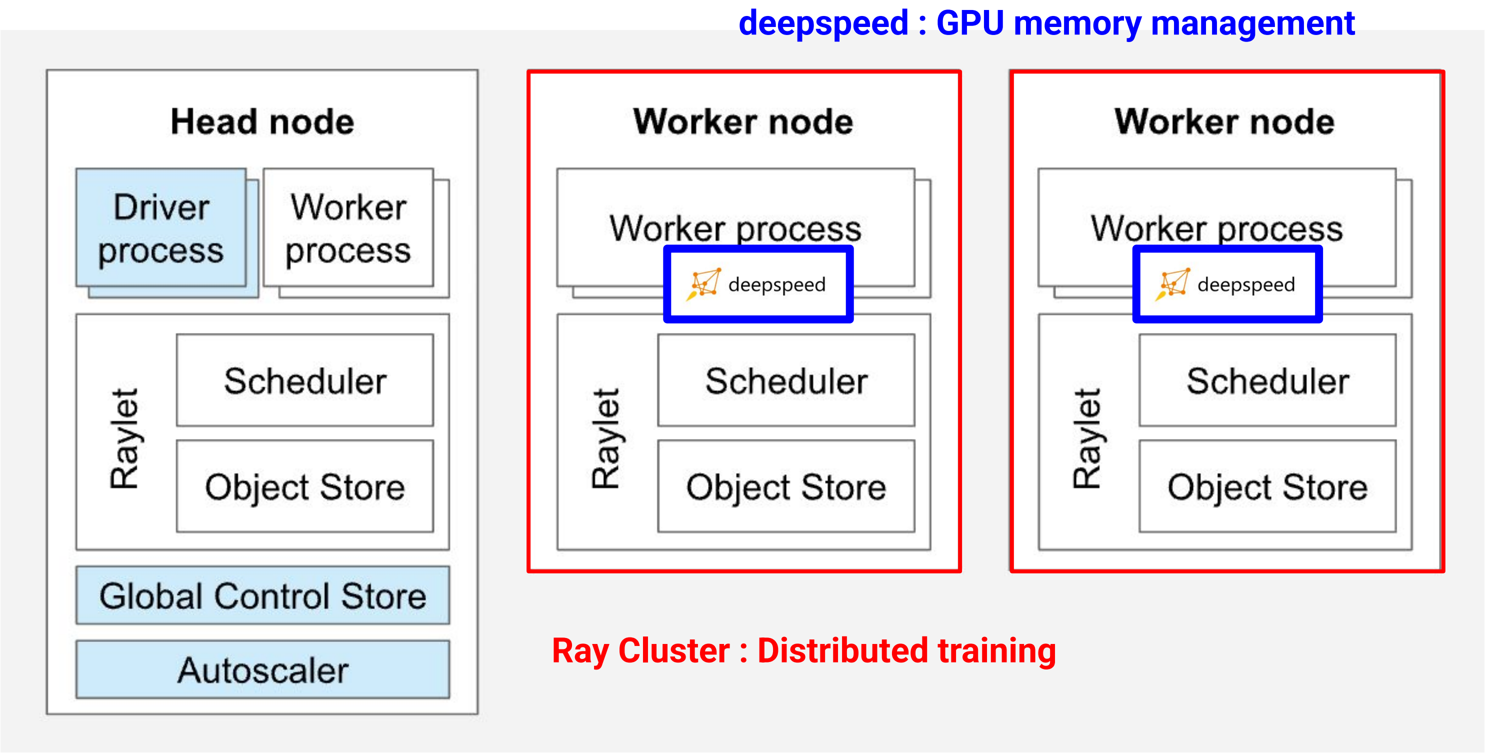
LLM Training on Vertex AI

Hangsik Shin
AI Specialist
Google Cloud

Agenda

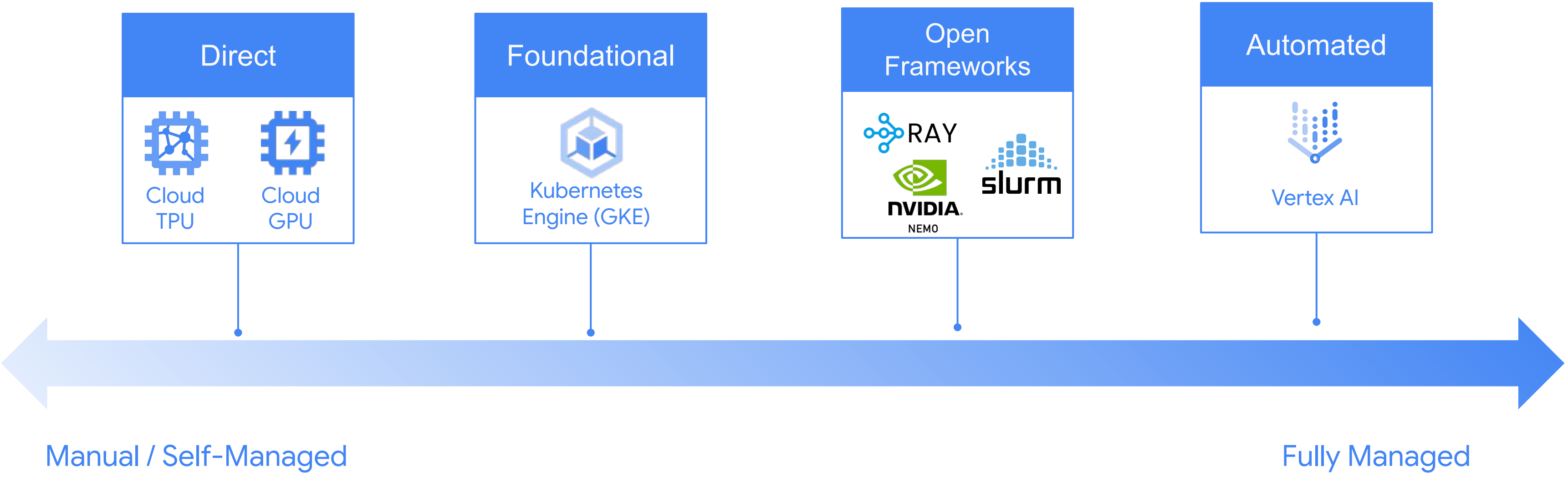
01. Vertex AI for managed training service
02. Distributed training on Ray on Vertex AI
03. deepspeed with Ray on Vertex AI
04. LLM Inferencing on Vertex AI
05. Evaluation services for LLM

Deepspeed with Ray on Vertex AI



01 Vertex AI for managed training service

Fine tuning on GCP, Select your platform



Challenges to operationalizing ML training

Lack of infrastructure management skills

Large computational resources are hard to manage and orchestrate

Cost and time

Scaled training can take days running on expensive compute resources

Disconnected and disparate toolsets

Open source and point capabilities need to be integrated into an end-to-end solution

Enterprise Security

Data is central to ML must be designed with top-class security

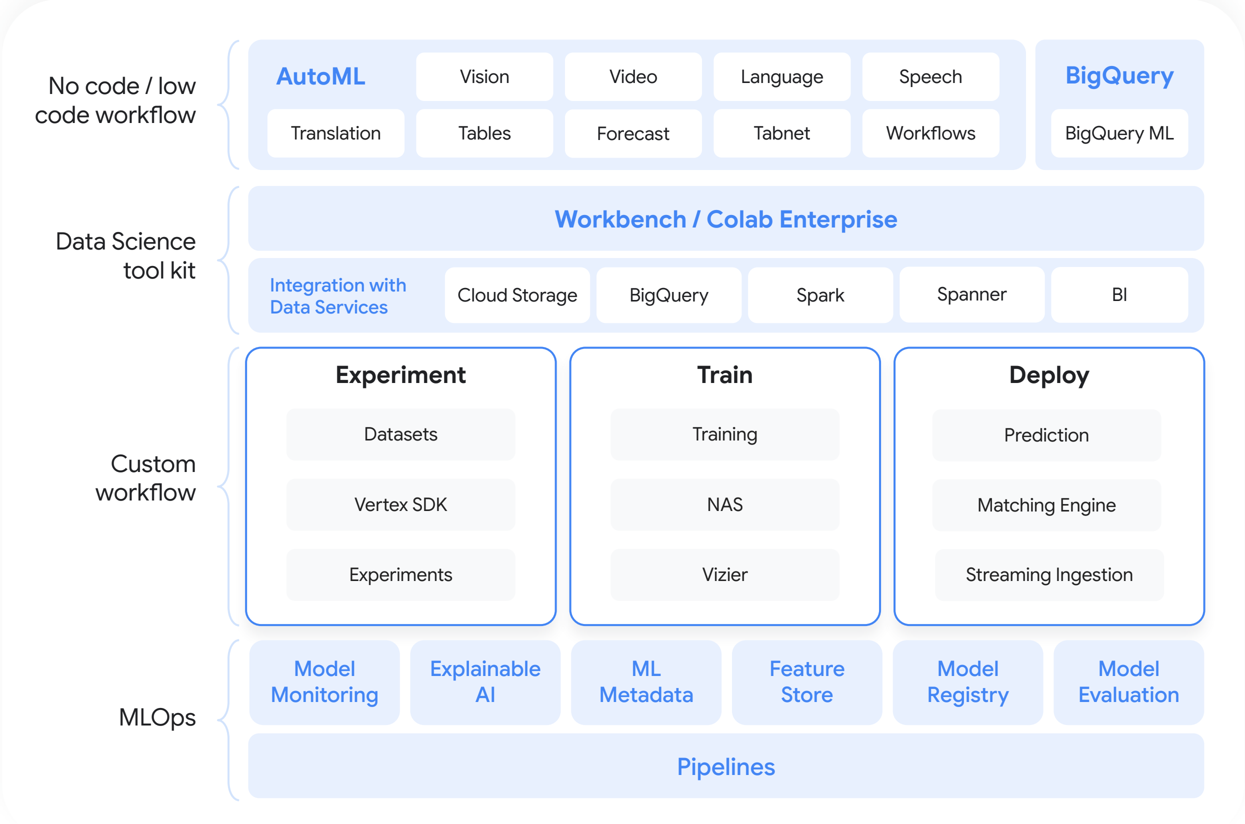
Vertex AI

- an easier way to run distributed training & to serve large models
- delivers similar performance to GKE
- an one-stop shop for all AI needs

