# Week - 0

**Introduction** 00:05

* Alejandra Fetis, a Google Cloud architect, introduces the session on getting started with Google Cloud and building AI agents.
* The session is designed for startups and developers.

**Agenda** 00:30

* **What is Google Cloud?**
* **Google Cloud Definitions**
* **Cloud Identity Management**
* **Cloud IAM (Identity and Access Management)**
* **Administrative Tools**
* **AI with Google**
* Demos are included between sections.

**What is Google Cloud?** 01:05

* Google operates over nine services with over 1 billion users each.
* Services like Maps, YouTube, Android, Google Photos, and Chrome run on the same infrastructure as Google Cloud Platform (GCP).
* Google has extensive infrastructure:
  + Over 41 **cloud regions** and 124 **zones** (each region has at least three zones).
  + **Edge locations** for improved networking speed and latency.
  + **CDN locations** and **dedicated interconnect locations**.
  + Over 20 Google data centers.
  + Renewable energy projects within regions.
  + 33 subsea cables connecting continents.
* Google maintains its own subsea cables. 03:33
* Google is a server manufacturer. 03:53

**Shared Responsibility Matrix** 04:41

* **Software as a Service (SaaS):** Google manages almost everything; users manage access policies and content (e.g., Gmail, Drive).
* **Platform as a Service (PaaS):** Users manage usage, access policies, and content; Google handles deployment (e.g., BigQuery - serverless).
* **Infrastructure as a Service (IaaS):** Provides building blocks for users to build what they need, offering total control over identity and security.

**Google Cloud Definitions** 05:49

* **Resource Manager:** Google Cloud infrastructure is organized into a hierarchy:
  + **Organization Node:** Represents the company (domain name). Manages everything underneath.
  + **Folders:** Represent departments, teams, or products.
  + **Projects:** Where resources (VMs, storage, services) reside.
* Folders can be nested. A simple structure is Organization > Folder > Project.
* **Organizations:** Provide visibility and control over all projects and resources. Each workspace/Cloud Identity account is associated with one organization.
* **Folders:** Grouping mechanism for projects, allowing delegation of administration rights and policy inheritance.
* **Project Resource:** The base organizing entity in GCP, where resources live. Billing, permissions, credentials, and API enablement are managed at the project level.
* **Billing Account:** 10:02 Defines who pays for resources.
  + Includes a payment instrument.
  + Can be set up directly or via invoice (offline) or through a partner.
  + Resource consumption is measured by usage, time, items, or feature use.
  + Free tiers are available for some services.
* Billing accounts are associated with projects, not folders.
* Projects do not correspond to a particular geographic region. 12:21

**Cloud Identity** 12:59

* Manages user authentication and authorization.
* **Authentication:** Verifies the user's identity (e.g., using a Gmail account or an organizational account like blank.io).
* **Authorization:** Determines what the user can do (roles) within GCP, managed by Cloud IAM.
* **User Account Types:**
  + **Consumer Users:** Gmail accounts managed by Google. Not recommended for organizations.
  + **Organization-Managed Users:** Managed by the organization, allowing for auditing, control, and security. Highly recommended.
* **Consoles for User Access Management:**
  + **admin.google.com:** Manages Google Workspace (documents, Gmail) and user accounts within the organization.
  + **console.cloud.google.com:** Manages Google Cloud Platform resources, roles, and authorization.
* Users are created in Google Admin and then added to Cloud IAM on GCP.
* **Groups:** Used to manage users within admin.google.com.
* Google Admin roles manage aspects of Cloud Identity (user/group management), while Cloud IAM roles control access to GCP resources.

**Cloud IAM (Identity and Access Management)** 19:01

* GCP control panel for authorization.
* Manages access and defines who can access what within the organization.
* Adopts the principle of least privilege.
* Access can be granted to:
  + Google Cloud accounts (users)
  + Service accounts (for applications/VMs)
  + Google Groups
  + Google Workspace domains
  + Cloud Identity domains
* **Service Accounts:** Accounts not managed by a person, used to automate services or processes. Can be impersonated.
* **IAM Roles:** Define what actions a user or service account can perform on GCP resources.
* **Conditions:** Can be set on IAM rules to restrict access based on time or other factors.
* **Types of Roles:** 22:22
  + **Primitive Roles:** Legacy roles (Owner, Editor, Viewer) that span multiple services. Broad and generally not recommended.
  + **Predefined Roles:** Fine-grained access to specific services (e.g., BigQuery User).
  + **Custom Roles:** Created by combining predefined roles.
* **Service Accounts:** Represent applications or VMs. Important for AI agents.

**Administrative Tools** 25:56

* Mobile app for console access.
* Google Cloud SDK (gcloud commands) for command-line management.
* Cloud Console and Shell (a temporary VM with pre-installed SDK).
* RESTful APIs for programmatic control.
* **Google Cloud Shell:** 27:32
  + Temporary VM with persistent disk (5GB free).
  + Pre-installed with Google Cloud SDK.
  + Web preview functionality.
  + Built-in organization access.

**AI with Google** 28:41

* AI is built into Google's DNA.
* **Building Blocks:**
  + **Google AI Studio:** 29:01 Fast prototyping with Gemini APIs. Outside of GCP.
  + **Vertex AI Studio:** 29:24 Enterprise-ready AI platform within GCP. Supports custom models, privacy policies, and enterprise features.
* **Vertex AI includes:**
  + Access to Google foundational models (Gemini, Imagen, Chirp, Kodi).
  + Task-specific models (vision, translation, speech-to-text).
  + Domain-specific models (Med-PaLM, Sec-PaLM).
  + Partner models (Claude, Llama).
  + Open-source models (Gemma).
* **Google Family Models:**
  + **Gemini:** Largest and most capable AI model.
  + **Gemma:** Open model that can be downloaded and trained.

**Google Cloud Console Tour** 31:44

* **Google Cloud Setup:** Provides guided setup for proof of concept, production, or enhanced security.
* **Creating an Organization:** Important for managing accounts. Can be created through Cloud Identity or Workspace.
* **Billing:** Creating billing groups allows for easy management of billing access.
* **Resource Hierarchy:** Organization > Folder > Project.
* **APIs:** Can be enabled/disabled to control which services are used within a project.
* **Cloud Shell:** A small machine running in GCP for command-line access.
* **Gemini Cloud Assist:** 36:32 Allows asking questions about how to do things in GCP.

