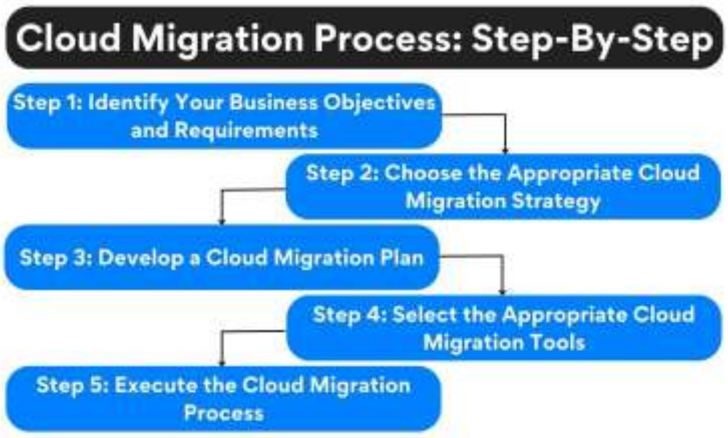
AI deployment techniques on cloud platforms (AWS, GCP), using APIs.

1. Introduction

After building and training an AI or Machine Learning model (e.g., in Python, TensorFlow, or PyTorch), the next critical step is **deployment,** making the model available for real-world use.


Cloud Migration Process: Step-By-Step
Step 1: Identify Your Business Objectives and Requirements
Step 2: Choose the Appropriate Cloud Migration Strategy
Step 3: Develop a Cloud Migration Plan
Step 4: Select the Appropriate Cloud Migration Tools
Step 5: Execute the Cloud Migration Process

Deployment means:

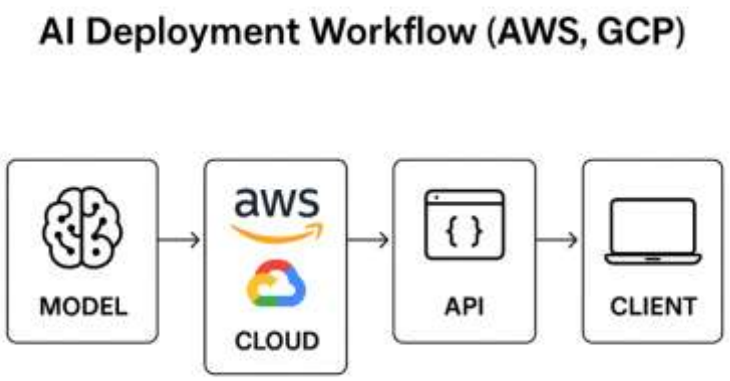
- Hosting the model on a **cloud platform** (like AWS or GCP).
- Exposing it as an **API endpoint** so that other applications (web, mobile, IoT) can send data and receive predictions.
- A **cloud platform** provides **on-demand access** to computing resources — such as **servers, storage, databases, networking, and AI tools** — over the internet. Instead of owning physical hardware, you "rent" these services from cloud providers.
- Cloud platforms allow individuals and organizations to **run applications, store data, and deploy AI models** without managing physical infrastructure.

Abbreviation	**Full Form**	**Provider**
AWS	**Amazon Web Services**	Provided by **Amazon**
GCP	**Google Cloud Platform**	Provided by **Google**

---

2. Why Deploy on the Cloud?

Advantage	Explanation
Scalability	Cloud platforms can automatically handle increasing traffic and workloads.
Accessibility	Models can be accessed globally through REST APIs.
Cost-Effectiveness	Pay-as-you-go pricing; no need for on-premise servers.
Security	Built-in encryption, IAM roles, and managed infrastructure.
Integration	Easy integration with databases, storage, monitoring, and pipelines.



