MCP Toolbox para BigQuery y Bases de Datos en la Nube

Una implementación integral del Model Context Protocol (MCP) Toolbox para integrar Google BigQuery y otras bases de datos en la nube con Claude Desktop y Google AI Development Kit (ADK). Este repositorio demuestra cómo construir agentes de IA potenciados por bases de datos usando interfaces de lenguaje natural.

Tabla de Contenidos

Descripción General

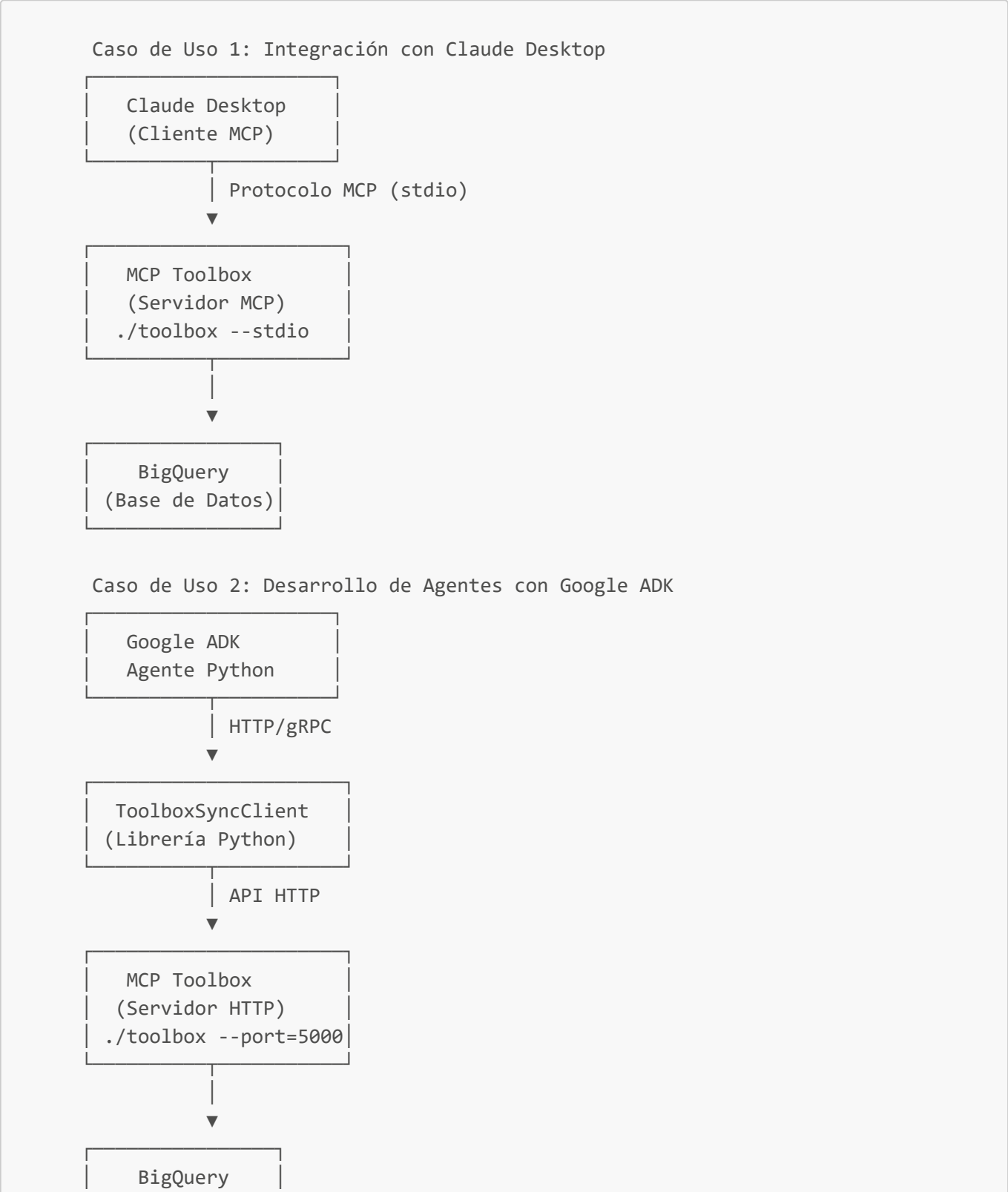
El MCP Toolbox permite a los modelos de IA interactuar directamente con bases de datos a través de un protocolo estandarizado. Esta implementación se enfoca en la integración con Google BigQuery, proporcionando:

- **Consultas a Bases de Datos en Lenguaje Natural**: Convierte preguntas de usuarios en consultas SQL automáticamente

- **Soporte Multi-Base de Datos**: Conecta con BigQuery, Cloud SQL, AlloyDB, Spanner, y más de 20 otras bases de datos
- **Autenticación Segura**: Soporte para service accounts y Application Default Credentials
- **Desarrollo de Agentes**: Construye agentes de IA personalizados usando el framework ADK de Google
- **Integración con Claude Desktop**: Acceso directo a bases de datos desde la aplicación de escritorio de Claude

Arquitectura

El MCP Toolbox soporta dos casos de uso distintos con arquitecturas separadas:



```
      │ (Base de Datos)│
      └────────────────┘
```

Puntos Clave:

- Claude Desktop y Google ADK son sistemas completamente separados e independientes
- Ambos pueden usar MCP Toolbox pero a través de diferentes interfaces:
  - Claude Desktop: Usa el protocolo MCP vía comunicación stdio
  - Google ADK: Usa HTTP/gRPC vía la librería Python ToolboxSyncClient
- No hay conexión directa entre Claude Desktop y Google ADK