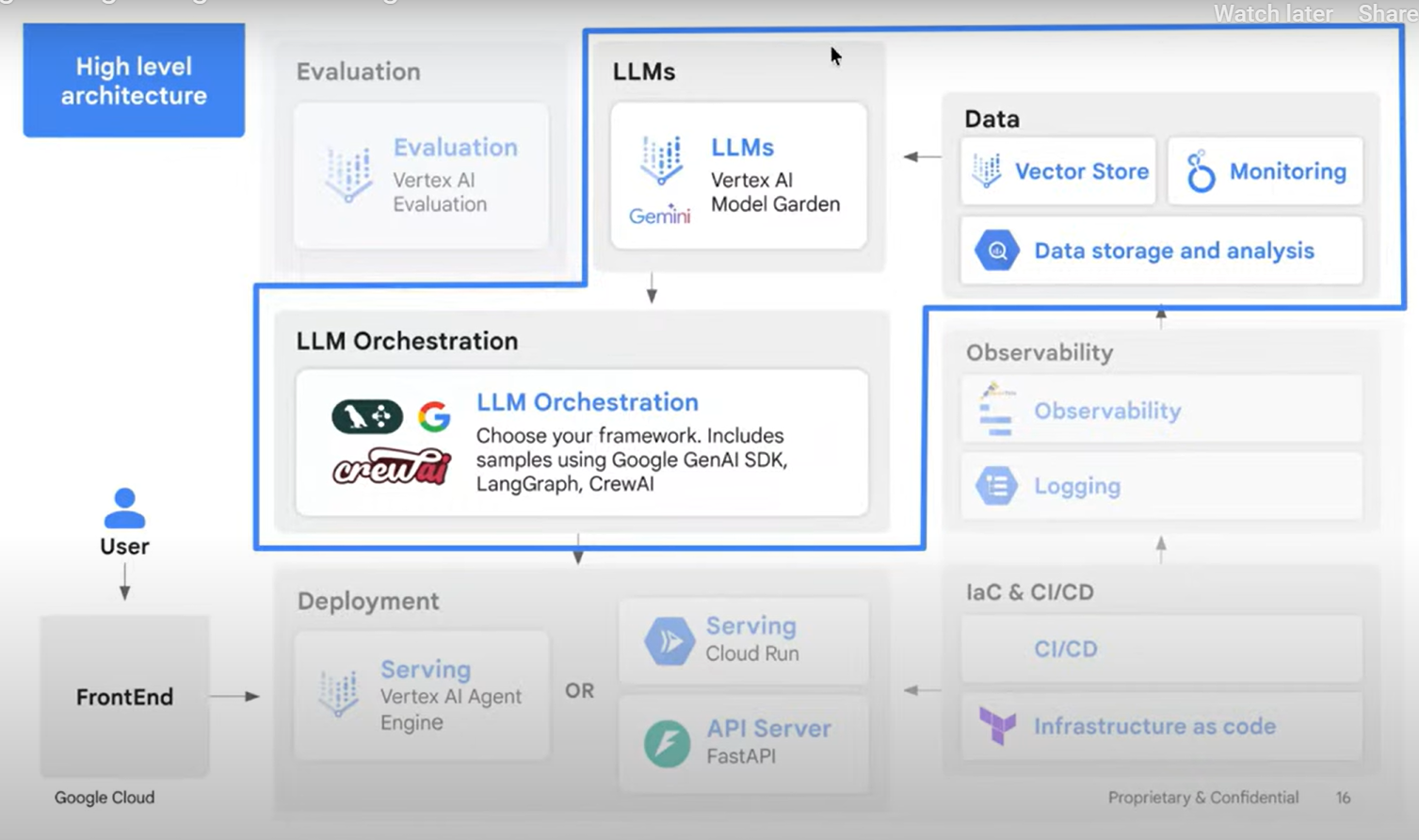
**Startup Credits** 38:03

* **Formation Tier:** $2,000 for prototyping.
* **Ecosystem Tier:** $25,000 for building an MVP and fundraising.
* **Google for Startup Scaleup Program:** $100,000 (or up to $250,000 for AI companies) in credits for first-party GCP tools.
* Marketplace purchases are explicitly excluded from the credits.
* Commitment-based contracts offer further discounts.
* QR code provided to apply for the program and connect with the startup success team.

# Week 1

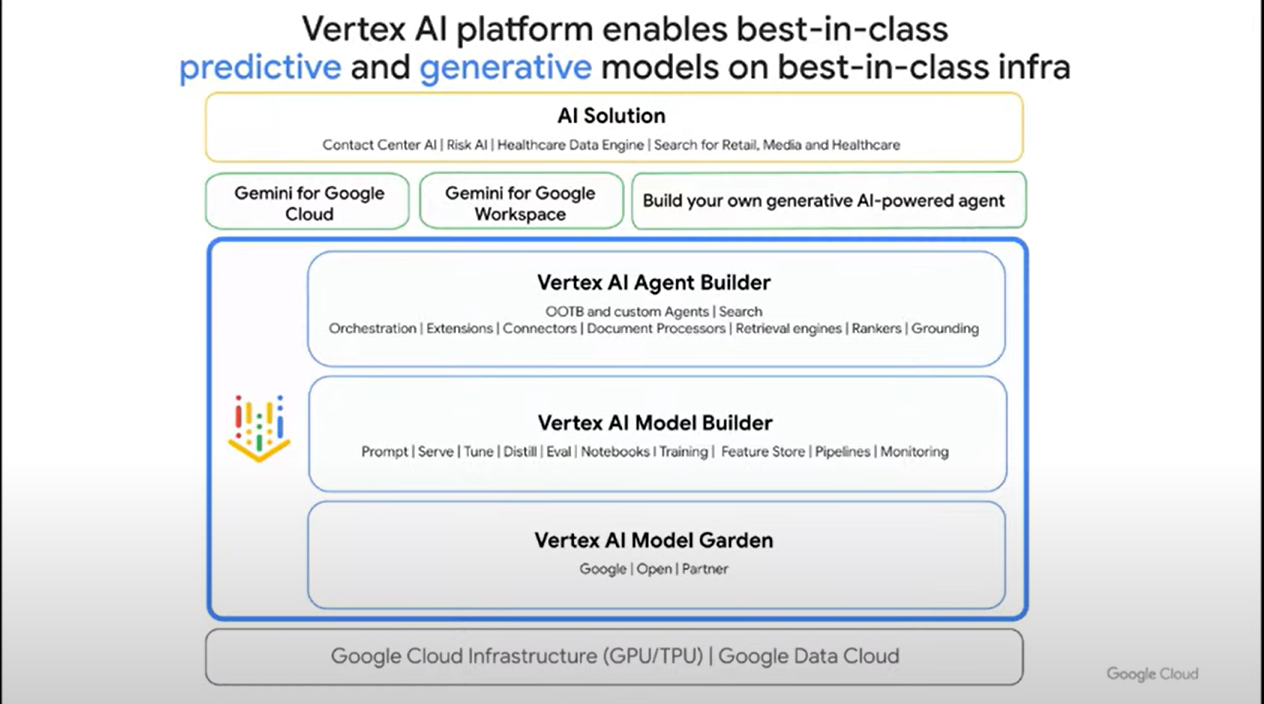
A screenshot of a computer

AI-generated content may be incorrect.

