Agent Development Kit 徹底解説！マルチエージェントによるカスタム AI エージェント開発

Google Cloud Next Tokyo

Proprietary

しもじょう ゆうじ

下門 祐二

Google Cloud
カスタマー エンジニア

Agenda

01 AI エージェントの登場と進化

02 Agent Development Kit (ADK) 詳解

03 Vertex AI Agent Engine のご紹介

04 Google Agentspace との統合

05 まとめ

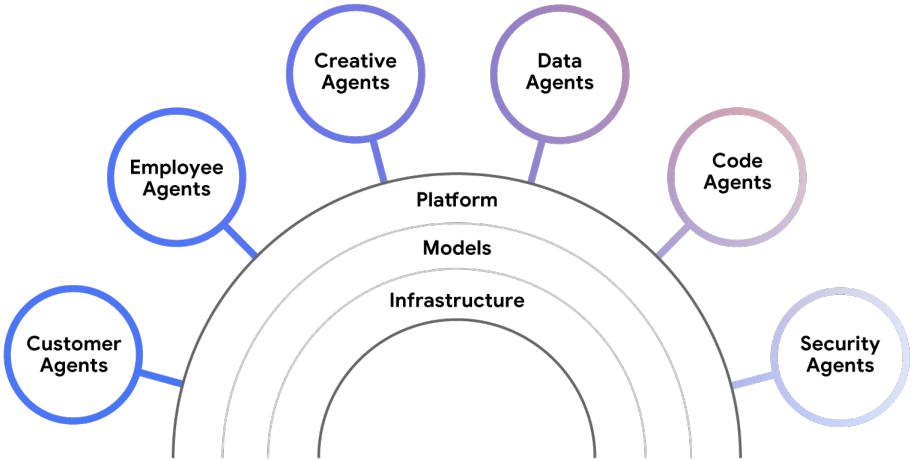
01. AI エージェントの登場と進化

AI エージェントとは？

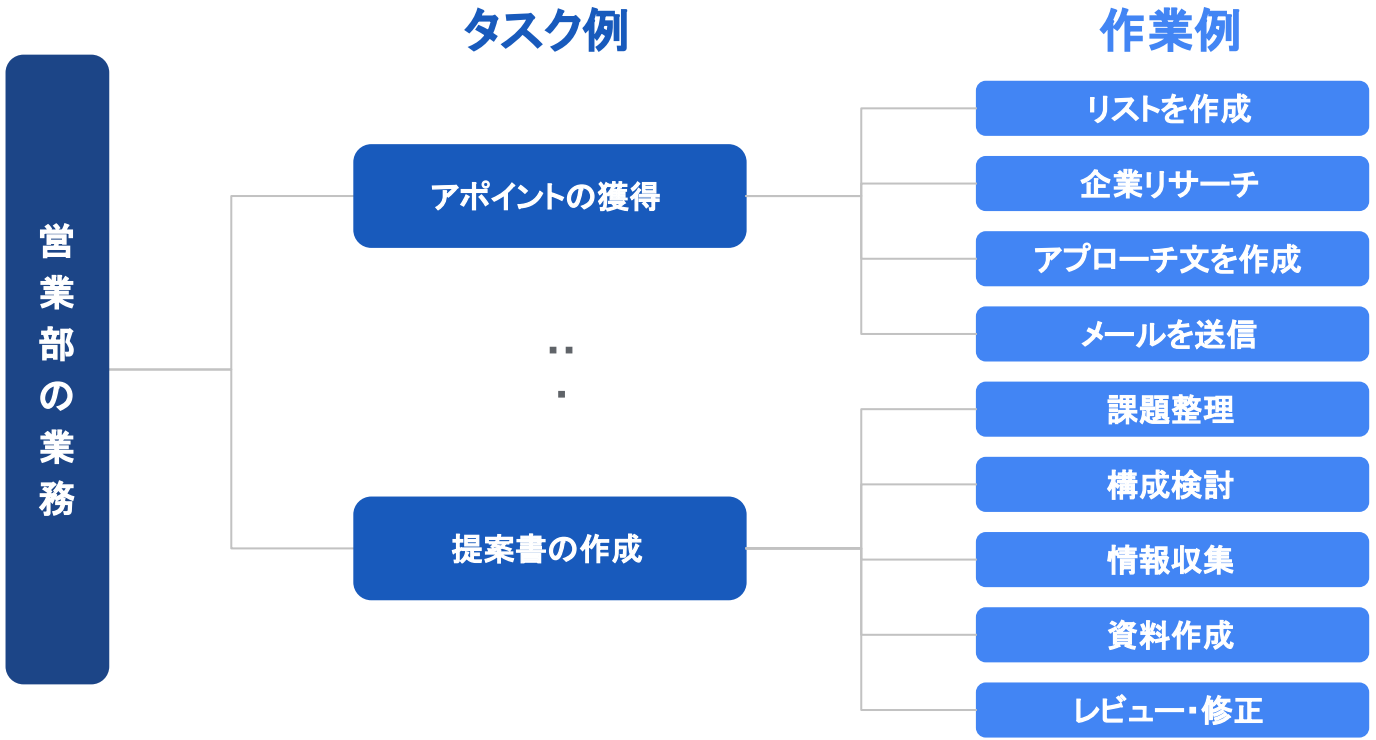
人間の代わりに、特定の**タスク**を一連の**作業**に分解し、

各作業（**思考** + **アクション**）を、利用可能な**ツール**を活用して

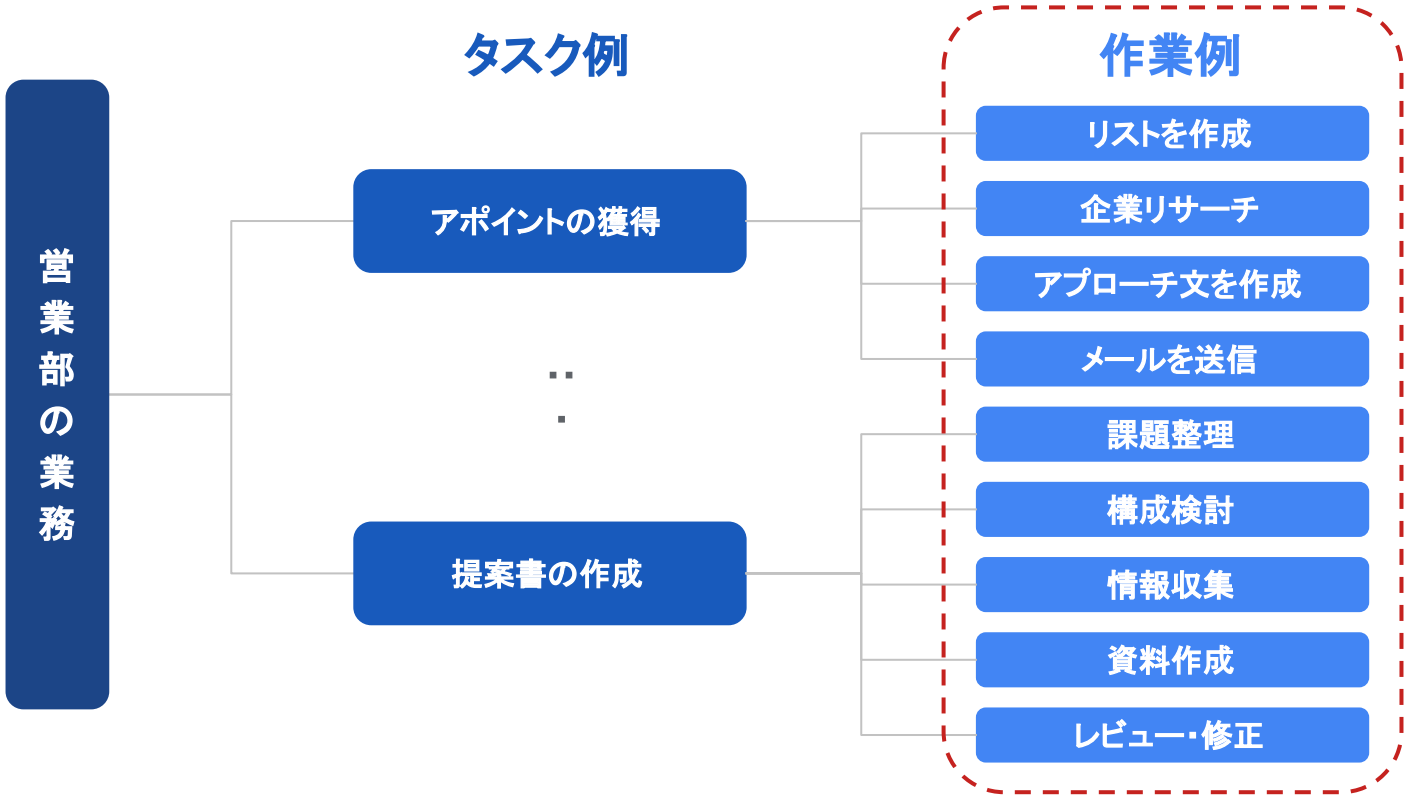
自律的に遂行するアプリケーション



「作業の指示」から「タスクの依頼」へ

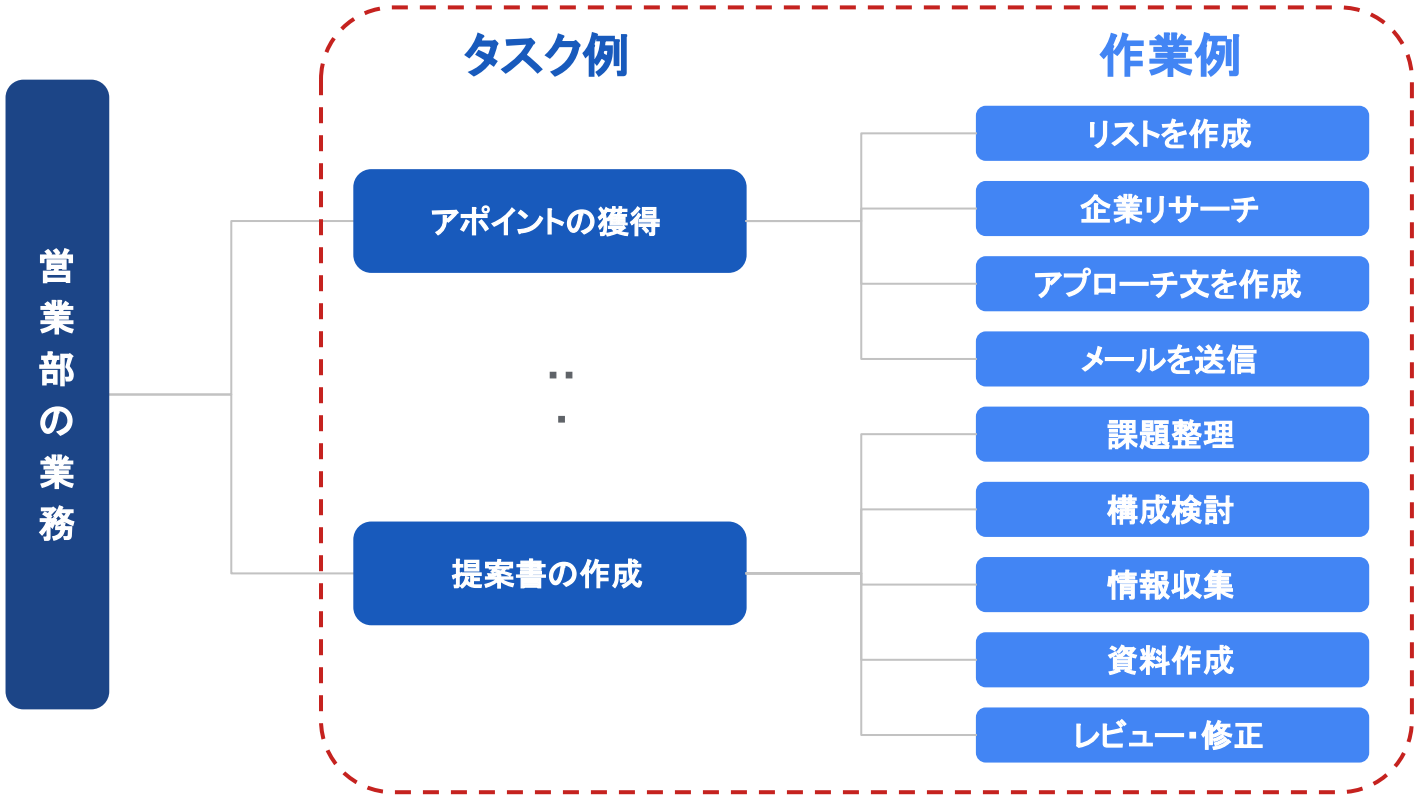


「作業の指示」から「タスクの依頼」へ



従来の LLM 活用：
作業単位で指示、作業オーナーは人

「作業の指示」から「タスクの依頼」へ



