

1. MLOps platforms and the ML lifecycle

The purpose of an MLOps platform is to automate tasks involved in building ML-enabled systems and to make it easier to get value from ML. There are many steps involved in building ML models and getting value from them: steps such as exploring and cleaning the data, executing a long-running training process and deploying and monitoring a model. An MLOps platform can be thought of as a collection of tools for achieving the tasks involved in getting value from ML. But a good platform is not only a collection of tools, the tools should fit together into a shared approach that provides consistency, both end-to-end consistency in how activities are handled and also consistency across the organization as a single platform for different projects and departments.

Here is an end-to-end picture of the ML lifecycle:





To get a more detailed picture of what MLOps platforms do, we need to understand the ML lifecycle — the process of building and getting value from ML. If there were a common understanding of the ML lifecycle, MLOps platforms would be easier to understand than they currently are. Unfortunately, the industry hasn't agreed on the details. If you search for 'ML lifecycle', you'll find diagrams that only roughly agree on where the lifecycle begins and ends and what activities fall under it.

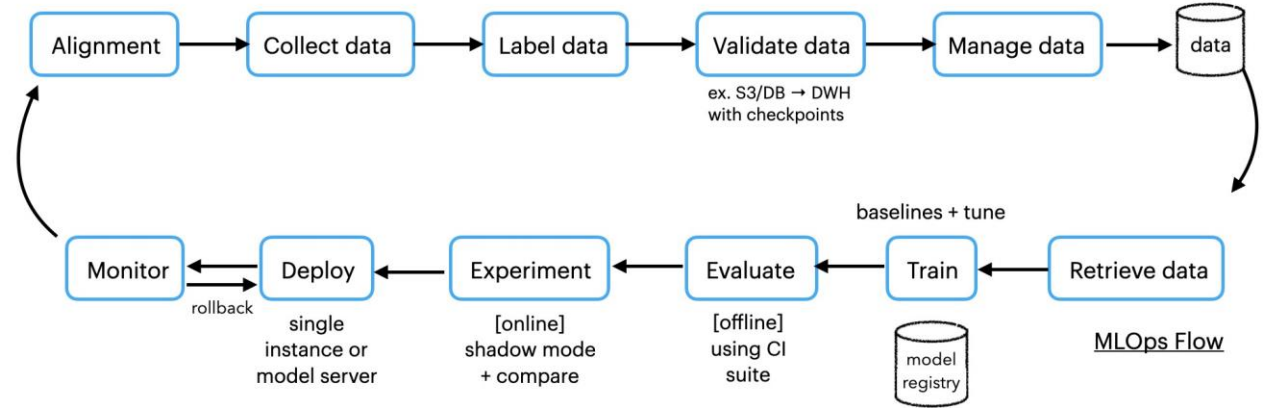
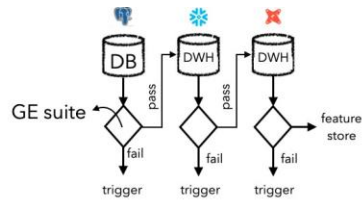
Notice this lifecycle starts with discovery activities based on data — finding data, exploring it and understanding the problem. The lifecycle here ends with monitoring — observing the performance of the model in live and gathering data to assess performance. The arrow back to the beginning represents improvements that can be made in the light of performance data.

It's important to emphasize that the ML lifecycle diagram begins with discovery activities and includes a feedback loop from monitoring. Not every ML project will require this but too many projects underestimate the need to handle on-going change and management of data. If this is not all handled by an MLOps platform itself, then it can be handled with integrations to other tools and platforms. An evaluation process needs to identify these points of integration.

DataOps Flow

Business metrics  System metrics

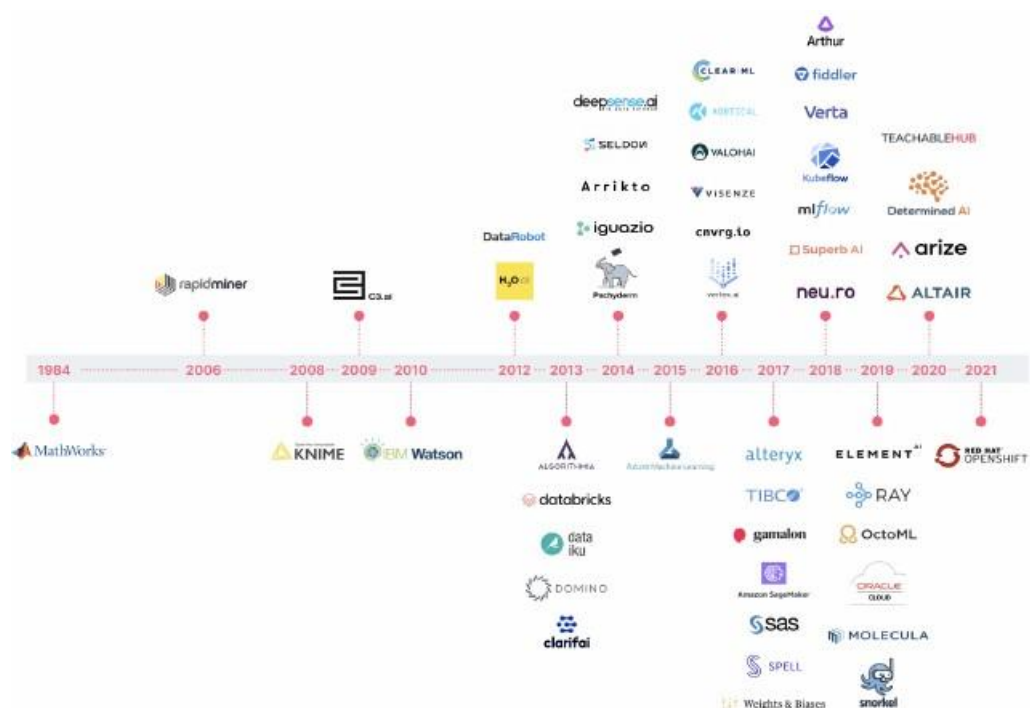
 Extract, generate, etc. and store in DB, DWH, object store, etc.



2. MLOps Platform Landscape

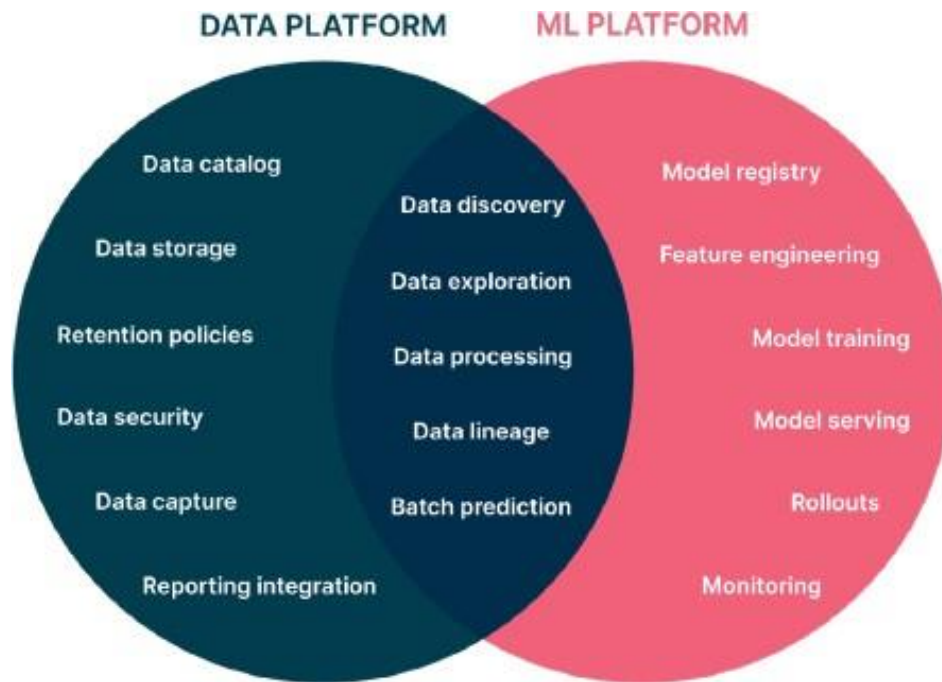
- MLOps Platform timeline(see page 3)
- Overlaps between Data and ML platforms(see page 3)
- Who performs which MLOps tasks?(see page 4)
- Making sense of the landscape(see page 5)
- Installation and integration models(see page 8)
- Choosing the right MLOps platform(see page 8)
- Structuring an MLOps platform evaluation(see page 9)
- MLOps platform Workflow(see page 10)
- References(see page 10)

2.1 MLOps Platform timeline



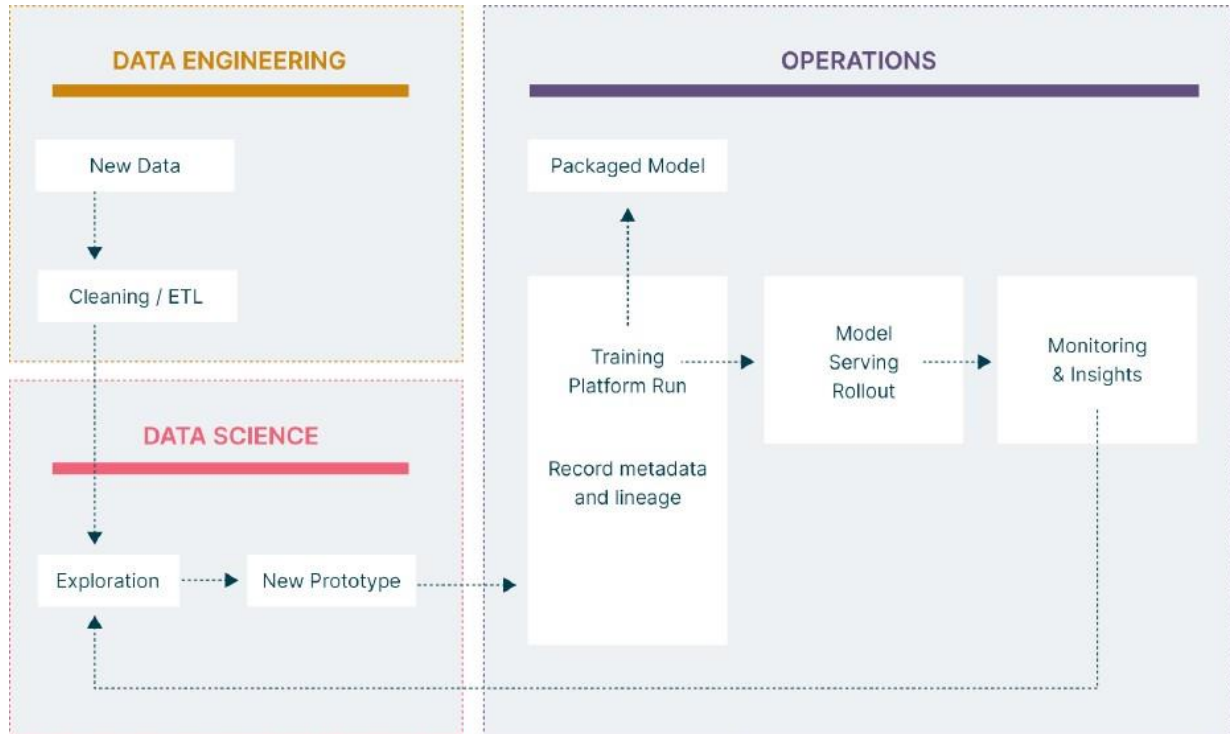
2.2 Overlaps between Data and ML platforms

For organizations that already have a data platform, overlaps lead to questions about what parts of an MLOps platform are needed by the organization. Tools used for data processing and ETL, such as Airflow, can also be used for automating training and delivering machine learning models. Some organizations might therefore find it sufficient to train models with Airflow. If such an organization also has adequate ways of deploying and monitoring ML models then they may not need to introduce an end-to-end MLOps platform.



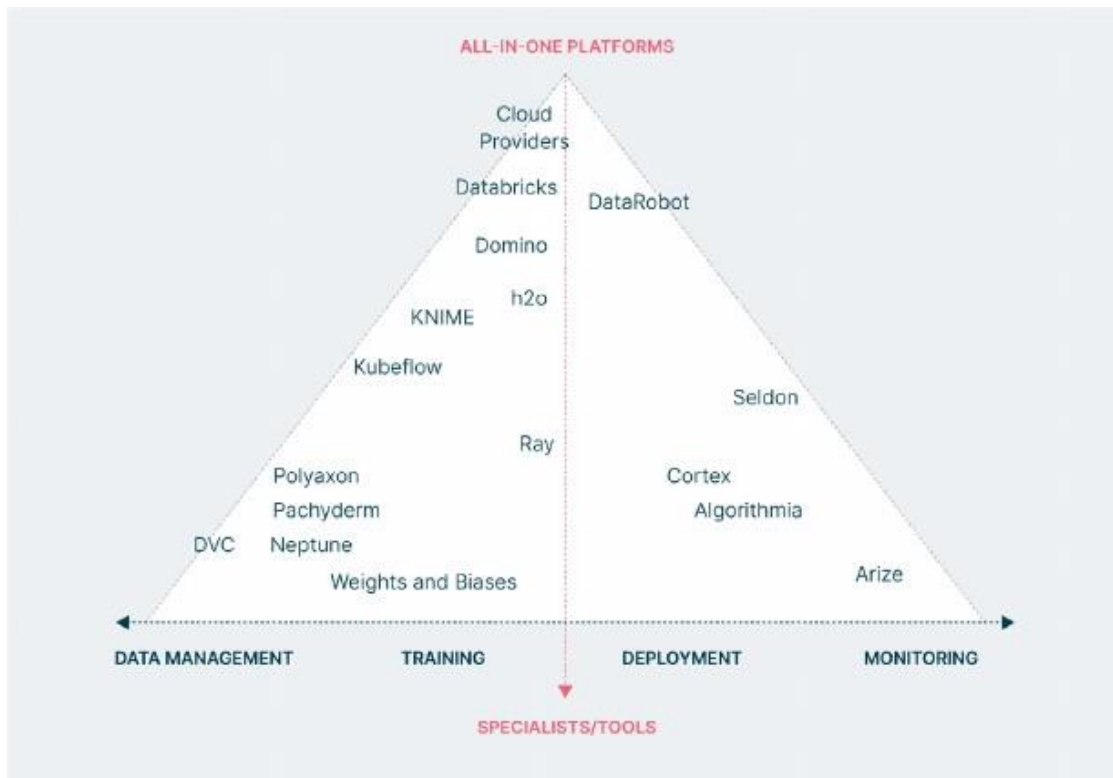
2.3 Who performs which MLOps tasks?

Another axis of variation is who the MLOps platform serves. In some organizations one might find work on ML products divided as follows:



2.4 Making sense of the landscape

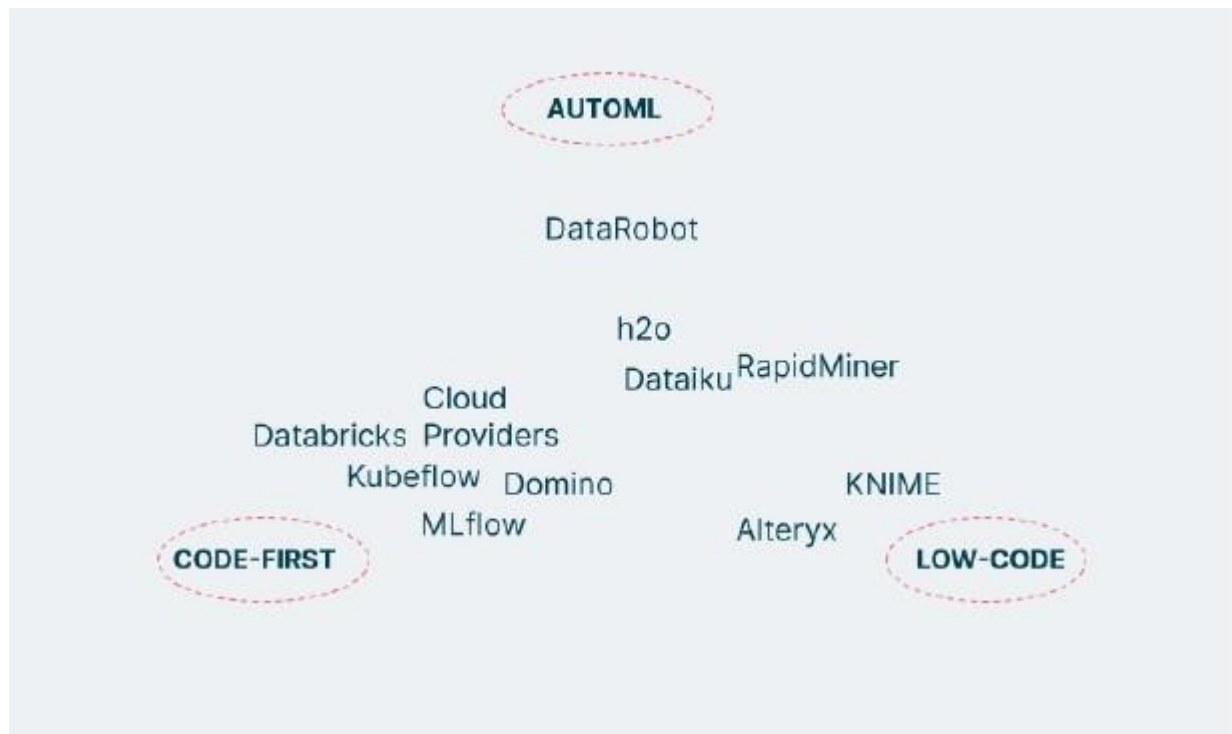
MLOps software as a pyramid with tools targeted at lifecycle stages and all-in-one platforms targeting all stages (but mostly with a certain leaning). The products on the bottom-right are focused on deployment and monitoring, those on the bottom-left focus on training and tracking. Those at the very top do the whole spectrum. Those in the middle-top do most or all of the spectrum with a leaning one way or another.



MLOps software with poles of specialization in lifecycle stages and all-in-one platforms gravitating towards poles (or middle if no particular pole), In this way of thinking, the most balanced all-in-one platforms are in the middle if they cover different aspects equally. Specialized products are closest to their pole of specialization, though many specialize in more than one thing.



MLOps software with poles of specialization in approach with vendors gravitating towards poles (or middle if no particular pole)



- In a code-first approach, data scientists can create models using whatever code-based tools they choose.
- An AutoML approach uses wizards to configure what search the platform will perform to find the best model to fit the supplied data.
- A low-code approach to model building provides a visual drag-and-drop environment to create the model training workflow from configurable steps.

Most vendors offer a mixture of these approaches, with many putting most emphasis on one of the three.

2.5 Installation and integration models

MLOps platforms typically have a strategy for how they will be installed and what integrations they offer. Imagine that an organization is evaluating the popular open source options of Kubeflow and MLflow. If the organization is using Kubernetes already, then Kubeflow being targeted at Kubernetes might be appealing, as might Kubeflow's distributions for different cloud providers. If the organization is not using Kubernetes, MLflow's more neutral approach might be more appealing.

Other organizations may not want to install and manage a platform themselves. For example, they may not want to deal with the low-level resource management or the need to handle updates. In such cases, a hosted PaaS might be more appealing. Some platforms have a 'one-stop-shop' strategy. They try to meet all MLOps needs in a single place. This can fit well with a PaaS model as organizations who want everything from one platform often don't want to operate the infrastructure for it. This model tends to be lighter on integration options.

