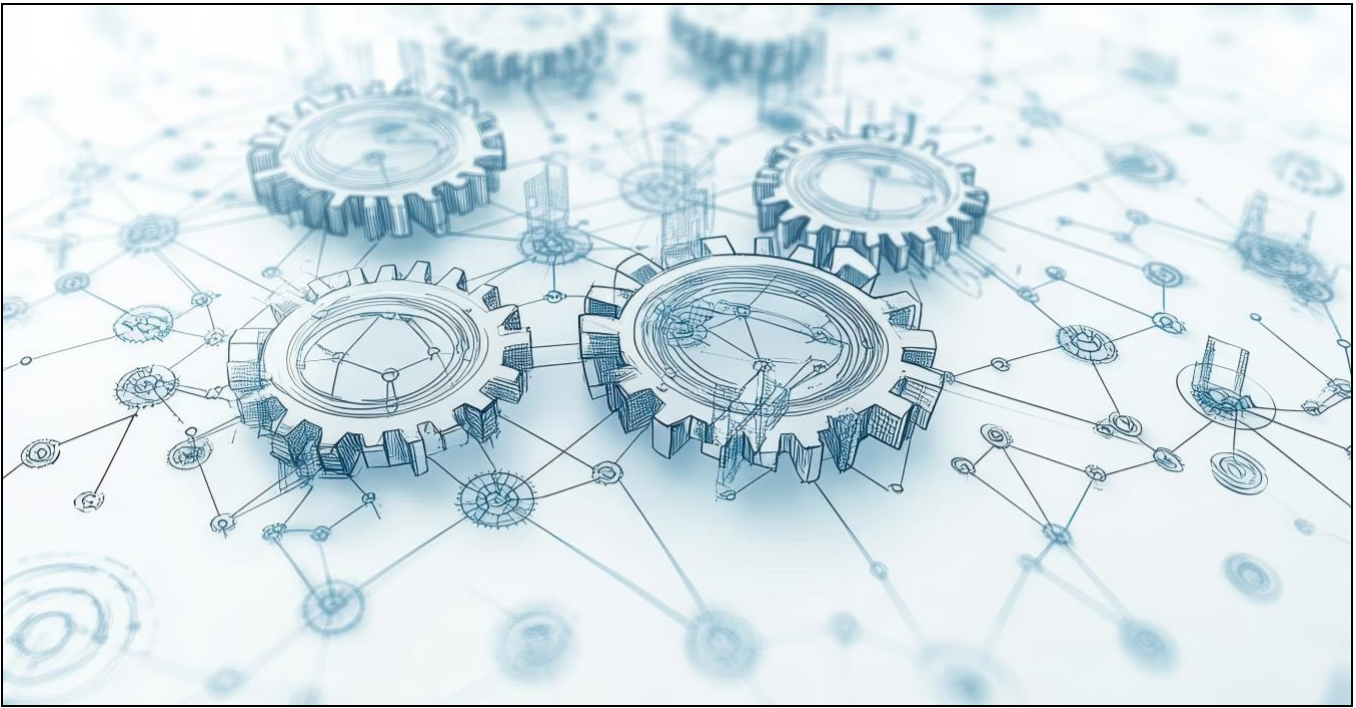


Level 4

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

LEGAL NOTICES

This information was developed for products and services offered in the USA. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119 Armonk, NY 10504-1785
United States of America

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you. This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice. Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All names and references for organizations and other business institutions used in this deliverable's scenarios are fictional. Any match with real organizations or institutions is coincidental. All names and associated information for people in this deliverable's scenarios are fictional. Any match with a real person is coincidental.

TRADEMARKS

IBM, the IBM logo, and [ibm.com](https://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <https://www.ibm.com/legal/copyright-trademark>.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates. INFOBLOX is a trademark of Infoblox Inc. or its affiliated companies, registered in the United States and other countries.

Microsoft®, Azure, Windows, Windows NT, Windows Server, Microsoft® Excel®, Microsoft® Word®, Microsoft® Outlook®, Microsoft® Excel® logo, Microsoft® Word® logo, Microsoft® Outlook® logo, Microsoft® Windows App, Microsoft® Windows App logo and the Windows logo are trademarks of Microsoft® Corporation in the United States, other countries, or both.

Salesforce and Salesforce logo are registered trademarks of SALESFORCE, INC. (CORPORATION; CALIFORNIA, USA).

Slack is a registered trademark of Salesforce, Inc (CORPORATION; DELAWARE, USA).

Red Hat®, JBoss®, OpenShift®, Fedora®, Hibernate®, Ansible®, CloudForms®, RHCA®, RHCE®, RHCSA®, Ceph®, and Gluster® are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Google, Google logo, Google Tasks, Google Cloud Platform, GMAIL, GMAIL logo are registered trademarks of GOOGLE LLC and this material is not endorsed by or affiliated with Google in any way.

Elasticsearch, Elastic search logo, Elastic, Elastic logo are registered trademarks of Elasticsearch BV (LIMITED LIABILITY COMPANY; NETHERLANDS) in United States, other countries or both.

Milvus, Milvus logo are registered trademarks of LF Projects, LLC (LIMITED LIABILITY COMPANY; DELAWARE, USA) in United States, other countries or both.

Meta, Meta logos, Llama, Llama logos are registered trademarks of META PLATFORMS, INC. (CORPORATION; California, USA) in United States, other countries or both.

Mistral, Mistral logo, Mistral AI, Mistral AI logo are registered trademarks of Mistral AI (SOCIÉTÉ PAR ACTIONS SIMPLIFIÉE (SAS); FRANCE) in United States, other countries or both.

PyCharm, PyCharm logo are registered trademarks of JetBrains s.r.o. (CORPORATION; CZECH REPUBLIC) in United States, other countries or both.

Bruno is a trademark held by Anoop M D. Bruno logo is sourced from OpenMoji. License: CC BY-SA 4.0

Apple, Apple logo, Apple store, App store, App store logo are registered trademarks of Apple Inc. (CORPORATION; CALIFORNIA, USA) in United States, other countries or both.

VMWARE, VMWARE logos are registered trademarks of VMware LLC (LIMITED LIABILITY COMPANY; Delaware, USA) in United States, other countries or both.

RANCHER, RANCHER logo, Rancher Desktop, Rancher Desktop logo are registered trademarks of SUSE LLC (LIMITED LIABILITY COMPANY; MASSACHUSETTS, USA) in United States, other countries or both.

Langfuse, Langfuse logo are registered trademarks of Langfuse GmbH (GESELLSCHAFT MIT BESCHRÄNKTER HAFTUNG (GMBH); Germany, Germany) in United States, other countries or both.

OpenMeteo is a registered trademark owned by Patrick Zippenfenig (Individual, Germany) in United States, other countries or both.

© Copyright International Business Machines Corporation 2025. This document may not be reproduced in whole or in part without the prior written permission of IBM.

Table of Contents

Introduction	1
Lab overview.....	1
What you will learn	2
Environment	2
How to get support.....	2
Prerequisites	2
Life insurance agent.....	3
Tool and agent creation	3
Agent testing	8
Monthly payment agent	10
Tool and agent creation	10
Agent creation and testing	13
Collaborating agents	18
Manager agent	18
Testing the manager agent	20
Langfuse trace.....	22
Access the trace	22
Analyze the trace	26
Summary.....	30
Appendix: Troubleshooting.....	31
System unresponsive.....	31
LLM Error.....	31
Appendix: Reset your environment	32

Introduction

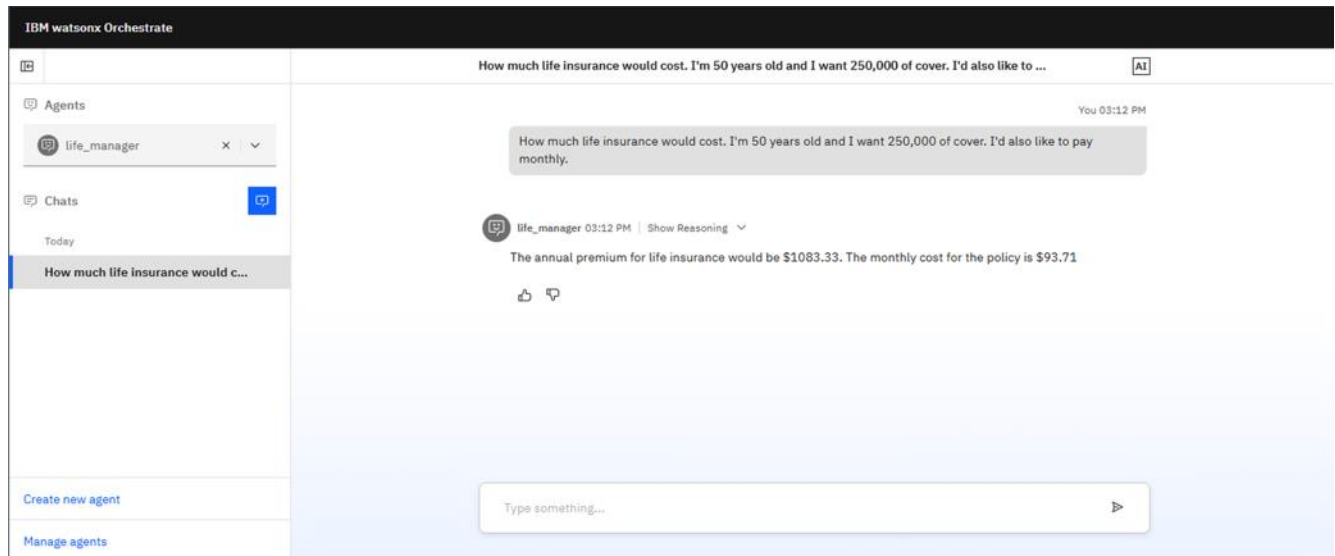
Lab overview

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:

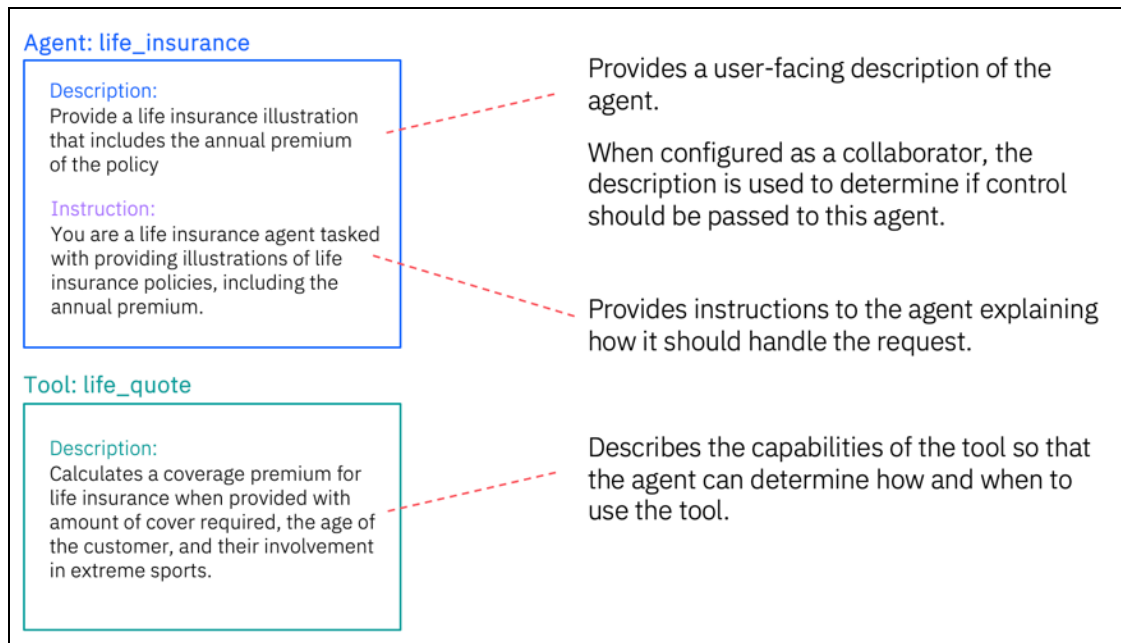
- You have performed the Preparation Lab guide including the environment validation step
- You have your ADK environment up and running (Orchestrate server started after validating the environment in the preparation guide OR after having performed another ADK related lab)

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 this 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.

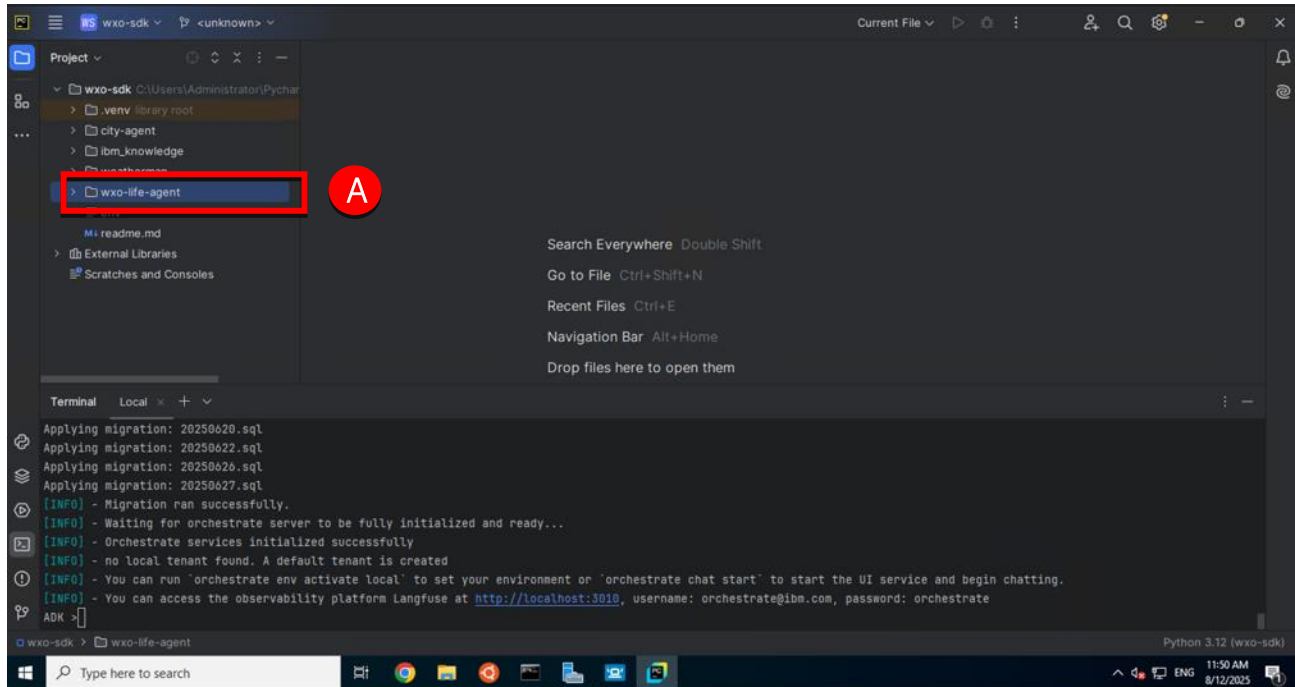


In addition to the tool description, any input and output parameters used by the tool are analyzed and added to the tool configuration. At runtime, the agent shares this configuration as part of the system prompt and is used for planning.

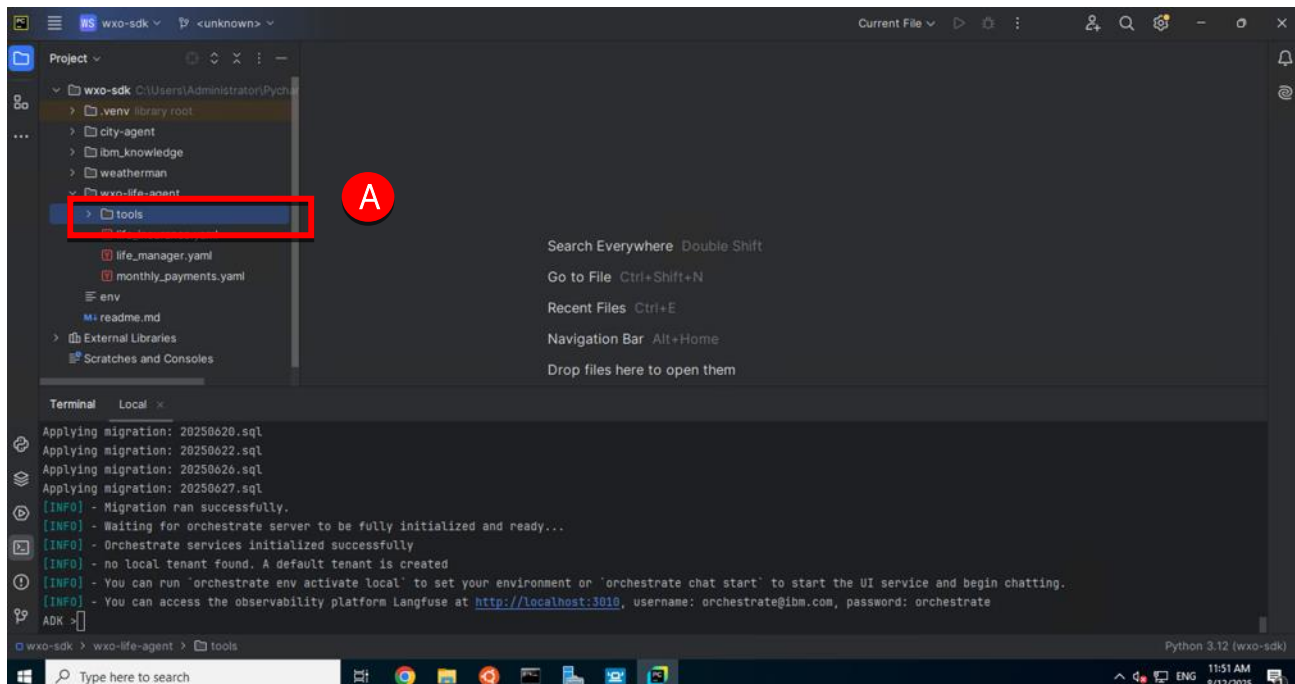
Tool and agent creation

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.

