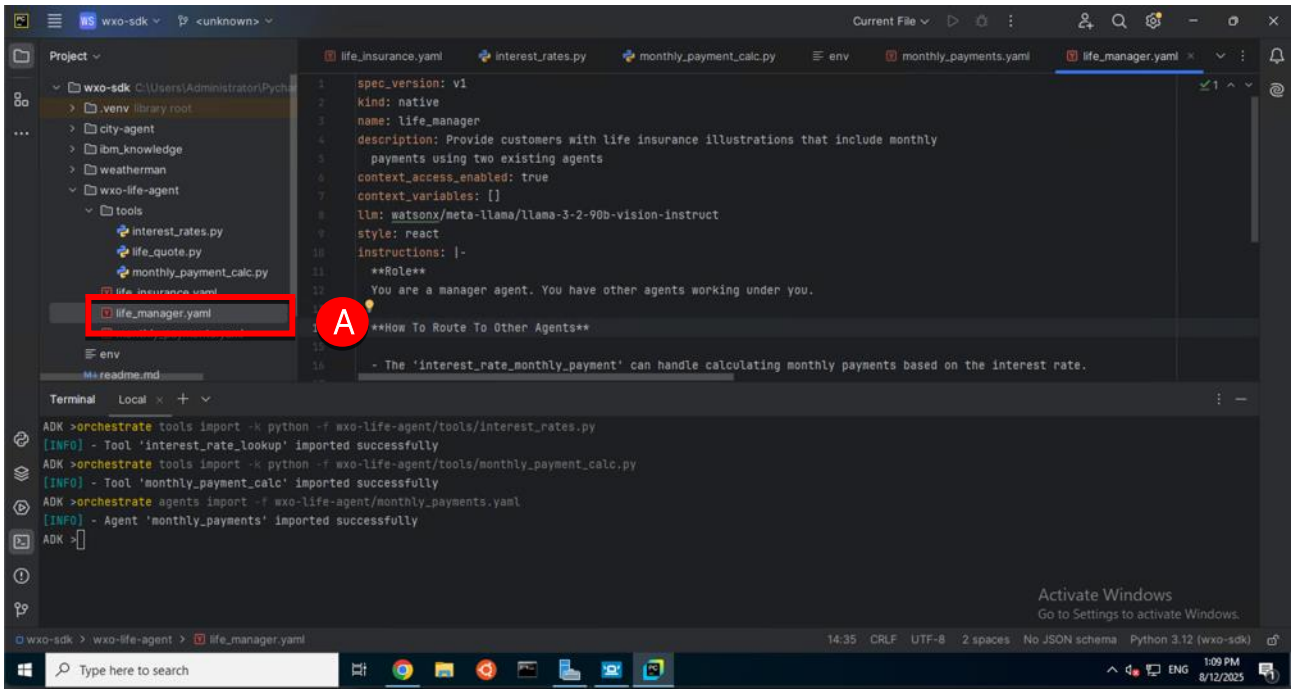
Next you will add a further agent that is able to collaborate with the two existing agents to answer more complex enquires that involve a life cover quotation, combined with monthly payments. The diagram below provides an overview of the expanded configuration used in this section. Tools used by the first two agents have been omitted.



Manager agent

In this section you will review the new agent definition and import it into watsonx Orchestrate using the command line interface.

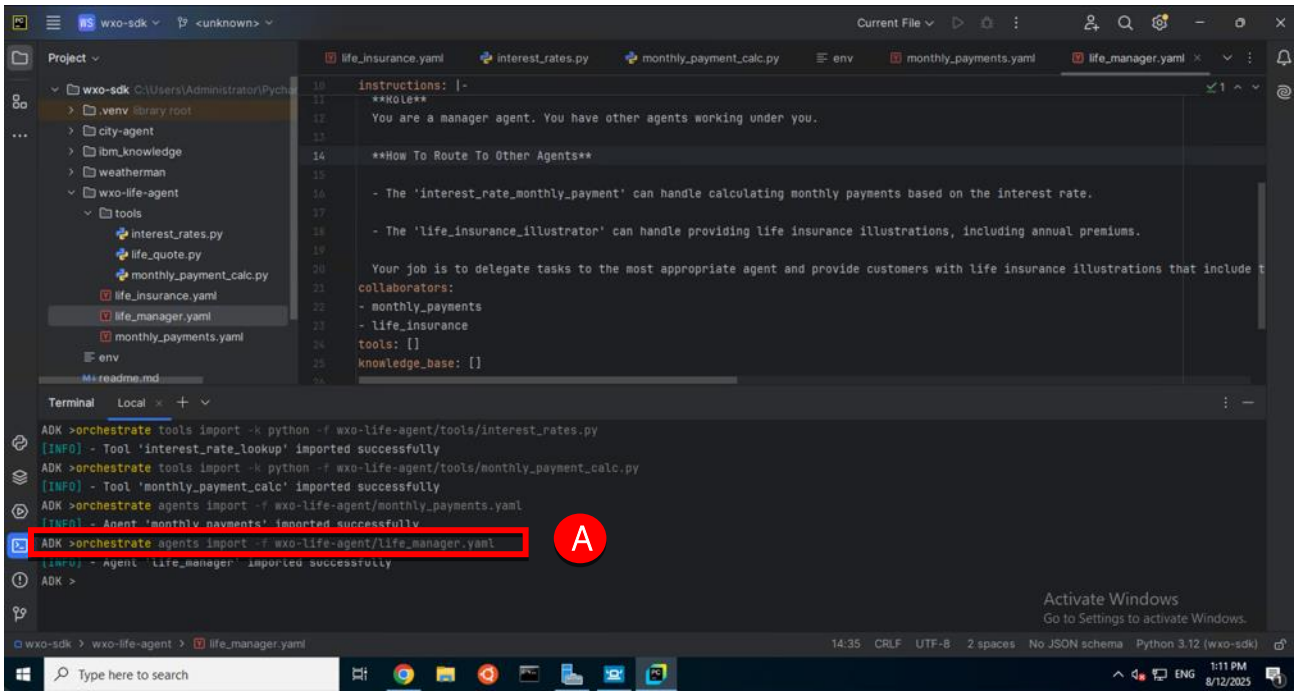
1. Double-click **life_manager** (A).



Review the agent definition and note the addition of a collaborator agents.

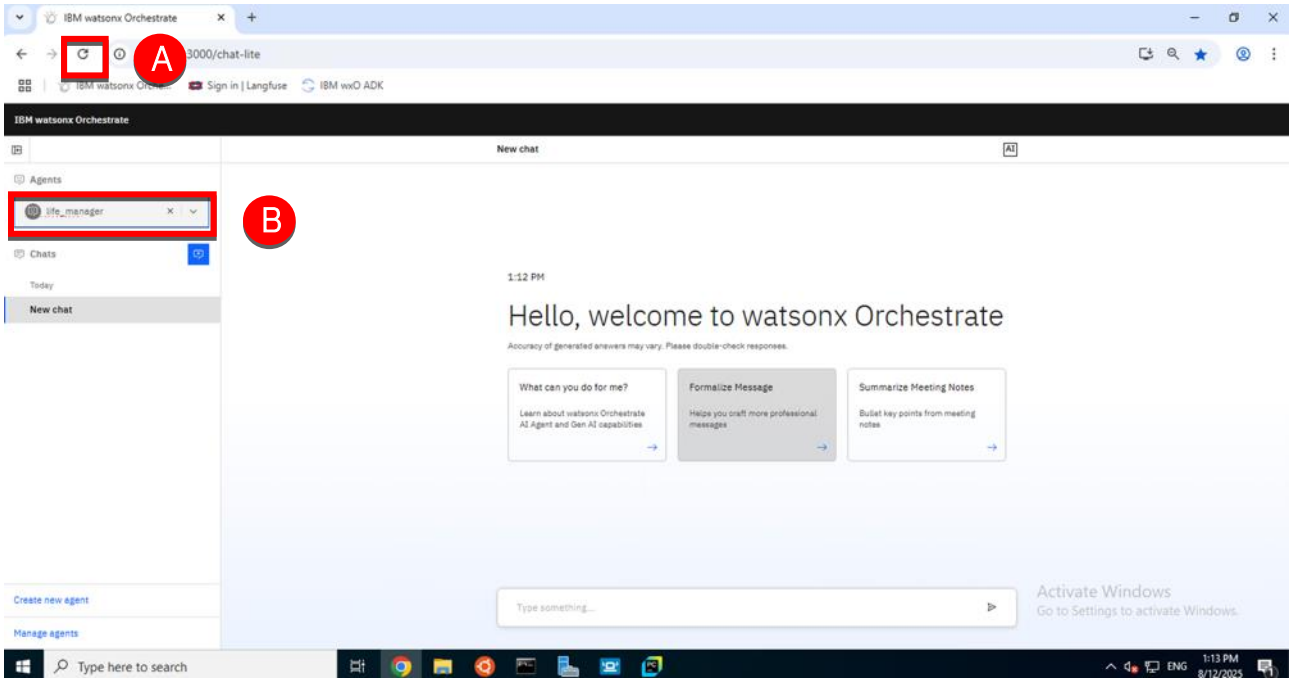
2. Enter the following command into the terminal (A) to add the agent.

```
orchestrate agents import -f wxo-life-agent/life_manager.yaml
```

