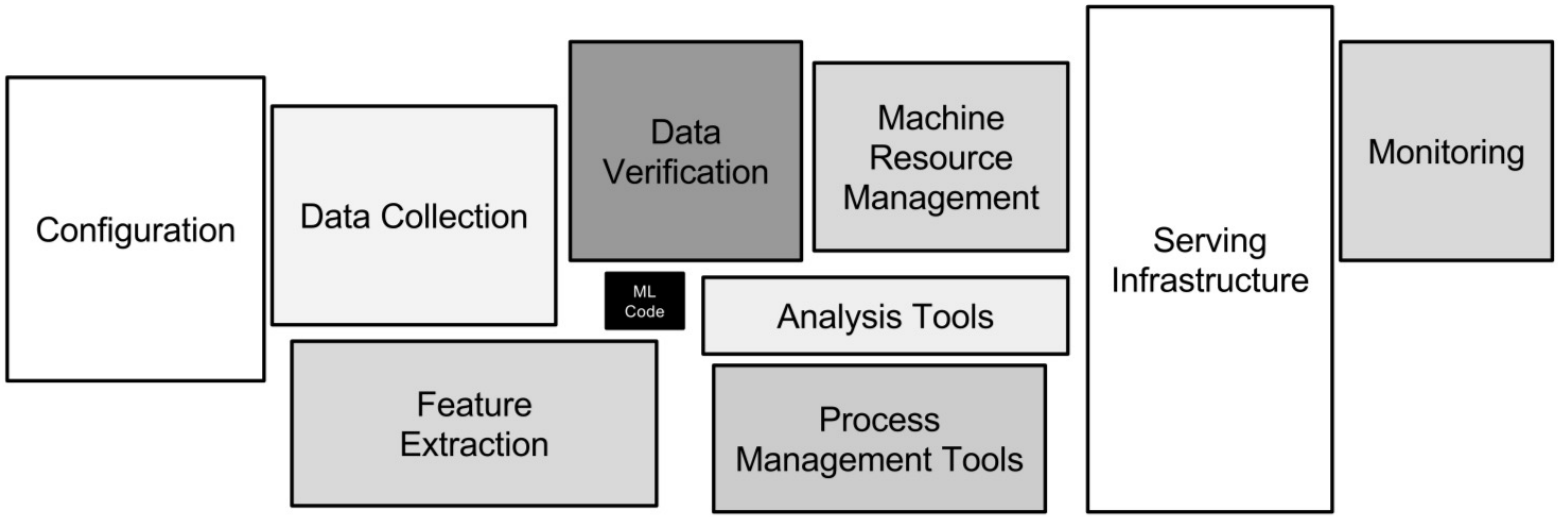
[COMSE6998-015] Fall 2024
Introduction to Deep Learning and LLM based Generative AI Systems

Parijat Dube and Chen Wang

Lecture 6 10/01/24

Operationalizing Machine Learning

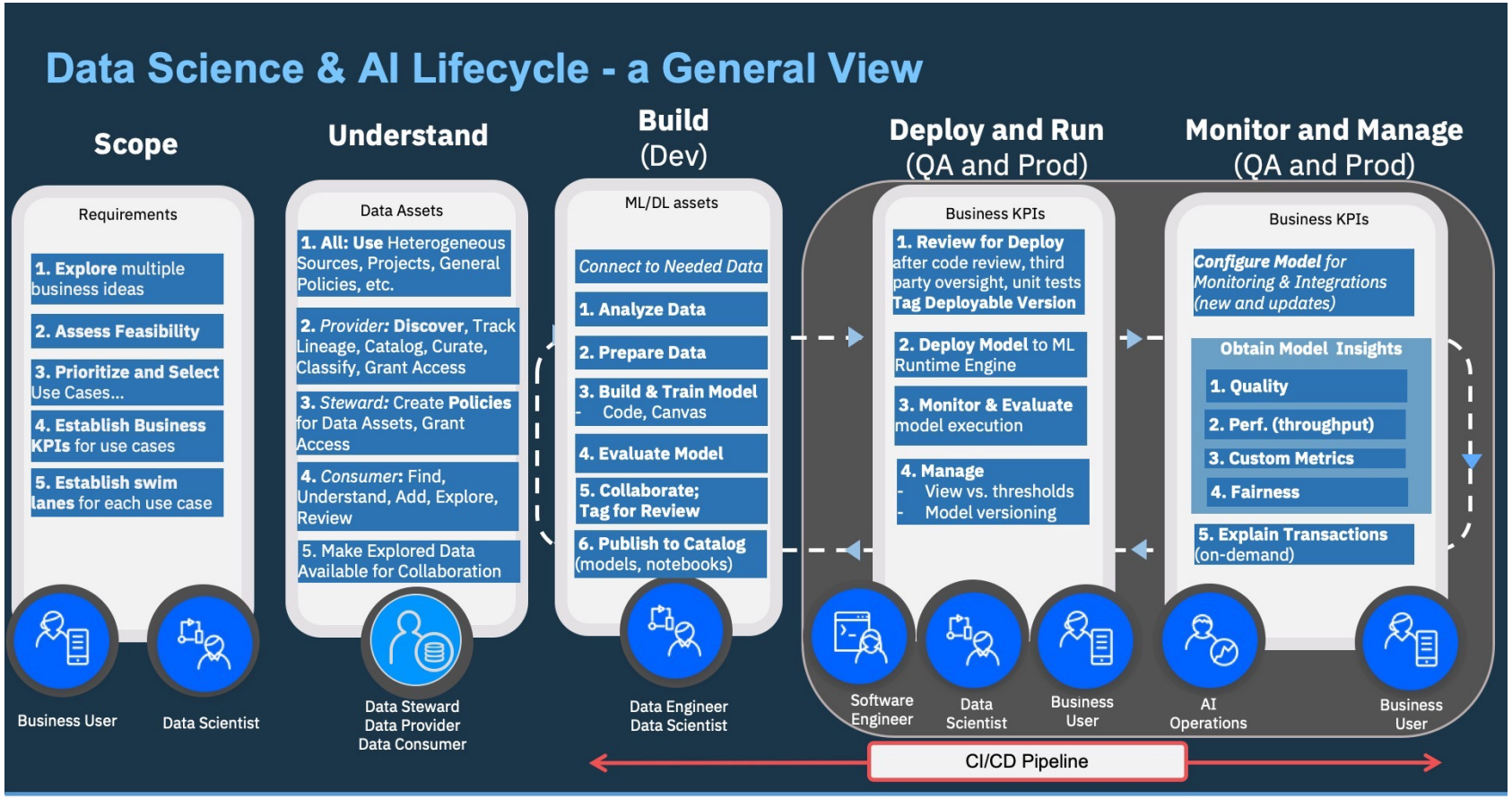
Production Grade Machine Learning Systems



