Cloud Run - Agent Development Kit

Source URL: https://google.github.io/adk-docs/deploy/cloud-run/

Deploy to Cloud Run¶

Cloud Run is a fully managed platform that enables you to run your code directly on top of Google's scalable infrastructure.

To deploy your agent, you can use either the `adk deploy cloud_run` command *(recommended for Python)*, or with `gcloud run deploy` command through Cloud Run.

Agent sample¶

For each of the commands, we will reference a the `Capital Agent` sample defined on the LLM agent page. We will assume it's in a directory (eg: `capital_agent`).

To proceed, confirm that your agent code is configured as follows:

PythonJava

1. Agent code is in a file called `agent.py` within your agent directory.
2. Your agent variable is named `root_agent`.
3. `__init__.py` is within your agent directory and contains `from . import agent`.
4. Agent code is in a file called `CapitalAgent.java` within your agent directory.
5. Your agent variable is global and follows the format `public static BaseAgent ROOT_AGENT`.
6. Your agent definition is present in a static class method.

Refer to the following section for more details. You can also find a sample app in the Github repo.

Environment variables¶

Set your environment variables as described in the [Setup and Installation](#) guide.

```
export GOOGLE_CLOUD_PROJECT=your-project-id
export GOOGLE_CLOUD_LOCATION=us-central1 # Or your preferred location
export GOOGLE_GENAI_USE_VERTEXAI=True
```

(Replace `your-project-id` with your actual GCP project ID)

Deployment commands¶

Python - adk CLIPython - gcloud CLIJava - gcloud CLI

adk CLI¶

The `adk deploy cloud_run` command deploys your agent code to Google Cloud Run.

Ensure you have authenticated with Google Cloud (`gcloud auth login` and `gcloud config set project <your-project-id>`).

Setup environment variables¶

Optional but recommended: Setting environment variables can make the deployment commands cleaner.

```
# Set your Google Cloud Project ID
export GOOGLE_CLOUD_PROJECT="your-gcp-project-id"

# Set your desired Google Cloud Location
export GOOGLE_CLOUD_LOCATION="us-central1" # Example location

# Set the path to your agent code directory
export AGENT_PATH="./capital_agent" # Assuming capital_agent is in the
```

```
# Set a name for your Cloud Run service (optional)
export SERVICE_NAME="capital-agent-service"

# Set an application name (optional)
export APP_NAME="capital-agent-app"
```

Command usage¶

Minimal command¶

```
 adk deploy cloud_run \
--project=$GOOGLE_CLOUD_PROJECT \
--region=$GOOGLE_CLOUD_LOCATION \
$AGENT_PATH
```

Full command with optional flags¶

```
 adk deploy cloud_run \
--project=$GOOGLE_CLOUD_PROJECT \
--region=$GOOGLE_CLOUD_LOCATION \
--service_name=$SERVICE_NAME \
--app_name=$APP_NAME \
--with_ui \
$AGENT_PATH
```

Arguments¶

- `AGENT_PATH`: (Required) Positional argument specifying the path to the directory containing your agent's source code (e.g., `$AGENT_PATH` in the examples, or `capital_agent/`). This directory must contain at least an `__init__.py` and your main agent file (e.g., `agent.py`).

Options¶

- `--project TEXT`: (Required) Your Google Cloud project ID (e.g., `$GOOGLE_CLOUD_PROJECT`).
- `--region TEXT`: (Required) The Google Cloud location for deployment (e.g., `$GOOGLE_CLOUD_LOCATION`, `us-central1`).
- `--service_name TEXT`: (Optional) The name for the Cloud Run service (e.g., `$SERVICE_NAME`). Defaults to `adk-default-service-name`.
- `--app_name TEXT`: (Optional) The application name for the ADK API server (e.g., `$APP_NAME`). Defaults to the name of the directory specified by `AGENT_PATH` (e.g., `capital_agent` if `AGENT_PATH` is `./capital_agent`).
- `--agent_engine_id TEXT`: (Optional) If you are using a managed session service via Vertex AI Agent Engine, provide its resource ID here.
- `--port INTEGER`: (Optional) The port number the ADK API server will listen on within the container. Defaults to 8000.
- `--with_ui`: (Optional) If included, deploys the ADK dev UI alongside the agent API server. By default, only the API server is deployed.
- `--temp_folder TEXT`: (Optional) Specifies a directory for storing intermediate files generated during the deployment process. Defaults to a timestamped folder in the system's temporary directory. *(Note: This option is generally not needed unless troubleshooting issues).*
- `--help`: Show the help message and exit.

