Publishing AI Agents on Agentspace with Google Cloud Marketplace (delivered as SaaS) v1.0

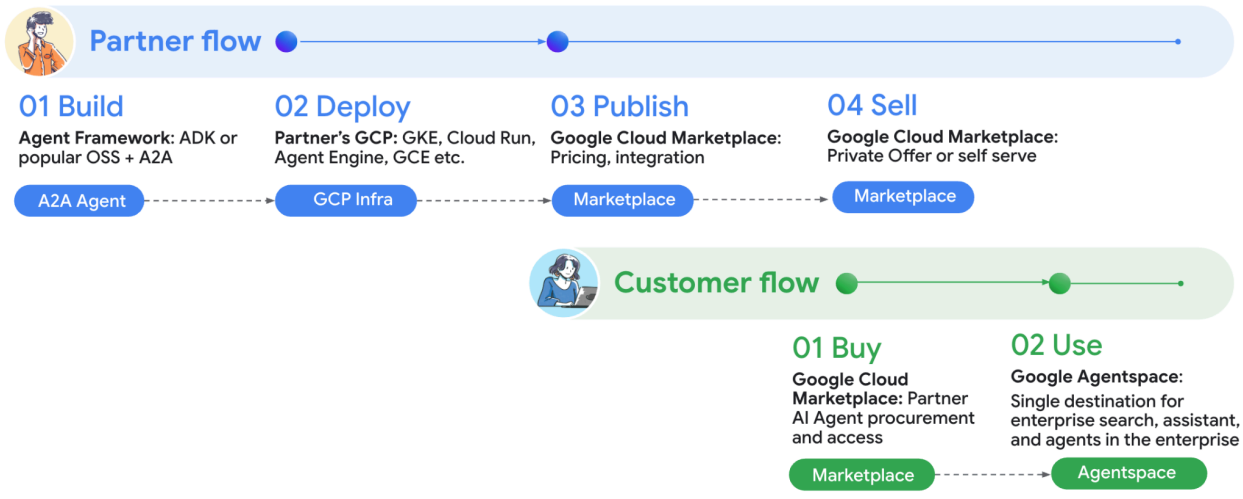
Last Updated: May 28, 2025

Welcome to Early Access!

Welcome, Google Cloud Marketplace partners! This guide provides you with the essential information and steps to develop, publish, and scale your innovative AI Agents as Software-as-a-Service (SaaS) solutions that can be integrated with Agentspace, leveraging the power and reach of the Google Cloud Marketplace to monetize your AI agents. Building the core intelligence of your AI agent is a critical first step, typically involving Google's AI tools like the **Agent Development Kit (ADK), Agent2Agent protocol,** to define behavior and integration with LLMs such as **Gemini via Vertex AI**. While this guide will point you to resources for that core agent development, its primary focus is on the subsequent journey: publishing that agent as a sellable SaaS product and integrating it with the Marketplace.

End-to-End Journey: Partner and Customer Perspectives

Before diving into the detailed steps of publishing your AI Agent, it's crucial for you, as a partner, to understand the complete lifecycle from both your perspective and that of your potential customers. This holistic view will help you align your development, integration, and support efforts for a successful Marketplace offering. The journey is designed to be seamless for customers while ensuring you have the necessary controls and information to manage your SaaS business effectively.



The Partner Journey: From Development to Operation

1. Foundation & Onboarding:
    - You'll start by becoming an official Google Cloud Partner through the Partner Advantage program and completing necessary agreements and profiles.
2. AI Agent SaaS Development:
    - You will develop your AI Agent, leveraging Google Cloud's AI tools (like ADK, Vertex AI), ensuring it's hosted on GCP and architected as a secure, multi-tenant SaaS solution.
3. Marketplace Listing & Publishing:
    - Using the Producer Portal, you'll define your product details, set your pricing strategy (free, subscription, usage-based, or combined), and critically, implement the technical integrations. This involves:
        - Setting up your backend to listen to Marketplace events (via Pub/Sub) for new subscriptions, plan changes, or cancellations.
        - Using the Partner Procurement API to approve and manage these customer entitlements.
        - If you offer usage-based pricing, accurately measuring and reporting usage to Google's Service Control API.
        - Providing frontend URLs for customer signup and login, where customers will be directed post-purchase.
    - Your solution will then undergo several reviews by Google (product details, pricing, technical integration, security, and business validation) before you can publish it.
4. Operations & Management:
    - Once live, your backend systems will continuously manage customer entitlements based on Marketplace signals.
    - You'll provide all customer and technical support for your agent.
    - You will receive monthly payouts from Google, reconciled using detailed financial reports provided by Marketplace.