The portion of ML training code in a production-grade ML system is a lot smaller than the technologies and processes needed for supporting it.

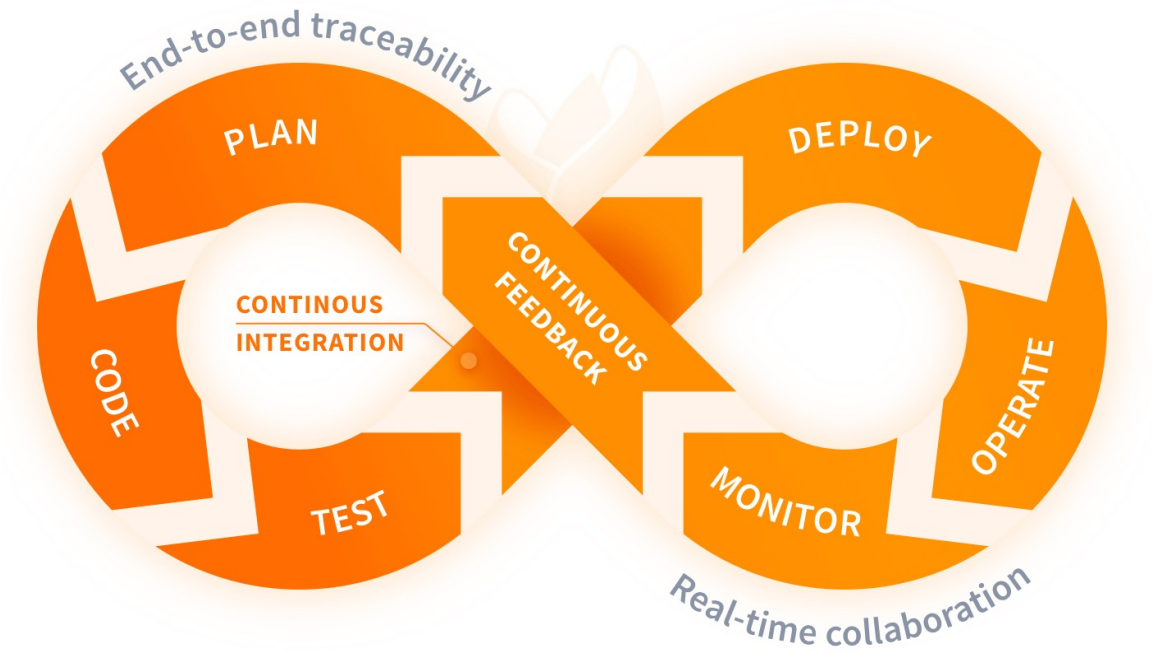
Operationalizing AI

ML Models → ML Systems → Deployed ML Service → Business Value



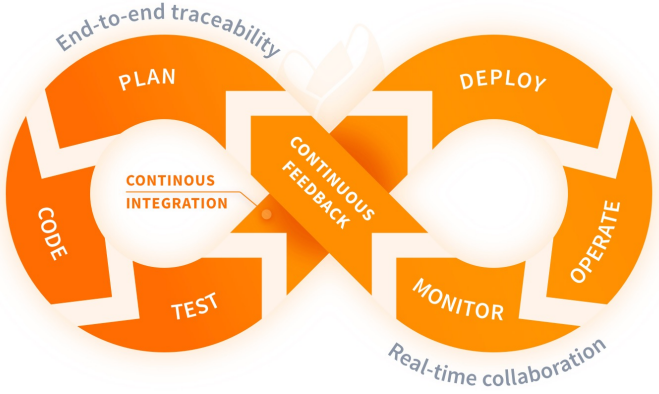
IBM

Devops principles in Software Engineering



Software Engineering in ML Systems

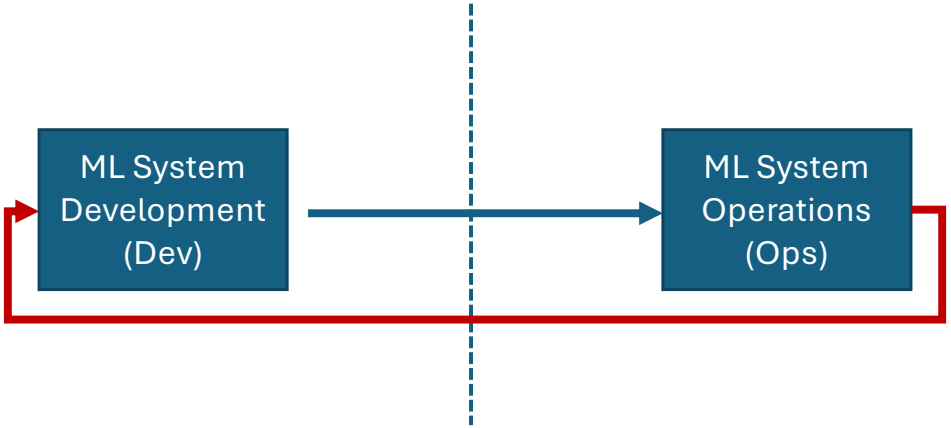
- Machine learning applications run as pipelines that ingest data, compute features, identify model(s), discover hyperparameters, train model(s), validate and deploy model(s).
- Making a model as a production-capable web service
  - Containerization (docker), cluster deployment (K8s)
  - APIs exposed as web service (Tensorflow serving/ONNX runtime)
- Workflow engines (e.g., Kubeflow) to automate ML pipeline
- Deployment monitoring and operational analytics
- Devops principles applicable to ML Systems:
  - Continuous Integration, Continuous delivery (CI/CD)
  - Predictability
    - "A model may be unexplainable—but an API cannot be unpredictable"
  - Reproducibility and Traceability
    - Provenance for Machine Learning Artifacts



ML Specific testing and monitoring apart from traditional software testing

- Data testing
- Infrastructure testing
- Model testing
- Production testing

MLOps: The Assembly Line for ML



MLOps is an ML engineering culture and practice that aims at unifying ML system development (Dev) and ML system operation (Ops)

Goal: Accelerate model life cycle (from development to deployment)
Maintain high quality model in production

Approach: automation and monitoring through development of tool-chain covering all steps of ML system construction, including development, integration, testing, releasing, deployment and infrastructure management.

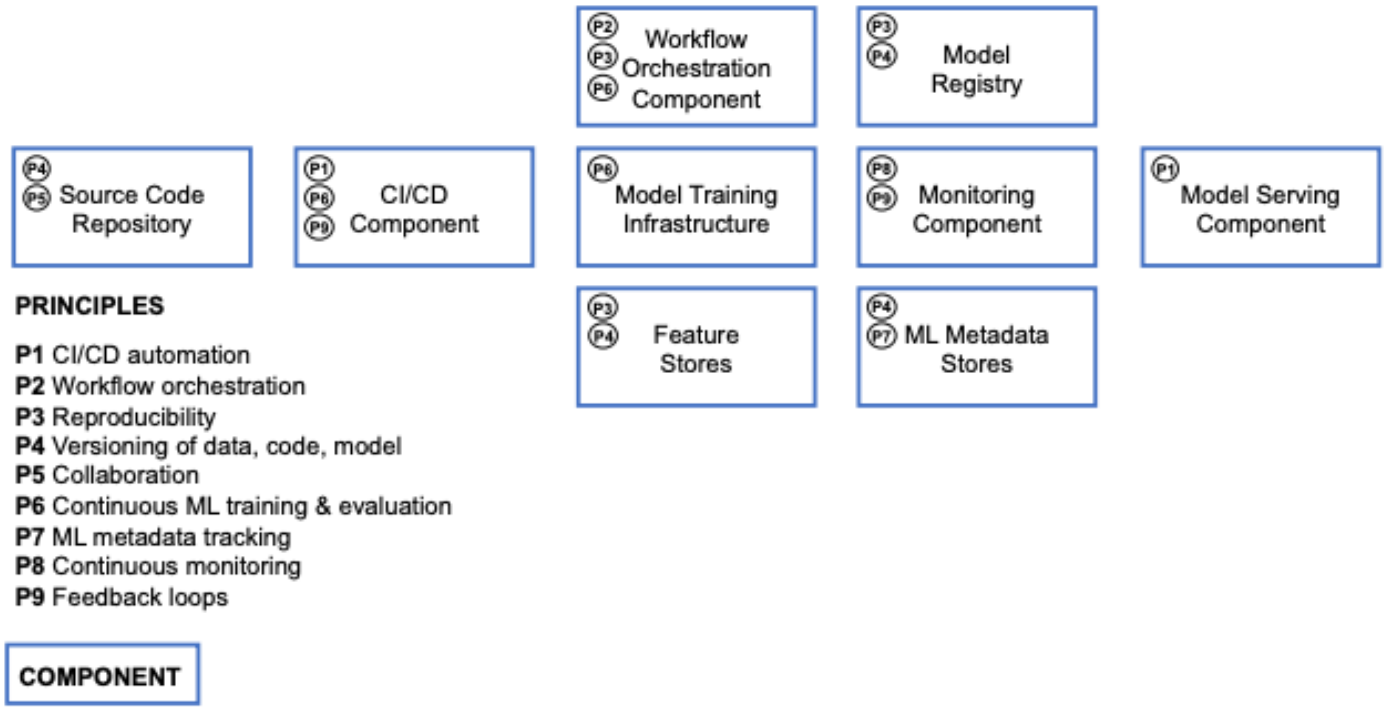
Is MLOps same as DevOps ?