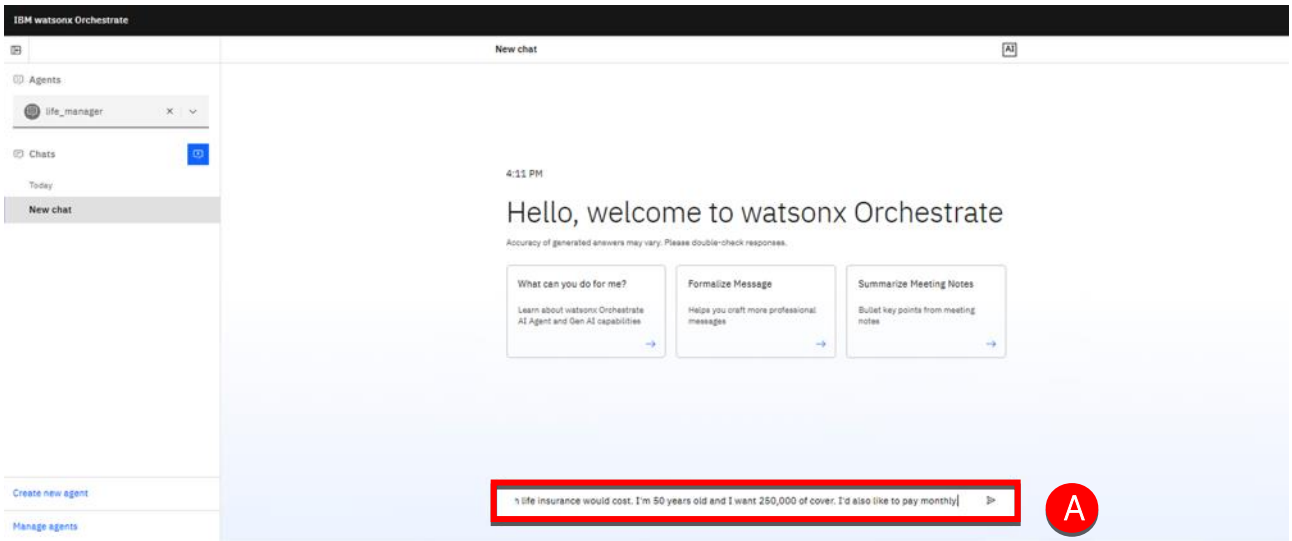


Testing the manager agent

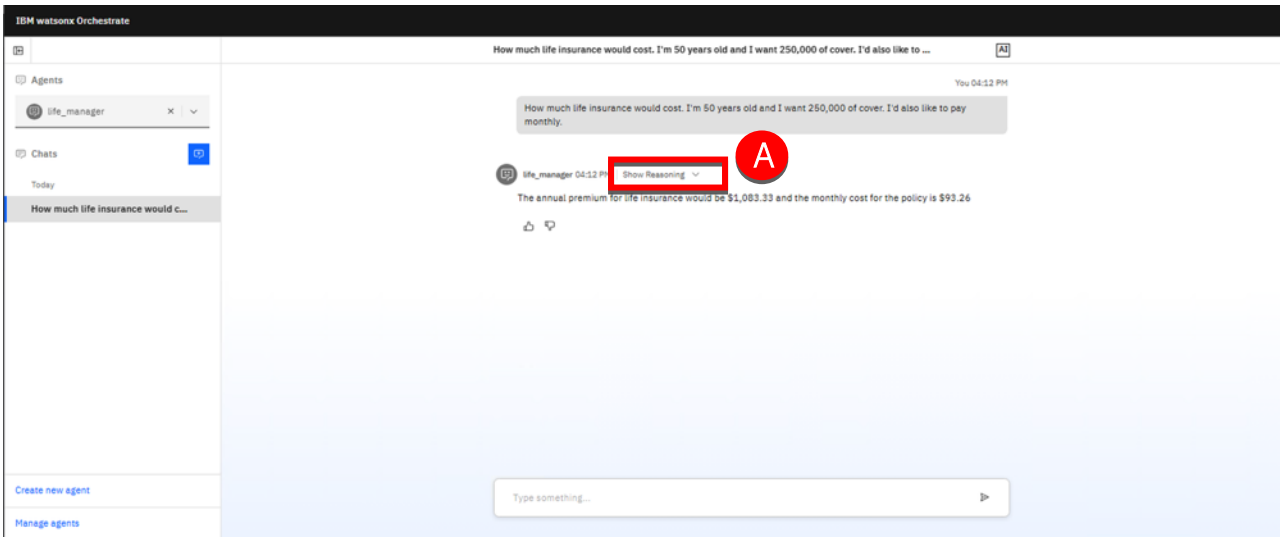
1. Return to the browser window and click refresh (A) and ensure the **life_manager** is selected from the **Agents** list (B).



2. Enter "`How much life insurance would cost. I'm 50 years old and I want 250,000 of cover. I'd also like to pay monthly.`" into the chat window (A), then press Enter.



3. When the agent responds, expand the **Show Reasoning** panel (A).



The reasoning for this request has required several steps that are summarized below:

- Step 1: Transfer to agent: life_insurance
- Step 2: Tool call: life_quote
- Step 3: Transfer to agent: monthly_payments
- Step 4: Tool call: interest_rate_lookup
- Step 5: Tool call: monthly_payment_calc

Expand each step to review the tool usage and agent transfer.

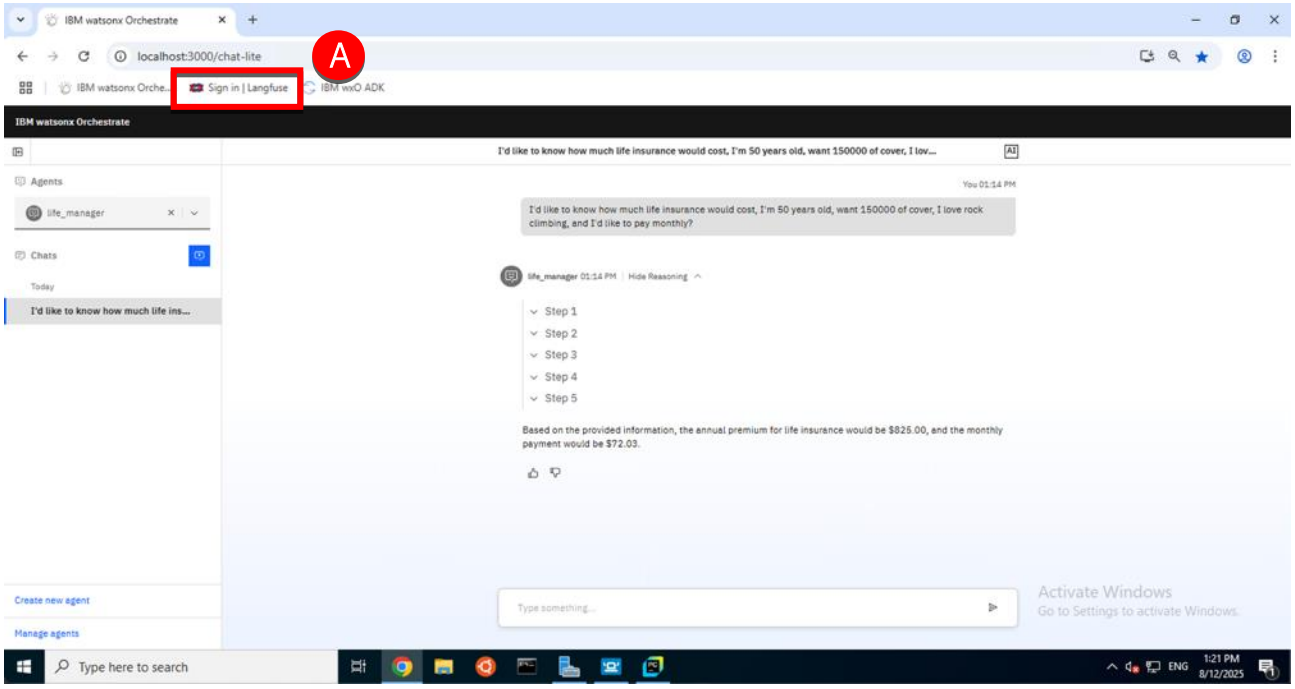
It would also have been possible to achieve the same result by adding the additional tools and instructions into a single top-level agent. However, decomposing this functionality into separate agents keeps the agents smaller, simpler, promotes reuse and should improve the reasoning accuracy of the solution.

