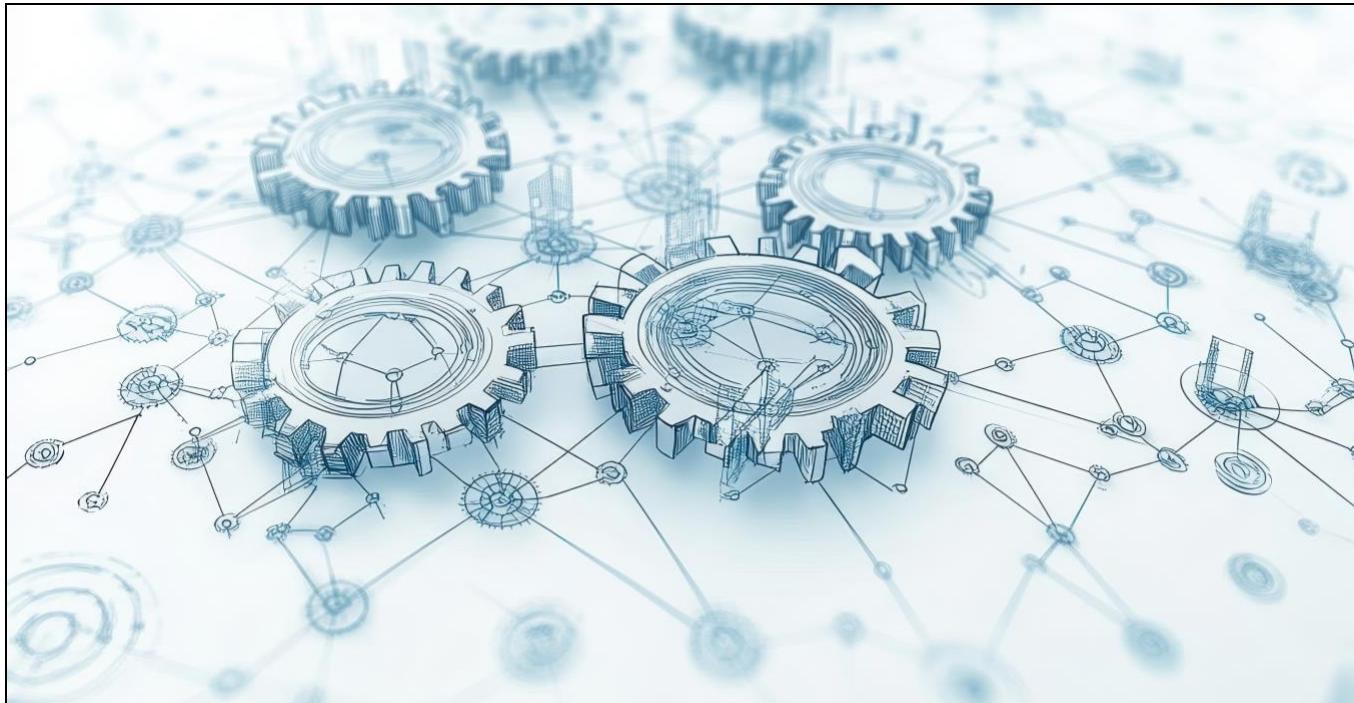


Level 4

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](#) 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.

The screenshot shows the IBM Watsonx Orchestrate interface. On the left, there's a sidebar with 'Agents' (life_manager selected), 'Chats' (Today), and buttons for 'Create new agent' and 'Manage agents'. The main area is a chat window. The user message is: "How much life insurance would cost. I'm 50 years old and I want 250,000 of cover. I'd also like to ...". The agent response is: "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." Below this, the agent adds: "The annual premium for life insurance would be \$1083.33. The monthly cost for the policy is \$93.71". At the bottom, there's a text input field with "Type something..." and a send button.

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:

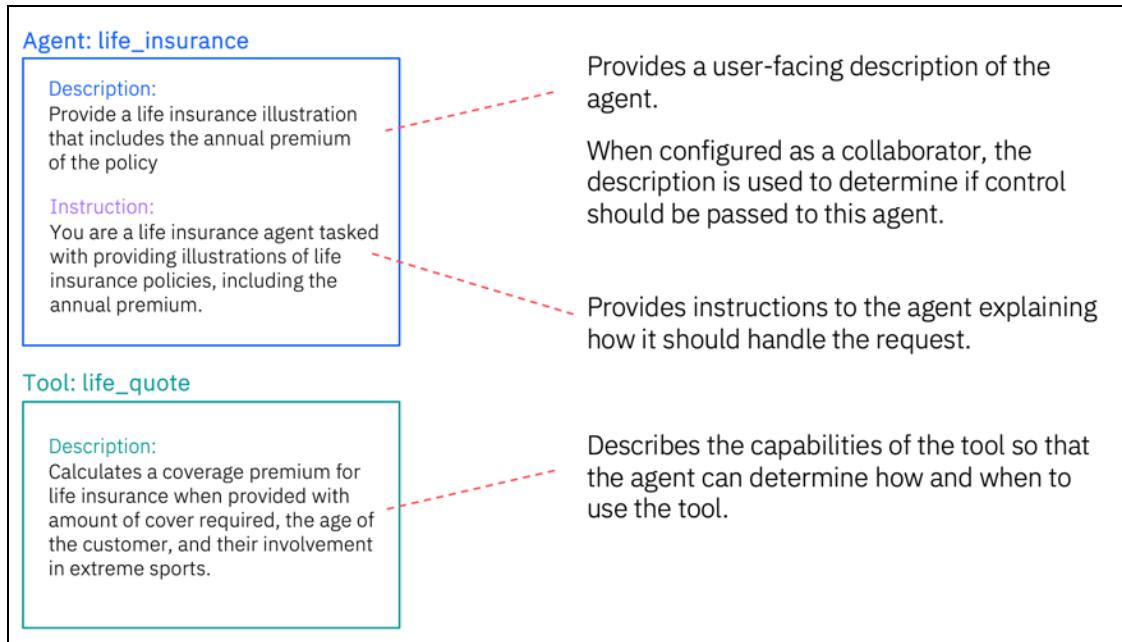
- You have performed the Preparation Lab guide including the environment validation step
- You have your ADK environment up and running (Orchestrate server started after validating the environment in the preparation guide OR after having performed another ADK related lab)

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.

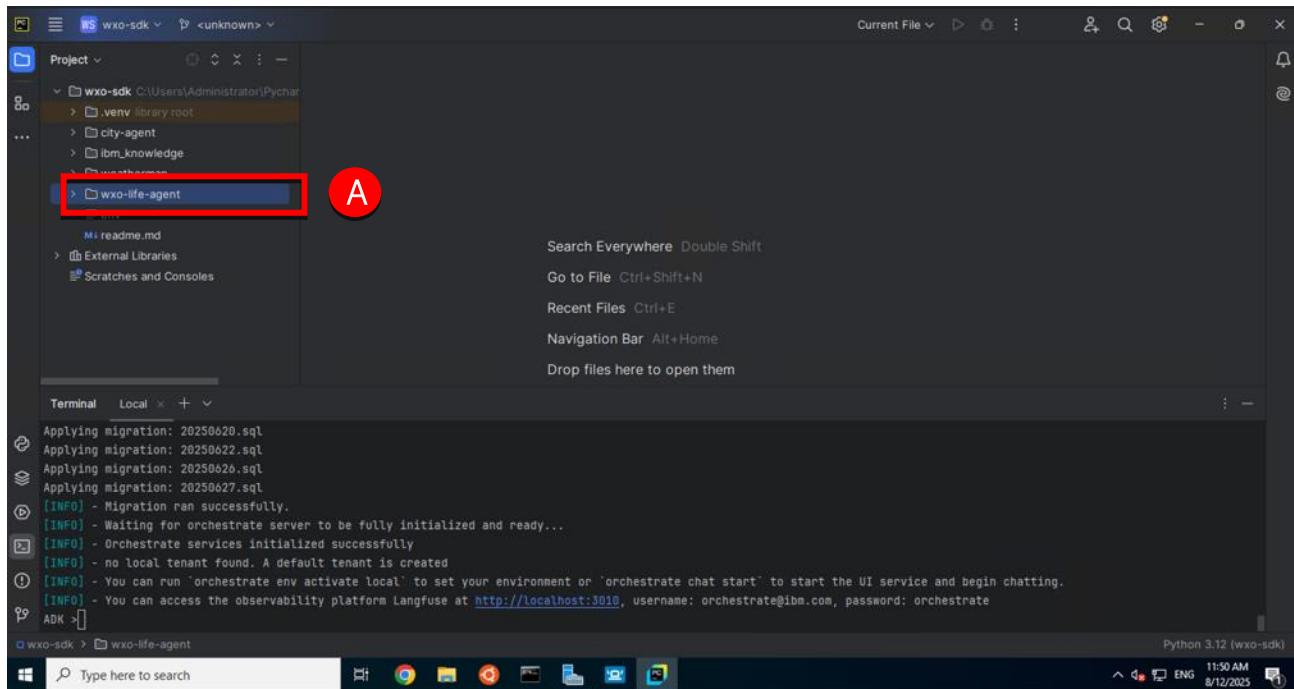


In addition to the tool description, any input and output parameters used by the tool are analyzed and added to the tool configuration. At runtime, the agent shares this configuration as part of the system prompt and is used for planning.

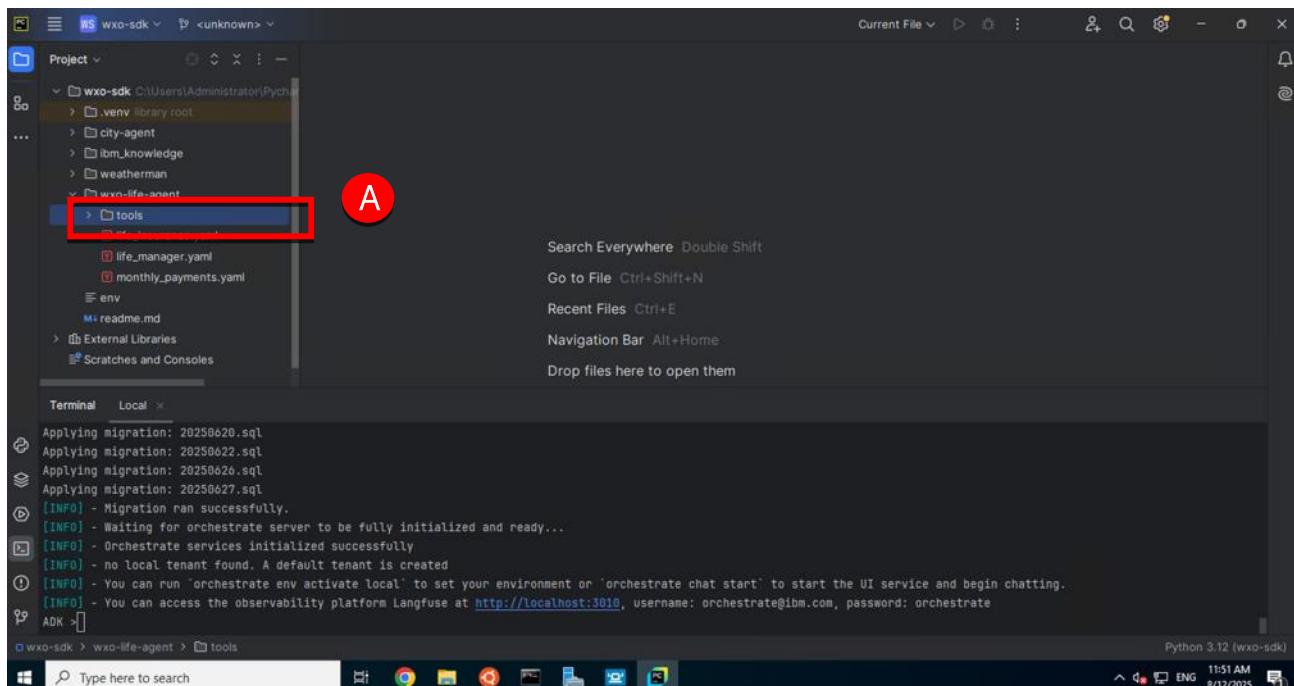
Tool and agent creation

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.

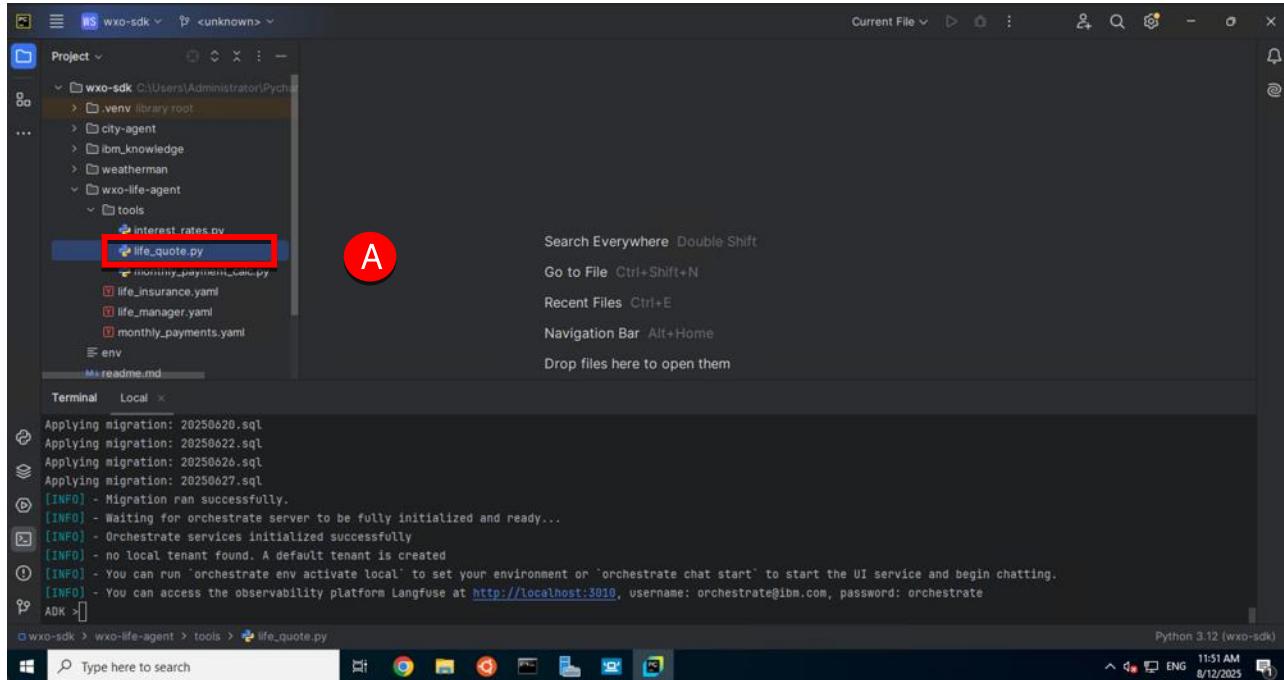
1. Open the ADK environment and start the watsonx Orchestrate server if not yet done. Refer to the [Reset your environment appendix](#) if your orchestrate server is not started.
1. Double-click the **wxo-life-agent** folder (A).



