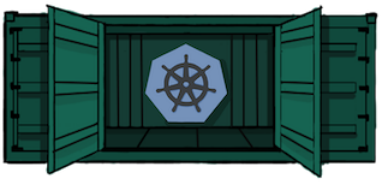


Dmitri Lerko

Personal blog where I cover my experiences and best practices with GCP, GKE, GitOps, Certifications and DevOps.

Google Cloud Professional Machine Learning Engineer Certification Preparation Guide

Aug 16, 2020
31 min read
certifications gcp ml beta

Last updated on 11th of October, 2020. Certification is expected to go into GA on 15th of October.

Authors

Dmitri - knew GCP, didn’t know any of the ML stuff

Steven - knew ML, didn’t know any of the GCP stuff

Perfect reason to collaborate!

New certification

Google announced a new Machine Learninng Engineer beta certification in July with certifications taking place from 15th of July to 21st of August. Dmitri has attempted it on 16th of August. Most of the material below was written before the attempt and are a true reflection of a study guide, not an exam dump. Dmitri is a strong advocate of fair certifications therefore this guide is to motivate a wider audience to learn more about ML and GCP. Steven

Prior knowledge

Dmitri works with GCP professionally for over 2 years. I have GCP 7 certificates, including Data Engineer and DevOps Engineer ones that I find most helpful for this certification. I knew nothing about ML and Google's ML production prior to starting my ~3-week preparation in the evenings and weekends. Steven is a recent graduate and has taken up a number of Machine Learning modules, so he has a strong theoretical knowledge, but lacked practical application with GCP.

Approach taken with this guide

We used Google's own certification guide as a bedrock for our studies. It gave us a broad range of topic to try and find knowledge for. Additionally, we are listing materials that we used to prepare and lastly some final tips that don't fall into either of these categories. Domain of ML and GCP is absolutely huge, so unless you are expert in ML and only need to brush up on GCP, give this certification plenty of time to learn. Even if you'll learn everything listed below it won't guarantee a Pass. Apply your newly aquired knowledge in practice and learn from your mistakes!

Learning resources used

Certification Guides

- My take on Professional Machine Learning Engineer(Beta) Exam by Parth Mehta
- Google Cloud Professional Machine Learning Engineer Certification: Post Exam Impressions by Carlos Timoteo

Courses

- Free. Most videos can be watched at x2 speed.
  - Machine Learning Crash Course
  - Problem Framing
  - Data Prep
  - Testing and Debugging
  - Rules of ML

- Paid. However, 7-day trial and fast learning can work in your favour. Pluralsight has all the same content too, so if you've got sa ubscription to it, then you do not need to use Coursera.
    - Google Cloud Platform Big Data and Machine Learning Fundamentals
    - Building Batch Data Pipelines on GCP
    - Building Resilient Streaming Analytics Systems on GCP
    - Smart Analytics, Machine Learning, and AI on GCP
    - How Google does Machine Learning
    - Launching into Machine Learning
    - Intro to TensorFlow
    - Feature Engineering
    - Art and Science of Machine Learning
    - End-to-End Machine Learning with TensorFlow on GCP - not attempted prior to the exam.
    - Production Machine Learning Systems - not attempted prior to the exam.
    - Image Understanding with TensorFlow on GCP - not attempted prior to the exam.
    - Sequence Models for Time Series and Natural Language Processing - not attempted prior to the exam.
    - Recommendation Systems with TensorFlow on GCP - not attempted prior to the exam.
    - Building end to end machine learning workflows with Kubeflow - (Pluralsight only) *highly recommend*

Tutorials

- TFX on Cloud AI Platform Pipelines (Free)

Google Cloud Solutions (Free)

- Architecture for MLOps using TFX, Kubeflow Pipelines, and Cloud Build
- Best practices for performance and cost optimization for machine learning
- Building production-ready data pipelines using Dataflow: Overview
- Minimizing real-time prediction serving latency in machine learning

Relevant Open Source tools

- what-if-tool
- katib
- kubeflow

Machine Learning Glossary

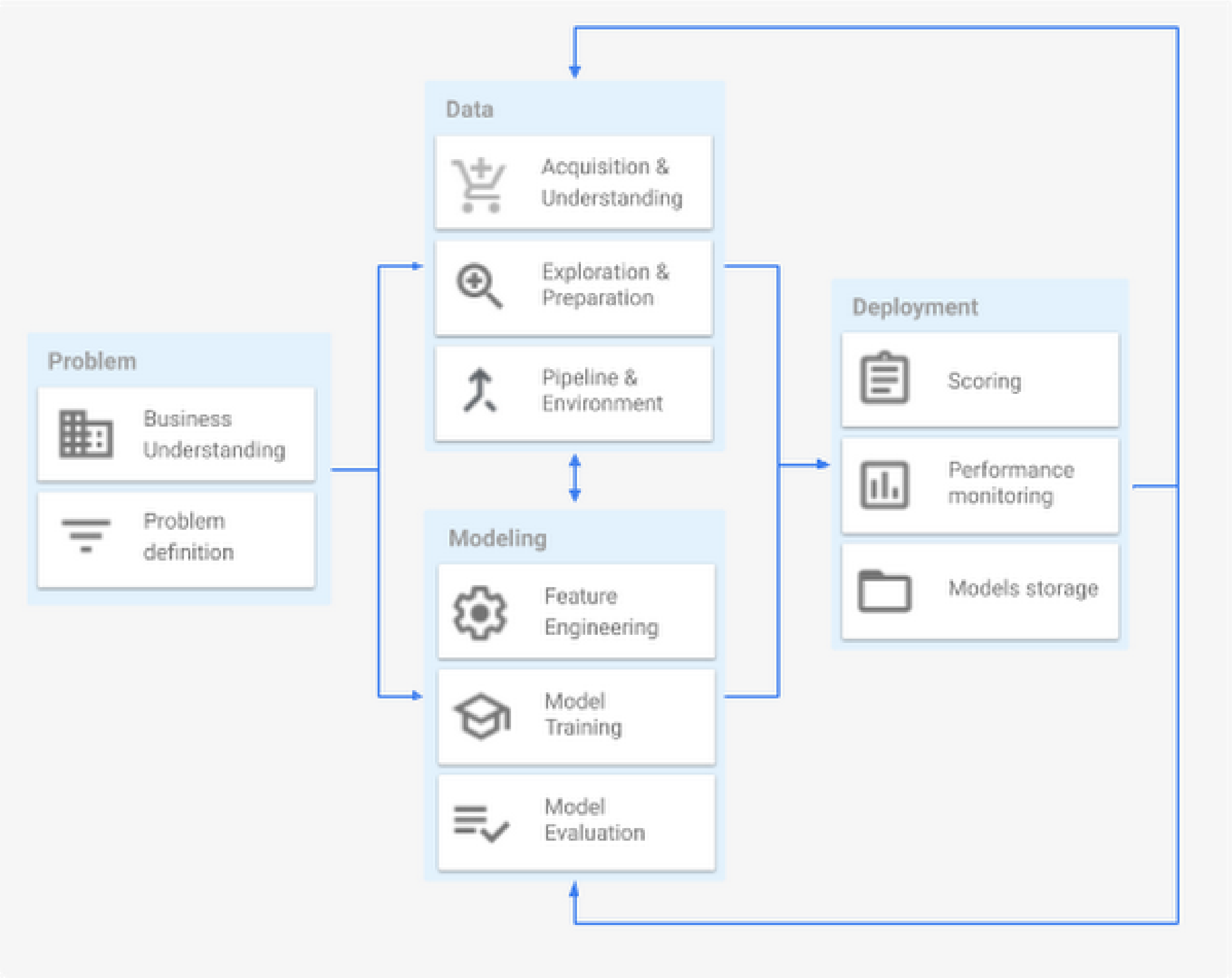
Things you should know really well even if they are not in the guide

- Differences between ANNs, CNNs, RNNs, LSTM's. Which is Appropriate?
- How to appropriately split your data into Train/Validate/Test examples.
- Neuron Activations: Many possibilities, which is appropriate?
- Output Activation: How many outputs? Is the output binary? Probabilistic?
- Gradient Descent: Batch, Mini-Batch, Stochastic. What are the differences? What are the consequences of these differences?
- Optimisers: Know the differences, and relative strengths of each.
- Evaluating model performance. Accuracy is rarely a meaningful metric and frequently misleading.
- How to recognise and deal with Over and Under fitting. eg. achieved Better/Worse performance on training than testing.
- How to recognise and deal with Class Imbalance. Can look like overfitting.
- Hyperparameters: Learning Rate, Dropouts etc. Are they appropriate?
- Regularisation: L1, L2 what are the differences? Is regularisation appropriate?
- How to recognise and deal with Vanishing/Exploding Gradients.
- Collaborative filtering
- Savings costs on developing, training and serving ML models in GCP
- `tf.data`
- `TFRecord`
- AI Platform, Dataflow - learn and try out all you can
- Optimising for training speed, serving speed
- Dealing with TensorFlow and machine Out of Memory issues
- Re-using existing models in Google products
- Where and why use more CPU, memory, GPU or TPU.
- Understand customer lifetime value
- Study as many solutions as you can, they are well written, full of useful content and great bridge between theory and practice.