Vertex AI

A Unified ML Platform for **Solving All Business Problems**

- **Unified** development and deployment platform for machine learning **at scale**
- Increase **productivity** of data scientists and ML engineers
- Improve **time to value**

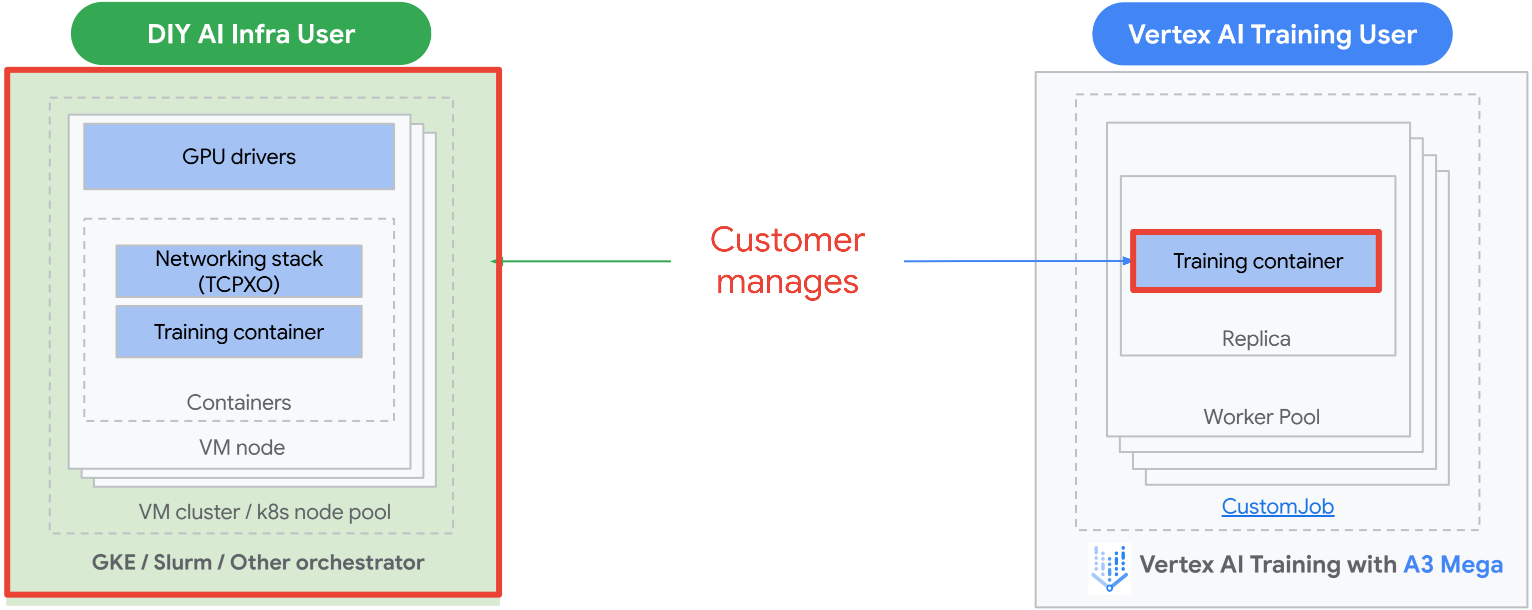


What is "Managed"?

Vertex AI Training	Training on DIY Infra
Managed infrastructure (you don't see the underlying machines, and you can't customize them - e.g. choose an OS type)	Provision machines and/or clusters
GPU drivers are automatically installed on underlying machines	Manually install and configure GPU drivers (on VMs, node pools)
TCPXO/FasTrak for A3+ is automatically available for ML jobs. No need to install manually.	Manually install and configure TCPXO/FasTrak plugin (add TCPXO sidecar containers to pods running in privileged mode, etc)
Optimized collective communications for distributed jobs using Reduction Server, for machine shapes with 100 Gbps networking (e.g. A2).	Reduction Server is not available.
Built-in job lifecycle management. Ray on Vertex provided as a standalone managed offering (built on top of Vertex persistent resources)	Install your choice of cluster / job management (Slurm, Ray, etc). Job lifecycle management is based on your chosen ML stack.
DWS is transparently integrated into Vertex AI (via advanced scheduler)	Create manual DWS requests and coordinate jobs accordingly.
Pre-built training containers for a range of popular ML frameworks.	Build your own training container images.
Connect to other GCP resources through VPC-peering (e.g. storage).	Create and configure cloud resources, connect to your VMs and clusters.
Integrated with the rest of Vertex AI for MLOps (e.g. first party integration with Vertex AI Pipelines, TensorBoard, resource monitoring, etc)	Bring your own solution(s) for MLOps, build your own bespoke integrations.

- TCPXO stands for TCP Xtender Offload. It's a technology developed by Google Cloud that optimizes network communication between Virtual Machines (VMs) within the same Google Cloud project.

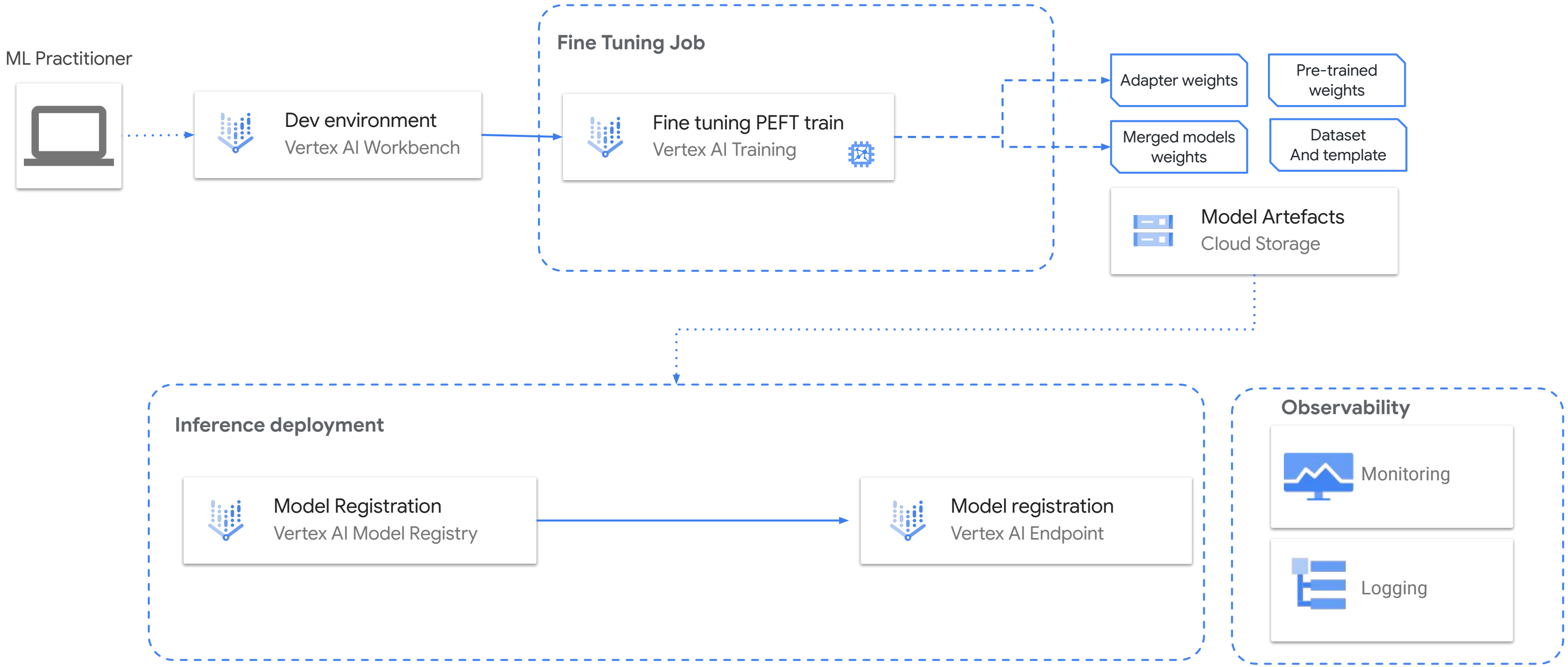
What is "Managed"? Focus on the training code, not on clusters