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 screenshot shows the PyCharm IDE interface with the code editor open to the 'life_quote.py' file. The code defines a function 'life_quote' with an annotation '@tool'. The code is as follows:

```

import ...

@tool(name="life_quote",
      description="Calculates a coverage premium for life insurance when provided with amount of cover required and the age of the insured")
def life_quote(amount:int, age:int):

    # the baseline cost for cover is 0.1% of the cover amount
    base_cost = amount * 0.001
    age_factor = (age / 1.5) * 0.1
    age_supplement = base_cost * age_factor

    quote = base_cost + age_supplement
    return quote

```

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.

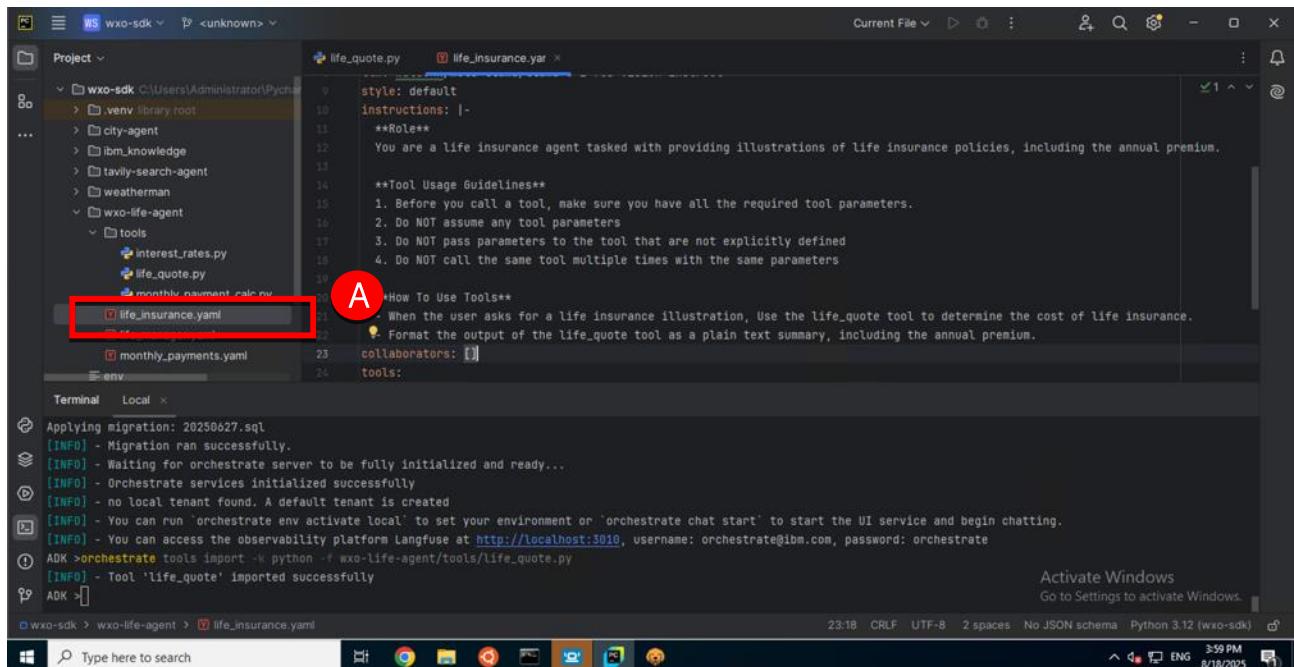
The function receives two parameters, the **amount** of cover required, and the **age** of the customer. When the tool is created, information about these parameters will be added to the configuration and available to any agent using this tool.

- 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
```

A screenshot of a Windows desktop environment. On the left is a file explorer window showing a project structure under 'wxo-sdk'. The 'wxo-life-agent/tools' folder contains several Python files: 'interest_rates.py', 'life_quote.py', 'monthly_payment_calc.py', 'life_insurance.yaml', 'life_manager.yaml', and 'monthly_payments.yaml'. A red box highlights the terminal window on the right, which shows the command 'ADK >orchestrate tools import -k python -f wxo-life-agent/tools/life_quote.py'. The terminal output shows migration logs and orchestrate service initialization messages. A red circle with the letter 'A' is positioned over the terminal window, pointing to the command line area.

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



```

Project : life_quote.py life_insurance.yaml
wxo-sdk
  .venv
    library root
    city-agent
    ibm_knowledge
    tavity-search-agent
    weatherman
  wxo-life-agent
    tools
      interest_rates.py
      life_quote.py
      monthly_payment_calc.py
      life_insurance.yaml
      life_manager.yaml
      monthly_payments.yaml
  env
  env

style: default
instructions: |-
  **Role**
  You are a life insurance agent tasked with providing illustrations of life insurance policies, including the annual premium.

  **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 asks for a life insurance illustration, Use the life_quote tool to determine the cost of life insurance.
  Format the output of the life_quote tool as a plain text summary, including the annual premium.

collaborators: []
tools:

Terminal Local x
Applying migration: 20250627.sql
[INFO] - Migration ran successfully.
[INFO] - Waiting for orchestrate server to be fully initialized and ready...
[INFO] - Orchestrator services initialized successfully
[INFO] - no local tenant found. A default tenant is created
[INFO] - You can run 'orchestrate env activate local' to set your environment or 'orchestrate chat start' to start the UI service and begin chatting.
[INFO] - You can access the observability platform Langfuse at http://localhost:3010, username: orchestrate@ibm.com, password: orchestrate
ADK >orchestrate tools import -k python -f wxo-life-agent/tools/life_quote.py
[INFO] - Tool 'life_quote' imported successfully
ADK >
ADK >

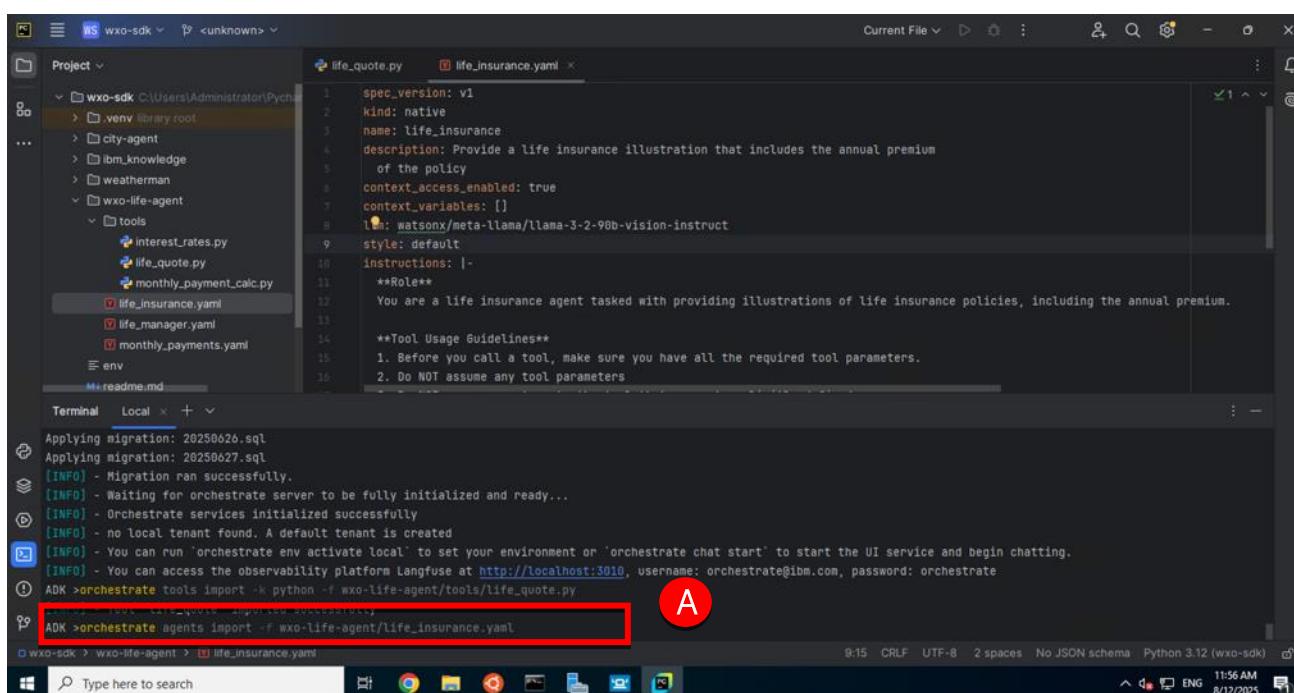
23:18 CRLF UTF-8 2 spaces No JSON schema Python 3.12 (wxo-sdk)
Type here to search 3:59 PM ENG 8/18/2025

```

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
```



```

Project : life_quote.py life_insurance.yaml
wxo-sdk
  .venv
    library root
    city-agent
    ibm_knowledge
    tavity-search-agent
    weatherman
  wxo-life-agent
    tools
      interest_rates.py
      life_quote.py
      monthly_payment_calc.py
      life_insurance.yaml
      life_manager.yaml
      monthly_payments.yaml
  env
  env

spec_version: v1
kind: native
name: life_insurance
description: Provide a life insurance illustration that includes the annual premium
of the policy
context_access_enabled: true
context_variables: []
l10n: watsonx/meta-llama/llama-3-2-90b-vision-instruct
style: default
instructions: |-
  **Role**
  You are a life insurance agent tasked with providing illustrations of life insurance policies, including the annual premium.

  **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

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

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

23:18 CRLF UTF-8 2 spaces No JSON schema Python 3.12 (wxo-sdk)
Type here to search 11:56 AM ENG 8/12/2025

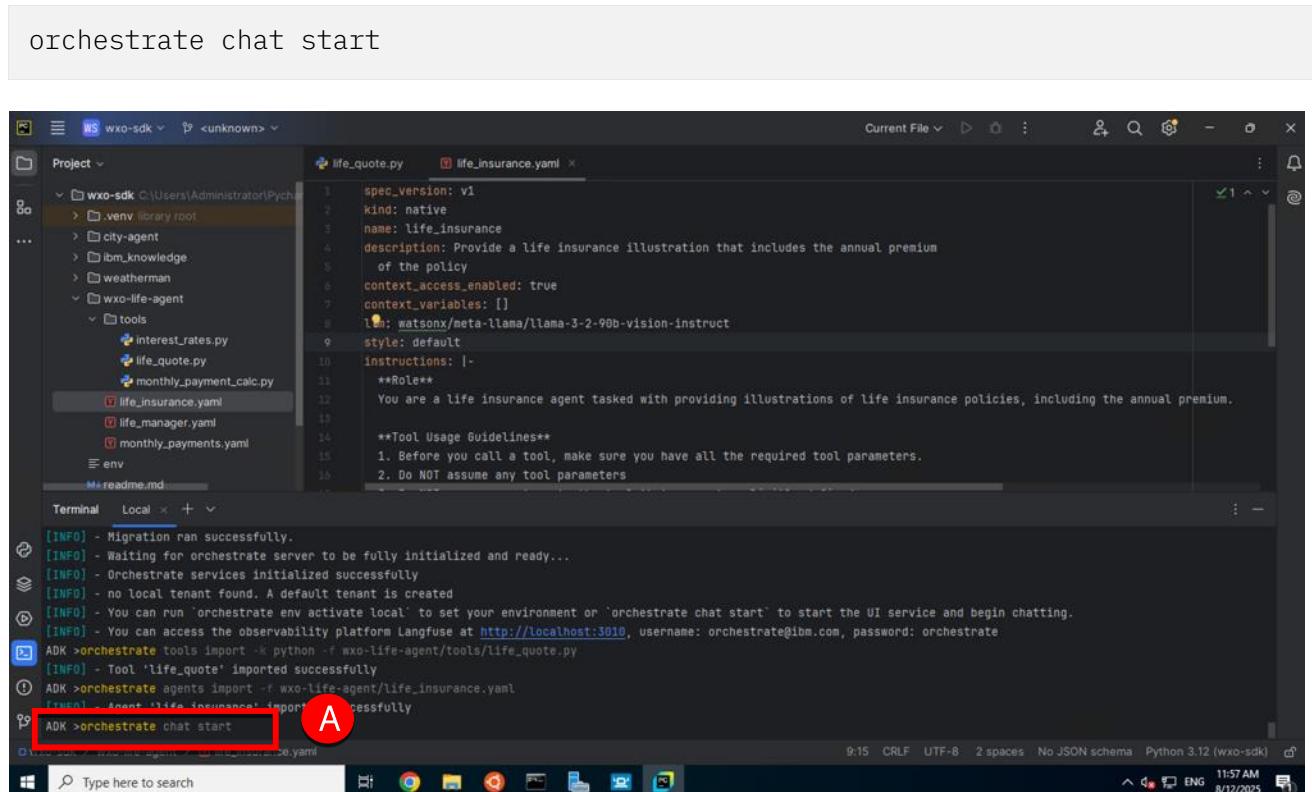
```