Google Cloud Professional Machine Learning Engineer Certification Preparation Guide

Section 1: ML Problem Framing

A machine learning project lifecycle mainly comprises four major stages, executed iteratively:



In traditional software engineering, you can reason from requirements to a workable design, but with machine learning, it will be necessary to experiment to find a workable model.

1.1 Translate business challenge into ML use case. Considerations include:

- Defining business problems
    - dentifying Good Problems for ML
    - Cast as ML problem - In basic terms, ML is the process of training a piece of software, called a model, to make useful predictions using a data set. This predictive model can then serve up predictions about previously unseen data. We use these predictions to take action in a product; for example, the system predicts that a user will like a certain video, so the system recommends that video to the user.
        - Does business problem satisfy above criteria?
        - What is being predicted? What is being classified?
        - What data do is needed?
    - ML problem in question of software
        - What is the API for the problem during prediction?
        - Who will use this service? How are they doing it today?

    - Data problem
        - What data are we analyzing?
        - What data are we predicting?
        - What data are we reaching to?

- Identifying nonML solutions

    - Don’t be afraid to launch a product without machine learning
        - Machine learning is cool, but it requires data. Theoretically, you can take data from a different problem and then tweak the model for a new product, but this will likely underperform basic heuristics. If you think that machine learning will give you a 100% boost, then a heuristic will get you 50% of the way there. For instance, if you are ranking apps in an app marketplace, you could use the install rate or number of installs as heuristics. If you are detecting spam, filter out publishers that have sent spam before. Don’t be afraid to use human editing either. If you need to rank contacts, rank the most recently used highest (or even rank alphabetically). If machine learning is not absolutely required for your product, don’t use it until you have data.
    - Path to ML
        - Individual contributor - prototype and try ideas manually. Fail fast. Experiment.
            - Dangers of skipping this step as it is a prerequisite to the ability to scale and validates the correctness of assumptions.
            - Dangers of lingering here too long
                - One person gets skilled up and leaves
                - Failed to scale process in time to meet demand
        - Delegation - gentry ramp up to include more people
            - Dangers of skipping this step
                - Not forced to formalise the process
                - Inherent diversity in human responses become a testbed - a great product learning opportunity
                - Great ML systems will need humans in the loop
            - Dangers of lingering here too long
                - Paying a high marginal cost to serve each user
                - More voices will say automation is not possible
                - Organizational lock-in
        - Digitization - automate mundane parts of the process
            - Dangers of skipping this step
                - You will always need infrastructure
                - IT project and ML success tied and the whole project will fail if either does need humans in the loop.

      - Dangers of lingering here too long
        - Competitors are collecting data and tuning their offers from these new insights
    - Big Data and Analytics - measure and achieve data-driven success
      - Dangers of skipping this step
        - Unclean data means no ML training
        - You can’t measure success
      - Dangers of lingering here too long
        - Limit complexity of problems you can solve
        - Google search has been here for years

- Defining output use
  - Properties of Good Output
  - Using the Output
- Managing incorrect results
- Identifying data sources

1.2 Define ML problem. Considerations include:

- Defining problem type (classification, regression, clustering, etc.)
  - **Classification** - Pick one of N labels - Cat, dog, horse, or bear
  - **Regression** - Predict numerical values - Click-through rate
  - **Clustering** - Group similar examples - Most relevant documents (unsupervised)
  - **Association Rule Learning** - Infer likely association patterns in data - If you buy hamburger buns, you’re likely to buy hamburgers (unsupervised)
  - **Structured outputs** - Create complex output - Natural language parse trees, image recognition bounding boxes
  - **Ranking** - Identify position on a scale or status - Search result ranking
- Defining outcome of model predictions
  - Problem Framing
  - Don’t overthink which objective you choose to directly optimize
  - Choose a simple, observable and attributable metric for your first objective.
  - Good idea to set accuracy benchmark before ever creating the model, then start with the simplest solution as a baseline.
- Defining the input (features) and predicted output format

- Which features are actually important? Which features seek to only add noise? Are there any Linear dependencies between features? What is the fewest number of features required for good performance? What is the maximum number of features we are willing to use?
- What is the target audience/platform for the output? What format do they expect?
- Is the output data streamed? Published at set intervals? Published adhoc?

1.3 Define business success criteria. Considerations include:

- Success metrics
  - Success and Failure Metrics
  - First, design and implement metrics Before formalizing what your machine learning system will do, track as much as possible in your current system. By being more liberal about gathering metrics, you can gain a broader picture of your system. Notice a problem? Add a metric to track it! Excited about some quantitative change on the last release? Add a metric to track it! Do this for the following reasons:
    - It is easier to gain permission from the system's users earlier on.
    - If you think that something might be a concern in the future, it is better to get historical data now.
    - If you design your system with metric instrumentation in mind, things will go better for you in the future. Specifically, you don't want to find yourself grepping for strings in logs to instrument your metrics!
    - You will notice what things change and what stays the same. For instance, suppose you want to directly optimize one-day active users. However, during your early manipulations of the system, you may notice that dramatic alterations of the user experience don't noticeably change this metric.
  - How do we define success? How do we define when the model is performing well? Accuracy is Misleading!!, F1 Precision/Recall?, ROC? Training Loss?, RMSE?
  - Choose your evaluation metrics in light of acceptable tradeoffs between **False Positives** and **False Negatives**
- Determination of when a model is deemed unsuccessful
  - Tolerances should be set in ADVANCE of model training
  - The general aim for a model to perform better than simple heuristics or match complex heuristics, but be more manageable/maintainable

1.4 Identify risks to feasibility and implementation of ML solution. Considerations include:

- Assessing and communicating business impact
- Assessing ML solution readiness
- Assessing data readiness
    - FACETS gives users a quick understanding of the distribution of values across features of their datasets.
        - Unexpected feature values
        - Features with high percentages of missing values
        - Features with unbalanced distributions
        - Distribution skew between datasets
- Aligning with Google AI principles and practices (e.g. different biases)
    - **Reporting bias** - occurs when the frequency of events, properties, and/or outcomes captured in a data set does not accurately reflect their real-world frequency. This bias can arise because people tend to focus on documenting circumstances that are unusual or especially memorable, assuming that the ordinary can “go without saying.”
    - **Automation bias** - is a tendency to favour results generated by automated systems over those generated by non-automated systems, irrespective of the error rates of each.
    - **Selection bias** - occurs if a data set’s examples are chosen in a way that is not reflective of their real-world distribution. Selection bias can take many different forms:
        - **Coverage bias**: Data is not selected in a representative fashion.
        - **Non-response bias (participation bias)**: Data ends up being unrepresentative due to participation gaps in the data-collection process.
        - **Sampling bias**: Proper randomization is not used during data collection.
    - **Group attribution bias** is a tendency to generalize what is true of individuals to an entire group to which they belong. Two key manifestations of this bias are:
        - **In-group bias**: A preference for members of a group to which you also belong, or for characteristics that you also share.
        - **Out-group homogeneity bias**: A tendency to stereotype individual members of a group to which you do not belong, or to see their characteristics as more uniform.
    - **Implicit bias** occurs when assumptions are made based on one’s own mental models and personal experiences that do not necessarily apply more generally.

Section 2: ML Solution Architecture

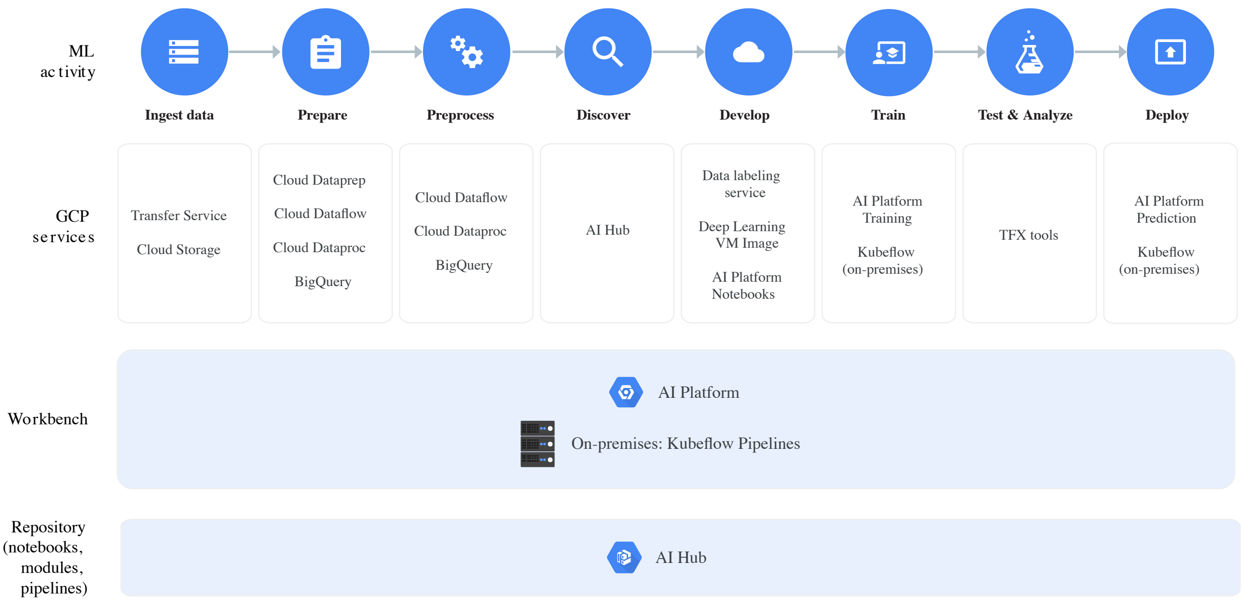
2.1 Design reliable, scalable, highly available ML solutions. Considerations include:

- Optimizing data use and storage

    - There are ways to optimise data for faster ingestion, cheaper storage. Think of ways to avoid ingestion pipeline bottlenecks.
- Data connections
    - Think of all the ways data can travel to a ML model
        - On prem sources
        - Proprietary formats
        - Events streaming from IoT
        - GCS
        - Cloud SQL
        - BigQuery
        - For all of the above, there are various ways to ingest the data, pre-process it and make it available for current or future training. This is a vast topic you should become familiar in.
- Automation of data preparation and model training/deployment
    - KubeFlow, TFX, Dataflow, PubSub, BigQuery and GCS are likely to be core components of this.
    - It is well worth knowing that GCS can send you events when you place new files into the bucket.
- SDLC best practices
    - Source control changes
    - Reproducible builds by automation
    - Reproducible deployments by automation
    - Version models

2.2 Choose appropriate Google Cloud software components. Considerations include:

- A variety of component types - data collection; data management *know what they do and why they exist*. Know these products really well!



ML activity	Ingest data	Prepare	Preprocess	Discover	Develop	Train	Test & Analyze	Deploy
GCP services	Transfer Service Cloud Storage	Cloud Dataprep Cloud Dataflow Cloud Dataproc BigQuery	Cloud Dataflow Cloud Dataproc BigQuery	AI Hub	Data labeling service Deep Learning VM Image AI Platform Notebooks	AI Platform Training Kubeflow (on-premises)	TFX tools	AI Platform Prediction Kubeflow (on-premises)

Workbench: AI Platform; On-premises: Kubeflow Pipelines

Repository (notebooks, modules, pipelines): AI Hub

- AI Hub - is an enterprise-grade hosted repository for discovering, sharing, and reusing artificial intelligence (AI) and ML assets. To store trained and validated models, plus their relevant metadata, you can use AI Hub as a model registry.
- AI Platform Training not only supports TensorFlow, Scikit-learn, and XGboost models, but also supports models implemented in any framework using a user-provided custom container. In addition, a scalable, Bayesian optimization-based service for a hyperparameter tuning is available.
- AI Platform Prediction - Trained models can be deployed to AI Platform Prediction as a microservice that has a REST API.
- AI Platform Notebooks
- AI Platform Pipelines
- Cloud SQL
- Cloud Bigtable
- Cloud Spanner
- Cloud Data Fusion
- BigQuery
- PubSub
- BigTable
- Dataproc
- Cloud Dataflow - is a fully managed, serverless, and reliable service for running Apache Beam pipelines at scale on Google Cloud. Dataflow is used to scale the following processes:
  - Computing the statistics to validate the incoming data.
  - Performing data preparation and transformation.
  - Evaluating the model on a large dataset.
  - Computing metrics on different aspects of the evaluation dataset.

- Cloud Storage - is a highly available and durable storage for binary large objects. Cloud Storage hosts artifacts produced throughout the execution of the ML pipeline, including the following:
    - Data anomalies (if any)
    - Transformed data and artifacts
    - Exported (trained) model
    - Model evaluation metrics
- Cloud Data Loss Prevention API
- Exploration/analysis
    - Some of the tools available for the task:
        - AI Notebooks
        - ML Frameworks
        - Dataprep
- **Feature engineering**
    - What makes a good feature?
        - Feature should have a strong correlation with the output
        - Want to limit features that are strongly correlated with each other! Try Transformative techniques
        - Be known at prediction-time. e.g. you can't have data in the model which is only made available once a month and use it for daily predictions.
        - Be numeric with a meaningful magnitude, Categorical values have been assigneed numeric values
        - Have enough samples - at least 5 samples. Avoid once-off samples, avoid hot samples (over-exposed)
        - Bring human insight to the problem
    - How to represent numeric features which do not have a meaningful magnitude? e.g. *employee id*
        - One-hot encoding. Similar to a bitmap array. Known as a sparse-column in TensorFlow.
- Logging/management *
- Automation
    - Cloud Build
    - TensorFlow Extended (TFX)
    - KubeFlow
    - AI Platform
- Monitoring
    - Cloud Monitoring (formerly Stackddriver)
    - Tensorboard
- Serving
    - BigQuery ML serving
    - AI Platform Prediction

    - Auto ML
    - Run model on a device via exportable TensorFlow models
    - TensorFlow exported model can be used `gcloud ml-engine predict --model <model_name> --json-instances data.json`. Can predict locally with ``gclud ml-engine local predict –model-dir <dir_path> –json-instances test_json.txt`

2.3 Choose appropriate Google Cloud hardware components. Considerations include:

- Selection of quotas and compute/accelerators with components

2.4 Design architecture that complies with regulatory and security concerns.

Considerations include:

- Building secure ML systems
- Privacy implications of data usage
    - Considerations for Sensitive Data within Machine Learning Datasets
- Identifying potential regulatory issues

Section 3: Data Preparation and Processing

3.1 Data ingestion. Considerations include:

- Ingestion of various file types (e.g. Csv, json, img, parquet or databases, Hadoop/Spark)
    - As it says on the tin, learn how to do it using Google technologies and ML frameworks.
- Database migration
- Streaming data (e.g. from IoT devices)
    - PubSub sounds like a best-fitted technology here as it scales to huge numbers of producers, but allows to process data in the most convenient way for ingestion. Whether it is streaming or batch. PubSub will protect underlying datastores (GCS, DB, BigQuery) from a flood of tiny write requests.

3.2 Data exploration (EDA). Considerations include:

- Visualization
    - Use BigQuery to explore and preprocess large amounts of data
    - **Cloud Dataprep** - is an intelligent data service for visually exploring, cleaning, and preparing structured and unstructured data for analysis, reporting, and machine learning. Because Cloud Dataprep is serverless and works at any scale, there is no infrastructure to deploy or manage. Your next ideal data

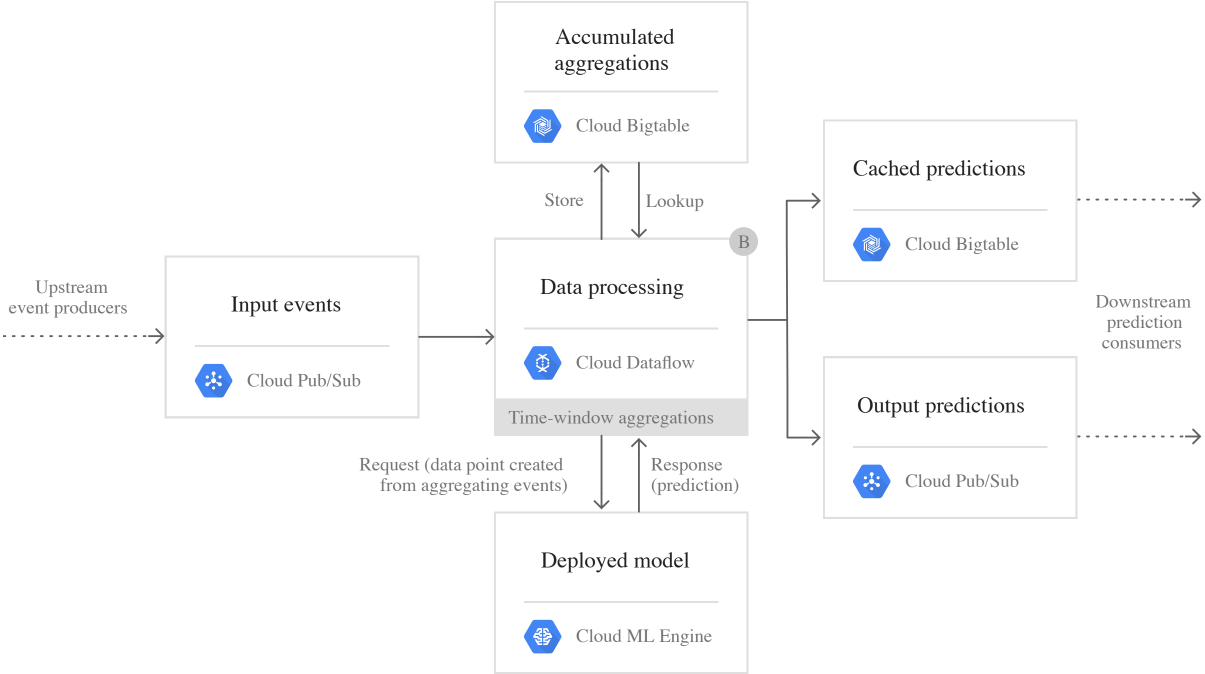
transformation is suggested and predicted with each UI input, so you don’t have to write code.

    - Helps to understand:
      - Which values happen frequently and infrequently
      - Find outliers
      - Look for missing values
      - Statistical analysis
      - Explore distributions of values
      - Collaborate with Domain Experts
    - Uses Wranglers (writes beam code in Dataflow)
      - Data ingestion
      - Data Cleansing
      - Aggregations
      - Joins, Unions
      - Transformations
      - Type Conversions
    - Wrangler examples:
      - *aggregate* - Groups values and performs aggregate functions
      - *countpattern* - Counts the number of matches
      - *deduplicate* - Removes duplicate rows, where values in every column are the same
      - *delete* - Removes rows to satisfy a condition
      - *derive* - Creates a new column with the result of a formula
      - *drop* - Removes one or more columns
  - Scatter-plot - show sample of data to see whether there are any patterns to it or it is all noise. It is not feasible to plot all the data.
  - Percentiles - group data by percentiles to visualise various groups.
  - Data Studio - free tool provided by Google to visualise data. Supports numerous sources where BigQeury and Google Sheets are very popular.
  - Facets - available as part of Google Colab.
    - Unexpected feature values
    - Missing feature values for a large number of examples
    - Training/serving skew
    - Training/test/validation set skew
  - Visualizations (Pandas Documentation)
- Statistical fundamentals at scale
  - Minimum
  - Maximum
  - Average
  - Standard deviation
- Evaluation of data quality and feasibility

  - Things commonly done in preprocessing
    - Remove examples that you do not want to train on (*BQ* or *Beam*)
    - Compute vocabularies for categorical columns (*BQ* or *Beam*)
    - Compute aggregate statistics for numeric columns (*BQ* or *Beam*)
    - Compute time-window statistics for use as input features (*Beam*)
    - Scaling, discretization, etc of numeric features (*TensorFlow* or *Beam*)
    - Splitting, lower-casing, etc of textual features (*TensorFlow* or *Beam*)
    - Resizing of input images (*TensorFlow* or *Beam*)
    - Normalizing the volume level of input audio (*TensorFlow* or *Beam*)

3.3 Design data pipelines. Considerations include:

- Batching and streaming data pipelines at scale High-level architecture using stream data for prediction in Dataflow



  - Data preprocessing for machine learning: options and recommendations
  - **Cloud Dataflow** (Apache Beam) supports both batching and streaming. Sample code:
    - Input `p = beam.Pipeline() p(`
    - Read | `beam.io.ReadFromText('gs://..')`
    - Transform | `beam.Map(Transform)`
    - Group | `beam.GroupByKey()`
    - Filter | `beam.FlatMap(Filter)`
    - Write `beam.io.WriteToText('gs://..')`
    - Output `p.run();`
  - Use `ParDo` in Java and `beam.Map` and `beam.FlatMap` in Python for stateless, distributed computation. e.g.
    - Filtering (choosing which inputs to emit)

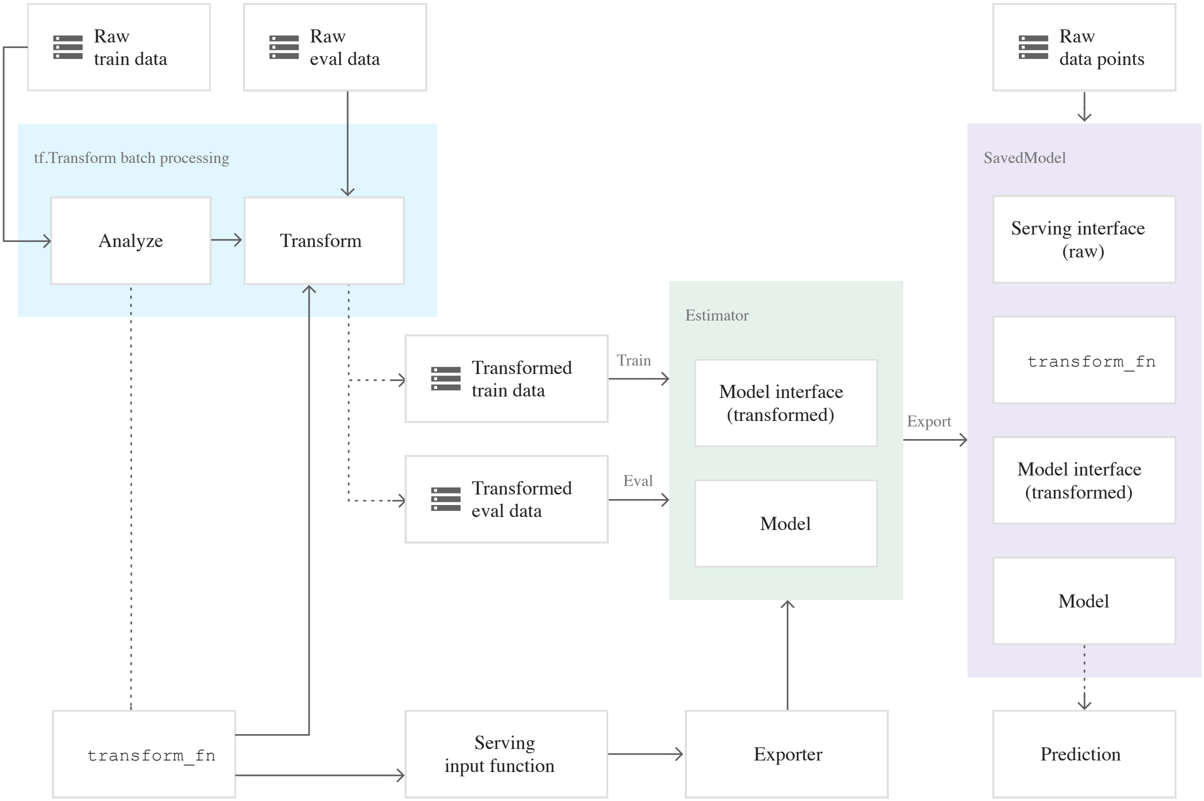
        - Extracting parts of an input (e.g. fields of TableRow)
        - Converting one Java type to another
        - Calculating values from different parts of inputs
    - Use `GroupByKey` to shuffle
    - Combine can be used for aggregation `Combine.globally(sum)` and `Combine.perKey(sum)`
- Data privacy and compliance
    - Filtering for PII
    - Cloud Data Loss Prevention
- Monitoring/changing deployed pipelines
    - Know the freshness requirements of your system.
    - Detect problems before exporting models.
    - Watch for silent failures.
    - Monitor and improve your GPU utilization
    - Monitor the resource utilization of training jobs

3.4 Build data pipelines. Considerations include:

- Data validation
    - Data and model validation
- Handling missing data
    - Avoid missing values where possible
    - Working with Missing Data (Pandas Documentation)
    - If you have sufficient data, then it might be an attractive option to simply delete samples with missing data. However, keep in mind that it could create bias in your data. Perhaps the missing data follows a pattern that you miss out on.
- Handling outliers
    - **Clipping** - A technique for handling outliers. Specifically, reducing feature values that are greater than a set maximum value down to that maximum value. Also, increasing feature values that are less than a specific minimum value up to that minimum value. For example, suppose that only a few feature values fall outside the range 40–60. In this case, you could do the following:
        - Clip all values over 60 to be exactly 60.
        - Clip all values under 40 to be exactly 40.
    - **Isolation Forest** - advanced technique. In the Isolation Forest Algorithm, the keyword is Isolation. In essence, the algorithm checks how easily a sample

can be isolated. This results in an isolation number which is calculated by the number of splits, in a Random Decision Tree, needed to isolate a sample. This isolation number is then averaged over all trees.

- Managing large samples (TFRecords)
    - Preprocess the data once and save it as a TFRecord file
    - Using Dataset API (e.g. `tf.data.TextLineDataset(file.csv).map(decode_line)`)
    - Shuffling, epochs and batching: `dataset.shuffle(1000).repeat(15).batch(128)`
    - Read a set of sharded CSV files `tf.data.Dataset.list_files("train.csv-*").flat_map(tf.data.TextLineDataset).map(decode_line)`
- Transformations (TensorFlow Transform) - Combines Beam analysis phases, with just-in-time processing of TensorFlow transform phase. Behavior of tf.Transform for preprocessing and transforming data



    - tf.rransform provides two PTransforms
        - AnalyzeAndTransformDataset - executed in beam to create the preprocessed training dataset
        - TransformDataset - executed in beam to create the evaluation dataset

3.5 Feature engineering. Considerations include:

- Data leakage and augmentation
  - When splitting data for training, validation and test it is easy to mix training data with validation/test. When that happens, live production predictions are poor in quality. Pay attention to how you split the data.
- Appropriately splitting the data.
  - What is the input data? For general regression/classification 80/20 train/validate is a good split, however time Series data cannot be be shuffled, or randomly split, sequences must be kept intact or you run the risk of losing any meaningful trends. eg. harmonic oscillation.
  - k-fold cross validation is a good trade off between decreasing the uncertainty around performance, and computational cost. However note that this is merely a validation strategy, it only increases your understanding of how well the model is performing, it DOES NOT increase the performance of the model. DO NOT confuse this with bagging which would appear similar
- Encoding structured data types
  - Introduction to Data Transformation
  - Binning - Representing floating point values or ranges of integers as binary vectors where fractions don't make sense. e.g. Longitudes Categorical features have a discreet set of possible values. We convert strings into numeric values creating a vocabulary of possible values. We can also group a long tail into OOV (Out-of-vocabulary) bucket. A better way is to use one-hot-encoding (binary vector with For values that apply to the example, set corresponding vector elements to 1, otherwise 0) and multi-hot- encoding (binary vector where multiple values are 1)
  - Sparse representation is a method of only storing non-zero values. An independent weight is still learned for each feature value.
  - Scaling feature values - converting floating-point feature values from their natural range (for example, 100 to 900) into a standard range (for example, 0 to 1 or -1 to +1) Helps gradient descent converge more quickly. Helps avoid the "NaN trap". Helps the model weight appropriate weight for each feature.
- Feature selection
  - Avoid rarely used discrete feature values (5+ appearances)
    - Good: house_type: victorian
    - Bad: unique_house_id: 12312ioho1uih
  - Prefer clear and obvious meanings
    - Good: house_age_years: 27
    - Bad: house_age: 123123
  - Don't mix "magic" values with actual data
    - Good: quality_rating: 0.82
    - Bad:quality_rating: -1
  - Account for upstream instability

        - Good: city_id: “br/sao_paulo”
        - Bad: inferred_city_cluster: “219”
    - Start with directly observed and reported features as opposed to learned features.
    - Turn heuristics into features, or handle them externally.
- **Class imbalance** - The learning phase and the subsequent prediction of machine learning algorithms can be affected by the problem of an imbalanced data set. The balancing issue corresponds to the difference of the number of samples in the different classes. With a greater imbalanced ratio, the decision function favour the class with the larger number of samples, usually referred as the majority class.
    - Imbalanced data
    - Can display very high training accuracy if there is a majority class. Use proper validation measures for the data such as Balanced Accuracy, Precision-Recall Curves, F1-score, or AUC for better insights as to model performance.
    - Use Over and Undersampling Techniques to combat.
    - More on this: over-sampling, Imballanced Classes, 4 Tips for Advanced Feature Engineering and Preprocessing
- **Feature crosses** - is a synthetic feature formed by multiplying (crossing) two or more features. Crossing combinations of features can provide predictive abilities beyond what those features can provide individually.
    - Combine and modify existing features to create new features in human-understandable ways.
    - Feature crosses use memorization, which is the opposite of ML’s goal of generalization.
    - Feature crosses + massive data is an efficient way for learning highly complex spaces.
    - Memorization works when you have lots of data
    - In TensorFlow `tf.feature_column.crossed_column([dayofweek, hourofday], 24 * 7)`
    - Interactive exercise 1 and 2

Section 4: ML Model Development

4.1 Build a model. Considerations include:

- Choice of framework and model
    - Plan to launch and iterate.
    - Keep your first model simple.
        - Output Activations
            - Is The output binary? eg. Is this a picture of Dima? Use Sigmoid
            - Is The output probabalistic? eg. Which of Dima or Steven is this a picture of? Use Softmax

- Focus on ensuring data pipeline correctness.
  - Use a simple, observable metric for training & evaluation.
  - Own and monitor your input features.
  - Treat your model configuration as code: review it, check it in.
  - Write down the results of all experiments, especially “failures.”
- Modelling techniques given interpretability requirements
  - Starting with an interpretable model makes debugging easier.
  - Scientific method applied to ML model development.
- **Transfer learning** - Transferring information from one machine learning task to another. For example, in multi-task learning, a single model solves multiple tasks, such as a deep model that has different output nodes for different tasks. Transfer learning might involve transferring knowledge from the solution of a simpler task to a more complex one, or involve transferring knowledge from a task where there is more data to one where there is less data. Most machine learning systems solve a single task. Transfer learning is a baby step towards artificial intelligence in which a single program can solve multiple tasks.
- Model generalization
  - Split dataset into Training, Validation and Test (80/10/10 split) to ensure that model performs well on the new data. Test is used as a final Go/No Go arbiter. If you have too little data to have a Test split, consider using cross-validation to have no waste. In BigQuery you can use `FARM_FINGERPRINT` function combined with `ABS` and `MOD` to split dataset.
  - Example flow:
    - Start with a NN model with one set of hyperparameters
    - Train model using training dataset
    - Increase model complexity
    - Is it *overfitting?* (evaluate using validation dataset)
      - No - go back to “Increase model complexity” step
      - Yes - Use model with hyperparameters of last non-overfit model for prediction
- Overfitting - Training Performance Excedes Test Performance
  - Can apply regularization to penalize model complexity, .i.e. generalize.
    - L1Regularization - A type of regularization that penalizes weights in proportion to the sum of the absolute values of the weights. In models relying on sparse features, L1 regularization helps drive the weights of irrelevant or barely relevant features **to exactly 0**, which removes those features from the model.
    - L2Regularization - A type of regularization that penalizes weights in proportion to the sum of the squares of the weights. L2 regularization helps drive outlier weights (those with high positive or low negative

values) **closer to 0 but not quite to 0**. L2 regularization always improves generalization in linear models.
    - Other methods include: Early stopping, Max-norm regularization, Dataset Augmentation, Noise robustness, Sparse representation.
    - Is the overfitting a result of class imbalance? Combating this may require some data engineering solutions.

4.2 Train a model. Considerations include:

- Productionizing
  - Do not use TensorFlow in Eager mode for production.
  - TensorFlow Extended (TFX) is an end-to-end platform for deploying production ML pipelines.
  - Static (offline) inference - meaning that you make all possible predictions in a batch, using a MapReduce or something similar. You then write the predictions to an SSTable or Bigtable, and then feed these to a cache/lookup table.
    - Don’t need to worry much about the cost of inference
    - Can likely use batch quota
    - can do post-verifications on data before pushing
  - Dynamic (online) inference - meaning that you predict on demand, using a server.
    - Can predict any new items on the fly
    - - compute intensive, latency sensitive - may limit model complexity
    - - monitoring needs are more intensive
- Training a model as a job in different environments
  - The best way to make sure that you train like you serve is to save the set of features used at serving time, and then pipe those features to a log to use them at training time.
  - Static training - model is trained offline. That is, we train the model exactly once and then use that trained model for a while.
    - Easy to build and test, batch training
    - Requires monitoring of inputs
    - Easy to let this grow stale
  - Dynamic training - model is trained online. That is, data is continually entering the system and we’re incorporating that data into the model through continuous updates.
    - Continue to feed in training data over time, regularly sync and update version
    - Use progressive validation
    - Needs monitoring, model rollback & data quarantine capabilities
    - Will adapt to changes, staleness issues avoided
- Tracking metrics during training

  - Monitoring training jobs
    - View status of the job `gcloud ml-engine jobs describe job_name`
    - Inspect latest logs `gcloud ml-engine jobs stream-logs job_name`
    - Filter jos `gcloud ml-engine jobs list --filter='createTime>2017-01-15T19:00'`
  - Loss functions
    - During training
    - Harder to understand
    - Indirectly connected to business goals
  - Performance Metrics
    - Types
      - Confusion matrices
      - Precision - (True Positives) / (All Positive Predictions) - When model said “positive” class, was it right?
      - Recall - (True Positives) / (All Actual Positives) - Out of all possible positives, how many did the model correctly identify?
    - After training
    - Easier to understand
    - Directly connected to business goals
- Retraining/redeployment evaluation

4.3 Test a model. Considerations include:

- Unit tests for model training and serving
  - Test the infrastructure independently from the machine learning.
- Model performance against baselines, simpler models, and across the time dimension
  - Try to quantify observed undesirable behaviour.
- Model explainability on Cloud AI Platform

4.4 Scale model training and serving. Considerations include:

- Distributed training
  - For TensorFlow - `tf.estimator.train_and_evaluate(estimator, ...)` You’ll need an estimator, RunConfig, TrainSpec and EvalSpec
    - RunConfig - tells the estimator where and how often to write Checkpoints and Tensorboard logs (“summaries”) `tf.estimator.RunConfig(model_dir=output_dir, save_summary_steps=100,save_checkpoint_steps=2000`
    - TrainSpec - tells the estimator how to get training data `tf.estimator.TrainSpec(input_fn=train_input_fn, max_steps=50000)

        - EvalSpec - controls the evaluation and the checkpointing of the model since they happen at the same time `tf.estimator.EvalSpec(input_fn=eval_input_fn, steps=100, throttle_secs=600, exporters=…)
    - **Mirrored strategy** - multiple GPUs/TPUs on a node train on a subset of data and then sync with each other (*Single node, syncronous*)
    - **Multi-worked distributed strategy** - data is split across multiple node, where each node splits data further into subset per GPU/TPU (*Multiple nodes, syncrhnous*)
- Hardware accelerators
    - TensorFlow and PyTorch benefit from accelerators.
    - scikit-learn and XGboost don't benefit from accelerators. However, scikit-learn benefits from memory-optimised machines.
    - Use the tf.data API to best utilize your accelerators
    - Use Cloud TPU for supported models
    - Use MultiWorkerMirroredStrategy for distributed training of TensorFlow models
- Scalable model analysis (e.g. Cloud Storage output files, Dataflow, BigQuery, Google Data Studio)

Section 5: ML Pipeline Automation & Orchestration

5.1 Design pipeline. Considerations include:

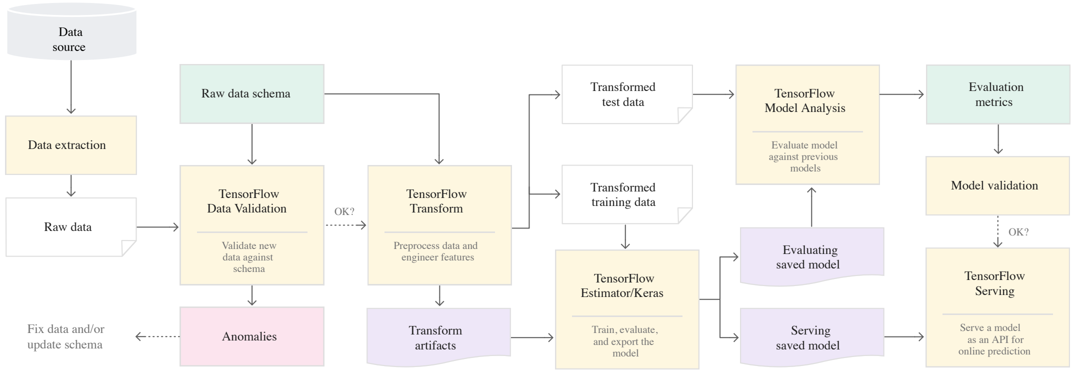
Google has a solution document just for this: Architecture for MLOps using TFX, Kubeflow Pipelines, and Cloud Build. I highly recommend going through it as it covers a large portion of sub-section considerations.

- Identification of components, parameters, triggers, and compute needs

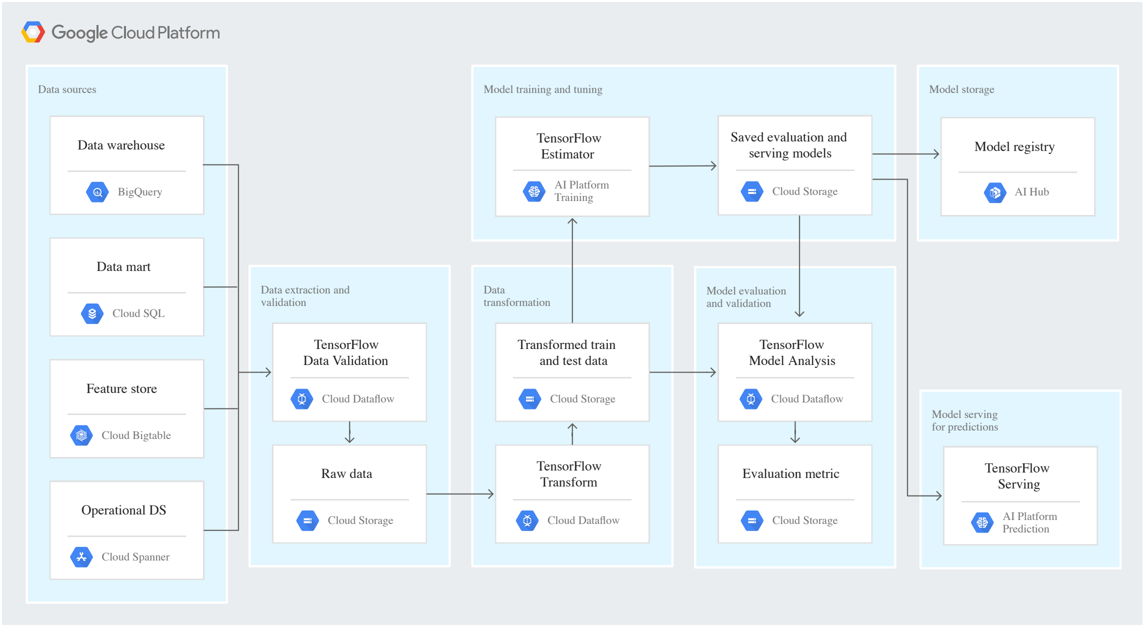
    - Kubeflow Pipeline explains it quite well. (*Completing this tutorial may take 45-60 minutes.*)
        - Define pipeline using domain specific language (DSL)
        - DSL Compile
        - Static YAML
        - Pipeline service
    - TFX on Cloud AI Platform Pipelines(Long guide)