The Customer Journey: Discovery to Utilization

1. Discovery & Evaluation:
    - Customers will find your AI Agent on the Google Cloud Marketplace (either the public catalog or their organization's curated private marketplace).
    - They'll review your listing details, pricing, documentation, and security validations to assess its suitability.
2. Purchase & Subscription:

- Customers purchase or subscribe to your agent directly through Marketplace, using their existing Google Cloud billing account.

3. Onboarding & Agent Activation:
   - Customers will register the procured agent in Agentspace by running CURL command.
   - Partner-Assisted Onboarding (If needed for advanced setup or if not fully automated): If the automated flow isn't fully implemented for all scenarios, or for customers needing to access advanced settings on your portal, they will be redirected to your SaaS application's signup URL. Here, they'll create an account (which your system links to their Marketplace purchase via a JWT token) and can then access your dashboard for API keys, specific A2A registration instructions, or advanced agent management.
4. Agent Utilization in Agentspace:
   - Once registered, your AI Agent becomes available in the customer's Agentspace.
   - Customer applications or Agentspace itself will then interact with your partner-hosted AI agent using the A2A protocol.
5. Subscription Management:
   - Customers manage their subscription (upgrades, downgrades, cancellations) through their standard Google Cloud Console / Marketplace interface. Your backend receives notifications of these changes to update entitlements.
6. Support:
   - For any issues or questions, customers will reach out to your support channels.

Understanding these interconnected journeys is key to building a solution that not only meets the technical requirements but also provides a smooth and intuitive experience for your customers, ultimately driving adoption and success on the Google Cloud Marketplace.

Publishing Your AI Agent

This section outlines how to build and list your AI Agent as a SaaS offering using the A2A protocol for Agentspace. The detailed steps below follow a sample application structure designed to illustrate this process for A2A agents.

Becoming a Google Cloud Marketplace Partner

Before you can publish, you need to be an official Google Cloud Marketplace partner:

1. **Join Google Cloud Partner Advantage:** If you haven't already, enroll in the Partner Advantage program.
2. **Meet Eligibility Requirements:** Ensure your company and solution meet the criteria for Marketplace vendors. This includes having a production-ready solution (even for this non-production MVP phase, the SaaS infrastructure must be sound), an enterprise-ready presence (including a professional website, sales motion, and customer support capabilities), and adherence to security best practices. Your solution must be primarily hosted on Google Cloud.
3. **Sign the Vendor Agreement:** You'll need to accept the Google Cloud Marketplace Vendor Agreement. This agreement outlines terms, including revenue share and your responsibilities.
4. **Payment Profile:** Set up a payment profile in the Google payments center to receive disbursements from sales.

Core Components of Your SaaS AI Agent Solution

Your SaaS offering will typically consist of four main, partner-hosted components:

1. **Producer Portal Setup:** Your product listing and configuration on Google Cloud Marketplace.
2. **Backend Application:** Manages customer entitlements (subscriptions) based on Marketplace purchases and reports usage to Google for billing.
3. **Frontend Application:** Handles post Marketplace purchase, provides agent access credentials (like API keys) and registration instructions to Agentspace Admin.
4. **Agent Application:** The core AI agent logic, hosted by you, responding to A2A requests from Agentspace.

(The following steps are based on a sample application structure for A2A agents. You can adapt these principles to your specific architecture, ensuring it's hosted on Google Cloud, e.g., using Cloud Run, Google Kubernetes Engine (GKE), or Google

Compute Engine.)