Agent testing

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

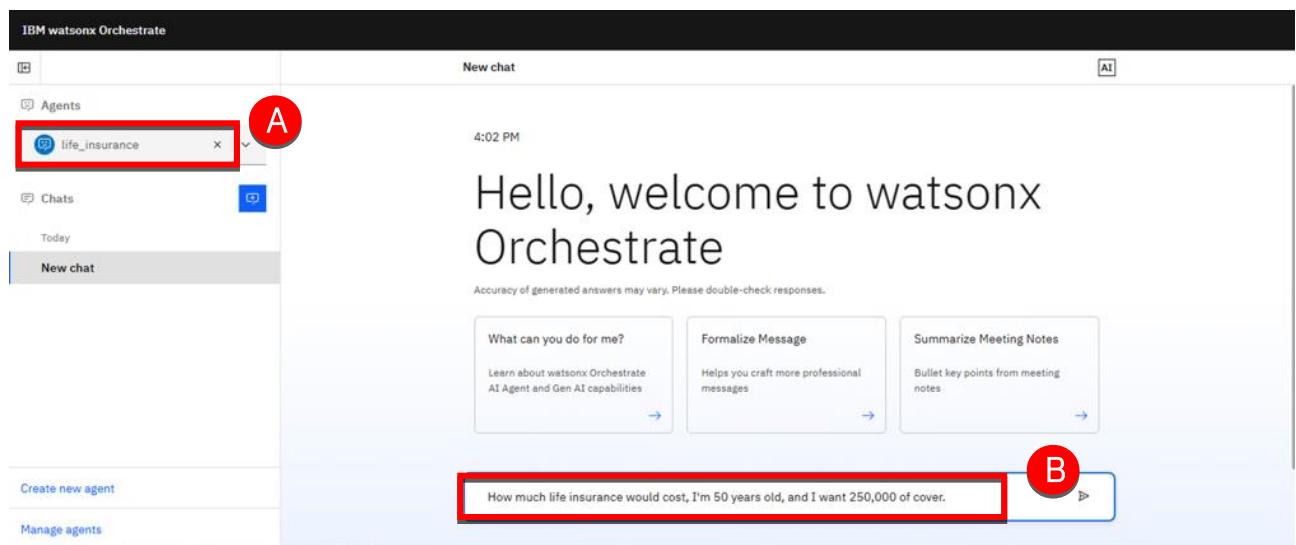
- Enter the following command into the terminal (A) to start the chat server.

```
orchestrate chat start
```



```
[INFO] - Migration ran successfully.
[INFO] - Waiting for orchestrate server to be fully initialized and ready...
[INFO] - Orchestrator services initialized successfully
[INFO] - no local tenant found. A default tenant is created
[INFO] - You can run 'orchestrate env activate local' to set your environment or 'orchestrate chat start' to start the UI service and begin chatting.
[INFO] - You can access the observability platform Langfuse at http://localhost:3010, username: orchestrate@ibm.com, password: orchestrate
ADK >orchestrator tools import -r wxo-sdk/tools/life_quote.py
[INFO] - Tool 'life_quote' imported successfully
ADK >orchestrator agents import -r wxo-life-agent/life_insurance.yaml
[INFO] - Agent 'life_insurance' imported successfully
ADK >orchestrator chat start
```

- 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).

IBM Watsonx Orchestrate

How much life insurance would cost, I'm 50 years old, and I want 250,000 of cover.

Agents

Chats

Today

How much life insurance would c...

life_insurance 04:03 PM Show Reasoning A

How much life insurance would cost, I'm 50 years old, and I want 250,000 of cover.

You 04:03 PM

The annual premium for a 50-year-old with \$250,000 of cover would be \$1,083.33.

Type something... ▶

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

IBM Watsonx Orchestrate

How much life insurance would cost, I'm 50 years old, and I want 250,000 of cover.

Agents

Chats

Today

How much life insurance would c...

life_insurance 04:03 PM Hide Reasoning A

How much life insurance would cost, I'm 50 years old, and I want 250,000 of cover.

You 04:03 PM

Step 1

Tool: life_quote

Input - 4 Lines

```
{ "age": "50", "amount": "250000" }
```

Output - 1 Line

```
1083.333333333333
```

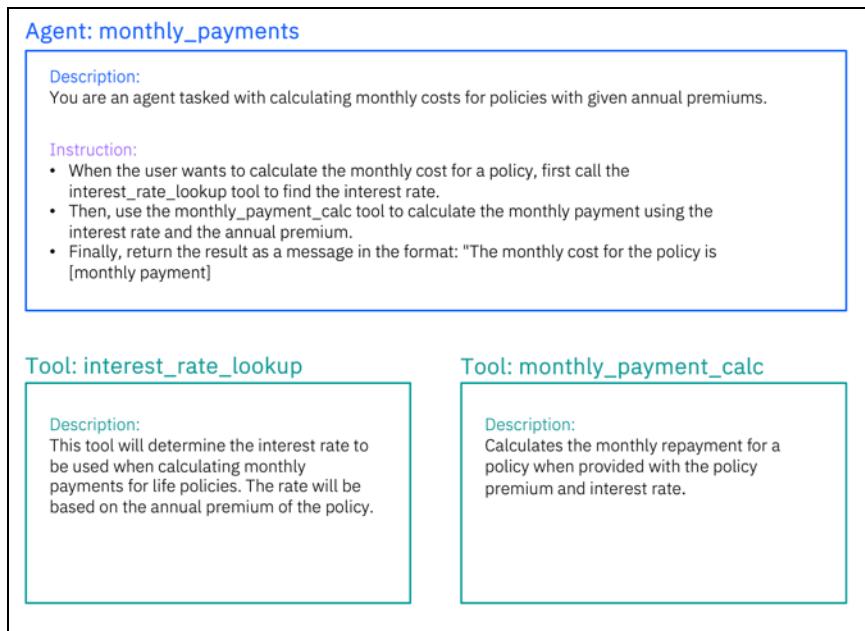
Type something... ▶

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.

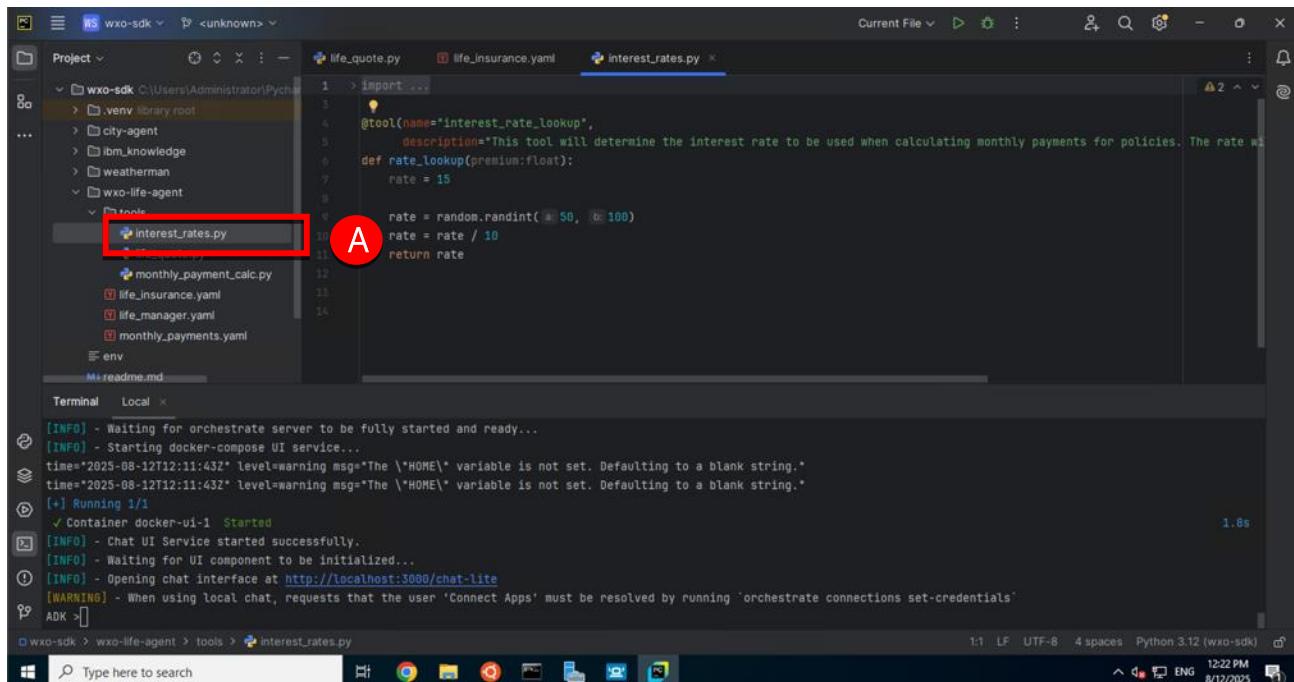
The diagram below provides an overview of the new agent configuration.



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 screenshot shows the PyCharm interface with the project tree on the left and the code editor on the right. The file `interest_rates.py` is selected in the project tree, highlighted with a red box and labeled 'A'. The code editor displays the following Python script:

```
1 > import ...
3     @tool(name="interest_rate_lookup",
4         description="This tool will determine the interest rate to be used when calculating monthly payments for policies. The rate will be between 5% and 10% based on a random number generator. The policy premium is passed as a parameter but is not used in the calculation.")
5             def rate_lookup(premium:float):
6                 rate = 15
7
8                 rate = random.randint(5, 10)
9                 rate = rate / 10
10
11             return rate
```

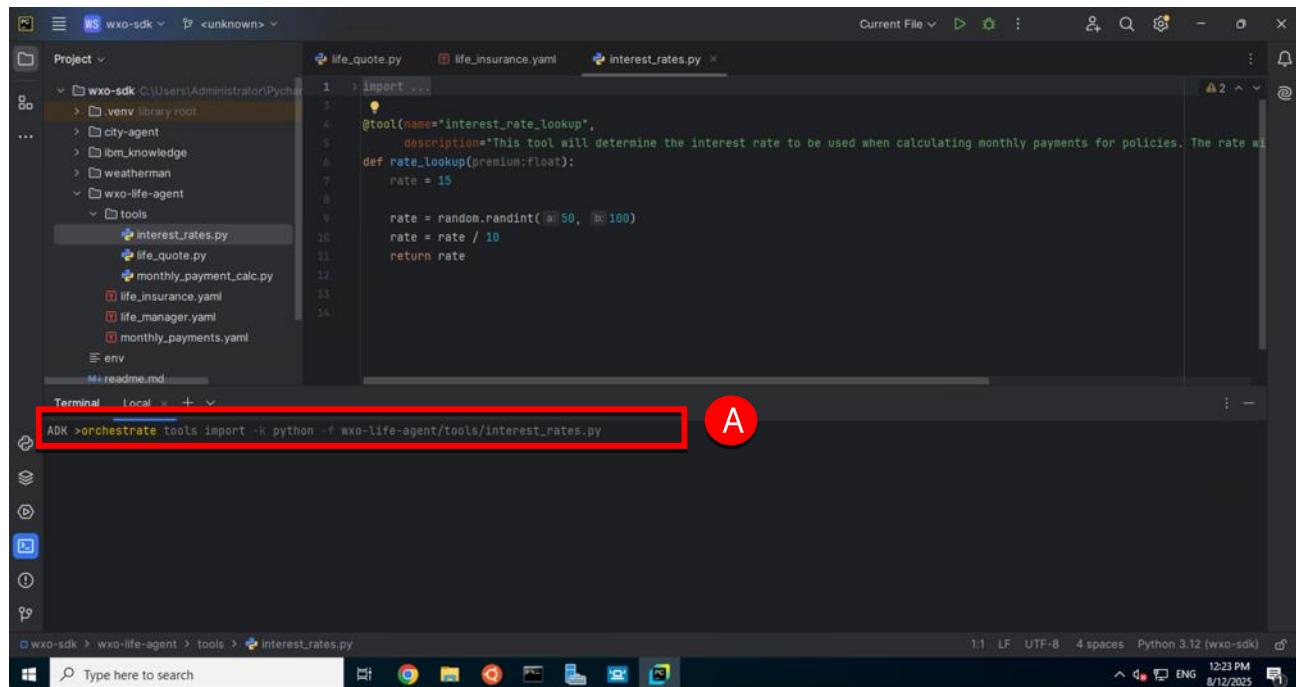
The terminal window at the bottom shows the output of the Docker compose service starting, indicating the application is ready.

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.