Prerrequisitos

Requisitos del Sistema

- **Sistema Operativo**: Windows 10+, Ubuntu 20.04+, macOS 11+
- **Memoria**: Mínimo 4GB RAM (8GB recomendado)
- **Espacio en Disco**: 500MB para toolbox y dependencias
- **Red**: Conexión a internet para acceso a bases de datos en la nube

Dependencias de Software

1. **Google Cloud SDK** (requerido para acceso a BigQuery)
2. **Claude Desktop** (para integración con cliente MCP)
3. **Python 3.8+** (para desarrollo de agentes ADK)
4. **Git** (para gestión del repositorio)

Requisitos de Google Cloud

- Proyecto activo de Google Cloud
- API de BigQuery habilitada
- Service account con permisos apropiados
- Facturación habilitada para uso en producción

Instalación

Instalación en Windows

1. Instalar Google Cloud SDK

```powershell
# Descargar el instalador
Invoke-WebRequest -Uri
https://dl.google.com/dl/cloudsdk/channels/rapid/GoogleCloudSDKInstaller.exe -
OutFile GoogleCloudSDKInstaller.exe

# Ejecutar el instalador (seguir las instrucciones en pantalla)
.\GoogleCloudSDKInstaller.exe
```

```
# Inicializar gcloud
gcloud init
```

2. Descargar el Binario de MCP Toolbox

```
# Establecer versión
$VERSION = "0.15.0"

# Descargar binario para Windows
Invoke-WebRequest -Uri "https://storage.googleapis.com/genai-
toolbox/v$VERSION/windows/amd64/toolbox.exe" -OutFile toolbox.exe

# Verificar instalación
.\toolbox.exe --version
```

3. Configurar Variables de Entorno

```
# Establecer credenciales de Google Cloud
$env:GOOGLE_APPLICATION_CREDENTIALS = "D:\repos\mcp-toolbox\complete-tube-421007-
208a4862c992.json"

# Agregar a las variables de entorno del sistema (permanente)
[System.Environment]::SetEnvironmentVariable('GOOGLE_APPLICATION_CREDENTIALS',
'D:\repos\mcp-toolbox\complete-tube-421007-208a4862c992.json',
[System.EnvironmentVariableTarget]::User)
```

Instalación en Linux

1. Instalar Google Cloud SDK

```
# Agregar la URI de distribución del Cloud SDK como fuente de paquetes
echo "deb [signed-by=/usr/share/keyrings/cloud.google.gpg]
https://packages.cloud.google.com/apt cloud-sdk main" | sudo tee -a
/etc/apt/sources.list.d/google-cloud-sdk.list

# Importar la clave pública de Google Cloud
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key --
keyring /usr/share/keyrings/cloud.google.gpg add -

# Actualizar e instalar el Cloud SDK
sudo apt-get update && sudo apt-get install google-cloud-cli

# Componentes adicionales
sudo apt-get install google-cloud-cli-app-engine-python
sudo apt-get install google-cloud-cli-app-engine-python-extras
```

```
# Inicializar gcloud
gcloud init
```

2. Descargar el Binario de MCP Toolbox

```
# Establecer versión
export VERSION=0.15.0

# Descargar binario para Linux
curl -O https://storage.googleapis.com/genai-toolbox/v$VERSION/linux/amd64/toolbox
chmod +x toolbox

# Mover a la ruta del sistema (opcional)
sudo mv toolbox /usr/local/bin/

# Verificar instalación
toolbox --version
```

3. Configurar Variables de Entorno

```
# Agregar a ~/.bashrc o ~/.zshrc
export GOOGLE_APPLICATION_CREDENTIALS="/ruta/a/tu/service-account-key.json"

# Aplicar cambios
source ~/.bashrc
```

Instalación en macOS

```
# Instalar vía Homebrew (recomendado)
brew tap googleapis/toolbox
brew install mcp-toolbox

# O descargar binario directamente
export VERSION=0.15.0

# Para Apple Silicon
curl -O https://storage.googleapis.com/genai-
toolbox/v$VERSION/darwin/arm64/toolbox

# Para Intel Macs
curl -O https://storage.googleapis.com/genai-
toolbox/v$VERSION/darwin/amd64/toolbox

chmod +x toolbox
```

Configuración de Google Cloud

Configuración de Service Account

1. Crear Service Account

```
# Establecer tu ID de proyecto
export PROJECT_ID="complete-tube-421007"

# Crear service account
gcloud iam service-accounts create mcp-toolbox-sa \
    --display-name="Cuenta de Servicio MCP Toolbox" \
    --project=$PROJECT_ID

# Otorgar permisos de BigQuery
gcloud projects add-iam-policy-binding $PROJECT_ID \
    --member="serviceAccount:mcp-toolbox-sa@$PROJECT_ID.iam.gserviceaccount.com" \
    --role="roles/bigquery.user"

gcloud projects add-iam-policy-binding $PROJECT_ID \
    --member="serviceAccount:mcp-toolbox-sa@$PROJECT_ID.iam.gserviceaccount.com" \
    --role="roles/bigquery.dataEditor"
```

2. Generar Clave de Service Account

```
# Crear y descargar clave
gcloud iam service-accounts keys create service-account-key.json \
    --iam-account=mcp-toolbox-sa@$PROJECT_ID.iam.gserviceaccount.com

# Establecer variable de entorno
export GOOGLE_APPLICATION_CREDENTIALS="$(pwd)/service-account-key.json"
```

