Google Agent Development Kit (ADK)

Text-to-SQL & Agentic AI Reference Guide

For AIAlgoTradeHits.com Development

Document Type:	Technical Reference Guide
Version:	1.0
Date:	December 07, 2025
Platform:	Google Cloud Platform
Target:	Claude Code / AI Development Agents

Table of Contents

1. Executive Summary

This document provides a comprehensive reference for implementing Google's Agent Development Kit (ADK) with text-to-SQL capabilities for the AIAlgoTradeHits.com trading platform. It covers the ADK framework, MCP Toolbox for database integration, BigQuery natural language queries, multi-agent orchestration patterns, and deployment strategies.

Key Technologies

Technology	Purpose	Status
Google ADK	Agent orchestration framework	Open Source (v1.19.0+)
MCP Toolbox	Database connectivity for agents	Open Source
BigQuery NL2SQL	Natural language to SQL queries	GA on Vertex AI
Vertex AI Agent Engine	Production deployment	Fully Managed
Gemini 2.5	LLM for agent reasoning	Production Ready

2. Google ADK Framework Overview

The Agent Development Kit (ADK) is Google's open-source framework for building, evaluating, and deploying sophisticated AI agents. It is model-agnostic and deployment-agnostic, providing the same framework that powers Google Agentspace and Customer Engagement Suite.

Core Capabilities

- **Multi-Agent Architecture:** Compose specialized agents in hierarchies with parent-child relationships
- **Flexible Orchestration:** Sequential, Parallel, Loop agents plus LLM-driven dynamic routing
- **Model Ecosystem:** Gemini, Vertex AI Model Garden, LiteLLM (Anthropic, Meta, Mistral, AI21)
- **Rich Tool Ecosystem:** Pre-built tools, custom functions, MCP tools, OpenAPI specs
- **Deployment Options:** Local, Cloud Run, GKE, Vertex AI Agent Engine (fully managed)
- **Built-in Observability:** Logging, Cloud Trace, AgentOps, Arize, MLflow integrations

Language Support

Language	Version	Requirements
Python	v1.19.0+	Python 3.10+
Go	v0.2.0	Go 1.20+
Java	v0.4.0	Java 11+

3. Installation & Setup

Python Installation

```
# Install Google ADK
pip install google-adk

# Install MCP Toolbox for database connectivity
pip install toolbox-core

# Verify installation
python -c "from google.adk.agents import Agent; print('ADK installed successfully')"
```

Go Installation

```
# Install ADK for Go
go get google.golang.org/adk
```

Environment Setup

```
# Set Google Cloud credentials
export GOOGLE_APPLICATION_CREDENTIALS="/path/to/service-account.json"
export GOOGLE_CLOUD_PROJECT="your-project-id"

# For Gemini API (if using API key)
export GOOGLE_API_KEY="your-api-key"

# For Vertex AI
export GOOGLE_CLOUD_LOCATION="us-central1"
```

Note: For AIAlgoTradeHits.com, ensure your GCP project has the following APIs enabled: Vertex AI API, BigQuery API, Cloud SQL Admin API, and Secret Manager API.

4. Text-to-SQL with MCP Toolbox

MCP Toolbox for Databases is an open-source MCP server that enables AI agents to securely connect to enterprise data sources. It supports 30+ database types including BigQuery, AlloyDB, Cloud SQL, Spanner, and self-managed PostgreSQL/MySQL.

Supported Databases

Database Type	Features	Best For
BigQuery	SQL, Schema Discovery, AI Forecasting	Analytics, Data Warehouse
AlloyDB PostgreSQL	Standard + Natural Language Queries	High-Performance OLTP
Cloud SQL	PostgreSQL, MySQL, SQL Server	Traditional Databases
Spanner	Globally Distributed SQL	Global Scale Applications
PostgreSQL (Self-Managed)	Full SQL Support	Existing Infrastructure
MySQL (Self-Managed)	Full SQL Support	Existing Infrastructure