Authenticated access¶

During the deployment process, you might be prompted: `Allow unauthenticated invocations to [your-service-name] (y/N)?`.

- Enter `y` to allow public access to your agent's API endpoint without authentication.
- Enter `N` (or press Enter for the default) to require authentication (e.g., using an identity token as shown in the "Testing your agent" section).

Upon successful execution, the command will deploy your agent to Cloud Run and provide the URL of the deployed service.

gcloud CLI¶

Alternatively, you can deploy using the standard `gcloud run deploy` command with a `Dockerfile`. This method requires more manual setup compared to the `adk` command but offers flexibility, particularly if you want to embed your agent within a custom FastAPI application.

Ensure you have authenticated with Google Cloud (`gcloud auth login` and `gcloud config set project <your-project-id>`).

Project Structure¶

Organize your project files as follows:

```
your-project-directory/
├── capital_agent/
│   ├── __init__.py
│   └── agent.py       # Your agent code (see "Agent sample" tab)
├── main.py            # FastAPI application entry point
├── requirements.txt   # Python dependencies
└── Dockerfile         # Container build instructions
```

Create the following files (`main.py`, `requirements.txt`, `Dockerfile`) in the root of `your-project-directory/`.

Code files¶

1. This file sets up the FastAPI application using `get_fast_api_app()` from ADK:

main.py

``` import os

import uvicorn from google.adk.cli.fast_api import get_fast_api_app

# Get the directory where main.py is located AGENT_DIR = os.path.dirname(os.path.abspath(**file**)) # Example session DB URL (e.g., SQLite) SESSION_DB_URL = "sqlite:///./sessions.db" # Example allowed
```

origins for CORS ALLOWED_ORIGINS = ["http://localhost", "http://localhost:8080", "*"] # Set web=True if you intend to serve a web interface, False otherwise SERVE_WEB_INTERFACE = True

# Call the function to get the FastAPI app instance # Ensure the agent directory name ('capital_agent') matches your agent folder app = get_fast_api_app(agents_dir=AGENT_DIR, session_db_url=SESSION_DB_URL, allow_origins=ALLOWED_ORIGINS, web=SERVE_WEB_INTERFACE,)

# You can add more FastAPI routes or configurations below if needed # Example: # @app.get("/hello") # async def read_root(): # return {"Hello": "World"}

if **name** == "**main**": # Use the PORT environment variable provided by Cloud Run, defaulting to 8080 uvicorn.run(app, host="0.0.0.0", port=int(os.environ.get("PORT", 8080)))

~~~

Note: We specify `agent_dir` to the directory `main.py` is in and use `os.environ.get("PORT", 8080)` for Cloud Run compatibility. 2. List the necessary Python packages:

requirements.txt

``` google_adk # Add any other dependencies your agent needs

``` 3. Define the container image:

Dockerfile

``` FROM python:3.13-slim WORKDIR /app

COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt

RUN adduser --disabled-password --gecos "" myuser && \ chown -R myuser:myuser /app

COPY . .

USER myuser
~~~

ENV PATH="/home/myuser/.local/bin:$PATH"

CMD ["sh", "-c", "uvicorn main:app --host 0.0.0.0 --port $PORT"]

```

Defining Multiple Agents¶

You can define and deploy multiple agents within the same Cloud Run instance by creating separate folders in the root of `your-project-directory/`. Each folder represents one agent and must define a `root_agent` in its configuration.