Configuración del Dataset de BigQuery

1. Crear Dataset y Tabla

```
-- Crear dataset
CREATE SCHEMA IF NOT EXISTS `complete-tube-421007.test`
OPTIONS(
  location="US",
  description="Sistema de reservas de hotel para demo de MCP Toolbox"
);

-- Crear tabla de hoteles
CREATE OR REPLACE TABLE `complete-tube-421007.test.hotels` (
  id INT64 NOT NULL,
  name STRING NOT NULL,
```

```
  location STRING NOT NULL,
  price_per_night NUMERIC(10,2),
  available_rooms INT64,
  rating FLOAT64,
  booked BOOL DEFAULT FALSE,
  checkin_date DATE,
  checkout_date DATE,
  amenities ARRAY<STRING>,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
);

-- Insertar datos de ejemplo
INSERT INTO `complete-tube-421007.test.hotels`
(id, name, location, price_per_night, available_rooms, rating, amenities)
VALUES
  (1, 'Grand Plaza Hotel', 'Nueva York', 250.00, 50, 4.5, ['WiFi', 'Piscina',
'Gimnasio']),
  (2, 'Seaside Resort', 'Miami Beach', 180.00, 30, 4.2, ['Acceso a Playa', 'Spa',
'Restaurante']),
  (3, 'Mountain Lodge', 'Aspen', 350.00, 20, 4.8, ['Acceso a Esquí', 'Chimenea',
'Jacuzzi']),
  (4, 'City Center Inn', 'Chicago', 120.00, 40, 3.9, ['WiFi', 'Estacionamiento',
'Desayuno']),
  (5, 'Luxury Suites', 'Las Vegas', 400.00, 25, 4.7, ['Casino', 'Piscina',
'Entretenimiento']);
```

Configuración

Archivos de Definición de Herramientas

toolsdb.yaml - Sistema de Reservas de Hotel

```
sources:
  my-bigquery-source:
    kind: bigquery
    project: complete-tube-421007
    location: US  # Debe coincidir con la ubicación de tu dataset

tools:
  search-hotels-by-name:
    kind: bigquery-sql
    source: my-bigquery-source
    description: Buscar hoteles por nombre con coincidencia parcial
    parameters:
      - name: name
        type: string
        description: Nombre del hotel o nombre parcial
        required: true
    statement: |
      SELECT
```

```
      id,
      name,
      location,
      price_per_night,
      available_rooms,
      rating,
      booked,
      ARRAY_TO_STRING(amenities, ', ') as amenities_list
    FROM `test.hotels`
    WHERE LOWER(name) LIKE LOWER(CONCAT('%', @name, '%'))
    ORDER BY rating DESC;

search-hotels-by-location:
  kind: bigquery-sql
  source: my-bigquery-source
  description: Encontrar hoteles en una ubicación específica
  parameters:
    - name: location
      type: string
      description: Ciudad o nombre del área
      required: true
  statement: |
    SELECT
      id,
      name,
      location,
      price_per_night,
      available_rooms,
      rating,
      booked
    FROM `test.hotels`
    WHERE LOWER(location) LIKE LOWER(CONCAT('%', @location, '%'))
    ORDER BY price_per_night ASC;

search-available-hotels:
  kind: bigquery-sql
  source: my-bigquery-source
  description: Obtener todos los hoteles disponibles (no reservados)
  statement: |
    SELECT
      id,
      name,
      location,
      price_per_night,
      available_rooms,
      rating
    FROM `test.hotels`
    WHERE booked = FALSE
    ORDER BY rating DESC
    LIMIT 10;

book-hotel:
  kind: bigquery-sql
  source: my-bigquery-source
```

```
    description: Reservar una habitación de hotel por ID
    parameters:
      - name: hotel_id
        type: integer
        description: Identificador único del hotel
        required: true
      - name: checkin_date
        type: string
        description: Fecha de check-in (AAAA-MM-DD)
        required: true
      - name: checkout_date
        type: string
        description: Fecha de check-out (AAAA-MM-DD)
        required: true
    statement: |
      UPDATE `test.hotels`
      SET
        booked = TRUE,
        checkin_date = PARSE_DATE('%Y-%m-%d', @checkin_date),
        checkout_date = PARSE_DATE('%Y-%m-%d', @checkout_date),
        updated_at = CURRENT_TIMESTAMP()
      WHERE id = @hotel_id AND booked = FALSE;

  cancel-booking:
    kind: bigquery-sql
    source: my-bigquery-source
    description: Cancelar una reserva de hotel
    parameters:
      - name: hotel_id
        type: integer
        description: ID del hotel a cancelar
        required: true
    statement: |
      UPDATE `test.hotels`
      SET
        booked = FALSE,
        checkin_date = NULL,
        checkout_date = NULL,
        updated_at = CURRENT_TIMESTAMP()
      WHERE id = @hotel_id;

  get-booking-details:
    kind: bigquery-sql
    source: my-bigquery-source
    description: Obtener información de la reserva actual
    parameters:
      - name: hotel_id
        type: integer
        description: ID del hotel
        required: true
    statement: |
      SELECT
        name,
        location,
```

```
        price_per_night,
        booked,
        FORMAT_DATE('%Y-%m-%d', checkin_date) as checkin,
        FORMAT_DATE('%Y-%m-%d', checkout_date) as checkout,
        DATE_DIFF(checkout_date, checkin_date, DAY) as nights,
        price_per_night * DATE_DIFF(checkout_date, checkin_date, DAY) as
total_cost
      FROM `test.hotels`
      WHERE id = @hotel_id;

toolsets:
  hotel-management:
    - search-hotels-by-name
    - search-hotels-by-location
    - search-available-hotels
    - book-hotel
    - cancel-booking
    - get-booking-details
```

