🚀 Deploy ADK Agent to Vertex AI Agent Engine

Welcome to the final day of the Kaggle 5-day Agents course!

In the previous notebook you learned how to use Agent2Agent Protocol to make your agents interoperable.

Now, let's take the final step: deploying your agents to production using Vertex AI Agent Engine.

💡 Scaling Your Agent

You've built an amazing AI agent. It works perfectly on your machine. You can chat with it, it responds intelligently, and everything seems ready. But there's a problem.

> **Your agent is not publicly available!**

It only lives in your notebook and development environment. When you stop your notebook session, it stops working. Your teammates can't access it. Your users can't interact with it. And this is precisely why we need to deploy the agents!

🎯 What You'll Learn

In this notebook, you'll:

- ✅ Build a production-ready ADK agent
- ✅ Deploy your agent to **Vertex AI Agent Engine** using the ADK CLI
- ✅ Test your deployed agent with Python SDK
- ✅ Monitor and manage deployed agents in Google Cloud Console
- ✅ Understand how to add Memory to your Agent using Vertex AI Memory Bank
- ✅ Understand cost management and cleanup best practices

‼️ Please Read

> ❌ ℹ️ **Note: No submission required!** This notebook is for your hands-on practice and learning only. You **do not** need to submit it anywhere to complete the course.

> ⏸️ **Note:** When you first start the notebook via running a cell you might see a banner in the notebook header that reads **"Waiting for the next available notebook"**. The queue should drop rapidly; however, during peak bursts you might have to wait a few minutes.

> ❌ **Note:** Avoid using the **Run all** cells command as this can trigger a QPM limit resulting in 429 errors when calling the backing model. Suggested flow is to run each cell in order - one at a time. See FAQ on 429 errors for more information.

For help: Ask questions on the Kaggle Discord server.

📖 Get started with Kaggle Notebooks

If this is your first time using Kaggle Notebooks, welcome! You can learn more about using Kaggle Notebooks in the documentation.

Here's how to get started:

1. Verify Your Account (Required)

To use the Kaggle Notebooks in this course, you'll need to verify your account with a phone number.

You can do this in your Kaggle settings.

2. Make Your Own Copy

To run any code in this notebook, you first need your own editable copy.

Click the `Copy and Edit` button in the top-right corner.

▲ 0 **Copy & Edit** ⋮

This creates a private copy of the notebook just for you.

3. Run Code Cells

Once you have your copy, you can run code.

Click the ▶️ Run button next to any code cell to execute it.

```
%pip install
```

Run the cells in order from top to bottom.

4. If You Get Stuck

To restart: Select `Factory reset` from the `Run` menu.

For help: Ask questions on the Kaggle Discord server.

⚙️ Section 1: Setup

1.1: ⚠️ **Important: Prerequisites**

This notebook requires a **Google Cloud Platform (GCP) account** to deploy agents to Vertex AI Agent Engine.

If you don't have a GCP account yet:

✅ Step 1. **Create a free Google Cloud account** - Sign up here

- New users get **$300 in free credits** valid for 90 days
- No charges during the free trial period

✅ Step 2. **Enable billing on your account** - Required even for free tier

- A credit card is needed for verification
- You won't be charged unless you explicitly upgrade
- This demo stays within the free tier if you clean up resources promptly

✅ Step 3. **Understand the free tier** - Know what's included

- Deploy up to 10 agents at no cost
- Check free tier details
- Review common questions

💡 **Quick Setup Guide:** Watch this 3-minute setup video for a walkthrough

1.2: Import components

Now, import the specific components you'll need for this notebook. This keeps your code organized and ensures we have access to the necessary building blocks.

```
import os
import random
import time
import vertexai
from kaggle_secrets import UserSecretsClient
from vertexai import agent_engines

print("✅ Imports completed successfully")
```

1.3: Add Cloud Credentials to Secrets

1. In the top menu bar of the notebook editor, select `Add-ons` then `Google Cloud SDK`.
2. Click on `Link Account`
3. Select your Google Cloud Account
4. Attach to the notebook

This cell retrieves your Google Cloud credentials from Kaggle Secrets and configures them for use. These credentials allow the notebook to authenticate with Google Cloud services like Vertex AI Agent Engine.

```
# Set up Cloud Credentials in Kaggle
user_secrets = UserSecretsClient()
user_credential = user_secrets.get_gcloud_credential()
user_secrets.set_tensorflow_credential(user_credential)

print("✅ Cloud credentials configured")
```

1.4: Set your PROJECT_ID

Important: Make sure to replace `"your-project-id"` with your actual Google Cloud project ID. You can find your project ID in the Google Cloud Console.

```
## Set your PROJECT_ID
PROJECT_ID = "your-project-id"  # TODO: Replace with your project ID
os.environ["GOOGLE_CLOUD_PROJECT"] = PROJECT_ID

if PROJECT_ID == "your-project-id" or not PROJECT_ID:
    raise ValueError("⚠️ Please replace 'your-project-id' with your actual Google Cloud Project ID.")

print(f"✅ Project ID set to: {PROJECT_ID}")
```

1.5: Enable Google Cloud APIs

For this tutorial, you'll need to enable the following APIs in the Google Cloud Console.