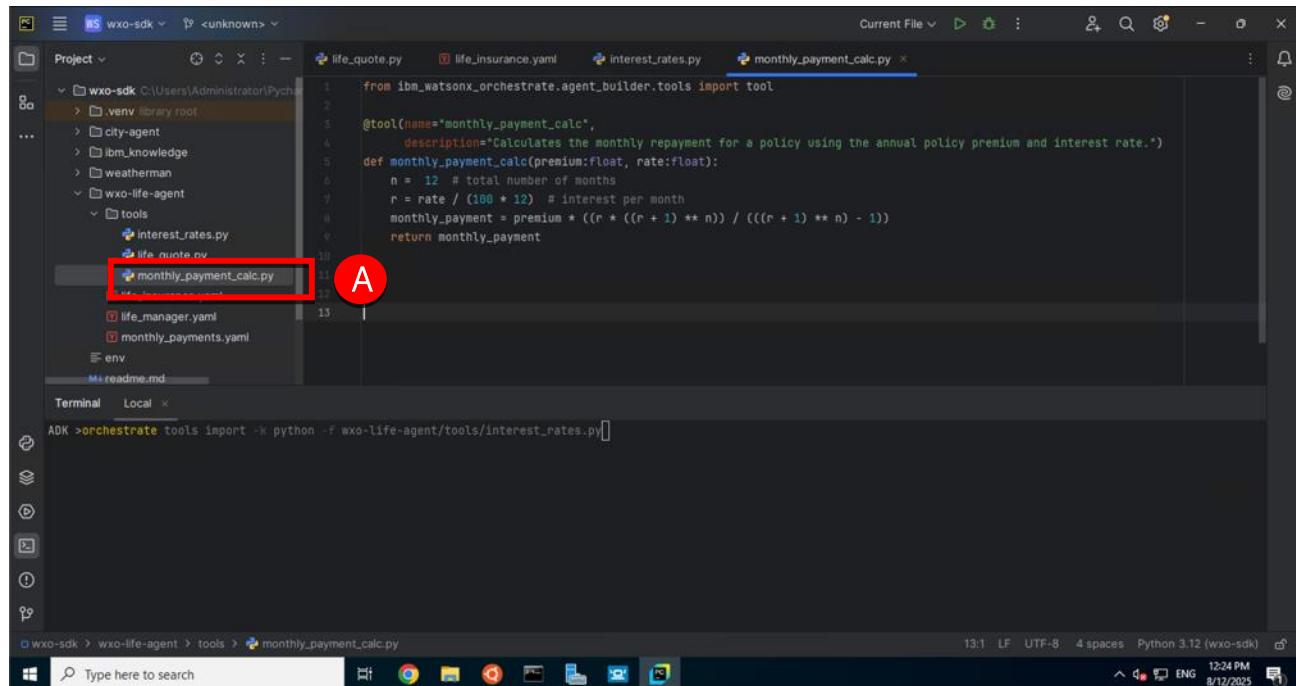
```
orchestrate tools import -k python -f wxo-life-agent/tools/interest_rates.py
```



The screenshot shows the PyCharm IDE interface. The terminal window at the bottom has the command `ADK >orchestrate tools import -k python -f wxo-life-agent/tools/interest_rates.py` highlighted with a red box and labeled 'A'. The code editor shows the `interest_rates.py` file with its contents:

```
1  > import ...
2
3  @tool(name="interest_rate_lookup",
4      description="This tool will determine the interest rate to be used when calculating monthly payments for policies. The rate wi
5  def rate_lookup(premium:float):
6      rate = 15
7
8
9  rate = random.randint( # 50, # 100)
10 rate = rate / 10
11
12
13
14
```

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



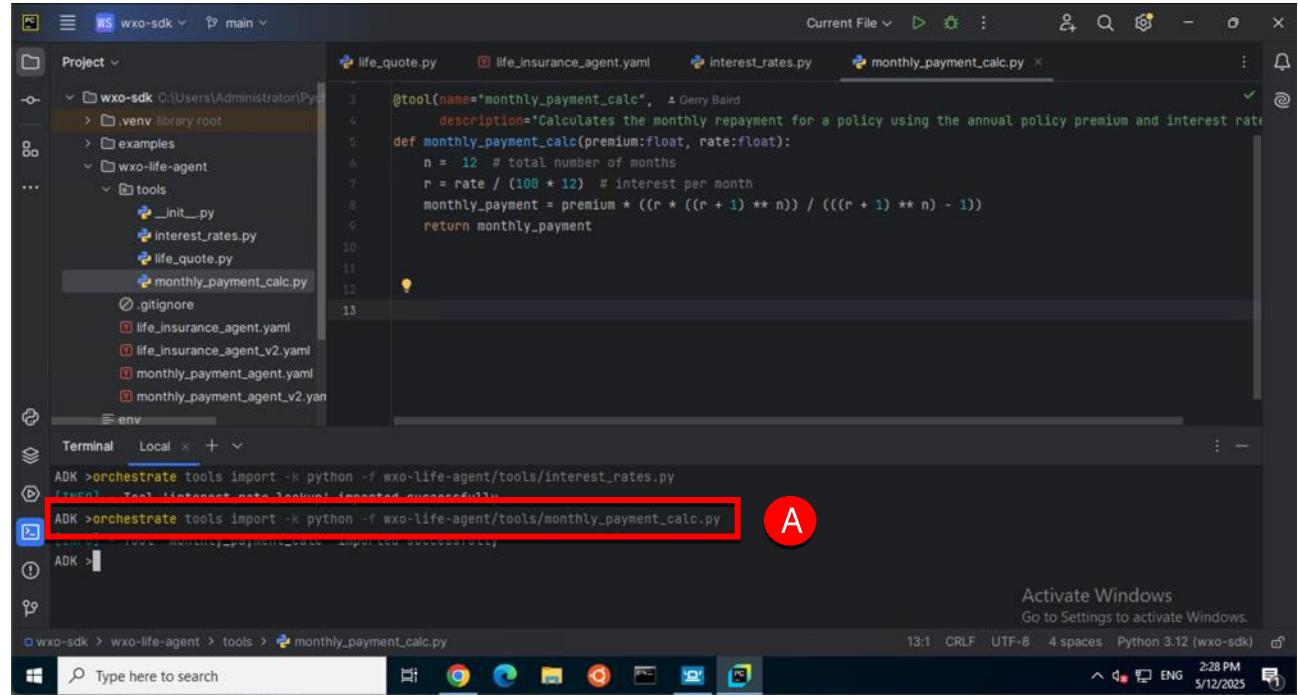
The screenshot shows the PyCharm IDE interface. The terminal window at the bottom has the command `ADK >orchestrate tools import -k python -f wxo-life-agent/tools/interest_rates.py` highlighted with a red box and labeled 'A'. The code editor shows the `monthly_payment_calc.py` file with its contents:

```
1  from ibm_watsonx_orchestrate.agent_builder.tools import tool
2
3  @tool(name="monthly_payment_calc",
4      description="Calculates the monthly repayment for a policy using the annual policy premium and interest rate.")
5  def monthly_payment_calc(premium:float, rate:float):
6      n = 12 # total number of months
7      r = rate / (100 * 12) # interest per month
8      monthly_payment = premium * ((r * ((r + 1) ** n)) / (((r + 1) ** n) - 1))
9      return monthly_payment
```

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.

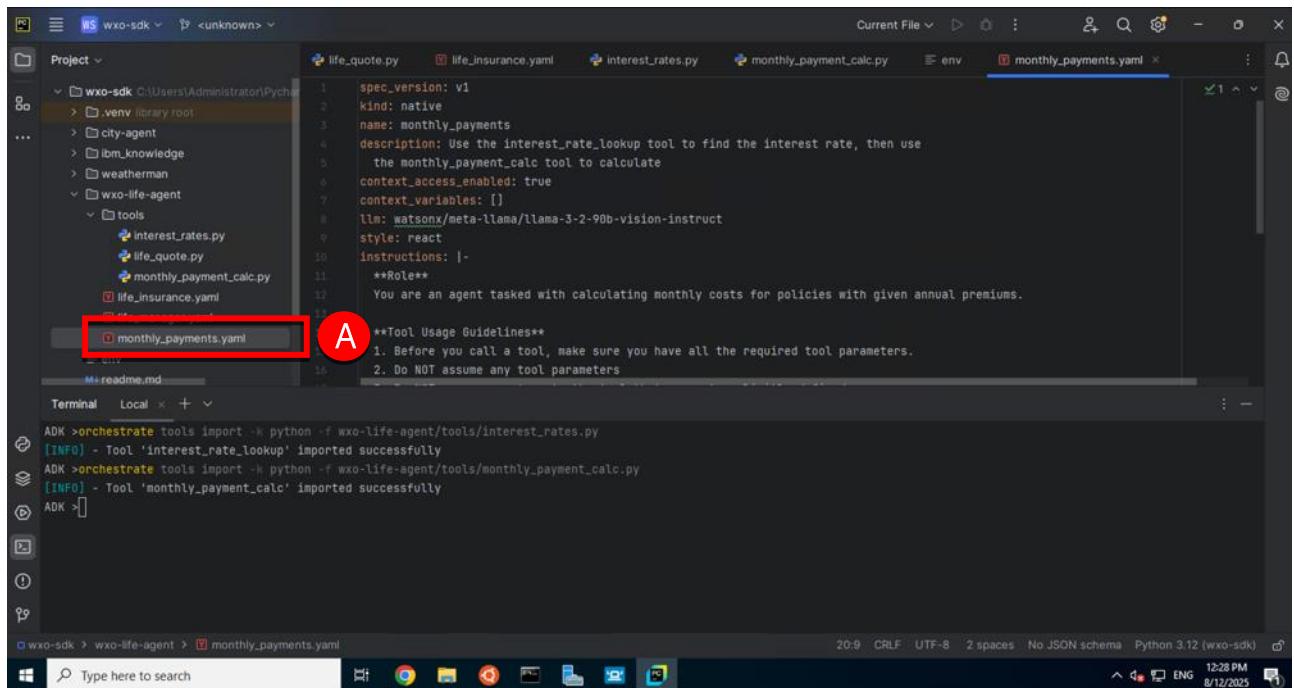
```
orchestrate tools import -k python -f wxo-life-agent/tools/monthly_payment_calc.py
```



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.



```

Project: wxo-sdk <unknown>
  life_quote.py life_insurance.yaml interest_rates.py monthly_payment_calc.py env monthly_payments.yaml
  .venv library root
  city-agent
  ibm_knowledge
  weatherman
  wxo-life-agent
    tools
      interest_rates.py
      life_quote.py
      monthly_payment_calc.py
      life_insurance.yaml
      monthly_payments.yaml
  README.md

monthly_payments.yaml
  spec_version: v1
  kind: native
  name: monthly_payments
  description: Use the interest_rate_lookup tool to find the interest rate, then use the monthly_payment_calc tool to calculate
  context_access_enabled: true
  context_variables: []
  llm: watsonx/meta-llama/llama-3-2-90b-vision-instruct
  style: react
  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.

Terminal Local + 
ADK >orchestrate tools import -k python -f wxo-life-agent/tools/interest_rates.py
[INFO] - Tool 'interest_rate_lookup' imported successfully
ADK >orchestrate tools import -k python -f wxo-life-agent/tools/monthly_payment_calc.py
[INFO] - Tool 'monthly_payment_calc' imported successfully
ADK >[

Type here to search
  20:9 CRLF UTF-8 2 spaces No JSON schema Python 3.12 (wxo-sdk)
  12:28 PM 8/12/2025

```

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.

```

10  instructions: |-
11    **Role**
12    You are an agent tasked with calculating monthly costs for policies with given annual premiums.
13
14    **Tool Usage Guidelines**
15    1. Before you call a tool, make sure you have all the required tool parameters.
16    2. Do NOT assume any tool parameters
17    3. Do NOT pass parameters to the tool that are not explicitly defined
18    4. Do NOT call the same tool multiple times with the same parameters
19
20    **How To Use Tools**
21    - When the user wants to calculate the monthly cost for a policy, first call the interest_rate_lookup tool to find the interest rate.
22    - Then, use the monthly_payment_calc tool to calculate the monthly payment using the interest rate and the annual premium.
23    - Finally, return the result as a message in the format: "The monthly cost for the policy is [monthly payment]"

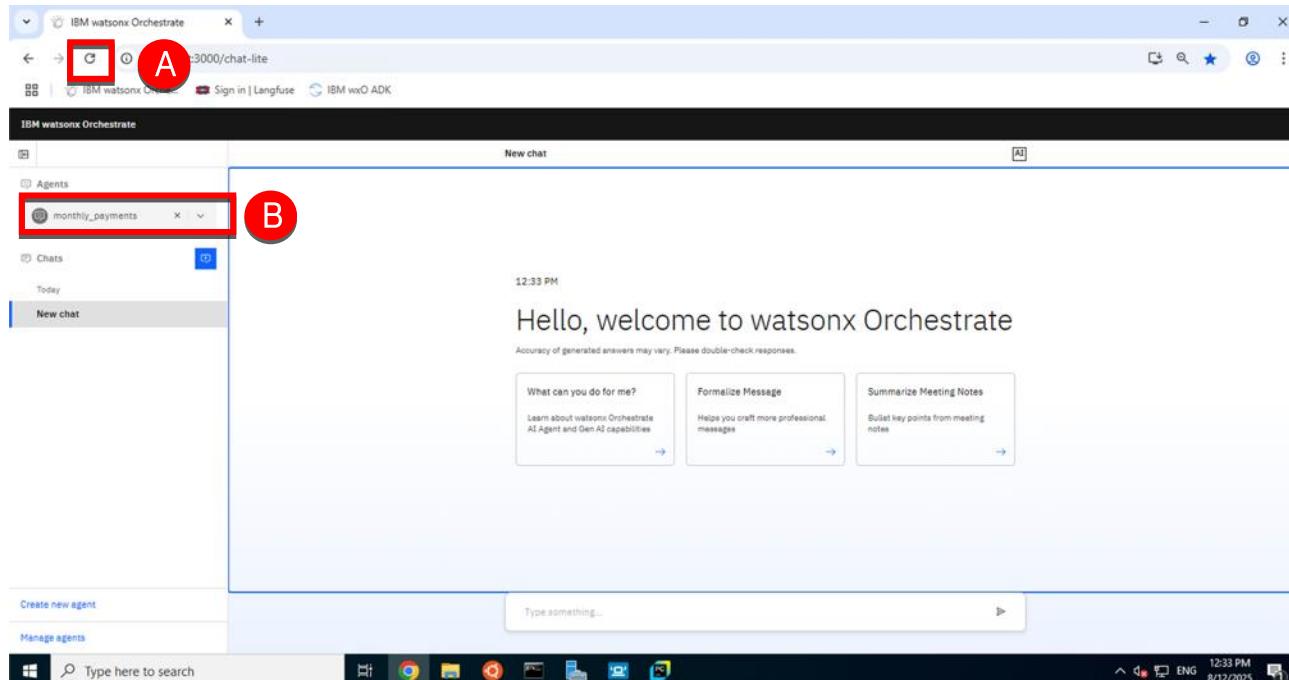
```