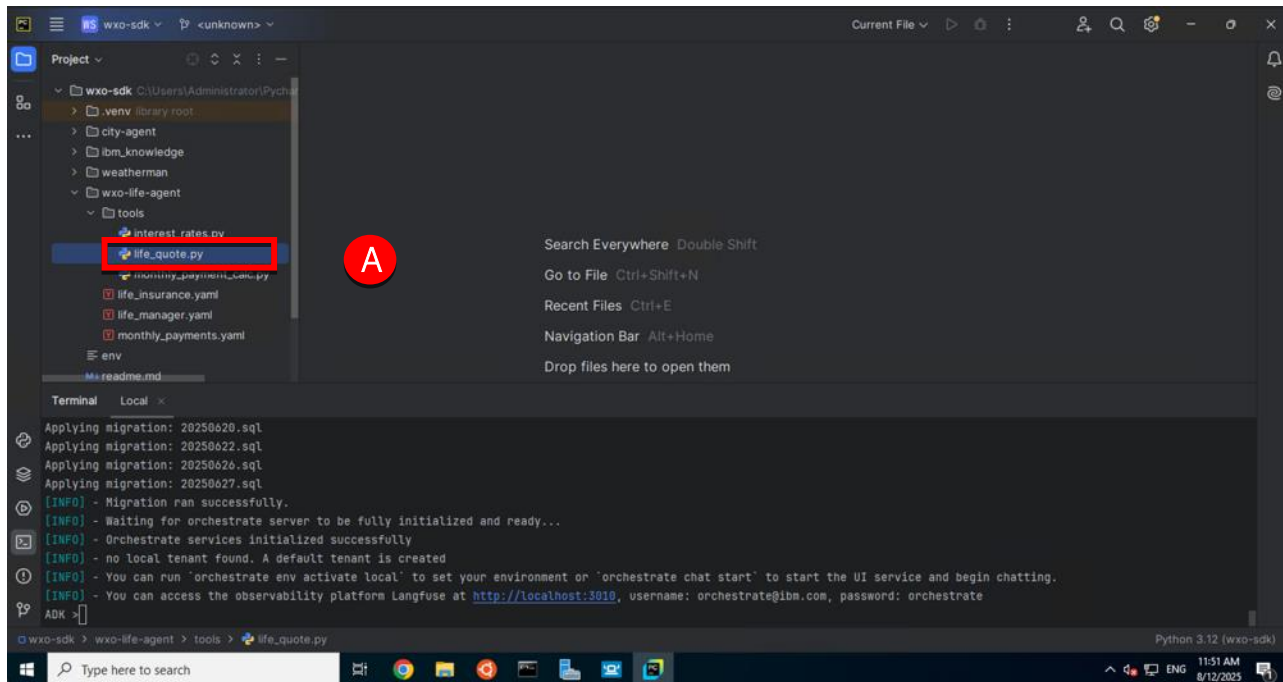
1. Open the ADK environment and start the watsonx Orchestrator server if not yet done. Refer to the [Reset your environment appendix](#) if your orchestrator server is not started.
1. Double-click the **wxo-life-agent** folder (A).



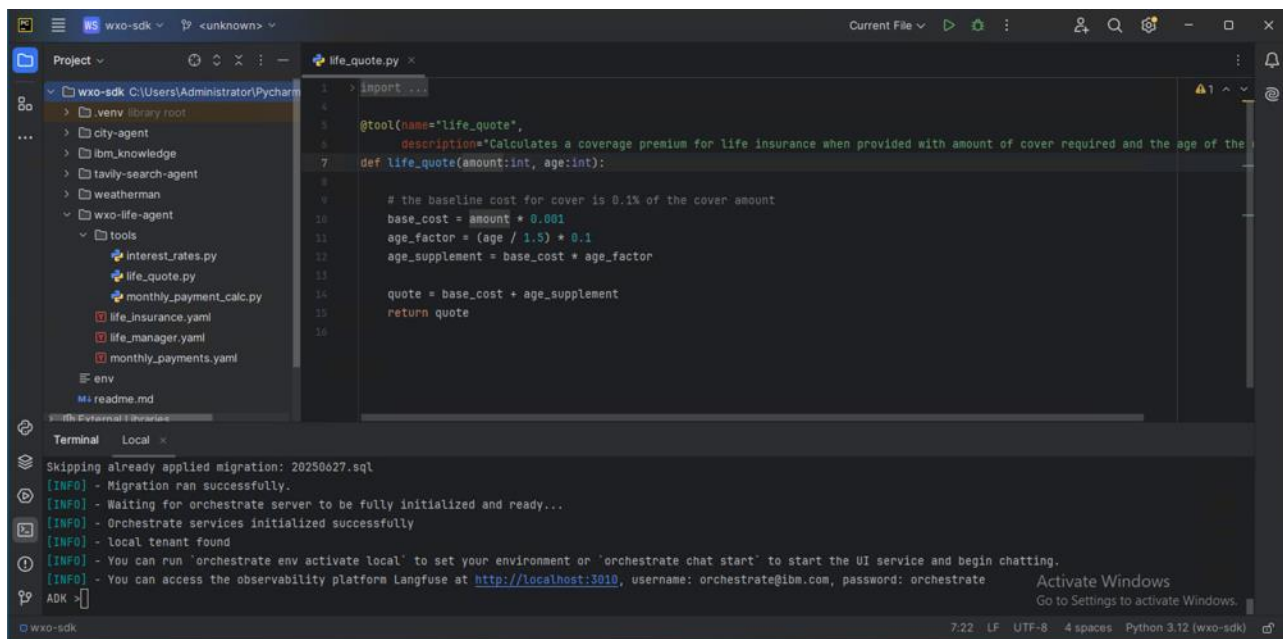
2. Double-click the **tools** folder (A).



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



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

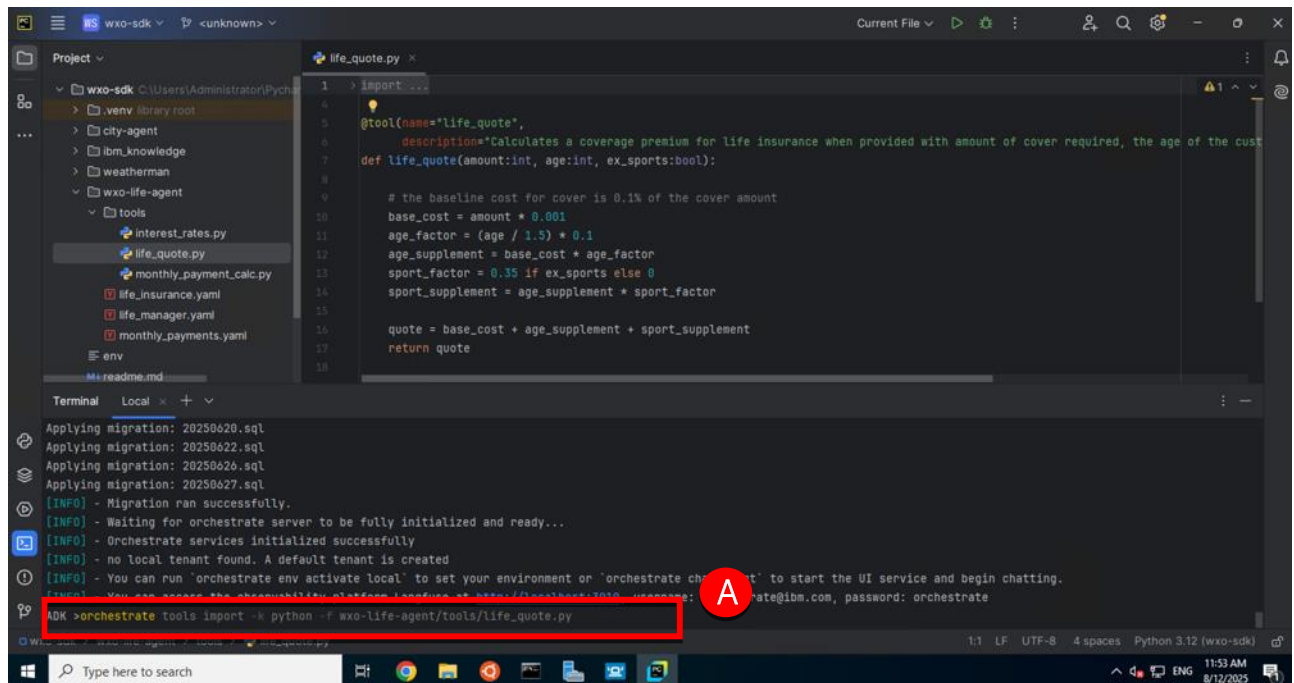


The `@tool` annotation on line 5 identifies this function as an Orchestrate tool, this annotation also provides the **name** of the tool and a **description** of its capabilities. The Python function starts on line 7 and implements a simplistic quotation calculation based on 0.1% of the amount of cover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

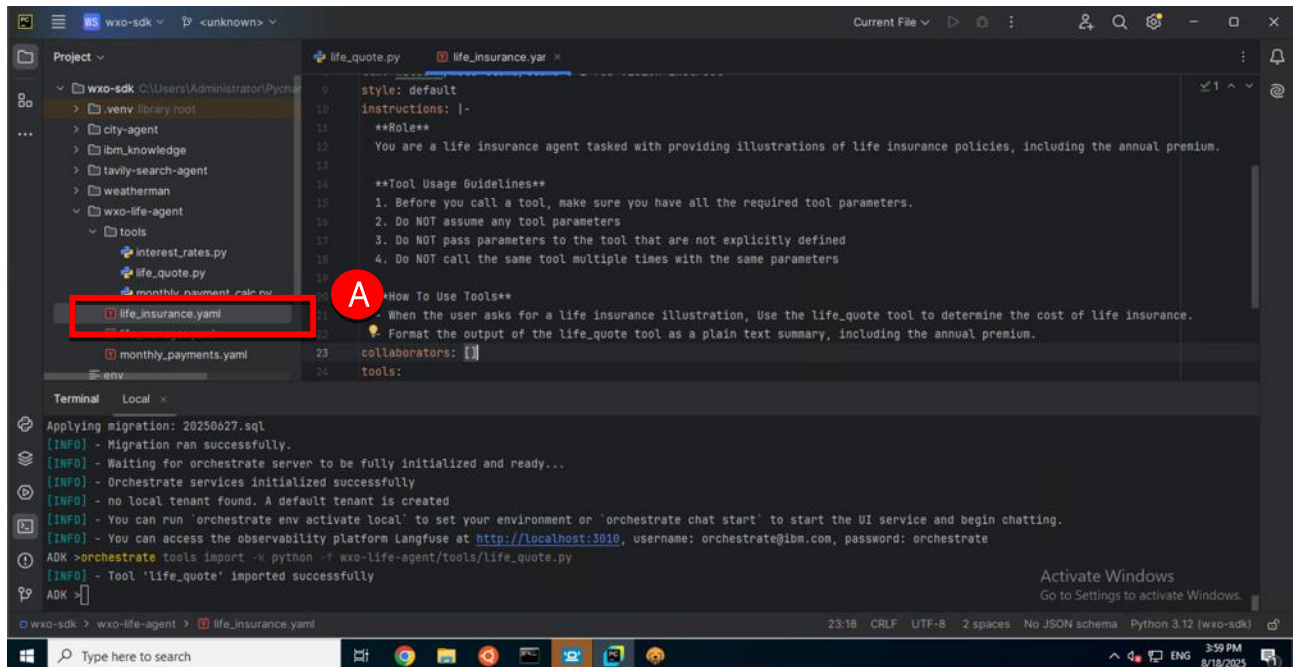
The function receives two parameters, the **amount** of cover required, and the **age** of the customer. When the tool is created, information about these parameters will be added to the configuration and available to any agent using this tool.

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
```



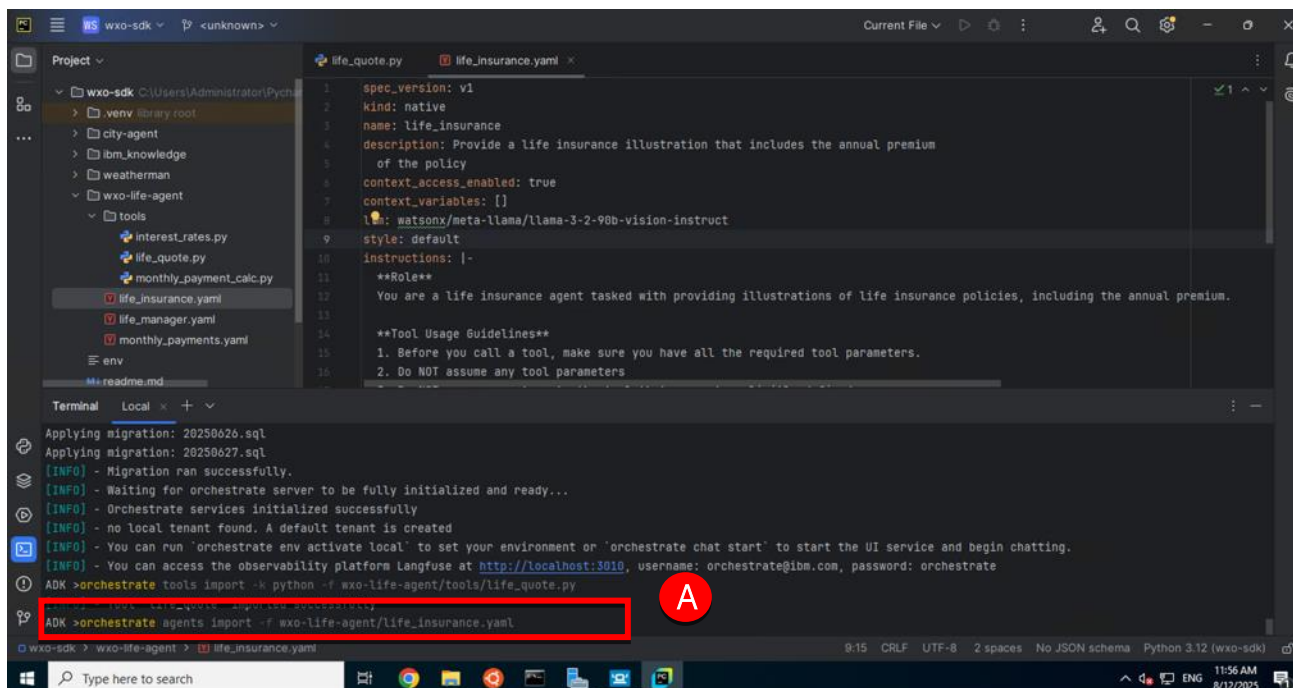
6. Double-click `life_insurance.yaml` (A) and review the definition of the agent.



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
```