Step-by-Step Application Development

Prerequisites & Environment Setup

- **GCP Project:**
  - Ensure your Google Cloud project is properly configured. Follow the environment setup guide for integrated SaaS.
- **Service Account(s):**
  - Create a service account with appropriately scoped roles. For example, roles like Cloud SQL Editor (if using Cloud SQL), Pub/Sub Editor, Service Controller Consumer (for Partner Procurement API), and Service Usage Admin (for Service Control API) are used. For production, ensure least privilege.
  - Securely manage the keys for this service account (e.g., using Secret Manager).
- **Database Setup (Example: Cloud SQL for MySQL):**
  - You'll need a database (like Cloud SQL) to store user information, entitlements, and usage data.
  - **MVP Database Schema** *(this is just an example, you can use whatever schema you want to store the user, entitlements and usage in production)***:**

```
-- Table to store user data (linking your user ID to Google's procurement ID)
CREATE TABLE `saas_agent_database`.`Users` (
    Username VARCHAR(255) PRIMARY KEY,
    Password VARCHAR(255), -- Ensure this is securely hashed
    ProcurementId VARCHAR(255) UNIQUE
);

-- Table to store entitlements (subscriptions)
CREATE TABLE `saas_agent_database`.`entitlements_v2` (
    username VARCHAR(255),
    product VARCHAR(255), -- Your Marketplace product ID
    plan VARCHAR(255),    -- The plan ID purchased by the user
    private_key VARCHAR(255), -- The API key generated for the agent
    usage_reporting_id VARCHAR(255), -- From the entitlement, for usage
reporting
    PRIMARY KEY (username, product)
);

-- Table to store usage reports (pending values to report to Google)
```

```
CREATE TABLE `saas_agent_database`.`usage_reports` (
  usage_reporting_id VARCHAR(255) PRIMARY KEY,
  last_report_time VARCHAR(255),
  value_to_report VARCHAR(255), -- The accumulated usage delta
  product VARCHAR(255)
);
```

- Follow instructions to create and manage Cloud SQL databases.

Producer Portal Setup

Access the Producer Portal to create and configure your AI Agent listing.

1. **Getting Approved as a Producer:** Contact your Google Cloud representative to initiate this process.
2. **Creating Pub/Sub Subscription for Marketplace Events:**
   - Create a Pub/Sub subscription to receive notifications about customer purchases, cancellations, etc.
   - The topic is projects/cloudcommerceproc-prod/topics/YOUR_PROJECT_ID (replace YOUR_PROJECT_ID with the project ID where your Marketplace product is configured).
   - Your backend will listen to this subscription. See Pub/Sub integration details.
3. **Product Creation & Details:**
   - In the Producer Portal, create a new SaaS product.
   - Fill in all required product details (name, description, logos, documentation links, and categories). After publishing, the listing will be tagged as 'AI Agents' by Google Marketplace.
4. **Pricing:**
   - Define your pricing plans. Refer to the Marketplace SaaS pricing guide.
   - **Example Plans:** plan_free, plan_per_api_call.
   - **Advanced Agent Pricing Models:**
     - **Flat Fee:** Fixed charge, regardless of usage.
     - **Per API Call:** Charge a certain fee per agent query/answer. To configure, create a metric with a "count" reporting unit and a descriptive name like "API request".
     - **Variable Usage Per API Call:** For LLM-based agents with variable costs. Create a usage metric with "count" reporting unit, a custom unit name (e.g., "Processed Units"), and enable floating-point usage if needed. Clearly state any maximum value per query in your plan description.
   - Define the prices for your plans and metrics.
5. **Technical Integration:**

- **Billing Integration Tab:**
  - **Partner Procurement API Integration:** Ensure enabled.
  - **Cloud Pub/Sub Integration:** Confirm your subscription is set up.
  - **Service API Integration:** This refers to the Service Control API. Your service name will be like your-product-name.endpoints.YOUR_PROJECT_ID.cloud.goog.
  - Link your created Service Account to these integrations. Green checkmarks should appear next to each section.
- **Frontend Integration Tab:**
  - **Signup URL:** In case there is a frontend app for new user registration (e.g., https://[YOUR_FRONTEND_CLOUDRUN_URL]/signup). Once user accounts are created they will access the agent via Agentspace.
  - **Login URL:** URL for users to log into your service.
  - Initially, you might put placeholders and update these after deploying your frontend.

Backend App Setup (Partner-Hosted SaaS)

The following section explains how the backend app is set up for the MVP example. Please see this link for general information and codelabs on setting up backend apps.

The backend handles entitlements and usage reporting. The sample runs this as a local Python application with two threads, but for real use, deploy it to a robust GCP environment like Cloud Run or GKE.