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

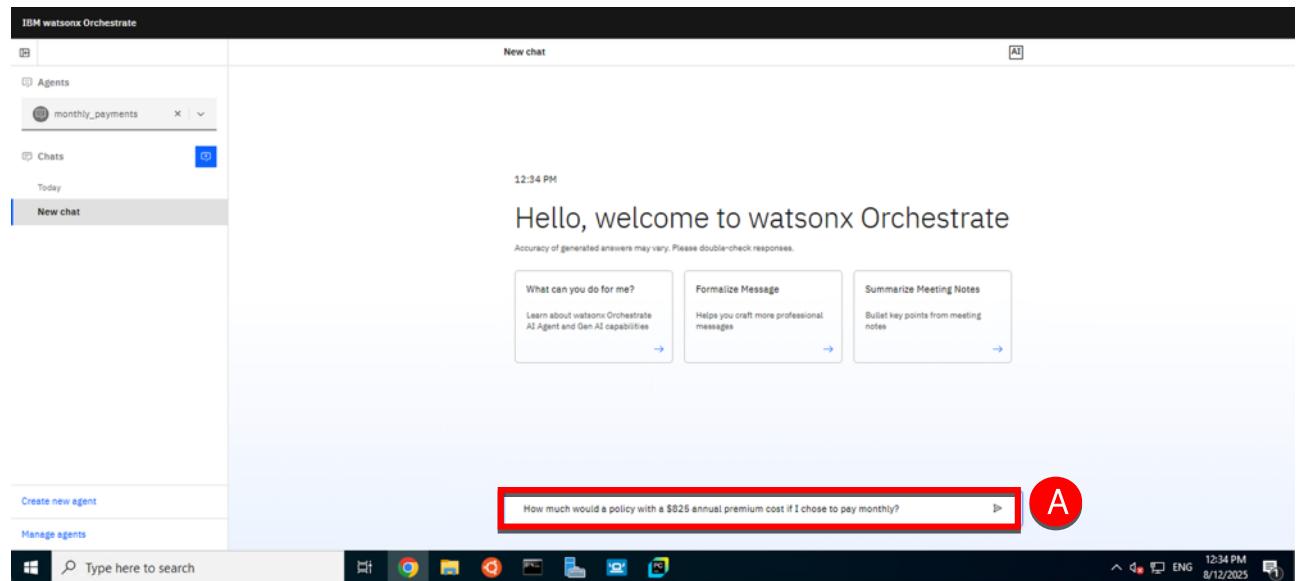
```
orchestrate agents import -f wxo-life-agent/monthly_payments.yaml
```

```
ADK >orchestrate tools import -k python -f wxo-life-agent/tools/interest_rates.py
[INFO] - Tool 'interest_rate_lookup' imported successfully
ADK >orchestrate tools import -k python -f wxo-life-agent/tools/monthly_payment_calc.py
[INFO] - Tool 'monthly_payment_calc' imported successfully
ADK >orchestrate agents import -f wxo-life-agent/monthly_payments.yaml
```

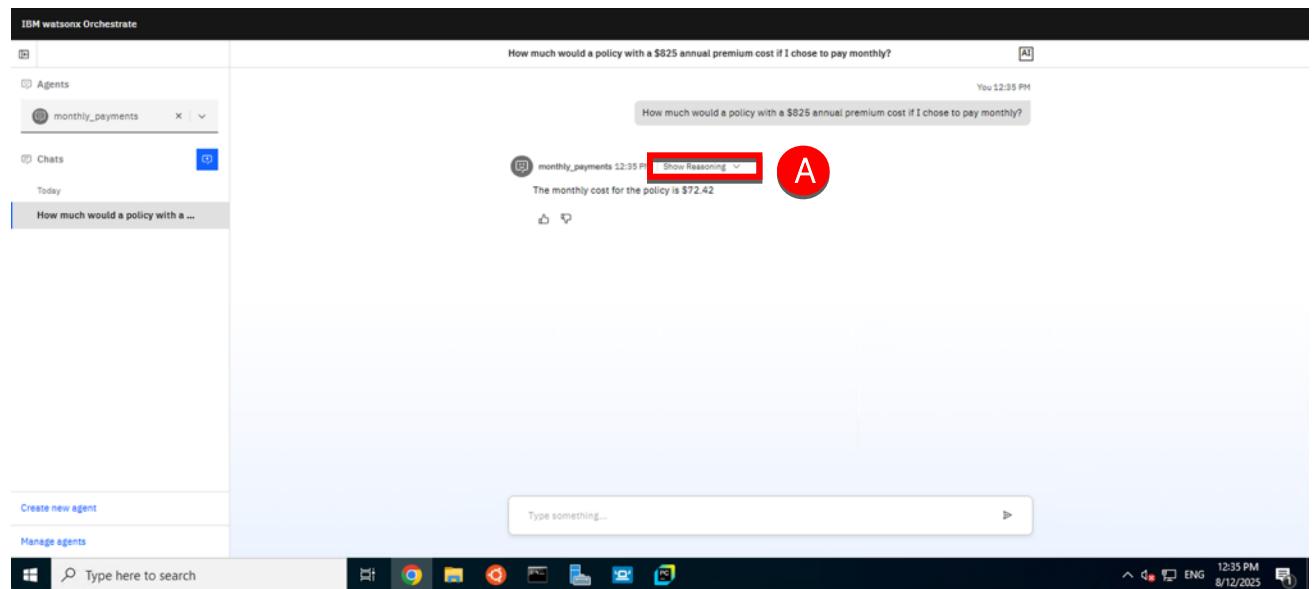
7. In the browser window click the refresh icon (A) then select **monthly_payments** from the Agents list (B).



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 screenshot shows the IBM Watsonx Orchestrate interface. On the left, there's a sidebar with 'Agents' and 'Chats'. A chat window for 'monthly_payments' is open, showing a message from the agent: 'How much would a policy with a \$825 annual premium cost if I chose to pay monthly?'. Below this, the response 'How much would a policy with a \$825 annual premium cost if I chose to pay monthly?' is shown. A red box highlights the 'Step 1' section, which contains the tool 'interest_rate_lookup'. The input field shows the JSON: { "premium": "825" }. The output field shows 'Observation: 9.7'. A red circle with the letter 'A' is in the top right corner.

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

11. Expand Step 2 (A).

The screenshot shows the IBM Watsonx Orchestrate interface. The sidebar and chat window are similar to the previous screenshot. The response 'How much would a policy with a \$825 annual premium cost if I chose to pay monthly?' is still present. A red box highlights the 'Step 2' section, which contains the tool 'monthly_payment_calc'. The input field shows the JSON: { "rate": "9.7", "premium": "825" }. The output field shows 'Observation: 72.41564742062328'. A red circle with the letter 'A' is in the top right corner.

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

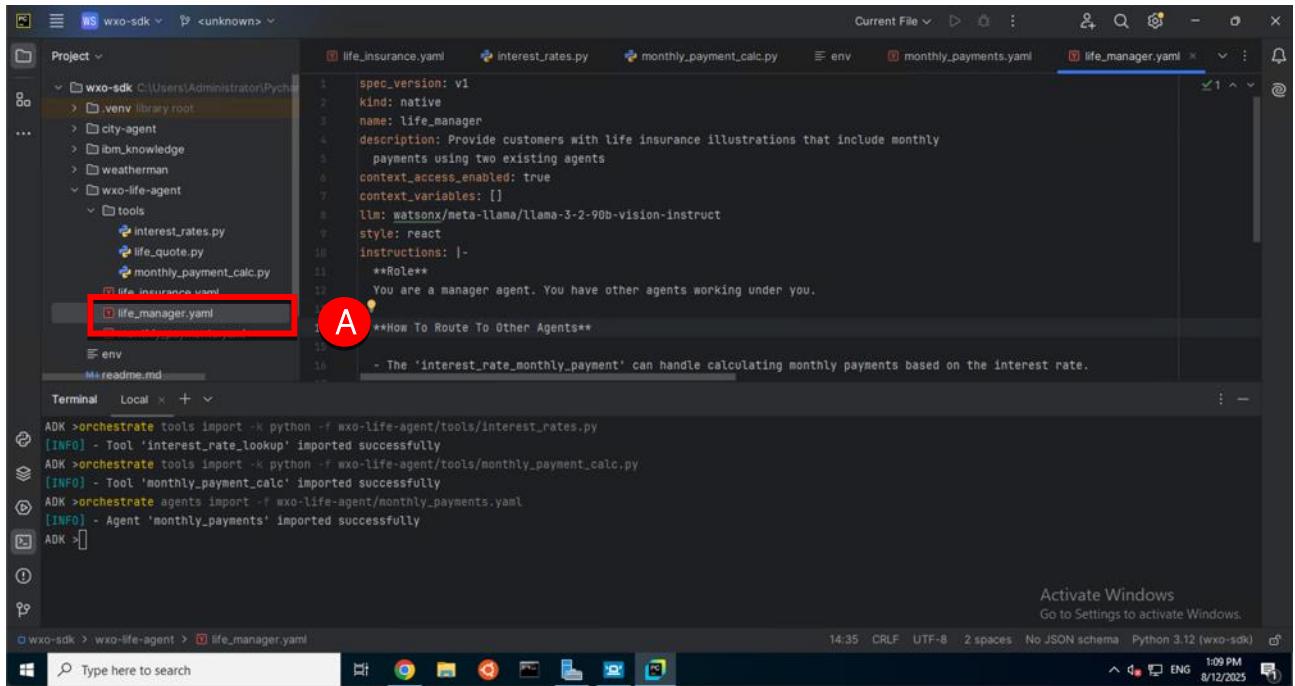
Next you will add a further agent that is able to collaborate with the two existing agents to answer more complex enquires that involve a life cover quotation, combined with monthly payments. The diagram below provides an overview of the expanded configuration used in this section. Tools used by the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

<p>Agent: life_manager</p> <p>Description: Provide customers with life insurance illustrations that include monthly payments using two existing agents</p> <p>Instruction: You are a manager agent. You have other agents working under you.</p> <ul style="list-style-type: none">- The 'monthly_payments' can handle calculating monthly payments based on the interest rate.- The 'life_insurance' can handle providing life insurance illustrations, including annual premiums. <p>Your job is to delegate tasks to the most appropriate agent and provide customers with life insurance illustrations that include the annual premium. If the customer requests monthly payments, use the 'monthly_payments' agent to calculate and include the monthly payment in the illustration.</p>	<p>Agent: life_insurance</p> <p>Description: Provide a life insurance illustration that includes the annual premium of the policy</p> <p>Instruction: You are a life insurance agent tasked with providing illustrations of life insurance policies, including the annual premium.</p>	<p>Agent: monthly_payments</p> <p>Description: You are an agent tasked with calculating monthly costs for policies with given annual premiums.</p> <p>Instruction: <ul style="list-style-type: none">• 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]</p>
---	---	---

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).



```

Project: wxo-sdk C:\Users\Administrator\PycharmProjects\wxa-life-agent> life_manager.yaml
  spec_version: v1
  kind: native
  name: life_manager
  description: Provide customers with life insurance illustrations that include monthly payments using two existing agents
  context_access_enabled: true
  context_variables: []
  llm: watsonx/meta-llama/llama-3-2-90b-vision-instruct
  style: react
  instructions: |-
    **Role**
    You are a manager agent. You have other agents working under you.

    **How To Route To Other Agents**
    - The 'interest_rate_monthly_payment' can handle calculating monthly payments based on the interest rate.

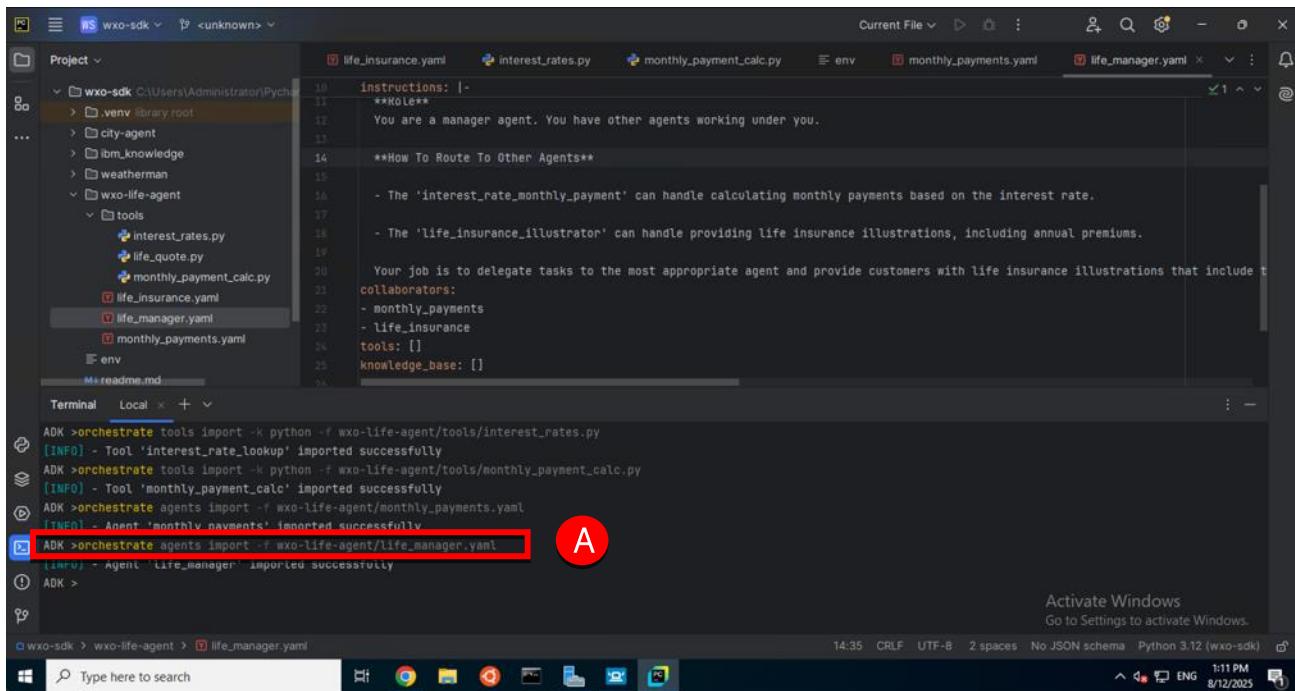
ADK >orchestrate tools import -k python -f wxo-life-agent/tools/interest_rates.py
[INFO] - Tool 'interest_rate_lookup' imported successfully
ADK >orchestrate tools import -k python -f wxo-life-agent/tools/monthly_payment_calc.py
[INFO] - Tool 'monthly_payment_calc' imported successfully
ADK >orchestrate agents import -f wxo-life-agent/monthly_payments.yaml
[INFO] - Agent 'monthly_payments' imported successfully
ADK >

```

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
```



```

Project: wxo-sdk C:\Users\Administrator\PycharmProjects\wxa-life-agent> life_manager.yaml
  spec_version: v1
  kind: native
  name: life_manager
  description: Provide customers with life insurance illustrations that include monthly payments using two existing agents
  context_access_enabled: true
  context_variables: []
  llm: watsonx/meta-llama/llama-3-2-90b-vision-instruct
  style: react
  instructions: |-
    **Role**
    You are a manager agent. You have other agents working under you.