tools.yaml - Sistema de Release Notes

```
sources:
  gcp-public-data:
    kind: bigquery
    project: complete-tube-421007

tools:
  search_release_notes_recent:
    kind: bigquery-sql
    source: gcp-public-data
    description: Obtener release notes recientes de Google Cloud de los últimos N
días
    parameters:
      - name: days_back
        type: integer
        description: Número de días hacia atrás para buscar
        default: 7
    statement: |
      SELECT
        product_name,
        description,
        published_at,
        release_note_type,
        product_version
      FROM
        `bigquery-public-data.google_cloud_release_notes.release_notes`
      WHERE
        DATE(published_at) >= DATE_SUB(CURRENT_DATE(), INTERVAL @days_back DAY)
      ORDER BY published_at DESC
      LIMIT 50;
```

```
  search_release_notes_by_product:
    kind: bigquery-sql
    source: gcp-public-data
    description: Buscar release notes para un producto específico de GCP
    parameters:
      - name: product_name
        type: string
        description: Nombre del producto GCP (ej., 'BigQuery', 'Cloud Storage')
        required: true
    statement: |
      SELECT
        product_name,
        description,
        published_at,
        release_note_type,
        product_version
      FROM
        `bigquery-public-data.google_cloud_release_notes.release_notes`
      WHERE
        LOWER(product_name) LIKE LOWER(CONCAT('%', @product_name, '%'))
      ORDER BY published_at DESC
      LIMIT 20;

toolsets:
  release_notes_tools:
    - search_release_notes_recent
    - search_release_notes_by_product
```

Variables de Entorno

Crear un archivo `.env` (agregar a `.gitignore`):

```
# Configuración de Google Cloud
GOOGLE_APPLICATION_CREDENTIALS=/ruta/a/service-account-key.json
GOOGLE_CLOUD_PROJECT=complete-tube-421007
BIGQUERY_DATASET=test
BIGQUERY_LOCATION=US

# Configuración de MCP Toolbox
TOOLBOX_PORT=5000
TOOLBOX_HOST=127.0.0.1
TOOLBOX_LOG_LEVEL=info

# Opcional: Configuración de proxy
HTTP_PROXY=
HTTPS_PROXY=
NO_PROXY=localhost,127.0.0.1
```

Integración con Claude Desktop

Ubicación del Archivo de Configuración

- **Windows**: `%APPDATA%\Claude\claude_desktop_config.json`
- **macOS**: `~/Library/Application Support/Claude/claude_desktop_config.json`
- **Linux**: `~/.config/Claude/claude_desktop_config.json`

Configuración Completa

```
{
  "mcpServers": {
    "database-toolbox": {
      "command": "D:\\repos\\mcp-toolbox\\toolbox.exe",
      "args": [
        "--tools-file",
        "D:\\repos\\mcp-toolbox\\toolsdb.yaml",
        "--stdio"
      ],
      "env": {
        "GOOGLE_APPLICATION_CREDENTIALS": "D:\\repos\\mcp-toolbox\\complete-tube-
421007-208a4862c992.json",
        "GOOGLE_CLOUD_PROJECT": "complete-tube-421007"
      }
    },
    "release-notes-server": {
      "command": "D:\\repos\\mcp-toolbox\\toolbox.exe",
      "args": [
        "--tools-file",
        "D:\\repos\\mcp-toolbox\\tools.yaml",
        "--stdio"
      ],
      "env": {
        "GOOGLE_APPLICATION_CREDENTIALS": "D:\\repos\\mcp-toolbox\\complete-tube-
421007-208a4862c992.json"
      }
    }
  }
}
```

Verificando la Integración

1. **Reiniciar Claude Desktop** completamente (cerrar y volver a abrir)
2. Buscar el **icono de martillo (🔨)** en la interfaz de chat
3. Hacer clic en el martillo para ver las herramientas disponibles
4. Probar con: "Buscar hoteles en Nueva York"

Desarrollo de Agentes con Google ADK

Instalando ADK

```
# Instalar Google ADK
pip install google-adk google-genai-toolbox

# Instalar cliente de toolbox
pip install toolbox-core
```

Agente de Reservas de Hotel

gcp-hotel-agent/agent.py:

```
from google.adk.agents.llm_agent import Agent
from toolbox_core import ToolboxSyncClient
import os

# Inicializar cliente de toolbox
toolbox = ToolboxSyncClient("http://127.0.0.1:5000")

# Cargar herramientas desde la configuración
tools = toolbox.load_toolset('hotel-management')

# Configurar el agente
root_agent = Agent(
    name="asistente_reservas_hotel",
    model="gemini-2.0-flash",
    description="Asistente de IA para búsqueda y reserva de hoteles",
    instruction="""
    Eres un asistente útil de reservas de hotel con acceso a una base de datos de
hoteles.
    Puedes:
    1. Buscar hoteles por nombre o ubicación
    2. Mostrar hoteles disponibles
    3. Hacer reservas con fechas de check-in/check-out
    4. Cancelar reservas existentes
    5. Proporcionar detalles de reserva y calcular costos

    Siempre confirma los detalles de la reserva antes de hacer una reservación.
    Calcula los costos totales al mostrar información de reserva.
    Sé útil y sugiere alternativas si los hoteles solicitados no están
disponibles.
    """,
    tools=tools,
    temperature=0.7,
    max_tokens=2048
)
```