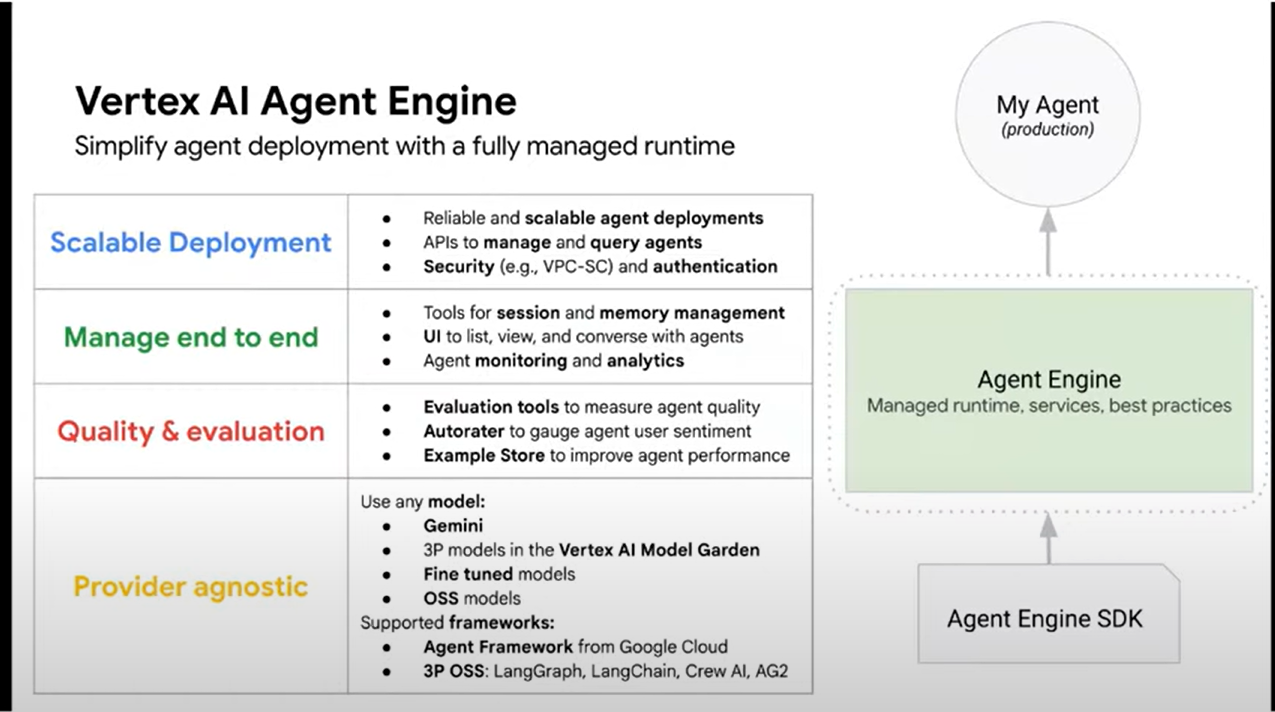


A whiteboard with text boxes

AI-generated content may be incorrect.







A screenshot of a computer

AI-generated content may be incorrect.

**Google Cloud's Role**26:41

* Google Cloud offers solutions for models, tooling, and platform.
* **Gemini** family of models.
* **ADK** (Agent Development Kit) for building agents.
* **Vertex AI** platform for building and deploying agents.

**Gemini**28:13

* A family of foundation models that are truly multimodal.
* **Gemini Pro:** Best performing model, ideal for reasoning and coding.
* **Gemini Flash:** Fast, cheap, and good for many use cases.
* AI Studio is recommended for testing Gemini.

**Gemini 2.5**32:31

* Hybrid models offering enhanced performance and reasoning.
* Useful for pair programming and generating ideas.
* Tops the LLM arena leaderboard.
* AI Studio is recommended for experimentation.

**Gemini Demo Breakdown**34:24

* Multimodal input (text, image, video, audio).
* Connects to Google Search for data retrieval.
* Features multimodal understanding and generation.
* Massive context window (1 million).
* Native connection with tools.
* Support for MCP.

**ADK (Agent Development Kit)**35:41

* Flexible and modular framework for developing and deploying agents.
* Can be used with various LLMs, not just Gemini.
* Offers tight integration with the Google ecosystem.
* Sits in the middle in terms of complexity and ease of use.
* Allows quick prototyping and building of agents.
* Code samples are available for customer service, academic research, and RAG solutions.
* Supports instructions, tools, and models.
* Supports A2A (Agent-to-Agent) protocol and MCP.

**Vertex AI**45:38

* Google's end-to-end machine learning platform.
* Covers the entire ML workflow, from data pre-processing to deployment.
* **Model Garden:** Single source for all models (Google's and open-source). Supports over 200 models.
* **Vertex AI Model Builder:** Allows prompt design, management, inference, and serving.
* **Fine-tuning:** Adapts general-purpose LLMs for specific use cases.
* **Distillation:** Transfers skills from large models to smaller models for cost savings and edge deployment.
* **Agent Builder:** Builds agents using UI or custom code. Provides access to extensions, connectors, and retrieval engines.

**Agent Engine**50:42

* Scales agents, manages APIs, and ensures security.
* Provides session management and memory management.
* Offers monitoring and logging capabilities.
* Works with various models and frameworks.
* Supports Gemini, LangChain, LangGraph, and other frameworks.
* Provides a fully managed and secure runtime environment.
* Offers context management and agent evaluation services.

**Agent Engine Architecture**53:49

* Agent Frameworks: ADK, LangGraph, LangChain, CrewAI, LlamaIndex.
* Tools: Custom-built tools, Google Cloud tools, MCP, OpenAI tools.
* Models: Any model, not just Gemini.
* Runtime Solution: Agent Engine provides a fully managed and secure environment.

# Week – 2

**Introduction**01:12

* Tim Simmon introduces the session on **Agentic AI**, focusing on the **Agent Development Kit (ADK)**, **Langchain**, and **MCP**.
* The session is part of a startup school, currently in week two, focusing on unlocking agent potential.
* Participants are encouraged to use the provided labs and notebooks for practical experience.
* Credits are available for the labs, and instructions for redeeming them are provided.
* Questions can be asked in the Q&A section, and the session will be recorded and available on demand.

**Christos' Introduction and Agent Definition**05:53

* Christos, a machine learning architect specializing in generative AI, is introduced as the speaker.
* He will cover the agentic landscape, ADK, agent deployment, and evaluation.
* An **agent** is defined as an AI system that can **reason**, **take actions**, and **use tools** to complete a task or goal. 07:06
* Agents have a **role**, a **goal**, and **memory**.
* Key components of an agent include a **model** (LLM), **tools** (APIs, code execution), **orchestration** (workflow engine), and **runtime** (state management, retries, logging).

**Multi-Agent Architectures**09:45

* **Modularity**: Breaking down complex tasks into smaller problems.
* **Specialization**: Creating specialized agents for different parts of a system (e.g., data scientist agent, researcher agent).
* **Collaboration**: Agents communicating with each other to complete tasks.
* **Scalability**: Easier to update, debug, test, and scale individual agents.

**When to Use Agents**12:20

* **Real-world integration**: When the flow involves calling APIs or using tools.
* **Multiple steps**: When achieving a goal requires multiple steps.
* **Execution path**: When the execution path is not predictable and requires dynamic decision-making.
* **Adaptation**: When the system needs to adapt based on user behavior or data.
* Examples: Study tool, refund system, factory maintenance.
* Agents are beneficial when you need to **think**, **decide**, and **act**, and the workflow changes.