1. **Configuration (MVP Example - config_variables.py):**
   - Set PROJECT_ID, PUBSUB_SUBSCRIPTION name.
   - Provide Cloud SQL database connection credentials (DB_CONNECTION_CREDENTIALS, DB_INSTANCE_CONNECTION_NAME).
   - REPORTING_SERVICE_NAME: Your service name from Producer Portal (e.g., your-product-name.endpoints.YOUR_PROJECT_ID.cloud.goog).
   - REPORTING_METRIC_NAME: The ID of the metric you defined for usage-based billing (e.g., your-product-name.endpoints.YOUR_PROJECT_ID.cloud.goog/api_requests).
   - PRODUCT_NAME: Your product ID from Producer Portal.
2. **Running the Backend (MVP Example):**
   - Set GOOGLE_APPLICATION_CREDENTIALS to your service account key file path.
   - Install dependencies (pip install -r requirements.txt).
   - Run the main application (python3.11 -m main).
3. **Entitlement Handling:**

- **Pub/Sub Message Consumption:**
  - The backend subscribes to the Pub/Sub topic.
  - A callback function processes incoming messages (e.g., handle_entitlement_message).
  - The MVP example uses a single-threaded executor and flow control to manage messages, re-processing unacknowledged messages after a timeout. For production, consider more robust queueing.
- **Entitlement Handler Logic (handle_entitlement_message):**
  1. Retrieves the full entitlement details using the Partner Procurement API (procurement_api_client.get_entitlement(entitlement_id)).
  2. Extracts procurement_account_id.
  3. **Checks if the customer account is registered in your system** (i.e., if the procurement_account_id exists in your Users table from a previous frontend signup).
     - If not registered, the message is **not acknowledged**, causing Pub/Sub to redeliver it later. This ensures entitlements are processed only after user signup.
  4. Processes based on event_type:
     - **ENTITLEMENT_CREATION_REQUESTED** (and state is ENTITLEMENT_ACTIVATION_REQUESTED):
       - Call self.procurement_api_client.approve_entitlement(entitlement_id) to approve.
       - The system then waits for a subsequent ENTITLEMENT_ACTIVE message.
     - **ENTITLEMENT_ACTIVE**:
       - Calls a function like __add_or_update_entitlement to write/update the entitlement in your database.
     - **ENTITLEMENT_PLAN_CHANGED**:
       - Similar to ENTITLEMENT_ACTIVE, updates the plan details in your database once the change is active.
     - **ENTITLEMENT_CANCELLED / ENTITLEMENT_DELETED**:
       - Remove or deactivate the entitlement in your database.
- **Adding/Updating Entitlements in Database (__add_or_update_entitlement):**
  1. Extracts product, plan, and usageReportingId (if present) from the entitlement. If the pricing model is flat fee based, no need to extract/store usageReportingId or report usage.
  2. If the entitlement exists for the procurement_account_id and product,

update it.
3. If new, add it to your entitlements_v2 table. **Generate a unique API key (private_key)** for the user at this point (e.g., using uuid.uuid4()).
4. Creates an initial record in your usage_reports table for this usageReportingId (even if usage is 0 initially).

4. **Usage Reporting (run_usage_reporting thread):**
   - Skip this part if the pricing model is flat fee.
   - The MVP example runs a loop (e.g., every 60 seconds).
   - Fetches all usage reports due for the PRODUCT_NAME from your usage_reports table.
   - For each report with a report_value > 0 (or even 0 to signal activity):
     - Calls the Service Control API to report the usage delta: self.serviceusage_api_client.report_usage(usage_reporting_id, product, report_start_time, report_end_time, report_value).
     - report_start_time is the last_report_time from your table.
     - report_end_time is the current time.
     - After successful reporting, updates your usage_reports table: decreases value_to_report by the amount reported and sets last_report_time to report_end_time. This "delta" approach helps prevent race conditions.
   - Adhere to best practices for usage reporting.

Frontend App Setup (Partner-Hosted SaaS)

The frontend handles user signup and displays access credentials. The example uses a Flask app deployed to Cloud Run.

1. **Configuration (Example - config_variables.py):**
   - PROJECT_ID.
   - AUDIENCE: The URL of your deployed Cloud Run frontend service (e.g., yourfrontendapp-xxxxxxxxxx-uc.a.run.app). This is crucial for JWT validation.
   - Database connection credentials.
   - FLASK_SESSIONS_SECRET_KEY: A unique secret for session management.
   - PRODUCTS_INFORMATION: A dictionary mapping your product ID to details like the agent's URL and the curl command for registration in Agentspace. The httpUri in the curl command template will be like: https://YOUR_AGENT_APP_URL/agent?user_id={USER_ID}.
2. **Deployment (Example to Cloud Run):**
   - Navigate to the frontend app directory.
   - Run gcloud run deploy --source ..
   - **Cloud Run Configuration:**
     - Set Cloud SQL connection.