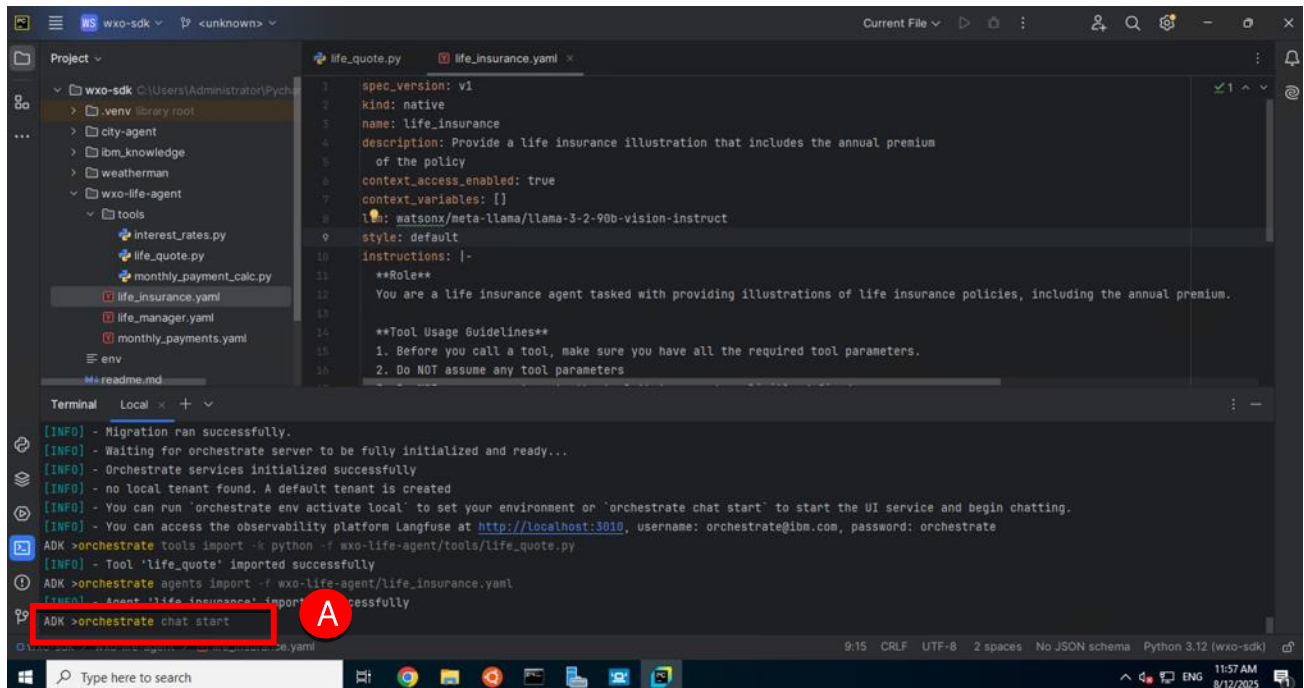


Agent testing

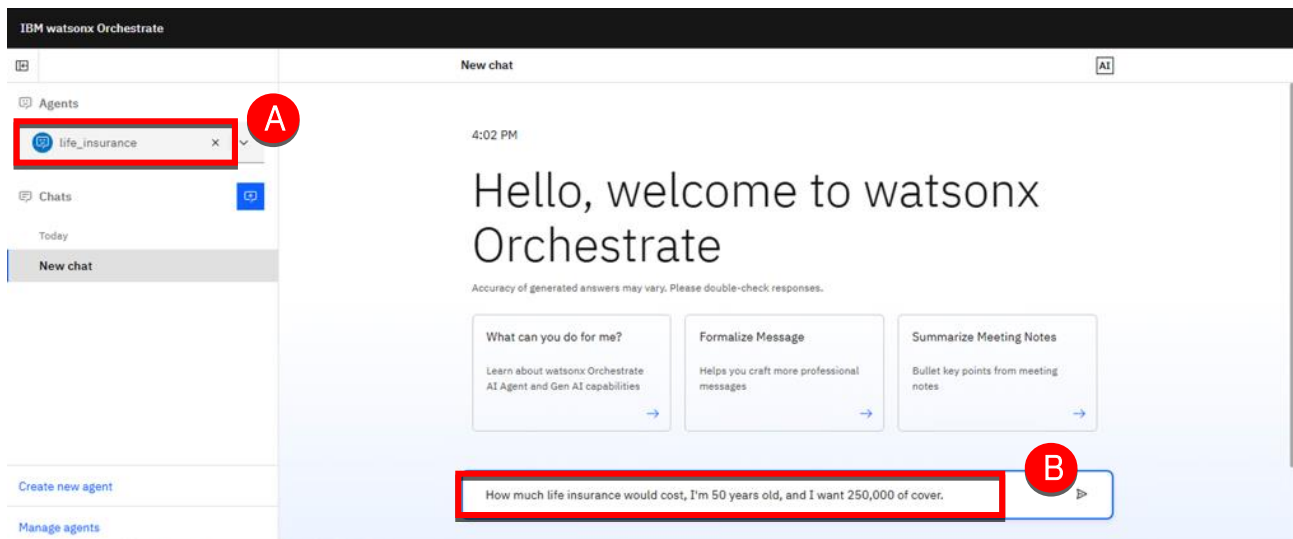
In this section you will start the chat interface, test the agent and review its reasoning in the chat UI.

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

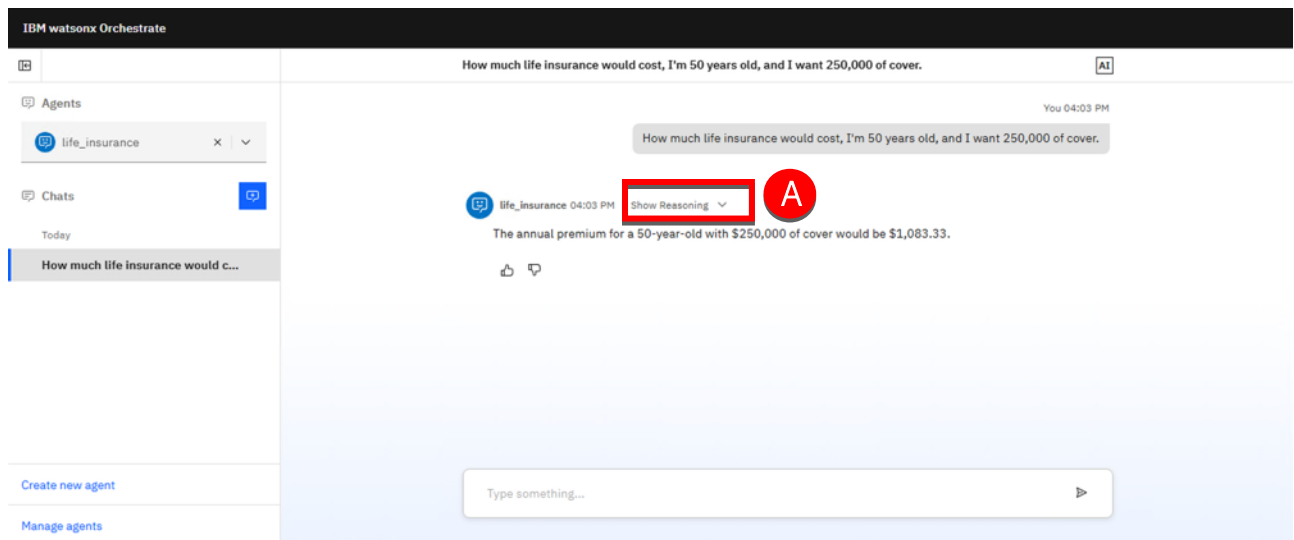
```
orchestrate chat start
```



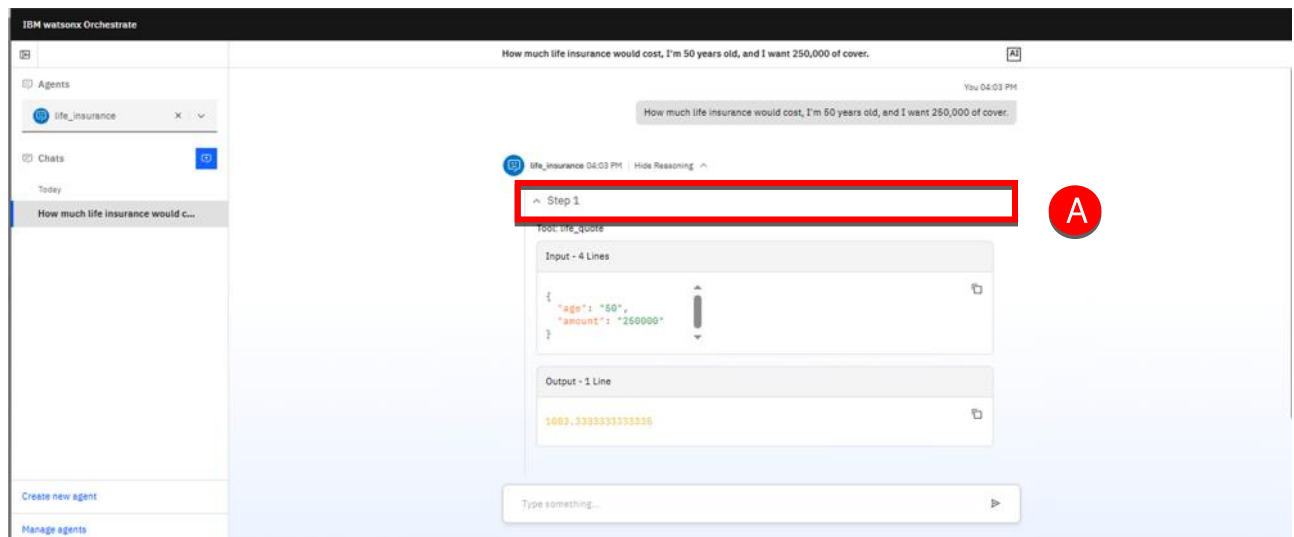
2. When the browser opens, ensure **life_insurance** is selected (A) then enter "How much life insurance would cost, I'm 50 years old, and I want 250,000 of cover." into the chat window (B), then press Enter. If the agent is not available, refresh your browser.



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).

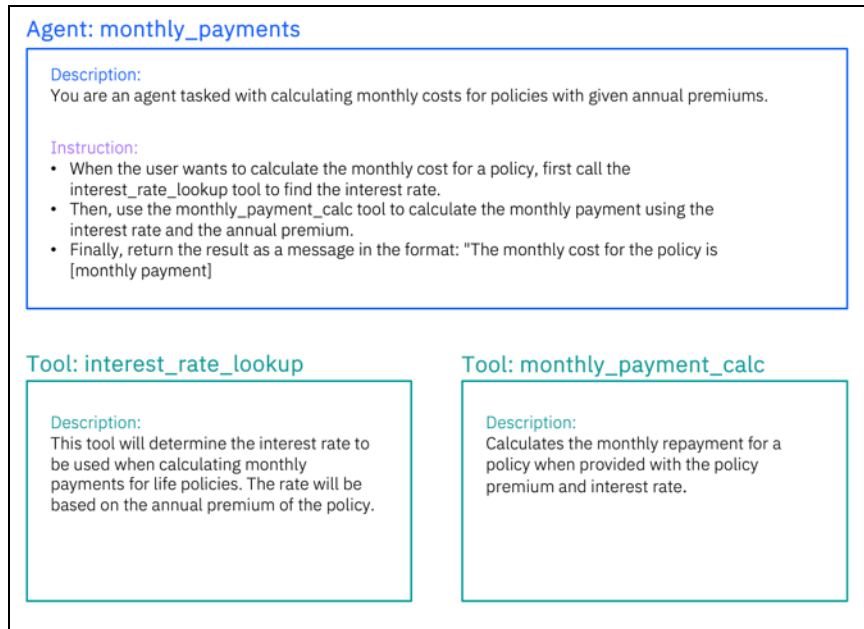


The reasoning panel shows the input and output parameters passed between the agent and the tool.

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.

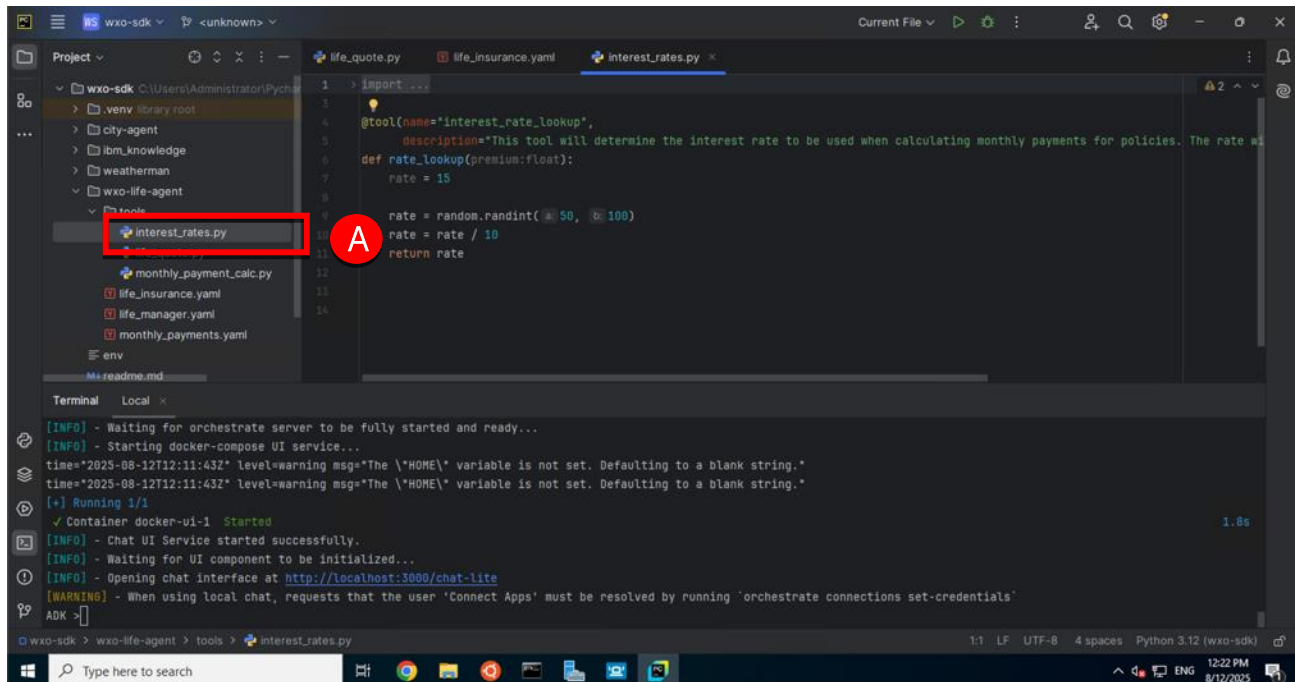
The diagram below provides an overview of the new agent configuration.



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).

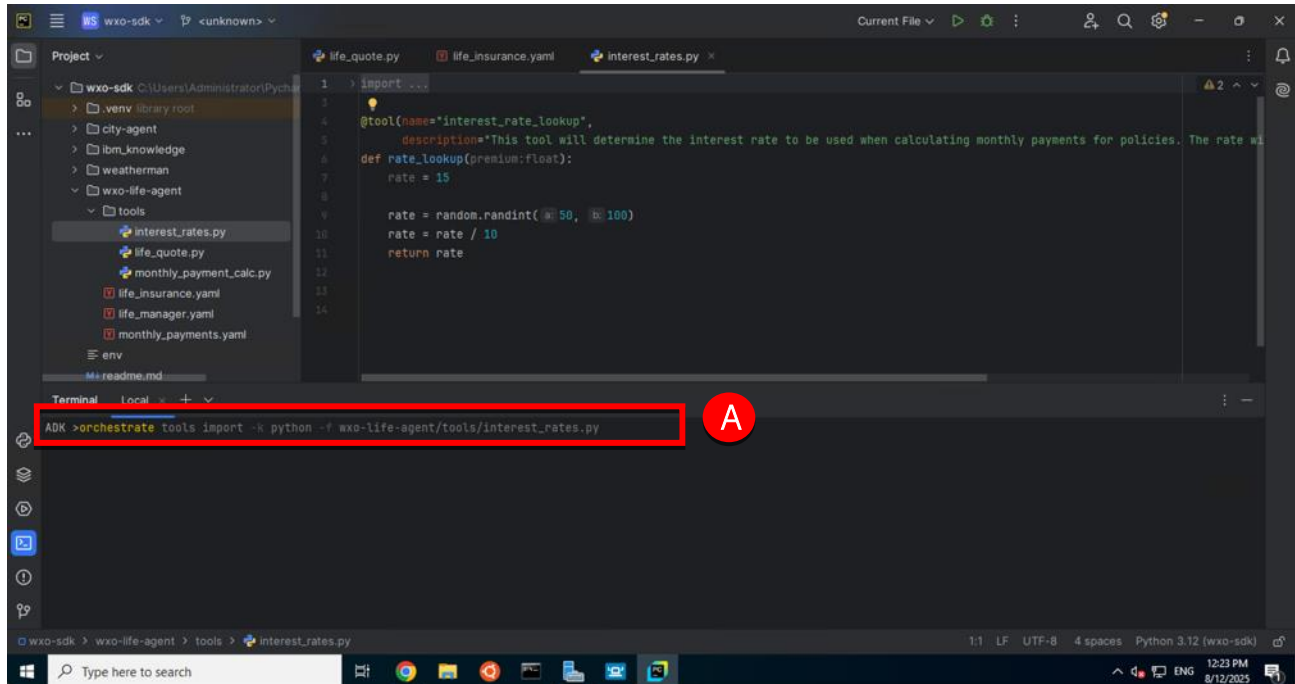


The `@tool` annotation on line 4 identifies the function on line 6 as an Orchestrate tool. This annotation also provides the **name** of the tool and a **description** of its capabilities. The Python function generates a random interest rate lookup between 5-10%. The policy premium is passed as a parameter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

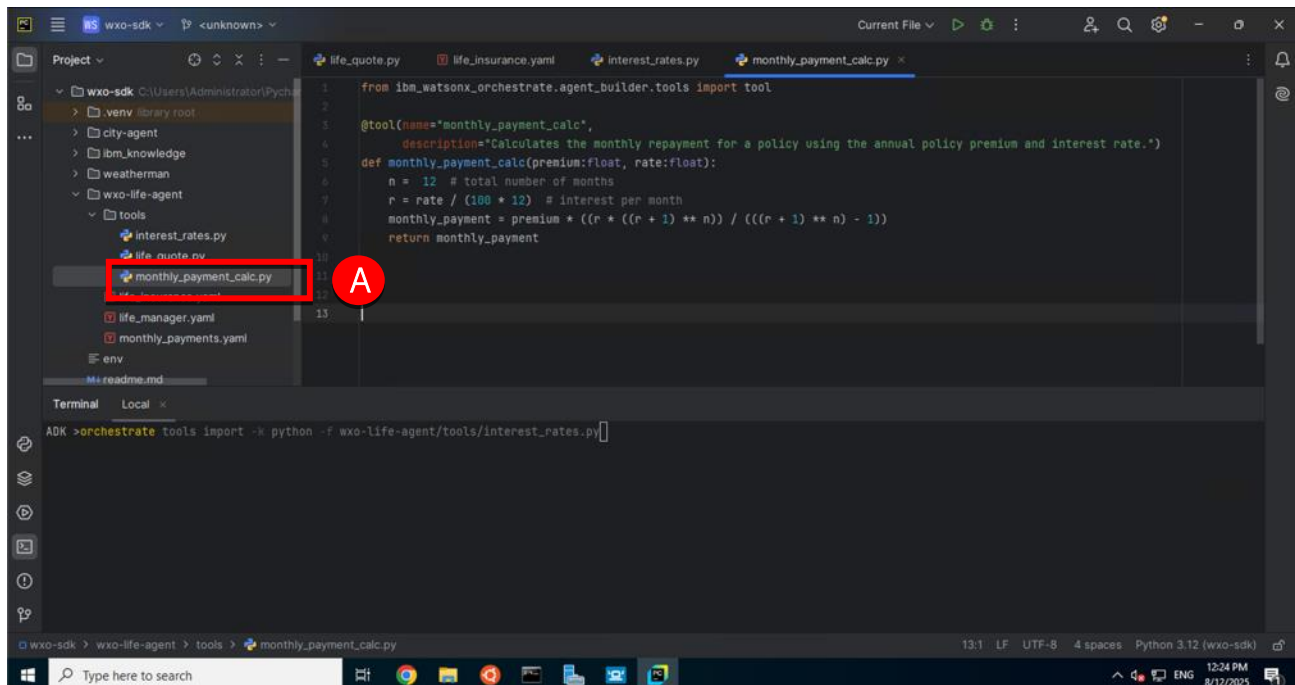
This function is simply a random number generator, and as such, the calculations performed using its response will vary.