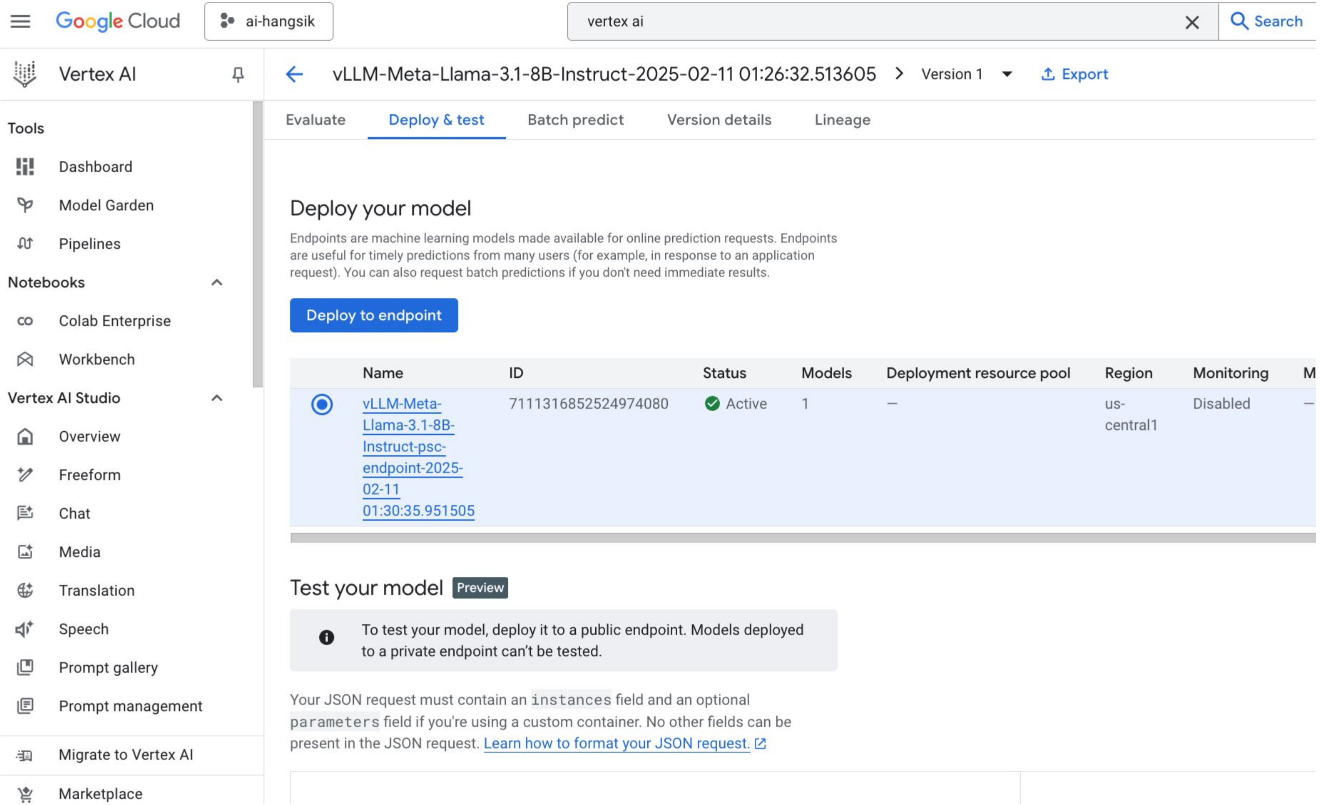
Stacks not for technical accuracy, but for explanation.

Stacks not for technical accuracy, but for explanation.

Solution overview - Open Source LLM Fine tuning on Vertex AI



Demo : GCP Console navigation



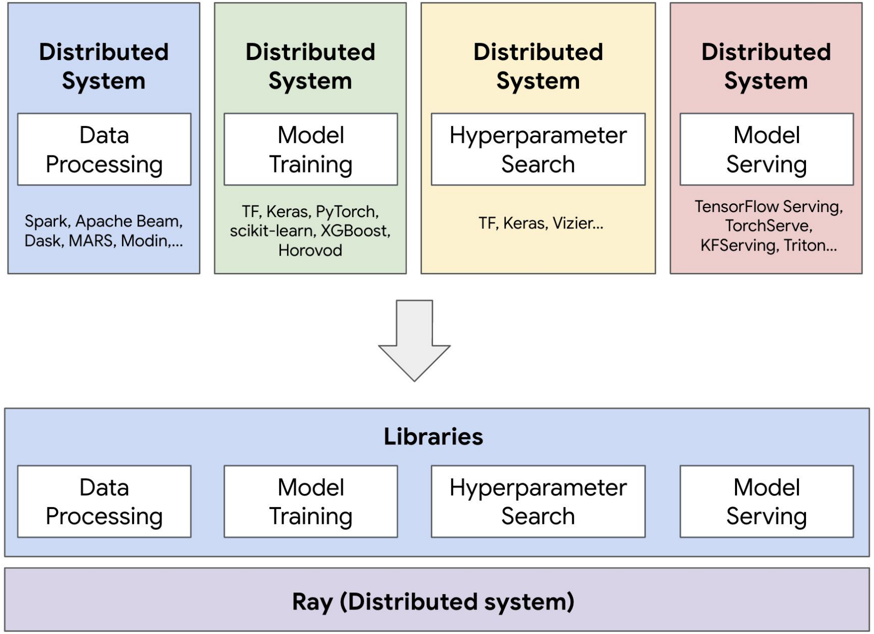
02 Distributed training Ray on Vertex AI

https://docs.ray.io/en/latest/ray-overview/index.html

Scaling ML today

Distributed System	Distributed System	Distributed System	Distributed System
Data Processing	Model Training	Hyperparameter Search	Model Serving
Spark, Apache Beam, Dask, MARS, Modin,...	TF, Keras, PyTorch, scikit-learn, XGBoost, Horovod	TF, Keras, Vizier...	TensorFlow Serving, TorchServe, KFServing, Triton...

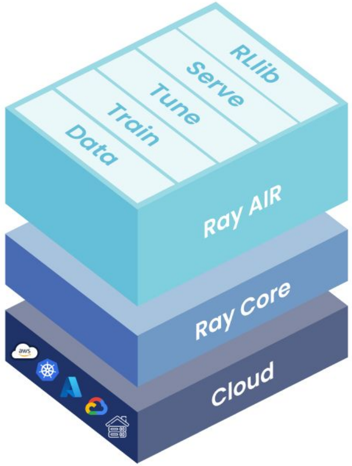
Scaling ML with Ray



Ray

is an Python open-source, scalable, and **distributed computing framework** designed to **make it easy to build and run distributed ML applications**.

Ray components



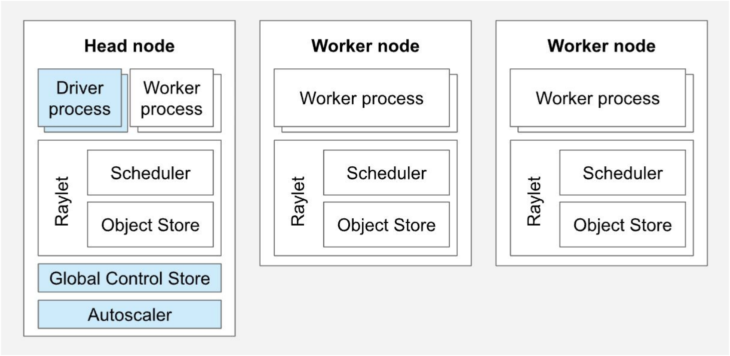
Ray Clusters - A set of clusters to scale up and down according to ML workloads.

Ray Core - An Python distributed computing library to accelerate any workloads.

Ray AI Libraries - A set of Python libraries to scale ML workloads.

Ray Cluster

- Consist of nodes (head and worker) to execute tasks
- Each node have Raylets to manage worker processes and they consist task scheduler and object store
    - Task scheduler ensure tasks have the necessary resources to run and handles dependency resolution with object store.
    - Object store ensures that workers can access objects created on different nodes by managing a shared pool of memory across workers.
- Head node has also autoscaler and GCS to verify the status of each nodes.



Ray Core

- Python library for distributed computing.
- It has three main concepts
  - **Tasks** which are **Python functions** decorated with `@ray.remote` become distributed tasks, allowing asynchronous execution on a cluster.
  - **Actors** which are `@ray.remote`-decorated **Python classes** create distributed actors, maintaining state and executing methods as remote tasks.
  - **Objects** which are Ray's distributed **object** store efficiently manages data, allowing tasks to share and pass results as objects.