An introduction to MLOps on Google Cloud

ML specific challenges to DevOps

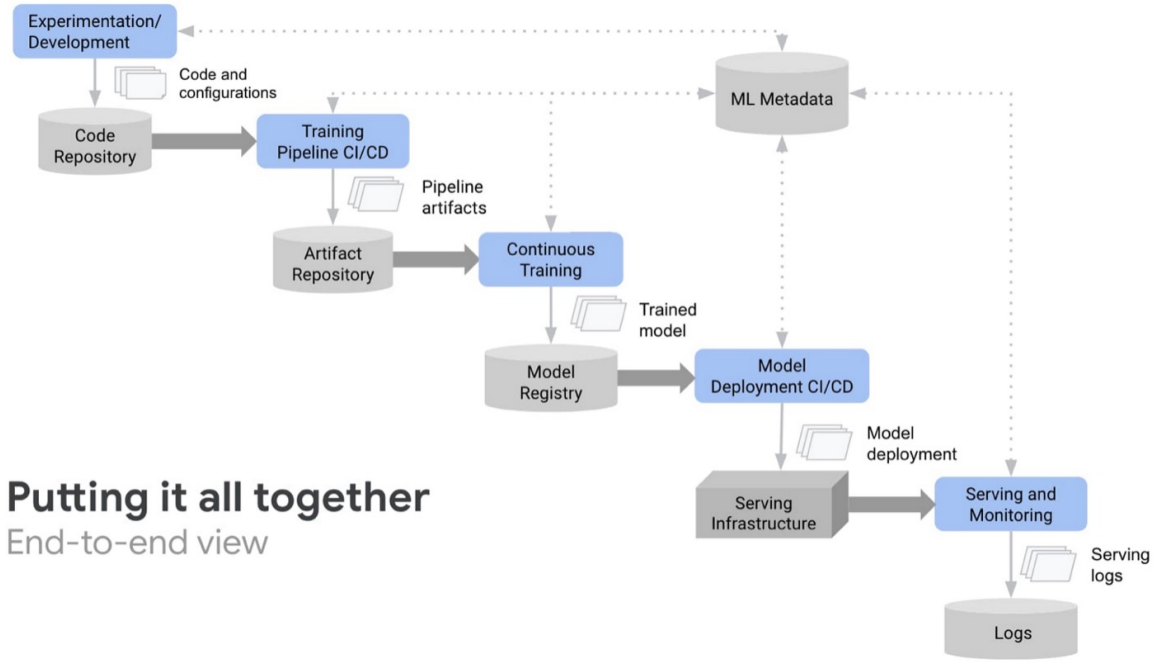
- Continuous Integration (CI) is not only about testing and validating code and components, but also testing and validating data, data schemas, and models.
- Continuous Delivery (CD) is not only about a single software package or a service, but a system (an ML training pipeline) that should automatically deploy another service (model prediction service).
- Continuous Training (CT) is a new property, unique to ML systems, that's concerned with automatically retraining candidate models for testing and serving.
- Continuous Monitoring (CM) is not only about catching errors in production systems, but also about monitoring production inference data and model performance metrics tied to business outcomes.

MLOps Principles

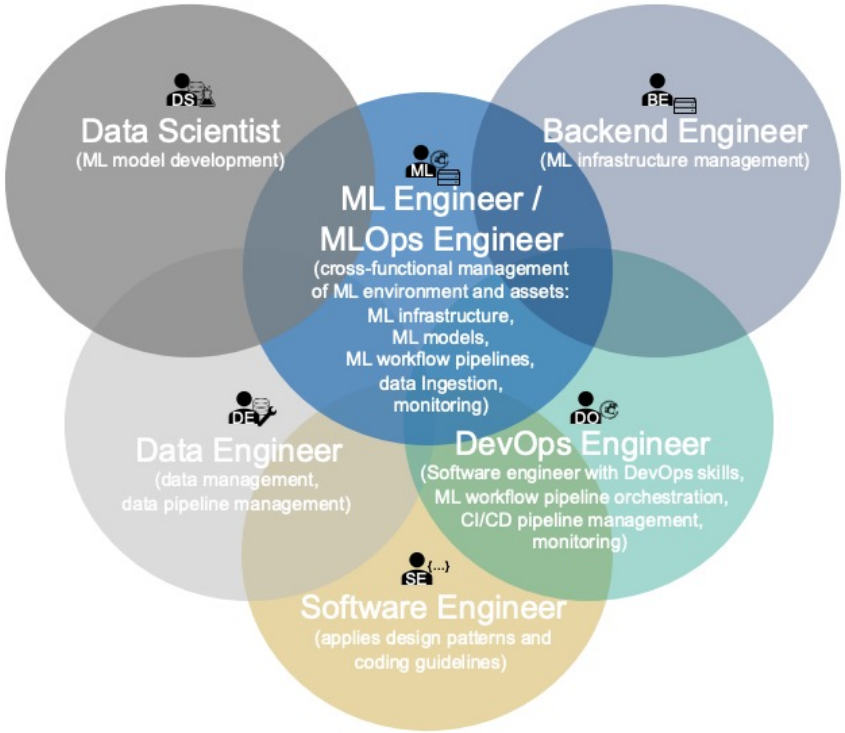


MLOps with CI/CD

Repeatable and reliable pipelines
Lineage tracking of trained models



MLOps Roles

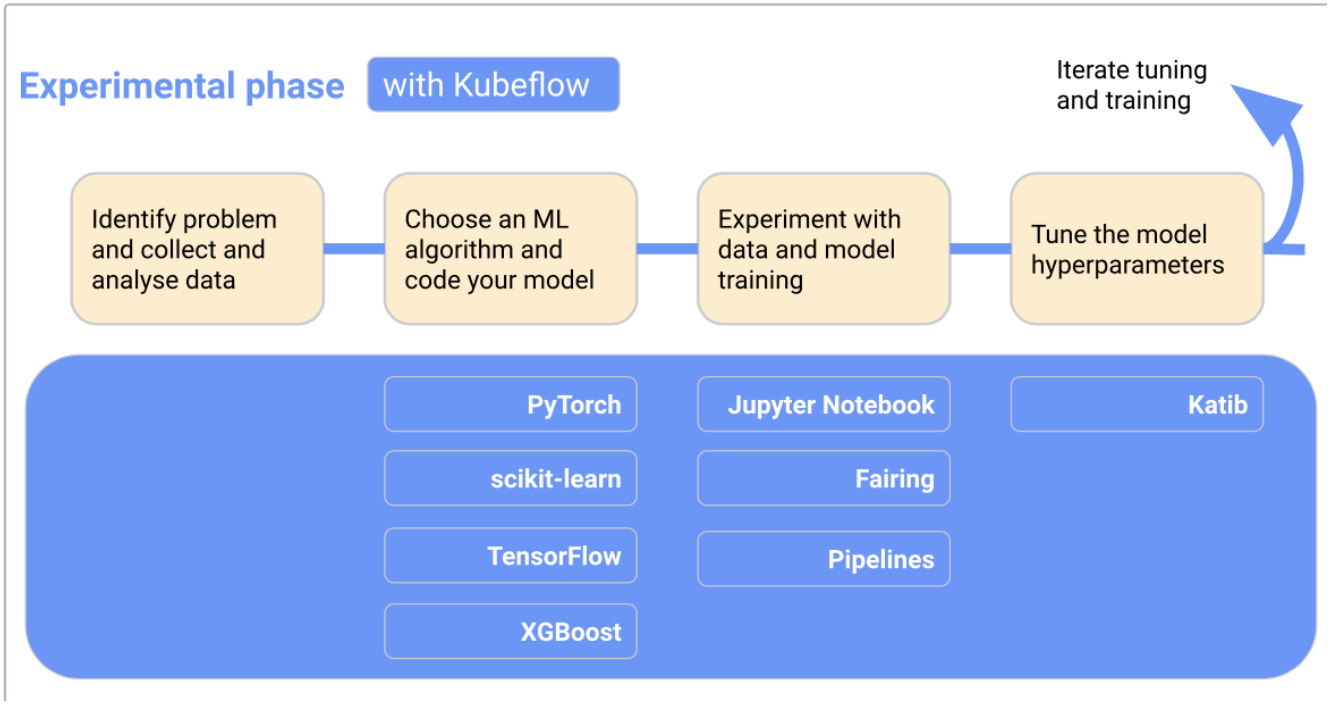

DS
Data Scientist
(ML model development)
BE
Backend Engineer
(ML infrastructure management)
ML
ML Engineer /
MLOps Engineer
(cross-functional management
of ML environment and assets:
ML infrastructure,
ML models,
ML workflow pipelines,
data Ingestion,
monitoring)
DE
Data Engineer
(data management,
data pipeline management)
DO
DevOps Engineer
(Software engineer with DevOps skills,
ML workflow pipeline orchestration,
CI/CD pipeline management,
monitoring)
SE
Software Engineer
(applies design patterns and
coding guidelines)

Kubeflow

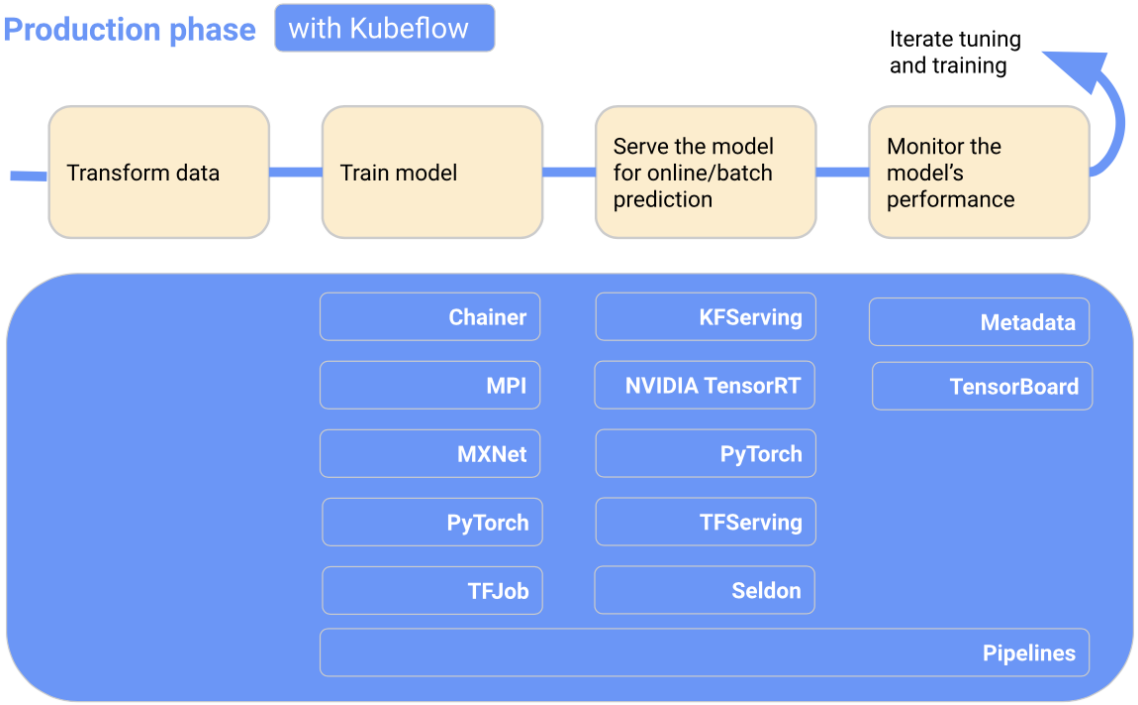
- *Ecosystem* of Kubernetes based components for each stage inthe AI/ML Lifecycle
- Kubeflow is an effort to *standardize deployment of ML apps and managing the entire lifecycle from development to production*
- Goal is to making deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable
- Multi-architecture, multi-cloud framework for running entire machine learning pipelines
- Open source; built on top of K8S

https://www.kubeflow.org/docs/about/kubeflow/

ML workflow using Kubefow – experimental phase



ML workflow with Kubeflow – production phase



Kubeflow Logical Components

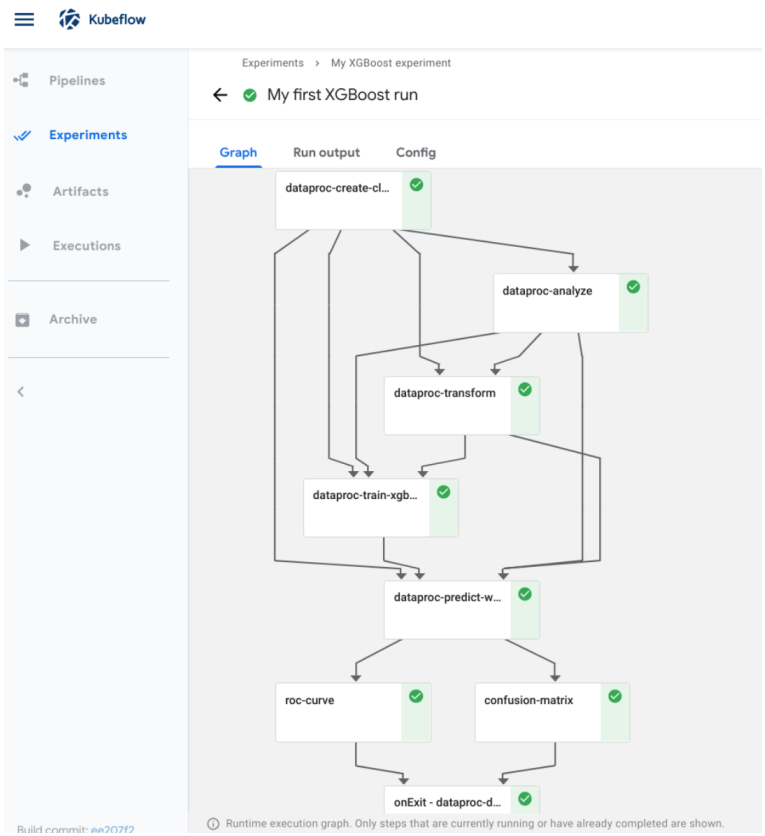
- Central Dashboard
- Kubeflow Notebooks
- Kubeflow Pipelines
- Katib (for hyperparameter tuning and NAS)
- Training Operators
- Spark Operator
- Model Registry

Kubeflow Pipelines

- A platform for building, deploying, and managing multi-step ML workflows based on Docker containers.
- The Kubeflow Pipelines platform consists of:
  - A user interface (UI) for managing and tracking experiments and runs.
  - An engine for scheduling multi-step ML workflows.
  - An SDK for defining and manipulating pipelines and components.
  - Notebooks for interacting with the system using the SDK.
- https://www.kubeflow.org/docs/pipelines/pipelines-quickstart/

Pipeline

- Description of ML workflow
- Pipeline is formed using pipeline components
- Pipeline includes
  - Components and their inputs and outputs
  - Definition of the inputs (parameters) required to run the pipeline
- https://www.kubeflow.org/docs/pipelines/overview/pipelines-overview/



Pipeline Components

- https://www.kubeflow.org/docs/pipelines/overview/concepts/component/
- A *pipeline component* is a self-contained set of user code, packaged as a Docker image, that performs one step in the pipeline.
- A component for data preprocessing, data transformation, model training, data visualization…
- A component is analogous to a function, in that it has a name, parameters, return values, and a body.
- Each component in a pipeline executes independently.
- The components do not run in the same process and cannot directly share in-memory data.

Component Specification

Example of a component specification

A component specification takes the form of a YAML file, `component.yaml`. Below is an example:

```
name: xgboost4j - Train classifier
description: Trains a boosted tree ensemble classifier using xgboost4j

inputs:
- {name: Training data}
- {name: Rounds, type: Integer, default: '30', help: Number of training rounds

outputs:
- {name: Trained model, type: XGBoost model, help: Trained XGBoost model}

implementation:
  container:
    image: gcr.io/ml-pipeline/xgboost-classifier-train@sha256:b3a64d57
    command: [
      /ml/train.py,
      --train-set, {inputPath: Training data},
      --rounds,    {inputValue: Rounds},
      --out-model, {outputPath: Trained model},
    ]
```

https://www.kubeflow.org/docs/pipelines/reference/component-spec/

Metadata (name, description), input and output interfaces, implementation (docker image url)

Real world component specifications
Link here

ML Commons

MLCommons builds and measures the following benchmark suites:

AI Safety Benchmarks
The MLCommons AI Safety benchmarks aim to assess the safety of AI systems.
Learn more →

MLPerf Inference: Mobile
The MLPerf Mobile benchmark suite measures how fast systems can process inputs and produce results using a trained model.
Learn more →

MLPerf Inference: Tiny
The MLPerf Tiny benchmark suite measures how fast systems can process inputs and produce results using a trained model.
Learn more →

MLPerf Storage
The MLPerf Storage benchmark suite measures how fast storage systems can supply training data when a model is being trained.
Learn more →

AlgoPerf: Training Algorithms Benchmark Results
The AlgoPerf: Training Algorithms benchmark measures how much faster we can train neural network models to a given target performance by changing the underlying training algorithm.
Learn more →

MLPerf Training
The MLPerf Training benchmark suite measures how fast systems can train models to a target quality metric.
Learn more →

MLPerf Training: HPC
The MLPerf HPC benchmark suite measures how fast systems can train models to a target quality metric.
Learn more →

MLPerf Inference: Datacenter
The MLPerf Inference: Datacenter benchmark suite measures how fast systems can process inputs and produce results using a trained model.
Learn more →

MLPerf Inference: Edge
The MLPerf Edge benchmark suite measures how fast systems can process inputs and produce results using a trained model.
Learn more →

https://mlcommons.org

125+
MLCommons Members and Affiliates

6
Benchmark Suites

56,000+
MLPerf Performance Results to-date

Benchmark ML Training

- After an ML practitioner selects a data set, optimizer, and DNN model, the system trains the model to its state-of-the-art quality (e.g., Top-1 accuracy for image classification)
- Provided the system meets this requirement, the practitioner can make different operation, implementation, and numerical-representation choices to maximize system performance—that is, how fast the training executes.
- An ML performance benchmark must ensure that systems under test achieve state-of-the-art quality while providing sufficient flexibility to accommodate different implementations.
- Tradeoff between quality and performance is challenging because multiple factors affect both the final quality and the time to achieve it.

MLPerf Training

- Benchmark suite that measures how fast systems can train models to a target quality metric

Benchmark	Data set	Model	Quality Threshold
Image classification	ImageNet (Deng et al., 2009)	ResNet-50 v1.5 (MLPerf, 2019b)	74.9% Top-1 accuracy
Object detection (lightweight)	COCO 2017 (Lin et al., 2014)	SSD-ResNet-34 (Liu et al., 2016)	21.2 mAP
Instance segmentation and object detection (heavyweight)	COCO 2017 (Lin et al., 2014)	Mask R-CNN (He et al., 2017a)	37.7 Box min AP, 33.9 Mask min AP
Translation (recurrent)	WMT16 EN-DE (WMT, 2016)	GNMT (Wu et al., 2016)	21.8 Sacre BLEU
Translation (nonrecurrent)	WMT17 EN-DE (WMT, 2017)	Transformer (Vaswani et al., 2017)	25.0 BLEU
Recommendation	MovieLens-20M (GroupLens, 2016)	NCF (He et al., 2017b)	0.635 HR@10
Reinforcement learning	Go (9x9 Board)	MiniGo (MLPerf, 2019a)	40.0% Professional move prediction

