Vertex AI for MLOps - Learning Path

NovaSphere Sol
02/20/26

Table of Contents

Vertex AI Pipelines

Vertex AI Pipelines is Google Cloud's managed orchestration system for ML workflows.

Instead of manually running:

- Load data
- Preprocess
- Train
- Evaluate
- Deploy

You define a DAG (Directed Acyclic Graph) that:

- Runs in managed containers
- Stores artifacts in GCS
- Tracks metadata and lineage
- Is reproducible and auditable

Core Concepts

Pipeline

A connected sequence of steps forming a DAG.
Example: Data → Train → Evaluate → Deploy.

Component

A step definition. Takes inputs and produces outputs.

Task

A runtime execution of a component.

Pipeline Run

One execution instance of a pipeline.
Each run stores:

- Logs

- Artifacts
- Metadata
- Unique ID

Artifact

Files produced by steps:

- Dataset
- Model
- Metrics

Stored in GCS and tracked by Vertex.

Pipeline Root (Staging Bucket)

GCS location where:

- Artifacts
- Intermediate outputs
- Metadata files are stored.

Service Account

Pipelines run as a service account (not your user).
It must have:

- Vertex AI access
- Storage access
- Artifact Registry access (for containers)

Phase 0 — One-Time Setup

Goal: Reliable infrastructure with proper IAM and region consistency.

1. Create Project

- Create a new GCP project.
- Choose one region (recommended: us-central1).
- Use the same region everywhere.

2. Enable APIs

- Vertex AI API
- Cloud Storage API
- IAM API

3. Create GCS Bucket

- Regional bucket
- Same region as Vertex
- Used as pipeline root

Example:

gs://<project>-vertex-pipelines

4. Create Dedicated Service Account

Do not use default Compute Engine SA.

Grant:

- Vertex AI User
- Storage Object Admin
- Storage Admin (if needed)
- Artifact Registry access

Grant bucket-level permissions explicitly.

5. Your Personal IAM

Grant yourself:

- Logs Viewer
- Monitoring Viewer

6. Development Environment

Recommended: Vertex AI Workbench.

Install:

- kfp
- google-cloud-aiplatform

- google-cloud-pipeline-components

Deliverable

A minimal “hello pipeline” runs successfully.

Phase 1 — Understand Pipeline Lifecycle

Goal: Master define → compile → run → observe.

What You Build

Simple 2–3 step pipeline:

- Generate value
- Transform value
- Print result

No ML yet.

What You Learn

- Component vs task
- DAG formation via I/O wiring
- Compilation to YAML
- Containerized step execution
- Where logs and artifacts appear

Lifecycle Model

1. Define pipeline in Python
2. Compile to YAML
3. Submit run
4. Vertex executes containers
5. Artifacts written to GCS
6. Metadata stored
7. DAG visible in UI

What to Inspect

Vertex AI → Pipelines:

- DAG graph
- Step logs
- Inputs and outputs
- Execution order

Success Criteria

You can explain:

- What a pipeline run is
- How outputs determine execution order
- What the pipeline root bucket stores
- How to inspect logs

Do not move to ML yet. First understand orchestration.

Phase 2 — Lightweight Components + Artifact I/O

Goal: Build real ML pipelines using artifacts.

Key Distinction

Parameter
Primitive values (int, string, float).

Artifact
Files tracked by Vertex:

- Dataset
- Model
- Metrics

Artifacts are written to paths provided by Vertex.

What You Build

3-step ML pipeline:

1. preprocess
   produces Dataset artifact

2. train
   consumes Dataset
   produces Model artifact

3. evaluate
   consumes Model + Dataset
   produces Metrics artifact

What You Learn

- How artifact paths are injected
- How to write/read artifact files
- Dataset / Model / Metrics tracking
- Lineage visualization
- Reproducible runs

What to Inspect

In Vertex UI:

- Dataset artifact node
- Model artifact node
- Metrics visualization
- GCS artifact paths
- Run lineage graph

Success Criteria

You can:

- Explain parameter vs artifact
- See artifacts in UI
- Re-run with different parameters
- Locate files in bucket

At this stage, you understand artifact-driven MLOps.

Phase 3 — Production Pipeline (Custom Container Path)

Goal: Run managed training via custom containers, register model, and deploy conditionally.

There are two training paths:

- Prebuilt training container (simpler)
- Custom container (production-realistic)

You will follow the custom container path.

Phase 3 Architecture

Pipeline shape:

1. Custom Training Job (Docker container)
2. Evaluation component
3. Quality gate (threshold)
4. Model upload (Model Registry)
5. Endpoint creation
6. Model deployment

Step 1 — Create Artifact Registry

Console → Artifact Registry → Create repository:

- Format: Docker
- Region: us-central1
- Name: vertex-mlops

This stores:

- Training image
- Serving image

Step 2 — Project Structure

Example layout:

phase3_custom_container/

trainer/

train.py

serving/

app.py

Dockerfile.train

Dockerfile.serve

Step 3 — Training Script (train.py)

Responsibilities:

- Read training data
- Train model
- Save model to directory provided by Vertex
- Output metrics

This runs inside a Vertex Custom Training Job.

Difference from Phase 2:
Training is no longer a lightweight component.
It is a managed Vertex job.

Step 4 — Serving Application (app.py)

Implement a minimal prediction server (e.g., FastAPI):

- Accepts prediction requests
- Loads saved model
- Returns predictions

This becomes your serving container.

Step 5 — Build and Push Images

Using Workbench terminal:

1. Configure Docker authentication
2. Build training image
3. Build serving image
4. Push both to Artifact Registry

Images look like:

us-central1-docker.pkg.dev/<project>/vertex-mlops/train:1

us-central1-docker.pkg.dev/<project>/vertex-mlops/serve:1

Step 6 — Build the Phase 3 Pipeline

Pipeline parameters:

- n_rows
- min_accuracy
- train_image_uri
- serve_image_uri

Core Components Used

From google_cloud_pipeline_components:

- Custom training job run
- Model upload
- Endpoint create
- Model deploy

Plus:

- Lightweight evaluate component

Step 7 — Quality Gate

Add parameter:
min_accuracy

Logic:

- If accuracy ≥ threshold → upload + deploy
- Else → stop

This is the core production safety mechanism.

Step 8 — Model Registry

Training output is uploaded to Vertex Model Registry.

Visible in:
Vertex AI → Models

Models are versioned automatically.

Step 9 — Endpoint Deployment

If threshold passes:

- Create endpoint (or reuse)
- Deploy model

Visible in: Vertex AI → Endpoints

What to Inspect After Run

In Vertex AI:

Pipelines:

- DAG
- Conditional branch execution
- Step logs

Training:

- Custom job logs

Models:

- Registered model version

Endpoints:

- Active deployment

GCS:

- Artifacts under pipeline root

Phase 3 Success Criteria

You are done when:

- Training ran as a Vertex Custom Job
- Docker image was used successfully
- Model appears in Model Registry
- Accuracy metric is logged
- Deployment is gated by threshold
- Model deploys to endpoint when threshold passes

Execution Order

Follow in sequence:

1. Intro to Vertex Pipelines codelab
2. Lightweight components + artifact I/O
3. Conditionals in pipelines
4. Custom training with GCPC
5. Model upload and deployment
6. Inspect lineage and metrics

By the end of Phase 3:

- You understand orchestration

- You use artifact-driven pipelines
- You run managed training jobs
- You build and push containers
- You register models
- You gate production deployments

You are operating a structured MLOps system, not a notebook workflow.