Example structure:

~~~
your-project-directory/
├── capital_agent/
│   ├── __init__.py
│   └── agent.py       # contains `root_agent` definition
├── population_agent/
│   ├── __init__.py
│   └── agent.py       # contains `root_agent` definition
└── ...
~~~

Deploy using `gcloud`¶

Navigate to `your-project-directory` in your terminal.

~~~
gcloud run deploy capital-agent-service \
--source . \
--region $GOOGLE_CLOUD_LOCATION \
--project $GOOGLE_CLOUD_PROJECT \
--allow-unauthenticated \
--set-env-vars="GOOGLE_CLOUD_PROJECT=$GOOGLE_CLOUD_PROJECT,GOOGLE_CLOU
~~~
```

```
# Add any other necessary environment variables your agent might need
```

- `capital-agent-service`: The name you want to give your Cloud Run service.
- `--source .`: Tells gcloud to build the container image from the Dockerfile in the current directory.
- `--region`: Specifies the deployment region.
- `--project`: Specifies the GCP project.
- `--allow-unauthenticated`: Allows public access to the service. Remove this flag for private services.
- `--set-env-vars`: Passes necessary environment variables to the running container. Ensure you include all variables required by ADK and your agent (like API keys if not using Application Default Credentials).

`gcloud` will build the Docker image, push it to Google Artifact Registry, and deploy it to Cloud Run. Upon completion, it will output the URL of your deployed service.

For a full list of deployment options, see the `gcloud run deploy` reference documentation.

gcloud CLI¶

You can deploy Java Agents using the standard `gcloud run deploy` command and a `Dockerfile`. This is the current recommended way to deploy Java Agents to Google Cloud Run.

Ensure you are authenticated with Google Cloud. Specifically, run the commands `gcloud auth login` and `gcloud config set project <your-project-id>` from your terminal.

Project Structure¶

Organize your project files as follows:

```
your-project-directory/
├── src/
│   └── main/
```

```
|       └── java/
|           └── agents/
|               ├── capitalagent/
|                   └── CapitalAgent.java    # Your agent code
├── pom.xml                                  # Java adk and adk-dev
└── Dockerfile                               # Container build instr
```

Create the `pom.xml` and `Dockerfile` in the root of your project directory. Your Agent code file (`CapitalAgent.java`) inside a directory as shown above.

Code files¶

1. This is our Agent definition. This is the same code as present in [LLM agent](#) with two caveats:

2. The Agent is now initialized as a **global public static variable**.

3. The definition of the agent can be exposed in a static method or inlined during declaration.

CapitalAgent.java

~~~

``` 2. Add the following dependencies and plugin to the pom.xml file.

pom.xml

``` com.google.adk google-adk 0.1.0 com.google.adk google-adk-dev 0.1.0

org.codehaus.mojo exec-maven-plugin 3.2.0 com.google.adk.web.AdkWebServer compile

``` 3. Define the container image:

Dockerfile

``` # Use an official Maven image with a JDK. Choose a version appropriate for your project. FROM maven:3.8-openjdk-17 AS builder
~~~

WORKDIR /app

COPY pom.xml . RUN mvn dependency:go-offline -B

COPY src ./src

# Expose the port your application will listen on. # Cloud Run will set the PORT environment variable, which your app should use. EXPOSE 8080

# The command to run your application. # TODO(Developer): Update the "adk.agents.source-dir" to the directory that contains your agents. # You can have multiple agents in this directory and all of them will be available in the Dev UI. ENTRYPOINT ["mvn", "exec:java", \ "-Dexec.mainClass=com.google.adk.web.AdkWebServer", \ "-Dexec.classpathScope=compile", \ "-Dexec.args=--server.port=${PORT} --adk.agents.source-dir=src/main/java" \ ]

```

### Deploy using `gcloud`¶

Navigate to `your-project-directory` in your terminal.

```
gcloud run deploy capital-agent-service \
--source . \
--region $GOOGLE_CLOUD_LOCATION \
--project $GOOGLE_CLOUD_PROJECT \
--allow-unauthenticated \
--set-env-vars="GOOGLE_CLOUD_PROJECT=$GOOGLE_CLOUD_PROJECT,GOOGLE_CLOU
# Add any other necessary environment variables your agent might need
```