3. Key Steps in AI Deployment Workflow

1. **Model Training**
   - Train the model locally or on the cloud (e.g., AWS SageMaker, GCP Vertex AI).
   - Save the trained model as `.pkl`, `.h5`, `.onnx`, etc.
2. **Model Packaging**
   - Prepare inference script (`predict.py`).
   - Define dependencies (`requirements.txt`).
   - Containerize using Docker (optional).
3. **Model Hosting**
   - Upload the model to a **cloud service** or a **container registry**.
   - Deploy it to a **managed service** (like AWS SageMaker Endpoint or GCP AI Platform Prediction).
4. **API Creation**
   - Expose the model via a REST API endpoint.
   - Integrate with backend services (Flask, FastAPI, etc.).
5. **Monitoring & Scaling**

    - Monitor latency, prediction volume, and model drift.
    - Use auto-scaling policies to manage load.

---

4. Deployment Techniques

A. Managed Services

Platforms handle the infrastructure, scaling, and API management.

Platform	**Service**	**Description**
AWS	**SageMaker Endpoints**	Managed service for training & deploying ML models.
GCP	**Vertex AI Endpoints**	Unified service for ML model deployment.

How They Work

1. **You create an account** on AWS or GCP.
2. **Choose a service** (e.g., virtual machine, model deployment, or database).
3. **Configure and deploy** it — the cloud provider manages hardware, scaling, and uptime.
4. **Access via browser or API** — applications, mobile apps, or systems can connect to it.

Example: AI Deployment

Let’s say you trained an image classifier model locally.

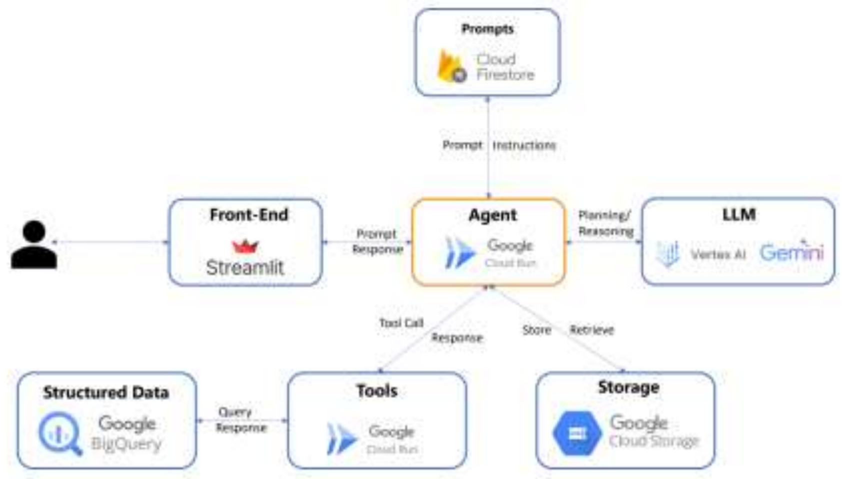
You can:

- Upload it to **AWS SageMaker** or **GCP Vertex AI**.
- Deploy it as an **API endpoint**.
- Applications send data → Cloud model returns predictions.

Example Flow:

```
User → Web App → Cloud API (AWS/GCP) → Model → Prediction
```

Deploying AI Agents on Google Cloud Platform (GCP)



AI agents are becoming essential for automating tasks, answering queries, and integrating with enterprise workflows. Google Cloud Platform (GCP) provides a robust stack for building and deploying scalable AI agents using *Cloud Run, Vertex AI, and BigQuery*.

Architecture Overview

A typical AI agent in GCP consists of:

- ***Cloud Run*** – A serverless compute platform to host the AI agent as an API, enabling automatic scaling and efficient request handling.
- ***Vertex AI*** – A managed ML platform for hosting and serving machine learning models. The agent can offload model inference to Vertex AI for better performance and model versioning.
- ***BigQuery*** – A serverless data warehouse that allows the agent to fetch real-time insights from structured data sources.
- **Firestore (for Prompt Management)** – A NoSQL database to store domain-specific prompt templates, enabling the agent to retrieve structured instructions dynamically.
- ***Front-End Interface*** – The agent can be integrated into a Streamlit, Gradio, Flask, or React-based web app for easy interaction, or connected to messaging platforms like Slack, Telegram, or a custom chatbot UI.