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
```



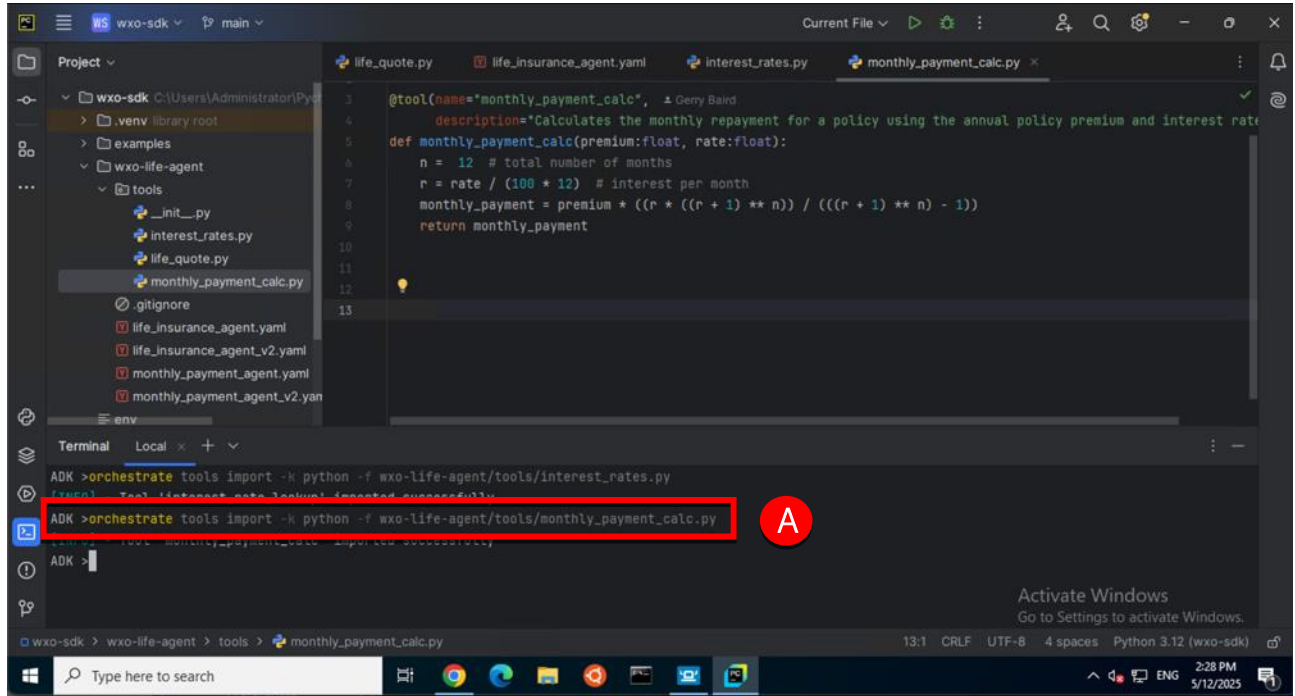
3. Double-click `monthly_payment_calc.py` (A).



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.

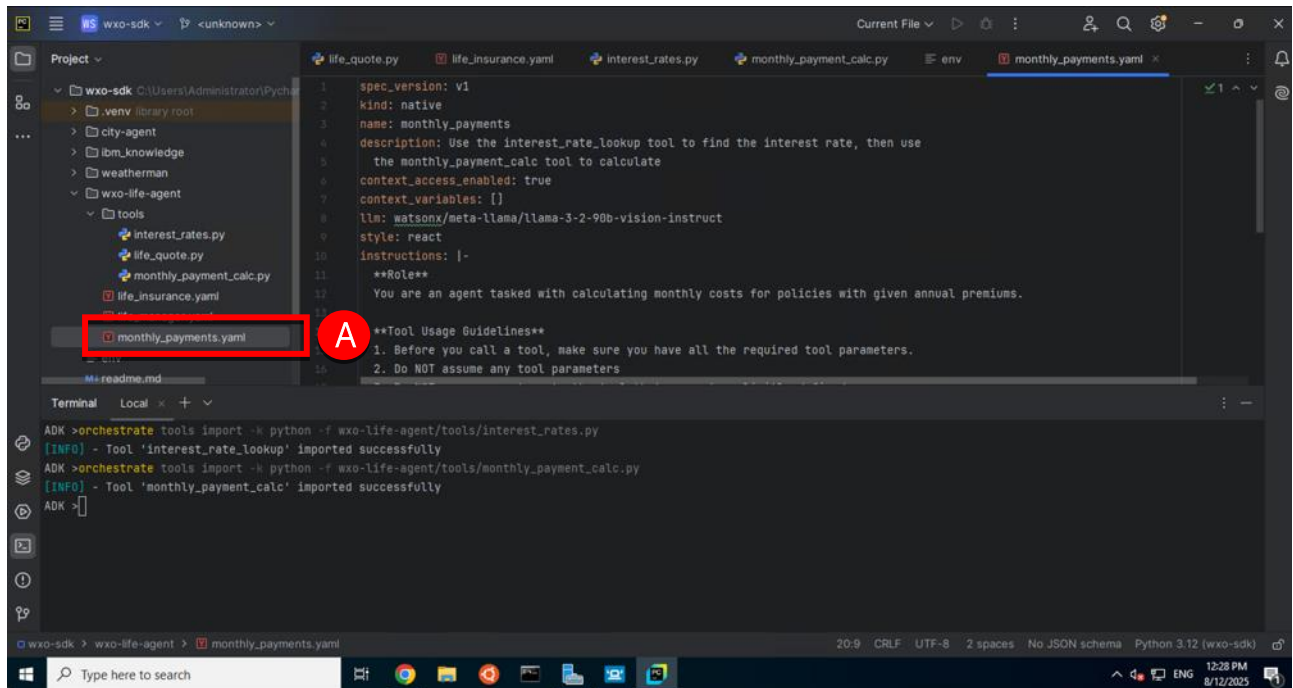
```
orchestrate tools import -k python -f wxo-life-agent/tools/monthly_payment_calc.py
```



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.

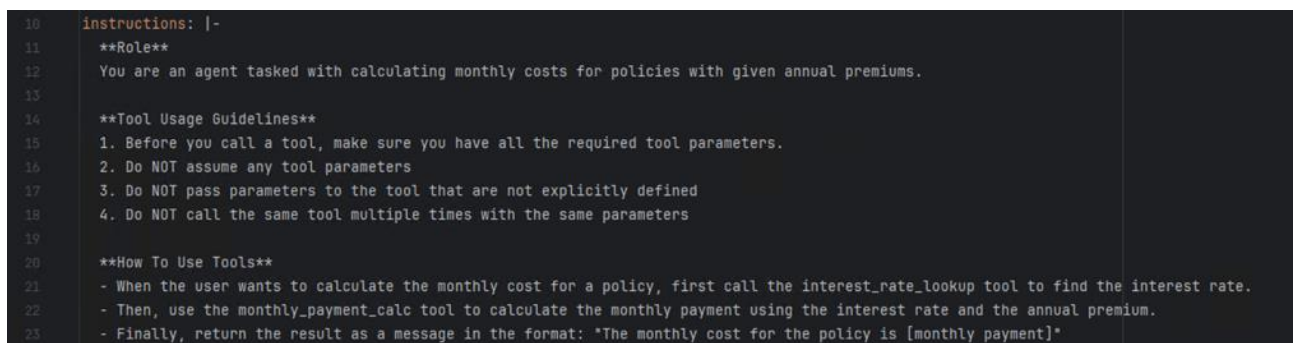


This yaml file defines the agent, its description and behavior, and the LLM it uses for planning and reasoning.



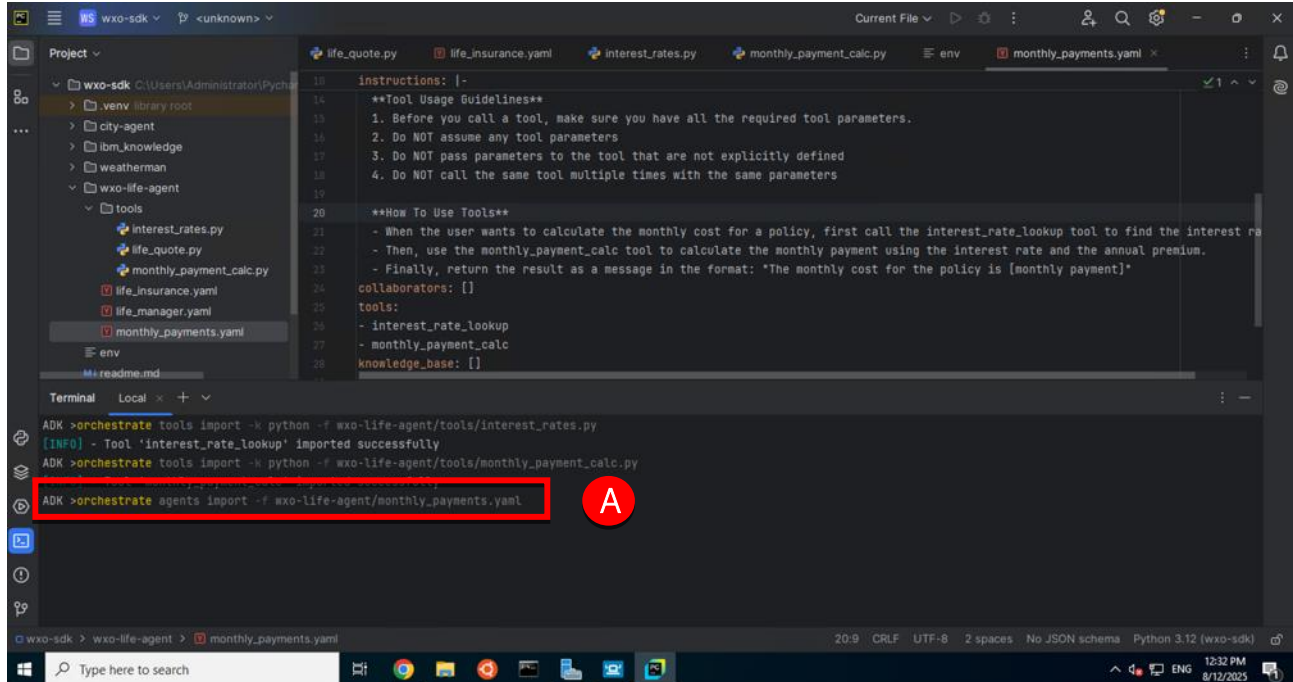
Most LLM's will try and generate an approximate monthly payment. This approximate calculation may appear reasonable, but it won't be based on an accurate amortization formula like the one used in the tool. As LLM's become more capable, identifying these hallucinations is increasingly difficult.

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

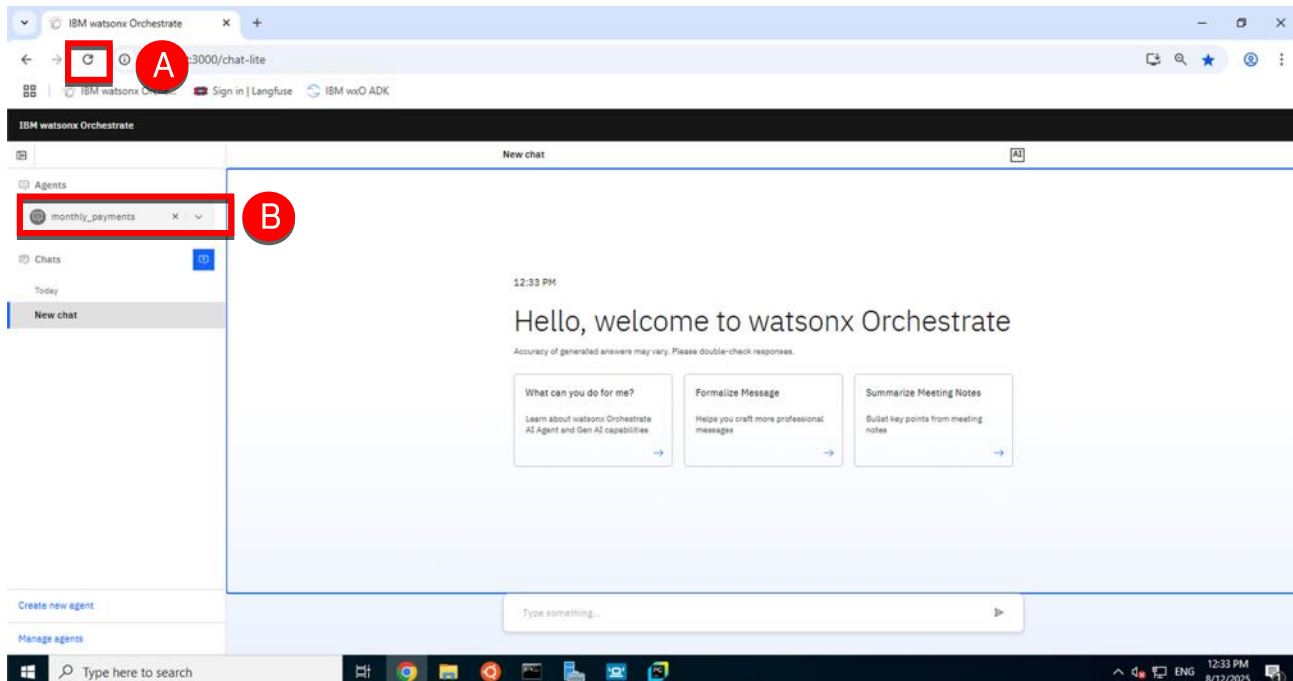


6. Enter the following command into the terminal (A) to create the agent.

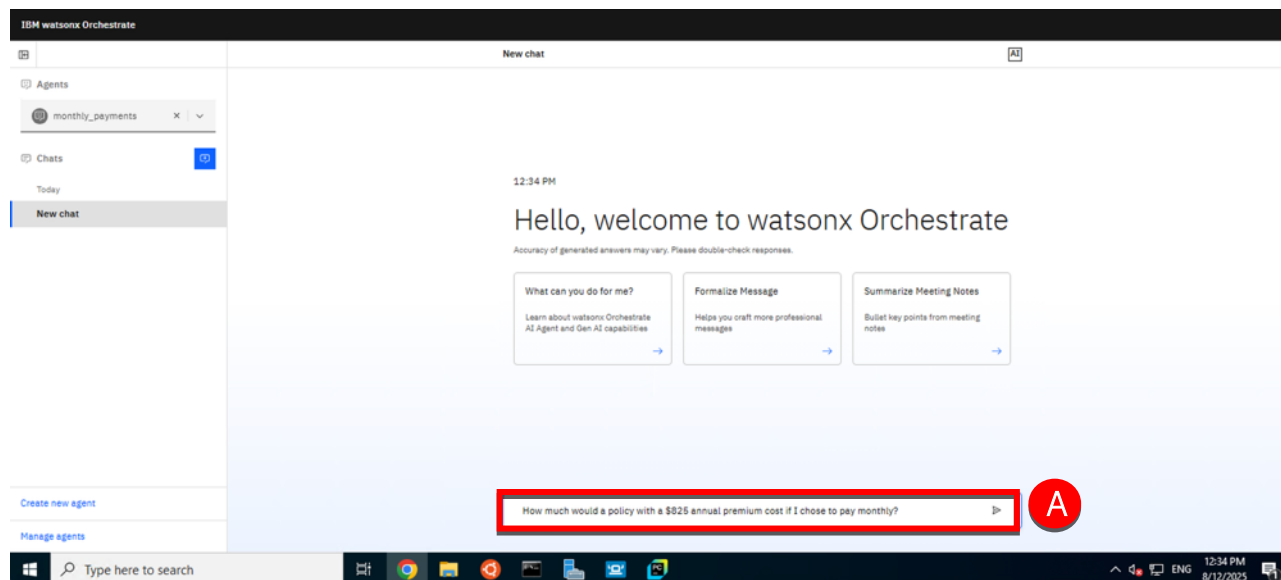
```
orchestrate agents import -f wxo-life-agent/monthly_payments.yaml
```



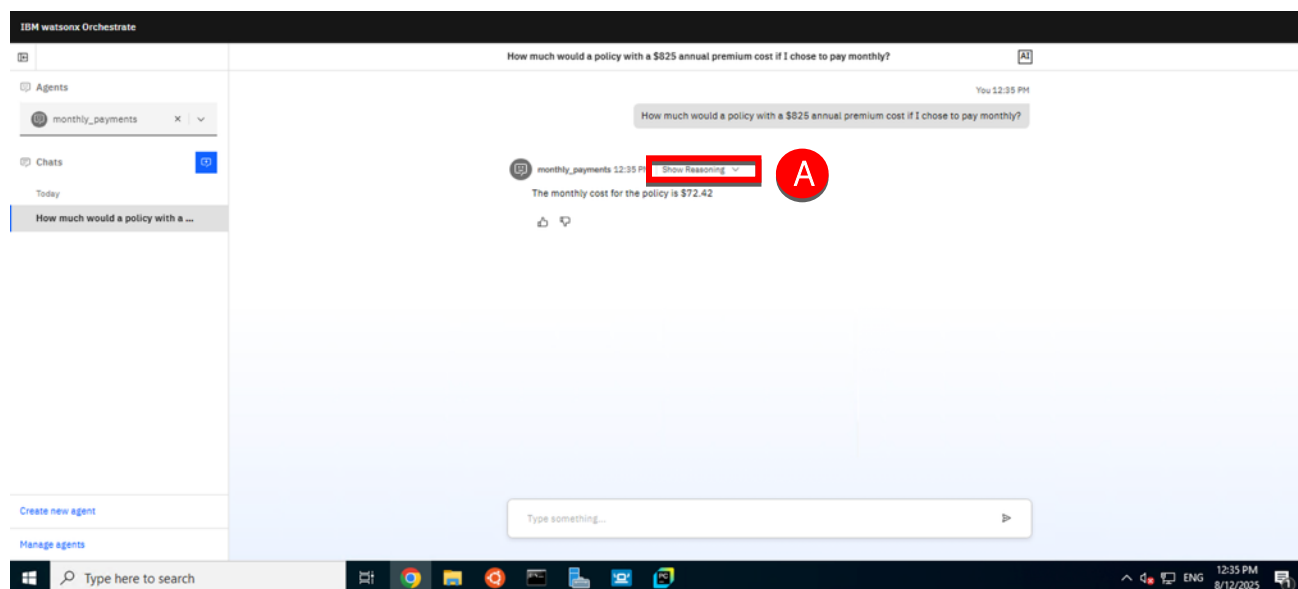
7. In the browser window click the refresh icon (A) then select **monthly_payments** from the **Agents** list (B).



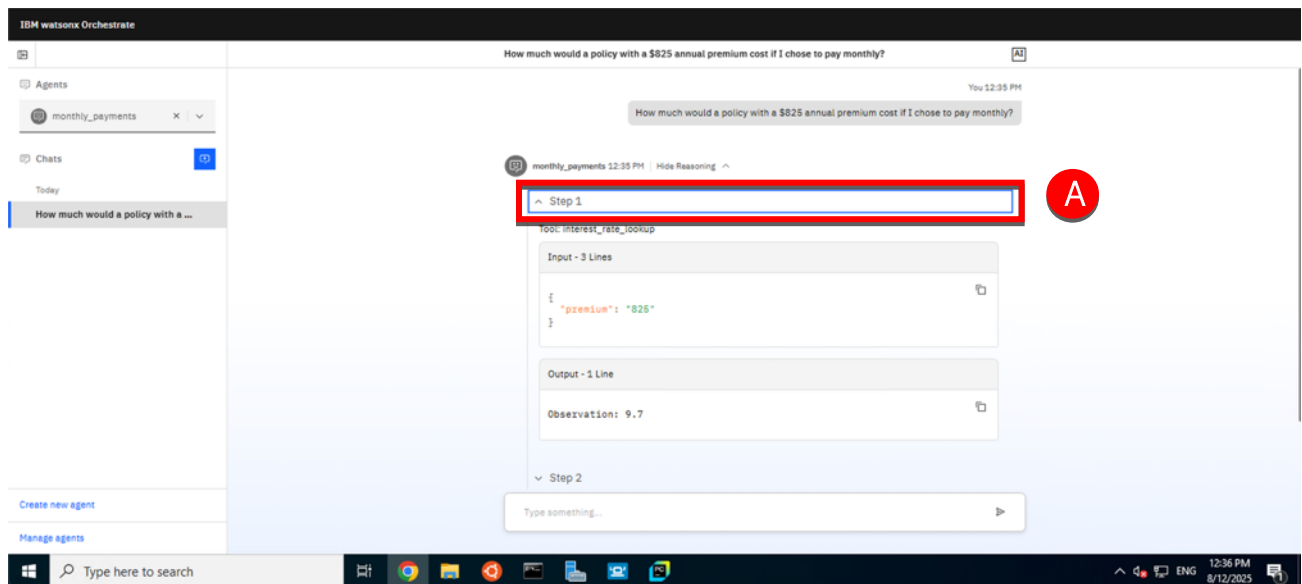
8. Enter “How much would a policy with a \$825 annual premium cost if I chose to pay monthly?” into the chat window (A), then press Enter.



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

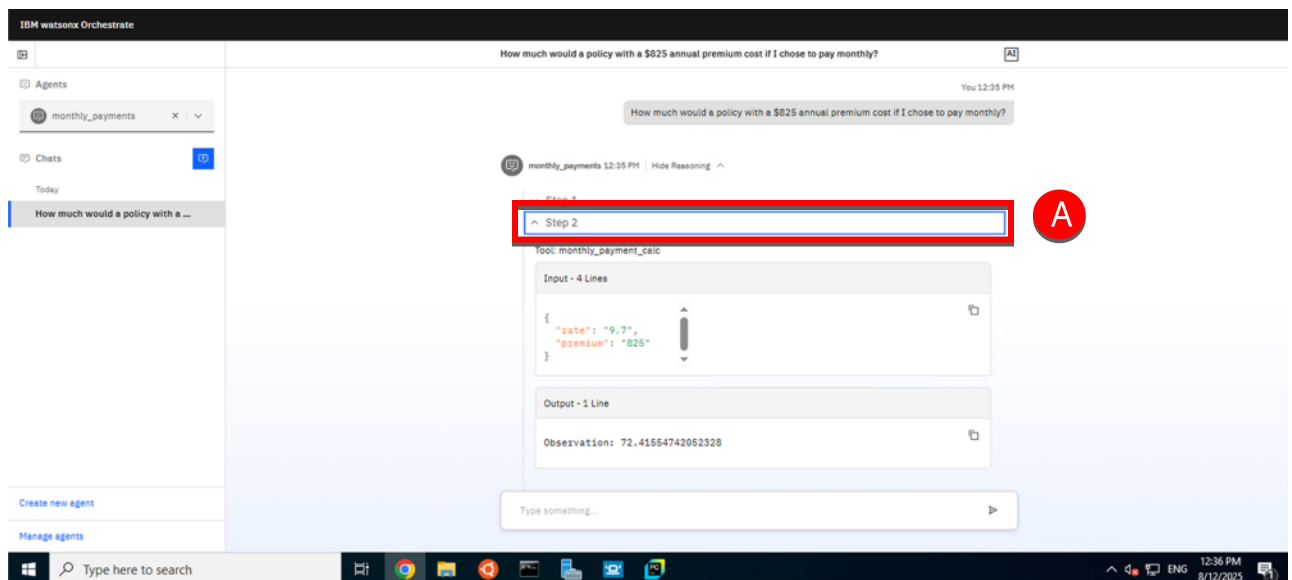


10. Expand Step 1 (A).



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

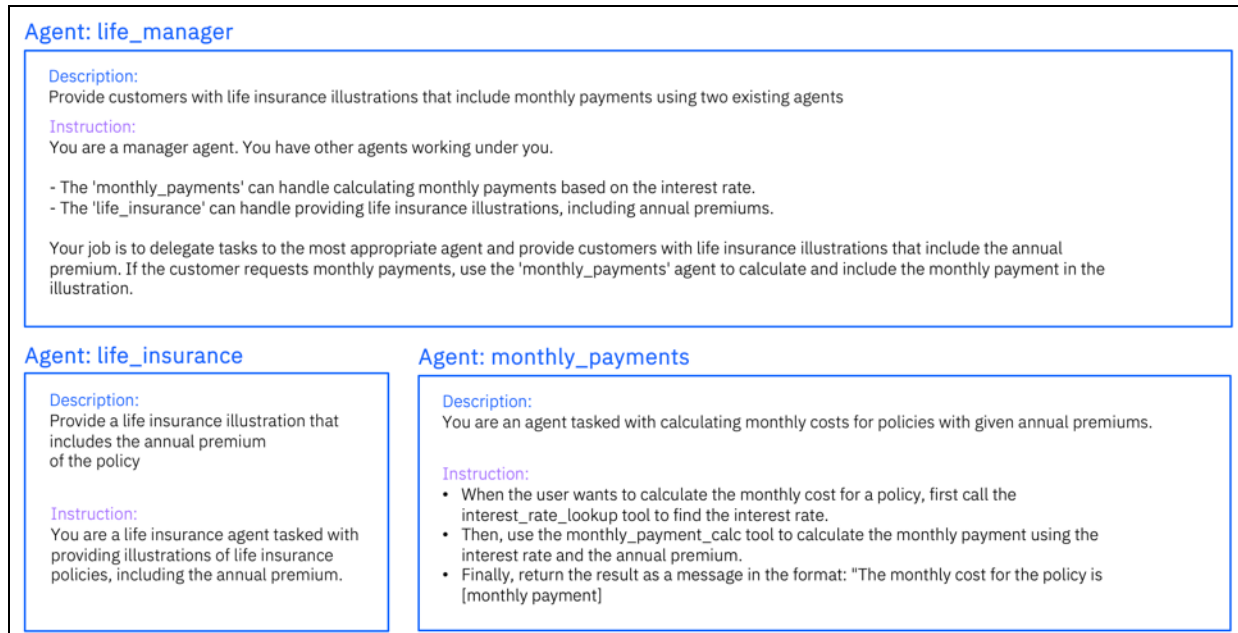
11. Expand Step 2 (A).



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

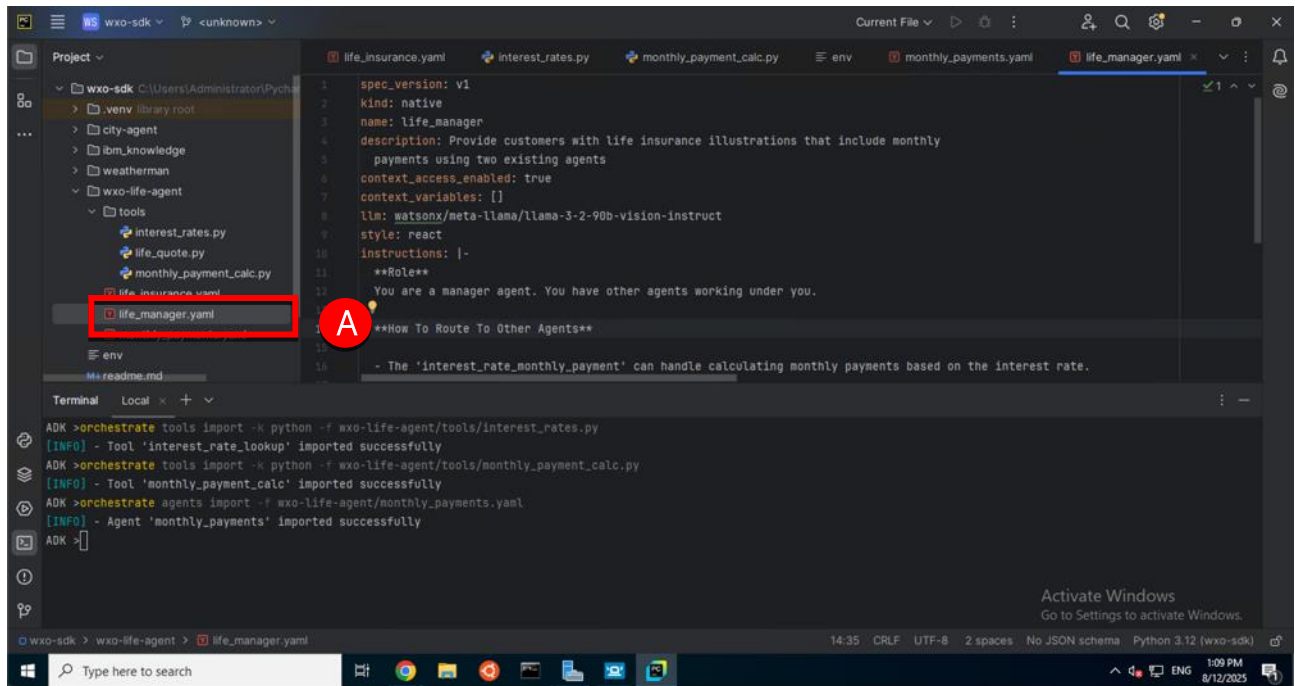
Next you will add a further agent that is able to collaborate with the two existing agents to answer more complex enquires that involve a life cover quotation, combined with monthly payments. The diagram below provides an overview of the expanded configuration used in this section. Tools used by the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

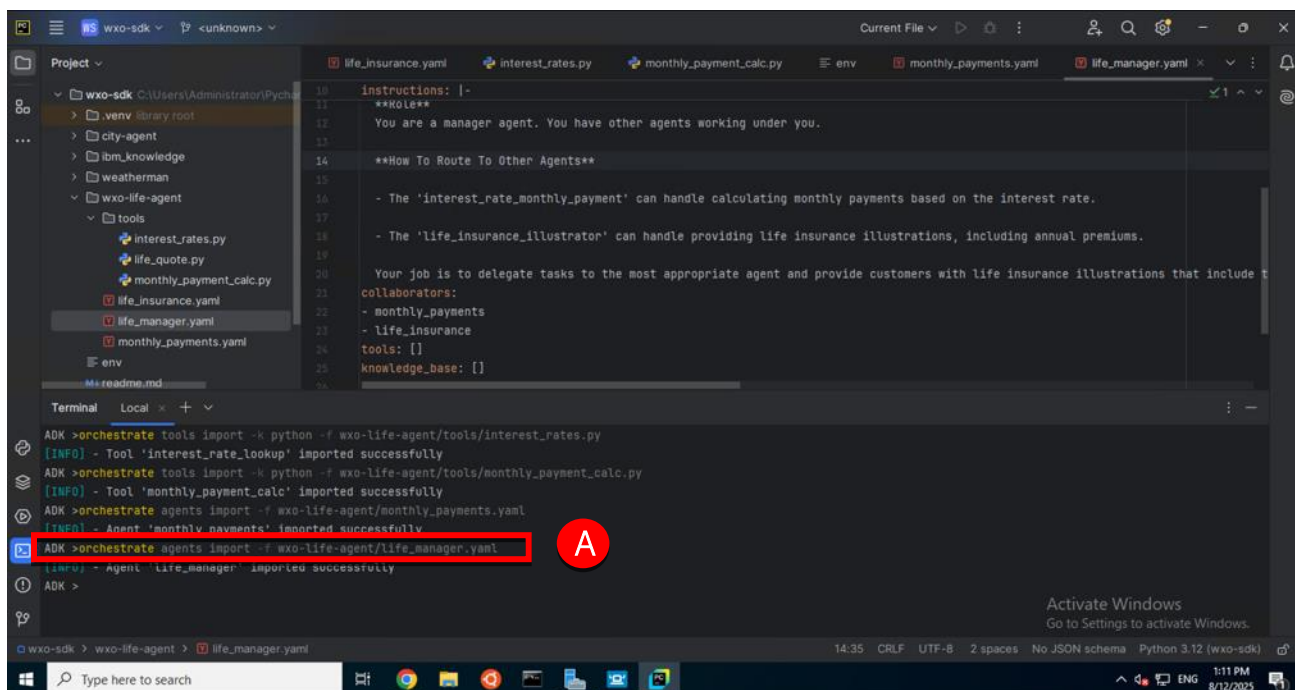
1. Double-click `life_manager` (A).



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

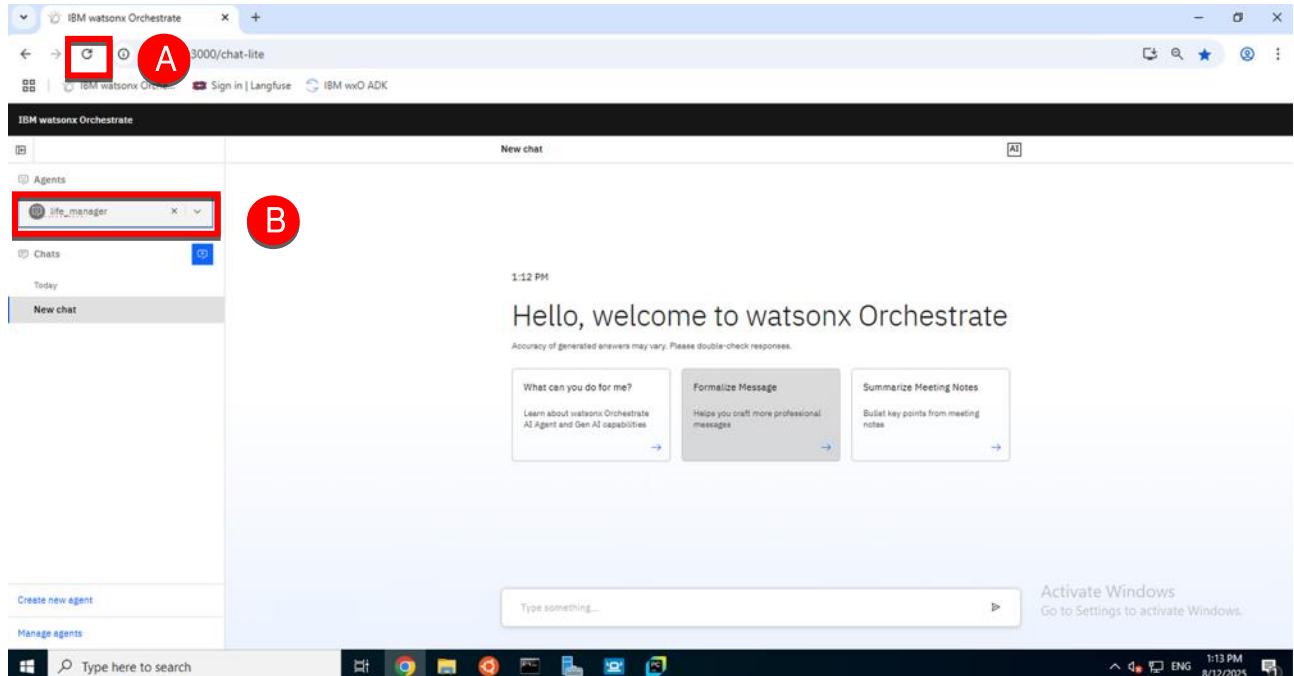
2. Enter the following command into the terminal (A) to add the agent.

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

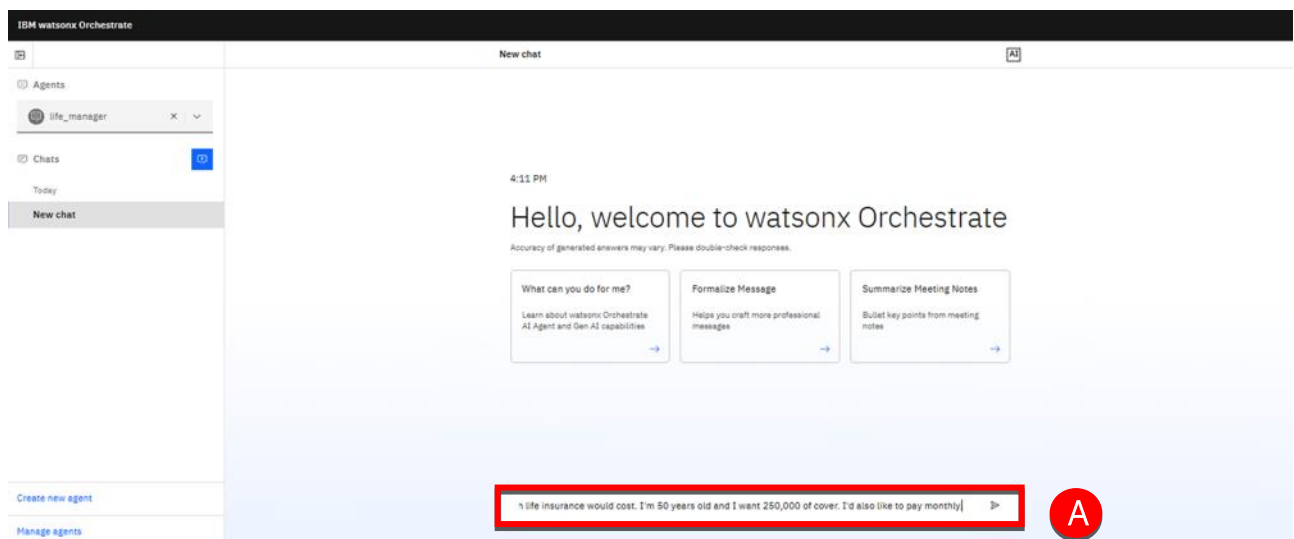


Testing the manager agent

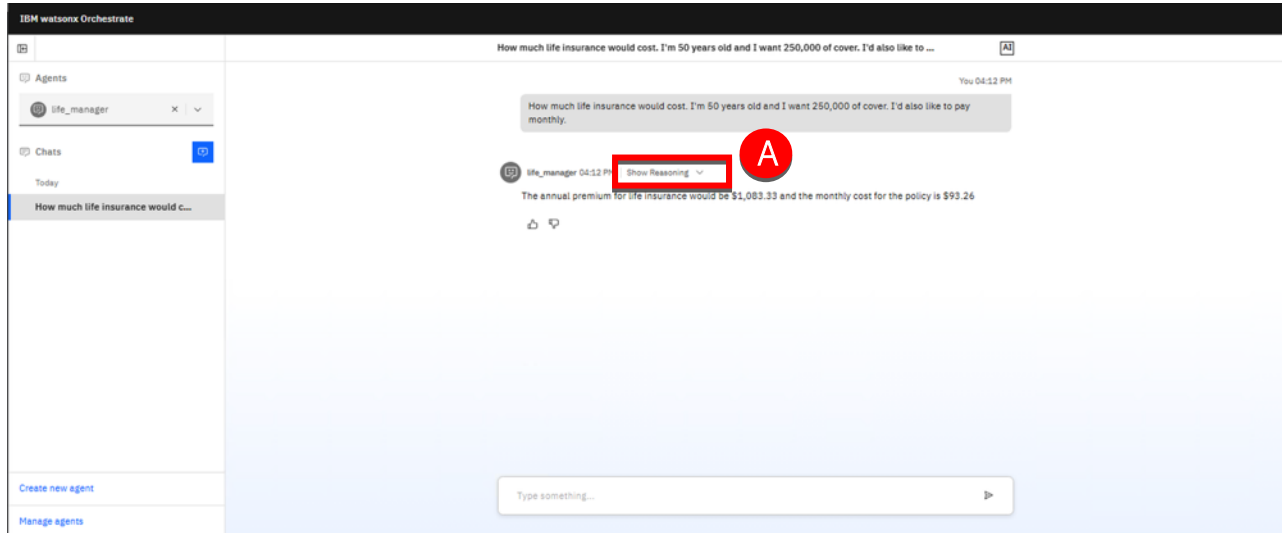
1. Return to the browser window and click refresh (A) and ensure the **life_manager** is selected from the **Agents** list (B).



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.

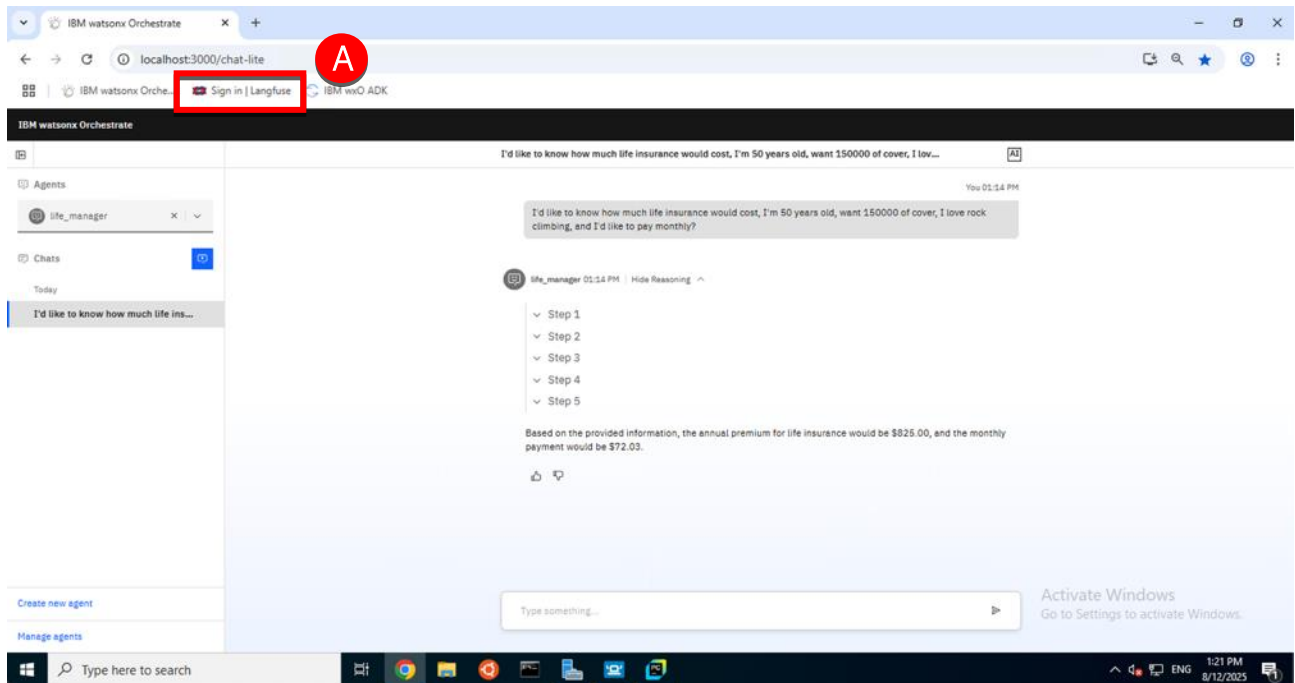
It would also have been possible to achieve the same result by adding the additional tools and instructions into a single top-level agent. However, decomposing this functionality into separate agents keeps the agents smaller, simpler, promotes reuse and should improve the reasoning accuracy of the solution.

Langfuse trace

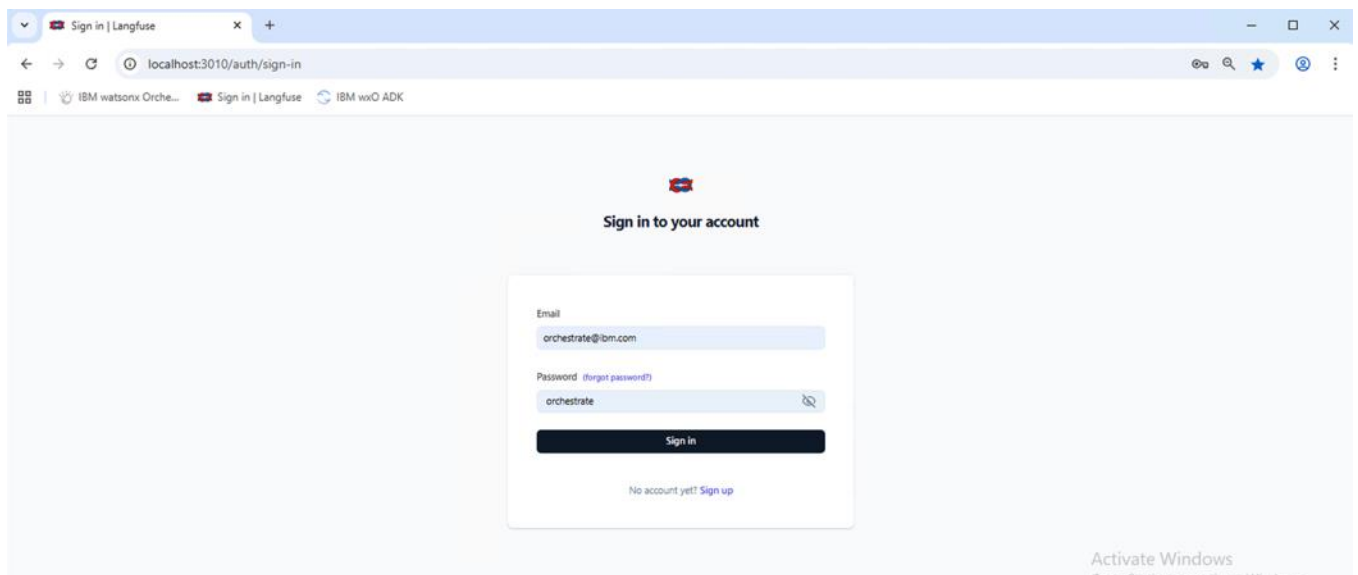
In this section you will review the trace information captured in Langfuse and understand the chain of thought used by the agent to produce the response.

Access the trace

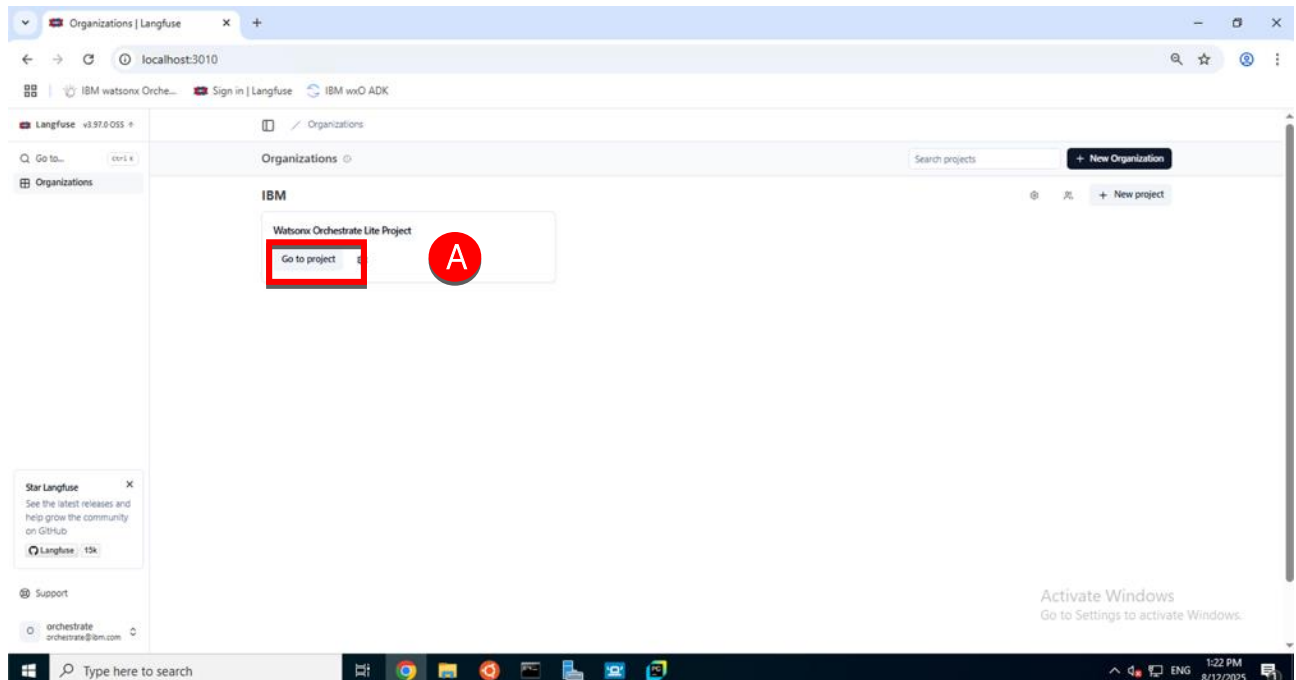
1. Click the **Sign in | Langfuse** bookmark in the browser (A).



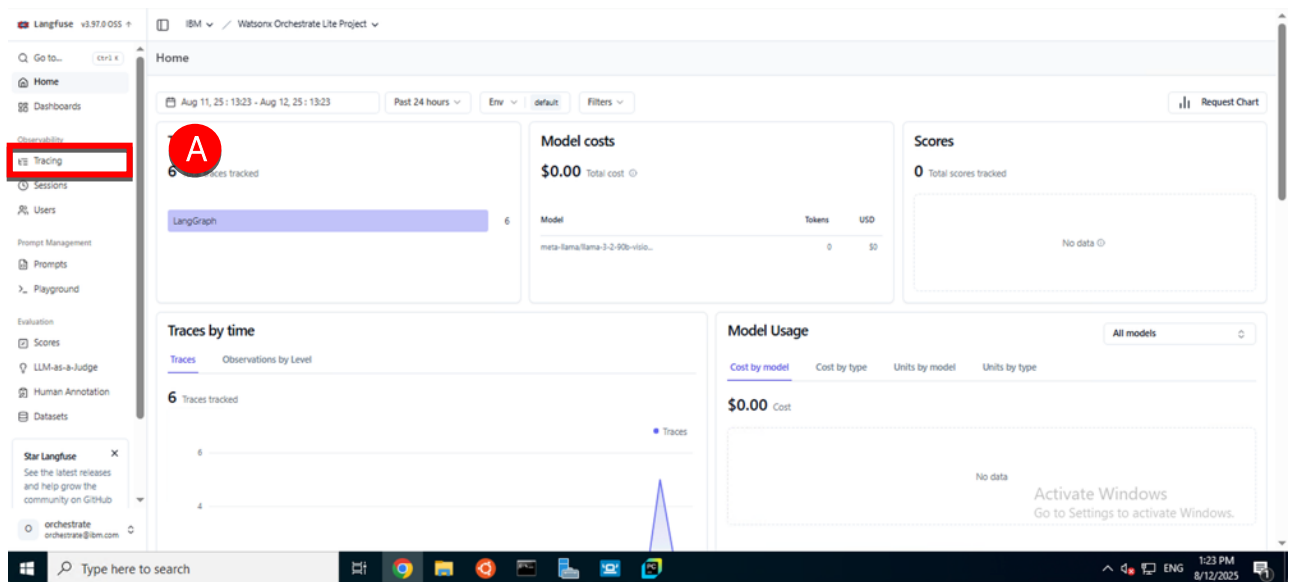
2. The login page should be pre-populated, enter credentials if needed, and click **Sign in** (A).



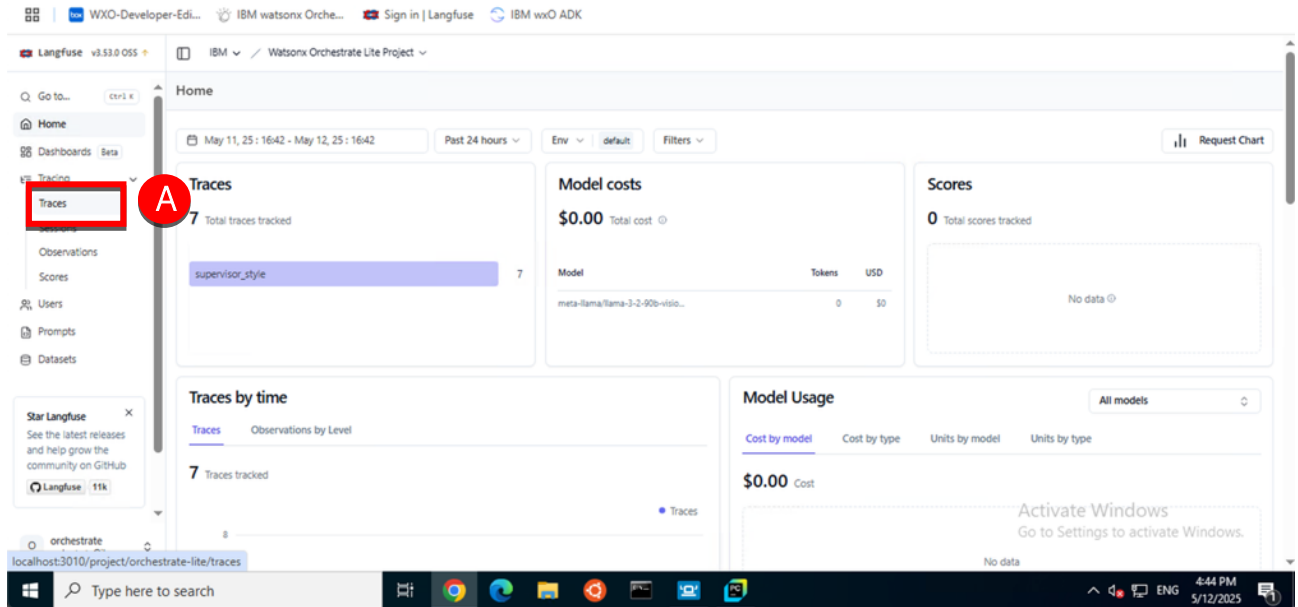
- Click **Go to project** (A).



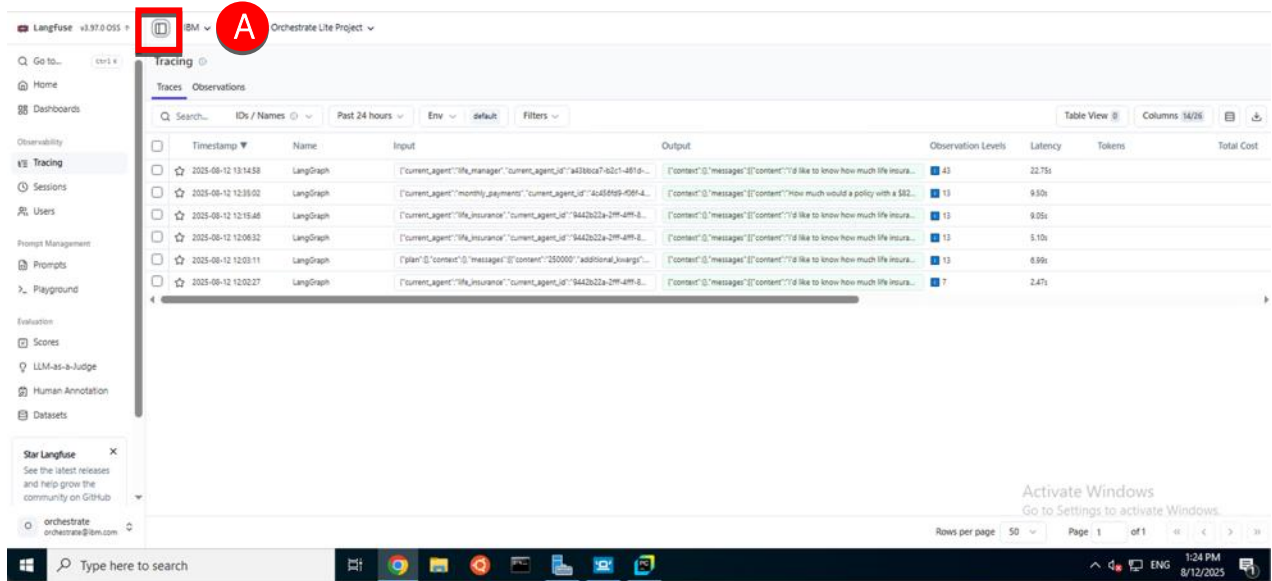
- Click **Tracing** (A).



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.