**When Not to Use Agents**16:16

* **Single turns**: Stateless interactions where memory is not needed.
* **Predictable flow**: When the flow is straightforward, such as generating marketing copy.
* **Traditional Q&A systems**: Using retrieval-augmented generation (RAG) for data retrieval.
* **Heavily regulated industries**: When existing solutions are working well and changes are risky.

**Levels of Abstraction**18:11

* **Level 0: Do it yourself**: Directly using LLM APIs, requiring manual management of memory, prompts, and tools.
* **Level 1: Low-level LLM frameworks (Langchain)**: Provides some tools for building flows and managing memory, but still requires significant manual work.
* **Level 2: LangGraph**: Defines agents as nodes and edges, requiring developers to connect them.
* **Level 3: Agent Development Kit (ADK)**: Focuses on defining agents, instructions, and tools, with the framework managing the execution flow.
* **Level 4: No-code platforms**: For quick prototyping and non-technical teams.
* Higher levels offer less flexibility but speed up development.

**Agent Development Kit (ADK)**23:02

* ADK is a flexible, modular framework for developing and deploying AI agents developed by Google.
* It can be used with various LLMs, including Gemini and open-source tools.
* ADK provides a wrapper for using Langchain and CrewAI tools.
* It integrates well with the Google ecosystem and Gemini models, offering live streaming and bidirectional streaming.

**ADK Concepts**24:30

* **Conversational Context**:
  + **Session**: Corresponds to a current conversation thread and stores the history of interactions.
  + **State**: Variables stored within the session in a key-value format.
  + **Memory**: Knowledge that persists across chats.
  + **Artifacts**: Binary data (e.g., images, PDFs) identified by unique file names.

**Sessions**25:33

* Stores the history of interactions between the user and the agentic system.
* Maintained through the conversation, allowing the agent to make better decisions.
* Types of Sessions:
  + **In-memory service**: Stores session data in application memory (non-persistent, good for prototyping).
  + **Persistent session**: Allows users to resume conversations.
    - **Custom database**: Defining your own database (e.g., MySQL, SQLite).
    - **Vertex AI session service**: Managed service on Vertex AI.

**State**29:13

* Lives within the session and acts as a scratchpad for storing variables in a key-value format.
* Provides information to agents, such as user names or profiles.
* Avoid saving complex objects; use dictionaries instead.

**Memory**32:31

* Knowledge that persists across chats.
* Two types of implementations:
  + **In-memory memory service**: For prototyping.
  + **Vertex AI RAG memory service**: For long-term storage of knowledge.

**Artifacts**35:05

* Binary data (e.g., images, PDFs) identified by unique file names.
* Types of Artifact Services:
  + **In-memory artifact service**: For temporary storage.
  + **GCS artifact service**: Persistent storage on Google Cloud Storage.

**Callbacks**37:17

* Ability to intercept different stages of the flow.
* Callbacks before and after:
  + Calling an agent.
  + Calling a model.
  + Calling a tool.
* Allows controlling aspects of the flow, adding guardrails, and validating responses.

**Agent Implementation**39:04

* ADK uses a base agent (root agent) as the entry point.
* Agents are defined by:
  + **Model**: The LLM to use.
  + **Name**: For role-playing.
  + **Instruction**: What the agent is supposed to do.
* Agents are packaged into an application and run using a runner.
* User ID and application name are required to start a conversation.

**Tools**48:45

* Allow agents to connect to the outside world.
* The agent sends a request to an LLM with tool options.
* The LLM suggests a tool and arguments.
* The framework executes the tool and sends the result back to the LLM.
* Types of Tools:
  + **Built-in tools**: Provided by ADK (e.g., code creation).
  + **Third-party tools**: From Langchain and CrewAI.
  + **Function tools**: Building custom tools using Python functions.

**Model Context Protocol (MCP)**57:02

* Provides a standardized way for AI models to access and interact with external tools, data, and other systems.
* Makes tools and resources reusable by other agents.
* Context includes tools, resources (documents, prompts), etc.
* ADK has a built-in MCP client.

A screenshot of a computer program

AI-generated content may be incorrect.

**Agent as Tool**60:07

* Allows other agents to act as tools.
* The user interacts with an assistant agent, which then asks another agent in the background.
* Different from sub-agents, where there is a transfer of conversation to another agent.

**Agent with Sub-Agents**61:16

* The conversation is transferred to another agent.
* Defined using the sub\_agents variable on the agent.

**Workflows**66:15

* Using workflows to make agents behave in a specific way.

A screenshot of a computer

AI-generated content may be incorrect.

* Types of Workflows:
  + **Sequential agent**: Runs agents in a specific order.
  + **Parallel agents**: Runs multiple agents at once.

A diagram of a system

AI-generated content may be incorrect.

* + **Loop agent**: Runs a sequence of agents repeatedly.

A diagram of a loop agent

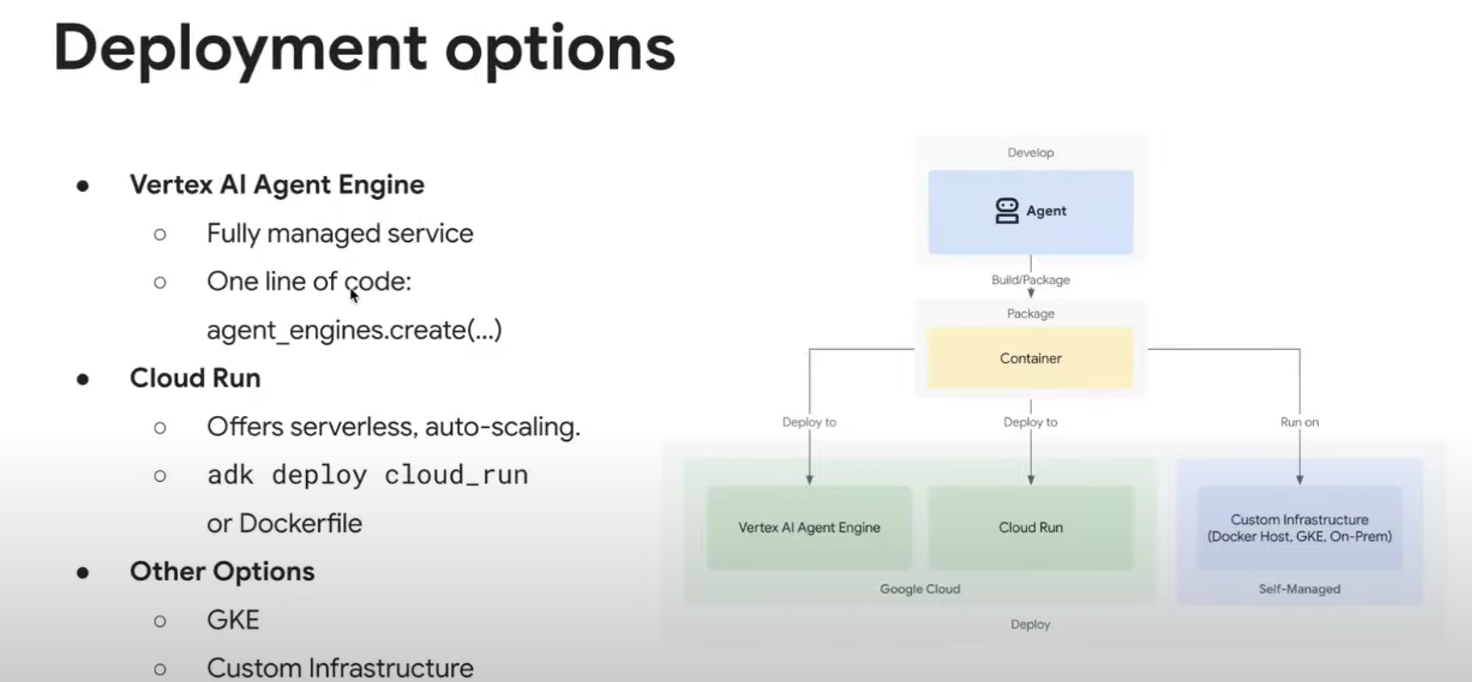
AI-generated content may be incorrect.