MLPerf Inference

- Benchmark suite that measures how fast systems can process inputs and produce results using a trained model

Area	Task	Reference Model	Data Set	Quality Target
Vision	Image classification (heavy)	ResNet-50 v1.5 25.6M parameters 8.2 GOPS / input	ImageNet (224x224)	99% of FP32 (76.456%) Top-1 accuracy
Vision	Image classification (light)	MobileNet-v1 224 4.2M parameters 1.138 GOPS / input	ImageNet (224x224)	98% of FP32 (71.676%) Top-1 accuracy
Vision	Object detection (heavy)	SSD-ResNet-34 36.3M parameters 433 GOPS / input	COCO (1,200x1,200)	99% of FP32 (0.20 mAP)
Vision	Object detection (light)	SSD-MobileNet-v1 6.91M parameters 2.47 GOPS / input	COCO (300x300)	99% of FP32 (0.22 mAP)
Language	Machine translation	GNMT 210M parameters	WMT16 EN-DE	99% of FP32 (23.9 SacreBleu)

MLPerf Storage

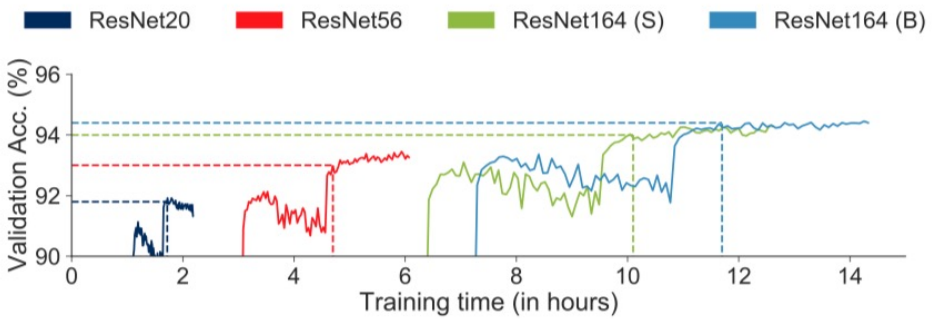
- Benchmark suite measures how fast storage systems can supply training data when a model is being trained.

https://mlcommons.org/benchmarks/storage/

Time to Accuracy (TTA) Metric

- TTA measures time for a system to train to a target, near-state-of-the-art accuracy level on a held-out dataset
- TTA combines both generalization and speed
- Dawnbench was the first multi-entrant benchmark competition to use the TTA metric
- MLPerf benchmark also uses TTA as its primary metric
- Entries compete to achieve target accuracy in the fastest time
  - Imagenet training from 30 mins to less than 2 mins
  - Very large-scale distributed training, with large batch sizes, GPUs, CPUs, TPUs
  - Major companies Google, Intel, NVIDIA compete with optimized solutions with the goal to reduce TTA
  - Entries provide an opportunity to study ML systems optimized heavily for *training performance*

Identifying Optimal DL Architecture



Accuracy threshold	Time to threshold	Fastest model
91.8%	1h 43m	ResNet20
93%	4h 42m	ResNet56
94%	10h 6m	ResNet164 (simple)
94.4%	11h 42m	ResNet164 (bottleneck)

Figure 6: Validation accuracy vs. training time for different ResNet architectures on CIFAR10. Horizontal lines indicate accuracy thresholds of 91.8%, 93%, 94%, and 94.4%. ResNet20, ResNet56, ResNet164 (with simple building blocks), and ResNet164 (with bottleneck building blocks) are fastest to the corresponding accuracy thresholds.

For lower accuracy thresholds, shallower architectures reach the threshold faster.

Coleman et al. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. NIPS 2017

Training Cost vs Training Time

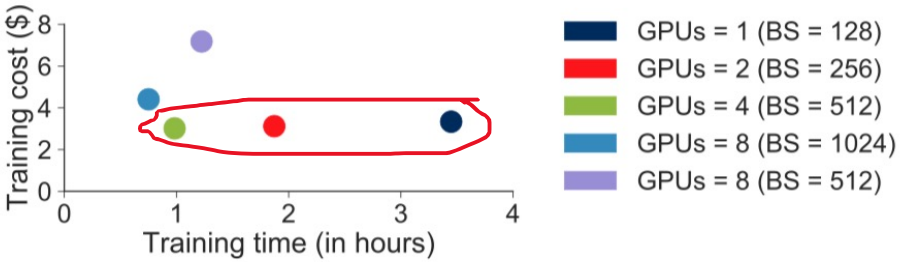


Figure 7: Training cost vs. training time for ResNet56 on the CIFAR10 dataset, using different numbers of GPUs, with an accuracy threshold of 92.5%. The cost of training stays roughly the same, regardless of the number of GPUs used, until 8 GPUs. Training time scales almost linearly with the inverse of the number of GPUs.

- Scaling from 1 to 4 GPUs
  - Training time scales perfectly linearly with the inverse of the number of GPUs used
  - Cost remains constant despite training time going down
- Scaling to 8 GPUs
  - Increase in the cost of training
  - Training time does not decrease enough to counter the doubling in instance cost per unit time (related to scaling efficiency)

Coleman et al. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. NIPS 2017