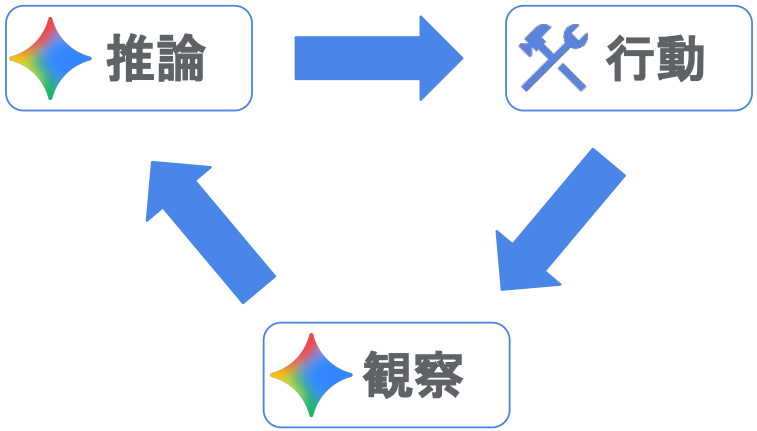
AI エージェント：
タスク単位で依頼、作業オーナーは　AI

ReAct : AI エージェントが自律的に行動する仕組み

ReAct (Reasoning and Acting) とは「**推論**」と「**行動**」のループによりLLM の性能を向上させるアプローチ

次のサイクルを繰り返すことにより回答の精度や信頼性を向上させる仕組み

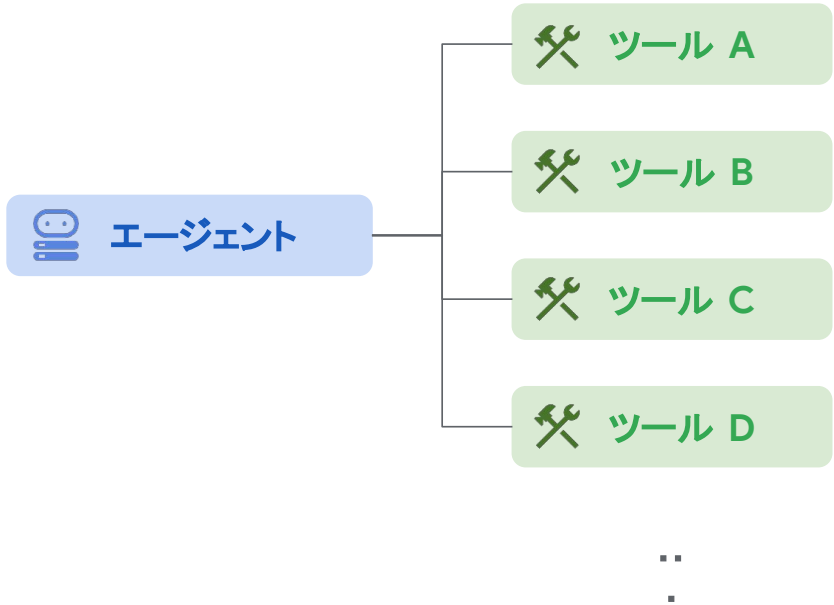
1. 思考 (Thought)
2. 行動 (Action)
3. 観察 (Observation)



<参考> Google Research Blog: ReAct: Synergizing Reasoning and Acting in Language Models

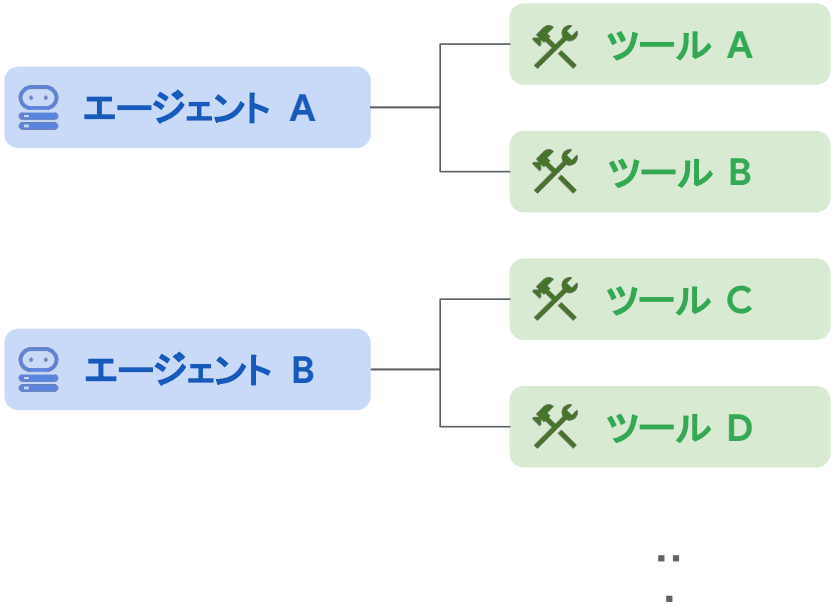
シングル エージェント + マルチツール

- コンテキストの肥大化
- プロンプトの複雑化による保守性・再利用性の低下
- 複雑・複数ステップのタスクへの対応が困難（思考が発散）



マルチ エージェント + マルチツール

- 複雑なタスクの分解と処理
  (決定論的なフロー制御の導入など)
- コンテキスト・プロンプトの最適化・簡素化
- モジュール化による保守性・再利用性の向上
- 専門性の確立