    - Set the appropriate Service Account in Security settings.
    - Set instances to 1 (for Example) and allow unauthenticated traffic (as Google Marketplace will redirect users here).
  - **Update Producer Portal:** Once deployed, update the Signup URL in the Producer Portal with the actual Cloud Run URL.
3. **User Signup Logic:**
  - **POST to /signup:**
    1. Marketplace redirects users here via POST after purchase, including an x-gcp-marketplace-token (JWT) in the form data.
    2. Your app validates the JWT using utils.jwt_to_marketplace_id(jwt_token, config_variables.AUDIENCE).
    3. The decoded procurement_id is stored in the user's web session (e.g., flask.session["pending_procurement_registration"]).
    4. User is redirected to a signup form page (e.g., GET /signup_page).
  - **POST to /signup_page (after user submits form):**
    1. Retrieves procurement_id from the session.
    2. Gets username and password from the form.
    3. Calls a user registration function (e.g., client_handler.register_user(username, password, procurement_id)).
  - **User Registration Function (register_user in client_handler):**
    1. Retrieves the Marketplace account details using procurement_api_client.get_account(procurement_id).
    2. Checks the signup_approval state within the account object.
    3. If state is PENDING:
      - Calls procurement_api_client.approve_account(procurement_id). This signals to Google that the user has successfully completed signup on your site.
    4. If approval fails or account is in an unexpected state, raise an error.
    5. If successful, add the username, hashed password, and procurement_id to your Users database table.
4. **User Login and Dashboard:**
  - After successful registration/login, the user is directed to a dashboard.
  - The dashboard should query your entitlements_v2 table (for the logged-in user, linked via procurement_id then username) to display their active agent subscriptions and the corresponding private_key (API key).
  - Crucially, display the pre-formatted curl command (from PRODUCTS_INFORMATION config) filled with the user's specific USER_ID (your system's username) and their API_KEY. This command is what the

end-user runs to register your HTTP agent in their Agentspace instance.

Agent App Setup (Partner-Hosted SaaS - A2A Agent)

This component is the heart of your offering – the AI Agent itself. Building the core intelligence of your agent will typically involve leveraging Google's powerful AI tools. This includes using the **Google Agent Development Kit (ADK)** to define your agent's behavior, integrating with large language models such as **Gemini via Vertex AI**, and potentially adding custom logic and integrating external tools.

The following steps, based on the sample, outline how to structure the application service (e.g., a Flask app on Cloud Run) that hosts this agent logic. This service exposes your agent via **an A2A compliant interface** for Agentspace and integrates with your SaaS backend for essential functions like authentication and usage tracking.

For comprehensive guidance on designing and developing the core capabilities of your AI agent using the Agent Development Kit and Vertex AI, please refer to the official Agent Development Kit (ADK) documentation and the resources available for Vertex AI Agent Builder and Vertex AI models.

1. **Configuration (Sample - config_variables.py):**
    - PRODUCT_NAME (your Marketplace product ID).
    - Database connection credentials.
    - GEMINI_API_KEY (if using Gemini directly via its API, though typically ADK/Vertex AI handles model interaction). Or configure ADK/Vertex AI appropriately within your agent code.
2. **Deployment (Sample to Cloud Run):**
    - Navigate to the agent app directory.
    - Run gcloud run deploy --source ..
    - Configure Cloud Run permissions and Cloud SQL connection similar to the frontend app.
3. **Agent Request Handling (A2A Protocol Interaction):**
    1. **Authentication & Authorization:**
        - The A2A protocol will define how agent identity and user context are transmitted and verified. Your agent must implement authentication and authorization mechanisms compliant with A2A specifications, using credentials (e.g., API keys, service accounts) provisioned during the customer onboarding and agent registration process.
        - Validate these credentials against your Users and entitlements_v2 tables to ensure the requesting entity is legitimate and has an active subscription. Return appropriate A2A error responses if invalid.

2. **Record Usage (record_usage method):**
   - Query the user's plan from your entitlements_v2 table.
   - If it's a paid, usage-based plan (e.g., not plan_free), increment a usage counter (delta_value = 1 for one A2A transaction/API call) in your usage_reports table for the corresponding usage_reporting_id. The backend app will pick this up for reporting.
3. **Process Query (A2A Interaction):**
   - **Parse the incoming request according to the A2A protocol specifications.** This will likely involve structured messages rather than simple JSON text parts.
   - Convert the A2A request to the format expected by your agent logic (e.g., types.Content for ADK, or other internal representations).
   - Manage conversation sessions based on session identifiers provided through the A2A protocol. For production, use a persistent session store like Memorystore or Firestore.
   - Execute your agent logic (e.g., self.runner.run_async if using ADK). The ADK provides rich capabilities for defining your agent's instructions, personality, connecting to external tools, and selecting appropriate AI models. The sample illustrates a basic setup; consult the full ADK documentation for advanced agent design and development.
4. Return Response (A2A Interaction):
   - Format the agent's reply according to the A2A protocol specifications expected by Agentspace.