Langfuse trace

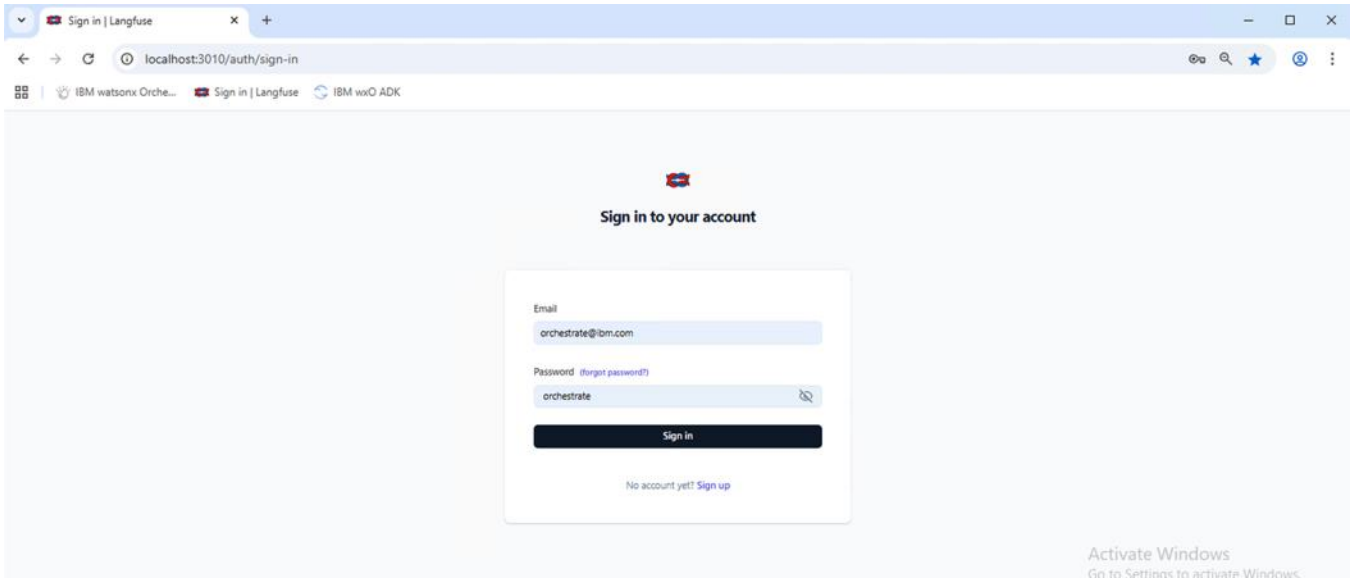
In this section you will review the trace information captured in Langfuse and understand the chain of thought used by the agent to produce the response.

Access the trace

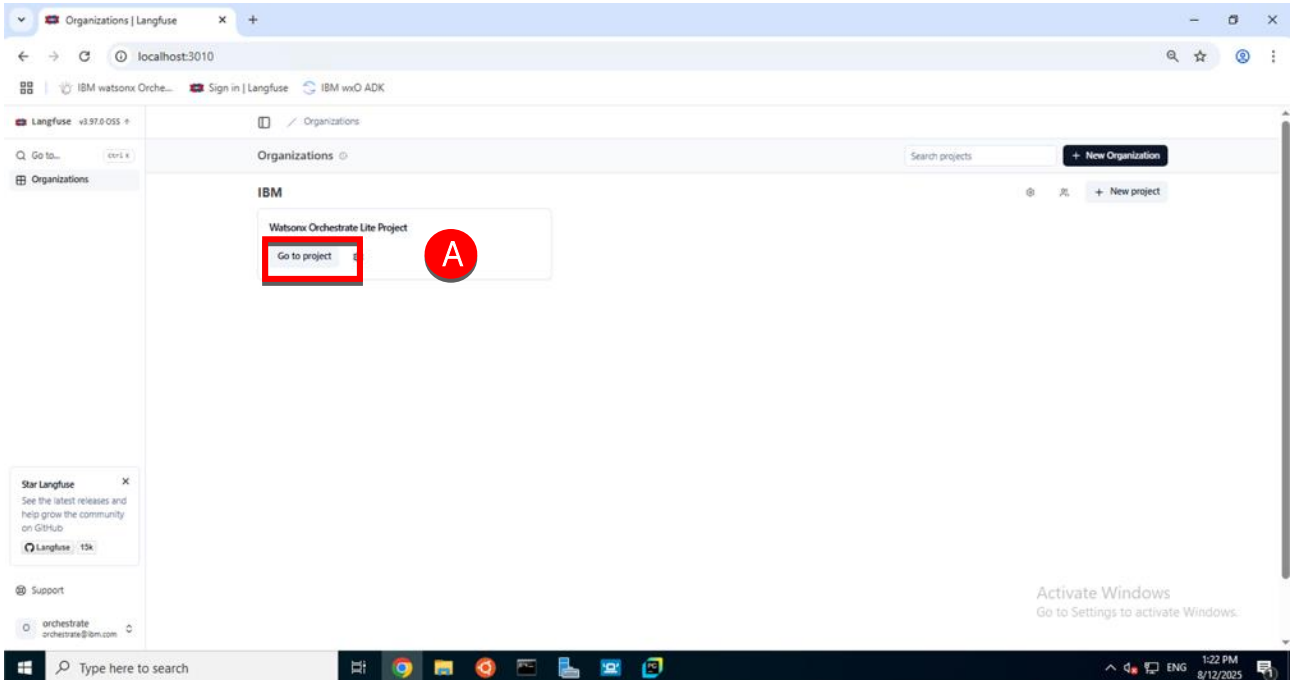
1. Click the **Sign in | Langfuse** bookmark in the browser (A).



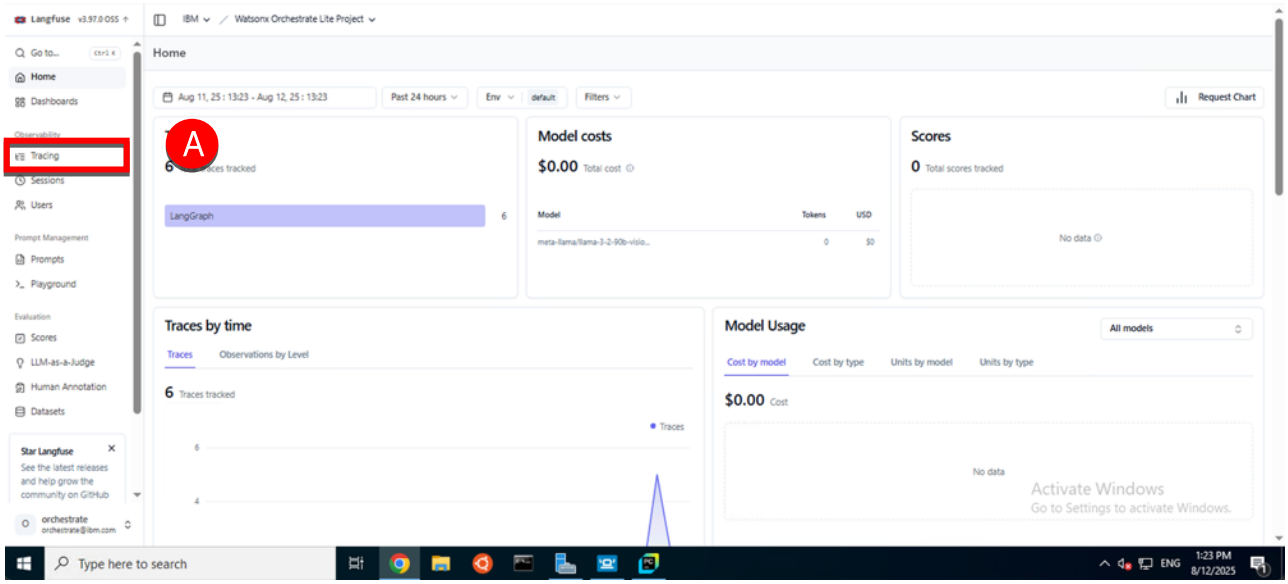
2. The login page should be pre-populated, enter credentials if needed, and click **Sign in** (A).



