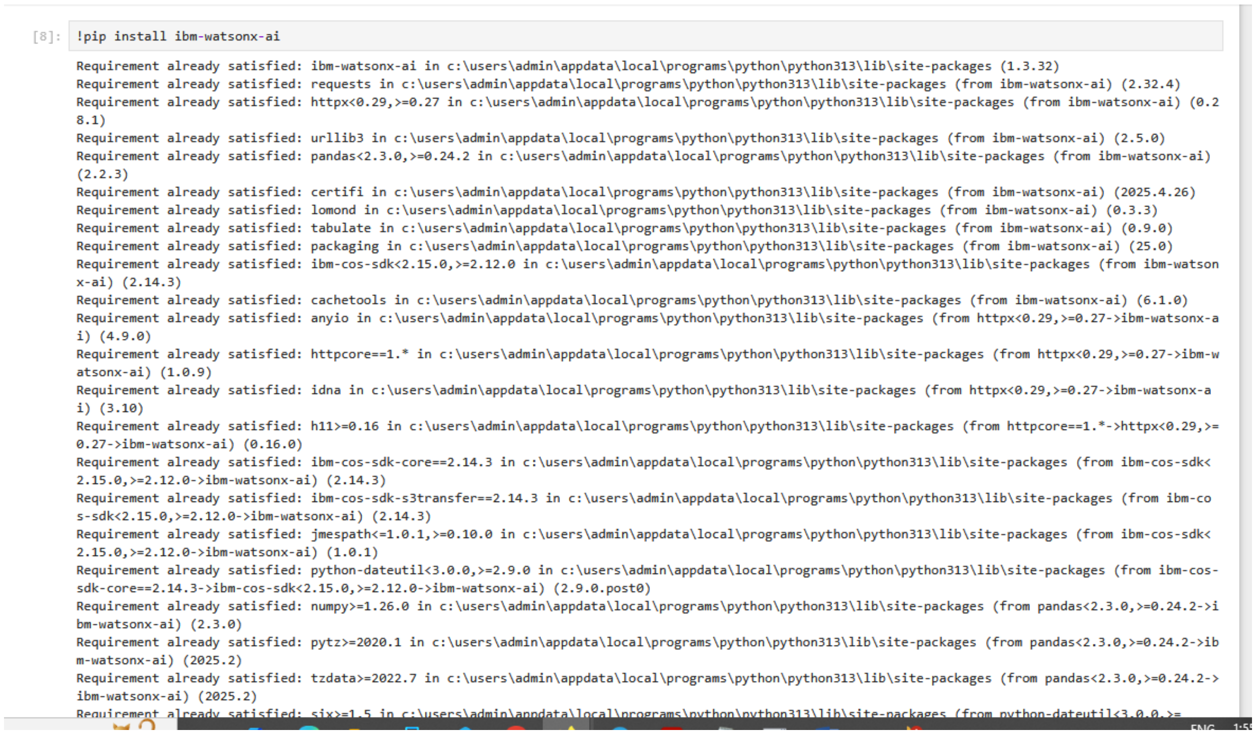


```
[8]: !pip install ibm-watsonx-ai
```

```
Requirement already satisfied: ibm-watsonx-ai in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (1.3.32)
Requirement already satisfied: requests in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai) (2.32.4)
Requirement already satisfied: httpx<0.29,>=0.27 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai) (0.2
8.1)
Requirement already satisfied: urllib3 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai) (2.5.0)
Requirement already satisfied: pandas<2.3.0,>=0.24.2 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai)
(2.2.3)
Requirement already satisfied: certifi in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai) (2025.4.26)
Requirement already satisfied: lomond in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai) (0.3.3)
Requirement already satisfied: tabulate in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai) (0.9.0)
Requirement already satisfied: packaging in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai) (25.0)
Requirement already satisfied: ibm-cos-sdk<2.15.0,>=2.12.0 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watson
x-ai) (2.14.3)
Requirement already satisfied: cachetools in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai) (6.1.0)
Requirement already satisfied: anyio in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from httpx<0.29,>=0.27->ibm-watsonx-a
i) (4.9.0)
Requirement already satisfied: httpcore==1.* in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from httpx<0.29,>=0.27->ibm-w
atsonx-ai) (1.0.9)
Requirement already satisfied: idna in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from httpx<0.29,>=0.27->ibm-watsonx-a
i) (3.10)
Requirement already satisfied: h11>=0.16 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from httpcore==1.*->httpx<0.29,>=
0.27->ibm-watsonx-ai) (0.16.0)
Requirement already satisfied: ibm-cos-sdk-core==2.14.3 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-cos-sdk<
2.15.0,>=2.12.0->ibm-watsonx-ai) (2.14.3)
Requirement already satisfied: ibm-cos-sdk-s3transfer==2.14.3 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-co
s-sdk<2.15.0,>=2.12.0->ibm-watsonx-ai) (2.14.3)
Requirement already satisfied: jmespath<=1.0.1,>=0.10.0 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-cos-sdk<
2.15.0,>=2.12.0->ibm-watsonx-ai) (1.0.1)
Requirement already satisfied: python-dateutil<3.0.0,>=2.9.0 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-cos-
sdk-core==2.14.3->ibm-cos-sdk<2.15.0,>=2.12.0->ibm-watsonx-ai) (2.9.0.post0)
Requirement already satisfied: numpy>=1.26.0 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from pandas<2.3.0,>=0.24.2->i
bm-watsonx-ai) (2.3.0)
Requirement already satisfied: pytz>=2020.1 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from pandas<2.3.0,>=0.24.2->ib
m-watsonx-ai) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from pandas<2.3.0,>=0.24.2->
ibm-watsonx-ai) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from python-dateutil<3.0.0,>=
```

[9]:
```python
from ibm_watsonx_ai.foundation_models import ModelInference
from ibm_watsonx_ai import Credentials
```

[10]:
```python
import os
from ibm_watsonx_ai import APIClient, Credentials
import getpass

credentials = Credentials(
    url="https://eu-gb.ml.cloud.ibm.com",
    api_key=getpass.getpass("Please enter your api key (hit enter): ")
)
```

```
Please enter your api key (hit enter):  ········
```

[12]:
```python
client = APIClient(credentials)
```

Connecting to a space

A space will be be used to host the promoted AI Service.

