**Tech Stack Document: Google Ads KPI MCP Server Prototype (Enhanced for Artifacts - Merged)**

**1. Introduction**

This document outlines the technology stack required to develop the Google Ads KPI MCP server prototype for a Windows environment. This prototype will leverage the Python Google Ads Client Library SDK to interact with the Google Ads API and the Python MCP SDK to serve data to the Claude Desktop front end, enabling business data visualizations using Claude Artifacts. This document also includes considerations for environment management using Conda, containerization, logging, and documentation.

**2. High-Level Architecture**

The system will consist of the following components:

* **Front End:** Claude Desktop (running on a Windows computer). This application will consume data exposed by the backend server via the Model Context Protocol (MCP).
* **Backend Server:** A Python-based server running on a Windows computer. This server will:
  + Use the Python Google Ads Client Library SDK to fetch Google Ads KPI data.
  + Implement the Model Context Protocol (MCP) using the Python MCP SDK to serve the fetched data to Claude Desktop, formatted for visualization within Claude Artifacts.

**3. Dependencies and Tooling (Windows)**

This section details the software and libraries required for the backend server development on a Windows operating system.

* **Programming Language:**
  + **For Now (Prototype):** Python (Version 3.10.x will be targeted for compatibility with both SDKs). Download from the official Python website: <https://www.python.org/downloads/windows/>
  + **Future Plans (Iteration & Enhancement):** Python is expected to remain the core language.
* **Python Package and Environment Manager:**
  + **For Now (Prototype):** Conda (version included with Anaconda or Miniconda). Miniconda is recommended for a minimal installation: <https://docs.conda.io/en/latest/miniconda.html>
  + **Future Plans (Iteration & Enhancement):** Conda will continue to be used for environment management.
* **Python Google Ads Client Library SDK:**
  + **For Now (Prototype):** google-ads==27.0.0. Installation via conda: conda install -c conda-forge google-ads=27.0.0 or via pip within the conda environment: pip install google-ads==27.0.0.
  + **Future Plans (Iteration & Enhancement):** Maintain the use of the official library, updating to stable versions as needed.
* **Python MCP SDK:**
  + **For Now (Prototype):** modelcontextprotocol==0.4.0. Installation via conda (if available on conda-forge) or via pip within the conda environment: pip install modelcontextprotocol==0.4.0. Check the repository for the latest recommended version.
  + **Future Plans (Iteration & Enhancement):** Stay updated with the latest stable versions of the MCP SDK.
* **Web Framework (Optional but Recommended for MCP Server):**
  + **For Now (Prototype):** Flask==3.0.0. Installation via conda (if available on conda-forge) or via pip within the conda environment: pip install Flask==3.0.0.
  + **Future Plans (Iteration & Enhancement):** Consider FastAPI for potential performance benefits in the future.
* **gRPC & Protocol Buffers:**
  + **For Now (Prototype):** These are likely dependencies of the MCP SDK and Google Ads SDK and should be managed by Conda or pip during the installation of those SDKs. We will verify this during setup.
  + **Future Plans (Iteration & Enhancement):** Ensure these dependencies remain compatible with the chosen SDK versions.
* **Development Environment:** Any suitable Python IDE or text editor (e.g., VS Code, PyCharm, Sublime Text).

**4. Credentials and API Access**

The backend server will require the following credentials to interact with the Google Ads API:

* Google Ads Client ID: Obtained through the Google Cloud Console when setting up an OAuth 2.0 application (while the prototype uses a Developer Token, this is noted for future potential OAuth implementation).
* Google Ads Basic API Access Developer Token: Generated within your Google Ads Manager account.

These credentials will need to be configured within the backend server application, likely through environment variables or a configuration file.

**5. File Structure Tree (ASCII)**

google\_ads\_kpi\_mcp\_server/  
├── .gitignore  
├── README.md  
├── requirements.txt  
├── src/  
│ ├── main.py # Main application entry point  
│ ├── google\_ads\_client.py # Handles Google Ads API interactions  
│ ├── mcp\_server.py # Implements the MCP server logic  
│ └── config.py # Configuration settings (e.g., API credentials)  
├── examples/  
│ └── mcp\_request\_example.json # Example MCP request for testing  
└── tests/  
 └── test\_google\_ads\_client.py

**6. requirements.txt**

This file lists the Python packages that need to be installed for the project, along with their specific versions to ensure consistency. While Conda will be the primary environment manager, this file serves as a record and can be used with pip within the Conda environment if needed.

google-ads==27.0.0  
modelcontextprotocol==0.4.0  
Flask==3.0.0  
protobuf # Version will be managed by google-ads and modelcontextprotocol  
grpcio # Version will be managed by google-ads and modelcontextprotocol

**7. Possible Enhancements and Considerations**

* **Environment Management:**
  + **For Now (Prototype):** Use Conda to create an isolated environment for the project's dependencies. This will prevent version conflicts with other Python projects. To create and activate a Conda environment:  
    Bash  
    conda create --name gads\_mcp\_env python=3.10  
    conda activate gads\_mcp\_env  
    Then, install the dependencies using either conda install -c conda-forge <package>==<version> or pip install -r requirements.txt within the activated environment.
  + **Future Plans (Iteration & Enhancement):** Continue using Conda for environment management. Ensure the environment configuration is well-documented for easier setup and reproducibility.
* **Containerization:**
  + **Future Plans (Iteration & Enhancement):** Consider planning for containerization using Docker. Docker can package the application and its dependencies into a portable container, making it easier to deploy and scale in different environments (including non-Windows systems) in the future. This would involve creating a Dockerfile and potentially a docker-compose.yml file.
* **Logging and Error Handling:**
  + **For Now (Prototype):** Implement basic logging within your src modules using Python's built-in logging module. This will help in tracking the application's behavior and debugging issues. Ensure proper error handling (using try-except blocks) to gracefully handle potential exceptions, such as API errors or network issues.
  + **Future Plans (Iteration & Enhancement):** Implement more robust logging using a dedicated library like Loguru. Explore centralized logging solutions for production environments. Enhance error handling to provide more informative messages and potential recovery mechanisms.
* **Documentation Updates:**
  + **For Now (Prototype):** Document the basic setup and usage in the README.md file.
  + **Future Plans (Iteration & Enhancement):** Periodically review the official documentation of each component (google-ads, modelcontextprotocol, Flask, etc.) to stay informed about the latest stable releases, potential breaking changes, and security updates. Ensure the project's documentation is comprehensive and up-to-date.

This merged and enhanced tech stack document provides a comprehensive view of the project's technical requirements, emphasizing the use of Conda for environment management and including important considerations for development best practices and future scalability, all while keeping the goal of enabling visualizations within Claude Artifacts in mind.