Implementation Best Practices

- ***Handling Authentication:*** *Use service accounts with* fine-grained IAM roles to securely connect Cloud Run with Vertex AI and BigQuery. Implement *token refresh mechanisms* to prevent authentication failures.
- ***Streaming Responses for Low Latency:*** *Instead of waiting for full model inference, structure responses using* server-sent events (SSE) or gRPC streaming, making interactions more dynamic.
- ***Optimizing Cost & Performance*:**
  - Reduce *cold starts* by using Cloud Run’s minimum instance settings.
  - Implement *caching layers* (e.g., Redis) to minimize redundant BigQuery queries.

- Use *asynchronous processing* (e.g., Cloud Tasks or Pub/Sub) for long-running tasks.

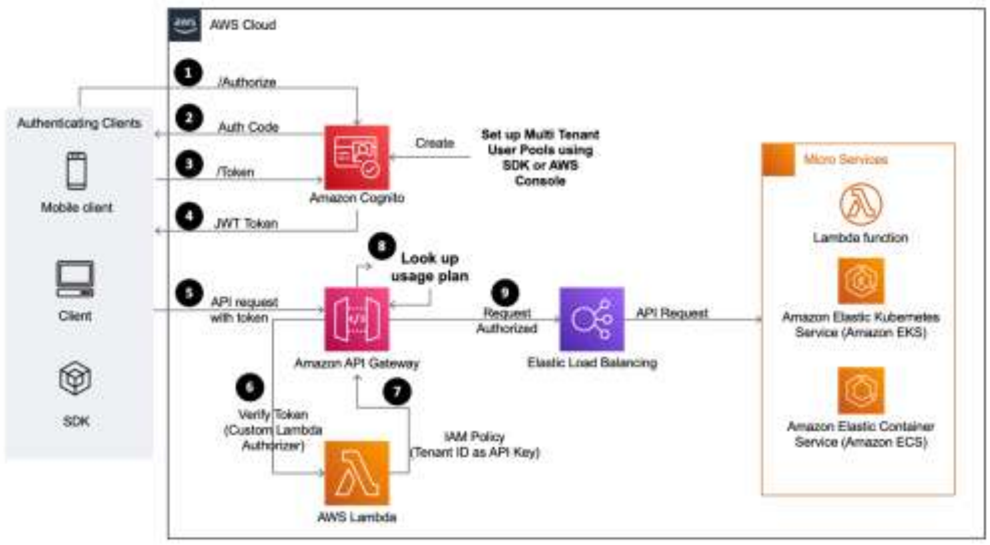
Use Cases:

AI agents built on this architecture can power:

- Virtual assistants for customer support
- Intelligent search and knowledge retrieval
- Automated data analysis and reporting

This approach enables a *scalable, cost-effective, and maintainable AI agent* that integrates seamlessly into cloud-based applications.

Deploying AI Agents on (AWS) Amazon Web services



This architecture shows the flow of user requests:

1. The client application sends a request to Amazon Cognito using the /oauth/authorize or /login API. Amazon Cognito authenticates the user credentials.
2. Amazon Cognito redirects using an authorization code grant and prompts the user to enter credentials. After authentication, it returns the authorization code.
3. It then passes the authorization code to obtain a JWT from Amazon Cognito.
4. Upon successful authentication, Amazon Cognito returns a JWT, such as acccess_token, id_token, refresh_token. The access/id token stores information about the granted permissions including tenant ID to which this user belongs to.
5. The client application invokes the REST API that is deployed in API Gateway. The API request passes the JWT as a bearer token in the API request Authorization header.
6. Since the tenant ID is hidden in the encrypted JWT token, the Lambda authorizer function validates and decodes the token, and extracts the tenant ID from the JWT.
7. The Lambda token authorizer function returns an IAM policy along with tenant ID from the decoded token to which a user belongs.
8. The application's REST API is configured with usage plans against a custom API key, which is the tenant ID in API Gateway. API Gateway evaluates the IAM policy and looks up the usage

policy using the API key. It throttles API requests if the number of requests exceed the throttle or quota limits in the usage policy.