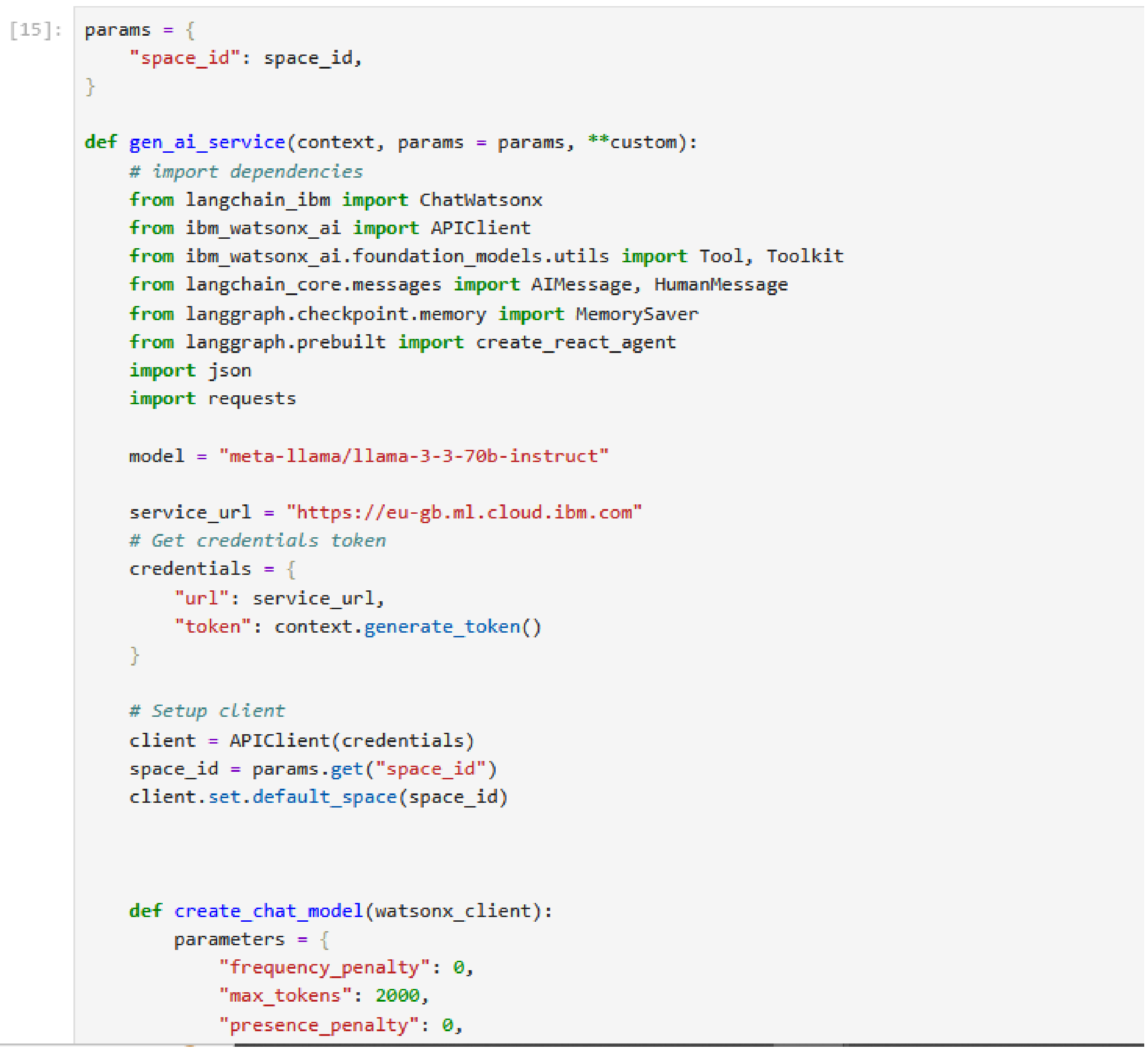
[13]:
```python
space_id = "fee31274-660c-4816-9932-98bb17284cd0"
client.set.default_space(space_id)
```

[13]:
```
'SUCCESS'
```

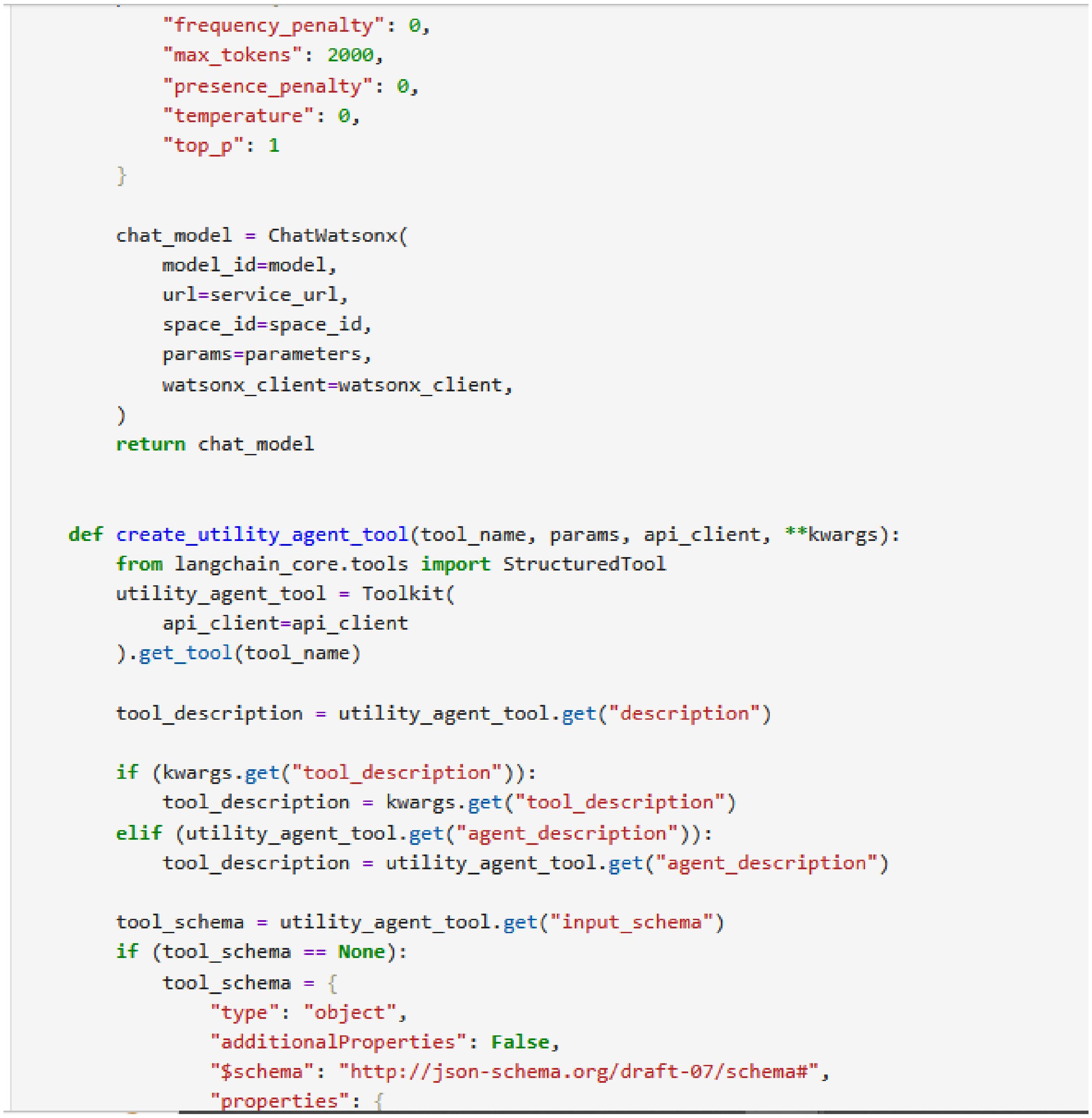
Promote asset(s) to space

We will now promote assets we will need to stage in the space so that we can access their data from the AI service.