    **How To Route To Other Agents**
    - The 'interest_rate_monthly_payment' can handle calculating monthly payments based on the interest rate.
    - The 'life_insurance_ilustrator' can handle providing life insurance illustrations, including annual premiums.

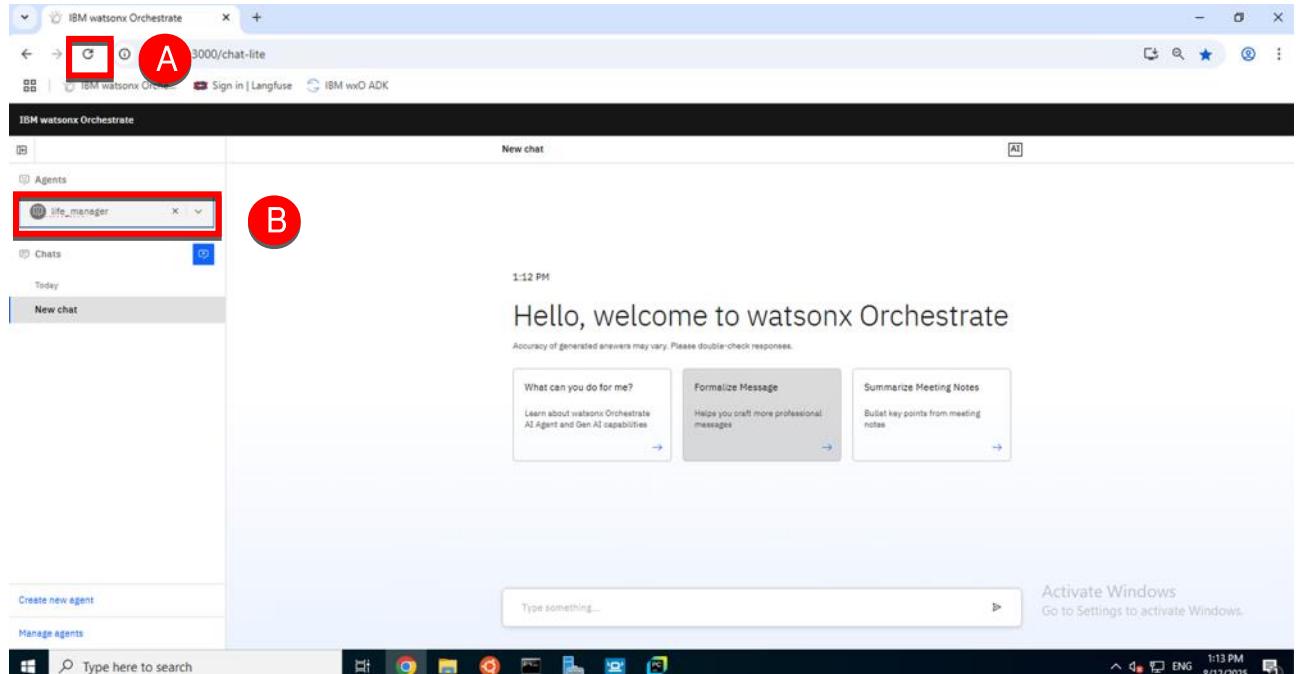
    Your job is to delegate tasks to the most appropriate agent and provide customers with life insurance illustrations that include t
    collaborators:
    - monthly_payments
    - life_insurance
    tools: []
    knowledge_base: []

ADK >orchestrate tools import -k python -f wxo-life-agent/tools/interest_rates.py
[INFO] - Tool 'interest_rate_lookup' imported successfully
ADK >orchestrate tools import -k python -f wxo-life-agent/tools/monthly_payment_calc.py
[INFO] - Tool 'monthly_payment_calc' imported successfully
ADK >orchestrate agents import -f wxo-life-agent/monthly_payments.yaml
[INFO] - Agent 'monthly_payments' imported successfully
ADK >orchestrate agents import -f wxo-life-agent/life_manager.yaml
[INFO] - Agent 'Life_Manager' imported successfully
ADK >

