GKE - Agent Development Kit

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

Deploy to GKE¶

GKE is Google Clouds managed Kubernetes service. It allows you to deploy and manage containerized applications using Kubernetes.

To deploy your agent you will need to have a Kubernetes cluster running on GKE. You can create a cluster using the Google Cloud Console or the `gcloud` command line tool.

In this example we will deploy a simple agent to GKE. The agent will be a FastAPI application that uses `Gemini 2.0 Flash` as the LLM. We can use Vertex AI or AI Studio as the LLM provider using a Environment variable.

Agent sample¶

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

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

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`.

Environment variables¶

Set your environment variables as described in the Setup and Installation guide. You also need to install the `kubectl` command line tool. You can find instructions to do so in the Google Kubernetes Engine Documentation.

```
export GOOGLE_CLOUD_PROJECT=your-project-id # Your GCP project ID
export GOOGLE_CLOUD_LOCATION=us-central1 # Or your preferred location
export GOOGLE_GENAI_USE_VERTEXAI=true # Set to true if using Vertex AI
export GOOGLE_CLOUD_PROJECT_NUMBER=$(gcloud projects describe --format
```

If you don't have `jq` installed, you can use the following command to get the project number:

```
gcloud projects describe $GOOGLE_CLOUD_PROJECT
```

And copy the project number from the output.

```
export GOOGLE_CLOUD_PROJECT_NUMBER=YOUR_PROJECT_NUMBER
```

Deployment commands¶

gcloud CLI¶

You can deploy your agent to GKE using the `gcloud` and `kubectl` cli and Kubernetes manifest files.

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

Enable APIs¶

Enable the necessary APIs for your project. You can do this using the `gcloud` command line tool.

```
gcloud services enable \
    container.googleapis.com \
    artifactregistry.googleapis.com \
    cloudbuild.googleapis.com \
    aiplatform.googleapis.com
```

Create a GKE cluster¶

You can create a GKE cluster using the `gcloud` command line tool. This example creates an Autopilot cluster named `adk-cluster` in the `us-central1` region.

> *If creating a GKE Standard cluster, make sure Workload Identity is enabled. Workload Identity is enabled by default in an AutoPilot cluster.*

```
gcloud container clusters create-auto adk-cluster \
    --location=$GOOGLE_CLOUD_LOCATION \
    --project=$GOOGLE_CLOUD_PROJECT
```

After creating the cluster, you need to connect to it using `kubectl`. This command configures `kubectl` to use the credentials for your new cluster.

```
gcloud container clusters get-credentials adk-cluster \
    --location=$GOOGLE_CLOUD_LOCATION \
    --project=$GOOGLE_CLOUD_PROJECT
```

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 fastapi import FastAPI 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: FastAPI = 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"]

```

Build the container image¶

You need to create a Google Artifact Registry repository to store your container images. You can do this using the `gcloud` command line tool.

```
gcloud artifacts repositories create adk-repo \
    --repository-format=docker \
    --location=$GOOGLE_CLOUD_LOCATION \
    --description="ADK repository"
```

Build the container image using the `gcloud` command line tool. This example builds the image and tags it as `adk-repo/adk-agent:latest`.

```
gcloud builds submit \
    --tag $GOOGLE_CLOUD_LOCATION-docker.pkg.dev/$GOOGLE_CLOUD_PROJECT/
    --project=$GOOGLE_CLOUD_PROJECT \
    .
```

Verify the image is built and pushed to the Artifact Registry:

```
gcloud artifacts docker images list \
  $GOOGLE_CLOUD_LOCATION-docker.pkg.dev/$GOOGLE_CLOUD_PROJECT/adk-repo
  --project=$GOOGLE_CLOUD_PROJECT
```

Configure Kubernetes Service Account for Vertex AI¶

If your agent uses Vertex AI, you need to create a Kubernetes service account with the necessary permissions. This example creates a service account named `adk-agent-sa` and binds it to the `Vertex AI User` role.

> *If you are using AI Studio and accessing the model with an API key you can skip this step.*

```
kubectl create serviceaccount adk-agent-sa
```

```
gcloud projects add-iam-policy-binding projects/${GOOGLE_CLOUD_PROJECT
    --role=roles/aiplatform.user \
    --member=principal://iam.googleapis.com/projects/${GOOGLE_CLOUD_PR
    --condition=None
```

Create the Kubernetes manifest files¶

Create a Kubernetes deployment manifest file named `deployment.yaml` in your project directory. This file defines how to deploy your application on GKE.

deployment.yaml