**ADK Web and Evaluations**74:24

* Built-in tool to ADK that provides a web interface for interacting with agents.
* Helps with evaluation and tracing.
* Allows saving conversations as evaluation sets and running them to ensure consistent performance.

**Deployments**77:24

* Options for deploying agents:
  + **Vertex AI agent engine**: Fully managed service.
  + **Cloud Run**: Deploying the agent as a service.
  + **GKE or custom infrastructure**.



**A2A (Agent-to-Agent)**79:06

* Open protocol to handle agency collaboration.
* Allows agents to talk over standard protocols with other agents.
* ADK provides native support for A2A.

**Recap**80:32

* The session covered agents, multi-agent architectures, when to use agents, building agents with ADK, evaluation, deployment, and A2A.

**Q&A**81:19

* Discussion on when to use ADK versus simpler solutions like Langchain and RAG.
* Evaluation of agent decisions and ensuring quality in critical processes.
* Customizing metadata for events.
* Possibility of using Firebase storage for persistent storage.

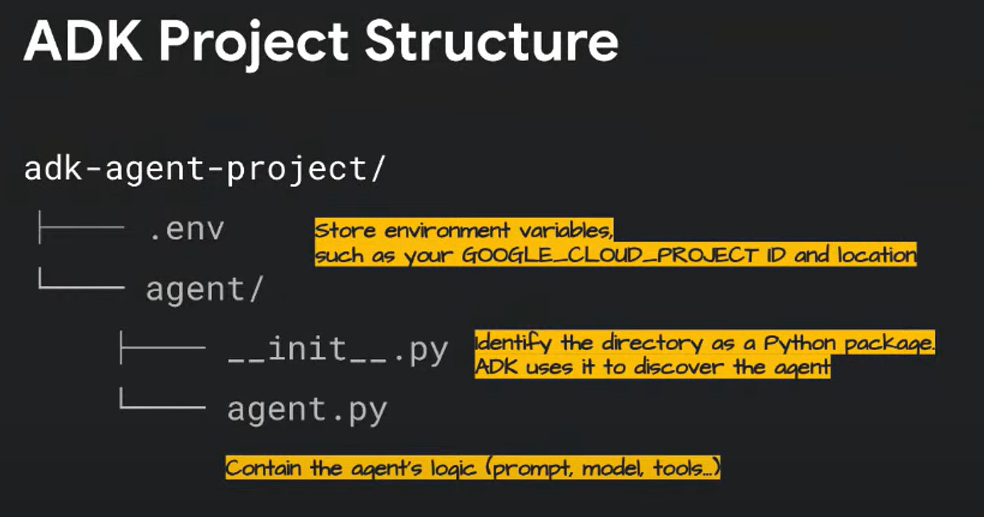
# Week – 3

A diagram of a stack

AI-generated content may be incorrect.

**Introducing Pierre and ADK**05:40

* Pierre Epolito is introduced to demonstrate the practical application of ADK for building agents.
* The session will cover:
  + **ADK foundations**
  + **Tool integration**
  + **Evaluation**
  + **Building multi-agent systems**
  + **Deployment**
  + **Introduction to the Agent-to-Agent protocol**



**Challenges in Building AI Agents**06:34

* **Fragmented landscape:** Constantly evolving tools, models, and frameworks.
* **Integration difficulties:** Managing dependencies and delegating tasks effectively.
* **Lack of operations and governance:** Moving from prototypes to reliable, scalable production systems.

**Google's Solution: A Cohesive Stack**07:56

* A standardized toolkit for efficient and repeatable development.
* Protocols like **Model Context Protocol (MCP)** and **Agent-to-Agent Protocol**.
* An agent platform for handling deployments and operations.

**Key Components of the Stack**08:42

* **Agent Development Kit (ADK):** Open-source framework for building, evaluating, and deploying agents.
* **Model Context Protocol (MCP):** Open protocol for providing context (files, data) to LLMs.
* **Vertex AI Agent Engine:** Managed serverless platform for running and scaling AI agents in production.
* **Agent-to-Agent Protocol:** Open standard for communication between different agents.

**ADK Foundations: Crash Course**10:33

* **Easy-to-use interface** for quick starts.
* **Modular and flexible** design.
* **Instant local testing** for rapid iteration.
* **Native support for multimodal workloads.**
* **Deployment-ready** from day one.

**Core Concepts**11:18

* **LLM Agent:** The "brain" using an LLM to think and act. Key parameters: name, model, instructions, and tools.
* **Tools:** Agent skills, implemented as Python functions. ADK handles schema generation.
* **Runner:** Executes the agent (e.g., ADK run, ADK web).
* **Session State:** Key for memory. A Python dictionary for passing data between turns or tools.

**ADK Project Structure**13:23

* Use ADK init to create a project structure.
* Agents reside in an "agent" folder.
* \_\_init\_\_.py indicates a Python package.
* .env file handles environment variables (e.g., project ID).

