Agent Engine - Agent Development Kit

Source URL: https://google.github.io/adk-docs/deploy/agent-engine/

Deploy to Vertex AI Agent Engine¶

Currently supported in Python

Agent Engine is a fully managed Google Cloud service enabling developers to deploy, manage, and scale AI agents in production. Agent Engine handles the infrastructure to scale agents in production so you can focus on creating intelligent and impactful applications.

```
from vertexai import agent_engines

remote_app = agent_engines.create(
    agent_engine=root_agent,
    requirements=[
        "google-cloud-aiplatform[adk,agent_engines]",
    ]
)
```

Install Vertex AI SDK¶

Agent Engine is part of the Vertex AI SDK for Python. For more information, you can review the Agent Engine quickstart documentation.

Install the Vertex AI SDK¶

```
pip install google-cloud-aiplatform[adk,agent_engines]
```

Info

Agent Engine only supported Python version >=3.9 and <=3.12.

Initialization¶

```
import vertexai

PROJECT_ID = "your-project-id"
LOCATION = "us-central1"
STAGING_BUCKET = "gs://your-google-cloud-storage-bucket"

vertexai.init(
    project=PROJECT_ID,
    location=LOCATION,
    staging_bucket=STAGING_BUCKET,
)
```

For `LOCATION`, you can check out the list of supported regions in Agent Engine.

Create your agent¶

You can use the sample agent below, which has two tools (to get weather or retrieve the time in a specified city):

```
import datetime
from zoneinfo import ZoneInfo
from google.adk.agents import Agent

def get_weather(city: str) -> dict:
    """Retrieves the current weather report for a specified city.

    Args:
        city (str): The name of the city for which to retrieve the wea

    Returns:
        dict: status and result or error msg.
```

```
    """
    if city.lower() == "new york":
        return {
            "status": "success",
            "report": (
                "The weather in New York is sunny with a temperature o
                " Celsius (77 degrees Fahrenheit)."
            ),
        }
    else:
        return {
            "status": "error",
            "error_message": f"Weather information for '{city}' is not
        }

def get_current_time(city: str) -> dict:
    """Returns the current time in a specified city.

    Args:
        city (str): The name of the city for which to retrieve the cur

    Returns:
        dict: status and result or error msg.
    """

    if city.lower() == "new york":
        tz_identifier = "America/New_York"
    else:
        return {
            "status": "error",
            "error_message": (
                f"Sorry, I don't have timezone information for {city}.
            ),
        }

    tz = ZoneInfo(tz_identifier)
```

```
    now = datetime.datetime.now(tz)
    report = (
        f'The current time in {city} is {now.strftime("%Y-%m-%d %H:%M:
    )
    return {"status": "success", "report": report}

root_agent = Agent(
    name="weather_time_agent",
    model="gemini-2.0-flash",
    description=(
        "Agent to answer questions about the time and weather in a cit
    ),
    instruction=(
        "You are a helpful agent who can answer user questions about t
    ),
    tools=[get_weather, get_current_time],
)
```

Prepare your agent for Agent Engine¶

Use `reasoning_engines.AdkApp()` to wrap your agent to make it deployable to Agent Engine

```
from vertexai.preview import reasoning_engines

app = reasoning_engines.AdkApp(
    agent=root_agent,
    enable_tracing=True,
)
```

Try your agent locally¶

You can try it locally before deploying to Agent Engine.

Create session (local)¶

```
session = app.create_session(user_id="u_123")
session
```

Expected output for `create_session` (local):

```
Session(id='c6a33dae-26ef-410c-9135-b434a528291f', app_name='default-
```

List sessions (local)¶

```
app.list_sessions(user_id="u_123")
```

Expected output for `list_sessions` (local):

```
ListSessionsResponse(session_ids=['c6a33dae-26ef-410c-9135-b434a52829
```

Get a specific session (local)¶

```
session = app.get_session(user_id="u_123", session_id=session.id)
session
```

Expected output for `get_session` (local):

```
Session(id='c6a33dae-26ef-410c-9135-b434a528291f', app_name='default-
```

Send queries to your agent (local)¶

```
for event in app.stream_query(
    user_id="u_123",
```

```
    session_id=session.id,
    message="whats the weather in new york",
):
print(event)
```

Expected output for `stream_query` (local):

```
{'parts': [{'function_call': {'id': 'af-a33fedb0-29e6-4d0c-9eb3-00c40
{'parts': [{'function_response': {'id': 'af-a33fedb0-29e6-4d0c-9eb3-00
{'parts': [{'text': 'The weather in New York is sunny with a temperatu
```

Deploy your agent to Agent Engine¶

```
from vertexai import agent_engines

remote_app = agent_engines.create(
    agent_engine=root_agent,
    requirements=[
        "google-cloud-aiplatform[adk,agent_engines]"
    ]
)
```

This step may take several minutes to finish. Each deployed agent has a unique identifier. You can run the following command to get the resource_name identifier for your deployed agent:

```
remote_app.resource_name
```

The response should look like the following string:

```
f"projects/{PROJECT_NUMBER}/locations/{LOCATION}/reasoningEngines/{RE
```

For additional details, you can visit the Agent Engine documentation deploying an agent and managing deployed agents.

Try your agent on Agent Engine¶

Create session (remote)¶

```
remote_session = remote_app.create_session(user_id="u_456")
remote_session
```

Expected output for `create_session` (remote):

```
{'events': [],
'user_id': 'u_456',
'state': {},
'id': '7543472750996750336',
'app_name': '7917477678498709504',
'last_update_time': 1743683353.030133}
```

`id` is the session ID, and `app_name` is the resource ID of the deployed agent on Agent Engine.

List sessions (remote)¶

```
remote_app.list_sessions(user_id="u_456")
```

Get a specific session (remote)¶

```
remote_app.get_session(user_id="u_456", session_id=remote_session["id
```

Note

While using your agent locally, session ID is stored in `session.id`, when using your agent remotely on Agent Engine, session ID is stored in `remote_session["id"]`.

Send queries to your agent (remote)¶

```
for event in remote_app.stream_query(
    user_id="u_456",
    session_id=remote_session["id"],
    message="whats the weather in new york",
):
    print(event)
```

Expected output for `stream_query` (remote):

```
{'parts': [{'function_call': {'id': 'af-f1906423-a531-4ecf-a1ef-723b0
{'parts': [{'function_response': {'id': 'af-f1906423-a531-4ecf-a1ef-72
{'parts': [{'text': 'The weather in New York is sunny with a temperatu
```

Clean up¶

After you have finished, it is a good practice to clean up your cloud resources. You can delete the deployed Agent Engine instance to avoid any unexpected charges on your Google Cloud account.

```
remote_app.delete(force=True)
```

`force=True` will also delete any child resources that were generated from the deployed agent, such as sessions.