Configuration File (tools.yaml)

```
# tools.yaml - Database source configuration
sources:
  trading-db:
    kind: postgres
    host: 127.0.0.1
    port: 5432
    database: trading_platform
    user: trading_user
    password: ${DB_PASSWORD}  # Use environment variable

  bigquery-source:
    kind: bigquery
    project: aialgotradedhits-project
    dataset: market_data

tools:
  search-trades-by-symbol:
    kind: postgres-sql
    source: trading-db
    description: Search for trades by stock/crypto symbol
    parameters:
      - name: symbol
        type: string
        description: The trading symbol (e.g., BTC, AAPL)
    statement: |
      SELECT * FROM trades
      WHERE symbol ILIKE '%' || $1 || '%'
      ORDER BY timestamp DESC
      LIMIT 100;

  get-ohlcv-data:
    kind: bigquery-sql
    source: bigquery-source
    description: Get OHLCV candlestick data for analysis
```

```
    parameters:
      - name: symbol
        type: string
      - name: interval
        type: string
      - name: limit
        type: integer
    statement: |
      SELECT timestamp, open, high, low, close, volume
      FROM `market_data.ohlcv`
      WHERE symbol = @symbol AND interval = @interval
      ORDER BY timestamp DESC
      LIMIT @limit;

  calculate-technical-indicators:
    kind: postgres-sql
    source: trading-db
    description: Get pre-calculated technical indicators
    parameters:
      - name: symbol
        type: string
    statement: |
      SELECT symbol, timestamp, rsi, macd, macd_signal,
             sma_20, sma_50, sma_200, bb_upper, bb_lower
      FROM technical_indicators
      WHERE symbol = $1
      ORDER BY timestamp DESC
      LIMIT 50;

toolsets:
  trading-toolset:
    - search-trades-by-symbol
    - get-ohlcv-data
    - calculate-technical-indicators
```

5. BigQuery NL2SQL Integration

BigQuery supports natural language queries through Vertex AI integration, allowing users to ask questions in plain English that are automatically converted to SQL queries.

First-Party BigQuery Tools

- **list_dataset_ids:** Fetch all dataset IDs in a project
- **get_dataset_info:** Fetch metadata and schema information
- **execute_sql:** Execute SQL queries and return results

Example Natural Language Queries

- "What are my top 10 performing trades this month?"
- "Show average return on BTC trades where RSI was above 70"
- "Which trading strategy had the highest Sharpe ratio in Q4?"
- "List all symbols with bullish MACD crossover in the last 24 hours"
- "What is the total trading volume by asset class this week?"

BigQuery ML for NL2SQL

```
-- Create embeddings for semantic search
SELECT ML.GENERATE_EMBEDDING(
  MODEL `project.dataset.embedding_model`,
  (SELECT 'What are my best trades?' AS content)
) AS query_embedding;

-- Use LLM for text generation from SQL
SELECT ML.GENERATE_TEXT(
  MODEL `project.dataset.gemini_model`,
  (SELECT CONCAT(
    'Analyze this trading data and provide insights: ',
    TO_JSON_STRING(results)
  ) AS prompt
  FROM trade_summary AS results),
  STRUCT(0.2 AS temperature, 1024 AS max_output_tokens)
) AS analysis;
```

6. Multi-Agent Architecture Patterns

ADK supports sophisticated multi-agent systems where specialized agents collaborate to accomplish complex tasks. Agents can be organized in hierarchies with parent-child relationships.

Workflow Agent Types

Agent Type	Behavior	Use Case
SequentialAgent	Execute agents in order	Pipeline processing
ParallelAgent	Run agents concurrently	Independent data gathering
LoopAgent	Iterative execution	Refinement, polling
LLM Router	Dynamic routing based on context	Intent classification

Trading Platform Agent Architecture

```
Root Agent (Trading Coordinator)
■■■ Market Data Agent
■   ■■■ Polygon.io API Integration
■   ■■■ Twelve Data API Integration
■   ■■■ Real-time WebSocket Handler
■■■ Technical Analysis Agent
■   ■■■ RSI, MACD, Stochastic Calculations
■   ■■■ Elliott Wave Pattern Detection
■   ■■■ Support/Resistance Identification
■■■ Database Query Agent (MCP Toolbox)
■   ■■■ BigQuery for Historical Data
■   ■■■ PostgreSQL for Trade Records
■   ■■■ Natural Language Query Handler
■■■ Signal Generation Agent
■   ■■■ AI Prediction Models (Vertex AI)
■   ■■■ Pattern Recognition
■   ■■■ Sentiment Analysis Integration
■■■ Risk Management Agent
    ■■■ Position Sizing Calculator
    ■■■ Stop-Loss/Take-Profit Logic
    ■■■ Portfolio Exposure Monitor
```