Feature	Tasks	Actors
State	Stateless	Stateful
Lifetime	Short-lived	Long-lived
Execution	Independent, parallel	Encapsulated, sequential within an actor
Use Cases	Parallel computations, data processing	Distributed services, stateful operations

Ray AI Runtime Libraries (AIR)

Parquet ARROW pandas DELTA LAKE ...

XGBoost ...

mlflow W&B OPTUNA HYPEROPT Nevergrad ...

FastAPI arize gradio ALIBI ...

ray.data

```
dataset = ray.data.read_datasource(...)
train_ds, valid_ds = train_test_split(
    dataset, test_size=0.3)
test_ds = valid_ds.drop_columns(["target"])
preprocessor = StandardScaler(columns=["mean radius"])
```

ray.train

```
trainer = ray.train.xgboost.XGBoostTrainer(
   scaling_config=ScalingConfig(num_workers=128),
   label_column="target",
   datasets=dict(train=train_ds, valid=valid_ds},
   preprocessor=preprocessor)
result = trainer.fit()
```

ray.tune

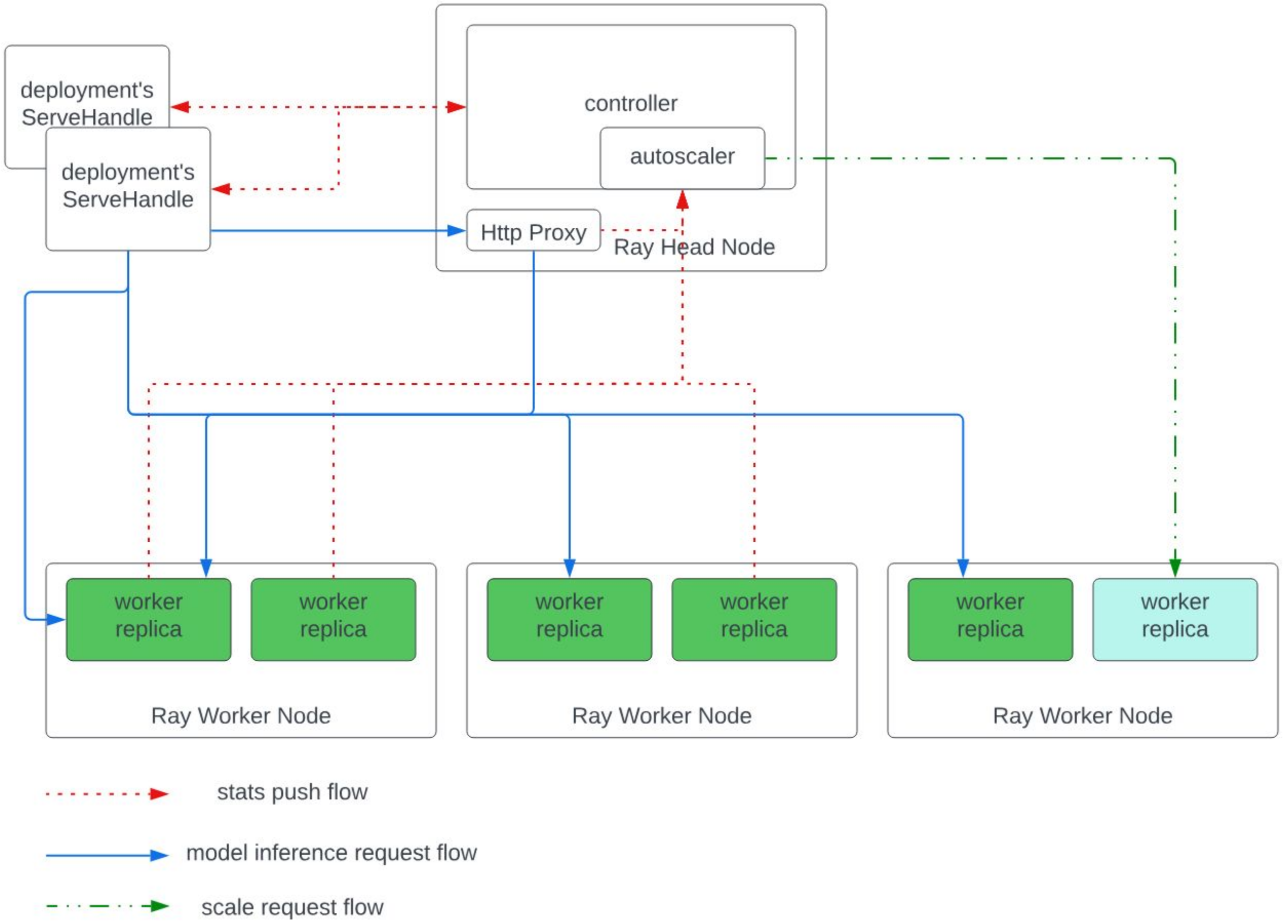
```
tuner = ray.tune.Tuner(
   trainer,
   param_space={"params": {"depth": tune.randint(1, 9)}},
   tune_config=TuneConfig(
      num_samples=5, metric="logloss", mode="min"))
checkpoint = tuner.fit().get_best_result().checkpoint
```

```
batch_predictor = BatchPredictor.from_checkpoint(
    checkpoint, XGBoostPredictor)
predicted_probabilities = batch_predictor.predict(test_ds)
predicted_probabilities.show()
```

ray.serve

```
deployment = PredictorDeployment.options(name="XGB_Serve")
deployment.deploy(XGBoostPredictor, checkpoint, ...)
print(deployment.url)
```

An unified API to enable you to pass data and models seamlessly between data processing, training, tuning, and inference (online and offline).


deployment's ServeHandle
deployment's ServeHandle
controller
autoscaler
Http Proxy
Ray Head Node
worker replica
worker replica
worker replica
worker replica
worker replica
worker replica
Ray Worker Node
Ray Worker Node
Ray Worker Node
stats push flow
model inference request flow
scale request flow

Demo : Ray simple demo - Core, Data

https://github.com/shins777/llmOps_vertexAI/tree/main/training/ray/concept

```
48

[24]: ray.init(num_cpus= num_logical_cpus)

2025-02-11 09:52:57,766 INFO worker.py:1715 -- Started a local Ray instance. View the dashboard at 127.0.0
```

[24]: RAY

Python version:	3.10.16
Ray version:	2.9.3
Dashboard:	http://127.0.0.1:8265

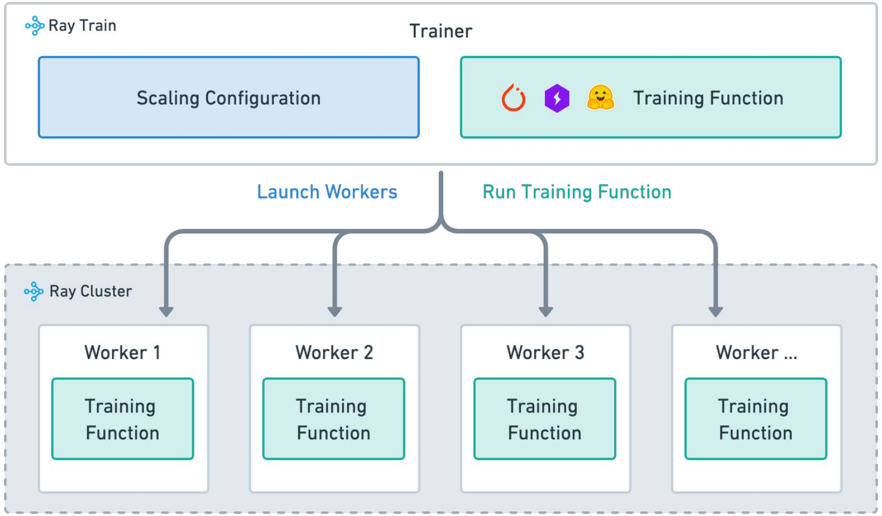
Disconnect

```
(print_current_datetime pid=27929) 2025-02-11 09:53:11.714653
(print_current_datetime pid=27929) 2025-02-11 09:53:13.873636
(print_current_datetime pid=27929) 2025-02-11 09:53:17.931943
(print_current_datetime pid=27928) 2025-02-11 09:56:28.163617
(print_current_datetime pid=27928) 2025-02-11 09:56:31.717879
(print_current_datetime pid=27929) 2025-02-11 09:56:37.623654
```

[30]:
```
# Ray Task
@ray.remote
def print_current_datetime():
    time.sleep(3)
    current_datetime = datetime.datetime.now()
    print(current_datetime)
```

Ray Train overview

1. Training function: A Python function that contains your model training logic.
2. Worker: A process that runs the training function.
3. Scaling configuration: A configuration of the number of workers and compute resources (for example, CPUs or GPUs).
4. Trainer: A Python class that ties together the training function, workers, and scaling configuration to execute a distributed training job



https://docs.ray.io/en/latest/train/getting-started-pytorch.html

Ray Train provides support for many frameworks:

PyTorch Ecosystem	More Frameworks
PyTorch	TensorFlow
PyTorch Lightning	Keras
Hugging Face Transformers	Horovod
Hugging Face Accelerate	XGBoost
DeepSpeed	LightGBM

Tuning LLM on Ray on Vertex AI

Step 1:
Set up for
Ray on Vertex AI

Step 2:
Create a Ray cluster

Step 3:
Develop an application
on the Ray on Vertex AI
cluster

Step 4:
Use Vertex AI Tensorboard
to validate training

Step 5:
Use Ray on Vertex AI to
get batch predictions

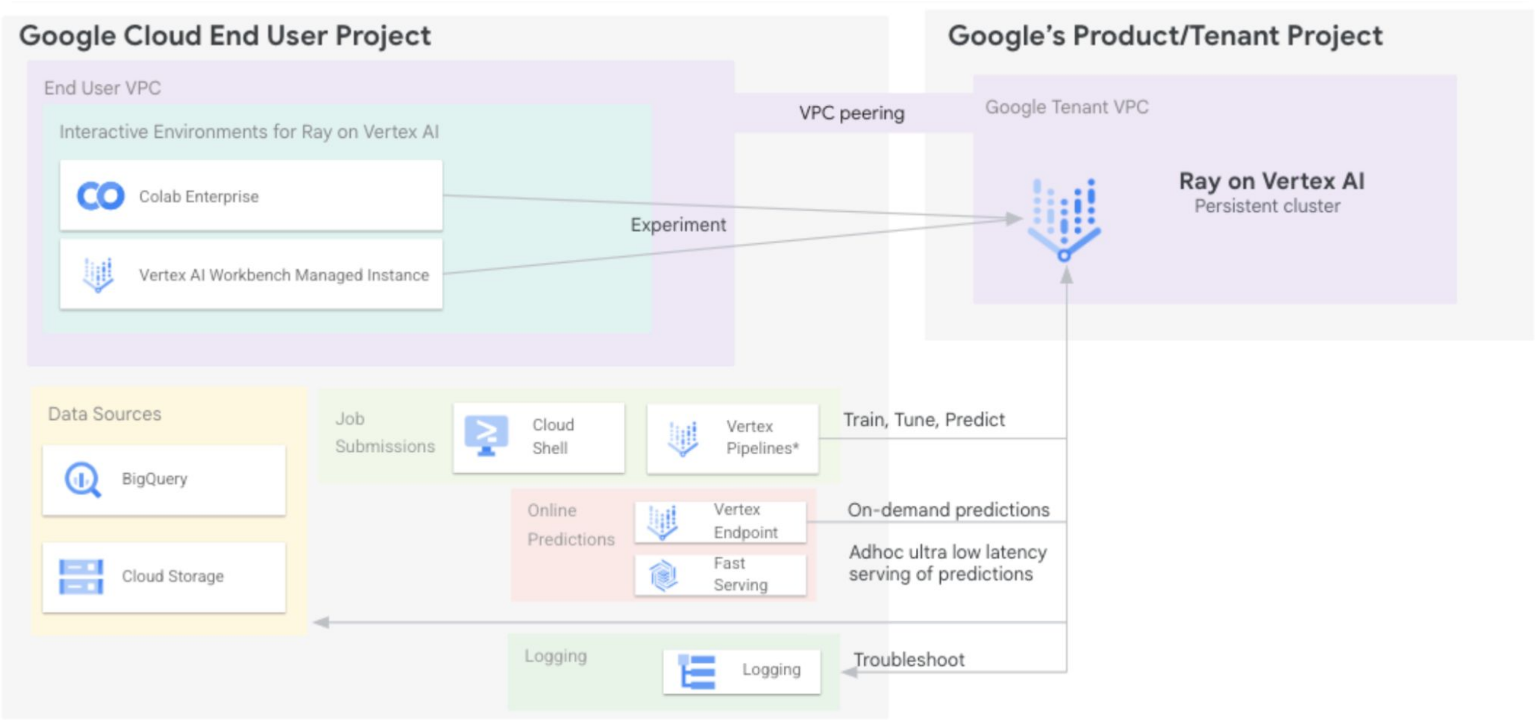
Ray container on Vertex AI (Link)

2.33.0 (Python 3.10)	GPU (CUDA 11.x)	September 16, 2025	September 16, 2026	• `us-docker.pkg.dev/vertex-ai/training/ray-gpu.2-33.py310:latest` • `europe-docker.pkg.dev/vertex-ai/training/ray-gpu.2-33.py310:latest` • `asia-docker.pkg.dev/vertex-ai/training/ray-gpu.2-33.py310:latest`

Included dependencies

PyPI packages	Ubuntu packages
absl-py 2.1.0 cloud-tpu-client 0.10 cloudml-hypertune python-json-logger 2.0.7 google-cloud-resource-manager 1.12.4 setuptools 69.5.1 kubernetes Other Deep Learning Containers dependencies	ca-certificates-java libatlas-base-dev liblapack-dev g++ libio-all-perl libyaml-0-2 Other Deep Learning Containers dependencies

Architecture of Ray on Vertex AI



Demo : pytorch distributed training with Ray on Vertex AI

https://github.com/shins777/llmOps_vertexAI/tree/main/training/ray/pytorch

```
48

[24]: ray.init(num_cpus= num_logical_cpus)

2025-02-11 09:52:57,766 INFO worker.py:1715 -- Started a local Ray instance. View the dashboard at 127.0.0

[24]:
```

RAY

Python version:	3.10.16
Ray version:	2.9.3
Dashboard:	http://127.0.0.1:8265

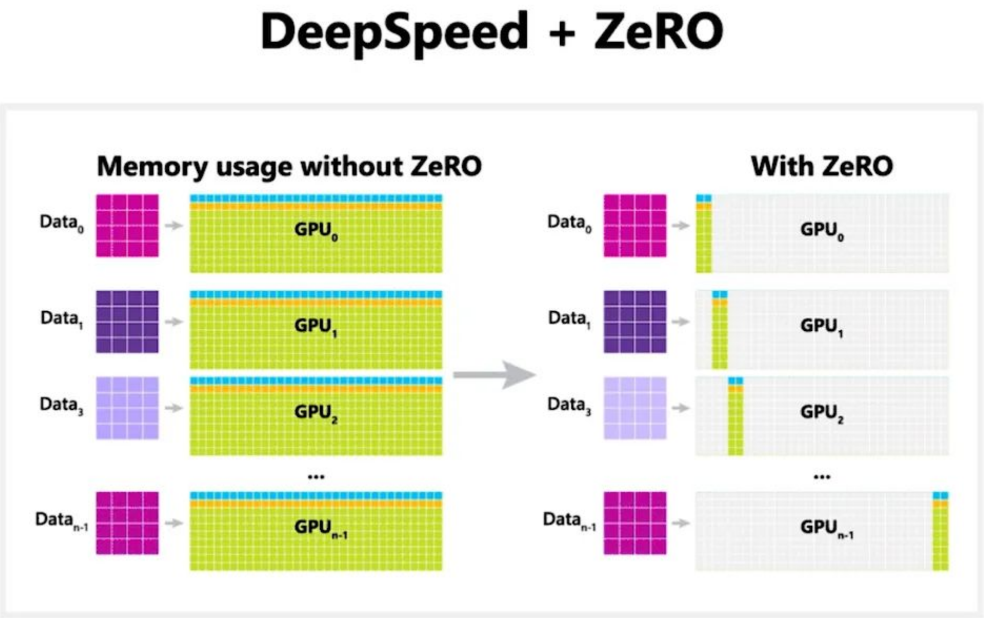
Disconnect

```
(print_current_datetime pid=27929) 2025-02-11 09:53:11.714653
(print_current_datetime pid=27929) 2025-02-11 09:53:13.873636
(print_current_datetime pid=27929) 2025-02-11 09:53:17.931943
(print_current_datetime pid=27928) 2025-02-11 09:56:28.163617
(print_current_datetime pid=27928) 2025-02-11 09:56:31.717879
(print_current_datetime pid=27929) 2025-02-11 09:56:37.623654
```

```
[30]: # Ray Task
@ray.remote
def print_current_datetime():
    time.sleep(3)
    current_datetime = datetime.datetime.now()
    print(current_datetime)
```

