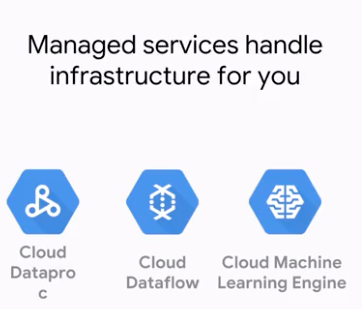
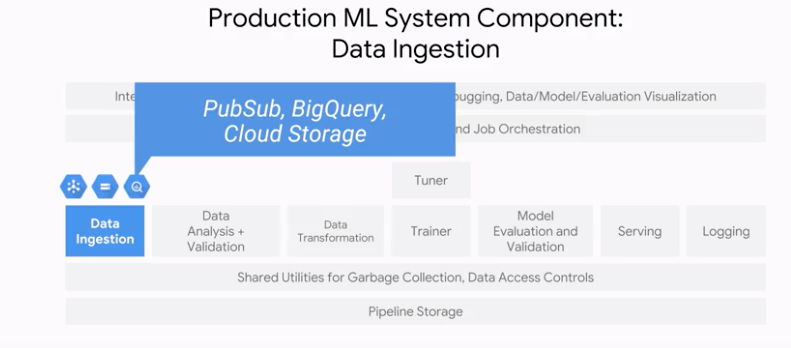
# Components of ML



Spark, Apache Beam, Tensor Flow



## Data Validation

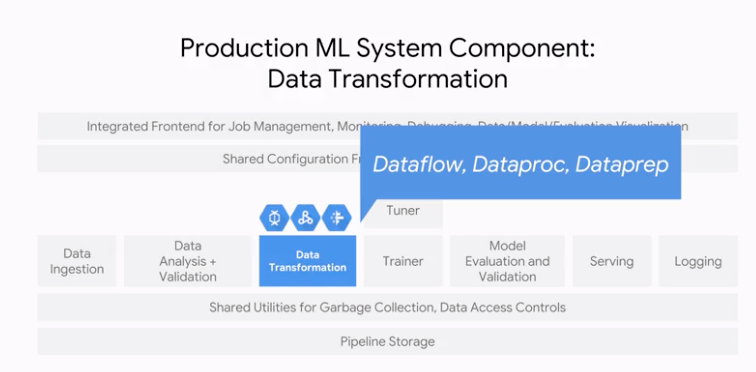
## 

Distribution of the data can be looked at for production systems

## 

## Data Transformation

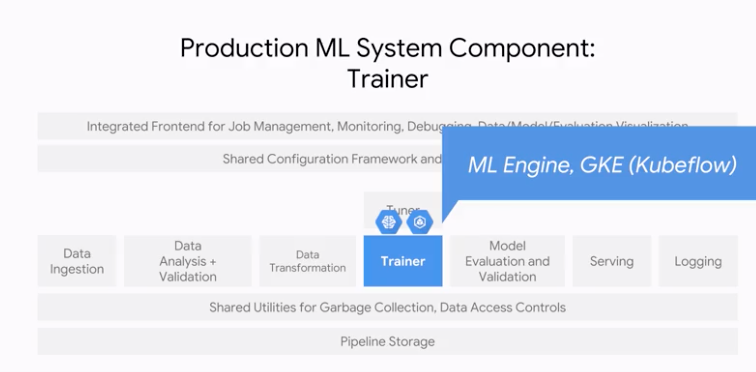
The data transformation component allows for feature wrangling. It can do things like generate feature to integer mappings. Critically, whatever mappings that are generated must be saved and reused at serving time. Failure to do this consistently results in a problem we'll be talking more about in a later module, called Training Serving Skill

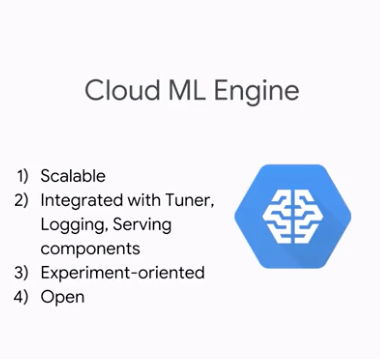


Trainer:

The trainer is responsible for training your model. It should be able to support data parallelism and model parallelism, and scale to large numbers of workers.