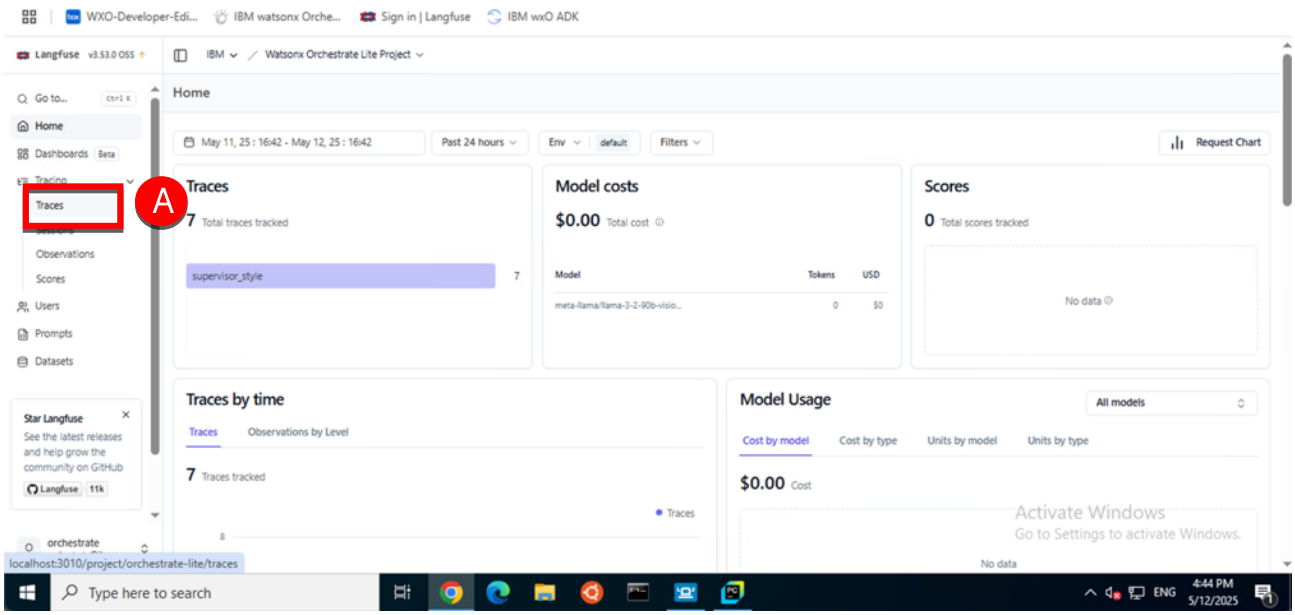
3. Click **Go to project** (A).



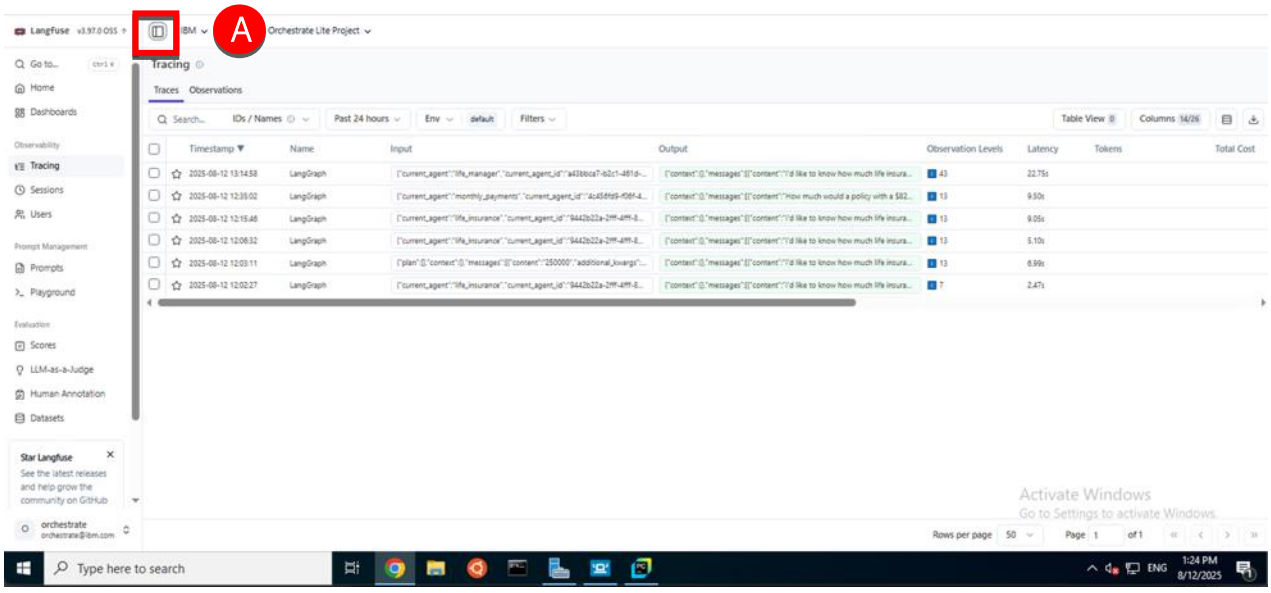
4. Click **Tracing** (A).



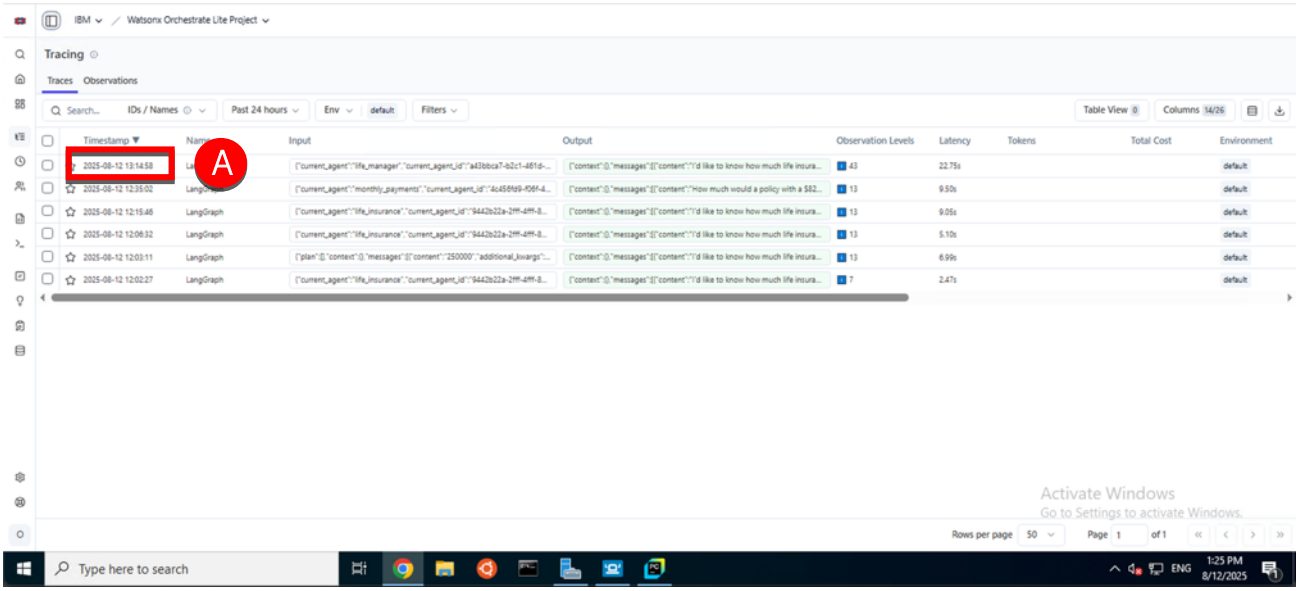
5. Click **Traces** (A).