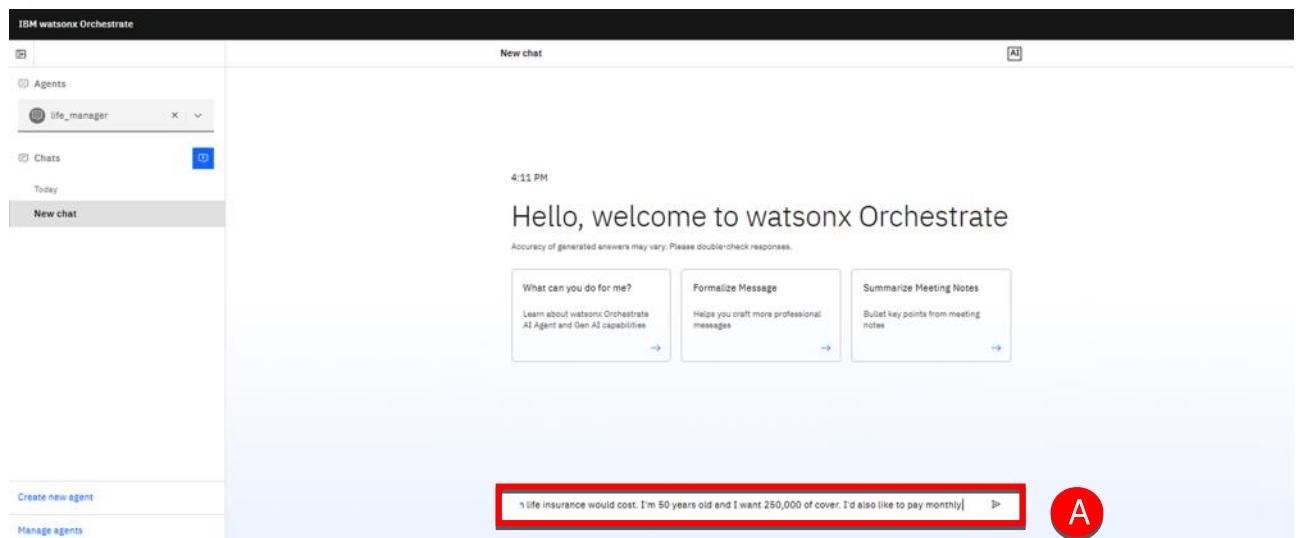
```

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 screenshot shows the IBM Watsonx Orchestrate interface. On the left, there's a sidebar with 'Agents' (selected), 'Chats' (Today), and buttons for 'Create new agent' and 'Manage agents'. The main area is a chat window. A user message at the top says: 'How much life insurance would cost. I'm 50 years old and I want 250,000 of cover. I'd also like to ...'. Below it, an 'AI' button and the timestamp 'You 04:12 PM' are visible. An agent message follows: '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.' This message has a 'Show Reasoning' dropdown arrow pointing down. A large red circle with the letter 'A' is drawn around this dropdown. The agent's response continues: 'The annual premium for life insurance would be \$1,083.33 and the monthly cost for the policy is \$93.26'. At the bottom of the screen is a search bar with 'Type something...' and a magnifying glass icon.

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.

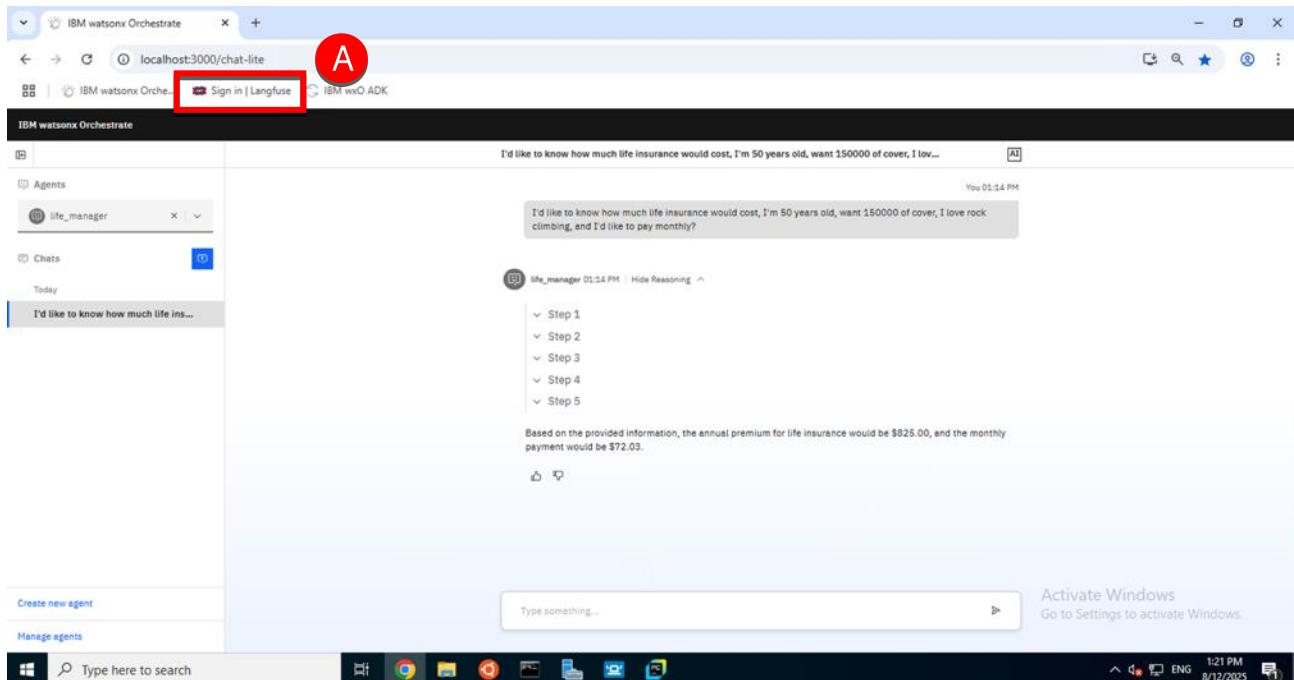
It would also have been possible to achieve the same result by adding the additional tools and instructions into a single top-level agent. However, decomposing this functionality into separate agents keeps the agents smaller, simpler, promotes reuse and should improve the reasoning accuracy of the solution.

Langfuse trace

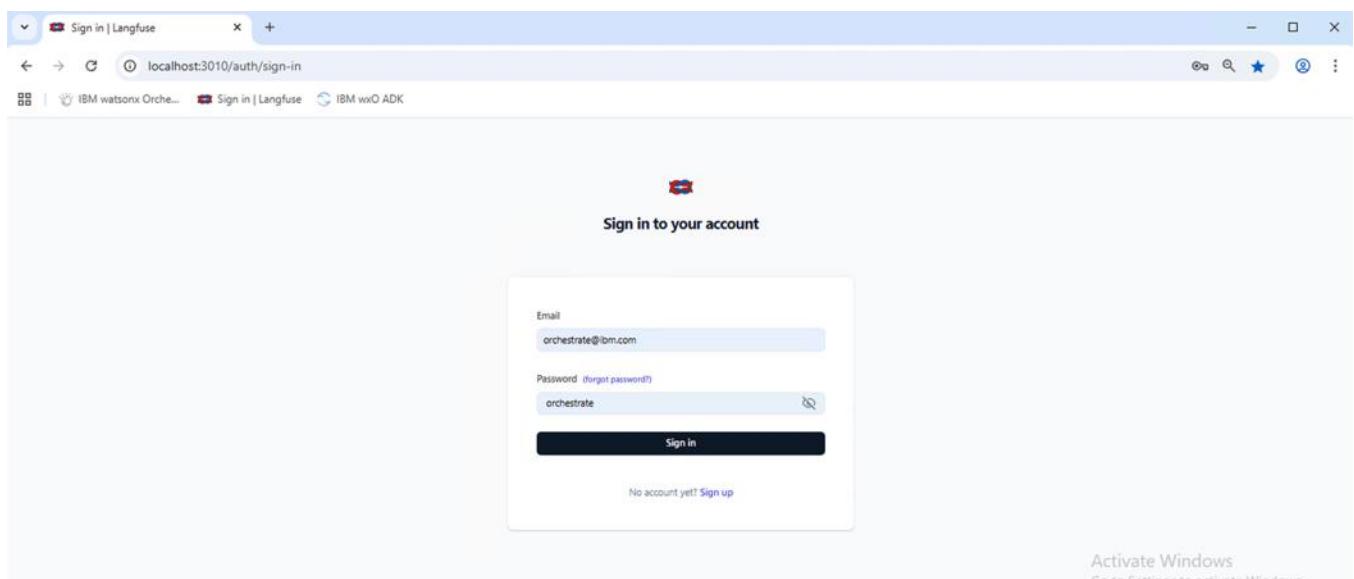
In this section you will review the trace information captured in Langfuse and understand the chain of thought used by the agent to produce the response.

Access the trace

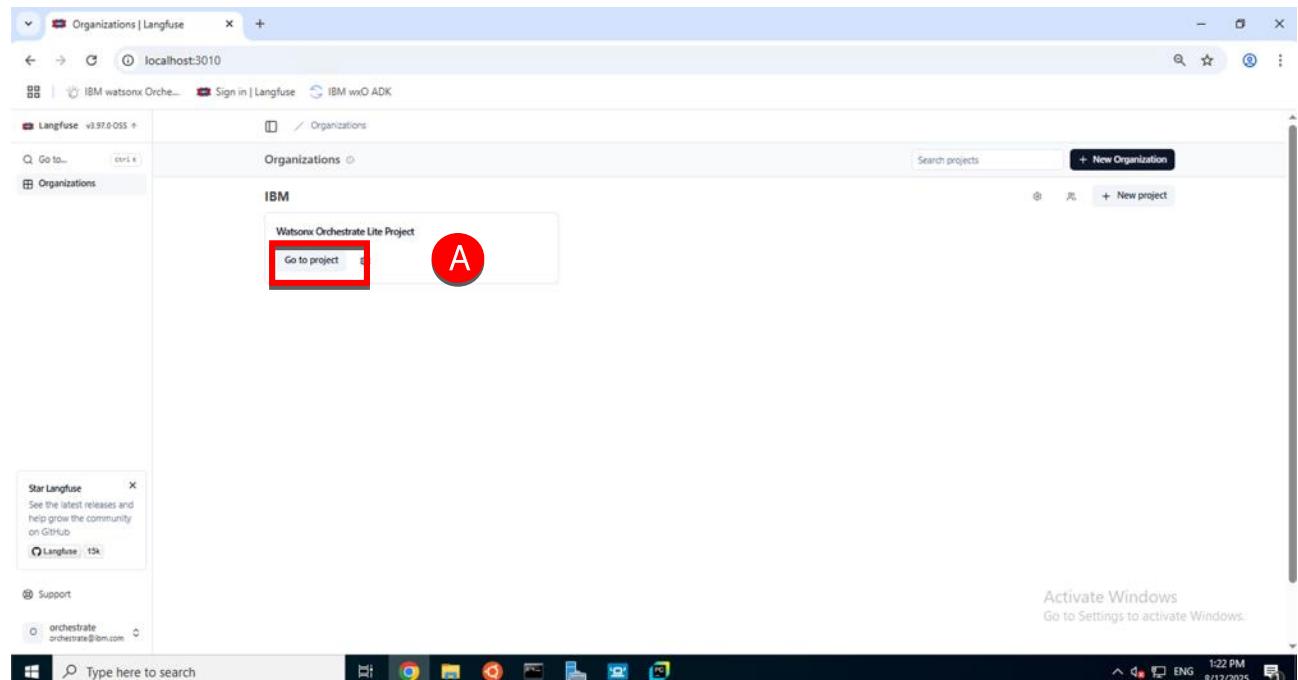
1. Click the **Sign in | Langfuse** bookmark in the browser (A).



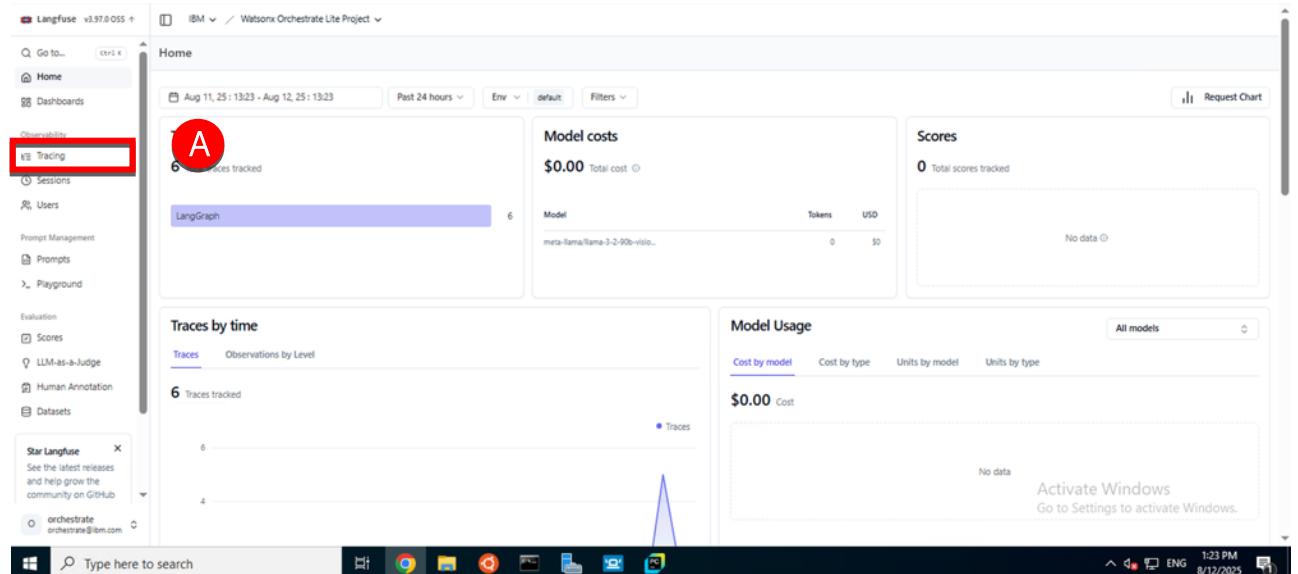
2. The login page should be pre-populated, enter credentials if needed, and click **Sign in** (A).



3. Click Go to project (A).



4. Click Tracing (A).



5. Click Traces (A).

The screenshot shows the Watsonx Orchestrate Lite Project Home page. The left sidebar has a 'Traces' icon highlighted with a red box and a circled 'A'. The main area displays 'Model costs' (\$0.00), 'Scores' (0 total scores tracked), and 'Traces by time' (7 traces tracked). Below these are sections for 'Model Usage' and 'Activate Windows'. The bottom status bar shows the date as 5/12/2025.

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

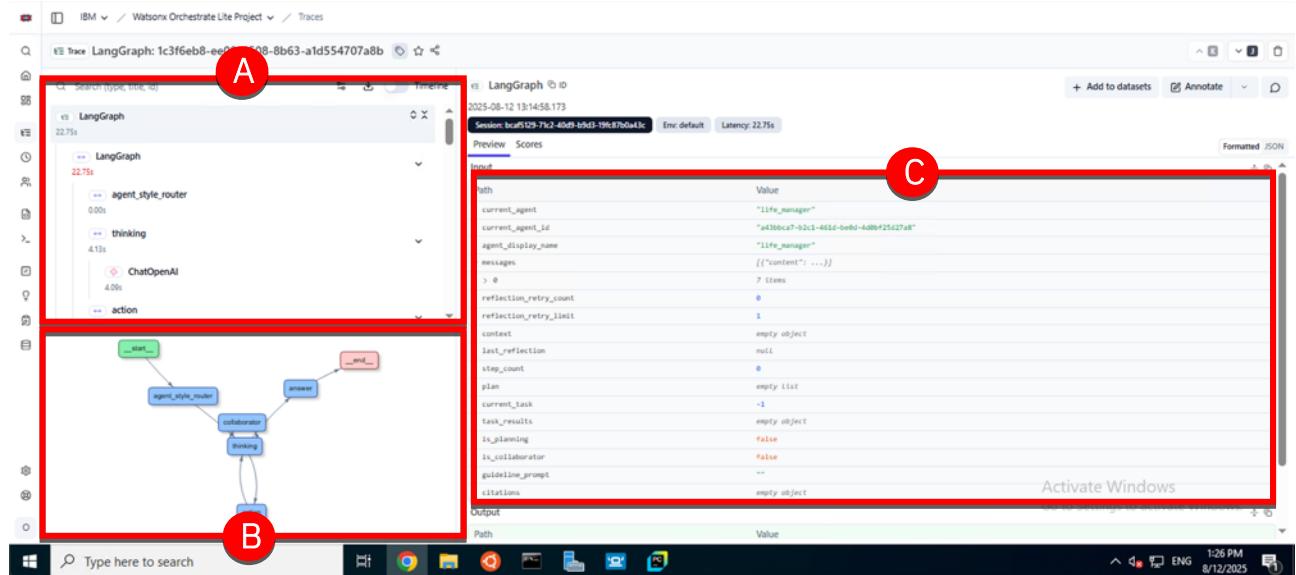
The screenshot shows the Watsonx Orchestrate Lite Project Tracing page. The left sidebar is expanded, showing 'Observability' (Tracing selected), 'Sessions', 'Users', 'Prompt Management', 'Prompts', and 'Playground'. The main area displays a table of tracing data with columns: Timestamp, Name, Input, Output, Observation Level, Latency, Tokens, and Total Cost. The bottom status bar shows the date as 8/12/2025.

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

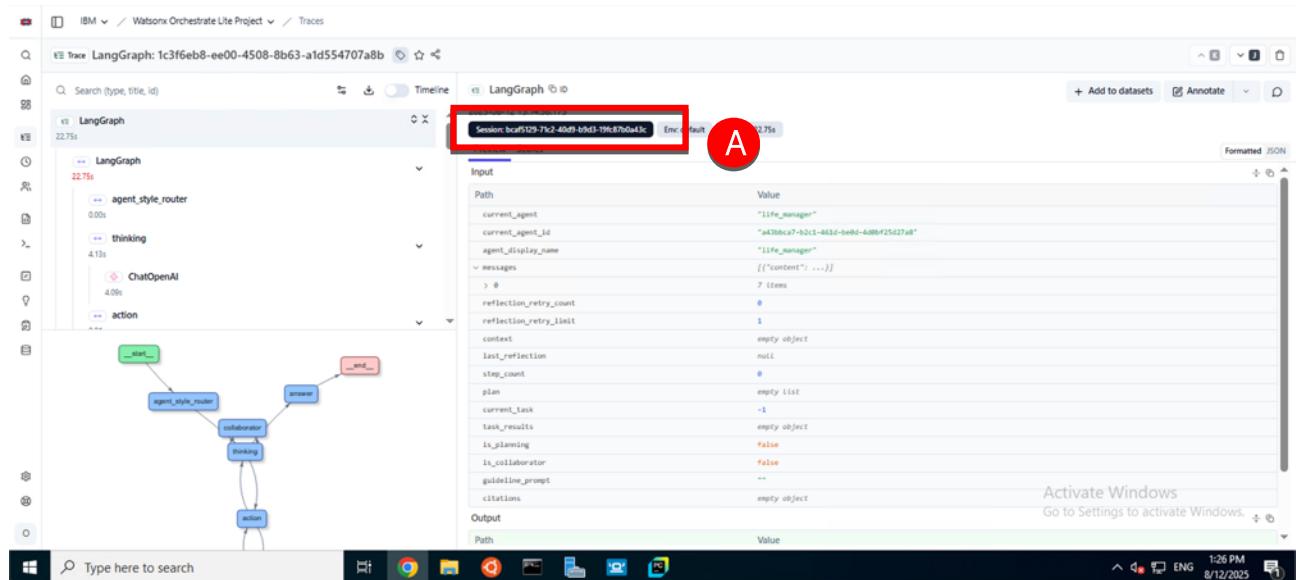
8. Click the expand icon (A), this will show the trace in full screen.

Analyze the trace

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



- 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).

The screenshot shows the Watsonx Orchestrate Lite Project Traces panel. A red circle labeled 'B' highlights the 'ChatOpenAI' entry in the search results list. A red circle labeled 'C' highlights the detailed view of the 'ChatOpenAI' entry, which includes a system prompt and a flowchart diagram.

System

You run in a loop of Thought, Action, PAUSE, Observation. At the end of the loop you output the Answer.

- Answer the user's question as best as possible using the available tools.
- If parameters required to execute a tool is missing, try to infer it from conversation history else ask the user for it one by one.
- If the user greets you (e.g., "hi", "hello"), respond with: "Hello! I am Watsonx Orchestrate, an AI assistant, created by IBM. How can I help you today?" and nothing else.

You have access to the following tools to use as actions: [{}], {type: 'function', 'function': 'monthly_payments', 'description': 'Use the interest_rate_lookup tool to find the Interest rate, then use the monthly_payment_calc tool to calculate', 'parameters: [{}]}, {type: 'function', 'function': 'life_insurance', 'description': 'Provide a life insurance illustration that includes the annual premium of the policy', 'parameters: [{}]}]

Use the following format: Thought: Your thoughts about what to do next Action: the action to take in the form of a JSON tool call with json formatted input parameters. The object must contain two fields: name (string, name of the tool to call) and arguments (object, the arguments to pass to the tool) For example, Action: {"name": "run_code", "args": {"code": "print('Hello World')", "language": "python"} } When generating tools calls for actions PAUSE

If there is no action to take, output the Answer that this question is beyond my capability. If an action is required, you must PAUSE and wait for the next call.

After the PAUSE, the user will respond with the result of the action. DO NOT TRY TO PLAN AHEAD. YOU MUST WAIT FOR THE USER TO RESPOND. The user response will look like this: Observation: the result from taking the action ... This Thought, Action, Input, PAUSE, Observation series can repeat N-times.

If parameters are missing you will respond like this: Collect: Your question to collect missing parameters

Once you know the final answer, you respond like this: Thought: I know the final answer: Answer: your final answer to the original input question

User Instructions: Role you are a manager agent. You have other agents working under you.

How To Route To Other Agents

- The 'interest_rate_monthly_payment' can handle calculating monthly payments based on the interest rate.
- The 'life_insurance_illustrator' can handle providing life insurance illustrations, including annual premiums.

It is up to you to delegate tasks to the most appropriate agent and provide customers with life insurance illustrations that include the annual premium. If the customer requests monthly payments, use the 'interest_rate_monthly_payment' agent to calculate and include the monthly payment in the illustration.

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 Watsonx Orchestrate Lite Project Traces panel. A red circle labeled 'A' highlights the scroll bar. A red circle labeled 'B' highlights the 'User' panel, which contains a message from the user: "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". A red circle labeled 'C' highlights the 'Assistant' panel, which contains a thought from the agent: "Thought: To provide a life insurance illustration with the annual premium and monthly payments, I need to delegate tasks to the 'life_insurance_illustrator' and 'interest_rate_monthly_payment' agents. First, I'll ask the 'life_insurance_illustrator' to provide the annual premium. Actions: {"name": "life_insurance", "arguments": {"age": "50", "cover_amount": "150000", "hazardous_activities": "rock climbing"} } PAUSE".

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).

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

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

The screenshot shows the Watsonx Orchestrate Lite Project Traces panel. A red circle labeled 'A' points to the bottom of the main trace list area, where a ChatOpenAI step from 1.08s is highlighted with a red box. A red arrow points from this box down to the detailed view of the ChatOpenAI step. The detailed view shows the step's configuration, including its name, latency, and the JSON input it received. It also includes a diagram of the workflow route and the step's internal logic.

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

The screenshot shows the Watsonx Orchestrate Lite Project Traces panel. A red circle labeled 'A' points to the scroll bar on the right side of the trace detail panel, indicating that the panel has been scrolled down. The 'Tool' panel (B) and 'Assistant' panel (C) are now visible at the bottom of the trace detail panel. The 'Tool' panel contains an observation about monthly costs, and the 'Assistant' panel contains a thought about the final answer. A red arrow points from the scroll bar down to the 'Tool' and 'Assistant' panels.

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.

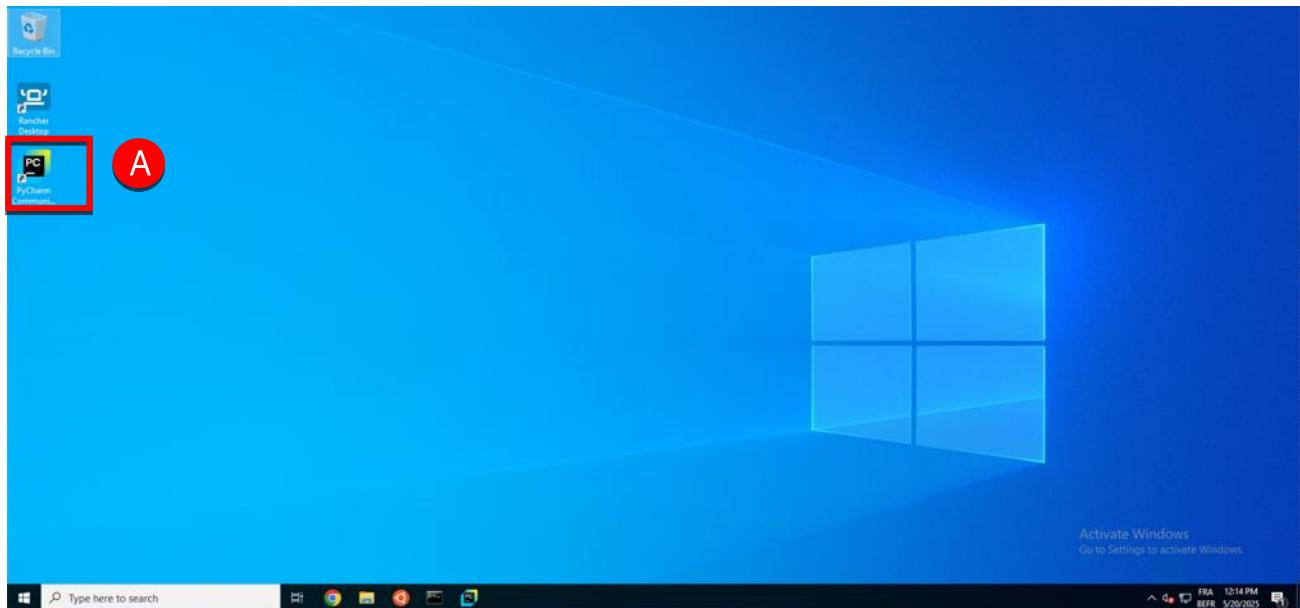
LLM Error

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.

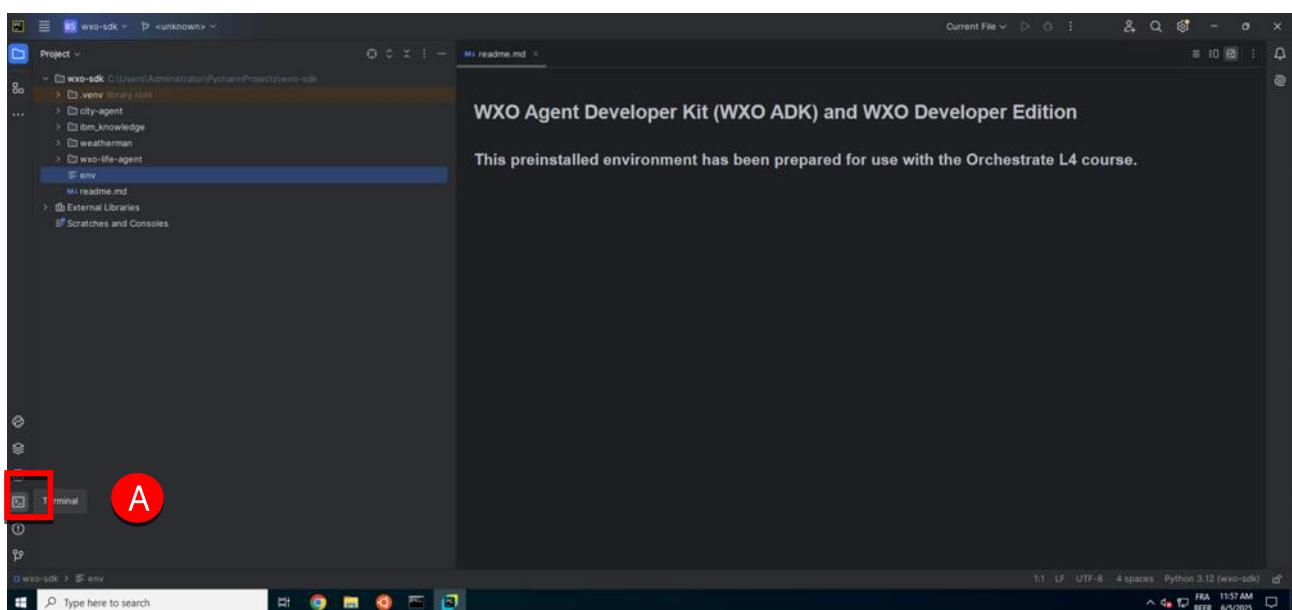
Appendix: Reset your environment

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.)

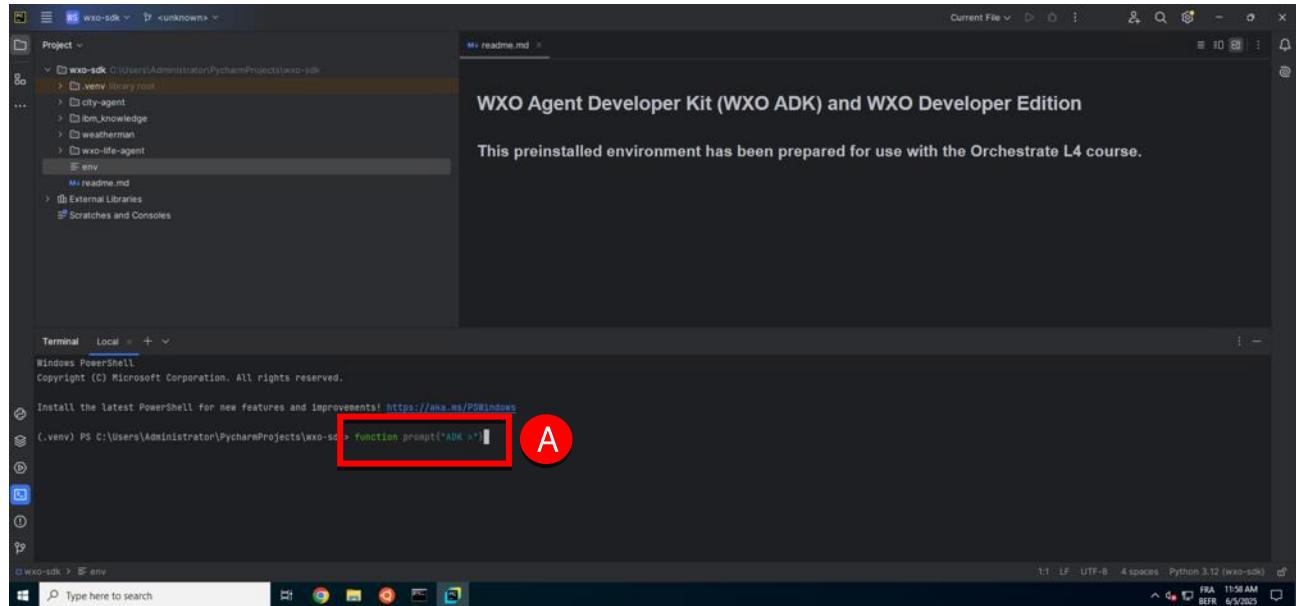


3. Click the terminal icon (A), to open the terminal (if required).



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
```