The screenshot shows the IBM Watsonx Orchestrate Tracing interface. A table lists several traces. The top trace has a timestamp of 2025-08-12 13:14:58, which is highlighted with a red box and labeled 'A'. The table columns include Timestamp, Name, Input, Output, Observation Levels, Latency, Tokens, Total Cost, and Environment.

Timestamp	Name	Input	Output	Observation Levels	Latency	Tokens	Total Cost	Environment
2025-08-12 13:14:58	LangGraph	[{"current_agent": "life_manager", "current_agent_id": "a43bca7-62c1-481a-8..."}]	[{"content": "messages", "content": "I'd like to know how much life insura..."}]	43	22.75s			default
2025-08-12 12:35:02	LangGraph	[{"current_agent": "monthly_payments", "current_agent_id": "a43bca7-62c1-481a-8..."}]	[{"content": "messages", "content": "How much would a policy with a \$62..."}]	13	9.50s			default
2025-08-12 12:15:48	LangGraph	[{"current_agent": "life_insurance", "current_agent_id": "9442b22a-2ff-49f-8..."}]	[{"content": "messages", "content": "I'd like to know how much life insura..."}]	13	9.05s			default
2025-08-12 12:08:32	LangGraph	[{"current_agent": "life_insurance", "current_agent_id": "9442b22a-2ff-49f-8..."}]	[{"content": "messages", "content": "I'd like to know how much life insura..."}]	13	5.10s			default
2025-08-12 12:03:11	LangGraph	[{"plan": "content", "messages": [{"content": "250000", "additional_info": "..."}]}]	[{"content": "messages", "content": "I'd like to know how much life insura..."}]	13	6.99s			default
2025-08-12 12:02:27	LangGraph	[{"current_agent": "life_insurance", "current_agent_id": "9442b22a-2ff-49f-8..."}]	[{"content": "messages", "content": "I'd like to know how much life insura..."}]	7	2.47s			default

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

The screenshot shows the IBM Watsonx Orchestrate Tracing interface with a trace expanded in full screen. The expand icon is highlighted with a red box and labeled 'A'. The expanded view shows the trace details, including the input, output, and a flow diagram of the LangGraph process.

LangGraph ID: 2025-08-12 13:14:58.173
Session: bca5129-74c2-40a8-8b63-a1d554707a8b
Env: default
Latency: 22.75s

Preview

Input

Path	Value
current_agent	"life_manager"
current_agent_id	"a43bca7-62c1-481a-8b63-a1d554707a8b"
agent_display_name	"life_manager"
messages	[{"content": "..."}]
> 0	7 times
reflection_retry_count	0
reflection_retry_limit	1
content	empty object
last_reflection	null
step_count	0
plan	empty list
current_task	-1
task_results	empty object
is_planning	false
is_collaborator	false
guideline_prompt	""
citations	empty object

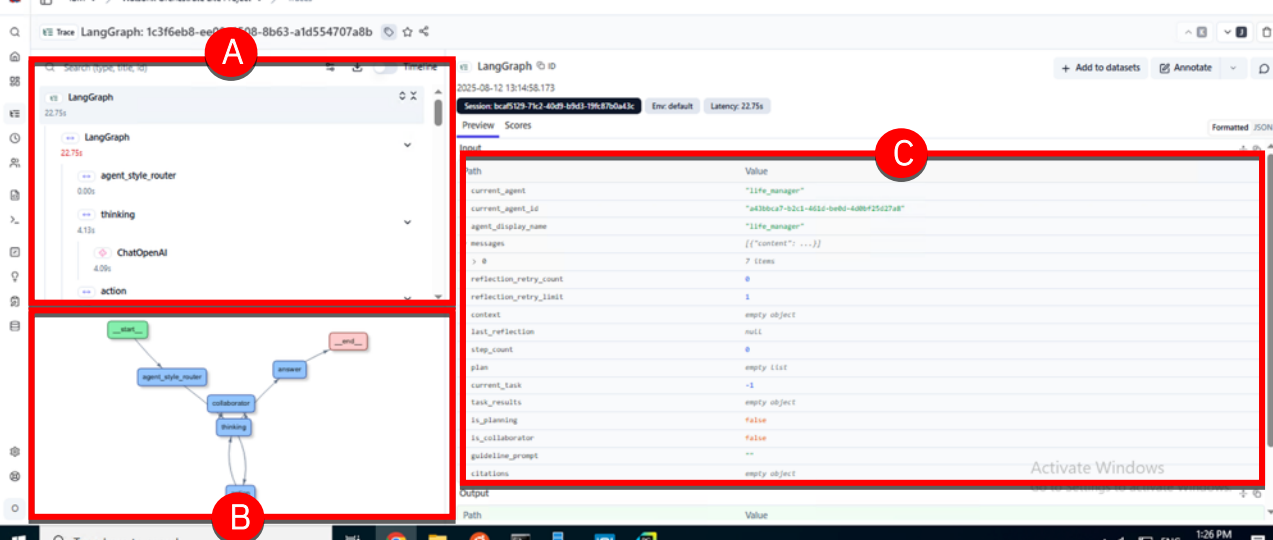
Output

Path	Value
Path	Value

Metadata

Analyze the trace

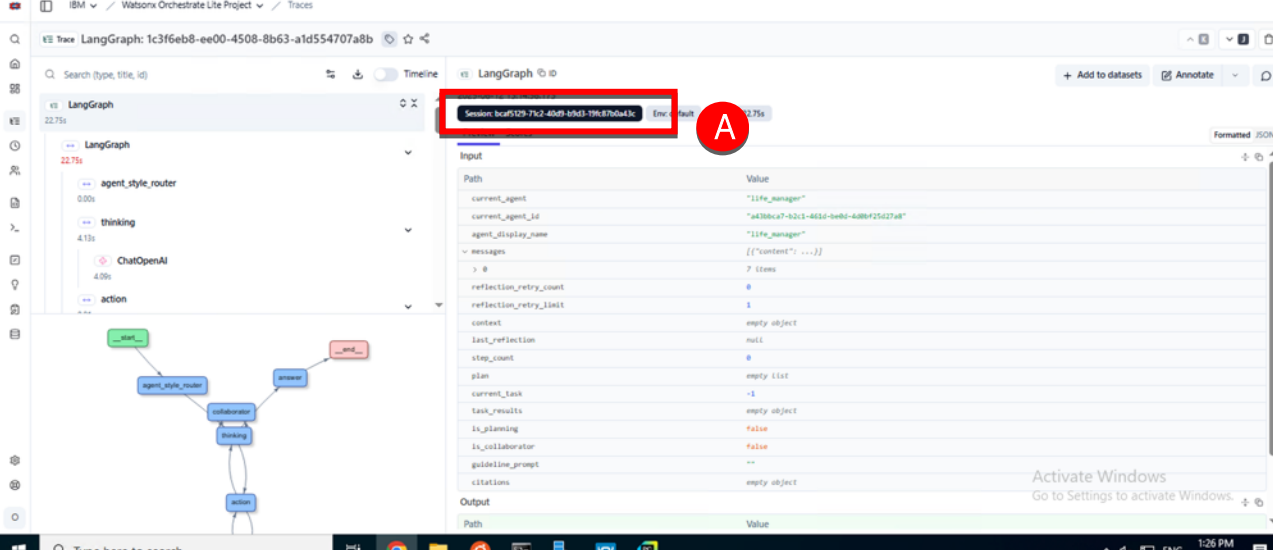
1. Review the Langfuse trace display containing the trace (A), the agent graph (B) and the trace detail (C).



The screenshot shows the Langfuse trace display for a session. The interface is divided into three main sections:

- Trace (A):** The top left section shows the trace timeline with steps: `LangGraph` (22.75s), `agent_style_router` (0.00s), `thinking` (4.13s), `ChatOpenAI` (4.09s), and `action`.
- Agent Graph (B):** The bottom left section shows a flow diagram of the agent's state transitions. It includes nodes for `start`, `agent_style_router`, `collaborator`, `thinking`, `action`, and `end`.
- Trace Detail (C):** The right section shows the detailed JSON output of the trace. It includes fields like `current_agent`, `current_agent_id`, `agent_display_name`, `messages`, `reflection_retry_count`, `reflection_retry_limit`, `context`, `last_reflection`, `step_count`, `plan`, `current_task`, `task_results`, `is_planning`, `is_collaborator`, `guideline_prompt`, and `citations`.

2. Note the Session ID (A).



The screenshot shows the Langfuse trace display for a session. The interface is divided into three main sections:

- Session ID (A):** The top right section shows the session ID, which is highlighted with a red box and labeled 'A'. The session ID is `bc9f129-7a2-40b8-b63-a1d554707a8b`.