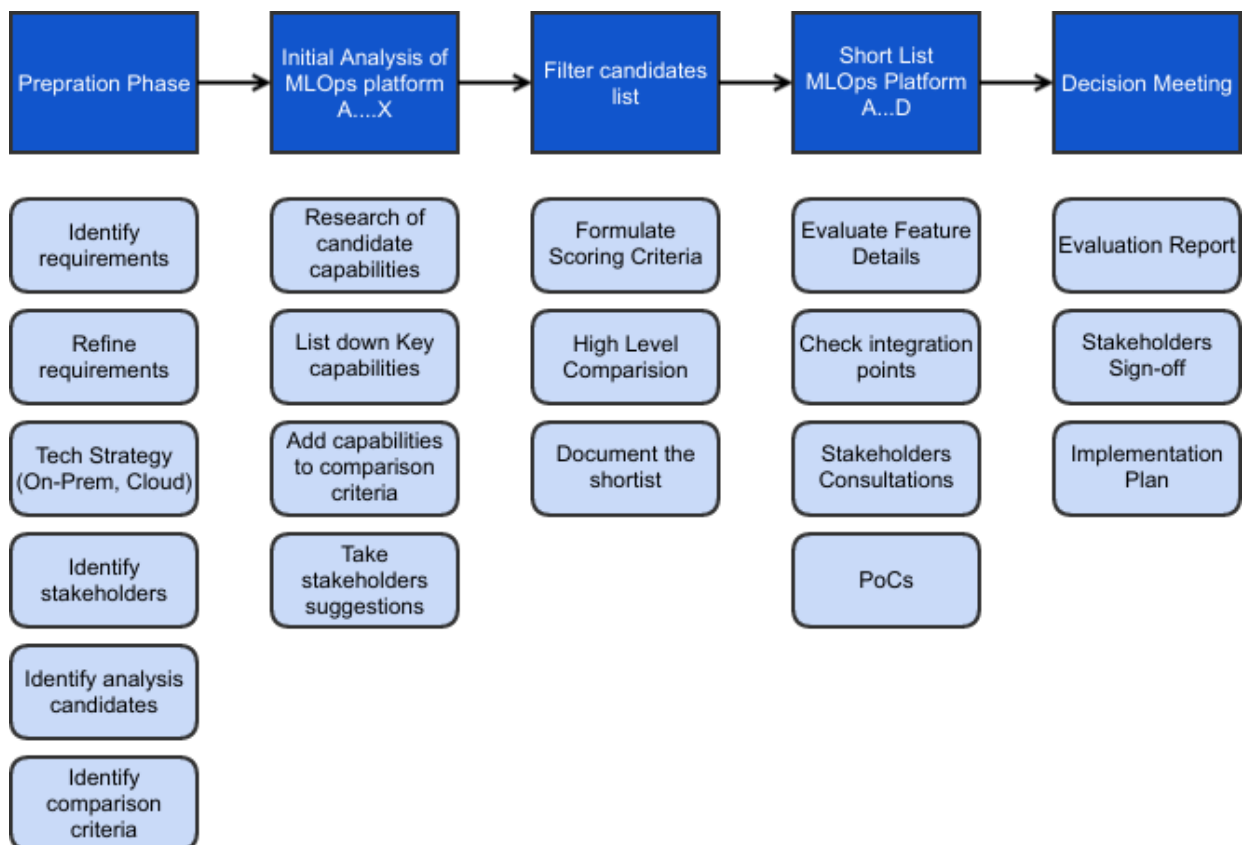
2.6 Choosing the right MLOps platform

It can be difficult to decide on the best buying criteria for a platform and it can be tempting to look for 'completeness' of features. It is risky to allow breadth of features to dominate considerations because:

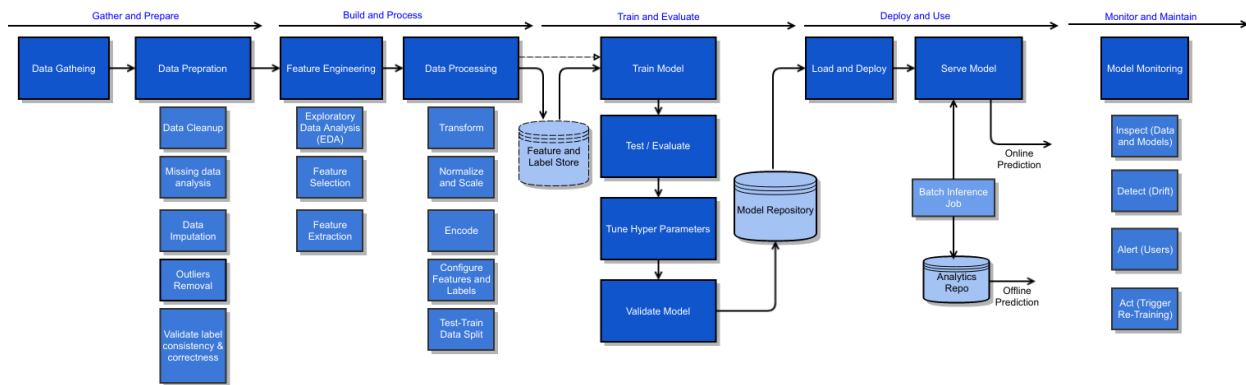
- All platforms are adding features over time.
- Range of features can come at a cost of inflexibility.
- Many use cases don't require a wide range of features from a platform.
- Where required features are missing, it is often sufficient to stitch together tools.
- The variation of use cases makes 'complete' impossible to define in a neutral and detailed way.

2.7 Structuring an MLOps platform evaluation

Flow of an MLOps platform evaluation with artifacts and activities



2.8 MLOps platform Workflow



2.9 References

- <https://ljvmiranda921.github.io/notebook/2021/05/10/navigating-the-mlops-landscape/>
- <https://ljvmiranda921.github.io/notebook/2021/05/15/navigating-the-mlops-landscape-part-2/>
- <https://ljvmiranda921.github.io/notebook/2021/05/30/navigating-the-mlops-landscape-part-3/>

3. MLOps Platform Requirements

Requirements	Area	Sub-Area	Details
Deployment options			<ul style="list-style-type: none">• On-prem and public cloud deployments.• Deployment on multiple cloud platforms (AnyCloud Strategy)
Managing Data	Data Ingestion		<ul style="list-style-type: none">• Data Lake capabilities (capability to store and process petabytes of data).• Support for online Raw Data - Event streaming and Stream processing capabilities.• Support for offline Raw Data - Historical Data. Data in files, APIs, etc.. Push / pull possible.• Data Sources connectors and pluggability.- A pluggable adapter to fetch data from different data sources with known/unknown schema and transforms/structure the structured/unstructured data.• Dedicated data pipeline for each Data Source so that so all of them are processed and stored independently and concurrently.• Capability to configure data pipelines at runtime.• Data pipeline should be capable of scheduling to fetch the data at frequent intervals of time.• Notification for further processing that data is ingested/available in data store.
	Data Storage		<ul style="list-style-type: none">• Storage for pre-processed/processed data.• Should support data consistency, reliability, low latency/faster access in accessing/ fetching the data.• Should support export/Import and backup and restore of data.• Should be scalable from small to large volumes of data (correspondingly support very small to very large network sizes).• Configurable duration of stored data with ability for periodic cleanup.• Should be able to purge any form of unused/ old/time expired data.• Batch processing and querying capabilities.• Should be possible to integrate with customer deployed storage architecture (future).

	Data Quality and Governance		<ul style="list-style-type: none"> • Data quality APIs - Incorporate data quality monitoring. • Data analytics - descriptive analytics on the data • Auto-discovery of data schema if data schema is already present. • Provide a way create, view, and update data metadata. • Support for data modelling frameworks. • Data Governance, Catalog, data lineage.
	Data Jobs	Data Processing Orchestration	<ul style="list-style-type: none"> • Data Processing Orchestration capabilities (e.g., Airflow, Data Jobs). • Fully Automated data processing orchestration via data jobs using workflow components. • Capability to provide and manage data workflow components. • Scalability for managing hundreds of daily data jobs on terabytes of data to create features and labels.
		Data Preparation/ Processing Jobs	<ul style="list-style-type: none"> • Data validation - Ensure correctness of raw data. • Data Converter - To convert unstructured data into structured data • Data Imputation - Fill the missing data • Data Cleansing - Remove skewed data/noise/ outlier from the data. • Data Transformation - To enrich/transform/ normalize the data • Feature Extraction and Feature Tuning - Should be able to rank the impact of each feature and discard the less impactful features. • Feature amalgamation (Parameter grouping/ Feature Set) - Group different features from same data source or different data source, as a feature set identified by feature set id. • Data Segregation - To segregate Train Data and Test data.

	Feature Store		<ul style="list-style-type: none">• Availability of feature store (e.g. Feast).• Data jobs will integrate to Feature store to transform the relevant data ingested from raw data stores into features and labels we can use for our models.• Storing features/feature sets.• Support for versioning of enriched data set.• Store fully processed, scaled, normalized and transformed data.• Support custom data transforms specific to models.• Ability to serve features in a low-latency way such that predictions can be obtained for models when in production serving.• This setup ensures that the exact same features we train on will be available in production while serving inferences to clients.• Ability to selectively cleanup old data once corresponding raw data is deleted.• Feature set Ranking - Rank the feature sets based on the accuracy of results yielded by the ml models.
--	----------------------	--	---

Managing Models			<ul style="list-style-type: none"> • Support for basic machine learning Frameworks. • Support for Deep Learning Frameworks. • Scalability of individual stages in a pipeline. • Model Pipeline consistency between exploration and serving predictions in production. • Ways to explore data and model combinations. • Model Training - Should be scheduled or Triggered based on time/events. • Configurable resources for training, validation, and evaluation. • Dedicated pipeline for each Model so that they can run concurrently/in parallel. • Hyper parameter Tuning capabilities. • Model Repository - Consistent representation of models with model metadata. • Model versioning capabilities. • Data Parallelism - Should support running of training data concurrently by the model if model library supports parallelism. • Model Parallelism - Should support running of model training concurrently by the model if framework supports distributed training. • Model Evaluation/Scoring Service - Accuracy of Model is evaluated and relevant Scores are assigned to each Models along with its combined Hyper-parameters and Model Configs and then Stored in Model Repo. • Model Scoring Metadata - Should contain information like Model Identifier, No. of times model is run, Time taken for each run, distribution of features used, predicted vs actual results ratio etc. • GPU Usage, CPU/GPU switch • AutoML capabilities.
Managing Model Serving			<ul style="list-style-type: none"> • Model Deployment - Should be able to prepare containerized images of the Trained Finalized Models based on Deployment Spec for Model Server • Model Serving Framework support (e.g. TFServing, TorchServe, Sheldon Serving). • Online model hosting. • Capability of handling tens of thousands of requests per second to API endpoints. • API integration along with Hosting Endpoints.

Managing Inferences			<ul style="list-style-type: none"> • Online Inference capabilities • Batch inferences - Batch prediction and inference capability. • Analytics Storage - Storage for insights, knowledge, inferences, etc. • Support for framework for hosting the Analytics results to end users.
Managing Monitoring			<ul style="list-style-type: none"> • ML Model/System monitoring capabilities. • Drift Detection - Data Drift and Concept Drift. • ML Model decay detection and Alarming capabilities. • Auto Model retraining with most recent features and labels. • Managing the meta data related to training and inference, evaluating this data for trigger for retraining and reporting purposes. • Mechanism to auto trigger retraining that a "better" model could be readily deployed and rectify the situation.
Managing Operations	Workflow Management		<ul style="list-style-type: none"> • Workflow/Pipeline Management. • Data pipeline workflow diagrams • ML pipeline workflow diagrams
	Visualization		<ul style="list-style-type: none"> • Visualization/Insight Analytics capabilities. • Ability to visualize data trends. • Ability to visualize insights and associated data, features and algorithms. • Ability to visualize trends of predictions vs. actual data. • Interface to see history of model outputs..
	Troubleshooting		<ul style="list-style-type: none"> • Model Explainability. • AutoML Pipeline Explainability. • Debugging capabilities - Logging and Trace Viewer for Logs. • NLP on logs for better troubleshooting if any. • Alerting/Notification for Events offered

CI/CD/CT/CM			<ul style="list-style-type: none"> • Continuous Integration (CI) - the testing and validating code and components by adding testing and validating data and models. • Continuous Delivery (CD) - delivery of an ML training pipeline that automatically deploys the ML model prediction service. • Continuous Training (CT) - unique to ML systems property, which automatically retrains ML models for re-deployment. • Continuous Monitoring (CM) - monitoring production data and model performance metrics, which are bound to business metrics.
Non Functional Capabilities			<ul style="list-style-type: none"> • High Availability • Scalability • Backup and Restore • Security • Geo-Redundancy

4. MLOps Platform Details

- a. [AWS Sagemaker](#)(see page 17)
- b. [Azure Machine Learning](#)(see page 20)
- c. [Databricks](#)(see page 24)
- d. [Dataiku](#)(see page 26)
- e. [DataRobot](#)(see page 29)
- f. [Google Vertex AI Platform](#)(see page 29)
- g. [H2O.ai](#)(see page 33)
- h. [KNIME](#)(see page 38)
- i. [Kubeflow](#)(see page 43)
- j. [MLFlow](#)(see page 51)

4.1 AWS Sagemaker

- [Prepare](#)(see page 18)
- [Build](#)(see page 19)
- [Train and Tune](#)(see page 19)
- [Deploy and Manage](#)(see page 20)

AWS SageMaker aims to be both comprehensive and integrated. It has services addressed at all parts of the ML lifecycle and a variety of ways to interact with them, including its own dedicated web-based IDE (SageMaker Studio - based on [JupyterLab](#)¹).



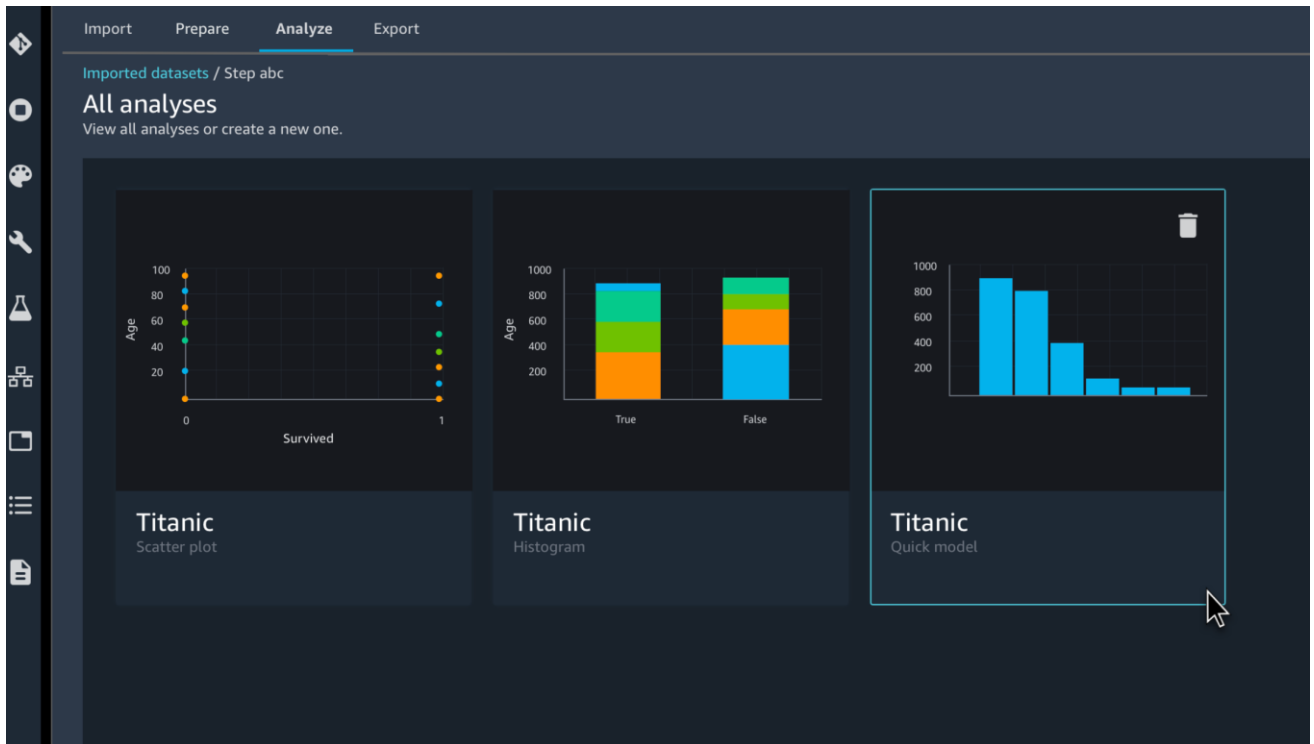
The services marked as 'New' in the above were mostly announced at re:invent in December 2021.

¹ <https://docs.aws.amazon.com/sagemaker/latest/dg/studio-ui.html>

4.1.1 Prepare

SageMaker Ground Truth is a labelling service, [similar to Google's](#)² but with more automation features and less reliance on humans. SageMaker Ground Truth's automation makes it competitive with [specialist labelling tools](#)³ (a [whole area in itself](#)⁴).

Data Wrangler allows data scientists to visualize, transform and analyze data from supported data sources from within SageMaker Studio:



² <https://cloud.google.com/vertex-ai/docs/datasets/data-labeling-job>

³ <https://neptune.ai/blog/data-labeling-software>

⁴ <https://anthony-sarkis.medium.com/the-5-best-ai-data-annotation-platforms-for-machine-learning-2021-ec17c15142f3#de98>

Data flow / S3-072320-campaignMovieSpecials.csv

S3-072320-campaignMovieSpecials.csv

String App String Category Decimal Rating Integer Reviews String Size String Installs

Photo Editor Coloring bookCamera &	ART_AND_DESIGN	4.1	159	19M	10,000+
Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+
U Launcher Lite	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+
Themes, Hide Apps	ART_AND_DESIGN	4.5	215644	25M	50,000,000+
Sketch - Draw & Paint	ART_AND_DESIGN	4.3	967	2.8M	100,000+
Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.4	167	5.6M	50,000+
Paper flowers instructions	ART_AND_DESIGN	3.8	178	19M	50,000+
Smoke Effect Photo Maker - Smoke	ART_AND_DESIGN	4.1	36815	29M	1,000,000+
Editor	ART_AND_DESIGN	4.4	13791	33M	1,000,000+
Infinite Painter	ART_AND_DESIGN	4.7	121	3.1M	10,000+
Garden Coloring Book	ART_AND_DESIGN	4.4	13880	28M	1,000,000+
Kids Paint Free - Drawing Fun	ART_AND_DESIGN	4.4	8788	12M	1,000,000+
Text on Photo - Fontee	ART_AND_DESIGN	4.2	44829	20M	10,000,000+
Name Art Photo Editor - Focus n Filters	ART_AND_DESIGN	4.6	4326	21M	100,000+
Tattoo Name On My Photo Editor	ART_AND_DESIGN	4.4	1518	37M	100,000+

Back to data flow

TRANSFORMS

Learn more about transforms.

Add Previous

Suggested transformers

- Find-Replace
- Rename column
- Replace rare
- Impute missing categorical
- Impute missing numeric
- Tokenizer
- TF-IDF text embedding
- Ordinal encode
- Onehot encode
- Drop column
- Duplicate column

Data Wrangler is also integrated with [Clarify](#)⁵ (which handles explainability), to highlight bias in data. This streamlines feature engineering and the resulting features can go directly to SageMaker Feature Store. Custom code can be added and SageMaker also separately has support for [Spark processing jobs](#)⁶.