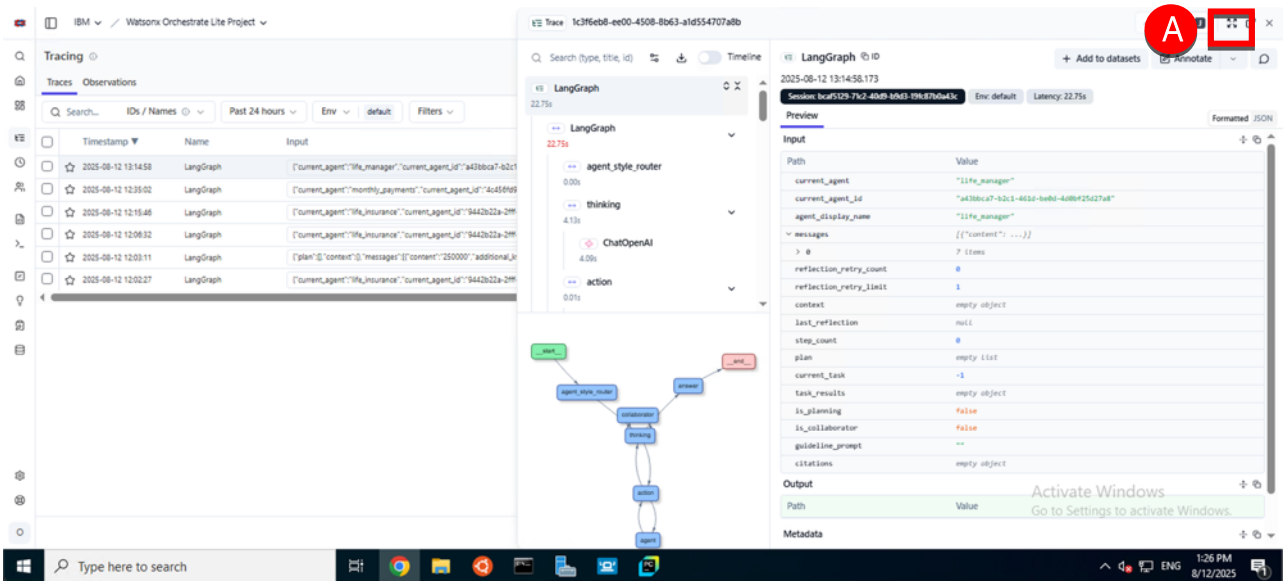
6. Click the panel icon (A) to hide the side-menu.