Agente de Release Notes

gcp-releasenotes-agent-app/agent.py:

```
from google.adk.agents.llm_agent import Agent
from toolbox_core import ToolboxSyncClient
import logging

# Configurar logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Inicializar conexión
toolbox = ToolboxSyncClient("http://127.0.0.1:5000")

# Cargar herramientas de release notes
tools = toolbox.load_toolset('release_notes_tools')

root_agent = Agent(
    name="analista_release_notes_gcp",
    model="gemini-2.0-flash",
    description="Experto en release notes y actualizaciones de Google Cloud
Platform",
    instruction="""
    Eres un analista experto en release notes de Google Cloud Platform.
    Tu rol es:
    1. Proporcionar información sobre actualizaciones recientes de productos GCP
    2. Buscar release notes de productos específicos
    3. Resumir cambios importantes y nuevas características
    4. Ayudar a los usuarios a entender el impacto de las actualizaciones

    Al presentar release notes:
    - Agrupa por producto cuando muestres múltiples actualizaciones
    - Resalta cambios que rompen compatibilidad o deprecaciones
    - Menciona la fecha de lanzamiento para contexto
    - Explica términos técnicos en lenguaje simple cuando sea necesario
    """,
    tools=tools,
    response_format="markdown"
)
```

Ejecutando el Toolbox

Modo Standalone

```
# Ejecutar con archivo de herramientas específico
./toolbox --tools-file="toolsdb.yaml"

# Ejecutar como servidor HTTP
./toolbox --tools-file="toolsdb.yaml" --port=5000

# Ejecutar con logging detallado
./toolbox --tools-file="toolsdb.yaml" --log-level=debug
```

```
# Ejecutar con múltiples archivos de herramientas
./toolbox --tools-file="toolsdb.yaml,tools.yaml"
```

Modo STDIO (para Claude Desktop)

```
# Requerido para comunicación con protocolo MCP
./toolbox --tools-file="toolsdb.yaml" --stdio
```

Despliegue con Docker

```
# Dockerfile
FROM golang:1.21-alpine AS builder

WORKDIR /app
COPY . .
RUN go build -o toolbox .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/

COPY --from=builder /app/toolbox .
COPY *.yaml .
COPY service-account-key.json .

ENV GOOGLE_APPLICATION_CREDENTIALS=/root/service-account-key.json
EXPOSE 5000

CMD ["./toolbox", "--tools-file=toolsdb.yaml", "--port=5000"]
```

Construir y ejecutar:

```
docker build -t mcp-toolbox .
docker run -p 5000:5000 \
  -v $(pwd)/service-account-key.json:/root/service-account-key.json \
  mcp-toolbox
```

Pruebas y Validación

Pruebas Manuales

```
# Probar configuración de herramientas
./toolbox --tools-file="toolsdb.yaml" --validate
```

```
# Probar herramienta específica
./toolbox --tools-file="toolsdb.yaml" --test-tool="search-hotels-by-name" --
param="name=Plaza"

# Pruebas interactivas
./toolbox --tools-file="toolsdb.yaml" --interactive
```

Usando MCP Inspector

```
# Instalar MCP Inspector
npm install -g @modelcontextprotocol/inspector

# Ejecutar inspector
mcp-inspector ./toolbox --tools-file="toolsdb.yaml" --stdio
```

Script de Prueba en Python

```
import requests
import json

# Probar servidor de toolbox
def probar_busqueda_hotel():
    url = "http://localhost:5000/tools/search-hotels-by-location"
    payload = {
        "parameters": {
            "location": "Nueva York"
        }
    }

    response = requests.post(url, json=payload)
    assert response.status_code == 200

    data = response.json()
    print(f"Encontrados {len(data['results'])} hoteles en Nueva York")
    return data

# Ejecutar prueba
if __name__ == "__main__":
    resultados = probar_busqueda_hotel()
    for hotel in resultados['results']:
        print(f"- {hotel['name']}: ${hotel['price_per_night']}/noche")
```

Solución de Problemas

Problemas Comunes y Soluciones

1. Claude Desktop No Detecta el Servidor MCP

Síntomas: No aparece el icono de martillo en la interfaz de Claude

Soluciones:

```
# Verificar sintaxis JSON
python -m json.tool < claude_desktop_config.json

# Verificar rutas absolutas
realpath toolbox.exe
realpath toolsdb.yaml

# Probar toolbox manualmente
./toolbox --tools-file="toolsdb.yaml" --stdio

# Revisar logs de Claude (Windows)
type %APPDATA%\Claude\logs\mcp.log
```

2. Fallos de Autenticación

Error: "No se pudieron encontrar credenciales predeterminadas"

Soluciones:

```
# Verificar clave de service account
gcloud auth activate-service-account --key-file=service-account-key.json

# Probar autenticación
gcloud auth application-default print-access-token

# Establecer credenciales explícitamente
export GOOGLE_APPLICATION_CREDENTIALS="$(pwd)/service-account-key.json"
gcloud config set project complete-tube-421007
```

3. Errores de Permisos en BigQuery

Error: "Permiso denegado en la tabla"

Soluciones:

```
-- Otorgar permisos necesarios
GRANT `roles/bigquery.dataViewer` ON SCHEMA `test`
TO "serviceAccount:mcp-toolbox-sa@complete-tube-421007.iam.gserviceaccount.com";

GRANT `roles/bigquery.dataEditor` ON TABLE `test.hotels`
TO "serviceAccount:mcp-toolbox-sa@complete-tube-421007.iam.gserviceaccount.com";
```

4. Timeouts de Conexión

Error: "Context deadline exceeded"