[14]:
```python
source_project_id = "7ced1da6-f122-408d-ba19-c2cb332fa8d4"
```



```
[15]: params = {
          "space_id": space_id,
      }

      def gen_ai_service(context, params = params, **custom):
          # import dependencies
          from langchain_ibm import ChatWatsonx
          from ibm_watsonx_ai import APIClient
          from ibm_watsonx_ai.foundation_models.utils import Tool, Toolkit
          from langchain_core.messages import AIMessage, HumanMessage
          from langgraph.checkpoint.memory import MemorySaver
          from langgraph.prebuilt import create_react_agent
          import json
          import requests

          model = "meta-llama/llama-3-3-70b-instruct"

          service_url = "https://eu-gb.ml.cloud.ibm.com"
          # Get credentials token
          credentials = {
              "url": service_url,
              "token": context.generate_token()
          }

          # Setup client
          client = APIClient(credentials)
          space_id = params.get("space_id")
          client.set.default_space(space_id)



          def create_chat_model(watsonx_client):
              parameters = {
                  "frequency_penalty": 0,
                  "max_tokens": 2000,
                  "presence_penalty": 0,
```

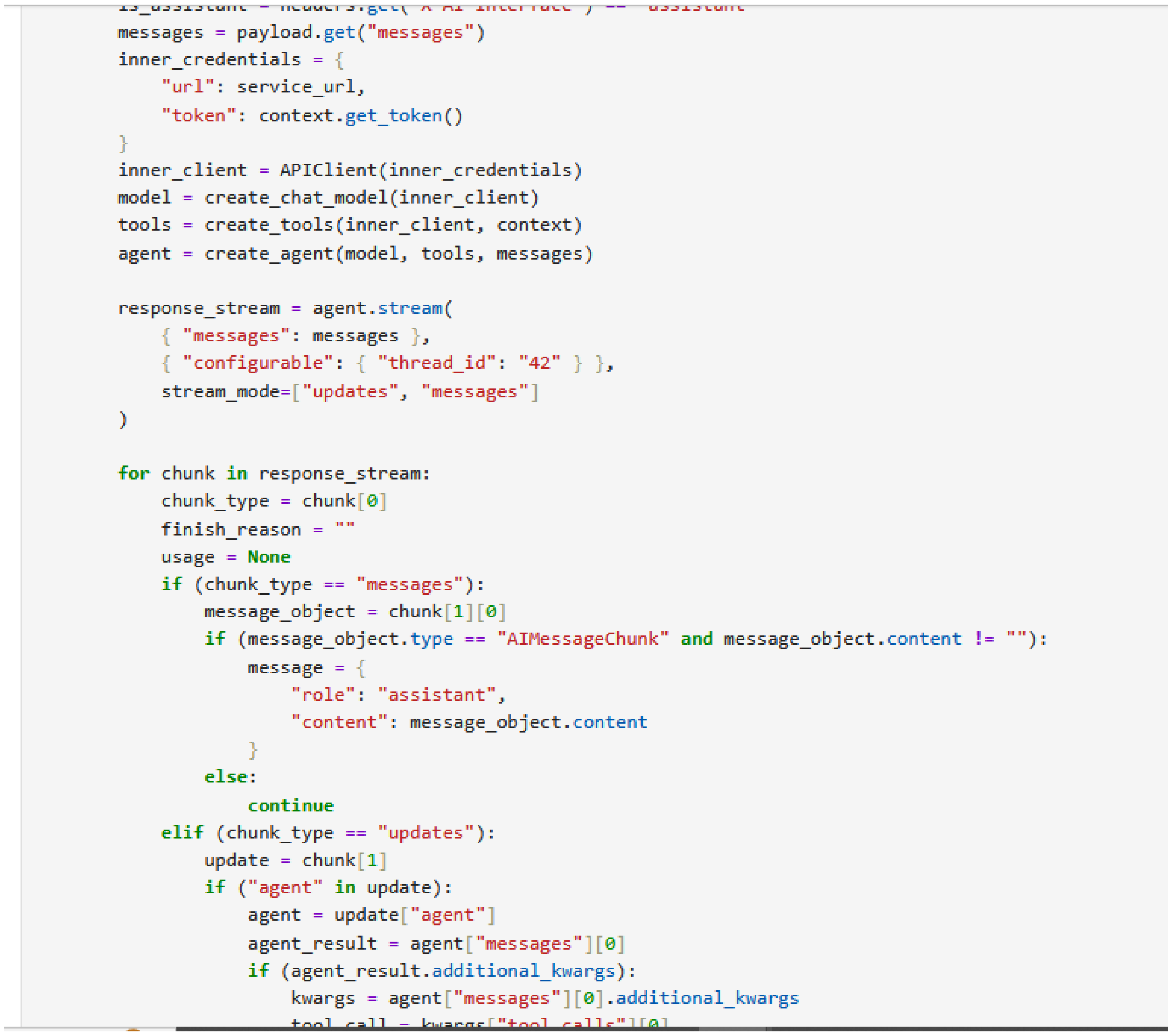


```
        "frequency_penalty": 0,
        "max_tokens": 2000,
        "presence_penalty": 0,
        "temperature": 0,
        "top_p": 1
    }

    chat_model = ChatWatsonx(
        model_id=model,
        url=service_url,
        space_id=space_id,
        params=parameters,
        watsonx_client=watsonx_client,
    )
    return chat_model


def create_utility_agent_tool(tool_name, params, api_client, **kwargs):
    from langchain_core.tools import StructuredTool
    utility_agent_tool = Toolkit(
        api_client=api_client
    ).get_tool(tool_name)

    tool_description = utility_agent_tool.get("description")

    if (kwargs.get("tool_description")):
        tool_description = kwargs.get("tool_description")
    elif (utility_agent_tool.get("agent_description")):
        tool_description = utility_agent_tool.get("agent_description")

    tool_schema = utility_agent_tool.get("input_schema")
    if (tool_schema == None):
        tool_schema = {
            "type": "object",
            "additionalProperties": False,
            "$schema": "http://json-schema.org/draft-07/schema#",
            "properties": {
```