03 deepspeed with Ray on Vertex AI

Deepspeed Concept



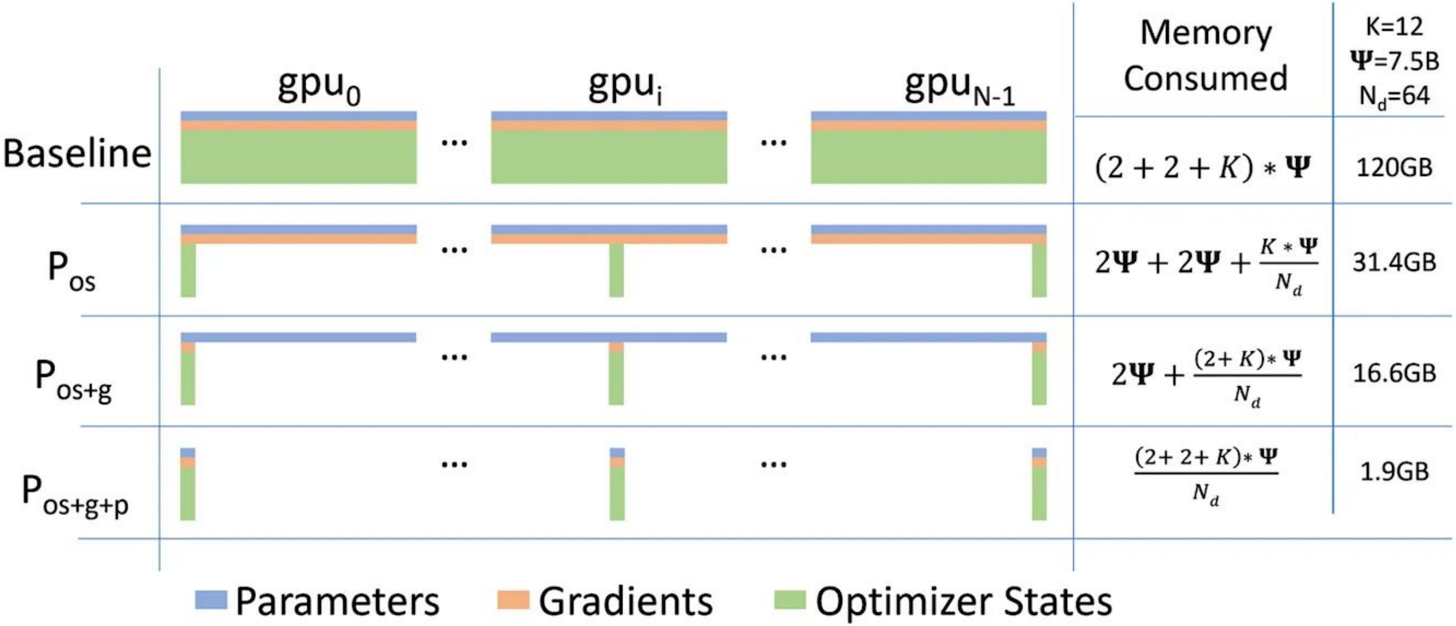
Scale
- 100B parameter
- 10X bigger

Speed
- Up to 5X faster

Cost
- Up to 5X cheaper

Usability
- Minimal code change

Comparing the per-device memory consumption



https://arxiv.org/pdf/1910.02054

Demo : deepspeed with Ray on Local machine

https://github.com/shins777/llmOps_vertexAI/blob/main/evaluation/compare_generative_ai_models.ipynb

deepspeed on Local Ray

- https://docs.ray.io/en/latest/train/examples/deepspeed/gptj_deepspeed_fine_tuning.html

Configuration

In [1]:

```
!pip install --user -q "google-cloud-aiplatform[ray]>=1.56.0" \
                        "ray[data,train,tune,serve]>=2.9.3" \
                          "transformers==4.27.4" \
                          "deepspeed>=0.14.4" \
                          "torch==2.1.2"
```

In [2]:

```
import numpy as np
import pandas as pd
import os

import ray
```

In [71]:

```
model_name = "EleutherAI/gpt-j-6B"
use_gpu = True
num_workers = 2
cpus_per_worker = 2
```

Demo : deepspeed with Ray on Vertex AI

https://github.com/shins777/llmOps_vertexAI/blob/main/evaluation/compare_generative_ai_models.ipynb

Deepspeed with Ray on Vertex AI

- https://docs.ray.io/en/latest/train/examples/deepspeed/gptj_deepspeed_fine_tuning.html

Configuration

In [1]:
```
%pip install --user -q "google-cloud-aiplatform[ray]>=1.56.0" \
                       "ray[data,train,tune,serve]==2.33.0"
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import numpy as np
import pandas as pd
import os

import ray
```

In [3]:
```
ray.__version__
```

Out[3]:
```
'2.33.0'
```

04 LLM Inferencing on Vertex AI

https://github.com/shins777/llmOps_vertexAI/tree/main/inference

Model Upload(Link)

upload(

display_name: typing.Optional[str] = None,
description: typing.Optional[str] = None,
project: typing.Optional[str] = None,
location: typing.Optional[str] = None,
credentials: typing.Optional[google.auth.credentials.Credentials] = None,
labels: typing.Optional[typing.Dict[str, str]] = None,
staging_bucket: typing.Optional[str] = None, sync=True,

model_id: typing.Optional[str] = None,
serving_container_image_uri: typing.Optional[str] = None, *,
artifact_uri: typing.Optional[str] = None,
serving_container_predict_route: typing.Optional[str] = None,
serving_container_health_route: typing.Optional[str] = None,

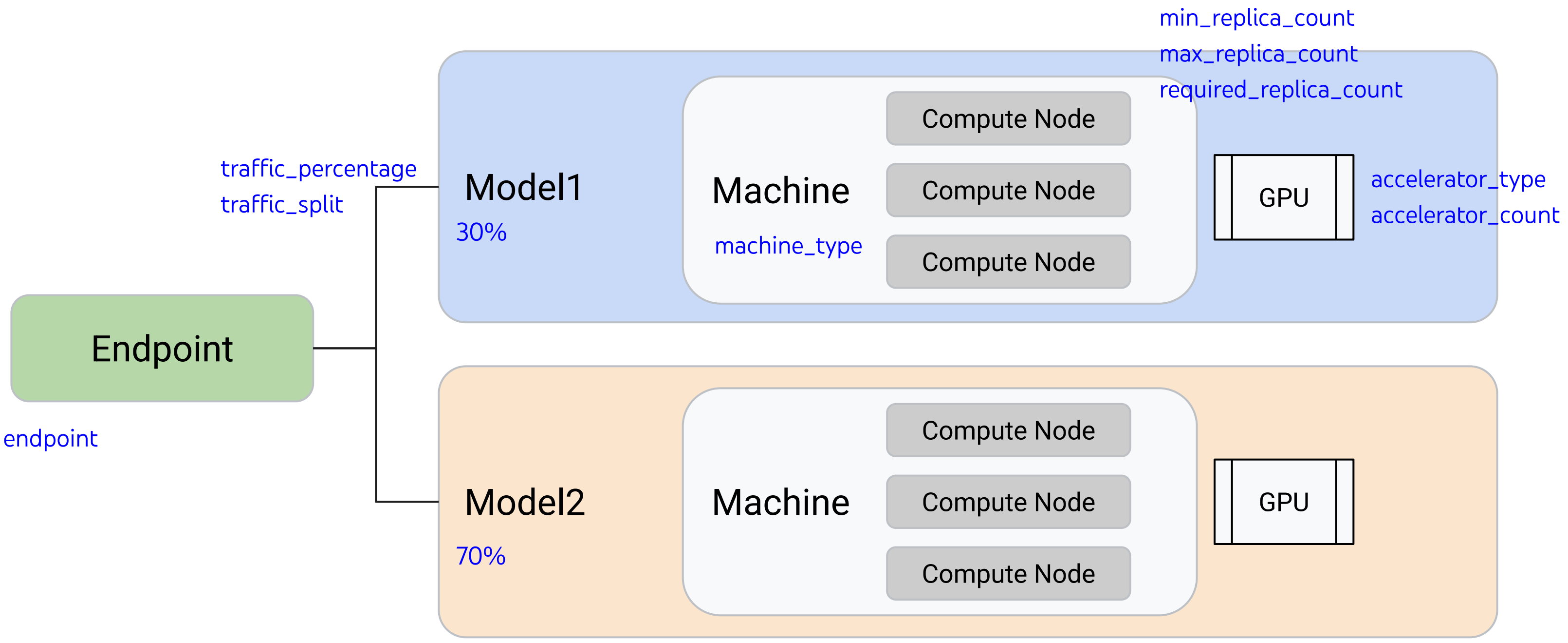
serving_container_command: typing.Optional[typing.Sequence[str]] = None,
serving_container_args: typing.Optional[typing.Sequence[str]] = None,
serving_container_environment_variables: typing.Optional[typing.Dict[str, str]] = None, **serving_container_ports**:
typing.Optional[typing.Sequence[int]] = None,
serving_container_grpc_ports: typing.Optional[typing.Sequence[int]] = None,
local_model: typing.Optional[LocalModel] = None,

Model deploy (Link)