7. Deployment Options

Local Development

```
# Interactive browser UI for testing
adk web

# Terminal-based execution
adk run

# Run specific agent
adk run --agent trading_coordinator
```

Production Deployment Options

Option	Description	Best For
Vertex AI Agent Engine	Fully managed, auto-scaling	Production (Recommended)
Cloud Run	Containerized, serverless	Custom infrastructure
GKE	Kubernetes deployment	Complex orchestration
Docker	Any container environment	Hybrid/On-prem

Vertex AI Agent Engine Deployment

```
# Deploy to Vertex AI Agent Engine
from google.cloud import aiplatform

aiplatform.init(project="aialgotradedhits-project", location="us-central1")

# Create agent deployment
agent_deployment = aiplatform.Agent.create(
    display_name="trading-coordinator-agent",
    agent_config={
        "model": "gemini-2.5-flash",
        "tools": ["trading-toolset"],
        "instructions": "You are a trading analysis agent..."
    },
    scaling_config={
        "min_replicas": 1,
        "max_replicas": 10,
        "target_cpu_utilization": 70
    }
)

print(f"Agent deployed: {agent_deployment.resource_name}")
```

8. Implementation for Trading Platform

Complete Agent Implementation

```
from google.adk.agents import Agent
from google.adk.agents.workflow_agents import ParallelAgent, SequentialAgent
from toolbox_core import ToolboxSyncClient

# Initialize MCP Toolbox connection
toolbox = ToolboxSyncClient("https://127.0.0.1:5000")
trading_tools = toolbox.load_toolset('trading-toolset')

# Define specialized agents
market_data_agent = Agent(
    name="market_data_analyst",
    model="gemini-2.5-flash",
    instruction="""You are a market data specialist.
    Fetch and analyze real-time market data from APIs.
    Provide OHLCV data, volume analysis, and price trends.""",
    tools=[fetch_market_data, get_realtime_quotes]
)

technical_analysis_agent = Agent(
    name="technical_analyst",
    model="gemini-2.5-flash",
    instruction="""You are a technical analysis expert.
    Calculate and interpret technical indicators including:
    - RSI, MACD, Stochastic oscillators
    - Moving averages (SMA, EMA)
    - Bollinger Bands, Elliott Wave patterns
    - Support and resistance levels""",
    tools=trading_tools
)

database_agent = Agent(
    name="database_analyst",
    model="gemini-2.5-flash",
    instruction="""You query trading databases to retrieve
    historical data, trade records, and calculated metrics.
    Convert natural language questions to SQL queries.""",
    tools=trading_tools
)

signal_agent = Agent(
    name="signal_generator",
    model="gemini-2.5-pro",  # Use Pro for complex reasoning
    instruction="""Generate trading signals by combining:
    - Technical indicator analysis
    - Pattern recognition results
    - Sentiment data
    - AI price predictions
    Provide BUY/SELL/HOLD recommendations with confidence scores.""",
    tools=[generate_signal, get_ai_prediction]
)

# Compose multi-agent workflow
data_gathering = ParallelAgent(
    name="data_gathering",
    sub_agents=[market_data_agent, database_agent]
```

```
)

trading_workflow = SequentialAgent(
    name="trading_analysis_pipeline",
    sub_agents=[
        data_gathering,
        technical_analysis_agent,
        signal_agent
    ]
)

# Root coordinator agent
root_agent = Agent(
    name="trading_coordinator",
    model="gemini-2.5-flash",
    instruction="""You coordinate trading analysis by:
    1. Understanding user queries about markets
    2. Delegating to specialized agents
    3. Synthesizing results into actionable insights
    4. Providing clear trading recommendations""",
    sub_agents=[trading_workflow]
)

# Execute agent
async def analyze_trade(query: str):
    response = await root_agent.run(query)
    return response.content
```

