# Google Agent Development Kit (ADK): Deployment and Authentication

## Deployment Options

Once you've built and tested your agent using ADK, the next step is to deploy it so it can be accessed, queried, and used in production or integrated with other applications. Deployment moves your agent from your local development machine to a scalable and reliable environment.

Your ADK agent can be deployed to a range of different environments based on your needs for production readiness or custom flexibility:

### 1. Agent Engine in Vertex AI

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.

**Key features:** - Fully managed service on Google Cloud - Auto-scaling capabilities - Simplified deployment process - Integrated with Google Cloud ecosystem

**Deployment process:** 1. Install the Vertex AI SDK: `python pip install google-cloud-aiplatform[adk,agent_engines]`

1. Initialize and create your agent for deployment: ```python from vertexai import agent_engines

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

1. Test your agent locally before deployment
2. Deploy your agent to Agent Engine
3. Grant the deployed agent necessary permissions
4. Interact with your agent through the deployed endpoints

**Requirements:** - Agent Engine only supports Python version >=3.9 and <=3.12 - Proper authentication setup for Google Cloud

### 2. Cloud Run

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

**Key features:** - Containerized deployment - Scalable infrastructure - Pay-per-use pricing model - Support for various authentication methods

**Deployment process:** You can deploy your agent to Cloud Run using either:

1. The ADK CLI (recommended): `bash adk deploy cloud_run`
2. The gcloud CLI: `bash gcloud run deploy`

**Authentication options during deployment:** 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)

## Authentication Mechanisms

Many tools in ADK need to access protected resources (like user data in Google Calendar, Salesforce records, etc.) and require authentication. ADK provides a comprehensive system to handle various authentication methods securely.

### Core Authentication Concepts

The key components involved in ADK authentication are:

1. **`AuthScheme`** : Defines how an API expects authentication credentials (e.g., as an API Key in a header, an OAuth 2.0 Bearer token). ADK supports the same types of authentication schemes as OpenAPI 3.0, using specific classes like:
2. `APIKey`
3. `HTTPBearer`
4. `OAuth2`
5. `OpenIdConnectWithConfig`

6. **`AuthCredential`** : Holds the initial information needed to start the authentication process (e.g., your application's OAuth Client ID/Secret, an API key value). It includes an `auth_type` specifying the credential type.

The general flow involves providing these details when configuring a tool. ADK then attempts to automatically exchange the initial credential for a usable one (like an access token) before the tool makes an API call. For flows requiring user interaction (like OAuth consent), a specific interactive process involving the Agent Client application is triggered.

## Supported Initial Credential Types

1. **API_KEY**:
2. For simple key/value authentication
3. Usually requires no exchange

4. Example: API keys for services like Google Maps or weather APIs

5. **HTTP**:

6. Can represent Basic Auth (not recommended/supported for exchange)
7. Can represent already obtained Bearer tokens

8. If it's a Bearer token, no exchange is needed

9. **OAUTH2**:

10. For standard OAuth 2.0 flows
11. Requires configuration (client ID, secret, scopes)
12. Often triggers the interactive flow for user consent

13. Common for Google APIs, social media platforms, etc.

14. **OPEN_ID_CONNECT**:

15. For authentication based on OpenID Connect
16. Similar to OAuth2, often requires configuration and user interaction

17. Provides identity verification in addition to authorization

18. **SERVICE_ACCOUNT**:

19. For Google Cloud Service Account credentials
20. Can use JSON key or Application Default Credentials
21. Typically exchanged for a Bearer token
22. Used for server-to-server authentication without user interaction

## Configuring Authentication on Tools

Authentication is set up when defining your tool, with different approaches depending on the tool type:

1. **RestApiTool / OpenAPIToolset**:
2. Pass `auth_scheme` and `auth_credential` during initialization ```python from google.adk.tools.openapi_tool.auth.auth_helpers import token_to_scheme_credential

auth_scheme, auth_credential = token_to_scheme_credential( "apikey", "query", "apikey", YOUR_API_KEY_STRING )

toolset = OpenAPIToolset( spec_path="path/to/spec.json", auth_scheme=auth_scheme, auth_credential=auth_credential ) ```

1. **Google API Toolsets**:
2. Use the toolset's specific configuration method ```python from google.adk.tools.google_api_tool import calendar_tool_set

client_id = "YOUR_GOOGLE_OAUTH_CLIENT_ID.apps.googleusercontent.com" client_secret = "YOUR_GOOGLE_OAUTH_CLIENT_SECRET"

calendar_tools = calendar_tool_set.get_tools() for tool in calendar_tools: tool.configure_auth(client_id=client_id, client_secret=client_secret) ```

1. **APIHubToolset / ApplicationIntegrationToolset**:
2. Pass `auth_scheme` and `auth_credential` during initialization ```python from google.adk.tools.apihub_tool.apihub_toolset import APIHubToolset

auth_scheme, auth_credential = token_to_scheme_credential( "apikey", "query", "apikey", YOUR_API_KEY_STRING )

api_toolset = APIHubToolset( name="sample-api", description="A tool using an API protected by API Key", apihub_resource_name="...", auth_scheme=auth_scheme, auth_credential=auth_credential ) ```

## Handling Interactive Authentication Flows

For authentication methods requiring user interaction (like OAuth consent screens), ADK implements a special flow:

1. The agent execution pauses when authentication is needed
2. A special event with the function call `adk_request_credential` is generated

3. The client application (your UI, CLI, etc.) must detect this event and handle the user interaction
4. After user completes authentication, the client provides the obtained credentials back to ADK
5. The agent execution resumes with the authenticated credentials

This approach allows ADK to handle complex authentication flows while keeping the user experience smooth and secure.

## Authentication for Custom Tools

When building custom tools that require authentication, developers can implement authentication logic directly within the tool function. This approach is useful for tools that need to access protected resources but don't fit into the standard authentication patterns.

The tool function can: 1. Check if valid credentials exist in the session state 2. Request new credentials if needed 3. Use the credentials to access the protected resource 4. Store the credentials in the session state for future use

This flexibility allows developers to implement custom authentication flows while still benefiting from ADK's security and state management capabilities.

## Conclusion

ADK provides a comprehensive set of deployment options and authentication mechanisms to support a wide range of use cases. Whether you're building a simple agent with API key authentication or a complex multi-agent system with OAuth flows, ADK offers the tools and patterns to deploy securely and handle authentication effectively. The framework's design balances security, flexibility, and user experience, making it suitable for both simple prototypes and production-ready applications.