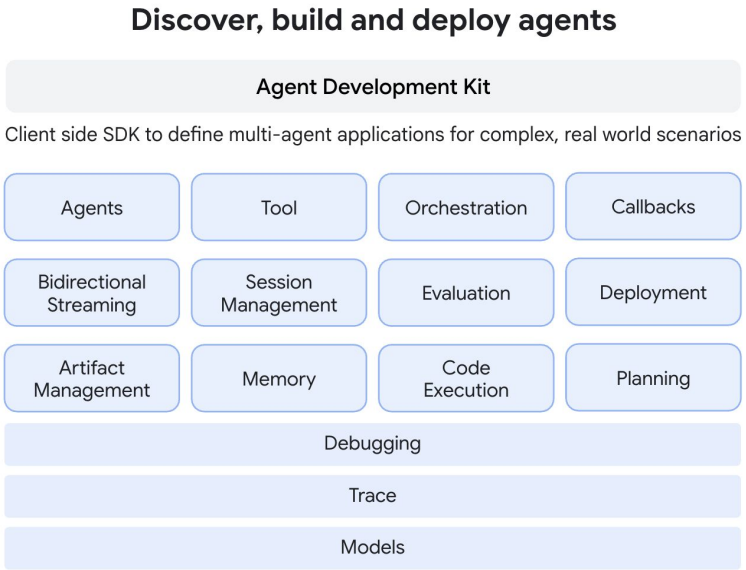
Proprietary

02. Agent Development Kit (ADK) 詳解

Agent Development Kit (ADK) の概要

AI エージェント開発に最適化されたPython および Java 向けのフレームワーク

- マルチエージェント設計
- 豊富なツール エコシステム
- 柔軟なオーケストレーション
- 統合された開発者ツール
- ネイティブなストリーミング サポート
- エージェント評価機能
- 幅広い LLM をサポート



直感的かつ少ないコード量でマルチ　エージェント開発を実現

シングル エージェントの場合

```
from google.adk.agents import LlmAgent
from google.adk.tools import google_search

root_agent = LlmAgent(
  name="search_assistant",
  model="gemini-2.5-flash",
  instruction="You are a helpful assistant. Answer user
questions using Google Search when needed.",
  description="An assistant that can search the web.",
  tools=[google_search]
)
```

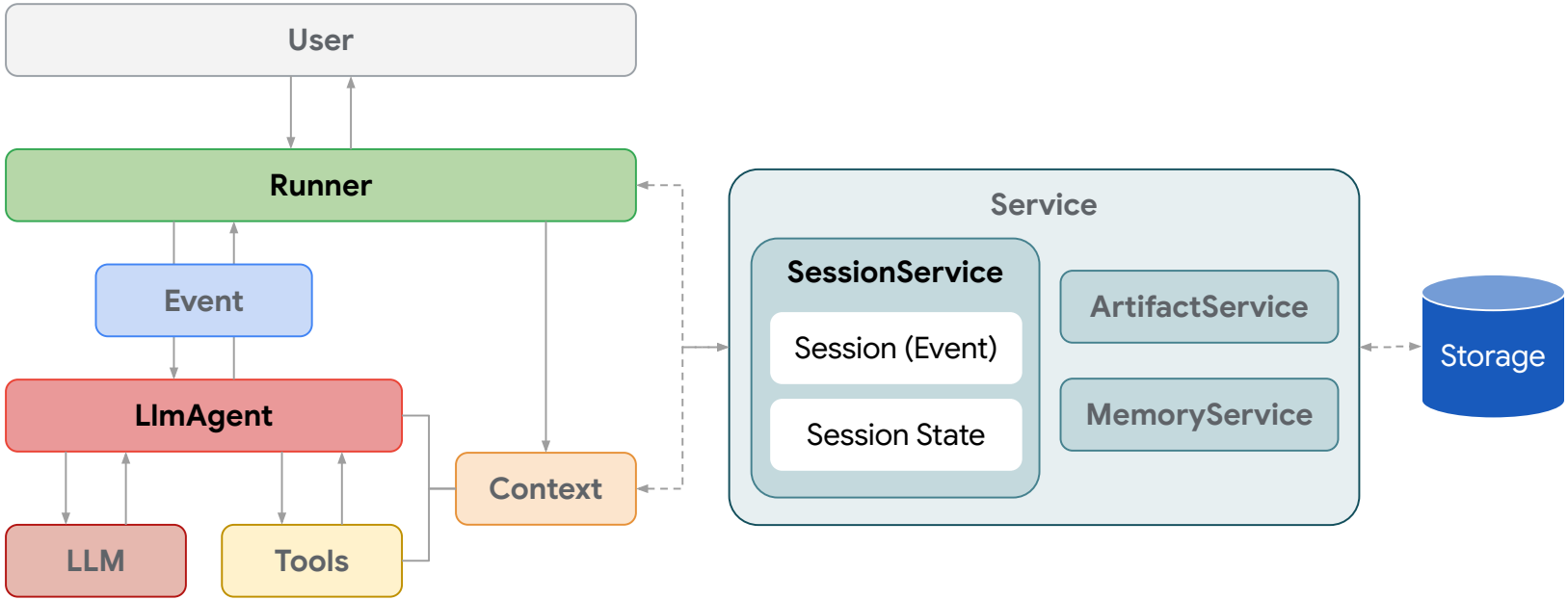
マルチ エージェントの場合

```
from google.adk.agents import LlmAgent

# Define individual agents
greeter = LlmAgent(name="greeter",
model="gemini-2.5-flash", ...)
task_executor = LlmAgent(name="task_executor",
model="gemini-2.5-flash", ...)

# Create parent agent and assign children via sub_agents
coordinator = LlmAgent(
   name="Coordinator",
   model="gemini-2.5-flash",
   description="I coordinate greetings and tasks.",
   sub_agents=[ # Assign sub_agents here
       greeter,
       task_executor
   ]
)
```

ADK のオブジェクト基本構成



主要オブジェクトは **Runner**, **LlmAgent**, **SessionService** の 3 つ

Events

- ADK 内で発生する全てのイベントの履歴を記録するオブジェクト
- Runner と Agent 間の情報の伝達に利用
- ユーザーとの対話のほか、ツールの呼び出し、内部的な判断、ステータスの変化、エラーなど、処理の全行程を記録

例) User Query : “What is the weather like in London?”

```
content=Content(
 parts=[
   Part(
     function_call=FunctionCall(
       args={
         'city': 'London'
       },
       id='adk-927fc321-68f0-4c6a-bf75-ec2d2e910dee',
       name='get_weather'
     )
   ),
 ],
 role='model'
) ...
content=Content(
 parts=[
   Part(
     function_response=FunctionResponse(
       id='adk-927fc321-68f0-4c6a-bf75-ec2d2e910dee',
       name='get_weather',
       response={
         'report': "It's cloudy in London with a temperature of 15°C.",
         'status': 'success'
       }
     )
   ),
 ],
 role='user'
) ...
content=Content(
 parts=[
   Part(
     text='The weather in London is cloudy with a temperature of 15°C.'
   ),
 ],
 role='model'
) ...
```

Tools

1. **Function Tools**
   - 独自に定義した関数をツールとして登録 (= Function Calling)
2. **Built-in Tools**
   - Google Search, Code Execution, Vertex AI Search, BigQuery 向けの組み込みツールを提供
3. **Third party Tools**
   - LangChain, CrewAI など他エージェント フレームワークのツールを利用
4. **Google Cloud Tools**
   - Google Cloud のサービスにシームレスに連携
   - 100 以上のオンプレ・SaaS コネクタも提供
5. **MCP Tools**
   - ADK を MCP クライアントとして外部の MCP サーバーが提供するツールに接続

ADK の Function Tools

- 関数の型ヒントや Doc String の情報から LLM が関数を呼び出すべきか判断
  = 事前の**関数宣言**の定義は不要
- Function Call イベントが発生するとフレームワーク側で自動的に関数を実行
  Function Response 内にツールの実行結果を記録

```python
def get_current_time(city: str) -> dict:
    """Returns the current time in a specified city.

    Args:
        city (str): The name of the city for which to retrieve the current time.

    Returns:
        dict: status and result or error msg.
    """

    if city.lower() == "new york":
        tz_identifier = "America/New_York"
    else:
        return {
            "status": "error",
            "error_message": (
                f"Sorry, I don't have timezone information for {city}."
            ),
        }

    tz = ZoneInfo(tz_identifier)
    now = datetime.datetime.now(tz)
    report = (
        f'The current time in {city} is {now.strftime("%Y-%m-%d %H:%M:%S %Z%z")}'
    )
    return {"status": "success", "report": report}
```

Built-in Tool 使用例

例) Google Search ツールを利用

```
root_agent = Agent(
   name="basic_search_agent",
   model="gemini-2.5-flash",
   description="Agent to answer questions using Google Search.",
   instruction="I can answer your questions by searching the internet. Just ask me anything!",
   tools=[google_search]
)
```