- Orchestration framework

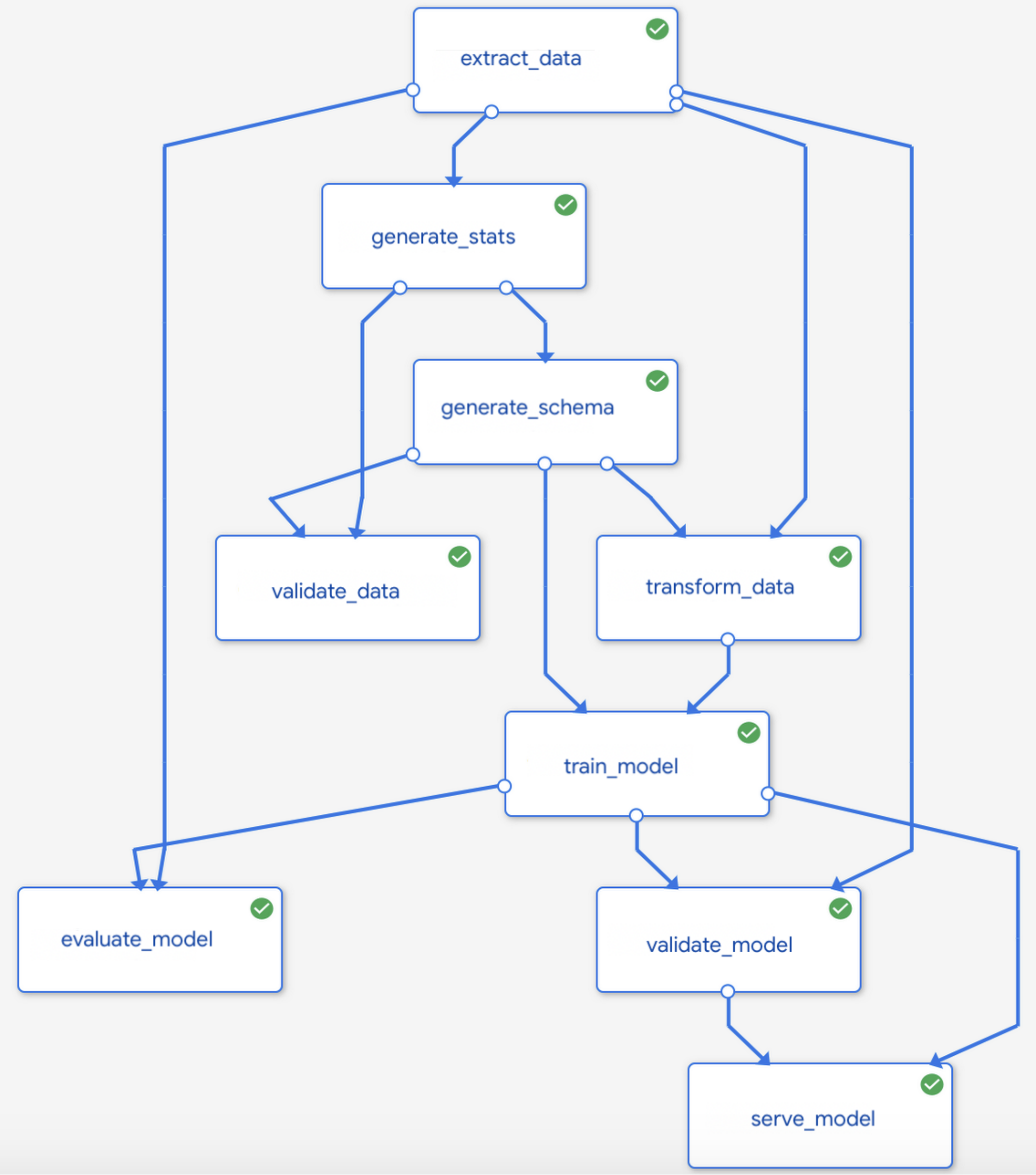
- TensorFlow Extended (TFX) TFX Extended


Data source
Data extraction
Raw data
Raw data schema
TensorFlow Data Validation
Validate new data against schema
OK?
Anomalies
Fix data and/or update schema
TensorFlow Transform
Preprocess data and engineer features
Transform artifacts
Transformed test data
Transformed training data
TensorFlow Estimator/Keras
Train, evaluate, and export the model
TensorFlow Model Analysis
Evaluate model against previous models
Evaluating saved model
Serving saved model
Evaluation metrics
Model validation
OK?
TensorFlow Serving
Serve a model as an API for online prediction

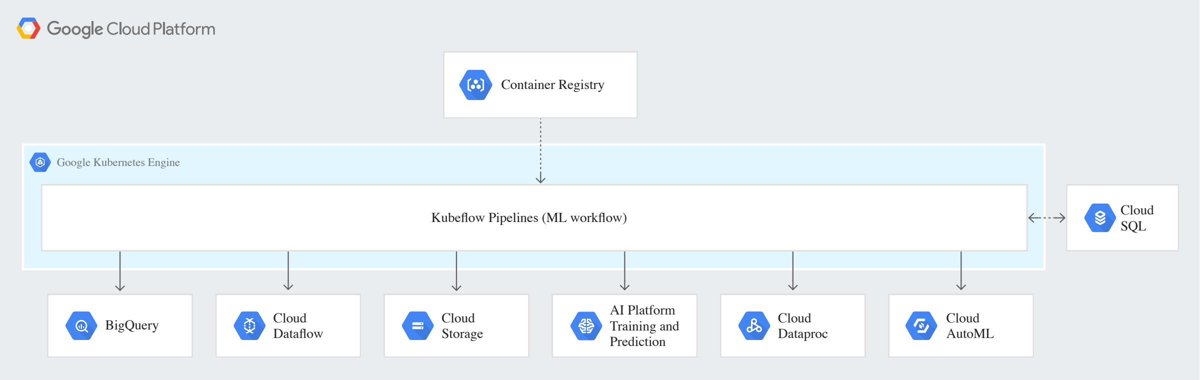
[TFX ML System on Google Cloud]



- KubeFlow A sample graph of Kubeflow Pipelines.



ML pipeline with Kubeflow Pipelines and Google Cloud managed services.



- Hybrid or multi-cloud strategies
    - End to end hybrid and multi-cloud ML workloads

5.2 Implement training pipeline. Considerations include:

- Decoupling components with Cloud Build
    - See CI/CD architecture
- Constructing and testing of parameterized pipeline definition in SDK
    - TFJob
    - Kubeflow - Building Pipelines with SDK
- Tuning compute performance
- Performing data validation
- Storing data and generated artifacts
    - GCS
    - Persistent storage
    - AI Hub