7. Click the Timestamp of the top trace in the table (A), this will open the trace from the last request.

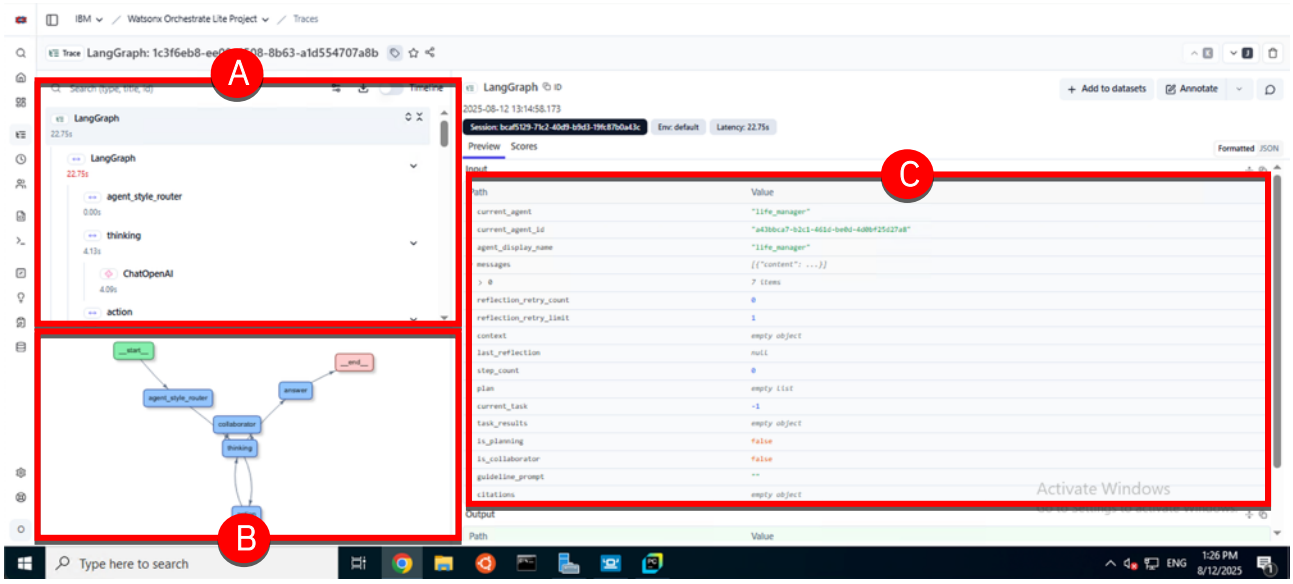


8. Click the expand icon (A), this will show the trace in full screen.

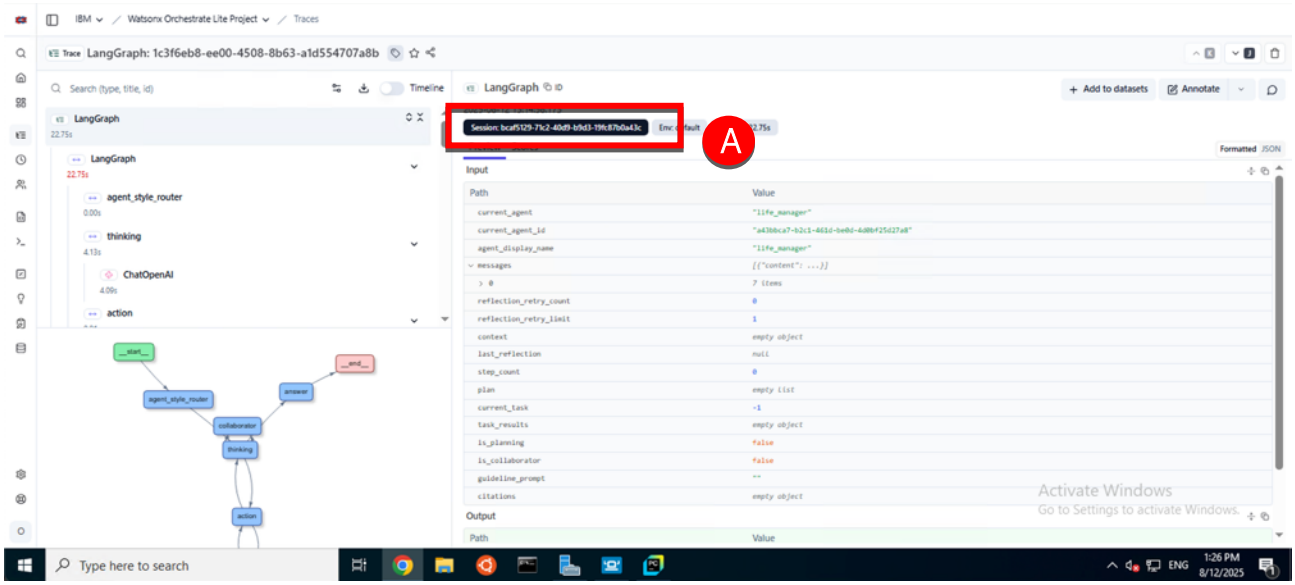


Analyze the trace

1. Review the Langfuse trace display containing the trace (A), the agent graph (B) and the trace detail (C).

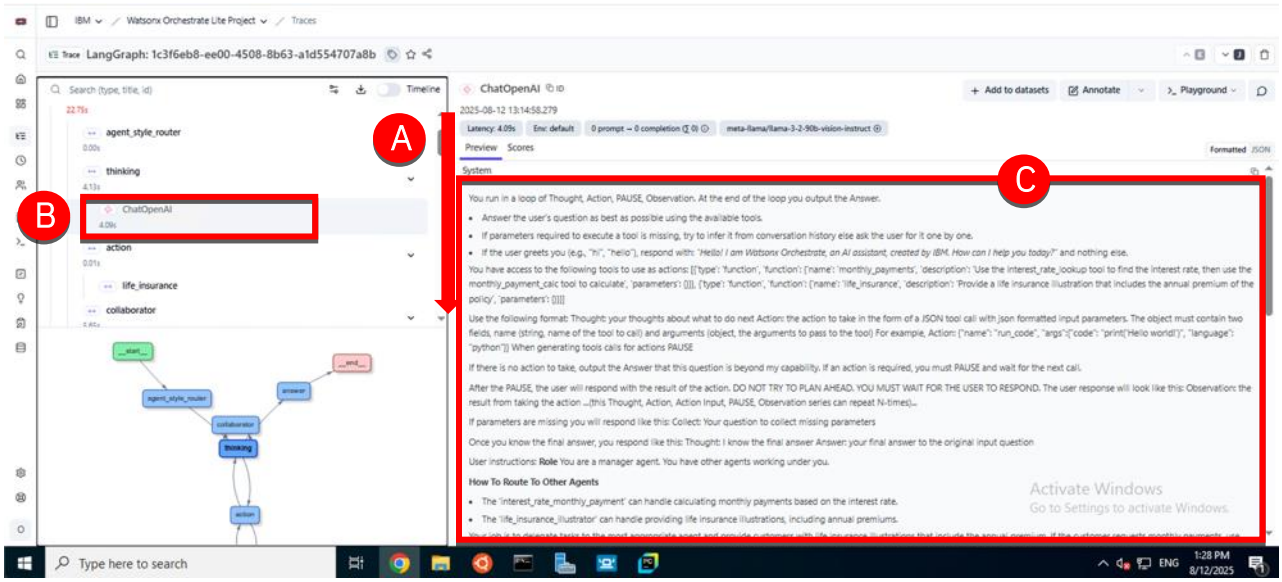


2. Note the Session ID (A).



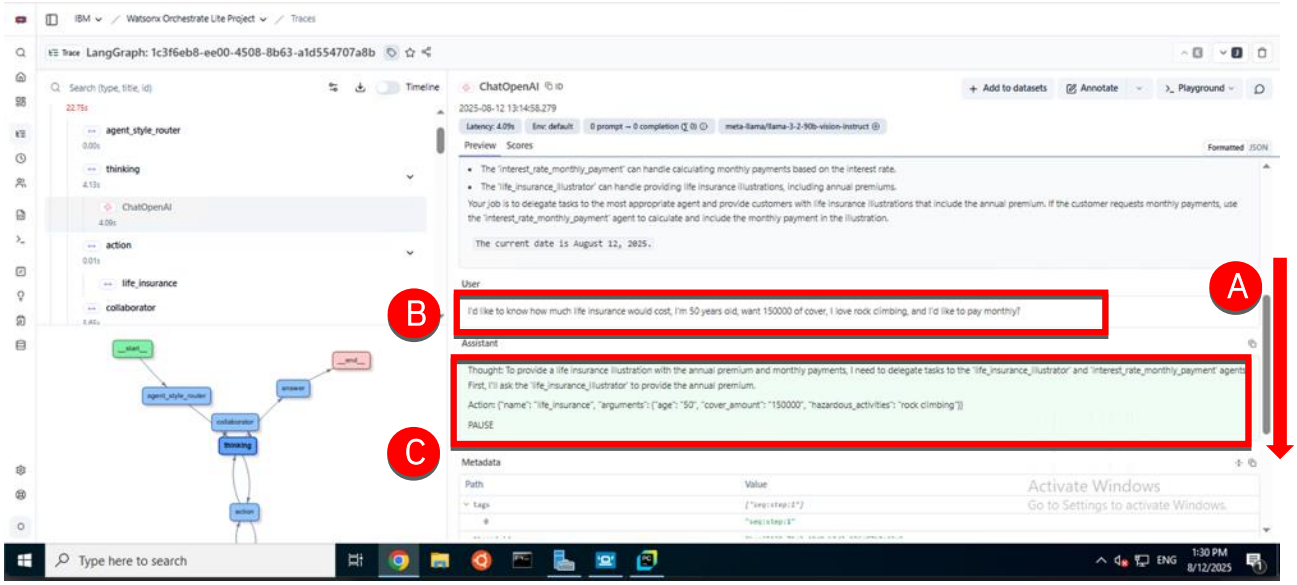
Multiple steps in the same conversation will have the same session ID. The traces are grouped by Langfuse and can be viewed together by clicking on the session ID. This makes it much easier to debug multi-step conversations. Note that if you start a new chat in Orchestrate, a new session ID is created.

3. Scroll down in the trace panel (A) and click the first entry called **ChatOpenAI** (B). The trace detail displays the **System** prompt used by the agent (C).



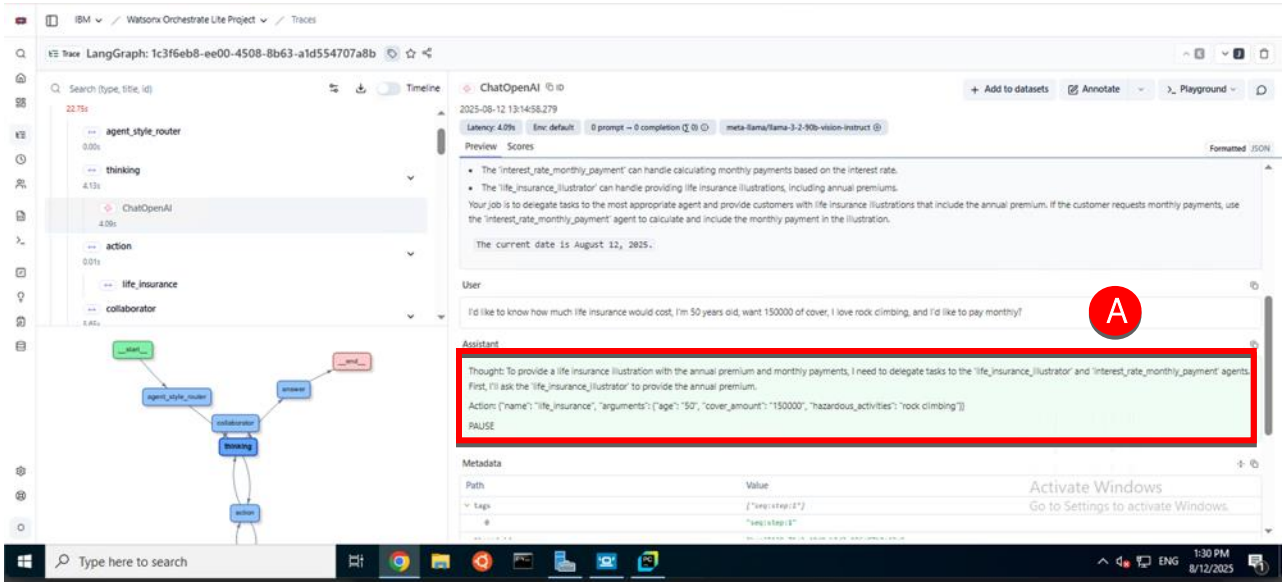
This prompt determines how the LLM used by the agent will approach the problem. In addition to the general instructions the prompt also includes information about the collaborators that are available to the agent.

4. Scroll past the prompt (A) to display the **User** panel (B) and the **Assistant** panel (C).



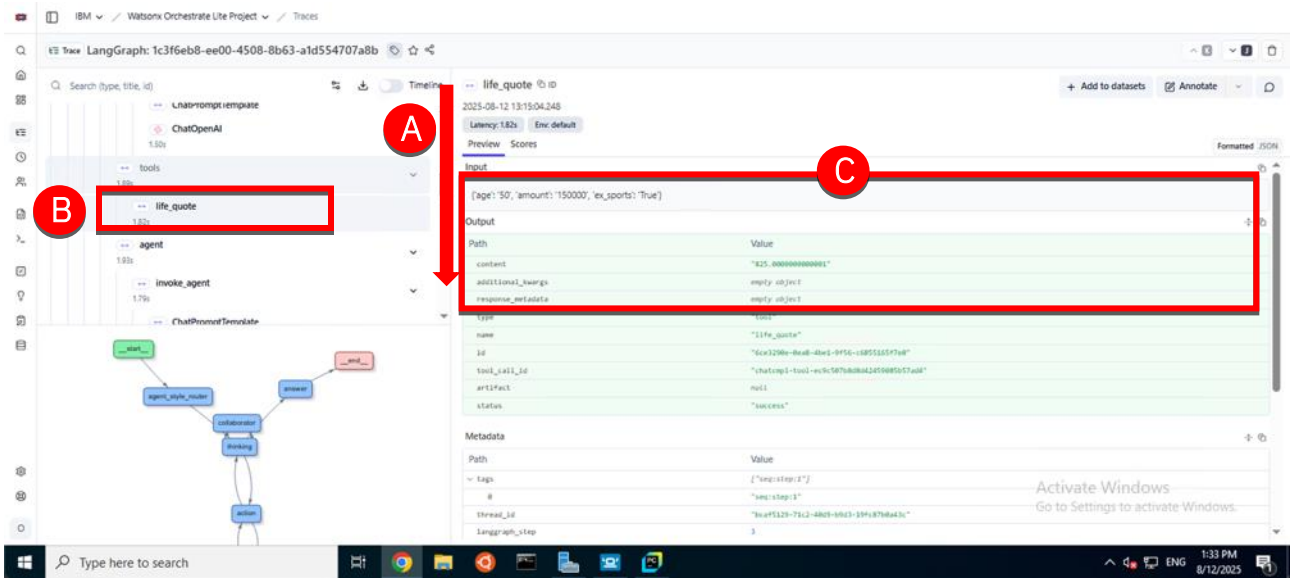
The User panel displays the text entered by the user.

