Google Cloud

Partner Certification Academy

Professional Machine Learning Engineer

The information in this presentation is classified:

Google confidential & proprietary

⚠️ This presentation is shared with you under NDA.

- Do **not** record or take screenshots of this presentation.
- Do **not** share or otherwise distribute the information in this presentation with anyone **inside** or **outside** of your organization.

Thank you!

Google Cloud

Source Materials

Some of this program's content has been sourced from the following resources:

- Google Cloud certification site
- Google Cloud documentation
- Google Cloud console
- Google Cloud courses and workshops
- Google Cloud white papers
- Google Cloud Blog
- Google Cloud YouTube channel
- Google Cloud samples
- Google codelabs
- Google Cloud partner-exclusive resources

Google Cloud

Google Cloud Skills Boost for Partners

- Professional Machine Learning Engineer Certification
- Cloud Skills Boost for Partners Professional Machine Learning Engineer Learning Path
- Partner Learning Services Instructor-Led PMLE Curriculum

Google Cloud Partner Advantage

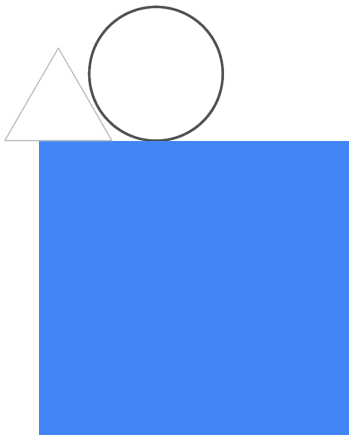
- Best practices for implementing machine learning on Google Cloud
- Artificial Intelligence
- End-to-End MLOps Go-to-Market Kit

Session Logistics

- When you have a question, please:
  - Click the Raise hand button in Google Meet.
  - Or add your question to the Q&A section of Google Meet.
  - Please note that answers may be deferred until the end of the session.
- These slides are available in the Student Lecture section of your Qwiklabs classroom.
- The session is **not recorded.**
- Google Meet does not have persistent chat.
  - If you get disconnected, you will lose the chat history.
  - Please copy any important URLs to a local text file as they appear in the chat.

Google Cloud Partner Learning Programs

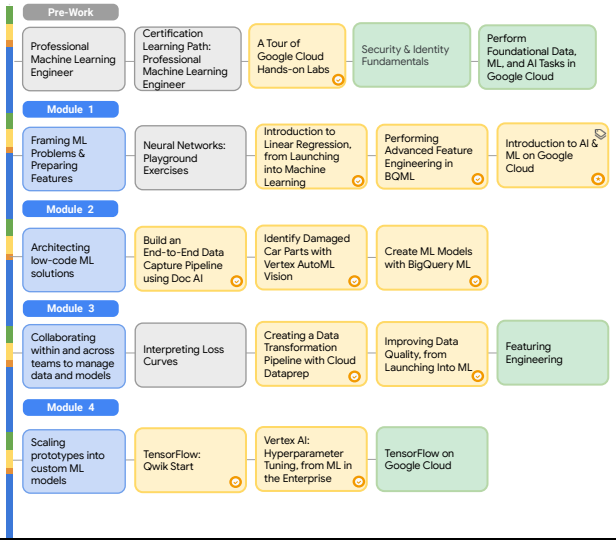
- Partner Certification Academy
- Partner Delivery Readiness Index (DRI)
- Cloud Skills Boost for Partners
- Partner Advantage



PARTNER CERTIFICATION ACADEMY

Professional Machine Learning Engineer

On-demand Course
Hands-on Skill Badge
Hands-on Lab
Resource
Live Virtual Workshop
required to earn exam voucher



A Professional Machine Learning Engineer builds, evaluates, productionizes, and optimizes ML models by using Google Cloud technologies and knowledge of proven models and techniques. The ML Engineer:

- handles large, complex datasets and creates repeatable, reusable code.
- considers responsible AI and fairness throughout the ML model development process, and collaborates closely with other job roles to ensure long-term success of ML-based applications.
- has strong programming skills and experience with data platforms and distributed data processing tools.
- is proficient in the areas of model architecture, data and ML pipeline creation, and metrics interpretation.
- is familiar with foundational concepts of MLOps, application development, infrastructure management, data engineering, and data governance.
- makes ML accessible and enables teams across the organization.

By training, retraining, deploying, scheduling, monitoring, and improving models, the ML Engineer designs and creates scalable, performant solutions.

Recommended candidate:

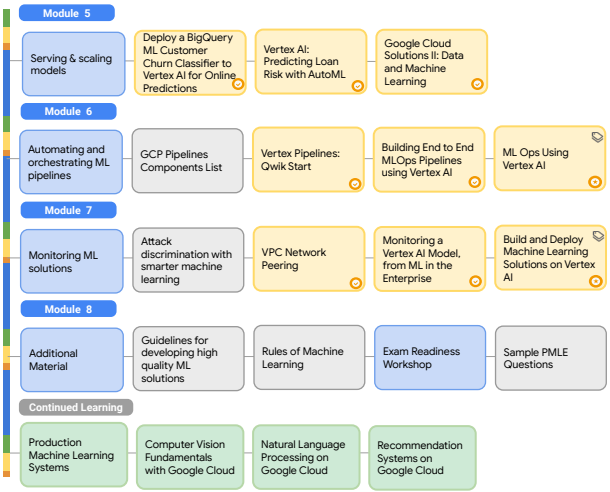
- Has in-depth experience setting up cloud environments for an organization
- Has experience deploying services and solutions based on business requirements

Google Cloud

PARTNER CERTIFICATION ACADEMY

Professional Machine Learning Engineer

On-demand Course
Hands-on Skill Badge
Hands-on Lab
Resource
Live Virtual Workshop
required to earn exam voucher



A Professional Machine Learning Engineer builds, evaluates, productionizes, and optimizes ML models by using Google Cloud technologies and knowledge of proven models and techniques. The ML Engineer:

- handles large, complex datasets and creates repeatable, reusable code.
- considers responsible AI and fairness throughout the ML model development process, and collaborates closely with other job roles to ensure long-term success of ML-based applications.
- has strong programming skills and experience with data platforms and distributed data processing tools.
- is proficient in the areas of model architecture, data and ML pipeline creation, and metrics interpretation.
- is familiar with foundational concepts of MLOps, application development, infrastructure management, data engineering, and data governance.
- makes ML accessible and enables teams across the organization.

By training, retraining, deploying, scheduling, monitoring, and improving models, the ML Engineer designs and creates scalable, performant solutions.

Recommended candidate:

- Has in-depth experience setting up cloud environments for an organization
- Has experience deploying services and solutions based on business requirements

Google Cloud

Learner Commitment

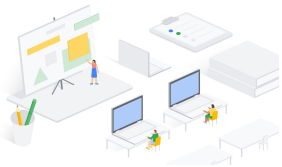
Each week, learners are to complete the learning path's course content, Cloud Skills Boost for Partner Quests/Challenge Labs and material that the mentor has recommended that will support learning.

- **Workshop Day:** Meet for the cohort's weekly 'general session'. **(≅ 2 hours)**
- **During the week:** Complete the week's course, perform hands-on labs, review any additional material suggested material for the week. **(≅ 8 - 16 hours)**

➔ **Important:** Learners must allocate time between each weekly session to study and familiarize themselves with any prerequisite knowledge they may lack. It is also recommended that learners complete the next week's course prior to the scheduled workshop.

Path to Service Excellence

Certification



Advanced Solutions Training

Delivery Readiness Index

Certification is just one step on your professional journey. Google Cloud also offers our partners access to advanced solutions training, and a new quality-focused program called Delivery Readiness Index (DRI) to help you achieve service excellence with your customers.

Benchmark your skills with DRI

DRI helps to benchmark partner proficiency and capability at any point during the customer journey however should be used primarily as a lead measure to predict and prepare for partner delivery success.

DRI assesses and analyzes Partner Consultant GCP proficiency by creating a DRI Profile inclusive of their GCP knowledge, skills, and experience.

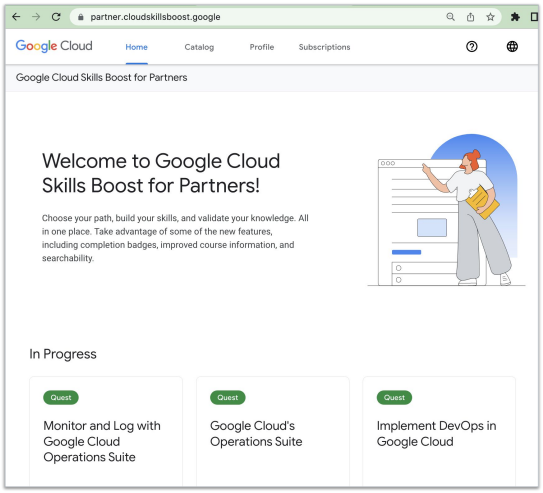
With the DRI insights, we can prescriptively advise the partner project team on the ground and bridge niche capability gaps.

DRI also takes action. For partner consultants, DRI generates a tailored L&D plan that prescribes personalized learning, training, and skill development to build GCP proficiency.

Google Cloud Skills Boost for Partners

https://partner.cloudskillsboost.google/

- On-demand course content
- Hands-on labs
- Skill Badges
- **FREE** to Google Cloud Partners!

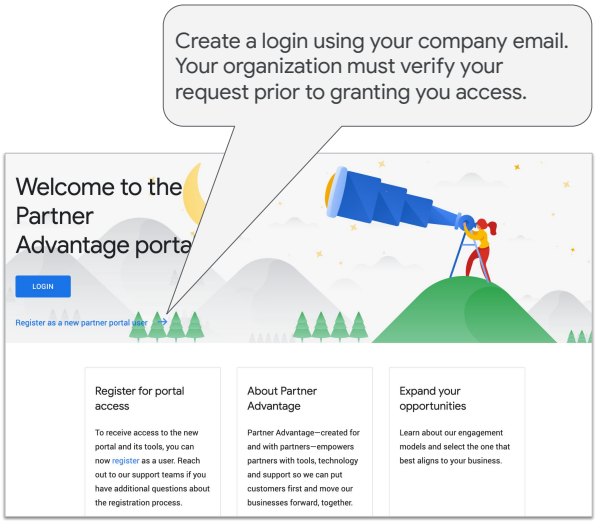


Proprietary + Confidential

Google Cloud

Partner Advantage

- Resources for Google Cloud partner organizations:
  - Recent announcements
  - Solutions/role-based training
  - Live/pre-recorded webinars on various topics
    - Partner Advantage Live Webinars
- Complements the certification self-study material presented on Google Cloud Skills Boost for Partners
- Helpful Links:
  - Getting started on Partner Advantage
  - Join Partner Advantage
  - Get help accessing Partner Advantage



https://www.partneradvantage.goog

Google Cloud

The getting started link:
https://support.google.com/googlecloud/topic/9198654#zippy=

Note the top section, **"Getting Started & User Guides"** and two key documents → Direct Partners to this if they need to enroll into Partner Advantage

1. Logging in to the Partner Advantage Portal - Quick Reference Guide
2. Enrolling in the Partner Advantage Program - Quick Reference Guide

Focus from this point on:

Some context on enrolling in PA:
Access to Partner Portal is given in 2 ways

- Partner Admin Led: Partner Administrator at Partner Company can set up users
- User Led: User can go through Self Registration
  - https://www.partneradvantage.goog/GCPPRM/s/partneradvantageportallogin?language=en_US
  - Or directly to the User Registration Form, https://www.partneradvantage.goog/GCPPRM/s/partnerselfregistration?language=en_US

Please Note

- After a user self-registers, they receive an email that essentially states:

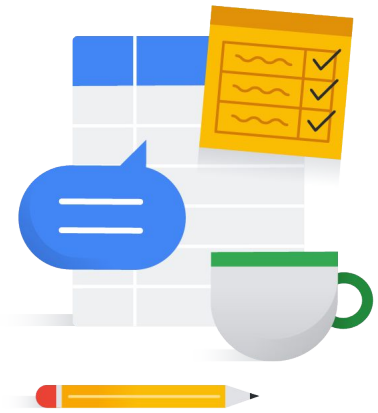
    - “Hi {Partner Name}, you are one step away from joining the Google Cloud Partner Advantage Community. Please click to continue with the user registration process. See you in the cloud, The Partner Advantage Team
- Once registered, they can access limited content until their **Partner Administrator approves the user**
- Their Partner Administrator also receive an email notifying them that a member of their organization has registered themselves on their organization’s Google Cloud Partner Advantage account.
    - It also states that this user has limited access to the portal
    - They are provided instructions on how to review and provision the appropriate access for the user that has registered
- Once their admin approves the user, they receive an email that states:
    - Hi {User Name}, Your Partner Administrator has updated your access to the Google Cloud Partner Advantage portal. You have been granted edit access to additional account information on the portal on behalf of your organization to help build your business. For additional access needs, please work with your Partner Administrator. See you in the cloud, The Partner Advantage Team

The net takeaway is, on the Support Page (the first link on this slide) Google Cloud Partner Advantage Support, there’s a section **“Issue accessing Partner Advantage Portal? Click here for troubleshooting steps”**

- The source of their issue can be related to the different items shown
- Additionally, there’s a Partner Administrator / Partner Adminstrator Team at their partner organization that has to approve their access.. Until that step is completed, they will have access issues/limitation. They will need to identify who this person or team is at their organization

Program issues or concerns?

- Problems with **accessing** Cloud Skills Boost for Partners
  - cloud-partner-training@google.com
- Problems with **a lab** (locked out, etc.)
  - support@qwiklabs.com
- Problems with accessing Partner Advantage
  - https://support.google.com/googlecloud/topic/9198654



- Problems with accessing **Cloud Skills Boost for Partners**
  - cloud-partner-training@google.com
- Problems with a **lab** (locked out, etc.)
  - support@qwiklab.com
- Problems with accessing **Partner Advantage**
  - https://support.google.com/googlecloud/topic/9198654

Module 5

Scaling prototypes into custom ML models, Part 2

&

Serving and scaling models

Module Agenda

01 Distributed Training Strategies with TensorFlow

02 Choosing appropriate hardware for training: GPUs & TPUs

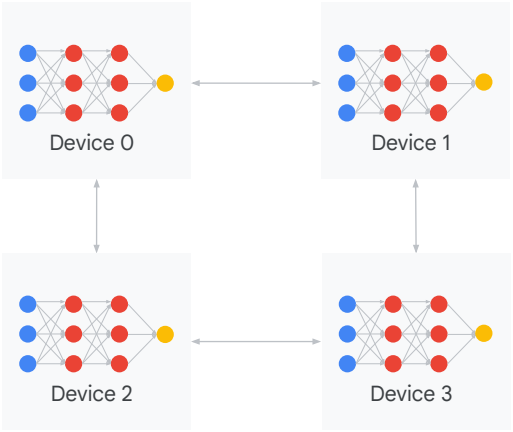
03 Serving models

04 Scaling online model serving

05 Vertex AI Feature Store

Distributed Training Strategies with TensorFlow

How can you make model training faster?



Source: Machine Learning on Google Cloud v3.0

If your training takes a few mins to a few hours, it will make you productive and happy. You can try out different ideas fast.

If it takes a few days, you could still deal with it by running a few things in parallel...

If it starts to take a week or more, your progress will slow down because you cannot try out new ideas quickly.

And.. if it takes more than a month, then i think it's not even worth thinking about...

And this is not an exaggeration.. Training deep neural networks such as ResNet50 can take upto a week on one GPU.

A natural question to ask is - how can you make training fast?

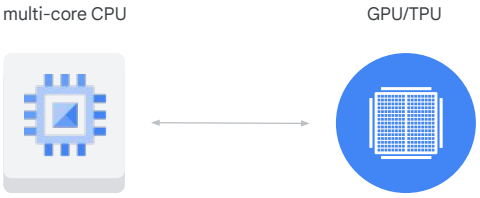
There are a number of things you can do to make training faster:

- You can use more powerful device such as TPUs or Tensor Processing Units.
- Your input pipeline might be slow, you can make that faster.

- There are in fact many performance guidelines on the TensorFlow website that you can try. We'll look at some of those guidelines in this module.

Let's look at distributed training - i.e. running training in parallel on many devices such as CPUs or GPUs or TPUs, in order to make your training faster.

Start training on a machine with a multi-core CPU, then add a single accelerator.

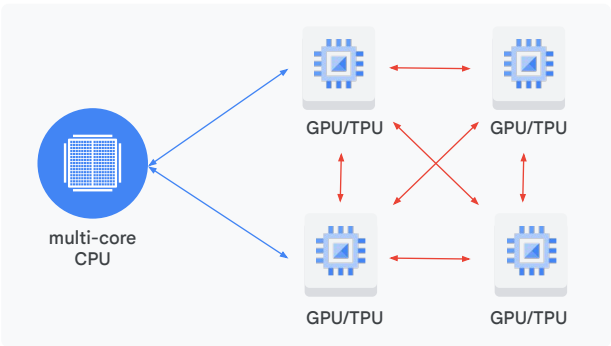


Source: Machine Learning on Google Cloud v3.0

Let’s say you start training on a machine with multi-core CPU. TensorFlow automatically handles scaling on multiple cores.

You may speed up your training by adding an accelerator to your machine such as GPU or TPU.

Add multiple accelerators to a single device with a distributed training framework like Horovod

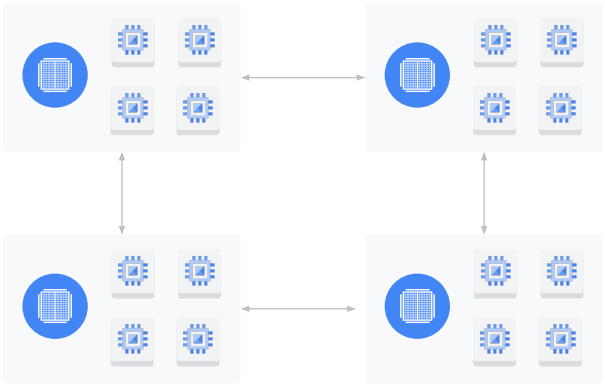


Source: Machine Learning on Google Cloud v3.0

With distributed training, you can go further. You can go from using one machine with a single device to a machine with multiple devices attached to it.

For this, you may likely use Horovod: https://github.com/horovod/horovod

Adding many machines with many possible devices



Source: Machine Learning on Google Cloud v3.0

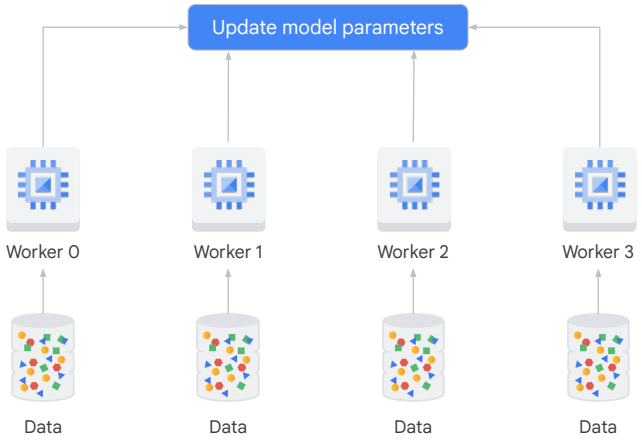
And finally to multiple machines with possibly multiple devices each, connected over a network.

Eventually, with various approaches you can scale up to 100s of devices (and that's indeed what we do in several google systems).

In the rest of this module we may use the term "device" or "worker" or "accelerator" interchangeably to refer to processing units such as GPUs or TPUs.

How does the distributed training actually work? Like everything else in software engineering, there are a few different ways to scale training. Which one you choose depends on the size of your model, amount of training data, and available devices.

Data parallelism



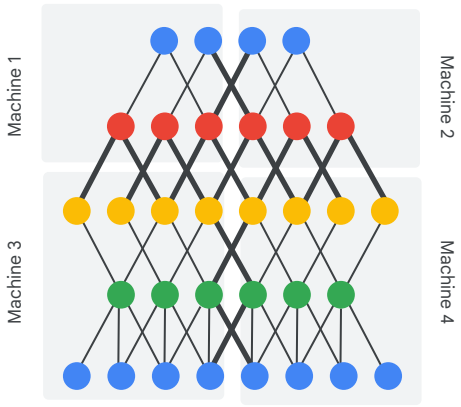
Source: Machine Learning on Google Cloud v3.0

The most common architecture for distributed training is known as 'data parallelism'.

In "data parallelism", you run the same model & computation on every device, but train each of them using different training samples.

Each device computes loss and gradients based on training samples it sees. Then we update the model's parameters using these gradients. The updated model is then used in the next round of computation.

Model parallelism



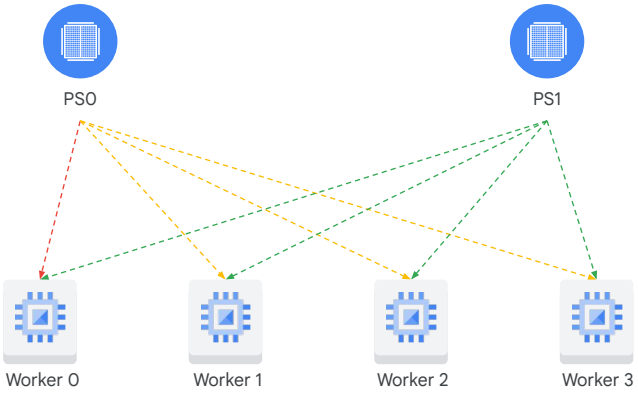
Source: Machine Learning on Google Cloud v3.0

Besides data parallelism, there is another type of distributed training called “model parallelism”.

A simple way to describe model parallelism is when your model is so big that it doesn’t fit on one device’s memory. So you divide it into smaller parts that compute over the same training samples on multiple devices.

For example, you could put different layers on different devices. We will also look at this in a later lesson.

Async parameter server



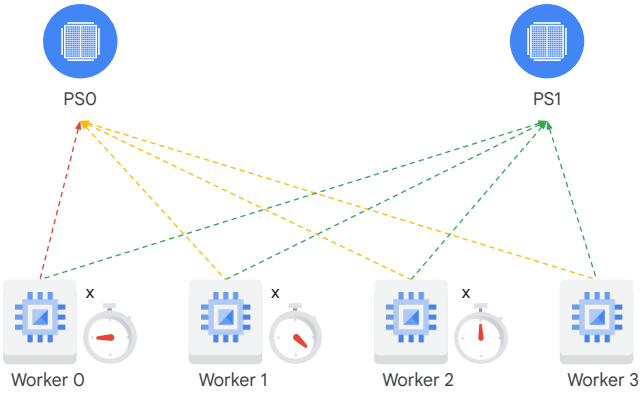
Source: Machine Learning on Google Cloud v3.0

There are currently 2 approaches used to update the model using gradients from various devices.

The first one is commonly known as the async parameter server architecture. In async parameter server architecture, some devices are designated to be parameter servers, and others as workers.

Each worker independently fetches the latest parameters from the PS and computes gradients based on a subset training samples.

Async parameter server



Source: Machine Learning on Google Cloud v3.0

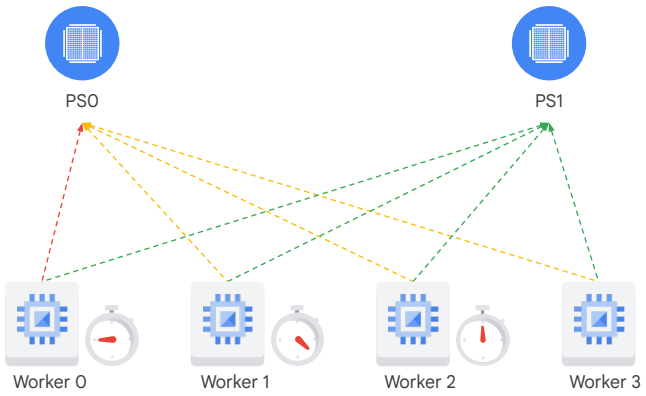
It then sends the gradients back to the PS. Which then updates its copy of the parameters with those gradients.

Each worker does this independently. This allows it to scale well to a large number of workers.

It has worked well for many models in Google, where training workers might be preempted by higher priority production jobs, or a machine may go down for maintenance, or where there is asymmetry between the workers.

These don’t hurt the scaling because workers are not waiting for each other.

Async parameter server



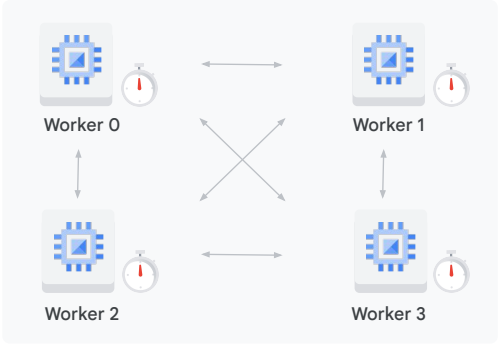
Source: Machine Learning on Google Cloud v3.0

The downside of this approach, however, is that workers get out of sync. They compute parameter updates based on stale values and this can delay convergence.

This architecture has been around for sometime, and it is what we did in the first specialization when we called the train_and_evaluate() method of the Estimator.

We'll look at this in more detail shortly.

Sync allreduce architecture



Source: Machine Learning on Google Cloud v3.0

The **second approach** is called synchronous allreduce.

With the rise of powerful accelerators such as TPUs and GPUs over the last few years, this approach has become common.

In this approach, each worker holds a copy of the model's parameters - there are no special servers holding the parameters.

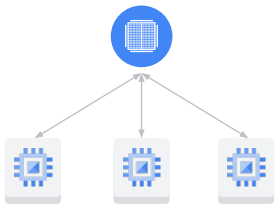
Each worker computes gradients based on the training samples they see and communicate between themselves to propagate the gradients and update their parameters.

All workers are synchronized - conceptually the next forward pass doesn't begin until each worker has received the gradients and updated their parameters.

With fast devices in a controlled environment, the variance between the step time on each worker is small. When combined with strong communication links between the workers, the overhead of synchronization is also small. So, overall this approach can lead to faster convergence.

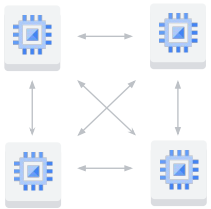
There isn't one right answer, but here are some considerations

Consider async parameter server if...



- Many low-power or unreliable workers.
- More mature approach.
- Constrained by I/O.

Consider synchronous approach if...



- Multiple devices on one host.
- Fast devices with strong links (e.g. TPUs).
- Better for multiple GPUs.
- Constrained by compute power.

Source: Machine Learning on Google Cloud v3.0

Given these 2 broad strategies, that is the asynchronous parameter server approach and the synchronous all reduce approach, when should you pick one over the other?

There isn't one right answer, but here are some considerations.

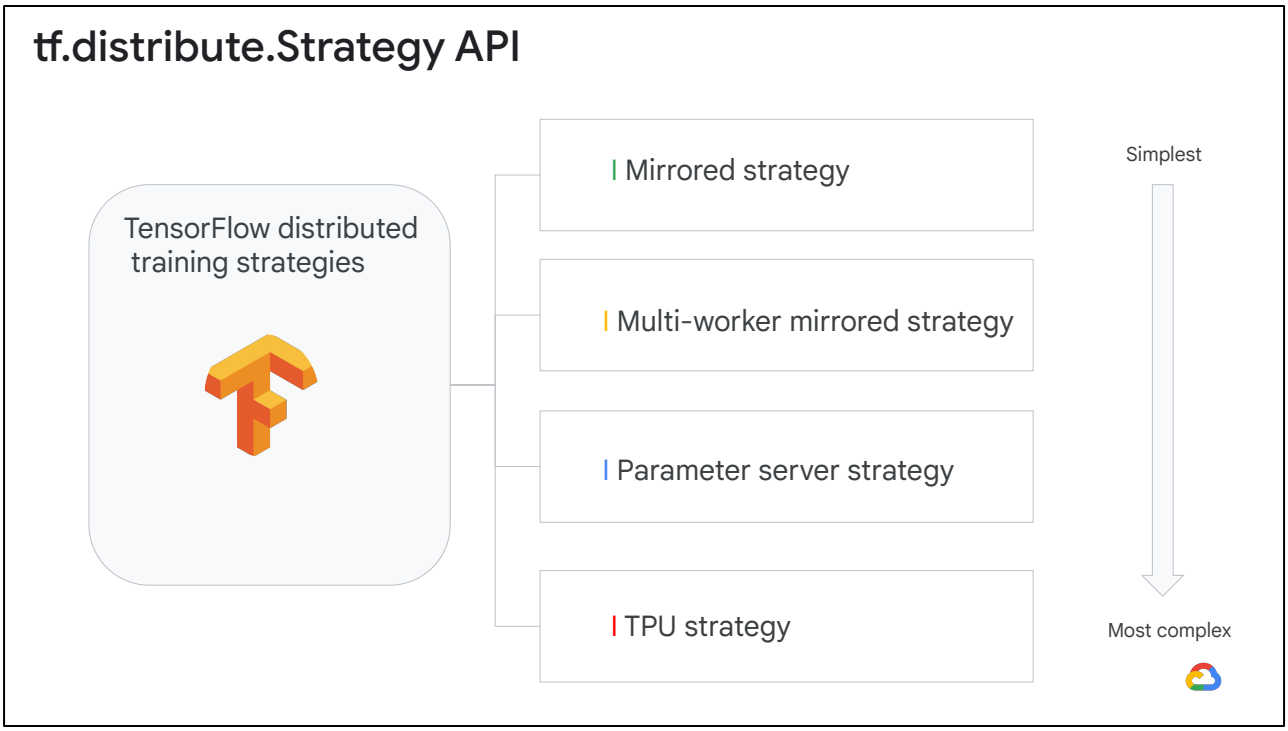
- Parameter server approach should be preferred when there are a large number of not-so-powerful or unreliable workers, such as a cluster of machines with just CPUs. If you have fast devices with strong communication links, such as multiple GPUs on one host, or TPUs, allreduce is a better choice. So, parameter-server for multiple-machines, all-reduce for multiple-devices on one machine.

- Another consideration is the relative maturity of the two approaches:
    - Parameter server approach is supported well by TensorFlow by the Estimator API's train_and_evaluate() method.
    - Async parameter server is definitely the more mature approach.

  The AllReduce method is gaining a lot more traction recently because of the improvements in hardware. For example, TPUs use the allreduce approach out of the box.

With recent improvements to TensorFlow, you can scale your training with all reduce on multiple GPUs. We will look at this later in this module, but all the code is still in tf.contrib, not core TensorFlow. So, the AllReduce may be better if you have machine with multiple GPUs or if you have a TPU but it is still experimental, and the APIs are subject to change.

- A third consideration is what performance constraint you will run into. If your model is primarily I/O bound, then using multiple machines is a better approach and you should use the parameter server approach. If your model is primarily compute bound, then you want to use more powerful processors and the AllReduce approach is better.



Source: Custom Screenshots from course Production Machine Learning Systems

`tf.distribute.Strategy` is a TensorFlow API to distribute training across multiple GPUs, multiple machines, or TPUs. Using this API, you can distribute your existing models and training code with minimal code changes.

`tf.distribute.Strategy` has been designed with these key goals in mind:

- Easy to use and support multiple user segments, including researchers, machine learning engineers, etc.
- Provide good performance out of the box.
- Easy switching between strategies.

You can distribute training using `tf.distribute.Strategy` with a high-level API like Keras `Model.fit`, as well as custom training loops (and, in general, any computation using TensorFlow).

In TensorFlow 2.x, you can execute your programs eagerly, or in a graph using `tf.function`. `tf.distribute.Strategy` intends to support both these modes of execution, but works best with `tf.function`. Eager mode is only recommended for debugging purposes and not supported for `tf.distribute.TPUStrategy`. Although training is the focus of this guide, this API can also be used for distributing evaluation and prediction on different

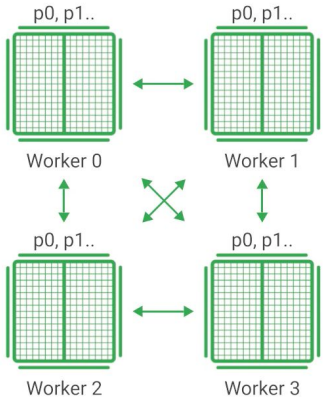
You can use `tf.distribute.Strategy` with very few changes to your code, because the underlying components of TensorFlow have been changed to become strategy-aware. This includes variables, layers, models, optimizers, metrics, summaries, and checkpoints.

In this guide, you will learn about various types of strategies and how you can use them in different situations. To learn how to debug performance issues, check out the Optimize TensorFlow GPU performance guide.

There are four TensorFlow distributed training strategies that support data parallelism.

Mirrored Strategy

- Synchronous Strategy one machine with many accelerators
- Creates a replica of the model on each GPU
- Mini batch is split into n
- Data distribution and Gradient updates are automatically updated

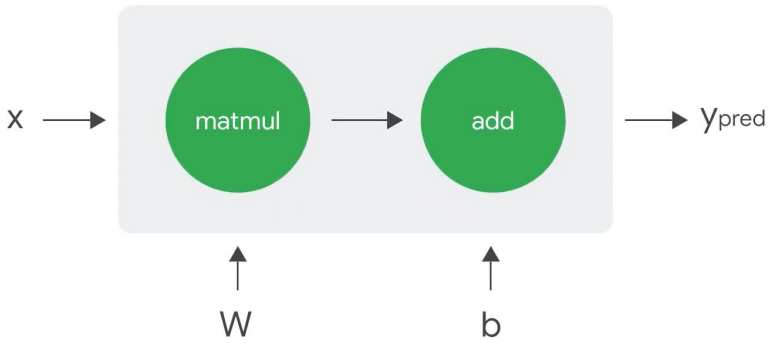


Source: Custom Screenshots from course Production Machine Learning Systems

Mirrored strategy is the simplest way to get started with distributed training. You can use mirrored strategy when you have a single machine with multiple GPU devices. Mirrored strategy will create a replica of the model on each GPU. During training, one minibatch is split into n parts, where "n" equals the number of GPUs, and each part is fed to one GPU device. For this setup, mirrored strategy manages the coordination of data distribution and gradient updates across all of the GPUs.. From there, map, shuffle, and prefetch the data. You then call model.fit on the training data. Here we're going to run five passes of the entire training dataset. Now let's explain what actually happens when we call model.fit before adding a strategy. For simplicity, imagine you have a simple linear model instead of the ResNet 50 architecture. For more information on optimization, please refer to the guide titled, "Optimize

TensorFlow GPU performance using the Profiler" at tensorflow.org/guide/profiler.

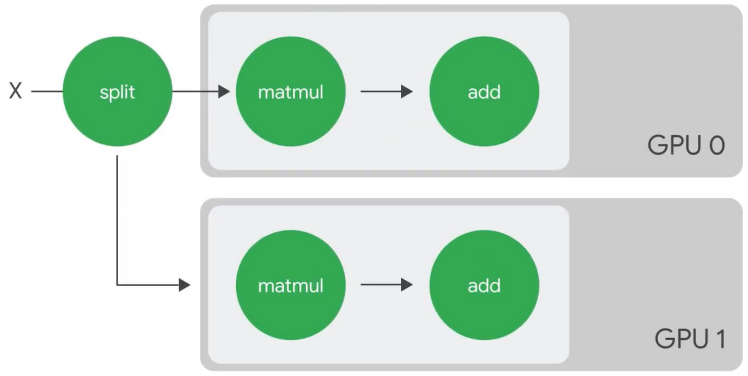
DAG without Mirrored Strategy



Source: Custom Screenshots from course Production Machine Learning Systems

In TensorFlow, you can think of this simple model in terms of its computational graph or directed acyclic graph, here referred to as a DAG. Here the matmul op takes in the x and W tensors, which are the training batch and weights, respectively. The resulting tensor is then passed to the add op with the tensor b, which is the model's bias terms.

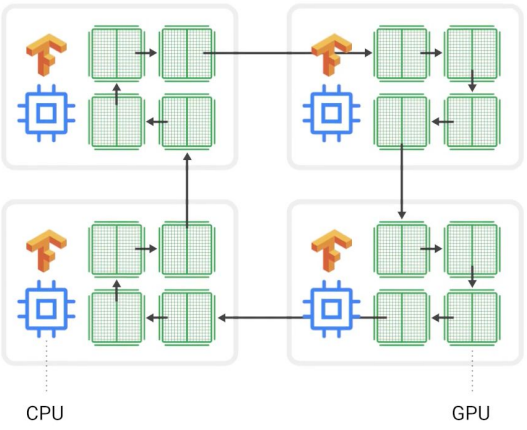
DAG with Mirrored Strategy



Source: Custom Screenshots from course Production Machine Learning Systems

Now this is an example of data parallelism with two GPUs. The input batch, x, is split in half, and one slice is sent to GPU 0 and the other to GPU 1. In this case, each GPU calculates the same ops, but on different slices of the data.

Multi-worked Mirrored Strategy



Source: Custom Screenshots from course Production Machine Learning Systems

Multi-worker mirrored strategy is very similar to mirrored strategy. It implements synchronous distributed training across multiple workers, each with potentially multiple GPUs. Similar to mirrored strategy, it creates copies of all variables in the model on each device across all workers. If you've mastered single-host training and are looking to scale training even further, then adding multiple machines to your cluster can help you get an even greater performance boost.

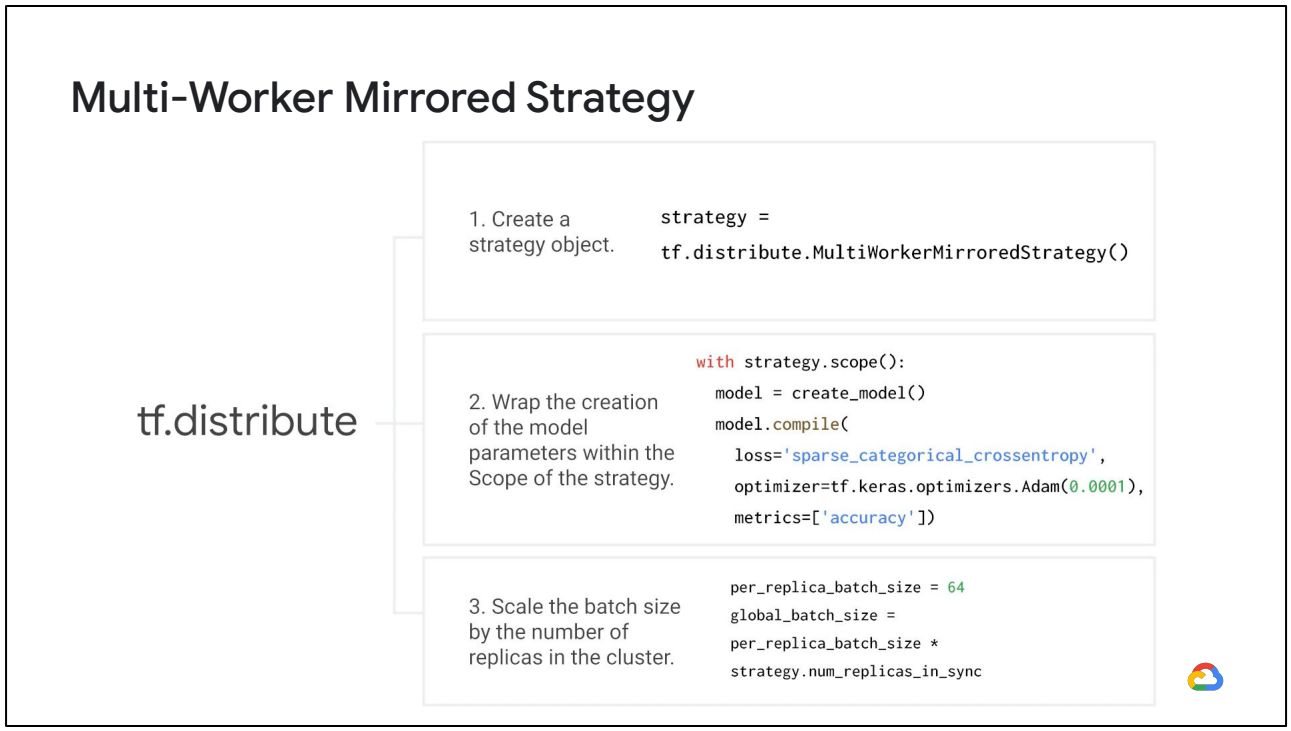
Multi-worked Mirrored Strategy

```
os.environ["TF_CONFIG"] = json.dumps({
   "cluster": {
       "chief": ["host1:port"],
       "worker": ["host2:port", "host3:port"],
   },
```

Source: Custom Screenshots from course Production Machine Learning Systems

You can make use of a cluster of machines that are CPU only or that each have one or more GPUs. Like its single-worker counterpart, mirrored strategy, multi-worker mirrored strategy is a synchronous data parallelism strategy that can be used with only a few code changes. However, unlike mirrored strategy, for a multi-worker setup, TensorFlow needs to know which machines are part of the cluster. However, unlike mirrored strategy, for a multi-worker setup, TensorFlow needs to know which machines are part of the cluster. In most cases, this is specified with the environment variable "TF_CONFIG". In this simple TF_CONFIG example, the "cluster" key contains a dictionary with the internal IPs and ports of all the machines. In multi-worker mirrored strategy, all machines are designated as "workers", which are the physical machines on which the replicated computation is executed. In addition to each machine being a worker, there needs to be

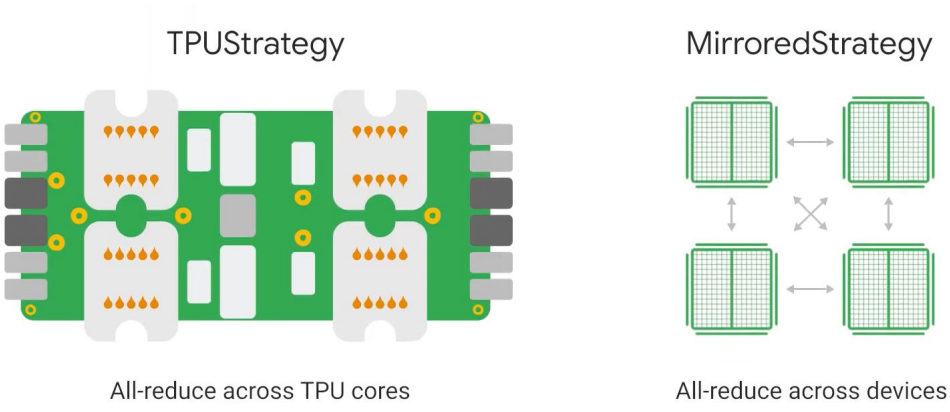
one worker that takes on some extra work, such as saving checkpoints and writing summary files to TensorBoard. This machine is known as the "chief", or by its deprecated name, "master".



Source: Custom Screenshots from course [Production Machine Learning Systems]

As with any strategy in the tf.distribute module, step one is to create a strategy object. Step two is to wrap the creation of the model parameters within the Scope of the strategy. This is crucial, because it tells mirrored strategy which variables to mirror across the GPU devices. And the third and final step is to scale the batch size by the number of replicas in the cluster

TPU Strategy



Source: Custom Screenshots from course [Production Machine Learning Systems]

Similar to MirroredStrategy, TPUStrategy uses a single machine where the same model is replicated on each core with its variables synchronized-- mirrored-- across each replica of the model. The main difference, however, is that TPUStrategy will all-reduce across TPU cores, whereas MirroredStrategy will all-reduce across GPUs. Tf.distribute. TPUStrategy lets you run your TensorFlow training on Tensor Processing Units--TPUs. TPUs are Google's specialized ASICs designed to dramatically accelerate machine learning workloads. TPUs provide their own implementation of efficient all-reduce and other collective operations across multiple TPU cores which are used in TPUStrategy

TPU Strategy Implementation

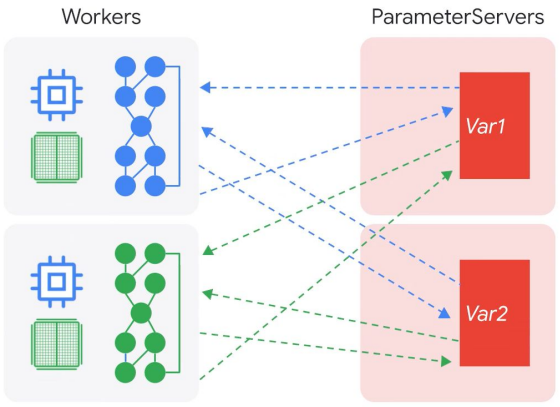
```
strategy = tf.distribute.TPUStrategy()


with strategy.scope():
  model = tf.keras.Sequential(...)
  model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(0.0001),
    metrics=['accuracy'])
```

Source: Custom Screenshots from course Production Machine Learning Systems

You'll also need a variable called "strategy." But this time, you will choose the tf.distrubute. TPFStrategy method. Because TPUs are very fast, many models ported to the TPU end up with a data bottleneck. The TPU is sitting idle, waiting for data for the most part of each training epoch. TPUs read training data exclusively from Google Cloud Storage-- GCS. And GCS can sustain a pretty large throughput if it is continuously streaming from multiple files in parallel. Following best practices will optimize the throughput. With too few files, GCS will not have enough streams to get max throughput. With too many files, time will be wasted accessing each individual file. Let's summarize the distribution strategies using code. Our base scope is a Keras Sequential model. Now, to improve training, we can use the MirroredStrategy. Or for faster training, the MultiWorkerMirroredStrategy. And for really fast training, the TPUStrategy.

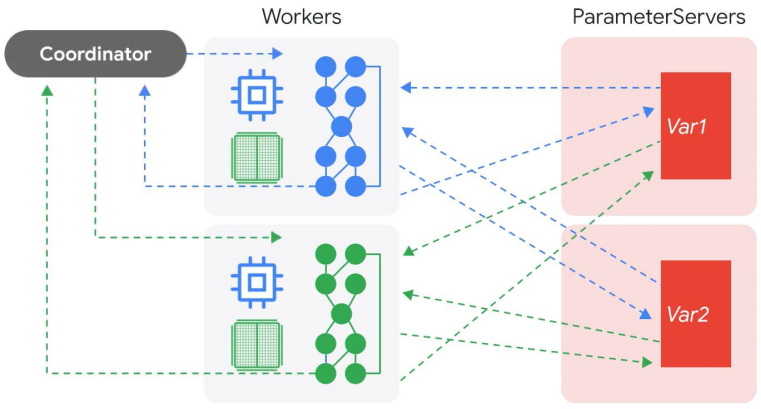
Parameter Server Strategy



Source: Custom Screenshots from course [Production Machine Learning Systems]

we explored the asynchronous parameter server architecture. A parameter server training cluster consists of Workers and ParameterServers. Variables are created on ParameterServers, and they are read and updated by Workers in each step. By default, Workers read and update these variables independently without synchronizing with each other.

Parameter Server Strategy with Coordinator



Source: Custom Screenshots from course [Production Machine Learning Systems]

The TensorFlow parameter server strategy introduces a central coordinator. The Coordinator is a special task type that creates resources, dispatches training tasks, writes checkpoints and deals with task failures.

Choosing appropriate hardware for training: GPUs & TPUs

GPUs

- Models with a significant number of custom TensorFlow/PyTorch/JAX operations that must run at least partially on CPUs
- Models with TensorFlow ops that are not available on Cloud TPU (see the list of available TensorFlow ops)
- Medium-to-large models with larger effective batch sizes

TPUs

- Models dominated by matrix computations
- Models with no custom TensorFlow/PyTorch/JAX operations inside the main training loop
- Models that train for weeks or months
- Large models with large effective batch sizes

Source: Introduction to Cloud TPU

Show this demo of how they work.

CPU (central processing unit)
GPU (graphics processing unit)
TPU (tensor processing unit)

CPUs

- Quick prototyping that requires maximum flexibility
- Simple models that do not take long to train
- Small models with small, effective batch sizes
- Models that contain many custom TensorFlow operations written in C++
- Models that are limited by available I/O or the networking bandwidth of the host system

GPUs

- Models with a significant number of custom TensorFlow/PyTorch/JAX operations that must run at least partially on CPUs
- Models with TensorFlow ops that are not available on Cloud TPU (see the list of available TensorFlow ops)
- Medium-to-large models with larger effective batch sizes

TPUs

- Models dominated by matrix computations
- Models with no custom TensorFlow/PyTorch/JAX operations inside the main

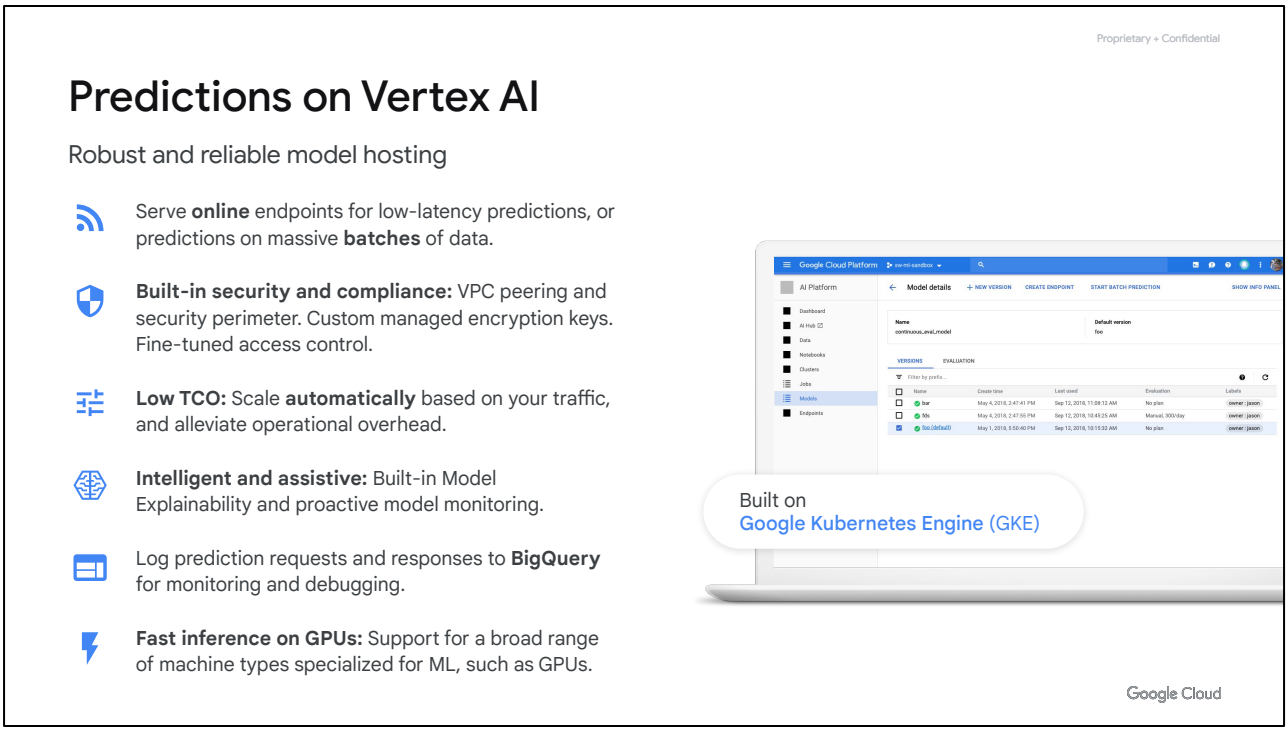
- training loop
- Models that train for weeks or months
- Large models with large effective batch sizes

Cloud TPUs are not suited to the following workloads:

- Linear algebra programs that require frequent branching or contain many element-wise algebra operations
- Workloads that access memory in a sparse manner
- Workloads that require high-precision arithmetic
- Neural network workloads that contain custom operations in the main training loop

Introduction to TPUs

Serving models



Vertex AI offers two methods for getting prediction:

- Online predictions are synchronous requests made to a model endpoint. Before sending a request, you must first deploy the model resource to an endpoint. This associates compute resources with the model so that it can serve online predictions with low latency. Use online predictions when you are making requests in response to application input or in situations that require timely inference.
- Batch predictions are asynchronous requests. You request a batchPredictionsJob directly from the model resource without needing to deploy the model to an endpoint. Use batch predictions when you don't require an immediate response and want to process accumulated data by using a single request.

These can be used for AutoML models, custom trained models, or BigQuery ML models.

For online endpoints, you can split traffic so a new version gets 10% of your traffic as you roll out an updated model.

You can host regular or private endpoints, which allow you to peer your network to a secure VPC containing the endpoints.

When serving custom models, you can provide your own container, or Vertex AI

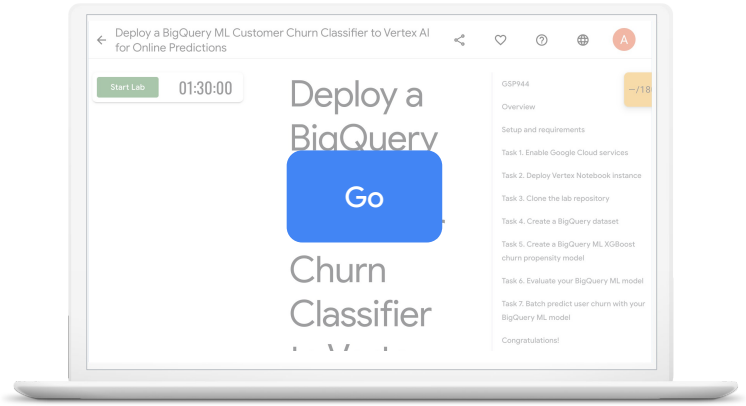
provides containers to serve common frameworks: scikit-learn, XGBoost, TensorFlow, or PyTorch.

From: Overview of getting predictions on Vertex AI

Source: MLOps Architectures with CI/CD (slides) | Y22
https://docs.google.com/presentation/d/1OiGZOviGJf6XX7E91-41PGBy3aoRPZcibmF9dtH1Cms/edit?resourcekey=0-0LF8xS6di73GKb_vAVDZjw#slide=id.g1043fd8a725_0_2031

Recommended Lab

Deploy a BigQuery ML Customer Churn Classifier to Vertex AI for Online Predictions



From the Course
Build and Deploy Machine Learning Solutions on Vertex AI

Google Cloud

[Deploy a BigQuery ML Customer Churn Classifier to Vertex AI for Online Predictions]
or if less time
[Building Demand Forecasting with BigQuery ML]

Scaling online model serving

Optimizing Model Performance in Production

Why optimize models?

- Size reduction
  - Smaller storage size
  - Smaller download size to users' devices
  - Less memory usage
- Latency reduction
- Accelerator compatibility for more limited accelerators on edge devices, like Edge TPUs

Docs: [TensorFlow Model Optimization]

Types of optimization

- **Quantization** works by reducing the precision of the numbers used to represent a model's parameters, which by default are 32-bit floating point numbers. This results in a smaller model size and faster computation.
- **Pruning** works by removing parameters within a model that have only a minor impact on its predictions. Pruned models are the same size on disk, and have the same runtime latency, but can be compressed more effectively. This makes pruning a useful technique for reducing model download size.
- **Clustering** works by grouping the weights of each layer in a model into a predefined number of clusters, then sharing the centroid values for the weights belonging to each individual cluster. This reduces the number of unique weight values in a model, thus reducing its complexity.

Quantization works by reducing the precision of the numbers used to represent a model's parameters, which by default are 32-bit floating point numbers. This results in a smaller model size and faster computation.

Pruning works by removing parameters within a model that have only a minor impact on its predictions. Pruned models are the same size on disk, and have the same runtime latency, but can be compressed more effectively. This makes pruning a useful technique for reducing model download size. In the future, TensorFlow Lite will provide latency reduction for pruned models.

Clustering works by grouping the weights of each layer in a model into a predefined number of clusters, then sharing the centroid values for the weights belonging to each individual cluster. This reduces the number of unique weight values in a model, thus reducing its complexity. As a result, clustered models can be compressed more effectively, providing deployment benefits similar to pruning.

Docs: TensorFlow Model Optimization

What does it look like to apply optimization?

```
import tensorflow as tf
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quant_model = converter.convert()
```

The default dynamic range quantization converts to 8-bit integers.

Quantization works by reducing the precision of the numbers used to represent a model's parameters, which by default are 32-bit floating point numbers. This results in a smaller model size and faster computation.

Pruning works by removing parameters within a model that have only a minor impact on its predictions. Pruned models are the same size on disk, and have the same runtime latency, but can be compressed more effectively. This makes pruning a useful technique for reducing model download size. In the future, TensorFlow Lite will provide latency reduction for pruned models.

Clustering works by grouping the weights of each layer in a model into a predefined number of clusters, then sharing the centroid values for the weights belonging to each individual cluster. This reduces the number of unique weight values in a model, thus reducing its complexity. As a result, clustered models can be compressed more effectively, providing deployment benefits similar to pruning.

Docs: TensorFlow Model Optimization

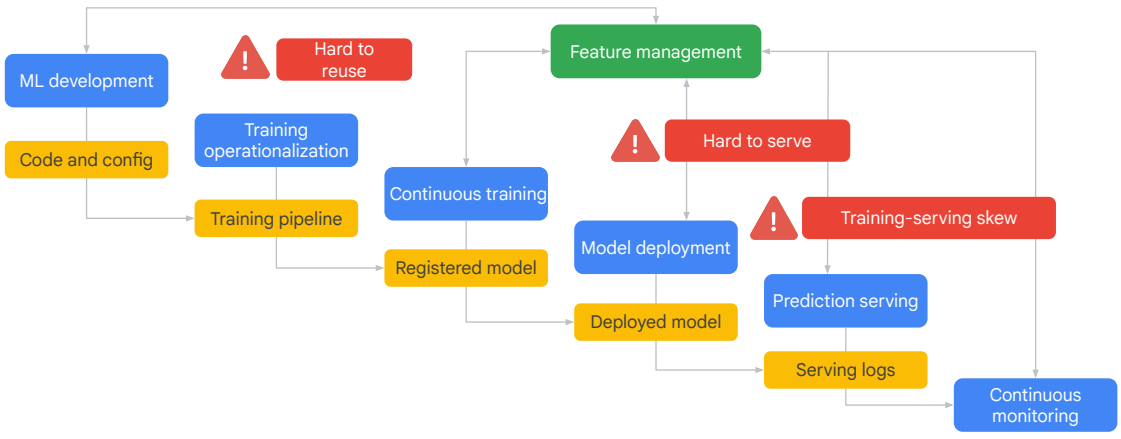
Vertex AI Feature Store

LAB: Using Feature Store (2 hrs)

Learning objectives:

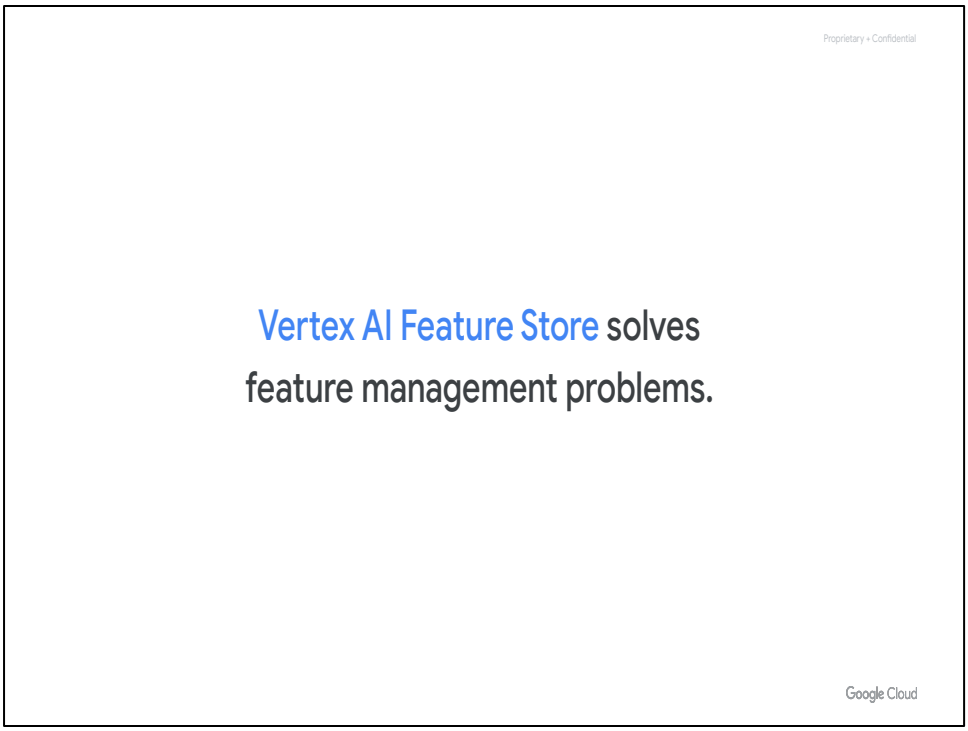
- Import your features into Feature Store.
- Serve online prediction requests using the imported features.
- Access imported features in offline jobs, such as training jobs.

Feature management pain points



These three key challenges are all considered feature management pain points.

https://cloud.google.com/blog/topics/developers-practitioners/kickstart-your-organizations-ml-application-development-flywheel-vertex-feature-store



How can the expanding machine learning team of XYZ solve these challenges? Vertex AI Feature Store solves these feature management problems.

Vertex Feature Store

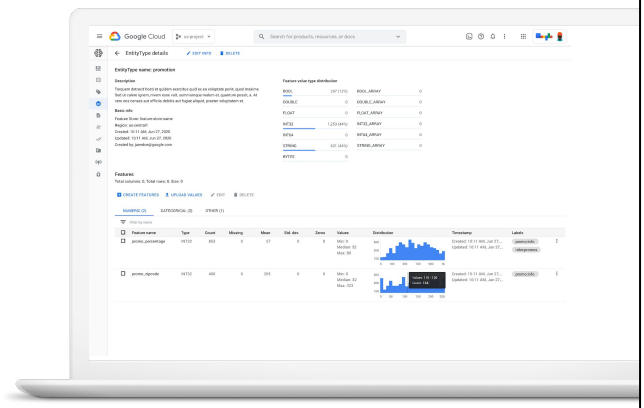
A rich feature repository to serve, share and re-use ML features.

Share and reuse ML features across use cases
Centralized feature repository with easy APIs to search and discover features, fetch them for training/serving, and manage permissions.

Serve ML features at scale with low latency
Offload the operational overhead of handling infrastructure for low latency scalable feature serving.

Alleviate training serving skew

- Compute feature values once, re-use for training and serving
- Track and monitor for drift and other quality issues



Source: MLOps Architectures with CI/CD (slides) | Y22
https://docs.google.com/presentation/d/1OiGZOviGJf6XX7E91-41PGBy3aoRPZcibmF9dtH1Cms/edit?resourcekey=0-0LF8xS6di73GKb_vAVDZjw#slide=id.g1043fd8a725_0_2031

Vertex AI Feature Store provides a centralized repository for organizing, storing, and serving ML features. Using a central featurestore enables an organization to efficiently share, discover, and re-use ML features at scale, which can increase the velocity of developing and deploying new ML applications.

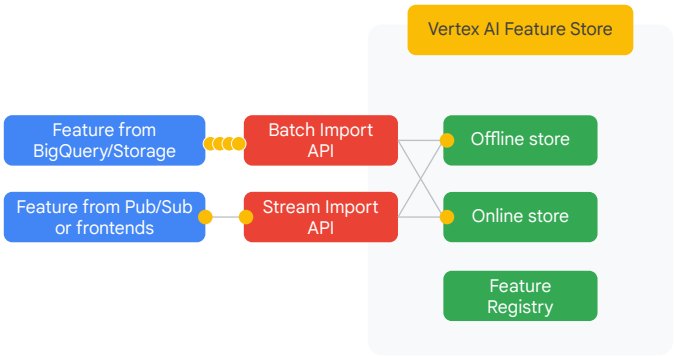
Vertex AI Feature Store is a fully managed solution, which manages and scales the underlying infrastructure such as storage and compute resources. This solution means that data scientists can focus on the feature computation logic instead of worrying about the challenges of deploying features into production.

Vertex AI Feature Store is an integrated part of Vertex AI. You can use Vertex AI Feature Store independently or as part of Vertex AI workflows. For example, you can fetch data from Vertex AI Feature Store to train custom or AutoML models in Vertex AI. Use Vertex AI Feature Store to create and manage resources, such as a featurestore. A featurestore is a top-level container for your features and their values. When you set up a featurestore, permitted users can add and share their features without additional engineering support. Users can define features and then ingest

(import) feature values from various data sources.

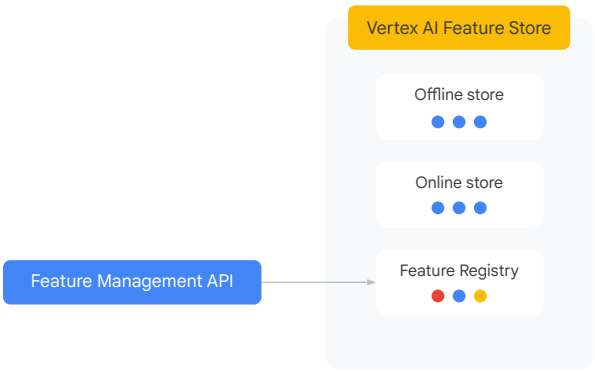
Any permitted user can search and retrieve values from the featurestore. For example, you can find features and then do a batch export to get training data for ML model creation. You can also retrieve feature values in real time to perform fast online predictions.

With Vertex AI Feature Store, you can store features with Batch and Stream Import APIs...



With Vertex AI Feature Store, the team can store features with Batch and Stream Import APIs

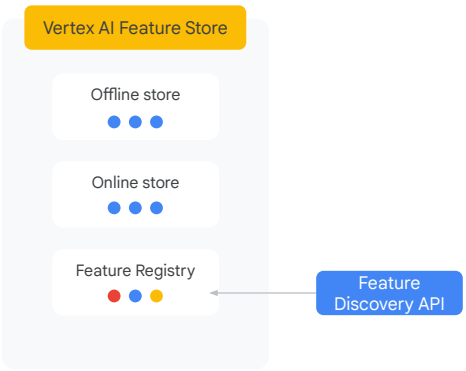
...and register the feature to its Feature Registry.



and register the feature to its Feature Registry.

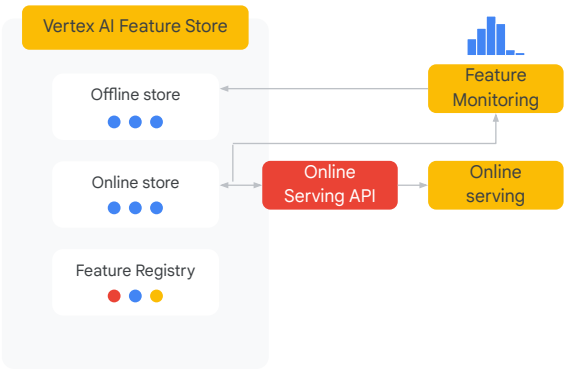
Reason for a Feature Registry?

So, other researchers and ML engineers can easily find the feature with Discovery API...



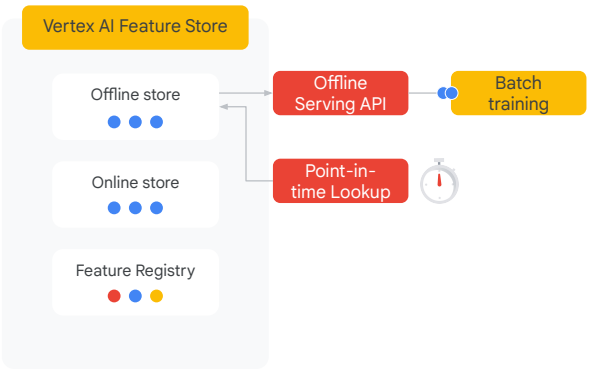
This allows other team members, such as the data analysts, ML engineers, the software developer and the data scientist, to easily find the feature with Discovery API

...and retrieve the feature value for fast online serving with continuous Feature Monitoring.



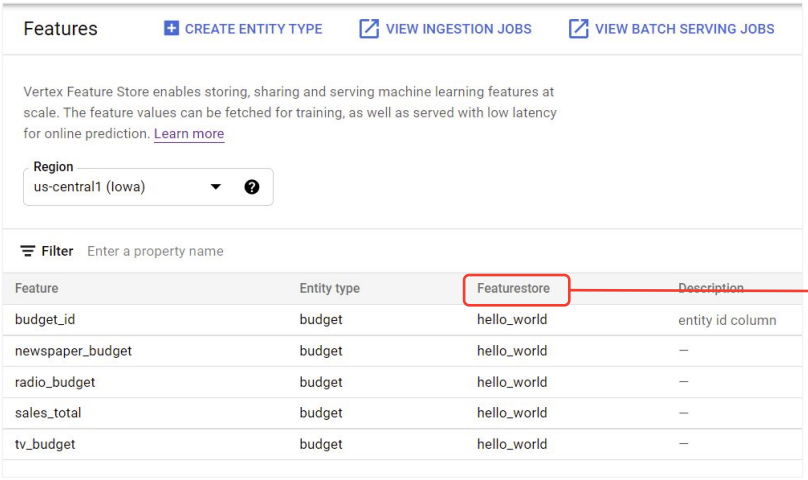
and retrieve the feature value for fast online serving with continuous Feature Monitoring.

Also, retrieve feature batches for training jobs, with point-in-time lookup to prevent data leaks.



They can also retrieve feature batches for training jobs with Point-in-time Lookup to prevent data leakage.

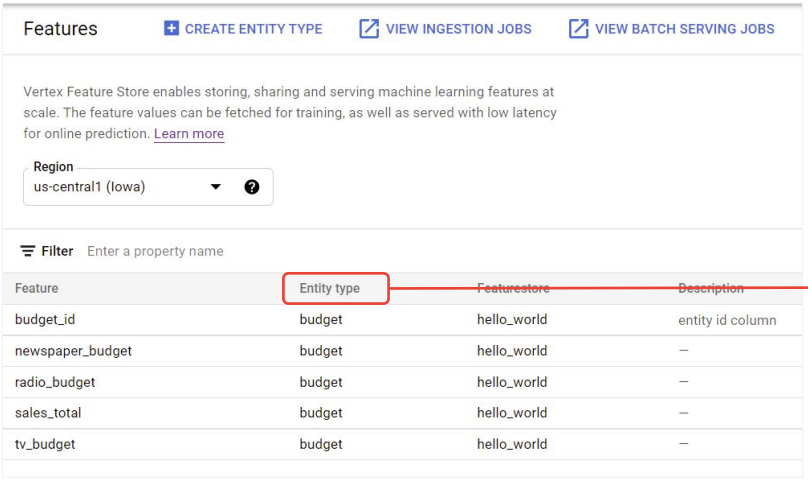
Feature Store



- Feature Store is a top-level container for features and their values.
- Permitted users can add and share their features.
- Users can define features and ingest values from various sources.

A featurestore is a top-level container for your features and their values. When you set up a featurestore, permitted users can add and share their features without additional engineering support. Users can define features and then ingest (import) feature values from various data sources.

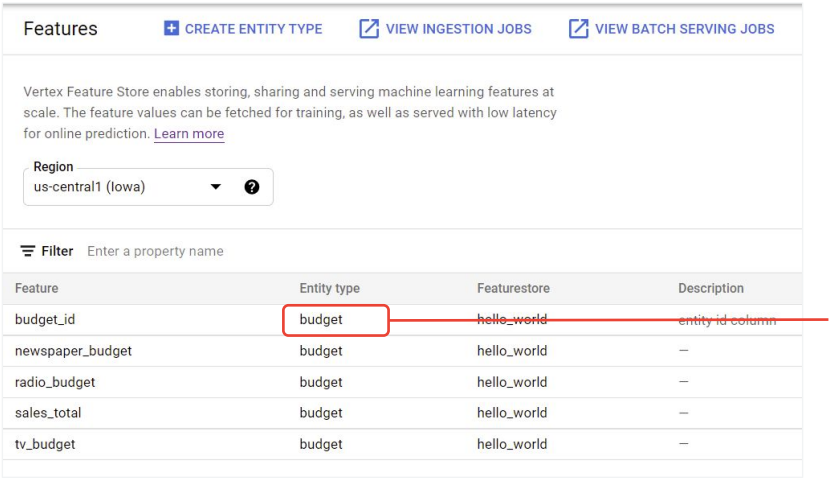
Entity type



- An *entity type* is a collection of semantically related features.
- You define your own entity types based on the concepts that are relevant to your use case.

An entity type is a collection of semantically related features. You define your own entity types based on the concepts that are relevant to your use case. For example, a movie service might have the entity types movies and users that group related features that correspond to movies or customers.

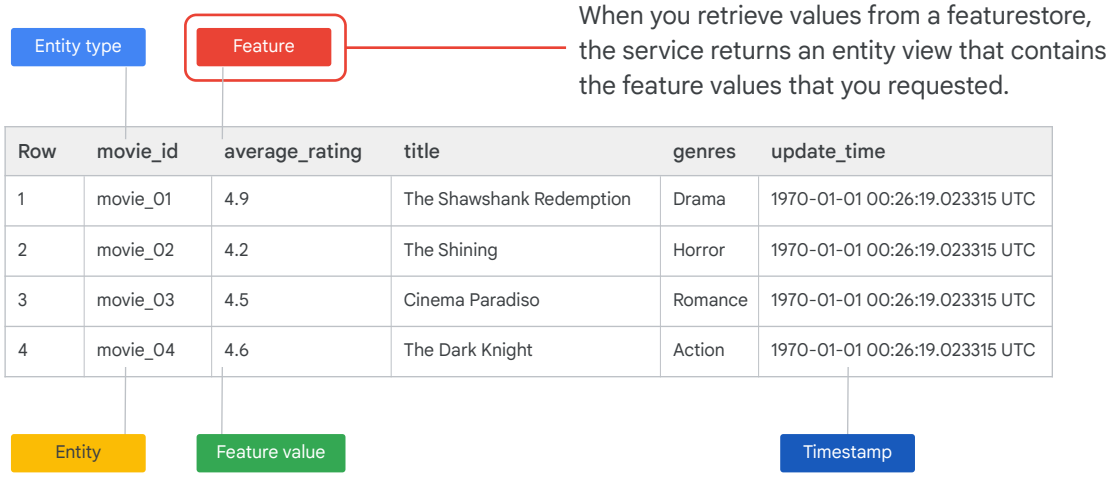
Entity



- An *entity* is an instance of an entity type.
- *Movie_01* and *movie_02* are entities of the *movies* entity type.
- Each entity must have a unique ID and must be of type STRING.

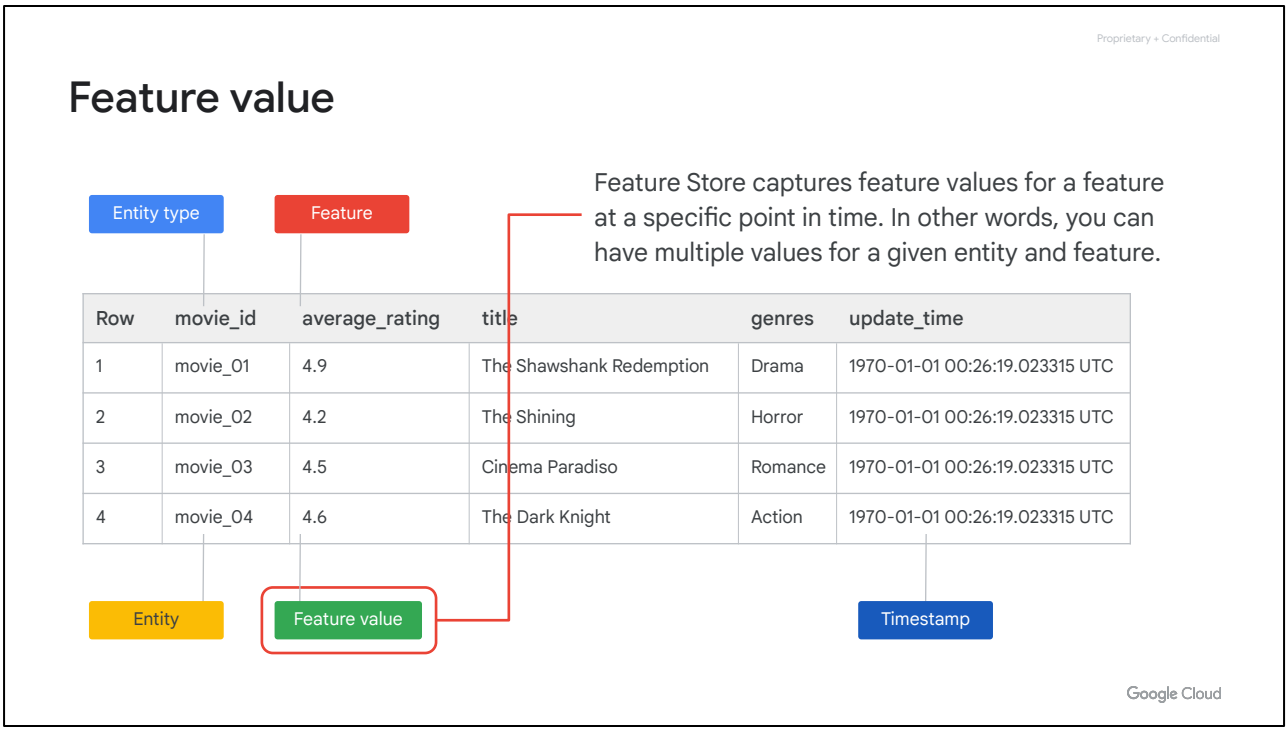
An entity is an instance of an entity type. For example, movie_01 and movie_02 are entities of the movies entity type. In a featurestore each entity must have a unique ID and must be of type STRING.

Entity view



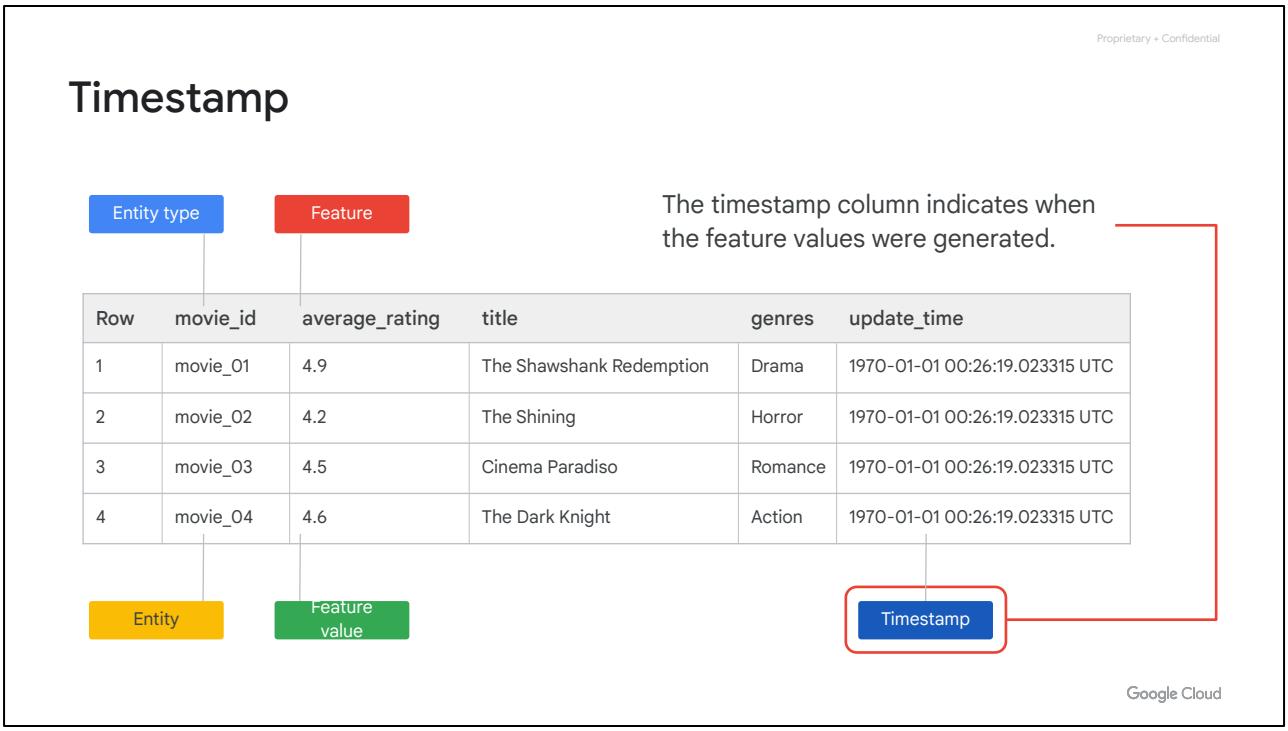
When you retrieve values from a featurestore, the service returns an entity view that contains the feature values that you requested. You can think of an entity view as a projection of the features and values that Feature Store returns from an online or batch serving request:

- For online serving requests, you can get all or a subset of features for a particular entity type.
- For batch serving requests, you can get all or a subset of features for one or more entity types. For example, if features are distributed across multiple entity types, you can retrieve them together in a single request that you can feed to a machine learning or batch prediction request.



4: Feature Engineering

Feature Store captures feature values for a feature at a specific point in time. In other words, you can have multiple values for a given entity and feature. For example, the movie_01 entity can have multiple feature values for the average_rating feature. The value can be 4.4 at one time and 4.8 at some later time. Feature Store associates a tuple identifier with each feature value (entity_id, feature_id, timestamp), which it then uses to look up values at serving time.



4: Feature Engineering

The timestamp column indicates when the feature values were generated. In the featurestore, the timestamps are an attribute of the feature values, not a separate resource type. If all feature values were generated at the same time, you are not required to have a timestamp column. You can specify the timestamp as part of your ingestion request.

Feature Store keeps feature values up to the data retention limit. This limit is based on the timestamp associated with the feature values, not when the values were imported.

Summary: Feature list

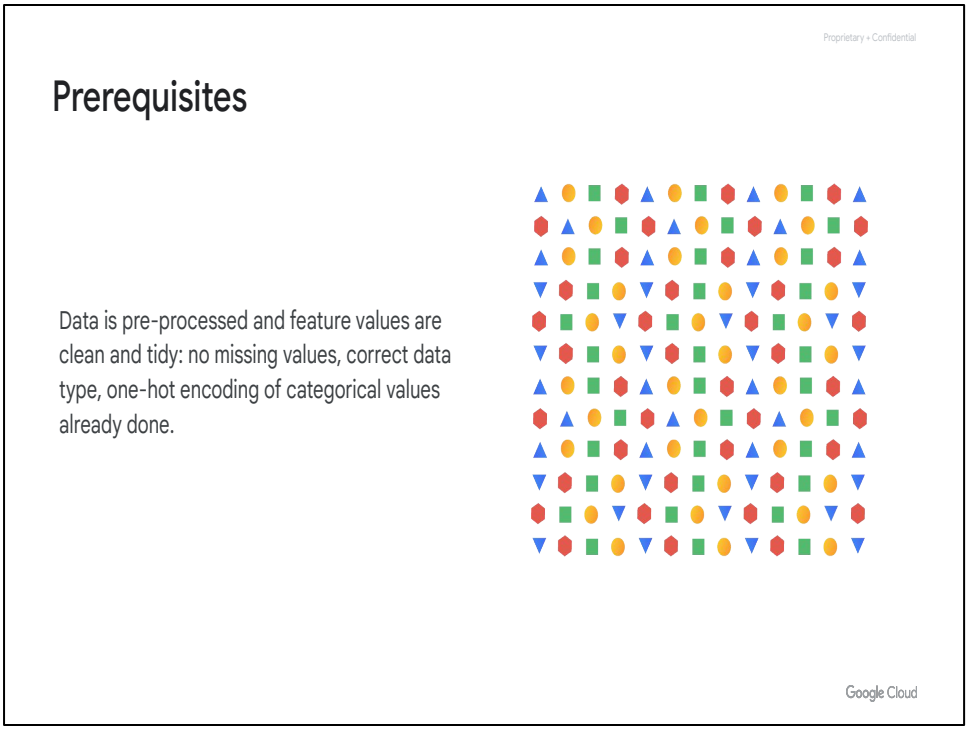
Filter Enter a property name

Feature	Type	Count	Missing	Mean	Std. dev.	Zeros	Values	Most common	Unique	Distribution
budget_id	STRING	1,200	0	–	–	–	–	"budget_999" Frequency: 2	1,200	
newspaper_budget	INT64	1,200	0	30.1	21.715	12	Min: 0 Median: 25 Max: 114	–	–	
radio_budget	INT64	1,200	0	22.79	14.789	24	Min: 0 Median: 23 Max: 49	–	–	
sales_total	INT64	1,200	0	13.54	5.24	0	Min: 1 Median: 12 Max: 27	–	–	
tv_budget	INT64	1,200	0	146.57	85.61	6	Min: 0 Median: 149 Max: 296	–	–	

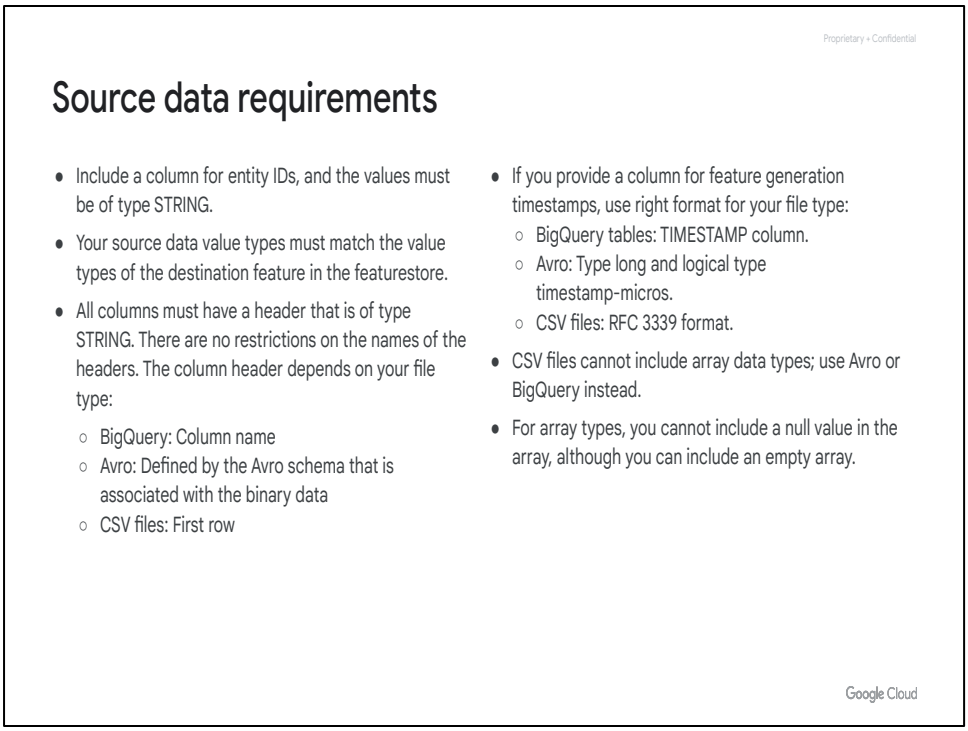
Let’s examine an entity view that shows a list of five features for the entity type budget in the featurestore “hello world.”

Summary: Feature Store data model

The Feature Store data model includes an entity type, entities, features, and feature values.



Before creating a featurestore, you'll need to pre-process your data. Ensure that your features are "clean and tidy," which means that there are no missing values, datatypes are correct, and any one-hot encoding of categorical values has already been done.

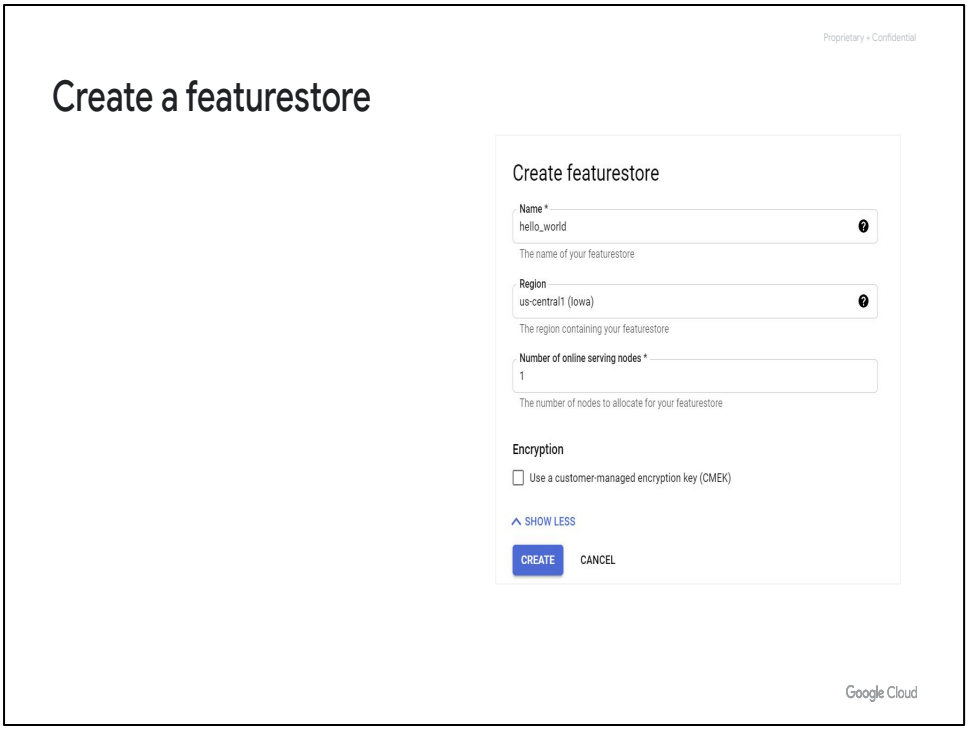


There are some requirements for your source data. Vertex AI Feature Store can ingest data from tables in BigQuery or files in Cloud Storage. For files in Cloud Storage, they must be in the Avro or CSV format:

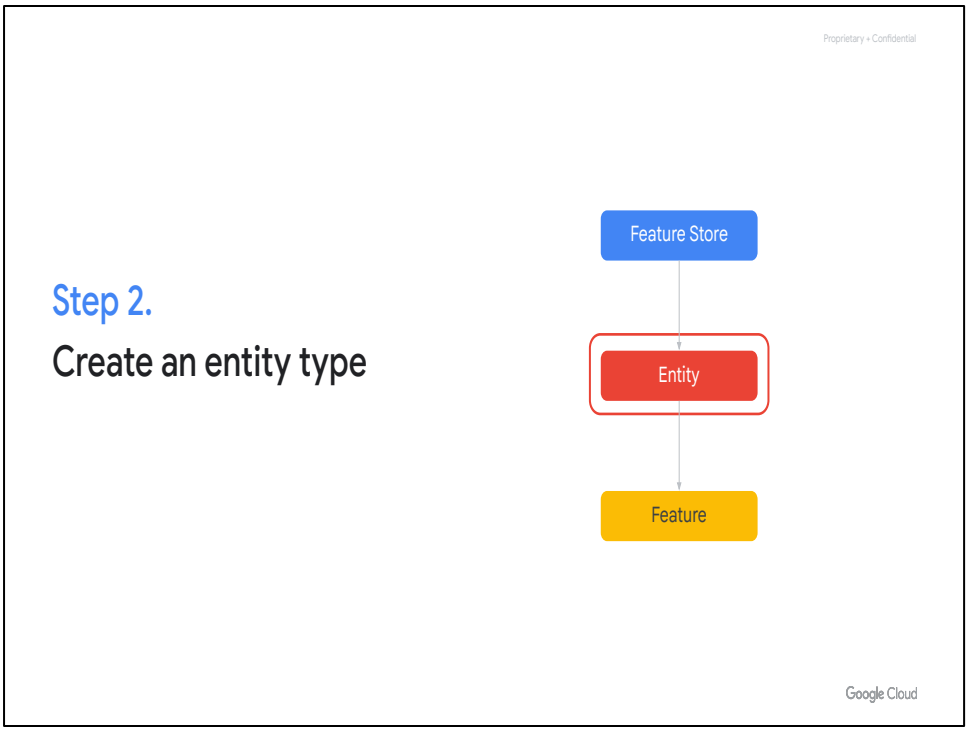
- You must have a column for entity IDs, and the values must be of type STRING. This column contains the entity IDs that the feature values are for.
- Your source data value types must match the value types of the destination feature in the featurestore. For example, boolean values must be ingested into a feature that is of type BOOL.
- All columns must have a header that is of type STRING. There are no restrictions on the names of the headers.
    - For BigQuery tables, the column header is the column name.
    - For Avro, the column header is defined by the Avro schema that is associated with the binary data.
    - For CSV files, the column header is the first row.
- If you provide a column for feature generation timestamps, use one of the following timestamp formats:
    - For BigQuery tables, timestamps must be in the TIMESTAMP column.
    - For Avro, timestamps must be of type long and logical type timestamp-micros.
    - For CSV files, timestamps must be in the RFC 3339 format.
- CSV files cannot include array data types. Use Avro or BigQuery instead.
- For array types, you cannot include a null value in the array, although you can

- include an empty array.

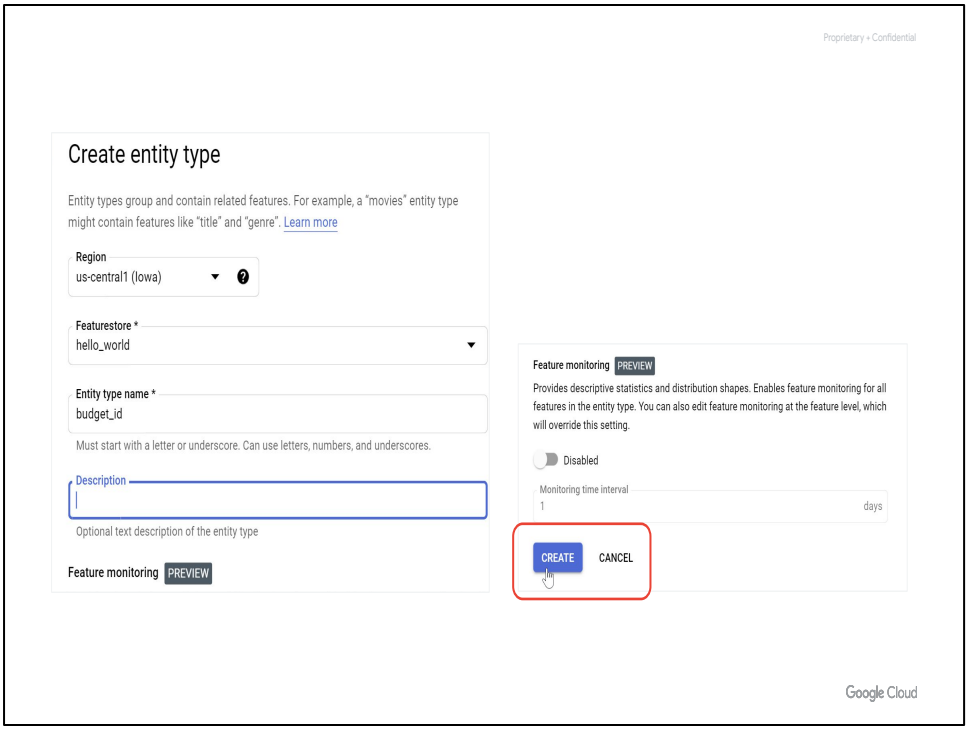
After you pre-process data, you're ready to begin. You can create a featurestore in the Vertex AI Console or the Vertex AI Workbench using the API.



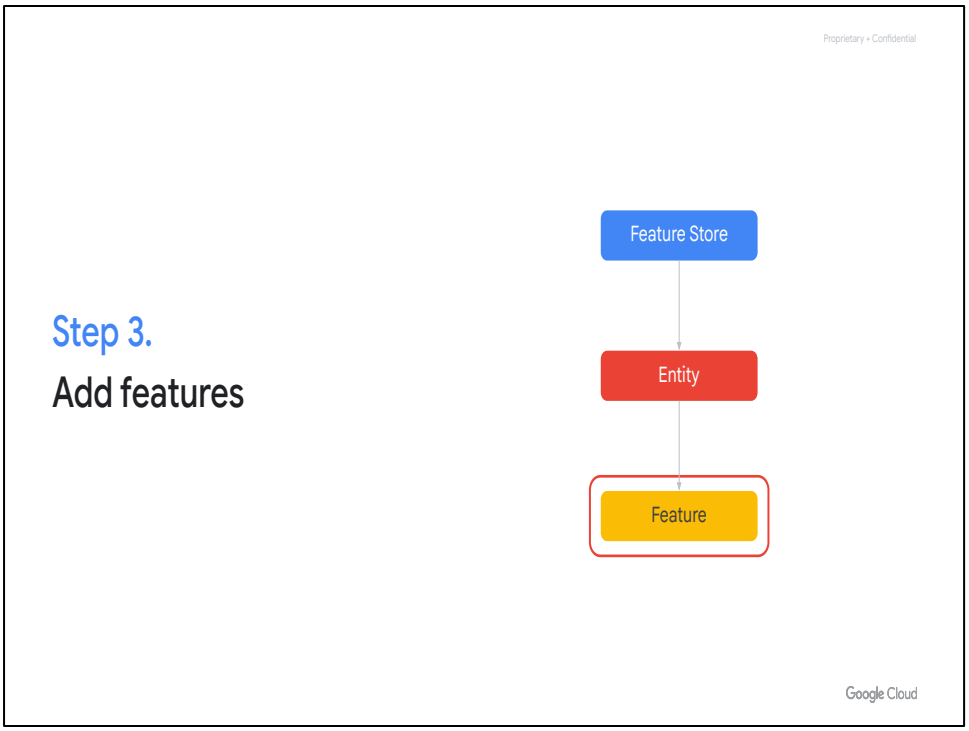
- Demo the FeatureStore Creation Process on your LAB



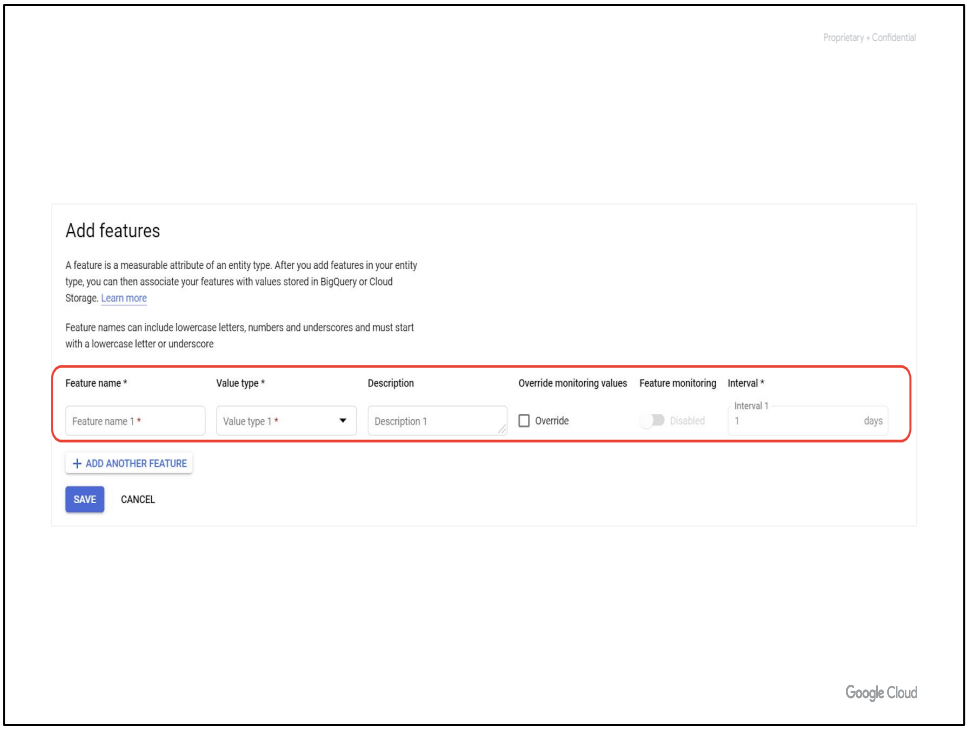
Step 2 is to create an entity type.



Click **Create** to create the entity type.

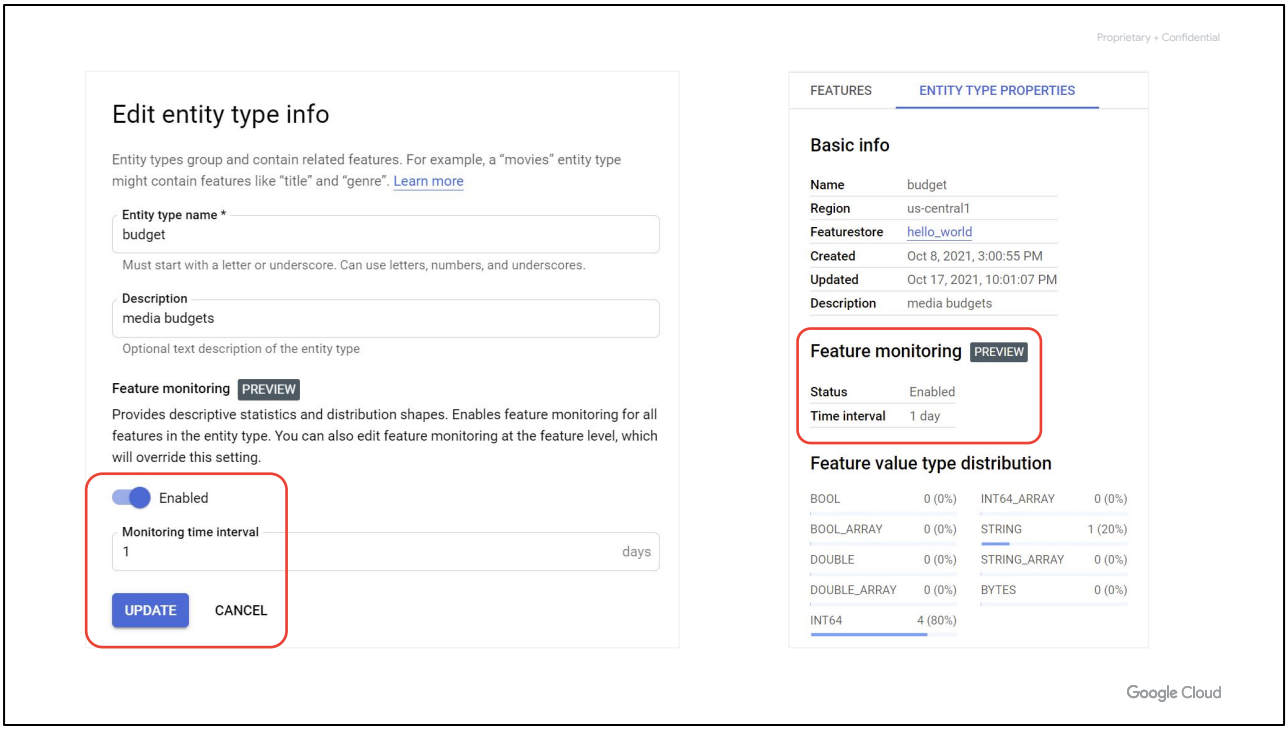


Step three is to add features.



The **Add Feature** window displays input fields for feature name, value type, description, override monitoring values, feature monitoring, and interval. Of the six input fields, only three are required: feature name, value type, and interval.

Before adding features, let's look at XYZ team's dataset.

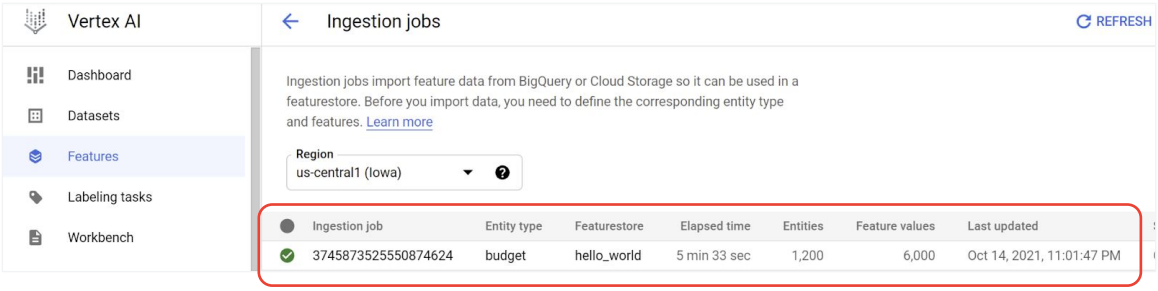


Click **Enabled** and then **Update**. After updating, check the entity properties again and note that Feature monitoring is enabled with a time interval of 1 day.

Note that monitoring can be enabled at any time.

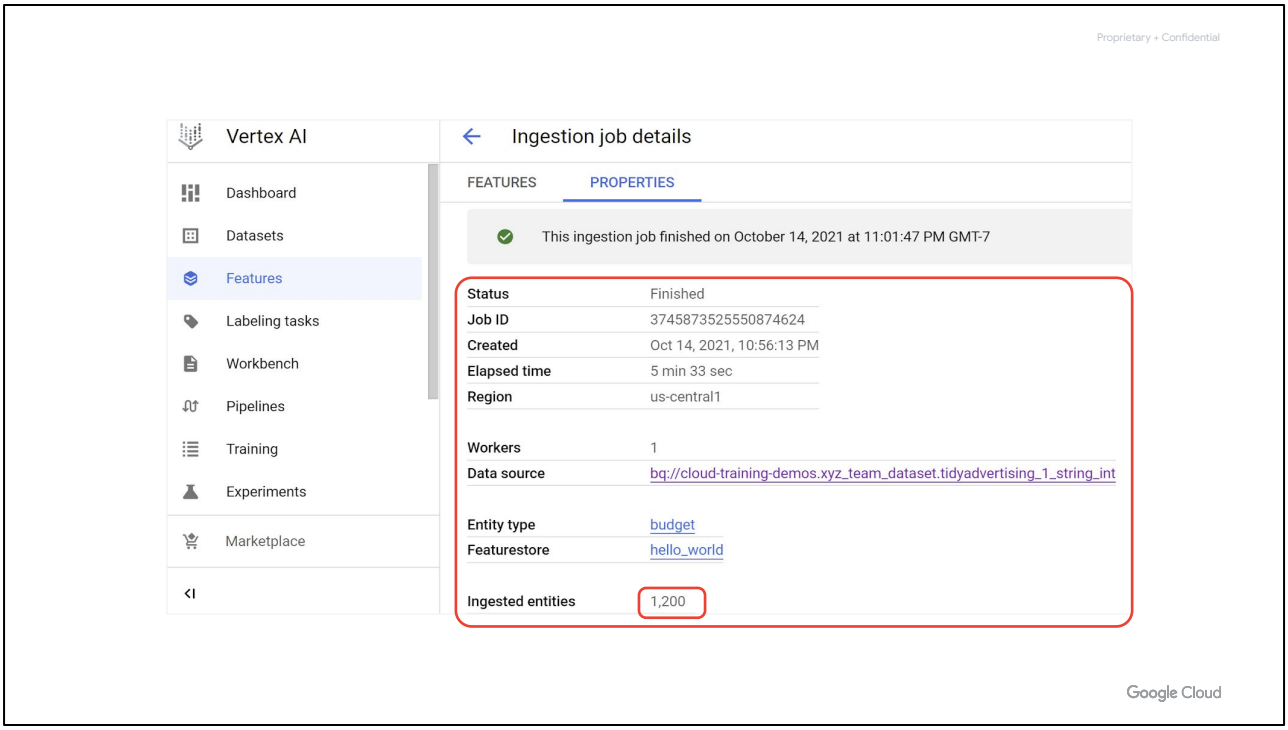
Feature owners, such as data scientists, might monitor feature values to detect data drift over time. In Feature Store, you can monitor and set alerts on featurestores and features.

Ingestion jobs



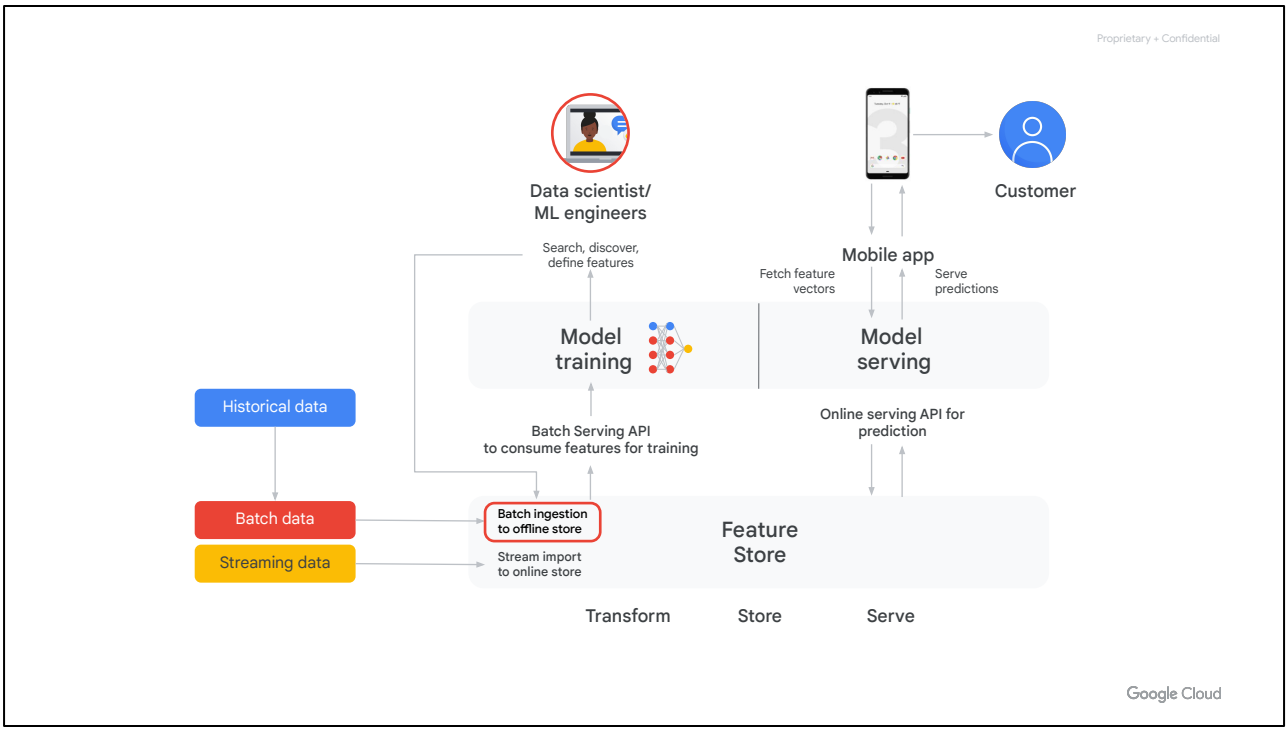
The team also set up an ingestion job. Ingestion jobs import feature data from BigQuery or Cloud Storage so it can be used in a featurestore. Before you import data, you need to define the corresponding entity type and features.

Feature Store offers batch ingestion so that you can do a bulk ingestion of values into a featurestore. For example, your computed source data might live in locations such as BigQuery or Cloud Storage. You can then ingest data from those sources into a featurestore so that feature values can be served in a uniform format from the central featurestore.



Selecting the ingestion job takes you to the ingestion properties, which identify when the job was created, how long it took to process, the region, the number of workers, and a link to the data source. The properties also identify the entity type and the name of the featurestore.

Note the number of ingested entities of 1,200, which barely meets the minimum number of 1,000 rows required for a dataset to be uploaded into Vertex AI.

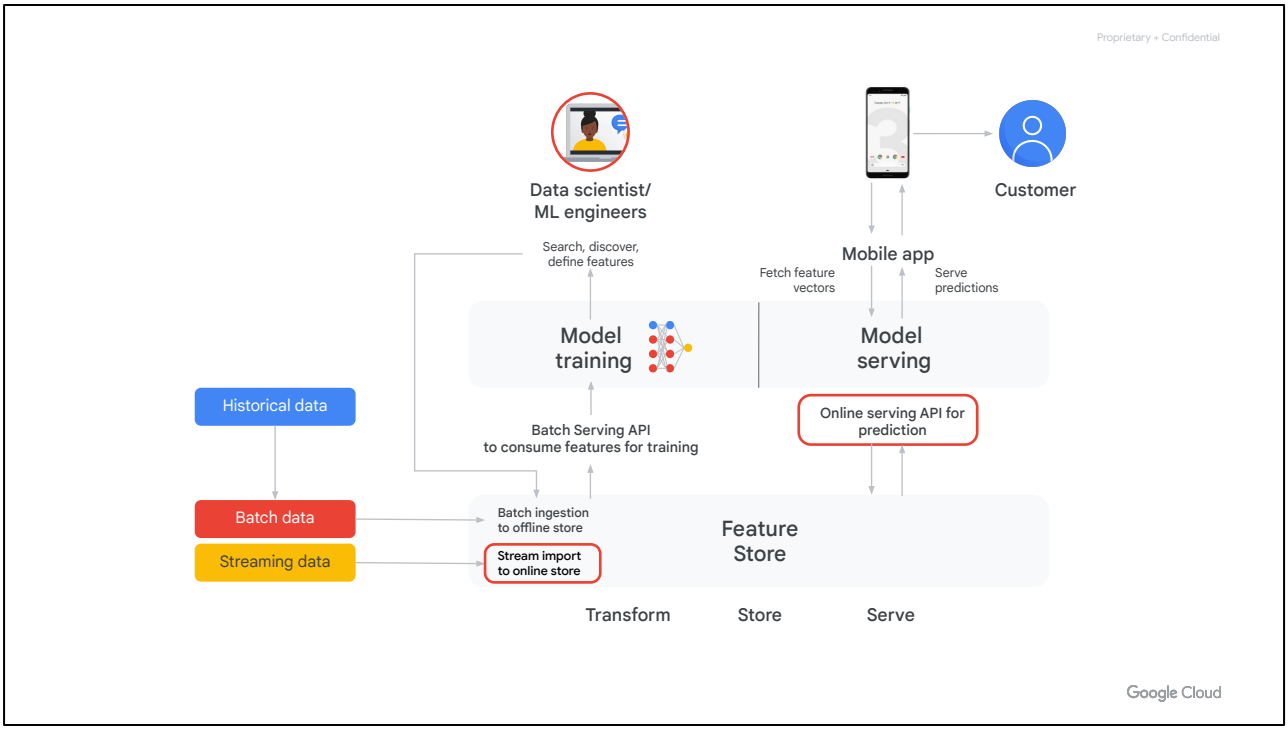


Batch ingestion lets you ingest feature values in bulk from a valid data source. For each request, you can import values for up to 100 features for a single entity type. Note that you can run only one batch ingestion job per entity type to avoid any collisions.

In a batch ingestion request, specify the location of your source data and how it maps to features in your featurestore. Because each batch ingestion request is for a single entity type, your source data must also be for a single entity type.

After the import has successfully been completed, feature values are available to subsequent read operations such as for model training.

In our baby weight example, our historical features are ingested in batch and made available (or served) to a mobile app.



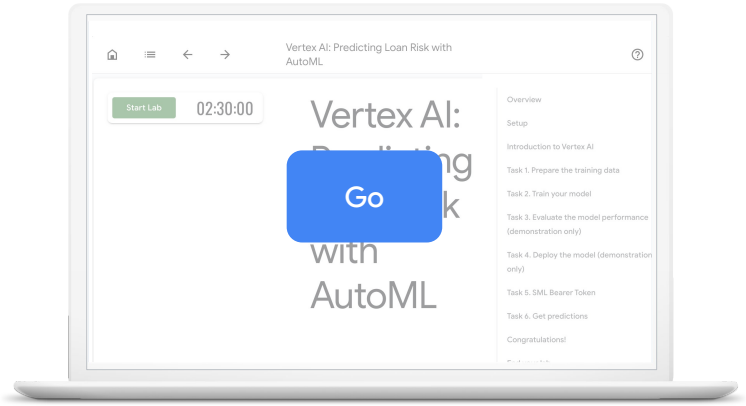
Another way to add feature values is by streaming near real-time data with the Online Serving API and calling the API with a list of required features.

Recommended Lab

Vertex AI: Predicting Loan Risk with AutoML

From the Course
Introduction to AI and Machine Learning on Google Cloud

Select this lab from the section:
“AI Development Workflow”



Vertex AI: Predicting Loan Risk with AutoML

From:
Introduction to AI and Machine Learning on Google Cloud Course

https://partner.cloudskillsboost.google/course_sessions/4835604/labs/387539

Questions and answers

Thank you for attending this training!

We love your feedback! Please take a minute to complete the survey and help us improve our courses.


Google Cloud