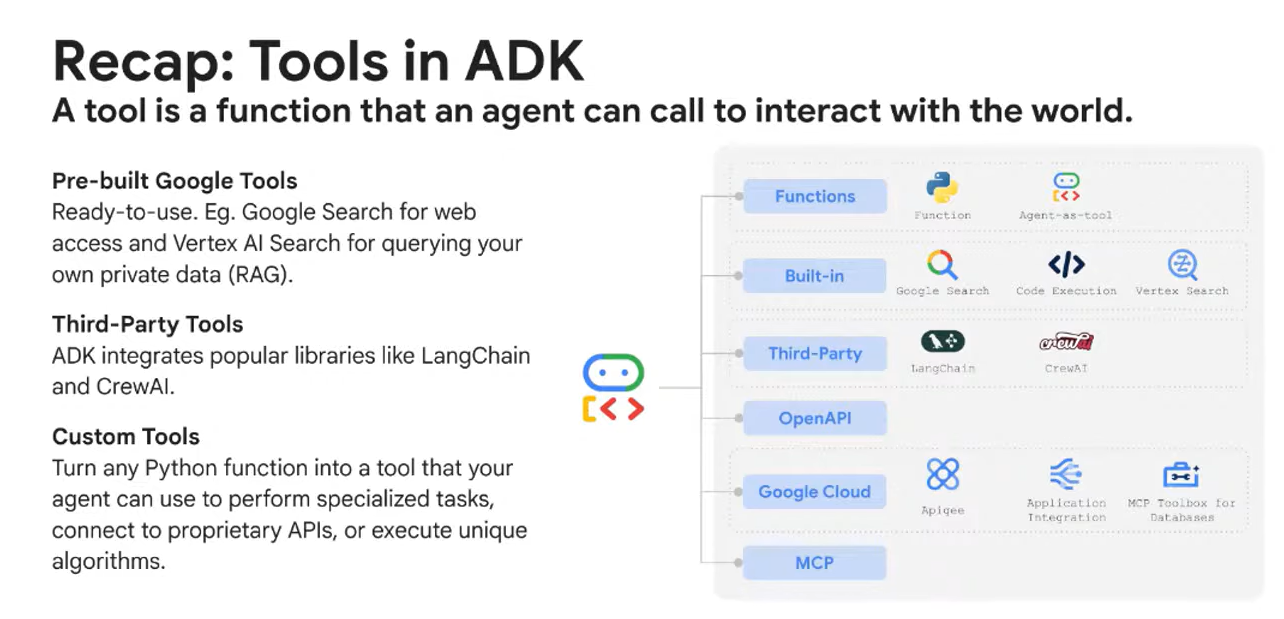
**Ways to Run Agents**13:30

* **Dev UI (ADK web):** Web interface for visual debugging, chatting, and inspecting traces.
* **Programmatic Execution:** Integrating the agent into a Python script for background tasks.
* **CLI (ADK run):** Text-based functional testing for quick verification.

**Demo: Single Agent**15:45

* Demonstrates creating a simple agent with a name, model (Gemini Flash), description, and instructions.
* Uses ADK web to run the agent and interact with it through a chat interface.
* Shows how to inspect events, state, and sessions for debugging.

**Tool Integration**20:53



* Tools are functions that agents can call to interact with the world.
* ADK provides:
  + Pre-built Google tools (e.g., Google Search, Vertex AI Search).
  + Integrations for third-party libraries (e.g., Langchain, CrewAI).
  + Custom tools (any Python function).

**Agent Tool Lifecycle**22:52

1. Agent receives a request.
2. LLM reasons and selects a tool.
3. ADK executes the tool (Python function).
4. Results are passed back to the agent.
5. Agent interprets results and generates a response.

**Creating Custom Tools**23:28

* Add a docstring to the Python function to describe its purpose, arguments, and return values.
* Use the tool\_context parameter to access the session state dictionary for memory.

**Demo: Single Agent with Tools**25:16

* Shows how to add tools to an agent, including custom tools (JSON validator) and pre-made tools from Langchain, CrewAI, and MCP.
* Demonstrates using the agent with tools like archive search, web scraping, and Reddit interaction.
* Highlights the agent's ability to combine and match different tools.
* Shows the agent graph for visualizing agent and tool interactions.

**Model Context Protocol (MCP)**31:50

* Standardizes how AI applications interact with external systems (file systems, databases).
* Allows providing context to agents in a standard way.
* ADK integrates seamlessly with MCP, allowing the use of existing MCP servers as tools or exposing ADK tools via MCP.

**Evaluation**33:52

A close-up of words

AI-generated content may be incorrect.

* Evaluating agents involves assessing:
  + Quality of response
  + Reasoning and planning
  + Tool usage
  + Self-reflection
  + Memory
  + Safety
  + Operational metrics
* ADK supports defining test cases in test.json (unit tests) and eval\_set.json (multi-turn integration tests).
* Evaluation can be run through the visual UI, programmatically with pytest, or with the ADK eval command.

A close-up of a document

AI-generated content may be incorrect.

**Evaluation Metrics**36:27

* **Tool trajectory:** Average score.
* **Response match score:** Checks the final answer.

**Demo: Evaluation**38:06

* Demonstrates creating test cases in test.json and eval\_set.json.
* Shows how to run evaluations using ADK eval and pytest.
* Highlights the importance of testing agents to ensure they meet minimum standards.

**Multi-Agent Systems**46:12

* Multi-agent systems can improve performance compared to single agents.
* ADK organizes agents in a tree-like structure for granular control.
* A root agent (manager) delegates tasks to sub-agents (specialists).
* Information is passed through a shared session state.

**Specialized Agent Types**49:55

* **Sequential Agent:** Runs sub-agents in a fixed linear order.
* **Loop Agent:** Repeatedly executes sub-agents until a condition is met (e.g., refining documents).
* **Parallel Agent:** Executes all sub-agents simultaneously (e.g., searching multiple sources).

A white and blue chart with black text

AI-generated content may be incorrect.

**Demo: Multi-Agent System**53:44

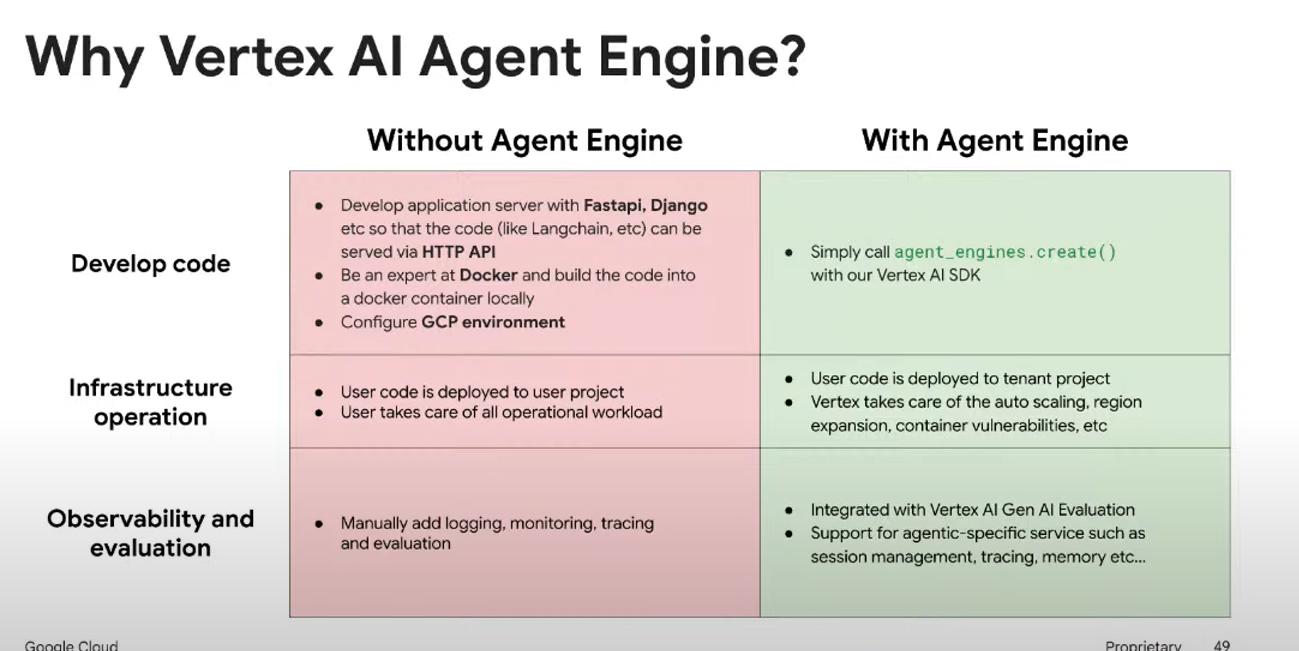
* Shows how to create a multi-agent system with a root agent and sub-agents for search and code execution.
* Demonstrates using the agent to write Fibonacci code.
* Highlights the increased complexity of multi-agent systems.