5. Review the contents of the **Assistant** panel (A).

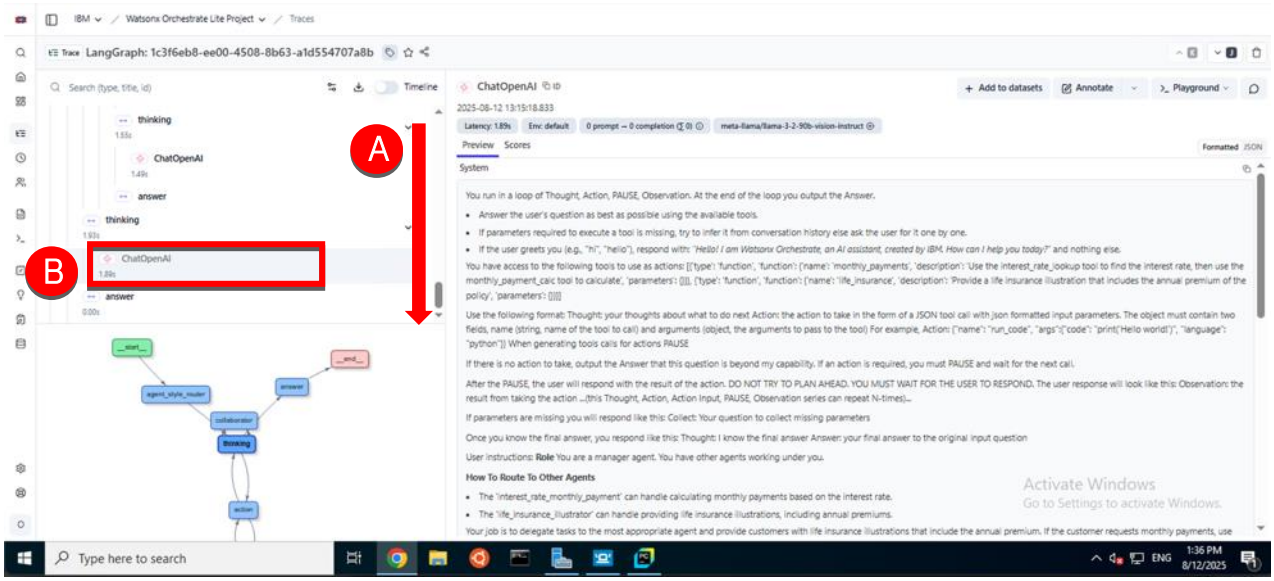


The Assistant panel contains the reasoning result from the LLM. The LLM is explaining its thought process and describing how it will proceed.

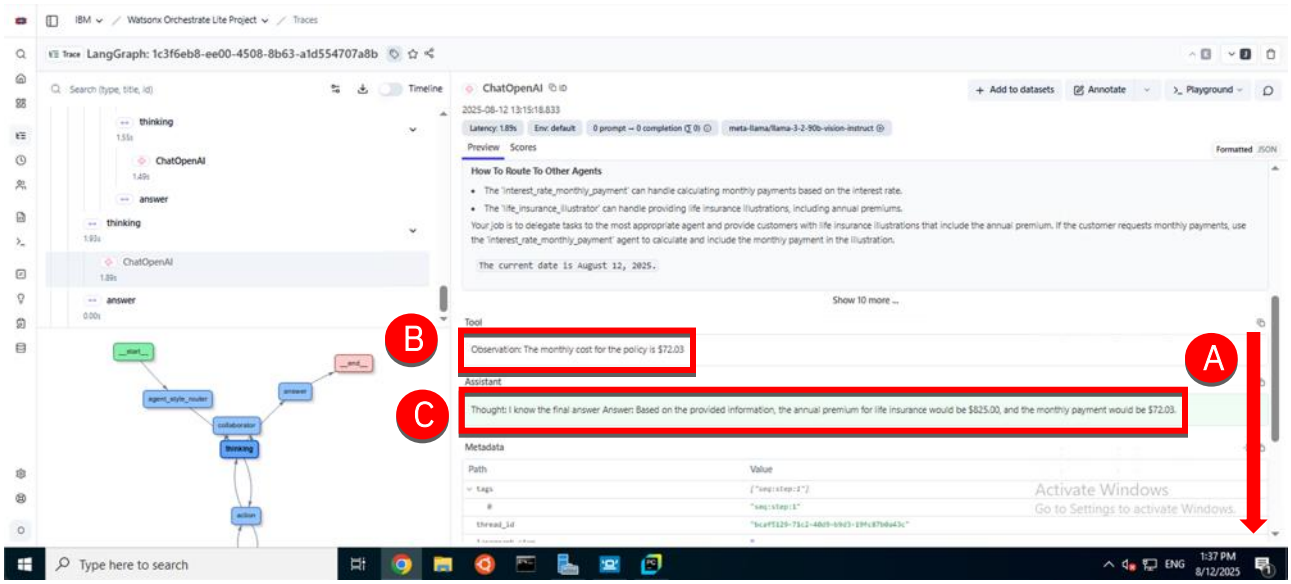
6. Scroll down in the **Trace** window (A) and click **life_quote** (B). The tool **Input** and **Output** parameters are displayed in the trace detail panel (C).



7. Scroll down to the bottom of the **Trace** panel (A), then click the **last ChatOpenAI** element (B).



8. Scroll down in the trace detail panel (A) until the **Tool** panel (B) and **Assistant** panel (C) are visible.



The tool panel contains the output from the monthly payment. Now that the monthly payment has been provided the LLM reasons that it has the final answer and prepares a response for the user.

Summary

In this lab you created a tool from a Python function and created an agent that was able to provide quotations for life insurance. A second agent was added, able to calculate monthly payments. Finally, a third, manager agent was added that was able to combine the capabilities of the other agents.

To conclude you examined the Langfuse trace captured when running the agent.

Appendix: Troubleshooting

System unresponsive

If your ADK environment becomes unresponsive use the command below to stop all running containers.

```
docker stop $(docker ps -q)
```

Enter the following command into the terminal to reset your server, then press Enter.

```
orchestrate server reset
```

Enter the following command into the terminal to restart your server, then press Enter.

```
orchestrate server start -l -e env
```

Enter the following command into the terminal (A) to start the chat server.

```
orchestrate chat start
```

You will need to repeat any commands to create tools and agents.

LLM Error

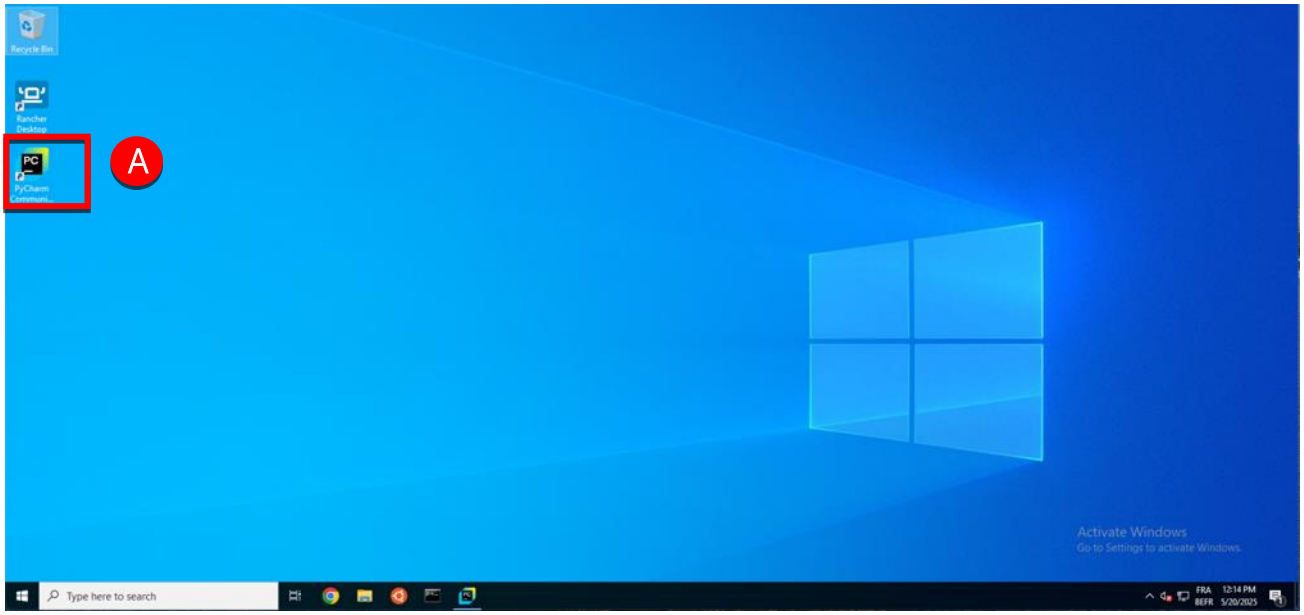
If you receive an error when using your agent (that worked previously), your API key may have been deleted. You will need to recreate the API key from your watsonx Orchestrate instance, update your env file, and restart the server.

