Google ADK
Agent Development Kit 2025
Complete Beginner Guide | Agents | Tools | Multi-Agent

■ What is Google ADK?

Google ADK (Agent Development Kit) = Open-source framework by Google for building, deploying, and orchestrating AI agents. Built on Gemini models with enterprise-grade features.

Feature	Description
Multi-Agent Systems	Build hierarchical agent teams that collaborate
Rich Tool Ecosystem	Built-in tools + custom function tools
Memory & State	Session memory, persistent storage, context management
Streaming	Real-time streaming responses
Async Support	Full async/await support for performance
Callbacks	Hooks for logging, monitoring, debugging
Model Agnostic	Works with Gemini, GPT, Claude, Ollama
Vertex AI Ready	Deploy directly to Google Cloud

■■ Installation & Setup

Package	Command	Purpose
Core ADK	pip install google-adk	Basic installation
With Vertex AI	pip install "google-adk[vertexai]"	Cloud deployment
With All Extras	pip install "google-adk[all]"	All features
Dev Version	pip install git+https://github.com/google/adk-python	Latest features

```
# Installation
pip install google-adk

# Set up API key (choose one method)
# Method 1: Environment variable
export GOOGLE_API_KEY='your-api-key'

# Method 2: In code
import os
os.environ['GOOGLE_API_KEY'] = 'your-api-key'

# Method 3: .env file
# Create .env file with: GOOGLE_API_KEY=your-api-key
from dotenv import load_dotenv
load_dotenv()

# Basic imports
from google.adk.agents import Agent
from google.adk.runners import Runner
from google.adk.sessions import InMemorySessionService
```

■ Core Concepts

Concept	Description	Example
Agent	AI entity that performs tasks	Assistant, Coder, Researcher
Tool	Function an agent can call	search_web(), calculate()
Session	Conversation context/history	Chat session with memory
Runner	Executes agent interactions	Handles message flow
Model	LLM powering the agent	gemini-2.0-flash
Instruction	System prompt for agent	You are a helpful assistant
Callback	Hook for events	on_tool_call, on_response
Artifact	Generated files/data	Images, code, documents

■ Your First Agent

```
from google.adk.agents import Agent
from google.adk.runners import Runner
from google.adk.sessions import InMemorySessionService

# 1. Create an Agent
agent = Agent(
name='my_assistant',
model='gemini-2.0-flash-exp', # or gemini-1.5-pro, gemini-1.5-flash
instruction='You are a helpful assistant. Be concise and friendly.',
description='A general-purpose AI assistant',
)

# 2. Create Session Service (manages conversation state)
session_service = InMemorySessionService()

# 3. Create Runner (executes the agent)
runner = Runner(
agent=agent,
app_name='my_app',
session_service=session_service,
)

# 4. Run the agent
async def main():
session = await session_service.create_session(
app_name='my_app',
user_id='user_123'
)

response = await runner.run(
user_id='user_123',
session_id=session.id,
new_message='Hello! What can you do?'
)
print(response.content)

import asyncio
asyncio.run(main())
```

■ Agent Configuration

Parameter	Type	Description

name	str	Unique identifier for the agent
model	str	LLM model to use (gemini-2.0-flash-exp)
instruction	str	System prompt / persona
description	str	What the agent does (for multi-agent)
tools	list	List of tools agent can use
sub_agents	list	Child agents (for hierarchical)
output_key	str	Key for output in state
input_schema	dict	Expected input format
output_schema	dict	Expected output format

Supported Models

Model	Model ID	Best For
Gemini 2.0 Flash	gemini-2.0-flash-exp	Fast, multimodal, agents (recommended)
Gemini 2.5 Pro	gemini-2.5-pro-preview-05-06	Complex reasoning, thinking
Gemini 1.5 Pro	gemini-1.5-pro	Long context (1M tokens)
Gemini 1.5 Flash	gemini-1.5-flash	Fast, cost-effective
GPT-4o (via LiteLLM)	openai/gpt-4o	OpenAI integration
Claude (via LiteLLM)	anthropic/claude-sonnet-4-20250514	Anthropic integration
Ollama (Local)	ollama/llama3.2	Local/private deployment

Tools (Function Calling)

Tools are functions that agents can call to perform actions or get information

Built-in Tools

Tool	Import	Purpose
Google Search	from google.adk.tools import google_search	Web search
Code Execution	from google.adk.tools import code_execution	Run Python code
Vertex AI Search	from google.adk.tools import vertex_ai_search	Enterprise search
Load Web Page	from google.adk.tools import load_web_page	Fetch URL content
Load PDF	from google.adk.tools import load_pdf	Extract PDF text