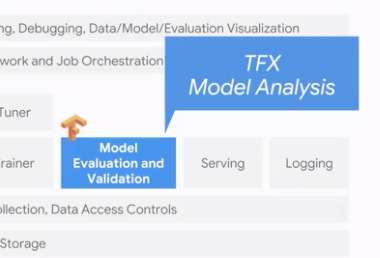
Hyperparameter tunning:





Finally, the trainer should also support hyperparameter tuning. There are two products that aligned with this component in GCP, ML Engine which provides the managed service for TensorFlow and GKE which provides a managed environment for hybrid ML models in Kubeflow.

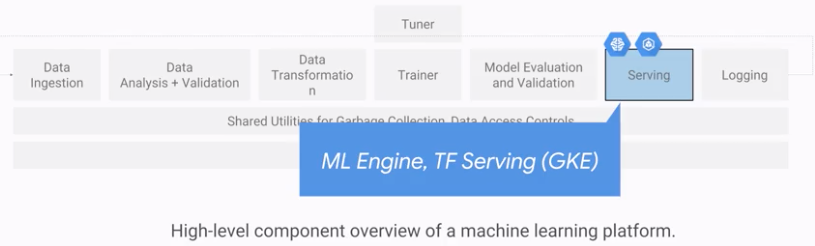
## Model Validation

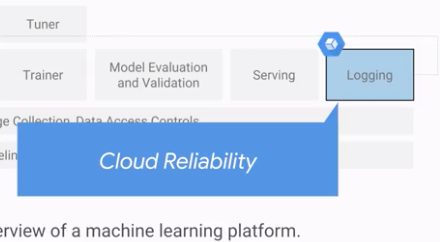


TF.tranform, tf. Model analysis and tf serving

## Serving

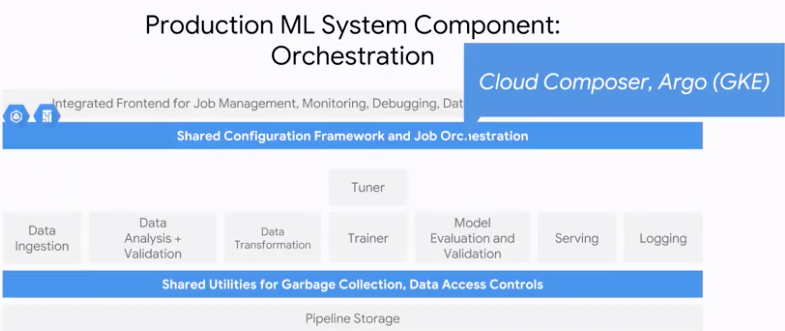
You can either use a fully managed TensorFlow serving service which is ML Engine or you can run TF serving on Kubernetes engine.





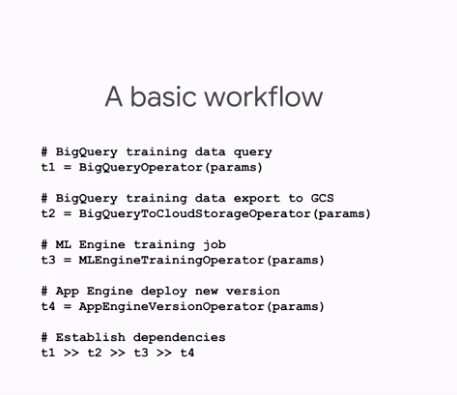
With cloud reliability, you get easy integration with all the other GCP products, the ability to craft alerting policies, and the ability to detect when new errors occur.

## Orchestration + Workflow



In GCP, orchestration can be done with Cloud composer, which is managed Apache airflow. There are airflow operators for all the GCP components that we've considered so far, including Cloud Storage, BitQuery, Dataflow, and ML Engine. So, you can orchestrate all these tasks from composer. Another option for orchestration is to use Argo on Google Kubernetes engine. Argo is a container management tool. If each of your tasks, data ingest, data transformation, or model training or running containers, then Argo is a good way to orchestrate the ML pipeline consisting of such containers.

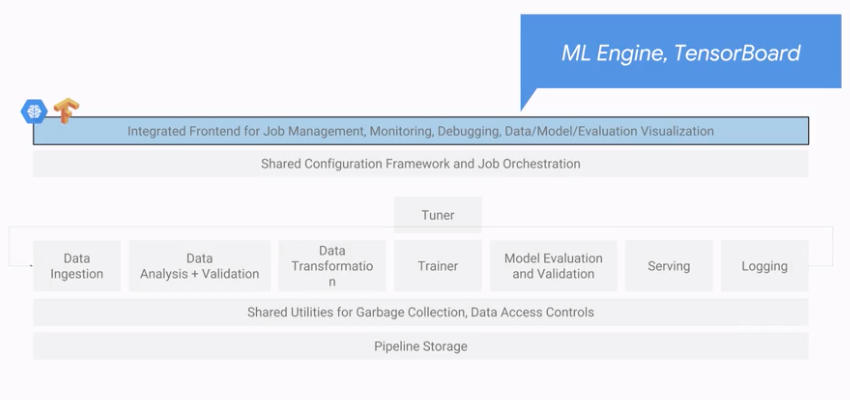


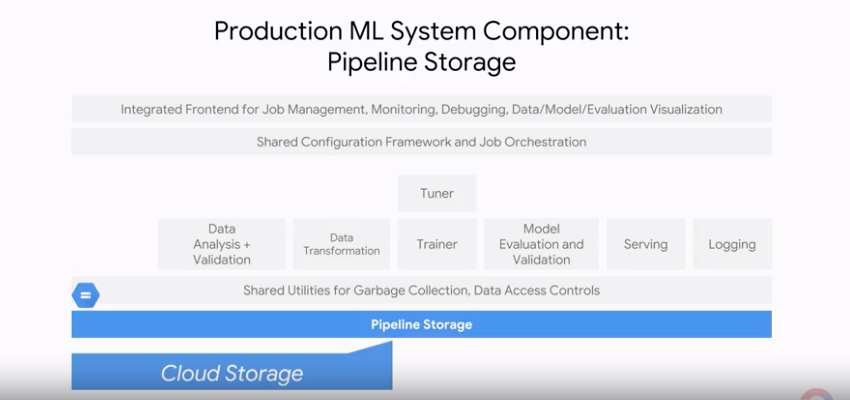


## Integrated Front End + Storage

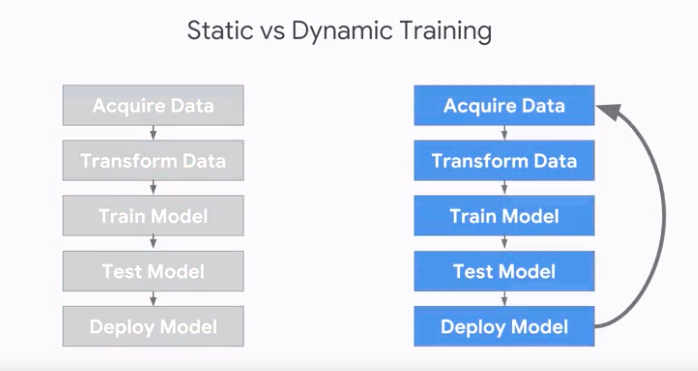
GCP – Cloud ML Engine , Tensor Board

In GCP, you can use TensorBoard and Cloud ML Engine. TensorBoard is the visualization software that comes bundled with TensorFlow.



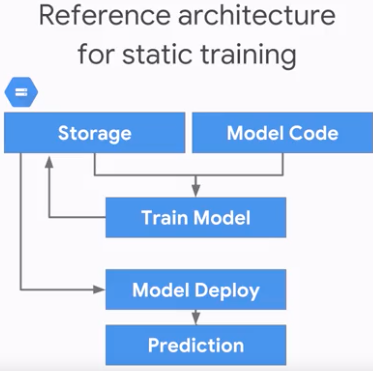


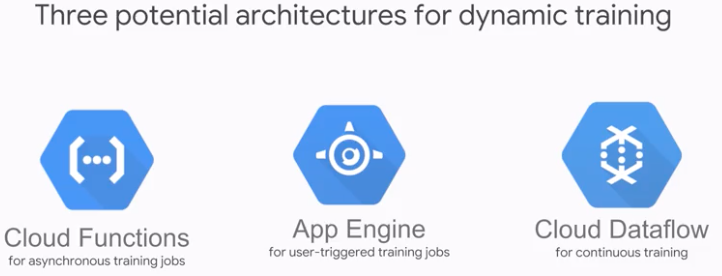
Training Design Decisions:

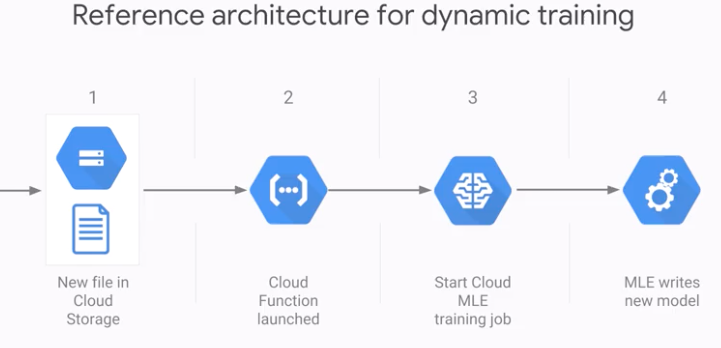


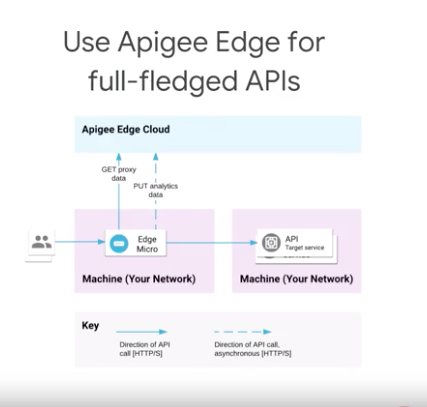


Part of the reason the dynamic is harder to build and test is our that new data may have all sorts of bugs in it. And that's something we'll talk about more deeply in a later course on designing adoptable ML systems. Can you think of other reasons why the engineering might be harder? The reason is that we need more monitoring, model rollback, and data quarantine capabilities. Let's test our understanding with some new use cases.

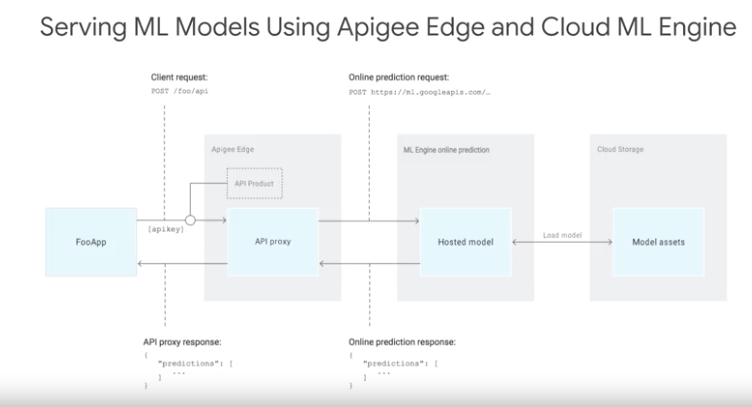




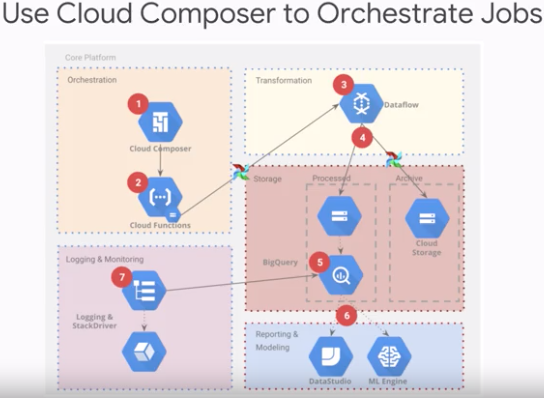




But when you actually develop your models, you'll want to have a full fledged API with security and access control policies, third-party access, monetization, and rate-limiting, and quota enforcement. One easy way to do this is with Apigee Edge. Apigee Edge creates a layer of abstraction in between Cloud ML Engine and the applications that want predictions.

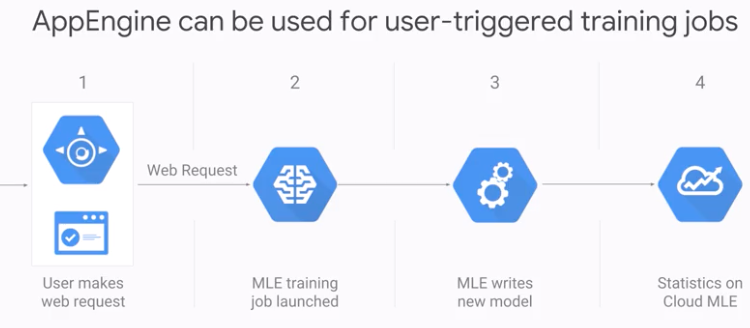


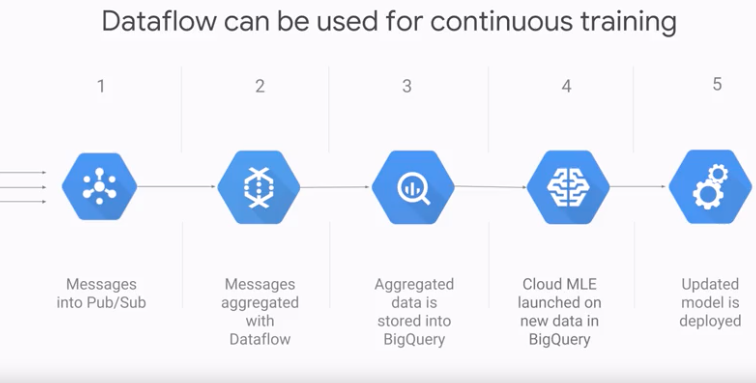
Next using cloud composer:



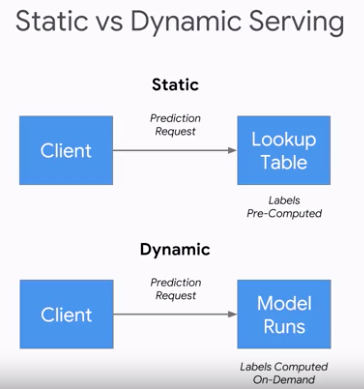
Here we have an orchestration layer with Cloud Composer which has cloud functions that trigger cloud data flow processing jobs on new data based on new data files being dropped into Google Cloud Storage.

Then a Cloud ML Engine training job is launched, and the Cloud ML Engine job writes a new model to Cloud storage. And then finally, the statistics of the training job are displayed to the user when the job is complete.





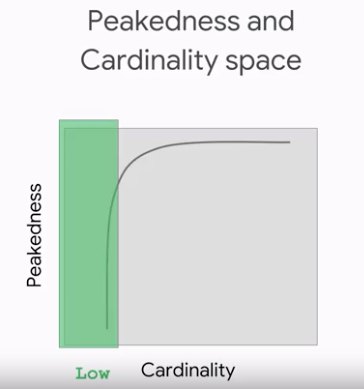
Serving Design Decisions:



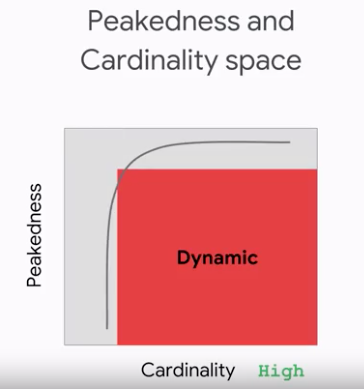
Distribution of the prediction workload is concentrated. (Inverse entropy)

Cardinality refers to the number of values in a set. In this case, the set is the set of all possible things we might have to make predictions for.

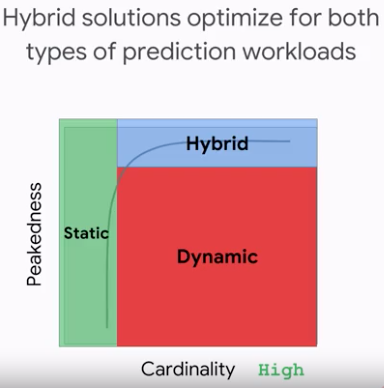
Peakedness refers to the extent to which the distribution of the prediction workload is concentrated. You can also think of it as inverse entropy. For example, a model that predicts the next word given the current word, which you might find in your mobile phone keyboard app, would be highly peaked.



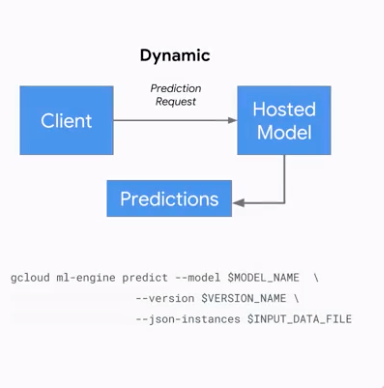
Low -> Static table

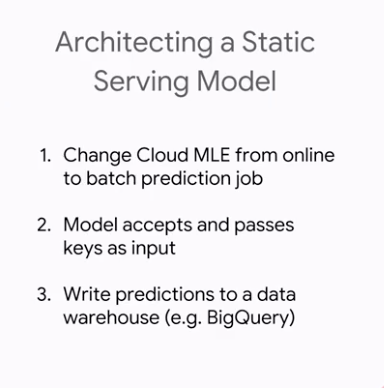


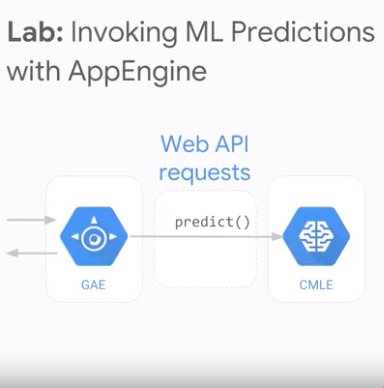
Dynamic

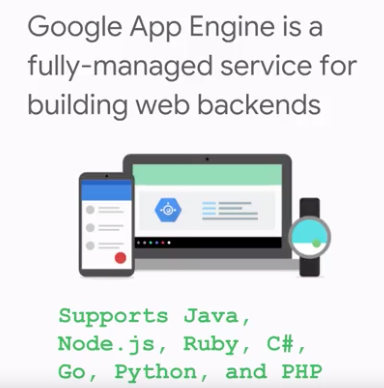


Frequently predictions cached and the tail computed on demand.

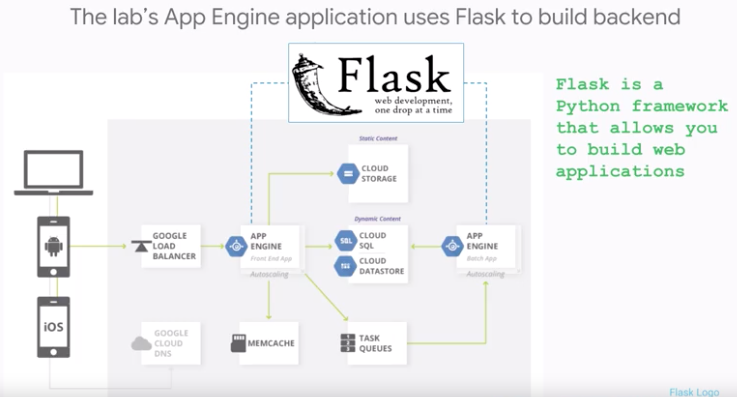








Our web application uses Flask, which is a lightweight framework for developing and deploying web apps and Python. In this case, our Flask app is a web form that takes the features of our model, and will show the predicted weight. And this is the interface that you'll build.



**Start Cloud Shell**

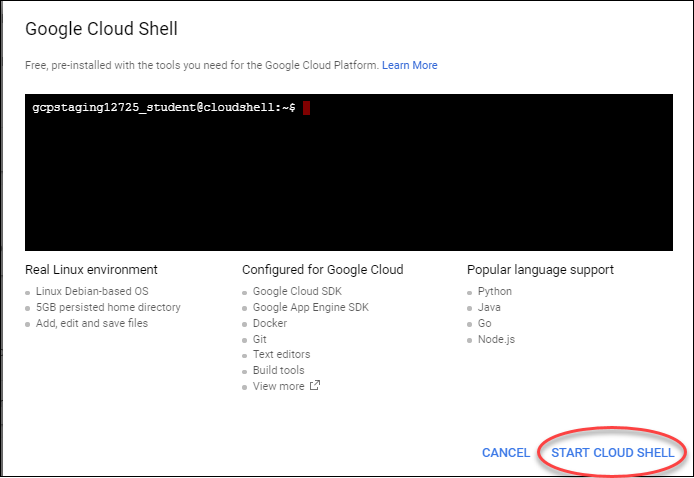
Activate Google Cloud Shell

Google Cloud Shell provides command-line access to your GCP resources.

From the GCP Console click the **Cloud Shell** icon on the top right toolbar:

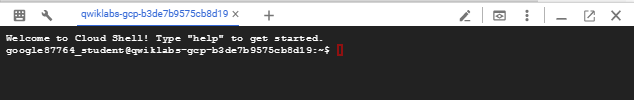


Then click **START CLOUD SHELL**:



You can click **START CLOUD SHELL** immediately when the dialog comes up instead of waiting in the dialog until the Cloud Shell provisions.