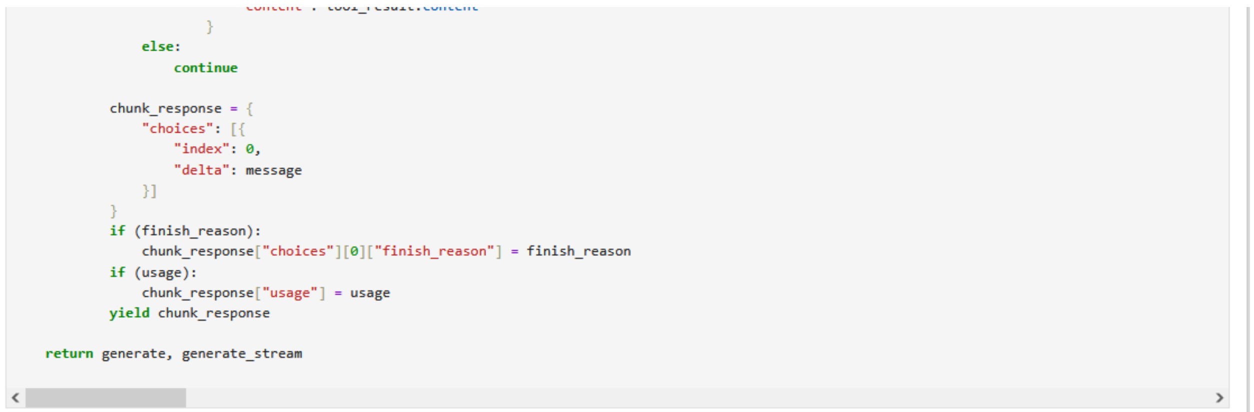
```
            "type": "object",
            "additionalProperties": False,
            "$schema": "http://json-schema.org/draft-07/schema#",
            "properties": {
                "input": {
                    "description": "input for the tool",
                    "type": "string"
                }
            }
        }

    def run_tool(**tool_input):
        query = tool_input
        if (utility_agent_tool.get("input_schema") == None):
            query = tool_input.get("input")

        results = utility_agent_tool.run(
            input=query,
            config=params
        )

        return results.get("output")

    return StructuredTool(
        name=tool_name,
        description = tool_description,
        func=run_tool,
        args_schema=tool_schema
    )


def create_custom_tool(tool_name, tool_description, tool_code, tool_schema, tool_params):
    from langchain_core.tools import StructuredTool
    import ast

    def call_tool(**kwargs):
        tree = ast.parse(tool_code, mode="exec")
```

```python
        def call_tool(**kwargs):
            tree = ast.parse(tool_code, mode="exec")
            custom_tool_functions = [ x for x in tree.body if isinstance(x, ast.FunctionDef) ]
            function_name = custom_tool_functions[0].name
            compiled_code = compile(tree, 'custom_tool', 'exec')
            namespace = tool_params if tool_params else {}
            exec(compiled_code, namespace)
            return namespace[function_name](**kwargs)

        tool = StructuredTool(
            name=tool_name,
            description = tool_description,
            func=call_tool,
            args_schema=tool_schema
        )
        return tool

    def create_custom_tools():
        custom_tools = []


    def create_tools(inner_client, context):
        tools = []

        config = None
        tools.append(create_utility_agent_tool("GoogleSearch", config, inner_client))
        return tools

    def create_agent(model, tools, messages):
        memory = MemorySaver()
        instructions = """# Notes
- Use markdown syntax for formatting code snippets, links, JSON, tables, images, files.
- Any HTML tags must be wrapped in block quotes, for example ```<html>```.
- When returning code blocks, specify language.
- Sometimes, things don't go as planned. Tools may not provide useful information on the first few tries. You should always try a few different approach
- When the tool doesn't give you what you were asking for, you must either use another tool or a different tool input.
- If you need to call a tool to compute something, always call it instead of saying you will call it.

If a tool returns an IMAGE in the result, you must include it in your answer as Markdown.

Example:

Tool result: IMAGE({commonApiUrl}/wx/v1-beta/utility_agent_tools/cache/images/plt-04e3c91ae04b47f8934a4e6b7d1fdc2c.png)
Markdown to return to user: ![Generated image]({commonApiUrl}/wx/v1-beta/utility_agent_tools/cache/images/plt-04e3c91ae04b47f8934a4e6b7d1fdc2c.png)

You are a friendly Travel planner agent AI and should begin by collecting key trip details such as the destination, travel dates, departure city, number
        for message in messages:
            if message["role"] == "system":
                instructions += message["content"]
        graph = create_react_agent(model, tools=tools, checkpointer=memory, state_modifier=instructions)
        return graph

    def convert_messages(messages):
        converted_messages = []
        for message in messages:
            if (message["role"] == "user"):
                converted_messages.append(HumanMessage(content=message["content"]))
            elif (message["role"] == "assistant"):
                converted_messages.append(AIMessage(content=message["content"]))
        return converted_messages

    def generate(context):
        payload = context.get_json()
        messages = payload.get("messages")
        inner_credentials = {
            "url": service_url,
            "token": context.get_token()
        }

        inner_client = APIClient(inner_credentials)
        model = create_chat_model(inner_client)
        tools = create_tools(inner_client, context)
        agent = create_agent(model, tools, messages)
```

```
    generated_response = agent.invoke(
        { "messages": convert_messages(messages) },
        { "configurable": { "thread_id": "42" } }
    )

    last_message = generated_response["messages"][-1]
    generated_response = last_message.content

    execute_response = {
        "headers": {
            "Content-Type": "application/json"
        },
        "body": {
            "choices": [{
                "index": 0,
                "message": {
                   "role": "assistant",
                   "content": generated_response
                }
            }]
        }
    }

    return execute_response

def generate_stream(context):
    print("Generate stream", flush=True)
    payload = context.get_json()
    headers = context.get_headers()
    is_assistant = headers.get("X-Ai-Interface") == "assistant"
    messages = payload.get("messages")
    inner_credentials = {
        "url": service_url,
```