Soluciones:

```
# Aumentar timeout en configuración de herramientas
sources:
  my-bigquery-source:
    kind: bigquery
    project: complete-tube-421007
    timeout: 30s  # Aumentar timeout
    max_retries: 3
    retry_delay: 2s
```

5. Problemas de Serialización de Parámetros

Bug Conocido: Solo la primera llamada MCP con parámetros tiene éxito (Issue #4192)

Solución Temporal:

```
# Usar parámetros nombrados consistentemente
parameters:
  - name: hotel_id
    type: integer
    description: ID del hotel
    required: true  # Marcar parámetros requeridos
```

Modo Debug

Habilitar logging detallado:

```
# Establecer variables de entorno
export TOOLBOX_LOG_LEVEL=debug
export GOOGLE_CLOUD_ENABLE_LOGGING=true

# Ejecutar con salida de debug
./toolbox --tools-file="toolsdb.yaml" --log-level=debug --verbose 2>&1 | tee
debug.log
```

Mejores Prácticas de Seguridad

1. Gestión de Service Accounts

```
# Usar principio de menor privilegio
gcloud iam roles create mcp_toolbox_role \
    --project=complete-tube-421007 \
    --title="Rol Personalizado MCP Toolbox" \
    --permissions=bigquery.tables.getData,bigquery.tables.update

# Rotar claves regularmente
gcloud iam service-accounts keys create nueva-clave.json \
    --iam-account=mcp-toolbox-sa@complete-tube-421007.iam.gserviceaccount.com

# Eliminar claves antiguas
gcloud iam service-accounts keys delete KEY_ID \
    --iam-account=mcp-toolbox-sa@complete-tube-421007.iam.gserviceaccount.com
```

2. Almacenamiento Seguro de Configuración

```
# Encriptar archivos sensibles
openssl enc -aes-256-cbc -salt -in service-account-key.json -out service-account-
key.enc

# Usar gestión de secretos
gcloud secrets create mcp-service-account \
    --data-file=service-account-key.json

# Acceder en la aplicación
gcloud secrets versions access latest --secret=mcp-service-account
```

3. Seguridad de Red

```
# Configurar VPC Service Controls
sources:
  secure-bigquery:
    kind: bigquery
    project: complete-tube-421007
    vpc_service_control:
      perimeter: projects/12345/accessPolicies/policy/servicePerimeters/perimeter
    private_ip: true
```

4. Logging de Auditoría

```
-- Habilitar logs de auditoría de BigQuery
CREATE OR REPLACE TABLE `audit.mcp_toolbox_logs` AS
SELECT
  timestamp,
  protoPayload.authenticationInfo.principalEmail as email_usuario,
  protoPayload.methodName as operacion,
```

```
  protoPayload.resourceName as recurso,
  protoPayload.request as detalles_solicitud
FROM
  `complete-tube-421007.cloud_audit_logs.data_access`
WHERE
  protoPayload.serviceName = 'bigquery.googleapis.com'
  AND protoPayload.authenticationInfo.principalEmail LIKE '%mcp-toolbox%';
```

Ejemplos

Ejemplo 1: Flujo de Búsqueda y Reserva de Hotel

```
# Flujo de trabajo completo de reserva
async def flujo_reserva_hotel():
    # Buscar hoteles
    hoteles = await toolbox.execute_tool(
        "search-hotels-by-location",
        {"location": "Miami Beach"}
    )

    # Seleccionar un hotel
    hotel_seleccionado = hoteles[0]

    # Hacer reserva
    resultado_reserva = await toolbox.execute_tool(
        "book-hotel",
        {
            "hotel_id": hotel_seleccionado["id"],
            "checkin_date": "2024-12-25",
            "checkout_date": "2024-12-30"
        }
    )

    # Obtener confirmación
    detalles = await toolbox.execute_tool(
        "get-booking-details",
        {"hotel_id": hotel_seleccionado["id"]}
    )

    return detalles
```

Ejemplo 2: Consulta Multi-Base de Datos

```
# Configuración avanzada multi-fuente
sources:
  bigquery-prod:
    kind: bigquery
    project: proyecto-produccion
```

```
  cloudsql-analytics:
    kind: postgres
    host: ${CLOUDSQL_HOST}
    database: analytics

  firestore-cache:
    kind: firestore
    project: complete-tube-421007
    database: cache

tools:
  busqueda-combinada:
    kind: custom
    sources:
      - bigquery-prod
      - cloudsql-analytics
      - firestore-cache
    description: Buscar en todas las fuentes de datos
    implementation: |
      # Lógica personalizada para consultar múltiples fuentes
      # y combinar resultados
```

Ejemplo 3: Análisis de Tendencias con Release Notes

```
# Analizar tendencias en actualizaciones de GCP
async def analizar_tendencias_gcp():
    # Obtener release notes de los últimos 30 días
    notas_recientes = await toolbox.execute_tool(
        "search_release_notes_recent",
        {"days_back": 30}
    )

    # Agrupar por producto
    productos = {}
    for nota in notas_recientes:
        producto = nota["product_name"]
        if producto not in productos:
            productos[producto] = []
        productos[producto].append(nota)

    # Analizar frecuencia de actualizaciones
    analisis = {
        "total_actualizaciones": len(notas_recientes),
        "productos_actualizados": len(productos),
        "producto_mas_activo": max(productos, key=lambda k: len(productos[k])),
        "tipos_cambios": {}
    }

    # Contar tipos de cambios
    for nota in notas_recientes:
        tipo = nota.get("release_note_type", "General")
```

```
        analisis["tipos_cambios"][tipo] = analisis["tipos_cambios"].get(tipo, 0) +
1

    return analisis
```