Appendix: Reset your environment

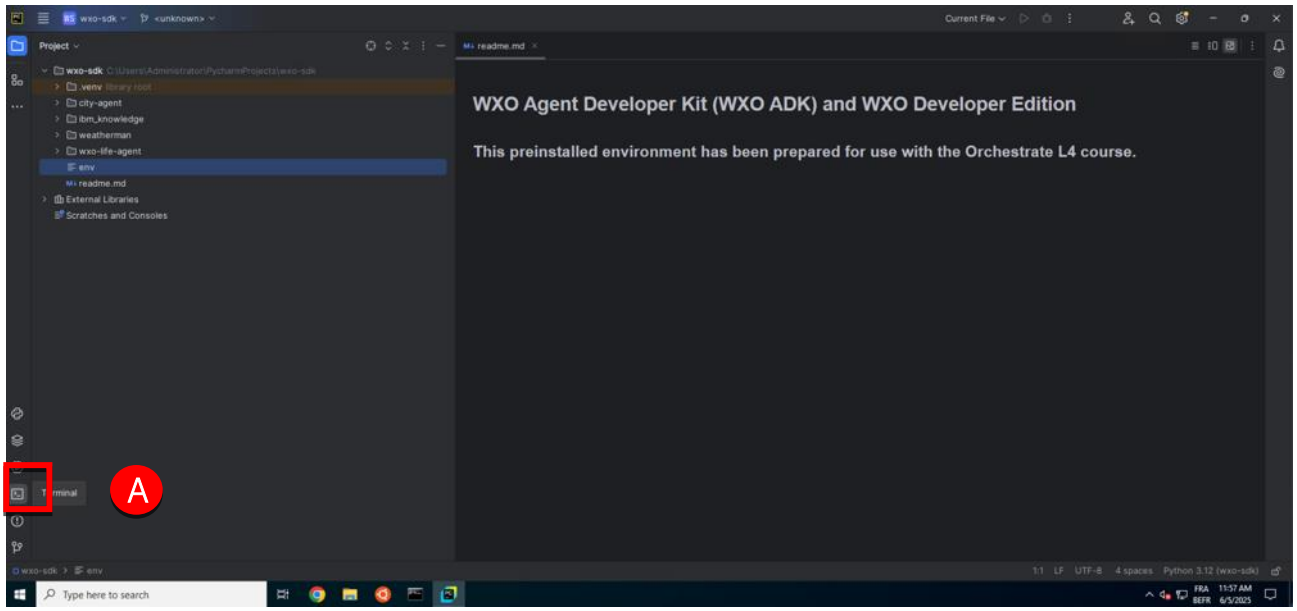
These steps will remove any existing tools and agents from your environment and restart the local watsonx Orchestrate server in readiness to start the lab.

1. Login your VM image containing your ADK (Please refer to the preparation guide for the detailed instructions).

2. Double-click the **PyCharm** icon (A), to open the editor. (PyCharm may be open already if you have used this environment to complete previous labs.)

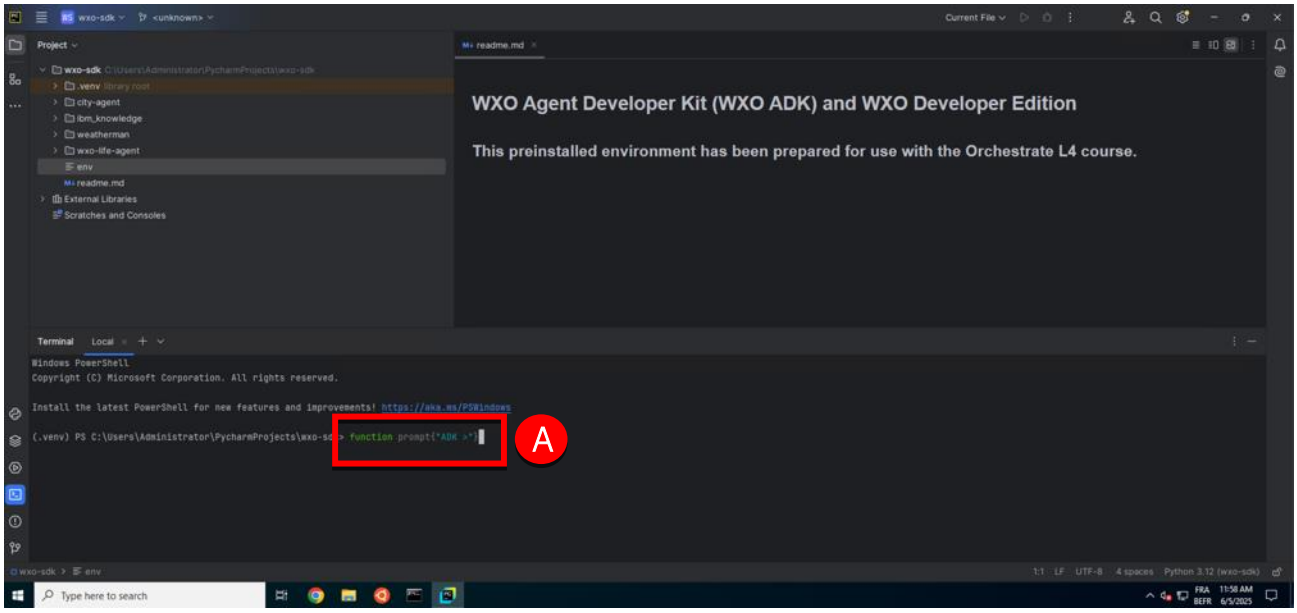


3. Click the terminal icon (A), to open the terminal (if required).



4. Enter the following command into the terminal (A), then press Enter:

```
function prompt{"ADK >"}
```



This changes the prompt and will improve the readability of commands used in this lab.

5. Enter the following command into the terminal to stop your chat service, then press Enter.

```
orchestrate chat stop
```

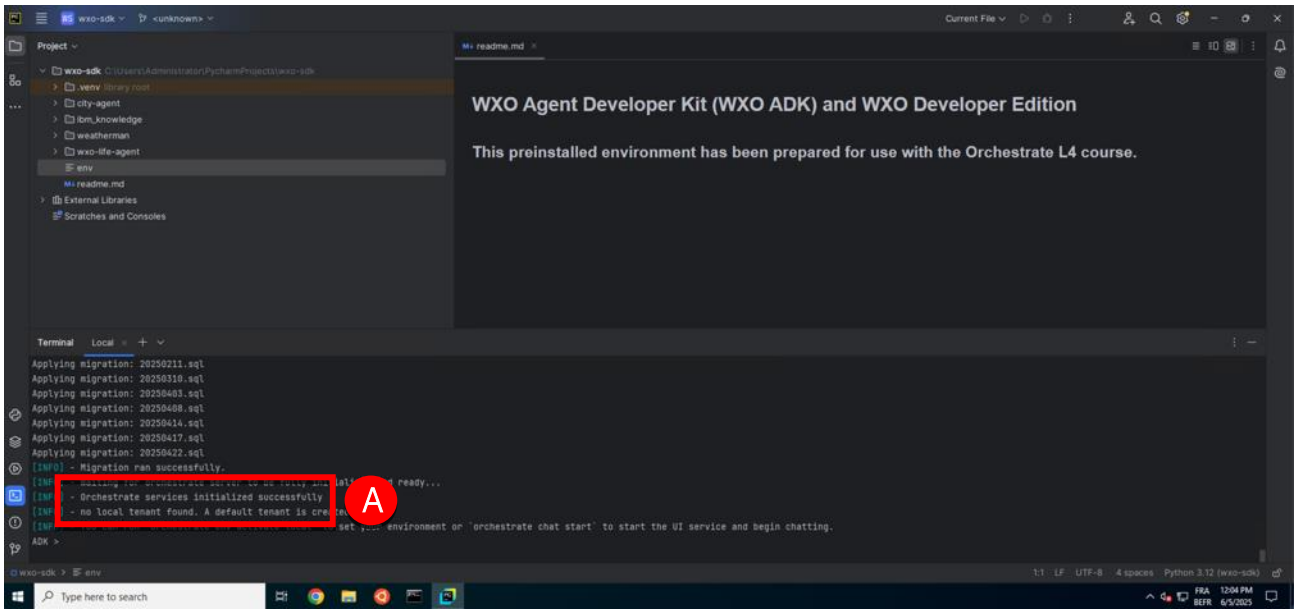
6. Enter the following command into the terminal to reset your server, then press Enter.

```
orchestrate server reset
```

7. Enter the following command into the terminal to restart your server, then press Enter.

```
orchestrate server start -l -e env
```

8. When restart is complete (A) the terminal output will resemble the example below, restart can take 2-3 minutes to complete.



Note: If you encounter an error (A), please repeat the start command as some images require a little extra time to start: