LLM Prediction

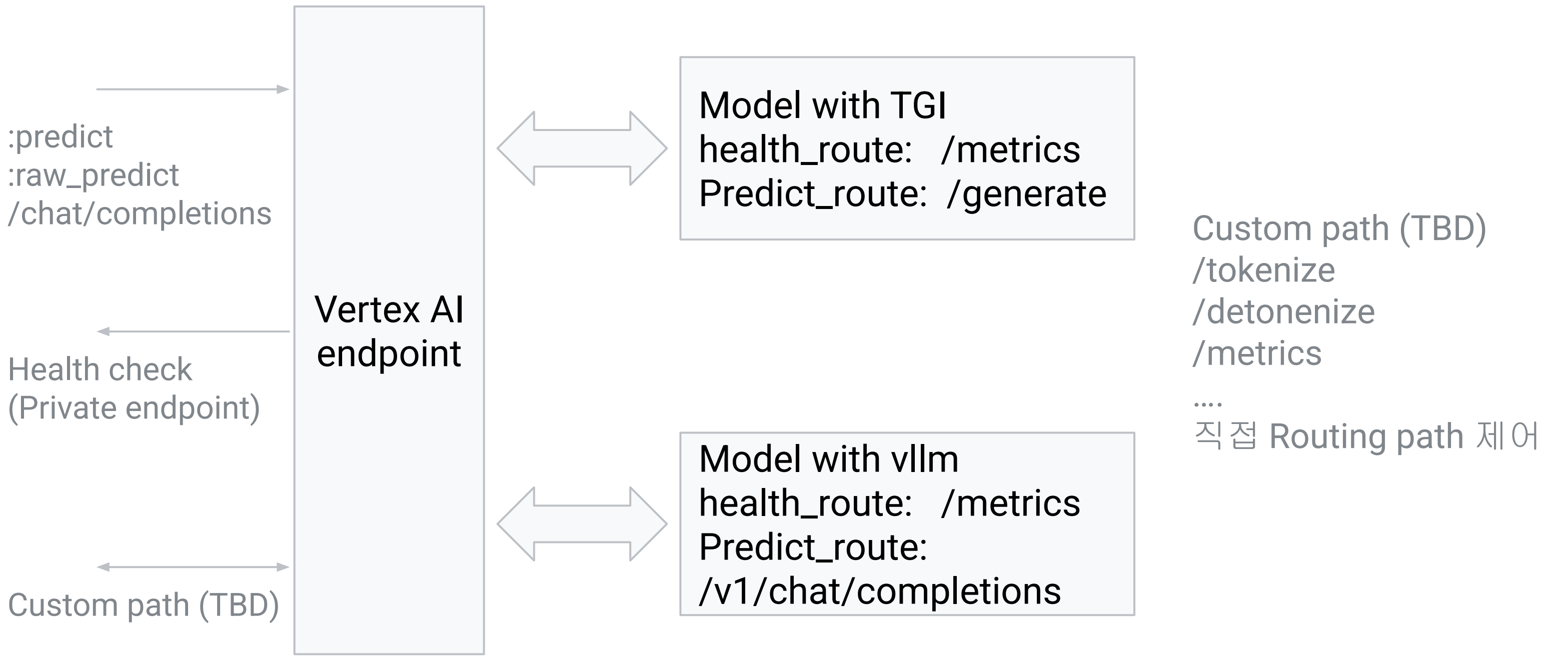
Hangsik Shin
AI Specialist
Google Cloud Korea

Agenda

01. Prediction with JSON output
02. Model load and deploy
03. Container Customization

01 Customize route /generate and /metrics

https://github.com/jk1333/mlops/tree/main/vai_endpoint



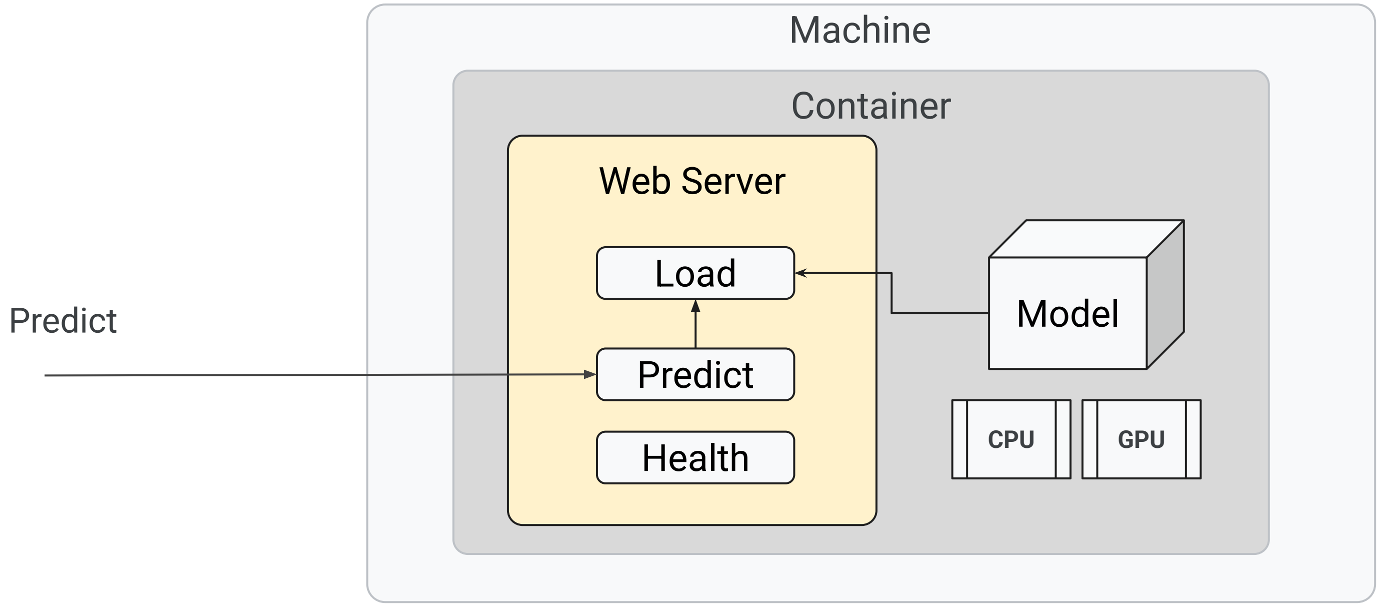
- Private endpoint 에서는 C3, L4, TPU 같은 인스턴스 이용이 불가합니다.
- 프로젝트 내에서 모든 Private endpoint 는 모두 동일한 VPC subnet 만 지정 가능합니다.
- Private endpoint 는 A/B test를 지원하지 않습니다. 그래서 단일 모델만 배포 가능합니다.
- Private endpoint 로 연결하는 Client 는 retry 를 구현하는게 좋습니다.
- Health check 같은 경우 특정 모델 index 를 지칭할 수 없기 때문에 Backend 의 인스턴스 중 하나가 출력됩니다.

02 Container Customization

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

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

Prediction container architecture



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

Custom container requirements for prediction

To use a custom container to serve predictions from a custom-trained model, you must provide Vertex AI with a Docker container image that runs an HTTP server. the container must listen and respond to liveness checks, health checks, and prediction requests

- Run the HTTP server
- Liveness checks
- Health checks
- Prediction requests

HTTP Server

- Run an HTTP server by using an ENTRYPOINT instruction, a CMD instruction, or both in the Dockerfile that you use to build your container image. Read about the interaction between CMD and ENTRYPOINT
  - serving_container_command : ENTRYPOINT
  - serving_container_args : CMD
- HTTP server must listen for requests on 0.0.0.0, on a port
  - AIP_HTTP_PORT : Vertex AI sends liveness checks, health checks, and prediction requests to this port on the container. Your container's HTTP server must listen for requests on this port.

Liveness checks

- Liveness checks
  - Vertex AI performs a liveness check when your container starts to ensure that your server is running
  - When you deploy a custom-trained model to an Endpoint resource, Vertex AI uses a TCP liveness probe to attempt to establish a TCP connection to your container on the configured port.
  - HTTP server doesn't need to perform any special behavior to handle these checks.

Health checks - Configuration

- Optionally specify startup_probe or health_probe.
- To perform a probe, Vertex AI executes the specified `exec` command in the target container. If the command succeeds, it returns 0, and the container is considered to be alive and healthy.
- HTTP GET requests to a configurable health check path(AIP_HEALTH_ROUTE)
- The startup probe checks whether the container application has started.
  - `serving_container_startup_probe_exec`
  - `serving_container_startup_probe_period_seconds`
  - `serving_container_startup_probe_timeout_seconds`
- The health probe checks whether a container is ready to accept traffic.
  - `serving_container_health_probe_exec`
  - `serving_container_health_probe_period_seconds`
  - `serving_container_health_probe_timeout_seconds`

Health checks - Operation

- If the probe receives 4 consecutive unhealthy responses(including no response within 10 seconds), Vertex AI stops routing prediction traffic to the container. (If the DeployedModel resource is scaled to use multiple prediction nodes, Vertex AI routes prediction requests to other, healthy containers.)

- Vertex AI doesn't restart the container; instead the health probe continues sending intermittent health check requests to the unhealthy server. If it receives a healthy response, it marks that container as healthy and starts to route prediction traffic to it again.

- Possible to purposefully design the HTTP server to respond to health checks with an unhealthy status at certain times. For example, you might want to block prediction traffic to a node for a period so that the container can perform maintenance.

Prediction

- Forwards this request as an HTTP POST request to a configurable prediction path(AIP_PREDICT_ROUTE)
- Request :
  - If the model is deployed to a public endpoint, each prediction request must be 1.5 MB or smaller.
- Response
  - If the model is deployed to a public endpoint, each prediction response must be 1.5 MB or smaller.

Request JSON body

```
{
  "instances": INSTANCES,
  "parameters": PARAMETERS
}
```

Response JSON body

```
{
  "predictions": PREDICTIONS
      }
```

Container image publishing

- Publishing
  - Location
    - If you plan to create a Model on the us-central1-aiplatform.googleapis.com endpoint, the full name of your container image must start with **us-central1-docker.pkg.dev/**
- Loading
  - If the container image includes the model artifacts that you need to serve predictions, there is no need to load files from Cloud Storage. However, if you do provide model artifacts by specifying the artifactUri field, the container must load these artifacts when it starts running.
  - When Vertex AI starts your container, it sets the AIP_STORAGE_URI environment variable to a Cloud Storage URI that begins with gs://. Your container's ENTRYPOINT instruction can download the directory specified by this URI in order to access the model artifacts.

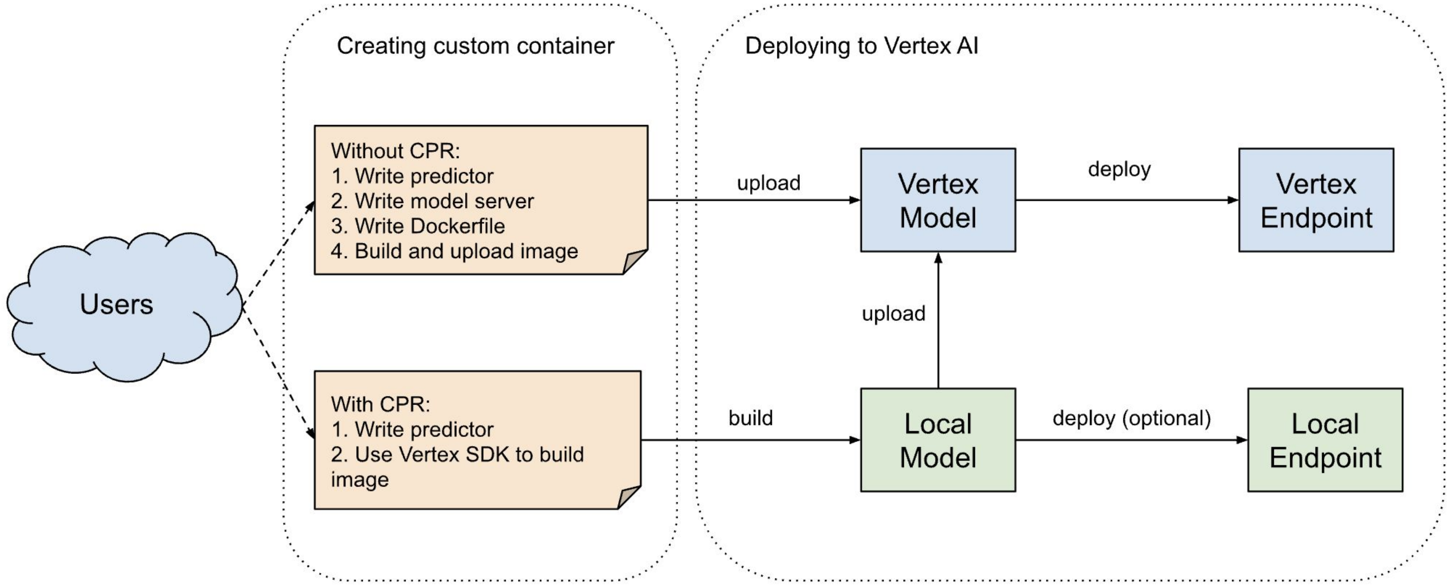
https://cloud.google.com/vertex-ai/docs/reference/rest/v1/ModelContainerSpec

Configurable variables set by Vertex AI

- AIP_MACHINE_TYPE
  - This variable specifies the type of VM that the container is running on.
- AIP_ACCELERATOR_TYPE
  - If applicable, this variable specifies the type of accelerator used by the virtual machine (VM) instance that the container is running on.
- AIP_HEALTH_ROUTE
  - This variables specifies the HTTP path on the container that Vertex AI sends health checks to.
- AIP_PREDICT_ROUTE
  - This variable specifies the HTTP path on the container that Vertex AI forwards prediction requests to.
- AIP_HTTP_PORT
  - Vertex AI sends liveness checks, health checks, and prediction requests to this port on the container. Your container's HTTP server must listen for requests on this port.

https://cloud.google.com/vertex-ai/docs/reference/rest/v1/ModelContainerSpec

Custom prediction routines



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

Benefits of CPR(Custom prediction routines)

Simplified pre- and post-processing: Custom prediction routines (CPRs) make it easier to write code for transforming data before and after prediction. No need to set up a web server in container..

Python-based: You can define your data transformations using Python code.

Automated containerization: The Vertex AI SDK automatically builds a custom container with your code.

Local testing: You can test your containerized prediction routine locally before deploying it.

Cloud deployment: Vertex AI handles deploying your container to the cloud for serving predictions.

Alternative to custom containers: CPRs provide a simpler alternative to building custom containers from scratch for pre-processing and post-processing.

https://cloud.google.com/vertex-ai/docs/predictions/custom-prediction-routines

03 Model load and deploy on Vertex AI

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,
```

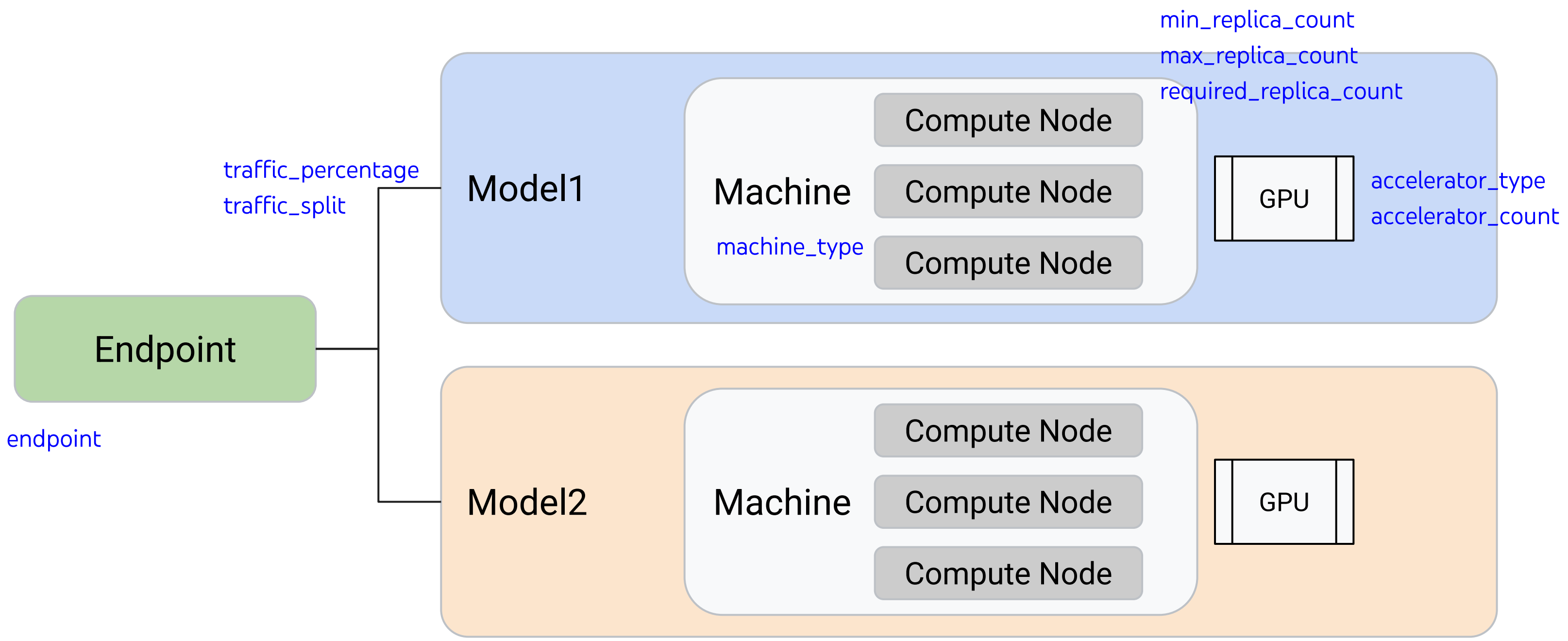
Model load and deploy

Model deploy (Link)

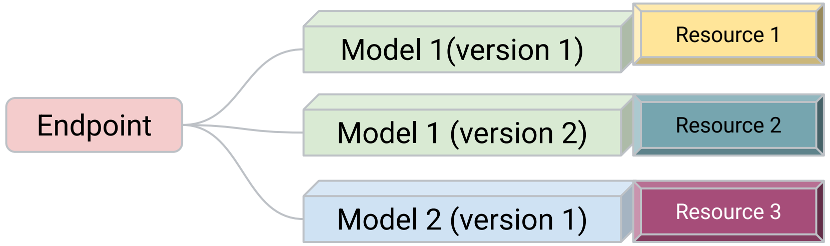
```
deploy(
        endpoint: typing.Optional[ typing.Union[ google.cloud.aiplatform.models.Endpoint, google.cloud.aiplatform.models.PrivateEndpoint, ] ] = None,
        deployed_model_display_name: typing.Optional[str] = None,
        traffic_percentage: typing.Optional[int] = 0,
        traffic_split: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

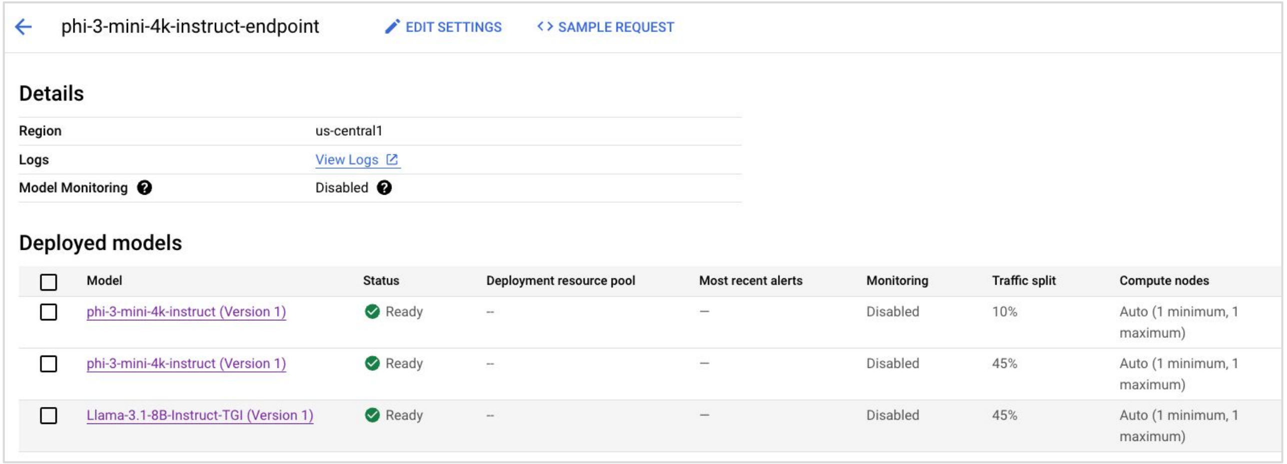
Endpoint - Model - Machine - Compute node



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.