Ejemplo 4: Dashboard Integrado

```
# Dashboard para monitorear reservas y actualizaciones
class MCPDashboard:
    def __init__(self):
        self.toolbox = ToolboxSyncClient("http://127.0.0.1:5000")

    async def obtener_metricas_hotel(self):
        """Obtener métricas de ocupación de hoteles"""
        # Buscar todos los hoteles
        hoteles = await self.toolbox.execute_tool(
            "search-available-hotels",
            {}
        )

        metricas = {
            "hoteles_disponibles": len(hoteles),
            "capacidad_total": sum(h["available_rooms"] for h in hoteles),
            "precio_promedio": sum(h["price_per_night"] for h in hoteles) /
len(hoteles),
            "rating_promedio": sum(h["rating"] for h in hoteles) / len(hoteles)
        }

        return metricas

    async def obtener_actualizaciones_relevantes(self):
        """Obtener actualizaciones de productos GCP relevantes"""
        productos_interes = ["BigQuery", "Cloud SQL", "Cloud Storage"]
        actualizaciones = []

        for producto in productos_interes:
            notas = await self.toolbox.execute_tool(
                "search_release_notes_by_product",
                {"product_name": producto}
            )
            actualizaciones.extend(notas[:3])  # Top 3 por producto

        return sorted(actualizaciones,
                      key=lambda x: x["published_at"],
                      reverse=True)

    async def generar_reporte(self):
        """Generar reporte consolidado"""
        metricas = await self.obtener_metricas_hotel()
        actualizaciones = await self.obtener_actualizaciones_relevantes()
```

```
        reporte = f"""
        === REPORTE DE DASHBOARD MCP ===

        MÉTRICAS DE HOTELES:
        - Hoteles Disponibles: {metricas['hoteles_disponibles']}
        - Capacidad Total: {metricas['capacidad_total']} habitaciones
        - Precio Promedio: ${metricas['precio_promedio']:.2f}/noche
        - Rating Promedio: {metricas['rating_promedio']:.1f}/5

        ACTUALIZACIONES RECIENTES DE GCP:
        """

        for act in actualizaciones[:5]:
            reporte += f"\n- [{act['product_name']}] {act['description']
[:100]}..."

        return reporte
```

Configuración Avanzada

Configuración de Múltiples Entornos

```yaml
# config/dev.yaml
environments:
  development:
    sources:
      bigquery-dev:
        kind: bigquery
        project: ${DEV_PROJECT_ID}
        dataset: dev_test

# config/prod.yaml
environments:
  production:
    sources:
      bigquery-prod:
        kind: bigquery
        project: ${PROD_PROJECT_ID}
        dataset: production
        # Configuraciones de seguridad adicionales
        use_private_ip: true
        ssl_required: true
```

Configuración de Cache

```yaml
# Configurar cache para optimizar rendimiento
cache:
  enabled: true
  type: redis
```

```
    config:
      host: ${REDIS_HOST}
      port: 6379
      ttl: 3600  # 1 hora
      max_entries: 1000

  tools:
    cached-search:
      kind: bigquery-sql
      source: my-bigquery-source
      cache:
        enabled: true
        ttl: 1800  # 30 minutos
      statement: |
        SELECT * FROM `test.hotels`
        WHERE rating > 4.0
```

Configuración de Webhooks

```
# Configurar webhooks para notificaciones
webhooks:
  booking-notification:
    url: ${WEBHOOK_URL}
    events:
      - hotel.booked
      - hotel.cancelled
    headers:
      Authorization: Bearer ${WEBHOOK_TOKEN}

tools:
  book-hotel-with-notification:
    kind: bigquery-sql
    source: my-bigquery-source
    webhooks:
      - booking-notification
    statement: |
      UPDATE `test.hotels`
      SET booked = TRUE
      WHERE id = @hotel_id
```

Monitoreo y Observabilidad

Configuración de Métricas

```
# Configurar exportación de métricas
metrics:
  enabled: true
  exporters:
    - type: prometheus
```

```
      endpoint: /metrics
      port: 9090
    - type: stackdriver
      project_id: ${GOOGLE_CLOUD_PROJECT}

  custom_metrics:
    - name: mcp_toolbox_queries_total
      type: counter
      description: Total de consultas ejecutadas
    - name: mcp_toolbox_query_duration_seconds
      type: histogram
      description: Duración de las consultas
```

Configuración de Logging Estructurado

```
import logging
import json
from pythonjsonlogger import jsonlogger

# Configurar logging estructurado
logHandler = logging.StreamHandler()
formatter = jsonlogger.JsonFormatter()
logHandler.setFormatter(formatter)
logger = logging.getLogger()
logger.addHandler(logHandler)
logger.setLevel(logging.INFO)

# Ejemplo de uso
logger.info("query_executed", extra={
    "tool": "search-hotels-by-location",
    "parameters": {"location": "Miami"},
    "duration_ms": 150,
    "results_count": 5
})
```