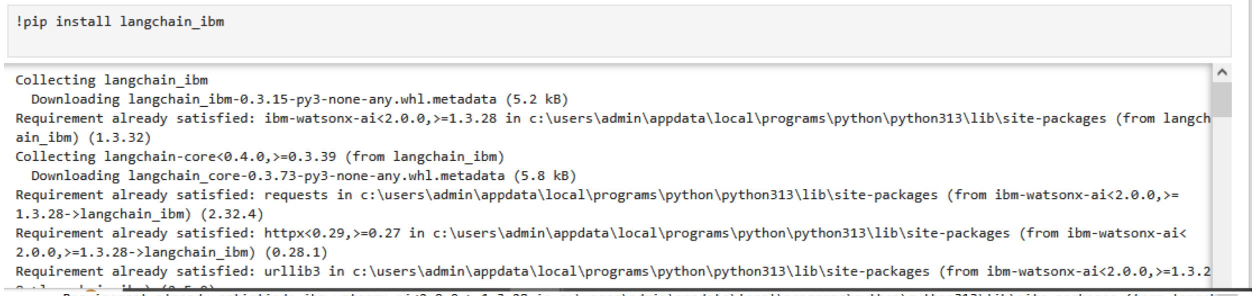


```python
    is_assistant = headers.get("X-AI-Interface") == "assistant"
    messages = payload.get("messages")
    inner_credentials = {
        "url": service_url,
        "token": context.get_token()
    }
    inner_client = APIClient(inner_credentials)
    model = create_chat_model(inner_client)
    tools = create_tools(inner_client, context)
    agent = create_agent(model, tools, messages)

    response_stream = agent.stream(
        { "messages": messages },
        { "configurable": { "thread_id": "42" } },
        stream_mode=["updates", "messages"]
    )

    for chunk in response_stream:
        chunk_type = chunk[0]
        finish_reason = ""
        usage = None
        if (chunk_type == "messages"):
            message_object = chunk[1][0]
            if (message_object.type == "AIMessageChunk" and message_object.content != ""):
                message = {
                    "role": "assistant",
                    "content": message_object.content
                }
            else:
                continue
        elif (chunk_type == "updates"):
            update = chunk[1]
            if ("agent" in update):
                agent = update["agent"]
                agent_result = agent["messages"][0]
                if (agent_result.additional_kwargs):
                    kwargs = agent["messages"][0].additional_kwargs
                    tool_call = kwargs["tool_calls"][0]
        agent_result = agent["messages"][0]
        if (agent_result.additional_kwargs):
            kwargs = agent["messages"][0].additional_kwargs
            tool_call = kwargs["tool_calls"][0]
            if (is_assistant):
                message = {
                    "role": "assistant",
                    "step_details": {
                        "type": "tool_calls",
                        "tool_calls": [
                            {
                                "id": tool_call["id"],
                                "name": tool_call["function"]["name"],
                                "args": tool_call["function"]["arguments"]
                            }
                        ]
                    }
                }
            else:
                message = {
                    "role": "assistant",
                    "tool_calls": [
                        {
                            "id": tool_call["id"],
                            "type": "function",
                            "function": {
                                "name": tool_call["function"]["name"],
                                "arguments": tool_call["function"]["arguments"]
                            }
                        }
                    ]
                }
        elif (agent_result.response_metadata):
            # Final update
            message = {
                "role": "assistant",
                "content": agent_result.content
            message = {
                "role": "assistant",
                "content": agent_result.content
            }
            finish_reason = agent_result.response_metadata["finish_reason"]
            if (finish_reason):
                message["content"] = ""

            usage = {
                "completion_tokens": agent_result.usage_metadata["output_tokens"],
                "prompt_tokens": agent_result.usage_metadata["input_tokens"],
                "total_tokens": agent_result.usage_metadata["total_tokens"]
            }
    elif ("tools" in update):
        tools = update["tools"]
        tool_result = tools["messages"][0]
        if (is_assistant):
            message = {
                "role": "assistant",
                "step_details": {
                    "type": "tool_response",
                    "id": tool_result.id,
                    "tool_call_id": tool_result.tool_call_id,
                    "name": tool_result.name,
                    "content": tool_result.content
                }
            }
        else:
            message = {
                "role": "tool",
                "id": tool_result.id,
                "tool_call_id": tool_result.tool_call_id,
                "name": tool_result.name,
                "content": tool_result.content
            }
    else:
        continue
```



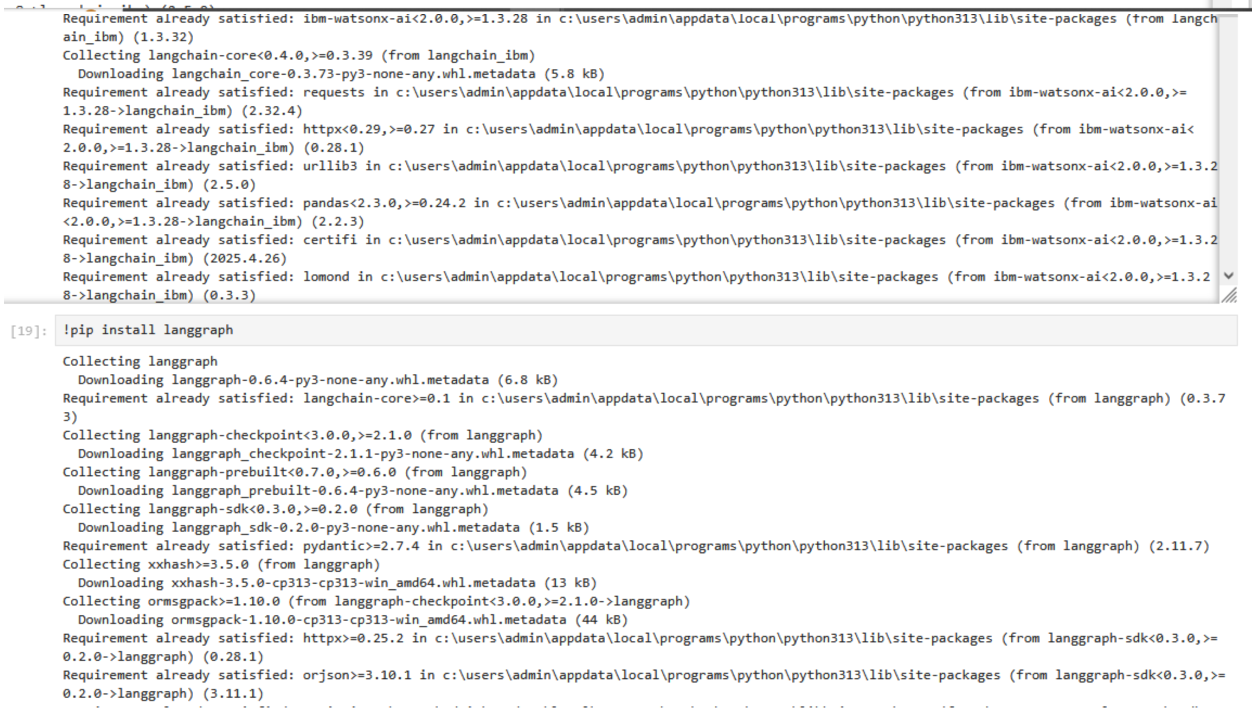
```
                            "content": tool_result.content
                        }
                else:
                    continue

            chunk_response = {
                "choices": [{
                    "index": 0,
                    "delta": message
                }]
            }
            if (finish_reason):
                chunk_response["choices"][0]["finish_reason"] = finish_reason
            if (usage):
                chunk_response["usage"] = usage
            yield chunk_response

    return generate, generate_stream
```