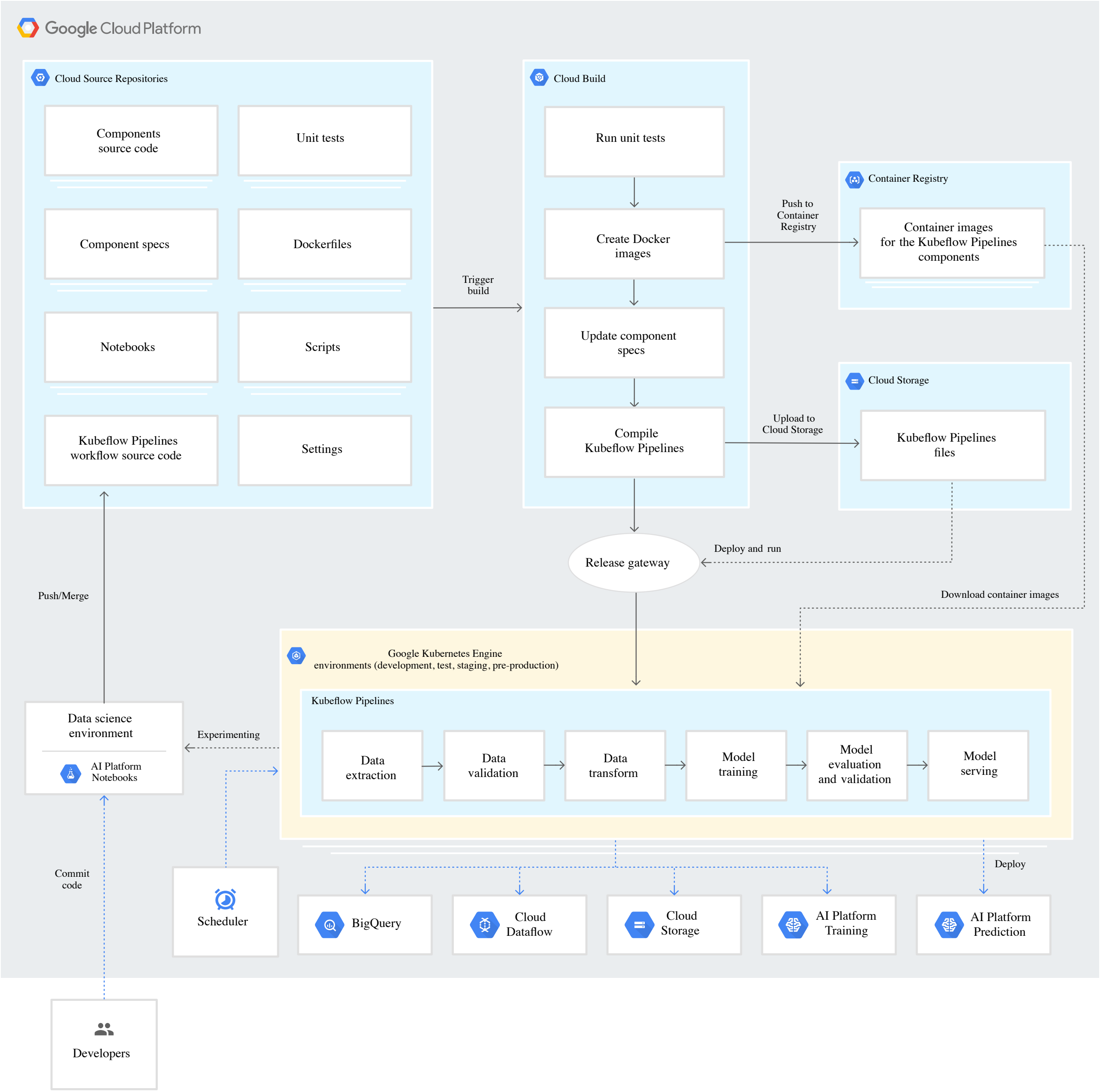
5.3 Implement serving pipeline. Considerations include:

- Model binary options
- Google Cloud serving options
    - AI Platform Predict
    - BigQuery ML
    - AutoML
    - Cloud ML
    - Host model on App Engine, Compute Engine, GKE
- Testing for target performance
- Setup of trigger and pipeline schedule
    - On a schedule, using Cloud Scheduler.
    - Responding to an event, using Pub/Sub and Cloud Functions. For example, the event can be the availability of new data files in a Cloud Storage bucket.
    - As part of a bigger data and process workflow, using Cloud Composer or Cloud Data Fusion.

5.4 Track and audit metadata. Considerations include:

- Organization and tracking experiments and pipeline runs
    - Kubeflow Experiment
- Hooking into model and dataset versioning
    - Use Code Repositories and Cloud Build for model versioning
    - Use new data triggers for dataset versioning, e.g. GCS new file event
- Model/dataset lineage
    - Metadata management

5.5 Use CI/CD to test and deploy models. Considerations include:



- Hooking modes into existing CI/CD deployment system
    - How to carry out CI/CD in Machine Learning (“MLOps”) using Kubeflow ML pipelines (#3)
    - Kubeflow (kfctl) GitHub Action for AI/ML CI/CD
    - MLOps: Continuous delivery and automation pipelines in machine learning
- AB and Canary testing
    - Split traffic in production with small portion going to a new version of the model and verify that all metrics are as expcted, gradually increase the traffic split or rollback.

Section 6: ML Solution Monitoring, Optimization, and Maintenance

6.1 Monitor ML solutions. Considerations include:

- Performance and business quality of ML model predictions

    - Measure the delta between models.
    - When choosing models, utilitarian performance trumps predictive power.
- Logging strategies
    - Joining Data Logs
    - The best way to make sure that you train like you serve is to save the set of features used at serving time, and then pipe those features to a log to use them at training time.
- Establishing continuous evaluation metrics
    - For **TensorFlow** can use TensorBoard that automatically gathers and displays metrics (metrics wary depending on the model)
        - Loss
        - Accuracy
        - Precision
        - Recall
        - Learning rate
        - DenseNeuralNetwork estimator

6.2 Troubleshoot ML solutions. Considerations include:

- Permission issues (IAM)
- Common training and serving errors
    - Re-use code between your training pipeline and your serving pipeline whenever possible.
    - Starting with an interpretable model makes debugging easier.
    - If you produce a model based on the data until January 5th, test the model on the data from January 6th and after.
    - **Shape problems** - mismatching sizes of tensors. You'll likely need to adjust the ranges. Can also happen due to the batch size, or using vector instead of scalar or vice versa.
        - tf.reshape() - can change [2,3] to [3,2]
        - tf.exapnd_dims() - inserts a dimension of 1 into a tensor's shape. (3,2) > (3,1,2)
        - tf.slice() - a way of extracting a part of a tensor
        - tf.squeeze() - removes a dimension of size 1 for the shape of a tensor
    - **Data type** - "Tensor conversion requested dtype float32 for Tensor with dtype int32"
        - tf.cast()
    - Vanishing Gradients - The gradients for the lower layers (closer to the input) can become very small. When the gradients vanish toward 0 for the lower layers, these layers train very slowly, or not at all. *The ReLU activation function can help prevent vanishing gradients.*

    - Exploding Gradients - If the weights in a network are very large, then the gradients for the lower layers involve products of many large terms. In this case you can have exploding gradients: gradients that get too large to converge. *Batch normalization can help prevent exploding gradients, as can lowering the learning rate.*
    - Dead ReLU Units - Once the weighted sum for a ReLU unit falls below 0, the ReLU unit can get stuck. It outputs 0 activation, contributing nothing to the network's output, and gradients can no longer flow through it during backpropagation. With a source of gradients cut off, the input to the ReLU may not ever change enough to bring the weighted sum back above 0.. *Lowering the learning rate can help keep ReLU units from dying.* **!!Leaky-Relu can help to address this, as can choice of optimiser eg. ADAM!!**
    - Dropout Regularization -Randomly shut off neurons for a training step thus preventing preventing training. The more you drop out, the stronger the regularization. Helps with Overfitting, too much can lead to underfitting.
- ML system failure and biases
    - **Prediction bias** - value indicating how far apart the average of predictions is from the average of labels in the dataset. Possible root causes:
        - Incomplete feature set
        - Noisy data set
        - Buggy pipeline
        - Biased training sample
        - Overly strong regularization
    - Try to avoid using **calibration layer** to fix prediction biases as they only increase complexity and do not address the root cause
    - **Data leakage** - Data leakage refers to a mistake make by the creator of a machine learning model in which they accidentally share information between the test and training data-sets.
    - **Data splits** - it is important to know what data represents to create good splits.
    - **TTraining-serving skew((is a difference between performance during training and performance during serving. This skew can be caused by:
        - A discrepancy between how you handle data in the training and serving pipelines.
        - A change in the data between when you train and when you serve.
        - A feedback loop between your model and your algorithm.

6.3 Tune performance of ML solutions for training & serving in production. Considerations include:

Best practices for performance and cost optimization for machine learning

- Optimization and simplification of input pipeline for training

    - **Reliability** - what happens when signal is not available? Would you know?
    - **Versioning** - Does the system that computes the signal ever change? How often? What would happen?
    - **Necessity** - Does the usefulness of the signal justify the cost of including it?
    - **Correlations** - Are any of my input signals so tied together that we need additional strategies to tease apart?
    - **Feedback loops** - Which of my input signals may be impacted by my model's outputs?
- Simplification techniques
    - Use reduced-precision floating-point types
    - Reduce model size using post-training quantization
    - Take advantage of the TensorFlow Model Optimization Toolkit
    - Clean up features you are no longer using.
- Identification of appropriate retraining policy
    - Optimize the frequency of retraining the model
    - ML Pipeline triggers

Find posts by tags

beta certifications cks cloud-functions cost dataflow finops flux gcp gcr gitops gke golang jekyll kubernetes label_replace logging ml prometheus recording-rules redis sensu