- Vertex AI API
- Cloud Storage API
- Cloud Logging API
- Cloud Monitoring API
- Cloud Trace API
- Telemetry API

You can use this link to open the Google Cloud Console and follow the steps there to enable these APIs.

🏗️ Section 2: Create Your Agent with ADK

Before we deploy, we need a functional agent to host. We'll build a **Weather Assistant** designed to serve as our sample agent.

This agent is optimized for production testing with the following configuration:

- **Model:** Uses gemini-2.5-flash-lite for low latency and cost-efficiency.
- **Tools:** Includes a `get_weather` function to demonstrate tool execution.
- **Persona:** Responds conversationally to prove the instruction-following capabilities.

This demonstrates the foundational ADK architecture we are about to package: **Agent + Tools + Instructions**.

We'll create the following files and directory structure:

```
sample_agent/
├── agent.py                  # The logic
├── requirements.txt          # The libraries
├── .env                      # The secrets/config
└── .agent_engine_config.json # The hardware specs
```

2.1: Create agent directory

We need a clean workspace to package our agent for deployment. We will create a directory named `sample_agent`.

All necessary files - including the agent code, dependencies, and configuration—will be written into this folder to prepare it for the `adk deploy` command.

```
## Create simple agent - all code for the agent will live in this directory
!mkdir -p sample_agent

print(f"✅ Sample Agent directory created")
```

2.2: Create requirements file

The Agent Engine builds a dedicated environment for your agent. To ensure it runs correctly, we must declare our dependencies.

We will write a `requirements.txt` file containing the Python packages needed for the agent.

```
%%writefile sample_agent/requirements.txt

google-adk
opentelemetry-instrumentation-google-genai
```

2.3: Create environment configuration

We need to provide the agent with the necessary cloud configuration settings.

We will write a `.env` file that sets the cloud location to `global` and explicitly enables the Vertex AI backend for the ADK SDK.

```
%%writefile sample_agent/.env

# https://cloud.google.com/vertex-ai/generative-ai/docs/learn/locations#global-endpoint
```

```
GOOGLE_CLOUD_LOCATION="global"

# Set to 1 to use Vertex AI, or 0 to use Google AI Studio
GOOGLE_GENAI_USE_VERTEXAI=1
```

Configuration explained:

- `GOOGLE_CLOUD_LOCATION="global"` - Uses the `global` endpoint for Gemini API calls
- `GOOGLE_GENAI_USE_VERTEXAI=1` - Configures ADK to use Vertex AI instead of Google AI Studio

2.4: Create agent code

We will now generate the `agent.py` file. This script defines the behavior of our **Weather Assistant**.

Agent Configuration:

- 🧠 Model: Uses `gemini-2.5-flash-lite` for low latency and cost-efficiency.
- 🛠️ Tools: Accesses a `get_weather` function to retrieve data.
- 📝 Instructions: Follows a system prompt to identify cities and respond in a friendly tone.

```
%%writefile sample_agent/agent.py
from google.adk.agents import Agent
import vertexai
import os

vertexai.init(
    project=os.environ["GOOGLE_CLOUD_PROJECT"],
    location=os.environ["GOOGLE_CLOUD_LOCATION"],
)

def get_weather(city: str) -> dict:
    """
    Returns weather information for a given city.

    This is a TOOL that the agent can call when users ask about weather.
    In production, this would call a real weather API (e.g., OpenWeatherMap).
    For this demo, we use mock data.

    Args:
        city: Name of the city (e.g., "Tokyo", "New York")

    Returns:
        dict: Dictionary with status and weather report or error message
        """
        # Mock weather database with structured responses
        weather_data = {
            "san francisco": {"status": "success", "report": "The weather in San Francisco is sunny with a temperature of 72°F (22°C)."},
            "new york": {"status": "success", "report": "The weather in New York is cloudy with a temperature of 65°F (18°C)."},
            "london": {"status": "success", "report": "The weather in London is rainy with a temperature of 58°F (14°C)."},
            "tokyo": {"status": "success", "report": "The weather in Tokyo is clear with a temperature of 70°F (21°C)."},
            "paris": {"status": "success", "report": "The weather in Paris is partly cloudy with a temperature of 68°F (20°C)."}
        }

        city_lower = city.lower()
        if city_lower in weather_data:
            return weather_data[city_lower]
        else:
            available_cities = ", ".join([c.title() for c in weather_data.keys()])
            return {
                "status": "error",
                "error_message": f"Weather information for '{city}' is not available. Try: {available_cities}"
            }

    root_agent = Agent(
        name="weather_assistant",
        model="gemini-2.5-flash-lite",  # Fast, cost-effective Gemini model
        description="A helpful weather assistant that provides weather information for cities.",
        instruction="""
        You are a friendly weather assistant. When users ask about the weather:

        1. Identify the city name from their question
        2. Use the get_weather tool to fetch current weather information
        3. Respond in a friendly, conversational tone
        4. If the city isn't available, suggest one of the available cities

        Be helpful and concise in your responses.
        """,
        tools=[get_weather]
    )
```