Once features are in the Feature Store, they are available to be searched for and used by other teams. They can also be used at the serving stage as well as the training stage.

4.1.2 Build

The 'Build' heading is for offerings that save time throughout the whole process. [AutoPilot](#)⁷ is SageMaker's AutoML service that covers automated feature engineering, model building and selection. The various models it builds are all visible so you can evaluate them and choose which to deploy.

JumpStart is a set of CloudFormation templates for common ML use cases.

4.1.3 Train and Tune

Training with SageMaker is typically done from the python sdk, which is used to invoke training jobs. A training job runs inside a container on an EC2 instance. You can use a [pre-built docker image](#)⁸ if your training job is for a natively supported [algorithm](#)⁹ and framework. Otherwise you can use your [own docker image](#)¹⁰ that conforms to the

⁵ <https://aws.amazon.com/sagemaker/clarify/>

⁶ <https://docs.aws.amazon.com/sagemaker/latest/dg/processing-job.html>

⁷ <https://aws.amazon.com/sagemaker/autopilot/>

⁸ <https://docs.aws.amazon.com/sagemaker/latest/dg/docker-containers.html>

⁹ <https://docs.aws.amazon.com/sagemaker/latest/dg/algos.html>

¹⁰ <https://towardsdatascience.com/deploying-a-custom-docker-model-with-sagemaker-to-a-serverless-front-end-with-s3-8ee07edc24e6>

[requirements](#)¹¹. The SDK can also be used for [distributed training jobs](#)¹² for frameworks that support distributed training.

Processing and training steps can be chained together in [Pipelines](#)¹³. The resulting models can be [registered](#)¹⁴ in the model registry and you can [track lineage](#)¹⁵ on artifacts from steps. You can also [view and execute pipelines](#)¹⁶ from the SageMaker Studio UI.

4.1.4 Deploy and Manage

The SageMaker SDK has a 'deploy' operation for which you specify what type of instance you want your model deployed to. As with training, this can be either a custom image or a built-in one. The expectation is training and deployment will both happen with SageMaker but this [can be worked around](#)¹⁷ if you want to deploy a model that you've trained outside of SageMaker. Serving real-time HTTP requests is the typical case but you can also perform [batch](#)¹⁸ predictions and chain inference steps in [inference pipelines](#)¹⁹.

Deployed models get some monitoring by default with integration to CloudWatch for basic invocation metrics. You can also set up [scheduled monitoring jobs](#)²⁰. SageMaker can be configured to [capture request and response data](#)²¹ and to perform various comparisons on that data such as comparing against training data or triggering alerts based on constraints.

4.2 Azure Machine Learning

- [Workspaces](#)(see page 21)
- [Datasets](#)(see page 22)
- [Environments](#)(see page 22)
- [Experiments](#)(see page 23)
- [Pipelines](#)(see page 23)
- [Models](#)(see page 24)
- [Endpoints](#)(see page 24)

The Azure Machine Learning offering is consciously pitched at multiple roles (especially Data Scientists and Developers) and different skill levels. It is aimed to support team collaboration and automate the key problems of MLOps, across the whole ML lifecycle. This comes across in the prominence Azure gives to [workspaces](#)²² and [git repos](#)²³ and there's also increasing support for [Azure ML with VSCode](#)²⁴ (along with the web-based GUI called Studio).

11 <https://docs.aws.amazon.com/sagemaker/latest/dg/your-algorithms-training-algo-dockerfile.html>

12 https://sagemaker-examples.readthedocs.io/en/latest/training/distributed_training/pytorch/data_parallel/mnist/pytorch_smdataparallel_mnist_demo.html

13 <https://docs.aws.amazon.com/sagemaker/latest/dg/build-and-manage-steps.html>

14 <https://docs.aws.amazon.com/sagemaker/latest/dg/build-and-manage-steps.html#step-type-register-model>

15 <https://docs.aws.amazon.com/sagemaker/latest/dg/pipelines.html>

16 <https://docs.aws.amazon.com/sagemaker/latest/dg/pipelines-studio.html>

17 <https://towardsdatascience.com/deploying-a-custom-docker-model-with-sagemaker-to-a-serverless-front-end-with-s3-8ee07edc24e6>

18 <https://docs.aws.amazon.com/sagemaker/latest/dg/batch-transform.html#batch-transform-large-datasets>

19 <https://docs.aws.amazon.com/sagemaker/latest/dg/inference-pipelines.html>

20 <https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-scheduling.html>

21 <https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-data-capture.html>

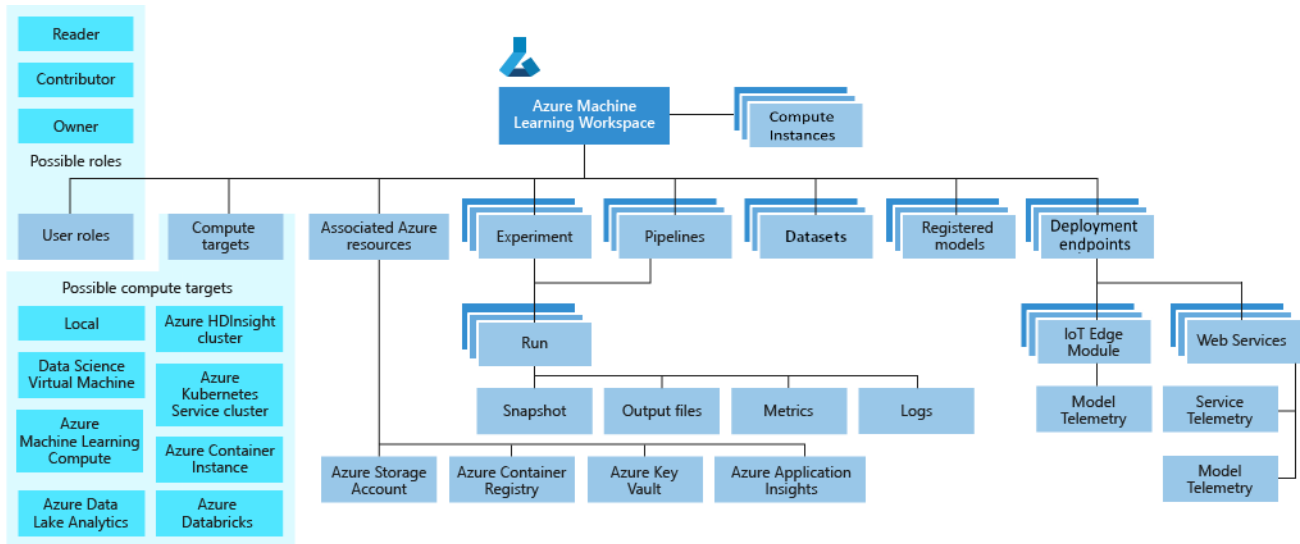
22 <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-manage-workspace?tabs=python>

23 <https://docs.microsoft.com/en-us/azure/machine-learning/concept-train-model-git-integration>

24 <https://visualstudiomagazine.com/articles/2021/05/05/vscode-azureml.aspx>

The cloud providers are all looking to leverage existing relationships in their MLOps offerings. For Microsoft this appears to be about developer relationships (with GitHub and VSCode) as well as a reputation as a compute provider. They seem keen on integrations and [references to open source tools](#)²⁵ and integrations to the Databricks platform are prominent in the documentation.

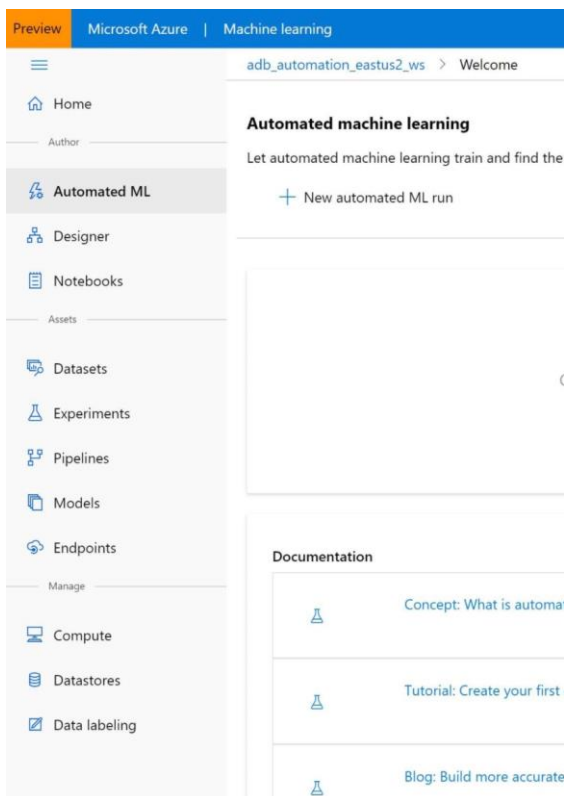
4.2.1 Workspaces



With Azure Machine Learning everything belongs to a workspace by default and workspaces can be shared between users and teams. The assets under a workspace are shown in the [studio web UI](#)²⁶.

²⁵ <https://docs.microsoft.com/en-us/azure/machine-learning/concept-open-source>

²⁶ <https://docs.microsoft.com/en-us/azure/machine-learning/overview-what-is-machine-learning-studio>



Let's walk through the key Azure ML concepts to get a feel for the platform.

4.2.2 Datasets

Datasets are references to where data is stored. The data itself isn't in the workspace but the dataset abstraction lets you work with the data through the workspace. Only metadata is copied to the workspace. Datasets can be [FileDataSets](#) or [TabularDataSets](#)²⁷. The data can be on a range of supported types of storage, including [blob storage](#), [databases](#) or [the Databricks file system](#)²⁸.

4.2.3 Environments

An environment is a configuration with variables and library dependencies, used for [training and for serving models](#)²⁹. Plays a similar role to pipenv but is instantiated through docker under the hood.

²⁷ <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-create-register-datasets#dataset-types>

²⁸ <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-access-data#supported-data-storage-service-types>

²⁹ <https://docs.microsoft.com/en-us/azure/machine-learning/concept-environments>

4.2.4 Experiments

Experiments are groups of training runs. Each time we train a model with a set of parameters, that falls under an experiment that can be automatically recorded. This allows us to review what was trained when and by whom.

Here's a [simple script](#)³⁰ to submit a training run with the Azure ML Python SDK:

```
from azureml.core import ScriptRunConfig, Experiment
from azureml.core.environment import Environment

exp = Experiment(name="myexp", workspace = ws)
# Instantiate environment
myenv = Environment(name="myenv")

# Configure the ScriptRunConfig and specify the environment
src = ScriptRunConfig(source_directory=".", script="train.py", compute_target="local", environment=myenv)

# Submit run
run = exp.submit(src)
```

Here we're referring to another script called "train.py" that contains typical model training code, [nothing azure-specific](#)³¹. We name the experiment that will be used and also name the environment. Both are instantiated automatically and the submit operation runs the training job.

The above is [run from the web studio](#)³² with the files in the cloud already. Training can also be run from a notebook or from local by having the CLI configured and submitting a [CLI command with a YAML specification](#)³³ for the environment image and to point to the code.

Training parameters and metrics can be automatically logged as [Azure integrates with mlflow's open source approach to tracking](#)³⁴. If you submit a run from a directory under git then [git information is also tracked for the run](#)³⁵.

4.2.5 Pipelines

Azure [Machine Learning Pipelines](#)³⁶ are for training jobs that have multiple long-running steps. The steps can be chained to run in different containers so that some can run in parallel and if an individual step fails then you can retry/resume from there.

³⁰ <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-use-environments#use-environments-for-training>

³¹ <https://docs.microsoft.com/en-us/azure/machine-learning/tutorial-1st-experiment-sdk-train>

³² <https://docs.microsoft.com/en-us/azure/machine-learning/tutorial-1st-experiment-hello-world>

³³ <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-train-cli>

³⁴ <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-train-cli>

³⁵ <https://docs.microsoft.com/en-us/azure/machine-learning/concept-train-model-git-integration>

³⁶ <https://docs.microsoft.com/en-us/azure/machine-learning/concept-ml-pipelines>

4.2.6 Models

[Models](#)³⁷ are either created in Azure through training runs or you can register models created elsewhere. A registered model can be deployed as an endpoint.

4.2.7 Endpoints

An endpoint sets up hosting so that you can make requests to your model in the cloud and get predictions. Endpoint hosting is inside a [container image](#)³⁸ - so basically an Environment, which could be the same image/Environment used for training. There are some [prebuilt images available](#)³⁹ to use as [a basis](#)⁴⁰. Or you can build an image from scratch that [conforms to the requirements](#)⁴¹.

[Azure ML's managed endpoints](#)⁴² have traffic-splitting [features](#)⁴³ for rollout and can work with GPUs. Inference can be real-time or batch. There's also integration to [monitoring](#)⁴⁴ features. Managed endpoints and monitoring are both in Preview/Beta release at the time of writing.

4.3 Databricks

- [Databricks Lakehouse Platform](#)(see page 24)
- [Data Science](#)(see page 25)
- [Machine Learning](#)(see page 25)
- [AutoML](#)(see page 26)

4.3.1 Databricks Lakehouse Platform

The Databricks Lakehouse Platform aims to bring data engineering, machine learning and analytics. One of its selling points is a unified architecture to try to break down silos and encourage collaboration. The platform contains areas that address particular needs, which they break down as:

- Delta Lake (flexible storage with built-in history)
- ETL
- Machine Learning
- Data Science
- Databricks SQL
- Security and Administration

[Delta Lake](#)⁴⁵ is the reason why lakehouse gets its name. It's a format based on Parquet that makes it possible to ACID transactions to a data lake in cloud object storage. This makes it suitable for streaming as well as batch

37 <https://docs.microsoft.com/en-us/azure/machine-learning/concept-azure-machine-learning-architecture#models>

38 <https://docs.microsoft.com/en-us/azure/machine-learning/concept-compute-target#deploy>

39 <https://docs.microsoft.com/en-us/azure/machine-learning/concept-prebuilt-docker-images-inference>

40 <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-extend-prebuilt-docker-image-inference>

41 <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-custom-container>

42 <https://docs.microsoft.com/en-us/azure/machine-learning/concept-endpoints>

43 <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-managed-online-endpoints>

44 <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-monitor-online-endpoints>

45 <https://databricks.com/product/data-science>

processing. Delta Lake can also enforce a schema, support schema modification and keep a transaction log. So we get a combination of the features of a data lake with those of a data warehouse - hence 'lakehouse'.

We will focus primarily on the [Data Science](#)⁴⁶ and [Machine Learning](#)⁴⁷ parts of the platform.

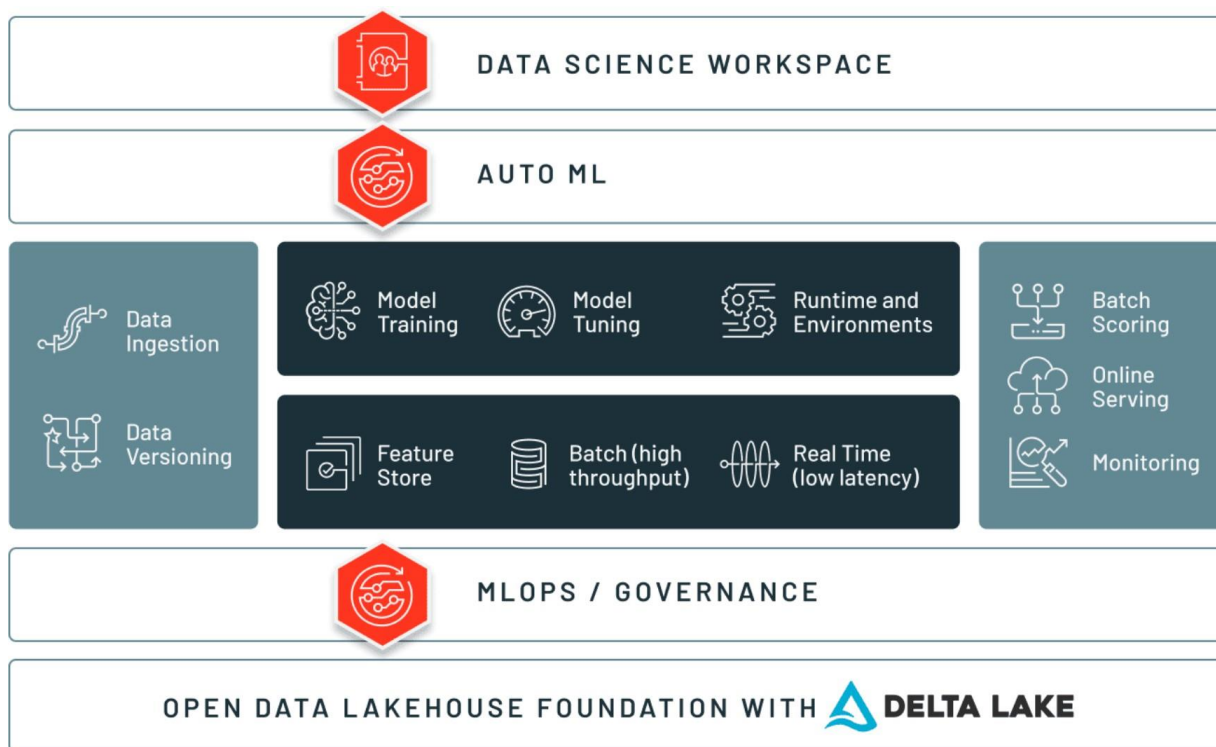
4.3.2 Data Science

Delta Lake is [relevant to data scientists](#)⁴⁸ insofar as it helps bring a schema to otherwise schemaless data. This helps to ensure data quality and reduce friction between the data engineering and data science stages.

Databricks notebooks support a range of languages and ways of working with data. We can perform Spark operations, SQL, use python, R and more, all in the same notebook.

Different types of data work will have different compute needs. Databricks supports this with different [types of clusters](#)⁴⁹, including requests for specialist underlying compute like GPUs.

4.3.3 Machine Learning



Prepared features can be stored in the [platform's feature store](#)⁵⁰ so that they can be discovered and shared with other teams.

⁴⁶ <https://databricks.com/product/data-science>

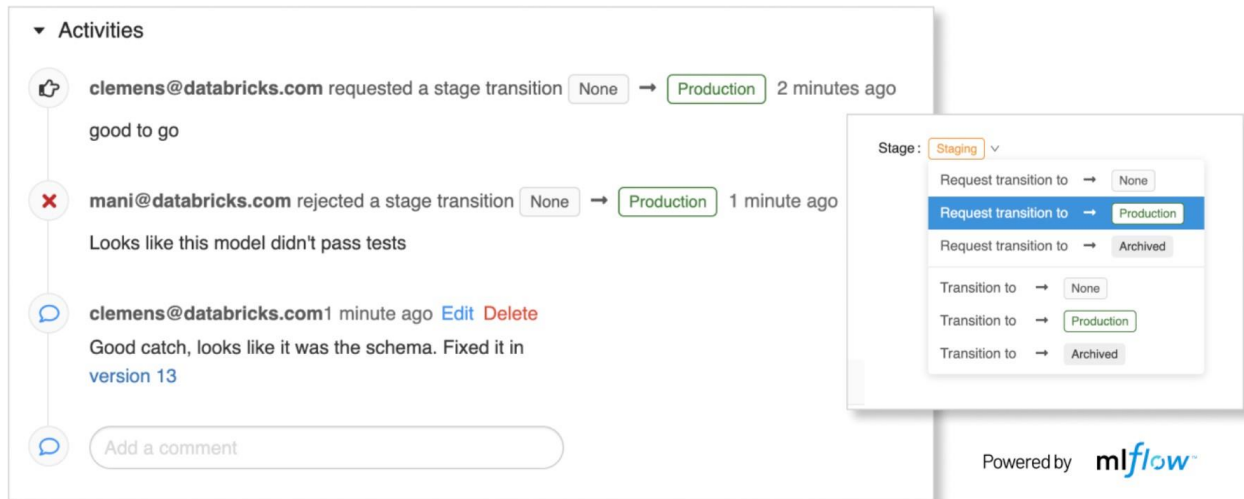
⁴⁷ <https://databricks.com/product/machine-learning>

⁴⁸ <https://databricks.com/discover/demos/machine-learning-with-mlflow>

⁴⁹ <https://docs.databricks.com/clusters/index.html>

⁵⁰ <https://docs.databricks.com/applications/machine-learning/preprocess-data/index.html>

Model training support is integrated with notebooks - [training runs are tracked with mlflow](#)⁵¹ and the training job history can be seen from a panel next to the notebook. Trained models can go to the [model registry](#)⁵², from which they can be transitioned between environments:



Models can be deployed for [real-time inference/serving](#)⁵³. [Batch inference is also supported](#)⁵⁴, either on models in a popular framework or using spark mllib. Spark also has support for streaming.