It takes a few moments to provision and connects to the environment:



The Cloud Shell is a virtual machine loaded with all the development tools you’ll need. It offers a persistent 5GB home directory, and runs on the Google Cloud, greatly enhancing network performance and authentication.

Once connected to the cloud shell, you'll see that you are already authenticated and the project is set to your *PROJECT\_ID*:

gcloud auth list

Output:

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)

**Note:** gcloud is the powerful and unified command-line tool for Google Cloud Platform. Full documentation is available on [Google Cloud gcloud Overview](https://cloud.google.com/sdk/gcloud). It comes pre-installed on Cloud Shell and supports tab-completion.

gcloud config list project

Output:

[core]

project = <PROJECT\_ID>

**Copy trained model**

**Step 1**

Set necessary variables and create a bucket:

REGION=us-central1

BUCKET=$(gcloud config get-value project)

TFVERSION=1.7

gsutil mb -l ${REGION} gs://${BUCKET}

**Step 2**

Copy trained model into your bucket:

gsutil -m cp -R gs://cloud-training-demos/babyweight/trained\_model gs://${BUCKET}/babyweight

**Deploy trained model**

**Step 1**

Set necessary variables:

MODEL\_NAME=babyweight

MODEL\_VERSION=ml\_on\_gcp

MODEL\_LOCATION=$(gsutil ls gs://${BUCKET}/babyweight/export/exporter/ | tail -1)

**Step 2**

Deploy trained model:

gcloud ml-engine models create ${MODEL\_NAME} --regions $REGION

gcloud ml-engine versions create ${MODEL\_VERSION} --model ${MODEL\_NAME} --origin ${MODEL\_LOCATION} --runtime-version $TFVERSION

**Code for your frontend**

**Step 1**

Clone the course repository:

cd ~

git clone https://github.com/GoogleCloudPlatform/training-data-analyst

**Step 2**

You can use the Cloud Shell code editor to view and edit the contents of these files.

Click on the (b8ebde10ba2a31c8.png) icon on the top right of your Cloud Shell window to launch Code Editor.

Once launched, navigate to the ~/training-data-analyst/courses/machine\_learning/deepdive/06\_structured/labs/servingdirectory.

**Step 3**

Open the **application/main.py**and **application/templates/form.html** files and notice the *#TODO*s within the code. These need to be replaced with code. The next section tells you how.

**Modify main.py**

**Step 1**

Open the main.py file by clicking on it. Notice the lines with *# TODO* for setting credentials and the api to use.

Set the credentials to use Google Application Default Credentials (recommended way to authorize calls to our APIs when building apps deployed on AppEngine):

credentials = GoogleCredentials.get\_application\_default()

Specify the api name (ML Engine API) and version to use:

api = discovery.build('ml', 'v1', credentials=credentials)

**Step 2**

Scroll further down in main.py and look for the next *#TODO* in the method get\_prediction(). In there, specify, using the **parent** variable, the name of your trained model deployed on Cloud MLE:

parent = 'projects/%s/models/%s' % (project, model\_name)

**Step 3**

Now that you have all the pieces for making the call to your model, build the call request by specifying it in the **prediction** variable:

prediction = api.projects().predict(body=input\_data, name=parent).execute()

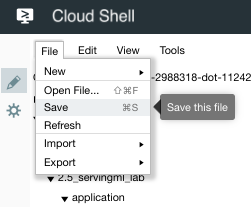
**Step 4**

The final *#TODO* (scroll towards bottom) is to get gestation\_weeks from the form data and cast into a float within the **features** array:

features['gestation\_weeks'] = float(data['gestation\_weeks'])

**Step 5**

Save the changes you made using the **File** > **Save** button on the top left of your code editor window.



**Modify form.html**

form.html is the front-end of your app. The user fills in data (features) about the mother based on which we will make the predictions using our trained model.

**Step 1**

In code editor, navigate to the application/templates directory and click to open the form.html file.

**Step 2**

There is one *#TODO* item here. Look for the div segment for **Plurality** and add options for other plurality values (2, 3, etc).

<md-option value="2">Twins</md-option>

<md-option value="3">Triplets</md-option>

**Step 3**

Save the changes you made using the **File** > **Save** button on the top left of your code editor window.