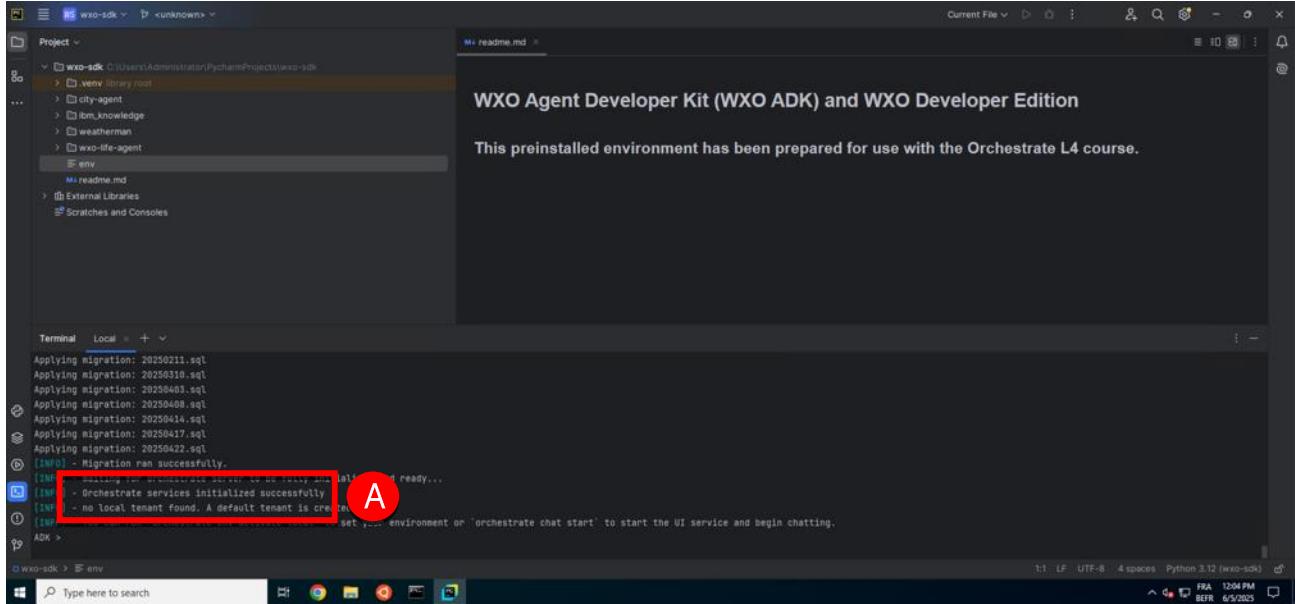
6. Enter the following command into the terminal to reset your server, then press Enter.

```
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.



The terminal output shows the following logs:

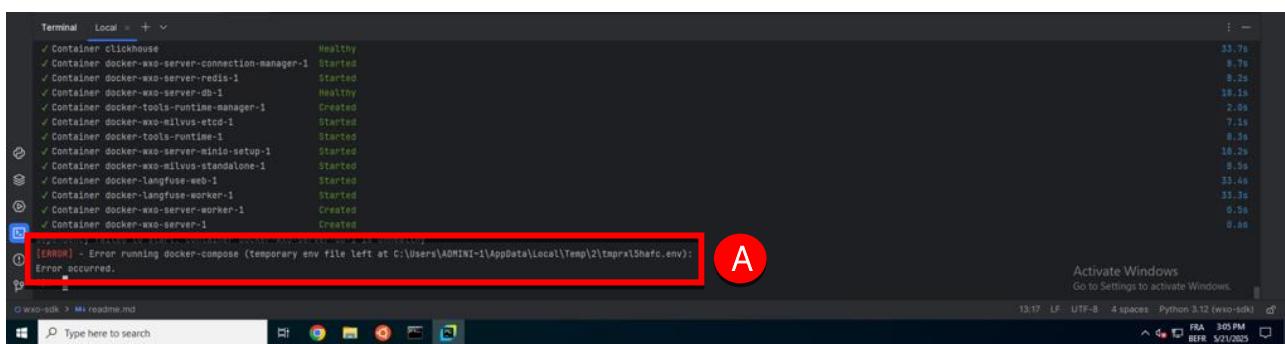
```

Terminal Local + v
Applying migration: 20230211.sql
Applying migration: 20230310.sql
Applying migration: 20230403.sql
Applying migration: 20230408.sql
Applying migration: 20230414.sql
Applying migration: 20230417.sql
Applying migration: 20230422.sql
[INFO] - Migration ran successfully.
[INFO] - Waiting for Orchestrate services to start up... all ready...
[INFO] - Orchestrate services initialized successfully
[INFO] - no local tenant found. A default tenant is created
[INFO] - Set 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:



The terminal output shows the following error:

```

Container clickhouse
✓ Container docker-wxo-server-connection-manager-1 Healthy
✓ Container docker-wxo-server-redis-1 Started
✓ Container docker-wxo-server-db-1 Healthy
✓ Container docker-tools-runtime-manager-1 Created
✓ Container docker-wxo-milvus-etcd-1 Started
✓ Container docker-tools-runtime-1 Started
✓ Container docker-wxo-server-milvus-setup-1 Started
✓ Container docker-wxo-milvus-standalone-1 Started
✓ Container docker-tangfuse-web-1 Started
✓ Container docker-tangfuse-worker-1 Started
✓ Container docker-wxo-server-worker-1 Created
✓ Container docker-wxo-server-1 Created

[ERROR] - Error running docker-compose (temporary env file left at C:\Users\ADMINI-1\AppData\Local\Temp\1\tmprx15hafc.env):
Error occurred.

ADK >

```