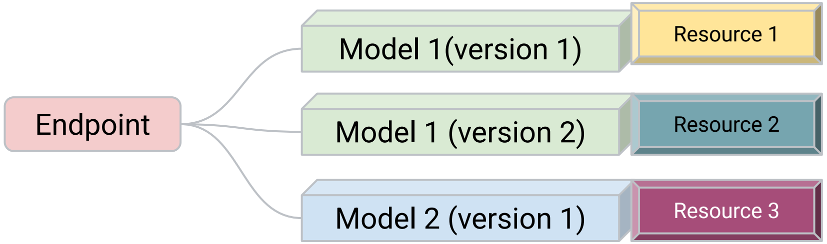
```
deploy(
        endpoint: typing.Optional[ typing.Union[ google.cloud.aiplatform.models.Endpoint, google.cloud.aiplatform.models.PrivateEndpoint, ] ] = None,
        deployed_model_display_name: typing.Optional[str] = None,
        traffic_percentage: typing.Optional[int] = 0,
        traffic_split: typing.Optional[typing.Dict[str, int]] = None,
        service_account: typing.Optional[str] = None,

        network: typing.Optional[str] = None, sync=True,
        machine_type: typing.Optional[str] = None,
        min_replica_count: int = 1,
        max_replica_count: int = 1,
        required_replica_count: typing.Optional[int] = 0,
        accelerator_type: typing.Optional[str] = None,
        accelerator_count: typing.Optional[int] = None,
        autoscaling_target_cpu_utilization: typing.Optional[int] = None,
        autoscaling_target_accelerator_duty_cycle: typing.Optional[int] = None,
```

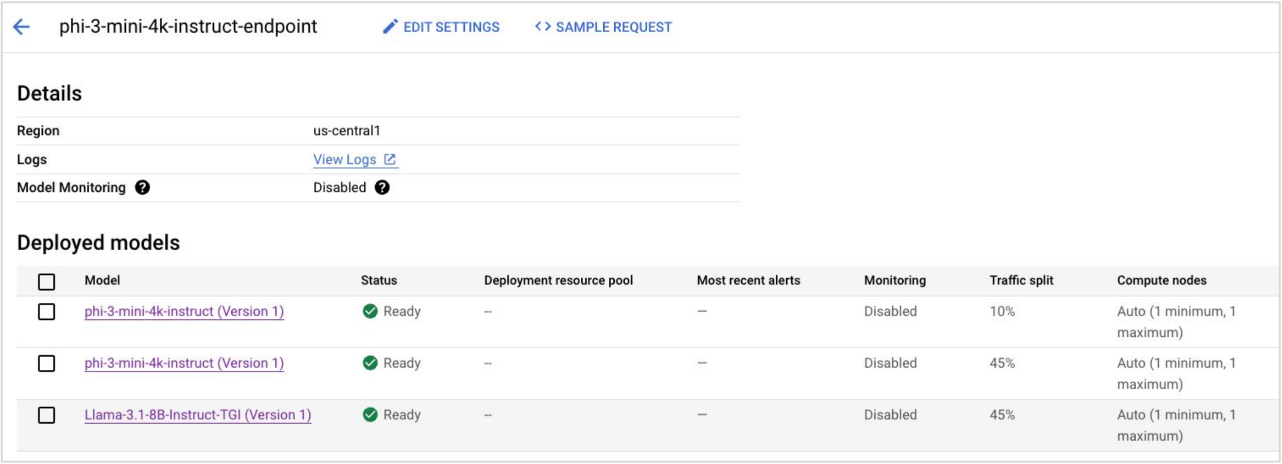
Endpoint - Model - Machine - Compute node



Deploy multiple model to the same endpoint(Link)



- Deploying **multiple models to the same endpoint** lets you gradually replace one model with the others.
- **No latency when change the traffic split.**



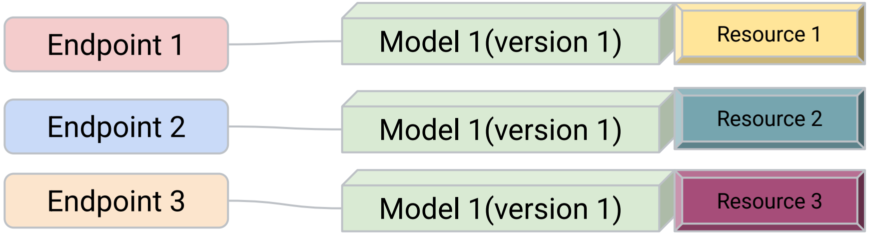
Details

Region	us-central1
Logs	View Logs
Model Monitoring	Disabled

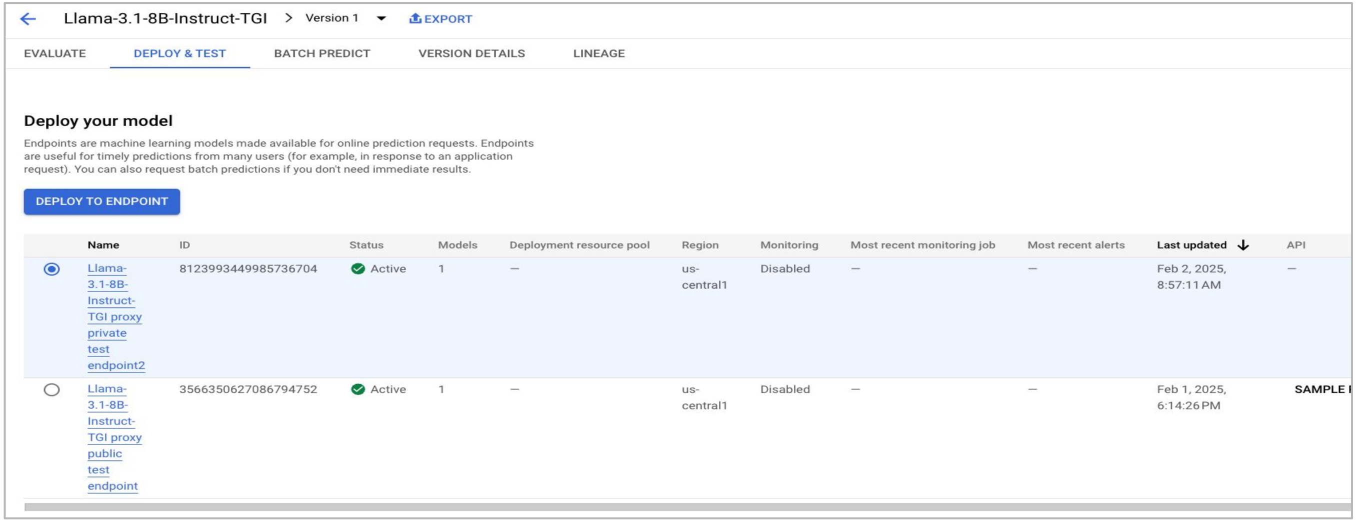
Deployed models

Model	Status	Deployment resource pool	Most recent alerts	Monitoring	Traffic split	Compute nodes
phi-3-mini-4k-instruct (Version 1)	Ready	–	–	Disabled	10%	Auto (1 minimum, 1 maximum)
phi-3-mini-4k-instruct (Version 1)	Ready	–	–	Disabled	45%	Auto (1 minimum, 1 maximum)
Llama-3.1-8B-Instruct-TGI (Version 1)	Ready	–	–	Disabled	45%	Auto (1 minimum, 1 maximum)

Deploy a model to more than one endpoint(Link)

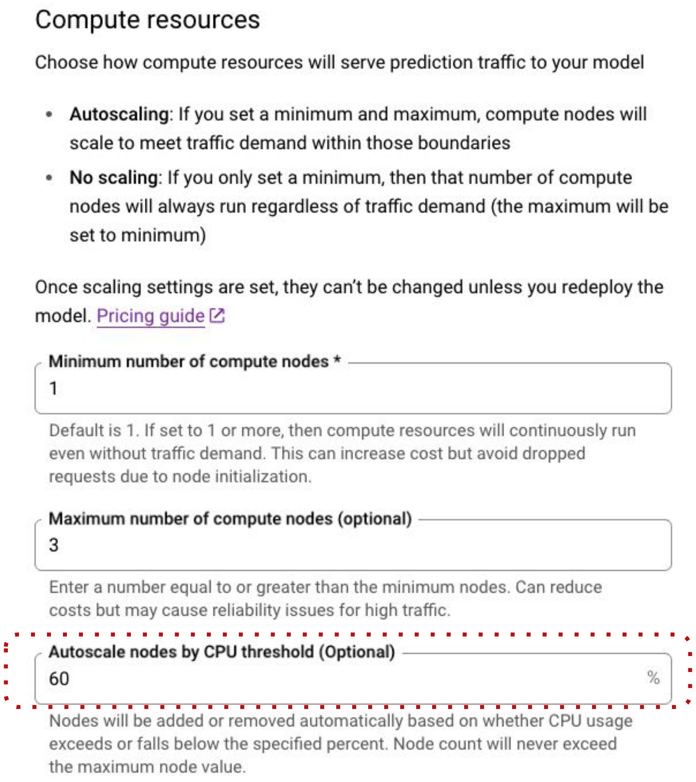


- Deploy **the same model with different resources** for different application environments, such as **testing and production or different SLOs**



Name	ID	Status	Models	Deployment resource pool	Region	Monitoring	Most recent monitoring job	Most recent alerts	Last updated	API
Llama-3.1-8B-Instruct-TGI proxy private test endpoint2	8123993449985736704	Active	1	–	us-central1	Disabled	–	–	Feb 2, 2025, 8:57:11 AM	–
Llama-3.1-8B-Instruct-TGI proxy public test endpoint	3566350627086794752	Active	1	–	us-central1	Disabled	–	–	Feb 1, 2025, 6:14:26 PM	SAMPLE I

Scaling behavior (Link)



Compute resources

Choose how compute resources will serve prediction traffic to your model

- **Autoscaling**: If you set a minimum and maximum, compute nodes will scale to meet traffic demand within those boundaries
- **No scaling**: If you only set a minimum, then that number of compute nodes will always run regardless of traffic demand (the maximum will be set to minimum)

Once scaling settings are set, they can't be changed unless you redeploy the model. Pricing guide

Minimum number of compute nodes *

1

Default is 1. If set to 1 or more, then compute resources will continuously run even without traffic demand. This can increase cost but avoid dropped requests due to node initialization.