2.2 Test locally



```
!pip install langchain_ibm
```

```
Collecting langchain_ibm
  Downloading langchain_ibm-0.3.15-py3-none-any.whl.metadata (5.2 kB)
Requirement already satisfied: ibm-watsonx-ai<2.0.0,>=1.3.28 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from langch
ain_ibm) (1.3.32)
Collecting langchain-core<0.4.0,>=0.3.39 (from langchain_ibm)
  Downloading langchain_core-0.3.73-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: requests in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai<2.0.0,>=
1.3.28->langchain_ibm) (2.32.4)
Requirement already satisfied: httpx<0.29,>=0.27 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai<
2.0.0,>=1.3.28->langchain_ibm) (0.28.1)
Requirement already satisfied: urllib3 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai<2.0.0,>=1.3.2
```



```
Requirement already satisfied: ibm-watsonx-ai<2.0.0,>=1.3.28 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from langch
ain_ibm) (1.3.32)
Collecting langchain-core<0.4.0,>=0.3.39 (from langchain_ibm)
  Downloading langchain_core-0.3.73-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: requests in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai<2.0.0,>=
1.3.28->langchain_ibm) (2.32.4)
Requirement already satisfied: httpx<0.29,>=0.27 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai<
2.0.0,>=1.3.28->langchain_ibm) (0.28.1)
Requirement already satisfied: urllib3 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai<2.0.0,>=1.3.2
```

```
8->langchain_ibm) (2.5.0)
Requirement already satisfied: pandas<2.3.0,>=0.24.2 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai
<2.0.0,>=1.3.28->langchain_ibm) (2.2.3)
Requirement already satisfied: certifi in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai<2.0.0,>=1.3.2
8->langchain_ibm) (2025.4.26)
Requirement already satisfied: lomond in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from ibm-watsonx-ai<2.0.0,>=1.3.2
8->langchain_ibm) (0.3.3)
```

```
[19]: !pip install langgraph
```

```
Collecting langgraph
  Downloading langgraph-0.6.4-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: langchain-core>=0.1 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from langgraph) (0.3.7
3)
Collecting langgraph-checkpoint<3.0.0,>=2.1.0 (from langgraph)
  Downloading langgraph_checkpoint-2.1.1-py3-none-any.whl.metadata (4.2 kB)
Collecting langgraph-prebuilt<0.7.0,>=0.6.0 (from langgraph)
  Downloading langgraph_prebuilt-0.6.4-py3-none-any.whl.metadata (4.5 kB)
Collecting langgraph-sdk<0.3.0,>=0.2.0 (from langgraph)
  Downloading langgraph_sdk-0.2.0-py3-none-any.whl.metadata (1.5 kB)
Requirement already satisfied: pydantic>=2.7.4 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from langgraph) (2.11.7)
Collecting xxhash>=3.5.0 (from langgraph)
  Downloading xxhash-3.5.0-cp313-cp313-win_amd64.whl.metadata (13 kB)
Collecting ormsgpack>=1.10.0 (from langgraph-checkpoint<3.0.0,>=2.1.0->langgraph)
  Downloading ormsgpack-1.10.0-cp313-cp313-win_amd64.whl.metadata (44 kB)
Requirement already satisfied: httpx>=0.25.2 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from langgraph-sdk<0.3.0,>=
0.2.0->langgraph) (0.28.1)
Requirement already satisfied: orjson>=3.10.1 in c:\users\admin\appdata\local\programs\python\python313\lib\site-packages (from langgraph-sdk<0.3.0,>=
0.2.0->langgraph) (3.11.1)
```

```python
# Initialize AI Service function locally
from ibm_watsonx_ai.deployments import RuntimeContext

context = RuntimeContext(api_client=client)

streaming = False
findex = 1 if streaming else 0
local_function = gen_ai_service(context,  space_id=space_id)[findex]
messages = []
```

```python
local_question = "Change this question to test your function"

messages.append({ "role" : "user", "content": local_question })

context = RuntimeContext(api_client=client, request_payload_json={"messages": messages})

response = local_function(context)

result = ''

if (streaming):
    for chunk in response:
        print(chunk, end="\n\n", flush=True)
else:
    print(response)
```

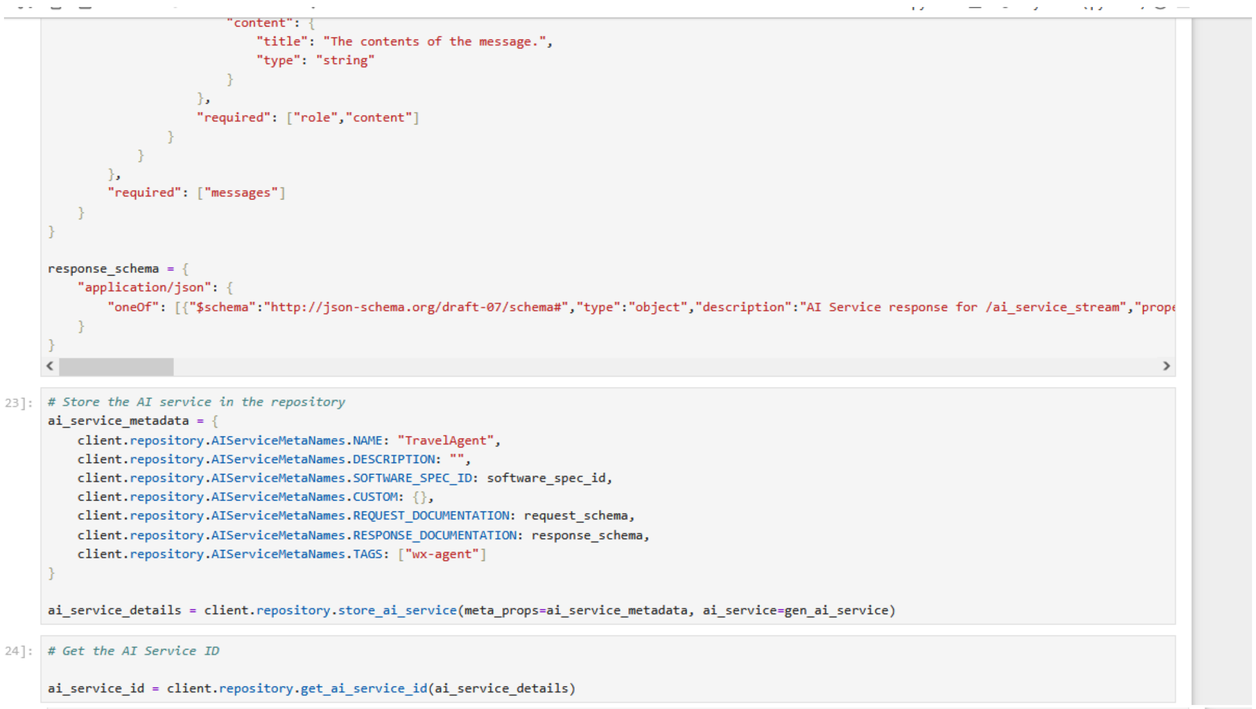
```
{'headers': {'Content-Type': 'application/json'}, 'body': {'choices': [{'index': 0, 'message': {'role': 'assistant', 'content': 'The search results pro
vide various information on the concept of "test function" in different contexts, including mathematics, programming, and health. \n\nIn mathematics, a
test function is a type of function used in the definition and application of distributions, which are topological vector spaces. \n\nIn programming, t
esting a function refers to the process of verifying that a function behaves as expected and produces the correct output for a given input. \n\nIn heal
th, Function Health is a company that offers lab tests and personalized action plans to help individuals identify lifestyle changes and promote healthy
living. \n\nThe search results also include links to resources on optimization test functions and datasets, as well as information on how to test a fun
ction in specific programming languages like Julia and Swift. \n\nOverall, the search results provide a range of information on the concept of "test fu
nction" and its applications in different fields.'}}]}}
```

3. Store and deploy the AI Service

Before you can deploy the AI Service, you must store the AI service in your watsonx.ai repository.