Section 3: Deploy to Agent Engine

ADK supports multiple deployment platforms. Learn more in the ADK deployment documentation.

You'll be deploying to Vertex AI Agent Engine in this notebook.

Vertex AI Agent Engine

- **Fully managed** service specifically for AI agents
- **Auto-scaling** with session management built-in
- **Easy deployment** using Agent Starter Pack
- Deploy to Agent Engine Guide

Note: To help you get started with the runtime, Agent Engine offers a monthly free tier, which you can learn more about in the documentation. The agent deployed in this notebook should stay within the free tier if cleaned up promptly. Note that you can incur costs if the agent is left running.

Other Deployment Options

Cloud Run

- Serverless, easiest to start
- Perfect for demos and small-to-medium workloads
- Deploy to Cloud Run Guide

Google Kubernetes Engine (GKE)

- Full control over containerized deployments
- Best for complex multi-agent systems
- Deploy to GKE Guide

3.1: Create deployment configuration

The `.agent_engine_config.json` file controls the deployment settings.

```
%%writefile sample_agent/.agent_engine_config.json
{
    "min_instances": 0,
    "max_instances": 1,
    "resource_limits": {"cpu": "1", "memory": "1Gi"}
}
```

Configuration explained:

- `"min_instances": 0` - Scales down to zero when not in use (saves costs)
- `"max_instances": 1` - Maximum of 1 instance running (sufficient for this demo)
- `"cpu": "1"` - 1 CPU core per instance

- `"memory": "1Gi"` - 1 GB of memory per instance

These settings keep costs minimal while providing adequate resources for our weather agent.

3.2: Select deployment region

Agent Engine is available in specific regions. We'll randomly select one for this demo.

```
regions_list = ["europe-west1", "europe-west4", "us-east4", "us-west1"]
deployed_region = random.choice(regions_list)

print(f"✅ Selected deployment region: {deployed_region}")
```

About regions:

Agent Engine is available in multiple regions. For production:

- Choose a region close to your users for lower latency
- Consider data residency requirements
- Check the [Agent Engine locations documentation]

3.3: Deploy the agent

This uses the ADK CLI to deploy your agent to Agent Engine.

```
!adk deploy agent_engine --project=$PROJECT_ID --region=$deployed_region sample_agent --agent_engine_config_file=sample_agent/.agent_e
```

What just happened:

The `adk deploy agent_engine` command:

1. Packages your agent code (`sample_agent/` directory)
2. Uploads it to Agent Engine
3. Creates a containerized deployment
4. Outputs a resource name like: `projects/PROJECT_NUMBER/locations/REGION/reasoningEngines/ID`

Note: Deployment typically takes 2-5 minutes.

Section 4: Retrieve and Test Your Deployed Agent

4.1: Retrieve the deployed agent

After deploying with the CLI, we need to retrieve the agent object to interact with it.

```
# Initialize Vertex AI
vertexai.init(project=PROJECT_ID, location=deployed_region)

# Get the most recently deployed agent
agents_list = list(agent_engines.list())
if agents_list:
    remote_agent = agents_list[0]  # Get the first (most recent) agent
    client = agent_engines
    print(f"✅ Connected to deployed agent: {remote_agent.resource_name}")
else:
    print("❌ No agents found. Please deploy first.")
```

What happened:

This cell retrieves your deployed agent:

1. Initializes the Vertex AI SDK with your project and region
2. Lists all deployed agents in that region
3. Gets the first one (most recently deployed)
4. Stores it as `remote_agent` for testing

4.2: Test the deployed agent

Now let's send a query to your deployed agent!

```
async for item in remote_agent.async_stream_query(
    message="What is the weather in Tokyo?",
    user_id="user_42",
):
    print(item)
```

What happened:

This cell tests your deployed agent:

1. Sends the query "What is the weather in Tokyo?"
2. Streams the response from the agent

Understanding the output:

You'll see multiple items printed:

1. **Function call** - Agent decides to call `get_weather` tool
2. **Function response** - Result from the tool (weather data)
3. **Final response** - Agent's natural language answer

Section 5: Long-Term Memory with Vertex AI Memory Bank

What Problem Does Memory Bank Solve?