**Deploy and test your app**

**Step 1**

In Cloud Shell, run the deploy.sh script to install required dependencies and deploy your app engine app to the cloud.

cd training-data-analyst/courses/machine\_learning/deepdive/06\_structured/labs/serving

./deploy.sh

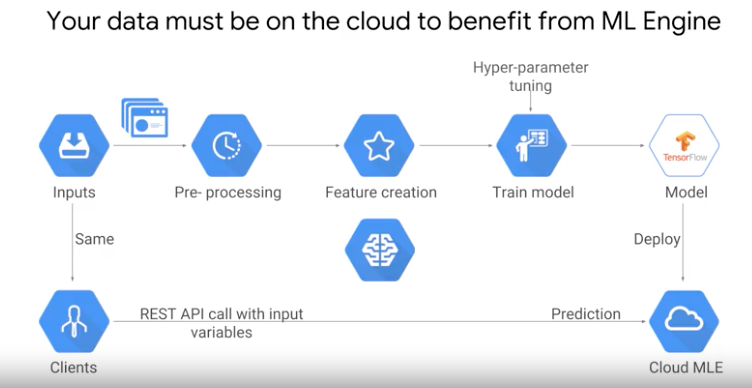
Note: Choose a region for App Engine when prompted and follow the prompts during this process

**Step 2**

Go to the url https://<PROJECT-ID>.appspot.com and start making predictions.

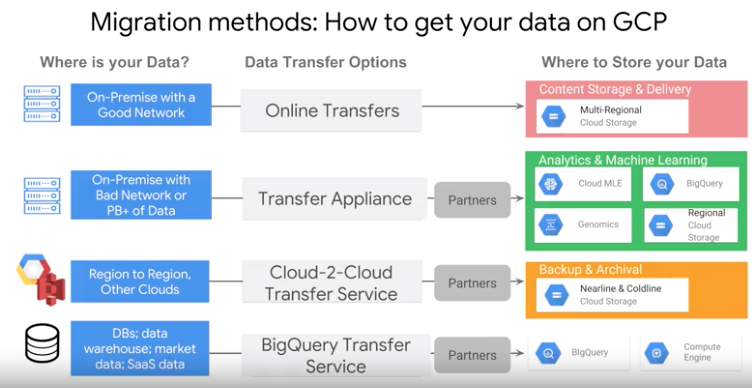
*Note: Replace <PROJECT-ID> with your Project ID.*

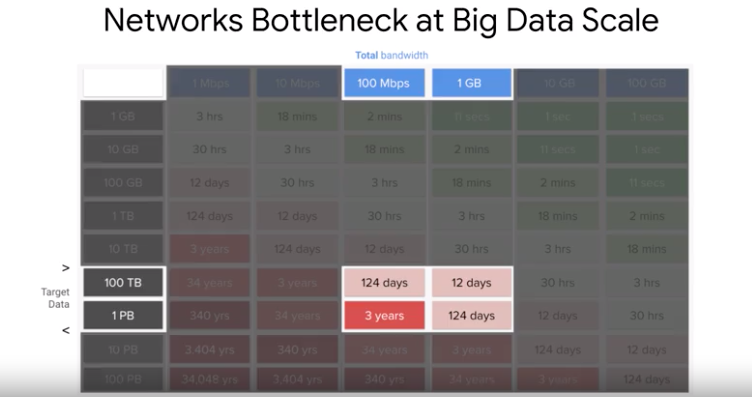
Architecture Production ML Systems

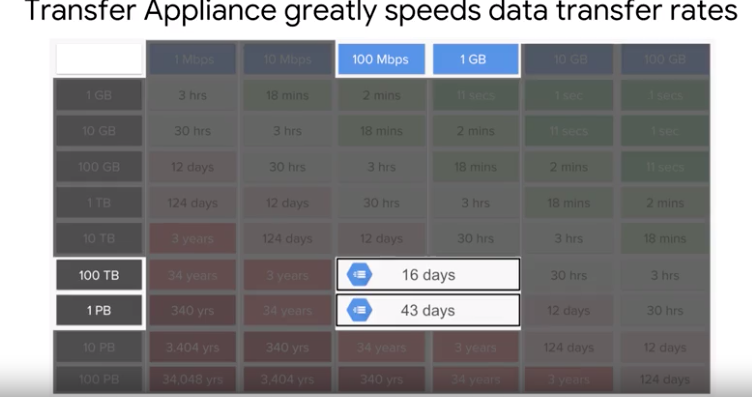


Most important thing : The data must be in cloud to take advantage of the fully managed services.

Gsutil to get the data into GCS.







Cloud Transfer: It would be from source to sink.