- Trace (A):** The top left section shows the trace timeline with steps: `LangGraph` (22.75s), `agent_style_router` (0.00s), `thinking` (4.13s), `ChatOpenAI` (4.09s), and `action`.
- Agent Graph (B):** The bottom left section shows a flow diagram of the agent's state transitions. It includes nodes for `start`, `agent_style_router`, `collaborator`, `thinking`, `action`, and `end`.
- Trace Detail (C):** The right section shows the detailed JSON output of the trace. It includes fields like `current_agent`, `current_agent_id`, `agent_display_name`, `messages`, `reflection_retry_count`, `reflection_retry_limit`, `context`, `last_reflection`, `step_count`, `plan`, `current_task`, `task_results`, `is_planning`, `is_collaborator`, `guideline_prompt`, and `citations`.

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).

The screenshot shows the IBM Watsonx Orchestrator interface. On the left, the 'Trace' panel (A) lists various components: 'agent_style_router', 'thinking', 'ChatOpenAI' (highlighted with a red box and labeled B), 'action', 'life_insurance', and 'collaborator'. Below this is a flow diagram. On the right, the 'ChatOpenAI' detail panel (C) displays the 'System' prompt. The prompt includes instructions on how to handle user questions, use tools, and route tasks to other agents. A red arrow points from the 'ChatOpenAI' entry in the trace panel to its detail view.

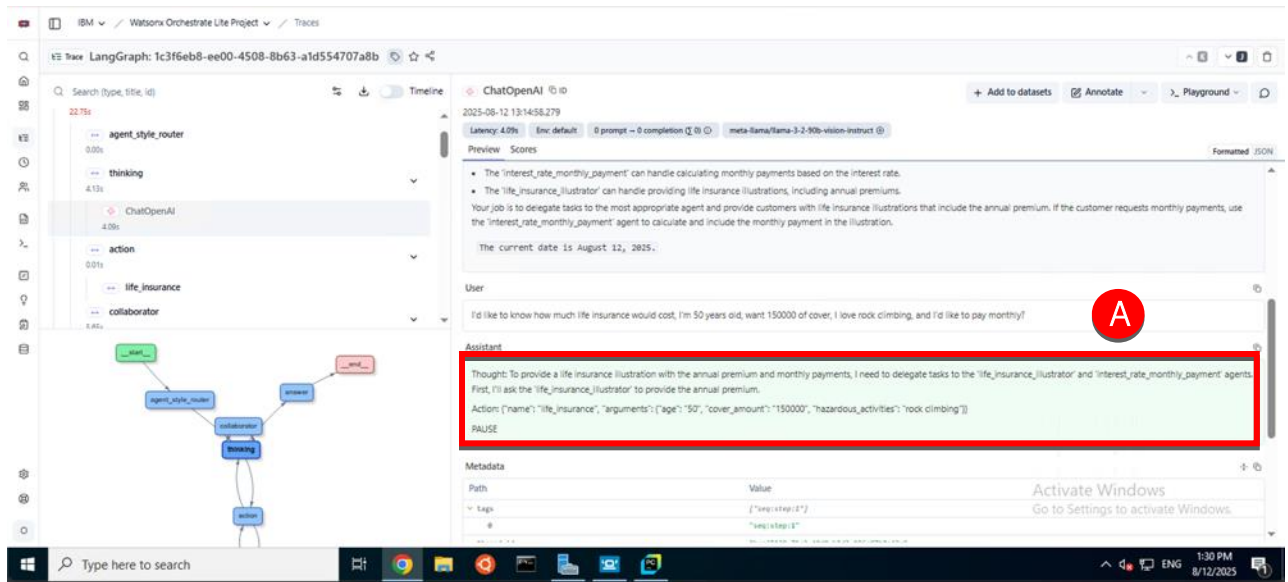
This prompt determines how the LLM used by the agent will approach the problem. In addition to the general instructions the prompt also includes information about the collaborators that are available to the agent.

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

The screenshot shows the IBM Watsonx Orchestrator interface. On the left, the 'Trace' panel (A) lists various components: 'agent_style_router', 'thinking', 'ChatOpenAI' (highlighted with a red box and labeled B), 'action', 'life_insurance', and 'collaborator'. Below this is a flow diagram. On the right, the 'ChatOpenAI' detail panel (C) displays the 'User' and 'Assistant' panels. The 'User' panel (B) shows the user's input: 'I'd like to know how much life insurance would cost, I'm 50 years old, want 150000 of cover, I love rock climbing, and I'd like to pay monthly?'. The 'Assistant' panel (C) shows the agent's response, including a 'Thought' section, an 'Action' section, and a 'PAUSE' status. A red arrow points from the 'ChatOpenAI' entry in the trace panel to its detail view.

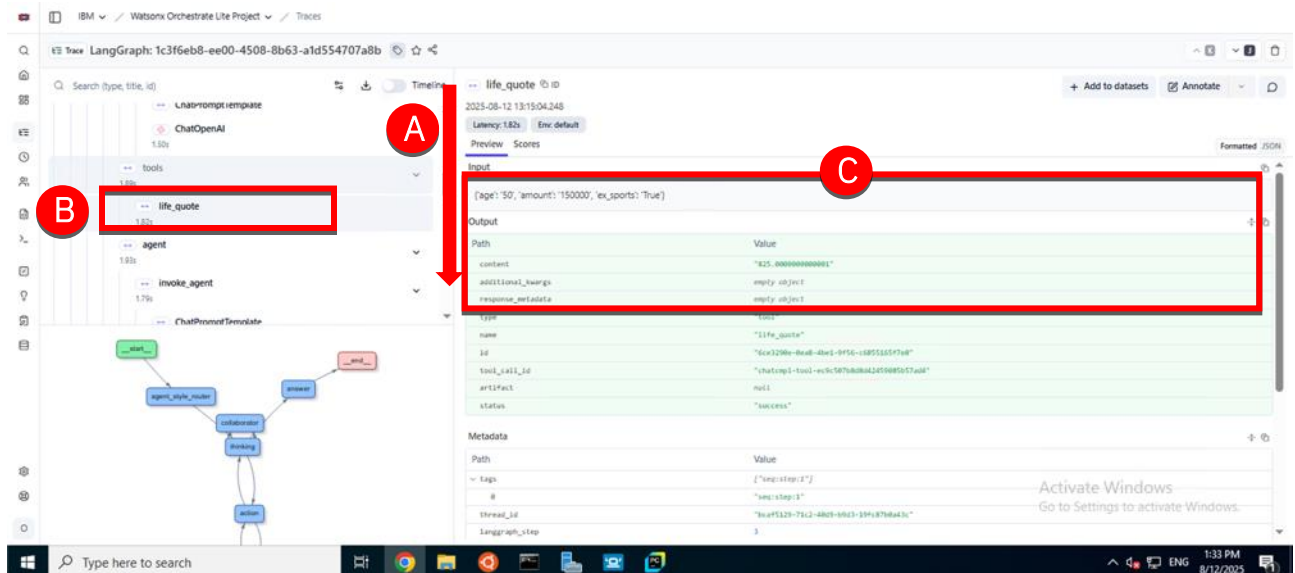
The User panel displays the text entered by the user.

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).

The screenshot shows the IBM Watsonx Orchestrate interface. The Trace panel (A) is on the left, displaying a flowchart with nodes like 'agent_style_router', 'subrouter', 'thinking', and 'action'. The ChatOpenAI element (B) is highlighted with a red box. The right panel shows the system prompt and instructions for the ChatOpenAI model.

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

The screenshot shows the IBM Watsonx Orchestrate interface. The Trace panel (A) is on the left, displaying a flowchart with nodes like 'agent_style_router', 'subrouter', 'thinking', and 'action'. The ChatOpenAI element (B) is highlighted with a red box. The right panel shows the tool panel (B) and assistant panel (C) with the output of the monthly payment calculation.

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.

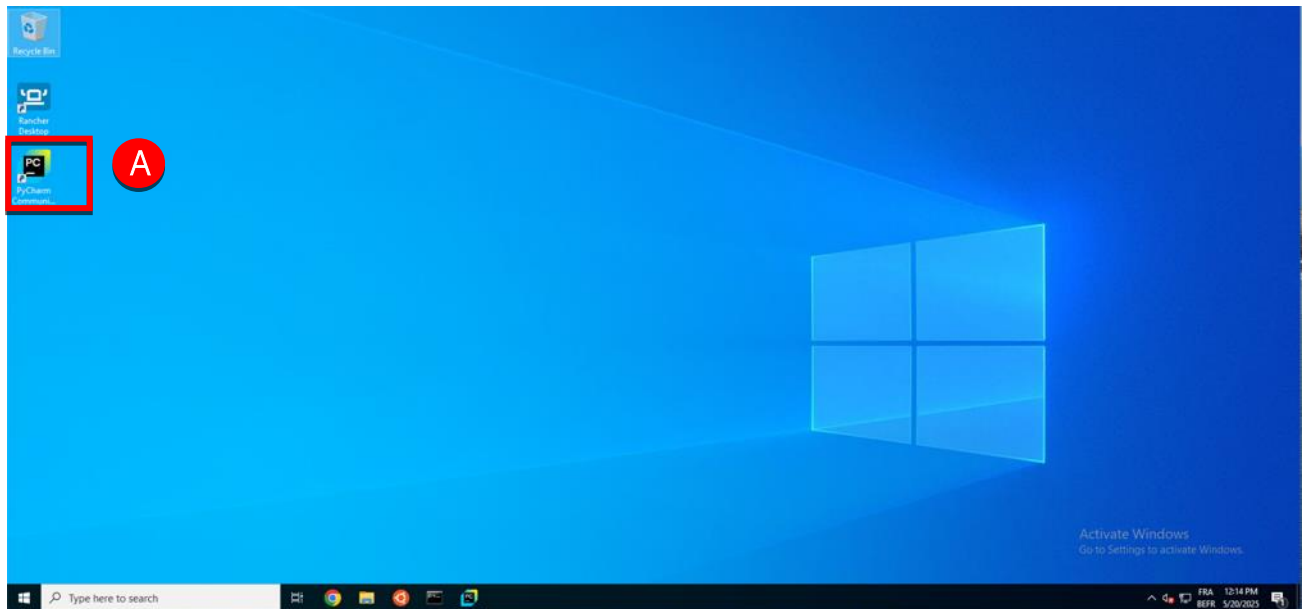
LLM Error

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.

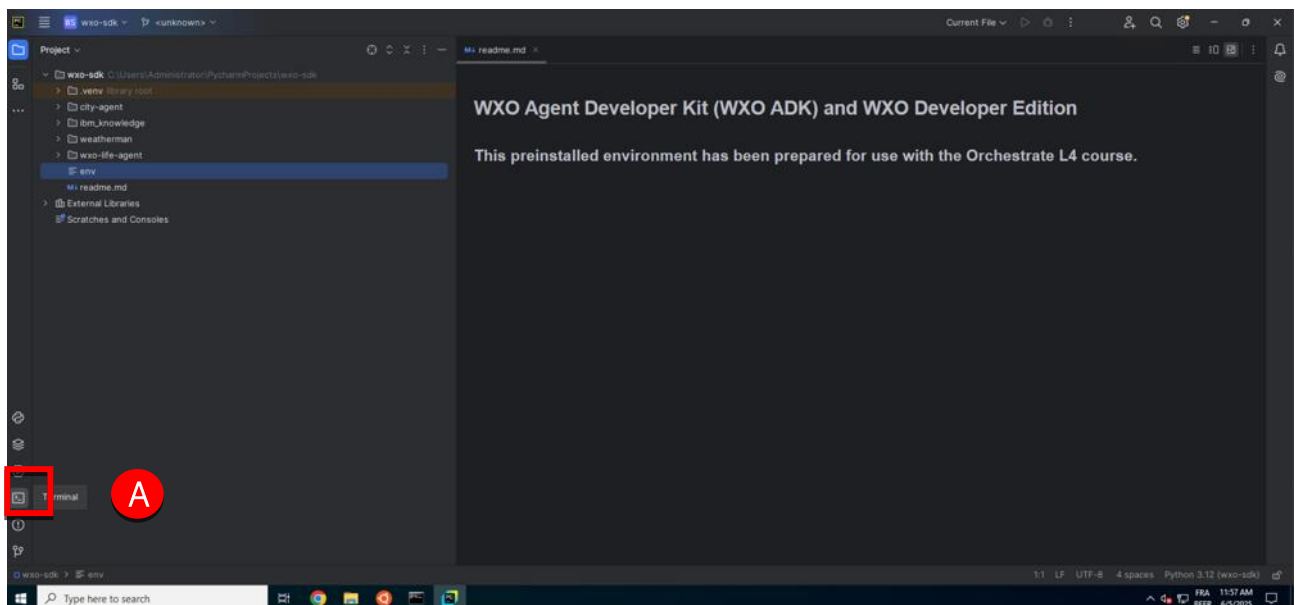
Appendix: Reset your environment

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.)

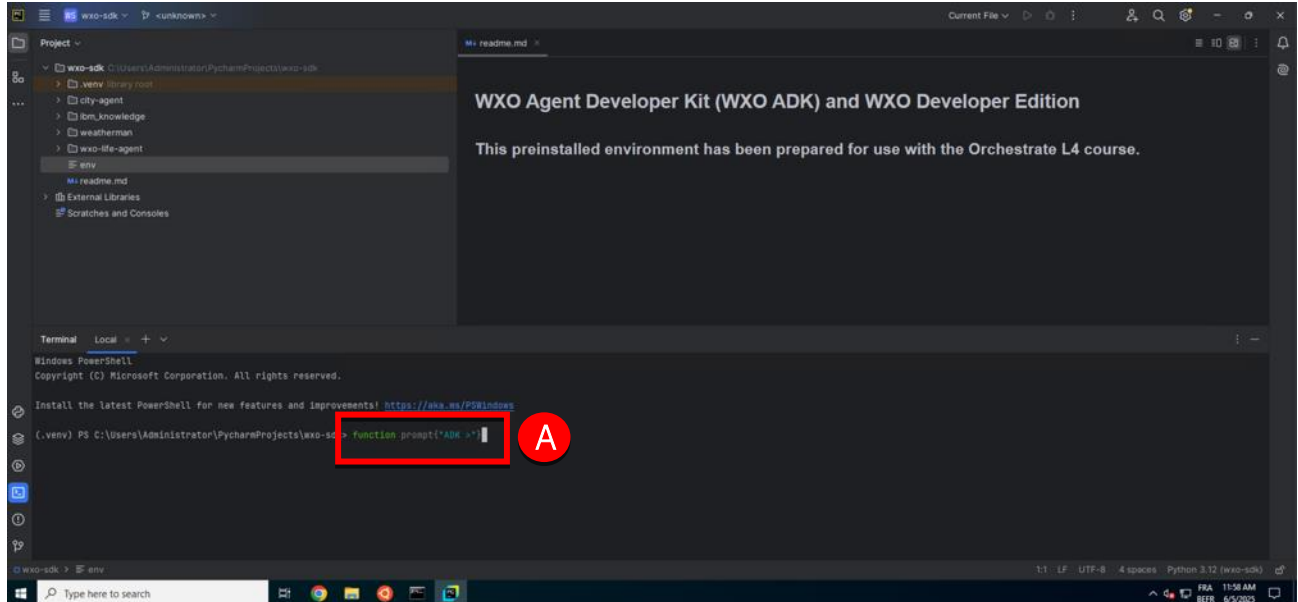


3. Click the terminal icon (A), to open the terminal (if required).



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
```