Your deployed agent has **session memory** - it remembers the conversation while you're chatting. But once the session ends, it forgets everything. Each new conversation starts from scratch.

The problem:

- User tells agent "I prefer Celsius" today
- Tomorrow, user asks about weather → Agent gives Fahrenheit (forgot preference)
- User has to repeat preferences every time

What is Vertex AI Memory Bank?

Memory Bank gives your agent **long-term memory across sessions**:

Session Memory	Memory Bank
Single conversation	All conversations
Forgets when session ends	Remembers permanently
"What did I just say?"	"What's my favorite city?"

How it works:

1. **During conversations** - Agent uses memory tools to search past facts
2. **After conversations** - Agent Engine extracts key information ("User prefers Celsius")
3. **Next session** - Agent automatically recalls and uses that information

Example:

- **Session 1:** User: "I prefer Celsius"
- **Session 2 (days later):** User: "Weather in Tokyo?" → Agent responds in Celsius automatically ✨

🔧 Memory Bank & Your Deployment

Your Agent Engine deployment **provides the infrastructure** for Memory Bank, but it's not enabled by default.

To use Memory Bank:

1. Add memory tools to your agent code (`PreloadMemoryTool`)
2. Add a callback to save conversations to Memory Bank
3. Redeploy your agent

Once configured, Memory Bank works automatically - no additional infrastructure needed!

📚 Learn More

- **ADK Memory Guide** - Complete guide with code examples
- **Memory Tools** - PreloadMemory and LoadMemory documentation
- **Get started with Memory Bank on ADK** - Sample notebook that demonstrates how to build ADK agents with memory

---

🧹 Section 6: Cleanup

⚠️ **IMPORTANT: Prevent unexpected charges: Always delete resources when done testing!**

Cost Reminders

As a reminder, leaving the agent running can incur costs. Agent Engine offers a monthly free tier, which you can learn more about in the documentation.

Always delete resources when done testing!

When you're done testing and querying your deployed agent, it's recommended to delete your remote agent to avoid incurring additional costs:

```
agent_engines.delete(resource_name=remote_agent.resource_name, force=True)

print("✅ Agent successfully deleted")
```

What happened:

This cell deletes your deployed agent:

- `resource_name=remote_agent.resource_name` - Identifies which agent to delete
- `force=True` - Forces deletion even if the agent is running

The deletion process typically takes 1-2 minutes. You can verify deletion in the Agent Engine Console.

✅ Congratulations! You're Ready for Production Deployment

You've successfully learned how to deploy ADK agents to Vertex AI Agent Engine - taking your agents from development to production!

You now know how to deploy agents with enterprise-grade infrastructure, manage costs, and test production deployments.

📚 Learn More

Refer to the following documentation to learn more:

- ADK Documentation
- Agent Engine Documentation
- ADK Deployment Guide

Other Deployment Options:

- Cloud Run Deployment
- GKE Deployment

Production Best Practices:

- Delete test deployments when finished to avoid costs
- Enable tracing (`enable_tracing=True`) for debugging
- Monitor via Vertex AI Console
- Follow security best practices

🎯 Course Recap: Your 5-Day Journey

Over the past 5 days, you've learned:

- **Day 1:** Agent fundamentals - Building your first agent with tools and instructions
- **Day 2:** Advanced tools - Custom tools, built-in tools, and best practices
- **Day 3:** Sessions & Memory - Managing conversations and long-term knowledge storage
- **Day 4:** Observability & Evaluation - Monitoring agents and measuring performance
- **Day 5:** Production Deployment - Taking your agents live with Agent Engine

You now have the complete toolkit to build, test, and deploy production-ready AI agents!

🚀 What's Next?

Thank you for completing the 5-day AI Agents course!

Now it's your turn to build:

- Start creating your own AI agents with ADK
- Share your projects with the community on Kaggle Discord
- Explore advanced patterns in the ADK documentation

Happy building! 🚀

Authors
Lavi Nigam