- `capital-agent-service`: The name you want to give your Cloud Run service.
- `--source .`: Tells gcloud to build the container image from the Dockerfile in the current directory.
- `--region`: Specifies the deployment region.
- `--project`: Specifies the GCP project.
```

- `--allow-unauthenticated`: Allows public access to the service. Remove this flag for private services.
- `--set-env-vars`: Passes necessary environment variables to the running container. Ensure you include all variables required by ADK and your agent (like API keys if not using Application Default Credentials).

`gcloud` will build the Docker image, push it to Google Artifact Registry, and deploy it to Cloud Run. Upon completion, it will output the URL of your deployed service.

For a full list of deployment options, see the `gcloud run deploy` reference documentation.

## Testing your agent¶

Once your agent is deployed to Cloud Run, you can interact with it via the deployed UI (if enabled) or directly with its API endpoints using tools like `curl`. You'll need the service URL provided after deployment.

UI TestingAPI Testing (curl)

### UI Testing¶

If you deployed your agent with the UI enabled:

- **adk CLI:** You included the `--with_ui` flag during deployment.
- **gcloud CLI:** You set `SERVE_WEB_INTERFACE = True` in your `main.py`.

You can test your agent by simply navigating to the Cloud Run service URL provided after deployment in your web browser.

```
 # Example URL format
# https://your-service-name-abc123xyz.a.run.app
```

The ADK dev UI allows you to interact with your agent, manage sessions, and view execution details directly in the browser.

To verify your agent is working as intended, you can:

1. Select your agent from the dropdown menu.
2. Type a message and verify that you receive an expected response from your agent.

If you experience any unexpected behavior, check the Cloud Run console logs.

API Testing (curl)¶

You can interact with the agent's API endpoints using tools like `curl`. This is useful for programmatic interaction or if you deployed without the UI.

You'll need the service URL provided after deployment and potentially an identity token for authentication if your service isn't set to allow unauthenticated access.

Set the application URL¶

Replace the example URL with the actual URL of your deployed Cloud Run service.

```
export APP_URL="YOUR_CLOUD_RUN_SERVICE_URL"
# Example: export APP_URL="https://adk-default-service-name-abc123xyz.
```

Get an identity token (if needed)¶

If your service requires authentication (i.e., you didn't use `--allow-unauthenticated` with `gcloud` or answered 'N' to the prompt with `adk`), obtain an identity token.

```
export TOKEN=$(gcloud auth print-identity-token)
```

If your service allows unauthenticated access, you can omit the `-H "Authorization: Bearer $TOKEN"` header from the `curl` commands below.

List available apps¶

Verify the deployed application name.

```
curl -X GET -H "Authorization: Bearer $TOKEN" $APP_URL/list-apps
```

(Adjust the `app_name` in the following commands based on this output if needed. The default is often the agent directory name, e.g., `capital_agent`).

Create or Update a Session¶

Initialize or update the state for a specific user and session. Replace `capital_agent` with your actual app name if different. The values `user_123` and `session_abc` are example identifiers; you can replace them with your desired user and session IDs.

```
curl -X POST -H "Authorization: Bearer $TOKEN" \
    $APP_URL/apps/capital_agent/users/user_123/sessions/session_abc \
    -H "Content-Type: application/json" \
    -d '{"state": {"preferred_language": "English", "visit_count": 5}}
```

Run the Agent¶

Send a prompt to your agent. Replace `capital_agent` with your app name and adjust the user/session IDs and prompt as needed.

```
curl -X POST -H "Authorization: Bearer $TOKEN" \
    $APP_URL/run_sse \
    -H "Content-Type: application/json" \
    -d '{
    "app_name": "capital_agent",
    "user_id": "user_123",
    "session_id": "session_abc",
    "new_message": {
        "role": "user",
        "parts": [{
        "text": "What is the capital of Canada?"
        }]
    },
    "streaming": false
    }'
```

- Set `"streaming": true` if you want to receive Server-Sent Events (SSE).
- The response will contain the agent's execution events, including the final answer.