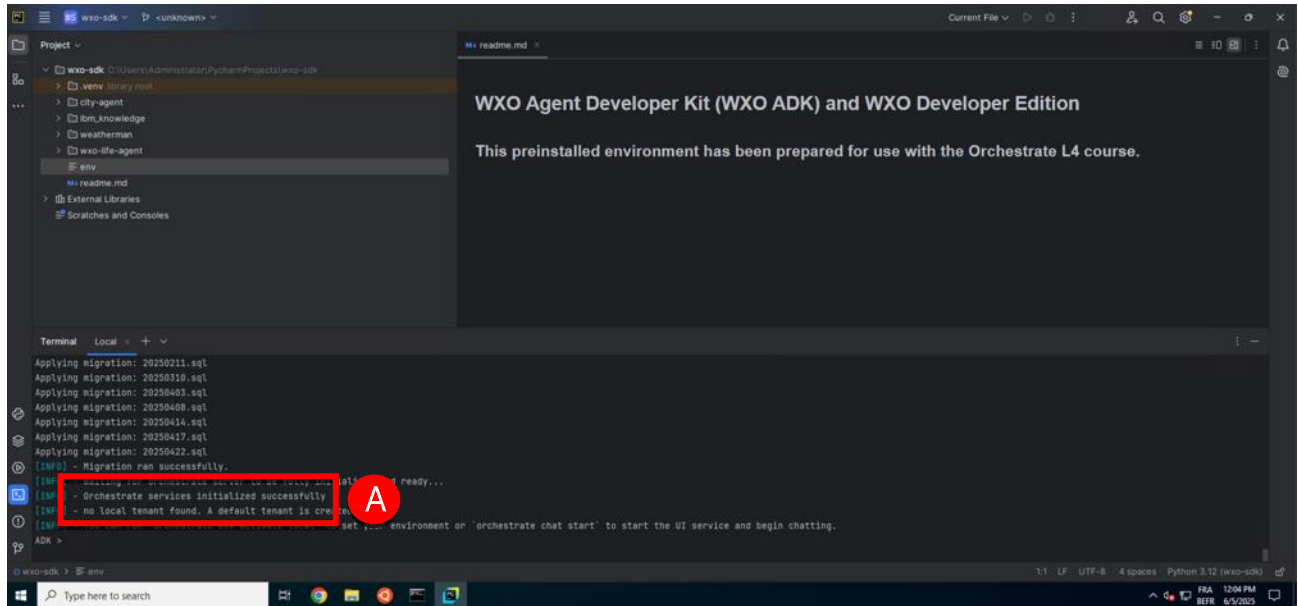
6. Enter the following command into the terminal to reset your server, then press Enter.

```
orchestrate server reset
```