9. If the number of API requests is within the limit, then API Gateway sends requests to the downstream application REST API. This could be deployed using containers, Lambda, or an Amazon EC2 instance.

Comparison

Feature	AWS	GCP
Market Share	Largest	Third largest
AI & ML Tools	SageMaker	Vertex AI, AutoML
Ease of Use	More complex, advanced	Simplified, user-friendly
Strengths	Variety of services, enterprise-grade	AI/ML integration, data analytics
Pricing	Pay-as-you-go	Pay-as-you-go + sustained use discounts

---

In Simple Terms

Term	Meaning
Cloud	Internet-based infrastructure and services
AWS	Amazon's cloud service for computing, AI, and storage
GCP	Google's cloud service with strong AI and data tools
API	Interface that lets apps talk to cloud-hosted models
Serverless	Run code without managing physical servers
SaaS/PaaS/IaaS	Service models defining what you control vs what the cloud manages

---

AWS (Amazon Web Services) and **GCP (Google Cloud Platform)** are leading cloud platforms that provide tools to **host, deploy, and scale applications and AI models** via APIs, containers, and serverless environments.

B. Containerized Deployment

Package your model in a **Docker container** and deploy using:

- **AWS ECS / EKS** (Elastic Container Service / Kubernetes)
- **GCP GKE** (Google Kubernetes Engine)

C. Serverless Deployment

No need to manage servers, run inference on demand:

- **AWS Lambda**
- **GCP Cloud Functions**

D. API Gateway Integration

- Use **AWS API Gateway** or **GCP API Gateway** to expose REST APIs that trigger model inference.

---

5. Model Deployment on AWS

5.1 Using AWS SageMaker

Step-by-Step Procedure

1. **Upload Model Artifacts**
    - Save model (`model.pkl`) to an **S3 bucket**.
2. **Create Model in SageMaker**
    - Specify model location (S3) and Docker image (inference environment).
3. **Deploy Model to an Endpoint**
    - Use SageMaker's **real-time endpoint** for inference.

Python Example

```
import boto3
import json

# Connect to SageMaker runtime
runtime = boto3.client('sagemaker-runtime')

# Invoke endpoint
response = runtime.invoke_endpoint(
    EndpointName='my-sagemaker-endpoint',
    ContentType='application/json',
    Body=json.dumps({"data": [5.1, 3.5, 1.4, 0.2]})
)
```

```
result = json.loads(response['Body'].read())
print(result)
```

 Output: `{'prediction': 'Iris-setosa'}`

5.2 Using AWS Lambda + API Gateway

1. Package your model and inference logic as a **Lambda function**.
2. Connect Lambda to **API Gateway** to expose an HTTP endpoint.
3. Example URL:
4. `https://xyz123.execute-api.us-east-1.amazonaws.com/predict`
5. Trigger inference via HTTP POST request.

5.3 Using Containers

- Build a Docker image with your model and inference server (Flask/FastAPI).
- Push it to **Amazon Elastic Container Registry (ECR)**.
- Deploy via **ECS** (serverless Fargate mode) or **EKS (Kubernetes)**.

---

6. Model Deployment on Google Cloud Platform (GCP)

6.1 Using Vertex AI

Step-by-Step Procedure

1. **Upload Model Artifact**
   - Store trained model in **Google Cloud Storage (GCS)**.
2. **Create a Model Resource**
   - Register your model on **Vertex AI**.
3. **Deploy to an Endpoint**
   - Vertex AI automatically creates an API endpoint for prediction.

Python Example

```
from google.cloud import aiplatform

endpoint = aiplatform.Endpoint('projects/123456/locations/us-
central1/endpoints/987654321')

response = endpoint.predict(
    instances=[{"feature_1": 5.1, "feature_2": 3.5, "feature_3": 1.4,
"feature_4": 0.2}]
)
```

```
print(response.predictions)
```

□ Output: `[ "Iris-setosa" ]`

6.2 Using Cloud Functions + Cloud Run

Option	Purpose
Cloud Functions	Deploy small models or lightweight APIs (event-driven, serverless).
Cloud Run	Deploy Docker containers for ML inference, automatically scalable.