Testing the Solution

1. **End-to-End Flow:**
   - Ensure your backend, frontend, and agent applications are running/deployed.
   - In the Producer Portal, go to your product and open the "Full preview."
   - **Use a Test Billing Account** to "purchase" your agent.
   - You should be redirected to your frontend's Signup URL. Complete the signup.
   - Your backend should process the Pub/Sub messages and approve the entitlement.
   - Log in to your frontend dashboard. You should see the agent subscription and the API key.
   - Follow the A2A registration instructions provided on your dashboard to register your agent in Agentspace.
   - Test invoking the agent from Agentspace according to A2A testing procedures and protocols.
2. **Validate Plan Changes:** Test upgrades, downgrades, and cancellations.

3. **Verify Usage Reporting:** For usage-based plans, confirm that your agent app records usage and your backend app reports it correctly, and that this reflects in test billing.

Technical and Business Validation

Before your agent goes live, it will undergo reviews:

- **Testing Your Integration:** As detailed in the "Testing the Solution" step above.
- **Security Review:**
  - **General GCP Security:** Implement best practices for IAM, network security (VPCs, firewalls), data encryption, and secure credential management (use Secret Manager).
  - **AI-Specific Security:** Familiarize yourself with Google's Secure AI Framework (SAIF). Leverage tools like AI Protection in Security Command Center, Model Armor, and Sensitive Data Protection if your agent handles sensitive information.
  - **CASA Review:** For solutions that handle restricted user data, a Cloud Application Security Assessment (CASA) by a third-party firm against standards like OWASP ASVS might be required.
- **Business Review:**
  - Google will review your overall offering for enterprise-readiness, including your company website, the clarity of your solution's value proposition, and your support processes.
  - Ensure your solution aligns with the Google Cloud Well-Architected Framework where applicable.

Publishing and Support

- **Publish:** Once all reviews are passed and tests are successful, you can publish your AI Agent solution via the Producer Portal.
- **Customer Support:**
  - As per the Google Cloud Marketplace Vendor Agreement, **you (the Vendor) are solely responsible for providing all customer and technical support** for your AI Agent.
  - Your support response times and attention must be at least as urgent as those for versions of similar products you offer through other channels.
  - Clearly display support contact information on your website and in your Marketplace listing.

Scaling Your AI Agent Business

Once your AI Agent is live, focus on growth and scalability:

- **Multi-Tenancy:** Design your SaaS architecture for multi-tenancy to efficiently serve multiple customers.
  - **GKE:** Use namespaces, resource quotas, network policies, and consider tenant-per-cluster or tenant-per-node-pool models for strong isolation.
  - **Compute Engine:** Employ stateless application design, Managed Instance Groups (MIGs) for scaling, and potentially sole-tenant nodes for strict isolation if required.
  - **App Engine:** Leverage Identity Platform for tenant user management and design your application logic to segregate tenant data.
  - Isolate tenant data using separate database schemas, database instances, or row-level security, depending on your needs.
- **Scalability & Reliability:**
  - Use load balancing and auto-scaling (MIGs, GKE HPA/VPA) for your backend, frontend, and agent applications.
  - Ensure your database (e.g., Cloud SQL) is configured for high availability and can handle read/write loads (e.g., using read replicas).
  - Implement robust monitoring and alerting.
- **Monitoring & Logging:**
  - Utilize Cloud Monitoring and Cloud Logging extensively to track performance, errors, and usage patterns. Create dashboards to visualize key metrics.
- **Leverage Partner Advantage:**
  - Explore co-selling opportunities with Google.
  - Utilize partner marketing resources to promote your AI Agent.
  - Strive for specializations to showcase your expertise.
- **Reporting and Analytics:**
  - Google Cloud Marketplace provides reports on charges, usage, and disbursements. Set up delivery of these reports to a GCS bucket or Google Drive for analysis. This data is crucial for understanding your revenue, customer adoption, and usage trends.

We are thrilled to partner with you to bring innovative AI Agents to customers through Agentspace and Google Cloud Marketplace.

Appendix:

- Tutorial of deploying a containerized agent built with the Google ADK to GKE [Link]
- Host AI apps and agents on Cloud Run [Link]