例) BigQuery ツールを利用

```
bigquery_agent = Agent(
   model="gemini-2.5-flash",
   name="bigquery_agent",
   description=(
       "Agent to answer questions about BigQuery data and models and execute SQL queries."
   ),
   instruction="""\
       You are a data science agent with access to several BigQuery tools.
       Make use of those tools to answer the user's questions.
   """,
   tools=[bigquery_toolset],
)
```

MCP Tool 使用例

例) Google Maps MCP サーバーに接続

```
root_agent = LlmAgent(
   model='gemini-2.5-flash',
   name='maps_assistant_agent',
   instruction='Help the user with mapping, directions, and finding places using Google Maps tools.',
   tools=[
       MCPToolset(
           connection_params=StdioConnectionParams(
               server_params = StdioServerParams(
                   command='npx',
                   args=[
                       "-y",
                       "@modelcontextprotocol/server-google-maps",
                   ],
                   env={
                       "GOOGLE_MAPS_API_KEY": google_maps_api_key
                   }
               ),
           ),
       )
   ],
)
```

SessionService

ユーザーとエージェント間の個々の会話履歴(Session オブジェクト) を管理するサービス
Session に格納されるデータは Events と State の 2 種類

SessionService の種類

種類	格納場所	永続性	用途
InMemorySessionService	インメモリに保存	なし	開発、ローカルテスト、サンプル実行
VertexAiSessionService	Vertex AI Agent Engine サービス内に保存	あり	Vertex AI 機能との統合を必要とするスケーラブルな本番アプリケーション
DatabaseSessionService	任意のリレーショナルデータベースに保存	あり	自身で管理する信頼性の高い永続ストレージが必要なアプリケーション

Session State

各セッション内でデータを格納・共有するためのオブジェクト

エージェント間でデータを共有するための短期記憶

データはシリアライズ可能なキーと値のペアで保存

Session State にアクセスする方法：

- Agent 内で output_key を指定することで State への保存ができ、instruction 内で {...} プレースホルダーを利用してキーを指定して読み取り
- Callback Context や Tool Context 経由でアクセス

```
greeting_agent = LlmAgent(
    name="Greeter",
    model="gemini-2.5-flash",
    instruction="Generate a short, friendly greeting.",
    output_key="last_greeting"
)

responder_agent = LlmAgent(
    name="Responder",
    model="gemini-2.5-flash",
    instruction="""
        You received this greeting: {last_greeting}.
        Respond to it politely.
        """,
    output_key="final_response"
)
```

state['last_greeting'] にレスポンスを保存

state['last_greeting'] の値を取得

MemoryService

複数のセッションをまたいだデータ(長期記憶) を格納・共有するためのオブジェクト

- 情報の記録 :memory_service.add_session_to_memory(session) を利用
- 情報の検索 :load_memory ツール、search_memory ツールを利用

MemoryService の種類

種類	格納場所	永続性	用途
InMemoryMemoryService	インメモリに保存	なし	開発、ローカルテスト、サンプル実行
VertexAiRagMemoryService	Vertex AI RAG Engine 上の RAG Corpus に保存	あり	セマンティック検索で関連性の高い検索が必要なアプリケーション
VertexAiMemoryBankService	Vertex AI Agent Engine サービス内に保存	あり	セマンティック検索で関連性の高い検索が必要、スケーラブルな本番アプリケーション

ArtifactService

バイナリデータ(画像, 音声, PDF) を格納するためのバージョン管理されたオブジェクト

特定の session_id に紐づくセッション スコープ (デフォルト) または user_id に紐づくユーザース コープを選択可能

ArtifactService の種類

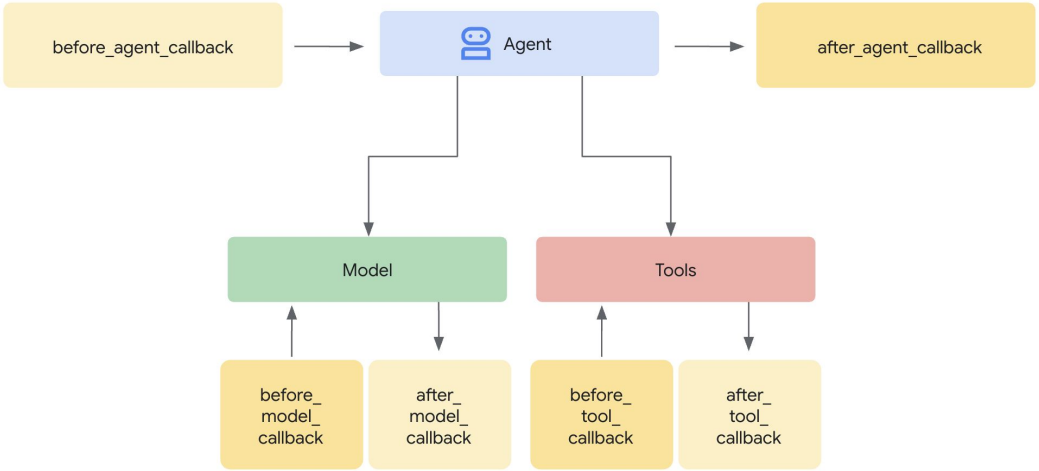
種類	格納場所	永続性	特徴
InMemoryArtifactService	インメモリに保存	なし	シンプルかつ低レイテンシ
GcsArtifactService	GCS に保存	あり	GCS のスケーラビリティと耐久性を提供

Callbacks

ユーザー定義の関数を任意のタイミングで呼び出すことが可能

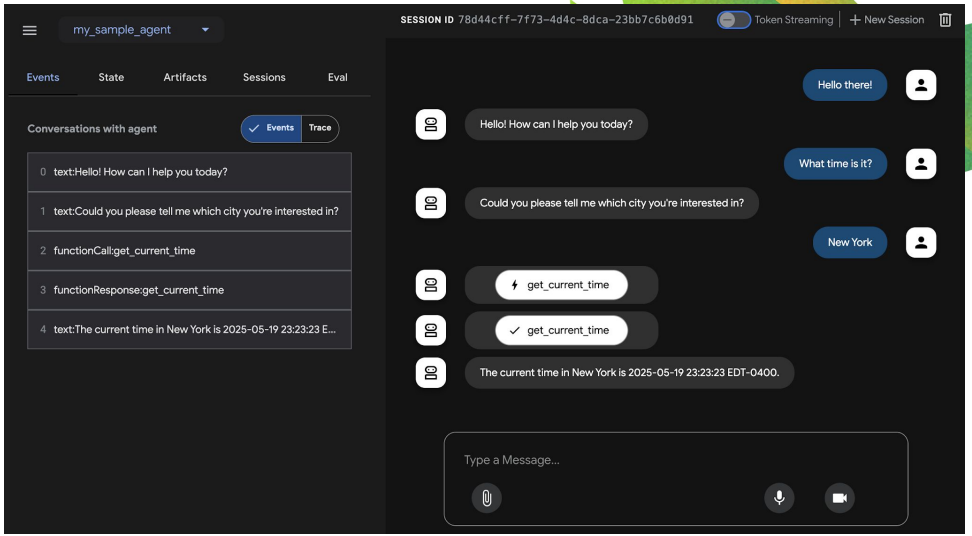
ユースケース:

- ガードレールとポリシーの実施
- 動的な状態管理
- ログとモニタリング
- キャッシュ
- リクエスト / レスポンスの変更
- 実行のスキップ
- ツール固有の認証や制御
- アーティファクトの処理



ローカル開発・テスト

1. Dev UI (adk web)
   - 開発とデバッグのためのインタラクティブな開発者向けUI
2. CLI (adk run)
   - ターミナル上で直接エージェントを実行
3. API Server (adk api_server)
   - エージェントを FastAPI ベースの API サーバーとして起動



組み込みの開発者向けUI

Workflow Agents

サブ エージェントの実行フローを制御するためのコンポーネント

LLM による確率論的なワークフローに加えて事前に定義されたロジックに基づいて動作する決定論的なフロー制御をハイブリッドに構成することが可能

Workflow Agents の種類

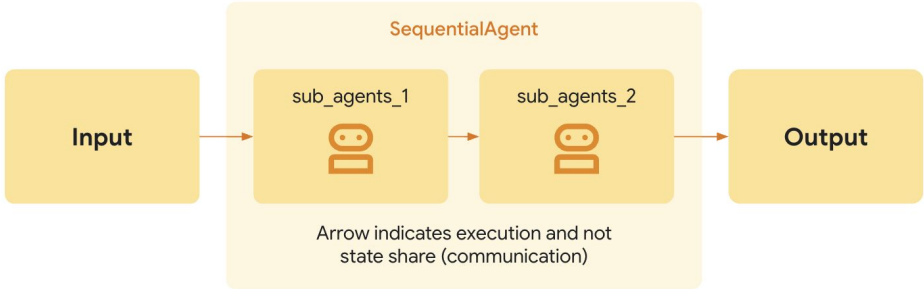
1. Sequential Agents
2. Parallel Agents
3. Loop Agents