**Deployment**58:30

A screenshot of a computer

AI-generated content may be incorrect.

* Deployment options include:
  + **Vertex AI Agent Engine:** Fully managed service (one-line deployment).
  + **Cloud Run:** Serverless autoscaling (single command or Dockerfile).
  + **Google Kubernetes Engine (GKE):** More customization.
* Agent Engine simplifies deployment by handling session management, tracing, and memory.



**Demo: Deployment to Agent Engine**61:38

* Shows how to deploy an agent to Agent Engine using the Vertex AI SDK.
* Demonstrates querying the deployed agent and viewing sessions in the Agent Engine interface.

**Agent-to-Agent Protocol**64:16

* Enables communication between agents deployed by different organizations.
* Allows agents to advertise themselves, negotiate communication methods, manage tasks and state, and collaborate.

**Key Capabilities of A2A**65:31

* **Discovery:** Agents advertise themselves to clients.
* **Negotiation:** Clients and agents agree on communication methods.
* **Task and State Management:** Mechanism to communicate task changes and dependencies.
* **Collaboration:** Dynamic interaction and clarification requests.

**Building a Simple A2A System**67:37

* Agent B publishes an agent card (JSON file) with its name, description, URL, skills, and capabilities.
* A2A uses HTTPS and JSON RPC 2.0 for secure communication.
* Messages represent turns in a conversation and contain parts (text, files, data).
* An agent executor processes requests.
* Tasks represent jobs and have a lifecycle (submitted, working, input required, completed, failed).
* Streaming (Server-Sent Events) provides live progress updates.

**Demo: Agent-to-Agent System**73:51

* Shows how to build a simple A2A system with a trending topics agent, a trend analyzer agent, and a host agent.
* Demonstrates connecting to the remote client and starting communication.

**Recap and Resources**76:42

* Reviews the challenges of building agents and the solutions offered by Google's stack.
* Provides links to documentation, code samples, and contact information.

**Q&A**83:04

* Discusses using ADK with RAG capabilities (Vertex AI Search).
* Explains how to author an MCP server using ADK.
* Covers key considerations for evaluating the quality of AI agents (tool trajectory, response matching).
* Explores the future of multi-agent systems and the potential for new technologies.

## DATADOG

**Introduction**00:00

* Timmon, the host, introduces the session focusing on **observing LLM applications** with Datadog.
* The session addresses the challenges of ensuring LLM applications perform well after development, covering **monitoring** and **debugging**.
* The importance of **Skills Boost labs** for practical learning is highlighted. 02:51
* Instructions are provided for accessing **Skills Boost** and redeeming credits. 03:15
* Housekeeping rules are mentioned, including the use of the **Q&A panel** for questions and the availability of session recordings. 04:00
* Pier from Datadog is introduced as the speaker. 05:43

**Datadog Overview**06:30

* LLM applications are complex and can be unpredictable, making debugging difficult.
* Datadog's LLM observability aims to help maintain control over GenAI applications.
* Datadog was founded in 2010 and has grown to serve over 30,000 customers. 07:58
* A significant portion (30%) of Datadog's revenue is invested in R&D. 08:26
* Datadog's platform started with infrastructure, logs, and traces, and has expanded to include digital experience monitoring, cloud security, and software delivery. 09:08
* The platform offers over 900 out-of-the-box integrations. 10:34
* Datadog utilizes AI/ML tools for anomaly detection, root cause analysis, and codefix suggestions. 11:06
* Datadog aims to collect information from various sources into a single pane of glass. 12:06

**LLM Observability Explained**12:45

* **Observability** goes beyond monitoring by making sense of the collected information.
* LLMs power various applications beyond chatbots, including code generation and text summarization.
* Challenges with LLM applications include:
  + **Hallucination**: Outputs that seem correct but are wrong. 14:32
  + **Complexity**: Difficulty understanding the inner workings of LLMs. 14:39
  + **Predictability**: Inconsistent behavior. 14:45
  + **Quality**: Constant evaluation of output quality and user experience. 14:52
  + **Cost**: LLMs can be expensive to run. 15:14
  + **Security and Safety**: Vulnerabilities to prompt injection and data leakage. 15:35

**How Datadog Helps**15:52

* Datadog provides an end-to-end view of the LLM chain for efficient troubleshooting.
* It captures input and output at every stage, helping pinpoint issues like misfired tools or broken RAG pipelines.
* Datadog helps improve performance by providing a unified dashboard to compare different models.
* Real-time metrics and token consumption tracking help manage LLM spending.
* Datadog boosts quality and security with out-of-the-box quality checks for failures, hallucinations, and input injection.
* Information is grouped by topic and cluster to identify areas for optimization.

**Demo of Datadog LLM Observability**18:10

* The demo showcases a chatbot called "Swagbot" built around a RAG architecture.
* The Datadog dashboard provides a quick view of model performance, error rates, duration, cost, and token usage. 19:22
* Breakdown by topic (product feedback, greetings, etc.) and product is available.
* Security features include detection of input toxicity and prompt injection. 21:09
* Evaluation metrics include failure to answer, hallucination, and sentiment output. 21:52
* Drilling down into errors reveals the concatenation of events, from user input to model response. 23:31
* Integration with APM provides a full view of the application, including logs, metrics, and infrastructure. 24:04
* Out-of-the-box quality checks identify issues like failure to answer and hallucination. 25:35
* Clustering helps identify areas needing optimization based on request patterns. 28:52

**Instrumentation**30:11

* Instrumentation is made easy with a few lines of code using Datadog's tracing library.
* The library supports auto-instrumentation, detecting libraries like Gemini and automatically identifying prompts and inputs.
* Decorators can be used for more detailed information.
* Native support for instrumentation libraries in Python and JavaScript.

**Wrap-up**32:16

* Datadog offers end-to-end visibility, easy setup, and root cause analysis.
* Native integration with observability tools and various platforms.
* Quality evaluation is key to understanding model performance.
* Support for various platforms and models, including Gemini and OpenAI.
* New features like **datasets and experiments** allow comparing different models. 33:44
* Native agentic monitoring is also available. 35:03
* Datadog supports the entire LLM development lifecycle.