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

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

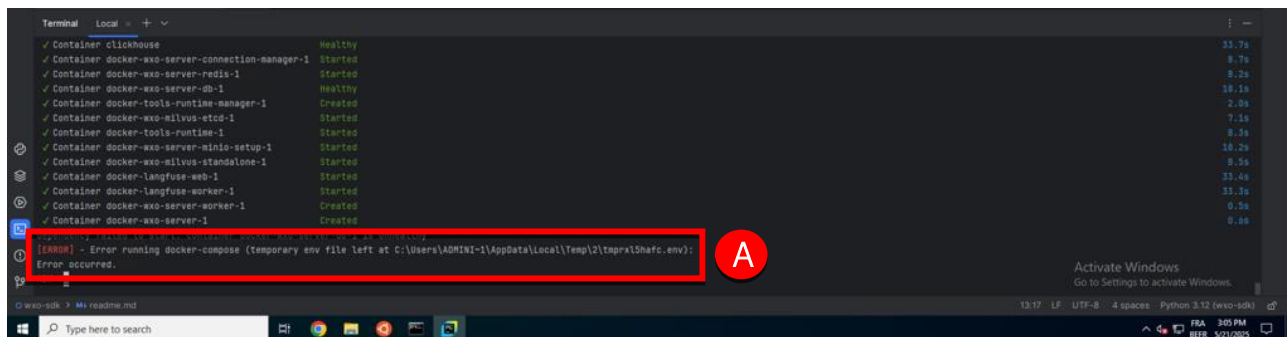
8. When restart is complete (A) the terminal output will resemble the example below, restart can take 2-3 minutes to complete.



```
Project - wso-sdk C:\Users\Administrator\PycharmProjects\wso-sdk
  .venv library root
  .city-agent
  .ibm_knowledge
  .weatherman
  .wso-life-agent
  .env
  .readme.md
  External Libraries
  Scratches and Consoles

Terminal Local +
Applying migration: 20250211.sql
Applying migration: 20250310.sql
Applying migration: 20250403.sql
Applying migration: 20250408.sql
Applying migration: 20250414.sql
Applying migration: 20250417.sql
Applying migration: 20250422.sql
[INFO] - Migration ran successfully.
[INFO] - Creating the WSO ADK environment... ready...
[INFO] - Orchestrate services initialized successfully
[INFO] - no local tenant found. A default tenant is created.
[INFO] - set the environment or 'orchestrate chat start' to start the UI service and begin chatting.
ADK >
```

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



```
Terminal Local +
Container clickhouse Healthy 33.7s
Container docker-wso-server-connection-manager-1 Started 8.7s
Container docker-wso-server-redis-1 Started 8.2s
Container docker-wso-server-db-1 Healthy 16.1s
Container docker-tools-runtime-manager-1 Created 2.0s
Container docker-wso-milvus-etcd-1 Started 7.1s
Container docker-tools-runtime-1 Started 8.3s
Container docker-wso-server-minio-setup-1 Started 16.2s
Container docker-wso-milvus-standalone-1 Started 8.5s
Container docker-langfuse-web-1 Started 33.4s
Container docker-langfuse-worker-1 Started 33.3s
Container docker-wso-server-worker-1 Created 0.5s
Container docker-wso-server-1 Created 0.6s
[ERROR] - Error running docker-compose (temporary env file left at C:\Users\ADMINI-1\AppData\Local\Temp\2\temp13hafc.env):
Error occurred.
```