4.3.4 AutoML

Databricks offers [AutoML features](#)⁵⁵ within the context of notebooks. Python code is used to trial different algorithms and models for a dataset. In the background databricks creates notebooks specifically for each of the permutations that it trials, so that we can see what happened under the hood and leverage the code.

4.4 Dataiku

- [Data Exploration and Transformation](#)(see page 27)
- [Model Building and Deployment](#)(see page 28)

Its platform emphasises the visual, attempting to make data tasks faster and more accessible. Users are able to explore and clean data using Data Science Studio's simple visual interface and can also write code for transforming data. Pipelines can be built up visually and transformation steps ('recipes') can be pre-built drag-and-drop or crafted using a chosen language/tool. Rather than competing with Spark or Hadoop, Dataiku integrates with these and other tools. It also includes features for deployment, monitoring and collaboration.

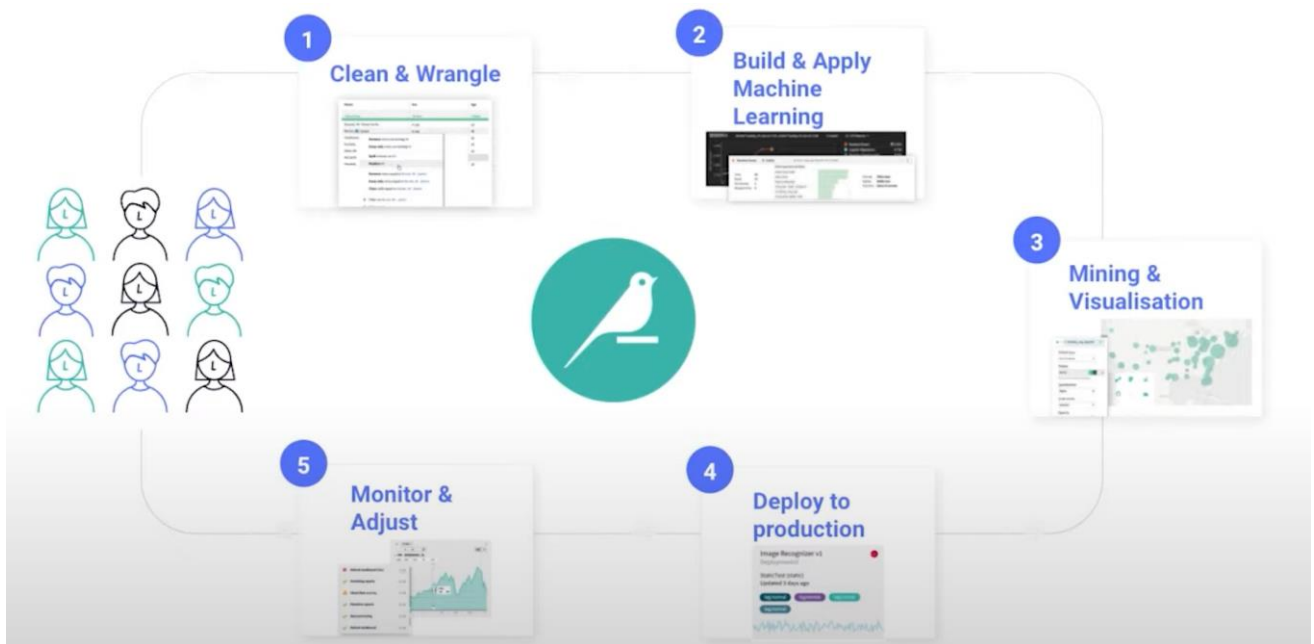
⁵¹ <https://docs.databricks.com/applications/machine-learning/train-model/ml-quickstart.html>

⁵² <https://docs.databricks.com/applications/machine-learning/manage-model-lifecycle/index.html>

⁵³ <https://docs.databricks.com/applications/machine-learning/model-deploy/index.html>

⁵⁴ <https://docs.databricks.com/applications/machine-learning/model-inference/index.html>

⁵⁵ <https://docs.databricks.com/applications/machine-learning/automl.html>



4.4.1 Data Exploration and Transformation

Work in Data Science Studio happens within a project. A project can contain datasets, which can be selected from a wide variety of sources including cloud blob storage, databases, hadoop, spreadsheets, sharepoint, salesforce and social media. This variety is because the platform is designed to be used by a wide range of users. Where possible Data Science Studio keeps the data in its original location and performs the processing at source (e.g. by integrating with Hadoop).

Once the dataset is in the project then it can be explored and manipulated. Data Science Studio infers the data types of columns and can highlight missing values or outliers. There's an easy, spreadsheet-like interface for applying operations or writing formulas to handle missing values, apply transformations or create new columns:

Yr...	ArrTime	ArrDelay	Cancelled	CancellationCode	Diverted	CRSElapsedTi...	ActualElapsedTime	AirTime	Distance	CarrierDelay	WeatherDel...	NASDelay	SecurityDelay
string	string	integer	boolean	string	boolean	integer	integer	integer	integer	integer	integer	integer	integer
10	1028	-7	0		0	160	155	142	1072				
11	1205	15	0		0	100	97	77	534				
17	1721	1	0		0	180	181	160	1041				
11	1213	43	0		0	130	135	118	719				
12	1244	-6	0		0	155	153	137	882				
11	1047	-23	0		0	200	177	165	1121				
15	1528	-12	0		0	110	100	79	591				
10	0956	-4	0		0	110	110	80	591				
19	1850	-30	0		0	115	91	79	591				
18	1809	-21	0		0	120	102	92	777				
0	0101	46	0		0	120	106	87	777	44	0	0	0
16	1621	11	0		0	105	85	74	570				
7	0716	-19	0		0	85	69	58	395				
10	0954	-21	0		0	90	71	58	395				
10	1004	-11	0		0	135	124	109	861				
1	0201	56	0		0	135	107	95	861	56	0	0	0
16	1624	-11	0		0	90	77	66	468				
14	1500	45	0		0	100	93	77	611	3	0	0	0
23	0007	32	0		0	70	55	42	307	32	0	0	0

There are many built-in transformations for different data types and they can be previewed against a small sample of data before applying to the whole set. This kind of interface with the ability to interface to large databases and hadoop has a clear appeal to business-focused data analysts (which is a [key focus for Dataiku](#)⁵⁶) and is designed to also save time for those who are [comfortable getting into the code](#)⁵⁷.

Within a Data Science Studio we create Flows that correspond to Pipelines in other platforms. Within a Flow we can embed Recipes that transform Datasets. These Recipes can be pre-built, constructed visually (e.g. visual join operations) or written using code:



4.4.2 Model Building and Deployment

Data Science Studio provides a [Lab for assisted building of models](#)⁵⁸. Here we can build models using configurable AutoML with many tunable parameters (or we can use the defaults). We can also obtain predictions, explain predictions and output reports.

The Lab is a separate area from a Flow so to build a model within the Flow it needs to be 'Deployed' into the Flow. This creates a training Recipe in the Flow and a Model as the output of the Recipe. The Flow could then have a step to generate predictions using the model (a Scoring step) or it could Deploy the model as an API. New runs will then [automatically replace the live version](#)⁵⁹ (with old versions kept as options to roll back to).

⁵⁶ <https://www.youtube.com/watch?v=MUwloqMJ8BQ>

⁵⁷ <https://www.youtube.com/watch?v=ryZRRijQ5Z8>

⁵⁸ <https://www.youtube.com/watch?v=cT4IRTNW9ns>

⁵⁹ https://www.youtube.com/watch?v=zWs_B_cVjtc

Models are not restricted to the AutoML features and can be [written with custom code](#)⁶⁰ or using [Spark MLLib](#)⁶¹ or [H2O's Sparkling Water](#)⁶². Deploying a model creates an API endpoint. [API endpoints](#)⁶³ can also be created for custom code functions or even SQL queries. Data Science Studio can also [host a frontend](#)⁶⁴ with tools for building our own interactive Webapp or we instead [publish a simple dashboard](#)⁶⁵.

[Metrics](#)⁶⁶ can be tracked for Datasets (such as number of records or records in a value range) or for Models (such as ROC AUC). Checks can then be created against these Metrics so that warnings are displayed if the Checks fail.

4.5 DataRobot

Quiet famous for its Automated ML solution: <https://www.datarobot.com/platform/automated-machine-learning/>

Multi-cloud compatible with end-to-end MLOps Stack: <https://www.datarobot.com/platform/mlops/>

ML stack for AI in telecommunications: <https://www.datarobot.com/solutions/telecom/>

(Network Operations: Energy management for base stations, RAN Upgrade planning, Capacity forecasting, Radio hole coverage etc)

4.6 Google Vertex AI Platform

- [Pipelines for Orchestration](#)(see page 29)
- [Revamped AutoML](#)(see page 31)
- [Training Models](#)(see page 32)
- [Deployment](#)(see page 33)
- [Monitoring](#)(see page 33)

Vertex is Google's [newly-unified](#)⁶⁷ AI platform. The main ways in which it is unified are:

- Everything falls logically under vertex headings in the google cloud console and the APIs to the services should be consistent. (Previously AutoML was [visibly a separate function](#)⁶⁸).
- Pipelines can be used as an orchestrator for most of the workflow ([including AutoML](#)⁶⁹).

4.6.1 Pipelines for Orchestration

This idea of pipelines as an orchestrator across offerings is illustrated here ([from TechCrunch](#)⁷⁰):

⁶⁰ <https://doc.dataiku.com/dss/latest/machine-learning/algorithms/in-memory-python.html#custom-models>

⁶¹ <https://doc.dataiku.com/dss/latest/machine-learning/algorithms/mllib.html>

⁶² <https://doc.dataiku.com/dss/latest/machine-learning/algorithms/sparkling-water.html>

⁶³ <https://doc.dataiku.com/dss/8.0/apinode/endpoints.html>

⁶⁴ <https://doc.dataiku.com/dss/8.0/webapps/index.html#introduction-to-dss-webapps>

⁶⁵ <https://doc.dataiku.com/dss/latest/dashboards/index.html>

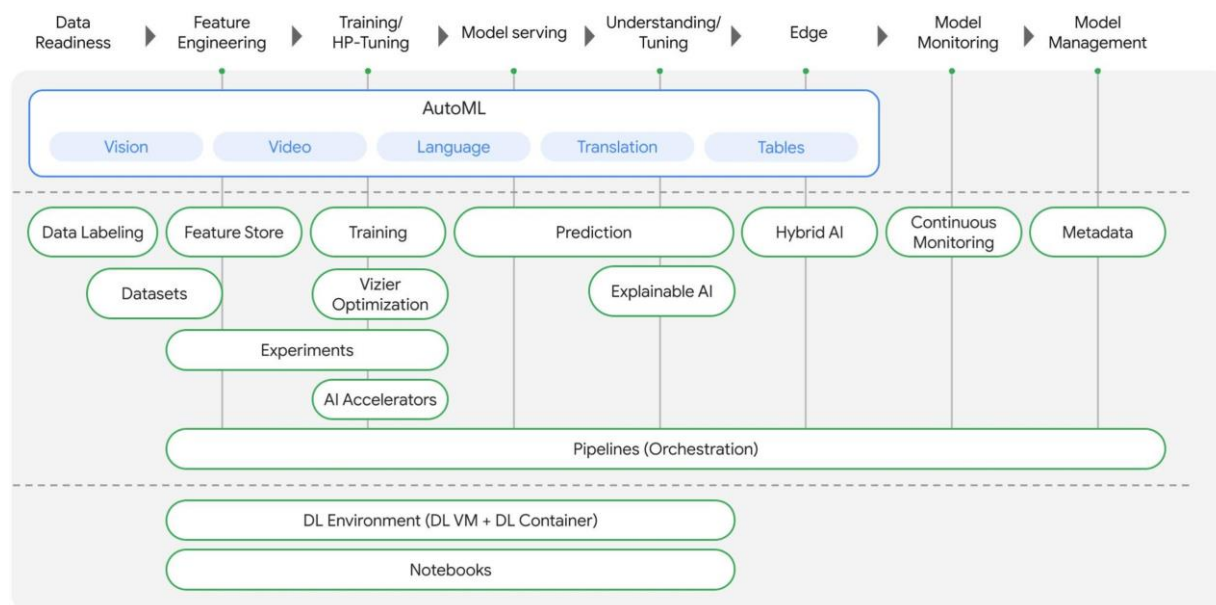
⁶⁶ <https://knowledge.dataiku.com/latest/courses/automation/metrics-checks-hands-on.html>

⁶⁷ <https://techcrunch.com/2021/05/18/google-cloud-launches-vertex-a-new-managed-machine-learning-platform/>

⁶⁸ <https://fuzzylabs.ai/blog/vertex-ai-the-hype/>

⁶⁹ <https://cloud.google.com/blog/topics/developers-practitioners/use-vertex-pipelines-build-automl-classification-end-end-workflow>

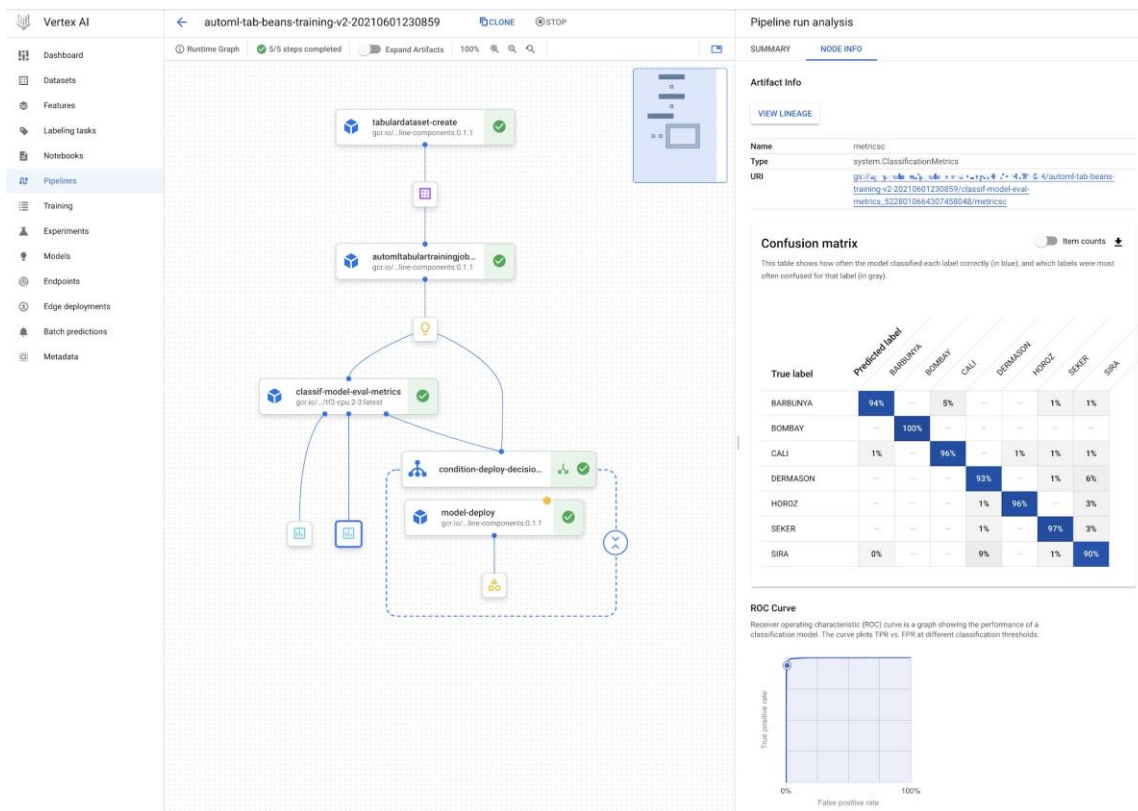
⁷⁰ <https://techcrunch.com/2021/05/18/google-cloud-launches-vertex-a-new-managed-machine-learning-platform/>



This could be confusing to those familiar with kubeflow pipelines (which is [what vertex pipelines are under the hood](https://cloud.google.com/vertex-ai/docs/pipelines/build-pipeline)⁷¹) as kubeflow pipelines started out as a distributed training system, with each step executing in a separate container, along with a UI to inspect runs and ways to resume from a failed step. Pipelines are usable for distributed training but pipelines can also be used to perform other tasks beyond training. This is illustrated in the below [screenshot](https://cloud.google.com/blog/topics/developers-practitioners/use-vertex-pipelines-build-automl-classification-end-end-workflow)⁷²:

⁷¹ <https://cloud.google.com/vertex-ai/docs/pipelines/build-pipeline>

⁷² <https://cloud.google.com/blog/topics/developers-practitioners/use-vertex-pipelines-build-automl-classification-end-end-workflow>

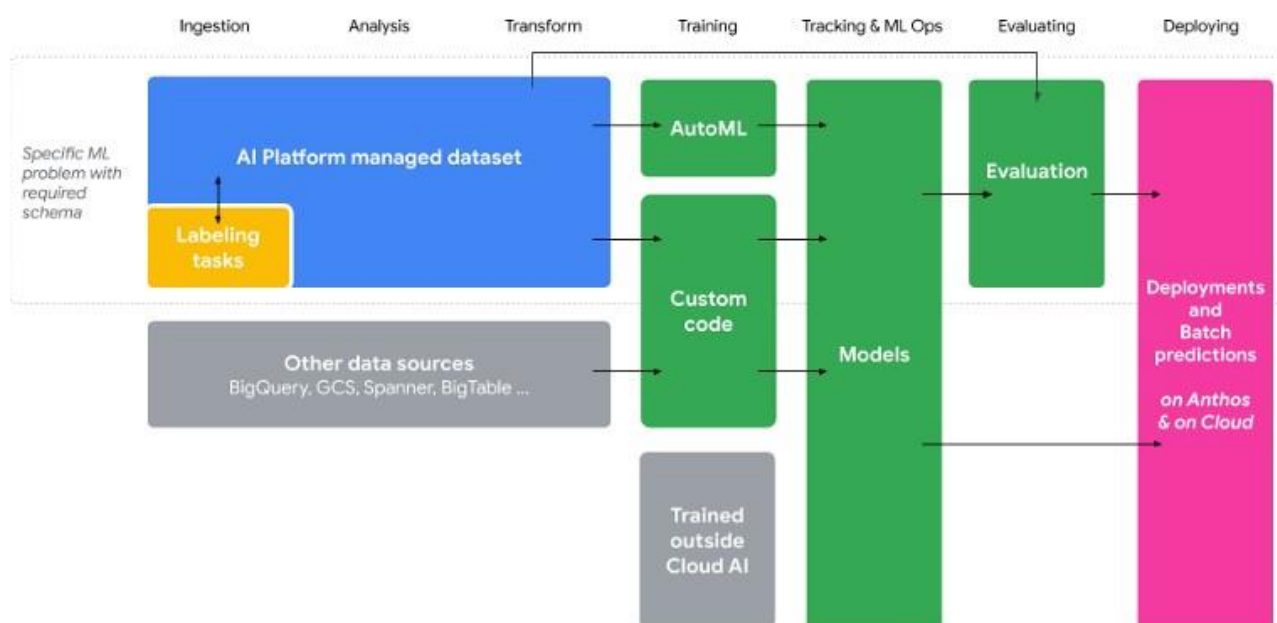


Here there is a conditional deployment decision to decide whether the model is good enough to deploy or not. If it passes the test then the model is deployed from the pipeline.

4.6.2 Revamped AutoML

The AutoML offerings are now more consistent with other parts of Google's AI stack. Basically different ways to input your data can lead to the same training path and different training paths can lead to the same deployment path. Here's a diagram from Henry Tappen and Brian Kobashikawa ([via Lak Lakshmanan](https://towardsdatascience.com/giving-vertex-ai-the-new-unified-ml-platform-on-google-cloud-a-spin-35e0f3852f25)⁷³):

⁷³ <https://towardsdatascience.com/giving-vertex-ai-the-new-unified-ml-platform-on-google-cloud-a-spin-35e0f3852f25>



There is also some difference, as can be seen in how datasets are handled. The dataset concept is broken into managed (which has specific metadata and lives on [specific google data products](#)⁷⁴) or not managed. The managed datasets [are mostly for just AutoML for now](#)⁷⁵.

4.6.3 Training Models

We could think of Google as having three basic routes to training models - AutoML, pipelines (which is intended more as an orchestrator) and [custom training jobs](#)⁷⁶. Where pipelines use multiple containers with each step in a different container, custom training jobs are a single-container training function. It has automatic [integration to TensorBoard](#)⁷⁷ for results. It can do distributed training via the underlying ML code framework, [providing the framework supports it](#)⁷⁸. It supports GPUs and you can [watch/inspect runs](#)⁷⁹, though inspecting runs looks a bit basic compared to pipelines.

There's a facility [native to the custom training jobs for tuning](#)⁸⁰ hyperparameters. In addition to this, google has launched Vizier. Instead of being native to training jobs Vizier has an API and you tell it what you've tried and it makes suggestions for what to try next. This may be less integrated but Vizier is [able to go deeper](#)⁸¹ in what it can tune.

⁷⁴ <https://towardsdatascience.com/giving-vertex-ai-the-new-unified-ml-platform-on-google-cloud-a-spin-35e0f3852f25>

⁷⁵ <https://cloud.google.com/vertex-ai/docs/training/code-requirements>

⁷⁶ <https://cloud.google.com/vertex-ai/docs/training/code-requirements>

⁷⁷ <https://cloud.google.com/vertex-ai/docs/experiments/tensorboard-training>

⁷⁸ <https://cloud.google.com/vertex-ai/docs/training/distributed-training>

⁷⁹ <https://cloud.google.com/vertex-ai/docs/training/monitor-debug-interactive-shell>

⁸⁰ https://cloud.google.com/vertex-ai/docs/vizier/overview#how_differs_from_custom_training

⁸¹ https://cloud.google.com/vertex-ai/docs/vizier/overview#how_differs_from_custom_training

4.6.4 Deployment

To deploy your models to get predictions, you have two options. If your model is built using a [natively supported framework](#)⁸², you can tell google to load your model (serialized artifact) into a pre-built container image. If your model framework is not supported or you have custom logic then you can supply a [custom image meeting the specification](#)⁸³. Google can then [create an endpoint](#)⁸⁴ for you to call to get predictions. You can create this either through the API or using the web console wizard.

4.6.5 Monitoring

Running models can be monitored for training/serving skew. For skew you supply your training set when you [create the monitoring job](#)⁸⁵. The monitoring job stores data going into your model in a storage bucket and uses this for comparison to the training data. You can get alerts based on configurable thresholds for individual features. Drift is similar but [doesn't require training data](#)⁸⁶ (it's monitoring change over time). Feature [distributions are shown in the console](#)⁸⁷ for any alerts.

4.7 H2O.ai

- [H2o-3](#)(see page 33)
- [Sparkling Water](#)(see page 35)
- [Steam](#)(see page 35)
- [Driverless AI](#)(see page 35)
- [MLOps](#)(see page 36)
- [Wave](#)(see page 36)
- [Hybrid Cloud Platform](#)(see page 37)
- [Additional Points](#)(see page 37)

[H2o.ai](#)⁸⁸ provides a range of products. We'll go over the key ones and how they relate to each other. Let's start with the open source product 'h2o', which is the oldest one and shares its name with the company. The latest version is called h2o-3 (which also helps separate the product from the company names).

4.7.1 H2o-3

H2o-3 comes with a UI called Flow. Within Flow you get a notebook-like environment for working with data and building models. Cells can be written with code or you can use a wizard-like interface to choose steps like [loading a file or imputing missing values](#)⁸⁹. Models can be trained [using built-in algorithms](#)⁹⁰.

82 <https://cloud.google.com/vertex-ai/docs/predictions/pre-built-containers>

83 <https://cloud.google.com/vertex-ai/docs/predictions/custom-container-requirements>

84 <https://cloud.google.com/vertex-ai/docs/predictions/deploy-model-console#custom-trained>

85 https://colab.research.google.com/github/GoogleCloudPlatform/vertex-ai-samples/blob/master/notebooks/official/model_monitoring/model_monitoring.ipynb#scrollTo=XV-vru2Pm1oX

86 <https://cloud.google.com/vertex-ai/docs/model-monitoring/using-model-monitoring?hl=nb#analyzing-skew-drift>

87 <https://cloud.google.com/vertex-ai/docs/model-monitoring/using-model-monitoring?hl=nb#analyzing-skew-drift>

88 <http://H2o.ai>

89 <https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-munging.html>

90 <https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science.html>

H2O FLOW

Flow Demo

95	71	1	1	1	13.4	44.2	7
96	1	75	1	1	16	18.7	7
97	70	1	3	1	11.200000000000001		7
98	58	1	1	1	7		6
99	64	1	1	1	29.1		6
100	1	66	1	3	9.5	28.1	7

CLIPS

Click on a clip to insert it into notebook. Click the button and execute the clip.

System (8)

- CS assist
- CS importFiles
- CS getFrames
- CS getModels
- CS getPredictions
- CS getJobs
- CS buildModel
- CS predict

buildModel

Build a Model

Select an algorithm

- (Algorithm)
- kmeans
- glm
- word2vec
- gbm**
- deeplearning

The execution behind all this is an in-memory compute engine running in the Java Virtual Machine but we [can work with](#)⁹¹ Python or R and can interface to Hadoop or Spark.

Models can be [exported](#)⁹² to run in other Java environments (e.g. [a Spring Boot app](#)⁹³) or [outside of Java with the provided runtime libraries](#)⁹⁴.

⁹¹ <https://docs.h2o.ai/h2o/latest-stable/h2o-docs/quick-start-videos.html#quick-start-videos>

⁹² <https://towardsdatascience.com/getting-started-with-h2o-using-flow-b560b5d969b8>

⁹³ <https://aws.amazon.com/blogs/machine-learning/training-and-serving-h2o-models-using-amazon-sagemaker/>

⁹⁴ <https://www.h2o.ai/products/h2o-driverless-ai/mojo-deployment-options/>



4.7.2 Sparkling Water

This is [h2o-3 but on top of Spark](#)⁹⁵. So we can [run h2o algorithms on Spark](#)⁹⁶.

4.7.3 Steam

Steam is for [managing h2o jobs on top of Spark or Kubernetes](#)⁹⁷.

4.7.4 Driverless AI

Where h2o-3 gives us assistance in building models, Driverless AI is an alternative approach that is full-on AutoML. With h2o-3 we pick the algorithm but with Driverless AI a range of algorithms get tried out for us to pick the best. We start with data representing historical observations with outcomes.

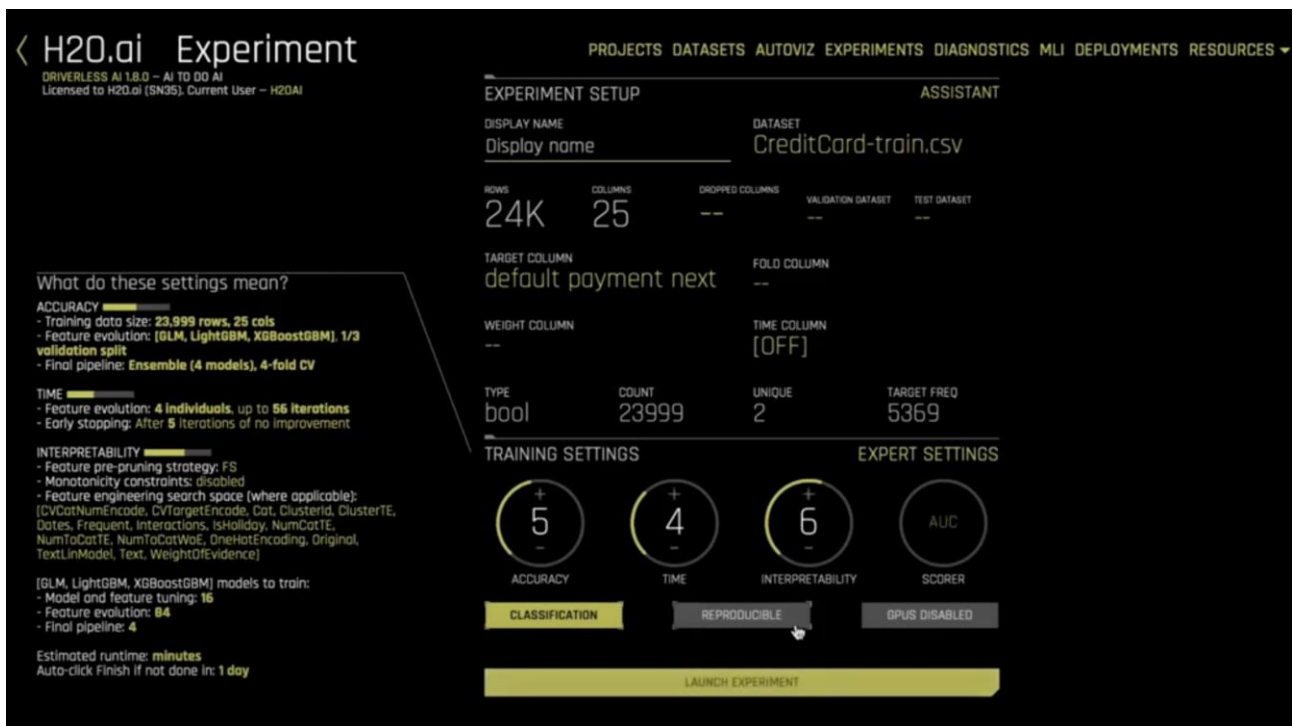
We connect Driverless AI to our data (it supports a range of sources) and it then lets us explore the data and visualize column stats/correlations.

It does test-train split and lets us set some options before running experiments. It automatically evaluates a range of different algorithms and compositions of features:

⁹⁵ <https://docs.h2o.ai/#sparkling-water>

⁹⁶ <https://docs.h2o.ai/sparkling-water/3.1/latest-stable/doc/about.html>

⁹⁷ <https://www.h2o.ai/enterprise-support/#enterprise-security>



Driverless AI takes the approach of offering defaults and letting more advanced users drill into the details of how the experiments are conducted.

It then provides insights into the model created and which features are most important for which predictions (explanations). Models can be exported to be run in a range of environments, much like h2o-3 models.

4.7.5 MLOps

[H2o MLOps](#)⁹⁸ is a model deployment and monitoring suite that runs on kubernetes. It handles models built with h2o-3 or h2o Driverless AI or we can Bring Your Own Model (provided we [specify a detailed schema for it](#)⁹⁹).

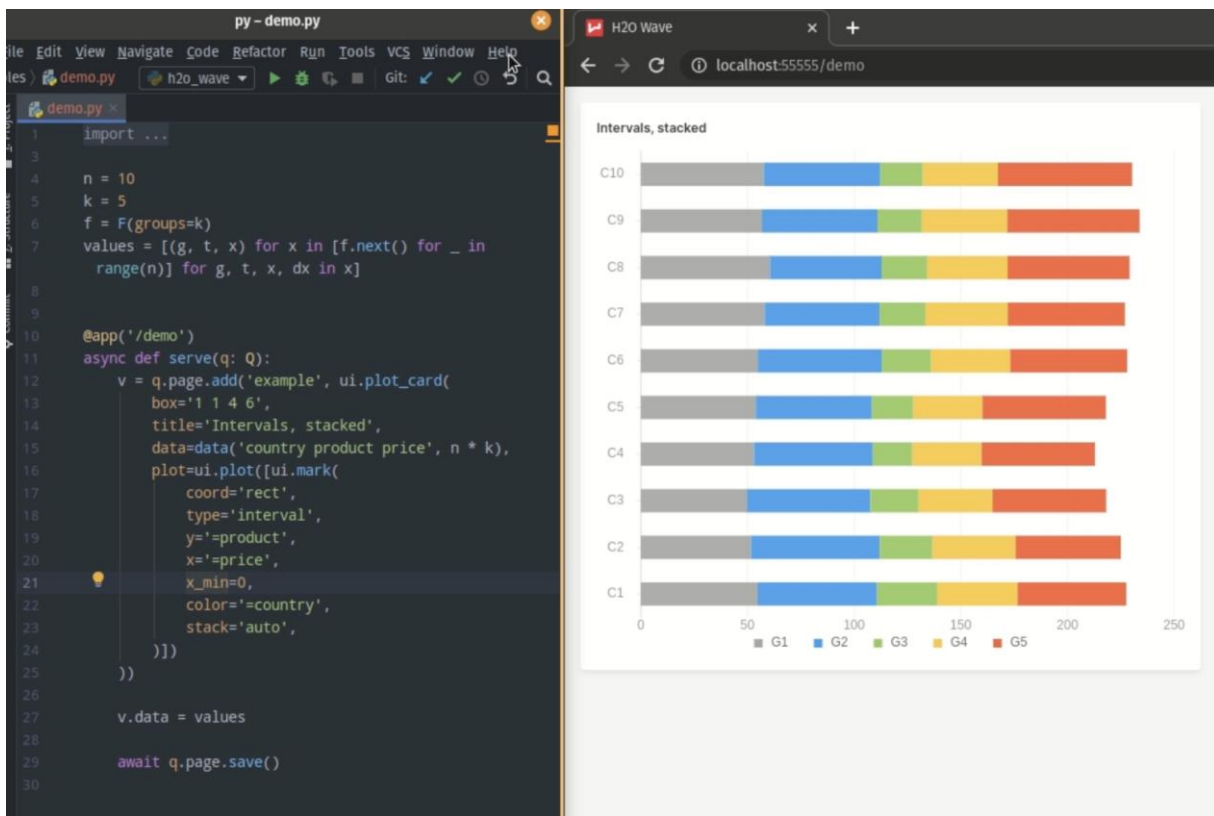
4.7.6 Wave

[Wave](#)¹⁰⁰ is for developing ML-enabled and data-focused applications in python. Developers use python to write the UI as well as the backend and the data-handling. This makes it easy to embed graphs and charts.

⁹⁸ <https://docs.h2o.ai/mlops-release/latest-stable/docs/userguide/index.html>

⁹⁹ <https://docs.h2o.ai/mlops-release/latest-stable/docs/userguide/byom.html#mlops-byom>

¹⁰⁰ <https://www.h2o.ai/products/h2o-wave/>



4.7.7 Hybrid Cloud Platform

H2o Hybrid Cloud Platform¹⁰¹ brings together other h2o products to provide an integrated setup for developing, deploying, hosting and sharing applications. It runs on kubernetes so it can be installed on public, private or hybrid clouds.

The public face of the platform is the **Appstore**¹⁰². This shows a set of tiles for published apps built with Wave. The platform allows apps built with Wave to be operationalised and published to the Appstore. **Driverless AI and h2o-3 projects can also be leveraged**¹⁰³. The Driverless AI projects are automatically available in the MLOps screens in the platform. Deployed models can be leveraged by apps built with Wave.

Deployed models can be monitored using the features from the MLOps product. Steam is also integrated for managing h2o jobs.

4.7.8 Additional Points

Quiet famous for its Automated ML solution: <https://www.h2o.ai/products/h2o-driverless-ai/>

Multi-cloud compatible with end-to-end MLOps Stack: <https://www.h2o.ai/resources/product-brief/h2o-mlops/>

Claim to have ML stack for AI in telecommunications: <https://www.h2o.ai/telecom/>

¹⁰¹ <https://docs.h2o.ai/h2o-ai-cloud/index.html>

¹⁰² <https://docs.h2o.ai/h2o-ai-cloud/docs/userguide/basic-concepts>

¹⁰³ <https://www.h2o.ai/hybrid-cloud/request-demo/>

4.8 KNIME

- [KNIME Analytics Platform](#)(see page 38)
 - [Understanding Workflows](#)(see page 38)
 - [Nodes and Ports](#)(see page 40)
 - [Do We Really Not Need Code?](#)(see page 40)
 - [Visualizations](#)(see page 40)
 - [Local vs Hosted Working and File Handling](#)(see page 41)
- [KNIME Server](#)(see page 41)
 - [Deploying Workflows with KNIME Server](#)(see page 41)
 - [Monitoring](#)(see page 42)
 - [Web Portal](#)(see page 43)
 - [Collaboration Features](#)(see page 43)

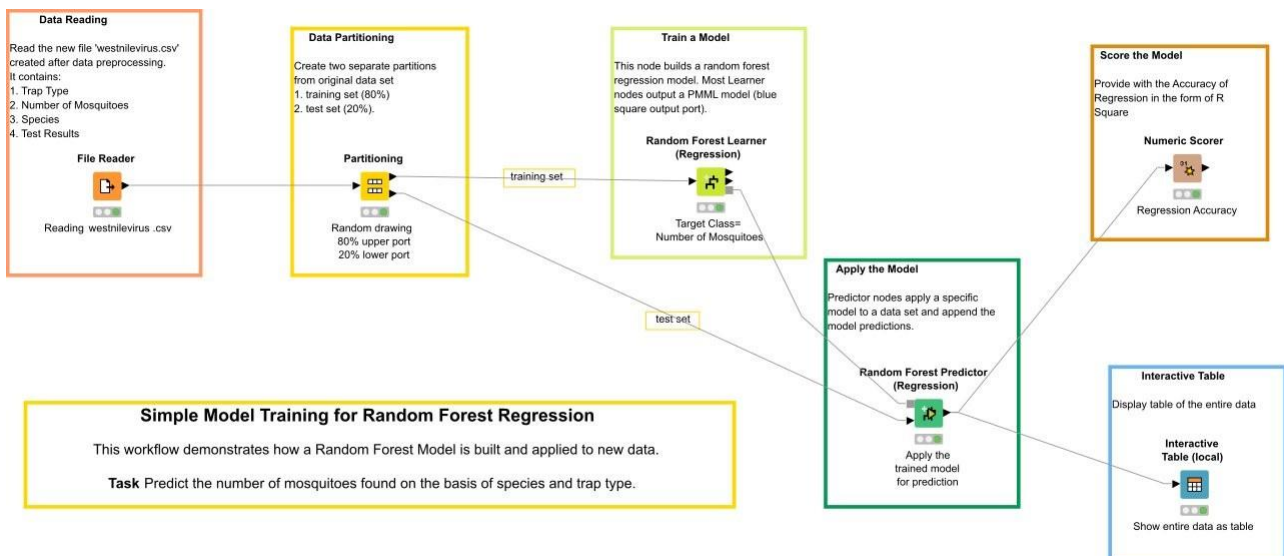
KNIME Analytics Platform is a free, open source visual development environment for data science. The Platform is used to build workflows which can then be deployed for production usage with KNIME Server, which is a paid enterprise offering. The KNIME approach to data science is unusual when compared to other MLOps platforms. It is better to think of KNIME as a platform for data science and analytics and to think of its MLOps capabilities as complementing its approach to data science.

4.8.1 KNIME Analytics Platform

Understanding Workflows

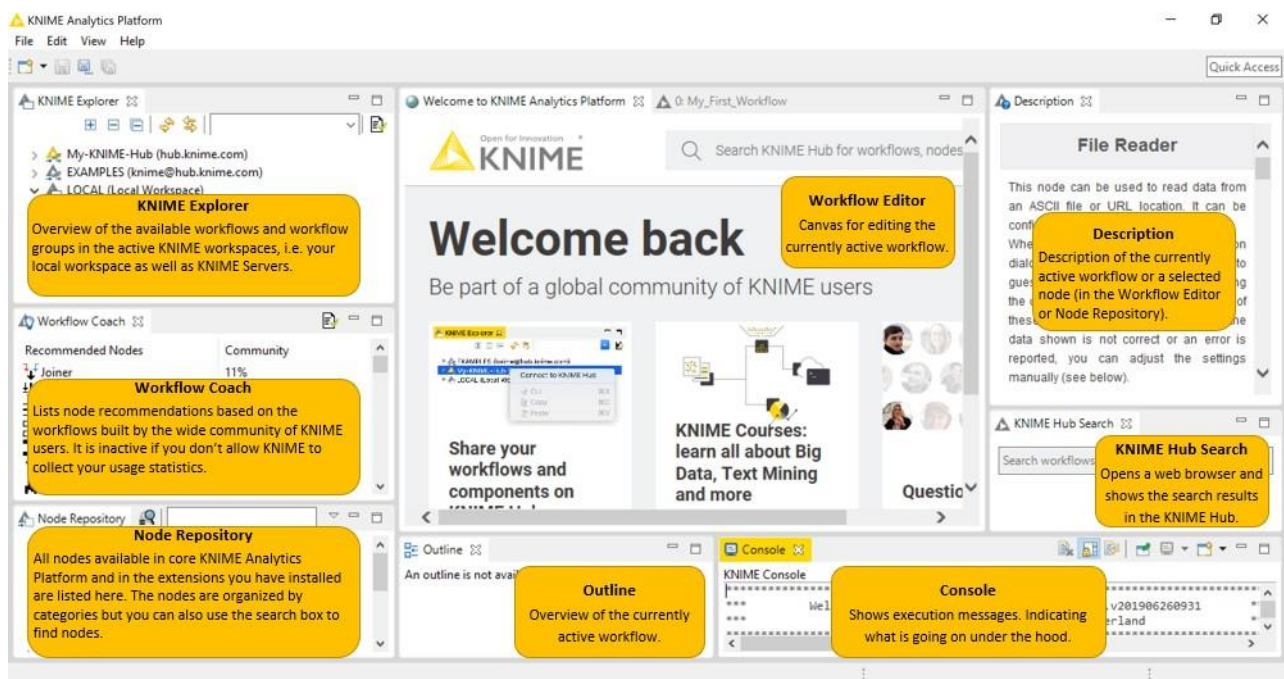
KNIME Analytics Platform is a codeless environment for building data science, data analysis and data manipulation workflows. Let's think about why one might want to take a low-code approach to data science. Given that we essentially have a sequence of operations in a workflow, KNIME handles it visually. Each operation can be a node. If the platform supports the operations, then we can configure the operations in the nodes in the platform and not deal with code or the nuances of libraries. The above code is very much like [this KNIME workflow](#)¹⁰⁴:

¹⁰⁴ <https://hub.knime.com/raksharawat/spaces/Public/latest/project/9.%20Random%20Forest%20Regression~6dkpKR7RShJyNV9d>



Workflows generally have lots of text and boxes on them. These are annotations - they're not functional. They're comments. It's just the icon-like boxes that capture operations - these are the nodes.

We can compose workflows by dragging and dropping nodes onto a canvas from the node repository. We then configure exactly what each node does. The configuration is a pop-up for each node and may not show in the diagram but is part of the saved workflow. In the above we start with a File Reader node that loads a CSV file, the data set is then passed on a Partitioning node and the training set goes on to train a Random Forest model. we draw and configure these workflows in the [KNIME Analytics Platform workbench](https://docs.knime.com/2019-06/analytics_platform_workbench_guide/index.html#the-knime-workbench)¹⁰⁵, which is a desktop application:



¹⁰⁵ https://docs.knime.com/2019-06/analytics_platform_workbench_guide/index.html#the-knime-workbench

Workflows live in the top-left. There's a canvas in the middle and nodes can be dropped onto the canvas from the bottom-left. The console at the bottom right is for debugging as workflows can also be executed from within the workbench.

Let's now understand more about how KNIME workflows work.

Nodes and Ports

Nodes are connected through ports. The most common ports are data ports [handling tabular data](#)¹⁰⁶. Many reader nodes like the CSV reader output data as data tables. As we expect, this is a type that contains columns and rows and columns can be of the [usual types](#)¹⁰⁷. Nodes that output tables typically write them to disk and there's a KNIME file format for data tables ([table](#)¹⁰⁸). Output can also be seen in the workbench. The workbench makes it easy to explore data has integrations to help see how the data flows through the workflow It's not only tabular data that flows between nodes. An output could be an image or a ML model (which can be written as [PMML](#)¹⁰⁹ but doesn't have to be written out beyond being a node in the workflow). There are also flow variables, which might be reference values used in calculations.

Do We Really Not Need Code?

There are many, many nodes available and the community contributes to keep this growing. There are [integrations available to frameworks such as keras](#)¹¹⁰ so that we can compose a keras-based deep learning architecture that we can see visually and without having to write python code.

We can dive into code in KNIME with script nodes. For example, there's a [python script node](#)¹¹¹ that can receive input data as pandas data frames. There are also integrations for [Java](#)¹¹² and [R](#)¹¹³. There's also an option to code up our [own nodes](#)¹¹⁴ and components ([which bundle the functionality of multiple nodes](#)¹¹⁵). For steps that use dependencies, there's a [conda environment propagation node](#)¹¹⁶, which avoids having to preinstall the right libraries before we can run a workflow.

Visualizations

KNIME workflows are not limited to processing of data - they can also feature points of display and interaction. Some [nodes perform plots](#)¹¹⁷ and graphs that are displayed to the user in the workbench when the workflow is run.

¹⁰⁶ https://docs.knime.com/2018-12/analytics_platform_workbench_guide/index.html#data-table

¹⁰⁷ https://docs.knime.com/2018-12/analytics_platform_workbench_guide/index.html#data-table

¹⁰⁸ https://hub.knime.com/knime/spaces/Examples/latest/01_Data_Access/01_Common_Type_Files/06_Table_Reader~ua4RudFCVylfMOzF

¹⁰⁹ <https://forum.knime.com/t/exporting-machine-learning-models/20831>

¹¹⁰ <https://www.knime.com/deeplearning/keras>

¹¹¹ <https://hub.knime.com/knime/extensions/org.knime.features.python2/latest/org.knime.python2.nodes.script2.Python2ScriptNodeFactory2>

¹¹² https://hub.knime.com/knime/spaces/Examples/latest/07_Scripting/01_Java/01_Example_of_Java_Snippet~L6cZ5ZEx93PIU7G

¹¹³ https://hub.knime.com/knime/spaces/Examples/latest/07_Scripting/02_R/01_Example_of_R_Snippet~lc8PFcTAmnIC2rsE

¹¹⁴ <https://www.knime.com/first-steps-on-how-to-create-your-own-node>

¹¹⁵ https://docs.knime.com/2020-07/analytics_platform_components_guide/index.html#introduction

¹¹⁶ <https://www.knime.com/blog/how-to-manage-python-environments-conda-and-knime>

¹¹⁷ https://hub.knime.com/knime/spaces/Examples/latest/03_Visualization/02_JavaScript/12_Bivariate_Visual_Exploration_with_Scatter_Plot~by7OzxDGdOREyi_0

we can also combine javascript-based nodes to make components [involving multiple related plots](#)¹¹⁸ of the same data.

Components offering views of data can become hosted web pages with [KNIME Server's WebPortal](#)¹¹⁹. Then they can be used outside of KNIME's workbench. We'll understand this better when we cover KNIME Server.

Local vs Hosted Working and File Handling

Everything we've discussed about reading and writing files so far has probably suggested files loaded from paths on the local machine. KNIME was originally designed for a local machine working environment and significant changes were made to [support more cloud-hosted files](#)¹²⁰. Now there's a concept of a [dynamic port](#)¹²¹, which is basically a data port that is decoupled from the file system. These allow KNIME to work with files in different types of file systems and file hosting, including cloud storage buckets. Files can also be parameters so that they're specified by the user on execution.

4.8.2 KNIME Server

KNIME server can be thought of as a hosting platform for workflows. KNIME Server can run on a different machine from installs of the Analytics Platform. Each member of a team would have their own install of the Analytics Platform but there might only be one KNIME Server. The Server has [executors/workers to do the work of executing workflows](#)¹²². Executors can be run in elastic cloud-based infrastructure or on premise.

In addition to running workflows, KNIME Server also hosts workflow artefacts. So it can be used as a kind of repository as well as a hosting platform.

Deploying Workflows with KNIME Server

What gets deployed to KNIME server is not simply a model. It is a workflow. To understand this, let's consider an example workflow:

This example is drawn from Jim Falgout's contribution to the whitepaper '[MLOps: Continuous Delivery for Machine Learning on AWS](#)¹²³'. We source data from S3, preprocess that data, train a model and use the withheld test data to make predictions. We then get a visual of performance on the test data.

When deploying this, we probably don't want that visual of performance on the test data. We also don't want to go through the training steps each time that a REST call is made to get a prediction. But we may want to perform some of the same pre-processing steps that we did during training, since if we had to do that on the training data then we'll likely have to do it on the live data too.

For this KNIME lets we specify parts of the workflow that should be used in the KNIME server workflow. Here for example is the workflow we discussed previously, now with parts specified for the deployment workflow:

118 https://docs.knime.com/2020-07/analytics_platform_components_guide/index.html#img-JSON-format-view

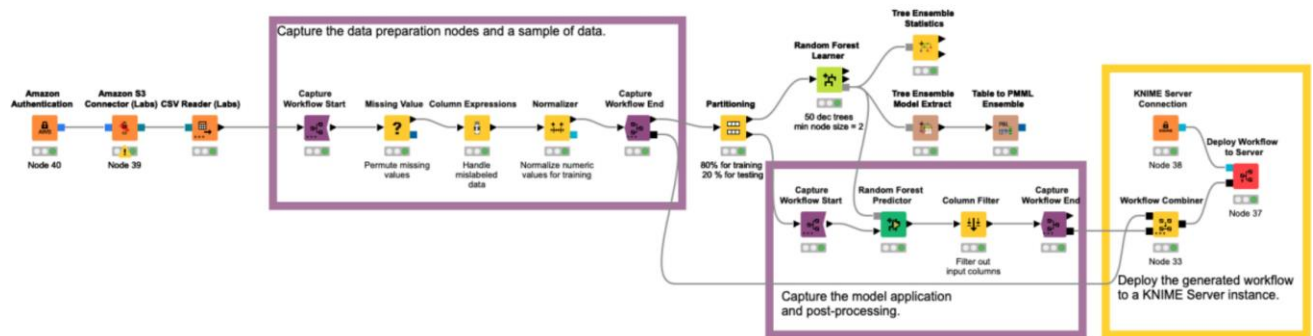
119 <https://www.knime.com/knime-software/knime-webportal>

120 https://docs.knime.com/latest/analytics_platform_file_handling_guide/index.html#introduction

121 https://docs.knime.com/latest/analytics_platform_file_handling_guide/index.html#old-new-nodes-distinguish

122 <https://www.knime.com/elastic-hybrid-execution>

123 <https://d1.awsstatic.com/whitepapers/ml-ops-continuous-delivery-machine-learning-on-aws.pdf>



The purple 'Capture Workflow Start' and 'Capture Workflow End' nodes show the start and end of sections to be captured. These parts of the workflow are saved and stitched together in the background and the resulting workflow is [deployed to KNIME server using the step at the end](#)¹²⁴.

Notice this is the amalgamation of the two purple 'Capture Workflow' sections above. The beginning and end of the workflow are container nodes. These are placeholders for input and output, which in live execution will be via REST. So a REST call's payload will go through the transformations in yellow before going into the model and then via the filter before the output is returned via the output container node. Note that in the previous workflow there was a random forest learner feeding into the random forest predictor. Here simply go straight into the predictor - effectively we've already trained the model and it is simply being used to make predictions in the deployed workflow.

The deployment step does not have to be invoked from within a KNIME workflow in the Analytics Platform. It's also possible to export the generated workflow above e.g. to an S3 bucket. This write could trigger a CI pipeline that deploys to KNIME Server. There could be tests run in KNIME server to [validate the workflow for production](#)¹²⁵.

Monitoring

we may not simply want to deploy the model without checking that it performs as well as the model it is taking over from. And we may want to monitor performance over time and trigger retraining if performance drops. These performance-based rollouts and on-going monitoring activities supported in KNIME with the [KNIME Model Process Factory](#)¹²⁶. The idea is basically to add orchestration workflows which can call out to other workflows. The Process Factory contains templates for this - for details the reader is best to consult the [official KNIME white paper on the subject](#)¹²⁷.

If we're looking to check model performance for drift or retraining, there's a [built-in model monitoring node for classifiers](#)¹²⁸.

There's also monitoring at the platform/executor level, as well monitoring that we build for individual models. KNIME Server can send notification emails for [job success or failure](#)¹²⁹ (KNIME Server can execute scheduled workflows as well as hosting for REST calls) and has a management portal where the [status of running workflows can be inspected](#)¹³⁰.

¹²⁴ <https://www.knime.com/integrated-deployment>

¹²⁵ <https://forum.knime.com/t/how-do-you-use-knime-testing-framework-on-knime-server-no-readmes-out-there-yet/33096/6>

¹²⁶ <https://www.knime.com/blog/the-knime-model-process-factory>

¹²⁷ https://www.knime.com/sites/default/files/inline-images/Model_Process_Management_20170404_1.pdf

¹²⁸ <https://www.knime.com/blog/model-monitoring-in-data-science-context>

¹²⁹ <https://www.knime.com/knime-server-course/chapter2>

¹³⁰ https://youtu.be/NuEhV7TXh1Y?list=PLZ3mQ6OITi0YGAcCMxua_sDWB9Q3quZox&t=481

Web Portal

We've seen that we can deploy workflows and make parts of workflows available for REST calls. we can also deploy workflows that contain javascript nodes. This feature of KNIME Server is called [Web Portal](#)¹³¹. It lets we deploy interactive reports as web applications so that users can interact with these reports without needing KNIME Analytics Platform.

So we've seen that nodes can specify interactive reports based on data and that data inputs can be parameterised so that they come from the user when running. Given that this interaction around workflows can be applied to the serving of predictions from models, could it also be applied to building models themselves? This idea is the basis of what KNIME calls '[Guided Analytics](#)'¹³². we can embed points of interaction in workflows so that users can choose what data to upload in order to retrain the model. This allows for something a bit like AutoML in KNIME. we can deploy workflows where users interactively upload data and the workflow builds models and the user chooses which is best. It is called 'Guided Analytics' as the approach is applicable to lots of data analysis and reporting workflows that might not involve any machine learning.

Collaboration Features

KNIME Server can host workflows and groups of workflows. we can think of this as a folder structure for collaboration like a shared drive. KNIME Analytics Platform can be connected to a KNIME server and then the hosted workflows and workflow groups become visible from its explorer, provided we [have permission](#)¹³³. we can also then choose to deploy workflows to that KNIME Server instance.

The interface in the explorer allows for exploring workflows like a shared drive but KNIME Server's hosting is as much like git as a shared drive. It has a version history and the ability to [compare versions](#)¹³⁴ as well as permissions with [owners and delegation of rights](#)¹³⁵.

4.9 Kubeflow

- [Kubeflow Overview](#)(see page 44)
- [Components](#)(see page 45)
 - [Kubeflow UI \(Central Dashboard\)](#)(see page 45)
 - [Pipelines](#)(see page 45)
 - [Metadata](#)(see page 46)
 - [Notebooks](#)(see page 47)
 - [Serving](#)(see page 47)
 - [Katib](#)(see page 47)
 - [Training Operators](#)(see page 47)
 - [Other Components](#)(see page 47)
- [Distributions](#)(see page 48)

When kubeflow was first launched it was designed to be [composable, scalable and portable](#)¹³⁶:

- Composable: we can pick and choose which parts to use.

¹³¹ <https://www.knime.com/knime-software/knime-webportal>

¹³² <https://www.knime.com/blog/principles-of-guided-analytics>

¹³³ <https://www.knime.com/knime-server-course/chapter1>

¹³⁴ <https://www.knime.com/knime-server-course/chapter3>

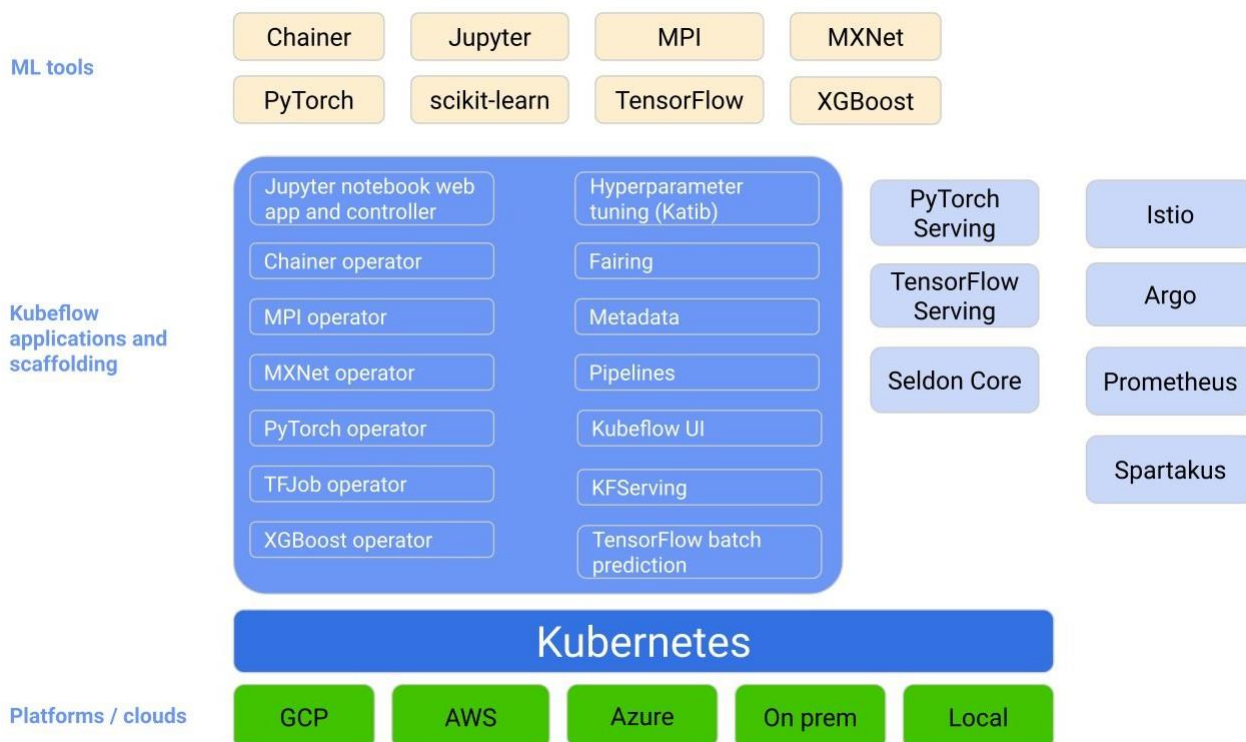
¹³⁵ <https://www.knime.com/knime-server-course/chapter1>

¹³⁶ <https://kubernetes.io/blog/2017/12/introducing-kubeflow-composable/>

- Scalable: we projects can use more or less resources as needed and leverage kubernetes features for scalability.
- Portable: we can use it on different clouds and infrastructure (including on-prem and hybrid cloud).