**Next Steps**36:24

* Scan the QR code or visit dtg.co/cod-glm for a free Datadog trial and implementation guide.
* Datadog offers a free Pro account for startups for one year with up to $100,000 in credits. 36:56

**Q&A Highlights**41:22

* **When to start monitoring:** It's advisable to start early to build a culture of accountability, but prioritize based on resource constraints.
* **Hallucination detection for Gemini:** Datadog's hallucination detection uses LLMs (which can include Gemini) to identify hallucinations.
* **Privacy:** Datadog offers sensitive data scanning and encryption options to protect sensitive information. 47:25

**Additional Resources**39:00

* **Google for Startups Cloud Program:** Offers free credits for building AI solutions on GCP and Gemini.
* **AI Agent Handbook:** Provides practical hacks for building AI agents.
* **Discord Community:** A community for developers to share code and collaborate.

# DEPLOYMENT

**Alvin Prauda's Presentation: Deploying AI Agents with Confidence**06:53

* The presentation focuses on practical strategies for deploying AI agents. 07:20
* The core dilemma is moving agents from a local development environment to a scalable, reliable production environment. 08:23

**Understanding Agent Deployment**08:44

* The primary challenge is not just running code but addressing nuanced issues.
* There's a difference between **development focus** and **deployment demands**. 09:37

**Development Focus**10:26

* Agent functionality: Correctness, performance, consistency.
* Prompt engineering: Crafting optimal prompts.
* Agent architecture: Workflow design, single vs. multi-agent.
* Data handling: Accessing databases, using MCP, external tools.
* Testing: Performance and accuracy.
* Context management: Stateful vs. stateless agents.

**Deployment Demands**13:11

* Robustness: Handling unexpected inputs, recovering from failures.
* Data access: Secure and efficient data retrieval.
* Maintainability: Easy updates, patching, and debugging.
* Security: Protecting user data, preventing unauthorized access.
* Integrations: Connecting with other systems and APIs.
* Scalability: Handling sudden traffic surges.

**The "Works on My Machine" Dilemma**15:31

* Production environments introduce complexities not fully simulated locally.
* Networking, security, scaling, and resource management are key considerations.

**Deployment Strategy: Simplicity, Cost, and Control**17:25

* Three primary factors to consider and prioritize: **Simplicity, Cost, and Control**.
* Rare to perfectly balance all three; understanding which are most critical is key.

**Simplicity**18:00

* Ease of deployment: Quick setup with minimal configuration.
* Engineer learning curve: How easily engineers can learn to deploy.
* Production standards: Built-in features for logging, monitoring, and security.

**Cost**19:19

* Total cost of ownership, including human effort.
* Learning curve: Time and effort to master the deployment platform.
* Prototype access: Quick deployment for feedback and iteration.
* Maintainability: Ongoing operational costs for updates and decommissioning.

**Control**20:39

* Granular control over the deployment environment (CPU, instances, memory, GPU).
* Specific deployment strategies (canary releases, blue/green deployments).
* Specific hardware requirements.

**High-Level Architecture**23:03

* User interacts with the front end.
* Front end connects to the deployment layer (Vertex AI Agent Engine, Cloud Run, GKE, VM).
* Deployed agent interacts with the LLM orchestration layer (Google ADK, Langchain, Crew AI).
* LLM orchestration layer relies on LLMs (Gemini, Vertex AI Model Garden).
* Agent accesses data via vector stores, CloudSQL, or BigQuery.
* Observability via Cloud Logging, Cloud Trace, and monitoring solutions.
* Infrastructure as Code (Terraform) and CI/CD (Cloud Build) for orchestration.

**Cloud Run: Effortless Deployment on Scalable Infrastructure**25:41

* Cloud Run is a fully managed platform for running stateless containers.
* Combines container flexibility with serverless simplicity.
* Boosts developer productivity by abstracting infrastructure management.
* Offers per-instance concurrency control.
* Supports any programming language, library, and framework.
* Portable and interoperable with Google Kubernetes Engine (GKE).
* Two primary resource types: **Services** (for HTTP traffic) and **Jobs** (for task completion). 28:44
* Deployment options: Container image, source code, or function. 30:11

**Demo 1: Deploying an ADK-Based Agent to Cloud Run**31:38

* Demonstrates deploying a weather agent using the G-Cloud run deploy command.
* The agent uses the Gemini 1.5 Flash model and includes a custom tool for fetching weather information.
* FastAPI application acts as the entry point.
* Cloud SQL PostgreSQL database is used for session management.
* Docker file defines how the application is packaged into a container image.
* The deployment command includes flags for connecting to Cloud SQL instances and setting environment variables.

**Demo 2: Autoscaling Capabilities**43:03

* Demonstrates Cloud Run's autoscaling feature by limiting concurrency and performing load testing.
* A new revision of the agent is deployed with a concurrency limit of 10.
* A Python script using the Locust library simulates multiple concurrent users.
* The instance count graph shows Cloud Run automatically scaling out to handle the increased traffic.

**Demo 3: Traffic Management (Canary Deployment)**49:19

* Illustrates Cloud Run's built-in traffic splitting for safe deployments.
* A new version of the agent with a prompt update is deployed with 0% traffic.
* Traffic is gradually shifted to the new version, enabling a canary deployment.

**Demo 4: Cloud Trace Integration**53:55

* Demonstrates how Cloud Trace integrates with agents deployed on Cloud Run.
* Distributed tracing visualizes the flow of requests, identifying bottlenecks.
* ADK provides built-in telemetry that automatically generates insightful spans.

**Code Lab**59:34

* A code lab is available with step-by-step instructions to replay the demos.

**Beyond Deployment: Ongoing Challenges**60:36

* Deployment is a continuous process requiring ongoing attention.
* Critical questions include:
  + Is the agent working properly with multiple users?
  + What is the optimal concurrency limit?
  + Are we exceeding LLM API rate limits?
  + How easy is it to debug production issues?

**Architectural Questions for Scalability**62:58

* Will each agent instance have the same context?
* How do we handle racing conditions when multiple instances update the same user session?
* Can the database handle connections from scaled-out agent services?

**Key Recommendations**64:35

* Carefully consider AI agent context and session management with scalability in mind.
* Always test your deployment rigorously (integration tests, load tests, concurrency tests).
* Prioritize observability (proper logging, request/response tracing, cost visibility).
* Employ a gradual release strategy with easy rollback capabilities.
* Use semantic versioning for deployments.

**Agent Starter Pack**69:00

* The Agent Starter Pack on GitHub helps accelerate development and deployment of production-ready agents.
* Provides pre-built templates and tools, incorporating best practices.

**Google for Startups Gemini Kit**71:07

* Launched the Google for Startups Gemini Kit, a complete AI toolkit.
* Offers instant API access, cloud credits, and on-demand learning resources.
* Announcing Gemini API sprints events and the Gemini Founders Forum.

**Google for Startups Cloud Program**72:42

* Startups can apply for cloud credits to build GenAI solutions.
* Support offers for AI-first and Web3 companies.

**AI Agent Handbook**74:03

* A strategic asset for sharing the vision of AI agents with teams.
* Provides 10 specific actionable use cases.

**AI Startup School Discord Channel**75:19

* Join the Discord channel to connect with other founders.