Sequential Agent

サブ エージェントを指定された順序で実行するワークフロー エージェント

例) コード開発パイプライン

```
SequentialAgent(
    sub_agents=[
        CodeWriterAgent,
        CodeReviewerAgent,
        CodeRefactorerAgent
    ]
)
```



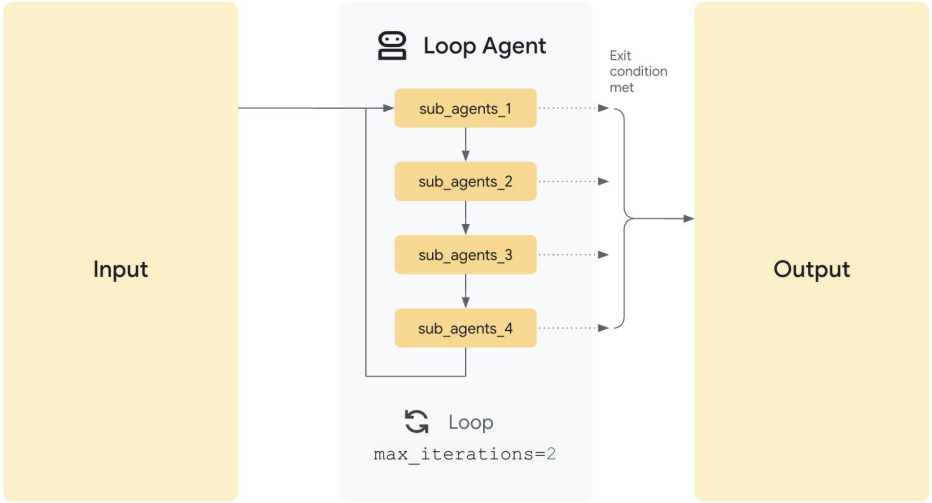
Loop Agent

サブ エージェントを反復的に実行するワークフロー エージェント

いつループを停止するかの終了条件をユーザー側で定義する必要あり

例) 反復的なドキュメントの改善

```
LoopAgent(
    sub_agents=[WriterAgent, CriticAgent],
    max_iterations=5
)
```



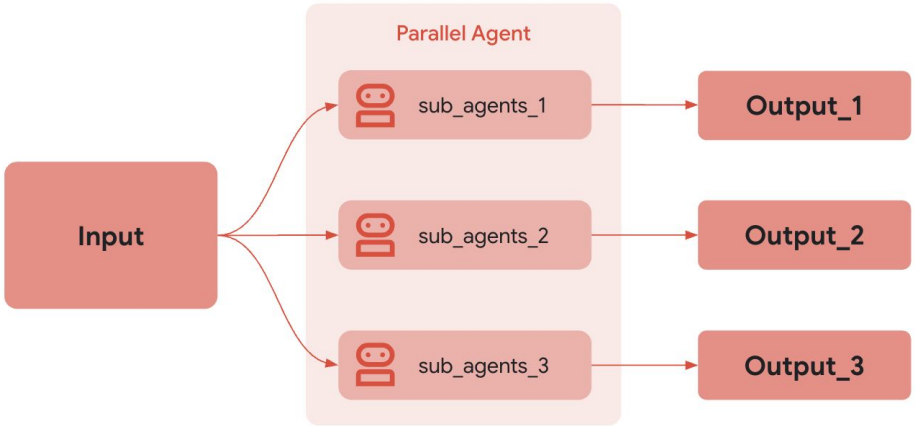
Parallel Agent

サブ エージェントを並列に実行するワークフローエージェント

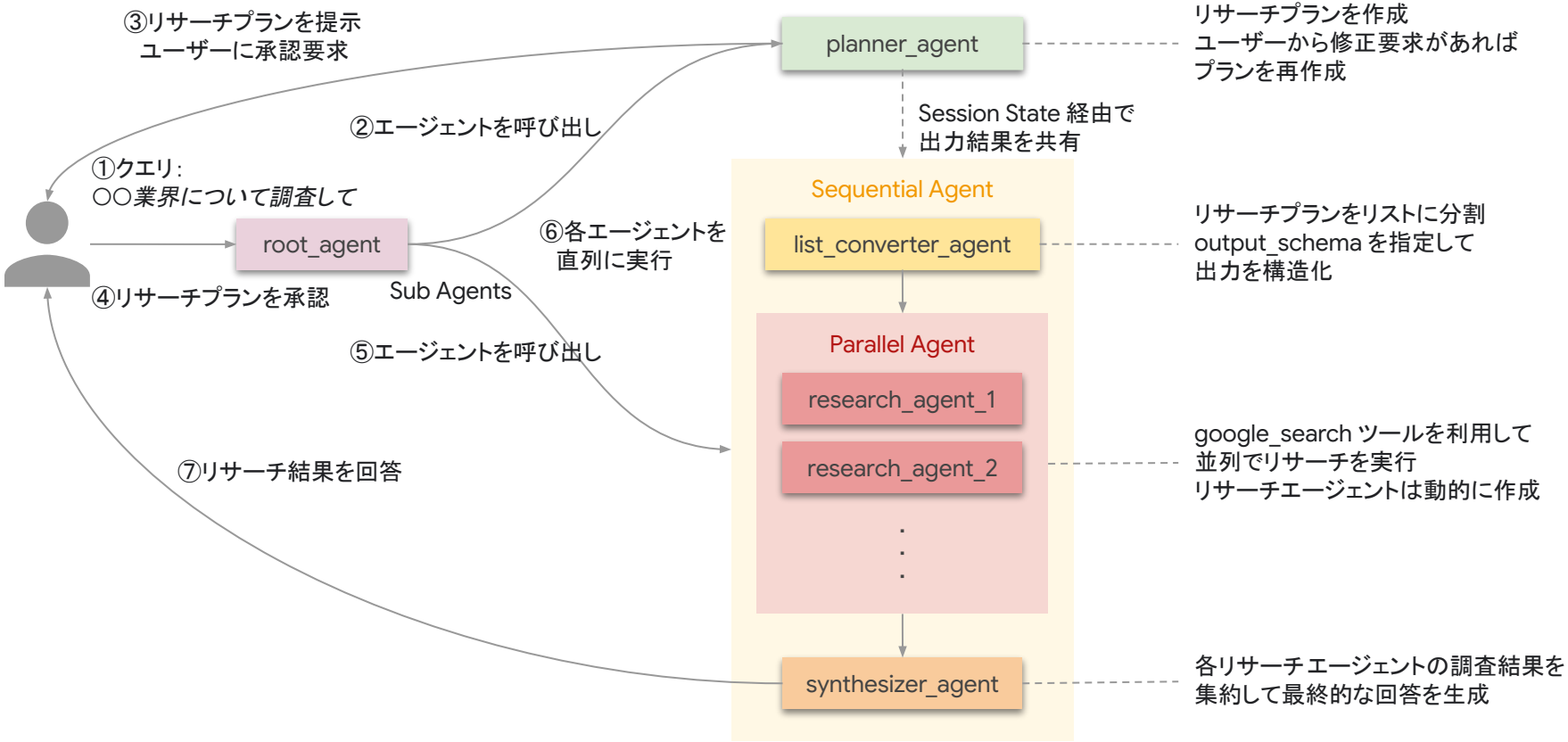
各サブエージェントは独立して実行

例) 並列でのウェブ調査

```
ParallelAgent(
    sub_agents=[
        ResearcherAgent1,
        ResearcherAgent2,
        ResearcherAgent3,
    ]
)
```