Kubeflow is designed for kubernetes so it aims to make the benefits of kubernetes available to machine learning. ML workloads with kubeflow run in containers and kubernetes will manage keeping these containers up. Kubernetes itself is cloud-agnostic so kubeflow gets portability from this. Kubeflow has further portability because there are different [distributions](#)¹³⁷ of kubeflow for integrating to different clouds. These different distributions leverage kubeflow's composability by plugging in cloud-specific features in place of some of the kubeflow vanilla ones.

4.9.1 Kubeflow Overview



The yellow/orange boxes on the top are supported ML tools (scikit-learn, PyTorch, Jupyter etc.). The darker blue and green boxes at the bottom represent the infrastructure with kubernetes as the base platform and deployment available for different cloud providers or on-prem or local with minikube (there's also some local interactions with an SDK).

The darker blue box on the middle left shows components. A lot of these are training operators - they may take up a lot of space but they're not the most important part of kubeflow. When we discuss the components we'll explain these but we'll put most emphasis on the kubeflow UI, pipelines, metadata and KFServing. The lighter boxes on the far right are external components - some parts that kubeflow relies on but doesn't expose to the user (argo, istio) and others that can be optionally used with kubeflow. We'll see though that there's not a hard line between [optional add-ons](#)¹³⁸ and core components.

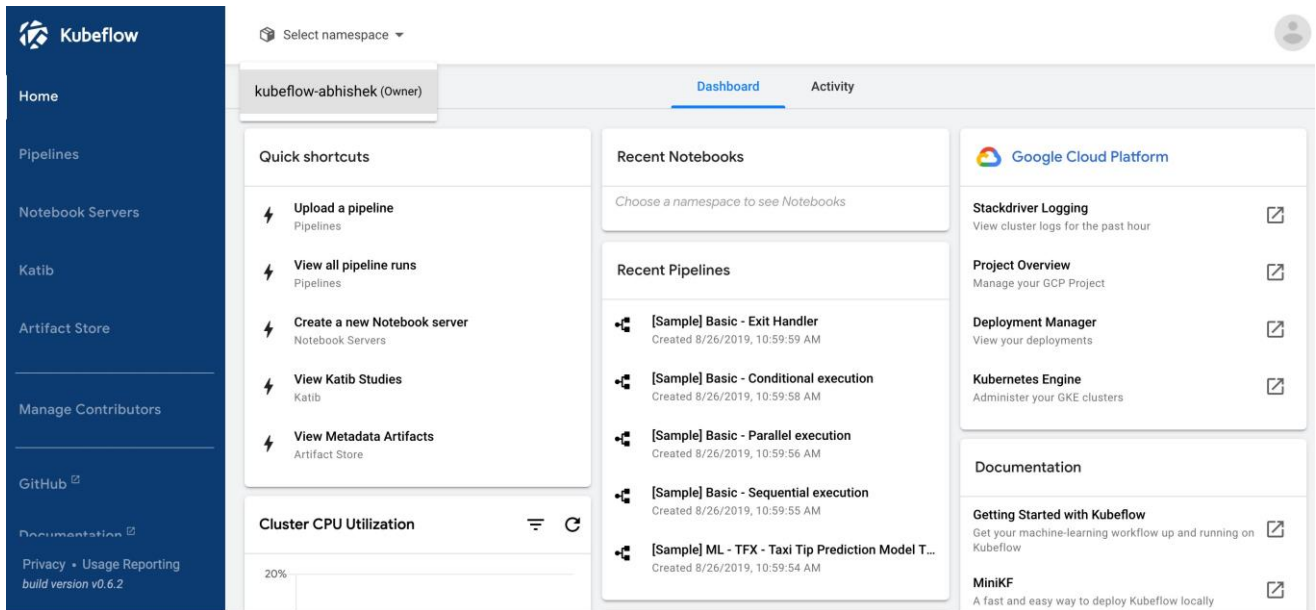
¹³⁷ <https://www.kubeflow.org/docs/distributions/>

¹³⁸ <https://www.kubeflow.org/docs/external-add-ons/>

4.9.2 Components

Kubeflow UI (Central Dashboard)

Before we can get into the kubeflow UI, we [first register a user](#)¹³⁹. Every user will have one or more profiles and with its primary profile it will own a kubernetes namespace. This provides a grouping for notebooks and other kubeflow resources and the kubernetes workloads underlying them (kubernetes resources can be restricted at a namespace level). Once into the UI, we can select namespaces that we have access to:



The UI provides a route to navigate between key resources such as notebooks and pipelines. The exact sections that appear in the UI is [customizable](#)¹⁴⁰ and some parts will appear different for different [distributions](#)¹⁴¹ of kubeflow.

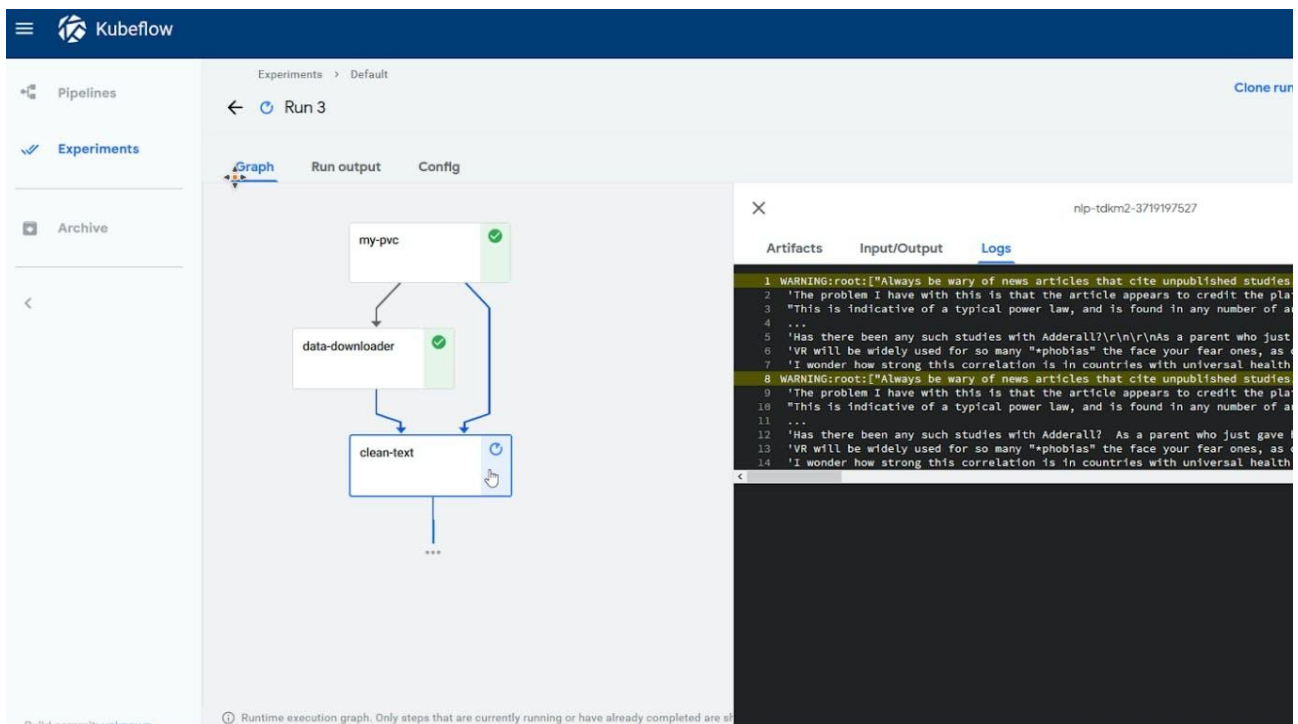
Pipelines

Kubeflow pipelines provide a way to run ML pre-processing, training and deployment steps within a continuous workflow. Each step can be a separate container that will be scheduled by kubernetes when its resource requirements can be met. Long-running jobs can be executed with access to the resources they need with steps running in parallel and using conditional logic. Multiple pipelines can also be executed in parallel with different values for their parameters.

¹³⁹ <https://www.kubeflow.org/docs/components/multi-tenancy/getting-started/>

¹⁴⁰ <https://www.kubeflow.org/docs/components/central-dash/customizing-menu/>

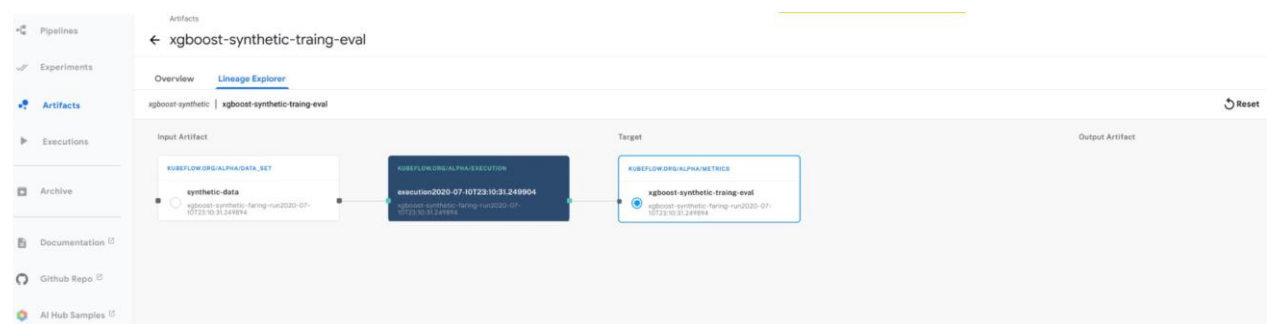
¹⁴¹ <https://www.kubeflow.org/docs/distributions/>



Each step in a pipeline is implemented as an operation in python code using the Kubeflow Pipelines DSL. Components can be written to be reusable across different pipeline implementations. The finished pipeline python code is compiled to kubernetes tarball of kubernetes yaml by the pipelines sdk (specifically kubernetes yaml that uses the argo orchestration tool). The tarball is then uploaded to the pipelines server component running in the cluster and from there runs can be scheduled by specifying different parameters for the pipelines.

Metadata

Kubeflow [metadata](#)¹⁴² provides an SDK for logging metadata about artifacts, executions and datasets. This is then logged in the kubeflow setup this is logged in the kubeflow metadata store and can be seen under the 'Artifacts' [tab](#)¹⁴³ in the UI:



There are different distributions for kubeflow to integrate to cloud-specific features. On the Azure distribution the metadata storage can be [replaced with an Azure mysql database](#)¹⁴⁴ and [for AWS an amazon mysql can be used](#)¹⁴⁵.

¹⁴² <https://blog.kubeflow.org/jupyter/2020/10/01/lineage.html>

¹⁴³ <https://blog.kubeflow.org/jupyter/2020/10/01/lineage.html>

¹⁴⁴ <https://www.kubeflow.org/docs/distributions/azure/azuremysql/>

¹⁴⁵ <https://www.kubeflow.org/docs/distributions/aws/customizing-aws/rds/>

Notebooks

There's a hosted [jupyter notebook](#)¹⁴⁶ installation in kubeflow. These notebooks are hosted in kubernetes and can be [shared between users](#)¹⁴⁷. Code in notebooks also has easy access to other kubeflow components such as metadata, training or serving.

Serving

The primary serving component in kubeflow is now [KFServing](#)¹⁴⁸ (but [seldon is packaged in many distributions](#)¹⁴⁹ and [other serving add-ons are available](#)¹⁵⁰). It is designed for use on top of a bundled knative, which allows it to scale up and down the amount of resource allocated to deployments based on the traffic they receive. There's [an integrated UI](#)¹⁵¹ to see which models are being served and some basic monitoring.

Katib

[Katib](#)¹⁵² saves data scientists time and effort by searching different sets of parameters and recording the results to see which results in the best models. It is most used for hyperparameter tuning and supports a range of AutoML algorithms aimed at data scientists.

Training Operators

A [training operator](#)¹⁵³ is a kubernetes custom resource that can be submitted to the cluster and it will run a hosted training job to create a machine learning model. Jobs can be split across multiple containers for distributed training. There are specific operators per ML framework. Kubeflow pipelines can also be used to train models but pipelines as a tool is at the orchestration level and can include steps other than training (including data prep and deployment). A kubeflow pipeline could use a training operator as a step. Or it could train models without using a training operator. we may also choose to train a model with an operator and not use a pipeline.

Other Components

Kubeflow has some other notable components that are more on the fringes. [Fairing](#)¹⁵⁴, for example, makes it easy to write just a small amount of code to train a model and also have it deployed to the kubernetes cluster. This seems to have become a less popular feature over time. The [FEAST feature store](#)¹⁵⁵ can be installed with kubeflow but the integration is not yet very deep (the best integration right now is [with KFServing](#)¹⁵⁶). There's also [Kale](#)¹⁵⁷, which can translate jupyter notebooks into kubeflow pipelines.

¹⁴⁶ <https://www.kubeflow.org/docs/components/notebooks/why-use-jupyter-notebook/>

¹⁴⁷ <https://www.kubeflow.org/docs/components/multi-tenancy/getting-started/#managing-contributors-through-the-kubeflow-ui>

¹⁴⁸ <https://www.kubeflow.org/docs/components/kfserving/kfserving/>

¹⁴⁹ <https://github.com/kubeflow/manifests/search?q=seldon>

¹⁵⁰ <https://www.kubeflow.org/docs/external-add-ons/serving/>

¹⁵¹ <https://www.kubeflow.org/docs/components/kfserving/webapp/>

¹⁵² <https://www.kubeflow.org/docs/components/katib/overview/>

¹⁵³ <https://www.kubeflow.org/docs/components/training/>

¹⁵⁴ <https://www.kubeflow.org/docs/external-add-ons/fairing/fairing-overview/>

¹⁵⁵ <https://www.kubeflow.org/docs/external-add-ons/feature-store/getting-started/>

¹⁵⁶ <https://github.com/kubeflow/kfserving/tree/master/docs/samples/v1beta1/transformer/feast>

¹⁵⁷ <https://www.kubeflow.org/docs/external-add-ons/kale/>

4.9.3 Distributions

Kubeflow is available as a vanilla install to just use the native features of kubernetes. But the [preferred route is distributions](#)¹⁵⁸, which are specific to clouds and providers. These distributions hook into the functionality of specific hosting environments. For example, the AWS distribution can hook into other AWS functionality. This can open up further integrations and reduce infrastructure management time.

4.9.4 Kubeflow Vs MLFlow

- [Criteria](#)(see page 48)
 - [Tracking and Versioning](#)(see page 48)
 - [Pipeline Orchestration](#)(see page 48)
 - [Model Deployment](#)(see page 48)
 - [Model Monitoring](#)(see page 48)
 - [Traditional Machine Learning](#)(see page 49)
 - [Deep Learning](#)(see page 49)
 - [Exploration](#)(see page 49)
 - [Productization](#)(see page 49)
 - [Citizen Data Scientist](#)(see page 49)
 - [Expert Data Scientist](#)(see page 49)
 - [Specialized Approach](#)(see page 49)
 - [End-to-end-Approach](#)(see page 49)
- [Comparison](#)(see page 49)
- [Summary](#)(see page 51)

Criteria

Tracking and Versioning

While executing the machine learning code, there's an API and UI for logging parameters, code versions, metrics, and output files so you can visualize them later.

Pipeline Orchestration

Pipelines allow us to build and manage multistep machine learning workflows run in Docker containers.

Model Deployment

Capability to deploy and serve models as part of the pipelines.

Model Monitoring

Capability to monitor deployed models on various performance parameters and also detect model decay and drift using external addons.

¹⁵⁸ <https://www.kubeflow.org/docs/started/installing-kubeflow/>

Traditional Machine Learning

Products that focus on traditional machine learning are built for structured data (SQL, Excel, etc.) and efficient processing through, for example, Spark. These MLOps platforms may also offer additional capabilities for data analysis and data manipulation in visual tools.

Deep Learning

Products for deep learning are built to handle massive amounts of unstructured data such as images, videos, or audio. Significant emphasis is placed on utilizing powerful GPU machines as these models may take days to train even on the most powerful hardware.

Exploration

Exploration focused platforms emphasize data analytics, experiment tracking, and working in notebooks.

Productization

Productization focused platforms primarily concentrate on machine learning pipelines, automation, and model deployment.

Citizen Data Scientist

Citizen data scientists are more subject matter experts rather than technical experts. Some MLOps platforms are focused on this category of users as they offer capabilities for teams with less engineering expertise to build and deploy machine learning models.

Expert Data Scientist

Some platforms focused on expert data scientists with engineering expertise or teams that contain significant data science and engineering expertise. These platforms tend to avoid proprietary code as much as possible and focus on supporting as many existing languages and frameworks as possible.

Specialized Approach

Focus of a specialized area such as Tracking and Versioning, Model Registry etc.

End-to-end-Approach

User should be able to automatically train, evaluate, and deploy a model in a single platform.