[21]:
```python
# Look up software specification for the AI service
software_spec_id_in_project = "45f12dfe-aa78-5b8d-9f38-0ee223c47309"
software_spec_id = ""

try:
    software_spec_id = client.software_specifications.get_id_by_name("runtime-24.1-py3.11")
except:
    software_spec_id = client.spaces.promote(software_spec_id_in_project, source_project_id, space_id)
```

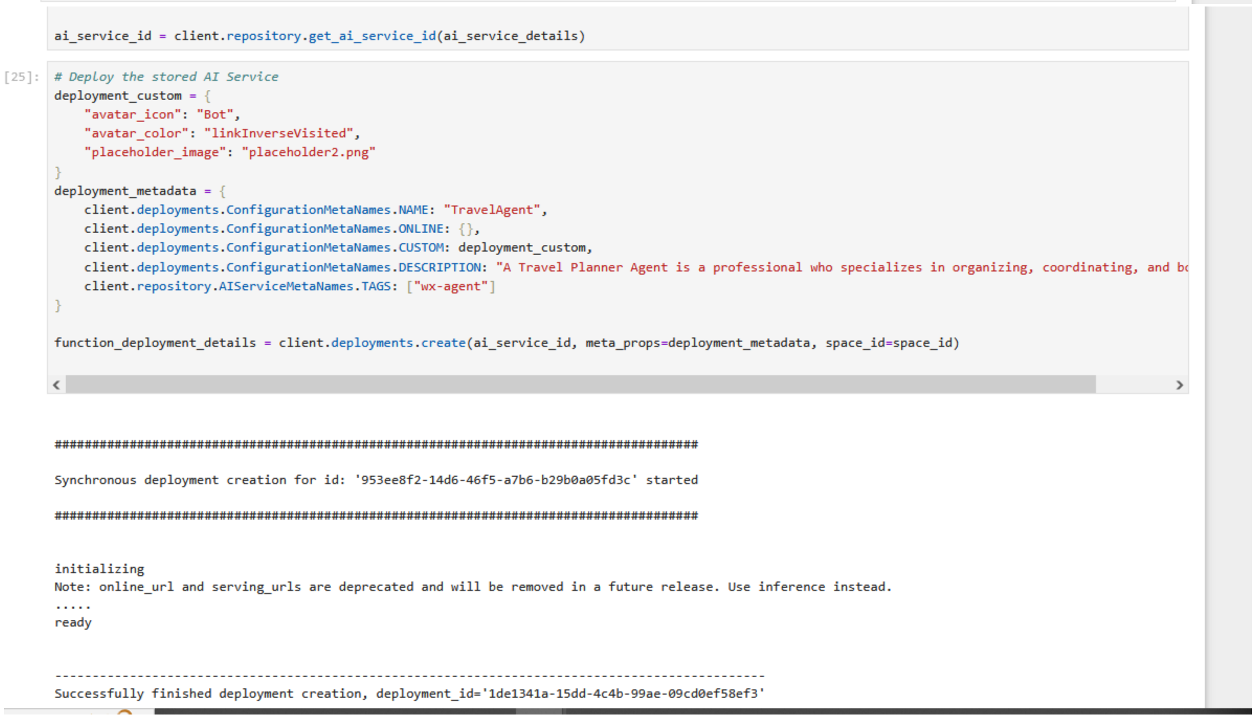
[22]:
```python
# Define the request and response schemas for the AI service
request_schema = {
    "application/json": {
        "$schema": "http://json-schema.org/draft-07/schema#",
        "type": "object",
        "properties": {
            "messages": {
                "title": "The messages for this chat session.",
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "role": {
                            "title": "The role of the message author.",
                            "type": "string",
                            "enum": ["user","assistant"]
                        },
                        "content": {
                            "title": "The contents of the message.",
                            "type": "string"
```



```
                        "content": {
                            "title": "The contents of the message.",
                            "type": "string"
                        }
                    },
                    "required": ["role","content"]
                }
            }
        },
        "required": ["messages"]
    }
}

response_schema = {
    "application/json": {
        "oneOf": [{"$schema":"http://json-schema.org/draft-07/schema#","type":"object","description":"AI Service response for /ai_service_stream","prope
    }
}
```

23]:
```
# Store the AI service in the repository
ai_service_metadata = {
    client.repository.AIServiceMetaNames.NAME: "TravelAgent",
    client.repository.AIServiceMetaNames.DESCRIPTION: "",
    client.repository.AIServiceMetaNames.SOFTWARE_SPEC_ID: software_spec_id,
    client.repository.AIServiceMetaNames.CUSTOM: {},
    client.repository.AIServiceMetaNames.REQUEST_DOCUMENTATION: request_schema,
    client.repository.AIServiceMetaNames.RESPONSE_DOCUMENTATION: response_schema,
    client.repository.AIServiceMetaNames.TAGS: ["wx-agent"]
}

ai_service_details = client.repository.store_ai_service(meta_props=ai_service_metadata, ai_service=gen_ai_service)
```

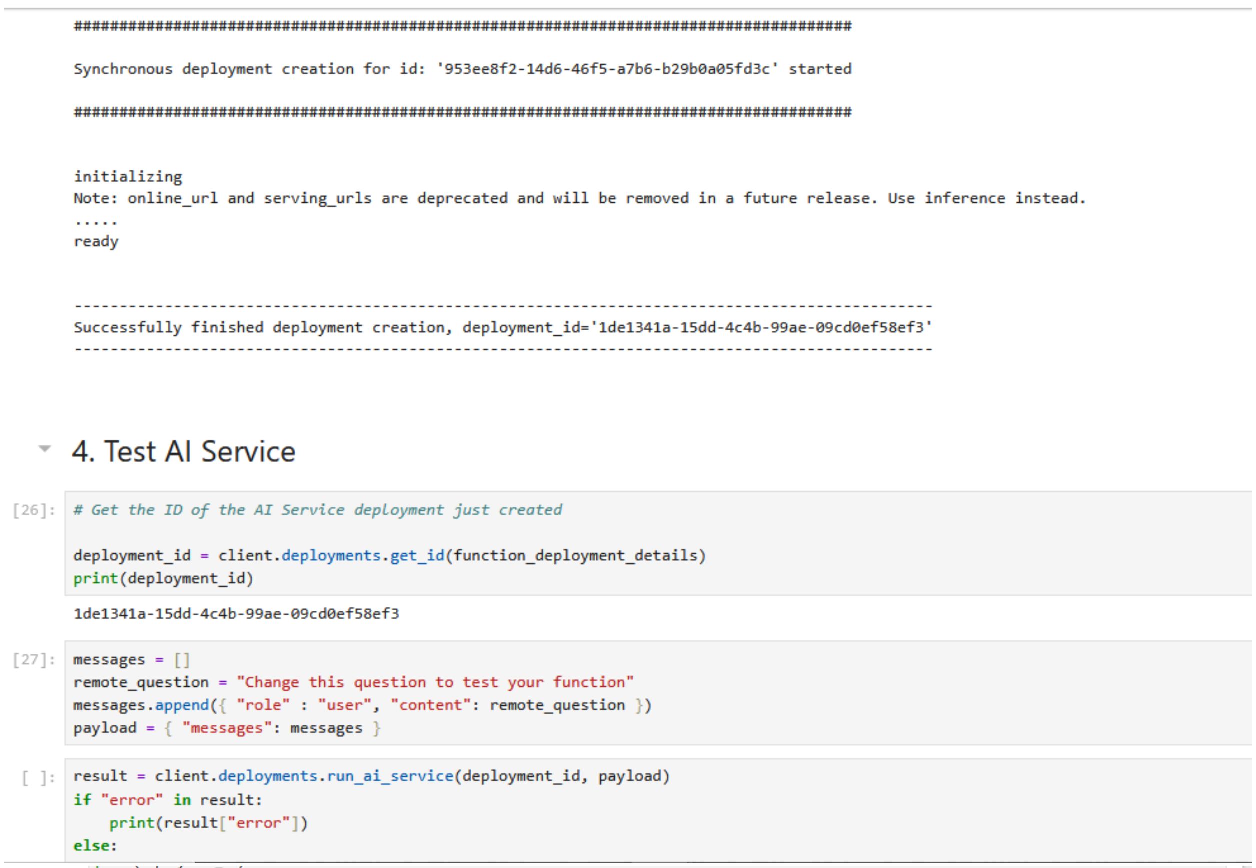
24]:
```
# Get the AI Service ID

ai_service_id = client.repository.get_ai_service_id(ai_service_details)
```



```
ai_service_id = client.repository.get_ai_service_id(ai_service_details)
```