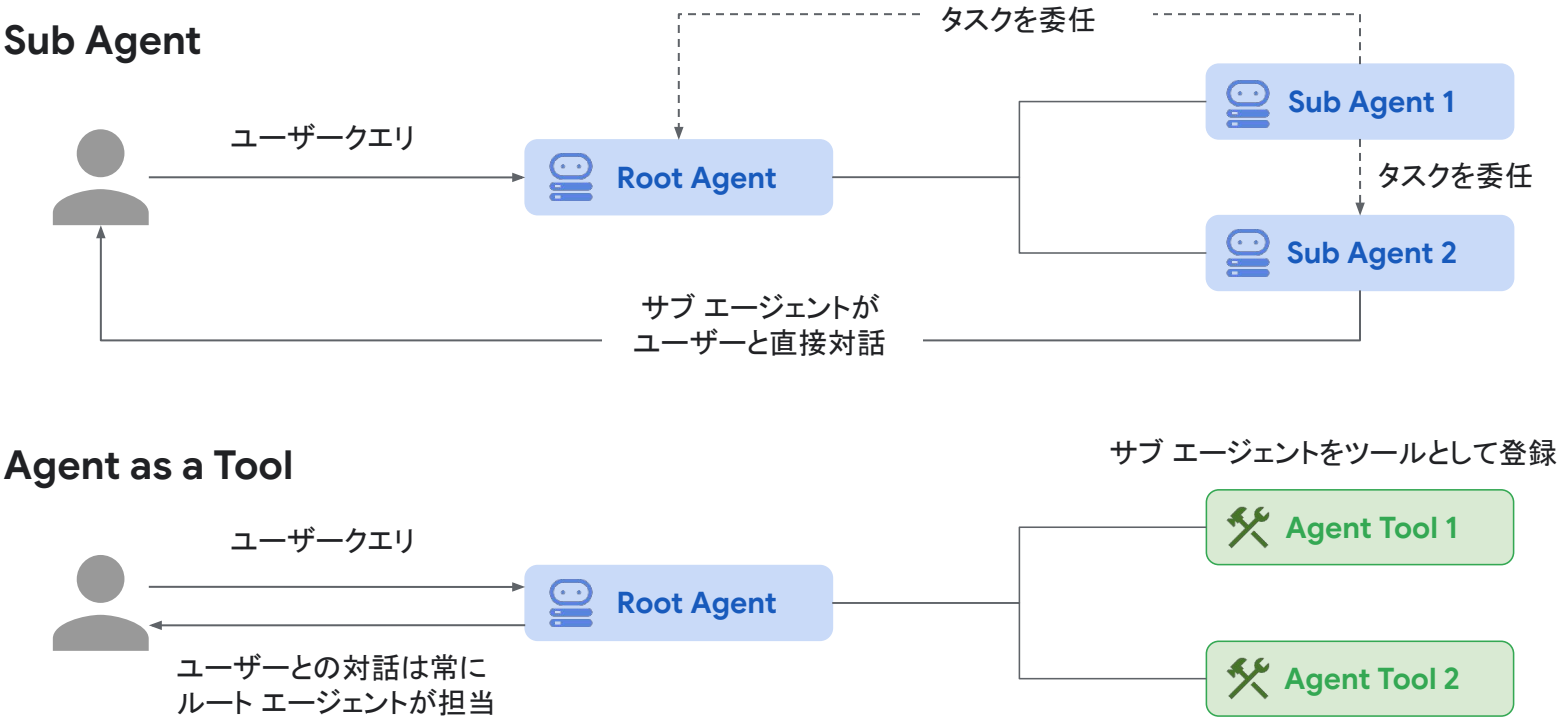


[デモ] リサーチ エージェント



動画あり
アーカイブ動画をご視聴ください

「Sub Agent」と「Agent as a Tool」の比較



Agent2Agent プロトコル

Agent 同士のやり取りのオープンな共通規格とエコシステム

エージェント同士のやり取り

エージェントは自身の機能を公開し、ユーザーとのインタラクション方法（テキスト、フォーム、双方向音声/ビデオ）を交渉しながら、安全に連携

オープン

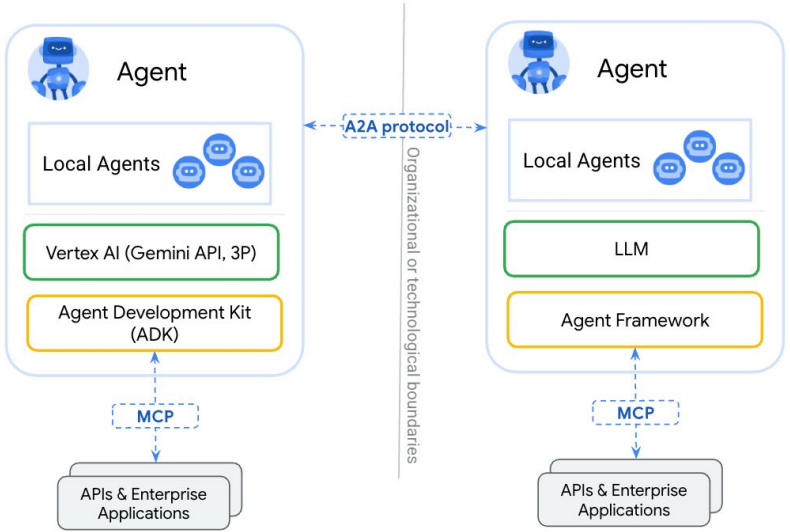
Linux Foundation に寄贈され、2025 年 7 月時点で 100 社以上の企業が A2A プロトコルをサポート

フレームワーク非依存

異なるエコシステムに属するエージェントが、基盤となるフレームワークやベンダーに関わらず、相互に通信が可能に

ADK との相互運用性

ADK との相互運用性を実現する機能を実験版として提供



03. Vertex AI Agent Engine のご紹介

Vertex AI Agent Engine

スケーラブルなデプロイメント

- API と**エージェントへのクエリ** を**管理**
- **セキュリティ** (e.g. VPC-SC) と **認証**
- **Agentspace** へのエージェントの登録
  ※ 2025 年 8 月現在 Private Preview 中

エンド ツー エンドの管理

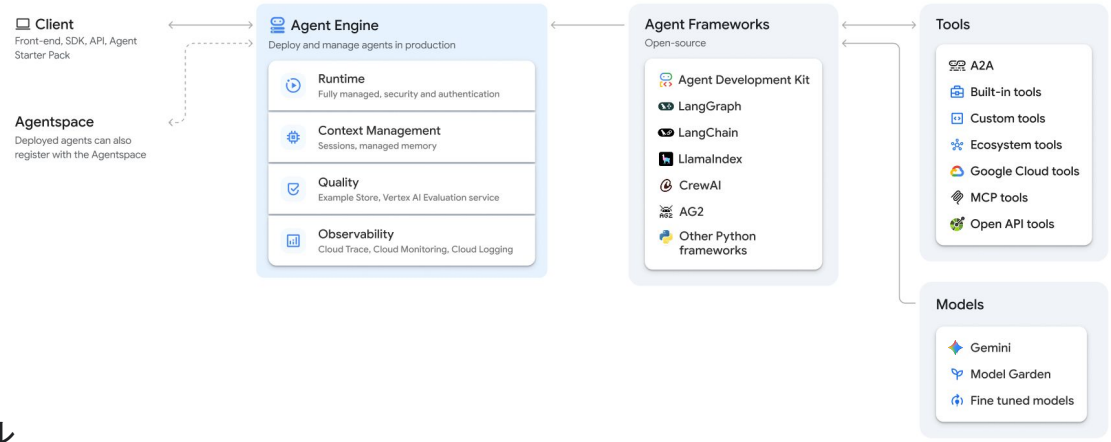
- **セッション** と**メモリ管理** のためのツール
- エージェントを一覧表示し会話を行う **UI**
- エージェントの**モニタリング** と**評価**

モデルへの非依存

- **Gemini**
- **Model Garden** の パートナー と OSS モデル
- **ファインチューンされたモデル**

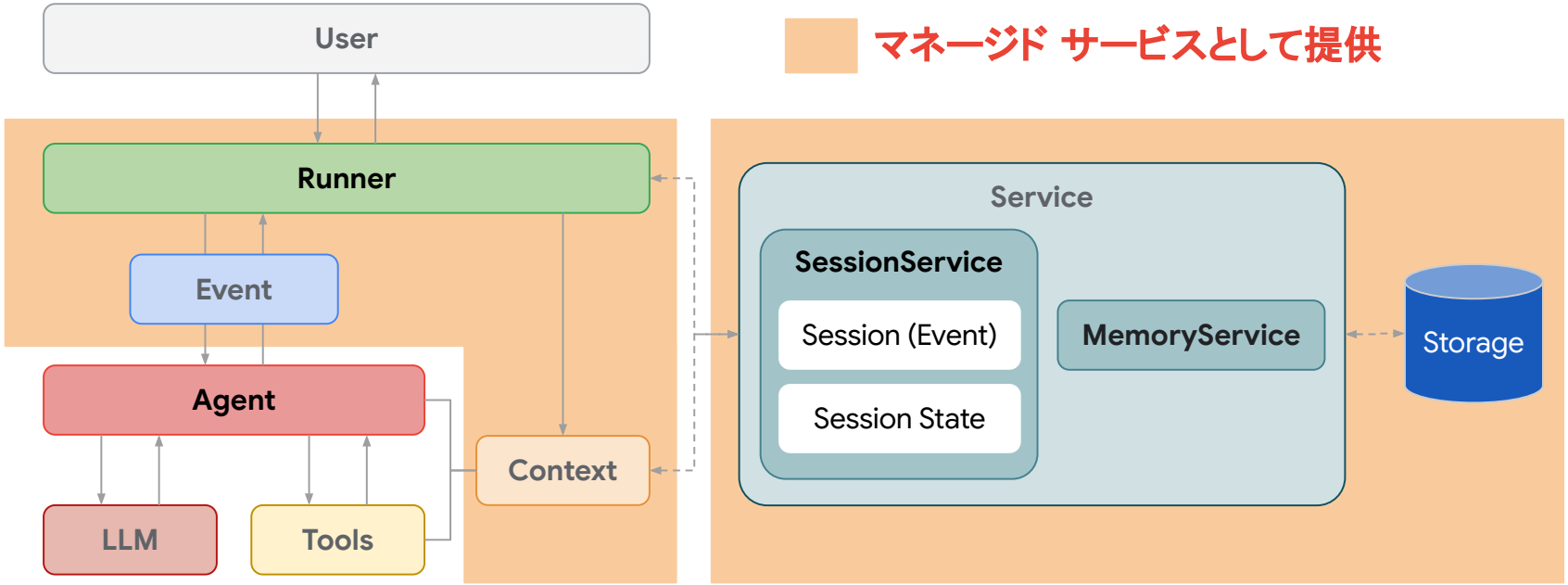
フレームワークへの非依存

- **Google Agent Development Kit**
- **3P OSS**: LangGraph, LangChain, Crew AI, AG2