Comparison

	Track ing and Versio ning	Pipeli ne Orche strati on	Mod el Depl oyment	Mod el Moni toring	Traditi onal Machi ne Learn ing	Dee p Learn ing	Exp lora tion	Prod uctiz ation	Citize n Data Scien tist	Exper t Data Scien tist	Speci alized Appro ach	End- to- end- Appro ach
--	---------------------------------------	---------------------------------------	-----------------------------	-----------------------------	--	--------------------------	---------------------	------------------------	--------------------------------------	----------------------------------	---------------------------------	-------------------------------------

ML Flow	✓		✓		✓		✓		✓		✓	
Kubeflow	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓

	MLFlow	Kubeflow
MLOps Approach	MLflow is a Python program for tracking experiments and versioning model. With MLflow, the actual training happens where we choose to run it, and the MLflow service merely listens in on parameters and metrics.	Kubeflow is, at its core, a container orchestration system. When we train a model in Kubeflow, everything happens within the system (or within the Kubernetes infrastructure it orchestrates).
Collaborative environment	Experiment tracking is at the core of MLflow. It favors the ability to develop locally and track runs in a remote archive via a logging process. This is suitable for exploration.	The same capability is made possible through Kubeflow metadata. However, it requires higher technical know-how.
Pipelines and scalability	This capability is not available in MLFlow.	Orchestrating both parallel and sequential jobs is what Kubeflow was originally built for. For use cases where we may be running end-to-end ML pipelines or large-scale hyperparameter optimization, and we need to utilize cloud computing, Kubeflow is the choice of the two.
Model deployment	MLflow achieves this by utilizing the model registry. Through this, MLflow provides organizations with a central location to share machine learning models as well as a space for collaboration on how to move them forward for implementation and approval in the real world. The MLflow model registry has a set of APIs and UIs to manage the complete lifecycle of the MLflow model more collaboratively. The registry also provides model versioning, model lineage, annotations, and stage transitions.	In Kubeflow, this is achieved through Kubeflow pipelines, a distinct component that focuses on model deployment and continuous integration and delivery (CI/CD). Kubeflow pipelines may be used, independent of the rest of Kubeflow's capabilities. Kubeflow, allows for a collection of serving components on top of a Kubernetes cluster. This may be at the expense of a more amount of development effort and time.

Operational Aspects	<p>MLflow is often more favored among data scientists. MLflow is easier to set up since it is just a single service, and it's also easier to adapt thr ML experiments to MLflow as the tracking is done via a simple import in the code.</p>	<p>Kubeflow is often characterized as an complex tool, but most of the added complexity is due to the infrastructure orchestration capabilities (which require an understanding of Kubernetes).</p> <p>Kubeflow ensures reproducibility to a greater extent than MLflow because it manages the orchestration.</p>
----------------------------	--	---

Summary

Kubeflow solves infrastructure orchestration and experiment tracking with the added cost of being rather demanding to set up and maintain, while MLflow just solves experiment tracking (and model versioning).

Kubeflow meets the requirements of larger teams responsible for delivering custom production ML solutions. These teams often have more specialized roles and have the required resources to manage the Kubernetes infrastructure.

MLflow, on the other hand, more meets the needs of data scientists looking to organize themselves better around experiments and machine learning models. For such teams, ease of use and setup is often the main driver.

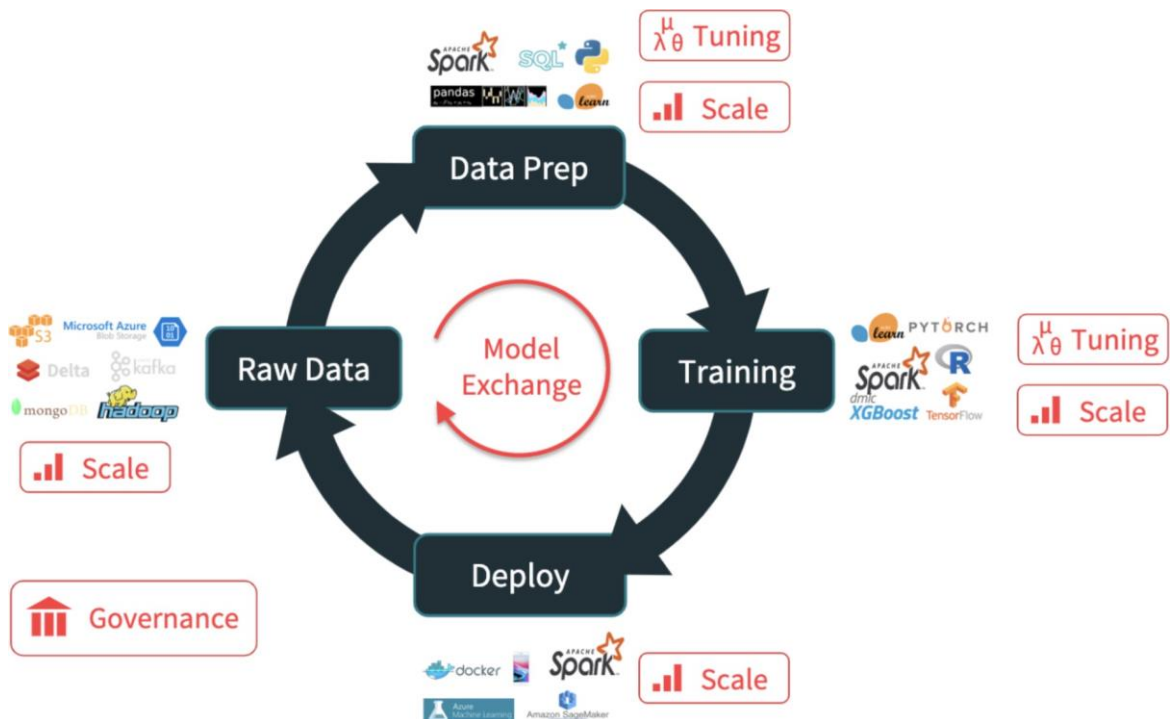
4.10 MLFlow

- [Tracking](#)(see page 53)
- [Projects](#)(see page 54)
- [Models](#)(see page 55)
- [Model Registry](#)(see page 56)

MLflow takes tracking, reproducibility and portability as its key problems to solve. Rather than provide a hosted solution with components to address each concern, mlflow instead caters to the local experience and the range of options users might choose for hosting. Rather than bake a set of supported libraries and algorithms into the flow, mlflow emphasises portability by adding minimal standardization to project structure and capturing projects as docker-based or conda-based environments.

The below picture of the ML lifecycle (from [a talk by Jules Damji, Databricks¹⁵⁹](#)) helps to show how mlflow tackles key problems:

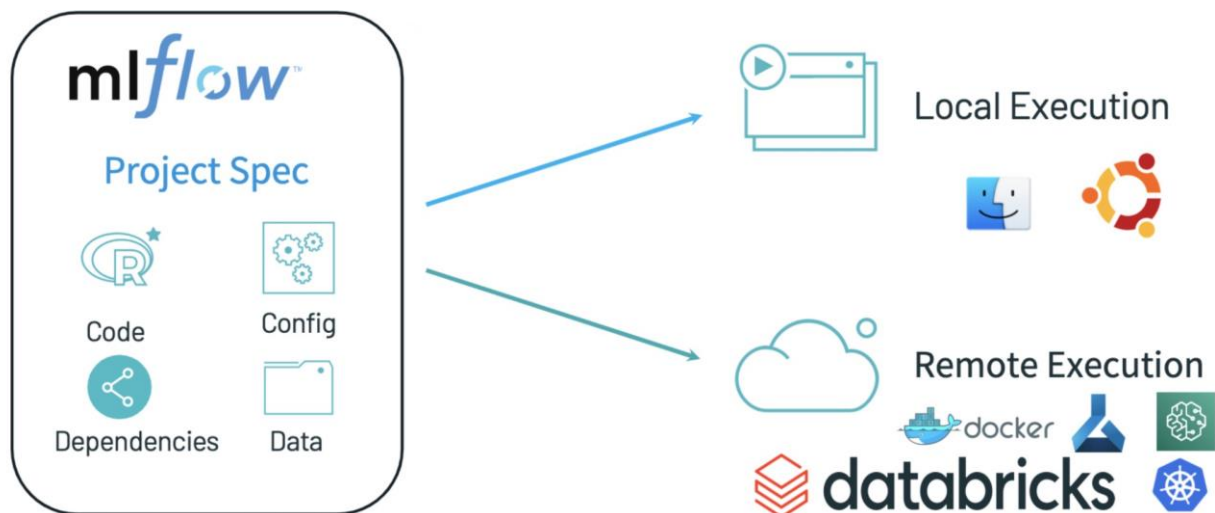
¹⁵⁹ <https://databricks.com/discover/managing-machine-learning-lifecycle/mlflow-projects-and-models>



At each stage of the lifecycle we see that users want to work with a range of very different tools. Users also want to work locally and then employ different tools in order to scale their work (e.g. training with a small slice of data locally and then a larger slice on a hosted environment to run a longer job). MLflow therefore makes life cycle transitions and local-to-hosted transitions smoother by providing a standard framework of interaction and an exchange format so that artifacts are portable.

We can get a good feel of mlflow by walking through its key concepts ([diagram by databricks¹⁶⁰](https://databricks.com/blog/2021/02/03/ray-mlflow-taking-distributed-machine-learning-applications-to-production.html)):

¹⁶⁰ <https://databricks.com/blog/2021/02/03/ray-mlflow-taking-distributed-machine-learning-applications-to-production.html>



4.10.1 Tracking

The tracking features of mlflow are for recording what experiments have taken place. It enables experiment runs to be stored along with all the parameters for each experiment and which models were produced. This can then be searched and referred back to, either by the original author or by others in the team. To use this feature some tracking code is added to the model training code:

```
with mlflow.start_run():
    lr = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, random_state=42)
    lr.fit(train_x, train_y)

    predicted_qualities = lr.predict(test_x)

    (rmse, mae, r2) = eval_metrics(test_y, predicted_qualities)

    mlflow.log_param("alpha", alpha)
    mlflow.log_param("l1_ratio", l1_ratio)
    mlflow.log_metric("rmse", rmse)
    mlflow.log_metric("r2", r2)
    mlflow.log_metric("mae", mae)
```

The above is using the python API (there's also R, Java and REST APIs) and comes from an [official mlflow example](https://www.mlflow.org/docs/latest/tutorials-and-examples/tutorial.html)¹⁶¹. By default the run parameters and metrics will be logged to a [local directory](https://www.mlflow.org/docs/latest/tracking.html#where-runs-are-recorded)¹⁶². When we then run the command `mlflow ui` using the CLI then we can access the [history of this experiment](https://www.mlflow.org/docs/latest/tutorials-and-examples/tutorial.html)¹⁶³ (among other things):

¹⁶¹ <https://www.mlflow.org/docs/latest/tutorials-and-examples/tutorial.html>

¹⁶² <https://www.mlflow.org/docs/latest/tracking.html#where-runs-are-recorded>

¹⁶³ <https://www.mlflow.org/docs/latest/tutorials-and-examples/tutorial.html>

	Date	User	Source	Version	Parameters		Metrics		
					alpha	l1_ratio	mae	r2	rmse
<input type="checkbox"/>	2018-06-04 23:00:10	mlflow	train.py	05e956	1	1	0.649	0.04	0.862
<input type="checkbox"/>	2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.5	0.648	0.046	0.859
<input type="checkbox"/>	2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.2	0.628	0.125	0.823
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	1	0	0.619	0.176	0.799
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	1	0.648	0.046	0.859
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.5	0.628	0.127	0.822
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.2	0.621	0.171	0.801
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0	0.615	0.199	0.787
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0	1	0.578	0.288	0.742
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.5	0.578	0.288	0.742
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.2	0.578	0.288	0.742
<input type="checkbox"/>	2018-06-04 23:00:08	mlflow	train.py	05e956	0	0	0.578	0.288	0.742

[Artifacts](#)¹⁶⁴ can be recorded as well as parameters and artifacts can then be downloaded from the UI. The storage of run data can be [remote instead of local](#)¹⁶⁵, to support sharing run info between data scientists. There's also features to [visualize/graph metrics](#)¹⁶⁶ and a [new feature](#)¹⁶⁷ to replace all the logging/tracking code with just `mlflow.autolog()` for specific machine learning frameworks.

4.10.2 Projects

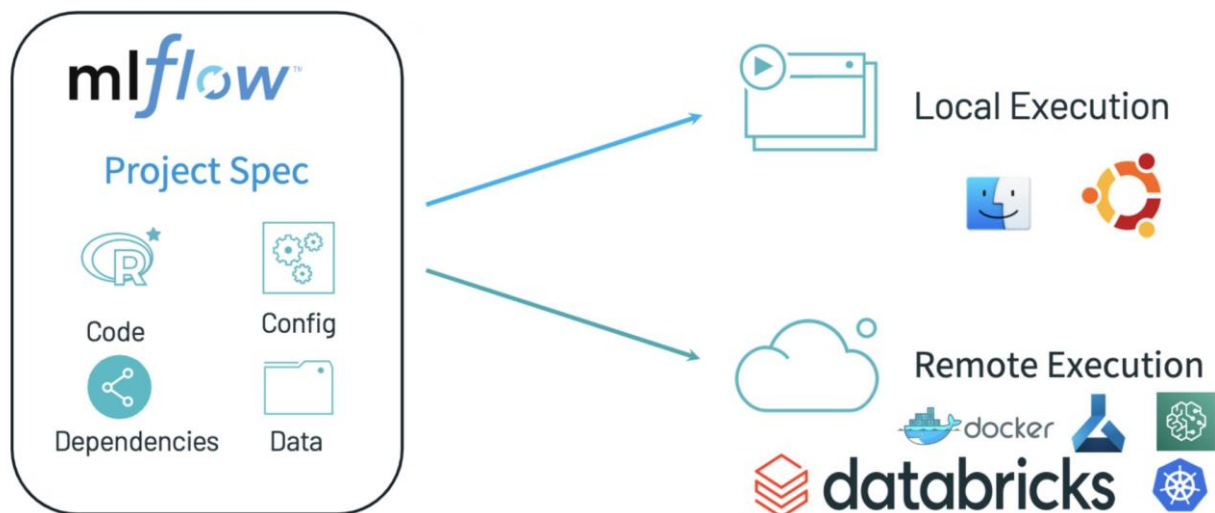
Projects are a convention for packaging ML code so that it can run in different environments without modification. Projects also facilitate reproducibility since they capture the versions, configuration and environment that is used for a training run or serving deployment:

¹⁶⁴ <https://www.mlflow.org/docs/latest/tracking.html#id23>

¹⁶⁵ <https://www.mlflow.org/docs/latest/tracking.html#scenario-4-mlflow-with-remote-tracking-server-backend-and-artifact-stores>

¹⁶⁶ <https://www.mlflow.org/docs/latest/tracking.html#id57>

¹⁶⁷ <https://www.mlflow.org/docs/latest/tracking.html#automatic-logging>



Essentially a project is a directory with an [MLProject file](#)¹⁶⁸ at the top with the MLProject file specifying the environment (e.g. docker or conda, with relevant env files also in the directory). Projects also have [entry points](#)¹⁶⁹, which are commands that are meant to run for the project e.g. to train the model (entry points in an MLProject are similar to targets in a Makefile).

The mlflow CLI can then read a project and know how to recreate its environment for running. we can call `mlflow run` on a local or git directory and specify an entry point (or leave this for the default one) and mlflow will then run that entry point e.g. a training job. Entrypoints can also be chained together to make [multi-step workflows](#)¹⁷⁰.

Projects can also have multiple backends, which are configurations for running the project in different types of environments. For example, a [kubernetes backend](#)¹⁷¹ can be added for running in kubernetes.

4.10.3 Models

There are many ways of packaging machine learning models ([tensorflow format](#)¹⁷², pickling, onnx, pmml, docker) and many tools that support specific methods of packaging. Rather than pick certain methods of packing to support, mlflow defines a way of specifying how a model can be packaged. The idea is to provide a way to package machine learning models so that they can be run in a range of deployment tools for real-time serving or batch scoring. The variation is handled with a concept of "[flavors](#)¹⁷³". This is best understood by [example](#)¹⁷⁴.

Here is what is written automatically by the `mlflow.sklearn.save_model` function on a project with an sklearn model:

```
# Directory written by mlflow.sklearn.save_model(model, 'my_model')
my_model/
├── MLmodel
```

¹⁶⁸ <https://www.mlflow.org/docs/latest/projects.html#mlproject-file>

¹⁶⁹ <https://www.mlflow.org/docs/latest/projects.html>

¹⁷⁰ https://github.com/mlflow/mlflow/tree/master/examples/multistep_workflow

¹⁷¹ <https://mlflow.org/docs/latest/projects.html#execution-guide>

¹⁷² https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/saved_model/README.md

¹⁷³ <https://www.mlflow.org/docs/latest/models.html#built-in-model-flavors>

¹⁷⁴ <https://www.mlflow.org/docs/latest/models.html#storage-format>

```
|— model.pkl
|— conda.yaml
|— requirements.txt
```

So here the model is serialized as a pickle file. To deserialize a python pickle, the target environment needs to have dependencies that satisfy those of the serialized code. Hence here the conda.yaml and requirements.txt files capture the dependencies. The MLmodel file contains:

```
time_created: 2018-05-25T17:28:53.35
```

```
flavors:
  sklearn:
    sklearn_version: 0.19.1
    pickled_model: model.pkl
  python_function:
    loader_module: mlflow.sklearn
```

Here sklearn and python_function are flavors. Both are entered for this model because the pickle mechanism works for either. It is a general-purpose way of packaging python code (the python_function flavor is for any python code, including custom functions) and is also the favored way of serializing sklearn models. There are a range of other [flavors in mlflow](#)¹⁷⁵ including R and spark.

Any model that supports python_function or R (crate) flavors can be served locally with `mlflow models serve -m my_model`. Certain flavors can also be deployed to specific serving tools. For example, models that support python_function can be deployed to sagemaker with `mlflow sagemaker deploy -m my_model`. There's also built-in support for [azureml and spark](#)¹⁷⁶, [plug-in support](#)¹⁷⁷ for others and other tools that support the format but which are not listed in the mlflow documentation.

4.10.4 Model Registry

When lots of models are being produced and many versions of a single model, then there are challenges with managing these models. We want to be able to find which version is the latest, which version is running in production and how a model version was trained. The mlflow model registry makes these details [searchable and easy to access](#)¹⁷⁸:

¹⁷⁵ <https://www.mlflow.org/docs/latest/models.html#built-in-model-flavors>

¹⁷⁶ <https://mlflow.org/docs/latest/models.html#built-in-deployment-tools>

¹⁷⁷ <https://www.mlflow.org/docs/latest/plugins.html#deployment-plugins>

¹⁷⁸ <https://databricks.com/blog/2019/10/17/introducing-the-mlflow-model-registry.html>

Name	Latest Version	Staging	Production	Last Modified
Item_Recommender	Version 5	Version 5	Version 4	2019-10-11 15:30:02
Airline_Delay_Scikit	Version 3	—	Version 1	2019-10-11 12:41:43
Airline_Delay_SparkML	Version 5	Version 5	Version 3	2019-10-11 12:45:15
Transaction_Fraud_Classifier	Version 1	—	—	2019-10-11 15:18:05
Icon_GAN	Version 1	—	—	2019-10-12 08:20:12
Power_Forecasting_Model	Version 1	—	Version 1	2019-10-07 15:38:27
Product_Image_Classifier	Version 6	—	Version 5	2019-10-12 00:38:56
Comment_Summarizer	Version 3	Version 2	Version 3	2019-10-12 00:39:40
Movie_Recommender	Version 5	Version 5	Version 3	2019-10-10 14:07:07
Translation_Alpha	—	—	—	2019-10-11 16:45:01

Recording models to the registry is integrated with the [tracking API](#)¹⁷⁹. Calling `mlflow.<model_flavor>.log_model()` records that version of the model in the registry. Alternatively, models can be registered by selecting their artifacts [in the tracking UI](#)¹⁸⁰.

The stages of models can also be transitioned (e.g. record as promoted to production) [via the API](#)¹⁸¹ (e.g. from a CI system) or [from the UI](#)¹⁸².

¹⁷⁹ <https://mlflow.org/docs/latest/model-registry.html#id7>

¹⁸⁰ <https://mlflow.org/docs/latest/model-registry.html#id4>

¹⁸¹ <https://mlflow.org/docs/latest/model-registry.html#transitioning-an-mlflow-models-stage>

¹⁸² <https://mlflow.org/docs/latest/model-registry.html#ui-workflow>