9. Code Examples & Templates

Custom Tool Definition

```
from google.adk.tools import tool

@tool
def calculate_position_size(
    account_balance: float,
    risk_percent: float,
    entry_price: float,
    stop_loss_price: float
) -> dict:
    """Calculate optimal position size based on risk management rules.

    Args:
        account_balance: Total account balance in USD
        risk_percent: Maximum risk per trade (e.g., 2.0 for 2%)
        entry_price: Planned entry price
        stop_loss_price: Stop loss price level

    Returns:
        Position sizing details including shares/units and dollar amount
    """
    risk_amount = account_balance * (risk_percent / 100)
    price_risk = abs(entry_price - stop_loss_price)
    position_size = risk_amount / price_risk if price_risk > 0 else 0

    return {
        "position_size": round(position_size, 4),
        "dollar_amount": round(position_size * entry_price, 2),
        "risk_amount": round(risk_amount, 2),
        "risk_reward_ratio": None  # Calculate if take_profit provided
    }

@tool
def fetch_sentiment_score(symbol: str) -> dict:
    """Fetch sentiment analysis from X (Twitter) and news sources.

    Args:
        symbol: Trading symbol (e.g., BTC, AAPL)

    Returns:
        Sentiment scores and analysis summary
    """
    # Integration with X Premium+ API and news APIs
    # Returns aggregated sentiment data
    pass
```

Session Management

```
from google.adk.sessions import InMemorySessionService

# Initialize session service for conversation memory
session_service = InMemorySessionService()

# Create session for user
session = session_service.create_session(
```

```
    app_name="aialgotradedhits",
    user_id="user_123",
    metadata={
        "preferred_assets": ["BTC", "ETH", "AAPL"],
        "risk_tolerance": "moderate",
        "trading_style": "swing"
    }
)

# Use session with agent
response = await root_agent.run(
    query="Analyze BTC for potential entry points",
    session_id=session.session_id
)
```

10. Cost Considerations

The Google ADK framework itself is free and open-source. Costs are incurred from the underlying Google Cloud services used by your agents.

Service	Pricing Model	Estimated Monthly Cost
ADK Framework	Free (Open Source)	$0
Vertex AI (Gemini)	Per 1K tokens	$50-150
BigQuery	Per TB scanned	$20-50
Cloud SQL / AlloyDB	Instance hours	$50-200
MCP Toolbox	Free (Open Source)	$0
Cloud Run (if used)	Per request + compute	$10-50

Budget Alignment: Based on the AIAlgoTradeHits.com business plan, the current budget allocates $70/mo for GCP infrastructure and $150/mo for Vertex AI, which should accommodate the ADK-based agent system for initial development and moderate production usage.

11. Resources & References

Official Documentation

- **ADK Documentation:** https://google.github.io/adk-docs/
- **ADK Python GitHub:** https://github.com/google/adk-python
- **ADK Go GitHub:** https://github.com/google/adk-go
- **MCP Toolbox GitHub:** https://github.com/googleapis/genai-toolbox
- **Vertex AI Agents:** https://cloud.google.com/vertex-ai/docs/agents
- **BigQuery ML:** https://cloud.google.com/bigquery/docs/bqml-introduction

Tutorials & Codelabs

- Google Codelabs: ADK Quickstart
- Google Codelabs: MCP Toolbox for Databases
- Google Codelabs: Building Multi-Agent Systems
- Google Codelabs: BigQuery + Vertex AI Integration

Key APIs to Enable in GCP

- Vertex AI API
- BigQuery API
- Cloud SQL Admin API
- Secret Manager API
- Cloud Run API (for deployment)
- Artifact Registry API (for containers)

Document Purpose: This reference guide is designed for use by Claude Code and other AI development agents working on the AIAlgoTradeHits.com trading platform. It provides the essential information needed to implement Google ADK with text-to-SQL capabilities for building sophisticated multi-agent trading analysis systems.