[25]:
```
# Deploy the stored AI Service
deployment_custom = {
    "avatar_icon": "Bot",
    "avatar_color": "linkInverseVisited",
    "placeholder_image": "placeholder2.png"
}
deployment_metadata = {
    client.deployments.ConfigurationMetaNames.NAME: "TravelAgent",
    client.deployments.ConfigurationMetaNames.ONLINE: {},
    client.deployments.ConfigurationMetaNames.CUSTOM: deployment_custom,
    client.deployments.ConfigurationMetaNames.DESCRIPTION: "A Travel Planner Agent is a professional who specializes in organizing, coordinating, and bo
    client.repository.AIServiceMetaNames.TAGS: ["wx-agent"]
}

function_deployment_details = client.deployments.create(ai_service_id, meta_props=deployment_metadata, space_id=space_id)
```

```
######################################################################################

Synchronous deployment creation for id: '953ee8f2-14d6-46f5-a7b6-b29b0a05fd3c' started

######################################################################################


initializing
Note: online_url and serving_urls are deprecated and will be removed in a future release. Use inference instead.
.....
ready


------------------------------------------------------------------------------------------------
Successfully finished deployment creation, deployment_id='1de1341a-15dd-4c4b-99ae-09cd0ef58ef3'
```



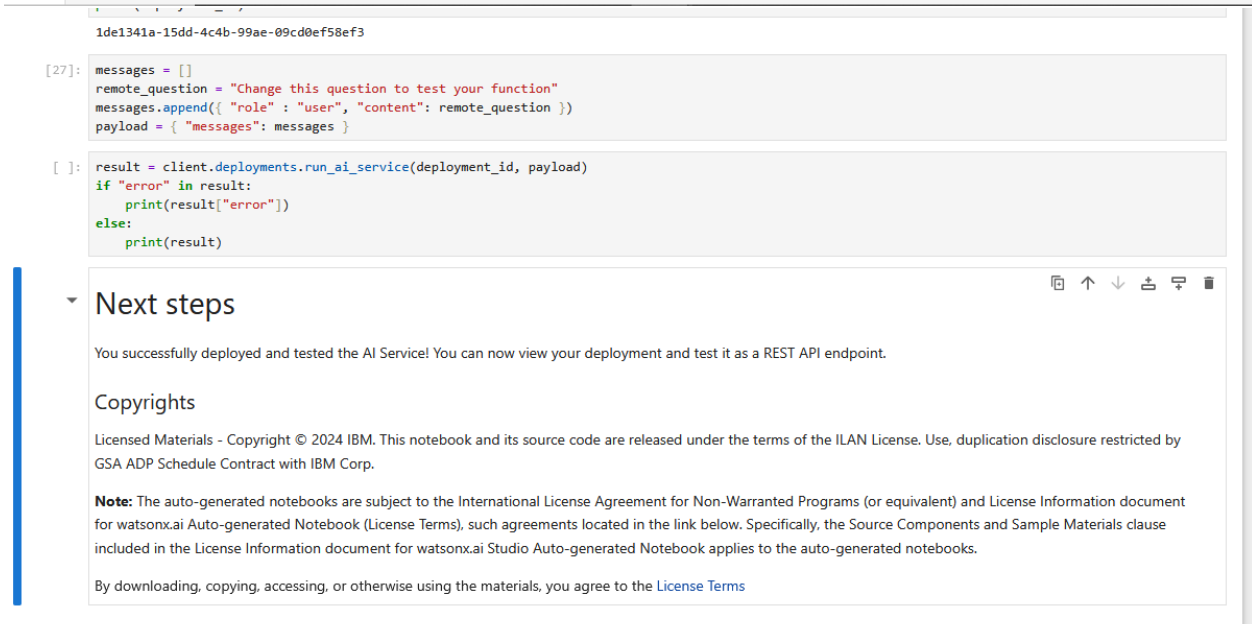
```
######################################################################################

Synchronous deployment creation for id: '953ee8f2-14d6-46f5-a7b6-b29b0a05fd3c' started

######################################################################################


initializing
Note: online_url and serving_urls are deprecated and will be removed in a future release. Use inference instead.
.....
ready


-----------------------------------------------------------------------------------------------
Successfully finished deployment creation, deployment_id='1de1341a-15dd-4c4b-99ae-09cd0ef58ef3'
-----------------------------------------------------------------------------------------------
```

4. Test AI Service

```
[26]: # Get the ID of the AI Service deployment just created

      deployment_id = client.deployments.get_id(function_deployment_details)
      print(deployment_id)

      1de1341a-15dd-4c4b-99ae-09cd0ef58ef3
```

```
[27]: messages = []
      remote_question = "Change this question to test your function"
      messages.append({ "role" : "user", "content": remote_question })
      payload = { "messages": messages }
```

```
[ ]:  result = client.deployments.run_ai_service(deployment_id, payload)
      if "error" in result:
          print(result["error"])
      else:
```



```
      1de1341a-15dd-4c4b-99ae-09cd0ef58ef3
```

```
[27]: messages = []
      remote_question = "Change this question to test your function"
      messages.append({ "role" : "user", "content": remote_question })
      payload = { "messages": messages }
```

```
[ ]:  result = client.deployments.run_ai_service(deployment_id, payload)
      if "error" in result:
          print(result["error"])
      else:
          print(result)
```

Next steps

You successfully deployed and tested the AI Service! You can now view your deployment and test it as a REST API endpoint.

Copyrights