phi-3-mini-4k-instruct-endpoint

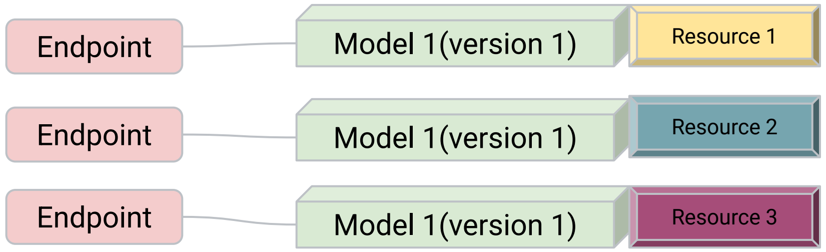
Details

Region	us-central1
Logs	View Logs
Model Monitoring	Disabled

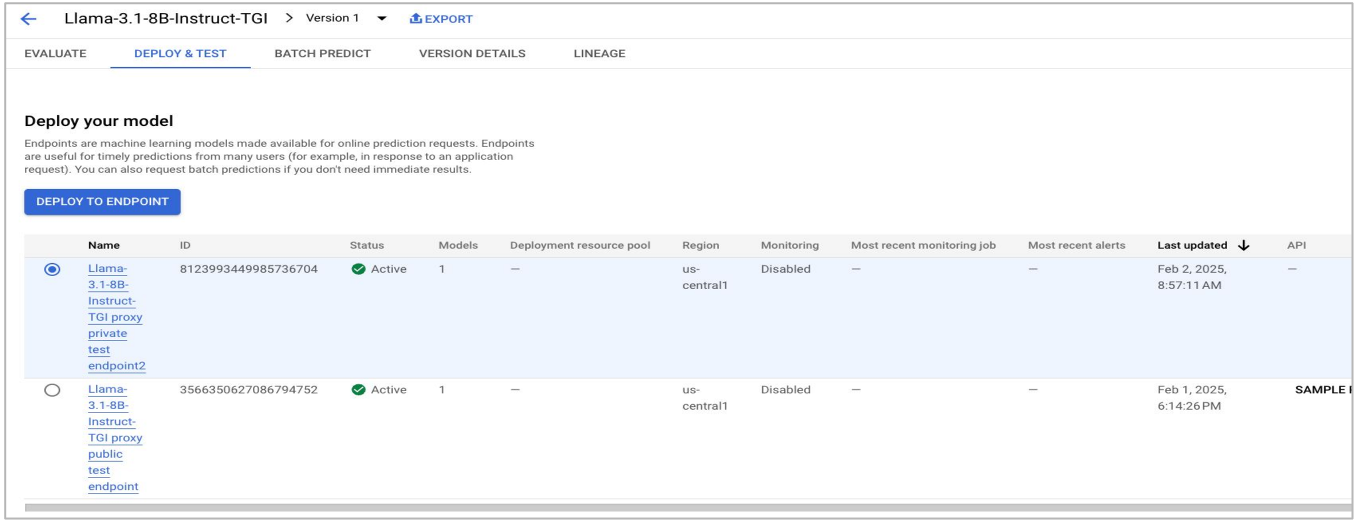
Deployed models

	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 your models with different resources for different application environments, such as testing and production or different SLOs

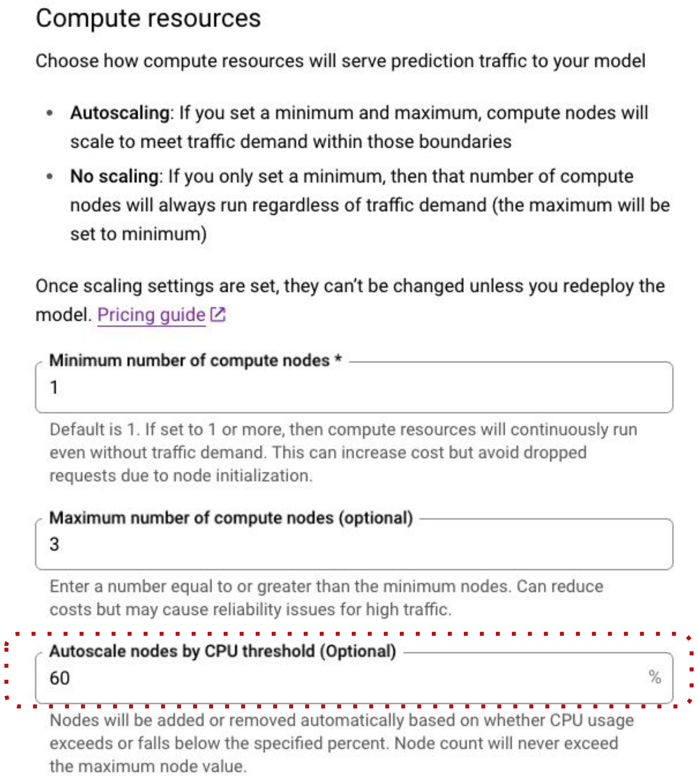


Deploy your model

Endpoints are machine learning models made available for online prediction requests. Endpoints are useful for timely predictions from many users (for example, in response to an application request). You can also request batch predictions if you don't need immediate results.

	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.

Thank You