```
 cat <<  EOF > deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: adk-agent
spec:
  replicas: 1
  selector:
    matchLabels:
      app: adk-agent
  template:
    metadata:
      labels:
        app: adk-agent
    spec:
      serviceAccount: adk-agent-sa
      containers:
      - name: adk-agent
        imagePullPolicy: Always
        image: $GOOGLE_CLOUD_LOCATION-docker.pkg.dev/$GOOGLE_CLOUD_PRO
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
            ephemeral-storage: "128Mi"
          requests:
            memory: "128Mi"
            cpu: "500m"
            ephemeral-storage: "128Mi"
        ports:
        - containerPort: 8080
        env:
          - name: PORT
            value: "8080"
          - name: GOOGLE_CLOUD_PROJECT
```

```
            value: GOOGLE_CLOUD_PROJECT
          - name: GOOGLE_CLOUD_LOCATION
            value: GOOGLE_CLOUD_LOCATION
          - name: GOOGLE_GENAI_USE_VERTEXAI
            value: GOOGLE_GENAI_USE_VERTEXAI
          # If using AI Studio, set GOOGLE_GENAI_USE_VERTEXAI to false
          # - name: GOOGLE_API_KEY
          #   value: GOOGLE_API_KEY
          # Add any other necessary environment variables your agent m
---
apiVersion: v1
kind: Service
metadata:
  name: adk-agent
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: adk-agent
EOF
```

Deploy the Application¶

Deploy the application using the `kubectl` command line tool. This command applies the deployment and service manifest files to your GKE cluster.

```
kubectl apply -f deployment.yaml
```

After a few moments, you can check the status of your deployment using:

```
kubectl get pods -l=app=adk-agent
```

This command lists the pods associated with your deployment. You should see a pod with a status of `Running`.

Once the pod is running, you can check the status of the service using:

```
kubectl get service adk-agent
```

If the output shows a `External IP`, it means your service is accessible from the internet. It may take a few minutes for the external IP to be assigned.

You can get the external IP address of your service using:

```
kubectl get svc adk-agent -o=jsonpath='{.status.loadBalancer.ingress[
```

Testing your agent¶

Once your agent is deployed to GKE, 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:

You can test your agent by simply navigating to the kubernetes service URL in your web browser.

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 pod logs for your agent using:

```
kubectl logs -l app=adk-agent
```

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.

Set the application URL¶

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

```
export APP_URL="KUBERNETES_SERVICE_URL"
```

List available apps¶

Verify the deployed application name.

```
curl -X GET $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 \
    $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 $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.

Troubleshooting¶

These are some common issues you might encounter when deploying your agent to GKE:

403 Permission Denied for `Gemini 2.0 Flash`¶

This usually means that the Kubernetes service account does not have the necessary permission to access the Vertex AI API. Ensure that you have created the service account and bound it to the `Vertex AI User` role as described in the [Configure Kubernetes Service Account for Vertex AI]() section. If you are using AI Studio, ensure that you have set the `GOOGLE_API_KEY` environment variable in the deployment manifest and it is valid.

Attempt to write a readonly database¶

You might see there is no session id created in the UI and the agent does not respond to any messages. This is usually caused by the SQLite database being read-only. This can happen if you run the agent locally and then create the container image which copies the SQLite database into the container. The database is then read-only in the container.

```
sqlalchemy.exc.OperationalError: (sqlite3.OperationalError) attempt to
[SQL: UPDATE app_states SET state=?, update_time=CURRENT_TIMESTAMP WHE
```

To fix this issue, you can either:

Delete the SQLite database file from your local machine before building the container image. This will create a new SQLite database when the container is started.

```
rm -f sessions.db
```

or (recommended) you can add a `.dockerignore` file to your project directory to exclude the SQLite database from being copied into the container image.

.dockerignore

```
sessions.db
```

Build the container image abd deploy the application again.

Cleanup¶

To delete the GKE cluster and all associated resources, run:

```
gcloud container clusters delete adk-cluster \
    --location=$GOOGLE_CLOUD_LOCATION \
    --project=$GOOGLE_CLOUD_PROJECT
```

To delete the Artifact Registry repository, run:

```
gcloud artifacts repositories delete adk-repo \
    --location=$GOOGLE_CLOUD_LOCATION \
    --project=$GOOGLE_CLOUD_PROJECT
```

You can also delete the project if you no longer need it. This will delete all resources associated with the project, including the GKE cluster, Artifact Registry repository, and any other resources you created.

```
gcloud projects delete $GOOGLE_CLOUD_PROJECT
```