Cloud Run Example

1. **Build Dockerfile**
2. `FROM python:3.10`
3. `COPY . /app`
4. `WORKDIR /app`
5. `RUN pip install -r requirements.txt`
6. `CMD ["python", "app.py"]`
7. **Deploy**
8. `gcloud run deploy my-ml-api --source . --platform managed --allow-unauthenticated`
9. Cloud Run generates an HTTPS endpoint automatically:
10. `https://my-ml-api-abcdef.a.run.app/predict`

6.3 Using APIs

To make predictions:

```
curl -X POST https://my-ml-api-abcdef.a.run.app/predict \
-H "Content-Type: application/json" \
-d '{"data": [5.1, 3.5, 1.4, 0.2]}'
```

Response:

```
{"prediction": "Iris-setosa"}
```

---

7. API-Based Model Deployment (General Workflow)

7.1 Architecture Diagram

```
Client → API Gateway → Cloud Function / Container → Model → Response
```

7.2 Example Using FastAPI

```
from fastapi import FastAPI
import joblib
import numpy as np

app = FastAPI()
model = joblib.load("model.pkl")

@app.post("/predict")
def predict(data: list):
    prediction = model.predict([np.array(data)])
    return {"prediction": prediction.tolist()}
```

Deploy this app:

- On AWS: using **Lambda** + **API Gateway**, or **ECS**.
- On GCP: using **Cloud Run** or **App Engine**.

---

8. Best Practices

Category	Best Practice
Model Packaging	Include dependencies (requirements.txt, Dockerfile).
Scalability	Use auto-scaling endpoints (SageMaker or Vertex AI).
Monitoring	Use CloudWatch (AWS) or Cloud Monitoring (GCP).
Security	Use IAM roles, API keys, or OAuth 2.0.
Versioning	Manage multiple model versions on endpoints.
Automation	Use CI/CD pipelines (AWS CodePipeline, GCP Cloud Build).

---

9. Comparison: AWS vs GCP for AI Deployment

Feature	AWS SageMaker	GCP Vertex AI
Model Hosting	SageMaker Endpoints	Vertex AI Endpoints
Auto ML	SageMaker Autopilot	Vertex AI AutoML
Training Jobs	Managed distributed training	Managed training + notebooks
Serverless Options	Lambda, API Gateway	Cloud Functions, Cloud Run
Containerized Deployments	ECS, EKS	GKE, Cloud Run
Monitoring	CloudWatch, SageMaker Model Monitor	Cloud Monitoring
Ease of Use	Mature ecosystem	Simplified UI & integration
Pricing	Pay-per-instance-hour	Pay-per-prediction or runtime

10. Summary

Step	AWS Technique	GCP Technique
Model Storage	S3	GCS
Deployment Service	SageMaker Endpoint	Vertex AI Endpoint
Serverless API	Lambda + API Gateway	Cloud Functions / Cloud Run
Container Deploy	ECS / EKS	GKE / Cloud Run
Monitoring	CloudWatch	Cloud Monitoring

11. Real-World Example

Use Case: Deploying a Credit Risk Prediction Model

Step	Action
Train model	Locally in Python (XGBoost)
Save model	`model.pkl`
Upload	To S3 or GCS
Deploy	Using AWS SageMaker Endpoint or GCP Vertex AI Endpoint
Create API	API Gateway (AWS) or Cloud Run (GCP)
Consume	Frontend web app sends JSON input → receives risk score

12. Key Points

- Both **AWS** and **GCP** offer **end-to-end AI pipelines** — from training to deployment.
- Models can be deployed **as APIs**, **containers**, or **serverless functions**.
- Choose platform and method based on:
  - Model size
  - Latency requirements
  - Cost considerations
  - Maintenance capabilities