Creating Custom Tools

```
from google.adk.tools import FunctionTool

# Method 1: Using decorator (Recommended)
from google.adk.tools import tool

@tool
def get_weather(city: str) -> str:
'''Get the current weather for a city.

Args:
city: Name of the city to get weather for

Returns:
Weather information as a string
'''
# Your implementation here
return f'Weather in {city}: 25C, Sunny'

@tool
def calculate(expression: str) -> float:
'''Calculate a mathematical expression.

Args:
expression: Math expression to evaluate

Returns:
Result of the calculation
'''
return eval(expression)

# Add tools to agent
agent = Agent(
name='assistant',
model='gemini-2.0-flash-exp',
instruction='You help with weather and calculations.',
tools=[get_weather, calculate], # Add tools here
)
```

■ Async Tools

```
import aiohttp
from google.adk.tools import tool

@tool
async def fetch_data(url: str) -> str:
'''Fetch data from a URL asynchronously.

Args:
url: The URL to fetch

Returns:
Response content as string
'''
async with aiohttp.ClientSession() as session:
async with session.get(url) as response:
return await response.text()
```

■ Multi-Agent Systems

Pattern	Description	Use Case
Sequential	Agents run one after another	Pipeline processing
Parallel	Agents run simultaneously	Independent tasks
Hierarchical	Parent agent delegates to children	Complex workflows
Router	Dispatcher routes to specialist agents	Customer support triage

Loop	Agents iterate until condition met	Refinement, verification

Hierarchical Agent Example

```
# Create specialist sub-agents
researcher = Agent(
name='researcher',
model='gemini-2.0-flash-exp',
instruction='You research topics and gather information.',
description='Researches and gathers information on topics',
tools=[google_search],
)

writer = Agent(
name='writer',
model='gemini-2.0-flash-exp',
instruction='You write clear, engaging content.',
description='Writes articles and content',
)

editor = Agent(
name='editor',
model='gemini-2.0-flash-exp',
instruction='You edit and improve written content.',
description='Edits and polishes content',
)

# Create parent orchestrator agent
orchestrator = Agent(
name='orchestrator',
model='gemini-2.0-flash-exp',
instruction='''You coordinate a team to create content.
1. Use researcher to gather information
2. Use writer to create content
3. Use editor to polish the final result''',
description='Orchestrates content creation workflow',
sub_agents=[researcher, writer, editor], # Add sub-agents
)
```

Session & Memory

Service	Description	Use Case
InMemorySessionService	Stores sessions in memory	Development, testing
DatabaseSessionService	Persists to database	Production
VertexAISessionService	Google Cloud managed	Enterprise deployment

```
from google.adk.sessions import InMemorySessionService

# Create session service
session_service = InMemorySessionService()

# Create a new session
session = await session_service.create_session(
app_name='my_app',
user_id='user_123',
session_id='optional_custom_id', # Optional
)

# Get existing session
session = await session_service.get_session(
app_name='my_app',
user_id='user_123',
session_id='session_456'
)

# List user sessions
sessions = await session_service.list_sessions(
app_name='my_app',
user_id='user_123'
)

# Delete session
await session_service.delete_session(
app_name='my_app',
user_id='user_123',
session_id='session_456'
)
```

State Management

```
# Access session state in tools
from google.adk.tools import tool
from google.adk.agents import ToolContext

@tool
def save_preference(key: str, value: str, ctx: ToolContext) -> str:
'''Save a user preference to session state.'''
ctx.session.state[key] = value
return f'Saved {key} = {value}'

@tool
def get_preference(key: str, ctx: ToolContext) -> str:
'''Get a user preference from session state.'''
return ctx.session.state.get(key, 'Not found')
```

Streaming Responses

```
# Stream responses in real-time
async def stream_response():
session = await session_service.create_session(
app_name='my_app', user_id='user_123'
)

# Use run_stream for streaming
async for event in runner.run_stream(
user_id='user_123',
session_id=session.id,
new_message='Tell me a story'
):
if event.type == 'content':
print(event.content, end='', flush=True)
elif event.type == 'tool_call':
print(f'\nCalling tool: {event.tool_name}')
elif event.type == 'tool_result':
print(f'Tool result: {event.result}')
elif event.type == 'end':
print('\n--- Done ---')
```