Dashboard de Monitoreo

```
# dashboard_monitor.py
import asyncio
import time
from datetime import datetime, timedelta

class ToolboxMonitor:
    def __init__(self):
        self.metrics = {
            "total_queries": 0,
            "successful_queries": 0,
            "failed_queries": 0,
            "avg_response_time": 0,
```

```
            "queries_by_tool": {}
        }

    async def track_query(self, tool_name, parameters, duration, success):
        """Rastrear métricas de consultas"""
        self.metrics["total_queries"] += 1

        if success:
            self.metrics["successful_queries"] += 1
        else:
            self.metrics["failed_queries"] += 1

        # Actualizar promedio de tiempo de respuesta
        current_avg = self.metrics["avg_response_time"]
        total = self.metrics["total_queries"]
        self.metrics["avg_response_time"] = (
            (current_avg * (total - 1) + duration) / total
        )

        # Rastrear por herramienta
        if tool_name not in self.metrics["queries_by_tool"]:
            self.metrics["queries_by_tool"][tool_name] = 0
        self.metrics["queries_by_tool"][tool_name] += 1

    def get_health_status(self):
        """Obtener estado de salud del sistema"""
        error_rate = (
            self.metrics["failed_queries"] / self.metrics["total_queries"]
            if self.metrics["total_queries"] > 0 else 0
        )

        if error_rate > 0.1:
            status = "CRITICAL"
        elif error_rate > 0.05:
            status = "WARNING"
        else:
            status = "HEALTHY"

        return {
            "status": status,
            "error_rate": f"{error_rate * 100:.2f}%",
            "avg_response_time_ms": f"{self.metrics['avg_response_time']:.2f}",
            "total_queries": self.metrics["total_queries"]
        }
```

Integración con CI/CD

GitHub Actions Workflow

```
# .github/workflows/mcp-toolbox-deploy.yml
name: Deploy MCP Toolbox
```

```yaml
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Setup Google Cloud
        uses: google-github-actions/setup-gcloud@v0
        with:
          service_account_key: ${{ secrets.GCP_SA_KEY }}
          project_id: ${{ secrets.GCP_PROJECT_ID }}

      - name: Test Toolbox Configuration
        run: |
          ./toolbox --tools-file="toolsdb.yaml" --validate
          ./toolbox --tools-file="tools.yaml" --validate

      - name: Run Integration Tests
        run: |
          python -m pytest tests/integration/

  deploy:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
      - uses: actions/checkout@v2

      - name: Deploy to Cloud Run
        run: |
          gcloud run deploy mcp-toolbox \
            --image gcr.io/${{ secrets.GCP_PROJECT_ID }}/mcp-toolbox:latest \
            --platform managed \
            --region us-central1 \
            --allow-unauthenticated
```

Terraform Configuration

```hcl
# infrastructure/main.tf
provider "google" {
  project = var.project_id
  region  = var.region
}
```

```
# BigQuery Dataset
resource "google_bigquery_dataset" "mcp_dataset" {
  dataset_id                  = "mcp_toolbox"
  friendly_name               = "MCP Toolbox Dataset"
  description                 = "Dataset para MCP Toolbox"
  location                    = "US"
  default_table_expiration_ms = 3600000

  labels = {
    env = "production"
    app = "mcp-toolbox"
  }
}

# Service Account
resource "google_service_account" "mcp_sa" {
  account_id   = "mcp-toolbox-sa"
  display_name = "MCP Toolbox Service Account"
}

# IAM Bindings
resource "google_project_iam_binding" "bigquery_user" {
  project = var.project_id
  role    = "roles/bigquery.user"

  members = [
    "serviceAccount:${google_service_account.mcp_sa.email}",
  ]
}

# Cloud Run Service
resource "google_cloud_run_service" "mcp_toolbox" {
  name     = "mcp-toolbox"
  location = var.region

  template {
    spec {
      containers {
        image = "gcr.io/${var.project_id}/mcp-toolbox:latest"

        env {
          name  = "GOOGLE_CLOUD_PROJECT"
          value = var.project_id
        }

        resources {
          limits = {
            cpu    = "2000m"
            memory = "2Gi"
          }
        }
      }
```

```
      service_account_name = google_service_account.mcp_sa.email
    }
  }

  traffic {
    percent         = 100
    latest_revision = true
  }
}
```

Contribuyendo

¡Damos la bienvenida a las contribuciones! Por favor sigue estas pautas:

1. Haz fork del repositorio
2. Crea una rama de feature (`git checkout -b feature/caracteristica-increible`)
3. Confirma tus cambios (`git commit -m 'Agregar característica increíble'`)
4. Empuja a la rama (`git push origin feature/caracteristica-increible`)
5. Abre un Pull Request

Configuración de Desarrollo

```
# Clonar repositorio
git clone https://github.com/tuusuario/mcp-toolbox.git
cd mcp-toolbox

# Instalar dependencias de desarrollo
pip install -r requirements-dev.txt

# Ejecutar pruebas
pytest tests/

# Ejecutar linting
black .
flake8 .

# Ejecutar pruebas de cobertura
pytest --cov=mcp_toolbox tests/
```

Guía de Estilo de Código

- **Python**: Seguir PEP 8
- **YAML**: Usar 2 espacios para indentación
- **SQL**: Usar MAYÚSCULAS para palabras clave
- **Commits**: Usar mensajes descriptivos en español o inglés

Licencia

Este proyecto está licenciado bajo Apache License 2.0 - ver el archivo LICENSE para más detalles.

Reconocimientos

- Google Cloud Platform por BigQuery y servicios en la nube
- Anthropic por la especificación del Model Context Protocol
- Google AI Development Kit por el framework de agentes
- Comunidad MCP por el desarrollo y soporte del toolbox

Soporte

Para problemas y preguntas:

- GitHub Issues: Reportar bugs o solicitar características
- Discord: Comunidad MCP
- Documentación: Docs Oficiales de MCP
- Soporte de Google Cloud: Documentación de BigQuery

Recursos Adicionales

- Codelab de MCP Toolbox con BigQuery
- Guía de Inicio de GenAI Toolbox
- Documentación de Google ADK
- Ejemplos de MCP