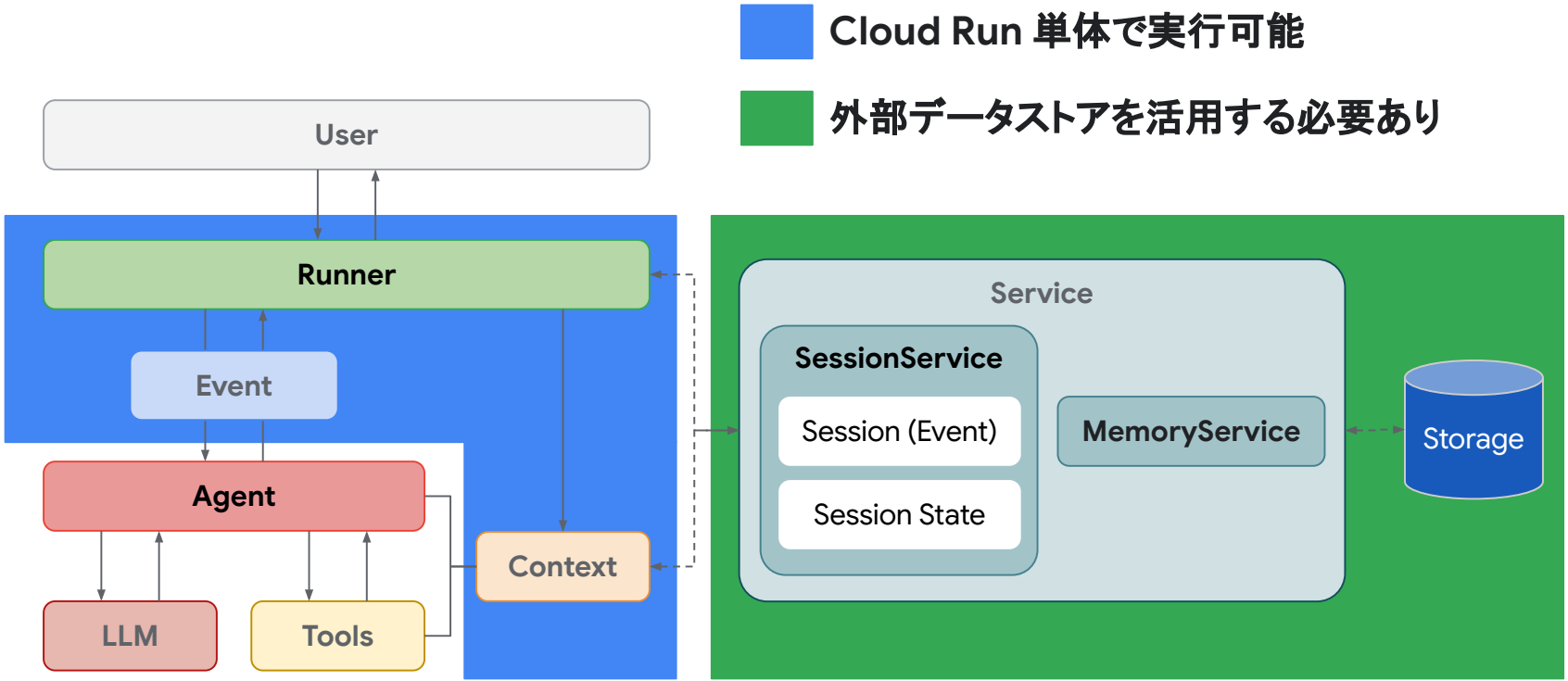


Agent Engine 活用でエージェント開発に専念

マネージドな Runner および SessionService, MemoryService を提供

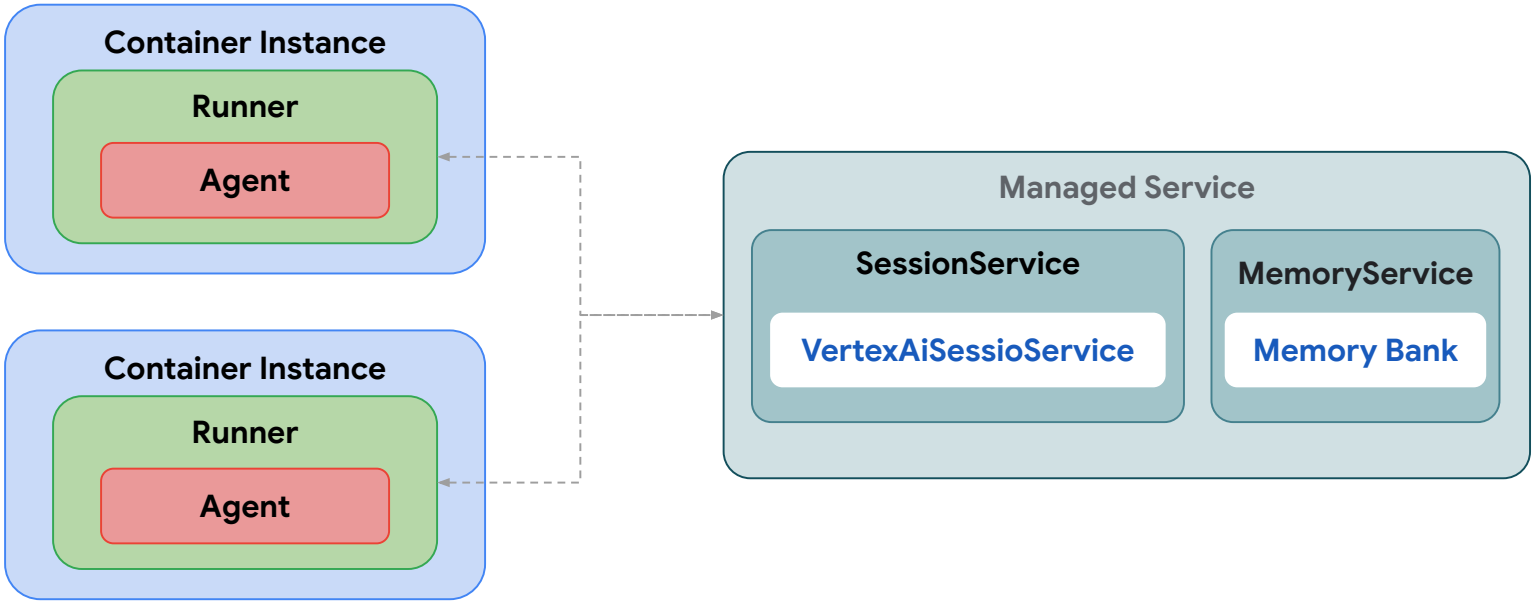


Cloud Run にデプロイする際の注意点



複数インスタンス間でも会話履歴を共有

セッション情報をインメモリではなく、独立した環境で提供するため、
複数コンテナでスケールアウトした場合にもセッション情報の共有が可能

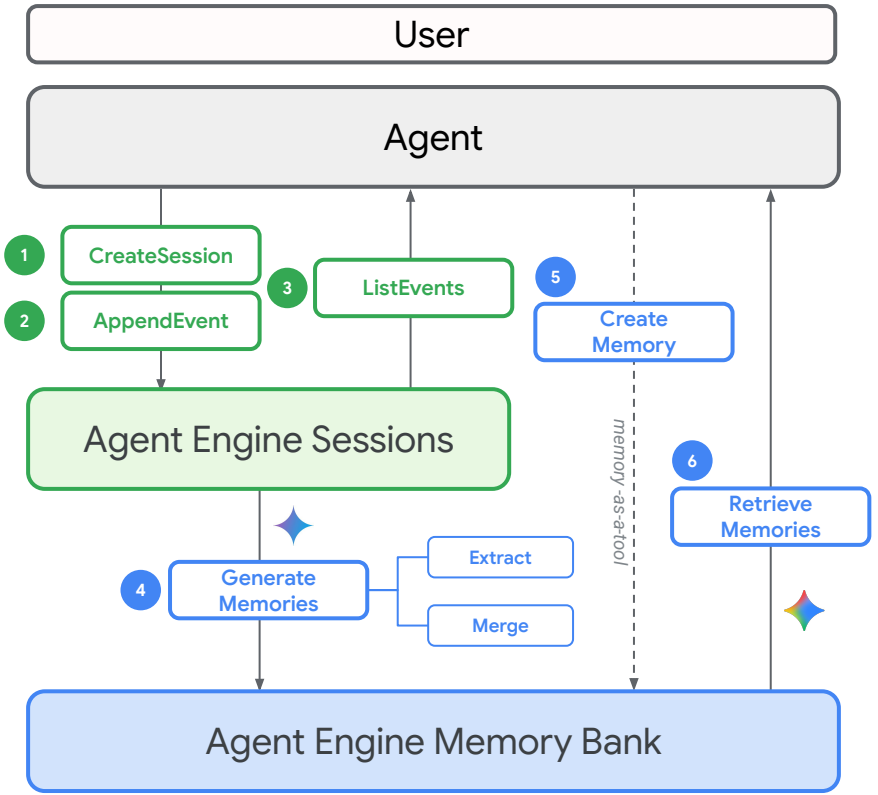


Memory Bank

Vertex AI Agent Engine 上で提供する複数セッションをまたいでデータ(長期記憶) の格納・共有をするための永続化されたストレージサービス

ADK の VertexAiMemoryBankService 経由で利用

メモリ検索 (search_memory) ではキーワードマッチではなく高度なセマンティック検索を提供



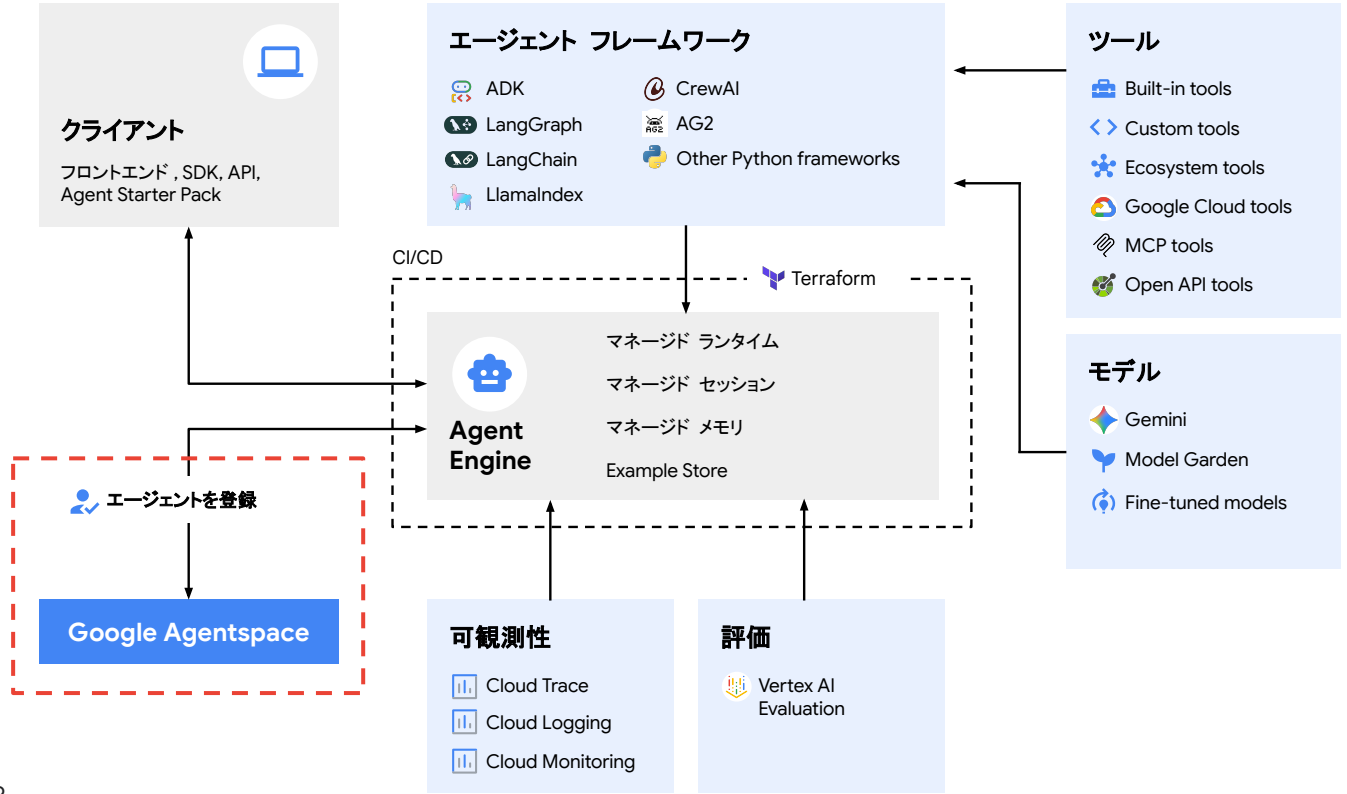
Proprietary

04. Google Agentspace との統合

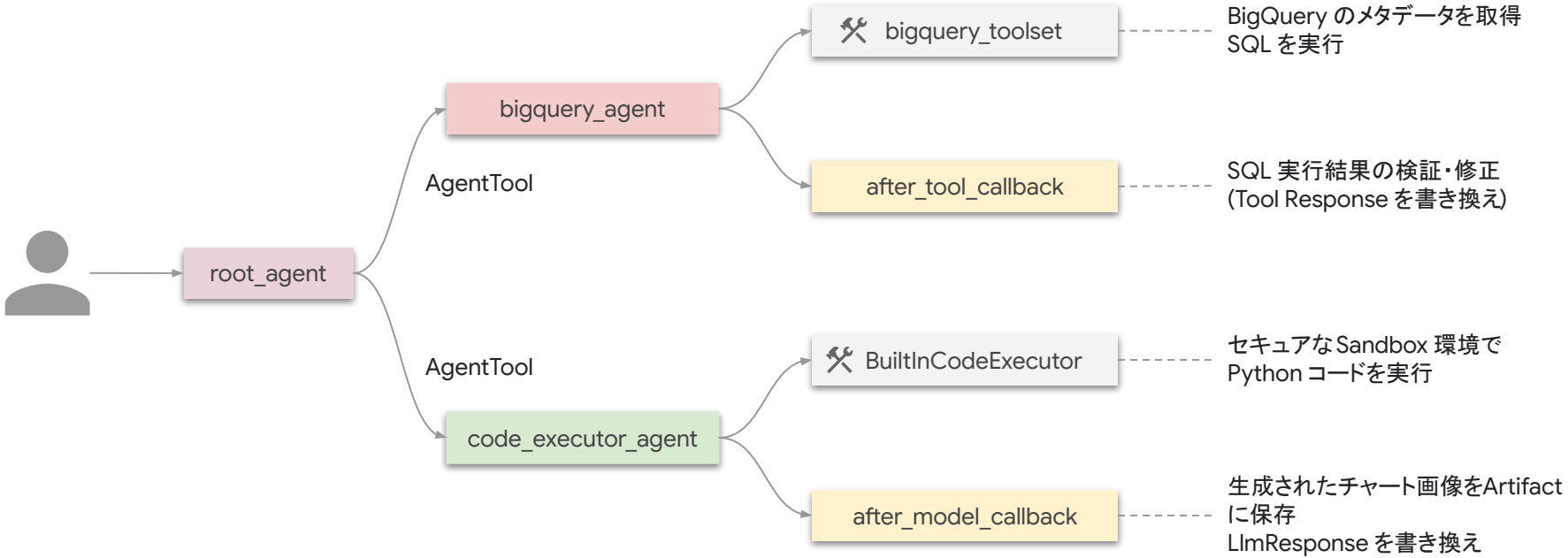
Google Agentspace に独自のカスタム エージェントを統合

Vertex AI Agent Engine 上のエージェントを Google Agentspace に登録して利用

※ 2025 年 8 月現在 Private Preview 中



[デモ] データ サイエンス エージェント

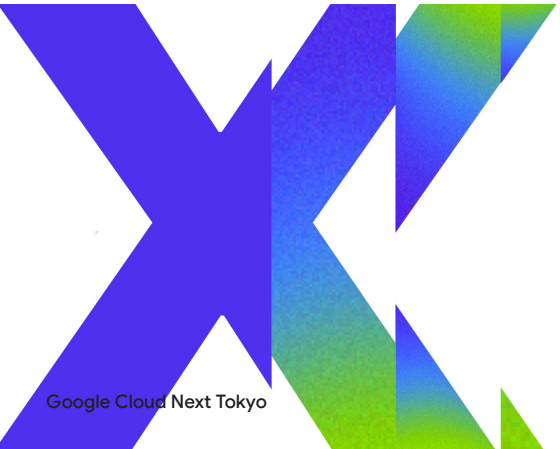


動画あり
アーカイブ動画をご視聴ください

※ この動画では待ち時間が発生する部分を編集で早送りしています。

まとめ

AI エージェントは、ビジネスにおけるAI 活用の可能性をさらに広げ、新たな価値を創出する「AI Transformation」の推進力

Google Cloud では、エンド ツー エンドでエージェント開発を支援するプラットフォームを提供

- **構築**: ADK で複雑なマルチ エージェント アプリケーションを効率的に開発
- **実行**: フルマネージドな Vertex AI Agent Engine で本番環境で求められるスケーラブルで安定的な実行環境をインフラ管理の手間なく実現
- **活用**: Google Agentspace をエージェントのハブとして組織全体で活用を促進