■ Callbacks & Events

```
from google.adk.agents import CallbackHandler

class MyCallbacks(CallbackHandler):
async def on_message_start(self, message):
print(f'User: {message}')

async def on_tool_call_start(self, tool_name, args):
print(f'Calling {tool_name} with {args}')

async def on_tool_call_end(self, tool_name, result):
print(f'{tool_name} returned: {result}')

async def on_response_start(self):
print('Agent is responding...')

async def on_response_end(self, response):
print(f'Agent: {response}')

async def on_error(self, error):
print(f'Error: {error}')

# Use callbacks
runner = Runner(
agent=agent,
app_name='my_app',
session_service=session_service,
callbacks=MyCallbacks(), # Add callbacks
)
```

■ CLI & Development Server

Command	Description
adk init my_agent	Create new agent project
adk run	Run agent in dev mode
adk web	Start web UI for testing
adk deploy	Deploy to Vertex AI
adk eval	Run evaluation tests

```
# Project structure
my_agent/
|-- __init__.py
|-- agent.py # Your agent definition
|-- tools.py # Custom tools
|-- config.yaml # Configuration
|-- requirements.txt

# agent.py
from google.adk.agents import Agent
from .tools import my_tool

root_agent = Agent( # Must be named root_agent
name='my_agent',
model='gemini-2.0-flash-exp',
instruction='Your instructions here',
tools=[my_tool],
)

# Run dev server
# $ adk web
# Opens browser at http://localhost:8000
```

■ Deployment Options

Platform	Method	Best For
Local	adk run / Python script	Development
Vertex AI Agent Engine	adk deploy	Production (managed)
Cloud Run	Docker container	Scalable serverless
GKE	Kubernetes deployment	Enterprise scale
Cloud Functions	Serverless function	Event-driven

```
# Deploy to Vertex AI Agent Engine
# 1. Set up Google Cloud
gcloud auth login
gcloud config set project YOUR_PROJECT_ID

# 2. Deploy
adk deploy --project=YOUR_PROJECT_ID --region=us-central1

# Programmatic deployment
from google.adk.deployment import deploy_agent

deployment = await deploy_agent(
agent=root_agent,
project_id='your-project',
region='us-central1',
display_name='My Production Agent',
)
```

■ Complete Working Example

```
# complete_agent.py - Full working example
import asyncio
from google.adk.agents import Agent
from google.adk.runners import Runner
from google.adk.sessions import InMemorySessionService
from google.adk.tools import tool

# Custom tools
@tool
def add_numbers(a: float, b: float) -> float:
'''Add two numbers together.'''
return a + b

@tool
def get_greeting(name: str) -> str:
'''Generate a personalized greeting.'''
return f'Hello, {name}! Nice to meet you!'

# Create agent
agent = Agent(
name='helpful_assistant',
model='gemini-2.0-flash-exp',
instruction='You are a helpful assistant that can greet users and do math.',
tools=[add_numbers, get_greeting],
)

# Setup and run
async def main():
session_service = InMemorySessionService()
runner = Runner(agent=agent, app_name='demo', session_service=session_service)
session = await session_service.create_session(app_name='demo', user_id='user1')

response = await runner.run(user_id='user1', session_id=session.id,
new_message='Hi! My name is Manoj. What is 42 + 58?')
print(response.content)

asyncio.run(main())
```

■ Best Practices

Category	Best Practice
Instructions	Be specific and clear in agent instructions
Tools	Write detailed docstrings - they help the LLM understand when to use tools
Error Handling	Use try-except in tools and return helpful error messages
Naming	Use descriptive names for agents, tools, and parameters
Sub-Agents	Give each sub-agent a clear, focused responsibility
Testing	Use adk web for interactive testing during development
Sessions	Use persistent session storage in production
Streaming	Use run_stream() for better UX in chat applications
Callbacks	Implement callbacks for logging and monitoring
Security	Never expose API keys; use environment variables
Model Choice	Use gemini-2.0-flash-exp for agents, 1.5-pro for long context

■ Quick Reference

Task	Code
Create Agent	Agent(name="x", model="gemini-2.0-flash-exp", instruction="...")
Add Tools	Agent(..., tools=[tool1, tool2])
Add Sub-Agents	Agent(..., sub_agents=[agent1, agent2])
Create Session	await session_service.create_session(app_name, user_id)
Run Agent	await runner.run(user_id, session_id, new_message)
Stream Response	async for event in runner.run_stream(...):
Create Tool	@tool decorator + docstring
Access State	ctx.session.state[key] in tool function

Created by Manoj | The AI Dude Tamil ■

YouTube: The AI Dude Tamil | Master AI, Automation & Prompt Engineering

■ Build powerful AI agents with Google ADK!