Maximum number of compute nodes (optional)

3

Enter a number equal to or greater than the minimum nodes. Can reduce costs but may cause reliability issues for high traffic.

Autoscale nodes by CPU threshold (Optional)

60 %

Nodes will be added or removed automatically based on whether CPU usage exceeds or falls below the specified percent. Node count will never exceed the maximum node value.

- Need to consider Quota and price.
  - a2-highgpu-2g machine type, each active replica will count as 24 CPUs and 2 GPUs against your project's quota
- The prediction nodes for batch prediction don't automatically scale.

- Only CPU
  - **60 %** default target value

- CPU + GPU(if machineSpec.accelerator_count is greater than 0)
  - `autoscaling_target_cpu_utilization`
  - `autoscaling_target_accelerator_duty_cycle`
  - Depending on CPU or GPU usage, **whichever is higher, matches the default 60% target value.**

Scaling behavior - Manage resource usage, Thread handling (Link)

- Keep in mind that each replica runs **only a single container.**
- This means that if a prediction container can't fully use the selected compute resource, such as **single threaded code for a multi-core machine**, or a custom model that calls another service as part of making the prediction, **your nodes may not scale up**.
- if you are using FastAPI, or any model server that has a configurable number of workers or threads
  - We generally recommend starting with **one worker or thread per core**. If you notice that CPU utilization is low, especially under high load, or your model isn't scaling up because CPU utilization is low, then increase the number of workers.
  - On the other hand, if you notice that utilization is too high and your latencies increase more than expected under load, try using fewer workers. If you are already using only a single worker, try using a smaller machine type.

Scaling behavior and lag (Link)

- Vertex AI adjusts the number of replicas every 15 seconds using data from the previous 5 minutes window. For each 15 second cycle, the system measures the server utilization and generates a target number of replicas based on the following formula:
- target # of replicas = Ceil(current # of replicas * (current utilization / target utilization))
- For example, if you have two replicas that are being utilized at 100%, the target is 4:
  - **4 = Ceil(3.33) = Ceil(2 * (100% / 60%))**
- Another example, if you have 10 replicas and utilization drops to 1%, the target is 1:
  - 1 = Ceil(.167) = Ceil(10 * (1% / 60%))
- Keep in mind that even after Vertex AI adjusts the number of replicas, it takes time to start up or turn down the replicas.
  - the time to provision and start the Compute Engine VMs
  - the time to download the container from the registry
  - the time to load the model from storage
- The best way to understand the real world scaling behavior of your model is to run a load test and optimize the characteristics that matter for your model and your use case. If the autoscaler isn't scaling up fast enough for your application, provision enough min_replicas to handle your expected baseline traffic.

Endpoint monitoring metrics(Link)

Performance metrics

- **Predictions per second**: The number of predictions per second across both online and batch predictions. If you have more than one instance per request, each instance is counted in this chart.
- **Prediction error percentage**: The rate of errors that your model is producing. A high error rate might indicate an issue with the model or with the requests to the model. View the response codes chart to determine which errors are occurring.
- **Model latency** (for tabular and custom models only): The time spent performing computation.
- **Overhead latency** (for tabular and custom models only): The total time spent processing a request, outside of computation.
- **Total latency duration**: The total time that a request spends in the service, which is the model latency plus the overhead latency.

Resource usage

- **Replica count**: The number of active replicas used by the deployed model.
- **Replica target**: The number of active replicas required for the deployed model.
- **CPU usage**: Current CPU core usage rate of the deployed model replica. 100% represents one fully utilized CPU core, so a replica may achieve more than 100% utilization if its machine type has multiple cores.
- **Memory usage**: The amount of memory allocated by the deployed model replica and currently in use.
- **Network bytes sent**: The number of bytes sent over the network by the deployed model replica.
- **Network bytes received**: The number of bytes received over the network by the deployed model replica.
- **Accelerator average duty cycle**: The average fraction of time over the past sample period during which one or more accelerators were actively processing.
- **Accelerator memory usage**: The amount of memory allocated by the deployed model replica.

Demo : private endpoint with vLLM on Vertex AI

https://github.com/shins777/llmOps_vertexAI/blob/main/inference/private_endpoint/psc_endpoint_vllm.ipynb

Deepspeed with Ray on Vertex AI

- https://docs.ray.io/en/latest/train/examples/deepspeed/gptj_deepspeed_fine_tuning.html

Configuration

In [1]:
```
%pip install --user -q "google-cloud-aiplatform[ray]>=1.56.0" \
                        "ray[data,train,tune,serve]==2.33.0"
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import numpy as np
import pandas as pd
import os

import ray
```

In [3]:
```
ray.__version__
```

Out[3]:
```
'2.33.0'
```

05 Evaluation services for LLM

Evaluation criteria

Pointwise

- Evaluate **one model or n models** and generate scores based on the criteria
- When you need **a score for each model** being evaluated
- When it **not difficult to define the rubric** for each score.
- Examples:
  - *Understanding how your model behaves in production.*
  - *Explore strengths and weaknesses of a single model.*
  - *Identifying which behaviors to focus on when tuning.*

Pairwise

- **Compare two models** against each other, generating a preference based on the criteria
- When you want to compare two models and **a score is not necessary**
- When the score rubric for pointwise is **difficult to define.**
- Examples:
  - *Determining which model to put into production.*
  - *Choose between model types. For example, Gemini-Pro versus Claude 3.*
  - *Choose between different prompts.*

Demo : Evaluation process : Text quality

https://github.com/shins777/llmOps_vertexAI/blob/main/evaluation/gen_ai_evaluation.ipynb

Gen AI Evaluation Service

In [1]:

```
# @title Install Vertex AI Python SDK
%pip install --user --quiet google-cloud-aiplatform[evaluation]
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
# @title Set GCP information
PROJECT_ID = "ai-hangsik"  # @param {type:"string"}
LOCATION = "us-central1"  # @param {type:"string"}
EXPERIMENT_NAME = "my-eval-task-experiment"  # @param {type:"string"}
```

In [3]:

```
# @title Authentication to GCP
import sys

if "google.colab" in sys.modules:
    from google.colab import auth
    auth.authenticate_user()
```

Set Google Cloud project information and initialize Vertex AI SDK

In [4]:

```
# @title Initialize Vertex AI SDK
import vertexai

vertexai.init(project=PROJECT_ID, location=LOCATION)
```

Demo : Evaluation process : Model comparison

https://github.com/shins777/llmOps_vertexAI/blob/main/evaluation/compare_generative_ai_models.ipynb

Compare Generative AI Models | Gen AI Evaluation SDK

- Compare Generative AI Models | Gen AI Evaluation SDK Tutorial

In [1]:
```
# @title Install Vertex AI Python SDK
%pip install --user --quiet google-cloud-aiplatform[evaluation] plotly
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
# @title Set GCP information
PROJECT_ID = "ai-hangsik"  # @param {type:"string"}
LOCATION = "us-central1"  # @param {type:"string"}
EXPERIMENT_NAME = "my-eval-task-experiment"  # @param {type:"string"}
```

In [3]:
```
# @title Authentication to GCP
import sys

if "google.colab" in sys.modules:
    from google.colab import auth
    auth.authenticate_user()
```

Demo : Evaluation process : Model parameter comparison

https://github.com/shins777/llmOps_vertexAI/blob/main/evaluation/compare_gemini_model_settings.ipynb

Evaluate and Compare Gen AI Model Settings

- Evaluate and Compare Gen AI Model Settings | Gen AI Evaluation SDK Tutorial

In [1]:
```
# @title Install Vertex AI Python SDK
%pip install --upgrade --user --quiet google-cloud-aiplatform[evaluation]
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 16.7 MB/s eta 0:00:00
  WARNING: The script tb-gcp-uploader is installed in '/root/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-l
n.
```

In [2]:
```
# @title Define constants

PROJECT_ID="ai-hangsik" # @param {type:"string"}
LOCATION="us-central1" # @param {type:"string"}
```

In [3]:
```
# @title GCP Authentication

# Use OAuth to access the GCP environment.
import sys
if "google.colab" in sys.modules:
    from google.colab import auth
    auth.authenticate_user(project_id=PROJECT_ID)
```

Thank You